64
R1

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES
CUAUTITLAN

" DISEÑO DE UN SISTEMA MINIMO BASADO EN EL MICROPROCESADOR 8088 "

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERA MECANICA ELECTRICISTA

P R E S E N T A N

MA. DE LA CRUZ GARCIA CONTRERAS
LUCIA GARCIA SOTO

ASESOR: ING. JORGE BUENDIA GOMEZ

CUAUTITLAN IZCALLI, EDO. DE MEX. [redacted]

1997

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN
UNIDAD DE LA ADMINISTRACION ESCOLAR
DEPARTAMENTO DE EXAMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN

DEPARTAMENTO DE EXAMENES PROFESIONALES

ASUNTO: VOTOS APROBATORIOS

DR. JAIME KELLER TORRES
DIRECTOR DE LA FES-CUAUTITLAN
P R E S E N T E .

AT'N: Ing. Rafael Rodriguez Ceballos
Jefe del Departamento de Examenes
Profesionales de la F.E.S. - C.

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS TITULADA:
"Diseño de un Sistema Mínimo basado en el Microprocesador 8098"

que presenta la pasante: María de la Cruz García Contreras
con número de cuenta: 8324964-0 para obtener el TITULO de:
Ingeniera Mecánica Electricista .

Considerando que dicha tesis reúne los requisitos necesarios para ser discutida en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E .
"POR MI RAZA HABLARA EL ESPIRITU"
Cuautitlán Izcalli, Edo. de Méx., a 28 de Agosto de 199 6

PRESIDENTE	Ing. Antonio Herrera Mejía
VOCAL	Ing. José Luis Rivera López
SECRETARIO	Ing. Jorge Buendía Gómez
PRIMER SUPLENTE	Ing. José Ubaldo Ramírez Urizar
SEGUNDO SUPLENTE	Ing. Blanca Gisel de la Peña Valencia

UAE/DEP/VAP/02

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN
UNIDAD DE LA ADMINISTRACION ESCOLAR
DEPARTAMENTO DE EXAMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS SUPERIORES-CUAUTITLAN

DEPARTAMENTO DE EXAMENES PROFESIONALES

ASUNTO: VOTOS APROBATORIOS

DR. JAIME KELLER TORRES
DIRECTOR DE LA FES-CUAUTITLAN
P R E S E N T E .

AT'N: Ing. Rafael Rodriguez Ceballos
Jefe del Departamento de Exámenes
Profesionales de la F.E.S. - C.

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS TITULADA:
"Diseño de un Sistema Mínimo basado en el Microprocesador 8088"

que presenta la pasante: Lucía García Soto
con número de cuenta: 8225914-5 para obtener el TITULO de:
Ingeniera Mecánica Electricista .

Considerando que dicha tesis reúne los requisitos necesarios para ser discutida en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E .
"POR MI RAZA HABLARA EL ESPIRITU"
Cuautitlan Izcalli, Edo. de Méx., a 28 de Agosto de 1996 .

PRESIDENTE	Ing. Antonio Herrera Mejía	
VOCAL	Ing. José Luis Rivera López	
SECRETARIO	Ing. Jorge Buendía Gómez	
PRIMER SUPLENTE	Ing. José Ubaldo Ramírez Urizar	
SEGUNDO SUPLENTE	Ing. Blanca Gisel de la Peña Valencia	

UAE/DEP/VAP/02

A Dios:

Gracias por haberme permitido realizar uno de mis más grandes anhelos.

Gracias porque nunca te has alejado de mí.

A mis padres con mucho cariño y respeto:

Cruz y Miguel

Gracias por todo el cariño que me han brindado siempre.
Gracias por todos sus sacrificios.
Gracias por todo el apoyo que han ofrecido.
Gracias por darme la vida.

A mis hermanos:

Ricardo, Araceli y Eduardo

Gracias por todo su cariño y apoyo que siempre me han brindado.

Los quiero.

MARICRUZ

A mi esposo:

Benjamín

Gracias por todo tu amor, ternura y paciencia.

Te amo.

MARICRUZ

AL ING. UBALDO RAMIREZ URIZAR:

Gracias por su apoyo, su confianza y su amistad.

MARICRUZ

DEDICO ESTE TRABAJO CON CARIÑO A MI MADRE PORQUE GRACIAS A SUS SACRIFICIOS HE CAMINADO FELIZ Y A SU EJEMPLO QUE ME HA SERVIDO DE INSPIRACIÓN.

AGRADEZCO A MIS HERMANOS RUBÉN Y GABRIELA, POR EL APOYO QUE ME HAN BRINDADO.

A MI FAMILIA POR SU AYUDA Y CONFIANZA.

A AGUSTÍN POR SU AMOR Y PACIENCIA.

LUCÍA GARCÍA SOTO

Gracias a todos nuestros amigos y compañeros del Grupo de Desarrollo por su interés y por toda la ayuda brindada para la realización de este proyecto.

Encomienda tus obras a Dios y tus proyectos se llevarán a cabo.

Proverbios 16 : 3

INDICE

CAPITULO 1
INTRODUCCION

1.1 INTRODUCCION

Las computadoras se han venido utilizando de forma general desde los años 50's. En un principio las computadoras digitales eran máquinas grandes y caras. El tamaño y forma de la computadora digital ha cambiado gracias a un nuevo dispositivo denominado microprocesador. El microprocesador es un Circuito Integrado que tiene la mayoría de las capacidades de procesamiento de las grandes computadoras. El microprocesador es un pequeño pero extremadamente complejo, dispositivo LSI programable.

Al adoptar IBM el Intel 8088 para entrar en el mercado de las microcomputadoras se produjo una gran aceptación para un CPU. Debido a su bus externo de ocho bits el 8088 puede utilizar, como soporte, chips de 8 bits. La característica del chip como un 8/16 se desprende de su bus de datos de ocho bits. Este chip tiene un bus de direcciones de 20 bits que permite un espacio de direccionamiento de 1 Megabyte.

Evolución de los microprocesadores de Intel

Intel lanzó en 1971 su primer microprocesador: el 4004, de 4 bits que contenía un conjunto de instrucciones que ofrecían sólo 45 instrucciones y el 8008 de 8 bits. Más tarde introdujo el microprocesador 8080 que maneja

palabras de datos de 8 bits y tiene 16 líneas de dirección y un puntero de pila de 16 bits. El 8085 es una versión mejorada del 8080, integra el reloj, control del sistema y prioridad de las interrupciones en el microprocesador, reduciendo así el número de circuitos integrados utilizados en la mayoría de los sistemas.

Una tendencia en la evolución de los microprocesadores ha sido integrar más funciones en menos Circuitos Integrados.

El 8086 fue el microprocesador de 16 bits. Se diseñó para que fuese compatible con los microprocesadores 8080/8085 de 8 bits. La compatibilidad permite que los programas del 8080/8085 sean convertidos fácilmente para que puedan correr en el 8086. El 8088 se introdujo un tiempo después que el 8086 y, funcionalmente, es el mismo, pero caracteriza un bus de datos de 8 bits en lugar del bus de datos de 16 bits que se encuentra en el 8086.

El 8088/8086 y sus parientes de 16 bits (80186, 80286) y 32 bits (80386, 80486) se han convertido en algunos de los microprocesadores de propósito general más ampliamente utilizados en el mundo.

1.2 PLANTEAMIENTO DEL PROBLEMA

El presente trabajo surgió de la opción de contar en el laboratorio de la escuela con una microcomputadora basada en el microprocesador 8088.

1.3 ALCANCES, LIMITACIONES Y OBJETIVOS

Dentro de los objetivos principales de este trabajo es presentar los aspectos esenciales del microprocesador, sus características y su funcionamiento dentro de una microcomputadora.

En la presente tesis se desarrolla el diseño y construcción del prototipo de un sistema mínimo de propósito general.

Este sistema mínimo permite la entrada de información desde un teclado, muestra la información en un display de 7 segmentos. El usuario tiene acceso a los programas grabados en una memoria EPROM. El programa principal grabado en la EPROM del sistema es el programa monitor, que contiene las funciones básicas del sistema.

1.4 ORGANIZACION DEL TRABAJO

El diseño del sistema mínimo se dividió en dos partes principales: diseño del hardware y elaboración del software. El diseño del hardware consiste en la conexión de los componentes físicos para obtener el sistema mínimo deseado. El diseño de la programación trata del desarrollo de los programas para una aplicación particular. En la elaboración de programas para la operación del microprocesador se debe estar familiarizado con la configuración de los componentes que conforman el sistema mínimo, además de tener en cuenta los problemas asociados con la aplicación particular.

Para armar el prototipo se optó por utilizar la técnica conocida como WIRE WRAP. Esta es una técnica muy versátil y provee de una buena firmeza en las conexiones. Tal técnica requiere de una tarjeta con una matriz de orificios tipo estándar, un alambrador, cable y bases para circuitos integrados, adecuados para trabajar con dicha técnica.

El software desarrollado permite la interacción del sistema con el usuario, ofreciendo las siguientes opciones:

- Escritura
- Ejecución
- Desplegado

La presente tesis está dividida en tres partes:

En la parte 1 se presenta el diseño del hardware, el cual se divide en etapas para describir el diseño del sistema mínimo, así como los componentes, su función y operación dentro del sistema.

En la parte 2 se desarrolla el diseño del software. Se mencionan las diferentes formas de programar el 8088, así como el procedimiento utilizado para este trabajo.

En la parte 3 se verifica el funcionamiento del sistema observando las señales de operación y los programas de prueba, así como se integra la guía del usuario en la cual se dan los pasos para operar adecuadamente el sistema.

CAPITULO 2
CARACTERISTICAS DEL 8088

2.1 INTRODUCCION

En esta sección se describe la función de las terminales del microprocesador, la arquitectura interna y su modelo de programación.

2.2 PRINCIPALES CARACTERISTICAS DEL 8088

EL microprocesador 8088 es un procesador de alta ejecución, maneja 8 - 16 bits por lo que es compatible con microprocesadores que manejan 8 y 16 bits.

Características principales:

- Interface del bus de datos de 8 bits.
- Arquitectura interna de 16 bits.
- Acceso directo a memoria para manejar 1 Mbyte.
- Modos de operación de direccionamiento.
- Operaciones de byte, palabra y bloques.
- 8 y 16 bits para operaciones aritméticas con signo y sin signo en binario o decimal incluyendo multiplicación y división.

El 8088 requiere de una alimentación de +5 V cd con una tolerancia de +/- 10%., tiene un consumo máximo de corriente de 340 mA.

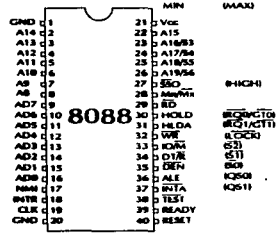


FIGURA 2-1 Terminales del 8088.

2.2.1 TERMINALES DEL 8088

Existen varios grupos de señales de control y señales de estado. Que se mencionan con más detalle a continuación :

AD7 - AD0 (Datos y direcciones). Estas líneas constituyen las direcciones de memoria de E/S y datos multiplexados en tiempo.

Estas líneas se activan en alto y flotan al 3er. estado durante un reconocimiento de interrupción y a una petición local de bus.

A15 - A8 (líneas de direcciones). Estas líneas proveen los bits de direcciones del 8 al 15.

A19/S6, A18/S5, A17/S4, Y A16/S3. (Direcciones de Estado). Terminales de direcciones/estado multiplexadas que contiene los bits del bus de direcciones A16-A19 y también contiene los bits de status S6-S3.

El bit de estado S6 siempre permanece en 0 lógico, el bit S5 indica que la condición del bit de bandera de interrupción, y los bits S4 y S3 indican que segmento es accesado durante el ciclo del bus. En la tabla 2-1 muestra los códigos de S4 y S3.

S4	S3	FUNCION
0	0	Segmento Extra
0	1	Segmento Stack
1	0	Segmento código o no segmento
1	1	Segmento de datos

TABLA 2-1

RD (Lectura). Esta señal indica que el procesador está ejecutando un ciclo de lectura con la memoria o entrada/salida, dependiendo del estado de la

terminal IO/M. Esta señal se usa para leer los dispositivos que reciben o están conectados en el bus local del 8088.

READY (Listo). Esta terminal de entrada que cuando se controla se puede emplear para introducir estados de espera en la temporización del microprocesador. Si esta terminal tiene un nivel alto (1), no tiene efecto en el funcionamiento del microprocesador. Si lleva un 0 lógico, el microprocesador introduce estados de espera y permanece en el mismo ciclo del bus.

INTR (Petición de interrupción). Se emplea para solicitar una interrupción por hardware.

TEST (Prueba). Esta entrada es examinada por la instrucción WAIT. Si la entrada TEST es baja la ejecución continúa, de otra manera el procesador espera un estado inactivo hasta que TEST tenga un estado bajo. Esta señal se sincroniza internamente durante cada ciclo de reloj en el flanco de subida .

NMI (Interrupción no mascarable). Es una señal disparada por nivel, la cual provoca una interrupción.

RESET (Reinicialización). Señal de entrada provoca que el procesador inmediatamente termine su actividad para realizarla nuevamente. La señal debe estar en alto por lo menos 4 ciclos de reloj, y se reinicia la ejecución cuando el reset vuelve a nivel bajo y se va a la dirección FFF0H y esta terminal se sincroniza internamente.

CLK (Reloj). Una entrada que da la temporización básica para el microprocesador y controlador de bus, ésta es asimétrica con un 33% para que sea una temporización interna óptima.

IO/M (Memoria o entrada/salida). Esta terminal indica cuando el bus de direcciones contiene una dirección de memoria o una dirección de entrada o salida. Esta salida flota en alta impedancia durante un reconocimiento de H&A.

WR (Escritura). Señal de salida que indica cuando el bus de datos contiene datos válidos para ser almacenados en memoria o en E/S.

INTA (Reconocimiento de interrupción). Es el reconocimiento de interrupción, es decir, la respuesta durante una petición de interrupción.

ALE (Habilitación de la dirección). Esta terminal es usada para indicar que el bus de datos y direcciones contienen una dirección de memoria o una dirección de E/S.

DT/R (Transmitir y recibir datos). Esta terminal de entrada es usada para controlar la dirección de datos que fluye a través de los buffers del bus de datos conectado externamente.

DEN (Habilitador del bus de datos). Esta señal de salida indica que el bus de datos y direcciones contiene un dato válido.

HOLD (Retener). Es una entrada usada para solicitar un acceso directo a memoria (DMA). Cuando HOLD está activado el 8088 flotará sus líneas de direcciones, datos y control.

HLDA (Reconocimiento). Es una señal de salida que es usada para indicar que el bus está libre.

SSO (Líneas de estado). Esta línea determina, en combinación IO/M y el DT/R, el estado en que se encuentra el microprocesador (en obtención de código, escritura en memoria, lectura de memoria, escritura en puertos E/S, lectura de un puerto de E/S).

A continuación se muestra la tabla 2-2 de estados del microprocesador:

IO/M	DT/R	SSO	FUNCION
0	0	0	Indica un reconocimiento de interrupción.
0	0	1	Indica lectura de memoria.
0	1	0	Indica escritura de memoria.
0	1	1	Indica un alto.
1	0	0	Indica acceso a código.
1	0	1	Indica lecturas a puertos E/S.
1	1	0	Indica lectura a puertos E/S.
1	1	1	Permanece pasivo.

Tabla 2-2

2.3 ARQUITECTURA BASICA

A continuación se menciona una breve descripción de la arquitectura básica del 8088.

2.31 ARQUITECTURA BASICA INTERNA

El 8088 busca sus instrucciones en la memoria. La figura 2.2 ilustra la secuencia de eventos para el 8088. Se observa que el bus está casi siempre ocupado (algunas veces el bus está libre, pero no frecuentemente). Este microprocesador contiene dos unidades internas EU (Unidad de Ejecución) y BIU (Unidad de Interface del bus). La BIU es responsable de traer una instrucción o el operando de una instrucción o datos desde la memoria, y la EU es responsable de ejecutar las instrucciones.

En la figura 2.2a muestra La operación normal de un 8085 el cual es típico del microprocesador de 8 bits. Las instrucciones se recuperan de la memoria por una operación de lectura. Mientras el 8085 ejecuta la instrucción el sistema de memoria está ocioso. El 8088, figura 2.2b, hace uso de este tiempo ocioso de memoria para realizar la "pre-búsqueda" de la siguiente instrucción mientras se ejecuta la actual.

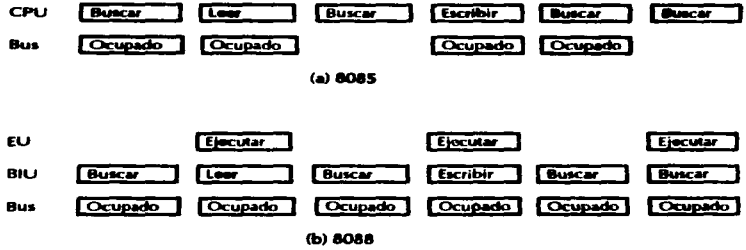


FIGURA 2-2. Actividad del bus.

La figura 2-3 muestra el diagrama a bloques de la estructura interna del 8088 y la comunicación entre ellos.

El 8088 está habilitado para utilizar el bus con la máxima eficiencia porque contiene la memoria interna en forma de una cola o "FIFO" (Primera entrada - Primera salida) de memoria.

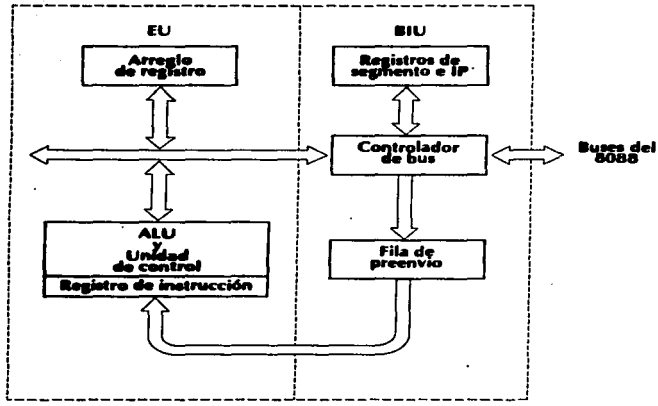


FIGURA 2-3 Estructura interna del 8088 EU y BIU.

La figura 2-4 ilustra la EU y la BIU en la cual la cola es localizada. El microprocesador tiene una cola de 1 byte de ancho que es de 4 bytes. Esta cola lleva las instrucciones mientras que la EU está ocupada ejecutando otras. Esto lleva al microprocesador a tener un sistema de memoria más eficiente que sus antecesores.

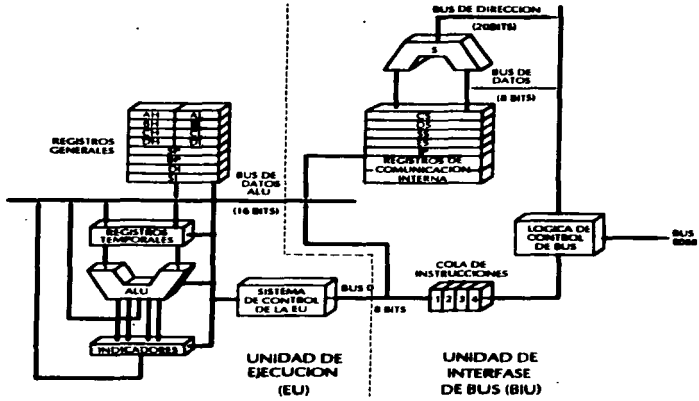


FIGURA 2-4

La BIU contiene una cola llamada "prefetch", un controlador de bus, registros de segmento y el apuntador de instrucciones (IP). El propósito principal de la BIU para guardar la cola "prefetch" llena con instrucciones, es para generar y aceptar las señales del sistema de control, para proveer el sistema con un direccionamiento de memoria o número de puerto E/S, y para actuar como una ventana entre la EU y la memoria para datos.

La BIU asegura que la cola está llena con instrucciones llevando el siguiente simple byte de una instrucción si la cola tiene espacio. El prefetch o "prellevado" permite a la EU obtener la siguiente instrucción directamente desde la BIU en vez de la memoria. Como los viejos microprocesador de 8 bits. El microprocesador ejecuta el software mucho más rápido que si cada instrucción fuera llevada directamente desde la memoria.

El propósito de la EU es de llevar fuera instrucciones que son traídas desde la cola "prefetch". La unidad de ejecución contiene una ALU, un registro de instrucción y un arreglo de registros. La ALU ejecuta operaciones aritméticas y lógicas en memoria o datos de registros. El registro de instrucción recibe las instrucciones desde la cola prefetch, a cualquier tiempo son decodificados directamente a la operación de la EU. El arreglo de registro, una memoria portapapeles, retiene información temporalmente. También contiene apuntadores o registros de índice utilizadas para direccionar la operación de datos localizada en la memoria. Los operandos de datos son también direccionados y transferidos a través de la BIU.

2.3.2 ARQUITECTURA DEL SISTEMA

La figura 2-4 ilustra la arquitectura del sistema. La comunicación del sistema la realiza a través de tres buses: datos, direcciones y control. El bus de direcciones provee una dirección de memoria para el sistema de memoria y también direcciones de E/S para los dispositivos del sistema de E/S. El bus de datos transfiere datos entre el microprocesador y la memoria e E/S definidos

para el sistema. El bus de control provee señales de control que causa la memoria o E/S para ejecutar una operación de RD o WR.

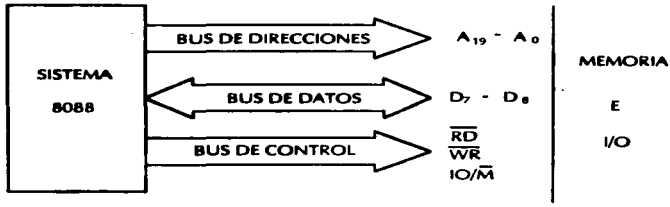


FIGURA 2-5

2.3.3 MEMORIA

El 8088 direcciona 1 MByte (1,048,576 bits) de memoria.

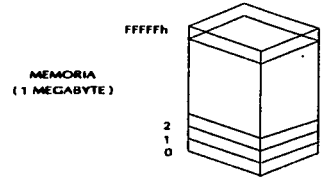


FIGURA 2-6

Existen dos tipos de memoria referida a la memoria del microprocesador: La memoria física y memoria lógica.

La memoria lógica es aquella que los programadores definen a través del software y la física en el sistema la cual está definida por la organización de la memoria que el diseñador establece.

Para el microprocesador 8088 la memoria física y la memoria lógica es la misma, inicia en la localidad 00000H y termina hasta la FFFFFH. La memoria es de 8 bits y el rango de direccionamiento especificado de 1 MByte de memoria disponible en el sistema. Una palabra de direccionamiento y se extiende para 2 bytes consecutivos. Por ejemplo: la palabra se localiza 00122H es almacenada en los bytes 00122H y 00123H, con el byte menos significativo almacenado en la localidad 00122H.

2.4 REGISTROS

En esta sección se describe la estructura de los registros internos del 8088 y se explica como la memoria es direccionada por los registros de segmento.

La figura 2-7 muestra el orden de los registros internos del 8088. Este arreglo consiste de tres grupos de registros: los de propósito general, los de puntero e índice y los registros de segmento. En adición a estos grupos existe un registro de bandera, que indica las condiciones de operación de la ALU.

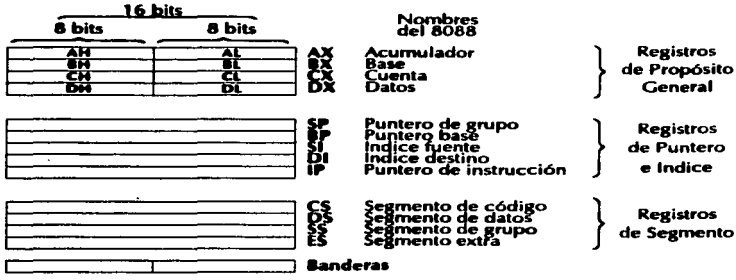


FIGURA 2-7

A) Registros de Propósito General

Estos se utilizan de acuerdo a lo que se desee programar. Cada registro está direccionado a un registro de 16 bits (AX, BX, CX y DX) a dos registros de 8 bits (AL AH, BH, BL, CL, CH, DH, DL). Algunas de las instrucciones realizan tareas específicas que se describen en el capítulo 3. Estos son: el Acumulador, Base, Contador y Datos. En el lenguaje ensamblador el registro de propósito general está siempre definido a una combinación de letras. Las funciones principales de los registros de propósito general incluyen:

AX (Acumulador). Se utiliza para retener temporalmente el resultado después de una operación lógica o aritmética.

BX (Base). Se utiliza para retener la dirección en base a un dato localizado en la memoria.

CX (Contador). Contiene el conteo de ciertas instrucciones.

DX (Dato). Un registro de propósito general que además retiene la parte más significativa del producto de una multiplicación de 16 bits y la parte más significativa del dividendo *antes* de una división y el número de puerto E/S para un instrucción E/S variable.

B) Registros Puntero e Indice

Aunque los registros de puntero e índice son además de propósito general en naturaleza, son más frecuentemente utilizados para indexar y apuntar a la localidad de memoria que retiene el dato del operando para muchas instrucciones.

Estos registros incluyen:

SP (Puntero de Almacenamiento). Utilizado para direccionar datos en una memoria de acumulador tipo LIFO (última entrada - primera salida). Esto ocurre frecuentemente cuando las instrucciones PUSH y POP son ejecutadas o cuando una subrutina es llamada (CALL) o regresada (RET) del programa.

BP (Puntéro de Base). Un puntero de propósito general usado para direccionar un arreglo de datos en una memoria de almacenamiento.

SI (Indice Fuente). Se usa para direccionar una fuente de datos indirectamente para utilizarla con la cadena de instrucciones.

DI (Indice Destino). Utilizado para indirectamente destinar la dirección de datos y utilizarla con cadenas de instrucciones.

IP (Puntero de Instrucción). Utilizado para direccionar la siguiente instrucción a ejecutarse por el 8088, la localización real de la siguiente instrucción está formada al sumarle el contenido del IP a CS X 10H como se describe en el capítulo 3.

Los datos son direccionados indirectamente a través de cuatro de los cinco registros de 16 bits, pero nunca por un puntero de instrucción.

C) **Registros de Segmento**

Los registros de segmento son diseñados para poder direccionar los 20 bits de dirección, esto es, ya que los registros de índice y puntero son de 16 bits de ancho, y la memoria en el 8088 es de 1 MByte, el cual requiere direccionar 20 bits.

Un segmento de memoria es un bloque de 64 Kbytes de memoria (ver figura 2-8), direccionado por un registro especial llamado *registro de segmento*. Cuatro segmentos están simultáneamente en el espacio de

memoria: El segmento de código, segmento de datos, segmento de almacenamiento y el segmento extra.

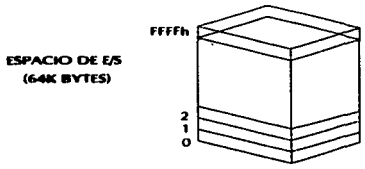


Figura 2-8

Los datos son indexados o punteados a un segmento por el registro índice, puntero base o de instrucción.

Cada registro de segmento retiene una porción de 16 bits de los 20 bits de la dirección de inicio de un segmento de 64 Kbyte de memoria. La dirección de 20 bits se forma al anexar los registros de segmento con un 0000H colocados en el número menos significativo en el registro de segmento. La tabla 2.3 muestra ejemplos de los contenidos de registros de segmentos y las direcciones indirectamente direccionadas por ellos.

REGISTROS DE SEGMENTO	RANGO DE DIRECCIONAMIENTO DE MEMORIA
0100H	01000H - 10FFFH
0101H	01010H - 1100FH
1234H	12340 - 2233FH
2000H	20000H - 2FFFFH

TABLA 2-3

D) Registros de Bandera

Son llamados también registros de estado o palabras de estado o estado de programa que es un registro de 16 bits. Los 8 bits más altos contienen las banderas C, P, A, Z y S y los 8 bits más bajos contienen las banderas T, I, D y O.

La figura 2-9 ilustra las posiciones de bit relativo de cada bit de bandera y la designación utilizada para identificar cada bit de bandera.

Los bits de bandera son:

C (Acarreo). Indica llevando fuera o tomando dentro de la posición de un bit menos significativo siguiendo una operación lógica o aritmética.

P (Paridad). Se refiere a la paridad del resultado de una operación lógica o aritmética. Si el resultado contiene un número de 1s. el bit de paridad está

activado para indicar paridad, es decir; es limpiado para indicar una paridad impar.

A (Acarreo Auxiliar). Representa llevando o tomando prestado entre byte de una operación lógica o aritmética de 8 bits utilizando el registro AL.

Z (Cero). Indica que el resultado de una operación lógica o aritmética es cero. Si Z01 el resultado es cero.

S (Signo). Indica el signo del resultado de una operación. Si un 1 lógico aparece, el resultado de la operación es negativo.

T (Trampa). Provoca que el 8088 entre dentro de un modo de paso simple de operación.

I (Habilitador de interrupciones). Habilita o deshabilita la terminal INTR (Interrup Request). Si I=1 entonces INTR es habilitado.

D (Dirección). Selecciona el modo de autoincremento o decremento de operación para el índice destino (DI) y el índice fuente (SI) en las operaciones de cadena.

O (Sobreflujo). Indica un sobreflujo aritmético después de realizar una operación.

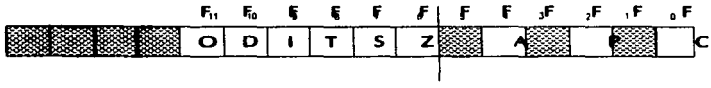


FIGURA 2-9 Banderas de estado

2.5 ESTRUCTURA DE MEMORIA DE SEGMENTACION

Los registros de segmento del 8088 fueron diseñados para resolver un problema peculiar de este microprocesador. Este problema está dado porque los registros de índice y apuntadores tienen un ancho de 16 bits, y la memoria en el microprocesador es de 1 Mbyte, el cual requiere direcciones de veinte bits, los registros índices y apuntadores no son lo suficientemente anchos para poder direccionar directamente cualquier localidad de memoria de 20 bits.

Un segmento de memoria es un bloque de 64 Kbytes de memoria direccionada por un registro especial llamado registro de segmento. Cuatro registros de segmento independientes pueden existir simultáneamente en el espacio de memoria: segmento de código, segmento de datos, segmento de pila y segmento extra. Los datos son apuntados o indexados en un segmento por los registros índices, registros apuntadores, registro base o apuntador de instrucciones.

Cada registro de segmento almacena 16 bits de los 20 bits de la dirección de inicio de 64 Kbytes de memoria. Los 20 bits de direcciones se forman

anexando al registro de segmento 0000_2 (0h) colocados en la posición menos significativa del registro del segmento.

Cada registro de segmento tiene una función especial y es asociada con uno o más registros índices o apuntadores. Para generar una localidad de memoria, el contenido de un registro de segmento, que es la dirección del segmento, se suma a un registro índice o apuntador, siendo éste el offset de la dirección.

La dirección real es la suma de la dirección del segmento y el offset de la dirección. Por ejemplo, si el segmento de datos (DS) tiene 1000H, entonces el segmento inicia en la localidad de memoria 10000H ya que se multiplica por 10. Entonces si el offset de la dirección contenida en el registro de base (BX) es 0010H, la dirección real es:

10010H = 1000H *10H + 0010H.

En la siguiente figura 2-10 se ilustra con un ejemplo el cálculo de la dirección real:

	SEGMENTO	DX	1000H
	OFFSET	BX	0010H
	DESPLAZAMIENTO (*10)		10000H
+	OFFSET	+	010H
			10010H

FIGURA 2-10

2.6 MODOS DE DIRECCIONAMIENTO

El 8088 utiliza nueve modos de direccionamiento para realizar operaciones sobre datos, memoria y dispositivos de E/S.

Modo Inmediato, el operando sigue inmediatamente al código de operación en la memoria del programa.

El direccionamiento de registro es rápido, ya que el operando puede estar en cualquiera de los cuatro registros de propósito general.

En el direccionamiento directo la dirección de la memoria del operando se da inmediatamente después del código de operación en la memoria de programa.

El direccionamiento basado en la dirección del operando se calcula tomando el contenido del registro BX, o BP, y sumándole el desplazamiento que sigue al código de operación del programa. El uso del registro BX implica indexación a través del segmento de datos, mientras que el uso del registro BP implica indexación a través del segmento de pila.

El direccionamiento indexado es similar al direccionamiento basado excepto que el desplazamiento se suma al contenido del registro SI, o DI, y siempre implica el uso del segmento de datos. Suman el contenido del registro BX, o del BP, al contenido del registro SI, o del DI, y al desplazamiento que sigue

al código de operación para encontrar la dirección del operando se denomina direccionamiento indexado basado.

El direccionamiento de cadena es un modo que se utiliza cuando las operaciones son realizadas sobre cadenas de datos. Dos direcciones, una fuente y otra destino, se calculan utilizando los registros SI y DI junto con los registros DS y ES. Los dispositivos de entrada y salida pueden ser accesados utilizando el modo de direccionamiento de E/S; este modo de direccionamiento utiliza el registro DX para sacar al bus de direcciones una dirección de E/S de 16 bits.

El 8088 tiene 25 modos diferentes de direccionamiento o reglas para localizar un operando de una instrucción. Para una misma operación se puede encontrar varios modos de direccionamiento, la operación puede referirse a un registro, a un operando inmediato o a un operando específico para una dirección de memoria, si ésta se declara de modo diferente.

Los modos de direccionamiento más comunes son:

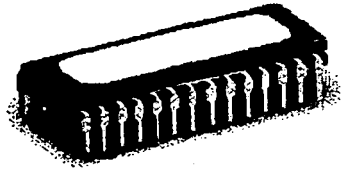
- Modo implícito.
- Modo de registro.
- Modo indirecto de registro.
- Modo inmediato.
- Modo de direccionamiento directo.
- Direccionamiento de la página cero.
- Direccionamiento de la página presente.
- Direccionamiento relativo.

- Direccionamiento indexado.
- Direccionamiento de registro base.
- Direccionamiento indirecto.
- Direccionamiento indirecto indexado.

Estos modos utilizan hasta cuatro cantidades para obtener la dirección de un operando en memoria:

- Dirección de segmento.
- Dirección de la base.
- Desplazamiento del índice.
- Desplazamiento constante.

PARTE 1



DISEÑO DEL HARDWARE

CAPITULO 3
ESPECIFICACION DE UN SISTEMA MINIMO

3.1 INTRODUCCION

Esta primera parte abarca los aspectos del hardware del sistema mínimo. Se definen como primer punto los componentes del sistema y la forma como se comunican entre sí; la organización del microprocesador y sus operaciones internas y externas.

¿ En qué consiste un sistema mínimo ?

Un sistema mínimo esta formado por un microprocesador, memoria e interconexiones de E/S. Los diferentes componentes que forman el sistema están enlazados por medio de buses que transfieren instrucciones, datos, direcciones e información de control entre los componentes.

El esquema de la figura 3-1 muestra el diagrama a bloques de un sistema mínimo. Este sistema solo tiene un microprocesador 8088. Una memoria RAM y una ROM que combinadas forman el tamaño de memoria. Las unidades de interconexión se comunican con dispositivos externos a través del bus de E/S. El microprocesador selecciona una de las unidades por medio del bus de direcciones. Los datos se transfieren de la unidad seleccionada al microprocesador vía el bus de datos. La información de control se transfiere por medio de líneas individuales, cada una especificando una función de control particular.

El microprocesador es la CPU (Unidad Central de Proceso) que interpreta códigos de instrucción recibidos de la memoria y ejecuta operaciones aritméticas, lógicas y de control basadas en datos almacenados en registros internos, palabras de memoria o unidades de interconexión. El microprocesador contiene un número de registros, una unidad lógica aritmética, una unidad de tiempo y una lógica de control. Externamente, éste entrega un sistema de buses para transferir instrucciones, datos e información de control hacia los módulos conectados con él.

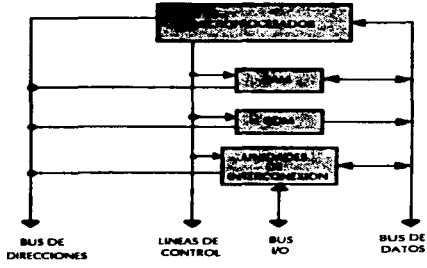


FIGURA 3-1

La unidad de memoria está formada por una RAM y una ROM. La RAM se utiliza para almacenar datos, parámetros variables y resultados intermedios que necesitan renovación y que están sujetos a cambio. La ROM se utiliza para almacenar programas y constantes que no están sujetas a cambios.

Las unidades de interconexión presentan los caminos necesarios para transferir información entre el microprocesador y los dispositivos externos de entrada/salida conectados al bus de E/S. El microprocesador recibe información de condiciones y datos de los dispositivos externos por medio de interfaces, que más adelante se explican.

La comunicación entre el microprocesador, memoria y demás dispositivos externos se lleva a cabo a través del bus de datos y el de direcciones. El bus de direcciones es unidireccional desde el microprocesador a otras unidades. La información que el microprocesador coloca en el bus de direcciones especifica una palabra de memoria particular en la RAM o ROM. El bus de direcciones se utiliza para seleccionar una de las diferentes unidades interconectadas al sistema o a un registro particular de una interface.

El bus de datos transfiere los datos del microprocesador a la memoria o interface y viceversa, la cual es seleccionada por el bus de direcciones. El bus de datos es bidireccional, lo cual significa que la información puede fluir en cualquier dirección.

3.2 DESCRIPCION DEL SISTEMA MINIMO

El sistema mínimo desarrollado en la presente tesis está basado en un microprocesador 8088 que se interrelaciona con otros dispositivos.

El sistema mínimo está formado por: Microprocesador, Generador de reloj, Dispositivos de memoria, Puertos de entrada y salida, Decodificador de direcciones.

En la figura 3-2 muestra el diagrama a bloques de los componentes del sistema mínimo: Microprocesador 8088 (1), bus de datos (2), bus de direcciones (3), bus de alimentación (4), señales de control (5), generador de reloj (6), dispositivos E/S (7), interfaces de E/S (8), unidad de memoria (9).

La operación adecuada de un microprocesador requiere que se presenten ciertas señales de control y de tiempo para lograr funciones específicas y que otras señales de control sean medidas para determinar el estado del microprocesador .

El bus de alimentación hace llegar la corriente proveniente de la fuente de alimentación a los diversos circuitos integrados del sistema. El bus de alimentación suministra un voltaje de +5v cd.

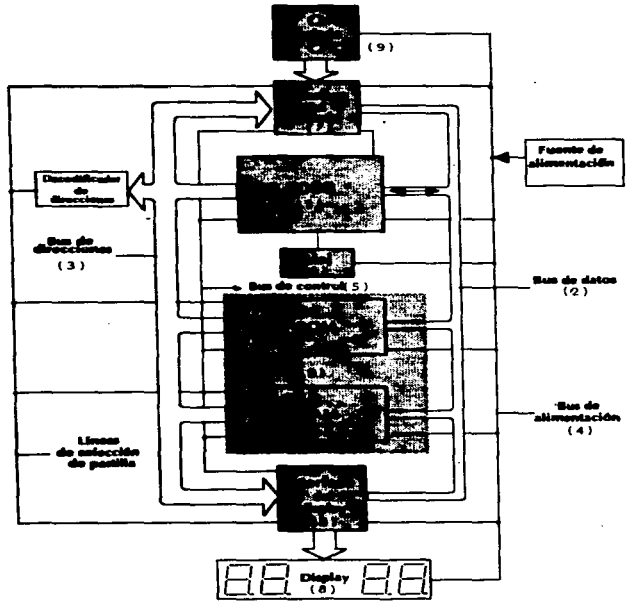

(9)
Fuente de alimentación
Bus de direcciones (3)
Bus de datos (2)
Bus de control (5)
Bus de alimentación (4)
Líneas de selección de pantalla
Display (6)

FIGURA 3-2

La unidad de memoria está formada por 2 tipos de memorias: RAM y EPROM.

El sistema mínimo cuenta con un teclado como dispositivo de entrada, conectado a su interface de comunicación con el resto del sistema. Esta interface está a cargo principalmente del 8279 y se encarga de almacenar y coordinar las entradas del teclado.

El sistema cuenta además con un visualizador (display) que trabaja como dispositivo de salida, su interface, al igual que el teclado, está a cargo del 8279.

Además de las interfaces de entrada y salida del teclado y display, el sistema tiene otro dispositivo de entrada/salida: el 8255 llamado PPI (Interface Periférica Programable), que proporciona los puertos del sistema.

Debido a que el sistema está formado por componentes que realizan funciones importantes para el sistema, éste se dividió en etapas para la descripción de cada una de las secciones principales, su función y actividad dentro del sistema mínimo.

Estas etapas son las siguientes:

ETAPA 1. Circuito de reloj.
ETAPA 2 Unidad Central de Proceso.
ETAPA 3 Bus de datos y direcciones.

ETAPA 4 Señales de control.

ETAPA 5 Unidad de memoria.

ETAPA 6 Puertos de E/S.

ETAPA 7. Periféricos.

ETAPA 1. Circuito de reloj.

En esta etapa se generan las señales de reloj que requiere el sistema: PCLK y CLK además de activar la señal RESET.

ETAPA 2. Unidad Central de Proceso.

Esta etapa la forma el Microprocesador 8088, que se encarga de organizar a los dispositivos que forman parte del sistema mínimo.

ETAPA 3. Bus de Datos y Direcciones.

Los buses de datos y direcciones juegan un papel importante para la comunicación y operación de todas las partes del sistema.

ETAPA 4. Lógica de control.

Las señales de control llevan información sobre temporización, órdenes de activación de memorias u otro dispositivo, dirección de datos (lectura o escritura), etc.

ETAPA 5. Unidad de memoria.

La memoria es otra etapa importante del sistema. Su misión consiste en almacenar datos y programas.

ETAPA 6. Puertos de entrada y salida.

Los puertos de entrada y salida se encargan de transferir datos de o hacia el microprocesador a un dispositivo periférico.

ETAPA 7. Periféricos.

Son dispositivos externos que se encuentran conectados al sistema a través de una interface.

CAPITULO 4
CIRCUITO DE RELOJ

4.1 INTRODUCCION

El sistema mínimo basado en el microprocesador 8088 requiere de una lógica adicional encargada de generar las señales de sincronización para todo el sistema.

La sincronización es muy importante para el sistema, ya que los eventos deben suceder en el tiempo exacto y en el adecuado para la ejecución de tareas y correcta coordinación de todo el sistema.

Para generar esta señal Intel diseñó el 8284A, un chip que genera los pulsos de reloj que se acopla perfectamente al 8088. Este chip además proporciona un inicio completamente sincronizado y un reinicio (reset) manual.

4.2 EL GENERADOR DE PULSOS DE RELOJ

El generador de reloj es un componente adicional para los microprocesadores 8086 y 8088. Si no hubiera un generador de reloj, se necesitarían muchos circuitos adicionales para generar la señal de reloj (CLK) en un sistema basado en el 8086/8088. El generador 8284A produce las siguientes funciones o señales básicas: generación de reloj,

sincronización de reinicialización, sincronización de la señal READY y señal periférica de reloj de nivel TTL.

Este generador de pulsos de reloj, 8284A de Intel, junto con un cristal oscilador externo, es un chip diseñado específicamente para estas tareas.

El 8284A, figura 4-1, es un chip con 18 terminales que se utiliza para generar los pulsos de reloj para el 8088 y sus periféricos (ver Apéndice A). Los pulsos de reloj determinan la velocidad de funcionamiento del sistema.

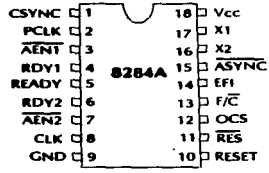


FIGURA 4-1

El generador de pulsos 8284A necesita un cristal oscilador, o una señal lógica externa como fuente de frecuencia. La opción se especifica conectando una de sus terminales o bien a tierra, o bien a la fuente de alimentación de 5 volts. La fuente de frecuencia proporciona una

frecuencia triple de la señal resultante del 8284A, es decir; la señal del oscilador debe ser 3 veces mayor que la señal de reloj deseada.

Para conseguir un rendimiento óptimo de los procesadores, los pulsos de reloj generados por el 8284A se mantienen a tensión alta durante un 33 % del período.

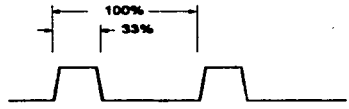


FIGURA 4-2

a) **Terminales del C. I. 8284A**

AEN_1 y AEN_2 (terminales 3 y 7). Habilitadores de direccionamiento. Estas terminales de entrada habilitan el bus de las señales ready RDY1 y RDY2, estas son usadas para provocar estados de espera a través de estas dos entradas.

RDY1 y RDY2 (4 y 6). Bus listo. Las terminales de entrada en conjunción con las terminales AEN1 y AEN2 provocan estados de espera en el sistema.

ASYNC (15). Selector de sincronización disponible. Terminal de entrada usada para seleccionar uno o dos estados de sincronización para las entradas RDY1 y RDY2.

READY (5). Listo. Esta terminal es conectada a la señal Ready del 8088 además está sincronizada con RDY1 y RDY2.

X1 y X2. (17 y 16). Entradas del cristal. En estas dos terminales de entrada se conectan un cristal externo usado como generador de reloj.

F/C (13). Frecuencia/cristal. Esta terminal de entrada es usada para seleccionar la fuente de señal para el 8284A. Si la terminal es enviada a alto indica que un reloj externo es proporcionado por la terminal EFI. Si esta terminal se conecta en bajo indica que se conecta un cristal en las terminales X1 y X2.

EFI (14) Entrada de frecuencia externa.

CLK (8). Clock. Terminal de salida que proporciona la señal de reloj para el 8088 con un frecuencia de 4.77 MHz.

PCLK (2). Reloj periférico. Terminal de salida que proporciona una señal de reloj que es una sexta parte del cristal o de la entrada EFI de frecuencia y que tiene un 50 % de duración del ciclo.

OSC (12). Salida del oscilador. Esta señal que tiene la misma frecuencia que la entrada del cristal o de EFI proporciona una señal a otros generadores.

RESET (10). Salida reset. Esta señal es conectada a la entrada de la terminal RST del 8088.

RES (11). Entrada reset. Esta terminal es conectada a un circuito RC para proveer la señal de reinicio.

CSYNC. Es una terminal usada siempre que la entrada EFI proporcione sincronización a sistemas con varios procesadores, si el OSC interno del cristal es usado esta terminal debe ser aterrizada.

b) **Operación del 8284A**

La figura 4-3 muestra el diagrama a bloques de la operación del 8284A.

Funcionamiento de la sección de reloj. La mitad superior del diagrama lógico representa la sección del reloj y de sincronización de reinicialización del 8284A. Como se ve en el diagrama, el oscilador del cristal tiene dos entradas: X1 y X2. Si se agrega un cristal a X1 y X2, el oscilador genera una señal de onda cuadrada a la misma frecuencia que la del cristal. La señal de onda cuadrada se alimenta a una compuerta AND y también a un inversor acoplador que produce la señal de salida OSC; ésta, a veces, se emplea como entrada EFI para otros 8284A.

Analizando la compuerta AND revela que cuando F/C es un 0 lógico, la salida del oscilador se dirige hasta el contador de dividir entre 3. Si F/C es un 1 lógico, entonces se envía EFI hacia el contador.

La salida del contador divisor entre 3 genera la temporización para la señal RDY; una señal a otro contador (divisor entre 2) y la señal CLK para el microprocesador. La señal CLK también se acopla para aumentar su capacidad de corriente antes que salga del generador de reloj. Los dos contadores en cascada producen la salida dividida entre 6 en PCLK, la salida del reloj periférico.

Funcionamiento de la sección de reinicialización. La sección de reinicialización consiste de un disparador Schmitt y un solo circuito flip-flop del tipo D; éste garantiza que se cumpla con los requisitos de temporización de la entrada RESET. Este circuito aplica la señal reset en el flanco negativo (transición de 1 a 0) de cada ciclo de reloj.

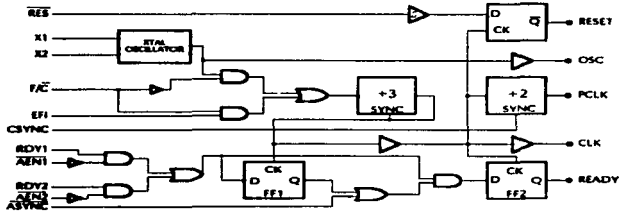


FIGURA 4-3

Existen 3 opciones para generar la señal de reloj con el 8284A. Se puede colocar un cristal en dos de sus terminales, con una señal generada externa, o con algún esclavo 8284A.

Las entradas X1 y X2, son las entradas del oscilador de cristal interno. La salida del oscilador es dividida entre 3 y sale a través del buffer. El reloj dividido también entre dos y se va a otro buffer.

La salida PCLK, proporciona una frecuencia que es la mitad de la salida CLK con 50 % del ciclo.

4.3 DISEÑO DEL CIRCUITO DE RELOJ

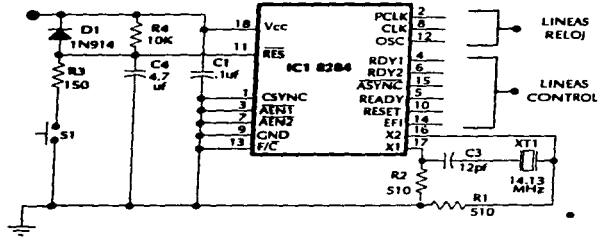


FIGURA 4-4

El "latido" del 8088 es un reloj 4.77 MHz proporcionado por el 8284A con los elementos correspondientes.

Este circuito utiliza un oscilador de cristal, por lo que la terminal F/C se conecta en bajo y EFI es ignorada.

Debido a que el oscilador interno es similar a un amplificador, se debe mantener la resistencia del circuito del cristal lo más baja posible y la ganancia del oscilador no decaiga. Se conectó un capacitor en serie con el cristal. Esto tiende a cancelar los efectos de defasamiento del oscilador. El valor del capacitor depende de la frecuencia del cristal, que para este caso la frecuencia es de 14.77 MHz. Se utiliza un capacitor de 12pf.

Las dos resistencias de 510Ω reducen el cambio de frecuencia debido a efectos presentes en el sistema, como excesiva capacitancia permanente en la tableta, variación de temperatura y fluctuaciones de voltajes, como se recomienda en el apéndice A.

Después de que el 8284A corra la entrada RES a través de un Schmit trigger antes de que haga cualquier cosa, se colocará un simple circuito RC y la longitud del pulso será el producto de RC. En este circuito se utilizó un capacitor de 4.7µf y la resistencia es de 10KΩ. El diodo opera como una válvula de seguridad para el circuito, de igual manera como se utiliza en una fuente de alimentación.

El 8284A provee una manera conveniente de generar pulsos de alimentación y de reset para el 8088.

El mínimo ancho de pulso que el 8088 requiere es de 50 µseg.

El reset manual se encuentra provisto de un push button. Cuando éste es presionado, RES se va a tierra (nivel de 0v) y el 8284A envía un pulso de reset al 8088.

El circuito RC provee un 0 lógico hacia la entrada RES va un 1 lógico porque el capacitor se carga hasta +5V a través de la resistencia de 10kΩ.

Se utiliza un push button, éste lleva al microprocesador la señal reset por el usuario.

El tiempo correcto de reset requiere que la entrada reset vaya a 1 lógico no después de 4 ciclos de reloj después que el sistema de alimentación es aplicado y retenido en alto hasta después de 30 µseg. El flip flop se encarga de que el reset vaya a alto en 4 ciclos y la constante de tiempo asegura que permanezca en alto hasta después de 50 µseg.

CAPITULO 5
UNIDAD CENTRAL DE PROCESO

5.1 INTRODUCCION

En esta sección se describe la operación de las terminales que participan en la configuración para modo mínimo del 8088 y su función como Unidad Central de Proceso.

El microprocesador es un circuito electrónico de alto grado de integración de componentes, el cual es capaz de reconocer y ejecutar instrucciones propias de un CPU.

5.2 FUNCIONES DEL CPU

El CPU contiene los elementos de memoria denominados registros y circuitería computacional denominada unidad aritmética y lógica (ALU). También contiene circuitería de decodificación de instrucciones y una sección de temporización y control, además de las conexiones necesarias de entrada y salida.

Las funciones principales del CPU son:

1. Buscar, decodificar y ejecutar instrucciones del programa en el orden adecuado.
2. Transferir datos entre memoria y las secciones de entrada/salida.

3. **Responder a las interrupciones externas.**
4. **Proporcionar señales de temporización y control del sistema completo.**

La mayoría de los CPU de los microprocesadores contienen como mínimo los elementos mostrados en la figura 5-1. Las secciones principales incluyen diversos registros, la unidad lógica aritmética, el decodificador de instrucciones y la sección de temporización de control junto a las entradas y salidas.

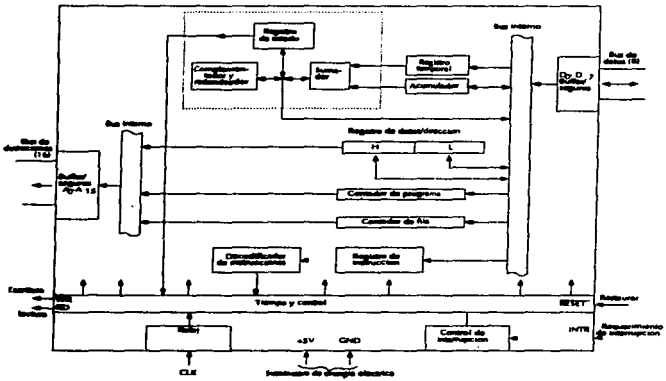


FIGURA 5-1

La Unidad lógica aritmética (ALU) del CPU realiza operaciones como sumar, desplazar, comparar, incrementar, decrementar, negar, etc. Algunas de las secciones funcionales de una ALU típica se muestran en la figura 5-2. La ALU contiene un sumador y desplazador, que está conectado al acumulador, vía bus interno de datos.

El registro de estado localizado en la ALU es también llamado registro de códigos de condición o de señalizadores. El registro es realmente un grupo de flip flops individuales que pueden ser puestos a 1 o 0 según las condiciones creadas por la última operación de la ALU. Los flip flops individuales, o señalizadores, incluyen indicadores para cero, resultados negativos, arrastres des el MSB, etc. Los señalizadores son utilizados para tomar decisiones cuando posteriormente se emplean instrucciones de bifurcación.

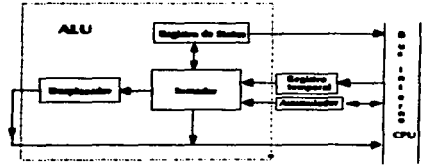


FIGURA 5-2

La sección de Temporización y Control de la figura 5-1 es la sección más compleja del CPU. Maneja todos los eventos del CPU.

El contador de programa se muestra como parte del CPU de la figura 5-1. Este es responsable de seguir la pista de la dirección de siguiente instrucción de memoria que va a buscar.

La secuencia de búsqueda-decodifica-ejecuta la instrucción es fundamental para la operación del sistema. La primera instrucción buscada, de la memoria de programa, es el código de operación de la primera instrucción y es colocada en el registro de instrucciones por la sección de control del CPU .

La organización del CPU mostrada en la figura 5-1 es una versión simplificada de ésta.

5.5 TERMINALES DE CONEXION EN MODO MINIMO

La operación del 8088 en modo mínimo se obtiene conectando la terminal MN/MX directamente a 5 Vcd.

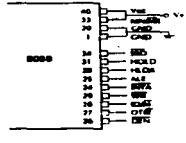


FIGURA 5-2. Señales del 8088 en modo mínimo.

Por su actividad se dividen en 3 grupos: terminales de control, de direccionamiento y datos y de alimentación y reloj.

a) TERMINALES DE CONTROL

Estas señales envían las instrucciones de operación a todos los dispositivos involucrados.

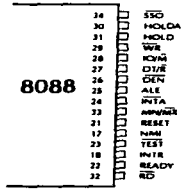


FIGURA 5-3

b) TERMINALES DE DIRECCIONAMIENTO

Las terminales de direccionamiento del 8088 están divididas funcionalmente en 3 grupos: AD0 a AD7, A8 a A15 y A16 a A19. El primer grupo, de AD0 a AD7 es el más versátil. Durante la primera parte de un ciclo de instrucción, el 8088 pone la parte menos significativa de la dirección en estas terminales. Tan pronto como el

CPU obtiene la segunda parte de ciclo de instrucción, aparta esta dirección de la línea y flotan en tercer estado para obtener la lectura de siguiente trabajo. Cuando el 8088 llega a la tercera parte de este ciclo de instrucción, es usada esta terminal para la línea de datos. Estas terminales tienen la capacidad de ser bidireccionales durante este estado del ciclo de instrucción.

El segundo grupo, de A8 a A15, son un conjunto de direcciones, con estado válido para el ciclo de instrucción completo.

El último grupo, de A16 a A19, estas líneas llevan datos válidos de dirección al mismo tiempo que el primer grupo de direcciones (el inicio de cada ciclo de instrucción), y después lleva señales de estado los cuales indican el segmento de dirección en uso actual, (S3 y S4), el estado de la bandera de interrupción habilitada, (S5), y una bandera que indica que el 8088 tiene control del bus, ver figura 5-4.

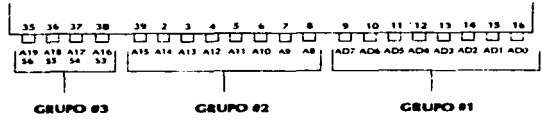


FIGURA 5-4 Terminales de direcciones y datos.

c) TERMINALES DE ALIMENTACION Y RELOJ

Existen dos terminales de tierra (GND). El voltaje VCC debe ser 5VCD +/- 10 % de tolerancia. Es conveniente conectar un pequeño capacitor de bypass (0.1μF). Esto es importante porque ambas tierras no están eléctricamente conectadas en el chip, (ver figura 5-5).

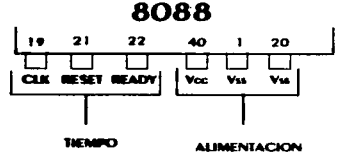


FIGURA 5-5

Terminales de reloj. Se obtienen tres señales del Generador de Pulsos de Reloj 8584 hacia el procesador. Estas tres señales son CLK (señal de reloj), RESET y la READY. Las dos últimas señales están sincronizadas con CLK.

La misión de la señal RESET es la de reinicializar los valores del sistema como si ésta se hubiese apagado y vuelto a encender. Esta señal imprescindible para solventar los casos en que un programa se mete en un ciclo infinito.

La función de la señal READY es la de sincronizar el procesador con los dispositivos externos más lentos. La señal READY va desde el dispositivo externo al procesador, pasando a través del generador de pulsos de reloj. Cuando el procesador quiere accesar a un dispositivo que no está preparado para la transferencia, el dispositivo envía un 0 por la línea READY. Por el procesador recibe esta señal, entra en un ciclo de espera hasta que aparece un 1 por dicha línea.

El 8284A produce otra señal, PCLK (reloj periférico), que funciona a la mitad de la frecuencia de CLK, con un ciclo de trabajo al 50% (un 50% a nivel de tensión alta y un 50% a nivel 0).

La figura 5-6 muestra las conexiones de tiempo y alimentación hacia el 8088.

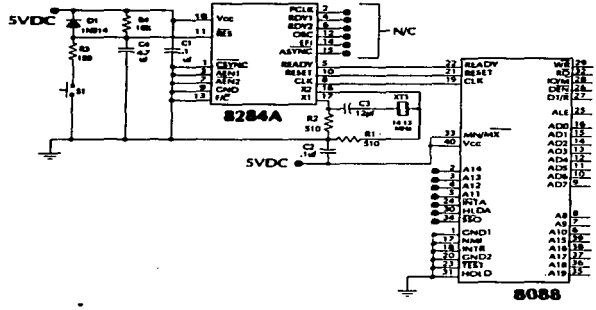


FIGURA 5-6 Diagrama del circuito de reloj hacia el 8088.

CAPITULO 6
BUS DE DATOS Y DIRECCIONES

6.1 INTRODUCCION

El bus de datos/direcciones del 8088 es multiplexado para tener el menor número de terminales requeridas para el integrado del microprocesador. Por lo que se tiene la tarea de demultiplexar o extraer la información contenida en las terminales multiplexadas.

Antes de conectar el 8088 con las interfaces de memoria y de I/O, es necesario demultiplexar los buses de datos y de direcciones.

6.2 LOS BUSES DEL 8088

El sistema tiene tres buses: (1) un bus de direcciones que dispone a la memoria e I/O con la dirección de la memoria o con el número de puerto I/O, (2) un bus de datos que transfiere datos hacia y desde la memoria e I/O del sistema, y (3) un bus de control que proporciona la información de control a la memoria e I/O.

6.3 DEMULTIPLEXION DE LOS BUSES

Para demultiplexar los buses se utilizan dos 74LS373 (Latches

transparentes) son utilizados para demultiplexar las conexiones del bus de direcciones/datos AD0-AD7 y las conexiones multiplexadas de dirección/datos A19/S6 - A16/S3.

Estos latches transparentes los cuales son como alambres, cuando los latches de dirección son habilitados por la terminal ALE van a 1 lógico y pasan las entradas a las salidas. Después, ALE regresa a la condición de 0 lógico lo cual provoca que los latches recuerden las entradas al mismo tiempo que cambian a 0 lógico. En este caso, A7 - A0 son recordadas en la parte baja del latch y A19 - A16 en la parte alta del latch. Esto produce un bus de direcciones separado con conexiones independientes de A19 a A0, éstas llevan al 8088 a manejar 1 Mbyte de espacio de direccionamiento de memoria.

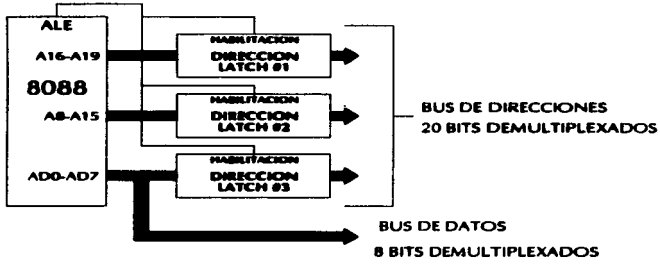


FIGURA 6-1

6.4 DEMULTIPLEXACION PARA EL 8088

La memoria e I/O requieren que las direcciones permanezcan válidas y estables durante una lectura o una escritura.

Los latches de direccionamiento son necesarios para "grabar" o mantener fija la información de las direcciones durante T1 de un ciclo de máquina. La figura 6-2 muestra el diagrama del sistema de bus de datos y direcciones.

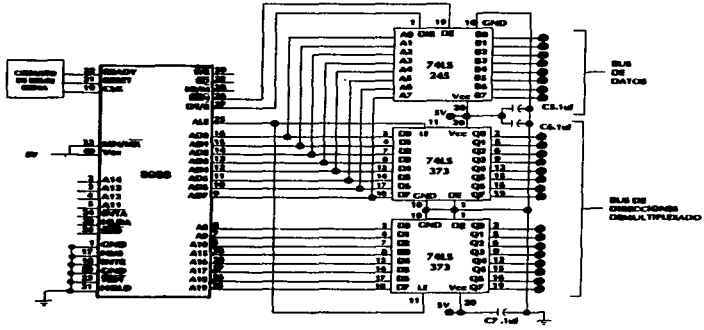


FIGURA 6-2

AD0 - AD7 y A16 - A19 son conectados a las entradas de los 373's. La señal ALE se utiliza para habilitar éstos. La información de las direcciones, es retenida en la salida de los latches después de que ALE está en bajo.

El 8088 puede controlar directamente los 373's que son latches con la señal ALE. Una vez que esta señal se encuentra activa, en alto, la terminal #1 de los 373's es habilitado. Si conectamos esas líneas juntas los 373 latcharán las direcciones cuando el 8088 indique que el bus de direcciones tiene una dirección válida. La otra terminal de control es el control Output Enable (terminal #1), esta es una señal activa en bajo, que está conectada en todo momento a tierra.

El 74245 se utiliza como un fusible octal. Está aislado el bus de datos del 8088 del resto del circuito y, como el 373, tiene salidas que pueden estar en tercer estado. Las entradas y salidas del 245 son activadas con el control de dirección (terminal #1DIR). Cuando ésta línea se encuentra en alto el 245 transmite datos desde lasterminales A a las terminales B y si ésta línea está esta en bajo la dirección podría ser revertida.

El 8088 desde DT/R siempre dejará conocer sí el 8088 quiere transmitir o leer datos en el bus. Conectando las entradas A del 245 a las salidas de datos del 8088 y usando DT/R para controlar la señal DIR del 245, los buffers del 245 estarán siempre apuntando a la dirección correcta. El control OE en el 245 es exactamente el mismo como en el 373, pero es

manejado directamente con DEN. Cuando la línea DEN del 8088 es activa es una indicación de que la dirección es demultiplexada y las líneas de datos (terminal #9 a la terminal # 16) están siendo utilizadas para datos. Esta señal controla las salidas del 245 significa que las salidas del 245 sólo serán habilitadas cuando el 8088 va a esperar a recibir datos o transmitirlos.

Una vez que el bus de datos esta demultiplexado, puede ser conectado a algún dispositivo periférico o a un componente de memoria.

CAPITULO 7
SEÑALES DE CONTROL

7.1 INTRODUCCION

Las unidades de interconexión presentan los caminos necesarios para transferir información entre el microprocesador y los dispositivos externos de entrada y salida conectados al bus de I/O. El microprocesador recibe información de condiciones y datos de los dispositivos externos por medio de la interconexión. Este responde, a su vez, enviando información de control y datos para los dispositivos externos.

7.2 PRINCIPALES SEÑALES DE CONTROL

La operación adecuada de un microprocesador requiere que se presentan ciertas señales de control y tiempo para lograr funciones específicas y que otras señales de control sean medidas para determinar el estado del microprocesador.

La figura 7-1 muestra un diagrama funcional completo de todas las señales de control, el bus de datos, bus de direcciones y alimentación.

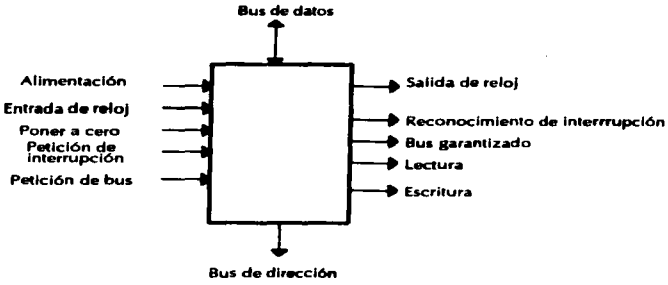


FIGURA 7-1 Señales de control.

La terminal de entrada de petición del bus es una solicitud al microprocesador para suspender su operación y llevar todos los buses a su estado de mayor impedancia. Una vez reconocida la solicitud, el microprocesador responde habilitando la línea de salida de control del bus. Así cuando un dispositivo externo desea transferir la información directamente a la memoria, éste solicita que el microprocesador abandone el control del bus común. Una vez que el bus sea inhabilitado por microprocesador el dispositivo que originó la petición toma el control sobre el bus de direcciones y datos para conducir las transferencias de memoria sin la intervención del procesador. Esta característica se llama *acceso directo a memoria* (DMA).

Lectura y escritura son líneas de control que informan el componente seleccionado por el bus de direcciones de la dirección de la transferencia esperada en el bus de datos. La línea de escritura indica que el procesador está en el modo de salida y que los datos válidos están disponibles en el bus de datos.

El sistema requiere de la combinación de señales para el control de los dispositivos.

Las secciones que requieren de la combinación de señales para controlar su operación son la unidad de memoria y la unidad de E/S.

7.3 CIRCUITERIA DE CONTROL PARA LA UNIDAD DE MEMORIA

El 8088 no sabe que memoria tiene en su mapa de direcciones. Se puede poner la RAM, ROM u otro dispositivo de I/O en cualquier posición en el bus pero se necesita algún método para generar las diferentes combinaciones de las señales de control que se necesitan para cada tipo diferente de memoria.

La primera consideración es habilitar la RAM en un solo tiempo cuando el 8088 quiere utilizarla, las terminales que participan son las líneas de direccionamiento de la A11 a A19. Considerando la distribución del mapa de memoria esas líneas estarán en bajo cuando se comunique con la RAM y en alto en comunicación con la ROM. Conmutando la línea A19 la línea

más significativa, cuando esta línea se encuentra en bajo direcciona la parte alta de 512k de espacio de direccionamiento del 8088. y los otros 512k direccionan a la ROM .

Ya definida la habilitación para el acceso de la RAM. La línea IO/M indica que tipo de operación se realizará, si ésta línea se encuentra en alto se realiza operación de entrada/salida y si la línea se encuentra en bajo realiza una operación de la memoria.

Se toman en cuenta otras dos líneas que realizan operaciones importantes éstas son WR escritura, RD lectura.

Como resultado de las señales más importantes obtenemos la siguiente tabla de verdad.

TABLA DE VERDAD PARA LAS SEÑALES DE CONTROL

ENTRADAS				SALIDAS					
IO/M	A19	WR	RD	RAMSEL	RAMRW	ROMSEL	ROMRD	IOSEL	ACTIVIDAD
L	L	H	L	L	H	H	L	L	RAMRD
L	L	L	H	L	L	H	H	L	RAMWR
L	H	H	L	H	H	L	L	L	ROMRD
L	H	L	H	H	L	L	H	L	- -
H	X	X	X	H	H	H	H	H	IOSEL

TABLA No. 7-1

Para seleccionar la RAM se obtiene la señal RAMSEL se activa cuando la línea A19 y IO/M se encuentren en bajo. Para activar la ROM se obtiene mediante la señal ROMSEL será cuando IO/M se encuentre el bajo y A19 en alto.

Como resultado de la tabla de verdad anterior se obtiene la siguiente circuitería de control.

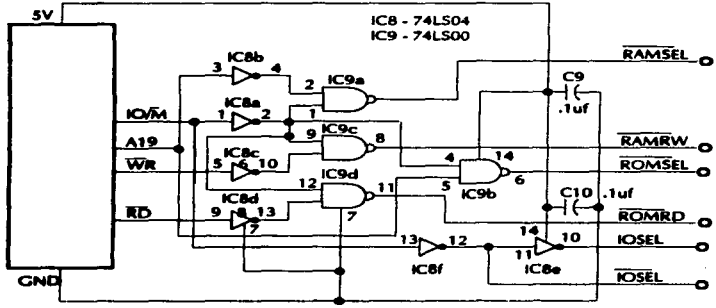


FIGURA 7-2 Lógica de control.

7.4 CONTROL ENTRADAS/SALIDAS

El primer paso en el diseño de la circuitería de control es decodificar las señales provenientes del 8088 y obtener dos señales importantes. Las señales que se requirieron son IORD e IOWR, lectura y escritura respectivamente para I/O.

TABLA DE VERDAD PARA LA LOGICA DE CONTROL I/O					
ENTRADAS			SALIDAS		
IOSEL	RD	WR	IORD	IOWR	ACTIVIDAD
LOW	HIGH	LOW	HIGH	LOW	IOWR
LOW	LOW	HIGH	LOW	HIGH	IORD
LOW	LOW	LOW	---	---	No Importa
LOW	HIGH	HIGH	---	---	Estado imposible
HIGH	X	X	HIGH	HIGH	Operación de memoria

TABLA No. 7-2

La manera de realizar la lógica combinacional se realiza de la misma manera que para las señales de control para la unidad de memoria.

Para obtener estas señales se realizó la lógica combinacional mostrada en la tabla 7-2.

Las compuertas OR generan las dos señales que se necesitan(ver figura 7-3). Se generarán sólo cuando el 8088 haga una solicitud de puerto de E/S.

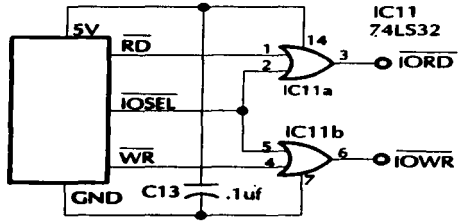


FIGURA 7-3 Lógica de control para E/S.

CAPITULO 8
UNIDAD DE MEMORIA

8.1 INTRODUCCION

Una unidad de memoria es una colección de registros de almacenamiento que conjuntamente con los circuitos asociados transfieren información adentro y afuera del microprocesador.

8.2 UNIDAD DE MEMORIA

La memoria es otro dispositivo que forma parte del sistema. Su misión consiste en almacenar, básicamente datos y programas. La memoria puede verse como una colección de celdas individuales, las cuales llevan asociado un número al que se le da el nombre de dirección. La celda no almacena su dirección; sino el microprocesador y los circuitos de control se sirven de la dirección para localizar y seleccionar una celda particular en la memoria. Todas las transferencias de información de celda a celda o entre celdas y el microprocesador se hacen vía del bus de datos, utilizando el bus de direcciones para seleccionar las celdas y el bus de control para iniciar y realizar el proceso.

8.3 TIPOS DE MEMORIAS

En los sistemas con microprocesadores se utilizan dos tecnologías principales de memoria. La memoria activa RAM es una memoria de lectura/escritura. Su contenido puede leerse o modificarse en cualquier momento. Su principal inconveniente es su volatilidad: cuando se interrumpe la alimentación se pierde el contenido de la memoria. La memoria activa se utiliza pues esencialmente para almacenar datos.

El segundo tipo de memoria usado es la memoria pasiva (ROM), que es una memoria de sólo lectura. Una vez definido el contenido de esta memoria no puede modificarse. Siempre es posible leerla, pero no puede escribirse una vez grabada la información. No es volátil y se utiliza para almacenar programas.

A) MEMORIA RAM

La **RAM** (Memoria de Acceso Aleatorio) opera como dispositivo de almacenamiento temporal de lectura/escritura.

La memoria activa se designa por RAM "Memoria de Acceso Aleatorio". Dos son las tecnologías usadas en la fabricación de memorias activas: la estática y la dinámica.

La memoria RAM estática emplea un flip flop para almacenar cada bit de información. Su contenido permanece estable indefinidamente hasta mientras no se corte la alimentación.

La memoria dinámica almacena los diferentes bits de información en forma de cargas eléctricas. Para poder retener la información almacenada en la RAM dinámica hay que proceder a refrescar la memoria cada 1 o 2 milisegundos.

B) MEMORIA ROM

Una memoria ROM es esencialmente un dispositivo en el cual se almacena un conjunto fijo de información binaria. La información binaria debe especificarse por el usuario; ésta, viene con enlaces internos especiales que pueden estar fusionados para formar los caminos del circuito necesarios.

La memoria ROM almacena permanentemente programas que son residentes para el sistema y no cambian cuando el circuito es desconectado.

La ROM es disponible en varias formas: La EPROM (Memoria Programable de Solo Lectura Borrable), es más comúnmente utilizada cuando el software almacenado puede ser cambiado. Las EPROM's pueden ser programadas en un dispositivo llamado programador de EPROM's. Estas memorias pueden ser borradas exponiéndolas a rayos ultravioleta en alta densidad alrededor de 30 minutos o menos,

dependiendo del tipo de EPROM. Las PROM (memorias Programables de sólo lectura) pueden también ser programadas mediante la quema de fusibles de nicromo u óxido de silicio, pero una vez programadas ya no se puede borrar.

La **EPROM** (Memoria Programable Borrable de Solo Lectura) es la memoria permanente que contiene el programa monitor del sistema.

En la memoria RAM se puede accesar directamente a cualquier celda de la memoria especificando su dirección. La diferencia entre ambas reside en el hecho de que, mientras en una RAM se puede leer y escribir (almacenar y extraer información), en las ROM sólo se puede leer. La información se almacena en las ROM durante un proceso especial llamado de programación o reprogramación.

El programa almacenado en la parte de la ROM de un sistema con microprocesador es un programa en lenguaje ensamblador. Como la RAM es una memoria volátil, la cortar el suministro de potencia y activarlo de nuevo se destruye la información almacenada en ella. La ROM es una memoria no volátil y el programa almacenado en ella está disponible cada vez que se le suministre potencia.

8.4 CONEXIONES DE LA UNIDAD DE MEMORIA.

Las conexiones que todo dispositivo de memoria tienen en común son entradas de dirección, salida o entrada de datos, algún tipo de entrada de

selección y al menos una entrada de control para leer o escribir datos.

A) CONEXIONES DE DIRECCIONAMIENTO

Los los dispositivos de memoria tienen un grupo de entradas de dirección utilizadas para seleccionar una localidad de memoria dentro de los dispositivos de memoria. El número de terminales de dirección encontrados en un dispositivo de memoria es determinado por el número de localidades de memoria encontrados en éste.

B) CONEXIONES PARA DATOS

Las conexiones de datos son puntos en los cuales los datos son registrados para almacenarlos o extraerlos para su lectura. En este sencillo dispositivo de memoria hay ocho conexiones I/O lo cual significa que el dispositivo de memoria puede almacenar 8 bits de datos en cada una de sus localidades de memoria.

Un dispositivo de memoria de 8 bits es llamado también memoria de 1 byte de ancho. No todos los dispositivos de memoria son de 8 bits de ancho sin embargo, algunos son de 4 bits o también de 1 bit.

C) CONEXIONES DE SELECCION

Cada dispositivo de memoria tiene una entrada (algunas veces más de una), para seleccionar o habilitar los dispositivos de memoria. Esta especie de entrada es Chip Select (CS) o CHIP ENABLE (CE). La RAM

generalmente tiene la entrada **CS** o **S** en bajo y la ROM un **CE**.

Si el **CE** o **CS** es activado (con un 0 lógico), el dispositivo de memoria puede realizar una lectura o escritura. Si esta entrada no se encuentra activada la memoria no podrá realizar ninguna operación.

D) CONEXIONES DE CONTROL.

Todos los dispositivos de memoria tienen una entrada de control. Una ROM tienen usualmente sólo una entrada mientras que una RAM cuenta con dos entradas de control.

La entrada de control en una ROM es la conexión de salida de habilitación **OE**.

La memoria RAM puede tener una o dos señales de control. Si hay una entrada de control esta es R/W. Este pin selecciona una operación de lectura o escritura siempre y cuando CS esté activada. Si la RAM tiene dos entradas de control WE y OE, WE puede ser activado para realizar una escritura a memoria. Cuando alguna de esas dos señales de control no estén presentes no se puede realizar alguna operación.

8.5 LA UNIDAD DE MEMORIA

La unidad de memoria para el sistema mínimo esta formada por una SRAM 6116 y una EPROM 2716.

La figura 8-1 ilustra a una EPROM y una RAM. Estos dispositivos contiene 11 entradas de direcciones y 8 salidas de datos. A estos dispositivos por su capacidad se les llama también memorias de 2K X 8.

El sistema tiene 4K de memoria: 2k de memoria ROM y 2k de memoria RAM.

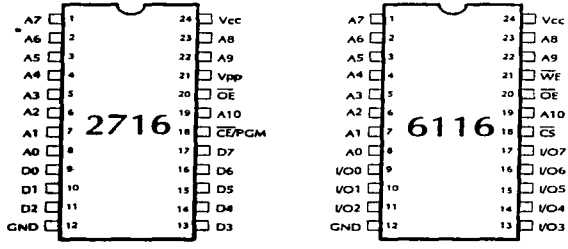


FIGURA 8-1 Terminales de las memorias EPROM y SRAM.

8.6 MAPA DE MEMORIA

Al desarrollar el diseño del circuito de control se hace la distribución del mapa de memoria del sistema. De esta manera se decide la distribución de los dispositivos que la ocupan. Se tiene un total de direcciones de la 00000h hasta FFFFFh. Se utilizarán 4k de memoria dividida entre la memoria RAM y ROM.

Cuando se alimenta al 8088 tiene que buscar en algún lugar la primera instrucción. Esto es una dirección alambrada en el CPU que el microprocesador coloca en el bus cada vez que sea arrancado el sistema.

Después el 8088 ejecuta automáticamente la instrucción, el 8088 encuentra la dirección absoluta FFFF0h, se colocó la memoria permanente en lo alto del rango de direccionamiento. El 8088 ejecuta instrucciones secuenciales en la memoria cuenta con 16 localidades desde la localidad de encendido de FFFF0h hasta el término del rango de direccionamiento FFFFFh.

Poniendo la ROM en la parte alta de la memoria se almacena una instrucción en la localidad de encendido. El 8088 se habilitará para direccionar alguna localidad desde FF800h a FFFFFh. La RAM para el sistema es del mismo tamaño, ésta se localiza en la parte más baja del rango de memoria, direccionando desde la localidad 00000h hasta 007FFh (figura 8-2).

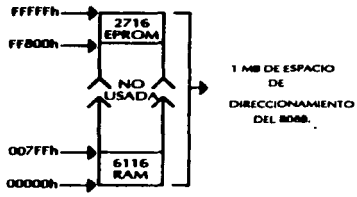


FIGURA 8-2

Localizando a estos dos dispositivos en el mapa de memoria del sistema es más fácil conectar las direcciones y líneas de datos.

La RAM y la ROM son conectadas a las 11 líneas de direccionamiento. La diferencia que existe en el direccionamiento entre la RAM y la ROM son las líneas de direccionamiento (A11 a A19). Ya que ambas memorias sólo tienen 11 pines de direcciones y están conectados a las mismas terminales del 8088, en estas condiciones si se realiza un acceso a memoria la RAM y la ROM ponen al mismo tiempo datos en el bus. Para evitar ésto se necesita construir un circuito de control que seleccione y habilite automáticamente a los dispositivos que componen la memoria.

Para lograr ésto primero se estudian las líneas de control de los dispositivos. La memoria 6116 tiene dos terminales de control y una terminal para habilitar la escritura. La línea chip select pin #18 habilita al

chip. Cuando es activado esta en "stand by" y las salidas están flotando en tercer estado. La línea **Enable Output** pin # 20 controla los datos, este pin es necesario porque el 6116 usa las mismas terminales para los datos de entrada y los datos de salida. Cuando OE esta en bajo las líneas de datos son los pines de salida y cuando esta línea se encuentra en alto las líneas son terminales de entrada. La línea **Write Enable**, terminal # 21, cuando el chip es seleccionado y esta línea esta en bajo el 6116 almacena los datos que están presentes en la localidad direccionada (figura 8-3).

Cuando la línea está en alto los pines de salida llevan los datos almacenados a la localidad de direccionamiento.

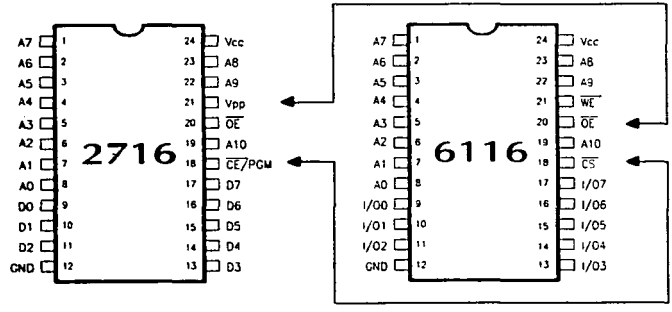


FIGURA 8-3

Se conectan juntos el **Chip Select** y el pin **Output Enable** porque no hay necesidad de tener el control de los pines por separado. Cuando se selecciona el chip los datos esta en el bus y son controlados por el pin Write Enable.

8.7 AGREGANDO LA UNIDAD DE MEMORIA

La primera consideración es encontrar alguna manera de que sólo se active la RAM cuando lo solicite el 8088. Se utiliza algún pin de dirección de la A11 a A19. Considerando la manera en que está mapeada la memoria, esas líneas pueden estar en bajo cuando se tiene acceso a la RAM y en alto cuando se accesa a la ROM. Seleccionando A19, la línea más significativa, esta línea está en bajo cuando accesamos abajo de 512K del espacio de direccionamiento del 8088 y en alto cuando solicitamos la parte alta de la memoria. De esta manera se podrá habilitar hasta 512K de RAM y 512K de ROM sin modificar la circuitería.

El control del circuito debe ser seguro, de que solo selecciona uno de los dispositivos a la vez. El 8088 utiliza la línea IO/M para indicar al sistema que operación se va a ejecutar.

Para seleccionar la RAM se selecciona cuando las líneas A19 y IO/M del 8088 están en alto y esto se muestra en la figura 8-4. La línea RAMSEL habilita al sistema RAM siempre que el 8088 quiera accesarla. Lo mismo sucede para generar RAMRW, esta línea se activa siempre que se desee realizar una lectura o escritura.

Las señales para la ROM son similares, hay un inversor de diferencia entre **ROMSEL** y **RAMSEL**. La línea **ROMRD** es producida por la combinación de las líneas IO/M y RD desde el 8088. Cuando se desee hacer una lectura desde la ROM, **ROMSEL** habilita la memoria apropiada y ROMRD habilita las salidas.

CAPITULO 9
PUERTOS DE ENTRADA Y SALIDA

9.1 INTRODUCCION

Esta sección de E/S permite al sistema enviar datos al mundo exterior o recibir datos de este. Los periféricos tales como el teclado y display y el PPI son conectados en esta sección. Estos permiten al usuario comunicarse con el mundo exterior. A la interface entre el microprocesador y los dispositivos de E/S son llamados puertos.

9.2 INTERFACES BASICAS DE ENTRADA/SALIDA

Una operación de entrada o salida es el acto de transferir datos a o desde un dispositivo periférico seleccionado. El microprocesador es el foco de todas las operaciones, por tanto una entrada significa que el dato fluye hacia el microprocesador, mientras que una salida significa que el dato fluye del microprocesador. Donde se realiza esta acción, en la que el dato sale o entra se denomina puertos de entrada o salida.

El microprocesador a través del software utiliza las instrucciones IN (entrar) y OUT (salir) para transferir datos a y desde los puertos de entrada/salida. Estas instrucciones de transferencia se mencionan en el siguiente capítulo.

9.3 TECNICAS DE INTERFACE PARA E/S

Existen 3 opciones para el manejo de E/S que son:

- Utilizando las instrucciones de entrada/salida
- Aislar las entradas/salida
- Mapa de memoria

A) INSTRUCCIONES IN/OUT

El 8088 contiene una instrucción para transferir información hacia un dispositivo E/S (OUT) y otro para leer información desde un dispositivo de E/S (IN).

Ambas instrucciones transfieren datos entre un dispositivo de entrada/salida y el microprocesador. La dirección de E/S es almacenada en el registro DX como una dirección E/S de 16 bits o en el byte inmediatamente seguido por el código como un direccionamiento de 8 bits. Intel llama a la dirección de 8 bits compuesta porque es almacenada con una instrucción, en una ROM. La dirección de 16 bits es llamada variable porque esta es almacenada en un registro, el cual puede ser cambiado.

Sin embargo, los datos son transferidos utilizando las instrucciones IN u OUT, las direcciones de E/S, a menudo llamado como número de puerto,

que aparece en el bus de direcciones. La interface externa de E/S entonces decodifica de la misma manera como una dirección de memoria es decodificada. Si el número de puerto es de 8 bits, entonces aparece en A7-A0, y si es de 16 bits, aparecerá en A15 - A0.

B) METODOS: AISLADO Y MAPA DE MEMORIA DE ENTRADA/SALIDA

Hay dos métodos completamente diferentes de interfaces de E/S para el microprocesador: E/S aisladas y mapa de memoria de E/S. En el E/S aislado (o directo), IN y OUT transfieren datos entre el microprocesador y un dispositivo de E/S. En el mapa de memoria de E/S alguna instrucción de referencia de memoria acompaña la transferencia. Ambos métodos son usados en el presente sistema mínimo.

E/S AISLADO. La técnica más común de transferencia de E/S, es aislar las entradas/salidas, en el cual los dispositivos de entrada/salida en el sistema son aislados desde la memoria. La figura muestra los mapas de memoria y aislado. Las direcciones de los dispositivos aislados de E/S, llamados puertos E/S son separados desde la memoria. Como resultado se expande la memoria a 1 Mbyte sin utilizar algún espacio de memoria para los dispositivos de E/S. Una desventaja de aislar las E/S es que los datos transferidos entre los dispositivos de E/S y el microprocesador puede estar en AL o AX. La señal IO/M indica que el bus de direccionamiento contiene una dirección de memoria o un número de puerto.

Con la organización del E/S aislado, el microprocesador especifica en sí mismo cuando la dirección en el bus de direcciones es para una palabra de memoria o para un registro de interconexión. Esto se realiza por medio de la línea de control E/S. El microprocesador debe entregar instrucciones de entrada y salida diferentes y cada una de ellas debe asociarse con una dirección. Cuando el microprocesador busca y decodifica el código de operación de una instrucción de entrada o salida, éste lee la dirección asociada con la instrucción y la coloca en el bus de direcciones. Al mismo tiempo hace la línea de control IO/M igual a 1 para informar a los componentes externos que esta dirección es para una interconexión y no para la memoria.

Este método aisla la memoria y las direcciones E/S de manera que no afecte el espacio de memoria por la asignación de la dirección de la interconexión.

MAPA DE MEMORIA E/S. Como se ha mencionado estos dos métodos no utilizan las instrucciones IN y OUT para transferir los datos. Más bien, utiliza alguna instrucción que transfiere datos entre el microprocesador y la memoria. Un mapa de memoria para un dispositivo de E/S es tratado como una localidad de memoria dentro del mapa de memoria. La principal desventaja de esta técnica de E/S es que utiliza la parte alta del espacio de memoria. Una ventaja es que la señal IO/M no necesita ser decodificada.

9.4 EL PPI 8255

El 8255 Interface Periférica Programable (PPI) es muy popular y de bajo costo, es un circuito integrado de 40 terminales.

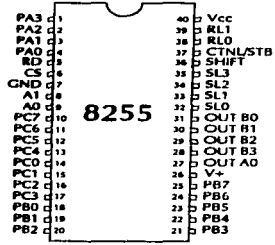


FIGURA 9-1 Terminales del 8255.

Este chip es encontrado en muchas aplicaciones. Tiene 24 terminales para entrada/salida, programable en grupos de 12, que son utilizados en tres modos de operación:

- El modo 0 Entrada/Salida Básica.
- El modo 1 Entrada/Salida Habilitada.
- El modo 2 Bus bidireccional.

Las 24 líneas se agrupan en dos grupos de 8 bits y dos grupos de 4. En modo 0, hay 16 maneras de definir las direcciones (entrada o salida) de los cuatro grupos de bits. Los datos sencillamente se envían o se reciben por ellas. En modo 1, los dos grupos de 4 bits sirven de control y estado, y los grupos de 8 bits son de datos. Cada grupo de 8 bits se puede definir como de entrada o salida. En este modo, el usuario mira un cierto bit de estado para saber si el dispositivo externo está preparado. Si lo está, el byte de datos se puede transmitir.

Tiene cuatro registros separados. Los primeros tres pueden ser programados individualmente, el cuarto puerto es el que lo hace flexible: escribiendo los diferentes valores en el puerto comando, se definen las funciones de los otros 3 puertos. Después las características de los puertos depende de la palabra de control cargada en el comando registrado el 8255 puede ser dinámicamente configurado por software.

El PPI es programable por software.

Antes de desarrollar las características de programación del PPI se describe a continuación el diseño de la circuitería para decodificar los puertos individuales que serán direccionados por el 8088.

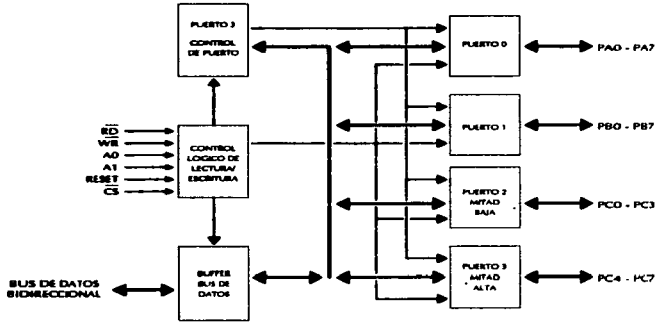


FIGURA 9-2

Ya que el 8088 puede direccionar más que 65,000 puertos, una parte básica de cualquier subsistema de E/S es la lógica que asegura que el 8088 esté conectado al puerto correcto cuando se efectúa una operación de E/S. El 8255 tiene 4 puertos, el primer paso efectuado en el diseño de la circuitería fue decidir el espacio que ocupa en el sistema. Estos son los primeros 4 puertos con los que cuenta el sistema adicionados al circuito y colocados en el mapa de memoria como puerto #0 hasta puerto #3. Los primeros 3 son puertos de E/S programables de propósito general del 8255 y el último, el puerto #3 es el registro de comandos del PPI.

9-5 DECODIFICACION DE E/S

Al igual que en la RAM y ROM debemos considerar el Chip Select del PPI además de las señales **IORD** y **IOWR**.

Un alto en la línea CS provoca que las líneas de datos estén en tercer estado.
Las condiciones que se deben de tomar en cuenta que se presentan cuando el 8088 quiere accesar a algún puerto del PPI.

1. La dirección del puerto es puesta en el bus de dirección.
2. La línea IO/M está en alto para indicar una operación E/S.
3. Cualquiera de las señales de lectura RD o escritura WR son activadas.

Se utilizó un decodificador 74LS259. Este chip es muy flexible ya que se utiliza como decodificador octal y además cuenta con un latch de 8 bits interno. Configurando propiamente el circuito no sólo se controla la entrada CS del PPI, pero escribiendo el software adecuado se mantiene seleccionado aún después de finalizada la operación de E/S.

Conectando la terminal de reset del 259 a la línea A7 del 8088 y la terminal que habilita IO/M a un inversor, el 259 trabajará como un decodificador octal y seleccionará una salida mediante la decodificación de las 3 líneas bajas del bus de dirección (A0 a A2). El 259 es alambrado para poner en bajo a todas las salidas no seleccionadas. Conectar la terminal D a V+, esto hace que la salida seleccionada vaya a alto y

distinguirla de las salidas no seleccionadas.

Ya que el 259 está decodificando las direcciones de E/S y selecciona las 8 salidas. El 8088 ocupa los primeros 4 puertos debido a esto se realiza un arreglo de señales para poder obtener la señal que habilita al PPI y esto se realiza combinando las señales a través de compuertas NOR y NAND como se muestra en la figura 9-3.

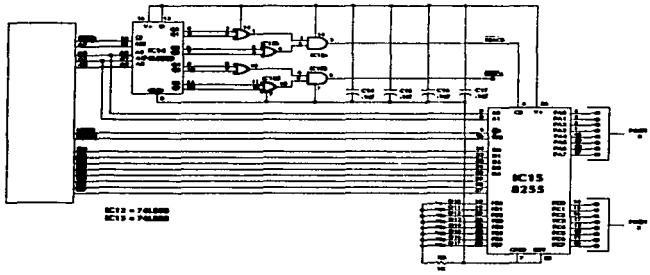


FIGURA 9-3 Diagrama eléctrico del circuito de E/S.

Utilizando el 74LS259 se deja abierta la posibilidad de poder conectar otro PPI en paralelo en el sistema y poder decodificar los segundos cuatro puertos.

CAPITULO 10
INTERFACE DISPLAY TECLADO

10.1 INTRODUCCION

En esta sección se describe las interfaces de teclado y display, que son dispositivos de entrada y salida respectivamente. Estas interfaces están a cargo principalmente por el CI 8279.

10.2 DESCRIPCION BASICA DEL 8279

El 8279 es una interface programable para teclado y display que rastrea y codifica puede realizar el barrido hasta para 64 teclas y hasta 16 displays. La sección de teclado de este circuito integrado tiene una construcción interna FIFO (primera entrada, primera salida), el buffer almacena 8 caracteres antes de que el microprocesador pueda captar un caracter. La sección del display puede buscar hasta 16 displays desde la memoria RAM interna de 16 X 8 que almacena el código para el display y teclado.

El 8279 fue diseñado para proporcionar una interface sencilla para los microprocesadores 8088/8086. La figura 10-1 muestra las terminales de este circuito integrado.

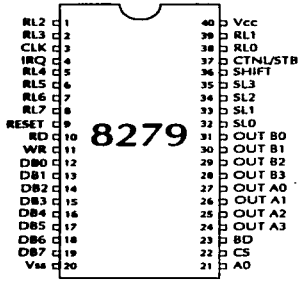


FIGURA10-1 Terminales del 8279.

DB7-DB0. Terminales bidireccionales que se conectan al bus de direcciones/datos del microprocesador.

CLK (reloj). Una terminal de entrada utilizada para recibir la señal de tiempo. Es normalmente conectada al PLCK del 8284.

RESET (Reinicialización). Es una entrada que se conecta al reset del microprocesador.

CS (Selección de integrado). Una entrada que habilita al 8279 para programar, leer el teclado y la información de status, el control de escritura y datos del display.

A0 (Entrada de dirección). A0 una terminal utilizada para seleccionar datos o control para lectura o escritura en el microprocesador y el 8279, un 0 lógico en la terminal selecciona para datos y un 1 lógico para control.

RD (lectura). Una terminal de entrada conectada directamente al RD del 8088 en modo mínimo.

WR (escritura). Terminal de entrada que es conectada directamente a la terminal WR del 8088.

IORQ (Interrupción REQUEST). Una salida que lleva un 1 lógico cuando los datos del teclado son disponibles para el microprocesador.

VSS tierra GND.

VCC alimentación +5v del sistema.

SL3 - SL0 (líneas de búsqueda). Salidas utilizadas para buscar el teclado y display.

RL7 RL0 (líneas de retorno). Entradas utilizadas para sensar alguna tecla presionada en la matriz del teclado.

Shift Terminal de entrada conectada a la tecla shift del teclado.

CNTL/STB (control/habilitación). Una entrada normalmente conectada a la

tecla de control en el teclado .

OUT A3-OUT A0. Terminales de salida que se utilizan para enviar datos más significativos al display ().

OUT B3-OUT B0. Salidas: Terminales utilizadas para enviar los datos menos significativos al display.

BD (Apagar). Una salida utilizada para limpiar el display.

Existen dos versiones de este chip. El 8279-5 utilizado es compatible con la versión de 5 MHz. del 8088, el 8279 no lo es. La señal PCLK es conectada a la entrada CLK del 8279-5, lo cual significa que la entrada CLK de este es de 2.5 MHz.

El 8279-5 es decodificado a la dirección de I/O de 1 byte de puerto 07H para datos y direcciones, 08H para control. A_0 es conectada en A_0 del 8088.

10.3 INICIALIZACION Y COMUNICACION CON UN 8279

El primer paso en la inicialización a un dispositivo programable es determinar el direccionamiento base del sistema, el direccionamiento interno y el direccionamiento del sistema para las partes internas. Como un ejemplo el que utilizamos en el circuito el direccionamiento base FFE8H. El 8279 tiene solo 2 direccionamientos internos los cuales son

seleccionados mediante el nivel lógico de A0 encontrándose en bajo cuando es seleccionado entonces el 8279 está habilitado para leer o escribir datos; estando en alto selecciona el registro control/status interno. Para este circuito, la entrada A0 es conectando a la línea de direccionamiento de datos para este es FFE8H y para el de control/status la dirección es FFEAH.

10.3.A INTERFACE DEL TECLADO

Se conectó un teclado de 24 teclas a través del 8279-5. La figura 10-2 muestra la interface del 8279-5 hacia el 8088 y la del teclado. El teclado forma una matriz de 6 X 4, cada punto de intersección representa un push button normalmente abierto que esta conectado a una línea horizontal y otra vertical cuando es presionado.

El decodificador 74LS138 genera 8 señales activos en bajo para el teclado. La selección de las terminales de salida SL_2 - SL_0 secuencialmente buscan en cada columna del teclado y el circuito interno del 8279-5 busca en las entradas RL comprobando en cada fila para detectar si se pulsó alguna tecla.

Antes de que alguna tecla sea detectada, el 8279-5 tiene que ser programado a través de 8 palabras de control. Los primeros 3 bits de el número de salida al control del puerto contiene el número de palabras de control. La programación del 8279 se desarrolla en la parte 3.

10.3.8 INTERFACE DEL DISPLAY

La figura 10-3 muestra la interface del 8279-5 con el 8088 y el display numérico de 8 dígitos. El software para este sistema mínimo utiliza el puerto 07H para datos y el 08H para control. En este circuito, los segmentos de datos son aplicados por las terminales de salida A y B. B0 es el bit más a la derecha de los datos de salida para el display, y A3 es el bit más hacia la izquierda. Esas salidas son enviadas a través de un manejador de segmentos 74LS244 para los segmentos del display.

Un 74LS138 decodificador de 3 a 8 líneas habilita los cátodos de los interruptores de cada posición del display para el 74LS138 desde el 8279-5. El display colocado en la izquierda del display es el número 0 (0000) y el colocado a la derecha del display es el número 7 (0111). Esas son las direcciones de cada display como está indicado en las palabras de control.

El valor de las resistencias conectadas a las salidas del 74LS244 son de 47 Ω.

Se utilizaron exhibidores de 7 segmentos con cátodo común, éstos son activados con una señal en bajo que es enviada a través de los colectores de los transistores 2N2222, la base de estros transistores (Q0-Q7) están conectados a las salidas del 74LS138 a través de un arreglo de resistencias de 150Ω como se muestra en la figura 10-3.

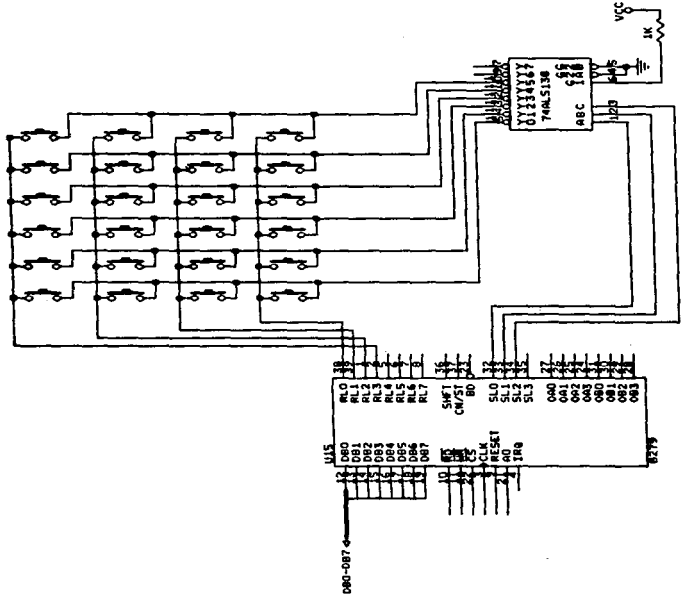

VCC
1K
74HLS138
A
B
C
G1
G2A
G2B
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
RL0
RL1
RL2
RL3
RL4
RL5
RL6
RL7
SHIFT
CNTL
BD
SL0
SL1
SL2
SL3
OA0
OA1
OA2
OA3
OB0
OB1
OB2
OB3
8279
D0-D7

FIGURA 10-2

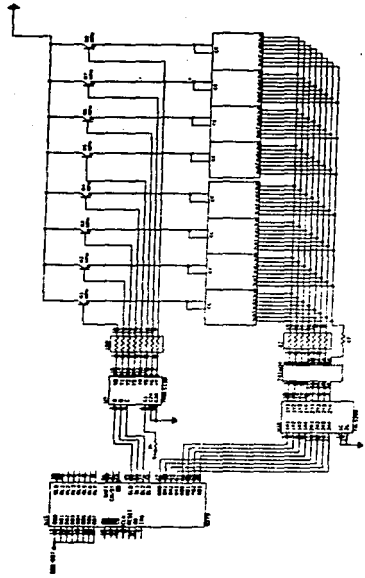
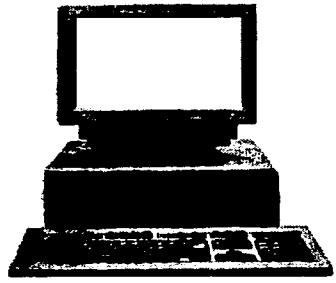


FIGURA 10-3

PARTE 2



DISEÑO DEL SOFTWARE

CAPITULO 11
ESPECIFICACION DEL SOFTWARE

11.1 INTRODUCCION

El software de microprocesadores puede definirse como el conjunto de actividades asociadas a la operación y al desarrollo de un sistema; es lo que guía o manda al computador, mientras que el hardware es el que realiza la operación en sí.

Es importante en el diseño de sistemas el conocimiento del hardware para poder interrelacionarlo con el software, ya que de esta manera se podrán aprovechar las ventajas que ofrece la integración de éstos.

En este capítulo se presentan las herramientas necesarias para el diseño del software. En primer lugar se clasifican los tipos de software, ésto con el objeto de comprender cuáles herramientas son las más viables.

Posteriormente se describen diferentes niveles de lenguaje de programación, así como las ventajas y desventajas de cada uno de éstos: lenguaje de máquina, lenguaje ensamblador y lenguaje de alto nivel.

En los siguientes capítulos se presenta la opción elegida para este trabajo, así como el procedimiento de aplicación en la elaboración de los programas de prueba y de sistema.

11.2 TIPOS DE SOFTWARE

En este punto se hace una clasificación general del tipo de software que corre en una computadora:

1) Software del sistema: conjunto de programas necesario para la creación, preparación y ejecución de otros programas.

2) Software de aplicación: programas del usuario con la finalidad de utilizar la computadora en la solución de sus problemas.

Los principales programas del software del sistema son:

1) Sistema Operativo. Es la interface entre el usuario de una computadora o sistema y el hardware de ésta. Sus funciones son:

- Ejecución de otros programas (de sistema o de usuario)
- Operaciones de entrada/salida
- Manejo de archivos
- Detección y manejo de errores durante la ejecución

2) Editores de texto. Sirve para meter, modificar o borrar texto en un archivo, que puede ser un programa fuente (escrito en lenguaje ensamblador o de alto nivel).

3) Ensambladores y Compiladores. Convierten las instrucciones de lenguaje ensamblador y de alto nivel respectivamente a lenguaje de máquina.

4) Encadenador. Junta varias partes de un programa compilado o ensamblado separado, y produce un programa ejecutable con direcciones relativas al inicio de la memoria.

5) Relocalizador. Resuelve las direcciones relativas de un programa generado por el encadenador.

11.3 TIPOS DE LENGUAJE

Un programa está formado por datos e instrucciones. Las instrucciones de cualquier computadora son de dos tipos. El primer tipo sirve para manejar datos:

- Leerlos o escribirlos en memoria
- Utilizarlos en operaciones aritméticas: suma, resta, multiplicación y división
- Utilizarlos en operaciones lógicas: and, or, xor, etc., y
- también en operaciones de corrimiento.

El segundo tipo de instrucciones controla el flujo de ejecución del programa.

- Instrucciones de salto (condicional e incondicional)
- Llamadas
- Retornos
- Interrupciones
- Instrucciones para control del sistema

Para correr un programa, un microprocesador debe tener las instrucciones almacenadas en forma binaria en localidades sucesivas de memoria, para ésto se usa los circuitos de almacenamiento vistos anteriormente.

Para escribir un programa para el microprocesador, se pueden usar tres niveles de lenguaje, los cuales presentamos a continuación:

11.2.1 LENGUAJE DE MAQUINA

Se pueden escribir programas como una simple secuencia de códigos binarios para las instrucciones que el microprocesador ejecutará.

La forma de representación en binario del programa a ejecutar se conoce como *lenguaje máquina*, porque ésta es la forma que necesita la máquina (en el medio de los programadores, al microprocesador se le llama máquina); sin embargo, es muy difícil, pero no imposible, para un programador memorizar los cientos de instrucciones en código binario para un microprocesador, como el 8088, también es muy fácil que al momento de trabajar con este tipo de código, exista un error al colocar un "0" por un "1" o viceversa, generalmente para ayudar a la representación de estos datos, se utiliza el código hexadecimal y así disminuyen los errores, pero de todos modos son cientos de códigos de instrucciones a cubrir.

El lenguaje de máquina consiste en códigos de instrucción (operaciones) y constantes o direcciones de memoria (operandos). Las instrucciones pueden tener diferentes formatos, dependiendo entre otras cosas, del número y tipo

de sus operandos. Alguien que programe en este nivel debe conocer y tener en cuenta hasta las características menos significativas para su aplicación como la dirección donde será ubicado el programa, los códigos de las instrucciones en todas sus variantes, etc. No cuidar este aspecto puede conducir a errores difíciles de detectar producidos por generar código impuro, es decir, rutinas que pueden automodificarse.

Programar en lenguaje de máquina tiene la ventaja de que no se requiere de algún programa adicional para el desarrollo de aplicaciones.

Sin embargo, tiene las siguientes desventajas:

- No posee mecanismos de abstracción de datos o programas.
- No existe una distinción clara entre datos y programas.
- Los programas no son legibles.
- Son difíciles de mantener (revisar, probar y depurar).
- Cualquier modificación tiene repercusiones en otras partes del desarrollo del programa.

En esta sección no se desarrollará el código de máquina para cada una de las instrucciones. En el apéndice C se muestra una tabla con el código de máquina para las diversas variantes de cada instrucción.

11.2.2 LENGUAJE ENSAMBLADOR

La dificultad para escribir programas en lenguaje de máquina es el motivo del desarrollo de lenguajes más fáciles de entender, aprender y usar por los programadores.

Durante la escritura de programas en lenguaje de máquina se reconoció que era posible realizar en forma automática muchas de las acciones involucradas en el desarrollo, como por ejemplo, obtener el código de una instrucción. Esto dio lugar a varios conceptos importantes: los lenguajes de programación y sus traductores.

El lenguaje de programación más simple consiste de nombres simbólicos para las instrucciones (nemónicos) y para los datos (constantes, direcciones de variables y rutinas). El traductor de este lenguaje, *llamado ensamblador,* convierte los nombres simbólicos en códigos de máquina y direcciones (de memoria) de manera que pueda ser cargado en memoria y correr. El texto que será traducido por el ensamblador se le llama programa fuente y al resultado del proceso se le llama programa objeto.

La aplicación requiere ahora de un programa adicional que no forma parte de ella, pero que facilita su desarrollo y reduce el tiempo de espera para su liberación; éste es el concepto de utilería.

Suponiendo una escala ascendente en la expresividad del lenguaje, que parte de la computadora y termina con el lenguaje natural, se dice que el ensamblador y el lenguaje de máquina son lenguajes de bajo nivel por la

reducida capacidad de abstracción que se puede manejar. Al igual que en el lenguaje de máquina los programas en lenguaje ensamblador dependen del procesador que se esté utilizando: no son transportables.

Las letras usadas para los nemónicos del ensamblador son usualmente iniciales o palabras abreviadas; en inglés, de la operación que se esté representando. Por ejemplo, para restar se utiliza el nemónico SUB y resta en inglés es subtract, el nemónico para la operación O-exclusiva es XOR y el nemónico para la operación que copia datos de una localidad a otra es MOV.

El lenguaje ensamblador tiene la desventaja de que requiere de programas adicionales para el desarrollo de una aplicación. En cambio, resuelve muchas de las desventajas de la programación en lenguaje de máquina, en mayor medida.

Una sentencia de lenguaje ensamblador consiste de campos. La mayoría de los ensambladores consisten de cuatro campos, como se indica a continuación, figura 11-1:

CAMPO DE ETIQUETA	CAMPO DE CODIGO DE OPERACION	CAMPO DE OPERANDO	CAMPO DE COMENTARIOS
SIGUIENTE:	ADD	AL,07H	; FACTOR DE CORRECCION

FIGURA 11-1

El primer campo en la declaración es llamado el campo de la etiqueta, donde una *etiqueta* es un símbolo o un grupo de símbolos usados para

representar una dirección que no se conoce en el momento de estar escribiendo el programa, las etiquetas van seguidas por dos puntos y además sólo se colocan cuando son necesarias.

El campo del código de operación contiene el nemónico de la instrucción que será ejecutada, así que los nemónicos son comúnmente llamados *códigos de operación*. El nemónico ADD en el ejemplo indica que la instrucción es la de suma.

El campo de operandos contiene los datos, la dirección de memoria, la dirección del puerto o el nombre del registro en el cual la operación se ejecuta, el *operando* es el nombre que se les da a los elementos que intervienen en la instrucción. En el ejemplo anterior, hay dos operandos, AL y 07H. AL representa a la parte baja del registro AX, es decir, al registro AL, y 07H representa el número hexadecimal 07. Por lo tanto, la declaración en ensamblador dice que le sume el número 07H al registro AL. Por convención de INTEL, el resultado de la suma es guardado en el registro que aparezca antes de la coma en el campo de operandos, por lo que siguiendo con este ejemplo, el resultado queda en el registro AL, por ejemplo, la declaración en ensamblador ADD BH, AL una vèz convertida a lenguaje máquina y corriendo, guarda el resultado de la suma del registro BH con el registro AL en el registro BH.

Mirando de nuevo el ejemplo anterior, se observa el campo de comentarios que inicia con un punto y coma, este campo es importante y no forma parte del código de máquina generado. Este campo permite escribir comentarios para recordar la acción de una instrucción o de un grupo de instrucciones.

La mayoría de los campos son opcionales, sin embargo, no puede aparecer el campo de operandos sin el de código de operación.

Además, existe un conjunto de nombres simbólicos, llamadas directivas, que instruyen al ensamblador en la forma de generación del código de máquina. (Ver apéndice C).

Para resumir la declaración en lenguaje ensamblador de la instrucción ADD para el 8088, se tiene que:

ADD destino, fuente

La fuente puede ser un número escrito en la instrucción, el contenido de un registro o el contenido de una localidad de memoria. El destino puede ser un registro o una localidad de memoria; sin embargo, en una instrucción, el fuente y el destino no pueden ser al mismo tiempo localidades de memoria.

Una vez terminado de escribir el código en ensamblador, hay que traducirlo a lenguaje de máquina, ésto se puede hacer de dos formas: la primera es traducir a código de máquina bit por bit cada línea del programa usando las plantillas o formatos proporcionados por el fabricante en las hojas de especificaciones del microprocesador (Apéndice C), esta tarea es tediosa y susceptible de errores; la segunda forma para traducir el código es usando el lenguaje *ensamblador*. Un ensamblador puede correr en una computadora personal o en un sistema de desarrollo. Este lee las instrucciones en lenguaje ensamblador y genera el código binario correcto para cada una de ellas.

A) PROGRAMACION EN ENSAMBLADOR

Como se ha mencionado un ensamblador es un programa de utilidad que traduce un programa escrito en lenguaje ensamblador al programa correspondiente en lenguaje de máquina.

Para precisar la terminología, se usa el nombre del programa objeto a la salida producida por el ensamblador. Un programa objeto no está en forma ejecutable ya que requiere de información del lugar donde se instala en memoria, así como, de determinar las posibles referencias de rutinas o variables externas.

Cuando un error de programación es detectado por el ensamblador, éste produce un mensaje explicativo del lugar y tipo de falla encontrado en el texto fuente.

B) CARACTERISTICAS FUNCIONALES DEL ENSAMBLADOR

Un ensamblador lee línea a línea el texto donde se encuentra el programa, casi siempre de un medio de almacenamiento secundario como el disco.

Este proceso de lectura -llamado paso- puede realizarse más de una vez dependiendo del tipo de ensamblador.

El programa objeto generado consiste básicamente de dos partes: información de las características del programa como extensión en bytes, punto de inicio, referencias a otros programas, etc. y, el código en lenguaje de máquina. El ensamblador almacena al programa objeto en disco casi siempre en forma de archivos binarios.

En adición al programa objeto, el ensamblador puede generar otro tipo de archivos: un reporte de la tabla de símbolos y un listado del resultado del ensamblaje. Este listado contiene el texto fuente posiblemente intercalado con el código de máquina de cada instrucción en caso de no haber encontrado errores. Si se presentaron errores se produce el texto fuente junto con las indicaciones correspondientes.

C) CONJUNTO DE INSTRUCCIONES

El conjunto de instrucciones del 8088 incluye las instrucciones equivalentes a cualquier microprocesador de 8 bits. Las instrucciones se dividen en seis categorías las cuales se ven a continuación:

1) Transferencia de datos
2) Aritmética entera binaria
3) Manipulación de bits
4) Instrucciones de manejo de cadenas

5) Transferencia de programa y
6) Control de proceso.

Para estas instrucciones se usan nemónicos, los cuales indican si los operandos son en octetos o palabras o si la fuente son datos inmediatos. Para mayor referencia de la lista de nemónicos correspondientes ver apéndice C.

1) TRANSFERENCIA DE DATOS

Las operaciones de transferencia de datos son las encargadas de mover datos de un sitio a otro de la computadora como: la memoria, los registros, el espacio de entrada/salida y la pila de ejecución.

El conjunto de instrucciones incluye 14 instrucciones de transferencia de datos que mueven bytes o palabras de datos entre la memoria y los registros, así como el acumulador y los puertos de entrada/salida.

2) ARITMETICAS

Las operaciones en aritmética binaria de punto fijo permiten al procesador realizar cálculos con los números enteros positivos o negativos en la representación complemento a dos.

El 8088 es capaz de sumar, restar, multiplicar y dividir datos, ya sean bytes o palabras, este sistema es capaz de sumar y restar bytes o palabras con signo

y sin signo, además de usar formato BCD o ASCII para los datos. Multiplica y divide operaciones de 8 ó 16 bits con signo y sin signo o números ASCII.

3) MANIPULACION DE BITS

Trece instrucciones proporcionan la capacidad de manipular bits en el microprocesador 8088. Esas instrucciones incluyen *operaciones lógicas, corrimientos y rotaciones.*

4) INSTRUCCIONES DE MANEJO DE CADENAS

Son usadas para manejar cadenas de datos en la memoria. Cada cadena está compuesta por bytes o palabras de hasta 64k bytes de longitud (65535 bits). Las instrucciones de cadena utilizan los registros SI y DI para direccionar y el registro CX para contar el número de bytes o palabras que se están utilizando. Las operaciones de cadenas una vez a no ser que aparezcan los prefijos REP, REPE/REPZ o REPNE/REPNZ, así que si una instrucción de cadena aparece con un prefijo, se repite un número de veces hasta que el registro CX es cero.

5) TRANSFERENCIA DE PROGRAMA

Las instrucciones de transferencia de programa incluyen saltos, llamadas y regresos de programas así como instrucciones de ciclos.

6) CONTROL DE PROCESO

Las instrucciones de control del proceso habilitan o deshabilitan interrupciones, modifican las banderas y sincronizan eventos externos, el 8088 incluye 12 instrucciones de control de proceso.

D) ESTRUCTURA BÁSICA DE UN PROGRAMA EN ENSAMBLADOR

El lenguaje ensamblador usado es un subconjunto para los procesadores Intel 8086/8088/80186/80188. Se debe tener en cuenta las siguientes consideraciones de diseño:

- Definir un lenguaje sencillo que conserve características en común con la mayoría de los ensambladores.
- Mostrar la forma de organización de memoria del procesador.
- El lenguaje y su ensamblador deben ser:

  - Fáciles de entender.
  - Fáciles de extender.
  - Fáciles de programar.
  - Susceptibles de optimización.

- El lenguaje y su ensamblador deben construirse a partir de un conjunto reducido de instrucciones y directivas.

- En lo posible, debe ser homogéneo y consistente en métodos y estructuras de datos con el desensamblador.

- Definir y usar símbolos externos para combinar varios módulos objeto.

- Definir un formato compatible de archivos objeto para que sean aceptados por el encadenador/cargador.

11.2.3 LENGUAJE DE ALTO NIVEL

Un lenguaje de alto nivel semeja un diálogo en lenguaje natural (por lo regular inglés) con ciertas reglas sintácticas que establecen la forma de como se van a construir las sentencias del programa.

El lenguaje de alto nivel aumenta el poder expresivo de sus construcciones que resulta en una mayor facilidad para el desarrollo de aplicaciones. Consecuencia de ésto es la independencia del lenguaje de la máquina donde se va a ejecutar, es decir, los programas son transportables.

A pesar de ésto, a medida que se aproxima a los mismos términos del lenguaje natural, el programa correspondiente en lenguaje de máquina es menos eficiente.

Algunos lenguajes de alto nivel como BASIC, FORTRAN, PASCAL o C usan declaraciones que se parecen más a frases en inglés que las que usa el lenguaje ensamblador, así que cada declaración en lenguaje de alto nivel puede representar muchas instrucciones en código máquina.

Para trasladar de un lenguaje de alto nivel a declaraciones en lenguaje de máquina se utiliza un *programa intérprete* o un *programa compilador*, según

el lenguaje utilizado. Los programas en lenguaje de alto nivel se escriben más rápidamente que en lenguaje ensamblador porque los lenguajes de alto nivel trabajan con bloques grandes preconstruidos, por lo que el código generado por el compilador o por el intérprete es más grande que el escrito en lenguaje ensamblador.

Aquí lo importante es la sencillez con la que un programa de aplicación puede ser desarrollado en lenguaje de alto nivel, es decir; de una manera más rápida y amigable, que además permita identificar y corregir los errores de programación más fácilmente, que usando lenguaje ensamblador.

Se encontró un programa que precisamente genera a partir de un archivo ejecutable, un archivo binario (.BIN) que puede ser cargado en una memoria ROM, es decir; genera un archivo "romeable".

Por todo lo antes expuesto, la mejor opción que se ha seleccionado para el desarrollo del software, dadas las ventajas que ofrece, es utilizar un lenguaje de alto nivel, el lenguaje de programación de más alto uso es el lenguaje C.

Como se ha mencionado el lenguaje C es un lenguaje de alto nivel de propósito general. Posee cualidades que lo hacen preferible a otros lenguajes, entre las cuales se encuentran:

- **Disponible.** Existen compiladores para una extensa variedad de computadoras.
- **Sencillo.** El número de elementos del lenguajes es relativamente reducido.

- Expresivo. Cuenta con un vasto conjunto de operadores.
- Uniforme. Es consistente en el uso de los elementos del lenguaje como el manejo de operadores.
- Eficiente. Los elementos del lenguaje son directamente representados en el "hardware".
- Estructurado. Las reglas para construir estructuras (de datos y de control) están bien definidas.
- Transportable. Está suficientemente bien estandarizado para que los programas corran sin cambio en cualquier máquina que soporte al lenguaje.
- Permite combinarse con programas escritos en ensamblador o en otros lenguajes.

Existen varias opciones a elegir entre compiladores de C existentes, en este caso se ha elegido el compilador "Turbo C 2.0", perteneciente a la marca Borland. Este compilador tiene las ventajas antes mencionadas, aunque el usuario puede utilizar cualquier otro compilador de C, sólo debe tomar en cuenta que lo que se requiere es la mayor simpleza posible en la programación.

Para diseñar los programas de prueba y el programa monitor (programa que maneja las entradas/salidas básicas del sistema mínimo) a través del lenguaje C se necesita un buen conocimiento de la estructura del mismo. Para empezar a trabajar con él, se deben hacer algunos ajustes que precisamente se tratan en el siguiente capítulo.

CAPITULO 12
CONVERSION DE PROGRAMA EN LENGUAJE C A CODIGO BINARIO

12.1 INTRODUCCION

Para la conversión de lenguaje C a código binario se cuenta con un programa de utilería, obtenido de la revista "The C Users Journal", el cual puede modificar un programa de MS-DOS.EXE a un archivo con formato binario.

Este programa llamado ROMLDR tiene el principal propósito de adaptar un programa en C para usarlo en memoria EPROM. Otros usos incluyen extensiones de BIOS, aplicaciones de DOS basadas en EPROM y relocalización de programas de MS-DOS en memoria de PC, hasta los 640 k de memoria del MS-DOS. Este programa puede incluso ser usado con archivos .EXE generados por otros lenguajes.

El ROMLDR puede ser compilado usando Turbo C o Microsoft C.

Para usar programas ejecutables en una EPROM se deben proporcionar varios componentes adicionales así como resolver algunas técnicas de programación. El número de componentes que se necesita, como el código de inicio, depende del compilador, el hardware y la aplicación.

Una vez que se han atendido todos esos detalles, se pueden compilar y encadenar varios programas en un archivo .EXE. Sin embargo, no se puede colocar directamente el archivo ejecutable en una EPROM y correr, antes se deben realizar una serie de pasos que el MS-DOS ejecuta implícitamente cuando carga un archivo ejecutable.

El MS-DOS modifica el programa cuando lo carga en memoria, esta modificación (también conocida como relocalización) consiste en la modificación de los valores de los segmentos del programa por la dirección actual del programa donde correrá. Al hacer la relocalización, el MS-DOS puede cargar un programa ejecutable en cualquier parte de la memoria (el MS-DOS puede cargar pequeños programas con extensión .COM donde sea y correrlos sin ninguna modificación).

El DOS realiza la relocalización sumando el valor del segmento de la dirección de carga a todas las direcciones de los segmentos almacenados en las áreas de códigos y datos del programa.

En este capítulo se muestra el procedimiento para generar de un archivo .EXE, un archivo en código binario.

En primer lugar se describe el programa ROMLDR, posteriormente se menciona que es lo que hace y al final se muestra el procedimiento que el usuario debe seguir para obtener de un programa en C, un archivo en código binario.

12.1 DESCRIPCION DEL PROGRAMA ROMLDR

El programa ROMLDR utiliza un archivo ejecutable (.EXE) y su archivo .MAP para crear un archivo en binario que puede ser colocado en una EPROM. El programa principal (main) inicia asignando memoria a un buffer que guardará el segmento del programa. Este buffer será de 10000H bytes de longitud para garantizar que cualquier tamaño de código sea procesado. Existe una rutina simple de error que se llama *term_error* que genera mensajes de error para una variedad de problemas potenciales y terminaciones del programa.

Después de asignar la memoria al buffer, el ROMLDR lee el archivo de configuración (.CFG) proporcionado en la línea de comandos. Este archivo contiene los nombres de los archivos .EXE, .MAP y .BIN, direcciones de carga para EPROM y RAM y el nombre de clase para el primer segmento en RAM.

12.2 ¿QUE HACE EL PROGRAMA ROMLDR?

A) LEYENDO UN ARCHIVO .MAP

El programa ROMLDR realiza un ciclo *while* para leer y procesar líneas de texto desde el archivo .MAP (el archivo .MAP debe ser la versión corta que sólo contenga la tabla de segmentos). El ROMLDR obtiene los nombres de clases de las localidades de memoria más su dirección de inicio y su dirección de final y las almacena en un arreglo de estructuras llamado

maptable, también realiza una verificación de la longitud usando la variable *class_loc* para determinar si la siguiente línea contiene la información de segmentos. El ROMLDR espera la línea que será colocada en formato de columna con el nombre de clase *class_loc*. El ROMLDR convierte los valores de las direcciones en enteros largos y compara los nombres de las clases con *ram_class*, que es una variable de configuración usada para definir el principio de la RAM. El principio de la dirección del segmento es almacenado en *ramdata*. El programa es capaz de procesar un máximo de 120 segmentos.

B) PROCESANDO LA CABECERA

Una vez que el ROMLDR ha leído el archivo .MAP, lee la cabecera del archivo EXE con la función *gethdr*. Esta función lee los primeros 32 bytes de la cabecera para determinar su tamaño y entonces lee el resto de la cabecera. Esta función almacena la información de la cabecera en un arreglo llamado *header* que puede contener un máximo de 10000 bytes.

El programa entonces ordena los apuntadores que se van a relocalizar en orden ascendente, la función *sort_table* utiliza la función *qsort* para ordenar los apuntadores. La función *cmp_ptr* es utilizada como argumento de *qsort*, compara los valores para *qsort* al convertir los apuntadores de la forma *segmento:offset* a enteros largos.

Después de ordenar los apuntadores, el programa ejecuta la relocalización de cada uno de los segmentos utilizando un ciclo *for* para hacer las iteraciones de todos los segmentos. La función *read_segm* lee cada segmento

desde el archivo EXE y regresa el tamaño del segmento. Si la longitud del segmento es diferente de cero, ROMLDR invoca a la función *fix_segm* para ejecutar la relocalización adecuada e invoca a la función *write_segm* para crear el archivo .BIN.

El programa ROMLDR se ha capturado de acuerdo al listado del apéndice E.

12.3 PROCEDIMIENTO PARA LA CONVERSION DE UN PROGRAMA EN LENGUAJE C A BINARIO

Los pasos que deben seguirse para la obtención del archivo .BIN, se citan a continuación:

1. **Edición del programa de aplicación en lenguaje C**
2. **Compilación del programa**
3. **Generación del archivo MAP**
4. **Creación del archivo de configuración**
5. **Ejecutar archivo ROMLDR.EXE dándole como parámetro el archivo .CFG, automáticamente se genera el archivo .BIN, el cual se puede grabar directamente a la EPROM.**

Descripción de cada punto :

1. **Edición del programa de aplicación en lenguaje C.**

Se refiere a la elaboración del programa diseñado por cada usuario, esta edición depende básicamente de los conocimientos del lenguaje.

2. Compilación del programa de aplicación.

Consiste en la creación del archivo ejecutable libre de errores.

3. Generación del archivo MAP.

Se refiere a la generación automática del archivo .MAP por parte del compilador. Para que esto sea posible, se le tiene que indicar al compilador por medio del menú **Options->Linker->Map file->Segments**. Una vez hecho ésto, cada vez que se genere un programa, automáticamente aparece el archivo .MAP correspondiente. (Ver figura 12-1).

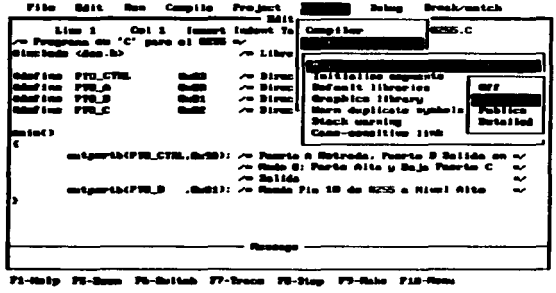


FIGURA 12-1

4. Creación del archivo de configuración.

Este archivo se creará en un editor de textos, debe contener el nombre del archivo .MAP, el nombre del archivo .EXE y el nombre del archivo .BIN, además de tener las direcciones ROM y RAM y el nombre del primer segmento de RAM, (para todos los programas a elaborar este archivo será el mismo).

5. Ejecución del archivo ROMLDR.

El paso final es la ejecución del archivo ROMLDR teniendo como parámetro en línea el nombre del archivo del configuración, con el cual se obtendrá el archivo .BIN.

En la figura 12-2 se tienen tres pasos:

a) El compilador genera el archivo .OBJ

b) El compilador genera el archivo .EXE y el archivo .MAP encadenado las librerías CS.LIB y C)S.OBJ

c) El programa ROMLDR genera el archivo .BIN de los archivos .EXE, .MAP y .CFG.

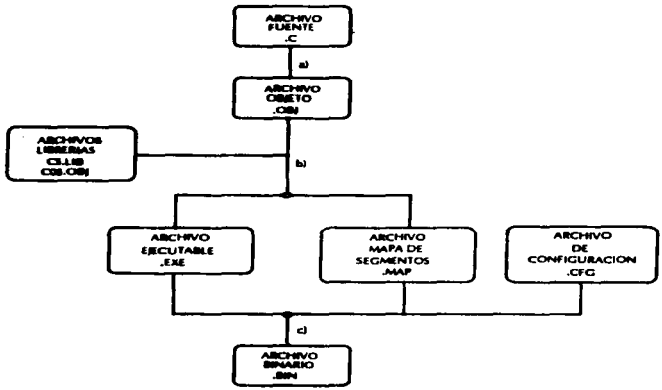


FIGURA 12-2

Diagrama a bloques del procedimiento para convertir un archivo fuente a código binario.

12.4.1 EJEMPLO DEL PROCEDIMIENTO ANTERIOR:

1. Se tiene el siguiente programa en C (8255.C):

```
/* Programa para programar el 8255 */
```

```
#include <dos.h>

main()
{
        outportb(0x03,0x90);
        outportb(0x01,0x01);
}
```

2. Al compilar este programa, se obtiene el archivo EXE (8255.EXE).

```
**********  **********

          8255.exe

**********  **********

Offset 0, hex 0
00000000: 4D 5A 20 01 06 00 00 00 - 20 00 00 00 FF FF 02 00  MZ.... ..... .....
00000010: 00 09 00 00 00 00 00 00 - 22 00 00 00 01 00 FB 20  ........"..... .....
00000030: 73 6A 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  sj.....  ..... .....
00000040: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000050: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000060: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000070: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000080: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000090: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000000A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000000B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000000C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000000D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000000E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000000F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000100: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000110: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000120: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000130: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000140: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000150: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000160: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000170: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000180: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000190: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000001A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000001B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000001C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000001D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000001E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
000001F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....

Offset 512, hex 200
00000200: B8 00 01 8E D8 8E C0 BB - 00 00 8E D8 BB 00 09 8B  ..... ..... .....
00000210: E0 E8 00 00 BB 90 E6 03 - BB 01 E6 01 C3 00 00 00  ..... ..... .....
00000220: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000230: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000240: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000250: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
00000260: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ..... ..... .....
```

```
00000270: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000280: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000290: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000002A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000002B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000002C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000002D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000002E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000002F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000300: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000310: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000320: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000330: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000340: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000350: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000360: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000370: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000380: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000390: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000003A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000003B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000003C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000003D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000003E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000003F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................

Offset 1,024, hex 400
00000400: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000410: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000420: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000430: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000440: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000450: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000460: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000470: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000480: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000490: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000004A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000004B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000004C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000004D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000004E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000004F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000500: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000510: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000520: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000530: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000540: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000550: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000560: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000570: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000580: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000590: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000005A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000005B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000005C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000005D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000005E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000005F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................

Offset 1,536, hex 600
00000600: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000610: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000620: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000630: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000640: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000650: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000660: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000670: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000680: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000690: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
```

```
00000AD0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000AE0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000AF0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B00: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B10: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B20: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B30: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B40: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B50: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B60: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B70: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B80: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B90: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000BA0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000BB0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000BC0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000BD0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000BE0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000BF0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C00: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C10: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C20: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C30: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C40: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C50: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C60: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C70: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C80: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000C90: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000CA0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000CB0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000CC0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000CD0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000CE0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000CF0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................

Offset 2,048, hex 800
00000800: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000810: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000820: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000830: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000840: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000850: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000860: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000870: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000880: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000890: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000008A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000008B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000008C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000008D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000008E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000008F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000900: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000910: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000920: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000930: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000940: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000950: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000960: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000970: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000980: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000990: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000009A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000009B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000009C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000009D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000009E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000009F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................

Offset 2,560, hex A00
00000A00: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A10: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A20: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A30: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A40: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A50: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A60: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A70: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A80: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000A90: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000AA0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000AB0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000AC0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
```

```
00000AD0: 00 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000AE0: 00 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000AF0: 00 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B00: 00 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B10: 00 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B20: FB 52 08 02 1E 00 00 00 - 05 00 18 00 00 00 01 00  .R..  ........
00000B30: 01 00 02 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B40: 20 0B 00 00 00 00 00 00 - 01 00 00 00 00 00 00 00  ................
00000B50: 01 00 00 00 14 00 00 00 - 00 03 00 00 00 00 00 00  ................
00000B60: 00 00 00 00 00 00 00 00 - 00 05 00 00 00 00 00 00  ................
00000B70: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000B80: 00 04 00 00 01 00 00 02 - 00 80 FF FF FF FF 00 00  ................
00000B90: 00 03 00 00 02 00 00 04 - 00 08 80 FF FF FF FF 00  ................
00000BA0: 00 06 00 00 04 00 00 06 - 00 00 00 00 80 FF FF FF  ................
00000BB0: 7F 08 00 00 01 00 00 08 - 00 00 00 00 00 FF 00 00  ................
00000BC0: 00 09 00 00 02 00 00 0A - 00 00 00 00 00 FF FF 00  ................
00000BD0: 00 0A 00 00 04 00 00 0C - 00 00 00 00 00 FF FF FF  ................
00000BE0: FF 0D 00 00 04 00 00 00 - 00 0F 00 00 08 00 00 00  ................
00000BF0: 00 10 00 00 0A 00 00 00 - 00 0E 00 00 08 00 00 00  ................

Offset 3,072, hex C00
00000C00: 00 28 00 00 01 00 00 00 - 00 0C 00 00 01 00 00 13  (...............
00000C10: 00 00 00 00 00 00 FF 00 00 - 00 07 00 00 08 00 00 00  ................
00000C20: 00 0B 00 00 00 08 00 00 00 - 00 2B 00 00 0A 00 00 00  ........+.......
00000C30: 00 00 00 00 00 00 00 00 00 - 00 5F 6D 61 69 6E 00 43  ........_main.C
00000C40: 30 53 2E 41 53 4D 00 43 - 30 53 00 38 32 35 35 2E  08.ASM C0S 8255.
00000C50: 43 00 38 32 35 35 00                               C.8255
```

3. Al obtener el archivo .EXE, se genera al mismo tiempo el archivo .MAP. Para este caso : 8255.MAP.

Start	Stop	Length	Name	Class
00000H	00020H	00021H	_TEXT	CODE
00022H	00022H	00000H	_DATA	DATA
00022H	00022H	00000H	_BSS	BSS
00030H	001AFH	00180H	_STACK	STACK

Program entry point at 0000:0000

4. Se edita ahora el archivo de configuración (8255.CFG).

```
8255.map 8255.exe 8255.bin
0x00 0x00
ram_class
```

5. Se corre el archivo ROMLDR.EXE

```
c:\> romldr 8255.cfg
```

Se obtiene el archivo .BIN:

```
••••••••••••••••••••••••••••
        8255.bin
••••••••••••••••••••••••••••

Offset 0, hex 0
00000000: B8 00 01 8E D8 8E C0 B8  - 00 08 8E D0 B8 00 09 8B  ................
00000010: E0 E8 00 00 B0 90 E6 03  - B0 01 E6 01 C3 00        ..............
```

CAPITULO 13
MODIFICACION DE LIBRERIAS DE TURBO C

13.1 INTRODUCCION

Como se ha estado mencionando, el programa ROMLDR puede transportar de un archivo ejecutable en formato DOS a un archivo en binario para poder ser leído por cualquier grabador de EPROM's, pero es el momento de hacer una aclaración importante: el programa ROMLDR puede transportar cualquier programa ejecutable hecho por cualquier compilador a un código binario. Tomemos el siguiente programa en C:

```
/* Programa en 'C' para el 8255 */
#include <dos.h>                    /* Librería de Turbo C Versión 2.0    */

#define  PTO_CTRL  0x03     /* Dirección del Registro de Control   */
#define  PTO_A     0x00     /* Dirección del Puerto A del 8255     */
#define  PTO_B     0x01     /* Dirección del Puerto B del 8255     */
#define  PTO_C     0x02     /* Dirección del Puerto C del 8255     */

main()
{
  outportb(PTO_CTRL,0x90); /* Puerto A Entrada, Puerto B Salida en   */
                           /* Modo 0; Parte Alta y Baja Puerto C     */
                           /* Salida.                                */
  outportb(PTO_B  ,0x01);  /* Manda Pin 18 de 8255 a Nivel Alto      */
}
```

Este programa manda un nivel alto a la terminal 18 del 8255 y si se compila directamente del ambiente integrado de Turbo C se obtiene el archivo .MAP siguiente:

```
 Start    Stop     Length     Name           Class

 0000DH 0054FH  0053F2H  _TEXT          CODE
 00550H 0059CH  0004DH CODEOO          CODE
 005A0H 0B6FFH  01A04H  _DATA          DATA
 007000H 007033H  00004H  _EMUSEG      DATA
 007040H 007053H  00002H  _CBTSEG      DATA
 007060H 007060H  00000H  _CVTSEG      DATA
```

```
0B7B6H 0B7B6H 0000BH  _SCNSEG     DATA
0B7B6H 0B7B6H 0000BH  DATOS       DATA
0B7B6H 0B7ABH 0004DH  _BSS        BSS
0B7ACH 0B7ACH 0000BH  _BSSEND     STACK
0B7B8H 0B7CFH 0000BH  _STACK      STACK

Program entry point at 0000:0000
```

Se observa que el ambiente integrado crea los segmentos que le son necesarios para correr bajo el sistema operativo, que en este caso es el DOS. El archivo ejecutable generado (.EXE) tiene un tamaño de 4,188 bytes y el archivo generado para la ROM (.BIN) tiene una longitud de 1,868 bytes.

Si el archivo .BIN se graba en una ROM y es colocado en una tarjeta madre de una PC compatible, el programa se ejecutará correctamente, pero si se coloca en el sistema mínimo, el programa no funcionaría porque éste, está diseñado para trabajar *bajo el sistema operativo DOS* y utilizar las rutinas del BIOS (Basic I/O System).

El tamaño del archivo ejecutable y el binario es grande, debido a que Turbo C inserta sus rutinas propias para el intercambio de recursos con el sistema operativo. Se muestra a continuación el archivo .BIN generado con las librerías de Turbo C:

```
0259.bin

Offset: 0, base: 0
00000000: BA 26 00 2B 00 16 F8 01 - B4 30 CD 21 8B 2E 02 00  .V.....0.!....
00000010: 00 1E 2C 00 8E DA A3 92 - 00 8C 06 90 00 89 1E 8C  ..,.............
00000020: 00 89 2E AC 00 C7 06 96 - 00 F7 FF E8 34 01 C4 3E  ..........4..>
00000030: 8A 00 00 C7 00 D8 00 FF - 77 26 81 3D 38 37 75 19  .........w&.=87u.
00000040: 26 8B 35 02 00 FA 3D 75 - 10 00 E6 DF F7 06 96 00  &.U...=u........
00000050: 00 F6 29 75 04 F7 06 96 - 00 F2 AE E3 01 43 26 38  ..)u.........C&8
00000060: 05 75 D6 00 CD 00 F7 D9 - 00 0E 8A 00 B0 01 00 D3  .u..............
00000070: E3 03 C3 00 E3 E3 F6 00 - 1E 8E 00 8C DA 2B EA 8B  .............+..
00000080: 3E 82 01 01 FF 00 02 73 - 07 BF 00 02 89 3E 82 01  >.....s.....>..
00000090: 81 C7 BC 01 72 28 03 3E - 9A 01 72 22 B1 04 D3 EF  ....r(.>..r"....
000000A0: 47 3B EF 72 19 03 3E 9C - 01 00 74 07 E3 3E 9A 01  G;.r..>...t..>..
000000B0: 00 73 00 BF 00 10 3B EF - 77 07 00 FD EB 03 E9 21  .s....;.w......!
000000C0: 01 8B EF 03 DA 00 1E A4 - 00 00 1E A3 00 A1 90 00  ................
```

```
00000DD0: 2B D8 8E C0 B4 4A 57 CD - 21 5F D3 E7 FA 8E D2 BB  +....JW.!_......
00000DE0: E7 FB 33 C0 2E 8E 06 F8 - 01 BF A6 01 B9 EC 01 2B  ..3............+
00000DF0: CF F3 AA 0E FF 16 A0 01 - E8 35 01 E8 1D 02 B4 00  .........5......
00000E00: CD 1A B9 16 00 B0 00 E6 - 9A 66 FF 16 A4 01 FF 36  .........f.....6
00000E10: 66 00 FF 36 B6 00 FF 36 - B4 00 E8 33 04 50 E8 DA  f..6...6...3.P..
00000E20: 00 2E 8E 1E F8 01 E8 7C - 00 0E FF 16 A2 01 33 C0  .......|......3.
00000E30: 89 D8 B9 2F 00 99 FC 02 - 04 80 D6 00 46 E2 FB 2D  .../........F..-
00000E40: 37 0D 00 74 0A B9 19 00 - 90 BA 2F 00 E8 B5 00 EB  7..t....../.....
00000E50: BC B4 4C 8A 46 02 CD 21 - B9 0E 00 90 BA 48 00 E9  ..L.F..!.....H..
00000E60: 07 00 1E B8 00 35 CD 21 - B9 1E 74 00 8C 06 76 00  .....5.!..t...v.
00000E70: B8 04 35 CD 21 B9 1E 78 - 00 8C 06 7A 00 B8 05 35  ..5.!..x...z...5
00000E80: CD 21 B9 1E 70 00 8C 06 - 7E 00 B8 06 35 CD 21 B9  .!..p...~...5.!.
00000E90: 1E 80 09 8C 06 82 00 B8 - 00 25 8C CA 8E DA BA 58  .........%.....X
00000EA0: 01 CD 21 1F C3 1E B8 00 - 25 C5 16 74 00 CD 21 1F  ..!.....%..t..!.
00000EB0: 1E B8 04 25 C5 16 78 00 - CD 21 1F 1E B8 05 25 C5  ...%..x..!....%.
00000EC0: 16 7C 00 CD 21 1F 1E B8 - 06 25 C5 16 80 00 CD 21  .|..!....%.....!
00000ED0: 1F C3 C7 06 96 00 00 00 - CB C3 B4 40 BB 02 00 CD  ...........@....
00000EE0: 21 C3 B8 1E 00 90 BA 56 - 00 2E 8E 1E F8 01 E8 E9  !......V........
00000EF0: FF B8 83 00 50 E8 29 FF - 00 00 C3 55 8B EC E8 0A  ....P.)....U....

Offset 512, hex 200
00000200: 8B 1E 96 01 D1 E3 FF 97 - A6 01 A1 9E 01 FF 06 9E  ................
00000210: 01 0B C0 75 EB FF 16 96 - 01 FF 16 96 01 FF 16 98  ...u............
00000220: 01 FF 76 04 E8 FA FE 59 - 5D C3 00 00 00 00 00 00  ..v....Y]......
00000230: 2E 8F 06 2A 02 2E 8C 1E - 2C 02 FC 8E 06 90 00 8E  ...*....,.......
00000240: 0E 00 32 B4 26 AC 40 8C - C3 F7 D6 93 EB 36 FA 00  ..2.&.@......6..
00000250: B3 C6 02 B9 01 00 8D 3E - 92 00 03 72 11 8E 06 8C  .......>...r....
00000260: 00 8B FE B1 7F 32 C0 F2 - AE E3 76 B0 F1 7F 83 EC  .....2....v...D.
00000270: 02 B8 01 0B 03 C3 03 C1 - 25 FE FF 0B FC 2B FE 72  ........%....+.r
00000280: 40 8B E7 8C C0 8E D8 8C - D8 8E C0 51 49 F3 A6 32  @..........QI...2
00000290: C0 AA 8E DD 07 F3 07 D9 - 8B C3 8B D0 43 E8 19 00  ............C...
000002A0: 77 07 72 40 E8 12 00 77 - F9 3C 20 74 08 3C 0D 74  w.r@...w.< t.<.t
000002B0: 04 3C 09 75 E8 32 C0 EB - E4 E8 C0 74 07 42 AA 0A  .<.u.2.....t.B..
000002C0: C0 75 01 43 86 E0 32 C0 - F9 E3 19 AC 40 2C 22 74  .u.C..2.....L"t
000002D0: 0F 04 22 3C 3C 75 07 80 - 3C 22 75 02 AC 49 0B F6  ..".<u..<"u..I..
000002E0: C3 E9 FE FE 99 03 CA 2E - 8E 1E 2C 02 B9 1E 84 00  ..Y.......,.....
000002F0: 43 03 D8 8B F4 8B 3C 2E - EB 72 E4 8B E5 99 2E 86  C.....<..r......
00000300: 00 E3 08 8F 76 00 E3 C3 - 02 36 AC 0A C0 E0 FA 74  ....v....6.....t
00000310: F0 33 C9 B9 46 00 2E F9 - 26 2A 02 8B 0E 8A 00 51  .3..F...&*.....Q
00000320: E8 47 01 59 8B F0 0B C0 - 74 24 1E 1E 07 8E 1E 8C  .G.Y....t$......
00000330: 00 33 FC F3 A6 1F EB - F8 06 FF 36 8E 00 E8 29  .3.........6...)
00000340: 01 E3 C4 02 8B D8 07 A3 - 88 00 0B C0 75 03 E9 91  ............u...
00000350: FE 33 C0 B9 FF FF B9 3F - 83 C3 02 F2 AE 26 3B 05  .3.....?.....&;.
00000360: 75 F4 B9 07 C3 33 C0 EC - 83 3E 9E 01 20 75 05 E8  u....3...>.. u..
00000370: 01 00 EB 19 B8 46 04 8B - 1E 96 01 D1 E3 B9 87 A6  .....F..........
00000380: 01 FF 06 9E 01 33 C0 EB - 00 5D C3 55 8B EC 56 57  .....3...]..U..VW
00000390: 8B 7E 04 8B 45 06 A3 E2 - 01 3B C7 73 08 C7 06 E2  .~..E....;.s....
000003A0: 01 00 00 EB 10 8B 75 04 - 8B 1E E2 01 B9 77 04 A1  ......u......w..
000003B0: E8 01 B9 04 06 3F 3E 5E 5D - C3 55 8B EC 56 57 8B 7E  .....?>^]..U..VW.~
000003C0: 04 8B 45 06 29 05 8B 35 - 83 FF 8B 46 06 40 89 04  ..E.).5....F.@..
000003D0: B9 7C 02 A1 E6 01 3B C7 - 75 08 B9 56 E6 01 EB 08  .|....;.u..V....
000003E0: 8B FE 83 7E 06 00 75 02 - EB C6 05 04 00 EB 00 5F  ...~..u........_
000003F0: 5E 5D C3 55 8B EC 56 8B - 46 04 33 D2 23 FF FF 81  ^]..U..V.F.3.#...

Offset 1,024, hex 400
00000400: E2 00 00 52 50 E8 F4 00 - 59 59 8B F0 83 FE FF 75  ...RP...YY.....u
00000410: 04 33 C0 EB 18 A1 E6 01 - B9 44 02 8B 46 04 40 89  .3.......D..F.@.
00000420: 04 89 36 E6 01 A1 E6 01 - 05 04 00 EB 00 5E 5D C3  ..6..........^].
00000430: 55 8B EC 56 8B 46 04 33 - D2 25 FF FF 81 E2 00 00  U..V.F.3.%......
00000440: 52 50 E8 B7 00 59 59 8B - F0 83 FE FF 75 04 33 C0  RP...YY.....u.3.
00000450: EB 15 B9 36 EA 01 B9 36 - E6 01 8B 46 04 40 89 04  ...6...6...F.@..
00000460: 8B C6 05 04 00 EB 00 5E - 5D C3 55 8B EC 56 57 8B  .......^]..U..VW.
00000470: 7E 04 8B FF 72 04 33 C0 - EB 9A 8B C7 05 0B 00 25  ~...r.3.....7...%
00000480: F0 FF 8B F0 83 3E EA 01 - 00 75 07 57 E8 A1 FF 59  .....>...u.W...Y
00000490: EB 43 8B 36 E8 01 8B C6 - 08 C0 76 31 8B 04 8B D7  .C.6......v1....
000004A0: 03 C2 25 3B C2 72 09 57 - 56 E8 0D FF 59 59 EB 24  ..%;.r.WV...YY.$
000004B0: 8B 04 3B C7 72 0E 56 E8 - D1 FE 59 FF 04 8B C6 05  ..;.r.V...Y.....
000004C0: 04 00 EB 10 8B 74 06 3B - 36 E8 01 73 CF 57 E8 22  .....t.;6..s.W."
000004D0: FF 59 EB 00 5F 5E 5D C3 - 55 8B EC 8B 46 04 8B D4  .Y.._^].U...F...
000004E0: 81 EA 00 01 3B C2 73 07 - A3 9E 00 33 C0 EB 0B C7  ....;.s....3....
000004F0: 06 9E 00 00 00 00 8B FF FF - EB 00 5D C3 55 8B EC 8B  ..........].U...
```

```
00000300: 46 04 8B 56 06 03 06 9E - 00 83 D2 00 8B C8 81 C1  F..V..........
00000310: 00 01 83 D2 00 8B D2 75 - 0A 3B CC 79 06 87 06 9E  .......u.;.y....
00000320: 00 EB 08 C7 06 94 00 00 - 00 8B FF FF EB 00 5D C3  ............].
00000330: 55 8B EC FF 76 04 E8 9F - FF 59 EB 00 5D C3 55 8B  U...v....Y..].U.
00000340: EC 8B 46 04 99 33 50 E8 - E2 FF 58 EB 00 5D C3 55  ..F..3P.....].U
00000350: 55 8B EC 8B 46 04 8B 00 - 00 8B 46 5D C3 00 00 00  U.....F]...
00000360: 00 00 00 00 54 75 72 62 - 6F 2D 43 20 2D 20 43 6F  ....Turbo-C - Co
00000370: 70 79 72 69 67 68 74 20 - 28 63 29 20 31 39 38 38  pyright (c) 1988
00000380: 20 42 6F 72 6C 61 6E 64 - 20 00 6E 74 6C 2E 00 4E   Borland Intl..N
00000390: 75 6C 6C 20 70 6F 69 6E - 74 65 72 20 61 73 73 69  ull pointer assi
000003A0: 67 6E 6D 65 6E 74 0D 0A - 44 69 76 69 64 65 20 65  gnment..Divide e
000003B0: 72 72 6F 72 0D 0A 61 62 - 6E 6F 72 6D 61 6C 20 70  rror..Abnormal p
000003C0: 72 6F 67 72 61 6D 20 74 - 65 72 6D 69 6E 61 74 69  rogram terminati
000003D0: 6F 6E 0D 0A 00 00 00 00 - 00 00 00 00 00 00 00 00  on..............
000003E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000003F0: 00 00 00 00 00 00 00 00 - 00 00 00 00 EC 01 EC 01  ................

Offset 1,536, hex 600
00000600: EC 01 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000610: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000620: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000630: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000640: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000650: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000660: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000670: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000680: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000690: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000006A0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000006B0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000006C0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000006D0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000006E0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
000006F0: 00 00 00 00 FA 01 FA 01 - FA 01 00 00 00 00 10 00  ................
00000700: D2 01 D2 01 D0 01 00 00 - 00 00 00 00 00 00 00 00  ................
00000710: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000720: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000730: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00  ................
00000740: 00 00 00 00 00 00 00 00 - 00 00 00 00              ............
```

La parte subrayada muestra el código que realmente se necesita, todo lo que está de más se genera en el momento en que el compilador encadena el archivo objeto con las librerías de Turbo C para crear el archivo ejecutable. Como se ha mencionado, todo el código que aquí se muestra, trabajaría en una ROM colocada en una tarjeta madre de una PC compatible, por lo que el código que aparece, es el necesario para correr aplicaciones dentro de una PC utilizando las rutinas del DOS y del BIOS.

Esto representa un problema porque se tiene que modificar de alguna forma el código generado por el compilador, para poder crear aplicaciones que

funcionen en un sistema mínimo que utilice como microprocesador un 8088 y ésto es el tema de la siguiente sección.

13.2 CREACION DE LIBRERIAS COMPATIBLES CON UN SISTEMA MINIMO

Cuando se instala la versión completa del Turbo C, se genera un árbol de directorios como el que se muestra:

C:\	Directorio Raíz
TC	Directorio de Turbo C
LIB	Librerías Objeto de Turbo C
INCLUDE	Archivos Definición de Funciones Turbo
SYS	Archivos Definición de Funciones Turbo
BGI	Librerías Manejo de Gráficos
EXA	Ejemplos

En el directorio TC se encuentran los programas ejecutables de Turbo C; como son el compilador TC.EXE, el ensamblador TASM.EXE, el generador de librerías TLIB.EXE, etc.

El directorio LIB contiene las librerías que proporciona Borland como utilerías del sistema, las cuales son funciones para mandar caracteres a pantalla, funciones para poder leer de teclado, funciones para poder mandar a la impresora, etc.

El directorio INCLUDE y SYS tiene la declaración de las funciones que se encuentran dentro de las librerías de Turbo C.

El directorio BGI tiene los manejadores de video para trabajar en modo gráfico y el directorio EXA tiene algunos ejemplos que el fabricante proporciona, además aquí se encuentra un archivo que es muy importante para efectuar la modificación del código generado por Turbo C, que es el archivo C0.ASM, posteriormente se hablará de este archivo.

Para comprender cómo modificar las librerías de Turbo C, es necesario conocer en forma básica el modo de operación del compilador. A continuación se describe de forma breve la manera de trabajar de Turbo C:

En el ambiente de Turbo C, es posible leer archivos que hayan sido creados por un editor de textos con formato ASCII estándar, así que los archivos fuente que se generen con el editor de Turbo C pueden ser leídos por cualquier editor de textos que acepte el código ASCII.

Se les llama archivos fuente a los archivos que tienen extensión .C y que tiene el código en lenguaje C. Cuando se tiene el código fuente completo, el siguiente paso es compilar el programa para crear el archivo ejecutable con la secuencia Alt-C-B (figura 13-1).

Lo primero que hace el compilador es verificar que el código no tenga errores de sintaxis, es decir, que no se haya omitido algún paréntesis o alguna coma y que el nombre de las funciones utilizadas sean las correctas.

Cuando se comprueba que el código es correcto, se insertan automáticamente las definiciones de funciones del directorio INCLUDE o SYS, según sea el caso y se crea el archivo objeto .OBJ, que es un paso intermedio entre el archivo fuente y el archivo ejecutable; este archivo tiene el código de nuestro programa en binario, pero no es posible utilizarlo a este nivel, porque el programa ROMLDR lee archivos ejecutables (.EXE) y además de que no están colocadas aún las direcciones correctas de los segmentos de código y pila.

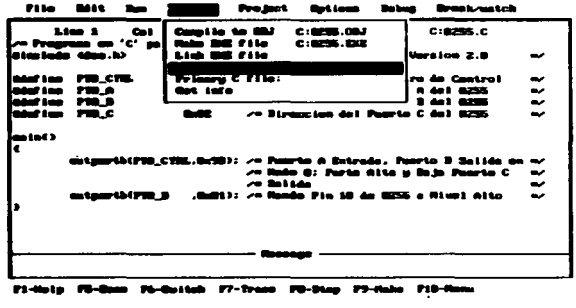


FIGURA 13-1

La última fase es encadenar el archivo .OBJ con las librerías de Turbo C, primeramente se encadena el archivo .OBJ con los archivos C0S.OBJ y

CS.LIB, ambos archivos se encuentran dentro del directorio LIB, finalmente se obtiene el archivo ejecutable.

El directorio LIB tiene varios archivos OBJ así como varios archivos LIB, según el modelo de memoria que se utiliza al momento de compilar, se encadena con el tipo de archivo OBJ y LIB que le corresponda. En este caso se emplea el modelo de memoria Small (modelo pequeño de compilación de Turbo C) por lo que los archivos que se encadenarán serán C0S.OBJ y CS.LIB.

El archivo C0S.OBJ tiene el código de inicio del programa de aplicación y el archivo CS.LIB tiene las librerías de Turbo C que se invocan desde el programa fuente; así que lo que se necesita modificar son los archivos C0S.OBJ y CS.LIB.

13.2.1 MODIFICACION DEL ARCHIVO C0S.OBJ

Este archivo tiene el código de inicio de cualquier programa que se compila con cualquier modelo de memoria, así como la definición de procedimientos para trabajar bajo el sistema operativo. Dentro del directorio EXA se encuentra el archivo C0.ASM que es el archivo base para crear cualquier otro archivo C0S.OBJ, está escrito en ensamblador y el código se puede ver en el apéndice D.

Del listado del apéndice D se puede observar la cantidad de código que Turbo C utiliza para el correcto funcionamiento de los programas de

aplicación bajo el ambiente de DOS, pero para crear una aplicación que trabaje en un sistema mínimo, no es necesario tanta definición de procedimientos ni variables, por lo que el código que se necesita para crear la aplicación para un sistema mínimo se muestra a continuación:

```
;[]------------------------------------------------------------[]
;  CSB.ASM -- CODIGO DE INICIALIZACION DEL SISTEMA MINIMO      |
;        PARA LA CREACION DE PROGRAMAS DE APLICACION           |
;        EN LENGUAJE 'C' UTILIZANDO TURBO C 2.0                |
;                                                              |
;     UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO                  |
;     FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN               |
;                                                              |
;  MARIA DE LA CRUZ GARCIA CONTRERAS      LUCIA GARCIA SOTO    |
;[]------------------------------------------------------------[]

; Definición de constantes
MINPILA equ   128             ;Define Tamaño Segmento Pila

; Definición de variables externas
  extrn  _main:near           ;Nombre de la función principal en C

_TEXT  SEGMENT BYTE PUBLIC 'CODE'     ;SEGMENTO DE CODIGO
  ASSUME  CS:_TEXT, DS:_DATA, SS:_STACK
CSB   PROC  FAR                       ;Inicio de procedimiento CSB
  mov   ax, 0100h                     ;Coloca la dirección deseada
  mov   ds, ax                        ;a los segmentos DS y ES
  mov   es, ax                        ;
  mov   ax, offset _STACK:_fondo_pila ;Coloca el offset del fondo
  mov   ss, ax                        ;de la pila al segmento SS
  mov   ax, offset _STACK:_tope_pila  ;Coloca el offset del tope
  mov   sp, ax                        ;de la pila al registro SP
  call  _main                         ;Invoca al programa en "C"
CSB   ENDP                            ;Fin de procedimiento CSB
_TEXT  ENDS                           ;Fin de segmento _TEXT

_DATA  SEGMENT WORD PUBLIC 'DATA'     ;SEGMENTO DE DATOS
_DATA  ENDS                           ;Fin de segmento _DATA

_BSS   SEGMENT WORD PUBLIC 'BSS'      ;SEGMENTO _BSS DE TURBO C 2.0
_BSS   ENDS                           ;Fin de segmento _BSS

_STACK  SEGMENT STACK 'STACK'         ;SEGMENTO DE PILA
  org   0000h                         ;Dirección de la pila
_fondo_pila   label  byte             ;Fondo de la pila
  db    MINPILA*2 dup (0)             ;Crea el espacio de la pila
_tope_pila    label  byte             ;Tope de la pila
_STACK  ENDS                          ;Fin de segmento _STACK

  END   CSB                           ;Fin de Programa
```

Se observa que se coloca un valor de 100H en el segmento de datos y en el segmento extra (DS y ES), este valor puede ser cualquier valor que se tenga

en la memoria RAM del sistema mínimo, también al segmento de pila y al apuntador de pila (SS y SP) se les coloca un valor que va de acuerdo a la dirección que se le asigne dentro de la RAM del sistema mínimo y por último se invoca al procedimiento principal del código en C llamado **main**.

Este programa en lenguaje ensamblador se compila usando el TASM.EXE como se muestra a continuación:

```
C:\> tasm /ml/la c0s.asm
```

Las opción /ml le dice al TASM que distinga las letras mayúsculas de las minúsculas porque el lenguaje C es un lenguaje que hace distinción de letras mayúsculas y minúsculas, ésto es con la finalidad de que cuando se encadene el archivo .OBJ con el C0S.OBJ pueda reconocer a la función principal de C (main); la opción /la genera un listado que nos servirá para comprobar la generación del código correcto.

Una vez hecho ésto, se genera el archivo C0S.OBJ y el archivo C0S.LST, hasta este punto ya se tiene el archivo C0S.OBJ modificado y sólo nos falta obtener el archivo CS.LIB.

13.2.2 MODIFICACION DEL ARCHIVO CS.LIB

El archivo CS.LIB original tiene un tamaño de 104,894 bytes porque tiene todas las funciones de utilería que ofrece Borland para trabajar bajo DOS. Todas estas funciones no son necesarias para el sistema mínimo, por lo que el archivo original CS.LIB se respalda para crear un nuevo archivo CS.LIB.

Para la creación del nuevo archivo se necesita tener el archivo C0S.OBJ con la siguiente línea de comando:

```
C:\> tlib cs.lib +c0s.obj
```

El programa TLIB.EXE crea una nueva librería de nombre CS.LIB y le agrega el archivo objeto C0S.OBJ, quedando el nuevo archivo CS.LIB con un tamaño de 1024 bytes.

De esta forma, se han modificado las librerías de Turbo C para poder crear aplicaciones en el sistema mínimo.

EJEMPLO:

Se toma el mismo programa fuente del 8255.C

```
/* Programa en 'C' para el 8255 */
#include <dos.h>                /* Libreria de Turbo C Version 2.0    */

#define  PTO_CTRL  0x03    /* Direccion del Registro de Control    */
#define  PTO_A     0x00    /* Direccion del Puerto A del 8255      */
#define  PTO_B     0x01    /* Direccion del Puerto B del 8255      */
#define  PTO_C     0x02    /* Direccion del Puerto C del 8255      */

main()
{
  outportb(PTO_CTRL,0x90); /* Puerto A Entrada, Puerto B Salida en  */
                           /* Modo 0; Parte Alta y Baja Puerto C    */
                           /* Salida                                */
  outportb(PTO_B  ,0x01);  /* Manda Por 1B de 8255 a Nivel Alto     */
}
```

Desde el ambiente integrado se compila este programa y se crea un archivo ejecutable, se corre el programa ROMLDR y se obtiene un archivo .BIN mucho menor que al creado con las librerías de Turbo C, el cual se muestra a continuación:

```
**************************
        8255.bin
**************************

Offset: 0, hex: 0
00000000: B8 00 01 8E D8 8E C0 E8 - 00 08 8E D0 BC 00 09 8B  ................
00000010: E0 E8 00 00 [redacted] - [redacted] C3 00  ................
```

Este código es el que se necesita y se puede cargar a una ROM.

En comparación con el archivo generado con las librerías de Turbo C, este ejecutable tiene un tamaño de 3,159 bytes y el archivo .BIN un tamaño de 30 bytes.

Para comprobar que es el código correcto y para ver que significa los valores hexadecimales antes y después del código se hace el siguiente procedimiento:

Se genera ahora el código en ensamblador de el programa 8255.C con la línea siguiente:

```
C:\> tcc -a 8255.c
```

Esta línea de comando crea el archivo en ensamblador del archivo fuente 8255.c que es el archivo 8255.ASM; se genera ahora un listado completo del archivo 8255.ASM con la línea siguiente:

```
C:\> tasm /la 8255.asm
```

Se obtiene un listado que nos facilita la identificación del código creado por el compilador con el código del archivo .BIN.:

```
8255.ASM
   1                                    ifndef ??version
   2                                ?debug macro
   3                                    endm
   4                                    endif
   5                                    ?debug S "8255.c"
   6        0000                    _TEXT   segment byte public 'CODE'
   7                                DGROUP  group   _DATA,_BSS
   8                                        assume cs:_TEXT,ds:DGROUP,ss:DGROUP
   9        0000                    _TEXT   ends
  10        0000                    _DATA   segment word public 'DATA'
  11        0000                    d@      label   byte
  12        0000                    d@w     label   word
  13        0000                    _DATA   ends
  14        0000                    _BSS    segment word public 'BSS'
  15        0000                    b@      label   byte
  16        0000                    b@w     label   word
  17                                        ?debug C E95C70C31E04383233552E63
  18                                        ?debug C
E900101D1113433A3C56435C4944E434C5544455C646F732E60
  19        0000                    _BSS    ends
  20        0000                    _TEXT   segment byte public 'CODE'
  21                                ;       ?debug L 9
  22        0000                    _main   proc    near
  23                                ;       ?debug L 11
  24        0000 B0 90                        mov     al,144
  25        0002 E6 03                        out     3,al
  26                                ;       ?debug L 14
  27        0004 B0 01                        mov     al,1
  28        0006 E6 01                        out     1,al
  29        0008                    @1:
  30                                ;       ?debug L 15
  31        0008 C3                           ret
  32        0009                    _main   endp
  33        0009                    _TEXT   ends
  34                                        ?debug C E9
  35        0000                    _DATA   segment word public 'DATA'
  36        0000                    d@      label   byte
  37        0000                    _DATA   ends
  38        0009                    _TEXT   segment byte public 'CODE'
  39        0009                    _TEXT   ends
  40                                        public  _main
  41                                        end

Symbol Table
Symbol Name                             Type    Value
??DATE                                  Text    "06/03/95"
??FILENAME                              Text    "8255   "
??TIME                                  Text    "14:03:15"
??VERSION                       Number 0205
@1                                      Near    _TEXT:0008
@CPU                                    Text    0101H
@CURSEG                                 Text    _TEXT
@FILENAME                               Text    8255
@WORDSIZE                               Text    2
B@                                      Byte    _BSS:0000
B@W                                     Word    _BSS:0000
D@                                      Byte    _DATA:0000
D@W                                     Word    _DATA:0000
S@                                      Byte    _DATA:0000
_MAIN                                   Near    _TEXT:0000

Groups & Segments               Bit Size Align  Combine Class
```

```
DGROUP                          Group
  _BSS                          16  0000 Word   Public  BSS
  _DATA                         16  0000 Word   Public  DATA
_TEXT                           16  0009 Byte   Public  CODE
```

En las líneas 24, 25, 27 y 28 está el código que se encuentra subrayado en el archivo .BIN y en la línea 31 se observa que el código siguiente es un regreso de subrutina.

Para determinar que significado tiene el código que está al principio del archivo, se muestra en seguida el archivo C0S.LST que se obtiene cuando se ejecuta la línea:

```
C:\> tasm /ml/la c0s.asm
```

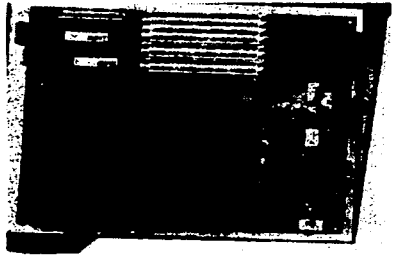
Se muestra el archivo C0S.LST

```
c0s.asm
  1
  2                             ;[]------------------------------------------------------[]
  3                             ;  C0S.ASM -- CODIGO DE INICIALIZACION DEL SISTEMA MINIMO  |
  4                             ;       PARA LA CREACION DE PROGRAMAS DE APLICACION       |
  5                             ;       EN     LENGUAJE 'C' UTILIZANDO   TURBO C 2.0      |
  6                             ;                                                         |
  7                             ;    UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO              |
  8                             ;      FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN         |
  9                             ;                                                         |
 10                             ;MARIA DE LA CRUZ GARCIA CONTRERAS      LUCIA GARCIA SOTO |
 11                             ;[]------------------------------------------------------[]
 12
 13                             ; Definicion de constantes
 14 =            0080           MINPILA equ   128                   ;Define Tamaño Segmento Pila
 15
 16                             ; Definicion de variables externas
 17                                     extrn  _main:near           ;Nombre dela funcion principal en C
 18
 19 0000                        _TEXT   SEGMENT BYTE PUBLIC 'CODE'   ;SEGMENTO DE CODIGO
 20                                     ASSUME CS:_TEXT, DS:_DATA, SS:_STACK  ;
 21 0000                        C0S     PROC   FAR                   ;Inicio de procedimiento C0S
 22 0000 8E C0 0100                     mov    ax, 0100h            ;Coloca la direccion deseada
 23 0003 8E D8                          mov    ds, ax               ;a los segmentos DS y ES
 24 0005 8E C0                          mov    es, ax               ;
 25 0007 B8 0000r                       mov    ax, offset   _STACK:_fondo_pila  ;Coloca el offset del fondo
 26 000A 8E D0                          mov    ss, ax               ;de la pila al segmento SS
 27 000C B8 0000r                       mov    ax, offset   _STACK:_tope_pila   ;Coloca el offset del tope
 28 000F 8B E0                          mov    sp, ax               ;de la pila al registro SP
 29 0011 E8 0000e                       call   _main                ;Invoca al programa en "C"
 30 0014                        C0S     ENDP                        ;Fin de procedimiento C0S
 31 0014                        _TEXT   ENDS                        ;Fin de segmento _TEXT
```

```
32
33    0000                    _DATA  SEGMENT WORD PUBLIC 'DATA'      ;SEGMENTO DE DATOS
34    0000                    _DATA  ENDS                             ;Fin de segmento _DATA
35
36
37    0000                    _BSS   SEGMENT WORD PUBLIC 'BSS'   ;SEGMENTO _BSS TURBO C 2.0
38    0000                    _BSS   ENDS                             ;Fin de segmento _BSS
39
40
41    0000                    _STACK SEGMENT STACK 'STACK'       ;SEGMENTO DE PILA
42                                    org   0800h                       ;Direccion de la pila
43    0800                    _fondo_pila   label  byte                 ;Fondo de la pila
44    0800 0100*(00)                  db          MINPILA*2 dup (0)     ;Crea el espacio de la pila
45    0900                    _tope_pila    label  byte                 ;Tope de la pila
46    0900                    _STACK ENDS                               ;Fin de segmento _STACK
47
48                                    END   COS                         ;Fin de Programa

Symbol Table
Symbol Name                     Type    Value

??DATE                          Text    "06/03/95"
??FILENAME                      Text    "cls"
??TIME                          Text    "14:05:18"
??VERSION               Number 0205
@CPU                            Text    0101H
@CURRSEG                        Text    _STACK
@FILENAME                       Text    COS
@WORDSIZE                       Text    2
COS                             Far     _TEXT:0000
MINPILA                         Number 0080
_FONDO_PILA                     Byte    _STACK:0800
_MAIN                           Near    ----:---- Extern
_TOPE_PILA                      Byte    _STACK:0900

Groups & Segments               Bit Size Align  Combine Class

_BSS                            16  0000 Word   Public  BSS
_DATA                           16  0000 Word   Public  DATA
_STACK                          16  0900 Para   Stack   STACK
_TEXT                           16  0014 Byte   Public  CODE
```

De la línea 21 a la línea 27 se observa el código generado para cargar con los valores correctos a los segmentos de datos, pila y extra, así como al apuntador de pila (DS, SP, ES y SP respectivamente). La línea 28 muestra el llamado de la función **"main"** del programa fuente en C, de esta forma, el código que se genera corresponde al código que se encuentra en el archivo .BIN y por lo tanto, es posible crear código "romeable" desde el compilador de Turbo C.

PARTE 3



INTEGRACION DE HARDWARE Y SOFTWARE

CAPITULO 14
INTEGRACION HARDWARE - SOFTWARE

14.1 INTRODUCCION

En este capítulo se menciona como primer punto la sincronización del sistema. Como ya se ha mencionado, la operación del sistema está basada en señales de reloj que sincronizan la operación de todo el sistema.

Como segundo punto se muestran los programas de prueba y las señales más importantes que participan en la operación del sistema en sus diferentes etapas.

14.2 TEMPORIZACION EN GENERAL

Antes de programar y arrancar el sistema es necesario entender la temporización del sistema, ya que la operación del sistema se rige por la señal de reloj que sincroniza a todo el sistema para la correcta ejecución de los programas.

14.2A Temporización de los buses

La señal de reloj es el latido del sistema y el encargado de dar la pauta en la sincronización de las funciones.

El reloj divide el ciclo base en varios estados T. En el primer estado T1, los 20 bits del bus de direcciones A0 - A19 están presentes, validados según la señal ALE. En el segundo estado T2, el bus pone el control de ciclo de máquina sobre los bits A16 y A19, que se convierten en S3 - S6. Las direcciones son reemplazadas sobre A0 - A15. En el tercer estado T3, los datos los datos muestreados en lectura o validados en escritura, en el estado T4 permite al bus prepararse para el siguiente ciclo de máquina (ver figura 14-1).

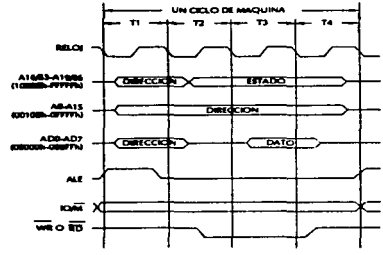


DIAGRAMA DE TIEMPO DE DIRECCIONES Y ALE.

FIGURA 14-1

Si la memoria seleccionada no ha sido lo suficientemente rápida como para responder al final de T3, la señal READY debe ser invalidada al principio de T3, lo que obliga al procesador a insertar estados T3 denominados TWait (que son idénticos a T3). Si la señal READY es válida durante un TW el procesador terminar el ciclo por un estado T4. Esta descripción de los estados es especialmente simple y repetitiva. El Microprocesador intercambia datos con la memoria y los periféricos, independientemente de la ejecución de las instrucciones. Esto es una consecuencia de la arquitectura interna del procesador: las unidades de control del bus y las de ejecución están separadas.

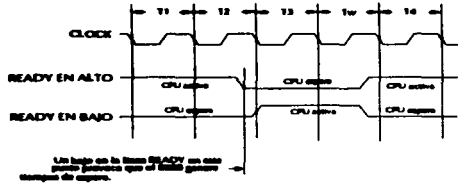


FIGURA 14-2

14.2B Temporización básica de los buses

Los tres canales -dirección, datos y control funcionan como sigue: si los datos son escritos a memoria ver figura 14-3, las salidas del

microprocesador del direccionamiento de memoria en el bus de direcciones, saca los datos para ser escritos dentro de la memoria en el bus de datos y edita una escritura (WR) para la memoria y IO/M es igual a 0. Si los datos son leídos desde memoria (ver figura 14-4) las salidas del microprocesador del direccionamiento en el bus de direcciones, edita una señal de memoria (RD) y acepta datos vía bus de datos.

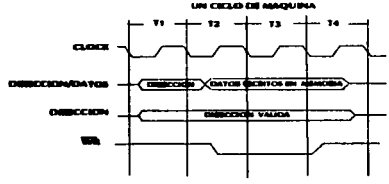


FIGURA 14-3

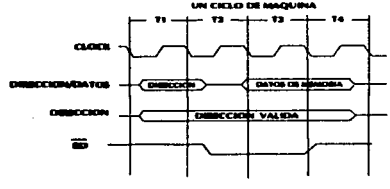


FIGURA 14-4

El 8088 utiliza la memoria e E/S en períodos de tiempo llamados ciclos de bus, el cual es igual a 4 períodos del sistema de reloj (estados T). El reloj del sistema opera a 4.77 MHz (frecuencia con el cual opera el sistema) entonces un ciclo de reloj es completado en 763.2 nseg. Esto significa que el microprocesador lee o escribe datos, memoria e E/S en un rango de 1.3 X106 de veces por segundo.

T1 Durante el primer periodo de reloj en un ciclo de bus el cual es llamado T1, suceden varias cosas: El direccionamiento de las localidades de memoria o E/S es enviada vía bus de direcciones y las conexiones bus de datos/direcciones. El bus de datos/direcciones es multiplexado y a veces contiene información memoria-dirección y otras veces datos. Además la salida durante T1 son las señales de control: ALE, DT/R e IO/M. La IO/M indica ya sea que el bus de direcciones contiene una dirección de memoria o un número de E/S.

T2 Durante T2 el microprocesador edita la señal RD o WR, DEN y en caso de escritura, el dato a ser escrito. Esos eventos provoca al dispositivo de E/S a ejecutar escritura/lectura. La señal DEN turna en los buffers del bus de datos, si ellos se presentan en el sistema, tanto que la memoria E/S puede recibir datos para ser escritos o el microprocesador puede aceptar la lectura de datos desde la memoria o una operación de lectura E/S. Si esto sucede para escribir ciclo del bus, entonces los datos son enviados fuera de la memoria o a través del bus de datos.

READY es mostrado al final de T2 como se muestra en la figura 14-5. Si READY es bajo en este tiempo T3, comienza un estado de espera (Tw).

T3 Este período de reloj provee para llevar el tiempo de memoria al acceso de datos. Si el ciclo de bus. Esto es también el tiempo cuando el microprocesador muestra las conexiones del bus de datos para datos, esos son leídos desde la memoria o E/S. En suma a este punto, el flanco del filo de la señal WR transfiere datos de la memoria o E/S el cual activa y escribe cuando la señal WR regresa a un nivel 1 lógico.

14.2C Temporización de Lectura

La figura 14-5 también muestra el tiempo de lectura para el microprocesador. El diagrama de tiempo permite identificar a todos los eventos principales escritos para cada estado T.

El concepto más importante en el diagrama de lectura es la cantidad de tiempo permitido a la memoria o E/S para lectura de datos. La memoria es seleccionada por el tiempo de acceso la cual es una cantidad de tiempo fijo que el microprocesador le permite en accesar datos para la operación de lectura. Por lo tanto es de suma importancia que la memoria que se elija cumpla con las limitaciones del sistema.

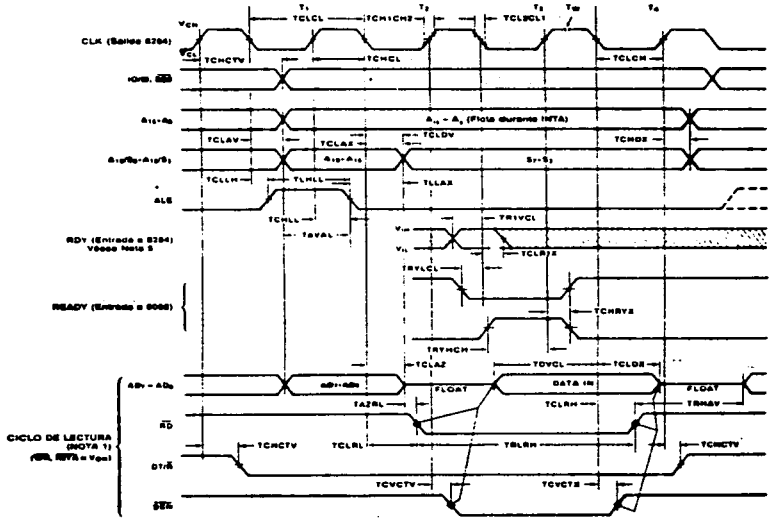

T1
T2
T3
Tw
T4
TCLCL
TCH1CH2
TCL2CL1
CLK (Salida 8284)
TCHCTV
TCHCL
TCLCH
A15-A8
A19 - A16 (Floto durante INTA)
TCLAV
TCLAX
TCLDV
TCHDX
A15/S6-A12/S3
S7-S3
TCLLH
TLHLL
TLLAX
ALE
TCHLL
TAVAL
RDY (Entrada a 8284)
Vease Nota 5
TRIVCL
TCLR1X
TRYLCL
READY (Entrada a 8088)
TCHRY2
TRYHCH
TCLAZ
TDVCL
TCLDX
AD7 - AD0
FLOAT
DATA IN
TAZRL
TCLRH
TRHAV
CICLO DE LECTURA
(NOTA 1)
TCHCTV
TCLRL
TRLRH
TCHCTV
DT/R
TCVCTV
TCVCTX
DEN

FIGURA 14-5

Los diagramas de tiempo, no siempre, proporcionan directamente a los tiempos de memoria de hecho es necesario combinar varios tiempos para llegar a un tiempo de acceso.

El tiempo de acceso de memoria comienza cuando aparece la dirección en el bus de dirección de memoria y continua hasta que el microprocesador muestrea los datos de memoria T3. Transcurren los tres estados T entre estos tiempos, pero no con exactitud.

La dirección no aparece hasta el tiempo Tclaw a 100 nseg. para el presente caso (reloj a 5MHz.) después del inicio de T1. Esto significa que el tiempo Tclaw debe ser sustraído de 3 estados de reloj 600nseg. que separan la apariencia de la dirección de T1 y el muestreo de datos T3. Algún otro tiempo debe ser sustraído de T3. El tiempo de acceso a memoria es de este modo 3 estados de reloj menos la suma Tclav - Tdvcl. Porque Tdvcl es de 30 nseg. con un reloj de 5MHz. el tiempo de acceso de memoria es de solo 400 nseg. (tiempo de acceso = 600nseg - 110 nseg - 30 nseg).

Actualmente, los dispositivos de memoria seleccionados para conectarlos a los 8088, que trabajan a 5MHz, deben estar disponibles para accesar datos a menos de 460 nseg, debido a tiempo de retardo introducido por los decodificadores de dirección y buffers en el sistema. Cuando menos un margen de 30-40 nseg para la operación correcta del 8088.

14.2D Tiempo de Escritura

Las diferencias principales entre la temporización para lectura y para escritura son mínimas. La señal **RD** se cambia por **WR**, el bus de datos contiene información para la memoria en vez de información de la memoria y **DT/R** es un en vez de un 0 lógico durante todo el ciclo de bus.

14.3 PRUEBAS DE INTEGRACION HARDWARE Y SOFTWARE

En esta sección se muestra la operación del sistema, que es el trabajo del hardware en conjunción con el software, mostrando los programas y las señales más importantes que participan.

Las señales se observaron a través de un osciloscopio y de un analizador de estados lógicos. Estos dos equipos son herramientas muy importantes en la electrónica en los cuales se puede observar y verificar el funcionamiento del sistema.

Para la construcción del sistema se utilizó una técnica de alambrado llamada WIRE WRAP, esta técnica es muy versátil y provee de una buena firmeza para las conexiones. Se necesitó una tarjeta con una matriz de orificios tipo estándar, un alambrador, cable y bases para circuitos integrados, todo esto fabricado especialmente para trabajar con esta técnica.

Los programas de prueba fueron grabados a través de un grabador de memorias EPROM conectado a una computadora con el software correspondiente. El paquete de software es el EPROM. Este grabador es un equipo que consta de una tarjeta electrónica que tiene un conector para 2 tamaños diferentes de memorias de 24 y 28 terminales. A través del software "EPROM" se indica el nivel de voltaje a programar la memoria EPROM y especificando el programa que se desea grabar.

Nuevamente se dividió en etapas para realizar las pruebas de integración, estas etapas son:

- Circuito de reloj
- Unidad de Memoria
- Puertos de Entrada/Salida
- Display
- Teclado

14.3.1 CIRCUITO DE RELOJ

A través del osciloscopio se observaron las siguientes señales del C.I. 8284

- CLK
- PCLK
- READY

Otra señal generada en el circuito de reloj es la señal RESET (Pin #11, 8284). Esta señal se encuentra en bajo al oprimir el botón de reset ejecuta en alto la señal de reset para luego regresar a bajo.

14.3.2 CIRCUITO DE PRUEBA PARA EL BANCO DE MEMORIA

Al sistema se anexó un circuito de prueba que graba los datos enviados desde la unidad de memoria. Se conectó un 74LS373 a las terminales de datos de las memorias SRAM y EPROM y a cada terminal de salida (Q's) de este 74LS373 se conecta un LED. Los cátodos de estos LED's se conectan a un bus común y a una resistencia de 1kΩ a tierra.

Este latch es controlado directamente por el 8088 agrando dos inversores en la línea IO/M. La razón de agregar estos dos inversores es el tiempo, es decir, el latch se activa antes que el resto del circuito, ya que las líneas de control para las memorias consumen más tiempo y estas compuertas son utilizadas para sincronizar los latches con el resto del circuito.

Para hacer esta prueba se grabó en memoria un sencillo programa en el que encenderá el led #0 del 373.

El procedimiento es el siguiente: El 8088 busca la primera instrucción en la dirección FFF0h cuando se enciende o reinicializa. Una vez que la 2716 es activa habilita cualquier dirección 80000h en adelante, recordando que la

memoria tiene un tamaño de 7F0h buscará esta dirección en la memoria ROM.

El programa es el siguiente:

Programa #1.

```
                ORG     7F0h

START:  MOV     AL,90h
                OUT     13h,AL
                MOV     AL,01h
                OUT     01h,AL
                ENDS

                END     START
```

El programa en código de máquina en la memoria EPROM es el siguiente:

```
00000     00 00 00 00 00 00 00 00 --00 00 00 00 00 00 00 00
00010     00 00 00 00 00 00 00 00 --00 00 00 00 00 00 00 00
-  -      -  -  -  -  -  -  -  -     -  -  -  -  -  -  -  -     -  -
-  -      -  -  -  -  -  -  -  -     -  -  -  -  -  -  -  -     -  -
007D0     00 00 00 00 00 00 00 00 --00 00 00 00 00 00 00 00
007E0     00 00 00 00 00 00 00 00 --00 00 00 00 00 00 00 00
007F0     B0 01 E6 10 EB FC 00 00 --00 00 00 00 00 00 00 00
```

Al arrancar el sistema obtenemos las siguientes señales:

- SRAM (S)
- ROM (CS)
- 373'S (LE)
- ALE (8088)

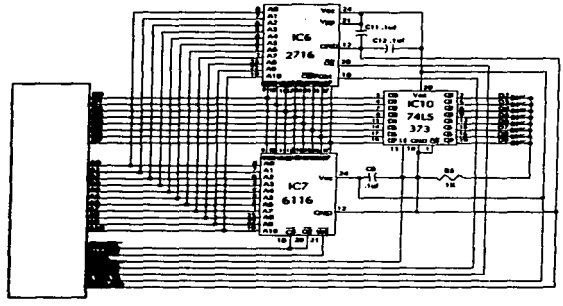


FIGURA 14-6

14.3.2 PROGRAMA DE PRUEBA PARA EL PPI

Otra sección del sistema que se programa para verificar su funcionamiento es el PPI.

En las terminales del puerto B del 8255 se conectaron 8 LED's, como se muestra en la figura 14-7. (Ver Apéndice B)

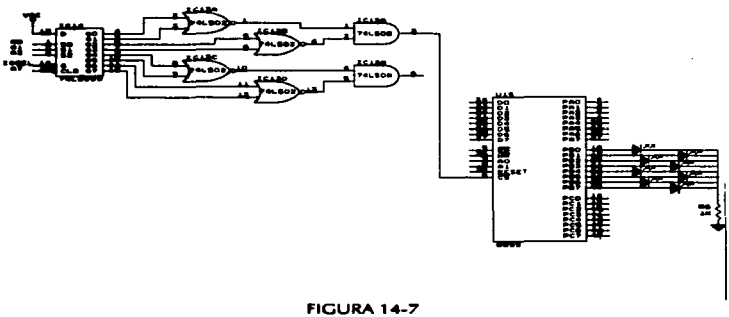


FIGURA 14-7

El programa de prueba para el circuito del del PPI consiste en que el PPI encienda el led 0 conectado en el puerto B es el siguiente:

Programa #2.

```
/* Programa en 'C' para el 8255 */
#include <dos.h>                          /* Libreria de Turbo C Version 2.0          */

#define          PTO_CTRL   0x03    /* Direccion del Registro de Control   */
#define          PTO_A      0x00    /* Direccion del Puerto A del 8255     */
#define          PTO_B      0x01    /* Direccion del Puerto B del 8255     */
#define          PTO_C      0x02    /* Direccion del Puerto C del 8255     */

main()
{
  outportb(PTO_CTRL,0x90); /* Puerto A Entrada, Puerto B Salida en*/
                           /* Modo 0; Parte Alta y Baja Puerto C*/
                           /* Salida                           */
  outportb(PTO_B  ,0x01);          /* Manda Pin 18 de 8255 a Nivel Alto*/
}
```

Código de máquina del programa

```
00000 B0 90 E6 03 EB 15 90 C7    06 00 01 FF 3F FF 0E 00
00010 01 E6 01 83 3E 00 01 00    75 F3 C3 B1 02 B0 01 E8
00020 E5 FF F6 E1 E8 E0 FF 3C    80 75 F7 F6 F1 E8 D7 FF
00030 3C 02 75 F7 EB E7 00 00    00 00 00 00 00 00 00 00
00040 00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
007D0 00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
007E0 00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
007F0 E9 0D F8 00 00 00 00 00    00 00 00 00 00 00 00 00
```

14.3.3 PROGRAMA DE PRUEBA DE LA INTERFACE DEL DISPLAY

En este programa se establecen como primer punto las palabras de control para la operación del 8279.

Este programa envía información hacia los displays, en los cuales se despliega la letra "A", activando las salidas del 74LS244 que corresponden a los segmentos de los displays, así como las salidas del 74LS138 para activar los habilitadores de los displays.

Programa #3.

```
/* Programa para verificar el display  */

#include <dos.h>

#define PTO_DIR   0x0000
#define PTO_DAT 0x0000

#define MODO      0x00          /* Modo para entrada izquierda 2-key lokout */
                                /* encoded scan */
#define   CLK               0x38    /* Divisor entre 24 */
#define   CLEAR   0xC0      /* Borra Display */
#define WDRAI     0x90      /* Write Display SRAM Auto-incremento, 1era Localidad */
```

```
main()
{
        /* Inicializacion del Display */
        outportb( PTO_DIR, MODO );          /* Programa el modo */
        outportb( PTO_DIR, CLK );           /* Programa el divisor */
        outportb( PTO_DIR, CLEAR );         /* Borra display a ceros */

        /* Manda caracteres al display */
        outportb( PTO_DIR, WDBAI );  /* Selecciona modo de escritura */
        outportb( PTO_DAT, 0x6F );   /* Manda codigo de la "A"  al 8279 */
        outportb( PTO_DAT, 0x6F );   /* Manda codigo del la "A" al 8279 */
        outportb( PTO_DAT, 0x6F );   /* Manda codigo del la "A" al 8279 */
        outportb( PTO_DAT, 0x6F );   /* Manda codigo del la "A" al 8279 */
        outportb( PTO_DAT, 0x6F );   /* Manda codigo del la "A" al  8279 */
        outportb( PTO_DAT, 0x6F );   /* Manda codigo del la "A" al  8279 */
        outportb( PTO_DAT, 0x6F );   /* Manda codigo del la "A" al  8279 */
}
```

Código de máquina del programa

```
00000  B8 00 00 8E D8 8E C0 B8      00 06 8E D0 B8 00 07 8B
00010  E8 E8 00 00 B0 00 E6 07      E0 38 E6 07 B0 90 E6 07
00020  B0 6F E6 06 B0 6F E6 06      B0 6F E6 06 B0 6F E6 06
00030  B0 6F E6 06 B0 6F E6 06      B0 6F E6 06 C3 00 00 00
00040  00 00 00 00 00 00 00 00      00 00 00 00 00 00 00 00
-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
-  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
007E0  00 00 00 00 00 00 00 00      00 00 00 00 00 00 00 00
007F0  00 00 00 00 00 00 00 00      00 00 00 00 00 00 00 00
007F0  E9 0D F8 00 00 00 00 00      00 00 00 00 00 00 00 00
```

Señales obtenidas:

- **Habilitación del 8279 (CS)**
- **Salidas OA0 - OA3**
- **Salidas OB0 - OB3**

14.3.5 PROGRAMA DE PRUEBA PARA LA INTERFACE DEL TECLADO

Este programa al igual que el programa anterior, se especifican las palabras de control para el teclado. En esta prueba también participa el display que nos sirve para verificar el funcionamiento del teclado.

Al detectar que una tecla ha sido presionada envia un mensaje que es recibido por los displays.

Programa #4.

```
/* Programa para verificar el display y el teclado*/

#include <dos.h>

#define PTO_DIR   0x0007
#define PTO_DAT 0x0006

#define MODO      0x00          /* Modo para entrada izquierda 2-key lokout */
                                /* encoded scan */
#define   CLK           0x38    /* Divisor entre 24 */
#define   CLEAR  0xC0     /* Borra Display */
#define WDRAI     0x90    /* Write Display SRAM Auto-Incremento, 1ers localidad */
#define RFR             0x40    /* Read Fifo SRAM */

#define   CHECA  0x01     /* Mascara para verificar caracteres leidos */
#define NODATO    0x00    /* No hay caracteres leidos */


main()
{
unsigned char dato_leido=0;

      /* Inicializacion del Display */
      outportb( PTO_DIR, MODO );   /* Programa el modo */
      outportb( PTO_DIR, CLK  );   /* Programa el divisor */
      outportb( PTO_DIR, CLEAR);   /* Borra display a ceros */
      outportb( PTO_DIR, WDRAI ); /* Selecciona modo de escritura  */
      outportb( PTO_DAT, 0x6D );   /* Manda codigo leido al display  */
      outportb( PTO_DAT, 0x7B );   /* Manda codigo leido al display  */
      outportb( PTO_DAT, 0x70 );   /* Manda codigo leido al display  */
      outportb( PTO_DAT, 0x6F );   /* Manda codigo leido al display  */
```

```
        while(1) {
            /* Pregunta por teclas presionadas y en caso de no encontrarlas */
            /* espera hasta que una presionada alguna tecla */
            do {
                    dato_leido = inportb( PTO_DIR );
            } while( (dato_leido & CHECA) == NODATO );

            /* Lee caracter del teclado */
            outportb( PTO_DIR, RFR );           /* Codigo de lectura de FIFO SRAM */
            dato_leido = inportb( PTO_DAT );/* Lee de la SRAM */

            /* Manda caracteres al display */
            outportb( PTO_DIR, WDRAI );         /* Selecciona modo de escritura          */

            outportb( PTO_DAT, 0x7C );          /* Manda codigo leido al display  */
            outportb( PTO_DAT, 0x76 );          /* Manda codigo leido al display  */
            outportb( PTO_DAT, 0x5E );          /* Manda codigo leido al display  */
            outportb( PTO_DAT, 0x38 );          /* Manda codigo leido al display  */
        } /* Fin del While Infinito */
}
```

Código de máquina del programa

```
00000  B8 00 00 8E D8 8E C0 B8    00 06 8E D0 B8 00 07 8B
00010  E0 E8 00 00 55 8B EC 83    EC 02 C6 46 FF 00 B0 00
00020  E6 07 B0 38 E6 07 B0 C0    E6 07 B0 90 E6 07 B0 6D
00030  E6 06 B0 7B E6 06 B0 70    E6 06 B0 6F E6 06 EB 28
00040  E4 07 8B 46 FF F6 46 FF    01 74 F5 B0 40 E6 07 E4
00050  06 88 46 FF B0 90 E6 07    B0 7C E6 06 B0 76 E6 06
00060  B0 5E E6 06 B0 38 E6 06    EB D6 8B E5 5D C3 00 00
00070  00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
-      -  -  -  -     -  -  -     -  -  -
-      -  -  -  -     -  -  -     -  -  -
007D0 00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
007E0  00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
007F0  E9 0D F8 00 00 00 00 00    00 00 00 00 00 00 00 00
```

Obteniendo las siguientes señales de operación:

- **Salidas Y0 - Y5 del 74LS138**
- **Entradas RL0- RL3 del 8279**

CAPITULO 15
GUIA DE USUARIO

15.1 INTRODUCCION

En este capítulo se describirá el sistema mínimo, los principales aspectos de operación, identificación de partes, conexiones necesarias para arrancarlo así como programas de autoprueba; proporcionando de esta manera las herramientas para el mejor aprovechamiento del sistema.

15.2 CARACTERISTICAS

Cuando a un microprocesador 8088 se le asocian varios circuitos, como es el generador de reloj, latches de direcciones y de datos, es posible generar las señales del bus de direcciones, datos y control que son necesarias para tener un sistema mínimo del 8088; si además a este sistema se le integran memorias de tipo ROM y RAM, el sistema es capaz de poder correr los programas que se tengan almacenados en la ROM; si se quieren leer señales o que se genere señales, se colocan puertos de entrada o de salida.

Este sistema por lo tanto cuenta con:

1 .- Generación de la señal de reloj. Es la circuitería encargada de proporcionar las señales de sincronización que necesita el microprocesador.

2.- Bus de Datos y Bus de Direcciones. El microprocesador 8088 tiene un bus de datos y direcciones multiplexado, es decir, que los dos tipos de buses comparten los mismos pines en el microprocesador, por esa razón, es necesario separar los dos tipos de buses. El bus de direcciones que nos sirve para poder tener acceso a cualquier dirección dentro del espacio de memoria del microprocesador. El bus de datos es el encargado de efectuar la transferencia de información entre el microprocesador y las memorias o con los dispositivos de entrada o salida.

3.- Bus de Control. Las señales provenientes del microprocesador que no son de datos o de direcciones son de control y son las que le pueden decir a la memoria que proporcione un dato al bus de datos o que lea un datos del mismo bus.

4.- Memoria ROM. Memoria donde se tienen almacenados los programas de aplicación del sistema de manera permanente.

5.- Memoria RAM. Memoria volátil donde se almacena código y valores de manera temporal.

6.- Puertos de Entrada/Salida. Las señales que se lean desde afuera del sistema podrán leerse de manera digital por el puerto de entrada y las señales que se generen hacia afuera, se escribirán en el puerto de salida de manera digital.

7.- Diodos Indicadores de Luz. Están conectados a la salida del puerto A del circuito de interfaz paralelo y su función es la de mostrar las señales que se envían por el puerto.

8.- Despiegados de 7 segmentos.

9.- El teclado tipo mecánico proporciona 24 caracteres diferentes.

El programador deberá previamente codificar su programa en hexadecimal y por medio del teclado podrá introducirlo al sistema mínimo en una forma ordenada y sin errores, en el caso que suceda un error se deberá reiniciar. De esta manera los programas podrán correr en RAM para leer o generar señales.

15.3 IDENTIFICACION DE PARTES

Los elementos principales del sistema se muestran en la FIGURA 15-1.

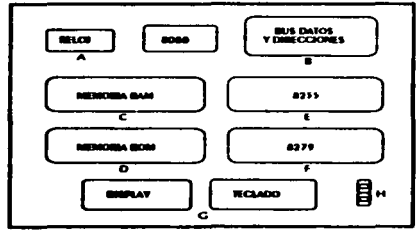


FIGURA 15-1. Diagrama del Circuito Mínimo

Sección A. En esta parte del circuito se localiza la lógica del generador de señal de reloj, en la cual se encuentra junto con el chip generador de pulsos 8284 de Intel un cristal oscilador externo y su circuitería adicional.

El pulso de reloj que sale del pin 8 del 8284 se alimenta directamente al microprocesador, estos pulsos determinan la velocidad de funcionamiento del sistema.

Sección B. Se localiza la lógica de bus de direcciones y de datos, esta sección consta de buffers, latches así como compuertas lógicas. Estos elementos resultan necesarios dado que las señales de los procesadores pueden no ser potentes para controlar el sistema o no corresponden directamente a las señales requeridas por el resto del sistema. Estos elementos se conectan a las memorias RAM y ROM.

Sección C. Memoria RAM, como ya se sabe su misión es almacenar datos y programas, se puede leer y escribir durante el funcionamiento del sistema.

Sección D. Memoria ROM, la información se almacena en la ROM por medio de la programación o reprogramación fundamentalmente en forma temporal.

Controladores de Dispositivos:

Sección E. Se tiene el Controlador Programable Paralelo de Interfaz 8255 (PPIC: Programmable Peripheral Interface), el cual sirve de ayuda en la conexión al sistema de dispositivos que envían bytes enteros, lo cual es útil

para aplicaciones que utilicen dispositivos que requieran transmisiones a gran velocidad pero no muy alejados del sistema. Conectados a un puerto del 8255 se encuentran una sección de led's el cual sirve principalmente para verificación del funcionamiento de este controlador.

Sección F. Corresponde al Controlador Programable de Teclado/Display de Interfaz 8279 (PKDI: Programmable Keyboard/Display Interface). El 8279 es una interfaz programable para teclado y display, puede manejar hasta un teclado de 64 teclas y hasta 16 displays. La sección de manejo de teclado tiene integrado circuitería para manejar un buffer que puede almacenar hasta 8 caracteres antes de que el microprocesador los reciba, este buffer es de tipo FIFO. La sección de display puede manejar hasta 16 desplegados numéricos con una RAM de 16X8 que almacena el código de los patrones que se despliegan.

Sección G. Se han colocado en esta sección un teclado tipo calculadora de 24 teclas y 7 desplegados de 7 segmentos, respectivamente. Para este sistema las teclas se clasifican en numéricas y de control, siendo las numéricas las siguientes:

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Como teclas de control se designan las siguientes funciones:

- Entrar
- Salir
- Borrar

- **Correr**
- **Editar**
- **Avanzar (+)**
- **Retroceder (-)**

La distribución de las teclas se muestra en la siguiente FIGURA 15-2:

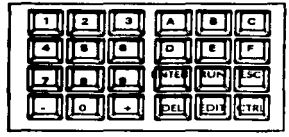


FIGURA 15-2. Distribución de teclas

La última sección (K) corresponde a las terminales de alimentación del sistema, se muestra un diagrama de conexión, figura 15-3:

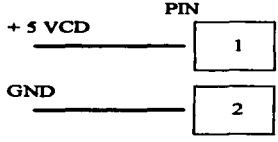


FIGURA 15-3. Sección de alimentación

15.4 ¿QUE EQUIPO SE REQUIERE PARA UTILIZAR EL SISTEMA MINIMO?

El usuario sólo necesita una fuente de corriente directa que proporcione 5 VCD., y conectar a la sección H del circuito.

Es importante mencionar que el circuito no cuenta con un dispositivo de protección en caso de conectar la alimentación en forma invertida, por lo tanto antes de conectar verifique la polarización de la fuente de poder que alimentará al sistema.

15.5 INICIO DEL PROGRAMA MONITOR

Una vez alimentado el circuito, éste comenzará el arranque de un programa de inicio llamado programa Monitor, por medio de este programa se podrá introducir el código hexadecimal de los programas de usuario además de poder ejecutar una rutina de autoprueba.

El programa Monitor es como un pequeño sistema operativo que se encarga de leer los datos del teclado para colocarlos en la memoria para su ejecución.

15.5.1 RUTINAS DE AUTOPRUEBA

El arranque del programa Monitor sigue la siguiente secuencia:

El sistema ejecutará una rutina de autoprueba a las siguientes secciones:

- Memorias
- PPI
- Visualización

En la rutina de Memorias se desplegará mediante los led's correspondientes el número 0A (hex), en seguida enviará al PPI un corrimiento de led's de izquierda a derecha y de derecha a izquierda, posteriormente se observará en los desplegados la frase "HOLA COMO ESTAS", para comprobar la sección de visualización.

15.5.2 OPERACION

Cuando el sistema despliega el mensaje "HOLA COMO ESTAS" quiere decir que está listo para la sesión de captura de programa.

Para poder introducir un programa, el usuario debe tener, ya; precodificado su programa en hexadecimal y debidamente estructurado, con la secuencia de programa adecuado, de lo contrario este no funcionará y se tendrá que ejecutar un RESET al sistema. Los datos del programa se introducirán byte por byte en direcciones consecutivas de memoria.

Se puede crear el código en hexadecimal con ayuda del lenguaje C, para hacer esto, se compila el programa que se necesite con las librerías modificadas, después se corre el programa ROMLDR para generar el archivo *.BIN y por último se corre el programa HEX2ASC para crear un archivo con

extensión *.TXT, en este archivo se tiene en forma secuencial el código en hexadecimal para ser introducido en el sistema.

A) INICIO

Para dar inicio se presiona la tecla EDITAR, con esta instrucción el sistema advierte un inicio de programa.
Con las teclas de AVANZAR Y RETROCEDER puedo regresar a modificar un sólo byte a la vez, para la verificación de captura antes de ejecutar el programa.

Para modificar se utiliza la tecla BORRAR, esta funciona únicamente para un sólo byte, pero no elimina el byte sino que lo sustituye por otro.

Para dar por terminado un byte dar la tecla ENTRAR.

Al terminar de capturar el programa se presiona inmediatamente la tecla CORRER.

B) REINICIALIZACION

Si se desea dar por terminada la ejecución del programa se presiona el botón de RESET del sistema y se reinician los pasos de captura de programa.

C) EJECUCION DE PROGRAMAS

Como se ha mencionado ésta se dará inmediatamente después de capturar el programa al presionar la tecla CORRER. Si el programa no ha sido introducido debidamente y se presiona esta tecla el programa no dará los resultados deseados y el sistema deberá ser reseteado.

15.6 LISTADO DEL PROGRAMA MONITOR :

```
/* Programa Monitor */
#include <dos.h>                                    /* Libreria Este sobre de C          */

#define         DATO_8279       0x06        /* Dir. Registro de Datos del 8279 */
#define         CTRL_8279       0x07        /* Dir. Registro Control del 8279  */

#define         SIDATOS         1                   /* Indica que hay caracteres a leer*/
#define         NODATOS                             /* Indica que no hay caracteres    */
                                0

#define         ENTRAR          0x10        /* Codigo asignado para las teclas */
#define         SALIR           0x11        /* de funciones del teclado        */
#define         EDITAR          0x12
#define         CORRER          0x13
#define BORRAR          0x14
#define         AVANZA          0x15
#define         RETROCEDE       0x16

#define         FIN                         0xAA    /* Indica fin de cadena            */

/* Definiciones para el 8279 */
/* ------------------------------------------------------------ */
#define         MOD_8279        0x00        /* Modo para entrada izquierda 2-key lokout */
                                                                        /* entidad scan */
#define         CLK_8279        0x38        /* Divisor entre 24 */
#define         CLR_8279        0xC8        /* Borra Display */
#define WRA_8279        0x90        /* Write Display Ram Auto-incremento, 1ra Localidad */
#define RPR_8279        0xe0        /* Read Fifo Ram */
/* ------------------------------------------------------------ */
                        /* Codigo de los caracteres del teclado */
                        /*  1,  2,  3,  A,  B,  C,  */
                        /*  4,  5,  6,  D,  E,  F,  */
                        /*  7,  8,  9, ENT, RUN, ESC,  */
                        /*  +,  0,  -, DEL, EDIT,  .  */

                        /* 0x00, 0x01, 0x02, 0x03, 0x04, 0x05,  */
                        /* 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,  */
                        /* 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,  */
                        /* 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D,  */

        /* Codigo de caracteres a desplegar por el  display de 7 segmentos  */
                        /*  0, 1, 2, 3, 4, 5, 6, 7,  */
                        /*  8, 9, A, B, C, D, E, F,  */
```

```
                        /* 0x7B,0x0A,0x37,0x1F,0x4E,0x5D,0x7D,0x0B, */
                        /* 0x7F,0x5F,0x6F,0x7C,0x71,0x3E,0x75,0x65, */
/* ------------------------------------------------------------ */
#define         PTO_CTRL        0x03    /* Direccion del Registro de Control 8255*/
#define         PTO_A           0x00    /* Direccion del Puerto A del 8255          */
#define         PTO_B           0x01    /* Direccion del Puerto B del 8255          */
#define         PTO_C           0x02    /* Direccion del Puerto C del 8255          */

clase_8255()
{
int x=0;
        outportb(PTO_CTRL,0x80);    /* Puerto A Entrada, Puerto B Salida en     */
                                                                /* Modo 0; Parte Alta y Baja Puerto C
        */
                                                                /* Salida
                                    */
        outportb(PTO_B      ,0xFF); /* Manda salida de 8255 a Nivel Alto        */
        for(x=0;x<10000;x++);               /* Retrazo de tiempo                                    */
        outportb(PTO_B      ,0x00); /* Manda salida de 8255 a Nivel Bajo        */
}
/* ------------------------------------------------------------ */

clase_display()
{
int x=0;

        /* Inicializacion del Display */
        outportb( CTRL_8279, MOD_8279 );        /* Programa el modo */
        outportb( CTRL_8279, CLK_8279 );        /* Programa el divisor */
        outportb( CTRL_8279, CLR_8279 );        /* Borra display a ceros */

        /* Se prepara para escribir */
        outportb( CTRL_8279, WRA_8279 );        /* Selecciona modo de escritura */

        /* Manda caracteres al display */
        outportb( DATO_8279, 0x6F );    /* Manda codigo del Nuevo al 8279 */
        outportb( DATO_8279, 0x6F );    /* Manda codigo del Nuevo al 8279 */
        outportb( DATO_8279, 0x6F );    /* Manda codigo del Nuevo al 8279 */
        outportb( DATO_8279, 0x6F );    /* Manda codigo del Nuevo al 8279 */
        outportb( DATO_8279, 0x6F );    /* Manda codigo del Nuevo al 8279 */
        outportb( DATO_8279, 0x6F );    /* Manda codigo del Nuevo al 8279 */
        outportb( DATO_8279, 0x6F );    /* Manda codigo del Nuevo al 8279 */

        for(x=0;x<10000;x++);           /* Retrazo de tiempo */

        outportb( CTRL_8279, CLR_8279 );        /* Borra display a ceros */
}
/* ------------------------------------------------------------ */

/* Rutina para verificar la memoria RAM que se encuentra a partir de */
/* la direccion XXXXXX hasta la direccion YYYYY */

#define     SEGMENTO                0x0800 /* Segmento de memoria */
#define OFFSET_INICIAL      0x0000  /* Direccion inicial de memoria */
#define OFFSET_FINAL        0x0000  /* Direccion final de memoria  */

#define ERROR       -1
#define BIEN         1

clase_memoria()
{
int offset;                                             /* Offset en memoria */
int dato_escrito = 0xAA;
int dato_leido = 0x00;                  /* Dato de lectura  */
int resultado = 0;

        /* Escribe dato en memoria */
        for( offset=OFFSET_INICIAL; offset <= OFFSET_FINAL; offset++ )
                pokeb( SEGMENTO, offset, dato_escrito );
```

```
        /* Lee dato de memoria */
        for( offset=OFFSET_INICIAL; offset <= OFFSET_FINAL; offset++ ){
                dato_leido = peekb( SEGMENTO, offset );
                if( dato_leido != dato_escrito ){
                        resultado = ERROR;
                        break;
                } /* Fin del if */
                resultado = BIEN;
        } /* Fin del For */

        outportb(PTO_CTRL,0x90);        /* Puerto A Entrada, Puerto B Salida en      */
                                                                        /* Modo 0; Parte Alta y Baja Puerto C
        */                                                              /* Salida
                                        */
        if ( resultado == BIEN )
                outportb( PTO_B , 0x01 );
        else
                outportb( PTO_B , 0x80 );
}

/* Funcion que borra el display */
borra_display()
{
        outportb( CTRL_8279, CLR_8279 );                /* Borra display a ceros */
        outportb( CTRL_8279, WRA_8279 );                /* Selecciona modo de escritura */
}

/* Funcion que decodifica los caracteres leidos */
unsigned char decodifica_caracter_leido( unsigned char dato )
{
        switch( dato ){
                case 0x19:
                        return 0x0;
                case 0x00:
                        return 0x1;
                case 0x01:
                        return 0x2;
                case 0x02:
                        return 0x3;
                case 0x08:
                        return 0x4;
                case 0x09:
                        return 0x5;
                case 0x0A:
                        return 0x6;
                case 0x10:
                        return 0x7;
                case 0x11:
                        return 0x8;
                case 0x12:
                        return 0x9;
                case 0x03:
                        return 0xA;
                case 0x04:
                        return 0xB;
                case 0x05:
                        return 0xC;
                case 0x0B:
                        return 0xD;
                case 0x0C:
                        return 0xE;
                case 0x0D:
                        return 0xF;
                case 0x13:
                        return ENTRAR;
                case 0x15:
                        return SALIR;
                case 0x1C:
```

```
                        return EDITAR;
                case 0x14:
                        return CORRER;
                case 0x1B:
                        return BORRAR;
                case 0x1E:
                        return AVANZA;
                case 0x1A:
                        return RETROCEDE;
                case 0x1D:
                        return 0x1D;
        }
}

/* Funcion que manda un caracter al display por medio del 8279 */
escribe_caracter_display( unsigned char dato )
{
unsigned char caracter;
        switch( dato ){
                case 0x19:
                        caracter =0x7B;
                case 0x00:
                        caracter =0x0A;
                case 0x01:
                        caracter =0x37;
                case 0x02:
                        caracter =0x1F;
                case 0x08:
                        caracter =0x4E;
                case 0x09:
                        caracter =0x5D;
                case 0x0A:
                        caracter =0x7D;
                case 0x10:
                        caracter =0x0B;
                case 0x11:
                        caracter =0x7F;
                case 0x12:
                        caracter =0x5F;
                case 0x03:
                        caracter =0x6F;
                case 0x04:
                        caracter =0x7C;
                case 0x05:
                        caracter =0x71;
                case 0x0H:
                        caracter =0x3E;
                case 0x0C:
                        caracter =0x75;
                case 0x0D:
                        caracter =0x65;
        }
        /*
        El CPU avisa al 8279 que va a hacer una escritura
                        Write Display RAM
        */
        outportb( DATO_8279, caracter );        /* Manda codigo leido al display */

}

/* Funcion que coloca la linea leida de memoria al display */
escribe_linea_display( unsigned char bufer[], unsigned int cuantos )
{
unsigned int cual;
        for( cual = 0; cual < cuantos; cual++ )
                escribe_caracter_display( bufer[cual] );
}
```

```
/* Funcion que lee un caracter del teclado por medio del 8279 */
unsigned char lee_caracter_teclado()
{
unsigned char caracter_1;                       /* caracter leido */
unsigned char caracter_d;                       /* caracter decodificado */
        /*
          El CPU avisa al 8279 que va a hacer una lectura
          Read FIFO/Sensor RAM
        */
        outporta( CTRL_8279, RFR_8279 );
        /*
        Se efectua una lectura al registro de datos
        */
        caracter_1 = inporta( DATO_8279 );
        caracter_d = decodifica_caracter_leido( caracter_1 );
        escribe_caracter_display( caracter_1 );         /* Manda al display lo del teclado */
        return caracter_d;
}

/* Funcion que verifica si existen datos disponibles en 8279 */
unsigned char check_estado()
{
unsigned char estado;
        /*
        Lee la palabra de control
                FIFO Status Word
        */
        estado = inporta( CTRL_8279 );              /* Lee el estado */
        estado = estado & 0x07;                                 /* operacion "AND" para saber si hay */
                                                                /* caracteres a leer */
        if( estado == 0 )
                return NODATOS;             /* no hay caracteres a leer */
        else
                return SIDATOS;
}
/* ------------------------------------------------------------ */
unsigned int offset_m   = 0x0200;

/* Funcion que manda el contenido del bufer a la memoria RAM */
manda_memoria_bufer( unsigned char bufer[], unsigned int num )
{
unsigned int o = 0;
unsigned int segmento = 0x0000;

        for ( o = 0; o < num; o++ ){
                pokeb( segmento, offset_m, bufer[o] );
                offset_m = offset_m + 1;
        }
}
/* Funcion que lee el contenido de la memoria RAM */
lee_memoria( unsigned char bufer[], unsigned int num )
{
unsigned int o = 0;
unsigned int segmento = 0x0000;

        for ( o = num; o >= 0; o-- ){
                bufer[o] = peekb( segmento, offset_m );
                offset_m = offset_m - 1;
        }
}
/* ------------------------------------------------------------ */


 void far (*programa)();                    /* funcion apuntando al inicio del area de */
                                                        /* memoria donde se guarda el programa    */
```

```
main()
{
unsigned char dato_leido;  /* Guarda el caracter leido por el teclado */
unsigned char buffer[10];  /* Buffer interno para 1 caso valido       */
unsigned int indice;       /* Indice del buffer interno               */
unsigned int indice_aux;   /* Indice auxiliar                         */


        ebase_8255();                       /* verifica el 8255 mandando un nivel */
        ebase_display();            /* verifica el display */
        ebase_memoria();            /* verifica la memoria */
        /*
          Inicializa el modo de operacion del 8279
          Keyboard/Display Modo Set
        */
        entrada( CTRL_8279, MOD_8279 );             /* Programa el modo */
        /*
          Inicializa el Reloj del 8279
          Program Clock
        */
        entrada( CTRL_8279, CLK_8279 );             /* Programa el divisor */
        /*
          Borra el display
          Clear
        */
        entrada( CTRL_8279, CLR_8279 );             /* Borra display a ceros */

        /* Se prepara para escribir */
        entrada( CTRL_8279, WRA_8279 );             /* Selecciona modo de escritura */

        indice = 0;
        while( 1 ){
                while( ebase_estado() == NODATOS )
                        ;
                dato_leido = lee_caracter_teclado();
                switch( dato_leido ){
                        case 0:
                        case 1:
                        case 2:
                        case 3:
                        case 4:
                        case 5:
                        case 6:
                        case 7:
                        case 8:
                        case 9:
                        case 0xA:
                        case 0xB:
                        case 0xC:
                        case 0xD:
                        case 0xE:
                        case 0xF:
                                buffer[indice] = dato_leido;
                                indice++;
                                break;
                        case ENTRAR:
                                if ( indice < 2 )
                                        break;
                                buffer[indice] = FIN;
                                escribe_caracter_buffer( buffer,indice );
                                indice_aux = indice;
                                indice = 0;
                                borra_display();
                                break;
                        case SALIR:
                                borra_display();
                                indice = 0;
```

```
                    break;
                case EDITAR:
                    borra_display();
                    lee_memoria( buffer, indice_aux );
                    escribe_letra_display( buffer, indice_aux );
                    break;
                case COMER:
                    borra_display();
                    /* Hacer que IP apunte a memoria */
                    /* Ejecutar el codigo del programa */
                    programa = MK_FP(0x0000,0x0200);
                    programa();
                    break;
                case BORRAR:
                    borra_display();                    /* Borra lo tecleado */
                    indice = 0;
                    break;
                case AVANZA:
                    indice = indice + 1;
                    break;
                case RETROCEDE:
                    indice = indice - 1;
                    break;
            }/* fin del switch */
        }/* fin del while infinito */
}
```

CAPITULO 16
CONCLUSIONES

El surgimiento de la idea de construir un sistema mínimo basado en el 8088 nace de la inquietud de contar con una herramienta de apoyo en el laboratorio de electrónica con esta nueva generación de microprocesadores de Intel.

Inicialmente se hizo un análisis de la utilización de los sistemas mínimos basados en un procesador de 16 bits y su aplicación en prácticas de laboratorios así como su uso en la industria y se pensó que si sería factible la idea de la construcción del sistema mínimo.

En la actualidad el estudio de los microprocesadores es muy importante ya que se han desarrollado muchos sistemas electrónicos basados en estos como computadoras, computadores de flujo, alarmas para casas y automóviles, equipos de control, etc. utilizados en la industria y hasta en el campo de la medicina.

Se recopiló la información de los dispositivos que participaron en el sistema mínimo y la integración de sus partes fundamentales: hardware y software. La integración de estas partes resulta una etapa interesante y de gran aprendizaje para cualquier alumno de ingeniería, ya que se logra interrelación de varias áreas de nuestra formación universitaria.

Este sistema es de propósito general, es decir; es capaz de manejar distintos problemas para diversas aplicaciones del usuario, contrariamente a un sistema de propósito predeterminado, que se diseña específicamente para manejar una aplicación particular. La aplicación que este sistema pueda desarrollar, dependerá de las inquietudes y de las necesidades del usuario.

Durante el desarrollo del prototipo se presentaron anomalías que impedían el buen funcionamiento del sistema. Para localizar las fallas y, poder eliminarlas éstas fueron clasificadas en dos clases:

1) Fallas de Hardware. Estas fallas fueron como falsos contactos en el alambrado, conexiones incorrectas, mala elección de dispositivos y la más difícil dispositivos dañados.

2) Fallas de Software. Errores cometidos en el manejo de códigos en la programación y fallas de quema de memorias con los programas de prueba.

La localización de estas fallas requirieron de tiempo y su solución nos proporcionó mucha experiencia.

Una de las fallas que quedó pendiente es que el teclado no responde a la solicitud del microprocesador para recibir datos y enviarlos al display, y que presenta como despliegue una infinidad de caracteres, cuando se oprime una tecla.

Para la localización de fallas es muy importante contar con herramientas como el osciloscopio, un analizador de estados lógicos y una computadora personal, así como la información de todos los dispositivos disponible en forma de manuales.

Para la quema de memorias Eproms fue una gran ventaja contar con un grabador de memorias Eproms, ya que con éste se ahorra mucho tiempo y la información grabada es más confiable.

Todas las etapas que intervinieron en la terminación del proyecto, la construcción y su programación requirió de una buena cantidad de horas de trabajo. Finalmente, el esfuerzo brindó su fruto y se obtuvo el funcionamiento del sistema, que aunque quede pendiente el correcto funcionamiento del teclado, y debido a el tiempo invertido se tiene que concluir, es una gran satisfacción el saber que funciona y que más adelante se puede utilizar como es el objetivo del presente trabajo

El diseñar, construir y programar el prototipo del sistema ha sido una gran experiencia.

APENDICE A

HOJAS DE ESPECIFICACIONES DE CIRCUITOS INTEGRADOS

FUJITSU

N-MOS 8-BIT MICROPROCESSOR

MBL 8080
MBL 8080-2
MBL 8080-1

February 1976
Edition 4.0

MBL8 8 BIT MICROPROCESSOR

The Fujitsu MBL 8080 is a new generation high performance microprocessor implemented in N-channel, depletion load silicon gate technology (MOS). and packaged in a 40-pin ceramic or plastic DIP. The processor has a reduced set of both 8 and 16 bit instructions. It is directly compatible with MBL 8080 software and most 8080A/8085 hardware and peripherals.

- 8 Bit Data Bus Interface
- 16 Bit Internal Architecture
- Direct Addressing Capability to 1 Mbyte of Memory
  Direct Software Compatibility with MBL 8080 CPU
  14 Word by 16 Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Byte, Word and Block Operations
- 6 Bit and 16 Bit Signed and Unsigned Arithmetic in Binary or Decimal, Including Multiply and Divide
- Compatible with 8155-2, 8755A-2 and 8185-2 Multiplexed Peripherals
- Two Clock Rates
  5MHz for MBL 8080,
  8MHz for MBL 8080-2,
  10MHz for MBL 8080-1
- 40-Pin DIP
  Ceramic DIP (Suffix...C)
  Plastic DIP (Suffix...P)

Fig 1 - BLOCK DIAGRAM

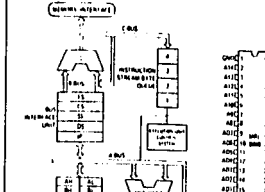
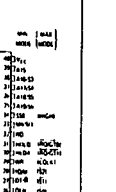


Fig 2 - PIN CONFIGURATION



Portions Reprinted by permission of Intel Corporation
© Intel Corporation 1980

MBL 8080
MBL 8080-2
MBL 8080-1

TABLE 1 – PIN DESCRIPTION

The following are functional descriptions of the MBL 8080 modules in either maximum or minimum mode. The "local bus" or direct descriptions is the direct multiplex and bus interface architecture to the MBL 8080 functions required to understand how buffered.

Symbol	Pin No.	Type	Name and Function
AD_7-AD_0	9-16	I/O	Address Data Bus: These lines constitute the time multiplexed memory/I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".
A_{15} - A_8	2-8 39	O	Address Bus: These lines provide address bits 8 through 15 for the entire bus cycle (T_1 - T_3). These lines do not have to be latched by ALE to remain valid. A_{15} - A_8 are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".
A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3	35-38	O	Address/Status: During T_1, these are the four most significant address lines for memory operations. During I/O operations, these lines are LOW. During memory and I/O operations, status information is available on these lines during T_2, T_3, T_W, and T_4. S_5 reflects the status of the interrupt enable flip-flop. S_4 is a combined 2 bit binary definition. The 2 bit binary code for S_4 S_3 is defined as follows: S_4 S_3 — 0 0: Alternate Data; 0 1: Stack; 1 0: Code or None; 1 1: Data The status information changes where required during a machine cycle but is guaranteed to be valid HIGH on T_1 and the MBL 8228 latches it on the high transition. These lines float to 3-state OFF during local bus "hold acknowledge".
$\overline{RD}$	32	O	Read: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the IO/$\overline{M}$ pin or S_1. This signal is used to read devices which reside on the MBL 8080 local bus. $\overline{RD}$ is active LOW during T_2, T_3, and T_W of any read cycle, and is guaranteed to remain HIGH on T_2 until the MBL 8080 has sampled the high level. This signal floats to 3-state OFF in "hold acknowledge".
READY	22	I	READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory or I/O is synchronized by the MBL 8284A clock generator to form READY. This signal is active HIGH. The MBL 8080 receives the READY input one clock period before the actual data transfer occurs. If the READY input is not used, it must be connected to V_{CC} to indicate that the device always responds before the end of T_3.
INTR	10	I	Interrupt Request: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt cycle. The request can be issued by an interrupting device or by setting a bit in the internal interrupt enable register. If the processor is not issued to recognize an interrupt request, it can be masked internally by software reset of the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
TEST	23	I	TEST: input is examined by the "wait for test" instruction. If the TEST input is LOW, execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
HOLD	17	I	Bus Hold: HOLD: request is an input requesting that the processor relinquish control of the local bus at the conclusion of the current bus cycle. Upon receiving the HOLD request, the processor will issue HLDA (HIGH) in the middle of the next clock period after the current bus cycle is completed. During HOLD the processor will float the local bus and control lines. The processor will maintain this condition until the HOLD request is removed. At that time the HLDA will go LOW. HOLD is not internally synchronized.

TABLE 1 – PIN DESCRIPTION (Continued)

Symbol	Pin No.	Type	Name and Function
[illegible]	24	I	RESET [illegible] program counter to zero and interrupt processor. The signal must be at least three clock periods long. During this time program counter contents are destroyed. Next instruction after RESET is taken from location zero. RESET is externally synchronized.
[illegible]	39	I	Clock pulses. The clock is driven by the processor and has a constant frequency. It is not necessary to maintain a constant duty cycle, except for standard memory timing.
[illegible]	40		V_{CC} +5V ± 5% power supply pin.
GND	1, 20		GND reference pin.
[illegible]	18	I	Maskable interrupt. Interrupt request is recognized at the end of the current instruction or between the execution of any restart instruction and the following instruction. The processor must be in the interrupt enable state for the interrupt to be recognized. Otherwise, interrupt request will be ignored. This signal is used to step the processor through a program one instruction at a time.

The following pin functions are for the 8080 CPU operating in its maximum mode (i.e. MN/$\overline{MX}$ = +5 Volts). All other pin functions are described above.

Symbol	Pin No.	Type	Name and Function
[illegible]	34	O	Status. Indicates the type of operation being performed by the processor. Status is stable during ϕ_2 and may be used for memory or I/O control. Status is active during the first clock cycle of any machine cycle. It becomes invalid during the third clock cycle unless a HOLD or a WAIT state is generated.
[illegible]	29	O	Write. Indicates that the data on the data bus will be stable for a write memory or output function. Write is active low during ϕ_2 of T2, T3, and TW when a memory or output function is selected. Write is also active low during interrupt acknowledge cycle.
[illegible]	33	O	DBIN. Indicates to external circuits that the data bus is in the input mode. This signal should be used to enable the gating of data onto the data bus from memory or input devices. DBIN is active high during ϕ_2 and ϕ_1 of T2, T3, TW, and T5. It is the same as the SYNC signal.
[illegible]	25	O	Address Latch Enable is provided by the processor to indicate that bits 8-15 of the address are present on the address bus. In a minimum configuration the ALE signal is used to strobe the status information and the state of the DBIN and WR lines into a latch.
[illegible]	27	O	Data Bus Enable (DBIN) is provided to control the direction of data flow through the transceiver. When DBIN is active (high) the transceiver is set to receive data from the data bus. DBIN is active during T2, T3, TW, and T5 states of a memory or input operation and for HLDA states.
[illegible]	26	O	Data Bus Enable (DBIN) is provided to control the direction of data flow through the transceiver. When DBIN is active (high) the transceiver is set to receive data from the data bus. DBIN is active during T2, T3, TW, and T5 states of a memory or input operation and for HLDA states.
HOLD HLDA	30, 31	I O	HOLD indicates that another master is requesting a use of the local bus. If the processor receives HOLD it will relinquish the use of the local bus at the completion of the current bus transfer. Any number of masters can be connected in a daisy chain by connecting the HLDA output to the next device in the chain. The processor will recognize HOLD only during T3 and will relinquish the bus during the following clock period. HLDA is active high during the HOLD state. The processor will lower HLDA when the HOLD request is removed.

Must be used to synchronize input devices externally to the processor. Should be grounded if not used.

TABLE 1 – PIN DESCRIPTION (Continued)

Symbol	Pin No.	Type	Name and Function
SSO	34	O	Status bus. A high pin output equal to SO on the maximum mode. The combination of SSO, DT/$\overline{R}$ and DT/$\overline{R}$ allows the system to completely decode the current bus cycle status.

SSO	DT/$\overline{R}$	SSB	Characteristics
1 (HIGH)	0	0	Interrupt acknowledge
	0	1	Read I/O Port
	1	0	Write I/O Port
	1	1	Halt
0 (LOW)	0	0	Code Access
	0	1	Read Memory
	1	0	Write Memory
	1	1	Passive

The following pin function descriptions are for the 8080A 8080A-8270 system in maximum mode (i.e. 8086/8088-GND). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function
S2 S1 S0	26 28	O	Status. Is active during clock cycle. The status becomes valid in the clock cycle in which ALE is activated. The following table defines the status of the 8288 bus commander. All of these signals are active low (logic 0) when in the active state and go inactive (logic 1) during T3 of any bus cycle. S2 may be used as a logical memory/I/O indicator, and S1 as a data/code indicator.

S2	S1	S0	Characteristics
0 (LOW)	0	0	Interrupt acknowledge
	0	1	Read I/O Port
	1	0	Write I/O Port
	1	1	Halt
1 (HIGH)	0	0	Code Access
	0	1	Read Memory
	1	0	Write Memory
	1	1	Passive

RQ/GT0 RQ/GT1	30 31	I/O	Request/Grant. These pins are used by other local processors to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is a bidirectional signal with RQ/GT0 having higher priority than RQ/GT1. RQ/GT1 has an internal pull-up resistor so may be left unconnected. The request/grant sequence is as follows (see Fig. 8): 1. A pulse of one CLK wide from another bus master indicates a local bus request (labeled 1 in the 8086 8088 timing diagram). 2. During T4 or T1 of the next CLK cycle a pulse one clock wide from the 8086 8088 to the requesting master (pulse 2) indicates that the 8086 8088 has allowed the local bus to float and that it will be safe for the bus to be granted to another master starting at the next CLK. The RQ/GT signal level is changed one clock cycle prior to the actual grant of the bus so that proper set-up time is maintained. 3. A pulse one CLK wide from the requesting master indicates to the 8086 8088 (pulse 3) that the 'hold' request is about to end and that the 8086 8088 can reclaim the bus at the next CLK. The CPU then outputs 1.

TABLE 1 – PIN DESCRIPTION (Continued)

Symbol	Pin No.	Type	Name and Function
AD.C16 AD.C11	20 31	I/O	Each address multiplexed with data bus is a sequence of three pulses. There must be one idle CLK cycle after each bus exchange. Pulses are active HIGH. If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_3 of the cycle when all the following conditions are met: 1. Request occurs on or before T_2. 2. Current cycle is not the last bus of a word. 3. Current cycle is not the last of a sequence of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. If the local bus is idle when the request is made the two possible events will follow: 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Note the last rule for a currently active memory cycle applies with condition number 1 already satisfied.
LOCK	29	O	LOCK indicates that other system bus masters are not to gain control of the system bus while LOCK is active HIGH. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is activated LOW and floats to 3-state OFF in "hold acknowledge".
QS1 QS0	24 25	O	Queue Status: provide status to allow external tracking of the internal 8086 8-BB instruction queue. The queue status is valid during the CLK cycle after which the queue operation is performed. QS1 QS0: 0 0 – No operation; 0 1 – First byte of opcode from queue; 1 0 – Empty the queue; 1 1 – Subsequent byte from queue.
	34	O	For 34 is always high on the maximum mode

FUNCTIONAL DESCRIPTION

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000H to FFFFFH. The memory is logically divided into code, data, extra data and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries (See Fig. 3).

All memory references are made relative to base addresses contained in high speed segment registers. The segment type used dictates based on the addressing mode of programs. The segment registers can be altered under program control by program providing for the ease of the following data. An independent extra segment type share the same logical addressing area, code or data. By convention, memory areas assigned to each segment register are specified as follows: local memory segment registers, program area (code) segment, data segment, extra data segment, and stack segment.

Word (16-bit) operands can be located on even or odd address boundaries. The address and data segments for the least significant byte of the word is stored at the lower valued address location and the most significant byte at the next higher address location. The BIU will automatically execute two bus cycles to read 16-bit operands.

Certain locations in memory are reserved for specific CPU operations. (See Fig. 4.) Locations from addresses FFFF0H through FFFFFH are reserved for operations including a jump to the initial program loading routine. Following RESET the CPU will always begin execution at location FFFF0H where the jump must be located. Locations 00000H through 00301H are reserved for interrupt operations. Four bytes are used for each of the 256 interrupt vectors consisting of a 16-bit segment address and a 16-bit offset address (used by programs which use one of the 256 possible interrupt service routines). The lowest addresses are assigned to the first seven of these respect of pointers to service and maintain order for the occurrence of interrupts.

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum 8086 8088 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently the 8086 8088 is configured under a single pin (MN/MX) which defines the system configuration. The definition of a certain subset of the pins changes, dependent on the condition of the strap pin. When the MN/MX pin is strapped to GND, the 8086 8088 defines pins 24 through 31 and 34 in maximum mode. When the

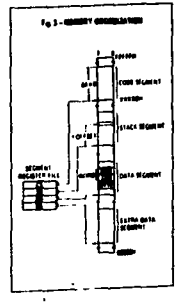


Fig. 3 – MEMORY ORGANIZATION

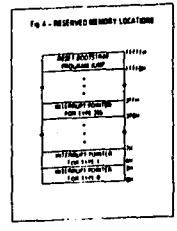


Fig. 4 – RESERVED MEMORY LOCATIONS

MN/MX pin is strapped to V_{CC}, the 8086 8088 generates bus control signals itself on pins 24 through 31 and 34.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
General Data	DATA (DS)	Data references when relative to stack, destination of string operations or explicitly overridden.
Extra (Global) Data	EXTRA (ES)	Destination of string operations. Explicitly selected using a segment override.

The minimum mode iAPX 8288 can be used with either a multiplexed or demultiplexed bus. The multiplexed bus configuration is compatible with the iSBX 251 multimodule bus and the iRMX 86, 88, 286 and 386 operating systems. The demultiplexed bus configuration (See Fig. 5) provides the user with a maximum performance system. This architecture provides the iAPX 8088 processing power in a highly integrated form.

The demultiplexed mode requires one latch (for 64K addressability) or two latches (for a full megabyte of addressing). A transceiver can be used for buffering if the address bus loading requires it. An iAPX 8286 or iAPX 8287 may also be used if data bus buffering is required. (See Fig. 6.) The iAPX 8288 provides IOR and IOW to control the transceiver and ALE to latch the addresses. This configuration of the minimum mode provides the standard demultiplexed bus structure with heavily bus buffering and reduced bus latching requirements.

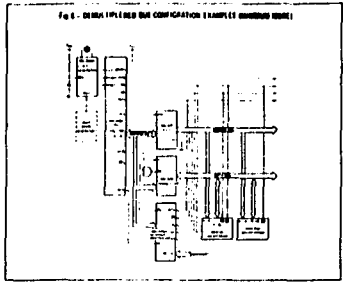
The maximum mode employs the iAPX 8288 bus controller. (See Fig. 7.) The iAPX 8288 decodes status lines $\overline{S_0}$, $\overline{S_1}$, and S_2, and provides the system with all bus command signals. Allowing the bus command to be driven by the iAPX 8288 provides better source and sink current capability to the command lines, and frees the iAPX 8088 pins for extended large system features. Multibus® bus master status and bus request/grant interfaces are provided by the iAPX 8289 bus arbiter chip. These features allow co-processors on local bus and remote bus configurations

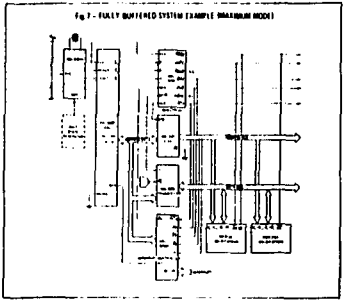
Fig 5 MULTIPLEXED BUS CONFIGURATION

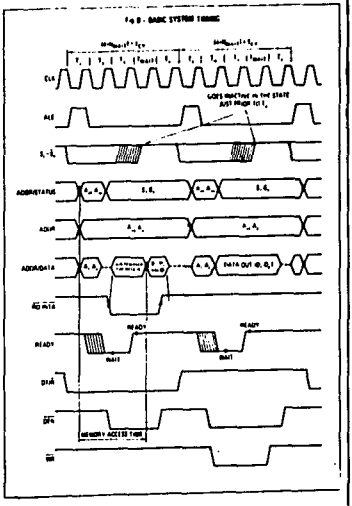
1-232

* Trade Mark of Intel Corporation, USA

1-232






CLK
ALE
S0-S2
ADDR/STATUS
ADDR
ADDR/DATA
AD RTE
READY
DT/R
DEN
MEMORY ACCESS TIME

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T_1, T_2, T_3, and T_4 (See Fig. 8). The address is presented on the data bus during T_1, and data transfer occurs on the bus during T_2, and T_3. It is also possible to stretch T_2 and T_3 by any integral number of CLK cycles. This is done by the addition of the WAIT state. The WAIT state may be inserted between T_2 and T_3. Each inserted WAIT state is of the same duration as a CLK cycle. Periods can occur between MB 8080 bus cycles. These are called T_I or idle states. If a HOLD request occurs during a bus cycle, the processor goes into the HOLD state after the next T_3 or WAIT state. In the HOLD state, the address bus (A0-A15) and the data bus (D0-D7) are placed in the high impedance state and remain so as long as HOLD is active. Control of the address bus and data bus is given to the requesting device when the processor acknowledges the HOLD request. Other CPU signals remain active during a HOLD. For example, if a HOLD request occurs during a memory read operation, memory will continue to read data into the CPU until the HOLD is recognized.

During T_1, the processor sends the 16 bit address over the address bus. Address bit A0 is the least significant address bit. Data bus bits are defined D0 (the least significant bit) through D7 (the most significant bit). During T_1, the 8080 CPU also outputs information which characterizes the machine cycle currently being performed. This information is available on the data bus during T_2, and may be latched on the falling edge of T_2, T_3, or T_4. The machine cycle status consists of eight bits of information which are defined as follows:

S_0	S_1	S_2	CHARACTERISTICS
0 (Low)	0	0	Interrupt Acknowledge
0	0	1	Write I/O
0	1	0	Read I/O
0	1	1	Halt
1 (High)	0	0	Opcode Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data in Memory
1	1	1	Acquire (Interrupt)

S_0 is defined as the MEMR command signal but S_0 is always equal to 0.

I/O ADDRESSING

In the 8080 8080 I/O operations are addressed by a maximum of 8 bit I/O registers. The I/O address appears on the same lines as the memory address on bus lines A_0 - A_7. The address lines A_8 - A_{15} are zero in I/O operations. The register I/O instructions specify one register (R) as a source from which data is output, or destination into which data is input. The other register in the pair contains the I/O address. I/O instructions directly address one of 256 I/O devices. The 8-bit I/O address allows 256 input and 256 output devices. The same register is commonly used for both input and output.

Memory-mapped I/O uses the basic 8080 memory reference instructions to address I/O ports. Any of the 8080 instructions which can reference memory may be used for I/O. With memory-mapped I/O, a memory address is assigned to each I/O port. The program then uses the appropriate memory instructions to read from and write to the I/O port. The 8080 8080 does not support memory-mapped I/O.

EXTERNAL INTERFACE

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with a minimum delay of the RESET pin. The 8080 8080 RESET is designed to be insensitive to rise time and fall time of the incoming signal. The 8080 8080 contains a Schmitt trigger circuit that senses the low to high going edge of RESET and can respond to an interrupt request as long as RESET is active. The low-going transition of RESET triggers an internal reset sequence that requires exactly 3 clock cycles. After this interval the 8080 8080 operates normally, beginning with the instruction at program location 0 (zero). See Fig. 11. The RESET input is externally synchronized to the processor clock. An external flip-flop is provided on the 8224 clock generator chip for this purpose. The user must provide a resistor and capacitor network to the RESET input to assure that the voltage at the pin will remain below 0.8 volts for at least 50 milliseconds after power has been applied. This will allow sufficient time for the internal clock generator circuitry to stabilize the frequency of oscillation. The RESET input may be driven directly from a Schmitt trigger source. The 8224 clock generator provides a Schmitt trigger function for this purpose.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes: software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are controlled by the instruction set description of the 8080 8080 Family User's Manual. Hardware interrupts can be classified as non-maskable or maskable.

Interrupts result in a number of control or data processing operations. A 256 element table containing address pointers to the interrupt service routines for each interrupt is provided. Each pointer is two words long. The first word contains the 16-bit address of the interrupt service routine. The second word contains the 16-bit address of the data segment base. The pointer table is addressed internally by the interrupt processing logic as a function of the interrupt level. Software initiated interrupts are provided for by the RST instruction. This instruction is a one byte call instruction to any one of eight locations. The three bits that select the restart location are user definable. Any of the eight restart instructions may be inserted cheaply into user software for subroutine linkage. Any of the eight restart locations may be used for hardware interrupts.

NON-MASKABLE INTERRUPT (TRAP)

The processor provides a single non-maskable interrupt (TRAP) pin which has a higher priority than the maskable interrupt request (INTR) pin. A typical use would be to initiate a power failure routine. The TRAP is edge triggered on a LOW to HIGH transition. The acceptance of this pin causes a type 2 interrupt.

TRAP is guaranteed to have a minimum of one HIGH state and program time (one clock cycle), but is not recognized to be valid unless it is set for more than a specified minimum of 0.5 usec. A typical use would be to initiate a power failure routine. The TRAP must not be recognized again until after the return instruction of the current interrupt is executed unless a second TRAP occurs. This feature allows nesting of TRAP interrupts. If another TRAP occurs before the return instruction is executed, only one more will be recognized.

MASKABLE INTERRUPT (INTR)

The 8080 8080 provides a single interrupt request input (INTR) which can be masked internally by both software and the occurrence of an interrupt acknowledge (INTA) signal. The interrupt request input is level triggered. It is normally synchronized during each clock cycle on the high going edge of CLK. To be recognized, an INTR must be present (HIGH) during the clock period preceding the end of the last instruction or at the end of a whole wait for a HOLD type instruction. During interrupt acknowledge cycle further interrupts are disabled. The program can re-enable interrupts at some later point through the EI/DI instructions. The EI instruction sets the interrupt enable flip-flop one instruction cycle after it is executed. The DI instruction resets the interrupt enable flip-flop immediately. Interrupts are also disabled by the internal reset and TRAP signals. Both of these automatically reset the interrupt enable flip-flop. The current state of the interrupt enable flip-flop is stored in bit position 3 of the PSW (Processor Status Word) during the execution of a PUSH PSW instruction. The interrupt enable flip-flop may be read by storing the PSW in memory and testing bit position 3. The interrupt enable flip-flop is reset by the acceptance of an interrupt, so that program interrupts will be disabled at the beginning of an interrupt service routine unless the interrupt enable flip-flop is set by an EI instruction executed by the interrupt service routine.

During the interrupt sequence (See Fig. 9) the processor executes two successive input (data bus) interrupt acknowledge cycles. The MEMR, 8080 sends the LOCK signal HIGH when each data bus. If the last two cycles are used S_1 of the second. A bit of bus, hold request will not be honored until the end of the second bus cycle so the second bus cycle is also a trusted input the processor interrupt system (e.g. 8214 8259A PIC) which stimulates the branch routine of the interrupt. This latter is multiplexed by four and used as a pointer into the interrupt vector lookup table. An eight-level 8080 8080 will be continued by responded to confirm the acceptance of the enable bus and sample period. The interrupt request instruction exchanges a four word which returns the status of the original interrupt request to restore the flags.

HALT

When a software HALT instruction is executed, the processor indicates that it is entering the HALT state in one of two ways: depending upon which mode is assigned by processor mode, the processor issues ALE (data out by one clock cycle) to allow the system to latch the halt status. Halt status is available on IO/M, D1-R, and SSO. In any case mode the processor issues appropriate HALT status on S_0, S_1, and S_2, and the MEMR, 8080 bus remain inactive and ALE. The 8080 8080 will not leave the HALT state unless a valid bus hold is received while in HALT. In this case the processor exits the HALT state prior to the end of the next bus hold. An interrupt request or RESET will leave the 8080 8080 out of the HALT state.

BI-DIRECTIONAL/TRI-STATE (SHARED BUS) OPERATIONS VIA LOCK

The LOCK signal information is provided for the one person when conversion bus cycles are required during the execution of an instruction. This allows the processor to perform indivisibility in the read-modify-write sequence of the exchange register with memory, semaphore and the shared system bus master accessing memory. This is useful in multiple-bus master system configurations to prevent test and set lock operations. The LOCK signal is programmed (LOW) in the clock cycle following the issue of the LOCK prefix instruction. It is guaranteed at the end of the last bus cycle of the instruction but using the LOCK prefix. While LOCK is active a request on a HOLD line will be recorded, and then honored at the end of the LOCK.

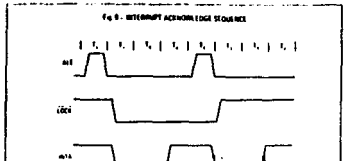


Fig. 9 - INTERRUPT ACKNOWLEDGE SEQUENCE

EXTERNAL SYNCHRONIZATION VIA TEST

As an alternative to interrupts, the MBL 8088 provides a single software-testable input pin (TEST). This input is utilized by executing a WAIT instruction. The single WAIT instruction is repeatedly executed until the TEST input goes active (LOW). The execution of WAIT does not consume bus cycles once the queue is full.

If a local bus request occurs during WAIT execution, the MBL 8088 3-states all output drivers. If interrupts are enabled, the MBL 8088 will recognize interrupts and process them. The WAIT instruction is then refetched, and reexecuted.

BASIC SYSTEM TIMING

In minimum mode, the MN/MX pin is strapped to V_{CC} and the processor emits bus control signals compatible with the 8085 bus structure. In maximum mode, the MN/MX pin is strapped to GND and the processor emits coded status information which the MBL 8288 bus controller uses to generate MULTIBUS* compatible bus control signals.

SYSTEM TIMING – MINIMUM SYSTEM

(See Fig. 9)

The read cycle begins in T_1 with the assertion of the address latch enable (ALE) signal. The trailing (low going) edge of this signal is used to latch the address information, which is valid on the address/data bus (AD_0–AD_7) at this time, into the MBL 8282/8283 latch. Address lines A_8 through A_{15} do not need to be latched because they remain valid throughout the bus cycle. From T_1 to T_4 the IO/M signal indicates a memory or I/O operation. At T_2 the address is removed from the address/data bus and the bus goes to a high impedance state. The read control signal is also asserted at T_2. The read (RD) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later, valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver (MBL 8286/8287) is required to buffer the MBL 8088 local bus, signals DT/R and DEN are provided by the MBL 8088.

A write cycle also begins with the assertion of ALE and the emission of the address. The IO/M signal is again asserted to indicate a memory or I/O write operation. In T_2, immediately following the address emission, the processor emits the data to be written into the addressed location. This data remains valid until at least the middle of T_4. During T_2, T_3, and T_W, the processor asserts the write control signal. The write (WR) signal becomes active at the beginning of T_2, as opposed to the read, which is delayed somewhat into T_2 to provide time for the bus to float.

---

*Trade Mark of Intel Corporation, USA

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge (INTA) signal is asserted in place of the read (RD) signal and the address bus is floated. (See Fig. 9) In the second of two successive INTA cycles, a byte of information is read from the data bus, as supplied by the interrupt system logic (i.e. MBL 8259A priority interrupt controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into the interrupt vector lookup table, as described earlier.

BUS TIMING – MEDIUM COMPLEXITY SYSTEMS

(See Fig. 10)

For medium complexity systems, the MN/MX pin is connected to GND and the MBL 8288 bus controller is added to the system, as well as an MBL 8282/8283 latch for latching the system address, and an MBL 8286/8287 transceiver to allow for bus loading greater than the MBL 8088 is capable of handling. Signals ALE, DEN, and DT/R are generated by the MBL 8288 instead of the processor in this configuration, although their timing remains relatively the same. The MBL 8088 status outputs (S_2, S_1, and S_0) provide type of cycle information and become MBL 8288 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The MBL 8288 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The MBL 8288 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence, data is not valid at the leading edge of write. The MBL 8286/8287 transceiver receives the usual T and OE inputs from the MBL 8288's DT/R and DEN outputs.

The pointer into the interrupt vector table, which is passed during the second INTA cycle, can derive from an MBL 8259A located on either the local bus or the system bus. If the master MBL 8259A priority interrupt controller is positioned on the local bus, a TTL gate is required to disable the MBL 8286/8287 transceiver when reading from the master MBL 8259A during the interrupt acknowledge sequence and software "poll".

THE MBL 8088 COMPARED TO THE MBL 8086

The MBL 8088 CPU is an 8-bit processor designed around the MBL 8086 internal structure. Most internal functions of the MBL 8088 are identical to the equivalent MBL 8086 functions. The MBL 8088 handles the external bus the same way the MBL 8086 does with the distinction of handling only 8 bits at a time. Sixteen-bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the MBL 8088 and MBL 8086 are outlined below. The engineer who is unfamiliar with the MBL 8086 is referred to the MBL 8086 Family User's Manual, Chapters 2 and 4, for function description and instruction set information.

Internally, there are three differences between the MBL 8088 and the MBL 8086. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the MBL 8088, whereas the MBL 8086 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The MBL 8088 BIU will fetch a new instruction to load into the queue each time there is a 1 byte hole (space available) in the queue. The MBL 8086 waits until a 2 byte space is available.
- The internal execution time of the instruction set is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU is also limited by the speed of instruction fetches. This latter problem only occurs when a series of simple operations occur. When the more sophisticated instructions of the MBL 8088 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

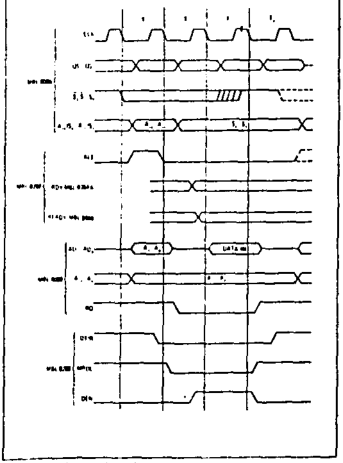
The MBL 8088 and MBL 8086 are completely software compatible by virtue of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally as well on an MBL 8088 or an MBL 8086.

The hardware interface of the MBL 8088 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes:

- A_8–A_{15} — These pins are only address outputs on the MBL 8088. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- BHE has no meaning on the MBL 8088 and has been eliminated.

- SSO provides the SO status information in the minimum mode. This output occurs on pin 34 in minimum mode only. DT/R, IO/M, and SSO provide the complete bus status in minimum mode.
- IO/M has been inverted to be compatible with the MCS-85* bus structure.
- ALE is delayed by one clock cycle in the minimum mode when entering HALT, to allow the status to be latched with ALE.

Fig 10 - MEDIUM COMPLEXITY SYSTEM TIMING



*Trade Mark of Intel Corporation, USA

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-0.5 to +7V
Power Dissipation	2.5 Watt

*NOTE: Permanent device damage may occur if ABSOLUTE MAXIMUM RATINGS are exceeded. Functional operation should be restricted to the conditions as detailed in the operational sections of this data sheet. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

(MBL 8088: V_{CC} = 5V ± 10%, T_A = 0°C to 70°C)
(MBL 8088-2: V_{CC} = 5V ± 5%, T_A = 0°C to 70°C)
(MBL 8088-1: V_{CC} = 5V ± 5%, T_A = 0°C to 70°C)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400 μA
I_{CC}	Power Supply Current MBL 8088 / MBL 8088-2 / MBL 8088-1		340 / 350 / 360	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	-0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Capacitance of Input Buffer (All input except AD_0 - AD_7, RQ/GT)		15	pF	f_c = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD_0 - AD_7, RQ/GT)		15	pF	f_c = 1 MHz

FUJITSU

MBL 8284A
MBL 8284A-1

BIPOLAR CLOCK GENERATOR AND DRIVER FOR MBL

- Generates the System Clock for the Processors
  5MHz and 8MHz with MBL 8284A
  10MHz with MBL 8284A-1
- Uses a Crystal or a TTL Signal for Frequency Source
- Provides Local READY and Multibus* READY Synchronization
- Generates System Reset Output from Schmitt Trigger Input
- Capable of Clock Synchronization with Other MBL 8284As
- Single +5V Power Supply
- 18 Pin Ceramic (Suffix, CZ)

Fig 1 - BLOCK DIAGRAM

Fig 2 - PIN CONFIGURATION

*Trade Mark of Intel Corporation

Portions Reprinted by permission of Intel Corporation

©Intel Corporation, 1983

FUJITSU MBL 8284A
MBL 8284A-1

PIN DESCRIPTION

Table 1 - PIN DESCRIPTION

Symbol	Type	Name and Function
AEN1, AEN2	I	Address Enable. AEN1 is an active LOW signal. AEN2 is an active HIGH signal. Each is synchronized internally to provide a command signal (RDY1 or RDY2) to the bus ready synchronization circuit when its respective bus master is ready to transfer data. These inputs are used in systems configurations where access to the bus is controlled by two bus masters. In such systems, each bus master generates a bus request signal (CRQLCK) which is applied to the bus priority resolving circuitry. When a bus master receives control of the bus, it activates its AEN signal. These signals prevent the 8284A from passing a ready condition to a bus master that does not have control of the bus.
RDY1, RDY2	I	Bus Ready (Transfer Completed). RDY1 or RDY2 (depending on which bus master has control of the bus) is an active HIGH signal used to indicate to the 8284A that a data transfer has been completed. When a bus master has control of the bus, it responds to its HLDA signal by activating its RDY input. This RDY signal is synchronized by the 8284A and then sent to the CPU as the READY signal.
ASYNC	I	Ready Synchronization Select. When ASYNC is in the low state, two sets of synchronization flip-flops are used to synchronize the RDY, AEN1, and AEN2 signals to the CPU clock. When ASYNC is in the high state, only one set of synchronization flip-flops is used for these signals. This is sufficient when the RDY, AEN1, and AEN2 signals are derived from a source that is synchronized to the CPU clock.
READY	O	Ready. READY is an active HIGH signal which is the synchronized RDY input signal. READY is cleared after the guaranteed hold time to the processor has been met.
X1, X2	I	Crystal In. X1 and X2 are the pins to which a crystal is attached. The crystal frequency is 3 times the desired processor clock frequency.
F/C	I	Frequency/Crystal Select. When F/C is at a logic high, the frequency of the signal applied to EFI is multiplied by three to generate the clock output. When F/C is at a logic low, a crystal is used to generate the clock output. When F/C is pulled HIGH, CLK is generated from the EFI input.
EFI	I	External Frequency. When F/C is pulled HIGH, CLK is generated from the input frequency applied to this pin. The input signal is a square wave 3 times the frequency of the desired CLK output.
CLK	O	Processor Clock. CLK is the clock output used by the processor and all devices which directly connect to the processor's local bus. The frequency is one-third the crystal or EFI input frequency and has a 33% duty cycle. The output HIGH of 4.5 volts (V_{CC} = 5V) is provided on this pin to drive MOS devices.
PCLK	O	Peripheral Clock. PCLK is a TTL level square wave clock signal whose output frequency is 1/2 that of CLK and has a 50% duty cycle.
OSC	O	Oscillator Output. OSC is the TTL level output of the internal oscillator circuitry. Its frequency is equal to that of the crystal.
RES	I	Reset In. RES is an active LOW signal which is used to generate RESET. The MBL 8284A provides a Schmitt trigger input so that an RC combination can be used to establish the power up reset of proper duration.
RESET	O	Reset. RESET is an active HIGH signal which is used to reset the MBL 8086 family processors. Its timing characteristics are determined by RES.
CSYNC	I	Clock Synchronization. CSYNC is an active HIGH signal which allows multiple MBL 8284A to be synchronized to provide clocks that are in phase. When CSYNC is HIGH the internal counters are reset. When CSYNC goes LOW the internal counters are allowed to resume count using CSYNC rising to the internally synchronized to EFI. When using the internal oscillator CSYNC should be hardwired to ground.
GND		Ground
V_{CC}		Power +5V supply

FUNCTIONAL DESCRIPTION

GENERAL

The MB 8284A is a single chip clock generator/driver for the MB 8086, MB 8088, MB 80186 and MB 80188 microprocessors. The chip contains a crystal-controlled oscillator, a divide-by-three counter, complete MULTIBUS™ ready synchronization and reset logic. Refer to Figure 1 for a block diagram and Figure 2 for Pin Configuration.

OSCILLATOR

The oscillator circuit of the MB 8284A is specially designed for use with an external series resonant fundamental mode crystal from which the basic operating frequency is derived.

The crystal frequency should be selected at three times the required clock rate. A and B2 are the two crystal input (XTAL) connections. For the most stable operation of the oscillator circuit, a parallel resonant, fundamental mode, crystal should be employed. The crystal is connected between A and B2 with A connected through a series resistor to the crystal input and B2 connected directly to ground. Both A and B2 are specified to have a maximum capacitance of 7 pF. The crystal should be mounted as close as possible to the inputs to minimize lead length and stray capacitance. The oscillator may also be driven from an external frequency source. In this configuration, the external source is connected to A and B2 is left open circuited.

The system's clock is derived by dividing the crystal or input frequency by three and is specified to have a 33% duty cycle (high for one-third of a period, low for two-thirds of a period). The clock output is designed to drive the MB 8086, MB 8088, MB 80186, MB 80188 microprocessors directly. Refer to the TTL and peripheral driver specification section for the output drive characteristics.

The clock output oscillates at the crystal frequency. It is not synchronized to the internal or external timing signals but is used internally to synchronize the READY, RES and ASYNC signals.

RESET LOGIC

The reset logic provides a Schmitt trigger input (RES) and a synchronizing flip-flop to generate the reset timing. The reset signal is synchronized to the falling edge of CLK. A simple RC network can be used to provide power-on reset by utilizing the internal pull-up resistor.

READY SYNCHRONIZATION

Two READY inputs (RDY1, RDY2) are provided for asynchronous, two Multi-Master system buses. Each input has a qualifier (AEN1 and AEN2 respectively). The AEN signals validate their respective RDY signals. If a Multi-Master system is not being used, the AEN pin should be tied low.

Synchronization is required for all asynchronous active going edges of either RDY input to guarantee that the RDY setup and hold times are met. Inactive going edges of RDY in multiple master systems do not require synchronization but must satisfy RDY setup and hold in a master-of-system design.

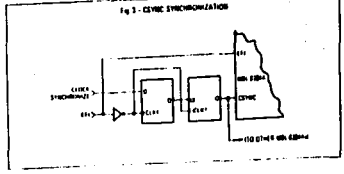
The ASYNC input defines two modes of READY synchronization operation.

When ASYNC is LOW, two stages of synchronization are provided for either of RDY input signals. Positive going asynchronous READY inputs will first be synchronized to the high level at the next falling edge of CLK, then will pass on to the next rising edge of CLK to propagate a high level at the next falling edge of CLK, after which time the READY output will go active (HIGH). Negative going asynchronous READY inputs will be synchronized directly to the flip-flop on the falling edge of CLK, after which time the READY output will go inactive. This mode of operation is intended for use by asynchronous (non-multi-master) devices in the system which cannot be guaranteed to meet the required RDY setup times.

When ASYNC is high or left open, the first READY flip-flop is bypassed so the READY synchronization logic READY inputs are synchronized by the flip-flop on the falling edge of CLK before they are presented to the processor. This mode is available for synchronous devices that can be guaranteed to meet the required RDY setup time.

ASYNC can be changed on every bus cycle to select the appropriate mode of synchronization for each device in the system.

Fig 3 - CSYNC SYNCHRONIZATION



ABSOLUTE MAXIMUM RATINGS*

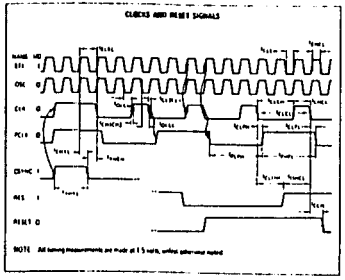
Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +115°C
Supply Voltage	-0.5V to +7.0V
All Input and Output Voltages	-0.5V to V_{CC}
Power Dissipation	1.0W

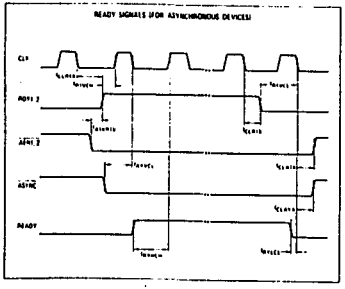
*NOTE: Exposure above damage may occur if ABSOLUTE MAXIMUM RATINGS are exceeded. Functional operation should be restricted to the conditions as detailed in the operational sections of this data sheet. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C)

Symbol	Parameter		Min	Max	Units	Test Conditions
I_F	Forward Input Current	ASYNC		1.3	mA	V_F = 0.45V
		Other Inputs		0.5	mA	V_F = 0.45V
I_R	Reverse Input Current	ASYNC		50	μA	$V_R = V_{CC}$
		Other Inputs		50	μA	V_R = 5.25V
V_C	Input Forward Clamp Voltage			-1.0	V	I_C = -5mA
I_{CC}	Power Supply Current			160	mA	
V_{IL}	Input LOW Voltage			0.8	V	
V_{IH}	Input HIGH Voltage		2.6		V	
V_{RES}	Reset Input HIGH Voltage		2.6		V	
V_{OL}	Output LOW Voltage			0.45	V	I_{OL} = 5mA
V_{OH}	Output HIGH Voltage	CLK	4		V	I_{OH} = -1mA
		Other Outputs	2.4		V	I_{OH} = -1mA
V_{HYS}	RES Input Hysteresis		0.25		V	

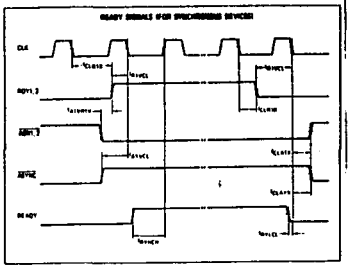
WAVEFORMS

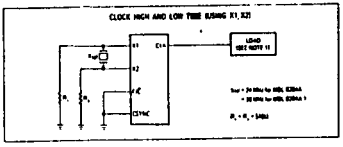


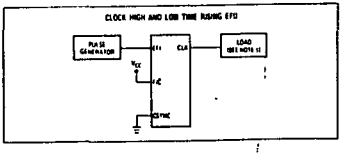


1.306

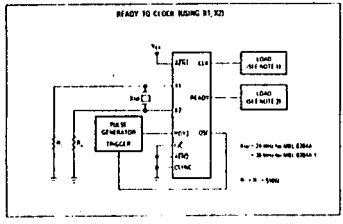
WAVEFORMS (Continued)

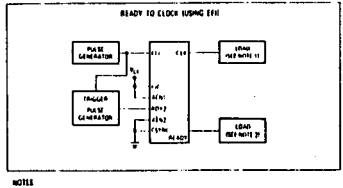






1.307



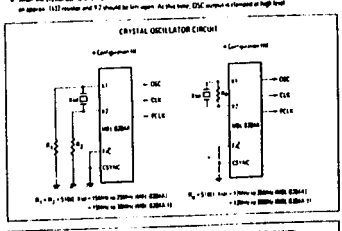


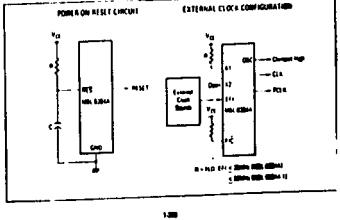
NOTES
1. C_L = 100 pF
2. C_L = 30 pF

NOTES FOR USING MSL 82844

For MSL 82844's stable operation the following things should be noted.

1. The fundamental mode crystal should be used.
2. Stray capacitances between X1 and X2 should be limited to less than 10pF.
3. External resistors for stable crystal oscillation should be connected as Configuration III or IV shown below. Especially for Configuration III circuit the maximum frequency of crystal is limited to 15MHz. A low frequency crystal (less than 15MHz) with Configuration III circuit may cause an abnormal oscillation. For more stable oscillation over the specified frequency range (170kHz to 25MHz) at 82844-1 Configuration III circuit is recommended.
4. The rise time of the power supply voltage V_{CC} should be more than 10ms. A slow V_{CC} rising (with the rise time less than 10ms) may cause an abnormal oscillation.
5. In the power on reset circuit shown below, the ground terminal of the external capacitor C should be grounded at near to the GND pin of MSL 82844 as possible.
6. When the crystal oscillation circuit is not used (i.e. an external clock source is used) X1 should be pulled up to V_{CC} with an approx. 1kΩ resistor and X2 should be left open. At this time OSC output is clamped at high level.





intel

2716

MEMORIA PROM BORRABLE POR UV DE 16K (2K × 8)

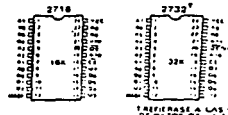
- **Tiempo de acceso rápido:**
  - **350 ns máx. 2716-1.**
  - **390 ns máx. 2716-2.**
  - **450 ns máx. 2716.**
  - **490 ns máx. 2716-5.**
  - **650 ns máx. 2716-6.**
- **Patillas compatibles con el EPROM 2732 de Intel®.**
- **Requisitos sencillos de programación:**
  - **Programación con posicionamiento único.**
  - **Programas con un impulso de 50 ms.**
- **Alimentación de +5 V unicamente.**
- **Entradas y salidas compatibles con TTL durante lectura y programa.**
- **Baja disipación de potencia:**
  - **525 mW potencia máxima en activo.**
  - **132 mW potencia máxima en reserva.**
- **Completamente estática.**

La Intel® 2716 es una memoria EPROM de 16 384 bits para lectura solo, programable electricamente y borrable con luz UV. La 2716 funciona con una fuente de alimentación sencilla de 5 V, posee un modo de reserva estático y ofrece una programación de direccionamiento único y rápido. Hace que el diseño con EPROM sea más rapido, más fácil y más económico.

La 2716, con su fuente de alimentación única de 5 V y con un tiempo de acceso de hasta 350 ns, es ideal para usarla con los nuevos microprocesadores de elevada eficiencia, alimentados a 5 V, tal como los Intel 8085 y 8086. También están disponibles las 2716-5 y 2716-6 para aplicaciones de menos velocidad. La 2716 es, además, la primera EPROM que presenta un modo de reserva estático que reduce la potencia disipada sin aumentar el tiempo de acceso. La máxima disipación de potencia en activo es de 525 mW, mientras que en reserva es de solo 132 mW, con un ahorro del 75 %.

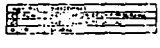
La 2716 ofrece el método mas simple y mas rápido conocido en la actualidad para programar EPROM; o sea, la programación monopulso con nivel TTL. No hay necesidad de impulsos de mayor tensión, ya que todos los controles de programación están gobernados por las señales TTL. Se puede programar cualquier dirección en cualquier momento, sea individual, secuencial o aleatoriamente, gracias a la programación con dirección única de la 2716. El tiempo total para programar todos los 16 384 bits es de 100 segundos solamente.

CONFIGURACION DE PATILLAS



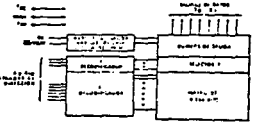
*REFERASE A LAS HOJAS DE DATOS DE LA 2732 PARA VER LAS CARACTERISTICAS

NOMBRE DE LAS PATILLAS



SELECCION DE MODOS

DIAGRAMAS DE BLOQUES



PROGRAMACION

Las características de programación se describen en el Apartado de Instrucciones para la Programación de PROM/ROM del Catálogo de Datos.

Regímenes máximos absolutos*

Temperatura bajo polarización	−10°C a +80°C
Temperatura de almacenamiento	−65°C a +125°C
Todas las tensiones de entrada o salida con respecto a tierra	+6 V a 0,3 V
Tensión de alimentación V_{PP} con respecto a tierra	−26,5 V a −0,3 V

*COMENTARIO. Los excesos por encima de los valores indicados bajo los regímenes máximos absolutos pueden ocasionar daños permanentes al dispositivo. Esto es un régimen de esfuerzo solamente y el funcionamiento del dispositivo no en esta indicado bajo estas o otras condiciones por encima de las indicadas en el apartado siguiente de estas especificaciones. La exposición de los dispositivos a las condiciones de los regímenes máximos absolutos puede afectar a la fiabilidad del dispositivo.

Condiciones de funcionamiento en c.c. y c.a. durante la lectura

	2716	2716-1	2716-2	2716-5	2716-6
Margen de temperatura	0°C – 70°C	0°C – 70°C	0°C – 70°C	0°C – 70°C	0°C – 70°C
Tensión continua de alimentación V_{CC} [1,2]	5 V ± 5 %	5 V ± 10 %	5 V ± 5 %	5 V ± 5 %	5 V ± 5 %
Tensión continua de alimentación V_{PP} [2]	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}

FUNCIONAMIENTO EN LECTURA

Corriente continua y características de funcionamiento

Símbolo	Parámetro	Límites			Unidad	Condiciones
		Mín.	Tip.[3]	Máx.		
I_{LI}	Corriente de carga a la entrada			10	μA	$V_{IN} = 5{,}25$ V
I_{LO}	Corriente de fuga a la salida			10	μA	$V_{OUT} = 5{,}25$ V
I_{PP1} [2]	Corriente V_{PP}			5	mA	$V_{PP} = 5{,}25$ V
I_{CC1} [2]	Corriente V_{CC} reserva		10	25	mA	$\overline{CE} = V_{IH}$, $\overline{OE} = V_{IL}$
I_{CC2} [2]	Corriente V_{CC} (activada)		75	100	mA	$\overline{OE} = \overline{CE} = V_{IL}$
V_{IL}	Tensión de entrada para nivel bajo	−0,1		0,8	V	
V_{IH}	Tensión de entrada para nivel alto	2,0		V_{CC} + 1	V	
V_{OL}	Tensión de salida para nivel bajo			0,45	V	$I_{OUT} = 2{,}1$ mA
V_{OH}	Tensión de salida para nivel alto	2,4			V	$I_{OUT} = -400$ μA

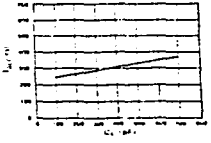
NOTAS.

1. V_{CC} debe ser aplicada simultáneamente o antes que V_{PP} y retirada simultáneamente o después que V_{PP}.
2. V_{PP} puede conectarse directamente a V_{CC} excepto durante la programación. La corriente de alimentación sería entonces la suma de I_{CC} e I_{PP}.
3. Los valores típicos son para $T_A = 25$°C y tensiones de alimentación nominales.
4. Este parámetro se mide en una muestra y no se prueba al 100 %.

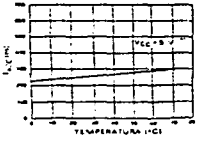
Características típicas

CORRIENTE I_{CC} FRENTE A TEMPERATURA

TIEMPO DE ACCESO FRENTE A CAPACITANCIA



TIEMPO DE ACCESO FRENTE A TEMPERATURA



CARACTERISTICAS DE BORRADO

Las características de borrado de la 2716 son tales que el borrado comienza en cuanto se la expone a luz de longitudes de onda más cortas que unos 4000 Angstroms (A). Hay que notar que la luz solar y la de algunos tipos de lámparas fluorescentes contienen longitudes de onda dentro de la gama de 3000 a 4000 A. Los datos revelan que la exposición constante al nivel de iluminación fluorescente de una habitación podría borrar una 2716 típica en aproximadamente 3 años, mientras que la exposición directa a la luz solar lo haría en una semana. Intel dispone de etiquetas opacas para las cavidades en que la 2716 haya de ser expuesta a estas condiciones de iluminación durante períodos prolongados de tiempo. Estas etiquetas se deben colocar sobre la ventana de la 2716 para evitar su borrado involuntario.

El procedimiento recomendado para el borrado (vease el Apéndice de Instrucciones de Programación de PROM,ROM del Catálogo de Datos) de la 2716 es exponerla a la luz ultravioleta de una longitud de onda, especialmente la de 2537 A. La dosis integrada (esto es, intensidad UV x tiempo de exposición) para el borrado debe ser como mínimo de 15 W-seg/cm². El tiempo de borrado para esta dosis es aproximadamente de 15 a 20 minutos utilizando una lámpara ultravioleta de régimen de potencia de 1200 μW/cm². La 2716 debe colocarse a menos de 2,5 cm de los tubos de la lámpara durante el borrado. Algunas lámparas poseen filtro en el tubo, que es preciso retirar antes del borrado.

FUNCIONAMIENTO DEL DISPOSITIVO

Los cinco modos de funcionamiento de la 2716 están enumerados en la tabla I. Nótese que todas las entradas para los cinco modos son niveles TTL. Las tres tensiones de alimentación son V_{CC} = +5 V y una V_{PP}. La fuente V_{PP} debe suministrar 25 V durante tres modos de programación y 5 V en los otros dos.

TABLA I. SELECCION DE MODOS

Patillas / Modo	CE/PGM (18)	OE (20)	V_{PP} (21)	V_{CC} (24)	Salidas (9 a 11, 13 a 17)
Lectura	V_IL	V_IL	+5	+5	Datos
Reserva	V_IH	Indiferente	+5	+5	Alta impedancia
Programa	Impulso de nivel bajo	V_IH	+25	+5	Entrada
Verificación del programa	V_IL	V_IL	+25	+5	Datos
Inhibición del programa	V_IH	V_IH	+25	+5	Alta impedancia

MODO DE LECTURA

La 2716 posee dos funciones de control, las cuales han de ser satisfechas para obtener los datos a su salida. Habilitación de chip (CE) es el control de alimentación y debe usarse para seleccionar la pastilla (chip). Habilitación de salida (OE) es el control de salida y debe usarse para seleccionar los datos en las pastillas de salida, independientemente de la selección de chip. Supuesto que las direcciones son estables, el tiempo de acceso a la dirección (t_{ACC}) es igual al retardo desde CE hasta la salida (t_{CE}). Los datos están disponibles a la salida t_{CE} − t_{OE} después del flanco de bajada de OE, suponiendo que CE era ya de nivel bajo y que las direcciones han permanecido estables durante un intervalo mínimo de t_{ACC} − t_{OE}.

MODO DE RESERVA

La 2716 posee un modo de reserva que reduce la disipación de energía activa en un 75 %, de 525 mW a 132 mW. La 2716 se pone en modo de reserva aplicando a su entrada CE un nivel TTL alto, sea cual sea el de OE. Cuando está en modo de reserva, las salidas se encuentran en el estado de alta impedancia, independientemente de la entrada OE.

SALIDA/INTERCONEXION OR

Como la 2716 se suelen utilizar en formaciones grandes de memoria, Intel ha previsto una función de control para 2 líneas que comprende este uso de conexiones a memorias múltiples. La función de control de dos líneas permite:

a) la disipación mínima posible de potencia en la memoria y

b) cerciorarse (lectura de que no habrá contención a la salida del bus.

Para el uso más eficaz de estas dos líneas de control, se recomienda que CE (patilla 18) se decodifique y use como función principal de selección de dispositivo, mientras que OE (patilla 20) se haga conexión común a todos los dispositivos del array y se conecte a la línea de lectura (READ) desde el bus de control del sistema. Ello asegura que todos los dispositivos de memoria seleccionados estén en el modo de reserva de consumo bajo y que las pastillas de salida estén solo activas cuando se desea extraer datos de un dispositivo de memoria determinado.

PROGRAMACION

Al comienzo, despues de cada borrado, todos los bits de la 2716 están en el estado «1». La información se introduce programando selectivamente «0» en las posiciones deseadas de bits. Aunque sólo se programan «ceros», ambos estados «1» y «0» pueden estar presentes en cada palabra de datos. La única manera de cambiar un «0» a un «1» es el borrado con luz ultravioleta.

La 2716 está en el modo de programación cuando la fuente de alimentación conectada a V_{PP} suministre 25 V y OE toma el valor V_{IH}. La información a programar se presenta de a 8 bits en paralelo en las patillas de salidas de datos. Los niveles necesarios para las direcciones y los datos de entrada son TTL.

Cuando la dirección y los datos se estabilizan se aplica un impulso TTL de programación de 50 mS, activo de nivel alto, a la entrada CE/PGM. Hay que aplicar un impulso de programación a cada una de las direcciones de las posiciones que se han a programar. Se puede programar cualquier dirección en cualquier momento, incluso individual, secuencial o aleatoriamente. El impulso de programación tendrá una anchura máxima de 55 mS. La 2716 no será programada aplicando una señal de nivel bajo a la entrada CE/PGM.

Se puede llevar a cabo fácilmente la programación múltiple de varias 2716 conectadas en paralelo, con la misma información dada la sencillez de los requisitos de programación. Las entradas iguales de las 2716 cone(t)adas en paralelo pueden unirse cuando se las programe con los mismos datos. La programación de todas las unidades 2716 conectadas en paralelo se consigue mediante un impulso de nivel alto TTL aplicado a las entradas CE/PGM.

INHIBICION DEL PROGRAMA

La programación de varias 2716 en paralelo con datos distintos se lleva a cabo combinando fácilmente. Con la excepción de CE/PGM, todas las entradas análogas (OE/inclusive) de las 2716 conectadas en paralelo pueden estar juntas. La 2716 que reciben en su entrada CE/PGM un impulso de programación de nivel TTL junto con 25 V en V_{PP} resulta programada. Las que reciben un nivel bajo en su entrada CE/PGM quedan inhibidas y no reciben los datos.

VERIFICACION DEL PROGRAMA

Debe efectuarse la comprobación de los bits programados para cerciorarse de que hayan sido estado lógico correcto. La verificación puede efectuarse con 25 V en V_{PP}. Excepto durante la programación y la verificación del programa, V_{PP} debe estar a 5 V.

CARACTERISTICAS EN C.A.

Símbolo	Parámetro	Límites (ns) 2716 Mín.	2716 Máx.	2716-1 Mín.	2716-1 Máx.	2716-2 Mín.	2716-2 Máx.	2716-5 Mín.	2716-5 Máx.	2716xx Mín.	2716xx Máx.	Condiciones de prueba
t_{ACC}	Retraso entre dirección y salida		450		350		390		450		450	$\overline{CE} = \overline{OE} = V_{IL}$
t_{CE}	Retraso entre $\overline{CE}$ y salida		450		350		390		490		650	$\overline{OE} = V_{IL}$
t_{OE}	Retraso entre habilitación de salida y salida		120		120		120		160		200	$\overline{CE} = V_{IL}$
t_{DF}	De nivel alto para la habilitación de salida a salida flotante	0	100	0	100	0	100	0	100	0	100	$\overline{CE} = V_{IL}$
t_{OH}	Retención de la salida desde las direcciones, $\overline{CE}$ u $\overline{OE}$, cualquiera que ocurra la primera	0		0		0		0		0		$\overline{CE} = \overline{OE} = V_{IL}$

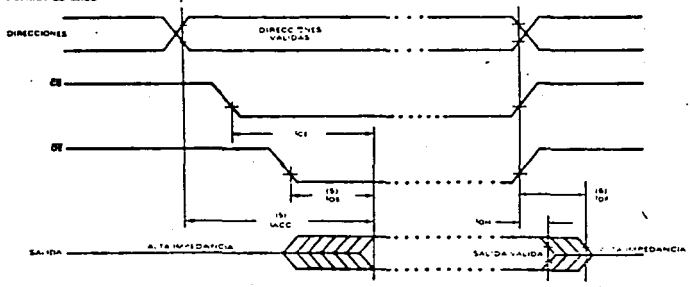
Capacitancia[a] $T_A = 25°C$, f = 1 MHz

Símbolo	Parámetro	Tip.	Máx.	Unidad	Condiciones
C_{IN}	Capacitancia de entrada	4	6	pF	$V_{IN} = 0V$
C_{OUT}	Capacitancia de salida	8	12	pF	$V_{OUT} = 0V$

Condiciones de prueba en c.a.

Carga de salida: 1 puerta TTL y una capacidad $C_L = 100$ pF.
Tiempos de subida y bajada para la entrada ≤ 20 ns.
Niveles del impulso de entrada: 0,8 a 2,2 V.
Nivel de referencia para la medida del tiempo
Entradas 1 y 2 V.
Salidas 0,8 y 2 V.

Formas de onda[1]



NOTAS
1. V_{CC} debe ser aplicado simultáneamente o antes que V_{PP} y retirado simultáneamente o después que V_{PP}.
2. Vpp puede conectarse directamente a V_{CC} excepto durante la programación. La corriente de alimentación máxima entonces la suma de I_{CC} e I_{PP}.
3. Los valores típicos son para $T_A = 25°C$ y tensiones de alimentación nominales.
4. Este parámetro se muestra, pero no se prueba al 100%.
5. $\overline{CE}$ puede estar retenida hasta $t_{ACC} - t_{OE}$ después de la dirección de CE sin afectar a t_{ACC}.
6. t_{DF} está determinada por $\overline{CE}$ o $\overline{OE}$, la que ocurra antes.

CDM6116A

TERMINAL ASSIGNMENT

CMOS 2048-Word by 8-Bit Static RAM

Features

- Fully static operation
- Single power supply: 4.5 V to 5.5 V
- All inputs and outputs directly TTL compatible
- 3 state outputs
- Industry standard 24 pin configuration
- Chip enable (per) and output buffer control
- Data retention voltage: 2 V min

	CDM6116A 2	CDM6116A 3	CDM6116A 5
Access time (max.)	200 ns	150 ns	120 ns
Cycle time (min.)	200 ns	150 ns	120 ns
Operating frequency	0 to 1 MHz		dc to 8 MHz
Operating current (max.)	15 mA	15 mA	40 mA
Standby current (max.)	10 µA	10 µA	100 µA

The RCA CDM6116A is a CMOS 2048-word by 8-bit static random access memory. It is designed for use in memory systems where high speed, low power and simple system interface are desired. The device has common data input and data output and utilizes a single power supply of +5 V ±10% V. A chip enable input and an output enable input are provided for memory expansion and output buffer control.

The chip enable (CE) controls the output buffers to eliminate bus contention.

The CDM6116A 2 and CDM6116A 3 have an operating temperature range of 0° to 70°C. The CDM6116A 5 has an operating temperature range of -40° to +85°C.

The CDM6116A 2 and CDM6116A 3 are designed with a fast read cycle time (access time) of 150 ns. The CDM6116A 5 is supplied in a 24 lead dual-in-line plastic package (E suffix) and a 24 lead dual-in-line side-brazed ceramic package (D suffix).

The chip enable (CE) gates the address and output buffers and powers down the chip to the low power standby mode.

Fig. 1 - Functional block diagram.

TRUTH TABLE

CE	OE	WE	A0 TO A10	MODE	I/O TO I/O8	DEVICE CURRENT
H	X	X	X	NOT SELECTED	HIGH Z	STANDBY
L	L	H	STABLE	READ	DATA OUT	ACTIVE
L	H	L	STABLE	WRITE	DATA IN	ACTIVE
L	L	L	STABLE	WRITE	DATA IN	ACTIVE

L = LOW, H = HIGH, X = don't care.

File Number 1472

CDM6116A

MAXIMUM RATINGS, Absolute-Maximum Ratings

SUPPLY-VOLTAGE RANGE, (VDD)

(Voltage referenced to VSS terminal) 0.5 to +7 V

INPUT VOLTAGE RANGE, ALL INPUTS -0.5 to +7 V

D.C. SUPPLY-CURRENT, (IDD) (Per Package) ±10 mA

POWER DISSIPATION PER PACKAGE (PD):

For TA = -40° to +85°C (PACKAGE TYPE E) 500 mW

For TA = 60° to 85°C (PACKAGE TYPE E) Derate linearly at 8 mW/°C to 300 mW

For TA = 40° to 85°C (PACKAGE TYPE D) 500 mW

POWER DISSIPATION PER OUTPUT TRANSISTOR (PT) 100 mW

FOR TA = FULL PACKAGE-TEMPERATURE RANGE and Package Type

STORAGE TEMPERATURE RANGE (Tstg) -65 to +150°C

LEAD TEMPERATURE (DURING SOLDERING):

CDM6116A 5 (PACKAGE TYPE E) 265°C max. for 10 s max.

CDM6116A 5 (PACKAGE TYPE D) 300°C max. for 10 s max.

OPERATING CONDITIONS at TA = 0 to +70°C (CDM6116A 2, CDM6116A 3); TA = -40° to +85°C (CDM6116A 5)

For maximum reliability, operating conditions should be selected so that operation is always within the following ranges:

CHARACTERISTIC		LIMITS ALL TYPES MIN.	LIMITS ALL TYPES MAX.	UNITS
Supply-Voltage Range	VDD	4.5	5.5	V
Input Voltage Range	VI	VSS - 0.5	VDD + 0.5	V
	VIH	0.8 VDD	VDD	V
Input Signal Rise or Fall Time	tr, tf	-	5	µs

Data retention: VDD = 2 V minimum, other inputs at VDD or VSS, CE at VDD, see note.

STATIC ELECTRICAL CHARACTERISTICS at TA = 0 to +70°C (CDM6116A 2, CDM6116A 3); TA = -40° to +85°C (CDM6116A 5); VSS = 0 V, VDD = 5 V ±10%, except as noted

CHARACTERISTIC	CONDITIONS	CDM6116A 2 MIN.	CDM6116A 2 TYP.*	CDM6116A 2 MAX.	CDM6116A 3 MIN.	CDM6116A 3 TYP.*	CDM6116A 3 MAX.	CDM6116A 5 MIN.	CDM6116A 5 TYP.*	CDM6116A 5 MAX.	UNITS
Standby Current	IDD1	-	0.5	-	-	0.5	-	-	0.5	2	mA
	IDD2	-	1	10	-	1	10	-	1	100	µA
Output Voltage High Level	VOH, IO = -1 mA	2.4	-	-	2.4	-	-	2.4	-	-	V
Output Voltage Low Level	VOL, IO = 2 mA	-	0.1	0.4	-	0.1	0.4	-	0.1	0.4	V
Input Leakage Current High Level	IIH, VIH = VDD	-	VDD 0.1	±1	-	VDD 0.1	±1	-	VDD 0.1	±1	µA
Input Leakage Current Low Level	IIL, VIL = VSS	-	±0.1	±1	-	±0.1	±1	-	±0.1	±1	µA
3-State Output Leakage Current	IOZ, CE = VIH or OE = VIH, VO = VSS or VDD	-	±0.5	±2	-	±0.5	±2	-	±0.5	±2	µA
Operating Current	IDD, VI = VSS or VDD	-	30	35	-	30	35	-	30	40	mA
Input Capacitance	CI, VI = VSS, f = 1 MHz, TA = 25°C	-	5	8	-	5	8	-	5	8	pF
Output Capacitance	CO, VO = VSS, f = 1 MHz, TA = 25°C	-	5	8	-	5	8	-	5	8	pF

*Typical values are for TA = 25°C and nominal supply voltage. †Output open (all outputs). Cycle time = Read cycle, duty = 100%.

CDM6116A

SIGNAL DESCRIPTIONS

A0-A10 (Address Inputs) These inputs must be stable prior to a write operation but may change during read operations.

I/O1-I/O8 8 bit bidirectional data bus.

$\overline{CE}$ (Chip Enable) Powers down chip, disables read and write functions and gates off address circuits.

$\overline{OE}$ (Output Enable) Enables tri-state outputs if $\overline{CE}$ is low and $\overline{WE}$ is high.

$\overline{WE}$ (Write Enable) Enables write function if $\overline{CE}$ is low and $\overline{WE}$ and $\overline{OE}$ are low. All inputs are then disabled for retention and during write cycle.

V_{SS}, V_{DD} Power supply connections.

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = 0 to 70°C (CDM6116A 2, CDM6116A 3), T_A = -40° to +85°C (CDM6116A 5), V_{DD} = 5 V ± 10%, Input t_r, t_f = 10 ns, C_L = 100 pF and 1 TTL Load, Input Pulse Levels: 0.8 V to 2.4 V

CHARACTERISTIC		LIMITS						UNITS
		CDM6116A 2		CDM6116A 3		CDM6116A 5		
Read Cycle Times See Fig. 3		MIN	MAX	MIN	MAX	MIN	MAX	
Read Cycle Time	t_{RC}	150		150		250		ns
Address Access Time	t_{AA}		150		150		250	
Chip Enable Access Time	t_{ACE}		150		150		250	
Chip Enable to Output Active	t_{CLZ}	10		10		10		
Output Enable to Output Active	t_{OLZ}	0		0		0		
Output Enable to Output Valid	t_{OLQV}		70		70		100	
Chip Enable to Output High Z	t_{CHZ}	0	60	0	60	0	80	
Output Disable to Output High Z	t_{OHZ}	0	60	0	60	0	80	
Output Hold from Address Change	t_{OH}	10		10		10		

*Time required by a read cycle for the next write cycle.

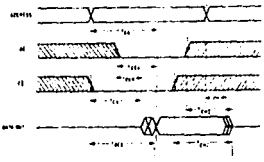


Fig. 2 - Read cycle timing waveforms

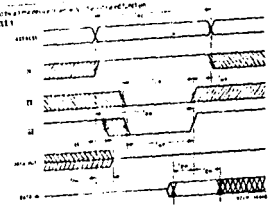
CDM6116A

DYNAMIC ELECTRICAL CHARACTERISTICS at T_A = 0 to 70°C (CDM6116A 2, CDM6116A 3), T_A = -40° to +85°C (CDM6116A 5), V_{DD} = 5 V ± 10%, Input t_r, t_f = 10 ns, C_L = 100 pF and 1 TTL Load, Input Pulse Levels: 0.8 V to 2.4 V

CHARACTERISTIC		LIMITS						UNITS
		CDM6116A 2		CDM6116A 3		CDM6116A 5		
Write Cycle Times See Fig. 3		MIN	MAX	MIN	MAX	MIN	MAX	
Write Cycle Time	t_{WC}	150		150		250		ns
Write Pulse Width	t_{WP}	90		90		150		
Chip Enable to End of Write	t_{CW}	150		150		200		
Address Valid to End of Write	t_{AW}	150		150		200		
Address Setup Time	t_{AS}	0		0		0		
Write Recovery Time	t_{WR}	10		10		10		
Write Pulse Width	t_{WP}	70		70		100		
Data Setup Time	t_{DS}	70		70		100		
Data Hold Time	t_{DH}	10		10		10		
Write Enable to Output High Z	t_{WHZ}	0	60	0	60	0	80	
Data Valid to Output Active	t_{DOW}	60		60		100		
Output Disable Setup Time	t_{OES}	10		10		10		
Output Disable Hold Time	t_{OEH}	10		10		10		
Output Disable to Output High Z	t_{OHZ}	0	60	0	60	0	80	

*Time required by a write cycle for the next write cycle.

WRITE CYCLE 1



WRITE CYCLE 2 - $\overline{OE}$ LOW

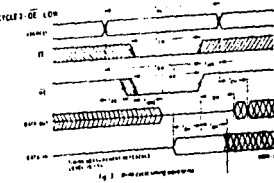


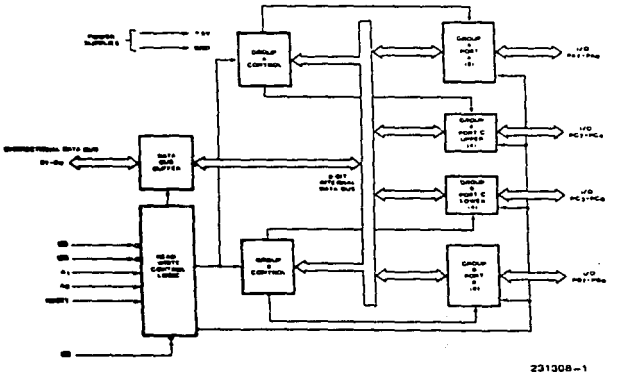
Fig. 3 - Write cycle timing waveforms

intel

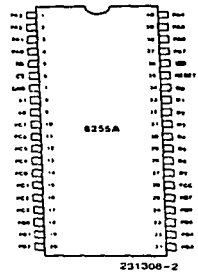
8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- **MCS-85™ Compatible 8255A-5**
- **24 Programmable I/O Pins**
- **Completely TTL Compatible**
- **Fully Compatible with Intel Microprocessor Families**
- **Improved Timing Characteristics**
- **Direct Bit Set/Reset Capability Easing Control Application Interface**
- **Reduces System Package Count**
- **Improved DC Driving Capability**
- **Available in EXPRESS**
  - **— Standard Temperature Range**
  - **— Extended Temperature Range**
- **40 Pin DIP Package or 44 Lead PLCC**
  (See Intel Packaging: Order Number: 231369)

The Intel 8255A is a general purpose programmable I/O device designed for use with Intel microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.



231308-1



231308-2

Figure 2. Pin Configuration

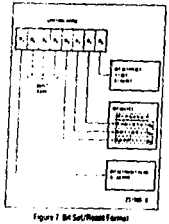


Figure 7. Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set-Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be enabled or disabled by setting or resetting the associated INTE flip-flop, using the bit set-reset function of port C.

The function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT SET)- INTE is set—interrupt enable

(BIT RESET)- INTE is RESET—interrupt disable

NOTE:
All Mask flip-flops are automatically reset during mode selection and device Reset.

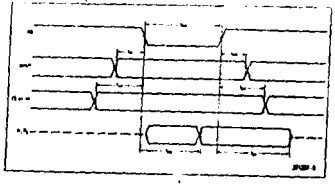
Operating Modes

MODE 0 (Basic Input/Output): This functional configuration provides simple input and output operations for each of the three ports. No handshaking is required, data is simply written to or read from a specified port.

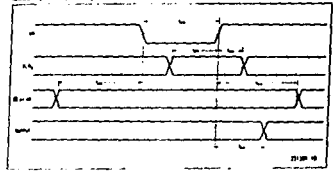
Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

MODE 0 (BASIC INPUT)



MODE 0 (BASIC OUTPUT)



MODE 0 PORT DEFINITION

A		B		Group A			Group B	
D_4	D_3	D_1	D_0	Port A	Port C (upper)	#	Port B	Port C (lower)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

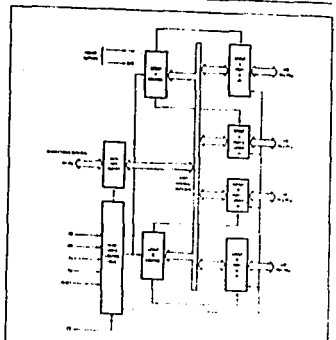
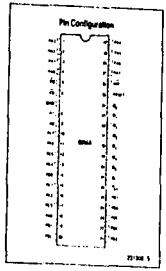


Figure 4. 8225A Block Diagram Showing Group A and Group B Control Functions



Pin Names	
D_7-D_0	Data Bus (Bi-Directional)
RESET	Reset Input
$\overline{CS}$	Chip Select
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A0, A1	Port Address
PA7-PA0	Port A (BIT)
PB7-PB0	Port B (BIT)
PC7-PC0	Port C (BIT)
V_{CC}	+5 Volts
GND	0 Volts

8255A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 – Basic Input/Output

Mode 1 – Strobed Input/Output

Mode 2 – Bi-Directional Bus

When the reset input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the reset is removed the 8255A can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single output instruction. This allows a single 8255A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be tailored to almost any I/O structure. For instance Group B can be programmed in Mode 0 to monitor simple switch closings or display computed output results. Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt driven basis.

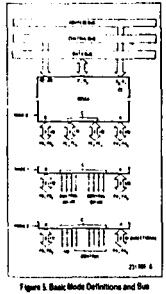


Figure 5. Basic Mode Definitions and Bus Interface

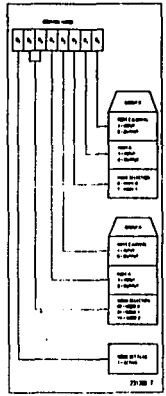


Figure 6. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device specification a simple, logical I/O approach will surface. The design of the 8255A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control based applications.

8255A FUNCTIONAL DESCRIPTION

General

The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3 state bidirectional 8 bit buffer is used to interface the 8255A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select. A "low" on this input pin enables the communication between the 8255A and the CPU.

(RD)

Read. A "low" on this input pin enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255A.

(WR)

Write. A "low" on this input pin enables the CPU to write data or control words into the 8255A.

(A_0 and A_1)

Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1).

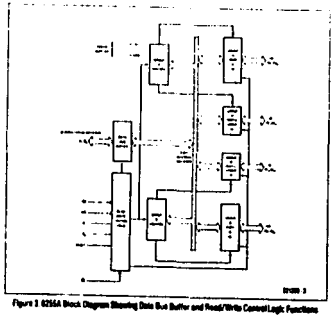


Figure 3. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions

8255A BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input Operation (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
					Output Operation (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
					Disable Function
X	X	X	X	1	Data Bus → 3 State
1	1	0	1	0	Illegal Condition
X	X	1	1	0	Data Bus → 3 State

(RESET)

Reset. A "high" on this input clears the control register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU outputs a control word to the 8255A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A – Port A and Port C upper (C7-C4)

Control Group B – Port B and Port C lower (C3-C0)

The Control Word Register can only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

The 8255A contains three 8 bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255A.

Port A: One 8 bit data output latch/buffer and one 8 bit data input latch.

Port B: One 8 bit data input/output latch/buffer and one 8 bit data input buffer.

Port C: One 8 bit data output latch/buffer and one 8 bit data input buffer (no latch for input). This port can be divided into two 4 bit ports under the mode control. Each 4 bit port contains a 4 bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

8279/8279-5
PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- Simultaneous Keyboard Display Operations
- Scanned Keyboard Mode
- Scanned Sensor Mode
- Strobed Input Entry Mode
- 8-Character Keyboard FIFO
- 2-Key Lockout or M-Key Rollover with Contact Debounce
- Dual 8- or 16-Numerical Display
- Single 16-Character Display
- Right or Left Entry 16-Byte Display RAM
- Mode Programmable from CPU
- Programmable Scan Timing
- Interrupt Output on Key Entry
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64 contact key matrix. The keyboard portion will also provide a right entry array of sensors or a strobed interface for keyboard such as the hall effect and bounce free. Key depressions can be 2 key lockout or M-key rollover. Keyboard entries are debounced and stored in an 8 character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16x8 display RAM which can be organized into dual 16x4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto increment of the display RAM address.

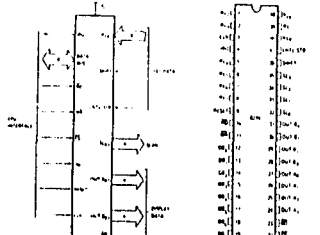


Figure 1 Logic Symbol

Figure 2 Pin Configuration

September 1987
Order Number: 280133-002

HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

Table 1 Pin Descriptions

Symbol	Pin No.	Name and Function
DB_0-DB_7	19-12	**BI DIRECTIONAL DATA BUS.** All data and commands between the CPU and the 8279 are transmitted on these lines.
CLK	3	**CLOCK.** Used by the internal timing.
RESET	9	**RESET.** A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8 bit character display - left entry 2) Encoded scan keyboard - 2 key lockout Along with this the program clock prescaler is set to 31.
$\overline{CS}$	22	**CHIP SELECT.** A low on this pin enables the internal functions. In the case of a transmit as a command or status. A low will also place the data in or out of the data buffers.
A_0	21	**BUFFER ADDRESS.** A high on this line indicates the signals in or out are a command or a command or status. A low indicates that they are data.
$\overline{RD}$ $\overline{WR}$	10, 11	**INPUT/OUTPUT READ AND WRITE.** These signals enable the data buffers to either send data to the internal bus or receive it from the internal bus.
IRQ	4	**INTERRUPT REQUEST.** In a keyboard mode this line is kept high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor matrix mode, the interrupt line goes high when a change in a sensor is detected.
V_{SS} V_{CC}	20, 40	**GROUND AND POWER SUPPLY PINS.**
SL_0-SL_3	32-35	**SCAN LINES.** Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).
R_0-R_7	39, 38, 1, 2, 5, 8	**RETURN LINE.** These return inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8 bit input in the Strobed Input mode.
SHIFT	36	**SHIFT.** The shift input status is stored along with the key position on key closure in the Scanned keyboard modes. It has an active internal pullup to keep it high until a switch closure pulls it low.
CNTL/STB	37	**CONTROL/STROBE INPUT MODE.** For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed Input mode (Rising Edge). It has an active internal pullup to keep it high until a switch closure pulls it low.
OUT A_0, OUT A_3 OUT B_0, OUT B_3	27,24 31-28	**OUTPUTS.** These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL_0-SL_3) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8 bit port.
$\overline{BD}$	23	**BLANK DISPLAY.** This output is used to blank the display during digit switching or by a program blanking command.

FUNCTIONAL DESCRIPTION

Simultaneous keyboard and display operations are an integral part of many microprocessor designs. The system designer usually develops routines that continually service these functions without causing a large load on the CPU. The 8279 provides the hardware for these operations.

The 8279 has two basic sections: keyboard and display. The keyboard section has an 8-character FIFO memory for storing up to 64 keystrokes without losing data. Special scanned keyboard mode automatically debounces one key rollover or simultaneous depression of two keys. Any key depression can also be programmed to generate an interrupt.

The display section scans the display and can be programmed for either left or right entry. Both read and write operations are available on the display RAM. Transparent write operations allow only the addressed display RAM location to be blanked for minimum display interference.

The 8279 can be programmed by the CPU to accept a variety of keyboard and display formats:

- 8- or 16-character displays
- 8- or 16-key keyboard
- Dual 8- or 16-key keyboard
- 8 by 8 matrix keyboard

The 8279 provides two 8-bit bidirectional ports which may be used for expanding the number of display characters or keyboard sensors. Additional features include:

- Programmable scan timing
- Interrupt output on key entry
- Two-key lockout or N-key rollover
- Special error modes
- 128-byte internal display RAM
- Programmable output configurations

PRINCIPLES OF OPERATION

The following is a description of the major elements of the 8279 Programmable Keyboard/Display Interface device. Refer to the block diagram in Figure 3.

I/O Control and Data Buffers

The I/O control section uses the $\overline{CS}$, A_0, $\overline{RD}$ and $\overline{WR}$ inputs to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by $\overline{CS}$. The character of the information given or received by the CPU is identified by A_0. A logic one denotes the information is a command or status. A logic zero denotes the information is data. $\overline{WR}$ and $\overline{RD}$ determine the direction of data flow through the Data Buffers. The Data Buffers are bidirectional buffers that connect the internal bus to the external bus. When the chip is not selected ($\overline{CS}$ = 1) the drivers are in a high impedance state. Flow diagrams appear during both read and write operations.

Control and Timing Registers and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. The modes are programmed by commands issued by the processor. Any command may be loaded into the control registers at any time and is used by other sections within the 8279 to establish their functional configuration. Status words may be read to the CPU upon request.

The timing chain counters are programmable to yield any desired internal frequency. The first counter is a divide-by-N counter which can be programmed to divide down the external frequency. The remaining counters divide down the internal frequency to provide the proper key scan time and debounce time.

Scan Counter

The scan counter has two modes. In the encoded mode the counter provides a binary count that must be externally decoded to provide the scan lines for the keyboard and display. In the decoded mode, the two counter bits are decoded to provide one-of-four scan lines. The scan lines are active high outputs. In the decoded mode, the four scan lines are active low outputs.

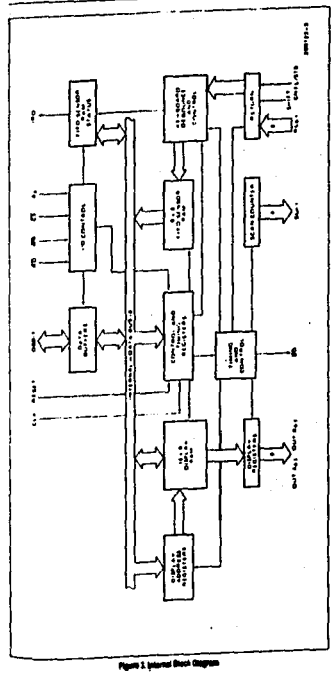


Figure 3. Internal Block Diagram

Return Buffers and Keyboard Debounce and Control

The 8 return lines are buffered and latched by the Return Buffers. In the keyboard mode, these lines are scanned, looking for key closures. In that mode, the debounce circuit detects a closed switch. It waits about 10 msec to check if the switch remains closed. If it does, the address of the switch in the matrix, plus the status of Shift (Y) and Control (Z), are transferred to the FIFO. In the N-key Rollover Sensor Matrix mode, the contents of the return lines is directly transferred to the corresponding row of the Sensor RAM if a Change in any of the return lines is detected. In the Strobed Input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CN1/SL3 line pulse.

FIFO/Sensor RAM and Status

The block is a dual function 8 x 8 RAM. In keyboard or Strobed Input modes, it is a FIFO. In N-key rollover or sensor matrix type, it is the RAM which contains and reflects the current status of each switch in the matrix. The RAM can be read by an FIFO read (CS low and A0 high). The status logic and pointers are also affected when the FIFO is read. In N-key and Sensor Matrix modes, the contents of the Sensor RAM row in the Sensor RAM is loaded into the status fifo when the Sensor RAM is loaded with the status of the corresponding row of switches in the sensor matrix. In this mode, IRQ is high if a change in a sensor is detected.

Display Address Registers and Display RAM

The Display Address Registers hold the address of the word currently being written to or read by the CPU and the two 4 bit nibbles being displayed. The read and/or write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The Display RAM can be over-written by the CPU after the correct mode and address are set. The addresses for the A and B nibbles are also made available to the CPU. A read to data entry by the CPU. The A and B nibbles can be updated independently, or at one word according to the mode that is set by the CPU. Data entry to the display can be set to either left or right entry. Two nibble or 1 character blank on display.

SOFTWARE OPERATION

8279 Commands

The following commands program the 8279 operating modes. The commands are sent on the Data Bus with CS low and A_0 high and are loaded to the 8279 on the rising edge of WR.

Keyboard/Display Mode Set

MSB LSB

Code: | 0 | 0 | 0 | D | D | K | K | K |

Where DD is the Display Mode and KKK is the Keyboard Mode

DD

0 0 8 8 bit character display - Left entry

0 1 16 8 bit character display - Left entry*

1 0 8 8 bit character display - Right entry

1 1 16 8 bit character display - Right entry

For description of right and left entry, see Interface Considerations. Note that when the scan is set in strobed mode the display is reset and to 8 character independent of display mode set.

KKK

0 0 0 Encoded Scan Keyboard - 2 Key Lock Out*

0 0 1 Decoded Scan Keyboard - 2 Key Lock Out

0 1 0 Encoded Scan Keyboard - N Key Roll Over

0 1 1 Decoded Scan Keyboard - N Key Roll Over

1 0 0 Encoded Scan Sensor Matrix

1 0 1 Decoded Scan Sensor Matrix

1 1 0 Strobed Input, Encoded Display Scan

1 1 1 Strobed Input, Decoded Display Scan

*Power on reset

Program Clock

Code: | 0 | 0 | 1 | P | P | P | P | P |

All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock (pin 3) by a programmable integer. Bits PPPPP determine the value of this integer which ranges from 2 to 31. Choosing a divisor that yields 100 kHz will give the specified scan and debounce times. For instance, a 4 MHz signal on pin 3 of the 8279 is being clocked by a 2 MHz signal. PPPPP should be set to 10100 to divide the clock by 20 to yield the proper 100 kHz operating frequency.

Read FIFO/Sensor RAM

Code: | 0 | 1 | AI | X | A | A | A | A | X = Don't Care

The CPU sets the 8279 into a read of the FIFO/Sensor RAM by first writing this command. In the Scan Keyboard Mode, the AI auto increment flag (AI) and the RAM address bits (AAAA) are irrelevant. The 8279 will automatically drive the data first for each key of queue and (A_0) = 0 in the same sequence in which the data first entered the FIFO. All subsequent reads will be from the FIFO until another command is issued.

In the Sensor Matrix Mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM if the AI flag is set (AI = 1) each successive read will be from the subsequent row of the Sensor RAM.

Read Display RAM

Code: | 0 | 1 | AI | A | A | A | A | A |

The CPU sets up the 8279 for a read of the Display RAM by first writing this command. The address bits AAAA will be of the 16 rows of the Display RAM if the AI flag is set (AI = 1) the row address will be incremented after each following read. In order to the Display RAM. Since the same command is used for both reading and writing, this command sets the next read or write address and the sense of the AUTO increment mode for both operations.

Write Display RAM

Code: | 1 | 0 | AI | A | A | A | A | A |

The CPU sets up the 8279 for a write to the Display RAM by first writing this command. After writing the command with A_0 = 1 all subsequent writes with A_0 = 0 will be to the Display RAM. The addressing and AUTO increment functions are identical to those for the Read Display RAM. However, this command does not affect the source of subsequent Data Reads. The CPU will read from while from RAM (Data Entry of FIFO/Sensor) which was last specified. If, however, the Display RAM was last specified, the Write Display RAM will nevertheless change the next Read location.

Display Write Inhibit/Blanking

	A	B	A	B
Code	1	0	1	IW	IW	BL	BL

The IW Bits can be used to mask nibble A and nibble B in applications requiring separate 4 bit display ports. By setting the IW flag (IW = 1) for one of the ports, the port becomes masked so that entries to the Display RAM from the CPU do not affect that port. Thus, if each nibble is input to a BCD decoder, the CPU may write a digit to the Display RAM without affecting the other digit being displayed. It is important to note that bit B_0 corresponds to bit D_0 on the CPU bus, and that bit A corresponds to bit D_4.

If the user wishes to blank the display, the BL flags are available for each nibble. The last Clear command issued determines the order to be used as a "blank." The code defaults to all zeros after a reset. Note that both BL flags must be set to blank a display formatted with a single 8 bit port.

Clear

Code: | 1 | 1 | 0 | C_D | C_D | C_D | C_A |

The C_D bits are available in the command to clear all rows of the Display RAM to a selectable blanking code as follows:

C_D C_D C_D	
0 0 0	All Rows = Zero Code
0 0 1	All Rows = Hex 20 in ASCII
0 1 X	All Rows = 0
1 X X	All Rows = 1

C_A = 0 Enables Clear Display

C_A = 1 Enables Clear All

20019-10

During the time the Display RAM is being cleared (~160 µs) it may not be written to. The most significant bit of the FIFO status word is set during this time. When the Display RAM becomes available again, it automatically resets.

If the C_F bit is asserted (C_F = 1) the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

C_A, the Clear All bit, has the combined effect of C_D and C_F. It clears the C_D clearing mode on the Display RAM and also clears FIFO status. Furthermore, it resets however, the internal timing chain.

End Interrupt/Error Mode Set

[illegible] Data bus

[illegible] sensor matrix modes, they command lowers the scan line and enables further writing into RAM. (FIFO has already been cleared upon the detection of a change or a sensor error). This would [illegible] disabled further writing into the RAM until new [illegible]

[illegible] the N-key rollover mode, if the [illegible] to give one character in the FIFO, the flag will signal all the same [illegible] key is held down (N-key rollover) [illegible]

Status Word

The status word is sent to the CPU when read command is received. When [illegible] the CPU address line (A0) is high and (RD) and (CS) are low. Status is updated [illegible] to read the status word.

Data Read

During a read cycle, A0 is high and RD and CS are low. The selected FIFO data is sent to the CPU via the internal bidirectional bus buffer. A pulse on the (RD) input increments the address of the RAM being addressed. The [illegible] is reset to the address of the first word in the FIFO. If auto-increment is not programmed, the address will remain [illegible]

Data Write

Data is written into the RAM starting at the address specified by the Write RAM command. The address is pre-incremented after each write so that successive writes will load successive locations. The auto-increment feature allows the CPU to write into consecutive RAM locations without having to issue a new command for each location. The display will not be blanked when the CPU writes to the display RAM, to avoid annoying blinking. However, data written into the display RAM will not be displayed until the next display cycle for that word.

INTERFACE CONSIDERATIONS

Scanned Keyboard Mode, 2-Key Lockout

[illegible] are three possible combinations of conditions that can occur during debounce scanning. When a key is depressed, the debounce logic is set. Other depressed keys are locked out during the next two scans. If there are unprocessed 4 N-key single key depressions and the key condition is entered into the [illegible] A keyboard mode can only indicate a switched closure. To make the system more robust, the designer should both formally group the sensors by row and column, and the format in which the CPU will read them.

The debounce circuit ignores any switch status changes as given not at the end of 4 sensor matrix scan. The [illegible] is activated by the first data read operation of the Auto-Increment flag is set to zero or by the End Interrupt command if the Auto-increment flag is set to one.

NOTE

In regular changes on the matrix. Assumed by [illegible] Note: [illegible] multiple changes. Only [illegible] in the two-word Mode 3 Format may cause the [illegible] two multiple changes.

Data Format

In the Scanned Keyboard mode, the characters are stored in the FIFO corresponding to the pressed keys of the switch in the keyboard plus the status of the Shift and Control keys (encountered). Shift is the MSB of the character and Scan 0 is the next most significant. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate in which column the key was encountered.

MSB LSB

[illegible] N KEY CODE

N_KEY ROLLOVER DATA FORMAT

In Sensor Matrix mode, the data in the return lines is sampled one by one at the end of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each bank in sensor matrix mode directly to a Sensor RAM position. The Shift 1 and Shift 2 inputs are ignored in this mode. Note that these keys are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the is an input can enter data to the return line inputs. Eight multiplexed input ports could be used to the return lines and scanned by the 8-79.

MSB LSB

[illegible]

In Sensor Input mode, the data is also entered to the FIFO from the return lines. The data is entered by the rising edge of a CNTL/STB line pulse. Data can come from another encoded keyboard or simple switch matrix. The return lines can also be used as a general purpose strobed input

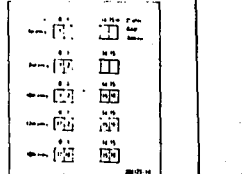
MSB LSB

[illegible]

Display

Left Entry

Left Entry mode is the simplest display format in that each display position directly corresponds to a byte (or nibble) in the Display RAM. Address 0 in the RAM is the left most display character and address 15 (or address 7 in 8 character displays) is the right most display character. Entering characters from position zero causes the display to fill from the left. The 16th (8th) character is entered back at the left most position and filling again proceeds from there.



Right Entry

Right entry is the method used by most calculator calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost

Scanned Keyboard Mode, N Key Rollover

[illegible] Keyboard mode, the character [illegible] are stored in the FIFO, corresponding to the pressed keys of the switch in the keyboard plus the status of the Shift and Control keys (encountered). Shift is the MSB of the character and Scan 0 is the next most significant. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate in which column the key was encountered.

MSB LSB

[illegible] N KEY CODE

N_KEY ROLLOVER DATA FORMAT

In Sensor Matrix mode, the data in the return lines is sampled one by one at the end of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each bank in sensor matrix mode directly to a Sensor RAM position. The Shift 1 and Shift 2 inputs are ignored in this mode. Note that these keys are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the is an input can enter data to the return line inputs. Eight multiplexed input ports could be used to the return lines and scanned by the 8-79.

MSB LSB

[illegible]

In Sensor Input mode, the data is also entered to the FIFO from the return lines. The data is entered

Scanned Keyboard Mode, N Key Rollover

[illegible] Keyboard mode, the character [illegible] are stored in the FIFO, corresponding to the pressed keys of the switch in the keyboard plus the status of the Shift and Control keys (encountered). Shift is the MSB of the character and Scan 0 is the next most significant. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate in which column the key was encountered.

MSB LSB

[illegible] N KEY CODE

N_KEY ROLLOVER DATA FORMAT

In Sensor Matrix mode, the data in the return lines is sampled one by one at the end of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each bank in sensor matrix mode directly to a Sensor RAM position. The Shift 1 and Shift 2 inputs are ignored in this mode. Note that these keys are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the is an input can enter data to the return line inputs. Eight multiplexed input ports could be used to the return lines and scanned by the 8-79.

MSB LSB

[illegible]

In Sensor Input mode, the data is also entered to the FIFO from the return lines. The data is entered

Sensor Matrix Mode

[illegible] Sensor Matrix mode, the debounce logic is omitted. The state of the sensor switch is inputted directly to the Sensor RAM. In this way, the Sensor RAM keeps an image of the state of the keys that is [illegible] by the scan register. Although debouncing is not provided, this mode has the advantage that the CPU knows how long the sensor switch closed and when it

by the rising edge of a CNTL/STB line pulse. Data can come from another encoded keyboard or simple switch matrix. The return lines can also be used as a general purpose strobed input

MSB LSB

[illegible]

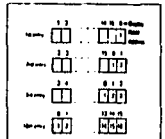
Display

Left Entry

Left Entry mode is the simplest display format in that each display position directly corresponds to a byte (or nibble) in the Display RAM. Address 0 in the RAM is the left most display character and address 15 (or address 7 in 8 character displays) is the right most display character. Entering characters from position zero causes the display to fill from the left. The 17th (9th) character is entered back at the left most position and filling again proceeds from there.

Right Entry

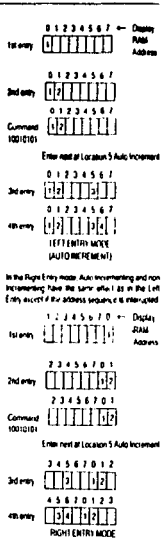
Right entry is the method used by most calculator calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost



Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM address 0 with sequential entry is recommended.

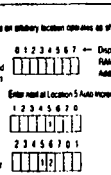
Auto Increment

In the Left Entry mode, Auto incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Auto Increment mode has no undesirable side effects and the result is predictable.



In the Right Entry mode, Auto incrementing and non incrementing have the same effect as in the Left Entry except if the address sequence is interrupted.

Starting at an arbitrary location operates as shown below:



Entry appears to be from the empty entry point.

8/16 Character Display Formats

If the display mode is set to an 8 character display the on duty-cycle is double what it would be for a 16 character display (e.g., 5.1 ms scan time for 8 characters vs. 10.2 ms for 16 characters with 100 kHz internal frequency).

G. FIFO Status

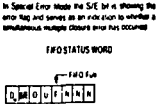
The G status is used in the Keyboard and Strobed Input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. There are two types of errors possible, overrun and underrun. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

The FIFO status word also has a bit to indicate that the Display RAM was unreadable because a Clear Display or Clear All command had not completed its clearing operation.

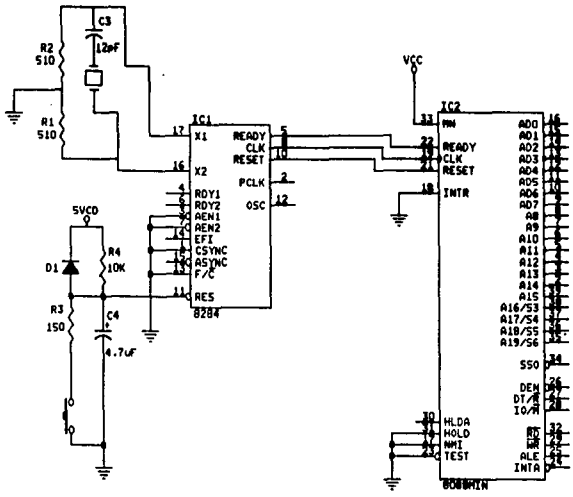
In a Sensor Matrix mode, a bit is set in the FIFO status word to indicate that at least one sensor closure indication is contained in the Sensor RAM.

In Special Error Mode the S/E bit is showing the error flag and serves as an indication to whether a simultaneous multiple closure error has occurred.

FIFO STATUS WORD



DIAGRAMAS ELECTRICOS



Title

CIRCUITO DE RELOJ

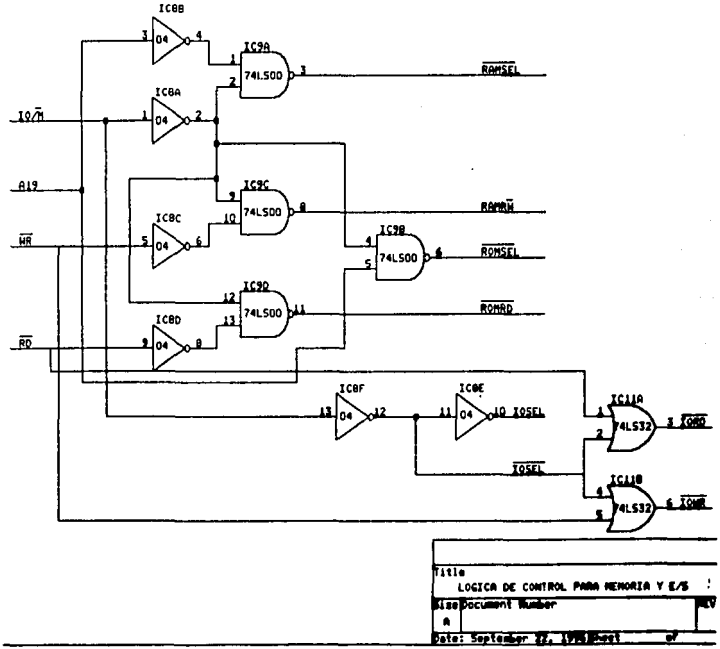
Size Document Number

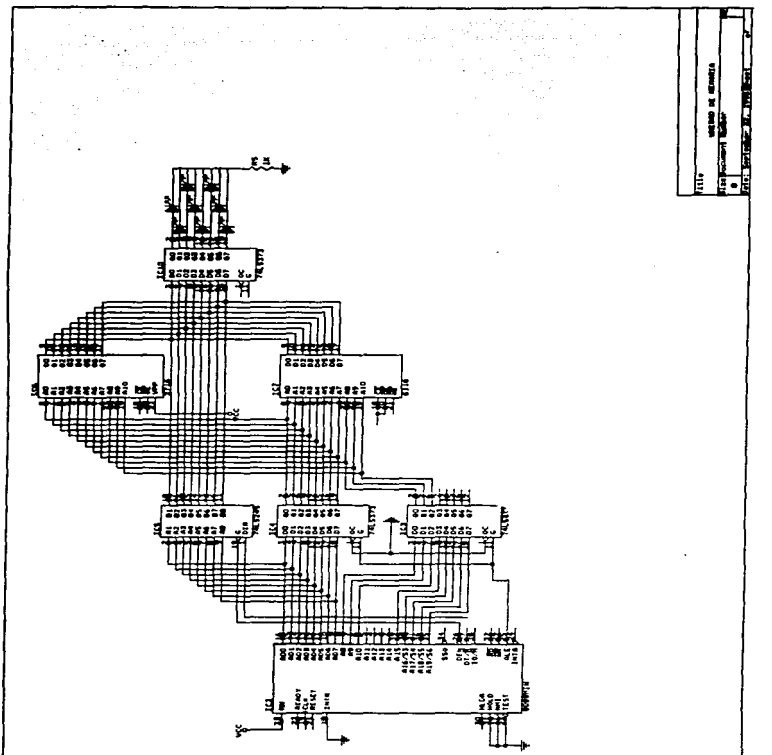
A

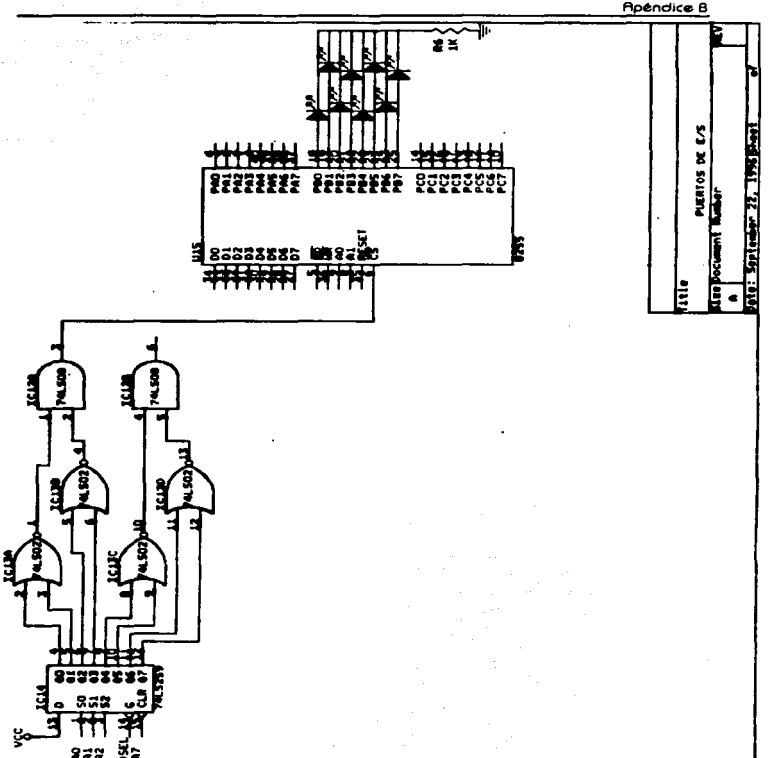
Date: September 20, 199 Sheet of

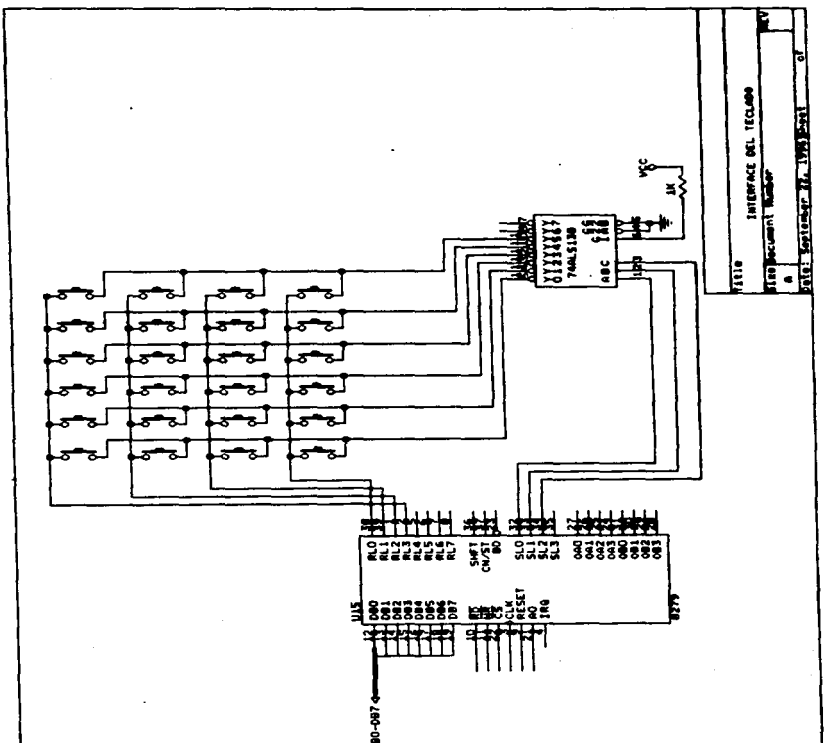
Apéndice B

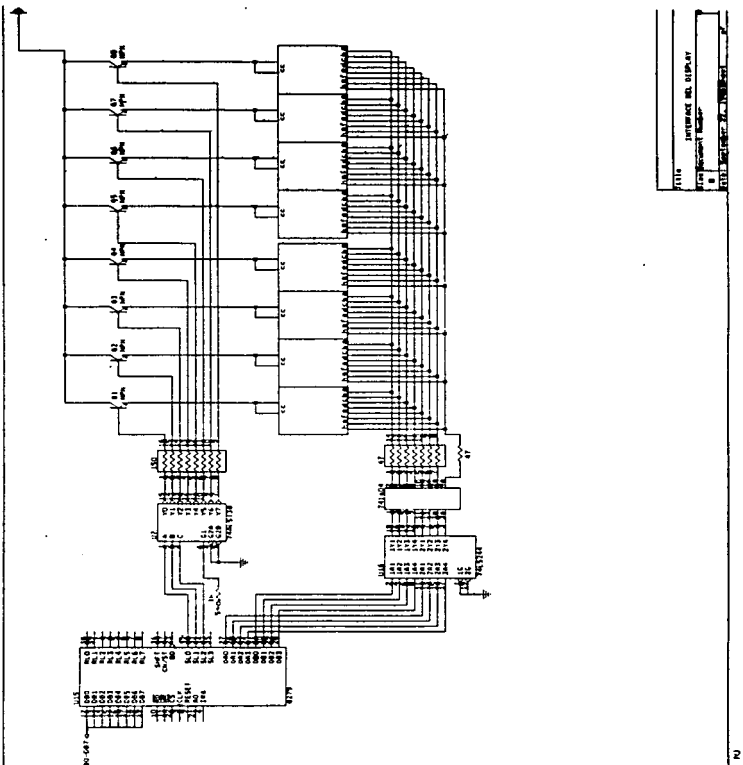
210

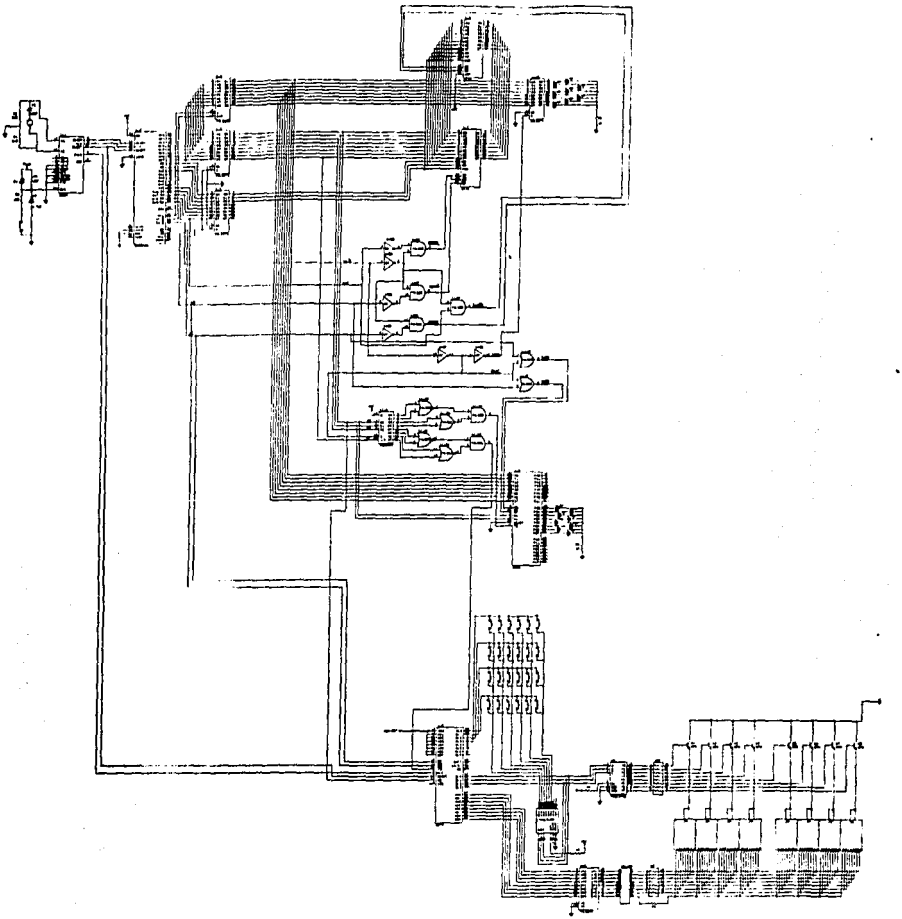

IC8B
04
IC9A
74LS00
RAMSEL
IC8A
04
IO/M
A19
IC9C
74LS00
RAMRW
IC8C
04
WR
IC9B
74LS00
ROMSEL
IC9D
74LS00
ROMRD
IC8D
04
RD
IC8F
04
IC8E
04
IO IOSEL
IOSEL
IC11A
74LS32
IORD
IC11B
74LS32
IOWR
Title
LOGICA DE CONTROL PARA MEMORIA Y E/S
Size
A
Document Number
REV
Date: September 22, 1988
Sheet
of




R6
1K
VDD
U15
8255
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
D0
D1
D2
D3
D4
D5
D6
D7
A0
A1
RESET
CS
IC14
74LS139
IC14A
74LS08
IC14B
74LS08
IC13A
74LS02
IC13B
74LS02
IC13C
74LS02
IC13D
74LS02
VCC
A0
A1
IOSEL
A7
Title
PUERTOS DE E/S
Size
A
Document Number
REV
Date: September 28, 1995
Sheet
of


INTERFACE DEL TECLADO
Title
Number
A
Date: Saturday, May 19, 1990
Sheet
of
Vcc
1K
74ALS138
8279
RL0
RL1
RL2
RL3
RL4
RL5
RL6
RL7
SHIFT
CNTL/STB
SL0
SL1
SL2
SL3
OUT0
OUT1
OUT2
OUT3
OUT0
OUT1
OUT2
OUT3
IRQ
RESET
CLK
CS
A0
RD
WR
D0
D1
D2
D3
D4
D5
D6
D7
D0-D7





JUEGO DE INSTRUCCIONES

1) TRANSFERENCIA DE DATOS

A continuación se muestra las instrucciones de transferencia de datos:

CODIGO DE OPERACION	FUNCION
MOV	Mueve un byte o una palabra.
PUSH	Coloca una palabra en la pila.
PUSHF	Coloca las banderas en la pila.
LAHF	Mueve las banderas al registro AH.
SHAF	Carga las banderas del registro AH.
POP	Saca una palabra de la pila.
POPF	Saca las banderas de la pila.
IN	Lleva datos de un dispositivo de entrada al acumulador.
OUT	Saca datos del acumulador a un dispositivo de salida
XCHG	Intercambia bytes o palabras.
XLAT	Traslada un byte de la tabla de búsqueda al registro AL.
LEA	Carga la dirección efectiva.
LDS	Carga el registro DX y el operando con una dirección de 32 bits
LES	Carga el registro EX y el operando con una dirección de 32 bits

2) ARITMETICAS

CODIGO DE OPERACION	FUNCION
ADD	Suma bytes o palabras.
ADC	Suma bytes o palabras más acarreo.
AAA	Ajusta a ASCII después de la suma.
DAA	Ajusta a BCD después de la suma.
INC	Suma 1 a un byte o palabra.
SUB	Resta bytes o palabras.
SBB	Resta bytes o palabras menos acarreo.
NEG	Cambia el signo de bytes o palabras (complemento a dos).
CMP	Compara bytes o palabras.
AAS	Ajusta a ASCII la resta.
DAS	Ajusta a BCD la resta.
DEC	Resta 1 a un byte o palabra.
MUL	Multiplica bytes o palabras sin signo.
IMUL	Multiplica bytes o palabras con signo.
AAM	Ajusta a ASCII después de la multiplicación.
DIV	Divide bytes o palabras sin signo.
IDIV	Divide bytes o palabras con signo.
CBW	Convierte bytes a palabras.
CWB	Convierte palabras a palabras dobles.
AAD	Ajusta a ASCII después de la división.

3) MANIPULACION DE BITS

CODIGO DE OPERACION	FUNCION
AND	Realiza la operación AND con bytes o palabras.
OR	Realiza la operación OR con bytes o palabras.
XOR	Realiza la operación OR-EXCLUSIVA con bytes o palabras.
NOT	Invierte bytes o palabras.
TEST	Realiza la operación AND y actualiza las banderas.
SHL	Corrimiento lógico a la izquierda.
SHR	Corrimiento lógico a la derecha.
SAL	Corrimiento aritmético a la izquierda.
SAR	Corrimiento aritmético a la derecha.
ROL	Rotación de un byte o palabra a la izquierda.
RCL	Rotación de un byte o palabra a la izquierda a través del acarreo.
ROR	Rotación de un byte o palabra a la derecha.
RCR	Rotación de un byte o palabra a la derecha a través del acarreo.

4) INSTRUCCIONES DE MANEJO DE CADENAS

CODIGO DE OPERACION	FUNCION
REP	Prefijo; repite hasta que CX = 0.
REPE/REPZ	Prefijo; repite hasta que CX = 0 ó bandera ZF ≠ 1.
REPNE/REPNZ	Prefijo; repite hasta que CX = 0 ó bandera ZF = 1.
MOVS	Mueve bytes o palabras.
CMPS	Compara bytes o palabras.
SCAS	Busca un byte o palabra.
LODS	Carga AL o AX con un byte o palabra.
STOS	Guarda en un byte o palabra AL o AX.

5) TRANSFERENCIA DE PROGRAMA

CODIGO DE OPERACION	FUNCION
CALL	Llama a subrutina.
RET	Regresa de subrutina.
JMP	Brinca a otra parte del programa.
INT 3	Interrupción tipo 3.
INTO	Interrupción en desborde (overflow) OF = 1.
IRET	Regreso de interrupción.
JA/JNBE	Brinca si es mayor/brinca si no es menor o igual. Sin signo
JAE/JNB	Brinca si es mayor o igual/brinca si no es menor. Sin signo
JB/JNAE	Brinca si es menor/brinca si no es mayor o igual. Sin signo
JBE/JNA	Brinca si es menor o igual/brinca si no es mayor. Sin signo
JC	Brinca si la bandera de acarreo CF = 1.
JE/JZ	Brinca si es igual/brinca si la bandera de cero ZF = 1.
JG/JNLE	Brinca si es mayor/brinca si no es menor o igual. Con signo
JGE/JNL	Brinca si es mayor o igual/brinca si no es menor. Con signo
JL/JNGE	Brinca si es menor/brinca si no es mayor o igual. Con signo
JLE/JNG	Brinca si es menor o igual/brinca si no es mayor. Con signo

CODIGO DE OPERACION	FUNCION
JNC	Brinca si la bandera de acarreo CF = 0.
JNE/JNZ	Brinca si es diferente/brinca si la bandera de cero ZF = 0.
JNO	Brinca si no hay desborde OF = 0.
JNP/JPO	Brinca si no hay paridad/brinca si paridad impar PF = 0.
JNS	Brinca si no tiene signo SF = 0. Positivo
JO	Brinca si hay desborde OF = 1.
JP/JPE	Brinca si hay paridad/brinca si paridad par PF = 1.
JS	Brinca si tiene signo SF = 1. Negativo.
LOOP	Ejecuta un ciclo hasta que CX = 0.
LOOPE/LOOPZ	Ejecuta ciclo mientras sea igual/mientras ZF = 1 CX ≠ 0.
LOOPNE/LOOPNZ	Ejecuta ciclo mientras sea diferente/mientras ZF = 0 CX ≠ 0.
JCXZ	Brinca si CX = 0.

6) CONTROL DE PROCESO

CODIGO DE OPERACION	FUNCION
STC	**Coloca bandera de acarreo CF = 1.**
CLC	**Limpia bandera de acarreo CF = 0.**
CMD	**Complementa el estado de la bandera de acarreo.**
STD	**Selecciona modo de auto decremento.**
CLD	**Selecciona modo de auto incremento.**
STI	**Habilita interrupciones.**
CLI	**Deshabilita interrupciones.**
HLT	**Detiene hasta que exista una interrupción o reset**
WAIT	**Espera hasta que el pin ~TEST = 0.**
ESC	**Escapa al coprocesador externo.**
LOCK	**Prefijo; bloquea el bus durante la instrucción siguiente.**
NOP	**Ejecuta una no operación.**

LIBRERIAS DE LENCUAJE "C"

LIBRERIAS CORRESPONDIENTES A TURBO "C" 2.0

```
        NAME    c0
        PAGE    60,132
;[]------------------------------------------------[]
;;      C0.ASM -- Start Up Code                     |
;;                                                  |
;;      Turbo-C Run Time Library        version 2.0 |
;;                                                  |
;;      Copyright (c) 1988 by Borland International Inc.  |
;;      All Rights Reserved.                        |
;[]------------------------------------------------[]

        INCLUDE RULES.ASI

_Strict87_      equ     false                   ; emulation skips peculiar details

;       Segment and Group declarations

_TEXT   SEGMENT BYTE PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT PARA PUBLIC 'DATA'
_DATA   ENDS
IFNDEF  __NOFLOAT__
_EMUSEG SEGMENT WORD COMMON 'DATA'
_EMUSEG ENDS
ENDIF
_CRTSEG SEGMENT WORD COMMON 'DATA'
_CRTSEG ENDS
_CVTSEG SEGMENT WORD PUBLIC 'DATA'
_CVTSEG ENDS
_SCNSEG SEGMENT WORD PUBLIC 'DATA'
_SCNSEG ENDS
IFNDEF  __HUGE__
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS
_BSSEND SEGMENT BYTE PUBLIC 'STACK'
_BSSEND ENDS
ENDIF
IFNDEF  __TINY__
_STACK  SEGMENT STACK 'STACK'
_STACK  ENDS
ENDIF

IFNDEF  __NOFLOAT__

IF      LDATA
IFNDEF  __HUGE__
DGROUP  GROUP   _DATA, _EMUSEG, _CRTSEG, _CVTSEG, _SCNSEG, _BSS, _BSSEND
```

```
ELSE
DGROUP GROUP  _DATA, _EMUSEG, _CRTSEG, _CVTSEG, _SCNSEG
ENDIF
ELSE
IFNDEF __TINY__
DGROUP GROUP  _DATA, _EMUSEG, _CRTSEG, _CVTSEG, _SCNSEG, _BSS, _BSSEND
ELSE
DGROUP GROUP  _TEXT, _DATA, _EMUSEG, _CRTSEG, _CVTSEG, _SCNSEG, _BSS, _BSSEND
ENDIF
ENDIF


ELSE
IF    LDATA
IFNDEF __HUGE__
DGROUP GROUP  _DATA, _CRTSEG, _CVTSEG, _SCNSEG, _BSS, _BSSEND
ELSE
DGROUP GROUP  _DATA, _CRTSEG, _CVTSEG, _SCNSEG
ENDIF
ELSE
IFNDEF __TINY__
DGROUP GROUP  _DATA, _CRTSEG, _CVTSEG, _SCNSEG, _BSS, _BSSEND
ELSE
DGROUP GROUP  _TEXT, _DATA, _CRTSEG, _CVTSEG, _SCNSEG, _BSS, _BSSEND
ENDIF
ENDIF

ENDIF

    ASSUME  CS:_TEXT, DS:DGROUP

;   External References

ExtProc@    main,  __CDECL__
ExtProc@    _setargv, __CDECL__
ExtProc@    _setenvp, __CDECL__
ExtProc@    exit,  __CDECL__

IF    LDATA EQ false
ExtSym@ _heaplen, WORD, __CDECL__
ENDIF
ExtSym@ _stklen, WORD, __CDECL__

    SUBTTL  Start Up Code
    PAGE
;/*                                                    */
;/*----------------------------------------------------*/
;/*                                                    */
;/*   Start Up Code                                    */
;/*   -------------                                    */
;/*                                                    */
;/*----------------------------------------------------*/
;/*                                                    */
PSPHigh     equ   00002h
PSPEnv      equ   0002ch
PSPCmd      equ   00080h

IFDEF   __NOFLOAT__
        MINSTACK      equ     128     ; minimal stack size in words
ELSE
```

```
        MINSTACK        equ     256     ; minimal stack size in words
ENDIF
;
;   At the start, DS and ES both point to the segment prefix.
;   SS points to the stack segment except in TINY model where
;   SS is equal to CS
;
_TEXT   SEGMENT
IFDEF   __TINY__
        ORG     100h
ENDIF
START   PROC    NEAR
;   Save general information, such as :
;       DGROUP segment address
;       DOS version number
;       Program Segment Prefix address
;       Environment address
;       Top of the heap
;
IFDEF   __TINY__
        mov     dx, cs          ; DX = GROUP Segment address
ELSE
        mov     dx, DGROUP      ; DX = GROUP Segment address
ENDIF
        mov     cs:DGROUP@@, dx
        mov     ah, 30h
        int     21h
        mov     bp, dx:[PSPHigh]; BP = Highest Memory Segment Addr
        mov     bx, dx:[PSPEnv] ; BX = Environment Segment address
        mov     ds, dx
        mov     _version@, ax   ; Keep major and minor version number
        mov     _psp@, es       ; Keep Program Segment Prefix address
        mov     _envseg@, bx    ; Keep Environment Segment address
        mov     word ptr _heaptop@ + 2, bp
        mov     _8087@, -1
;
;   Save several vectors and install default divide by zero handler.
;
        call    SaveVectors
;
;   Look for a '87' environment variable, and use this loop to
;   count the number of environment variables and to compute the
;   environment size.
;   Each variable is ended by a 0 and a zero-length variable stops
;   the environment. The environment can NOT be greater than 32k.
;
        les     di, dword ptr _envLng@
        mov     ax, di
        mov     bx, es
        mov     cx, 07FFFh      ; Environment cannot be > 32 Kbytes
Is8087Var:  label   near
        cmp     word ptr es:[di], '78'
        jne     GotVarLng
        mov     dx, es:[di+2]
        cmp     dl, '='
        jne     GotVarLng
        and     dh, not ''
        inc     _8087@
        cmp     dh, 'V'
        jne     GotVarLng
```

```
            inc     _8087@
GetVarLng   label   near
            regsav  <ds,es>
            jcxz    InitFailed      ; Bad environment !!!
            inc     bx              ; BX = Nb environment variables
            cmp     es:[di], al
            jnz     Isl87Var        ; Next variable ...
            or      ch, 10000000b
            neg     cx
            mov     _envLng@, cx    ; Save Environment size
            mov     cx, dPtrSize / 2
            shl     bx, cl
            add     bx, dPtrSize * 4
            and     bx, not ((dPtrSize * 4) - 1)
            mov     _envSize@, bx   ; Save Environment Variables Nb.

;   Determine the amount of memory that we need to keep

IF  LDATA
            mov     dx, ss
            sub     bp, dx          ; BP = remaining size in paragraphs
IFDEF   __HUGE__
            mov     di, seg _stklen@
            mov     es, di
            mov     di, es:_stklen@ ; DI = Requested stack size
ELSE
            mov     di, _stklen@    ; DI = Requested stack size
ENDIF
;
; Make sure that the requested stack size is at least MINSTACK words.
;
            cmp     di, 2*MINSTACK  ; requested stack big enough ?
            jae     AskedStackOK
            mov     di, 2*MINSTACK  ; no --> use minimal value
IFDEF   __HUGE__
            mov     es:_stklen@, di ; override requested stack size
ELSE
            mov     _stklen@, di ; override requested stack size
ENDIF
AskedStackOK    label   near
            mov     cl, 4
            shr     di, cl          ; $$$ Do not destroy CL $$$
            inc     di              ; DI = Stack size in paragraphs
            cmp     bp, di
            jnb     ExcessOfMemory  ; Much more available than needed
ELSE
            mov     dx, ds
            sub     bp, dx          ; BP = remaining size in paragraphs
            mov     di, _stklen@    ; DI = Requested stack size
;
; Make sure that the requested stack size is at least MINSTACK words.
;
            cmp     di, 2*MINSTACK  ; requested stack big enough ?
            jae     AskedStackOK
            mov     di, 2*MINSTACK  ; no --> use minimal value
            mov     _stklen@, di    ; override requested stack size
AskedStackOK    label   near
            add     di, offset DGROUP: edata@
            jb      InitFailed      ; DATA segment can NOT be > 64 Kbytes
            add     di, _heaplen@
```

```
        jb      InitFailed      ; DATA segment can NOT be > 64 Kbytes
        mov     cl, 4
        shr     di, cl          ; $$$ Do not destroy CL $$$
        inc     di              ; DI = DS size in paragraphs
        cmp     bp, di
        jb      InitFailed      ; Not enough memory
        cmp     _stklen@, 0
        je      ExpandDS        ; Expand DS up to 64 Kb
        cmp     _heaplen@, 0
        jne     ExcessOfMemory  ; Much more available than needed
ExpandDS        label   near
        mov     di, 1000h
        cmp     bp, di
        jb      ExcessOfMemory  ; Enough to run the program
        mov     di, bp
        jmp     short ExcessOfMemory  ; Enough to run the program
ENDIF

;       All initialization errors arrive here

InitFailed      label   near
        jmp     near ptr abort@

;       Return to DOS the amount of memory in excess
;       Set far heap base and pointer

ExcessOfMemory  label   near
        mov     bx, di
        add     bx, dx
        mov     word ptr _heapbase@ + 2, bx
        mov     word ptr _brklvl@ + 2, bx
        mov     ax, _psp@
        sub     bx, ax          ; BX = Number of paragraphs to keep
        mov     es, ax          ; ES = Program Segment Prefix address
        mov     ah, 04Ah
        push    di              ; preserve DI
        int     021h            ; this call clobbers SI,DI,BP !!!!!!
        pop     di              ; restore  DI
;
;       Set the program stack.  Take care to prevent the disastrous
;       interrupt that could happen with a stack that is half switched.
;
        shl     di, cl          ; $$$ CX is still equal to 4 $$$

                cli
        mov     ss, dx
        mov     sp, di
                sti

IFNDEF  __HUGE__

;       Reset uninitialized data area

        xor     ax, ax
        mov     es, cs:DGROUP@@
        mov     di, offset DGROUP: bdata@
        mov     cx, offset DGROUP: edata@
        sub     cx, di
        rep     stosb
ENDIF
```

```
IFNDEF __NOFLOAT__

;       Install floating point software

        push    cs              ;Simulates a FAR call
        call    ds:[__emu1st]
ENDIF

;       Prepare main arguments

        call    _setargv@
        call    _setenvp@

        mov     ah, 0
        int     1ah                             ; get current BIOS time in ticks
        mov     word ptr _StartTime@,dx         ; save it for clock() fn
        mov     word ptr _StartTime@+2,cx

IFNDEF __OLDCONIO__
IF      LPROG
        push    cs              ; Simulates a FAR call
ENDIF
        call    ds:[__crt1st]   ; Initialize window sizes, etc.
ENDIF

;       ExitCode = main(argc,argv,envp);

IF      LDATA
        push    word ptr environ@+2
        push    word ptr environ@
        push    word ptr _argv@+2
        push    word ptr _argv@
ELSE
        push    word ptr environ@
        push    word ptr _argv@
ENDIF
        push    _argc@
        call    main@

;       Flush and close streams and files

        push    ax
        call    exit@

;-------------------------------------------------------------------------
;       _exit()
;
;   Restore interrupt vectors taken during startup.  signal() functions
;       could have grabbed vectors 0, 4, 5 or 6.
;
;       Check for NULL pointer errors.
;
;       Exit to DOS.
;
;NOTE : _exit() doesn't close any files or run exit functions.  This is a
;       minimal 'cleanup & quit' program exit.
;-------------------------------------------------------------------------
PubProc@        _exit,  __CDECL__
        mov     ds, cs:DGROUP@@
```

```
IF    LPROG
        call   far ptr _restorezero@ ; restore captured INT vectors
ELSE
        call   near ptr _restorezero@
ENDIF

IFNDEF __NOFLOAT__

;    Restore interrupt vectors taken by __emu1st

        push   cs           ;Simulation of a FAR call
        call   dx:[__emuLast]
ENDIF

IF    LDATA EQ false
IFNDEF __TINY__

;    Check for null pointers before exit

        xor    ax, ax
        mov    si, ax
        mov    cx, lgth_CopyRight
        cld
ComputeChecksum label  near
        add    al, [si]
        adc    ah, 0
        inc    si
        loop   ComputeChecksum
        sub    ax, Checksum
        jz     ExitToDOS
        mov    cx, lgth_NullCheck
        mov    dx, offset DGROUP: NullCheck
        call   ErrorDisplay
ENDIF
ENDIF

;    Exit to DOS

ExitToDOS     label  near
        mov    bp,sp
        mov    ah,4Ch
        mov    al,[bp+cPtrSize]
        int    21h              ; Exit to DOS
ExitProc@     _exit, __CDECL__
STARTX        ENDP

    SUBTTL Vector save/restore & default Zero divide routines
    PAGE
;[]------------------------------------------------------------[]
;                                                              |
; Interrupt Save/Restore routines and default divide by zero   |
; handler.                                                     |
;                                                              |
;[]------------------------------------------------------------[]

ZeroDivision   PROC   FAR
        mov    cx, lgth_ZeroDivMSG
        mov    dx, offset DGROUP: ZeroDivMSG
        jmp    MsgExit3
```

```
ZeroDivision    ENDP

;---------------------------------------------------------------
;       saveVectors()
;
;       Save vectors for 0, 4, 5 & 6 interrupts.  This is for extended
;       signal()/raise() support as the signal functions can steal these
;       vectors during runtime.
;---------------------------------------------------------------
SaveVectors     PROC    NEAR
                push    ds
; Save INT 0
                mov     ax, 3500h
                int     021h
                mov     word ptr _Int0Vector@, bx
                mov     word ptr _Int0Vector@+2, es
; Save INT 4
                mov     ax, 3504h
                int     021h
                mov     word ptr _Int4Vector@, bx
                mov     word ptr _Int4Vector@+2, es
; Save INT 5
                mov     ax, 3505h
                int     021h
                mov     word ptr _Int5Vector@, bx
                mov     word ptr _Int5Vector@+2, es
; Save INT 6
                mov     ax, 3506h
                int     021h
                mov     word ptr _Int6Vector@, bx
                mov     word ptr _Int6Vector@+2, es
;
;       Install default divide by zero handler.
;
                mov     ax, 2500h
                mov     dx, cs
                mov     ds, dx
                mov     dx, offset ZeroDivision
                int     21h

                pop     ds
                ret
SaveVectors     ENDP

;---------------------------------------------------------------
;       restorezero() puts back all the vectors that SaveVectors took.
;
;NOTE : TSRs must BE AWARE that signal() functions which take these
;       vectors will be deactivated if the keep() function is executed.
;       If a TSR wants to use the signal functions when it is active it
;       will have to save/restore these vectors itself when activated and
;       deactivated.
;---------------------------------------------------------------
PubProc@        _restorezero, __CDECL__

IFDEF   __HUGE__
        push    ds
        mov     ds, ax; DGROUP@@
ENDIF
        push    ds
```

```
        mov   ax, 2508h
        lds   dx, _Int8Vector@
        int   21h
              pop     ds

              push    ds
        mov   ax, 2504h
        lds   dx, _Int4Vector@
        int   21h
              pop     ds

              push    ds
        mov   ax, 2505h
        lds   dx, _Int5Vector@
        int   21h
              pop     ds

IFNDEF  __HUGE__
              push    ds
ENDIF
        mov   ax, 2506h
        lds   dx, _Int6Vector@
        int   21h
        pop   ds

        ret
EndProc@      _restorezero, __CDECL__

    SUBTTL Miscellaneous
    PAGE
;[]------------------------------------------[]
;|                                            |
;|   Miscellaneous functions                  |
;|                                            |
;[]------------------------------------------[]

IFNDEF  __NOFLOAT__
NoEmulator    PROC    FAR
        mov   _8087@, 0
        ret
NoEmulator    ENDP
ENDIF

IFNDEF  __OLDCONIO__
Proc@         NoConsole, __CDECL__
        ret
EndProc@      NoConsole, __CDECL__
ENDIF

ErrorDisplay  PROC    NEAR
        mov   ah, 040h
        mov   bx, 2
        int   021h
        ret
ErrorDisplay  ENDP

PubProc@      abort, __CDECL__
        mov   cx, [@$_abortMSG
        mov   dx, offset DGROUP: abortMSG
MsgExit3      label   near
```

```
        mov     dx, cs: DGROUP@@
        call    ErrorDisplay
CallExit3       label   near
        mov     ax, 3
        push    ax
        call    _exit@          ; _exit(3);
EndProc@        abort, __CDECL__

;       The DGROUP@ variable is used to reload DS with DGROUP

PubSym@ DGROUP@, <dw    ?>, __PASCAL__
_TEXT   ENDS

        SUBTTL  Start Up Data Area
        PAGE
;[]------------------------------------------------------[]
;|    Start Up Data Area                                  |
;|                                                        |
;|    WARNING      Do not move any variables in the data  |
;|                 segment unless you're absolutely sure  |
;|                 that it does not matter.               |
;|                                                        |
;[]------------------------------------------------------[]
_DATA   SEGMENT
;       The CopyRight string must NOT be moved or changed without
;       changing the null pointer check logic

CopyRight       db      4 dup(0)
                db      'Turbo-C - Copyright (c) 1988 Borland Intl.',0
lgth_CopyRight  equ     $ - CopyRight

IF      LDATA EQ false
IFNDEF  __TINY__
CheckSum        equ     00D37h
NullCheck       db      'Null pointer assignment', 13, 10
lgth_NullCheck  equ     $ - NullCheck
ENDIF
ENDIF

ZeroDivMSG      db      'Divide error', 13, 10
lgth_ZeroDivMSG equ     $ - ZeroDivMSG

abortMSG        db      'Abnormal program termination', 13, 10
lgth_abortMSG   equ     $ - abortMSG

;
;                       Interrupt vector save areas
;
;       Interrupt vectors 0,4,5 & 6 are saved at startup and then restored
;       when the program terminates.  The signal/raise functions might
;       steal these vectors during execution.
;
PubSym@         _Int0Vector <dd     0>,         __CDECL__
PubSym@         _Int4Vector <dd     0>,         __CDECL__
PubSym@         _Int5Vector <dd     0>,         __CDECL__
PubSym@         _Int6Vector <dd     0>,         __CDECL__
;
;                       Miscellaneous variables
;
PubSym@         _argc,      <dw     0>,         __CDECL__
```

```
@FarPtr@    _argv,      0,                  __CDECL__
@FarPtr@    environ,    0,                  __CDECL__
Public@     _envLng,      <dw   0>,           __CDECL__
Public@     _envseg,      <dw   0>,           __CDECL__
Public@     _osvMinor,    <dw   0>,           __CDECL__
Public@     _osp,       <dw   0>,           __CDECL__
Public@     _version,     <label word>,       __CDECL__
Public@     _osmajor,     <db   0>,           __CDECL__
Public@     _osminor,     <db   0>,           __CDECL__
Public@     _cmm,       <dw   0>,           __CDECL__
Public@     _8087,      <dw   0>,           __CDECL__
Public@     _StartTime,   <dw   0,0>,         __CDECL__
;       Memory management variables
IF      LDATA EQ false
Public@     __heapbase,   <dw   DGROUP:edata@>, __CDECL__
Public@     __brklvl,     <dw   DGROUP:edata@>, __CDECL__
Public@     __heaptop,    <dw   DGROUP:edata@>, __CDECL__
ENDIF
Public@     __heapbase,   <dd   0>,           __CDECL__
Public@     __brklvl,     <dd   0>,           __CDECL__
Public@     __heaptop,    <dd   0>,           __CDECL__
IF      LDATA EQ false
        IFNDEF __NOFLOAT__
;                               Emulator variables
                INCLUDE         emuvars.asi
        ENDIF
ENDIF
_DATA   ENDS
IFNDEF  __NOFLOAT__
_EMUSEG SEGMENT
__emuInt    dw      NoEmulator
__emuLast   dw      NoEmulator
_EMUSEG ENDS
ENDIF
IFNDEF  __OLDCONIO__
_CRTSEG SEGMENT
__crtInt    dw      NoConsole@
_CRTSEG ENDS
ENDIF
_CVTSEG SEGMENT
Public@ _RealCvtVector, <label word>, __CDECL__
_CVTSEG ENDS
_SCNSEG SEGMENT
Public@ _BcdToIVector, <label word>, __CDECL__
_SCNSEG ENDS
IFNDEF  __HUGE__
_BSS    SEGMENT
bdata@  label   byte
_BSS    ENDS
_BSSEND SEGMENT
edata@  label   byte
_BSSEND ENDS
ENDIF
IFNDEF  __TINY__
_STACK  SEGMENT
    dw    64 dup (?)
        IF      LDATA
                or$     0
                IFNDEF __NOFLOAT__
                                        Emulator variables
```

```
                        INCLUDE         emuvars.ani
                                        cvtas
                ENDIF
PUBLIC          emuTop@                                 ; for use in stack-underflow checks.
                emuTop@         label   byte
        ENDIF
_STACK  ENDS
ENDIF
    END     START%
```

LISTADO DEL PROGRAMA ROMLDR.C

A continuación se muestra el listado del programa ROMLDR:

```
/*
        ROMLDR.C
                EXE locator program written by: Charles B. Allison
                Editado en la revista "The C Users Journal"
                Marzo de 1994; Volumen 12; Numero 3; Página 35
*/
#include <sys\stat.h>
#include <io.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <conio.h>
#include <ctype.h>
#include <fcntl.h>
#include <string.h>

#define   BC      1       /* Se asume Borland C */

typedef struct {
        unsigned sig;                   /* signature = 4D5Ah        */
        unsigned last_sec_leng;         /* length of last sector in file mod 512 */
        unsigned file_size; /* size of file in 512 byte pages includes hdr */
        unsigned num_reloc;             /* number of relocation items */
        unsigned hdr_siz;   /* # of 16 bytes paragraphs in header */
        unsigned min_ld_para;           /* min # of paragraphs above load file */
        unsigned max_ld_para;           /* max # of paragraphs requested by file */
        unsigned disp_stack_seg;        /* rel displacement of stack segment */
        unsigned sp;                    /* contents of stack ptr on entry to prog */
        unsigned chksm;     /* check sum of file */
        unsigned ip;                    /* beginning instruction ptr */
```

```
        unsigned rel_cs_seg;        /* relative cs segment */
        unsigned off_reloc; /* offset to 1st relocation item typ. 1e */
        unsigned over_lay; /* overlay number */
        unsigned rsrved;    /* ?? reserved ?? */
                            /* relocation item format is seg:off location
                             relative to the beginning of the code section */
} EXE_HDR;

struct MP_TBL {
        long addrs;                 /* segment beg. address */
        long haddrs;                /* segment high addrs */
        char class[12];             /* class of object */
} maptable[120];

void sort_table( void );
void gethdr( char * );
long hexcvt( char * );
size_t read_block( char far *, size_t );
long read_segm( int );
size_t write_block( char far *, size_t );
long write_segm( long );
int fix_segm( int );
int config( char * );
void term_error( int );

unsigned(*ch_ptr)[2];       /* pointer to translation table [0] = offset, [1] = segment */
char mfile[14] = "rom.map"; /* dumy file names */
char bfile[14] = "rom.bin";
char efile[14] = "rom.exe";

/* columns for starting and ending addresses in MAP */

#define         LCOL    1
#define         HCOL    8

#ifdef          BC
```

```
#define         MAPCOL          41      /* bc map file class column */
#else
#define         MAPCOL          45      /* ms map file class column */
#endif

int class_loc = MAPCOL;
FILE *mapfile, *exefile, *binfile;

#define         BUF_SIZE 60

char mapstring[BUF_SIZE];
long fsize;                             /* number of bites in exe file */
int nsegs;                      /* number of segments in map */
unsigned runnode = 0x0000, runnode = 0x40;
char header[10000];
EXE_HDR *filhdr = (EXE_HDR *)header;
char far *seg_buffer;
int next_fix = 0;
char run_class[15] = "FAR_DATA";
unsigned runsdata;              /* beginning ram segment */

/* ---------- MAIN ---------- */
int main( int argc, char *argv[] )
{
int r_class_flag = 1, i;
long tmp, exesz;
        //0x10000L
        if ( ( seg_buffer = ( char far * ) farmalloc( 0x10000L ) ) == NULL )
          term_error( 0 );
        if ( argc == 1 )                /* any cfg file name ? */
          term_error( -1 );
        config( argv[1] );
         i = 0;
        while ( fgets( mapstring, BUF_SIZE, mapfile ) ){
                                                        /* process map file from mapstring input
maptable[1] - n contains the segments -
```

```
        class STACK should be last one ends with i having n + 1 segments */
        if ( (int)strlen(mapstring) > class_loc + 1 ){
                        /* get ride of \n at end of string */
              mapstring[(int)strlen(mapstring)-1] = '\0';
              if ( (tmp = hexcvt(&mapstring[LCOL])) >= 0){
                maptable[i].addrs = tmp;
                maptable[i].haddrs=hexcvt(&mapstring[HCOL]);

                                        //&mapstring[class_loc]);
                strcpy(maptable[i].class, &mapstring[class_loc] );
                if( r_class_flag )
                      if( strcmp(&mapstring[class_loc],rsm_class ) == 0 ){
                        /* set it to first class ocurrence */
                        rsmdata = (maptable[i].addrs) >> 4;
                        r_class_flag = 0;
                      }
                printf("\n Segmento %4.4lx Clase %s",maptable[i].addrs/16,
maptable[i].class );
                i++;
              } /* end - if hexcvt */
         } /* end if strlen */
         if( i >= 119 )                    /* error too many segments */
              break;
        } /* end of while */
        if( feof(mapfile) )
          printf("\nFin de archivo\n");
        else
          printf("\nError leyendo archivo map\n");
        nsegs = i - 1;                    /* number of segments [1 to nsegs] */
        gethdr( header );                 /* read in the exe header info */
                                  /* size of object section file */
        fsize = (long)(512L * (filhdr->file_size-1)) +
          filhdr->lst_soc_lng - 16L * filhdr->hdr_siz );
        printf("\nDirección de Inicio %4.4x:%4.4x\n", romasadr+filhdr->rel_cs_seg,
          filhdr->ip );
        printf("Tamaño de Rom %lx\n", fsize );
```

```
                                                        /* process the exe file header - sort fix ups */
        sort_table();
                                                        /* read in exe file by segment do fixups frommap
 table and write to output file */
        for( i = 0; i < nsegs; i++ ){
          if( (asize=read_segm(i)) > 0L ){
                                                        /* ignore 0 length segments */
              fix_segm(i);
              write_segm(asize);
          }
        }
                                                        /* done - end the program */
        farfree( seg_buffer );
        fcloseall();
        return 0;
}
/* ---------------------------- */
/* qsort routine for far pointers */
int cmp_ptr( const void *a, const void *b )
{
long vala, valb;
        vala = ((long)(((unsigned*)a)[0]) + ((((unsigned*)a)[1] << 4 );
        valb = ((long)(((unsigned*)b)[0]) + ((((unsigned*)b)[1] << 4 );
        vala -= valb;
        if( vala < 0 )
                return -1;
        if( vala > 0 )
                return 1;
        return 0;
}
/* ------------ sort_table ------------ */
/* sort header table */
void sort_table( void )
{
        qsort((void*)&headptr[fillhdr->off_reloc], fillhdr->num_reloc, 4, cmp_ptr );
}
```

```
/* ---------- read_block ---------- */
size_t read_block( char far *segbuf, size_t segsz )
{
        if( fread(segbuf, 1, segsz, exefile ) != segsz )
                term_error( -7 );
        return segsz;
}
/* ---------- read_segm ---------- */
long read_segm( int i )
{
long segsize;
        segsize =  maptable[i].haddrs - maptable[i].addrs;
        if( !segsize )
                return 0;
        segsize += maptable[i+1].addrs - (maptable[i].haddrs);
        if( segsize <= 0x8000 )
                read_block( seg_buffer, (size_t)segsize );
        else {
                read_block( seg_buffer, 0x8000 );
                read_block( (&seg_buffer[0x8000]), (size_t)(segsize-0x8000) );
        }
        return segsize;
}
/* ---------- write_block ---------- */
size_t write_block( char far* segbuf, size_t segsz )
{
        if( fwrite( segbuf, 1, segsz, binfile ) != segsz )
                term_error( -8 );
        return segsz;
}
/* ---------- write_segm ---------- */
long write_segm( long segsize )
{
        if( !segsize )
                return 0;
        if( segsize <= 0x8000 )
```

```
                write_block( seg_buffer, segsize );
        else {
                write_block( seg_buffer, 0x8000 );
                write_block( (&seg_buffer[0x8000]), (size_t)(segsize-0x8000) );
        }
        return segsize;
}
/* ---------- fix_segm ---------- */
int fix_segm( int i )
{
unsigned tmp, segg, fixup;
unsigned far * fixptr;
        segg = (unsigned)(segtable[i].addrs/16L);
        while( next_fix < fillhdr->num_reloc ){
                if( ch_ptr[next_fix][1] > segg )
                        break;
                tmp = ch_ptr[next_fix][0];
                                                        /* offset into buffer */
                fixptr = (unsigned far*)&seg_buffer[tmp];
                fixup = *fixptr;
                                                        /* modify segment fixup according to type */
                if( fixup >= romdata ){
                                                        /* modify for rom */
                        fixup -= romdata;
                        fixup += romaddr;
                }
                else    /* handle as rom */
                        fixup += romaddr;
                *fixptr = fixup;
                next_fix++;
        } /* end of while */
        return 0;
}
/* ---------- hexcvt ---------- */
/* do hex digit to unsigned long */
long hexcvt( char * num )
```

```
{
char *term;
long value;
        value = strtoul( num, &term, 16 );
        return value;
}
/* ---------- gethdr ---------- */
void gethdr( char *buf )
{
int i, j = 32;                                  /* index counter */
        if( fread( &buf[0], 1, 32, exefile ) < 32 )
                term_error( -6 );
                /* have filhdr contents so get size of full header */
        if ( fread(&buf[j],16,filhdr->hdr_siz-2, exefile) < filhdr->hdr_siz-2 )
                term_error( -6 );
        (unsigned *)ch_ptr = (unsigned*)(&buf[filhdr->off_reloc]);
                                                        /* get address of relocation table - ch_ptr[n][m]
                                                           m - 0 offset, 1 - seg, n relocation # */
}
/* ---------- config ---------- */
/* get configuration data */
int config( char * cfgfile )
{
FILE *cfg;
char buf[80];
        if ( (cfg = fopen(cfgfile,"r"))==NULL )
                term_error(-1);
        if( fgets( buf, 80, cfg )== NULL )
                term_error(-2);
        if( sscanf( buf, "%s %s %s", &mfile, &efile, &bfile) != 3 )
                term_error(-2);
                                                        /* now try to open input file 1 */
        if( (mapfile=fopen(mfile,"r"))==NULL )
                term_error(-3);
        if( (exefile=fopen(efile,"rb"))==NULL )
                term_error(-4);
```

```
        if( (binfile=fopen(bfile,"wb"))==NULL )
                term_error(-5);
        if( fgets( buf, 80, cfg )==NULL )
                term_error(-7);
        if( sscanf(buf,"%4x %4x", &romseadr, &ramseadr ) != 2 )
                term_error(-7);
        if( fgets( buf, 80, cfg )== NULL )
                term_error(-7);
        if( sscanf( buf, "%s", &ram_class ) != 1 )
                term_error(-7);
        return 0;
}
/* ---------- error handler ---------- */
char *errlist[10] = {
        "Error en Petición de Memoria",                         //"Memory Allocation Error",
        "Archivo de Configuración Inválido, Use: romldr config.cfg",
        //"No configuration file,Usage:romldr cfgfile.cfg",
        "Error en Archivo de Configuración - Nombres de Archivos",
        //"Configuration file error - File names",
        "Error al Abrir Archivo Map",                           //"Map file open error",
        "Error al Abrir Archivo Exe",                           //"Exe file open error",
        "Error al Abrir Archivo Bin",                           //"Bin file open error",
        "Error al Leer Cabecera",                               //"Error reading header",
        "Error al Leer Archivo Exe",                            //"Error reading exe file",
        "Error al Escribir Archivo Bin",                        //"Error writing Bin file",
        " "
};
void term_error( int errnum )
{
        errnum = abs( errnum );
        if ( errnum >= 9 )
                exit(-1);
        printf("%s\n", errlist[errnum] );
        farfree( msg_buffer );
        exit( errnum );
}
```

LOS MICROPROCESADORES INTEL
ARQUITECTURA, PROGRAMACION E INTERFACES,
BARRY B. BREY,
EDITORIAL PRENTICE HALL HISPANOAMERICANA,
3A. EDICION, MEXICO 1995.

FUNDAMENTOS DE LOS MICROPROCESADORES
TOKHEIM ROGER L.
EDITORIAL MC GRAW HILL.
2DA. EDICION, MEXICO

INTRODUCCION AL MICROPROCESAOR 8086/8088.
CRISTOPHER L. MORGAN - MITCHELL WAITE.
EDITORIAL MC GRAW HILL
U.S.A. INC. 1982

LOGICA DIGITAL Y DISEÑO DE COMPUTADORES
MORRIS MANO M.
EDITORIAL PRENTICE HALL
PRENTICE HALL, INC 1982

MICROPROCESSORS AND INTERFACING
ROGRAMMING AND HARDWARE
HALL DOUGLAS V.
EDITORIAL MC GRAW HILL
INTERNATIONAL EDITION, 1986

MICROPROCESSORS AND INTERFACING
ROGRAMMING AND HARDWARE
HALL DOUGLAS V.
EDITORIAL MC GRAW HILL
INTERNATIONAL EDITION 1986

MICROPROCESADORES DE 16 BITS
IAN R. WHITWORTH
COLECCION CIENCIA ELECTRONICA
EDITORIAL GUSTAVO GILI, S.A.
EDICION CASTELLANA, 1986

PERIPHERALS
INTEL
IL, USA. 1990

MEMORY I
INTEL
IL, USA. 1990

MEMORY II
INTEL
IL, USA. 1990

CMOS LOGIC DATABOOK
NATIONAL SEMICONDUCTOR
CA, USA. 1988

TURBO C BIBLE
BARKAKATI, NABAJYOTI
EDITORIAL HOWARD W. SAMS & CO.
IN , USA. 1990

USING TURBO C
SCHILDT, HERBERT
EDITORIAL Mc GRAW-HILL
CA , USA. 1989