UNIVERSIDAD POLITECNICA

DE

LAS PALMAS

Escuela Universitaria Politécnica

INGENIERIA TECNICA DE TELECOMUNICACION

Especialidad:

Equipos Electrónicos

PROYECTO:

"Microprocesador de 16 bits, 8086."

Aplicación para un Sistema Multiusuario.

Tutor: Juan Gómez Mena

Alumno: Benigno Blanco Montenegro

INDICE:

Pagina

TEMA I "DESCRIPCION DEL µp 8086"

Pagina

Pagina

Pagina

TEMA III "LENGUAJE ENSAMBLADOR DEL 8086"

Pagina

Pagina

TEMA IV "DIAGRAMA FUNCIONAL DE BLOQUES DEL SDK-86"

TEMA I:

"DESCRIPCION DEL pp 8086"

I) ARQUITECTURA DE LA CPU 8086.

I-1) ARQUITECTURA INTERNA.

El 8086 se diseñó para realizar, al mismo tiempo, las principales funciones de tranferencia de datos y busqueda de instrucciones. Para ello, el procesador 8086 consta de dos procesadores interconectados en la misma pieza de silicio (chip).

Uno está encargado de buscar instrucciones y otro de ejecutarlas. La unidad encargada de buscar instrucciones utiliza un método llamado de estructura tubular (Pipeline) o por cola para almacenar nuevas instrucciones hasta que se necesiten.

Los dos procesadores son la E.U. y el B.I.U.

I-1-1.- E.U.:

Es la unidad de ejecución y es el procesador principal.

La E.U. consta de una ALU de 16 bits, de los registros generales, de los flags y del decodificador de instrucciones.

Todos los registros y datos de paso en la E.U. son de 16 bits para una rápida transferencia interna.

La E.U. lee los códigos de operación de una cola de espera (Queue) y los ejecuta. La cola de espera se haya en el otro procesador B.I.U. Es decir, la E.U. obtiene las instrucciones de la B.I.U. que las mantiene en su cola.

Cuando una instrucción requiere acceso a memoria o a un dispositivo periférico, la E.U. requiere al B.I.U. a obtener o almacenar un dato.

I-1-2.- B.I.U.:

Es la unidad de interfaz del bus.

La B.I.U. ejecuta todas las operaciones del bus para la E.U. El dato es transferido entre la CPU y memoria dispositivos de entrada/salida por una demanda desde la E.U. En suma, durante los periodos en que la E.U. está ocupada ejecutando instrucciones, la B.I.U. mira adelante y busca más instrucciones de memoria.

Las instrucciones son almacenadas en una RAM interna llamada instrucción Queue (cola de instrucciones). Esta es de 6 bits, y el acceso se

hace siempre por palabras de 16 bits.

La B.I.U. no inicia una busqueda hasta que no hay dos byte de la queue vacios. La B.I.U. normalmente obtiene dos byte de instrucción para la busqueda; si un programa transferido obliga a la busqueda de una dirección impar, la B.I.U. automáticamente lee un byte de una dirección impar y entonces reanuda la busqueda de dos byte palabras desde la subsiguiente dirección par.

Luego el trabajo principal de la B.I.U. es la de localizar instrucciones y de transferir todos los datos entre registros y el mundo exterior.

La B.I.U. se interrumpe en dos casos particulares:

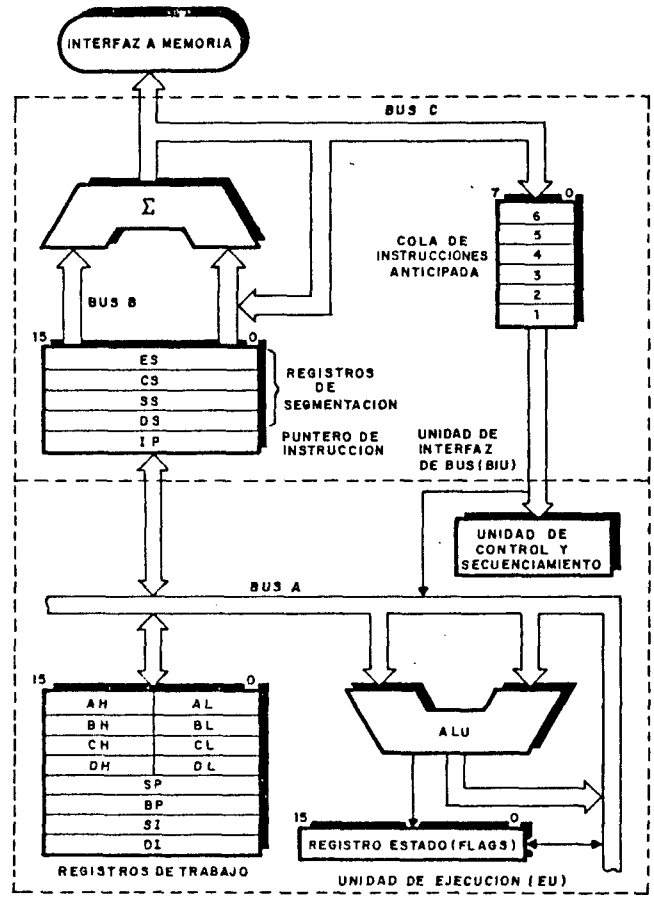
- En el primer caso es donde la E.U. encuentra instrucciones de acceso a memoria,pide a la B.I.U. la lectura o escritura de un dato en memoria.
- En el segundo caso es en el que la E.U. encuentra un salto de programa (JMP,CALL,...), el contenido de la cola de espera ya no está al día y la E.U. debe pedir una reinicialización a partir de la dirección de salto.

La B.I.U. consta de los registros segmentos, del registro puntero de instrucción (IP),de la cola de instrucciones y del generador de dirección y

control del bus.

La función primordial de la B.I.U. es la de generar una dirección física de 20 bits, lo que se realiza por medio de registros de base conteniendo la dirección de principio de segmentos de 64 bits, y de un desplazamiento de 16 bits (offset).

En la siguiente figura vemos el diagrama de bloques de la CPU 8086, en la que se ve las dos unidades del procesador.



I-2) REGISTROS DE LA CPU 8086.

El procesador 8086 dispone de cuatro tipos de registros internos:

- Registros generales.
- Registros segmentos.
- Registros puntero e índice.
- Registro IP (puntero de instrucción) y flags.

El 8086 no es una máquina ortogonal en el sentido de que no es posible utilizar cualquier registro con cualquier instrucción, tampoco es una máquina completamente dedicada, ya que la noción de acumulador está ampliamente extendida. El compromiso al que se ha llegado, es que ciertos registros se dedican para ciertas operaciones,estando libre para otras.

Así, el registro CX es un registro general pero se utiliza como contador en las operaciones repetitivas. Este esquema presenta varias ventajas, primero una codificación más densa de las instrucciones: no es necesario codificar los registros en la instrucción. Seguidamente, la programación en ensamblador conllevando una regla en la elección de los registros, si se quiere codificar de una manera metódica, la elección es

directamente impuesta en el 8086 y codificada en las instrucciones máquina, dando una universalidad de empleo a los registros.

El 8086 contiene 14 registros de 16 bits. Algunos pertenecen a la E.U. y otros a la B.I.U.

Los de la E.U. se suelen usar para direccionamientos.

- Registros de la E.U.:
  - Registros generales: AX, BX, CX, DX.
  - Registros punteros e índices: SP, BP, SI, DI.
  - Registros indicadores (flags).
- Registros de la B.I.U.:
  - Registros segmentos: CS, DS, SS, ES.
  - Registro puntero de instrucción (IP).

I-2-1.- Registros generales:

Hay cuatro registros generales de 16 bits, que pueden subdividirse (y direccionados separadamente) en registros de 8 bits:

- AX: acumulador, registro de 16 bits. Este registro se puede dividir en dos registros de 8 bits cada uno.
  - AH: registro de 8 bits, contiene los 8 bits de mayor peso del acumulador (AX).

AL: registro de 8 bits, contiene los 8 bits de menor peso del acumulador (AX).

Se dedica a las operaciones de multiplicar byte, dividir byte, E/S byte, traducción, aritmética decimal.

- BX: base, registro de 16 bits. Este registro se puede dividir en dos registros de 8 bits cada uno.

BH: registro de 8 bits, contiene los 8 bits de mayor peso de la base (BX).

BL: registro de 8 bits, contiene los 8 bits de menor peso de la base (BX).

El registro BX se ultiliza como registro base para los direccionamientos a memoria.

- CX: contador, registro de 16 bits. Este registro se puede dividir en dos registros de 8 bits cada uno.

CH: registro de 8 bits, contiene los 8 bits de mayor peso del contador (CX).

CL: registro de 8 bits, contiene los 8 bits de menor peso del contador (CX). Se dedica a las operaciones de desplazamiento y rotación variable.

El registro CX se usa amenudo para almacenar datos, para operaciones con cadenas, para bucles (LOOPS).

- DX: dato, registro de 16 bits. Este registro se puede dividir en dos registros de 8 bits cada uno.

  DH: registro de 8 bits, contiene los 8 bits de mayor peso del registro dato.

  DL: registro de 8 bits, contiene los 8 bits de menor peso del registro dato.

El registro DX se utiliza para multiplicar palabras, dividir palabras, direcccionamiento indirecto de puertos de E/S.

I-2-2.- Registros puntero e índice:

Son dos registros índices y dos punteros de 16 bits cada uno.

- SI: registro índice fuente.
- DI: registro índice destino.
- SP: registro puntero de pila.
- BP: registro puntero de base.

SI y DI son dos registros índices que contienen los desplazamientos de los datos a partir de los registros de base ES y DS. SI y DI implican en

general la utilización de DS. Sin embargo, en el caso de las instrucciones con cadenas de caracteres, SI se asocia a la zona apuntada por DS, y DI a la zona destino apuntada por ES. BP sirve de registro base para direccionar la pila. SP y BP se asocian al registro segmento de pila (SS).

I-2-3.- Registros segmento:

Hay cuatro registros segmento de 16 bits cada uno.

- CS: registro segmento código. Las direcciones son buscadas desde este segmento.
- DS: registro segmento dato. En general contiene variables de programa.
- SS: registro segmento de pila. Las operaciones de pila son ejecutadas sobre localizaciones sobre este segmento.
- ES: registro segmento extra. Es típicamente usado para almacenar datos.

El megabyte del espacio de memoria está dividido en segmentos lógicos de 64 kbyte cada uno. La CPU tiene acceso directo a los cuatro segmentos a la vez; sus direcciones bases (localizaciones de comienzo) están contenidas en los registros segmento.

Los registros son accesibles en programas y pueden ser manipulados con diversas instrucciones.

I-2-4.- Registro puntero de instrucción (IP) y flags:

I-2-4-1.- Registro puntero de instrucción:

El registro IP es de 16 bits y es análogo al contador de programa (PC) del 8085.

El IP es actualizado por la B.I.U. así como el contenido del offset de la próxima instrucción del comienza del CS (segmento código).

Durante una ejecución normal, IP contiene el offset de la próxima instrucción a ser buscada por el B.I.U.

Los programadores no tienen acceso directo al IP, pero las instrucciones le hacen cambiar y ser guardado y restaurado desde la pila (stack).

I-2-4-2.- Flags:

Se dividen en dos:

A- Flags de estado.

El 8086 tiene seis flags de estado de un bit cada uno, en los que la E.U. registra el resultado de una operación aritmético-lógica.

- CF: flag de acarreo. Si es "1", ha habido un acarreo o un acarreo negativo, en el bit de mayor orden (8 ó 16 bits). Se usa para sumas, restas y multiplicación de números. Instrucciones de rotación pueden también separar un bit en memoria o un registro colocándolo en el flags de acarreo.
- PF: flag de paridad. Si es "1", el resultado tiene paridad par, un número par de unos. Este flag puede usarse para comprobar errores en la transmisión de datos.
- AF: acarreo auxiliar. Si es "1", ha habido un acarreo en el cuarto bit del byte de menor peso o de mayor peso. Es usado por instrucciones aritméticas decimal.
- ZF: flag de cero. Si es "1", el resultado de la operación ha sido cero.

- SF: flag de signo. Si es "1", el bit de mayor orden del resultado es uno. Los números binarios negativos son representados en complemento a dos, SF indica el signo del resultado (0=positivo, 1=negativo).

- OF: flags de desbordamiento (overflow). Si es "1", indica un rebosamiento de capacidad para las operaciones aritméticas con signo.

B- Flags de control.

Hay tres flags de control disponibles, que pueden ser puestos a "1" o a "0" por programa para alterar las operaciones del procesador.

- IF: flag de interrupción. Autoriza o prohibe globalmente las interrupciones externas. Si es "1", permite a la CPU reconocer una interrupción (mascarable) requerida externamente.

Si es "0", no se permite esa interrupción.

IF no afecta a ninguna interrupción generada internamente o a ninguna no enmascarable externa.

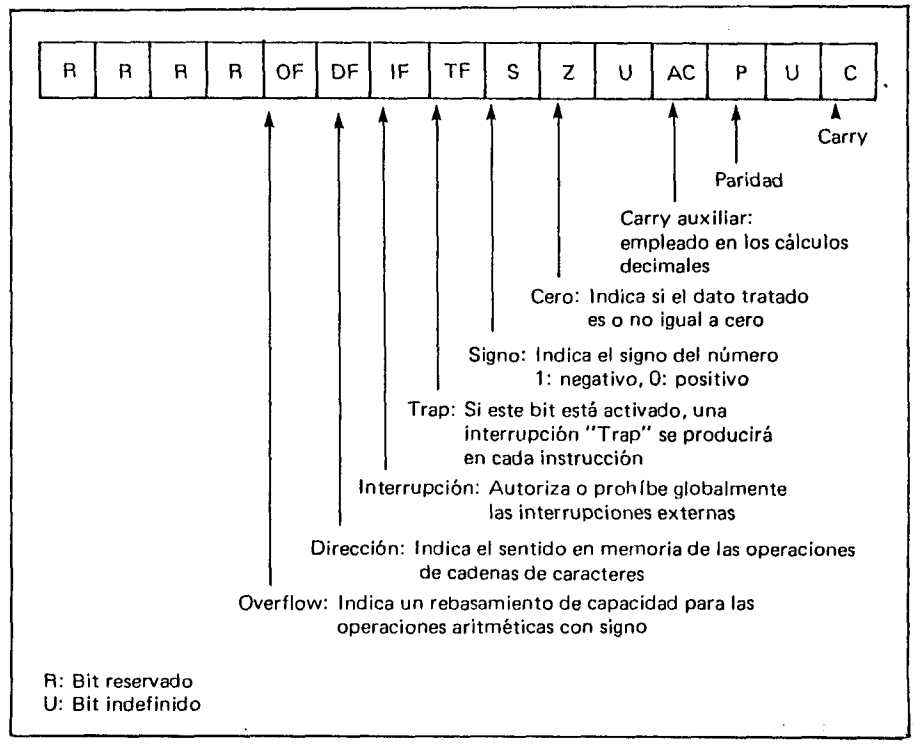
- DF: flag de dirección. Indica el sentido de desplazamiento en operaciones sobre cadenas. Si es "1", causa que las instrucciones de cadenas se auto-decrementen; eso es, un proceso de cadenas de una dirección alta a una baja, o de

derecha a izquierda.

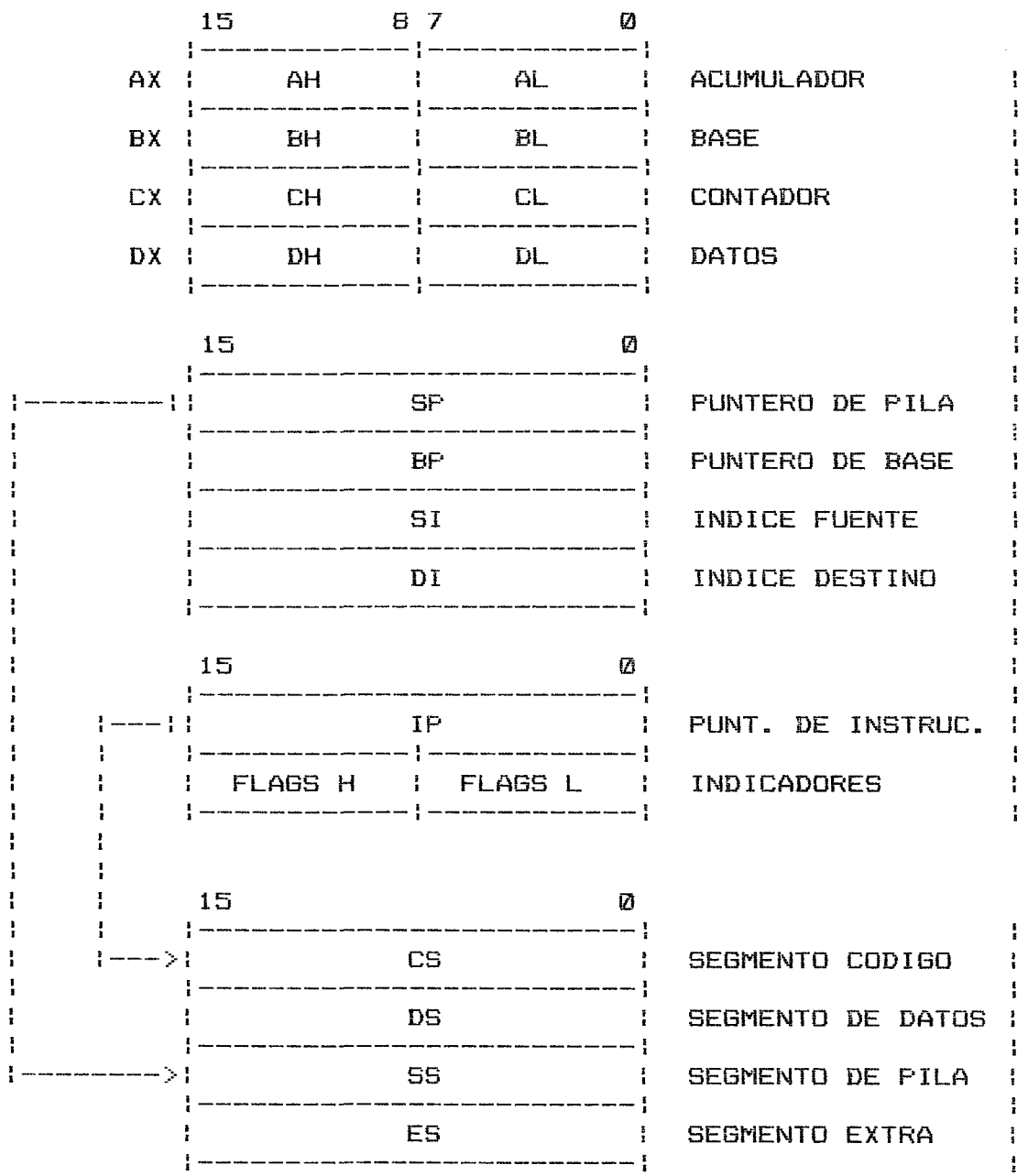
Si es "0", causa que las instrucciones de cadenas se auto-incrementen, o un proceso de cadenas de izquierda a derecha.

- TF: flag trap. Si este bit está activado, una interrupción trap se producirá en cada instrucción. Es decir, pone al procesador en un modo single-step para depuración. En este modo la CPU genera automáticamente una interrupción externa después de cada instrucción, permitindo examinar un programa y ejecutar instrucción a instrucción.

La siguiente figura nos muestra el registro de flags del 8086:



En la siguiente figura podemos ver el juego de registros del 8086:



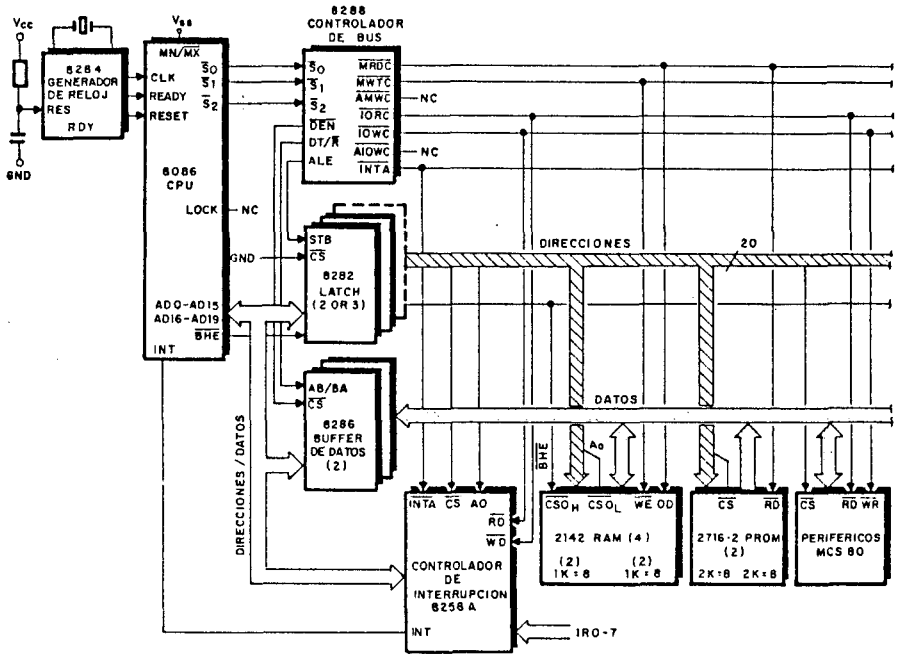
I-3) MODOS DE SELECCION.

El procesador 8086 tiene un pin (pin 33) que permite conectarlo de dos formas diferentes:

- En modo máximo.
- En modo mínimo.

I-3-1.- En modo máximo:

En este modo se conecta el pin 33 (MX) a tierra. En la figura vemos la configuración del 8086 en este modo.



Esta es la configuración que debe tener el 8086 si se quiere trabajar con el procesador de datos numéricos 8087 y el procesador de E/S 8089.

El acceso al bus del sistema (multibús), se efectúa a través de los chip adicionales 8289/8288, controladores de bus, necesarios para generar el conjunto completo de señales de control del bus.

Con esto el procesador ya no genera directamente el bus de control, pero saca tres líneas de control que son S0, S1 y S2, que se decodifican e interpretan por el 8288 para generar un bus de control compatibles con las especificaciones del multibús.

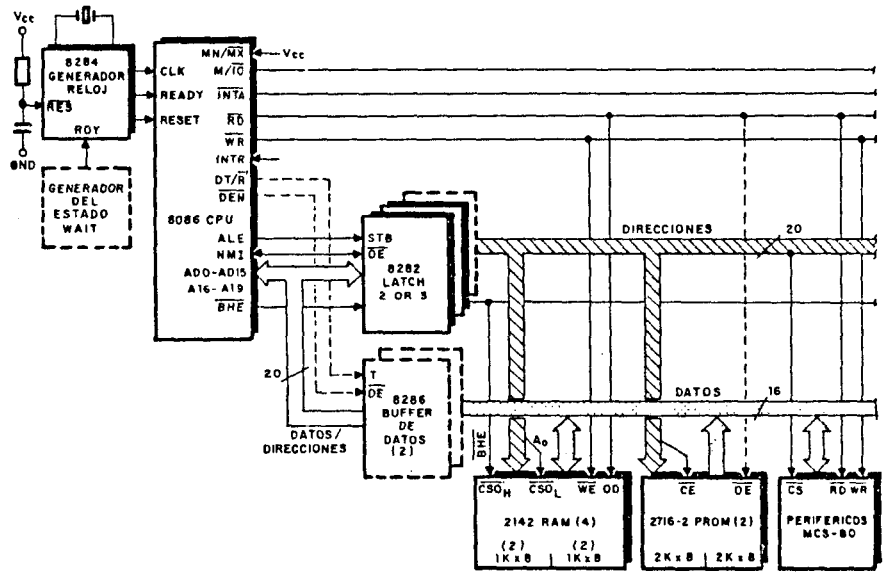
I-3-2.- <u>En modo mínimo:</u>

En este modo se conecta el pin 33 (MN) a la tensión de alimentación (Vcc = 5V.).

En esta configuración el 8086 trabaja de una forma más autónoma, y es el que genera los tres buses: el de dirección, control y datos.

En este caso el 8086 no puede utilizar los coprocesadores asociados para ser empleados en un sistema multiprocesador.

En la figura vemos la configuración del 8086 en este modo:



II) MEMORIA.

II-1) CAPACIDAD DE DIRECCIONAMIENTO.

El 8086 tiene un bus de direccionamiento de 20 bits, lo que le provee de una capacidad de direccionamiento de un megabyte de memoria:

2^{20} = 1.048.576 bytes de memoria = 1 Megabyte

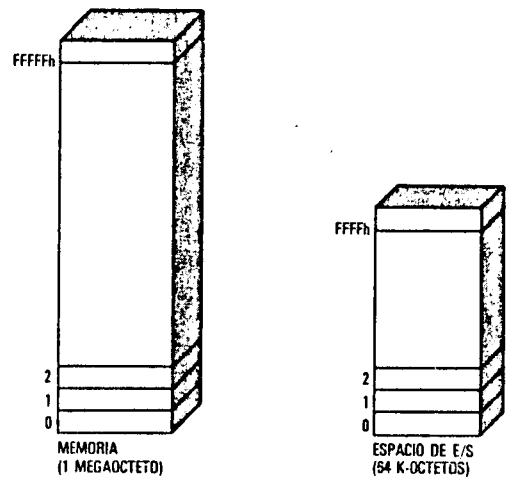
Pero el registro de direccionamiento del 8086 tiene únicamente una amplitud de 16 bits. Que corresponde a :

2^{16} = 64 kbytes

Este procesador usa un método llamado "segmentación" para permitir el direccionamiento a todo el megabyte de memoria.

El 8086 tiene otra memoria separada, llamada "espacio de E/S", que puede considerarse como una memoria extra para direccionar, en la cual se encuentran los dispositivos de E/S (cableadas las direcciones).

En la siguiente figura se muestran la memoria y el espacio de E/S:

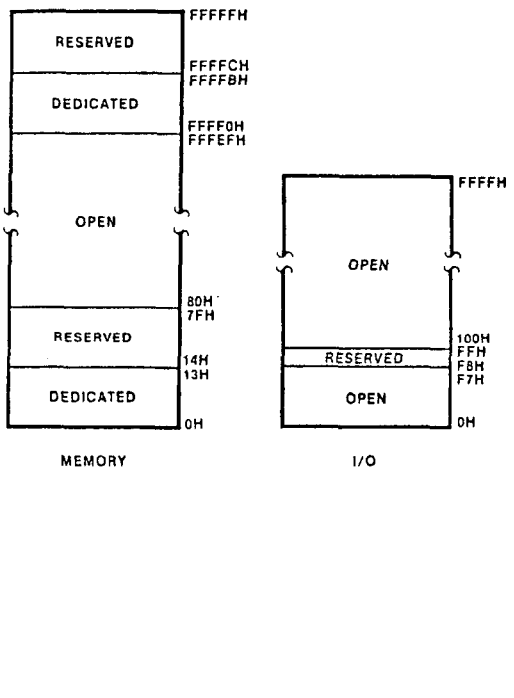


En el 8086, los dispositivos pueden conectarse a la memoria principal o al espacio de E/S. Así mismo, la memoria puede conectarse a la memoria principal o al espacio de E/S. El espacio de E/S usa un direccionamiento de 16 bits, que permiten direccionar 64 kbyte.

El direccionamiento de 16 bits, posibilita el poner muchas cosas en el espacio de E/S, incluyendo entre ellas, la memoria asociada para el IOP-8089, así como el controlador de aparatos de E/S.

En la memoria y en le espacio de E/S hay unas zonas dedicadas a especificar las funciones del procesador o reservadas para usarse por productos hardware o software de Intel.

En la siguiente figura se ven las zonas reservadas de la memoria y del espacio de E/S:

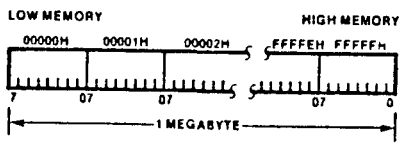


Las áreas de memoria 0H a 7FH (128 bytes) y FFFF0H a FFFFFH (16 bytes), son usadas para interrupciones y procesamiento de System Reset del 8086.

II-2) ORGANIZACION DEL ALMACENAMIENTO.

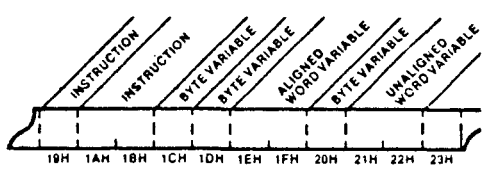
El megabyte de memoria está organizado lógicamente en 512 k palabras. La memoria se divide físicamente en dos conjuntos de 512 kbyte (idénticos), a los que se accede por parejas de 16 bits.

Organización del almacenamiento:



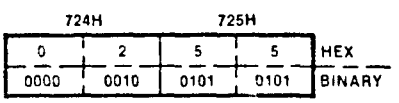
Las instrucciones, datos byte y datos palabras, pueden ser libremente almacenados en cualquier byte de dirección, sin considerar la alineación, con lo cual se aprovecha el espacio de memoria para permitir un código densamente empaquetado en la memoria.

Almacenamiento variable e instrucción:



Las direcciones impares (no alineadas) de palabras variables, sin embargo, no toman ventaja de la capacidad del 8086 de transferir 16 bits al mismo tiempo.

Siguiendo el convenio de Intel, la palabra dato siempre es almacenada con el byte más significativo en la dirección alta de la localización de memoria. A continuación vemos como se almacena una palabra variable:



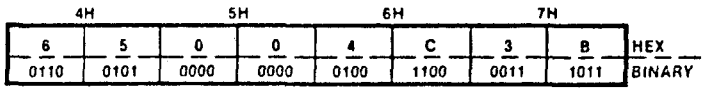
Valor de la palabra almacenada en la posición 724H : 5502H.

La mayor parte del tiempo este convenio es invisible a cualquier trabajo con el procesador.

Una clase especial de datos es almacenada como doble-palabra (ej: dos palabras consecutivas).

Estas son llamadas punteros, y son usadas en direcciones de datos y códigos que son sacados al actual segmento direccionado. La dirección baja de un puntero contiene el valor del offset, y la dirección alta de palabra contiene una dirección del segmento base. Cada palabra es almacenada

convencionalmente con el byte alto de dirección conteniendo los 8 bits más significativos de la palabra. En la siguiente figura se ve como se efectúa el almacenamiento de punteros variables:



Valor del puntero almacenado en 4H:

Dirección del segmento base: 3B4CH

Offset: 65H

II-3) SEGMENTACION.

Es un método de acceso a memoria en el cual toda dirección se compone de dos cantidades. Un identificador de segmento y un desplazamiento.

El identificador de segmento apunta a un área general de memoria (el segmento), mientras que el desplazamiento apunta a una dirección dentro de ese segmento.

El desplazamiento es el valor que aparece en los programas como dirección del dato, o como etiqueta, o como dirección de una instrucción del programa;

mientras el identificador de segmento aparece sólo cuando se ha de cambiar de segmento. Es decir, una vez seleccionado el segmento, el programador y el procesador deben recordar en que segmento se hallan, para poder interpretar las direcciones del programa como direcciones dentro de ese segmento, hasta que se especifique un nuevo segmento. En ese momento se establece un nuevo identificador. Normalmente los cambios de segmento se hacen al comienzo del programa, cuando se debe acceder a áreas de datos distintas, o cuando se pasa control de una sección de código a otra.

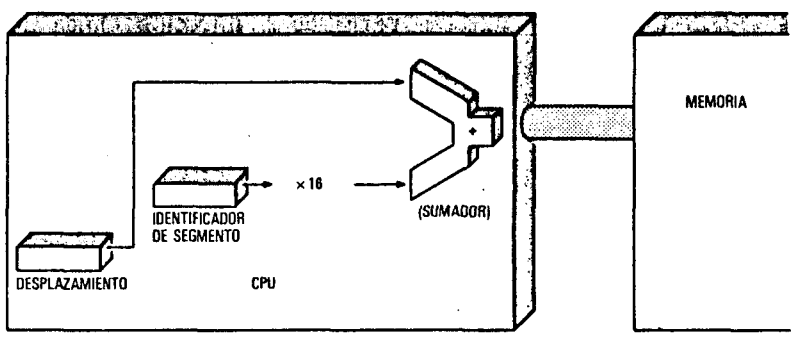
En las microcomputadoras de 8 bits, programación en lenguaje ensamblador, cada acceso a memoria corresponde a un desplazamiento, y toda la memoria es de hecho, un único y gran segmento. En los procesadores de 16 bits, el espacio de direccionamiento es tan grande que el rango de direcciones accesibles al programador (algunas veces sólo 64 kbyte) suele ser inadecuado para cubrir toda la memoria disponible. La segmentación permite que el programador especifique sus propios rangos dentro del espacio direccionable real. En vez de tener un sólo rango, puede ser interesante definir varios rangos

de direccionamiento. En el 8086 existen cuatro áreas de este tipo llamadas "segmentos". Estas áreas vienen definidas por cuatro registros especiales llamados registros de segmentación.

Estos registros son : CS, DS, ES y SS.

Los contenidos de estos registros (identificadores de segmento) se multiplican por 16 y se suman a la información proveniente del resto de la CPU (el desplazamiento) para calcular las direcciones de memoria real.

A continuación vemos como se realiza la gestión de memoria en la segmentación, para hallar la dirección de memoria real:



En el 8086, los segmentos pueden comenzar en cualquier frontera de 16 bits, y acabar en cualquier posición (hasta 64 kbytes más adelante). Luego el espacio de memoria de 1 megabyte, se describe como un grupo de segmentos que se definen para cada aplicación.

Con 20 bits, se pueden direccionar 2^{20} = 1 megabyte. Con la técnica de la segmentación, este espacio total se divide en trozos de 64 kbytes , que reciben el nombre de segmentos.

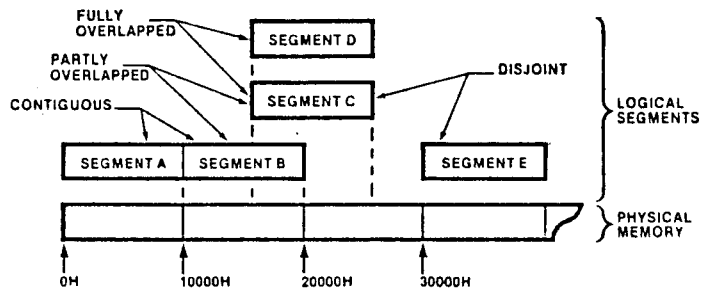
Estos registros segmentos actúan como registros base para obtener la dirección de memoria. Al multiplicarlos por 16, lo que en binario significa añadirles cuatro ceros a la derecha y convertirlos en una magnitud de 20 bits, se les suma un desplazamiento con lo que se obtiene la dirección de memoria real:

Dirección real de memoria = registro segmento * 16
+ desplazamiento.

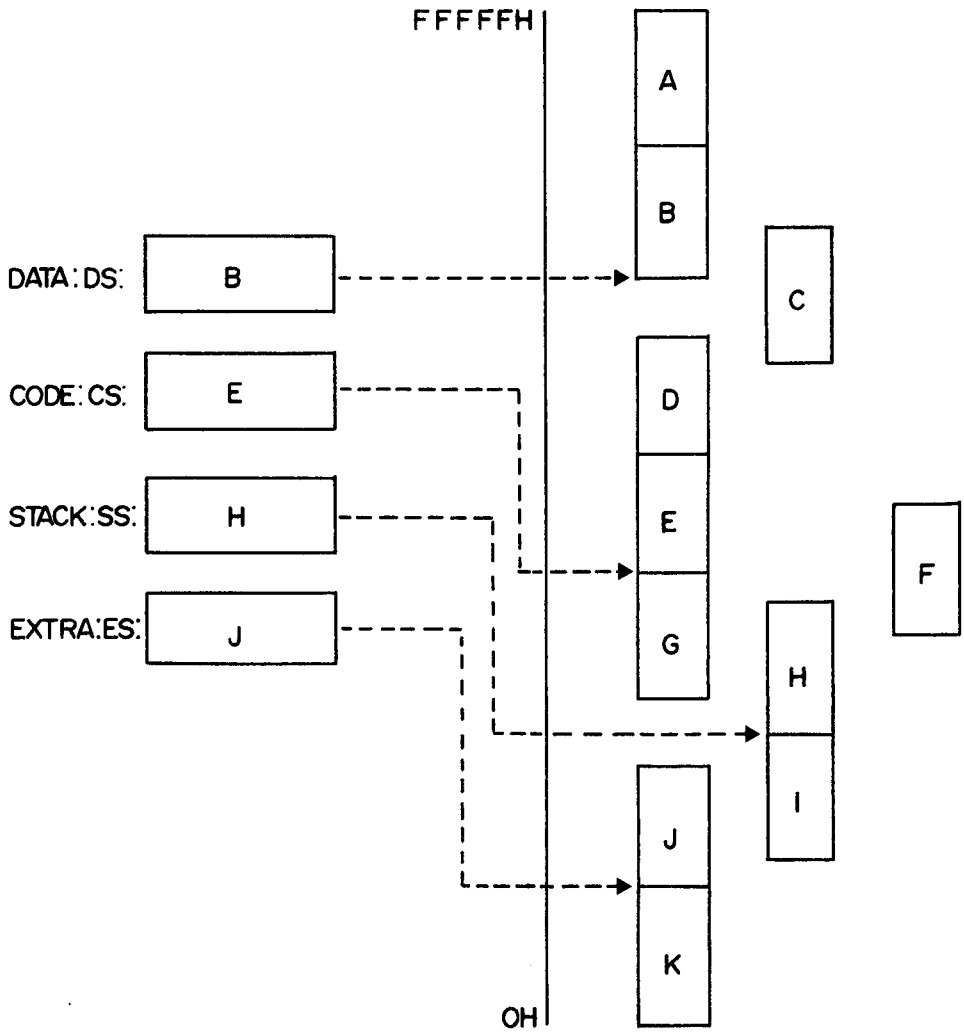
Los segmentos de 64 kbytes pueden estar separados entre si, ser adyacentes o superpuestos.

A cada segmento se le asigna (por software) una dirección base, la cual es la localización de comienzo del espacio de memoria.

En la figura vemos como pueden estar los segmentos en la memoria física:



Luego una posición física de memoria puede estar contenida dentro de uno o más segmentos lógicos. Como se ve en la siguiente figura, los registros segmentos, que contienen los valores de la dirección base, apuntan a las cuatro direcciones actuales de los segmentos:



Los programadores obtienen acceso a códigos y datos en otros segmentos, cambiando el valor de los registros segmentos por los deseados.

En muchas aplicaciones se pueden escribir con una simple inicialización y olvidarse de ellos.

II-4) GENERACION DE LA DIRECCION FISICA.

Usualmente cada localización de memoria tiene dos clases de direcciones, física y lógica.

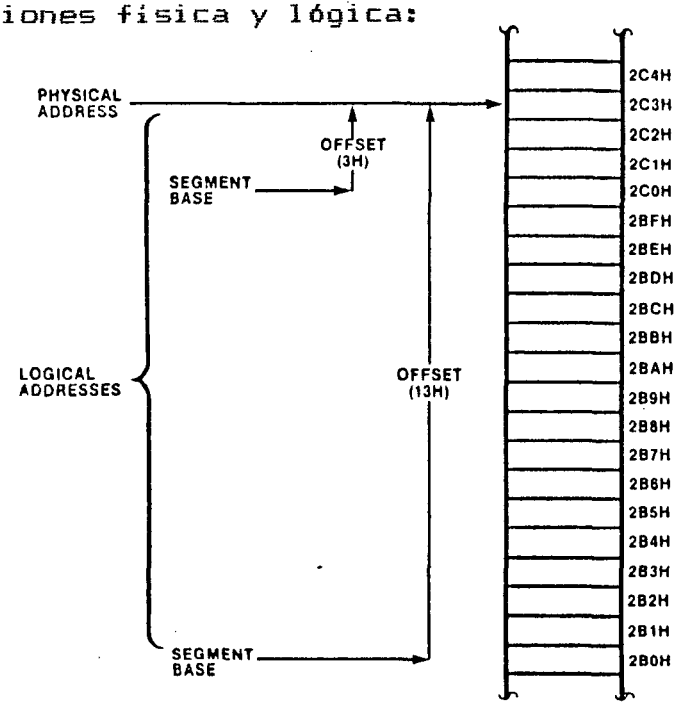
Una dirección física es de 20 bits únicos que identifican cada byte en el espacio de memoria de 1 megabyte. La dirección física va de 0H a FFFFFH. Todos los cambios entre la CPU y la memoria usan esta dirección física.

Mejor es la dirección lógica, que permite desarrollar códigos sin conocimiento previo de donde ha de ser localizado el código en memoria y facilita el conocimiento dinámico de los recursos de la memoria. Una dirección lógica consiste en un valor de segmento base y un valor de offset (desplazamiento). Para obtener cualquier localización de memoria, el valor del segmento base se localiza en el primer byte del contenido del segmento y el valor del offset es la distancia, en byte, de la posición del objetivo desde el comienzo del segmento.

El segmento base y el valor del offset son cantidades sin signo de 16 bits; el byte de menor orden en una dirección de segmento tiene un offset de cero.

Diferentes direcciones lógicas pueden ser mapeadas a la misma posición física, como se ve en la siguiente figura.

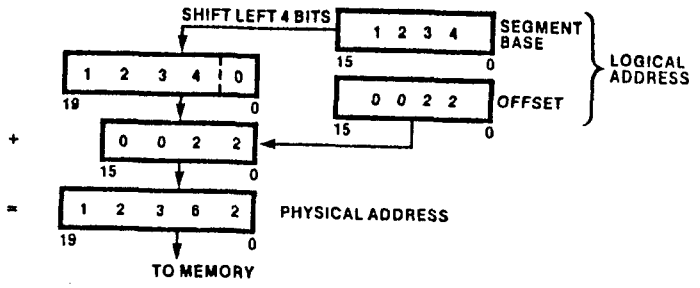
Direcciones física y lógica:



En esta figura, la posición física de memoria 2C3H está contenida en dos diferentes segmentos superpuestos, uno empezando en la 2B0H y el otro en la 2C0H.

Siempre que la B.I.U. accede a memoria, a buscar una instrucción o a obtener una variable almacenada, es generada una dirección física desde una dirección lógica. Esto se hace desplazando el segmento base cuatro bits a la izquierda y sumando el offset.

A continuación se muestra un ejemplo de generación de una dirección física:



Nótese que este proceso es proporcionado para módulos direccionados de 64 kbytes.

La B.I.U. obtiene la dirección lógica de una posición de memoria de diferentes fuentes, dependiendo del tipo de referencia hecha.

Fuentes de dirección lógica:

TIPO DE REFER. A MEMORIA	SEG. BASE	SEG. BASE ALTERNATIVO	OFFSET
Instrucción Fetch	CS	Ninguno	IP
Operación Stack	SS	Ninguno	SP
Variable	DS	CS,ES,SS	Dir.efect.
Fuente cadena	DS	CS,ES,SS	SI
Destino cadena	ES	Ninguno	DI
BP usado como registro base	SS	CS,DS,ES	Dir.efect.

Siempre la instrucciones son buscadas (fetch) desde el CS actual; IP contiene el offset de la instrucción objetivo desde el comienzo del segmento.

Las instrucciones de la pila (stack) siempre operan sobre el SS; SP contiene el offset de la cima de la pila.

Muchas variables (operandos de memoria) son asumidas a residir en el actual DS, aunque un programa puede instruir al B.I.U. a acceder a una variable en una de las otras direcciones de segmentos actuales. El offset de una variable de memoria es calculada por la E.U.

Este calculo está basado en uno de los modos de direccionamiento especificado en la instrucción, el resultado es llamado dirección efectiva del operando (EA).

Las cadenas son direccionadas diferentemente que las otras variables. El operando fuente de una instrucción de cadena está asumido a estar en el actual DS, pero pueden especificarse otros segmentos actualmente direccionados. El offset se toma del SI.

El operando destino de una instrucción de cadena siempre reside en el actual ES. El offset se toma del DI.

Las instrucciones de cadena ajustan automáticamente SI y DI, según procesen un byte o una palabra al mismo tiempo.

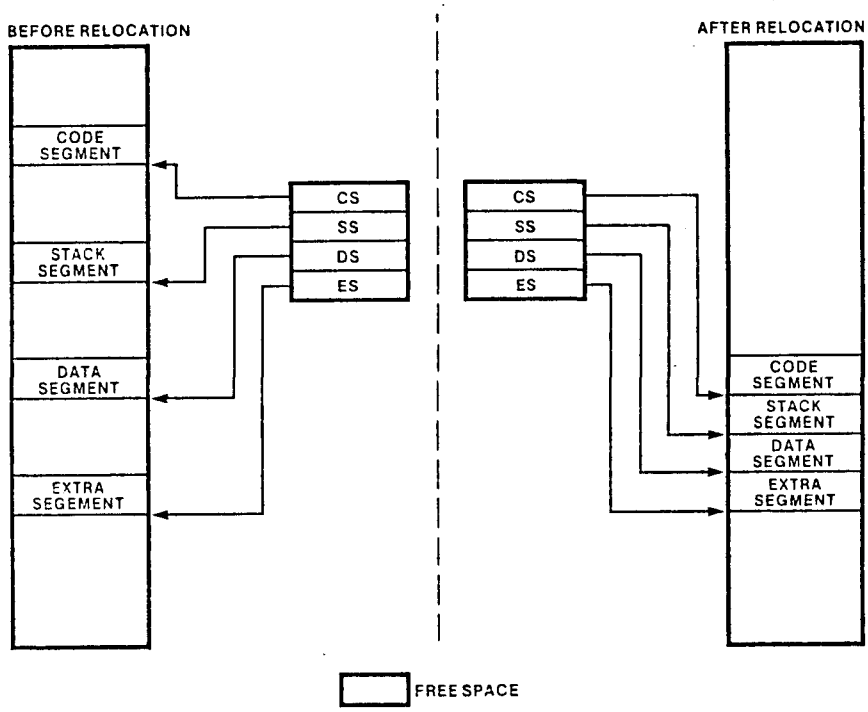
Cuando el registro BP, es designado como un registro base en una instrucción, la variable es asumida a residir en el actual SS. Así el registro BP proporciona un camino conveniente a la dirección del dato sobre el stack. BP puede ser usado, sin embargo, para meter el dato en cualquiera de los otros segmentos actualmente direccionados.

En la mayoría de los casos, los segmentos de la B.I.U. se ponen a conveniencia del programador. Esto es posible, sin embargo, por un programador para explicitar directamente el acceso de la B.I.U. a una variable en cualquiera de los segmentos actualmente direccionados (la única excepción es el operando destino de una instrucción de cadena, el cual debe estar en el ES). Esto se hace precediendo una instrucción con un segmento prefijado. esta instrucción máquina le dice al B.I.U. que registro segmento es usado para acceder a la variable referida en la siguiente instrucción.

II-5) CODIGO DINAMICAMENTE RELOCALIZABLE.

La estructura de memoria segmentada del 8086 hace posible escribir programas que están en posiciones independientes, o dinámicamente relocalizables. La relocalización dinámica permite a un sistema multiprograma o multitarea hacer particularmente efectiva el uso de la memoria disponible. Programas inactivos pueden ser pasados a disco y el espacio que ellos ocupan puede ser designado a otros programas. Si el disco donde reside el programa se necesita más tarde, este puede ser metido en cualquier posición de memoria.

Similarmente, si un programa necesita un gran bloque continuo de almacenamiento, y la cantidad total está solamente disponible en fragmentos no adyacentes, otros segmentos de programas pueden ser comprimidos para dejar libres los espacios continuos. Este proceso se puede ver en la siguiente figura:



Para ser dinámicamente relocalizable, un programa no debe ser cargado o alterados sus registros segmento y no debe transferirse directamente a una localización fuera del actual segmento código. En otras palabras, todo offset en el programa debe ser relativo a valores fijos contenidos en los registros segmento. Esto permite al programa ser movido de cualquier lugar de memoria, tan lejos como estén actualizados los registros segmento, al punto de nueva dirección base.

II-6) IMPLEMENTACION DE LA PILA (STACK).

La pila juega en este procesador un papel importante, ya que ha sido previsto un modo de direccionamiento que utiliza la pila mediante un registro base sobre porciones de la pila.

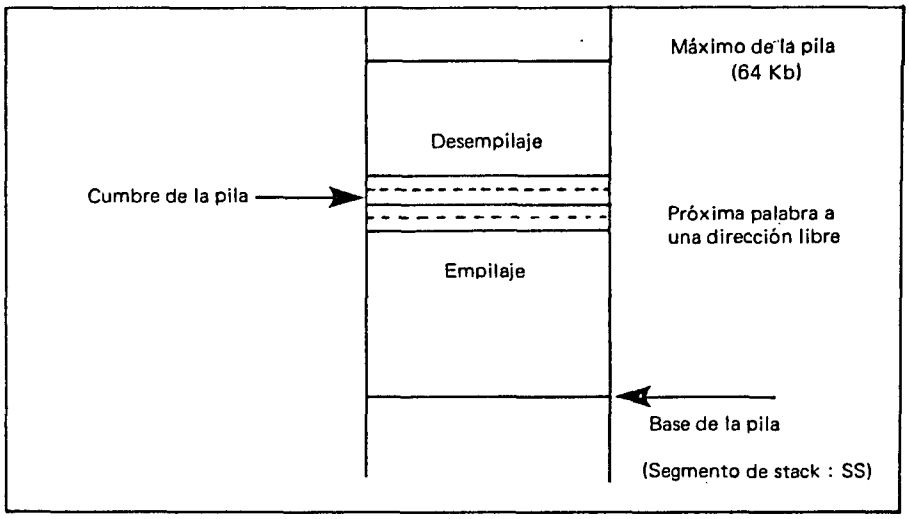
La pila en el 8086 es implementada en memoria y es localizada por el registro segmento de pila (SS) y el registro puntero de pila (SP). La pila puede tener 64 kbyte de longitud, que es la máxima longitud de un segmento.

Funciona con palabras de 16 bits y se pueden efectuar operaciones de empilar y desempilar.

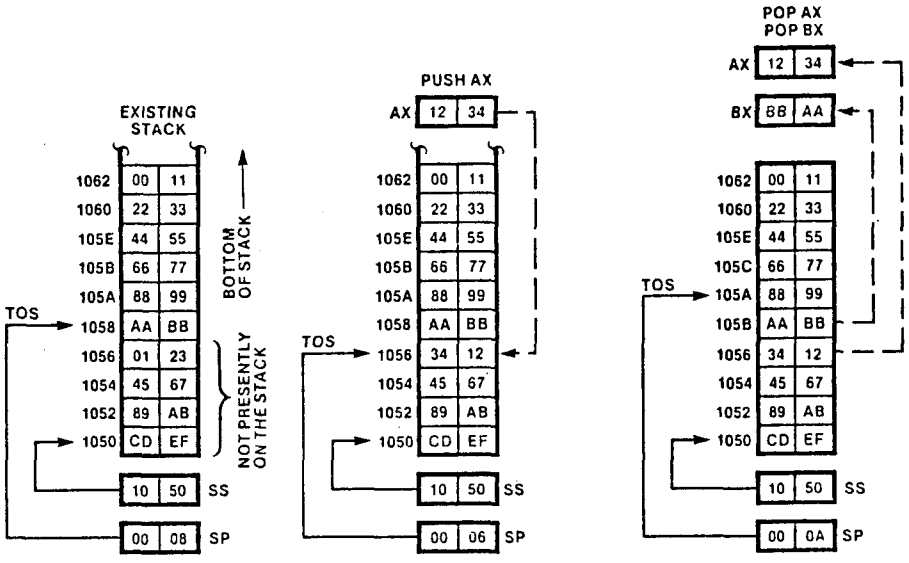
El direccionamiento físico se hace a través del registro SS y un desplazamiento contenido en el puntero de pila (SP). El puntero de pila se decrementa al empilar (PUSH) y se incrementa al desempilar (POP).

Es decir, el SP contiene el offset de la pila desde la dirección base del SS, que es el segmento identificador.

En la siguiente figura vemos el funcionamiento de la pila en el 8086:



A continuación veremos un ejemplo de operación con la pila :



En este ejemplo, el código de secuencia empleado es el siguiente:

```
PUSH  AX
POP   AX
POP   BX
```

El SS contiene la dirección base de la pila, que es la 1050H, y el SP contiene el desplazamiento desde el SS, que es el 0008H. Al hacer la instrucción PUSH AX, se mete el contenido del registro AX a partir de la dirección señalada por el SP. Una vez metido el valor del registro AX, que es 1234H, el SP se decrementa y pasa a señalar la última posición ocupada,que es la 0006H.

Ahora se hace la instrucción POP AX, entonces el contenido de la dirección señalada por el SP se lleva al AX, y el SP se incrementa señalando la última dirección metida, que es la 0008H. Si a continuación se hace la instrucción POP BX, el contenido de la dirección señalada por el SP se lleva al registro BX, que es la BBAAH. Al mismo tiempo el SP se incrementa y señala la posición 000AH.

Como vemos, el registro segmento SS no se modifica durante la operación con la pila, ya que es la dirección base.

Luego el SP es el que nos dá el desplazamiento dentro de la pila.

III) ENTRADAS/SALIDAS.

El 8086 proporciona un gran espacio de E/S que está separado del espacio de memoria, y las instrucciones que transfieren datos entre la CPU y los dispositivos localizados en el espacio de E/S. Los dispositivos de E/S también pueden ser posicionados en el espacio de memoria para dar la potencia a la puesta total de instrucciones y direccionar los modos de E/S del procesamiento.
Para altas velocidades de transferencia, la CPU puede ser usada con los tradicionales controladores de acceso directo a memoria o con el controlador de E/S 8089.

III-1) ESPACIO DE ENTRADA/SALIDA.

El 8086 permite la conexión de 64 kbyte (65.536) puertos de E/S de 8 bits o de 32 kbyte puertos de 16 bits, 32.768 puertos de 16 bits. Las instrucciones IN (entrada) y OUT (salida), transfieren los datos entre el acumulador (AL para la transferencia de bytes, AX para la transferencia de palabras) y puertos localizados en el espacio de E/S.

El espacio de E/S no es segmentado; para el acceso a un puerto la B.I.U. simplemente coloca la dirección del puerto (0-64 kbytes) sobre las 16 líneas bajas del bus de dirección. Las diferentes formas de las instrucciones de E/S permiten ser direccionadas específicamente como un valor fijo en la instrucción o como una variable tomada desde el registro DX.

III-2) LOCALIZACIONES RESTRINGIDAS DE E/S.

De la localización F8H a la FFH (8 de las 64 k posiciones) en el espacio de E/S están reservadas por Intel para ser usadas por futuros productos software o hardware de Intel. Usando estas localizaciones para cualquier otro propósito pueden dejar de ser compatibles con futuros productos de Intel.

III-3) DIFERENTES ACCESOS DE E/S AL 8086.

El 8086 puede transferir 8 ó 16 bits al mismo tiempo a un dispositivo localizado en el espacio de E/S. Un dispositivo de 16 bits debe ser localizado en una dirección par, así que la

palabra será transferida en un ciclo de bus simple. Un dispositivo de 8 bits puede ser localizado en una dirección par o impar; sin embargo, los registros internos en un dispositivo dado, deben estar asignados en todas las direcciones impares o pares.

III-4) MEMORIA-MAPEADO DE E/S.

Los dispositivos de E/S también pueden ser posicionados en el espacio de memoria del 8086. Tanto como la respuesta de los dispositivos como los componentes de memoria, la CPU no sabe la diferencia.

El mapeado de memoria de E/S proporciona programaciones adicionales flexibles. Cualquier instrucción que se refiera a la memoria puede ser usada para acceder a los puertos de E/S localizados en el espacio de memoria. Por ejemplo, la instrucción MOV (mover) puede transferir datos entre cualquier registro del 8086 y puertos, o las instrucciones AND, OR y TEST pueden ser usadas para manipular los bits en los registros de E/S del dispositivo. Además, el mapeado de memoria de E/S puede obtener ventaja de

los modos de direccionamiento de memoria del 8086. Un grupo de terminales, por ejemplo, puede ser tratado como un array en memoria, con un registro índice seleccionando un terminal en el array.

Por supuesto, debe pagarse un precio por la flexibilidad de programación que el mapeado de memoria de E/S proporciona. Dedicando parte del espacio de memoria a los dispositivos de E/S, se reduce el número de direcciones disponibles para memoria, aunque con un megabyte de espacio de memoria esto raramente sería una reducción. Las instrucciones de referencia a memoria se hacen más largas para ejecutar y son algo menos compactas que las simples instrucciones IN y OUT.

III-5) ACCESO DIRECTO A MEMORIA.

Cuando está configurado en modo mínimo, el 8086 proporciona las señales HOLD y HLDA que son compatibles con los controladores tradicionales DMA tal como el 8257 y 8237. Un controlador DMA puede requerir el uso del bus, para la transferencia directa de un dato entre un dispositivo de E/S y la memoria por la activación de HOLD.

La CPU completará el actual ciclo de bus, si uno está en marcha, y entonces HLDA se entrega y se concede el bus al controlador DMA. La CPU no usará el bus hasta que HOLD quede inactivo.

Las direcciones de memoria del 8086 están físicamente organizadas en dos bancos separados, una que contiene los bytes de dirección par y otra que contiene los bytes de dirección impar. Un controlador DMA de 8 bits debe alternativamente seleccionar estos bancos para acceder lógicamente a los bytes adyacentes en memoria. El 8089 proporciona un simple camino para interface de alta velocidad para un dispositivo de 8 bits, para un sistema basado en el 8086.

III-6) PROCESADOR DE E/S 8089 (IOP).

El 8086 está diseñado para usarse con el 8089 en ejecuciones rápidas, para aplicaciones de E/S. El 8089 conceptualmente se parece a un microprocesador con dos DMA y una puesta de instrucciones específicamente construido para operaciones de E/S. Distintos controladores simples de DMA, el 8089 puede servir directamente a dispositivos de E/S, cambian esta tarea desde la CPU.

Además, éste puede transferir datos en su propio bus o en el bus del sistema, puede seleccionar periféricos de 8 ó 16 bits o buses de 8 ó 16 bits, y puede transferir datos desde la memoria a memoria y desde el dispositivo de E/S al dispositivo de E/S.

III-7) PROCESAMIENTO PARALELO Y MULTIPROCESO.

Una importante tendencia actual es la utilización de varios procesadores en una misma computadora. Por ejemplo, la impresora, los terminales o el plotter pueden llevar incorporados sus propios procesadores.

La idea de multiproceso consiste en incorporar varios procesadores a la computadora para aumentar su potencia de cálculo. Una posibilidad es tener dos procesadores centrales idénticos en la misma computadora. El secreto para que todos contribuyan a realizar las tareas eficientemente, consiste en dividir los programas en procesos que se ejecuten en procesadores.

En un sistema de procesamiento paralelo, dos o más procesadores comparten el mismo flujo de instrucciones. Es decir, todos los procesadores

están realizando el mismo programa, turnándose en la ejecución de las instrucciones. La idea principal es que algunas instrucciones se ejecutan mejor en unos procesadores que en otros.

III-7-1.- Multiproceso:

En un sistema de multiproceso, dos o más procesadores comparten la memoria, pero operan sobre distintos flujos de instrucciones. Un procesador concreto puede tener el control, distribuyendo trabajo entre los otros procesadores a través de mensajes; pero cada uno de ellos, una vez inicializado, ejecuta independientemente del primero su propio programa. Un ejemplo es el procesador de E/S 8089 (IOP).

La ventaja del multiproceso es que el procesador central se ve liberado de una gran cantidad de trabajo, pudiéndose concentrar en la gestión global del sistema. El procesador central se limitaría a distribuir tareas entre otros procesadores especialmente diseñados para realizar estas tareas eficientemente. Por ejemplo, el IOP 8089 gestiona las transferencias de E/S mucho mejor que la CPU. Con el IOP 8089 ciertas transferencias

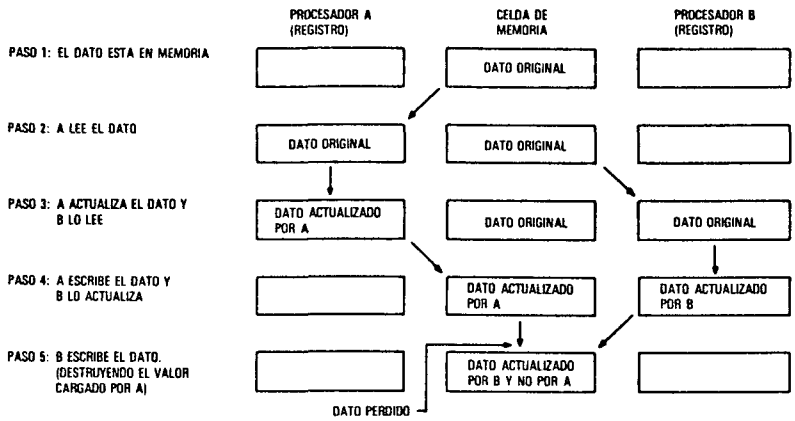
se pueden llegar a realizar 16 veces más rápidas que con la CPU.

Para controlar esta cooperación entre procesadores, se necesitan algunas instrucciones especiales.

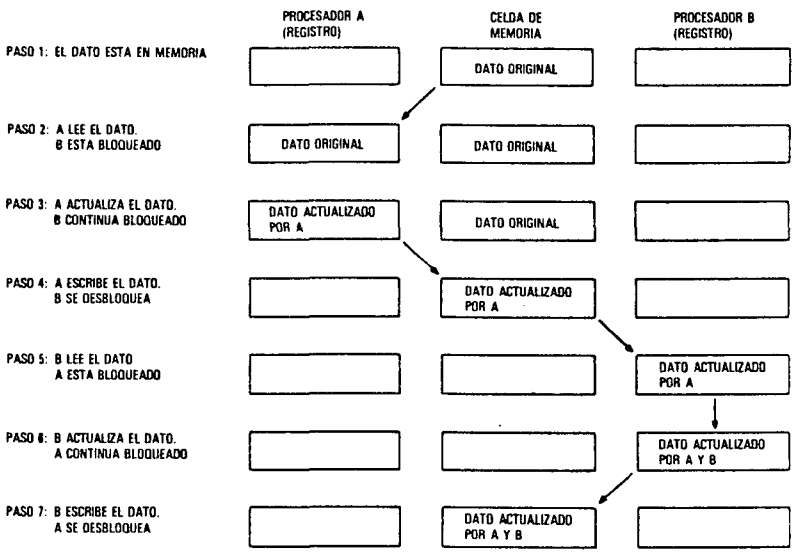
Una de las instrucciones especiales para el multiproceso es la LOCK del 8086. LOCK envía una señal al resto de los procesadores del sistema, avisándoles que no deben utilizar el bus hasta que se haya completado la instrucción siguiente. La importancia de esta instrucción, reside en el hecho de que, en multiproceso, hay que proteger los datos y programas para evitar que se acceda a ellos simultáneamente desde dos procesadores distintos. Supongamos por ejemplo, que dos procesadores intentan actualizar el mismo dato a la vez. Cada procesador leerá el dato, lo cargará en uno de sus registros, operará con él y devolverá el valor modificado a memoria. Dependiendo de quién lo haga primero el resultado será distinto. Si el procesador A lee el dato y el procesador B lee la misma posición de memoria antes de que A haya devuelto el valor, ambos procesadores contendrán el mismo valor original del dato. Si suponemos ahora que ambos procesadores operan el dato, y devuelven el

resultado a memoria, primero el A y luego el B, el resultado final es el mismo que si sólo hubiese trabajado el procesador B, ya que al devolver el valor destruye el valor devuelto por A.

En las siguientes figuras vemos este caso:



Dato sin protección al que acceden dos procesadores a la vez.

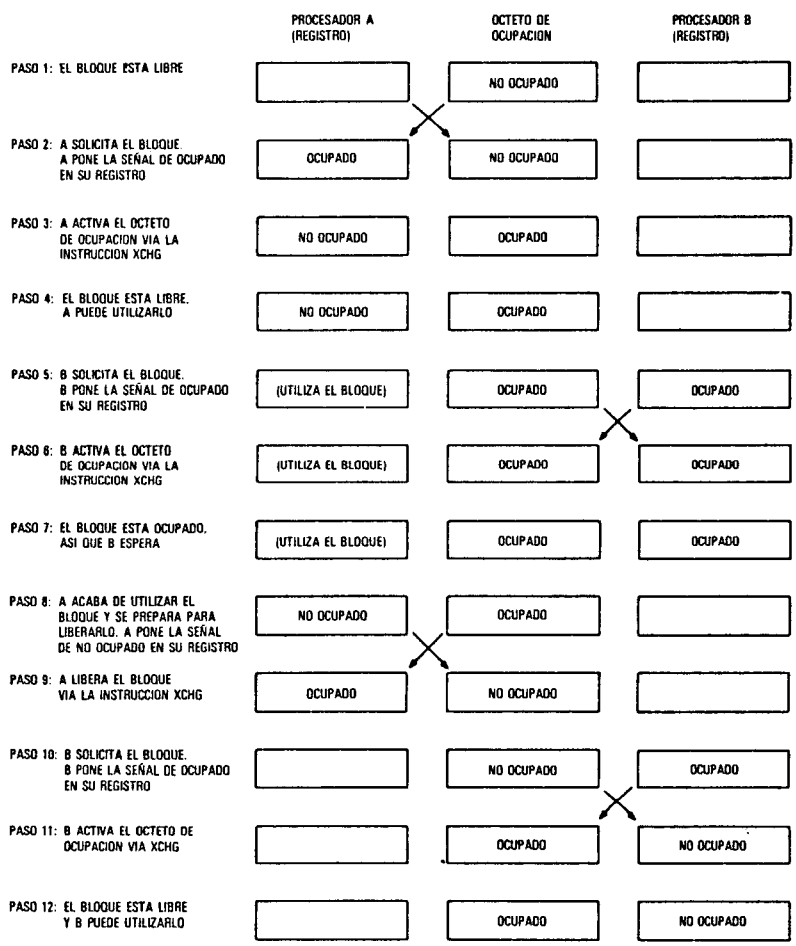


Dato protegido al que acceden dos procesadores.

Para prevenir y evitar conflictos de este tipo entre procesadores, cada bloque de memoria que se utilice para datos o programas de cualquier procesador del sistema, contiene un octeto especial llamado "octeto de ocupación".

Todo proceso que desee utilizar un bloque de memoria debe preguntar antes por su octeto de ocupación, para saber si el bloque está ocupado o no. Si lo está, el procesador activa el octeto de ocupación. El único problema surge cuando dos procesadores preguntan o intentan activar simultáneamente el octeto de ocupación. En tal caso, prodria ser que ambos procesadores accediesen al mismo bloque a la vez. La solución a este problema está en utilizar la instrucción LOCK cada vez que se accede al octeto de ocupación. La instrucción LOCK se encarga de evitar que los otros procesadores puedan hacer algo hasta que el procesador en cuestión haya activado el octeto de ocupación, cosa que puede hacerse con la instrucción XCHG. Si el bloque está libre, el procesador lo usa; y si el bloque está ocupado, el procesador espera y vuelve a preguntar más tarde.

En la siguiente figura podemos ver el uso del octeto de ocupación:



III-7-2.- Procesamiento paralelo:

La gran ventaja del procesamiento paralelo es que el procesador central se ve liberado de una gran cantidad de trabajo. El procesador central distribuirá las tareas.

Como ejemplo de este sistema tenemos el procesador de datos numéricos NDP 8087. El NDP 8087 opera con tipos de datos más largos y complicados que la CPU y aumenta en un 10 % el rendimiento del sistema.

Para permitir el procesamiento paralelo, las instrucciones se dividen en conjuntos asignados a cada procesador. Si un procesador recibe una instrucción ajena, no la intenta ejecutar sino que la devuelve al sistema para que la ejecute el procesador correcto. En consecuencia, cada procesador del sistema lleva asociado uno o más conjuntos de instrucciones ficticias. La instrucción ficticia del 8086 ESC se utiliza para definir el conjunto de instrucciones del NDP 8087, y el resto de las instrucciones del 8086 son ficticias para el 8087.

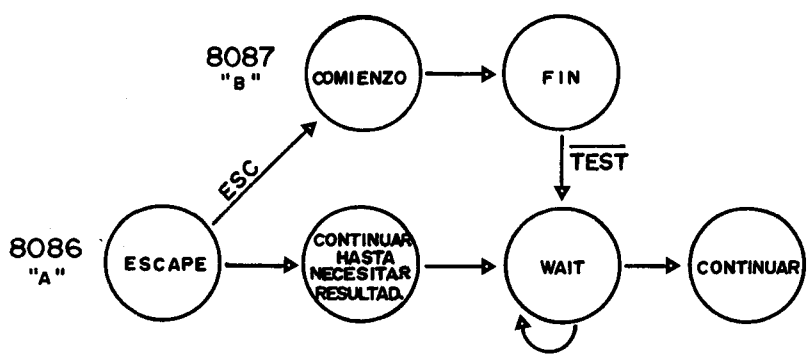
Es decir, la instrucción ESC no tiene efecto alguno sobre el 8086. El 8087 opera en paralelo con la CPU del 8086, leyendo todas las instrucciones al mismo

tiempo que este. Cuando el 8087 detecta una instrucción ESC, sabe que es para él y comienza a operar. Cuando termina la operación lo advierte a través de la activación de una señal.

El 8086, continúa ejecutando instrucciones hasta que necesita los resultados del NDP 8087, momento en el que lanza una instrucción WAIT. Esta instrucción detiene a la CPU hasta la activación de la línea TEST, momento en el que continúa. Mediante la interconexión de la señal de fin de operación del NDP 8087 y la entrada TEST, se obtiene la sincronización del procesamiento paralelo.

La sincronización del procesamiento paralelo puede realizarse totalmente por hardware en vez de la combinación hardware/software.

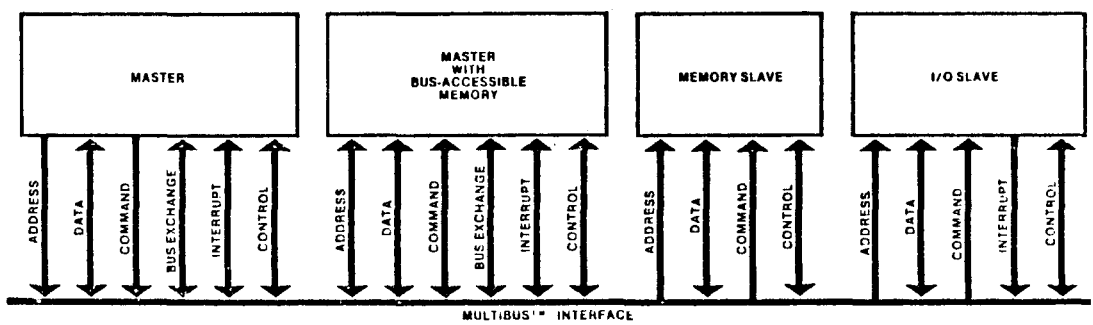
A continuación vemos la ejecución de dichas señales:



III-8) ARQUITECTURA MULTIBUS.

Intel ha diseñado un multiproceso del bus de propósito general llamado multibús. Cuando el 8086 está configurado en modo máximo, el controlador de bus 8288 saca las señales que son eléctricamente compatibles con el protocolo del multibús. Los diseñadores de sistemas de multiproceso usan la arquitectura multibús en el diseño de sus productos para reducir el coste del desarrollo y el tiempo, y para ser compatibles con la amplia variedad de placas disponibles en los productos iSBC.

La arquitectura multibús proporciona un versátil canal de comunicaciones que puede ser usado coordinadamente con la amplia variedad de modulos de computación. En la siguiente figura vemos un sistema basado en la arquitectura multibús:



Los módulos en un sistema multibús son diseñados como maestros o esclavos. Los maestros pueden obtener el uso del bus e iniciar la transferencia de datos sobre el mismo. Los esclavos son solamente los objetos de las transferencias del dato. La arquitectura del multibús permite a ambos 8 y 16 bits maestros ser entremezclados en un sistema. Además de las 16 líneas de datos, el bus diseñado proporciona 20 líneas de direccción, 8 líneas de interrupción multinivel, y líneas de control y arbitraje. Un bus auxiliar de potencia también es proporcionado para dirigir la potencia de reserva para las memorias si la alimentación normal falla.

La arquitectura multibús mantiene su propio reloj, independientemente de los relojes de los módulos linkados juntos. Estas diferentes velocidades de los maestros permite compartir el bus y permite a los maestros operar asíncronamente con respecto a cada uno de los otros. La lógica de arbitraje del bus permite velocidades lentas a los maestros para competir igualmente por el uso del bus. Una vez que el módulo ha obtenido el bus, sin embargo, las velocidades de transferencia dependen solamente de la capacidad de transmisión y recepción de los

módulos. Finalmente, el multibús estandar define la forma de los factores y los requerimientos físicos de los módulos que se comunican sobre este bus.

III-9) 8289 BUS DE ARBITRAJE.

Los sistemas de multiproceso requieren unos recursos de coordinación de los procesadores para el uso del bus compartido. El 8289 bus de arbitraje trabaja en conjunción con el 8288 controlador del bus para proporcionar este control para sistemas basados en el 8086. Esto es compatible con la arquitectura multibús y también puede ser usado en otros diseños de bus-compartido. El 8289 elimina condiciones de carrera, resuelve contenciones de bus y comparaciones de procesadores operando asíncronamente con respecto a cada uno. Cada procesador sobre el bus tiene asignada una prioridad diferente. Cuando llegan peticiones simultáneas para el bus, el 8289 resuelve la contención y concesión del bus al procesador con la más alta prioridad; tres diferentes técnicas de prioridad pueden ser usadas, ver manual del fabricante.

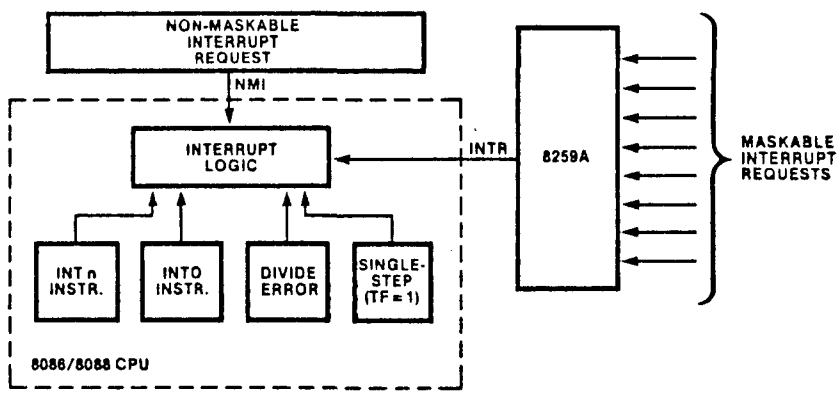
IV) INTERRUPCIONES.

El 8086 tiene un simple y versátil sistema de interrupción. A cada interrupción se le asigna un tipo de código que es identificado por la CPU.

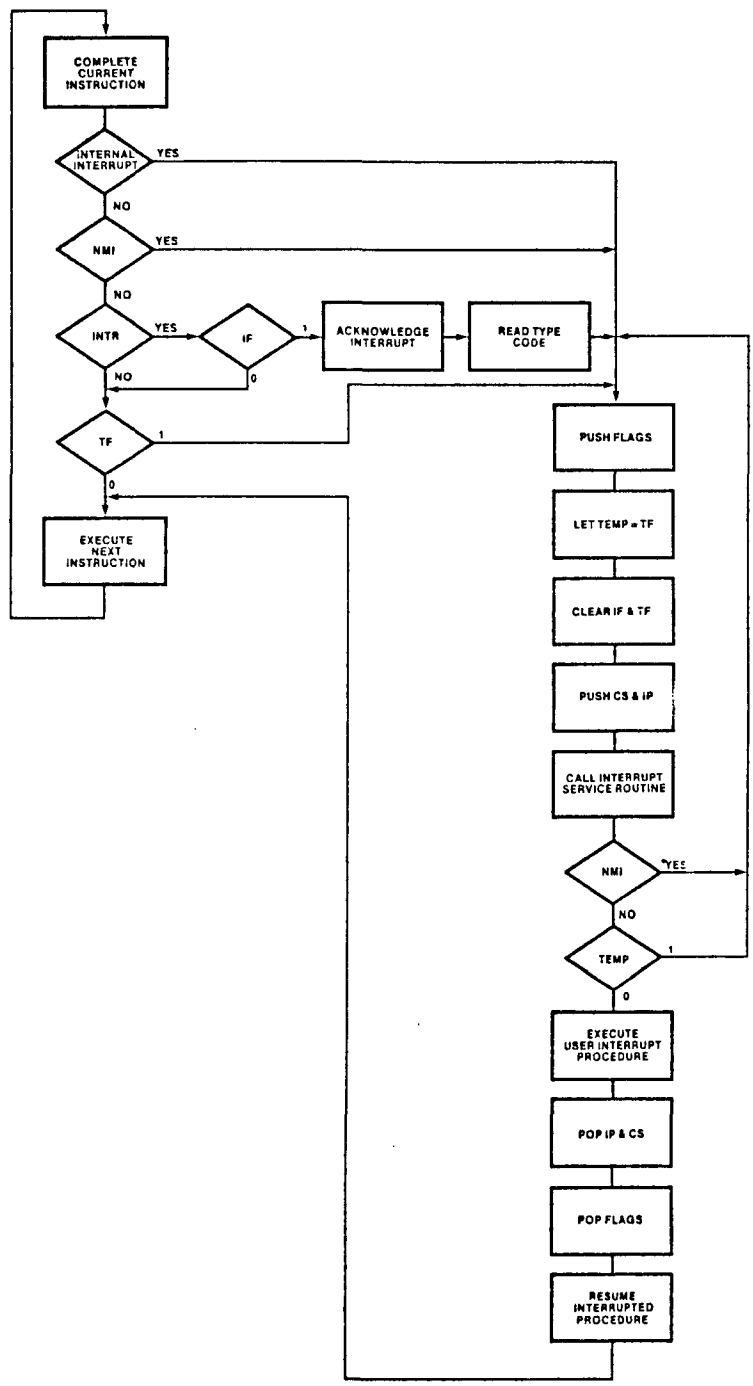
El 8086 utiliza una tabla de 256 tipos de interrupción diferentes, numerados del 0 al 255. A cada tipo le corresponde un puntero de 32 bits (4 bytes), dando la dirección del procedimiento a ejecutar.

Las interrupciones pueden ser inicializadas por dispositivos externos a la CPU; en suma, ellas también pueden ser disparadas por instrucciones de interrupción software y, bajo ciertas condiciones, por la misma CPU.

Fuentes de interrupción:



La siguiente figura ilustra la respuesta básica del 8086 a una interrupción:



Las interrupciones se clasifican según su naturaleza en:

- Interrupciones externas:

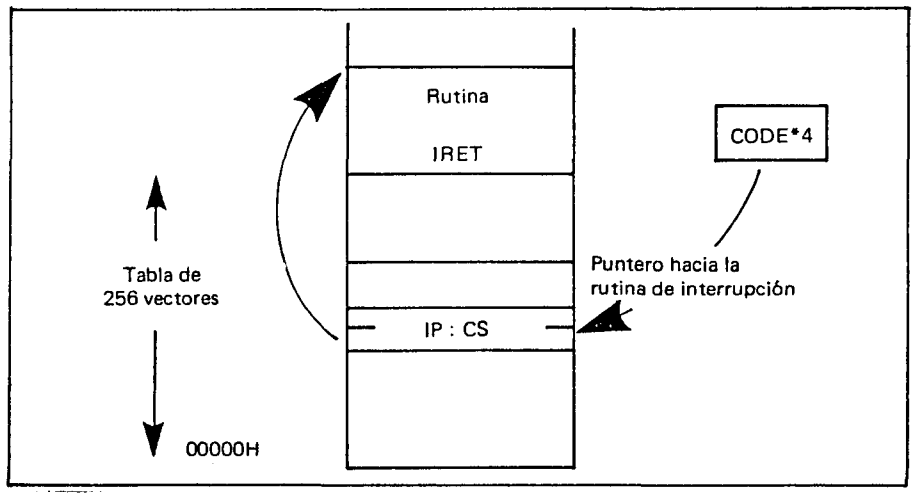
  Son aquellas provocadas por un periférico que debe suministrar al procesador un tipo de interrupción (8 bits). Un componente codificador de prioridades se emplea habitualmente para facilitar la generación de este tipo.

- Interrupciones internas:

  Son aquellas provocadas por ciertos estados del procesador o por instrucciones especiales y son vectorizadas automáticamente, es decir, el tipo de interrupción está fijado por la arquitectura del 8086.

El tipo de interrupción se lee (caso externo) o se fabrica (caso interno), a continuación se multiplica por cuatro, lo que indica de esta manera un desplazamiento en una tabla de 1020 bytes, situada en la dirección física 00000H y compuesta de 256 punteros.

Un puntero direcciona entonces una rutina de tratamiento de la interrupción situada en el espacio de 1 megabyte del procesador.



- Acceso a la rutina de interrupción.

Es decir, el desplazamiento en esta tabla de 1020 bytes, 255 punteros por 4, se obtiene:

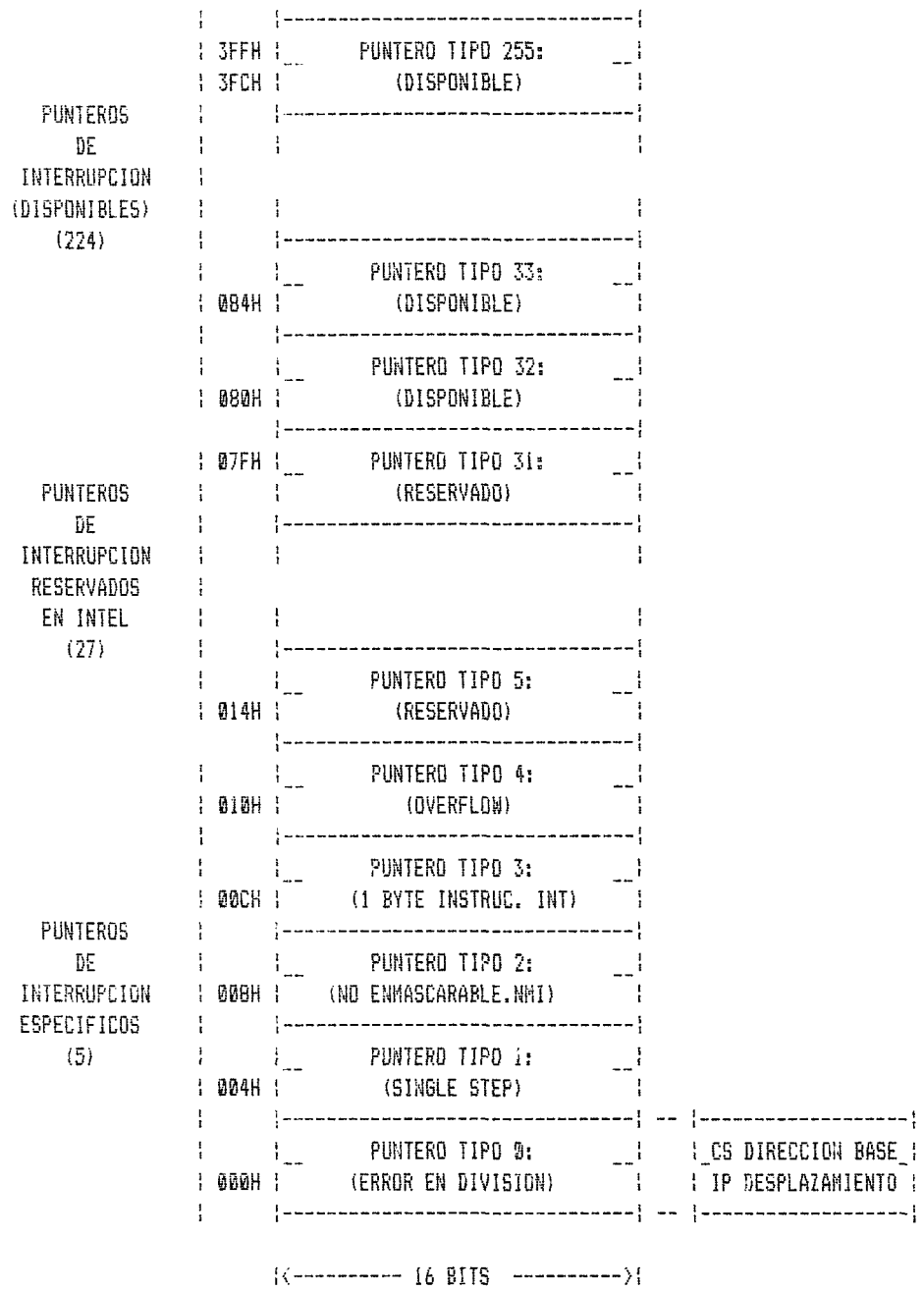
4 x n = posición en la tabla.

Siendo n el tipo de interrupción.

Luego, a cada uno de estos punteros de 4 bytes (32 bits) se le asigna un número del 0 al 255, según su posición en la memoria.

No tienen por qué cargarse todos los punteros, pero antes de que se use una interrupción de cierto tipo, el correspondiente puntero de la tabla de interrupciones deberá cargarse para su rutina de servicio.

A continuación se muestra la tabla de punteros de interrupción en el 8086:



IV-1) INTERRUPCIONES EXTERNAS.

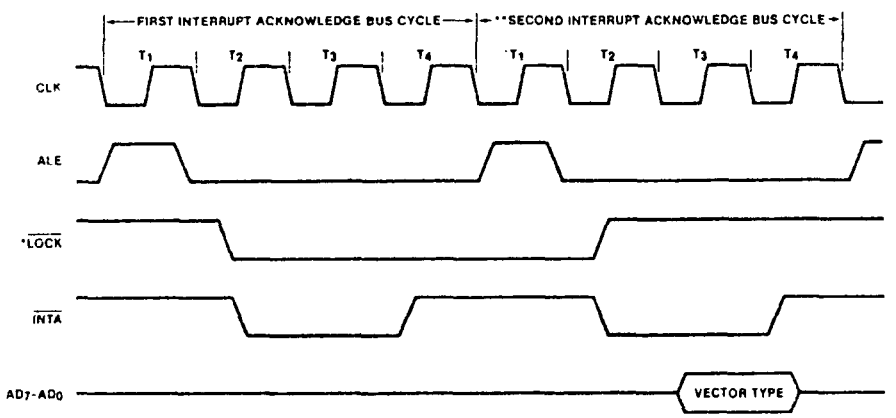
El 8086 tiene dos líneas que pueden ser usadas por los dispositivos externos para señal de interrupción:

INTR y NMI.

IV-1-1.-Interrupciones enmascarables: INTR

Las interrupciones se piden por la señal INTR (requerimiento de interrupción) que es reconocida con un nivel de +5 v. Al llegar esta señal, el procesador acaba la instrucción en curso (esto puede ser largo en el caso de instrucciones de multiplicación o división).

Como muestra la siguiente figura, cuando una interrupción enmascarable es reconocida, la CPU ejecuta dos ciclos de bus de reconocimiento de la interrupción.



*MAXIMUM MODE ONLY
**SEVERAL (3 TYPICAL) IDLE CLOCK STATES OCCUR BETWEEN THE FIRST AND SECOND INTERRUPT ACKNOWLEDGE BUS CYCLES IN THE 8086 CPU (DURING THIS INTERVAL THE BUS IS DRIVEN).

El 8086 envía a continuación dos ciclos de validación INTA (reconocimiento de interrupción). En el primer ciclo INTA,durante los estados T1 a T2, el Bus está libre, el procesador efectúa dos ciclos IDLE (desocupado). En el modo mínimo, la CPU no reconocerá el requerimiento hold desde otro bus maestro hasta que la secuencia total de reconocimiento de la interrupción es completada. En el modo máximo, la CPU activa la salida LOCK desde el estado T2 del primer ciclo de bus hasta el estado T2 del segundo ciclo de bus, para decirle a todos los árbitros de bus 8289 del sistema que el bus no puede ser accedido por otro procesador.

Luego efectúa un segundo INTA durante el cual lee sobre el bus el tipo de interrupción, y este tipo se multiplica por 4. El dispositivo externo indica, mediante un byte en el bus de datos, que tipo de interrupción desea, y este número lo usa el 8086 para localizar el puntero de la tabla de interrupciones. La CPU salva la palabra de estado en la pila y un puntero con la dirección de retorno, y utiliza el tipo como vector de indirección, reemplaza el CS y el IP actual por los encontrados en la dirección apuntada por el

tipo. Esto hace que el procesador ejecute la rutina de servicio.

Las interrupciones, bits IF y TF de la palabra de estado, son prohibidas. La rutina de interrupción así activada debe terminarse por una instrucción específica IRET que desempila la dirección de retorno y restaura la palabra de estado entera. La salvaguardia del resto de los registros se efectúa por el código de la rutina de interrupción.

IV-1-2.- Interrupción no enmascarable:NMI

Se refiere al hecho de que esta clase de interrupciones no pueden desactivarse aclarando el indicador (flag) de interrupciones, IF.

El procesador dispone de una patilla, NMI, que es una entrada de interrupciones sensible al flando ascendente de la señal NMI. La activación de esta entrada provoca una interrupción del tipo 2; es decir, su puntero se encuentra en la dirección 4 x n = 4 x 2 = 8 de la memoria. Las interrupciones no enmascarables se reservan para casos de emergencia como fallos de potencia o errores de memoria. Esta entrada conviene ponerla a la tensión de referencia en caso de no utilizarla.

IV-2) INTERRUPCIONES INTERNAS.

El 8086 dispone de dos clases de interrupciones internas. Aquellas generadas automáticamente por el procesador y las llamadas por Software, provocadas por instrucciones específicas.

IV-2-1.- Interrupciones automáticas:

- Tipo 0: Error en división. Esta interrupción es provocada al dividir por cero o por un cociente demasiado grande para ser contenido en el registro previsto.
- Tipo 1: Single-step (paso a paso). Se produce (cuando asi se especifica por el programador, bit TF=1) al termino de cada instrucción, con lo que puede utilizarse para depurar programas paso a paso.
- Tipo 4: Overflow (desbordamiento). Se produce cuando se detecta que se ha excedido la capacidad de un registro para albergar la información resultante de alguna operación.

IV-2-2.- Interrupciones por software:

- Tipo 3: Punto de parada (breakpoint). Provocada por la instrucción INT 3 que se codifica sobre un octeto, es decir, el código máquina tiene un sólo octeto en vez de los dos usuales, y sirve para escribir programas cuyo correcto funcionamiento queremos comprobar con los puntos de parada programados. Un monitor puede ser empleado para reemplazar el código de la operación de la instrucción sobre la que se quiere parar mediante INT 3. En la ejecución de la instrucción, el monitor se activará, repondrá el código de operación, y el usuario podrá entonces examinar el estado del programa y continuar su ejecución. Es decir, el programador selecciona una dirección donde desea detener el procesador y examinar el contenido de los registros e indicadores (flags).

El programador inserta el tamaño de octeto código máquina para esta interrupción en la direción seleccionada y ejecuta el programa. Cuando el procesador llega a este punto, salta a la rutina de servicio de tipo 3. Esta rutina debe devolver el control al programador para que

haga lo que quiera, quizás mostrar el contenido de los registros, quizás un volcado de memoria u otra cosa.

- Tipo n: Son las interrupciones provocadas por las instrucciones INT n, donde n vale entre 0 y 255.

IV-3) TABLA DE PUNTEROS DE INTERRUPCION.

La tabla de punteros de interrupción es la unión entre un tipo de código de interrupción y el procedimiento que ha sido diseñado para el servicio de las interrupciones asociadas con ese código.

Esta tabla ocupa el primer K byte de la parte baja de la memoria. Hay 256 entradas en la tabla, una para cada tipo de interrupción que pueden ocurrir en el sistema. Cada entrada en la tabla es un puntero de doble palabra, conteniendo la dirección del procedimiento que sirve a la interrupción de este tipo. La dirección alta de la palabra del puntero contiene la dirección base del segmento conteniendo el procedimiento. La dirección baja de la palabra contiene el offset del procedimiento desde el comienzo del segmento. Cada entrada tiene

4 byte de tamaño, la CPU puede calcular la posición de la entrada correcta para dar un tipo de interrupción por una simple multiplicación (tipo x 4).

Si múltiples interrupciones requeridas llegan simultáneamente, el procesador activa el procedimiento de interrupción en orden prioritario. En la siguiente tabla vemos el orden de prioridades de las interrupciones:

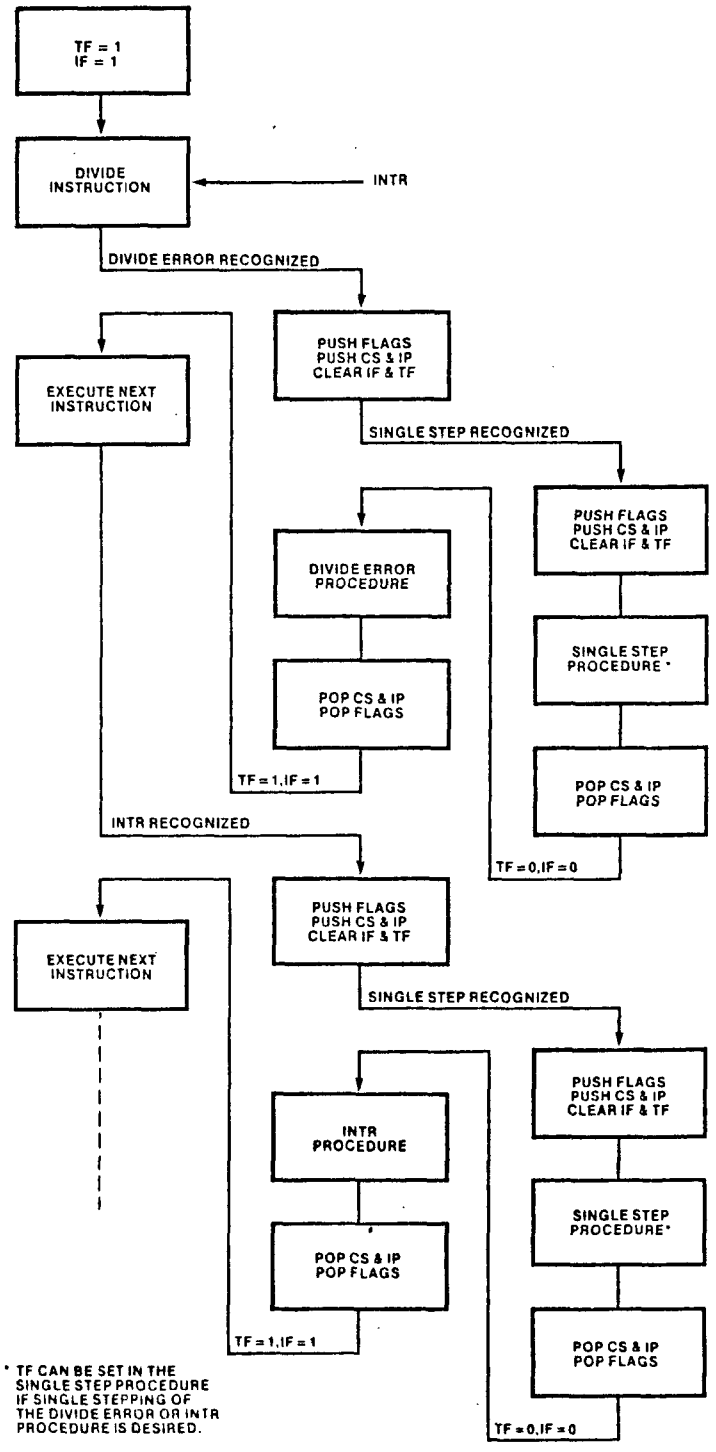
INTERRUPCION	PRIORIDAD
Error en división, INT n, INTO	Mayor
NMI	
INTR	
Single-step	Menor

El tiempo de procesamiento para las distintas clases de interrupción se muestran en la siguiente tabla:

TIPO DE INTERRUPCION	TIEMPO DE PROCESAMIENTO
INTR	61 clocks
NMI	50 clocks
INT (con vector)	51 clocks
INT TIPO 3	52 clocks
INTO	53 clocks
SINGLE-STEP	50 clocks .

Nótese que el tiempo mostrado en la tabla anterior, sólo representa el tiempo requerido para procesar el requerimiento de la interrupción después de que ésta ha sido reconocida. Para determinar la latencia (el intervalo de tiempo entre el anuncio del requerimiento de la interrupción y la ejecución de las instrucciones dentro de la rutina de interrupción), debe ser incluido un tiempo adicional para completar la instrucción que estaba siendo ejecutada cuando se anunció la interrupción, para salvar el contenido de cualquier registro adicional anterior al procesamiento de la interrupción (las interrupciones sólo salvan automáticamente los registros CS, IP y flags) y para cualquier estado de espera que pueda ser incluido durante el procesamiento de la interrupción.

La siguiente figura muestra el procedimiento de activación en un caso extremo. El procesador está corriendo en modo single-step con las interrupciones externas permitidas. Durante la ejecución de una instrucción de división, INTR es activada. Además la instrucción genera una interrupción de error en división.



La figura muestra el turno de reconocimiento de las interrupciones, en orden a su prioridad excepto para INTR. INTR no es reconocida hasta después del seguimiento de la instrucción porque un reconocimiento temprano de las interrupciones aclara IF. Por supuesto las interrupciones pueden ser repermitidas en cualquiera de las rutinas de respuesta de interrupción si una respuesta temprana a INTR se desea.

En la figura anterior, todos los códigos de linea principal son ejecutados en modo single-step.

También, porque del orden de procesamiento de la interrupción, existe la oportunidad en cada ocurrencia de la rutina de single-step de elegir si las rutinas pendiente de interrupción (rutinas error en división e INTR en este ejemplo) son ejecutadas a la velocidad tope o en modo single-step.

IV-4) PROCEDIMIENTO DE INTERRUPCION.

Cuando un procedimiento de servicio de interrupción es entrado, los flags, CS e IP son metidos en la pila (stack) y TF e IF son borrados. El procedimiento puede repermitir las interrupciones externas con la instrucción STI (poner a uno el flags de permiso de interrupción), de este modo permite así mismo ser interrumpido por un requerimiento INTR. (Nota, sin embargo, esas interrupciones no son actualmente permitidas hasta que la siguiente instrucción STI ha sido ejecutada). Un procedimiento de interrupción siempre puede ser interrumpido por un requerimiento llegado de NMI. El software o el proceso de iniciación de ocurrencia de interrupciones dentro del procedimiento también interrumpirá el procedimiento.

Como todos los procedimientos, los procedimientos de interrupción deben salvar cualquier registro que ellos usen antes de ser actualizados y restaurados antes de terminar.

Es una buena práctica para el procedimiento de interrupción permitir interrupciones externas para

todo menos para la sección crítica del código (esas secciones no pueden ser interrumpidas sin el riesgo de resultados erróneos). Si las interrupciones externas son prohibidas por largo tiempo en un procedimiento, los requerimientos de interrupción sobre INTR pueden ser potencialmente perdidas.

Todos los procedimientos de interrupción deben ser terminados con una instrucción IRET. La instrucción IRET asume que la pila está en la misma condición que estaba cuando el procedimiento fue entrado. Esto lleva la cima de las 3 palabras stack al IP, CS y los flags, de este modo retornamos a la instrucción que fue ejecutada cuando el procedimiento de interrupción fué activado.

El actual proceso hecho para el procedimiento está sirviendo a un dispositivo externo, éste deberá sacar un comando para el dispositivo instruyéndole a quitar la interrupción requerida. Este puede entonces leer la información de estado desde el dispositivo, determinar la causa de la interrupción y entonces tomar la acción oportuna.

IV-5) INTERRUPCION SINGLE-STEP (TRAP).

Cuando TF está puesto a uno, el 8086 se dice que está en el modo single-step. En este modo el procesador genera una interrupción tipo 1 despues de cada instrucción. Señal de llamada que como parte de su proceso de interrupción, la CPU automáticamente empuja los flags hacia la pila y barre el TF y el IF. De este modo el procesador no está en el modo single-step cuando el procedimiento de interrupción single-step está entrado; este corre normalmente. Cuando el procedimiento single-step termina, la imagen vieja del flag es restaurada desde la pila, colocando de nuevo la CPU en modo single-step. Single-step es una valiosa herramienta de depuración. Esto permite al procedimiento single-step actuar como una ventana dentro del sistema a través del cual las operaciones pueden ser observadas instrucción a instrucción. Con este camino el flujo exacto de un programa puede ser trazado en detalle, y el punto en el cual ocurren discrepancias puede ser determinado.

Otros posibles servicios que puede proporcionar la rutina single-step son:

- Escribir un mensaje cuando una específica posición de memoria o un puerto de E/S cambia de valor.
- Proporcionar diagnósticos selectivos.
- Ejecutar una rutina un número de veces antes de proporcionar el diagnóstico.

El 8086 no tiene una instrucción para poner a uno o aclarar el TF directamente. Mejor, TF puede ser cambiado modificando la imagen de los flags sobre la pila.

Las instrucciones PUSHF y POPF están disponibles para meter y sacar los flags directamente (TF puede ser puesto a uno por una OR de la imágen de los flags con 0100H y aclarado por una AND con FEFFH). Después de poner a uno TF de esta manera, la primera interrupción single-step ocurre despues de la primera instrucción siguiendo al IRET desde el procedimiento single-step.

Si el procesador está en single-step, este procesa una interrupción (interna o externa) como siguie. El control pasa normalmente (flags, CS e IP se guardan) al procedimiento designado para tratar el tipo de interrupción que ha ocurrido. Sin

embargo, antes que la primera instrucción de ese procedimiento se ejecute, la interrupción single-step es reconocida y el control se pasa normalmente al procedimiento de interrupción tipo 1. Cuando el procedimiento single-step termina, el control vuelve al procedimiento de interrupción previo.

IV-6) INTERRUPCION BREAKPOINT.

La interrupción Breakpoint produce una interrupción tipo 3. Un breakpoint está generalmente en cualquier lugar en un programa donde una ejecución normal se detiene para que alguna clase especial de proceso pueda ser ejecutado. Típicamente los breakpoints son insertados en programas durante depuraciones como un camino de visualización de registros, posiciones de memoria, etc..., en puntos cruciales en el programa.

La instrucción INT 3 (breakpoint) tiene un byte de longitud. Esto hace fácil el establecer un breakpoint en cualquier sitio en un programa.

La instrucción breakpoint también puede usarse para insertar nuevas instrucciones en un programa sin recopilar o reensamblar de nuevo. Esto puede

hacerse reservando un byte de instrucción, y reemplazarlo con una instrucción máquina INT 3 (CCH). El breakpoint puede contener la nueva instrucción byte y decrementar IP sobre la pila antes de retornar, para que la instrucción desplazada pueda ser ejecutada después de insertar las nuevas instrucciones'

IV-7) SYSTEM RESET.

La línea RESET del 8086 proporciona un camino para comenzar o reiniciar la ejecución de un sistema. Cuando el procesador detecta el flanco positivo de un pulso en RESET, éste termina toda actividad hasta que la señal baja, en cuyo momento inicializa el sistema como se muestra en la siguiente tabla:

COMPONENTE DE LA CPU	CONTENIDO
Flags	Aclarados
IP	0000H
Registro CS	FFFFH
Registro DS	0000H
Registro SS	0000H
Registro ES	0000H
Cola (Queue)	Vacia .

Después de que el registro segmento código (CS) contiene FFFFH y el puntero de instrucción (IP) contiene ØH, el procesador ejecuta su primera instrucción después de system reset (reset del sistema) desde la localización absoluta de memoria FFFFØH. Esta localización normalmente contiene una instrucción JMP intersegmentaeda directa cuyo objetivo es el comienzo actual del programa del sistema. Como las interrupciones externas (mascarables) son prohibidas por el system reset, el sistema software debe repermitir interrupciones tan pronto como el sistema sea inicializado en el punto donde ellas puedan ser procesadas.

IV-8) ESTADO DE LA INSTRUCCION QUEUE (COLA).

Cuando está en modo máximo, el 8086 proporciona información acerca de las operaciones de la instrucción queue en las líneas QS1 y QS2. En la siguiente tabla se interpretan los cuatro estados de estas líneas:

QS0	QS1	OPERACION QUEUE EN EL ULTIMO CICLO DE CLOCK.
0	0	No operación; falta valor.
0	1	Primer byte de una instrucción fue fue cogido desde la queue (cola).
1	0	La cola fue reinicializada.
1	1	Subsiguiente byte de un instrucción fue cogido desde la cola.

Las líneas de estado de la cola son proporcionadas para procesadores externos que reciben instrucciones y/o operandos vía la instrucción ESC del 8086.

Tal procesador debe serr el monitor del bus para ver cuando una instrucción ESC es buscada y entonces rastrear la instrucción a través de la cola para determinar cuando (y si) la instrucción es ejecutada.

IV-9) PARADA (HALT) DEL PROCESADOR.

Cuando la instrucción HLT (HALT) es ejecutada, el 8086 entra en un estado de parada. Esta condición puede ser interpretada como una parada total de las operaciones hasta que ocurra una interrupción

externa o el sistema sea reseteado (reset). Las señales no están circulando durante el estado de Halt, y el contenido de los buses de dirección y datos están indefinidos.

Un bus hold requerido llega sobre la línea HOLD (es una petición de detención momentánea del proceso en la configuración de modo mínimo) o una u otra línea Rquest/Grant (en la configuración de modo máximo) es reconocida normalmente mientras el procesador está parado.

El estado halt puede ser usado cuando un evento impide al sistema funcionar correctamente. Un ejemplo puede ser una interrupción por fallo de potencia. Después de reconocer que la pérdida de potencia es inminente, la CPU puede usar el tiempo restante para llevar los registros, flags y variables vitales (por ejemplo) a un área de batería de potencia RAM CMOS y entonces para, hasta que la señal de potencia vuelva por una interrupción o system reset.

IV-10) LINEAS DE ESTADO.

Cuando está configurado en modo máximo, el 8086 emite 8 señales de estado que pueden ser usadas por dispositivos externos. Las líneas S0, S1 y S2 identifican el tipo de ciclo de bus que la CPU está empezando a ejecutar. En la siguiente tabla vemos los distintos ciclos de bus:

S2	S1	S0	TIPOS DE CICLOS DE BUS
0	0	0	Reconocimiento de interrupción.
0	0	1	Lectura E/S.
0	1	0	Escritura E/S.
0	1	1	Parada.
1	0	0	Búsqueda de la instrucción.
1	0	1	Lectura memoria.
1	1	0	Escritura memoria.
1	1	1	Inactivo; no ciclo de bus.

Estas líneas son típicamente decodificadas por el controlador de bus 8288. S3 y S4 indican que registro segmento fué usado para construir la dirección física que ha sido usada en este ciclo de bus.

La siguiente tabla nos muestra los diferentes registros segmento utilizados:

S4	S3	REGISTROS SEGMENTO.
0	0	ES
0	1	SS
1	0	CS o ninguno (E/S o vector interrupción).
1	1	DS

La línea S5 refleja el estado del flag de permiso de interrupción. S6 está siempre a cero. S7 es una línea de repuesto cuyo contenido está indefinido.

V) HARDWARE DE LA CPU 8086.

V-1) SEÑALES DEL 8086 .

Las señales del 8086 se pueden agrupar de la siguiente manera:

- Alimentación.
- Reloj.
- Control y estado.
- Direcciones.
- Datos.

Hay tres terminales para la alimentación: tierra (GND) en los terminales 1 y 20, y una tensión de entrada de 5 voltios (Vcc) en el terminal 40.

Hay una señal de reloj en el terminal 19.

El 8086 puede configurar sistemas mínimos, que no admiten la multitarea, y sistemas máximos, capaces de soportar un bus local, para ampliar directamente el 8086, y un bus del sistema (Multibus), que permite configuraciones con varios procesadores.

La patilla denominada MN/MX selecciona el funcionamiento del 8086 en modo mínimo o máximo.

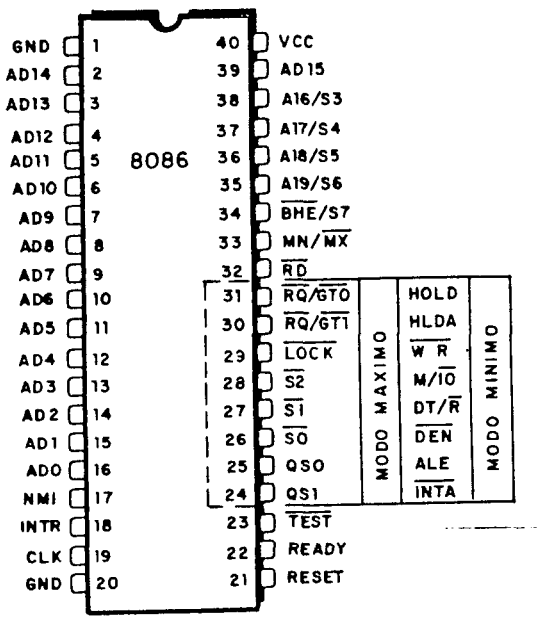
En el modo máximo, el 8086 no genera directamente las señales del bus del sistema, sino que saca

tres líneas, S0, S1 y S2, que son decodificadas por el circuito 8288, de cuya salida se obtienen las señales compatibles con el Multibus.

Los requerimientos de control, tanto en modo mínimo comok máximo, precisan más de 40 patillas; de ahí, la necesidad deï multiplexado de datos y direcciones controlado por la señal ALE, asi como el cambio de función que se produce en otras líneas según se opere en modo mínimo o máximo.

En el 8086 hay 20 bits de dirección. Los cuatro bits más significativos de la dirección comparten terminales con algunas de las señales de estado. Los 16 bits menos significativos se comparten con los bits de datos. En ciertos instantes, tales terminales conducen parte de una dirección, y en otros llevan información sobre el estado y los datos.

En la siguiente figura se muestra el diagrama de conexionado del 8086:



Descripción de las patillas:

READY:

Se emplea para sincronizar el procesador con la memoria o los dispositivos de E/S, que son más lentos. Si está a nivel alto, indica a la CPU que la memoria o los periféricos están dispuestos para enviar o recibir datos. Si está a nivel bajo, la CPU entrará en un estado de espera (WAIT) hasta que la señal se reciba.

RESET:

Se emplea para reinicializar el procesador. La señal RESET borra la cola de instrucciones, el registro de estado se pone a cero, así como DS, SS, ES e IP. El registro CS se pone a FFFFH. Al poner al registro CS a FFFFH, fuerza al procesador a leer el primer byte de instrucción en la dirección FFFFØH.

TEST:

Se utiliza para enlazar el 8086 con un procesador paralelo, tal como el procesador numérico 8087, sincronizando el procesador principal con los otros. Es una entrada utilizada, sólo, por la instrucción WAIT.

Cuando la CPU ejecuta esta instrucción, queda en un estado de espera del que no sale hasta la activación de TEST.

RD:

Indica un ciclo de lectura de memoria o entradas y salidas (a nivel bajo).

BHE/S7:

BHE sirve para la selección de bytes o palabras en el primer ciclo de la instrucción. S7 no tiene un significado definido y actúa en los siguientes ciclos de la instrucción.

AD0-AD15:

Son las 16 líneas que llevan, multiplexadamente, las direcciones y los datos.

A16/S3-A19/S6:

Durante el estado Ti de un ciclo, contienen los bits de mayor peso de la dirección. Durante los estados T2, T3 y T4, contienen información sobre:

- Tipo de segemento utilizado (S4 y S3).
- Máscara de interrupciones (S5).
- Registro de reubicación (S6).

INTR:

Petición de interrupción mascarable.

NMI:

Petición de interrupción no mascarable.

Vcc:

Alimentación de +5v.

MN/MX:

Para poner al proesador en modo mínimo o en modo máximo. MN, configuración en modo mínimo, se pone el terminal a la tensión de alimentación (+5v). MX, configuración en modo máximo, se pone el terminal a tierra (GND).

CLK:Entrada de la señal de reloj, procedente del generador 8284.

GND:

Tierra.

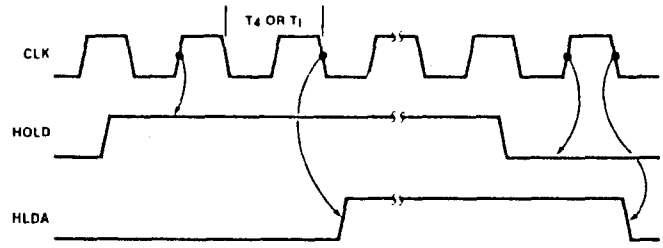
V-1-1.- Señales en modo mínimo (MN).

En el modo mínimo (MN conectado a +5v), la CPU forma un sistema con un sólo procesador que consta de unos pocos dispositivos y que usa el bus del sistema mejor que los soportes de la arquitectura del Multibus. En este modo, la CPU genera todas las señales de control del bus (DT/R, DEN, ALE y una u otra M/IO o IO/M) y la señal de salida de comando (RD, WR o INTA), y proporciona una mecanismo para el requerimiento de acceso al bus (HOLD/HLDA) que es compatible con los controladores del bus maestro (ej. los controladores DMA de Intel 8237 y 8257).

En el modo mínimo, cuando un bus maestro requiere acceso al bus, éste activa la entrada HOLD en la CPU.

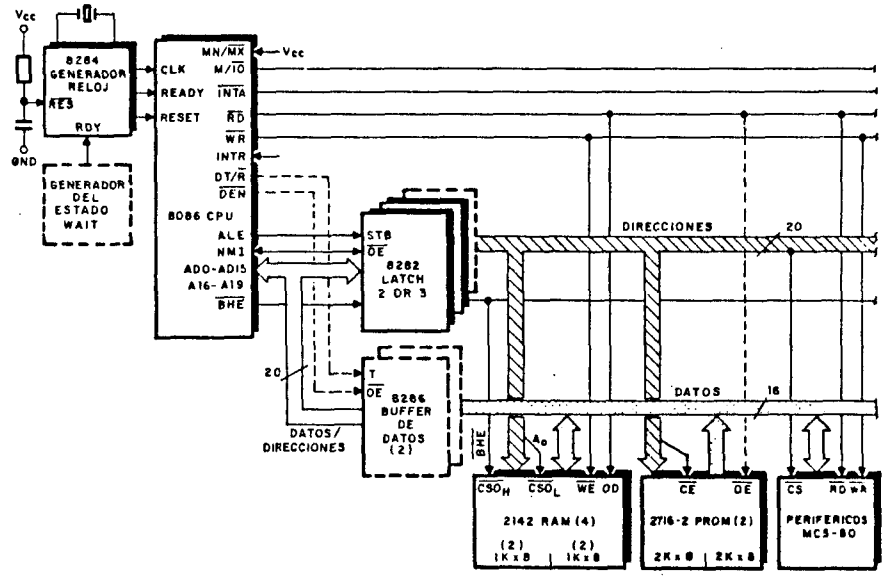
La CPU en respuesta al requerimiento HOLD, activa HLDA como un reconocimiento a el bus maestro del requerimiento de bus y simultáneamente el bus del sistema y las líneas de control quedan liberadas.

La CPU saca una muestra de la entrada HOLD en la transición positiva de cada señal de reloj y, como se muestra en la siguiente figura, activa HLDA al final del actual ciclo de bus (si un ciclo de bus está en marcha) o en un periodo de reloj desocupado.



El estado HOLD se mantiene hasta que el bus maestro inactiva la entrada HOLD en cuyo momento la CPU recupera el control del bus del sistema. Note que durante el estado HOLD, la CPU continuará ejecutando instrucciones hasta que es requerido un ciclo de bus.

En la siguiente figura vemos la configuración mínima del 8086:



Las señales en modo mínimo son:

HOLD:

Petición del bus por un periférico exterior.

Cuando otro procesador o un aparato, como un controlador DMA, quiere acceder al control del bus, manda una señal al 8086 a través de la línea HOLD. Cuando está preparado para hacerlo, el 8086 pone sus líneas de datos/direcciones y la mayoría de líneas de control en el estado de alta impedancia. El procesador toma de nuevo el control de los buses sólo cuando la señal HOLD desaparece.

HLDA:

Aparece como respuesta a una petición HOLD. Indica que el bus ha quedado libre. El otro aparato puede usar ahora el bus. Se mantendrá en un nivel alto de tensión hasta que la señal HOLD desaparezca.

WR:

Control de escritura.

M/IO:

Indica la realización de una operación sobre memoria o sobre E/S.

DT/R:

Indica el sentido del movimiento de la información (transmisión o recepción).

DEN:

Datos accesibles (data enable). Para controlar la transferencia de datos, el 8086 precisa de la colaboración del circuito auxiliar 8286 (8287). Este se gobierna por la señal DT/R, que indica la transmisión/recepción de datos, y por DEN, que confirma la validación de los datos.

ALE:

(Address Latch Enable). La señal ALE le dice al Latch de direcciones octal 8282 donde encontrar la dirección, es decir, la señal ALE activa el latch 8282 cuando viene una dirección por las líneas AD0-AD15.

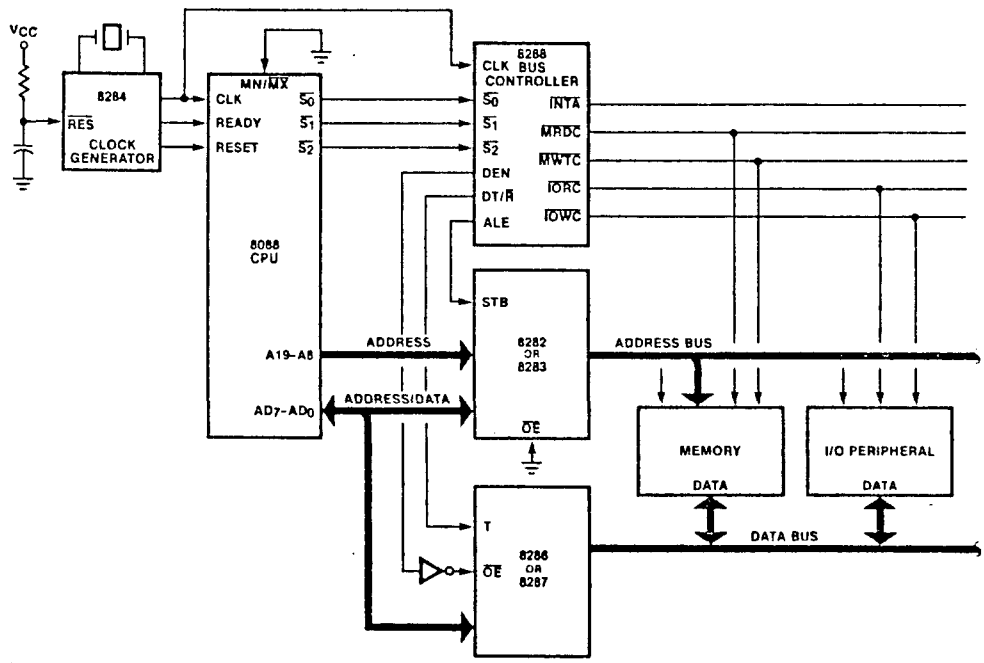
INTA:

Reconocimiento de interrupción. Se envía esta señal en respuesta a una petición de interrupción INTR cuando ésta ha sido aceptada.

V-1-2.- Señales en modo máximo (MX).

En el modo máximo (MX conectado a tierra), el controlador de bus 8288 es añadido para proporcionar un sofisticado control de bus y compatible con la arquitectura del Multibus (combinando el 8289, árbitro del bus del sistema, con el 8288 permiten a la CPU soportar múltiples procesadores sobre el bus del sistema). Como se muestra en la siguiente figura, el controlador del bus, mejor que la CPU, proporciona todo el control del bus y comandos de salida.

La siguiente figura nos muestra una configuración en modo máximo del 8086:



Las señales en modo máximo son:

S0, S1 y S2:

Decodifican el estado del procesador, de acuerdo con la tabla vista en el apartado IV-10.

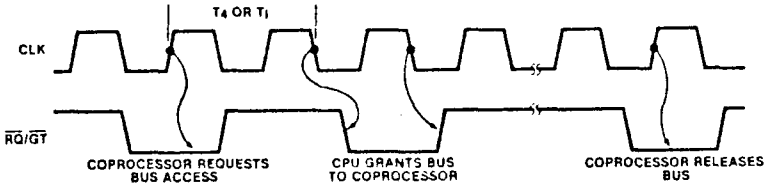
RQ/GT1, RG/GT0:

Las líneas de señal Request/Grant (petición/concesión) proporcionan el mecanismo de acceso al bus de la CPU en el modo máximo (operan de la misma forma que lo hacen HOLD y HLDA para transferir el control de bus en el modo mínimo), y están expresamente diseñadas para aplicaciones con multiprocesador usando el procesador de E/S 8089 en este modo u otros procesadores que pueden soportar esta función.

Como muestra la siguiente figura, la secuencia request/grant está en tres fases del ciclo: request (petición), grant (concesión) y liberar. La secuencia es iniciada por otro procesador, sobre el bus del sistema, cuando sale un pulso sobre una de las líneas RQ/GT para requerir acceso al bus (fase de petición). En respuesta, la CPU saca un pulso (sobre la misma línea) al final de uno u otro ciclo de bus actual (si un ciclo de bus está en proceso) o periodo de reloj desocupado,

para indicar al procesador que éste tiene el bus del sistema flotando y que se desconectará lógicamente, desde el bus controlador, en el próximo ciclo de reloj (fase de concesión) y entrará en un estado de HOLD. Note que la E.U. de la CPU continua ejecutando las instrucciones en la cola hasta que una instrucción de requerimiento de acceso al bus es encontrada o hasta que la cola se vacie. En la tercera fase (liberar), el requerimiento del procesador emite de nuevo un pulso en la línea RQ/GT.

Este pulso le dice a la CPU que el procesador está listo para liberar el bus. La CPU recupera el acceso al bus en el próximo ciclo de reloj. Note que el cambio de pulsos es sincronizado y, acordadamente, ambos, la CPU y el procesador requerido, deben ser referidos a la misma señal de reloj.



Las líneas request/grant (petición/concesión) son prioritarias, siendo RQ/GT0 prioritaria sobre RQ/GT1.

Si una petición llega a ambas líneas simultáneamente, al procesador en RQ/GT0 se le concede el bus (el requerimiento sobre RQ/GT1 es concedido cuando el bus es liberado por el primer procesador siguiendo a uno o dos retardos del reloj del canal transferido). Ambas líneas RQ/GT (y la línea HOLD en el modo mínimo) tienen más alta prioridad que una interrupción pendiente.

La latencia de request/grant (el intervalo de tiempo entre la recepción de un pulso de petición y el retorno de un pulso de concesión) para varias condiciones viene dado en la siguiente tabla:

CONDICION DE OPERACION .	RETARDO REQUEST/GRANT
Procesamiento normal de la instruc,LOCK inactivo.	3-6 (10) clocks
Ejecutando ciclo INTA, LOCK activo.	15 clocks
Procesando instruc.locked XCHG, LOCK activo.	24-31 (39) clocks

Los números de clocks entre paréntesis, se aplican cuando la instrucción es ejecutada referida a palabras y a una dirección impar.

El número de clocks contenidos en el paréntesis cuando la instrucción ha sido ejecutada se refiere a un operando palabra en el límite de una dirección impar.

La latencia durante un procesamiento de una instrucción normal (LOCK inactivo) puede ser tan corto como tres ciclos de reloj (por ejemplo, durante la ejecución de una instrucción que no se refiera a la memoria) y no más de diez ciclos de reloj. Siempre que la salida de LOCK es activa (LOCK es activado durante un ciclo de interrupción reconocida o durante la ejecución de una instrucción con un prefijo lock), la latencia es incrementada. En el caso de la ejecución de una instrucción cerrada XCHG (usada durante la examinación de señales), el máximo de latencia es limitado a 39 ciclos de reloj. Las grandes latencias ocurren cuando una gran instrucción es cerrada.

Al final de la actividad del procesador, el 8086 no recupera su control y buses de datos, hasta los dos ciclos de reloj siguientes a la recepción del

pulso de liberación (o los dos ciclos de reloj despuésde ponerse HOLD inactivo en el modo mínimo). Una petición de Hold es inmediatamente aceptada, siguiendo a un reset de la CPU, si la línea HOLD es activada cuando la línea RESET está inactiva. Esta acción facilita el downloading de programas y, más específicamente, la puesta de la localización de memoria FFFFØH previa a la activación de la CPU. Note que el mismo resultado puede ser efectuado en el modo máximo, a través de la línea RQ/GT, por generación del pulso de petición en el primer o segundo ciclo de reloj despues de RESET inactivo.

LOCK:

La salida LOCK es usada en conjunción con el Intel 8298 árbitro de bus, para garantizar el acceso exclusivo, en un sistema de bus compartido, durante la duración de una instrucción. Esta salida es controlada por software.

Cuando el prefijo Lock es decodificado por el E.U., la E.U. informa a la B.I.U. para activar la salida LOCK durante el próximo ciclo de reloj. Esta señal permanece activa hasta un ciclo de reloj después de que la ejecución de la instrucción asociada es concluída.

QS1, QS0:

Las salidas QS1 y QS0 (estado de la cola) permiten una comprobación externa de la cola de instrucción interna de la CPU.

La codificación de los bits QS1 y QS0 se muestra en la siguiente tabla:

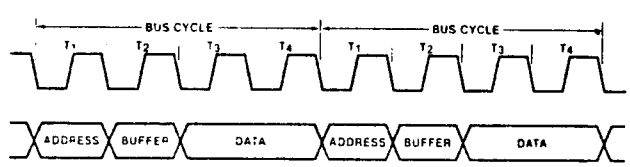
QS1	QS0	ESTADO DE LA COLA
0	0	No operación durante el último ciclo bajo de reloj, no se tomó nada de la cola.
0	1	Primer byte. El byte tomado de la cola era el primer byte de la instrucción.
1	0	Cola vacia. La cola ha sido alto reinicializada como resultado de una instrucción de transferencia.
1	1	Subsiguiente byte. El byte tomado desde la cola era el byte subsiguiente de la instrucción.

El estado de la cola se valida en el ciclo de reloj siguiente al de la actividad indicada.

V-2) OPERACION DEL BUS.

Para explicar la operación del bus multiplexado en el tiempo, debe ser examinado el ciclo del bus. Esencialmente, un ciclo de bus es un suceso asincrono en el cual la dirección de un periférico de E/S o de una posición de memoria es presentada, seguida por una u otra señal de control de lectura (capturar o leer el dato desde el dispositivo seleccionado) o una señal de control de escritura y el dato asociado (transmitir o escribir el dato al dispositivo direccionado). El dispositivo seleccionado (memoria o periférico E/S) acepta el dato sobre el bus durante un ciclo de escritura o posiciona el dato requerido sobre el bus durante un ciclo de lectura. A la terminación del ciclo, el dispositivo latchea el dato escrito o quita el dato leído.

Como muestra la siguiente figura, todos los ciclos de un bus constan de un mínimo de 4 ciclos de reloj o estados T identificados como T1, T2, T3 y T4.

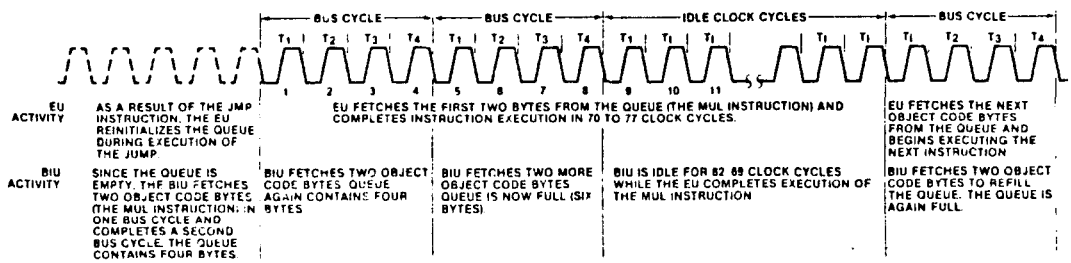


La CPU posiciona la dirección de la localización de memoria o dispositivo de E/S sobre el bus durante el estado T1. Durante el ciclo de escritura en bus, la CPU posiciona el dato sobre el bus desde el estado T2 hasta el estado T4. Durante un ciclo de lectura en bus, la CPU acepta los datos presentes sobre el bus en los estados T3 y T4, y el bus multiplexado dirección/dato está flotando en el estado T2 para permitir a la CPU desde el modo de escritura (salida dirección) cambiar a modo lectura (entrada dato).

Es importante notar, que la B.I.U. ejecuta sólo un ciclo de bus cuando un ciclo de bus es requerido por la E.U. como parte de la ejecución de una instrucción o cuando debe rellenar la cola de instrucción. Consecuentemente, los periodos de reloj en los cuales no hay actividad de la B.I.U. pueden ocurrir entre ciclos de bus.

Estos periodos de reloj inactivos son referidos como estados desocupados (TI). Mientras que los estados desocupados de reloj resultan de varias condiciones (ej. acceso al bus concedido a un coprocesador), como un ejemplo, considerar el caso de la ejecución de una gran instrucción. En el siguiente ejemplo, una instrucción de

multiplicación (MUL) de un registro de 8 bits (la cual requiere entre 70 y 77 ciclos de reloj) es ejecutada por el 8086. Asumiendo que la rutina de multiplicación es entrada como un resultado de un salto de programa (el cual causa que la cola de instrucción sea reinicializada cuando el salto sea ejecutado) y que los bytes del código objeto son alineados sobre los límites del byte-par, la secuencia de los ciclos de bus de la B.I.U. pueden aparecer como se muestra en la siguiente figura:



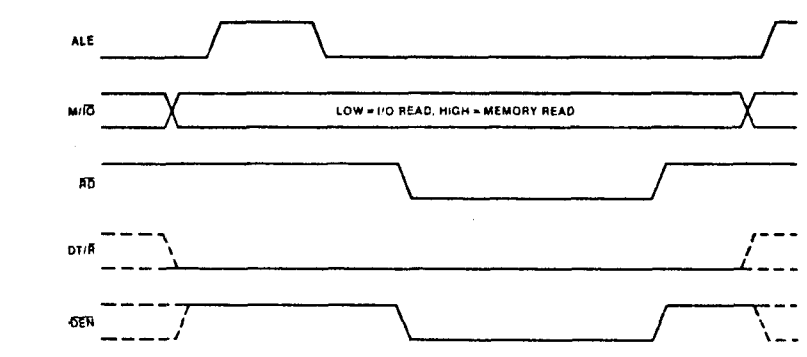
Además, en el estado desocupado descrito anteriormente, la CPU 8086 incluye un mecanismo para insertar estados-T adicionales en el ciclo de bus, para ajustar a los dispositivos (memoria o E/S) que no pueden transferir datos a la máxima velocidad. Estos estados-T extras se llaman estados de espera (Tw), y, cuando son requeridos, son insertados entre los estados T3 y T4. Durante

un estado de espera, los datos sobre el bus permanecen inalterados. Cuando el dispositivo puede completar la transferencia (presenta o acepta el dato), señala a la CPU para salir del estado de espera y entrar en el estado T4.

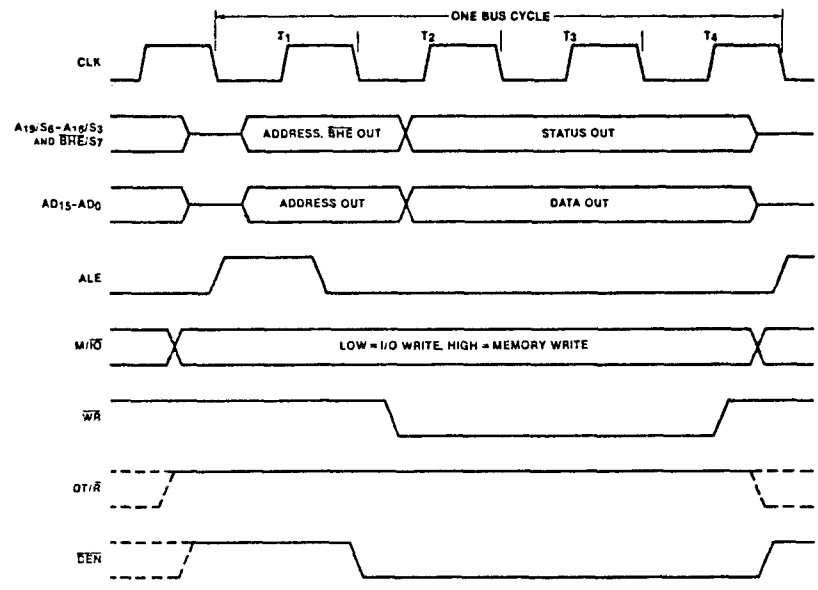
Como se muestra en el siguiente diagrama de tiempo, el actual ciclo de tiempo del bus varía según sea un ciclo de bus de lectura o de escritura.

A continuación se ilustran los diagramas de tiempo para el ciclo de bus de lectura y escritura en modo mínimo.

CICLO DE LECTURA DEL BUS



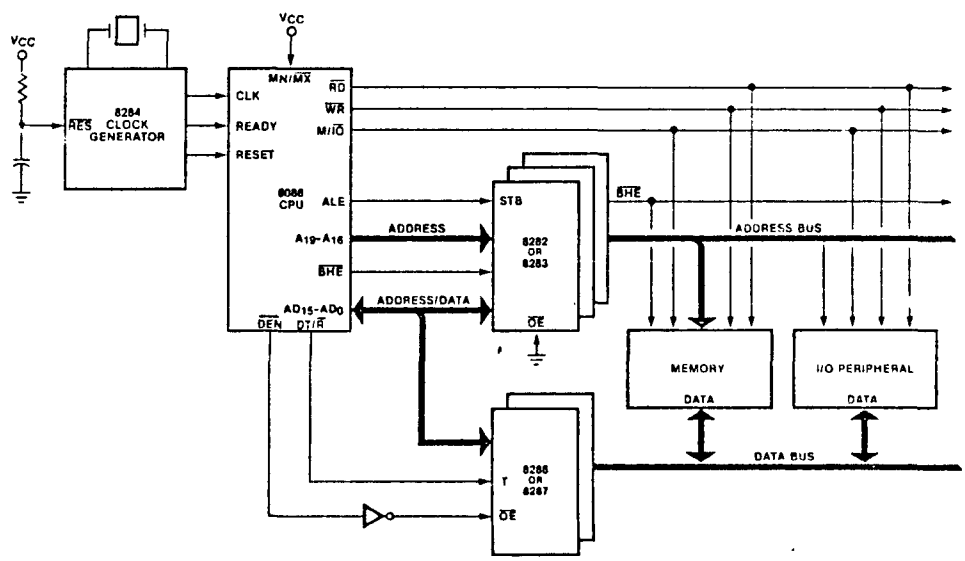
CICLO DE ESCRITURA DEL BUS



Refiriéndonos a los diagramas anteriores, la CPU 8086 posiciona una dirección de 20 bits sobre el bus multiplexado de dirección/dato durante el estado T1. Durante el estado T2, la CPU quita la dirección desde el bus y el tri-state de las 16 líneas bajas de dirección/dato, preparándolas para el ciclo de lectura (ciclo de lectura del bus) o posicionando el dato a escribir sobre estas líneas (ciclo de escritura del bus). Al mismo tiempo, el estado del ciclo del bus está disponible sobre las

líneas de dirección/estado. Durante el estado T3, el estado del ciclo del bus se mantiene sobre las líneas de dirección/estado y la escritura del dato se mantiene, o la lectura del dato se muestra sobre las 16 líneas bajas de dirección/dato. El ciclo del bus se termina en el estado T4 (las líneas de control son prohibidas y las direcciones del dispositivo son deseleccionadas desde el bus). El bus de datos no puede ser demultiplexado debido a los diferentes tiempos entre los ciclos de lectura y escritura, y los múltiples tiempos de respuesta de lectura entre los periféricos y memorias. Consecuentemente, el multiplexado del bus de datos puede usarse directamente o con un buffer. Cuando la memoria y periféricos de E/S son conectados directamente a un bus no buffer, es esencial que durante un ciclo de lectura, un dispositivo sea acoplado para prevenir la perdida de la dirección presentada sobre el bus durante el estado T1. Para Asegurar que la dirección no sea alterada, una función de permiso de salida permite la operación de los dispositivos de salida de los drivers (mejor que la función de selección del chip de los dispositivos) controlados por la señal de lectura de la CPU. (La familia de procesadores del MCS-86 garantiza que la señal de lectura no se

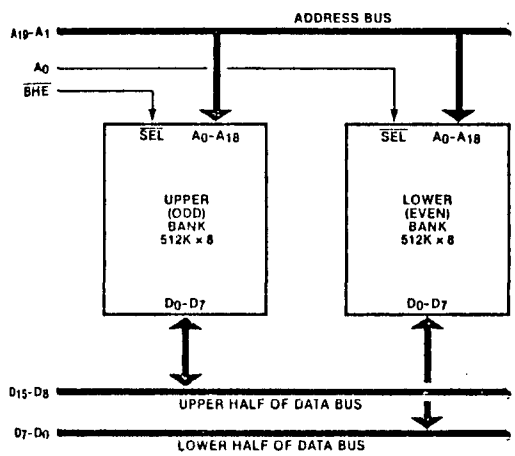
validará hasta después de que la dirección ha sido latcheada por la ALE). Muchos periféricos de Intel, ROM/EPROM y circuitos RAM proporcionan una función de permiso de salida para permitir el interface a un bus multiplexado de dirección/dato no buffer. La alternativa de usar un bus de datos buffer puede ser considerada puesto que esto simplificaría los requerimientos de interface y ofrecería aumentos de capacidad del actual drive e inmunidad de carga. El Intel 8286 (no inversor) y 8287 (inversor) Transceptores Octal de Bus, en modo mínimo, están expresamente diseñados como buffer del bus de datos. Estos transceptores usan las señales DEN y DT/R de control de la CPU para permitir y controlar la dirección del dato sobre el bus. Estas señales proporcionan la apropiada relación de tiempo para garantizar el aislamiento de la dirección que se presenta sobre el bus multiplexado durante el estado T1.



V-3) DIRECCIONAMIENTO DE LA MEMORIA EXTERNA.

La CPU 8086 tiene un bus de dirección de 20 bit y son suficientes para acceder al megabyte del espacio de dirección de memoria.

El espacio de dirección de memoria del 8086 consiste en una secuencia de un millón de bytes individuales, en el cual cualquier par de bytes consecutivos pueden ser accedidos como un dato palabra de 16 bit. Como muestra la siguiente figura, el espacio de dirección de memoria está físicamente dividido en dos bancos de 512 Kbytes cada uno:



Un banco está asociado con la mitad baja de los 16 bit del bus de datos de la CPU (bits de datos D7-D0), y el otro banco está asociado con la mitad alta del bus de datos (bits de datos D15-D8). Los

bits de dirección A19 a A1 son usados simultáneamente para direccionar un byte de una posición especifica en ambos bancos alto y bajo, y el bits de dirección A0 no es usado en el direccionamiento de memoria. A0 es usado para seleccionar el banco de memoria. El banco bajo, el cual contiene los bytes de dirección par, es seleccionado cuando A0=0. El banco alto, contiene los byte de dirección impar (A0=1), es seleccionado por una señal a parte, BHE (BUS HIGH ENABLE). La siguiente tabla define el mecanismo de selección del banco mediante BHE-A0 :

BHE	A0	BYTE TRANSFERIDO
0 (bajo)	0	Ambos bytes.
0	1	Byte alto a/desde la dir. impar.
1 (alto)	0	Byte bajo a/desde la dir. par.
1	1	Ninguno.

BHE : Es activo a nivel bajo.

A0 : Es activo a nivel alto.

Cuando accedemos a un dato byte en una dirección par, el byte es transferido a o desde el banco bajo sobre los bits de menor peso del bus de datos (D7-D0). En este caso, el nivel inactivo del bit

de dirección A0 permite direccionar el byte en el banco bajo, y el nivel inactivo de la señal BHE prohibe direccionar el byte en el banco alto. Recíprocamente, cuando se ejecuta el acceso a un byte en una dirección impar, el dato byte es transferido a o desde el banco alto sobre los bits de mayor peso del bus de datos (D15-D8). El nivel activo de la señal BHE permite direccionar el byte en el banco alto, y el nivel activo del bit de dirección A0 prohibe direccionar el banco bajo.

Como indica la tabla anterior, el 8086 puede acceder a un byte en ambos bancos, alto y bajo, simultáneamente como una palabra de 16 bits. Cuando el byte de menor orden de la palabra es accedido sobre una dirección par (que es cuando el byte de menor orden está en el banco bajo), la palabra se dice que está alineada y puede ser accedida en una única operación (un único ciclo de bus). Como con el byte transferido, previamente descrito, los bits de dirección A19 a A1 direccionan ambos bancos, excepto que ahora BHE está activo (seleccinando el banco alto) y A0 está inactivo (seleccionando el banco bajo) para acceder ambos byte.

Cuando el byte de menor peso de la palabra es accedido sobre una dirección impar (cuando el byte de menor peso está en el banco alto), la palabra se dice que no está alineada y debe ser accedida en en dos ciclos de bus. Durante el primer ciclo, el byte de menor orden de la palabra es transferido a o desde el banco alto como se describió para acceder a un byte en una dirección impar (A0 y BHE activo). La dirección de memoria es entonces incrementada, lo cual causa que A0 se ponga a nivel inactivo (seleccionando el banco bajo), y el acceso al byte en una dirección par es ejecutado durante el próximo ciclo de bus, para transferir el byte de mayor peso de la palabra a o desde el banco bajo. La secuencia anterior es inicializada automáticamente por el 8086, siempre que un acceso a una palabra en una dirección impar es ejecutado. También, la dirección de los bytes alto y bajo del registro de palabra interno del 8086 del bus de datos, es ejecutada automáticamente y, excepto para los cuatro ciclos de reloj adicionales requeridos para ejecutar el segundo ciclo de bus , la operación es transparente para el programa.

TEMA II:

"MANUAL DEL USUARIO DEL SDK-86"

II) MANUAL DEL USUARIO DEL SDK-86

II-1) INTRODUCCION.

Este capítulo describe como comunicarse con el SDK-86 a través del programa Monitor de teclado.

El programa monitor reside en los 4 Kbytes de ROM situados en las posiciones de más peso de la memoria.

Este programa es inicializado o leído siempre que se encienda (se alimente) el SDK-86 o en cualquier momento que sea presionada la tecla SYSTEM RESET, y permite ejecutar las operaciones siguientes usando el teclado y display:

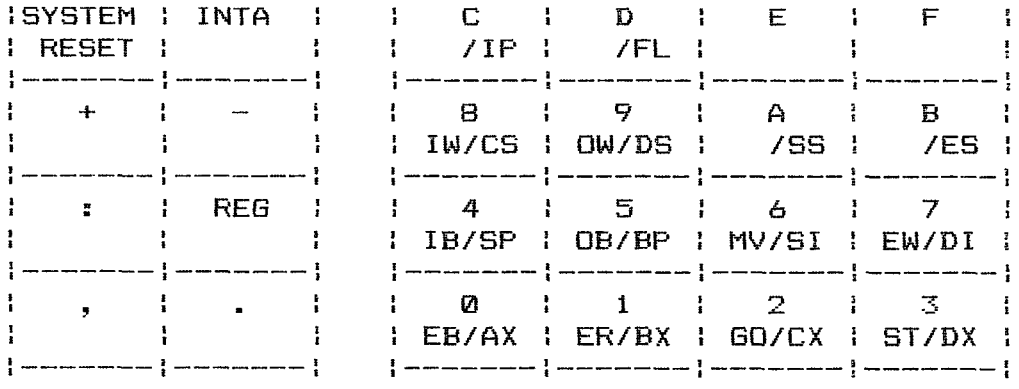
- Examinar y modificar registros dentro del 8086.
- Examinar y modificar lugares de memoria.
- Entrar e iniciar la ejecución de nuestros propios programas o subrutinas.
- Evaluar la ejecución (depuración) de tu programa a través del single-step del monitor y facilitar el punto de ruptura (Breakpoint).
- Seleccionar movimientos de bloques de memoria de un lugar a otro.
- Escribir o leer datos a/o desde un puerto de E/S.

II-2) TECLADO.

Con el programa monitor de teclado, entran los comandos y datos al presionar las teclas individuales del teclado (el monitor se comunica con nosotros a través del display).

Como se ve en la siguiente figura, el teclado está dividido en dos grupos lógicos; las 16 teclas hexadecimales y las 8 teclas de función:

TECLADO DEL SDK-86



TECLAS DE FUNCION

TECLAS HEXADECIMALES

Las teclas hexadecimales, además de representar el dígito hexadecimal marcado en primer lugar, pueden utilizarse también para referenciar registros internos de la CPU (a la derecha de la barra) o para asignar comandos del programa monitor (a la izquierda de la barra).

La siguiente tabla nos define los comandos y registros asociados con las teclas hexadecimales:

SIGNIFICADO DE LAS TECLAS HEXADECIMALES

TECLA	COMANDO		REGISTRO	
	SIGNO	NOMBRE	SIGNO	NOMBRE
0 EB/AX	EB	EXAMINAR BYTE	AX	ACUMULADOR
1 ER/BX	ER	EXAMINAR REGISTRO	BX	BASE
2 GO/CX	GO	GO	CX	CONTADOR
3 ST/DX	ST	SINGLE STEP	DX	DATO
4 IB/SP	IB	ENTRADA BYTE	SP	PUNTERO PILA
5 OB/BP	OB	SALIDA BYTE	BP	PUNTERO BASE

TECLA	COMANDO		REGISTRO	
	SIGNO	NOMBRE	SIGNO	NOMBRE
6 MV/SI	MV	TRANSFERIR	SI	INDICE FUENTE
7 EW/DI	EW	EXAMINAR PALABRA	DI	INDICE DESTINO
8 IW/CS	IW	ENTRADA PALABRA	CS	SEGMENTO CODIGO
9 OW/DS	OW	SALIDA PALABRA	DS	SEGMENTO DATO
A /SS			SS	SEGMENTO PILA
B /ES			ES	SEGMENTO EXTRA
C /IP			IP	PUNTERO INSTRUCCION
D /FL			FL	FLAG
E				
F				

Las operaciones de las 8 teclas de función son:

- SYSTEM RESET:

  Permite interrumpir cualquier actividad presente y retorna al SDK-86 al estado inicial. Al pulsarse esta tecla, el 8086 señala, mediante un mensaje en el display, que el monitor está listo para la aceptación de un comando.

- INTR:

  Genera una interrupción no enmascarable (NMI), del tipo 2. El vector NMI se inicializa cada vez que se conecta la alimentación o se pulsa la tecla RESET. Apunta a una subrutina del monitor, que ocasiona el almacenamiento de todos los registros de la CPU en el stack. El control vuelve al monitor por el subsiguiente comando de entrada.

-" + " :

  Permite sumar dos valores hexadecimales, simplificando el direccionamiento relativo, ya que autoriza a calcular una dirección relativa a partir de una dirección base.

-" - " :

Resta dos valores hexadecimales.

-" : " :

Se utiliza para separar una dirección en dos partes, para ser entradas: un valor de segmento y otro de offset o desplazamiento.

- REG:

La tecla REG (Registro) permite usar el contenido de cualquier registro del 8086 como si fuera una dirección o dato de entrada.

-" , " :

Separa entradas de teclado e incrementa el campo de dirección a la siguiente posición de memoria.

-" . " :

Comando de finalización. Cuando se pulsa, el comando en curso se ejecuta. Nótese que cuando se usa el comando "GO", el 8086 empieza a ejecutar el programa en la dirección especificada cuando la tecla es presionada.

II-3) DISPLAY.

El display, que comunica al sistema con el mundo exterior, consta de 8 dígitos, que, según trabaje el monitor, puede realizar las funciones siguientes:

- Visualizar el contenido de un registro o posición de memoria.
- Visualiza, repitiendo la tecla hexadecimal pulsada de entrada de información.
- Visualiza señales del monitor.
- Visualiza información o mensaje de estado.

El display está dividido en dos grupos de caracteres:

- Los cuatro dígitos de la izquierda visualizan el campo de la dirección.
- Los cuatro de la derecha, el campo de los datos.

Todos los valores visualizados están en hexadecimal.

II-4) COMANDOS DEL MONITOR DEL SDK-86

Cuando se hace uso del monitor de teclado, será por medio del display por donde se pida el tipo de entrada que se requiere.

Cuando el monitor espera que le entre un comando, aparece un guión en el dígito más significativo del display del campo de direcciones. Cuando el comando es presionado, desaparece el guión y aparece un punto decimal en el dígito menos significativo del campo de direcciones, para indicar que la subsiguiente entrada desde el teclado será dirigida al campo de direcciones.

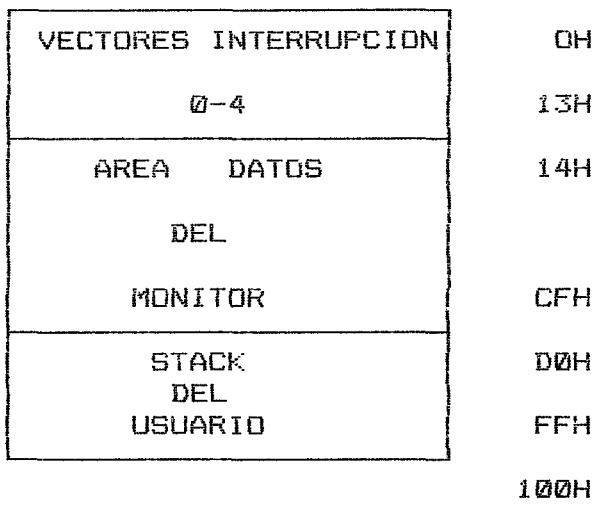
La operación del monitor desde este momento está determinada por el comando introducido.

En el próximo encendido o siempre que la tecla de Reset del sistema sea presionada, el monitor inicializa el SDK-86 y visualiza el mensaje de monitor encendido (se visualiza "86" en los dos dígitos menos significativos del campo de direcciones y el número de versión del programa en los dos dígitos menos significativos del campo de datos) y el comando se indica con el guión.

Cuando inicializamos, los registros del 8086 serán puestos a los valores que indica la siguiente tabla por el monitor:

REGISTRO	VALOR
CS	0H
DS	0H
ES	0H
SS	0H
IP	0H
FL	0H
SP	0100H

Las primeras 256 posiciones de memoria (0000H a 00FFH) están reservadas para el monitor y el área del stack del usuario, como muestra la siguiente figura:



Luego la primera posición de memoria RAM que se puede usar será la 0100H.

Siempre que el SDK-86 sea encendido o que la tecla de Reset del sistema sea pulsada, el monitor termina inmediatamente su actual actividad y salta a su rutina de inicialización. Esta rutina inicializa los vectores de interrupción del 1 al 3 como sigue:

Interrupción 1: Single-Step.

Interrupción 2: NMI; tecla INTR.

Interrupción 3: Breakpoint; usada con el comando GO.

Cuando el monitor esté reentrado como resultado de una interrupción Single-Step, NMI o Breakpoint, el monitor guarda los contenidos de los registros del 8086 en el stack del usuario, y, subsiguientemente, saca los contenidos de los registros del stack antes de pedir la entrada de un comando. Hasta que el SP sea inicializado a 0100H (base del stack), el stack inicial reservado al usuario es de 48 bytes (D0H a 0FFH), de los que 26 bytes deben reservarse para los contenidos de los registros por si ocurriera una de las interrupciones vistas anteriormente.

Nótese que en la descripción de los siguientes comandos, el monitor siempre calcula una dirección de memoria física de 20 bits, a partir de un valor del segmento de dirección de 16 bits y un valor de offset (desplazamiento) de la dirección de 16 bits. El valor del segmento de la dirección se introduce primero, se pulsa la tecla ":" (para separar ambas entradas) y luego se introduce el valor de offset de la dirección.

Los dos valores introducidos, cada uno formado por 4 bytes (se meten en hexadecimal), son sumados como muestra el siguiente ejemplo para obtener la dirección de memoria física de 20 bits.

En este ejemplo metemos un valor de segmento y lo multiplicamos por 16 (desplazar 4 bits a la izquierda), y a este valor le sumamos el offset, con lo que obtenemos la dirección de memoria física:

Entra: FE00H:107AH

Valor del segmento	F E 0 0	
(16 bits)	A19	A4
	: +	
Valor del offset	1 0 7 A	
(16 bits)	A15	A0
Dirección física	F F 0 7 AH	
(20 bits)	A19	A0

Si sólo se introduce uno de los valores de la dirección, los dos puntos se pueden omitir, el monitor lo interpreta como una entrada de un valor de offset de la dirección y el contenido actual del registro CS se usa como valor del segmento de dirección. El contenido del CS y del offset que se introdujo, se suman como se muestra en el siguiente ejemplo para formar la dirección de memoria física de 20 bits:

Entra: 2AH (offset) y el CS contiene 010H.

Valor del CS	0 0 1 0	
(16 bits)	A19	A4
	+	
Valor del offset	0 0 2 A	
(16 bits)	A15	A0
Dirección física	0 0 1 2 A	
(20 bits)	A19	A0

En la explicación de los comandos se usará la siguiente sintaxis:

X	Indica una tecla del teclado.
[A]	Indica que "A" es opcional.
[A]*	Indica una o más funciones opcionales de "A".
<B>	Indica que "B" es una variable.

COMANDOS DEL PROGRAMA MONITOR

COMANDOS	FUNCION / SINTAXIS
EXAMINAR BYTE	DISPLAYA / MODIFICA EL CONTENIDO DE UN BYTE DE MEMORIA EB <direc> , [[<dato>] ,]% .
EXAMINAR PALABRA	DISPLAYA / MODIFICA EL CONTENIDO DE UNA PALABRA DE MEMORIA EW <direc> , [[<dato>] ,]% .
EXAMINAR REGISTRO	DISPLAYA / MODIFICA EL CONTENIDO DE UN REGISTRO ER <tecla reg>[[<dato>] ,]%[.]
ENTRADA BYTE	DISPLAYA EL BYTE DE ENTRADA DE UN PUERTO IB <direc. puerto> , [,]% .
ENTRADA PALABRA	DISPLAYA LA PALABRA DE ENTRADA DE UN PUERTO IW <direc. puerto> , [,]% .
SALIDA BYTE	MANDA UN BYTE A UN PUERTO DE SALIDA OB <direc. puerto> , <dato>[, <dato>]% .
SALIDA PALABRA	MANDA UNA PALABRA A UN PUERTO DE SALIDA OW <direc. puerto> , <dato>[, <dato>]% .
GO	TRANSFIERE CONTROL DESDE EL MONITOR A UN PROG. DE USUARIO GO [<direc.>][, <direc. ruptura>] .
TRANSFERIR	TRANSFIERE BLOQUES DE DATOS DE UNA POSICION A OTRA MV <direc. comienzo> , <direc. final> , <direc. destino> .
SINGLE STEP	EJECUTA PASO A PASO LAS INSTRUCCIONES DE UN PROGRAMA ST <direc. comienzo> , [[<direc. comienzo>] ,]% .

II-4-1.- Comandos Examinar byte (EB) y Examinar palabra (EW).

Se utilizan para examinar los contenidos de las localizaciones de memoria seleccionadas. Si están en la memoria RAM, pueden ser cambiados.
La sintaxis es:

EB: EB <dir.> , [[<dato>] ,]* .

EW: EW <dir.> , [[<dato>] ,]* .

Para usarlas, presionar la tecla EB o EW cuando el guión de petición de comando se visualice. Al presionar uno de los comandos, aparecerá el punto a la derecha del campo de direcciones, y desde el teclado, se introduce la dirección de memoria del byte o la palabra que se quiere examinar, primero el carácter más significativo. Cuando no se especifica el valor segmentado, se tomará el contenido actual del registro CS. Cuando se especifica un valor segmento, se introducen los dos puntos como separador, y la segunda dirección entrada es el valor de offset. La capacidad de entrada de un campo de direcciones está limitada a cuatro caracteres, y solo son válidos los cuatro

últimos caracteres visualizados.

Después de introducir la dirección, presionar la tecla ",". El dato de un byte o palabra contenido en la posición de memoria direccionada será presentado en el campo de datos, y un punto decimal aparecerá a la derecha del campo de datos, para indicar que cualquier subsiguiente entrada hexadecimal por el teclado será dirigida al campo de datos.

Nótese que cuando se usa el comando EW, el byte contenido en la posición de memoria presentada en el display aparece en los dos dígitos menos significativos del campo de datos, y el byte contenido en la siguiente dirección de memoria (dirección de memoria más uno) aparece en los dos dígitos más significativos del campo de datos.

Nótese que cuando se usa el comando EW, el byte contenido en la posición de memoria presentada en el display, aparece en los dos dígitos menos significativos del campo de datos, y el byte contenido en la siguiente dirección de memoria (dirección de memoria más uno) aparece en los dos dígitos más significativos del campo de datos.

Si los contenidos de la posición de memoria direccionada sólo van a ser examinados, presione la tecla "." para terminar la operación, o presione la tecla "," para examinar la siguiente posición de memoria (EB), o las dos siguientes posiciones de memoria (EW). Para modificar el contenido de una posición de memoria, introduzca el nuevo dato desde el teclado. El dato visualizado no será cargado en memoria hasta que se presione la tecla "," o la ".". Si se presiona la tecla "." el comando es finalizado, y el guión de petición de comando será visualizado. Si se presiona la tecla "," serán visualizados la dirección de offset y el dato contenido en la siguiente posición de memoria (EB) o siguientes posiciones de memoria (EW).

Intentar modificar una posición de memoria que no existe, o una posición de memoria de solo lectura, lleva a error, que no se detecta hasta que se presione la tecla "," o la ".". Entonces se visualizarán los caracteres "ERR" con el guión de petición en el campo de direcciones.

A continuación se verán unos ejemplos.

EJEMPLO 1: Examinar una serie de posiciones de byte de memoria relativas al registro CS.

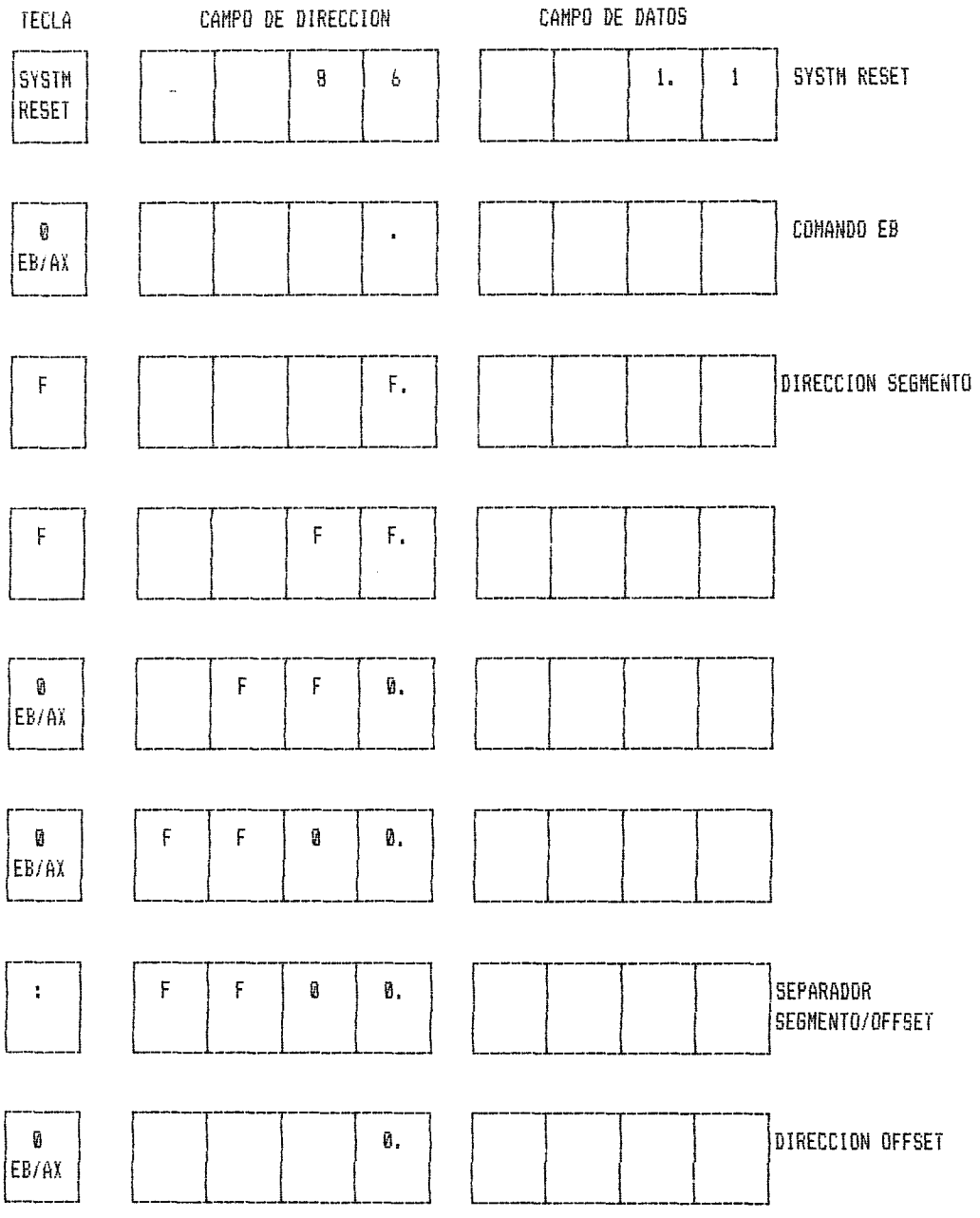
TECLA	CAMPO DE DIRECCION	CAMPO DE DATOS	
SYSTM RESET	- 0 6	1. 1	SYSTM RESET
0 EB/AX	.		COMANDO EB
1 ER/BX	1.		PRIMERA POSICION DE MEMORIA A EXAMINAR
9 OW/DS	1 9.		
,	1 9	X X.	DATO CONTENIDO EN MEMORIA
,	1 A	X X.	SIGUIENTE POSICION DE MEMORIA Y SU CONTENIDO
,	1 b	X X.	SIGUIENTE POSICION DE MEMORIA Y SU CONTENIDO
,	1 C	X X.	SIGUIENTE POSICION DE MEMORIA Y SU CONTENIDO
.	-		TERMINACION Y PETICION DE COMANDO

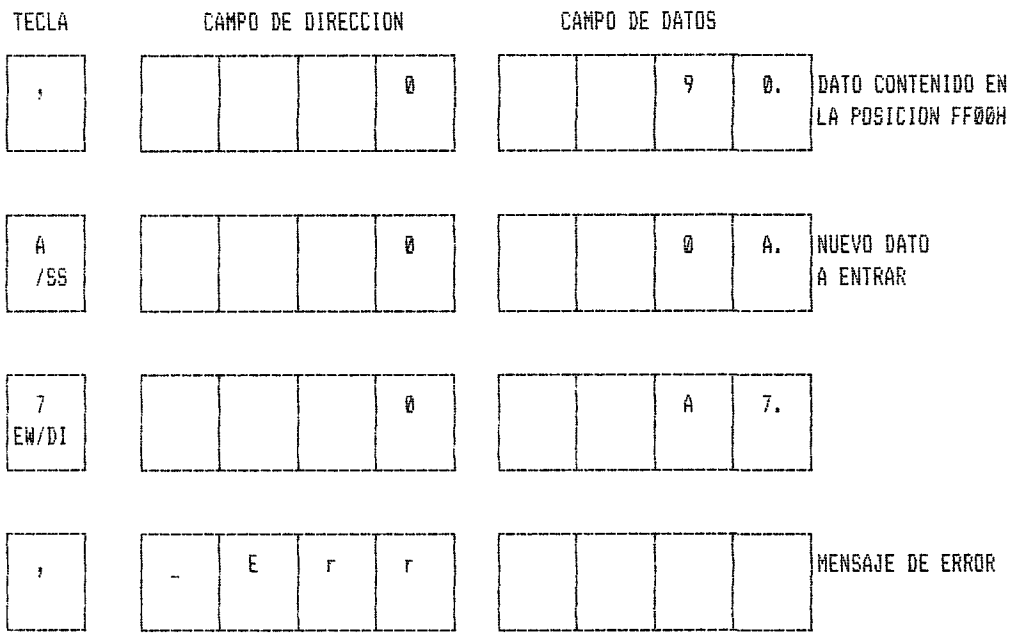
EJEMPLO 2: Examinar y modificar la palabra de memoria cuya dirección es la 10H relativo al registro DS.

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
SYSTM RESET	-		8	6			1.	1	SYSTM RESET
7 EW/DI				.					COMANDO EW
REG	r			.					ENTRADA REGISTRO
9 OW/DS				0.					REGISTRO DS
:				0.					SEPARAR SEGMENTO/OFFSET
1 ER/BX				1.					DIRECCION OFFSET
0 EB/AX			1	0.					
,			1	0	X	X	X	X.	DATO CONTENIDO EN MEMORIA

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
8 IW/CS			1	0	0	0	0	8	NUEVO DATO A SER ENTRADO
C /IP			1	0	0	0	8	C	
F			1	0	0	8	C	F.	
B /ES			1	0	8	C	F	b.	
.	-								DATO ENTRADO, TERMINACION Y PETICION DE COMANDO

EJEMPLO 3: Intentar modificar en la memoria PROM.





II-4-2.- Comando Examinar registro (ER).

Se usa para examinar, y si se desea, modificar el contenido de cualquiera de los registros del 8086. La sintaxis es:

ER: ER<tecla reg.>[[<dato>] ,]*[.]

Para examinar el contenido de un registro, presione la tecla ER cuando se pida la entrada de un comando.

Cuando la tecla es presionada, se iluminará el punto decimal a la derecha del campo de direcciones. La siguiente entrada por teclado será interpretada como el nombre de un registro, y no como su valor hexadecimal.

Cuando se presione la tecla hexadecimal, la abreviación del registro correspondiente aparecerá visualizada en el campo de direcciones, el contenido del registro de 16 bits será visualizado en el campo de datos, y se iluminará el punto decimal a la derecha del campo de datos. En la siguiente tabla se define el nombre de los registros del 8086, la sigla hexadecimal del teclado, la abreviación visualizada y la secuencia en la cual será examinado el registro.

NOMBRE DEL REGISTRO	TECLA	ABREVIATURA DISPLAY
Acumulador	AX	A
Base	BX	b
Contador	CX	C
Dato	DX	d
Puntero de pila	SP	SP
Puntero de base	BP	bP
Indice fuente	SI	SI
Indice destino	DI	dI
Segmento código	CS	CS
Segmento dato	DS	dS
Segmento de pila	SS	SS
Segmento extra	ES	ES
Puntero de instrucción	IP	IP
Flags	FL	FL

El contenido de un registro se puede modificar. Al teclear un valor desde el teclado hexadecimal éste será visualizado en el campo de datos, y el contenido del registro será sobregrabado, con el valor del dato visualizado, cuando sea presionada una de las teclas "," ó "." . Si se presiona la tecla "." se termina el comando, y el carácter de petición de comando será visualizado.

Si se presiona la tecla ",", la abreviación y el contenido del siguiente registro son visualizados, y queda abierto para una modificación. Nótese que la secuencia no es circular y que presionando la tecla ",", cuando el registro FL se visualiza, se terminará el comando ER y se retorna el control al carácter de petición de comando.

A continuación se verán algunos ejemplos.

EJEMPLO 1: Examinar y modificar un registro.

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
SYSTM RESET	-		8	6			1.	1	SYSTM RESET
I ER/BX				.					COMANDO ER
B /ES			E	S	0	0	0	0.	CONTENIDO DEL REGISTRO ES
I ER/BX			E	S	0	0	0	1.	NUEVO CONTENIDO DEL REGISTRO
0 EB/BX			E	S	0	0	1	0.	
.	-								REGISTRO GRABADO TERMINACION Y PETICION DE COMANDO

EJEMPLO 2: Examinar una serie de registros.

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
SYSTM RESET	-		8	6			1.	1	SYSTM RESET
1 ER/BX				.					COMANDO ER
9 OW/DS			d	S	0	0	0	0.	CONTENIDO DEL REGISTRO DS
,			S	S	0	0	0	0.	CONTENIDO DEL REGISTRO SS
,			E	S	0	0	0	0.	CONTENIDO DEL REGISTRO ES
,			I	P	0	0	0	0.	CONTENIDO DEL REGISTRO IP
,			F	L	0	0	0	0.	CONTENIDO DEL REGISTRO FL
,	-								TERMINACION Y PETICION DE COMANDO

II-4-3.- Comandos Entrada de byte (IB) y Entrada de palabra (IW):

Se usan para habilitar la entrada de un byte de 8 bits o de una palabra de 16 bits desde un puerto de entrada.

La sintaxis es:

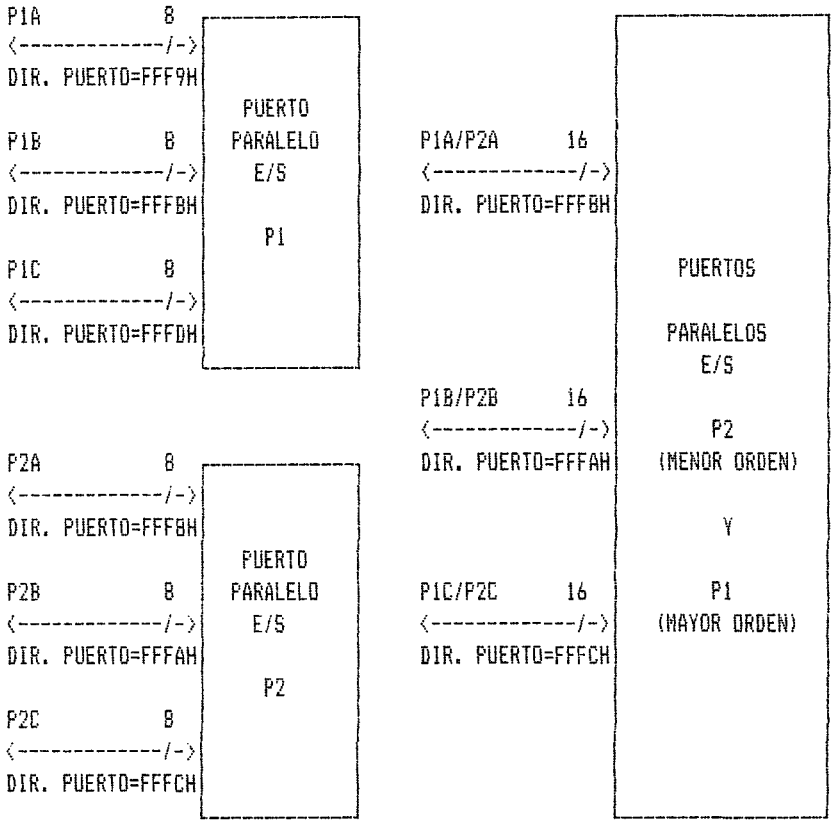
IB: IB<dir.puerto> , [,]* .

IW: IW<dir.puerto> , [,]* .

Para usarlas se presiona la tecla hexadecimal correspondiente cuando aparezca el guión de petición de comando. Cuando se pulsa cualquiera de estas dos teclas, se iluminará el punto decimal a la derecha del campo de direcciones para indicar que se requiere la entrada de la dirección de un puerto. Introduzca la dirección del puerto que debe leerse, usando el teclado hexadecimal. Nótese que desde que el direccionamiento de E/S está limitado a 64 K (máxima dirección FFFFH), no se permiten valores segmentados de las direcciones de puertos.

Después de que se haya introducido la dirección del puerto, presione la tecla "," . El byte de entrada o la palabra de entrada del puerto direccionado será visualizado en el campo de datos. Presionando otra vez la tecla "," se sobregrabará el display del campo de datos con el byte de entrada o palabra de entrada que hay ahora en el puerto de entrada direccionado. Presionando la tecla "." se termina el comando y se visualiza el carácter de petición de comando. El SDK-86 incluye dos circuitos 8255A puertos de E/S paralela que pueden usarse con los comandos de entrada de byte y de entrada de palabra para introducir datos que provengan de dispositivos periféricos.

Estos dos circuitos se denominan P1 y P2. Cada circuito, a su vez, consiste en tres puertos individuales de 8 bits, que se denominan A, B y C, como se muestra en la siguiente figura. Cada puerto opera independientemente durante operaciones de bytes. Durante operaciones de palabra, operan juntos pares de puertos para formar datos de palabra de 16 bits, correspondiendo P2 al byte de menor peso (si se utiliza, por ejemplo, el par P1A-P2A).

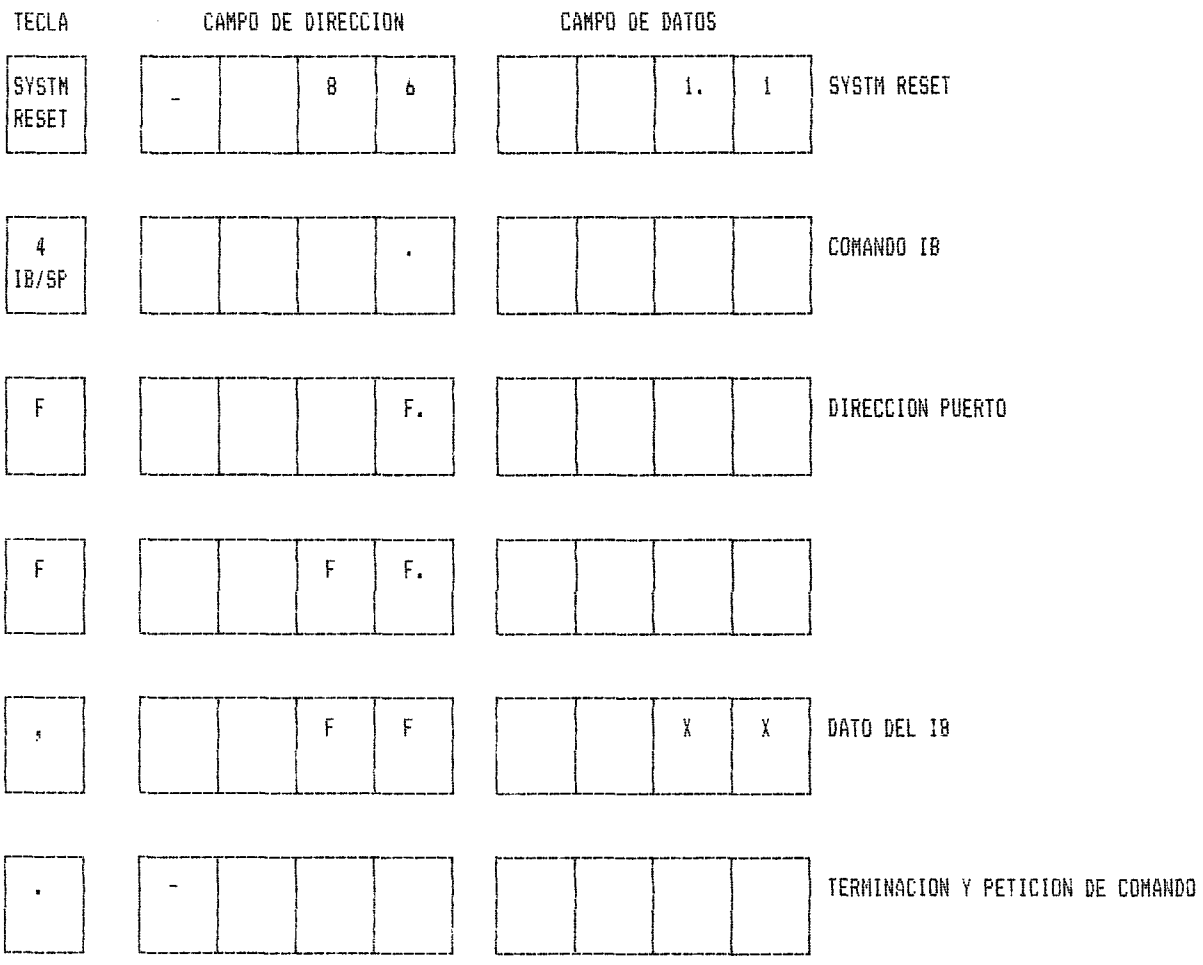


La siguiente tabla, nos muestra las direcciones de puertos individuales. Nótese que durante operaciones de palabras, se introduce la dirección del puerto de menor peso (p2), ya que la correspondiente dirección del puerto de mayor peso será direccionada automáticamente.

PUERTO	DIRECCION
P2A	FFF8H
P1A	FFF9H
P2B	FFFAH
P1B	FFFBH
P2C	FFFCH
P1C	FFFDH

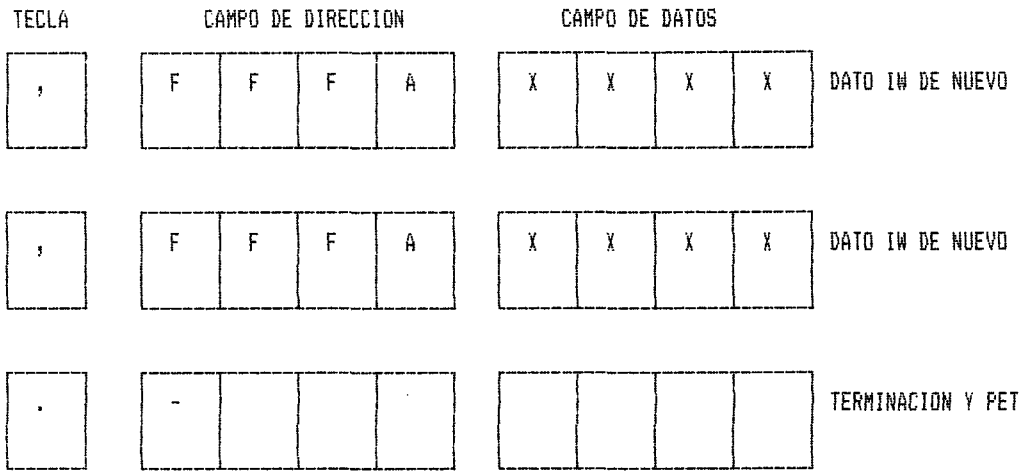
Los circuitos de puertos de E/S paralela están programados como entradas para el momento de encendido o cuando quiera que pulse la tecla System Reset. Si los circuitos han sido programados como salida previamente, presione la tecla System Reset, antes de presionar la tecla del comando a usar, o saque el valor apropiado del byte o palabra del puerto de control del circuito para programar los puertos como entradas.

EJEMPLO 1: Entrada de un sólo byte desde el puerto ØFFH (puerto no incluído en el SDK-86).



EJEMPLO 2: Entrada de una palabra desde los puertos de E/S paralela 1B y 2B.

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
SYSTM RESET	-		8	6			1.	1	SYSTM RESET (INICIALIZAR PUERTOS PARA ENTRADA)
B IW/CS				.					COMANDO IW
F				F.					DIRECCION PUERTO B
F			F	F.					
F		F	F	F.					
A /SS	F	F	F	A.					
,	F	F	F	A	X	X	X	X	DATO IW DESDE EL PUERTO



II-4-4.- Comandos Salida de byte (OB) y Salida de palabra (OW):

Se usan para sacar datos o palabras por un puerto de salida.

La sintaxis es:

OB: OB<dir.puerto> , <dato>[, <dato>]* .

OW: OW<dir.puerto> , <dato>[, <dato>]* .

Para usarlos, presione la tecla hexadecimal correspondiente cuando aparezca el guión de petición de entrada de un comando. Al presionar una de las teclas, se iluminará el punto decimal a

la derecha del campo de direcciones para indicar que se requiere la entrada de una dirección de puerto de salida. Las direcciones están limitadas a 64 k y no se permiten valores segmentados. Después de introducir la dirección del puerto, presione la tecla ",". Se iluminará el punto decimal a la derecha del campo de datos para indicar que se puede introducir el byte o la palabra que va a ser sacada. Después de que el dato se ha introducido, presione la tecla "." para sacar el byte o la palabra al puerto de salida y terminar el comando, o presione la tecla "," si se va a mandar un dato adicional al puerto de salida direccionado.

Como los circuitos de puertos de E/S están programados como entradas cuando se enchufa el SDK-86 y cuando se pulsa la tecla System Reset, se deben programar como salidas (sacando los datos de byte o de palabra apropiados a los puertos de control del circuito), antes de que los datos sean sacados por los puertos asociados. La siguiente tabla define el direccionamiento del puerto de control y los datos asociados de byte o de palabra a ser sacados al puerto de control.

PUERTO	DIRECCION PUERTO	DATO BYTE O PALABRA ENTRADA	SALIDA
P2	FFFEH	9BH	80H
P1	FFFFH	9BH	80H
P2/P1	FFFEH	9B9BH	8080H

EJEMPLO 1: Sacar el contenido del registro DI por el puerto de salida cuya dirección es OC5H. Este puerto no está incluído en el SDK-86.

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
SYSTM RESET	-		8	6			1.	1	SYSTM RESET
9 OW/DS				.					COMANDO OW
C /IP				C.					DIRECCION PUERTO DE SALIDA
5 OB/BP			C	5.					
,			C	5				.	PERMITIDO DATO ENTRADA

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
REG			C	5	r			.	REG (REGISTRO ENTRADA)
7 EW/DI			C	5	X	X	X	X.	CONTENIDO REGISTRO DI
.	-								DATO SALIDA TERMINACION Y PETICION DE COMANDO

EJEMPLO 2: Programación del puerto P1 como salida.

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
SYSTM RESET	-		8	6			1.	1	SYSTM RESET (INICIALIZAR PUERTO COMO ENTRADA)
5 OB/BP				.					COMANDO OB
F				F.					CONTROL P1 DIRECCION PUERTO
F			F	F.					

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
F		F	F	F.					
F	F	F	F	F.					
,	F	F	F	F				.	PERMITIDO DATO ENTRADA
8 IW/CS	F	F	F	F				8.	CONTROL BYTE PARA SALIDA
0 EB/AX	F	F	F	F			8	0.	
.	-								CONTROL BYTE SALIDA TERMINACION Y PETICION DE COMANDO

II-4-5.- Comando GO:

Se usa para transferir el control del 8086 desde el programa monitor al programa del usuario que se encuentre en memoria.

La sintaxis es:

GO: GO [<dir.>][, <dir.breakpoint (ruptura)>] .

Para usarlo, presione la tecla GO cuando se le pida la entrada de un comando. Cuando la tecla se presiona, el contenido actual del registro IP se visualiza en el campo de direcciones, el dato byte contenido en la posición de memoria direccionada por el IP se visualiza en el campo de datos, y se iluminará el punto decimal a la derecha del campo de direcciones, para indicar que se puede introducir una dirección de comienzo alternativa. Si se requiere una dirección de comienzo alternativa, introduzca tal dirección desde el teclado. Cuando se teclea una dirección, el campo de datos es borrado.

Para que empiece la ejecución del programa (en la instrucción presente o en la dirección alternativa), presione la tecla "." . Cuando la tecla es presionada, el monitor visualiza una "E" en el dígito más significativo del campo de direcciones antes de transferir el control al programa.

Se produce error al intentar una ruptura en una instrucción de la memoria de sólo lectura.

Para salir de la ejecución del programa y devolver el control al programa monitor, presione una de las teclas System Reset o INTR. Si se pulsa la

tecla System Reset se reintroduce el monitor y los registros correspondientes del 8086 son inicializados. Si se pulsa la tecla INTR, se reintroduce el monitor, se almacenan todos los registros del 8086 y el monitor visualiza el guión de petición de comando. Cuando se reintroduce el monitor, debido a que se pulse la tecla INTR, presionando la tecla GO ocurre que el valor actual del IP (el offset de la dirección de la próxima instrucción del programa, que iba a ser ejecutada cuando se pulsó la tecla INTR) y el byte contenida en aquella dirección (direccionada tanto por el IP como por el CS) serán visualizados.

Presionando la tecla "." se devuelve el control, desde el monitor al programa, a la instrucción direccionada y la ejecución del programa continúa.

El comando GO permite, opcionalmente, que sea introducida una dirección de ruptura (breakpoint). Una dirección de ruptura tiene el mismo efecto que presionar la tecla INTR mientras se está ejecutando una programa.

Para introducir una dirección de ruptura, presione la tecla "," después de introducir la dirección de comienzo, e introduzca la dirección de ruptura.

Nótese, que cuando se especifica una dirección de ruptura, el valor segmentado de la dirección que falta puede ser tanto el valor segmentado de la dirección de inicio (si se especifica), como el contenido actual de CS (si no se especifica un valor segmentado con la dirección de inicio). En conjunto, la posición que se especifica por medio de la dirección de ruptura debe contener el primer byte de la instrucción (código de operación o prefijo).

Cuando se presione la tecla "." , el monitor sustituye la instrucción en la dirección de ruptura con una instrucción de interrupción y almacena la instrucción de ruptura antes de transferir el control al programa del usuario. Cuando el programa obtiene la dirección de ruptura, el control se devuelve al monitor, la instrucción de ruptura se carga en el programa, se almacenan los contenidos de los registros y el monitor hace que se visualice el guión de petición de comando para permitir que cualquiera de los registros sea examinado.

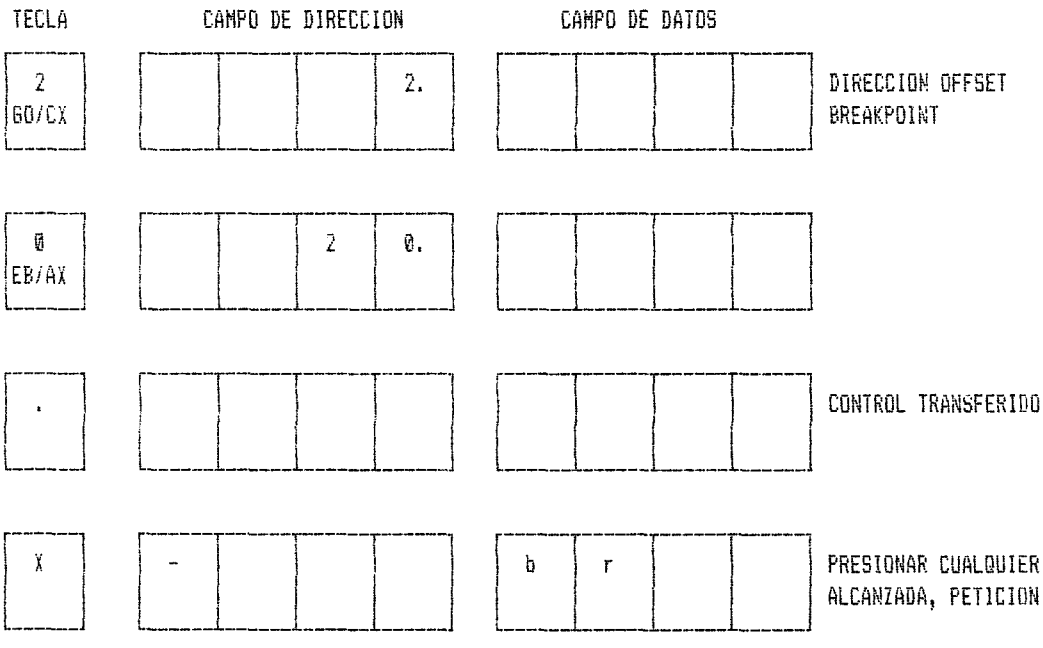
Nótese, que hasta que la instrucción de ruptura sea cargada, cuando el control se devuelve al monitor, la dirección de ruptura debe ser especificada cada vez que el programa se va a ejecutar con una ruptura.

EJEMPLO 1: Transferencia de control a un programa.

TECLA	CAMPO DE DIRECCION	CAMPO DE DATOS	
SYSTM RESET	- 8 6	1. 1	SYSTM RESET
2 GO/CX	0.	X X	COMANDO GO (REGISTRO IP DIRECCION OFFSET Y DATO CONTENIDO)
1 ER/BX	1.		SEGMENTO (REGISTRO CS) DIRECCION
0 EB/AX	1 0.		
:	1 0.		SEPARADOR SEGMENTO/OFFSET
0 EB/AX	0.		DIRECCION OFFSET
.			CONTROL TRANSFERIDO A 0100H

EJEMPLO 2: Introducción y ejecución de una ruptura (Breakpoint) en un programa.

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
SYSTM RESET	-		8	6			1.	1	SYSTM RESET
2 GO/CX				0.			X	X	COMANDO GO
I ER/BX				1.					SEGMENTO (REGISTRO CS) DIRECCION
0 EB/AX			1	0.					
:			1	0.					SEPARADOR SEGMENTO/OFFSET
0 EB/AX				0.					DIRECCION OFFSET
,				.					



II-4-6.- Comando mover (MV):

Permite que se puedan mover (trasladar) bloques de datos (un bloque de datos) dentro de la memoria.

La sintaxis es:

MV: MV <dir.comienzo> , <dir.final> ,
<dir.destino> .

El formato del comando es único, en el cual se hacen tres entradas sucesivas en el campo de direcciones. Para usar el comando MOVE, presione la tecla MV cuando aparezca el guión de petición de comando. Cuando se presiona la tecla . aparecen tres puntos decimales para indicar que se requieren tres entradas. Cada vez que se hace una entrada (de datos), el punto decimal que esté más a la izquierda desaparece, de modo que el número de puntos decimales encendidos coincida con el número de entradas que todavía se requieren.
Por orden, las entradas son:

1- Dirección de comienzo del bloque de datos que se va a desplazar.
2- Dirección del final del bloque de datos que se va a desplazar.
3- Dirección de comienzo (dirección de destino) a la cual se quiere desplazar el bloque de datos.

Nótese que no se permiten valores segmentados con una dirección de final y que el bloque que movamos está limitado consecuentemente a los 64 Kbytes.

Cuando se presiona la tecla ".", se desplaza el dato y se visualiza el guión de petición de comando. Nótese que cuando se desplaza el bloque de datos, el dato contenido en las posiciones de memoria originales no son alterados (a menos que la dirección de destino caiga dentro del bloque de datos original, en cuyo caso las posiciones de memoria superpuestas se sobregrabarán con el dato desplazado).

Cuando se lleva a cabo un movimiento de un byte, cada vez, el comando MOVE puede para cargar un bloque de memoria con una constante. Esto se realiza especificando una dirección de destino que sea mayor que la dirección de comienzo. El bloque de posiciones de memoria, desde la posición de comienzo hasta la posición de final más uno, se cargará con el valor contenido en la posición de la dirección de inicio. El comando EB se puede usar para especificar el contenido de la dirección de comienzo.

Intentar trasladar un dato dentro de la memoria de sólo lectura o a una posición inexistente lleva a error.

EJEMPLO 1: Trasladar un programa de la posición 0100H a la posición 0200H.

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
SYSTM RESET	-		8	6			1.	1	SYSTM RESET
6 MV/SI		.	.	.					COMANDO MV
1 ER/BX		.	.	1.					DIRECCION COMIENZO (0100H)
0 EB/AX		.	1.	0.					
:		.	1.	0.					
0 EB/AX		.	.	0.					
;			.	.					

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
4 IB/SP			.	4.					OFFSET DIRECCION FINAL (Ø4DH)
D /FL			4.	d.					
,				.					
2 GO/CX				2.					DIRECCION DESTINO (Ø2ØØH)
Ø EB/AX			2	Ø.					
:			2	Ø.					
Ø EB/AX				Ø.					
.	-								PROGRAMA TRASLADADO, PETICION DE COMANDO

II-4-7.- Comando (Single-Step) Step (ST):

Permite que las instrucciones de un programa en memoria sean ejecutadas individualmente. Con cada una de las instrucciones ejecutadas, el programa devuelve el control al monitor.
La sintaxis es:

ST: ST <dir.comienzo> , [[<dir.comienzo>] ,]* .

Para usar el comando STEP, presione la tecla ST cuando aparezca el guión de petición de comando. Si se quiere otra dirección de comienzo que no sea la que está visualizada, teclee la dirección deseada. Cuando la tecla "," sea presionada, la instrucción direccionada se ejecuta y se visualiza la dirección de la próxima instrucción a ejecutar, en el campo de direcciones, y su byte de instrucción asociado en el campo de datos. Presionando otra vez la tecla "," se ejecuta la instrucción en curso y se pasa el programa a la siguiente instrucción que va a ser ejecutada.

Restricciones:

1- Si ocurre una interrupción antes de que se complete una instrucción de Single-Step (paso a paso), o si la instrucción de Single-Step genera una interrupción, cuando el monitor es reentrado, los registros CS e IP contendrán la dirección de la rutina de servicio de la interrupción. Consecuentemente, una interrupción del tipo 3 (breakpoint) (ØCCH ó ØCDH) no debería ser ejecutada paso a paso hasta que su ejecución tuviera lugar en el monitor.

2- Una instrucción que sea parte de una secuencia de instrucciones que conmute con el SP (por ejemplo, intercambiar los contenidos de los registros SS y SP) no puede ser ejecutada paso a paso.

3- Una instrucción MOV o POP que modifica un segmento de un registro no puede ser ejecutada paso a paso. El control se devuelve al monitor después de ejecutarse la siguiente instrucción (la intrucción que sigue inmediatamente a la instrucción MOV o POP).

EJEMPLO 1: Programa paso a paso (Single-step).

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS				
3 ST/DX				0.			X	X	COMANDO ST
1 ER/BX				1.					DIRECCION COMIENZO DEL PROGRAMA
0 EB/AX			1	0.					
:			i	0.					
0 EB/AX				0.					
,				2.			8	E	PROXIMA INSTRUCCION
,				7.			b	0	

TECLA	CAMPO DE DIRECCION				CAMPO DE DATOS			
,				9.			E	E
,				A.			b	A
,				d.			E	C
,				E.			2	4
,			1	0.			7	4
,				d.			E	C

TEMA III:

"LENGUAJE ENSAMBLADOR DEL 8086"

III) LENGUAJE ENSAMBLADOR DEL 8086.

III-1) MODOS DE DIRECCIONAMIENTOS.

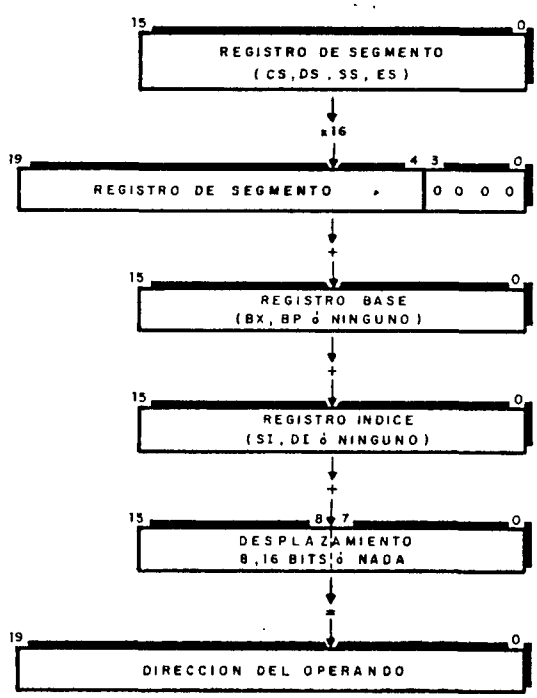
Las instrucciones del 8086 están codificadas con cero, uno o dos operandos (registro, memoria). Las operaciones se hacen entre registros o registros y memoria, pero nunca entre memoria y memoria (salvo para las cadenas de caracteres).

Los modos de direccionamiento de los operandos indican la manera de calcular la dirección real de éstos.

El 8086 tiene 25 modos diferentes de direccionamiento o reglas para localizar un operando de una instrucción. Estos modos pueden verse como casos especiales de referencia a registros y referencia a memoria. En el primer caso, el operando está localizado en un registro especifico. Sin embargo, deben sumarse cuatro cantidades para obtener la dirección de un operando en memoria. Dichas cantidades son:

- -Dirección de segmento.
- -Dirección base.
- -Una cantidad índice.
- -Un decalage.

La siguiente figura nos muestra como se calcula la dirección de un operando, que consiste (en la forma más compleja) en sumar un registro base, un registro índice y un decalage al valor del registro de segmento, multiplicado por 16:



La dirección del segmento se almacena en el registro de segmentación (DS,ES,SS o CS). El contenido del registro segmento se multiplica por 16 (delplazar cuatro dígitos binarios a la derecha). El registro de segmentación siempre se usa para referenciar a memoria.

La base se almacena en el registro base (BX o BP). El índice se almacena en el registro índice (SI o DI). Cualquiera de estas cantidades, las dos, o ninguna, pueden utilizarse para calcular la dirección real.

La base y el índice son variables o dinámicas, ya que están almacenadas en registros de la CPU de propósito general. Es decir, pueden modificarse fácilmente mientras se ejecuta un programa.

También se usan unos decalages (desplazamientos) de 16 bits, 8 bits o 0 bits. Este decalage es una cantidad estática. Se fija el tiempo de ensamblaje (paso de código fuente a código máquina) y no puede cambiarse durante la ejecución del programa. El decalage se utiliza para cosas como compilar datos, organizar la memoria y reubicar más rápida y fácilmente.

En la siguiente tabla se ven los distintos modos de direccionamiento del 8086:

mm	aaa	Parte de desplazamiento de la dirección
00	000	(BX) + (SI)
00	001	(BX) + (DI)
00	010	(BP) + (SI)
00	011	(BP) + (DI)
00	100	(SI)
00	101	(DI)
00	110	*Dirección directa*
00	111	(BX)
01	000	(BX) + (SI) + número 8 bits
01	001	(BX) + (DI) + número 8 bits
01	010	(BP) + (SI) + número 8 bits
01	011	(BP) + (DI) + número 8 bits
01	100	(SI) + número 8 bits
01	101	(DI) + número 8 bits
01	110	(BP) + número 8 bits
01	111	(BX) + número 8 bits
10	000	(BX) + (SI) + número 16 bits
10	001	(BX) + (DI) + número 16 bits
10	010	(BP) + (SI) + número 16 bits
10	011	(BP) + (DI) + número 16 bits
10	100	(SI) + número 16 bits
10	101	(DI) + número 16 bits
10	110	(BP) + número 16 bits
10	111	(BX) + número 16 bits
11	000	registro AX (palabra) o AL (octeto)
11	001	registro CX (palabra) o CL (octeto)
11	010	registro DX (palabra) o DL (octeto)
11	011	registro BX (palabra) o BL (octeto)
11	100	registro SP (palabra) o AH (octeto)
11	101	registro BP (palabra) o CH (octeto)
11	110	registro SI (palabra) o DH (octeto)
11	111	registro DI (palabra) o BH (octeto)

Nota: mm significa los 2 primeros bits del octeto de direccionamiento y aaa los 3 últimos bits de dicho octeto.

En general, los modos de direccionamiento del 8086 se dividen en dos grandes grupos:

- Modos de direcionamiento de la memoria de programa.
- Modos de direccionamiento de la memoria de datos.

III-1-1.- Modos de direccionamiento de la memoria de programa:

En la búsqueda de una instrucción, su dirección se obtiene sumando el desplazamiento, contenido en el IP, al valor del registro de segmento CS, multiplicado por 16. Normalmente, al terminar la ejecución de una instrucción, el IP se incrementa, con lo que se pasa a direccionar la siguiente instrucción. Las instrucciones de salto y salto a subrutina pueden modificar el contenido del IP de tres formas diferentes:

- Por direccionamiento relativo. Al contenido del IP, se suma, de forma inmediata, un desplazamiento de 8 ó 16 bits con signo, proporcionado por la misma instrucción.
- Por direccionamiento directo. Se carga, en el IP, una nueva dirección presente en la instrucción.
- Por direccionamiento indirecto. El dato, obtenido por cualquiera de las formas de direccionado de la memoria de datos, es interpretado por las instrucciones de salto, como la dirección a la que se debe saltar.

III-1-2.- Modos de direccionamiento de la memoria de datos:

- Modo inmediato. El operando se proporciona en el byte o bytes que siguen al código de operación de la instrucción.

  ADD CX,385F

- Modo de direccionamiento por registro. Un registro, definido por la instrucción, contiene el operando.

  ADD CX,AX

- Modo directo. El byte o par de bytes que siguen al código de OP representan un desplazamiento que se suma al contenido de uno de los registros índice (DI o SI). El contenido de DS se añade al resultado de la suma, con lo que se obtiene la dirección del operando. Incrementando o decrementando los registros índice, se puede acceder a posiciones de memoria consecutivas.

- Modo indirecto. La dirección del operando es el contenido de uno de los siguientes registros:

  BP,BX,DI o SI.

- Por registro base indexado. El desplazamiento que ha de sumarse a un registro segmento se halla sumando el contenido de un registro índice y un desplazamiento de 8 ó 16 bits, contenido en la instrucción, al contenido de un registro base. Este modo es el más complejo, habiéndose representado anteriormente.

- Direccionamiento implicado. Es una versión del modo anterior, produciéndose cuando no se especifica desplazamiento en la instrucción. El contenido de un registro índice se suma al DS.

- Modo relativo. Admite dos formas:

  A- Direccionamiento relativo a memoria de datos, en el que interviene el registro DS.

  B- Direccionamiento relativo al stack, en el que interviene el registro SS.

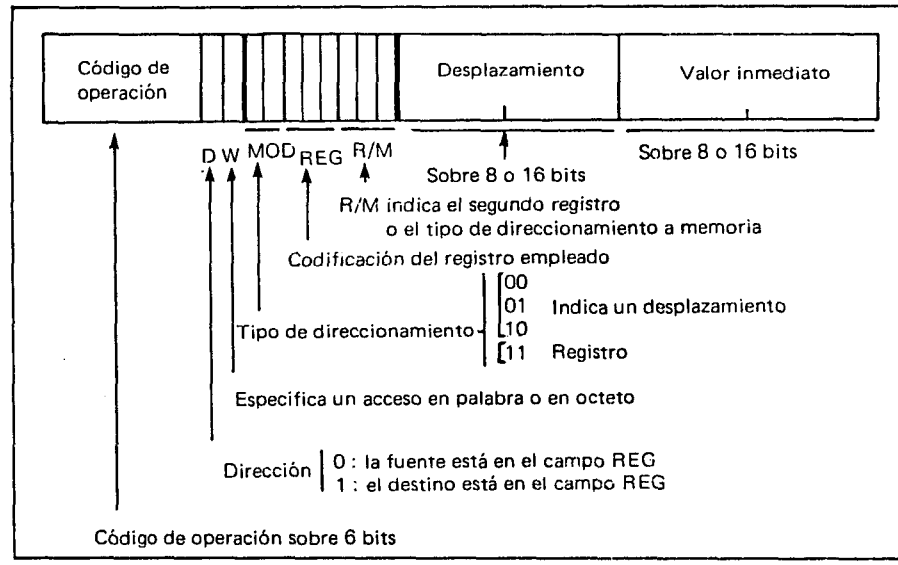
  En cualquiera de las dos modalidades de este direccionamiento, el registro BX proporciona la operación base para la dirección efectiva.

- Para operaciones sobre caracteres, se utilizan los registros índice, con autoincremento y autodecremento.

III-2) CODIFICACION DE INSTRUCCIONES.

Las instrucciones se codifican sobre cuatro campos y tienen un tamaño de uno a seis bytes. El primer campo es el código de operación, el segundo el modo de direccionamiento, el tercero el desplazamiento del dato y el cuarto el valor inmediato.

En la siguiente tabla vemos la codificación de las instrucciones:



III-3) SINTAXIS DE LAS INSTRUCCIONES.

Las instrucciones se codifican en cuatro campos:

- Campo de etiqueta:

  Es opcional, constituída de 1 a 31 caracteres. Deben empezar por una letra o por uno de los tres símbolos especiales : ? , , -

- Campo del código de operación:

  Son los nemónicos dados por el fabricante.

- Campo de operandos:

  Constituido por 0, 1 ó 2 operandos.

- Campo de comentarios:

  Es opcional. Si se pone debe empezar por un punto y coma (;).

Cada campo debe estar separado del anterior, por lo menos, por un espacio en blanco.

III-4) INSTRUCCIONES DEL 8086.

Para las instrucciones del 8086 usaremos los nemotécnicos de Microsoft. Con estos nemotécnicos muchas instrucciones indican (como parte de su código de operación) si los operandos son en octetos o palabras, o si la fuente es un dato inmediato. Una "B" extra en el nemotécnico de la operación indica modo byte (datos de 8 bits), mientras que su ausencia indica modo palabra (datos de 16 bits). Una "I" extra en el nemotécnico de la operación indica datos inmediatos en la fuente.

Cuando la fuente es un dato inmediato de 8 bits, entonces aparecerán la "B" y la "I" (en este orden) en el nemotécnico de la operación. El uso de la "B" y de la "I" como parte del código de operación no debe confundirse con los modos de direcionamiento y la forma en que éstos afectan al operando.

Algunas instrucciones no tienen operandos, otras tienen uno y otras tienen dos. Para estas últimas, siempre se escribe primero el destino, separándolo con una coma del fuente. El destino es donde se almacena el resultado, mientras que el fuente es la entrada y no es modificado por la operación.

III-4-1.- Instrucciones de Transferencia de Datos:

Este grupo de instrucciones permite realizar la transferencia de datos entre registros, posiciones de memoria o puertos de entrada/salida.

Las subdividiremos en cuatro tipos:

- Instruciones generales.
- Instrucciones de E/S.
- Instrucciones de manejo de pila.
- Instrucciones especiales de transferencia.

III-4-1-1.- Instrucciones Generales:

. MOV destino, fuente

La instrucción MOV transfiere el contenido (byte o palabra) del operando fuente al operando destino.

El operando fuente puede ser una posición de memoria, un registro general o un registro de segmento. El operando destino puede ser también un valor inmediato. Todas las combinaciones de estos operandos son posibles excepto la transferencia entre dos posiciones de memoria.

Esta instrucción queda optimizada usando AX.

. XCHG destino, fuente

Intercambia los contenidos de los operandos especificados. El operando destino puede ser un registro de uso general (de 8 ó 16 bits) o una posición de memoria, mientras que el fuente ha de ser un registro de uso general.

Esta instrucción se optimiza con Ax y registro de 16 bits.

III-4-1-2.- Instrucciones de E/S:

. IN puerto

Esta instrucción realiza la entrada de información desde un puerto al acumulador. Posee dos formatos:

En el primero, el puerto es un valor inmediato de 8 bits y la información es de 8 bits transfiriéndose al registro AL.

En el segundo, la dirección se encuentra en el registro DX y la información (de 16 bits) se transfiere al registro AX.

. OUT puerto

Esta instrucción es idéntica a la anterior exceptuando que la transferencia se realiza desde el acumulador al puerto especificador.

III-4-1-3.- Instrucciones de manejo de Pila:

. PUSH fuente

La instrucción PUSH transfiere el contenido del operando fuente a la pila, decrementando el puntero de pila (SP) en dos. La transferencia es siempre de 16 bits. El operando puede ser un registro de uso general, de segmento o una posición de memoria.

. POP destino

Recoge una palabra de 16 bits de la pila y la deposita en el operando destino, incrementando el SP en dos unidades. El operando puede ser un registro de uso general, de segmento o una posición de memoria.

Estas instrucciones se optimizan si el operando es un registro.

III-4-1-4.- Instrucciones Especiales:

. XLAT Tabla

Esta instrucción lleva a cabo una carga del acumulador (AL) con el contenido de un elemento de una tabla. En el registro BX se encontrará la dirección de la tabla, y en el AL el número del elemento al que deseamos acceder. El resultado es que el contenido de la posición de memoria indicada por la suma de BX y AL es trasladado al registro AL.

((BX) + (AL)) ------> AL

Esta instrucción se utiliza con gran frecuencia en la traducción de códigos a través de tablas.

. LEA destino, fuente

Transfiere la dirección efectiva (offset) del operando fuente al operando destino. El operando fuente ha de ser una posición de memoria y el operando destino un registro general de 16 bits.

. LDS destino, fuente

Transfiere los 32 bits de una variable puntero desde el operando fuente, al operando destino y el registro DS. Los 16 bits del offset del puntero son transferidos al operando destino, el cual tiene que ser un registro general de 16 bits.

La palabra que contiene la base del puntero es transferida al registro DS. El operando fuente será una dirección de memoria donde se inicia el almacenamiento del puntero.

. LES destino, fuente

La operación de esta instrucción es la misma que la LDS, salvo que la base del puntero se transfiere al registro ES.

III-4-2.- Instrucciones Aritméticas y Lógicas:

III-4-2-1.- Formato de los Datos:

Este microprocesador puede operar aritméticamente con cuatro tipos diferentes de datos. Estos son:

- Números Binarios sin signo (enteros positivos).
- Números Binarios con signo (enteros).
- Números Decimales sin signo.
- Números de Cifra Decimal sin signo.

Los números binarios sin signo pueden tener 8 bits (de Ø a 255) ó 16 bits (de Ø a 65.535) de longitud. Las operaciones permitidas con este tipo de datos son: suma, resta, multiplicación y división.

Los números binarios con signo pueden ser de 8 ó 16 bits. El bit de mayor orden indica el signo (Ø positivo, 1 negativo). Los números negativos son representados en complemento a 2. Como el bit de mayor orden es utilizado para el signo, el rango de este tipo de datos es de -128 a 127 para los de 8 bits y de -32.768 a 32.767 para los de 16 bits. El valor Ø está considerado como positivo.

Los números decimales sin signo son almacenados en bytes. Estos se consideran divididos en dos mitades, correspondiente cada cuatro bits a un digito decimal. Sólo son válidas las combinaciones de cuatro bits correspondiente a los digitos Ø al 9, el resto de las combinaciones serán rechazadas en las operaciones o darán un resultado indefinido. El rango de este tipo de datos se encuentra entre Ø y 99.

En las operaciones aritméticas el indicador AF (acarreo intermedio) puede ser considerado como las centenas.

En operaciones de suma y resta, es necesario un proceso posterior denominado ajuste decimal, para el que hay instrucciones como DAA y DAS. El citado proceso consiste en restablecer el formato de número decimal al resultado de la operación de suma o resta.

Para operaciones de multiplicación y división no existen instrucciones que realicen el ajuste decimal, debiendo quedar el resultado en binario.

Los números de cifra decimal son almacenados en bytes, siendo significativos únicamente los 4 bits más bajos. En estos 4 bits son válidas únicamente las combinaciones correspondientes a los dígitos Ø al 9. Los bits más significativos deben estar a Ø.

Para la suma, resta y multiplicación, estos números son tratados como binarios sin signo, aunque es necesario también un ajuste posterior que puede llevarse a cabo a través de las instrucciones:

AAA para la suma.

AAS para la resta.

AAM para la multiplicación.

Respecto a la división, es necesario un ajuste previo del registro AL.

La siguiente tabla muestra un ejemplo de como son tratados los distintos tipos de datos citados:

HEX	MODELO	BIN	BIN CON SIGNO	DEC	CIF DEC
07	00000101	7	+7	7	7
89	10001001	137	-119	89	X
C5	11000101	197	-59	X	X

III-4-2-2.- Instrucciones de dos Operandos:

Este grupo de operaciones posee un tipo de operandos común. Todas disponen de un operando destino que puede ser un registro o una posición de memoria de 8 bits o de 16 bits. El operando fuente puede ser un registro, una posición de memoria o un valor inmediato de 8 ó 16 bits. Todas las combinaciones son posibles, exceptuando cuando ambos operandos son posiciones de memoria.

Los indicadores (flags) afectados en todas ellas son: AF, CF, OF, PF, SF y ZF.

DESTINO	FUENTE
Registro	Registro
Registro	Memoria
Memoria	Registro
Registro	V. inmediato
Memoria	V. inmediato

. ADD destino, fuente

Efectúa la suma de los contenidos de los operandos, los cuales pueden ser bytes o palabras. El resultado queda en el operando destino.

Esta instrucción se optimiza usando AX.

. ADC destino, fuente

Realiza una operación similar a la anterior, salvo que se suman los dos operandos y el indicador de acarreo. Esta instrucción se utiliza cuando se quiere sumar dos cantidades con longitud mayor de 16 bits, siendo necesario en ese caso, realizar varias sumas parciales de 16 bits, arrastrando los acarreos.

Esta instrucción se optimiza usando AX.

. SUB destino, fuente

Se resta el contenido del operando destino menos el operando fuente, quedando el resultado en el operando destino.

Esta instrucción se optimiza usando AX.

. SBB destino, fuente

Esta operación es similar a la SUB, salvo que realiza la resta del operando destino menos el fuente y menos el indicador de acarreo (CF). Se utiliza cuando se desea restar cantidades con longitud mayor de 16 bits.

Esta instrucción se optimiza usando AX.

. AND destino, fuente

Realiza la operación lógica AND entre el operando fuente y el operando destino, dejando el resultado en el operando destino.

Los indicadores OF y CF son siempre 0 y AF es indefinido.

. OR destino, fuente

Realiza la operación lógica OR entre el operando fuente y el operando destino, dejando el resultado en el operando destino.

Los indicadores OF y CF son siempre Ø, y AF es indefinido.

. XOR destino, fuente

Realiza la operación lógica XOR entre el operando fuente y el operando destino, dejando el resultado en el operando destino.

Los indicadores OF y CF son siempre Ø, y AF es indefinido.

. CMP destino, fuente

Resta el contenido del operando destino menos el contenido del operando fuente, pero no deposita el resultado en el operando destino, tan sólo afecta a los indicadores para que éstos puedan ser testeados por una instrucción de salto posterior.

. TEST destino, fuente

Realiza la operación lógica AND entre los operandos, pero no deposita el resultado en el operando destino, tan sólo afecta a los indicadores SF, ZF y PF, con objeto de que éstos puedan ser usados en las instrucciones de salto condicional.

III-4-2-3.- Instrucciones de un sólo Operando:

. INC destino

Esta instrucción realiza el incremento en una unidad del valor del operando especificado. El operando puede ser un registro o una posición de memoria de 8 ó 16 bits y es tratado como un binario sin signo.

Afecta a los indicadores: AF, OF, PF, SF y ZF.

Se suele usar para el control de bucles.

. DEC destino

Esta instrucción realiza el decremento en una unidad del valor del operando especificado. El operando puede ser un registro o una posición de memoria de 8 ó 16 bits y es tratado como un binario sin signo.

Afecta a los indicadores: AF, OF, PF, SF y ZF.

Se suele usar para el control de bucles.

. NEG destino

Realiza el complemento a dos del operando especificado. El operando puede ser un registro o una posición de memoria de 8 ó 16 bits. Si se intenta negar el valor -127 ó - 32.768, no

cambia el valor del operando pero se activa el indicador del overflow (OF).

Afecta a los indicadores: AF, OF, PF, SF y ZF (el indicador CF se activa siempre excepto cuando el operando tiene un valor 0).

. NOT destino

Realiza el complemento a 1 del operando especificado. El operando puede ser un registro o una posición de memoria de 8 ó 16 bits. No afecta a ningún indicador.

III-4-2-4.- Instrucciones de Multiplicación y División:

. MUL fuente

Efectúa una multiplicación sin signo entre el operando fuente y el acumulador. El operando puede ser un registro o una posición de memoria de 8 ó 16 bits.

Si el operando fuente es un byte, es multiplicado por el registro AL y el resultado, que será de doble longitud, se depositará en los registros AH y AL.

Si el operando fuente es una palabra de 16 bits, la operación se efectúa con el registro AX y el resultado pasa a los registros DX y AX, ya que este es de doble longitud.

Los operandos son tratados como números binarios sin signo. Si en el resultado los registros AH (para fuentes de 8 bits) o DX (para fuentes de 16 bits) son distintos de 0, CF y OF son 1. En cualquier otro caso estos indicadores son 0.

Los contenidos de los indicadores AF, PF, SF y ZF son indefinidos.

. IMUL fuente

Efectúa una multiplicación con signo entre el operando fuente y el acumulador. Esta instrucción es idéntica a la anterior, salvo que se toma en consideración el signo de los operandos, produciendo un resultado con signo.

Si en los registros AH (para fuentes de 8 bits) o DX (para fuentes de 16 bits) no se conserva la extensión del signo del resultado, CF y OF son puestos a 1, indicando que AH o DX guardan dígitos significativos.

. DIV fuente

Efectúa una división sin signo en el que el dividendo es el acumulador y el divisor es el operando fuente. El operando puede ser un registro o una posición de memoria de 8 ó 16 bits.

Si el operando fuente es de 8 bits, se toma como dividendo el registro AX, depositando el cociente del resultado en el registro AL y el resto en el registro AH.

Si el operando fuente es de 16 bits, el dividendo se toma formado por los registros DX y AX, dando el cociente del resultado en el registro AX y el resto en el registro DX.

Si en el resultdo, el cociente excede la capacidad del operando destino, o cuando se intenta una división por cero, se genera una interrupción tipo 0 y el cociente y el resto son indefinidos.

El contenido de los indicadores AF, CF, OF, PF, SF y ZF es indefinido.

. IDIV fuente

Esta instrucción realiza la misma operación que la anterior, salvo que toma en consideración el signo de los operandos y lo conserva convenientemente en el resultado. Se aplican aquí las mismas consideraciones que para la instrucción DIV.

Hay dos instrucciones asociadas con la división, que permiten manejar el signo de los operandos con mayor facilidad. Estas son:

. CWB

Extiende el bit de signo del registro AL en el AX.Convierte por tanto, el valor del registro AL (8 bits en una palabra (16 bits) en el registro AX.Puede ser utilizada para generar un dividendo de doble longitud a partir de un byte en el AL, con objeto de llevar a cabo una división con signo de tipo byte.

. CWD

Extiende el bit de signo del registro AX al DX. Transforma un valor entero de 16 bits en uno de 32 bits. Se utiliza para generar un dividendo doble palabra a partir de un dato ubicado en el registro AX.

III-4-2-5.- Instrucciones de Ajuste:

Como la CPU realiza todas las operaciones aritméticas en código binario, este conjunto de instrucciones se utiliza para pasar los resultados a decimal.

. AAA

Después de ser afectada una operación de suma con dos números de cifra decimal que sean válidos, el resultado lo tenemos en el acumulador en binario.

La instrucción AAA ajusta el contenido del acumulador para que corresponda a un número de cifra decimal válido. Afecta a los indicadores AF y CF, siendo indefinidos los valores de los indicadores OF, PF, SF y ZF.

. DAA

Esta instrucción es idéntica a la AAA cuando los operandos son de tipo decimal.

. AAS

Esta operación es idéntica a la AAA salvo que realiza el ajuste a un número de cifra decimal

después de la resta. El resto de las consideraciones son las mismas que para la instrucción AAA.

. DAS

Esta instrucción es idéntica a la AAS salvo que realiza el ajuste a un número decimal después de la resta.

. AAM

Después de haberse realizado la multiplicación entre dos números de cifra decimal válidos, los contenidos de los registros AH y AL son transformados en dos números de cifra decimal correspondiendo cada uno de ellos a un registro.
Los operandos que se hayan multiplicado, habían de tener los cuatro bits más significativos a 0, para que se produzca un resultado correcto en esta instrucción.
Se actualizan los indicadores PF, SF y ZF.
El contenido de AF, CF y OF es indefinido.

. AAD

Si se desea que el resultado de una división corresponda a un número de cifra decimal, es necesario primeramente que el valor del acumulador sea de cifra decimal.

Para conseguirlo, se emplea esta instrucción, que convierte el valor del registro AL en valor de cifra decimal, poniendo AH a 0. La división posterior introducirá el cociente en el registro AH y el resto en el AL.

Se actualizan los indicadores PF, SF y ZF.

El contenido de AF, CF y OF es indefinido.

III-4-3.- Instrucciones de Transferencia de Control de Programa:

La secuencia de ejecución de instrucciones de un programa viene determinada por el registro de segmento de código (CS) y el puntero de instrucciones (IP). La combinación de ambos registros define la dirección física de memoria donde serán buscadas las instrucciones.

Hay cuatro grupos de instrucciones de transferencia:

- Instrucciones de transferencia incondicional.
- Instrucciones de transferencia condicional.
- Instrucciones de iteración.
- Instrucciones de interrupción.

III-4-3-1.- Instrucciones de Transferencia Incondicional:

Dado que en la búsqueda de instrucciones entran en juego los dos registros citados CS e IP, cuando se desee alterar el flujo normal de instrucciones (forzado por el incremento automático del registro IP), se deberá modificar el contenido del registro IP o de ambos (IP y CS). Si sólo se modifica el contenido de IP, el salto se produce dentro del mismo segmento de código y la transferencia se denomina Intrasegmento.

Si la transferencia se desea realizar fuera del segmento es imprescindible modificar el contenido de ambos registros (CS e IP), cargando en el registro CS la nueva extensión de segmento y en el IP el nuevo desplazamiento dentro del segmento. Este tipo de transferencia se denomina Intersegmento.

La diferencia entre ambos tipos de instrucciones consiste en que en el primer caso, en la instrucción figura únicamente el valor (16 bits) que tomará el registro IP, mientras que en el segundo, se especifican 32 bits, 16 correspondientes al registro de segmento CS y 16 al registro IP.

En los programas escritos en ensamblador, no se establece diferencia ninguna, y son las herramientas automáticas de traducción, las que determinan la necesidad de uso de uno u otro tipo de instrucciones.

Existen además tres formas de llevar a cabo la transferencia: relativa a programa directa e indirecta.

- La transferencia relativa a programa posee la característica de que el valor que se indica en la instrucción (16 bits) será sumado o restado (dependiendo del signo) al registro IP, indicando por tanto la nueva dirección de memoria donde proseguirá el programa. Este tipo de transferencia se realiza solo intrasegmento.
- La transferencia directa provoca que los 32 bits que figuran en la propia instrucción, sean cargados directamente en los registros CS e IP.

Este tipo de transferencia se lleva a cabo intersegmento y por tanto no se puede omitir ninguno de los 32 bits en la instrucción.

- En la transferencia indirecta, no figuran en la instrucción los nuevos valores del (de los) registro (s), sino que en ella se referencia un registro general o una posición de memoria donde se encuentran los nuevos valores de IP y/o CS.

  Aquí hay dos posibilidades:

  Si la transferencia es intrasegmento se puede utilizar un registro general o una posición de memoria de 16, bits para contener el nuevo valor de IP.

  si la transferencia es intersegmento, ha de utilizarse una doble palabra puntero en memoria, indicándose la dirección inicial en la propia instrucción. En esta doble palabra se encontrarán los nuevos valores de IP y CS.

Las instrucciones son:

. JMP destino

Transfiere el control a una posición concreta del programa. El tipo de transferencia puede ser de cualquiera de los tipos mencionados:

- Relativa a programa intrasegmento (denominada por Intel directa intrasegmento).
- Indirecta intrasegmento.
- Directa intersegmento.
- Indirecta intersegmento.

```
JMP destino ;salto directo en el mismo segmento
                (intrasegmento).
JMP destino,segmento ; salto directo a un nuevo
                       segmento (intersegmento).
JMP destino ;salto corto (8 bits).
JMP destino ;salto indirecto en el mismo segmento
                (intrasegmento).
JMP destino ;salto indirecto a un nuevo segmento
                (intersegmento).
```

. CALL destino

Esta instrucción produce un salto a la dirección especificada, guardando la dirección de vuelta en la pila, con objeto de que una vez terminado el subprograma, pueda continuarse en el punto donde se dejó el programa principal.

Los tipos de transferencia son los mismos que para la instrucción JMP. En las transferencias intrasegmento se guarda en la pila el registro IP decrementándose el puntero de pila (SP) en 2. En las transferencias intersegmento se guarda el registro CS en la pila, decrementándose en 2 el puntero de pila (SP) y a continuación se guarda el IP volviéndose a decrementar en 2 el SP.

. RET (valor opcional)

Transfiere el control a la instrucción de vuelta de la subrutina, que fué guardada en la pila cuando se ejecutó la instrucción CALL. Para ello recoge de la pila la información de vuelta e incrementa el SP.

Hay dos tipos de instrucciones RET, dependiendo del tipo de la instrucción de llamada CALL: intersegmento e intrasegmento. En la instrucción RET intersegmento, se recoge de la

pila tanto el registro IP como el CS, incrementando el SP en 4. En la instrucción RET intrasegmento, se recoge sólo el IP de vuelta, y por tanto se incrementa el SP en 2. Esto implica que las subrutinas están catalogadas en intersegmento e intrasegmento, y no puede ser utilizada una CALL intersegmento para una subrutina intrasegmento, ya que ésto implicaría la ejecución de una instrucción RET errónea.

Además, si se especifica un valor opcional en la propia instrucción, este valor será sumado al SP una vez incrementado. Esto se posibilita con objeto de desechar posibles parámetros que se hayan pasado a través de la pila antes de llevar a cabo la llamada mediante CALL.

III-4-3-2.- Instrucciones de Transferencia Condicional:

Las instrucciones de transferencia condicional son saltos que transfieren el control del programa dependiendo del estado de los indicadores (flags). Si la condición que testean es verdadera, el control es transferido a la instrucción destino especificada, en caso contrario se sigue en la secuencia normal del programa.

Todos los saltos condicionales son del tipo "relativo a programa", admitiéndose un desplazamiento comprendido entre -128 y 127 (desplazamiento de 8 bits). Si el desplazamiento es negativo, debe figurar en complemento a 2.

A- Salto condicionado por los flags:

Instrucción	Condicion	Salta si...
JC objetivo	CF=1	Hay acarreo.
JNC "	CF=0	No hay acarreo.
JE "	ZF=1	es igual.
JNE "	ZF=0	no es igual.
JZ "	ZF=1	es cero.
JNZ "	ZF=0	no es cero.
JS "	SF=1	es negativo.
JNS "	SF=0	no es negativo.
JO "	OF=1	hay overflow.
JNO "	OF=0	no hay overfl.
JP "	PF=1	hay paridad.
JPO "	PF=0	no hay paridad.
JPE "	PF=0	paridad par.

B- Otro grupo de instrucciones de transferencia condicional lo constituyen aquellas que, a través de un indicador o una combinación de varios, permiten comparar dos valores, mediante el uso de la instrucción CMP previo a la de salto. En estos casos hay que distinguir dos casos: cuando el valor es entero positivo (sin signo) o cuando el valor es entero con signo.

Para valores sin signo:

Instrucción	Condición	Salta si...
JB objetivo	CF=1	es inferior.
JNAE "	CF=1	no es superior o igual.
JNB "	CF=0	no es inferior.
JAE "	CF=0	es superior o igual.
JBE "	(CF or ZF)=1	es inferior o igual.
JNA "	(CF or ZF)=1	no es superior.
JNBE "	(CF or ZF)=0	no es inferior o igual.
JA "	(CF or ZF)=0	es mayor.

Para valores con signo:

Instrucción	Condición	Salta si...
JL objetivo	(SF xor OF)=1	es menor.
JNGE "	(SF xor OF)=1	no es mayor o igual.
JNL "	(SF xor OF)=0	no es menor.
JGE "	(SF xor OF)=0	es mayor o igual.
JLE "	((SF xor OF) or ZF)=1	es menor o igual.
JNG "	"	no es mayor.
JNLE "	((SF xor OF) or ZF)=0	no es menor o igual.
JG "	"	es mayor.

III-4-3-3.- Instrucciones de Iteración:

Las instrucciones de iteración son utilizadas para controlar la repetición de los bucles de programa. Estas instrucciones usan el registro CX como contador de repeticiones. Son instrucciones de transferencia del tipo "relativo a programa", admitiéndose un desplazamiento entre -128 y 127. Se colocan al final del bucle.

. LOOP destino

Decrementa CX en 1 y transfiere control al operando destino si CX no es cero. Cuando CX es Ø se ejecuta la siguiente instrucción a LOOP.

No se modifica ningún flags.

. LOOPE/LOOPZ destino

Esta instrucción, que admite dos nemotécnicos, decrementa CX en 1 y transfiere el control si CX es distinto de Ø y ZF es 1 (el decemento de CX no afecta a los indicadores). Si no se cumple la anterior condición el control es transferido a la instrucción siguiente. Con esta instrucción se puede realizar el bucle con doble condición, un número máximo de veces y la igualdad de dos operandos establecida mediante una instrucción previa, posiblemente un CMP (repetir hasta n veces o hasta que A <>B).

. LOOPNE/LOOPNZ destino

Es idéntica a la anterior con la diferencia que transfiere el control al lugar especificado si CX es distinto de Ø y ZF es Ø. La doble condición aquí la constituye: la repetición un número máximo de veces y la desigualdad de dos

operandos (repetir hasta n veces o hasta que A=B).

. JCXZ destino

Transfiere control al operando destino si CX = 0. Esta instrucción utilizada al principio de un bucle facilita la omisión de su ejecución si se especificó un número de veces igual a 0 (CX = 0).

III-4-3-4.- Instrucciones de Interrupción:

Una interrupción provoca la ejecución de un subprograma forzado por un evento externo. En este caso, en el ciclo de reconocimiento de interrupción, además del puntero de instrucción IP y del registro CS, se guarda en la pila el registro de indicadores. Esto implica que, a la vuelta, ha de restablecerse dicho registro de indicadores, recogiéndolo de la pila. Por tanto, no puede utilizarse la instrucción RET normal, sino que ha de usarse la IRET que realiza la función citada.

. IRET

Instrucción de vuelta del subprograma de interrupción.

. INTO

Genera una interrupción software tipo 4 (interrupción de overflow) si el indicador OF esta activado (OF=1). Si OF estuviera desactivado esta instrucción no tiene ningún efecto.

Esta instrucción suele utilizarse despues de una operación aritmética que pueda provocar overflow, con objeto de que sea tratado este evento.

Con objeto de facilitar el debugging de programas, este microprocesador ofrece la posibilidad de provocar interrupciones por programa, lo que facilita la realización de pruebas controladas, acerca de la respuesta a eventos externos. En este sentido la ejecución de la instrucción INT permite emular mediante programa la aparición de una interrupción externa.

. INT tipo de interrupción

Guarda en la pila el contenido de los registros CS, IP y de indicadores saltando a la rutina de tratamiento del tipo de interrupción especificado. El tipo de interrupción debe encontrarse entre 0 y 255. También desactiva los indicadores IF y TF, con lo que quedan prohibidas las interrupciones y se desactiva el modo paso a paso.

III-4-4.- Instrucciones de Control:

Hay un conjunto de instrucciones que manejan algunos indicadores directamente, activándolos o desactivándolos de forma individual.

. CLC

Pone a 0 el indicador de acarreo CF.

. CMC

Complementa el valor del indicador de acarreo CF.

. STC

Activa (pone a 1) el indicador de acarreo CF.

. CLD

Pone a 0 el indicador de dirección DF, usado en las instrucciones de manejo de bloques.

. STD

Pone a 1 el indicador de dirección DF.

. CLI

Prohibe las interrupciones desactivando el indicador de permiso de interrupción IF=0. Las interrupciones afectadas por este indicador son las externas exceptuando las NMI.

. STI

Permite las interrupciones activando el indicador de permiso de interrupción IF=1.

Hay un conjunto de instrucciones de control del procesador, que realizan funciones diversas, algunas de ellas relacionadas con la existencia de varios procesadores cooperantes en un entorno multiproceso.

. HLT

Esta instrucción provoca la parada del microprocesador, hasta que se active la línea RESET, la NMI o alguna otra interrupción (si éstas estaban permitidas).

. NOP

Esta instrucción no realiza operación alguna en su ejecución.

. WAIT, ESC y LOCK

Se usan para sistemas con Multiproceso.

ESC: Pone en funcionamiento el procesador paralelo.

WAIT: Para sincronizar la CPU y el procesador paralelo (NOP 8087). Esta hace que la CPU pare hasta que le entre una señal por el terminal TEST indicándole que continúe.

El terminal TEST de la CPU se conecta en estos casos al terminal BUSY del procesador paralelo.

LOCK: Para prevenir conflictos. Evita que el bus sea utilizado por más de un procesador durante la instrucción siguiente a ella (IOP 8089).

Estas instrucciones han sido estudiadas cuando se analizaron las facilidades que ofrece el 8086 para llevar a cabo sistemas con multiproceso.

. LAHF

Carga en el registro AH algunos de los indicadores: SF (bit 7), ZF (bit 6), AF (bit 4), PF (bit 2), CF (bit 0), el contenido de los bits 5, 3 y 1, son indefinidos. No afecta a los flags.

. SAHF

Esta instrucción realiza una operación similar a la LAHF, excepto que se transfieren los indicadores contenidos en el registro AH al registro de indicadores. Los indicadores OF, DF, IF y TF no son afectados.

. PUSHF

Esta instrucción transfiere el contenido del registro de indicadores a la pila, decrementando el puntero de pila (SP) en dos.

. POPF

Recoge una palabra de la pila y la deposita en el registro de indicadores, incrementando el puntero de pila (SP) en dos.

III-4-5.- Instrucciones de Desplazamiento y Rotación:

Este grupo de instrucciones permite llevar a cabo desplazamientos de los bits de una palabra de 8 ó 16 bits.

Estas rotaciones o desplazamientos pueden efectuarse un número de veces especificado en la instrucción mediante un operando denominado contador. Este operando puede ser: la constante "1" o el contenido del registro CL.

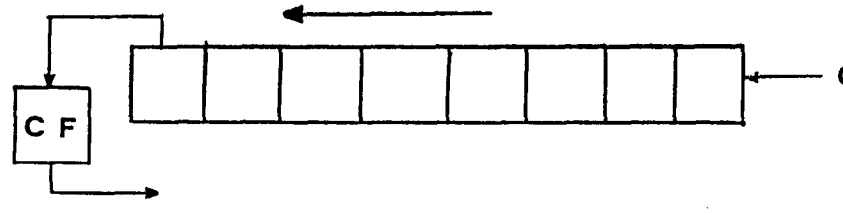
En estas instrucciones se afectan algunos indicadores, entre los que se encuentra: el flag AF cuyo valor es siempre indefinido, los indicadores PF, SF y ZF, que son actualizados normalmente, el CF que contiene siempre el bit expulsado del registro que se desplaza y el OF cuyo valor está definido únicamente si el desplazamiento se realiza una vez, activándose (valor 1) si el valor del bit de mayor orden cambia, quedando a cero en caso contrario.

El operando destino puede ser un registro general o una posición de memoria.

. SHL/SAL destino

  SHL/SAL destino, contador

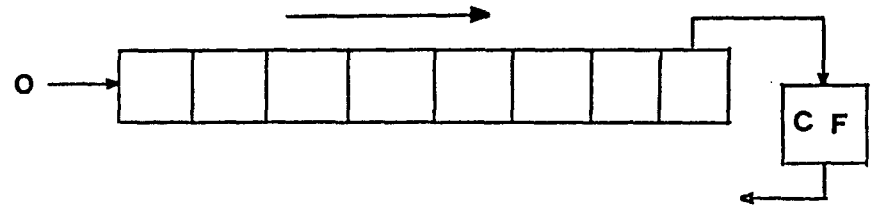
  Ambos nemotécnicos, indican "desplazamiento Lógico a la izquierda" y "desplazamiento aritmético a la izquierda" respectivamente. Representan a la misma instrucción, cuyo efecto es el de desplazar los bits hacia la izquierda un número de veces especificado. Los lugares que quedan por la derecha son rellenados con ceros. Los bits que salen por la izquierda pasan por el indicador CF, que almacenará el último de ellos.



. SHR destino

SHR destino, contador

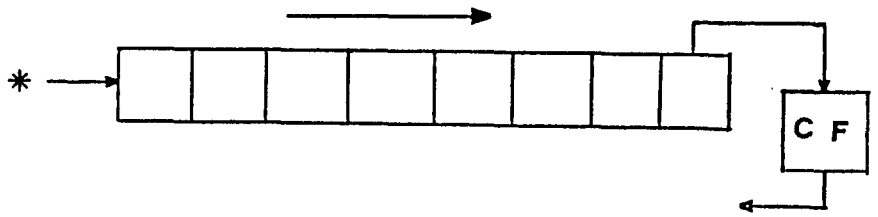
Desplaza los bits hacia la derecha un número de veces especificado. Los lugares que quedan por la izquierda son rellenados con ceros. Los bits que salen por la derecha pasan por el indicador CF, que almacenará el último de ellos.



. SAR destino

SAR destino, contador

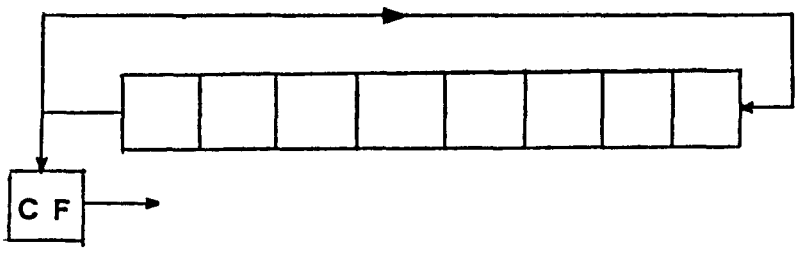
Desplaza los bits hacia la derecha un número de veces especificado. Los lugares que quedan por la izquierda son rellenados con el bit de signo original con objeto de mantener el signo inicial. Los bits que salen por la derecha pasan por el indicador CF, que almacenará el último de ellos.



. ROL destino

ROL destino, contador

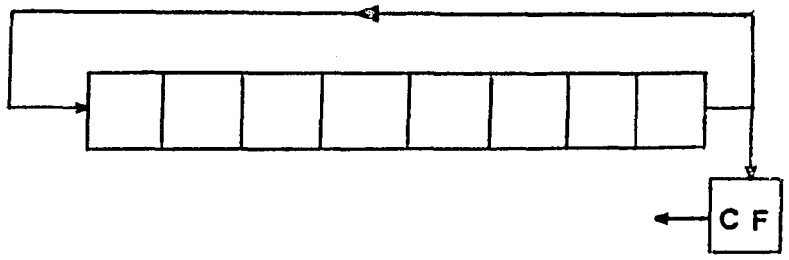
Rota los bits del operando hacia la izquierda un número de veces especificado. En este caso el bit expulsado por la izquierda vuelve a entrar por la derecha, además de pasar al indicador CF.



. ROR destino

ROR destino, contador

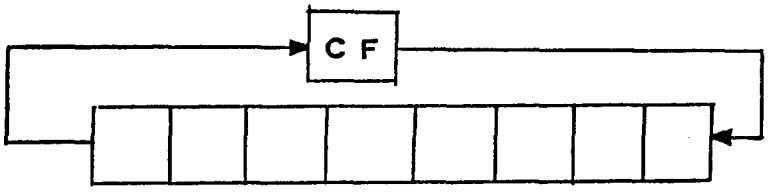
Es idéntica a la ROL, salvo que la rotación se realiza hacia la derecha.



. RCL destino

RCL destino, contador

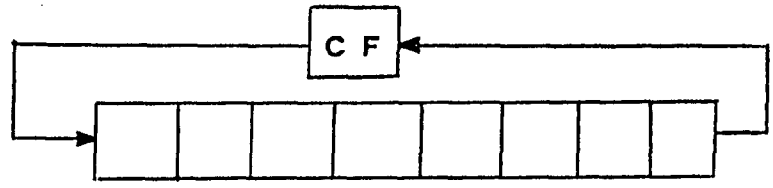
Rota los bits del operando hacia la izquierda utilizando el CF como parte del ciclo en la rotación, estando su lugar entre el bit de mayor orden y el de orden mas bajo.



. RCR destino

RCR destino, destino

Es idéntica a la RCL, salvo que la rotación se realiza hacia la derecha.



III-4-6.- Instrucciones de Manejo de Strings (cadenas):

Un string es una cadena de elementos (de 8 o 16 bits) que pueden llegar a ocupar 64 K bytes de memoria. En este microprocesador existen instrucciones que facilitan el manejo de dichos strings. Entre estas facilidades existen las de trasladar, buscar y comparar strings.

Para aumentar la velocidad de operación con ellos, existe un conjunto de prefijos especiales, que provocan la repetición automática de la operación. Esta repetición se controla por hardware, siempre mas rápida que por programa.

Una instrucción de este tipo puede tener operandos fuente, operandos destino o ambos. El operando fuente ha de residir en el segmento de datos y el operando destino en el segmento extra, manejados por los registros DS y ES respectivamente.

Para direccionar los elementos del string se utilizan los registros indice DI y SI. El registro SI contiene el desplazamiento (respecto del origen del segmento) del string fuente, mientras que el registro DI contiene el del string destino.

Estos registros (SI y DI) deben ser inicializados, indicando el comienzo del string, antes de la ejecución de una instrucción de este tipo, para ello se recomienda utilizar las instrucciones LDS, LES y LEA.

Las instrucciones de string actualizan automáticamente ambos registros SI y DI, apuntando al siguiente elemento a procesar. El indicador DF indica si para operar con el siguiente elemento han de incrementarse (DF=0) o decrementarse (DF=1)

los registros índice. Si los elementos del string son de 8 bits, la actualización de los registros se produce en una unidad, si son de 16 bits la actualización es de dos unidades.
Si se ha utilizado un prefijo de repetición, se usa el registro CX como contador, el cual se decrementa en uno cada vez que se ejecuta la instrucción. El registro CX ha de ser inicializado con el número de repeticiones, antes de la ejecución de la instrucción de string. Si CX = 0 el control pasa a la instrucción siguiente:

Los prefijos de repetición son:

. REP

  Repite mientras no finalice el string (y mientras CX <> 0).

. REPZ/REPE

  Repite mientras no termine el string y mientras ZF <> 0. El indicador ZF debe ser puesto a 1 antes de ejecutar la instrucción.

. REPNZ/REPNE

Repite mientras no termine el string y mientras ZF =0.

Las instrucciones de string (cadenas) son:

. MOVS (cadena destino, cadena fuente)

Transfiere un byte o palabra desde el string fuente al string destino, actualizando los registros SI y DI. Si se utiliza con un prefijo de repetición mueve bloques de memoria.

Normalmente el programa Ensamblador tiene suficiente información para determinar el tamaño de la información a transferir, si no fuera así, pueden utilizarse los nemotécnicos: MOVSB que indica 8 bits y MOVSW que indica 16 bits.

. CMPS (cadena destino, cadena fuente)

Compara dos elementos de los string fuente y destino, restando el elemento destino menos el fuente sin generar resultado alguno, pero afectando los indicadores.

Si CMPS tiene como prefijo REPE la operación se interpreta como: "buscar dos elementos iguales en los strings". Si el prefijo es REPNZ la operación sería: "buscar dos elementos distintos en los strings".

Actualiza los registros SI y DI.

. SCAS (cadena destino)

Compara el elemento del string, con el acumulador (AL o AX dependiendo de si se opera con 8 o 16 bits respectivamente). Para ello resta el registro AL/AX menos el elemento del string, sin producir resultado, afectando únicamente los indicadores. Actualiza el registro DI.

Con los prefijos de repetición, la instrucción busca en el string, un elemento igual (REPE), o distinto (REPNE), del contenido en el acumulador.

. LODS (cadena fuente)

Transfiere el byte o palabra del elemento del string direccionado por SI, al AL o AX respectivamente. Esta instrucción no se suele preceder de ningún prefijo de repetición, ya que

en el acumulador quedaría únicamente el último elemento del string.

. STOS (cadena destino)

Transfiere el contenido del registro AL o AX (dependiendo de si se opera con bytes o palabras) al elemento del string direccionado por el registro DI. Esta instrucción utilizada con un prefijo de repetición proporciona una forma conveniente de inicializar un string.

III-5) LENGUAJE DE ENSAMBLE. SINTAXIS.

El programa Ensamblador para el 8086 admite un formato libre en la construcción de las sentencias pudiéndose colocar cada campo en cualquier parte de la línea, siempre que se mantenga el orden relativo de ellos.

Las sentencias del lenguaje están clasificadas en dos tipos: Instrucciones y Pseudoinstrucciones. Las primeras se vieron ya en el apartado anterior. Veremos aquí, por tanto, las pseudoinstrucciones.

El formato de las sentencias es el que sigue:

(etiqueta:) (prefijo) nemotécnico (operando,...) (;comentario).

El formato de las Pseudoinstrucciones es:

(etiqueta) nemotécnico (operando,...)(;comentario).

Los nombres de variables y las etiquetas admiten hasta 31 caracteres alfanuméricos sin restricción. Se acepta también el carácter "-" como parte del identificador.

Los nemotécnicos fueron estudiados en el apartado anterior.

Respecto a los operandos, estos pueden ser: nombres de registros, valores constantes (representados por cantidades numéricas o identificadores) o posiciones de memoria (representando a variables o constantes).

Cuando se hace referencia a una posición de memoria (variable o constante), puede especificarse a través de los siguientes elementos:

(reg. segmento:) ([reg base]) ([reg index]) Identificador.

Por ejemplo: MOV AX, ES: [BP][SI] ALMACEN.

Cualquiera de los campos puede ser omitido. Los registros que figuran en los corchetes, son usados para calcular la dirección efectiva, y pueden ser, como ya se vió en un apartado anterior:

III-5-1.- Segmentos:

Los programas escritos en lenguaje de ensamble, han de estar organizados en segmentos, ya que esta es la forma que el 8086 tiene de operar. Todas las instrucciones, variables y constantes deben estar situadas dentro de uno de ellos.

El programador puede definir el número de segmentos que desee sin limitación alguna.

Los segmentos se definen en el programa por medio de pseudoinstrucciones directivas. Una de ellas va al principio y otra al final, conteniendo al segmento.

```
etiqueta        SEGMENT
                   :
                   :
                   :
etiqueta        ENDS
```

Existe la posibilidad de indicar al programa ensamblador el tipo de tratamiento que se hará con el segmento por el programa montador (LINKER). Esto se especifica mediante un operando que puede ser:

- ..... (omitido). En este caso el segmento no será encadenado con ningún otro.
- PUBLIC. El segmento será encadenado con otros del mismo nombre.
- COMMON. Indica que el segmento irá superpuesto con otros del mismo nombre.
- AT direcc. Asigna dirección al segmento en tiempo de compilación.
- STACK. Indica que el segmento contendrá a la pila.

Es posible que en un programa tengamos varios segmentos de código, varios de datos y de pila. Los segmentos se utilizan para establecer los direccionamientos en el programa. Por ejemplo, si existen dos segmentos de código y queremos realizar un salto de uno a otro, el programa ensamblador generará una instrucción de salto intersegmento, mientras que si la etiqueta referenciada en la instrucción se encuentra dentro del mismo segmento, se generará un salto intrasegmento.

El uso de segmentos proporciona al programa gran versatilidad e independencia de manera que el cambio de la ubicación de un segmento, no altera más que las referencias que existen externas a él, permaneciendo inalterados los elementos internos a él. Facilita enormemente el diseño modular.

III-5-1-1.- Directiva ASSUME:

El programa ensamblador, al realizar la traducción de las sentencias, puede optar por varias alternativas y para dilucidar cual es la adecuada, necesita saber en cada momento que segmentos se están utilizando. Esto se logra con la pseudoinstrucción ASSUME, que tiene una misión exclusviamente informativa y no vinculante. Con la pseudo ASSUME se informa de los segmentos que se asocian con los registros de segmento. Su formato es:

ASSUME (reg. segmento: nomb segmento, ...)

donde reg. segmento puede ser: CS, DS, ES o SS. La directiva ASSUME puede ir en cualquier parte del programa y aparecer el número de veces que desee.

La pseudo ASSUME no afecta en ningún caso el contenido de los registros de segmento, e incluso puede no coincidir lo que afirma esta pseudo, con el contenido real del registro, lo que en todo caso será responsabilidad del programador.
Esta directiva no puede ser omitida, salvo en el caso de que en toda referencia a datos se especifique el registo de segmento que se utiliza.

III-5-1-2.- Directiva GROUP:

Existe una Directiva que permite agrupar dos o más segmentos bajo un mismo nombre. Se denomina GROUP y su formato es:

nombre grupo GROUP (nom segmento,....).

El nombre del grupo de segmentos podrá ser utilizado en la misma forma que el nombre de un segmento.
Antes de poder acceder a los datos del programa, es necesario inicializar los registros de segmento con los valores adecuados. Repetimos que la directiva ASSUME no realiza ninguna acción en este sentido. Es imprescindible inicializar los

registros de segmento DS y SS. El ES sólo ha de inicializarse si va a ser utilizado. El registro CS no se inicializa puesto que su valor es fijado en el RESET y actualizado en las instrucciones de Transferencia de control de programa.

III-5-2.- Segmentos de Datos:

Las constantes que se pueden utilizar en este lenguaje de ensamble son: binarias, decimales, octales y hexadecimales.
Todos los valores constantes han de ser enteros representables en 16 bits, incluyendo el bit de signo. Los números negativos son representados en complemento a dos.

III-5-2-1.- Definición de Datos:

Todos los datos a utilizar por el programa tienen que estar definidos en un segmento de datos. El formato de definición de datos es:

(etiqueta) nemotécnico operando, ... (;comentario).

Los nemotécnicos utilizados son: EQU, DW, DB y DD.

La pseudo EQU sirve para asignar una constante a un identificador. Esta constante puede ser usada en el programa a través de su identificador.

Por ejemplo:

```
contador  EQU 20H.
```

Las pseudos DB, DW y DD, sirven para reservar espacio donde almacenar bytes, palabras de 16 bits y dobles palabras (32 bits) respectivamente. Estas directivas pueden inicializar con un valor las variables para las que reservan espacio. Su formato es:

```
                DB   "valor inicial" o "?",...
identificador   DW                o
                DD   n DUP ("valor inicial" o "?").
```

Con el formato superior se reserva un elemento por cada valor (o "?") del tamaño especificado, inicializándose con este (no se inicializan si figura "?"). Si el formato es el que figura en la parte inferior, se reservan "n" elementos y se inicializan con el valor especificado (o no se inicializan en caso de que figure el "?").

Veamos algunos ejemplos:

```
ALFA   DB  ?   (reserva un byte sin inicializar).
BETA   DW  9   (reserva una palabra inicializada
                con el valor 8).
DELTA  DD  10  DUP  (0FH)  (reserva 10 dobles
                palabras inicializadas a 0FH).
GAMMA  DB  'MENSAJE'  (reserva 7 bytes y los
                inicializa con los caracteres ASCII).
```

Cuando se define una variable mediante el operador DUP, en realidad lo que se está definiendo es un "array" y podrá ser accedido a través de la indexación mediante un registro.

```
TABLA    DW    10 DUP (?).
           :
           :
         MOV   AX, TABLA [SI]
```

III-5-2-2.- El Indicador PTR:

Supongamos que en un programa se tienen las instrucciones:

```
MOV    CX, [BX].
MOV    [BX], VARIABLE.
DEC    [BX]
```

En los dos primeros casos el ensamblador sabe perfectamente si el dato es de 8 o 16 bits, ya que se le está indicando en la instrucción, en el primero con CX y en el segundo con VARIABLE, que debe estar definida anteriormente. En el tercer caso, el programa ensamblador no sabe si la posición de memoria que indica el registro BX va a ser tratada como byte o como palabra al decrementarla. Para estos casos existe el indicador PTR.

Instrucciones de este tipo quedarían:

```
MOV     WORD PTR    [DI], 95.
MOV     BYTE PTR    [BX], OFFH.
JMP     DWORD PTR   [BX].
INC     BYTE PTR    [SI].
```

III-5-3.- Tipos de Datos:

Al igual que en algunos lenguajes estructurados, aqui se pueden definir tipos de datos, que serán utilizados para la posterior definición de variables y manipulación en el programa. No entraremos excesivamente a fondo en las posibilidades de estos tipos de datos, sino mas bién en su significación en el entorno de un lenguaje de ensamble.

III-5-3-1.- Records:

A diferencia de algunos lenguajes de alto nivel, el RECORD aquí se define como un "conjunto de bits agrupados según un modelo o plantilla que definimos". La longitud del RECORD puede ser de 8 o 16 bits, no estando obligado a utilizar la totalidad de dichos bits.

Este modelo ha de ser "declarado", identificando los bits y los campos del RECORD. Cada campo es un subconjunto de bits consecutivos. El total de los bits de todos los campos no puede superar, obviamente, los 8 (o 16) bits. El formato de la declaración es:

nombre RECORD nombre_campo : n_bits=valor (,.....).

Si el número de bits total de todos los campos es inferior a 9 el RECORD será de tipo byte, sino de tipo palabra.

III-5-3-2.- Estructuras:

Son variables compuestas, en las que se puede definir un modelo general de datos utilizando los tipos de datos predefinidos. Este tipo de datos, si es mas similar a aquel definido en los lenguajes de alto nivel (PL/M, PASCAL). Es aï igual que el RECORD un modelo, sobre el que se plasmarán posteriormente las variables. La declaración de la estructura no produce reserva alguna de memoria.

El formato de la declaración:

```
nombre_estructura     STRUC
      nombre_campo   (DB, DW, DD)  expresión
                          :
                          :
                          :
nombre-estructura     ENDS
```

Por ejemplo:

```
HARDWARE     STRUC
    CLASE    DB    10 DUP (?)
    REF      DW    0
    POWER    DB    0
HARDWARE     ENDS
```

Para crear una variable del tipo declarado se utiliza la sentencia:

nombre nombre_estructura <expresión>

Por ejemplo:

FIRST HARDWARE <>

reserva espacio para una estructura y lo inicializa con los valores del modelo.

CAJA HARDWARE 7 DUP (<,5,>)

reserva espacio para 7 estructuras cambiando el valor del campo REF, por el valor 5.

III-5-3-3.- Formas de Referenciar los Datos:

Hay distintas formas de acceder a los datos de un array o de una estructura. Los modos de direccionamiento a través de los registros base (BX y BP) y los registros índice (SI y DI), ofrecen enormes posibilidades.

- Si ALFA es un array y SI contiene el valor 4, entonces ALFA [SI] es el quinto elemento del array. ALFA [SI+2] selecciona el séptimo elemento. Téngase en cuenta que ALFA [SI+2] =ALFA +2 [SI].

- Si ALFA es de tipo estructura y BETA un campo de dicha estructura, ALFA.BETA, selecciona el campo BETA de la estructura ALFA.

- Si el registro BX contiene la dirección base de una estructura y BETA es un campo de la estructura entonces [BX].BETA se refiere al campo beta de la estructura.

- Si el registro BX contiene la dirección de un array,entonces [BX][SI] corresponde al elemento SI del array.

- Si BX contiene la dirección base de una estructura, de la cual el campo ALFA es un array, entonces [BX].ALFA[SI] selecciona el elemento SI del array dentro de la estructura.

- Si BX es la base de una estructura en la cual el campo ALFA es a su vez otra estructura. La variable [BX].ALFA.BETA se refiere al campo BETA de la estructura ALFA.

- Si BX contiene la base de una estructura en la que el campo ALFA es un array y cada elemento es una estructura, entonces [BX].ALFA[SI+3].BETA se refiere al campo BETA del elemento SI + 3 del array ALFA.

III-5-4.- Segmentos de Pila:

En un segmento de pila hay que definir la longitud de la pila que se va a manejar. La forma de llevar a cabo esto es la siguiente:

```
etiqueta    SEGMENT    (AT dirección)
            DW         n_posiciones DUP (?)
nombre      LABEL      WORD
etiqueta    ENDS
```

Por ejemplo:

```
PILA        SEGMENT    AT 100H
            DW         20 DUP (?)
TOPE        LABEL      WORD
PILA        ENDS
```

Se utilizará la etiqueta TOPE para cargar al SP, ya que este puntero siempre se decrementa al introducir los valores en la pila.

III-5-5.- Procedimientos:

Como ya se comentó en apartados anteriores, las subrutinas o procedimientos en este lenguaje de ensamble, pueden ser de dos tipos, dependiendo de si la transferencia de control se llevaba a cabo dentro o fuera del segmento, con alteración del registro CS.

Por ello el programa ensamblador, necesita conocer este dato y de ahi la necesidad de identificar y catalogar las subrutinas.

En este sentido existen dos pseudoinstrucciones que permiten definir un procedimiento: PROC y ENDP. El formato es el siguiente:

```
nombre    PROC    [NEAR/FAR]
            :
            :
       (cuerpo proced)
            :
            :
           RET
            :
nombre    ENDP
```

donde NEAR indica que el procedimiento será llamado desde el propio segmento y FAR que será llamado desde otro segmento. Dependiendo de este dato, el programa ensamblador generará el correspondiente código para la instrucción RET.

III-5-6.- Directivas de Montaje:

Cuando el programa haya de montarse (linkarse) con otros módulos, será necesario el establecimiento de un mecanismo para resolver referencias externas al módulo.

Esto se logra mediante las directivas EXTRN y PUBLIC.

La pseudoinstrucción EXTRN sirve para indicar que el identificador al que se hace referencia no se encuentra en el módulo presente y ha de esperarse a la fase de montaje para resolver la incógnita.

Su formato es el siguiente:

```
EXTRN    nombre:  tipo , ...
```

donde el tipo puede ser:

- BYTE, WORD, DWORD usados con las referencias a variables.
- NEAR, FAR utilizado con subrutinas y saltos.

En cuanto a la directiva PUBLIC su utilización surge por la necesidad de indicar cuales de los identificadores van a poder ser referenciados por otros módulos.

Su formato es el siguiente:

PUBLIC símbolo, ...

Todos los símbolos que aparecen en la lista han de estar definidos en el módulo.

TEMA IV:

"DIAGRAMA FUNCIONAL DE BLOQUES DEL SDK-86"

IV) DIAGRAMA FUNCIONAL DE BLOQUES DEL SDK-86.

IV-1) INTRODUCCION.

El SDK-86, es un Kit diseñado por Intel, que constituye un sistema microcomputador completo, basado en el 8086. Dispone de un teclado elemental, con display, controlado por un sistema operativo residente, que permite la carga y depuración de programas, así como la visualización y modificación de posiciones de memoria y registros internos de la CPU. El reducido teclado implica la introducción de las instrucciones en código hexadecimal.

El SDK-86, puede conectarse directamente con un teletipo o un terminal CRT. Dispone también de una zona libre, dedicada a la posible implementación del hardware auxiliar que precisen las aplicaciones de los usuarios.

Las especificaciones generales del SDK-86 son:

- CPU: 8086, con una frecuencia de reloj de 2, 5 ó 5MHz. Seleccionable con puente eléctrico. Tiempo del ciclo de instrucción: 800 nsg (5 MHz),

- Memoria: ROM: 8 Kbytes (2316/2716).

Direcciones:

FE000H-FFFFFH.

RAM: 2 Kbytes, ampliables a 4 kbytes (2142).

Direcciones:

0-7FFH (0-FFFH con 4 kbytes).

- Entradas/Salidas:

Paralelo: 48 líneas (con dos 8255A).

Serie: RS232 o bucle de corriente (8251A).

Relación de Baudios seleccionables desde 110 a 4800 Baudios.

- Señal de interface:

Bus CPU: Todas las señales compatibles con TTL.

Paralelo E/S: Todas las señales compatibles con TTL.

Serie: 20 mA de corriente de bucle o RS232.

- Interrupciones:

Externas: Enmascarables y no enmascarables.

El monitor del SDK-86 reserva el vector de interrupción 2, para la NMI.

Internas: El monitor reserva el vector de interrupción 1 para el paso a paso (Single-Step) y el 3 para Breakpoint.

- DMA:

  Solicitud de HOLD: seleccionable con puente eléctrico, entrada compatible con TTL.

- Software:

  Sistema monitor: preprogramable 2316 ó 2716 ROM, direcciones FE000H-FFFFFH.

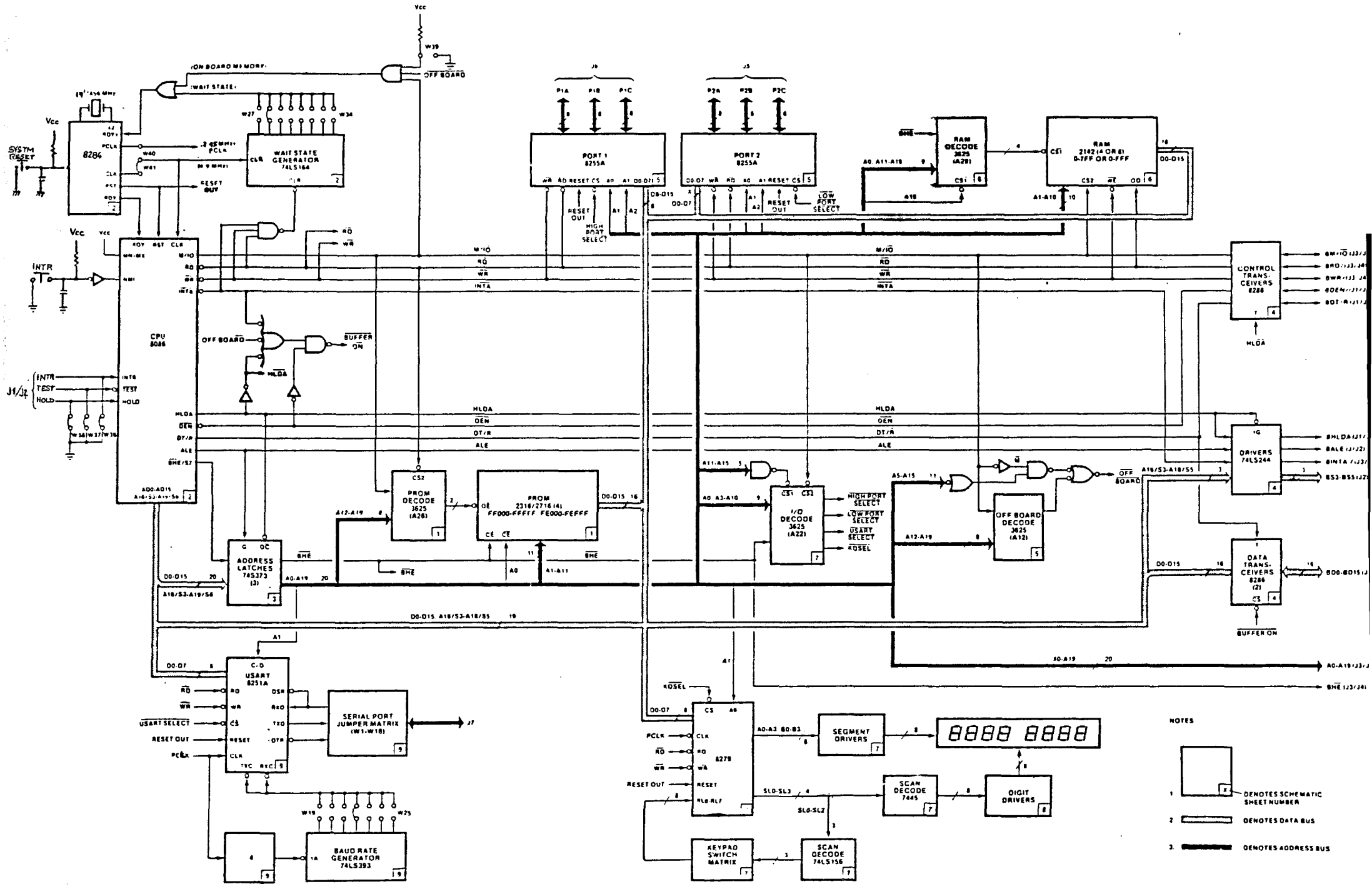
  Monitor E/S: teclado y serie (teletipo o CRT).

-Alimentación:

  Vcc: +5 v(±5%), 3,5 A.

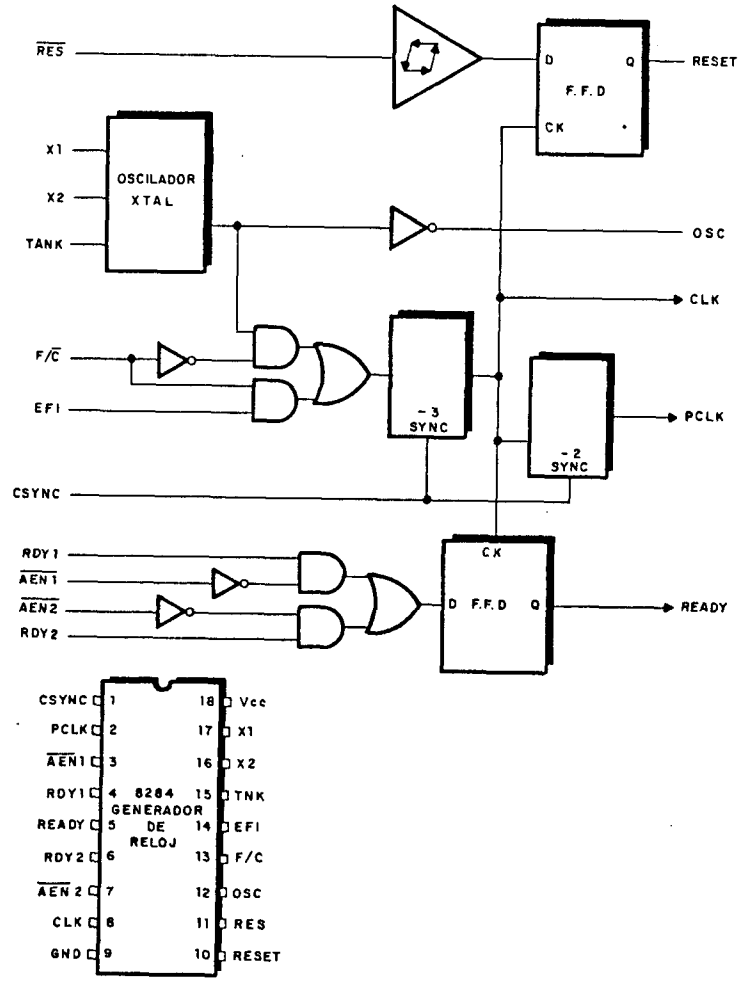
  VTTY: terminal o teletipo en la puerta serie: -12 v (±10%), 0,3 A.

A continuación se muestra el diagrama de bloques del SDK-86:



IV-2) GENERADOR DE SEÑALES DE RELOJ (8284).

Estructura interna:



Cualquier sistema basado en el 8086 requiere una lógica adicional encargada de generar las señales de sincronización para todo el sistema. El generador de señales de reloj 8284 de Intel, junto con el cristal oscilador (externo), está diseñado para esta tarea.

El 8284, puede utilizarse para generar las señales de reloj para el 8086, 8088, 8087, 8089 y sus periféricos. Los pulsos de reloj determinan la velocidad de funcionamiento del sistema.

La frecuencia máxima especificada para el 8284 es de 5 MHz, esto es, 200 nsg por ciclo.

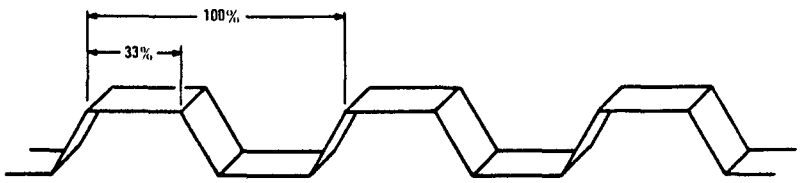
Para el correcto funcionamiento del 8284, se necesita la señal estabilizada procedente de un cristal de cuarzo o de un elemento exterior. Según el cristal o la señal externa que se emplee, se puede alcanzar una frecuencia de trabajo cercana a los 8 MHz, esto es, 125 nsg por ciclo.

Tanto los chip standar, como las versiones especiales más rápidas, admiten teóricamente una velocidad mínima de 2 MHz, aunque en la práctica funcionan incluso a frecuencias menores.

Si se usa una fuente externa, la velocidad puede variar desde un ciclo cada vez (control manual) hasta unos 8MHz.

Para conseguir un rendimiento óptimo de los procesadores, los pulsos de reloj generados por el 8284 se mantienen a tensión alta durante un 33 por 100 del periodo, es decir, una tercera parte del tiempo total de ciclo. Esto significa que el reloj

está activo una tercera parte del tiempo y desactivo las dos terceras partes del tiempo:



Como la frecuencia de la fuente es triple que la de salida del 8284, es relativamente fácil generar este tipo de onda.

El 8284 proporciona tres señales a la CPU:

- RESET:

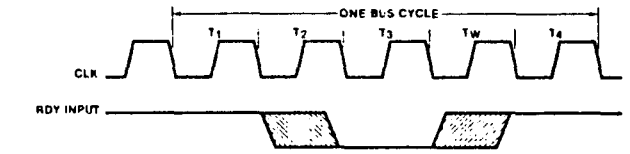
  Su misión es la de reinicializar los valores de la computadora. Esta señal es imprescindible en los casos en que un programa se mete en un bucle infinito, o que un ruido de alimentación afecte a partes del programa en curso.

- READY:

Su misión 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 acceder a un dispositivo que no está preparado para la transferencia, el dispositivo envía un "0" por la línea Ready. Cuando el procesador recibe esta señal, entra en un ciclo de espera hasta que aparece un "1" por dicha línea. Sólo entonces continúa el programa.

La salida READY, la cual está sincronizada a la señal de CLK es acoplada directamente a la entrada READY de la CPU. Como muestra la siguiente figura, cuando el dispositivo direccionado necesita insertar uno o más estados de espera en un ciclo de bus, desactiva la entrada READY antes del final del estado T2 lo cual causa que la salida READY sea desactivada al final del estado T2. El estado de espera resultante (TW) es insertado entre los estados T3 y T4. A la salida del

estado de espera, el dispositivo activa la entrada RDY del 8284 la cual causa que la entrada READY de la CPU sea activada al final del actual estado de espera y permite a la CPU entrar el estado T4.



- PCLK:

  Reloj periférico, que funciona a una frecuencia mitad que CLK, con un ciclo de trabajo al 50 por 100 (un 50 por 100 a nivel alto de tensión y un 50 por 100 a nivel bajo,cero). Está previsto para sincronizar aquella lógica que requiera esta forma de temporización.

En el SDK-86, el cristal de entrada es de 14.7456 MHz (seleccionado como un múltiplo de la velocidad de transmisión del reloj, para proporcionar una frecuencia conveniente a la CPU). El 8284, divide esta frecuencia por tres para producir los 4.9 MHz de la señal de reloj requerida por la CPU. Además, el 8284 realiza una división por dos para la salida PCLK, la cual es la señal principal usada por el resto de los circuitos.

Las dos señales de control de salida Ready y reset, están sincronizadas a los 4.9 MHz de la señal de reloj.

La señal Reset se activa a nivel bajo.

La señal Ready se activa cuando el RDY1 de entrada procedente del generador de estado de espera está activado. El RDY1 de entrada se activa cuando esté direccionada la memoria de la placa o cuando esté seleccionado el número de estados de espera.

El eslabón W4Ø, está conectado en la placa para operación de baja velocidad (2.45 MHz) y permite que la señal PCLK sea dirigida a la entrada del CLK de la CPU en lugar del CLK de salida desde el 8284. Nótese que esta es la configuración que tiene el Kit una vez ensamblado, la cual permite que la memoria de la placa y las operaciones E/S puedan

ser ejecutadas sin la necesidad de estados de espera.

Cuando el 8086 debe operar a la velocidad de 5 (4.9) MHz, basta con quitar el puente W40 y ponerlo en el puente W41.

El SDK-86, puede ser alimentado con una CPU que tenga una frecuencia máxima de reloj de entrada de 4 MHz y, por lo tanto, sólo puede operar con el reloj de 2.45 MHz (con el eslabón en W40).

IV-3) GENERADOR DE ESTADOS DE ESPERA (74LS164).

Este circuito permite insertar unos estados de espera dentro de los ciclos de la CPU, para compensar la lentitud de los periféricos E/S o del circuito de memoria o permite a la memoria de la placa y a las operaciones de E/S una correcta función cuando la CPU está operando a la velocidad de 5 MHz. La memoria del SDK-86 y los dispositivos de E/S no requieren estados de espera cuando el reloj es usado a 2.45 MHz, y requieren un estado de espera cuando es usado a 4.9 MHz.

Cuando uno o más estados de espera son requeridos para la correcta operación de la CPU, el puente instalado originalmente en el W27 (estado de espera

cero), es quitado y se pone en el sitio necesario, según los estados de espera correspondientes a ese ciclo, de acuerdo con la tabla siguiente:

POSICION DE LA CLAVIJA	ESTADO DE ESPERA
W27	0
W28	1
W29	2
W30	3
W31	4
W32	5
W33	6
W34	7

Si no se coloca ninguna clavija, es como si estuviera puesta en W27, es decir, no habria ningún estado de espera.

El 74LS164, es borrado (CLR: puesta a cero a nivel bajo), cuando están desactivados los ciclos de lectura (RD), escritura (WR) e interrupción (INTA), y está permitido cuando está activado uno, dos o tres ciclos (WR, RD o INTA), que es cuando el Clear del generador de estados de espera está desactivado. (ver esquema general).

Cuando está permitido el generador de estados de espera, empieza a desplazar un uno a través del registro.

Cuando se selecciona el número de desplazamientos (Ø-7), se activa el puente eléctrico de salida, el cual activa el RDY1 (de entrada) del generador de señales de reloj, lo cual hace que el RDY del generador de señales de reloj active el RDY de la CPU.

Cuando un estado de espera está seleccionado, las operaciones de E/S sobre la placa, están sujetas al número de estados de espera seleccionados, mientras que las operaciones de memoria sobre las placas son ejecutadas sin restricción de estado de espera.

Como se ve en el diagrama de bloques, la señal RDY1 de entrada al 8284, es activada por:

- Los estados de espera.
- O por las señales que llegan a la puerta lógica AND (Vcc, OFF BOARD y M/IO de la CPU).

La salida de la puerta es activada durante la lectura de memoria sobre la placa y las operaciones de escritura (M/IO y OFF BOARD a nivel alto). EL eslabón W39 cuando se une a la entrada de la puerta AND, inhabilita permanentemente la puerta y motiva que las operaciones de memoria sobre la

placa estén sujetas al número de estados de espera seleccionados.

IV-4) CPU 8086.

Su estudio se vió en el tema anterior. Para ver sus conexiones en la placa, ir al diagrama de bloques mostrado al principio del tema.

Hay que señalar, que las entradas INTR, HOLD y TEST, están inhabilitadas por los puentes W36, W38 y W37, y cuando un circuito periférico está interconexionado al bus de expansión y necesita usar una de estas señales, debe quitarse el eslabón correspondiente.

IV-5) PUERTOS PARALELOS DE E/S (8255A).

El controlador programable paralelo de interfaz 8255A (PPIC), sirve de ayuda en la conexión a la computadora de dispositivos que envían bytes completos cada vez (o incluso palabras de 12, 16 ó 24 bits).

La transmisión paralelo es útil en todas aquellas aplicaciones que requieran unas transmisiones a gran velocidad, y utilicen dispositivos no demasiado alejados del computador central. No hay ninguna sincronización especial en las transmisiones paralelo. Los bytes se envían tan rápidamente, tan lentamente, como permite el software. La velocidad máxima de transmisión viene limitada por la rapidez con que el sistema pueda sacar los datos. La mayor velocidad se obtiene utilizando el DMA (8237).

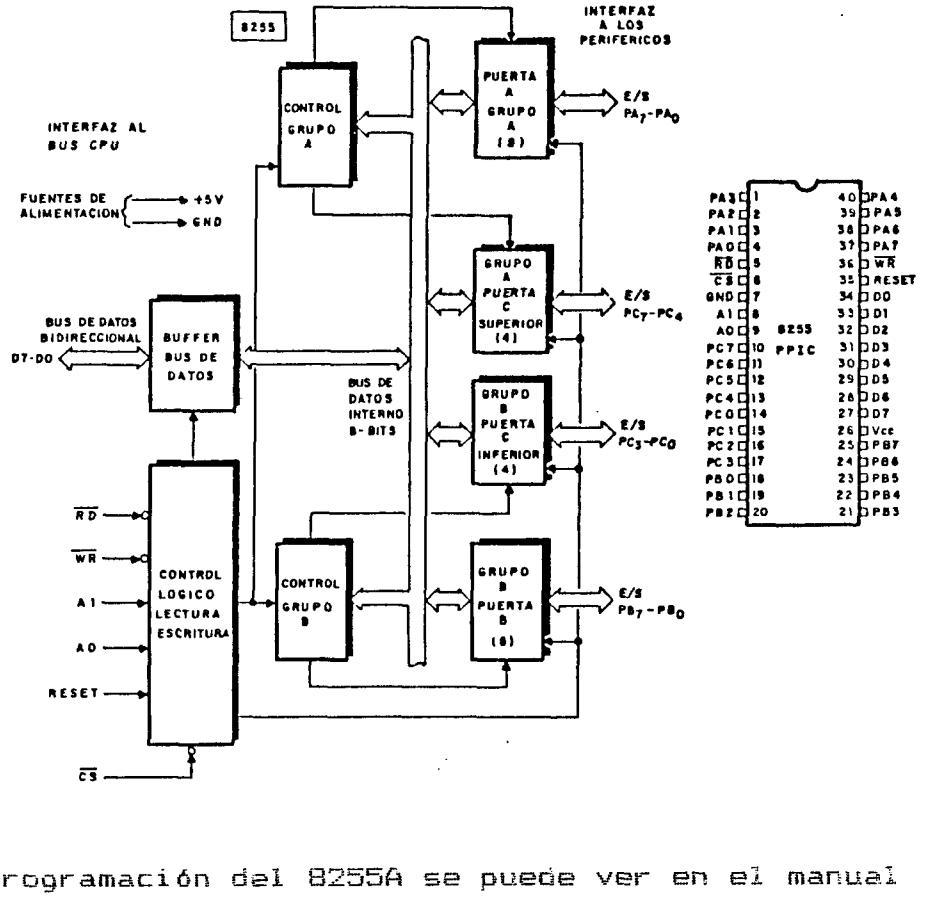
El SDK-86 tiene dos chip 8255A. Cada uno de estos circuitos contienen tres puertos de datos de 8 bits cada uno, (A, B, C), y un puerto de control de sólo lectura. Los bytes de datos de mayor orden están designados al puerto 1 (D8-D15) y los de menor orden al puerto 2 (D0-D7). Todos los puertos pueden ser direccionados individualmente (ej. puerto P1A, P2C, control P1,...), o por el par de puertos correspondientes (P1A y P2A, P1B y P2B, P1C y P2C) pueden ser simultáneamente direccionados formando un único puerto de datos de 16 bit.

Las direcciones asignada al puerto de E/S para los dos circuitos paralelos de puerto de E/S, están definidos en la siguiente tabla:

PUERTO	DIRECCION	PUERTO	DIRECCION
P2A	FFF8H	P1A	FFF9H
P2B	FFFAH	P1B	FFFBH
P2C	FFFCH	P1C	FFFDH
P2 control	FFFEH	P1 control	FFFFH

Durante las operaciones de los bytes, el decodificador de E/S activa el CS de entrada conveniente (es decir, activa el chip de los bytes de mayor peso o el de menor peso), mientras que durante las operaciones de palabras, la dirección del puerto 2 es usada para la dirección de los puertos pares deseados, y el decodificador de E/S genera ambas HIGH PORT SELECT y LOW PORT SELECT coincidentemente.

La estructura interna del 8255A es:



la programación del 8255A se puede ver en el manual de componentes de Intel.

IV-6) MEMORIA RAM (2142).

El SDK-86, incluye 2 Kbytes de memoria RAM, y la placa está prevista para la instalación 2 Kbytes más.

La memoria está en el comienzo de la memoria, de 0H a 07FFH ó de 0H a 0FFFH.

La disposición del direccionamiento de memoria de la CPU 8086 permite la lectura o escritura simultánea de los 16 bits (localizados los dos byte adyacentes) o la lectura o escritura de uno u otro bit alto (D8-D15) o bajo (D0-D7).

La RAM es permitida por la salida del decodificador de RAM 3625 (A29) y por la señal M/IO (de la CPU) a nivel alto, control de línea (operación de memoria). El decodificador de RAM 3625 (A29), es permitido por el estado bajo el bit de dirección A19 (indicando una dirección debajo de 80000H) en la entrada CS1.

El 3625 (A29), decodifica el BHE y los bits de dirección A0 y A11 a A18. Cuando los bits de dirección están inactivos (bits A12 a A18), está indicada una dirección por debajo de 1000H. El bit de dirección A11 determina si la dirección está entre 0H y 07FFH (A11 inactivo) o entre 0800H y 0FFFH (A11 activo). El bit de dirección A0 y el BHE determinan el byte o bytes permitidos. La siguiente tabla define la decodificación de la RAM:

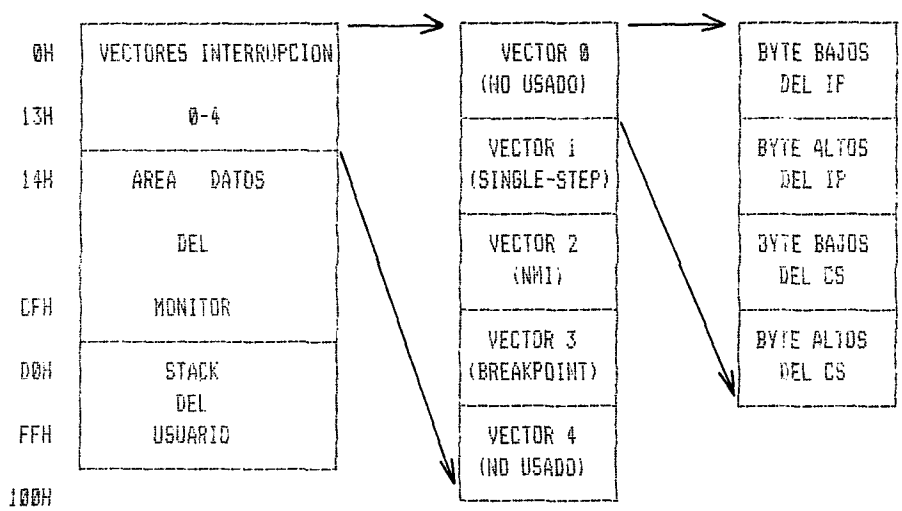
ENTRADAS DECOD.RAM				SALIDAS DEC				BYTE(S) SELECCION.
A12-A18	A11	BHE	A0	04	03	02	01	(DIRECCIONBLOQUES)
0	0	0	0	1	1	0	0	Ambos bytes(0-07FF)
0	0	0	1	1	1	0	1	Byte alto (0-07FF)
0	0	1	0	1	1	1	0	Byte bajo (0-07FF)
0	1	0	0	0	0	1	1	Amb.byte(0800-0FFF)
0	1	0	1	0	1	1	1	Bytes altos "
0	1	1	0	1	0	1	1	Bytes bajos "
Todos los otros estados.				1	1	1	1	Ninguno

La salida del decodificador de RAM, permite los correspondientes byte de RAM.

Así mismo, la RAM es direccionada por los bits de dirección A1 a A10 y es controlada por las señales RD y WR (lectura y escritura).

Los primeros 256 (en decimal) bytes de RAM de (0H a 0FFH) están reservados para el programa monitor.

La siguiente ilustración nos muestra la actual localización del monitor en memoria (RAM):



IV-7) MEMORIA PROM (2316 ó 2716).

El SDK-86 incluye 8 kbytes de memoria PROM; 4 kbytes (en dos de 2Kx8, ROM 2316 ó EPROM 2716) contiene el programa monitor Keypad y 4 Kbytes (en otra de 2Kx8 ROM o EPROM) contiene el programa monitor serie. Los 8 Kbytes de PROM están localizados en el final de la memoria FF000H a FFFFFH). El sistema principal del programa monitor (keypad o serie) está en FF000H a FFFFFH (zócalo A27 y A30). El resto de los 4K están en los zócalos A36 y A37 (FE000H a FEFFFH) pueden ser usados por otro programa monitor o para almacenar un programa en ROM o PROM. (Ver diagrama general). La PROM es activada por la salida correspondiente del decodificador de PROM 3625 (A26), por el BHE y por el bit de dirección de entrada A0. Al decodificador de PROM le entra la señal M/IO, y los bits de dirección A12-A19 determinan cual de los 4Kbytes (FE000H a FEFFFH ó FF000H a FFFFFH) van a ser direccionados, mientras que el BHE y la entrada A0 permite que uno u otro bit de orden alto o bajo o palabra (ambos bytes) salgan.

Como se ve en los esquemas del SDK-86, hay un espacio para dos PROM adicionales. Las salidas (CSX y CSY) van al bus de expansión J2 y permiten 2 Kbytes de memoria ROM o PROM adicionales (cuyas direcciones son FC000 a FCFFFH y FD000H a FDFFFH), que pueden ser incorporadas sin requerir decodificar direcciones externas.

Tabla de direcciones de PROM decodificadas:

ENTRADAS DECODIF. DE PROM				SALIDAS DEC. DE PROM				PROM DIRECCION SELECCION BLOQUES
M/IO	A14-A19	A13	A12	04	03	02	01	
1	1	1	1	1	1	1	0	FF000-FFFFFH
1	1	1	0	1	1	0	1	FE000-FEFFFH
1	1	0	1	1	0	1	1	FD000-FDFFFH(CSX)
1	1	0	0	0	1	1	1	FC000-FCFFFH(CSY)
Todos los estados.		otros		1	1	1	1	NIMGUND

Las salidas del decodificador de PROM, permiten las correspondientes direcciones en la memoria PROM.

IV-8) DECODIFICADOR DE E/S, 3625 (A22).

Es activado por un nivel bajo de M/IO (indicando una operación de E/S) y una dirección entre FE00H y FFFFH (los bits A11 a A15 deben estar activados, es decir, que al ser una puerta NAND, sólo activará el decodificador de E/S cuando sean todos uno). Cuando es activado, decodifica el BHE y los bits de direcciones A0, A3 a A10, y genera la correspondiente señal (o señales) de selección de puerto de E/S de acuerdo con la siguiente tabla:

ENTRADAS DECOD. PROM					SALIDAS DECODIFICADOR PROM			
					HIGH PORT	LOW PORT	USART	KDSEL
A5-A10	A4	A3	BHE	A0	SEL.(04)	SEL.(03)	SEL(02)	(01)
1	0	1	0	0	1	1	1	0
1	0	1	1	0	1	1	1	0
1	1	0	0	0	1	1	0	1
1	1	0	1	0	1	1	0	1
1	1	1	0	0	0	0	1	1
1	1	1	0	1	0	1	1	1
1	1	1	1	0	1	0	1	1
					1	1	1	1

Las salidas del decodificador de PROM seleccionan el dispositivo de E/S.

La siguiente tabla define las direcciones individuales asignadas a los puertos de E/S:

DIRECCION PUERTO	FUNCION DEL PUERTO
0000	
a	ABIERTO
FFDF	
FFE8	R/W 8279 DISPLAY RAM
	O LECTURA 8279 FIFO
FFE9	
FFEA	LECTURA ESTADO 8279
	O ESCRITURA COMANDO 8279
FFEB	
FFEC	RESERVADO
FFED	
FFEE	RESERVADO
FFEF	
FFF0	R/W DATO 8251A
FFF1	
FFF2	LECTURA ESTADO 8251
	O ESCRITURA CONTROL 8251
FFF3	
FFF4	RESERVADO
FFF5	
FFF6	RESERVADO
FFF7	

DIRECCION PUERTO	FUNCION DEL PUERTO
FFF8	R/W PUERTO P2A 8255A
FFF9	R/W PUERTO P1A 8255A
FFFA	R/W PUERTO P2B 8255A
FFFB	R/W PUERTO P1B 8255A
FFFC	R/W PUERTO P2C 8255A
FFFD	R/W PUERTO P1C 8255A
FFFE	ESCRITURA P2 CONTROL 8255A
FFFF	ESCRITURA P1 CONTROL 8255A

IV-9) DECODIFICADOR OFF-BOARD, 3625 (A12).

El decodificador lógico OFF-BOARD es el responsable de la generación de la señal OFF-BOARD.

Esta señal es usada por el generador de estados de espera a impulsar a operaciones de memoria Off-board sujetas al número de estados de espera seleccionado, y también es responsable de la generación de la señal BUFFER ON, que es usada para permitir la transmisión de datos sobre el bus de expansión, durante ambas operaciones de E/S y acceso a memoria Off-board.

El decodificador Off-board PROM (A12) acepta los bits de dirección del A12 al A19, y la señal M/IO genera la señal OFF-BOARD cuando la memoria Off-board localiza las direcciones. La siguiente tabla nos define la decodificación de la PROM. Nótese que la tabla muestra los estados inactivos en los cuales la PROM no genera la señal OFF-BOARD:

ENTRADAS PROM									SALIDAS	DIRECCIONES DEL BLOQUE
M/IO	A19	A18	A17	A16	A15	A14	A13	A12	PROM(01)	CORRESPOND.
1	0	0	0	0	0	0	0	0	1 inact.	0-0FFFH (ON BOARD RAM
1	1	1	1	1	1	1	0	1	1 inact.	FE000-FEFFF (ON BOARD PRO
1	1	1	1	1	1	1	1	1	1 inact.	FF000-FFFFF (ON BOARD PRO
todos los otros estados									0 activa	01000-FDFFF (0FF-BOARD)

La señal OFF-BOARD también es generada cuando un puerto Off-board E/S es direccionado (dirección del puerto de 0H a 0FFDFH). Durante una operación de E/S, la puerta que se muestra encima de la PROM, en el diagrama de bloques, es permitida para

un estado bajo de M/IO y genera la señal OFF-BOARD si cualquiera de los bits de dirección del A5 al A15 están a nivel bajo (una dirección debajo de 0 FFE0H).

IV-10) KEYPAD/DISPLAY.

El teclado/display lógico es controlado por el controlador de teclado/display INTEL 8279. Es el responsable para el Debouncing (efecto de rebote) de las teclas, la codificación de la matriz de teclado y el refresco de los elementos del display. Como vimos en el apartado IV-8, el 8279 ocupa dos puertos de E/S dentro del espacio de direcciones del On-board E/S (es decir que ocupa dos puertos de E/S dentro de la placa), puesto que el 8279 es interface para los byte de orden bajo del bus de datos (D0-D7). Ambos puertos tienen un número determinado de dirección de puerto (FFE8H y FFEAH). La función del puerto individual 8279 está determinada por el bit de dirección A1 y la señal RD y WR, como se vé en la siguiente tabla:

PUERTOS E/S KEYPAD/DISPLAY:

ENTRADA 8279			DIRECCION	
A1	RD	WR	PUERTO	FUNCION DEL PUERTO
0	0	1	FFE8H	LECTURA DISPLAY RAM O KEYBOARD FIF0
0	1	0	FFE8H	ESCRITURA DISPLAY RAM
1	0	1	FFEAH	LECTURA ESTADO
1	1	0	FFEAH	ESCRITURA COMANDO

Puesto que el bit de direccion A2 no es decodificado por el decodificador de E/S PROM, las direcciones de puerto FFECH y FFEEH son decodificadas como direcciones del puerto FFE8H y FFEAH, respectivamente. (Ver como en la tabla del apartado IV-8, estas dos direcciones adicionales FFECH y FFEEH, están reservadas).

El programa monitor a través del comando de puerto de escritura, pone en orden el 8279 como sigue:

- 8 digitos, 8 bit, entran por la izquierda al display y exploran el teclado codificado con 2 teclas de bloqueo (poner el modo de comando teclado/display; valor byte 00H).

- Un factor de preescala de 25 proporciona 5nsg de tiempo de exploración del teclado y display, y 10nsg de tiempo debounce (efecto de rebote), (comando de programa de CLOCK;valor Byte 039H).

Viendo el diagrama de bloques, puesto que el modo de exploración del decodificacador de teclado está especificado de la salida SL0 a SL3, representa un contador binario.

El 7445 (un decodificador BCD a 10 líneas) convierte al contador binario a 8, línea única de salida (dos salidas no son usadas). Las cuales son usadas para permitir los elementos del display individualmente. Las salidas SL0 a SL2 son llevadas al 74LS156 (decodificador exploradors configurado como un decodificador de 3 a 8 líneas) proporciona tres pilas de exploración de señales de entrada a la matríz de conmutación de teclas.

Las salidas desde la matríz de conmutación van a las entradas del 8279, de la RL0 a la RL7, representan las 8 columnas de los conmutadores (como se vé en la hoja 7 de los esquemas). Cuando un conmutador es presionado, mientras una fila está siendo explorada, la columna correspondiente es permitida. El 8279 usa un valor de un bit de columna como permiso, y el valor de la fila es

explorado (SL0-SL2) y genera un código de 6 bits representando a la tecla presionada. Este es el código que es almacenado en el FIFO y accede a la CPU a través de la lectura del puerto de E/S del teclado FIFO.

Las salidas A0 a A3 y B0 a B3 del 8279 forman una única salida en paralelo de 8 bit, conteniendo los bits que permiten el segmento individual. La siguiente tabla contiene la correlación entre los bits individuales y los elementos del segmento del display (un nivel alto de salida enciende el correspondiente segmento del display).

Definición de segmentos:

SEGMENTO DISPLAY	BIT SALIDA 8279	SEGMENTO PERMITIDO	BIT SALIDA 8279	SEGMENTO PERMITIDO
a	A0	e	B0	a
f| |b	A1	f	B1	b
g	A2	g	B2	c
e| |c	A3	DP	B3	d
d .DP				

IV-11) INTERFACE EN SERIE : USART (8251A).

El interface en serie, es un INTEL-8251A (USART), que es un interface programable de comunicación (universal transmisor/receptor síncrono-asíncrono), el cual opera en modo asíncrono. Tiene asociado un generador de velocidad de transmisión, seleccionable mediante puentes (W19-W25) que van desde 75 a 4800 Bandios, para ser usados por la USART.

La matriz de puentes de puerto en serie es usado para configurar las señales de entrada/salida de la USART, para conectar al interface en serie en J7 de modo stand-alone (modo independiente) o "MDS SLAVE" operando con uno u otro TTY (bucle de corriente 20 mA), o CRT terminal (RS232) interface capacitado.

El programa monitor escribiendo en modo comando de 0CFH al puerto de control de la USART. La USART está configurada para:

- 8 bits de caracteres de longitud.
- inhabilitado paridad.
- 2 stop bits
- velocidad de transmisión por un factor de 64x.

El generador de velocidad de transmisión (74LS393 dual contador binario de 4 bit), usa la señal 614.4KHZ (PCLK/4), proporciona una serie de frecuencias de velocidades de transmisión. Cuando se usa la USART en el modo de operación de 64x, estas frecuencias son 64 veces las correspondientes velocidades de transmisión. La siguiente tabla define la selección de velocidad de transmisión de la frecuencia del generador de salida.

VELOCIDAD DE TRANSMISION	POSICION DEL PUENTE	FRECUENCIA DE SALIDA
4800	W25	307.2 KHZ
2400	W24	153.6 KHZ
1200	W23	76.8 KHZ
600	W22	38.4 KHZ
300	W21	19.2 KHZ
150	W20	9.6 KHZ
75	W19	4.8 KHZ
110	W20,W26	6.98 KHZ

En el esquema del SDK-86 vemos que los 110 Baudios de velocidad de transmisión, están derivados de la señal 76.8KHZ a la salida 2A del segundo contador binario. Cuando el puente se pone en W26, a la puerta NAND van las salidas 20A, 20B, 20D y ponen a cero el segundo contador binario, y sobre la cuenta once proporciona la señal requerida de 6.98 KHZ en la salida 20C.

El USART ocupa dos puertos E/S dentro de la placa en el espacio de las dirección de E/S y, puesto que el USART es interfaced para los byte de menor orden del Bus de datos (D0-D7), ambos puertos tienen un número detrminado de direcciones de puerto (FFF0H y FFF2H). La función del puerto individual USART está determinada por el bit de dirección A1 y la señal RD y WR como se ve en la siguiente tabla:

USART PUERTOS E/S:

ENTRADA USART			DIRECCIONES	FUNCION
A1	RD	WR	PUERTO	PUERTO
0	0	1	FFF0H	LECTURA DATO USART
0	1	0	FFF0H	ESCRITURA DATO USART
1	0	1	FFF2H	LECTURA ESTADO USART
1	1	0	FFF2H	ESCRITURA CONTROL USART

Puesto que el bit de dirección A2 no es decodificado por el decodificador de E/S PROM, las direcciones del puerto FFF4H y FFF6H son decodificadas como direcciones de puerto FFF0H y FFF2H respectivamente.

En la tabla del apartado III-8, vemos que estas dos direcciones adicionales están reservadas.

La matriz de puentes del puerto en serie, determina los pins individuales asignados de la señal de entrada/salida de la USART, hacia el conector J7 y la definición de la señal (corriente de lazo ó RS232) y polaridad. La siguiente tabla define los puentes que deben instalarse para las varias configuraciones de interface.

CONFIGURACION DE INTERFACE	PUENTES	PC BOARD SILKSCREEN.
STAND ALONE CRT TERMINAL	W1-W5	CRT
STAND ALONE TELETYPE WRITE	W8-W16	TTY
INTELLEC SLAVE CRT TERMINAL	W3-W7	MDS-CRT
INTELLEC SLAVE TELETYPE WRITER	W14-W18	MDS-TTY

IV-12) BUS EXPANSION.

El Bus de expansión lógica permite a los circuitos periféricos ser interconectados al SDK-86. El Bus de expansión lógica, el cual consiste en el control del transceptor y excitador, el transceptor de datos y el latch de dirección, proporciona todas las direcciones requeridas, datos y señales de control para una u otra tarjeta (usa el area designada), o circuitos periféricos interconectados fuera de la tarjeta (OFF-BOARD), conectados al Bus de expansión en los conectores J1/J2 y J3/J4.

El control transceptor (un intel 8286 bus bidireccional excitador), determina la fuente de las cinco señales de control bidimensional.

Cuando a un dispositivo periférico "master" se le permite el control del BUS por la CPU 8086, la señal HLDA transmitida (T) a la entrada del transceptor se activa y motiva la función del transceptor como receptor (el origen de las señales de control es desde el circuito periférico).

Recíprocamente, mientras que el 8086 tiene el control del BUS, la señal HLDA está desactivada, y las señales de control son originadas desde la CPU 8086.

El (control) excitador (un 74LS244 latch tri-state) permite asociar a la CPU 8086 el control y señales de estado sobre el BUS de expansión, mientras que el 8086 tiene el control del BUS (HLDA desactivada) y, cuando se permite el control al circuito periférico (HLDA activada), el excitador (drivers) está en posición de estado de alta impedancia.

El transceptor de datos (son dos INTEL 8286 BUS bidireccional excitador), determina la fuente de los bytes de alto y bajo orden durante la memoria fuera de la tarjeta (OFF-BOARD), y operaciones de E/S y reconocimiento de ciclos de interrupción.

Refiriéndonos al diagrama de bloques, el transceptor es permitido por la señal BUFFER-ON. Esta señal se activa durante los ciclos de instrucción T2 a T4 (DEN, activa), para una memoria Off-board y operación de E/S (activo OFF-BOARD), o un reconocimiento de ciclo de la interrupción (INTA activa).

Cuando el transceptor es permitido, el estado de la señal DT/R (CPU) en la entrada T, determina la dirección del transceptor.

Las líneas de dirección del BUS de expansión (y la señal BHE), son originadas directamente desde el latch de dirección (tres 745373 latches tri-state), cuando el 8086 tiene el control del BUS (HLDA inactivo). Cuando se le permite el control del BUS a los dispositivos periféricos (HLDA activada), el latch está en estado de alta Impedancia.

- El Tranceptor 8286:

Es un chip de 20 terminales cuya misión es servir de interfaz entre el procesador y el BUS de datos. Se utiliza como almacenamiento intermedio para los datos que llegan y salen del procesador, y es necesario por varias razones.

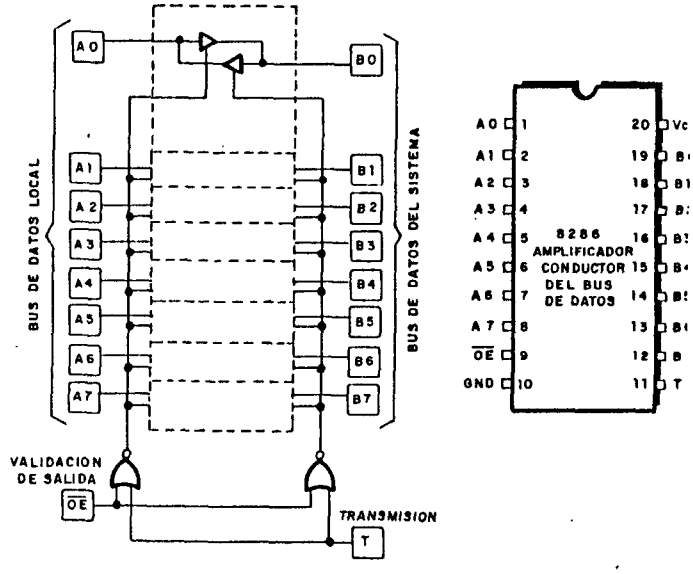
Por ejemplo, algunas veces las líneas de datos del procesador no pueden suministrar la corriente necesria para la carga de un dispositivo externo, y tienen que amplificarse antes de llegar al bus del sistema.

Otro uso del transceptor, es el de convertir las señales bidireccionales que ciertos Buses de datos externos necesitan en señales unidireccionales.

Otra posibilidad es la de ayudar a distinguir las señales de datos sobre las señales de direcciones.

Observe que en el 8086, las señales de datos y direcciones están multiplexadas en los mismos terminales.

Diagrama del tranceptor de datos octal 8286.



Es decir que el 8286 (Amplificador conductor del bus de datos), se encarga, fundamentalmente, de amplificar las corrientes de las líneas bidireccionales del BUS de datos. Además, cuando es preciso, convierte las señales en unidireccionales y se encarga de separar la información multiplexada que, como se sabe, consiste en sacar, por las mismas líneas, los datos y direcciones.

APENDICE A:

"SET DE INSTRUCCIONES"

Instruction		Object Code	Bytes	Clock Periods
AAA		37	1	4
AAD		D5 0A	2	60
AAM		D4 0A	2	83
AAS		3F	1	4
ADC	ac,data	0001010w kk [jj]	2 or 3	4
ADC	mem/reg$_1$,data	100000sw mod 010 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
ADC	mem/reg$_1$,mem/reg$_2$	000100dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 3 mem to reg: 9 + EA reg to mem: 16 + EA
ADD	ac,data	0000010w kk [jj]	2 or 3	4
ADD	mem/reg,data	100000sw mod 000 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
ADD	mem/reg$_1$,mem/reg$_2$	000000dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 3 mem to reg: 9 + EA reg to mem: 16 + EA
AND	ac,data	0010010w kk [jj]	2 or 3	4
AND	mem/reg,data	1000000w mod 100 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem:17 + EA
AND	mem/reg$_1$,mem/reg$_2$	001000dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 3 mem to reg: 9 + EA reg to mem: 16 + EA
CALL	addr	9A kk jj hh gg	5	28
CALL	disp16	E8 kk jj	3	19
CALL	mem	FF mod 011 r/m [DISP] [DISP]	2, 3 or 4	32-bit mem pointer: 37 + EA
CALL	mem/reg	FF mod 010 r/m [DISP] [DISP]	2, 3, or 4	16-bit reg pointer: 16 16-bit mem pointer: 21 + EA

Instruction		Object Code	Bytes	Clock Periods
CBW		98	1	2
CLC		F8	1	2
CLD		FC	1	2
CLI		FA	1	2
CMC		F5	1	2
CMP	ac,data	0011110w kk (jj)	2 or 3	4
CMP	mem/reg,data	100000sw mod 111 r/m [DISP] [DISP] kk (jj)	3, 4, 5 or 6	reg: 4 mem: 10 + EA
CMP	mem/reg_1,mem/reg_2	001110dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 3 mem to reg: 9 + EA reg to mem: 9 + EA
CMPS		1010011w	1	22 9 + 22/repetition*
CWD		99	1	5
DAA		27	1	4
DAS		2F	1	4
DEC	mem/reg	1111111w mod 001 r/m [DISP] [DISP]	2, 3 or 4	reg: 3 mem: 15 + EA
DEC	16-bit reg	01001rrr	1	2
DIV	mem/reg	1111011w mod 110 r/m [DISP] [DISP]	2, 3 or 4	8-bit reg: 80 — 90 16-bit reg: 144 — 162 8-bit mem (86 — 96) + EA 16-bit mem: (150 — 168) + EA
ESC	mem/reg	11011xxx mod xxx r/m [DISP] [DISP]	2, 3 or 4	mem: 8 + EA reg: 2
HLT		F4	1	2
IDIV	mem/reg	1111011w mod 111 r/m [DISP] [DISP]	2, 3 or 4	8-bit reg: 101 — 112 16-bit reg: 165 — 184 8-bit mem: (107 — 118) + EA 16-bit mem: (171 — 190) + EA
IMUL	mem/reg	1111011w mod 101 r/m [DISP] [DISP]	2, 3 or 4	8-bit reg: 80 — 98 16-bit reg: 128 — 154 8-bit mem: (86 — 104) + EA 16-bit mem: (134 — 160) + EA
IN	ac, DX	1110110w	1	8
IN	ac, port	1110010w	2	10

* When preceded by REP prefix

Instruction		Object Code	Bytes	Clock Periods
INC	mem/reg	1111111w mod 000 r/m [DISP] [DISP]	2, 3 or 4	reg: 3 mem: 15 + EA
INC	16-bit reg	01000rrr	1	2
INT		11001100* 11001101 type	1 2	52 51
INTO		CE	1	interrupt: 53 no interrupt: 4
IRET		CF	1	32
JA JNBE	disp	77 disp	2	4/No Branch 16/Branch
JAE JNB	disp	73 disp	2	4/No Branch 16/Branch
JB JNAE	disp	72 disp	2	4/No Branch 8/Branch
JBE JNA	disp	76 disp	2	4/No Branch 16/Branch
JCXZ	disp	E3 disp	2	6/No Branch 18/Branch
JE JZ	disp	74 disp	2	4/No Branch 16/Branch
JG JNLE	disp	7F disp	2	4/No Branch 16/Branch
JGE JNL	disp	7D disp	2	4/No Branch 16/Branch
JL JNGE	disp	7C disp	2	4/No Branch 16/Branch
JLE JNG	disp	7E disp	2	4/No Branch 16/Branch
JMP	addr	EA kk ii hh gg	5	15
JMP	disp	EB disp	2	15
JMP	disp16	E9 kk ii	3	15
JMP	mem	FF mod 101 r/m [DISP] [DISP]	2, 3 or 4	mem ptr 32: 24 + EA
JMP	mem/reg	FF mod 100 r/m [DISP] [DISP]	2, 3 or 4	reg ptr 16: 11 mem ptr 16: 16 + EA
JNE JNZ	disp	75 disp	2	4/No Branch 16/Branch
JNO	disp	71 disp	2	4/No Branch 16/Branch
JNP JPO	disp	7B disp	2	4/No Branch 16/Branch
JNS	disp	79 disp	2	4/No Branch 16/Branch
JO	disp	70 disp	2	4/No Branch 16/Branch

* Implied type = 3

Instruction		Object Code	Bytes	Clock Periods
JP	disp	7A disp	2	4/No Branch 16/Branch
JPE				
JS	disp	78 disp	2	4/No Branch 16/Branch
LAHF		9F	1	4
LDS	reg,mem	C5 mod rrr r/m [DISP] [DISP]	2, 3 or 4	16 + EA
LEA	reg,mem	8D mod rrr r/m [DISP] [DISP]	2, 3 or 4	2 + EA
LES	reg,mem	C4 mod rrr r/m [DISP] [DISP]	2, 3 or 4	16 + EA
LOCK		F0	1	2
LODS		1010110w	1	12 9 + 13/repetition*
LOOP	disp	E2 disp	2	5/No Branch 17/Branch
LOOPE LOOPZ	disp	E1 disp	2	6/No Branch 18/Branch
LOOPNE LOOPNZ	disp	E0 disp	2	5/No Branch 19/Branch
MOV	mem/reg$_1$,mem/reg$_2$	100010dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 2 reg to mem: 8 + EA mem to reg: 9 + EA
MOV	reg,data	1011wrrr kk [jj]	2 or 3	4
MOV	ac,mem	1010000w kk jj	3	10
MOV	mem,ac	1010001w kk jj	3	10
MOV	segreg,mem/reg	8E mod 0rr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 2 mem to reg: 8 + EA
MOV	mem/reg,segreg	8C mod 0rr r/m [DISP] [DISP]	2, 3 or 4	reg to reg: 2 reg to mem: 9 + EA
MOV	mem/reg,data	1100011w mod 000 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg/mem: 10 + EA
MOVS		1010010w	1	18 9 + 17/repetition*

* When preceded by REP prefix

Instruction		Object Code	Bytes	Clock Periods
MUL	mem/reg	1111011w *mod* 100 r/m [DISP] [DISP]	2, 3 or 4	8-bit reg: 70 — 77 16-bit reg: 118 — 133 8-bit *mem*: (76 — 83) + EA 16-bit mem: (124 — 139) + EA
NEG	mem/reg	1111011w *mod* 011 r/m [DISP] [DISP]	2, 3 or 4	reg: 3 *mem*: 16 + EA
NOP		90	1	3
NOT	mem/reg	1111011w mod 010 r/m [DISP] [DISP]	2, 3 or 4	reg: 3 mem: 16 + EA
OR	ac,data	0000110w kk [jj]	2 or 3	4
OR	mem/reg,data	1000000w mod 001 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
OR	mem/reg_1,mem/reg_2	000010dw *mod rrr r/m* [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg to reg: 3 mem to reg: 9 + EA reg to mem: 16 + EA
OUT	DX,ac	1110111w	1	8
OUT	port,ac	1110011w yy	2	10
POP	mem/reg	8F mod 000 r/m [DISP] [DISP]	2, 3 or 4	reg: 8 mem: 17 + EA
POP	reg	01011rrr	1	8
POP	segreg	000ss111	1	8
POPF		*9D*	1	8
PUSH	mem/reg	FF mod 110 r/m [DISP] [DISP]	2, 3 or 4	reg: 11 mem: 16 + EA
PUSH	reg	01010rrr	1	10
PUSH	segreg	000ss110	1	10
PUSHF		9C	1	10
RCL	mem/reg,count	110100cw *mod* 010 r/m [DISP] [DISP]	2, 3 or 4	count = 1 *reg*: 2 mem:15 + EA count = [CL] *reg*: 8 + (4 * N) mem: 20 + EA + (4 * N)

N = count value in CL

Instruction		Object Code	Bytes	Clock Periods
RCR	mem/reg,count	110100cw mod 011 r/m [DISP] [DISP]	2, 3 or 4	*count* = 1 reg: 2 *mem*:15 + EA count = [CL] *reg*: 8 + (4 * N) mem: 20 + EA + (4 * N)
REP	/REPE/REPNE	1111001z	1	2
RET	(inter-segment)	CB	1	24
RET	(intra-segment)	C3	1	16
RET	disp16(inter-segment)	CA kk jj	3	23
RET	disp16(intra-segment)	C2 kk jj	3	20
ROL	mem/reg,count	110100cw mod 000 r/m [DISP] [DISP]	2, 3 or 4	count = 1 reg: 2 *mem*:15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)
ROR	mem/reg,count	110100cw *mod 001 r/m* [DISP] [DISP]	2, 3 or 4	count = 1 *reg*: 2 mem:15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)
SAHF		9E	1	4
SAR	mem/reg,count	110100cw mod 111 r/m [DISP] [DISP]	2, 3 or 4	count = 1 *reg*: 2 mem:15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)
SBB	ac,data	0001110w kk [jj]	2 or 3	4
SBB	mem/reg,data	100000sw mod 011 r/m [DISP] [DISP] *kk* [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
SBB	mem/reg_1,mem/reg_2	000110dw mod rr r/m [DISP] [DISP]	2, 3 or 4	*reg from reg*: 3 mem from reg: 9 + EA *reg from mem*: 16 + EA
SCAS		1010111w	1	15 9 + 15/repetition*
SEG	segreg	001ss110	1	2
SHL SAL	*mem/reg,count*	110100cw mod 100 r/m [DISP] [DISP]	2, 3 or 4	*count* = 1 reg: 2 mem:15 + EA count = [CL] reg: 8 + (4 * N) *mem*: 20 + EA + (4 * *N*)

* When preceded by REP prefix
N = count value in CL

Instruction		Object Code	Bytes	Clock Periods
SHR	mem/reg,count	110100cw mod 101 r/m [DISP] [DISP]	2, 3 or 4	count = 1 reg: 2 mem:15 + EA count = [CL] reg: 8 + (4 * N) mem: 20 + EA + (4 * N)
STC		F9	1	2
STD		FD	1	2
STI		FB	1	2
STOS		1010101w	1	11 9 + 10/repetition*
SUB	ac,data	0010110w kk [jj]	2 or 3	4
SUB	mem/reg,data	100000sw mod 101 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
SUB	mem/reg_1,mem/reg_2	001010dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg from reg: 3 mem from reg: 9 + EA reg from mem: 16 + EA
TEST	ac,data	1010100w kk [jj]	2 or 3	4
TEST	mem/reg,data	1111011w mod 000 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 5 mem: 11 + EA
TEST	reg,mem/reg	1000010w mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg with reg: 3 reg with mem: 9 + EA
WAIT		9B	1	3(min.) + 5n
XCHG	reg,ac	10010rrr	1	3
XCHG	reg,mem/reg	1000011w mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg with reg: 4 reg with mem: 17 + EA
XLAT		D7	1	11
XOR	ac,data	0011010w kk [jj]	2 or 3	4
XOR	mem/reg,data	1000000w mod 110 r/m [DISP] [DISP] kk [jj]	3, 4, 5 or 6	reg: 4 mem: 17 + EA
XOR	mem/reg_1,mem/reg_2	001100dw mod rrr r/m [DISP] [DISP]	2, 3 or 4	reg with reg: 3 mem with reg: 9 + EA reg with mem: 16 + EA

*When preceded by REP prefix

n = clocks per sample of the $\overline{TEST}$ input

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
00	mod reg r/m	[disp][disp]	ADD mem/reg,reg (byte)
01	mod reg r/m	[disp][disp]	ADD mem/reg,reg (word)
02	mod reg r/m	[disp][disp]	ADD reg, mem/reg (byte)
03	mod reg r/m	[disp][disp]	ADD reg, mem/reg (word)
04	kk		ADD AL,kk
05	kk	jj	ADD AX, jjkk
06			PUSH ES
07			POP ES
08	mod reg r/m	[disp][disp]	OR mem/reg,reg (byte)
09	mod reg r/m	[disp][disp]	OR mem/reg,reg (word)
0A	mod reg r/m	[disp][disp]	OR reg,mem/reg (byte)
0B	mod reg r/m	[disp][disp]	OR reg,mem/reg (word)
0C	kk		OR AL,kk
0D	kk	jj	OR AL,jjkk
0E			PUSH CS
0F			Not used
10	mod reg r/m	[disp][disp]	ADC mem/reg,reg (byte)
11	mod reg r/m	[disp][disp]	ADC mem/reg,reg (word)
12	mod reg r/m	[disp][disp]	ADC reg,mem/reg (byte)
13	mod reg r/m	[disp][disp]	ADC reg,mem/reg (word)
14	kk		ADC AL,kk
15	kk	jj	ADC AX,jjkk
16			PUSH SS
17			POP SS
18	mod reg r/m	[disp][disp]	SBB mem/reg,reg (byte)
19	mod reg r/m	[disp][disp]	SBB mem/reg,reg (word)
1A	mod reg r/m	[disp][disp]	SBB reg,mem/reg (byte)
1B	mod reg r/m	[disp][disp]	SBB reg,mem/reg (word)
1C	kk		SBB AL,kk
1D	kk	jj	SBB AX,jjkk
1E			PUSH DS
1F			POP DS
20	mod reg r/m	[disp][disp]	AND mem/reg,reg (byte)
21	mod reg r/m	[disp][disp]	AND mem/reg,reg (word)
22	mod reg r/m	[disp][disp]	AND reg,mem/reg (byte)
23	mod reg r/m	[disp][disp]	AND reg,mem/reg (word)
24	kk		AND AL,kk
25	kk	jj	AND AX,jjkk
26			SEG ES
27			DAA
28	mod reg r/m	[disp][disp]	SUB mem/reg,reg (byte)
29	mod reg r/m	[disp][disp]	SUB mem/reg,reg (word)
2A	mod reg r/m	[disp][disp]	SUB reg,mem/reg (byte)
2B	mod reg r/m	[disp][disp]	SUB reg,mem/reg (word)
2C	kk		SUB AL,kk
2D	kk	jj	SUB AX,jjkk
2E			SEG CS
2F			DAS

Object Code			Mnemonic
Byte ≠ 0	Byte # 1	Succeeding Bytes	
30	mod reg r/m	[disp][disp]	XOR mem/reg,reg (byte)
31	mod reg r/m	[disp][disp]	XOR mem/reg,reg (word)
32	mod reg r/m	[disp][disp]	XOR reg,mem/reg (byte)
33	mod reg r/m	[disp][disp]	XOR reg,mem/reg (word)
34	kk		XOR AL,kk
35	kk	jj	XOR AX,jjkk
36			SEG SS
37			AAA
38	mod reg r/m	[disp][disp]	CMP mem/reg,reg (byte)
39	mod reg r/m	[disp][disp]	CMP mem/reg,reg (word)
3A	mod reg r/m	[disp][disp]	CMP reg,mem/reg (byte)
3B	mod reg r/m	[disp][disp]	CMP reg,mem/reg (word)
3C	kk		CMP AL,kk
3D	kk	jj	CMP AX,jjkk
3E			SEG DS
3F			AAS
40			INC AX
41			INC CX
42			INC DX
43			INC BX
44			INC SP
45			INC BP
46			INC SI
47			INC DI
48			DEC AX
49			DEC CX
4A			DEC DX
4B			DEC BX
4C			DEC SP
4D			DEC BP
4E			DEC SI
4F			DEC DI
50			PUSH AX
51			PUSH CX
52			PUSH DX
53			PUSH BX
54			PUSH SP
55			PUSH BP
56			PUSH SI
57			PUSH DI
58			POP AX
59			POP CX
5A			POP DX
5B			POP BX
5C			POP SP
5D			POP BP
5E			POP SI
5F			POP DI
60-6F			Not Used

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
70	disp		JO disp
71	disp		JNO disp
72	disp		JB or JNAE or JC disp
73	disp		JNB or JAE or JNC disp
74	disp		JE or JZ disp
75	disp		JNE or JNZ disp
76	disp		JBE or JNA disp
77	disp		JNBE or JA disp
78	disp		JS disp
79	disp		JNS disp
7A	disp		JP or JPE disp
7B	disp		JNP or JPO disp
7C	disp		JL or JNGE disp
7D	disp		JNL or JGE disp
7E	disp		JLE or JNG disp
7F	disp		JNLE or JG disp
80	mod 000 r/m	[disp][disp] kk	ADD mem/reg,kk
80	mod 001 r/m	[disp][disp] kk	OR mem/reg,kk
80	mod 010 r/m	[disp][disp] kk	ADC mem/reg,kk
80	mod 011 r/m	[disp][disp] kk	SBB mem/reg,kk
80	mod 100 r/m	[disp][disp] kk	AND mem/reg,kk
80	mod 101 r/m	[disp][disp] kk	SUB mem/reg,kk
80	mod 110 r/m	[disp][disp] kk	XOR mem/reg, kk
80	mod 111 r/m	[disp][disp] kk	CMP mem/reg,kk
81	mod 000 r/m	[disp][disp] kkjj	ADD mem/reg,jjkk
81	mod 001 r/m	[disp][disp] kkjj	OR mem/reg,jjkk
81	mod 010 r/m	[disp][disp] kkjj	ADC mem/reg,jjkk
81	mod 011 r/m	[disp][disp] kkjj	SBB mem/reg,jjkk
81	mod 100 r/m	[disp][disp] kkjj	AND mem/reg,jjkk
81	mod 101 r/m	[disp][disp] kkjj	SUB mem/reg,jjkk
81	mod 110 r/m	[disp][disp] kkjj	XOR mem/reg,jjkk
81	mod 111 r/m	[disp][disp] kkjj	CMP mem/reg,jjkk
82	mod 000 r/m	[disp][disp] kk	ADD mem/reg,kk (byte)
82	xx 001 xxx		Not used
82	mod 010 r/m	[disp][disp] kk	ADC mem/reg,kk (byte)
82	mod 011 r/m	[disp][disp] kk	SBB mem/reg,kk (byte)
82	xx 100 xxx		Not used
82	mod 101 r/m	[disp][disp] kk	SUB mem/reg,kk (byte)
82	xx 110 xxx		Not used
82	mod 111 r/m	[disp][disp] kk	CMP mem/reg,kk (byte)
83	mod 000 r/m	[disp][disp] kk	ADD mem/reg,jjkk (word-sign extended)
83	xx 001 xxx		Not used
83	mod 010 r/m	[disp][disp] kk	ADC mem/reg,jjkk (word-sign extended)
83	mod 011 r/m	[disp][disp] kk	SBB mem/reg,jjkk (word-sign extended)
83	xx 100 r/m		Not used
83	mod 101 r/m	[disp][disp] kk	SUB mem/reg,jjkk (word-sign extended)
83	xx 110 xxx		Not used
83	mod 111 r/m	[disp][disp] kk	CMP mem/reg,jjkk (word-sign extended)
84	mod reg r/m	[disp][disp]	TEST mem/reg,reg (byte)
85	mod reg r/m	[disp][disp]	TEST mem/reg,reg (word)
86	mod reg r/m	[disp][disp]	XCHG reg,mem/reg (byte)
87	mod reg r/m	[disp][disp]	XCHG reg,mem/reg (word)
88	mod reg r/m	[disp][disp]	MOV mem/reg,reg (byte)
89	mod reg r/m	[disp][disp]	MOV mem/reg,reg (word)

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
8A	mod reg r/m	[disp][disp]	MOV reg,mem/reg (byte)
8B	mod reg r/m	[disp][disp]	MOV reg,mem/reg (word)
8C	mod 0ss r/m	[disp][disp]	MOV mem/reg,segreg
8C	xx 1xxxxx		Not used
8D	mod reg r/m	[disp][disp]	LEA reg,addr.
8E	mod 0ss r/m	[disp][disp]	MOV segreg, mem/reg
8E	xx 1xxxxx		Not used
8F	mod 000 r/m	[disp][disp]	POP mem/reg
8F	xx 001 xxx		Not used
8F	xx 010 xxx		Not used
8F	xx 011 xxx		Not used
8F	xx 100 xxx		Not used
8F	xx 101 xxx		Not used
8F	xx 110 xxx		Not used
8F	xx 111 xxx		Not used
			Not used
90			NOP
91			XCHG AX,CX
92			XCHG AX,DX
93			XCHG AX,BX
94			XCHG AX,SP
95			XCHG AX,BP
96			XCHG AX,SI
97			XCHG AX,DI
98			CBW
99			CWD
9A	kk	jj hh gg	CALL addr
9B			WAIT
9C			PUSHF
9D			POPF
9E			SAHF
9F			LAHF
A0	qq	pp	MOV AL,addr
A1	qq	pp	MOV AX,addr
A2	qq	pp	MOV addr,AL
A3	qq	pp	MOV addr,AX
A4			MOVS BYTE
A5			MOVS WORD
A6			CMPS BYTE
A7			CMPS WORD
A8	kk		TEST, AL,kk
A9	kk	jj	TEST AX,jjkk
AA			STOS BYTE
AB			STOS WORD
AC			LODS BYTE
AD			LODS WORD
AE			SCAS BYTE
AF			SCAS WORD

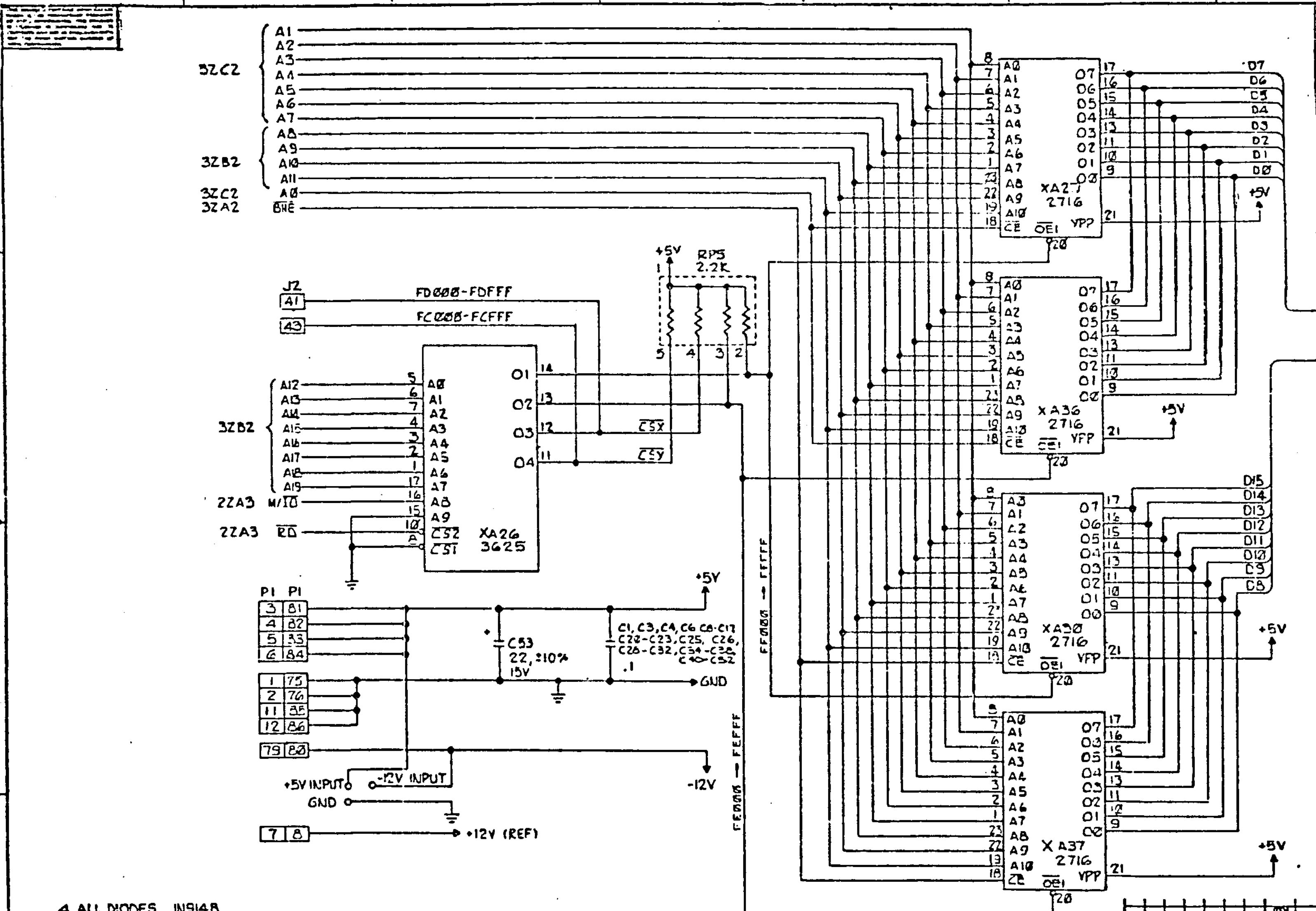
Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
B0	kk		MOV AL,kk
B1	kk		MOV CL,kk
B2	kk		MOV DL,kk
B3	kk		MOV BL,kk
B4	kk		MOV AH,kk
B5	kk		MOV CH,kk
B6	kk		MOV DH,kk
B7	kk		MOV BH,kk
B8	kk	jj	MOV AX,jjkk
B9	kk	jj	MOV CX,jjkk
BA	kk	jj	MOV DX,jjkk
BB	kk	jj	MOV BX,jjkk
BC	kk	jj	MOV SP,jjkk
BD	kk	jj	MOV BP,jjkk
BE	kk	jj	MOV SI,jjkk
BF	kk	jj	MOV DI,jjkk
C0			Not used
C1			Not used
C2	kk	jj	RET jjkk
C3			RET
C4	mod reg r/m	[disp][disp]	LES reg,addr
C5	mod reg r/m	[disp][disp]	LDS reg,addr
C6	mod 000 r/m	[disp][disp] kk	MOV mem,kk
C6	xx 001 xxx		Not used
C6	xx 010 xxx		Not used
C6	xx 011 xxx		Not used
C6	xx 100 xxx		Not used
C6	xx 101 xxx		Not used
C6	xx 110 xxx		Not used
C6	xx 111 xxx		Not used
C7	mod 000 r/m	[disp][disp] kkjj	MOV mem,jjkk
C7	xx 001 xxx		Not used
C7	xx 010 xxx		Not used
C7	xx 011 xxx		Not used
C7	xx 100 xxx		Not used
C7	xx 101 xxx		Not used
C7	xx 110 xxx		Not used
C7	xx 111 xxx		Not used
C8			Not used
C9			Not used
CA	kk	jj	RET jjkk
CB			RET
CC			INT 3
CD	type		INT Type
CE			INTO
CF			IRET

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
D0	mod 000 r/m	[disp][disp]	ROL mem/reg,1 (byte)
D0	mod 001 r/m	[disp][disp]	ROR mem/reg,1 (byte)
D0	mod 010 r/m	[disp][disp]	RCL mem/reg,1 (byte)
D0	mod 011 r/m	[disp][disp]	RCR mem/reg,1 (byte)
D0	mod 100 r/m	[disp][disp]	SAL or SHL mem/reg,1 (byte)
D0	mod 101 r/m	[disp][disp]	SHR mem/reg,1 (byte)
D0	xx 110 xxx		Not used
D0	mod 111 r/m	[disp][disp]	SAR mem/reg,1 (byte)
D1	mod 000 r/m	[disp][disp]	ROL mem/reg,1 (word)
D1	mod 001 r/m	[disp][disp]	ROR mem/reg,1 (word)
D1	mod 010 r/m	[disp][disp]	RCL mem/reg,1 (word)
D1	mod 011 r/m	[disp][disp]	RCR mem/reg,1 (word)
D1	mod 100 r/m	[disp][disp]	SAL or SHL mem/reg,1 (word)
D1	mod 101 r/m	[disp][disp]	SHR mem/reg,1 (word)
D1	xx 110 xxx		Not used
D1	mod 111 r/m	[disp][disp]	SAR mem/reg,1 (word)
D2	mod 000 r/m	[disp][disp]	ROL mem/reg,CL (byte)
D2	mod 001 r/m	[disp][disp]	ROR mem/reg,CL (byte)
D2	mod 010 r/m	[disp][disp]	RCL mem/reg,CL (byte)
D2	mod 011 r/m	[disp][disp]	RCR mem/reg,CL (byte)
D2	mod 100 r/m	[disp][disp]	SAL or SHL mem/reg,CL (byte)
D2	mod 101 r/m	[disp][disp]	SHR mem/reg,CL (byte)
D2	xx 110 xxx		Not used
D2	mod 111 r/m	[disp][disp]	SAR mem/reg,CL (byte)
D3	mod 000 r/m	[disp][disp]	ROL mem/reg,CL (word)
D3	mod 001 r/m	[disp][disp]	ROR mem/reg,CL (word)
D3	mod 010 r/m	[disp][disp]	RCL mem/reg,CL (word)
D3	mod 011 r/m	[disp][disp]	RCR mem/reg,CL (word)
D3	mod 100 r/m	[disp][disp]	SAL or SHL mem/reg,CL (word)
D3	mod 101 r/m	[disp][disp]	SHR mem/reg,CL (word)
D3	xx 110 xxx		Not used
D3	mod 111 r/m	[disp][disp]	SAR mem/reg,CL (word)
D4	0A		AAM
D5	0A		AAD
D6			Not used
D7			XLAT
D8	mod xxx r/m	[disp][disp]	ESC mem/reg
D9	mod xxx r/m	[disp][disp]	ESC mem/reg
DA	mod xxx r/m	[disp][disp]	ESC mem/reg
DB	mod xxx r/m	[disp][disp]	ESC mem/reg
DC	mod xxx r/m	[disp][disp]	ESC mem/reg
DD	mod xxx r/m	[disp][disp]	ESC mem/reg
DE	mod xxx r/m	[disp][disp]	ESC mem/reg
DF	mod xxx r/m	[disp][disp]	ESC mem/reg
E0	disp		LOOPNE/LOOPNZ disp
E1	disp		LOOPE/LOOPZ disp
E2	disp		LOOP disp
E3	disp		JCXZ disp
E4	kk		IN AL,kk
E5	kk		IN AX,kk
E6	kk		OUT kk,AL
E7	kk		OUT kk,AX
E8	disp	disp	CALL disp16
E9	disp	disp	JMP disp16

Object Code			Mnemonic
Byte # 0	Byte # 1	Succeeding Bytes	
EA	kk	jj hh gg	JMP addr
EB	disp		JMP disp
EC			IN AL,DX .
ED			IN AX,DX
EE			OUT DX,AL
EF			OUT DX,AX
F0			LOCK
F1			Not used
F2			REPNE or REPNZ
F3			REP or REPE or REPZ
F4			HLT
F5			CMC
F6	mod 000 r/m	[disp][disp] kk	TEST mem/reg,kk
F6	xx 001 xxx		Not used
F6	mod 010 r/m	[disp][disp]	NOT mem/reg (byte)
F6	mod 011 r/m	[disp][disp]	NEG mem/reg (byte)
F6	mod 100 r/m	[disp][disp]	MUL mem/reg (byte)
F6	mod 101 r/m	[disp][disp]	IMUL mem/reg (byte)
F6	mod 110 r/m	[disp][disp]	DIV mem/reg (byte)
F6	mod 111 r/m	[disp][disp]	IDIV mem/reg (byte)
F7	mod 000 r/m	[disp][disp] kkjj	TEST mem/reg,jjkk
F7	xx 001 xxx		Not used
F7	mod 010 r/m	[disp][disp]	NOT mem/reg (word)
F7	mod 011 r/m	[disp][disp]	NEG mem/reg (word)
F7	mod 100 r/m	[disp][disp]	MUL mem/reg (word)
F7	mod 101 r/m	[disp][disp]	IMUL mem/reg (word)
F7	mod 110 r/m	[disp][disp]	DIV mem/reg (word)
F7	mod 111 r/m	[disp][disp]	IDIV mem/reg (word)
F8			CLC
F9			STC
FA			CLI
FB			STI
FC			CLD
FD			STD
FE	mod 000 r/m	[disp][disp]	INC mem/reg (byte)
FE	mod 001 r/m	[disp][disp]	DEC mem/reg (byte)
FE	xx 010 xxx		Not used
FE	xx 011 xxx		Not used
FE	xx 100 xxx		Not used
FE	xx 101 xxx		Not used
FE	xx 110 xxx		Not used
FE	xx 111 xxx		Not used
FF	mod 000 r/m	[disp][disp]	INC mem/reg (word)
FF	mod 001 r/m	[disp][disp]	DEC mem/reg (word)
FF	mod 010 r/m	[disp][disp]	CALL mem/reg
FF	mod 011 r/m	[disp][disp]	CALL mem
FF	mod 100 r/m	[disp][disp]	JMP mem/reg
FF	mod 101 r/m	[disp][disp]	JMP mem
FF	mod 110 r/m	[disp][disp]	PUSH mem
FF	xx 111 xxx		Not used

APENDICE B:

"ESQUEMAS DEL SDK-86"



IC	+5V VCC	GND
8086	40	1,20
8255A	26	7
8251A	26	4
8284	18	9
8279	40	20
2716	24	12
2142	20	10
3625	16	8
8286	20	10
74LS00	14	7
74LS04	14	7
74LS10	14	7
74LS14	14	7
74LS20	14	7
74S30	14	7
74LS74	14	7
74LS155	16	8
74LS164	14	7
74LS244	20	10
74LS393	14	7
7445	16	8
74S133	16	8
74S373	20	10
ULN2003A	9	8

REVISIONS

LTR	DESCRIPTION
1	ENGR RELEASE
2	ECO 2-2353
A	ECO 2-3151
—	PROD RLSE, RW0016

4. ALL DIODES IN914B.
3. ALL TRANSISTORS Q2T2905.
2. ALL CAPACITANCE VALUES ARE IN UF, +80-20%, 50V
1. ALL RESISTANCE VALUES ARE IN OHMS, ± 5%, 1/4 WATT.

NOTES: UNLESS OTHERWISE SPECIFIED,

LAST USED	NOT USED	TYPE		
R25 W41	W35	Q2T2905	Q1	1
A46 Q3	R1			
ZP8 S24	A+20-23			
Y1 C53 E77	C2 C5, C24 C27			
C58 CR3 J7	R21			
REF DESIGNATION		SPARE GATES		

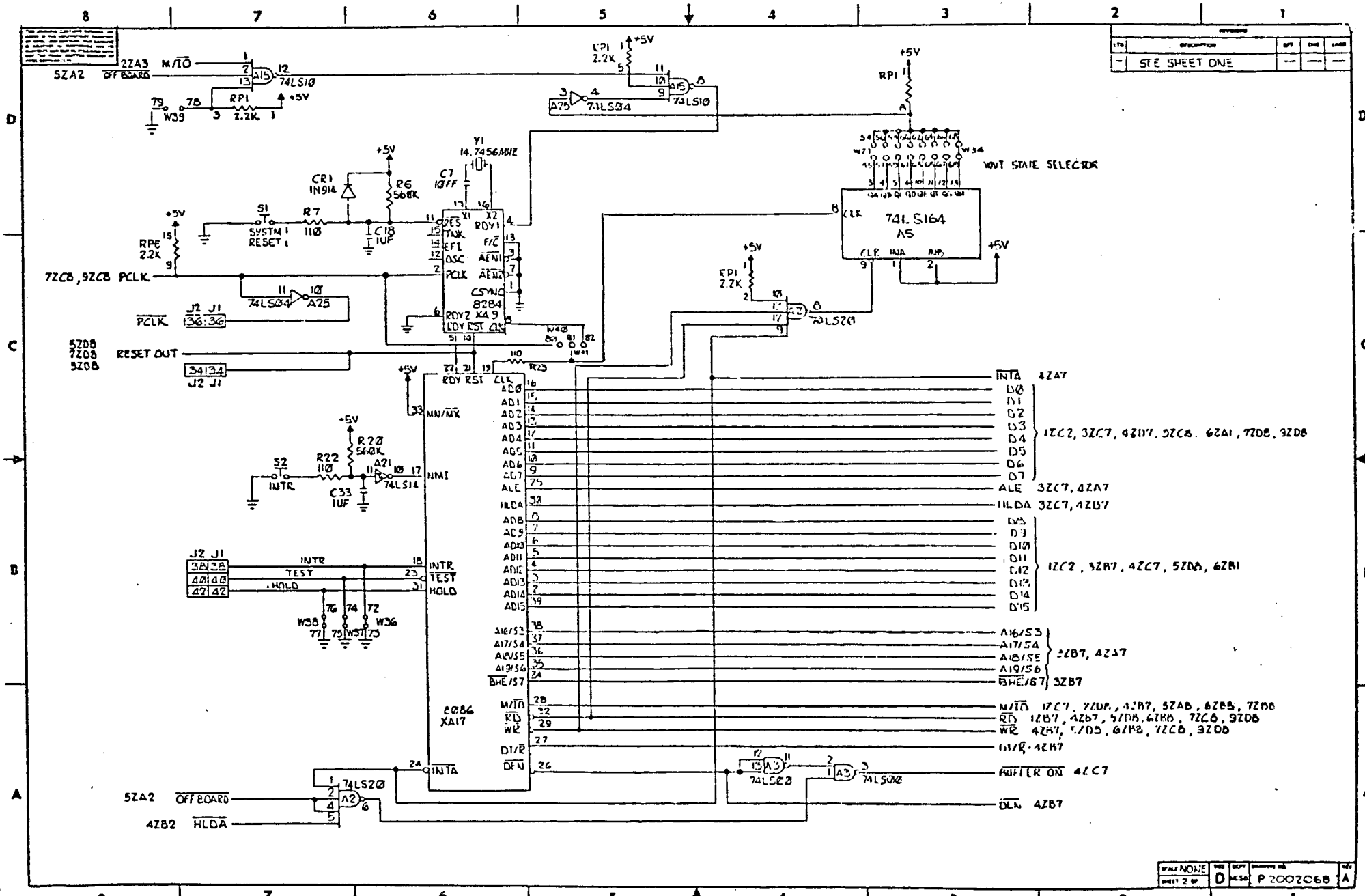
SCALE NONE

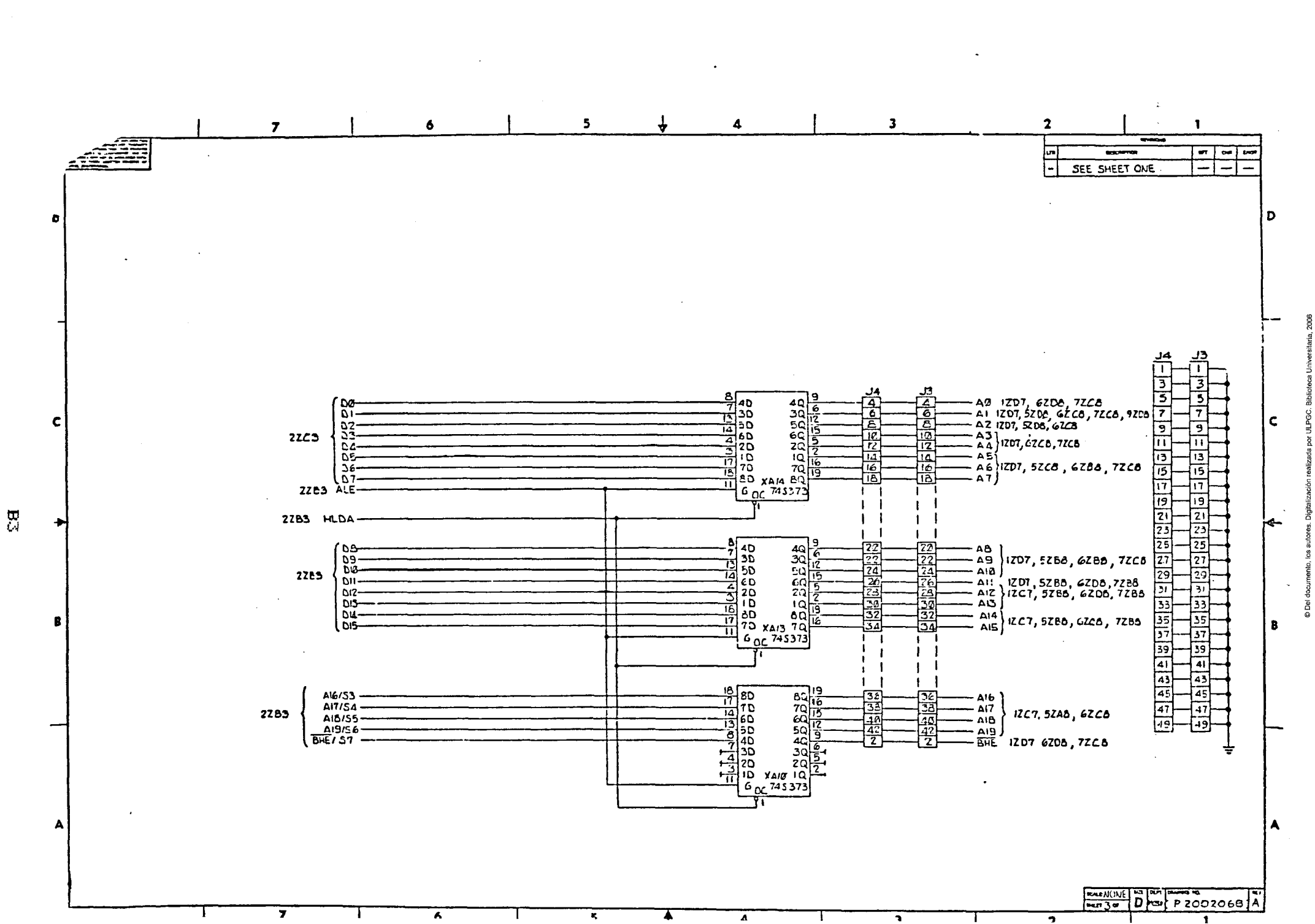
intel

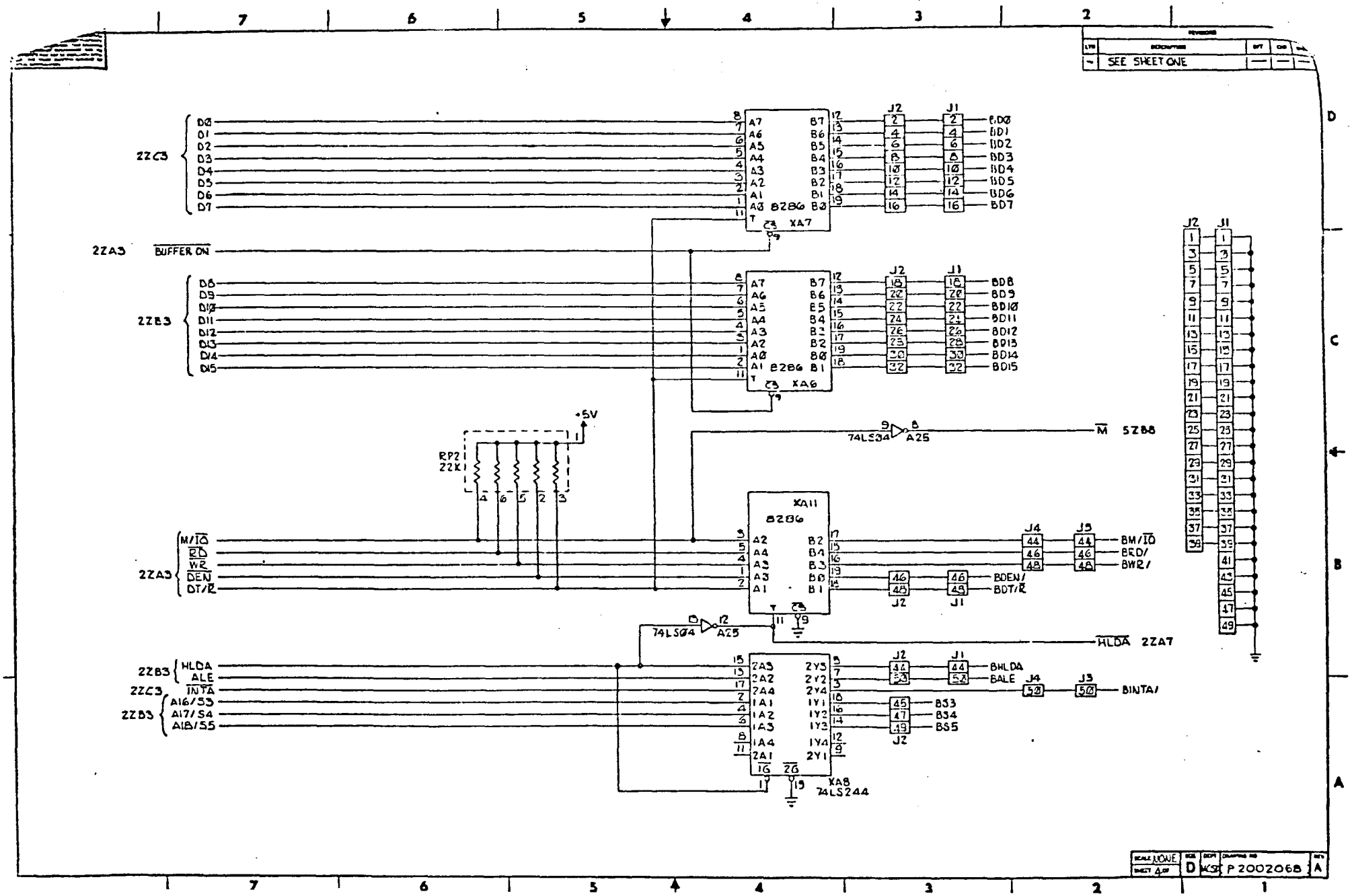
SCHEMATIC SDK-86

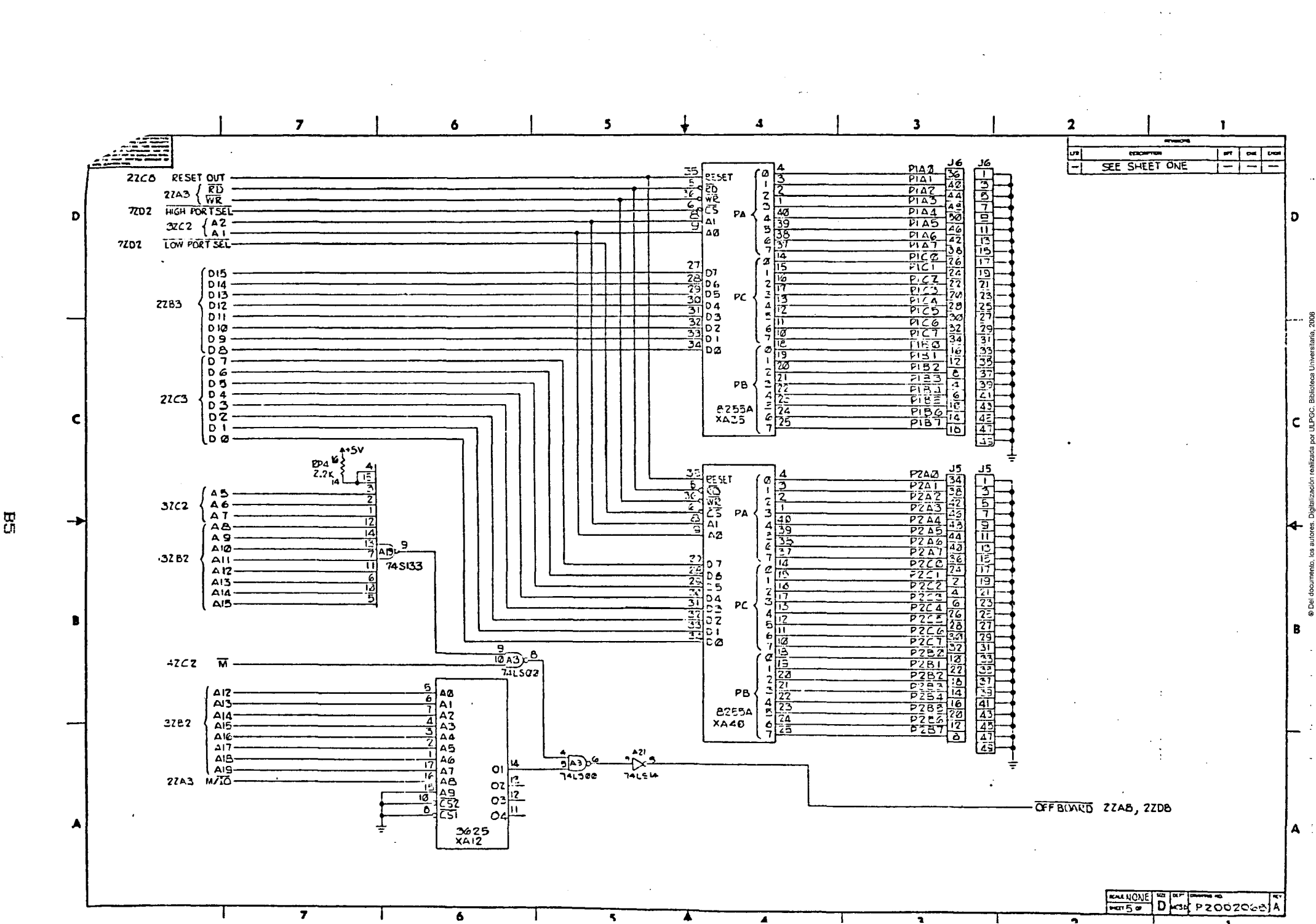
D | MCSO | P2002068 | A

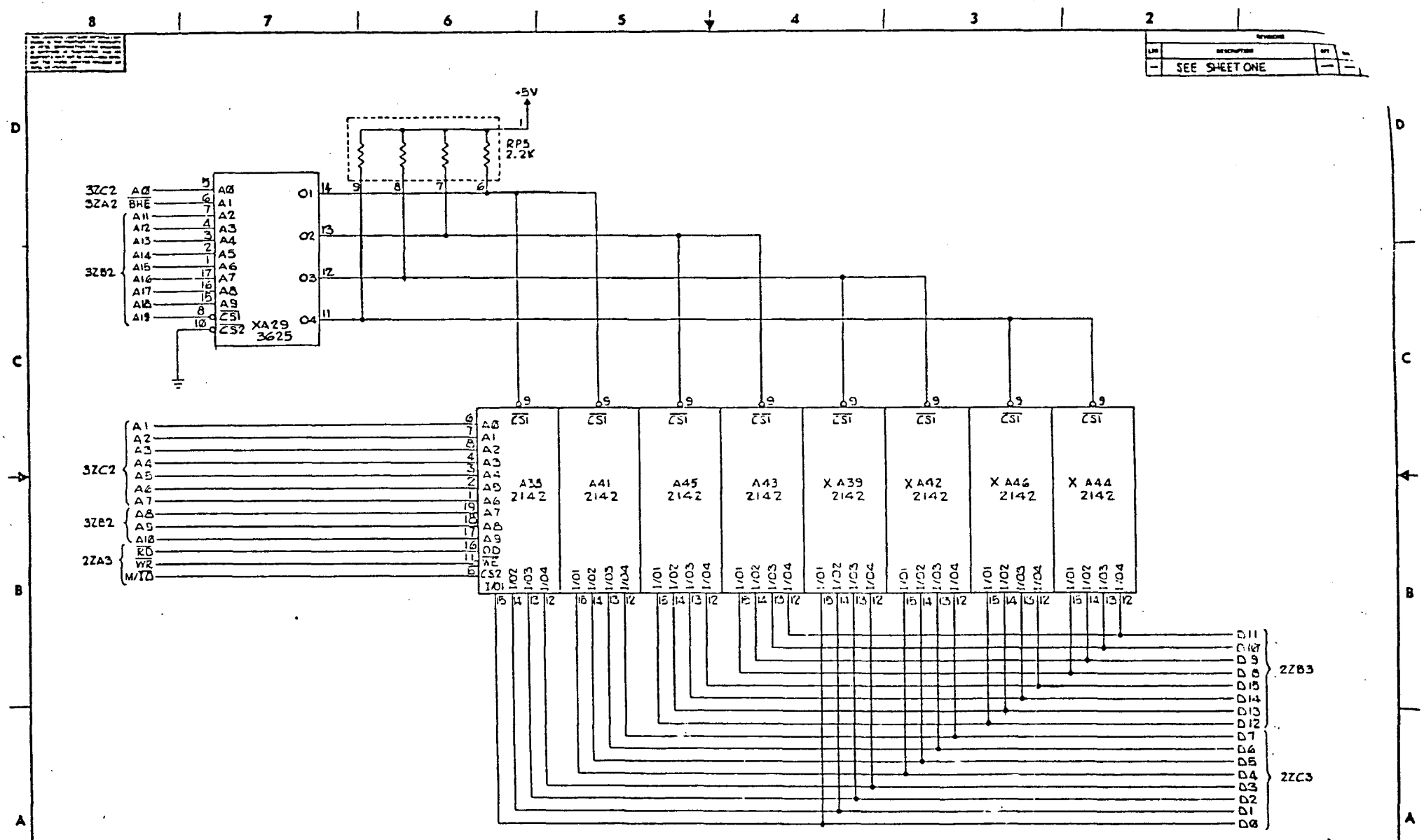
SHEET 1 OF 5

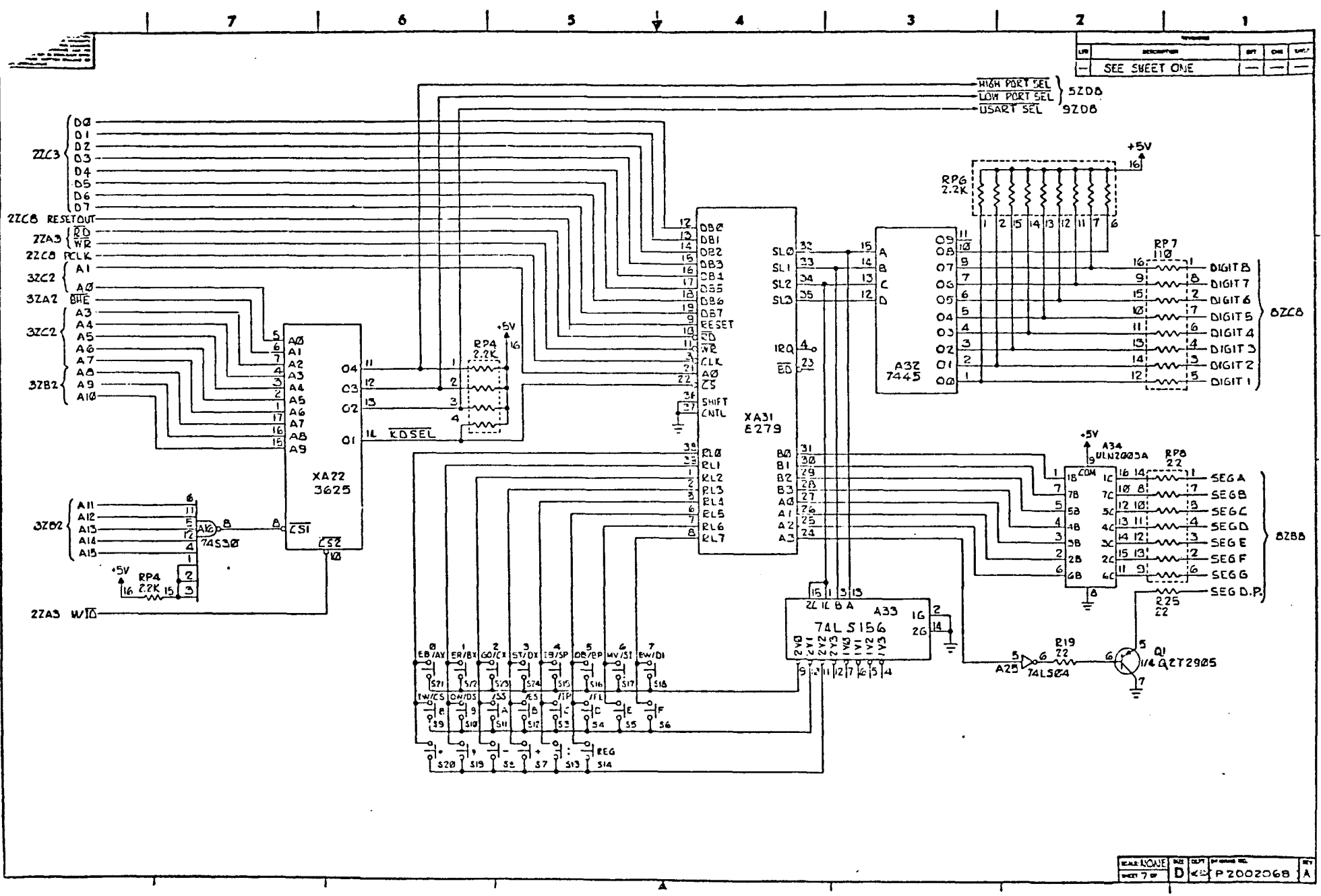


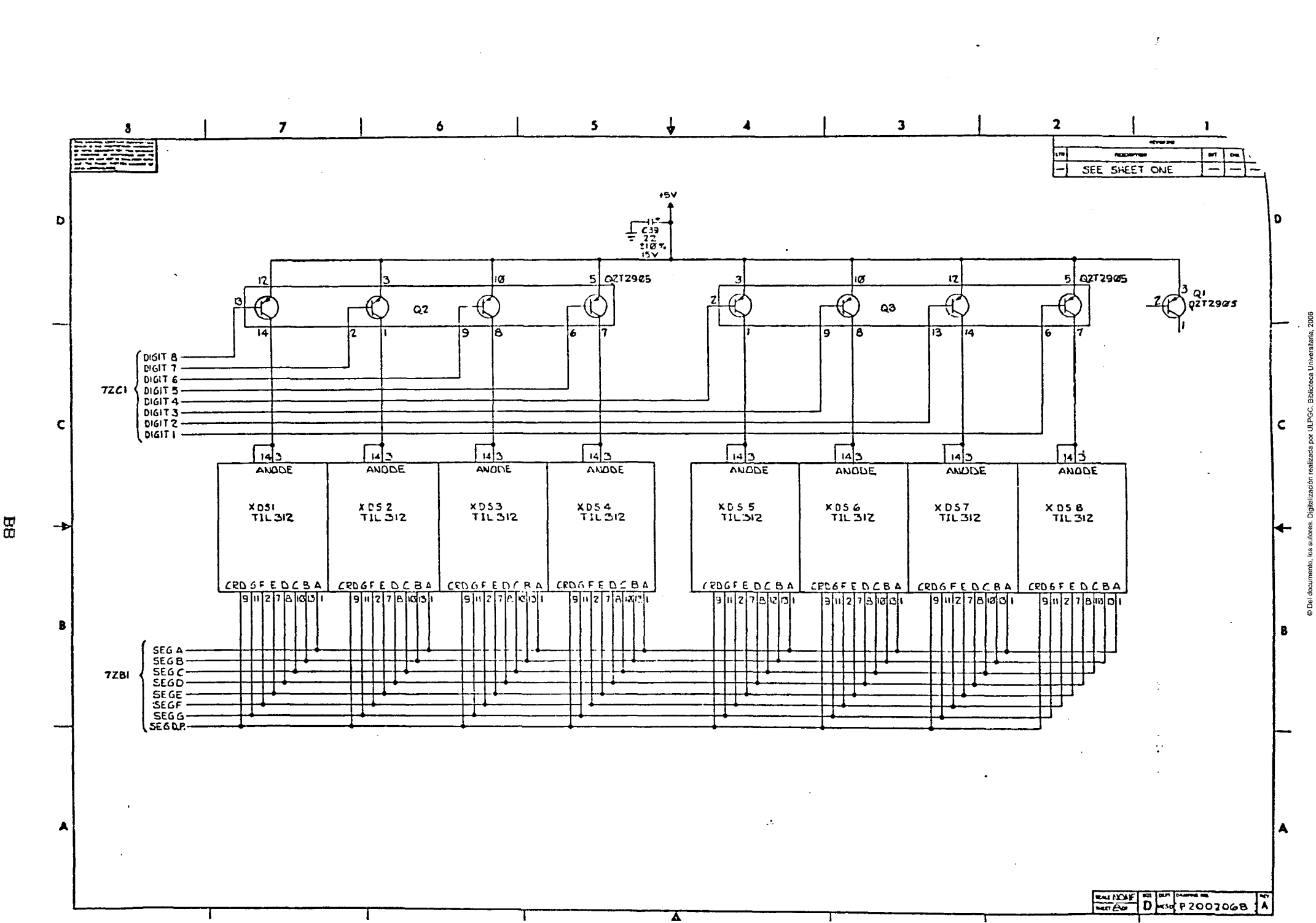


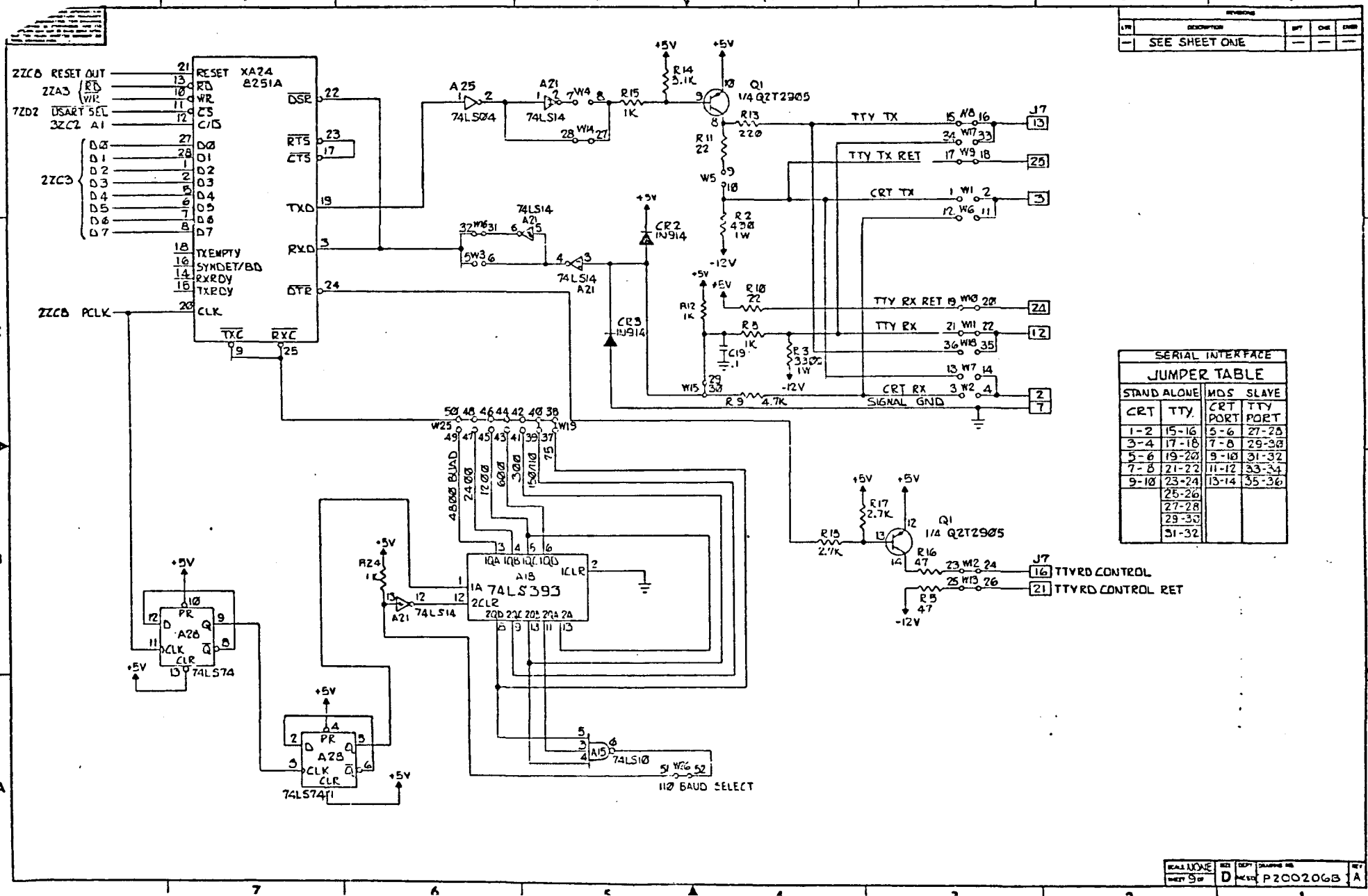












APLICACION:

"SISTEMA MULTIUSUARIO"

APLICACION PARA UN SISTEMA MULTIUSUARIO

Vamos a describir un pequeño sistema multiusuario.

Se basa en la capacidad de un sistema para procesar aparentemente más de una función a un tiempo.

I- Hardware del sistema:

El sistema está formado por cinco circuitos integrados:

- La CPU 8086
- El 8155: Memoria Ram, E/S y dispositivo Timer.

  Memoria: 256 palabras de 8 bits.

  E/S: 2 puertos programables de 8 bits y uno de 6 bits.

  Timer.

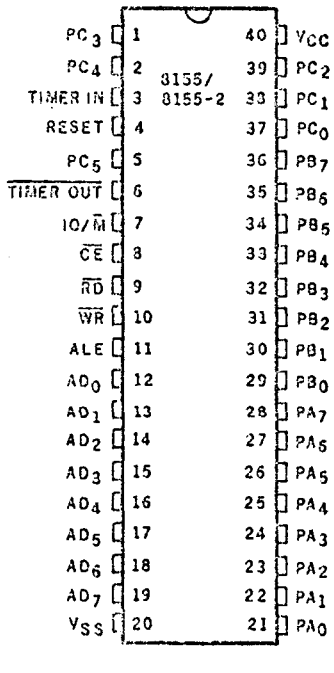
  El bit de menor peso del puerto A se toma como línea serie de salida para el usuario 1. Ao= OUT

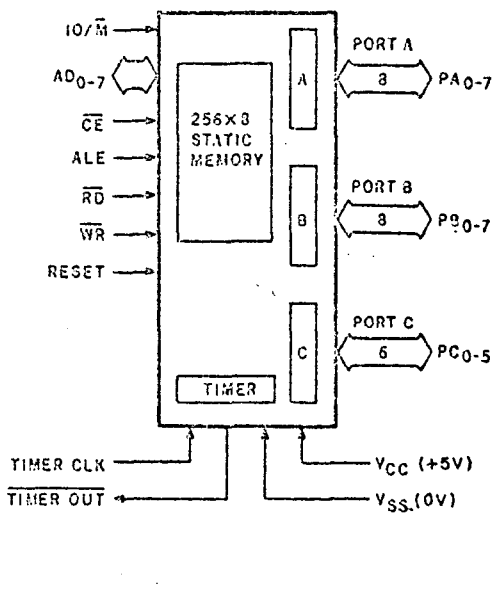
  El bit de mayor peso del puerto B se toma como línea serie de entrada para el usuario 1. B7= IN

  Las direcciones en el sistema son:

Registro comando-status	00H
Puerto A	01H
Puerto B	02H
Puerto C	03H
Timer (bajo)	04H
Timer (alto)	05H

En la siguiente figura se muestra la pastilla 8155:





- El 8755A: Memoria EPROM y E/S.

  E/S: 2 puertos programables de 8 bits.

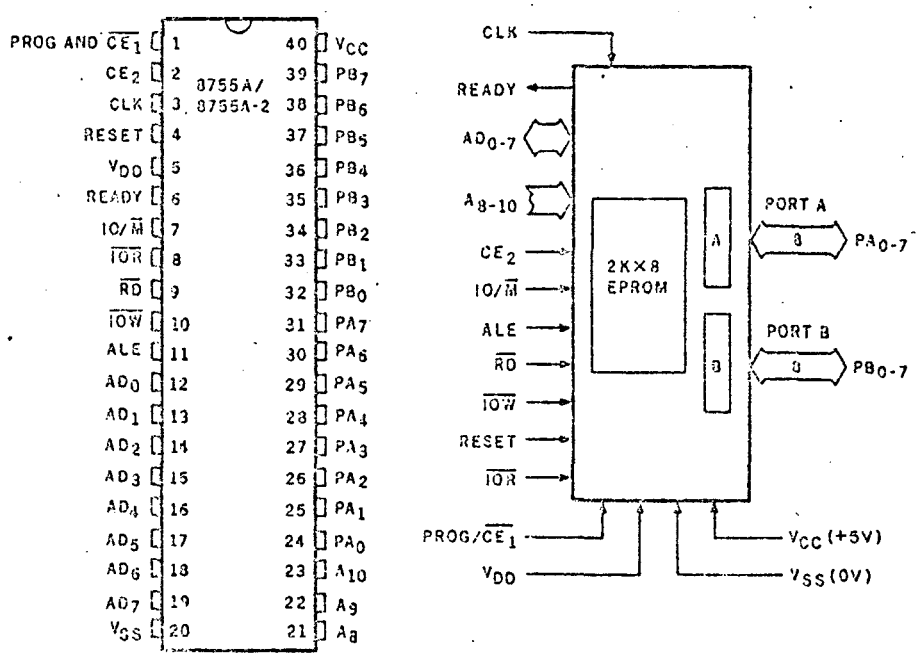
  El bit de menor peso del puerto A se toma como línea serie de salida para el usuario 2. Ao= OUT

  El bit de mayor peso del puerto B se toma como línea serie de entrada para el usuario 2. B7= IN

  Las direcciones en el sistema son:

Puerto A	F000H
Puerto B	F001H
Dato-dirección registro A	F002H
Dato-dirección registro B	F003H

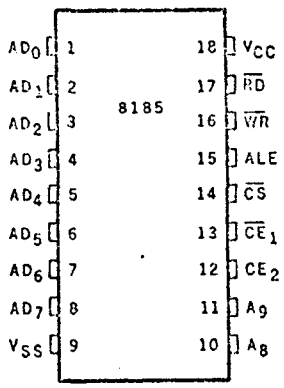
En la siguiente figura se muestra la pastilla 8755A:

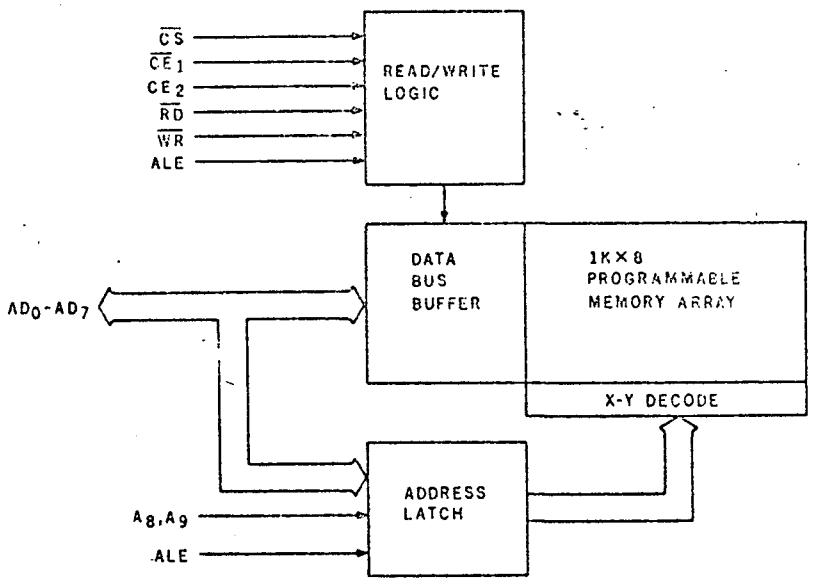


- El 8185:

1 Kbytes de memoria estática programable.

En la siguiente figura se muestra la pastilla 8185:

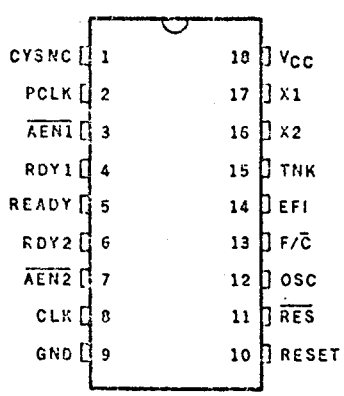


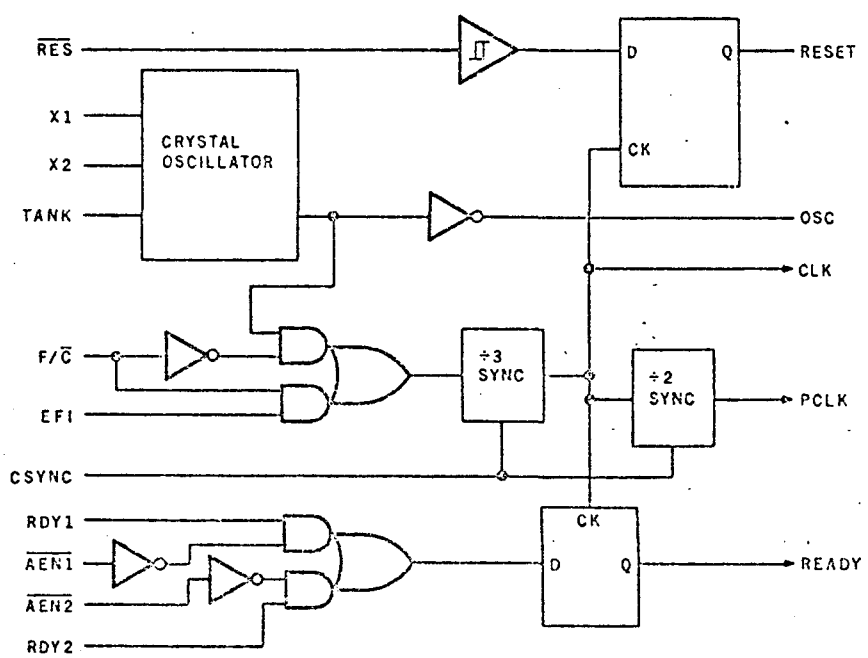


-El 8284:

Generador de señales de reloj.

En la siguiente figura se muestra la pastilla 8284:





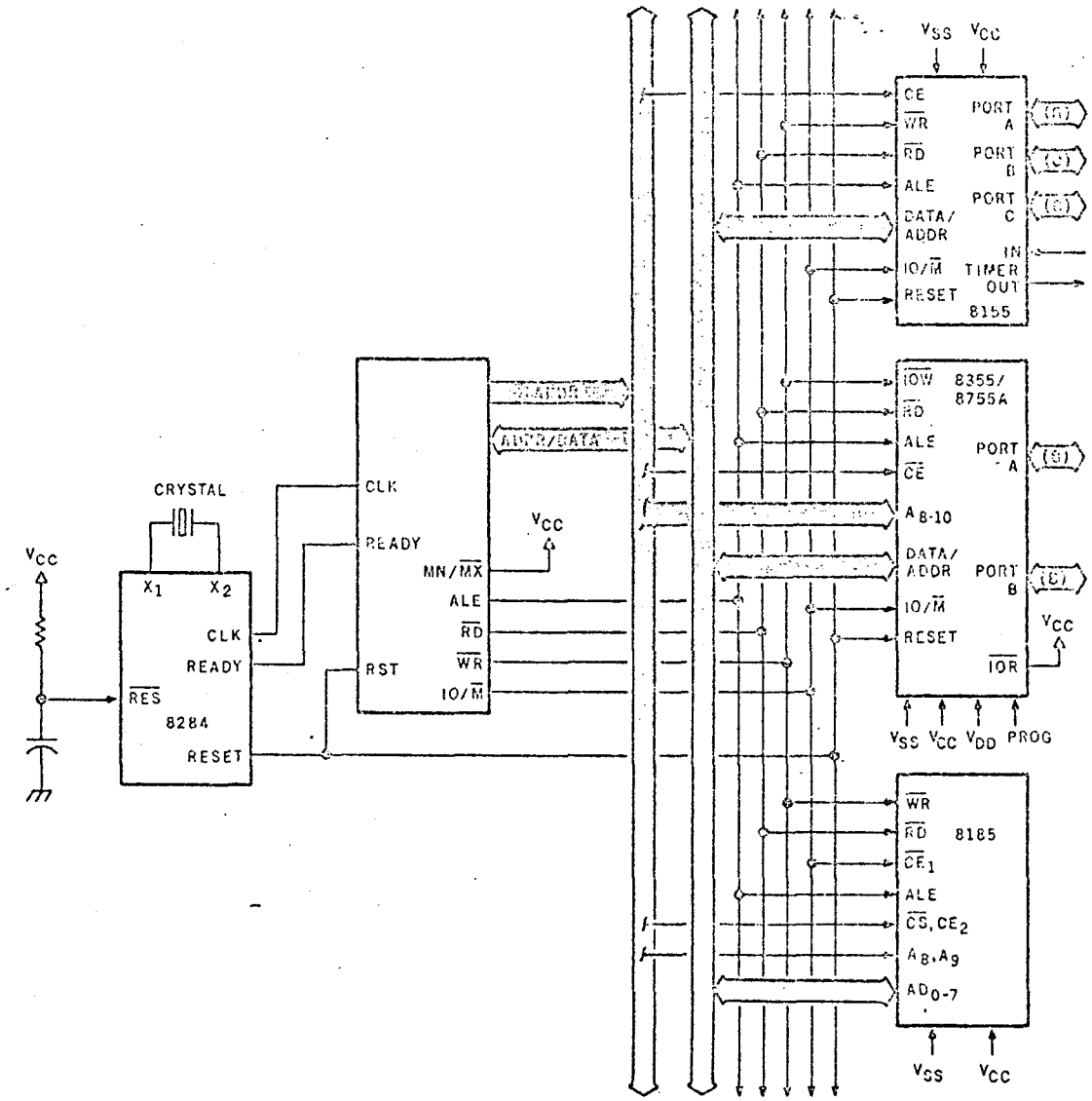
Todos estos circuitos integrados están específicamente diseñados para trabajar con el bus de datos y direcciones multiplexados. Por consiguiente, no hay necesidad de tener latches de salida para proporcionar las señales de dirección para sus operaciones. Los latches de direcciones para cada dispositivo están provistos internamente.

Todos los circuitos integrados usados en este diseño son compatibles con la señal de 5 Mhz generada por el 8284; sin embargo, el timer/contador del 8155 trabaja mejor con excitación de 2,5 Mhz, que es la salida de la señal PCLK del 8284.

En la siguiente figura se ve el flujo de datos del sistema, así como las direcciones de la memoria y puertos de E/S.

Esto permite servir a multiples usuarios simultáneamente.

La línea timer-IN en el 8155 va por un hilo a la señal PCLK del 8284. La línea timer-OUT en el 8155 va por un hilo a la señal NMI del 8086.



II- Desarrollo del sistema:

El permiso para la entrada y salida de los diferentes usuarios es facil, puesto que tanto el 8155 como el 8755A proporcionan dos puertos de E/S de 8 bits. Estos se necesitan para usar uno como entrada y el otro como salida. Un bit de dato se lleva a in o out a cada interrupción desde el timer.

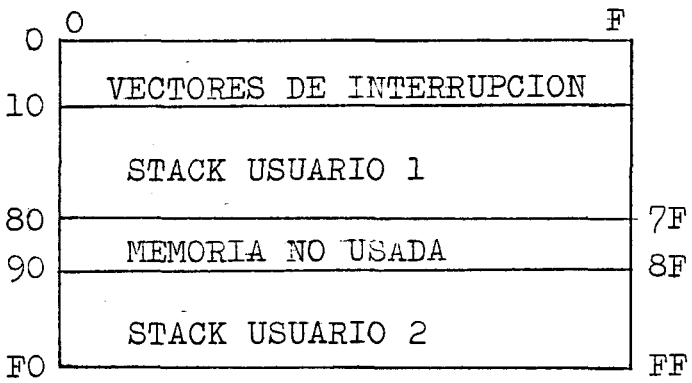
La velocidad de los datos para la comunicación con los terminales de los usuarios, se obtiene usando el timer programable del 8155 como generador de velocidad de dato. El contador binario de 14 bits se pone a preset durante la rutina de inicialización del sistema. Una vez puesto, el contador continua contando y genera una señal de interrupción cuando se alcanza el valor especificado.

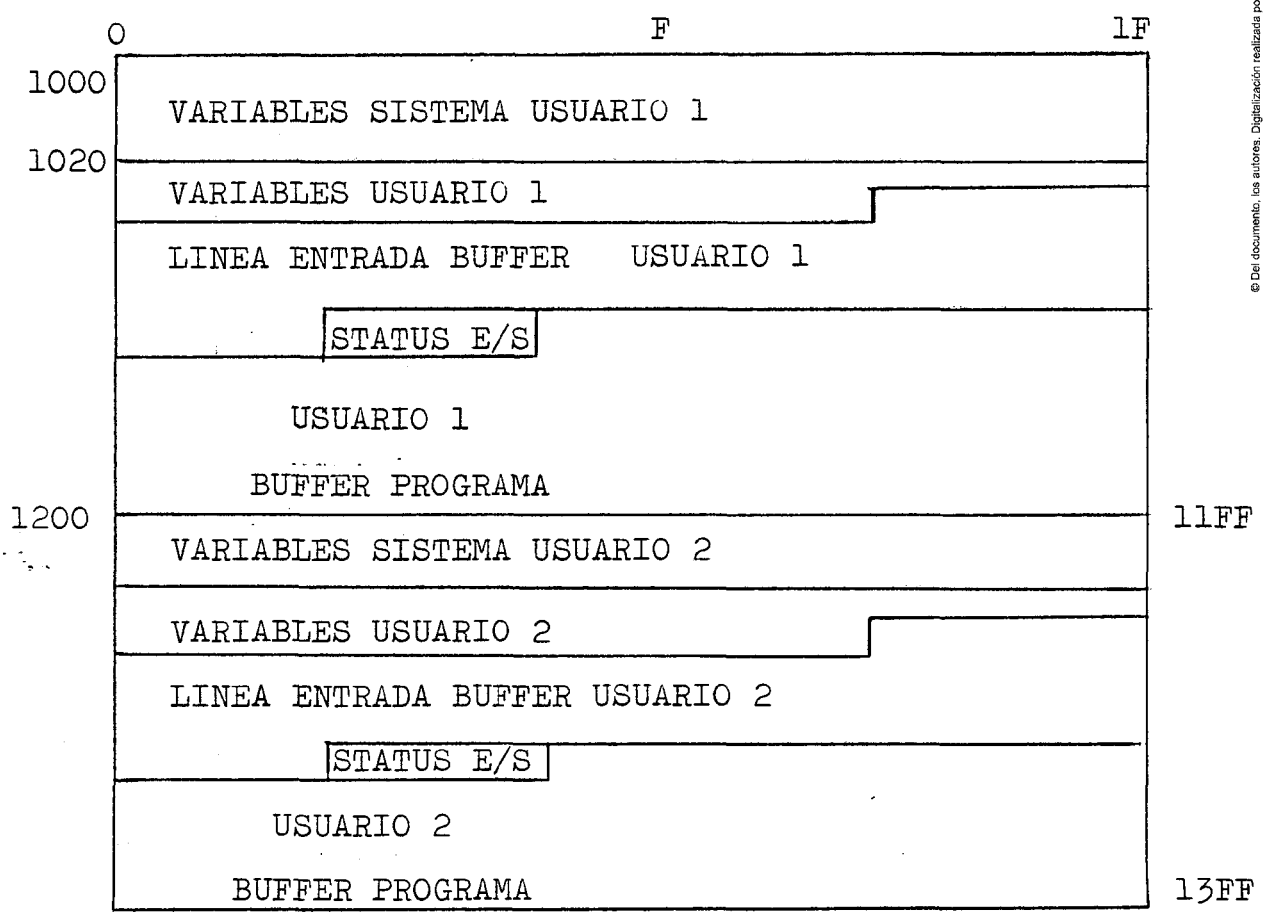
El valor puesto en el contador determina la velocidad de transferencia del dato. En este sistema el valor del contador está contenido en la EPROM y no es por tanto facil de cambiar.

La velocidad de los datos debe ser escogido, y el valor del contador computado, antes de la programación de la EPROM.

Dividiendo la memoria entre los usuarios, la tarea se realiza fácilmente. Todo lo que se necesita es asignar a cada usuario un área específica para ser usada por el programa, buffers y stack.

En la siguiente figura se muestra el mapa de memoria:





El problema de posición de memoria lo soluciona el procesador diferenciando las zonas de los distintos usuarios.

Los dos Kbytes de la memoria EPROM se utilizan para almacenar el sistema operativo, para inicializar los terminales de los usuarios.

Otra consideración en interes de la eficiencia del sistema total, es la de otorgar más tiempo de ejecución a uno de los usuarios si el otro está trabajando en la ejecución de alguna clase de loop de espera de E/S.

Normalmente el procesador conmutará la actual tarea del usuario que ha sido ejecutada cada vez que recibe una interrupción desde el timer. De esta manera, cada tarea de usuario recibirá una cantidad igual de tiempo de ejecución sobre el sistema.

Sin embargo, mientras el sistema está esperando por la entrada de los comandos de un usuario o mientras éste está enviando información al terminal, éste no produce tarea a ejecutar por el usuario.

Si ambos usuarios están en un modo E/S, como en un tiempo de comienzo del sistema, el procesador entrará en una espera loop, esperando una interrupción del timer.

De esta manera el procesador partirá el tiempo por igual con los dos usuarios.

III- Solución del problema:

El principal asunto, diferenciar entre los dos usuarios y sus programas, se resuelve fácilmente con el 8086. Este procesador direcciona todas las posiciones de memoria usando uno de los cuatro registros segmento. Todos los saltos (JUMPS) y llamadas (CALLS) a subrutinas dentro de un programa son hechos relativos a la actual posición de ejecución, sí los registros segmento han sido puestos correctamente.

De aquí, los saltos y llamadas no son especificados a los segmentos de memoria donde un código de sección de programa es puesto. El código puede ser movido de un lugar a otro dentro de la memoria, y perteneciendo a un usuario en un modo relativo, y modificar el área actual de la memoria accedida por un cambio del registro segmento apuntando al área que contiene la tarea del usuario que nosotros queremos trabajar.

Como se ve en el mapa de memoria, el 8155 está dividido en dos áreas, una para cada usuario para el stack .

Los un Kbytes de memoria del 8185, están divididos en cuatro áreas para cada usuario. Es decir, se usan para almacenar datos variados pertenecientes a las tareas de los usuarios.

Las dos áreas de stack del 8155 están separadas por 80 bytes en decimal.

Cada uno de los buffers en el área del progama buffer de memoria en el 8185, están separados por 200H bytes.

Cuando el procesador necesita acceder a un área dada de memoria, la dirección efectiva de la memoria que va a ser accedida, es computabilizada por el apropiado registro segmento multiplicado por 16, y se le suma el desplazamiento dentro del segmento.

Por ejemplo, si el procesador quiere cargar el byte en la posición 154H dentro del segmento, y el registro segmento contiene el valor 14H, la dirección efectiva se calcula:

$$14H \times 10H + 154H = 294H$$

Luego la dirección real en memoria es la 294H.

Por tanto, si queremos que el procesador acceda al stack del usuario uno, tenemos que poner el registro segmento stack a cero. Con lo cual,al multiplicar por 10H, estamos dentro del área de stack del usuario uno:

10h a 7FH

Si queremos que el procesador acceda al stack del usuario dos, tenemos que poner el registro segmento stack con el valor 8H. Con lo cual, al multiplicar por 10H, estamos dentro del área de stack del usuario dos:

90H a FFH

Los buffers del programa siguen esencialmente el mismo camino. Para el usuario uno, el registro segmento dato y el registro segmento extra son puestos a cero, y el programa es escrito en las direcciones 1000H a 11FFH. Para acceder al programa del usuario dos, se cargan los registros segmento con el valor 12H. Con lo cual, el procesador computabilizará la dirección efectiva de la dirección 1200H a la 13FFH.

Usando esta caracteristica, se puede trabajar con varios usuarios commutando entre ellos por la sóla manipulación de los registros segmentos.

Cuando el procesador es reset, la rutina de inicialización inicializa todos los puertos de E/S y saca la señal inicial del stop-bit al terminal.

Es también puesta el área del stack del usuario dos, así que el procesador empezará la ejecución en la rutina

Start cuando es procesada a través del usuario uno.
Después de poner la correcta velocidad de transferencia del dato para los terminales de los usuarios, la rutina de inicialización salta a Start para el usuario uno.
La rutina de inicialización es requerida, para que los registros, áreas de buffers y los stacks sean puestos correctamente para cada usuario antes de que cualquier otro proceso empieze.
Una vez empezado el normal procesamiento, la rutina que dirige la interrupción timer-out, conmuta la tarea de cada usuario sobre cada ciclo de interrupción y determina el tiempo de entrada y salida de información a los terminales.
La rutina timer-out se muestra en el organigrama del final.
Cuando se llama en respuesta a una interrupción, ésta se procesa como sigue:
Después de salvar los registros del usuario actual, así esa información almacenada en ellos será aprovechada cuando se reanude la ejecución sobre la tarea de los u-

suarios, las rutinas leen un byte desde cada uno de los puertos de entrada. Esto se hace de tal modo que las entradas ocurran a la vez.

A continuación, el dato es sacado a los terminales.

Al efectuar esta tarea, un byte de status de tarea es reservado en memoria para cada usuario.

Sí éste byte es 1, indica que el terminal está en modo salida.

Sí éste byte es 2, indica que el terminal está en modo entrada.

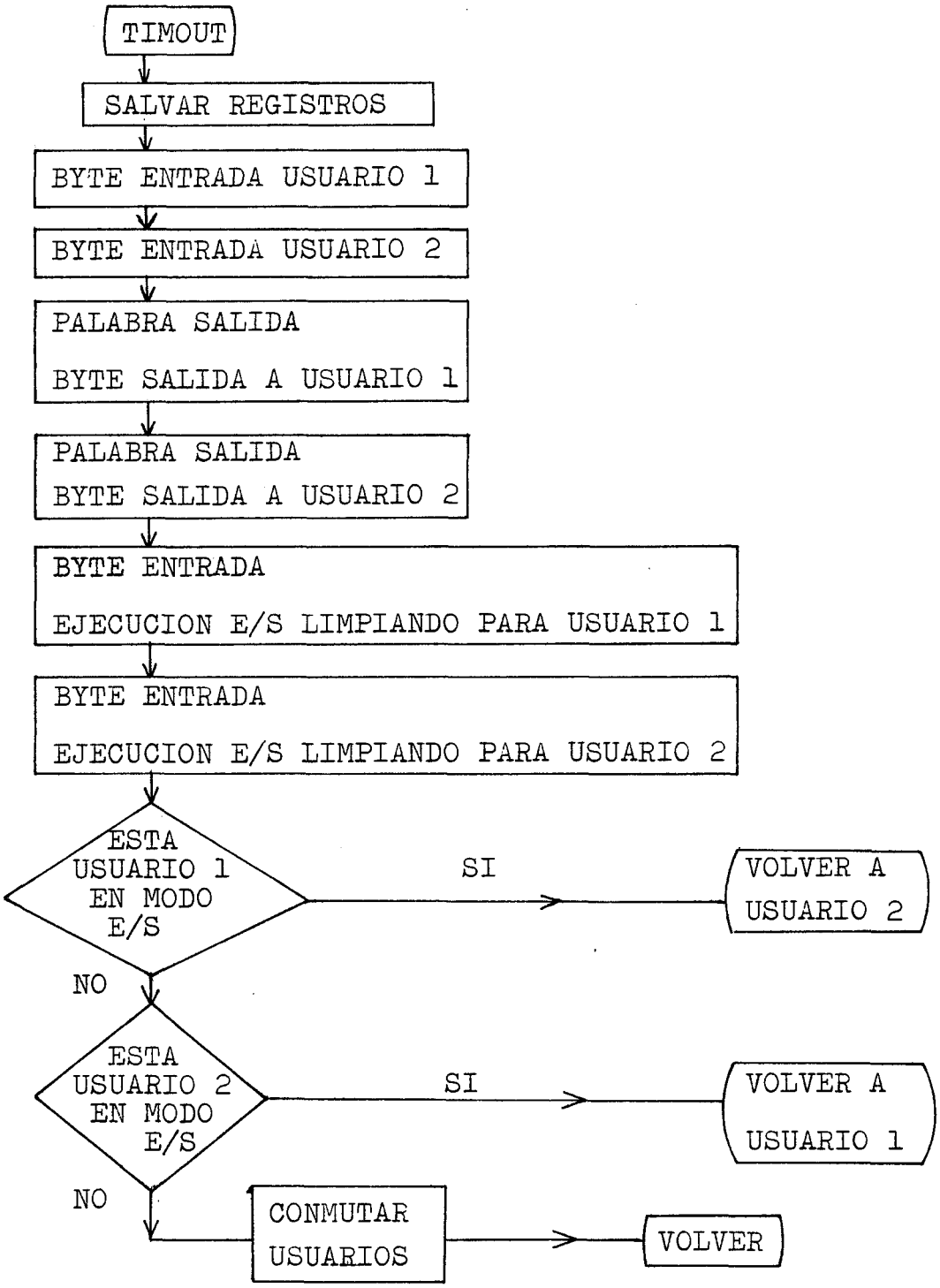
Sí éste byte es 0, indica que la tarea del usuario está actualmente ejecutándose sin ejecutar operaciones de E/S.

Cuando la E/S ha sido hecha, el procesador determina cual tarea-usuario es servida a continuación.

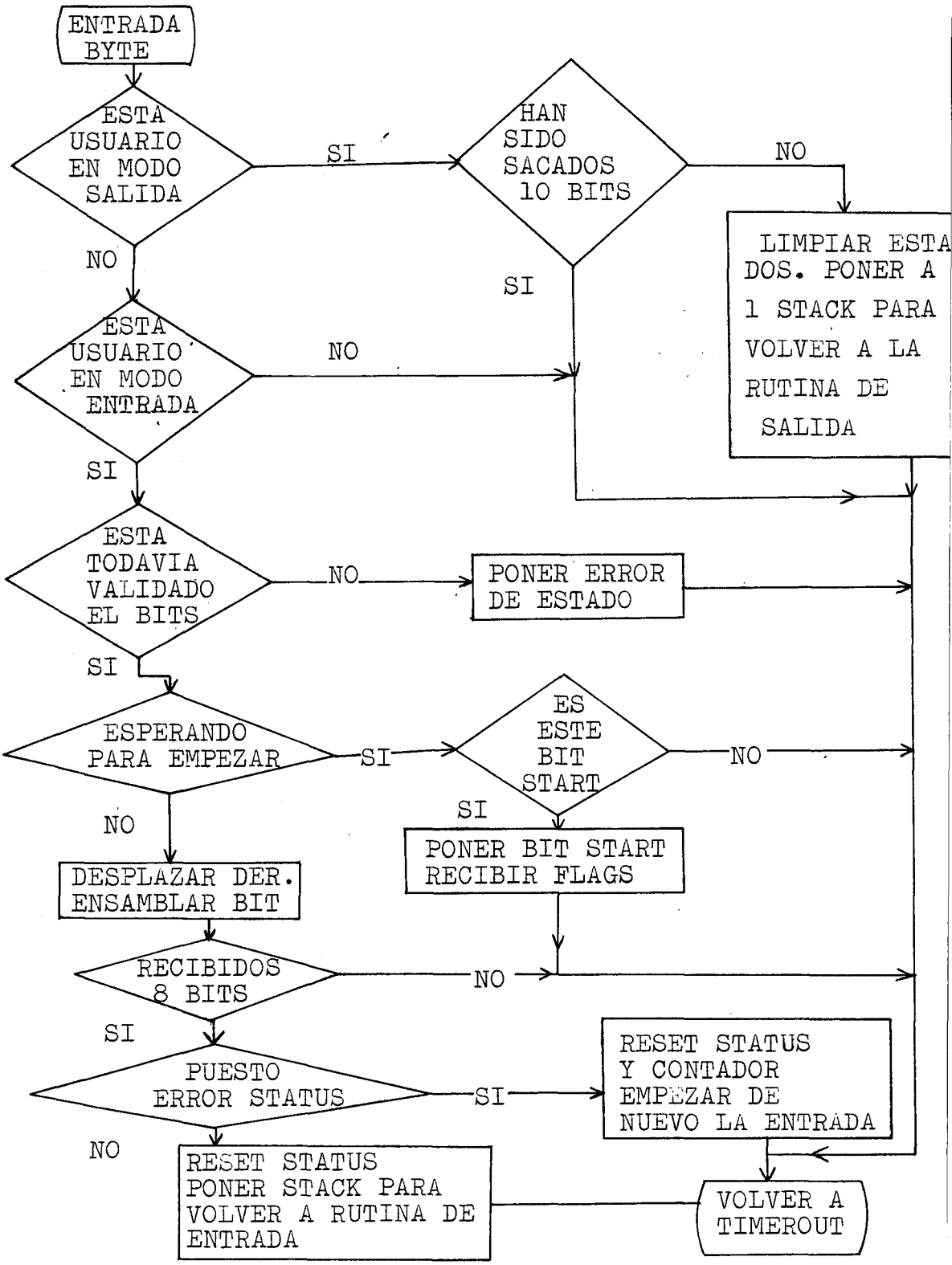
La rutina timer-out conmuta la actual tarea-usuario, procediendo a trabajar sobre la tarea que no ha sido recientemente procesada, al menos que el usuario esté todavía en modo de entrada o salida. Sí ese usuario está en un modo de E/S, el control vuelve a la tarea que estaba siendo ejecutada cuando la interrupción timer-out ocurrió.

Este proceso de conmutación permite a ambos usuarios ser servidos simultáneamente por el mismo procesador.

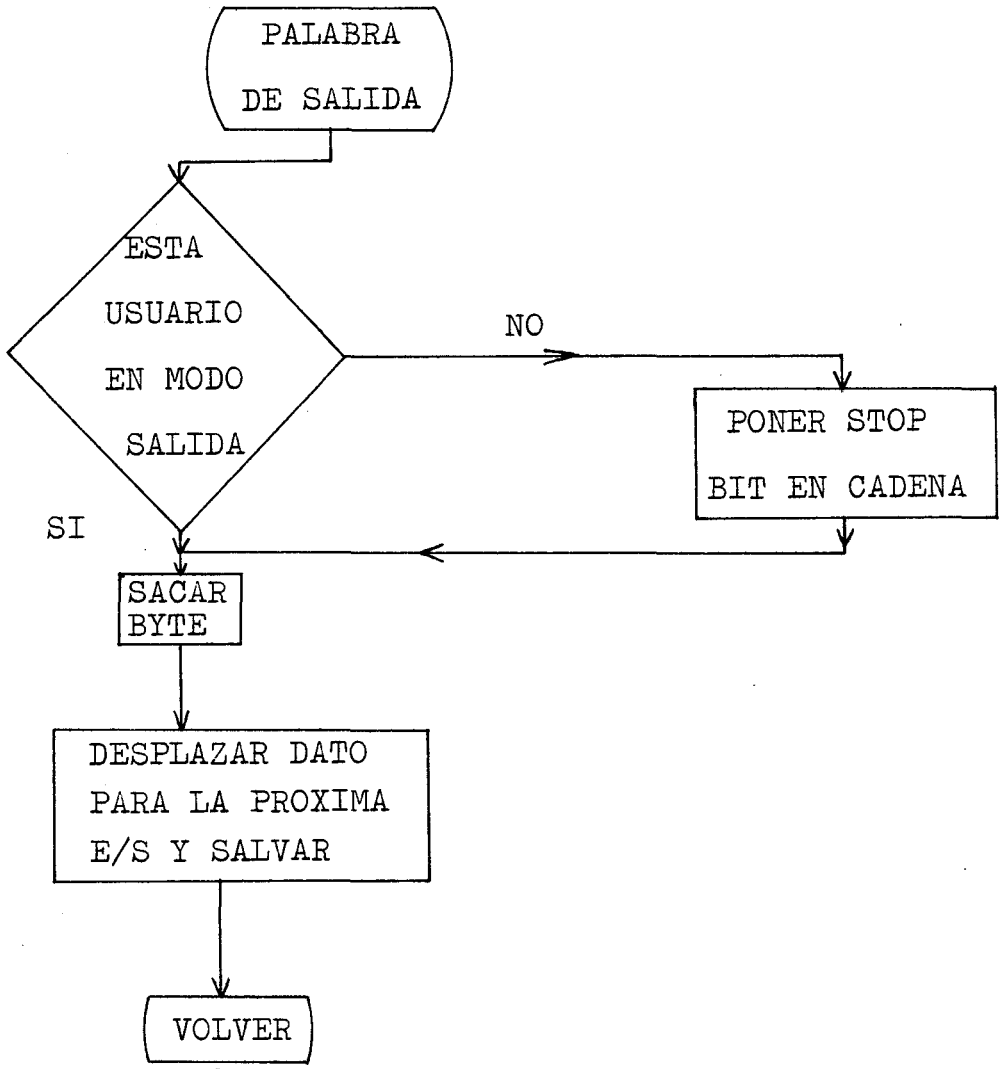
A continuación se muestra la rutina Timer-out:



A continuación se muestra la rutina para recibir entrada desde uno de los usuarios:



A continuación se muestra la rutina para enviar salida desde uno de los usuarios:



Código multitarea que permite a dos usuarios ser servidos por el mismo procesador, aparentemente simultáneamente. Cuando un usuario no requiere servicio, el procesador ejecuta un loop.

Cuando una operación debe ser llevada a cabo, esta rutina debe supervisar el proceso. Varias operaciones de E/S y el contador causan que este código sea entrado.

A continuación vemos el listado:

```
DIR.   CODIGO
HEX.   HEX.          ETIQ.   INST.OPER.      COMENTARIO
FE28   EB5990                JMP  USUAR
FE2B   EBFE          IORTI:  JMP  IORTI      ;Loop en si mismo
FE2D   AOADOO        CIRT:   MOV  AL,BYTEIN  ;Volver aquí
FE30   C3                    RET
FE31   50            CO÷     PUSH AX         ;Salvar registros
FE32   D1E0                  SAL  AX,1       ;Desplazar izquierda
FE34   0D000F                OR   AX,OFOOH   ;o poner stop
FE37   A3AB00                MOV  WORDOT,AX  .
FE3A   C606AA000090          MOV  OUTCYC,0   ;Reset ciclo salida
FE40   B001                  MOV  AL,1       ;Poner status a salida
FE42   EBD8                  JMP  COMP       ;Ver si necesita ir
FE44   58            CORT:   POP  AX
FE45   C3                    RET
```

```
FE46    E81801        TIMEOUT:  CALL  SVREG     ;Salvar registros
FE49    BA0200                  MOV   DX,INPORT
FE4C    EC                      IN    AL,DX     ;Entrada usuario 1
FE4D    8AE0                    MOV   AH,AL
FE4F    BA01F0                  MOV   DX,INPRT2
FE52    EC                      IN    AL,DX     ;Entrada usuario 2
FE53    50                      PUSH  AX        ;Salvar para futuro
                                                 usuario
FE54    8BC8                    MOV   CX,AX     ;Entrada dato, salvar
FE56    BA0100                  MOV   DX,OUTPORT  ;Poner salida,
                                                   usuario 1
FE59    E85700                  CALL  OUTWORD     ;Salida
FE5C    8926A700                MOV   STACKP,SP   ;Próximo bit o
                                                   bit stop
FE60    BA00F0                  MOV   DX,OUTPT2
FE63    B80800                  MOV   AX,00008H
FE66    8ED0                    MOV   SS,AX     ;Poner segmentos para
                                                 usuario 2
FE68    B82000                  MOV   AX,20H
FE6B    8ED8                    MOV   DS,AX
FE6D    8B26A700                MOV   SP,STACKP
FE71    E83F00                  CALL  OUTWORD ;Próximo bit salida
                                              o bit chip
FE74    BA01F0                  MOV   DX,INPRT2
FE77    E84E00                  CALL  INBYTE  ;Proceso E/S,usuario 2
```

```
FE7A    59                  POP CX          ;Restaurar entradas
FE7B    8ACD                MOV CL,CH
FE7D    BA0200              MOV DX,INPORT   ;Proceso E/S usuario 2
FE80    E84500              CALL INBYTE
FE83    A0AE00      USUAR:  MOV AL,STATUS   ;Determinar cual usua-
                                             rio a restaurar
FE86    2403                AND AL,03H
FE88    7406                JZ  CKU2        ;Si UI IN CO OR CI
FE8A    B80800              MOV AX,00008H   ;UI IN CO OR CI
FE8D    EB0F90              JMP PRETI       ;Ir a U2
FE90    A0AE02      CKU2:   MOV AL,STATS2
FE93    2403                AND AL,03H
FE95    7509                JNZ PRET        ;Usuario 2 IN CO OR CI
                                             volver a UI
FE97    36A10C00    SWUS:   MOV AX,SS:STACKS
FE9B    350800              XOR AX,0008H    ;Conmutar usuarios
FE9E    8ED0        PRETI:  MOV SS,AX
FEA0    D1E0                SAL AX,1
FEA2    D1E0                MOV SP,STACKP
FEA4    8ED8                SAL AX,1
FEA6    8B26A700            MOV DS,AX       ;Poner STACK y segmento
                                             dato
FEAA    07                  POP ES
FEAB    5D                  POP BP          ; Restaurar registros
FEAC    5F                  POP DI
```

```
FEAD    5E                      POP SI
FEAE    5A                      POP DX
FEAF    59                      POP CX
FEB0    5B                      POP BX
FEB1    58                      POP AX
FEB2    CF                      IRET            ;Volver al lugar donde
                                                 se interrumpió
FEB3    A1AB00      OUTWORD:    MOV AX,WORDDOT  ;Cargar palabra salida
FEB6    8A1EAE00                MOV BL,STATUS   ;Cargar byte status
FEBA    80CBFE                  OR  BL,0FEH
FEBD    F6D3                    NOT BL          ;Hacer BL=00 si IN,
                                                 CO OR 01 es NOT
FEBF    0AC3                    OR AL,BL
FEC1    EE                      OUT DX,AL       ;Salida byte
FEC2    D1F8                    SAR AX,1        ;Desplazar para próximo
                                                 bit
FEC4    A3AB00                  MOV WORDDOT,AX  ;y salvar palabra para
                                                 próximo ciclo
FEC7    C3                      RET
FEC8    8A1EAE00    INBYTE:     MOV BL,STATUS
FECC    8AFB                    MOV BH,BL       ;Ver si usuario en
                                                 modo salida
FECE    80E301                  AND BL,01H
FED1    7431                    JZ CKIN         ;No, ir a CKIN
FED3    FE06AA00                INC OUTCYC      ;En modo salida,incre-
                                                 mentar bits salida
```

```
FED7    803EAA000A          CMP OUTCYC,10    ;Sacar 10 bits?
FEDC    7515                JNE BRET         ;No volver
FEDE    C606AE000090        MOV STATUS,00H   ;si,Reset status y
FEE4    BB44FE              MOV BX,OFFSET    ;Poner regreso
                                (CGROUP:CORT)
FEE7    8926A700    RSST:   MOV STACKP,SP    ;Para cambiar salida o
                                              entrada
FEEB    83C414              ADD SP,20
FEEE    53                  PUSH BX
FEEF    8B26A700            MOV SP,STACKP
FEF3    59          BRET:   POP CX
FEF4    8926A700            MOV STACKP,SP    ;Poner registros para
                                              usuario 1
FEF8    33C0                XOR  AX,AX
FEFA    8ED0                MOV  SS,AX
FEFC    8ED8                MOV  DS,AX
FEFE    8B26A700            MOV SP,STACKP
FF02    51                  PUSH CX
FF03    C3                  RET
FF04    8ADF        CKIN:   MOV BL,BH        ;Ver si IN modo entrad
FF06    80E302              AND BL,02H       ;Si no, volver
FF09    74E8                JZ  BRET
FF0B    EC                  IN AL,DX         ;Entrada de nuevo y
                                              verificar dato valida
FF0C    3AC1                CMP AL,CL        ;Validado
```

```
FF0E    7408                JZ  BITGD       ;Si, bit es bueno
FF10    800EAE008090        OR STATUS,80H   ;No,bit error en status
FF16    EBDB                JMP BRET
FF18    80E180      BITGD:  AND CL,80H      ;Esperando por bit star
FF1B    80E704              AND BH,04H
FF1E    7435                JZ  WAITST      ;Si, ir a WAITST
FF20    A0AD00              MOV AL,BYTEIN
FF23    D0E8                SHR AL,1        ;Desplazar una vez
                                             para nuevo bit
FF25    0AC1                OR AL,CL
FF27    A2AD00              MOV BYTEIN,AL   ;Salvar byte entrada
FF2A    FE06A900            INC INCYCL
FF2E    803EA90008          CMP INCYCL,8    ;Ver si 8 bits entrada
FF33    75BE                JNE BRET        ;No
FF35    C606A9000090        MOV INCYCL,0    ;Si contador reset de
                                             bits entrada
FF3B    A0AE00              MOV AL,STATUS
FF3E    2480                AND AL,80H      ;Ver bit
FF40    7408                JZ  DELFI       ; No, caracter bueno
FF42    C606AE000290        MOV STATUS,2    ;Unidad mala,reset
                                             status y
FF48    EBA9                JMP BRET
FF4A    C606AE000090 DELFI: MOV STATUS,0    ;Reset status
FF50    BB2DFE              MOV BX,OFFSET
                                 (CGROUP:CIRT)
```

```
FF53    EB92                    JMP RSST        ;Preparar para volver
FF55    0AC9        WAITST:     OR CL,CL        ;Ver si bit start entra
FF57    759A                    JNZ BRET        ;No todavía
FF59    C606AE000690            MOV STATUS,06H  ;Si, reset status
FF5F    EB92                    JMP BRFT
FF61    891E0000    SVREG:      MOV BL,BX
FF65    5B                      POP BX          ;Salvar registros
FF66    50                      PUSH AX
FF67    FF360000                PUSH BL
FF6B    51                      PUSH CX
FF6C    52                      PUSH DX
FF6D    56                      PUSH SI
FF6E    57                      PUSH DI
FF6F    55                      PUSH BP
FF70    06                      PUSH ES
FF71    8926A700                MOV STACKP,SP   ;y poner stack y dato
FF75    8CD1                    MOV CX,SS
FF77    33C0                    XOR AX,AX       ;Segmentos para usuario
FF79    8ED0                    MOV SS,AX
FF7B    8ED8                    MOV DS,AX
FF7D    8B26A700                MOV SP,STACKP
FF81    36890E0C00              MOV SS:STACKS,CX
FF86    53                      PUSH BX
FF87    C3                      RET
```

```
                        CODE ENDS
                        CONST2 SEGMENT
FFF0                    ORG OFFF0H
FFF0    B8                      MOV AX,DGROUP
FFF3    8ED8                    MOV DS,AX
FFF5    BC7F0090                MOV SP,OFFSET(STK)
FFF9    EA              DB      OEAH
FFFA    94FD            DW OFFSET INIT
FFFC    0000            DW 0
                        CONST2 ENDS

                        CONST  SEGMENT
0055    483F            HOW     DB  'H?',CR
0057    0D
0058    4F4B            OK      DB  'OK',CR
005A    0D
005B    573F            WHAT    DB  'W?',CR
005D    0D
005E    53              SORRY   DB  'S',CR
005F    0D
```

BIBLIOGRAFIA

- Introducción al microprocesador 8086/8088.
  Christopher L. Morgan - Mitchell Waite.
  Byte Books / McGraw-Hill. Ed.
- El microprocesador de 16 bits 8086/8088.
  A. B. Fontaine.
  Masson. Ed.
- Microprocesadores de 16 bits.
  Jose Mª. Angulo Usategui.
  Paraninfo. Ed.
- SDK-86. MCS-86 System Design Kit. User´s Guide.
  INTEL.
- The 8086 Family User´s Manual.
  INTEL.
- MCS-86. Product Description.
  INTEL.
- Microprocesadores. Fundamentos, diseño y aplicaciones en la industria y los microcomputadores.
  Jose Mª. Angulo Usategui.
  Paraninfo. Ed.