Introducción al Curso

INEL 4206 – Microprocesadores
Texto: Barry B Brey, The Intel Microprocessors: Architecture, Programming and Interfacing. 8va. Ed., Prentice Hall, 2009

Prof. José Navarro
Of. T-214 xt. 3097

E-mail: jose.navarrofigueroa@upr.edu
jnavarro@ece.uprm.edu

- Discusión del prontuario
- Qué espera el profesor de sus estudiantes
- Qué esperan los estudiantes de su profesor

Historia de los microprocesadores

Intel 4004:
- accedía 4K localizaciones de 4-bits
- 45 instrucciones
- 50 KIPs
- ENIAC – 100KIPs, 30 toneladas
- Peso 4004: menos de una onza

Intel 4040:
- Un 4004 más rápido
-

TI TMS-1000: Otro micro de 4 bits

Intel 8008 (1971):
- accedía a 16KB de memoria
- 48 instrucciones

Intel 8080 (1973):

- primer micro moderno de 8 bits
- 64 KB
- 500 KIPs
- compatible con lógica TTL

Motorola MC6800:

- micro de 8 bits
- lanzado 6 meses después del 8080

MITS 8800 (1974):

- primera computadora personal
- basada en el 8080
- interpretador de BASIC por B. Gates

Intel 8085 (1977):

- actualización del 8080
- 769,230 inst/s
- 246 instrucciones
- generador de clock y controlador del sistema interno

Intel 8086 (1978), 8088 (1979):

- 16 bits y acceso a 1MB
- 4 a 6 bites de memoria cache
- set de instrucciones más grande
- microprocesador tipo CISC

En 1981 IBM decidió utilizar al 8088 en sus computadoras personales, asegurando así la supremacía de los procesadores de Intel.

Intel 80286 (1983):

- set de instrucciones del 8086,
- 4MIPS (8MHz)
- acceso a 16MB

Intel 80386 (1986):

- Procesador de 32 bits
- hasta 4GB de memoria

Intel 80486 (1989):

- 80386 + cop aritmético + 8KB cache
- 50MHz
- versiones 66MHz/33MHz (double-
- clocked) y 100MHz/33MHz (triple-
- clocked).
- versiones con un cache de 16KB

AMD

- Versión (triple-clocked) con
- 120MHz/40MHz

Intel Pentium (1993):

- 8KB cache para insts y 8KB para data
- 66MHz, versiones overclocked hasta 233MHz
- 4GB de memoria
- transferencia de data de 64bits
- dos procesadores de enteros internos
- tecnología para predecir saltos
- coprocesador aritmético interno 5 veces más rápido que el 80486.
- se hizo una versión MMX

Intel Pentium-Pro(1995):

- 3 unidades para enteros
- 1 para operaciones de punto flotante
- 150-166MHz
- 16KB L1 cache, 256KB L2 cache
- tres motores de ejecución para instrucciones que pueden confligir
- acceso 4GB de memoria
- pueden trabajar hasta 4 procesadores en el mismo sistema, orientado a los servidores.

Intel Pentium II (1997):

- Un board en lugar de un circuito integrado
- cache L2 en el board con bus a 133MHz (66MHz en el PII) y con 512KB. Es un Pentium Pro sin cache L2 interno y con extensiones MMX.
- En el 1998 la velocidad del bus 1e aumentó a 100MHz (de 66MHz) por que esta parte resultaba ser un cuello de botella para la ejecución.
- Velocidad interna subió hasta 450MHz
- La memoria se tuvo que mejorar a 8ns.

Intel Pentium II Xeon (1998):

- Cache L1 32KB, L2 de 512KB, 1MB o 2MB
- Orientado a sistemas de alto rendimiento

Intel Pentium III

- Hasta 1.3 GHz
- 133 MHz System Bus
- 70 instrucciones adicionales mayormente orientadas a mejorar las aplicaciones orientadas a internet
- 32KB L1 cache, 256 or 512KB L2 cache

Intel Pentium 4

- Hasta 1.8 GHz
- 400 MHz System Bus
- Enhanced floating point/multimedia
- Streaming extensions
- Hyper-pipelined technology

Intel Pentium 4

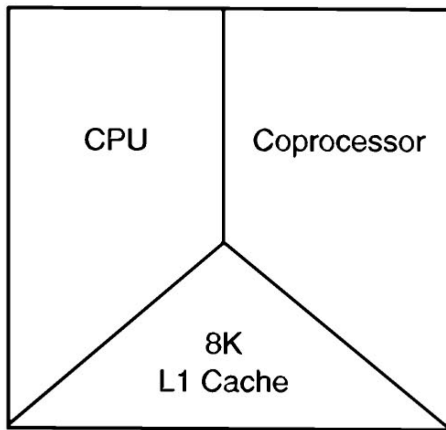
- Hasta 1.8 GHz
- 400 MHz System Bus
- Enhanced floating point/multimedia
- Streaming extensions
- Hyper-pipelined technology

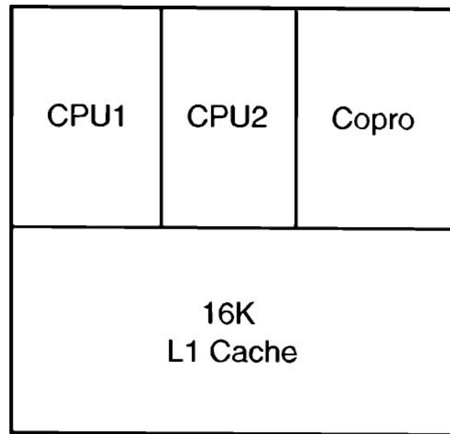
Nuevos Intel Pentium 4 y Core 2

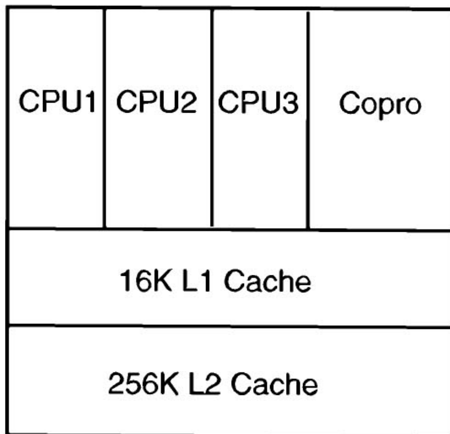
- Procesador de 64 bits
- 40 líneas de address (hasta 1T)
- Versiones Dual & Quad Core
- Se esperan versiones de 16 núcleos
- En 2007 se presenta prototipo de 80 núcleos (link)
    - Distinto set de instrucciones

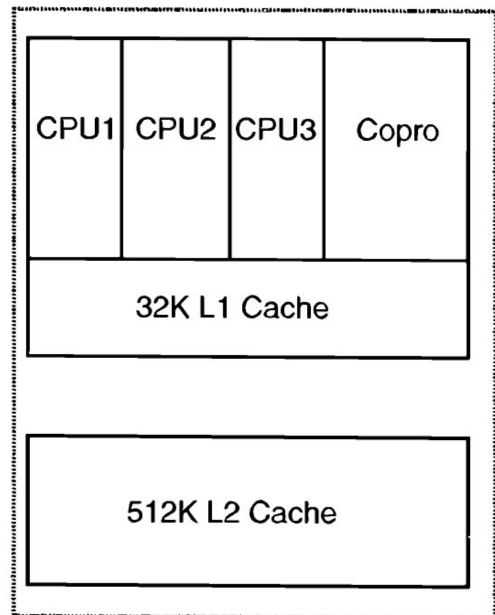
¿El Futuro?

- Hyperpipelines
- Multicores (¿uso eficiente?)
- Lowpower
- Wireless



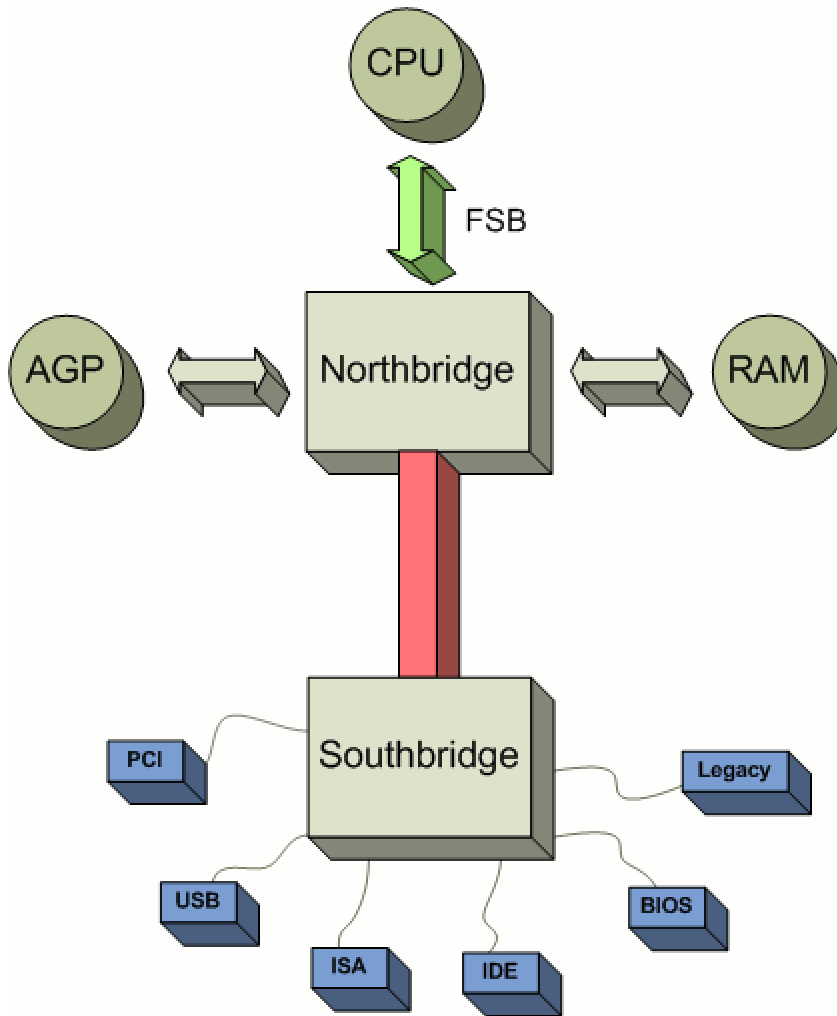






Brey, Figura 1.4, Págs 14-15

Chipsets



Autor: Alexander Taubenkorb (en wikipedia.com)

Componentes de un Sistema de Computadora

- el procesador
- memoria
- dispositivos externos de entrada y salida.

Buses

- Bus de direcciones
- Bus de data
- Bus de control

Memoria y Sistemas de Entrada y Salida

Memoria PC se divide en tres partes principales:

- TPA (Transient Program Area)
    - contiene al sistema operativo y otros programas que controlan la computadora.
    - 640KB, 12KB para OS y 628KB para aplicaciones

- System Area
    - Programas y data en RAM y ROM para control del sistema.

- XMS (Extended Memory System)
  - Sólo existe en los sistemas con del 80286 en adelante.

El TPA y el system area se conocen como la **memoria real** porque los microprocesadores de Intel están diseñados para trabajar con ellas en su modo real.

Los sistemas que utilizan la memoria extendida se conocen como **sistemas AT**

ISA: 8-bits peripheral bus en los primeros sistemas
ISA: 16-bits peripheral bus en los sistemas AT (80286) en adelante

EISA: 32-bits peripheral bus, sistemas 80386 y 80486, 8MHz.

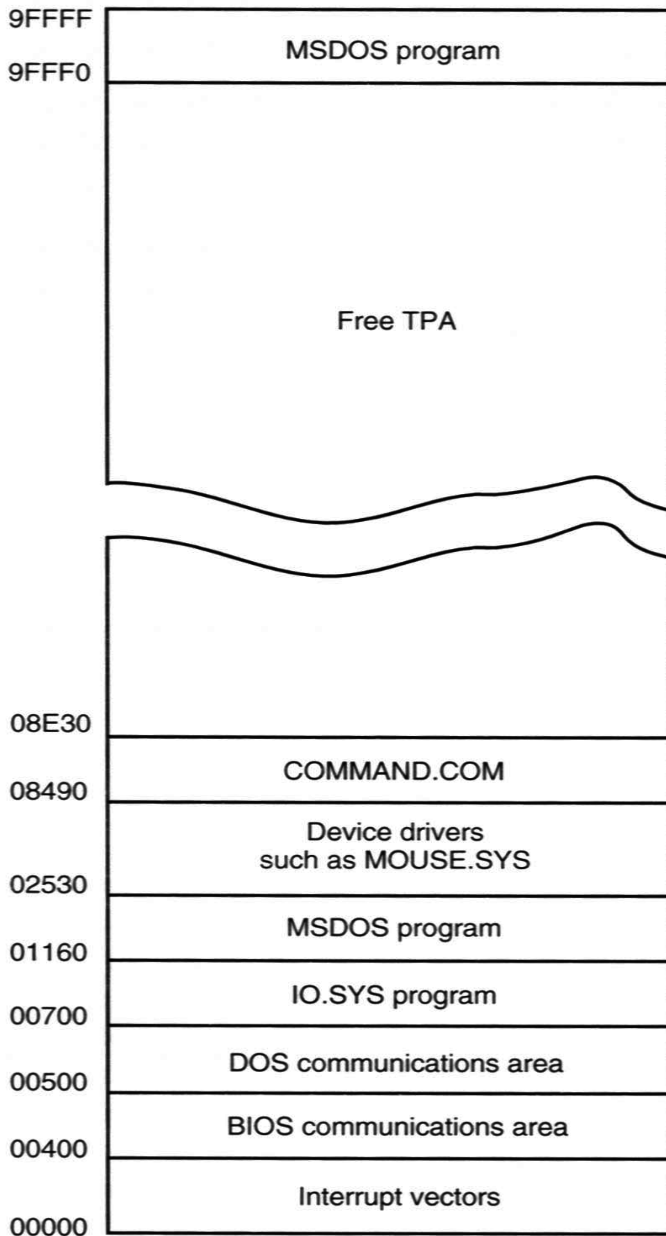
VESA Local Bus: 32-bits y conecta los discos y video al bus local para que se comuniquen a la velocidad de éste.

PCI: Diseñado para trabajar directamente con Pentium al Pentium II y se conectan al Bus local a una frecuencia de 33MHz con 32 o 64 bits.

USB (Universal Serial Bus): Periféricos a través de una conexión serial dando la ventaja de menos conexiones. Opera a 10Mbps y se espera que alcance 100Mbps.

AGP: Para tarjetas de video, 64bits a 66MHz

Distribución de la Memoria del TPA



Brey, Figura 1-7: Mapa de la memoria de un TPA en una computadora personal

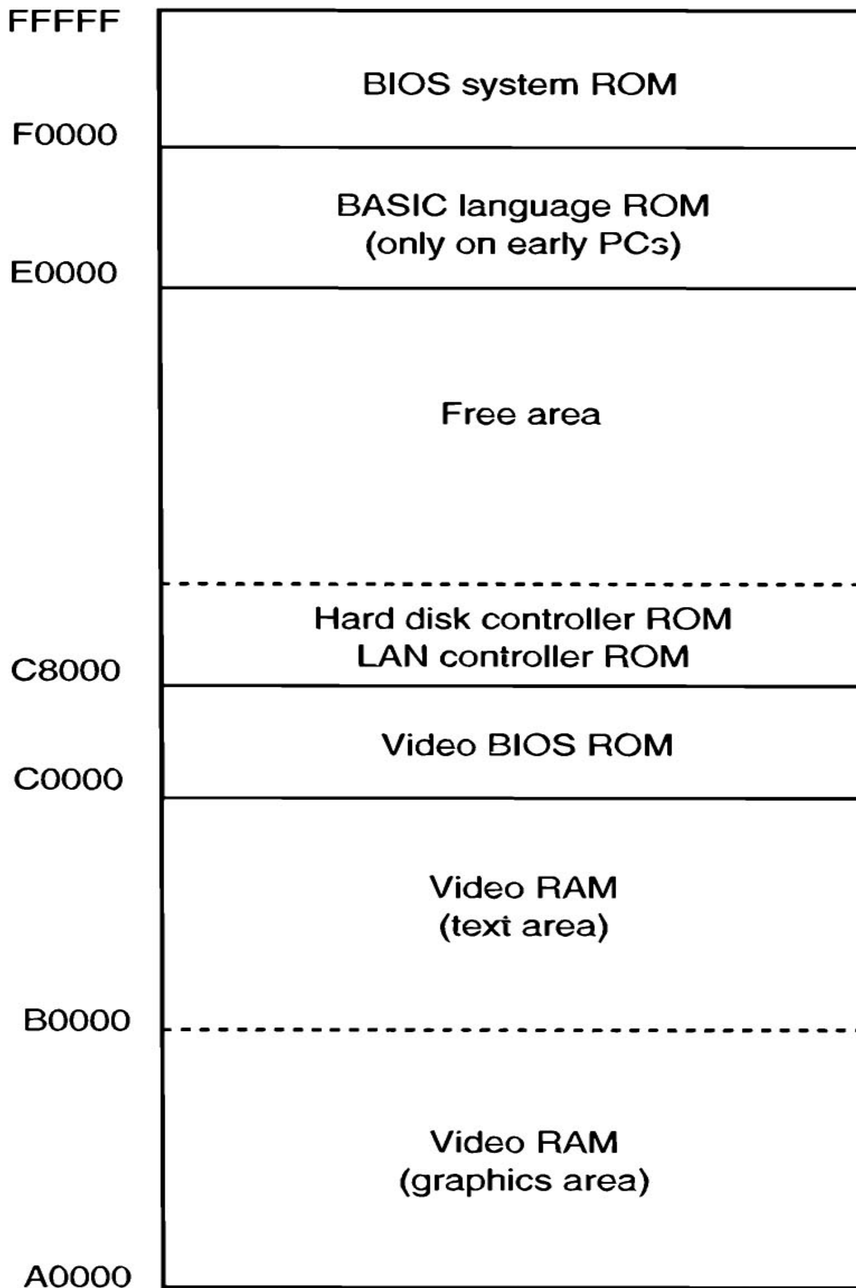
BIOS (Basic I/O System): Programas para manejar dispositivos conectados a la computadora. Se encuentra en ROM o EEPROM

Interrupt vectors: Direcciones para acceder rutinas del BIOS, DOS y aplicaciones

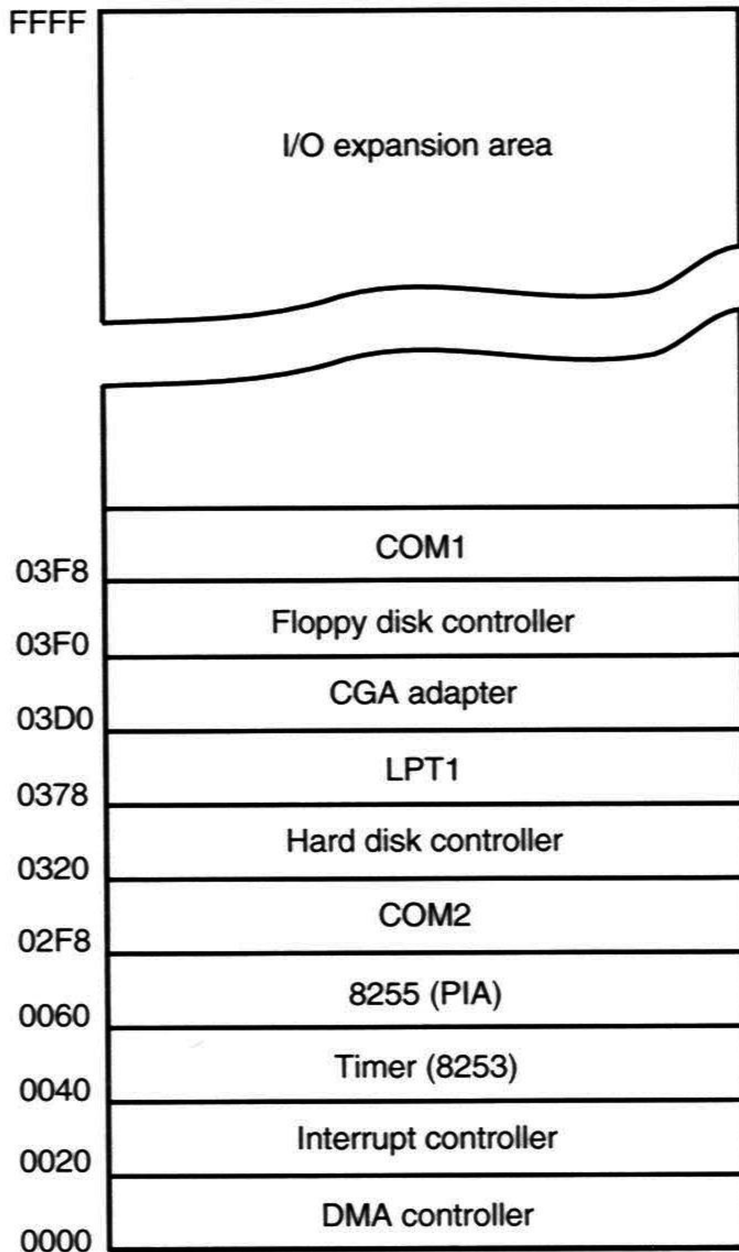
BIOS y DOS communication areas: Data para establecer comunicación con las rutinas del BIOS y DOS

IO.SYS: Contiene programas que le permite al sistema operativo comunicarse con dispositivos tales como el teclado, video, impresora y otros.

COMMAND.COM: Controla los comandos para el sistema operativo entrados por el teclado.



Brey, Figura 1-8.Área de sistema para una computadora personal típica



Brey, Figura 1.10: Mapa de I/O para una computadora personal mostrando algunas de las muchas áreas para los dispositivos de I/O.

El Microprocesador

El microprocesador o CPU (Central Processing Unit) se encarga de tres tareas principales en la computadora:

1. La transferencia de data entre él y la memoria y dispositivos de I/O
2. La realización de tareas simples de aritmética y lógica
3. Control del flujo de los programas mediante la toma de decisiones

Operaciones aritméticas y lógicas simples

Operación	**Comentarios**
Suma	Con y sin signo
Resta	Con y sin signo
Multiplicación	Con y sin signo
División	Con y sin signo
And	
Or	
Not	
Neg	
Shift Rotate	

Algunos Tipos de Condiciones que Permiten Analizar los Procesadores

- Zero: Verificar si un número es o no cero

- Sign: Verificar sin un número es negativo o no

- Carry: Verificar si una operación de suma produce un excedente o si una resta requiere “tomar prestado”

-Oveflow: Verificar si el resultado de una operación produce un valor con el signo equivocado

-Paridad: Verificar si el total de unos en un valor es par o impar

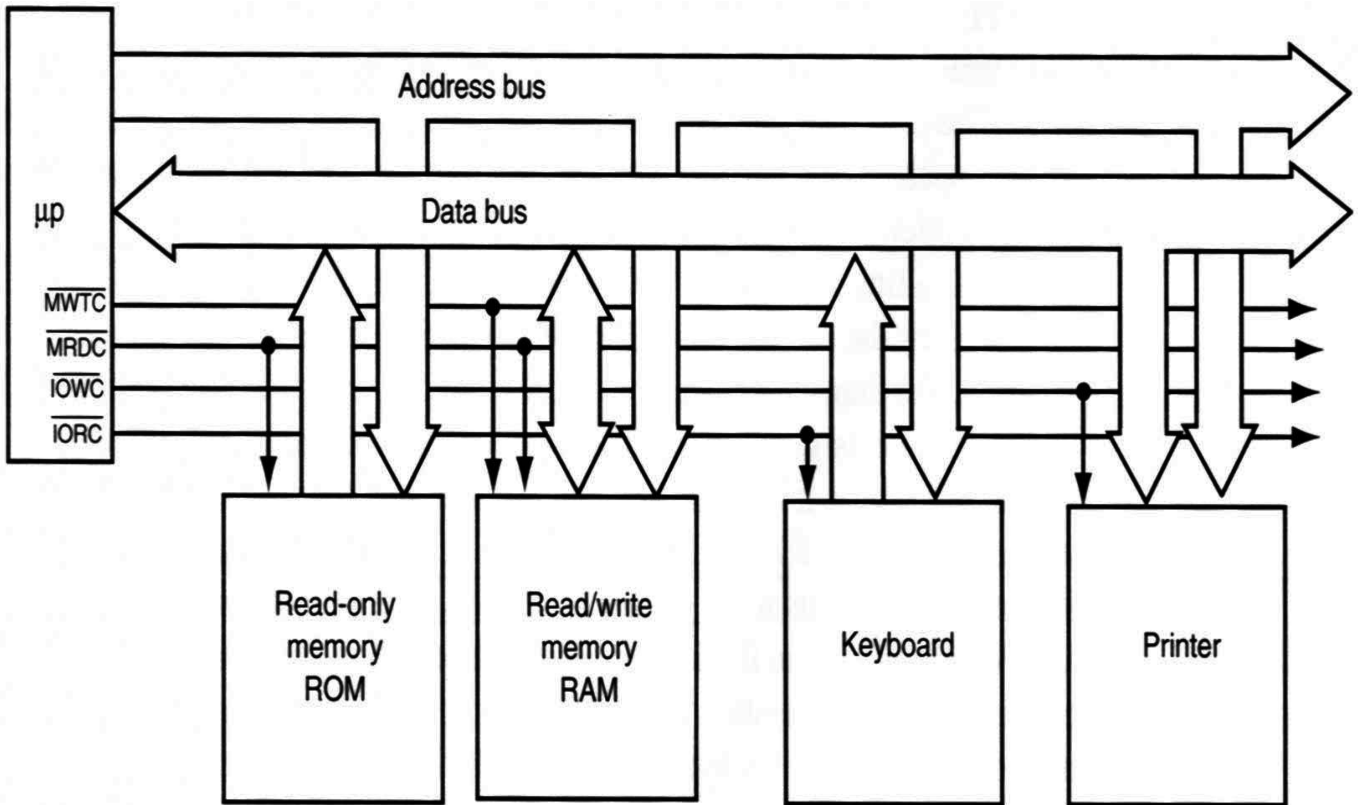
Buses

Un bus es un grupo de líneas de comunicación de un mismo tipo que conectan a los componentes de una computadora. Se dividen en

- **data bus**: por donde transita la data

- **address bus**: por donde se envían las direcciones de donde se quieren recibir o enviar valores

- **control bus**: señales para mantener la sincronización y armonía entre las actividades del sistema



Brey, Figura 1-10. Diagrama de bloques de un sistema de computadora mostrando los buses de direccionamiento, data y control

Brey, Tabla 1-11. Tamaño de los buses y memoria para los microprocesadores de la familia Intel

Microprocessor	*Data Bus Width*	*Address Bus Width*	*Memory Size*
8086	16	20	1M
8088	8	20	1M
80186	16	20	1M
80188	8	20	1M
80286	16	24	16M
80386SX	16	24	16M
80386DX	32	32	4G
80386EX	16	26	64M
80486	32	32	4G
Pentium	64	32	4G
Pentium OverDriv	32	32	4G
Pentium Pro	64	32	4G
Pentium Pro	64	36	64G
Pentium II	64	32	4G
Pentium II	64	36	64G

Notas sobre Sistemas Numéricos Utilizados en las Computadoras

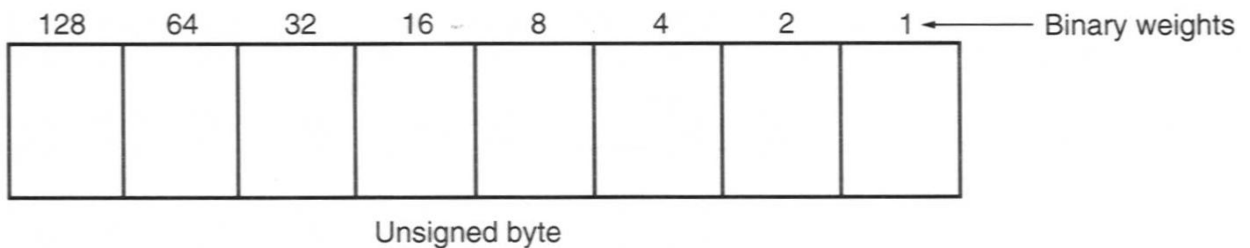
Sistema BCD: En el sistema BCD (Binary Coded Decimal) se utilizan grupos de cuatro bits para representar los dígitos decimales (0-9). Existen dos versiones del formato BCD.

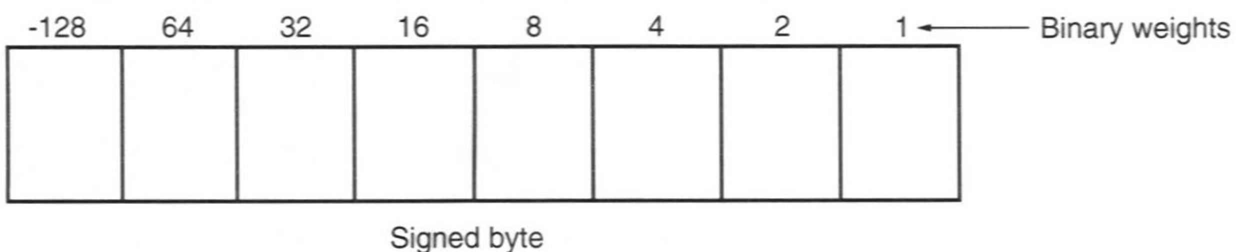
packed BCD: En este formato se almacenan dos dígitos BCD en un BYTE. Por ejemplo, el número 25 decimal se representa 00100101, en donde los primeros cuatro bits (0010) del bite representan el 2 y los otros cuatro (0101) el 5.

unpacked BCD: En este formato se utiliza un bite para cada uno de los dígitos decimales. Pr ejemplo, el número 25 decimal se representa 00000010 00000101.

Números Enteros sin Signo y con Signo

Los microprocesadores manejan tanto números sin signo como números con signo. Para los números sin signo se utiliza la representación binaria corriente. Para representar los valores negativos se utiliza la notación de complemento de 2.





Brey, Figura 1-13. Pesos para las posiciones de los bits para bytes sin signo y con signo

Representación de Números Reales o de Punto Flotante

- Requiere dos partes: la mantisa y el exponente
- El bit más significativo representa el signo

Formato de precisión sencilla.

- 8 bits para el exponente (del 23 al 30)
- 24 bits para la mantisa (del 0 al 22)
  - Físicamente son 23 porque el primer bit es un 1 implícito (forma normalizada)

- Los exponentes se representa utilizando la forma “biased exponent”. En este formato se le suma 127 al exponente en caso de que se esté usando precisión sencilla y 1023 si se está utilizando precisión doble.

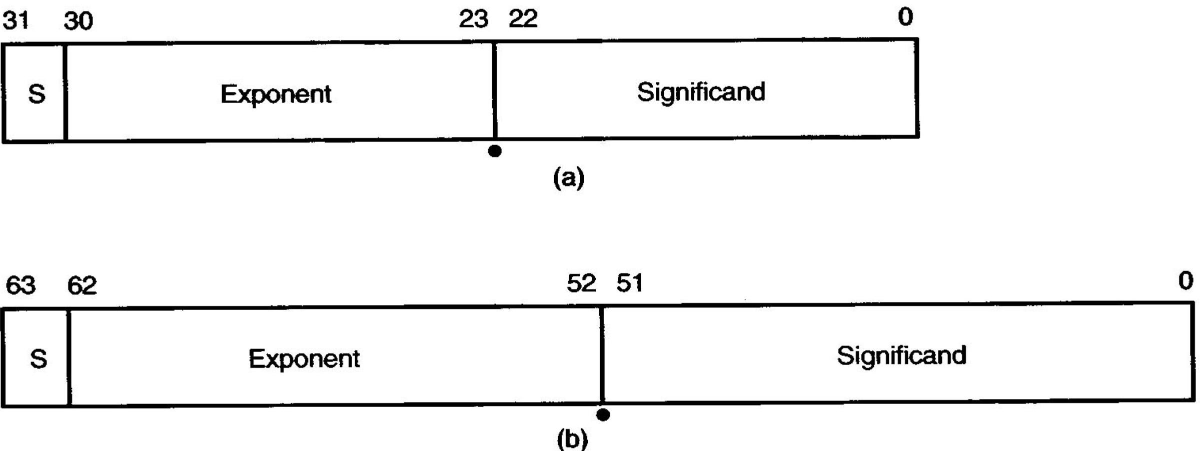
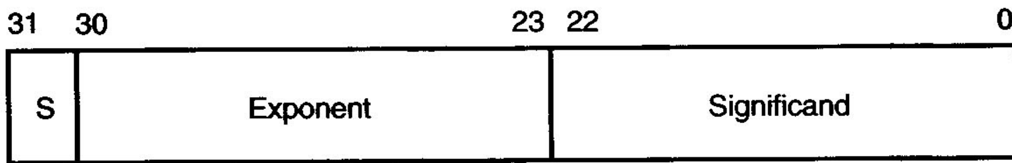
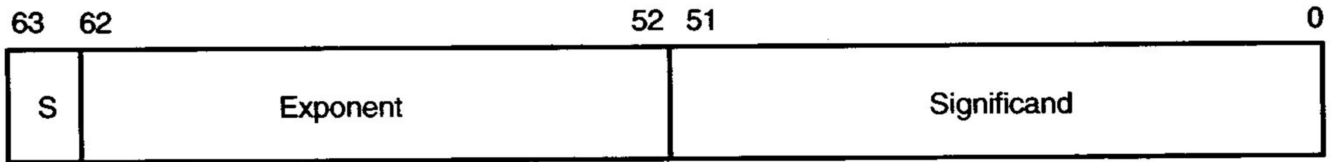


FIGURE 1–15 The floating-point numbers (a) single-precision using a bias of 7FH and (b) double-precision using a bias of 3FFH.

TABLE 1–10 Single-precision real numbers

Decimal	*Binary*	*Normalized*	*Sign*	*Biased Exponent*	*Mantissa*
+12	1100	1.1×2^3	0	10000010	1000000 00000000 00000000
−12	1100	-1.1×2^3	1	10000010	1000000 00000000 00000000
+100	1100100	1.1001×2^6	0	10000101	1001000 00000000 00000000
−1.75	1.11	-1.11×2^0	1	01111111	1100000 00000000 00000000
+0.25	.01	1.0×2^{-2}	0	01111101	0000000 00000000 00000000
+0.0	0	0	0	00000000	0000000 00000000 00000000





Brey, Figura 1-16. Formato para la representación de números con Punto flotante. (a) precisión sencilla, (b) precisión doble

Brey, Tabla 1-10. Ejemplos de números reales en precisión sencilla

Decimal	*Binary*	*Normalized*	*Sign*	*Biased Exponent*	*Mantissa*
+12	1100	1.1×2^{3}	0	10000010	1000000 00000000 00000000
−12	1100	-1.1×2^{3}	1	10000010	1000000 00000000 00000000
+100	1100100	1.1001×2^{6}	0	10000101	1001000 00000000 00000000
−1.75	1.11	-1.11×2^{0}	1	01111111	1100000 00000000 00000000
+0.25	.01	1.0×2^{-2}	0	01111101	0000000 00000000 00000000
+0.0	0	0	0	00000000	0000000 00000000 00000000

Existen dos casos especiales al utilizar este formato (que es el mismo adoptado por la IEEE, IEEE-754 version 10.0). El cero es almacenado con todos los bits en 0; infinito es almacenado con todos 1s en el exponente y 0s en la mantisa. El signo se utiliza para identificar si estamos trabajando con más infinito o menos infinito.

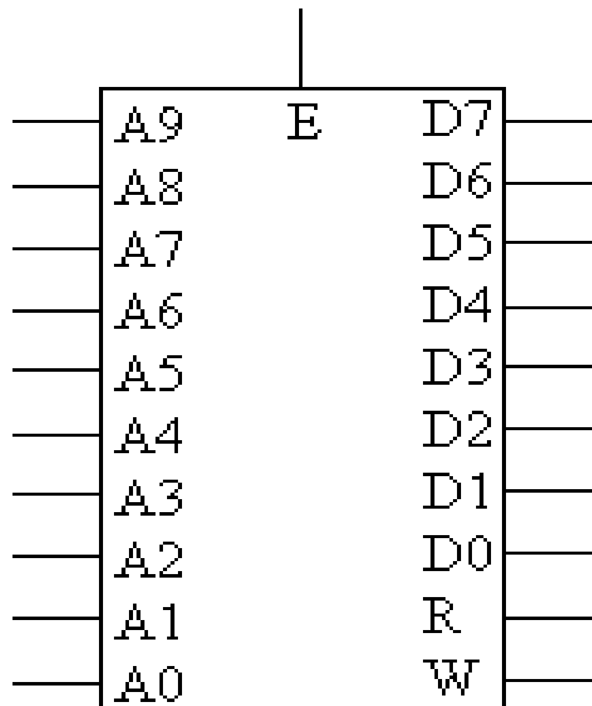
Introducción a la Organización de Computadoras

Características de la Memorias

- **tamaño de palabra**
- **cantidad de localizaciones**

$$totaLocalizaciones = 2^{totalLineasDeDireccionamiento}$$

- **líneas de data**
- **líneas de dirección**
- **líneas de control**



Microprocesadores

Operaciones de lectura y escritura

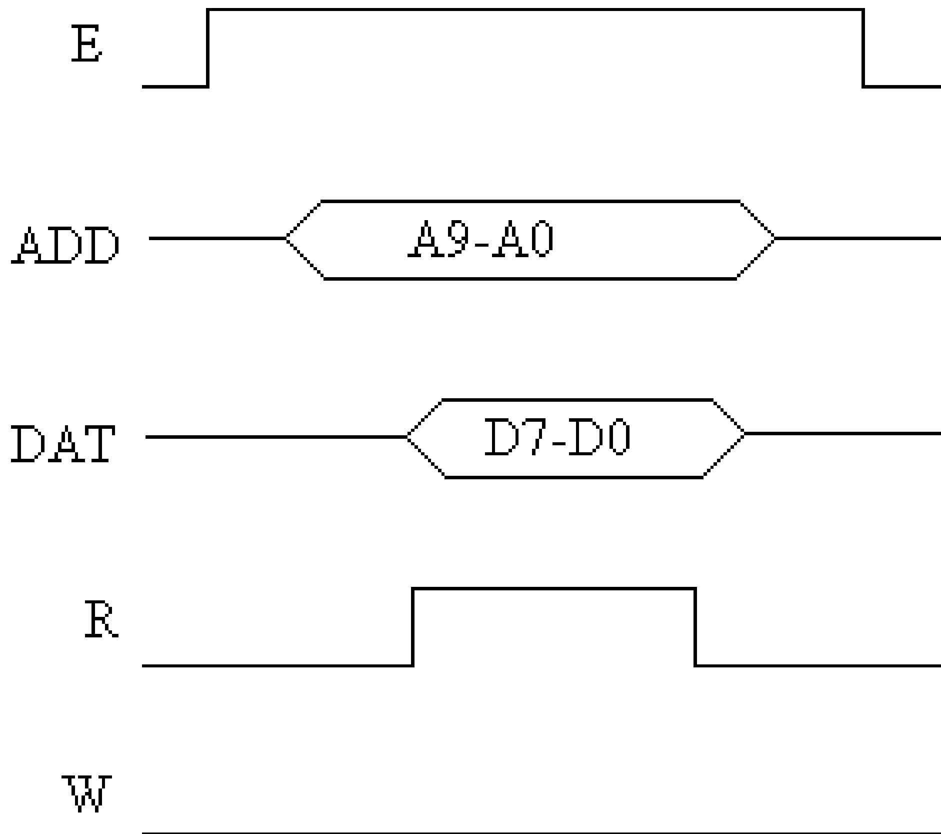


Figura 2. Operación de lectura de una memoria

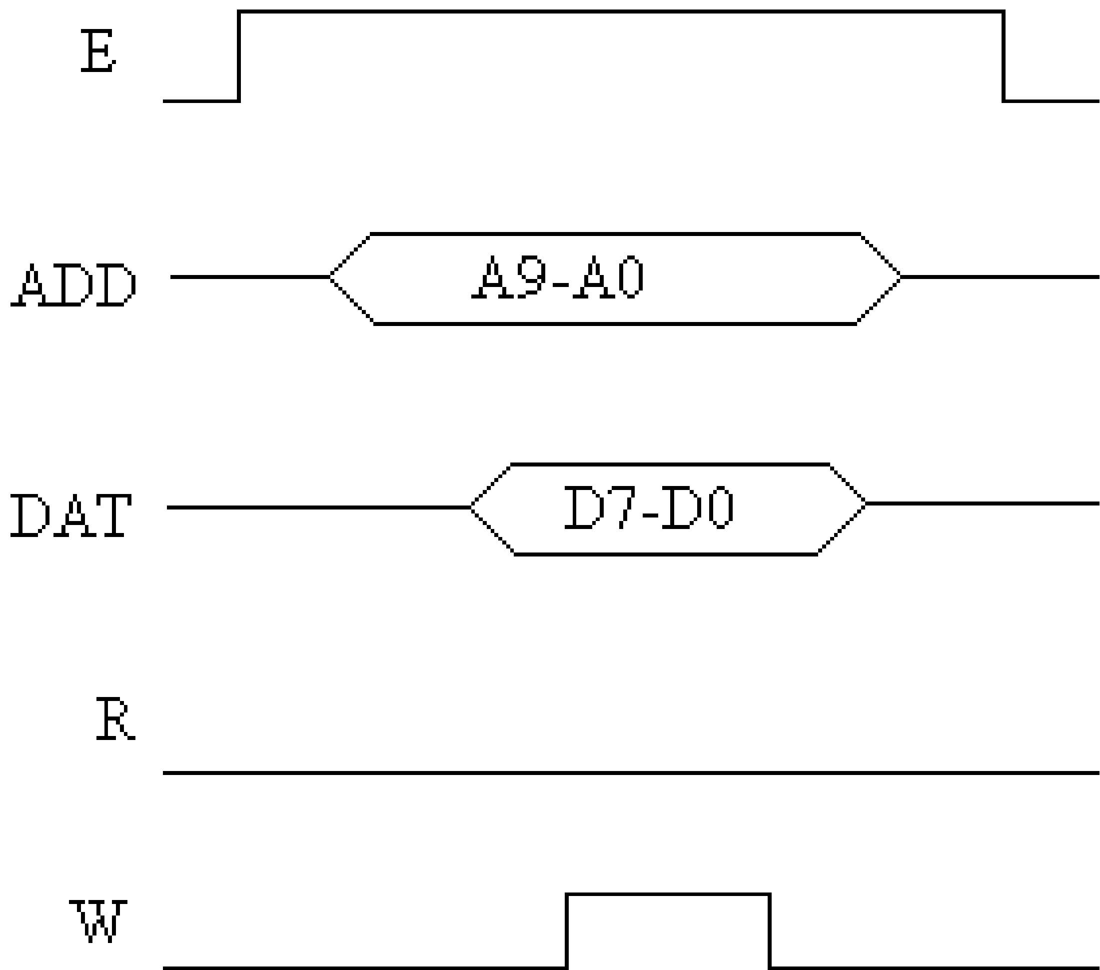


Figura 3. Diagrama de tiempo para una operación de escritura

Tipos de memoria

Volátiles y Permanentes

RAM – Random Access Memory
ROM – Read Only Memory
PROM – Programmable ROM
EPROM – Erasable PROM
EEPROM – Electrically EPROM

Arquitectura del Microprocesador

Registros de Propósito General o Registros de Data:

AX: (acumulador) El procesador está diseñado para favorecer las operaciones aritméticas que se realizan con este registro. De hecho, algunas de las operaciones de este tipo sólo se pueden realizar utilizando este registro.

BX: (base) Se puede utilizar para almacenar la localización en memoria de un valor o instrucción y hacer referencia a la misma utilizando el registro.

CX: (contador) Varias instrucciones del procesador que implementan ciclos utilizan este registro como el contador que controla los mismos.

DX: (data) Juega un papel especial en las operaciones de multiplicación y división que requieren específicamente su uso

Registros de Segmento

CS: (code segment) Indica la localización de memoria en donde comienza el código del programa.

DS: (data segment) Indica la localización de memoria en donde comienza la data del programa.

SS: (stack segment) Indica la localización de memoria en donde comienza el stack.

ES: (extra segment) Similar al DS se utiliza para señalar la localización de un área adicional para data.

FS, GS: Registros de segmentos adicionales. Se añadieron a partir del 80386

Registros de Índice

BP: (Base pointer) Se puede utilizar para representar una distancia relativa con respecto al SS (base) de forma similar a como se utiliza un apuntador a un stack.

SP: (stack pointer) Contiene una localización relativa al tope del stack. Se utiliza en conjunto con SS para indicar el tope del stack.

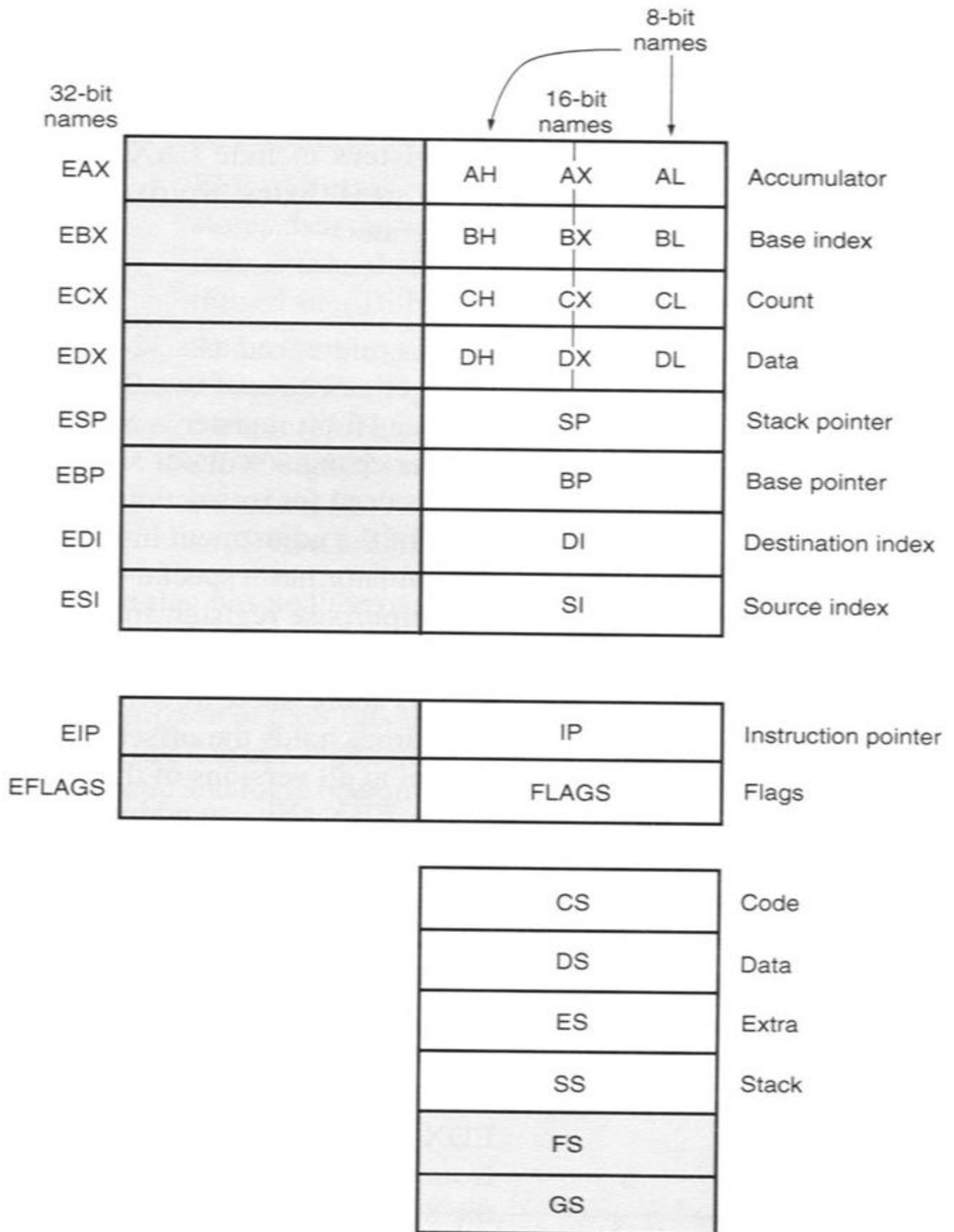
SI: (source index) Se utiliza como un índice para manejar arreglos.

DI: (destination index) Se utiliza como un índice para manejar arreglos.

Registros de Estatus y de Control:

IP: (instruction pointer) Indica, en conjunto con CS, la localización en donde se encuentra la próxima instrucción que se ejecutará.

Flags: (program status word) Contiene un conjunto de bits que indica que condiciones se dieron durante la ejecución de una instrucción


8-bit names
32-bit names
16-bit names
EAX
AH
AX
AL
Accumulator
EBX
BH
BX
BL
Base index
ECX
CH
CX
CL
Count
EDX
DH
DX
DL
Data
ESP
SP
Stack pointer
EBP
BP
Base pointer
EDI
DI
Destination index
ESI
SI
Source index
EIP
IP
Instruction pointer
EFLAGS
FLAGS
Flags
CS
Code
DS
Data
ES
Extra
SS
Stack
FS
GS

Microprocesadores

Flags en los procesadores 80x86 y Pentiums

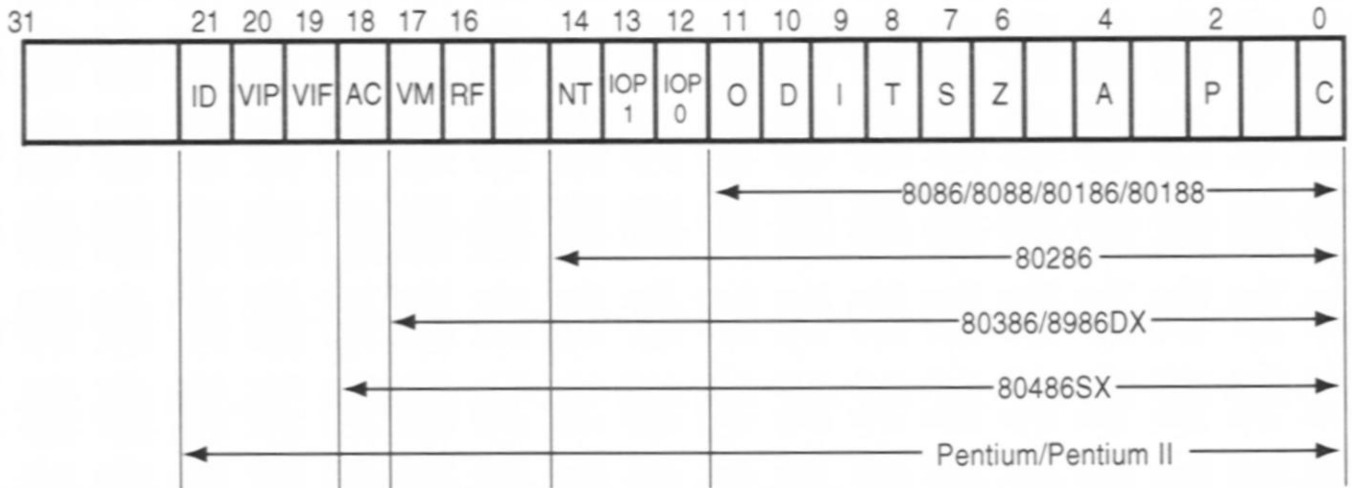


Figura 2-2: Registros EFLAG y FLAG

Flags 8086, 8088, 80186, 80188

- C : carry
- P : parity
- A : auxiliary carry
- Z : zero
- S : sign
- T : trap
- I : interrupt

D : direction

O : overflow

Direccionamiento de Memoria en Modo Real

Dirección_física =
contenido_registro_segmento * 16 +
desplazamiento

Nota: Los segmentos son se 64KB

Uso por Defecto de los Registros de Segmento

CS: para acceder las instrucciones (junto a IP)
DS: para acceder la data, manejo de strings (junto a SI)
SS: para acceder el stack (junto a SP y BP)
ES: para manejo de strings (junto a DI)
FS, GS: para uso general

Nota: El uso de las extensiones de 32 bits es similar

Direccionamiento de Memoria en Modo Protegido

- 80286 en adelante
- Permite acceder la memoria más allá de 1MB
- Se eliminó el uso de los segmentos de la forma en que se hace en el modo real.
- Los registros de segmento contiene un "selector" que apunta a una entrada en una tabla de descriptores.
- En el modo protegido se tienen dos tablas de descriptores con 8192 descriptores cada una.

80286 descriptor

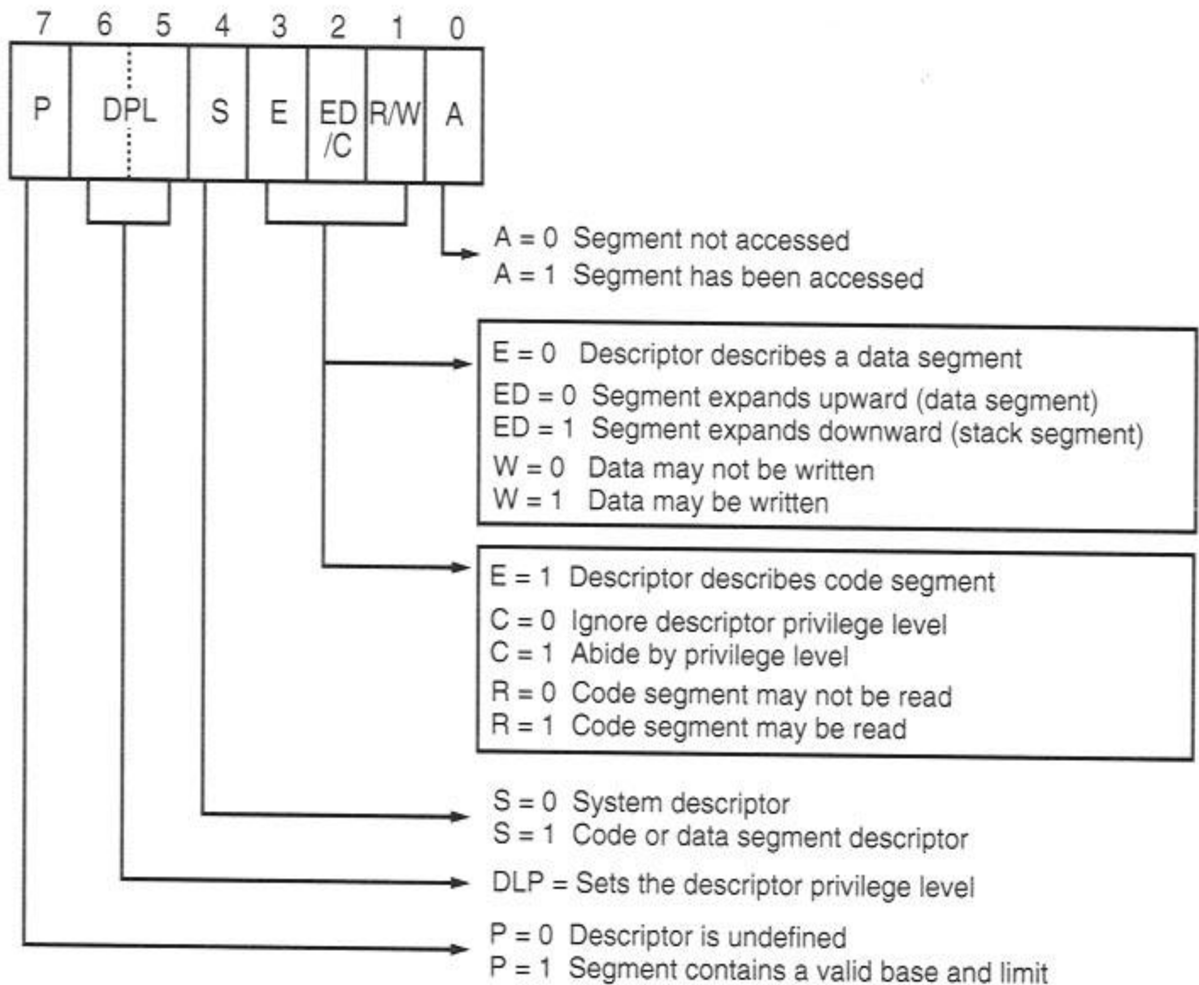
7	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	6
5	Access rights	Base (B23–B16)	4
3	Base (B15–B0)		2
1	Limit (L15–L0)		0

80386/80486/Pentium/Pentium Pro/Pentium II descriptor

7	Base (B31–B24)	G	D	0	AV	Limit (L19–L16)	6
5	Access rights					Base (B23–B16)	4
3	Base (B15–B0)						2
1	Limit (L15–L0)						0

En el descriptor se indica la base del segmento con 24 bits en el 80286 y 32 bits del 80386 en adelante. Para el 80286 se utilizan 16 bits para indicar el límite o longitud del segmento por lo que se limitan los segmentos a 64KB. El límite para los procesadores del 80386 en adelante utiliza 20 bits. Estos 20 bits pueden proveer un tamaño máximo para el segmento de 1MB. Sin embargo el descriptor provee el bit G que puede modificar el tamaño del segmento. Si el bit G = 0 el valor del límite se multiplica por 1, si G = 1 entonces el valor del límite se multiplica por 4K lo que permite tener una tamaño máximo de 4GB. Este bit se conoce como el bit de granularidad.

El descriptor provee un bit (AV) para indica si el segmento está o no disponible. El bit D se utiliza para indicar si por defecto se tratarán las instrucciones como si fueran de 16 bits o de 32 bits (se puede modificar la forma en que se tratan por medio de programación). El descriptor provee además de un bite para definir los derechos de acceso que se resume en la figura 2-7.



Note: Some of the letters used to describe the bits in the access rights bytes vary in Intel documentation.

FIGURE 2–7 The access rights byte for the 80286, 80386, 80486, Pentium, Pentium II, and Pentium Pro descriptor.

Modos de direccionamiento para la data

Los modos de direccionamiento son las alternativas que nos provee el lenguaje para acceder la data o instrucciones

El formato de la instrucción *mov* es el siguiente:

mov dst,src

Direccionamiento por registro

mov ax,cx

Direccionamiento inmediato

```
mov cx,5
```

Direccionamiento directo

```
mov ax,valor
```

Direccionamiento indirecto por registro

```
mov cx,[bx]
```

Direccionamiento por base mas índice

```
mov ax,[bx+di]
```

Direccionamiento relativo a registro

```
mov ax,lista[bx]

mov dx,[bx+3]
```

Direccionamiento relativo a base mas índice

```
mov al,lista[bx+di]

mov al,[bx+di+3]
```

Direccionamiento por índice por escala

```
mov ax,[ebx+2*ecx]
```

Modos de direccionamiento para el programa

JMP (jump) y CALL se utilizan para indicar la localización de la próxima instrucción que se desea ejecutara.

Direccionamiento directo

En este modo la dirección física de la instrucción que se desea ejecutar se codifica como parte de la instrucción.

Direccionamiento relativo

En este modo la dirección de la instrucción que se ejecutará se establece en relación al valor del IP (instruction pointer).

Direccionamiento indirecto

La dirección (o parte de ésta) a la que se desea brincar está contenida en un registro. La ventaja que ofrece este modo es que permite que se pueda alterar la dirección de forma dinámica.

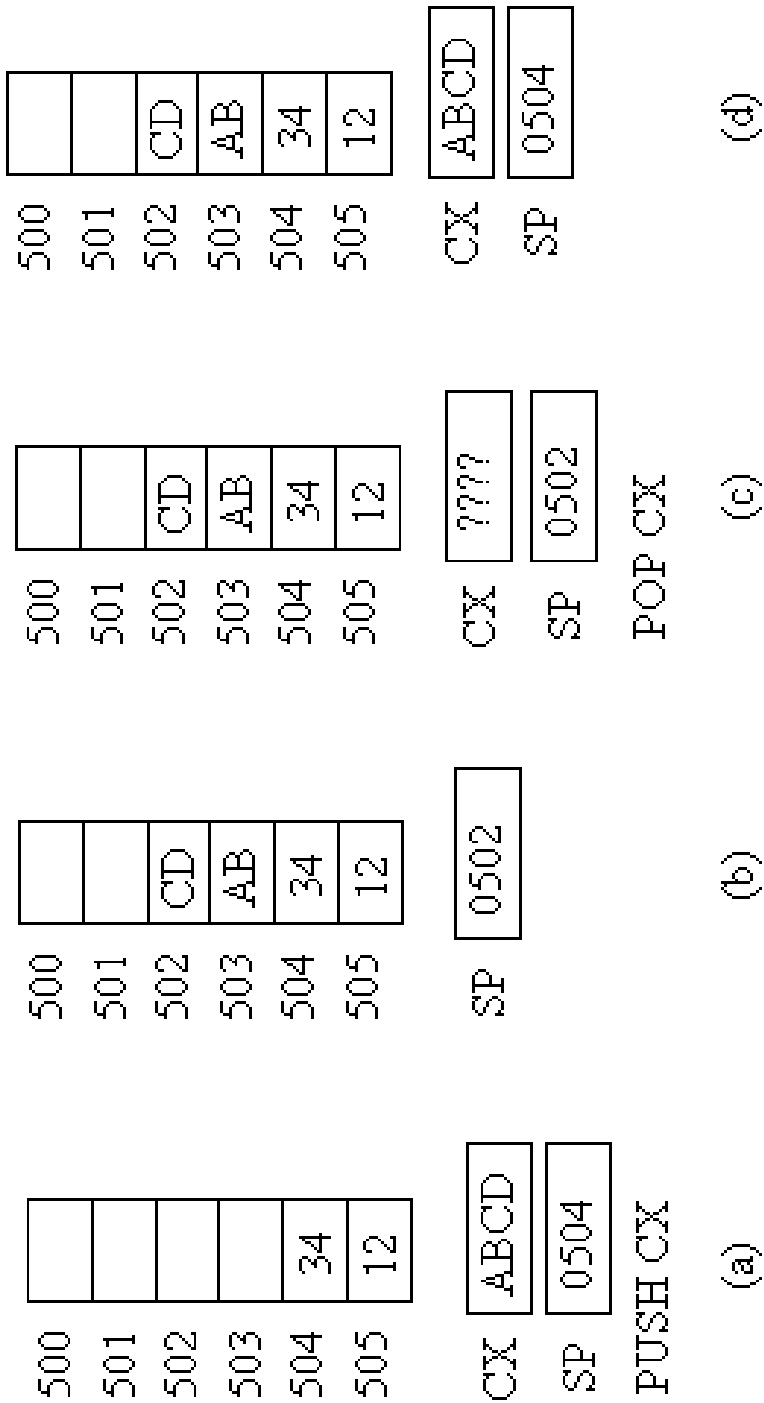
```
jmp case[bx]
jmp ax
jmp cx
```

Modos de direccionamiento con el stack

En los procesadores de Intel el stack no almacena bites individuales sino words. Del 80386 en adelante también se pueden double-words

La instrucción para insertar valores en el stack es *push* y la de extraer valores es *pop*.

Ejemplo de la operación del stack


500
501
502
503
504
505
34
12
CX
ABCD
SP
0504
PUSH CX
(a)
CD
AB
SP
0502
(b)
CX
????
POP CX
(c)
(d)

Ejemplo de la ejecución de push del valor 1234H.


```
AX=150F  BX=0000  CX=0032  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=15F8  ES=15E6  SS=15FA  CS=15F6  IP=0010   NV UP EI PL NZ AC PE NC
15F6:0010 FF360400      PUSH    [0004]                      DS:0004=1234
-d ss:00f0
15FA:00F0  CC 92 88 66 0F 15 0F 15-00 00 10 00 F6 15 43 0F   ...f......
15FA:0100  50 E8 8D FF 8B 46 F6 C9-C3 52 8B D0 E8 C2 FC 72   P....F...R
```

```
AX=150F  BX=0000  CX=0032  DX=0000  SP=00FE  BP=0000  SI=0000  DI=0000
DS=15F8  ES=15E6  SS=15FA  CS=15F6  IP=0014   NV UP EI PL NZ AC PE NC
15F6:0014 FF360600      PUSH    [0006]                      DS:0006=2222
-d ss:00f0
15FA:00F0  CC 92 88 66 0F 15 00 00-14 00 F6 15 43 0F 34 12   ...f......
15FA:0100  50 E8 8D FF 8B 46 F6 C9-C3 52 8B D0 E8 C2 FC 72   P....F...R
```

Formato para un programa en lenguaje del ensamblador

```
01  TITLE Mi primer programa
02  .MODEL SMALL
03  .STACK 100H
04  .DATA
05  marca          db '>>>>'
06  valor1     db 6
07  valor2     db 10
08  valor3     db 1
09  resultado db ?
10  .CODE
11  main  proc
12        mov ax,@DATA
13        mov ds,ax
```

```
14            mov al,valor1
15            add al,valor2
16            sub al,valor3
17            mov resultado,al
18            mov ah,4ch
19            int 21h
20            main endp
21    end main
```

Directriz *.MODEL*

Indica el modelo de memoria que se utilizará para el programa. Las opciones son:

TINY: Todo el código y lada ta se acomodará en un mismo segmento. El archivo que se generará es *.com*.

SMALL: Un segmento de data y un segmento de código

MEDIUM: Un segmento de data y todos los segmentos de código que se requieran

COMPACT: Un segmento de código y todos los segmentos de data que se requieran

LARGE: Cualquier número de segmentos de código y de data

HUGE: Similar a large pero los segmentos de data pueden ocupar más de 64KB

FLAT: Sólo disponible en para MASM 6.0 en adelante. Toda la data y el código se acomodan en un segmento de 512KB.

Tipos disponibles para las definiciones de “variables” son:

-**DB (define byte):** separa un bite de memoria

-**DW (define word):** separa un word (dos bites) de memoria

-**DD: define double word**: separa dos words de memoria

- **DQ: define quad word**: separa cuatro words de memoria

-**DT: define ten bytes** : separa diez bytes de memoria

Ensamblador, linker y debugger

MASM

Crea, entre otros, al archivo objeto (.obj). El formato recomendado es

masm *filename.asm,,,,*

Linker

Crea el archivo ejecutable. El formato recomendado es

link filename,,,,,

Debugger

Nos permite ejecutar el programa y ver (y mpodificar) el ambiente de ejecución mientras se efectúan las instrucciones. El formato para ejecutarlo es

debug filename.exe

Algunos de los comandos del debug:

-t: Trace ejecuta la instrucción que apunta el instruction pointer (IP). Se muestra el contenido de los registros luego de la ejecución, la memoria (si alguna) a la que se hizo acceso y la próxima instrucción que se ejecutará.

-**t #:** # es el número de instrucciones que se desean ejecutar. Es el equivalente a ejecutar el comando trace # veces.

-g: Go ejecuta e programa hasta el final

-r: Registers muestra el contenido de los registros

-d: Dump muestra el contenido de la memoria. Por defecto se muestra el contenido del segmento de data

-**d ds:100**: muestra el contenido del segmento de data a partir de la localización 100H

-**d cs:0**: muestra el contenido del segmento de código a partir de la localización 0.

-**u: Unassembly** desensambla el contenido de la memoria indicada. No se muestra el nombre de las etiquetas sino la dirección asignada a las mismas.

-q: Quit termina la ejecución de debug.

Ejemplo del uso del debbuger

```
AX=15F0  BX=0000  CX=0020  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=15DF  ES=15DF  SS=15F1  CS=15EF  IP=0003   NV UP EI PL NZ NA PO NC
15EF:0003 8ED8          MOV     DS,AX
-t

AX=15F0  BX=0000  CX=0020  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=15F0  ES=15DF  SS=15F1  CS=15EF  IP=0005   NV UP EI PL NZ NA PO NC
15EF:0005 A00C00        MOV     AL,[000C]                          DS:000C=06
-t

AX=1506  BX=0000  CX=0020  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=15F0  ES=15DF  SS=15F1  CS=15EF  IP=0008   NV UP EI PL NZ NA PO NC
15EF:0008 02060D00      ADD     AL,[000D]                          DS:000D=0A
-t

AX=1510  BX=0000  CX=0020  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=15F0  ES=15DF  SS=15F1  CS=15EF  IP=000C   NV UP EI PL NZ AC PO NC
15EF:000C 2A060E00      SUB     AL,[000E]                          DS:000E=01
-t

AX=150F  BX=0000  CX=0020  DX=0000  SP=0100  BP=0000  SI=0000  DI=0000
DS=15F0  ES=15DF  SS=15F1  CS=15EF  IP=0010   NV UP EI PL NZ AC PE NC
15EF:0010 A20F00        MOV     [000F],AL                          DS:000F=00
```

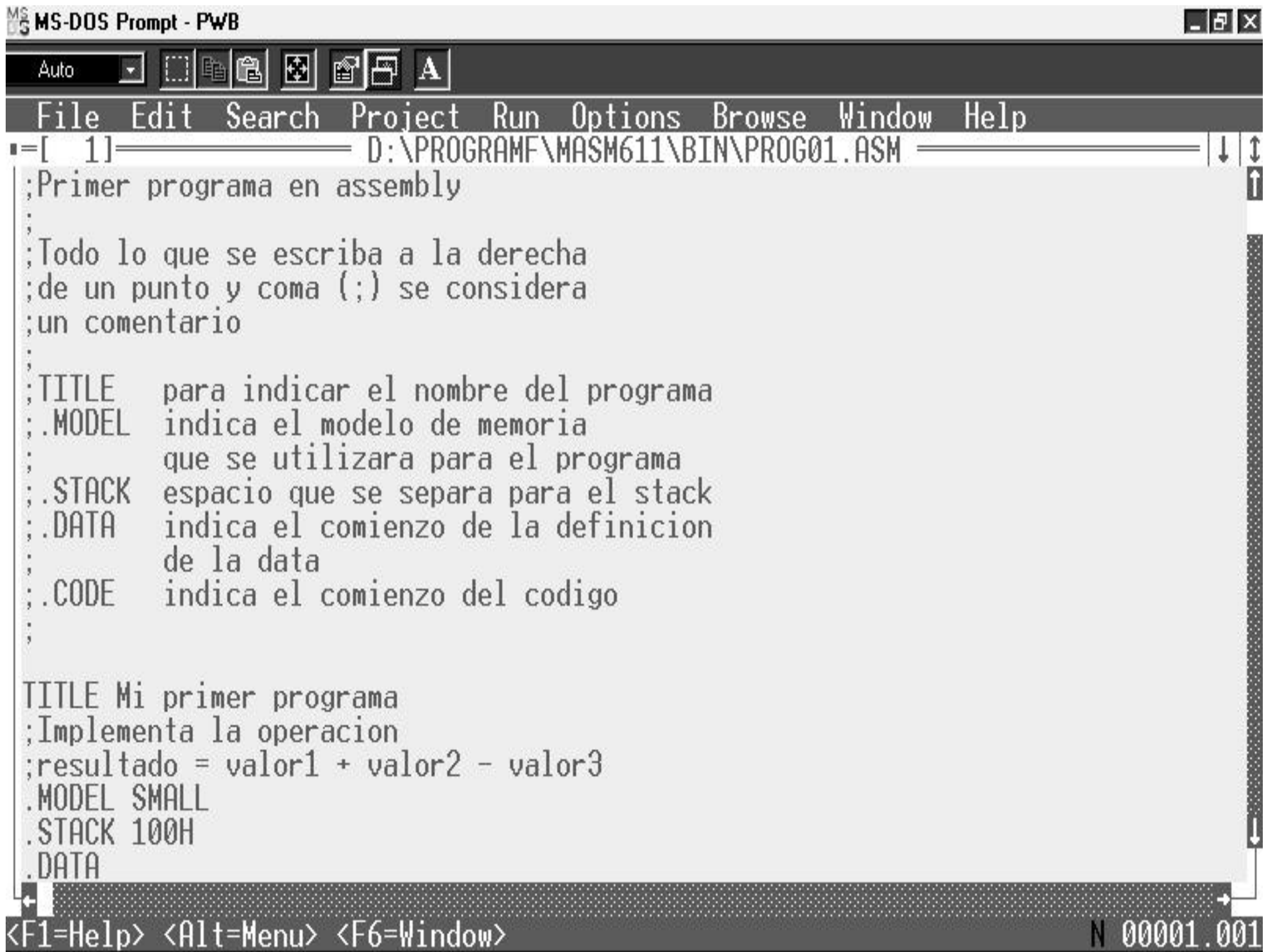
Ejemplo del contenido de la memoria

```
-d
15EF:0000  B8 F0 15 8E D8 A0 0C 00-02 06 0D 00 2A 06 0E 00   ............*...
15EF:0010  A2 0F 00 B4 4C CD 21 00-3E 3E 3E 3E 06 0A 01 0F   ....L.!.>>>>....
15EF:0020  1E B4 52 8B 07 A3 AA 16-8B D8 C1 E3 02 03 1E 50   ..R............P
15EF:0030  56 8E 06 52 56 26 C7 07-00 00 8B 1E AA 16 C1 E3   V..RV&..........
15EF:0040  02 03 1E 50 56 8E 06 52-56 26 C7 47 02 00 04 C7   ...PV..RV&.G....
15EF:0050  45 02 00 00 8D 86 3A FF-50 E8 52 60 6A 00 FF 76   E.....:.P.R`j..v
15EF:0060  0A 6A 00 6A 79 8D 46 90-50 E8 A8 70 83 C4 0A 8D   .j.jy.F.P..p....
15EF:0070  46 90 68 FF 0F 50 E8 FB-5F 8A 46 06 25 02 00 3D   F.h..P.._.F.%..=
```

Ejemplo del producto de ejecutar el comando U (Unassembly).

```
-u cs:0
15EF:0000 B8F015        MOV     AX,15F0
15EF:0003 8ED8          MOV     DS,AX
15EF:0005 A00C00        MOV     AL,[000C]
15EF:0008 02060D00      ADD     AL,[000D]
15EF:000C 2A060E00      SUB     AL,[000E]
15EF:0010 A20F00        MOV     [000F],AL
15EF:0013 B44C          MOV     AH,4C
15EF:0015 CD21          INT     21
15EF:0017 003E3E3E      ADD     [3E3E],BH
15EF:001B 3E            DS:
15EF:001C 06            PUSH    ES
15EF:001D 0A01          OR      AL,[BX+DI]
15EF:001F 001EB452      ADD     [52B4],BL
-
```

Utilizando el IDE del MASM 6.11



Algunos Comandos Básicos

- Crear un nuevo programa
  File|New
- Abrir un programa existente
  File|Open (Browse típico de aplicaciones para MS-DOS)
- Guardar un programa
  File|Save
- Compilar el programa
  Project|Compile
- Ejecutar el programa
  Run|Execute
- Ejecutar el debugger
  Run|Debug

Ejemplo de la pantalla del debugger

```
MS-DOS Prompt - CV
Auto
 File  Edit  Search  Run  Data  Options  Calls  Windows  Help
-[3]-------------------------------- source1 CS:IP -----------------   -[7]reg-
2193:0000 B89421          MOV     AX,2194                               AX = 0000
2193:0003 8ED8            MOV     DS,AX                                 BX = 0000
2193:0005 A00C00          MOV     AL,BYTE PTR [000C]                    CX = 0000
2193:0008 02060D00        ADD     AL,BYTE PTR [000D]                    DX = 0000
2193:000C 2A060E00        SUB     AL,BYTE PTR [000E]                    SP = 0100
2193:0010 A20F00          MOV     BYTE PTR [000F],AL                    BP = 0000
2193:0013 B44C            MOV     AH,4C                                 SI = 0000
2193:0015 CD21            INT     21                                    DI = 0000
2193:0017 003E3E3E        ADD     BYTE PTR [3E3E],BH                    DS = 2183
2193:001B 3E06            PUSH    ES                                    ES = 2183
2193:001D 0A01            OR      AL,BYTE PTR [BX+DI]                   SS = 2195
2193:001F 0000            ADD     BYTE PTR [BX+SI],AL                   CS = 2193
2193:0021 0000            ADD     BYTE PTR [BX+SI],AL                   IP = 0000
2193:0023 0000            ADD     BYTE PTR [BX+SI],AL                   FL = 3200
2193:0025 0000            ADD     BYTE PTR [BX+SI],AL
2193:0027 0000            ADD     BYTE PTR [BX+SI],AL                  NV UP EI PL
2193:0029 0000            ADD     BYTE PTR [BX+SI],AL                  NZ NA PO NC
2193:002B 0000            ADD     BYTE PTR [BX+SI],AL
-[5]----------------------- memory1 b DS:0 ---------------------------
2183:0000  CD 20 00 A0 00 9A F0 FE 1D F0 96 02 05
2183:000D  14 97 03 05 14 01 00 BE 08 A1 15 01 01
2183:001A  01 00 01 01 07 FF FF FF FF FF FF FF FF
2183:0027  FF FF FF FF FF B2 15 94 27 68 11 14 00
2183:0034  18 00 83 21 FF FF FF FF 00 00 00 00 07
2183:0041  0A 00 00 00 00 00 00 00 00 00 00 00 00
2183:004E  00 00 CD 21 CB 00 00 00 00 00 00 00 00
2183:005B  00 00 20 20 20 20 20 20 20 20 20 20 20
2183:0068  00 00 00 00 00 20 20 20 20 20 20 20 20
2183:0075  20 20 20 00 00 00 00 00 00 00 00 00 0D
2183:0082  00 00 00 00 00 00 00 00 54 4D 50 3D 43   ........TMP=C
2183:008F  3A 5C 57 49 4E 44 4F 57 53 5C 54 45 4D   :\WINDOWS\TEM
2183:009C  50 00 54 45 4D 50 3D 43 3A 5C 57 49 4E   P.TEMP=C:\WIN
2183:00A9  44 4F 57 53 5C 54 45 4D 50 00 50 52 4F   DOWS\TEMP.PRO
2183:00B6  4D 50 54 3D 24 70 24 67 00 77 69 6E 62   MPT=$p$g.winb
=[9]====================== command ===================================
CV1053 Warning:  TOOLS.INI not found
CV0101 Warning:  no CodeView information for 'D:\PROGRAMF\MASM611\
>
<F8=Trace> <F10=Step> <F5=Go> <F3=S1 Fmt> <Sh+F3=M1 Fmt>             DEC
```

Microprocesadores

Opciones de Ventanas para el Debugger

```
Restore               Ctrl+F5
Move                  Ctrl+F7
Size                  Ctrl+F8
Minimize              Ctrl+F9
Maximize             Ctrl+F10
Close                 Ctrl+F4
Tile                 Shift+F5
Arrange                Alt+F5

0. Help                 Alt+0
1. Locals               Alt+1
2. Watch                Alt+2
3. Source 1             Alt+3
4. Source 2             Alt+4
5. Memory 1             Alt+5
6. Memory 2             Alt+6
7. Register             Alt+7
8. 8087                 Alt+8
•9. Command             Alt+9

View Output                F4
```

Instrucciones para el movimiento de data

MOV

mov dst, src

- Copia en *dst* el contenido de *src*
- No transferencia de memoria a memoria
- No transferencia memoria a reg. de segmento
- Operandos pueden ser registros o combinación de un registro y una referencia a memoria
- Operandos tienen que ser de tipos compatibles

PUSH & POP

- Se utilizan para almacenar y extraer data del stack (16 & 32 bits)

push opr
pop opr

Variaciones de la instrucción push

Inst.	Ejemplo	Descripción
pushw	pushw 1234H	Inserta un word inmediato
pushd	pushd 12345678H	Inserta un double-word inmediato
pusha	pusha	Inserta todos los registros de 16 bits
pushad	pushad	Inserta todos los registros de 32 bits
pushf	pushf	Inserta el registro de los flags
pushfd	pushfd	Inserta el registro de los flags

Variaciones de la instrucción *pop*

Inst.	Ejemplo	Descripción
popa	popa	Extrae todos los registros de 16 bits
popad	popad	Extrae todos los registros de 32 bits
popf	popf	Extrae el registro de los flags
popfd	popfd	Extrae el registro de los flags

Direcciones efectivas

En algunas ocasiones se necesita determinar la dirección de memoria que contiene o podría contener un valor en específico. Para estas situaciones el lenguaje del ensamblador provee instrucciones que permiten determinar el desplazamiento dentro de un segmento que posee una dirección en particular y también para determinar el segmento.

Instrucción	Descripción
lea reg,memref	Carga el registro *bx* con la dirección relativa de *valor*
lds reg,memref	Carga el registro *si* con la dirección relativa de *valor* y al registro *ds* con el segmento que la contiene
les reg,memref	Carga el registro *bx* con la dirección relativa de *valor* y al registro *es* con el segmento que la contiene

lfs reg,memref	Carga el registro *di* con la dirección relativa de *valor* y al registro *fs* con el segmento que la contiene
lgs reg,memref	Carga el registro *eax* con la dirección relativa de *valor* y al registro *gs* con el segmento que la contiene
lss reg,memref	Carga el registro *sp* con la dirección relativa de *valor* y al registro *ss* con el segmento que la contiene

Instrucciones para la transferencia de strings

- Utilizan el valor del *flag de dirección* (direction flag, *D*) para determinar la forma en que se procesarán los strings

- Utilizan los registros DI y SI para el procesamiento de los strings.

La instrucción *lods* carga al registro *al, ax* o *eax* con el contenido de la localización del segmento de data (apuntado por *ds*) señalada por el registro *si*. La cantidad de bites que se leen a partir de esa localización (1, 2 o 4) depende del postfijo que se le añada a *lods* o del tipo (byte, word o double-word) de la data a la que hace referencia.

Si el valor del flag de dirección D = 0, el valor de *SI* se incrementa por la cantidad indicada en la tabla. Si el valor del flag de dirección D = 1, el valor de *SI* se decrementa por la cantidad indicada en la tabla.

instrucción	operación que realiza
LODSB	AL = DS:[SI]; SI = SI ± 1
LODSW	AX = DS:[SI]; SI = SI ± 2
LODSD	EAX = DS:[SI]; SI = SI ± 4
LODS memref8	AL = DS:[SI]; SI = SI ± 1
LODS mref16	AX = DS:[SI]; SI = SI ± 2
LODS mref32	EAX = DS:[SI]; SI = SI ± 4

STOS

- Almacenan en memoria el contenido de los registros AL, AX o EAX. -- El contenido se almacena en el segmento indicado por ES
- En lugar de utilizarse el registro SI se utiliza el registro DI.

instrucción	operación que realiza
STOSB	ES:[DI] = AL; DI = DI ± 1
STOSW	ES:[DI] = AX; DI = DI ± 2
STOSD	ES:[DI] = EAX; DI = DI ± 4
STOS memref8	ES:[DI] = AL; DI = DI ± 1
STOS mref16	ES:[DI] = AX; DI = DI ± 2
STOS mref32	ES:[DI] = EAX; DI = DI ± 4

Notas

Prefijo de repetición (REP)

Este prefijo se puede añadir a todas las instrucciones de movimiento de strings (Excepto LODS) para indicar la cantidad de valores que se desean mover. Combinado con el incremento o decremento automático de los registros SI y DI permite lograr, con poco código, el movimiento de un bloque de valores. El prefijo provoca que CX se decremente por 1 cada vez que la instrucción se ejecuta. Cuando CX llega a 0 la instrucción se ejecuta y luego se continúa con la próxima instrucción en secuencia.

```
MOV CX, 10
REP  STOSW
```

El registro DI se incrementa o decrementa dependiendo del valor del flag de dirección.

Instrucción MOVS para mover un string de memoria a memoria

Permite el movimiento de data de memoria a memoria. Junto con REP, podemos utilizar MOVS para mover un bloque de memoria de una localización a otra. La instrucción tiene las variantes MOVSB, MOVSW y MOVSD que mueven valores del segmento de data al segmento extra e incrementan o decrementan los valores de SI y DI de acuerdo al valor del flag de dirección. Los valores de estos registros se modifican acorde al tamaño del valor que se mueve.

Instrucción para intercambio de valores XCHG

Se utiliza para intercambiar el contenido de un registro con el contenido de cualquier otro registro (menos los registros de segmentos) o localización de memoria (siempre que los tipos sean compatibles).

Operaciones de entrada y salida con puertos: IN y OUT

La instrucción IN transfiere información de un puerto al acumulador y OUT del acumulador al puerto.

El acumulador puede ser AL, AX o EAX.

La dirección del puerto se puede indicar de forma inmediata (este formato se conoce como *fixed-port*) o utilizando el registro DX (este formato se conoce como *variable-port*).

Instrucciones de para mover data extendiendo el signo o el cero MOVSX y MOVZX (80386) en adelante

En algunas ocasiones necesitamos mover el contenido de un registro o localización de memoria a un registro definido con un tipo que implica un mayor tamaño.

Si el valor que vamos a mover no tiene signo, entonces podemos utilizar la instrucción MOVZX que convierte en 0 los bits de la mitad más significativa del destinatario.

Si el número tiene signo utilizamos MOVSX. Esta instrucción copia en todos los bits de la mitad más significativa del destinatario el bit más significativo (bit del signo) de la fuente.

Note que en el formato de complemento de dos esta extensión no afecta el valor original.

Instrucción de intercambio dentro del mismo registro BSWAP (80486 en adelante)

Esta instrucción se utiliza para intercambiar el contenido de registros de 32 bits. La instrucción intercambia el bite más significativo con el menos significativo y el segundo con el tercero. Por ejemplo

```
MOV EAX, 12345678h
BSWAP EAX
```

Provoca que el contenido del registro EAX sea 78563412H.

Utilización de prefijos para indicar el segmento que se desea utilizar en un movimiento de data

En algunas ocasiones se desean mover valores de una localización a otra pero se desea utilizar un segmento que no es el que utiliza la instrucción por de facto. El lenguaje del ensamblador permite expresar explícitamente el segmento que deseamos utilizar. Por ejemplo:

```
MOV AX,SS:[SI]
```

El uso de los prefijos se debe limitar en lo posible porque su ejecución es significativamente más lenta que cuando se utilizan los segmentos que las instrucciones utilizan por de facto.

Utilización de mascarillas para modificar bits

Convirtiendo bits en 1

Utilizamos una mascarilla con un 0 en cada posición que deseamos dejar inalterada y con un 1 en cada posición que deseamos convertir en 1.

Hacemos un OR por bit (bitwise) de la mascarilla con la memoria/registro deseada.

*Recuerde: $X + 0 = X$ & $X + 1 = 1$

Ejemplo:
Convertir en 1 los bits 0 y 3 del registro AL.

Solución:

```
OR AL,00001001B
```

Convirtiendo bits en 0

Se utiliza la función AND

Una mascarilla con un 1 en cada posición que se desee dejar inalterada y un 0 en cada posición que se desea convertir en 0.

La operación lógica AND tiene las siguientes características.

$$X * 0 = 0$$

y

$$X * 1 = X.$$

Ejemplo: Convertir en 0 a los bits 0 y 3 de AL.

Solución:

```
AND AL,11110110B.
```

Invirtiendo el valor de los bits

Para invertir el valor de bits específicos se utiliza la función XOR. Esta función presenta el siguiente comportamiento.

$$0 \oplus 1 = 1$$

$$1 \oplus 1 = 0$$

Por lo tanto,

$$X \oplus 1 = \overline{X}$$

Por otro lado como

$$0 \oplus 0 = 0$$

podemos concluir que

$$X \oplus 0 = X$$

Ejemplo: si deseáramos invertir los bits en las posiciones 0 y 3 del registro AL utilizaríamos la instrucción

```
XOR AL,00001001B
```

Combinación de las operaciones para manejar bits

La siguiente secuencia de instrucciones invierte el bit 0 de AL, convierten en 0 el bit 1 y convierten en 1 el bit 2.

```
XOR AL,00000001b
AND AL,11111101b
OR  AL,00000100b
```

Memory mapped I/O

-I/O separado

Las direcciones de los dispositivos de I/O no se mezclan con las de acceso a memoria.

El procesador indica por medio de una señal si el acceso es a memoria o I/O.

Diferentes instrucciones para el movimiento de data

-Memory-mapped I/O

Las direcciones de los dispositivos de I/O se tratan como si fueran localizaciones de memoria convencional

Programa para escribir en la memoria de video

```
title Escribe memoria de video
.model small
.code
main proc
    mov ax,@data
    mov ds,ax
    mov ax,0b800h       ;para inicializar segmento
    mov es,ax           ;se usara el segmento extra
    mov cx,2000         ;2000 caracteres (25 * 80)
    mov bx,0h           ;se usara bx como indice

proxDir:                ;mueve el caracter a memoria
      mov es:[bx],0010000001000001b
      inc bx
      inc bx
    loop proxDir      ;pasa a la prox. localizacion
     mov ax,4c00h
    int 21h
main endp
end main
```

Ejemplo del uso de mascarillas para activar la bocina interna de la computadora

```
title Activa bocina
.model small
.code
main proc

    in al,61h           ;lee mem bocina interna
    or al,00000011b     ;últimos dos bits a 1
    out 61h,al          ;envía a la mem bocina
    mov cx,0ffffh       ;inicializa contador para
again:                  ;ciclo de espera
    loop again          ;alarga el sonido
    in al,61h           ;lee mem bocina interna
    and al,11111100b    ;últimos dos bits a 0
    out 61h,al          ;silencia bocina

    mov ax,4c00h
    int 21h
main endp
end main
```

Ejemplo del uso de mascarillas manejar la bocina interna de la computadora

```
TITLE Speaker Demo Program
;This program plays a series of ascending notes on
;an IBM-PC or compatible computer

.model  small
.stack  10H
.data
speaker EQU 61h             ;address of speaker port
timer    EQU 42h            ;address of timer port
delay1   EQU 500
delay2   EQU 0D000h         ;delay between notes

.code
main    proc
        in      al, speaker     ;get speaker status
        push    ax              ;save status
        or      al,00000011b    ;set lowest 2 bits
        out     speaker,al      ;turn speaker on
```

```
        mov     al, 60      ;startin pitch
L2:     out     timer, al   ;timer port: pulses speaker

        ;Create a delay loop between pitches

        mov     cx, delay1
L3:     push    cx                  ;outer loop
        mov     cx, delay2
L3a:                                ;inner loop
        loop    L3a
        pop cx
        loop    L3
        sub     al,1                ;raise pitch
        jnz     L2                  ;play another note
        pop     ax                  ;get original status
        and     al,11111100b        ;clear lowest 2 bits
        out     speaker, al         ;turn speaker off
      mov     ah, 4ch
      int     21h
main    endp
END     main
```

Operaciones Ariméticas

Suma

Mnemónico: ADD
Descripción: Suma el contenido de dos valores y lo almacena en el primer operando

Formato: ADD dst,src
Ejecución: dst←dst + src
Ejemplo:

```
MOV AX,5
MOV BX,7
ADD AX,BX
```

Luego de ejecutar las instrucciones AX contendrá un 12

Incremento

Mnemónico: INC

Descripción: incrementa el contenido del operando

Formato: INC dst

Ejecución: dst←dst + 1

Ejemplo:

```
MOV AL,5
MOV valor,AL
INC valor
```

Luego de ejecutar las instrucciones

'valor' contendrá un 6

Suma con carry

Mnemónico: ADC
Descripción: Suma dos valores mas el contenido del flag de carry (0 o 1)
Formato: ADC dst,src
Ejecución: dst←dst + src + carry
Ejemplo:

```
MOV AL,200
MOV BL,100
MOV AH,15
MOV BH,8
ADD AL,BL   ;200 + 100 = 300,
             ;AL ← 44, c←1
ADC AH,BH  ;AH←15 + 8 + c,
            ;AH←24
```

Resta

Mnemónico: SUB

Descripción: Resta el contenido de dos operandos y almacena el resultado

Formato: SUB dst,src

Ejecución: dst←dst - src

Ejemplo:

MOV AL,20

SUB AL,5

Luego de ejecutar las instrucciones AL contendrá un 15

Decremento

Mnemónico: DEC
Descripción: Decrementa el contenido de un registro o localización de memoria
Formato: DEC dst
Ejecución: dst←dst 1 1
Ejemplo:

```
MOV AL,5
MOV valor,AL
DEC valor
```

Luego de ejecutar las instrucciones ‘valor’ contendrá un 4

Resta con el carry

Mnemónico: SBB

Descripción: Suma el contenido del segundo operando y el carry y le resta el resultado al primer operando

Formato: SBB dst,src

Ejecución: dst←dst – src - c

Ejemplo:

```
STC             ;set carry
MOV AL,10
SBB AL,4
```

Luego de ejecutar las instrucciones AL contendrá un 5

Comparación

Mnemónico: CMP

Descripción: Realiza la misma operación de SUB pero sin almacenar el valor. Su uso principal preparar el contenido de los flags previo a ejecutar una instrucción de brinco.

Formato: CMP op1,op2

Ejecución: op1 – op2

Ejemplo:

```
MOV AL,8
MOV AH,8
CMP AL,AH
JZ valores_iguales  ;brinca a la etiqueta
                    ;indicada si ZF = 1
```

Multiplicación sin signo

Mnemónico: MUL
Descripción: Multiplica el contenido del acumulador por el valor en el registro o contenido de memoria indicado asumiendo valores sin signo.

Formato: MUL op1
Ejecución: AX ← AL* op1(8)
DX:AX←AX*op1(16)
EDX:EAX←EAX*op1(32)

Ejemplo:

```
MOV CX,4000
MOV AX,4000
MUL CX
```

16000000D = F42400H.
DX = 00F4H y el de AX = 2400H

Multiplicación con signo

Mnemónico: IMUL

Descripción: Multiplica el contenido del acumulador por el valor en el registro o contenido de memoria indicado asumiendo valores con signo.

Formato: IMUL op1

Ejecución: AX ← AL* op1(8)

DX:AX←AX*op1(16)

EDX:EAX←EAX*op1(32)

Ejemplo:

```
MOV BL,5
MOV AL,-3
IMUL BL
```

-15D en complemento de 2 es 11110001B. Por lo tanto AX = FFF1H

División sin signo

Mnemónico: DIV

Descripción: Divide el contenido del acumulador o combinación del registro D con el acumulador por un registro o referencia a memoria

Formato: DIV op

Ejecución: AL ← AX div op(8)
AH ← AX mod op(8)

AX ← DX:AX div op(16)
DX ← DX:AX mod op(16)

EAX ← EDX:EAX div op(32)
EDX ← EDX:EAX mod op(32)

Instrucciones para Implementar Funciones Lógicas

Función AND

Mnemónico: AND
Descripción: Realiza un AND bit por bit (bitwise) entre los bits de ambos operandos guardando el resultado en el primer operando

Formato: AND dst,src
Ejecución: dst ← dst AND src
Ejemplo:

```
MOV AL,11110000B
MOV AH,11000011B
AND AL,AH
```

Función OR

Mnemónico: OR

Descripción: Realiza un OR bit por bit (bitwise) entre los bits de ambos operandos guardando el resultado en el primer operando

Formato: OR dst,src

Ejecución: dst ← dst OR src

Ejemplo:

```
MOV AL,11110000B
MOV AH,11000011B
OR AL,AH
```

Función XOR

Mnemónico: XOR
Descripción: Realiza un XOR bit por bit (bitwise) entre los bits de ambos operandos guardando el resultado en el primer operando

Formato: XOR dst,src
Ejecución: dst ← dst XOR src
Ejemplo:

```
MOV AL,11110000B
MOV AH,11000011B
XOR AL,AH
```

Función TEST

Mnemónico: TEST

Descripción: Igual que el AND pero no almacena el resultado. Generalmente se utiliza para modificar el contenido de los flags y luego realizar brincos (instrucciones de jump) basándose en los resultados.

Formato: TEST op1, op2

Ejecución: op1 AND op2

Complemento de uno

Mnemónico: NOT
Descripción: Obtiene el complemento de uno (invierte todos los bits) del operando y almacena el resultado en el mismo.

Formato: NOT dst
Ejecución: dst ← complemento de 1 de (dst)
Ejemplo:

```
MOV AL,11110000B
NOT AL
```

Complemento de dos

Mnemónico: NEG

Descripción: Obtiene el complemento de dos del operando y almacena el resultado en el mismo. Esto es el equivalente a multiplicar el operando por –1.

Formato: NEG dst

Ejecución: dst ← Comp de dos de (dst)

Ejemplo:

```
MOV AL,11110000B
NEG AL
```

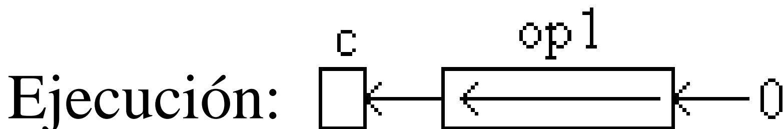
Instrucciones de Desplazamiento

Desplazamiento Lógico a la Izquierda

Mnemónico: SHL

Descripción: Desplaza los bits del operando una posición hacia la izquierda. El bit más significativo se mueve al flag de carry (CF) y en el menos significativo se almacena un 0.

Formato: SHL op1

Ejecución: 

Ejemplo: MOV AL,11110000B
SHL AL

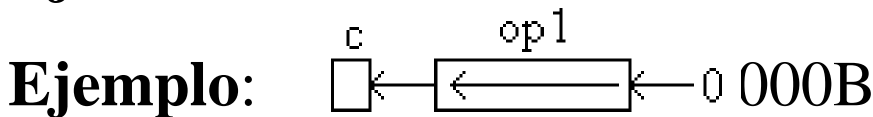
Desplazamiento Aritmético a la Izquierda

Mnemónico: SAL

Descripción: Desplaza los bits del operando una posición hacia la izquierda. El bit más significativo se mueve al flag de carry (CF) y en el menos significativo se almacena un 0. Produce el mismo resultado que el SHL

Formato: SAL op1

Ejecución:



Ejemplo: 000B

SAL AL

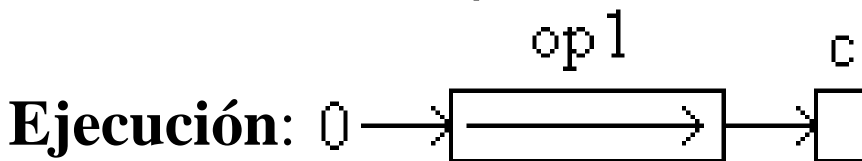
Desplazamiento Lógico a la Derecha

Mnemónico: SHR

Descripción: Desplaza los bits del operando una posición hacia la derecha. El bit menos significativo se mueve al flag de carry (CF) y en el más significativo se almacena un 0.

Formato: SHR op1

Ejecución:



Ejemplo: MOV AL,11110000B
SHR AL

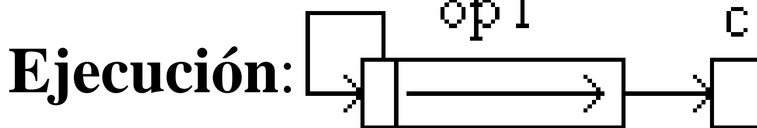
Desplazamiento Aritmético a la Derecha

Mnemónico: SAR

Descripción: Desplaza los bits del operando una posición hacia la derecha. El bit menos significativo se mueve al flag de carry (CF) y en el más significativo se mantiene el valor existente antes de realizar la operación. Esto es, se mantiene el signo del valor

Formato: SAR op1

Ejecución:



Ejemplo: MOV AL,11110000B

SAR AL

Instrucciones de Rotación

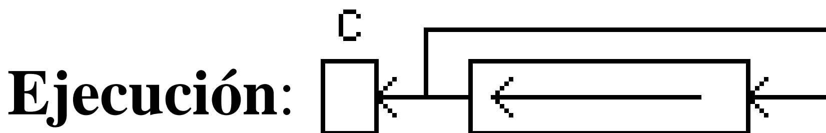
Rotación a la Izquierda

Mnemónico: ROL

Descripción: Mueve todos los bits del operando una posición a la izquierda. El bit más significativo se mueve a la posición del menos significativo y además, se almacena en CF.

Formato: ROL op1

Ejecución:



Ejemplo: MOV AL,11110000B
ROL AL

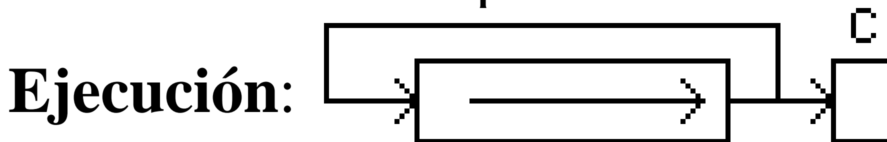
Rotación a la Derecha

Mnemónico: ROR

Descripción: Mueve todos los bits del operando una posición a la derecha. El bit menos significativo se mueve a la posición del más significativo y además, se almacena en CF.

Formato: ROR op1

Ejecución:



Ejemplo: MOV AL,11110000B

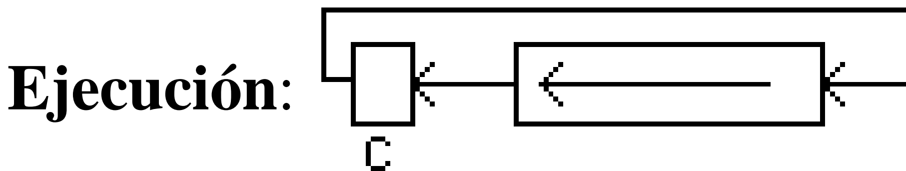
ROR AL

Rotación a la Izquierda a través del Carry

Mnemónico: RCL

Descripción: Mueve todos los bits del operando una posición a la izquierda. El bit más significativo se mueve al CF y el contenido del CF se mueve a la posición del menos significativo.

Formato: RCL op1

Ejecución: 

Ejemplo: MOV AL,11110000B
RCL AL

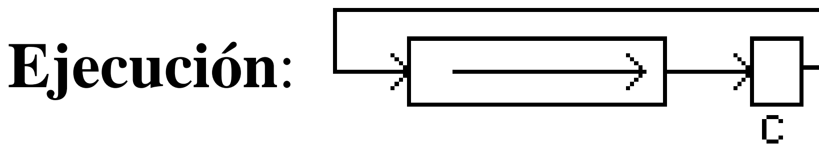
Rotación a la Derecha a través del Carry

Mnemónico: RCR

Descripción: Mueve todos los bits del operando una posición a la derecha. El bit menos significativo se mueve al CF y el contenido del CF se mueve a la posición del más significativo.

Formato: RCR op1

Ejecución:



Ejemplo: MOV AL,11110000B

RCR AL

Instrucciones para Realizar Brincos

Brinco Incondicional

Mnemónico: JMP
Descripción: Transfiere el control del programa a la localización indicada por el operando a la derecha.
Formato: JMP dirección
Ejecución: IP ← dirección

Ejemplo #1:

```
        JMP FINAL
LAB:    MOV AX,5
        MUL AX
        MOV CX,DX
FINAL:  MOV Z,CX
```

Ejemplo #2:

```
TITLE Indirect Jump
.MODEL SMALL
.STACK 100H
.DATA
vector_direcciones  dw code0
                    dw code1
                    dw code2

.CODE
MAIN PROC
        mov ax,@DATA
        mov ds,ax
        mov si,1        ;indice seleccionado
                        ;en vector de direcciones
        mov ax,2        ;porque direcciones ocupan 2 bytes
        mul si
        mov di,offset vector_direcciones
        add di,ax       ;para apuntar a entrada en tabla
        mov bx,[di]     ;para obtener direcci¢n almacenada
        jmp bx          ;realiza el brinco
code0:
        mov cx,0
        jmp fin
code1:
        mov cx,1
        jmp fin
code2:
        mov cx,2
fin:    mov ah,4ch
        int 21h
        main endp
END MAIN
```

Ejemplo #3:

Para implementar la lógica del ejemplo #2 también se pudo almacenar el valor del índice en SI, multiplicarlo por 2 y entonces utilizar la tabla para hacer referencia al código como se muestra a continuación:

```
mov si,índice
add si,si
jmp vector_direcciones[si]
```

Brincos Condicionales

Los brincos condicionales transfieren el control del programa a una de dos localizaciones.

Ejemplo:

```
CMP AX,BX
JE OTRO_SITIO
MOV AX,5
```

Son muchas las operaciones que modifican los flags, pero las más que se utilizan en combinación con los brincos condicionales son CMP y TEST.

Brincos Condicionales

Los brincos condicionales transfieren el control del programa a una de dos localizaciones.

Ejemplo:

```
CMP AX,BX
JE OTRO_SITIO
MOV AX,5
```

Son muchas las operaciones que modifican los flags, pero las más que se utilizan en combinación con los brincos condicionales son CMP y TEST.

Código	Cuándo realiza el brinco
Jc	Si hay carry
Jnc	Si no hay carry
Jno	Si no hay overflow
Jns	Si el resultado no es negativo
Jnp/jpo	Si no hay paridad o es impar
Jo	Si hay overflow

Jp/jpe	Si hay paridad o es par
Js	Si el resultado es negativo
Jcxz	Si CX es 0
Jecxz	Si ECX es 0
Je/jz	Si los valores son iguales
Jne/jnz	Si los valores son distintos

Notas

Ja	Si el primero es mayor	Núms. sin signo
Jae	Si el primero es mayor o igual	Núms. sin signo
jb	Si el primero es menor	Núms. sin signo
Jbe	Si el primero es menor o igual	Núms. sin signo

Jg	Si el primero es mayor	Núms. con signo
Jge	Si el primero es mayor o igual	Núms. con signo
Jl	Si el primero es menor	Núms. con signo
Jle	Si el primero es menor o igual	Núms. con signo

Existen condicionales para el manejo de números con signos y para números sin signos. Cuando el condicional utiliza las palabras 'greater' o 'less' trabaja con números con signo. Cuando se utilizan las palabras 'above' o 'below' se hace referencia a números sin signo.

En procesadores del 8086 al 80286 los condicionales sólo pueden hacer referencia a localizaciones que estén entre 128 bites antes de la localización instrucción (-128) y 127 bites después de la localización de la instrucción. Esto implica que para acceder a localizaciones más distantes es necesario complementar la instrucción con un brinco incondicional.

Ejemplo:

```
          CMP AL,BL
          JA DISTANTE
          MOV AL,5
             :
             :
DISTANTE:
```

```
          CMP AL,BL
          JBE CERCA
          JMP DISTANTE
CERCA:    MOV AL,5

DISTANTE:
```

- Below y Above indica valores sin signo
- Greater y Lower indica valores con signo
- Los brincos condicionales tienen un alcance de –128 a 127 bytes de memoria. A continuación presentamos una solución a esta limitación

Ejemplo:

CMP AL,BL	CMP AL,BL
JA DISTANTE	JBE CERCA
MOV AL,5	JMP DISTANTE
:	CERCA:
:	MOV AL,5
DISTANTE:	DISTANTE:

Implementación de Ciclos

LOOP

Mnemónico: Loop

Descripción: Decrementa el contenido CX por uno. Si CX no es cero se brinca a la etiqueta a la derecha de la instrucción, en caso contrario se continúa con la próxima instrucción que sucede a Loop

Formato: Loop *etiqueta*

Ejecución: DEC CX
JNZ *etiqueta*

Ejemplo:

```
                MOV AX,0
            MOV CX,100
NEXT_SUM:   ADD AX,CX
            LOOP NEXT_SUM
```

Loops Condicionales

LOOPZ (LOOPE)

Mnemónico: LOOPZ (LOOPE)

Descripción: Ambos mnemónicos representan la misma instrucción. La instrucción opera de forma similar a la instrucción LOOP pero añade una condición adicional para que el brinco a la etiqueta se realice, **Z = 1.**

Formato: LOOPZ *etiqueta*

Ejecución:
CX ← CX - 1
If CX != 0 && Z = 1 goto *etiqueta*

Ejemplo:

```
            MOV BX,0
            MOV CX,10
READ_NEXT:  IN AL,14H
            INC BX
            CMP AL,0
            LOOPE READ_NEXT
```

LOOPNZ (LOOPNE)

Mnemónico: LOOPNZ (LOOPNE)

Descripción: Ambos mnemónicos representan la misma instrucción. La instrucción opera de forma similar a la instrucción LOOP pero añade una condición adicional para que el brinco a la etiqueta se realice, **Z = 0.**

Formato: LOOPNZ *etiqueta*

Ejecución:
CX ← CX - 1
If CX != 0 && Z = 0 goto *etiqueta*

Ejemplo:

```
            MOV BX,0
            MOV CX,10
READ_NEXT:  IN AL,14H
            INC BX
            TEST AL,1
            LOOPNZ READ_NEXT
```

Subrutinas

La construcción de las subrutinas hacen uso de las instrucciones CALL y RET y las directrices PROC y ENDP.

La **estructura general** de una subrutina es similar a la siguiente:

```
nombreSubrutina    PROC
                   bloque de código
nombreSubrutina    ENDP
```

Ejemplo:

```
Title Ejemplo de procedures
;Muestra el uso de una subrutina para
;obtener el mayor de tres valores tipo
;byte sin signo
.Model Small
.stack 100H
.Data
x         db 5
y         db 10
z         db 7
masGrande db ?
.code
```

```
main     proc                    ;comienza módulo principal
         mov ax,@data
         mov ds,ax
         mov al,x                ;prepara los valores
         mov ah,y                ;que se le enviarán
         mov bl,z                ;a la rutina
         call retMayor           ;llamada a la rutina
         mov masGrande,al        ;guarda el resultado
         mov ah,4cH
         int 21h                 ;de regreso a dos

main     endp                    ;termina módulo principal

retMayor proc                    ;comienza rutina retMayor
           ;Devuelve en AL el mayor
           ;de los valores en AL, AH y BL

           cmp al,ah
           ja proxComp           ;si al > ah brinca
           xchg al,ah            ;mueve el mayor a al
proxComp:
           cmp al,bl
           ja salida             ;si al > bl brinca
           xchg al,bl            ;mueve el mayor a bl
salida:    ret                   ;regresar
retMayor   endp                  ;termina código subrutina

end    main                      ;termina programa
```

Subrutinas

La construcción de las subrutinas hace uso de las instrucciones CALL y RET y las directrices PROC y ENDP.

La **estructura general** de una subrutina es similar a la siguiente:

nombreSubrutina PROC

bloque de código

nombreSubrutina ENDP

Operaciones de Entrada y Salida de Caracteres Utilizando INT

Instrucción INT

- Se puede usar para llamar funciones del BIOS
- Tiene un operando numérico, entre 00h y FFH, que indica cuál es el servicio que se invoca.
- Las primeras 1,024 localizaciones de memoria corresponden al "interrupt vector table"
- Cada entrada del IVT ocupa 32 bits, CS:IP

Notas

Eventos cuando se ejecuta la instrucción INT

1. Se almacenan los flags en el stack
2. IF = 0, TF =0
3. CS se almacena en el Stack
4. CS adquiere el valor correspondiente del IVT
5. IP se guarda en el stack
6. IP recibe el valor correspondiente del IVT
7. Se ejecuta la rutina, regresa cuando se encuentra IRET regreso de la rutina.

Establecer la Posición del Cursor (INT 10H, Servicio 2)

Valores en los registros

AH: 2
BH: Número de la página de video
DH: Fila del cursor
DL: Columna del cursor

Registros modificados en el retorno: Ninguno

Descripción:
Coloca el cursor en la fila indicada por DH y la columna indicada por DL. BH es el número de página de video a utilizar.

Ejemplo:

El siguiente ejemplo posiciona el cursor en la fila 10 y columna 20 de la pantalla respectivamente.

```
mov ah,2
mov bh,0
mov dh,10
mov dl,20
int 10h
```

Despliegue de un Caracter (INT 21H, Servicio 2)

Valores en los registros

AH: 2
DL: Código ASCII del Caracter

Registros modificados en el retorno: Ninguno

Descripción:
Despliega en la pantalla el caracter correspondiente al valor ASCII almacenado en el registro DL. El caracter aparece en la posición actual del cursor y el cursor se mueve a la próxima posición.

Ejemplo:

El siguiente código despliega un asterisco en la pantalla.

```
mov ah,2
mov dl,'*'
int 21h
```

Despliegue de un String (INT 21H, Servicio 9)

Valores en los registros

AH: 9
DX: Desplazamiento de la localización del string
DS: Segmento de la dirección del string

Registros modificados en el retorno: Ninguno

Descripción:
Despliega en la pantalla el string cuyo segmento y desplazamiento están especificados en los registros DS y DX respectivamente. Es requisito queel string utilice el carácter **$** para indicar el final del mismo.

Ejemplo:

El siguiente código despliega el string **Got Micro?** en la pantalla. Se asume que el valor del registro DS no se necesita modificar.

```
.data
mess db "Got Micro?$"
      :
      :
.code
      :
      :
mov dx, offset mess
mov ah,9
int 21h
```

Notas:

Lectura con Eco de un Caracter del Teclado (INT 21H, Servicio 1)

Valores en los registros
AH: 1

Registros modificados en el retorno:
Se devuelve el ASCII del caracter en AL.

Descripción:
Lee un caracter del teclado haciendo eco del mismo en la pantalla (desplegándolo en la pantalla) y lo devuelve en el registro AL.

Ejemplo: El siguiente código lee un caracter del teclado

```
mov ah,1
int 21h
```

Lectura sin Eco de un Caracter del Teclado (INT 21H, Servicio 8)

Valores en los registros

AH: 8

Registros modificados en el retorno:

Se devuelve el ASCII del caracter en AL.

Descripción:

Lee un caracter del teclado haciendo sin hacer eco del mismo y lo devuelve en el registro AL. Si al ejecutar este servicio se presiona Ctrl+C o Ctrl+Break se ejecuta la acción correspondiente.

Ejemplo:

El siguiente código lee un caracter del teclado sin eco

```
mov ah,8
int 21h
```

Lectura Directa un Caracter (INT 21H, Servicio 6, Función 0FFH)

Valores en los registros

AH: 6
DL: 0FFH

Registros modificados en el retorno:

ZF = 0 si se obtuvo un caracter del buffer del teclado y ZF = 1 si no se obtuvo un caracter. En caso de obtenerse el caracter el código ASCII se devuelve en el registro AL.

Descripción:
Verifica en el buffer del teclado (si no hay redirección) si hay un caracter disponible. De haber un caracter disponible se devuelve el mismo en el registro AL y se almacena un 0 en el flag ZF. Si no se obtiene ningún carácter entonces ZF = 0.

Ejemplo:

El siguiente código borra el contenido del buffer del teclado.

```
ClearBuffer proc
        push ax
        push dx
next:
        mov ah,6
        mov dl,0ffh
        int 21H
        jnz next
        pop dx
        pop ax
        ret
ClearBuffer endp
```

Lectura de un String del Teclado (INT 21H, Servicio 0AH)

Valores en los registros

AH: 0AH
DX: Desplazamiento de la dirección del buffer
DS: Segmento de la dirección del buffer

Registros modificados en el retorno: Ninguno.

Descripción:
Lee un string del teclado permitiendo establecer un largo máximo.

- El buffer comienza en la dirección especificada por DS:DX

- EL primer bite de esta dirección indicar el largo máx. permitido al string

- Si se lee un string de largo mayor que el max. Se trunca el excedente.
- DOS almacena en el 2do bite del buffer el número de caracteres leídos.
- La función espera que presionen “Enter” para devolver el string.
- El ‘beep’ de la computadora suena si se entran caracteres en exceso.

Ejemplo:

El siguiente código lee del teclado un string con un largo máximo de 20 caracteres y despliega el mismo en la pantalla.

```
.data
keybdBuffer  label byte
bufferLength db 20
stringLength  db ?
content     db 20 dup($)
.code
            :
mov ah,0ah
mov dx, offset keybdBuffer
int 21h

mov dx,offset content
mov ah,9
int 21h
```

Introducción al Uso de Macros

- Bloque de enunciados al cual se le asigna un nombre para poder invocarlo desde diferentes lugares en un programa

- Cuando se invoca el macro se copia el bloque de enunciados en el lugar donde se realiza la invocación.

- Ejecución más rápida que una subrutina

- En general su uso provoca programas más grandes que utilizando subrutinas en su lugar

- Permite usar parámetros formales

•Se debe ser cauteloso con el manejo de los parámetros

•El uso registros como parámetros puede ser fuente de problemas

•Tienen que aparecer definidos antes de su invocación

Estructura de un Macro

NombreDelMacro **macro** *lista_de_argumentos*

:

código del macro

:

endm

Ejemplo:

Macro que compara el contenido de los registros AH y AL y los organiza de forma tal que el mayor se guarde en AH y el menor en AL.

```
mOrganiza    macro
             cmp ah,al
             jge fin
             xchg ah,al
fin:
endm
```

Ejemplo:

Versión mejorada del macro anterior para permitir el manejo de argumentos

```
mOrganiza   macro  valor1, valor2
            push ax
            mov ah,valor1
            mov al,valor2
            cmp ah,al
            jge fin
            xchg ah,al
            mov valor1,ah
            mov valor2,al
fin:        pop ax
endm
```

Uso de la directriz Local para evitar el conflicto de etiquetas repetidas

```
mOrganiza   macro  valor1, valor2
            local fin
            push ax
            mov ah,valor1
            mov al,valor2
            cmp ah,al
            jge fin
            xchg ah,al
            mov valor1,ah
            mov valor2,al
fin:        pop ax
endm
```

Uso de Macros para Enmarcar Subrutinas

Ejemplo de llamada a subrutina

```
mov ah,largo
mov dx, offset arreglo
call sorteaproc
```

Macro para enmarcarla

```
sortea  macro cantidad,lista
        push ax
        push dx
        mov ah,cantidad
        mov dx,offset lista
        call sorteaproc
        pop dx
        pop ax
endm
```

Ejemplo de llamada utilizando el macro

sortea largo,arreglo

Parámetros Requeridos

Ejemplo:

```
sortea  macro cantidad:REQ, lista:REQ
        push ax
        push dx
        mov ah,cantidad
        mov dx,offset lista
        call sorteaproc
        pop dx
        pop ax
endm
```

Ejemplo de un macro para implementar un Loop-Extendido

```
mXTLoop     macro dst
            local regresa,salir
            loop regresa
            jmp salir
regresa:    jmp dst
salir:
endm
```

Macro para multiplicar dos valores de 16 bits

El siguiente código presenta un macro para multiplicar dos números de 16 bits. Se asume que el resultado también se puede almacenar en 16 bits. Ambos operandos pueden ser registros o localizaciones de memoria. El segundo argumento puede ser un valor inmediato.

```
mMul  macro  dst:REQ, src:REQ
      push ax
      push dx
      mov ax,dst
      mov dx,src
      mul dx
      mov dst,ax
      pop dx
      pop ax
endm
```

Programa de Ejemplo en el uso de Macros

```
mSetCursor macro fila,columna
    push ax
    push bx
    push dx
    mov ah,2
    mov bh,0
    mov dh,fila
    mov dl,columna
    int 10h
    pop dx
    pop bx
    pop ax
endm
```

```
mReadChar macro char
    push ax
    mov ah,1
    int 21h
    mov char,al
    pop ax
endm

mCLS macro
    push ax
    push cx
    mov ax,0b800h
    mov es,ax
    mov cx,2000
    mov ah,5
    mov al,' '
    rep stosw
    pop cx
    pop ax
endm
```

```
main proc
          mov ax,@data
          mov ds,ax

          mov al,10
          mov bl,40
          mCLS
again:    mSetCursor al,bl
          mReadChar car
          dec bl
          mov dl,car
          cmp dl,'$'
          jne again
          mCLS

          mov ax,4c00h
          int 21h
main endp

end main
```

Microprocesadores (8086/8088)

Características Físicas y Eléctricas

- 40 pines DIP (dual in-line package)
- Rango de temperatura: 32 a 180 grados F
- Fuente de voltaje de 5.V ($\pm$ 10%)
- Requisitos corriente:
  8086 – 360mA
  8088 – 340mA

-Voltajes de Entrada

Nivel Lógico	Voltaje
0	0.8V max.
1	2.0V min.

-Voltajes de Salida

Nivel Lógico	Voltaje
0	0.45V max.
1	2.40V min.

*El voltaje de salida bajo no es 100% compatible con la lógica TTL (0.40 V) y reduce la tolerancia al ruido de 400mV (0.8V – 0.40V) a 350mV.

** El Fan-Out para los pines de salidas es de 10 unidades de carga

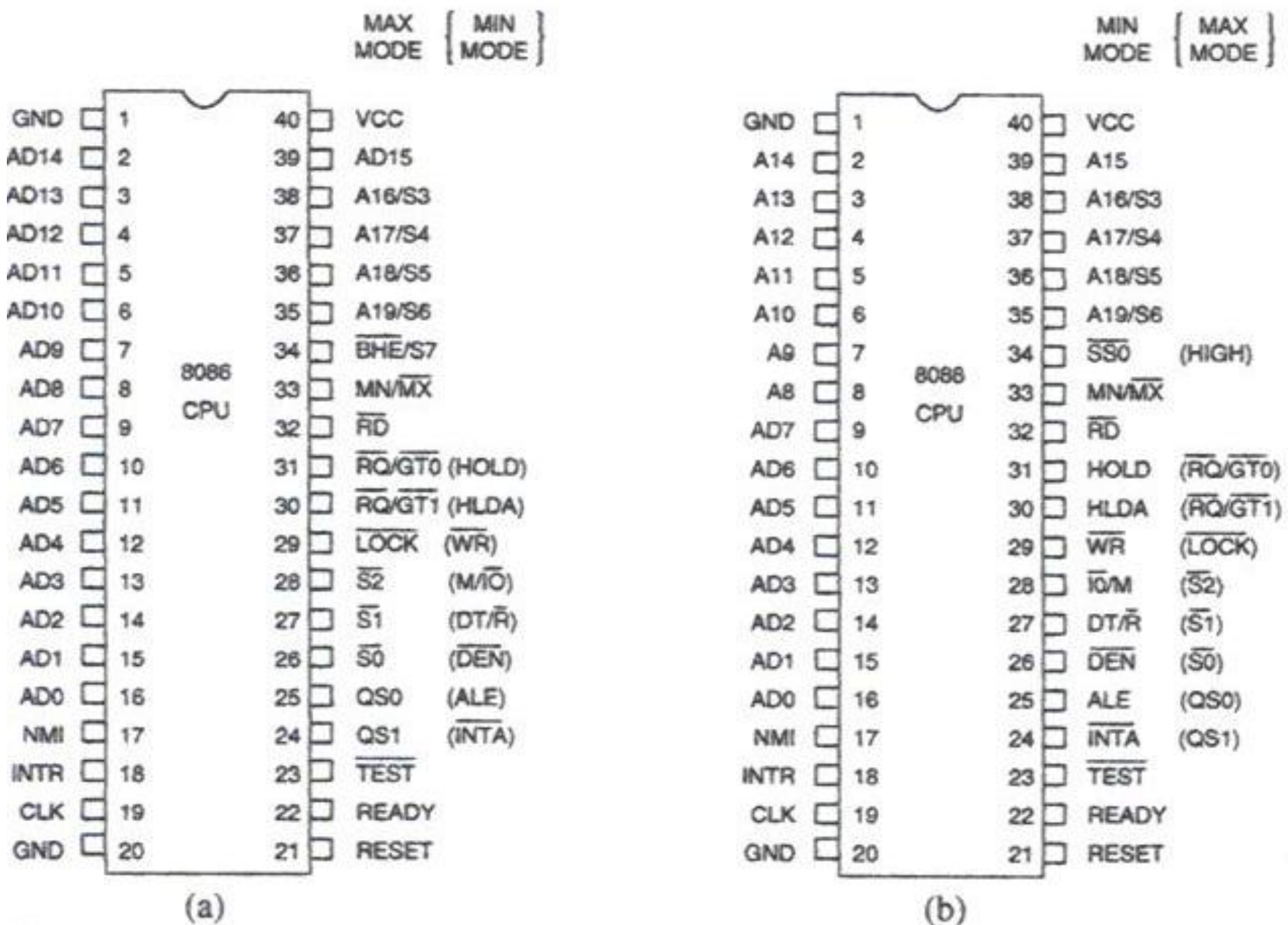
Pines del Procesador

AD7-ADO: 8088 address/data bus lines

A15-A8: 8088 high address bus

AD15-AD8: 8086 address/data bus lines

A19/S6-A16/S3: address/status bus bits



$\overline{RD}$: read signal

READY: This input is controlled to insert wait states into the timing of the microprocessor.

INTR: Interrupt request is used to request a hardware interrupt.

TEST: The Test pin is an input that is tested by the WAIT instruction. If TEST is a logic 0, the WAIT instruction functions as a NOP. If TEST is a logic 1, the WAIT instruction waits for TEST to become a logic 0. This pin is most often connected to the 8087 numeric coprocessor.

NMI: The non-maskable interrupt. If NMI is activated, this interrupt input uses interrupt vector 2.

RESET: Causes the microprocessor to reset itself if this pin is held high for a minimum of four clocking periods. After that, it begins executing instructions at memory location FFFFOH and disables future interrupts by clearing the IF flag bit.

CLK: The clock pin provides the basic timing signal to the microprocessor.

Vcc: This power supply input provides a +5.0 V, ±10 % signal to the microprocessor.

GND: Two pins labeled GND-both must be connected to ground for proper operation.

MN/MX: Minimum/maximum mode pin

$\overline{\text{BHE/S7}}$: The bus high enable pin is used in the 8086 to enable the most-significant data bus bits (D15-D8) during a read or a write operation. The state of S7 is always a logic 1.

Pines para el Modo Mínimo

$IO/\overline{M}$ or $M/\overline{IO}$: The IO/M (8088) or the M/IO- (8086) pin selects memory or I/0.

$\overline{WR}$: The write line is a strobe that indicates that the 8086/8088 is outputting data

$\overline{INTA}$: The interrupt acknowledge signal is a response to the INTR input pin. The pin is normally used to gate the interrupt vector number onto the data bus in response to an interrupt request.

ALE: Address latch enable

$DT/\overline{R}$: The data transmit/receive

DEN : Data bus enable activates external data bus buffers.

HOLD: The hold input requests a direct memory access (DMA).

HLDA: Hold acknowledge indicates that the 8086/8088 has entered the hold state.

$\overline{SS0}$: The SSO status line is equivalent to the SO pin in maximum mode operation of the microprocessor. This signal is combined with $IO/\overline{M}$ and $DT/\overline{R}$ to decode the function of the current bus cycle.

Pines en el Modo Máximo

$\overline{S2}, \overline{SI}$, and $\overline{SO}$: The status bits indicate the function of the current bus cycle. These signals are normally decoded by the 8288 bus controller described later in this chapter. Table 9-6 shows the function of these three status bits in the maximum mode.

$\overline{RO}/\overline{GT}1$ *and* $\overline{RO}/\overline{GT}0$: The request/grant pins request direct memory accesses (DMA) during maximum mode operation. These lines are bidirectional, and are used to both request and grant a DMA operation.

$\overline{LOCK}$: The lock output is used to lock peripherals off the system. This pin is activated by using the LOCK: prefix on any instruction.

QS1 and QS0: The queue status bits show the status of the internal instruction queue. These pins are provided for access by the numeric coprocessor (8087).

TABLE 9–5 Bus cycle status (8088) using $\overline{SS0}$.

IO/$\overline{M}$	*DT/$\overline{R}$*	*$\overline{SS0}$*	*Function*
0	0	0	Interrupt acknowledge
0	0	1	Memory read
0	1	0	Memory write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	I/O read
1	1	0	I/O write
1	1	1	Passive

TABLE 9–6 Bus control functions generated by the bus controller (8288) using $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$.

$\overline{S2}$	*$\overline{S1}$*	*$\overline{S0}$*	*Function*
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

TABLE 9–7 Queue status bits.

QS1	*QS0*	*Function*
0	0	Queue is idle
0	1	First byte of opcode
1	0	Queue is empty
1	1	Subsequent byte of opcode

Manejo de los Buses Multiplexados

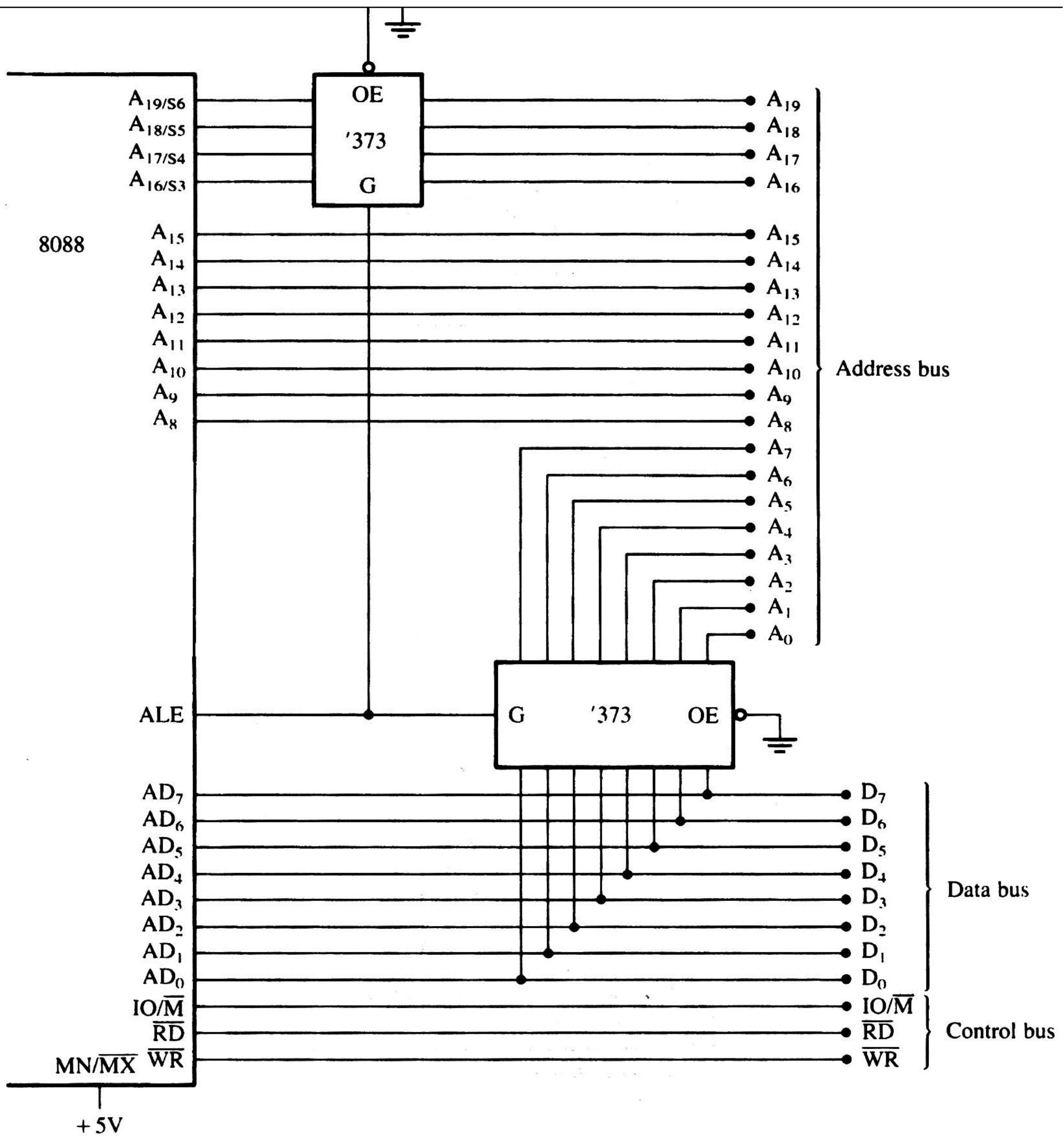


FIGURE 9–5 The 8088 microprocessor shown with a demultiplexed address bus. This is the model used to build many 8088-based systems.

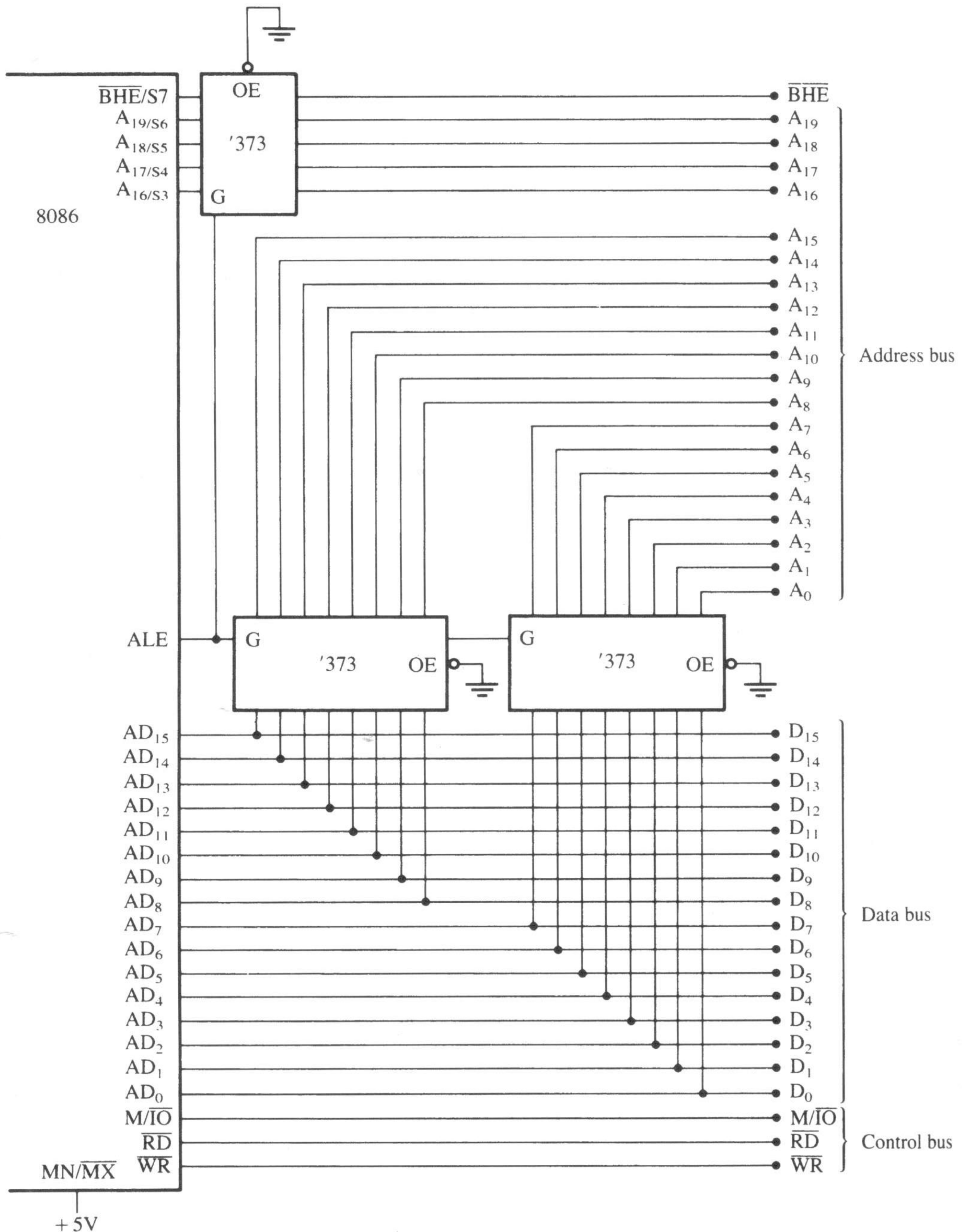


FIGURE 9–6 The 8086 microprocessor shown with a demultiplexed address bus. This is the model used to build many 8086-based systems.

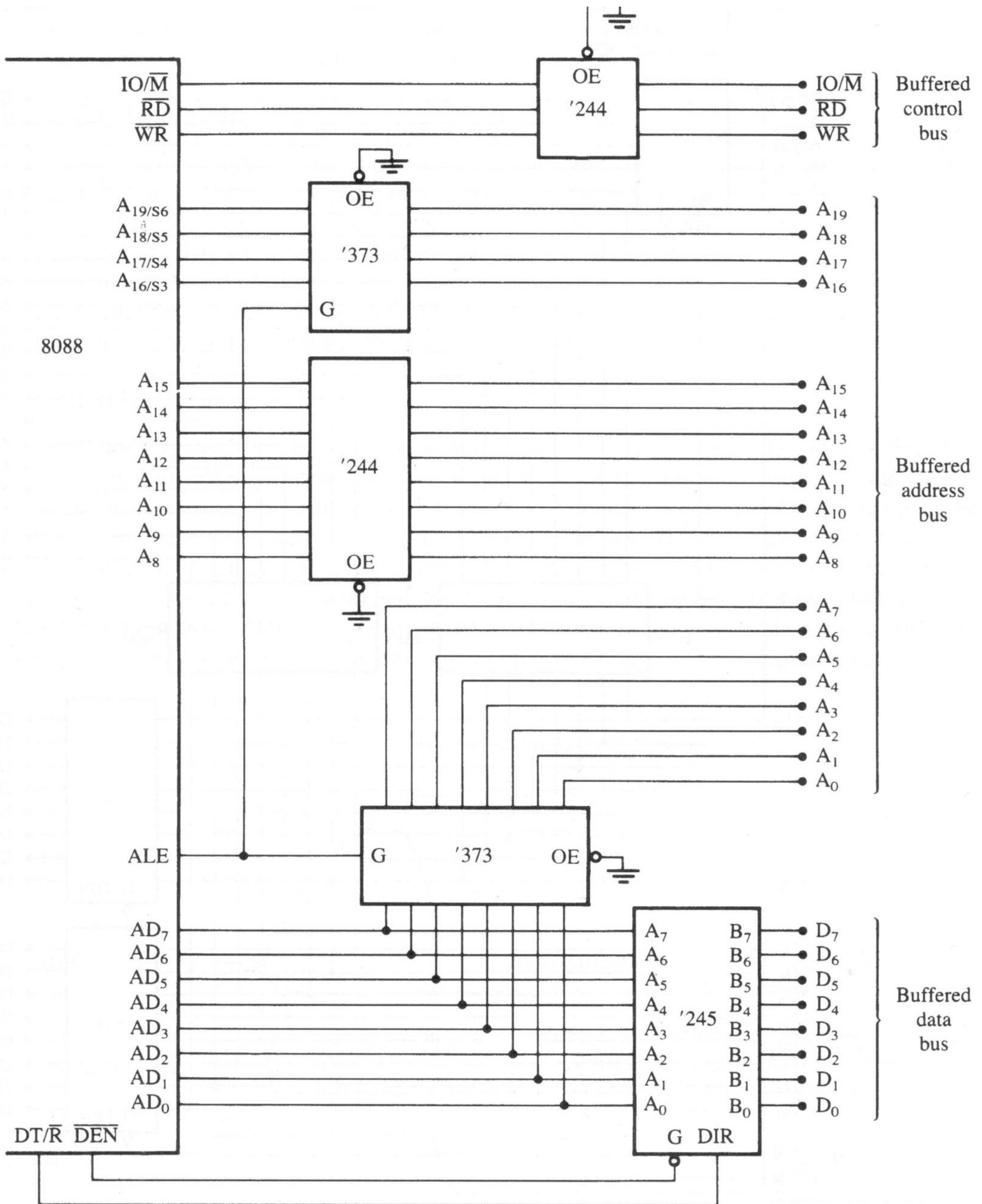


FIGURE 9–7 A fully buffered 8088 microprocessor.

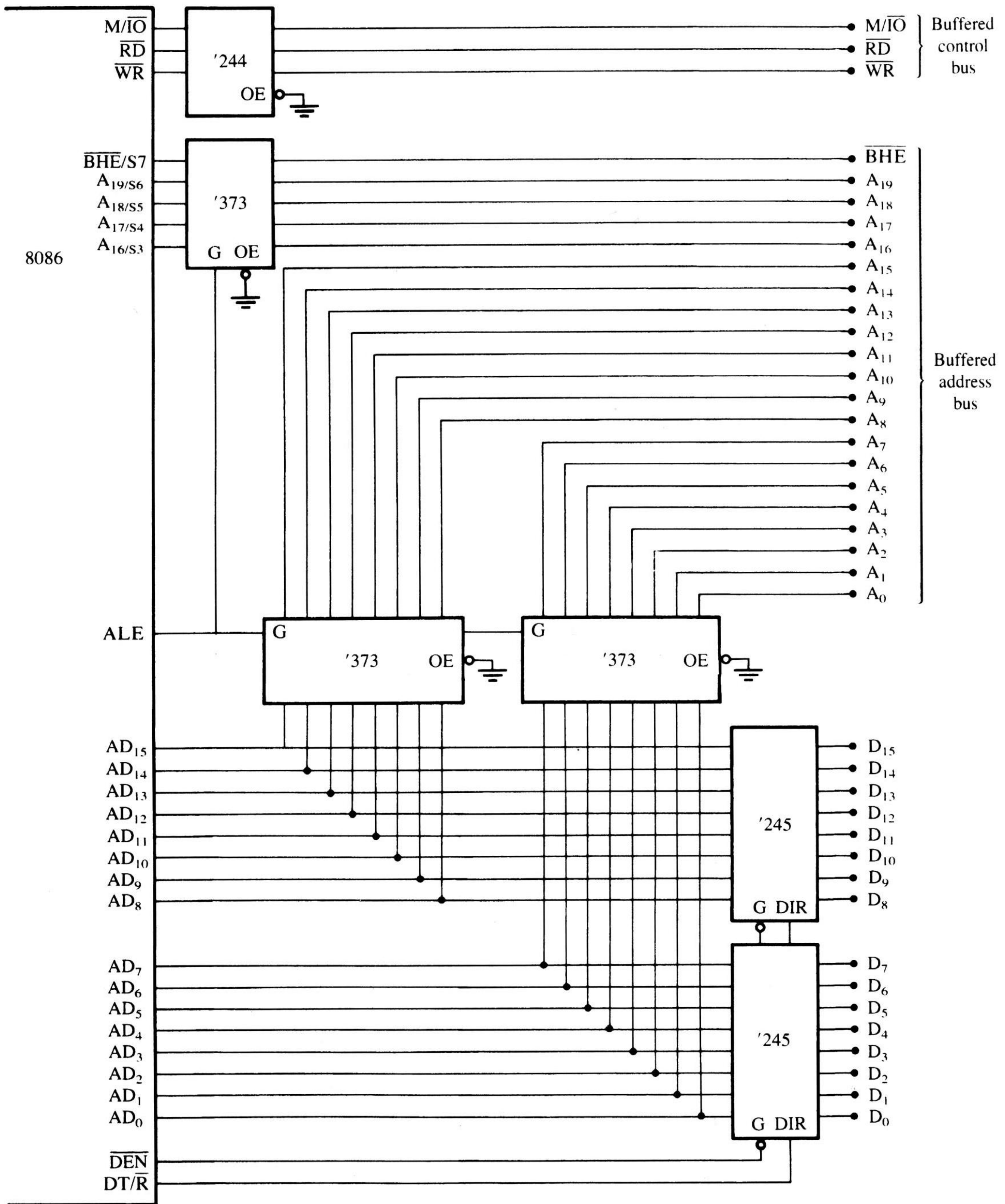


FIGURE 9–8 A fully buffered 8086 microprocessor.

Diagramas de Tiempo para las Operaciones de Lectura y Escritura

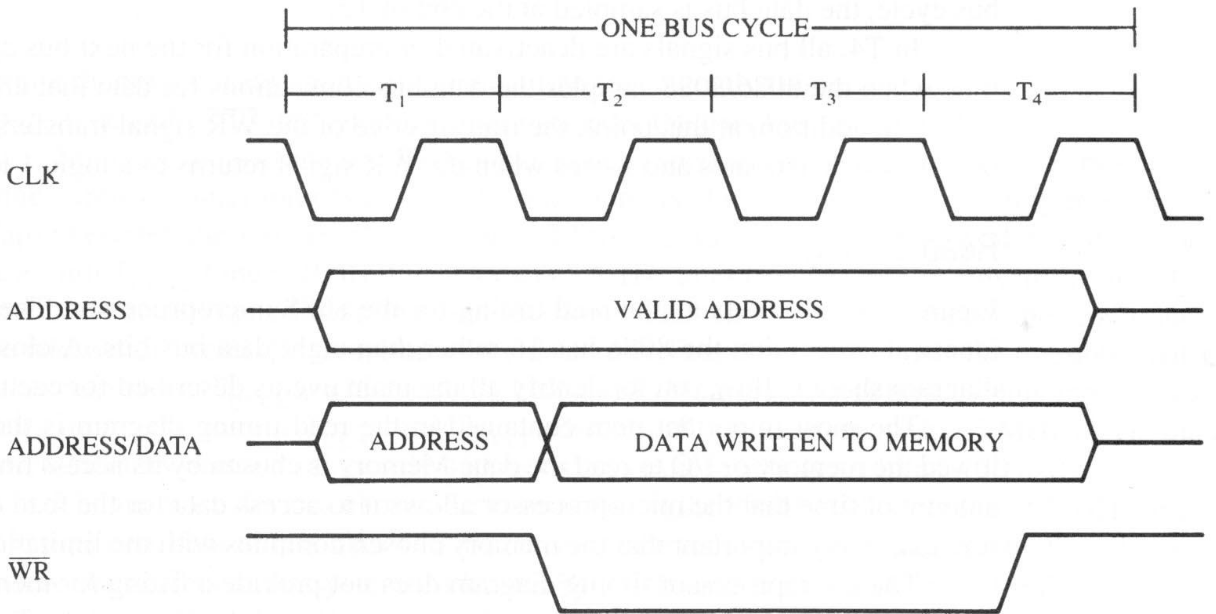


FIGURE 9–9 Simplified 8086/8088 write bus cycle.

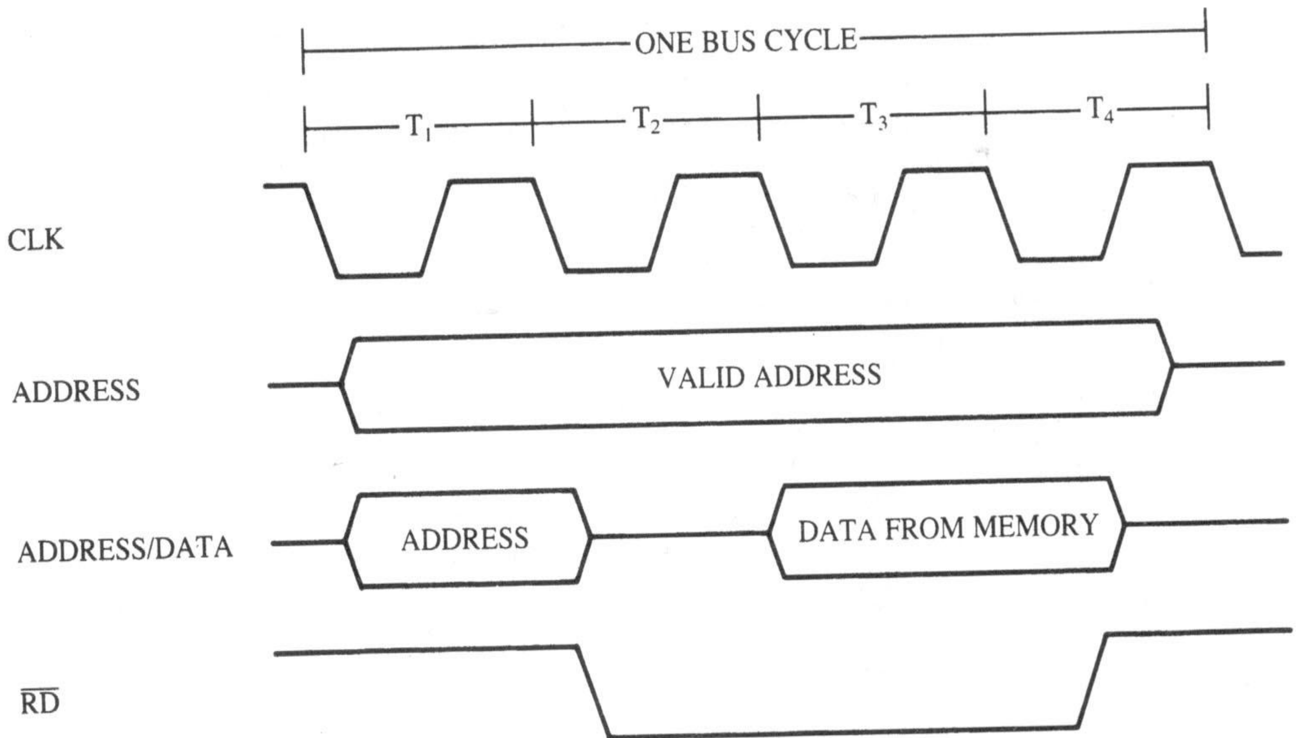


FIGURE 9–10 Simplified 8086/8088 read bus cycle.

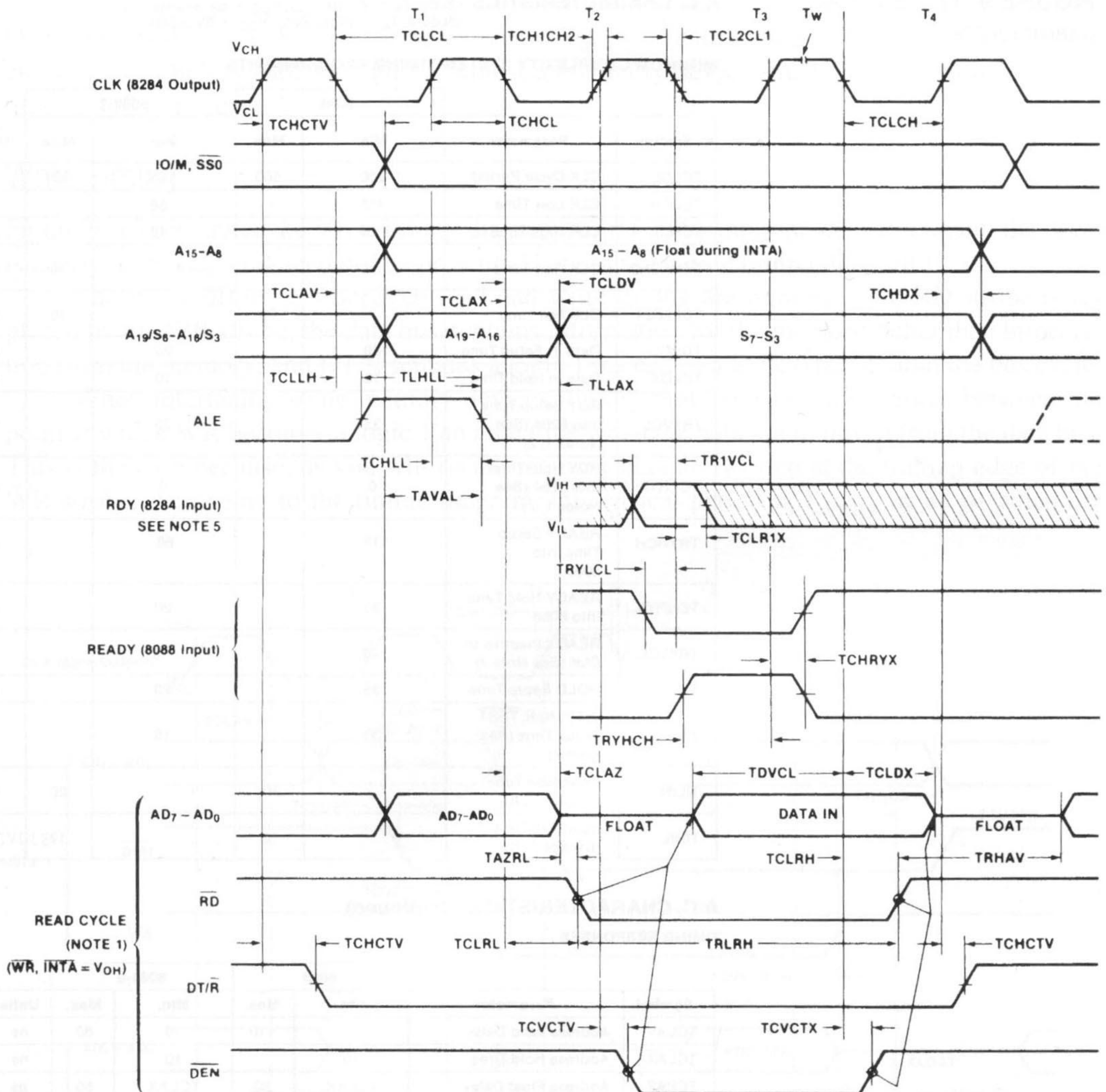


FIGURE 9–11 Minimum mode 8088 bus timing for a read operation.

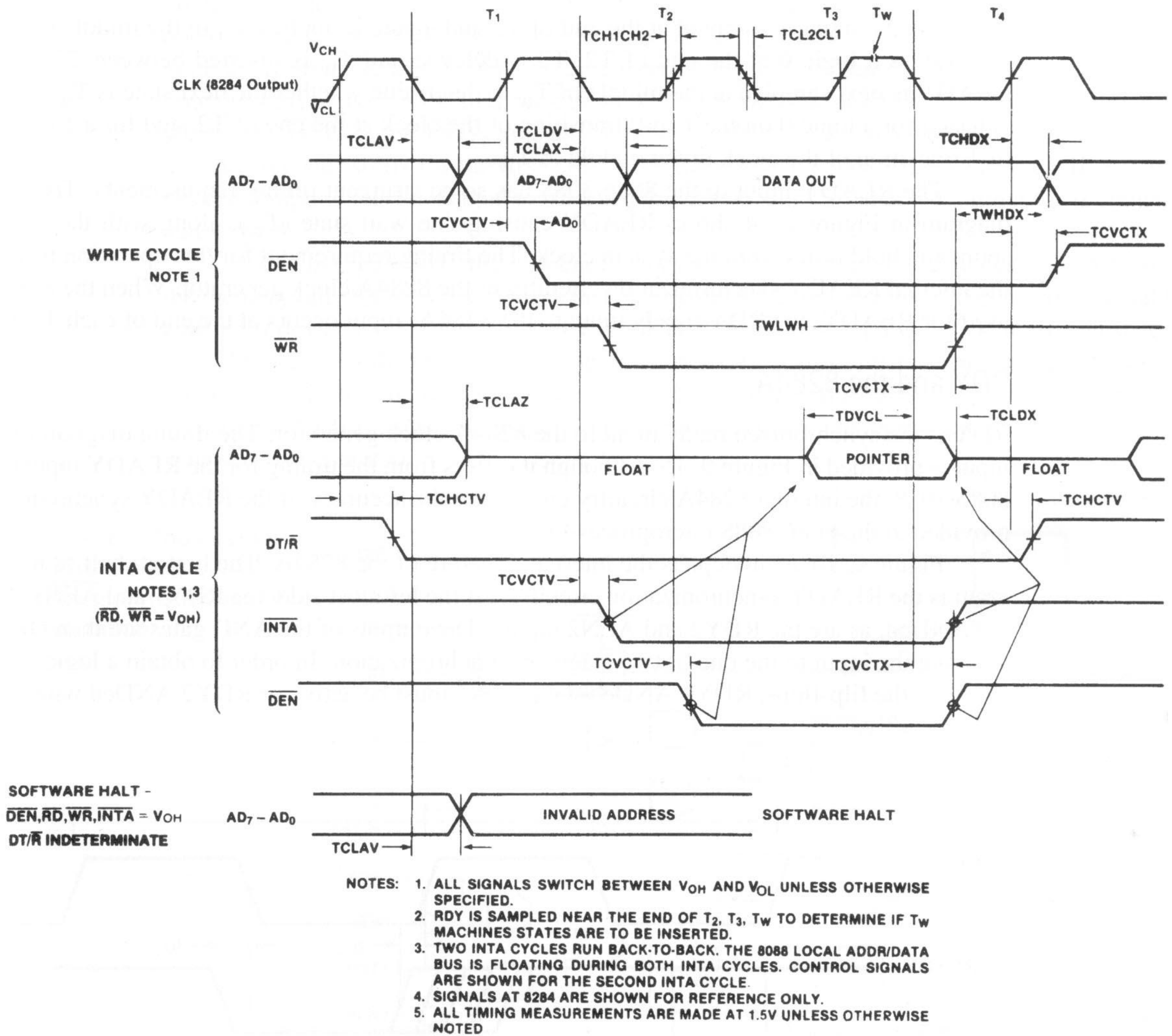


FIGURE 9–13 Minimum mode 8088 write bus timing.

Señal READY y Estado de Espera (Wait State)

- Un 0 en READY hace que se inserte un estado de espera Tw entre T2 y T3
- Tw se conoce como estado de espera y alarga el ciclo de ejecución de la instrucción.

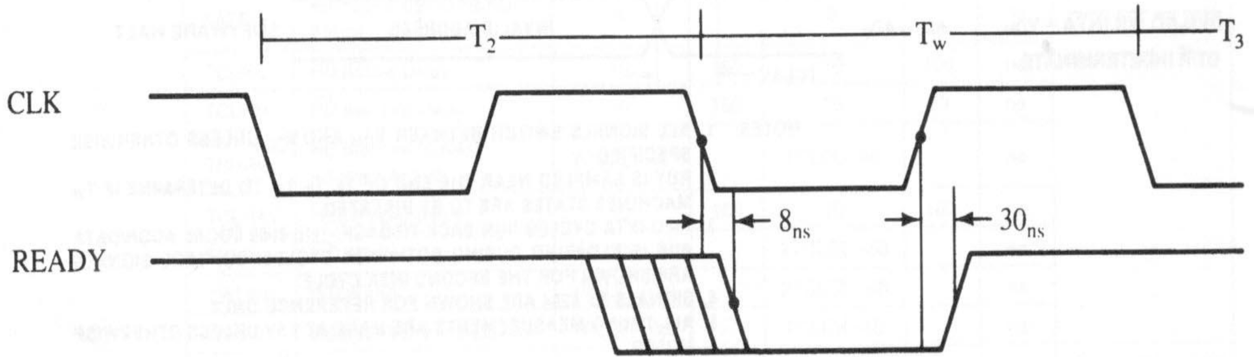


FIGURE 9–14 8086/8088 READY input timing.

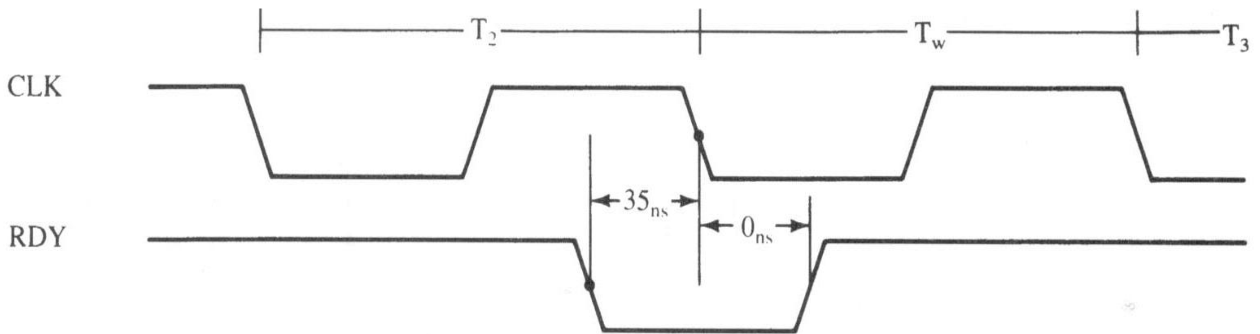


FIGURE 9–15 8284A RDY input timing.

Diagrama de un Sistema con el 8088 en Modo Mínimo

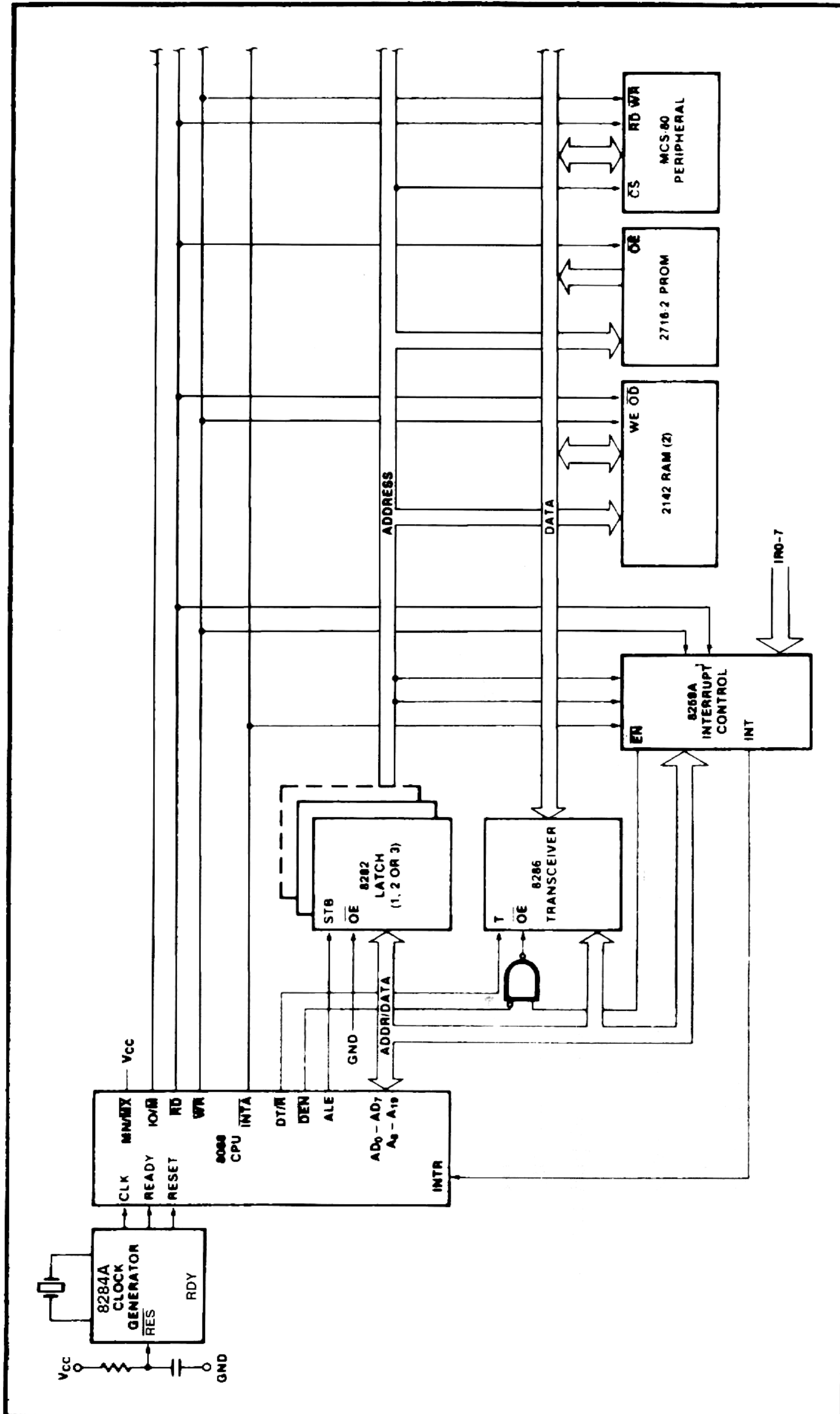


FIGURE 9–19 Minimum mode 8088 system.

Memorias

Características de la Memorias

- tamaño de palabra
- cantidad de localizaciones
- líneas de salida
- líneas de direccionamiento
- $\text{total localizaciones} = 2^{\text{total líneas de dirección}}$
- línea de lectura
- línea de escritura
- línea de habilitación (enable)

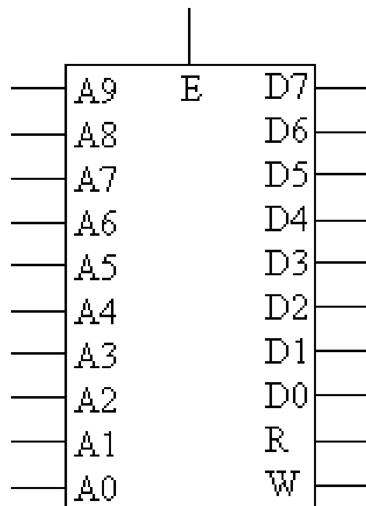


Figura 1. Diagrama de una memoria de 1KB

Operaciones de lectura y escritura

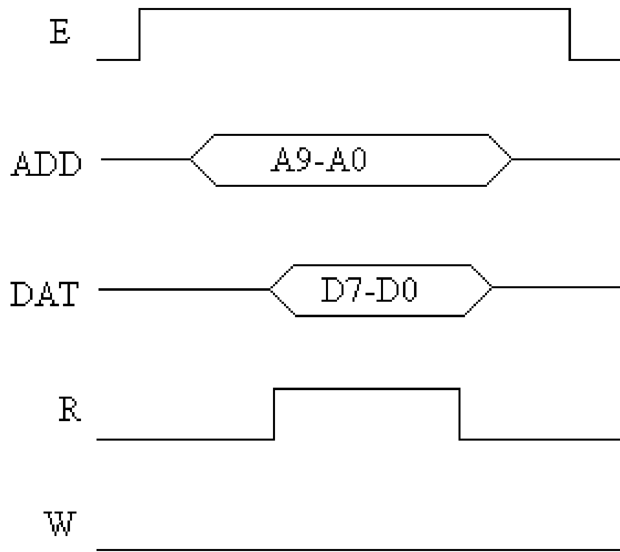
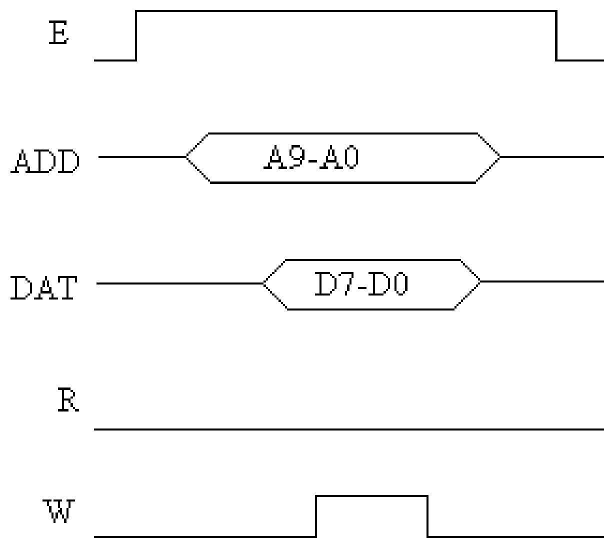


Figura 2. Operación de lectura de una memoria



Tipos de memoria

- RAM
- ROM
- PROM
- EPROM
- EEPROM (Flash Memory)

* Las memorias también se clasifican en volátiles y permanentes
** Algunas memorias poseen un bit extra de paridad

Combinación de 4 chips de 1KB para formar una memoria de 4KB

A11
A10
A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
A9 E D7
A8 D6
A7 D5
A6 D4
A5 D3
A4 D2
A3 D1
A2 D0
A1 R
A0 W
D7 D6 D5 D4 D3 D2 D1 D0 R W

Organización de la Memoria para Sistemas Basados en el Intel 8086

La organización de la memoria para el Intel 8086 varía con respecto al modelo que vimos anteriormente debido a que la misma utiliza bancos diferentes para las localizaciones de memoria pares e impares.

Para determinar cuál o cuáles bancos de memoria se activan, par o impar, es necesario tomar en consideración los valores de BHE y A0. La siguiente tabla indica como la combinación de estas dos señales indica el tamaño del valor que se almacenará y si se hará en una localización par o impar.

$\overline{BHE}$	A0($\overline{BLE}$)	Tamaño/Localización
0	0	word/par
0	1	byte/impar
1	0	byte/par
1	1	memoria inactiva

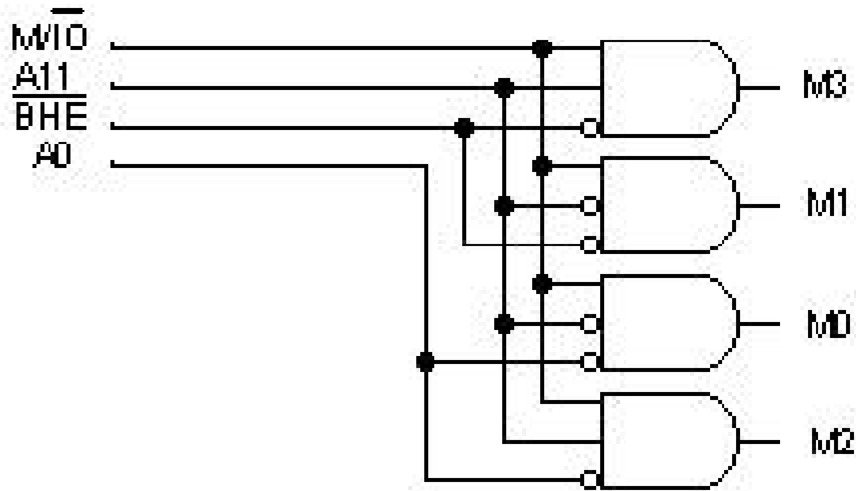
Recuerde que al almacenarse un word en una localización par se almacenan los 8 bits menos significativos en la localización par y los más significativos en la localización impar que le sucede.

Para mostrar como se organiza la memoria para un sistema basado en el Intel 8086 haremos un diseño de ejemplo. En este diseño construiremos un sistema de memoria de 4 KB utilizando módulos de 1 KB. Se mostrará el decodificador, los módulos de memoria y sus conexiones con las líneas de los buses.

Tabla de la Verdad para el Diseño del Decodificador

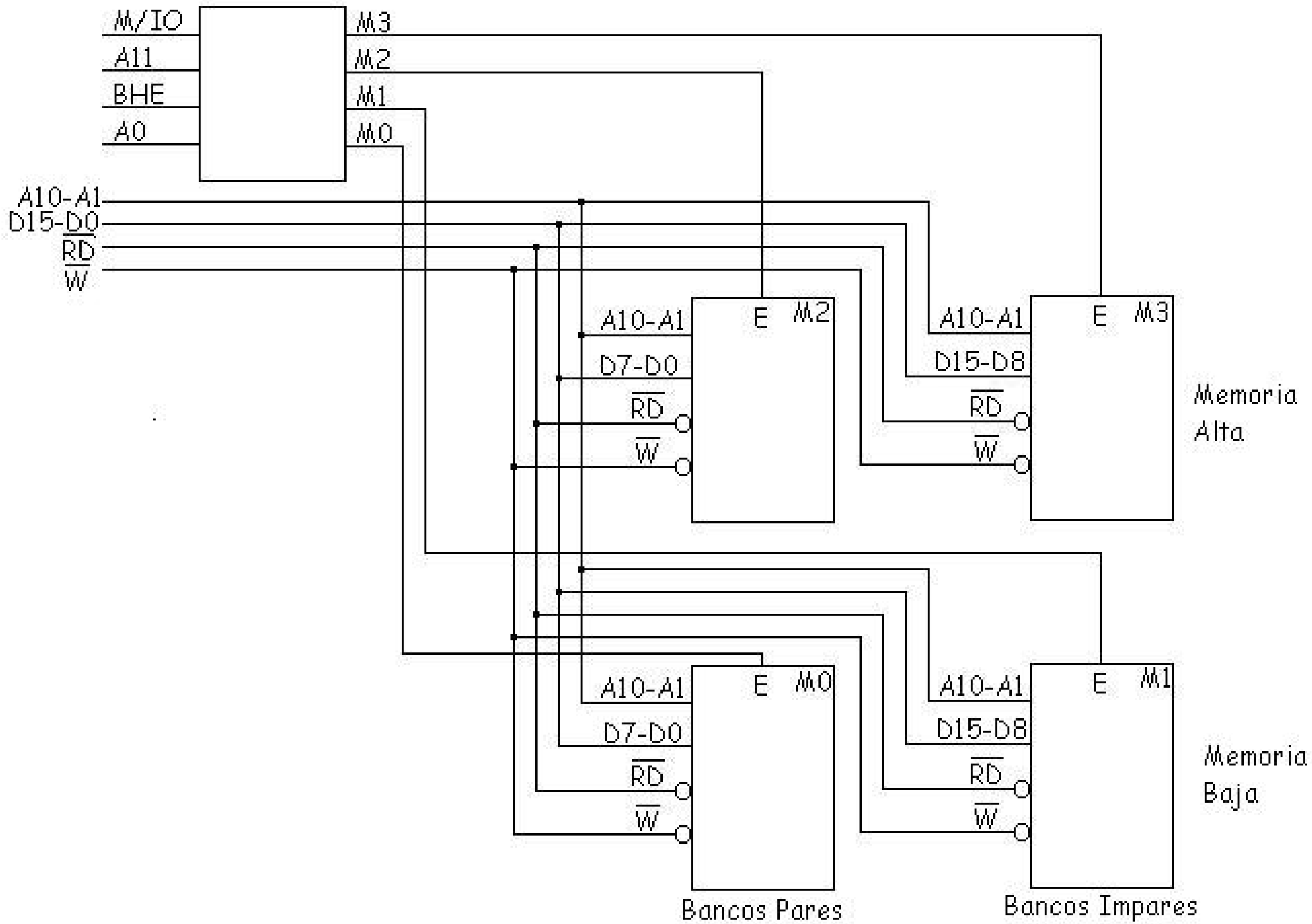
$M/\overline{IO}$	A11	$\overline{BHE}$	A0	M3	M2	M1	M0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	1	1
1	0	0	1	0	0	1	0
1	0	1	0	0	0	0	1
1	0	1	1	0	0	0	0
1	1	0	0	1	1	0	0
1	1	0	1	1	0	0	0
1	1	1	0	0	1	0	0
1	1	1	1	0	0	0	0

Diseño del Decodificador



Decodificador para Controlar la Activación de los Módulos de Memoria

Organización de una Memoria de 4KB Utilizando Módulos de 1KB



Diseño de Prueba para Estudiar el Manejo de las Memorias en LogicWorks