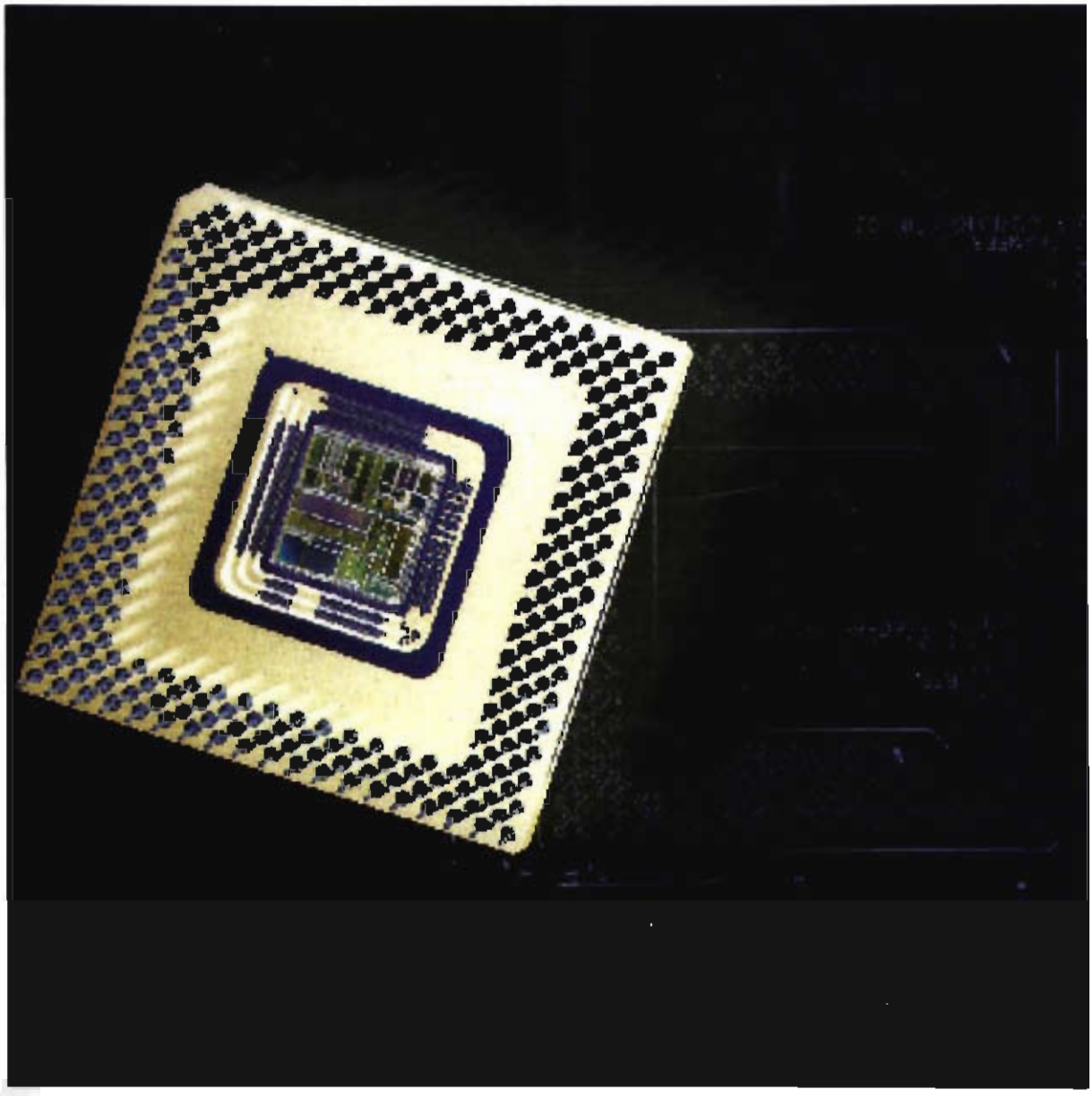
29

Organización de máquinas digitales I

José Ignacio Vega Luna, Gerardo Salgado Guzmán y Roberto Sánchez González



UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco

José Ignacio Vega Luna es ingeniero electrónico y maestro en Ciencias de la Computación de la UAM-Azcapotzalco. En la misma institución ha impartido materias de circuitos lógicos, sistemas digitales y organización de máquinas digitales. En la Universidad Nacional de Ingeniería de Managua, Nicaragua, impartió cursos de microprocesadores. En la actualidad sus áreas de trabajo son las siguientes: redes de computadoras, sistemas operativos y aplicaciones de microprocesadores y controladores.

Gerardo Salgado Guzmán realizó estudios de Ingeniería Electrónica en la UAM-Azcapotzalco. En esta universidad ha impartido cursos de circuitos lógicos, sistemas digitales, laboratorio de sistemas digitales y laboratorio de máquinas digitales. Actualmente cursa estudios de maestría en Ciencias de la Computación. Sus áreas de trabajo son: redes novell y aplicaciones de microprocesadores y microcontroladores.

Roberto Sánchez González cursó estudios de Ingeniería Electrónica en la UAM-Azcapotzalco. Ha impartido cursos de circuitos lógicos, sistema digitales, organización de máquinas digitales y laboratorio de máquinas digitales en la misma universidad. Actualmente sus áreas de trabajo son: redes de computadoras, sistemas operativos y aplicaciones de microprocesadores y microcontroladores.

ORGANIZACIÓN DE MÁQUINAS DIGITALES I

Colección
Libros de Texto y Manuales de Práctica

Serie
Material de apoyo a la docencia
(Teoría y prácticas de laboratorio; problemarios)

Organización de máquinas digitales I

José Ignacio Vega Luna
Gerardo Salgado Guzmán
Roberto Sánchez González

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo
Azcapotzalco

Portada
Virginia Flores/Sans Serif Editores

Composición tipográfica, diseño, producción y cuidado editorial
Sans Serif Editores, telfax 674 60 91

Primera edición 1998

ISBN: 970-654-075-X

Impreso en México
Printed in Mexico

PRÓLOGO

El objetivo de este trabajo es apoyar el proceso de enseñanza-aprendizaje de la materia Organización de Máquinas Digitales I correspondiente a la carrera de Ingeniería Electrónica de la UAM-Azcapotzalco, en la que se aborda el estudio de dos microprocesadores de 16 bits (el 8086 de Intel y el MC68000 de Motorola), así como de un microcontrolador (el 8051 de Intel). Estos temas comprenden todo el plan de estudios de la materia. Se trata de proporcionar al estudiante una guía, un complemento de lo enseñado por el profesor en el aula.

Se consultaron diversos títulos sobre el tema. Las tablas de instrucciones y algunas figuras del microprocesador 8086 fueron tomadas de *The 8086/88 Family* de John Uffenbeck, por considerar que reflejan la esencia y el funcionamiento de las instrucciones tratadas.

Finalmente, deseamos agradecer al profesor Raymundo Lira, jefe del Área de Sistemas Digitales, el apoyo material y moral para la realización de este trabajo, así como a los estudiantes de Ingeniería Electrónica las observaciones y comentarios sobre el contenido, y de manera muy especial a Rocío de los Ángeles Hernández Loyo su ayuda invaluable en todo momento para elaborar y concluir el mismo.

LOS AUTORES

PROGRAMA DEL CURSO

EL MICROPROCESADOR 8086/88

- Familia de microprocesadores de 16 bits de Intel
- Arquitectura y set de instrucciones
- Técnicas de programación
- El módulo de la CPU del 8086
- Sistemas multiprocesador

EL MICROCONTROLADOR 8051

- La familia MCS-51
- Arquitectura interna
- Registros
- Puerto serie
- Interrupciones

EL MICROPROCESADOR MC68000

- Arquitectura interna
- Operación y temporización del bus
- Procesamiento de excepciones
- Modos de direccionamiento
- Interfaz con los periféricos de la familia 6800

MICROPROCESADORES-TÓPICOS

1. Arquitectura de la CPU

- Registros internos
- Banderas
- Set de instrucciones

2. Interface eléctrica

- Bus de datos
- Bus de direcciones
- Bus de control
- Circuito de reloj

a) Arquitectura de la CPU 8086/88
b) Memoria segmentada
c) Modos de direccionamiento
d) Instrucciones de transferencia de datos
e) Instrucciones para el manejo de cadenas *(strings)*
f) Instrucciones lógicas
g) Instrucciones aritméticas
h) Instrucciones de transferencia de control
i) Instrucciones de control del procesador

CAPÍTULO I

EL MICROPROCESADOR 8086/88

FAMILIA DE MICROPROCESADORES DE 16 BITS

- Intel: 5 microprocesadores.
  5 coprocesadores.

Ejemplos:
iAPX 86/21-Sistema de tres circuitos integrados basado en el microprocesador 8086, el coprocesador numérico de datos (NDP) 8087 y el procesador de entrada/salida (IOP) 8089.
8087 & 8089 expanden las capacidades del 8086:

- Con el 8087, el 8086 realiza operaciones matemáticas cien veces más rápido.
- 8088 idéntico al 8086 pero puede leer de memoria sólo 8 bits a la vez.
- 805186 & 805188 *upgrades* del 8086 y el 8088 e incorporan de 1 a 20 componentes en un circuito integrado.
- 80286 multiusuarios y *multitasking* (ejecución de varios programas a la vez).
- 386
- 80287 procesador numérico de datos para el 80286 en la versión APX 286/20.
- 805130 & 80510 procesadores de sistemas operativos (OSP's): iRMX86 *Operating System* y CP/M86.

El 8088 es un microprocesador que tiene un bus de datos interno de 16 bits y un bus de datos externo de 8 bits. Requiere dos operaciones de lectura a memoria para obtener la misma información que lee el 8086, por lo que se deduce que el 8088 opera menos eficientemente que el 8086.

El 8086 y el 805186 leen datos de la memoria organizada en dos bancos: uno para los bytes con dirección par y otro para los bytes con dirección impar. En esta forma el microprocesador puede leer (o escribir) un byte con dirección par y otro con dirección impar simultáneamente.

Para leer una instrucción de 6 bytes, un microprocesador de 16 bits requerirá 3 ciclos de lectura a memoria, mientras que uno de 8 bits requerirá 6 ciclos. Los ciclos adicionales alentan al microprocesador.

CUADRO I.1. *Ejemplos de configuración de sistemas para microprocesadores de 16 bits de la familia Intel*

	Microprocesadores					*Coprocesadores*				
Nombre del sistema	*8086*	*8088*	*80186*	*80188*	*80286*	*8087*	*8089*	*80130*	*80150*	*80287*
iAPX86/10	1									
iAPX86/11	1						1			
iAPX86/12	1						2			
iAPX86/20	1					1				
iAPX86/21	1					1	1			
iAPX86/22	1					1	2			
iAPX86/30	1							1		
iAPX86/40	1					1		1		
iAPX86/50	1								1	
iAPX88/10		1								
iAPX88/11		1					1			
iAPX88/12		1					2			
iAPX88/20		1				1				
iAPX88/21		1				1	1			
iAPX88/22		1				1	2			
iAPX88/30		1						1		
iAPX88/40		1				1		1		
iAPX88/50		1							1	
iAPX186/10			1							
iAPX186/11			1				1			
iAPX186/20			1			1				
iAPX186/21			1			1	1			
iAPX186/30			1					1		
iAPX186/40			1			1		1		
iAPX186/50			1						1	
iAPX188/10				1						
iAPX188/11				1			1			
iAPX188/20				1		1				
iAPX188/21				1		1	1			
iAPX188/30				1				1		
iAPX188/40				1		1		1		
iAPX188/50				1					1	
iAPX286/10					1					
iAPX286/20					1					1

8086 Microprocesador de 16 bits.
8088 Idéntico al 8086 excepto por el bus de datos de 8 bits.
80186 Microprocesador de 16 bits de alta integración. Incluye generador de reloj, controlador de interrupciones programable, dos canales de DMA, tres temporizadores programables de 16 bits y lógica programable de selección de chip.
80188 Idéntico al 80186 excepto por el bus de datos de 8 bits.
80286 8086 de alto rendimiento con espacio de direccionamiento de 16M-bytes. Manejo de memoria integrado y soporte para memoria virtual y sistemas de operación.
8087 Procesador numérico de datos. Usado con el 8086/88 y el 80186/188 para reducir hasta en cien veces el tiempo de procesamiento de funciones matemáticas complejas.
8089 Procesador de entrada/salida. Usado con el 8086/88 y el 80186/188 para proporcionar capacidades de DMA de alta velocidad.
80130 Procesador para el sistema operativo iRMX 86 que contiene muchas de las funciones primitivas del sistema operativo.
80150 Igual que el 80130, pero éste soporta el sistema operativo CP/M 86.
80287 Igual que el 8087, sólo que éste soporta microprocesador 80286.

ARQUITECTURA DE LA CPU 8086/88

Funciones

- Generar todas las señales de sincronización del sistema.
- Sincronizar la transferencia de datos entre memoria, dispositivos de entrada/salida y elementos internos.

La BIU *(bus interface unit)* ejecuta todas las funciones de hardware para la transferencia de datos entre el mundo exterior y la EU *(execute unit)*, que recibe los códigos de operación y los datos de la BIU, ejecuta esas instrucciones y almacena los resultados en los registros de propósito general.

- 8088 BIU de 8 bits; queue de 4 bytes.
- 8086 BIU de 16 bits; queue de 6 bytes.

La EU es la misma en los dos, por lo que los programas pueden correr en ambos sin cambio alguno.

Overlap

Puesto que la CPU tiene la BIU separada de la EU, se pueden realizar ciclos de búsqueda *fetch* y de ejecución.

¿Por qué?

1. La BIU saca el IP *(instruction pointer register)* en el bus de direcciones, para leer el byte o palabra seleccionado.
2. El IP = IP + 1.
3. El código de la instrucción pasa a la queue (registro *first-in first-out*).
4. Si inicialmente la queue está vacía, la EU saca inmediatamente el código y comienza la ejecución.
5. Mientras la EU ejecuta la instrucción, la BIU busca un nuevo código. Dependiendo del tiempo de ejecución de la primera instrucción, la BIU puede llenar la queue antes de que la EU esté lista para ejecutar otra instrucción.

La BIU buscará nuevos códigos siempre que la queue tenga lugar para 1 (en el 8088) o 2 (en el 8086) bytes. Entonces la EU puede ejecutar instrucciones continuamente, sin esperar a que la BIU entregue un nuevo código.

La EU entra en un modo *wait* cuando la instrucción accesa a una localidad de memoria cuyo contenido no esté en la queue o bien cuando la instrucción es un salto (los bytes en la queue son descartados).

La BIU espera cuando la ejecución de una instrucción tarda (es lenta).

Ejemplo:
AAM (ASCII *Adjust for multiplication*).
La instrucción tarda 86 ciclos de reloj.
La BIU espera a que la EU saque 1 o 2 bytes de la queue.

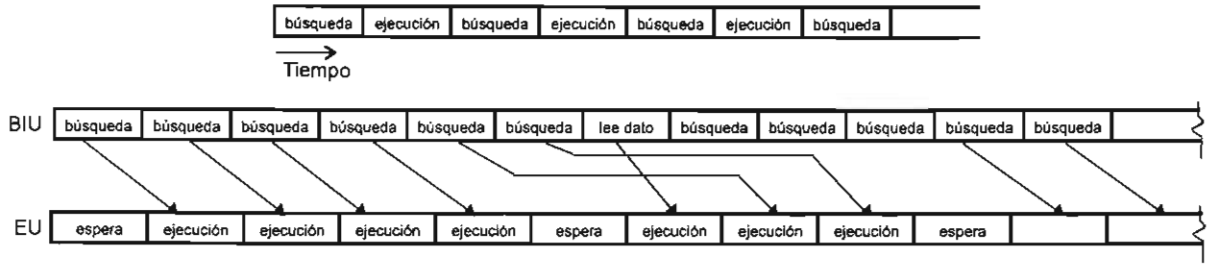


Figura I.1. Modelo pipe-line del 8086/88.

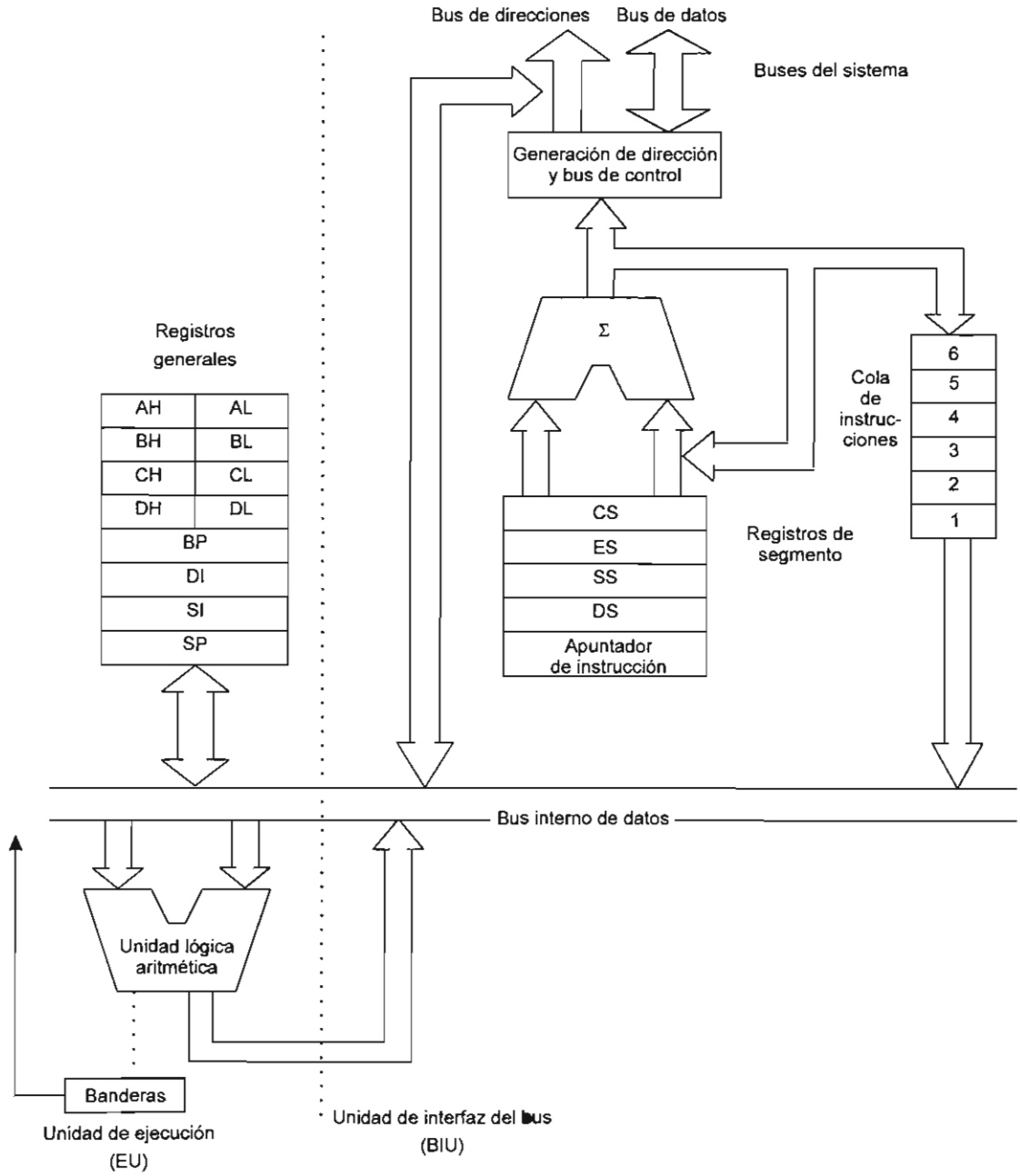


Figura I.2. Diagrama de bloques interno de una CPU 8086.

REGISTROS INTERNOS DE LA CPU

Grupo de datos: acumulador, BX, CX, DX

Pueden accesarse como bytes o palabras y se usan para almacenar temporalmente resultados.

Grupo de apuntadores e índices

Registros de 16 bits.

Apuntadores a memoria.

Ejemplo:
La instrucción `MOV AH, [SI]`

3000H	AF
3001H	15
3002H	80
3003H	25
3004H	C6
3005H	F2
3006H	70
3007H	26

Si SI = 3000H
entonces AH = AFH.

Algunas veces un registro apuntador se utilizará como apuntador a un byte de memoria y otras veces a una palabra *(word)* de memoria.

Ejemplo:
Si se utiliza [SI + 1:SI] = 15AFH.

```
MOV AH, BYTE PTR [SI]
MOV AX, BYTE PTR [SI]
```

Cuando se almacenan palabras (16 bits) en memoria, el byte bajo se almacena en la localidad de memoria de menor orden.

El registro IP *(instruction pointer)* es parte de la BIU y el programador no tiene control directo sobre él.

Registro de status y banderas

	$FLAGS_H$							$FLAGS_L$							
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF
15															0

CUADRO I.2. *Descripción de los bits del registro de banderas*

Regs.	*Bandera*	*Resumen*
CF	*Carry Flag*	Valor del bit de acarreo saliente.
PF	*Parity Flag*	Puesta a "1" si el número de 1's del byte del resultado es un número par.
AF	*Auxiliar Carry Flag*	Acarreo del bit 3 al bit 4 del registro AL
ZF	*Zero Flag*	Puesta a "1" si el resultado es cero.
SF	*Sign Flag*	Toma el valor del bit más significativo del resultado (0-positivo, 1-negativo).
TF	*Trap Flag o Single Step Flag*	Cuando es puesta a "1", ocurre una interrupción después de la instrucción que la activa.
IF	*Interrupt Enable Flag*	Habilita las interrupciones mascarables.
DF	*Direction Flag*	En instrucciones de manejo de cadenas o bloques, el registro índice usado se "autodecrementa" si la bandera está puesta a "1"; si es puesta a cero el registro índice se "autoincrementa".
OF	*Overflow Flag*	Es puesta a "1" si el resultado no se puede expresar con el número de bits del operando destino. Se usa en operaciones con números signados.

Ejemplo:

Si AL = 82H

después de ejecutar la instrucción:

```
ADD AL, 1
AL = 83H = 1000 0011

CF = 0; PF = 0 ; AF = 0
ZF = 0; SF = 1; OF = 1    (el resultado excede +127)
```

El 8086/88 tiene instrucciones para transferir el control del programa dependiendo del estado de las banderas.

Ejemplo:

```
ADD AL, 1
JNZ 0300H
```

Las banderas que pueden modificarse por el programador y que se usan para controlar la operación del procesador son TF, IF, DF.

Cuando TF está activada, el control se pasa a una dirección especial después de la ejecución de cada instrucción. Esta bandera es utilizada para "debuggear" un programa.

Con IF el control se transfiere a una rutina de servicio, la cual termina con IRET.

Grupo de segmento

La BIU usa este tipo de registros para determinar la dirección de la localidad de memoria a accesar.

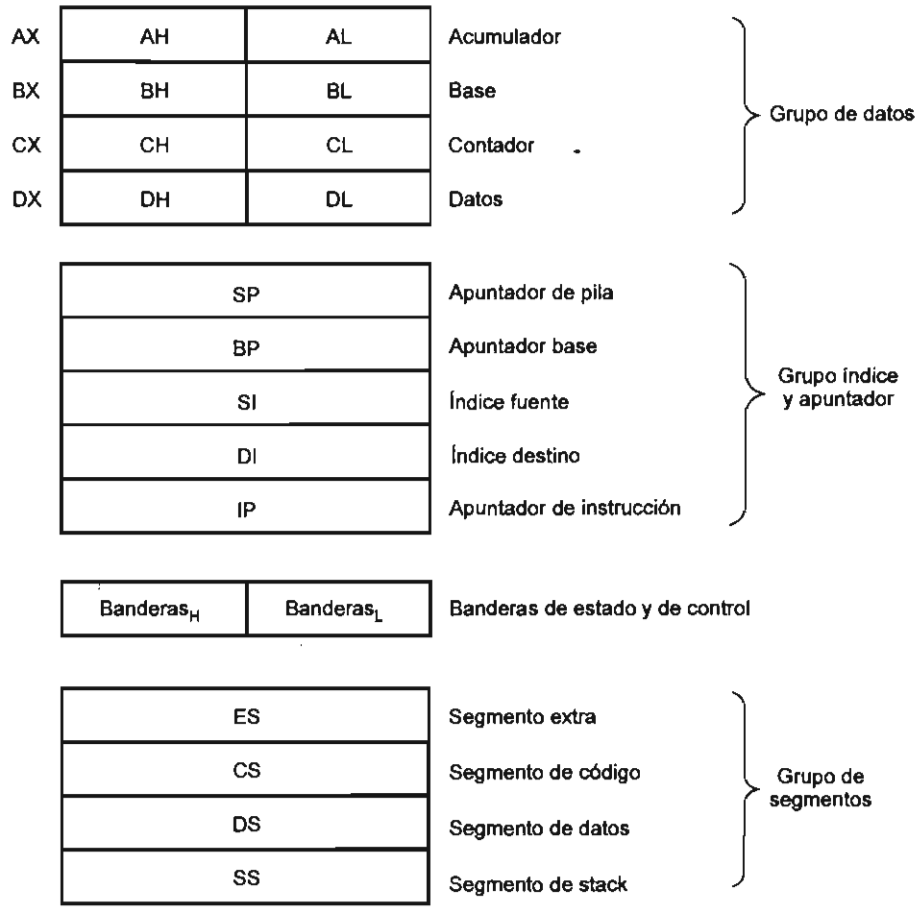


Figura I.3. Registros internos del 8086/88.

Memoria segmentada

El 8086/88 divide la memoria en segmentos.

El microprocesador 8086 es de 16 bits y el bus de datos será de 16 bits; sin embargo puede trabajar con memorias de 8 bits, ya que algunos códigos de operación son de 8 bits, al igual que muchos dispositivos de entrada/salida.

Bus de direcciones = 20 bits = 2^{20} = 1 048 576 bytes = 524 287 palabras del 8086/88

El 8086 lee 16 bits de memoria simultáneamente; organiza la memoria en un banco de direcciones pares y en un banco de direcciones impares.

El 8088 tiene un bus de datos de 8 bits y se interfaza a 1MB de memoria con un solo banco. Disminuye el "performance".

El 8086 lee 16 bits de memoria simultáneamente, leyendo un byte con dirección impar y uno con dirección par. Si se lee una palabra almacenada en los bytes 5 y 6, la CPU debe realizar dos ciclos de lectura a memoria: uno para buscar el byte de bajo orden y otro para el de alto orden.

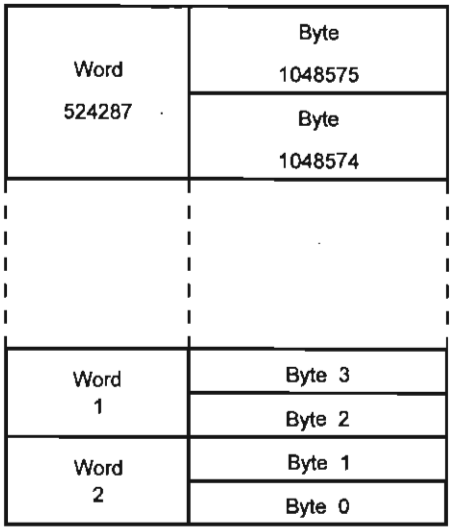


Figura I.4. Organización de bancos de direcciones en el 8086.

Mapa de memoria

El espacio de memoria del 8086/88 se divide en 16 bloques de 64K cada uno.

Localidades reservadas = productos futuros de hardware/software.

Localidades dedicadas = procesamiento de interrupciones y reset.

Registros de segmento:

El 8086/88 define 4 segmentos de 64K cada uno:

a) Segmento de código = códigos de instrucciones del programa.

b) Segmento DATA = datos del programa.

c) Segmento EXTRA = datos compartidos.

d) Segmento de stack = "dirección de regreso" de interrupciones y subrutinas.

Microprocesadores de 8 bits, cantidad limitada de memoria. CS, DS, ES y SS apuntan a la localidad 0 (dirección base) de cada segmento.

Si bus de direcciones = 20 bits

y los registros de segmento = 16 bits

¿qué pasa entonces?

La BIU le adiciona ceros a los 4 bits menos significativos del registro de segmento; esto asegura que la dirección sea divisible por 16.

```
CS =  1300H + 0H  =  13000H
SS =  1300H + 0H  =  13000H
```

Los segmentos no necesitan definirse separadamente. Las localidades de memoria no se pueden accesar sin haber definido el registro de segmento correspondiente, el cual se define sólo vía

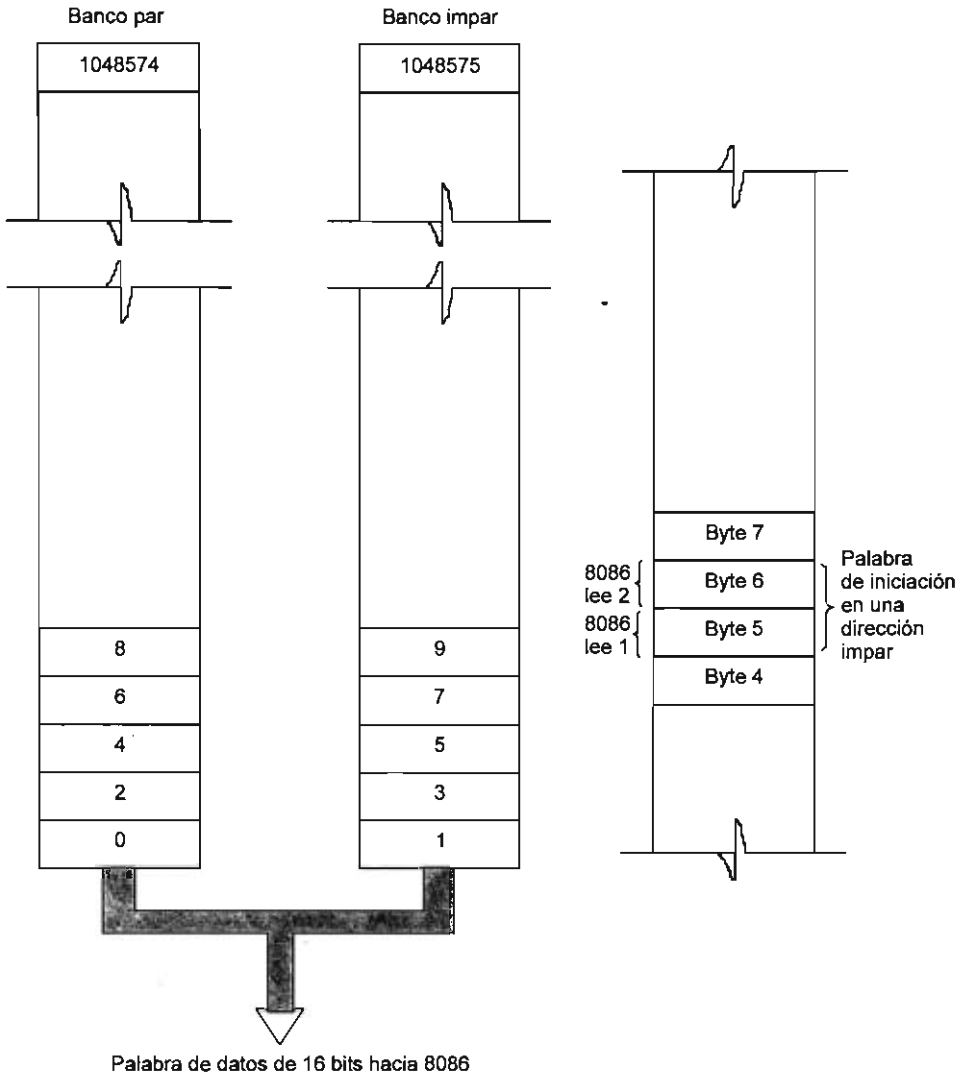


Figura I.5. Organización de memoria en bancos pares e impares.

software, generalmente en las primeras instrucciones del programa. Pueden utilizarse máximo 256K (64K × 4) bytes de memoria.

Direcciones lógicas y físicas

Las direcciones en un segmento van desde la 0000H hasta la FFFFH. A una dirección dentro de un segmento se le llama *offset* o *dirección lógica*.

Ejemplo:
La dirección 0007H en el segmento de código corresponde a la dirección real:

```
B3FF0H + 7 = B3FF7H
```

que es la dirección real o física,

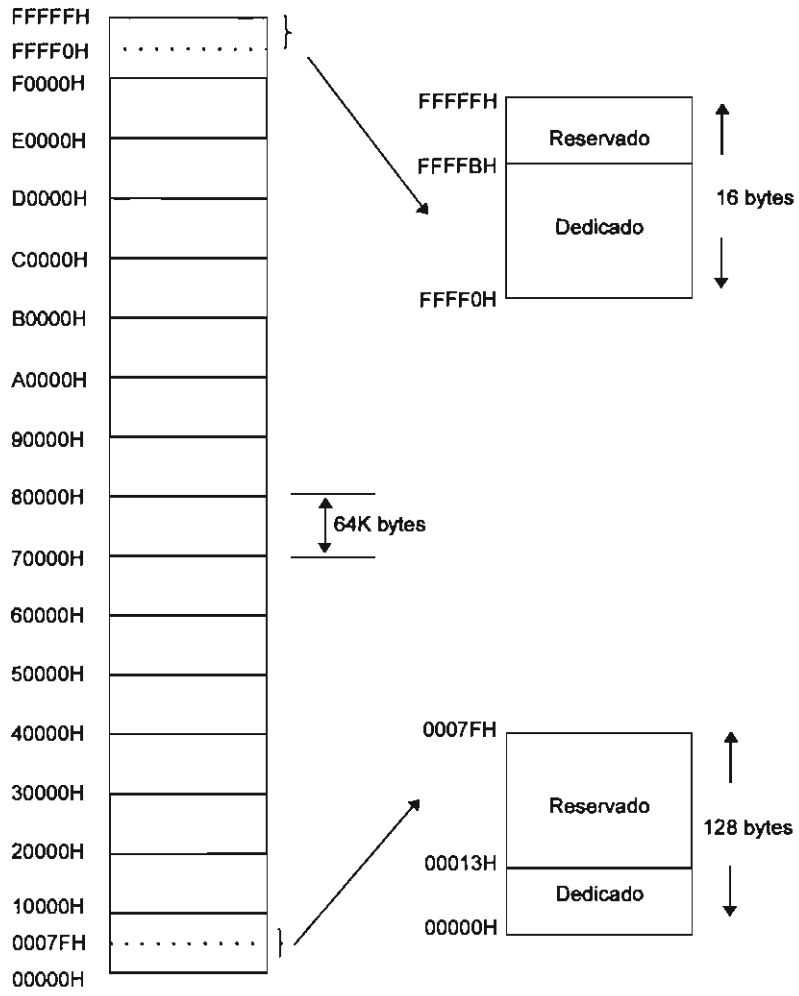


Figura I.6. Mapa de memoria del 8086/88.

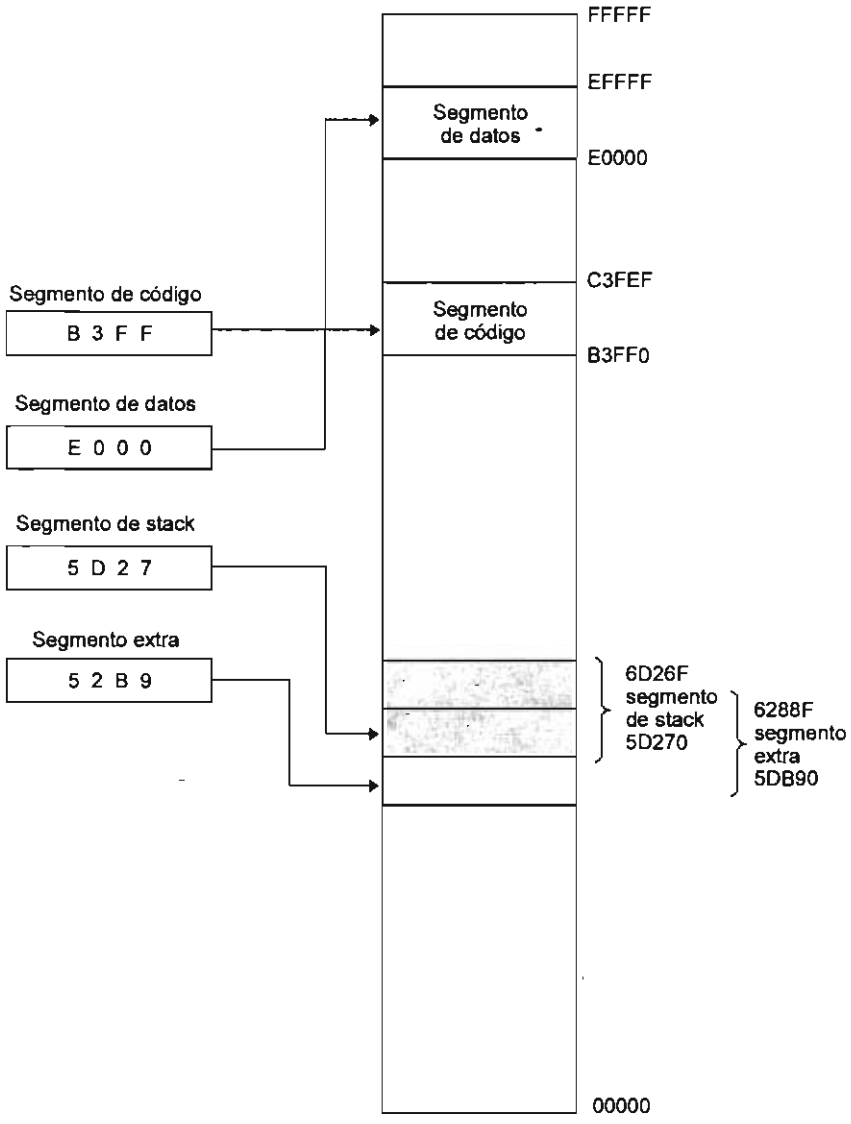


Figura I.7. Distribución de segmentos para una CPU 8086/88.

suponiendo que CS: B3FFH.

Concretando, la dirección física es de 20 bits y es la que entrega o saca la BIU; la dirección lógica es un offset a partir de la localidad 0000H en un segmento dado.

Ejemplos:

Calcule la dirección física que corresponde a la dirección lógica C580H en el segmento extra y repítala para la dirección lógica 3EA1H en el segmento de stack.

```
Extra Seg.          52B90H     (dirección base)
                  + C580H      (offset)
                    5F110H

Stack Seg.          5D270H     (dirección base)
                  + 3EA1H      (offset)
                    61111H
```

Cuando se traslapan dos segmentos, dos direcciones lógicas comparten la misma dirección física. Para especificar direcciones, generalmente se usa la siguiente convención:

Segmento: offset

o

Segmento: dirección lógica

```
ES: 5F110H = 52B90H: C580H
```

Ejemplo:

```
        La dirección 2D90H          La dirección lógica D470H
        en el segmento de stack:    en el segmento extra:

        Dir. física SS: 2D90H       ES: D470H

(dirección base)        5D270H          52B90H      (dirección base)
(offset)              + 2D90H         + D470H       (offset)
                        60000H?         60000H
```

Por otra parte, cada instrucción que hace referencia a memoria tiene un registro de segmento por "default", según el cuadro I.3, programado en la BIU.

Ejemplos:

1. Si el IP = 1000H & CS = B3FFH, la BIU formará la dirección física B3FF0H + 1000H = B4FF0H.

2. La instrucción MOV [BP], AL usará el segmento de stack, ya que cuando BP se utiliza como apuntador se usa el segmento de stack.

```
Supóngase       5D270H  (SS)
BP=2C30H      + 2C30H
                5FEA0H
```

CUADRO I.3. *Segmentos accesados por default*

Tipo de referencia a memoria	*Segmento default*	*Segmento alterno*	*Offset (dirección lógica)*
Fetch de una instrucción	CS	No hay	P
Operación con el stack	SS	No hay	SP
Datos generales	DS	CS, ES, SS	Dirección efectiva
Cadena fuente	DS	CS, ES, SS	SI
Cadena destino	ES	No hay	DI
BX como apuntador	DS	CS, ES, SS	Dirección efectiva
BP como apuntador	SS	CS, ES, DS	Dirección efectiva

En el cuadro I.3 podemos ver que BP también puede apuntar dentro de los segmentos de código, datos o el extra.

Las ventajas de la segmentación de memoria son que un programa puede trabajar con diferentes conjuntos de datos, que los programas que utilizan direcciones lógicas pueden cargarse y correr en cualquier parte de la memoria y que pueden trabajar en ambientes *multitasking*.

Multitasking es un ambiente en el que se realizan varios trabajos o tareas a la vez. Un programa inactivo se puede salvar a disco y su lugar lo toma un nuevo programa sin conocer direcciones físicas. A estos programas se les conoce como *relocalizables* (corren en cualquier área de memoria). Las referencias no se hacen a direcciones físicas y el diseñador no debe hacer cambios a los registros de segmento.

```
DATO:    DB    0H
SALTA:   JP 10 0H
```

Las direcciones de localidades de memoria dependen del contenido de los registros de segmento. En lenguaje ensamblador es conveniente asignar a las direcciones lógicas, etiquetas o nombres.

Ejemplo:

```
DATA     SEGMENT
TECLA    DB ?
DATA     ENDS
```

La secuencia anterior asigna la etiqueta TECLA al byte de la localidad 0000H del DATA SEGMENT.
Seudoinstrucciones:

- `DATA SEGMENT`: dice al ensamblador que almacene los siguientes códigos en el segmento DATA.
- "?": indica que el contenido del byte es indefinido.
- `DATA ENDS`: termina el segmento DATA.

Existen también las siguientes seudoinstrucciones:

- `PALABRA DW 1234H`: inicializa una palabra (2 bytes) de memoria y le asigna el nombre PALABRA.
- DD: palabra doble-32 bits.
- DQ: palabra cuádruple-8 bytes.

- `DT`: 10 bytes, usado con el NDP 8087.
- `100 DUP ?`: reserva 100 bytes consecutivos con elementos no inicializados.
- `100 DUP (0)`.
- `DB "mensaje_uno"`: reserva 11 bytes, insertando los valores ASCII de la cadena.

Conclusión: Estas seudoinstrucciones le permiten al programador definir localidades de memoria por nombre y no por su dirección específica.

Modos de direccionamiento

Instrucción = código de operación + 0, 1 o 2 operandos.

Identifica la operación a realizar, fuente y destino de los datos sobre los que se operará.

Los operandos pueden especificar: un registro de la CPU, una localidad de memoria, un puerto de entrada/salida.

Las diferentes formas en las cuales el microprocesador genera las direcciones de los operandos se llaman *modos de direccionamiento*. El 8086/88 tiene nueve modos de direccionamiento.

Modo de direccionamiento inmediato

Las instrucciones obtienen su dato como parte de la misma instrucción, y como caso especial el dato se almacena en el segmento de código (CS).

Ejemplo:
La instrucción

```
MOV AX, 1000H.
```

El dato se almacena en el CS, cargando 1000H en el registro AX.

Este modo no opera con registros de segmento, por lo que no se puede cargar un registro de segmento de manera directa.

Modo de direccionamiento de registro

Permite operar o mover datos entre registros de la CPU.

Ejemplos:

```
MOV DX, CX
INC BH
```

Este modo puede combinarse con el anterior para cargar un registro de segmento:

```
MOV AX, 8010H      ; Carga con 8010H.
MOV DS, AX         ; Copia AX en DS.
```

Por lo tanto DS = 80510H.

Modo de direccionamiento directo

Todos los modos siguientes realizan al menos un acceso a memoria para obtener su(s) operando(s).

En este modo la dirección de memoria se suministra directamente como parte de la instrucción.

Ejemplo:
La instrucción

```
MOV AH, TECLA o MOV AH, [TECLA]
```

almacena en AH el contenido de la localidad de memoria TECLA.

Modo de direccionamiento de acceso indirecto a memoria

El modo de direccionamiento directo es útil para accesos a memoria no frecuentes. En cambio cuando una localidad de memoria necesita ser escrita o leída varias veces en un programa, la búsqueda de la dirección lógica de 2 bytes se hace ineficiente. Para resolver este problema existen los modos de direccionamiento indirecto, que almacenan la dirección de memoria en un registro apuntador o en un índice (BX, BP, SI, DI) y le suma un desplazamiento en complemento a 2.

Combinaciones posibles

La dirección resultante de las combinaciones anteriores se llama dirección efectiva (EA).

```
LEA TECLA;
MOV AH, [TECLA + SI + 05H)
```

Veremos a continuación otra forma de escribir operandos en el modo de direccionamiento indirecto:

```
MOV AX, TABLA [SI]
```

donde TABLA es una etiqueta dada a una localidad de memoria (generalmente a la base). También puede escribirse

```
MOV AX, [TABLA + SI]
```

Los operadores BYTE PTR y WORD PTR le indican al ensamblador si se accesará un byte o una palabra de memoria. Por ejemplo, la instrucción `INC [BP]` no indica si se incrementará la palabra o el byte apuntado por BP. Si la instrucción utiliza dos operandos, el registro operando es el que dice de qué tipo será el acceso (word o byte).

Ejemplo:

```
MOV AX, [BP]  ; mueve al registro AX el contenido de la loc.;
                apuntada por BX.
```

Indica la palabra apuntada por BP, ya que AX es el destino de 16 bits.

CUADRO I.4. *Modos de direccionamiento que accesan memoria*

	Dirección efectiva		
Modo de direccionamiento	*Desplazamiento*	*Reg. base*	*Reg. índice*
Registro indirecto	No hay	BX o BP	No hay
	No hay	No hay	SI o DI
Indexado	+127 a –128	No hay	SI o DI
Basado *(based)*	+127 a –128	BX o BP	No hay
Basado e indexado	No hay	BX o BP	SI o DI
Basado e indexado con desplazamiento	+127 a –128	BX o BP	SI o DI

NOTA: Para todos los modos de direccionamiento indirecto, el segmento por default es el registro de segmento de stack cuando se utiliza el registro BP, y el registro de segmento de datos cuando se utilizan BX, SI o DI.

Modo de direccionamiento de cadena o *"string"*

Cadena *(string):* secuencia de bytes o words almacenados en memoria.

El 8086/88 tiene varias instrucciones para manejar caracteres. Usa DS: SI para apuntar a la cadena fuente y ES: DI para apuntar al destino de la cadena.

Ejemplo:

```
MOVSB  ; usa los segmentos de datos y extra
       y realiza la siguiente operación:
```

[ES: DI] ← [DS: SI]

Si DF = 0, entonces SI ← –SI + 1; DI ← DI + 1
Si DF = 1, entonces SI ← –SI – 1; DI ← DI – 1

SET DE INSTRUCCIONES DEL 8086/88

Instrucciones de transferencia de datos

Existen un poco más de 3 000 diferentes instrucciones para el 8086/88 si cada código de operación se combina con cada modo de direccionamiento.

La instrucción MOV

En el cuadro I.6 se muestran diferentes tipos de transferencia de datos en los que se usa MOV.

CUADRO I.5. *Modos de direccionamiento del microprocesador 8086/88*

Modo de direccionamiento	*Código objeto*	*Ejemplo de codificación*			
		Mnemónico	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
Inmediato	B8 00 10	MOV AX, 1000H	Código	AH ← 10H; AL 00	La fuente de datos está en la instrucción.
Registro	8B D1	MOV DX, CX	Con CPU	DX ← CX	Los registros de la CPU son fuente y destino de los datos.
Directo	8A 26 00 10	MOV AH, [MEMBDS][a]	Datos (DATA)	AH ← [1000H]	La dirección de memoria es suministrada en la instrucción.
Registro indirecto	8B 04 FF 25 FE 46 00 FF 0F	MOV AX, [SI] JMP [DI] INC BYTE PTR [BP][b] DEC WORD PTR [BX][b]	Datos (DATA) Datos (DATA) Pila Datos	AL ← [SI]; AH ← [SI + 1] IP ← [DI + 1: DI] [BP] ← [BP] + 1 [BX + 1: BX] ← [BX: +1] –1	La dirección de memoria es proporcionada en un registro índice o apuntador.
Indexado	8B 44 06 FF 65 06	MOV AX, [SI + 6][c] JMP [DI + 6][c]	Datos Datos	AL ← [SI + 6]; AH ← [SI + 7] IP ← [DI + 7: DI + 6]	La dirección de memoria es la suma del registro base BX o BP más el desplazamiento indicado en la instrucción.
Basado	8B 46 02 FF 67 02	MOV AX, [BP + 2][c] JMP [BX + 2][c]	Pila Datos	AL ← [BP + 2]; AH ← [BP + 3] IP ← [BX + 3: BX + 2]	La dirección de memoria es la suma de un registro índice y un registro base, BX o BP más el desplazamiento indicado en la instrucción.
Basado e indexado	8B 00 FF 21 FF 02 FF 0B	MOV AX, [BX + SI] JMP [BX + DI] INC BYTE PTR[BP + SI][b] DEC WORD PTR[BP + DI][b]	Datos Datos Pila Pila	AL ← [BX + SI] AH ← [BX + SI + 1] IP ← [BX + DI + 1: BX + DI] [BP + SI] ← [BP + SI] + 1 [BP + DI + 1: BP + DI] ← [BP + DI + 1: BP + DI] –1	La dirección de memoria es la suma de un registro índice y un registro base.
Basado e indexado con desplazamiento	8B 40 05 FF 61 05 FE 42 05 FF 4B 05	MOV AX, [BX + SI + 5][c] JMP [BX + DI + 5][c] INC BYTE PTR[BP + SI + 5][b, c] DEC WORD PTR [BP + DI + 5][b, c]	Datos Datos Pila Pila	AL ← [BX + SI + 5] AH ← [BX + SI + 6] IP ← [BX + DI + 6: BX + DI + 5] [BP + SI + 5] ← [BP + SI + 5] + 1 [BP + DI + 6: BP + DI + 5] ← [BP + DI + 6: BP + DI + 5] – 1	La dirección de memoria es la suma de un registro índice, un registro base y un desplazamiento indicado en la instrucción.

CUADRO I.5 *(concluye)*

Modo de direccionamiento	*Código objeto*	*Ejemplo de codificación*			
		Mnemónico	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
Cadenas	A4	MOVSB	Extra, datos	[ES: DI] ← [DS: SI] Si DF = 0 entonces SI ← SI + 1; DI ← DI + 1 Si DF = 1 entonces SI ← SI – 1; DI ← DI – 1	La dirección fuente de memoria es el registro SI en el segmento de datos, y la dirección destino de memoria es el registro DI en el segmento extra.

[a] MEMBDS se asume que apunta a la localidad 1000H en el segmento de datos. Los corchetes son opcionales.
[b] BYTE PTR y WORD PTR.
[c] El desplazamiento se sumado al apuntador o registro base como un número binario signado en complemento a 2.

MOV destino, fuente

Se pueden transferir bytes o palabras, y los registros apuntadores, índice y de segmento sólo se pueden accesar como words. El destino y la fuente no pueden ser localidades de memoria al mismo tiempo. Las banderas no son afectadas.

Ejemplo:

```
MOV CH, 0
MOV CL, CONTADOR
```

La localidad de memoria (byte) CONTADOR contiene el número de veces que debe realizarse una operación.

Instrucciones especiales de transferencia de datos

Hay otras instrucciones para la transferencia de datos que no usan el mnemónico MOV. La instrucción XCHG realiza transferencias del tipo registro-a-registro o registro-a-memoria, que requeriría tres instrucciones MOV. Por ejemplo, los 8 bits menos significativos del registro de banderas pueden almacenarse en o cargarse del registro AH (instrucciones LAHF & SAHF). Las instrucciones IN & OUT permiten al 8086/88 transferir datos con el exterior, pero un byte o word a leer o sacar debe pasar por el acumulador. AL debe ser la fuente o destino para operaciones de 8 bits.

La dirección del dispositivo de entrada/salida (puerto) se puede especificar de modo directo (la instrucción suministra la dirección pero es limitada de 0-255) o de modo indirecto (el registro DX especifica la dirección de 16 bits: 65 536 puertos).

Ejemplo:

Saque la palabra almacenada en BX por los puertos 3000H y 3001H.

```
MOV DX, 3000H    ;DX apunta a la dirección puerto.
MOV AX, BX       ;El dato debe estar en AX.
OUT DX, AX       ;Saca el dato.
```

El registro DX se cargó usando el modo de direccionamiento inmediato MOV DX, 3000H; pero si no conocemos la dirección de memoria, podemos usar la instrucción LEA *(load effective address)*. Por ejemplo, supongamos que a la localidad 3000H se le asignó la etiqueta DATO. La instrucción LEA BX, DATO carga el registro BX con la dirección efectiva de DATO. Así, el programador no necesariamente debe saber la dirección de la etiqueta, pues el ensamblador se encarga de ello.

Las instrucciones LDS *(load pointer using* DS) y LES *(load pointer using* ES) permiten cargar los registros de 16 bits DS y ES (registros de segmento) con el contenido de un operando double-word en memoria.

Ejemplo:

```
WORD  PTR 8
BYTE  PTR 16
DWORD PTR 32

LEA SI,PALABRA_DOBLE     (1)
LDS BX,DWORD PTR[SI]     (2)
```

⇑

El contenido de la doble palabra apuntada por SI.

CUADRO I.6. *Instrucción* MOV[a]

Modo de direccionamiento	*Código objeto*	*Ejemplo de codificación*			
		Mnemónico	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
MOV Destino, fuente	8B C3	MOV AX, BX	Con CPU	AX ← BX	Registro a registro
	8A E3	MOV AH, BL	Con CPU	AH ← BL	Registro a registro
	A1 00 10	MOV AX, MEMWDS[b]	Datos	AL ← 1000H AH ← 1001H	Memoria a registro
	A0 02 10	MOV AL, MEMBDS[c]	Datos	AL ← 1002H	Memoria a registro
	89 1E 00 10	MOV MEMWDS, BX[b]	Datos	[1000H] ← BL [1001H] ← BH	Registro a memoria
	88 1E 02 10	MOV MEMBDS, BL[c]	Datos	[1002] ← BL	Registro a memoria
	C7 06 00 10 34 12	MOV MEMWDS, 1234H[b]	Datos	[1000H] ← 34H [1001H] 12H	Dato inmediato a memoria
	C6 06 02 10 34	MOV MEMBDS, 34H[c]	Datos	[1002H ← 34H	Dato inmediato a memoria
	B0 10	MOV AL, 10H	Código	AL ← 10H	Dato inmediato a registro
	B8 00 10	MOV AX, 1000H	Código	AL ← 00H; AH ← 10H	Dato inmediato a registro
	8E D8	MOV DS, AX	Con CPU	DS ← AX	Registro general a registro de segmento
	8C C2	MOV DX, ES	Con CPU	DX ← ES	Registro de segmento a registro general
	8E 06 00 10	MOV ES, MEMWDS[b]	Datos	ES ← [1001H:1000H]	Memoria a registro de segmento
	8C 0E 00 10	MOV MEMWDS, CS[b]	Datos	[1001H:1000H] ← CS	Registro de segmento a memoria

[a] La localidad de memoria puede especificarse usando cualquiera de los modos de direccionamiento del cuadro II.2, donde también se muestra, para todos los ejemplos de acceso a memoria, el modo de direccionamiento directo.

[b] MEMWDS es asumido para apuntar a la palabra de inicialización en la localidad 1000H en el segmento de datos.

[c] MEMBDS es asumido para apuntar al byte en la localidad 1002H en el segmento de datos.

(1) Carga SI con la dirección de PALABRA_DOBLE (1000H).
(2) Carga BX y DS con el contenido de la palabra doble apuntada por SI.

```
BX = 8010H y DS = E000H
```

Ya que BX apunta por default al segmento de datos, la dirección física a la que apunta BX es:

```
E0000H + 8010H  =  E8010H.
```

Lo anterior también se puede hacer con la instrucción

```
LDS BX, PALABRA_DOBLE

LDS BX, [PALABRA_DOBLE]
```

Segment override

Cada instrucción que accesa a memoria tiene un registro de segmento que se usa para determinar la dirección efectiva del operando. Sin embargo, el segmento por *default* puede cambiarse usando el prefijo CS:, DS:, ES: o SS:. Una aplicación típica de este caso es permitir almacenar datos en el segmento de código.

Ejemplo:

```
CODE        SEGMENT
CONTADOR    DB 0FFH
MOV         AL, CS: CONTADOR
```

La variable CONTADOR se define dentro del segmento de código y la instrucción MOV cambia el segmento para accesar esta localidad de memoria. Algunos ensambladores tienen la capacidad de determinar el segmento de la variable accesada.
Otra forma de almacenar variables en el segmento de código es definir CS = DS.

Ejemplo:

```
MOV CX, CS              ; Copia CS en CX
MOV DS, CX              ; Copia CS en DS
MOV AL, CONTADOR        ; Carga AL con el contenido de CONTADOR
```

DS ← CS
No se necesita *segment override*

NOTA: Los programas que almacenan datos en el segmento de código utilizan un byte extra de código por cada acceso de datos, lo que implica que ocupan más memoria y son más lentos.

CUADRO I.7. *Instrucciones especiales de transferencia de datos*[a]

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
XCHG	Destino, fuente	93	XCHG AX, BX	Con CPU	AX ↔ BX	Intercambia el contenido de la palabra o byte del operando fuente con el operando destino; ninguna bandera es afectada.
		86 C7	XCHG AL, BH	Con CPU	AL ↔ BH	
		87 14	XCHG [SI], DX	Datos	[SI] ↔ DL; [SI + 1] ↔ DH	
LAHF		9F	LAHF	Con CPU	AH ← Banderas	Copia el byte de menor orden del registro de banderas en AH.
SAHF		9E	SAHF	Con CPU	Banderasl ← AH	Copia AH en el byte de menor orden del registro de banderas.
IN	Acumulador, puerto	EA 26	IN AL, 26H	[b]	AL ← puerto 26H	Introduce un byte o palabra desde un puerto directo de E/S 0-255. Introduce un byte o palabra desde un puerto indirecto de E/S 0-65535; la dirección del puerto está en DX; ninguna bandera es afectada.
		E5 26	IN AH, 26H	[b]	AL ← puerto 26H; AH ← puerto 27H	
		EC	IN AL, DX	[b]	AL ← puerto DX	
		ED	IN AX, DX	[b]	AL ← puerto DX; AH ← puerto DX + 1	
OUT	Puerto, acumulador	E6 26	OUT 26H, AL	[b]	Puerto 26H ← AL	Saca un byte o palabra a un puerto directo de E/S 0-255.
		E7 26	OUT 26H, AX	[b]	Puerto 26H ← AL; Puerto 27H ← AH	
		EE	OUT DX,AL	[b]	Puerto DX ← AL	Saca un byte o palabra a un puerto indirecto de E/S 0-65535; la dirección del puerto está en DX; ninguna bandera es afectada.
		EF	OUT DX, AX	[b]	Puerto DX ← AL; Puerto DX + 1 ← AH	
LEA	Destino, fuente[c]	8D 1E 00 10	LEA BX, MEMBDS[c]	Datos	BL ← 00; BH ← 10H	La dirección efectiva del operando fuente es transferida al operando destino; ninguna bandera es afectada.

LDS	Destino, fuente[d]	C5 1C	LDS BX, DWORD PTR [SI]	Datos	BL ← [SI]; BH ← [SI + 1]; DS ← [SI + 3: SI + 2]	Transfiere una variable apuntador de 32-bits desde el operando fuente en memoria al registro destino y registro DS o ES; ninguna bandera es afectada.
LES	Destino, fuente[d]	C4 1C	LES BX, DWORD PTR [SI]	Datos	BL ← [SI]; BH ← [SI + 1]; ES ← [SI + 3: SI + 2]	
XLAT		D7	XLAT	Datos	AL ← [BX + AL]	Reemplaza el byte en AL con uno de los 256 bytes de la tabla que empieza en [BX]; usa a AL como un offset dentro de esta tabla; ninguna bandera es afectada.

[a] Los operandos pueden ser accesados usando cualquiera de los modos de direccionamiento mostrados en el cuadro II.2. En esta tabla el modo de registro indirecto es usado solamente como un ejemplo.

[b] Las instrucciones IN y OUT no implican registros de segmento.

[c] Se asume que MEMBDS es un apuntador a un byte en la localidad 1000H del segmento de datos.

[d] El destino debe ser un registro de la CPU de 16 bits.

CUADRO I.8. *Prefijo de invalidación de segmento*

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
CS:		2E A1 00 10 2E 89 4E 00	MOV AX, CS: MEMWCS[a] MOV CS: [BP], CX	Código Código	AX ← CS: [1001H:1000H] CS: [BP] ← CX	El segmento de memoria por default para el operando fuente o destino es invalidado para ser ahora el segmento de código.
ES:		26 A1 00 10 26 89 4E 00	MOV AX, ES: MEMWES[a] MOV ES: [BP], CX	Extra Extra	AX ← ES: [1001H:1000H] ES: [BP] ← CX	El segmento de memoria por default para el operando fuente o destino es invalidado para ser ahora el segmento extra.
DS:		3E 89 4E 00	MOV DS: [BP], CX	Datos	DS: [BP] ← CX	El segmento de memoria por default para el operando fuente o destino es invalidado para ser ahora el segmento de datos.
SS:		36 A1 00 10 36 89 0F	MOV AX, SS: MEMWSS[a] MOV SS: [BP], CX	Pila Pila	AX ← SS: [1001H:1000H] SS: [BP] ← CX	El segmento de memoria por default para el operando fuente o destino es invalidado para ser ahora el segmento de pila.

[a]El ensamblador generará automáticamente la invalidación de segmento si estas palabras de memoria han sido previamente definidas en el segmento apropiado. Se asume que cada palabra comienza en la dirección 1000H en el segmento definido.

Instrucciones para el manejo de cadenas (strings)

Este tipo de instrucciones utilizan como fuente DS: SI y como destino el área de memoria apuntada por ES: DI. Se puede modificar el segmento de la dirección fuente, pero el destino debe ser el segmento extra.

Los apuntadores de offset (SI y DI) se incrementan o decrementan automáticamente (dependiendo de DF) en 1 (para bytes) o en 2 (para palabras).

Las instrucciones STOS *(store strings)* y LODS *(load strings)* transfieren un byte o una palabra del acumulador a la memoria o de la memoria al acumulador. La instrucción MOVS *(move string)* transfiere un byte o palabra de la fuente en memoria al destino en memoria. Las instrucciones SCAS *(scan string)* y CMPS *(compare string)* permiten comparar un byte o palabra con el acumulador (SCAS) o con un operando en memoria (CMPS). Después de la ejecución, las banderas reflejan el resultado.

Instrucciones de repetición de ciclos

Cuando la instrucción REP precede a las instrucciones STOS y MOVS, estas últimas se repetirán un número de veces igual al contenido del registro CX.

Ejemplo:

```
MOV AL, 20H          ; AL contiene el byte del dato.
LEA DI, BLOQUE       ; DI  ←  La dirección de BLOQUE.
MOV CX, 03E7H        ; Carga CX con 1000.
REP STOSB            ; Almacena AL en ES:DI
                     ; incrementando DI y repite
                     ; 1000 veces.
```

NOTA: suponiendo que DF = 0.

Llena un bloque de 1 000 bytes, localizando en el segmento extra, con 20H, y el bloque comienza en la dirección de BLOQUE.

Las instrucciones REPE/REPZ (repite si es igual o la bandera de cero es "1") y REPNE/REPNZ (repite si no es igual o la bandera de cero no es "1") se utilizan con las instrucciones SCAS y CMPS para verificar un cuadro de datos o probar si dos strings son iguales.

Instrucciones lógicas: NOT, AND, OR, XOR, TEST

El 8086/88 tiene funciones lógicas disponibles. que se realizan bit a bit entre los operandos fuente y destino.

Ejemplo:

```
MOV AL, 6DH          ; Carga AL con 6DH
MOV BH, 40H          ; Carga BH con 40H
AND AL, BH           ; AND AL con BH.

AND          0110   1101 (AL)
             0100   0000 (BH)
             ----------------
             0100   0000 = 40H (AL)
```

CUADRO I.9. *Instrucciones de cadena*

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
STOSB	Ninguno	AA	STOSB	Extra	ES: [DI] ← AL Si DF = 0, DI ← DI + 1 Si DF = 1, DI ← DI – 1	Transfiere un byte o palabra desde el registro AL o AX hacia el elemento de la cadena direccionado por DI en el segmento extra; si DF = 0, incrementa DI, en otro caso decrementa DI; las banderas no son afectadas.
STOSW	Ninguno	AB	STOSW	Extra	ES: [DI] ← AL ES: [DI + 1] ← AH Si DF = 0, DI ← DI + 2 Si DF = 1, DI ← DI – 2	
STOS	Destino[a]	AA	STOS MEMBES	Extra	Igual a STOSB si DI = MEMBES	
		AB	STOS MEMWES	Extra	Igual a STOSW si DI = MEMWES	
LODSB		AC	LODSB	Datos	AL ← DS: [SI] Si DF = 0, SI ← SI + 1 Si DF = 1, SI ← SI – 1	Transfiere un byte o palabra desde el elemento de la cadena direccionado por SI en el segmento de datos al registro AL o AX; si DF = 0, incrementa SI; las banderas no son afectadas.
LODSW		AD	LODSW	Datos	AL ← DS: [SI] AH ← DS: [SI + 1] Si DF = 0, SI ← SI + 2 Si DF = 1, SI ← SI – 2	
LODS	Fuente[a]	AC	LODS MEMBDS	Datos	Igual a LODSB si SI = MEMBDS	
		AD	LODS MEMWDS	Datos	Igual a LODSW si SI = MEMWDS	
MOVSB	Ninguno	A4	MOVSB	Datos, extra	ES: [DI] ← DS: [SI] Si DF = 0, DI ← DI + 1 SI ← SI + 1 Si DF = 1, DI ← DI – 1 SI ← SI – 1	Transfiere un byte o palabra desde el elemento de la cadena direccionado por SI en el segmento de datos hacia el elemento de la cadena direccionado por DI en el segmento extra; si DF = 0, incrementa SI y DI, en otro caso decrementa SI y DI; las banderas no son afectadas.
MOVSW	Ninguno	A5	MOVSW	Datos, extra	ES: [DI] ← DS: [SI] ES: [DI + 1] ← DS: [SI + 1] Si DF = 0, DI ← DI + 2 SI ← SI + 2 Si DF = 1, DI ← DI – 2 SI ← SI – 2	

MOVS	Destino, fuente[a]	A4	MOVS MEMBES, MEMBDS	Extra, datos	Igual a MOVSB si SI = MEMBDS y DI = MEMBES	
		A5	MOVS MEMWES, MEMWDS	Extra, datos	Igual a MOVSW si SI = MEMWDS y DI = MEMWES	
SCASB		AE	SCASB	Extra	AL – ES: [DI]; actualiza banderas Si DF = 0, DI ← DI + 1 Si DF = 0, DI ← DI – 1	Resta el byte o palabra del elemento de la cadena direccionado por DI en el segmento extra de AL o AX; si DF = 0, incrementa DI, en otro caso decrementa DI; las banderas son actualizadas para reflejar la relación del operando destino al operando fuente.
SCASW		AF	SCASW	Extra	AX – ES: [DI + 1: DI]; actualiza banderas Si DF = 0, DI ← DI + 2 Si DF = 1, DI ← DI – 2	
SCAS	Destino[a]	AE	SCAS MEMBES	Extra	Igual a SCASB si DI = MEMBES	
		AF	SCAS MEMWES	Extra	Igual a SCASW si DI = MEMWES	
CMPSB		A6	CMPSB	Extra, datos	DS: [SI] – ES: [SI]; actualiza banderas Si DF = 0, DI ← DI + 1 SI ← SI + 1 Si DF = 1, DI ← DI – 1 SI ← SI – 1	Resta un byte o palabra del elemento de la cadena destino direccionado por DI en el segmento extra del byte o palabra del elemento de la cadena fuente direccionado por SI en el segmento de datos; si DF = 0, incrementa DI y SI; en otro caso decrementa SI y DI; las banderas son actualizadas para reflejar la relación del operando destino al operando fuente
CMPSW		A7	CMPSW	Extra, datos	DS: [SI + 1: SI] – ES: [DI + 1: DI]; actualiza banderas Si DF = 0, DI ← DI + 2 SI ← SI + 2 Si DF = 1, DI ← DI – 2 SI ← SI – 2	
CMPS	Destino, fuente[a]	A6	CMPS MEMBES, MEMBDS	Extra, datos	Igual a CMPSB si SI =MEMBDS y DI = MEMBES	
		A7	SCMPS MEMWES, MEMWDS	Extra, datos	Igual a CMPSW si SI = MEMWDS y DI = MEMWES	

[a] El operando fuente o destino debe identificar al elemento de la cadena como un byte o palabra. Especificando el operando hace que la instrucción intente limpiarlo, sin embargo, el código objeto es el mismo al de la forma sin operando.

CUADRO I.10. *Prefijo* REP

Mnemónico general / *Código de operación*	*Código objeto*	*Ejemplo de codificación* — *Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
REP	F3 AA	REP STOSB	Extra	STOBS; CX ← CX – 1 Repite hasta CX = 0	La instrucción de cadena que sigue al prefijo REP es repetida hasta que CX es decrementado para llegar a 0
	F3 AB	REP STOSW	Extra	STOSW; CX ← CX – 1 Repite hasta CX = 0	
	F3 A4	REP MOVSB	Extra, datos	MOVSB; CX ← CX – 1 Repite hasta CX = 0	
	F3 A5	REP MOVSW	Extra, datos	MOVSW; CX ← CX – 1 Repite hasta CX = 0	
REPE/REPZ[a]	F3 AE	REPZ SCASB	Extra	SCASB; CX ← CX – 1 Repite si ZF = 1 y CX ≠ 0	Repite la operación de cadena si la búsqueda o comparación es igual (es decir, ZF = 1) y CX ≠ 0; decrementando CX no se afectan las banderas.
	F3 AF	REPZ SCASW	Extra	Como arriba excepto SCASW	
	F3 A6	REPZ CMPSB	Extra, datos	Como arriba excepto CMPSB	
	F3 A7	REPZ CMPSW	Extra, datos	Como arriba excepto CMPSW	
REPNE/REPNZ[a]	F2 AE	REPNE SCASB	Extra	SCASB; CX ← CX – 1 Repite si ZF = 0 y CX ≠ 0	Repite la operación de cadena si la búsqueda o comparación no es igual (es decir, ZF = 0 y CX ≠ 0: decrementando CX no se afectan las banderas
	F2 AF	REPNE SCASW	Extra	Como arriba excepto SCASW	
	F2 A6	REPNE CMPSB	Extra, datos	Como arriba excepto CMPSB	
	F2 A7	REPNE CMPSW	Extra, datos	Como arriba excepto CMPSW	

[a] Cualquiera de las dos formas del mnemónico puede ser usada.

Banderas:

CF = 0 Limpiada con la instrucción AND.
PF = 0 40H tiene un numero impar de 1's.
AF = X No definida.
ZF = 0 El resultado no es cero.
SF = 0 El bit 7 es cero.
OF = 0 Limpiada por la instrucción AND.

Entre las aplicaciones de la instrucción AND figura enmascarar (forzar a 0) algunos bits del operando fuente:

```
AND AL, BH;
JZ INICIO
```

Si BH = 40H, transfiere el control a la localidad INICIO si el bit 6 del registro AL es cero. Usando la instrucción AND se alterna el contenido del operando destino. Con la instrucción TEST se realiza la misma función y no se alteran los operandos destino.

Ejemplo:

```
TEST AL, 40H
JZ INICIO
```

Instrucciones de corrimiento y rotación

Pueden ser de 8 o de 16 bits y pueden realizarse sobre un registro de la CPU o una localidad de memoria.

Corrimiento o shift

Los bits que salen al final del registro se pierden.

Rotaciones

Los bits del operando no se pierden y se vuelven a cargar de manera circular. Hay corrimientos aritméticos (SAL y SAR) y lógicos (SHL y SHR). En los aritméticos el bit de signo (bit 7 o bit 15) no cambia con el corrimiento. Forma de uso:

```
SAL/SHL     destino, contador
SAR         destino, contador
SHR         destino, contador
RCL         destino, contador
```

Si deseamos rotar el contenido de un registro, por ejemplo DX, podemos cargar el registro CL con el número de veces que deberá realizarse la rotación, por ejemplo:

```
MOV CL, 5       LD A,1
RCL DX, CL      LD A,01H
```

CUADRO I.11. *Instrucciones lógicas*

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*[a]	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
NOT	Destino	F7 D3	NOT BX	Con CPU	BX ← $\overline{BX}$	Complementa todos los bits del byte o palabra del operando; no son afectadas las banderas.
		F6 14	NOT BYTE PTR [SI]	Datos	[SI] ← [$\overline{SI}$]	
AND	Destino, fuente	23 CA	AND CX, DX	Con CPU	CX ← CX · DX	Realiza una AND bit-a-bit entre los bytes o palabras de los operandos fuente y destino almacenando el resultado en el operando destino; AF es indefinida, todas las demás banderas se actualizan.[b]
		22 3C	AND BH, BYTE PTR [SI]	Datos	BH ← BH · [SI]	
		25 00 80	AND AX, 8000H	Código	AX ← AX · 8000H	
OR	Destino, fuente	0B CA	OR CX, DX	Con CPU	CX ← CX + DX	Realiza una OR bit-a-bit entre los bytes o palabras de los operandos fuente y destino almacenando el resultado en el operando destino; AF es indefinida, todas las demás banderas se actualizan.[b]
		0A 3C	OR BH, BYTE PTR [SI]	Datos	BH ← BH + [SI]	
		0D 00 80	OR AX, 8000H	Código	AX ← AX + 8000H	
XOR	Destino, fuente	33 CA	XOR CX, DX	Con CPU	CX ← CX ⊕ DX	Realiza una OR-exclusiva bit-a-bit entre los bytes o palabras de los operandos fuente y destino almacenando el resultado en el operando destino; AF es indefinida, todas las demás banderas se actualizan.[b]
		32 3C	XOR BH, BYTE PTR [SI]	Datos	BH ← BH ⊕ [SI]	
		35 00 80	XOR AX, 8000H	Código	AX ← AX ⊕ 8000H	
TEST	Destino, fuente	85 D1	TEST CX, DX	Con CPU	CX · DX; actualiza banderas	Realiza una AND bit-a-bit entre los bytes o palabras de los operandos fuente y destino; los operandos permanecen sin cambios; AF es indefinida, todas las demás banderas se actualizan.[b]
		84 3C	TEST BH, BYTE PTR [SI]	Datos	BH · [SI]; actualiza banderas	
		A9 00 80	TEST AX, 8000H	Código	AX · 8000H; actualiza banderas	

[a] Cualquiera de los modos de direccionamiento puede ser usado.
[b] CF y OF son puestas a cero.

Ejemplo:

```
MOV     CL, 3           ????  ????  ??03
MOV     AX, 7FH         007F  ????  ??03
MOV     BX, 0505H       007F  0505  ??03
ROL     AX, CL          03F8  0505  ??03
AND     AH, AL          01F8  0505  ??03
OR      BL, AL          0505  ??03

AX      0000 0011 0111 1111
AH      0000 0011
BH      0000 0101
  ------------------------
AH      0000 0001

BL      0000 0101
AL      1111 1000
  ------------------------
BL      1111 1101
```

```
Al finalizar:
  AX: 01F8H
  BX: 05FDH
  CX: ??03H
```

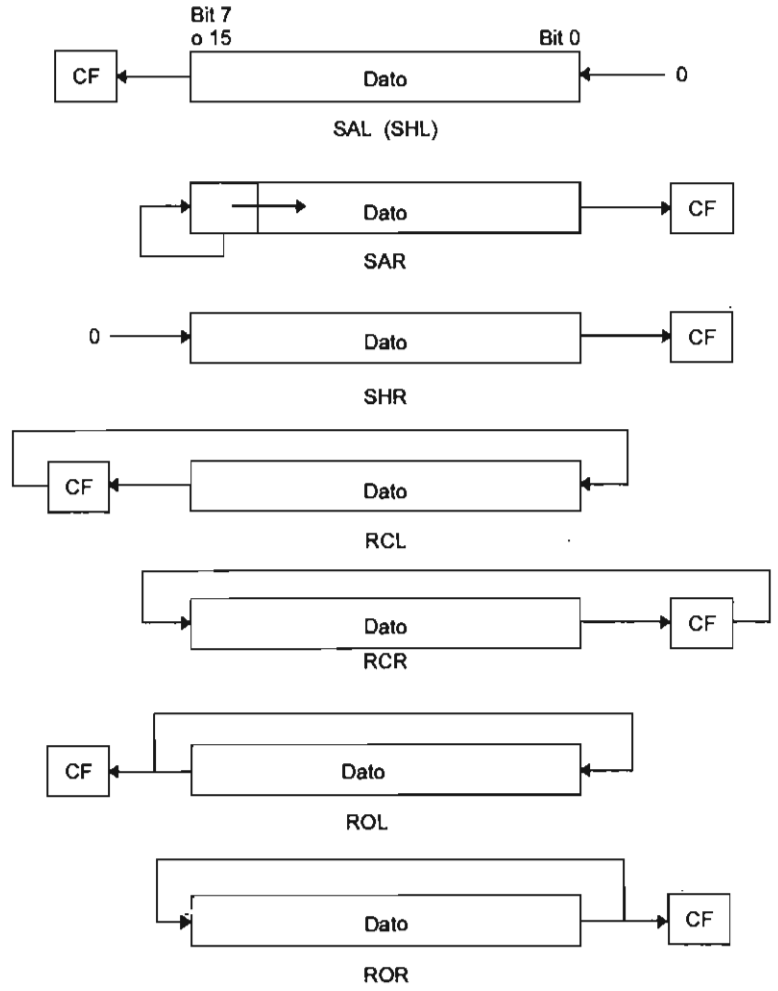


Figura I.8. Instrucciones de corrimiento y rotación.

CUADRO I.12. *Instrucciones de corrimiento y rotación*

Mnemónico general operando			*Ejemplo de codificación*[a]			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
SAL/SHL[b]	Destino, cuenta	D1 E0	SAL AX, 1	Con CPU	Véase fig. II.12	Hace un corrimiento al byte o palabra del operando, a la izquierda o derecha una o CL veces; AF es indefinida, todas las demás banderas son actualizadas; para corrimientos de un solo bit, OF es puesta si el signo del operando cambia.
		D3 E0	SAL AX, CL	Con CPU	Véase fig. II.12	
SAR	Destino, cuenta	D0 F8	SAR AL, 1	Con CPU	Véase fig. II.12	
SHR	Destino, cuenta	D2 F8	SAR AL, CL	Con CPU	Véase fig. II.12	
		D1 2C	SHR WORD PTR [SI], 1	Datos	Véase fig. II.12	
		D2 2C	SHR BYTE PTR [SI], CL	Datos	Véase fig. II.12	
RCL	Destino, cuenta	D1 D3	RCL BX, 1	Con CPU	Véase fig. II.12	Hace una rotación al byte o palabra del operando, a la izquierda o derecha una o CL veces; solamente CF y OF son afectadas; OF es puesta si el signo del operando cambia.
		D3 D3	RCL BX, CL	Con CPU	Véase fig. II.12	
RCR	Destino, cuenta	D0 DB	RCR BL, 1	Con CPU	Véase fig. II.12	
		D2 DB	RCR BL, CL	Con CPU	Véase fig. II.12	
ROL	Destino, cuenta	D1 04	ROL WORD PTR [SI], 1	Datos	Véase fig. II.12	
		D2 04	ROL BYTE PTR [SI], CL	Datos	Véase fig. II.12	
ROR	Destino, cuenta	D1 0E 00 10	ROR MEMWDS, 1[c]	Datos	Véase fig. II.12	
		D2 0E 04 10	ROR MEMBDS, CL[d]	Datos	Véase fig. II.12	

[a] Cualquiera de los modos de direccionamiento puede ser usado.
[b] Cualquiera de las dos formas del mnemónico puede ser usada.
[c] Se asume que MEMWDS apunta a la palabra que empieza en la localidad 1000H del segmento de datos.
[d] Se asume que MEMBDS apunta al byte en la localidad 1004H del segmento de datos.

Instrucciones aritméticas

En microprocesadores de 8 bits sólo había operaciones de 8 bits de suma y resta. El 8086/88 puede sumar y restar números de 8 o 16 bits de cualquier registro de propósito general de la CPU, y multiplicar y dividir números con o sin signo, usando ciertos registros dedicados.

Adición y sustracción

Los operandos destino y fuente pueden ser: registro y registro; registro y memoria; memoria y registro; dato inmediato y registro, o dato inmediato y memoria.

Se puede sumar y restar con *carry* o sin *carry*.

Ejemplo:

Sume dos números de 32 bits contenidos en los registros BX: AX y DX: CL almacenando el resultado en DX: CX.

```
  BX AX        ADD CX, AX  ;  CX ← CX  + AX
+ DX CX        ADC DX, BX  ;  DX ← DX + BX + CF
-------
  DX CX
```

Las instrucciones SUB (resta) y SBB (resta con *borrow*) trabajan de manera similar pero CF representa al *"borrow"*. Para sumar o restar uno de una variable o apuntador se pueden utilizar INC y DEC. La instrucción NEG (negación) obtiene el complemento a 2 del operando. Resta el operando del 0. (No hay que confundirla con el NOT.)

Ejemplo:

```
                MOV AL, 5
                NEG AL

        AL - 0000 0000 - 0000 0101  =  1111 1011  =  FBH  =  -5

CF = 1          ; la instrucción NEG pone a "1" CF, excepto cuando
                  el operando es 0.

PF = 0; AF = 1  ; hay un préstamo (borrow) del bit 6 al 4.

ZF = 0; SF = 1  ; OF = 0
```

La instrucción CMP (compara) sirve para determinar el tamaño relativo de dos operandos, seguida normalmente de una instrucción de salto condicional.

CUADRO I.13. Instrucciones de adición y sustracción

Mnemónico General			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
ADD	Destino, fuente	03 F2	ADD SI, DX	Con CPU	SI ← SI + DX	Reemplaza el byte o palabra destino con la suma de los operandos fuente y destino; actualiza todas las banderas.
		00 2F	ADD BYTE PTR [BX], CH	Datos	[BX] ← [BX] + CH	
		81 C7 00 80	ADD DI, 8000H	Con CPU	DI ← DI + 8000H	
		81 06 00 10	ADD MEMWDS	Datos	[1001H: 1000H] ←	
		00 80	8000H[a]		[1001H: 1000H] + 8000H	
ADC	Destino, fuente	13 F2	ADC SI, DX	Con CPU	SI ← SI + DX + CF	Reemplaza el byte o palabra destino con la suma de los operandos fuente y destino más el acarreo; actualiza banderas.
		10 2F	ADC BYTE PTR [BX], CH	Datos	[BX] ← [BX] + CH + CF	
		81 D7 00 80	ADC DI, 8000H	Con CPU	DI ← DI + 8000H + CF	
		81 16 00 10	ADC MEMWDS,	Datos	[1001H: 1000H] ←	
		00 80	8000H[a]		[1001H: 1000H] + 8000H + CF	
SUB	Destino, fuente	2B F2	SUB SI, DX	Con CPU	SI ← SI − DX	Reemplaza el byte o palabra destino con la diferencia entre el operando destino y el operando fuente; actualiza todas las banderas.
		28 2F	SUB BYTE PTR [BX], CH	Datos	[BX] ← [BX] − CH	
		81 EF 00 80	SUB DI, 8000H	Con CPU	DI ← DI − 8000H	
		81 2E 00 10	SUB MEMWDS,	Datos	[1001H: 1000H] ←	
		00 80	8000H[a]		[1001H: 1000H] − 8000H	
SBB	Destino, fuente	1B F2	SBB SI, DX	Con CPU	SI ← SI − DX − CF	Reemplaza el byte o palabra destino con la diferencia entre el operando destino y el operando fuente más el acarreo; actualiza todas las banderas.
		18 2F	SBB BYTE	Datos	[BX] ← [BX] − CH − CF	
		81 DF 00 80	PTR [BX], CH			
		81 1E 00 10	SBB DI, 8000H	Con CPU	DI ← DI − 8000H − CF	
		00 80	SBB MEMWDS,	Datos	[1001H: 1000H] ←	
			8000H[a]		[1001H: 1000H] − 8000H − CF	

INC	Destino[b]	FE C3	INC BL	Con CPU	BL ← BL + 1	Suma uno al byte o palabra del operando destino; almacena el resultado en el operando destino; todas las banderas, excepto CF, son actualizadas
		FF 05	INC WORD PTR [DI]	Datos	[DI + 1: DI] ← [DI + 1: DI] + 1	
		FE 06 04 10	INC MEMBDS[c]	Datos	[1004H ← [1004H] + 1	
DEC	Destino[b]	FE CB	DEC BL	Con CPU	BL ← BL – 1	Resta uno del byte o palabra del operando destino, almacenando el resultado en el operando destino; todas las banderas, excepto CF, son actualizadas
		FF 0D	DEC WORD PTR [DI]	Datos	[DI + 1: DI] ← [DI + 1: DI] –1	
		FE 0E 04 10	DEC MEMBDS[c]	Datos	[1004H ← [1004H] – 1	
NEG	Destino[b]	F6 DB	NEG BL	Con CPU	BL ← BL –1	Realiza el complemento a 2 del byte o palabra del operando destino; actualiza todas las banderas (CF = 1 excepto cuando el operando es cero).
			NEG WORD PTR [DI]	Datos	[DI + 1: DI] ← 0 – [DI + 1: DI]	
			NEG MEMBDS[c]	Datos	[1004H] ← 0 – [1004H]	
CMP	Destino, fuente	3A C4	CMP AL, AH	Con CPU	AL – AH; actualiza banderas	Resta el byte o palabra del operando fuente del operando destino; los operandos permanecen sin cambio; las banderas son actualizadas para reflejar la relación del operando destino al operando fuente.
		39 0D	CMP [DI], CX	Datos	[DI + 1: DI] – CX; actualiza banderas	
		81 3E 00 10 00 80	CMP MEMWDS, 8000H[a]	Datos	[1001H: 1000H] – 8000H; actualiza banderas	
		81 FF 00 80	CMP DI, 8000H	Con CPU	DI – 8000H; actualiza banderas	

[a] Se asume que MEMWDS apunta a la palabra que empieza en la localidad 1000H en el segmento de datos.
[b] Operandos inmediatos no son permitidos.
[c] Se asume que MEMBDS apunta al byte en la localidad 1004H en el segmento de datos.

Multiplicación y división

En la multiplicación, el operando fuente puede ser una localidad de memoria o un registro de la CPU, pero el operando destino debe ser el registro AX (y el DX para resultados de 32 bits):

MUL fuente

Ejemplo:
Escriba un programa para leer por los puertos A0H y B0H 2 números de 8 bits y saque su producto de 16 bits por el puerto 7080H:

```
IN AL, 0A0H          ;Lee el primer número en AL.
MOV BL, AL           ;Sálvalo en BL.
IN AL, 0B0H          ;Lee el segundo número.
MUL BL               ;El producto en AX.
MOV DX, 7080H        ;La dirección del producto en DX.
OUT DX, AX           ;Saca el producto.
```

La división puede realizarse con una palabra almacenada en AX o con una doble palabra en DX:AX. El divisor puede ser un registro de la CPU o un byte o palabra almacenado en memoria. El residuo queda en AH o en DX.

Ejemplo:
Escriba un programa que divida una palabra, leída por el puerto 8000H, entre 500D. ¿Qué pasará si el dato leído por el puerto = 56 723?

```
MOV DX, 8000H          ; DX tiene la dirección del puerto
MOV BX, 01F4H          ; BX el divisor (500D)
IN AX, DX              ;
MOV DX, 0000H          ; Limpia DX
DIV BX                 ; Calcula el cociente en AX y el
                       ; residuo en DX.
```

Resultado:

AX = Entero (56 723/500) = 113 = 71H
DX = MOD (56 723/500) = 223 = 00DFH

NOTA: cuando el divisor es muy pequeño, de forma tal que el cociente no se puede almacenar en el registro del resultado, ocurrirá una interrupción de división por cero *(divide by zero software interrupt)* y el control del programa se transferirá automáticamente a las localidades 00000-00003H.

Ejemplo:
Si se divide 65 536 entre 2, entonces el resultado no cabe en AL.

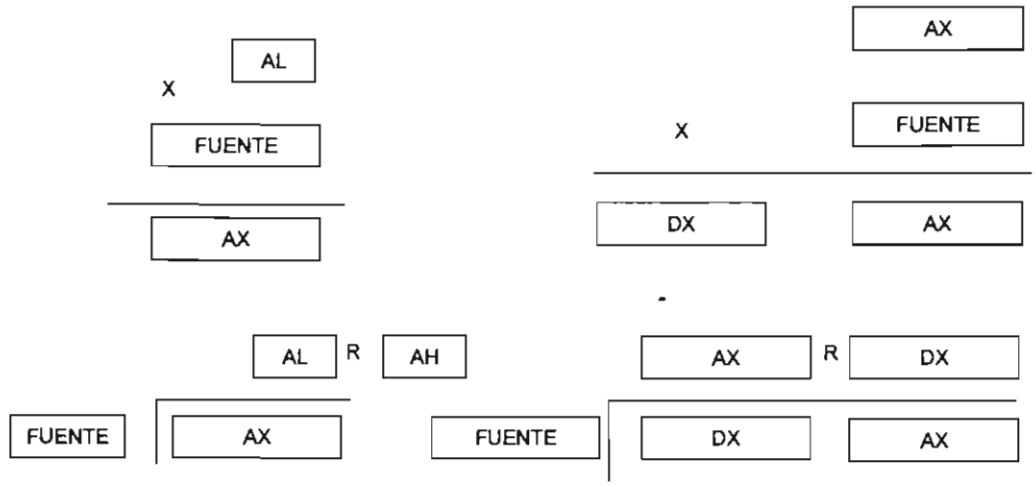


Figura I.9. Instrucciones de multiplicación y división.

Instrucciones de transferencia de control

Instrucciones de saltos incondicionales

Ejemplo:
Pertenecen a un tipo de saltos llamados *near jumps*.

```
ADDR  HEX CODES   LABELS    OP-CODES  OPERANDS     COMMENTS
0000  B3 04                 MOV       BL, 04H   ;  Cargar el divisor en BL
0002  E5 06       REPEAT:   IN        AX, 06H   ;  Obtiene el dato
0004  F6 F3                 DIV       BL        ;  Divide por cuatro
0006  E7 9A                 OUT       9AH, AX   ;  Salida del resultado
0008  E9  F7                JMP       REPEAT    ;  Repite el ciclo
000B  FF
```

El salto usado se llama *salto relativo* (el control se transfiere a una nueva dirección relativa al valor de IP).

```
  000BH
+ FFF7H
 1|0002H
```

FFF7H = –9 en complemento a 2.

A este tipo de salto también se le llama de forma directa. Cuando la dirección del salto está dentro de +127 y –128 se ahorra 1 byte de código y se llama *short jump*. El programa anterior también se puede escribir como:

```
ADDR  HEX CODES   LABELS    OP-CODES    OPERANDS     COMMENTS
0000  B3 04                 MOV         BL, 04H   ;  Cargar el divisor en BL
0002  E5 06       REPEAT:   IN          AX, 06H   ;  Obtiene el dato
0004  F6 F3                 DIV         BL        ;  Divide por cuatro
0006  E7 9A                 OUT         9AH, AX   ;  Salida del resultado
0008  E9 F8                 JMP short   REPEAT    ;  Repite el ciclo
000A
```

CUADRO I.14. *Instrucciones de multiplicación y división*

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*[a]	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
MUL	Fuente	F6 E3	MUL BL	Con CPU	AX ← AL * BL	Multiplicación sin signo del byte o palabra del operando fuente y el acumulador; la palabra resultado es almacenada en AX y la palabra doble resultado es almacenada en DX: AX (véase fig.II2.14); si el resultado no puede ser almacenado en un solo byte (para multiplicación de bytes) o una sola palabra (para multiplicación de palabras) CF y OF son puestas, o borradas en otro caso; todas las demás banderas son indefinidas.
		F7 E1	MUL CX	Con CPU	DX: AX ← AX * CX	
		F6 27	MUL BYTE PTR [BX]	Datos	AX ← AL * [BX]	
		F7 26 00 10	MUL MEMWDS[b]	Datos	DX: AX ← AX * [1001H: 1000H]	
IMUL	Fuente	F6 EB	IMUL BL	Con CPU	AX ← AL * BL (con signo)	Igual a MUL excepto que son permitidos números con signo; el operando fuente está limitado de –128 a +127 para multiplicaciones de byte y –32768 a +32767 para multiplicaciones de palabras; CF y OF son puestas a uno si el resultado no puede ser representado en el registro de menor orden del resultado; y son borradas en otro caso, si el signo es extendido al registro de mayor orden; las otras banderas no son
		F7 E9	IMUL CX	Con CPU	DX: AX ← AX * CX (con signo)	
		F6 2F	IMUL BYTE PTR [BX]	Datos	AX ← AL * [BX] (con signo)	
		F7 2E 00 10	IMUL MEMWDS[b]	Datos	DX: AX ← AX * [1001H: 1000H] (con signo)	

DIV	Fuente	F6 F3	DIV BL	Con CPU	AX ← AL / BL	División sin signo del acumulador (para divisores de byte) o el acumulador y DX (para divisores de palabra); para divisores de byte el resultado es regresado en AL con el residuo en AH; para divisores de palabra el resultado es regresado en AX con el residuo en DX (véase fig. II.14); si el cociente excede la capacidad del registro destino (AL o AX) es generada una interrupción de tipo 0; todas las banderas son indefinidas.
		F7 F1	DIV CX	Con CPU	DX: AX ← AX / CX	
		F6 37	DIV BYTE PTR [BX]	Datos	AX ← AL / [BX]	
		F7 36 00 10	DIV MEMWDS[b]	Datos	DX: AX ← AX / [1001H: 1000H]	
IDIV	Fuente	F6 FB	IDIV BL	Con CPU	AX ← AL / BL (con signo)	Igual a DIV excepto que son permitidos números con signo; el operando fuente está limitado de –128 a +127 para divisiones de byte y –32768 a 32767 para divisiones de palabra.
		F7 F9	IDIV CX	Con CPU	DX: AX ← AX / CX (con signo)	
		F6 3F	IDIV BYTE PTR [BX]	Datos	AX ← AL / [BX] (con signo)	
		F7 3E 0C 10	IDIV MEMWDS[b]	Datos	DX: AX ← AX / [1001H: 1000H] (con signo)	

[a] No son permitidos operandos inmediatos (excepto con el 80186).
[b] Se asume que MEMWDS apunta a una palabra que empieza en la localidad 1000H en el segmento de datos.

Aquí el desplazamiento del IP sólo se especifica con 1 byte. La palabra SHORT es opcional, ya que si el salto es de +127 y –128 (máximo), el ensamblador genera automáticamente un short jump.

También existe un tipo de salto indirecto, en el cual el salto no es relativo y se puede especificar cualquier dirección dentro del segmento de código; pertenece asimismo a los near jumps.

Ejemplo:
En el programa anterior podemos reemplazar la instrucción JMP SHORT REPEAT por las instrucciones:

```
MOV BX, 0002H     ; BX apunta a REPEAT.
JMP BX            ; salta a la dirección. 0002H.
```

Resumiendo, la ventaja de usar saltos relativos es que el programa resultante es relocalizable en cualquier parte del segmento de código, esto es, las direcciones de saltos no deben cambiarse si el origen del programa se modifica. La ventaja de los saltos indirectos es que pueden saltar a cualquier dirección, pero utilizan una instrucción extra para indicarla.

Finalmente, el control se puede transferir a una dirección localizable fuera o en otro segmento de código con lo que se llama *far jump*. Este tipo de salto no es relativo y puede ser directo (el operador requiere ensamblador FAR PTR, el cual indica que la etiqueta a la que se realizará el salto está en un segmento de código *nuevo*) o indirecto (se debe especificar una palabra doble, *double word*, para los valores nuevos de los registros CS & IP).

Instrucciones de saltos condicionales

Realizan un short jump basado en las condiciones de las banderas. El cuadro I.15 muestra las banderas que se pueden probar para realizar un salto de este tipo.

Ejemplo:
El siguiente programa:

```
MOV       BL, 47H
IN        AL, 36H
CMP       AL, BL
JE        IGUAL
JA        MAYOR
JMP       MENOR
```

lee el dato por el puerto 36H y lo compara con 47H. Si ambos operandos son iguales, el control se transfiere a la dirección IGUAL. Si el byte de entrada es 74H, el control se transfiere a MAYOR, y si no se alcanza ninguna de estas condiciones, el control se pasa a MENOR.

NOTA: las localidades IGUAL, MAYOR y MENOR deben encontrarse dentro de –128 y +127 bytes con respecto al salto.

La instrucción JCXZ es un salto condicional que no prueba banderas sino que prueba el registro CX y transfiere el control si CX = 0. Véase el cuadro I.15 (mnemónicos de instrucciones de salto).

CUADRO I.15. *Instrucciones de salto*

Mnemónicogeneral			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
JMP	Etiqueta cercana	E9_ _[a]	JMP MEMN[b]	Código	IP ← MEMN	Transfiere el control (cercano) dentro del segmento; el modo de direccionamiento puede ser directo, memoria indirecta o registro indirecto.
		FF 26 00 10	JMP [MEMWDS][c]	Dato	IP ← [MEMWDS+: MEMWDS]	
		FF 27	JMP [BX]	Dato	IP ← [BX + 1: BX]	
		FF E0	JMP AX	Dentro de la CPU	IP ← AX	
JMP SHORT	Etiqueta	EB_[d]	JMP SHORT MEMS[e]	Código	IP ← MEMS	Transfiere el control (corto); no hay formas indirectas en esta instrucción.
JMP	Etiqueta lejana	EA 03 00	JMP FAR PTR	Código	IP ← 0003H;	Transferencia de control lejana, es decir fuera del segmento.
		D3 9E	EMF[f]		CS ← 9ED3H	
		FF 2E 05 10	JMP [MEMWWDS][g]	Dato	IP ← [1006H: 1005H]; IP ← [1008H: 1007H]	
		FF 2F	JMP DWORD PTR [BX]	Dato	IP ← [BX + 1: BX]; IP ← [BX + 3: BX + 2]	
JCOND	Etiqueta lejana	73_[d]	JNC MEMS	Código	Si CF = 0, entonces IP ← MEMS	Transfiere el control (corto) si la condición es verdadera. Todos los saltos condicionados requieren saltos cortos.
JCXZ	Etiqueta corta	E3_[d]	JCXZ MEMS	Código	Si CX = 0, entonces IP ← MEMS	Transfiere el control (corto) si CX = 0

[a] Estos dos bytes son el complemento a dos del offset entre la localidad de memoria cercana MEMN y la instrucción inmediata siguiente a la instrucción JMP. Por ejemplo, si el salto es de 12 localidades adelante, el offset es 00 0C. Similarmente, 12 localidades atrás requiere un offset de FFF4. El offset es limitado a 32767 localidades adelante (7FFFH) y –32768 localidades atrás (8000H).

[b] MEMN es asumido para apuntar a una localidad de memoria cercana (por ejemplo dentro del segmento)

[c] MEMWDS es asumido para apuntar a una palabra que empieza en la localidad 1000H en el segmento de datos. Los corchetes son opcionales porque MEMWDS define una palabra, no una localización para salto.

[d] Este byte es el complemento a dos del offset entre la localidad de memoria corta MEMS y la instrucción inmediata siguiente a la instrucción de salto corto. El offset es limitado a 127 localidades hacia adelante (7FH) y 128 hacia atrás (80H).

[e] MEMS es asumido para apuntar a una localidad de memoria corta (por ejemplo +127 o –128 localidades desde la dirección de la instrucción inmediatamente siguiente a la instrucción de salto corto). El operador SHORT es opcional si MEMS ya es conocido por el ensamblador.

[f] MEMF es asumido para apuntar a la localidad de memoria lejana 9ED3H: 0003H.

[g] MEMWWDS es asumido para apuntar a una doble palabra que inicia en la localidad 1005H en el segmento de datos. Los corchetes son opcionales porque MEMWWDS define una doble palabra, no una localidad para salto.

[h] En el cuadro II.15 aparece la lista de condiciones que pueden ser probadas.

Instrucciones de lazos (loops)

Este tipo de instrucciones combina instrucciones para el decremento de contadores y de transferencia de control en una misma instrucción.

Ejemplo:

Haga un programa para sacar por el puerto de salida A0H un cuadro de 256 bytes que comienza en la localidad TABLA.

```
        LEA   SI, TABLA   ; Carga SI con la dirección base de TABLA
        MOV   CX, 0100H   ; CX con el número de bytes a sacar.
OTRO:   LODS              ; Carga un byte de la tabla en AL e
                          ; incrementa SI.
        OUT   0A0H        ; Saca el dato.
        LOOP  OTRO        ; Repite hasta que CX = 0.
```

NOTA: Se asume que la bandera DF = 0.

CUADRO I.16. *Instrucciones de saltos condicionales*

Mnemónico	*Condición*
	Operaciones con signo
JG/JNLE	Mayor/no menor ni igual ((SF ⊕ OF) + ZF) = 0
JGE/JNL	Mayor o igual/ni menor (SF ⊕ OF) = 0
JL/JNGE	Menor/no mayor ni igual (SF ⊕ OF) = 1
JLE/JNG	Menor o igual/no mayor ((SF ⊕ OF) + ZF) = 1
JO	Sobreflujo (OF = 1)
JS	Signo (SF = 1)
JNO	No sobreflujo (OF = 0)
JNS	No signo (SF = 0)
	Operaciones sin signo
JA/JNBE	Arriba/no abajo ni igual (CF ⊕ ZF) = 0
JAE/JNB	Arriba o igual/no abajo (CF = 0)
JB/JNAE	Abajo/no arriba ni igual (CF = 1)
JBE/JNA	Abajo o igual/no arriba (CF ⊕ ZF) = 1
	Otros
JC	Carry (CF = 1)
JE/JZ	Igual/cero (ZF = 1)
JP/JPE	Paridad/paridad par (PF = 1)
JNC	No carry (CF = 0)
JNE/JNZ	No igual/no cero (ZF = 0)
JNP/JPO	No paridad/paridad impar (PF = 0)

CUADRO I.17. *Instrucciones de lazos*

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
LOOP	Etiqueta corta	E2_[a]	LOOP MEMS[b]	Código	CX ← 1 Si CX ≠ 0, entonces IP ← MEMS	Decrementa CX y transfiere control a una dirección corta si CX ≠ 0.
LOOPE[c] LOOPZ	Etiqueta corta	E1_a	LOOPZ MEMS[b]	Código	CX ← 1 Si (CX ≠ 0) · (ZF = 1), entonces IP ← MEMS	Decrementa CX y transfiere control a una dirección corta si CX ≠ 0 AND la ultima instrucción que afecta el resultado de las banderas en cero (ZF = 1).
LOOPNE[c] LOOPNZ	Etiqueta corta	E0_[a]	LOOPNZ MEMS[b]	Código	CX ← 1 Si (CX ≠ 0) · (ZF = 0) entonces IP ← MEMS	Decrementa CX y transfiere control a una dirección corta si CX ≠ 0 AND la ultima instrucción que afecta el resultado de las banderas en un resultado no cero (ZF = 0).

[a] Este byte es el complemento a 2 del offset entre la localización de memoria corta MEMS y la instrucción inmediata seguida de la instrucción LOOP. El offset es limitado a 127 ocalidades adelante (7FH) y 128 localidades atrás (80H).

[b] MEMS es asumida para apuntar a una localidad de memoria corta (por ejemplo: +127 o -128 localidades desde la dirección de la instrucción inmediatamente siguiente a la nstrucción LOOP).

[c] La forma del mnemónico tampoco puede ser usada.

La instrucción LODSB *(load string byte)* es el equivalente de:

```
MOV AL, [SI]
INC SI
```

y la instrucción LOOP OTRO equivale a:

```
DEC CX
JNZ OTRO
```

Las instrucciones LOOPE o LOOPZ *(loop if equal or zero)* y LOOPNE *(loop if not equal or not zero)* prueban CX y ZF, LOOPE continúa el loop o lo repite si CX es diferente de 0 y ZF = 1. Estas instrucciones sirven para comparar dos cadenas o *strings* o ver si un registro de la CPU es igual a un byte o una palabra de un cuadro, es decir, ver si un valor o elemento se encuentra en el cuadro.

NOTA: el loop se realizará 65 536 veces si CX = 0 inicialmente.

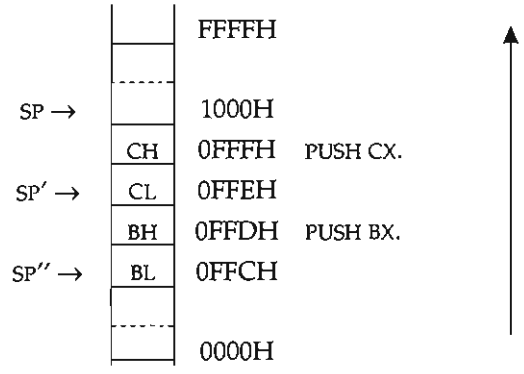
Instrucciones push y pop

Realmente no son instrucciones de transferencia de control. Usan el área de *stack*, que es un segmento de 64 Kbytes determinado por el registro SS, y normalmente apuntan dos registros a esta área: SP y BP.

El stack es un área de memoria del tipo *last-in, first-out* (LIFO) en la que el SP siempre apunta al ultimo contenido almacenado en el stack; los registros deben sacarse del stack en orden inverso al que fueron almacenados o introducidos. Antes de usar el stack, debe ser inicializado a una localidad alta de memoria para permitir que crezca.

Ejemplo:

```
MOV SP, 1000H
```



Instrucciones call y return

Permiten usar un grupo de instrucciones como una subrutina o procedimiento.

CUADRO I.18. *Instrucciones* PUSH *y* POP

Mnemónico general			*Ejemplo de codificación*			
	Modo de direccionamiento	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
PUSH	Fuente[a]	51	PUSH CX	Stack	SP ← SP–2; [SP + 1] ← CH; [SP] ← CL	Decrementa SP en 2 y transfiere la palabra desde el operador fuente a la parte más alta del stack, ahora apuntado por SP.
		1E	PUSH DS	Stack	SP ← SP–2; [SP + 1: SP] ← DS	
		FF 75 02	PUSH[DI + 2]	Stack, dato	SP ← SP–2; [SP + 1] ← [DI + 3]; [SP] ← [DI + 2]	
POP	Destino[a]	59	POP CX	Stack	CL ← [SP], CH ← [SP + 1]; SP ← SP + 2	Transfiere la palabra desde la parte mas alta del stack apuntado por SP al operando destino; incrementa SP en 2.
		1F	POP DS	Stack	DS ← [SP + 1: SP]; SP ← SP + 2	
		8F 45 02	POP[DI + 2]	Datos, stack	[DI + 3] ← [SP + 1]; [DI + 2 ← [SP]; SP ← SP + 2	
PUSHF	Ninguno	9C	PUSHF	Stack	SP ← SP–2; [SP + 1: SP] ← banderas	Coloca la palabra de bandera de 16-bits en el stack.
POPF	Ninguno	9D	POPF	Stack	Banderas ← [SP + 1: SP]; SP ← SP + 2	Retira de la parte alta del stack la palabra de bandera de 16-bits.

[a] Los operandos inmediatos no son permitidos (excepto dentro del 80186).

Ejemplo:
Retardo de aproximadamente 20 ms con un reloj de 5 MHz.

```
RETARDO     PROC NEAR          ; Define un procedimiento near.
            PUSH AX            ; Salva AX en el stack.
            MOV AX, 0000       ; Inicializa contador a 6, 36 ciclos.
REPITE      DEC AX             ; Decrementa el contador.
            JNZ REPITE         ; Hazlo 6, 36 veces.
            POP AX             ; Restaura AX.
            RET                ; Regresa al programa principal.
RETARDO     ENDP
```

La primera línea le indica al ensamblador que el siguiente grupo de instrucciones es un *near procedure* (subrutina dentro del mismo segmento) y le da el nombre de RETARDO. El llamado al procedimiento puede ser como sigue:

```
Programa principal
        .
        .
        .
    CALL RETARDO
(Continúa el programa principal)
        .
        .
        .
```

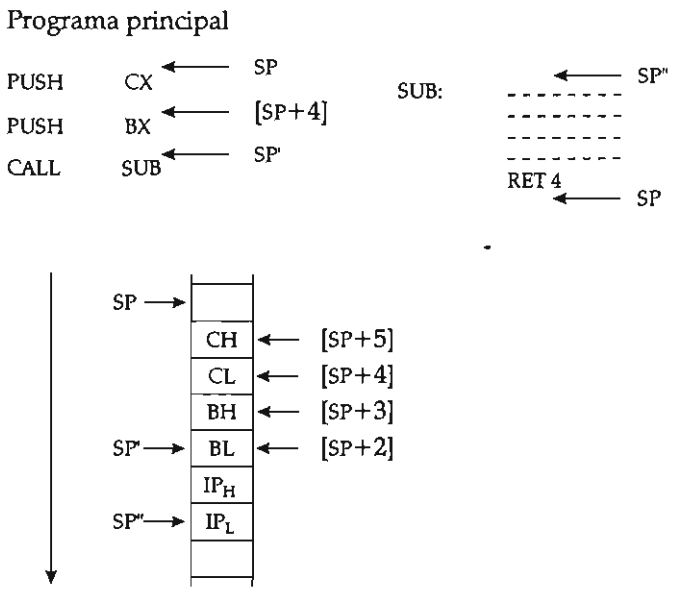
Con la instrucción CALL, el valor del IP se almacena en el tope del stack. También es posible utilizar un CALL del tipo *far* (*far* CALL) cuando el procedimiento se encuentra en un segmento de código diferente. El *far* CALL y el *near* CALL difieren en que en el primero el valor del CS y del IP son salvados en el tope del stack. Así entonces, también debe usar un *far* RET cuando se utilice un *far procedure*. El ensamblador generará automáticamente la instrucción RET propia si se ha definido el procedure utilizando el atributo NEAR o FAR.

Las instrucciones *near* CALL pueden ser directas o indirectas: Las *directas* son relativas y permiten que las subrutinas se encuentren dentro de +32767 a –32768 bytes con respecto al IP; no hay formas cortas (short). Las *indirectas* especifican la dirección absoluta de la subrutina en una localidad de memoria o en un registro de la CPU.

La forma directa de un CALL requiere el operador FAR PTR para decirle al ensamblador que la subrutina se encuentra en otro segmento. La forma indirecta requiere un *double word* para especificar el valor nuevo del IP y el CS. En realidad, en una instrucción CALL el operador *far* o *near* le dice al ensamblador de qué tipo será la instrucción RET, es decir, si debe sacar una o dos palabras del tope del stack para obtener la dirección de regreso.

Las instrucciones RET también pueden especificar un valor opcional *pop value*, el cual indica en cuánto debe incrementarse el valor del SP después de haber sacado del stack el valor del IP (dirección de regreso al programa principal). Es decir, sirve para desechar datos que ya no se utilizarán.

Ejemplo:



La instrucción RET4 causa que el valor del *stack pointer* pase de SP' a SP. El dato del stack puede ser accesado en la subrutina usando al registro BP como apuntador.

Interrupciones por software (software interrupts)

Una interrupción es una solicitud hacia el procesador para suspender el programa actual y transferir el control a un nuevo programa llamado rutina de servicio a interrupción *(Interrupt Service Routine-*ISR). La activación ("1" lógico) de las líneas INTR o NMI genera una solicitud de interrupción en el 8086/88.

Una interrupción por software se inicia con la instrucción INT tipo, almacenando en el stack las banderas, el CS y el IP. A continuación carga los nuevos valores para CS e IP de una tabla de saltos a interrupciones, localizada de la dirección absoluta 00000H a la 003FFH. Ese kilobyte permite 256 diferentes interrupciones por software.

Ejemplo:

Si se ejecuta la instrucción INT 23H, el procesador multiplica 23H por 4 y obtiene la dirección 0008CH.

$$0010\ 0011 \times 4 = 1000\ 1100 = 008CH$$

El CS y el IP se cargarán con la palabra doble almacenada en las direcciones 0008CH-0008FH (CS con la palabra de alto orden e IP con la de bajo orden). Una rutina de servicio a una interrupción debe terminar con la instrucción IRET *(interrupt return)*, la cual saca del stack las banderas, el CS y el IP. Algunas localidades de la tabla de saltos a interrupciones se reservan para funciones especiales.

La instrucción INTO *(interrupt on overflow)* no necesita que se especifique el tipo de interrupción, ya que generará automáticamente una interrupción de tipo 4 si ha ocurrido una condición de overflow, después de haber realizado una instrucción aritmética signada.

CUADRO I.19. *Instrucciones* CALL *y* RETURN

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
CALL	Etiqueta cercana	E8_ _[a]	CALL MEMN[b]	Código	SP ← SP – 2; [SP + 1: SP] ← IP; IP ← MEMN	IP es colocado (PUSH) en la parte alta del stack, el control es transferido dentro del segmento a la dirección cercana.
		FF 16 00 10	CALL [MEMWDS][c]	Dato	SP ← SP – 2; [SP + 1: SP] ← IP; IP ← [1001H; 1000H]	
		FF 15	CALL [DI]	Dato	IP ← SP – 2; [SP + 1: SP] ← IP; IP ← [DI + 1: DI]	
		FF D7	CALL DI	Dentro de la CPU	SP ← SP – 2; [SP + 1: SP] ← IP; IP ← DI	
CALL	Etiqueta lejana	9A 00 10 D3 09	CALL FAR PTR MEMF[d]	Código	SP ← SP – 2; [SP + 1: SP] ← CS; CS ← 09D3H; SP ← SP – 2; [SP + 1: SP] ← IP; IP ← 1000H	CS e IP son colocados (PUSH) en la parte alta del stack y el control es transferido a un nuevo segmento y a una dirección lejana.
		FF 1E 00 10	CALL [MEMWWDS][e]	Dato	Igual que arriba, excepto: CS ← [1003H; 1002H];	
		FF 1D	CALL DWORD PTR [DI]	Dato	IP ← [1001H; 1000H] Igual que arriba, excepto: CS ← [DI + 3: DI + 2H]; IP ← [DI + 1H: DI]	

RET	n (cercano)[f]	C3	RET	Stack	IP ← [SP + 1: SP] SP ← SP + 2;	La palabra en la parte más alta del stack es obtenida (POP) dentro del control de transferencia IP a una nueva dirección; RET es normalmente usado para regresar el control a la instrucción siguiente de la llamada a la subrutina; si se incluye una obtención (POP) opcional el valor (n) es añadido al SP.
		C2 08 00	RET 8	Stack	IP ← [SP + 1: SP]; SP ← SP + 2 + 8	
RET	n (lejano)[f]	CB	RET	Stack	IP ← [SP + 1: SP]; SP ← SP + 2; CS ← [SP + 1: SP]; SP ← SP + 2	Es como arriba, excepto que la doble palabra de la parte alta del stack es obtenida (POP) del IP y el CS, transfiriendo el control a esta nueva dirección lejana.
		CA 08 00	RET 8	Stack	IP ← [SP + 1: SP]; SP ← SP + 2; CS ← [SP + 1: SP]; SP ← SP + 2 + 8	

[a] Estos dos bytes son el complemento a dos del offset entre la localidad de memoria cercana MEMN y la instrucción inmediata siguiente a la instrucción CALL. Por ejemplo, si CALL está 12 localidades adelante, el offset es 000C. Similarmente, 12 localidades atrás requiere un offset de FFF4.

[b] MEMN es asumido para apuntar a una localidad de memoria cercana, por ejemplo dentro del segmento.

[c] MEMWDS es asumida para apuntar a la palabra que empieza en la localidad 1000H en el segmento de datos. Los corchetes son opcionales porque MEMWDS define a una palabra, no a una localización para salto.

[d] FAR PTR indica que MEMF se localiza en un diferente segmento de código. En este caso se asume que MEMF apunta a la localidad 09D3H: 1000H.

[e] MEMWWDS es asumido para apuntar a la doble palabra empezando en la localización 1000H del segmento de datos. Los corchetes son opcionales porque MEMWWDS define a una doble palabra, no a una localización para salto.

[f] El mismo mnemónico es usado por un retorno cercano o lejano. El ensamblador generará el propio código basado en la declaración de la definición del procedimiento (cercano o lejano).

La ejecución de una instrucción de interrupción limpia las banderas IF y TF, asegurando que la rutina de servicio a la interrupción no se ejecutará paso a paso y ninguna interrupción externa mascarable (por el pin INTR) será reconocida. Las solicitudes de interrupción no-mascarable (pin NMI) sí serán aceptadas.

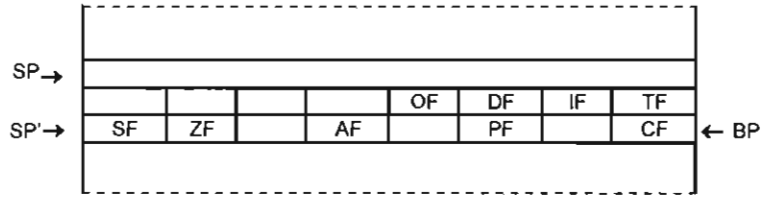


Figura I.10. Registro de banderas del 8086 en el área de stack.

NOTA: Los valores que tenían IF y TF en el programa principal no se pierden, ya que con la instrucción IRET se restaura el valor de las banderas.

Instrucciones de control del procesador

Se usan para el control del procesador y para limpiar o poner a "1" los indicadores de estado ("status"). Las banderas de *carry, direction* e *interrupt* pueden limpiarse o ponerse a "1" y el carry además puede ser *invertido*. Las banderas PF, AF, ZF y OF no se pueden accesar con este tipo de instrucciones. Las instrucciones STD *(set direction flag)* y CLD *(clear direction flag)* se usan con la bandera DF, que es utilizada con las instrucciones de movimiento de *string's*.

Las instrucciones STI *(set interrupt enable flag)* y CLI *(clear interrupt enable flag)* habilitan o deshabilitan las interrupciones mascarables (del pin INTR).

No hay instrucción para limpiar la bandera TF (genera una interrupción tipo 1 después de cada instrucción), lo cual posiblemente sea un error de Intel, pero se puede usar la siguiente secuencia:

```
PUSHF                          ; Almacena banderas en el stack.
MOV       BP, SP               ; BP apunta al tope del stack.
OR        BYTE PTR [BP + 1], 01 ; Pone a "1" el bit 0 (TF).
POPF                           ; Restaura el registro de banderas.
```

La instrucción HALT detiene al procesador y lo lleva a un *idle loop*, del cual sólo puede salir mediante una interrupción por hardware o un *system reset*. La instrucción NOP es "no-operation".

Existen tres instrucciones especiales de control usadas para coprocesadores:

a) La instrucción WAIT se usa para sincronizar el 8086/88 con el DP 8087 mediante la entrada TEST. Cuando el 8087 está ejecutando una instrucción y no puede aceptar un dato o instrucción del 8086/88 lleva a "1" lógico la entrada TEST. Así, si cada instrucción que requiere datos del 8087 es precedida por una instrucción WAIT, se asegura que los datos no se perderán entre los dos procesadores.

b) La instrucción LOCK evita que otros procesadores intenten controlar los buses del sistema. Previene que un coprocesador accese localidades de memoria que el 8086/88 esté accesando.

c) La instrucción ESC (escape) se usa como prefijo a las instrucciones del coprocesador, ya que el 8086/88 colocará en el bus de datos el operando, el cual es esperado por el 8087. El 8086/88 activa, o le avisa, al 8087 que ya está el dato con el prefijo ESC, lo que causa que el coprocesador lea el dato y comience la ejecución. Más adelante veremos los coprocesadores 8087 y 8089.

NOTA: el prefijo ESC enviado por el 8086/88 al 8087 identifica el tipo de operación que debe realizar este último.

CUADRO I.20. *Instrucciones de interrupción por software*

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
INT	Tipo	CD 23	INT 23H	Stack y salto a la tabla de interrupciones a 00000-003FFH	SP ← SP – 2; [SP + 1: SP] ← Flags IP ← 0; TF ← 0; SP ← SP – 2; [SP + 1: SP] ← CS CS ← [0008FH: 0008EH];[a] SP ← SP – 2 [SP + 1: SP] ← IP; IP ← [0008DH: 0008CH][b]	Salva las banderas y los registros CS e IP en el stack y transfiere el control a la dirección lejana almacenada en la doble palabra que empieza en la dirección absoluta tipo * 4.
INTO	Ninguno	CE	INTO	Stack y salto a la tabla de interrupciones a 00000-003FFH	Si OF = 1, entonces SP ← SP – 2; [SP + 1: SP] ← Flags IF ← 0; TF ← 0; SP ← SP – 2; [SP + 1: SP] ← CS; CS ← [00013H: 00012H];[a] SP ← SP – 2 [SP + 1: SP] ← IP; IP ← [00011H: 00010H][b]	Si existe una condición de sobreflujo (OF = 0), se ejecuta una interrupción de tipo 4.
IRET	Ninguno	CF	IRET	Stack	IP ← [SP + 1: SP]; SP ← SP + 2; CS ← [SP + 1: SP]; SP ← SP + 2; Flags ← [SP + 1: SP]; SP ← SP + 2;	Regresa el control al punto de interrupción por medio de la obtención (POP) de los registros IP, CS y el registro de banderas del stack; IRET es usada normalmente para salir de cualquier procedimiento de interrupción, ya sea activado por hardware o por software

[a] [(tipo * 4) + 3: (tipo * 4) + 2].
[b] [(tipo * 4) + 1: (tipo * 4)].

CUADRO I.21. *Instrucciones de control del procesador*

Mnemónico general			*Ejemplo de codificación*			
Código de operación	*Operando*	*Código objeto*	*Mnemónico*	*Segmento para acceso a memoria*	*Operación simbólica*	*Descripción*
STC	Ninguno	F9	STC	Dentro de la CPU	CF ← 1	Coloca la bandera de carry.
CLC	Ninguno	F8	CLC	Dentro de la CPU	C F ← 0	Limpia la bandera de carry.
CMC	Ninguno	F5	CMC	Dentro de la CPU	CF ← $\overline{CF}$	Complementa la bandera de carry.
STD	Ninguno	FD	STD	Dentro de la CPU	DF ← 1	Coloca la bandera de dirección (autodecrementa por instrucciones de cadena).
CLD	Ninguno	FC	CLD	Dentro de la CPU	DF ← 0	Limpia la bandera de dirección (autodecrementa por instrucciones de cadena).
STI	Ninguno	FB	STI	Dentro de la CPU	IF ← 1	Coloca la bandera de interrupción (habilitando interrupciones sobre la línea INTR).
CLI	Ninguno	FA	CLI	Dentro de la CPU	IF ← 0	Limpia la bandera de interrupción (habilitando interrupciones sobre la línea INTR).
HLT	Ninguno	F4	HLT	Dentro de la CPU	Ninguno	Halt
WAIT	Ninguno	9B	WAIT	Dentro de la CPU	Ninguno	Entra en estado de espera si la línea de $\overline{TEST}$ = 1.
LOCK	Instrucción	F0 A1 00 10	LOCK MOVE AX, MEMWDS[a]	Dato	Ninguno	La línea de salida $\overline{LOCK}$ = 0 mientras la siguiente instrucción LOCK ejecuta: usada para prevenir coprocesos de acceso al bus durante una instrucción particular.
NOP	Ninguno	90	NOP	Dato	Ninguno	No operación.
ESC	Número, fuente	DE 0E 00 10	ESC 31H, MEMWDS[a]	Dato	Bus de datos ← [MEMWDS].	Coloca el contenido del operando en memoria en el bus de datos y ejecuta un NOP; el primer operando identifica a una instrucción de escape particular para ser ejecutada por un coproceso.

[a] MEMWDS es asumido para apuntar a la palabra que inicia en la localidad 1000H en el segmento de datos.

TÉCNICAS DE PROGRAMACIÓN

Ejemplo 1

Realice un programa para controlar el siguiente sistema en el que se debe calentar una solución química antes de pasar a la siguiente etapa.

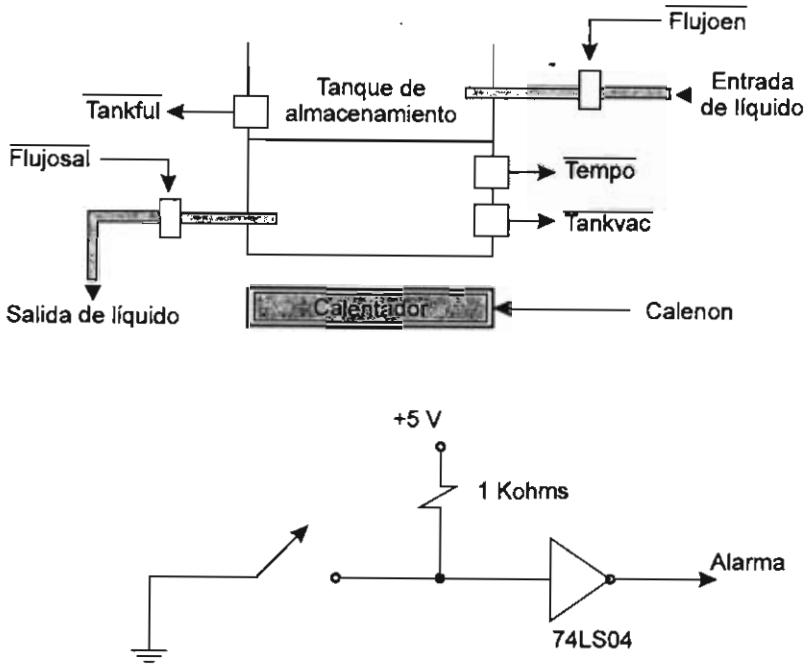


Figura I.11. Diagrama de bloques del ejemplo 1.

Especificaciones del programa

1. Para llenar el tanque:
   a) Abra la válvula: $\overline{\text{FLUJOEN}} = 0$
   b) Cierre válvula salida: $\overline{\text{FLUJOSAL}} = 1$
   c) Apague calentador: CALENON = 0
   d) Espere hasta que el tanque se llene: TANKFUL = 1

2. Caliente solución de tanque hasta que la temperatura sea de 90°C.
   a) Cierre válvula de entrada: $\overline{\text{FLUJOEN}} = 1$
   b) Encienda calentador: CALENON = 1
   c) Espere a que se caliente la solución: $\overline{\text{TEMPOK}} = 0$

3. Vacíe el tanque.
   a) Apague el calentador: CALENON = 0
   b) Abra válvula de salida: $\overline{\text{FLUJOSAL}} = 0$
   c) Espere a que el tanque se vacíe: $\overline{\text{TANKVAC}} = 0$
4. Regrese a la etapa 1.

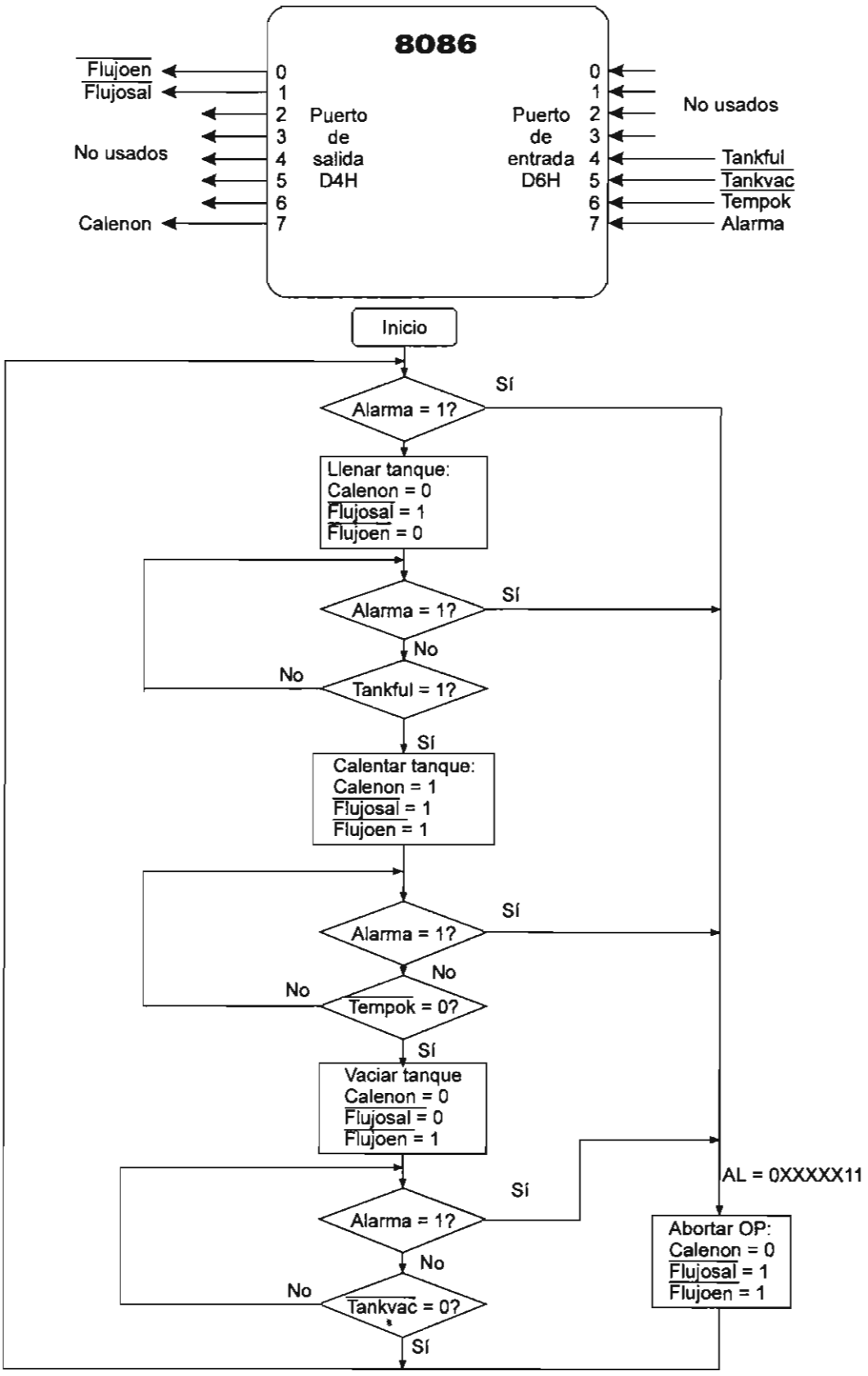


Figura I.12. Diagrama de flujo del ejemplo 1.

Programa núm. 1

```
; Este programa controla un proceso industrial con un
; elemento calefactor y una entrada y salida de líquido

; Asignación de constantes a etiquetas:

PSTAT           EQU             0D6H            ;Puerto de status.
PDAT            EQU             0D4H            ;Puerto de datos.
ALARMA          EQU             80H             ;Máscara de alarma.
TANKFUL         EQU             10H             ;Máscara de tanque lleno.
TEMPOK          EQU             40H             ;Máscara de temperatura OK.
TANKVAC         EQU             20H             ;Máscara de tanque vacío.
LLENATANK       EQU             02H             ;Código de llenar tanque.
CALENTANK       EQU             83H             ;Código de calentar tanque.
VACIATANK       EQU             01H             ;Código de vaciar tanque.
APAGA           EQU             03H             ;Código de apagar todo.

;Programa principal: le da el nombre "CODE" al segmento
;donde se almacenará el programa.

CODE  SEGMENT
      ASSUME    CS: CODE

;Prueba si hay "alarma" y llena el tanque:

INICIO:         IN              AL, PSTAT       ;Lee sensores.
                TEST            AL, ALARMA      ;¿Alarma?
                JNZ             SHTDWN          ;Shutdown.
                MOV             AL, LLENATANK   ;Si no, llena el tanque
                OUT             PDAT, AL
LLENA:          IN              AL, PSTAT       ;Lee sensores.
                TEST            AL, ALARMA      ;¿Alarma?
                JNZ             SHTDWN          ;Shutdown.
                TEST            AL,TANKFUL      ;¿Está lleno el tanque?
                JZ              LLENA           ;No continúa.

;El tanque está lleno, cierra la válvula de entrada y enciende
;el calentador

                MOV             AL, CALENTANK   ;Enciende calentador.
                OUT             PDAT, AL
HEAT:           IN              AL, PSTAT       ;Lee sensores.
```

```
                TEST            AL, ALARMA      ;¿Alarma?
                JNZ             SHTDWN          ;Shutdown.
                TEST            AL, TEMPOK      ;¿Temperatura OK?
                JNZ             HEAT            ;No continúa.
;El tanque ha sido calentado. Ahora abre la válvula de salida
;y vacía el tanque.

                MOV             AL, VACIATANK   ;Abre la válvula.
                OUT             PDAT, AL
VACÍO:          IN              AL, PSTAT       ;Lee sensores.
                TEST            AL, ALARMA      ;¿Alarma?
                JNZ             SHTDWN          ;Shutdown.
                TEST            AL, TANKVAC     ;¿Tanque vacío?
                JNZ             VACÍO           ;No continúa.
                JMP             INICIO          ;El tanque está vacío.
                                                ;Repite el ciclo.
;Rutina de shutdown. Todo se apaga.

SHTDWN          MOV             AL, APAGA
                OUT             PDAT, AL
                JMP             INICIO
CODE            ENDS                            ;Fin de segmento CODE.
                END             INICIO          ;Fin del programa.
```

INICIO: le dice al ligador cuál será la primera instrucción a ejecutar.

Las instrucciones del programa se definen a existir en un segmento llamado CODE. El ensamblador es avisado de que CS = CODE para las siguientes instrucciones (hasta el CODE ENDS). El programa termina con END INICIO, lo cual le dice al programa ligador cuál será la primera instrucción a ejecutarse.

En este programa sólo existe un segmento de código, pero posteriormente veremos algunos casos en los que no es así. Por esto, es buen hábito identificar el punto de inicio del programa para el ligador.

Ejemplo 2

Creación de una tabla de saltos

Una técnica muy común de programación para brincar a una de varias rutinas, dependiendo de una tecla de selección, es procesar el dato de entrada (tecla) y usarlo como índice a una tabla de saltos. En este ejemplo sólo usaremos 4 teclas (del 1 al 4 en un teclado ASCII). Si la tecla es correcta, los bits de alto orden son enmascarados para extraer de la tabla la dirección apropiada de la rutina.

El valor del dato se multiplica por 2 porque hay 2 bytes por cada entrada en la tabla. La tabla también tiene un offset de dos localidades relativo al valor de las teclas. Por ejemplo, la dirección de la rutina 1 se almacena en la dirección 0, por lo que es necesario restarle 2.

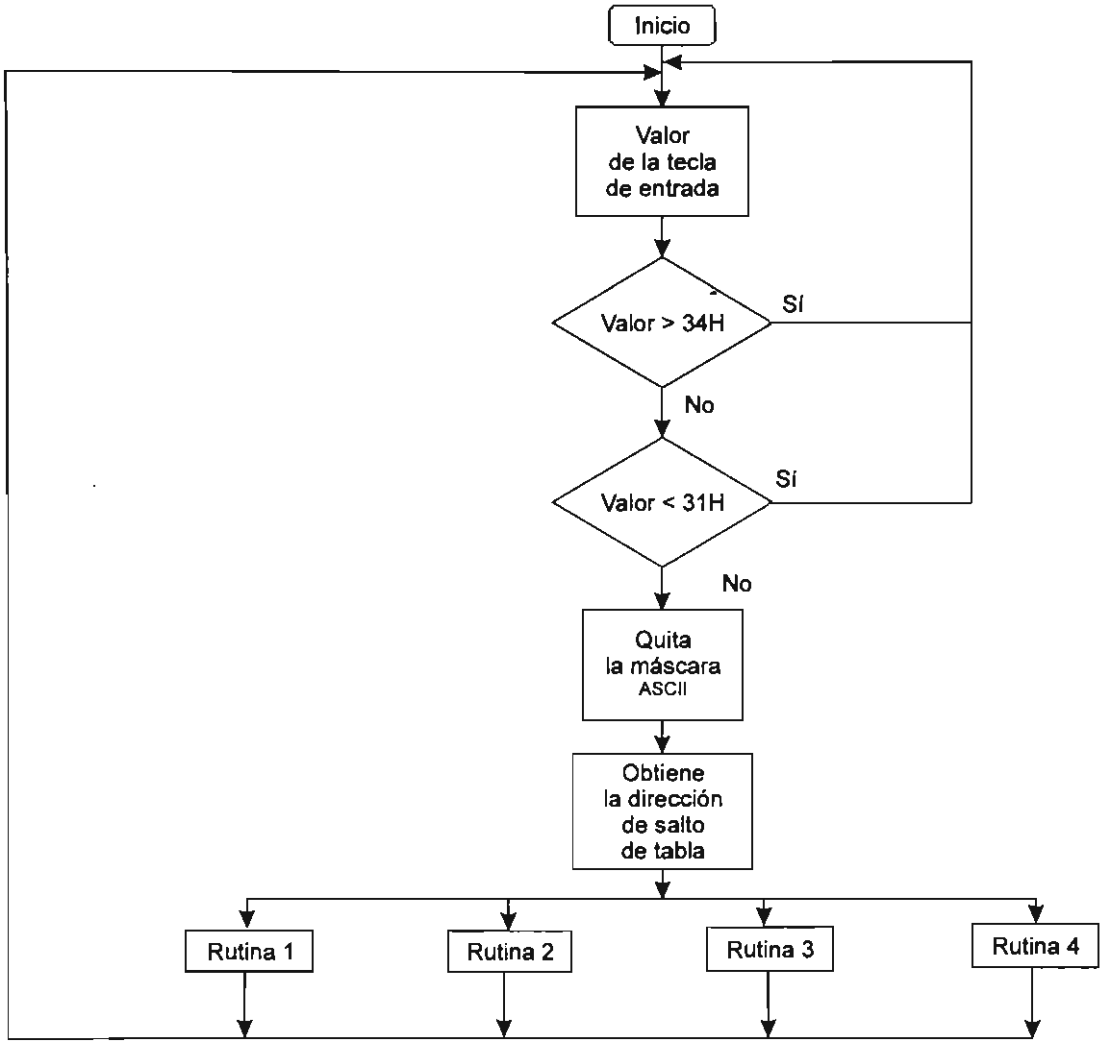


Figura I.13. Diagrama de flujo del ejemplo 2.

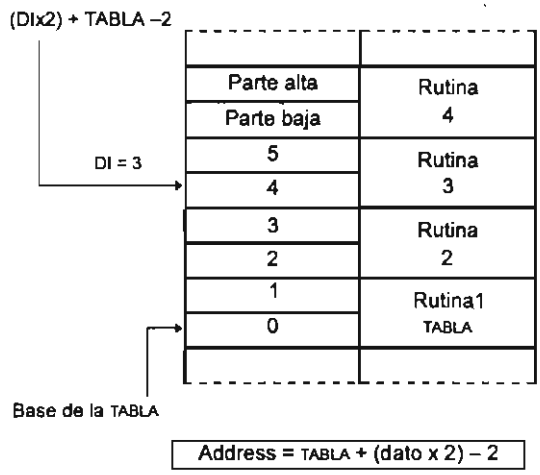


Figura I.14. Tabla de direcciones de rutinas del ejemplo 1.

Programa núm. 2

```
;Este programa lee el valor ASCII de una tecla desde un
;puerto de entrada. Si el valor de la tecla está entre 1
;y 4, el control es transferido a la rutina cuya dirección
;se encuentra almacenada en una tabla de salto.

PENTRADA    EQU         0D4H                ;Puerto de entrada.
MASK        EQU         07H                 ;Máscara ASCII.

; Define el segmento de datos llamado TABLA para almacenar la
; dirección de la primera instrucción.

TABLA:      SEGMENT
            DW          RUTINA 1
            DW          RUTINA 2
            DW          RUTINA 3
            DW          RUTINA 4
TABLA                   ENDS

CODE  SEGMENT
      ASSUME            CS: CODE, DS:TABLA

;Prueba si hay "alarma" y llena el tanque:

INICIO:     IN          AX, TABLA           ;El valor del DATA SEG.
            MOV         DS, AX              ;al regs. DS.
TECLA       IN          AL, PENTRADA        ;Lee el valor de tecla.
            CMP         AL, 34H             ;¿Muy grande?
            JA          TECLA               ;Lee otra vez.
            CMP         AL, 31H             ;¿Muy pequeño?
            JB          TECLA               ;Lee otra vez.

;Tecla válida. Busca la dirección de la tabla:

            AND         AL, MASK            ;AL tiene los bits 1-4.
            MOV         AH, 0               ;Offset en AX.
            MOV         DI, AX              ;Offset en DI.
            SHL         DI, 1               ;2 veces.
            JMP         [TABLA + DI-2]      ;Transfiere el control a
                                            ;la rutina.
;Las 4 rutinas se deben colocar aquí. Cada una termina con
;un salto a TECLA.
```

```
RUTINA: 1                                   ;Aquí se coloca el código
          ___________                       ;para la primera rutina.
          ___________
          ___________
          ___________
          JMP         TECLA                 ;Fin de la rutina.
RUTINA 2: ___________                       ;Aquí se coloca el código
          ___________                       ;para la segunda rutina.
          ___________
          ___________
          JMP         TECLA                 ;Fin de la rutina.
RUTINA 3: ___________                       ;Aquí se coloca el código
          ___________                       ;para la tercera rutina
          ___________
          ___________
          ___________
          JMP         TECLA                 ;Fin de la rutina.
RUTINA 4: ___________                       ;Aquí se coloca el código
          ___________                       ;para la cuarta rutina.
          ___________
          ___________
          JMP         TECLA
                                            ;Fin de la rutina.

CODE      ENDS                              ;Fin de segmento CODE.
          END INICIO                        ;Fin del programa.
```

El valor del dato se multiplica por 2 porque hay 2 bytes por cada entrada en el cuadro. El cuadro también tiene un offset de 2 localidades relativo al valor de las teclas. Por ejemplo, la dirección de la *rutina 1* se almacena en la dirección 0, por lo que es necesario restarle 2.

En el segmento creado y llamado TABLA se reserva una palabra para la dirección de cada rutina. Al encontrar la declaración DW RUTINA1, el ensamblador automáticamente inserta la dirección de la rutina 1 en la localidad.

Alineación de segmentos

Después de que el programa anterior se ensambla, debe ligarse con el comando LINK (suponiendo que el archivo se llame SALTO): LINK SALTO.OBJ. El resultado es un archivo ejecutable (SALTO.EXE) y (opcionalmente) un mapa de ligas *(link map)*. Este mapa indica las direcciones relativas de los diferentes segmentos del programa. Por ejemplo, el mapa para el programa anterior es como sigue:

START	STOP	LENGTH	NAME
00000H	00022H	0023H	CODE
00030H	00037H	0008H	TABLA

Se observa que las longitudes corresponden a las de los segmentos TABLA y CODE. Las direcciones son relativas y se determinan hasta que el programa se carga en memoria.

En el mapa anterior se puede ver que hay un hueco ("gap") al final del segmento CODE de 00023H a 0002FH. Entonces, ¿por qué el ligador no empezó el segmento TABLA inmediatamente después de que termina el segmento CODE?, y ¿por qué "desperdicia" 13 bytes?

La razón es que en la declaración TABLA SEGMENT no se incluyó lo que se llama "tipo de alineación del segmento", el cual le indica al ligador que el segmento es unido con el segmento anterior.

En la figura siguiente se muestra el resultado de usar el DEBUG para desensamblar el programa anterior de saltos. En la columna de la izquierda se ve que el programa se empezó a cargar en la dirección 09E3:0000 (CS = 09E3).

El hueco de 16 bytes se llenó con 0 y la tabla comienza en la dirección 09E3:0030H. La dirección física de la última instrucción del segmento de código es 09E30H + 0022H = 09E52H y el segmento TABLA se carga en la dirección del siguiente párrafo disponible: 09E30 + 0030H = 09E60H; es por esto que en la primera instrucción puede observarse que `MOV AX, TABLE`, donde la dirección del segmento es 09E6H.

```
                        MOV AX, 09E6
                        MOV DS, AX

09E3: 0000      B8E609      MOV     AX, 09E6
09E3: 0003      8ED8        MOV     DS, AX
09E3: 0005      E4D4        INB     D4
09E3: 0007      3C34        CMP     AL, 34
09E3: 0009      77FA        JA      0005
09E3: 000B      3C31        CMP     AL, 31
09E3: 000D      72F6        JC      0005
09E3: 000F      2407        AND     AL, 07
09E3: 0011      B400        MOV     AH, 00
09E3: 0013      8BF8        MOV     DI, AX
09E3: 0015      D1E7        SHL     DI
09E3: 0017      FFA5FEFF    JMP     [DI + FFFE]
09E3: 001B      EBE8        JMPS    0005
09E3: 001D      EBE6        JMPS    0005
09E3: 001F      EBE4        JMPS    0005
09E3: 0021      EBE2        JMPS    0005
09E3: 0023      0000        ADD     [BX + SI], AL
09E3: 0025      0000        ADD     [BX + SI], AL
09E3: 0027      0000        ADD     [BX + SI], AL
09E3: 0029      0000        ADD     [BX + SI], AL
09E3: 002B      0000        ADD     [BX + SI], AL
09E3: 002D      0000        ADD     [BX + SI], AL
09E3: 002F      001B        ADD     [BP + DI], BL
09E3: 0031      001D        ADD     [DI], BL
09E3: 0033      001F        ADD     [BX], BL
09E3: 0035      0021        ADD     [BX + DI], AH
09E3: 0037      0008        ADD     [BX + SI], CL
```

Por otra parte, la dirección del salto al segmento TABLA es JMP [DI + FFFE], ya que el offset de la tabla es 0000 (porque fue cargada en el inicio del párrafo). La dirección del salto indirecto es entonces

0000 + DI – 2 = DI + FFFEH. La alineación del segmento es pues definida en la declaración SEGMENT, cuya forma general es la siguiente:

nombre SEGMENT tipo-de-alineación tipo-de-combinación clase

El tipo-de-alineación puede ser byte, word o párrafo. El valor por default es párrafo. A continuación se muestra el resultado de cambiar la definición del segmento de la tabla de saltos por:

```
TABLA           SEGMENT     WORD

09E3: 0000      B8E509      MOV         AL, 09E5
09E3: 0003      8ED8        MOV         DS, AX
09E3: 0005      E4D4        INB         D4
09E3: 0007      3C34        CMP         AL,34
09E3: 0009      77FA        JA          0005
09E3: 000B      3C31        CMP         AL,,31
09E3: 000D      72F6        JC          0005
09E3: 000F      2407        AND         AL, 07
09E3: 0011      B400        MOV         AH, 00
09E3: 0013      8BF8        MOV         DI, AX
09E3: 0015      D1E7        SHL         DI
09E3: 0017      FFA50200    JMP         [DI + 0002]
09E3: 001B      EBE8        JMPS        0005
09E3: 001D      EBE6        JMPS        0005
09E3: 001F      EBE4        JMPS        0005
09E3: 0021      EBE2        JMPS        0005
09E3: 0023      001B        ADD         [BP + DI], BL
09E3: 0025      001D        ADD         [DI], BL
09E3: 0027      001F        ADD         [BX], BL
09E3: 0029      0021        ADD         [BX + DI], AH
09E3: 002B      001C        ADD         [SI], BL
```

```
BASE + DI-2 =
0000 + DI + FFFE =
0004 + DI-2 = DI + 2
```

El ligador elige ahora la primera dirección par (el tipo-de-alineación es por palabra) que sigue a la ultima instrucción del segmento CODE, para comenzar la tabla de saltos. En este caso el offset es 0024H (el único byte "desperdiciado" es el 0023H).

AX se cargó ahora con 09E5H porque TABLA comienza en la dirección física 09E3:0024 = 0930H + 0024H = 09E54H. Es decir, TABLA está en el segmento cuya dirección es 09E5H y tiene un offset de 0004 con respecto al inicio del segmento (09E50H). Por esto AX se carga con 09E5H, la BIU le adiciona un 0 y le suma el offset de tabla.

Conclusión

Puede parecer extraño en primera instancia, aunque no siempre, pero una tabla se carga a partir del offset o dirección 0000H de un segmento. Todo esto parece muy complicado, pero nos ayuda a conocer cómo alinea el ligador los segmentos de un programa "multi-segmentos". Esto también es importante cuando se desea optimizar el uso de la memoria.

Parece mejor alinear segmentos por byte, pues así no habría bytes desperdiciados, pero para el 8088 es aceptable, no así para el 8086, que requiere un ciclo de bus extra (4 ciclos de reloj) para accesar una palabra que comience en una dirección impar. Así, los segmentos de datos orientados a palabra (incluyendo stack) deben alinearse por palabra.

Ejemplo 3

Cálculo del tiempo de ejecución de un loop

Cada instrucción requiere un número específico de ciclos de reloj o estados T. Este número varía de acuerdo a la complejidad de la instrucción (la instrucción NOP sólo requiere tres estados T). Esto también depende del modo de direccionamiento.

Ejemplo:

Calcule el tiempo requerido para ejecutar la instrucción MOV AX, BX si la frecuencia del reloj del sistema es 5Mhz. Los números encerrados entre paréntesis representan estados T para el 80186.

CUADRO I.22. *Ciclos de reloj usados por instrucciones* MOV

Operandos de MOV	*Clocks*	*Transferencia*	*Bytes*	*Ejemplos de* MOV
Memoria, acumulador	10(9)	1	3	MOV ARRAY AL
Acumulador, memoria	10(8)	1	3	MOV AX, TEMP_RESULT
Registro, registro	2(2)	–	2	MOV AX, CX
Registro, memoria	8(12) + EA	1	2-4	MOV BP, STACK_TOP
Memoria, registro	9(9) + EA	1	2-4	MOV COUNT [DI], CX
Registro, inmediato	4(3-4)	–	2-3	MOV CL, 2
Memoria, inmediato	10(12-13) + EA	1	3-6	MOV MASK [BX] [S1], 2CH
Seg-reg, reg 16	2(2)	–	2	MOV ES, CX
Seg-reg, mem 16	8(9) + EA	1	2-4	MOV DS, SEGMENT_BASE
Reg16, seg-reg	2(2)	–	2	MOV BP, SS
Memoria, seg-reg	9(11) + EA	1	2-4	MOV [BX], SEG_SAVE, CS

Solución: se requieren 2 estados T o $2 \times 1/5\text{MHz} = 400$ ns.

Las instrucciones que calculan una EA requieren una cantidad adicional de tiempo según se muestra a continuación.

Ejemplo:

La instrucción MOV CX, [BX + SI + 3] requiere 8 + 11 = 19 estados T, 8 para la instrucción y 11 para el cálculo de EA.

CUADRO 1.23. *Ciclos de reloj usados para obtener operandos*

	EA componentes	*Clocks*[a]
Sólo desplazamiento		6
Sólo base o índice	(BX, BP, SI, DI)	5
Desplazamiento + base o índice	BX, BP, SI, DI)	9
Base + índice	BP + DI, BX + SI BP + SI, BX + DI	8
Desplazamiento + base + índice	BP + DI + DESP BX + SI + DESP	11
	BP + SI + DESP BX + DI + DESP	12

[a] Adicionar dos clocks para segment override.

Los estados T de las instrucciones suponen que la instrucción ya se encuentra en la queue. De lo contrario se deberán sumar cuatro estados T y también existirá el caso de que esta instrucción requiera un operando de memoria, circunstancia en que tendrán que sumarse otros cuatro estados T. El 8088 requiere dos ciclos de bus para accesar palabras en memoria, lo que implica que tendremos que sumar cuatro estados T a los tiempos de ejecución del 8086 para cada transferencia de 16 bits con el 8088.

Ejemplo:

La instrucción MOV CX, [BX + SI + 3] requiere 23 estados T con el 8088.

Así también, si el acceso a una palabra es a una dirección impar, se deben sumar cuatro estados T a la ejecución de la instrucción.

Ejemplo:

Calcule el parámetro RETARDO en el siguiente lazo, para producir un tiempo de retardo de 500 microseg con un reloj del sistema de 5MHz.

```
                MOV CX, RETARDO     [4]
CUENTA:         LOOP CUENTA         [17/5]
                 _ _ _ _ _ _ _      Estados T.
                 _ _ _ _ _ _ _
                 _ _ _ _ _ _ _
```

Solución: la instrucción LOOP toma 17 estados T si se realiza el salto, pero sólo 5 si no se realiza. Por tanto:

$$t = [4 + 17(\text{RETARDO}-1) + 5] \times 200 \text{ ns} = [4 + (17 \times \text{RETARDO}) - 17 + 5] \times 200 \text{ ns}$$
$$t = [4 + (17 \times \text{RETARDO}) - 12] \times 200 \text{ ns} = (3.4 \text{ microseg} \times \text{RETARDO}) - 1.6 \text{ microseg}$$

Si t = 500 mseg
entonces

$$\text{RETARDO} = 147$$

Ejemplo 4

Generador de onda cuadrada (1 KHz). Se usará un bit de un puerto de salida del 8086; este bit permanece en cada estado ("0" o "1" lógico) durante un periodo de 500 microsegundos.

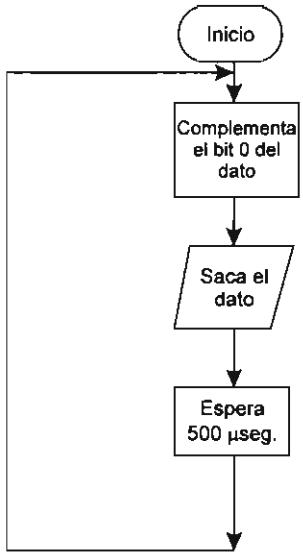


Figura I.15. Diagrama de flujo del ejemplo 4.

Este algoritmo usa el retardo calculado en el ejemplo anterior, pero cambiándolo a 146 en lugar de 147 debido a las instrucciones adicionales.

Un KHz es una frecuencia que se encuentra dentro del rango audible, por lo que la salida se puede conectar a una bocina indirectamente:

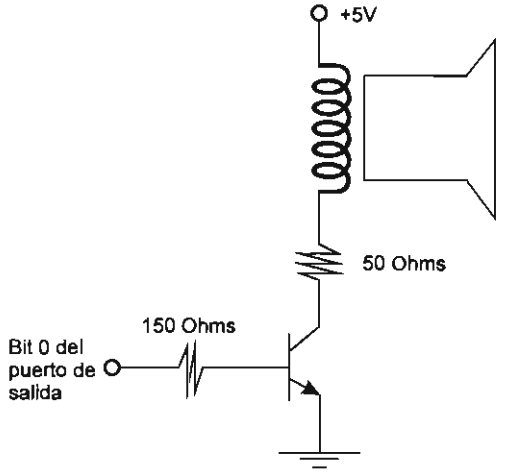


Figura I.16. Conexión de un bit del puerto de salida a una bocina.

```
The Microsoft MACRO Assembler                                   15-09-92
PAGE 1-1
                                Programa núm. 4
;Generador de onda cuadrada de 1 Khz. Versión 1.0
;Este programa genera una onda cuadrada de 1Khz por el
;puerto de salida 0D4H. Supone un µP 8086 de 5Mhz. Los
;estados T del 8086 se muestran del lado derecho en
;brackets

= 00D4          PSAL        EQU         0D4H        ;Puerto de salida.
= 0092          T1          EQU         146         ;Valor de retardo.

0000            CODE  SEGMENT
                      ASSUME      CS: CODE

0000 F6 D0      INICIO:     NOT         AL          ;Complementa [3]
0002 E6 D4                  OUT         PSAL, AL    ;Bit de salida [10]

;Las siguientes instrucciones generan un retardo de
;500 µseg (1/2 periodo).

0004 B9 0092                MOV         CX, T1      ;Carga contador. [41]
0007 E2 FE      CUENTA:     LOOP        CUENTA      ;Decrementa contador
                                                    ;hasta CX = 0. [17/5]
0009 E2 FE                  JMP         INICIO      ;Repite para el [15]
                                                    ;siguiente medio ciclo.
              ;Los estados T = 32 + (17*CX - 12) = 32 + 17CX - 12.
              ;Si C = 146, entonces estados T = 2502. A 5 Mhz, +
              ;Retardo = 2502* 200 ns = 500.4µs.
000B            CODE        ENDS                    ;Fin de segmento CODE.
                            END         INICIO      ;Fin del programa.
```

El tiempo que tarda en ejecutarse este bloque es:

$$= [3] + [10] + [4] + 17[CX - 1] + 5 + 15.$$

$$= 17 + 17CX - 17 + 20.$$

$$= 17CX + 20.$$

Procedimientos ("Procedures")

Se usan cuando ciertas secuencias de instrucciones se requieren varias veces en el mismo programa.

Los procedimientos (o subrutinas) se invocan con una instrucción CALL, la cual actúa como un JUMP pero salva en el stack el IP (y el CS para un far call), y facilitan la programación modular (un problema se divide en varios módulos pequeños o "procedimientos"). Por ejemplo, el programa controlador del proceso del ejemplo 1 se puede escribir de la siguiente forma modular:

```
INICIO:    CALL LLENA
           CALL HEAT
           CALL VACÍA
           JMP INICIO
```

Transferencia de parámetros a un procedimiento

Existen varias formas. Una de ellas es usando los registros de la CPU.

Ejemplo:

```
MOV CX, T1
CALL RETARDO
```

Donde T1 es el parámetro para el retardo.
Otra técnica es por medio de una localidad de memoria:

```
MOV TEMP, T1
CALL RETARDO
```

Una variación de esta ultima técnica es pasar la dirección de la localidad de memoria de la variable:

```
MOV SI, POINTER
CALL RETARDO
```

El "procedimiento" extrae el parámetro con una instrucción del tipo MOV CX, [SI] y esta técnica tiene la ventaja de poder pasar una tabla de datos entera. La desventaja de las dos primeras técnicas es que se usa registro o localidad de memoria dedicados para almacenar el parámetro cuando se llama al procedimiento; esto se puede volver confuso y complicado cuando un procedimiento llama a otro (procedimientos "anidados"). Una alternativa es usar el stack para pasar parámetros:

```
MOV DX, T1
PUSH DX
CALL RETARDO
```

Ejemplo 5

En el siguiente diagrama de flujo se muestra cómo producir un "beep" corto (de aproximadamente 0.25 seg) con una bocina conectada a un bit de un puerto de salida, lo cual puede ser útil como aviso de que ha ocurrido un error. El "beep" es de 500 medios ciclos de 1 KHz y el retardo es un procedimiento que toma el parámetro (tiempo de retardo) del stack.

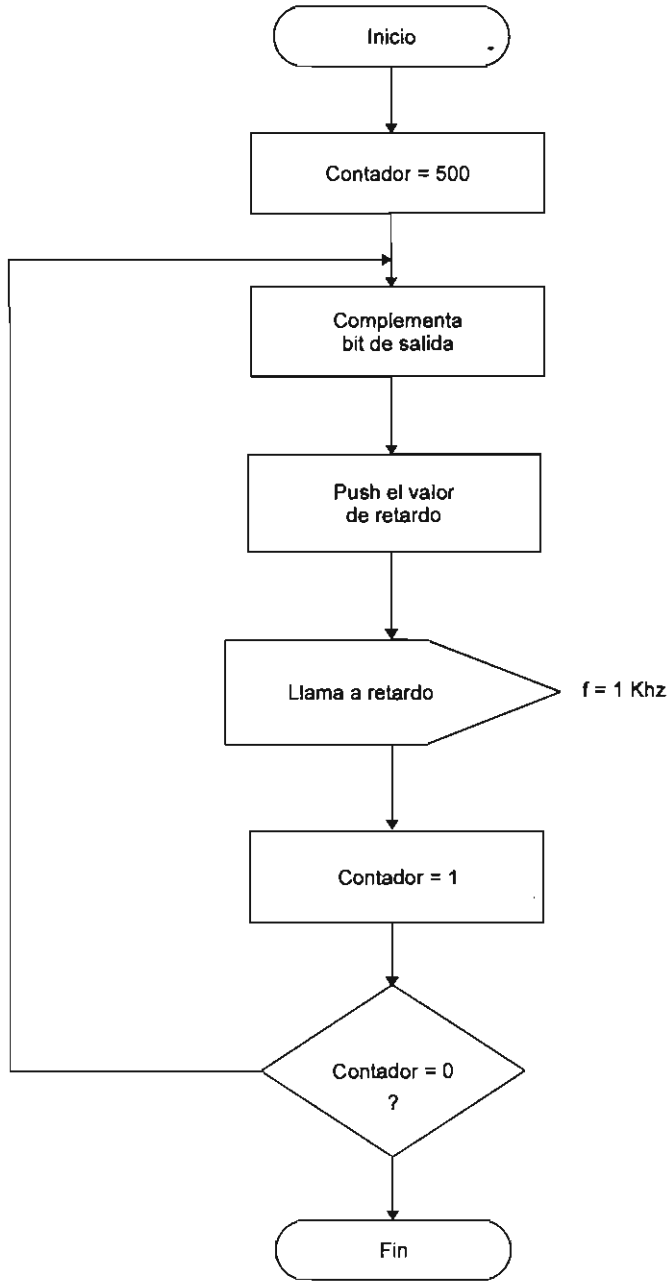


Figura I.17. Diagrama de flujo del ejemplo 5.

```
The Microsoft MACRO Assembler                                   15-09-92
PAGE 1-2
Beep V1.0
            TITLE                   BEEP        V1.0
                                Programa no. 5
;Este programa produce un beep de 0.25 seg usando una bocina
;conectada al bit 0 del puerto de salida 0D4H.
= 00D4      PSAL        EQU         0D4H        ;Puerto de salida.
= 00BE      T1          EQU         146H        ;500 µs de retardo.
= 01B4      CICLOS      EQU         500         ;Número de ciclos.

;Inicialización del segmento de stack.

0000        STACK       SEGMENT     WORD        STACK
0000 40  [              DW          64          DUP (?)
      ????]
0080        TOP         LABEL       WORD
0080        STACK       ENDS

;Define el segmento de código llamándole CODE.

0000        CODE          SEGMENT
                          ASSUME    CS: CODE, SS: STACK
0000 B8     --R INICIO:     MOV     AX, STACK   ;Carga el segmento del
0003 8E     D0              MOV     SS, AX      ;stack en SS.
0005 36:    8D26 0080R      LEA     SP, TOP     ;Apunta al inicio del
                                                stack.
000A B9     01F4            MOV     CX, CICLOS  ;Contador de ciclos.
000D F6     D0 OTRO:        NOT     AL          ;Complementa el bit de
000F E6     D4              OUT     PSAL, AL    ;salida.
0011 BA     008E            MOV     DX, T1    * ;Coloca el retardo.
0014 52                     PUSH    DX          ;en el stack.
0015 E8     001B R          CALL    RETARDO     ;Espera 1/2 ciclo.
0018 E2     F3              LOOP    OTRO        ;Hazlo CX veces.
001A F4                     HLT
;Aquí termina el programa principal. Sigue el procedimiento
;de retardo.
;Función: Retardo de tiempo.
;Entradas: Valor del retardo de 16 bits pasados en el stack.
;          T = [3.4 µs * T1] + 17 µs.
;Salidas: Ninguna.
```

```
The Microsoft MACRO Assembler
PAGE 2-2                                                    15-09-92
Beep        V1.0
            ;Llamadas:  Ninguna.
            ;Destruye:  Ninguna.
001B        RETARDO     PROC        NEAR
001B 51                 PUSH        CX          ;Salva todos los
001C 9C                 PUSHF                   ;registros y banderas.
001D 55                 PUSH        BP          ;
001E 8B EC              MOV         BP, SP      ;Toma del stack el
                                                ;valor
0020 8B 4E 08           MOV         CX, [BP + 8] ;de retardo.
0023 E2 FE  CUENTA:     LOOP        CUENTA      ;Loop de retardo.
0025 DE                 POP         BP          ;Restaura todos los
0026 9D                 POPF                    ;registros.
0027 59                 POP         CX
0028 C2     0002        RET         2           ;Descarta el
                                                ;parámetro de
                                                ;retardo.
002B        RETARDO     ENDP
;Aquí termina el procedimiento de retardo.
002B        CODE        ENDS                    ;Fin de segmento CODE.
                        END         INICIO      ;Fin del programa.
002B
```

El programa consiste en dos segmentos. Uno es STACK, el cual es alineado por palabra y consiste en 64 palabras *(words)* no definidas. La palabra STACK que está al final de la declaración SEGMENT identifica a este segmento como "stack combinable" (la opción tipo-de-combinación usada en la declaración del segmento, la veremos un poco más adelante). El otro segmento se llama CODE y contiene al programa principal y al procedimiento de RETARDO. El programa comienza con la etiqueta INICIO y carga al SS con la dirección del segmento de STACK. La declaración TOP LABEL WORD asigna la etiqueta TOP a la primera palabra *(topmost)* del stack segment, lo cual permite a la instrucción LEA SP, TOP cargar al SP con la dirección efectiva del tope del stack. Aquí existe un "segment override" generado automáticamente por el ensamblador, ya que LEA normalmente hace referencia al segmento de datos, pero TOP fue definido en el STACK segment.

Continuando con el programa, éste hace un "near call" a RETARDO (RETARDO está en el mismo segmento del programa principal). El procedimiento RETARDO requiere un parámetro de RETARDO de tiempo, el cual es un poco diferente del ejemplo anterior gracias a los PUSH y POP.

En la figura I.18 se muestra cómo el parámetro (T1) es pasado en el STACK. Entonces, si suponemos que inicialmente el SP apunta a la localidad 0080H y si el programa principal deposita (push) en el stack DX (T1) y realiza un near call a RETARDO, el SP se decrementa a 007CH. A continuación, RETARDO salva todos los registros que usara (CX, banderas y BP) y el SP se decrementa a 0076H.

Entonces el procedimiento RETARDO recupera el parámetro copiando SP en BP, por lo que la instrucción MOV CX, [BP + 8] usa el segmento de stack y copia T1 en CX, después se lleva a cabo el retardo para a continuación restaurar los valores de BP, banderas y CX. En este punto, SP apunta a la localidad 007CH, la dirección de regreso.

La siguiente instrucción (RET 2) saca esta palabra en IP (SP = 007EH) y suma dos al SP, lo cual causa que T1 sea removido del stack y SP tome su valor original (0080H).

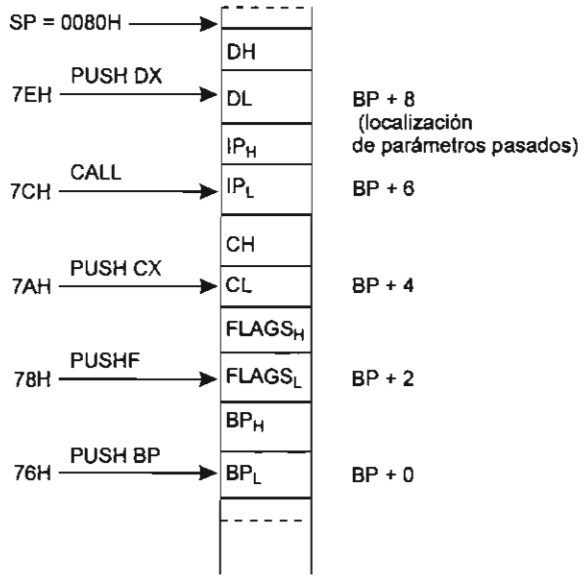


Figura I.18. Stack del programa 5.

Conclusión

Usar el stack para pasar parámetros es conveniente en ciertos casos (no se usan registros de la CPU ni localidades de memoria), pero requiere instrucciones adicionales de push y pop, lo cual aumenta el tiempo de respuesta de un procedimiento, por lo que si se desea pasar varios parámetros, es más eficiente usar un apuntador a memoria y pasarlo como parámetro.

Estructura de un programa en lenguaje ensamblador recomendada para los microprocesadores 8086/88

[Segmento(s) de datos]

[Segmento de stack]

[Segmento(s) de código con procedimientos]

[Segmento de código principal
[procedimientos]
[programa principal]]

De esta manera, todos los símbolos serán conocidos antes de que se ensamble el programa principal.

Tablas de datos

Cuando realizamos una búsqueda o un "barrido" de una tabla de datos secuenciales, podemos usar un apuntador inicializado en la base de la tabla, el cual se incrementará para accesar cada elemento de la tabla.

Ejemplo 6

En este ejemplo se manejará un *display* de siete segmentos sin utilizar un decodificador externo. En la figura I.19 se muestra el hardware, en donde cada segmento se maneja con un bit de un puerto de salida. El problema es mostrar en el display códigos de tal forma que parezca que está contando lentamente del 0 al 9 y de la A a la F.

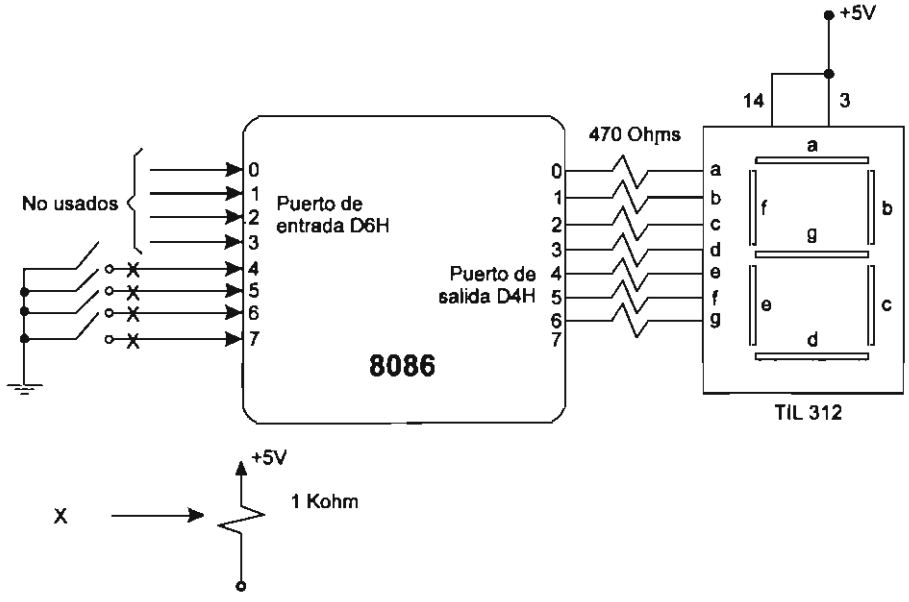


Figura I.19. Diagrama de bloques de la circuitería del ejemplo 6.

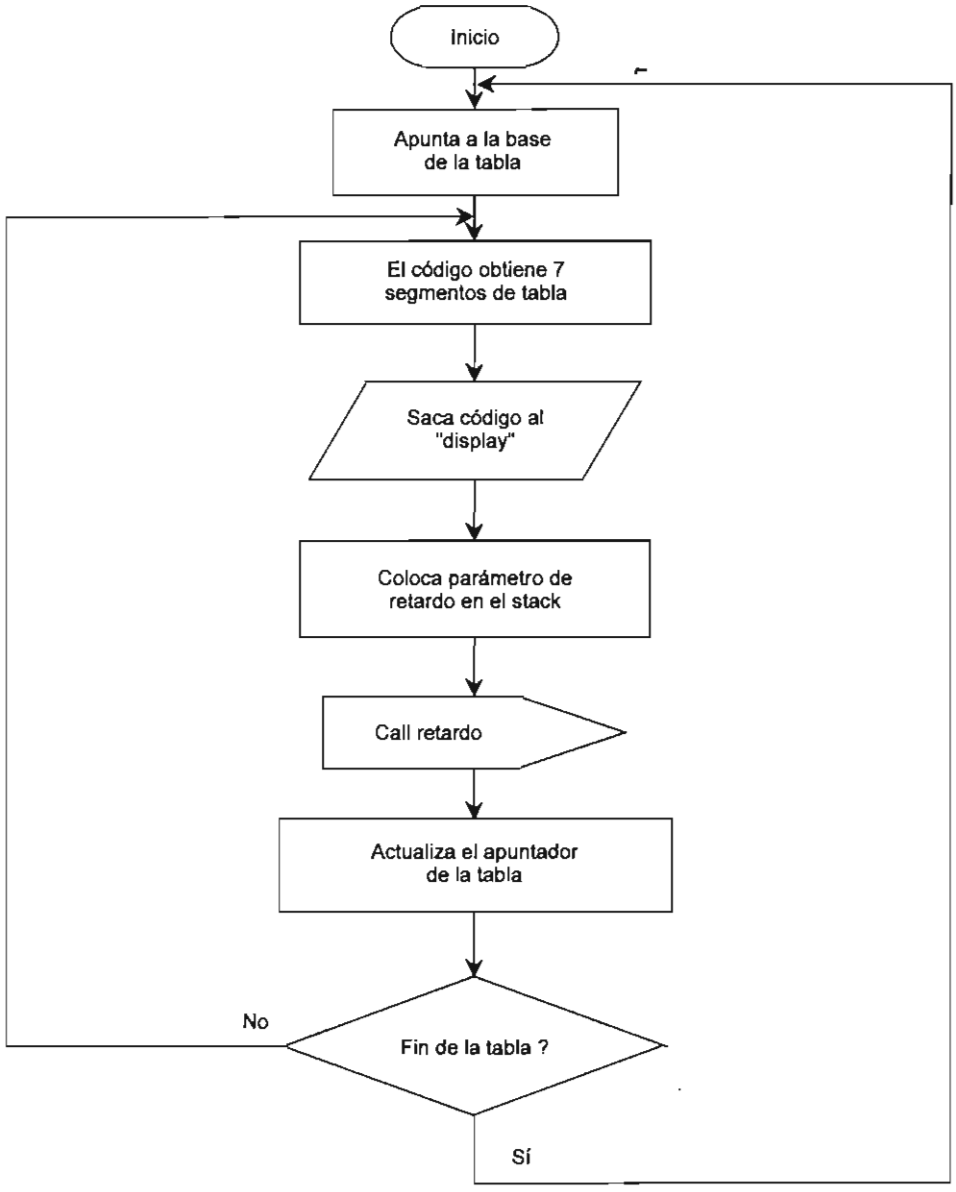


Figura I.20. Diagrama de flujo del ejemplo 6.

```
The Microsoft MACRO Assembler                                   15-09-92
PAGE 1-2
Contador siete segmentos

                        TITLE   Contador siete segmentos
                           Programa no. 6
;Este programa despliega los códigos hexadecimales de 0 a F
;en secuencia, en un display conectado al puerto 0D4H.
;Esto ilustra el uso de una tabla de datos para almacenar
;los códigos de cada uno de los dígitos.
0000            STACK           SEGMENT  WORD            STACK
0000 0A[                        DW       10              DUP (?)
        ????] TOP               LABEL          WORD
0014            STACK           ENDS

;Este segmento (de datos) se usa para almacenar los códigos
;de cada uno de los 16 dígitos.

0000                            SEVEN SEGMENT            WORD
0000            40 79 24 30 19 CODES  DB    40H, 79H, 24H, 30H, 19H
0005            12 02 78 00 18 08     DB    12H, 02H, 78H, 00H, 18H, 08H
000B            03 46 21 06 6E        DB    03, 46H, 21H, 06, 0EH
0010                            SEVEN ENDS

;Éste es un procedimiento de retardo que es llamado en el
;programa principal. Nótese que éste se encuentra en su
;propio segmento.

0000            PROCED          SEGMENT  BYTE            ;Mismo código que en el
                                                         ;ejemplo 5, excepto que
0000            DELAY           PROC     FAR             ;DELAY debe ser "far
                                ASSUME   CS: PROCED      ;procedure".
0000 51                         PUSH     CX              ;Salva todos los
0001 9C                         PUSHF                    ;registros y banderas.
0002 55                         PUSH     BP              ;
0003 8B EC                      MOV      BP, SP          ;Toma del stack el
                                                         ;valor de retardo.
0005 8B 4E 0A                   MOV      CX, [BP + 10]
0008 E2 FE      CUENTA:         LOOP     CUENTA          ;Loop de retardo.
000A DE                         POP      BP              ;Restaura todos los
000B 9D                         POPF                     ;registros.
000C 59                         POP      CX              ;Descarta el parámetro
                                                         ;de retardo.
000D CA 0002                    RET      2               ;FAR Return.
0010                            DELAY    ENDP
0010                            PROCED   ENDS
0000  CODE      SEGMENT         BYTE
                ASSUME          CS: CODE, DS: SEVEN, SS: STACK
;Aquí comienza el programa principal.
```

```
The Microsoft MACRO Assembler                                   15-09-92
PAGE 2-2
Contador de siete segmentos

= 00D4          PDATOS      EQU     0D4H            ;Puerto de salida.
= 0000          T1          EQU     0H              ;0.2 seg de retardo.

0000 B8----R    INICIO:     MOV     AX, STACK       ;Empieza cargando los
0003 8E D0                  MOV     SS, AX          ;registros SS, DS y SP.
0005 B8----R                MOV     AX, SEVEN       ;Apunta al tope del
0008 8E D8                  MOV     DS, AX          ;stack.
000A 36: 8D 26 0014 R       LEA     SP, TOP
000F 8D 36    0000 R        LEA     SI, CODES       ;Offset de la tabla.
                CICLO:
0013 FC                     CLD                     ;Autoincrementa SI.
0014 BB 0000                MOV     BX, T1          ;Obtiene el delay.
0017 B9 0010                MOV     CX, 16          ;16 dígitos.
001A AC         NUMDIS:     LODSB                   ;El byte a desplegar
001B E6 D4                  OUT     PDATOS, AL      ;en AL.
001D 53                     PUSH    BX              ;Pasa parámetro de
                                                    ;retardo.
001E 9A 0000---R            CALL    DELAY           ;Espera 0.2 segs.
0023 53                     PUSH    BX
0024 9A 0000---R            CALL    DELAY           ;Otros 0.2 segs.
0029 E2 EF                  LOOP    NUMDIG          ;Siguiente dígito.
002B EB E2                  JMP     CICLO           ;Repite forever.
002B EB E2                  CODE    ENDS            ;Fin de segmento.
CODE
002D                        END     INICIO          ;Fin del programa.
```

$$\text{DELAY} = 3.4\ \mu\text{seg} * \text{BX} + 1.7\ \mu\text{seg}$$

o

T1

Si T1 = 0FFFF H

$\Rightarrow$ Retardo$\approx$ 0.2 seg

En el programa anterior se usan cuatro segmentos etiquetados como: STACK, SEVEN, PROCED y CODE.

El segmento STACK consiste en 10 palabras no inicializadas que se usan para almacenar la dirección de regreso del procedimiento DELAY y algunos registros de la CPU.

El segmento llamado SEVEN tiene los códigos de siete segmentos de los 16 dígitos hexadecimales.

El segmento PROCED contiene el procedimiento de retardo (DELAY). Ya que este procedimiento está en un segmento diferente al del programa principal, debe ser declarado como far procedure, lo cual implica que el parámetro de retardo (pasado por el programa principal) se localiza 10 bytes relativo a la base del stack (en lugar de 8).

El segmento CODE contiene el programa principal, el cual comienza cargando SS y DS e inicializando SP al tope del stack.

La instrucción LEA SI, CODES carga en SI la dirección offset de la base de la tabla. A continuación se ejecuta un loop en el cual los códigos se buscan en la tabla (con la instrucción LODSB) y se sacan en el display. El procedimiento DELAY se invoca dos veces para producir un retardo de 0.4 segs. Finalmente, el LOOP NUMDIG prueba si ya han sido desplegados los 16 dígitos.

El mapa de ligas (que resulta al ligar los segmentos) se muestra a continuación. Se puede ver que el segmento PROCED sigue inmediatamente después del segmento CODE ya que éstos son alineables por byte. Los segmentos SEVEN y STACK son alineables por palabra y comienzan en el límite de una palabra.

```
START          STOP           LENGTH         NAME
00000H         0002CH         002DH          CODE
0002DH         0003CH         0010H          PROCED
0003EH         0004DH         0010H          SEVEN
0004EH         00061H         0014H          STACK
PROGRAM ENTRY POINT AT 0000:0000
```

Ejemplo 7

Búsqueda de elementos en una tabla de datos no secuenciales

Generalmente la búsqueda en este tipo de tablas depende de un índice. Una de las instrucciones más útiles en estos casos es XLAT, la cual reemplaza a AL con el contenido de la localidad apuntada por BX + AL.

En la figura I.21 se muestra esta técnica, en donde se lee un número binario de 4 bits; a continuación busque el código de 7 segmentos correspondiente a este número y saque el resultado a un display.

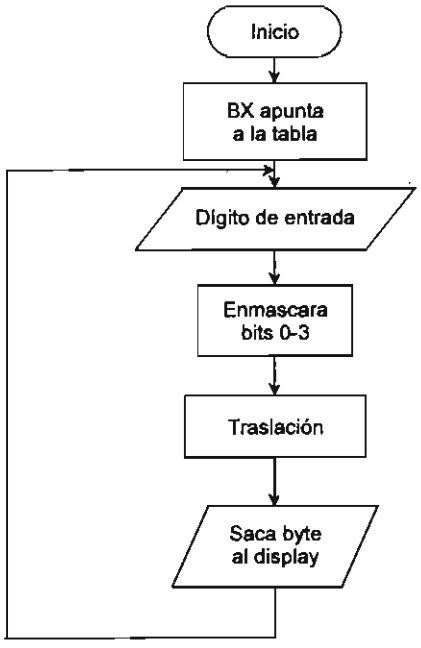


Figura I.21. Diagrama de flujo del ejemplo 7.

```
The Microsoft MACRO Assembler                                   15-09-92
PAGE 1-1
Búsqueda de tablas
                        TITLE           Búsqueda de tablas
                             Programa núm. 7
;Este programa lee los números binarios de 4 bits por el
;puerto 0D6H (4-7). Busca el código de 7 segmentos y lo
;saca a un display conectado al puerto 0D6H.

;Códigos para los dígitos 0-F

0000    SEVEN                   SEGMENT     WORD
0000    40 79 24 30 19 12       CODES DB    40H, 79H, 24H, 30H, 19H, 12H
0006    02 78 00 18 03                DB    02H, 78H, 00H, 18H, 08H, 03H
000C    46 21 06 0E                   DB    46H, 21H, 06, 0EH
        SEVEN           ENDS


0000    CODE SEGMENT BYTE
        ASSUME      CS: CODE, DS: SEVEN
;Aquí comienza el programa principal.

= 00D6                  PENT            EQU         0D6H
= 00D4                  PSAL            EQU         0D4H

0000    B8---R          INICIO: MOV     AX, SEVEN   ;Carga DS con la direcc.
0003    8E D8                   MOV     DS, AX      ;del segmento.
0005    8D 1E 0000 R            LEA     BX, CODES   ;Offset de la tabla.
0009    B1 04                   MOV     CL, 4       ;Necesita 4 rotaciones.
000B    E5 D6           OTRO:   IN      AL, PENT    ;Obtiene dígito.
000D    24 F0                   AND     AL, 0F0H    ;Enmascara bits.
000F    D2 C0                   ROL     AL, CL      ;Prepara para XTAL.
0011    D7                      XTAL                ;Busca código.
0012    E6 D4                   OUT     PSAL, AL    ;Despliega dígito.
0014    EB F5                   JMP     OTRO        ;Repite.
0016                    CODE    ENDS                ;Fin de segmento CODE.
                                END     INICIO      ;Fin del programa.
```

El programa comienza de manera similar al del ejemplo anterior y también utiliza un segmento de datos para almacenar los códigos de 7 segmentos (SEVEN). El programa también se almacena en el segmento llamado CODE. El registro BX apunta a la base de la tabla. El índice de la tabla es leído por 4 switches conectados a un puerto de entrada.

Para asegurar que los bits 0-6 sean cero, se usa la instrucción AND AL, 0F0H, a continuación se hacen cuatro rotaciones para poder utilizar la instrucción de traslación y poder sacar el dato al display.

NOTA: la instrucción XLAT sólo puede trabajar con tablas de máximo 256 bytes. Sin embargo, la instrucción MOV AX, [BX + SI] accesará un dato indexado por SI en una tabla con la dirección de la base en BX. Ya que SI es de 16 bits, la longitud de la tabla puede ser de hasta 65 536 bytes.

Ejemplo 8

Programación modular

Si reescribimos el programa del ejemplo 5 (titulado BEEP) podemos tener tres módulos del programa separados:

Módulo 1. Programa principal; inicialización del stack y llamada a BEEP.
Módulo 2. Procedimiento BEEP; produce un BEEP.
Módulo 3. Procedimiento DELAY; proporciona un retardo.

```
The Microsoft MACRO Assembler                                   15-09-92
PAGE 1-1
Programa principal
                                TITLE     PROGRAMA PRINCIPAL
;Este programa produce un BEEP en una bocina conectada al
;bit 0 del puerto 0D4H.
0000      STACK                 SEGMENT   WORD        STACK
0000 02   [                     DW 2      DUP (?)     ;Dos palabras
           ????]
0004      TOP                   LABEL     WORD
0004      STACK                 ENDS
          EXTRN                 BEEP: FAR             ;BEEP es un símbolo
                                                      ;localizado en un far
                                                      ;external segment.
0000      CODE                  SEGMENT
                                ASSUME    CS: CODE, SS: STACK
0000      INICIO:               MOV       AX, STACK   ;Carga SS con la
                                                      direcc.
0000  B8---R                   MOV       SS, AX      ;del segmento de
                                                      stack.
0003  8E D0                    LEA       SP, TOP     ;SP apunta a TOP.
0005  36: 8D 26 0004 R         CALL      BEEP        ;Éste es un far call.
000F  F4                       HLT
0010  CODE                     ENDS                  ;Fin de segmento CODE.
                                END       INICIO      ;Fin del programa.
```

Podemos ver en el listado del programa principal que el segmento de stack se alinea por palabra y que sólo se inicializa con dos palabras.

Antes de que el programa principal comience, se declara el símbolo BEEP como símbolo externo *(external far symbol)*, lo que evita que cuando se haga referencia a este símbolo, que se definió en otro módulo, el ensamblador dé un mensaje de símbolo indefinido *(undefined symbol)*. El tipo de atributo (far o near) se necesita para que el ensamblador sepa cuántos bytes debe reservar para su dirección. En este caso BEEP se localiza en un far segment, por lo que se reservan 4 bytes.

La directiva EXTRN también se usa para identificar un dato localizado en otros módulos. El tipo del dato puede ser BYTE, WORD o DWORD.

Ejemplo:

Para referenciar al byte llamado CUENTA y que se encuentra en otro módulo, se puede usar la siguiente declaración:

```
EXTRN     CUENTA: BYTE
```

El programa principal se localiza en un segmento llamado CODE, el cual comienza cargando SS y SP, para a continuación llamar a BEEP. Después el programa se detiene. En la práctica, HLT debe reemplazarse con un salto o regreso al sistema operativo. Debe notarse en el código objeto generado por el ensamblador que en el llamado a BEEP (CALL BEEP) se dejaron 2 bytes con 00 para el offset y 2 bytes indefinidos para el segmento, ya que la dirección de BEEP es desconocida. La E indica que esta dirección se localiza en un módulo externo y será resuelta por el ligador cuando todos los módulos del programa se liguen. A continuación se mostrará el procedimiento BEEP, en donde la declaración:

```
-STACK SEGMENT WORD STACK-
```

define un segmento llamado STACK, alineado por palabra y combinable por stack ("stack combinable"). La palabra STACK al final de la declaración significa que este segmento puede combinarse con el segmento de stack definido en el programa principal. En este ejemplo, puesto que se definen dos palabras para el stack, esto tendrá el efecto o causará que se adicionen dos palabras a la longitud del stack. Estas dos palabras se adicionan antes de la etiqueta TOP en el siguiente procedimiento, de manera que este símbolo sigue identificando al tope del stack.

```
The Microsoft MACRO Assembler                                  15-09-92
PAGE 1-1
Beep (far procedure)

            TITLE       BEEP        (FAR PROCEDURE)
            ;Función:   Produce un BEEP corto (de 0.25s).
            ;Entradas:  Ninguna.
            ;Salidas:   Ninguna.
            ;Llamadas:  DELAY (near procedure)
            ;Destruye:  CX, DX, AL, banderas.

0000        STACK       SEGMENT     WORD STACK
0000  02    [           DW          2     DUP (?)     ;Adiciona 2 a la
      ????]                                           ;long. del stack.
```

```
0004        STACK       ENDS
0000        SUBRT       SEGMENT     BYTE  PUBLIC      ;El código en este
                                                      ;segmento es público y
                                                      ;combinable por byte.

;Las siguientes 2 líneas declaran a BEEP como un símbolo
;público (accesible a otros módulos externos) y a DELAY un
;símbolo near(en el mismo segmento pero en un módulo ext.).
                PUBLIC          BEEP
                EXTRN                DELAY:NEAR
                ASSUME          CS: SUBRT
= 008E   T1 EQU   142                          ;500 µs delay.
= 00D4   PSAL     EQU   0D4H                        ;Puerto de salida.
= 01F4   CICLOS   EQU    500                    ;Número de ciclos. .

0000        BEEP        PROC        FAR
0000  B9 01F4              MOV       CX CICLOS        ;Contador de ciclos.
0003  F6 D0       OTRO:             NOT   AL         ;Complementa el bit de
0005  E6 D4                         OUT PSAL, AL          ;salida.
0007  BA 008E                       MOV   DX, T1     ;Coloca el retardo en
000A  52                            PUSH  DX         ;el stack.
000B  E8 0000 E                     CALL  DELAY      ;Éste es un near call.
000E  E2 F3                         LOOP  OTRO       ;Hazlo CX veces.
0010  CB                            RET              ;Far RET
0011        BEEP                    ENDP
0011        SUBRT                   ENDS
                                    END
```

El procedimiento anterior se localiza en un segmento llamado SUBRT. Este procedimiento es alineado por byte y el tipo de combinación es pública *(public)*. La declaración pública indica que este procedimiento puede ser referenciado por otros módulos del programa.

En general, el tipo de combinación de un segmento puede ser público, stack o privado. Por default es privado *(private)*, lo que significa que el código del segmento no se puede combinar (o accesar) por otro módulo del programa.

La declaración PUBLIC BEEP permite que el símbolo BEEP se pueda referenciar en otros módulos, por ejemplo en la declaración EXTRN BEEP:FAR del programa principal. Debe tenerse cuidado de que cada símbolo declarado como externo sea declarado como público en algún otro módulo del programa.

El procedimiento BEEP llama a la rutina DELAY localizada en otro módulo del programa, por lo que DELAY se declara externa. En este caso DELAY se localiza en el mismo segmento (SUBRT) que BEEP, por lo que tiene el atributo NEAR.

```
EXTRN       DELAY:NEAR
```

Nótese que en código objeto, para la instrucción CALL DELAY se reservan dos bytes para el offset de DELAY.

```
The Microsoft MACRO Assembler                                    15-09-92
PAGE 1-1
Delay (near procedure)
       TITLE                    DELAY               (NEAR PROCEDURE)

;Función:  Retardo de tiempo.
;Entradas: Valor del retardo de 16 bits pasado en el stack.

                          T = [3.4 µs * T1] + 17 µs.

;Salidas:  Ninguna.
;Llamadas: Ninguna.
;Destruye: Ninguna.

0000      STACK         SEGMENT     WORD  STACK
0000  03 [              DW          3     DUP (?)     ;Adiciona 3 a la
      ????]                                           ;long. del stack.
0006      STACK               ENDS

0000      SUBRT         SEGMENT           BYTE PUBLIC
                        PUBLIC            DELAY
                        ASSUME            CS:SUBRT

0000      DELAY         PROC  NEAR
0000 51                 PUSH  CX                      ;Salva todos los
0001 9C                 PUSHF                         ;registros y banderas.
0002 55                 PUSH  BP
0003 8B 4E 08           MOV   BP, SP                  ;Toma del stack el valor
0005 8B 4E 08           MOV   CX, [BP + 8]            ;de retardo.
0008 E2 FE CUENTA:      LOOP  CUENTA                  ;Loop de retardo.
000A 5D                 POP   BP                      ;Restaura todos los
000B 9D                 POPF                          ;registros.
000C 59                 POP   CX
000D C2 0002            RET 2                         ;Descarta el parámetro
0010    DELAY           ENDP                          ;de retardo.
0010    SUBRT           ENDS
                        END
```

En el listado del procedimiento DELAY se puede notar que se adicionan tres palabras más a la longitud del stack.

Ya que DELAY se declaró como NEAR en el procedimiento BEEP, DELAY debe localizarse en el segmento SUBRT, el cual es público y escrito como near. El símbolo DELAY también debe declararse público.

Una vez que los tres módulos anteriores fueron escritos y ensamblados, pueden ligarse de la siguiente forma:

```
LINK  MAIN.OBJ + BEEP.OBJ + DELAY.OBJ
```

que dará como resultado el archivo ejecutable MAIN.EXE. Si examinamos el mapa de ligas creado para los seis módulos del programa (MAIN.MAP) obtenemos el siguiente resultado:

START	STOP	LENGTH	NAME
00000H	0000FH	0010H	CODE
00010H	0001DH	000EH	STACK
0001EH	0003EH	0021H	SUBRT

Usando el comando DEBUG para cargar MAIN.EXE y luego desensamblar el código objeto, obtenemos el resultado mostrado a continuación.

```
09E3: 0000      B8E409          MOV         AL, 09E4
09E3: 0003      8EDO            MOV         SS, AX
09E3: 0005      36              SEG         SS
09E3: 0006      8D260E00        LEA         SP, [000E]
09E3: 000A      9A0E00E409      CALL        09E4: 000E
09E3: 000F      F4              HLT
09E3: 0010      0000            ADD         [BX + SI], AL
09E3: 0012      0000            ADD         [BX + SI], AL
09E3: 0014      0000            ADD         [BX + SI], AL
09E3: 0016      0000            ADD         [BX + SI], AL
09E3: 0018      0000            ADD         [BX + SI], AL
09E3: 001A      0000            ADD         [BX + SI], AL
09E3: 001C      0000            ADD         [BX + SI], AL
09E3: 001E      B9F401*         MOV         CX, 01F4
09E3: 0021      F6D0            NOT         AL
09E3: 0023      E6D4            0UTB        D4
09E3: 0025      BA8E00          MOV         DX, 008E
09E3: 0028      52              PUSH        DX
09E3: 0029      E80300          CALL        002F
09E3: 002C      E2F3            LOOP        0021
09E3: 002E      CB              RET         L
09E3: 002F      51              PUSH        CX
09E3: 0030      9C              PUSHF
09E3: 0031      55              PUSH        BP
09E3: 0032      8BEC            MOV         BP, SP
09E3: 0034      8B4E08          MOV         CX, [BP + DI]
09E3: 0037      E2FE            LOOP        0037
09E3: 0039      5D              POP         BP
09E3: 003A      9D              POPF
09E3: 003B      59              POP         CX
09E3: 003C      C20200          RET         0002
```

```
* →    09E30H
       001EH
      -------
       09E4EH          segmento = 09E4: 000E
```

Podemos ver que el programa principal ocupa de las localidades 00000 a 0000FH. A continuación empieza el stack segment en la dirección 00010H, el cual tiene 14 bytes de longitud, 2 de ellos para el programa principal, 2 para el procedimiento BEEP y 6 para el procedimiento DELAY. Inmediatamente después sigue el segmento SUBRT; ya que éste se definió a ser alineado por byte, nótese que:

longitud de BEEP (11H bytes) + DELAY (10H bytes) = 21H bytes

El sistema operativo cargó al segmento CODE en la dirección 09E3:0000. El segmento STACK empieza en 09E4:0000 (09E3:0010) y termina en la 09E4:000D (09E30:001D). El segmento SUBRT comienza en la localidad 09E3:001E (09E4:000E), que es la dirección vista en la instrucción CALL BEEP. El procedimiento BEEP termina con RET L *(far return)* para a continuación seguir el procedimiento DELAY, el cual se alinea por byte.

Librerías

LIB es otro programa de utilería (como el MASM y el LINK), el cual permite crear una librería de códigos objeto (.OBJ).

Una vez ensamblados los módulos, pueden adicionarse a la librería:

```
> LIB SET.LIB >... + BEEP.OBJ + DELAY.OBJ
```

lo cual suma los módulos objeto de BEEP y DELAY al archivo de librerías SET.LIB.

La ventaja de usar librerías es que los módulos individuales no necesitan mencionarse al ligar y crear el programa ejecutable. Por ejemplo, después de que BEEP y DELAY se han adicionado a la librería, el comando:

```
LINK MAIN.OBJ, MAIN.EXE/M, MAIN.MAP, SET.LIB
```

liga el archivo MAIN.OBJ, crea el archivo ejecutable MAIN.EXE y el mapa de ligas MAIN.MAP (con el switch /M), y busca la librería SET.LIB para símbolos externos.

El uso de LIB facilita que los módulos más utilizados sean adicionados a otros programas sin tener que recordar los nombres de los módulos individuales.

Macros

Los macros permiten crear nuevas instrucciones que serán reconocidas por el ensamblador. Por ejemplo, podemos escribir una instrucción (macro) llamada DELAY T1 para generar un retardo igual al valor de T1 en segundos. La ventaja de usar macros es que el código resultante de un programa es más fácil de leer y escribir, por lo que las LIBRERÍAS de macros se pueden escribir e incluirse en un código fuente.

La forma general de un macro es:

```
nombre   MACRO          arg 1      arg 2      arg 3...
         instrucciones...
          ---------
          ---------
          ---------
           ENDM
```

donde:

arg1 arg2 arg3... son los argumentos del macro.

Los argumentos son opcionales y permiten que el mismo macro se use en distintos lugares del programa y con diferentes conjuntos de datos. Cada argumento debe representar una constante (no se pueden especificar registros de la CPU).

Ejemplo 9

Supóngase que el programa requiere que el contenido del registro AX sea invertido y rotado hacia la derecha un número de veces dado. Con un macro llamado ROTA haga esta función:

Solución:

```
ROTA      MACRO     VECES
          NOT       AX
          MOV       CL, VECES
          ROR       AX, CL
          ENDM
```

Al adicionar este macro al programa, el ensamblador reconoce una nueva instrucción: ROTA n.

En el siguiente programa se muestra que la definición del macro se hace en el inicio del mismo, tal que al involucrarlo en el programa principal el macro ya es conocido.

Nótese que el ensamblador expande el macro con las instrucciones que él representa. El símbolo "+" indica que las declaraciones son parte de la definición del macro. El argumento del macro puede definirse literalmente (2) o con un "EQU (N1)".

Archivos include

No es necesario escribir un macro al inicio del programa que lo use. En lugar de eso, se puede crear un archivo especial —llamado MACRO.LIB– que contenga las definiciones de todos los macros y al inicio del programa colocar la declaración:

```
INCLUDE MACRO.LIB
```

lo que causará que el ensamblador incluya automáticamente todas las declaraciones del archivo MACRO.LIB. Además, la declaración INCLUDE se puede utilizar para incluir instrucciones y declaraciones de otros archivos que no necesariamente sean macros. Cuando se usa el INCLUDE, el ensamblador

expande los macros requeridos en el primer paso. Durante el segundo paso, todos los macros en el INCLUDE se listan, lo cual puede ser indeseable cuando el archivo INCLUDE es muy grande y muchos de los macros no se usan.

Con las siguientes declaraciones se puede incluir el archivo de LIBRERÍAS de macros, pero suprime el listado de cada macro:

```
IF1
INCLUDE   MACRO.LIB
ENDIF
```

El IF y el ENDIF son directivos condicionales del ensamblador. En este ejemplo, se le dice al ensamblador que incluya el archivo MACRO.LIB durante el primer paso (cuando los macros son expandidos) y no en el segundo (cuando los macros son listados). Por supuesto, el cuerpo de cada macro expandido sí se incluye en el listado del archivo (identificado con el símbolo "+"), de tal forma que pueda verse el código generado para cada macro invocado.

Nombres locales

Existe un problema cuando una dirección simbólica se usa en un macro. Supóngase el siguiente ejemplo: cuando el ensamblador expande un macro, no hay problema la primera vez; si este macro se usa una segunda vez en el mismo programa, el ensamblador envía el siguiente error: "Redefinition of symbol" o "Symbol is multidefined" (símbolo redefinido). El problema es la etiqueta CUENTA, lo cual se puede resolver si al inicio del macro todas las etiquetas usadas en él se declaran de tipo LOCAL.

Ejemplo:

```
DELAY     MACRO     DELTA
          LOCAL     CUENTA
```

lo cual define a CUENTA como una dirección local. Al ensamblar al programa el ensamblador reemplaza la etiqueta CUENTA con el nombre ??0000 la primera vez que se invoca el macro DELAY. Si una segunda vez se reemplaza con ??0001, una tercera vez con ??0002 y así sucesivamente.

En el siguiente programa se muestra cómo se usa el macro DELAY en el contador de 7 segmentos. En este ejemplo el macro se define al inicio del programa en lugar de incluirlo en un *macro library file.*

```
DELAY          MACRO          DELTA
               PUSH           CX
               PUSHF
               MOV            CX, DELTA
COUNT          LOOP           COUNT
               POPF
               POP            CX
               ENDM
```

```
The Microsoft MACRO Assembler                                   16-09-92
PAGE 1-2
Contador siete segmentos

            TITLE       Contador siete segmentos

                                Programa núm. 9
;Este programa despliega los códigos hexadecimales de 0 a F
;en secuencia, en un display conectado al puerto 0D4H.
;Esto ilustra el uso de una tabla de datos para almacenar
;los códigos de cada uno de los dígitos.

          DELAY           MACRO           DELTA
                    PUSH        CX
                    PUSHF
                    MOV         CX, DELTA
          COUNT:    LOOP        COUNT
                    POPF
                    POP         CX
                    ENDM

0000      STACK             SEGMENT     WORD        STACK
0000 0A   [         DW              10          DUP     (?)
     ???? ]TOP      LABEL                   WORD
0014      STACK             ENDS

;Este segmento (de datos) se usa para almacenar los códigos
;de cada uno de los 16 dígitos.

0000      SEVEN             SEGMENT     WORD
0000      40 79 24 30 19  CODES  DB      40H, 79H, 24H, 30H, 19H
0005      12 02 78 00 18 08       DB      12H, 02H, 78H, 00H, 18H, 08H
000B      03 46 21 06 0E          DB      03, 46H, 21H, 06, 0EH
              SEVEN     ENDS
; Aquí empieza el programa principal.
```

```
The Microsoft MACRO Assembler                                   16-09-92
PAGE 2-2
Contador siete segmentos
0000    CODE            SEGMENT  BYTE
                        ASSUME   CS: CODE, DS: SEVEN, SS: STACK

;Aquí comienza el programa principal.

= 00D4                  PDATOS   EQU      0D4H        ;Puerto de salida.
= 0000                  T1       EQU      0H          ;0.2s de retardo.

0000    B8-R            INICIO:  MOV      AX, STACK   ;Empieza cargando los
0003    8E D0                    MOV      SS, AX      ;registros SS, DS y SP.
0005    B8---R                   MOV      AX, SEVEN   ;Apunta al tope del

0008    8E D8                    MOV      DS, AX      stack.
000A    36: 8D 26 0014 R         LEA      SP, TOP

000F    8D: 36 0000 R CICLO:     LEA      SI, CODES   ;Offset de la tabla.
0013    FC                       CLD                  ;Autoincrementa SI.
0014    BB 0000                  MOV      BX, T1      ;Obtiene el delay.
0017    B9 0010                  MOV      CX, 16      ;16 dígitos.
001A    AC              NUMDIG:  LODSB                ;El byte a desplegar

001B    E6 D4                    OUT      PDATOS, AL ;en AL.
                                 DELAY    T1
001D    51 +                     PUSH     CX
001E    9C +                     PUSHF
001F    B9 0000 +                MOV      CX, DELTA
0022    E2 FE +         COUNT:   LOOP     COUNT
0024    9D +                     POPF
0025    59 +                     POP      CX
                                 DELAY    T1
0026    51 +                     PUSH     CX
0027    9C +                     PUSHF
0028    B9 0000 +                MOV      CX, DELTA
002B    E2 FE +         COUNT:   LOOP     COUNT
002D    9D +                     POPF
002E    59 +                     POP      CX
002F    E2 E9                    LOOP     NUMDIG      ;Siguiente dígito.

0031    EB DC                    JMP      CICLO       ;Repite forever.
0033                    CODE     ENDS                 ;Fin de segmento CODE.
                                 END      INICIO      ;Fin del programa.
```

Ejemplo 10

Ya que el ensamblador 8086 (el MACRO-86) no reconoce las instrucciones para el NDP 8087, se puede usar un macro para crear la nueva instrucción FDDECSTP del 8086 (decrementa el stack pointer del 8087), la cual tiene el siguiente código: D9F6.

Solución:

```
FDECSTP     MACRO
            DB          0D9H, 0F6H
            ENDM
```

El macro sustituye los bytes D9F6H cada vez que FDECSTP es encontrado. Esto también pudo haberse hecho con un procedimiento, pero necesita más bytes (la instrucción CALL requiere al menos tres bytes).

Conclusiones

Un macro y un procedimiento son muy similares, pero un procedimiento se invoca con una instrucción CALL y termina con una instrucción RET, y el código del procedimiento aparece una sola vez en el programa, sin importar cuántas veces es llamado.

Por otro lado, un macro únicamente se invoca cuando se está ensamblando el programa y no cuando se ejecuta el mismo. Cada vez que se requiere el macro, el ensamblador sustituye la secuencia definida de instrucciones, lo cual requiere más memoria que el procedimiento equivalente.

La ventaja de usar un macro es que no existe el *overhead* asociado con las instrucciones CALL y RET del procedimiento, lo cual implica que el macro se ejecutará más rápido y en algunos casos requiere menos código que el procedimiento equivalente.

EL MÓDULO DE LA CPU DEL 8086/88

Diseño de sistemas mínimos con 8086

La secuencia de operaciones de un microprocesador es:

1. Busca la siguiente instrucción en secuencia de la memoria.
2. Ejecuta la instrucción.
3. Regresa al punto 1.

Existen cinco operaciones únicas o ciclos de bus posibles:

1. Lectura a memoria *(memory read)*.
2. Escritura a memoria *(memory write)*.
3. Lectura a un puerto de entrada/salida (I/O read).
4. Escritura a un puerto de entrada/salida (I/O write).
5. *Bus idle* (operación interna que no requiere acceso a memoria o I/O).

El 8086 tiene tres buses dedicados a transferir datos entre la CPU y la memoria y unidades de entrada/salida: bus de direcciones, datos y control.

En la figura I.22 se muestran los ciclos de bus para los cuatro tipos de ciclos de bus activos:

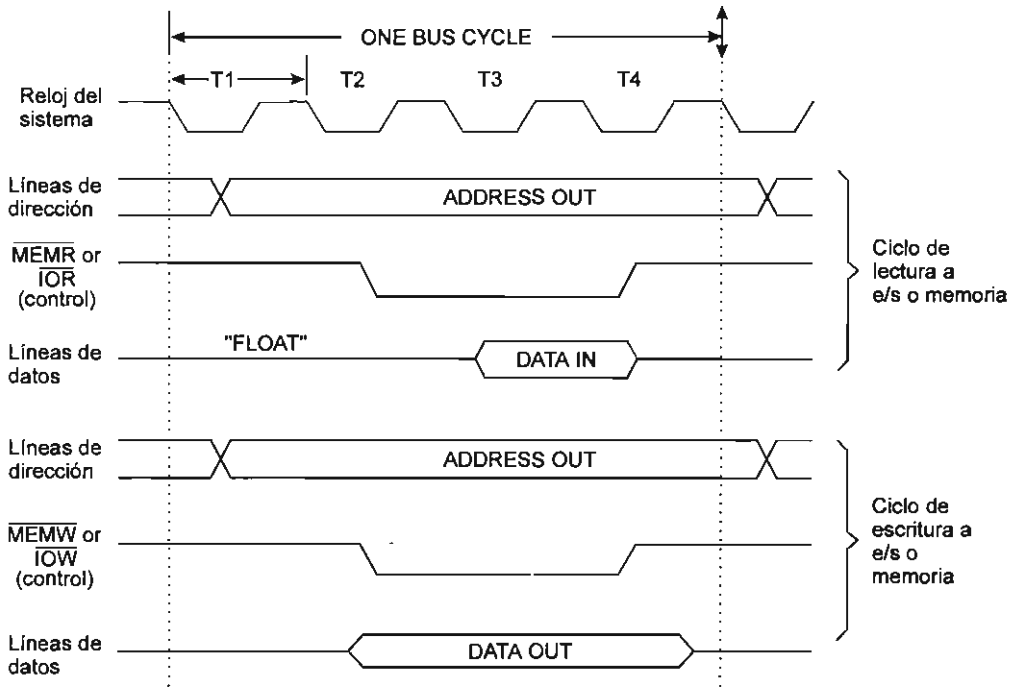


Figura I.22. Diagrama de tiempo de un ciclo de máquina del 8086/88.

Cada ciclo de bus comienza sacando la dirección de memoria o puerto de entra/salida durante el ciclo T1. Para el 8086 ésta puede ser una dirección de memoria de 20 bits, una dirección indirecta de 16 bits de un puerto de entrada/salida (usando DX) o una dirección directa de 8 bits de un puerto.

Las líneas paralelas de la figura I.22 indican que algunas líneas del bus serán "1" lógico y otras "0" lógico.

La memoria debe ser capaz de suministrar el dato seleccionado antes de que la señal $\overline{\text{MEMR}}$ pase a "1" lógico durante T4.

Ejemplo:

Describa el contenido de las líneas del bus de direcciones, datos y control cuando se ejecuta la instrucción MOV [1000H], BX. Supóngase que DS = 09D3H y BX = 123H.

Solución:

La dirección física de la localidad donde se escribirá es la 09D30H + 1000H = 0AD30H. La señal de control $\overline{\text{MEMW}}$ pasa a "0" lógico durante T2 y el bus de datos contiene 1234H (el contenido de BX). La memoria deberá "latchear" esta palabra antes de T4.

Diseño del hardware básico con un 8086

Descripción de terminales del 8086

El 8086 es un circuito integrado de 40 terminales, un bus de direcciones de 20 bits, un bus de datos de 16 bits, tres líneas de alimentación y tierra, y 17 líneas de señales de control y temporización.

Intel utiliza el "multiplexaje en tiempo" para algunas señales, en el cual cada línea del circuito tiene más de una función. Por ejemplo, las 16 líneas de datos (AD0-AD1) son líneas de dirección en el estado de reloj T1, y líneas de datos en los estados T2 → T4. Se requiere un circuito externo para "demultiplexar" o separar los datos y direcciones.

Las cuatro señales de un bus de control que se utilizan para determinar si el contenido del bus de direcciones es una dirección de memoria o de entrada/salida, son: $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$ y $\overline{\text{IOW}}$.

Examinando la figura I.22, veamos la secuencia de eventos que se lleva a cabo durante un ciclo de lectura a memoria:

- T1: el procesador saca la dirección de memoria de 20 bits y todas las líneas de control son deshabilitadas.
- T2: la línea de control $\overline{\text{MEMR}}$ y "0" lógico. La memoria debe reconocer este ciclo de lectura y debe prepararse para colocar el byte o word seleccionado en las líneas de datos.
- T3: el microprocesador configura sus líneas del bus de datos para entradas y da tiempo a la memoria para que ésta busque el dato (byte o word).
- T4: el microprocesador espera que el dato esté en las líneas del bus de datos. A continuación, "latchea" el contenido de esas líneas y libera la señal de control de lectura. Esto marca el final del ciclo de bus.

Lo más importante de lo anterior es que el microprocesador controla toda la temporización del bus.

Otra característica del 8086 es que puede trabajar en modo mínimo o en modo máximo. El modo mínimo se utiliza en sistemas con un solo procesador en una sola tarjeta de circuito impreso (PCB: *printed circuit board*). El modo máximo se usa para sistemas más complejos con tarjetas de memoria y de entrada/salida separadas. Este modo también soporta al NDP 8087 y al IOP 8089.

En la siguiente explicación de terminales supondremos que el 8086 se encuentra trabajando en modo mínimo:

Bus de datos (AD0 → AD1). Es el bus de datos bidireccional de la CPU. Estas líneas son válidas en los estados T2 → T4. Durante T1 son los 16 bits más bajos de una dirección de memoria o de I/O.

Bus de direcciones (AD0 → AD15 y A16/S3 → A19/S6). Forman el bus de direcciones de 20 bits y permiten accesar 1 048 576 localidades de memoria únicas; sólo son válidas durante el estado T1.

Address latch enable (ALE). Esta señal de salida se puede usar para demultiplexar las líneas de direcciones, datos y status en AD0 → AD15, A16/S3 → A19/S6 y $\overline{\text{BHE}}$/S7. En la figura I.24 se observa que para los ciclos de escritura y lectura (modo mínimo) cada uno comienza con el pulso ALE durante el estado T1. Los 20 bits de direcciones serán válidos cuando ALE pase de alto a bajo cerca del final de T1, por lo que puede ser usada para latchear la dirección.

Memory/$\overline{\text{IO}}$ (M/$\overline{\text{IO}}$). El 8086 no saca señales de lectura y escritura a memoria y puertos de entrada/salida (I/O) separados; en lugar de esto, saca la señal M/$\overline{\text{IO}}$ para identificar el ciclo de bus como una operación de memoria (Si M/$\overline{\text{IO}}$ = 1) o una operación de IO (Si M/$\overline{\text{IO}}$ = 0).

Read $\overline{\text{RD}}$. Señal activo bajo que indica que la dirección del flujo de datos en el bus es de la memoria o puerto de entrada/salida (I/O) hacia el procesador. Puede combinarse con M/$\overline{\text{IO}}$ para formar las señales $\overline{\text{MEMR}}$ e $\overline{\text{IOR}}$. La señal $\overline{\text{RD}}$ sale durante el estado T2 y desaparece en el estado T4. La memoria o

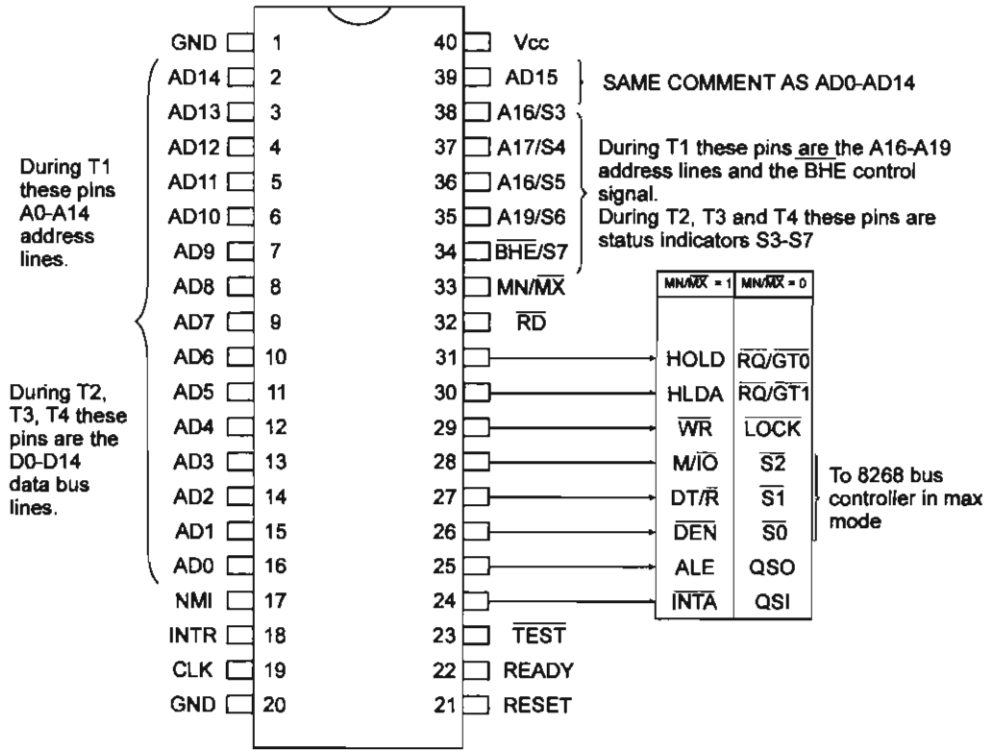


Figura I.23. La CPU 8086.

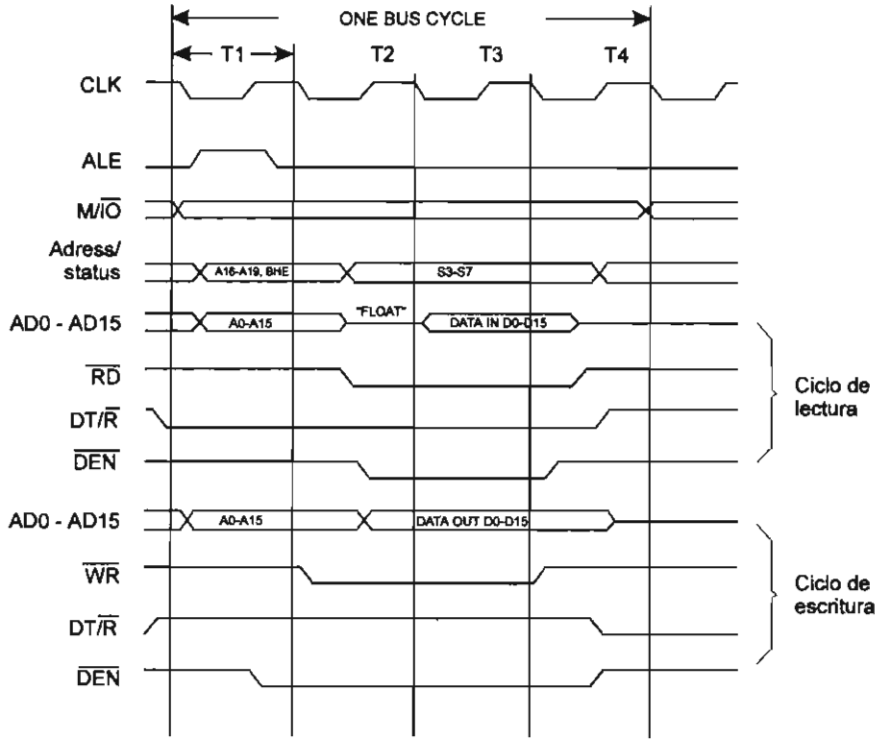


Figura I.24. Temporización de los ciclos de lectura y escritura en el 8086.

puerto de entrada/salida debe colocar el byte o word seleccionado en el bus de datos antes de que $\overline{RD}$ vuelva a "1" lógico.

Read $\overline{WR}$. Esta señal indica que el flujo de datos es de la CPU a la memoria o a un dispositivo de entrada/salida. El dato sale en el estado T2; esto da tiempo a la memoria o al dispositivo de entrada/salida de latchear el byte o word del dato antes de que WR se desactive durante T4.

En la figura I.25 se muestra cómo $\overline{RD}$, $\overline{WR}$ y M/$\overline{IO}$ se pueden combinar para generar un bus de control de cuatro líneas:

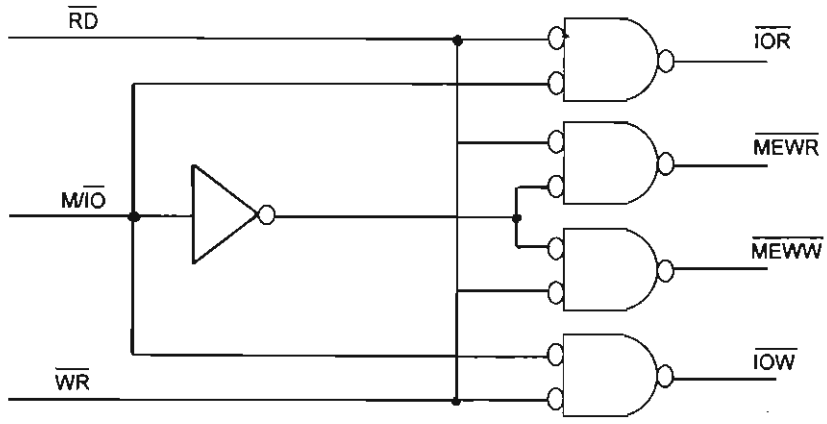


Figura I.25. Generación del bus de control en el 8086.

Clock (CLK). Todos los eventos del microprocesador se sincronizan al reloj del sistema aplicado en el terminal CLK. En el cuadro I.24 se indican las frecuencias máximas de reloj.

CUADRO I.24 *Especificaciones de la velocidad y potencia del 8086*

Procesador	$f_{máx}$ *(Mhz)*	I_{CC} *(máx)*	*Disipación (W)*
8086	5	340	1.7
8086-2	8	350	1.75
8086-1	10	360	1.8
8088	5	340	1.7
8088-2	8	350	1.75
P8088	5	250	1.25

Status (A16/S3 → A19/S6 y $\overline{BHE}$*/S7)*. Estas cinco señales de status se sacan en los estados T2 → T4, y se utilizan para propósitos de prueba y diagnóstico según se indica en el cuadro I.25:

CUADRO I.25. *Definición de los bits de estado (status) S3-S7*[a]

S4	*S3*	*Ciclo de acceso de bus*
0	0	Segmento extra
0	1	Segmento de stack
1	0	Segmento de código (o a ninguno)
1	1	Segmento de datos

[a] S5, IF (bandera de habilitación de interrupción).
S6, 0 (el 8086 está en el bus).
S7, bit de status spare (no usada).

Decodificando S3 y S4 tenemos cuatro espacios de direccionamiento separados de 1MB para los segmentos de stack, código, datos y extra. En la figura I.26 se muestra a bloques un circuito que intercepta las operaciones de lectura y escritura a memoria y accesa el bloque de memoria física apropiado. En la práctica esto se hace rara vez.

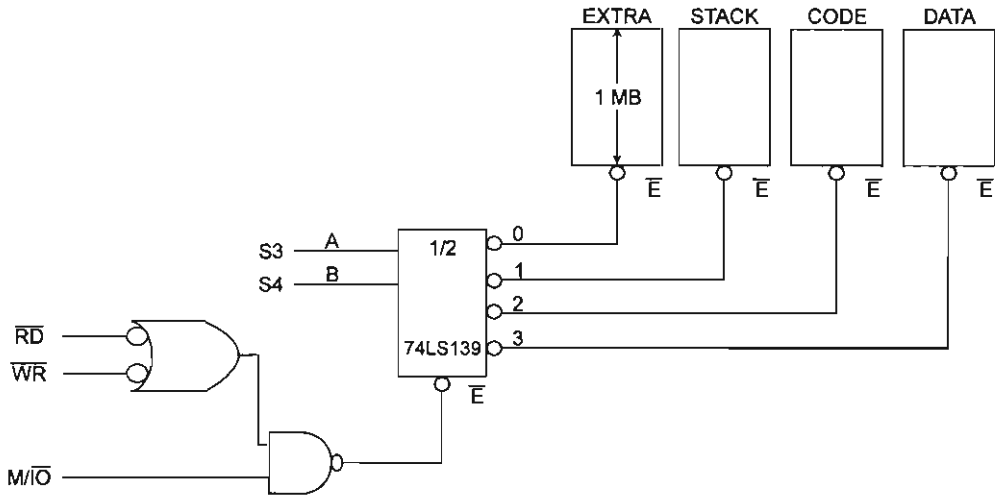


Figura I.26. Decodificación de las líneas S4 y S5.

Bus high enable ($\overline{BHE}$/*S7*). Esta señal es multiplexada con el indicador de status S7. Esta línea es salida sólo durante T1. Cuando $\overline{BHE}$ = "0" lógico, indica que las líneas AD8-AD15 están operando en la transferencia del dato. Esto puede ocurrir para accesos a palabras de memoria o de dispositivos de entrada/salida o cuando se está accesando un byte de una dirección impar. $\overline{BHE}$ y A0 se usan típicamente para seleccionar bancos de memoria o puertos de entrada/salida pares o impares.

CUADRO I.26. *Codificación del acceso a memoria del 8086*

BHE	*A0*	*Acción*
0	0	Palabra de acceso de 16 bits.
0	1	Byte impar de acceso de la D8-D15.
1	0	Byte par de acceso de la D0-D7.
1	1	No hay acción.

$\overline{BHE}$ = 1: desactivado.

Data transmit/receive (DT/ $\overline{R}$). Esta señal se usa para controlar la dirección del flujo de datos a través de *buffers* que estén conectados al bus de datos. Cuando está en "0" lógico indica una operación de lectura, en caso contrario indica escritura.

Data enable ($\overline{DEN}$). Esta señal se usa con DT/$\overline{R}$ y data bus, lo que previene la "disputa" del bus (dos circuitos intentando usar la misma línea del bus deshabilitando los buffers del bus de datos hasta el estado T2, cuando las líneas de direcciones/datos ya no tienen la dirección de memoria o puerto de entrada/salida.

Minimum/maximum mode (MN/ $\overline{MX}$). La función de los terminales 24 a 32 cambia dependiendo del nivel lógico-aplicado en este terminal. Más adelante veremos cómo opera el modo máximo (MN/$\overline{MX}$ = "1").

RESET. Un pulso activo alto causa que el 8086 termine la operación que está realizando y ejecute una secuencia de reset. El contexto actual se pierde. Se usa normalmente para inicializar al sistema.

TEST. Esta entrada se usa conjuntamente con la instrucción WAIT si en la entrada TEST hay un "1" lógico cuando se ejecute una instrucción WAIT.

Se suspende la ejecución del programa y la CPU entra en modo idle. Cuando TEST regresa a "0" lógico, la ejecución continúa (con la instrucción que sigue al WAIT). Normalmente esta entrada la maneja el 8087. WAIT se usa como un prefijo de las instrucciones que utilizan datos sobre los que opera el 8087, lo cual evita que la CPU accese un resultado en memoria antes de que el NDP termine.

READY. Esta entrada se muestrea con la subida del pulso de reloj T2. Si esta línea está en "0" lógico (not ready), el procesador inserta un estado T3 extra. Esto se repite hasta que READY pasa a "1" lógico. Usualmente esta entrada se maneja por una memoria lenta que no puede suministrar el dato tan rápido como lo solicita la CPU.

Interrupts (INTR, NMI e INTA). INTR y NMI son peticiones de interrupción iniciadas por hardware que funcionan exactamente como las interrupciones por software. NMI se dispara con la subida del pulso (transición "0" a "1" lógico) e INTR se dispara con un nivel activo alto. La entrada INTR se puede enmascarar limpiando el bit IF. NMI es una interrupción no mascarable que siempre será atendida; esta interrupción generalmente se utiliza para eventos catastróficos (errores de memoria o fallas de energía).

Cuando se activa NMI, el control del programa se transfiere a la dirección almacenada en las localidades 0008-0000BH. Cuando se activa INTR, se realiza un ciclo de reconocimiento de interrupción, el cual es similar al ciclo de lectura de memoria, excepto que se activa INTA en lugar de RD. El dispositivo que interrumpe debe colocar un "tipo" en las ocho líneas menos significativas del bus de datos. El control se transfiere a la dirección almacenada en las localidades tipo × 4 a la tipo × 4 + 3.

Hold y Hold acknowledge (HOLD y HLDA). HOLD es una entrada activo alto que causa que todas las líneas de los buses del procesador se comporten como circuito abierto, lo cual desconecta a la CPU de la memoria y dispositivos de entrada/salida, para permitir que otro procesador accese esos dispositivos (por ejemplo en un DMA: acceso directo a memoria, direct memory access). HLDA reconoce una petición de DMA al controlador de DMA.

Alimentación y tierra (Vcc y GND). El 8086 requiere una fuente de +5V y tiene dos terminales para tierra.

Generación del reloj del sistema y señal de reset para el 8086

El 8086 requiere una señal de reloj con tiempos de subida y de bajada rápidos (< 10 ns), niveles de "0" lógico de –0.5 a 0.6V y de "1" lógico de 3.9 a 5.0V, con un *duty cycle* de 33 por ciento.

La señal de RESET debe sincronizarse con el system clock y durar al menos cuatro estados T.

La señal de reloj:

1. Todas las actividades del 8086 son secuenciales y sincronizadas al system clock.
2. Durante el estado T1 de este reloj, la dirección de memoria o del puerto de entrada/salida sale; durante T2 las señales de control se activan; durante T6 la memoria o puerto de entrada/salida debe de responder, y, finalmente, en el estado T4 la CPU lee o escribe el dato.
3. La señal de reloj sirve para sincronizar los eventos anteriores, además de que se utiliza para "refrescar" las compuertas lógicas dinámicas con las cuales se diseña el procesador.

4. Por lo anterior, el reloj nunca debe de ser detenido o suministrado.
5. El 8086 tiene una frecuencia mínima de operación de 2 Mhz.

A continuación veremos cómo usar el circuito integrado 8284A para generar la señal de reloj del sistema para un 8086 y cómo poner en funcionamiento la función reset.

El generador de reloj y el driver para el procesador 8086

Descripción de terminales

$\overline{\text{ASYNC}}$ *(Ready syncronization select).* Entrada, activo bajo que define el modo de sincronización de la lógica del READY. Cuando $\overline{\text{ASYNC}}$ está en "0" lógico, se usan dos etapas de sincronización para READY. Cuando $\overline{\text{ASYNC}}$ está en "1" lógico o "al aire", se usa una sola etapa para la sincronización del READY.

READY (Ready). Salida, activo alto. Es una señal sincronizada con la entrada RDY.

X1, X2 (Crystal in). X1 y X2 son los terminales a los que se conectará el cristal. La frecuencia del cristal es tres veces la frecuencia del reloj para el procesador deseado.

F/ $\overline{C}$ *(Frecuencia/Crystal select).* Cuando está en "0" lógico esta entrada, permite que el reloj del procesador sea generado por el cristal. En caso contrario, CLK se genera de la señal presente en la entrada EFI.

EFI (External frecuency). Cuando F/$\overline{C}$ está en 1 lógico, CLK es generado de la frecuencia que aparece en esta entrada. La señal de entrada es una onda cuadrada con una frecuencia igual a tres veces la frecuencia de la salida CLK.

CLK (Processor clock). Es la salida de reloj usada por el procesador y otros dispositivos conectados directamente al bus local. CLK tiene una frecuencia de salida igual a 1/3 de la frecuencia del cristal o de la señal del terminal EFI. Tiene un duty cycle de 1/3 y una amplitud de 4.5V (Vcc = +5V).

PCLK (Peripheral clock). Señal de reloj de salida para periféricos. Tiene una frecuencia de la mitad de la frecuencia de CLK y un *duty cycle* de 50 por ciento.

OSC (Oscillator Output). Es la señal de salida de la circuitería interna del oscilador. La frecuencia de esta señal de salida es igual a la del cristal.

$\overline{\text{RES}}$*(Reset in).* Señal de entrada de activo bajo usada para generar el RESET. El 8284 tiene un *Schmitt trigger* en la entrada, tal que un circuito RC se puede usar para poner en marcha un reset al encender la fuente de alimentación.

RESET (Reset). Señal de salida activo alto usada como reset en la familia de procesadores 8086.

CSYNC (Clock synchronization). Señal de entrada activo alto usada para permitir que varios 8284A sean sincronizados para proporcionar señales de reloj que estén en fase. Cuando CSYNC está en "1" lógico los contadores internos se inicializan. Cuando CSYNC pasa a "0" lógico los contadores internos reasumen el conteo. CSYNC necesita sincronizarse externamente a EFI. Cuando se usa el oscilador interno CSYNC, debe alambrarse a tierra.

GND (Tierra)

Vcc (Alimentación). +5V.

$\overline{AEN}$1, $\overline{AEN}$2 *(Address enable).* Entradas activo alto. Sirven para calificar (o permitir el paso) su respectiva señal bus *Ready signal* (RDY1 *o* $\overline{\text{RDY}}$2). AEN1 valida $\overline{\text{RDY}}$1 y AEN2 valida RDY2. Existen dos señales $\overline{\text{AEN}}$ para permitir al procesador accesar dos sistemas multimaestro buses. En configuraciones no multimaster, la señal $\overline{\text{AEN}}$ debe colocarse a "0" lógico.

*RDY1, RDY2 (Bus ready).*Transferencia completa. Entradas activo alto, las cuales provienen de un dispositivo localizado en el sytem data bus y que indican que el dato ha sido recibido o está disponible.

intel

8284A/8284A-1
CLOCK GENERATOR AND DRIVER FOR iAPX 86, 88 PROCESSORS

- Generates the System Clock for the iAPX 86, 88 Processors: 5 MHz, 8 MHz with 8284A 10 MHz with 8284A-1
- Uses a Crystal or a TTL Signal for Frequency Source
- Provides Local READY and Multibus™ READY Synchronization
- 18-Pin Package
- Single +5V Power Supply
- Generates System Reset Output from Schmitt Trigger Input
- Capable of Clock Synchronization with Other 8284As
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

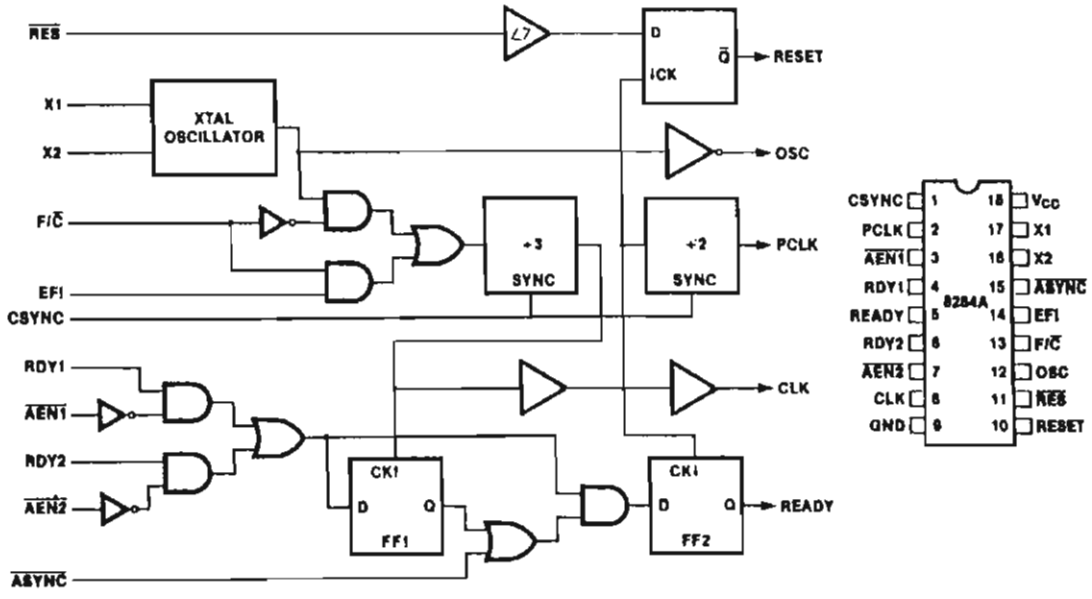


Figura I.27. Hoja de especificaciones electrónicas del 8284A.

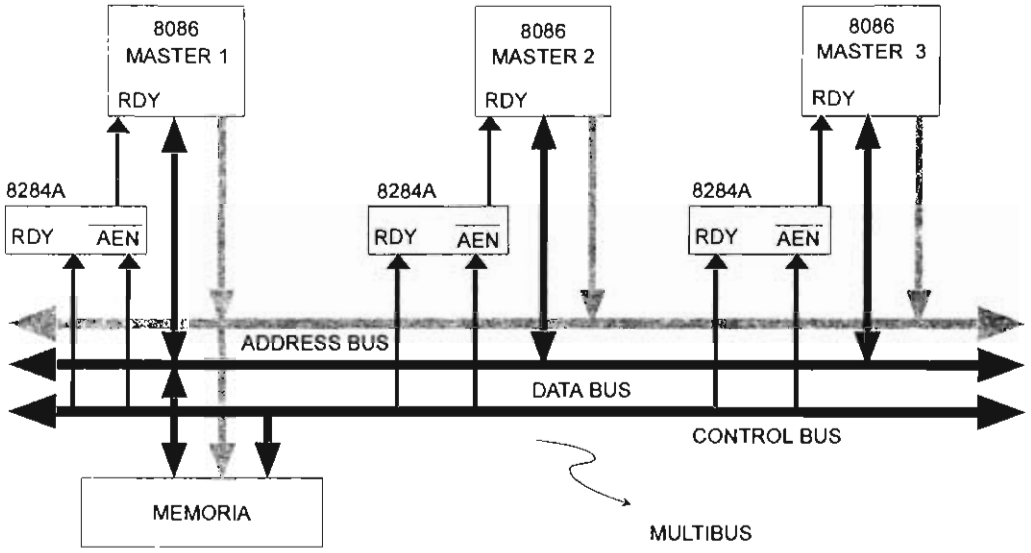


Figura I.28. Sistema multimaestro 8086.

Descripción funcional del 8284A

Clock generator. Consiste de un contador *(divide-by-three)* síncrono con una entrada de *clear* especial que inhibe el conteo. Esta entrada (SYNC) permite que el reloj de salida se sincronice con un evento externo (por ejemplo con el reloj de otro 8284A).

Ready synchronization. El 8284A tiene dos entradas de READY (RDY1, RDY2) para adaptarse a dos sistemas multimaestro; el terminal $\overline{\text{AEN}}$ debe estar en "0" lógico. Recordemos que esta línea valida a su respectiva señal de RDY ($\overline{\text{AEN}} \rightarrow$ RDY1; AEN2 $\rightarrow$ RDY2).

El 8284A requiere que se sincronice cualquier transición activa asíncrona de la entrada RDY para asegurar que la señal RDY que se suministra al procesador tenga la duración que especifica el fabricante.

La entrada ASYNC define dos modos de operación para sincronizar la señal READY.

Dispositivos asíncronos

Cuando $\overline{\text{ASYNC}}$ está en "0" lógico, se utilizan dos etapas de sincronización para las señales READY del 8284. Es decir, con este terminal se le indica al 8284A si los dispositivos que manejarán las entradas de RDY1 y RDY2 son síncronos ($\overline{\text{ASYNC}}$ = "1"), y por lo tanto sincronizados por CLK; o si son asíncronos (se usan dos etapas de sincronización).

Si los dispositivos que manejan RDY1 y RDY2 son asíncronos, es decir no sabemos cuándo activarán RDY1 o RDY2, éstos deberán activar estas señales antes del estado T2 del ciclo de bus del 8086, las cuales muestrearán tanto en la subida como en la bajada de este estado, para finalmente (en la bajada de T2) generar o activar la salida READY del 8284A. En otras palabras, la entrada RDY1 o RDY2 que se maneja por el dispositivo asíncrono, deberá activarse al menos el tiempo que dure el estado T2 para poder sincronizarse con el reloj del 8086.

Por otra parte, cuando un dispositivo asíncrono desactiva la entrada RDY1 o RDY2 (transición de bajada), dicha entrada únicamente se sincroniza con el segundo flip-flop y con la bajada CLK, y después de esto la salida READY del 8284A se desactiva.

La salida READY siempre se activará o desactivará con una transición de bajada.

Cuando la entrada RDY1 o RDY2 se desactiva, sólo se sincroniza con la bajada del pulso de reloj debido a la compuerta AND que se encuentra en la entrada del IFF2.

Dispositivos síncronos

Cuando $\overline{\text{ASYNC}}$ está a "1" lógico a "al aire", el primer flip-flop de la lógica de sincronización de READY no "funciona" y las entradas RDY1 y RDY2 del 8284 se sincronizan con el segundo flip-flop (con la bajada CLK) antes de que la salida READY del 8284A sea activada.

Este modo se usa para dispositivos que activan las señales RDY1 o RDY2 en sincronía con la señal de CLK.

La entrada $\overline{\text{ASYNC}}$ del 8284A se puede cambiar en cada ciclo de bus para seleccionar el modo apropiado de sincronización para cada dispositivo del sistema.

Tanto el 8284A como el 8086 y los dispositivos asíncronos que manejan las entradas RDY1 o RDY2 están conectados a un bus del sistema. Por esto el 8284A tiene estas dos señales de RDY1 y RDY2 de sincronización y este bus es el estándar llamado MUTLIBUS.

Por otra parte, las señales de entrada $\overline{\text{AEN1}}$ y $\overline{\text{AEN2}}$ permiten inhibir las entradas RDY1 y RDY2 respectivamente.

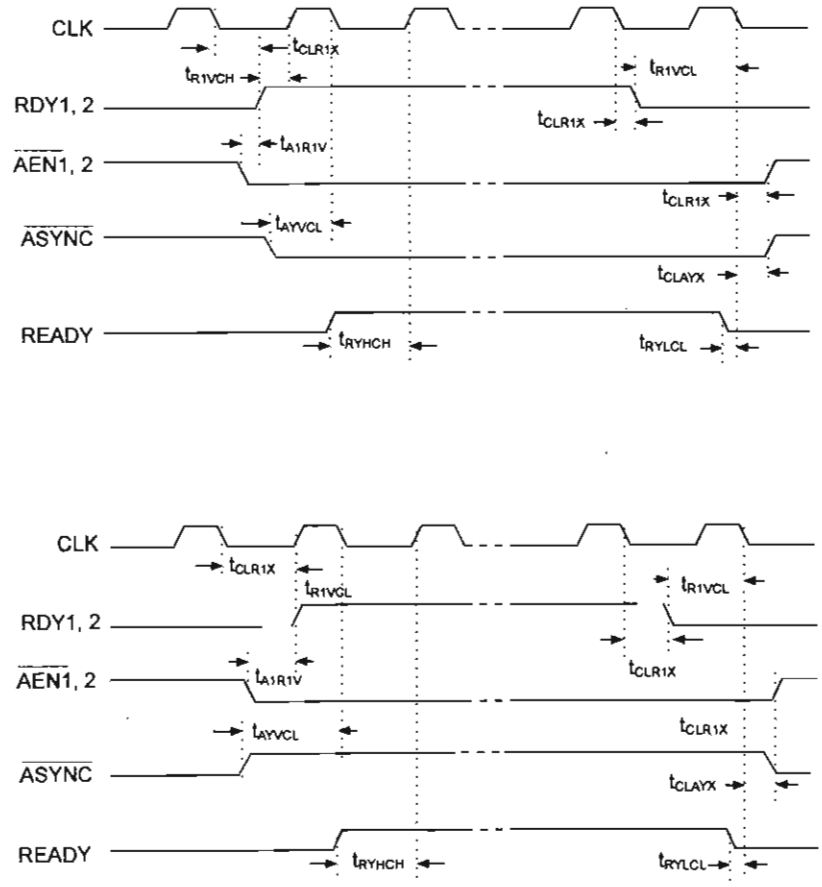


Figura I.29. Temporización de la señal RDY para dispositivos síncronos.

Ejemplo:

En la figura I.30 se muestran las conexiones de un 8284A para trabajar con un 8086-2 a 8Mhz, suponiendo un cristal de 24 Mhz.

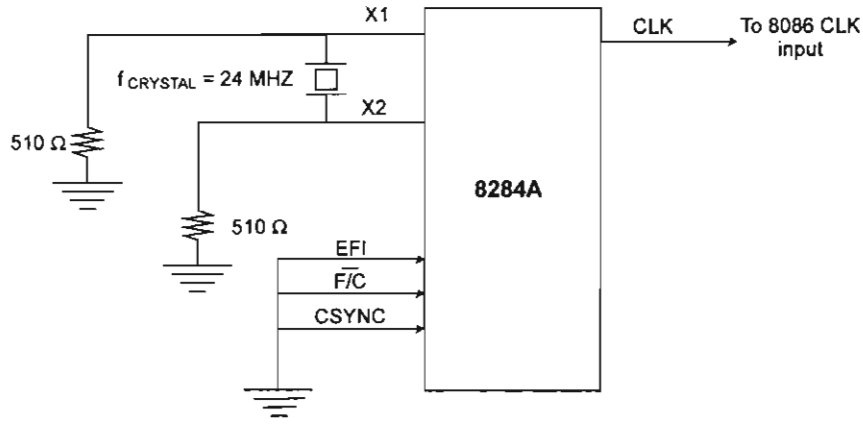


Figura I.30. Diagrama de conexiones de un 8284.

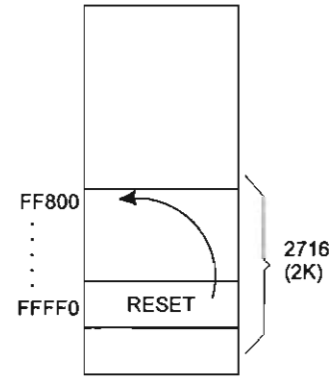
Las resistencias de 510 ohms a tierra las recomienda Intel para la estabilidad del oscilador.

Inicialización del 8086

Cuando el terminal de RESET se lleva a un activo alto, los registros de la CPU se inicializan a los valores mostrados a continuación:

CUADRO I.27. *Contenido de los registros del 8086 al activar RESET*

CPU	*Contenido*
Banderas	Limpio
Instruction pointer	0000H
Registro CS	FFFFH
Registro DS	0000H
Registro SS	0000H
Registro ES	0000H
Queue	Vacío



Nótese que el registro CS se inicializa con FFFFH, lo cual implica que la CPU busque la primera instrucción en la dirección física FFFF0H + 0000 = FFFF0H.

Entonces, la memoria RAM no puede usarse en esta dirección y, por otra parte, sólo existen 16 bytes; de la dirección FFFF0H a la FFFF0H normalmente es un salto a otra dirección donde reside el programa.

Puede utilizarse una EPROM 2716 (2K × 8) que comience en la localidad FF800H y termine en la FFFFFH, y en la FFFF0H colocar un salto a la dirección FFF80H, por ejemplo, donde residirá un programa de *bootstrap*, el cual se encargará de inicializar al sistema (por ejemplo puede cargar de disco en RAM el sistema operativo).

Generación de la señal reset

En la figura I.31 se muestra cómo se puede usar el 8284A para generar una señal de RESET según los requerimientos del 8086. La entrada $\overline{\text{RES}}$ se sincroniza con al caída del pulso de reloj del sistema.

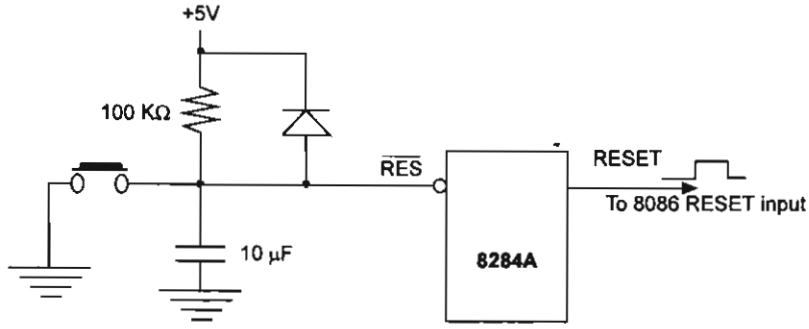


Figura I.31. Señal de reset del 8284A.

El procesador se puede inicializar (activar RESET) de dos formas: presionando el switch (la salida RESET permanecerá activada tantos ciclos como pulsos de reloj esté presionado el switch) automáticamente o *power-on reset*, según se muestra en la figura I.32.

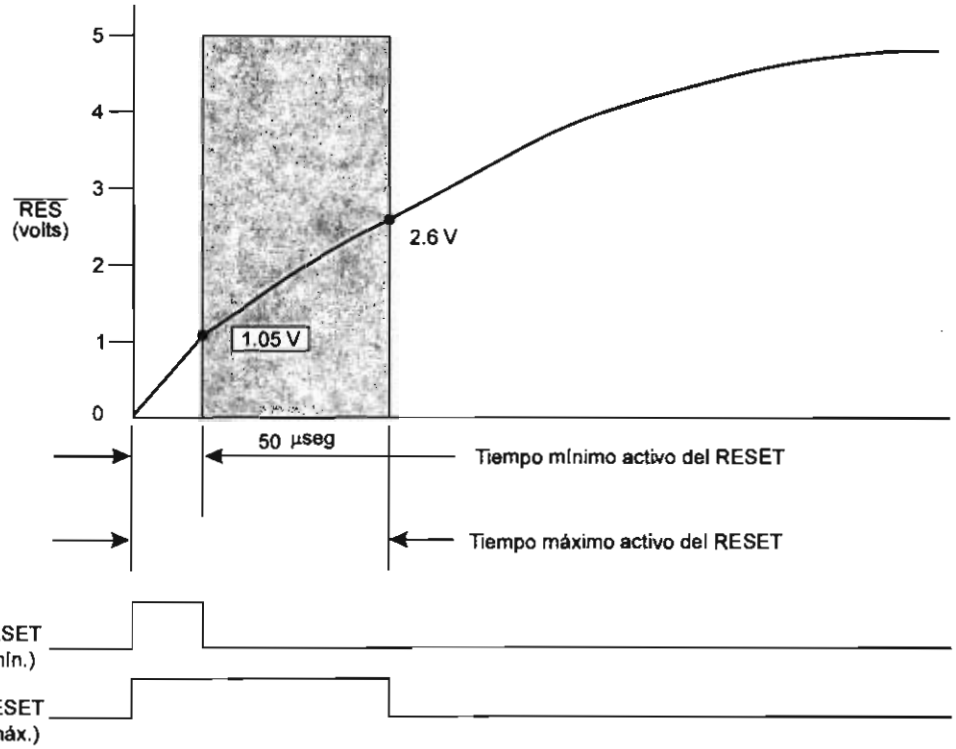


Figura I.32. Inicialización del 8086 (RESET).

Cuando se enciende la fuente de alimentación, el capacitor tiende a cargarse a +5V. Con una constante de tiempo adecuada, la entrada $\overline{\text{RES}}$ permanecerá en "0" lógico el tiempo requerido por la CPU.

El 8284A tiene un Schmitt trigger en su entrada RES para protegerlo de los picos de voltaje del capacitor. El pulso de salida no terminará antes de que la señal de entrada alcance 2.6V.

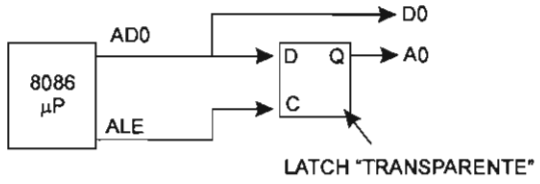
El diodo sirve como un camino de descarga del capacitor a través de la fuente de alimentación (en lugar del 8284A) cuando dicha fuente se apague.

El 8086 en modo mínimo

Cuando el pin MN/$\overline{MX}$ del 8086 se conecta a +5V, el procesador opera en modo mínimo. En este modo la CPU entrega todas las señales de control para el sistema. Esto se utiliza en diseños de pequeño y mediano tamaños con un solo procesador.

Por otra parte, muy pocos dispositivos de entrada/salida y memorias son compatibles con el bus de datos y direcciones multiplexado del 8086, por lo que requerimos separar las líneas de datos y direcciones, lo cual también se le conoce como "demultiplexar".

En la figura I.33 vemos cómo la línea AD0 es separada en dos líneas: línea de direcciones A0 y línea de datos D0. Durante el tiempo que ALE está en "1" lógico, el latch es "transparente" y en la salida Q se encuentra el valor de AD0 (una línea de direcciones); con la caída de ALE, el lacth almacena el bit de dirección A0, el cual permanece en los estados T2 a T4.



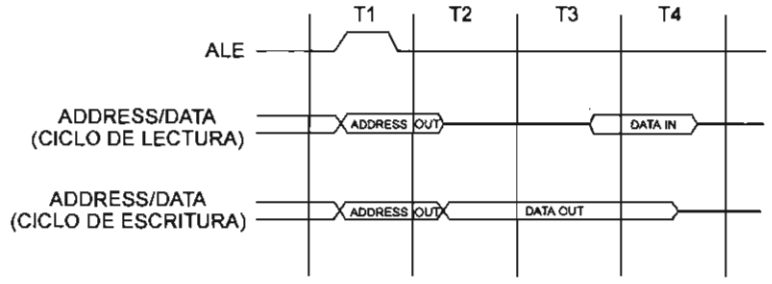


Figura I.33. Temporización datos/direcciones del 8086 en modo mínimo.

La línea de datos D0 no es demultiplexada, ya que la memoria o dispositivos de entrada/salida no accesan el bus de datos hasta el estado T2, mientras el contenido de D0 se considera *don't care*.

Para demultiplexar las líneas de direcciones, podemos utilizar diferentes circuitos integrados, uno de ellos es el 8282/8283 de Intel, el cual tiene ocho latches y una entrada de muestreo común *(strobe)*.

El 8282 tiene salidas no invertidas, mientras el 8283 invierte los datos de entrada.

Estos circuitos integrados también sirven como buffers para las líneas de direcciones, incrementando la cantidad de cargas que pueden manejar.

intel

8282/8283
OCTAL LATCH

- **Address Latch for iAPX 86, 88, 186, 188, MCS-80®, MCS-85®, MCS-48® Families**
- **High Output Drive Capability for Driving System Data Bus**
- **Fully Parallel 8-Bit Data Register and Buffer**
- **Transparent during Active Strobe**
- **3-State Outputs**
- **20-Pin Package with 0.3" Center**
- **No Output Low Noise when Entering or Leaving High Impedance State**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The 8282 and 8283 are 8-bit bipolar latches with 3-state output buffers. They can be used to implement latches, buffers, or multiplexers. The 8283 inverts the input data at its outputs while the 8282 does not. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with these devices.

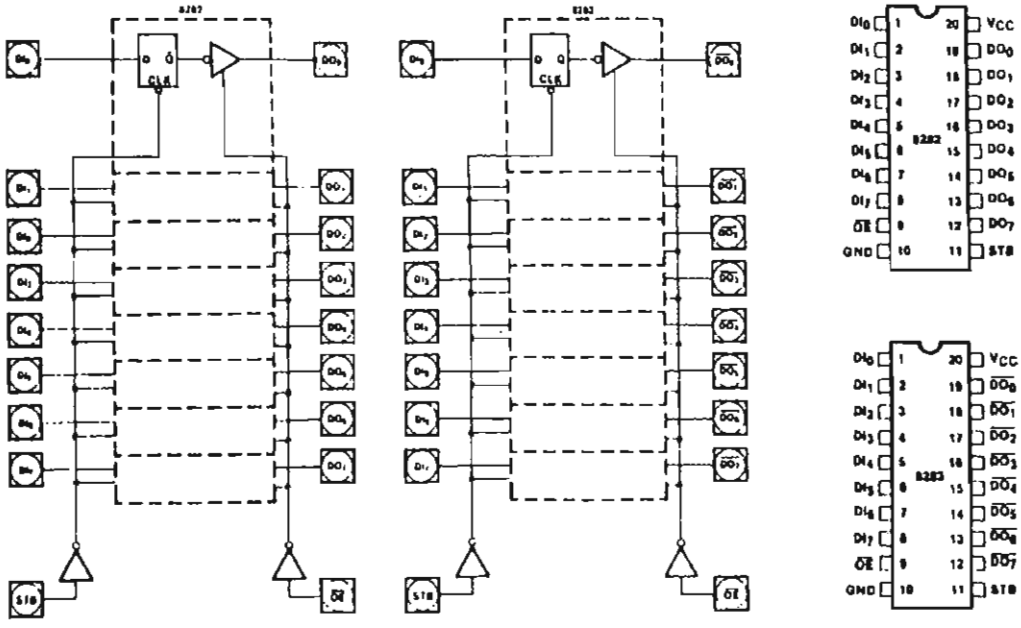


Figura I.34. Hoja de especificaciones eléctricas del 8282/8283.

Viendo algunas características eléctricas del 8282, los valores de salida para nivel bajo son 0.45V máximo para una *sink current* de 32mA máximo. Para el nivel alto, se especifica 2.4V máximo suministrando 5mA máximo.

Así entonces, el número máximo de cargas TTL standard (I_{IH} = 40 μA e I_{IL} = –1.6mA que pueden manejar el 8282 y el 8283 son:

- En el estado alto la corriente es 500 μA (5mA).
- Entonces 5000μA/40μA = 125 cargas.
- En el estado bajo 32 mA/1.6mA = 20 cargas.
- Por lo tanto el nivel bajo establece el límite, un total de 20 cargas TTL.

El 8282 y el 8283 introducen un tiempo de retardo que reduce el tiempo a la memoria o dispositivos de entrada/salida para leer o escribir. Este tiempo depende de la carga capacitiva de los latches. Por ejemplo si están manejando 20 cargas TTL, la carga capacitiva es aproximadamente de 100pF y el tiempo de propagación a través del latch es de 32ns máximo.

Para 125 cargas TTL, la carga capacitiva es de aproximadamente 600pF y el tiempo de propagación máximo es de 40 ns.

Ya que el 8283 tiene una etapa inversora menos que el 8282, "switcheará" más rápido y los tiempos de propagación serán de 22ns y 30ns máximo para las mismas condiciones de carga del 8282 vistas anteriormente.

Podemos ver que el sistema anterior se compone de un generador de reloj y reset 8284A, tres latches octales 8282 para demultiplexar las líneas de direcciones/datos y un arreglo de compuertas para las señales de control para la memoria y dispositivos de entrada/salida.

Las cuatro líneas S3-S6 no se usan, ya que básicamente son para diagnóstico y prueba. El bus de control incluye las señales $\overline{\text{BHE}}$ (demultiplexada), $\overline{\text{INTA}}$ y RESET, las cuales también pueden usarse o requerirse por la memoria o dispositivos de entrada/salida. La B indica que las líneas son "buffereadas".

Las líneas de control y de direcciones pasan a través de un buffer con la salida de tercer estado habilitado por la señal HLDA. Durante un ciclo de reconocimiento de una petición de DMA, esta señal pasa a "1" lógico, deshabilitando estos buffers y desconectando a la CPU del sistema. Las resistencias de *pull-up* en las señales de control sirven para asegurar que dichas líneas permanezcan inactivas cuando se conmuta entre un procesamiento normal y un ciclo de DMA.

La entrada $\overline{\text{TEST}}$ se activa, ya que cuando está en "1" lógico, y cada vez que se encuentra en el programa una instrucción WAIT, el programa se suspende y la CPU entra en modo idle. Recordemos que normalmente es manejada por el 8087.

La generación de las señales $\overline{\text{IOR}}$, $\overline{\text{IOW}}$, $\overline{\text{MEMR}}$ y $\overline{\text{MEMW}}$ es igual a como lo vimos anteriormente.

Las entradas NMI, INTR y HOLD se controlan por dispositivos externos.

Por otra parte, en el 8284A se observa que sus entradas F/$\overline{\text{C}}$ y EFI están a "0" lógico, lo que significa que este circuito trabajará con su oscilador interno.

La entrada CSYNC sirve para sincronizar al reloj que entrega el 8284A con un evento externo (limpia a los contadores internos).

Las señales de entrada de RDY1 y RDY2 del 8284A son controladas de forma externa, usualmente por memorias o dispositivos lentos para solicitar estados wait a la CPU (recordemos que la CPU espera hasta que la memoria esté lista, cuando su entrada READY pasa a "1" lógico).

El pin CLK de la CPU deberá tener una onda cuadrada con un *duty cycle* de 30% y una amplitud de 0V y 3.9V mínimo para "0" y "1" lógico respectivamente. La frecuencia debe ser un tercio de la frecuencia del cristal conectado en X1 y X2 del 8284A.

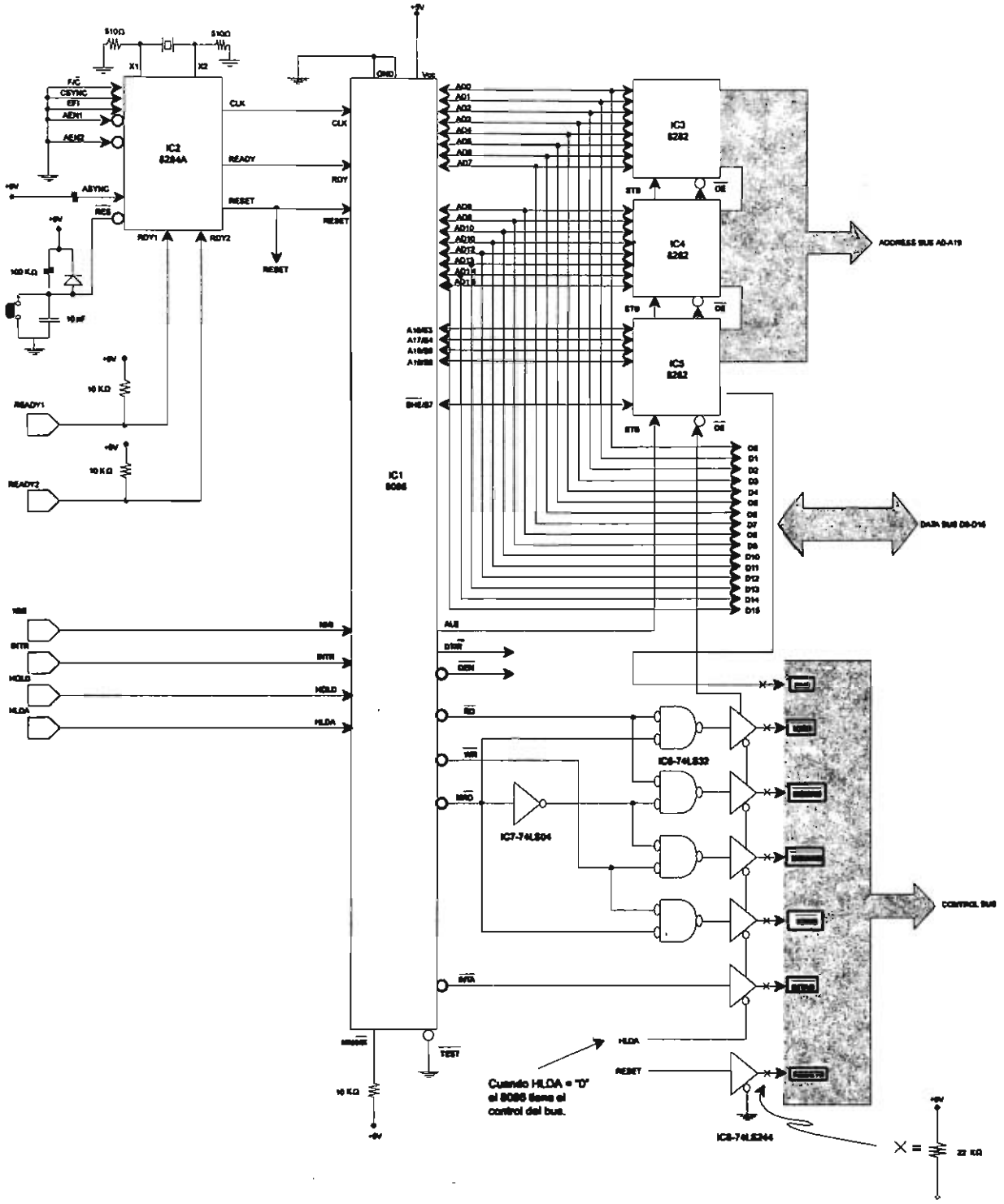


Figura I.35. Diagrama de bloques con un sistema de 8086 (modo mínimo).

SISTEMA MULTIPROCESADOR

El 8086 en modo máximo

Cuando el 8086 está operando en modo máximo, requiere el controlador de bus 8288 para generar muchas de las señales de control del sistema. Algunos terminales dedicados a funciones de control, ahora son redefinidos como señales de soporte para coprocesador. Estas señales proporcionan ahora el estado de la queue para seguir la ejecución de una instrucción y mecanismos para manejar el bus del sistema y así evitar conflictos cuando se accesen recursos (memoria y dispositivos de entrada/salida) compartidos entre la CPU y otros procesadores.

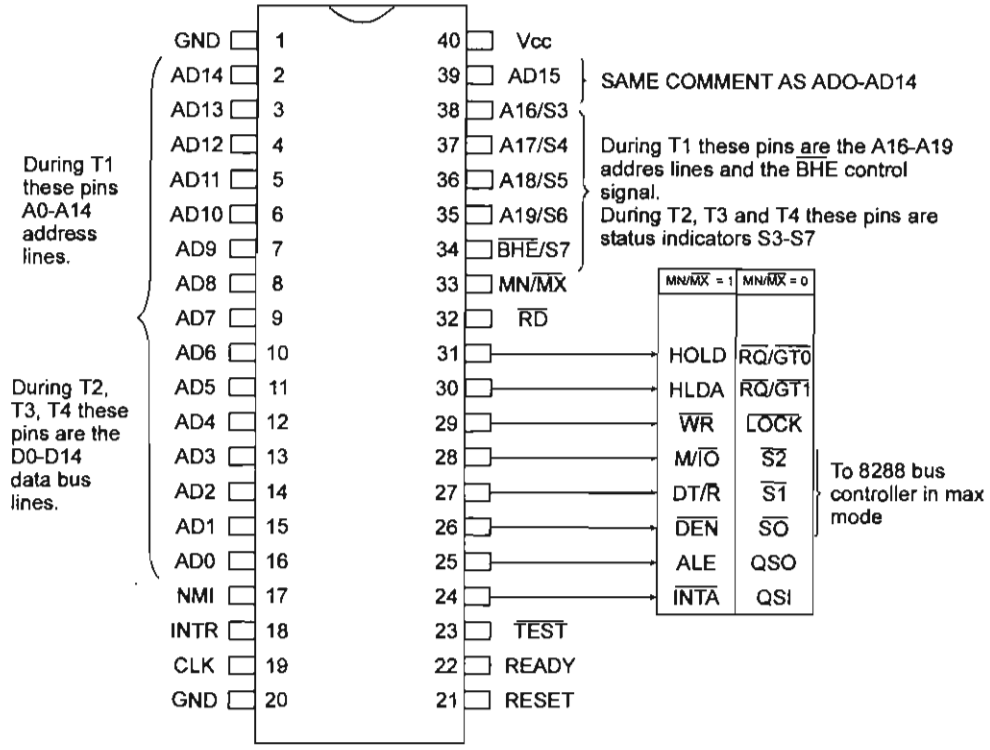


Figura I.36. La CPU 8086.

El bus del coprocesador

Cuando la línea MN/MX del 8086 se alambra a tierra, los terminales 24-31 de la CPU cambian su función, de tal manera que todas las señales del bus de control se pierden, excepto RD, es decir, ya no son generadas por la CPU.

Request/grant (RQ0/GT0, RQ1 GT1). Estas dos líneas son bidireccionales, y le permiten al coprocesador solicitar el control de los buses del sistema (similar a la solicitud de HOLD en el modo mínimo). El 8086 responde desconectándose de los buses del sistema y pulsando la línea RQ/GT como reconocimiento. Lo que realiza a continuación el coprocesador es un acceso directo a memoria. Al terminar, el coprocesador pulsa de nuevo la línea RQ/GT y la CPU toma los buses del sistema. Si se reciben solicitudes simultáneas en estos terminales, RQ0/GT0 tendrá prioridad.

$\overline{\text{LOCK}}$. Es una señal de salida que se usa en un esquema con árbitro con otro procesador. Árbitro se refiere al proceso o dispositivo que determina qué procesador tendrá el control de los buses del sistema en cualquier tiempo dado. La señal $\overline{\text{LOCK}}$, de salida, tendrá un activo bajo ("0" lógico) durante la ejecución de cualquier instrucción con el prefijo LOCK. Por ejemplo, la instrucción

```
LOCK XCHG    AL, STAT
```

intercambia el contenido de la localidad de memoria STAT con AL. Mientras se está ejecutando esta instrucción, la señal de $\overline{\text{LOCK}}$ se usa para evitar que otro procesador accese los buses del sistema.

Usado de esta forma, STAT podrá ser un semáforo. Un semáforo es una bandera que indica si un coprocesador (u otro recurso del sistema; una impresora compartida, por ejemplo) está ocupado. Antes de ejecutar la sección del programa que requiere al coprocesador, la CPU prueba la variable STAT. Si dicha variable tiene "0" lógico, el recurso está disponible y STAT es puesta a "1" lógico (el recurso está ocupado) para prevenir que otro procesador tome el recurso. Para prevenir también que otro procesador solicite al mismo tiempo al coprocesador cuando se realiza la instrucción de prueba de TEST, ésta deberá tener el prefijo LOCK.

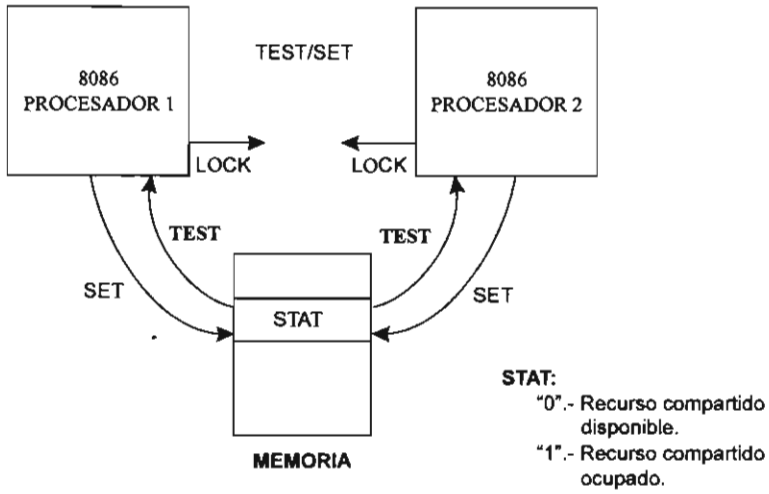


Figura I.37. Compartición de recursos.

Queue status (QS0, QS1). Estos dos terminales se utilizan para que los coprocesadores reciban sus instrucciones mediante el prefijo ESC. Permiten al coprocesador seguir el progreso de una instrucción en la queue y determinar cuándo accesar el bus para tomar el código de operación que tiene el prefijo ESC y el operando correspondiente, según lo muestra el cuadro I.28.

CUADRO I.28. *Significado de las líneas QS0 y QS1 del 8086*

QS1	*QS0*	*Instrucción corriente*
0	0	No operación
0	1	Primer byte del código de operación en la queue
1	0	Queue vacía
1	1	Byte subsecuente de la queue

Status output (S0, S1 y S2). Estas tres salidas se utilizan para el controlador de bus 8288. Codifican las señales del bus de control perdidas con el modo máximo. El cuadro I.29 muestra esta codificación. Estos terminales son válidos durante los cuatro estados de un ciclo de bus (incluyendo estados wait).

CUADRO I.29. *Significado de las líneas S0, S1 y S2 del 8288*

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	*Estado del procesador*	*Salida activa del 8288*
0	0	0	Interrupción aceptada	$\overline{\text{INTA}}$
0	0	1	Lectura del puerto entrada/salida	$\overline{\text{IORC}}$
0	1	0	Escritura del puerto de entrada/salida.	$\overline{\text{IOWC}}$ (también advanced $\overline{\text{IOWC}}$)
0	1	1	Halt	Ninguno
1	0	0	Acceso a código	$\overline{\text{MRDC}}$
1	0	1	Lectura a memoria	$\overline{\text{MRDC}}$
1	1	0	Escritura a memoria	$\overline{\text{MWTC}}$ (también advanced $\overline{\text{MWTC}}$)
1	1	1	Pasivo	Ninguno

intel

8288
BUS CONTROLLER
FOR iAPX 86, 88 PROCESSORS

- **Bipolar Drive Capability**
- **Provides Advanced Commands**
- **Provides Wide Flexibility in System Configurations**
- **3-State Command Output Drivers**
- **Configurable for Use with an I/O Bus**
- **Facilitates Interface to One or Two Multi-Master Busses**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel® 8288 Bus Controller is a 20-pin bipolar component for use with medium-to-large iAPX 86, 88 processing systems. The bus controller provides command and control timing generation as well as bipolar bus drive capability while optimizing system performance.

A strapping option on the bus controller configures it for use with a multi-master system bus and separate I/O bus.

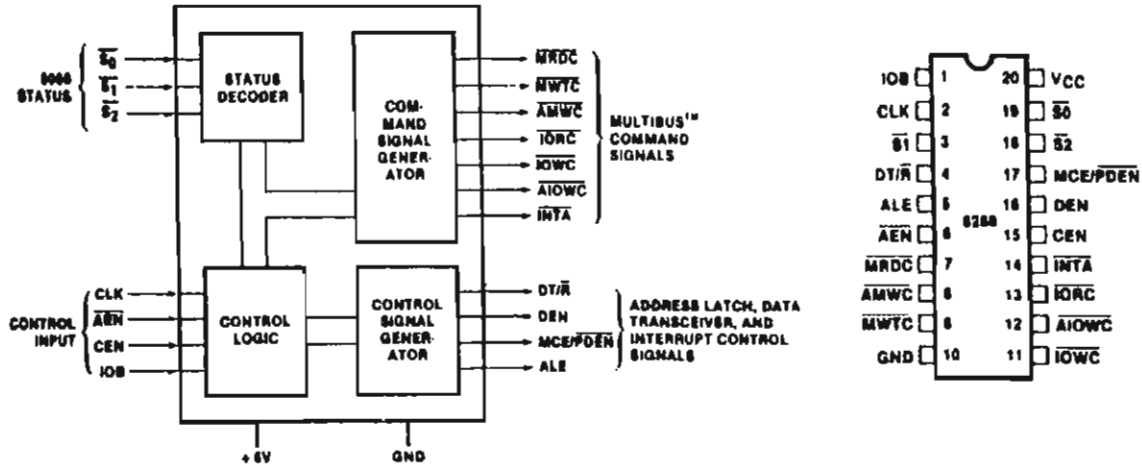


Figura I.38. Hoja de especificaciones eléctricas del 8288.

El controlador de bus 8288 se usa cuando el 8086 trabaja en modo máximo. Recibe cuatro entradas del 8086: las señales de estado S0 → S2 y CLK. Tiene dos grupos de salidas de control o de comando.

Un grupo son las señales de comando MULTIBUS, las cuales son las siguientes: $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$ e $\overline{\text{IOW}}$. Estas señales son renombradas como: $\overline{\text{MRDC}}$, $\overline{\text{MWTC}}$, $\overline{\text{IORC}}$ e $\overline{\text{IOWC}}$. El sufijo C indica comando. Dentro de este grupo también se encuentra la señal de reconocimiento de interrupción (interrupt acknowledge): INTA.

Las señales $\overline{\text{AMWC}}$ y $\overline{\text{AIOWC}}$ son señales de comando avanzadas para escritura a memoria y dispositivos de entrada/salida respectivamente. Estas señales se activan un ciclo de reloj antes que los comandos de escritura normal y son útiles con memorias y dispositivos que requieren un pulso de escritura más ancho.

El segundo grupo de señales de salida son el bus de control, y está compuesto por: DT/$\overline{\text{R}}$, DEN, ALE y MCE/$\overline{\text{PDEN}}$. Las primeras tres son similares a las que suministra el 8086 en modo mínimo (DEN es ahora activo alto). La señal MCE/$\overline{\text{PDEN}}$ es una salida con dos funciones, dependiendo del modo de operación del 8288:

a) I/O bus control	Modo bus de entrada/salida
b) System bus control	Modo bus del sistema (system bus)

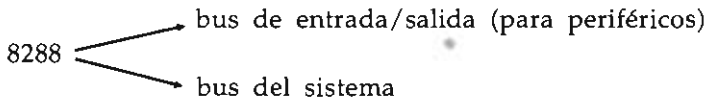
El modo de operación del 8288 lo determinan las tres señales de control CEN, IOB y $\overline{\text{AEN}}$ de la manera mostrada en el cuadro I.30:

CUADRO I.30. *Modos de operación del 8288*

CEN	*IOB*	$\overline{AEN}$	*Descripción*
1	1	X	Modo bus entrada/salida; todas las líneas de control son habilitadas; $\overline{\text{MC}}$/PDEN = PDEN.
1	0	1	Modo bus del sistema; todas las señales de control son deshabilitadas, el bus está ocupado, esto es, controlado por otro bus maestro.
1	0	0	Modo bus del sistema; todas las señales de control son activadas; el bus es libre para usarse; $\overline{\text{MCE}}$/PDEN = MCE.
0	X	X	Todas las salidas de comando y las salidas de DEN y $\overline{\text{PDEN}}$ son deshabilitadas (circuito abierto).

Modo bus de entrada/salida

Cuando CEN *(command enable)* e IOB se activan ("1" lógico), el 8288 opera en modo bus de entrada/salida. La salida MCE/$\overline{\text{PDEN}}$ actúa como una señal de habilitación de periférico *(peripheral data enable)*. Su función es idéntica a DEN excepto que ésta únicamente se activa durante instrucciones de entrada/salida. Esto le permite al 8288 controlar dos conjuntos de buses: buses normales del sistema (posiblemente compartidos con otros procesadores) y un bus de entrada/salida especial para periféricos.



$\overline{\text{ANE}}$ = Address enable
CEN = Command enable
IOB = Input/output bus
MCE = Master cascade enable

Modo bus del sistema

Las señales de control están activas sólo si las entradas $\overline{\text{AEN}}$ (address enable) e IOB son activo bajo. Esta forma permite que varios 8288 (y 8086) se puedan interfasar al mismo bus. Un árbitro de bus seleccionará al procesador activo habilitando únicamente un 8288 (mediante la entrada $\overline{\text{AEN}}$). En este modo MCE/$\overline{\text{PDEN}}$ significa MCE: master cascade enable.

Diseño de un módulo usando una CPU en modo máximo

En la figura I.39 se muestra un módulo que usa una CPU en modo máximo. Usa un 8284A para generar la señal de reloj y sincronizar las señales de RESET y READY. A diferencia del modo mínimo, se incluye un cuarto bus llamado "bus del coprocesador", el cual tiene señales para el 8087 NDP y el 8089 I/O Processor.

Las líneas de direcciones datos se demultiplexan usando latches 8282 como en el modo mínimo. La señal STB se deriva de ALE, generada ahora por el 8288.

En este sistema de un solo procesador, el 8288 opera en modo "bus de sistema" con sus señales de control permanentemente activadas (IOB = "0" y $\overline{\text{AEN}}$ = "0").

Las líneas de datos son buffereadas con un 8286, el cual es un *bus tranceiver* octal. Estos buffers sólo se activan cuando la salida DEN está activa en los pulsos de reloj T2 → T4. La dirección de los tranceivers la controla DT/$\overline{\text{R}}$. El sufijo B en cada señal significa buffereada.

Más adelante analizaremos un ejemplo donde el 8288 se configura para operar en "modo bus de entrada/salida".

SISTEMAS MULTIPROCESADOR

El diseño de un sistema computacional puede incluir más de una CPU, lo que incrementa de manera significativa sus capacidades. Esto se debe a que cada procesador puede operar en paralelo o concurrentemente con otros.

Por otro lado, también es común particionar un sistema computacional en subsistemas, cada uno controlado por su propio procesador. Por ejemplo, una CPU se puede dedicar a leer un teclado y otra a manejar las operaciones de lectura y escritura a un floppy disk. También se puede usar un coprocesador aritmético para cálculos científicos.

Cuando se necesita hacer el sistema tolerante a fallas se usan a veces varias CPU asignadas a la misma tarea. Un comparador verifica la salida de cada sistema. Si la salida de un sistema difiere con respecto a los otros, éste es aislado y un sistema de *backup* toma su lugar. Esto es posible hoy día gracias a la economía de la tecnología VLSI *(very large scale integration)*.

Sin embargo, existen dos principales problemas en un sistema "multi-CPU", y son: el arbitraje del bus y el uso del bus multiciclos.

El primer problema se relaciona con el hecho de que dos CPU no pueden accesar la misma localidad de memoria o dispositivos de entrada/salida simultáneamente. Es decir, se tiene que usar un árbitro para esas solicitudes.

CPU 8086
DISEÑO DE UN MÓDULO MÁXIMO

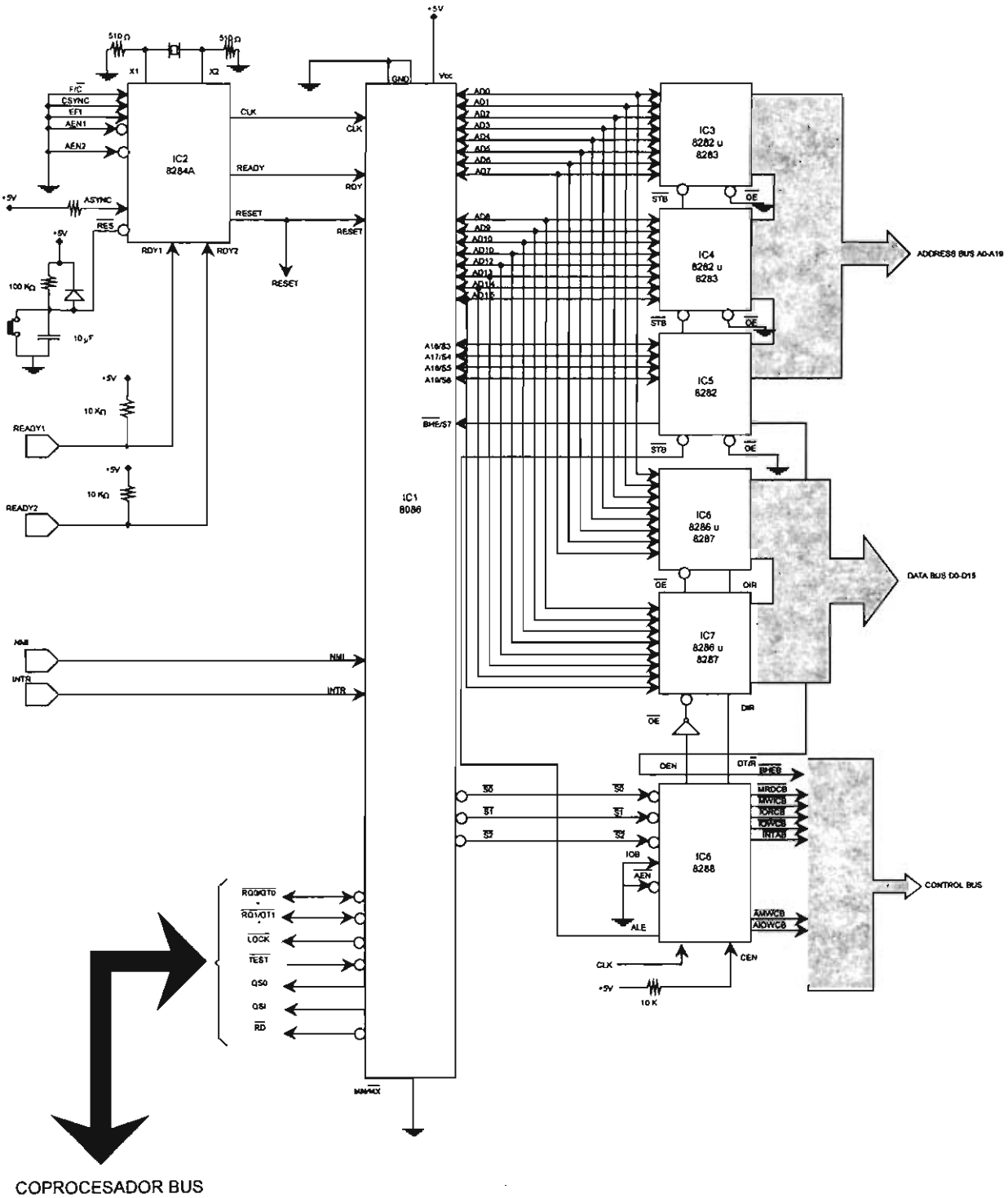


Figura I.39. Diagrama de bloques de un sistema con 8086 (modo máximo).

Una vez que la CPU ha ganado el control del recurso, ésta puede requerir varios ciclos de lectura o de escritura. Durante este tiempo, otro procesador no debe inspeccionar o modificar la información que está accesando la CPU que ganó.

El 8086 y el 8088 fueron pensados para facilitar el diseño de sistemas multiprocesador, usando tres métodos o modos de operación: bus simple compartido *(shared single-bus)*, bus de entrada/salida y bus residente *(resident bus)*.

Modo bus simple compartido

En este modo dos o más CPU comparten todos los recursos del sistema, incluyendo buffers, latches y controlador de bus. Un ejemplo de esto es un 8086 (u 8088) soportado por el NDP 8087.

Los dos procesadores se alambran en paralelo y buscan instrucciones de la memoria. El NDP busca sólo aquellos con el prefijo ESC. Una vez que se encuentra una instrucción con este prefijo, el 8086 calcula una dirección efectiva pero no almacena dato alguno; en lugar de esto, realiza una instrucción NOP (no operation). El 8087, por su parte, captura el dato y comienza la ejecución. A continuación, el 8086 está libre y puede continuar ejecutando instrucciones de la queue. Así, el NDP y la CPU *host* operan concurrentemente mientras el NDP está ejecutando una instrucción.

Cuando el resultado está listo para almacenarse en memoria, el NDP pulsa la línea $\overline{RQ}/\overline{GT}$ de la CPU para solicitar el uso de los buses. Un pulso de retorno en esta línea bidireccional otorga el control de los buses al NDP. Uno o más ciclos de escritura ocurrirán para que el 8087 almacene el resultado numérico en memoria. Un tercer pulso en $\overline{RQ}/\overline{GT}$ regresa el control de los buses a la CPU. Así entonces, la línea $\overline{RQ}/\overline{GT}$ establece un protocolo físico para prevenir la disputa del bus entre la CPU y el NDP.

Cuando el 8087 comienza a ejecutar una instrucción, activa su salida BUSY. Conectada a la entrada $\overline{TEST}$ del 8086, la CPU es avisada de que el resultado en memoria aún no está listo. Recordemos que las instrucciones que hacen referencia a esos resultados deben tener el prefijo WAIT.

Esto causará que la CPU entre en un estado de wait hasta que su entrada $\overline{TEST}$ sea "0" lógico.

El 8087 expande el set de instrucciones del 8086, por lo que el circuito de la siguiente figura es considerado como una CPU en dos circuitos integrados. Intel llama a esta configuración ¡iAPX 86/21. No se requiere hardware extra más que para el NDP y es muy simple, por lo que muchos fabricantes incluyen un socket vacío para el NDP, expandiendo el usuario su sistema a un ¡iAPX 86/21.

Conclusión

El compartir recursos del sistema tiene desventajas, ya que cuando se requiere un resultado del NDP, el 8086 debe pasar y esperar el resultado. Así también, la CPU debe suspender operaciones mientras el NDP está leyendo o escribiendo en memoria, perdiéndose algunas de las ventajas del procesamiento en paralelo.

Cuando el 8087 empieza a ejecutar la operación requerida por la CPU, el 8086 puede esperar a que el 8087 termine, avisándole este último por medio de la línea $\overline{TEST}$, o bien, cuando el 8087 comienza la ejecución de una operación, la CPU calcula una dirección efectiva en la que no almacena dato alguno y ejecuta una instrucción NOP para continuar la ejecución del programa. Una vez que el 8087 termina la ejecución de la operación, deposita el resultado en memoria solicitando el control o uso de los buses a la CPU por medio de una de las líneas $\overline{RQ}/\overline{GT}$. Este resultado estará finalmente disponible para la CPU.

En la figura I.42 se puede observar que un 8288 (1) se utiliza como controlador de un bus especial de entrada/salida (se activan los latches y tranceivers A) para realizar operaciones con los disposi-

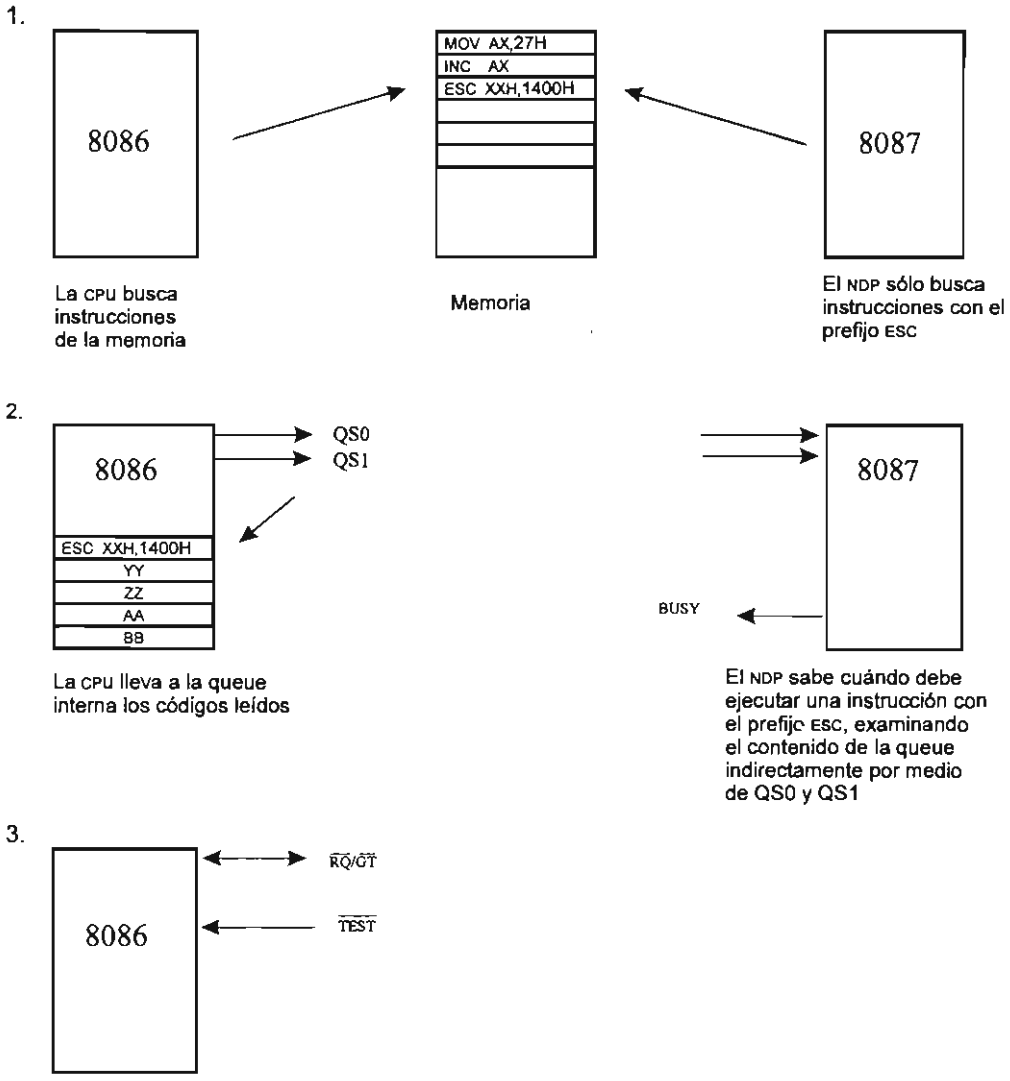


Figura I.40. Compartición de recursos entre dos procesadores.

tivos de entrada/salida usando un bus de entrada/salida local, y como controlador de un bus de sistema (se activan los latches y tranceivers B) para realizar operaciones con la RAM/ROM del sistema usando un bus de sistema.

El segundo 8288 (el 2) se usa como controlador de un bus de sistema, ya que el 8086 no realizará operaciones de entrada/salida y no utilizará el Local I/O bus.

Nótese entonces que el 8288 (1) sirve para que el I/O Processor 8089 pueda realizar operaciones o usar tanto el Local I/O bus como el bus de sistema. En otras palabras, este controlador permite que el 8089 pueda accesar dos tipos de buses o dos tipos de dispositivos (de entrada/salida o del sistema) de forma separada. El 8089 conmuta entre estos dos tipos de buses.

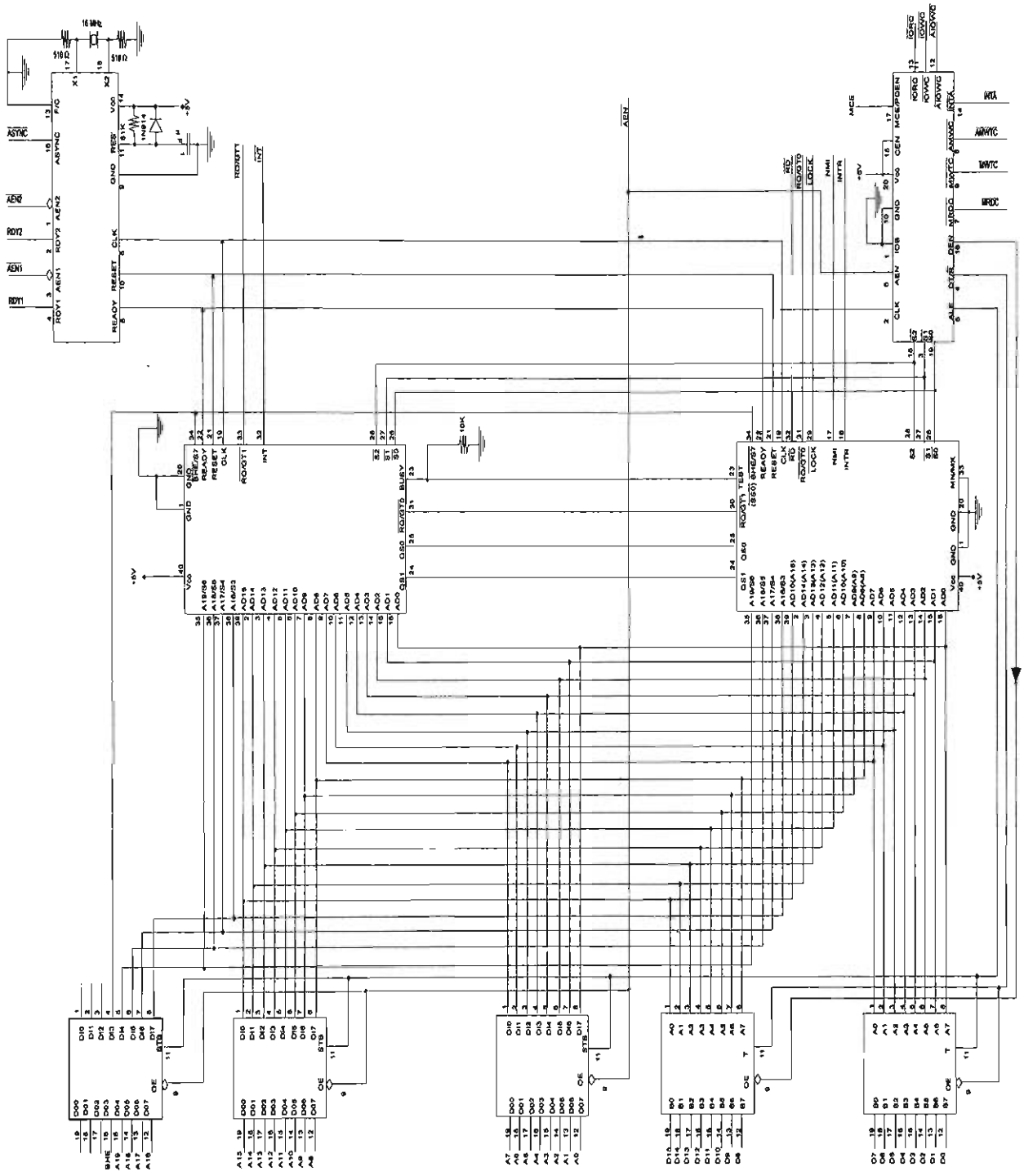
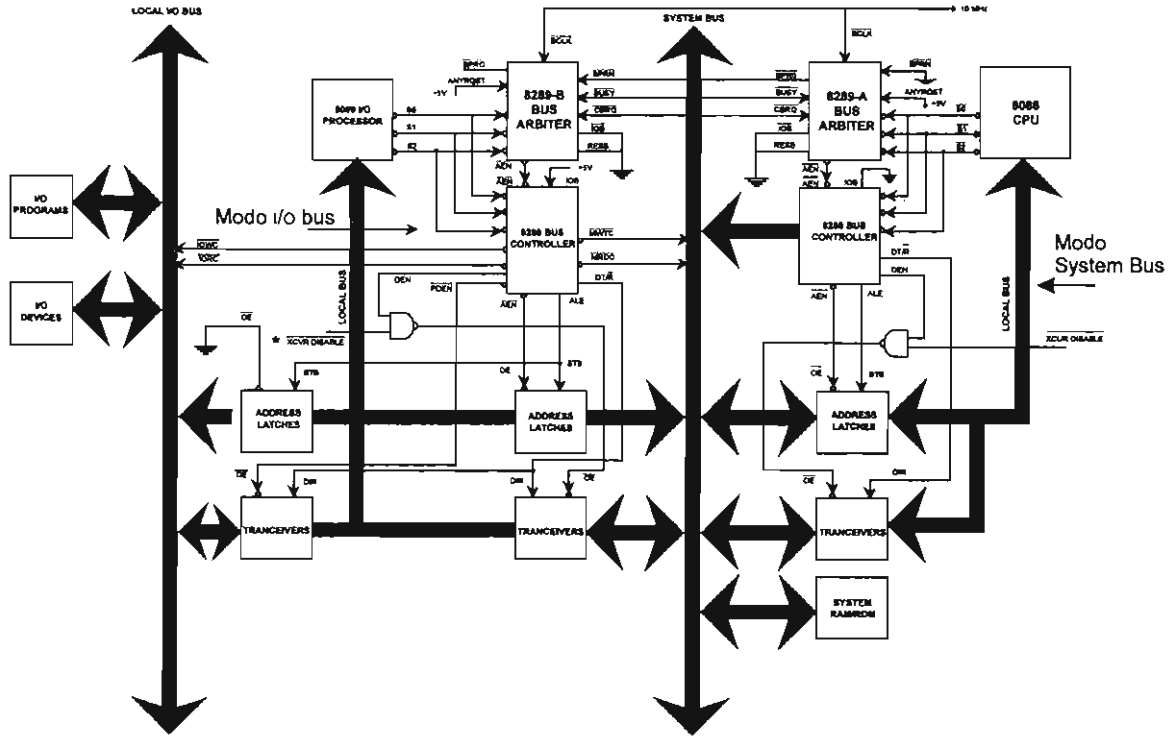


Figura 1.41. Interface del NDP 8087 con el 8086/88 en nodo máximo.



* Manejadas externamente para habilitar/deshabilitar la tarjeta.

Figura I.42. Sistema multiprocesador con un bus de entrada/salida y un bus de sistema.

Modo bus de entrada/salida

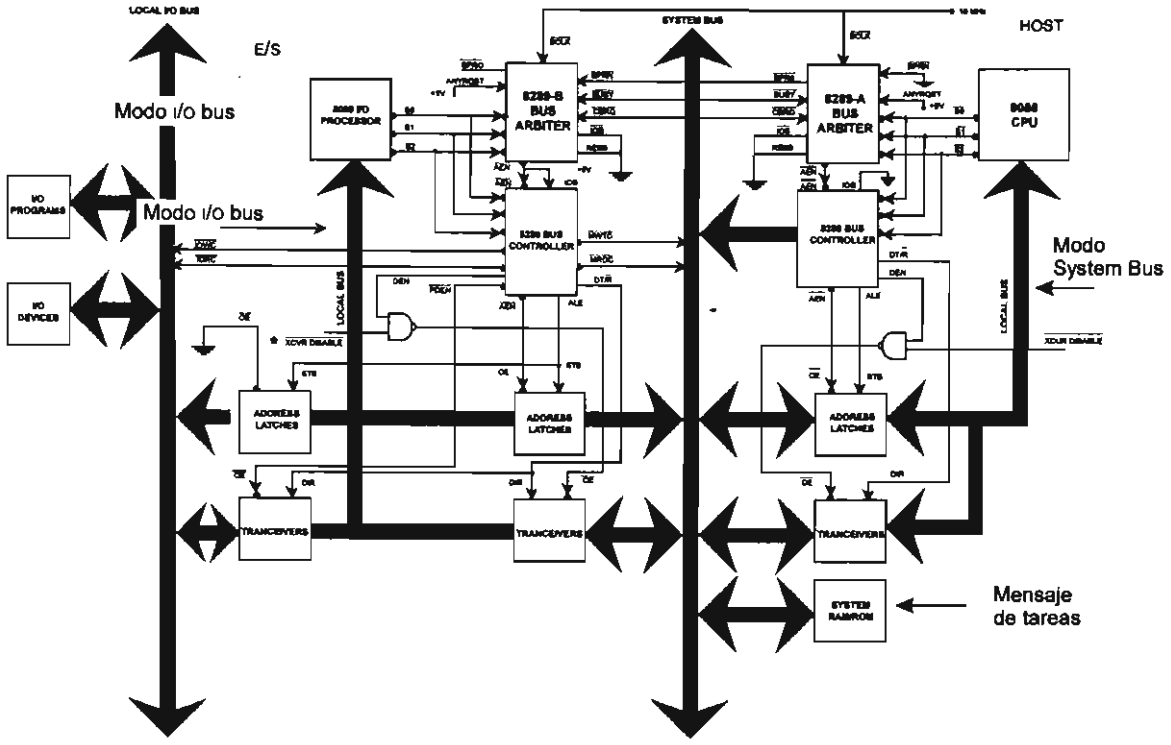
En este modo se usa un procesador especial para manejar las funciones de entrada/salida de la CPU.

La figura I.43 muestra un ejemplo donde se usa un IOP 8089 y dos conjuntos de buses. El bus de entrada/salida local (local I/O bus) sólo es accesible para el 8089. El bus de sistema (bus del sistema) se usa por el 8086 host y permite accesar a la ROM y RAM del sistema. Éste también sirve como canal de comunicaciones común entre los dos procesadores.

En este circuito el IOP se programa para modo remoto y busca instrucciones para realizar su tarea de una memoria local. Los dispositivos de entrada/salida también se mapean en este espacio. En una operación típica, la CPU almacena en la memoria del sistema mensajes relativos a la tarea a ser realizada. El IOP accesa la memoria del sistema para tomar los mensajes y llevar a cabo su tarea de manera independiente del host. Así entonces, los procesadores operan en paralelo (excepto cuando intercambian mensajes en la memoria del sistema).

Cada procesador es soportado por un controlador de bus 8288. En el lado del 8089, este circuito opera en modo bus de entrada/salida. En este modo, cualquier instrucción de entrada/salida activará $\overline{\text{PDEN}}$ (peripheral data enable). Esta señal se usa para habilitar los *tranceivers* existentes entre el IOP y el bus local.

Cuando se detecta un ciclo de lectura o escritura a memoria del 8089 (mediante $\overline{\text{S0}}$-S2), el árbitro de bus 8289 (8289-B) es alertado. Éste responde activando $\overline{\text{CBRQ}}$ *(common bus request)*. A continuación, ocurre un retardo antes de activarse la señal $\overline{\text{BPRN}}$ *(bus priority in)* mediante el 8289-A. Una vez habilitado, el árbitro activa $\overline{\text{AEN}}$, habilitando el 8288. Así, la dirección de memoria hace que las señales



* Manejadas externamente para habilitar/deshabilitar la tarjeta.

RESB = bus residente

CE = "1"
$\overline{IOB}$ = "0" } Bus de sistema
$\overline{AEN}$ = "0"

$\overline{IOB}$ = "1"
$\overline{AEN}$ = X } Bus de entrada/salida

Figura I.43. Sistema multiprocesador con un bus de entrada/salida y un bus de sistema.

de control pasen del IOP al bus del sistema mediante los latches de dirección y bus tranceiver ahora habilitados.

Cuando varias CPU necesitan usar el bus del sistema, se establece una prioridad para solicitudes simultáneas. En la figura I.43 se usa una serie técnica o *daisy chain*. La entrada $\overline{BPRN}$ del 8289-A es habilitada permanentemente, dándole la prioridad más alta. Cuando quiera este árbitro accesar el bus, su salida BPRO pasará a "1" lógico. El conectar esta salida a la entrada $\overline{BPRN}$ del siguiente árbitro en la cadena (8289-B) evita que CPU de más baja prioridad usen el bus del sistema. Sin embargo, el bus siempre puede ser solicitado por una CPU de más baja prioridad cuando $\overline{CBRQ}$ se active, pero deberá transcurrir un tiempo hasta que el dispositivo de alta prioridad no requiera el bus.

La ventaja de un bus de entrada/salida es que el host puede trabajar en otra tarea mientras el IOP controla los dispositivos de entrada/salida. Ya que el IOP puede tener almacenado su programa en una memoria local, ambos procesadores pueden operar concurrentemente. La principal desventaja es que se requiere hardware adicional, incluyendo un árbitro de bus para cada CPU y dos grupos de tranceivers y latches de direcciones.

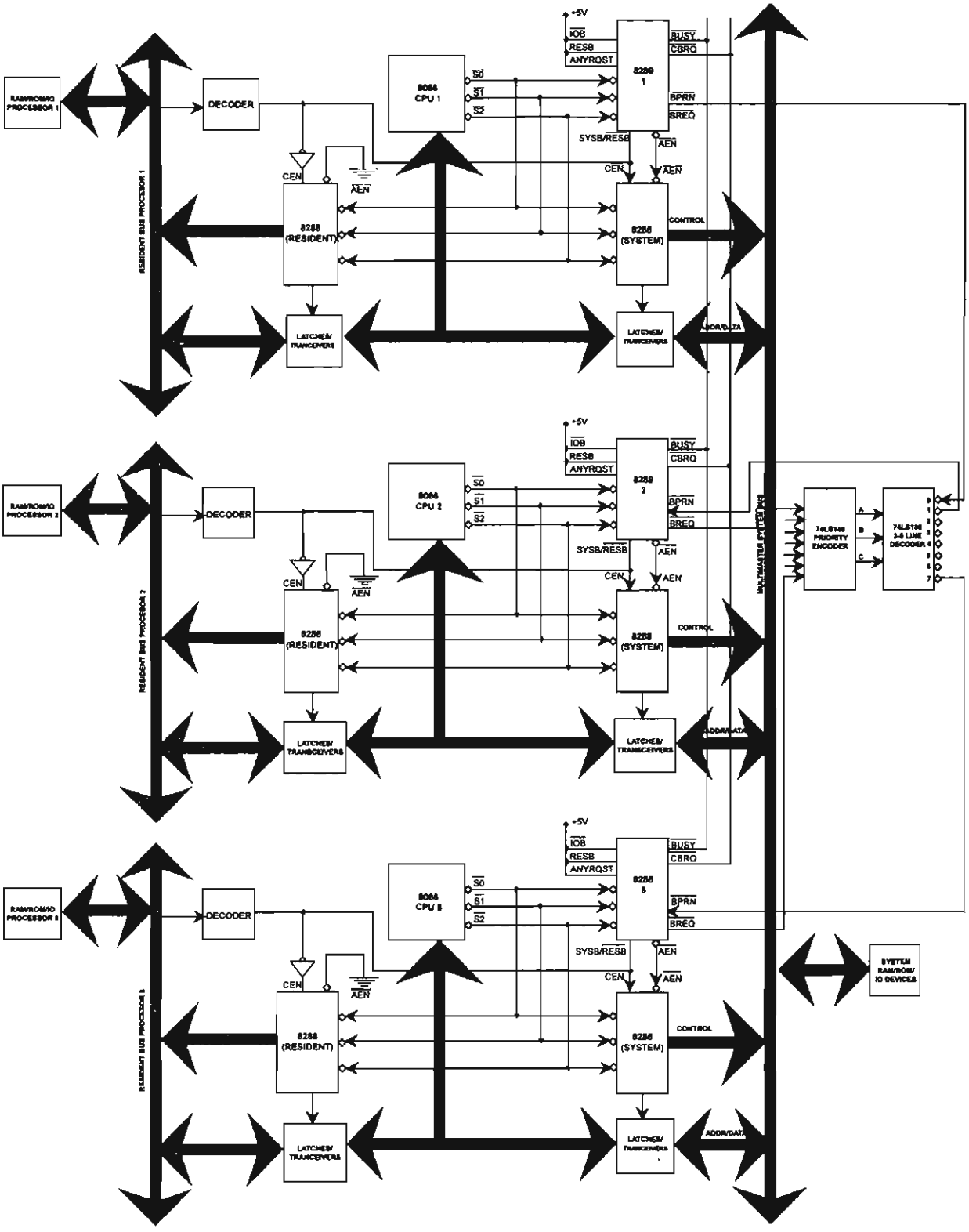


Figura I.44. Diagrama de bloques de un sistema con un bus residente.

Modo bus residente

En este modo se pueden conectar múltiples CPU de propósito general, cada una con su propio bus privado (o residente). En la figura I.44 se muestra un ejemplo donde se conectan ocho CPU con un bus del sistema común multiprocesador. Cada una puede accesar el bus del sistema o el bus local. En esta forma, se pueden compartir recursos caros entre todos los procesadores (por ejemplo discos magnéticos de alta capacidad).

Cada 8086 es soportado por dos controladores de bus 8288. Uno maneja el bus residente y el otro maneja el bus del sistema. Se utiliza también un decodificadôr de direcciones para definir el rango de direcciones de memoria como área del sistema *(system address)*. Cuando se accesa tal área, la salida del decodificador pasará a "1" lógico, causando que el 8288 residente se deshabilite y el 8288 del bus del sistema se habilite (ambos mediante las entradas CEN). Esto deshabilita todas las salidas de comando del bus residente.

El decodificador de direcciones también activa la entrada SYSB/$\overline{\text{RESB}}$ del árbitro del bus 8289, solicitando el uso del bus del sistema. El árbitro responderá activando $\overline{\text{BREQ}}$. Esta señal se conecta a un circuito que controla la prioridad de solicitudes en paralelo, compuesto por un 74LS138 y un 74LS148. El 74LS148 codifica las entradas que tienen "1" lógico y aplica el resultado a un decodificador 3x8 74LS138. Las ocho salidas de este circuito habilitan la entrada BPRN de uno de los ocho árbitros de bus.

El árbitro seleccionado habilita su correspondiente 8288 mediante $\overline{\text{AEN}}$, el cual manejará las señales de *memory read* y *memory write* del bus del sistema. Los recursos del sistema están disponibles ahora para la CPU solicitante.

Si se desea, el bus del sistema puede ser de uso exclusivo de esta CPU durante la transferencia de datos, colocándole el prefijo LOCK a cada instrucción. Si esto no se hace, otra CPU puede ganar el control del bus del sistema.

Cuando múltiples CPU se conectan cada una a su propio bus residente, se tendrá un sistema muy flexible, ya que cada CPU corre independiente de las otras excepto para accesar recursos comunes del sistema, lo cual permite que varios usuarios puedan accesar una base de datos común; por ejemplo, se debe tener cuidado en que los bloques de datos operados por un procesador no sean contaminados por otro usuario. Una forma de prevenirlos es con el prefijo LOCK en las operaciones que manipulan los datos. Es decir, se podría adicionar una lógica externa que tenga como entradas las salidas $\overline{\text{LOCK}}$ de cada una de las CPU, de forma tal que cuando una CPU está accesando los recursos comunes no permita que las otras CPU lo hagan, conectando la salida de esta lógica a la entrada $\overline{\text{BPRN}}$ o ANYRQST de los árbritos de bus.

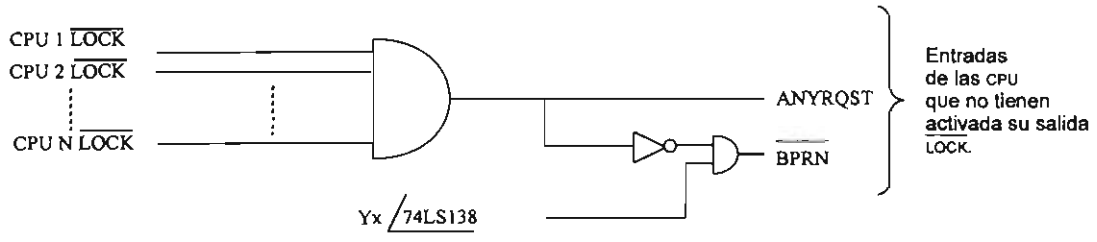


Figura I.45. Lógica externa.

CAPÍTULO II

EL MICROCONTROLADOR 8051

La familia MCS-51

La familia de microcontroladores de 8 bits MCS-51 se compone de los dispositivos comprendidos en el cuadro II.1.

Cuadro II.1. *Familia de microcontroladores de 8 bits*

Parte	*Tecnología*	*Memoria de programa*	*Memoria de datos*
8051	HMOS	4K-ROM	128
8031	HMOS	No tiene	128
8751H	HMOS I	4K-EPROM	128
80C51	CHMOS	4K-ROM	128
80C31	CHMOS	No tiene	128
8052	HMOS	8K-ROM	256
8032	HMOS	No tiene	256

Todos se basan en la arquitectura MCS-51. La versión original del 8051 se hizo con tecnología HMOS I. La actual versión con la que se fabrica el dispositivo es HMOS II y se llama 8051 AH. Sin embargo, se usa el término 8051 para referirse a todos los miembros de la familia MCS-51.

Los últimos de los miembros de la familia, el 8032 y el 8052, tienen más memoria interna y un temporizador/contador adicional de 16 bits.

Las principales características de la familia MCS-51 son:

- CPU de 8 bits.
- Oscilador y circuitería interna de reloj.
- 32 líneas de entrada/salida.
- Espacio de direccionamiento de 64 K para memoria externa de datos.
- Espacio de direccionamiento de 64 K para memoria externa de programa.
- Dos temporizador/contadores de 16 bits (tres en el 8032/8052).
- Una estructura de cinco fuentes de interrupción (seis en el 8032/802) con dos niveles de prioridad.
- Puerto serie full dúplex.
- Procesador booleano.

ARQUITECTURA INTERNA

Organización de memoria

El 8051 tiene espacios de direccionamiento separados para memoria de programa y memoria de datos. La memoria de programa puede ser de hasta 64 Kbytes. Los 4 K más bajos (8 K para el 8052) *pueden* estar en el chip. La memoria de datos puede ser de hasta 64 Kbytes en una RAM fuera del chip, y además existen 128 bytes (256 bytes para el 8052) dentro del chip, más un número de SFR (registros de funciones especiales-special function registers) que se listan a continuación:

CUADRO II.2. *Registros de funciones especiales*

Símbolo	*Nombre*	*Dirección*
*ACC	Acumulador	0E0H
*B	Registro B	0F0H
*PSW	Palabra de estado de programa	0D0H
SP	Stack pointer	81H
DPTR	Data pointer (consta de DPH y DPL)	83H
		82H
*P0	Puerto 0	80H
*P1	Puerto 1	90H
*P2	Puerto 2	9A0H
*P3	Puerto 3	0B0H
*IP	Control de prioridades de interrupciones	0B8H
*IE	Control de habilitación de interrupciones	0A8H
TMOD	Control de modo temporizador/contador	89H
*TCON	Control temporizador/contador	88H
+*T2CON	Control 2 temporizador/contador	OC8H
TH0	Temporizador/contador 0 (byte alto)	8CH
TL0	Temporizador/contador 0 (byte bajo)	8AH
TH1	Temporizador/contador 1 (byte alto)	8DH
TL1	Temporizador/contador 1 (byte bajo)	8BH
+TH2	Temporizador/contador 2 (byte alto)	0CDH
+TL2	Temporizador/contador 2 (byte bajo)	0CCH
+RCAP2H	Registro de captura de temporizador/contador 2 (byte alto)	0CBH
+PCAP2L	Registro de captura de temporizador/contador 2 (byte bajo)	0CAH
*SCON	Control del puerto serie	98H
SBUF	Buffer del puerto serie	99H
PCON	Control de potencia	87H

Los SFR con * se pueden direccionar por bit o por byte. Los SFR marcados con un signo más (+) sólo existen en el 8052.

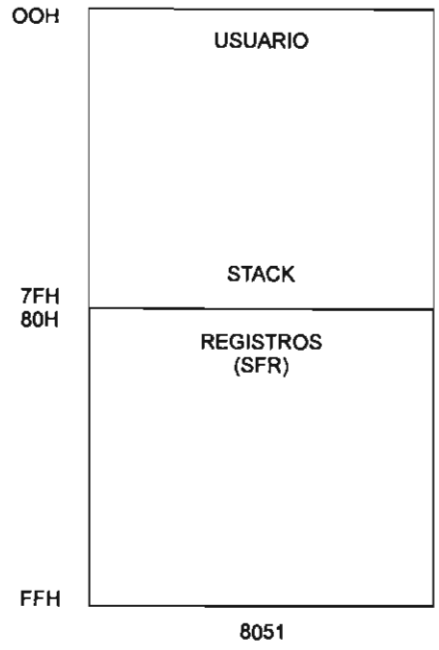


Figura II.1. Mapa de memoria de datos interna (RAM).

A continuación daremos un panorama general de los principales registros con los que cuenta el 8051 y más adelante los abordaremos más específicamente.

Acumulador

ACC es el registro acumulador. Los mnemónicos de instrucciones que usan el acumulador simplemente lo marcan como A.

Registro B

El registro B se usa durante operaciones de multiplicación y división. En otras instrucciones se puede usar como registro de uso temporal.

Palabra de estado de programa

El registro PSW contiene información del estado del programa:

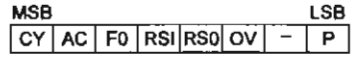


Figura II.2. Registro de estado de programa.

CUADRO II.3. *Bits de la palabra de estado de programa*

Símbolo	*Posición*	*Nombre y significado*
CY	PSW.7	Bandera de acarreo
AC	PSW.6	Bandera de acarreo auxiliar (para operaciones BCD)
F0	PSW.5	Bandera 0 (disponible para propósito general del usuario)
RSI	PSW.4	Selección del banco de registros
RS0	PSW.3	Puesta/limpiada por software para elegir el banco de registros de trabajo: (0, 0) – Banco 0 (00H-07H) (0, 1) – Banco 1 (08H-0FH) R0 → R7 (1, 0) – Banco 2 (10H-17H) (1, 1) – Banco 6 (18H-1FH)
OV	PSW.2	Bandera de sobreflujo
–	PSW.1	(Reservada)
P	PSW.0	Bandera de paridad. Puesta/limpiada por hardware cada ciclo de instrucción para indicar un número de "unos" (par/impar) del acumulador, es decir, cuando está activada indica paridad par.

Stack pointer

Este registro es de 8 bits. Se incrementa antes de que el dato se almacene con las ejecuciones de las instrucciones PUSH y CALL. El stack puede estar en cualquier parte de la RAM interna. El stack pointer se inicializa con 07H después de un reset. Esto causa que el área de stack comience en la localidad 08H.

Data pointer

El data pointer (DPTR) consta de un byte alto (DPH) y un byte bajo (DPL). Su función es almacenar una dirección de 16 bits. Se puede manejar como un registro de 16 bits o como dos registros independientes de 8 bits.

Puertos 0-3

P0, P1, P2 y P6 son los SFR latches de los puertos 0, 1, 2 y 3 respectivamente.

Buffer del puerto serie

El buffer del puerto serie son dos registros separados: uno para el buffer de transmisión y uno para el de recepción. Cuando un dato se transfiere a SBUF, pasa en realidad al buffer de transmisión, de donde se transmitirá en forma serie. Cuando un dato se lee de SBUF, es un dato que se ha recibido en forma serie.

Registros de los temporizadores (temporizador)

Los pares de registros (TH0, TL0), (TH1, TL1) y (TH2, TL2) son los registros contadores de 16 bits para los temporizador/contadores 0, 1 y 2 respectivamente.

Registros de captura

Los pares de registros (RCAP2H, RCAP2L) son los registros de captura usados cuando el temporizador 2 trabaja en modo de captura *(capture mode)*. En este modo, cuando se presenta una transición en el pin T2EX del 8052, el contenido de los registros TH2 y TL2 se copia en los registros RCAP2H y RCAP2L. El temporizador 2 tiene también un modo de autorrecarga de 16 bits, y RCAP2H y RCAP2L contienen el valor de recarga para este modo.

Registros de control

Los registros de funciones especiales (SRF) IP, IE, TMOD, TCON, T2CON, SCON y PCON contienen bits de control y estado para el sistema de interrupciones, los temporizador/contadores y el puerto serie. Estos registros los veremos más adelante.

El oscilador y el circuito de reloj

Las terminales XTAL1 y XTAL2 son la entrada y salida de un inversor interno de una etapa, el cual se puede configurar de manera externa como un oscilador Pierce, tal como se muestra en la figura II.3.

El oscilador controla el generador de reloj interno, mismo que proporciona la sincronización

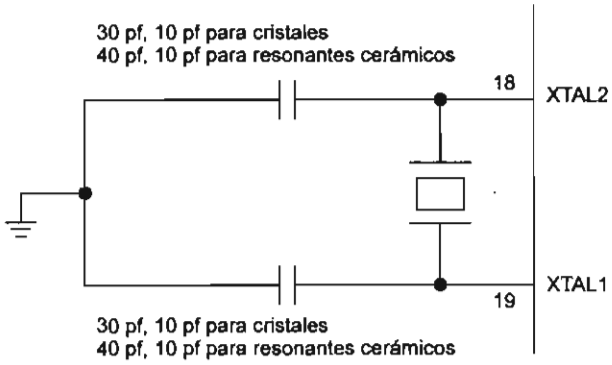


Figura II.3. Circuito de oscilador con un cristal.

interna del chip. La frecuencia de trabajo de las señales internas es la mitad de la frecuencia del oscilador y define las fases internas, estados, y ciclos de máquina, los cuales son descritos a continuación.

Sincronización de la CPU

Un ciclo de máquina consta de seis estados (12 periodos del oscilador), cada uno de los cuales se divide en dos fases:

Fase 1: Durante esta fase está presente el pulso de reloj P1.

Fase 2: En esta fase está activo el pulso de reloj P2, por lo que un ciclo de máquina consta de 12 periodos del oscilador numerados de S1P1 (state 1, phase 1) a S6P2 (state 6, phase 2). Típicamente, las operaciones aritméticas y lógicas tienen lugar durante la fase 1 y la transferencia entre registros internos durante la 2.

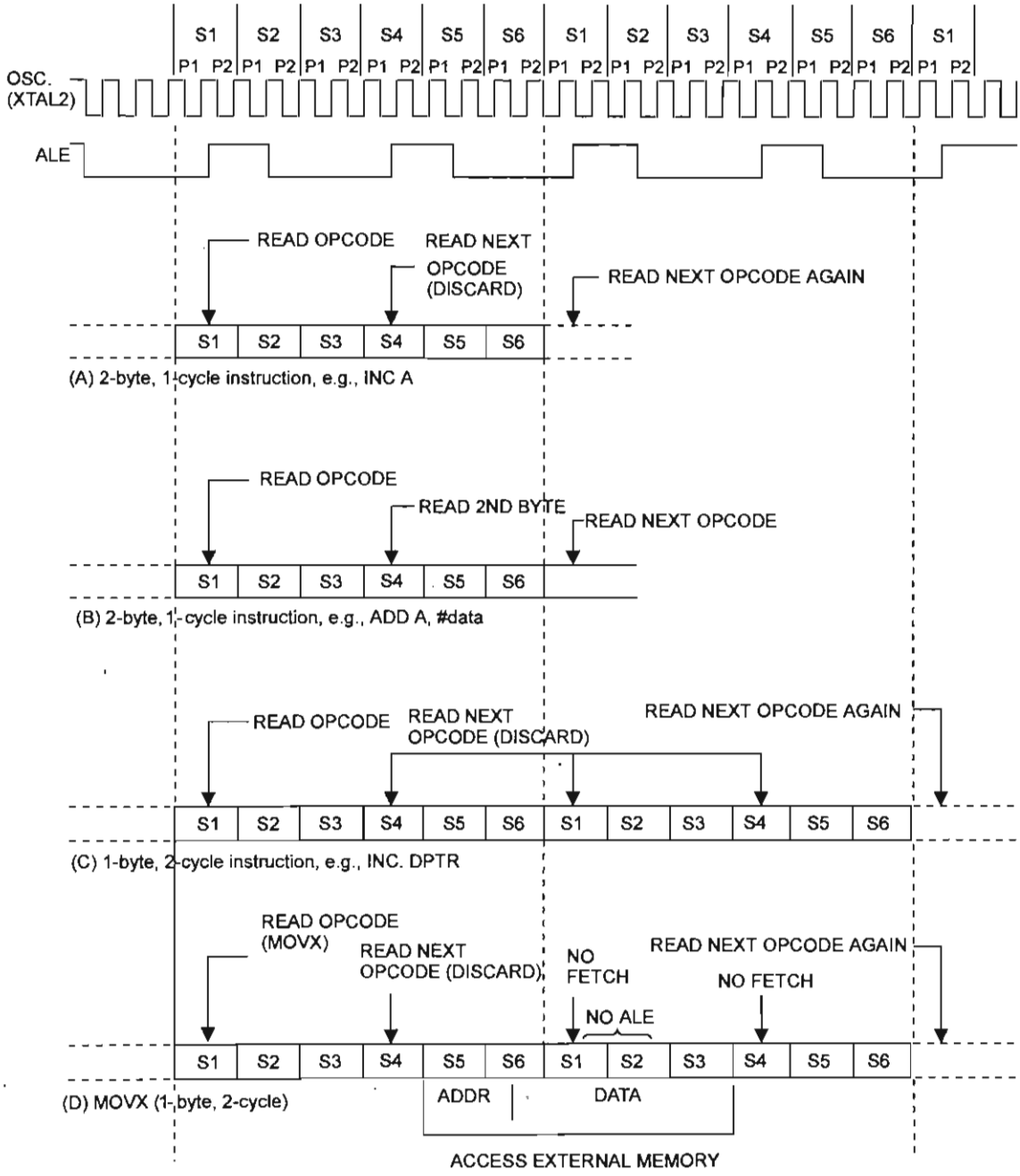


Figura II.4. Ciclo fetch y ciclo de ejecución en el 8051.

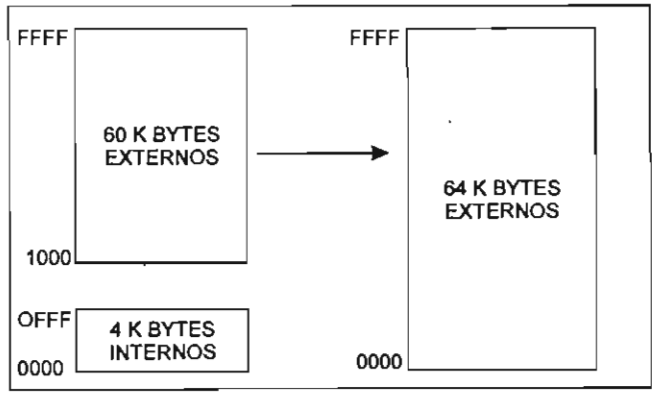
El diagrama de la figura II.4 muestra los ciclos fetch y los ciclos de ejecución referenciados a los estados y fases internos. Ya que estas señales de reloj no son accesibles para el usuario, las señales del oscilador XTAL2 y ALE (address latch enable) se muestran para referencias externas. ALE se activa normalmente dos veces cada ciclo de máquina: una vez durante S1P2 y S2P1 y otra durante S4P2, y S5P1.

La ejecución de una instrucción de un ciclo comienza en S1P2, cuando el código de operación es latcheado en el registro de instrucción. Si es una instrucción de dos bytes, el segundo byte se lee durante S4 del mismo ciclo de maquina. Si la instrucción es de un byte de código de operación, se realiza un fetch en S4, pero el byte leído es ignorado y no se actualiza el valor del program contador. En cualquiera de los casos la ejecución se completa al final de S6P2. La figura también muestra la temporización para instrucciones de un byte y de dos ciclos. Muchas instrucciones del 8051 se ejecutan en un ciclo. Solamente las instrucciones MUL y DIV toman más de dos ciclos para su ejecución.

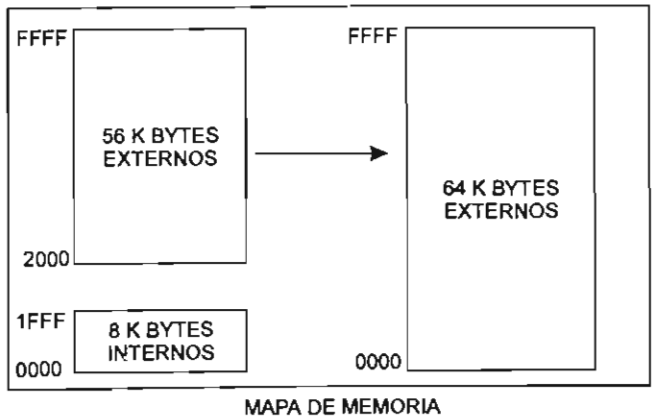
Organización de memoria

Memoria de programa

El 8051 tiene espacios separados de direccionamiento para memoria de programa y memoria de datos. La memoria de programa puede ser de hasta 64K. Los 4 K (8K para el 8052) más bajos pueden residir en el circuito integrado.



MAPA DE MEMORIA
DE PROGRAMA DEL 8051



MAPA DE MEMORIA
DE PROGRAMA DEL 8052

Figura II.5. Mapas de memoria de programa para el 8051 y el 8052.

Memoria de datos

El 8051 puede direccionar hasta 64K de memoria externa de datos. La instrucción MOVX se usa para accesar a la memoria externa de datos.

El 8051 tiene 128 bytes de RAM internos (26 bytes en el 802) más un número de registros de funciones especiales (SFR). Los 128 bytes más bajos de la RAM se pueden accesar directamente (MOV destino, fuente o MOV dirección, dato) o indirectamente (MOV @Ri).

La figura II.6 muestra la organización de la memoria de datos en el 8051 y en el 8052:

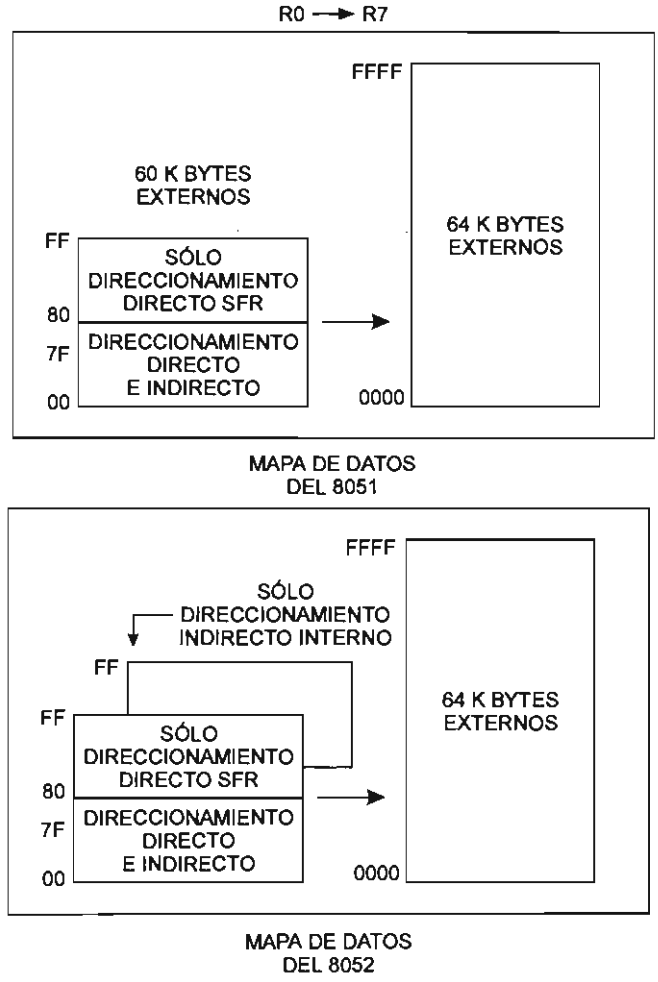


Figura II.6. Mapas de memoria de datos para el 8051 y el 8052.

Área de acceso indirecto

Nótese que en la figura anterior los SFR y el área de RAM de acceso indirecto tienen las mismas direcciones (80H-0FFH). Sin embargo, son dos áreas separadas y se accesan en dos formas diferentes.

Por ejemplo, la instrucción:

```
MOV 80H, #0A      (Acceso directo)
```

escribe 0AAH al puerto 0, el cual es uno de los SFR, y la instrucción:

```
MOV      R0, #80H
MOV  @R0, #0BBH      (Acceso indirecto)
```

escribe 0BBH en la localidad 80H de la RAM de datos. Así entonces, después de la ejecución de las instrucciones anteriores, el puerto 0 contendrá 0AAH y la localidad 80 de la RAM contendrá 0BBH.

Área de acceso directo e indirecto

Los 128 bytes de RAM que pueden accesarse *directa e indirectamente* se pueden dividir en tres segmentos según se muestra en la figura II.7.

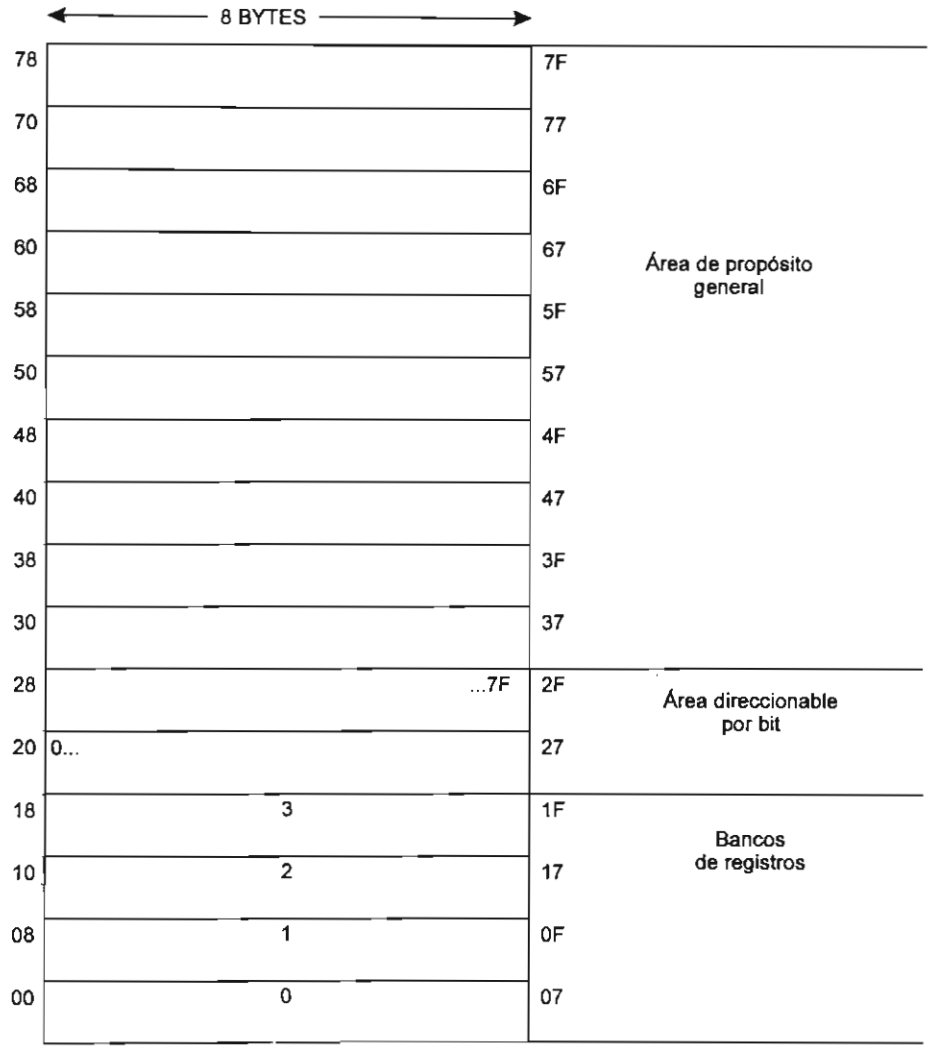


Figura II.7. 128 bytes de RAM direccionables directa e indirectamente.

1. Bancos de registros 0-3: comprenden las localidades 0 a la 1FH (32 bytes). Después de aplicar un reset al 8051, el banco es seleccionado por default en el 0. Para usar cualquiera de los otros, el usuario debe seleccionarlos por software. Cada banco contiene 8 registros de un byte del R0 al R7.

El reset también inicializa al stack pointer a la localidad 07H y es incrementado para comenzar desde la 08H, la cual es el primer registro (R0) del segundo banco. Así pues, si se usa más de un banco de registros, el SP debe inicializarse a una localidad diferente de la RAM, es decir, donde no se almacenen datos (por ejemplo, la parte alta de la RAM).

2. Área direccionable por bit: se han asignado 16 bytes para este segmento, del 20H al 2FH. Cada uno de los 128 bits de este segmento se puede direccionar directamente (del 0 al 7FH).

Los bits se pueden referenciar en dos formas: refiriéndose a su dirección, es decir, 0 a 7FH, y refiriéndose a los bytes, 20H a 2FH. Así entonces, los bits 0-7 se pueden referenciar también como los bits 20.0-20.7 y los bits del 8-FH como 21.0-21.7, y así sucesivamente.

	ÁREA DIRECCIONABLE POR BIT								BYTE
20.0-20.7	0	1	2	3	4	5	6	7	20H
21.0-21.7	8	9	A	B	C	D	E	F	21H
22.0-22.7	10	11	12	13	14	15	16	17	22H
2F.0-2F.7	78	79	7A	7B	7C	7D	7E	7F	2FH

Figura II.8. Área de memoria interna de datos direccionable por bit del 8051.

3. Área de propósito general (scratch pad area): los bytes 30H a 7FH están disponibles para el usuario. Sin embargo, si el stack pointer fue inicializado a esta área, debe existir un número de bytes libres suficientes para evitar la destrucción de datos por el SP.

Registros de funciones especiales

El cuadro II.4 lista todos los SFR y sus direcciones correspondientes. Indica también cuáles son o cuáles se pueden accesar por byte, o por byte y por bit (marcados con *). Asimismo señala los que únicamente están disponibles en el 8052 (con un signo +). También, en la figura II.9 se puede ver que todos los SFR que son direccionables o que se pueden accesar por byte y por bit, se localizan en la primera columna de la figura.

En el cuadro II.5 se lista el contenido de cada SFR después de un reset por hardware o al encender la fuente de alimentación *(power-on)*.

CUADRO II.4. *Registros de funciones especiales*

Símbolo	*Nombre*	*Dirección*
*ACC	Acumulador	0E0H
*B	Registro B	0F0H
*PSW	Palabra de estado de programa	0D0H
SP	Stack pointer	81H
DPTR	Data pointer (consta de DPH y DPL)	83H
		82H
*P0	Puerto 0	80H
*P1	Puerto 1	90H
*P2	Puerto 2	9A0H
*P3	Puerto 3	0B0H
*IP	Control de prioridad de interrupción	0B8H
*IE	Control de habilitación de interrupción	0A8H
*TMOD	Control de modo temporizador/contador	89H
TCON	Control temporizador/contador	88H
+*T2CON	Control 2 temporizador/contador	0C8H
TH0	Temporizador/contador 0 (byte alto)	8CH
TL0	Temporizador/contador 0 (byte bajo)	8AH
TH1	Temporizador/contador 1 (byte alto)	8DH
TL1	Temporizador/contador 1 (byte bajo)	8BH
+TH2	Temporizador/contador 2 (byte alto)	0CDH
+TL2	Temporizador/contador 2 (byte bajo)	0CCH
+RCAP2H	Registro de captura de temporizador/contador 2 (byte alto)	0CBH
+PCAP2L	Registro de captura de temporizador/contador 2 (byte bajo)	0CAH
*SCON	Control serial	98H
SBUF	Buffer serial de datos	99H
PCON	Control de potencia	87H

* Se puede accesar por byte o por bit.
+ Disponible únicamente en el 8052

Mapa de memoria de los SFR									
	8 bytes								
F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
AB	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	PI								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	PO	SP	DPL	DPH				PCON	87

Direccionable por byte o por bit.

Figura II.9. Ubicación de los registros de funciones especiales en el 8051.

CUADRO II.5. *Contenido de los SFR después de un reset*

Símbolo	*Nombre*	*Valor en binario*
*ACC	Acumulador	00000000
*B	Registro B	00000000
*PSW	Palabra de estado de programa	00000000
SP	Stack pointer	07H
DPTR	Data pointer (consta de DPH y DPL)	00000000 00000000
*P0	Puerto 0	11111111
*P1	Puerto 1	11111111
*P2	Puerto 2	11111111
*P3	Puerto 3	11111111
*IP	Control de prioridad de interrupción	8051 XXX00000 8052 XX000000
*IE	Control de habilitación de interrupción	8051 0XX00000 8052 0X000000
TMOD	Control de modo temporizador/contador	00000000
*TCON	Control temporizador/contador	00000000
+*T2CON	Control 2 temporizador/contador	00000000
TH0	Temporizador/contador 0 (byte alto)	00000000
TL0	Temporizador/contador 0 (byte bajo)	00000000
TH1	Temporizador/contador 1 (byte alto)	00000000
TL1	Temporizador/contador 1 (byte bajo)	00000000
+TH2	Temporizador/contador 2 (byte alto)	00000000
+TL2	Temporizador/contador 2 (byte bajo)	00000000
+RCAP2H	Registros de cap. temporizador/contador 2 (byte alto)	00000000
+PCAP2L	Registros de cap. temporizador/contador 2 (byte bajo)	00000000
*SCON	Control serial	00000000
SBUF	Buffer serial de datos	Indeterminado
PCON	Control de potencia	HMOS 0XXXXXXX CHMOS 0XXX0000

X = Indefinido.
* = Direccionable por bit.
+ = Sólo en el 8052.

Estructura y operación de los puertos paralelos

Los cuatro puertos del 8051 son bidireccionales. Cada uno consta de un latch (los registros de funciones especiales P0 a P3), un driver de salida y un buffer de entrada.

Los drivers de salida de los puertos 0 y 2 y los buffers de entrada del puerto 0 se usan para accesar a la memoria externa.

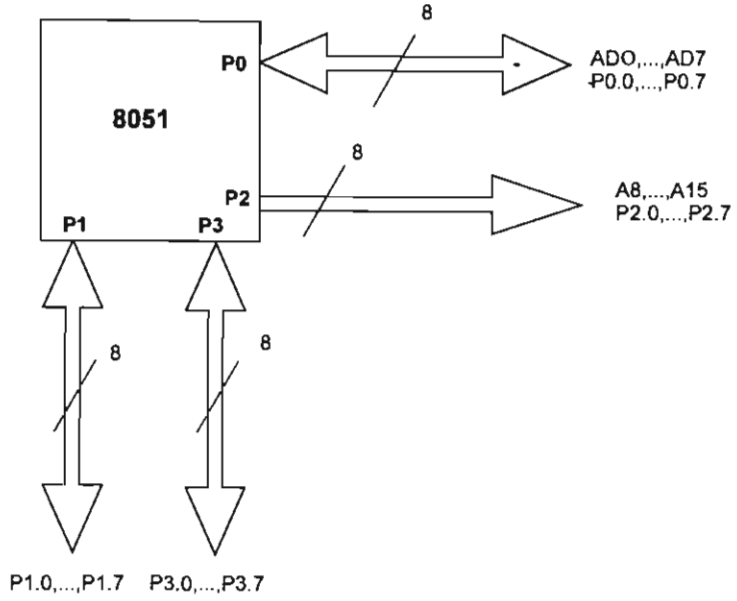


Figura II.10. Puertos paralelos del 8051.

En esta aplicación, el puerto 0 saca el byte bajo de la dirección de memoria externa, multiplexado en tiempo con el byte a leer o a escribir. El puerto 2 saca el byte alto de la dirección de memoria externa.

Todas las terminales del puerto 3, y (en el 8052) dos pines del puerto 1 son multifuncionales. Es decir, no sólo son líneas de un puerto, sino que también tienen las siguientes funciones:

CUADRO II.6. *Terminales multifuncionales de los puertos 1 y 3*

Pin puerto	*Función alternativa*
*P1.0	T2 (entrada externa del temporizador/contador 2)
*P1.1	T2EX (captura disparo del temporizador/contador 2)
P3.0	RXD (entrada del puerto serie)
P3.1	TXD (salida del puerto serie)
P3.2	$\overline{\text{INT0}}$ (interrupción externa 0)
P3.3	$\overline{\text{INT1}}$ (interrupción externa 1)
P3.4	T0 (entrada externa del temporizador/contador 0)
P3.5	T1 (entrada externa del temporizador/contador 1)
P3.6	$\overline{\text{WR}}$ (escritura a memoria de datos externa)
P3.7	$\overline{\text{RD}}$ (lectura de memoria de datos externa)

* P1.0 y P1.1 tienen estas funciones alternas sólo en el 8052. Las funciones alternas únicamente se pueden activar si el correspondiente bit-latch del puerto del SFR tiene un "1".

Configuraciones de entrada/salida

La figura II.11 muestra un diagrama funcional de un bit-latch y un buffer de entrada/salida de cada uno de los cuatro puertos. El bit-latch se representa como un flip-flop tipo D, el cual se dispara con una señal de escritura al latch proveniente de la CPU, almacenando el valor tomado del bus interno.

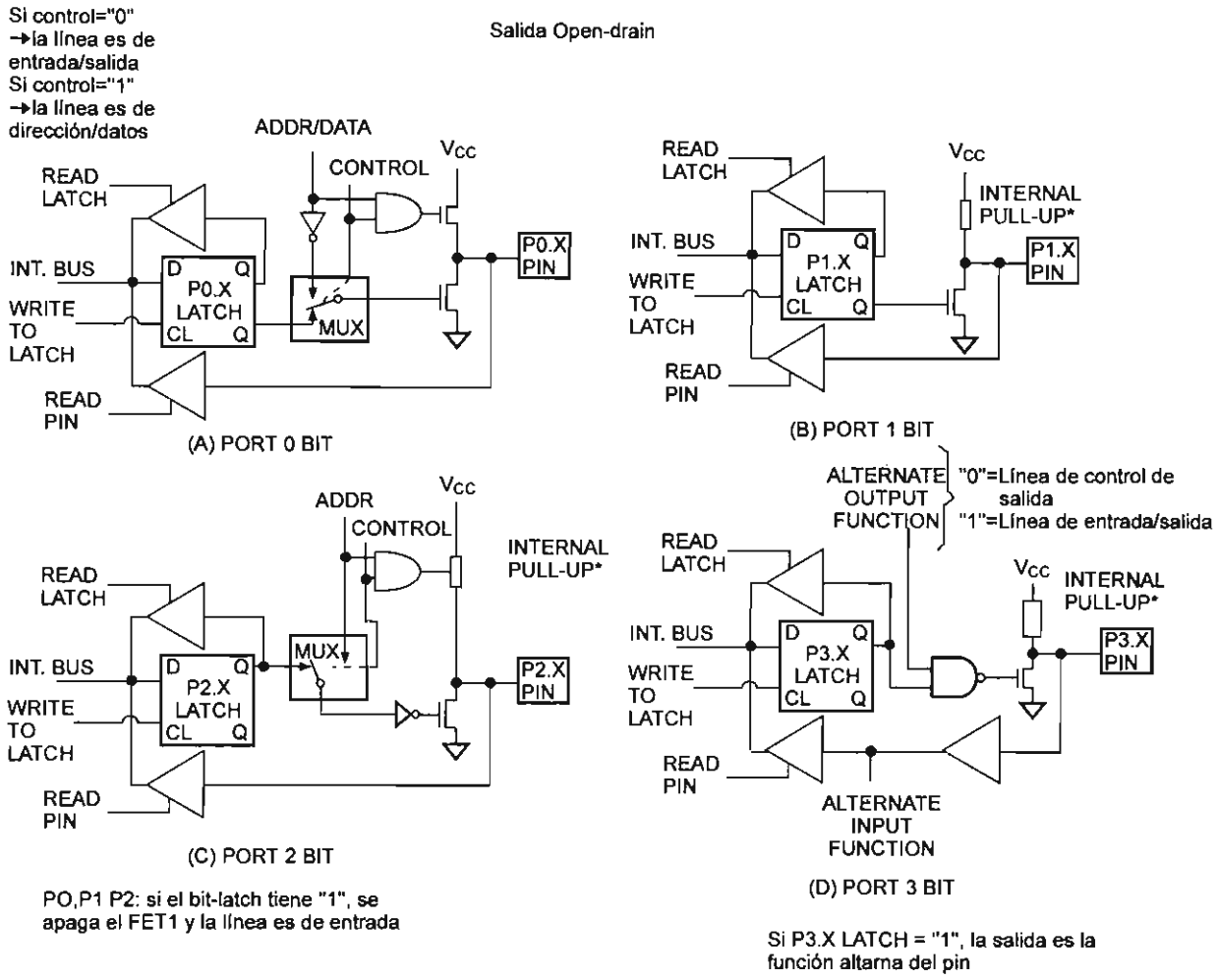


Figura II.11. Diagrama funcional de un bit-batch y un bufer entrada/salida.

La salida Q del flip-flop se coloca en el bus de datos interno en respuesta a una señal de lectura del latch *(read latch)* emitida por la CPU. El nivel presente en el pin del puerto es el que se toma y se coloca en el bus interno en respuesta a una señal de lectura del pin *(read pin)* enviada por la CPU. Algunas instrucciones que leen un puerto activan la señal de read latch y otras la de read pin.

¿Cuáles instrucciones que realizan una lectura a un puerto leen el latch y cuales el pin? Las instrucciones que leen el latch en lugar del pin son las que leen un valor, posiblemente lo cambian y lo reescriben en el latch. Estas instrucciones son del tipo *read-modify-write* y son las que se listan a continuación. Cuando el operando destino es un puerto, o un bit de un puerto, estas instrucciones leen el latch en lugar del pin.

CUADRO II.7. *Instrucciones read-modify-write del 8051*

ANL	AND lógica. Ejemplo: ANL P1, A
ORL	OR lógica. Ejemplo: OR P2, A
XRL	EX-OR lógica. Ejemplo: XRL P3, A
JBC	Salta si el bit = 1 y limpia el bit. Ejemplo: JBC P1.1, LABEL
CPL	Complementa el bit. Ejemplo: CPL P3.0
INC	Incrementa. Ejemplo: INC P2
DEC	Decrementa. Ejemplo: DEC P2
DJNZ	Decrementa y salta si no es cero. Ejemplo: DJNZ P3
MOV PX. Y, C	Mueve el bit de acarreo hacia el bit Y del puerto X
CLR PX. Y	Limpia el bit Y del puerto X
SET PX. Y	Pone el bit Y del puerto X

No es evidente que las últimas tres instrucciones sean del tipo read-modify-write, pero lo son, ya que leen el byte del puerto, modifican los bits direccionados y escriben el nuevo byte en el latch.

La razón por la cual las instrucciones read-modify-write accesan al latch es para evitar una posible mala interpretación del nivel de voltaje en el pin.

Por ejemplo, si se usa un bit de un puerto para manejar la base de un transistor y se escribe un 1 al bit, el transistor encenderá. Si la CPU leyera posteriormente el pin del puerto en lugar de leer el latch del puerto, tomaría el voltaje de la base del transistor y lo interpretaría como un "0". Leyendo el latch en lugar del pin tomaría el valor correcto de "1".

Por otra parte, en la figura anterior se puede observar que los drivers de salida de los puertos 0 y 2 se pueden conmutar a un bus de direcciones (ADDR) y un bus de direcciones/datos (ADDR/DATA) por medio de una señal de control interna para uso en accesos a memoria externa.

Descripción funcional de los puertos de entrada/salida del 8051

El puerto 0

Aquí pueden suceder cuatro eventos:

1. Que la señal CONTROL = "0" (significa que la línea del puerto 0 se está usando como entrada/salida) y que se escriba un "0" al LATCH P0.X, lo cual causa que el FET2 de la etapa de salida se encienda y por lo tanto la línea de salida del puerto sea *open drain* (el FET1 está apagado).
2. Que la señal de CONTROL = "0" y se escriba un "1" al LATCH P0.X, lo que causará que el FET2 de la etapa de salida se apague (el FET1 también está apagado) y el pin del puerto "0" flote, por lo que la línea trabajará como *entrada* de alta impedancia.
3. Que la señal CONTROL = "1" (significa que la línea del puerto 0 se está usando como una línea del bus de direcciones/datos) y la línea ADDR/DATA = "1", lo que causa que se encienda el FET1, el cual se usa como pull-up interno, y se apague el FET2, por lo que a la salida del puerto 0 se presentará un "1".
4. Que la línea ADDR/DATA = "0" y la señal CONTROL = "1", lo que causa que el FET1 se apague y la entrada del FET2 conmute hacia la señal ADDR/DATA = "1" y se encienda el FET2.

Podemos decir entonces que:

- Las líneas del puerto 0 no se pueden usar como líneas de entrada/salida de propósito general al mismo tiempo que se usan como líneas de entrada/salida.
- El FET en el driver de salida del puerto 0 se usa como pull-up sólo cuando la CPU está enviando "1" durante accesos a memoria externa.
- El puerto 0 se considera realmente bidireccional, ya que cuando una de las líneas se configura como entrada, se deberá escribir un "1" a su latch asociado, lo que causa que el pin flote, y cuando se configura como salida se deberá escribir un "0" a este latch, lo cual origina que esta salida sea open drain.
- Cuando la CPU realiza un acceso a memoria externa, escribe 0FFH en el latch asociando al puerto 0, por lo que la información almacenada en este latch *se puede perder*.
- La señal de CONTROL de la lógica interna del puerto 0 sólo se activará cuando la CPU realice un acceso a memoria externa.

El puerto 1

Este puerto tiene pull-up internos y cada una de las líneas se puede usar independientemente como entrada/salida de propósito general. Cuando se escribe un "1" lógico al latch asociado al puerto, se apagarán los FET drivers de la etapa de salida y las terminales de este puerto son forzadas a 1 por los pull-up internos, pero pueden asimismo ser llevados a "0" lógico por una fuente externa, por lo que entonces estos pines funcionarán como entradas, es decir, la carga es quien maneja el nivel en el pin del puerto. Si por el contrario, se escribe un "0" lógico al latch del puerto 1, se encenderán los FET de la etapa de salida y aparecerán "0" en las terminales del puerto 1.

El puerto 2

El funcionamiento de este puerto es similar al del puerto 0, con la diferencia de que el puerto 2 tiene "pull-up" internos fijos y sus líneas únicamente comparten su función con el bus de direcciones, por lo que a este puerto se le considera quasi-bidireccional, debido a que cuando sus líneas se configuran como entradas se deberá escribir 0FFH al latch asociado al puerto y se apagarán los FET de salida pero las terminales no flotan y llevan a "1" lógico su salida gracias a los pull-up internos fijos.

El puerto 3

Este puerto comparte sus funciones con líneas o señales de control del microcontrolador (por ejemplo, $\overline{\text{RD}}$, $\overline{\text{WR}}$, etc.). Si se desea utilizar este puerto como línea de entrada de propósito general, se deberá escribir 0FFH al latch asociado al puerto 3, e internamente la señal ALTERNATE OUTPUT FUNCTION (Función de salida alterna: $\overline{\text{RD}}$, $\overline{\text{WR}}$, etc.) deberá ser "1", lo que causa que los FET de la etapa de salida se apaguen y las líneas de entrada puedan ser manejadas por una fuente externa. Si las líneas del puerto se desean usar como salidas de propósito general, deberá escribir 00H al latch del puerto para encender los FET de salida y que en las líneas del puerto 3 aparezcan "0".

Cuando la línea del puerto 3 se utiliza como línea de entrada, el latch P3.X deberá tener "1" lógico, la señal ALTERNATE OUTPUT FUNCTION será "1" lógico y en la señal ALTERNATE INPUT FUNCTION estará presente el valor del pin P3.X, ya que el FET de salida de la línea del puerto 3 estará apagado y el valor del pin de entrada será manejado por la entrada.

Cuando la línea del puerto 3 se usa como línea de control de salida (por ejemplo, $\overline{\text{WR}}$), el latch P3.X

deberá tener "1" lógico y el valor de la señal ALTERNATE OUTPUT FUNCTION será quien apague o prenda el FET de salida, y por lo tanto será el valor que aparecerá en el pin P3.X.

Nota: Al aplicarle un reset al 8031, en todos los latches de los puertos se escribirá "1". Si a continuación se les escriben "0" (salidas), éstos se podrán reconfigurar nuevamente como entradas escribiéndoles "1".

Acceso a memoria externa

Los accesos a memoria externa son de dos tipos: a memoria externa de programa y a memoria externa de datos. Los primeros usan la señal $\overline{\text{PSEN}}$ (program store enable) como señal de lectura. Los accesos a memoria externa de datos usan las señales $\overline{\text{RD}}$ y $\overline{\text{WR}}$ (funciones alternas de P3.7 y P3.6) para interactuar con la memoria.

Los ciclos fetch de la memoria externa de programa siempre usan una dirección de 16 bits. Los accesos a memoria externa de datos pueden usar direcciones de 16 bits (MOVX @DPTR) o de 8 bits (MOVX @Ri).

Cuando se usa una dirección de 16 bits, el byte alto de la dirección sale por el puerto 2, donde éste permanece durante el ciclo de lectura o escritura. El puerto 2 tiene pull ups que se usan cuando los bits de direcciones son "1". Durante este tiempo el latch del puerto 2 no debe contener forzosamente los "1" de las direcciones, por lo que el SFR del puerto 2 no se modifica. Si el ciclo de acceso a memoria externa no es seguido por otro ciclo de acceso también a memoria externa, el contenido (no modificado) del SFR del puerto 2 reaparecerá en el siguiente ciclo, es decir, los "1" de la dirección que saca el puerto 2 no sustituyen al contenido del SFR del puerto 2.

Si se usa una dirección de 8 bits (MOVX @Ri), el contenido del puerto 2 será el contenido del SFR del puerto 2 durante el ciclo de acceso a memoria externa. Esto facilitara la paginación de memoria.

En cualquier caso, el byte bajo de la dirección es multiplexado en tiempo con el byte del dato en el puerto 0. La señal ADDR/DATA controla ambos FET de los buffers de salida del puerto 0. Entonces, en esta aplicación las terminales del puerto 0 no son salidas open-drain y no requieren pull-up externos. La señal ALE (address latch enable) se deberá usar para capturar el byte de la dirección en un latch externo. El byte de dirección es válido con la transición negativa de ALE. Así entonces, en un ciclo de escritura, el byte del dato aparece en el puerto 0 justo antes de que $\overline{\text{WR}}$ sea activada y permanece ahí hasta después de que $\overline{\text{WR}}$ sea desactivada. En un ciclo de lectura, el byte de entrada es leído del puerto 0 justo antes de que el pulso de lectura sea desactivado.

La señal $\overline{\text{EA}}$ (external access)

La memoria externa de programa se accesa en dos condiciones: cuando la señal $\overline{\text{EA}}$ esté activa, o cuando el programa contador (program contador) contenga un número mayor a 0FFFH (1FFFH para el 8052).(4K) - (8K).

Esto implica que las versiones que no tienen ROM interna deben tener la señal $\overline{\text{EA}}$ a "0" lógico para habilitar o indicar que los 4K bajos (8K para el 8032) de la memoria de programa sean buscados de memoria externa.

Cuando la CPU está ejecutando un programa que se encuentra en memoria externa, los 8 bits del puerto 2 se dedican a una función de salida y no pueden usarse para entrada/salida de propósito general. Durante este tiempo, los drivers del puerto 2 usan los pull ups para enviar los bits que sean "1" del byte alto del programa contador.

La señal PSEN (program store enable)

La señal de lectura para ciclos fetch de memoria externa es PSEN, la cual no se activa para ciclos fetch internos. Esta señal se activa dos veces en cada ciclo fetch (excepto en una instrucción MOVX). PSEN no se activa igual que RD. Un ciclo completo de RD, incluyendo activación y RD desactivación de ALE y, RD, toma 12 periodos de reloj. Un ciclo de PSEN toma 6 periodos de reloj. La figura II.12 muestra la secuencia de ejecución de estos dos tipos de ciclos de lectura.

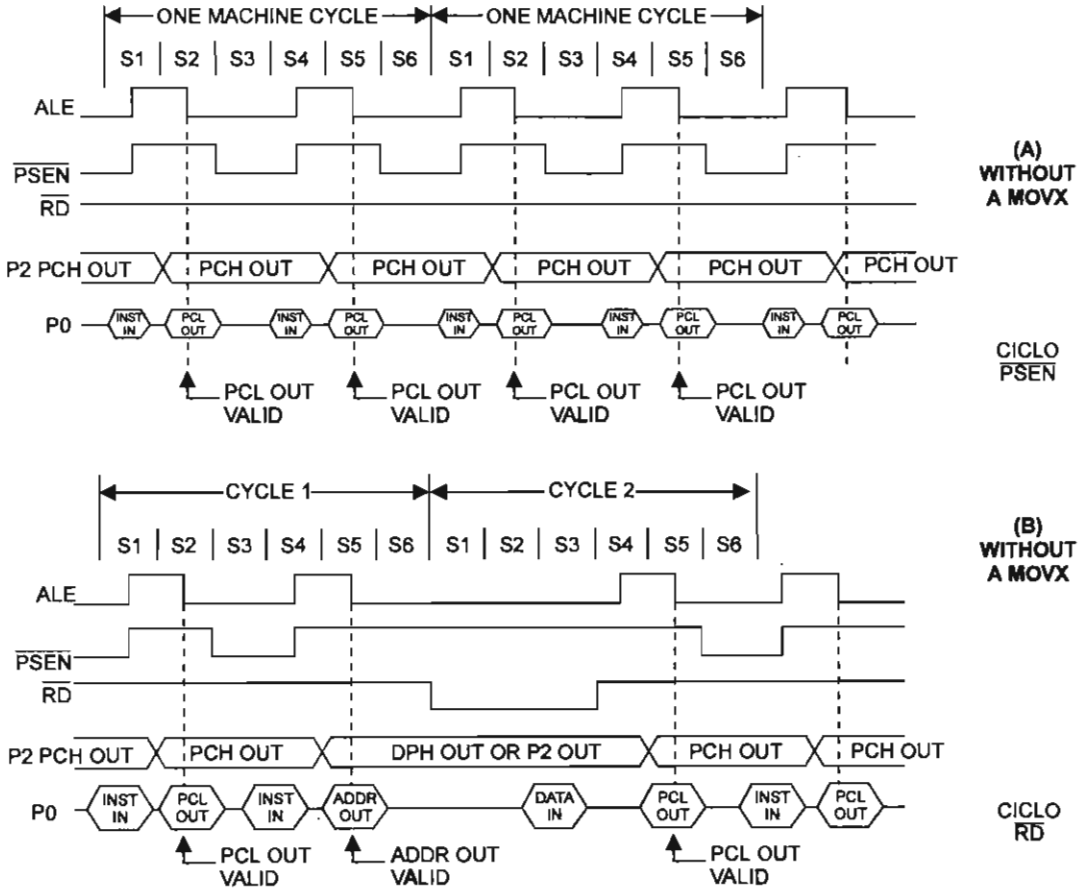


Figura II.12. Activación de la señal PSEN del 8051.

La señal ALE (address latch enable)

La principal función de ALE es proporcionar una señal para latchear el byte bajo de una dirección en un latch externo durante ciclos fetch de memoria externa de programa. Esta señal se activa dos veces cada ciclo de máquina. La única vez en que ALE no se activa es durante un acceso a memoria externa de datos del primer ALE del segundo ciclo de una instrucción MOVX. Así entonces, en cualquier sistema que no use memoria externa de datos, ALE se activa a una razón de 1/6 de la frecuencia del oscilador, y se puede usar como una señal de reloj o temporización de eventos externos.

Por otra parte, en algunas aplicaciones es necesario ejecutar un programa de la misma memoria física usada para almacenar datos. En el 8051 se pueden combinar los espacios externos de memoria de programa y memoria de datos usando una compuerta AND para las señales PSEN y RD. La única

restricción es que la memoria externa sea lo suficientemente rápida para responder a un ciclo de $\overline{\text{PSEN}}$ ya que un ciclo de $\overline{\text{PSEN}}$ es más rápido que uno de $\overline{\text{RD}}$.

Temporizador/contador

El 8051 tiene dos registros tipo temporizador/contador de 16 bits: el temporizador 0 y el 1. El 8052 tiene esos dos y uno más: el temporizador 2. Los tres se pueden configurar para trabajar como temporizadores (temporizador) o contadores (contador) de eventos.

En la función de temporizador, el registro se incrementa cada ciclo de máquina, por lo que se considera como un contador de ciclos de máquina. Ya que un ciclo de máquina consta de 12 periodos del oscilador, la razón de la cuenta es de 1/12 de la frecuencia del oscilador. En la función de contador (contador), el registro se incrementa cada vez que sucede una transición de "1" a "0" lógico en su correspondiente pin de entrada externo: T0, T1 o T2 (en el 8052).

En esta función, la entrada externa se muestrea durante S5P2 de cada ciclo de máquina. Cuando en la entrada existe un "1" lógico y en el siguiente ciclo un "0" lógico, la cuenta se incrementa. La cuenta nueva aparece en el estado S3P1 del ciclo siguiente del que se detectó la transición. Ya que esto toma dos ciclos de máquina (24 periodos del oscilador), la razón máxima de conteo es 1/24 de la frecuencia del oscilador. No existen restricciones en cuanto al ciclo de trabajo de la señal de entrada externa.

El temporizador 0 y el temporizador 1 tienen cuatro modos de operación, uno de los cuales se selecciona por el par de bits (M1, M0) del registro TMOD. Los modos 0, 1 y 2 son los mismos en ambos temporizador/contadores. El modo 3 es diferente. La función de temporizador o contador se selecciona por los bits C/T en el registro TMOD.

El modo 0

Programando cualquier temporizador en modo 0, éste se comportará como un contador de 8 bits con un preescalar de 32 bits. En este modo, el registro del temporizador se configura como un registro de 16 bits. Cuando el conteo pase de todos "1" a todos "0" se activa la bandera de temporizador interrupt flag TF1. En las siguientes figuras se muestra la operación del modo 0 para el temporizador 1.

La entrada de control para el contador es habilitada cuando TR1 = 1 y GATE = 0 o $\overline{\text{INT1}} = 1$ (inicializando GATE = 1 permite que el temporizador sea controlado por la entrada externa $\overline{\text{INT1}}$). TR1 es el bit de control en el registro TCON. GATE está en el TMOD.

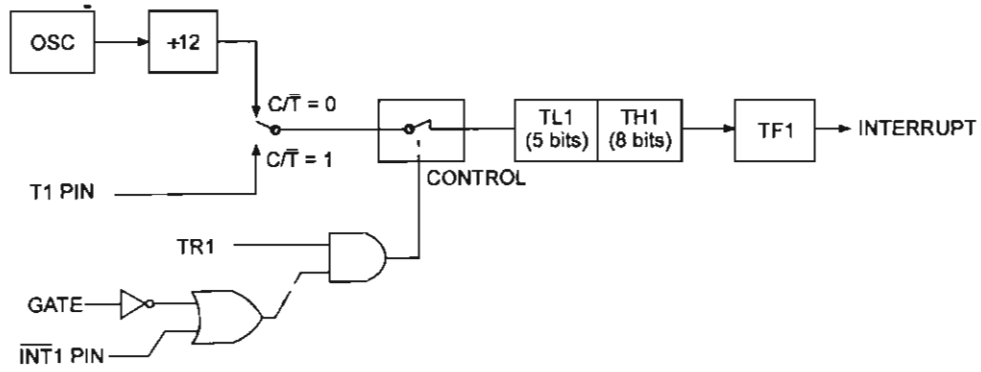


Figura II.13. Operación en el modo 0 de los temporizadores del 8051.

El registro de 13 bits se compone de los 8 bits de TH1 y los 5 bits más bajos de TL1. Los 3 bits más significativos de TL1 son indeterminados y deben ignorarse. Inicializando con "1" lógico el bit run flag (TR1), no serán limpiados estos registros.

El modo 0 funciona igual en el temporizador 0 que en el temporizador 1, únicamente que se debe sustituir TR0, $\overline{\text{TF0}}$ e $\overline{\text{INT0}}$ por sus correspondientes señales en la figura II.13. Hay dos bits diferentes de GATE, uno para el temporizador 1 (TMOD.7) y otro para el 0 (TMOD.3).

El registro TMOD *(timer mode)*

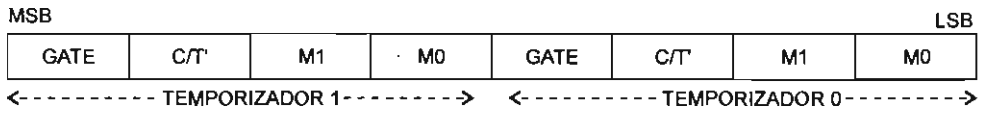


Figura II.14. Registro TMOD.

- GATE: si tiene "1" lógico es una señal de control de disparo que indica que el temporizador/contador es habilitado sólo cuando el pin $\overline{\text{INTx}}$ sea "1" lógico y el pin de control TRx también sea "1" lógico. Cuando GATE = "0" lógico, el temporizador es habilitado, con la condición de que el bit de control TRx sea "1" lógico.
- C/$\overline{\text{T}}$: Selector de temporizador ("0" lógico) o contador ("1" lógico).

CUADRO II.8. *Modo de operación del registro TMOD*

M0	*M1*	*Modo de operación*
0	0	El registro TLx sirve como un preescalar de 5 bits.
0	1	Es un temporizador/contador de 16 bits. THx y TLx están en cascada y no hay preescalar.
1	0	Es un temporizador/contador con autorrecarga de 8 bits. THx contiene un valor que será recargado en TLx cada vez que éste llegue a FFH.
1	1	Temporizador 0: TL0 es un temporizador/contador controlado por los bits de control del temporizador 0 y TH0 es un temporizador de 8 bits controlado únicamente por los del temporizador 1.

El registro TCON *(timer control)*

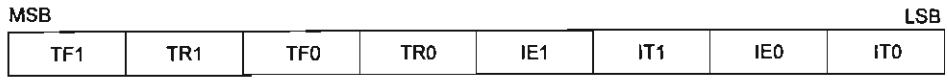


Figura II.15. Registro TCON.

CUADRO II.9. *Bits del registro* TCON

Símbolo	*Posición*	*Nombre y significado*
TF1	TCON.7	Bandera de sobreflujo del temporizador 1. Se activa por hardware cuando el temporizador/contador llega a 00FH. Se limpia por hardware cuando el procesador ejecuta la rutina de atención a la interrupción.
TR1	TCON.6	Bit de control de activación del temporizador 1. Se activa/desactiva por software para encender/apagar al temporizador/contador.
TF0	TCON	Bandera de sobreflujo del temporizador 0. Lo mismo que TF1 pero para el temporizador 0.
TR0	TCON.4	Bit de control de activación del temporizador 0. Lo mismo que TR1 pero para el temporizador 0.
IE1	TCON.6	Bandera de activación de interrupción. Se activa por hardware cuando se detecta una transición activa en el pin interrupción externa. Se limpia cuando el procesador atiende la interrupción.
IT1	TCON.2	Bit de control de tipo de activación de interrupción. Se activa/desactiva por software para especificar si la interrupción externa será disparada por una transición de bajada o un nivel activo bajo, respectivamente.
IE0	TCON.1	Lo mismo que IE1 pero para el temporizador 0.
IT0	TCON.0	Lo mismo que IT1 pero para el temporizador 0.

El modo 1

Es igual que el modo 0, excepto que el registro del temporizador opera con todos los 16 bits.

El modo 2

Este modo configura al registro del temporizador como un contador de 8 bits (TL1) con recarga automática como se muestra en la figura II.16. Cuando TL1 llega a FFH *(overflow)*, activa TF1 y recarga TL1 con el contenido del registro TH1, el cual fue inicializado por software. La recarga no modifica a TH1. Este modo de operación funciona igual en el temporizador (contador 0).

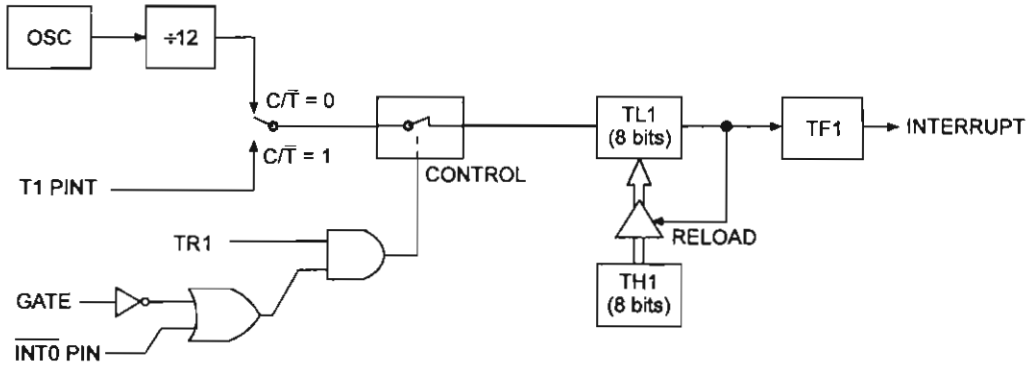


Figura II.16. Operación del modo 2 de los temporizadores del 8051.

El modo 3

Cuando el temporizador 1 opera en modo 3 simplemente detiene su conteo. El efecto es el mismo que inicializar TR1 con "0".

El temporizador 0 en modo 3 establece a TL0 y TH0 como dos contadores separados. La lógica para el temporizador 0 programado en modo 3 es la siguiente:

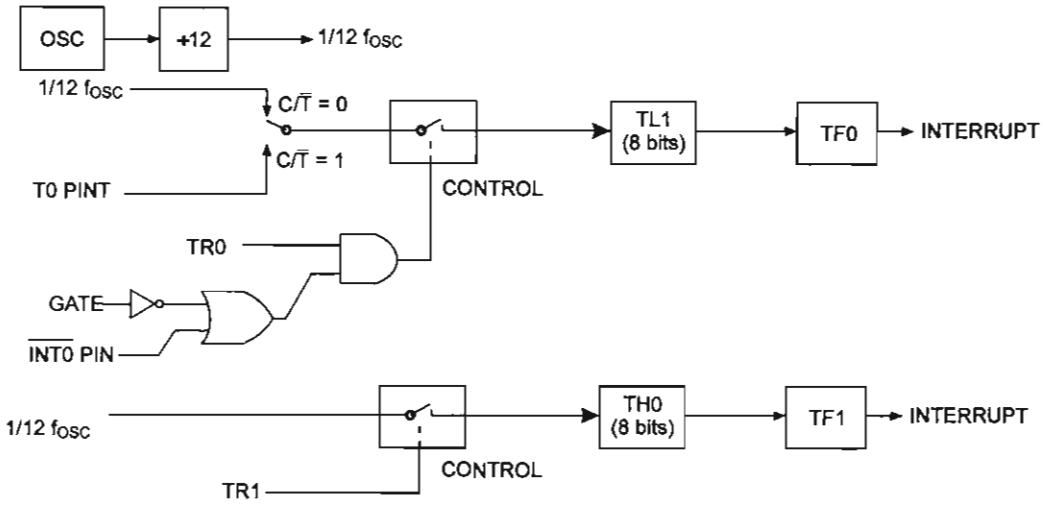


Figura II.17. Operación del modo 3 del temporizador 0 del 8051.

TL0 usa los bits de control del temporizador 0; C/$\overline{T}$, $\overline{GATE}$, TR0, INT0 y TF0. TH0 se fija a funcionar como temporizador (contador de ciclos de máquina) y usa los bits TR1 y TF1 del temporizador 1. Así entonces, TH0 controla ahora la interrupción del temporizador 1.

Este modo fue hecho para aplicaciones que requieren un temporizador o un contador de 8 bits extra. Con el temporizador 0 en modo 3, el 8051 se puede considerar un microcontrolador de tres temporizador/contadores. Así también, cuando el temporizador 0 está en modo 3, el temporizador 1 puede encenderse y apagarse activándolo y desactivándolo para funcionar en modo 3, es decir, metiéndolo o sacándolo del modo 3, o puede continuar usándose sólo generador de *baud rate* para el puerto serie, o en aplicaciones que no requieran interrupciones.

Puerto serie

El puerto serie del 8051 es full dúplex, es decir, puede transmitir y recibir simultáneamente. La recepción es buffereada, o sea que puede comenzar la recepción de un segundo byte antes de que el byte previamente recibido haya sido leído del registro de recepción. (Sin embargo, si el primer byte no ha sido leído cuando el tiempo de recepción del segundo byte se haya completado, uno de los bytes se perderá.) Los registros de transmisión y de recepción se accesan por medio del registro de funciones especiales SBUF. Una escritura a SBUF carga al registro de transmisión, y una lectura a SBUF accesa a un registro de recepción por separado.

El puerto serie puede operar en cuatro modos:

Modo 0

El dato serie entra y sale por medio del pin RxD. Por el pin TxD sale el reloj utilizado para llevar a cabo los corrimientos de cada uno de los bits que componen al dato a transmitir en forma serie (el *shift clock*). Se transmiten/reciben 8 bits : los 8 bits del dato, primero el bit menos significativo. El baud rate es fijo a 1/12 de la frecuencia del oscilador.

Modo 1

Se transmiten (por TxD) o se reciben (por RxD) 10 bits: un bit de start (0), 8 bits del dato (primero el LSB) y uno de stop (1). En la recepción el bit de stop se transfiere al bit RB8 del registro de funciones especiales SCON. El baud rate es variable.

Modo 2

Se transmiten (por TxD) o se reciben (por RxD) 11 bits: un bit de start (0), 8 bits del dato (primero el LSB), un noveno bit programable y uno de stop (1). En la transmisión, al noveno bit del dato (el bit TB8 del SCON) se le puede asignar el valor de 0 o 1. También se le puede asignar, por ejemplo, el valor del bit de paridad (P en el PSW), que deberá entonces ser transferido a TB8. En la recepción, el noveno bit del dato recibido pasa al bit RB8 del registro de funciones especiales SCON, mientras se ignora el bit de stop. El baud rate es programable a 1/32 o 1/64 de la frecuencia del oscilador.

Modo 3

Se transmiten (por TxD) o se reciben (por RxD) 11 bits: un bit de start (0), 8 bits del dato (primero el LSB), un noveno bit programable del dato y un bit de stop (1). Este modo es igual que el modo 2, excepto que en el modo 3 el baud rate es variable.

En todos los modos, la transmisión se inicia por una instrucción que usa al SBUF como registro destino. La recepción se inicia en el modo 0 por la condición RI = 0 y REN = 1. La recepción se inicia en los otros modos con la llegada del bit de start siempre y cuando REN = 1.

Comunicación mutliprocesador

Los modos 2 y 3 tienen la característica de que se pueden utilizar para la comunicación multiprocesador. En estos modos se reciben 9 bits del dato. El noveno bit pasa al bit RB8 y a continuación viene el bit de stop. El puerto serie se puede programar para que cuando se reciba el bit de stop del dato cuyo noveno bit es "1", se active la interrupción del puerto serie sólo si RB8 = 1. Esta característica se habilita inicializando a "1" lógico el bit SM2 del SCON. Una manera de usar esta característica en sistemas multiprocesador es la siguiente:

Cuando el procesador maestro quiere transmitir un bloque de datos a uno de varios esclavos, primero envía un byte de dirección que identifica al esclavo destino. Un byte de dirección difiere de un byte de dato en que el noveno bit es "0" en un byte de dato. Con SM2 = 1, ningún esclavo será interrumpido por un byte de dato. Un byte de dirección interrumpirá a todos los esclavos, para que cada esclavo pueda examinar el dato recibido y ver si es su dirección. El esclavo direccionado tendrá que limpiar su bit SM2 y prepararse para recibir los bytes de datos que llegarán. Los esclavos que no

fueron direccionados dejarán sin afectar ("1" lógico) su bit SM2 y continuarán su actividad, ignorando los bytes de datos. SM2 no tiene efecto en el modo 0, y en el modo 1 se puede usar para validar el bit de stop. En la recepción del modo 1, si SM2 = 1, la interrupción de la lógica de recepción no será activada si el bit de stop no es válido.

El registro de control del puerto serie

El registro de control y estado del puerto serie es el registro de funciones especiales SCON, el cual se mostrará a continuación. Este registro contiene no sólo los bits para el modo de selección, sino también el noveno bit del dato a transmitir o recibido (TB8 y RB8), y los bits de interrupción del puerto serie (TI y RI).

El registro de control del puerto serie (SCON)

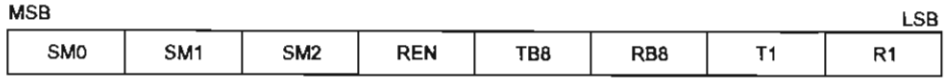


Figura II.18. Registro de control del puerto serie.

SM0 y SM1 especifican el modo del puerto serie como se indica a continuación:

CUADRO II.10. *Modos de operación del puerto serie*

SM0	*SM1*	*Modo*	*Descripción*	*Baud rate*
0	0	0	Registro de corrimiento	Fosc/12
0	1	1	UART de 8 bits	Variable
1	0	2	UART de 9 bits	Fosc/64 o fosc/32
1	1	3	UART de 9 bits	Variable

SM2 habilita la comunicación multiprocesador en los modos 2 y 3. En los modos 2 o 3, si SM2 es puesto a "1", RI no se activará si el noveno bit del dato recibido (RB8) es "0". En el modo 1, si SM2 = "1" entonces RI no se activará si el bit de stop recibido no es válido. En el modo 0, SM2 debe ser "0".

REN habilita la recepción serie. Se pone a "1" por software para habilitar la recepción.

TB8 es el noveno bit del dato que se transmitirá en los modos 2 y 3. Se activa o desactiva por software.

RB8 es el noveno bit del dato recibido en los modos 2 y 3. En el modo 1, si SM2 = 0, RB8 es el bit de stop recibido. En el modo 0, RB8 no se usa.

TI es la bandera de interrupción de la lógica de transmisión. Se activa por hardware al terminar de transmitirse el octavo bit en el modo 0, o al inicio del bit de stop en los otros modos. Se debe limpiar por software.

RI es la bandera de interrupción de la lógica de recepción. Se activa por hardware al terminar de recibirse el octavo bit en el modo 0, o a la mitad del bit de stop en los otros modos (véase también SM2). Se debe limpiar por software.

Baud rate

El baud rate en el modo 0 está dado por la frecuencia del oscilador/12.

El baud rate en el modo 2 depende del valor del bit SMOD del registro PCON. Si SMOD = "0", el baud rate es 1/64 de la frecuencia del oscilador. Si SMOD = "1", el baud rate es 1/32 de la frecuencia del oscilador:

$$\text{baud rate} = 2^{[SMOD]}/64 \times (\text{frecuencia del oscilador})$$

En el 8051, los baud rates en los modos 1 y 3 son determinados por el temporizador 1. En el 8052, esos baud rates pueden ser determinados por el temporizador 1 o por el temporizador 2 o por ambos (uno para el transmisor y otro para el receptor).

Uso del temporizador 1 para generar baud rates

Cuando el temporizador 1 se usa como generador de baud rates, los baud rates en los modos 1 y 3 se determinan por el temporizador 1 y el valor de SMOD:

$$\text{baud rate} = 2^{[SMOD]}/32 \times (\text{temporizador 1 overflow rate}) \text{ para modos 1 y 3}$$

La interrupción del temporizador 1 debe deshabilitarse para esta aplicación. El temporizador mismo puede configurarse como temporizador o como contador y puede operar en cualquiera de sus tres modos. En las aplicaciones típicas, se configura como temporizador en el modo de autorrecarga (el nibble alto de TMOD = 0010B). En este caso, el baud rate esta dado por:

$$\text{baud rate} = 2^{[SMOD]}/32 \times \text{frecuencia del oscilador}/12 \times [256-(TH1)]$$

Así entonces, se pueden obtener baud rates muy pequeños habilitando la interrupción del temporizador y configurándolo para trabajar como temporizador de 16 bits (el nibble alto de TMOD = 0001B), y usando la interrupción para realizar la recarga de 16 bits por software.

El cuadro II.11 lista los baud rates más comúnmente usados y cómo pueden obtenerse con el temporizador 1.

CUADRO II.11. *Baud rates comúnmente usados en el 8051*

		SMOD	*Temporizador 1*		
Baud rate	*Fosc*		$C/\overline{T}$	*Modo*	*Valor recargable*
Modo 0; máx: 1MHz	12 MHz	X	X	X	X
Modo 2; máx: 375K	12 MHz	1	X	X	X
Modos 1, 3: 62.5K	12 MHz	1	0	2	FFH
19.2K	11.059 MHz	1	0	2	FDH
9.6K	11.059 MHz	0	0	2	FDH
4.8K	11.059 MHz	0	0	2	FAH
2.4K	11.059 MHz	0	0	2	F4H
1.2K	11.059 MHz	0	0	2	E8H
137.5	11.059 MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

Interrupciones

El 8051 tiene cinco fuentes de interrupción y el 8052 seis. Estas fuentes de interrupción se muestran en la figura II.19.

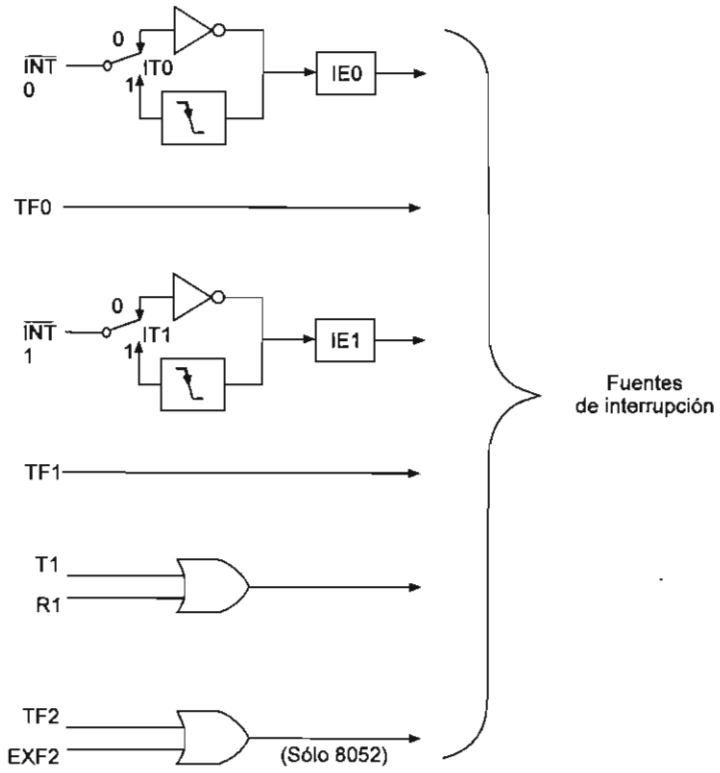


Figura II.19. Fuentes de interrupción del 8051.

Las interrupciones externas $\overline{\text{INT0}}$ e $\overline{\text{INT1}}$ pueden activarse por nivel o por transición, dependiendo de los bits IT0 e IT1 del registro TCON. Las banderas que se ven afectadas por estas interrupciones son los bits IE0 e IE1 del registro TCON. Cuando se genera una interrupción externa, la bandera correspondiente (IE0-IE1) se limpia por hardware cuando se ejecuta la rutina de servicio sólo si la interrupción fue activada por transición. Si la interrupción fue activada por nivel, la fuente que solicita dicha interrupción debe controlar (limpiar) la bandera de interrupción.

Las interrupciones del temporizador 0 y del temporizador 1 son generadas por TF0 y TF1. Estas últimas dos banderas se limpian por hardware cuando se ejecuta la rutina de servicio.

La interrupción del puerto serie se genera por una OR lógica de RI y TI. Ninguna de estas banderas se limpia por hardware cuando se ejecuta la rutina de servicio. La rutina de servicio tendrá que determinar si RI o TI generó la interrupción y el bit tendrá que ser limpiado por software.

Todos los bits que generan interrupciones pueden ser activados o desactivados por software, con el mismo resultado que si fueran afectados por hardware. Esto es, se pueden generar interrupciones o se pueden cancelar interrupciones pendientes por software. Cada fuente de interrupción puede habilitarse o deshabilitarse activando o desactivando un bit del registro de funciones especiales IE. Este registro contiene también una deshabilitación global, EA, el cual deshabilita todas las interrupciones a la vez.

Registro IE (interrupt enable)

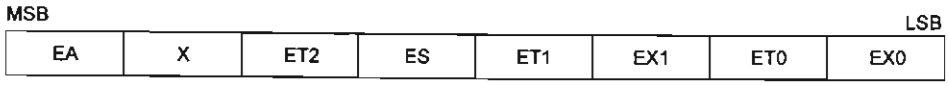


Figura II.20. Bits del registro IE.

CUADRO II.12. *Bits del registro IE*

Símbolo	*Función*
EA	Deshabilita las interrupciones. Si EA = 0, no se reconoce ninguna interrupción. Si EA = 1, cada fuente de interrupción se habilita o deshabilita individualmente.
X	Reservado.
ET2	Habilita o deshabilita la interrupción de sobreflujo o captura del temporizador 2
ES	Habilita o deshabilita la interrupción del puerto serie.
ET1	Habilita o deshabilita la interrupción de sobreflujo del temporizador 1.
EX1	Habilita o deshabilita la interrupción externa 1.
ET0	Habilita o deshabilita la interrupción de sobreflujo del temporizador 0.
EX0	Habilita o deshabilita la interrupción externa 0.

Estructura de niveles de prioridad

Cada fuente de interrupción se puede programar a uno de dos niveles de prioridad, inicializando a "1" o limpiando un bit del registro IP. Una interrupción con nivel de prioridad bajo puede ser interrumpida por otra de nivel de prioridad alto, pero no por otra de nivel de prioridad bajo. Una interrupción de prioridad alta no puede ser interrumpida por otra fuente de interrupción.

Si se reciben simultáneamente dos solicitudes de diferente nivel de prioridad, se atiende la solicitud de nivel alto.

Si se reciben solicitudes del mismo nivel de prioridad de manera simultánea, se aplica una segunda estructura de prioridad, que es la siguiente:

	Fuente	*Nivel de prioridad*
1	IE0	(el más alto)
2	TF0	
3	IE1	
4	TE1	
5	RI + TI	
6	TF2 + EXF2	(el más bajo)

Nótese que esta estructura de niveles de prioridad sólo se usa para solicitudes simultáneas del mismo nivel de prioridad.

Registro IP (interrupt priority register)

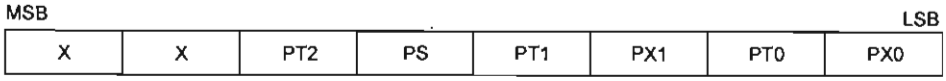


Figura II.21. Registro IP.

CUADRO II.13. *Bits del registro IP*

Símbolo	*Posición*	*Función*
—	IP.7	Reservado.
—	IP.6	Reservado.
PT2	IP.5	Define el nivel de prioridad parar la interrupción del temporizador 2. PT2 = 1 la programa al nivel más alto.
PS	IP.4	Define el nivel de prioridad parar la interrupción del puerto serie. PS = 1 la programa al nivel más alto.
PT1	IP.3	Define el nivel de prioridad parar la interrupción del temporizador 1. PT1 = 1 la programa al nivel más alto.
PX1	IP.2	Define el nivel de prioridad parar la interrupción externa 1. PX1 = 1 la programa al nivel más alto.
PT0	IP.1	Define el nivel de prioridad parar la interrupción del temporizador 0. PT0 = 1 la programa al nivel más alto.
PX0	IP.0	Define el nivel de prioridad parar la interrupción externa 0. PX0 = 1 la programa al nivel más alto.

Manejo de interrupciones

Las banderas de interrupción se muestrean en el estado S5P2 de cada ciclo de máquina. Si en el siguiente ciclo de máquina está activada alguna de las banderas de interrupción, el sistema de interrupciones genera una instrucción del tipo LCALL a una rutina de servicio correspondiente, a menos que sea bloqueada por una de las siguientes condiciones:

1. Se está atendiendo a una interrupción de igual o más alta prioridad.
2. El ciclo donde se está muestreando la interrupción no es el ciclo final de la ejecución de la instrucción que está ejecutándose.
3. La instrucción que se está ejecutando es RETI o cualquier acceso a los registros IE o IP.

La condición 2 asegura que la ejecución de la instrucción que está realizando la CPU se complete antes de atender a la interrupción, y la condición 3 asegura que se ejecute al menos una instrucción más antes de atender a la interrupción.

Si una instrucción no es atendida en virtud de una de las condiciones anteriores, y si la bandera de la interrupción no está activada cuando la condición es removida, entonces la interrupción no será atendida.

Rutina: _____ Rutina: RETI

_____

_____

_____

_____

_____

RETI

Diagrama de tiempo de la respuesta a una interrupción

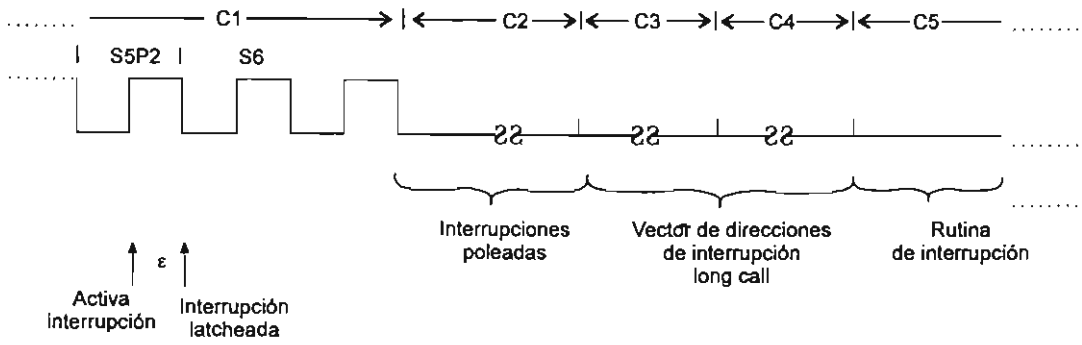


Figura II.22. Respuesta a una solicitud de interrupción en el 8051.

1. Se activa la interrupción.
2. Se latchea la interrupción.
3. Se vuelven a muestrear las interrupciones.
4. LCALL (long call) a la rutina de la interrupción.
5. Se ejecuta la rutina de interrupción.

Cuando el procesador reconoce una solicitud de interrupción, salva el contenido del programa contador en el tope del stack (no salva el PSW) y lo recarga con una dirección que depende de la fuente de interrupción, como se muestra a continuación:

CUADRO II.14. *Vectores de interrupción en el 8051*

Fuente	*Dirección del vector*
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI + TI	0023H
TF2 + EXF2	002BH

Operación single-step

La estructura de interrupciones del 8051 permite la ejecución "paso a paso" *(single-step)* con un *overhead* muy pequeño en el software.

Hemos dicho anteriormente que una solicitud de interrupción no será atendida mientras se esté atendiendo a una de igual prioridad o que se esté ejecutando un RETI, lo cual implica que cuando se está ejecutando una rutina de interrupción, ésta no puede volver a ejecutarse hasta que se ejecute al menos una instrucción del programa interrumpido.

Una forma de aprovechar esta característica es usar una de las interrupciones externas (por ejemplo $\overline{\text{INT0}}$) activada por nivel para implementar la operación single-step.

La rutina de servicio para la interrupción deberá terminar con el siguiente código:

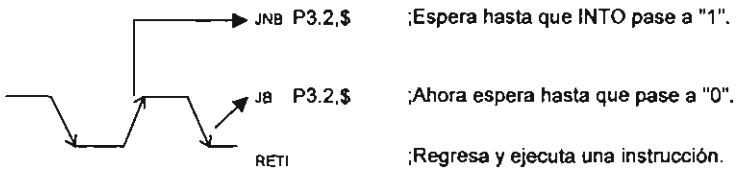


Figura II.23. Operación *single step*.

Si el pin $\overline{\text{INT0}}$ (P3.2) es llevado a "0", la CPU pasa a la rutina de servicio de la interrupción externa 0 y permanece ahí hasta que $\overline{\text{INT0}}$ pase a "1". A continuación se ejecuta el RETI y regresa al programa ejecutando una instrucción del mismo, para re-entrar inmediatamente a la rutina de la interrupción externa 0, por lo que se ejecuta una instrucción del programa cada vez que P3.2 es pulsada.

El reset

El pin RST es la entrada a un schmitt trigger y se activa manteniendo en "1" este pin por al menos 2 ciclos de máquina (24 periodos del oscilador). La CPU responde configurando ALE y $\overline{\text{PSEN}}$ como entradas. El reset interno se ejecuta en el segundo ciclo en el que RST está en "1" y se repite cada ciclo hasta que RST pase a "0".

Los registros internos se inicializan en los siguientes valores:

CUADRO II.15. *Valores de inicialización de los registros del 8051*

Símbolo	*Nombre*	*Valor en binario*
*ACC	Acumulador	00000000
*B	Registro B	00000000
*PSW	Palabra de estado de programa	00000000
SP	Stack pointer	07H
DPTR	Data pointer (consiste en DPH y DPL)	00000000 00000000
*P0	Puerto 0	11111111
*P1	Puerto 1	11111111
*P2	Puerto 2	11111111
*P3	Puerto 3	11111111
*IP	Control de prioridades de interrupciones	8051 XXX00000 8052 XX000000
*IE	Control de habilitación de interrupciones	8051 0XX00000 8052 0X000000
TMOD	Control de modo temporizador/contador	00000000
*TCON	Control temporizador/contador	00000000
+*T2CON	Control 2 temporizador/contador	00000000
TH0	Temporizador/contador 0 (byte alto)	00000000

CUADRO II.15 *(Concluye)*

Símbolo	*Nombre*	*Valor en binario*
TL0	Temporizador/contador 0 (byte bajo)	00000000
TH1	Temporizador/contador 1 (byte alto)	00000000
TL1	Temporizador/contador 1 (byte bajo)	00000000
+TH2	Temporizador/contador 2 (byte alto)	00000000
+TL2	Temporizador/contador 2 (byte bajo)	00000000
+RCAP2H	Registros de cap. temporizador/contador 2 (byte alto)	00000000
+PCAP2L	Registros de cap. temporizador/contador 2 (byte bajo)	00000000
*SCON	Control del puerto serie	00000000
SBUF	Buffer del puerto serie	Indeterminado
PCON	Control de potencia	HMOS 0XXXXXXX CHMOS 0XXX0000

La RAM interna no se afecta por el reset.

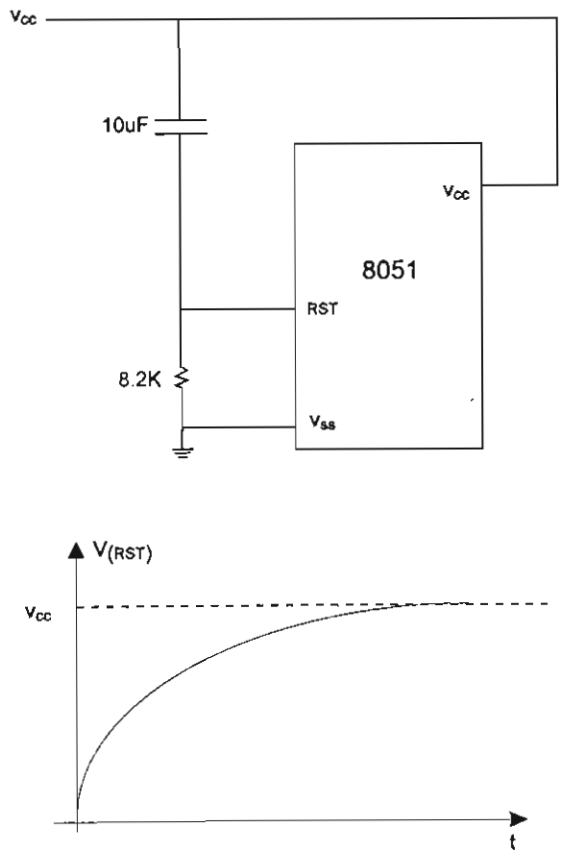


Figura II.24. Circuito de reset sugerido por el fabricante.

Modos de reducción de consumo de energía en las versiones CHMOS

Las versiones CHMOS tienen dos modos: *idle* y *power down*. La circuitería interna que implementa estas características se muestra en la figura II.25.

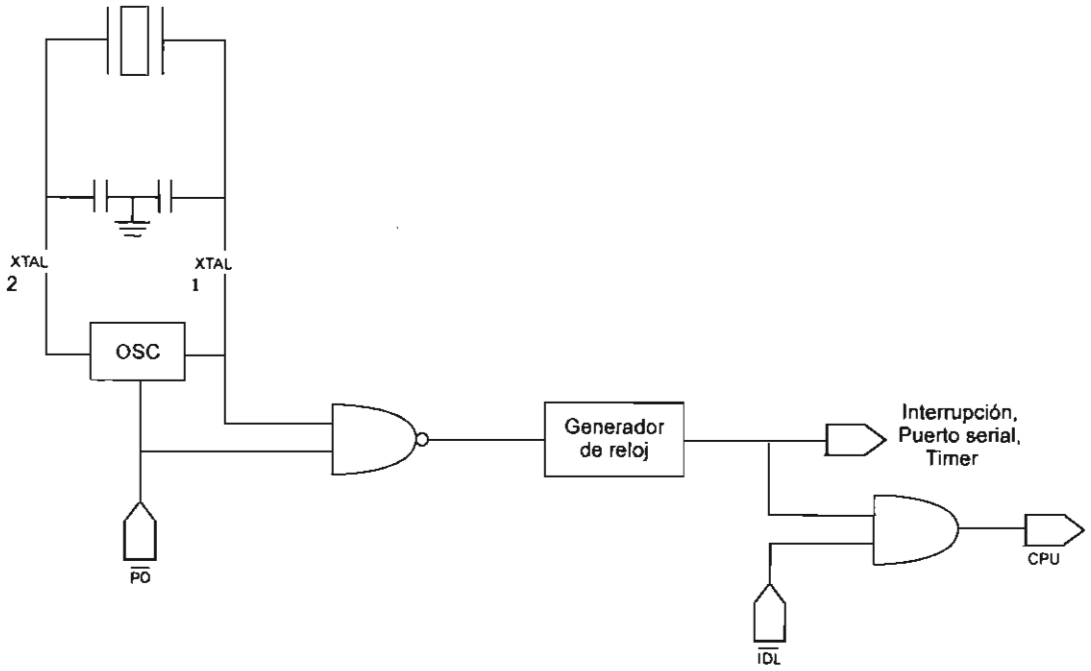


Figura II.25. Modo Idle y power down en el 8051.

En el modo idle (IDL = 1), el oscilador continúa trabajando y la lógica de interrupciones, temporizadores y puerto serie operan normalmente, pero la señal de reloj deja de ser suministrada a la CPU.

En el modo power down (PD = 1), el oscilador es congelado.

Tanto el modo idle como el modo power down se activan en el registro PCON, cuya dirección es la 87H y su contenido se muestra a continuación:

El registro PCON (power control)

MSB							LSB
SMOD	—	—	—	GF1	GF0	PD	IDL

Figura II.26. Registro PCON.

NOTA: Si se escribe 1 en PD e IDL al mismo tiempo, PD toma precedencia. El valor que toma PCON con la activación del reset es (0XXX0000).

CUADRO 11.16. *Bits del registro* PCON

Símbolo	*Posición*	*Nombre y función*
SMOD	PCON.7	Es el bit de doble baud rate. Cuando está activado el temporizador 1 se usa como generador de baud rate y el puerto serie trabaja en los modos 1, 2 o 3.
—	PCON.6	Reservado.
—	PCON.5	Reservado.
—	PCON.4	Reservado.
GF1	PCON.3	Bit para bandera de propósito general.
GF0	PCON.2	Bit para bandera de propósito general.
PD	PCON.1	Bit de modo power down bit. Colocando este bit en "1" activa la operación power down.
IDL	PCON.0	Bit de modo idle. Colocando este bit en "1" activa la operación en modo idle.

Modo IDLE

Una instrucción que coloca a "1" el bit PCON.0 causa que éste sea la última instrucción que se ejecute antes de pasar al modo idle. En este modo se le deja de suministrar señal de reloj a la CPU, pero el estado de la CPU se conserva: SP, PC, PSW, acumulador y el contenido de todos los registros restantes. Las terminales de los puertos conservan su valor y ALE y $\overline{\text{PSEN}}$ toman el valor de "1" lógico.

Hay dos formas para terminar el idle.

1) Se activa cualquier interrupción habilitada, lo cual causa que se limpie el bit PCON.0. Los bits de banderas GF0 y GF1 se pueden usar para indicar si una interrupción ocurrió durante una operación normal o durante un idle. Por ejemplo, una instrucción que activa el idle también puede colocar en "1" uno o ambos de estos bits, de manera que cuando el idle sea terminado por una interrupción, la rutina de servicio puede examinar estas banderas.

2) Con un reset por hardware, el cual debe de estar activo durante al menos dos ciclos de máquina (el oscilador continúa funcionando).

Modo power down

Una instrucción que activa el bit PCON.1 causa que ésta sea la última antes de pasar al modo power down.

En este modo el oscilador interno deja de funcionar. Con el reloj "congelado" todas las funciones se detienen, pero la memoria RAM interna y los registros conservan sus valores. Las terminales de los puertos también mantienen el valor que tenían, pero ALE y $\overline{\text{PSEN}}$ pasan a "0" lógico.

La única forma de salir de este modo es con un reset por hardware, lo cual redefine el contenido de todos los registros pero no modifica el contenido de la RAM interna.

En el modo power down, el valor de Vcc se puede reducir para disminuir el consumo de energía, pero se debe tener cuidado de no reducir Vcc antes de que el modo power down sea invocado. Vcc debe tomar su valor de nivel normal antes de que este modo termine.

El reset que termina el power down también libera al oscilador. El reset no debe activarse antes de que Vcc tome su valor normal y debe de estar activado el tiempo suficiente para permitir que el oscilador arranque y se estabilice (normalmente menos de 10 mseg).

CAPÍTULO III

EL MICROPROCESADOR MC68000

El microprocesador MC68000 fue el primero de 16 bits de Motorola; apareció después del 8086 de Intel y del Z8000 de Zilog, también de 16 bits. No es compatible en software con los microprocesadores de 8 bits de Motorola.

COMPARACIÓN DEL MC68000 CON EL 8086 Y EL Z8000

El MC68000 realiza el ciclo fetch de una instrucción decodificando y ejecutando las dos instrucciones anteriores. El 8086 también lo hace con un esquema pipe-line pero usando una cola de 6 bytes. El Z8000 sólo lo hace en algunas condiciones.

El 8086 y el Z8000 pueden trabajar configurados en un modo simple (mínimo) o en un modo complejo (máximo), el 8086 multiplexando funciones en una misma terminal. Estas terminales tienen una función en modo mínimo y otra en modo máximo. Hay dos versiones del Z8000: para aplicaciones complejas el Z8001 y para aplicaciones simples el Z8002. El MC68000 es un circuito de 64 terminales, por lo que no hay que multiplexar funciones en una misma terminal y puede trabajar en modo máximo o complejo sin modificar el hardware.

El MC68000 tiene circuitería interna para manejar el arbitraje de bus en sistemas multiprocesador. El 8086 y el Z8000 tienen circuitería similar. El MC68000 puede accesar hasta 16 Mb con su bus de direcciones de 24 líneas y hasta 64 Mb usando líneas llamadas de código de función. El 8086 puede direccionar hasta 1 Mb, en tanto el Z8000 puede direccionar hasta 64 Kb de manera directa y el Z8001 hasta 48 Mb usando registros de segmento.

El MC68000 puede trabajar en dos modos: modo supervisor o modo usuario. En modo supervisor se pueden ejecutar algunas instrucciones llamadas privilegiadas.

Existen apuntadores de pila o stack pointers separados para cada modo, por lo que el software del sistema (que se ejecuta en modo supervisor) se puede separar de los programas de aplicación (ejecutados en modo usuario). El modo supervisor es similar al modo system del Z8000 y el modo usuario al modo normal del Z8000. El 8086 no tiene modos de operación equivalentes.

El MC68000 tiene 17 registros de 32 bits, de los cuales 8 son registros de datos que se pueden accesar como registros de 8, 16 o 32 bits, y 9 son registros de direcciones que incluyen a los dos stack pointers. Los registros de dirección se pueden accesar como registros de 16 o 32 bits. Todos los registros pueden funcionar como registros índice. El 8086 sólo tiene 4 registros de datos de 16 bits y 3 registros índice de 16 bits. El Z8000 también tiene registros de 16 bits, los cuales pueden agruparse en parejas para formar registros de 32 bits. El 8086 tiene únicamente 4 registros de 16 bits y 3 registros índice de 16 bits.

El MC68000 tiene un bus de datos separado del bus de direcciones (es de 64 terminales). El 8086 y el Z8000 son más pequeños físicamente y el bus de datos está multiplexado con algunas líneas del bus de direcciones.

ARQUITECTURA INTERNA

Registros del MC68000

Tiene 17 registros de datos y direcciones de 32 bits cada uno, un contador de programa o program counter (PC) de 32 bits (de los cuales sólo usa 24) y un registro de estado o de *status* de 16 bits.

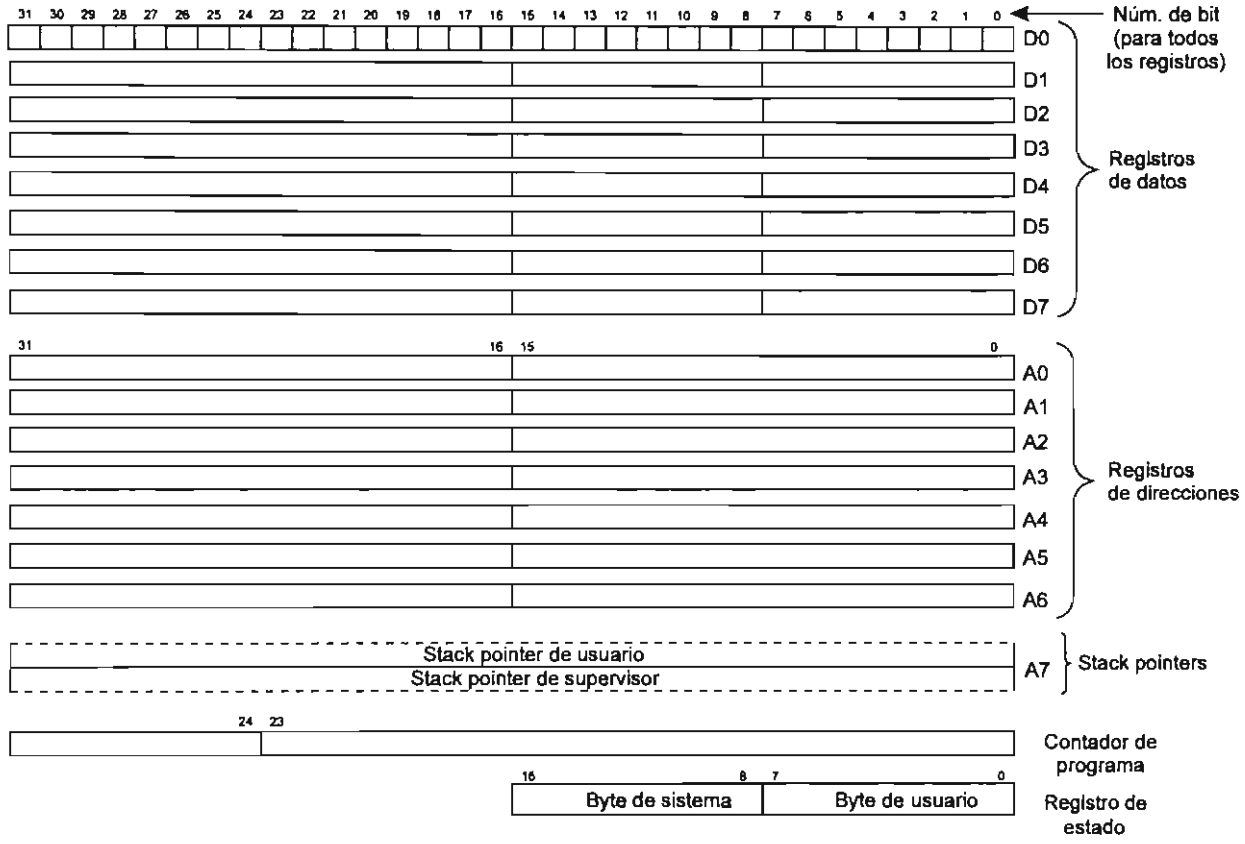


Figura III.1. Registros del MC68000.

Registros de datos

Se pueden accesar como registros de 8 bits, registros de una palabra de 16 bits o de palabras dobles *(long words)* de 32 bits. En la figura III.2 se muestran los diferentes tipos de operandos en los registros de datos.

Cuando un registro de datos se usa como operando fuente o destino sólo se modifica la parte menos significativa indicada en la operación y los bits restantes no se alteran. Por ejemplo, en la instrucción ASL (corrimiento aritmético a la izquierda), usando un operando de 8 bits, sólo se hará el corrimiento usando los 8 bits menos significativos (0-7) del registro de datos.

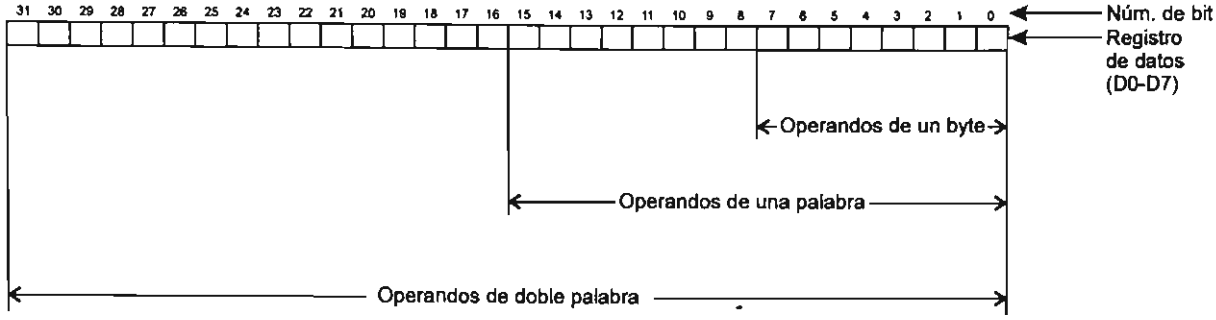


Figura III.2. Tipos de operandos de los registros de datos.

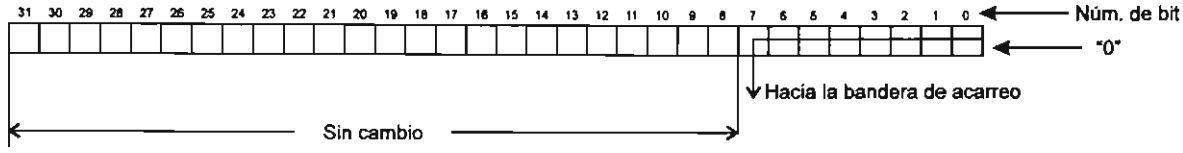


Figura III.3. Registro de datos usado para corrimiento.

Los registros de datos se pueden usar como fuente o destino o como registros índice o contadores en instrucciones.

Registros de direcciones

El MC68000 tiene 7 registros de direcciones (A0-A6) que pueden manejar operandos de 16 bits (palabras) o de 32 bits (long words-palabras dobles). Si se especifica un operando fuente contenido en uno de estos registros se usan los 16 bits menos significativos (los 16 bits más significativos no se afectan) o todos los 32 bits.

Si el operando destino es uno de estos registros, todo el contenido del registro es afectado, por lo que si el operando destino es una palabra para un registro de direcciones, automáticamente la palabra es expandida a un operando signado de 32 bits antes de cargarse en el registro.

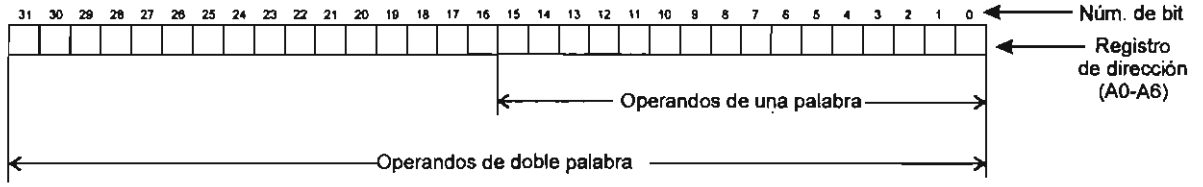


Figura III.4. Registro de direcciones.

Todos los registros de datos y de direcciones son de 32 bits y todos son de propósito general, al igual que en el Z8000.

Los únicos registros dedicados son el apuntador a la pila o stack pointer (registro A7, supervisor y usuario), el contador de programa (program counter) y el registro de estado (o de *status*).

Stack pointer

El bit S del registro de estado determina el modo de operación del MC68000. El modo supervisor generalmente se usa para diseñar sistemas operativos y el modo usuario para programas de aplicación.

El MC68000 tiene un grupo de instrucciones privilegiadas que sólo se pueden ejecutar en modo supervisor. Tanto el modo supervisor como el modo usuario tienen stack pointer separados (registro de dirección A7), de tal forma que cuando está trabajando en modo supervisor no se puede usar el stack pointer de usuario, y cuando está en modo usuario no se puede usar el stack pointer de supervisor. El área de stack es llenada de la parte alta de memoria hacia la parte baja en ambos modos.

Al llamar a una subrutina se requieren 4 bytes (2 palabras) de memoria para salvar el contenido del program counter (32 bits) de la manera mostrada en la figura III.5.

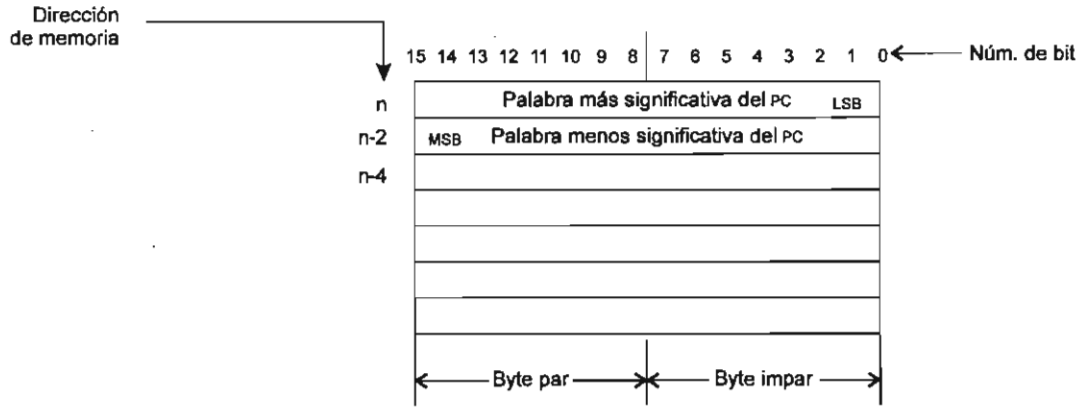


Figura III.5. Almacenamiento en el registro stack pointer.

Un dato almacenado en el stack siempre se escribe desde el inicio de una palabra, es decir, desde una localidad con dirección par.

Cuando se almacenan datos de 1 byte en el stack, éstos se escriben en la parte alta de una palabra en memoria, y la parte baja de dicha palabra (el byte con dirección impar) no cambia.

El MC68000 direcciona la memoria como bytes o como palabras (16 bits). Las palabras se accesan como direcciones pares. El MC68000 checa el direccionamiento a memoria para asegurar que todas las referencias a una palabra sean hechas a direcciones pares. En caso contrario, comienza el procesamiento de una excepción.

En la figura III.6 se muestra la distribución de bytes en memoria.

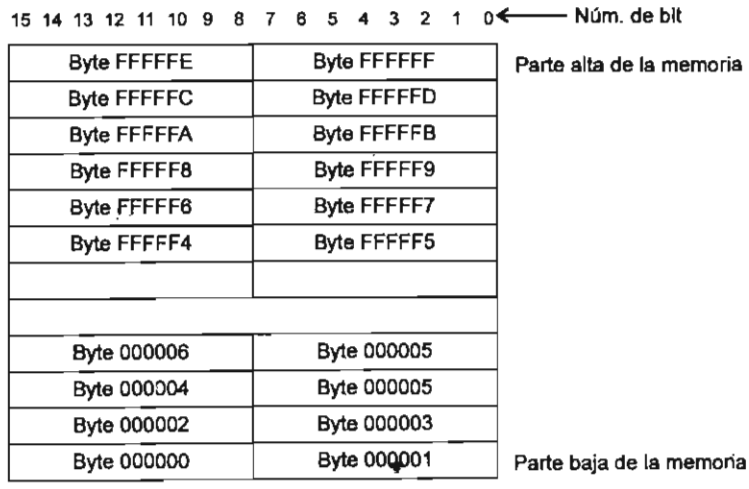


Figura III.6. Almacenamiento en el registro stack pointer.

Se puede observar que el primer byte en memoria (la dirección 000 000) es el más significativo de la primera palabra en memoria. Cuando se almacenan palabras en memoria sólo se accesan a partir de direcciones pares:

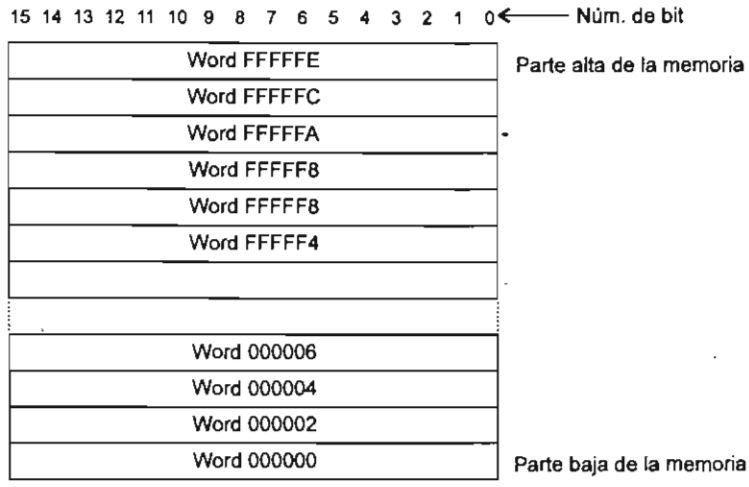


Figura III.7. Almacenamiento en el registro stack pointer.

Cuando se almacenan palabras dobles (o long words) de 32 bits en memoria, se usan dos palabras adyacentes en memoria o 4 bytes. La palabra más significativa de la palabra doble se almacena en las localidades de más alto orden, como se muestra en la figura III.8.

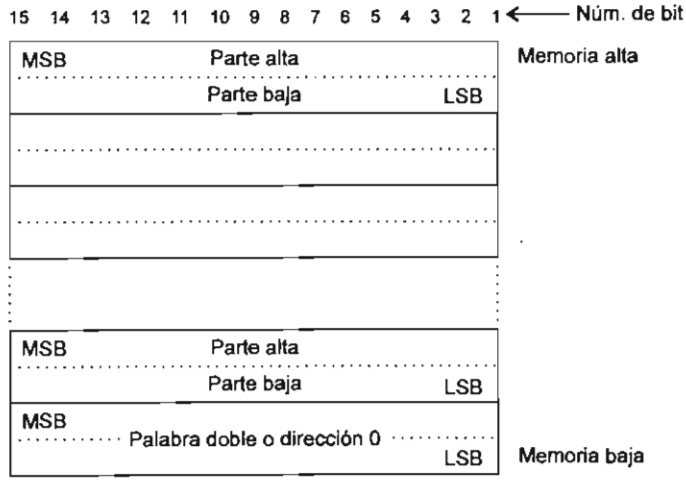


Figura III.8. Almacenamiento de palabras dobles.

El registro de estado

El MC68000 tiene un registro de estado o de *status* de 16 bits dividido en dos partes: el byte de sistema y el byte de usuario, como se muestra a continuación.

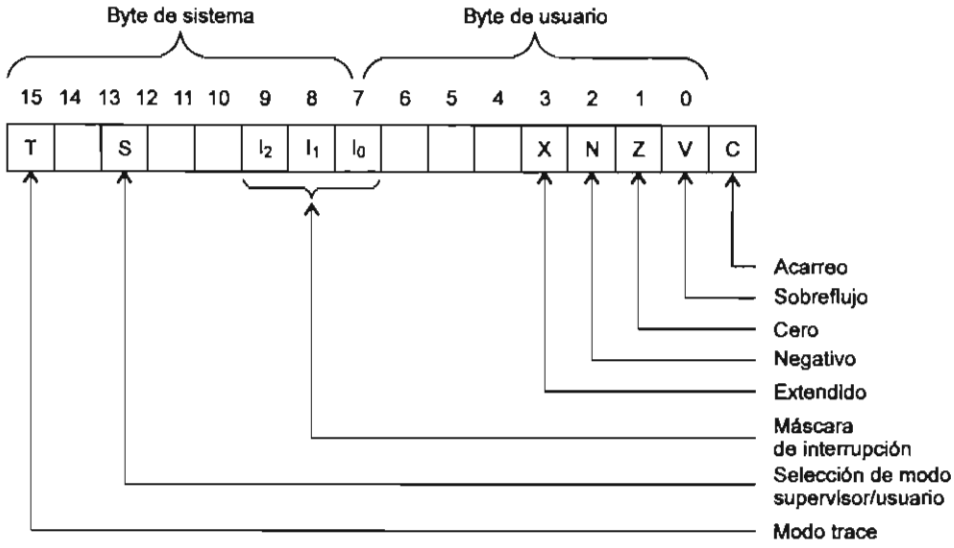


Figura III.9. Registro de estado.

La bandera de acarreo o carry (C) se activa si hay un acarreo del bit más significativo después de una adición, o si hubo un préstamo o borrow después de una sustracción. También puede ser afectado por algunas instrucciones de corrimiento y rotación.

La bandera de sobreflujo o de overflow (V) se activa cuando el resultado de una operación no se puede especificar con el tamaño del operando destino.

La bandera de cero o zero (Z) se activa cuando el resultado de una operación es cero.

La bandera de negativo o negative (N) es similar a la bandera de signo de otros microprocesadores y toma el valor del bit más significativo del resultado después de haber ejecutado una operación aritmética: 0-positivo 1-negativo.

El bit de extensión o extend (X) se usa en operaciones aritméticas multiprecisión. Toma el valor del carry cuando es afectado por una instrucción.

Los tres bits más significativos del byte de usuario no se usan y siempre son cero.

El byte de sistema contiene información relacionada con el propio sistema y puede modificarse cuando el MC68000 está en modo supervisor.

Máscara de interrupción

Son los tres bits menos significativos del byte de sistema del registro de estado.

El MC68000 tiene 7 niveles de interrupciones; el nivel de la interrupción es tomado y decodificado de las tres líneas de solicitud de interrupción y es comparado con el valor indicado por estos tres bits.

La interrupción con mayor prioridad es la que tiene nivel 7.

El nivel 7 es no mascarable y no se puede deshabilitar. El nivel 0 es una condición de "no solicitud de interrupción". Los niveles 1-6 son de interrupciones mascarables, de tal forma que si la máscara es 011 sólo se habilitan las interrupciones con nivel **4**, 5, 6 y 7.

Es decir, esta máscara indica cuál es el nivel que debe superar el nivel de una solicitud de interrupción.

Nivel de Interrupción	Máscara de interrupción			
	I_2	I_1	I_0	
Nivel 7	1	1	1	← Prioridad más alta (no mascarable)
Nivel 6	1	1	0	
Nivel 5	1	0	1	
Nivel 4	1	0	0	
Nivel 3	0	1	1	
Nivel 2	0	1	0	
Nivel 1	0	0	1	← Prioridad más baja
Nivel 0	0	0	0	← No solicitud de interrupción

Figura III.10. Máscara de interrupción.

El bit S del registro de estado

Especifica si el MC68000 está en modo supervisor o usuario:

1 Modo supervisor
0 Modo usuario

El bit T del registro de estado

Significa "trace". Cuando es "0", el MC68000 opera normalmente. Si es 1, el microprocesador se encuentra en el modo de operación "trace" (equivalente al modo single-step de otros procesadores), tal que se genera una trampa después de la ejecución de cada instrucción para monitorear los resultados de la misma y poder depurar así el programa.

Terminales

CUADRO III.1. *Descripción de las terminales del MC68000*

Nombre de la terminal	*Descripción*	*Tipo*
D0-D15	Bus de datos	Bidireccional, tercer estado
A1-A23	Bus de direcciones	Salida, tercer estado
$\overline{AS}$	Muestreo de direcciones	Salida, tercer estado
R/$\overline{W}$	Control de lectura/escritura	Salida, tercer estado
$\overline{UDS}$, $\overline{LDS}$	Upper, lower data strobes	Salida, tercer estado
$\overline{DTACK}$	Reconocimiento de transferencia de datos	Entrada
FC0, FC1, FC2	Salidas de funciones de código (estado)	Salida, tercer estado
$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	Solicitudes de interrupción	Entrada
$\overline{BERR}$	Error del bus	Entrada
$\overline{HALT}$	Operación HALT del procesador	Entrada/salida

CUADRO III.1 *(concluye)*

Nombre de la terminal	*Descripción*	*Tipo*
$\overline{\text{RESET}}$	Reset del procesador o de dispositivos externos	Entrada/salida
CLK	Reloj del sistema	Entrada
BR	Solicitud del bus	Entrada
$\overline{\text{BG}}$	Otorgamiento del bus	Salida
$\overline{\text{BGACK}}$	Reconocimiento del otorgamiento del bus	Entrada
E	Salida de habilitación (reloj)	Salida
$\overline{\text{VMA}}$	Dirección de memoria válida	Salida, tercer estado
$\overline{\text{VPA}}$	Dirección de periférico válida	Entrada
Vcc, GND	Alimentación (+5V) y tierra	

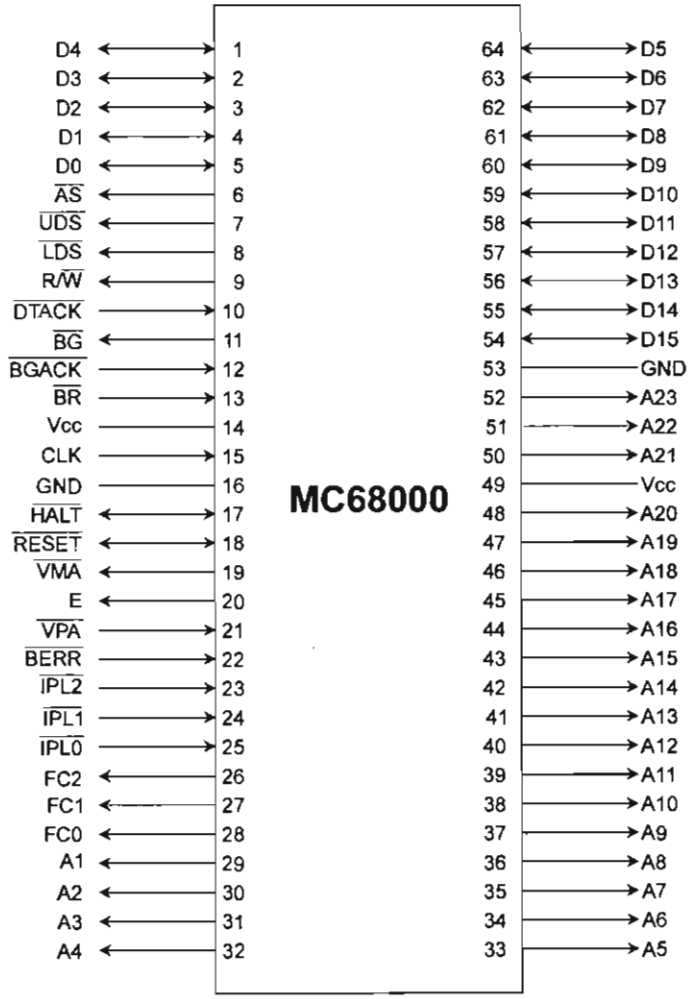


Figura III.11. Terminales del MC68000.

Bus de datos y bus de direcciones

D0-D15 son las líneas del bus de datos bidireccional.
A1-A23 son las líneas del bus de direcciones (salida).

El MC68000 no tiene multiplexados el bus de datos y direcciones, ya que es un circuito integrado de 64 terminales. No hay una línea A0, ya que este bit, junto con la especificación del tamaño del datos, son usados internamente para generar las señales UDS y LDS.

Señales UDS y LDS

Las señales de salida UDS *(upper data strobe)* y LDS *(lower data strobe)* indican si el MC68000 está usando el byte más significativo, el byte menos significativo, o ambos, para la transferencia de un dato por el bus de datos.

En el cuadro III.2 se muestra el significado de las señales UDS, LDS y R/W conjuntamente con el bus de datos.

CUADRO III.2. *Significado de las señales UDS, LDS y R/W*

USD	*LDS*	*R/$\overline{W}$*	*D8-D15*	*D0-D7*	*Operación*
Alto	Alto				
Bajo	Bajo	Alto	Bits de datos 8-15	Bits de datos 0-7	Lectura de palabra
Alto	Bajo	Alto		Bits de datos 0-7	Lectura de byte
Bajo	Alto	Alto	Bits de datos 8-15		Lectura de byte
Bajo	Bajo	Bajo	Bits de datos 8-15	Bits de datos 0-7	Escritura de palabra
Alto	Bajo	Bajo	Bits de datos 0-7	Bits de datos 0-7	Escritura de byte
Bajo	Alto	Bajo	Bits de datos 8-15	Bits de datos 8-15	Escritura de byte

Dato de entrada o salida no válido.

Si UDS es "0", se estará accesando un dato en memoria con una dirección par por medio de D8-D15.

Si LDS es "0", se estará accesando un dato en memoria con una dirección impar por medio de D0-D7.

Cuando se está accesando una palabra de datos se usan las 16 líneas del bus de datos activándose UDS y LDS.

Interface con memoria

En la figura III.12 se muestra la interface con memoria usando UDS y LDS.

Se selecciona la memoria que contiene los bytes con direcciones pares con la señal UDS, y la que contiene los bytes con direcciones impares con LDS.

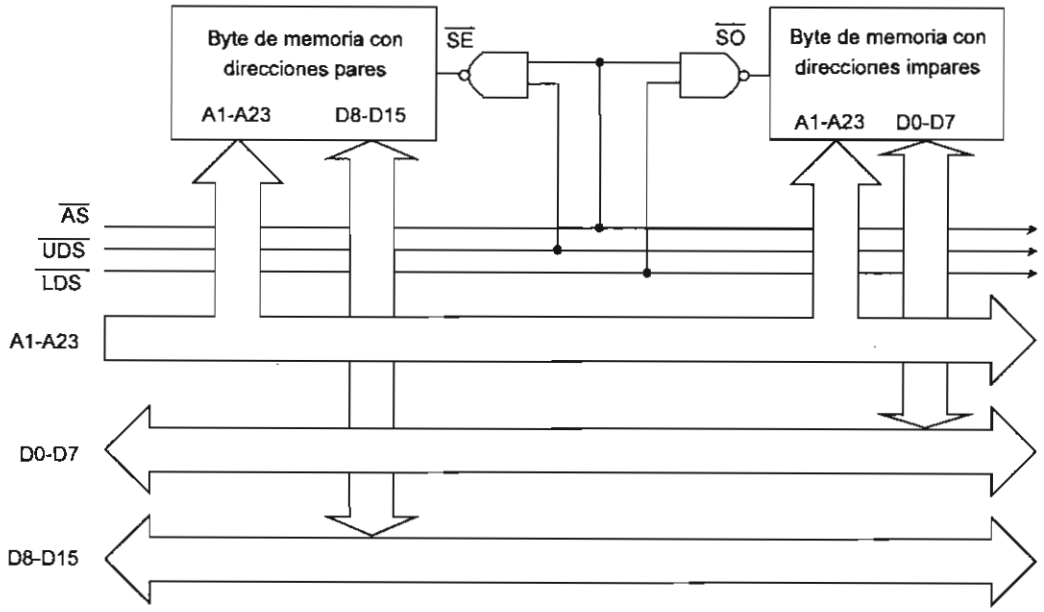


Figura III.12. Interface de memoria.

La señal AS *(address strobe)* es activada por el MC68000 para indicar que la dirección de un dato válido está saliendo por el bus de direcciones.

La señal DTACK (data transfer acknowledge)

Es una señal de entrada y debe ser generada por la lógica externa durante cada ciclo de lectura o escritura.

El MC68000 inserta ciclos de espera (o wait) automáticamente en un ciclo de lectura o escritura hasta recibir la señal DTACK. En el caso del 8086 y otros procesadores es lo contrario, ya que ellos cuentan con una entrada de wait por medio de la cual la lógica externa puede extender un ciclo de lectura o escritura.

El MC68000 puede trabajar con cualquier tipo de dispositivo sin importar su velocidad, ya que las operaciones del bus son completamente asíncronas, implicando que todos los dispositivos deben de tener una lógica para generar DTACK.

Salidas de código de función

Las salidas FC0, FC1 y FC2 también se llaman salidas de estado de ciclo del procesador e identifican el tipo de ciclo que está realizando el MC68000.

Estas salidas son válidas siempre que esté activada la salida AS.

Existen cinco diferentes tipos de ciclos: acceso a memoria de datos de supervisor, acceso a memoria de programa de supervisor, acceso a memoria de datos de usuario, acceso a memoria de programa de usuario y reconocimiento de interrupción.

Las salidas de código de función pueden usarse para dividir la memoria en cuatro tipos: usuario-supervisor, datos-programa.

CUADRO III.3. *Salidas FC0, FC1 y FC2*

FC0	*FC1*	*FC2*	*Tipo de ciclo de máquina*
0	0	0	
0	0	1	Acceso a memoria de datos de usuario
0	1	0	Acceso a memoria de programa de usuario
0	1	1	
1	0	0	
1	0	1	Acceso a memoria de datos de supervisor
1	1	0	Acceso a memoria de programa de supervisor
1	1	1	Reconocimiento de interrupción
			Reservado, actualmente indefinido

Así entonces, se puede ver que si se usan las salidas de código de función, el MC68000 puede accesar directamente hasta 64 Mbytes de memoria (4 bloques de 16 Mb cada uno).

Solicitudes de interrupción

Las entradas IPL0, IPL1 e IPL2 del MC68000 son las entradas de solicitud de interrupción y son decodificadas internamente para determinar el nivel de prioridad de la solicitud.

Si las tres entradas son "0", será una solicitud de interrupción no mascarable (nivel 7 de más alta prioridad) y siempre la reconocerá el MC68000. Si las tres entradas son "1", no se estará solicitando interrupción alguna.

Señal de bus error

BERR es una entrada que indica que ha sucedido un error en el bus. Al ser activada, el MC68000 realizará una secuencia similar a la respuesta de una solicitud de interrupción.

A esta secuencia se le conoce como procesamiento de excepción y la entrada BERR sirve para indicarle al MC68000 que un dispositivo externo no ha respondido (usando la entrada DTACK) dentro de la cantidad de tiempo esperado para una operación de lectura o escritura.

Esto implica que la lógica externa debe tener un mecanismo para evitar que el procesador espere indefinidamente la respuesta de un dispositivo que esté fallando (recuérdese que el MC68000 requiere que los dispositivos respondan en cada transferencia de datos). La lógica externa debe monitorear la actividad del bus y usar BERR para indicarle al MC68000 que el dispositivo está fallando. Cuando se intente accesar memoria protegida, esta lógica debe poder diferenciar si el error se debió a una respuesta enviada por una unidad de manejo de memoria (MMU-memory management unit).

En el procesamiento de la excepción se salva el contexto del procesador para permitirle ejecutar un programa y analizar la causa del error.

NOTA: Si se activa la señal BERR conjuntamente con HALT, el MC68000 reintentará ejecutar automáticamente el ciclo de bus donde sucedió el error.

La señal de HALT

Esta señal realiza varias funciones:

a) Conjuntamente con BERR se usa para volver a ejecutar ciclos de bus donde hubo errores.
b) Cuando se usa sola, coloca al MC68000 en un estado de HALT en el que el procesador se encuentra inactivo hasta que se desactive HALT.
c) Si se usa HALT junto con la señal RESET se inicializa al procesador.
d) La señal RESET también puede funcionar como salida, de tal forma que cuando el MC68000 ejecuta una instrucción RESET proporciona un pulso de bajada en la terminal RESET, lo cual se puede utilizar para inicializar otros dispositivos del sistema sin inicializar al procesador.
e) La señal HALT también es salida, de manera que cuando el procesador deja de ejecutar instrucciones, por ejemplo cuando ocurre una condición de doble error en el bus, el MC68000 activa la salida HALT. Esta salida puede ser usada por la lógica externa para detectar una condición catastrófica.

La entrada CLK

Es la entrada de la señal de reloj con niveles TTL, del cual se deriva toda la temporización interna del MC68000.

Señales para arbitraje del bus

Son las señales BR (solicitud del bus-bus request), BG (otorgamiento del bus-bus grant) y BGACK (reconocimiento del otorgamiento del bus-bus grant acknowledge).

Estas señales se usan en sistemas donde otros dispositivos, como controladores de Dma u otros procesadores, requieren el uso y control de los buses del sistema.

Cuando un dispositivo externo necesite usar el bus del sistema, deberá activar la entrada BR. Por su parte, el MC68000 terminará de ejecutar el ciclo de bus actual y activará la salida BG, con la que le indica al dispositivo que el bus estará disponible al final del ciclo del bus actual.

Una vez que el dispositivo externo toma el control del bus debe indicárselo al MC68000 por medio de la señal BGACK. Dicho dispositivo debe mantener activada esta señal hasta que termine de usar el bus.

La señal BGACK indica "bus ocupado" y les permite al MC68000 y a otros dispositivos del sistema saber que el bus no está disponible.

Señales de la familia 6800

Las señales E, VPA y VMA sirven para que el MC68000 se pueda conectar fácilmente a los dispositivos de la familia 6800.

Estos dispositivos usan un mecanismo síncrono para la transferencia de información. Usan la señal de habilitación de reloj del sistema E *(system clock enable)* distribuida en todos los dispositivos 6800, de modo que todas las transferencias de datos estén sincronizadas con esta señal de reloj.

Por lo tanto, la señal E proporcionada por el MC68000 es la equivalente a la señal E del 6800. La frecuencia de E es 1/10 de la frecuencia de la señal CLK. El periodo de E es igual a 10 periodos de CLK; E es "0" durante 6 ciclos de CLK y "1" durante 4 ciclos de CLK.

La señal VPA (dirección de periférico válida-*valid peripheral address*) es usada por los dispositivos del tipo 6800 para informarle al MC68000 que se requiere una transferencia de datos.

Para generar esta señal, en el sistema debe haber una lógica que decodifique o determine si se está accesando un dispositivo de la familia 6800 y generar la señal VPA. Al recibir la señal VPA, el MC68000 modifica la temporización de la transferencia de datos sincronizándola con la señal E. El MC68000 activará la salida VMA (dirección de memoria válida-*valid memory address*), la cual también es una señal del tipo 6800 y se activará sola.

Estas tres señales serán discutidas detalladamente más adelante.

Temporización y operación del bus

El número total de ciclos para cada instrucción está definido en la tabla del set de instrucciones del anexo.

El MC68000 mapea los dispositivos de entrada/salida en memoria. Cada periodo de reloj se divide en dos estados.

Temporización para la lectura de una palabra

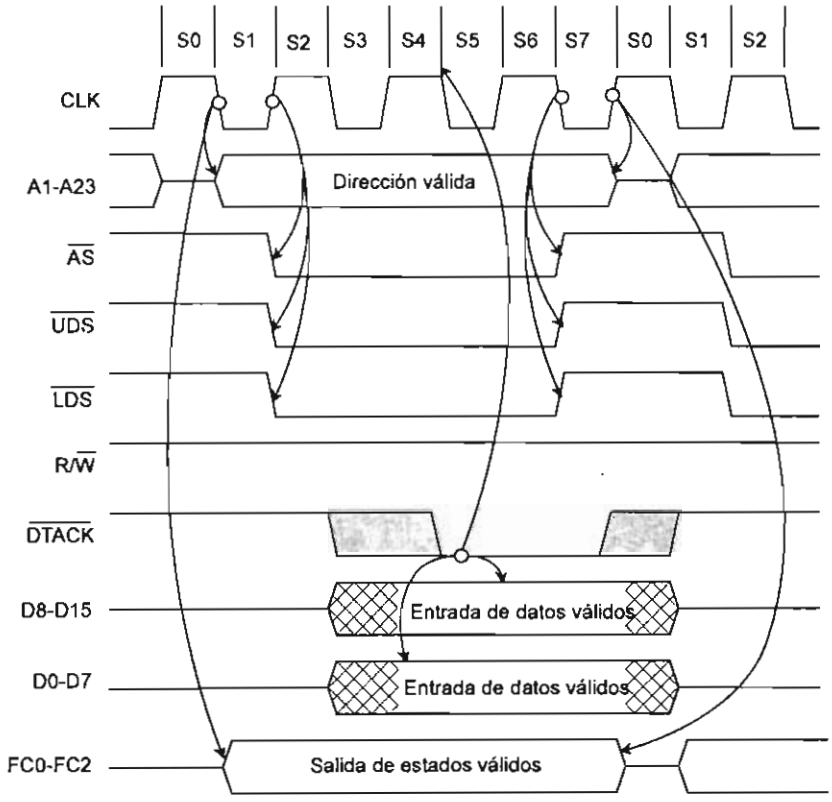


Figura III.13. Temporización para la lectura de una palabra.

Durante el estado S0, los buses de datos y direcciones se encuentran en alta impedancia. La dirección de la localidad de memoria sale al inicio del estado S1 en el bus de direcciones (A1-A23), mientras que por las salidas FC0-FC2 sale la información del estado de ciclo del procesador.

La señal AS (address strobe) se activa al inicio de S2 y puede ser usada por la lógica externa para latchear la información del bus de direcciones.

Al mismo tiempo, se activan las señales UDS (upper data strobe) y LDS (lower data strobe), las cuales se utilizan para habilitar la selección del byte más significativo y menos significativo de una palabra respectivamente. Estas señales no son de strobe, ya que aún no hay dato de entrada o de salida listo en el bus, sino que más bien son señales para habilitar la memoria que contiene al byte más significativo y al menos significativo de una palabra de 16 bits.

La señal R/W no cambia durante este ciclo.

A continuación, el MC68000 espera a que la memoria o el dispositivo de entrada/salida depositen el dato en el bus y activen la señal DTACK cuando dicho dato esté listo.

Si DTACK no está presente en S5, el MC68000 inserta automáticamente ciclos de espera (wait) en el ciclo de escritura.

Una vez activado DTACK, el ciclo de escritura continúa con S5. Al final de S6 se desactivan AS, UDS y LDS y el dato de D0-D15 se latchea en un registro interno del MC68000.

El dispositivo externo, al detectar que el MC68000 ha capturado el dato del bus (por medio de la transición bajo-alto de AS, UDS y LDS), debe desactivar DTACK inmediatamente para no interferir con el inicio del siguiente ciclo de bus.

Estados de espera (wait)

Los estados de espera (wait) ocurren entre el estado 4 y el 5 durante las operaciones de lectura. Durante los estados de espera, el MC68000 mantendrá direcciones de salida válidas y las señales AS, UDS y LDS estarán activadas hasta que DTACK sea desactivada.

Siempre habrá un número par de estados wait, ya que todas las operaciones del MC68000 están basadas en ciclos de CLK completos.

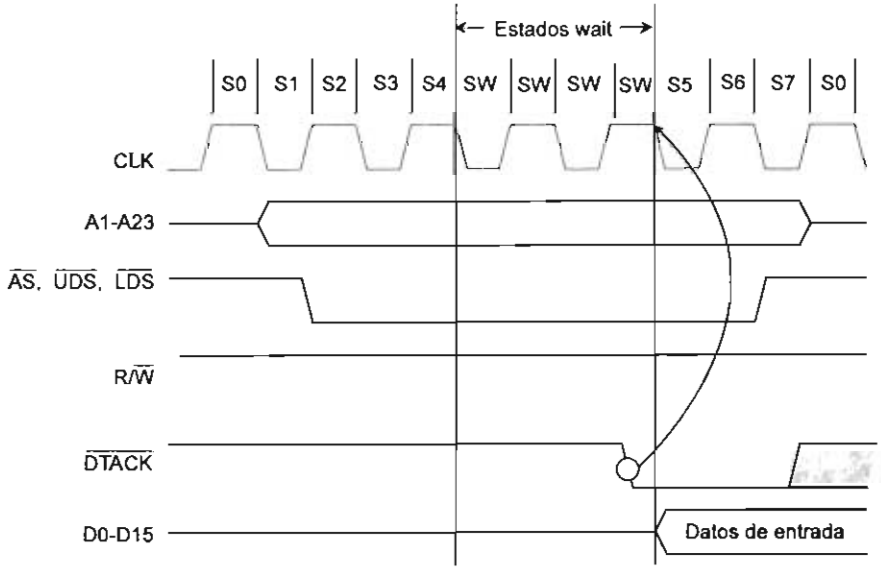


Figura III.14. Estado de espera (wait).

Temporización para la lectura de un byte

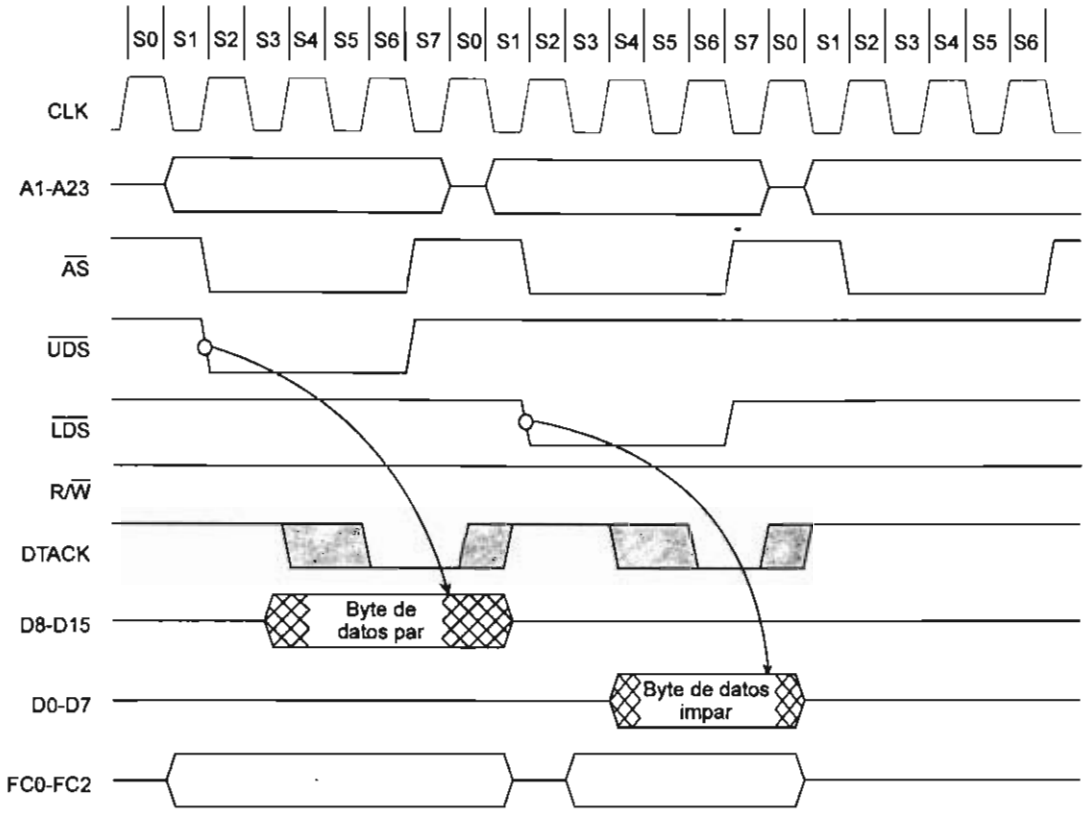


Figura III.15. Temporización para la lectura de un byte.

En la figura III.15 se puede ver primero la lectura de un byte de datos con dirección par y después la lectura de un byte con dirección impar. La única diferencia con respecto a la temporización de la lectura de una palabra es que sólo se activa UDS o LDS y únicamente son usadas 8 líneas del bus de datos.

Cuando se activa UDS se lee un byte de datos localizado en una dirección par por medio de las líneas D8-D15, y cuando se activa LDS se lee un byte de una dirección impar por medio de las líneas D0-D7.

NOTA: No debe entenderse de la figura anterior que el MC68000 siempre lee dos bytes adyacentes, par e impar, sino que en esta figura se muestran consecutivamente estas operaciones para ilustrar las mismas.

Temporización para la escritura de una palabra

Al inicio de S1 sale la dirección de la localidad de memoria o dispositivo de entrada/salida junto con el código de función que indica el tipo de ciclo de bus del procesador. Si en el ciclo anterior el MC68000 usó el bus de datos, el procesador regresa todas las salidas de datos a alta impedancia durante S1 y activa AS, pasando la salida read/write (R/W) a nivel bajo. En este caso la salida AS también se puede usar para latchear externamente la dirección y la salida R/W le indicará a los dispositivos externos que ha colocado un dato en el bus.

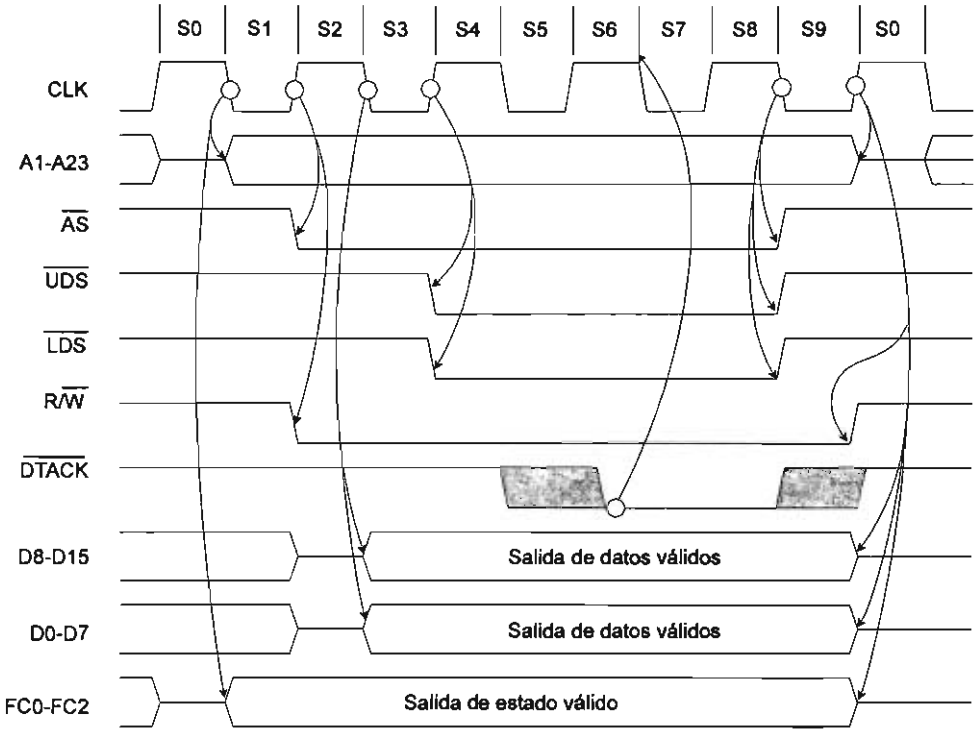


Figura III.16. Temporización para la escritura de una palabra.

Las señales UDS y LDS se activan al inicio de S4. Durante las operaciones de escritura, se pueden usar estas salidas como señal de strobe, ya que indican que el dato en el bus es válido.

A continuación, la lógica externa debe responder activando la entrada DTACK al inicio de S7. Si DTACK no está activada al inicio de S7, el MC68000 insertará automáticamente ciclos de espera, al igual que lo hizo para las operaciones de lectura, excepto que los ciclos wait se insertan en un punto diferente, tal como se muestra en la figura III.17.

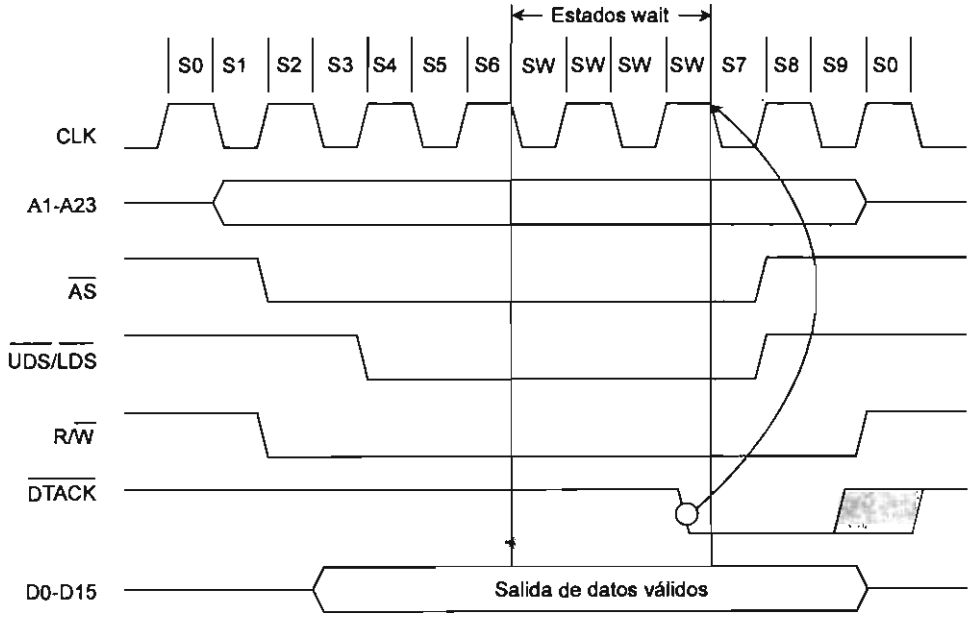


Figura III.17. Inserción automática de los ciclos wait.

En una operación de escritura, el MC68000 saca el dato por medio de las líneas D0-D15. Las señales AS, UDS y LDS se desactivan al inicio de S9 y la señal R/W pasa a nivel alto al final de S9. Los buses de datos y direcciones, así como las salidas de código de función pasan a un estado de alta impedancia, liberando el bus. La memoria o dispositivo externo debe liberar DTACK al detectar una transición en las señales de data strobe.

Temporización para la escritura de un byte

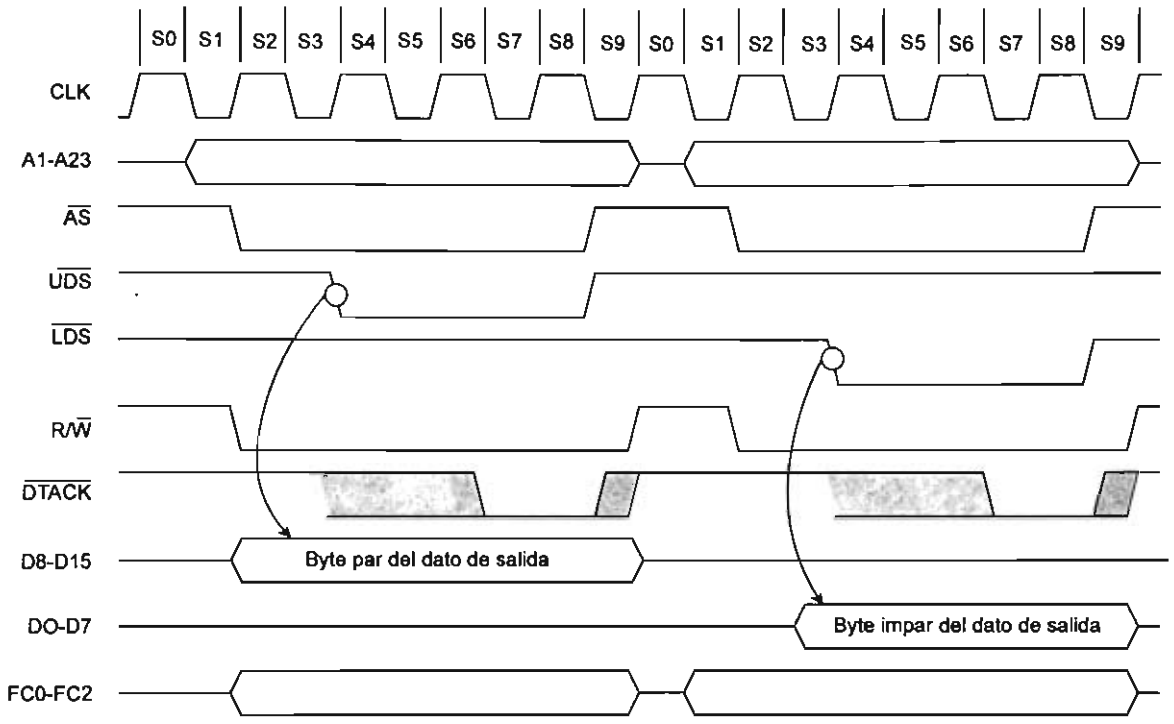


Figura III.18. Temporización para la escritura de un byte.

La única diferencia con respecto a la escritura de una palabra es que al escribirse un byte sólo se activan la señal UDS o la señal LDS.

Temporización para una operación del tipo lectura-modificación-escritura *(read-modify-write)*

No es común encontrar el ciclo lectura-modificación-escritura *(read-modify-write)* entre microprocesadores, sino que más bien es proporcionado por minicomputadoras. Este tipo de ciclo sólo es usado por el MC68000 cuando ejecuta la instrucción test and set (TAS).

Esta instrucción lee un byte de datos, establece códigos de función de acuerdo al contenido de ese byte, activa el bit 7 del byte y después escribe el byte en memoria. Esta instrucción sirve para proporcionar una comunicación segura entre microprocesadores en un sistema multiprocesador, ya que el ciclo de bus de esta instrucción no es interrumpible.

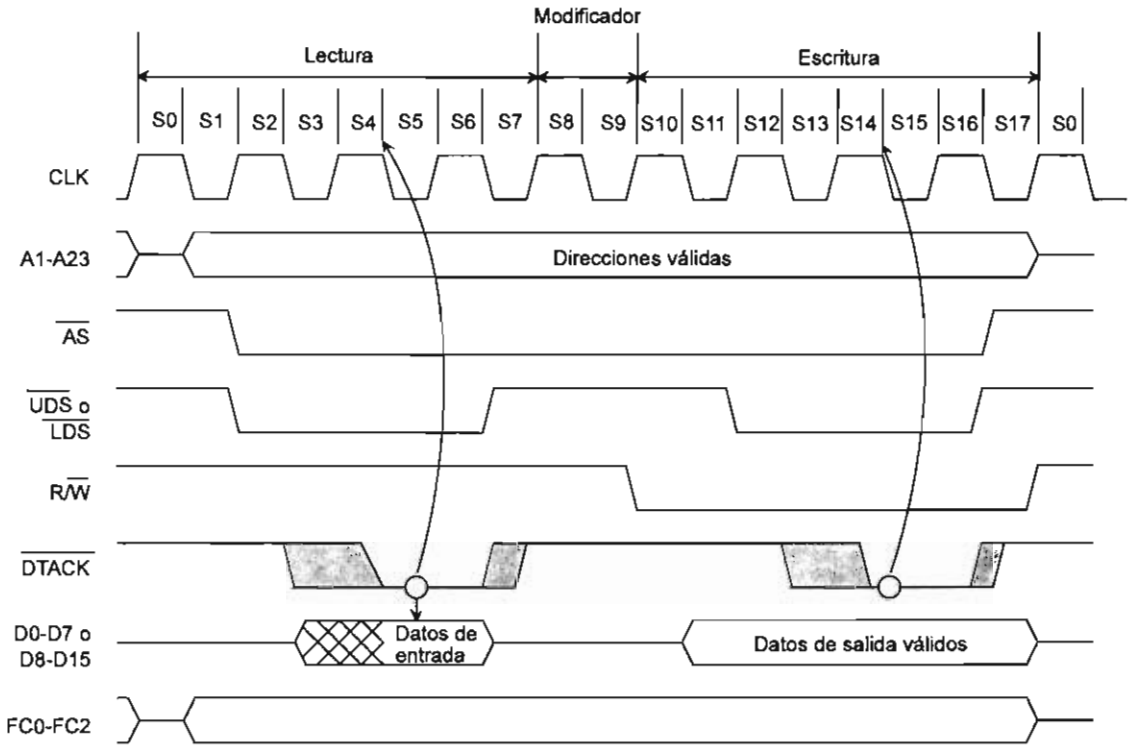


Figura III.19. Temporización para una operación de lectura-modificación-escritura.

El ciclo read-modify-write consiste en un ciclo de lectura de un byte seguido de un ciclo de escritura de un byte. Hay un periodo de reloj (S8 y S9) entre los ciclos de lectura y escritura en el que el byte leído es modificado internamente y posteriormente escrito. En este caso, también la lógica externa debe activar DTACK en el momento adecuado o en caso contrario se insertan ciclos wait.

Durante la operación read-modify-write, las señales UDS o LDS serán desactivadas, ya que la instrucción TAS opera sobre un byte y nunca sobre una palabra.

La operación RESET

El MC68000 tiene una entrada de RESET asíncrona. El microprocesador se inicializa (RESET) activando las señales RESET y HALT durante al menos 100 milisegundos. Después de que se desactivan estas señales, el MC68000 ejecuta las siguientes acciones:

1. El MC68000 lee las primeras cuatro palabras de memoria (bytes 000 000-000 007) y carga el contenido de estas localidades de memoria en el *system stack pointer* (SSP) y en el program counter (PC) de acuerdo a la figura III.20.
2. Se establece la máscara de interrupción del registro de estado con todos 1, de tal forma que sólo se habilitan las interrupciones de nivel 7. Todos los registros, excepto SSP, el PC y el registro de estado, contendrán valores indeterminados.
3. La ejecución del programa comienza en la localidad indicada por el program counter.

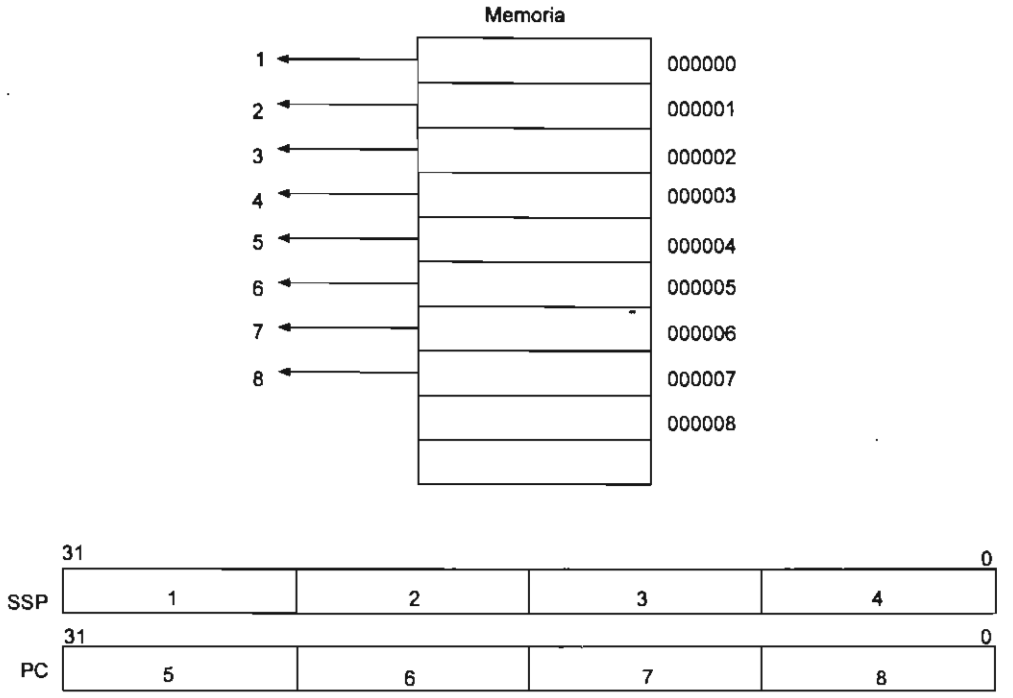


Figura III.20. Acciones después de una operación reset.

La instrucción RESET

El pin RESET es bidireccional, de tal forma que cuando el MC68000 ejecuta una instrucción RESET envía un pulso de bajada en el pin RESET. Éste es un pulso que se puede llevar a cabo por software y es "0" lógico durante 124 ciclos de CLK. Esta instrucción no afecta el estado interno y registros del microprocesador. La señal RESET es usada para inicializar otros dispositivos del sistema bajo control del MC68000.

El estado HALT

Se puede llevar al MC68000 a un estado HALT en el que los buses de datos y direcciones y salidas de código de función (FC0-FC2) son llevados a alta impedancia. Éste es similar al estado hold del 8086 y al estado stop del Z8000.

El estado HALT se puede usar para deshabilitar al MC68000 y liberar los buses del sistema para un acceso directo a memoria o para ser usados por otro procesador.

Por otro lado, el MC68000 cuenta con un subsistema para el arbitraje del bus, por lo que el estado HALT se usa más comúnmente para desarrollar el modo single-step por hardware.

Temporización de HALT

Cuando el MC68000 está en la mitad de un ciclo de bus y la señal HALT es activada, dicho ciclo de bus continúa hasta terminar normalmente. Al final del ciclo los buses de direcciones, datos y las señales FC0-FC2 pasan a alta impedancia y el MC68000 pasa al estado HALT

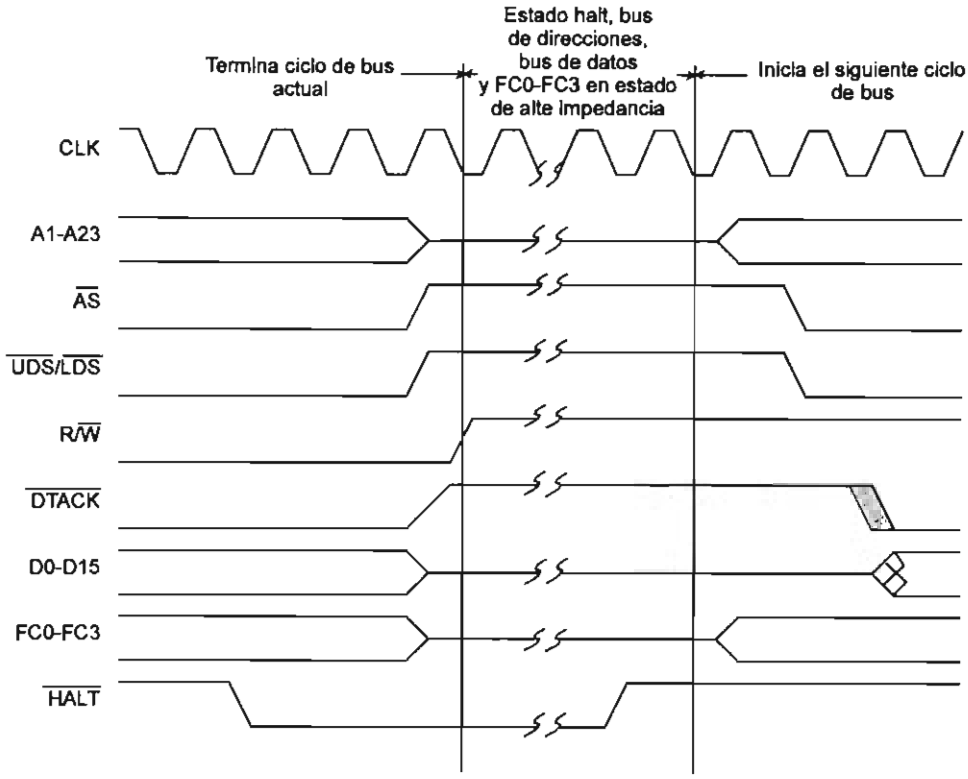


Figura III.21. Temporización de un estado HALT.

En este estado el procesador no hace operación alguna y espera que la señal HALT sea desactivada. El MC68000 no proporciona una señal de reconocimiento de HALT a la lógica externa.

Cuando el MC68000 está en estado HALT su circuitería de arbitraje del bus continúa operando, y cualquier solicitud del bus es concedida inmediatamente. Al desactivarse la señal HALT, el MC68000 sale del estado HALT en dos ciclos de reloj y comienza otro ciclo de bus.

Single-step con HALT

La ejecución de la mayoría de las instrucciones del MC68000 requiere varios ciclos de bus para buscar el código de operación y operandos y posiblemente almacenar resultados de la instrucción.

La secuencia de HALT puede ocurrir entre dos instrucciones o a la mitad de una sola instrucción, tal que si se usa la entrada HALT para implementar el modo de operación single-step, realmente será por ciclos de bus y no single-step por instrucciones.

Si se necesita tener single-step por instrucciones se deberá activar el bit T (función trace) en el byte de supervisor del registro de estado. Esta operación será vista posteriormente.

La señal de salida HALT

La señal HALT es bidireccional y es activada por el MC68000 cuando entra al estado HALT, en lugar de tener lógica externa que cause el HALT.

El MC68000 entra a un estado HALT automáticamente cuando hay una doble falla del bus (se verá más adelante).

Al entrar al estado HALT el MC68000 activa la salida HALT y permanece en esta condición hasta que sea iniciada externamente una operación de RESET con la señal RESET.

Así pues, cuando HALT pasa a "0" lógico el MC68000 indicará una falla catastrófica.

El estado STOP

La instrucción STOP

Después de la ejecución de la instrucción STOP, el MC68000 entra en estado STOP. La instrucción STOP sólo se puede ejecutar cuando el MC68000 está operando el modo supervisor. El estado STOP es similar al estado HALT, ya que el microprocesador no hace operación alguna.

Cuando se ejecuta la instrucción STOP, el registro del estado es cargado con un nuevo valor contenido en la instrucción. A continuación el PC es actualizado para apuntar a la instrucción siguiente y el MC68000 se detiene.

No hay señales especiales que indiquen que el MC68000 está en el estado STOP. El estado STOP termina por una de las condiciones de excepción tales como una solicitud de interrupción o un RESET. Cuando el MC68000 detecta una condición de excepción sale del estado STOP y procesa esta condición.

Temporización de reejecución de ciclo de bus

El MC68000 puede responder de dos formas a un error en el bus del sistema, indicado por la activación de BERR. Puede procesar la excepción o puede intentar reejecutar *(rerun)* el ciclo de bus donde ocurrió el error.

Si BERR fue activada se usa el método de manejo de procesamiento de la excepción (o método software) para tratar el error.

Si la señal BERR es activada junto con la señal HALT, entonces el MC68000 lo toma como una solicitud de reejecución del ciclo de bus.

En la figura III.22 se muestra un ciclo de escritura ejecutándose y esperando el MC68000 que el dispositivo externo responda con DTACK para que pueda complementarse el ciclo.

En lugar del reconocimiento, la lógica externa activa las señales BERR y HALT indicando que el ciclo no se completó exitosamente y que el MC68000 debe reejecutarlo.

El MC68000 procede a completar el ciclo y entra al estado HALT. Los buses de datos y direcciones, y las salidas de código de función pasan a alta impedancia, permaneciendo el microprocesador en estado HALT hasta que BERR y HALT sean desactivadas.

BERR debe desactivarse antes de HALT para evitar que el MC68000 interprete la activación de BERR como otro error en el bus (manejado por software). Después de que se desactiva HALT el MC68000 repite el ciclo que estaba ejecutando al recibir la solicitud de reejecución, y será repetida la misma información en los buses de datos y direcciones y en el código de función.

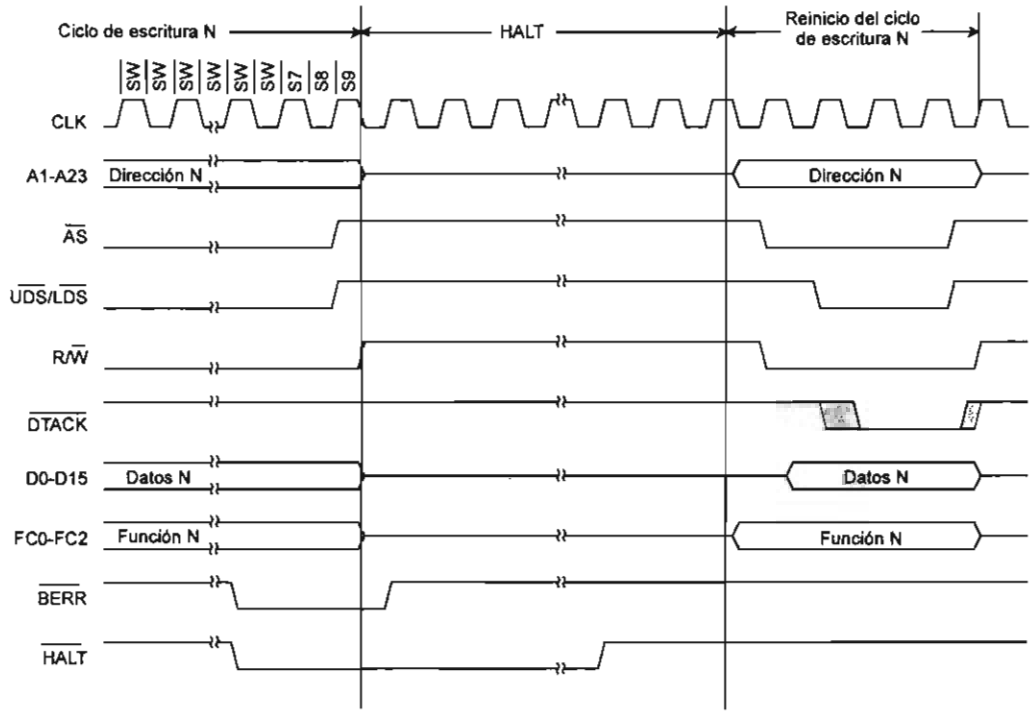


Figura III.22. Temporización de reejecución de ciclo de bus.

Reejecución exitosa

En la figura III.22 también se muestra la reejecución exitosa de un ciclo recibiéndose DTACK en el tiempo esperado. Sin embargo, la lógica externa puede continuar solicitando que el ciclo se reejecute un número ilimitado de veces usando BERR y HALT.

Por otra parte, si se está usando el método de procesamiento de excepción por software para manejar el error en el bus (sólo se activa BERR sin HALT), dos errores sucesivos en el bus serán considerados como un error catastrófico y el MC68000 entrará al estado HALT permaneciendo ahí hasta un RESET.

Si el MC68000 está realizando un ciclo read-modify-write y ocurre un error en el bus, no se reejecutará el ciclo, ya que la instrucción TEST and SET requiere integridad en su ciclo de ejecución. Si la lógica externa solicita la reejecución del ciclo read-modify-write, el MC68000 ejecuta la rutina de procesamiento de excepción de error en el bus, lo cual se verá más adelante.

Lógica de arbitraje del bus

El MC68000 no maneja prioridades para solicitudes del uso del bus por parte de dispositivos externos. El procesador asume que él es el dispositivo de más baja prioridad en el sistema y siempre otorga el bus a dispositivos que lo soliciten, siempre y cuando no lo esté usando.

El MC68000 permite que otros dispositivos usen el bus entre instrucciones y entre ciclos de bus de una sola instrucción.

Ya que no tiene una lógica interna para el arbitraje, debe haber alguna externa para asignar prioridades a las solicitudes del uso del bus del sistema para que un dispositivo con prioridad alta no sea interrumpido por otro con prioridad baja.

Hay tres señales que pueden usarse en una lógica de arbitraje del bus: bus request (BR), bus grant (BG) y bus grant acknowledge (BGACK).

Cuando el MC68000 está usando el bus, las entradas BR y BGACK estarán desactivadas, y la salida BG será "1" lógico.

Temporización para el arbitraje del bus

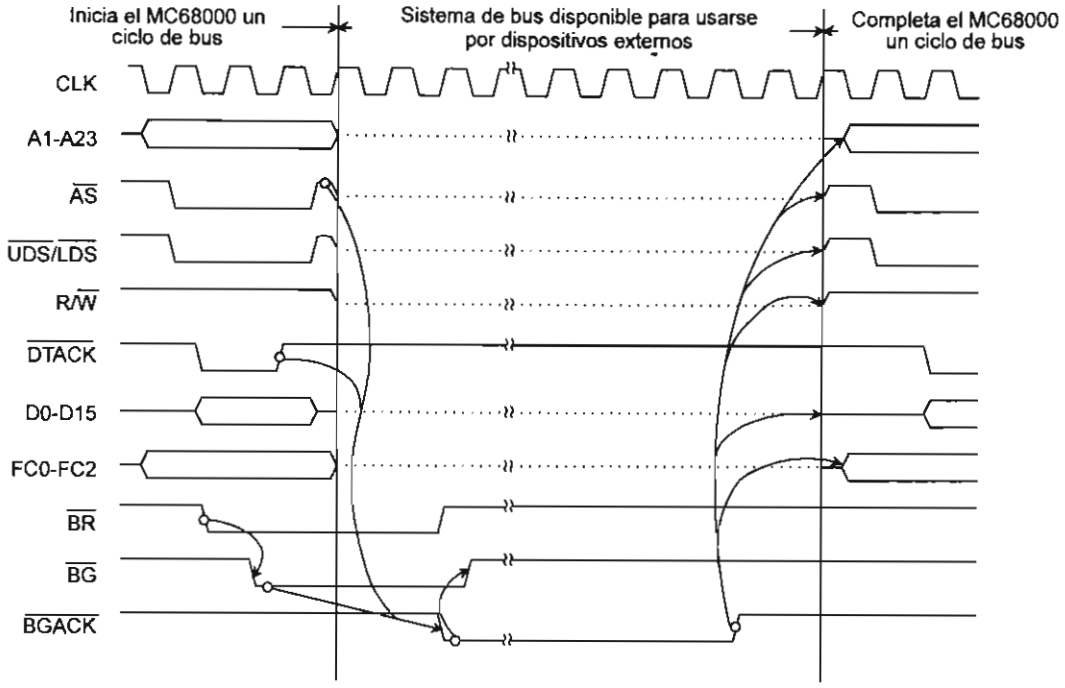


Figura III.23. Temporización para el arbitraje del bus.

El arbitraje del bus comienza cuando un dispositivo externo activa la entrada BR, entonces el MC68000 responde activando BG un pulso de reloj después. La única excepción a esta respuesta inmediata es cuando el MC68000 está en las etapas iniciales de un ciclo de bus y no ha enviado aún AS. En este caso el MC68000 espera a activar AS y un periodo de CLK después envía BG: el tiempo de respuesta es de máximo tres periodos CLK en este caso.

La señal bus grant no indica que el bus esté disponible para el dispositivo solicitante en este punto, ya que el MC68000 todavía puede estar usando el bus para completar su ciclo, por lo que el dispositivo solicitante debe monitorear otras señales.

Así el dispositivo solicitante debe esperar que se desactive AS, la cual indica que el MC68000 ha terminado el ciclo actual. El dispositivo que solicita el bus también debe esperar que la señal DTACK sea desactivada, lo cual indica que el dispositivo que estaba usando el bus realmente ha sido terminado de usar.

Finalmente, el dispositivo solicitante debe checar el estado de la señal BGACK. Si esta señal está activada, indicará que otro dispositivo del bus ha tomado el control de él y no ha terminado de usarlo. Sin embargo, si BGACK está desactivada, el bus del sistema estará disponible para poder usarse al final del ciclo.

Si se cumplen las condiciones anteriores, el dispositivo solicitante debe activar BGACK, con lo que informa al MC68000 que ha tomado el control del bus, y mantener dicha señal en "0" lógico

tanto tiempo como requiera el bus. El MC68000 mantiene en alta impedancia sus líneas de salida hasta que BGACK sea desactivada.

Cuando BGACK sea desactivada, el MC68000 podrá iniciar otro ciclo de bus, pero si hay pendiente otra solicitud del bus, el MC68000 otorgará el control del bus inmediatamente, sin ejecutar algún ciclo de bus.

Procesamiento de excepciones

Motorola usa el término "procesamiento de excepciones" para denotar el procesamiento de un amplio rango de eventos, los cuales incluyen a los conocidos como solicitudes de interrupción en otros microprocesadores.

La lógica para el procesamiento de excepciones es similar a las del 8086 y del Z8000, ya que hay una tabla de vectores usada para transferir el control del programa a la rutina de manejo de la excepción. El número de eventos que pueden causar una excepción en el MC68000 es mayor al número de eventos que causan una interrupción en el 8086 y el 28000.

El MC68000 proporciona una estructura de 7 niveles de prioridad para solicitudes de interrupciones externas.

Modos de operación

Cuando se inicializa el MC68000 usando la entrada RESET, empieza a operar en modo supervisor y permanece ahí hasta que sea ejecutada una de las siguientes instrucciones: regreso de excepción (RTE), mover al registro de estado (*move word to* SR), AND inmediato al registro de estado (EORI word a SR). Estas instrucciones pueden cambiar el bit 5 del registro de estado y no causan la transición al modo usuario automáticamente.

Cuando el MC68000 está operando en modo usuario, lo único que puede regresarlo a modo supervisor es una excepción. Todo el procesamiento de la excepción es en modo supervisor, independientemente del valor del bit 5 del registro de estado.

Al terminar de procesarse la excepción, la instrucción de regreso de excepción (RTE) permite regresar al modo usuario.

Tipos de excepciones

Excepciones generadas internamente

Se pueden dividir en tres categorías: errores detectados internamente, trampas de instrucción y la función trace.

Los errores detectados internamente que causan que el MC68000 inicie el procesamiento de una excepción son los siguientes:

a) Errores de direccionamiento. Cuando el MC68000 intenta accesar datos de palabra, long word o una instrucción en una dirección impar.
b) Violaciones de privilegio. Algunas instrucciones sólo se pueden ejecutar en modo supervisor: STOP, RESET, RTE, MOVE a SR, AND (word) inmediato a SR, EOR (word) inmediato a SR, MOVE USP.
c) Códigos de operación ilegales o no implementados.

Cuando el patrón de bits del código de una instrucción no está definido en los patrones de bits de instrucciones del MC68000, se iniciará el procesamiento de una excepción.

Hay dos patrones definidos como no implementados que no son ilegales: 1010 y 1111 en los bits 15-12. Si el MC68000 encuentra estos códigos de operación se inicia el procesamiento de una excepción especial, el cual permite usar estas instrucciones no implementadas como propias del programador.

Las trampas de instrucciones son excepciones causadas por la ejecución de instrucciones del programa. Hay una instrucción de TRAP estándar (similar a la instrucción system call del Z8000).

Existen también las instrucciones TRAPV, CHK, DIVS y DIVU que causan que sea iniciado el procesamiento de una excepción si se detectan ciertas condiciones como overflow aritmético o división entre cero.

El tercer tipo de excepción generada internamente es cuando el MC68000 está operando con la función trace. El bit T del registro de estado está activado y causa que una excepción sea procesada después de cada instrucción; esta función sirve para depurar programas, ya que se pueden analizar los resultados de cada instrucción.

Excepciones generadas externamente

Hay tres tipos:

1. Errores del bus. Cuando la señal BERR es activada por la lógica externa (y HALT está desactivada).
2. RESET. Cuando la lógica externa activa la señal RESET.
3. Solicitud de interrupción. Es la forma más familiar de procesar una excepción y es iniciada por la lógica externa por medio de las líneas de solicitud de interrupción IPL0, IPL1 e IPL2.

Prioridades de excepciones

Hay diferentes prioridades para los diversos tipos de excepciones y su procesamiento depende de su prioridad.

CUADRO III.4. *Prioridad de interrupciones*

Prioridad	*Fuente de excepción*	*Respuesta al procesamiento de excepción*
Más alto	$\overline{\text{RESET}}$ BERR (error de bus) Error de dirección	Aborta ciclo actual, después procesa la excepción
	Trace Solicitud de interrupción Código ilegal/no implementado Violación de privilegio	Completa la instrucción actual, después procesa la excepción
Más bajo	TRAP, TRAPV CHK División-por-cero	La ejecución de la instrucción inicia el procesamiento de excepción.

Las solicitudes de interrupción tienen adicionalmente un nivel de prioridad, como ya se ha visto. En el cuadro III.4 se puede ver que las excepciones causadas por instrucciones de trampa pueden iniciar el procesamiento de la excepción como parte de su ejecución normal.

Todas las excepciones de instrucciones de trampa tienen igual prioridad, ya que es imposible que dos de ellas generen excepciones simultáneamente.

Tabla de vectores de excepciones

Hay una tabla de vectores para el procesamiento de excepciones, la cual ocupa 1024 bytes (512 palabras) de memoria, de las localidades 000 000H-000 3FFH.

El cuadro III.5 contiene 256 vectores de 4 bytes. Cada vector es una dirección de 32 bits, la cual será cargada en el program counter como parte del procesamiento de la excepción. Algunos vectores del cuadro sirven para procesar las excepciones vistas anteriormente. Los vectores restantes son reservados para ser usados por Motorola y no deben usarse si se desea que el programa sea compatible en software con Motorola.

Los primeros 64 vectores de excepción tienen usos predefinidos, por lo que hay 192 vectores disponibles para solicitudes de interrupciones externas, cantidad suficiente para la mayoría de las aplicaciones.

Sin embargo, ya que los primeros 64 vectores no son protegidos por el MC68000, éstos pueden ser usados por interrupciones externas si es que se necesitan.

Secuencias del procesamiento de excepciones

Esta secuencia es la misma independientemente de la fuente de excepción, salvo algunas diferencias.

Procesamiento de excepciones generadas internamente

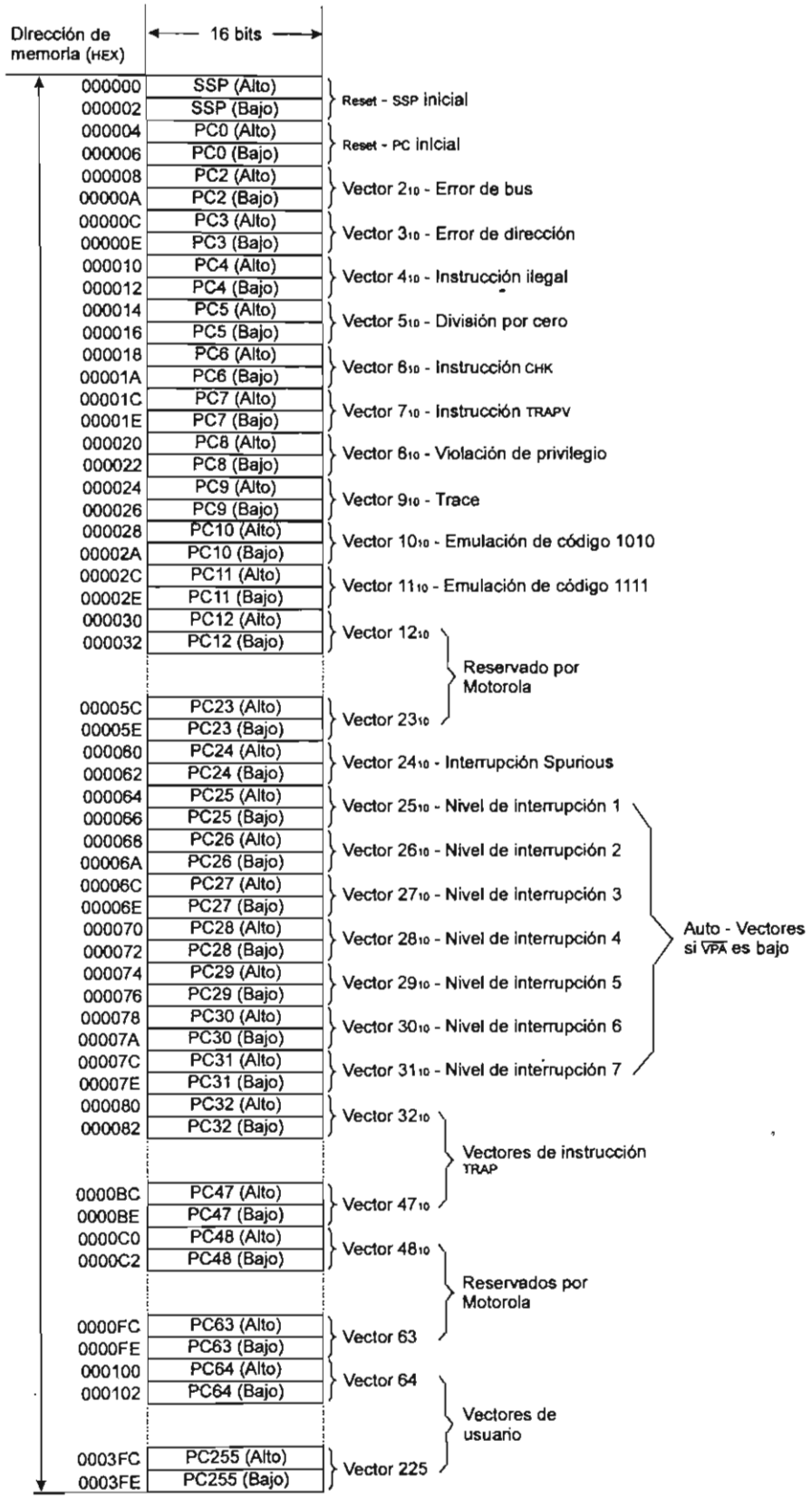
Función trace, instrucción TRAP, códigos de operación ilegales o no implementados o violación de privilegio. Ocurren los siguientes pasos:

1. El contenido del registro de estado es copiado a un registro interno.
2. Se coloca en "1" el bit S del registro de estado, colocando al MC68000 en el modo supervisor.
3. Se coloca en "0" el bit T del registro de estado para permitir la ejecución continua cuando se depura el programa usando trace.
4. El programa counter se almacena en el stack de supervisor. El SSP se decrementa en cuatro, ya que el PC es de 32 bits.
5. El registro de estado se almacena en el stack de supervisor. El SSP se decrementa en dos, ya que el registro de estado es de 16 bits.
6. El program counter se carga con el contenido de la localidad apropiada de la tabla de vectores.
7. La ejecución del programa continúa en la localidad indicada por el program counter.

Procesamiento de excepciones de error en el bus y error de dirección

Incluye varios pasos adicionales a los descritos anteriormente. Cualquiera de estos errores causa la terminación inmediata del ciclo bus.

CUADRO III.5. *Vectores de excepciones*



1. El contenido del registro de estado es copiado a un registro interno.
2. Se coloca en "1" el bit S del registro de estado.
3. Se coloca en "0" el bit T del registro de estado para deshabilitar las operaciones de trace.
4. El program counter se almacena en el stack de supervisor, decrementando en cuatro el SSP.
5. El registro de estado se almacena en el stack de supervisor. El SSP se decrementa en dos.
6. El contenido del registro de instrucción del MC68000, el cual es la primera palabra de la instrucción que se estaba ejecutando al ocurrir el error, se salva en el stack de supervisor y el SSP se decrementa en dos.
7. Se almacena en el stack de supervisor la dirección de 32 bits que se estaba usando en el ciclo de bus terminado y el SSP se decrementa en cuatro.
8. Se almacena en el stack de sistema una palabra que proporciona el tipo de ciclo que se estaba ejecutando y el SSP se decrementa en dos.
9. El program counter se carga con el valor tomado del vector apropiado, ya sea el vector de error en el bus o el vector de error de dirección.
10. La ejecución continúa en la localidad indicada por el program counter.

En la figura III.24 se muestra el orden en el cual la información se almacena en el stack de supervisor para procesar una excepción de un error en el bus o de dirección.

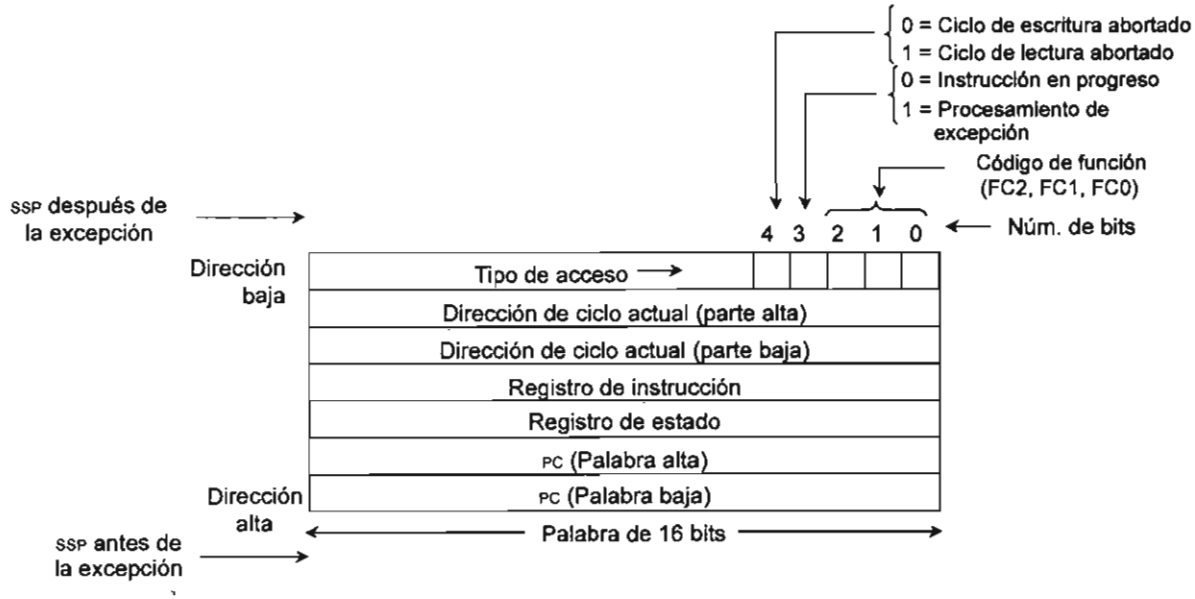


Figura III.24. Procesamiento de excepciones de error en el bus y error de dirección.

El valor salvado del program counter es la dirección de la primera palabra de la instrucción donde ocurrió el error más dos a diez bits, dependiendo de la longitud de la instrucción y de su información de direccionamiento.

Los cinco bytes menos significativos salvados en el stack indican el tipo de acceso que se estaba realizando en el momento del error en el bus o de dirección. El bit 3 indica el tipo de procesamiento que se estaba llevando a cabo al ocurrir el error.

Si ocurre un error durante el procesamiento de una excepción de un error en el bus, de un error de dirección o de una operación de RESET, el MC68000 entrará al estado HALT y permanecerá ahí.

La información almacenada en el stack de supervisor al procesar la excepción sirve para analizar las causas posibles del error.

Procesamiento de la excepción del RESET

Un RESET externo causa un procesamiento de excepción de un tipo especial, donde ocurre lo siguiente:

1. Se coloca en "1" el bit S del registro de estado.
2. Se coloca en "1" el bit T del registro de estado.
3. Se colocan en "1" los tres bits de la máscara de interrupción del registro de estado. Máscara de prioridad de interrupciones con nivel 7.
4. El stack pointer del supervisor (SSP) se carga con el contenido de los primeros cuatro bytes de memoria (000 000-000 003).
5. El program counter se carga con el contenido de los siguientes cuatro bytes (000 0004 - 000 007).
6. La ejecución comienza en la dirección indicada por el program counter, el cual debe ser el programa de inicialización de RESET/power-up.

Procesamiento de la excepción por solicitud de interrupción

Un dispositivo externo solicita una interrupción codificando un nivel de solicitud de interrupción en las entradas IPL0-IPL2.

El MC68000 compara estas entradas con la máscara de interrupción del registro de estado: si el nivel de prioridad codificado es menor o igual al especificado por la máscara, la solicitud de interrupción no será reconocida. Si el nivel de interrupción codificado es una prioridad mayor al nivel de la máscara de interrupción (o si es una solicitud de interrupción de nivel 7), entonces se procesará la interrupción.

El MC68000 responde al completar la instrucción que en ese momento está ejecutando y realiza lo siguiente:

1. Se salva internamente el contenido del registro de estado.
2. Se coloca en "1" el bit S del registro de estado.
3. Se coloca en "0" el bit T del registro de estado.
4. Los bits de la máscara de interrupción se actualizan con el nivel de la solicitud de interrupción y se codifican en las líneas IPL0-IPL2. La solicitud de interrupción se procesa sin ser interrumpida por eventos de menor prioridad.
5. El MC68000 realiza un ciclo de reconocimiento de interrupción, con lo que le indica al dispositivo solicitante que la interrupción se está atendiendo y espera un byte del dispositivo solicitante, el cual se toma como vector de excepción.

En la figura III.25 se muestra la temporización de un ciclo de reconocimiento de interrupción.

Este ciclo es ligeramente diferente a un ciclo de lectura.

Las líneas A1-A3 reflejarán los estados de las entradas IPL0-IPL2 para que una lógica externa pueda determinar cuál es la solicitud de interrupción que se está procesando. Las líneas de dirección restantes se colocan en "1".

El dispositivo solicitante debe colocar un byte que denota el vector de excepción en la parte baja

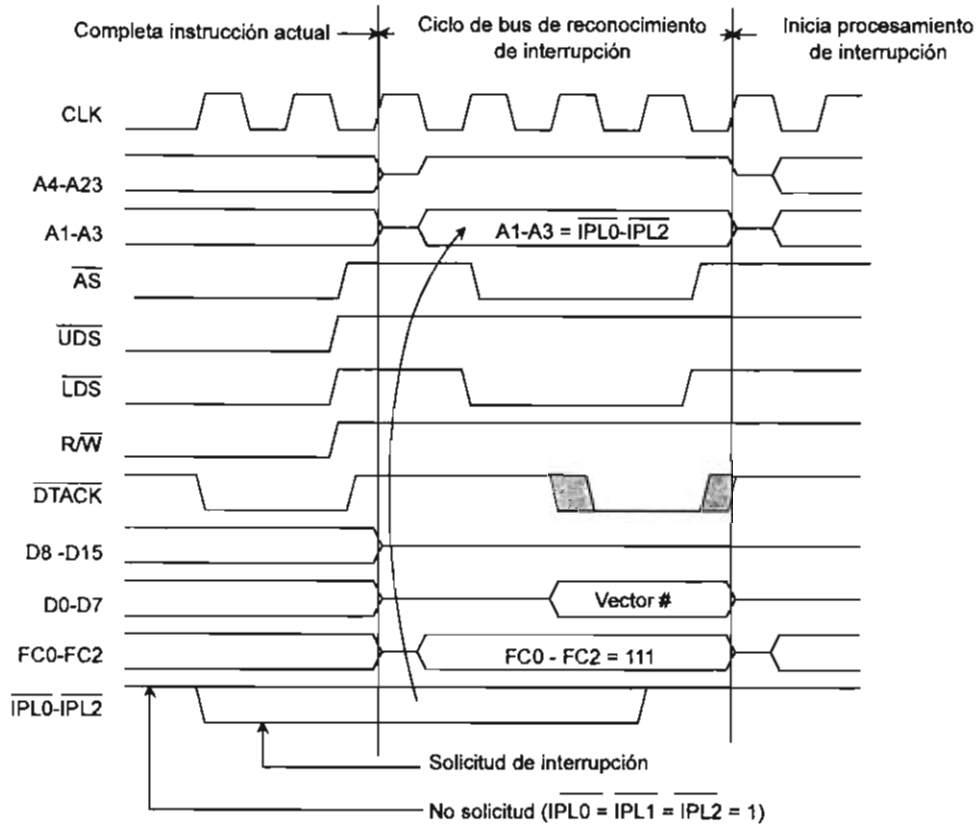


Figura III.25. Temporización de un ciclo de reconocimiento de interrupción.

del bus de datos. Se usa la señal DTACK para efectuar esta transferencia de datos como en un ciclo normal de lectura.

Durante el ciclo de reconocimiento de interrupción las salidas FC0-FC2 serán "1" lógico, lo cual representa el código de función de este reconocimiento. Una vez que el byte de vector ha sido leído, el MC68000 continúa con los siguientes pasos del procesamiento de la excepción.

6. El program counter se almacena en el stack de supervisor y el SSP se decrementa en cuatro.
7. El registro de estado se almacena en el stack de supervisor y el SSP se decrementa en dos.
8. El program counter se carga con el valor apropiado del cuadro de vectores de excepción. La dirección para este vector se obtiene como se muestra a continuación:

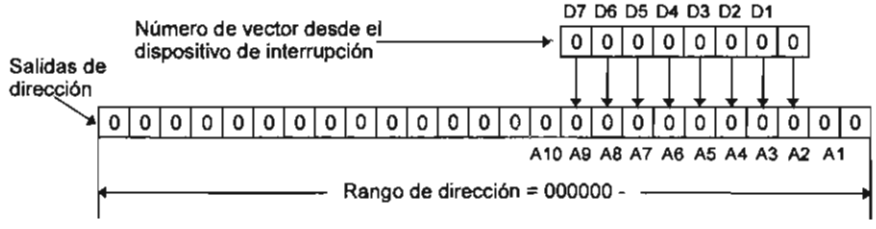


Figura III.26. El program counter cargado en la tabla de vectores de excepción.

Se pueden generar entonces las direcciones 000 000H-000 3FCH, que son los límites del cuadro de vectores.

Normalmente un dispositivo solicitante deberá generar direcciones de la 000 0FCH a la 000 3FCH, ya que las más bajas de la tabla tienen usos preasignados.

Interrupción spurious

Hay dos variaciones para procesar una solicitud de interrupción que hemos explicado anteriormente. La primera sucede si durante el reconocimiento de interrupción el dispositivo solicitante responde activando BERR en lugar de DTACK, lo que será tratado por el MC68000 como una interrupción spurious.

Respuesta a una interrupción autovector

En la tabla de vectores de excepción hay siete localidades para vectores destinadas a los siete niveles de prioridades de interrupción.

Estos vectores se usarán si el dispositivo solicitante responde en el ciclo de reconocimiento de interrupción activando la señal *valid peripheral address* (VPA) en lugar de proporcionar un vector. Si se activó VPA, el MC68000 responde activando la señal *valid memory address* (VMA).

El procesador usará el autovector de la tabla para obtener el nuevo valor del program counter. Esta característica sirve para emular la secuencia de interrupción de dispositivos periféricos de la familia Z8000 (VPA/VMA).

Modos de direccionamiento

El MC68000 tiene 14 diferentes modos de direccionamiento agrupados en seis tipos:

1. Direccionamiento de registro directo.
   a) Directo con registro de datos.
   b) Directo con registro de direcciones.
2. Direccionamiento de memoria directo.
   a) Corto absoluto.
   b) Largo absoluto.
3. Direccionamiento de memoria indirecto.
   a) Indirecto con registro.
   b) Indirecto con registro y post-incremento.
   c) Indirecto con registro y pre-decremento.
   d) Indirecto con registro y desplazamiento.
   e) Indirecto con índice y desplazamiento.
4. Direccionamiento de registro implícito.
5. Direccionamiento relativo al program counter.
   a) Relativo al PC con desplazamiento.
   b) Relativo al PC con índice y desplazamiento.
6. Direccionamiento de dato inmediato
   a) Inmediato.
   b) Inmediato rápido *(quick)*

NOTA: Cualquier registro de direcciones se puede usar para direccionamiento directo o indirecto, y cualquier registro se puede usar como registro índice.

El formato general de la palabra de operación de una instrucción con una sola dirección efectiva es el siguiente:

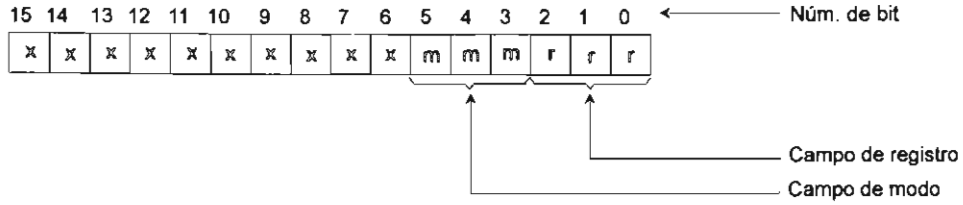


Figura III.27. Palabra de operación de una instrucción con una sola dirección efectiva.

Los dos campos menos significativos de 3 bits forman la dirección efectiva.

Extensión de la dirección efectiva

En algunos casos una o dos palabras adicionales se agregan a la instrucción para especificar completamente el operando y se llama(n) extensión de la dirección efectiva.

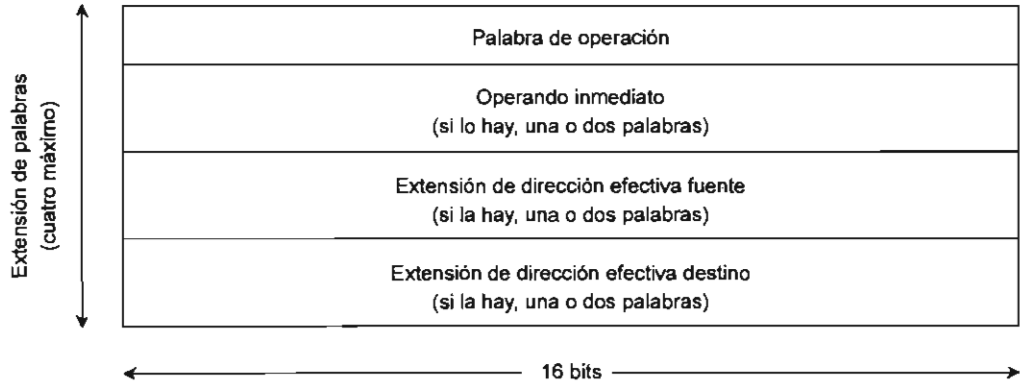


Figura III.28. Extensión de la dirección efectiva.

Es común encontrar en las explicaciones de los modos de direccionamiento del MC68000 las siguientes abreviaciones:

CUADRO III.6. *Abreviaciones utilizadas para los modos de direccionamiento*

An	*Registro de direcciones n (0-7)*
CCR	Código de condición, en el registro de estado.
dddd	Valor de desplazamiento.
Dn	Registro de datos n (0-7).
EA	Dirección efectiva (effective address).
N	Número de bytes del operando (1, 2 o 4).
PC	Program counter.
pppp	
qqqq	
xxxx	Dígitos

CUADRO III.6 *(concluye)*

An	*Registro de direcciones n (0-7)*
yyyy	Hexadecimales
zzzz	
Rn	Cualquier registro de direcciones o datos n (0-7).
rrr	El valor de 3 bits de n.
SP	El stack pointer activo.
SR	Registro de estado (status register)
SSP	Stack pointer de supervisor (supervisor stack pointer)
SSSS	Dígitos de extension de signos
USP	Stack pointer de usuario (user stack pointer)

Direccionamiento de registro directo

El operando está en uno de los ocho registros de datos o en uno de los ocho registros de direcciones (modo = 001).

Direccionamiento de dato absoluto

Hay dos formas en este modo de direccionamiento: la forma corta o direccionamiento corto absoluto y la forma larga o largo absoluto.

Corto absoluto

Necesita una palabra de extensión.

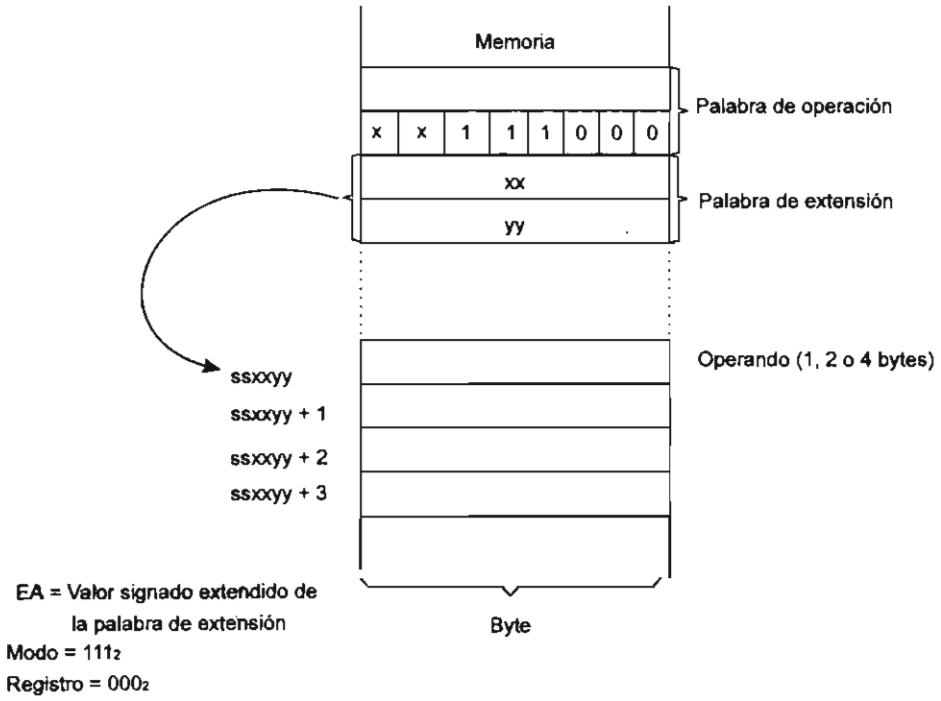


Figura III.29. Direccionamiento corto absoluto.

Largo absoluto

Necesita dos palabras de extensión y la dirección del operando es la concatenación entre ellas. La primera es la parte alta de la dirección y la segunda, la baja.

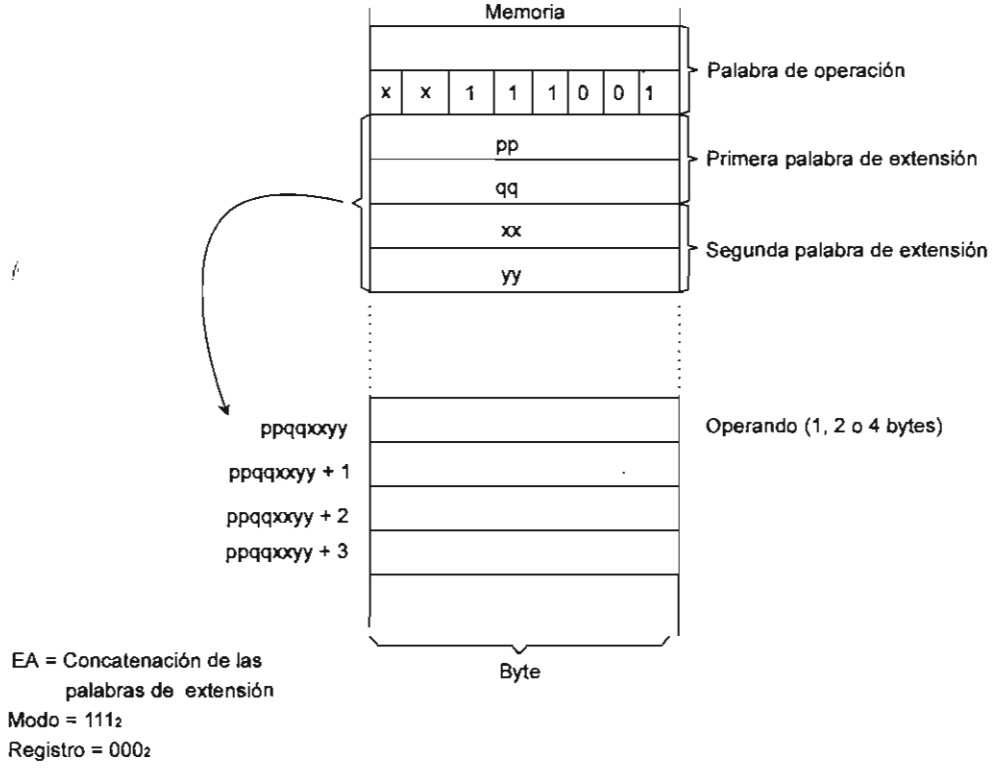


Figura III.30. Direccionamiento largo absoluto.

Direccionamiento de memoria indirecto

Los siguientes modos de direccionamiento hacen referencia a un operando en memoria.

Direccionamiento indirecto con registro

La dirección del operando está en el registro de direcciones especificado, tal como se muestra en la figura III.31.

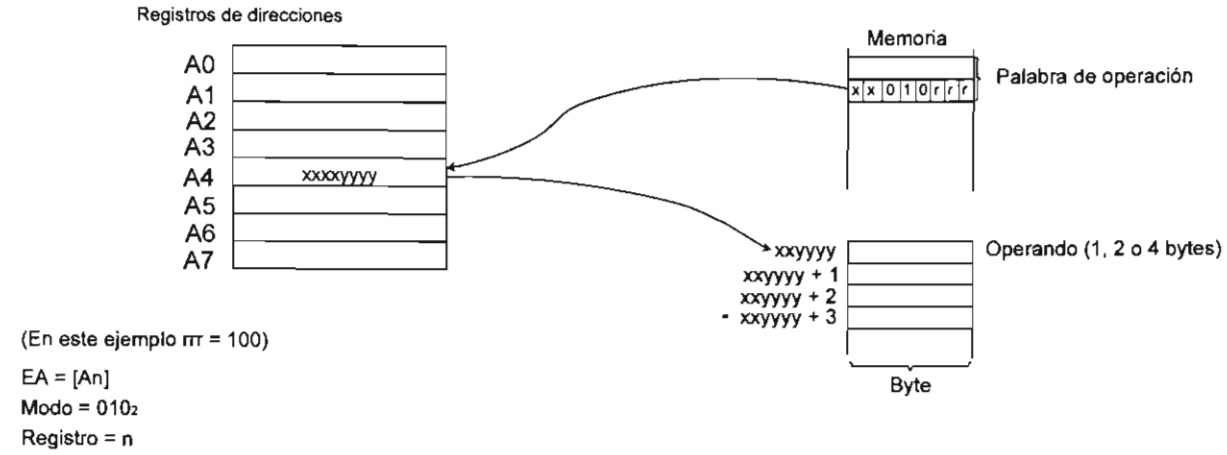


Figura III.31. Direccionamiento indirecto con registro.

Direccionamiento indirecto con registro y postincremento

La dirección del operando se encuentra en el registro de direcciones especificado y después de la ejecución de la instrucción dicho registro se incrementa en 1, 2 o 4, dependiendo del tamaño del operando.

Si el registro de dirección es A7 (SP), la dirección se incrementa en dos, independientemente del tamaño del operando, ya que el SP debe apuntar al inicio de una palabra.

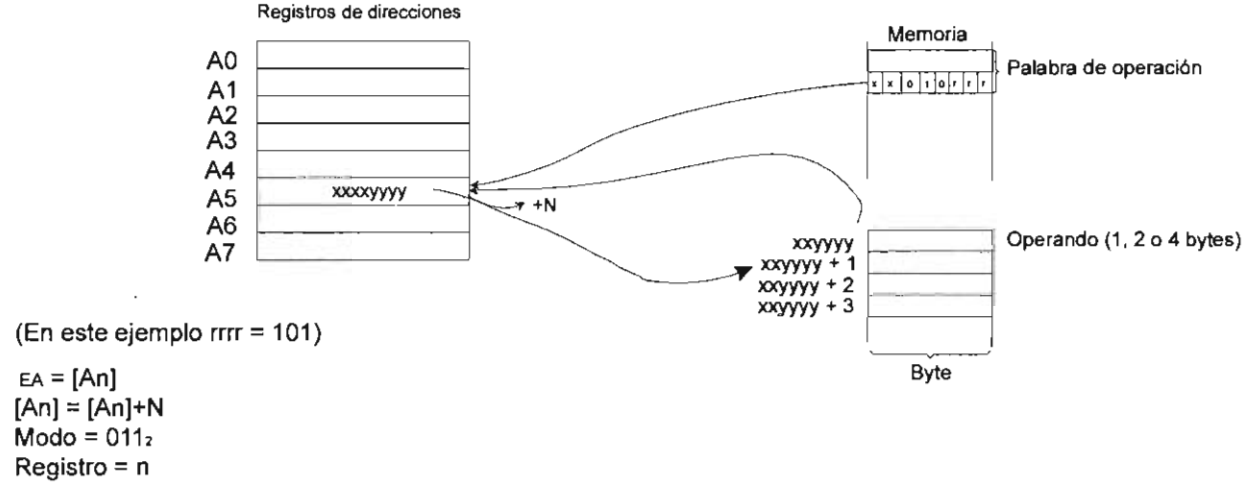
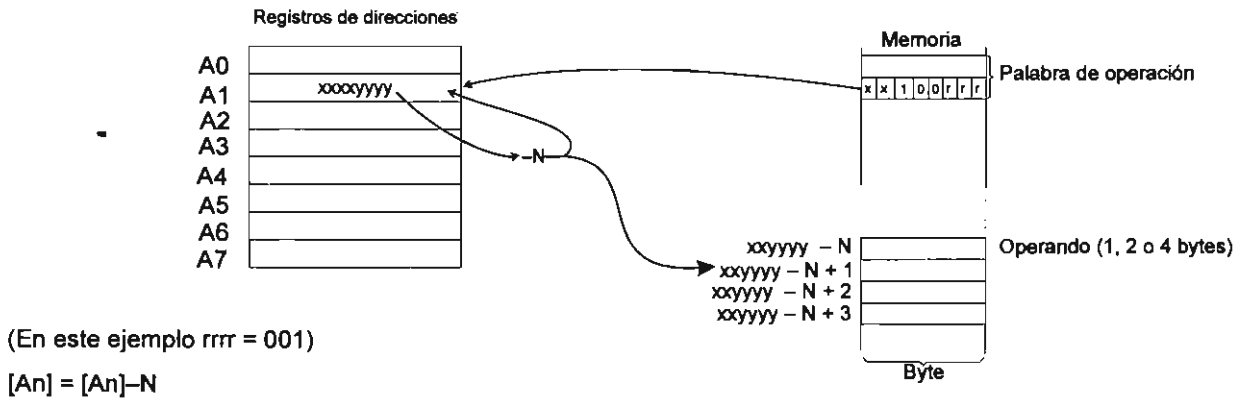


Figura III.32. Direccionamiento indirecto con registro y posincremento.

Direccionamiento indirecto con registro y predecremento

Es similar al modo anterior, pero el contenido del registro de dirección se decrementa antes de accesar el operando. Si el registro usado es A7, entonces la dirección se decrementa en dos.



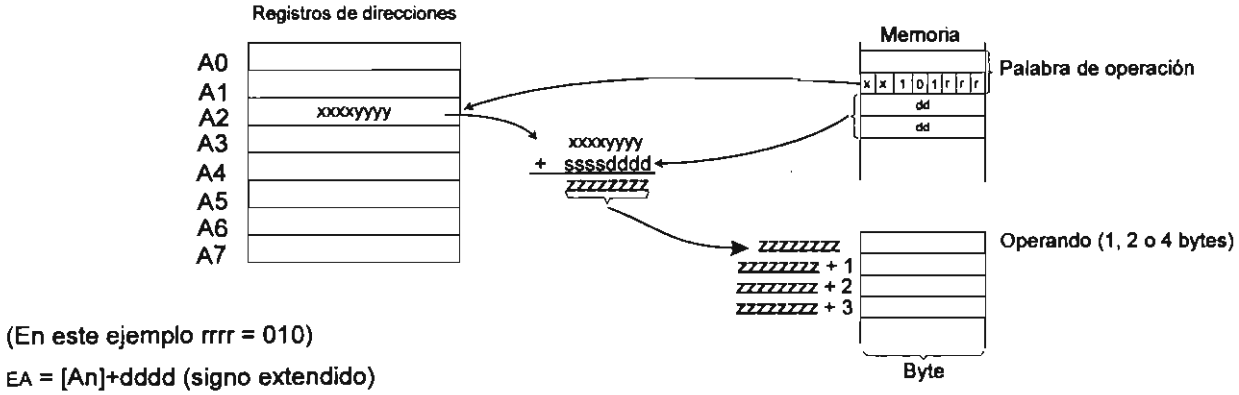
(En este ejemplo rrrr = 001)

[An] = [An]–N
EA = [An]
Modo = 100_2
Registro = n

Figura III.33. Direccionamiento indirecto con registro y predecremento.

Direccionamiento indirecto con registro y desplazamiento

Requiere una palabra de extensión. La dirección del operando es la suma del registro de dirección especificado y el desplazamiento (una palabra extendida y signada) indicado en la palabra de extensión.



(En este ejemplo rrrr = 010)

EA = [An]+dddd (signo extendido)

Modo = 101_2
Registro = n

Figura III.34. Direccionamiento indirecto con registro y desplazamiento.

Direccionamiento indirecto con índice y desplazamiento

Requiere una palabra de extensión con el siguiente formato:

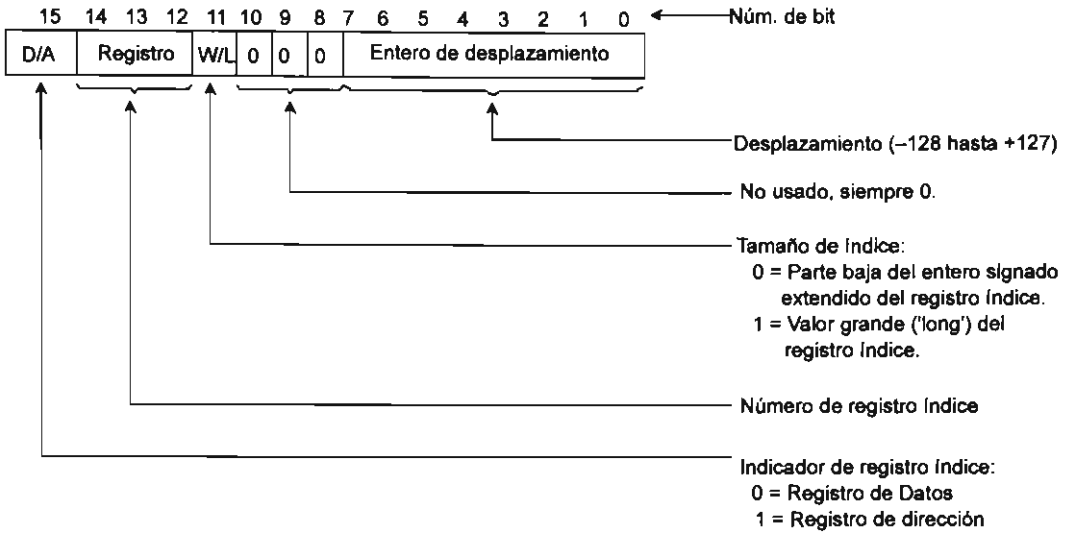


Figura III.35. Direccionamiento indirecto con índice y desplazamiento.

La dirección del operando es la suma del registro de dirección indicado, el entero del desplazamiento extendido y signado (en el byte menos significativo de la palabra de extensión) y el contenido del registro índice, como se muestra en la figura III.36.

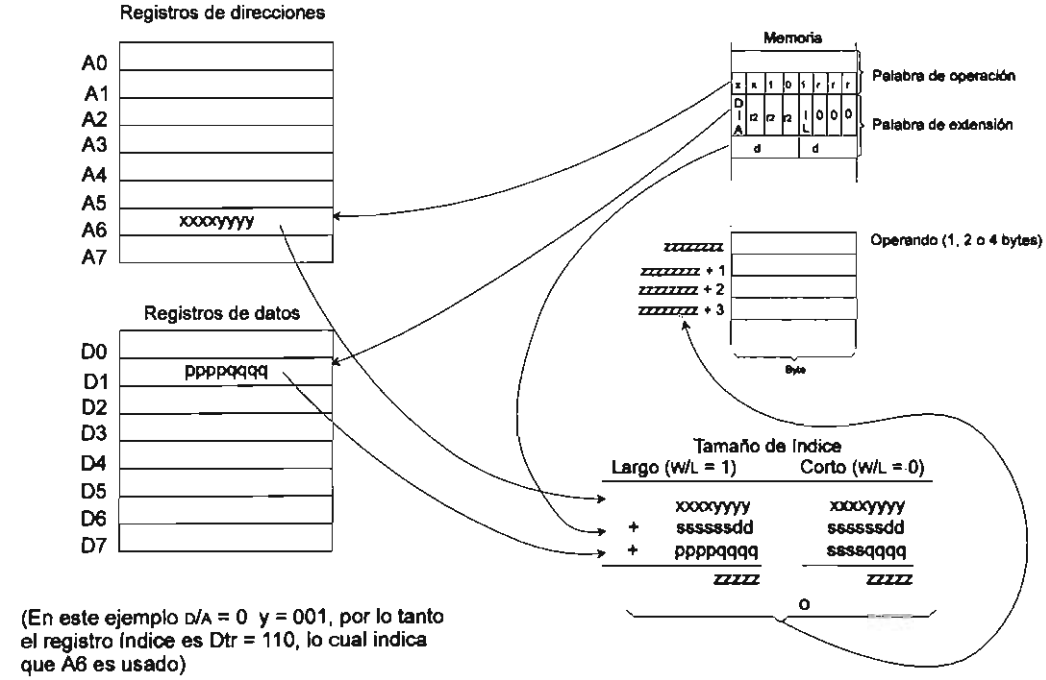


Figura III.36. Ejemplo de direccionamiento indirecto con índice y desplazamiento.

Direccionamiento de registro implícito

Algunas instrucciones implícitamente usan un registro específico, tal como el program counter (PC), stack pointer (SP-SSP o USP) y el registro de estado (SR). El cuadro III.7 muestra este tipo de instrucciones:

CUADRO III.7. *Instrucciones usadas en el direccionamiento de registro implícito*

Instrucción	*Registro(s) implicado*
Brinco condicional (Bcc), brinca siempre (BRA)	PC
Brinca a subrutina (BSR)	PC, SP
Checa registro en límites (CHK)	SSP, SR
Prueba condición, decrementa y brinca (DBcc)	PC
División signada (DIVS)	SSP, SR
División no signada (DIVU)	SSP, SR
Salto (JMP)	PSC
Salto a subrutina (JSR)	PC, SP
Liga y localiza (LINK)	SP
Mueve códigos de condición (MOVE CCR)	SR
Mueve registro de estado (MOVE SR)	SR
Mueve stack pointer del usuario (MOVE USP)	USP
Push a la dirección efectiva (PEA)	SP
Regresa de excepción	PC, SP, SR
Regresa y restaura códigos de condición (RTR)	PC, SP, SR
Regresa de subrutina (RTS)	PC, SP
Trampa (TRAP)	SSP, SR
Trampa por sobreflujo (TRAPV)	SSP, SR
Desliga (UNLK)	SP

Direccionamiento relativo al program counter

Hay dos formatos en este modo de direccionamiento; ambos requieren una palabra de extensión y proporcionan un desplazamiento. El segundo formato incluye además un índice.

El valor del program counter usado para el cálculo de la dirección es la dirección de la palabra de extensión.

Direccionamiento relativo al PC con desplazamiento

Este modo genera una dirección sumando el valor del program counter y el valor extendido en signo (o signado) de la palabra de extensión.

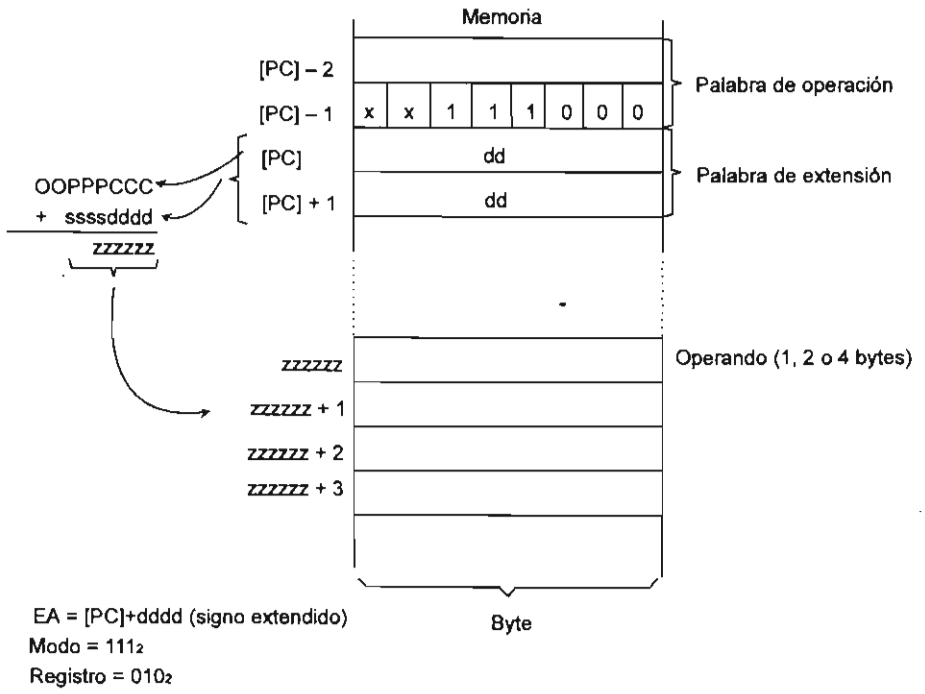


Figura III.37. Direccionamiento relativo al PC con desplazamiento.

Relativo al PC con índice y desplazamiento

Requiere una palabra de extensión con formato similar al modo indirecto con índice y desplazamiento. La dirección se calcula como se muestra en la figura III.38.

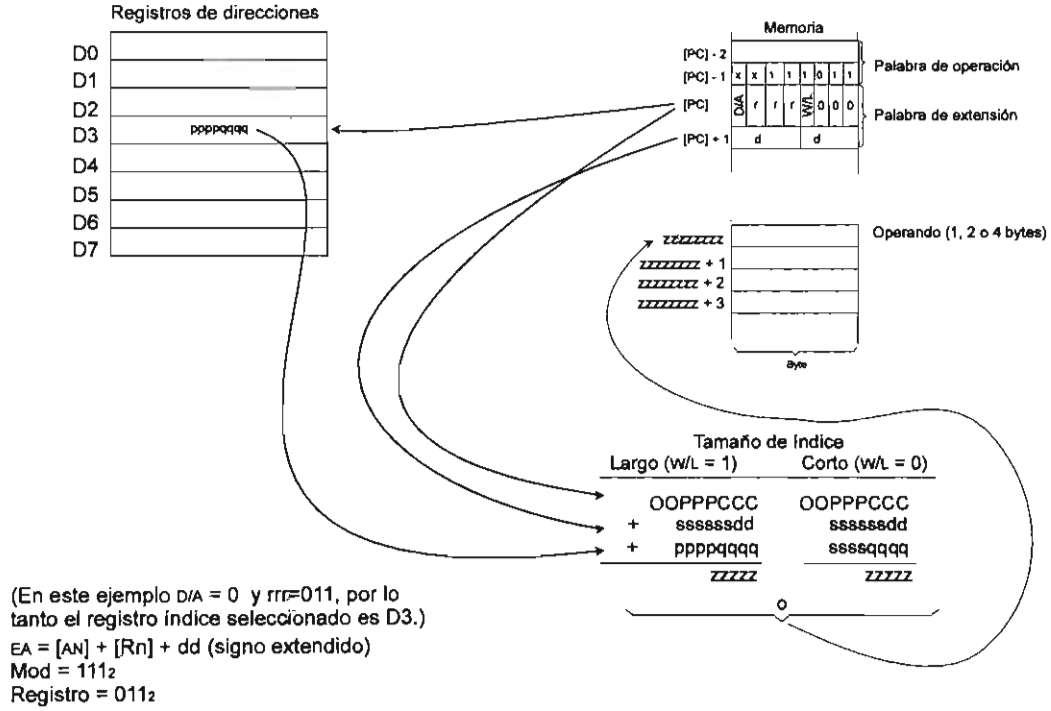
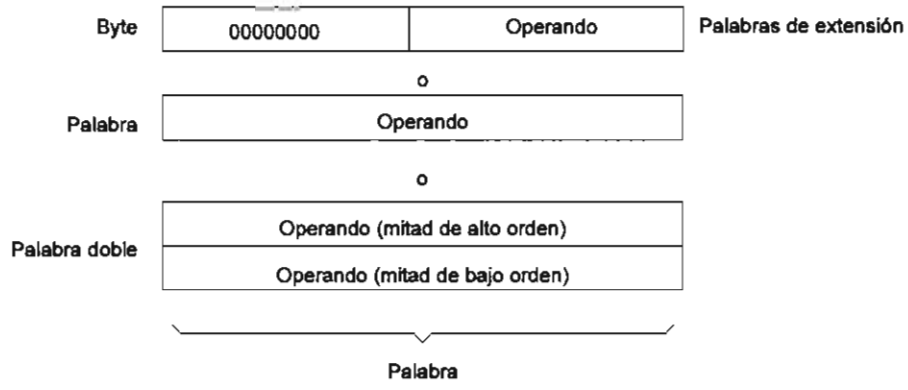


Figura III.38. Direccionamiento relativo al PC con índice y desplazamiento.

Direccionamiento de dato inmediato

El operando es el valor que sigue inmediatamente a la palabra de instrucción. Dependiendo del tamaño del operando, serán necesarias una o dos palabras de extensión, como se muestra en la figura III.39.



EA = No requerida; el operando es parte de la instrucción.
Modo = 111_2
Registro = 100_2

Figura III.39. Direccionamiento de dato inmediato.

Interface de un MC68000 con periféricos de la familia 6800

La mayoría de los periféricos desarrollados por Motorola y otros fabricantes son para microprocesadores de 8 bits (entre ellos el 6800). Se puede conectar cualquier periférico asíncrono al MC68000 con una pequeña lógica externa. Los periféricos de la familia 6800 son síncronos y el MC68000 es un procesador asíncrono que tiene tres señales para operaciones de lectura/escritura síncronas con dispositivos de la familia 6800: valid memory address (VMA), valid peripheral address (VPA) y enable (E).

Señales de los periféricos de la familia 6800

La lógica externa debe decodificar la información del bus de direcciones para saber si el MC68000 está direccionando un periférico de la familia 6800 después de activar AS.

En caso afirmativo la lógica externa debe activar la entrada VPA del MC68000, de tal forma que éste emule una transferencia de datos del microprocesador 6800. Dicha transferencia se sincroniza con la señal de reloj E.

Nótese que si es una operación de lectura, el dispositivo 6800 debe depositar el dato en el bus cuando E sea "1" lógico. No se usa DTACK, ya que no es una transferencia asíncrona.

Al terminar de leer el dato el MC68000 (bajada de la señal E) desactiva AS y lleva el bus a alta impedancia.

NOTAS: En este ejemplo un ciclo de lectura dura cuatro ciclos de CLK más que el de escritura, pero no siempre es así, ya que realmente la operación comienza y está sincronizada al activarse E.

El duty-cycle de E es de 40 por ciento:

"1" lógico-4 periodos de CLK
"0" lógico-6 periodos de CLK

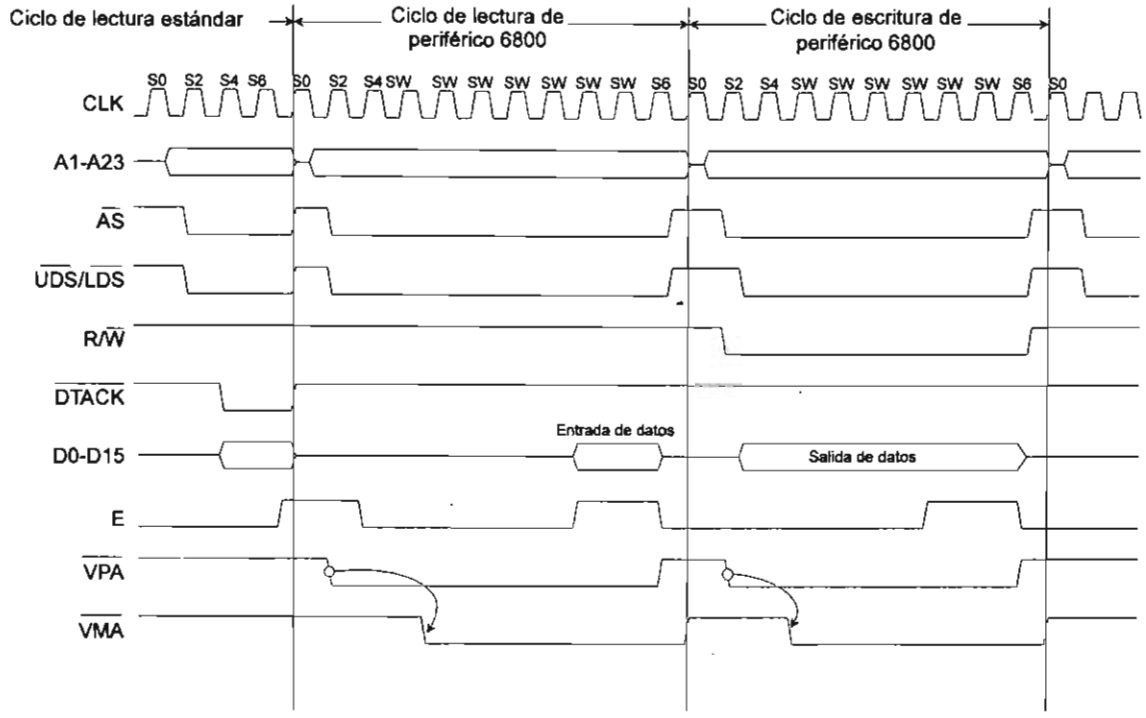


Figura III.40. Señales de los periféricos de la familia 6800.

Las instrucciones del MC68000 varían en el número de periodos de CLK necesarios para su ejecución.

Las señales E y VMA están sincronizadas con la ejecución de la instrucción, tal que en algunas instrucciones, después de activarse VPA, el MC68000 inserte automáticamente estados wait.

Al final del ciclo, el periférico o la lógica externa deben desactivar VPA dentro de un periodo de reloj después de que el MC68000 desactiva AS para evitar que el MC68000 también interprete al siguiente ciclo como un ciclo síncrono del tipo 6800.

Ejemplo:

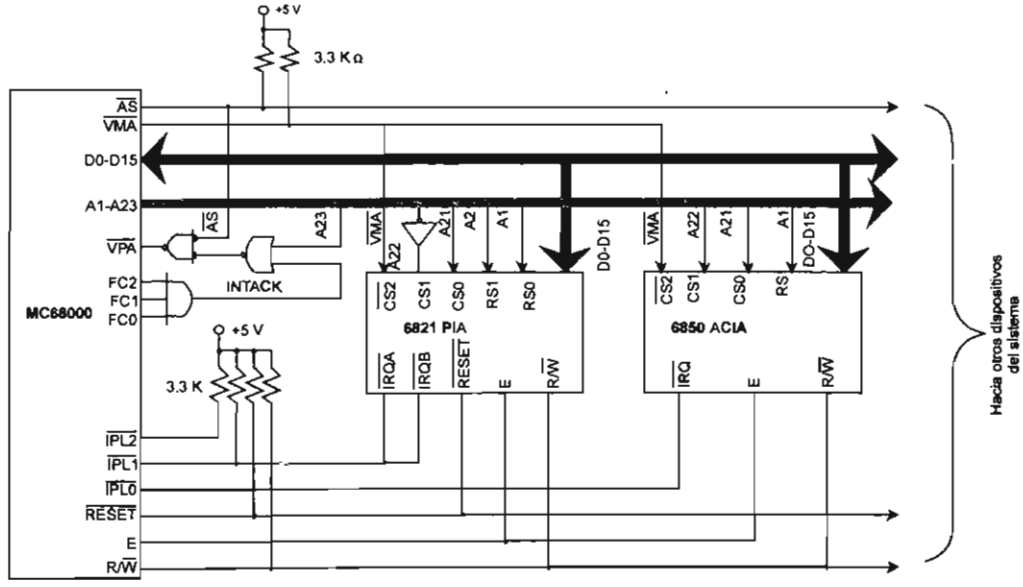


Figura III.41. Circuito ejemplo de un sistema mínimo usando el MC68000.

En este ejemplo se deja el espacio comprendido entre las localidades 000 000H-7FF FFFH para dispositivos asíncronos (incluyendo memoria) y se mapean los dispositivos síncronos del tipo 6800 de la siguiente manera:

PIA	*(peripheral interface adapter)*	-A00 000H a la B00 000H
ACIA	*(asynchronous communications interface adapter)*	-E00 000H a la F00 000H

En el caso de las solicitudes de interrupción, cuando el ACIA activa su salida IRQ genera una solicitud con nivel 1 en el MC68000. Si alguna salida IRQ del PIA se activa, se genera una solicitud con nivel 2 en el MC68000, ya que la entrada IPL2 del procesador está desactivada.

Finalmente, al reconocer el MC68000 la solicitud de interrupción, se activa la entrada VPA, lo que causa que el procesador no solicite un vector del dispositivo y obtenga el autovector de la tabla de vectores para procesar excepciones.

BIBLIOGRAFÍA

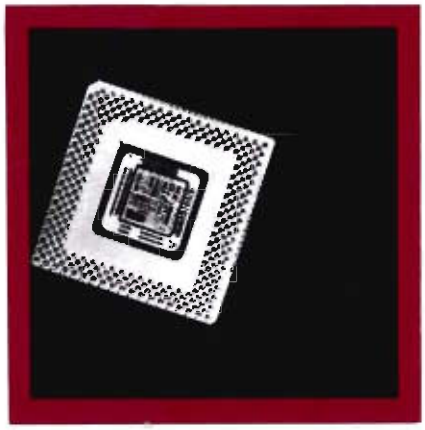
Brey, Barry B., *Microprocessors and Peripherals,* Maxwell Macmillan International Editions.
Giles, William B., *Assembly Language Programming,* Maxwell Macmillan International Editions.
Intel Corporation, *Microcontroller Handbook.*
Intel Corporation, *The 8086 Family User's Manual.*
Kane, Gerry, *6800 Microprocessor Handbook,* Osborne/McGraw Hill.
Scanlon, Leo J., *8086/88 Assembly Language Programming,* Prentice-Hall.
Uffenbeck, John, *The 8086/88 Family,* Prentice-Hall.

ÍNDICE

Organización de máquinas digitales I
se terminó de imprimir en enero de 1998
en los talleres de Editorial Ducere, S.A. de C.V.,
Rosa Esmeralda 3 bis,
col. Molino de Rosas, 01470 México, D.F.
El tiro consta de 1 000 ejemplares más sobrantes
para reposición.
La composición tipográfica, la formación y el cuidado
editorial estuvieron a cargo de Sans Serif Editores,
S.A. de C.V., telfax 674 60 91.

Otros títulos en esta colección

Patrick Staelens, *El trabajo de los menores*
Luis Rodríguez, *El diseño preindustrial*
José Dolores Juárez Cervantes, *Sistemas de distribución de energía eléctrica*
Rafael Quintero, *Electrónica física*
Adalberto Cantú Chapa, *Análisis de diseño con diodos y transistores*
Mariem Henaine-Abed, *Planeación y control de la producción*
José Vega Luna y Gerardo Salgado Guzmán, *Prácticas de laboratorio de sistemas digitales*
Ana Lilia Laureano, *Programación orientada a objetos: un enfoque con tipos abstractos*
Juan González Márquez, *Introducción al derecho bancario mexicano*
Violeta Múgica y José de Jesús Figueroa, *Contaminación ambiental, causas y control*
Fernando Toledo Toledo, *Métodos computacionales para el análisis de sistemas de potencia*
Raymundo López Callejas, Juan Ramón Morales Gómez, Mabel Vaca Mier, Araceli Lara Valdivia y David Sandoval Cardoso, *Problemario de mecánica de fluidos*
Tomás David Navarrete González y José Ángel Rocha Martínez, *Colección de problemas resueltos para el curso de energías mecánica y eléctrica*
Rafael López Bracho, María Paula Ortuño Sánchez, Felipe Carrillo Romero y María Teresa Rodríguez Martínez, *Paquete computacional "Optimización en redes (versión 2.0) para Windows"*



Organización de máquinas digitales contiene la teoría de los microprocesadores de 16 bits, el 8086 de Intel y el 68000 de Motorola, así como el microcontrolador de 8 bits 8051 de Intel. En esta obra se trata sobre la arquitectura interna de estos dispositivos y se ofrecen algunos ejemplos de programación. La obra también contiene ejemplos de sistemas multiprocesador. Por su contenido y tratamiento es una excelente guía para estudiantes de ingeniería electrónica, ingeniería en sistemas e ingeniería en computación; se recomienda su lectura después de conocer los fundamentos de operación de los microprocesadores.