Microprocesadores de la línea Intel

por Darío Alejandro Alpern

Ejemplos de programas en assembler

El microprocesador 4004

Historia del 4004

En 1969, Silicon Valley, en el estado de California (EEUU) era el centro de la industria de los semiconductores. Por ello, gente de la empresa Busicom, una joven empresa japonesa, fue a la compañía Intel (fundada el año anterior) para que hicieran un conjunto de doce chips para el corazón de su nueva calculadora de mesa de bajo costo.

Al principio se pensó que no se podía hacer, ya que Intel no estaba preparada para realizar circuitos "a medida". Pero Marcian Edward *Ted* Hoff, Jr., jefe del departamento de investigación de aplicaciones, pensó que habría una mejor forma de realizar el trabajo.

Durante el otoño (del hemisferio norte) de 1969 Hoff, ayudado por Stanley Mazor, definieron una arquitectura consistente en una CPU de 4 bits, una memoria ROM (de sólo lectura) para almacenar las instrucciones de los programas, una RAM (memoria de lectura y escritura) para almacenar los datos y algunos puertos de entrada/salida para la conexión con el teclado, la impresora, las llaves y las luces. Además definieron y verificaron el conjunto de instrucciones con la ayuda de ingenieros de Busicom (particularmente Masatoshi Shima).

En abril de 1970 Federico Faggin se sumó al staff de Intel. El trabajo de él era terminar el conjunto de chips de la calculadora. Se suponía que Hoff y Mazor habían completado el diseño lógico de los chips y solamente quedarían definir los últimos detalles para poder comenzar la producción. Esto no fue lo que Faggin encontró cuando comenzó a trabajar en Intel ni lo que Shima encontró cuando llegó desde Japón.

Shima esperaba revisar la lógica de diseño, confirmando que Busicom podría realizar su calculadora y regresar a Japón. Se puso furioso cuando vio que estaba todo igual que cuando había ido seis meses antes, con lo que dijo (en lo poco que sabía de inglés) "Vengo acá a revisar. No hay nada para revisar. Esto es sólo idea". No se cumplieron los plazos establecidos en el contrato entre Intel y Busicom.

De esta manera, Faggin tuvo que trabajar largos meses, de 12 a 16 horas por día.

Finalmente pudo realizar los cuatro chips arriba mencionados. El los llamó "familia 4000". Estaba compuesto por cuatro dispositivos de 16 pines: el 4001 era una ROM de dos kilobits con salida de cuatro bits de datos; el 4002 era una RAM de 320 bits con el port de entrada/salida (bus de datos) de cuatro bits; el 4003 era un registro de desplazamiento de 10 bits con entrada serie y salida paralelo; y el 4004 era la CPU de 4 bits.

El 4001 fue el primer chip diseñado y terminado. La primera fabricación ocurrió en octubre de 1970 y el circuito trabajó perfectamente. En noviembre salieron el 4002 con un pequeño error y el 4003 que funcionó correctamente. Finalmente el 4004 vino unos pocos días antes del final de 1970. Fue una lástima porque en la fabricación se habían olvidado de poner una de las máscaras. Tres semanas después vinieron los nuevos 4004, con lo que Faggin pudo realizar las verificaciones. Sólo encontró unos pequeños errores. En febrero de 1971 el 4004 funcionaba correctamente. En el mismo mes recibió de Busicom las instrucciones que debían ir en la ROM.

A mediados de marzo de 1971, envió los chips a Busicom, donde verificaron que la calculadora funcionaba perfectamente. Cada calculadora necesitaba un 4004, dos 4002, cuatro 4001 y tres 4003. Tomó un poco menos de un año desde la idea al producto funcionando correctamente.

Luego de que el primer microprocesador fuera una realidad, Faggin le pidió a la gerencia de Intel que utilizara este conjunto de chips para otras aplicaciones. Esto no fue aprobado, pensando que la familia 4000 sólo serviría para calculadoras. Además, como fue producido mediante un contrato exclusivo, sólo lo podrían poner en el mercado teniendo a Busicom como intermediario.

Después de hacer otros dispositivos utilizando la familia 4000, Faggin le demostró a Robert Noyce (entonces presidente de Intel) la viabilidad de estos integrados para uso general. Finalmente ambas empresas llegaron a un arreglo: Intel le devolvió los 60000 dólares que había costado el proyecto, sólo podría vender los integrados para aplicaciones que no fueran calculadoras y Busicom los obtendría más baratos (ya que se producirían en mayor cantidad).

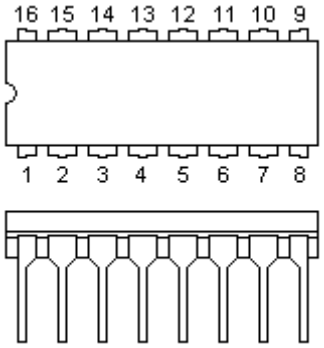
El 15 de noviembre de 1971, la familia 4000, luego conocida como **MCS-4** (*Micro Computer System 4-bit*) fue finalmente introducida en el mercado.

Descripción del 4004

Es un microprocesador de 4 bits de bus de datos, direcciona 32768 bits de ROM y 5120 bits de RAM. Además se pueden direccionar 16 ports de entrada (de 4 bits) y 16 ports de salida (de 4 bits). Contiene alrededor de 2300 transistores MOS de canal P de 10 micrones. El ciclo de instrucción es de 10,8 microsegundos.

Terminales del 4004

Este microprocesador estaba encapsulado en el formato **DIP** (*Dual Inline Package*) de 16 patas (ocho de cada lado). La distancia entre las patas es de 0,1 pulgadas (2,54 milímetros), mientras que la distancia entre patas enfrentadas es de 0,3 pulgadas (7,68 milímetros).



Nótese en el gráfico de la derecha el semicírculo que identifica la posición de la pata 1. Esto sirve para no insertar el chip al revés en el circuito impreso.

Las funciones de las 16 patas con las que se conecta el 4004 con el exterior son las siguientes:

Pata	Nombre	Descripción
1	**D_0**	Todas las direcciones y datos de RAM y ROM pasan por estas líneas
2	**D_1**	
3	**D_2**	
4	**D_3**	
5	**V_{SS}**	Referencia de tierra. Es la tensión más positiva.
6	**Clock phase 1**	Son las dos fases de entrada de reloj (*clock*)
7	**Clock phase 2**	
8	**Sync output**	Señal de sincronismo generada por el procesador. Indica el comienzo de un ciclo de instrucción.
9	**Reset**	Un "1" lógico aplicado en esta pata borra todos los flags y registros de estado y fuerza el contador de programa (PC) a cero. Para que actúe correctamente, esta línea deberá activarse por 64 ciclos de reloj (8 ciclos de máquina).
10	**Test**	La instrucción **JCN** verifica el estado de esta línea.
11	**CM-ROM (*Control Memory Outputs*)**	Esta señal está activa cuando el procesador necesita datos de la ROM
12	**V_{DD}**	Alimentación del microprocesador. La tensión debe ser de -15V +/- 5%
13	**CM-RAM_3**	Éstas son las señales de selección de banco para indicar a cuál RAM 4002 desea acceder el microprocesador
14	**CM-RAM_2**	
15	**CM-RAM_1**	
16	**CM-RAM_0**	

Instrucciones del 4004

Hay instrucciones de uno o dos bytes. Los primeros tardan 8 períodos de reloj (un ciclo de instrucción). Los segundos tardan 16 períodos de reloj (dos ciclos de instrucción).

Mnemónico	Descripción	OPR				OPA			
		D_3	**D_2**	**D_1**	**D_0**	**D_3**	**D_2**	**D_1**	**D_0**
NOP	No hace nada	0	0	0	0	0	0	0	0
	Salta a la dirección especificada por A_2 A_2 A_2 A_2 A_1 A_1 A_1 A_1 dentro de la misma ROM que contiene esta instrucción **JCN**, si se cumple la condición C_1 C_2 C_3 C_4, en caso contrario continúa ejecutando la próxima	0	0	0	1	C_1	C_2	C_3	C_4

JCN	instrucción. C_1=1: Invertir la condición de salto. C_2=1: Saltar si el acumulador es cero. C_3=1: Saltar si el acarreo vale uno. C_4=1: Saltar si la pata **TEST** está a cero.	0	0	1	0	R	R	R	0
		A_2	A_2	A_2	A_2	A_1	A_1	A_1	A_1
FIM	Cargar el dato D_2, D_1 (ocho bits) en el par de registros RRR	0	0	1	0	R	R	R	0
		D_2	D_2	D_2	D_2	D_1	D_1	D_1	D_1
SRC	Enviar la dirección (contenido del par de registros RRR) a la ROM y a la RAM en los tiempos X_2 y X_3 del ciclo de instrucción	0	0	1	0	R	R	R	1
FIN	Cargar en el par de registros RRR el dato de ROM apuntado por el par de registros cero	0	0	1	1	R	R	R	0
JIN	Salto indirecto según el par de registros RRR	0	0	1	1	R	R	R	1
JUN	Salto incondicional a la dirección de ROM A_3, A_2, A_1	0	1	0	0	A_3	A_3	A_3	A_3
		A_2	A_2	A_2	A_2	A_1	A_1	A_1	A_1
JMS	Salvar el viejo valor del contador de programa y saltar a la dirección de ROM A_3, A_2, A_1	0	1	0	1	A_3	A_3	A_3	A_3
		A_2	A_2	A_2	A_2	A_1	A_1	A_1	A_1
INC	Incrementar el contenido del registro RRRR	0	1	1	0	R	R	R	R
ISZ	Incrementar el registro RRRR. Si el resultado no es cero, saltar a la dirección A_2 A_2 A_2 A_2 A_1 A_1 A_1 A_1	0	1	1	1	R	R	R	R
	dentro de la misma ROM que contiene esta instrucción **ISZ**	A_2	A_2	A_2	A_2	A_1	A_1	A_1	A_1
ADD	Sumar el registro RRRR al acumulador con acarreo	1	0	0	0	R	R	R	R
SUB	Restar el registro RRRR del acumulador con préstamo	1	0	0	1	R	R	R	R
LD	Cargar el acumulador con el contenido del registro RRRR	1	0	1	0	R	R	R	R
XCH	Intercambiar los contenidos del acumulador y el registro RRRR	1	0	1	1	R	R	R	R
BBL	Retornar de subrutina y cargar el dato D D D D en el acumulador	1	1	0	0	D	D	D	D
LDM	Cargar el dato D D D D en el acumulador	1	1	0	1	D	D	D	D

Las siguientes instrucciones operan sobre las direcciones de RAM y ROM especificadas en la última instrucción SRC:

Cada chip de RAM tiene cuatro registros, cada uno con veinte caracteres de 4 bits subdivididos en 16 caracteres de memoria principal y 4 de estado. El número de chip, registro de RAM y carácter de memoria principal se selecciona mediante la instrucción SRC, mientras que los caracteres de estado (dentro de un registro) se seleccionan mediante el código de instrucción (OPA)

Mnemónico	Descripción	OPR	OPA
WRM	Escribir el acumulador en RAM	1110	0000
WMP	Escribir el acumulador en port de salida de RAM	1110	0001
WRR	Escribir el acumulador en port de salida de ROM	1110	0010
WPM	Escribir el acumulador en el medio byte especificado de RAM (se usa en los microprocesadores 4008 y 4009 solamente)	1110	0011
WR0	Escribir el acumulador en el carácter de estado de RAM 0, 1, 2, 3	1110	0100
WR1		1110	0101
WR2		1110	0110
WR3		1110	0111
SBM	Restar el contenido de la posición previamente especificada de RAM del acumulador con préstamo	1110	1000
RDM	Cargar en el acumulador el contenido de la posición de RAM	1110	1001
RDR	Cargar en el acumulador el contenido del port de entrada de ROM	1110	1010
ADM	Sumar el contenido de la posición previamente especificada de RAM al acumulador con acarreo	1110	1011
RD0	Almacenar en el acumulador el carácter de estado de RAM 0, 1, 2, 3	1110	1100
RD1		1110	1101
RD2		1110	1110
RD3		1110	1111

La siguiente tabla muestra el grupo de instrucciones del acumulador.

Mnemónico	Descripción	OPR	OPA
CLB	Limpiar el acumulador y el acarreo	1111	0000
CLC	Limpiar el indicador de acarreo	1111	0001
IAC	Incrementar el acumulador	1111	0010
CLC	Complementar el acarreo	1111	0011
CMA	Complementar el acumulador	1111	0100
RAL	Rotar acumulador y acarreo hacia la izquierda	1111	0101
RAR	Rotar acumulador y acarreo hacia la derecha	1111	0110
TCC	Sumar acarreo al acumulador y limpiar el acarreo	1111	0111
DAC	Decrementar el acumulador	1111	1000
TCS	Restar acarreo del acumulador y limpiar el acarreo	1111	1001
STC	Poner el acarreo a uno	1111	1010
DAA	Ajuste decimal del acumulador	1111	1011
KBP	Convierte un código 1 de 4 a binario en el acumulador	1111	1100
DCL	Designar línea de comando	1111	1101

El microprocesador 8008

Historia del 8008

En 1969 Computer Terminal Corp. (ahora Datapoint) visitó Intel. *Vic* Poor, vicepresidente de Investigación y Desarrollo en CTC quería integrar la CPU (unos cien componentes TTL) de su nueva terminal Datapoint 2200 en unos pocos chips y reducir el costo y el tamaño del circuito electrónico.

Ted Hoff observó la arquitectura, el conjunto de instrucciones y el diseño lógico que había presentado CTC y estimó que Intel podría integrarlo en un sólo chip, así que Intel y CTC firmaron un contrato para desarrollar el chip. El chip, internamente llamado 1201, sería un dispositivo de 8 bits. Pensado para la aplicación de terminal inteligente, debería ser más complejo que el 4004.

Al principio parecía que el 1201 saldría antes que el 4004 ya que Federico Faggin tenía que desarrollar cuatro chips, siendo el 4004 el último de ellos. Sin embargo, después de algunos meses de trabajo con el 1201, el diseñador, *Hal* Feeney, fue puesto a diseñar un chip de memoria, con lo que el proyecto del 1201 fue puesto en el "freezer".

Mientras tanto, CTC también contrató a la empresa Texas Instruments para hacer el diseño del mismo chip como fuente alternativa. Al final de 1970 Intel continuó con el proyecto del 1201 bajo la dirección de Faggin y Feeney fue puesto nuevamente a trabajar en este proyecto.

En junio de 1971, TI puso un aviso en la revista Electronics donde se detallaban las capacidades de este integrado MOS LSI. Con la leyenda "CPU en un chip" se acompañaba la descripción del circuito a medida para la terminal Datamation 2200. El aviso decía "TI lo desarrolló y lo está produciendo para Computer Terminal Corp.". Las dimensiones indicadas eran 5,46 por 5,71 mm, un chip enorme aun para la tecnología de 1971 y era 225% más grande que el tamaño estimado por Intel.

El chip de Texas Instruments, sin embargo, jamás funcionó y no se puso en el mercado. Sorprendentemente, TI patentó la arquitectura del 1201, que fue realizado por CTC con algunos cambios de Intel, con lo que luego hubo batallas legales entre Intel y TI.

Durante el verano (en el hemisferio norte) de 1971, mientras el trabajo con el 1201 estaba progresando rápidamente, Datapoint decidió que no necesitaba más el 1201. La recesión económica de 1970 había bajado el costo de los circuitos TTL de tal manera que ya no era rentable el circuito a medida. Datapoint le dejó usar la arquitectura a Intel y a cambio la última no le cobraba más los costos de desarrollo.

Intel decidió cambiarle el nombre al 1201: se llamaría **8008**. El primero de abril de 1972 se lanzó este microprocesador al mercado con un conjunto de chips de soporte, como una familia de productos llamado **MCS-8**. Estos chips de soporte eran integrados existentes con los nombres cambiados. El interés del mercado por el MCS-8 fue muy alto, sin embargo las ventas fueron bajas.

Para solucionar este inconveniente, se diseñaron herramientas de hardware y software, entrenamiento y sistemas de desarrollo. Estos últimos son computadoras especializadas para desarrollar y depurar programas (quitarles los errores) para el microprocesador específico. Un año después, Intel recibía más dinero de los sistemas de desarrollo que de los microprocesadores y chips de soporte.

A título informativo cabe destacar que este microprocesador de ocho bits poseía alrededor de 3500 transistores, direccionaba 16 KBytes y la frecuencia máxima de reloj (*clock*) era de 108 KHz.

Conjunto de registros del 8008

Este conjunto de registros forma la base para comprender el conjunto de registros de los siguientes procesadores, ya que se basan en éste. Éstos son:

Clasificación	Registro	Longitud (en bits)
Acumulador	**A**	8
Registros de uso general	**B**	8
	C	8
	D	8
	E	8
	H	8
	L	8
Contador de programa	**PC**	14

El 8008 no tiene registro de puntero de stack (SP). Tiene una pila interna de 8 posiciones para almacenar las direcciones de retorno en el caso de llamadas a subrutina.

Hay cuatro indicadores (o flags): Carry, Sign, Parity y Zero (C, S, P, Z).

Conjunto de instrucciones del 8008

SALTOS	LLAMADAS	RETORNOS	CONDICIÓN
JMP *Addr*	**CALL** *Addr*	**RET**	Incondicional
JNC *Addr*	**CNC** *Addr*	**RNC**	Si no hay carry
JNZ *Addr*	**CNZ** *Addr*	**RNZ**	Si no es cero
JP *Addr*	**CP** *Addr*	**RP**	Si es positivo
JPO *Addr*	**CPO** *Addr*	**RPO**	Si paridad impar
JC *Addr*	**CC** *Addr*	**RC**	Si hay carry
JZ *Addr*	**CZ** *Addr*	**RZ**	Si es cero
JM *Addr*	**CM** *Addr*	**RM**	Si es negativo
JPE *Addr*	**CPE** *Addr*	**RPE**	Si paridad par

MOVER DATO		ARITMETICA Y LOGICA	
MVI *reg,D8*	*reg* <- *D8*	**ADI** *D8*	A <- A + *D8*
MOV *reg*,A	*reg* <- A	**ADD** *reg*	A <- A + *reg*
MOV *reg*,B	*reg* <- B	**ACI** *D8*	A <- A + *D8* + Cy
MOV *reg*,C	*reg* <- C	**ADC** *reg*	A <- A + *reg* + Cy
MOV *reg*,D	*reg* <- D	**SUI** *D8*	A <- A - *D8*
MOV *reg*,E	*reg* <- E	**SUB** *reg*	A <- A - *reg*
MOV *reg*,H	*reg* <- H	**SBI** *D8*	A <- A - *D8* - Cy
MOV *reg*,L	*reg* <- L	**SBB** *reg*	A <- A - *reg* - Cy
		ANI *D8*	A <- A and *D8*
INCREMENTAR	**DECREMENTAR**	**ANA** *reg*	A <- A and *reg*
INR B: B<-B+1	**DCR** B: B<-B-1	**XRI** *D8*	A <- A xor *D8*
INR C: C<-C+1	**DCR** C: C<-C-1	**XOR** *reg*	A <- A xor *reg*
INR D: D<-D+1	**DCR** D: D<-D-1	**ORI** *D8*	A <- A or *D8*
INR E: E<-E+1	**DCR** E: E<-E-1	**ORA** *reg*	A <- A or *reg*
INR H: H<-H+1	**DCR** H: H<-H-1	**CPI** *D8*	A - *D8*
INR L: L<-L+1	**DCR** L: L<-L-1	**CMP** *reg*	A - *reg*

ROTAR ACUMULADOR	CONTROL	RESTART
RLC	**HLT**	**RST** *n* (0 <= *n* <= 7) Es una llamada a una subrutina cuya dirección absoluta es 8**n*
RRC	**NOP**	
RAL		
RAR		

ENTRADA DE PORT	SALIDA A PORT
IN *n* (0 <= *n* <= 7) A <- Port *n*	OUT *n* (8 <= *n* <=31) Port *n* <- *A*

donde *Addr* es una dirección de 14 bits y *D8* es un dato inmediato de 8 bits. *reg* puede ser cualquiera de los siete registros A, B, C, D, E, H o L o bien puede ser **M**. El último indica direccionamiento indirecto utilizando el par HL como puntero.

El microprocesador 8080

Historia del 8080

Durante el verano de 1971, Federico Faggin fue a Europa para realizar seminarios sobre el MCS-4 y el 8008 y para visitar clientes. Recibió una gran cantidad de críticas (algunas de ellas constructivas) acerca de la arquitectura y el rendimiento de los microprocesadores. Las compañías que estaban más orientadas hacia la computación eran las que le decían las peores críticas.

Cuando regresó a su casa, se le ocurrió una idea de cómo hacer un microprocesador de 8 bits mejor que el 8008, incorporando muchas de las características que esa gente estaba pidiendo, sobre todo, más velocidad y facilidad de implementación en el circuito.

Decidió utilizar el nuevo proceso **NMOS** (que utiliza transistores MOS de canal N) que se utilizaba en las últimas memorias RAM dinámicas de 4 kilobits, además le agregó una mejor estructura de interrupciones, mayor direccionamiento de memoria (16 KB en el 8008 contra 64 KB en el 8080) e instrucciones adicionales (como se puede apreciar en las descripciones de los conjuntos de instrucciones que se encuentran más abajo).

Al principio de 1972 decidió realizar el nuevo chip. Sin embargo Intel decidió esperar a que el mercado respondiera primero con el MCS-4 y luego con el MCS-8 antes de dedicar más dinero al desarrollo de nuevos diseños.

En el verano de 1972, la decisión de Intel fue comenzar con el desarrollo del nuevo microprocesador. Shima (el mismo de antes) comenzó a trabajar en el proyecto en noviembre.

La primera fabricación del 8080 se realizó en diciembre de 1973. Los miembros del grupo que hacían el desarrollo encontraron un pequeño error y el primero de abril de 1974 se pudo lanzar al mercado el microprocesador.

El 8080 realmente creó el verdadero mercado de los microprocesadores. El 4004 y el 8008 lo sugirieron, pero el 8080 lo hizo real. Muchas aplicaciones que no eran posibles de realizar con los microprocesadores previos pudieron hacerse realidad con el 8080. Este chip se usó inmediatamente en cientos de productos diferentes. En el 8080 corría el famoso sistema operativo **CP/M** (siglas de *Control Program for Microcomputers*) de la década del '70 que fue desarrollado por la compañía Digital Research.

Como detalle constructivo el 8080 tenía alrededor de 6000 transistores MOS de canal N (**NMOS**) de 6 micrones, se conectaba al exterior mediante 40 patas (en formato **DIP**) y necesitaba tres tensiones para su funcionamiento (típico de los circuitos integrados de esa época): +12V, +5V y -5V. La frecuencia máxima era de 2 MHz.

La competencia de Intel vino de Motorola. Seis meses después del lanzamiento del 8080, apareció el 6800. Este producto era mejor en varios aspectos que el primero. Sin embargo, la combinación de tiempos (el 8080 salió antes), "marketing" más agresivo, la gran cantidad de herramientas de hardware y software, y el tamaño del chip (el del 8080 era mucho menor que el del 6800 de Motorola) inclinaron la balanza hacia el 8080.

El mayor competidor del 8080 fue el microprocesador Z-80, que fue lanzado en 1976 por la empresa Zilog (fundada por Faggin). Entre las ventajas pueden citarse: mayor cantidad de instrucciones (158

contra 74), frecuencia de reloj más alta, circuito para el apoyo de refresco de memorias RAM dinámicas, compatibilidad de código objeto (los códigos de operación de las instrucciones son iguales) y una sola tensión para su funcionamiento (+5V). El Z-80 fue concebido por Federico Faggin y Masatoshi Shima como una mejora al 8080, comenzando el desarrollo a partir de noviembre de 1974 en la empresa presidida por el primero. Tal fue el éxito que tuvo esta CPU que luego varias empresas comenzaron a producir el chip: SGS-Ates, Mostek, Philips, Toshiba, NEC, Sharp, etc.

Este microprocesador ocupó rápidamente el lugar del anterior y se usó en todo tipo de microcomputadoras (incluyendo muchas de las "home computers" de la primera mitad de la década del '80).

Arquitectura del 8080

Debe notarse la gran semejanza en la arquitectura de los microprocesadores 8008 y 8080.

Conjunto de registros del 8080

Es una ampliación del conjunto del 8008, como puede observarse a continuación:

Clasificación	Registro	Longitud	Pares de registros	Longitud
Acumulador	**A**	8 bits		
Registros de uso general	**B**	8 bits	**BC**	16 bits
	C	8 bits		
	D	8 bits	**DE**	16 bits
	E	8 bits		
	H	8 bits	**HL**	16 bits
	L	8 bits		
Contador de programa	**PC**	16 bits		
Puntero de pila	**SP**	16 bits		
Indicadores	**F**	8 bits		

Hay cinco indicadores (Sign, Zero, Alternate Carry, Parity, Carry) ubicado en un registro de ocho bits llamado **F** (de *Flags*):

Bit	7	6	5	4	3	2	1	0
Flag	S	Z	0	AC	0	P	1	C

Los tres bits no usados siempre toman esos valores.

Conjunto de instrucciones del 8080

Incluye el conjunto de instrucciones del 8008. Además existen las siguientes:

SUMA 16 BITS	
DAD B	HL <- HL+BC
DAD D	HL <- HL+DE
DAD H	HL <- HL+HL
DAD SP	HL <- HL+SP

CARGA 16 BITS	
LXI B,*D16*	BC <- *D16*
LXI D,*D16*	DE <- *D16*
LXI H,*D16*	HL <- *D16*
LXI SP,*D16*	SP <- *D16*
LHLD *Addr*	HL <- (*Addr*)
SHLD *Addr*	(*Addr*) <- HL

INCREMENTAR	
INR M	(HL) <- (HL)+1
INR A	A <- A+1
INX B	BC <- BC+1
INX D	DE <- DE+1
INX H	HL <- HL+1
INX SP	SP <- SP+1

DECREMENTAR	
DCR M	(HL) <- (HL)-1
DCR A	A <- A-1
DCX B	BC <- BC-1
DCX D	DE <- DE-1
DCX H	HL <- HL-1
DCX SP	SP <- SP-1

CARGA 8 BITS	
LDAX B	A <- (BC)
LDAX D	A <- (DE)
STAX B	(BC) <- A
STAX D	(DE) <- A
LDA *Addr*	A <- (*Addr*)
STA *Addr*	(*Addr*) <- A

ESPECIALES	
XCHG	DE <-> HL
DAA	Ajuste decimal acumul.
CMA	A <- 0FFh - A
STC	Cy <- 1
CMC	Cy <- 1 - Cy

OPERACIONES CON LA PILA	
PUSH B	Push BC
PUSH D	Push DE
PUSH H	Push HL
PUSH PSW	Push AF
POP B	Pop BC
POP D	Pop DE
POP H	Pop HL
POP PSW	Pop AF
XTHL	HL <-> (SP)
SPHL	SP <- HL

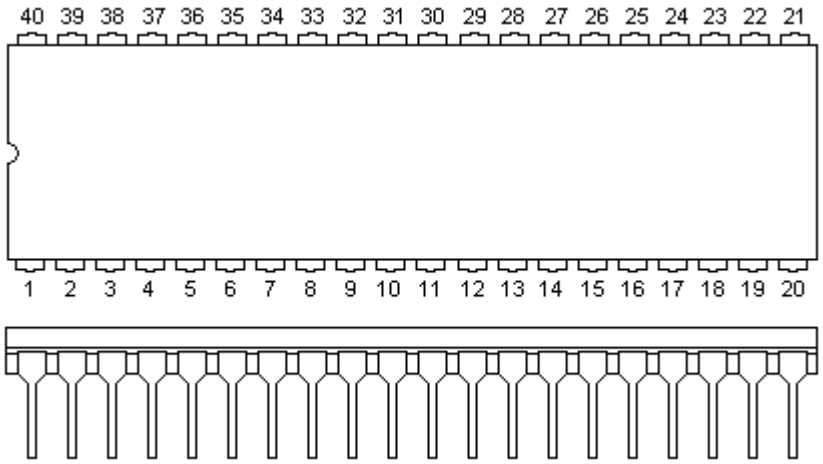
CONTROL	
DI	Deshabilitar interrupciones
EI	Habilitar interrupciones

ENTRADA/SALIDA	
IN *D8*	A <- Port *D8*
OUT *D8*	Port *D8* <- A

donde *Addr* es una dirección de 16 bits.

Si bien todas las instrucciones del 8008 están incluidas en el 8080, un programa grabado en ROM para el primer procesador no correrá para el segundo ya que los códigos de operación de las instrucciones son diferentes, por lo que se deberá volver a ensamblar el código fuente para que pueda funcionar en el 8080.

Terminales (*pinout*) del 8080



Este microprocesador estaba encapsulado en el formato **DIP** (*Dual Inline Package*) de 40 patas (veinte de cada lado). La distancia entre las patas es de 0,1 pulgadas (2,54 milímetros), mientras que la distancia entre patas enfrentadas es de 0,6 pulgadas (15,32 milímetros).

Nótese en el gráfico el semicírculo que identifica la posición de la pata 1. Esto sirve para no insertar el chip al revés en el circuito impreso.

Las funciones de las 40 patas con las que se conecta el 8080 con el exterior son las siguientes:

Pata	Nombre	Descripción

1	**A10**	Bus de direcciones
2	**GND**	Referencia de tierra. Todas las tensiones se miden con respecto a este punto.
3	**D4**	Si SYNC = 0: Bus de datos. Si SYNC = 1: Señal de control que indica salida a periférico.
4	**D5**	Si SYNC = 0: Bus de datos. Si SYNC = 1: Señal que indica si el uP está en ciclo de búsqueda de instrucción.
5	**D6**	Si SYNC = 0: Bus de datos. Si SYNC = 1: Señal de control que indica entrada de periférico.
6	**D7**	Si SYNC = 0: Bus de datos. Si SYNC = 1: Señal de control que indica lectura de memoria.
7	**D3**	Si SYNC = 0: Bus de datos. Si SYNC = 1: Señal que indica que el uP se ha detenido.
8	**D2**	Si SYNC = 0: Bus de datos. Si SYNC = 1: Señal que indica que se realiza una operación con el stack.
9	**D1**	Si SYNC = 0: Bus de datos. Si SYNC = 1: Modo lectura/escritura.
10	**D0**	Si SYNC = 0: Bus de datos. Si SYNC = 1: Señal de reconocimiento de interrupción.
11	**-5V**	Una de las tres patas de alimentación del 8080.
12	**RESET**	Señal de borrado de todos los registros internos del 8080. Para ello, ponerlo a uno durante tres ciclos de reloj como mínimo.
13	**HOLD**	Sirve para poner los buses en alta impedancia para el manejo de DMA (acceso directo a memoria).
14	**INT**	Señal de pedido de interrupción.
15	**CLK2**	Señal de reloj (debe venir del generador de reloj 8224).
16	**INTE**	Señal de aceptación de interrupción.
17	**DBIN**	Indica que el bus de datos está en modo lectura.
18	**/WR**	Indica que el bus de datos está en modo escritura.
19	**SYNC**	Este pin se pone a uno cuando comienza una nueva instrucción.
20	**+5V**	Una de las tres patas de alimentación del 8080.
21	**HLDA**	Reconocimiento de HOLD.
22	**CLK1**	Señal de reloj (debe venir del generador de reloj 8224).
23	**READY**	Sirve para sincronizar memorias o periféricos lentos (detiene al 8080 mientras se lee o escribe el dispositivo).
24	**WAIT**	Cuando vale "1", el 8080 está esperando al periférico lento.
25	**A0**	Bus de direcciones.
26	**A1**	
27	**A2**	

28	**+12V**	Una de las tres patas de alimentación del 8080.
29	**A3**	Bus de direcciones.
30	**A4**	
31	**A5**	
32	**A6**	
33	**A7**	
34	**A8**	
35	**A9**	
36	**A15**	
37	**A12**	
38	**A13**	
39	**A14**	
40	**A11**	

Cuando la pata SYNC está a "1" lógico, las patas D0-D7 pasan a ser señales de control, por lo que no se puede conectar directamente D0-D7 al bus de datos. Se debe intercalar un controlador y amplificador de bus 8228.

De esta manera se puede observar que el 8080 no funciona si no se agregan los circuitos integrados de soporte 8224 y 8228.

El microprocesador 8085

Introducción

El siguiente microprocesador creado por la empresa Intel fue el 8085 en 1977. La alimentación es única: requiere sólo +5V. Esto se debe a la nueva tecnología utilizada para la fabricación llamada **HMOS** (High performance N-channel MOS) que además permite una mayor integración, llegando a la VLSI (Very Large Scale of Integration o muy alta escala de integración) con más de diez mil transistores (el 8085 no es VLSI, pero sí el 8088, como se verá más adelante). Tiene incorporado el generador de pulsos de reloj con lo que sólo hace falta un cristal de cuarzo y un par de capacitores externos (para el 8080 se necesitaba el circuito integrado auxiliar que lleva el código 8224). Además está mejorado en lo que se refiere a las interrupciones. Incluye las 74 instrucciones del 8080 y posee dos adicionales (RIM y SIM) referidas a este sistema de interrupciones y a la entrada y salida serie. El bus de datos está multiplexado con los ocho bits menos significativos del bus de direcciones (utiliza los mismos pines para ambos buses), con lo que permite tener más pines libres para el bus de control del microprocesador (el 8080 necesitaba un integrado especial, el 8228, para generar el bus de control). Intel produjo ROMs, RAMs y chips de soporte que tienen también el bus multiplexado de la misma manera que el microprocesador. Todos estos integrados forman la familia **MCS-85**.

Debido a la gran densidad de integración comparado con el 8080, se utilizó mucho este microprocesador en aplicaciones industriales. Sin embargo, para aplicaciones de computación de uso general, se extendió más el uso del microprocesador Z-80 como se indicó en el apartado referente al 8080.

Interrupciones

El microprocesador 8085 posee un complejo y completo sistema de interrupciones. Esta uP posee cinco terminales destinados al tratamiento de interrupciones.

Recordemos que una interrupción es un artificio hardware/software por el cual es posible detener el programa en curso para que, cuando se produzca un evento predeterminado, después de concluir la instrucción que está ejecutando, efectúe un salto a una determinada subrutina en donde se efectuará el tratamiento de la interrupción; una vez acabado éste, el uP continúa con la instrucción siguiente del programa principal.

Así pues, el 8085 dispone de tres formas diferentes de tratar las interrupciones que le llegan por los citados cinco terminales. Los nombres de estos cinco terminales son:

- **INTR** (*Interrupt Request*): Por esta entrada se introduce una interrupción que es aceptada o no según haya sido previamente indicado por las instrucciones **EI** (Permitir interrupciones) o **DI** (No permitir interrupciones). Cuando una interrupción es permitida y ésta se ha producido, la CPU busca una instrucción **RST** (de un sólo byte), que es presentada por el bus de datos por el periférico que interrumpe. Este byte tiene el formato binario 11 XXX 111. La subrutina se ubicará en la dirección 00 XXX 000.

- **RST 5.5**, **RST 6.5** y **RST 7.5**: Los terminales de **RST 5.5** y **RST 6.5** detectan la interrupción sólo si la señal que se les aplica es un uno lógico o nivel alto de una cierta duración, lo mismo que la entrada anterior INTR; sin embargo, la entrada de interrupción correspondiente al terminal **RST 7.5** se excita por flanco ascendente, es decir, por una transición de cero a uno. Esta transición se memoriza en un biestable en el interior del uP.

Estas interrupciones se pueden habilitar o deshabilitar mediante las instrucciones **EI** y **DI**, como en el caso de INTR; pero además son enmascarables por software mediante la instrucción SIM (Set Interrupt Mask). Es posible leer tanto el estado de la máscara como las interrupciones que se han producido y aún no se atendieron mediante la instrucción RIM.

- **TRAP**: Es una interrupción no enmascarable que es activada cuando el terminal del mismo nombre se lleva a nivel lógico uno. Esta interrupción es la de más alta prioridad, por lo que puede ser usada para tratar los acontecimientos más relevantes, tales como errores, fallos de alimentación, etc.

Nivel de prioridad	Nombre de la interrupción	Valor leído en el bus de datos	Dirección de la subrutina en hexadecimal
Mayor prioridad	**TRAP**	No importa	0024
-	**RST 7.5**		003C
-	**RST 6.5**		0034
-	**RST 5.5**		002C
Menor prioridad	**INTR**	11000111	0000
		11001111	0008
		11010111	0010
		11011111	0018
		11100111	0020
		11101111	0028
		11110111	0030
		11111111	0038

Control de entrada/salida serie

Este microprocesador posee dos terminales denominados **SID** (Serial Input Data) y **SOD** (Serial Output Data). Estos terminales se pueden usar con propósitos generales. Por ejemplo el terminal **SID** se puede conectar a un interruptor y el **SOD** a un LED (a través de una compuerta inversora externa). Para leer el estado del terminal **SID** se ejecuta la instrucción RIM, con lo que se puede leer en el bit 7 del acumulador el estado de dicho terminal.

Para enviar un dato por el terminal **SOD** se ejecuta la instrucción SIM, donde el bit 7 del acumulador debe tener el valor a poner en el terminal, y el bit 6 debe estar a uno.

Conjunto de instrucciones del 8085

Aparte de las 74 instrucciones del 8080, este procesador posee dos instrucciones más.

- **SIM** (*Set interrupt mask*): Sirve para poner la máscara de interrupción de **RST 5.5**, **RST 6.5** y **RST 7.5** y para enviar un dato por la puerta serie (terminal **SOD**).

Estos datos deben estar cargados en el acumulador y son:

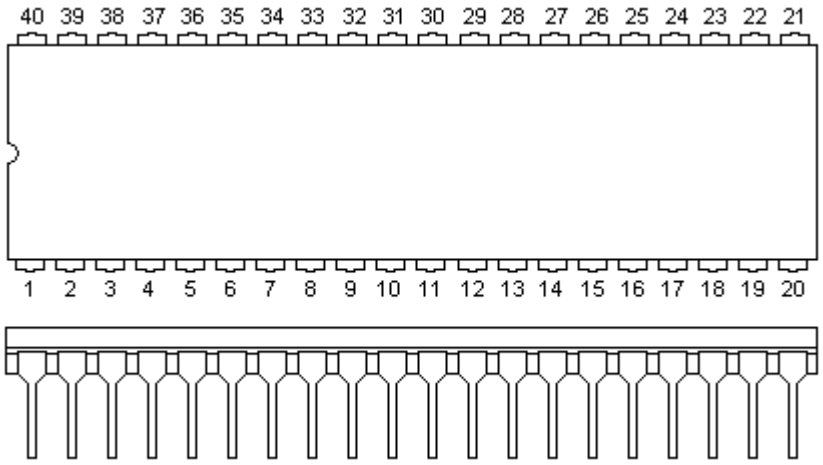
- Bit 7: Valor a enviar al terminal **SOD**
- Bit 6: Permiso para cambiar el estado del terminal **SOD**. Sólo se puede cambiar si vale 1.
- Bit 5: No usado.
- Bit 4: **R 7.5** (*Reset 7.5*): Bit para poner a cero el biestable de la interrupción **RST 7.5**.
- Bit 3: **MSE** (*Mask Select Enable*): Cuando vale 1, se puede cambiar la máscara de interrupción.
- Bit 2: **M 7.5** (*Mask 7.5*): Se habilita la interrupción **RST 7.5** si este bit vale 1 y se ejecutó previamente la instrucción **EI**.
- Bit 1: **M 6.5** (*Mask 6.5*): Se habilita la interrupción **RST 6.5** si este bit vale 1 y se ejecutó previamente la instrucción **EI**.
- Bit 0: **M 5.5** (*Mask 5.5*): Se habilita la interrupción **RST 5.5** si este bit vale 1 y se ejecutó previamente la instrucción **EI**.

- **RIM** (*Read interrupt mask*): Sirve para leer la máscara de interrupción general, y de **RST 5.5**, **RST 6.5**, **RST 7.5**, las interrupciones pendientes y para leer el dato de la puerta serie (terminal **SID**).

  Luego de la ejecución de esta instrucción, el acumulador tiene lo siguiente:

  - Bit 7: Valor leído del terminal **SID**
  - Bit 6: **I 7.5** (*Interrupt Pending 7.5*): Indica que todavía no se ejecutó la interrupción **RST 7.5**.
  - Bit 5: **I 6.5** (*Interrupt Pending 6.5*): Indica que todavía no se ejecutó la interrupción **RST 6.5**.
  - Bit 4: **I 5.5** (*Interrupt Pending 5.5*): Indica que todavía no se ejecutó la interrupción **RST 5.5**.
  - Bit 3: **IE** (*Interrupt Enable*): Cuando vale 1 la interrupción INTR está habilitada.
  - Bit 2: **M 7.5** (*Mask 7.5*): Si este bit y **IE** valen 1, la interrupción **RST 7.5** está habilitada.
  - Bit 1: **M 6.5** (*Mask 6.5*): Si este bit y **IE** valen 1, la interrupción **RST 6.5** está habilitada.
  - Bit 0: **M 5.5** (*Mask 5.5*): Si este bit y **IE** valen 1, la interrupción **RST 5.5** está habilitada.

Terminales (*pinout*) del 8085



Este microprocesador estaba encapsulado en el formato **DIP** (*Dual Inline Package*) de 40 patas (veinte de cada lado). La distancia entre las patas es de 0,1 pulgadas (2,54 milímetros), mientras que la distancia

entre patas enfrentadas es de 0,6 pulgadas (15,32 milímetros).

Nótese en el gráfico el semicírculo que identifica la posición de la pata 1. Esto sirve para no insertar el chip al revés en el circuito impreso.

Las funciones de las 40 patas con las que se conecta el 8085 con el exterior son las siguientes:

Pata	Nombre	Descripción
1	**X1**	Entre estas dos patas se ubica el cristal
2	**X2**	
3	**RESET OUT**	Para inicializar periféricos
4	**SOD**	Salida serie
5	**SID**	Entrada serie
6	**TRAP**	Entrada de interrupción no enmascarable
7	**RST 7.5**	Entrada de interrupción (máxima prioridad)
8	**RST 6.5**	Entrada de interrupción
9	**RST 5.5**	Entrada de interrupción
10	**INTR**	Entrada de interrupción (mínima prioridad)
11	**/INTA**	Reconocimiento de interrupción
12	**AD0**	Bus de direcciones y datos multiplexado
13	**AD1**	Bus de direcciones y datos multiplexado
14	**AD2**	Bus de direcciones y datos multiplexado
15	**AD3**	Bus de direcciones y datos multiplexado
16	**AD4**	Bus de direcciones y datos multiplexado
17	**AD5**	Bus de direcciones y datos multiplexado
18	**AD6**	Bus de direcciones y datos multiplexado
19	**AD7**	Bus de direcciones y datos multiplexado
20	**GND**	Referencia de tierra. Todas las tensiones se miden con respecto a este punto.
21	**A8**	Bus de direcciones
22	**A9**	Bus de direcciones
23	**A10**	Bus de direcciones
24	**A11**	Bus de direcciones
25	**A12**	Bus de direcciones
26	**A13**	Bus de direcciones
27	**A14**	Bus de direcciones
28	**A15**	Bus de direcciones
29	**S0**	Bit de estado del 8085

30	**ALE**	Cuando está uno indica que salen direcciones por las patas **AD*n***, en caso contrario, entran o salen datos
31	**/WR**	Cuando vale cero hay una escritura
32	**/RD**	Cuando vale cero hay una lectura
33	**S1**	Bit de estado del 8085
34	**IO/M**	Si vale 1: operaciones con ports, si vale 0: operaciones con la memoria
35	**READY**	Sirve para sincronizar memorias o periféricos lentos
36	**/RESET IN**	Cuando está a cero inicializa el 8085
37	**CLK OUT**	Salida del reloj para los periféricos
38	**HLDA**	Reconocimiento de HOLD
39	**HOLD**	Sirve para poner los buses en alta impedancia para el manejo de DMA (acceso directo a memoria)
40	**VCC**	tensión de alimentación: +5Vdc

Los microprocesadores 8086 y 8088

Historia del 8086/8088

En junio de 1978 Intel lanzó al mercado el primer microprocesador de 16 bits: el 8086. En junio de 1979 apareció el 8088 (internamente igual que el 8086 pero con bus de datos de 8 bits) y en 1980 los coprocesadores 8087 (matemático) y 8089 (de entrada y salida). El primer fabricante que desarrolló software y hardware para estos chips fue la propia Intel. Reconociendo la necesidad de dar soporte a estos circuitos integrados, la empresa invirtió gran cantidad de dinero en un gran y moderno edificio en Santa Clara, California, dedicado al diseño, fabricación y venta de sus sistemas de desarrollo que, como se explicó anteriormente, son computadoras autosuficientes con el hardware y software necesario para desarrollar software de microprocesadores.

Los sistemas de desarrollo son factores clave para asegurar las ventas de una empresa fabricantes de chips. La inmensa mayoría de ventas son a otras empresas, las cuales usan estos chips en aparatos electrónicos, diseñados, fabricados y comercializados por ellas mismas. A estas empresas se las llama "fabricantes de equipo original", o en inglés, OEM (Original Equipment Manufacturer). El disminuir el tiempo de desarrollo de hardware y software para las OEM es esencial, ya que el mercado de estos productos es muy competitivo. Necesitan soporte pues los meses que les puede llevar el desarrollo de las herramientas apropiadas les puede significar pérdidas por millones de dólares. Además quieren ser los primeros fabricantes en el mercado, con lo cual pueden asegurarse las ventas en dos áreas importantes: a corto plazo, ya que al principio la demanda es mucho mayor que la oferta, y a largo plazo, ya que el primer producto marca a menudo los estándares.

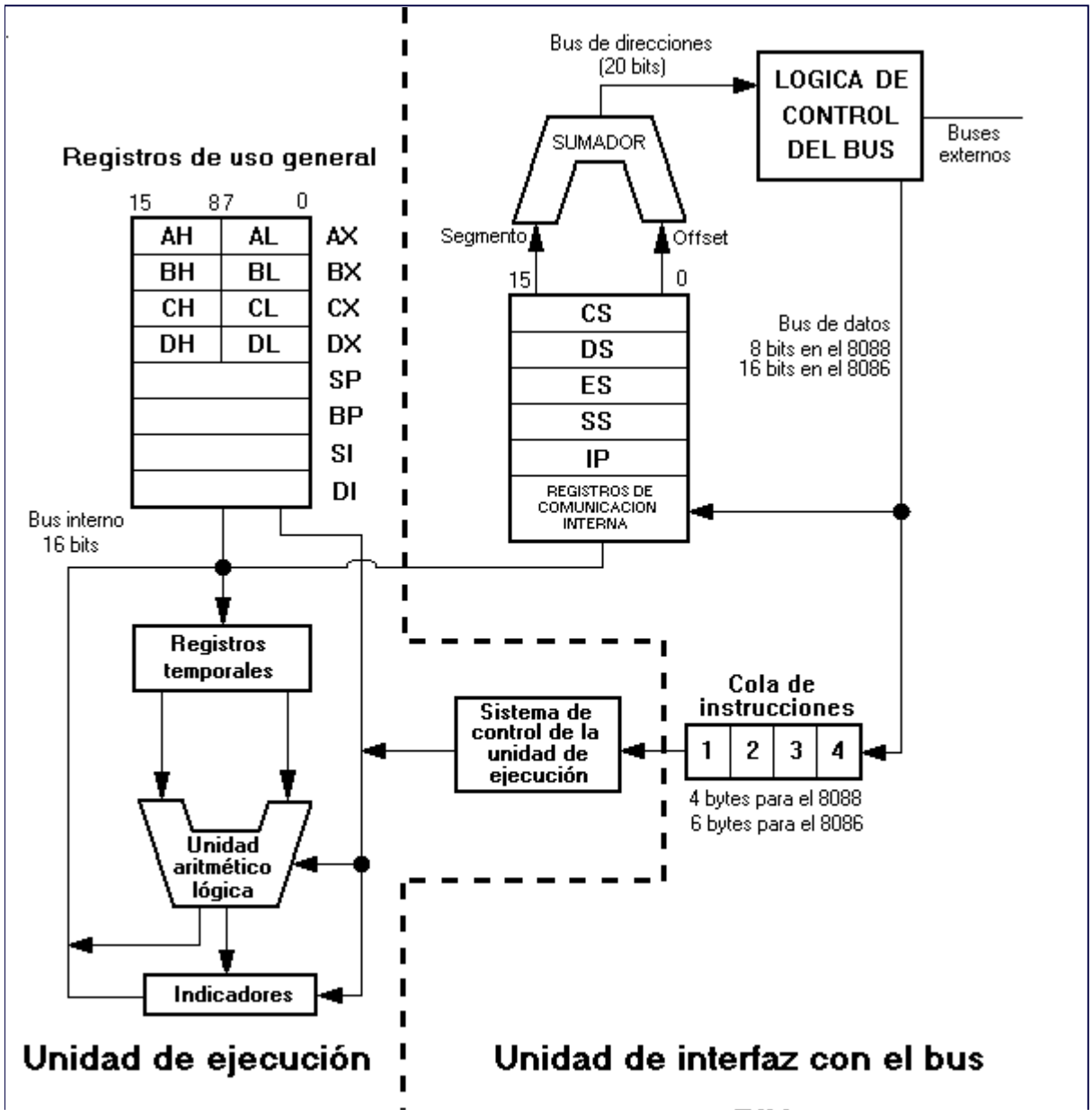
De esta manera la empresa Intel había desarrollado una serie completa de software que se ejecutaba en una microcomputadora basada en el 8085 llamada "Intellec Microcomputer Development System". Los programas incluían ensambladores cruzados (éstos son programas que se ejecutan en un microprocesador y generan código de máquina que se ejecuta en otro), compiladores de PL/M, Fortran y Pascal y varios programas de ayuda. Además había un programa traductor llamado CON V86 que convertía código fuente 8080/8085 a código fuente 8086/8088. Si se observan de cerca ambos conjuntos de instrucciones, queda claro que la transformación es sencilla si los registros se traducen así: A -> AL, B -> CH, C -> CL, D -> DH, E -> DL, H -> BH y L -> BL. Puede parecer complicado traducir LDAX B (por ejemplo) ya que el 8088 no puede utilizar el registro CX para direccionamiento indirecto, sin embargo, se puede hacer con la siguiente secuencia: MOV SI, CX; MOV AL, [SI]. Esto aprovecha el hecho que no se utiliza el registro SI. Por supuesto el programa resultante es más largo (en cantidad de bytes) y a veces más lento de correr que en su antecesor 8085. Este programa de conversión sólo servía para no tener que volver a escribir los programas en una primera etapa. Luego debería reescribirse el código fuente en assembler para poder obtener las ventajas de velocidad ofrecidas por el 8088. Luego debía correr el programa en la iSBC 86/12 Single Board Computer basado en el 8086. Debido al engorro que resultaba tener dos plaquetas diferentes, la empresa Godbout Electronics (también de California) desarrolló una placa donde estaban el 8085 y el 8088, donde se utilizaba un ensamblador cruzado provisto por la compañía Microsoft. Bajo control de software, podían conmutarse los microprocesadores. El sistema operativo utilizado era el CP/M (de Digital Research).

El desarrollo más notable para la familia 8086/8088 fue la elección de la CPU 8088 por parte de IBM (International Business Machines) cuando en 1981 entró en el campo de las computadoras personales. Esta computadora se desarrolló bajo un proyecto con el nombre "Acorn" (Proyecto "Bellota") pero se vendió bajo un nombre menos imaginativo, pero más correcto: "Computadora Personal IBM", con un precio inicial entre 1260 dólares y 3830 dólares según la configuración (con 48KB de memoria RAM y

una unidad de discos flexibles con capacidad de 160KB costaba 2235 dólares). Esta computadora entró en competencia directa con las ofrecidas por Apple (basado en el 6502) y por Radio Shack (basado en el Z-80).

Arquitectura de los procesadores 8088 y 8086:

El 8086 es un microprocesador de 16 bits, tanto en lo que se refiere a su estructura como en sus conexiones externas, mientras que el 8088 es un procesador de 8 bits que internamente es casi idéntico al 8086. La única diferencia entre ambos es el tamaño del bus de datos externo. Intel trata esta igualdad interna y desigualdad externa dividiendo cada procesador 8086 y 8088 en dos sub-procesadores. O sea, cada uno consta de una unidad de ejecución (EU: Execution Unit) y una unidad interfaz del bus (BIU: Bus Interface Unit). La unidad de ejecución es la encargada de realizar todas las operaciones mientras que la unidad de interfaz del bus es la encargada de acceder a datos e instrucciones del mundo exterior. Las unidades de ejecución son idénticas en ambos microprocesadores, pero las unidades de interfaz del bus son diferentes en varias cuestiones, como se desprende del siguiente diagrama en bloques:



La ventaja de esta división fue el ahorro de esfuerzo necesario para producir el 8088. Sólo una mitad del 8086 (el BIU) tuvo que rediseñarse para producir el 8088.

La explicación del diagrama en bloques es la siguiente:

Registros de uso general del 8086/8088:

Tienen 16 bits cada uno y son ocho:

1. **AX** = Registro acumulador, dividido en **AH** y **AL** (8 bits cada uno).
   Usándolo se produce (en general) una instrucción que ocupa un byte menos que si se utilizaran otros registros de uso general. Su parte más baja, AL, también tiene esta propiedad. El último registro mencionado es el equivalente al acumulador de los procesadores anteriores (8080 y 8085). Además hay instrucciones como DAA; DAS; AAA; AAS; AAM; AAD; LAHF; SAHF; CBW; IN y OUT que trabajan con AX o con uno de sus dos bytes (AH o AL). También se utiliza este registro (junto con DX a veces) en multiplicaciones y divisiones.
2. **BX** = Registro base, dividido en **BH** y **BL**.
   Es el registro base de propósito similar (se usa para direccionamiento indirecto) y es una versión más potente del par de registros HL de los procesadores anteriores.
3. **CX** = Registro contador, dividido en **CH** y **CL**.
   Se utiliza como contador en bucles (instrucción LOOP), en operaciones con cadenas (usando el prefijo REP) y en desplazamientos y rotaciones (usando el registro CL en los dos últimos casos).
4. **DX** = Registro de datos, dividido en **DH** y **DL**.
   Se utiliza junto con el registro AX en multiplicaciones y divisiones, en la instrucción CWD y en IN y OUT para direccionamiento indirecto de puertos (el registro DX indica el número de puerto de entrada/salida).
5. **SP** = Puntero de pila (no se puede subdividir).
   Aunque es un registro de uso general, debe utilizarse sólo como puntero de pila, la cual sirve para almacenar las direcciones de retorno de subrutinas y los datos temporarios (mediante las instrucciones PUSH y POP). Al introducir (push) un valor en la pila a este registro se le resta dos, mientras que al extraer (pop) un valor de la pila este a registro se le suma dos.
6. **BP** = Puntero base (no se puede subdividir).
   Generalmente se utiliza para realizar direccionamiento indirecto dentro de la pila.
7. **SI** = Puntero índice (no se puede subdividir).
   Sirve como puntero fuente para las operaciones con cadenas. También sirve para realizar direccionamiento indirecto.
8. **DI** = Puntero destino (no se puede subdividir).
   Sirve como puntero destino para las operaciones con cadenas. También sirve para realizar direccionamiento indirecto.

Cualquiera de estos registros puede utilizarse como fuente o destino en operaciones aritméticas y lógicas, lo que no se puede hacer con ninguno de los seis registros que se verán más adelante.

Además de lo anterior, cada registro tiene usos especiales:

Unidad aritmética y lógica

Es la encargada de realizar las operaciones aritméticas (suma, suma con "arrastre", resta, resta con "préstamo" y comparaciones) y lógicas (AND, OR, XOR y TEST). Las operaciones pueden ser de 16 bits o de 8 bits.

Indicadores (*flags*)

Hay nueve indicadores de un bit en este registro de 16 bits. Los cuatro bits más significativos están indefinidos, mientras que hay tres bits con valores determinados: los bits 5 y 3 siempre valen cero y el bit 1 siempre vale uno (esto también ocurría en los procesadores anteriores).

Registro de indicadores (16 bits)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Flag	--	--	--	--	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

CF (Carry Flag, bit 0): Si vale 1, indica que hubo "arrastre" (en caso de suma) hacia, o "préstamo" (en caso de resta) desde el bit de orden más significativo del resultado. Este indicador es usado por instrucciones que suman o restan números que ocupan varios bytes. Las instrucciones de rotación pueden aislar un bit de la memoria o de un registro poniéndolo en el CF.

PF (Parity Flag, bit 2): Si vale uno, el resultado tiene paridad par, es decir, un número par de bits a 1. Este indicador se puede utilizar para detectar errores en transmisiones.

AF (Auxiliary carry Flag, bit 4): Si vale 1, indica que hubo "arrastre" o "préstamo" del nibble (cuatro bits) menos significativo al nibble más significativo. Este indicador se usa con las instrucciones de ajuste decimal.

ZF (Zero Flag, bit 6): Si este indicador vale 1, el resultado de la operación es cero.

SF (Sign Flag, bit 7): Refleja el bit más significativo del resultado. Como los números negativos se representan en la notación de complemento a dos, este bit representa el signo: 0 si es positivo, 1 si es negativo.

TF (Trap Flag, bit 8): Si vale 1, el procesador está en modo paso a paso. En este modo, la CPU automáticamente genera una interrupción interna después de cada instrucción, permitiendo inspeccionar los resultados del programa a medida que se ejecuta instrucción por instrucción.

IF (Interrupt Flag, bit 9): Si vale 1, la CPU reconoce pedidos de interrupción externas enmascarables (por el pin INTR). Si vale 0, no se reconocen tales interrupciones. Las interrupciones no enmascarables y las internas siempre se reconocen independientemente del valor de IF.

DF (Direction Flag, bit 10): Si vale 1, las instrucciones con cadenas sufrirán "auto-decremento", esto es, se procesarán las cadenas desde las direcciones más altas de memoria hacia las más bajas. Si vale 0, habrá "auto-incremento", lo que quiere decir que las cadenas se procesarán de "izquierda a derecha".

OF (Overflow flag, bit 11): Si vale 1, hubo un desborde en una operación aritmética con signo, esto es, un dígito significativo se perdió debido a que tamaño del resultado es mayor que el tamaño del destino.

Sistema de control de la unidad de ejecución

Es el encargado de decodificar las instrucciones que le envía la cola y enviarle las órdenes a la unidad aritmética y lógica según una tabla que tiene almacenada en ROM llamada CROM (Control Read Only Memory).

Cola de instrucciones

Almacena las instrucciones para ser ejecutadas. La cola se carga cuando el bus está desocupado, de esta manera se logra una mayor eficiencia del mismo. La cola del 8086 tiene 6 bytes y se carga de a dos bytes por vez (debido al tamaño del bus de datos), mientras que el del 8088 tiene cuatro bytes. Esta estructura tiene rendimiento óptimo cuando no se realizan saltos, ya que en este caso habría que vaciar la cola (porque no se van a ejecutar las instrucciones que van después del salto) y volverla a cargar con instrucciones que se encuentran a partir de la dirección a donde se salta. Debido a esto las instrucciones de salto son (después de multiplicaciones y divisiones) las más lentas de este microprocesador.

Registros de la unidad de interfaz con el bus:

El programador puede acceder a cinco registros de 16 bits cada uno, siendo cuatro de ellos registros de segmento y el restante el puntero de instrucción (IP).

Los registros de segmento se llaman:

CS: Registro de segmento de código.
DS: Registro de segmento de datos.
ES: Registro de segmento extra.
SS: Registro de segmento de pila.

La utilización de estos registros se explica más adelante, en la sección que trata de direccionamiento a memoria.

Lógica de control del bus:

El cometido de este bloque es poder unir los bloques anteriormente mencionados con el mundo exterior, es decir, la memoria y los periféricos.

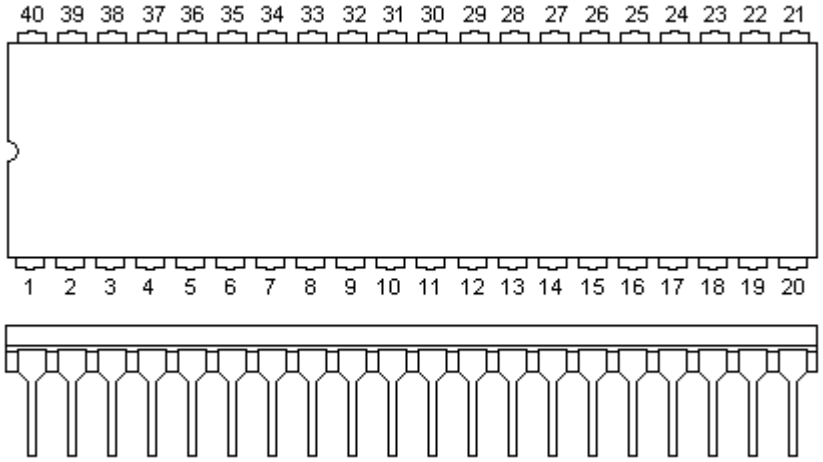
El 8088 tiene un bus de datos externo reducido de 8 bits. La razón para ello era prever la continuidad entre el 8086 y los antiguos procesadores de 8 bits, como el 8080 y el 8085. Teniendo el mismo tamaño del bus (así como similares requerimientos de control y tiempo), el 8088, que es internamente un procesador de 16 bits, puede reemplazar a los microprocesadores ya nombrados en un sistema ya existente.

El 8088 tiene muchas señales en común con el 8085, particularmente las asociadas con la forma en que los datos y las direcciones están multiplexadas, aunque el 8088 no produce sus propias señales de reloj como lo hace el 8085 (necesita un chip de soporte llamado 8284, que es diferente del 8224 que necesitaba el microprocesador 8080). El 8088 y el 8085 siguen el mismo esquema de compartir los terminales correspondientes a los 8 bits más bajos del bus de direcciones con los 8 bits del bus de datos, de manera que se ahorran 8 terminales para otras funciones del microprocesador. El 8086 comparte los 16 bits del bus de datos con los 16 más bajos del bus de direcciones.

El 8085 y el 8088 pueden, de hecho, dirigir directamente los mismos chips controladores de periféricos. Las investigaciones de hardware para sistemas basados en el 8080 o el 8085 son, en su mayoría, aplicables al 8088.

En todo lo recién explicado se basó el éxito del 8088.

Terminales (*pinout*) del 8088



Este microprocesador está encapsulado en el formato **DIP** (*Dual Inline Package*) de 40 patas (veinte de cada lado). La distancia entre las patas es de 0,1 pulgadas (2,54 milímetros), mientras que la distancia entre patas enfrentadas es de 0,6 pulgadas (15,32 milímetros).

Nótese en el gráfico el semicírculo que identifica la posición de la pata 1. Esto sirve para no insertar el chip al revés en el circuito impreso.

El 8086/8088 puede conectarse al circuito de dos formas distintas: el modo máximo y el modo mínimo. El modo queda determinado al poner un determinado terminal (llamado MN/MX) a tierra o a la tensión de alimentación. El 8086/8088 debe estar en modo máximo si se desea trabajar en colaboración con el Procesador de Datos Numérico 8087 y/o el Procesador de Entrada/Salida 8089 (de aquí se desprende que en la IBM PC el 8088 está en modo máximo). En este modo el 8086/8088 depende de otros chips adicionales como el Controlador de Bus 8288 para generar el conjunto completo de señales del bus de control. El modo mínimo permite al 8086/8088 trabajar de una forma más autónoma (para circuitos más sencillos) en una manera casi idéntica al microprocesador 8085.

Los 40 pines del 8088 en modo mínimo tienen las siguientes funciones:

1. **GND** (Masa)
2. **A14** (Bus de direcciones)
3. **A13** (Bus de direcciones)
4. **A12** (Bus de direcciones)
5. **A11** (Bus de direcciones)
6. **A10** (Bus de direcciones)
7. **A9** (Bus de direcciones)
8. **A8** (Bus de direcciones)
9. **AD7** (Bus de direcciones y datos)

10. **AD6** (Bus de direcciones y datos)
11. **AD5** (Bus de direcciones y datos)
12. **AD4** (Bus de direcciones y datos)
13. **AD3** (Bus de direcciones y datos)
14. **AD2** (Bus de direcciones y datos)
15. **AD1** (Bus de direcciones y datos)
16. **AD0** (Bus de direcciones y datos)
17. **NMI** (Entrada de interrupción no enmascarable)
18. **INTR** (Entrada de interrupción enmascarable)
19. **CLK** (Entrada de reloj generada por el 8284)
20. **GND** (Masa)
21. **RESET** (Para inicializar el 8088)
22. **READY** (Para sincronizar periféricos y memorias lentas)
23. **/TEST**
24. **/INTA** (El 8088 indica que reconoció la interrupción)
25. **ALE** (Cuando está uno indica que salen direcciones por AD, en caso contrario, es el bus de datos)
26. **/DEN** (Data enable: cuando vale cero debe habilitar los transceptores 8286 y 8287 (se conecta al pin de "output enable"), esto sirve para que no se mezclen los datos y las direcciones).
27. **DT/R** (Data transmit/receive: se conecta al pin de dirección de los chips recién indicados).
28. **IO/M** (Si vale 1: operaciones con ports, si vale 0: operaciones con la memoria)
29. **/WR** (Cuando vale cero hay una escritura)
30. **HLDA** (Hold Acknowledge: el 8088 reconoce el HOLD)
31. **HOLD** (Indica que otro integrado quiere adueñarse del control de los buses, generalmente se usa para DMA o acceso directo a memoria).
32. **/RD** (Cuando vale cero hay una lectura)
33. **MN/MX** (Cuando esta entrada está en estado alto, el 8088 está en modo mínimo, en caso contrario está en modo máximo)
34. **/SSO** (Junto con IO/M y DT/R esta salida sirve para determinar estados del 8088)
35. **A19/S6** (Bus de direcciones/bit de estado)
36. **A18/S5** (Bus de direcciones/bit de estado)
37. **A17/S4** (Bus de direcciones/bit de estado)
38. **A16/S3** (Bus de direcciones/bit de estado)
39. **A15** (Bus de direcciones)
40. **Vcc** (+5V)

En modo máximo (cuando se aplica +5V al pin 33) hay algunos pines que cambian de significado:

24.- **QS1**: Estado de la cola de instrucciones (bit 1).
25.- **QS0**: Estado de la cola de instrucciones (bit 0).
26.- **S0**: Bit de estado 0.
27.- **S1**: Bit de estado 1.
28.- **S2**: Bit de estado 2.
29.- **/LOCK**: Cuando vale cero indica a otros controladores del bus (otros microprocesadores o un dispositivo de DMA) que no deben ganar el control del bus. Se activa poniéndose a cero cuando una instrucción tiene el prefijo LOCK.
30.- **RQ/GT1**: Es bidireccional y tiene la misma función que HOLD/HLDA en modo mínimo.
31.- **RQ/GT0**: Como RQ/GT1 pero tiene mayor prioridad.
34.- Esta salida siempre está a uno.

Por ser este microprocesador mucho más complejo que el 8085, tiene más bits de estado que el recién mencionado. A título informativo se detallan los bits de estado:

S2	IO/M	DT/R	/SSO	Significado
		S1	S0	
1	0	0	0	Acceso a código (instrucciones)
1	0	0	1	Lectura de memoria
1	0	1	0	Escritura a memoria
1	0	1	1	Bus pasivo (no hace nada)
0	1	0	0	Reconocimiento de interrupción
0	1	0	1	Lectura de puerto de entrada/salida
0	1	1	0	Escritura a puerto de E/S
0	1	1	1	Estado de parada (Halt)

QS1	QS0	Significado
0	0	No hay operación
0	1	Primer byte del código de operación
1	0	Se vacía la cola de instrucciones
1	1	Siguiente byte de la instrucción

Modos de direccionamiento del 8086/8088:

Estos procesadores tienen 27 modos de direccionamiento (una cantidad bastante más grande que los microprocesadores anteriores) o reglas para localizar un operando de una instrucción. Tres de ellos son comunes a microprocesadores anteriores: **direccionamiento inmediato** (el operando es un número que se encuentra en la misma instrucción), **direccionamiento a registro** (el operando es un registro del microprocesador) y **direccionamiento inherente** (el operando está implícito en la instrucción, por ejemplo, en la multiplicación uno de los operandos siempre es el acumulador). El resto de los modos sirve para localizar un operando en memoria. Para facilitar la explicación de estos modos, se pueden resumir de la siguiente manera:

Deben sumarse cuatro cantidades: 1) **dirección de segmento**, 2) **dirección base**, 3) una **cantidad índice** y 4) un **desplazamiento**.

La dirección de segmento se almacena en el *registro de segmento* (DS, ES, SS o CS). En la próxima sección se indica la forma en que se hace esto. Por ahora basta con saber que el contenido del registro de segmento se multiplica por 16 antes de utilizarse para obtener la dirección real. El registro de segmentación siempre se usa para referenciar a memoria.

La base se almacena en el *registro base* (BX o BP). El índice se almacena en el *registro índice* (SI o DI). Cualquiera de estas dos cantidades, la suma de las dos o ninguna, pueden utilizarse para calcular la dirección real, pero no pueden sumarse dos bases o dos índices. Los registros restantes (AX, CX, DX y

SP) no pueden utilizarse para direccionamiento indirecto. El programador puede utilizar tanto la base como el índice para gestionar ciertas cosas, tales como matrices de dos dimensiones, o estructuras internas a otras estructuras, esquemas que se utilizan en las prácticas comunes de programación. La base y el índice son variables o dinámicas, ya que están almacenadas en registros de la CPU. Es decir, pueden modificarse fácilmente mientras se ejecuta un programa.

Además del segmento, base e índice, se usa un desplazamiento de 16 bits, 8 bits o 0 bits (sin desplazamiento). Ésta es una cantidad estática que se fija al tiempo de ensamblado (paso de código fuente a código de máquina) y no puede cambiarse durante la ejecución del programa (a menos que el programa se escriba sobre sí mismo, lo que constituye una práctica no aconsejada).

Todo esto genera los 24 modos de direccionamiento a memoria que se ven a continuación:

- **Registro indirecto**: 1) [BX], 2) [DI]. 3) [SI].
- **Basado**: 4) desp8[BX], 5) desp8[BP], 6) desp16[BX], 7) desp16[BP].
- **Indexado**: 8) desp8[SI], 9) desp8[DI], 10) desp16[SI], 11) desp16[DI].
- **Basado-indexado**: 12) [BX+SI], 13) [BX+DI], 14) [BP+SI], 15) [BX+DI].
- **Basado-indexado con desplazamiento**: 16) desp8[BX+SI], 17) desp8[BX+DI], 18) desp8[BP+SI], 19) desp8[BX+DI], 20) desp16[BX+SI], 21) desp16[BX+DI], 22) desp16[BP+SI], 23) desp16[BX+DI].
- **Directo**: 24) [desp16].

Aquí desp8 indica desplazamiento de 8 bits y desp16 indica desplazamiento de 16 bits. Otras combinaciones no están implementadas en la CPU y generarán error al querer ensamblar, por ejemplo, ADD CL,[DX+SI].

El ensamblador genera el tipo de desplazamiento más apropiado (0, 8 ó 16 bits) dependiendo del valor que tenga la constante: si vale cero se utiliza el primer caso, si vale entre -128 y 127 se utiliza el segundo, y en otro caso se utiliza el tercero. Nótese que [BP] sin desplazamiento no existe. Al ensamblar una instrucción como, por ejemplo, *MOV AL,[BP]*, se generará un desplazamiento de 8 bits con valor cero. Esta instrucción ocupa tres bytes, mientras que *MOV AL,[SI]* ocupa dos, porque no necesita el desplazamiento.

Estos modos de direccionamiento producen algunos inconvenientes en el 8086/8088. La CPU gasta tiempo calculando una dirección compuesta de varias cantidades. Principalmente esto se debe al hecho de que el cálculo de direcciones está programado en microcódigo (dentro de la CROM del sistema de control de la unidad de ejecución). En las siguientes versiones (a partir del 80186/80188) estos cálculos están cableados en la máquina y, por lo tanto, cuesta mucho menos tiempo el realizarlos.

Veamos un ejemplo: *MOV AL, ES:[BX+SI+6]*. En este caso el operando de la izquierda tiene direccionamiento a registro mientras que el de la derecha indica una posición de memoria. Poniendo valores numéricos, supongamos que los valores actuales de los registros sean: ES = 3200h, BX = 200h, SI = 38h. Como se apuntó más arriba la dirección real de memoria será:

ES * 10h + BX + SI + 6 = 3200h * 10h + 200h + 38h + 6 = 3223Eh

Estructura de memoria de segmentación: Como se ha mencionado anteriormente, el 8086/8088 usa un esquema ingenioso llamado segmentación, para acceder correctamente a un megabyte completo de memoria, con referencias de direcciones de sólo 16 bits.

Veamos cómo funciona. Cualquier dirección tiene dos partes, cada una de las cuales es una cantidad de 16 bits. Una parte es la dirección de segmento y la otra es el offset. A su vez el offset se compone de varias partes: un desplazamiento (un número fijo), una base (almacenada en el registro base) y un índice (almacenado en el registro índice). La dirección de segmento se almacena en uno de los cuatro registros de segmento (CS, DS, ES, SS). El procesador usa estas dos cantidades de 16 bits para calcular la dirección real de 20 bits, según la siguiente fórmula:

Dirección real = 16 * (dirección del segmento) + offset

Tal como veíamos antes, dado que 16 en decimal es 10 en hexadecimal, multiplicar por ese valor es lo mismo que correr el número hexadecimal a la izquierda una posición.

Hay dos registros de segmento que tienen usos especiales: el microprocesador utiliza el registro CS (con el offset almacenado en el puntero de instrucción IP) cada vez que se debe acceder a un byte de instrucción de programa, mientras que las instrucciones que utilizan la pila (llamados a procedimientos, retornos, interrupciones y las instrucciones PUSH y POP) siempre utilizan el registro de segmento SS (con el offset almacenado en el registro puntero de pila SP). De ahí los nombres que toman: CS es el segmento de código mientras que SS es el registro segmento de pila.

Para acceder a datos en la memoria se puede utilizar cualquiera de los cuatro registros de segmento, pero uno de ellos provoca que la instrucción ocupe un byte menos de memoria: es el llamado segmento por defecto, por lo que en lo posible hay que tratar de usar dicho segmento para direccionar datos. Este segmento es el DS (registro de segmento de datos) para todos los casos excepto cuando se utiliza el registro base BP. En este caso el segmento por defecto es SS.

Si se utiliza otro registro, el ensamblador genera un byte de prefijo correspondiente al segmento antes de la instrucción: CS -> 2Eh, DS -> 3Eh, ES -> 26h y SS -> 36h. El uso de estos diferentes segmentos significa que hay áreas de trabajo separadas para el programa, pila y los datos. Cada área tiene un tamaño máximo de 64 KBytes. Dado que hay cuatro registros de segmento, uno de programa (CS), uno de pila (SS) y dos de datos (segmento de datos DS y segmento extra ES) el área de trabajo puede llegar a 4 * 64 KB = 256 KB en un momento dado suponiendo que las áreas no se superponen.

Si el programa y los datos ocupan menos de 64 KB, lo que se hace es fijar los registros de segmento al principio del programa y luego se utilizan diferentes offsets para acceder a distintas posiciones de memoria. En caso contrario necesariamente deberán cambiarse los registros de segmento en la parte del programa que lo requiera. Los registros de segmento DS, ES y SS se cargan mediante las instrucciones MOV y POP, mientras que CS se carga mediante transferencias de control (saltos, llamadas, retornos, interrupciones) intersegmento.

Estructura de interrupciones del 8086/8088

Hay tres clases de interrupción: por hardware, por software e internas (a las dos últimas también se las llama "excepciones").

Veremos primeramente el caso de interrupciones por hardware: Como se mencionó anteriormente, el 8086/8088 tiene dos entradas de petición de interrupción: NMI e INTR y una de reconocimiento (INTA). La gran mayoría de las fuentes de interrupción se conectan al pin INTR, ya que esto permite enmascarar las interrupciones (el NMI no). Para facilitar esta conexión, se utiliza el circuito integrado controlador de interrupciones, que tiene el código 8259A. Este chip tiene, entre otras cosas, ocho patas para sendas

fuentes de interrupción (IRQ0 - IRQ7), ocho para el bus de datos (D0 - D7), una salida de INTR y una entrada de INTA. Esto permite una conexión directa con el 8088/8086. Al ocurrir una petición de alguna de las ocho fuentes, el 8259A activa la pata INTR. Al terminar de ejecutar la instrucción en curso, el microprocesador activa la pata INTA, lo que provoca que el 8259A envíe por el bus de datos un número de ocho bits (de 0 a 255) llamado tipo de interrupción (programable por el usuario durante la inicialización del 8259A), que el 8086/8088 utiliza para saber cuál es la fuente de interrupción. A continuación busca en la tabla de vectores de interrupción la dirección del manejador de interrupción (*interrupt handler*). Esto se hace de la siguiente manera. Se multiplica el tipo de interrupción por cuatro, y se toman los cuatro bytes que se encuentran a partir de esa dirección. Los dos primeros indican el offset y los dos últimos el segmento del manejador, como se muestra a continuación.

Posición memoria	00	02	04	06	08	0A	0C	0E	10	12	....	3FC	3FE
	IP	CS	IP	CS	IP	CS	IP	CS	IP	CS		IP	CS
Tipo de interrupción	00		01		02		03		04			FF	

Como se puede observar, la tabla ocupa el primer kilobyte de memoria (256 tipos * 4 bytes/tipo = 1024 bytes).

Una vez que se pusieron en la pila los flags, CS e IP (en ese orden), la CPU hace IF <- 0 (deshabilita interrupciones) y TF <- 0 (deshabilita la ejecución de instrucciones paso a paso) para que el manejador de interrupción no sea interrumpido, y carga IP y CS con los valores hallados en la tabla, con lo que se transfiere el control al manejador de interrupción, que es el procedimiento encargado de atender la fuente de interrupción. Luego de haber hecho su trabajo, el manejador debe terminar indicándole al controlador de interrupciones que ya fue atendido el pedido y finalmente deberá estar la instrucción IRET, que restaura los valores de los registros IP, CS y el registro de indicadores. El manejador debe poner en la pila todos los registros que use y restaurar sus valores antes de salir (en orden inverso al de su introducción en la pila), en caso contrario el sistema corre peligro de "colgarse", ya que, al ocurrir la interrupción en cualquier momento de la ejecución del programa, se cambiarían los valores de los registros en el momento menos esperado, con consecuencias imprevisibles.

Las interrupciones por software ocurren cuando se ejecuta la instrucción INT tipo. De esta manera se pueden simular interrupciones durante la depuración de un programa. El tipo de interrupción (para poder buscar el vector en la tabla) aparece en la misma instrucción como una constante de 8 bits. Muchos sistemas operativos (programas que actúan a modo de interfaz entre los programas de los usuarios (llamados también "aplicaciones") y el hardware del sistema) utilizan esta instrucción para llamadas a servicios, lo que permite no tener que conocer la dirección absoluta del servicio, permitiendo cambios en el sistema operativo sin tener que cambiar los programas que lo ejecutan. De esta manera, una de las primeras operaciones que debe realizar dicho sistema operativo es inicializar la tabla de vectores de interrupción con los valores apropiados.

Existen algunas interrupciones predefinidas, de uso exclusivo del microprocesador, por lo que no es recomendable utilizar estos tipos de interrupción para interrupciones por hardware o software.

- **Tipo 0:** Ocurre cuando se divide por cero o el cociente es mayor que el valor máximo que permite el destino.

- **Tipo 1:** Ocurre después de ejecutar una instrucción si TF (Trap Flag) vale 1. Esto permite la ejecución de un programa paso a paso, lo que es muy útil para la depuración de programas.

- **Tipo 2:** Ocurre cuando se activa la pata NMI (interrupción no enmascarable).

- **Tipo 3:** Existe una instrucción INT que ocupa un sólo byte, que es la correspondiente a este tipo. En los programas depuradores (debuggers) (tales como *Debug*, *CodeView*, *Turbo Debugger*, etc.), se utiliza esta instrucción como punto de parada (para ejecutar un programa hasta una determinada dirección, fijada por el usuario del depurador, se inserta esta instrucción en la dirección correspondiente a la parada y se lanza la ejecución. Cuando el CS:IP apunte a esta dirección se ejecutará la INT 3, lo que devolverá el control del procesador al depurador). Debido a esto, si se le ordena al depurador que ejecute el programa hasta una determinada dirección en ROM (memoria de sólo lectura) (por ejemplo, para ver cómo funciona una subrutina almacenada en dicha memoria), la ejecución seguirá sin parar allí (ya que la instrucción INT 3 no se pudo escribir sobre el programa). En el 80386, con su elaborado hardware de ayuda para la depuración, se puede poner un punto de parada en ROM.

- **Tipo 4:** Ocurre cuando se ejecuta la instrucción de interrupción condicional INTO y el flag OF (Overflow Flag) vale 1.

Los tipos 5 a 31 (1F en hexadecimal) están reservados para interrupciones internas (también llamados "excepciones") de futuros microprocesadores.

Prioridad entre diferentes fuentes de interrupción:

1) Error de división, INT *n* (no enmascarable), INTO.
2) NMI (no enmascarable).
3) INTR (enmascarable mediante IF).
4) Ejecución paso a paso (enmascarable mediante TF).

Instrucciones de 8086 y 8088

INSTRUCCIONES DE TRANSFERENCIA DE DATOS (No afectan flags)

MOV *dest,src*

Copia el contenido del operando fuente (*src*) en el destino (*dest*).
Operación: *dest <- src*
Las posibilidades son:

1. **MOV** *reg,{reg|mem|inmed}*
2. **MOV** *mem,{reg|inmed}*
3. **MOV** *{reg16|mem16},{CS|DS|ES|SS}*
4. **MOV** *{DS|ES|SS},{reg16|mem16}*

PUSH *src*

Pone el valor en el tope del stack.
Operación: *SP <- SP - 2, [SP+1:SP] <- src* donde src = {reg16|mem16|CS|DS|ES|SS}.

POP *dest*

Retira el valor del tope del stack poniéndolo en el lugar indicado.
Operación: *dest <- [SP+1:SP], SP <- SP + 2* donde dest = {reg16|mem16|DS|ES|SS}.

XCHG *reg,{reg|mem}*

Intercambia ambos valores.

IN *{AL|AX},{DX|inmed (1 byte)}*

Pone en el acumulador el valor hallado en el port indicado.

OUT *{DX|inmed (1 byte)},{AL|AX}*

Pone en el port indicado el valor del acumulador.

XLAT

Realiza una operación de traducción de un código de un byte a otro código de un byte mediante una tabla.
Operación: *AL <- [BX+AL]*

LEA *reg,mem*

Almacena la dirección efectiva del operando de memoria en un registro.
Operación: *reg <- dirección mem*

LDS *reg,mem32*

Operación: *reg <- [mem], DS <- [mem+2]*

LES *reg,mem32*

Operación: *reg <- [mem], ES <- [mem+2]*

LAHF

Copia en el registro AH la imagen de los ocho bits menos significativos del registro de indicadores.
Operación: *AH <- SF:ZF:X:AF:X:PF:X:CF*

SAHF

Almacena en los ocho bits menos significativos del registro de indicadores el valor del registro AH.
Operación: *SF:ZF:X:AF:X:PF:X:CF <- AH*

PUSHF

Almacena los flags en la pila.
Operación: *SP <- SP - 2, [SP+1:SP] <- Flags.*

POPF

Pone en los flags el valor que hay en la pila.
Operación: *Flags <- [SP+1:SP], SP <- SP + 2*

INSTRUCCIONES ARITMETICAS (Afectan los flags AF, CF, OF, PF, SF, ZF)

ADD *dest,src*

Operación: *dest <- dest + src.*

ADC *dest,src*

Operación: *dest <- dest + src + CF.*

SUB *dest,src*

Operación: *dest <- dest - src.*

SBB *dest,src*

Operación: *dest <- dest - src - CF.*

CMP *dest,src*

Operación: *dest - src (sólo afecta flags).*

INC *dest*

Operación: *dest <- dest + 1 (no afecta CF).*

DEC *dest*

Operación: *dest <- dest - 1 (no afecta CF).*

NEG *dest*

Operación: *dest <- - dest.*

donde *dest = {reg|mem}* y *src = {reg|mem|inmed}* no pudiendo ambos operandos estar en memoria.

DAA

Corrige el resultado de una suma de dos valores BCD empaquetados en el registro AL (debe estar inmediatamente después de una instrucción ADD o ADC). OF es indefinido después de la operación.

DAS

Igual que **DAA** pero para resta (debe estar inmediatamente después de una instrucción SUB o SBB).

AAA

Lo mismo que **DAA** para números BCD desempaquetados.

AAS

Lo mismo que **DAS** para números BCD desempaquetados.

AAD

Convierte AH:AL en BCD desempaquetado a AL en binario.
Operación: *AL <- AH * 0Ah + AL*, *AH <- 0*. Afecta PF, SF, ZF, mientras que AF, CF y OF quedan indefinidos.

AAM

Convierte AL en binario a AH:AL en BCD desempaquetado.
Operación: *AH <- AL / 0Ah*, *AL <- AL mod 0Ah*. Afecta PF, SF, ZF, mientras que AF, CF y OF quedan indefinidos.

MUL *{reg8|mem8}*

Realiza una multiplicación con operandos no signados de 8 por 8 bits.
Operación: *AX <- AL * {reg8|mem8}*. CF=OF=0 si AH = 0, CF=OF=1 en caso contrario. AF, PF, SF, ZF quedan indefinidos.

MUL *{reg16|mem16}*

Realiza una multiplicación con operandos no signados de 16 por 16 bits.
Operación: *DX:AX <- AX * {reg16|mem16}*. CF=OF=0 si DX = 0, CF=OF=1 en caso contrario. AF, PF, SF, ZF quedan indefinidos.

IMUL *{reg8|mem8}*

Realiza una multiplicación con operandos con signo de 8 por 8 bits.
Operación: *AX <- AL * {reg8|mem8}* realizando la multiplicación con signo. CF = OF = 0 si el resultado entra en un byte, en caso contrario valdrán 1. AF, PF, SF, ZF quedan indefinidos.

IMUL *{reg16|mem16}*

Realiza una multiplicación con operandos con signo de 16 por 16 bits.
Operación: DX:AX <- AX * {reg16|mem16} realizando la multiplicación con signo. CF = OF = 0 si el resultado entra en dos bytes, en caso contrario valdrán 1. AF, PF, SF, ZF quedan indefinidos.

CBW

Extiende el signo de AL en AX. No se afectan los flags.

CWD

Extiende el signo de AX en DX:AX. No se afectan flags.

INSTRUCCIONES LOGICAS (Afectan AF, CF, OF, PF, SF, ZF)

AND *dest,src*

Operación: *dest <- dest and src*.

TEST *dest,src*

Operación: *dest and src*. Sólo afecta flags.

OR *dest,src*
Operación: *dest <- dest or src*.

XOR *dest,src*
Operación: *dest <- dest xor src*.

Las cuatro instrucciones anteriores ponen CF = OF = 0, AF queda indefinido y PF, SF y ZF dependen del resultado.

NOT *dest*
Operación: *dest <- Complemento a 1 de dest*. No afecta los flags.

SHL/SAL *dest,{1|CL}*
Realiza un desplazamiento lógico o aritmético a la izquierda.

SHR *dest,{1|CL}*
Realiza un desplazamiento lógico a la derecha.

SAR *dest,{1|CL}*
Realiza un desplazamiento aritmético a la derecha.

ROL *dest,{1|CL}*
Realiza una rotación hacia la izquierda.

ROR *dest,{1|CL}*
Realiza una rotación hacia la derecha.

RCL *dest,{1|CL}*
Realiza una rotación hacia la izquierda usando el CF.

RCR *dest,{1|CL}*
Realiza una rotación hacia la derecha usando el CF.

En las siete instrucciones anteriores la cantidad de veces que se rota o desplaza puede ser un bit o la cantidad de bits indicado en CL.

INSTRUCCIONES DE MANIPULACION DE CADENAS:

MOVSB
Copiar un byte de la cadena fuente al destino.
Operación:
1. ES:[DI] <- DS:[SI] (un byte)
2. DI <- DI±1
3. SI <- SI±1

MOVSW
Copiar dos bytes de la cadena fuente al destino.
Operación:

1. ES:[DI] <- DS:[SI] (dos bytes)
2. DI <- DI±2
3. SI <- SI±2

LODSB

Poner en el acumulador un byte de la cadena fuente.
Operación:

1. AL <- DS:[SI] (un byte)
2. SI <- SI±1

LODSW

Poner en el acumulador dos bytes de la cadena fuente.
Operación:

1. AX <- DS:[SI] (dos bytes)
2. SI <- SI±2

STOSB

Almacenar en la cadena destino un byte del acumulador.
Operación:

1. ES:[DI] <- AL (un byte)
2. DI <- DI±1

STOSW

Almacenar en la cadena destino dos bytes del acumulador.
Operación:

1. ES:[DI] <- AX (dos bytes)
2. DI <- DI±2

CMPSB

Comparar un byte de la cadena fuente con el destino.
Operación:

1. DS:[SI] - ES:[DI] (Un byte, afecta sólo los flags)
2. DI <- DI±1
3. SI <- SI±1

CMPSW

Comparar dos bytes de la cadena fuente con el destino.
Operación:

1. DS:[SI] - ES:[DI] (Dos bytes, afecta sólo los flags)
2. DI <- DI±2
3. SI <- SI±2

SCASB

Comparar un byte del acumulador con la cadena destino.
Operación:

1. AL - ES:[DI] (Un byte, afecta sólo los flags)
2. DI <- DI±1

SCASW

Comparar dos bytes del acumulador con la cadena destino.

Operación:
1. AX - ES:[DI] (Dos byte, afecta sólo los flags)
2. DI <- DI±2

En todos los casos el signo **+** se toma si el indicador DF vale cero. Si vale 1 hay que tomar el signo **-**.

Prefijo para las instrucciones MOVSB, MOVSW, LODSB, LODSW, STOSB y STOSW:

- **REP:** Repetir la instrucción CX veces.

Prefijos para las instrucciones CMPSB, CMPSW, SCASB, SCASW:

- **REPZ**/**REPE**: Repetir mientras que sean iguales hasta un máximo de CX veces.
- **REPNZ**/**REPNE**: Repetir mientras que sean diferentes hasta un máximo de CX veces.

INSTRUCCIONES DE TRANSFERENCIA DE CONTROL (No afectan los flags):

JMP *label*
Saltar hacia la dirección *label*.

CALL *label*
Ir al procedimiento cuyo inicio es *label*. Para llamadas dentro del mismo segmento equivale a *PUSH IP*: *JMP label*, mientras que para llamadas entre segmentos equivale a *PUSH CS*: *PUSH IP*: *JMP label*.

RET
Retorno de procedimiento.

RET *inmed*
Retorno de procedimiento y *SP <- SP + inmed*.

Variaciones de la instrucción de retorno:

RETN *[inmed]*
En el mismo segmento de código. Equivale a *POP IP* [:*SP <- SP + inmed*].

RETF *[inmed]*
En otro segmento de código. Equivale a *POP IP*: *POP CS* [:*SP <- SP + inmed*]

Saltos condicionales aritméticos (usar después de CMP):

- Aritmética signada (con números positivos, negativos y cero)

  JL *etiqueta*/**JNGE** *etiqueta*
  Saltar a *etiqueta* si es menor.

  JLE *etiqueta*/**JNG** *etiqueta*
  Saltar a *etiqueta* si es menor o igual.

  JE *etiqueta*
  Saltar a *etiqueta* si es igual.

JNE *etiqueta*
Saltar a *etiqueta* si es distinto.

JGE *etiqueta*/**JNL** *etiqueta*
Saltar a *etiqueta* si es mayor o igual.

JG *etiqueta*/**JNLE** *etiqueta*
Saltar a *etiqueta* si es mayor.

- Aritmética sin signo (con números positivos y cero)

JB *etiqueta*/**JNAE** *etiqueta*
Saltar a *etiqueta* si es menor.

JBE *etiqueta*/**JNA** *etiqueta*
Saltar a *etiqueta* si es menor o igual.

JE *etiqueta*
Saltar a *etiqueta* si es igual.

JNE *etiqueta*
Saltar a *etiqueta* si es distinto.

JAE *etiqueta*/**JNB** *etiqueta*
Saltar a *etiqueta* si es mayor o igual.

JA *etiqueta*/**JNBE** *etiqueta*
Saltar a *etiqueta* si es mayor.

Saltos condicionales según el valor de los indicadores:

JC *label*
Saltar si hubo arrastre/préstamo (CF = 1).

JNC *label*
Saltar si no hubo arrastre/préstamo (CF = 0).

JZ *label*
Saltar si el resultado es cero (ZF = 1).

JNZ *label*
Saltar si el resultado no es cero (ZF = 0).

JS *label*
Saltar si el signo es negativo (SF = 1).

JNS *label*
Saltar si el signo es positivo (SF = 0).

JP/JPE *label*
Saltar si la paridad es par (PF = 1).

JNP/JPO *label*
Saltar si la paridad es impar (PF = 0).

Saltos condicionales que usan el registro CX como contador:

LOOP *label*
Operación: CX <- CX-1. Saltar a label si CX<>0.

LOOPZ/LOOPE *label*
Operación: CX <- CX-1. Saltar a label si CX <> 0 y ZF = 1.

LOOPNZ/LOOPNE *label*
Operación: CX <- CX-1. Saltar a label si CX <> 0 y ZF = 0.

JCXZ *label*
Operación: Salta a label si CX = 0.

Interrupciones:

INT *número*
Salva los flags en la pila, hace TF=IF=0 y ejecuta la interrupción con el número indicado.

INTO
Interrupción condicional. Si OF = 1, hace **INT** 4.

IRET
Retorno de interrupción. Restaura los indicadores del stack.

INSTRUCCIONES DE CONTROL DEL PROCESADOR

CLC
CF <- 0.
STC
CF <- 1.
CMC
CF <- 1 - CF.
NOP
No hace nada.
CLD
DF <- 0 (Dirección ascendente).
STD
DF <- 1 (Dirección descendente).
CLI
IF <- 0 (Deshabilita interrupciones enmascarables).
STI

IF <- 1 (Habilita interrupciones enmascarables).

HLT

Detiene la ejecución del procesador hasta que llegue una interrupción externa.

WAIT

Detiene la ejecución del procesador hasta que se active el pin TEST del mismo.

LOCK

Prefijo de instrucción que activa el pin LOCK del procesador.

OPERADORES

Operadores aritméticos

+, **-**, ***, /, **MOD** (resto de la división).

Operadores lógicos AND, **OR**, **XOR**, **NOT**, **SHR**, **SHL**.
Para los dos últimos operadores, el operando derecho indica la cantidad de bits a desplazar hacia la derecha (para **SHR**) o izquierda (para **SHL**) el operando izquierdo.

Operadores relacionales

Valen cero si son falsos y 65535 si son verdaderos.

- **EQ**: Igual a.
- **NE**: Distinto de.
- **LT**: Menor que.
- **GT**: Mayor que.
- **LE**: Menor o igual a.
- **GE**: Mayor o igual a.

Operadores analíticos

Descomponen operandos que representan direcciones de memoria en sus componentes.

SEG *memory-operand*: Retorna el valor del segmento.

OFFSET *memory-operand*: Retorna el valor del offset.

TYPE *memory-operand*: Retorna un valor que representa el tipo de operando: BYTE = 1, WORD = 2, DWORD = 4 (para direcciones de datos) y NEAR = -1 y FAR = -2 (para direcciones de instrucciones).

LENGHT *memory-operand*: Se aplica solamente a direcciones de datos. Retorna un valor numérico para el número de unidades (bytes, words o dwords) asociados con el operando. Si el operando es una cadena retorna el valor 1.

Ejemplo: Dada la directiva *PALABRAS DW 50 DUP (0)*, el valor de **LENGHT** PALABRAS es 50, mientras que dada la directiva *CADENA DB "cadena"* el valor de **LENGHT** CADENA es 1.

SIZE *memory-operand*: **LENGHT** *memory-operand* * **TYPE** *memory-operand*.

Operadores sintéticos

Componen operandos de direcciones de memoria a partir de sus componentes.

type **PTR** *memory-operand*: Compone un operando de memoria que tiene el mismo segmento y offset que el especificado en el operando derecho pero con el tipo (BYTE, WORD, DWORD, NEAR o FAR) especificado en el operando izquierdo.

THIS *type*: Compone un operando de memoria con el tipo especificado que tiene el segmento y offset que la próxima ubicación a ensamblar.

Operadores de macros

Son operadores que se utilizan en las definiciones de macros. Hay cinco: **&**, <>, **!**, **%** y **;;**.

&*parámetro*: reemplaza el parámetro con el valor actual del argumento.

<*texto*>: trata una serie de caracteres como una sola cadena. Se utiliza cuando el texto incluye comas, espacios u otros símbolos especiales.

!*carácter*: trata el carácter que sigue al operador **!** como un carácter en vez de un símbolo o separador.

%*texto*: trata el texto que sigue a continuación del operador **%** como una expresión. El ensamblador calcula el valor de la expresión y reemplaza el texto por dicho valor.

sentencia **;;***comentario*: Permite definir comentarios que aparecerán en la definición de la macro pero no cada vez que éste se invoque en el listado fuente que genera el ensamblador.

DIRECTIVAS (Instrucciones para el ensamblador)

Definición de símbolos

EQU: Define nombres simbólicos que representan valores u otros valores simbólicos. Las dos formas son:
nombre **EQU** *expresión*
nuevo_nombre **EQU** *viejo_nombre*
Una vez definido un nombre mediante EQU, no se puede volver a definir.

=: Es similar a **EQU** pero permite que el símbolo se pueda redefinir. Sólo admite la forma: *nombre* = *expresión*.

Definición de datos

Ubica memoria para un ítem de datos y opcionalmente asocia un nombre simbólico con esa dirección de memoria y/o genera el valor inicial para ese ítem.

[nombre] **DB** *valor_inicial [, valor_inicial...]*
donde *valor_inicial* puede ser una cadena o una expresión numérica cuyo resultado esté entre -255 y 255.

[nombre] **DW** *valor_inicial [, valor_inicial...]*
donde *valor_inicial* puede ser una expresión numérica cuyo resultado esté entre -65535 y 65535 o un operando de memoria en cuyo caso se almacenará el offset del mismo.

[nombre] **DD** *valor_inicial [, valor_inicial...]*
donde *valor_inicial* puede ser una constante cuyo valor esté entre -4294967295 y 4294967295, una expresión numérica cuyo valor absoluto no supere 65535, o bien un operando de memoria en cuyo caso se almacenarán el offset y el segmento del mismo (en ese orden).

Si se desea que no haya valor inicial, deberá utilizarse el símbolo **?**.

Otra forma de expresar el valor inicial es:
cuenta **DUP** *(valor_inicial [, valor_inicial...])* donde *cuenta* es la cantidad de veces que debe repetirse lo que está entre paréntesis.

Definición de segmentos

Organizan el programa para utilizar los segmentos de memoria del microprocesador 8088. Estos son **SEGMENT**, **ENDS**, **DOSSEG**, **ASSUME**, **GROUP**.

nombre_segm **SEGMENT** *[alineación][combinación]['clase']*
sentencias
nombre_segm **ENDS**

Alineación: define el rango de direcciones de memoria para el cual puede elegirse el inicio del segmento. Hay cinco posibles:

1. **BYTE**: El segmento comienza en el siguiente byte.
2. **WORD**: El segmento comienza en la siguiente dirección par.
3. **DWORD**: Comienza en la siguiente dirección múltiplo de 4.
4. **PARA**: Comienza en la siguiente dirección múltiplo de 16.
5. **PAGE**: Comienza en la siguiente dirección múltiplo de 256.

Si no se indica la alineación ésta será**PARA**.

Combinación: define cómo combinar segmentos que tengan el mismo nombre. Hay cinco posibles:

1. **PUBLIC**: Concatena todos los segmentos que tienen el mismo nombre para formar un sólo segmento. Todas las direcciones de datos e instrucciones se representan la distancia entre el inicio del segmento y la dirección correspondiente. La longitud del segmento formado será la suma de las longitudes de los segmentos con el mismo nombre.
2. **STACK**: Es similar a PUBLIC. La diferencia consiste que, al comenzar la ejecución del programa, el registro SS apuntará a este segmento y SP se inicializará con la longitud en bytes de este segmento.
3. **COMMON**: Pone el inicio de todos los segmentos teniendo el mismo nombre en la misma dirección de memoria. La longitud del segmento será la del segmento más largo.
4. **MEMORY**: Es igual a PUBLIC.

5. **AT** *dirección_de_segmento*: Hace que todas las etiquetas y direcciones de variables tengan el segmento especificado por la expresión contenida en dirección_de_segmento. Este segmento no puede contener código o datos con valores iniciales. Todos los símbolos que forman la expresión *dirección_de_segmento* deben conocerse en el primer paso de ensamblado.

Si no se indica *combinación*, el segmento no se combinará con otros del mismo nombre (combinación "privada").

Clase: Es una forma de asociar segmentos con diferentes nombres, pero con propósitos similares. Sirve también para identificar el segmento de código. Debe estar encerrado entre comillas simples.

El linker pone los segmentos que tengan la misma clase uno a continuación de otro, si bien siguen siendo segmentos diferentes. Además supone que los segmentos de código tiene clase **CODE** o un nombre con el sufijo CODE.

DOSSEG: Esta directiva especifica que los segmentos deben ordenarse según la convención de DOS. Esta es la convención usada por los compiladores de lenguajes de alto nivel.

GROUP: Sirve para definir grupos de segmentos. Un grupo es una colección de segmentos asociados con la misma dirección inicial. De esta manera, aunque los datos estén en diferentes segmentos, todos pueden accederse mediante el mismo registro de segmento. Los segmentos de un grupo no necesitan ser contiguos.
Sintaxis: *nombre_grupo* **GROUP** *segmento [, segmento...]*

ASSUME: Sirve para indicar al ensamblador qué registro de segmento corresponde con un segmento determinado. Cuando el ensamblador necesita referenciar una dirección debe saber en qué registro de segmento lo apunta.
Sintaxis: **ASSUME** *reg_segm:nombre [, reg_segm:nombre...]*
donde el nombre puede ser de segmento o de grupo, una expresión utilizando el operador SEG o la palabra NOTHING, que cancela la selección de registro de segmento hecha con un ASSUME anterior.

Control del ensamblador

ORG *expresión*: El offset del código o datos a continuación será la indicada por la expresión. Todos los símbolos que forman la expresión deben conocerse en el primer paso de ensamblado.

EVEN: Hace que la próxima instrucción o dato se ensamble en la siguiente posición par.

END *[etiqueta]*: Debe ser la última sentencia del código fuente. La *etiqueta* indica dónde debe comenzar la ejecución del programa. Si el programa se compone de varios módulos, sólo el módulo que contiene la dirección de arranque del programa debe contener la directiva **END** *etiqueta*. Los demás módulos deberán terminar con la directiva **END** (sin *etiqueta*).

Definición de procedimientos

Los procedimientos son secciones de código que se pueden llamar para su ejecución desde distintas partes del programa.

etiqueta **PROC** *{NEAR|FAR}*
sentencias
etiqueta **ENDP**

Ensamblado condicional

Verifican una condición determinada y si se cumple, ensambla una porción de código. Opcionalmente puede ensamblarse otra porción de código si la condición no se cumple. Son los siguientes: **IF**, **IF1**, **IF2**, **IFB**, **IFDEF**, **IFDIF**, **IFE**, **IFIDN**, **IFNB**, **IFNDEF**, **ENDIF**, **ELSE**.

*{***IF**|**IFE***}* *condición*
sentencias ;Se ejecutan si es cierta (**IF**) o falsa (**IFE**).
*[***ELSE**
sentencias] ;Se ejecutan si es falsa (**IF**) o cierta (**IFE**).
ENDIF

La directiva **ELSE** y sus *sentencias* son opcionales. **ENDIF** termina el bloque y es obligatorio. Se pueden anidar directivas condicionales.

IF1 permite el ensamblado de las sentencias sólo en el primer paso, mientras que **IF2** lo permite en el segundo paso.

IFDEF *nombre* permite el ensamblado de las sentencias si el *nombre* está definido, mientras que **IFNDEF** *nombre* lo permite si no está definido.

IFB <*argumento*> permite el ensamblado si el argumento en una macro es blanco (no se pasó el argumento).

IFNB <*argumento*> permite el ensamblado si el argumento en una macro no es blanco (se pasó el argumento).

IFIDN <*argumento1*>, <*argumento2*> permite el ensamblado si los dos parámetros pasados a la macro son idénticos.

IFDIF <*argumento1*>,<*argumento2*> permite el ensamblado si los dos parámetros pasados a la macro son diferentes.

Macros: Las macros asignan un nombre simbólico a un bloque de sentencias fuente. Luego se puede usar dicho nombre para representar esas sentencias. Opcionalmente se pueden definir parámetros para representar argumentos para la macro.

Definición de macros

nombre_macro **MACRO** *[parámetro [,parámetro...]]*
*[***LOCAL** *nombre_local[,nombre_local...]*
sentencias
ENDM

Los parámetros son opcionales. Si existen, entonces también aparecerán en algunas de las sentencias en la definición de la macro. Al invocar la macro mediante:

nombre_macro [argumento [,argumento..]]

se ensamblarán las sentencias indicadas en la macro teniendo en cuenta que cada lugar donde aparezca un parámetro se reemplazará por el argumento correspondiente.

El *nombre_local* de la directiva **LOCAL** es un nombre simbólico temporario que será reemplazado por un único nombre simbólico (de la forma *??número*) cuando la macro se invoque.

Todas las etiquetas dentro de la macro deberán estar indicadas en la directiva **LOCAL** para que el ensamblador no genere un error indicando que un símbolo está definido varias veces.

La directiva **EXITM** (usada dentro de la definición de la macro) sirve para que no se ensamblen más sentencias de la macro (se usa dentro de bloques condicionales).

PURGE *nombre_macro [,nombre_macro...]*: Borra las macros indicadas de la memoria para poder utilizar este espacio para otros símbolos.

Definición de bloques de repetición

Son tres: **REPT**, **IRP** e **IRPC**. Como en el caso de la directiva **MACRO**, se puede incluir la sentencias **LOCAL** y **EXITM** y deben terminarse con la directiva **ENDM**.

REPT *expresión*
sentencias
ENDM

La *expresión* debe poder ser evaluada en el primer paso del ensamblado y el resultado deberá estar entre 0 y 65535.

Esta *expresión* indica la cantidad de veces que debe repetirse el bloque.

IRP *parámetro, <argumento [,argumento...]>*
sentencias
ENDM

El parámetro se reemplaza por el primer argumento y se ensamblan las sentencias dentro del bloque. Luego el parámetro se reemplaza por el segundo argumento y se ensamblan las sentencias y así sucesivamente hasta agotar los argumentos.

IRPC *parámetro, cadena*

sentencias
ENDM

Es similar a **IRP** con la diferencia que el parámetro se reemplaza por cada carácter de la cadena. Si ésta contiene comas, espacios u otros caracteres especiales deberá encerrarse con paréntesis angulares (<>).

Procesador: Indican el tipo de procesador y coprocesador en el que se va a ejecutar el programa. Los de procesador son: **.8086**, **.186**, **.286**, **.386**, **.486** y **.586** para instrucciones en modo real, **.286P**, **.386P**, **.486P** y **.586P** para instrucciones privilegiadas, **.8087**, **.287** y **.387** para coprocesadores. Deben ubicarse al principio del código fuente. Habilitan las instrucciones correspondientes al procesador y coprocesador indicado. Sin estas directivas, sólo se pueden ensamblar instrucciones del 8086 y 8087.

Referencias externas al módulo

Sirve para poder particionar un programa en varios archivos fuentes o módulos. Son imprescindibles si se hace un programa en alto nivel con procedimientos en assembler. Hay tres: **PUBLIC**, **EXTRN** e **INCLUDE**.

PUBLIC *nombre[, nombre...]*: Estos nombres simbólicos se escriben en el archivo objeto. Durante una sesión con el linker, los símbolos en diferentes módulos pero con los mismos nombres tendrán la misma dirección.

EXTRN *nombre:tipo [,nombre:tipo...]*: Define una variable externa con el *nombre* y *tipo* (**NEAR**, **FAR**, **BYTE**, **WORD**, **DWORD** o **ABS** (número constante especificado con la directiva **EQU** o =)) especificado. El tipo debe ser el mismo que el del ítem indicado con la directiva **PUBLIC** en otro módulo.

INCLUDE *nombre_de_archivo*: Ensambla las sentencias indicadas en dicho archivo.

Segmentos simplificados

Permite definir los segmentos sin necesidad de utilizar las directivas de segmentos que aparecen más arriba.

.MODEL *modelo*: Debe estar ubicada antes de otra directiva de segmento. El *modelo* puede ser uno de los siguientes:

1. **TINY**: Los datos y el código juntos ocupan menos de 64 KB por lo que entran en el mismo segmento. Se utiliza para programas .COM. Algunos ensambladores no soportan este modelo.
2. **SMALL**: Los datos caben en un segmento de 64 KB y el código cabe en otro segmento de 64 KB. Por lo tanto todo el código y los datos se pueden acceder como **NEAR**.
3. **MEDIUM**: Los datos entran en un sólo segmento de 64 KB, pero el código puede ser mayor de 64 KB. Por lo tanto, código es **FAR**, mientras que los datos se acceden como **NEAR**.
4. **COMPACT**: Todo el código entra en un segmento de 64 KB, pero los datos no (pero no pueden haber matrices de más de 64 KB). Por lo tanto, código es **NEAR**, mientras que los datos se acceden como **FAR**.
5. **LARGE**: Tanto el código como los datos pueden ocupar más de 64 KB (pero no pueden haber matrices de más de 64 KB), por lo que ambos se acceden como **FAR**.
6. **HUGE**: Tanto el código como los datos pueden ocupar más de 64 KB (y las matrices también), por lo que ambos se acceden como **FAR** y los punteros a los elementos de las matrices también son

FAR.

.STACK *[size]*: Define el segmento de pila de la longitud especificada.

.CODE *[name]*: Define el segmento de código.

.DATA: Define un segmento de datos NEAR con valores iniciales.

.DATA?: Define un segmento de datos NEAR sin valores iniciales.

.FARDATA *[name]*: Define un segmento de datos FAR con valores iniciales.

.FARDATA? *[name]*: Define un segmento de datos FAR sin valores iniciales.

.CONST: Sefine un segmento de datos constantes.

Los siguientes símbolos están definidos cuando se usan las directivas anteriores:

- **@curseg**: Tiene el nombre del segmento que se está ensamblando.
- **@filename**: Representa el nombre del archivo fuente (sin la extensión)
- **@codesize**: Vale 0 para los modelos **SMALL** y **COMPACT** (código **NEAR**), y vale 1 para los modelos **MEDIUM**, **LARGE** y **HUGE** (código **FAR**).
- **@datasize**: Vale 0 para los modelos **SMALL** y **MEDIUM** (datos **NEAR**), vale 1 para los modelos **COMPACT** y **LARGE** (datos FAR) y vale 2 para el modelo **HUGE** (punteros a matrices **FAR**).
- **@code**: Nombre del segmento definido con la directiva **.CODE**.
- **@data**: Nombre del segmento definido con la directivas **.DATA**, **.DATA?**, **.CONST** y **.STACK** (los cuatro están en el mismo segmento).
- **@fardata**: Nombre del segmento definido con la directiva **.FARDATA**.
- **@fardata?**: Nombre del segmento definido con la directiva **.FARDATA?**.

El coprocesador matemático 8087

Introducción

El procesador de datos numérico (NDP) 8087 aumenta el juego de instrucciones del 8086/8088 mejorando su capacidad de tratamiento de números. Se utiliza como procesador paralelo junto al 8086/8088 añadiendo 8 registros de coma flotante de 80 bits así como instrucciones adicionales. Utiliza su propia cola de instrucciones para controlar el flujo de instrucciones del 8086/8088, ejecutando sólo aquellas instrucciones que le corresponden, e ignorando las destinadas a la CPU 8086/8088. El 8086/8088 deberá funcionar en modo máximo para poder acomodar el 8087. Las instrucciones del NDP 8087 incluyen un juego completo de funciones aritméticas así como un potente núcleo de funciones exponenciales, logarítmicas y trigonométricas. Utiliza un formato interno de números en coma flotante de 80 bits con el cual gestiona siete formatos exteriores.

Como detalle constructivo, cabe mencionar que el 8087 posee 45.000 transistores y consume 3 watt.

Los números y su tratamiento

Hay dos tipos de números que aparecen normalmente durante el cálculo: los números enteros y los números reales. Aunque los enteros no dejan de ser un subconjunto de los reales, la computadora trabaja de formas distintas con ambos. Los enteros son fáciles de tratar para la computadora. Los chips microprocesadores de propósito general trabajan con números enteros utilizando la representación binaria de números en complemento a dos. Pueden trabajar incluso con números que excedan el tamaño de la palabra a base de fragmentar los números en unidades más pequeñas. Es lo que se llama aritmética de precisión múltiple. Los números reales, sin embargo, son más difíciles. En primer lugar, la mayoría de ellos nunca pueden representarse exactamente. La representación en coma flotante permite una representación aproximada muy buena en la práctica de los números reales. La representación en coma flotante es en el fondo una variación de la notación científica que puede verse en el visualizador de cualquier calculadora. Con este sistema, la representación de un número consta de tres partes: el *signo*, el *exponente* y la *mantisa*. Antes de continuar, veremos por qué es necesaria esta representación. Para los principiantes en el tema, consideremos el siguiente ejemplo de notación científica: *4,1468E2*. En este ejemplo el signo es positivo (porque no está se supone positivo), la mantisa es *4,1468* y el exponente *2*. El valor representado por este número es: *4,1468 x 10 ^ 2 = 414,68*.

Precisión y rango

En la representación de números reales aparecen dos problemas fundamentales: la precisión y el rango.

Por precisión se entiende la exactitud de un número. En la representación de coma flotante, la mantisa es la encargada de la precisión. La mantisa contiene los dígitos significativos del número independientemente de dónde esté colocada la coma decimal. Si queremos aumentar la precisión de un esquema de representación de punto flotante, basta con añadir dígitos a la mantisa.

La precisión no es problema en el caso de los enteros, puesto que todo entero viene representado exactamente por su representación en complemento a dos. La representación precisa de los números reales sí es un problema ya que la mayoría de ellos tiene infinitos dígitos, cosa imposible de representar exactamente en una máquina con un número finito de componentes. Puesto que las computadoras permiten almacenar un número finito de dígitos, la representación de los números reales debe realizarse necesariamente por medio de aproximaciones.

El rango está relacionado con el tamaño de los números que se pueden representar. En los enteros, el rango depende del número de bits que se utilicen. Por ejemplo, con 16 bits pueden representarse números comprendidos entre -32768 y 32767. Para representar todos los enteros sería de nuevo necesario un número infinito de bits. Como puede verse, incluso los enteros tienen un rango restringido. En la notación de coma flotante, es el exponente el que fija el rango. Separando los problemas de precisión y de rango, la notación en coma flotante permite obtener rangos muy grandes con precisión razonable. El rango sigue siendo finito, porque sólo se puede representar un número finito de dígitos del exponente, pero tales dígitos permiten representar un número bastante grande de una forma compacta.

Implementación de la representación en coma flotante

Hay muchas formas de implementar la representación en coma flotante. Intel sigue el método propuesto por el instituto de normalización IEEE. Para ver cómo trabaja, volvamos de nuevo a la notación científica. Por ejemplo, en notación científica decimal el número *414,68* suele escribirse como *4,1468E2* y el número *-0,00345* suele representarse como *-3,45E-3*. En cada caso el número consta de un signo (en el primer número no aparece por ser positivo), una mantisa y un exponente. La "E" indica sencillamente "exponente" y puede leerse como "diez a la". Normalmente, la coma decimal se coloca a la derecha del primer dígito significativo. Cuando esto ocurre, se dice que el número está normalizado. El número cero es un caso particular que no admite normalización. Puede representarse por una mantisa cero. El exponente en tal caso es arbitrario, aunque se siguen algunos convenios dependiendo de la implementación particular.

En contraste con la notación científica usual que utiliza la base 10, las computadoras utilizan la base 2 a la vez como base de la expresión exponencial, y como base para representar la mantisa y el exponente. Así, por ejemplo, el número *5,325* sería *4 + 1 + 1/4 + 1/8 = 101,111* (base 2) y se escribe en notación científica "binaria" como *+1,01011E2*. En este caso "E" debe leerse como "2 a la", y el incrementar o decrementar el exponente significa desplazar convenientemente la coma binaria.

Cuando se almacena un número binario en coma flotante en la máquina, se utiliza un bit para el signo, varios para la mantisa y varios para el exponente. Normalmente el bit de signo precede a los demás, después vienen los bits de exponente y finalmente los de la mantisa. La forma más común de guardar el exponente es adicionarle una constante, que recibe el nombre de "exceso". El número que se guarda recibe el nombre de "exponente en notación de exceso". A la inversa, dada la representación de un exponente, para recuperar su valor verdadero, basta con restarle el exceso. Como exceso se toma un número cuyo valor es aproximadamente igual a la mitad del rango de los posibles exponentes, para poder representar más o menos la misma cantidad de exponentes positivos que negativos.

Como ejemplo práctico, veremos cómo se representan los números reales en los registros del NDP 8087. Este formato particular recibe el nombre de **formato real temporal**. Todos los datos en el interior del NDP se almacenan de esta manera.

Descripción	Signo	Exponente	Mantisa
Bits	79	78 ... 64	63 ... 0

El formato real temporal requiere 80 bits: uno para el signo, 15 para el exponente y 64 para la mantisa. El exceso es *2 ^ 14 - 1 = 16383*. El menor valor positivo que se puede representar es *2 ^ -16382 = 3,36 x 10 ^ -4932*, mientras que el mayor valor positivo representable es igual a *2 ^ 16384 = 1,19 x 10 ^ 4932*.

El NDP 8087 como procesador paralelo

El NDP 8087 actúa también como procesador paralelo. Esto es, comparte con la CPU el mismo bus, y el mismo flujo de instrucciones.

Las instrucciones del NDP aparecen mezcladas con el flujo de instrucciones de la CPU. Sin embargo, cada cual selecciona y ejecuta sólo las que le corresponden. Todas las instrucciones correspondientes al NDP comienzan con una variante de la instrucción **ESC** del 8086/8088. Esta es una instrucción ficticia, con un operando ficticio que puede especificarse con cualquiera de los 24 modos de direccionamiento a memoria y otro operando ficticio que no es más que un registro.

Cada instrucción que lee la CPU, también lo lee el NDP. Cuando la CPU lee una instrucción **ESC**, el NDP se da por enterado que pronto tendrá que ponerse a trabajar. El NDP lee el código de operación, mientras que la CPU calcula la dirección de memoria, la que pone en el bus de direcciones. La CPU finalmente pasa el control al NDP, que extrae los bytes de datos necesarios, hace los cálculos correspondientes, y coloca los resultados sobre el bus.

El NDP como ampliación del 8086/8088

El NDP 8087 amplía el juego de instrucciones del 8086/8088, incluyendo operaciones en coma flotante. Estas nuevas instrucciones implementan por hardware el paquete Intel de tratamiento de números en punto flotante, proveyendo un método mucho mejor de ejecución de estos algoritmos. Dicho de otra manera, las rutinas de tratamiento de números en coma flotante pueden realizarse a velocidades muy altas. Intel estima que el NDP 8087 ejecuta las operaciones de coma flotante hasta 100 veces más rápidamente que una CPU 8086 que los realizase por software. A continuación puede verse una comparación entre la velocidad del 8087 y del paquete de software del 8086. En ambos casos, los procesadores funcionaban a 5 MHz. Los tiempos están dados en microsegundos.

Operación en punto flotante	8087	8086
Suma en signo y magnitud	14	1600
Resta en signo y magnitud	18	1600
Multiplicación (precisión simple)	19	1600
Multiplicación (precisión doble)	27	2100
División	39	3200
Comparación	9	1300
Carga (doble precisión)	10	1700
Almacena (doble precisión)	21	1200
Raíz cuadrada	36	19600
Tangente	90	13000
Exponenciación	100	17100

El NDP amplía también el conjunto de registros de la CPU 8086/8088. Los registros de la CPU están en un chip (el 8086/8088, y hay ocho registros de coma flotante de 80 bits en el otro (el 8087). Dichos

registros guardan números en el formato real temporal antes estudiado.

El coprocesador matemático trabaja simultáneamente con el procesador principal. Sin embargo, como el primero no puede manejar entrada o salida a dispositivos, la mayoría de los datos los origina la CPU. La CPU y el NDP tienen sus propios registros, que son completamente separados e inaccesibles al otro chip. Los datos se intercambian a través de la memoria, ya que ambos chips pueden acceder la memoria.

Los ocho registros se organizan como una pila. Una pila es tal vez la mejor manera de organizar los datos para evaluar expresiones algebraicas complejas. Los nombres de estos registros son **ST(0)**, **ST(1)**, **ST(2)**, ..., **ST(7)**. El nombre simbólico **ST** (*Stack Top*) es equivalente a **ST(0)**. Al poner (push) un número en la pila, **ST(0)** contendrá el número recién ingresado, **ST(1)** será el valor anterior de **ST(0)**, **ST(2)** será el valor anterior de **ST(1)**, y así sucesivamente, con lo que se perderá el valor anterior de **ST(7)**. El NDP 8087 tiene algunos registros más: un registro de estado de 16 bits, un registro de modo de 16 bits, un registro indicador de 8 bits y cuatro registros de 16 bits de salvaguarda de instrucciones y punteros a datos para la gestión de condiciones excepcionales.

El NDP es también una ampliación hardware desde el momento que no es autosuficiente. El 8087 necesita tener al lado un 8086 o un 8088 que le proporcione datos y direcciones, y controle el bus que a su vez suministrará instrucciones y operandos.

Hay varias líneas de control que interconectan el NDP y la CPU: la señal de TEST del CPU que se conecta a BUSY (ocupado) del NDP, la línea RQ0/GT0 (petición/concesión del manejo del bus), y las señales de estado de la cola (QS1, QS0).

El terminal de TEST del 8086/8088 se conecta al de BUSY del 8087. Esto permite que el 8086/8088 pueda utilizar la instrucción WAIT para sincronizar su actividad con el NDP. La forma correcta de hacerlo es conseguir que el ensamblador genere automáticamente una instrucción WAIT antes de cada instrucción del NDP, y que el programador ponga en el programa una instrucción FWAIT (sinónimo de WAIT) después de cada instrucción del NDP que cargue los datos en memoria para que puedan ser inmediatamente utilizadas por la CPU. Mientras el NDP 8087 está realizando una operación numérica, pone a 1 el terminal BUSY (y por lo tanto el terminal WAIT del CPU). Mientras la CPU ejecuta la instrucción WAIT, suspende toda actividad hasta que el terminal TEST vuelve a su estado normal (cero). Así, la secuencia de una instrucción numérica del NDP seguida de un WAIT de la CPU hace que la CPU llame al NDP y espere hasta que el último acabe.

La línea RQ/GT0 (petición/concesión del manejo del bus) la utiliza el NDP para conseguir el control del bus compartido por el NDP y la CPU. La línea de interconexión es bidireccional. Una señal (petición) del NDP a la CPU indica que el NDP quiere utilizar el bus. Para que el NDP tome control del bus, debe esperar a que la CPU le conteste (con una concesión). Cuando el NDP acaba de usar el bus, envía una señal por el mismo pin indicando que ha terminado. En este protocolo, la CPU juega el papel de amo y el NDP de esclavo. El esclavo es el que pide el bus y el amo es el que lo concede tan pronto como puede.

Hay dos terminales de estado de la cola, QS1 y QS0. Ambas líneas permiten al NDP sincronizar su cola de instrucciones con la de la CPU.

Tipos de datos del NDP 8087

El NDP puede trabajar con siete tipos de datos distintos: enteros de tres longitudes distintas, un tipo de BCD empaquetado, y tres tipos de representación en coma flotante. Los siete tipos de datos se almacenan

internamente en el formato real temporal de 80 bits ya discutido. Todos los tipos de datos pueden guardarse con la suficiente precisión en este formato.

Los tipos de datos son los siguientes: palabra entera de 16 bits, entero corto de 32 bits, entero largo de 64 bits (en todos los casos en representación de complemento a dos), BCD empaquetado con 18 dígitos decimales (como son dos dígitos por byte, se requieren diez bytes, estando uno de ellos reservado para el signo), real corto de 32 bits (1 bit de signo, 8 de exponente y 23 de mantisa), real largo de 64 bits (1 de signo, 11 de exponente y 52 de mantisa) y real temporal (1 de signo, 15 de exponente y 64 de mantisa). Como se vio anteriormente, el primer bit de la mantisa siempre es uno (en caso contrario el número no estaría normalizado). Este uno no aparece en los formatos real corto y real largo, por lo que la precisión de la mantisa es de 24 y 53 bits, respectivamente.

Notación polaca inversa y uso de la pila del 8087

Para entender la *notación polaca inversa* (el inventor de esta notación fue el científico polaco Lukasiewicz) de operaciones algebraicas veremos un ejemplo: sea hallar el valor de la raíz cuadrada de $3^2 + 4^2$) paso a paso.

Lo primero que se hace es hallar 3 al cuadrado, luego 4 al cuadrado, luego se suman ambos resultados y finalmente se halla la raíz cuadrada. En calculadoras que utilizan la notación polaca inversa (como las Hewlett-Packard) se apretarían las siguientes teclas: 3, x^2, ENTER, 4, x^2, +, raíz cuadrada. La tecla ENTER en estas calculadoras guardan un resultado intermedio en una pila para poder utilizarlo más adelante (en este caso, en la suma). La tecla + utiliza el número que está en la pantalla y el que se guardó en la pila anteriormente para realizar la suma.

Otro ejemplo más complicado: (2 + 8) / (25 - 5^3) sería: 2, ENTER, 8, +, ENTER, 25, ENTER, 5, ENTER, 3, x^3, -, /.

Veremos la ejecución paso a paso de la secuencia recién mostrada (aquí **ST(0)** representa la pantalla, **ST(1)** lo que se puso en la pila mediante ENTER y así sucesivamente):

	ST(0)	ST(1)	ST(2)	ST3
2	2			
ENTER	2	2		
8	8	2		
+	10			
ENTER	10	10		
25	25	10		
ENTER	25	25	10	
5	5	25	10	
ENTER	5	5	25	10
3	3	5	25	10
x^3	125	25	10	
-	-100	10		
/	-0,1			

Es fundamental haber entendido ambos ejemplos, ya que el coprocesador trabaja de esta manera (primero se ponen los operandos en la pila y después se realizan las operaciones).

Hay dos operaciones básicas de acceso de datos en la pila del 8087 que son las de extraer (pop) e introducir (push). La introducción funciona de la siguiente manera: tiene un operando que es la fuente. El puntero de la pila se decrementa en uno de manera que apunte a la posición inmediata superior y, a continuación, se el dato se transfiere a este nuevo elemento superior. El extraer funciona de la siguiente manera: en primer lugar se transfiere el dato correspondiente al elemento superior de la pila a su destino, y seguidamente el puntero de pila se incrementa en uno, de manera que apunta ahora al nuevo elemento superior de la pila (que está en una posición por encima del anterior).

En el NDP, la pila está numerada respecto a su elemento superior. Dicho elemento se denota por **ST(0)**, o simplemente **ST**, y las posiciones inferiores se denotan respectivamente por **ST(1)**, **ST(2)**, ..., **ST(7)**.

Registro del 8087	Posición relativa con respecto al puntero de pila	
R0	**ST(4)**	
R1	**ST(5)**	
R2	**ST(6)**	
R3	**ST(7)**	
R4	**ST**	Puntero de pila
R5	**ST(1)**	
R6	**ST(2)**	
R7	**ST(3)**	

No es posible acceder a *Rn* por programa, sólo a *ST(n)*.

Juego de instrucciones del 8087

El NDP tiene una gran cantidad de instrucciones distintas, incluyendo transferencia y conversión de datos, comparaciones, las cuatro operaciones básicas, trigonométricas, exponenciales y logaritmos, carga de constantes especiales y de control.

Instrucciones de transferencia de números

FLD *mem*: Introduce una copia de *mem* en *ST*. La fuente debe ser un número real en punto flotante de 4, 8 ó 10 bytes. Este operando se transforma automáticamente al formato real temporal.

FLD *ST(num)*: Introduce una copia de *ST(num)* en *ST*.

FILD *mem*: Introduce una copia de *mem* en *ST*. La fuente debe ser un operando de memoria de 2, 4 u 8 bytes, que se interpreta como un número entero y se convierte al formato real temporal.

FBLD *mem*: Introduce una copia de *mem* en *ST*. La fuente debe ser un operando de 10 bytes, que se interpreta como un valor BCD empaquetado y se convierte al formato real temporal.

FST *mem*: Copia *ST* a *mem* sin afectar el puntero de pila. El destino puede ser un operando real de 4 u 8 bytes (no el de 10 bytes).

FST *ST(num)*: Copia *ST* al registro especificado.

FIST *mem*: Copia *ST* a *mem*. El destino debe ser un operando de 2 ó 4 bytes (no de 8 bytes) y se convierte automáticamente el número en formato temporal real a entero.

FSTP *mem*: Extrae una copia de *ST* en *mem*. El destino puede ser un operando de memoria de 4, 8 ó 10 bytes, donde se carga el número en punto flotante.

FSTP *ST(num)*: Extrae *ST* hacia el registro especificado.

FISTP *mem*: Extrae una copia de *ST* en *mem*. El destino debe ser un operando de memoria de 2, 4 u 8 bytes y se convierte automáticamente el número en formato temporal real a entero.

FBSTP *mem*: Extrae una copia de *ST* en *mem*. El destino debe ser un operando de memoria de 10 bytes. El valor se redondea a un valor entero, si es necesario, y se convierte a BCD empaquetado.

FXCH: Intercambia *ST(1)* y *ST*.

FXCH *ST(num)*: Intercambia *ST(num)* y *ST*.

Instrucciones de carga de constantes

Las constantes no se pueden dar como operandos y ser cargados directamente en los registros del coprocesador. Dicha constante debe estar ubicada en memoria con lo que luego se podrán usar las instrucciones arriba mencionadas. Sin embargo, hay algunas instrucciones para cargar ciertas constantes

(0, 1, pi y algunas constantes logarítmicas). Esto es más rápido que cargar las constantes muy utilizadas desde la memoria.

FLDZ: Introduce el número cero en *ST*.
FLD1: Introduce el número uno en *ST*.
FLDPI: Introduce el valor de pi en *ST*.
FLDL2E: Introduce el valor de log(2) e en *ST*.
FLDL2T: Introduce el valor de log(2) 10 en *ST*.
FLDLG2: Introduce el valor de log(10) 2 en *ST*.
FLDLN2: Introduce el valor de log(e) 2 en *ST*.

Instrucciones de transferencia de datos de control

El área de datos del coprocesador, o parte de él, puede ser guardado en memoria y luego se puede volver a cargar. Una razón para hace esto es guardar una imagen del estado del coprocesador antes de llamar un procedimiento (subrutina) y luego restaurarlo al terminar dicho procedimiento. Otra razón es querer modificar el comportamiento del NDP almacenando ciertos datos en memoria, operar con los mismos utilizando instrucciones del 8086 y finalmente cargarlo de nuevo en el coprocesador.

Se puede transferir el área de datos del coprocesador, los registros de control, o simplemente la palabra de estado o de control.

Cada instrucción de carga tiene dos formas: La forma *con espera* verifica excepciones de errores numéricos no enmascarados y espera a que sean atendidos. La forma *sin espera* (cuyo mnemotécnico comienza con "FN") ignora excepciones sin enmascarar.

FLDCW *mem2byte*: Carga la palabra de control desde la memoria.
F[N]STCW *mem2byte*: Almacena la palabra de control en la memoria.
F[N]STSW *mem2byte*: Almacena la palabra de estado en la memoria.
FLENV *mem14byte*: Carga el entorno desde la memoria.
F[N]STENV *mem14byte*: Almacena el entorno en la memoria.
FRSTOR *mem94byte*: Restaura el estado completo del 8087.
F[N]SAVE *mem94byte*: Salva el estado completo del 8087.

Instrucciones aritméticas

Cuando se usan operandos de memoria con una instrucción aritmética, el mnemotécnico de la instrucción distingue entre número real y número entero. No se pueden realizar operaciones aritméticas con números BCD empaquetados en la memoria, por lo que deberá usarse **FBLD** para cargar dichos números desde la memoria.

FADD: Hace ST(1) más ST, ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

FADD *mem*: Hace ST <- ST + [mem]. En *mem* deberá haber un número real en punto flotante.

FIADD *mem*: Hace ST <- ST + [mem]. En *mem* deberá haber un número entero en complemento a dos.

FADD *ST(num), ST*: Realiza ST(num) <- ST(num) + ST.

FADD *ST, ST(num)*: Realiza ST <- ST + ST(num).

FADDP *ST(num), ST*: Realiza ST(num) <- ST(num) + ST y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

FSUB: Hace ST(1) menos ST, ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

FSUB *mem*: Hace ST <- ST - [mem]. En *mem* deberá haber un número real en punto flotante.

FISUB *mem*: Hace ST <- ST - [mem]. En *mem* deberá haber un número entero en complemento a dos.

FSUB *ST(num), ST*: Realiza ST(num) <- ST(num) - ST.

FSUB *ST, ST(num)*: Realiza ST <- ST - ST(num).

FSUBP *ST(num), ST*: Realiza ST(num) <- ST(num) - ST y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

FSUBR: Hace ST menos ST(1), ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

FSUBR *mem*: Hace ST <- [mem] - ST. En *mem* deberá haber un número real en punto flotante.

FISUBR *mem*: Hace ST <- [mem] - ST. En *mem* deberá haber un número entero en complemento a dos.

FSUBR *ST(num), ST*: Realiza ST(num) <- ST - ST(num).

FSUBR *ST, ST(num)*: Realiza ST <- ST(num) - ST.

FSUBRP *ST(num), ST*: Realiza ST(num) <- ST - ST(num) y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

FMUL: Multiplicar el valor de ST(1) por ST, ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

FMUL *mem*: Hace ST <- ST * [mem]. En *mem* deberá haber un número real en punto flotante.

FIMUL *mem*: Hace ST <- ST * [mem]. En *mem* deberá haber un número entero en complemento a dos.

FMUL *ST(num), ST*: Realiza ST(num) <- ST(num) * ST.

FMUL *ST, ST(num)*: Realiza ST <- ST * ST(num).

FMULP *ST(num), ST*: Realiza ST(num) <- ST(num) * ST y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

FDIV: Dividir el valor de ST(1) por ST, ajusta el puntero de pila y pone el resultado en ST, por lo que

ambos operandos se destruyen.

FDIV *mem*: Hace ST <- ST / [mem]. En *mem* deberá haber un número real en punto flotante.

FIDIV *mem*: Hace ST <- ST / [mem]. En *mem* deberá haber un número entero en complemento a dos.

FDIV *ST(num), ST*: Realiza ST(num) <- ST(num) / ST.

FDIV *ST, ST(num)*: Realiza ST <- ST / ST(num).

FDIVP *ST(num), ST*: Realiza ST(num) <- ST(num) / ST y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

FDIVR: Hace ST dividido ST(1), ajusta el puntero de pila y pone el resultado en ST, por lo que ambos operandos se destruyen.

FDIVR *mem*: Hace ST <- [mem] / ST. En *mem* deberá haber un número real en punto flotante.

FIDIVR *mem*: Hace ST <- [mem] / ST. En *mem* deberá haber un número entero en complemento a dos.

FDIVR *ST(num), ST*: Realiza ST(num) <- ST / ST(num).

FDIVR *ST, ST(num)*: Realiza ST <- ST(num) / ST.

FDIVRP *ST(num), ST*: Realiza ST(num) <- ST / ST(num) y retira el valor de ST de la pila, con lo que ambos operandos se destruyen.

FABS: Pone el signo de ST a positivo (valor absoluto).

FCHS: Cambia el signo de ST.

FRNDINT: Redondea ST a un entero.

FSQRT: Reemplaza ST con su raíz cuadrada.

FSCALE: Suma el valor de ST(1) al exponente del valor en ST. Esto efectivamente multiplica ST por dos a la potencia contenida en ST(1). ST(1) debe ser un número entero.

FPREM: Calcula el resto parcial hallando el módulo de la división de los dos registros de la pila que están el tope. El valor de ST se divide por el de ST(1). El resto reemplaza el valor de ST. El valor de ST(1) no cambia. Como esta instrucción realiza sustracciones repetidas, puede tomar mucho tiempo si los operandos son muy diferentes en magnitud. Esta instrucción se utiliza a veces en funciones trigonométricas.

FXTRACT: Luego de esta operación, ST contiene el valor de la mantisa original y ST(1) el del exponente.

Control del flujo del programa:

El coprocesador matemático tiene algunas instrucciones que pone algunos indicadores en la palabra de estado. Luego se pueden utilizar estos bits en saltos condicionales para dirigir el flujo del programa. Como el coprocesador no tiene instrucciones de salto, se deberá transferir la palabra de estado a la memoria para que los indicadores puedan ser utilizados con las instrucciones del 8086/8088. Una forma sencilla de usar la palabra de estado con saltos condicionales consiste en mover su byte más significativo en el menos significativo de los indicadores de la CPU. Por ejemplo, se pueden usar las siguientes instrucciones:

```
fstsw mem16    ; Guardar palabra de estado en memoria.
fwait          ; Asegurarse que el coprocesador lo hizo.
mov ax,mem16 ; Mover a AX.
sahf           ; Almacenar byte más significativo en flags.
```

Palabra de estado		C3				C2	C1	C0	(Bits 15-8)
Indicadores 8088	SF	ZF	X	AF	X	PF	X	CF	(Bits 7-0)

De esta manera, C3 se escribe sobre el indicador de cero, C2 sobre el de paridad, C0 sobre el de arrastre y C1 sobre un bit indefinido, por lo que no puede utilizarse directamente con saltos condicionales, aunque se puede utilizar la instrucción TEST para verificar el estado del bit C1 en la memoria o en un registro de la CPU.

Las instrucciones de comparación afectan los indicadores C3, C2 y C0, como se puede ver a continuación.

Después de FCOM	**Después de FTEST**	**C3**	**C2**	**C0**	**Usar**
ST > fuente	ST es positivo	0	0	0	JA
ST < fuente	ST es negativo	0	0	1	JB
ST = fuente	ST es cero	1	0	0	JE
No comparables	ST es NAN o infinito proyectivo	1	1	1	JP

FCOM: Compara ST y ST(1).

FCOM *ST(num)*: Compara ST y ST(num).

FCOM *mem*: Compara ST y mem. El operando de memoria deberá ser un número real de 4 u 8 bytes (no de 10).

FICOM *mem*: Compara ST y mem. El operando deberá ser un número entero de 2 ó 4 bytes (no de 8).

FTST: Compara ST y cero.

FCOMP: Compara ST y ST(1) y extrae ST fuera de la pila.

FCOMP *ST(num)*: Compara ST y ST(num) y extrae ST fuera de la pila.

FCOMP *mem*: Compara ST y mem y extrae ST fuera de la pila. El operando de memoria deberá ser un número real de 4 u 8 bytes (no de 10).

FICOMP *mem*: Compara ST y mem y extrae ST fuera de la pila. El operando deberá ser un número entero de 2 ó 4 bytes (no de 8).

FCOMPP: Compara ST y ST(1) y extrae dos elementos de la pila, perdiéndose ambos operandos.

FXAM: Pone el valor de los indicadores según el tipo de número en ST. La instrucción se utiliza para identificar y manejar valores especiales como infinito, cero, números no normalizados, NAN (Not a Number), etc. Ciertas operaciones matemáticas son capaces de producir estos números especiales. Una descripción de ellos va más allá del alcance de este apunte.

Instrucciones trascendentales

F2XM1: Realiza ST <- 2 ^ ST - 1. El valor previo de ST debe estar entre 0 y 0,5.

FYL2X: Calcula ST(1) * log(2) ST. El puntero de pila se actualiza y luego se deja el resultado en ST, por lo que ambos operandos se destruyen.

FYL2XP1: Calcula ST(1) * log(2) (ST + 1). El puntero de pila se actualiza y luego se deja el resultado en ST, por lo que ambos operandos se destruyen. El valor absoluto del valor previo de ST debe estar entre 0 y la raíz cuadrada de 2 dividido 2. **FPTAN**: Calcula la tangente del valor en ST. El resultado es una razón Y/X, donde X reemplaza el valor anterior de ST y Y se introduce en la pila así que, después de la instrucción, ST contiene Y y ST(1) contiene X. El valor previo de ST debe ser un número positivo menor que pi/4. El resultado de esta instrucción se puede utilizar para calcular otras funciones trigonométricas, incluyendo seno y coseno.

FPATAN: Calcula el arcotangente de la razón Y/X, donde X está en ST e Y está en ST(1). ST es extraído de la pila, dejando el resultado en ST, por lo que ambos operandos se destruyen. El valor de Y debe ser menor que el de X y ambos deben ser positivos. El resultado de esta instrucción se puede usar para calcular otras funciones trigonométricas inversas, incluyendo arcoseno y arcocoseno.

Control del coprocesador

F[N]INIT: Inicializa el coprocesador y restaura todas las condiciones iniciales en las palabras de control y de estado. Es una buena idea utilizar esta instrucción al principio y al final del programa. Poniéndolo al principio asegura que los valores en los registros puestos por programas que se ejecutaron antes no afecten al programa que se comienza a ejecutar. Poniéndolo al final hará que no se afecten los programas que corran después.

F[N]CLEX: Pone a cero los indicadores de excepción y el indicador de ocupado de la palabra de estado. También limpia el indicador de pedido de interrupción del 8087.

FINCSTP: Suma uno al puntero de pila en la palabra de estado. No se afecta ningún registro.

FDECSTP: Resta uno al puntero de pila en la palabra de estado. No se afecta ningún registro.

FREE ST(num): Marca el registro especificado como vacío.

FNOP: Copia ST a sí mismo tomando tiempo de procesamiento sin tener ningún efecto en registros o memoria.

Los microprocesadores 80186 y 80188

Estos microprocesadores altamente integrados aparecieron en 1982. Por "altamente integrados" se entiende que el chip contiene otros componentes aparte de los encontrados en microprocesadores comunes como el 8088 u 8086. Generalmente contienen, aparte de la unidad de ejecución, contadores o "timers", y a veces incluyen memoria RAM y/o ROM y otros dispositivos que varían según los modelos. Cuando contienen memoria ROM, a estos chips se los llama *microcomputadoras en un sólo chip* (no siendo éste el caso de los microprocesadores 80186/80188).

Externamente se encapsulaban en el formato PGA (*Pin Grid Array*) de 68 pines.

Los microprocesadores 80188/80186 contenían, en su primera versión, lo siguiente:

Generador de reloj

El 80186/80188 contiene un oscilador interno de reloj, que requiere un cristal externo o una fuente de frecuencia con niveles TTL. La salida de reloj del sistema tiene una frecuencia de 8 MHz con 50% de ciclo de trabajo (la mitad del tiempo en estado alto y la otra mitad en estado bajo) a la mitad de frecuencia de oscilación del cristal (que debe ser de 16 MHz). Esta salida puede utilizarse para atacar las entradas de reloj (*clock*) de otros componentes, haciendo innecesario tener un chip externo dedicado a la generación de reloj.

Temporizadores

En estos microprocesadores se incluyen dos temporizadores / contadores programables para contar o medir tiempos de eventos externos y para generar formas de onda no repetitivas. El tercero, que no está conectado al exterior, es útil para implementar demoras y como un *prescaler* (divisor) para los otros dos que están conectados exteriormente. Estos temporizadores son muy flexibles y pueden configurarse para contar y medir tiempos de una variedad de actividades de entrada/salida.

Cada uno de los tres temporizadores está equipado con un registro contador de 16 bits que contiene el valor actual del contador/temporizador. Puede ser leído o escrito en cualquier momento (aunque el temporizador esté corriendo). Además cada temporizador posee otro registro de 16 bits que contiene el máximo valor que alcanzará la cuenta. Cada uno de los dos temporizadores conectados exteriormente posee otro registro de cuenta de 16 bits que permite alternar la cuenta entre dos valores máximos de cuenta (lo que sirve para generar señales con ciclo de trabajo diferente del 50%) programables por el usuario. Cuando se alcanza la cuenta máxima, se genera una interrupción y el registro que lleva la cuenta (el primero mencionado) se pone a cero.

Los temporizadores tienen modos de operación bastante flexibles. Todos pueden programarse para parar o poner la cuenta a cero y seguir corriendo cuando llegan al valor máximo. Los dos temporizadores conectados externamente pueden seleccionar entre el reloj interno (basado en la señal generada por el generador de reloj, explicado en el apartado anterior) y externo, alternar entre dos cuentas máximas (primero se usa una y después la otra) o usar una cuenta máxima (siempre el mismo valor), y pueden programarse para volver a disparar cuando ocurre un evento externo.

Canales de DMA

La unidad controladora de DMA (*Direct Memory Access*, lo que indica que no se utiliza la CPU para

realizar la transferencia) integrada en el 80186/80188 contiene dos canales independientes de DMA de alta velocidad. Las transferencias de DMA pueden ocurrir entre los espacios de memoria y la de entrada/salida (M - I/O) o entre el mismo espacio (M - M, I/O - I/O), lo que permite que los dispositivos de entrada/salida y los buffers de memoria puedan ubicarse en cualquiera de los espacios. Cada canal de DMA posee punteros fuente y destino de 20 bits que pueden ser incrementados, decrementados o sin cambiar después de cada transferencia (el último caso es útil para I/O). El usuario puede especificar diferentes modos de operación de DMA utilizando el registro de control de 16 bits.

Controlador de interrupciones

Este controlador resuelve las prioridades entre pedidos de interrupción que arriban simultáneamente. Puede aceptar interrupciones de hasta cinco fuentes externas (una no enmascarable (NMI) y cuatro enmascarables) y de fuentes internas (temporizadores y canales de DMA). Cada fuente de interrupción tiene un nivel de prioridad programable y un vector de interrupción predefinido. El hecho de que el tipo de vector (ver discusión sobre esto en el apartado "Estructura de interrupciones" del microprocesador 8086/8088) sea fijo incrementa la velocidad de respuesta a interrupciones en un 50%. Además tiene varios de los modos de operación del circuito integrado controlador de interrupciones 8259A.

Generación de *Chip Select* y *Ready*

El microprocesador 80186/80188 contiene una lógica de selección de chip programable para proveer señales de *chip select* para memorias y periféricos y también posee una lógica programable de generación de estados de espera (*wait state*) para componentes lentos. El resultado de esta lógica es una menor cantidad de circuitos integrados externos ya que se pueden ahorrar alrededor de diez chips TTL. Aparte del menor costo que esto significa, el rendimiento del sistema aumenta como resultado de la eliminación de demoras de propagación externas (las demoras de las señales en el interior de un chip son significativamente menores que las demoras en el exterior). Otra ventaja se refiere a la flexibilidad en la elección del tamaño y velocidad de acceso de las memorias. Pueden programarse tres rangos de memoria (menor, medio y mayor) con longitudes variables (1K, 2K, 4K, ..., 256K). Pueden programarse entre cero y tres estados de espera para poder utilizar memorias de alta velocidad o memorias de bajo costo (y más lentas). Con respecto a la selección de periféricos, pueden direccionarse hasta siete que pueden estar en la zona de memoria y/o de entrada/salida. También pueden programarse los estados de espera para los periféricos.

Unidad Central de Proceso (CPU) del 80186/80188

La funcionalidad agregada del 80186/80188 (temporizadores, DMA, controlador de interrupciones y selección de chip) utiliza registros de control de 16 bits por cada dispositivo integrado. Estos están contenidos en un bloque de control de 256 bytes incluido en la arquitectura de registros del 80186/80188. Este bloque de control puede estar en la zona de memoria o en la de entrada/salida, basado en la inicialización de un registro especial de reubicación. Exceptuando estos agregados, el resto de los registros son los mismos que los del 8086/8088.

Nuevas instrucciones del 80186/80188

El conjunto de instrucciones está ampliado con respecto al del 8086/8088. Las nuevas instrucciones son:

PUSHA: Almacena los registros de uso general en la pila, en el siguiente orden: AX, CX, DX, BX, SP, BP, SI, DI.

POPA: Extrae los registros de uso general de la pila, retirándolos en el sentido inverso a **PUSHA** (pero descarta la imagen de SP).

PUSH *inmed*: Ingresa un valor inmediato a la pila.

INSB: Operación: ES:[DI] <- Port DX (Un byte), DI<-DI+1 (si DF=0) o DI<-DI-1 (si DF=1).

INSW: Operación: ES:[DI] <- Port DX (Dos bytes), DI<-DI+2 (si DF=0) o DI<-DI-2 (si DF=1).

OUTSB: Operación: Port DX <- DS:[SI] (Un byte), SI<-SI+1 (si DF=0) o SI<-SI-1 (si DF=1).

OUTSW: Operación: Port DX <- DS:[SI] (Dos bytes), SI<-SI+2 (si DF=0) o SI<:-SI-2 (si DF=1).

Shift dest,inmed. Se puede especificar directamente (sin cargar primero el valor en el registro CL) la cantidad de bits del desplazamiento. *Shift* es una de las siguientes instrucciones: **ROL**, **ROR**, **RCL**, **RCR**, **SHL**, **SAL**, **SHR**, **SAR**.

IMUL *reg16,inmed* realiza reg16 <- reg16 * inmed

IMUL *reg16,mem16,inmed* reg16 <- mem16 * inmed

En los dos últimos casos el resultado debe entrar en 16 bits. Si se desea el resultado de 32 bits, debe utilizarse la versión que aparece en el conjunto de instrucciones del 8086/8088.

BOUND *reg16,mem32*. Verifica que el valor contenido en el registro se encuentre entre los dos valores indicados en la memoria (un valor está dado por los dos primeros bytes, y el otro por los dos últimos). Si está fuera de rango se ejecuta una interrupción interna de tipo 5. De esta manera se puede observar que **BOUND** es una instrucción de interrupción condicional, como **INTO**.

ENTER *local_variables_size*, *nesting_level* y **LEAVE** son instrucciones que sirven para facilitar a los compiladores de alto nivel la codificación de subrutinas o procedimientos. Para ello utilizan la pila para almacenar los parámetros y las variables locales. Estos valores se acceden mediante direccionamiento indirecto usando el registro BP. Al principio de la subrutina se deberá indicar, mediante la instrucción **ENTER**, el tamaño total (en bytes) de las variables locales de la subrutina (*local_variables_size*) y cuántos punteros a variables locales (estos se accederán usando [BP-*xxxx*] donde *xxxx* es la posición relativa de la variable local) y parámetros (los valores que se almacenaban en el registro BP) de subrutinas de nivel superior se necesitan ver (*nesting_level*) (en general, este valor debe ser cero) (en [BP] está almacenado el puntero a las variables locales y parámetros de la subrutina que llamó a la actual, en [BP+2] se obtienen los de la subrutina que llamó a la anterior (esto sólo si *nesting_level* > 0), y así sucesivamente. Al final de la subrutina, antes de la instrucción **RET** deberá haber una instrucción **LEAVE**. Cuando se usan estas instrucciones, el programa no debe manejar el registro BP.

Hay dos nuevas interrupciones internas en este microprocesador que se agregan a los del 8086/8088.

- Tipo 5 (BOUND): Ocurre cuando en la instrucción **BOUND** el registro está fuera de rango.

- Tipo 6 (Código de operación inválido): En procesadores anteriores, al ejecutar instrucciones no definidas en el manual del circuito integrado, el resultado es impredecible. En este microprocesador y los

siguientes se ejecuta esta interrupción cuando el código de operación no corresponde a ninguna instrucción.

El 80186 y 80188 tienen las mismas capacidades, excepto que el 80186 trabaja con un bus de datos externo de 16 bits, mientras que el 80188 opera con ocho. Ambos procesadores operan con un bus de datos interno de 16 bits y generan un bus de direcciones de 20 bits para poder acceder a 2^{20} = 1.048.576 bytes (1 MB).

Con lo que se pudo observar, es obvio que estos microprocesadores pueden reemplazar unos 30 circuitos integrados convencionales, con la consiguiente reducción de espacio físico, precio, y requerimientos de potencia, lo que permite utilizar fuentes de alimentación más sencillas, con el consiguiente beneficio económico. De esta manera, es posible encontrarlos en aplicaciones industriales y en algunas plaquetas que se conectan a las PC.

Con el avance de la tecnología CMOS, era necesario un nuevo 80186. Por ello en 1987 apareció la segunda generación de la familia 80186/80188: los microprocesadores 80C186 y 80C188. El 80C186 es compatible pin a pin con el 80186, y agrega nuevas características: una unidad para preservar energía (disminuye el consumo del microprocesador cuando no se necesita utilizar todos los recursos que brinda), una unidad de control de refresco de memorias RAM dinámicas y una interfaz directa al coprocesador matemático 80C187 (esto último no existe en el 80C188). La tecnología CHMOS III utilizada (la misma que para el 80386) permite que el 80C186 corra al doble de velocidad que el 80186 (con el proceso de fabricación HMOS).

En 1990 Intel puso en el mercado el 80C186Ex, con diseño de 1 micrón y velocidad de 25 MHz. Existen actualmente tres modelos: 80C186EA, 80C186EB y 80C186EC. El anterior 80C186 pasó a llamarse 80C186XL.

- El 80C186EA/80C188EA posee manejo de energía (*Power Management*) para reducir la cantidad de energía consumida por el chip cuando no se lo utiliza, operación a 3 volt, utilizando los chips 80L186EA/80L188EA y los mismos periféricos que el 80C186XL. Estos chips se encuentran en el formato de 68 pines PLCC (*Plastic Leaded Chip Carrier*) o bien en 80 pines QFP (*Quad Flat Pack*) o SQFP (*Shrink Quad Flat Pack*).

- El 80C186EB/80C188EB posee dos canales serie (que funcionan tanto para comunicaciones síncronas como asíncronas), dos puertos de entrada/salida multiplexados, la sección de *chip select* está mejorada, posee manejo de energía, operación a 3 volt o 3,3 volt usando el 80L186EB/80L188EB. Estos chips se encuentran en el formato de 84 patas PLCC y en 80 patas QFP o SQFP.

- El 80C186EC/80C188EC posee 4 canales de DMA y dos serie, temporizador watchdog (sirve para reinicializar el microprocesador cuando el programa se "cuelga"), 22 patas de entrada/salida, manejo de energía, operación a 3 volt o 3,3 volt usando el 80L186EC/80L188EC y el resto de los dispositivos del XL. Estos chips se encuentran en el formato de 100 patas, tanto PQFP (*Plastic Quad Flat Pack*), como QFP y SQFP.

El coprocesador matemático 80C187

El 80C187 es un coprocesador relativamente nuevo diseñado para soportar el microprocesador 80C186 (el 80188 no soporta ninguna clase de coprocesadores). Se introdujo en 1989 e implementa el conjunto de instrucciones del 80387.

Está disponible en el formato **CERDIP** (*CERamic Dual Inline Package*) de 40 pines y **PLCC** (*Plastic Leaded Chip Carrier*) de 44. La máxima frecuencia es 16 MHz. A dicha frecuencia el consumo máximo es de 780 mW.

El microprocesador 80286

Introducción

Este microprocesador apareció en febrero de 1982. Los avances de integración que permitieron agregar una gran cantidad de componentes periféricos en el interior del 80186/80188, se utilizaron en el 80286 para hacer un microprocesador que soporte nuevas capacidades, como la multitarea (ejecución simultánea de varios programas), lo que requiere que los programas no "choquen" entre sí, alterando uno los datos o las instrucciones de otros programas. El 80286 tiene dos modos de operación: modo real y modo protegido. En el modo real, se comporta igual que un 8086, mientras que en modo protegido, las cosas cambian completamente, como se explica a partir del próximo párrafo. Esto necesitó un nivel de integración mucho mayor. El 80286 contiene 134.000 transistores dentro de su estructura (360% más que el 8086). Externamente está encapsulado en formato PLCC (*Plastic Leaded Chip Carrier*) con pines en forma de *J* para montaje superficial, o en formato PGA (Pin Grid Array), en ambos casos con 68 pines.

El microprocesador 80286 ha añadido un nuevo nivel de satisfacción a la arquitectura básica del 8086, incluyendo una gestión de memoria con la extensión natural de las capacidades de direccionamiento del procesador. El 80286 tiene elaboradas facilidades incorporadas de protección de datos. Otras características incluyen todas las características del juego de instrucciones del 80186, así como la extensión del espacio direccionable a 16 MB, utilizando 24 bits para direccionar (2^{24} = 16.777.216).

El 80286 revisa cada acceso a instrucciones o datos para comprobar si puede haber una *violación de los derechos de acceso*. Este microprocesador está diseñado para usar un sistema operativo con varios niveles de privilegio. En este tipo de sistemas operativos hay un núcleo que, como su nombre indica, es la parte más interna del sistema operativo. El núcleo tiene el máximo privilegio y los programas de aplicaciones el mínimo. Existen cuatro niveles de privilegio. La protección de datos en este tipo de sistemas se lleva a cabo teniendo segmentos de código (que incluye las instrucciones), datos (que incluye la pila aparte de las variables de los programas) y del sistema (que indican los derechos de acceso de los otros segmentos).

Para un usuario normal, los registros de segmentación (CS, DS, ES, SS) parecen tener los 16 bits usuales. Sin embargo, estos registros no apuntan directamente a memoria, como lo hacían en el 8086. En su lugar, apuntan a tablas especiales, llamadas tablas de descriptores, algunas de las cuales tienen que ver con el usuario y otras con el sistema operativo. Actualmente a los 16 bits, cada registro de segmento del 80286 mantiene otros 57 bits invisibles para el usuario. Ocho de estos bits sirven para mantener los *derechos de acceso* (sólo lectura, sólo escritura y otros), otros bits mantienen la dirección real (24 bits) del principio del segmento y otros mantienen la longitud permitida del segmento (16 bits, para tener la longitud máxima de 64 KB). Por ello, el usuario nunca sabe en qué posición real de memoria está ejecutando o dónde se ubican los datos y siempre se mantiene dentro de ciertas fronteras. Como protección adicional, nunca se permite que el usuario escriba en el segmento de código (en modo real se puede escribir sobre dicho segmento). Ello previene que el usuario modifique su programa para realizar actos ilegales y potencialmente peligrosos. Hay también provisiones para prever que el usuario introduzca en el sistema un "caballo de Troya" que pueda proporcionarle un estado de alto privilegio.

El 80286 tiene cuatro nuevos registros. Tres de ellos apuntan a las tablas de descriptores actualmente en uso. Estas tablas contienen información sobre los objetos protegidos en el sistema. Cualquier cambio de privilegio o de segmento debe realizarse a través de dichas tablas. Adicionalmente hay varios indicadores nuevos.

Existen varias instrucciones nuevas, además de las introducidas con el 80186. Todas estas instrucciones se refieren a la gestión de memoria y protección del sistema haciendo cosas tales como cargar y almacenar el contenido de los indicadores especiales y los punteros a las tablas de descriptores.

Modo protegido del 80286

Debido a la complejidad del siguiente texto, para entenderlo es necesario leerlo detenidamente. Además es necesaria una práctica de estos temas con una PC para fijar los conocimientos.

Mecanismo de direccionamiento

Como en modo real, en modo protegido se utilizan dos componentes para formar la dirección física: un selector de 16 bits se utiliza para determinar la dirección física inicial del segmento, a la cual se suma una dirección efectiva (*offset*) de 16 bits.

La diferencia entre los dos modos radica en el cálculo de la dirección inicial del segmento. En modo protegido el selector se utiliza para especificar un índice en una tabla definida por el sistema operativo. La tabla contiene la dirección base de 24 bits de un segmento dado. La dirección física se obtiene sumando la dirección base hallada en la tabla con el offset.

Segmentación

La segmentación es un método de manejo de memoria. La segmentación provee la base para la protección. Los segmentos se utilizan para encapsular regiones de memoria que tienen atributos comunes. Por ejemplo, todo el código de un programa dado podría estar contenido en un segmento, o una tabla del sistema operativo podría estar en un segmento. Toda la información sobre un segmento se almacena en una estructura de ocho bytes llamada descriptor. Todos los descriptores del sistema están en tablas en memoria que reconoce el hardware.

Terminología

Los siguientes términos se utilizan en la discusión de descriptores, niveles de privilegio y protección.

- **PL** (*Privilege Level*): Uno de los cuatro niveles jerárquicos de privilegio. El nivel cero es el más privilegiado y el tres es el menos privilegiado. Los niveles más privilegiados son numéricamente menores que los menos privilegiados. También se llama **anillo** (*ring*).
  Al nivel cero se lo llama **anillo privilegiado** (*privileged ring*) mientras que los otros son los **anillos no privilegiados** (*non-privileged ring*).

- **RPL** (*Requestor Privilege Level*): Es el nivel de privilegio (PL) de la tarea que generó inicialmente este selector. Se determina leyendo los dos bits menos significativos del selector.

- **DPL** (*Descriptor Privilege Level*): Este es el nivel menos privilegiado para el cual una tarea puede acceder ese descriptor (y el segmento asociado con ese descriptor). El nivel de privilegio del descriptor se determina mediante los bits 6 y 5 en el byte de derechos de acceso del descriptor.

- **CPL** (*Current Privilege Level*): Es el nivel de privilegio en el cual una tarea está ejecutando actualmente. Este valor es igual al nivel de privilegio del segmento de código siendo ejecutado. El

CPL también puede se puede determinar leyendo los dos bits menos significativos del registro CS.

- **EPL** (*Effective Privilege Level*): Es el nivel menos privilegiado del RPL y DPL. Debido a que menores valores de privilegio indican mayor privilegio, el **EPL** es el máximo de RPL y DPL.
- **Tarea** (*task*): Una instancia en la ejecución de un programa. También se lo llama proceso.

- **Selector**: Entidad de dos bytes que apunta a un descriptor. Contiene tres campos:

Bit	15 ... 3	2	1 0
Campos	Índice	TI	RPL

  El índice selecciona uno de entre $2^{13} = 8192$ descriptores.
  El bit indicador de tabla (TI) especifica la tabla de descriptores a utilizar: si vale cero se utiliza la tabla de descriptores globales, mientras que si vale uno, se utiliza la tabla de descriptores locales.

- **Descriptor**: Estructura de ocho bytes que le indica al procesador el tamaño y ubicación de un segmento, así como información de control y estado. Los **descriptores** son creados por compiladores, linkers, programas cargadores, o el sistema operativo, pero nunca por los programas de aplicación. Si bien el tamaño de esta estructura es siempre de ocho bytes, el formato depende de la tabla de descriptores.

Tablas de descriptores

Estas tablas definen todos los segmentos utilizados en un sistema basado en el 80286. Hay tres tipos de tablas que mantienen descriptores: la tabla de descriptores globales o GDT (*Global Descriptor Table*), la tabla de descriptores locales o LDT (*Local Descriptor Table*) y la tabla de descriptores de interrupción o IDT (*Interrupt Descriptor Table*). Todas las tablas son arrays de longitud variable, que pueden tener entre 8 y 65.536 bytes. Cada tabla puede mantener hasta 8192 descriptores. Los 13 bits más significativos de un selector se usan como un índice dentro de la tabla de descriptores. Las tablas tienen registros asociados que contienen la dirección base de 24 bits y el límite de 16 bits de cada tabla.

Cada una de las tablas tiene un registro asociado. Estos se llaman GDTR, LDTR, IDTR (ver las siglas en inglés que aparecen en el párrafo anterior). Las instrucciones LGDT, LLDT y LIDT cargan (*Load*) la base y el límite de la tabla de descriptores globales, locales o de interrupción, respectivamente, en el registro apropiado. Las instrucciones SGDT, SLDT y SIDT almacenan (*Store*) los valores anteriormente mencionados de los registros en memoria. Estas tablas son manipuladas por el sistema operativo, por lo que las instrucciones de carga son instrucciones privilegiadas (sólo se ejecutan si el CPL vale cero).

Tabla de descriptores globales (GDT)

Esta tabla contiene descriptores que están disponibles para todas las tareas del sistema. La **GDT** puede contener cualquier clase de descriptores de segmento excepto los relacionados con interrupciones. Todos los sistemas basados en el 80286 en modo protegido contienen una **GDT**. Generalmente la **GDT** contiene los segmentos de código y datos usados por el sistema operativo y los TSS (cuya explicación aparece más adelante) y los descriptores para las LDT en un sistema.
La primera entrada de la **GDT** corresponde al selector nulo y no se usa.

Tabla de descriptores locales (LDT)

Las **LDT** contienen descriptores asociados con una tarea particular. Generalmente los sistemas operativos se diseñan de forma tal que cada tarea tenga su propia **LDT**. La **LDT** sólo puede contener descriptores de código, datos, pila, compuertas de tarea, y compuertas de llamada. Las **LDT** proveen un mecanismo para aislar los segmentos de código y datos de una tarea dada del sistema operativo y las otras tareas, mientras que la GDT contiene descriptores comunes a todas las tareas. Una tarea no puede acceder un segmento si no existe en la **LDT** actual o en la GDT. Esto permite el aislamiento y protección de los segmentos de una tarea, mientras que los datos comunes pueden ser compartidos por todas las tareas.

Tabla de descriptores de interrupción

La tercera tabla necesaria para sistemas 80286 que operan en modo protegido es la **IDT**. Esta tabla contiene los descriptores que apuntan a la ubicación de hasta 256 rutinas de servicio de interrupción (interrupt handlers). Debe conocerse cuál es el valor máximo del tipo de interrupción que se va a utilizar y poner el límite del IDTR igual a ocho veces ese valor (ya que, como se explicó anteriormente, cada descriptor ocupa ocho bytes). Cada interrupción utilizada por el sistema debe tener una entrada propia en la **IDT**. Las entradas de la **IDT** se referencian mediante instrucciones INT, vectores externos de interrupción y excepciones (interrupciones internas del microprocesador).

Registros de direcciones del sistema

Como se indicó anteriormente, existen cuatro registros del 80286 que apuntan a las tablas de descriptores.

GDTR e **IDTR**: Estos registros mantienen la dirección base de 24 bits y el límite de 16 bits de las tablas GDT e IDT, respectivamente. Estos segmentos, como son globales para todas las tareas en el sistema, se definen mediante direcciones físicas de 24 bits y límite de 16 bits.

LDTR y **TR**: Estos registros mantienen los selectores de 16 bits para el descriptor de LDT y de TSS, respectivamente. Estos segmentos, como son específicos para cada tarea, se definen mediante valores de selector almacenado en los registros de segmento del sistema. Éste apunta a un descriptor apropiado (de LDT o TSS). Nótese que un registro descriptor del segmento (invisible para el programador) está asociado con cada registro de segmento del sistema.

Descriptores

A continuación se brinda una descripción detallada de los contenidos de los descriptores utilizados en el microprocesador 80286 operando en modo protegido.

Bits de atributo del descriptor

El objeto apuntado por el selector se llama descriptor. Los descriptores son cantidades de ocho bytes que contienen atributos sobre una región de memoria (es decir, un segmento). Estos atributos incluyen la dirección base de 24 bits, la longitud de 16 bits, el nivel de protección (de 0 a 3), permisos de lectura, escritura o ejecución (esto último para segmentos de código), y el tipo de segmento. A continuación se muestra el formato general de un descriptor.

- **Byte 0**: Límite del segmento (bits 7-0).
- **Byte 1**: Límite del segmento (bits 15-8).
- **Byte 2**: Dirección base del segmento (bits 7-0).

- **Byte 3**: Dirección base del segmento (bits 15-8).
- **Byte 4**: Dirección base del segmento (bits 23-16).
- **Byte 5**: Derechos de acceso del segmento.
- **Bytes 6 y 7**: No utilizados (deben valer cero para lograr la compatibilidad con 80386 y posteriores).

El **byte de derechos de acceso** es el que define qué clase de descriptor es. El bit 4 (S) indica si el segmento es de código o datos (S = 1), o si es del sistema (S = 0). Veremos el primer caso.

- Bit 7: **Presente (P)**. Si P = 1 el segmento existe en memoria física, mientras que si P = 0 el segmento no está en memoria. Un intento de acceder un segmento que no está **presente** cargando un registro de segmento (CS, DS, ES o SS) con un selector que apunte a un descriptor que indique que el segmento no está **presente** generará una excepción 11 (en el caso de SS se generará una excepción 12 indicando que la pila no está **presente**). El manejador de la interrupción 11 deberá leer el segmento del disco rígido. Esto sirve para implementar memoria virtual.

- Bits 6 y 5: **Nivel de privilegio del descriptor (DPL)**: Atributo de privilegio del segmento utilizado en tests de privilegio.

- Bit 4: **Bit descriptor del segmento (S)**: Como se explicó más arriba, este bit vale 1 para descriptores de código y datos.

- Bit 3: **Ejecutable (E)**: Determina si el segmento es de código (E = 1), o de datos (E = 0).
  Si el bit 3 vale cero:
  - Bit 2: **Dirección de expansión (ED)**: Si E = 0, el segmento se expande hacia arriba, con lo que los offsets deben ser menores o iguales que el límite. Si E = 1, el segmento se expande hacia abajo, con lo que los offsets deben ser mayores que el límite. Si no ocurre esto se genera una excepción 13 (Fallo general de protección) (en el caso de la pila se genera una excepción 12).
  - Bit 1: **Habilitación de escritura (W)**: Si W = 0 no se puede escribir sobre el segmento, mientras que si W = 1 se puede realizar la escritura.

  Si el bit 3 vale uno:
  - Bit 2: **Conforme (C)**: Si C = 1, el segmento de código sólo puede ser ejecutado si CPL es mayor que DPL y CPL no cambia. Los segmentos conformes sirven para rutinas del sistema operativo que no requieran protección, tales como rutinas matemáticas, por ejemplo.
  - Bit 1: **Habilitación de lectura (R)**: Si R = 0, el segmento de código no se puede leer, mientras que si R = 1 sí. No se puede escribir sobre el segmento de código.
- Bit 0: **Accedido (A)**: Si A = 0 el segmento no fue accedido, mientras que si A = 1 el selector se ha cargado en un registro de segmento o utilizado por instrucciones de test de selectores. Este bit es puesto a uno por el microprocesador.

Si se lee o escribe en un segmento donde no está permitido o se intenta ejecutar en un segmento de datos se genera una excepción 13 (Violación general de protección). Si bien no se puede escribir sobre un segmento de código, éstos se pueden inicializar o modificar mediante un **alias**. Los **alias** son segmentos de datos con permiso de escritura (E = 0, W = 1) cuyo rango de direcciones coincide con el segmento de código.

Los segmentos de código cuyo bit C vale 1, pueden ejecutarse y compartirse por programas con diferentes niveles de privilegio (ver la sección sobre protección, más adelante).

A continuación se verá el formato del **byte de derechos de acceso** para descriptores de segmentos del sistema:

- Bit 7: **Presente (P)**: Igual que antes.
- Bits 6 y 5: **Nivel de privilegio del descriptor (DPL)**: Igual que antes.
- Bit 4: **Bit descriptor del segmento (S)**: Como se explicó más arriba, este bit vale 0 para descriptores del sistema.

- Bits 3-0: **Tipo de descriptor**: En el 80286 están disponibles los siguientes:

0000: Inválido	0100: Compuerta de llamada
0001: TSS disponible	0101: Compuerta de tarea
0010: LDT	0110: Compuerta de interrupción
0011: TSS ocupado	0111: Compuerta de trampa

  Los otros ocho tipos están reservados para el procesador 80386 y posteriores.

Como se pudo observar hay atributos en común entre los distintos descriptores: **P**, **DPL** y **S**.

Ahora se verá con más detalle los diferentes descriptores del sistema.

- **Descriptores de LDT** (S = 0, tipo = 2): Contienen información sobre tablas de descriptores locales. Las LDT contienen una tabla de descriptores de segmentos, únicos para cada tarea. Como la instrucción para cargar el registro LDTR sólo está disponible en el nivel de privilegio 0 (nivel más privilegiado), el campo DPL es ignorado. Los descriptores de LDT sólo se permiten en la tabla de descriptores globales (GDT).

- **Descriptores de TSS** (S = 0, tipos = 1, 3): Un descriptor de segmento de estado de la tarea (TSS: *Task State Segment*) contiene información sobre la ubicación, el tamaño y el nivel de privilegio de un TSS. Un TSS es un segmento especial con formato fijo que contiene toda la información sobre el estado de una tarea y un campo de enlace para permitir tareas anidadas. El campo de tipo se usa para indicar si la tarea está ocupada (tipo = 3), es decir, en una cadena de tareas activas, o si el TSS está disponible (tipo = 1). El registro de tarea (TR: Task Register) contiene el selector que apunta al TSS actual dentro de la GDT.

- **Descriptores de compuertas** (tipos = 4 a 7): Las compuertas se utilizan para controlar el acceso a puntos de entrada dentro del segmento de código objetivo. Los distintos tipos de compuerta son: de llamada (*call gate*), de tarea (*task gate*), de interrupción (*interrupt gate*) y de trampa (*trap gate*). Las compuertas proveen un nivel de indirección entre la fuente y el destino de la transferencia de control. Esta indirección permite al procesador realizar automáticamente verificaciones de protección. También permite a los diseñadores controlar los puntos de entrada a los sistemas operativos. Las compuertas de llamada se utilizan para cambiar niveles de privilegio (ver la sección que trata sobre privilegio, más adelante), las compuertas de tarea se utilizan para hacer cambios de tarea y las de interrupción y de trampa se utilizan para especificar rutinas de servicio de interrupción.

  A continuación se muestra el formato de los **descriptores de compuertas**, que difieren del formato general:

- Byte 0: Offset (bits 7-0).
- Byte 1: Offset (bits 15-8).
- Byte 2: Selector (bits 7-0).
- Byte 3: Selector (bits 15-0).
- Byte 4: Bits 4-0: Cantidad de palabras, bits 7-5: Cero.
- Byte 5: Derechos de acceso.
- Bytes 6 y 7: No usados (deben valer cero para lograr la compatibilidad con los procesadores 80386 y posteriores).

Las compuertas de llamado se utilizan para transferir el control del programa a un nivel más privilegiado. El descriptor correspondiente consiste en tres campos: el byte de derechos de acceso, un puntero largo (selector y offset) y una cantidad de palabras que especifica cuántos parámetros deben copiarse de la pila de la rutina llamadora a la pila de la rutina llamada. Este campo sólo es utilizado por las compuertas de llamada cuando hay un cambio de nivel de privilegio (PL). Los otros tipos de compuertas ignoran el campo de cantidad de palabras.

Las compuertas de interrupción y de trampa utilizan los campos de selector y offset como un puntero al inicio de la rutina que maneja la interrupción o trampa. La diferencia entre ambos tipos de compuertas es que la de interrupción deshabilita las interrupciones (IF <- 0), mientras que la de trampa no altera IF.

Las compuertas de tarea se usan para cambiar tareas. Las compuertas de tarea sólo se pueden referir a un TSS y por lo tanto sólo se utiliza el campo del selector, siendo ignorado el de offset.

Se genera una excepción 13 (Violación general de protección) si un selector no se refiere a un tipo de descriptor correcto (un segmento de código para una compuerta de interrupción, trampa o llamada y un segmento de estado de tarea para la compuerta de tarea).

Caché del descriptor del segmento

Aparte del valor del selector, cada registro de segmento tiene un registro caché de descriptor del segmento invisible para el programador. Al cargar un registro de segmento con un nuevo selector, el descriptor de ocho bytes asociado con ese selector se carga automáticamente en el chip. Una vez que se hizo esto, todas las referencias a ese segmento utilizan la información almacenada en el caché en vez de volver a acceder el descriptor. Como los cachés de los descriptores sólo varían cuando se carga un registro de segmento, los programas que deben modificar las tablas de descriptores deben volver a cargar los registros de segmento apropiados después de cambiar el valor de un descriptor.

Contenido de los cachés descriptores de segmento: El contenido varía dependiendo del modo en que opera el 80286. En modo protegido, como se explicó más arriba, los cachés se cargan con la información de los descriptores. En modo real, el contenido no se obtiene de la memoria, sino que toma unos valores fijos para lograr compatibilidad con el 8086. La base generada es 16 veces el valor del selector, el límite siempre es FFFFh, el bit P (presente) vale 1, el nivel de privilegio es cero (máximo privilegio), el bit de dirección de expansión indica que los offsets deben ser menores o iguales que el límite y están habilitados los permisos de lectura, escritura y ejecución. Todo esto hace que en modo real no se pueda acceder a los 16 MB de capacidad de direccionamiento del 80286, ya que el valor máximo ocurre cuando el selector vale FFFFh y el offset también, con lo que se logra una dirección máxima de FFFF0h + FFFFh = 10FFEFh (casi 1088 KB).

Protección

El 80286 tiene cuatro niveles de protección que están optimizados para soportar las necesidades de los sistemas operativos multitarea para aislar y proteger los programas de un usuario de otros y del sistema operativo. Los niveles de privilegio controlan el uso de instrucciones privilegiadas, instrucciones de entrada/salida, y el acceso a segmentos y descriptores de segmento. A diferencia de los sistemas tradicionales basados en microprocesadores donde esta protección sólo se logra a través de un hardware externo muy complejo con el correspondiente software, el 80286 provee esta protección como parte de la unidad de manejo de memoria (*MMU: Memory Management Unit*) incorporada.

El sistema de privilegio jerárquico de cuatro niveles es una extensión de los modos usuario/supervisor que se encuentran comúnmente en minicomputadoras. Los niveles de privilegio (*PL: Privilege Level*) se numeran de 0 a 3, siendo el 0 el nivel más privilegiado (más confiable).

Ejemplo de los niveles jerárquicos:

- **PL = 0**: Kernel (parte del sistema operativo).
- **PL = 1**: Servicios del sistema (parte del SO).
- **PL = 2**: Extensiones del sistema operativo.
- **PL = 3**: Aplicaciones.

Reglas de privilegio

El 80286 controla el acceso a los datos y procedimientos (o subrutinas) entre niveles de una tarea, de acuerdo a las siguientes reglas:

- Los datos almacenados en un segmento con nivel de privilegio P pueden ser accedidos solamente por código que se ejecuta a un nivel de privilegio por lo menos tan privilegiado como P.
- Un segmento de código/procedimiento con nivel de privilegio P sólo puede ser llamado por una tarea que se ejecuta al mismo o menor privilegio que P.

Privilegio de una tarea

En un momento determinado, una tarea en el 80286 se ejecuta en uno de los cuatro niveles de privilegio. Esto está especificado por el nivel de privilegio actual (**CPL**). El CPL de una tarea sólo puede cambiarse mediante transferencias de control utilizando los descriptores de compuerta a un segmento de código con un nivel de privilegio diferente. Por ejemplo, un programa de aplicación corriendo con PL = 3 puede llamar una rutina del sistema operativo con PL = 1 (mediante una compuerta) que causaría que CPL = 1 hasta que finalice la rutina del sistema operativo.

Nivel de privilegio del selector (RPL)

El nivel de privilegio de un selector se especifica en el campo RPL. El RPL está formado por los dos bits menos significativos del selector. Se utiliza para que un nivel de privilegio menos confiable que el nivel de privilegio actual (CPL) pueda utilizar dicho selector. Este nivel se llama nivel de privilegio efectivo (EPL) de la tarea, que se define como el nivel menos privilegiado (numéricamente mayor) del CPL de la tarea y el RPL del selector. Así, si RPL = 0, entonces CPL siempre especifica el nivel de privilegio para realizar un acceso usando el selector, mientras que si RPL = 3, entonces el selector sólo puede acceder segmentos

de nivel 3 independientemente del valor de CPL de la tarea. El RPL se utiliza generalmente para verificar que los punteros pasados a un procedimiento del sistema operativo no accede datos que son de mayor privilegio que el procedimiento que originó el puntero. Como dicho procedimiento puede especificar cualquier valor de RPL, la instrucción de ajuste de RPL (ARPL) se utiliza para forzar los bits de RPL a que sean iguales al CPL de la rutina que generó el selector.

Privilegio de entrada/salida

El nivel de privilegio de entrada/salida (**IOPL**, que ocupa los bits 13 y 12 del registro de indicadores) define el nivel menos privilegiado para el cual se pueden realizar instrucciones de I/O (**IN**, **OUT**, **INS**, **OUTS**, **REP INS**, **REP OUTS**). Si CPL > **IOPL**, al ejecutar alguna de estas instrucciones se generará una excepción 13. **IOPL** también afecta otras instrucciones, como **STI**, **CLI** y el prefijo **LOCK**. Además afecta si IF (indicador de interrupciones) puede cambiarse cargando un valor en el registro de indicadores (mediante **POPF**). Si CPL es menor o igual que **IOPL**, entonces IF se puede cambiar. Si CPL > **IOPL** el valor de IF no varía mediante la ejecución de la instrucción **POPF** (en este caso no se genera ninguna excepción).

Validación de privilegio

El 80286 provee algunas instrucciones para acelerar la verificación de punteros y mantener la integridad del sistema comprobando que el valor del selector se refiera a un segmento apropiado.
La verificación de punteros previene el problema común de una aplicación con PL = 3 que llama a un rutina del sistema operativo con PL = 0 pasándole un puntero "erróneo" que corrompe una estructura de datos que pertenece al sistema operativo. Si éste utilizara la instrucción ARPL para asegurar que el RPL del selector no tiene mayor privilegio que la rutina llamadora, este problema podría evitarse.

Las instrucciones para verificar punteros son ARPL, VERR, VERW, LSL y LAR. Todas estas instrucciones ponen el indicador de cero a uno si la verificación se pudo realizar; si no se puede realizar pone ZF <- 0, en vez de generar una excepción 13 como hace con otras instrucciones.

Acceso a descriptores

Básicamente hay dos tipos de accesos de segmentos: aquéllos que se refieren a los segmentos de código como las transferencias de control, y aquéllos que se refieren a segmentos de datos. Para determinar si una tarea puede acceder un segmento se necesita conocer el tipo de segmento a acceder, las instrucciones utilizadas, el tipo de descriptor utilizado y los CPL, RPL y DPL, como se describió más arriba.
Cada vez que una instrucción carga los registros de segmentos de datos (DS, ES), el 80286 hace validaciones de protección. Los selectores almacenados en esos registros sólo pueden referirse a segmentos de datos o segmentos de código con habilitación de lectura. Las reglas de accesos de datos se especificaron anteriormente. La única excepción a estas reglas la constituye el segmento de código legible con el bit C = 1 de los derechos de acceso que se puede acceder desde cualquier nivel.
Finalmente se realizan las verificaciones de privilegio. El CPL se compara con EPL y si EPL es más privilegiado que CPL se genera una excepción 13 (Violación general de protección).
Las reglas para el segmento de pila son ligeramente diferentes que aquéllos referidos a los segmentos de datos. Las instrucciones que cargan los selectores en SS deben referirse a descriptores de segmentos de datos con habilitación de escritura (ya que la pila debe poder ser leída y escrita). El DPL y RPL deben ser iguales al CPL. Cualquier otro tipo de descriptor o una violación de privilegio causará una excepción 13. Si el segmento de pila está marcado como no presente (P = 0) se genera una excepción 12. La excepción 11 ocurre en el caso de segmentos de código o datos no presentes.

Transferencias de nivel de privilegio

Las transferencias de control intersegmento ocurren cuando se carga un selector en el registro CS. Para un sistema típico la mayoría de esas transferencias ocurren simplemente como resultado de una llamada o un salto a otra rutina. Hay cinco tipos de control de transferencia que se resumen a continuación. Varias de estas transferencias resultan en un cambio de nivel de privilegio. El cambio de niveles de privilegio sólo se puede efectuar mediante compuertas y cambios de tarea.

Los tipos de descriptores que se usan para transferencia de control son los siguientes:

Tipos de transferencia de control	**Operaciones que la generan**	**Descriptor que se referencia**	**Tabla de descriptores que se utiliza para realizar la transferencia**
Transferencia de control intersegmento dentro del mismo nivel de privilegio	**JMP** **CALL** **RET** **IRET** (sólo si el flag NT vale 0)	Segmento de código	GDT LDT
Transferencia de control intersegmento al mismo o mayor nivel de privilegio	**CALL**	Compuerta de llamada	GDT LDT
Transferencia de control intersegmento al mismo o mayor nivel de privilegio	**INT** excepción interrupción externa	Compuerta de interrupción o trampa	Tabla de descriptores de interrupción
Transferencia de control intersegmento al mismo o menor nivel de privilegio	**RET** **IRET** (sólo si el flag NT vale 0)	Segmento de código	GDT LDT
Cambio de tarea a través del segmento de estado de tarea	**CALL** **JMP**	Segmento de estado de la tarea	Tabla de descriptores globales
Cambio de tarea a través de una compuerta de tarea	**CALL** **JMP**	Compuerta de tarea	GDT LDT
Cambio de tarea a través de una compuerta de tarea	**IRET** (sólo si el flag NT vale 1)	Compuerta de tarea	Tabla de descriptores de interrupción

Las transferencias de control sólo pueden ocurrir si la operación que cargó el selector se refiere al tipo correcto de descriptor. Cualquier violación de estas reglas causará una excepción 13 (por ejemplo, un salto a través de una compuerta de llamado, o un **IRET** desde una subrutina de interrupción).

Para que el sistema sea aún más seguro, todas las transferencias de control también se sujetan a las reglas de privilegio, que indican lo siguiente:

- Las transiciones de nivel de privilegio sólo ocurren a través de las compuertas.
- Los saltos (**JMP**) pueden realizarse a segmento de código no conformes con el mismo nivel de

privilegio o a segmento de código conformes con mayor o igual privilegio.
- Las llamadas (**CALL**) pueden realizarse a segmentos de código no conformes con el mismo nivel de privilegio o a través de una compuerta a un nivel más privilegiado.
- Las interrupciones que se manejan dentro de una tarea se manejan de la misma manera que las llamadas **CALL**.
- Los segmentos de código conformes son accesibles por niveles de privilegio que tienen el mismo o menor privilegio que el DPL (*Descriptor Privilege Level*) de ese segmento de código.
- El RPL (*Requested Privilege Level*) en el selector que apunta a una compuerta y el CPL (*Current Privilege Level*) de la tarea deben tener igual o mayor privilegio que el DPL (*Descriptor Privilege Level*) de la compuerta.
- El segmento de código seleccionado en la compuerta debe tener el mismo o mayor privilegio que el CPL (*Current Privilege Level*) de la tarea.
- Las instrucciones de retorno que no cambian tareas sólo pueden retornar el control a un segmento de código con el mismo o menor privilegio.
- Los cambios de tarea se realizan mediante instrucciones **CALL**, **JMP** o **INT** que referencian a una compuerta de tarea o un segmento de estado de tarea cuyo DPL (*Descriptor Privilege Level*) tiene el mismo o menos privilegio que el CPL (*Current Privilege Level*) de la vieja tarea.

Todas las transferencias de control que cambian CPL dentro de la misma tarea causan un cambio de pilas como resultado del cambio de privilegio. Los valores iniciales de SS:SP para los niveles de privilegio 0, 1 y 2 se almacenan en el segmento de estado de tarea (ver más abajo el formato de dicho segmento). Durante la ejecución de **JMP** o **CALL**, el nuevo puntero de pila se carga en los registros SS y SP y el puntero previo se pone en la nueva pila.

Al retornar al nivel de privilegio original, el uso de la pila menos privilegiada se restaura como parte de la ejecución de la instrucción **RET** o la **IRET**. Para subrutinas que pasan parámetros en la pila y cruzan niveles de privilegio, se copia un número fijo de palabras (especificado en el campo de cuenta de palabras de la compuerta) de la pila previa a la actual. La instrucción **RET** intersegmento con un valor de ajuste del puntero de pila ajusta correctamente el SP al efectuar el retorno.

Compuertas de llamado: Estas compuertas proveen llamadas (**CALL**) indirectas, en forma protegida. Uno de los usos más importantes de las compuertas es poder tener un método seguro de transferencias de privilegio dentro de una tarea. Como el sistema operativo define todas las compuertas en el sistema, puede asegurar que todas las compuertas permiten el acceso sólo a los procedimientos confiables (como aquéllos que realizan manejo de memoria u operaciones de entrada/salida). Un intento de acceso a otro lugar causará una excepción 13 (Violación general de protección).

Los descriptores de compuertas siguen las mismas reglas de privilegio que los datos, esto es, una tarea puede acceder una compuerta sólo si el EPL (*Effective Privilege Level*) es igual o más privilegiado que el DPL (*Descriptor Privilege Level*). Las compuertas siguen las reglas de privilegio de las transferencias de control y por lo tanto sólo pueden transferir el control a un nivel más privilegiado.

Las compuertas de llamada se acceden mediante una instrucción **CALL** y son sintácticamente idénticas a la llamada a una subrutina normal. Cuando se activa una llamada que cambia el nivel de privilegio, suceden las siguientes acciones:

1) Cargar CS:IP de la compuerta y verificar la validez.
2) Poner el viejo SS en la nueva pila.
3) Poner el viejo SP en la nueva pila.

4) Copiar la cantidad de parámetros de 16 bits como se indica en el campo de cuenta de palabras de la compuerta de la vieja a la nueva pila.
5) Poner la dirección de retorno en la pila.

Las compuertas de interrupción y de trampa trabajan de la misma forma que las compuertas de llamada, excepto que no hay copia de parámetros. La única diferencia entre las compuertas de interrupción y de trampa es que las primeras deshabilitan las interrupciones (IF <- 0), mientras que las otras no alteran el indicador de interrupciones IF.

Cambio de tareas

Un atributo muy importante de cualquier sistema operativo multitarea/multiusuario es su habilidad para cambiar rápidamente entre tareas o procesos. El 80286 soporta esta operación incluyendo una instrucción de **cambio de tarea** en el hardware. Dicha instrucción salva el estado entero de la máquina (todos los registros, espacio de direcciones y un puntero a la tarea anterior), carga un nuevo estado de ejecución, realiza verificaciones de protección y comienza la ejecución en menos de 20 microsegundos. Al igual que la transferencia de control por compuertas, la operación de **cambio de tareas** se invoca ejecutando una instrucción **JMP** o **CALL** intersegmento que se refiere a un segmento de estado de la tarea (TSS) o un descriptor de compuerta de tarea en el GDT (*Global Descriptor Table*) o en el LDT (*Local Descriptor Table*). Una instrucción **INT** *n*, una excepción, trampa o interrupción externa puede también invocar la operación de **cambio de tarea** si hay un descriptor de compuerta de tarea en la entrada correspondiente de la IDT (*Interrupt Descriptor Table*).

Segmento de estado de la tarea

El descriptor **TSS** apunta a un segmento que contiene el estado de la ejecución de 80286 mientras que un descriptor de compuerta de tarea contiene un selector de **TSS**.
El límite (cantidad de bytes que ocupa) del segmento debe ser mayor o igual a 2Ch = 44 bytes. Si es mayor, en el espacio adicional del segmento TSS, el sistema operativo puede almacenar información adicional acerca de las razones por la que la tarea está inactiva, el tiempo que la tarea estuvo corriendo, los archivos abiertos que pertenecen a esa tarea, etc.

El formato del **TSS** es como sigue:

00: Puntero selector del TSS
02: SP para CPL = 0
04: SS para CPL = 0
06: SP para CPL = 1
08: SS para CPL = 1
0A: SP para CPL = 2
0C: SS para CPL = 2
0E: IP (Punto de entrada)
10: Flags
12: Registro AX
14: Registro CX
16: Registro DX
18: Registro BX
1A: Registro SP
1C: Registro BP

1E: Registro SI
20: Registro DI
22: Selector ES
24: Selector CS
26: Selector SS
28: Selector DS
2A: Selector LDT de la tarea
2C: Disponible

Cada tarea debe tener un **TSS** asociado. El **TSS** actual se identifica mediante un registro especial en el 80286 llamado **TR** (*Task State Segment Register*). Este registro contiene un selector que se refiere al descriptor de **TSS** de la tarea. Un registro de base y límite del segmento (invisible para el programador) asociado con **TR** se carga cada vez que **TR** se carga con un nuevo selector. El retorno de una tarea se realiza mediante la instrucción **IRET**. Cuando se ejecuta **IRET**, el control retorna a la tarea que había sido interrumpida. El estado de la tarea que se está ejecutando se almacena en el **TSS** y el estado de la vieja tarea se restaura de su propio **TSS**. Algunos bits en el registro de indicadores y la palabra de estado de la máquina (MSW) dan información sobre el estado de una tarea que son útiles para el sistema operativo. El bit 14 del registro de indicadores (**NT** = *Nested Task*) controla la función de la instrucción **IRET**. Si **NT** = 0, dicha instrucción realiza un retorno normal, pero si **NT** = 1, **IRET** realiza una operación de cambio de tarea para volver a ejecutar la tarea anterior. El bit **NT** se pone a cero o uno de la siguiente manera: cuando una instrucción **CALL** o **INT** inicia un cambio de tarea, el nuevo **TSS** se marca como ocupado y el puntero del nuevo **TSS** se carga con el selector del viejo **TSS**. El bit **NT** de la nueva tarea se pone entonces a 1. Una interrupción que no causa un cambio de tareas pondrá a cero el bit **NT** (el bit **NT** será restaurado a su valor anterior luego de la ejecución del manejador de interrupciones). El bit **NT** también puede ser afectado por las instrucciones **POPF** o **IRET**.

El **segmento de estado de la tarea** (**TSS**) se marca como ocupado cambiando el campo de tipo de descriptor de tipo 1 a tipo 3. El uso de un selector que referencia un **TSS** ocupado resultará en una excepción 13 (Violación general de protección).

El estado del coprocesador no se salva automáticamente cuando ocurre un cambio de tareas, porque la nueva tarea puede no utilizar instrucciones del coprocesador. El bit 3 del **MSW** llamado **TS** (*Task Switched*) ayuda en esta situación. Cuando el microprocesador realiza un cambio de tarea, pone a uno el flag **TS**. El 80286 detecta el primer uso de una instrucción de coprocesador después del cambio de tarea y provoca una excepción 7 (Coprocesador inexistente). El manejador de la excepción puede entonces decidir si debe salvar el estado del coprocesador. Dicha excepción ocurre cuando se trata de ejecutar una instrucción **ESC** o **WAIT** (es decir, alguna referida al coprocesador) y los bits *Task Switched* y *Monitor coprocessor extension* están ambos a uno (es decir, TS = MP = 1).

Inicialización y transición a modo protegido

Como el 80286 comienza la ejecución (después de activar el pin RESET) en modo real, es necesario inicializar las tablas del sistema y los registros con los valores apropiados. Los registros GDTR e IDTR deben referirse a tablas de descriptores globales y de interrupción (respectivamente) que sean válidas.

Para entrar en modo protegido debe ponerse el bit **PE** (bit 0 de **MSW**) a 1 utilizando la instrucción LMSW. Después de entrar en modo protegido, la siguiente instrucción deberá ser un **JMP** intersegmento para cargar el registro CS y liberar la cola de instrucciones. El paso final es cargar todos los registros de segmentos de datos con los valores iniciales de los selectores.

Una forma alternativa de entrar en modo protegido que es especialmente apropiada en sistemas operativos multitarea consiste en realizar un cambio de tarea para cargar todos los registros. En este caso la GDT contendría dos descriptores de TSS además de los descriptores de código y datos necesarios para la primera tarea. La primera instrucción **JMP** en modo protegido saltaría al TSS causando un cambio de tarea y cargando todos los registros con los valores almacenados en el TSS. El registro **TR** (*Task State Segment Register*) deberá apuntar un descriptor de TSS válido ya que un cambio de tarea almacena el estado de la tarea actual en el TSS apuntado por **TR**.

Nuevas instrucciones del 80286

Aparte de las instrucciones del 8086/8088 y las nuevas del 80186, el 80286 posee nuevas instrucciones. Éstas corresponden todas al modo protegido y son las siguientes:

ARPL *dest, src* (*Adjust Requested Privilege Level of selector*): Compara los bits RPL de *dest* contra *src*. Si el RPL de *dest* es menor que el RPL de *src*, los bits RPL del destino se cargan con los bits RPL de *src* y el indicador ZF se pone a uno. En caso contrario ZF se pone a cero. Ver nota 1.

CLTS (*Clear Task Switched Flag*): Pone a cero el indicador **TS** (bit 3 de la palabra de control de la máquina **MSW**). Ver nota 2.

LAR *dest, src* (Load Access Rights): El byte más alto del registro destino se carga con el byte de derechos de acceso del segmento indicado por el selector almacenado en *src*. Pone ZF a uno si se puede realizar la carga. Ver notas 1 y 3.

LGDT *mem64* (*Load Global Table register*): Carga el valor del operando en el registro GDTR. Antes de ejecutar esta instrucción la tabla debe estar en memoria. Ver nota 2.

LIDT *mem64* (*Load Interrupt Table register*): Carga el valor del operando en el registro IDTR. Antes de ejecutar esta instrucción la tabla debe estar en memoria. Ver nota 2.

LLDT *{reg16|mem16}* (*Load Local Descriptor Table Register*): Carga el selector indicado por el operando en el registro LDTR. Antes de ejecutar esta instrucción la tabla deberá estar en memoria. Ver notas 1 y 2.

LMSW *{reg16|mem16}* (*Load Machine Status Word*): Carga el valor del operando en la palabra de estado de la máquina **MSW**. El bit **PE** (bit 0) no puede ser puesto a cero por esta instrucción, por lo que una vez que se cambió a modo protegido, la única manera de volver a modo real es mediante un RESET del microprocesador. Ver nota 2.

LSL *dest, src* (*Load Segment Limit*): Carga el límite del segmento de un selector especificado en *src* en el registro destino si el selector es válido y visible en el nivel de privilegio actual. Si ocurre lo anterior el indicador ZF se pone a uno, en caso contrario, se pone a cero. Ver notas 1 y 3.

LTR *{reg16|mem16}* (*Load Task Register*): Carga el selector indicado por el operando en el registro **TR**. El TSS (*Task State Segment*) apuntado por el nuevo **TR** deberá ser válido. Ver notas 1 y 2.

SGDT *mem64* (*Store Global Descriptor Table register*): Almacena el contenido del registro GDTR en el operando especificado.

SIDT *mem64* (*Store Interrupt Descriptor Table register*): Almacena el contenido del registro IDTR en el operando especificado.

SLDT *{reg16|mem16}* (*Store Global Descriptor Table register*): Almacena el contenido del registro LDTR (que es un selector a la tabla de descriptores globales) en el operando especificado. Ver nota 1.

SMSW *{reg16|mem16}* (*Store Machine Status Word*): Almacena la palabra de estado de la máquina **MSW** en el operando especificado. Ver nota 2.

STR *{reg16|mem16}* (*Store Task Register*): Almacena el registro de tarea actual (selector a la tabla de descriptores globales) en el operando especificado. Ver nota 1.

VERR/VERW *{reg16|mem16}* (*Verify Read/Write*): Verifica si el selector de segmento especificado en el operando es válido y se puede leer/escribir en el nivel de privilegio actual. En este caso se pone ZF a uno, en caso contrario se pone ZF a cero. Ver notas 1 y 3.

Notas:

1) Si se ejecuta en modo real ocurre una excepción 6 (Código de operación inválido).

2) Si se ejecuta en modo protegido en alguno de los anillos 1-3 ocurre una excepción 13 (Violación general de protección).

3) Cualquier violación de privilegio del selector indicado en el operando no causa una excepción 13, en vez de ello, el indicador ZF se pone a cero.

El coprocesador matemático 80287

Introducción

La interfaz coprocesador-CPU es totalmente diferente que en el caso del 8087. Como el 80286 implementa protección de memoria a través de un MMU basado en segmentación, hubiera sido demasiado caro duplicar esta lógica en el coprocesador, que una solución como la interfaz 8086/8088 a 8087 hubiera demandado. En vez de ello, en un sistema con 80286 y 80287 la CPU busca y almacena todos los códigos de operación para el coprocesador. La información se pasa a través de los **puertos F8h-FFh** del CPU. Como estos puertos son accesibles bajo el control del programa, se debe tener cuidado en los programas que no se escriban datos en dichos puertos, ya que esto podría corromper datos en el coprocesador.

La combinación 8087/8088 se puede caracterizar como una cooperación de compañeros, mientras que el 80286/287 es más una relación amo/esclavo. Esto hace más fácil la sincronización, ya que la instrucción completa y el flujo de datos del coprocesador pasa a través de la CPU. Antes de ejecutar la mayoría de las instrucciones del coprocesador, el 80286 verifica su pin **/BUSY**, que está conectado al 80287 y señala si aún está ejecutando una instrucción previa o encontró una excepción. Por lo tanto la instrucción **WAIT** antes de las instrucciones del coprocesador está permitida pero no es necesaria.

La unidad de ejecución del 80287 es prácticamente idéntica al del 8087, esto es, casi todas las instrucciones se ejecutan en la misma cantidad de ciclos de reloj en ambos coprocesadores. Sin embargo debido a la comunicación que debe realizar el 80287 con la CPU (alrededor de 40 ciclos de reloj), una combinación 80286/80287 puede tener menor rendimiento de punto flotante que un sistema 8086/8087 corriendo a la misma velocidad de reloj.

Versiones del 80287: El 80287 fue el coprocesador original para el procesador 80286 y se introdujo en 1983. Usa la misma unidad de ejecución interna que el 8087 y tiene la misma velocidad (a veces es más lento debido al tiempo que tarda la comunicación con el 80286). Como el 8087, no provee compatibilidad completa con la norma IEEE-754 de punto flotante que apareció en 1985. El 80287 fue realizado en tecnología NMOS con el formato externo CERDIP de 40 pines. La frecuencia máxima era de 10 MHz. El consumo de potencia era el mismo que el del 8087 (2,4 watt máximo).

El 80287 ha sido reemplazado pon su sucesor más rápido, el 287XL, basado en tecnología CMOS e introducido en 1990. Como tiene la misma unidad de ejecución que el 80387, cumple con la norma IEEE-754 y ejecuta bastante más rápido que su antecesor. A 12,5 MHz consume 675 mW (la cuarta parte que su antecesor).

Nuevas instrucciones del 80287

Aparte de las instrucciones que existen en el 8087, el 80287 posee las siguientes instrucciones: **FSTSW** *AX*: Almacena el registro de estado del 80287 en el registro AX.

FSETPM: Sirve para indicarle al coprocesador que la CPU entró en modo protegido y, que por lo tanto, las direcciones se tratan de una manera diferente que en modo real.

En el 287XL se incluyen las instrucciones que figuran en la sección Nuevas instrucciones del 80387.

El microprocesador 80386

Introducción

El 80386 consiste en una unidad central de proceso (**CPU**), una unidad de manejo de memoria (**MMU**) y una unidad de interfaz con el bus (**BIU**).

La **CPU** está compuesta por la unidad de ejecución y la unidad de instrucciones. La unidad de ejecución contiene los ocho registros de 32 bits de propósito general que se utilizan para el cálculo de direcciones y operaciones con datos y un *barrel shifter* de 64 bits que se utiliza para acelerar las operaciones de desplazamiento, rotación, multiplicación y división. Al contrario de los microprocesadores previos, la lógica de división y multiplicación utiliza un algoritmo de 1 bit por ciclo de reloj. El algoritmo de multiplicación termina la iteración cuando los bits más significativos del multiplicador son todos ceros, lo que permite que las multiplicaciones típicas de 32 bits se realicen en menos de un microsegundo.

La unidad de instrucción decodifica los códigos de operación (*opcodes*) de las instrucciones que se encuentran en una cola de instrucciones (cuya longitud es de 16 bytes) y los almacena en la cola de instrucciones decodificadas (hay espacio para tres instrucciones).

El sistema de control de la unidad de ejecución es el encargado de decodificar las instrucciones que le envía la cola y enviarle las órdenes a la unidad aritmética y lógica según una tabla que tiene almacenada en ROM llamada CROM (Control Read Only Memory).

La unidad de manejo de memoria (**MMU**) consiste en una unidad de segmentación (similar a la del 80286) y una unidad de paginado (nuevo en este microprocesador). La segmentación permite el manejo del espacio de direcciones lógicas agregando un componente de direccionamiento extra, que permite que el código y los datos se puedan reubicar fácilmente. El mecanismo de paginado opera por debajo y es transparente al proceso de segmentación, para permitir el manejo del espacio de direcciones físicas. Cada segmento se divide en uno o más páginas de 4 kilobytes. Para implementar un sistema de memoria virtual (aquél donde el programa tiene un tamaño mayor que la memoria física y debe cargarse por partes (páginas) desde el disco rígido), el 80386 permite seguir ejecutando los programas después de haberse detectado fallos de segmentos o de páginas. Si una página determinada no se encuentra en memoria, el 80386 se lo indica al sistema operativo mediante la excepción 14, luego éste carga dicha página desde el disco y finalmente puede seguir ejecutando el programa, como si hubiera estado dicha página todo el tiempo. Como se puede observar, este proceso es transparente para la aplicación, por lo que el programador no debe preocuparse por cargar partes del código desde el disco ya que esto lo hace el sistema operativo con la ayuda del microprocesador.

La memoria se organiza en uno o más segmentos de longitud variable, con tamaño máximo de 4 gigabytes. Estos segmentos, como se vio en la explicación del 80286, tienen atributos asociados, que incluyen su ubicación, tamaño, tipo (pila, código o datos) y características de protección.

La unidad de segmentación provee cuatro niveles de protección para aislar y proteger aplicaciones y el sistema operativo. Este tipo de protección por hardware permite el diseño de sistemas con un alto grado de integridad.

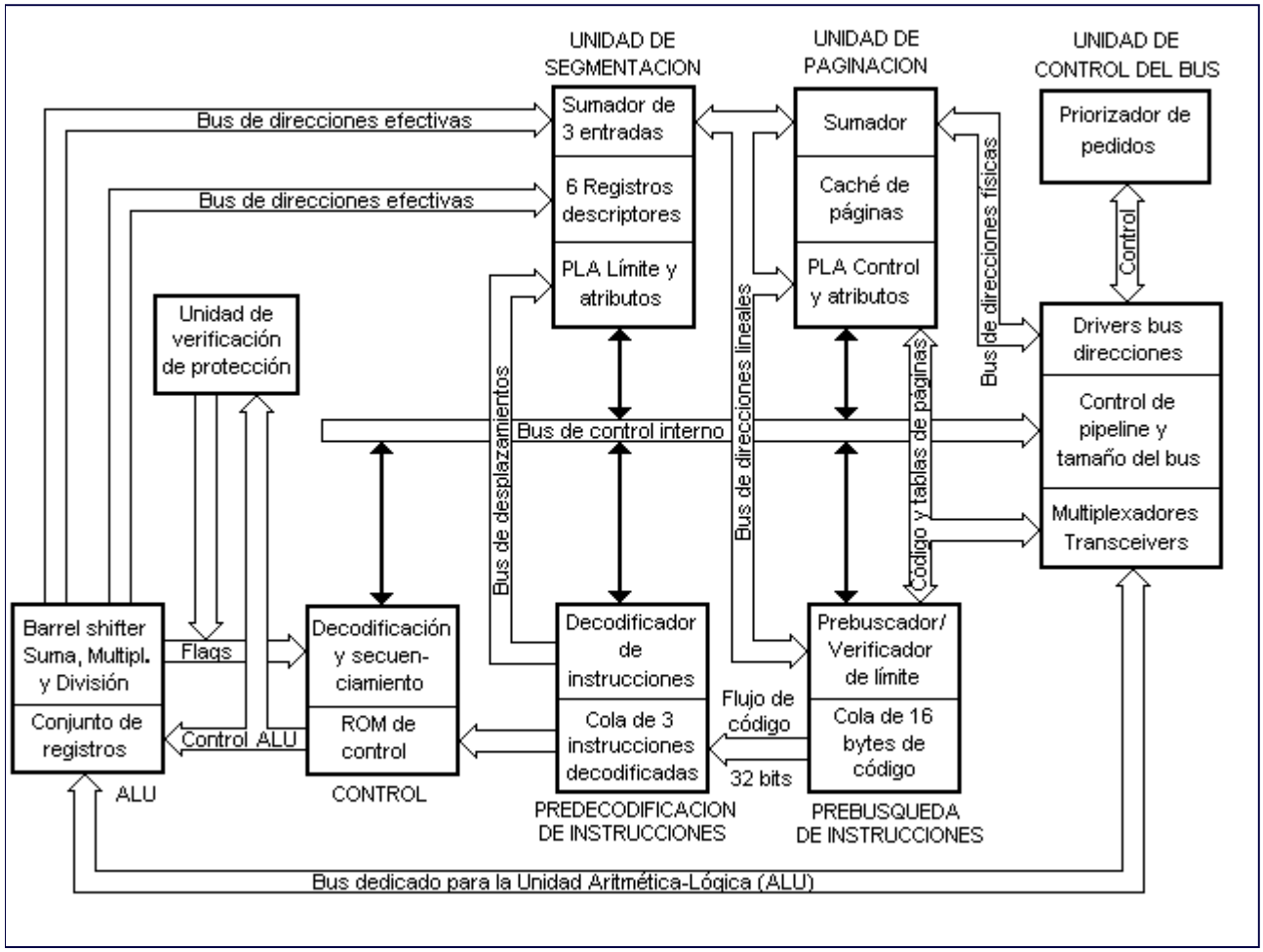
El 80386 tiene dos modos de operación: modo de direccionamiento real (modo real), y modo de direccionamiento virtual protegido (modo protegido). En modo real el 80386 opera como un 8086 muy

rápido, con extensiones de 32 bits si se desea. El modo real se requiere primariamente para preparar el procesador para que opere en modo protegido. El modo protegido provee el acceso al sofisticado manejo de memoria y paginado.

Dentro del modo protegido, el software puede realizar un cambio de tarea para entrar en tareas en modo 8086 virtual (*V86 mode*) (esto es nuevo con este microprocesador). Cada una de estas tareas se comporta como si fuera un 8086 el que lo está ejecutando, lo que permite ejecutar software de 8086 (un programa de aplicación o un sistema operativo). Las tareas en modo 8086 virtual pueden aislarse entre sí y del sistema operativo (que debe utilizar instrucciones del 80386), mediante el uso del paginado y el mapa de bits de permiso de entrada/salida (*I/O Permission Bitmap*).

Finalmente, para facilitar diseños de hardware de alto rendimiento, la interfaz con el bus del 80386 ofrece *pipelining* de direcciones, tamaño dinámico del ancho del bus de datos (puede tener 16 ó 32 bits según se desee en un determinado ciclo de bus) y señales de habilitación de bytes por cada byte del bus de datos. Hay más información sobre esto en la sección de hardware del 80386.

Diagrama en bloques del 80386



Versiones del 80386

- **80386**: En octubre de 1985 la empresa Intel lanzó el microprocesador **80386** original de 16 MHz,

con una velocidad de ejecución de 6 millones de instrucciones por segundo y con 275.000 transistores. La primera empresa en realizar una computadora compatible com IBM PC AT basada en el 80386 fue Compaq con su Compaq Deskpro 386 al año siguiente.

- **386SX**: Para facilitar la transición entre las computadoras de 16 bits basadas en el 80286, apareció en junio de 1988 el **80386 SX** con bus de datos de 16 bits y 24 bits de direcciones (al igual que en el caso del 80286). Este microprocesador permitió el armado de computadoras en forma económica que pudieran correr programas de 32 bits. El 80386 original se le cambió de nombre: **80386 DX**.

- **386SL**: En 1990 Intel introdujo el miembro de alta integración de la familia 386: el **80386 SL** con varias características extras (25 MHz, frecuencia reducida ó 0 MHz, interfaz para caché opcional externo de 16, 32 ó 64 KB, soporte de LIM 4.0 (memoria expandida) por hardware, generación y verificación de paridad, ancho de bus de datos de 8 ó 16 bits) que lo hacen ideal para equipos portátiles.

Registros del 80386

El 80386 tiene registros de 32 bits en las siguientes categorías:

- Registros de propósito general.
- Registros de segmento.
- Puntero de instrucciones
- Indicadores.
- Registros de control (nuevos en el 80386).
- Registros de direcciones de sistema.
- Registros de depuración (*debug*) (nuevos en el 80386).
- Registros de test (nuevos en el 80386).

Todos los registros de los microprocesadores 8086, 80186 y 80286 son un subconjunto de los del 80386.

La siguiente figura muestra los registros de la arquitectura base del 80386, que incluye los registros de uso general, el puntero de instrucciones y el registro de indicadores. Los contenidos de estos registros y de los selectores del párrafo siguiente son específicos para cada tarea, así que se cargan automáticamente al ocurrir una operación de cambio de tarea. La arquitectura base también incluye seis segmentos direccionables directamente, cada uno de 4 gigabytes de tamaño máximo. Los segmentos se indican mediante valores de selectores puestos en los registros de segmento del 80386. Si se desea se pueden cargar diferentes selectores a medida que corre el programa.

Registros de la arquitectura base

Tipo	Registros	Bits 31-16	Bits 15-0	Descripción
Uso general	**EAX**	EAX_{31-16}	EAX_{15-0} = **AX**	Acumulador
	EBX	EBX_{31-16}	EBX_{15-0} = **BX**	Base
	ECX	ECX_{31-16}	ECX_{15-0} = **CX**	Contador
	EDX	EDX_{31-16}	EDX_{15-0} = **DX**	Datos
	ESI	ESI_{31-16}	ESI_{15-0} = **SI**	Indice Fuente
	EDI	EDI_{31-16}	EDI_{15-0} = **DI**	Indice Destino
	EBP	EBP_{31-16}	EBP_{15-0} = **BP**	Puntero Base
	ESP	ESP_{31-16}	ESP_{15-0} = **SP**	Puntero de Pila
De segmento	**CS**	No aplicable: estos registros son de 16 bits	**CS**	Segmento de código
	SS		**SS**	Segmento de pila
	DS		**DS**	Segmento de datos
	ES		**DS**	Segmentos de datos extra
	FS		**DS**	
	GS		**DS**	
Otros	**EIP**	EIP_{31-16}	EIP_{15-0} = **IP**	Puntero de instrucciones
	EFlags	$EFlags_{31-16}$	$EFlags_{15-0}$ = **Flags**	Indicadores

Registros de propósito general

Los ocho registros de propósito general de 32 bits mantienen datos y direcciones. Estos registros soportan operandos de 1, 8, 16, 32 y 64 bits y campos de bits de 1 a 32 bits. Soportan operandos de direcciones de 16 y de 32 bits. Los nombres simbólicos son: **EAX**, **EBX**, **ECX**, **EDX**, **ESI**, **EDI**, **EBP** y **ESP**. Los 16 bits menos significativos se pueden acceder separadamente. Esto se hace usando los nombres **AX**, **BX**, **CX**, **DX**, **SI**, **DI**, **BP** y **SP**, que se utilizan de la misma manera que en los procesadores previos. Al igual que en el 80286 y anteriores, **AX** se divide en **AH** y **AL**, **BX** se divide en **BH** y **BL**, **CX** se divide en **CH** y **CL** y **DX** se divide en **DH** y **DL**.

Los ocho **registros de uso general** de 32 bits se pueden usar para direccionamiento indirecto. Cualquiera de los ocho registros puede ser la base y cualquiera menos **ESP** puede ser el índice. El índice se puede multiplicar por 1, 2, 4 u 8.

Ejemplos de direccionamiento indirecto:

- *MOV ECX,[ESP]*
- *MOV AL, [EAX + EDI * 8]*
- *ADD CL, [EDX + EDX + 8245525h]*
- *INC DWORD PTR TABLA[EAX * 4]*

En modo real y en modo 8086 virtual, la suma de la base, el índice y el desplazamiento debe estar entre 0 y 65535 para que no se genere una excepción 13. El segmento por defecto es **SS** si se utiliza **EBP** o **ESP** como base, en caso contrario es **DS**. En el caso de usar direccionamiento de 16 bits, sólo se pueden usar las mismas combinaciones que para el 8088. No se pueden mezclar registros de 16 y de 32 bits para direccionamiento indirecto.

Puntero de instrucciones

El **puntero de instrucciones** es un registro de 32 bits llamado **EIP**, el cual mantiene el offset de la próxima instrucción a ejecutar. El offset siempre es relativo a la base del segmento de código (**CS**). Los 16 bits menos significativos de **EIP** conforman el *puntero de instrucciones de 16 bits* llamado **IP**, que se utiliza para direccionamiento de 16 bits.

Registro de indicadores

Es un registro de 32 bits llamado **EFlags**. Los bits definidos y campos de bits controlan ciertas operaciones e indican el estado del 80386. Los 16 bits menos significativos (bits 15-0) llevan el nombre de **Flags**, que es más útil cuando se ejecuta código de 8086 y 80286. La descripción de los indicadores es la siguiente:

- **VM** (*Virtual 8086 Mode*, bit 17): Este bit provee el modo 8086 virtual dentro del modo protegido. Si se pone a 1 cuando el 80386 está en modo protegido, entrará al modo 8086 virtual, manejando los segmentos como lo hace el 8086, pero generando una excepción 13 (Violación general de protección) en instrucciones privilegiadas (aquéllas que sólo se pueden ejecutar en modo real o en el anillo 0). El bit **VM** sólo puede ponerse a 1 en modo protegido mediante la instrucción IRET (ejecutando en nivel de privilegio cero) y por cambios de tarea en cualquier anillo. El bit **VM** no cambia con la instrucción POPFD. La instrucción PUSHFD siempre pone un cero en el bit correspondiente en la pila, aunque el modo 8086 virtual esté activado. La imagen de **EFlags** puesta en la pila durante el procesamiento de una interrupción o salvada en un Task State Segment durante cambios de tarea, contendrá un uno en el bit 17 si el código interrumpido estaba ejecutando una tarea tipo 8086 virtual.

- **RF** (*Resume Flag*, bit 16): Este indicador se utiliza junto con los registros de depuración. Se verifica en las fronteras de instrucciones antes del procesamiento de los puntos de parada (*breakpoints*). Cuando **RF** vale 1, hace que se ignoren las *faltas* (hechos que ocasionan una excepción) de depuración. **RF** se pone automáticamente a cero luego de ejecutar correctamente cualquier instrucción (no se señalan faltas) excepto las instrucciones IRET y POPF y en los cambios de tarea causados por la ejecución de JMP, CALL e INT. Estas instrucciones ponen **RF** al valor especificado por la imagen almacenada en memoria. Por ejemplo, al final de la rutina de servicio de los puntos de parada, la instrucción IRET puede extraer de la pila una imagen de **EFlags** que tiene **RF** = 1 y resumir la ejecución del programa en la dirección del punto de parada sin generar otra falta de punto de parada en el mismo lugar.

- **NT** (*Nested Task*, bit 14): Este indicador se aplica al modo protegido. **NT** se pone a uno para indicar que la ejecución de la tarea está anidada dentro de otra tarea. Si está a uno, indica que el segmento de estado de la tarea (**TSS**) de la tarea anidada tiene un puntero válido al TSS de la tarea previa. Este bit se pone a cero o uno mediante transferencias de control a otras tareas. La instrucción IRET verifica el valor de **NT** para determinar si debe realizar un retorno dentro de la

misma tarea o si debe hacer un cambio de tarea. Un POPF o IRET afectará el valor de este indicador de acuerdo a la imagen que estaba en la pila, en cualquier nivel de privilegio.

- **IOPL** (*Input/Output Privilege Level*, bits 13-12): Este campo de dos bits se aplica al modo protegido. **IOPL** indica el CPL (*Current Privilege Level*) numéricamente máximo (esto es, con menor nivel de privilegio) permitido para realizar instrucciones de entrada/salida sin generar una excepción 13 (Violación general de protección) o consultar el mapa de bits de permiso de E/S (este mapa está ubicado en el segmento de estado de tarea (**TSS**) con el nuevo formato que provee el 80386). Además indica el máximo valor de CPL que permite el cambio del indicador **IF** (indicador de habilitación del pin **INTR**) cuando se ejecuta POPF e IRET. Estas dos últimas instrucciones sólo pueden alterar **IOPL** cuando CPL = 0. Los cambios de tarea siempre alteran el campo **IOPL**, cuando la nueva imagen de los indicadores se carga desde el TSS de la nueva tarea.

- **OF** (*Overflow flag*, bit 11): Si vale 1, hubo un desborde en una operación aritmética con signo, esto es, un dígito significativo se perdió debido a que tamaño del resultado es mayor que el tamaño del destino.

- **DF** (*Direction Flag*, bit 10): Define si **ESI** y/o **EDI** se autoincrementan (**DF** = 0) o autodecrementan (**DF** = 1) durante instrucciones de cadena.

- **IF** (***INTR** enable Flag*, bit 9): Si vale 1, permite el reconocimiento de interrupciones externas señaladas en el pin **INTR**. Cuando vale cero, las interrupciones externas señaladas en el pin **INTR** no se reconocen. **IOPL** indica el máximo valor de CPL que permite la alteración del indicador **IF** cuando se ponen nuevos valores en **EFlags** desde la pila.

- **TF** (*Trap enable Flag*, bit 8): Controla la generación de la excepción 1 cuando se ejecuta código paso a paso. Cuando **TF** = 1, el 80386 genera una excepción 1 después que se ejecuta la instrucción en curso. Cuando **TF** = 0, la excepción 1 sólo puede ocurrir como resultado de las direcciones de punto de parada cargadas en los registros de depuración (**DR0**-**DR3**).

- **SF** (*Sign Flag*, bit 7): Se pone a 1 si el bit más significativo del resultado de una operación aritmética o lógica vale 1 y se pone a cero en caso contrario. Para operaciones de 8, 16, 32 bits, el indicador **SF** refleja el estado de los bits 7, 15 y 31 respectivamente.

- **ZF** (*Zero Flag*, bit 6): Se pone a 1 si todos los bits del resultado valen cero, en caso contrario se pone a cero.

- **AF** (*Auxiliary carry Flag*, bit 4): Se usa para simplificar la adición y sustracción de cantidades BCD empaquetados. **AF** se pone a 1 si hubo un préstamo o acarreo del bit 3 al 4. De otra manera el indicador se pone a cero.

- **PF** (*Parity Flag*, bit 2): se pone a uno si los ocho bits menos significativos del resultado tienen un número par de unos (paridad par), y se pone a cero en caso de paridad impar (cantidad impar de unos). El indicador **PF** es función de los ocho bits menos significativos del resultado, independientemente del tamaño de las operaciones.

- **CF** (*Carry Flag*, bit 0): Se pone a uno si hubo arrastre (suma) o préstamo (resta) del bit más significativo del resultado.

Los bits 5 y 3 siempre valen cero, mientras que el bit 1 siempre vale uno.

Registros de segmento del 80386

Son seis registros de 16 bits que mantienen valores de selectores de segmentos identificando los segmentos que se pueden direccionar. En modo protegido, cada segmento puede tener entre un byte y el espacio total de direccionamiento (4 gigabytes). En modo real, el tamaño del segmento siempre es 64 KB.

Los seis segmentos direccionables en cualquier momento se definen mediante los registros de segmento **CS**, **DS**, **ES**, **FS**, **GS**, **SS**. El selector en **CS** indica el segmento de código actual, el selector en **SS** indica el segmento de pila actual y los selectores en los otros registros indican los segmentos actuales de datos.

Registros descriptores de segmento: Estos registros no son visibles para el programador, pero es muy útil conocer su contenido. Dentro del 80386, un **registro descriptor** (invisible para el programador) está asociado con cada registro de segmento (visible para el programador). Cada descriptor mantiene una dirección base de 32 bits, un límite (tamaño) de 32 bits y otros atributos del segmento.

Cuando un selector se carga en un registro de segmento, el **registro descriptor** asociado se cambia automáticamente con la información correcta. En modo real, sólo la dirección base se cambia (desplazando el valor del selector cuatro bits hacia la izquierda), ya que el límite y los otros atributos son fijos. En modo protegido, la dirección base, el límite y los otros atributos se cargan con el contenido de una tabla usando el selector como índice.

Siempre que ocurre una referencia a memoria, se utiliza automáticamente el **registro descriptor de segmento** asociado con el segmento que se está usando. La dirección base de 32 bits se convierte en uno de los componentes para calcular la dirección, el límite de 32 bits se usa para verificar si una referencia no supera dicho límite (no se referencia fuera del segmento) y los atributos se verifican para determinar si hubo alguna violación de protección u otro tipo.

Registros de control

El 80386 tiene tres registros de control de 32 bits, llamados **CR0**, **CR2** y **CR3**, para mantener el estado de la máquina de naturaleza global (no el específico de una tarea determinada). Estos registros, junto con los registros de direcciones del sistema, mantienen el estado de la máquina que afecta a todas las tareas en el sistema. Para acceder los registros de control, se utiliza la instrucción **MOV**.

CR0 (Registro de control de la máquina): Contiene seis bits definidos para propósitos de control y estado. Los 16 bits menos significativos de **CR0** también se conocen con el nombre de *palabra de estado de la máquina* (**MSW**), para la compatibilidad con el modo protegido del 80286. Las instrucciones LMSW y SMSW se toman como casos particulares de carga y almacenamiento de **CR0** donde sólo se opera con los 16 bits menos significativos de **CR0**. Para lograr la compatibilidad con sistemas operativos del 80286 la instrucción LMSW opera en forma idéntica que en el 80286 (ignora los nuevos bits definidos en **CR0**). Los bits definidos de **CR0** son los siguientes:

- **PG** (*Paging Enable*, bit 31 de **CR0**): Este bit se pone a uno para habilitar la unidad de paginado que posee el chip. Se pone a cero para deshabilitarlo. El paginado sólo funciona en modo protegido, por lo que si **PG** = 1, deberá ser PE = 1. Nuevo en 80386.

- **ET** (*Processor Extension Type*, bit 4 de **CR0**): Indica el tipo de coprocesador (80287, 80387)

según se detecte por el nivel del pin **/ERROR** al activar el pin **RESET**. El bit **ET** puede ponerse a cero o a uno cargando **CR0** bajo control del programa. Si **ET** = 1, se usa el protocolo de 32 bits del 80387, mientras que si **ET** = 0, se usa el protocolo de 16 bits del 80287. Para lograr la compatibilidad con el 80286, la instrucción LMSW no altera este bit. Sin embargo, al ejecutar SMSW, se podrá leer en el bit 4 el valor de este indicador. Nuevo en 80386.

- **TS** (*Task Switched*, bit 3 de **CR0**): Se pone a uno cuando se realiza una operación de cambio de tarea. Si **TS** = 1, un código de operación **ESC** del coprocesador causará una excepción 7 (Coprocesador no disponible). El manejador de la excepción típicamente salva el contexto del 80287/80387 que pertenece a la tarea previa, carga el estado del 80287/80387 con los valores de la nueva tarea y pone a cero el indicador **TS** antes de retornar a la instrucción que causó la excepción.

- **EM** (*Emulate Coprocessor*, bit 2 de **CR0**): Si este bit vale uno, hará que todos los códigos de operación de coprocesador causen una excepción 7 (Coprocesador no disponible). Si vale cero, permite que esas instrucciones se ejecuten en el 80287/80387 (éste es el valor del indicador después del RESET). El código de operación WAIT no se ve afectado por el valor del indicador.

- **MP** (*Monitor Coprocessor*, bit 1 de **CR0**): Este indicador se usa junto con TS para determinar si la instrucción WAIT generará una excepción 7 cuando TS = 1. Si MP = TS = 1, al ejecutar WAIT se genera la excepción. Nótese que TS se pone a uno cada vez que se realiza un cambio de tarea.

- **PE** (*Protection Enable*, bit 0 de **CR0**): Se pone a uno para habilitar el modo protegido. Si vale cero, se opera otra vez en modo real. **PE** se puede poner a uno cargando **MSW** o **CR0**, pero puede ser puesto a cero sólo mediante una carga en **CR0**. Poner el bit **PE** a cero es apenas una parte de una secuencia de instrucciones necesarias para la transición de modo protegido a modo real. Nótese que para la compatibilidad estricta con el 80286, **PE** no puede ponerse a cero con la instrucción LMSW.

CR2 (Dirección lineal de falta de página): Mantiene la dirección lineal de 32 bits que causó la última falta de página detectada. El código de error puesto en la pila del manejador de la falta de página cuando se la invoca provee información adicional sobre la falta de página.

Un cambio de tareas a través de un TSS que cambie el valor de **CR3**, o una carga explícita de **CR3** con cualquier valor, invalidará todas las entradas en la tabla de páginas que se encuentran en el caché de la unidad de paginación. Si el valor de **CR3** no cambia durante el cambio de tareas se considerarán válidos los valores almacenados en el caché.

Registros de direcciones del sistema

Cuatro registros especiales se definen en el modelo de protección del 80286/80386 para referenciar tablas o segmentos. Estos últimos son:

- GDT (Tabla de descriptores globales).
- IDT (Tabla de descriptores de interrupción).
- LDT (Tabla de descriptores locales).
- TSS (Segmento de estado de la tarea).

Las direcciones de estas tablas y segmentos se almacenan en registros especiales, llamados **GDTR**, **IDTR**, **LDTR** y **TR** respectivamente.

GDTR e **IDTR**: Estos registros mantienen la dirección lineal base de 32 bits y el límite de 16 bits de **GDT** e **IDT**, respectivamente. Los segmentos **GDT** e **IDT**, como son globales para todas las tareas en el sistema, se definen mediante direcciones lineales de 32 bits (sujeto a traducción de página si el paginado está habilitado mediante el bit PG) y límite de 16 bits.

LDTR y **TR**: Estos registros mantienen los selectores de 16 bits para el descriptor de LDT y de TSS, respectivamente. Los segmentos LDT y TSS, como son específicos para cada tarea, se definen mediante valores de selector almacenado en los registros de segmento del sistema. Nótese que un registro descriptor del segmento (invisible para el programador) está asociado con cada registro de segmento del sistema.

Registros de depuración

Al igual que en los procesadores anteriores, el 80386 tiene algunas características que simplifica el proceso de depuración de programas. Las dos características compartidas con los microprocesadores anteriores son:

1) El código de operación de punto de parada INT 3 (**0CCh**).

2) La capacidad de ejecución paso a paso que provee el indicador TF.

Lo nuevo en el 80386 son los **registros de depuración**. Los seis registros de depuración de 32 bits accesibles al programador, proveen soporte para depuración (*debugging*) por hardware. Los registros DR0-DR3 especifican los cuatro puntos de parada (*breakpoints*). Como los puntos de parada se indican mediante registros en el interior del chip, un punto de parada de ejecución de instrucciones se puede ubicar en memoria ROM o en código compartido por varias tareas, lo que no es posible utilizando el código de operación INT 3. El registro de control DR7 se utiliza para poner y habilitar los puntos de parada y el registro de estado DR6 indica el estado actual de los puntos de parada. Después del *reset*, los puntos de parada están deshabilitados. Los puntos de parada que ocurren debido a los registros de depuración generan una excepción 1.

Registros de direcciones lineales de puntos de parada (DR0 - DR3): Se pueden especificar hasta cuatro direcciones de puntos de parada escribiendo en los registros **DR0** a **DR3**. Las direcciones especificadas son direcciones lineales de 32 bits. El hardware del 80386 continuamente compara las direcciones lineales de los registros con las direcciones lineales que genera el software que se está ejecutando (una dirección lineal es el resultado de computar la dirección efectiva y sumarle la dirección base de 32 bits del segmento). Nótese que si la unidad de paginación del 80386 no está habilitada (mediante el bit PG del registro CR0), la dirección lineal coincide con la física (la que sale por el bus de direcciones), mientras que si está habilitada, la dirección lineal se traduce en la física mediante dicha unidad.

Registro de control de depuración (DR7): La definición de los bits de este registro es la siguiente:

Campo	**LEN3**		**RW3**		**LEN2**		**RW2**		**LEN1**		**RW1**		**LEN0**		**RW0**	
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

Campo	Indef.		GD	Indef.			GE	LE	G3	L3	G2	L2	G1	L1	G0	L0
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- Campo **LENi** (*Campo de especificación de longitud del punto de parada*): Por cada punto de parada existe un campo de dos bits **LEN**. Este campo especifica la longitud del campo que produce el punto de parada. En el caso de punto de parada por acceso de datos se puede elegir 1, 2 ó 4 bytes, mientras que los de lectura de instrucciones deben tener una longitud de un byte (**LENi** = 00). La codificación del campo es como sigue:

  LENi = 00 (1 byte): Los 32 bits de DRi se utilizan para especificar el punto de parada.
  LENi = 01 (2 bytes): Se utilizan los bits 31-1 de DRi.
  LENi = 10: Indefinido. No debe utilizarse.
  LENi = 11 (4 bytes): Se utilizan los bits 31-2 de DRi.

- Campo **RWi** (*Bits calificadores de acceso a memoria*): Existe un campo **RW** de dos bits por cada uno de los puntos de parada. Este campo especifica el tipo de uso de memoria que produce el punto de parada:

  RWi = 00: Ejecución de instrucciones solamente.
  RWi = 01: Escritura de datos solamente.
  RWi = 10: Indefinido. No debe utilizarse.
  RWi = 11: Lectura y/o escritura de datos solamente.

  Los puntos de parada (excepción 1) debido a la ejecución de instrucciones ocurren ANTES de la ejecución, mientras que los puntos de parada debido a acceso a datos ocurren DESPUES del acceso.

  Uso de LENi y RWi para poner un punto de parada debido a acceso a datos: Esto se realiza cargando la dirección lineal del dato en **DRi** (i=0-3). RWi puede valer 01 (sólo parar en escritura) o 11 (parar en lectura/escritura). LENi puede ser igual a 00 (un byte), 01 (2 bytes) ó 11 (4 bytes). Si un acceso de datos cae parcial o totalmente dentro del campo definido por **DRi** y LENi y el punto de parada está habilitado, se producirá la excepción 1.

  Uso de LENi y RWi para poner un punto de parada debido a la ejecución de una instrucción: Debe cargarse DRi con la dirección del comienzo (del primer prefijo si hay alguno). RWi y LENi deben valer 00. Si se está por ejecutar la instrucción que comienza en la dirección del punto de parada y dicho punto de parada está habilitado, se producirá una excepción 1 antes de que ocurra la ejecución de la instrucción.

- Campo **GD** (*Detección de acceso de registros de depuración*): Los registros de depuración sólo se pueden acceder en modo real o en el nivel de privilegio cero en modo protegido (no se pueden acceder en modo 8086 virtual). El bit **GD**, cuando vale uno, provee una protección extra contra cualquier acceso a estos registros aun en modo real o en el anillo 0 del modo protegido. Esta protección adicional sirve para garantizar que un depurador por software (Codeview, Turbo Debugger, etc.) pueda tener el control total de los recursos provistos por los registros de depuración cuando sea necesario. El bit **GD**, cuando vale 1, causa una excepción 1 si una instrucción trata de leer o escribir algún registro de depuración. El bit **GD** se pone automáticamente a cero al ocurrir dicha excepción, permitiendo al manejador el control de los registros de

depuración.

- Bits **GE** y **LE** (*Punto de parada exactamente cuando ocurre el acceso a datos, global y local*): Si **GE** o **LE** valen uno, cualquier punto de parada debido a acceso a datos se indicará exactamente después de completar la instrucción que causó la transferencia. Esto ocurre forzando la unidad de ejecución del 80386 que espere que termine la transferencia de datos antes de comenzar la siguiente instrucción. Si **GE** = **LE** = 0, la parada ocurrirá varias instrucciones después de la transferencia que lo debería haber causado. Cuando el 80386 cambia de tarea, el bit **LE** se pone a cero. Esto permite que las otras tareas se ejecuten a máxima velocidad. El bit **LE** debe ponerse a uno bajo control del software. El bit **GE** no se altera cuando ocurre un cambio de tarea. Los puntos de parada debido a la ejecución de instrucciones siempre ocurren exactamente, independientemente del valor de **GE** y **LE**.

- Bits **Gi** y **Li** (*Habilitación del punto de parada, global y local*): Si cualquiera de estos bits vale uno, se habilita el punto de parada asociado. Si el 80386 detecta la condición de parada ocurre una excepción 1. Cuando se realiza un cambio de tareas, todos los bits **Li** se ponen a cero, para evitar excepciones espúreas en la nueva tarea. Estos bits deben ponerse a uno bajo control del software. Los bits **Gi** no se afectan durante un cambio de tarea. Los bits **Gi** soportan puntos de parada que están activos en todas las tareas que se ejecutan en el sistema.

Registro de estado de depuración (DR6): Este registro permite que el manejador de la excepción 1 determine fácilmente por qué fue llamado. El manejador puede ser invocado como resultado de uno de los siguientes eventos:

1. Punto de parada debido a DR0.
2. Punto de parada debido a DR1.
3. Punto de parada debido a DR2.
4. Punto de parada debido a DR3.
5. Acceso a un registro de depuración cuando GD = 1.
6. Ejecución paso a paso (si TF = 1).
7. Cambio de tareas.

El registro DR6 contiene indicadores para cada uno de los eventos arriba mencionados. Los otros bits son indefinidos. Estos indicadores se ponen a uno por hardware pero nunca puestos a cero por hardware, por lo que el manejador de la excepción 1 deberá poner **DR6** a cero.

Condición	7	6	5	4	3	2	1
Indicador	BT	BS	BD	B3	B2	B1	B0
Bit	15	14	13	3	2	1	0

Registros de test

Se utilizan dos registros para verificar el funcionamiento del **RAM/CAM** (*Content Addressable Memory*) en el buffer de conversión por búsqueda (**TLB**) de la unidad de paginado del 80386. **TR6** es el registro de comando del test, mientras que **TR7** es el registro de datos que contiene el dato proveniente del **TLB**. El **TLB** guarda las entradas de tabla de página de uso más reciente en un caché que se incluye en el chip, para reducir los accesos a las tablas de páginas basadas en RAM.

Acceso a registros

Hay algunas diferencias en el acceso de registros en modo real y protegido, que se indican a continuación. Escr = Escritura del registro, Lect = Lectura del registro.

Registro	Modo Real		Modo Protegido		Modo virtual 8086	
	Escr	Lect	Escr	Lect	Escr	Lect
Registros generales	Sí	Sí	Sí	Sí	Sí	Sí
Registros de segmento	Sí	Sí	Sí	Sí	Sí	Sí
Indicadores	Sí	Sí	Sí	Sí	IOPL	IOPL
Registros de control	Sí	Sí	CPL=0	CPL=0	No	Sí
GDTR	Sí	Sí	CPL=0	Sí	No	Sí
IDTR	Sí	Sí	CPL=0	Sí	No	Sí
LDTR	No	No	CPL=0	Sí	No	No
TR	No	No	CPL=0	Sí	No	No
Registros de depuración	Sí	Sí	CPL=0	CPL=0	No	No
Registros de test	Sí	Sí	CPL=0	CPL=0	No	No

CPL=0: Los registros se pueden acceder sólo si el nivel de privilegio actual es cero.

IOPL: Las instrucciones PUSHF y POPF son sensibles al indicador IOPL en modo 8086 virtual.

Compatibilidad

Algunos bits de ciertos registros del 80386 no están definidos. Cuando se obtienen estos bits, deben tratarse como completamente indefinidos. Esto es esencial para la compatibilidad con procesadores futuros. Para ello deben seguirse las siguientes recomendaciones:

1. No depender de los estados de cualquiera de los bits no definidos. Estos deben ser enmascarados (mediante la instrucción AND con el bit a enmascarar a cero) cuando se utilizan los registros.

2. No depender de los estados de cualquiera de los bits no definidos cuando se los almacena en memoria u otro registro.

3. No depender de la habilidad que tiene el procesador de retener información escrita en bits marcados como indefinidos.

4. Cuando se cargan registros siempre se deben poner los bits indefinidos a cero.

5. Los registros que se almacenaron previamente pueden ser recargados sin necesidad de enmascarar los bits indefinidos.

Los programas que no cumplen con estas indicaciones, pueden llegar a no funcionar en 80486, Pentium y

siguientes procesadores, donde los bits no definidos del 80386 poseen algún significado en los otros procesadores.

Modo protegido en el 80386

Está basado en el modo protegido del 80286, por lo que se recomienda primero leer dicha información. Aquí se mostrarán las diferencias entre ambos microprocesadores.

Descriptores de segmento

Debido a la mayor cantidad de funciones del 80386, estos descriptores tienen más campos que los descriptores para el 80286.

El formato general de un descriptor en el 80386 es:

- **Byte 0**: Límite del segmento (bits 7-0).
- **Byte 1**: Límite del segmento (bits 15-8).
- **Byte 2**: Dirección base del segmento (bits 7-0).
- **Byte 3**: Dirección base del segmento (bits 15-8).
- **Byte 4**: Dirección base del segmento (bits 23-16).
- **Byte 5**: Derechos de acceso del segmento.
- **Byte 6**:
  - Bit 7: ***Granularidad*** (**G**): Si vale cero, el límite es el indicado por el campo límite (bits 19-0), mientras que si vale uno, a dicho campo se le agregarán 12 bits a uno a la derecha para formar el límite (de esta manera el límite se multiplica por 4096, pudiendo llegar hasta 4GB como límite máximo).
  - Bit 6: ***Default*** (para segmento de código)/***Big*** (para segmento de pila)(**D/B**): Para los descriptores de segmentos de código si este bit está a uno, el segmento será de 32 bits (se utiliza **EIP** como offset del puntero de instrucciones, mientras que si vale cero el segmento será de 16 bits (utiliza **IP** como offset del puntero de instrucciones). Para segmento de pila si el bit vale uno, el procesador usará el registro **ESP** como puntero de pila, en caso contrario usará **SP**.
  - Bits 5 y 4: Deben valer cero.
  - Bits 3 a 0: Límite del segmento (bits 19-16).
- **Byte 7**: Dirección base del segmento (bits 31-24).

El **byte de derechos de acceso** es el que define qué clase de descriptor es. El bit 4 (S) indica si el segmento es de código o datos (S = 1), o si es del sistema (S = 0). Veremos el primer caso.

- Bit 7: **Presente (P)**. Si P = 1 el segmento existe en memoria física, mientras que si P = 0 el segmento no está en memoria. Un intento de acceder un segmento que no está **presente** cargando un registro de segmento (CS, DS, ES, SS, FS o GS) con un selector que apunte a un descriptor que indique que el segmento no está **presente** generará una excepción 11 (en el caso de SS se generará una excepción 12 indicando que la pila no está **presente**). El manejador de la interrupción 11 deberá leer el segmento del disco rígido. Esto sirve para implementar memoria virtual.

- Bits 6 y 5: **Nivel de privilegio del descriptor (DPL)**: Atributo de privilegio del segmento utilizado en tests de privilegio.

- Bit 4: **Bit descriptor del segmento (S)**: Como se explicó más arriba, este bit vale 1 para descriptores de código y datos.

- Bit 3: **Ejecutable (E)**: Determina si el segmento es de código (E = 1), o de datos (E = 0).
  Si el bit 3 vale cero:
    - Bit 2: **Dirección de expansión (ED)**: Si E = 0, el segmento se expande hacia arriba, con lo que los offsets deben ser menores o iguales que el límite. Si E = 1, el segmento se expande hacia abajo, con lo que los offsets deben ser mayores que el límite. Si no ocurre esto se genera una excepción 13 (Fallo general de protección) (en el caso de la pila se genera una excepción 12).
    - Bit 1: **Habilitación de escritura (W)**: Si W = 0 no se puede escribir sobre el segmento, mientras que si W = 1 se puede realizar la escritura.

  Si el bit 3 vale uno:
    - Bit 2: **Conforme (C)**: Si C = 1, el segmento de código sólo puede ser ejecutado si CPL es mayor que DPL y CPL no cambia. Los segmentos conformes sirven para rutinas del sistema operativo que no requieran protección, tales como rutinas matemáticas, por ejemplo.
    - Bit 1: **Habilitación de lectura (R)**: Si R = 0, el segmento de código no se puede leer, mientras que si R = 1 sí. No se puede escribir sobre el segmento de código.
- Bit 0: **Accedido (A)**: Si A = 0 el segmento no fue accedido, mientras que si A = 1 el selector se ha cargado en un registro de segmento o utilizado por instrucciones de test de selectores. Este bit es puesto a uno por el microprocesador.

Si se lee o escribe en un segmento donde no está permitido o se intenta ejecutar en un segmento de datos se genera una excepción 13 (Violación general de protección). Si bien no se puede escribir sobre un segmento de código, éstos se pueden inicializar o modificar mediante un **alias**. Los **alias** son segmentos de datos con permiso de escritura (E = 0, W = 1) cuyo rango de direcciones coincide con el segmento de código.

Los segmentos de código cuyo bit C vale 1, pueden ejecutarse y compartirse por programas con diferentes niveles de privilegio (ver la sección sobre protección, más adelante).

A continuación se verá el formato del **byte de derechos de acceso** para descriptores de segmentos del sistema:

- Bit 7: **Presente (P)**: Igual que antes.
- Bits 6 y 5: **Nivel de privilegio del descriptor (DPL)**: Igual que antes.
- Bit 4: **Bit descriptor del segmento (S)**: Como se explicó más arriba, este bit vale 0 para descriptores del sistema.

- Bits 3-0: **Tipo de descriptor**: En el 80386 están disponibles los siguientes:

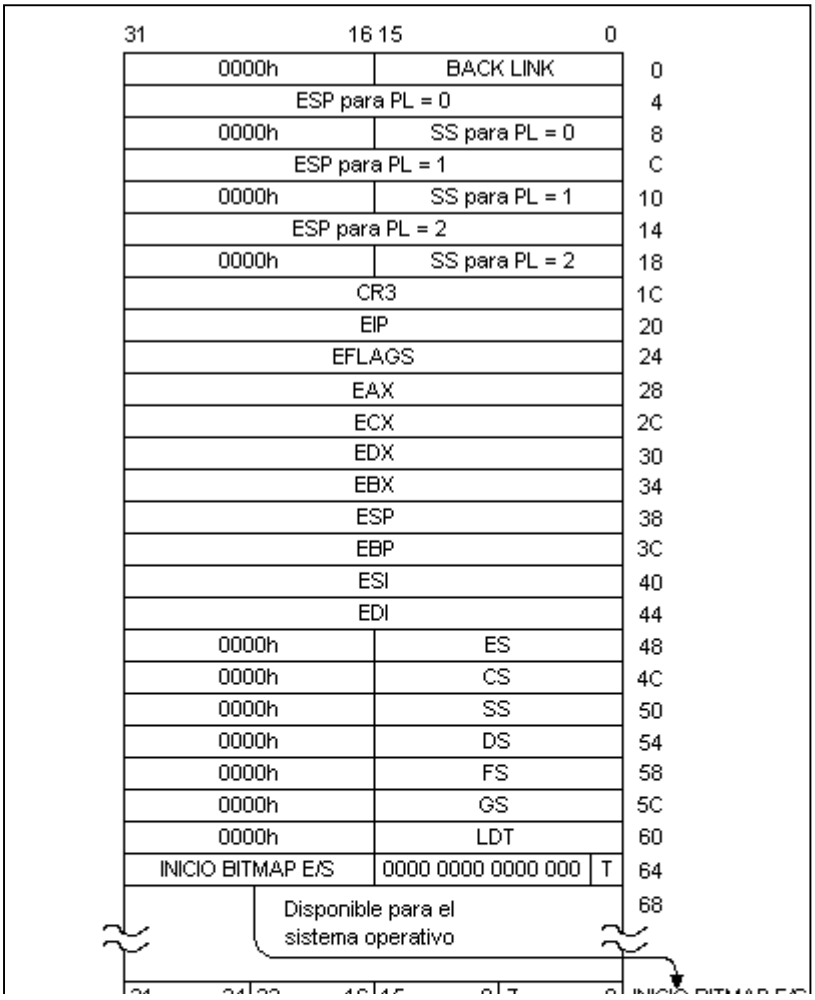
0000: Inválido	1000: Inválido
0001: TSS tipo 286 disponible	1001: TSS tipo 386 disponible
0010: LDT	1010: Inválido
0011: TSS tipo 286 ocupado	1001: TSS tipo 386 ocupado
0100: Compuerta de llamada tipo 286	1010: Compuerta de llamada tipo 386
0101: Compuerta de tarea tipo 286 ó 386	1011: Inválido
0110: Compuerta de interrupción tipo 286	1110: Compuerta de interrupción tipo 286
0111: Compuerta de trampa tipo 286	1111: Compuerta de trampa tipo 286

Vea la información sobre TSS tipo 286, LDT y compuertas en las secciones correspondientes del microprocesador 80286.

Segmento de estado de la tarea

El descriptor **TSS** apunta a un segmento que contiene el estado de la ejecución del 80386 mientras que un descriptor de compuerta de tarea contiene un selector de **TSS**.

En el 80386 hay dos tipos de **TSS**: tipo 286 y tipo 386. El primero es idéntico al TSS del microprocesador 80286, mientras que la del 80386 tiene el siguiente formato:



Mapa de bits de permisos de entrada/salida

Aparte de poder almacenar los registros nuevos del 80386, el segmento de estado de la tarea tiene este nuevo campo, como se puede apreciar en la parte inferior de la tabla.

Cuando el procesador debe acceder a un puerto de entrada/salida (usando una instrucción **OUT**, **IN**, **OUTS** o **INS**) en modo protegido, el procesador primero verifica si **CPL** <= **IOPL**. Si esto ocurre, la instrucción se ejecuta. En caso contrario si la tarea en ejecución está asociada a una TSS tipo 286 el 80386 lanza una excepción 13. Si está asociada con una TSS tipo 386, el procesador consulta esta tabla para saber si debe ejecutar la instrucción o si debe lanzar una excepción 13.

El mapa de bits de permisos de entrada/salida se puede ver como una cadena de bits cuya longitud puede ser entre cero y 65536 bits. Cada bit corresponde a un puerto (ver la figura que está más arriba). Si el bit está a cero, la instrucción de E/S se puede ejecutar. De esta manera se pueden proteger zonas de espacio de E/S en forma selectiva.

El inicio del mapa de bits está indicado por el puntero que está en el offset 66h del TSS. El mapa de bits puede ser truncado ajustando el límite del segmento TSS. Todos los puertos de E/S que no tengan una entrada en el mapa de bits debido a lo anterior, no se podrán acceder (es como si estuviera el bit correspondiente del mapa a "1").

Al final del mapa de bits debe haber un byte a FFh. Este byte debe estar dentro del límite del TSS.

Paginación en el 80386

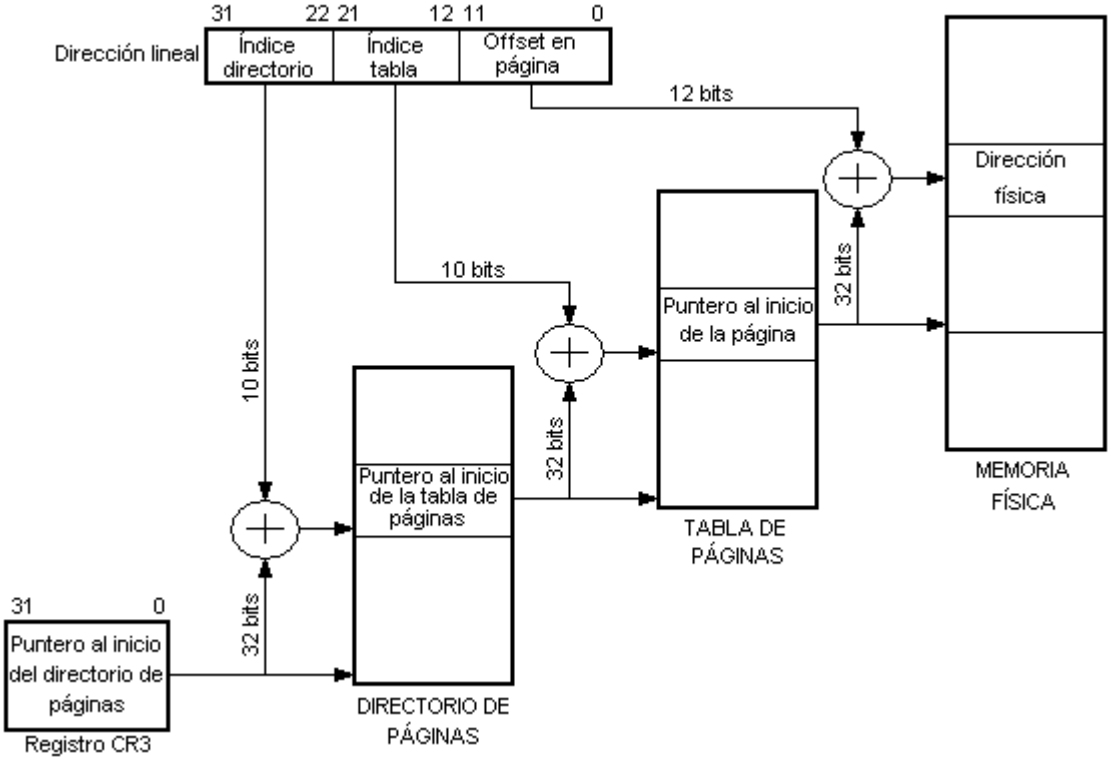
La paginación es un tipo de manejo de memoria útil para sistemas operativos multitarea que manejan memoria virtual. A diferencia de la segmentación que modulariza programas y datos en segmentos de longitud variable, la paginación divide los programas en varias páginas de tamaño fijo. Estas páginas no tienen niguna relación con la estructura lógica del programa.

Como la mayoría de los programas utilizan referencias cercanas (esto se llama efecto de *localidad*), sólo es necesaria una pequeña cantidad de páginas presentes en la memoria por cada tarea.

Mecanismo de paginación

El 80386 utiliza dos niveles de tablas para traducir las direcciones lineales (que vienen de la unidad de segmentación) en una dirección física. Los tres componentes del mecanismo de paginado son: el

directorio de páginas, las tablas de páginas y las páginas mismas. Cada uno de estos elementos ocupa 4KB en la memoria física. Un tamaño uniforme para todos los elementos simplifica el manejo de memoria, ya que no existe fragmentación. La siguiente figura muestra cómo funciona el mecanismo de paginación:



Registros de control usados para la paginación

El registro **CR2** es el que mantiene la dirección lineal de 32 bits que causó el último fallo de página detectado por el microprocesador.

El registro **CR3** contiene la dirección física inicial del directorio de páginas. Los doce bits menos significativos del registro siempre están a cero para que siempre el directorio de páginas esté dentro de una página determinada. La operación de carga mediante la instrucción *MOV CR3, reg* o bien un cambio de tareas que implique un **cambio** de valor del registro CR3 hace que se eliminen las entradas del caché de la tabla de páginas.

Directorio de páginas

La longitud es de 4KB y permite hasta 1024 entradas. Estas entradas se seleccionan mediante los bits 31-22 de la dirección lineal (que viene de la unidad de segmentación). Cada entrada contiene la dirección e información adicional del siguiente nivel de tabla, la tabla de páginas, como se muestra a continuación:

31 - 12	**11 - 9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Dirección tabla de páginas (31-12)	Libre	0	0	D	A	0	0	U/S	R/W	P

Más abajo se explica el significado de estos campos.

Tabla de páginas

Las tablas de páginas tienen una longitud de 4KB y permiten hasta 1024 entradas. Estas entradas se seleccionan mediante los bits 21-12 de la dirección lineal. Cada entrada contiene la dirección inicial e información estadística de la página. Los 20 bits más significativos de la dirección de la página se concatenan con los 12 bits menos significativos de la dirección lineal para formar la dirección física. Las tareas pueden compartir tablas de páginas. Además el sistema operativo puede enviar una o más tablas al disco (si la memoria RAM está llena).

El formato de una entrada de la tabla de páginas es:

31 - 12	11 - 9	8	7	6	5	4	3	2	1	0
Dirección física de la página (31-12)	Libre	0	0	D	A	0	0	U/S	R/W	P

Más abajo se explica el significado de estos campos.

Entradas del directorio y las tablas de páginas

Los doce bits menos significativos contienen información estadística acerca de las tablas de páginas y las páginas respectivamente. El bit 0 (Presente) indica si la entrada se puede utilizar para traducir de dirección lineal a física. Si P = 0 no se puede, mientras que si P = 1 sí. Cuando P = 0 los otros 31 bits quedan libres para que los use el software. Por ejemplo, estos bits podrían utilizarse para indicar dónde se encuentra la página en el disco.

El bit 5 (Accedido), es puesto a uno por el 80386 en ambos tipos de entradas cuando ocurre un acceso de lectura o escritura en una dirección que esté dentro de una página cubierta por estas entradas. El bit 6 (Dirty), es puesto a uno por el microprocesador cuando ocurre un acceso de escritura. Los tres bits marcados como libre los puede utilizar el software.

Los bits 2 (Usuario/Supervisor) y 1 (Lectura/Escritura) se utilizan para proteger páginas individuales, como se muestra en el siguiente apartado.

Protección a nivel de página

Para la paginación existen dos niveles de protección: usuario que corresponde al nivel de privilegio 3 y supervisor que corresponde a los otros niveles: 0, 1 y 2. Los programas que se ejecutan en estos niveles no se ven afectados por este esquema de protección.

Los bits U/S y R/W se utilizan para proveer protección Usuario/Supervisor y Lectura/Escritura para páginas individuales o para todas las páginas cubiertas por una tabla de páginas. Esto se logra tomando los bits U/S y R/W más restrictivos (numéricamente inferior) entre el directorio de páginas y la tabla de páginas que corresponda a la página en cuestión.

Por ejemplo, si los bits U/S y R/W para la entrada del directorio de páginas valen 10, mientras que los correspondientes a la tabla de páginas valen 01, los derechos de acceso para la página será 01.

La siguiente tabla muestra la protección que dan estos bits:

U/S	R/W	Acceso permitido nivel 3	Acceso permitido niveles 0, 1, 2
0	0	Ninguno	Lectura/Escritura
0	1	Ninguna	Lectura/Escritura
1	0	Sólo lectura	Lectura/Escritura
1	1	Lectura/Escritura	Lectura/Escritura

Caché de entradas de tablas

Con el esquema visto hasta ahora, el procesador debe realizar dos accesos a tablas por cada referencia en memoria. Esto afectaría notablemente el rendimiento del procesador (por cada acceso indicado por el software habría que hacer tres). Para resolver este problema, el 80386 mantiene un caché con las páginas accedidas más recientemente. Este caché es el buffer de conversión por búsqueda (*TLB = Translation Lookaside Buffer*). El TLB es un caché asociativo de cuatro vías que almacena 32 entradas de tablas de páginas. Como cada página tiene una longitud de 4KB, esto alcanza a 128KB. Para muchos sistemas multitarea, el TLB tendrá un porcentaje de éxito (*hit*) del 98%. Esto significa que el procesador deberá acceder a las dos tablas el 2% del tiempo.

Operación

El hardware de paginación opera de la siguiente manera. La unidad de paginación recibe una dirección lineal de 32 bits procedente de la unidad de segmentación. Los 20 bits más significativos son comparados con las 32 entradas del TLB para determinar si la entrada de la tabla de páginas está en el caché. Si está (*cache hit*), entonces se calcula la dirección física de 32 bits y se la coloca en el bus de direcciones.

Si la entrada de la tabla de páginas no se encuentra en el TLB (*cache miss*), el 80386 leerá la entrada del directorio de páginas que corresponda. Si P=1 (la tabla de páginas está en memoria), entonces el 80386 leerá la entrada que corresponda de la tabla de páginas y pondrá a uno el bit Accedido de la entrada del directorio de páginas. Si P=1 en la entrada de la tabla de páginas indicando que la página se encuentra en memoria, el 80386 actualizará los bits Accedido y Dirty según corresponda y luego accederá a la memoria. Los 20 bits más significativos de la dirección lineal se almacenarán en el TLB para futuras referencias. Si P=0 para cualquiera de las dos tablas, entonces el procesador generará una excepción 14 (Fallo de Página).

El procesador también generará una excepción 14, si la referencia a memoria viola los atributos de protección de página (bits U/S y R/W) (por ejemplo, si el programa trata de escribir a una página que es de sólo lectura). En el registro CR2 se almacenará la dirección lineal que causó el fallo de página. Como la excepción 14 se clasifica como un fallo (es recuperable), CS:EIP apuntará a la instrucción que causó el fallo de página.

En la pila se pondrá un valor de 16 bits que sirve para que el sistema operativo sepa por qué ocurrió la excepción. El formato de esta palabra es:

- Bits 15-3: Indefinido.

- Bit 2: Vale 1 si el procesador estaba ejecutando en modo usuario. Vale 0 si el procesador estaba ejecutando en modo supervisor. Nótese que un acceso a una tabla de descriptores siempre se considera modo supervisor, por más que el programa se esté ejecutando en el nivel 3.
- Bit 1: Vale 1 si el procesador estaba por realizar una escritura. Vale 0 si estaba por realizar una lectura.
- Bit 0: Vale 1 si hubo una violación de protección en la página. Vale 0 si la página no estaba presente.

Modo virtual 8086

El 80386 permite la ejecución de programas para el 8086 tanto en modo real como en modo virtual 8086. De los dos métodos, el modo virtual es el que ofrece al diseñador del sistema la mayor flexibilidad. El modo virtual permite la ejecución de programas para el 8086 manteniendo el mecanismo de protección del 80386. En particular, esto permite la ejecución simultánea de sistemas operativos y aplicaciones para el 8086, y un sistema operativo 80386 corriendo aplicaciones escritas para el 80286 y el 80386. El escenario más común consiste en correr una o más aplicaciones de DOS simultáneamente mientras se corren programas escritos para Windows, todo al mismo tiempo.

Una de las mayores diferencias entre los modos real y protegido es cómo se interpretan los selectores de segmentos. Cuando el procesador ejecuta en modo virtual los registros de segmento se usan de la misma manera que en modo real. El contenido del registro de segmento se desplaza hacia la izquierda cuatro bits y luego se suma al offset para formar la dirección lineal.

El 80386 permite al sistema operativo especificar cuales son los programas que utilizan el mecanismo de direccionamiento del 8086, y los programas que utilizan el direccionamiento de modo protegido, según la tarea que pertenezcan (una tarea determinada corre en modo virtual o en modo protegido). Mediante el uso del paginado el espacio de direcciones de un megabyte del modo virtual se puede mapear en cualquier lugar dentro del espacio de direccionamiento de 4GB. Como en modo real, las direcciones efectivas (offsets) que superen los 64KB causarán una excepción 13.

El hardware de paginación permite que las direcciones lineales de 20 bits producidos por el programa que corre en modo virtual se dividan en 1MB/4(KB/página) = 256 páginas. Cada una de las páginas se pueden ubicar en cualquier lugar dentro del espacio de direcciones físicas de 4GB del 80386. Además, como el registro **CR3** (el registro que indica la base del directorio de páginas) se carga mediante un cambio de tareas, cada tarea que corre en modo virtual puede usar un método exclusivo para realizar la correspondencia entre las páginas y las direcciones físicas. Finalmente, el hardware de paginado permite compartir el código del sistema operativo que corre en 8086 entre múltiples aplicaciones que corren en dicho microprocesador.

Todos los programas que corren en el modo virtual 8086 se ejecutan en el nivel de privilegio 3, el nivel de menor privilegio, por lo que estos programas se deben sujetar a todas las verificaciones de protección que ocurren en modo protegido. En el modo real, los programas corren en el nivel de privilegio cero, el nivel de mayor privilegio.

Cuando una tarea corre en modo virtual, todas las interrupciones y excepciones realizan un cambio de nivel de privilegio hacia el nivel cero, donde corre el sistema operativo 80386.Dicho sistema operativo puede determinar si la interrupción vino de una tarea corriendo en modo virtual o en modo protegido examinando el bit **VM** de la imagen de los indicadores en la pila (el sistema operativo no debe leer directamente el bit **VM** puesto que se pone a cero automáticamente al entrar a la rutina de atención de

interrupción).

Para entrar al modo virtual 8086, habrá que ejecutar una instrucción **IRET** cuando CPL = 0 y cuya imagen de los indicadores en la pila tenga el bit **VM** a uno. Otra posibilidad consiste en ejecutar un cambio de tarea hacia una tarea cuyo TSS tipo 386 tenga el bit **VM** a uno en la imagen de los indicadores.

Vectores de interrupción predefinidos

Las siguientes son las excepciones que puede generar el microprocesador, por lo que, en modo real, deberán estar los vectores correspondientes (en la zona baja de memoria) que apunten a los manejadores, mientras que en modo protegido, deberán estar los descriptores de compuerta correspondientes en la tabla de descriptores de interrupción (**IDT**).

Clase de excepción	Tipo	Instrucción que la causa	El manejador retorna a dicha instrucción
Error de división	0	DIV, IDIV	SÍ
Excepción de depuración	1	Cualquier instrucción	
Interrupción NMI	2	INT 2 o NMI	NO
Interrupción de un byte	3	INT 3	NO
Sobrepasamiento	4	INTO	NO
Fuera de rango	5	BOUND	SÍ
Código inválido	6	Instrucción ilegal	SÍ
El dispositivo no existe	7	ESC, WAIT	SÍ
Doble falta	8	Cualquier instrucción que pueda generar una excepción	
TSS inválido	10	JMP, CALL, IRET, INT	SÍ
Segmento no presente	11	Carga de registro de segmento	SÍ
Falta de pila	12	Referencia a la pila	SÍ
Violación de protección	13	Referencia a memoria	SÍ
Falta de página	14	Acceso a memoria	SÍ
Error del coprocesador	16	ESC, WAIT	SÍ
Reservado	17-32		
Interrupción de 2 bytes	0-255	INT n	NO

Nuevas instrucciones del 80386

Aparte de las instrucciones del 8088, y las nuevas del 80186 y del 80286, el 80386 tiene las siguientes nuevas instrucciones:

BSF *dest, src* (Bit Scan Forward): Busca el primer bit puesto a 1 del operando fuente *src* (comenzando

por el bit cero hasta el bit n-1). Si lo encuentra pone el indicador ZF a 1 y carga el destino *dest* con el índice a dicho bit. En caso contrario pone ZF a 0.

BSR *dest, src* (Bit Scan Reverse): Busca el primer bit puesto a 1 del operando fuente *src* (comenzando por el bit n-1 hasta el bit cero). Si lo encuentra pone el indicador ZF a 1 y carga el destino *dest* con el índice a dicho bit. En caso contrario pone ZF a 0.

BT *dest, src* (Bit Test): El bit del destino *dest* indexado por el valor fuente se copia en el indicador CF.

BTC *dest, src* (Bit Test with Complement): El bit del destino *dest* indexado por el valor fuente se copia en el indicador CF y luego se complementa dicho bit.

BTR *dest, src* (Bit Test with Reset): El bit del destino *dest* indexado por el valor fuente *src* se copia en el indicador CF y luego pone dicho bit a cero.

BTS *dest, src* (Bit Test with Set): El bit del destino *dest* indexado por el valor fuente *src* se copia en el indicador CF y luego pone dicho bit a uno.

CDQ (Convert Doubleword to Quadword): Convierte el número signado de 4 bytes en **EAX** en un número signado de 8 bytes en **EDX:EAX** copiando el bit más significativo (de signo) de **EAX** en todos los bits de **EDX**.

CWDE (Convert Word to Extended Doubleword): Convierte una palabra signada en el registro **AX** en una doble palabra signada en **EAX** copiando el bit de signo (bit 15) de **AX** en los bits 31-16 de **EAX**.

JECXZ *label* (Jump on ECX Zero): Salta a la etiqueta *label* si el registro **ECX** vale cero.

LFS, **LGS**, **LSS** *dest, src* (Load pointer using **FS**, **GS**, **SS**): Carga un puntero de 32 bits de la memoria *src* al registro de uso general destino *dest* y **FS**, **GS** o **SS**. El offset se ubica en el registro destino y el segmento en **FS**, **GS** o **SS**. Para usar esta instrucción la palabra en la dirección indicada por *src* debe contener el offset, y la palabra siguiente debe contener el segmento. Esto simplifica la carga de punteros lejanos (*far*) de la pila y de la tabla de vectores de interrupción.

MOVSX *dest, src* (Move with Sign eXtend): Copia el valor del operando fuente *src* al registro destino *dest* (que tiene el doble de bits que el operando fuente) extendiendo los bits del resultado con el bit de signo. Se utiliza para aritmética signada (con números positivos y negativos).

MOVZX *dest, src* (Move with Zero eXtend): Copia el valor del operando fuente *src* al registro destino *dest* (que tiene el doble de bits que el operando fuente) extendiendo los bits del resultado con ceros. Se utiliza para aritmética no signada (sin números negativos).

POPAD (Pop All Doubleword Registers): Retira los ocho registros de uso general de 32 bits de la pila en el siguiente orden: EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX. El valor de ESP retirado de la pila se descarta.

POPFD (Pop Doubleword Flags): Retira de la pila los indicadores completos (32 bits).

PUSHAD (Push All Doubleword Registers): Pone los ocho registros de uso general de 32 bits en la pila en el siguiente orden: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI.

PUSHFD (Push Doubleword Flags): Pone los 32 bits del registro de indicadores en la pila.

SETcc *dest* (Set Byte on Condition cc): Pone el byte del destino *dest* a uno si se cumple la condición, en caso contrario lo pone a cero. Las condiciones son las mismas que para los saltos condicionales (se utilizan las mismas letras que van después de la "J").

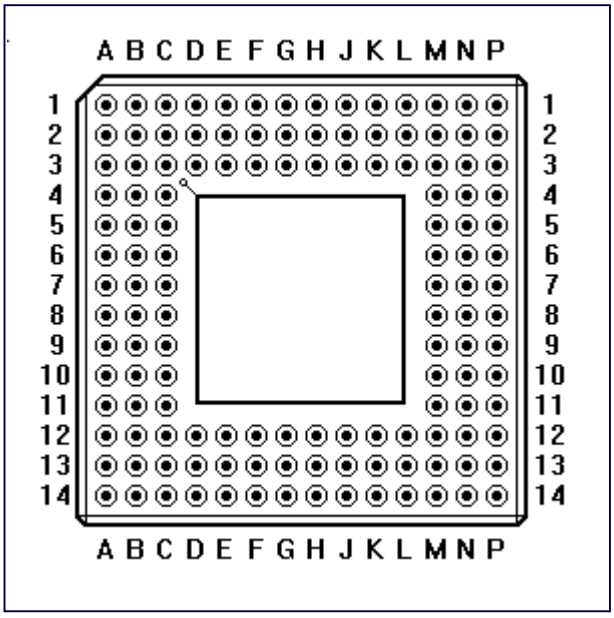
SHLD *dest, src, count* (Shift Left Double precision): Desplaza *dest* a la izquierda *count* veces y las posiciones abiertas se llenan con los bits más significativos de *src*.

SHRD *dest, src, count* (Shift Left Double precision): Desplaza *dest* a la derecha *count* veces y las posiciones abiertas se llenan con los bits menos significativos de *src*.

Hardware del microprocesador 80386

Terminales del 80386 DX

El 80386 DX está encapsulado en el formato PGA (Pin Grid Array) de 132 terminales. La distancia entre los terminales es de 0,1 pulgadas (2,54 milímetros). Los terminales se nombran mediante una letra y un número, como se puede apreciar en el siguiente gráfico:



En este caso el 80386 se ve desde la cara donde asoman los terminales (desde "abajo"). Nótese que al lado del pin A1 la diagonal que hace el borde es más pronunciada que en las otras tres esquinas. Esto sirve para la identificación mecánica del circuito integrado. La cápsula tiene forma cuadrada de 1,45 pulgadas (36,802 milímetros) de lado. En los terminales C3, C12, M3 y M12 hay una saliente en los costados para que no se ingrese el circuito integrado hasta el fondo del zócalo.

La distribución de los terminales según grupos funcionales es la siguiente (nota: si una señal tiene # al final indica que se activa cuando está en estado bajo):

Alimentación

El 80386 está implementado mediante la tecnología CHMOS III y tiene requerimientos modestos de potencia. Sin embargo, su alta frecuencia de operación y 72 buffers de salida (dirección, datos, control y HLDA) puede causar picos de potencia cuando los distintos buffers cambian de nivel (conmutan) simultáneamente. Para una distribución limpia de potencia, existen 20 terminales de **Vcc** (positivo) y 21 de **Vss** (referencia) que alimentan a las diferentes unidades funcionales del 80386. Los terminales de **Vcc** son: A1, A5, A7, A10, A14, C5, C12, D12, G2, G3, G12, G14, L12, M3, M7, M13, N4, N7, P2 y P8. Los terminales de **Vss** son: A2, A6, A9, B1, B5, B11, B14, C11, F2, F3, F14, J2, J3, J12, J13, M4, M8, M10, N3, P6, P14.

Las conexiones de potencia y masa se deben realizar a todos los terminales externos mencionados más arriba. En el circuito impreso, todos los terminales de **Vcc** deben conectarse a un plano de Vcc, mientras que los terminales de **Vss** deben ir al plano de GND.

Debe haber buenos capacitores de desacople cerca del 80386. El 80386 controlando los buses de datos y dirección puede causar picos de potencia, particularmente cuando se manejan grandes cargas capacitivas. Se recomiendan capacitores e interconexiones de baja inductancia para un mejor rendimiento eléctrico. La inductancia se puede reducir acortando las pistas del circuito impreso entre el 80386 y los capacitores de desacople tanto como sea posible.

Señal de reloj

La señal *clock* **CLK2** provee la temporización para el 80386. Se divide por dos internamente para generar el reloj interno del microprocesador que se utiliza para la ejecución de las instrucciones. El reloj interno posee dos fases: "fase uno" y "fase dos". Cada período de **CLK2** es una fase del reloj interno. Si se desea, la fase del reloj interno se puede sincronizar a una fase conocida aplicando la señal de RESET con los tiempos que se indican en el manual del circuito integrado. El terminal correspondiente es el F12.

Bus de datos

Bus de datos (**D0 - D31**): Estas señales bidireccionales proveen el camino de los datos de propósito general entre el 80386 y los otros dispositivos. Las entradas y salidas del bus de datos indican "1" cuando están en estado alto (lógica positiva). El bus de datos puede transferir datos en buses de 32 ó 16 bits utilizando una característica especial de este chip controlada por la entrada BS16#. Durante cualquier operación de escritura y durante los ciclos de halt (parada) y shutdown (apagado), el 80386 siempre maneja las 32 señales del bus de datos aunque el tamaño del bus (según las entrada BS16#) sea de 16 bits.

Los terminales son los siguientes:

x	0	1	2	3	4	5	6	7	8	9
D0x	H12	H13	H14	J14	K14	K13	L14	K12	L13	N14
D1x	M12	N13	N12	P13	P12	M11	N11	N10	P11	P10
D2x	M9	N9	P9	N8	P7	N6	P5	N5	M6	P4
D3x	P3	M5								

Bus de direcciones

Bus de direcciones (**A2 - A31**): Estas salidas de tres estados proveen las direcciones de memoria y de los puertos de entrada/salida. El bus de direcciones es capaz de direccionar 4 gigabytes de espacio de memoria física (00000000h-FFFFFFFFh) y 64 kilobytes de espacio de entrada/salida (00000000-0000FFFFh) para E/S programada. Las transferencias de E/S generadas automáticamente para la comunicación entre el 80386 y el coprocesador utilizan las direcciones 800000F8h-800000FFh, así que **A31** en estado alto junto con M/IO# en estado bajo proveen la señal de selección del coprocesador.

Las salidas de habilitación de byte (**BE0#** a **BE3#**), indican directamente cuáles bytes del bus de datos de 32 bits son los que realmente se utilizan en la transferencia. Esto es lo más conveniente para el hardware externo.

- BE0# se aplica a D0-D7
- BE1# se aplica a D8-D15
- BE2# se aplica a D16-D23
- BE3# se aplica a D24-D31

La cantidad de habilitaciones de byte activos indican el tamaño físico del operando que se está transfiriendo (1, 2, 3 ó 4 bytes).

Cuando ocurre un ciclo de escritura de memoria o de E/S y el operando que se transfiere ocupa sólo los 16 bits más significativos del bus de datos (D16-D31), los datos duplicados se presentan simultáneamente en los 16 bits menos significativos (D0-D15). Se realiza esta duplicación para lograr un rendimiento óptimo en buses de 16 bits. El patrón de los datos que se repiten depende de las habilitaciones de byte activos en el ciclo de escritura:

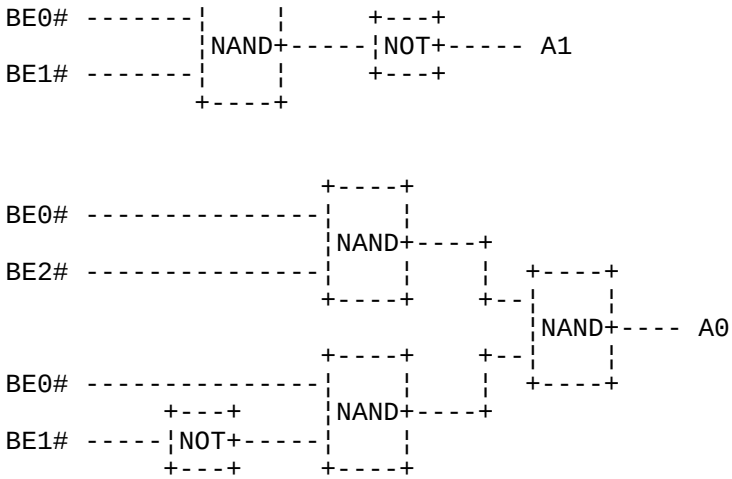
Habilitaciones de byte				Datos que se escriben				Duplicación
BE3#	BE2#	BE1#	BE0#	D24-D31	D16-D23	D8-D15	D0-D7	automática
Alto	Alto	Alto	Bajo	indef	indef	indef	A	no
Alto	Alto	Bajo	Alto	indef	indef	B	indef	no
Alto	Bajo	Alto	Alto	indef	C	indef	C	sí
Bajo	Alto	Alto	Alto	D	indef	D	indef	sí
Alto	Alto	Bajo	Bajo	indef	indef	B	A	no
Alto	Bajo	Bajo	Alto	indef	C	B	indef	no
Bajo	Bajo	Alto	Alto	D	C	D	C	sí
Alto	Bajo	Bajo	Bajo	indef	C	B	A	no
Bajo	Bajo	Bajo	Alto	D	C	B	indef	no
Bajo	Bajo	Bajo	Bajo	D	C	B	A	no

donde: D = Datos correspondientes a las posiciones D24-D31
C = Datos correspondientes a las posiciones D16-D23
B = Datos correspondientes a las posiciones D8-D15
A = Datos correspondientes a las posiciones D0-D7

Los terminales son los siguientes:

x	0	1	2	3	4	5	6	7	8	9
A0x			C4	A3	B3	B2	C3	C2	C1	D3
A1x	D2	D1	E3	E2	E1	F1	G1	H1	H2	H3
A2x	J1	K1	K2	L1	L2	K3	M1	N1	L3	M2
A3x	P1	N2								
BEx#	E12	C13	B13	A13						

Generación de A1 y A0 a partir de BE0#-BE3#:



Bus de Control

Señales de definición del ciclo de bus

Son cuatro: **W/R#**, **D/C#**, **M/IO#**, **LOCK#**). Estas salidas de tres estados definen el tipo de ciclo de bus que se está realizando. **W/R#** distingue entre ciclos de escritura y lectura, **D/C#** distingue entre ciclos de datos y de control, **M/IO#** distingue entre ciclos de memoria y de entrada/salida y **LOCK#** distingue entre ciclos de bus donde se bloquea (*lock*) o no el acceso a otros manejadores de bus (*bus masters*).

Las tres señales más importantes para definir el ciclo de bus son los tres primeros, ya que ésas son las señales que se activan cuando se activa la salida ADS# (*ADdress Status output*). La señal **LOCK#** se activa en el comienzo del primer ciclo de bloqueo, que, debido al *pipelining*, puede ocurrir después que se activó ADS#. La salida **LOCK#** se desactiva cuando se termina el último ciclo de bus bloqueado.

La siguiente tabla ocurre cuando se activa ADS# (ADS# = 0):

M/IO#	D/C#	W/R#	Tipo de ciclo de bus	Bloqueado
Bajo	Bajo	Bajo	Reconocimiento de interrupción	sí
Bajo	Bajo	Alto	No ocurre	--
Bajo	Alto	Bajo	Lectura de área de E/S	no
Bajo	Alto	Alto	Escritura en área de E/S	no
Alto	Bajo	Bajo	Lectura de código (instrucciones)	no
Alto	Bajo	Alto	Parada: Dirección = 2 (BE0# Alto, BE1# Alto, BE2# Bajo, BE3# Alto, A2-A31 Bajo)	no
			Apagado: Dirección = 0 (BE0# Bajo, BE1# Alto, BE2# Alto, BE3# Alto, A2-A31 Bajo)	
Alto	Alto	Bajo	Lectura de datos de memoria	algunos ciclos
Alto	Alto	Alto	Escritura de datos en memoria	

Señales de control del bus

Las siguientes señales permiten que el procesador indique cuando comienza un ciclo y permite que otro hardware controle el *pipelining* de direcciones, el ancho del bus de datos y la finalización del ciclo del bus.

Estado de las direcciones (**ADS#**)
: Esta salida triestado indica que se está enviando por los pines del 80386 las señales de definición del ciclo de bus y las direcciones (W/R#, D/C#, M/IO#, BE0#-BE3# y A2-A31). Es el terminal E14 en la figura que aparece más arriba

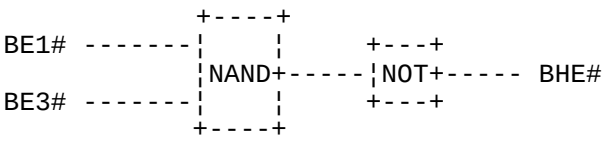
Reconocimiento de transferencia (**READY#**)
: Esta entrada indica que el ciclo de bus actual está completo, y que los bytes indicados por BE0# a BE3# y BS16# son aceptados (caso de escritura) o entregados (caso de lectura). Si READY# se encuentra activo cuando lo muestrea el 80386 durante una lectura, el microprocesador guarda el dato y da por terminado el ciclo de bus, mientras que si lo encuentra activo en un ciclo de escritura, el procesador termina el ciclo de bus. La señal READY# se ignora en el primer estado del ciclo de bus (se muestrea a partir del segundo estado). READY# debe activarse para reconocer cualquier ciclo de bus (como aparece en la tabla que figura más arriba), incluyendo la indicación de parada (halt) y de apagado (shutdown). Este es el pin G13.

Pedido de la siguiente dirección (**NA#**)

Esta señal se utiliza para pedir *pipelining* de direcciones. Indica que el sistema está preparado para aceptar nuevos valores de BE0#-BE3#, A2-A31, W/R#, D/C# y M/IO# del 80386 aunque no se haya reconocido el final del presente ciclo de bus mediante el terminal READY#. Si esta entrada está activa cuando el 80386 pide una muestra de esta señal, se envía la siguiente dirección por el bus. El terminal correspondiente es el D13.

Tamaño del bus de 16 bits (**BS16#**)

Esta señal permite que el 80386 se pueda conectar tanto a buses de 32 bits como de 16 bits. Activando esta señal hace que el ciclo actual de bus utilice la mitad menos significativa (D0-D15) del bus de datos, correspondiente a BE0# y BE1#. No tiene ningún efecto si solamente BE0# y/o BE1# se activan en el ciclo. Sin embargo, durante los ciclos de bus donde se tendrían que activar BE2# y/o BE3#, la activación de **BS16#** hará que el 80386 realice los ajustes necesarios para la transferencia correcta de los bytes superiores usando solamente las señales D0-D15. Si el operando ocupa ambas mitades del bus de datos y se activa **BS16#**, el 80386 realizará otro ciclo de bus de 16 bits automáticamente. Se requieren algunas compuertas externas para generar las señales A1, BHE y BLE, que son las señales de los procesadores de 16 bits para la generación de direcciones (BHE = Byte High Enable, BLE = Byte Low Enable). Las compuertas necesarias para generar A1 y BLE# (o A0) se muestran más arriba, mientras que para generar BHE# se utiliza:



El terminal correspondiente a BS16# es el C14.

Señales para arbitrar el bus

Aquí se describe el mecanismo por el cual el procesador cede el control de su bus local cuando se lo pide otro manejador de bus ("bus master").

Pedido de obtención del bus (**HOLD**)

Esta entrada indica que algún dispositivo aparte del 80386 necesita el control del bus. **HOLD** debe estar activo todo el tiempo en que otro dispositivo maneje el bus local. **HOLD** no se reconoce mientras está activa la señal RESET. Si se activa RESET cuando **HOLD** también lo está, RESET tiene prioridad y pone el bus en estado inactivo, en vez que en reconocimiento del pedido de obtención del bus, que se manifiesta por estar todas las salidas excepto HLDA en alta impedancia. **HOLD** es sensible al nivel y es una entrada sincrónica (depende de las fases del reloj). El terminal correspondiente a **HOLD** es el D14.

Reconocimiento del pedido de obtención del bus (**HLDA**)

El 80386 activa esta salida cuando ha cedido el control del bus local en respuesta a la entrada HOLD y se genera el estado de reconocimiento del pedido de obtención del bus. Este estado ofrece una aislación casi completa de las señales. **HLDA** es la única señal que genera el microprocesador. Las otras señales de salida o bidireccionales (D0-D31, BE0#-BE3#, A2-A31, W/R#, D/C#, M/IO#, LOCK# y ADS#) se ponen en el estado de alta impedancia así el otro manejador de bus puede controlarlos. Debido a esto, se recomienda poner resistores de pull-up de 20 Kohm en algunas señales (ADS# y LOCK#) para prevenir actividad espúrea cuando ningún manejador de bus (80386 o un dispositivo externo) genera la señal. Las entradas ERROR#, BUSY# y BS16# tienen una resistencia interna de pull-up de 20 Kohm aproximadamente, mientras que la entrada PEREQ tiene

una resistencia de pull-down del mismo valor. El terminal correspondiente a **HLDA** es el M14.

Señales para la interfaz con el coprocesador

Aquí se describen las señales dedicadas a la interfaz con el coprocesador numérico. El 80386 se puede conectar tanto con el 80287 como con el 80387.

Pedido del coprocesador (**PEREQ**)
: Cuando está activa, esta señal de entrada indica un pedido del coprocesador para que se transfiera un dato (operando) a/desde la memoria por el 80386. En respuesta, el 80386 transfiere la información entre el coprocesador y la memoria. Como el 80386 tiene almacenado internamente el código de operación del coprocesador que se está ejecutando, puede realizar la transferencia pedida en la dirección de memoria correcta y en el sentido correcto (lectura o escritura de memoria). El terminal correspondiente a **PEREQ** es el C8.

Coprocesador ocupado (**BUSY#**)
: Cuando está activa, esta entrada indica que el coprocesador está ejecutando una instrucción y que no puede aceptar otra. Cuando el 80386 encuentra cualquier instrucción del coprocesador que opera en la pila de números (es decir, introducir o extraer de la pila, o instrucciones aritméticas) o la instrucción WAIT, se muestrea automáticamente esta entrada hasta que esté negada (estado alto). Esto permite que el coprocesador pueda terminar la ejecución de la instrucción en curso antes de que se le envíe la siguiente instrucción. Se permite la ejecución de las instrucciones FNINIT y FNCLEX aunque el coproceador esté ocupado, ya que estas instrucciones se usan para la inicialización y el borrado de las excepciones. **BUSY#** tiene una función adicional. En el flanco descendente de RESET se muestrea la señal. Si está en estado bajo, el 80386 realiza un test interno. Si está en estado alto, no ocurre el test. El terminal correspondiente a **BUSY#** es el B9.

Error del coprocesador (**ERROR#**)
: Esta señal de entrada indica que la última instrucción del coprocesador generó un error de un tipo que no estaba enmascarado mediante el registro de control del coprocesador. Esta entrada se muestrea automáticamente cuando se encuentra una instrucción de coprocesador, y si está activa, el 80386 genera la excepción 16 para acceder el software que maneja el error. Algunas instrucciones del coprocesador, generalmente aquéllos que limpian los indicadores de errores numéricos o que salvan el estado del coprocesador, se ejecutan sin que el 80386 genere la excepción 16 aunque ERROR# esté activo. Estas instrucciones son: FNINIT, FNCLEX, FSTSW, FSTSW AX, FSTCW, FSTENV, FSAVE, FESTENV y FESAVE. **ERROR#** sirve para una función adicional. Si **ERROR#** está en estado bajo luego de 20 períodos de CLK2 después del flanco descendente de RESET y se mantiene en estado bajo por lo menos hasta que el 80386 comienza su primer ciclo, el microprocesador asume que hay un 80387 presente (pone el bit ET del registro CR0 a uno). De otra manera, supone que hay un 80287 o no hay coprocesador (pone dicho bit a cero). Por lo tanto, debe realizarse un test por software para distinguir entre un 80287 y la ausencia de coprocesador (en el último caso el programa deberá poner EM = 1). El terminal correspondiente a **ERROR#** es el A8.

Señales de interrupción

Las siguientes descripciones cubren entradas que pueden interrumpir o suspender la ejecución de la secuencia de instrucciones del microprocesador.

Pedido de interrupción enmascarable (**INTR**)
: Cuando está activa, esta entrada indica un pedido de servicio de interrupción, que puede ser

enmascarado mediante el indicador IF (bit 9 del registro de indicadores). Cuando el 80386 responde a la entrada **INTR**, realiza dos ciclos de reconocimiento de interrupción, y al final del segundo, guarda un vector de ocho bits para identificar la fuente que la circuitería externa debe ubicar en D0-D7. El estado de A2 diferencia entre el primero y el segundo ciclo: en el primero, la dirección que aparece en el bus es 4 (A31-A3 en estado bajo, A2 en estado alto, BE3#-BE1# en estado alto y BE0# en estado bajo), mientras que en el segundo es cero (A31-A2 bajo, BE3#-BE1# alto, BE0# bajo). **INTR** es sensible por nivel y se permite que sea asincrónica con respecto a la señal CLK2. Para asegurar que se reconozca **INTR**, la señal debe estar activa hasta que se produzca el primer ciclo de reconocimiento de interrupción. El terminal correspondiente a **INTR** es el B7.

Pedido de interrupción no enmascarable (**NMI**)

Esta entrada indica un pedido de servicio de interrupción que no puede ser enmascarado por software. El pedido se procesa utilizando el puntero o la compuerta en el elemento 2 de la tabla de interrupciones. Debido a que el vector es fijo, no se realizan ciclos de reconocimiento de interrupciones cuando se procesa NMI. **NMI** es sensible al flanco ascendente y se permite que se asincrónico con respecto a la señal CLK2. Para asegurar que se reconozca **NMI**, debe negarse por lo menos durante ocho períodos de CLK2 y luego activarse por lo menos ocho períodos. Una vez que comenzó el procesamiento de **NMI**, no se procesarán **NMI** posteriores hasta después de la siguiente instrucción IRET, que está típicamente al final de la rutina de interrupción. Si se activa otra vez **NMI** antes de ese momento, se "recordará" un flanco ascendente de **NMI** para su posterior procesamiento después que ocurra la instrucción IRET. El terminal correspondiente a **NMI** es el B8.

Inicialización del microprocesador (**RESET**)

El 80386 se inicializa activando **RESET** por 15 o más períodos de CLK2 (80 o más perídos de CLK2 antes de requerir el test de sí mismo). Cuando se activa **RESET**, se suspende cualquier operación que se esté realizando, las entradas se ignoran, y los otros terminales del bus se ponen en el estado indicado en la siguiente tabla:

Nombre del terminal	Nivel de la señal durante RESET
ADS#	Alto
D0-D31	Alta impedancia
BE0#-BE3#	Bajo
A2-A31	Alto
W/R#	Bajo
D/C#	Alto
M/IO#	Bajo
LOCK#	Alto
HLDA	Bajo

Si **RESET** y HOLD se activan simultáneamente, **RESET** tiene prioridad aunque el 80386 estuviera en el estado de reconocimiento de pedido de obtención del bus (HOLD Acknowledge) antes de que se active **RESET**.

RESET se activa por nivel y debe ser sincrónico con respecto a la señal CLK2. Si se desea, se puede lograr que el reloj interno del microprocesador se pueda sincronizar con la circuitería externa si se asegura que el flanco descendente de **RESET** cumple con condiciones apropiadas.

Si se pidió un test interno, el valor de EAX al ejecutarse la primera instrucción debe ser cero si el 80386 está bien. Otro valor indica que el chip está fallado. Si no se mantuvo **RESET** en estado alto durante el mínimo de 80 ciclos de CLK2, el valor que entrega el test en EAX puede ser distinto de cero aunque el 80386 esté bien.

Los otros registros después de **RESET** tienen los siguientes valores: IP = 0000FFF0h, CS = F000, DS = ES = FS = GS = SS = 0000h, DH = Identificador de componente (vale 3 para indicar 80386), DL = Identificador de revisión. Además todos los bits definidos del registro de indicadores y del registro CR0 se ponen a cero. El valor inicial de los otros registros no está definido.

El terminal correspondiente a **RESET** es el C9.

Terminales sin conexión

El 80386 posee ocho terminales que no están conectados interiormente (llamados **N.C.**). Deben quedar desconectados exteriormente. Estos terminales son: A4, B4, B6, B12, C6, C7, E13 y F13.

El coprocesador matemático 80387

Introducción

La interfaz en los sistemas 80386/80387 es muy similar a la de los sistemas 80286/80287. Sin embargo, para prevenir la corrupción de datos del coprocesador debido a errores de los programas que corren en la CPU, se utilizan los puertos de entrada/salida 800000F8h-800000FFh que no son accesibles a los programas. La comunicación ha sido optimizada ya que, al utilizar transferencias de 32 bits, se necesitan de 14 a 20 ciclos de reloj. La única manera de eliminar esta pérdida de tiempo consiste en integrar el coprocesador y la CPU en un único chip, como se hizo posteriormente en el microprocesador 80486.

Versiones del 80387

El **80387** fue la primera generación de coprocesadores específicamente diseñados para la CPU 80386. Fue introducido en 1986, un año después que el CPU 80386.

El 80387 fue superado por el **387DX**, que fue introducido en 1989. El viejo 80387 era 20% más lento que el 387DX. El 80387 estába empaquetado en el formato PGA de 68 pines y estaba manufacturado con la tecnología CHMOS III de 1,5 micrones. La máxima velocidad del 80387 fue de 20 MHz. El 387DX es la segunda generación. Esta versión está realizada en un proceso CMOS más avanzado (tecnología CHMOS IV) que permite una frecuencia de 33 MHz. Algunas instrucciones se han mejorado mucho más que el 20% de promedio. Por ejemplo, la instrucción FBSTP es 3,64 veces más rápido que en el 80387.

El **387SX** es el coprocesador que se aparea con el 386SX con un bus de datos de 16 bits, en vez de los 32 que tienen los anteriores. El 387SX tiene la misma unidad de ejecución que el 80387 original. Viene en formato PLCC (*Plastic Leaded Chip Carrier*) de 68 pines con una frecuencia máxima de 20 MHz.

El **387SL** (que se introdujo en 1992) se diseñó para ser utilizado en sistemas junto con el 386SL en notebooks y laptops. Está realizado con la tecnología CHMOS IV estática (no se pierden los datos internos si se detiene el reloj) y tiene la misma unidad de ejecución que el 387DX.

Nuevas instrucciones del 80387

FSIN: Calcula el seno del valor en ST. El resultado reemplaza el valor anterior de ST.

FCOS: Calcula el coseno del valor en ST. El resultado reemplaza el valor anterior de ST.

FSINCOS: Calcula el seno y el coseno del valor en ST. Cuando se completa la instrucción, el valor de ST es el coseno del ST original, mientras que en ST(1) se encuentra el seno. Como se realiza una introducción en la pila, el valor que antes se encontaba en ST(1) ahora estará en ST(2).

FUCOM *ST(i)*: Realiza una comparación entre el operando y ST. Si no se especifica parámetro se asume **ST(1)**. Actualiza los indicadores de punto flotante como sigue:

Relación	C3, C2, C0
No comparables	111
ST > Fuente	000
ST < Fuente	001
ST = Fuente	100

FUCOMP *ST(i)*: Hace lo mismo que **FUCOM** *ST(i)* y luego elimina el elemento que está en el tope de la pila ST.

FUCOMPP: Hace lo mismo que **FUCOM** *ST(1)* y luego retira los dos elementos de la pila.

FPREM1: Es similar al instrucción FPREM del coprocesador 8087, pero cumple con la norma IEEE 754.

Las instrucciones FLENV y F[N]STENV necesitan ahora un espacio de 18 bytes de memoria y FRSTOR y F[N]SAVE necesitan 98 bytes en memoria. Esto se debe a que las direcciones que maneja el 80387 son de 32 bits (los anteriores manejan direcciones de 16 bits).

El microprocesador 80486

Bloques que componen el 80486

Este microprocesador es básicamente un 80386 con el agregado de una unidad de punto flotante compatible con el 80387 y un caché de memoria de 8 KBytes. Por lo tanto los bloques que componen el 80486 son los siguientes:

1. **Unidad de ejecución**: Incluye los registros de uso general de 32 bits, la unidad lógico-matemática y un *barrel shifter* de 64 bits. La unidad de ejecución está mejorada con lo que se necesita un sólo ciclo de reloj para las instrucciones más frecuentes.

2. **Unidad de segmentación**: Incluye los registros de segmento, los cachés de información de descriptores y la lógica de protección. No tiene diferencias con respecto al 80386.

3. **Unidad de paginación**: Es la encargada de traducir las direcciones lineales (generadas por la unidad anterior) en direcciones físicas. Incluye el buffer de conversión por búsqueda (TLB). Los últimos modelos (DX4, algunos DX2) soportan páginas de 4MB aparte de las de 4KB del 80386.

4. **Unidad de caché**: La evolución de las memorias hizo que el tiempo de acceso de las mismas decrecieran lentamente, mientras que la velocidad de los microprocesadores aumentaba exponencialmente. Por lo tanto, el acceso a memoria representaba el cuello de botella. La idea del caché es tener una memoria relativamente pequeña con la velocidad del microprocesador. La mayoría del código que se ejecuta lo hace dentro de ciclos, con lo que, si se tiene el ciclo completo dentro del caché, no sería necesario acceder a la memoria externa. Con los datos pasa algo similar: también ocurre un efecto de localidad. El caché se carga rápidamente mediante un proceso conocido como "ráfaga", con el que se pueden transferir cuatro bytes por ciclo de reloj. Más abajo se da información más detallada de esta unidad.

5. **Interfaz con el bus**: Incluye los manejadores del bus de direcciones (con salidas de A31-A2 y BE0# a BE3# (mediante esto último cada byte del bus de datos se habilita por separado)), bus de datos de 32 bits y bus de control.

6. **Unidad de instrucciones**: Incluye la unidad de prebúsqueda que le pide los bytes de instrucciones al caché (ambos se comunican mediante un bus interno de 128 bits), una cola de instrucciones de 32 bytes, la unidad de decodificación, la unidad de control, y la ROM de control (que indica lo que deben hacer las instrucciones).

7. **Unidad de punto flotante**: Incluye ocho registros de punto flotante de 80 bits y la lógica necesaria para realizar operaciones básicas, raíz cuadrada y trascendentes de punto flotante. Es tres o cuatro veces más rápido que un 386DX y 387DX a la misma frecuencia de reloj. Esta unidad no está incluida en el modelo 486SX.

Unidad de caché

Estos procesadores tienen un caché interno que almacena 8KB de instrucciones y datos excepto el DX4 y el *Write-back enhanced* DX4 que tienen 16KB de caché interno. El caché aumenta el rendimiento del sistema ya que las lecturas se realizan más rápido desde el caché que desde la memoria externa. Esto también reduce el uso del bus externo por parte del procesador. Éste es un caché de primer nivel (también

llamado **L1**).

El procesador 80486 puede usar un caché de segundo nivel (también llamado **L2**) fuera del chip para aumentar aún más el rendimiento general del sistema.

Si bien la operación de estos cachés internos y externos son transparentes a la ejecución de los programas, el conocimiento de su funcionamiento puede servir para optimizar el software.

El caché está disponible en todos los modos de funcionamiento del procesador: modo real, modo protegido y modo de manejo del sistema.

Funcionamiento

El caché es una memoria especial, llamada **memoria asociativa**. Dicha memoria tiene, asociado a cada unidad de memoria, un *tag*, que almacena la dirección de memoria que contiene los datos que están en la unidad de memoria. Cuando se desea leer una posición de memoria mediante esta memoria asociativa, se comparan todos los tags con esta dirección. Si algún tag tiene esta dirección, se dice que hubo un **acierto** (*cache hit* en inglés) con lo que se puede leer la información asociada a ese tag. En caso contrario hay un **fallo** (*cache miss* en inglés), con lo que hay que perder un ciclo de bus para leer el dato que está en memoria externa.

En el caso del 80486, cada unidad de memoria son 16 bytes. Esta cantidad es una **línea del caché**. Las líneas pueden ser **válidas** (cuando contienen datos de la memoria principal) o **inválidas** (en este caso la línea no contiene información útil). Como el caché se llena por líneas completas (comenzando por direcciones múltiplos de 16), hay que tratar de no leer posiciones aleatorias de la memoria, ya que en este caso, si se leen bytes en posiciones alejadas unas de otras, el procesador usará cuatro ciclos de bus para leer 16 bytes (para llenar una línea) por cada byte que deseamos leer. Esto no es problema para el código o la pila (*stack*) ya que éstos se acceden generalmente de manera secuencial.

Hay dos clases de cachés: **write-through** y **write-back** (*retroescritura*)(implementado solamente en los modelos *write-back enhanced* DX2 y *write-back enhanced* DX4). La diferencia entre las dos radica en el momento de escritura. Las primeras siempre escriben en la memoria principal, mientras que las otras sólo escriben cuando se llena el caché y hay que desocupar una línea. Esto último aumenta el rendimiento del sistema.

Hay dos nuevos bits del registro de control **CR0** que controlan el funcionamiento del caché: **CD** (*Cache Disable*, bit 30) y **NW** (*Not write-through*, bit 29). Cuando CD = 1, el 80486 no leerá memoria externa si hay una copia en el caché, si NW = 1, el 80486 no escribirá en la memoria externa si hay datos en el caché (sólo se escribirá en el caché). La operatoria normal (caché habilitado) es CD = NW = 0. Nótese que si CD = NW = 1 se puede utilizar el caché como una RAM rápida (no hay ciclos externos de bus ni para lectura ni para escritura si hay **acierto** en el caché). Para deshabilitar completamente el caché deberá poner CD = NW = 1 y luego ejecutar una de las instrucciones para vaciar el caché.

Existen dos instrucciones para vaciar el caché: INVD y WBINVD.

Versiones del 80486

- **80486 DX**: En abril de 1989 la compañía Intel presentó su nuevo microprocesador: el **80486 DX**, con 1.200.000 transistores a bordo, el doble de la velocidad del 80386 y 100% de compatibilidad

con los microprocesadores anteriores. El consumo máximo del 486DX de 50 MHz es de 5 watt.

- **80486 SX**: En abril de 1991 introdujo el **80486 SX**, un producto de menor costo que el anterior sin el coprocesador matemático que posee el 80486 DX (bajando la cantidad de transistores a 1.185.000).

- **80486 DX2**: En marzo de 1992 apareció el **80486 DX2**, que posee un duplicador de frecuencia interno, con lo que las distintas funciones en el interior del chip se ejecutan al doble de velocidad, manteniendo constante el tiempo de acceso a memoria. Esto permite casi duplicar el rendimiento del microprocesador, ya que la mayoría de las instrucciones que deben acceder a memoria en realidad acceden al caché interno de 8 KBytes del chip.

- **80486 SL**: En el mismo año apareció el **80486 SL** con características especiales de ahorro de energía.

- **80486 DX4**: Siguiendo con la filosofía del DX2, en 1994 apareció el **80486 DX4**, que triplica la frecuencia de reloj y aumenta el tamaño del caché interno a 16 KBytes.

El chip se empaqueta en el formato PGA (*Pin Grid Array*) de 168 pines en todas las versiones. En el caso del SX, también existe el formato PQFP (*Plastic Quad Flat Pack*) de 196 pines. Las frecuencias más utilizadas en estos microprocesadores son: SX: 25 y 33 MHz, DX: 33 y 50 MHz, DX2: 25/50 MHz y 33/66 MHz y DX4: 25/75 y 33/100 MHz. En los dos últimos modelos, la primera cifra indica la frecuencia del bus externo y la segunda la del bus interno. Para tener una idea de la velocidad, el 80486 DX2 de 66 MHz ejecuta 54 millones de instrucciones por segundo.

Nuevas instrucciones del 80486

BSWAP *reg32* (*Byte Swap*): Cambia el orden de los bytes. Si antes de BSWAP el orden era B0, B1, B2, B3, después de BSWAP el orden será B3, B2, B1, B0.

CMPXCHG *dest*, *src* (*Compare and Exchange*): Compara el acumulador (**AL** o **EAX**) con *dest*. Si es igual, *dest* se carga con el valor de *src*, en caso contrario, el acumulador se carga con el valor de *dest*.

INVD (*Invalidate Cache*): Vacía el caché interno. Realiza un ciclo de bus especial que indica que deben vaciarse los cachés externos. Los datos en el caché que deben escribirse en la memoria se pierden.

INVLPG (*Invalidate Translation Look-Aside Buffer Entry*): Invalida una entrada de página en el buffer de conversión por búsqueda (**TLB**). Esta instrucción puede ser implementada de forma diferente en microprocesadores futuros.

WBINVD (*Write Before Invalidate Data Cache*): Realiza los cambios indicados en el caché en la memoria externa y luego lo invalida.

XADD *dest*, *src* (*Exchange and Add*): Suma los operandos fuente y destino poniendo el resultado en el destino. El valor original del destino se mueve a la fuente. La instrucción cambia los indicadores de acuerdo al resultado de la suma.

Además de las instrucciones mencionadas, todos los modelos del 486 excepto el SX incluyen todas las instrucciones del coprocesador matemático 80387.

Los últimos modelos (486DX4, SL) incluyen la instrucción **CPUID**, que se introdujo con el procesador Pentium. Además en el SL se incluye la instrucción **RSM** (sirve para volver del modo de *manejo de energía*).

El microprocesador Pentium

Introducción

El 19 de octubre de 1992, Intel anunció que la quinta generación de su línea de procesadores compatibles (cuyo código interno era el P5) llevaría el nombre **Pentium** en vez de 586 u 80586, como todo el mundo estaba esperando. Esta fue una estrategia de Intel para poder registrar la marca y así poder diferir el nombre de sus procesadores del de sus competidores (AMD y Cyrix principalmente).

Este microprocesador se presentó el 22 de marzo de 1993 con velocidades iniciales de 60 y 66 MHz (112 millones de instrucciones por segundo en el último caso), 3.100.000 transistores (fabricado con el proceso BICMOS (Bipolar-CMOS) de 0,8 micrones), caché interno de 8 KB para datos y 8 KB para instrucciones, verificación interna de paridad para asegurar la ejecución correcta de las instrucciones, una unidad de punto flotante mejorada, bus de datos de 64 bit para una comunicación más rápida con la memoria externa y, lo más importante, permite la ejecución de dos instrucciones simultáneamente. El chip se empaqueta en formato PGA (Pin Grid Array) de 273 pines.

Como el Pentium sigue el modelo del procesador 386/486 y añade unas pocas instrucciones adicionales pero ningún registro programable, ha sido denominado un diseño del tipo 486+. Esto no quiere decir que no hay características nuevas o mejoras que aumenten la potencia. La mejora más significativa sobre el 486 ha ocurrido en la unidad de punto flotante. Hasta ese momento, Intel no había prestado mucha atención a la computación de punto flotante, que tradicionalmente había sido el bastión de las estaciones de ingeniería. Como resultado, los coprocesadores 80287 y 80387 y los coprocesadores integrados en la línea de CPUs 486 DX se han considerado anémicos cuando se les compara con los procesadores RISC (Reduced Instruction Set Computer), que equipan dichas estaciones.

Todo esto ha cambiado con el Pentium: la unidad de punto flotante es una prioridad para Intel, ya que debe competir en el mercado de Windows NT con los procesadores RISC tales como el chip Alpha 21064 de Digital Equipment Corporation y el MIPS R4000 de Silicon Graphics. Esto puede ayudar a explicar por qué el Pentium presenta un incremento de 5 veces en el rendimiento de punto flotante cuando se le compara con el diseño del 486. En contraste, Intel sólo pudo extraer un aumento del doble para operaciones de punto fijo o enteros.

El gran aumento de rendimiento tiene su contraparte en el consumo de energía: 13 watt bajo la operación normal y 16 watt a plena potencia (3,2 amperes x 5 volt = 16 watt), lo que hace que el chip se caliente demasiado y los fabricantes de tarjetas madres (motherboards) tengan que agregar complicados sistemas de refrigeración.

Teniendo esto en cuenta, Intel puso en el mercado el 7 de marzo de 1994 la segunda generación de procesadores Pentium. Se introdujo con las velocidades de 90 y 100 MHz con tecnología de 0,6 micrones y Posteriormente se agregaron las versiones de 120, 133, 150, 160 y 200 MHz con tecnología de 0,35 micrones. En todos los casos se redujo la tensión de alimentación a 3,3 volt. Esto redujo drásticamente el consumo de electricidad (y por ende el calor que genera el circuito integrado). De esta manera el chip más rápido (el de 200 MHz) consume lo mismo que el de 66 MHz. Estos integrados vienen con 296 pines. Además la cantidad de transistores subió a 3.300.000. Esto se debe a que se agregó circuitería adicional de control de *clock*, un controlador de interrupciones avanzado programable (APIC) y una interfaz para procesamiento dual (facilita el desarrollo de motherboards con dos Pentium).

En octubre de 1994, un matemático reportó en Internet que la Pentium tenía un error que se presentaba

cuando se usaba la unidad de punto flotante para hacer divisiones (instrucción FDIV) con determinadas combinaciones de números. Por ejemplo:

962 306 957 033 / 11 010 046 = 87 402,6282027341 (respuesta correcta)
962 306 957 033 / 11 010 046 = 87 399,5805831329 (Pentium fallada)

El defecto se propagó rápidamente y al poco tiempo el problema era conocido por gente que ni siquiera tenía computadora. Este *bug* se arregló en las versiones D1 y posteriores de los Pentium 60/66 MHz y en las versiones B5 y posteriores de los Pentium 75/90/100 MHz. Los Pentium con velocidades más elevadas se fabricaron posteriormente y no posee este problema.

En enero de 1997 apareció una tercera generación de Pentium, que incorpora lo que Intel llama *tecnología MMX (MultiMedia eXtensions)* con lo que se agregan 57 instrucciones adicionales. Están disponibles en velocidades de 66/166 MHz, 66/200 MHz y 66/233 MHz (velocidad externa/interna). Las nuevas características incluyen una unidad MMX y el doble de caché. El Pentium MMX tiene 4.500.000 transistores con un proceso CMOS-silicio de 0,35 micrones mejorado que permite bajar la tensión a 2,8 volt. Externamente posee 321 pines.

Vías de acceso múltiples

Lo que comenzó con la técnica del 386/486 de tener vías de acceso múltiples para la ejecución de instrucciones, se ve refinado en el Pentium ya que tiene un diseño con doble vía de acceso. El objetivo de ésta es el de procesar múltiples instrucciones simultáneamente, en varios estados de ejecución, para obtener una velocidad de ejecución general de instrucciones de una instrucción por ciclo de reloj.

El resultado final de la estructura doble vía de acceso es un diseño superescalar que tiene la habilidad de ejecutar más de una instrucción en un ciclo de reloj dado. Los procesadores escalares, como la familia del 486, tienen sólo una vía de acceso.

Se puede pensar que el microprocesador moderno con vías de acceso doble es similar a una línea de producción que recibe en un extremo materias primas sin procesar y a medio procesar y que saca el producto terminado en el otro extremo. La línea de producción con vía de acceso doble del Pentium transforma la materia prima de información y de código de software en el producto terminado. El Pentium sigue el modelo de vía de acceso del 486, ejecutando instrucciones simples con enteros en un ciclo de reloj. Sin embargo es más exacto decir que aquellas instrucciones estaban en la etapa de ejecución de la vía de acceso durante un ciclo de reloj. Siempre se requieren ciclos adicionales de reloj para buscar, decodificar la instrucción y otros procesos vitales. La secuencia de funcionamiento de la vía de datos es como sigue: prebúsqueda, decodificación 1, decodificación 2, ejecución y retroescritura.

Esto es similar a una línea de producción que produce un artículo por minuto, pero que se demora varias horas para completar cada artículo individual, y siempre habrá una multitud de unidades en diferentes etapas del ensamblado. En el caso óptimo, las instrucciones estarían alineadas en la vía de acceso de forma que, en general, ésta ejecutará aproximadamente una instrucción por ciclo de reloj.

Los aspectos superescalares del Pentium dependen de su vía de acceso doble. Los procesadores superescalares permiten que se ejecute más de una instrucción por vez. El procesador tiene dos vías de acceso de enteros, una en forma de U y otra en forma de V y automáticamente aparea las instrucciones para incrementar la proporción de instrucciones por ciclo de reloj para que sea mayor que 1. Si el tener múltiples instrucciones pasando por dos vías suena como el equivalente de un tranque en el tráfico del

microprocesador, eso no es así, porque hay reglas y restricciones que evitan las colisiones y los retrasos.

Por ejemplo, los conflictos principales que tienen que ver con generar y ejecutar más de una instrucción al mismo tiempo incluyen dependencias de información (de un par de instrucciones que se emiten al mismo tiempo, la información de salida de una se necesita como entrada de otra, como por ejemplo INC AX, INC AX), dependencias de recursos (es una situación en la que ambas instrucciones que fueron emitidas al mismo tiempo compiten por el mismo recurso del microprocesador, por ejemplo, un registro específico. Hay técnicas avanzadas que permiten disminuir estas dependencias pero el Pentium no las tiene) o saltos en el código (llamadas dependencias de procedimiento).

Si se detectara una dependencia, el procesador reconoce que las instrucciones deben fluir en orden y asegura que la primera instrucción termine su ejecución antes de generar la segunda instrucción. Por ejemplo, el Pentium envía la primera instrucción por la vía U y genera la segunda y tercera instrucciones juntas, y así sucesivamente.

Las dos vías no son equivalentes, o intercambiables. La vía U ejecuta instrucciones de enteros y de punto flotante, mientras que la vía V sólo puede ejecutar instrucciones simples con enteros y la instrucción de intercambio de contenido de registros de punto flotante.

El orden en que viajan las instrucciones por las vías dobles del Pentium nunca es distinto al orden de las instrucciones en el programa que se ejecuta. También tanto las instrucciones para la vía U como la V entran a cada etapa de la ruta en unísono. Si una instrucción en una vía termina una etapa antes de que la instrucción en la otra vía, la más adelantada espera por la otra antes de pasar a la próxima etapa.

Las instrucciones de punto flotante, comúnmente utilizadas en programas de matemática intensiva, pasan las vías de entero y son manipuladas desde la vía de punto flotante en la etapa de ejecución. En definitiva las vías de enteros y el de punto flotante operan independiente y simultáneamente.

Dependencias de procedimiento

Puede ocurrir un problema potencial con la ejecución debido a las muchas trayectorias que la secuencia de una instrucción puede tomar. La predicción de la trayectoria a tomar es el método que debe usarse aquí. El Pentium dibuja algo parecido a un mapa de carreteras de los lugares a donde es posible que se dirija la instrucción y lo usa para tratar de agilizar la ejecución de la instrucción. Intel afirma que esta característica, por sí sola, aumenta el rendimiento un 25%.

Sin predecir las trayectorias a tomar, si un procesador superescalar doble estuviera ocupado procesando instrucciones en ambas vías de acceso y se encontrara una instrucción de salto que cambiara la secuencia de ejecución de la instrucción, ambas vías y el buffer de prebúsqueda de instrucción tendrían que borrarse y cargarse con nuevas instrucciones, lo que retrasaría al procesador. Con la predicción de la trayectoria a tomar, el procesador precarga las instrucciones de una dirección de destino que haya sido pronosticada de un juego alterno de buffers. Esto le da al procesador una ventaja para reducir los conflictos y las demoras. El resultado es una mejor utilización de los recursos del procesador.

Hay dos tipos de instrucciones de salto: condicional e incondicional. Un salto incondicional siempre lleva el flujo de la instrucción a una nueva dirección de destino y siempre se ejecuta. Una situación más incómoda es el salto condicional donde se puede o no desviar el flujo del programa de acuerdo a los resultados de una comparación o código de condición y puede incluir varios tipos de instrucciones.

Cuando no se ejecuta un salto condicional, el programa sigue ejecutando la próxima instrucción de la secuencia. Muchos programas tienen de un 10% a un 20% de instrucciones de salto condicional y hasta un 10% de saltos incondicionales. El porcentaje de veces que se ejecuta un salto condicional varía de programa a programa, pero es de un promedio de un 50%. Las instrucciones de lazo o de repetición hace que se tomen decisiones frecuentemente, hasta el 90% del tiempo en algunos casos. Un buen sistema de predicción de decisiones escogerá las trayectorias correctas más del 80% del tiempo.

Físicamente, la unidad de predicción de decisiones (BPU) está situada al lado de la vía de acceso, y revisa con anticipación el código de la instrucción para determinar las conexiones de las decisiones. El orden es algo así: La BPU inspecciona las instrucciones en la etapa de prebúsqueda, y si la lógica de predicción de decisiones predice que se va a realizar el salto, se le indica inmediatamente a la unidad de prebúsqueda (PU) que comience a buscar instrucciones de la dirección de destino de la dirección que se predijo. De forma alterna, si se determinó que no se iba a tomar la decisión, no se perturba la secuencia original de prebúsqueda. Si la trayectoria pronosticada resulta ser errónea, se vacía la vía de acceso y los buffers alternos de prebúsqueda, y se recomienza la prebúsqueda desde la trayectoria correcta. Se paga una penalidad de tres o cuatro ciclos de reloj por predecir una trayectoria de forma errónea.

El Pentium usa un buffer de decisión de destino (BTB) como su mecanismo. El BTB incluye tres elementos por cada entrada: la dirección de la instrucción de salto, la dirección de destino de la instrucción y los bits de historia. Se usa una tabla de hasta 256 entradas para predecir los resultados de las decisiones. El flujo está basado, y se administra directamente desde la vía U. Se usa la dirección de la vía U para la dirección de la instrucción de decisión del BTB, aún si la decisión está realmente en la vía V. Hay dos bits de historia que informan si se tomó la decisión anterior o no. El resultado es un procesador que corre con suavidad y que a menudo sabe lo que hará antes de completar la tarea.

Ejecución de punto flotante en el Pentium

Se ha reconstruido por completo la unidad de punto flotante (FPU), a partir de la de los 386 y 486 y ahora tiene algunas de las características de los RISC. Hay ocho etapas de vía y las cinco primeras se comparten con la unidad de enteros. La unidad cumple con la norma IEEE-754, usa algoritmos más rápidos y aprovecha la arquitectura con vías para lograr mejoras de rendimiento de entre 4 y 10 veces, dependiendo de la optimización del compilador.

Ahorro de energía

El Pentium usa un modo de administración de sistema (**SMM**) similar al que usa el 486 SL, que permite que los ingenieros diseñen un sistema con bajo consumo. La interrupción de administración del sistema activa el SMM por debajo del nivel del sistema operativo o de la aplicación. Se guarda toda la información sobre el estado de los registros para después restaurarla, y se ejecuta el código de manejador de SMM desde un espacio de direcciones totalmente separado, llamado RAM de administración del sistema (SMRAM). Se sale del SMM ejecutando una instrucción especial (RSM). Esto lleva al CPU de nuevo al mismo punto en que estaba cuando se llamó al SMM.

Algunos procesadores (100 MHz o más lentos) presentan problemas en este modo.

Nuevas instrucciones del microprocesador Pentium

Son las siguientes:

CMPXCHG8B *reg, mem64* (*Compare and Exchange 8 Bytes*): Compara el valor de 64 bits ubicado en EDX:EAX con un valor de 64 bits situado en memoria. Si son iguales, el valor en memoria se reemplaza por el contenido de ECX:EBX y el indicador ZF se pone a uno. En caso contrario, el valor en memoria se carga en EDX:EAX y el indicador ZF se pone a cero.

CPUID (*CPU Identification*): Le informa al software acerca del modelo de microprocesador en que está ejecutando. Un valor cargado en EAX antes de ejecutar esta instrucción indica qué información deberá retornar CPUID. Si EAX = 0, se cargará en dicho registro el máximo valor de EAX que se podrá utilizar en CPUID (para el Pentium este valor es 1). Además, en la salida aparece la cadena de identificación del fabricante contenido en EBX, ECX y EDX. EBX contiene los primeros cuatro caracteres, EDX los siguientes cuatro, y ECX los últimos cuatro. Para los procesadores Intel la cadena es "GenuineIntel". Luego de la ejecución de CPUID con EAX = 1, EAX[3:0] contiene la identificación de la revisión del microprocesador, EAX[7:4] contiene el modelo (el primer modelo está indicado como 0001b) y EAX[11:8] contiene la familia (5 para el Pentium). EAX[31:12], EBX y ECX están reservados. El procesador pone el registro de características en EDX a 1BFh, indicando las características que soporta el Pentium. Un bit puesto a uno indica que esa característica está soportada. La instrucción no afecta los indicadores.

RDMSR (*Read from Model-Specific Register*): El valor en ECX especifica uno de los registros de 64 bits específicos del modelo del procesador. El contenido de ese registro se carga en EDX:EAX. EDX se carga con los 32 bits más significativos, mientras que EAX se carga con los 32 bits menos significativos.

RDTSC (*Read from Time Stamp Counter*): Copia el contenido del contador de tiempo (TSC) en EDX:EAX (el Pentium mantiene un contador de 64 bits que se incrementa por cada ciclo de reloj). Cuando el nivel de privilegio actual es cero el estado del bit TSD en el registro de control CR4 no afecta la operación de esta instrucción. En los anillos 1, 2 ó 3, el TSC se puede leer sólo si el bit TSD de CR4 vale cero.

RSM (*Resume from System Management Mode*): El estado del procesador se restaura utilizando la copia que se creó al entrar al modo de manejo del sistema (SMM). Sin embargo, los contenidos de los registros específicos del modelo no se afectan. El procesador sale del SMM y retorna el control a la aplicación o sistema operativo interrumpido. Si el procesador detecta alguna información inválida, entra en el estado de apagado (shutdown).

WRMSR (*Write to Model-Specific Register*): El valor en ECX especifica uno de los registros de 64 bits específicos del modelo del procesador. El contenido de EDX:EAX se carga en ese registro. EDX debe contener los 32 bits más significativos, mientras que EAX debe contener los 32 bits menos significativos.

```
; Ejemplo de programa residente: al apretar una tecla, se escucha
; un sonido que depende de la tecla presionada.
;
; Hecho por Dario A. Alpern
;
FIRMA1          equ  2387h
FIRMA2          equ  7A8Dh
codigo             segment
                assume cs:codigo
                org 100h                ;Inicio de programa .COM
inicio:         jmp short verif_instalac ;Saltar al codigo de instalacion.
viejo_int9      label dword
offset_int9     dw 0
segmento_int9   dw 0
Estado          db 1                    ;1: Habilitado; 0: No habilitado.
Firma           dw FIRMA1,FIRMA2
;
;  Manejador de la interrupcion 9 (teclado)
;
int9handler:    cmp Estado,0            ;Sonido habilitado?
                jz ir_a_viejo_INT9      ;Saltar si no es asi.
                push ax                 ;Preservar registros utilizados.
            push cx
            push dx
                mov al,182              ;Generar onda cuadrada en timer 2
                out 67,al               ;para poder producir sonido.
                in al,96                ;Leer el controlador de teclado.
                cmp al,128              ;Ver si se apreto o solto una tecla.
                jnc silencio            ;Saltar si se solto.
                add al,al               ;Ajustar el codigo de exploracion:
                jns hallar_frec         ;cada dos valores se debe incrementar
                sub al,127              ;un semitono.
hallar_frec:    mov cl,al               ;CX = Cantidad de medios semitonos
                xor ch,ch               ;     por encima del LA (55 Hz).
                mov ax,21690            ;AX = 55 Hz (LA)
ciclo_frec:     mov dx,63670            ;DX = Medio semitono.
                mul dx                  ;DX = Siguiente medio semitono.
                mov ax,dx               ;AX = Siguiente medio semitono.
                loop ciclo_frec         ;Continuar calculando frecuencia.
                jmp short sonar         ;Ir a poner la nota.
silencio:       mov ax,40               ;Silencio.
sonar:          out 66,al               ;Poner la parte baja del contador.
            mov al,ah
                out 66,al               ;Poner la parte alta del contador.
                in al,97                ;Habilitar salida del parlante.
            or al,3
            out 97,al
                pop dx                  ;Restaurar registros utilizados.
            pop cx
            pop ax
ir_a_viejo_INT9:
                jmp viejo_int9          ;Ir al viejo manejador de la
                                        ;interrupcion del teclado.
;
; Instalacion del TSR.
;
verif_instalac: xor ax,ax               ;Apuntar a la tabla de interrupcion.
                mov es,ax
                lds bx,es:9*4           ;DS:BX = Puntero al manejador de la
                                        ;interrupcion del teclado.
                cmp word ptr [bx-4],FIRMA1  ;Si la firma no coincide,
                jne instalar                ;instalar el TSR.
                cmp word ptr [bx-2],FIRMA2  ;Si la firma no coincide,
                jne instalar                ;instalar el TSR.
                xor ds:Estado,1         ;Cambiar estado: sonido si/no.
```

```
66            push cs
67                 pop ds                 ;DS = Segmento de codigo.
68                 mov dx,offset sonido_si ;Puntero a texto de sonido activado.
69                 jnz mostrar_texto
70                 mov dx,offset sonido_no ;Puntero a texto de sonido desactivado.
71 mostrar_texto:  mov ah,9
72                 int 21h                ;Llamar a funcion DOS para mostrarlo.
73                 in al,97               ;Deshabilitar sonido.
74            and al,252
75            out 97,al
76                 mov ax,4c00h           ;Fin del programa.
77            int 21h
78 instalar:       mov offset_int9,bx     ;Preservar offset vieja interrupcion 9.
79                 mov segmento_int9,ds   ;Preservar segmento vieja int 9.
80                 cli                    ;Deshabilitar interrupciones.
81                 mov es:9*4,offset int9handler ;Nuevo offset interrupcion 9.
82                 mov es:9*4+2,cs        ;Nuevo segmento interrupcion 9.
83                 sti                    ;Habilitar interrupciones.
84                 push cs
85                 pop ds                  ;DS = Segmento de codigo.
86                 mov dx,offset sonido_si ;Mostrar texto de sonido habilitado.
87            mov ah,9
88            int 21h
89                 mov dx,offset verif_instalac  ;Instalar TSR.
90            int 27h
91 sonido_si       db "Sonido activado.",13,10,"$"
92 sonido_no       db "Sonido desactivado.",13,10,"$"
93 codigo             ends
94            end inicio
```

```
 1 ;-----------------------------------------------------------------------------;
 2 ; Programa para mostrar el modo protegido del 80386.                         ;
 3 ; El programa tiene dos tareas: una muestra una lista de números primos      ;
 4 ; mediante el método de divisiones sucesivas en la parte superior de la      ;
 5 ; pantalla, y la otra muestra una lista de números primos mediante el método ;
 6 ; de potenciación en la parte inferior de la pantalla.                       ;
 7 ; Además hay dos interrupciones: IRQ0 (timer - INT 08h) para el manejo del   ;
 8 ; "scheduler" (rutina que determina cuál es la tarea que debe correr en un   ;
 9 ; instante determinado en un sistema multitarea) e IRQ1 (teclado - INT 09h)  ;
10 ; para el manejo de las teclas de función y entrada de números.             ;
11 ;                                                                           ;
12 ; Hecho por Darío Alejandro Alpern                                          ;
13 ;                                                                           ;
14 ; Versiones del programa:                                                   ;
15 ; - Versión 2.0 (marzo de 1997): Se agregó la tarea que muestra números     ;
16 ;   primos mediante potenciación. Se agregó separación de millares en los   ;
17 ;   números primos que se muestran. Se agregó "equ" para aclarar el signifi- ;
18 ;   cado de ciertas variables.                                              ;
19 ;   Se agregó manejo de las teclas Pausa y Break.                           ;
20 ; - Versión 1.0 (octubre de 1995): Programa original.                       ;
21 ;-----------------------------------------------------------------------------;
22         .386p
23 ;Segmento inútil que sólo se utiliza para inicializar los saltos lejanos.
24 nada    segment use16 at 0
25 dummy   label far
26 nada    ends
27 ;-----------------------------------------------------------------------------
28 ; Segmento de código y datos.
29 codigo  segment use16
30         org 100h
31         assume cs:codigo,ds:codigo
32 comienzo:jmp inicio
33 ;-----------------------------------------------------------------------------
34 ; Definición de campos de bits del byte de derechos de acceso:
35 DA_SEGMENTO_PRESENTE  equ 80h  ;El segmento se encuentra en memoria.
36 DA_NIVEL_PRIVILEGIO_0 equ 00h  ;Mayor privilegio. Acceso a todos los recursos.
37 DA_NIVEL_PRIVILEGIO_1 equ 20h
38 DA_NIVEL_PRIVILEGIO_2 equ 40h
39 DA_NIVEL_PRIVILEGIO_3 equ 60h  ;Menor privilegio.
40 DA_SEGMENTO_CODIGO    equ 18h  ;Segmento con código ejecutable.
41 DA_SEGMENTO_DATOS     equ 10h  ;Segmento sin código ejecutable. Siempre se
42                                 ;puede leer.
43 DA_OFFSET_MENOR_LIM   equ 00h  ;El offset válido en el segmento de datos es
44                                 ;menor o igual que el límite.
45 DA_OFFSET_MAYOR_LIM   equ 04h  ;El offset válido en el segmento de datos es
46                                 ;mayor que el límite.
47 DA_LECTURA_ESCRITURA  equ 02h  ;El segmento de datos se puede leer y escribir.
48 DA_SOLO_LECTURA       equ 00h  ;El segmento de datos se puede leer pero no
49                                 ;escribir.
50 DA_LECTURA_EJECUCION  equ 02h  ;El segmento de código se puede leer y ejecutar.
51 DA_SOLO_EJECUCION     equ 00h  ;El segmento de código se puede ejecutar pero
52                                 ;no se pueden leer sus bytes.
53 ;Byte de derechos de acceso para el segmento de código que se usa aquí:
54 DA_CODIGO             equ DA_SEGMENTO_PRESENTE   + \
55                             DA_NIVEL_PRIVILEGIO_0  + \
56                             DA_SEGMENTO_CODIGO     + \
57                             DA_LECTURA_EJECUCION
58 ;Byte de derechos de acceso para los segmentos de datos que se usan aquí:
59 DA_DATOS              equ DA_SEGMENTO_PRESENTE   + \
60                             DA_NIVEL_PRIVILEGIO_0  + \
61                             DA_SEGMENTO_DATOS      + \
62                             DA_OFFSET_MENOR_LIM    + \
63                             DA_LECTURA_ESCRITURA
64 DA_TSS_286            equ 81h   ;Descriptor de TSS tipo 286.
65 DA_TSS_386            equ 89h   ;Descriptor de TSS tipo 386.
```

```
 66 DA_TSS_GATE           equ 85h   ;Descriptor de compuerta de tarea.
 67 ;---------------------------------------------------------------------------
 68 ; Macro para definir el descriptor (ocho bytes):
 69 descriptor macro base, limite, derechos_acceso
 70            IF derechos_acceso and 10h     ;; Segmento de código o datos.
 71               IF limite GT 0FFFFFh        ;; Si granularidad = 4KB:
 72                  dw ((limite) SHR 12) and 0FFFFh
 73               ELSE
 74                  dw (limite) and 0FFFFh   ;; Bytes 1-0 del límite.
 75               ENDIF
 76               dw (base) and 0FFFFh        ;; Bytes 1-0 de la base.
 77               db ((base) SHR 16) and 0FFh ;; Byte 2 de la base.
 78               db derechos_acceso          ;; Byte de derechos de acceso.
 79               IF limite GT 0FFFFFh
 80                  db 80h + ((limite) SHR 28) ;; Indicar granularidad = 4KB.
 81               ELSE
 82                  db (limite) SHR 16       ;; Byte 2 del límite.
 83               ENDIF
 84               db (base) SHR 24            ;; Byte 3 de la base.
 85            ELSE                           ;; Segmento de tabla del sistema.
 86               dw limite
 87               dw base
 88               db 0
 89               db derechos_acceso
 90               db 0
 91               db 0
 92            ENDIF
 93            ENDM
 94 ;---------------------------------------------------------------------------
 95 ; Valores que van en la tabla de descriptores globales (diez descriptores),
 96 ; ocho bytes por descriptor.
 97 gdt     equ qword ptr $ - 8  ;Inicio de la tabla de descriptores globales.
 98 ;
 99 ; 1ø descriptor: Segmento de código.
100 code_sel equ $ - gdt    ;Valor del selector del segmento de código.
101         descriptor 0, 0FFFFh, DA_CODIGO
102 ;
103 ; 2ø descriptor: Segmento de pila y datos (alias del segmento de código).
104 data_sel equ $ - gdt    ;Valor del selector del segmento de datos.
105         descriptor 0, 0FFFFh, DA_DATOS
106 ;
107 ; 3ø descriptor: Segmento de buffer de pantalla.
108 pant_sel equ $ - gdt    ;Valor del selector del buffer de pantalla.
109         descriptor 0B0000h, 0FFFFh, DA_DATOS ;El valor de la base es para
110                          ;adaptador monocromo. Si el adaptador es color, la base
111                          ;se cambia en tiempo de ejecución.
112 ;
113 ; 4ø descriptor: Segmento de parte inferior del buffer de pantalla.
114 pant_inf_sel equ $-gdt  ;Valor del selector de la parte inferior de pantalla.
115         descriptor 0B0000h+15*80*2, 0FFFFh, DA_DATOS
116 ;
117 ; 5ø Descriptor: Descriptor de segmento de estado de la tarea (TSS) inicial.
118 TSS_inic_sel equ $-gdt  ;Valor del selector del TSS inicial.
119         descriptor TSS_original, <size TSS_386 - 1>, DA_TSS_286
120 ;
121 ; 6ø Descriptor: Descriptor de segmento de estado de la tarea (TSS)
122 ;               primos_division.
123 TSS_pri1_sel equ $-gdt  ;Valor del selector del TSS de primos_division.
124         descriptor TSS_primos_division, <size TSS_386 - 1>, DA_TSS_386
125 ;
126 ; 7ø Descriptor: Descriptor de segmento de estado de la tarea (TSS)
127 ;               primos_potenciacion.
128 TSS_pri2_sel equ $-gdt  ;Valor del selector de TSS de primos_potenciacion.
129         descriptor TSS_primos_potenciacion, <size TSS_386 - 1>, DA_TSS_386
130 ;
```

```
131 ; 8ø Descriptor: Descriptor de segmento de estado de la tarea (TSS) int8.
132 TSS_int8_sel equ $-gdt  ;Valor del selector de TSS de la interrupción 8.
133         descriptor TSS_int8, <size TSS_386 - 1>, DA_TSS_286
134 ;

135 ; 9ø Descriptor: Descriptor de segmento de estado de la tarea (TSS) int9.
136 TSS_int9_sel equ $-gdt  ;Valor del selector de TSS de la interrupción 9.
137         descriptor TSS_int9, <size TSS_386 - 1>, DA_TSS_386
138 ;
139 ; 10ø descriptor: Segmento de pila (alias del segmento de código).
140 dir_cero_sel equ $-gdt  ;Valor del selector del segmento de datos.
141         descriptor 0, 0FFFFh, DA_DATOS
142 fin_gdt equ $
143 ; Valores que van en la tabla de descriptores de interrupción
144 ; (dos descriptores correspondientes a INT 8 e INT 9)
145 idt     equ $ - 64      ;Inicio de la tabla de interrupción.
146 ;
147 ; 1ø Descriptor: Compuerta de interrupción correspondiente a INT 8.
148         descriptor TSS_int8_sel, 0, DA_TSS_GATE
149 ;
150 ; 2ø Descriptor: Compuerta de interrupción correspondiente a INT 9.
151         descriptor TSS_int9_sel, 0, DA_TSS_GATE
152 fin_idt equ $
153 ;-------------------------------------------------------------------------
154 ; Estructura del segmento de estado de la tarea (TSS).
155 TSS_386         struc
156  back_link      dw 0,0
157  PL0_ESP_value  dd 0
158  PL0_SS_value   dd data_sel
159  PL1_ESP_value  dd 0
160  PL1_SS_value   dw 0,0
161  PL2_ESP_value  dd 0
162  PL2_SS_value   dw 0,0
163  CR3_value      dd 0
164  EIP_value      dd 0
165  EFLAGS_value   dd 00003202h
166  EAX_value      dd 0
167  ECX_value      dd 0
168  EDX_value      dd 0
169  EBX_value      dd 0
170  ESP_value      dd 0
171  EBP_value      dd 0
172  ESI_value      dd 0
173  EDI_value      dd 0
174  ES_value       dd 0
175  CS_value       dd code_sel
176  SS_value       dd data_sel
177  DS_value       dd data_sel
178  FS_value       dd 0
179  GS_value       dd 0
180  LDT_value      dd 0
181  DEBUG_value    dw 0
182  bitmap_offset  dw 0
183  bitmap_end     db 255
184 TSS_386         ends
185 TSS_286         struc
186  backlink       dw 0
187  PL0_SP_val     dw 0
188  PL0_SS_val     dw data_sel
189  PL1_SP_val     dw 0
190  PL1_SS_val     dw 0
191  PL2_SP_val     dw 0
192  PL2_SS_val     dw 0
193  IP_val         dw 0
194  FLAGS_val      dw 3202h
```

```
195  AX_val         dw 0
196  CX_val         dw 0
197  DX_val         dw 0
198  BX_val         dw 0
199  SP_val         dw 0
200  BP_val         dw 0
201  SI_val         dw 0
202  DI_val         dw 0
203  ES_val         dw 0
204  CS_val         dw code_sel
205  SS_val         dw data_sel
206  DS_val         dw data_sel
207  LDT_val        dw 0
208 TSS_286         ends
209 base            equ comienzo - 100h
210 TSS_original    TSS_286 <,,,,,,,,,,,,,,,,,0,0,0,0>
211 TSS_primos_division     TSS_386 <,pila_primos_division + 100h - base,,,,,,,primos
212 TSS_primos_potenciacion     TSS_386 <,pila_primos_potenciacion + 100h - base,,,,,
213 TSS_int8        TSS_286 <,pila_int8 + 100h - base,,,,,,int8han - base,,,20,10,,pi
214 TSS_int9        TSS_386 <,pila_int9 + 100h - base,,,,,,,int9han - base,3002h,,,,,
215 LSD_d_ajustado   dd 0
216 MSD_d_ajustado   dd 0
217 LSD_d_aux dd 0
218 MSD_d_aux dd 0
219 base_SPRP dd 0
220 res_LO  dd 0
221 res_HI  dd 0
222 bits_d  dw 0
223 valor_s dw 0
224 picint  db 0
225 gdtr    label fword     ;Información a almacenar en el GDTR (6 bytes).
226         dw fin_gdt - gdt - 1  ;Límite de la tabla de descriptores globales.
227         dw gdt          ;Base de la tabla de descriptores globales.
228         dw 0            ;Se llena durante la ejecución del programa.
229 idtr    label fword     ;Información a almacenar en el IDTR (6 bytes).
230         dw fin_idt - idt - 1  ;Límite de la IDT.
231         dw idt          ;Base de la tabla de descriptores de interrupción.
232         dw 0            ;Se llena durante la ejecución del programa.
233 real_idtr df 0          ;Espacio para almacenar el IDTR en modo real.
234 texto1  dw 11*160+0*2
235         db "Tiempo: 50%  ",24,": Más, ",25,": Menos, F1 = Detener,"
236         db " F2 = Continuar, F3 = Cambiar número",0
237 ;El ASCII 24 es la flecha hacia arriba, y el ASCII 25 es la flecha hacia abajo.
238 texto2  dw 12*160+8*2
239         db "Hecho por Darío Alejandro Alpern en marzo de 1997 (versión 2.0)",0
240 texto3  dw 13*160+0*2
241         db "Tiempo: 50%  ",25,": Más, ",24,": Menos, F4 = Detener,"
242         db " F5 = Continuar, F6 = Cambiar número",0
243 texto4  dw 12*160+8*2
244         db "               Nuevo número:                                    ",0
245 texto5  db "Este programa no puede correr en modo virtual 8086.",13,10
246         db "Pruebe eliminando la línea DEVICE=EMM386 en CONFIG.SYS",13,10,"$"
247 texto6  db "Este programa requiere 80386 o posterior.",13,10,"$"
248 ; Definición del byte de estado:
249 SCAN_CODE_E1    equ 128         ;El scan code anterior fue E1.
250 SCAN_CODE_E0    equ 64          ;El scan code anterior fue E0 (si bit 7=0).
251 SCAN_DESPUES_E1 equ 64          ;Cantidad de scan codes después del E1
252                                 ;(si bit 7=1).
253 APRETO_PAUSA    equ 32          ;Se apretó la tecla Pausa.
254 APRETO_ESC      equ 16          ;Se apretó la tecla Esc.
255 RUN_POTEN       equ 8           ;La tarea primos_potenciación está corriendo.
256 RUN_DIVISION    equ 4           ;La tarea primos_división está corriendo.
257 CAMBIANDO_POT   equ 2           ;Cambiando número tarea primos_potenciación.
258 CAMBIANDO_DIV   equ 1           ;Cambiando número tarea primos_división.
259 estado  db RUN_POTEN + RUN_DIVISION
```

```
port6845 dw 3B4h
;-------------------------------------------------------------;
; Tarea mitad superior pantalla (hallar primos por divisiones) ;
;-------------------------------------------------------------;
primos_division: mov esi,5      ;Primer divisor. Los divisores serán:
                                ;5,7,11,13,17,19,23,25,29,31,...
        jmp short comparar_divisor_div
;Realizar la división de 64 bits (EBX:EDI) extendidos a 96 por 32 bits (ESI).
;Lo que necesitamos nosotros es el resto (si vale cero, entonces EBX:EDI no
;es primo).
bucle_div_64_32:xor edx,edx     ;Extender el dividendo.
        mov eax,ebx             ;Poner la parte alta del dividendo.
        div esi                 ;Realizar la primera parte de la división.
        mov eax,edi             ;Poner la parte baja del dividendo.
        div esi                 ;Realizar la segunda parte de la división.
        and edx,edx             ;¿El resto vale cero?
        jz short prox_num_div   ;Si es así saltar porque el número no es primo.
        add esi,2               ;Obtener el siguiente divisor.
        xor edx,edx             ;Extender el dividendo.
        mov eax,ebx             ;Poner la parte alta del dividendo.
        div esi                 ;Realizar la primera parte de la división.
        mov eax,edi             ;Poner la parte baja del dividendo.
        div esi                 ;Realizar la segunda parte de la división.
        and edx,edx             ;¿El resto vale cero?
        jz short prox_num_div   ;Si es así saltar porque el número no es primo.
        add esi,4               ;Obtener el siguiente divisor.
comparar_divisor_div:
        cmp ebx,esi             ;Ver si es necesario seguir extendiendo el
                                ;dividendo a 96 bits.
        jae bucle_div_64_32     ;Saltar si es así.
;Realizar la división de 64 bits (EBX:EDI) por 32 bits (ESI).
;Lo que necesitamos nosotros es el resto (si vale cero, entonces EBX:EDI no
;es primo) y el cociente (si es menor que el divisor EBX:EDI es primo).
bucle_div_32_32:mov edx,ebx     ;Poner la parte alta del dividendo.
        mov eax,edi             ;Poner la parte baja del dividendo.
        div esi                 ;Realizar la división.
        and edx,edx             ;¿El resto vale cero?
        jz short prox_num_div   ;Si es así saltar porque el número no es primo.
        add esi,2               ;Obtener el siguiente divisor.
        mov edx,ebx             ;Poner la parte alta del dividendo.
        mov eax,edi             ;Poner la parte baja del dividendo.
        div esi                 ;Realizar la división.
        and edx,edx             ;¿El resto vale cero?
        jz short prox_num_div   ;Si es así saltar porque el número no es primo.
        add esi,4               ;Obtener el siguiente divisor.
        cmp eax,esi             ;¿El divisor supera al cociente?
        jae bucle_div_32_32     ;Saltar si no es así.
;En este punto se sabe que EBX:EDI es primo, por lo que hay que mostrarlo.
        call mostrar_numero
;Hallar el próximo valor de EBX:EDI para ver si es primo o no.
prox_num_div:
        add edi,ebp             ;Sumar el valor adecuado.
        adc ebx,0
        xor bp,2 xor 4          ;Cambiar el indicador de sumar 2 ó 4.
        jmp primos_division
;-------------------------------------------------;
; Subrutina para mostrar los números en la pantalla ;
; Esta subrutina está compartida por ambas tareas   ;
;-------------------------------------------------;
mostrar_numero:
        cli                     ;No debe ocurrir ningún cambio de tarea.
        push ebp                ;Preservar registros afectados.
        push edi
        push ebx
;Conversión binario (EBX:EDI = 64 bits) a decimal (hasta 20 dígitos).
```

```
325         mov bp,10               ;EBP = Divisor.
326         mov cx,3                ;Contador para separador de millares.
327         mov si,sp
328         add si,25+4+4+4         ;SI = Puntero al final del buffer temporario.
329 bucle_bin_a_dec1:
330         cmp ebx,ebp             ;¿Se puede realizar la división sin overflow?
331         jb short bucle_bin_a_dec2  ;Saltar si es así.
332         xor edx,edx             ;Extender el dividendo.
333         mov eax,ebx             ;Cargar la parte alta del dividendo.
334         div ebp                 ;Realizar la primera parte de la división.
335 segunda_division:
336         mov ebx,eax             ;Salvar la nueva parte alta del dividendo.
337         mov eax,edi             ;Cargar la parte baja del dividendo.
338         div ebp                 ;Realizar la segunda parte de la división.
339         mov edi,eax             ;Salvar la nueva parte baja del dividendo.
340         add dl,"0"              ;Convertir el resto a ASCII.
341         mov [si],dl             ;Guardar el resto en el buffer temporario
342                                 ;(de esta manera se obtiene el dígito
343                                 ;"de la derecha" o menos significativo).
344         loop no_insertar_separador1
345         dec si
346         mov byte ptr [si],"."
347         mov cl,3                ;Reinicializar contador para separador.
348 no_insertar_separador1:
349         dec si                  ;Apuntar al dígito anterior.
350         jmp bucle_bin_a_dec1    ;Seguir dividiendo.
351 bucle_bin_a_dec2:
352         mov edx,ebx             ;Cargar la parte alta del dividendo.
353         mov eax,edi             ;Cargar la parte baja del dividendo.
354         div ebp                 ;Dividir por 10.
355         mov edi,eax             ;Dejar la parte baja en el registro que estaba.
356         xor ebx,ebx             ;La parte alta debe ser cero.
357         add dl,"0"              ;Convertir el resto a ASCII.
358         mov [si],dl             ;Guardar el resto en el buffer temporario
359                                 ;(de esta manera se obtiene el dígito
360                                 ;"de la derecha" o menos significativo).
361         loop no_insertar_separador2
362         dec si
363         mov byte ptr [si],"."
364         mov cl,3                ;Reinicializar contador para separador.
365 no_insertar_separador2:
366         dec si                  ;Apuntar al dígito anterior.
367         and eax,eax             ;¿El número vale cero?
368         jnz bucle_bin_a_dec2    ;Seguir dividiendo si no es así.
369         cmp byte ptr [si+1],"."
370         jnz short no_es_separador
371         inc si
372 no_es_separador:
373         mov bp,sp
374         add bp,25+4+4+4         ;BP = Puntero al final del buffer temporario.
375         mov cx,bp
376         inc cx
377         sub cx,si               ;CX = Cantidad de dígitos del número.
378         add cx,cx               ;Convertir a bytes de buffer de pantalla.
379         add cx,[bp+1]           ;Agregarle la posición actual del buffer de
380                                 ;pantalla.
381         cmp cx,10*80*2          ;¿Se sobrepasa el final con lo que quedaría
382                                 ;el número cortado?
383         jbe short poner_numero  ;Saltar si no es así.
384 ;Hacer un "scroll" (correr una línea hacia arriba llenando la línea de abajo
385 ;con espacios) de la porción activa de la pantalla.
386         xchg bx,si              ;Preservar el registro SI.
387         xor di,di               ;DI = Posición del carácter superior izquierdo
388         mov si,160              ;SI = Extremo izquierdo de la segunda línea.
389         mov cx,9*80/2           ;CX = Cantidad de dwords a mover.
```

```
390         rep movs dword ptr es:[di],dword ptr es:[si]  ;Realizar el "scroll".
391         mov eax,07200720h       ;Dos caracteres de espacio con atributo blanco
392                                 ;sobre negro.
393         mov cl,80/2             ;CX = Cantidad de dwords a poner.
394         rep stos dword ptr es:[di]  ;Poner los espacios.
395         xchg bx,si              ;Restaurar el registro SI.
396         mov word ptr [bp+1],9*80*2 ;Indicar la nueva posición donde se deben
397                                 ;mostrar los caracteres.
398 poner_numero:
399         mov ah,07h              ;Atributo blanco sobre negro.
400         mov di,[bp+1]           ;Obtener la posición del buffer de pantalla.
401         inc si                  ;Apuntar a la primera posición del buffer.
402 poner_digito:
403         lodsb                   ;Obtener el dígito.
404         stosw                   ;Poner el carácter en buffer de pantalla.
405         cmp si,bp               ;¿Se terminó el buffer?
406         jbe poner_digito        ;Saltar si no es así.
407         mov al,179              ;Mostrar una barra vertical.
408         stosw
409         mov [bp+1],di           ;Preservar el puntero.
410         pop ebx                 ;Restaurar registros afectados.
411         pop edi
412         pop ebp
413         sti                     ;Permitir nuevamente interrupciones 8 y 9.
414         ret                     ;Fin de la subrutina.
415 posnum  equ 12*80+37
416 ;-----------------------------------------------------------------;
417 ; Tarea mitad inferior pantalla (hallar primos por potenciacion) ;
418 ;-----------------------------------------------------------------;
419 ; Como realizar potenciaciones es un proceso lento, descartamos aquellos
420 ; números que tengan algun divisor menor que 257.
421 primos_potenciacion: mov esi,5  ;Primer divisor. Los divisores serán:
422                                 ;5,7,11,13,17,19,23,25,29,31,...
423         mov ecx,257             ;Máximo divisor.
424         jmp short comparar_divisor_pot
425 ;Realizar la división de 64 bits (EBX:EDI) extendidos a 96 por 32 bits (ESI).
426 ;Lo que necesitamos nosotros es el resto (si vale cero, entonces EBX:EDI no
427 ;es primo).
428 segundo_bucle_div_64_32:
429         cmp esi,ecx             ;¿Se sobrepasó el máximo divisor?
430         ja short comenzar_potenciacion ;Comenzar con potenciación si es así.
431         xor edx,edx             ;Extender el dividendo.
432         mov eax,ebx             ;Poner la parte alta del dividendo.
433         div esi                 ;Realizar la primera parte de la división.
434         mov eax,edi             ;Poner la parte baja del dividendo.
435         div esi                 ;Realizar la segunda parte de la división.
436         and edx,edx             ;¿El resto vale cero?
437         jz short prox_num_pot   ;Si es así saltar porque el número no es primo.
438         add esi,2               ;Obtener el siguiente divisor.
439         xor edx,edx             ;Extender el dividendo.
440         mov eax,ebx             ;Poner la parte alta del dividendo.
441         div esi                 ;Realizar la primera parte de la división.
442         mov eax,edi             ;Poner la parte baja del dividendo.
443         div esi                 ;Realizar la segunda parte de la división.
444         and edx,edx             ;¿El resto vale cero?
445         jz short prox_num_pot   ;Si es así saltar porque el número no es primo.
446         add esi,4               ;Obtener el siguiente divisor.
447 comparar_divisor_pot:
448         cmp ebx,esi             ;Ver si es necesario seguir extendiendo el
449                                 ;dividendo a 96 bits.
450         jae segundo_bucle_div_64_32  ;Saltar si es así.
451 ;Realizar la división de 64 bits (EBX:EDI) por 32 bits (ESI).
452 ;Lo que necesitamos nosotros es el resto (si vale cero, entonces EBX:EDI no
453 ;es primo) y el cociente (si es menor que el divisor EBX:EDI es primo).
454 segundo_bucle_div_32_32:
```

```
455         cmp esi,ecx             ;¿Se sobrepasó el máximo divisor?
456         ja short comenzar_potenciacion  ;Comenzar con potenciación si es así.
457         mov edx,ebx             ;Poner la parte alta del dividendo.
458         mov eax,edi             ;Poner la parte baja del dividendo.
459         div esi                 ;Realizar la división.
460         and edx,edx             ;¿El resto vale cero?
461         jz short prox_num_pot   ;Si es así saltar porque el número no es primo.
462         add esi,2               ;Obtener el siguiente divisor.
463         mov edx,ebx             ;Poner la parte alta del dividendo.
464         mov eax,edi             ;Poner la parte baja del dividendo.
465         div esi                 ;Realizar la división.
466         and edx,edx             ;¿El resto vale cero?
467         jz short prox_num_pot   ;Si es así saltar porque el número no es primo.
468         add esi,4               ;Obtener el siguiente divisor.
469         cmp eax,esi             ;¿El divisor supera al cociente?
470         jae segundo_bucle_div_32_32     ;Saltar si no es así.
471 ;En este punto se sabe que EBX:EDI es primo, por lo que hay que mostrarlo.
472 mostrar_primo:
473         call mostrar_numero
474 ;Hallar el próximo valor de EBX:EDI para ver si es primo o no.
475 prox_num_pot:
476         add edi,ebp             ;Sumar el valor adecuado.
477         adc ebx,0
478         xor bp,2 xor 4          ;Cambiar el indicador de sumar 2 ó 4.
479         jmp primos_potenciacion
480 mostrar_primo_pot:
481         pop ebp                 ;Restaurar el indicador de sumar 2 ó 4.
482         jmp mostrar_primo       ;Mostrar el número primo.
483 ;
484 ; Verificar si el número es primo utilizando el método de potenciación que
485 ; figura en http://www.utm.edu/research/primes/prove2.html (tercer parágrafo)
486 ; de Chris K. Caldwell:
487 ;
488 ; Si tenemos que verificar el número impar "n", hacemos n-1=(2^s)d donde "d" es
489 ; impar y s>0: decimos que n es a-SPRP si a^d=1 (mod n) o bien (a^d)^(2^r)=-1
490 ; para algún valor r tal que 0<=r<s.
491 ;
492 ; Entonces:
493 ; - Si n < 9.080.191 es 31 y 73-SPRP entonces n es primo.
494 ; - Si n < 4.759.123.141 es 2, 7 y 61-SPRP entonces n es primo.
495 ; - Si n < 1.000.000.000.000 es 2, 13, 23 y 1662803-SPRP entonces n es primo.
496 ; - Si n < 2.152.302.898.747 es 2, 3, 5, 7 y 11-SPRP entonces n es primo.
497 ; - Si n < 3.474.749.660.383 es 2, 3, 5, 7, 11 y 13-SPRP entonces n es primo.
498 ; - Si n < 341.550.071.728.321 es 2, 3, 5, 7, 11, 13 y 17-SPRP entonces n es
499 ;   primo.
500 ;
501 ; Como este test no cubre números mayores que el último mencionado, en caso de
502 ; que aparezca algún número con esa condición deberá utilizarse el método de
503 ; la división (que para estos números es muy lento) hasta que alguien
504 ; encuentre cómo continúa la tabla.
505 ;
506 comenzar_potenciacion:
507         push ebp                ;Preservar sumando (2 ó 4).
508         push ebx                ;Preservar MSD del número a verificar.
509         push edi                ;Preservar LSD del número a verificar.
510         dec di                  ;Hallar n-1.
511         bsf ecx,edi             ;ECX = Bit menos significativo de EDI a uno.
512                                 ;      Este es el valor de s.
513         jz short LSD_n_es_cero  ;Saltar si EDI vale cero.
514         shrd edi,ebx,cl
515         shr ebx,cl              ;EBX:EDI = d = (n-1) / 2^s.
516         jmp short se_hallo_s
517 LSD_n_es_cero:
518         mov edi,ebx
519         xor ebx,ebx             ;EBX:EDI = (n-1) / 2^32
```

```
520         bsf ecx,edi             ;ECX = Bit menos significativo de EDI a uno.
521                                 ;      Este es el valor de (s-32).
522         shr edi,cl              ;EBX:EDI = (n-1) / 2^(32+s-32)
523                                 ;        = (n-1) / 2^s.
524         add cl,32               ;ECX = s.
525 se_hallo_s:
526         mov valor_s,cx          ;Almacenar el valor de s.
527 ;
528 ; Para realizar el cálculo de la potenciación más abajo, se utilizará el
529 ; método de elevar al cuadrado y multiplicar. Para ello, partiendo del bit
530 ; más significativo puesto a uno, se barrerán los diferentes bits de "d"
531 ; (sin incluir el bit a uno más significativo) hasta llegar al bit cero. Si el
532 ; bit vale cero, se debe elevar al cuadrado, si vale uno, además de elevar al
533 ; cuadrado hay que multiplicar por la base. Por esto hay que hallar la cantidad
534 ; de bits a procesar y hay que justificar el número "d" a la izquierda (así se
535 ; pueden barrer fácilmente los diferentes bits realizando desplazamientos a la
536 ; izquierda).
537 ;
538         bsr ecx,ebx             ;ECX = Bit más significativo del MSD de "d"
539                                 ;a uno.
540         jz short MSD_d_es_cero ;Saltar si el MSD de "d" vale cero.
541 ; Ahora hay que desplazar "d" hacia la izquierda (32-CL) bits. Para ello,
542 ; se aprovechará el hecho de que los procesadores 80386 y posteriores toman
543 ; la cuenta de desplazamiento módulo 32. De esta manera, desplazar (32-CL) bits
544 ; hacia la izquierda se puede codificar como si se desplazara -CL bits.
545         neg cl                  ;Desplazar hacia la izquierda -CL bits.
546         jz short bit_0_MSD_d_a_uno ;Si CL = 0 el bit 0 vale uno, por lo que el
547                                 ;MSD no tiene bits significativos.
548         shld ebx,edi,cl         ;Desplazar hacia la izquierda EDX:EAX.
549         shl edi,cl
550         sub cl,32               ;CL = -(Cantidad de bits significativos).
551         jmp short hallar_bits_d ;Hallar contador (negando CL).
552 bit_0_MSD_d_a_uno:
553         mov ebx,edi             ;Poner en el MSD los bits significativos.
554         mov cl,32               ;Indicar que hay 32 bits significativos.
555         jmp short almacenar_d_ajustado ;Almacenar d ajustado y cant de bits.
556 MSD_d_es_cero:
557         mov ebx,edi             ;Mover los bits significativos hacia el MSD.
558         bsr ecx,ebx             ;ECX = Bit más significativo del LSD de d
559                                 ;a uno.
560         neg cl                  ;Desplazar hacia la izquierda -CL bits.
561         jz short d_uno          ;Si CL = 0 el bit 0 vale uno, por lo que d
562                                 ;vale uno.
563         shl ebx,cl              ;Desplazar el factor hacia la izquierda.
564 hallar_bits_d:
565         neg cl                  ;CL = Cantidad de bits significativos de d.
566 almacenar_d_ajustado:
567         mov MSD_d_ajustado,ebx  ;Almacenar el valor de "d" ajustado.
568         mov LSD_d_ajustado,edi
569 d_uno:
570         mov bits_d,cx           ;Almacenar la cantidad de dígitos
571                                 ;significativos de "d".
572         pop edi                 ;Restaurar LSD del número a verificar.
573         pop ebx                 ;Restaurar MSD del número a verificar.
574         and ebx,ebx             ;¿El número entra en un dword?
575         jnz short verif_2do_caso ;Saltar si no es así.
576         cmp edi,9080191         ;Ver si el número entra dentro del primer caso.
577         jae short es_2do_caso   ;Saltar si no es así.
578         mov al,31               ;Comprobar si el número es 31-SPRP.
579         call verificar_SPRP
580         mov al,73               ;Comprobar si el número es 73-SPRP.
581         call verificar_SPRP
582         jmp mostrar_primo_pot   ;Si lo es, mostrar el número primo.
583 verif_2do_caso:
584         cmp ebx,1               ;Ver si el número entra en el segundo caso.
```

```
585         jnz short test_2do_caso
586         cmp edi,1BAA74C5h
587 test_2do_caso:
588         jae short verif_3er_caso ;Saltar si no es así.
589 es_2do_caso:
590         mov al,2                ;Comprobar si el número es 2-SPRP.
591         call verificar_SPRP
592         mov al,7                ;Comprobar si el número es 7-SPRP.
593         call verificar_SPRP
594         mov al,61               ;Comprobar si el número es 61-SPRP.
595         call verificar_SPRP
596         jmp mostrar_primo_pot   ;Si lo es, mostrar el número primo.
597 verif_3er_caso:
598         mov al,2                ;Comprobar si el número es 2-SPRP.
599         call verificar_SPRP
600         cmp ebx,0E8h            ;Ver si el número entra en el tercer caso.
601         jnz short test_3er_caso
602         cmp edi,0D4A51000h
603 test_3er_caso:
604         jae short es_4to_caso   ;Saltar si no es así.
605         mov al,13               ;Comprobar si el número es 13-SPRP.
606         call verificar_SPRP
607         mov al,23               ;Comprobar si el número es 23-SPRP.
608         call verificar_SPRP
609         mov eax,1662803         ;Comprobar si el número es 1.662.803-SPRP.
610         call verificar_dword_SPRP
611         jmp mostrar_primo_pot
612 es_4to_caso:
613         mov al,3                ;Comprobar si el número es 3-SPRP.
614         call verificar_SPRP
615         mov al,5                ;Comprobar si el número es 5-SPRP.
616         call verificar_SPRP
617         mov al,7                ;Comprobar si el número es 7-SPRP.
618         call verificar_SPRP
619         mov al,11               ;Comprobar si el número es 11-SPRP.
620         call verificar_SPRP
621         cmp ebx,000001F5h       ;Ver si el número entra en el cuarto caso.
622         jnz short test_11
623         cmp edi,1F3FEE3Bh
624 test_11:
625         jb mostrar_primo_pot    ;Si es así el número es primo.
626         mov al,13               ;Comprobar si el número es 13-SPRP.
627         call verificar_SPRP
628         cmp ebx,00000329h       ;Ver si el número entra en el quinto caso.
629         jnz short test_13
630         cmp edi,07381CDFh
631 test_13:
632         jb mostrar_primo_pot    ;Si es así el número es primo.
633         mov al,17               ;Comprobar si el número es 17-SPRP.
634         call verificar_SPRP
635         cmp ebx,000136A3h       ;Ver si el número entra en el sexto caso.
636         jnz short test_17
637         cmp edi,52B2C8C1h
638 test_17:
639         jb mostrar_primo_pot    ;Si es así el número es primo.
640         mov ecx,-1              ;Forzar divisiones sucesivas.
641         mov esi,257             ;Comprobar a partir del último divisor probado.
642         pop ebp                 ;Restaurar sumando (2 ó 4).
643         jmp comparar_divisor_pot ;Verificar si el número es primo.
644 ;
645 ; Subrutina para comprobar si EBX:EDI es AL-SPRP.
646 verificar_SPRP:
647         and eax,0FFh            ;EAX <- Base SPRP.
648 ;
649 ; Subrutina para comprobar si EBX:EDI es EAX-SPRP.
```

```
650 verificar_dword_SPRP:
651         test ebx,ebx            ;¿El número a verificar tiene más de 32 bits?
652         jnz short verificar_SPRP_64_bits ;Si es así, utilizar rutina de 64 bits
653         mov ebp,eax             ;EBP <- Base SPRP.
654         mov cx,bits_d           ;Obtener la cantidad de bits de d.
655         jcxz no_hay_ciclo_pot_32_bits ;Saltar si d no tiene bits significativos
656                                 ;Esto ocurre si d = 1 => n = 2^k+1 con k>=0.
657         mov ebx,MSD_d_ajustado  ;Obtener el valor de d justificado a la izq.
658 ; Con el siguiente ciclo se hallará EAX ^ d (mod EDI) (primera parte del test).
659 ciclo_pot_32_bits:
660         mul eax                 ;Elevar al cuadrado.
661         div edi                 ;Hallar módulo EDI.
662         mov eax,edx             ;EAX = Resto de la división.
663         shl ebx,1               ;Obtener siguiente bit de d.
664         jnc short fin_ciclo_pot_32 ;Si vale cero, no se debe multiplicar por
665                                 ;la base de SPRP.
666         mul ebp                 ;Multiplicar por la base.
667         div edi                 ;Hallar módulo EDI.
668         mov eax,edx             ;EAX = Resto de la división.
669 fin_ciclo_pot_32:
670         loop ciclo_pot_32_bits  ;Seguir procesando el resto de los bits de d.
671         xor ebx,ebx             ;Poner EBX a cero como estaba antes.
672 no_hay_ciclo_pot_32_bits:
673         cmp eax,1               ;¿La base ^ d (mod n) = 1?
674         jz short SPRP_OK        ;Si es así, se cumple el test.
675         mov cx,valor_s          ;Obtener el valor de s.
676 ; En el siguiente ciclo se elevará al cuadrado hasta s veces el resultado de
677 ; la primera parte del test. Si en algún momento el resultado es -1, el test
678 ; se cumple (segunda parte del test de SPRP).
679 ciclo_cuad_32_bits:
680         inc eax                 ;Si EAX = -1 (mod EDI), deberá ser EAX = EDI-1,
681         cmp eax,edi             ;ya que 0 <= EAX < EDI.
682         jz short SPRP_OK        ;Si EAX = -1 (mod EDI) se cumple el test.
683         dec eax                 ;Restaurar el valor.
684         mul eax                 ;Elevar al cuadrado.
685         div edi                 ;Hallar módulo EDI.
686         mov eax,edx             ;EAX = Resto de la división.
687         loop ciclo_cuad_32_bits ;Cerrar el ciclo.
688 ; Si no se cumplió ninguna de las dos partes del test, el número es compuesto.
689 es_compuesto:
690         pop ax                  ;No retornar a rutina de verificación SPRP.
691         pop ebp                 ;Restaurar sumando (2 ó 4).
692         jmp prox_num_pot        ;Hallar el próximo número a verificar.
693 SPRP_OK:
694         ret                     ;Fin de la subrutina.
695 verificar_SPRP_64_bits:
696         mov ecx,LSD_d_ajustado  ;Copiar el valor de d ajustado a otro lugar
697         mov LSD_d_aux,ecx       ;ya que se va a modificar (este valor se va
698         mov ecx,MSD_d_ajustado  ;a necesitar posteriormente).
699         mov MSD_d_aux,ecx
700         mov base_SPRP,eax       ;Preservar la base de SPRP.
701         xor edx,edx             ;EDX:EAX = Base de SPRP.
702         mov cx,bits_d           ;Obtener la cantidad de bits de d.
703         jcxz no_hay_ciclo_pot_64_bits ;Saltar si d no tiene bits significativos
704                                 ;Esto ocurre si d = 1 => n = 2^k+1 con k>=0.
705 ; Con el siguiente ciclo se hallará EAX ^ d (mod EDI) (primera parte del test).
706 ciclo_pot_64_bits:
707         shl LSD_d_aux,1         ;Obtener siguiente bit de d.
708         rcl MSD_d_aux,1
709         mov esi,eax
710         mov ebp,edx             ;EBP:ESI <- Multiplicador actual.
711 ; Si el bit vale cero, sólo hay que elevar al cuadrado, mientras que si vale
712 ; uno, aparte de elevar al cuadrado, hay que multiplicar por la base de SPRP.
713         jnc short multiplicador_ok ;El bit vale cero (no multiplicar por base)
714         push edx                ;EDX:EAX <- EDX:EAX * base_SPRP
```

```
715         mul base_SPRP
716         push edx
717         xchg ebp,eax
718         mul base_SPRP
719         pop edx
720         add edx,eax
721         mov eax,ebp
722         pop ebp
723 multiplicador_ok:
724         call mult_modulo_EBX_EDI ;EDX:EAX <- EBP:ESI * EDX:EAX  (mod EBX:EDI).
725         loop ciclo_pot_64_bits  ;Seguir procesando el resto de los bits de d.
726 no_hay_ciclo_pot_64_bits:
727         test edx,edx            ;¿La base ^ d (mod n) = 1?
728         jnz short hallar_cuadrados
729         cmp eax,1
730         jz SPRP_OK              ;Si es así, se cumple el test.
731 hallar_cuadrados:
732         mov cx,valor_s          ;Obtener el valor de s.
733 ; En el siguiente ciclo se elevará al cuadrado hasta s veces el resultado de
734 ; la primera parte del test. Si en algún momento el resultado es -1, el test
735 ; se cumple (segunda parte del test de SPRP).
736 ciclo_cuad_64_bits:
737         mov esi,edi             ;¿Es EDX:EAX = -1 (mod EBX:EDI)?
738         mov ebp,ebx
739         stc
740         sbb esi,eax
741         sbb ebp,edx
742         or ebp,esi
743         jz SPRP_OK              ;Si es así se cumple el test.
744         mov ebp,edx
745         mov esi,eax
746         call mult_modulo_EBX_EDI ;Elevar al cuadrado modulo EBX:EDI.
747         loop ciclo_cuad_64_bits ;Cerrar el ciclo.
748         jmp es_compuesto        ;Si no se cumplió el test, el número es
749                                 ;compuesto.
750
751 ; Subrutina para hallar EBP:ESI * EDX:EAX  (mod EBX:EDI) -> EDX:EAX.
752 mult_modulo_EBX_EDI:
753         mov res_LO,edi
754         sub res_LO,esi
755         mov res_HI,ebx
756         sbb res_HI,ebp          ;res = - EBP:ESI (mod EBX:EDI)
757         push cx                 ;Preservar contador.
758         bsr ecx,edx             ;ECX = Bit más significativo del MSD del factor
759                                 ;a uno.
760         jz short MSD_factor_es_cero ;Saltar si el MSD del factor vale cero.
761 ; Ahora hay que desplazar el factor hacia la izquierda (32-CL) bits. Para ello,
762 ; se aprovechará el hecho de que los procesadores 80386 y posteriores toman
763 ; la cuenta de desplazamiento módulo 32. De esta manera, desplazar (32-CL) bits
764 ; hacia la izquierda se puede codificar como si se desplazara -CL bits.
765         neg cl                  ;Desplazar hacia la izquierda -CL bits.
766         jz short bit_0_MSD_a_uno ;Si CL = 0 el bit 0 vale uno, por lo que el
767                                 ;MSD no tiene bits significativos.
768         shld edx,eax,cl         ;Desplazar hacia la izquierda EDX:EAX.
769         shl eax,cl
770         sub cl,32               ;CL = -(Cantidad de bits significativos).
771         jmp short hallar_contador ;Hallar contador (negando CL).
772 bit_0_MSD_a_uno:
773         mov edx,eax             ;Poner en el MSD los bits significativos.
774         mov cl,32               ;Indicar que hay 32 bits significativos.
775         jmp short ciclo_interno_64 ;Comenzar con el ciclo de multiplicación.
776 MSD_factor_es_cero:
777         mov edx,eax             ;Mover los bits significativos hacia el MSD.
778         bsr ecx,edx             ;ECX = Bit más significativo del LSD del factor
779                                 ;a uno.
```

```
780         jz short factor_cero    ;Si el LSD vale cero, el factor vale cero.
781         neg cl                  ;Desplazar hacia la izquierda -CL bits.
782         jz short factor_uno     ;Si CL = 0 el bit 0 vale uno, por lo que el
783                                 ;factor vale uno.
784         shl edx,cl              ;Desplazar el factor hacia la izquierda.
785 hallar_contador:
786         neg cl                ;CL = Cantidad de bits significativos del factor.
787 ; Ciclo para realizar la multiplicación.
788 ciclo_interno_64:
789         add esi,esi             ;Multiplicar factor por dos.
790         adc ebp,ebp
791         sub esi,edi             ;Hallar factor módulo EBX:EDI.
792         sbb ebp,ebx
793         jnc short ver_bit_factor ;Saltar si el valor hallado no es negativo.
794         add esi,edi             ;Hallar factor módulo EBX:EDI (valor positivo).
795         adc ebp,ebx
796 ver_bit_factor:
797         add eax,eax             ;Hallar el siguiente bit del factor.
798         adc edx,edx
799         jnc short fin_ciclo_interno_64 ;Saltar si el bit vale cero.
800         sub esi,res_LO          ;Sumar el factor inicial (módulo EBX:EDI).
801         sbb ebp,res_HI
802         jnc short fin_ciclo_interno_64 ;Saltar si es cero o positivo.
803         add esi,edi             ;Hallar factor módulo EBX:EDI (valor positivo).
804         adc ebp,ebx
805 fin_ciclo_interno_64:
806         loop ciclo_interno_64   ;Cerrar ciclo.
807 factor_uno:
808         mov eax,esi             ;Poner el factor en los registros resultado.
809         mov edx,ebp
810 factor_cero:
811         pop cx                  ;Restaurar el contador.
812         ret                     ;Fin de la subrutina.
813 ;-------------------------------------------------;
814 ; Manejador de la interrupción de teclado (INT 9) ;
815 ;-------------------------------------------------;
816 ; Nótese el manejo de la tecla Pausa, cuyo scan code es: E1 1D 45 E1 9D C5.
817 ; Esta tecla es la única cuyo prefijo es E1.
818 int9han:in al,60h               ;AL = Tecla apretada o soltada según bit 7.
819         test estado,SCAN_CODE_E1 ;¿El scan code anterior fue E1?
820         jnz short procesar_E1   ;Saltar si es así.
821         cmp al,0E1h             ;¿Se está apretando la tecla Pausa?
822         jz short apretando_Pausa
823         and estado,not APRETO_PAUSA ;Salir del modo Pausa (si estaba).
824         test estado,SCAN_CODE_E0 ;¿El scan code anterior fue E0?
825         jnz procesar_E0         ;Saltar si es así.
826         mov bx,offset tabla     ;Apuntar a la tabla de teclas.
827         mov cx,30               ;Cantidad de teclas reconocidas.
828 int9han1:cmp al,[bx]            ;Ver si se apretó la tecla correspondiente
829                                 ;a la entrada de la tabla.
830         jz short int9han2       ;Saltar si es así.
831         add bx,3                ;Apuntar a la próxima entrada de la tabla.
832         loop int9han1           ;Terminar el bucle.
833         jmp short int9han3      ;No procesar la tecla.
834 int9han2:call word ptr [bx+1]   ;Llamar a la subrutina correspondiente a la
835                                 ;tecla presionada.
836 int9han3:mov al,61h            ;Indicarle al PIC un fin de interrupción
837         out 20h,al              ;específico de IRQ 1 (interrupción 0).
838         iretd                   ;Terminar la interrupción.
839 ;Cuando se ejecute la próxima INT 9 va a ejecutar a partir de este punto.
840         jmp int9han             ;Volver a ejecutar la interrupción.
841 apretando_Pausa:
842         or estado,SCAN_CODE_E1  ;Indicar que llegó el scan code E1.
843         jmp int9han3            ;Terminar la interrupción.
844 procesar_E1:
```

```
845         add estado,SCAN_DESPUES_E1 ;Incrementar la cantidad de scan codes
846                                    ;después del E1.
847         jnc short int9han3         ;Todavía no llegaron todos los scan codes de
848                                    ;la tecla Pausa.
849         or estado,APRETO_PAUSA     ;Indicar que se apretó la tecla Pausa.
850         jmp int9han3               ;Fin de la interrupción.
851 ;Procesamiento de flecha arriba y flecha abajo.
852 tics_tarea2 equ <byte ptr TSS_int8.DX_val>
853 flecha: mov al,tics_tarea2         ;AL = Cantidad de tics de tarea 1.
854         add al,dl                  ;Sumar o restar 1 (dependiendo de la flecha)
855         cmp al,20                  ;¿Está fuera del límite?
856         ja int9han3                ;Terminar la interrupción si es así.
857         mov tics_tarea2,al         ;Actualizar la cantidad de tics.
858         mov bx,11*160+8*2          ;Mostrar en la pantalla el porcentaje de la
859         call mos_porc              ;tarea 1.
860         mov al,20
861         sub al,tics_tarea2
862         mov bx,13*160+8*2          ;Mostrar en la pantalla el porcentaje de la
863         call mos_porc              ;tarea 2.
864         jmp int9han3               ;Fin de la interrupción.
865 ;Subrutina para mostrar porcentaje: BX = Posición en la pantalla.
866 ;                                      AL = Cantidad de tics (0-20).
867 mos_porc:shr al,1                  ;AL = Decenas.
868         rcr ah,1                   ;Bit 7 de AH = cero o cinco.
869         cmp al,10                  ;¿Las decenas valen diez (mostrar dos dígitos)?
870         jnz short mos_porc1        ;Saltar si no es así.
871         mov byte ptr es:[bx],"1" ;Mostrar el primer dígito.
872         mov byte ptr es:[bx+2],"0" ;Mostrar el segundo dígito.
873         add bx,4                   ;Ir a la posición de las unidades.
874         jmp short mos_porc2        ;Mostrar las unidades.
875 mos_porc1:and al,al                ;¿Hay decenas?
876         jz short mos_porc2         ;Saltar si no hay (no mostrarlas).
877         add al,"0"                 ;Convertir las decenas a ASCII.
878         mov es:[bx],al             ;Poner el dígito en la pantalla.
879         add bx,2                   ;Apuntar a la posición de las unidades.
880 mos_porc2:and ah,ah                ;Ver cuánto valen las unidades.
881         mov al,"0"                 ;Suponer que valen cero.
882         jns short mos_porc3        ;Saltar si valen cero.
883         mov al,"5"                 ;Las unidades valen cinco.
884 mos_porc3:mov es:[bx],al           ;Poner las unidades en pantalla.
885         mov byte ptr es:[bx+2],"%" ;Mostrar el símbolo de porcentaje.
886         mov byte ptr es:[bx+4]," " ;Borrar el carácter de la derecha.
887 fin_subr:ret                       ;Terminar la subrutina.
888 tecBREAK:
889         test estado,CAMBIANDO_DIV or CAMBIANDO_POT
890                                    ;¿Se está ingresando un número?
891         mov estado,APRETO_ESC      ;Indicar que se apretó la tecla ESC (bit 4=1)
892         jz int9han3                ;Terminar si no se está ingresando un número.
893         call cancelar_entrada      ;Cancelar el ingreso de número.
894         jmp int9han3               ;Terminar el manejador de interrupción.
895 procesar_E0:and estado,not SCAN_CODE_E0 ;Indicar que ya se está procesando E0.
896         cmp al,48h                 ;¿La tecla presionada es la flecha arriba?
897         mov dl,1
898         jz short flecha            ;Saltar si es así.
899         cmp al,50h                 ;¿La tecla presionada es la flecha abajo?
900         mov dl,255
901         jz short flecha            ;Saltar si es así.
902         cmp al,46h                 ;¿La tecla presionada es CTRL-BREAK?
903         jz tecBREAK                ;Saltar si es así.
904         cmp al,1Ch                 ;¿La tecla presionada es el Enter gris?
905         jnz int9han3               ;Saltar si no es así.
906         call tecENT                ;Convertir el número entrado a binario.
907         jmp int9han3               ;Terminar el manejador de interrupción.
908 tecENT: test estado,CAMBIANDO_POT or CAMBIANDO_DIV
909                                    ;Ver si se está entrando un número.
```

```
910         jz finsubr              ;Salir de la subrutina si no es así.
911         mov si,posnum*2         ;Posición del primer dígito en la pantalla.
912         xor eax,eax             ;EAX = Va a contener el dígito a sumar.
913         xor ebx,ebx             ;ECX:EBX = Contiene el número que se está
914         xor ecx,ecx             ;formando.
915 tecENT1:mov ebp,ebx             ;EDI:EBP <- ECX:EBX.
916         mov edi,ecx
917         mov al,es:[si]          ;Obtener el dígito en ASCII.
918         add si,2                ;Apuntar al siguiente dígito en la pantalla.
919         sub al,"0"              ;Convertir a binario.
920         jc short tecENT2        ;Saltar si no es un dígito (se acabó).
921         add ebx,ebx             ;ECX:EBX <- ECX:EBX * 10
922         adc ecx,ecx             ;(correr un dígito hacia la izquierda).
923         add ebx,ebx
924         adc ecx,ecx
925         add ebx,ebp
926         adc ecx,edi
927         add ebx,ebx
928         adc ecx,ecx
929         add ebx,eax             ;ECX:EBX <- ECX:EBX + EAX
930         adc ecx,0               ;(sumar el dígito).
931         jmp tecENT1             ;Procesar el siguiente dígito.
932 ;En este momento el par de registros ECX:EBX contienen el número entrado.
933 tecENT2:xor edx,edx             ;Hallar el resto de ECX:EBX / 6.
934         mov eax,ecx
935         mov edi,6
936         div edi
937         mov eax,ebx
938         div edi
939         mov ebp,2               ;Sumando para el caso en que el resto valga 5.
940         cmp dl,5                ;¿El resto vale 5?
941         jz short tecENT3        ;Saltar si es así.
942         mov bp,4                ;Sumando para el caso en que el resto valga 1.
943         cmp dl,1                ;¿El resto vale 1?
944         jz short tecENT3        ;Saltar si es así.
945         add ebx,1               ;Seguro que el número no va a ser primo, por lo
946         adc ecx,0               ;que hay que incrementarlo.
947         jmp tecENT2             ;Hacer lo mismo con el número incrementado.
948 tecENT3:test estado,CAMBIANDO_DIV ;¿Qué número se estaba cambiando?
949         mov si,offset TSS_primos_potenciacion ;Suponer que era el de
950                                               ;primos_potenciacion.
951         jz short tecENT4          ;Saltar si se estaba cambiando el de
952                                   ;primos_potenciacion.
953         mov si,offset TSS_primos_division ;Se estaba cambiando el de
954                                   ;primos_division.
955 ;Mover primero la doble palabra (32 bits) más significativa y luego la
956 ;doble palabra menos significativa.
957 tecENT4:mov dword ptr [si+EBX_value],ecx
958         mov dword ptr [si+EDI_value],ebx
959         mov word ptr [si+EIP_value],offset primos_potenciacion
960         mov word ptr [si+ESP_value],offset pila_primos_potenciacion + 100h - base
961         jz short offset_puesto
962         mov word ptr [si+EIP_value],offset primos_division
963         mov word ptr [si+ESP_value],offset pila_primos_division + 100h - base
964 offset_puesto:
965         mov [si+EBP_value],ebp
966         and estado,not (CAMBIANDO_POT or CAMBIANDO_DIV)
967                                   ;Indicar que no se está entrando un número.
968         mov cx,26*80              ;Poner el cursor fuera de la pantalla así no
969         call ponercur             ;se lo ve.
970         mov bx,offset texto2      ;Mostrar el texto original.
971         jmp short mostex
972 tecF1:  and estado,not RUN_DIVISION  ;Detener primos_division.
973         ret
974 tecF2:  or estado,RUN_DIVISION   ;Continuar primos_division.
```

```
 975         ret
 976 tecF3:  mov al,1                ;Indicar que se cambiará el número de la
 977                                 ;tarea primos_division.
 978 camnum: and estado,not (CAMBIANDO_POT or CAMBIANDO_DIV)
 979                                 ;Indicar en la variable de estado cuál es el
 980         or estado,al            ;número que se está cambiando.
 981         mov bx,offset texto4    ;Mostrar el texto "Nuevo número:" en pantalla.
 982         call mostex

 983         mov cx,posnum           ;Situar el cursor a la derecha del texto.
 984 ;Subrutina para situar el cursor en una posición determinada de la pantalla.
 985 ;Entrada: CX = Línea * 80 + Columna del cursor.
 986 ponercur:mov dx,port6845        ;DX = Port de control del controlador de video.
 987         mov al,0eh              ;Indicarle que hay que escribir la parte alta
 988         out dx,al               ;(MSB) de la posición del cursor.
 989         jmp $+2
 990         jmp $+2
 991         inc dx                  ;Apuntar al port de datos del 6845.
 992         mov al,ch               ;Obtener el valor a enviar al 6845.
 993         out dx,al               ;Escribir el valor.
 994         jmp $+2
 995         jmp $+2
 996         dec dx                  ;Apuntar al port de control del 6845,
 997         mov al,0fh              ;Indicarle que hay que escribir la parte baja
 998         out dx,al               ;(LSB) de la posición del cursor.
 999         jmp $+2
1000         jmp $+2
1001         inc dx                  ;Apuntar al port de datos del 6845.
1002         mov al,cl               ;Obtener el valor a enviar al 6845.
1003         out dx,al               ;Escribir el valor.
1004         ret
1005 tecF4:  and estado,not RUN_POTEN ;Detener primos_potenciacion.
1006         ret
1007 tecF5:  or estado,RUN_POTEN     ;Continuar primos_potenciacion.
1008         ret
1009 tecF6:  mov al,CAMBIANDO_POT    ;Indicar que se cambiará el número de la
1010                                 ;tarea primos_potenciacion.
1011         jmp camnum              ;Ir a cambiar el número.
1012 tecESC: test estado,CAMBIANDO_DIV or CAMBIANDO_POT
1013                                 ;Ver si se está entrando un número.
1014         jnz short cancelar_entrada  ;Saltar si es así.
1015 tecESC1:mov estado,APRETO_ESC   ;Indicar que se apretó la tecla ESC (bit 4=1)
1016         ret                     ;y que ejecute la tarea nula (bits 3,2=00)
1017 cancelar_entrada:
1018         mov cx,26*80            ;Poner el cursor fuera de la pantalla así no
1019         call ponercur           ;se lo ve.
1020         and estado,not (CAMBIANDO_DIV or CAMBIANDO_POT)
1021                                 ;Indicar que no se está entrando un número.
1022         mov bx,offset texto2    ;Mostrar el texto original.
1023 ;Subrutina para mostrar texto. Entrada: BX = Puntero al texto.
1024 ;                                       SI = Posición en la pantalla.
1025 mostex: mov si,[bx]             ;Obtener la posición de pantalla.
1026         add bx,2                ;Apuntar al primer carácter del texto.
1027 mostex1:mov al,[bx]             ;Obtener el carácter del texto.
1028         mov es:[si],al          ;Ponerlo en la pantalla.
1029         inc bx                  ;Incrementar el puntero del texto.
1030         add si,2                ;Incrementar el puntero del buffer de pantalla.
1031         cmp byte ptr [bx],0     ;¿Se acabó el texto?
1032         jnz mostex1             ;Saltar si no es así.
1033 finsubr:ret                     ;Fin de la subrutina.
1034 tecnum: dec cx                  ;CL = Dígito presionado.
1035         cmp cl,10               ;Si se apretó un dígito de la línea superior
1036         jb short tecnum1        ;el número está bien, mientras que si se
1037         sub cl,10               ;apretó del pad numérico hay que restar 10.
1038 tecnum1:add cl,"0"              ;Convertir el dígito a ASCII.
```

```
1039         call obtencur           ;SI = Línea * 80 + Columna (posición cursor)
1040         cmp si,posnum+19        ;Ver si se llegó al límite derecho.
1041         jz finsubr              ;Salir de la subrutina si es así.
1042         push cx                 ;Preservar el dígito ASCII en la pila.
1043         mov cx,si               ;CX = Posición del cursor.
1044         inc cx
1045         call ponercur           ;Moverlo un lugar a la derecha.
1046         add si,si               ;Convertir a posición de buffer de pantalla.
1047         pop ax                  ;Obtener el dígito en ASCII.
1048         mov es:[si],al          ;Poner el dígito en la pantalla.
1049         ret                     ;Fin de la subrutina.
1050 tecBACK:test estado,CAMBIANDO_POT or CAMBIANDO_DIV
1051                                 ;Ver si se está entrando un número.
1052         jz finsubr              ;Salir de la subrutina si no es así.
1053         call obtencur           ;SI = Línea * 80 + Columna (posición cursor)
1054         cmp si,posnum           ;Ver si se llegó al límite izquierdo.
1055         jz finsubr              ;Salir de la subrutina si es así.
1056         dec si
1057         mov cx,si               ;CX = Nueva posición del cursor.
1058         call ponercur           ;Mover el cursor un lugar hacia la izquierda.
1059         add si,si               ;Convertir a posición de buffer de pantalla.
1060         mov byte ptr es:[si]," ";Poner un espacio en dicha posición.
1061         ret                     ;Fin de la subrutina.
1062 tecE0:  or estado,SCAN_CODE_E0  ;Indicar que el código recibido fue E0.
1063         ret                     ;Fin de la subrutina.
1064 ;Tabla de teclas y sus subrutinas.
1065 ;La tecla está representada por su "scan code".
1066 ;Las teclas que tienen dos bytes de scan code (el primero siempre es E0)
1067 ;se procesan aparte.
1068 tabla   db 3Bh              ;Tecla F1
1069         dw tecF1
1070         db 3Ch              ;Tecla F2
1071         dw tecF2
1072         db 3Dh              ;Tecla F3
1073         dw tecF3
1074         db 3Eh              ;Tecla F4
1075         dw tecF4
1076         db 3Fh              ;Tecla F5
1077         dw tecF5
1078         db 40h              ;Tecla F6
1079         dw tecF6
1080         db 01h              ;Tecla Esc
1081         dw tecESC
1082         db 1Ch              ;Tecla Enter o Enter gris (con E0 adelante)
1083         dw tecENT
1084         db 0Eh              ;Tecla Backspace
1085         dw tecBACK
1086         db 0E0h             ;Código especial del teclado expandido.
1087         dw tecE0
1088         db 49h              ;Tecla 9 (pad numérico)
1089         dw tecnum
1090         db 48h              ;Tecla 8 (pad numérico) o flecha arriba (con E0)
1091         dw tecnum
1092         db 47h              ;Tecla 7 (pad numérico)
1093         dw tecnum
1094         db 4Dh              ;Tecla 6 (pad numérico)
1095         dw tecnum
1096         db 4Ch              ;Tecla 5 (pad numérico)
1097         dw tecnum
1098         db 4Bh              ;Tecla 4 (pad numérico)
1099         dw tecnum
1100         db 51h              ;Tecla 3 (pad numérico)
1101         dw tecnum
1102         db 50h              ;Tecla 2 (pad numérico) o flecha abajo (con E0)
1103         dw tecnum
```

```
1104         db 4Fh               ;Tecla 1 (pad numérico)
1105         dw tecnum
1106         db 52h              ;Tecla 0 (pad numérico)
1107         dw tecnum
1108         db 0Ah              ;Tecla 9
1109         dw tecnum
1110         db 09h              ;Tecla 8
1111         dw tecnum
1112         db 08h              ;Tecla 7
1113         dw tecnum
1114         db 07h              ;Tecla 6
1115         dw tecnum
1116         db 06h              ;Tecla 5
1117         dw tecnum
1118         db 05h              ;Tecla 4
1119         dw tecnum
1120         db 04h              ;Tecla 3
1121         dw tecnum
1122         db 03h              ;Tecla 2
1123         dw tecnum
1124         db 02h              ;Tecla 1
1125         dw tecnum
1126         db 0Bh              ;Tecla 0
1127         dw tecnum
1128 ;Subrutina para obtener la posición del cursor en la pantalla.
1129 ;Salida: Registro SI = Línea * 80 + Columna del cursor.
1130 obtencur:mov dx,port6845     ;DX = Port de control del controlador de video.
1131         mov al,0Eh          ;Indicarle que hay que leer la parte alta
1132         out dx,al           ;(MSB) de la posición del cursor.
1133         jmp $+2
1134         jmp $+2
1135         inc dx              ;Apuntar al port de datos del 6845.
1136         in al,dx            ;Leer el valor.
1137         mov ah,al           ;Ponerlo en la parte alta de AX.
1138         jmp $+2
1139         jmp $+2
1140         dec dx              ;Apuntar al port de control del 6845,
1141         mov al,0Fh          ;Indicarle que hay que leer la parte baja
1142         out dx,al           ;(LSB) de la posición del cursor.
1143         jmp $+2
1144         jmp $+2
1145         inc dx              ;Apuntar al port de datos del 6845.
1146         in al,dx            ;Leer el valor.
1147         mov si,ax           ;Poner el resultado en SI.
1148         ret                 ;Fin de la subrutina.
1149 ;-------------------------------------------------;
1150 ; Manejador de la interrupción del timer (INT 8) ;
1151 ;-------------------------------------------------;
1152 int8han:mov al,60h          ;Indicarle al PIC un fin de interrupción
1153         out 20h,al          ;específico de la IRQ 0 (interrupción 8).
1154         jmp $+2
1155         jmp $+2
1156 ;Habilitar la interrupción de teclado y deshabilitar la del timer. Esto se hace
1157 ;así porque como en el descriptor de la tabla de interrupción IDT hay una
1158 ;compuerta de tarea, la interrupción no es reentrante. Si se llegara a ocurrir
1159 ;nuevamente la interrupción ocurriría una excepción 13.
1160         mov al,0FDh         ;Habilitar la interrupción de teclado.
1161         out 21h,al          ;y deshabilitar la del timer.
1162         loop tiempo_no_cero  ;Decrementar el contador de tiempo de tarea.
1163         mov cl,20           ;Si llegó a cero ponerlo al valor inicial.
1164 tiempo_no_cero:
1165         test estado,APRETO_PAUSA ;¿Se apretó la tecla Pausa?
1166         mov ax,TSS_inic_sel  ;Seleccionar la tarea nula.
1167         jnz short nueva_tarea ;Saltar si es así.
1168         mov al,estado       ;Bits 3 y 2 del estado: 00 = nada,
```

```
1169                                ;01 = primos_division, 10 = primos_potenciacion,
1170                               ;11 = ambas tareas.
1171         cmp cl,dl             ;¿Tarea primos_division o tarea primos_potenc?
1172         ja short ir_a_primos_pot ;Ir a tarea primos_potenciacion.
1173         test al,RUN_DIVISION ;Ver si la tarea primos_division está activa.
1174         mov ax,TSS_pri1_sel  ;Selector de la tarea primos_division.
1175         jmp short tarea_activa?
1176 ir_a_primos_pot:
1177         test al,RUN_POTEN    ;Ver si la tarea primos_potenciacion está activa.
1178         mov ax,TSS_pri2_sel  ;Selector de la tarea primos_potenciacion.
1179 tarea_activa?:
1180         jnz short nueva_tarea ;Saltar si es así.
1181         mov al,TSS_inic_sel  ;Selector de la tarea nula.
1182 nueva_tarea:
1183         mov salto,ax         ;Almacenar el selector de TSS para saltar allí.
1184         mov al,0FEh          ;Indicarle al PIC que deshabilite la
1185         out 21h,al           ;interrupción de teclado.
1186         mov byte ptr gdt[TSS_int8_sel + 5],81h ;TSS 286 desocupado.
1187         mov byte ptr gdt[TSS_pri1_sel + 5],89h ;TSS 386 desocupado.
1188         mov byte ptr gdt[TSS_pri2_sel + 5],89h ;TSS 386 desocupado.
1189         mov byte ptr gdt[TSS_inic_sel + 5],81h ;TSS 286 desocupado.
1190         jmp far ptr dummy        ;Cambiar de tarea.
1191         org $-2                  ;Mover la posición de ensamblado para que
1192                                  ;apunte al segmento del salto intersegmento.
1193 salto   dw 0
1194 ;Cuando se ejecute la próxima INT 8 va a ejecutar a partir de este punto.
1195         jmp int8han              ;Volver a ejecutar la interrupción.
1196 ;Subrutina para cambiar la velocidad del timer. Entrada: El registro CX
1197 ;contiene la máxima cuenta. El tiempo entre interrupciones será de
1198 ;(CX / 1193180) segundos. Si CX = 0, reemplazar en la fórmula por 65536.
1199 timer:  mov al,00110110b         ;bit 7,6 = (00) timer 0.
1200                                  ;bit 5,4 = (11) escribir LSB y luego MSB.
1201                                  ;bit 3-1 = (011) generar onda cuadrada.
1202                                  ;bit 0 = (0) contador binario.
1203         out 43h,al               ;salida al registro de control del PIT
1204                                  ;(Programmable Interval Timer).
1205         jmp $+2
1206         mov al,cl                ;Escribir el LSB de la cuenta del timer.
1207         out 40h,al
1208         jmp $+2
1209         mov al,ch                ;Escribir el MSB de la cuenta del timer.
1210         out 40h,al
1211         ret
1212 inicio: mov dx,offset texto6   ;Puntero al texto de procesador incorrecto.
1213 test86: pushf                  ;Sirve para rechazar un 8086. En este procesador
1214         pop ax                 ;los bits 15-12 del registro de indicadores
1215         and ax,0FFFh           ;siempre están a uno.
1216         push ax
1217         popf
1218         pushf
1219         pop ax
1220         add ax,1000h
1221         jc short mal
1222 test286:pushf                  ;Sirve para rechazar un 80286. En este
1223         pop ax                 ;procesador los bits 15-12 del registro de
1224         or ax,0F000h           ;indicadores siempre están a cero en modo real.
1225         push ax
1226         popf
1227         pushf
1228         pop ax
1229         and ax,0F000h
1230         jz short mal
1231 virtual?:smsw ax        ;Sirve para ver si el 80386 está en modo virtual.
1232         test al,1       ;Aquí no puede utilizarse MOV EAX,CR0 ya que esta
1233                         ;instrucción genera una excepción 13 en modo virtual.
```

```
1234         jz short modo_real
1235         mov dx,offset texto5
1236 mal:    mov ah,9              ;Mostrar el mensaje de error.
1237         int 21h
1238         mov ax,4c01h          ;Terminar el programa con código de salida 1.
1239         int 21h
1240 modo_real:in al,21h           ;Almacenar en un lugar temporal los IRQ habili-
1241         mov picint,al         ;tados en el controlador de interrupciones (PIC).
1242         mov real_cs,cs        ;Llenar el campo correspondiente al segmento en
1243                               ;el salto intersegmento que está más abajo.
1244         mov ah,0Fh            ;Obtener el tipo de pantalla: 7 = Monocromático,
1245         int 10h               ;otro valor = color.
1246         mov bx,0B000h         ;Segmento para buffer de video monocromo.
1247         cmp al,7              ;Ver si es monocromo.
1248         jz short inicio1      ;Saltar si es así.
1249         mov port6845,3D4h     ;Port de tarjeta color.
1250         mov bx,0B800h         ;Segmento para buffer de video color.
1251         or byte ptr gdt[pant_sel + 3],80h    ;Ajustar en la GDT las bases de
1252         or byte ptr gdt[pant_inf_sel + 3],80h;los segmentos para tarjeta color.
1253 inicio1:mov es,bx
1254         xor di,di             ;Limpiar la pantalla.
1255         mov cx,25*80          ;25 líneas * 80 caracteres.
1256         mov ax,0720h          ;Carácter espacio con atributo blanco sobre negro.
1257         cld                   ;Dirección de escritura hacia arriba.
1258         rep stosw             ;Llenar los 4000 bytes con los espacios.
1259         mov cl,80             ;Generar las dos líneas horizontales.
1260         mov si,10*80*2
1261 inicio2:mov byte ptr es:[si],196
1262         mov byte ptr es:[si+4*80*2],196
1263         add si,2
1264         loop inicio2
1265         mov bx,offset texto1  ;Mostrar los tres textos en la zona central.
1266         call mostex
1267         mov bx,offset texto2
1268         call mostex
1269         mov bx,offset texto3
1270         call mostex
1271         mov cx,26*80          ;Sacar el cursor de la pantalla poniéndolo en
1272         call ponercur         ;la línea 26.
1273         xor eax,eax
1274         mov ax,cs
1275         shl eax,4             ;EAX = Dirección lineal de inicio de este
1276                               ;segmento.
1277         add dword ptr gdt[code_sel + 2],eax ;Poner los verdaderos inicios
1278         add dword ptr gdt[data_sel + 2],eax ;(bases) de los segmentos.
1279         add dword ptr gdt[TSS_inic_sel + 2],eax
1280         add dword ptr gdt[TSS_pri1_sel + 2],eax
1281         add dword ptr gdt[TSS_pri2_sel + 2],eax
1282         add dword ptr gdt[TSS_int8_sel + 2],eax
1283         add dword ptr gdt[TSS_int9_sel + 2],eax
1284         add dword ptr gdtr[2],eax
1285         add dword ptr idtr[2],eax
1286         mov al,0FFh     ;Deshabilitar todas las IRQ del PIC.
1287         out 21h,al
1288         jmp $+2
1289         mov cx,3580     ;Hacer un tic del timer cada 0,003 segundos.
1290         call timer      ;Esta será la resolución del "scheduler" (interr. 8).
1291         sidt real_idtr  ;Almacenar en memoria el IDTR del modo real.
1292         lgdt gdtr       ;Cargar el GDTR.
1293         lidt idtr       ;Cargar el IDTR.
1294         push code_sel   ;Poner en la pila el descriptor y el offset del
1295         push offset modo_protegido ;código en modo protegido.
1296         smsw ax         ;Pasar a modo protegido poniendo a uno el bit 0
1297         or al,1         ;(Protection Enable) de CR0. Las instrucciones SMSW y
1298         lmsw ax         ;LMSW operan con la mitad (16 bits) baja de CR0 (MSW).
```

```
1299         retf            ;Ir a ejecutar código en modo protegido.
1300                         ;En este caso es la instrucción siguiente.
1301 modo_protegido:
1302         mov ax,TSS_inic_sel
1303         ltr ax          ;Cargar el registro puntero de TSS.
1304 ;Cargar los registros de segmento con el selector del segmento de pila. Es
1305 ;fundamental cargar todos los registros, ya que, como antes de comenzar el
1306 ;programa tienen cualquier valor, al realizar un cambio de tarea tomará el
1307 ;valor del segmento real como un selector en modo protegido. Como esta entrada
1308 ;generalmente no está definida en la GDT, ocurrirá una excepción 13.
1309         mov ax,data_sel
1310         mov ds,ax
1311         mov ss,ax
1312         mov es,ax
1313         mov fs,ax
1314         mov gs,ax
1315         mov al,0FEh     ;Habilitar únicamente la INT 8 (reloj).
1316         out 21h,al
1317         jmp $+2
1318 espera: test estado,APRETO_ESC ;¿Se apretó la tecla ESC?
1319         jz espera       ;Saltar si no es así.
1320         mov al,0FFh     ;Deshabilitar todas las interrupciones.
1321         out 21h,al
1322         jmp $+2
1323         mov eax,cr0     ;Volver a modo real poniendo a cero el bit 0 de CR0.
1324         and al,0feh     ;Con la instrucción LMSW no se puede poner a cero
1325         mov cr0,eax     ;este bit (por compatibilidad con el 80286).
1326         jmp far ptr dummy ;Aquí es absolutamente necesario un salto DIRECTO
1327                           ;intersegmento. No funciona ni el método de retf
1328                           ;(como arriba) ni un salto indirecto.
1329         org $-4           ;Mover la posición de ensamblado para que
1330                           ;apunte al offset del salto intersegmento.
1331         dw regreso_a_modo_real ;Offset del salto.
1332 real_cs dw 0              ;Segmento del salto. Debe copiarse aquí el valor del
1333                           ;segmento de código inicial (en modo real) en tiempo
1334                           ;de ejecución (si el programa hubiese sido .EXE esto
1335                           ;no hubiera sido necesario).
1336 regreso_a_modo_real:
1337         mov ax,cs         ;Restaurar el valor del segmento
1338         mov ss,ax         ;de pila en modo real.
1339         lidt ss:real_idtr ;Restaurar el valor de IDTR en modo real.
1340         xor cx,cx         ;Restaurar la velocidad del timer.
1341         call timer
1342         mov al,ss:picint  ;Restaurar las habilitaciones de la PIC.
1343         out 21h,al
1344         jmp $+2
1345 ;Como no se incrementó el tic de reloj durante la ejecución del programa,
1346 ;la hora que indicará el DOS estará atrasada, por lo que hay que leer la
1347 ;hora del reloj de tiempo real mediante la BIOS e indicárselo al DOS
1348 ;mediante la función de poner la hora. La BIOS devuelve la hora en BCD,
1349 ;mientras que el DOS espera la hora en binario.
1350         mov ah,2          ;Utilizar la función de BIOS para leer la hora.
1351         int 1Ah           ;CH = Hora, CL = Minutos, DH = Segundos (en BCD).
1352         mov al,ch
1353         mov ah,ch
1354         shr ah,4          ;AH = Decenas de horas.
1355         and al,0Fh        ;AL = Unidades de horas.
1356         aad
1357         mov ch,al         ;CH = Hora (en binario).
1358         mov al,cl
1359         mov ah,cl
1360         shr ah,4          ;AH = Decenas de minutos.
1361         and al,0Fh        ;AL = Unidades de minutos.
1362         aad
1363         mov cl,al         ;CL = Minutos (en binario).
```

```
1364          mov al,dh
1365          mov ah,dh
1366          shr ah,4          ;AH = Decenas de segundos.
1367          and al,0Fh        ;AL = Unidades de segundos.
1368          aad
1369          mov dh,al         ;DH = Segundos (en binario).
1370          mov dl,50         ;DL = Centésimas de segundo.
1371          mov ah,2Dh        ;Poner la hora del DOS.
1372          int 21h
1373          mov ax,4c00h      ;Terminar el programa.
1374          int 21h
1375          align 4           ;Hacer que el SP sea múltiplo de 4 (por velocidad).
1376 pila_primos_division        equ $
1377 pila_primos_potenciacion    equ $+100h
1378 pila_int8       equ $+200h
1379 pila_int9       equ $+300h
1380 codigo  ends
1381          end comienzo
```

</HTML>

```
 1 ; CALCULO DE PI Y LOGARITMO NATURAL DE 2 POR DARIO A. ALPERN
 2 ; USANDO DPMI (DOS PROTECTED MODE INTERFACE)
 3 ;
 4 ; Formulas utilizadas:
 5 ; ===================
 6 ;
 7 ; Calculo de pi:
 8 ;
 9 ; pi = 4 [ 6 arc tg (1/8) + arc tg (1/32) + arc tg (1/128) +
10 ;        + arc tg (13753/63664921) ]
11 ;
12 ; Calculo de ln 2:
13 ;
14 ; ln 2 =  5 arg tgh (1/8) + arg tgh (1/16) + arg tgh (1529 / 670751)
15 ;
16 ; El calculo del arco tangente se realiza mediante la formula de Taylor:
17 ;
18 ;                  3     5     7     9
19 ;                 x     x     x     x
20 ; arc tg x = x - --- + --- - --- + --- - ...
21 ;                 3     5     7     9
22 ;
23 ; El calculo del argumento de la tangente hiperbolica se realiza mediante la
24 ; formula de Taylor:
25 ;
26 ;                   3     5     7     9
27 ;                  x     x     x     x
28 ; arg tgh x = x + --- + --- + --- + --- + ...
29 ;                  3     5     7     9
30 ;
31 ;
32 ; Estas formulas se deducen teniendo en cuenta que:
33 ;
34 ;                                                          x + y
35 ;   pi = 4 arc tg 1     y   arc tg x + arc tg y = arc tg --------- ;
36 ;                                                        1 - xy
37 ;
38 ;                 3                                        x + y
39 ; ln 2 = arg tgh ---    y   arg tgh x + arg tgh y = arg tgh --------- .
40 ;                 5                                      1 + xy
41 ;
42 ; En el caso de procesadores de 16 bits es muy costoso en tiempo dividir por
43 ; 63664921 o por 670751, por lo que se divide por sus factores (mediante dos
44 ; divisiones). Las factorizaciones son: 63664921 = 1217 x 52313 y
45 ; 670751 = 347 x 1933.
46 ;
47 ;-----------------------------------------------------------------------------
48 ;
49 ; Durante el calculo de PI o LN 2, se utilizan tres buffers que contienen
50 ; diferentes numeros en binario: el primero (que comienza en "buffer")
51 ; contiene la suma parcial de los terminos; en el segundo (que comienza en
52 ; "buffer + 16384") se calculan las potencias que intervienen en los
53 ; diferentes terminos; mientras que en el ultimo (que comienza en
54 ; "buffer + 32768") se calcula un termino que luego se sumara o restara del
55 ; que comienza en "buffer".
56 ;
57 ;-----------------------------------------------------------------------------
58 ;
59 ; Definicion de constantes y variables inicializadas del programa.
60 ;
61 .386
62 eloop    macro destino            ;Macro para hacer loop en ECX.
63          db 67h                   ;Prefijo de longitud de direccion.
64          loop destino
65          endm
```

```
 66
 67 estruc_modo_real struc        ;Estructura para pasar parametros a modo real.
 68 reg_EDI  dd ?                 ;Imagen registro EDI.
 69 reg_ESI  dd ?                 ;Imagen registro ESI.
 70 reg_EBP  dd ?                 ;Imagen registro EBP.
 71 reserved dd ?
 72 reg_EBX  dd ?                 ;Imagen registro EBX.
 73 reg_EDX  dd ?                 ;Imagen registro EDX.
 74 reg_ECX  dd ?                 ;Imagen registro ECX.
 75 reg_EAX  dd ?                 ;Imagen registro EAX.
 76 reg_Flags  dw ?                ;Imagen registro de indicadores.
 77 reg_ES   dw ?                 ;Imagen registro ES modo real.
 78 reg_DS   dw ?                 ;Imagen registro DS modo real.
 79 reg_FS   dw ?                 ;Imagen registro FS modo real.
 80 reg_GS   dw ?                 ;Imagen registro GS modo real.
 81 reg_IP   dw ?                 ;Imagen registro IP modo real.
 82 reg_CS   dw ?                 ;Imagen registro CS modo real.
 83 reg_SP   dw ?                 ;Imagen registro SP modo real.
 84 reg_SS   dw ?                 ;Imagen registro SS modo real.
 85 estruc_modo_real ends
 86
 87 HANDLE_STDOUT      equ 1      ;Handle Standard Output (DOS).
 88 HANDLE_STDERR      equ 2      ;Handle Standard Error (DOS).
 89
 90 LN2                equ 1
 91 E                  equ 2
 92 Formato            equ 4
 93 MostrarCantDigitos equ 8
 94
 95 ASCII_CR           equ 13     ;Mover el cursor al extremo izquierdo.
 96 ASCII_LF           equ 10     ;Mover el cursor hacia abajo.
 97 port6845           equ 3D4h   ;Port del controlador de video.
 98
 99 codigo  segment use16
100         assume cs:codigo,ds:codigo
101         org 100h            ;Offset de arranque del programa en formato .COM
102 comienzo:jmp Inicio         ;Salto al inicio del programa.
103
104 divisor         dd 1        ;Divisor de cada termino. Va tomando los valores
105                             ;1, 3, 5,... como se aprecia en las formulas.
106
107 digitos         dd 0        ;Cantidad de digitos que posee la aproximacion que
108                             ;se desea mostrar.
109
110 PMode_entry     label dword ;Direccion punto de entrada a modo protegido.
111 PMode_entry_offs dw 0       ;Offset.
112 PMode_entry_segm dw 0       ;Segmento.
113
114 CantDWords      dd 0        ;Cantidad de DWords (32 bits) correspondientes a
115                             ;la cantidad de digitos pedidos.
116
117 UltDWord        dd    0
118 PrimDWord       dd    0     ;Palabra mas significativa distinta de cero.
119                             ;Al no tener que procesar las palabras que valen
120                             ;cero se duplica la velocidad.
121
122 Numerador       dd 13753*13753 ;Numerador (32 bits) del argumento del ultimo
123                             ;arc tg/arg tgh. Esta inicializado para el calculo
124                             ;de PI. En el caso del calculo de ln 2 este valor
125                             ;se sobreescribe con 1529*1529 al principio del
126                             ;programa. Ver formulas arriba.
127
128 Denominador     dd 63664921 ;Denominador (32 bits) del argumento del ultimo
129                             ;arc tg/arg tgh. Se inicializa al principio del
130                             ;programa con 63664921 (para PI) o 670751
```

```
131                             ;(para ln 2). Ver formulas arriba.
132
133 ; Multiplicadores para averiguar la cantidad de DWords W necesarios para
134 ; calcular n digitos en base b: W = 2^32*log(b)/log(2^32)
135 Multiplicadores label dword
136                 dd 134217728    ; Base 2
137                 dd 212730066    ; Base 3
138                 dd 268435456    ; Base 4
139                 dd 311643913    ; Base 5
140                 dd 346947794    ; Base 6
141                 dd 376796799    ; Base 7
142                 dd 402653184    ; Base 8
143                 dd 425460132    ; Base 9
144                 dd 445861641    ; Base 10
145                 dd 464317052    ; Base 11
146                 dd 481165522    ; Base 12
147                 dd 496664612    ; Base 13
148                 dd 511014527    ; Base 14
149                 dd 524373979    ; Base 15
150                 dd 536870912    ; Base 16
151                 dd 548609976    ; Base 17
152                 dd 559677860    ; Base 18
153                 dd 570147180    ; Base 19
154                 dd 580079369    ; Base 20
155                 dd 589526865    ; Base 21
156                 dd 598534780    ; Base 22
157                 dd 607142208    ; Base 23
158                 dd 615383250    ; Base 24
159                 dd 623287827    ; Base 25
160                 dd 630882340    ; Base 26
161                 dd 638190197    ; Base 27
162                 dd 645232255    ; Base 28
163                 dd 652027172    ; Base 29
164                 dd 658591707    ; Base 30
165                 dd 664940973    ; Base 31
166                 dd 671088640    ; Base 32
167                 dd 677047118    ; Base 33
168                 dd 682827704    ; Base 34
169                 dd 688440713    ; Base 35
170                 dd 693895588    ; Base 36
171
172 MaxPotencia       dd 0      ;Maxima potencia para la base pedida que entre
173                             ;en un doubleword.
174                             ;El valor es: Base^(int(log (2^32)/log Base))
175 LogMaxPotencia    dd 0      ;Exponente para la potencia arriba indicada.
176                             ;Cantidad de digitos por DWord.
177                             ;El valor es: int(log (2^32)/log Base)) *
178                             ;* 2^32*log Base/log(2^32)
179 SumaLogPotencia   dd 0      ;Suma de LogMaxPotencia para determinar la
180                             ;cantidad de digitos durante la conversion.
181 Exponente         dd 0      ;Indica la cantidad de digitos que se convieten
182                             ;por vez.
183 HandleMemoriaBajo dw 0      ;Handle de memoria que entrega
184 HandleMemoriaAlto dw 0      ;el servidor de DPMI.
185
186 PunteroSubrPotenciasDe2 dw PotBinParaPi
187                             ;Puntero a la subrutina para calcular las potencias
188                             ;binarias. Esta inicializada para el calculo de PI.
189                             ;En el caso del calculo de ln 2 este valor se
190                             ;sobreescribe con el correspondiente para ln 2.
191
192 handle          dw 1        ;Handle correspondiente al dispositivo estandar
193                             ;de salida (la pantalla por defecto). Se
194                             ;sobreescribe con el handle del archivo, si se
195                             ;especifica.
```

```
196
197 Indicadores     db 0        ;Sirve para saber si la linea de comando contiene
198                               ;los "switches" /pi o /ln2:
199                               ;Bit 0 = 1: Calculo de ln 2.
200                               ;Bit 0 = 0: Calculo de pi.
201                               ;Bit 1 = 1: Salida formateada.
202                               ;Bit 1 = 0: Salida sin formatear.
203
204 bufferdos       db "3,",60 dup (0) ;Buffer para la conversion de binario a
205                                      ;decimal.
206
207 BufferNumeroFormateado db "3,",75 dup (" "),ASCII_CR,ASCII_LF
208                               ;Buffer de una linea para obtener la salida
209                               ;formateada.
210
211 PosicionDigitoEnBuffer dw BufferNumeroFormateado + 2
212                               ;Puntero dentro del buffer de una linea.
213
214 InformacionDeFormateado dw 0      ;MSB = Numero de grupo (0-9).
215                                    ;LSB = Digito dentro del grupo (1-5).
216
217 DigitosMostrados        dd 0
218
219 Selector dw 0
220
221 Iniciales     db "dhms"
222 Dias          db 0
223 Horas         db 0
224 Minutos       db 0
225 Segundos      db 0
226 FraccSegundo  dd 0
227
228 OldInt8Handler label dword
229 OldInt8Offset  dw 0
230 OldInt8Segment dw 0
231
232 MostrarEnInt8  db 1
233 PunteroBase    dw 0
234 Base           db 10
235
236
237 CursorVideo    label dword
238 CursorVideoOffset  dw 0
239 CursorVideoSegment dw 0B800h
240
241 CopyRight db "Calculo de PI, ln 2 y e. Hecho por Dario Alejandro Alpern."
242         db ASCII_CR,ASCII_LF
243 LongitudCopyRight equ $ - CopyRight
244 Texto   db ASCII_CR,ASCII_LF
245         db "Uso del programa: PIDPMI cantidad_de_decimales [/F] [/LN2] [/E] "
246         db "[/B:base]",ASCII_CR,ASCII_LF," [arch_destino]",ASCII_CR,ASCII_LF
247         db "   /F: Muestra la salida formateada para editores de texto"
248         db ASCII_CR,ASCII_LF
249         db "   /LN2: Muestra el valor del logaritmo natural de 2 en vez de pi"
250         db ASCII_CR,ASCII_LF
251         db "   /E: Muestra el valor del numero e en vez de pi"
252         db ASCII_CR,ASCII_LF
253         db "   /B: Muestra el resultado en base 2 a 36"
254         db ASCII_CR,ASCII_LF
255         db "   arch_destino: Archivo en que se almacenara el numero."
256         db ASCII_CR,ASCII_LF
257         db "                  Si no se especifica, es la pantalla (STDOUT)."
258         db ASCII_CR,ASCII_LF
259         db ASCII_CR,ASCII_LF
260 LongitudTexto equ $ - Texto
```

```
261 NoHayDPMI db "No hay servidor de DPMI.",ASCII_CR,ASCII_LF
262 LongitudNoHayDPMI equ $ - NoHayDPMI
263 Servidor16Bits db "El servidor de DPMI no soporta aplicaciones de 32 bits"
264                db ASCII_CR,ASCII_LF
265 LongitudServidor16Bits equ $ - Servidor16Bits
266 NoHayMemoriaPServidor db "No hay memoria local para el servidor de DPMI."
267                db ASCII_CR,ASCII_LF
268 LongitudNoHayMemoriaPServidor equ $ - NoHayMemoriaPServidor
269 NoSePuedePasarAProtegido db "No se puede pasar a modo protegido."
270                db ASCII_CR,ASCII_LF
271 LongitudNoSePuedePasarAProtegido equ $ - NoSePuedePasarAProtegido
272 NoSeReservoSelector db "No se puede reservar un selector."
273                db ASCII_CR,ASCII_LF
274 LongitudNoSeReservoSelector equ $ - NoSeReservoSelector
275 MemoriaLlena db ASCII_CR,ASCII_LF
276         db "No hay suficiente memoria para procesar tantos "
277         db "decimales.",ASCII_CR,ASCII_LF
278 LongitudMemoriaLlena equ $ - MemoriaLlena
279 TextoSegundaLinea db "Digitos hallados: "
280 LongitudTextoSegundaLinea equ $ - TextoSegundaLinea
281 TiempoTotal db ASCII_CR,ASCII_LF,"Tiempo total: "
282 LongitudTiempoTotal equ $ - TiempoTotal
283 ModoReal estruc_modo_real <0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0>
284 ;=============================================================================
285 ; Mostrar el "Copyright"
286 ;=============================================================================
287 Inicio: mov dx,offset CopyRight ;Mostrar en pantalla el "copyright".
288         mov cx,LongitudCopyRight
289         mov bx,HANDLE_STDERR        ;Handle STDERR.
290         mov ah,40h
291         int 21h
292 ;=============================================================;
293 ; Determinacion de existencia de DPMI y pasaje a modo protegido  ;
294 ;=============================================================;
295 VerificarDPMI:
296         mov ModoReal.reg_DS,ds
297         pushf
298         pop ax
299         and ax,0FFFh
300         mov ModoReal.reg_Flags,ax
301         mov ax,1687h            ;Ver si hay un servidor de DPMI instalado en
302         int 2Fh                 ;memoria.
303         test ax,ax
304         jz short InicializarDPMI ;Saltar si hay servidor de DPMI.
305         mov dx,offset NoHayDPMI
306         mov cx,LongitudNoHayDPMI ;Mostrar que no hay servidor de DPMI.
307 error_en_real:
308         mov bx,HANDLE_STDERR    ;Handle STDERR.
309         mov ah,40h
310         int 21h
311         mov ax,4c01h            ;Fin del programa con codigo de error 1.
312         int 21h
313 InicializarDPMI:
314         mov PMode_entry_segm,es ;Preservar el punto de entrada del servidor
315         mov PMode_entry_offs,di ;de DPMI para entrar a modo protegido.
316         test bl,1               ;El servidor soporta aplicaciones de 32 bits?
317         mov dx,offset Servidor16Bits
318         mov cx,LongitudServidor16Bits
319         jz short error_en_real  ;Saltar si no es asi.
320         test si,si              ;El servidor necesita memoria local?
321         jz short ir_a_modo_protegido ;Saltar si no es asi.
322         push ds
323         pop es                  ;ES = Segmento del programa .COM
324         push si                 ;Preservar la cantidad de bytes que necesita
325                                 ;el servidor de DPMI.
```

```
326         mov ah,4Ah              ;Achicar al minimo el bloque de memoria que
327         mov bx,offset fin_codigo[15] ;incluye el programa .COM
328         shr bx,4
329         int 21h
330         pop bx
331         mov ah,48h
332         int 21h                 ;Reservar memoria para el servidor de DPMI.
333         mov cx,LongitudNoHayMemoriaPServidor
334         mov dx,offset NoHayMemoriaPServidor
335         jc short error_en_real  ;Saltar si no se pudo reservar memoria.
336         mov es,ax               ;Apunta al area de datos del servidor de DPMI.
337 ir_a_modo_protegido:
338         mov ax,1                ;Indicar que la aplicacion es de 32 bits.
339         call PMode_entry        ;Pasar a modo protegido.
340         mov cx,LongitudNoSePuedePasarAProtegido
341         mov dx,offset NoSePuedePasarAProtegido
342         jc short error_en_real
343         push ds
344         pop es                  ;Volver a poner ES = DS.
345         xor ax,ax
346         mov cx,1                ;Reservar un descriptor.
347         int 31h
348         mov cx,LongitudNoSeReservoSelector
349         mov dx,offset NoSeReservoSelector
350         jc short error_modo_prot
351         mov Selector,ax         ;Preservar el selector correspondiente.
352         mov bx,ax
353         mov ax,0009h
354         mov cx,40F2h
355         int 31h                 ;Poner el byte de derechos de acceso.
356 HallarCantDecimales:
357         mov bx,80h              ;Apuntar al byte anterior del principio de
358                                 ;la linea de comandos.
359         mov esi,10              ;ESI = Multiplicador.
360         xor eax,eax             ;Inicializar numero a formar a cero.
361         xor ecx,ecx             ;Inicializar digito a cero.
362 BusquedaDeDigito:
363         inc bx                  ;Apuntar al siguiente byte.
364         mov al,[bx]             ;Obtener el caracter de la linea de comandos.
365         cmp al,ASCII_CR         ;Se acabo la linea?
366         je short error          ;Saltar a error si es asi.
367         cmp al," "              ;Es un espacio o caracter no imprimible?
368         jbe BusquedaDeDigito    ;Saltear dicho caracter.
369         sub al,"0"              ;Ver si el caracter esta entre '0' y '9'.
370         cmp al,9
371         jbe short NumeroLineaDeComando ;Saltar si esta dentro del rango.
372                                 ;Caso contrario mostrar el error ya
373                                 ;que la linea debe comenzar con un numero.
374 error:
375         mov dx,offset Texto     ;Mostrar texto indicando el formato de la
376         mov cx,LongitudTexto    ;linea de comando.
377 error_modo_prot:
378         mov bx,HANDLE_STDERR    ;Handle STDERR.
379         mov ah,40h
380         call dos
381         mov ax,4c01h            ;Fin del programa con codigo de error 1.
382         int 21h
383 NuevoDigito:
384         mul esi                 ;Multiplicar numero a formar por diez.
385         jc NoHaySuficienteMemoria ;Saltar si el resultado es muy grande.
386         add eax,ecx             ;Sumar el digito hallado en la linea de comando
387         jc NoHaySuficienteMemoria ;Saltar si el resultado es muy grande.
388 NumeroLineaDeComando:
389         inc bx                  ;Apuntar al siguiente byte de la linea.
390         mov cl,[bx]             ;Obtener el caracter.
```

```
391         sub cl,"0"              ;Ver si esta entre '0' y '9'.
392         cmp cl,9
393         jbe NuevoDigito         ;Sumar digito y repetir si esta en el rango.
394         mov digitos,eax         ;Almacenar cantidad de digitos en variable.
395 ;
396 ; Ver si en la linea de comandos aparecen /F o /LN2. En el caso que existan,
397 ; se ponen a uno ciertos bits de la variable Indicadores. Para /F se pone a
398 ; 1 el bit Formato, mientras que para /LN2 se pone a 1 el bit LN2.
399 ;
400         call switch             ;Encender indicadores segun los switches
401                                 ;encontrados en la linea de comandos.
402         xor ecx,ecx
403         mov cl,Base
404         mov di,cx               ;Obtener la base.
405         shl di,2
406         add di,offset Multiplicadores[-8] ;Apuntar al logaritmo.
407         mov PunteroBase,di
408         mov si,-1               ;Inicializar exponente.
409         mov eax,1               ;Inicializar potencia.
410 CicloHallarPotencia:
411         inc si                  ;Actualizar exponente.
412         mul ecx
413         jnc CicloHallarPotencia
414         mov word ptr Exponente,si
415         div ecx
416         mov MaxPotencia,eax     ;Almacenar la maxima potencia.
417         mov eax,Exponente
418         mul dword ptr [di]
419         mov LogMaxPotencia,eax
420 HallarCantDWords:
421         mov eax,Digitos         ;EAX = Cantidad de digitos a calcular.
422         mov bx,PunteroBase      ;Obtener el puntero al multiplicador que
423                                 ;corresponde a la base.
424         mul dword ptr [bx]      ;EDX = Cantidad de DWords necesarias.
425         add edx,3               ;Reservar 3 DWords mas por el redondeo.
426         mov CantDWords,edx      ;Almacenar la cantidad de DWords necesarias.
427         shl edx,2               ;Convertir a cantidad de bytes.
428         lea edx,[edx * 2 + edx] ;Reservar espacio para 3 buffers.
429         or dx,0FFFh             ;Calcular el limite para pagina completa.
430         shld ecx,edx,16         ;CX = Parte alta del limite.
431         push cx                 ;Preservar la parte alta del limite.
432         push dx                 ;Preservar la parte baja del limite.
433         mov bx,Selector
434         mov ax,0008h
435         int 31h                 ;Poner el limite del segmento.
436         pop cx                  ;Restaurar la parte baja del limite.
437         pop bx                  ;Restaurar la parte alta del limite.
438         add cx,1                ;Se debe reservar un byte mas que el limite.
439         adc bx,0
440         mov ax,0501h            ;Reservar BX:CX bytes de memoria lineal.
441         int 31h                 ;BX:CX = Direccion lineal del bloque.
442                                 ;SI:DI = Handle de memoria.
443         jnc short LosDigitosEntran ;Saltar si no hubo errores.
444 NoHaySuficienteMemoria:
445         mov dx,offset MemoriaLlena ;Indicar la condicion de error.
446         mov cx,LongitudMemoriaLlena
447         mov ah,40h
448         call dos
449         mov ax,4C02h            ;Saltar si no es asi.
450         int 21h
451 LosDigitosEntran:
452         mov HandleMemoriaBajo,di  ;Preservar el handle de memoria ubicado
453         mov HandleMemoriaAlto,si  ;en SI:DI
454         mov dx,cx               ;DX = Parte baja de la base del segmento.
455         mov cx,bx               ;CX = Parte alta de la base del segmento.
```

```
456         mov ax,0007h
457         mov bx,Selector
458         int 31h                 ;Llenar la base del segmento.
459 InicializarPrimerTermino:
460         test Indicadores,E
461         jz NoEsCalculoE
462 ;-------------------------------------------------------------------------
463 ; Calculo del numero e:
464 ;
465         mov bufferdos,"2"       ;Parte entera de e.
466         call PonerVectorInterrupcion
467         mov ebx,CantDWords
468         mov ecx,ebx             ;Obtener cantidad de DWords del numero.
469         shl ecx,1               ;Obtener cantidad de DWords a poner a cero.
470         xor edi,edi             ;Apuntar al principio del numero ubicado en
471                                 ;memoria pedida al servidor de DPMI.
472         xor eax,eax             ;Poner ambos numeros a cero.
473         rep stos dword ptr es:[edi]
474         inc divisor                     ;Divisor = 2.
475         mov byte ptr es:[ebx*4+3],80h  ;Poner a 0,5 el termino.
476         mov byte ptr es:[3],80h        ;Poner a 2,5 la suma.
477 CicloPrincE:
478         mov ecx,CantDWords              ;ECX = Cantidad de DWords del numero.
479         inc divisor                     ;Siguiente divisor.
480         mov edi,ecx                     ;EDI = Cantidad de DWords del numero.
481         shl edi,2                       ;EDI = Puntero al termino.
482         mov eax,PrimDWord               ;EAX = Numero de palabra mas
483                                         ;significativa distinta de ceros.
484         sub ecx,eax                     ;ECX = Cantidad de palabras a procesar.
485         jbe BinarioADecimal             ;Saltar si es <= 0.
486         lea edi,[edi+eax*4]             ;EDI = Puntero a la palabra mas
487                                         ;significativa distinta de ceros.
488         mov esi,divisor                 ;ESI = Divisor en la formula de e.
489         xor edx,edx                     ;EDX = Resto de la division.
490         mov ebx,edi                     ;EBX = Puntero al MSD.
491         cld                             ;Procesar en direcciones ascendentes
492                                         ;(para realizar una division se deben
493                                         ;tomar primero los DWords mas signif.)
494         push ecx                        ;Preservar la cantidad de palabras a
495                                         ;procesar.
496         shr ecx,1
497         jnc short DivVariableE
498         mov eax,es:[edi]                ;Obtener el DWord del dividendo.
499         div esi                         ;Realizar la primera division.
500         stos dword ptr es:[edi]         ;Almacenar el cociente.
501 DivVariableE:
502         jecxz FinDivVariableE           ;Saltar si se termino la division.
503 CicloDivVariableE:
504         mov eax,es:[edi]                ;Obtener el DWord del dividendo.
505         div esi                         ;Realizar la primera division.
506         stos dword ptr es:[edi]         ;Almacenar el cociente.
507         mov eax,es:[edi]                ;Obtener el DWord del dividendo.
508         div esi                         ;Realizar la primera division.
509         stos dword ptr es:[edi]         ;Almacenar el cociente.
510         eloop CicloDivVariableE         ;Cerrar el ciclo.
511 FinDivVariableE:
512         cmp dword ptr es:[ebx],1        ;El MSD es cero?
513         adc PrimDWord,ecx               ;El MSD es el siguiente si lo era.
514         std                             ;Procesar en direcciones descendentes
515                                         ;(para realizar una suma se deben
516                                         ;tomar primero los DWords menos
517                                         ;significativos).
518         lea esi,[edi-4]                 ;Apuntar al LSD del termino.
519         mov eax,CantDWords              ;EAX = Cantidad de DWords del numero.
520         shl eax,2                       ;EAX = Cantidad de bytes del numero.
```

```
521         mov edi,esi
522         sub edi,eax                     ;EDI = Puntero al LSD de la suma.
523         pop ecx                         ;Restaurar la cantidad de palabras a
524                                         ;procesar.
525         shr ecx,1
526         jnc short SumaE                 ;Saltar si la cant. de DWords era par.
527         lods dword ptr es:[esi]         ;Obtener DWord del termino.
528         adc eax,dword ptr es:[edi]      ;Sumarselo a la suma.
529         stos dword ptr es:[edi]         ;Guardarlo en la suma.
530 SumaE:
531         jecxz FinSumaE                  ;Saltar si se termino el sumando.
532 CicloSumaE:
533         lods dword ptr es:[esi]         ;Obtener DWord del termino.
534         adc eax,dword ptr es:[edi]      ;Sumarselo a la suma.
535         stos dword ptr es:[edi]         ;Guardarlo en la suma.
536         lods dword ptr es:[esi]         ;Obtener DWord del termino.
537         adc eax,dword ptr es:[edi]      ;Sumarselo a la suma.
538         stos dword ptr es:[edi]         ;Guardarlo en la suma.
539         eloop CicloSumaE                ;Cerrar el ciclo.
540 FinSumaE:
541         jnc CicloPrincE                 ;Saltar si la suma se termino.
542 CicloSumaCarryE:
543         inc dword ptr es:[edi]          ;Incrementar el siguiente DWord mas
544                                         ;significativo.
545         lea edi,[edi-4]                 ;Apuntar al siguiente DWord mas
546                                         ;significativo.
547         jnz CicloPrincE                 ;Saltar si se acabo la suma.
548         jmp CicloSumaCarryE             ;Cerrar el ciclo.
549 ;-----------------------------------------------------------------------------
550 NoEsCalculoE:
551         mov eax,13753*4           ;Primer numerador para PI.
552         mov Numerador,13753*13753 ;Numerador para PI.
553         mov Denominador,63664921  ;Denominador para PI.
554         test Indicadores,LN2      ;Ver si se esta calculando PI o LN 2.
555         jz short FinInicializacionCtes  ;Saltar si se esta calculando PI.
556         mov PunteroSubrPotenciasDe2,offset PotBinParaLN2
557                                   ;Subrutina a llamar cuando se calculen las
558                                   ;potencias de 2.
559         mov eax,1529
560         mov Numerador,1529*1529   ;Numerador para LN 2.
561         mov Denominador,670751    ;Denominador para LN 2.
562         mov bufferdos,"0"         ;Indicar que el numero comienza con cero.
563 FinInicializacionCtes:
564         push eax                  ;Preservar el primer numerador.
565         call PonerVectorInterrupcion
566         pop edx                 ;Obtener el primer numerador.
567         mov ecx,CantDWords      ;Obtener cantidad de DWords del numero.
568         lea esi,[ecx*4]         ;Convertir a bytes.
569         xor edi,edi             ;Apuntar al principio del numero ubicado en
570                                 ;memoria pedida al servidor de DPMI.
571         mov es,Selector         ;El segmento de datos apunta a dicha memoria.
572         mov ebx,Denominador     ;Denominador del primer termino.
573         cld                     ;Se procesa en direcciones crecientes.
574 CicloCalcPrimerTermino:
575         xor eax,eax             ;La parte decimal del denominador es cero
576                                 ;en el primer termino.
577         div ebx                 ;Generar DWord del cociente.
578         mov es:[edi+esi],eax    ;Almacenarlo en el segundo numero.
579         stos dword ptr es:[edi] ;Almacenarlo en el primer numero.
580         eloop CicloCalcPrimerTermino ;Cerrar el ciclo.
581         mov al,256*4/32+256*4/128    ;Agregar potencias de 2 para Pi.
582         test Indicadores,LN2         ;Se esta calculando LN 2 ?
583         jz short PonerPrimeraPot2    ;Saltar si no es asi.
584         mov al,256*5/8+256*1/16      ;Agregar potencias de 2 para LN 2.
585 PonerPrimeraPot2:
```

```
586         mov es:[3],al              ;Almacenarlo en el primer numero.
587         mov es:[esi+3],al          ;Almacenarlo en el segundo numero.
588 CicloPrincipal:
589         call NuevoTermino          ;Hallar el nuevo termino.
590         test Indicadores,LN2       ;Se desea calcular LN 2?
591         jnz short CicloSumaDelTermino ;Los arg tgh solo contienen sumas.
592 CicloRestaDelTermino:
593         mov eax,es:[edi]           ;Obtener DWord de la suma.
594         sbb eax,es:[esi]           ;Restarle el Dword del sustraendo.
595         stos dword ptr es:[edi]    ;Guardar el DWord de la resta.
596         lea esi,[esi-4]            ;Apuntar al siguiente DWord.
597         eloop CicloRestaDelTermino ;Cerrar el ciclo.
598         jnc short TerminoParaSuma  ;Saltar si se termino la resta.
599 CicloSeguirRestando:
600         sbb dword ptr es:[edi],ecx ;Actualizar el DWord.
601         lea edi,[edi-4]            ;Actualizar el puntero.
602         jc CicloSeguirRestando     ;Saltar si no se acabo la resta,
603 TerminoParaSuma:
604         call NuevoTermino          ;Hallar el nuevo termino.
605 CicloSumaDelTermino:
606         lods dword ptr es:[esi]    ;Obtener el DWord del termino.
607         adc eax,es:[edi]           ;Sumarle el DWord de la suma.
608         stos dword ptr es:[edi]    ;Guardar el DWord de la suma.
609         eloop CicloSumaDelTermino  ;Cerrar el ciclo.
610         jnc CicloPrincipal         ;Saltar si la suma no se completo.
611 CicloSeguirSumando:
612         adc dword ptr es:[edi],ecx ;Actualizar el DWord.
613         lea edi,[edi-4]            ;Actualizar el puntero.
614         jc CicloSeguirSumando      ;Saltar si no se acabo la suma.
615         jmp CicloPrincipal         ;Cerrar el ciclo principal.
616 ;---------------------------------------------------------------------------
617 ; Subrutina PotBinParaLN2: Pone o saca las potencias binarias de 2 para el
618 ; calculo del logaritmo natural de 2.
619 ; Mediante un "xor", pone o saca las potencias de dos que figuran en la
620 ; formula que figura al principio del programa.
621 ;
622 PotBinParaLN2:
623         mov edi,divisor
624         lea edi,[edi*2+edi-3]
625         mov cx,di
626         mov eax,80000000h
627         ror eax,cl
628         shr edi,3
629         and di,0FFFCh
630         mov ecx,CantDWords
631         xor es:[edi+ecx*4],eax
632         mov eax,3
633         mul divisor
634         call CambiarBit
635         mov eax,4
636         mul divisor
637         jmp short CambiarBit
638 ;---------------------------------------------------------------------------
639 ; Subrutina PotBinParaPI: Pone o saca las potencias binarias de 2 para el
640 ; calculo de Pi.
641 ; Mediante un "xor", pone o saca las potencias de dos que figuran en la
642 ; formula que figura al principio del programa.
643 ;
644 PotBinParaPi:
645 ;
646 ; Hay que calcular 24 / 8 ^ (div), donde "div" es el divisor (3, 5, 7,..)
647 ; del termino. Para ello hay que obtener la posicion de los bits correspon-
648 ; dientes a "xorear" de la siguiente manera:
649 ;
650 ;          24 / 8 ^ (div) = 3 / 8 ^ (div - 1)
```

```
651 ;                         = 3 / 2 ^ (3 * div - 3)
652 ;                        = 2 ^ -(3 * div - 4) + 2 ^ -(3 * div - 3)
653 ;
654 ; Por lo tanto los bits a "xorear" son los correspondientes a las potencias
655 ; negativas de dos: 3 * div - 4 y 3 * div - 3.
656 ;
657 ; Ahora bien, como solo trabajamos con la parte decimal de los numeros, los
658 ; bits se numeran como bit 0 -> 2^(-1), bit 1 -> 2^(-2), etc., las posiciones
659 ; como numero de bit seran 3 * div - 5 y 3 * div - 4.
660 ;
661         mov edi,divisor             ;EDI <- div
662         lea edi,[edi*2+edi]         ;EDI <- 3 * div
663         sub edi,5                   ;EDI <- 3 * div - 5
664         jb short OtrasPotencias     ;Si hay carry es porque el divisor
665                                     ;es 1. En este caso 24 / 8^div es un
666                                     ;numero entero, por lo que no hay que
667                                     ;hacer nada.
668         mov cx,di                   ;Bits 4-0 de CL: Posicion de bit.
669         mov eax,0C0000000h          ;Inicializar para rotacion.
670         ror eax,cl                  ;Realizar rotacion.
671         shr edi,3                   ;Convertir numero de bit a numero de byte.
672         and di,0FFFCh               ;Alinear a DWord.
673         mov ecx,CantDWords          ;ECX = Cantidad de DWords del numero.
674         xor es:[edi+ecx*4],eax      ;Realizar "xor" para poner o sacar potencia
675 OtrasPotencias:
676         mov eax,5                   ;Poner o sacar potencia de (1/32).
677         call Potencia
678         mov eax,7                   ;Poner o sacar potencia de (1/128).
679 Potencia:
680         mul divisor
681         sub eax,2
682 CambiarBit:
683         mov cl,al
684         inc edx
685         ror edx,cl
686         shr eax,5
687         mov ecx,CantDWords
688         cmp eax,ecx
689         lea eax,[eax*4]
690         jae short FinSubrutina
691         xor es:[eax+ecx*4],edx
692 FinSubrutina:
693         ret
694 ;---------------------------------------------------------------------------
695 NuevoTermino:
696         call PunteroSubrPotenciasDe2   ;Retirar potencias de 2 del
697                                        ;termino anterior.
698         mov edi,ecx                    ;EDI = Cantidad de DWords del numero.
699         shl edi,2                      ;EDI = Puntero al termino.
700         mov eax,PrimDWord              ;EAX = Numero de palabra mas
701                                        ;significativa distinta de ceros.
702         sub ecx,eax                    ;ECX = Cantidad de palabras a procesar.
703         jbe short CantidadPalabrasDelTerminoOk    ;Saltar si es <= 0.
704         lea edi,[edi+eax*4]            ;EDI = Puntero a la palabra mas
705                                        ;significativa distinta de ceros.
706         mov esi,Denominador            ;ESI = Denominador del arc tg/arg tgh.
707         xor edx,edx                    ;EDX = Resto de la primera division
708                                        ;por el denominador.
709         xor ebx,ebx                    ;EBX = Resto de la segunda division
710                                        ;por el denominador.
711         cld                            ;Procesar en direcciones ascendentes
712                                        ;(para realizar una division se deben
713                                        ;tomar primero los Dwords mas signif.)
714         push ecx                       ;Preservar la cantidad de palabras a
715                                        ;procesar.
```

```
716 CicloDivPorDenominadorALa2:
717         mov eax,es:[edi]              ;Obtener el DWord del dividendo.
718         div esi                       ;Realizar la primera division.
719         xchg ebx,edx                  ;EDX = Resto de la segunda division
720                                       ;anterior.
721                                       ;EBX = Resto de la primera division
722                                       ;actual.
723         div esi                       ;Realizar la segunda division.
724         xchg ebx,edx                  ;EDX = Resto de la primera division
725                                       ;actual.
726                                       ;EBX = Resto de la segunda division
727                                       ;actual.
728         stos dword ptr es:[edi]       ;Almacenar el cociente.
729         eloop CicloDivPorDenominadorALa2  ;Cerrar el ciclo.
730         pop ecx                       ;Restaurar la cantidad de palabras a
731                                       ;procesar.
732         std                           ;Procesar en direcciones descendentes
733                                       ;(para realizar una multiplicacion se
734                                       ;deben tomar primero los Dwords menos
735                                       ;significativos).
736         sub edi,4                     ;Apuntar al DWord menos significativo.
737         mov esi,Numerador             ;ESI = Multiplicador.
738         mov ebx,esi                   ;EBX = Multiplicador / 2 (para redondeo)
739         shr ebx,1
740 CicloMultPorNumerador:
741         mov eax,es:[edi]              ;Obtener DWord del factor.
742         mul esi                       ;Realizar la multiplicacion.
743         add eax,ebx                   ;Sumar acarreo de multiplic. anterior.
744         adc edx,0                     ;Ajustar el DWord mas significativo.
745         mov ebx,edx                   ;Guardar acarreo de la multiplicacion.
746         stos dword ptr es:[edi]       ;Almacenar DWord del producto.
747         eloop CicloMultPorNumerador   ;Cerrar ciclo.
748         and eax,eax                   ;El DWord mas significativo vale cero?
749         jnz short CantidadPalabrasDelTerminoOk ;Saltar si no es asi.
750         inc PrimDWord                 ;El primer DWord es el siguiente.
751 CantidadPalabrasDelTerminoOk:
752         add divisor,2                 ;El divisor 3,5,7,... de la formula
753                                       ;debe ser incrementado.
754         call PunteroSubrPotenciasDe2  ;Poner las potencias binarias en
755                                       ;el termino.
756         mov esi,edi                   ;Posicion dentro del termino de la DWord
757                                       ;mas significativa.
758         mov edi,ecx                   ;EDI = Cantidad de DWords del numero.
759         shl edi,3                     ;Apuntar al principio del cociente.
760         add edi,esi                   ;Apuntar al DWord mas significativo
761                                       ;del cociente.
762         shr esi,2                     ;Convertir a cantidad de DWords.
763         sub ecx,esi
764         jbe short BinarioADecimalNoRET
765         xor edx,edx                   ;Inicializar resto de la division.
766         mov ebx,divisor               ;Obtener divisor.
767         cld                           ;Procesar en direcciones ascendentes
768                                       ;(para realizar una division se deben
769                                       ;tomar primero los DWords mas signif.)
770         add esi,CantDWords            ;
771         shl esi,2                     ;Convertir a cantidad de bytes.
772         push ecx
773 CicloDivisionPorValorVariable:
774         lods dword ptr es:[esi]
775         div ebx
776         stos dword ptr es:[edi]
777         eloop CicloDivisionPorValorVariable
778         pop ecx
779         mov edi,CantDWords
780         shl edi,2                     ;Convertir a bytes.
```

```
781         lea esi,[edi*2+edi-4]          ;Apuntar al DWord menos significativo
782                                        ;del termino.
783         add edi,-4                     ;Apuntar al DWord menos significativo
784                                        ;de ls suma.
785         std
786         ret
787 ;-------------------------------------------------------------------------
788 BinarioADecimalNoRET:
789         pop ax                         ;No volver a rutina llamadora.
790 BinarioADecimal:
791         mov al,bufferdos               ;Obtener la parte entera (en ASCII).
792         mov BufferNumeroFormateado,al  ;Almacenar el valor en caso de /f.
793         mov ax,4400h                   ;Llamada IOCTL.
794         mov bx,HANDLE_STDOUT           ;Handle STDOUT.
795         call dos
796         test dl,dl                     ;Bit 7 de DL = 1: Dispositivo.
797                                        ;            = 0: Archivo.
798         jns short TextoFormateado?     ;Saltar si se redirecciono STDOUT.
799         mov MostrarEnInt8,0            ;No mostrar texto en INT 8.
800         mov dx,offset TiempoTotal
801         mov cx,2
802         mov bx,cx                      ;BX = Handle STDERR.
803         mov ah,40h
804         call dos
805 TextoFormateado?:
806         test Indicadores,Formato       ;El texto debe estar formateado?
807         jnz short ComenzarConversion   ;Saltar si es asi.
808         mov dx,offset bufferdos
809         mov cx,2                       ;Mostrar parte entera y coma.
810         mov bx,handle
811         mov ah,40h
812         call dos
813 ;Convertir de binario a decimal.
814 ComenzarConversion:
815         std
816         mov eax,CantDWords
817         lea eax,[eax*4 - 4]
818         mov UltDWord,eax
819 MultiplicarPorMaxPotencia:
820         mov ecx,CantDWords
821         mov edi,UltDWord
822         mov esi,MaxPotencia             ;Cargar maxima potencia en un DWord.
823         xor ebp,ebp                     ;Carry primera multiplicacion.
824         xor ebx,ebx                     ;Carry segunda multiplicacion.
825 CicloMultPorMaxPotencia:
826         mov eax,es:[edi]                ;Obtener DWord del numero.
827         mul esi                         ;Realizar la primera multiplicacion.
828         add eax,ebp
829         adc edx,0
830         mov ebp,edx
831         mul esi                         ;Realizar la segunda multiplicacion.
832         add eax,ebx
833         adc edx,0
834         mov ebx,edx
835         stos dword ptr es:[esi]
836         eloop CicloMultPorMaxPotencia
837 ;En este momento EBP tiene los digitos mas significativos y EBX los siguientes.
838 FinMultEnConv:
839         mov eax,LogMaxPotencia          ;Corresponde a primera multiplicacion.
840         add SumaLogPotencia,eax
841         jnc short SegundaSumaLogs
842         sub UltDWord,4
843         dec CantDWords
844 SegundaSumaLogs:
845         add SumaLogPotencia,eax         ;Corresponde a segunda multiplicacion.
```

```
846         jnc short ObtenerSiguientesDigitos
847         sub UltDWord,4
848         dec CantDWords
849 ObtenerSiguientesDigitos:
850         mov di,offset bufferdos[-1] ;DI = Puntero al principio del buffer
851                                     ;temporario de digitos en pantalla.
852         mov cx,word ptr Exponente ;Cantidad de digitos del numero.
853         add di,cx
854         add di,cx               ;DI = Puntero al ultimo digito.
855         movzx esi,Base          ;ESI = Base.
856         call EBX_A_OtraBase
857         mov cx,word ptr Exponente ;Cantidad de digitos del numero.
858         mov ebx,ebp             ;Convertir los digitos mas significativos.
859         call EBX_A_OtraBase
860         mov ecx,digitos
861         mov eax,Exponente
862         add eax,eax
863         cmp ecx,eax
864         jbe short CantidadDeDigitosAMostrarOK
865         mov ecx,eax
866 CantidadDeDigitosAMostrarOK:
867         test Indicadores,Formato       ;Hay que formatear el texto?
868         jnz short FormatearTexto       ;Saltar si es asi.
869         jecxz FinConversion            ;Saltar si se indico cero decimales.
870         mov bx,handle
871         mov dx,offset bufferdos
872         mov ah,40h
873         call dos
874         mov eax,Exponente
875         shl eax,1
876         sub digitos,eax
877 OtrosDigitos:
878         jbe short FinConversion
879         jmp MultiplicarPorMaxPotencia
880 FormatearTexto:
881         jecxz UltimaLineaTextFormateado
882         call DigitosFormateados
883         cmp Digitos,0
884         jnz OtrosDigitos
885 UltimaLineaTextFormateado:
886         call MostrarBufferEnPantalla
887 FinConversion:
888 Final:
889         mov dx,OldInt8Offset     ;Poner nuevamente el viejo vector de
890         mov cx,OldInt8Segment    ;interrupcion 8 en modo real.
891         mov ax,0201h
892         mov bl,8
893         int 31h
894         mov dx,offset TiempoTotal ;CR - LF
895         mov cx,2
896         mov bx,cx              ;Handle STDERR.
897         mov ah,40h
898         call dos
899         mov dx,offset TiempoTotal
900         mov cx,LongitudTiempoTotal
901         mov bx,HANDLE_STDERR   ;Handle STDERR.
902         mov ah,40h
903         call dos
904         mov ax,0007h           ;Cambiar el descriptor para que apunte al buffer
905         mov bx,Selector        ;de video.
906         mov cx,000Bh
907         mov dx,8000h
908         int 31h
909         call obtencur          ;Obtener (en DI) la posicion del cursor.
910         mov es,Selector
```

```
911         call MostrarTiempo     ;Mostrar el tiempo total de ejecucion.
912         mov dx,offset TiempoTotal ;CR - LF
913         mov cx,2
914         mov bx,cx              ;Handle STDERR.
915         mov ah,40h
916         call dos
917         mov bx,Selector        ;Liberar el selector pedido al servidor de DPMI
918         mov ax,0001h
919         int 31h
920         mov si,HandleMemoriaAlto ;Liberar la zona de memoria pedida.
921         mov di,HandleMemoriaBajo
922         mov ax,0502h
923         int 31h
924         mov ax,4c00h           ;Fin del programa.
925         int 21h
926 ;---------------------------------------------------------------------------
927 ; Subrutina dos: Sirve para llamar al DOS en modo real.
928 ;
929 dos:
930         mov word ptr ModoReal.reg_EAX,ax  ;Poner los registros en la
931         mov word ptr ModoReal.reg_EBX,bx  ;estructura para modo real.
932         mov word ptr ModoReal.reg_ECX,cx
933         mov word ptr ModoReal.reg_EDX,dx
934         mov word ptr ModoReal.reg_ESI,si
935         mov word ptr ModoReal.reg_EDI,di
936         mov word ptr ModoReal.reg_SS,0
937         mov word ptr ModoReal.reg_SP,0
938         mov ax,0300h                      ;Llamar a interrupcion en modo real.
939         mov bx,0021h                      ;DOS = Interrupcion 21h.
940         xor cx,cx                         ;Cantidad de bytes a pasar en stack:
941                                           ;cero.
942         mov di,offset ModoReal            ;Apuntar a la estructura de modo real
943         movzx edi,di
944         push ds
945         pop es
946         int 31h
947         mov ah,byte ptr ModoReal.reg_Flags
948         sahf
949         mov es,Selector
950         mov ax,word ptr ModoReal.reg_EAX
951         mov bx,word ptr ModoReal.reg_EBX
952         mov cx,word ptr ModoReal.reg_ECX
953         mov dx,word ptr ModoReal.reg_EDX
954         mov si,word ptr ModoReal.reg_ESI
955         mov di,word ptr ModoReal.reg_EDI
956         ret
957 ;---------------------------------------------------------------------------
958 ; Manejador de la interrupcion 8 (timer). Debe mostrar la cantidad de digitos
959 ; y el tiempo transcurrido.
960 ;
961 Int8Handler:
962         push ds                       ;Preservar los registros utilizados.
963         push es
964         push esi
965         push di
966         push bx
967         push eax
968         push edx
969         push cx
970         push cs                       ;Cargar el registro de segmento de
971         pop ds                        ;datos.
972         cld
973         mov di,offset FraccSegundo
974         add dword ptr [di],235903917   ;2^32 / 18,20648193
975         jnc short CantDigitosAPantalla
```

```
 976         dec di                          ;Apuntar a los segundos.
 977         inc byte ptr [di]               ;Incrementar segundos.
 978         cmp byte ptr [di],60            ;Se termino el minuto?
 979         jnz short TiempoActualizado     ;Saltar si no es asi.
 980         mov byte ptr [di],0
 981         dec di                          ;Apuntar a los minutos.
 982         inc byte ptr [di]               ;Incrementar minutos.
 983         cmp byte ptr [di],60            ;Se termino la hora?
 984         jnz short TiempoActualizado     ;Saltar si no es asi.
 985         mov byte ptr [di],0
 986         dec di                          ;Apuntar a las horas.
 987         inc byte ptr [di]               ;Incrementar horas.
 988         cmp byte ptr [di],24            ;Se termino el dia?
 989         jnz short TiempoActualizado     ;Saltar si no es asi.
 990         mov byte ptr [di],0
 991         dec di                          ;Apuntar a dias.
 992         inc byte ptr [di]               ;Incrementar horas.
 993         cmp byte ptr [di],100           ;Pasaron cien dias?
 994         jnz short TiempoActualizado     ;Saltar si no es asi.
 995         mov byte ptr [di],0
 996 TiempoActualizado:
 997         cmp MostrarEnInt8,0             ;Hay que mostrar numero y tiempo?
 998         jz FinInter8                    ;Salir de la interrupcion si no es asi.
 999         les di,CursorVideo              ;Obtener la posicion del cursor.
1000         add di,15*2                     ;Apuntar al tiempo.
1001         call MostrarTiempo              ;Mostrar tiempo transcurrido.
1002 CantDigitosAPantalla:
1003         cmp MostrarEnInt8,0             ;Hay que mostrar numero y tiempo?
1004         jz FinInter8                    ;Salir de la interrupcion si no es asi.
1005         les di,CursorVideo              ;Obtener la posicion del cursor.
1006         mov si,PunteroBase
1007         xor eax,eax
1008         test Indicadores,E              ;Se esta calculando el numero e?
1009         jz short NoSeCalculaE
1010         mov edx,67108864                ;2^64 / 64
1011         div dword ptr [si]
1012         mul PrimDWord
1013         jmp short AjustarDigitos
1014 NoSeCalculaE:
1015         mov edx,6291456                 ;2^64 * log 8 / log (2^32) / 64.
1016         div dword ptr [si]              ;2^32 * log 8 / log Base
1017         mul Divisor                     ;EDX = Cantidad de digitos / 64.
1018 AjustarDigitos:
1019         shld edx,eax,6                  ;EDX = Cantidad de digitos.
1020         mov bl,0
1021         mov esi,1000000000
1022         call Div1Digito
1023         mov esi,100000000
1024         call Div1Digito
1025         mov esi,10000000
1026         call Div1Digito
1027         mov esi,1000000
1028         call Div1Digito
1029         mov esi,100000
1030         call Div1Digito
1031         mov esi,10000
1032         call Div1Digito
1033         mov si,1000
1034         call Div1Digito
1035         mov si,100
1036         call Div1Digito
1037         mov si,10
1038         call Div1Digito
1039         add dl,"0"
1040         mov es:[di],dl
```

```
1041 FinInter8:
1042         pop cx                  ;Restaurar los registros afectados.
1043         pop edx
1044         pop eax
1045         pop bx
1046         pop di
1047         pop esi
1048         pop es
1049         pop ds
1050         jmp cs:OldInt8Handler   ;Ir al viejo manejador de INT 8.
1051 ;-----------------------------------------------------------------------------
1052 PonerVectorInterrupcion:
1053         mov dx,offset TextoSegundaLinea
1054         mov cx,LongitudTextoSegundaLinea
1055         mov bx,HANDLE_STDERR    ;Handle STDERR.
1056         mov ah,40h
1057         call dos                ;Mostrar el texto.
1058         call obtencur           ;Obtener la direccion del cursor.
1059         mov CursorVideoOffset,di
1060         mov ax,0200h            ;Obtener el vector de interrupcion (modo real).
1061         mov bl,8                ;Numero de interrupcion.
1062         int 31h
1063         mov OldInt8Offset,dx    ;Preservar el vector.
1064         mov OldInt8Segment,cx
1065         mov ax,0201h            ;Setear el vector de interrupcion (modo real).
1066         mov bl,8                ;Numero de interrupcion.
1067         mov cx,ModoReal.reg_DS
1068         mov dx,offset Int8Handler
1069         int 31h
1070         ret                     ;Fin de la subrutina.
1071 ;-----------------------------------------------------------------------------
1072 ; Subrutina para obtener la posici¢n del cursor en la pantalla.
1073 ; Salida: DI = Posicion en buffer de video que corresponde al cursor.
1074 ;
1075 obtencur:mov dx,port6845     ;DX = Port de control del controlador de video.
1076         mov al,0Eh          ;Indicarle que hay que leer la parte alta
1077         out dx,al           ;(MSB) de la posici¢n del cursor.
1078         jmp $+2
1079         jmp $+2
1080         inc dx              ;Apuntar al port de datos del 6845.
1081         in al,dx            ;Leer el valor.
1082         mov ah,al           ;Ponerlo en la parte alta de AX.
1083         jmp $+2
1084         jmp $+2
1085         dec dx              ;Apuntar al port de control del 6845,
1086         mov al,0Fh          ;Indicarle que hay que leer la parte baja
1087         out dx,al           ;(LSB) de la posici¢n del cursor.
1088         jmp $+2
1089         jmp $+2
1090         inc dx              ;Apuntar al port de datos del 6845.
1091         in al,dx            ;Leer el valor.
1092         add ax,ax
1093         mov di,ax           ;Poner el resultado en DI.
1094         ret                 ;Fin de la subrutina.
1095 ;-----------------------------------------------------------------------------
1096 ; Subrutina MostrarTiempo: Mostrar el tiempo transcurrido.
1097 ; Entrada: ES:DI = Posicion en buffer de video.
1098 ;
1099 MostrarTiempo:
1100         mov si,offset Dias
1101         mov cx,4
1102         cld
1103 CicloMostrarTiempo:
1104         lodsb
1105         aam
```

```
1106         add ax,"00"
1107         mov es:[di],ah
1108         mov es:[di+2],al
1109         mov al,[si-5]
1110         mov es:[di+4],al
1111         mov byte ptr es:[di+6]," "
1112         add di,8
1113         loop CicloMostrarTiempo
1114         ret
1115 ;------------------------------------------------------------------------------
1116 Div1Digito:
1117         mov eax,edx
1118         xor edx,edx
1119         div esi
1120         add al,"0"
1121         cmp al,"0"
1122         jnz short MostrarElDigito
1123         test bl,bl
1124         jz short ApuntarSiguienteDigito
1125 MostrarElDigito:
1126         mov bl,1
1127         stosb
1128         inc di
1129 ApuntarSiguienteDigito:
1130         ret
1131 ;------------------------------------------------------------------------------
1132 ; Subrutina DigitosFormateados: Halla un digito del numero y, si esta
1133 ; habilitado con el switch /F, realiza el formateo de texto.
1134 ;
1135 ; Entrada: bufferdos: Digitos en ASCII.
1136 ;          CX = Cantidad de digitos a mostrar.
1137 ;          DI = Puntero al buffer donde se almacenan los digitos en ASCII.
1138 ;
1139 DigitosFormateados:
1140         push es          ;Preservar puntero al numero en binario.
1141         push ds
1142         pop es
1143         cld
1144         mov si,offset bufferdos
1145         mov dx,si
1146         add dx,word ptr Exponente
1147         add dx,word ptr Exponente
1148         mov ah,0         ;Mostrar los ceros mas significativos como espacios.
1149         test Indicadores,MostrarCantDigitos
1150         jnz short CicloMostrarSinFormatear
1151         test Indicadores,Formato
1152         jnz short MostrarFormateado
1153         mov ah,1         ;Mostrar los ceros mas significativos como ceros.
1154 CicloMostrarSinFormatear:
1155         lodsb
1156         cmp al,"0"
1157         jnz short DigitoSignificativo
1158         test ah,ah
1159         jnz short DigitoSignificativo
1160         cmp si,dx
1161         jz short MostrarDigito
1162         mov al," "
1163         jmp short MostrarDigito
1164 DigitoSignificativo:
1165         mov ah,1
1166 MostrarDigito:
1167         stosb
1168         loop CicloMostrarSinFormatear
1169         std
1170         pop es
```

```
1171         ret
1172 MostrarFormateado:
1173         dec digitos
1174         mov di,PosicionDigitoEnBuffer
1175         mov bx,InformacionDeFormateado  ;BH = Numero de grupo (0-9).
1176                                         ;BL = Digito dentro del grupo (1-5).
1177         inc bx                          ;Indicar siguiente digito del grupo.
1178         cmp bl,6                        ;Fin del grupo?
1179         jnz short PonerDigitoFormateado ;Saltar si no es asi.
1180         mov bl,1                        ;Indicar inicio del grupo.
1181         inc bh                          ;Indicar siguiente grupo.
1182         cmp bh,10                       ;Fin de la linea?
1183         jnz short PonerEspacio          ;Ir a mostrar espacio si no es asi.
1184         push si
1185         call MostrarBufferEnPantalla
1186         pop si
1187         mov di,offset BufferNumeroFormateado
1188                                 ;Borrar el buffer de digitos formateados.
1189         push cx                 ;Preservar contador de digitos a mostrar.
1190         cld
1191         mov cx,75
1192         mov al," "
1193         rep stosb
1194         pop cx                  ;Restaurar contador de digitos a mostrar.
1195         mov bx,HANDLE_STDOUT
1196         mov di,offset BufferNumeroFormateado + 1
1197                                 ;Apuntar al segundo byte del buffer.
1198 PonerEspacio:
1199         inc di                  ;Apuntar al siguiente caracter.
1200 PonerDigitoFormateado:
1201         movsb           ;Copiar el digito en buffer de digitos formateados.
1202         mov PosicionDigitoEnBuffer,di
1203         mov InformacionDeFormateado,bx
1204         inc DigitosMostrados    ;Indicar que se mostro un digito mas.
1205         loop MostrarFormateado
1206         std
1207         pop es
1208         ret                     ;Fin de la subrutina.
1209 ;---------------------------------------------------------------------------
1210 MostrarBufferEnPantalla:
1211         push cx
1212         or Indicadores,MostrarCantDigitos
1213         push esi
1214         mov ebx,DigitosMostrados
1215         mov di,offset BufferNumeroFormateado[74]
1216                                 ;DI = Puntero al final del buffer
1217                                 ;     del numero a mostrar a la derecha.
1218         mov esi,10              ;ESI = Base.
1219         mov cx,13               ;CX = Cantidad de digitos a convertir.
1220         call EBX_A_OtraBase
1221         mov cx,12
1222 CicloEliminarCerosIzq:
1223         inc di
1224         cmp byte ptr [di],"0"
1225         jnz short CerosEliminados
1226         mov byte ptr [di]," "
1227         loop CicloEliminarCerosIzq
1228 CerosEliminados:
1229         pop esi
1230         and Indicadores,not MostrarCantDigitos
1231         mov bx,handle
1232         mov dx,offset BufferNumeroFormateado
1233         mov cx,79
1234         push es
1235         mov ah,40h
```

```
1236         call dos
1237         pop es
1238         pop cx
1239         ret
1240 ;-------------------------------------------------------------------------------
1241 ; Conversion binario (EBX = 32 bits) a otra base.
1242 ;
1243 ; Entrada: EBX = Doble palabra a convertir.
1244 ;          ESI = Base.
1245 ;          CX = Cantidad de digitos a convertir.
1246 ; Salida:  bufferdos: Numero en ASCII.
1247 ;
1248 EBX_A_OtraBase:
1249 bucle_bin_a_dec:
1250         mov eax,ebx             ;Cargar el dividendo.
1251         xor edx,edx             ;Extender el dividendo.
1252         div esi                 ;Realizar la division.
1253         mov ebx,eax             ;Salvar el cociente, que sera el dividendo.
1254         add dl,"0"              ;Convertir a ASCII.
1255         cmp dl,"9"
1256         jbe short PonerDigito
1257         add dl,"A" - ("9" + 1)
1258 PonerDigito:
1259         mov [di],dl             ;Guardar el resto en el buffer temporario
1260                                 ;(de esta manera se obtiene el digito
1261                                 ;"de la derecha" o menos significativo).
1262         dec di                  ;Apuntar al digito anterior.
1263         loop bucle_bin_a_dec    ;Saltar si no se acabo el buffer.
1264         ret                     ;Fin de la subrutina.
1265 ;-------------------------------------------------------------------------------
1266 ; Subrutina switch: Procesa los switches de linea de comandos inicializando
1267 ; los flags correspondientes.
1268 ;
1269 ; Entrada: BX = Puntero a la linea de comandos.
1270 ;
1271 switch: cmp byte ptr [bx],"/"           ;Se encontro el caracter de switch?
1272         jz short SeEncontroSwitch       ;Saltar si es asi.
1273         cmp byte ptr [bx],ASCII_CR      ;Fin de la linea de comandos?
1274         jz short FinSwitch              ;No hay switches si es asi.
1275         inc bx                          ;Apuntar al sig. caracter de la linea.
1276         cmp byte ptr [bx-1]," "         ;Habia un espacio o caracter tab?
1277         jbe switch                      ;Saltearlo si es asi.
1278         lea dx,[bx-1]                   ;DX = Puntero al nombre del archivo.
1279 buscar_fin_de_nombre_de_archivo:
1280         inc bx                          ;Apuntar al siguiente caracter.
1281         cmp byte ptr [bx-1]," "         ;Fin del nombre de archivo?
1282         ja buscar_fin_de_nombre_de_archivo ;Seguir buscando si no es asi.
1283         mov [bx-1],bh                   ;Convertir el nombre a ASCIIZ.
1284         mov ah,3Ch                      ;Intentar crear el archivo.
1285         xor cx,cx
1286         call dos
1287         jc short JmpToError             ;Saltar si no se pudo.
1288         mov handle,ax                   ;Almacenar el handle.
1289 FinSwitch:
1290         ret                             ;Fin de la subrutina.
1291 JmpToError:
1292         jmp error                       ;Ir a la rutina de error.
1293 SeEncontroSwitch:
1294         mov ax,[bx+1]                   ;Obtener el tipo de switch.
1295         and ax,0DFDFh                   ;Convertir a mayusculas.
1296         cmp al,"E"                      ;Se desea calcular el valor de e?
1297         jnz short EsLN2?                ;Saltar si no es asi.
1298         or Indicadores,E                ;Indicar que se desea calcular e.
1299         inc bx                          ;Saltear el switch.
1300         inc bx
```

```
1301         jmp switch                      ;Ir a verificar si hay mas switches.
1302 EsLN2?:
1303         cmp ax,"NL"                     ;Es la primera parte de LN 2?
1304         jnz short EsFormato?            ;Saltar si no es asi.
1305         cmp byte ptr [bx+3],"2"         ;Es la segunda parte de LN 2?
1306         jnz short EsFormato?            ;Saltar si no es asi.
1307         or Indicadores,LN2              ;Indicar que se desea calcular ln 2.
1308         add bx,4                        ;Saltear el switch.
1309         jmp switch                      ;Ir a verificar si hay mas switches.
1310 EsFormato?:
1311         cmp al,"F"                      ;Es el switch de texto formateado?
1312         jnz short EsBase?               ;Saltar si no es.
1313         or Indicadores,Formato          ;Indicar que el texto debe formatearse
1314         add bx,2                        ;Saltear el switch.
1315         jmp switch                      ;Ir a verificar si hay mas switches.
1316 EsBase?:
1317         mov ax,[bx+1]
1318         and al,0DFh                     ;Convertir a mayusculas el primer
1319                                         ;caracter.
1320         cmp ax,":B"                     ;Es base?
1321         jnz JmpToError                  ;Ir a error si el switch es invalido.
1322         mov cl,0                        ;Inicializar la base.
1323         add bx,3                        ;Apuntar a los digitos.
1324 CicloBase:
1325         mov ch,[bx]                     ;Obtener el byte de la linea.
1326         sub ch,"0"                      ;Convertir de ASCII a binario.
1327         cmp ch,9                        ;Es un digito?
1328         ja short FinBase                ;Fin de busqueda de digitos si no es.
1329         mov al,10                       ;Multiplicar la base por 10.
1330         mul cl
1331         jc JmpToError                   ;Error si la base supera 256.
1332         add al,ch                       ;Obtener la nueva base.
1333         cmp al,36                       ;La base es mayor que 36?
1334         ja JmpToError                   ;Error si la base es mayor que 36.
1335         mov cl,al                       ;Preservar el valor.
1336         inc bx                          ;Apuntar al siguiente byte.
1337         jmp CicloBase                   ;Volver a pedir mas digitos.
1338 FinBase:
1339         cmp cl,2                        ;La base es menor que 2?
1340         jb JmpToError                   ;Error si es asi.
1341         mov Base,cl                     ;Almacenar la base.
1342         jmp switch
1343 ;-----------------------------------------------------------------------------
1344 fin_codigo:
1345 codigo  ends
1346         end comienzo
```

```
; Programa que muestra manejo de pantalla y generacion de numeros
; aleatorios a traves de un juego.
;
; Hecho por Dario Alpern el 7 de octubre de 1990.
;
esqui           segment
                assume cs:esqui,ds:esqui,es:esqui,ss:esqui
                org 100h
comienzo:       jmp menu_principal
nombre_programa db 'ESQUI.COM',0
mens_choque     db 30,24,'Crashhh!!! Apriete la barra espaciadora$'
mensm           db 7,24,'metros  Nivel$'
mens1           db 36,1,'DR-ESQUI$'
mens2           db 18,3,'Hecho por Dario Alejandro Alpern el 7/10/1990$'
mens3           db 33,7,'Tabla de Honor$'
mens4           db 34,21,'Velocidad (1-5):$'
mens5           db 23,23,'Entre cero para terminar el programa$'
mens6           db 29,24,'seis a nueve para demo$'
mens7           db 20,18,'Utilice la tecla CTRL para volver al menu$'
mens8           db 28,19,'las teclas SHIFT para moverse$'
mens_record     db 45,24,'Nuevo record!!!$'
recuadro1       db 23,5,'ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»$'
recuadro2       db 23,16,'ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼$'
beep            db 79,24,7,7,'$'
buffer_nombre   db 21,0,'                     '
buffer_nivel    db 2,0,0,0
maximo          db '1) 000501 Dario               $'
                db '2) 000366 Dario               $'
                db '3) 000336 Dario               $'
                db '4) 000288 Dario               $'
                db '5) 000206 Dario               $'
metros          db 0,24,'000000$'
choque          db 0
demo            db 0
posrec          dw 0
lineas_mismos_arboles  db 0
reloj           db 0
espera_tics     db 0
nivel           db 0
cant_arboles    db 0
semilla         dw 0
esquiador       db 0
;
; Subrutina para actualizar la pantalla durante el juego y en demo.
;
actualizar_pantalla:
                mov ax,0601h           ;Mover una línea hacia arriba
                                       ;una región de la pantalla.
                sub cx,cx              ;CH, CL = Fila superior y columna izq.
                mov dx,174Fh           ;DH, DL = Fila inferior y columna der.
                mov bh,7
                int 10h                ;Realizar el movimento de caracteres.
                mov cl,cant_arboles    ;CL = Cantidad de árboles por línea.
                xor ch,ch              ;CX = Cantidad de árboles por línea.
ciclo_poner_arboles:
                push cx                ;Preservar contador de árboles.
                mov ax,semilla         ;AX = Semilla generador nros aleatorios.
                mov dx,75              ;Próxima semilla = (semilla * 75)
                mul dx                 ;                  mod 65537.
                and ax,ax              ;(si la semilla era cero, considerarla
                jnz semilla_no_cero    ;como 65536).
                mov dx,75
semilla_no_cero:sub ax,dx
                jnc guardar_semilla
                inc ax
```

```
 66 guardar_semilla:mov semilla,ax         ;Preservar semiila.
 67                 mov al,80
 68                 mul ah                 ;AH = Número "aleatorio" entre 0 y 79.
 69                 mov dh,23              ;DH = Fila a poner el árbol.
 70                 mov dl,ah              ;DL = Columna a poner el árbol.
 71                 xor bh,bh
 72                 mov ah,2
 73                 int 10h                ;Posicionar el cursor.
 74                 mov ax,0A05h           ;05h = Carácter ASCII del árbol.
 75                 xor bh,bh
 76                 mov cx,1               ;Un sólo carácter (no repetir).
 77                 int 10h                ;Mostrar el árbol en pantalla.
 78                 pop cx                 ;Restaurar el contador de árboles.
 79                 loop ciclo_poner_arboles   ;Cerrar el ciclo.
 80                 cmp demo,cl            ;Modo juego o modo demo?
 81                 jz modo_juego          ;Saltar si el modo es juego.
 82                 mov al,es:046Dh        ;El bit 0 cambia cada 14 segundos.
 83                 ror al,1               ;Llevar este bit al Carry Flag.
 84                 sbb al,al              ;Poner el acumulador en 00h ó FFh.
 85                 stc
 86                 adc al,al              ;Poner el acumulador en 01h ó FFh.
 87                 mov demo,al            ;Almacenarlo. Si vale 01h el esquí se
 88                                        ;moverá preferentemente a la derecha,
 89                                        ;en caso contrario a la izquierda,
 90                                        ;siempre que no pueda ir hacia abajo.
 91                 mov al,esquiador       ;AL = Columna del esquiador.
 92                 mov ah,0-1             ;AH = Nro. de línea analizándose por
 93                                        ;debajo del esquiador para no chocar.
 94                 call analizar_linea_inferior
 95                                        ;Mover automáticamente al esquiador.
 96                 mov dl,esquiador       ;DL = Nueva columna del esquiador.
 97                 jmp short mostrar_esqui;Ir a mostrar el esquí.
 98 modo_juego:     mov ah,2               ;Obtener el estado de las teclas
 99                 int 16h                ;SHIFT en bits 0 y 1 de AL.
100                 mov dl,esquiador       ;DL = Columna del esquiador.
101                 ror al,1               ;El jugador apretó SHIFT derecho?
102                 jnc verif_SHIFT_izq    ;Saltar si no es así.
103                 cmp dl,79              ;Esquiador en la columna derecha?
104                 jz verif_SHIFT_izq     ;Saltar si es así.
105                 inc dl                 ;Una columna hacia la derecha.
106 verif_SHIFT_izq:ror al,1                ;El jugador apretó SHIFT izquierdo?
107                 jnc guardar_col_esqui  ;Saltar si no es así.
108                 or dl,dl               ;Esquiador en la columna izquierda?
109                 jz guardar_col_esqui   ;Saltar si es así.
110                 dec dl                 ;Una columna hacia la izquierda.
111 guardar_col_esqui: mov esquiador,dl    ;Guardar la columna del esquiador.
112 mostrar_esqui:  xor dh,dh              ;Fila del esquí.
113                 mov ah,2
114                 int 10h                ;Posicionar el cursor.
115                 xor bh,bh
116                 mov ah,8               ;Leer el carácter en la posición
117                 int 10h                ;del esquí.
118                 cmp al," "             ;Hay algún árbol?
119                 jz no_hubo_choque      ;Saltar si no es así.
120                 mov ax,0905h           ;05h = Carácter ASCII del árbol.
121                 mov bx,87h             ;Atributo blanco sobre negro + Flash.
122                 mov cx,1               ;Un sólo carácter (no repetir).
123                 int 10h                ;Mostrar el árbol parpadeando.
124                 mov choque,1           ;Indicar que el esquiador chocó.
125                 ret                    ;Fin de la subrutina.
126 no_hubo_choque: mov ax,0A48h           ;48h = Carácter ASCII del esquiador.
127                 xor bh,bh
128                 mov cx,1               ;Un sólo carácter (no repetir).
129                 int 10h                ;Mostrar el esquí.
130                 mov cx,6               ;Cantidad de dígitos de los metros.
```

```
131                 mov bx,offset metros[7] ;Puntero al dígito menos significativo
132 ciclo_act_metros:inc byte ptr [bx]     ;Incrementar la cantidad de metros.
133                 cmp byte ptr [bx],"9"  ;Dentro de rango?
134                 jbe mostrar_metros     ;Saltar si es así.
135                 mov byte ptr [bx],"0"  ;Poner el dígito a cero.
136                 dec bx                 ;Apuntar a sig. dígito más significat.
137                 loop ciclo_act_metros  ;Cerrar ciclo.
138 mostrar_metros: mov bx,offset metros   ;Apuntar a la posición a mostrar metros
139                 call mostrar_texto     ;Mostrar los metros en pantalla.
140                 cmp demo,0             ;Modo juego o modo demo?
141                 jnz act_info_arboles   ;Saltar si modo demo.
142                 push es                ;Preservar segmento extra.
143                 push cs
144                 pop es                 ;ES = Segmento de código.
145                 mov si,posrec          ;SI = Puntero al récord.
146                 mov di,offset metros[2] ;DI = Puntero a los metros.
147                 mov cx,6               ;CX = Cantidad de dígitos.
148                 rep cmpsb              ;Realizar la comparación.
149                 pop es                 ;Restaurar segmento extra.
150                 jae act_info_arboles   ;Saltar si no se llegó al récord.
151                 mov bx,offset mens_record ;Mostrar en la pantalla
152                 call mostrar_texto     ;que el jugador tiene el récord.
153 act_info_arboles: dec lineas_mismos_arboles  ;Decrementar cantidad de líneas
154                                        ;que faltan para incrementar la
155                                        ;cantidad de árboles.
156                 jnz fin_subr           ;Saltar si faltan líneas.
157                 mov lineas_mismos_arboles,50  ;Ahora faltan 50 líneas.
158                 cmp cant_arboles,25    ;Verificar si se muestran 25 por línea.
159                 jz fin_subr            ;Saltar si es así.
160                 inc cant_arboles       ;Incrementar cant de árboles por línea.
161 fin_subr:       ret                    ;Final de la subrutina.
162 ;
163 ; Subrutina recursiva que indica para dónde tiene que ir el esquiador.
164 ;
165 analizar_arboles:cmp al,80             ;Columna fuera de rango?
166                 jnc fin_subrut         ;Salir si es así.
167                 push ax                ;Preservar cantidad de líneas
168                                        ;analizadas por debajo del esquiador
169                                        ;y la columna del esquiador.
170                 mov dx,ax              ;DH = Fila y DL = Columna de análisis.
171                 mov ah,2
172                 int 10h                ;Posicionar el cursor.
173                 mov ah,8
174                 xor bh,bh
175                 int 10h                ;AL = Carácter ASCII leído allí.
176                 cmp al,5               ;Verificar si hay un árbol.
177                 pop ax                 ;Restaurar cantidad de líneas
178                                        ;analizadas por debajo del esquiador
179                                        ;y la columna del esquiador.
180                 jz fin_subrut          ;Saltar si hay un árbol.
181 analizar_linea_inferior:
182                 cmp ah,5               ;Se analizaron 5 líneas?
183                 jz no_hay_choque       ;Saltar si es así (no hay choque).
184                 push ax                ;Preservar cantidad de líneas
185                                        ;analizadas por debajo del esquiador
186                                        ;y la columna del esquiador.
187                 inc ah                 ;Incrementar cantidad de líneas.
188                 call analizar_arboles  ;Verificar si abajo hay un árbol.
189                 add al,demo
190                 call analizar_arboles  ;Verificar si abajo a la derecha (si
191                                        ;demo = 1) o izquierda (si demo = -1)
192                                        ;hay un árbol.
193                 sub al,demo
194                 sub al,demo
195                 call analizar_arboles  ;Verificar si abajo a la izquierda (si
```

```
196                                      ;demo = 1) o derecha (si demo = -1)
197                                     ;hay un árbol.
198                 pop ax              ;Restaurar cantidad de líneas.
199 fin_subrut:     ret                 ;Fin de la subrutina.
200 no_hay_choque:  add sp,18           ;Restaurar el valor de SP.
201                 pop ax              ;Obtener el valor de la columna del
202                                     ;esquiador en AL.
203                 mov esquiador,al    ;Almacenar la columna del esquiador.
204                 pop ax              ;Perder valores innecesarios que
205                 pop ax              ;estaban en la pila.
206                 ret                 ;Fin de la subrutina.
207 ;
208 ; Subrutina para mostrar un texto en pantalla.
209 ;
210 mostrar_texto:  mov dx,[bx]         ;Obtener fila (en DH) y columna (en DL) a
211                                     ;presentar en pantalla el mensaje.
212                 push bx             ;Preservar el puntero a la posición.
213                 xor bh,bh
214                 mov ah,2
215                 int 10h             ;Posicionar el cursor.
216                 pop dx              ;Restaurar el puntero a la posición.
217                 inc dx
218                 inc dx              ;Puntero al texto.
219                 mov ah,9
220                 int 21h             ;Mostrar el texto.
221                 ret                 ;Fin de la subrutina.
222 ;
223 ; Subrutina para borrar pantalla.
224 ;
225 borrar_pantalla:mov ax,0600h            ;Borrar región de la pantalla.
226                 mov bh,7                ;Atributo blanco sobre negro.
227                 xor cx,cx               ;Carácter superior izq. de la región.
228                 mov dx,184Fh            ;Carácter inferior der. de la región.
229                 int 10h                 ;Borrar toda la pantalla.
230                 ret                     ;Fin de la subrutina.
231 ;
232 ; Subrutina para mostrar la parte superior de la pantalla principal.
233 ;
234 pantalla_principal:
235                 call borrar_pantalla    ;Borrar la pantalla.
236                 mov bx,offset mens1     ;Mostrar mensajes zona superior
237                 call mostrar_texto      ;de la pantalla.
238                 mov bx,offset mens2
239                 call mostrar_texto
240                 mov bx,offset mens3
241                 call mostrar_texto
242                 mov bx,offset metros
243                 call mostrar_texto
244                 mov bx,offset recuadro1 ;Mostrar zona superior del recuadro
245                 call mostrar_texto      ;de la tabla de honor.
246                 mov bx,offset recuadro2 ;Mostrar zona inferior del recuadro
247                 call mostrar_texto      ;de la tabla de honor.
248                 mov cx,10               ;Cantidad de líneas del recuadro.
249 ciclo_recuadro: mov dl,23               ;Columna izquierda del recuadro.
250                 call mostrar_doble_linea_vertical
251                 mov dl,57               ;Columna derecha del recuadro.
252                 call mostrar_doble_linea_vertical
253                 loop ciclo_recuadro     ;Cerrar ciclo.
254                 mov cx,5                ;Cantidad récords a mostrar.
255                 mov dx,offset maximo    ;DX = Puntero a tabla de máximos.
256 ciclo_mostrar_records:
257                 push cx                 ;Preservar contador récords.
258                 push dx                 ;Preservar puntero a tabla de máximos.
259                 mov dx,0F19h            ;Columna 25.
260                 sub dh,cl               ;Fila según el nivel.
```

```
261                 xor bh,bh
262                 mov ah,2                ;Posicionar el cursor.
263                 int 10h
264                 pop dx                  ;DX = Puntero a tabla de máximos.
265                 push dx                 ;Preservar puntero a tabla de máximos.
266                 mov ah,9                ;Mostrar una línea de récord.
267                 int 21h
268                 pop dx                  ;DX = Puntero a tabla de máximos.
269                 pop cx                  ;CX = Contador récords.
270                 add dx,31               ;DX = Puntero a siguiente línea.
271                 loop ciclo_mostrar_records  ;Cerrar ciclo.
272                 ret                     ;Fin subrutina.
273 mostrar_doble_linea_vertical:
274                 push cx                 ;Preservar contador récords.
275                 mov dh,5
276                 add dh,cl               ;Fila dependiendo del contador.
277                 xor bh,bh
278                 mov ah,2
279                 int 10h                 ;Posicionar cursor.
280                 mov ax,0ABAh            ;0BAH = ASCII doble línea vertical.
281                 xor bh,bh
282                 mov cx,1                ;Un sólo carácter (no repetirlo).
283                 int 10h                 ;Mostrar la doble línea vertical.
284                 pop cx                  ;Restaurar contador récords.
285                 ret                     ;Fin subrutina.
286 ;
287 ;  Mostrar el menú principal.
288 ;
289 menu_principal: call pantalla_principal ;Mostrar pantalla principal.
290                 mov bx,offset mens4     ;Mostrar parte inferior del
291                 call mostrar_texto      ;menú principal.
292                 mov bx,offset mens5
293                 call mostrar_texto
294                 mov bx,offset mens6
295                 call mostrar_texto
296                 mov bx,offset mens7
297                 call mostrar_texto
298                 mov bx,offset mens8
299                 call mostrar_texto
300 entrar_nivel:   mov dx,1532h            ;Poner cursor en fila 15h
301                 xor bh,bh               ;y columna 32h para poder
302                 mov ah,2                ;entrar el nivel.
303                 int 10h
304                 mov dx,offset buffer_nivel  ;Vaciar buffer de teclado y
305                 mov ax,0C0Ah            ;permitir que el jugador entre
306                 int 21h                 ;el nivel.
307                 mov al,buffer_nivel[2]  ;AL = Nivel (en ASCII).
308                 cmp al,"0"              ;Si el nivel no es cero,
309                 jnz no_terminar_prog    ;seguir.
310                 call borrar_pantalla    ;Borrar la pantalla.
311                 mov ax,4C00h            ;Finalizar el programa.
312                 int 21h
313 no_terminar_prog:jc entrar_nivel        ;Si esta fuera de rango, volver a
314                                         ;entrar el nivel.
315                 cmp al,"9"              ;Si esta fuera de rango, volver a
316                 ja entrar_nivel         ;entrar el nivel.
317                 mov demo,0              ;Suponer que no se pidió demo.
318                 mov ah,"6"              ;Calcular la velocidad.
319                 sub ah,al
320                 ja velocidad_calculada  ;Si el nivel vale entre 1 y 5,
321                                         ;no hay demo. Almacenarla.
322                 add ah,4                ;Convertir a número de nivel.
323                 inc demo                ;Indicar que se pidió demo.
324 velocidad_calculada:
325                 mov nivel,ah            ;Almacenar el número de nivel.
```

```
326                 mov semilla,0           ;Inicializar la semilla para el
327                                         ;generador de números aleatorios.
328                 mov esquiador,40        ;Posición del esquiador.
329                 mov cant_arboles,1      ;Cantidad de árboles por línea.
330                 mov choque,0            ;Poner indicador de choque a cero.
331                 mov espera_tics,1       ;Cantidad de tics a esperar antes
332                                         ;de mover la pantalla hacia arriba.
333                 mov lineas_mismos_arboles,50  ;Cantidad de líneas que faltan
334                                         ;para cambiar la cantidad de árboles.
335                 sub al,"1"              ;AL = Indice según nivel.
336                 mov ah,31               ;Longitud de la línea del récord.
337                 mul ah                  ;AX = Indice al principio de la línea
338                                         ;del récord para el nivel pedido.
339                 add ax,offset maximo[3] ;AX = Puntero al récord.
340                 mov posrec,ax           ;Almacenar el puntero.
341                 mov bx,offset metros[2] ;Inicializar el contador de metros.
342                 mov cx,6                ;Seis dígitos para los metros.
343 ciclo_inic_metros:  mov byte ptr [bx],"0"   ;Poner el carácter a cero.
344                 inc bx                  ;Apuntar al próximo carácter.
345                 loop ciclo_inic_metros  ;Cerrar el ciclo.
346                 call borrar_pantalla    ;Borrar la pantalla.
347 ;
348 ; Comienzo del juego.
349 ;
350                 mov bx,offset mensm     ;Mostrar los textos "metros" y
351                 call mostrar_texto      ;"Nivel".
352                 mov dx,1815h            ;El nivel se debe mostrar en la
353                 xor bh,bh               ;fila 18h, columna 15h.
354                 mov ah,2
355                 int 10h
356                 mov ax,0A36h
357                 sub al,nivel            ;AL = Nivel en ASCII.
358                 mov bx,7                ;Frente blanco sobre fondo negro.
359                 mov cx,1                ;Un sólo carácter (no repetir).
360                 int 10h                 ;Mostrar el nivel.
361                 xor ax,ax
362                 mov es,ax               ;ES = Segmento cero.
363                 mov al,es:046ch         ;Obtener el tic de reloj de BIOS.
364                 mov reloj,al            ;Almacenarlo.
365 ciclo_principal:mov ah,2                ;Obtener el estado de las teclas
366                 int 16h                 ;SHIFT.
367                 and al,4                ;Se apretó alguna tecla CTRL?
368                 jz no_apreto_CTRL       ;Saltar si no es así.
369                 jmp menu_principal      ;Volver al menú principal.
370 no_apreto_CTRL: mov al,es:046Ch         ;Obtener el tic de reloj de BIOS.
371                 cmp reloj,al            ;Compararlo contra el anterior.
372                 jz ciclo_principal      ;Volver si no cambió.
373                 mov reloj,al            ;Actualizar el tic de reloj.
374                 dec espera_tics         ;Ver si pasó el tiempo según el nivel.
375                 jnz ciclo_principal     ;Volver si no es así.
376                 mov al,nivel            ;Esperar nuevamente según el nivel.
377                 mov espera_tics,al
378                 call actualizar_pantalla ;Actualizar la pantalla.
379                 mov ah,2                ;Poner el cursor fuera de la pantalla.
380                 mov dx,1A00h
381                 xor bx,bx
382                 int 10h
383                 cmp choque,1            ;El esquiador chocó contra un árbol?
384                 jnz ciclo_principal     ;Volver si no es así.
385 ;
386 ; Fin del juego.
387 ;
388                 mov bx,offset mens_choque  ;Mostrar mensaje de choque.
389                 call mostrar_texto
390                 mov bx,offset beep      ;Generar sonido.
```

```
391                 call mostrar_texto
392 esperar_espacio:xor ah,ah               ;Esperar a que el jugador apriete
393                 int 16h                 ;alguna tecla.
394                 cmp al," "              ;Es la barra espaciadora?
395                 jnz esperar_espacio     ;Seguir esperando si no es así.
396                 push cs
397                 pop es                  ;ES = Segmento de código.
398                 mov si,posrec           ;SI = Puntero al récord.
399                 mov di,offset metros[2] ;DI = Puntero a la cantidad de metros.
400                 mov cx,6                ;CX = Cantidad de dígitos a comparar.
401                 cld                     ;La cadena se debe procesar en
402                                         ;direcciones ascendentes.
403                 rep cmpsb               ;Realizar la comparación.
404                 jae ir_a_menu_principal ;Saltar si no es récord.
405                 mov si,offset metros[2] ;SI = Puntero a la cantidad de metros.
406                 mov di,posrec           ;DI = Puntero al récord.
407                 mov cx,3                ;CX = Cantidad de palabras a mover.
408                 rep movsw               ;Poner el nuevo récord.
409                 mov al," "              ;Rellenar con espacios.
410                 mov cl,21               ;Cantidad de caracteres.
411                 rep stosb               ;Borrar el nombre anterior.
412                 call pantalla_principal ;Mostrar parte sup. pantalla principal.
413                 mov dx,0F23h
414                 sub dh,nivel            ;Posición del cursor según el nivel.
415                 xor bh,bh
416                 mov ah,2
417                 int 10h                 ;Posicionar el cursor.
418 entrar_nombre:  mov dx,offset buffer_nombre  ;Vaciar buffer de teclado y
419                 mov ax,0C0Ah            ;permitir que el jugador entre
420                 int 21h                 ;su nombre.
421                 mov cl,buffer_nombre[1] ;CL = Cantidad de caracteres entrados.
422                 or cl,cl                ;Hubo alguno?
423                 jz entrar_nombre        ;Entrar de nuevo si no se entró nada.
424                 xor ch,ch               ;Extender a 16 bits.
425                 mov si,offset buffer_nombre[2] ;SI = Puntero al nombre entrado.
426                 mov di,posrec           ;DI = Puntero al récord.
427                 add di,7                ;DI = Puntero al nuevo nombre.
428                 rep movsb               ;Poner el nuevo nombre.
429                 mov ah,3Ch              ;Abrir el archivo ejecutable del
430                 mov dx,offset nombre_programa  ;programa truncándolo a cero
431                 int 21h                 ;bytes.
432                 jc ir_a_menu_principal  ;Volver a menú principal si no se
433                                         ;puede abrir el archivo.
434                 mov bx,ax               ;BX = Handle del archivo.
435                 mov cx,fin_programa - comienzo ;CX = Longitud del programa.
436                 mov dx,offset comienzo  ;DX = Comienzo del programa.
437                 mov ah,40h              ;Escribir el programa en el disco.
438                 int 21h
439                 jc final_anormal        ;Saltar si no se pudo.
440                 mov ah,3eh              ;Cerrar el archivo.
441                 int 21h
442                 jc final_anormal        ;Saltar si no se pudo.
443 ir_a_menu_principal:
444                 jmp menu_principal      ;Volver al menú principal.
445 final_anormal:  mov ah,3eh              ;Cerrar el archivo.
446                 int 21h
447                 mov ax,4c01h            ;Indicar final anormal del programa.
448                 int 21h
449 fin_programa    equ $
450 esqui           ends
451                 end comienzo
```