UNIVERSIDAD LASALLISTA BENAVENTE

ESCUELA DE INGENIERIA EN COMPUTACION

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

CLAVE : 8793 - 16

ARQUITECTURA Y PROGRAMACION DE UN PROCESADOR PENTIUM III

T E S I S

QUE PARA OBTENER EL TITULO DE :

INGENIERO EN COMPUTACION

P R E S E N T A :

Fortunato Garfias Rodríguez

ASESOR :

M. EN I. NOE DE JESUS VELA AGUIRRE

CELAYA, GTO., ENERO 2001.

AGRADECIMIENTOS

A MI QUERIDA ESPOSA
BAHIA BARBARA

Por su gran apoyo, comprensión y amor que me ha brindado durante mí profesionalismo para salir adelante.

A DIOS

Por darme la existencia y la razón de ser, por darme fuerza en los momentos más dificiles.
Por darme la alegría en los momentos más tristes. Por cargarme en los momentos de flaqueza. Y porque siempre está en mi familia.

A MI MADRE
MA. ISABEL

Por todo el impulso, apoyo y amor que me ha brindado durante toda mi vida. Que Dios esté siempre contigo.

A MI PADRE
FORTUNATO

Por su apoyo y comprensión que siempre me ha dado.

A MIS HERMANOS,
ESPECIALMENTE A MONICA

Por sus apoyos y por los momentos que hemos convivido y compartido juntos.

A LA FAMILIA ROMERO GARCIA

Por haberme permitido formar parte de ellos y por su gran apoyo.

A LA UNIVERSIDAD LASALLISTA BENAVENTE

Por haberme dado la oportunidad de seguir adelante en mi vida profesional.

A MI ASESOR, M. en I. NOE DE JESUS VELA AGUIRRE

Por su apoyo para el desarrollo de este trabajo y enriqueciéndolo más.

"G R A C I A S"

Indice general

ANTECEDENTES

Un procesador es un componente muy complejo, y más si se habla de un Pentium III, que es la enésima evolución de un concepto que nació allá por 1976.

Tras un estudio minucioso a cerca de procesadores, concretamente del procesador Pentium III, pude observar que la información que existe en libros y revistas es muy poca. Por otro lado en el *internet*[1] existen muchos concentrados o manuales disponibles, los cuales son demasiado complejos, enormes y en ingles, que a cualquier estudiante o persona interesada en saber a cerca de la *Arquitectura y Programación del Pentium III* le llevaría mucho tiempo en entender y comprender ésta información.

Derivado de lo anterior siento que sería de gran relevancia el presentar un concentrado de información acerca de las características más importantes del procesador Pentium III, así como su programación, de una manera mucho más fácil y concreta.

[1] http://developer.Intel.com/design/PentiumIII/manuals

INTRODUCCIÓN

El corazón de la computadora es el procesador, éste maneja las necesidades aritméticas, de lógica y de control de la computadora. El procesador tiene su origen en la década de los sesenta, cuando se diseño el circuito integrado (IC por sus siglas en ingles) al combinar varios componentes sobre un "chip" de silicio. A principios de los años sesenta Intel introdujo el chip 8008 el cual fue instalado en una terminal, dando así origen a la primer generación de procesadores.

En 1974 el 8008 evolucionó en el 8080. En 1978 Intel produjo la tercer generación de procesadores 8086, para proporcionar alguna compatibilidad con en el 8080. Después, Intel desarrollo una variación del 8086 para ofrecer un diseño ligeramente sencillo y compatibilidad con los dispositivos de entrada/salida de ese momento. Este nuevo procesador, el 8088, fue seleccionado por IBM para su computadora personal en 1981. Una versión mejorada del 8088 es el 80188, y versiones mejoradas del 8086 son los 80186, 80286, 80386, 80486 y el Pentium.

En los siguientes apartados nos abocaremos a realizar un estudio sobre la Arquitectura y Programación del procesador Pentium III, en los cuales nos daremos cuenta de los enormes avances que ha tenido dicho procesador.

Una de las grandes inquietudes por mejorar el procesador ha sido su velocidad, lo cual conlleva a realizar procesadores cada vez más pequeños y sofisticados, tal como lo es el procesador Pentium III, el cual permite velocidades en un rango de los 450 a los 1000 MHz.

La estructura interna de los procesadores Pentium III permite que, usando unas instrucciones nuevas, se pueda realizar la misma operación matemática sobre varios datos al mismo tiempo.

El procesador Pentium III introduce una serie de instrucciones llamadas SIMD, las cuales facilitan los cálculos del ordenador en operaciones matemáticas que trabajan sobre varios datos a la vez. Un ejemplo de ello son los gráficos en 3D, en los que hay que realizar bastantes operaciones. Además cuenta con 70 nuevas instrucciones, altas velocidades en audio y vídeo, memoria cache mejorada, entre otras mejoras.

En los capítulos I y II se verán las características generales que todo procesador tiene, así como la Arquitectura básica de la familia de procesadores Pentium. En el capítulo III se hará una descripción de cada una de las instrucciones, así como el mnemónico que utilizan. En los capítulos IV y V se tratarán temas como: tipos de datos, modos de direccionamiento, llamadas, interrupciones y excepciones. En el capítulo VI se describen las características generales de la Unidad de Punto Flotante, así como la importancia y funcionamiento que ésta tiene dentro del procesador. Por último en el capítulo VII se muestran una serie de ejemplos desarrollados en lenguaje ensamblador, en donde podremos ver el funcionamiento de algunas de las instrucciones y directivas del procesador Pentium III, así como la manera de estructurar un programa en ensamblador.

CAPITULO I

ACERCA DE LOS PROCESADORES

Objetivo.

Explicar las características más comunes de los procesadores, así como sus antecedentes.

Introducción.

La gran mayoría de los procesadores cuentan con una serie de características que los hacen ser únicos. Entre estas características podemos mencionar las conexiones de alimentación, el tamaño en bits, las líneas de datos, las líneas de dirección, las líneas de control, los registros internos, los modos de direccionamiento, entre otros. En este capitulo se verán cada una de éstas características, así como la evolución del procesador.

1.1 ¿Qué es un procesador[1]?

Un procesador puede definirse como una pastilla de muy alta escala de integración (VLSI), que realiza tareas de la unidad central de procesamiento de una computadora u otro sistema de control automático.

También se puede definir al procesador como un circuito integrado (CI) que tiene la mayoría de las capacidades de procesamiento de las grandes computadoras.

1.2 Características más comunes de los procesadores

La mayoría de los procesadores se caracterizan por tener características tales como:

- Conexiones de alimentación.
- Tamaño en bits.
- Líneas de datos.
- Líneas de dirección.
- Líneas de control.
- Registros internos.
- Modos de direccionamiento.

1.2.1 Conexiones de alimentación

La mayoría de los procesadores requieren una fuente de alimentación regulada de 5 v DC (corriente directa).

1.2.2 Tamaño en bits

Los procesadores se clasifican generalmente en unidades de 4,8,16 ó 32 bits. El tamaño en bits de un procesador a veces se denomina tamaño de palabra.

1.2.3 Líneas de datos

Por estas líneas o bus de datos bidireccional, el procesador transfiere datos e instrucciones entre la MPU (unidad microprocesadora) y memoria (E/S).

1.2.4 Líneas de dirección

Estas líneas o bus de dirección son utilizados para direccionar la memoria. Un bus de direcciones más ancho nos permite direccionar memorias mucho más grandes.

1.2.5 Líneas de control

La mayoría de los procesadores se caracterizan por tener todos o algunas de las siguientes líneas de control:

- Registros de control.
- Líneas de lectura/escritura
- Líneas de entrada/salida.
- Líneas de interrupción.
- Líneas de reinicialización.
- Líneas de control del bus.
- Líneas de status del ciclo.

1.2.6 Registros internos

La mayoría de los procesadores contienen todos o algunos de los siguientes registros:

- *Contador del programa:* El contador del programa (PC) es el registro que contiene la dirección de la siguiente instrucción del programa. La longitud del PC es igual que la anchura del bus de direcciones.

- *Acumulador:* El acumulador es el registro o registros asociados a las operaciones de E/S.

- *Registros de entrada o señalizados:* El registro del status está en todos los procesadores. Los bits individuales del registro se llaman señalizadores. Las condiciones de los señalizadores se asocian, generalmente, a las operaciones de la ALU y son utilizados por instrucciones de bifurcación posteriores para tomar decisiones.

- *Registros de propósito general:* Los registros de propósito general pueden utilizarse para almacenar datos temporalmente o para que contengan una dirección.

- *Registros índice:* El registro índice se utiliza para que contenga la dirección de un operando cuando se utiliza el modo de direccionamiento indexado.

- *Registro de puntero de pila:* El puntero de pila (SP) es un registro especializado que sigue la pista de la siguiente posición de memoria disponible en la pila. La pila es un área reservada de la RAM utilizada para almacenamiento temporal de datos, direcciones de vuelta y contenido de registros. La pila se utiliza durante las llamadas a subrutina y durante interrupciones.

1.2.7 Modos de direccionamiento

El modo de direccionamiento es la técnica utilizada para buscar el operando deseado durante la ejecución de una instrucción.

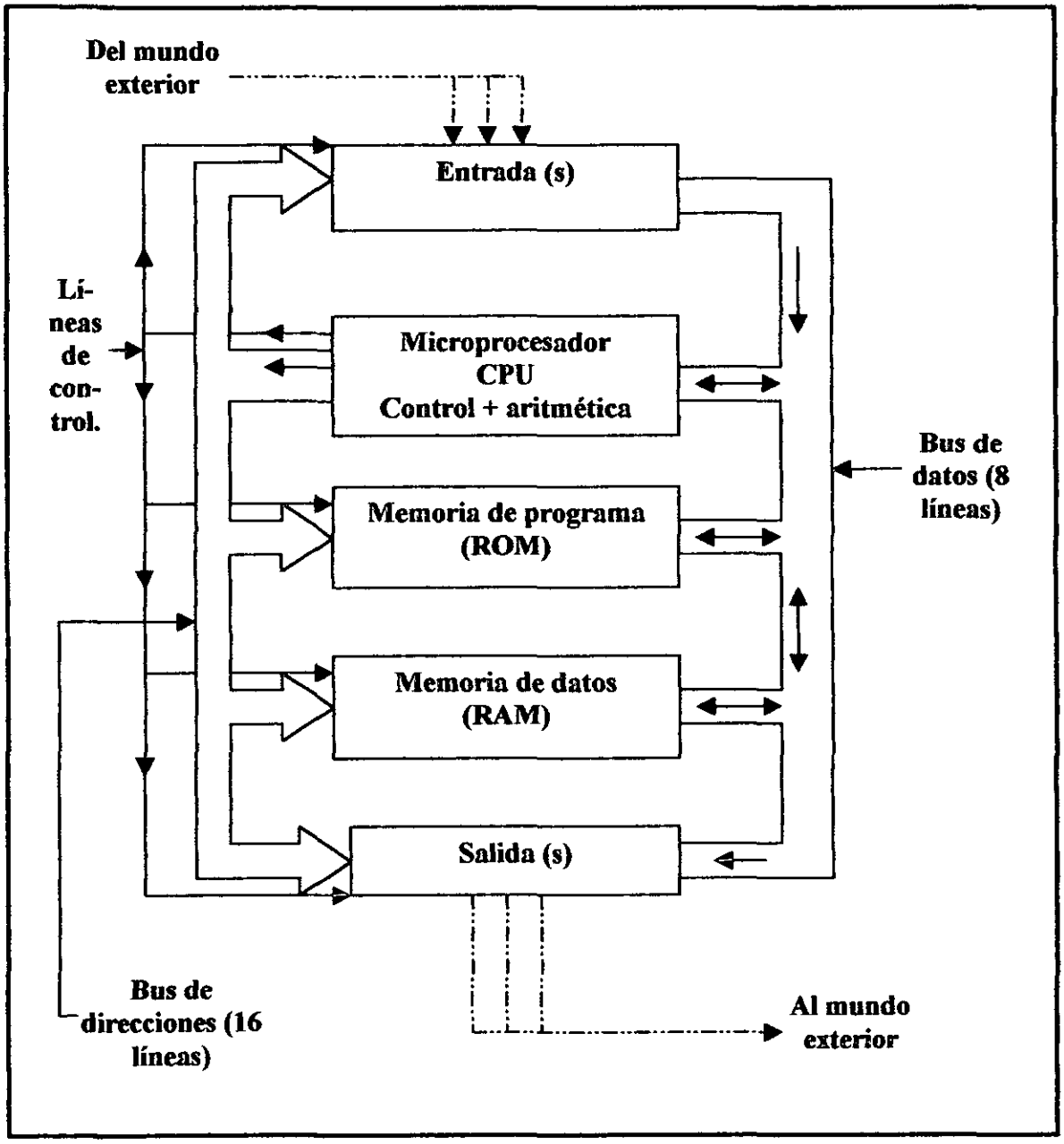
1.3 Influencia del procesador en un sistema microcomputador (computadora digital)

El procesador generalmente forma la sección de la CPU de un sistema microcomputador, cuya organización se muestra en la figura 1-1.

El procesador controla todas las unidades del sistema utilizando las líneas de control que aparecen a la izquierda de la figura 1-1. El bus de direcciones que aparece junto a las líneas de control selecciona una cierta posición de memoria, puerto de entrada o salida. El bus de datos (a la derecha de la figura 1-1) es un camino de doble dirección que se utiliza para introducir y sacar datos de la unidad microprocesadora.

Cualquier proceso que se quiera realizar, tal como sumar 2 números, transferir un dato, etcétera, es controlado por el procesador.

Figura 1-1 Organización de un sistema microprocesador[2]



1.4 Breve historia de la Arquitectura Intel

El desarrollo destacado de la AI (Arquitectura Intel) puede ser tomado a partir de los microprocesadores 8085 y 8080, y el microprocesador 4004 (Primer microprocesador diseñado por Intel en 1969). El primer procesador actual en la familia de la AI es el 8086. Poco después surge el 8088.

El procesador 8086 tenía un registro de 16 bits y un bus de datos de 16 bits, con 20 bits de direccionamiento, 1 MByte de espacio de direcciones. El 8088 es idéntico, excepto que tiene un pequeño bus de datos externo de 8 bits. En estos procesadores se introdujo la segmentación, pero únicamente en "Modo Real".

El procesador 80286 introdujo el modo de protección en la AI. Este nuevo modo utiliza el contenido del segmento de registros como selectores o apuntadores en las tablas descriptoras.

Los descriptores proporcionan 24-bits de direccionamiento base, permitiendo una memoria física de 16 MBytes.

El procesador Intel 386 introduce registros de 32 bits en su arquitectura, utilizándolos como direccionamiento. Se reduce a la mitad cada registro de 32 bits, reteniendo las propiedades de uno de los registros de 16 bits de las generaciones pasadas, lo que provee de cierta compatibilidad. Provee un nuevo modo virtual 8086, el cual brinda gran eficiencia cuando se están ejecutando programas creados por los procesadores 8086 y 8088 en la nueva máquina de 32 bits. El direccionamiento de 32 bits da un espacio de dirección de 4 GBytes, además permite que cada segmento se alargue hasta los 4

GBytes. Las instrucciones originales fueron aumentadas, con nuevos operandos y formas de direccionamiento de 32 bits. Por otro lado, se incluyen instrucciones para la manipulación de bits. El procesador Intel 386 introduce la paginación en la AI, con un tamaño de página de 4 Kbyte, produciendo un método para la administración de la memoria virtual, el cual fue significativamente superior, en comparación, para el estado de los segmentos.

Además, la capacidad para definir segmentos es tan larga como el espacio físico de dirección (4 GBytes); junto con la paginación, dejan creado el modelo "flot moder", el cual es un modelo que protege los sistemas de direccionamiento en la arquitectura, incluyendo implementaciones completas de uso central en el sistema operativo UNIA.

Requiere únicamente un espacio de 32 bits de dirección para accesar a algún espacio de direccionamiento.

El procesador Intel 486 agrega más ejecuciones paralelas compatibles con la decodificación de instrucciones del procesador Intel 386 y unidades de ejecución de cinco estados pipeline, donde cada estado (cuando necesita) opera en paralelo con las otras, arriba de las 5 instrucciones, en diferentes estados de ejecución. Cada estado puede hacer este trabajo en una instrucción y en un ciclo de reloj, y de este modo el procesador Intel 486 puede ejecutarse más rápidamente en una instrucción por ciclo de reloj del CPU.

El procesador Intel 486 agrega la memoria L1 cache, la cual incrementa el porcentaje de instrucciones que puede ejecutarse; en una escala porcentual del ciclo de reloj: la memoria de acceso a instrucciones estará incluida si los operandos están en la L1 cache.

Más tarde, en la generación del procesador Intel 486 se incorporan varias características diseñadas para economizar energía y otros sistemas de administración.

El procesador Intel Pentium agrega una segunda ejecución pipeline que alcanza un rendimiento superescalar (dos pipeline, conocido como U y V, además puede ejecutar dos instrucciones por ciclo de reloj).

Utiliza el protocolo MESI para soportar una mayor eficiencia en modo "write-back", y en el modo "write-though", que es usado por el procesador Intel 486 .

Los registros principales son todavía de 32 bits, pero las rutas de datos internos son de 128 y 256 bits, los cuales puedan ser incrementados hasta los 64 bytes.

El interruptor de control de avance programable (APIC) ha sido agregado para el reporte de sistemas con múltiples procesadores Pentium, y nuevos Pins y un modo especial (Procesamiento dual) diseñado para soportar dos sistemas de procesador.

El procesador Intel Pentium Pro introdujo la "ejecución dinámica". Esta tiene una arquitectura superescalar de "three-way", lo cual quiere decir que pueda ejecutar tres instrucciones por ciclo de reloj del CPU.

Con esto, existe más paralelismo que en el procesador Pentium. Tres instrucciones decodifican las unidades de trabajo en paralelo para decodificar el código del objeto en pequeñas operaciones llamadas "micro-ops"; éstas van

en una instrucción *pool* y pueden ser ejecutadas fuera del orden de las cinco unidades de ejecución en paralelo.

El procesador Pentium Pro expande el bus de direcciones a 32 bits, dando un espacio físico de direcciones de 64 Bytes.

El procesador Pentium II agrega instrucciones MMX para la arquitectura del procesador Pentium Pro, incorpora dos nuevos slots, slot 1 y slot 2, muestra una técnica de empacación. Estas nuevas técnicas de empacación mueven la L2 cache "off – Chip" or "off – dic". El slot 1 y el slot 2 utilizan un único conector, en lugar de un socket. El procesador Pentium II expande la L1 dato cache y L1 instrucciones cache de 16 KBytes cada uno, la L2 cache es de 256 Kbytes, 512 Kbytes y 1 Mbyte o 2 Mbyte (slot 2 únicamente).

El procesador Pentium utiliza múltiples estados de bajo poder, tal como el "Auto HALT", "Stop – Grant", "Sleep" y "Deep Sleep" para conservar el poder durante el tiempo desocupado (parado).

El nuevo procesador en la AI es el procesador Pentium III, el cual está basado en la arquitectura de los procesadores Pentium Pro y Pentium II. El procesador Pentium III introduce 70 nuevas instrucciones en el set de instrucciones de la AI. Estas instrucciones proporcionan una gran funcionalidad dentro de la arquitectura, como lo es la nueva instrucción SIMD – unidad de punto flotante. Más adelante se darán a conocer cada una de estas nuevas instrucciones, así como las mejoras que este nuevo procesador nos proporciona.

En la tabla 1-1 se muestra un resumen de las características más importantes que proporciona cada uno de los procesadores Intel.

Tabla 1-1 Características más comunes de los procesadores Intel[3]

Procesador Intel	Fecha de introducción del producto	Representación en MPI's	Frecuencia Max. al introducirse	No. De transistores.	Tamaño del registro principal del CPU	Tamaño del bus externo de datos	Espacio de direcciones externo	Caches en el CPU
8086	1978	0.8	8 MHz	29 K	16	16	1 MB	No
Intel 286	1982	2.7	12.5 MHz	134 K	16	16	16 MB	Nota
Intel 386 DX	1985	6.0	20 MHz	275 K	32	32	4 GB	Nota
Intel 486 DX	1989	20	25 MHz	1.2 M	32	32	4 GB	8 KB L1
Pentium	1993	100	60 MHz	3.1 M	32	64	4 GB	16 KB L1
Pentium Pro	1995	440	200 MHz	5.5 M	32	64	64 GB	16 KB L1; 256 KB o 512 KB L2
Pentium II	1997	466	266 MHz	7 M	32	64	64 GB	32 KB L1; 256 KB o 512 KB L2
Pentium III	1999	1000	500 MHz	8.2 M	32 GP 128 SIMD-FP	64	64 GB	32 KB L1; 512 KB L2

Nota: El procesador Intel 386 tuvo 8 bytes en el descriptor de caches por cada segmento de registro. El procesador Intel 486 algunas caches del procesador Intel 386, tal como la cache L1 de 8 KBytes de propósito general.

1.5 Introducción a la microarquitectura de la familia de procesadores P6

La familia de procesadores P6 (introducida por Intel en 1995) representa un gran avance tecnológico en los nuevos procesadores de la familia AI. Como predecesores de esta familia tenemos al procesador Pentium introducido por Intel en 1993 y el Pentium Pro, con un avance superescalar en su microarquitectura.

Una de las metas en el diseño de procesadores de la familia P6 fue el incrementar el rendimiento del procesador Pentium.

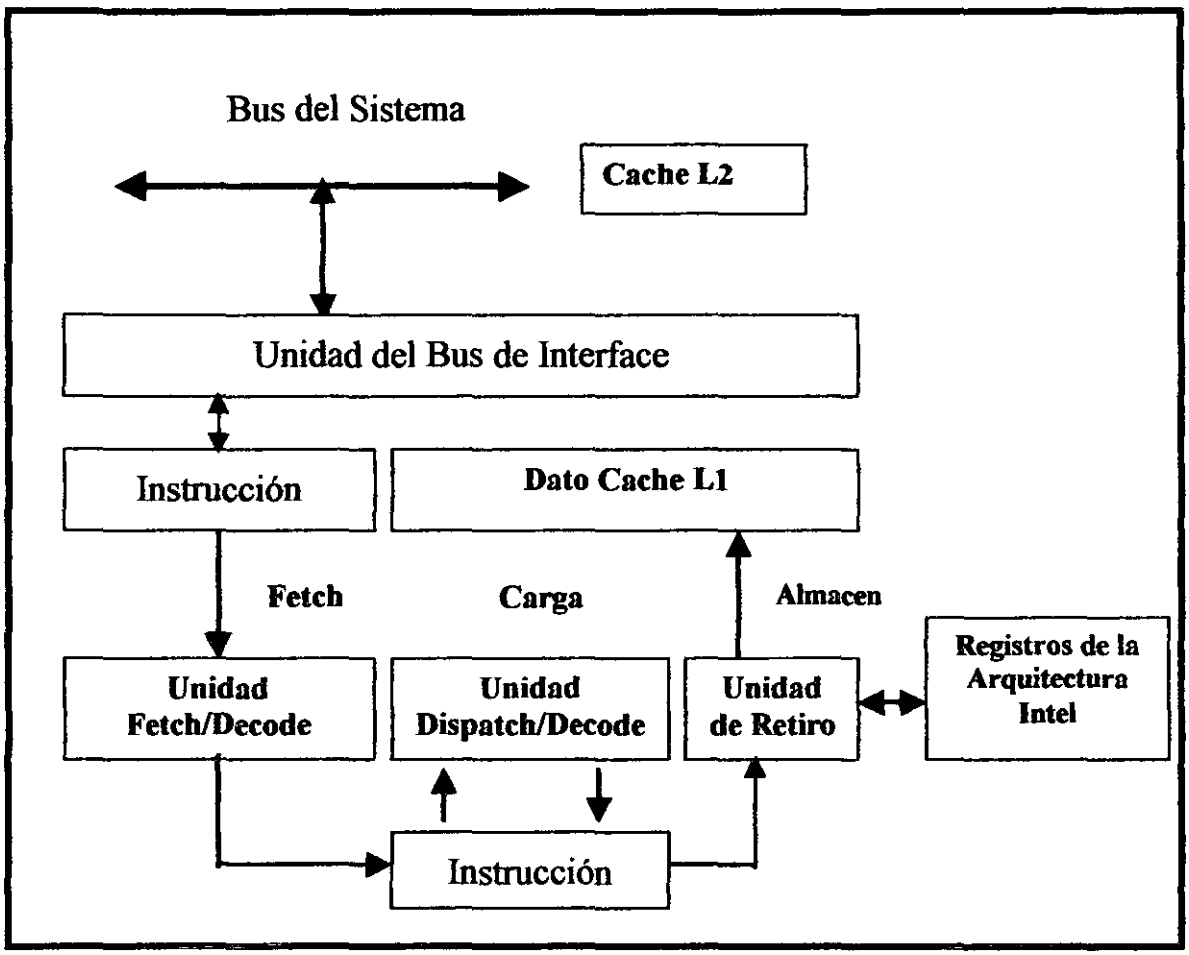
Uno de los avances de la microarquitectura de la familia de los procesadores P6 es el "three-way superescalar" y la arquitectura pipeline. El término "three-way superescalar" significa que usa técnicas de procesamiento en paralelo, con lo cual el procesador es capaz de codificar, despachar y completar la ejecución (retirarse) de tres instrucciones por ciclo de reloj. Para poder llevar acabo estas instrucciones, la familia de procesadores P6 usa un decoplador, doce etapas superpipeline que soporta ejecuciones "out-of-oder" (fuera de orden). En la figura 1-2 se muestra una vista conceptual de este pipeline, con el pipeline dividido en cuatro unidades de procesamiento (la unidad de "fetch/decode", unidad "dispatch/execute", unidad "retire" y la instrucción *Pool*). Las instrucciones y datos son suministrados por estas unidades a través de la unidad de interface de bus.

La familia de procesador P6 incorpora dos niveles de cache. La cache L1 proporciona una instrucción de 8 KByte y un dato cache de 8 Kbyte. La cache L2 es de 256 KByte, 512 KByte o un Mbyte de RAM estática.

La parte central de la familia de procesadores P6 es la innovación out-of-order, llamada "dynamic execution". La ejecución dinámica incorpora tres conceptos en el procesamiento de datos, los cuales son los siguientes:

- *Deep branch prediction* (predicción de ramas profundas).
- *Dynamic data flow analysis* (Análisis dinámico del flujo de datos)
- *Speculative execution* (Ejecución especulativa)

Figura 1-2 Vista conceptual del Pipeline[4]



La predicción de ramas permite al procesador codificar instrucciones más allá de las ramas para conservar la instrucción cuando el pipeline se llena. En la familia de procesadores P6, la unidad de instrucción fetch/decode utiliza un algoritmo de predicción de ramas para producir la dirección de la instrucción que corre, a través de múltiples niveles de ramas, llamadas a procedimientos y retornos.

El análisis dinámico de flujo de datos implica un tiempo real de análisis de flujo de datos a través del procesador para determinar la dependencia del dato y registro y para detectar las oportunidades para la ejecución de instrucción "out-of-order".

La unidad "dispatch/execute" de la familia de los procesadores P6 puede simultáneamente monitorear muchas instrucciones y ejecutar estas instrucciones en el orden que optimiza el uso de la unidad de múltiple ejecución del procesador, mientras mantiene la integridad de los datos.

La ejecución especulativa se refiere a la habilidad del procesador para ejecutar instrucciones antes del contador del programa, pero al final entrega los resultados en el orden de la instrucción general. Para hacer posible la ejecución especulativa, la familia de procesadores P6 desune el despachador y ejecución de instrucciones.

A través de la predicción de ramas profundas, del análisis dinámico de flujo de datos y de la ejecución especulativa, la ejecución dinámica remueve la fuerza de la secuencia de instrucción lineal entre el *fetch* tradicional y las faces de ejecución de instrucción.

La arquitectura del procesador Pentium Pro es la base de la arquitectura para los procesadores que le prosiguen.

El procesador Pentium II y el ahora Pentium III están basados en la arquitectura del procesador Pentium Pro.

---

[1] En ROGER L. Tokheim, *Mc Graw Hill*, FUNDAMENTOS DE LOS MICROPROCESADORES, 2a. ed., México, D.F., MC GRAW HILL, 1996, p.100

[2] En ROGER L. Tokheim, *Mc Graw Hill*, FUNDAMENTOS DE LOS MICROPROCESADORES, 2a. ed., México, D.F., MC GRAW HILL, 1996, p.4

[3] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 2-5

[4] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 2-7

CAPITULO II

ARQUITECTURA BÁSICA

Objetivo.

Explicar la arquitectura básica de la familia de Procesadores Pentium.

Introducción.

En este capítulo se describirá la arquitectura básica de la familia de Procesadores Pentium, así como su modo de operación, la organización de la memoria, los registros con los que cuenta, el apuntador de instrucción. Dicha Arquitectura nos permitirá entender mejor el funcionamiento de éste tipo de procesadores.

2.1 Descripción detallada de la microarquitectura de la familia de procesadores P6.

En la figura 2-2 se muestra un diagrama en bloques funcional de la microarquitectura de la familia de procesadores P6. En este diagrama se muestran las cuatro unidades de procesamiento y subsistema de memoria; las cuales son las siguientes:

- Subsistema de memoria.- bus de sistema, cache L2, unidad de interface del bus, instrucción cache (L1), unidad de datos cache (L1), unidad de memoria de interface, buffer de la memoria de orden.

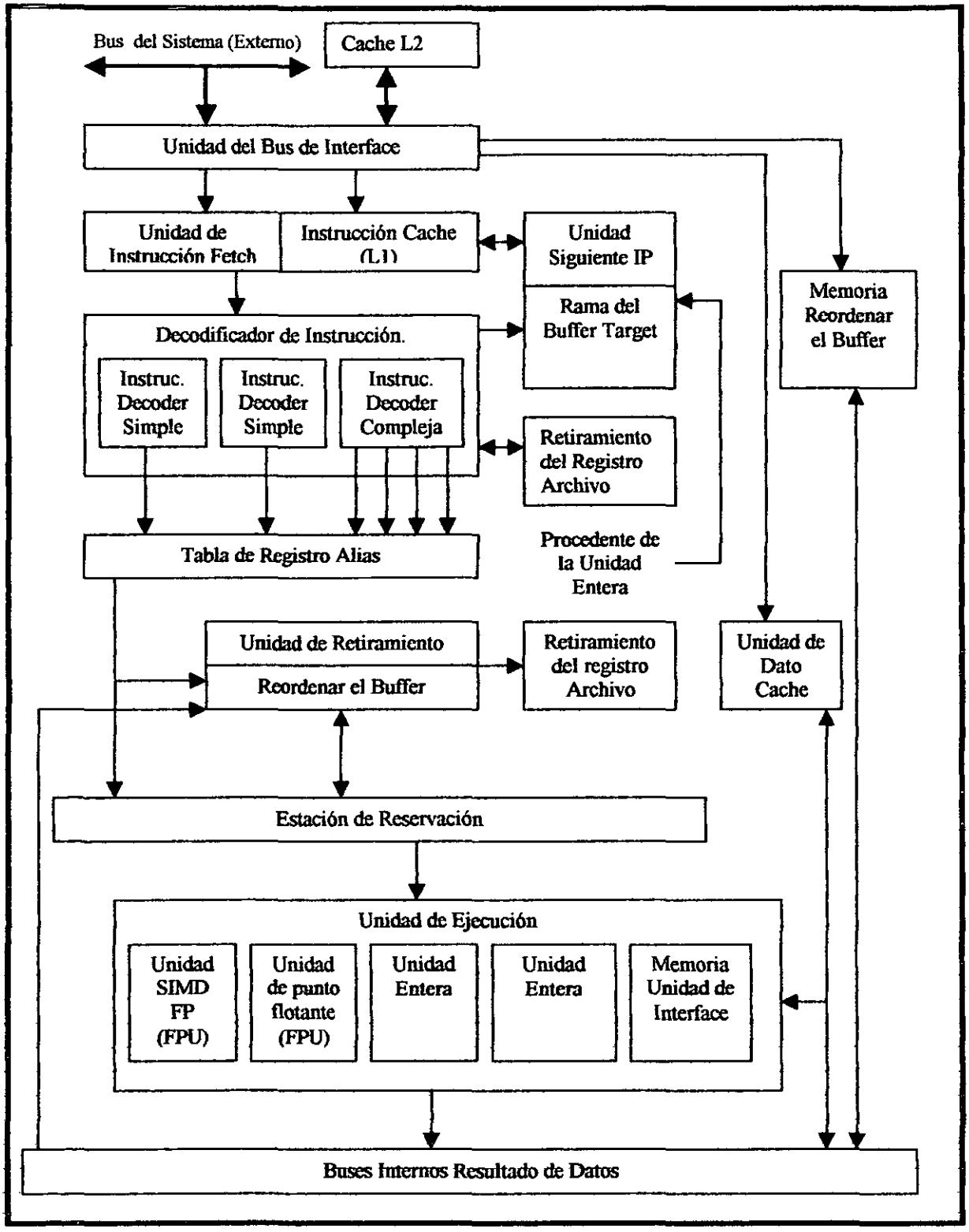
- Unidad "*fetch*/decode".- unidad de instrucción *fetch*, buffer de ramas, decodificador de instrucción, secuencia de microcódigo y tabla de registro alias.

- Instrucción *Pool*.- Reordenar el buffer.

- Unidad "dispatch/execute".- Reservación de estación, dos unidades de enteros, una unidad de punto flotante x87, dos unidades de generación de direcciones y dos unidades de punto flotante SIMD.

- Unidad de retiro y retiramiento del registro de archivo.

Figura 2-1 Diagrama a bloques funcional de la microarquitectura de procesadores de la familia P6[1]



2.1.1 Subsistema de memoria

El subsistema de memoria para los procesadores P6 consiste en un sistema principal de memoria, la cache primaria (L1), y la cache secundaria (L2). La unidad del bus de interface accesa al sistema de memoria a través del bus del sistema externo. Estos 64 bits del bus están en el bus de operación-orientación, sentido en el que cada acceso al bus es encargado de separar la petición y responder a las operaciones. Mientras que la unidad del bus de interface está esperando para responder a un bus de petición, éste puede dar numerosas peticiones.

La unidad del bus de interface accesa el "close/coupled" cache L2 a través de los 64 bits del bus cache. Este bus es además una operación orientada para soportar cuatro concurrencias de acceso cache y operar en el rápido llenado del reloj del procesador.

El acceso para las caches L1 es a través de los buses internos, y por el "full clock speed". Los 8 KByte de la instrucción cache L1 son cuatro caminos en asociación; los 8 KByte del dato cache L1 están en el "dual-ported" y dos caminos asociados soportando una carga y una operación por ciclo de reloj.

2.1.2 Unidad fetch/decode

En fetch/decode lee un grupo de instrucciones de la AI procedentes de la instrucción cache L1 y las decodifica en una serie de micro operaciones llamadas "micro-ops". Este micro op esta entonces enviando a la instrucción Pool.

La unidad de instrucción *fetch* computa la instrucción pointer basado en entradas del bus de ramas, el estado de excepción/interrupción y de las indicaciones de la predicción de ramas procedentes de las unidades de ejecución entera. La parte más importante de este proceso es la predicción de ramas llevada a cabo por el buffer de ramas.

La instrucción *decoder* contiene tres decodificadores paralelos: dos decodificadores de instrucción simple y una instrucción de decodificación compleja.

Cada decodificador decodifica una instrucción de la AI en una o más *triadic* "micro-ops" (Dos fuentes lógicas y un destino lógico por "micro-ops"). Los "micro-ops" son instrucciones primitivas que se están ejecutando por seis unidades de ejecución paralela del procesador. Muchas instrucciones de la AI están convertidas directamente en sencillos "micro-ops" por una simple instrucción de decodificadores y algunas instrucciones están decodificadas en uno de cuatro ""micro-ops""; las instrucciones más complejas de la AI están decodificadas en secuencia de programación "micro-ops" obtenidas de la secuencia de la instrucción de microcódigo.

2.1.3 Instrucción Pool (Reordenar el buffer)

El reordenar el buffer es un arreglo del contenido del direccionamiento de la memoria, arreglado en registro de 40 "micro-ops".

La unidad "dispatch/execute" puede ejecutar instrucciones del reordenador del buffer en algún orden.

2.1.4 Unidad dispatch/execute

La unidad "dispatch/execute" es una unidad "out-of-order" que programa y ejecuta los "micro-ops" almacenados en el ordenador del buffer según la dependencia de datos, recursos disponibles y los resultados almacenados temporalmente de estas ejecuciones especulativas.

2.1.5 Unidad de retiramiento

La unidad de retiramiento entrega los resultados de la ejecución especulativa "micro-ops" a un estado de máquina permanente y remueve el "micro-ops" procedente del ordenador del buffer.

La unidad de retiramiento (semejante a la estación de reservación) continuamente checa el estado del "micro-ops" en el ordenador del buffer.

La unidad de retiramiento puede retirar tres "micro-ops" por ciclo de reloj. Al retirar un "micro-op", ésta escribe los resultados en la memoria del registro archivo and/or. El retiramiento del registro archivo contiene los registros de la AI (8 registros de propósito general y 8 registros de datos de punto flotante). Después los resultados deben ser entregados al estado máquina, el "micro-op" es removido hacia el ordenador del buffer.

2.2 Modos de operación

La Arquitectura Intel soporta tres modos de operación: Modo de protección, modo de direccionamiento real y modo de administración de sistemas.

1. *Modo de protección:* es el estado nativo del procesador. En este modo las instrucciones y características de la arquitectura están disponibles, a condición de alto rendimiento y capacidad. Este es el modo recomendado para todas las aplicaciones nuevas y sistemas de operación.

2. Entre las capacidades del *modo de protección* está la habilidad para ejecutar directamente "El modo de dirección real" 8086, en un medio ambiente multitarea. Esta característica es llamada modo virtual 8086, aunque actualmente este no es un modo de protección. El modo virtual 8086 es actualmente un atributo de modo de protección que puede ser habilitado por alguna tarea.

3. *Modo de dirección real.* Proporcionado por el ambiente de programación del procesador Intel 8086 con algunas extensiones, tal como la habilidad para interrumpir el modo de protección y el modo de administración del sistema el procesador es puesto en modo de dirección real después de aplicar un *reset* al procesador.

4. *Modo de administración del sistema.* Una única característica estándar de la arquitectura para todo los procesadores Intel comienza con el procesador Intel 386. Este modo provee un sistema de operación o ejecución con un mecanismo transparente para implementar funciones de plataforma específica, tal como lo es el poder de administración y seguridad del sistema. El procesador entra en modo de administración de sistema cuando el *pin* de interrupción externo SMM (SM1#) es

activado, o un SMI es recibido por el interruptor de control de avance programable (APIC).

2.3 Organización de la memoria

La memoria que el procesador direcciona en este bus es llamada memoria física. La memoria física es organizada como una secuencia de 8-Bit Bytes. Cada Byte es asignado a una única dirección, llamada dirección física. El espacio de dirección física está en un rango de cero hasta 2^{36-1} (64 Gigabytes).

2.4 32 bits Vs 16 bits dirección y tamaño de operandos

El procesador puede estar configurado por 32 bits ó 32 bits de dirección y tamaño de operandos. Con los 32 bits de dirección y tamaño de operandos, la dirección real máxima o segmento *offset* (compensar) es FFFFFFFFH (2^{-32}) y el tamaño del operando típico es de 8 ó 16 bits. Con 16 bits de dirección y tamaño de operandos, la dirección lineal máxima o segmento offset es FFFF (2^{-16}) y el tamaño de operandos es de 8 ó 16 bits.

Cuando se están usando 32 bits de direccionamiento, la dirección lógica consiste de 16 bits de segmento selector y 32 bits offset. Cuando se están usando 16 bits de direccionamiento, ésta consiste de un segmento selector de 16 bits y un offset de 16 bits.

Cuando se esta operando en modo de protección, el segmento descriptor define la dirección y tamaño de operando *defaul* para el segmento de código que se está ejecutando actualmente. Un segmento descriptor es una estructura de sistema de datos no normalmente visible para el código de la aplicación.

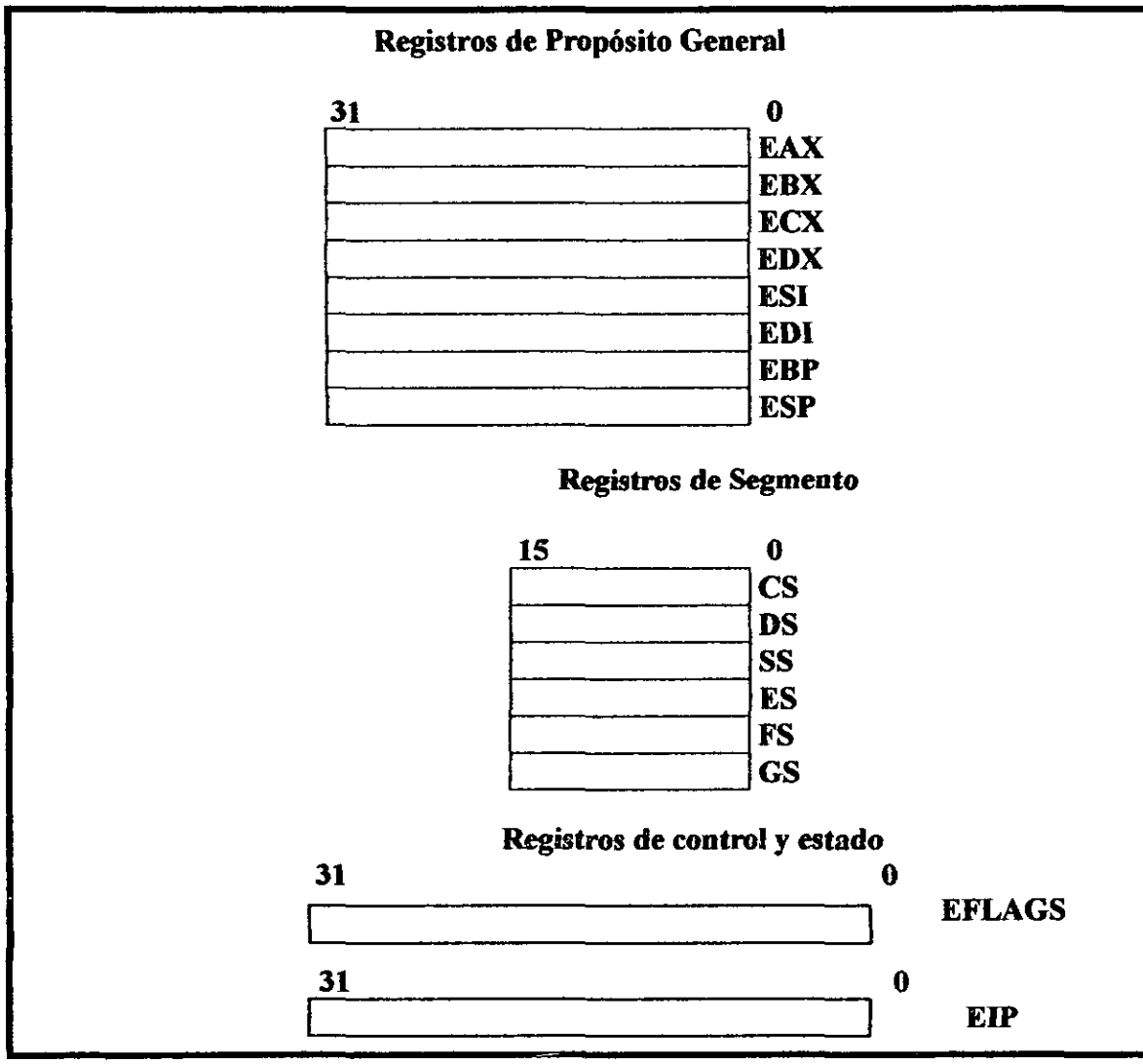
Cuando se esta operando en modo de dirección real, la dirección *defaul* y tamaño de operando es de 16 bits.

2.5 Registros

El procesador contiene 16 registros para uso general del sistema y programación de aplicaciones. Como se puede ver en la figura 2-2, estos registros son agrupados de la siguiente manera:

- *Registros de datos de propósito general.* Formado de ocho registros, los cuales están disponibles para almacenar operandos y apuntadores.
- *Registros de segmento.* Formado por seis registros de segmentos selectores.
- *Registros de estado y control.* Estos registros informan y permiten la modificación del estado del procesador o el programa al ser ejecutado.

Figura 2-2 registros del procesador[2]



2.5.1 Registros de datos de propósito general.

Los 32 bits del registro de datos de propósito general están divididos en los siguientes ocho registros.

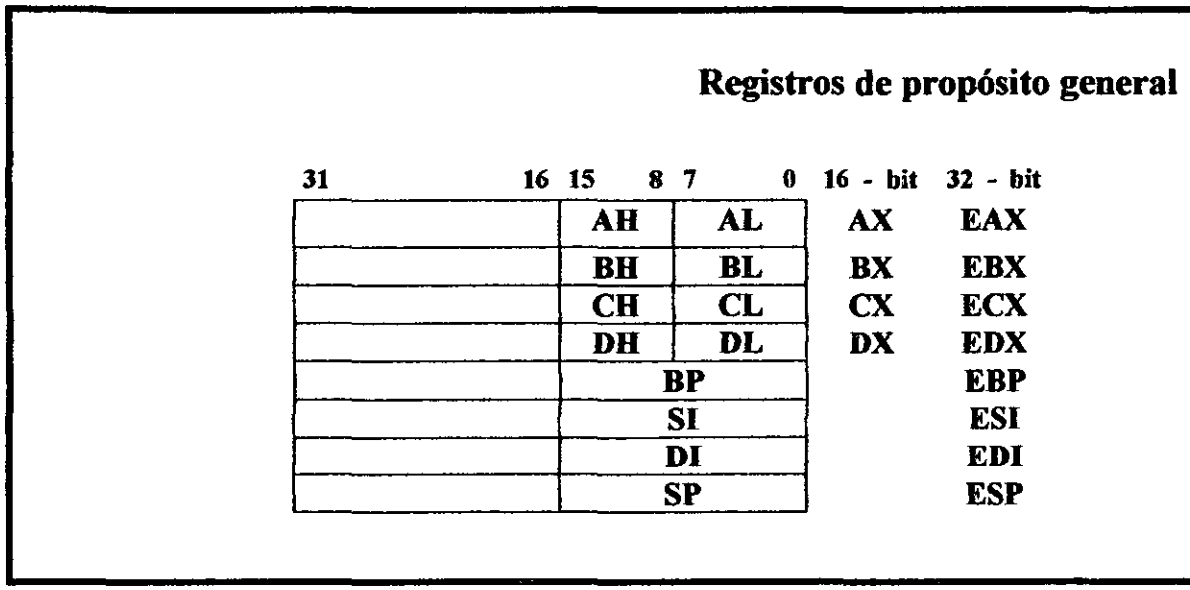
1. *EAX.* Acumulador para operando y resultado de datos.
2. *EBX.* Apuntador para el dato en el segmento DS.
3. *ECX.* Contador para operaciones *string y loop.*
4. *EDX.* Apuntador I/O.
5. *ESI.* Apuntador para el dato en el segmento apuntado por el registro DS; apuntador fuente para operaciones string.
6. *EDI.* Apuntador para dato (o destino) en el segmento apuntado por el registro ES; apuntador destino para operaciones *string.*
7. *ESP.* Apuntador de pila (En el segmento SS).
8. *EBP.* Apuntador da dato en la pila (En el segmento SS)

Aunque todos estos registros están disponibles para almacén general de operandos , resultados y apuntadores, se debe tener cuidado al usarse cuando se está haciendo referencia al registro ESP. El registro ESP contiene el apuntador de pila y como regla general no debe ser usado para ningún otro propósito.

En la figura 2-3 se muestran los registros de propósito general fundados por los procesadores Intel 8086 y 286 a los cuales se le puede hacer referencia

con los nombres AX, BX, CX, DX, BP, SP, SI, y DI. Cada uno de los dos bytes bajos de los registros EAX, EBX, ECX, y EDX pueden ser referenciados por los nombre AH, BH, CH, y DH (Byte alto) y AL, BL, CL, y DL (Byte bajo).

Figura 2-3 Nombres alternativos de los registros de propósito general[3]



2.5.2 Registros de segmento

Los registros de segmento (CS, DS, SS, ES, FS y GS) contienen 16 bits del segmento selector. Un segmento selector es un apuntador especial que identifica un segmento en la memoria. Para accesar a un segmento en particular en memoria, el registro selector debe estar presente en el registro del segmento apropiado.

Cada uno de los registros de segmentos está asociado con uno de estos tres tipos de almacenamiento: código, datos o pila.

1. *CS Segmento de código.* No debe ser cargado explícitamente por una aplicación del programa.
2. *DS, ES, FS, y GS Segmento de datos.* La disponibilidad de los cuatro segmentos de datos permite eficiencia y seguridad para accesar a los diferentes tipos de estructura de datos.
3. *SS Segmento de pila.* El procedimiento *stack* (Pila) está almacenado por el programa, tarea o manejador que actualmente está siendo ejecutado.

Los registros CS, DS, SS y ES, son los mismos que los fundados en los procesadores Intel 8086 e Intel 286; los registros FS y GS fueron introducidos en la AI con la familia de procesadores Intel 386.

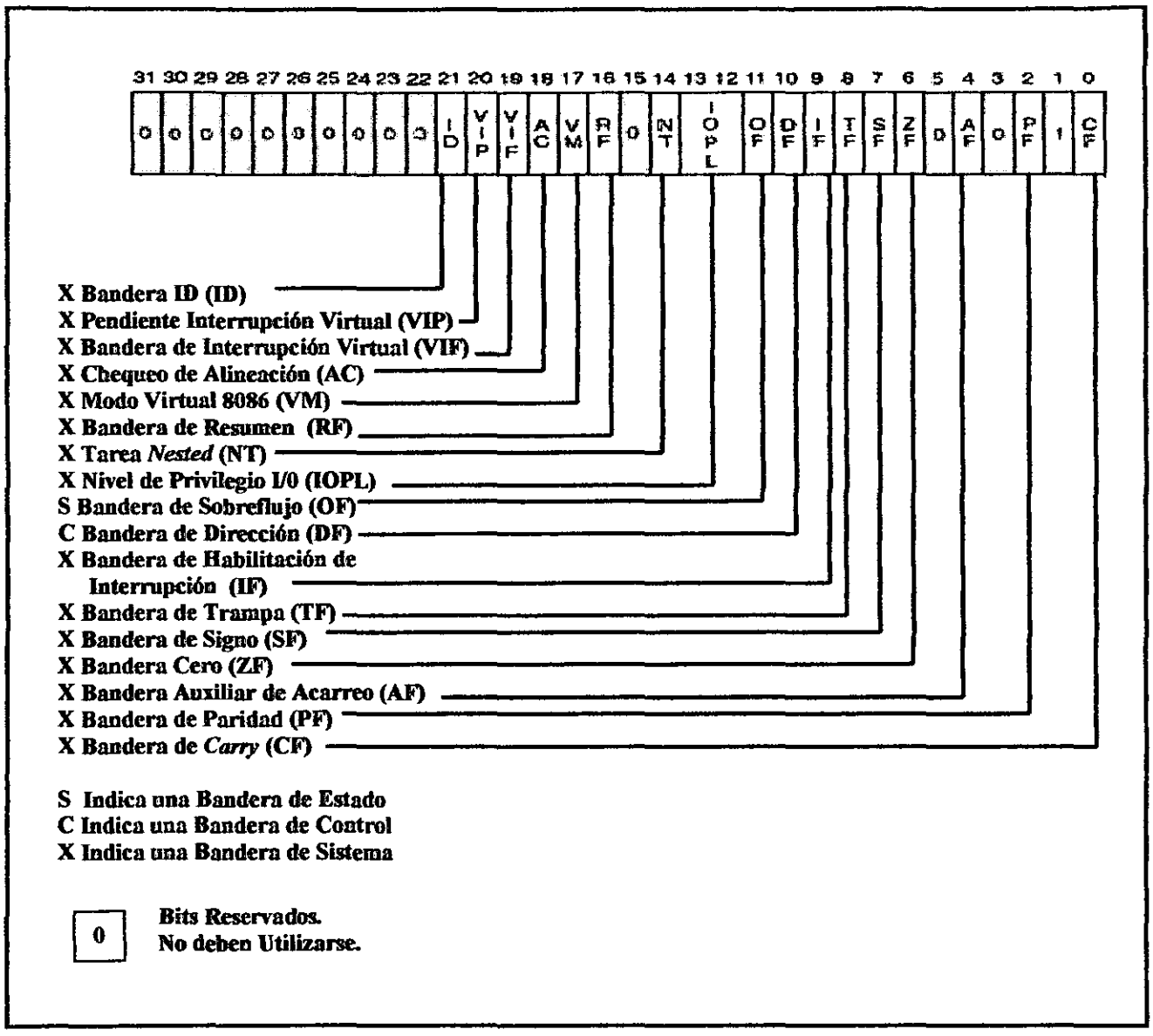
2.5.3 Registro de banderas

Los 32 bits del registro de banderas contienen un grupo de estado de banderas, control de banderas y sistemas de banderas.

En la siguiente figura 2-4 se muestra el estado de las banderas, el cual es de 00000002H. Los bits 1,3,5,15 y 22 al 31 de este registro están reservados. El software no debe usar o depender de ninguno de estos bits.

Algunas de las banderas en el registro de banderas puede ser modificado directamente, usando instrucciones de propósito especial.

Figura 2-4 Estado de las banderas[4]



2.5.3.1 Estado de banderas

Las banderas de estado (bits 0,2,4,6,7 y 11) del registro de banderas indican el resultado de instrucciones aritméticas, tal como las instrucciones ADD, SUB, MUL y DIV. Las funciones de las banderas de estado son las siguientes:

- *CF (bit 0) Bandera de carry.* Se activa si una operación aritmética genera un acarreo o un préstamo del bit más significativo del resultado. Esta bandera indica una condición de sobreflujo para la aritmética de enteros sin signo, de otra manera la bandera se limpia.
- *AF (bit 4) Bandera de ajuste.* Se activa si una operación aritmética genera un acarreo o un préstamo del bit 3 del resultado; de otra manera limpia. Esta bandera es usada en la aritmética de decimal codificado a binario (BCD).
- *PF (bit 2) Bandera de paridad.* Se activa si el bit menor significativo del resultado contiene un par de números de un bit; de otra manera limpia.
- *ZF (bit 6) Bandera cero.* Se activa si el resultado es cero; de otra manera limpia.
- *SF (bit 7) Bandera de signo.* Se activa igual para el bit más significativo del resultado, el cual es un entero del bit con signo o sin signo (0 indica un valor positivo y 1 indica un valor negativo).
- *OF (bit 11) Bandera de sobreflujo.* Se activa si el resultado de un entero es un número positivo largo o un número positivo corto para ajustarlo a el operando de destino; de otra manera limpia. Esta bandera indica una condición de sobreflujo para el entero sin signo (complemento a dos).

De estos estados de bandera, solamente la bandera CF pude ser modificada directamente usando las instrucciones STC, CLC y CAC. Además el bit de instrucciones CBT, BTS, BTR y BTC.

El estado de la bandera permite una sencilla aritmética de operación para producir resultados para tres diferentes tipos de datos: Enteros sin signo, enteros con signo y enteros BCD. Si el resultado de una operación aritmética es tratado como un entero sin signo, la bandera CF indica una condición fuera de rango (acarreo o préstamo); si se trata de un entero con signo (complemento a dos), la bandera OF indica un acarreo o préstamo, y si se trata como dígito BCD, la AF indica un acarreo o préstamo. La bandera FS indica el entero con signo o sin signo; la bandera ZF indica cualquier entero cero con signo o sin signo.

2.5.3.2 Bandera DF

La bandera de dirección (DF, localizada en el bit 10 del registro de banderas) controla las instrucciones string (MOVS, CMPS, SCAS, LODS Y STODS). Cuando se activa la bandera DF las instrucciones *string* se autodecrementan (los procesos string pasan de un direccionamiento alto a un direccionamiento bajo). Cuando la bandera DF se limpia, las instrucciones string se autodecrementan (los procesos string pasan de un direccionamiento bajo a un direccionamiento alto).

Las instrucciones STD y CLD activan y limpian la bandera DF, respectivamente.

2.5.3.3 Sistema de banderas y archivo IOPL

El sistema de banderas y archivo IOPL en el registro de banderas controla el sistema de operación o ejecución de instrucciones. Estos no pueden ser modificados por programas de aplicación. Las funciones de estos sistemas de banderas son las siguientes:

- *IF (bit 9) Bandera Habilitador de Interrupción:* Controla la respuesta del procesador para peticiones de interrupción mascarable. Se activa para responder a las interrupciones mascarables; limpia para inhibir las interrupciones mascarables.
- *TF (bit 8) Bandera trampa:* Se activa para habilitar el modo single-step.
- *IOPL (bits 12,13) Archivo del nivel de privilegio I/O*: Indica el nivel de privilegio I/O del actual programa o tarea que esta corriendo. El nivel de privilegio actual (CPL) del programa o tarea que actualmente esta corriendo es menor o igual que el nivel de privilegio I/O al accesar al espacio de dirección I/O. Este archivo únicamente puede ser modificado por las instrucciones POPF e IRET cuando operan en un CPL de cero.
- *NT (bit 14) Bandera de Tareas Anidadas:* Controla el encadenamiento de interrupciones y llamadas de tareas. Se activa cuando la actual tarea está vinculada a la tarea previamente ejecutada; se limpia cuando la actual tarea no está vinculada a la otra tarea.
- *RF (bit 16) Bandera de Resumen:* Controla las respuestas del procesador para las excepciones del DEBUG.
- *VM (bit 17) Bandera de Modo Virtual 8086:* Se activa para habilitar el modo virtual 8086; se limpia para regresar al modo de protección.

- *AC (bit 18) Bandera de Control de Alineación:* Se activa esta bandera y el bit AM en el registro CRO para habilitar el control de alineación de la memoria de referencia; se limpia la bandera AC y/o el bit AM para deshabilitar el control de alineación.
- *VIF (bit 19) Bandera de Interrupción Virtual:* Imagen virtual de la bandera IF. Usada en conjunción con la bandera VIP. (para usar esta bandera y la VIP, la extensión del modo virtual son habilitados para activar la bandera VME en el registro de control CR4).
- *VIP (bit 20) Bandera Pendiente de la Interrupción Virtual:* Se activa para indicar interrupciones pendientes; o limpia cuando las interrupciones no están pendientes (el software activa y desactiva esta bandera; el procesador únicamente lee esto). Es usada en conjunción con la bandera VIP.
- *ID (bit 21) Bandera de Identificación:* La habilidad de un programa para activar o limpiar esta bandera indica soporte para la instrucción CPVID.

2.6 Apuntador de instrucción

La instrucción pointer (EIP) contiene registros del offset en el código del segmento actual para la siguiente instrucción a ser ejecutada. Este es un adelanto de una instrucción limite para la siguiente en código de línea recta o es movida hacia delante o hacia atrás , por un número de instrucciones cuando se están ejecutando instrucciones COMP, Jcc, CALL, RET y IRET.

El registro EIP no puede ser accesado directamente por el software, este es controlado implícitamente por interrupciones de control de transferencia (así como Jmp, Jcc, CALL y RET), interrupciones y excepciones. El único

objetivo de leer el registro EIP es para ejecutar una instrucción CALL y entonces leer el valor del retorno de la instrucción pointer del procedimiento stack. El registro EIP no puede ser cargado indirectamente para modificar el valor de un retorno de la instrucción pointer en el procedimiento stack y ejecución de una instrucción de retorno (RET o IRET).

2.7 Atributos para el tamaño de operandos y dirección

Cuando el procesador se está ejecutando en modo de protección, cada segmento de código tiene un atributo *default* para el tamaño del operando y tamaño de dirección. Estos atributos seleccionados con la bandera D (tamaño default) en el segmento descriptor por el código del segmento.

Cuando la bandera está activada, el atributo del tamaño de operando y dirección es de 32 bits. Cuando la bandera se limpia, el tamaño del atributo es de 16 bits.

En la tabla 2-1 se muestra el tamaño efectivo del operando y la dirección (cuando se está ejecutando en modo de protección) dependiendo de la activación de la bandera D/B y el tamaño de operandos y dirección; solamente se aplica cuando la instrucción está sujeta a alguna condición.

Tabla 2-1 Tamaño efectivo del operando y dirección[5]

Bandera D en el Descriptor del Segmento de Código	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
Prefijo 66H en el tamaño de operando	**N**	**N**	**S**	**S**	**N**	**N**	**S**	**S**
Prefijo 67H en el tamaño de la dirección	**N**	**S**	**N**	**S**	**N**	**S**	**N**	**S**
Tamaño efectivo del operando	**16**	**16**	**32**	**32**	**32**	**32**	**16**	**16**
Tamaño efectivo de la dirección	**16**	**32**	**16**	**32**	**32**	**16**	**32**	**16**

[1] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 2-10

[2] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 3-6

[3] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 3-7

[4] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 3-11

[5] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 3-15

CAPITULO III
SET DE INSTRUCCIÓNES

Objetivo.

Describir el conjunto de instrucciones con las que cuenta el Procesador Pentium III.

Introducción

En este capítulo se describirán cada una de las instrucciones con las que cuenta el procesador Pentium III, las cuales nos permitirán realizar una serie de programas en ensamblador.

3.1. Set de instrucciones

A continuación se listan todas las instrucciones de la Arquitectura Intel, las cuales están divididas en cinco grandes grupos:

- Instrucciones enteras.
- Instrucciones de tecnología MMX.
- Instrucciones de punto flotante.
- Instrucciones del sistema.
- Instrucciones con extensión SIMD.

Para cada instrucción se da el mnemónico y su descripción.

3.1.1 Instrucciones enteras

Las instrucciones enteras realizan operaciones de aritmética, lógicas y control de operaciones del programa que comúnmente utilizamos para realizar aplicaciones y software del sistema para correr sobre un procesador de

Arquitectura Intel. En los siguientes apartados, las instrucciones enteras se dividen en los siguientes subgrupos de instrucciones:

- Instrucciones de movimiento de datos.
- Instrucciones de aritmética binaria.
- Instrucciones de aritmética decimal.
- Instrucciones lógicas.
- Instrucciones de cambio y traslado.
- Instrucciones bit y byte.
- Instrucciones de control de transferencia.
- Instrucciones cadena.
- Instrucciones para el control de banderas.
- Instrucciones diversas.

3.1.1.1 Instrucciones de movimiento de datos

Las instrucciones de movimiento mueven bytes, palabras, palabras dobles o cuádruples entre la memoria y los registros del procesador y entre los registros. Estas instrucciones están divididas en 4 grupos:

- Movimiento de datos de propósito general.
- Intercambio.
- Manipulación de la pila .
- Conversión.

A continuación se listan todas éstas instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
MOV	Movimiento.
CMOVE/CMOVZ	Movimiento condicional si es igual/Movimiento condicional si es cero.

INSTRUCCIÓN	DESCRIPCIÓN
CMOVA/CMOVNBE	Movimiento condicional si esta por encima / Movimiento condicional si no esta por abajo.
CMOVB/CMOVNAE	Movimiento condicional si esta por abajo /Movimiento condicional si no esta por abajo o es igual.
CMOVBE/CMOVNA	Movimiento condicional si esta por abajo o es igual /Movimiento condicional si no esta por encima.
CMOVG/CMOVNLE	Movimiento condicional si es el más grande /Movimiento condicional si es menor o igual.
CMOVL/CMOVNGE	Movimiento condicional si es menor /Movimiento condicional si no es el más grande o igual.
CMOVLE/CMOVNG	Movimiento condicional si es menor o igual / Movimiento condicional si no es el más grande.
CMOVC	Movimiento condicional si acarreo.
CMOVNC	Movimiento condicional si no acarreo.
CMOVO	Movimiento condicional si sobreflujo.
CMOVNO	Movimiento condicional si no sobreflujo.
CMOVS	Movimiento condicional si signo (negativo)
CMOVNS	Movimiento condicional si no signo (no negativo)
CMOVP/CMOVPE	Movimiento condicional si paridad / Movimiento condicional si paridad impar.
XCHG	Intercambio
BSWAP	Intercambio de byte
XADD	Intercambio y suma.
CMPXCHG	Compara e intercambia.

INSTRUCCIÓN	DESCRIPCIÓN
CMPXCHG8B	Compara e intercambia 8 bytes.
PUSH	Meter sobre la pila.
POP	Sacar de la pila.
PUSHA/PUSHAD	Meter los registros de propósito general de la pila.
POPA/POPAD	Sacar los registros de propósito general de la pila.
IN	Leer del puerto.
OUT	Escribir sobre el puerto.
CWD/CDQ	Convertir una palabra a palabra doble / Convertir una palabra doble a palabra cuádruple.
MOVSX	Mueve y extiende signo.
MOVZX	Mueve y extiende a cero.

3.1.1.2 Instrucciones de aritmética binaria

Las instrucciones de aritmética binaria operan en datos numéricos de 8, 16 y 32 bits codificados como enteros binarios con signo o sin signo. Incluye operaciones como la suma, la resta, la multiplicación y la división, así como el incremento, el decremento, la comparación y el cambio de signo (negación). Las instrucciones de aritmética binaria pueden ser utilizadas en algoritmos que operan en valores decimales (BCD). A continuación se muestran este conjunto de instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
ADD	Suma de enteros.
ADC	Suma con acarreo.
SUB	Substracción.

INSTRUCCIÓN	DESCRIPCIÓN
SBB	Substracción con préstamo.
IMUL	Multiplicación con signo.
MUL	Multiplicación sin signo.
IDIV	División con signo.
DIV	División sin signo.
INC	Incremento.
DEC	Decremento.
NEG.	Negación.
CMP	Comparación.

3.1.1.3 Instrucciones de aritmética decimal

Las instrucciones de aritmética decimal pueden ser desarrolladas en combinación con las instrucciones de aritmética binaria (ADD, SUB, MUL y DIV). Las instrucciones de aritmética llevan a cabo las siguientes operaciones:

- Ajustan los resultados de una operación de aritmética binaria previa al producir un resultado BCD valido.
- Ajusta los operandos de una operación de aritmética binaria subsecuente para que la operación pueda producir un resultado BCD valido.

Estas instrucciones operan solamente en valores de empacado y desempacado BCD. A continuación se listan estas instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
DAA	Ajuste decimal después de la suma.
DAS	Ajuste decimal después de la resta.
AAA	Ajuste ASCII después de la suma.

INSTRUCCIÓN	DESCRIPCIÓN
AAS	Ajuste ASCII después de la resta.
AAM	Ajuste ASCII después de la multiplicación.
AAD	Ajuste ASCCI antes de la división.

3.1.1.4 Instrucciones lógicas

Las instrucciones lógicas AND, OR, XOR (or exclusiva) y NOT representan las operaciones estándar boleanas. Las instrucciones AND, OR y XOR requieren dos operandos; la instrucción NOT opera en un operando simple. A continuación se listan estas instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
AND	And.
OR	Or.
XOR	Or exclusiva.
NOT	Not.

3.1.1.5 Cambio y traslado de instrucciones

Las instrucciones de cambio y traslado ordenan los bits dentro de un operando. Estas instrucciones están clasificadas en cambio, doble cambio y rotación. A continuación se listan éstas instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
SAR	Trasladar una instrucción aritmética a la derecha.
SHR	Trasladar una instrucción lógica a la derecha.
SAL/SHL	Trasladar una instrucción aritmética a la izquierda / Trasladar una instrucción lógica a la izquierda.
SHRD	Traslado doble a la derecha.
SHLD	Traslado doble a la izquierda.
ROR	Rotación a la derecha.
ROL	Rotación a la izquierda.
RCR	Rotación hacia la derecha a través del acarreo.
RCL	Rotación hacia la izquierda a través del acarreo.

3.1.1.6 Instrucciones bit y byte

Las instrucciones bit y byte operan sobre las cadenas bit o byte. Son divididas en cuatro grupos: instrucciones de prueba y modificación, instrucciones de rastreo de bit, byte sobre condición y prueba. A continuación se listan éstas instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
BT	Bit de prueba.
BTS	Bit de prueba y activación.
BTR	Bit de prueba y reset.
BTC	Bit de prueba y complemento.
BSF	Bit de búsqueda hacia adelante.

INSTRUCCIÓN	DESCRIPCIÓN
BSR	Bit de búsqueda hacia atrás.
SETE/SETZ	Activación del byte si es igual / Activación del byte si es cero.
SETNE/SETNZ	Activación de byte si no es igual / Activación del byte si no es cero.
SETA/SETNBE	Activación del byte si esta por encima / Activación del byte si no esta por abajo o es igual.
SETA/SETNB/SETNC	Activación del byte si esta por encima o es igual / Activación del byte si no esta por abajo / Activación de byte si no es acarreo.
SETBE/SETNA	Activación del byte si esta por abajo o es igual / Activación del byte si no esta por encima.
SETG/SETNLE	Activación del byte si es el más grande / Activación del byte si no es menor o igual.
SETL/SETNGE	Activación de byte si es menor / Activación del byte si no es el más grande o igual.
SETLE/SETNG	Activación del byte si es menor o igual / Activación del byte si no es el más grande.
SETS	Activación del byte si signo (negativo).
SETNS	Activación del byte si no signo (no negativo).
SETO	Activación del byte si sobre flujo.
SETNO	Activación del byte si no sobre flujo.
SETPE/SETP	Activación del byte si paridad impar / Activación del byte si paridad.

INSTRUCCIÓN	DESCRIPCIÓN
SETPO/SETNP	Activación del byte si paridad par / Activación del byte si no paridad.
TEST	Comparación lógica.

3.1.1.7 Instrucciones del control de transferencia

El procesador proporciona instrucciones del control de transferencias incondicional y condicional para dirigir el flujo en la ejecución del programa. Las transferencias condicionales son tomadas únicamente para especificar los estados de las banderas del registro EFLAGS. El control de transferencias incondicional son siempre ejecutadas. A continuación se listan estas instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
JMP	Salta
JE/JZ	Salta si es igual / Salta si es cero.
JNE/JNZ	Salta si no es igual / Salta si no es cero.
JA/JNBE	Salta si esta por encima / Salta si no esta por abajo o es igual.
JAE/JNB	Salta si esta por encima o es igual / Salta si no esta por abajo.
JB/JNAE	Salta si esta por abajo / Salta si no esta por encima o es igual.
JG/JNLE	Salta si es greater / Salta si no es menor o igual.
JGE/JNL	Salta si es greater o igual / Salta si no es menor.
JL/JNGE	Salta si es menor / Salta si no es greater o igual.

INSTRUCCIÓN	DESCRIPCIÓN
JLE/JNG	Salta si es menor o igual / Salta si no es greater.
JC	Salta si es acarreo.
JNC	Salta si no acarreo.
JO	Salta si sobre flujo.
JNO	Salta si no sobre flujo.
JS	Salta si signo (negativo).
JNS	Salta si no signo (no negativo).
JPO/JNP	Salta si paridad par / Salta si no paridad.
JPE/JP	Salta si paridad es impar / Salta si paridad.
JCXZ/JECXZ	Salta al registro CX cero / Salta al registro ECX cero.
LOOPZ/LOOPE	Repite con ECX y cero / Repite con ECX e igual.
LOOPNZ/LOOPNE	Repite con ECX y no cero / Repite con ECX y no igual.
CALL	Llamada a procedimiento.
RET	Retorno.
IRET	Retorno hacia la interrupción.
INT	Software de interrupción.
INTO	Interrupción en sobre flujo.
BOUND	Detección del valor fuera de rango.
LEAVE	Alto nivel del procedimiento de salida.

3.1.1.8 Instrucciones cadena

Las instrucciones MOVS (movimiento cadena), CMPS (comparación cadena), SCAS (rastreo cadena), LODS (carga cadena), y STOS (almacén cadena) permiten alargar la estructura de datos, tal como las cadenas de carácter alfanuméricos, son movidas y examinadas en memoria. Estas instrucciones operan sobre elementos individuales en una cadena, los cuales pueden ser un byte, palabra o doble palabra. Los elementos cadena. Los elementos cadena al ser operados son identificados por los registros ESI (fuente del elemento cadena) y EDI (destino del elemento cadena). Ambos registros contienen direcciones absolutas (offsets en un segmento) que apuntan al elemento cadena.

INSTRUCCIÓN	DESCRIPCIÓN
MOVS/MOVSB	Movimiento cadena / Movimiento byte cadena.
MOVS/MOVSW	Movimiento cadena / Movimiento de palabra cadena.
MOVS/MOVSD	Movimiento cadena / Movimiento de palabra doble cadena.
CMPS/CMPSB	Comparación cadena / Comparación byte cadena.
CMPS/CMPSW	Comparación cadena / Comparación de palabra cadena.
CMPS/CMPSD	Comparación cadena / Comparación de palabra doble cadena.
SCAS/SCASB	Búsqueda cadena / Búsqueda byte cadena.
SCAS/SCASW	Búsqueda cadena / Búsqueda de palabra cadena.

INSTRUCCIÓN	DESCRIPCIÓN
SCAS/SCASD	Búsqueda cadena / Búsqueda de palabra doble cadena.
LODS/LODSB	Carga cadena / Carga byte cadena.
LODS/LODSW	Carga cadena / Carga de palabra cadena.
LODS/LODSD	Carga cadena / Carga de palabra doble cadena.
STOS/STOSB	Almacén cadena / Almacén byte cadena.
STOS/STOSW	Almacén cadena / Almacén palabra cadena.
REPE/REPZ	Repite mientras es igual / Repite mientras es cero.
REPNE/REPNZ	Repite mientras no es igual / Repite mientras no es cero.
INS/INSW	Entrada de cadena al puerto / Entrada de palabra cadena al puerto.
INS/INSD	Entrada de cadena al puerto / Entrada de palabra doble cadena al puerto.
OUTS/OUTSB	Salida de cadena al puerto / Salida de byte cadena al puerto.
OUTS/OUTSW	Salida de cadena al puerto / Salida de palabra cadena al puerto.
OUTS/OUTSD	Salida de cadena al puerto / Salida de palabra doble cadena al puerto.

3.1.1.9 Instrucción para el control de banderas.

Las instrucciones para el control de las banderas dejan el estado de las banderas seleccionado en el registro EFLAGS al ser leído o modificado. A continuación se listan estas instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
STC	Activación de la bandera de acarreo.
CLC	Limpiar la bandera de acarreo.
CMC	Complemento a la bandera de acarreo.
CLD	Limpiar la dirección de la bandera.
STD	Activación de la bandera de dirección.
LAHF	Carga de banderas en el registro AH.
PUSHF/PUSHFD	Coloca las EFLAGS en la pila.
POPF/POPFD	Saca las EFLAGS de la pila.
STI	Activa la bandera de interrupción.

3.1.1.10 Instrucciones para los segmentos de registro

El procesador proporciona una variedad de instrucciones que direccionan los registros del procesador directamente. Estas instrucciones son utilizadas solamente cuando un sistema de operación o ejecución esta utilizando el segmento o el modelo de memoria en modo de dirección real.

INSTRUCION	DESCRIPCIÓN
LDS	Carga el apuntador usando el segmento DS.
LES	Carga el apuntador usando el segmento ES.
LFS	Carga el apuntador usando el segmento FS.

INSTRUCCIÓN	DESCRIPCIÓN
LGS	Carga el apuntador usando el segmento GS.
LSS	Carga el apuntador usando el segmento SS.

3.1.1.11 Instrucciones diversas

Las instrucciones para realizar diversas operaciones son de gran interés para los programadores de aplicaciones. A continuación se listan este conjunto de instrucciones.

INSTRUCCIÓN	DESCRIPCIÓN
LEA	Carga efectiva de la dirección.
NOP	No operación.
UB2	Instrucción indefinida.
XLAT/XLATB	Tabla de traslación con vista hacia arriba.
CPUID	Identificación del procesador.

3.1.2 Instrucciones de tecnología MMX

Las instrucciones MMX se ejecutan en los procesadores de la Arquitectura Intel. Estas instrucciones operan sobre empacado-byte, empacado-palabra, empacado-doble palabra y en operandos de palabra cuádruple. Estas instrucciones están divididas en los siguientes grupos.

- Instrucciones de transferencia de datos.
- Instrucciones de conversión MMX.
- Instrucciones de empacado de aritmética MMX.
- Instrucciones de comparación MMX.

- Instrucciones de lógica MMX.
- Instrucciones de traslación y rotación MMX.
- Administración del estado MMX.

3.1.2.1 Instrucciones de transferencias de datos MMX

INSTRUCCIÓN	DESCRIPCIÓN
MOVD	Movimiento de palabra doble.
MOVQ	Movimiento de palabra cuádruple.

3.1.2.2 Instrucciones de conversión MMX

INSTRUCCIÓN	DESCRIPCIÓN
PACKSSWB	Paquete de palabras en bytes con saturación signada.
PACKSSDW	Paquete de doble palabra en palabras con saturación sin nada.
PACKUSWB	Paquete de palabras en bytes con saturación sin signo.
PUNPCKHBW	Desempacado de alto arden de byte a partir de palabras.
PUNPCKHDQ	Desempacado de alto arden de palabra doble partir de palabra cuádruple.
PUNPCKLWD	Desempacado de bajo orden de palabras a partir de palabras dobles.
PUNPCKLDO	Desempacado de bajo orden de palabras dobles a partir de palabras cuádruples.

INSTRUCCIÓN	DESCRIPCIÓN
PUNPCKHWD	Desempacado de alto orden de palabras a partir de palabras dobles.

3.1.2.3 Instrucciones de empacado de aritmética MMX

INSTRUCCIÓN	DESCRIPCIÓN
PADDB	Suma de empacado de byte.
PADDW	Suma de empacado de palabras.
PADDD	Suma de empacado de palabras dobles.
PADDSB	Suma de empacado de byte con saturación.
PADDSW	Suma de empacado de palabras con saturación.
PADDUSB	Suma de empacado de byte sin signo con saturación.
PADDUSW	Suma de empacado de palabras sin signo con saturación.
PSUBB	Resta de empacado de bytes.
PSUBW	Resta de empacado de palabras.
PSUBD	Resta de empacado de palabras dobles.
PSUBSB	Resta de empacado de bytes con saturación.
PSUBSW	Resta de empacado de palabras con saturación.
PSUBUSW	Resta de empacado de palabras sin signo con saturación.
PMULHW	Multiplicación de empacado de palabras y almacén del resultado en alto.
PMULLW	Multiplicación de empacado de palabras y almacén del resultado en bajo.

INSTRUCCIÓN	DESCRIPCIÓN
PMADDWD	Multiplicación y suma de empacado de palabras.

3.1.2.4 Instrucciones de comparación MMX

INSTRUCCIÓN	DESCRIPCIÓN
PCMPEQB	Comparación de empacado de byte por igual.
PCMPEQW	Comparación de empacado de palabras por igual.
PCMPEQD	Comparación de empacado de palabras dobles por igual.
PCMPEQD	Comparación de empacado de dobles palabras por igual.
PCMPGTB	Comparación de empacado de bytes para el más grande que.
PCMPGTGW	Comparación de empacado de palabras para el más grande que.
PCMPGTD	Comparación de empacado de palabras dobles para el más grande que.

3.1.2.5 Instrucciones de lógica MMX

INSTRUCCIÓN	DESCRIPCIÓN
PAND	Lógica Bitwise and.
PANDN	Lógica Bitwise and not.
POR	Lógica Bitwise or.

INSTRUCCIÓN	DESCRIPCIÓN
PXOR	Lógica Bitwise or exclusiva.

3.1.2.6 Instrucciones de translación y rotación MMX

INSTRUCCIÓN	DESCRIPCIÓN
PSLLW	Traslado lógico de empacado de palabras hacia la izquierda.
PSLLD	Traslado lógico de empacado de palabras dobles hacia la izquierda.
PSLLQ	Traslado lógico de empacado de palabras cuádruples hacia la izquierda.
PSRLW	Traslado lógico de empacado de palabras hacia la derecha.
PSRLD	Traslado lógico de empacado de palabras dobles hacia la derecha.
PSRLQ	Traslado lógico de empacado de palabras cuádruples hacia la derecha.
PSRAD	Traslado aritmético de empacado de palabras dobles hacia la derecha.

3.1.2.7 Administración del estado MMX

INSTRUCCIÓN	DESCRIPCIÓN
EMMS	Estado vacío MMX.

3.1.3 Instrucciones de punto flotante

Estas instrucciones operan sobre punto flotante (real), enteros y operandos de decimal codificado a binario (BCD). Estas instrucciones están divididas en los siguientes grupos:

- Transferencia de datos.
- Aritmética básica.
- Comparación.
- Trascendental.
- Carga de constantes.
- Control de la unidad de punto flotante (FPU).

3.1.3.1 Transferencias de datos

INSTRUCCIÓN	DESCRIPCIÓN
FLD	Carga real.
FST	Almacén real.
FSTP	Almacén real y pop.
FILD	Carga de enteros.
FIST	Almacén de enteros.
FISTP	Almacén de enteros y pop.
FBLD	Carga BCD.
FBSTP	Almacén BCD y pop.
FCMOVE	Movimiento condicional de punto flotante si es igual.
FCMOVNE	Movimiento condicional de punto flotante si no es igual.

INSTRUCCIÓN	DESCRIPCIÓN
FCMOVB	Movimiento condicional de punto flotante si esta por abajo.
FCMOVBE	Movimiento condicional de punto flotante si esta por abajo o es igual.
FCMOVNB	Movimiento condicional de punto flotante si no esta por abajo.
FCMOVNBE	Movimiento condicional de punto flotante si no esta por abajo o es igual.
FVMOVU	Movimiento condicional de punto flotante si esta desordenado.
FVMOVNU	Movimiento condicional de punto flotante si no esta desordenado.

3.1.3.2 Aritmética básica

INSTRUCCIÓN	DESCRIPCIÓN
FADD	Suma real.
FADDP	Suma real y pop.
FIADD	Suma de enteros.
FSUB	Resta real.
FSUBP	Resta real y pop.
FSUBR	Resta real hacia atrás.
FSUBRP	Resta real hacia atrás y pop.
FISUBR	Resta de enteros hacia atrás.
FMUL	Multiplicación real.

INSTRUCCIÓN	DESCRIPCIÓN
FMULP	Multiplicación real y pop.
FIMUL	Multiplicación de enteros.
FDIV	División real.
FDIVP	División real y pop.
FIDIV	División de enteros.
FDIVR	División real hacia atrás.
FDIVRP	División real hacia atrás y pop.
FIDIVR	División de enteros hacia atrás.
FPREM	Renombrado parcial.
FPREMI	Renombrado parcial IEEE.
FABS	Valor absoluto.
FCHS	Cambio de signo.
FRNDINT	Redondeo para enteros.
FSCALE	Escala por poder de dos.
FSQRT	Raíz cuadrada.
FXTRACT	Extraer exponente y significado.

3.1.3.3 Comparación

INSTRUCCIÓN	DESCRIPCIÓN
FCOM	Comparación real.
FCOMP	Comparación real y pop.
FCOMPP	Comparación real y pop twice.
FUCOM	Comparación real desordenada.
FUCOMP	Comparación real desordenada y pop.

INSTRUCCIÓN	DESCRIPCIÓN
FUCOMPP	Comparación real desordenado y pop twice.
FICOM	Comparación de enteros.
FICOMP	Comparación de enteros y pop.
FCOMI	Comparación real y activación de las EFLAGS.
FUCOMI	Comparación real desordenado y activación de las EFLAGS.
FCOMIP	Comparación real, activación de las EFLAGS y pop.
FUCOMIP	Comparación real desordenada, activación de las EFLAGS y pop.
FTST	Prueba real.
FXAM	Examinación real.

3.1.3.4 Trascendental

INSTRUCCIÓN	DESCRIPCIÓN
FSIN	Seno.
FCOS	Coseno.
FPTAN	Tangente parcial.
FPATAN	Arcotangente parcial.
F2XM1	2^x-1
FYL2X	$y * \log_2 x$
FYL2XP1	$y * \log_2 (x+1)$

3.1.3.5 Carga de constantes

INSTRUCCIÓN	DESCRIPCIÓN
FLDI	Carga + 1.0
FLDZ	Carga + 0.0
FLDPI	Carga pi
FLDL2E	Carga $\log_2 e$
FLDLN2	Carga $\log_e 2$
FLDL2T	Carga $\log_2 10$
FLDLG2	Carga $\log_{10} 2$

3.1.3.6 Control de la unidad de punto flotante (FPU)

INSTRUCCIÓN	DESCRIPCIÓN
FINCSTP	Incremento del apuntador de pila en el registro FPU.
FDCSTP	Decremento del apuntador de pila en el registro FPU.
FINIT	Inicialización del FPU después de checar condiciones de error.
FININIT	Inicialización del FPU sin checar condiciones de error.
FCLEX	Limpia las banderas y excepciones de punto flotante después de chequeo por error.
FNCLEX	Limpia banderas y excepciones sin chequeo por error.
FSTCW	Control de almacén de palabra FPU después de checar condiciones de error.

INSTRUCCIÓN	DESCRIPCIÓN
FNSTCW	Control de almacén de palabra FPU sin checar condiciones de error.

3.1.4 Instrucciones del sistema.

Las siguientes instrucciones del sistema son utilizadas para el control de las funciones del procesador, las cuales se listan a continuación.

INSTRUCCIÓN	DESCRIPCIÓN
LGDT	Carga del registro en la tabla del descriptor global (GDT).
SGDT	Almacén del registro en la tabla del descriptor global (GDT).
LLDT	Carga del registro en la tabla del descriptor local (LDT).
LTR	Carga de la tarea del registro.
STR	Almacén de la tarea del registro.
LIDT	Carga del registro en la tabla del descriptor de interrupción (IDT).
SIDT	Almacén del registro en la tabla del descriptor de interrupción (IDT).
MOV	Carga y almacén de los registros de control.
LMSW	Carga de palabra en estado máquina.
SMSW	Almacén de palabra en estado máquina.
CLTS	Limpia la bandera de la tarea interrumpida.
ARPL	Ajusta el privilegio del nivel solicitado.

INSTRUCCIÓN	DESCRIPCIÓN
LAR	Carga de accesos hacia la derecha.
LSL	Carga el limite del segmento.
VERR	Verifica los segmentos para lectura.
VERW	Verifica los segmentos para escritura.
MOV	Carga y almacena los registros del debug.
INVD	Invalida la cache, no escribe después.
WBINVD	Invalida la cache, sin escribir después.
INVLPG	Invalida la entrada TLB.
LOCK (prefix)	Cierra el bus.
HLT	Parada del procesador.
RSM	Regresa al modo de administración del sistema.
RDMSR	Lee el registro de modelo especifico.
WRMSR	Escribe el registro de modelo especifico
RDPMC	Lee el funcionamiento monitoreando los contadores.
SYSENTER	Llamada del sistema, transferencia para un modo de protección Kernel en CPL=0
SYSEXIT	Llamada del sistema, transferencia para un modo de protección kernel en CPL=3

3.1.5 Instrucciones con extensiones SIMD

Las instrucciones con extensiones SIMD son ejecutadas por los procesadores de Arquitectura Intel. Estas instrucciones operan sobre paquetes de precisión simple de operandos de punto flotante; las cuales están divididas en los siguientes grupos.

- Instrucciones de transferencia de datos con extensión SIMD.
- Instrucciones para la conversión con extensión SIMD.
- Instrucciones de empacado de aritmética con extensión SIMD.
- Instrucciones de comparación con extensión SIMD.
- Instrucciones de lógica con extensión SIMD.
- Instrucciones de SHUFLE de datos con extensión SIMD.
- Instrucciones adicionales entero-SIMD con extensión SIMD.
- Instrucciones del control del habilitador de cache con extensión SIMD.
- Instrucciones del control del estado con extensión SIMD.

3.1.5.1 Instrucciones de transferencia de datos con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
MOVAPS	Movimiento empaquetado alineado de precisión simple de punto flotante.
MOVUPS	Movimiento empaquetado desalentado de precisión simple de punto flotante.
MOVHPS	Movimiento de alto empaquetado desalineado de precisión simple de punto flotante.
MOVLPS	Movimiento desalineado de bajo empaquetado de precisión simple de punto flotante.
MOVLHPS	Movimiento alineado de bajo empaquetado de precisión simple de punto flotante para un alto empaquetado de precisión simple de punto flotante.
MOVMSKPS	Movimiento enmascarado de precisión simple de punto flotante.

INSTRUCCIÓN	DESCRIPCIÓN
MOVSS	Movimiento escalar de precisión simple de punto flotante.

3.1.5.2 Instrucciones para la conversión con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
CVTPI2PS	Conversión de empacado de entero de 32-bit a empacado de precisión simple de punto flotante.
CVTSI2SS	Conversión escalar de entero de 32-bit a escalar de precisión simple de punto flotante.
CVTPS2PI	Conversión de empacado de precisión simple de punto flotante a empacado de entero de 32-bit.
CVTSS2SI	Conversión escalar de precisión simple de punto flotante a entero de 32-bit.
CVTTSS2SI	Conversión truncada escalar de precisión simple de punto flotante a entero escalar de 32-bit.

3.1.5.3 Instrucciones de empacado de aritmética con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
ADDPS	Suma empacada de precisión simple de punto flotante.
SUBPS	Resta empacada de precisión simple de punto flotante.
ADDSS	Suma escalar empacada de precisión simple de punto flotante.
SUBSS	Resta escalar empacada de precisión simple de punto flotante.
MULPS	Multiplicación empacada de precisión simple de punto flotante.
MULSS	Multiplicación escalar empacada de precisión simple de punto flotante.
DIVPS	División empacada de precisión simple de punto flotante.
SQRTPS	Raíz cuadrada empacada de precisión simple de punto flotante.
SQRTSS	Raíz cuadrada escalar de precisión simple de punto flotante.
MAXPS	Máximo empacado de precisión simple de punto flotante.
MAXSS	Máximo escalar de precisión simple de punto flotante.
MINPS	Mínimo empacado de precisión simple de punto flotante.

INSTRUCCIÓN	DESCRIPCIÓN
MINSS	Mínimo escalar de precisión simple de punto flotante.

3.1.5.4 Instrucciones de comparación con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
CMPPS	Comparación empacada de precisión simple de punto flotante.
CMPSS	Comparación escalar de precisión simple de punto flotante.
COMISS	Comparación escalar de precisión simple de punto flotante ordenada y activación de EFLAGS.
UCOMISS	Comparación desordenada escalar de precisión simple de punto flotante ordenado y activación de EFLAGS.

3.1.5.5 Instrucciones de lógica con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
ANDPS	Lógica de empacado Bit-wise AND por precisión simple punto flotante.
ANDNPS	Lógica de empacado Bit-wise AND NOT por precisión simple punto flotante.
ORPS	Lógica de empacado Bit-wise OR por precisión simple punto flotante.

INSTRUCCIÓN	DESCRIPCIÓN
XORPS	Lógica de empacado Bit-wise XOR por precisión simple punto flotante.

3.1.5.6 Instrucciones de shufle de datos con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
SHUFPS	Empacado shufle precisión simple punto flotante.
UNPCKHPS	Desempacado de alto empacado de precisión simple de punto flotante.
UNPCKLPS	Desempacado de bajo empacado de precisión simple de punto flotante.

3.1.5.7 Instrucciones adicionales entero-SIMD con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
PAVGB/PAVGW	Promedio sin signo fuente sub-operandos, sin pérdida de precisión.
PEXTRW	Extraer palabra de 16-bit del registro MMX.
PINSRW	Insertar palabra de 16-bit al registro MMX.
PMAXUB/PMAXSW	Máximo de empacado sin signo de enteros bytes o palabras enteras con signo.
PMINUB/PMINSW	Mínimo de empacado sin signo de enteros bytes o palabras enteras con signo.

INSTRUCCIÓN	DESCRIPCIÓN
PMOVMSKB	Movimiento de máscara de byte del registro MMX.
PMULHUW	Multiplicación de palabra entera de alto empacado sin signo en el registro MMX.
PSADBW	Suma de diferencias absolutas.
PSHUFW	Shufle empacado de palabra entera en el registro MMX.

3.1.5.8 Instrucciones del control del habilitador de cache con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
MASKMOVQ	Almacén de la mascara byte no temporal del entero empacado en un registro MMX.
MOVNTQ	Almacén no temporal del entero empacado en un registro MMX.
MOVNTPS	Almacén no temporal del empacado de punto flotante precisión simple.
PREFETCH	Carga 32 o más números de bytes.
SFENCE	Almacén de cerca.

3.1.5.9 Instrucciones de administración del estado con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
LDMXCSR	Carga el control de punto flotante SIMD y el estado del registro.
STMXCSR	Almacena el control de punto flotante SIMD y el estado del registro.

INSTRUCCIÓN	DESCRIPCIÓN
FXSAVE	Salva el estado MMX, punto flotante y punto flotante SIMD a memoria.
FXRSTOR	Carga el estado FP, MMX y punto flotante SIMD desde la memoria.

CAPITULO IV
TIPOS DE DATOS Y MODOS DE DIRECCIONAMIENTO

Objetivo

Explicar los tipos de datos y modos de direccionamiento con los que cuenta el procesador Pentium III.

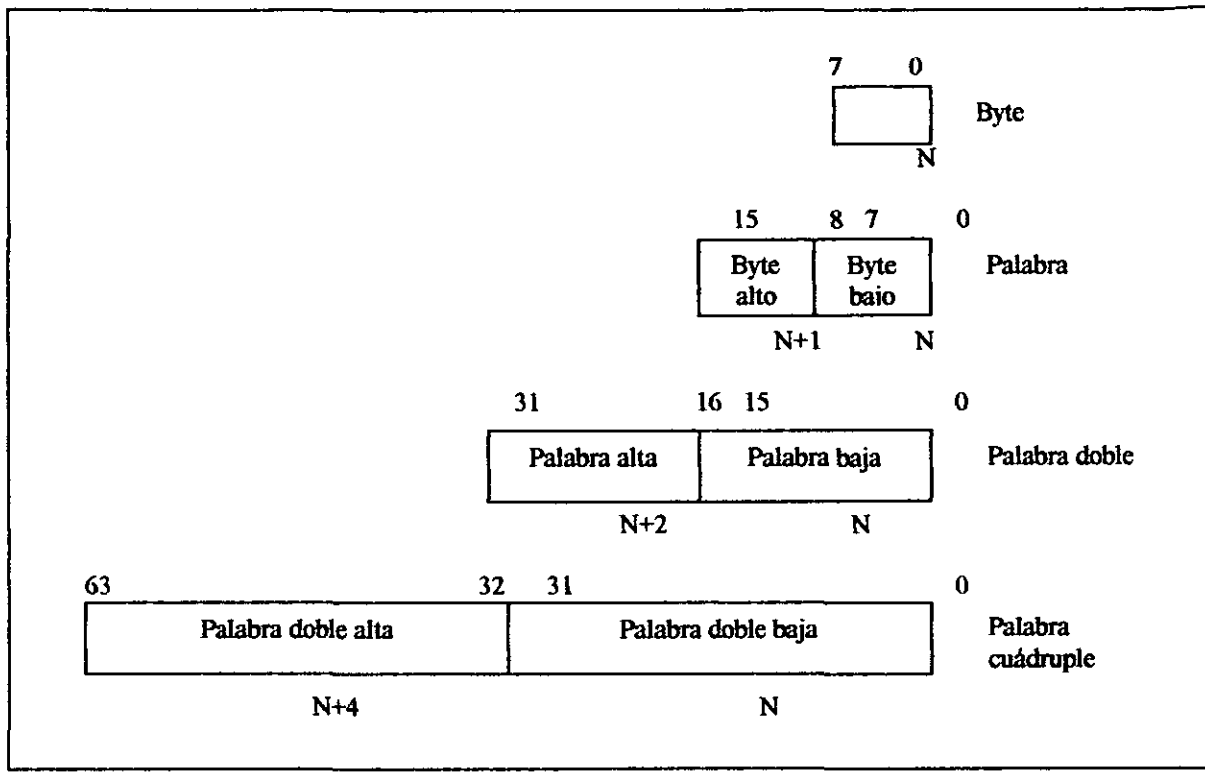
Introducción

En el siguiente capitulo se describen los tipos de datos y los modos de direccionamiento disponibles para programar procesadores con la Arquitectura Intel (IA).

4.1 Tipo de datos fundamentales

Los datos fundamentales de la Arquitectura Intel (IA) son de 4 tipos: bytes, palabras, palabras dobles y palabras cuádruples. En la figura 4.1 se muestran un esquema de estos tipos de datos. Un byte es de 8 bits, una palabra es de 2 bytes (16 bits), una palabra doble es de 4 bytes (32 bits) y una palabra cuádruple es de 8 bytes (64 bits).

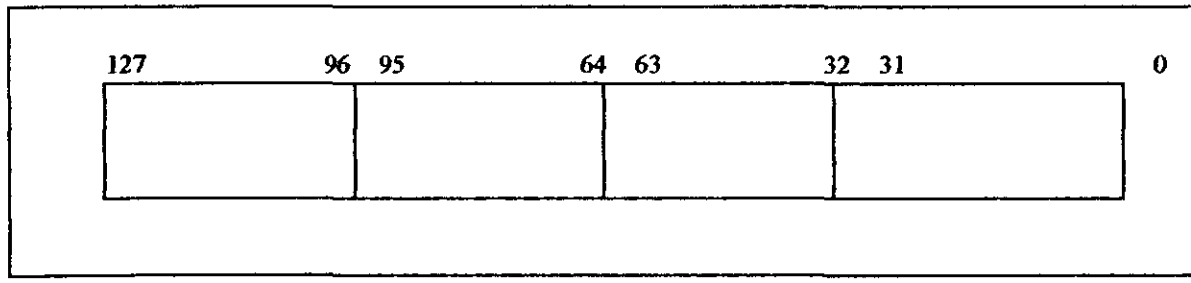
Figura 4-1 Tipos de datos fundamentales[1]



El procesador Pentium III introduce un nuevo tipo de dato: empacado de 128-bits. Este empacado es de precisión simple (32 bits) en números de punto flotante. Estos valores son los operandos para las operaciones SIMD en punto flotante.

En la figura 4-2 se muestra el orden de los bytes de cada uno de los tipos de datos fundamentales cuando se referencian como operandos en memoria.

Figura 4-2 Dato SIMD punto flotante[2]

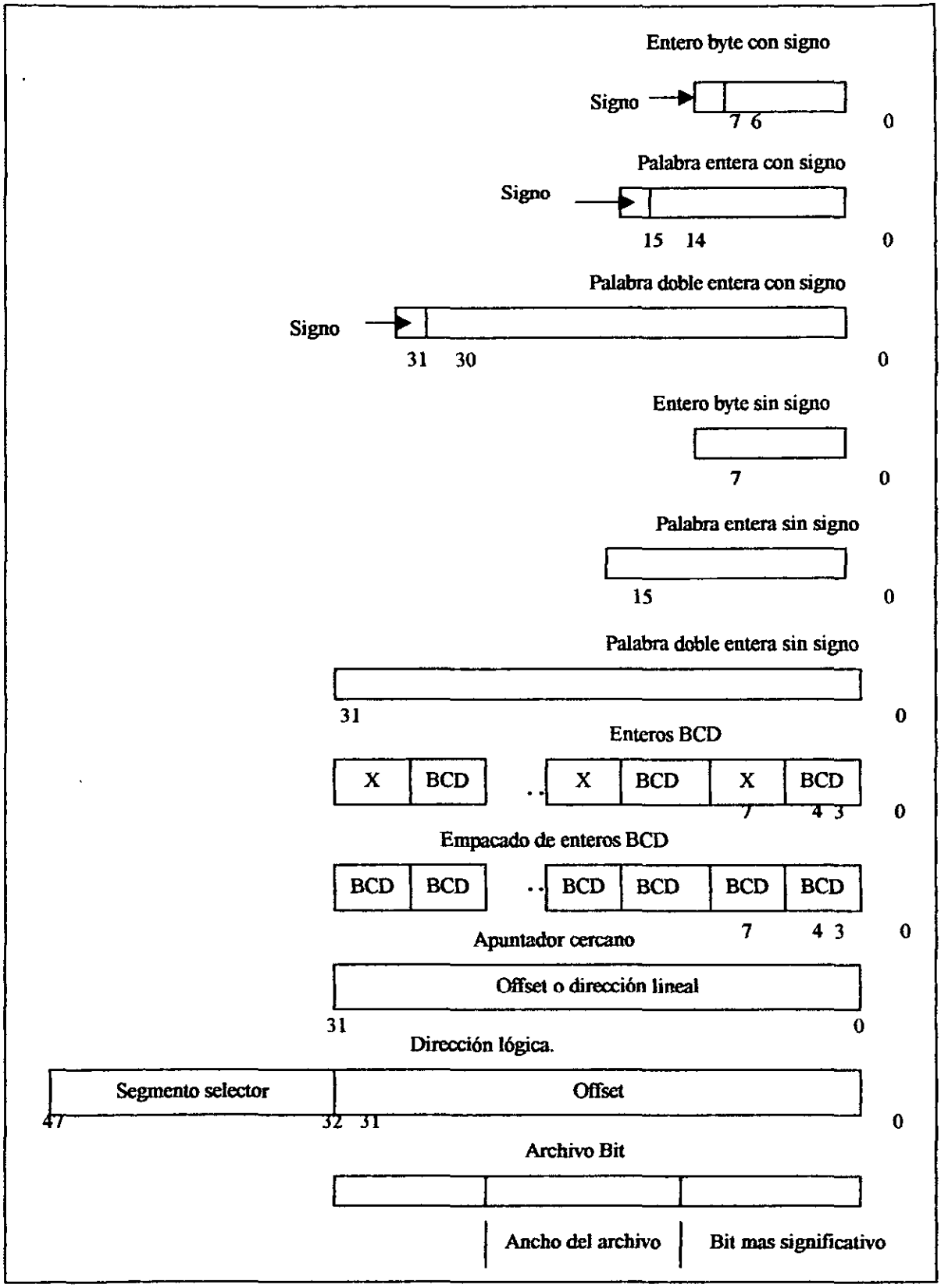


4.2 Tipo de datos numérico, apuntador, archivo bit y cadena

Los datos bytes, palabras, y palabras dobles son los datos fundamentales de la IA, algunas instrucciones soportan interpretaciones adicionales de estos tipos de datos para permitir operaciones que tienen que ser realizadas sobre tipos de datos numéricos (enteros con signo y sin signo, y enteros BCD). Algunas instrucciones reconocen y operan sobre datos adicionales de tipo apuntador, archivo bit y cadena. En la figura 4-3 se muestran estos tipos de datos.

En las siguientes secciones se describirán más a detalle estos tipos de datos.

Figura 4-3 Tipo de datos numérico, apuntador y archivo bit[3]



4.2.1 Enteros

Los enteros son números binarios con signo contenidos en un byte, en una palabra o en o en una palabra doble. Todas las operaciones adoptan una representación de complemento a dos. El bit con signo es localizado en el bit 7 en un entero byte, en el bit 15 en una palabra entera y en el bit 31 en una palabra doble entera. El bit con signo es activado por enteros negativos y desactivada o limpiada por enteros positivos y cero. El rango de valores va de -128 a +127 para un entero bit, de -32, 768 a +32,767 para una palabra entera, y de -2^{31} a $+2^{31}-1$ para una palabra doble entera.

4.2.2 Enteros sin signo

Los enteros sin signo son números enteros contenidos en un byte, en una palabra o en una palabra doble. Los valores para los enteros sin signo van de 0 a 255 para un entero byte sin signo, de 0 a 65, 535 para una palabra entera sin signo, y de 0 a 2^{32} -1 para una palabra doble sin signo. Los enteros in signo son algunas veces referenciados para los ordinales.

4.2.3 Enteros BCD

Los enteros de decimal codificado a binario (BCD) son enteros de 4-bit sin signo como valores validos en un rango de 0 a 9. Los enteros BCD pueden ser desempacados (un dígito BCD por byte). El valor de un entero BCD desempacado es el valor binario del byte medio bajo (bits 0 por 3). El byte medio alto (bits 4 por 7) puede ser algún valor durante la suma y resta, pero debe ser 0 durante la multiplicación y división.

Los enteros empacados BCD permites dos dígitos BCD para ser contenidos en un byte. Aquí, el dígito en el byte medio alto es más significativo que el dígito del byte medio bajo.

4.2.4 Apuntador

Los apuntadores son direcciones de localizaciones en memoria. El procesador Pentium Pro reconoce dos tipos de apuntadores: apuntador cercano (32 bits) y un apuntador lejano (48 bits). Un apuntador cercano es un offset de 32 bits (algunas veces es llamado dirección efectiva) dentro de un segmento.

Los apuntadores cercanos son utilizados para todas las referencias de memoria en un modelo de memoria flat o para referenciar un modelo segmentado en donde la identidad del segmento al que se esta accesando se sobre entiende. Un apuntador lejano es una dirección lógica de 48 bits, la cual consiste en un segmento selector de 16 bits y un offset de 32 bits. Los apuntadores lejanos son utilizados para referenciar memorias en un modelo de memoria segmentada en donde la identidad del segmento al que se esta accesando debe ser explícitamente especificado.

4.2.5 Archivos bit

Un archivo bit es una secuencia continua de bits. Este puede estar en algún bit de posición de algún byte en memoria y puede contener más de 32 bits.

4.2.6 Datos cadena

Los datos cadena son secuencias de bits, bytes, palabras o palabra dobles. Una cadena byte puede estar en algún bit de posición de algún byte y puede

contener más de $2^{32}-1$ bits. Una cadena byte pude contener bytes, palabras o palabras dobles y tiene un rango de 0 a $2^{32}-1$ byte (4 gigabytes).

4.2.7 Datos de punto flotante

Las instrucciones de punto flotante de un procesador reconoce un set de datos de tipo real, entero y enteros BCD. Más adelante se hace una descripción más de tallada sobre los datos de punto flotante.

4.2.8 Datos de tecnología MMX

Los procesadores con Arquitectura Intel que implementan la tecnología MMX reconoce un set de empacado de 64 bits.

4.2.9 Datos con extensión SIMD

Los procesadores de Arquitectura Intel que implementan las extensiones SIMD reconocen un set de datos de 128 bits.

4.3 Direccionamiento de operandos

Una instrucción máquina en la Arquitectura Intel son acciones sobre cero o más operandos. Algunos operandos están explícitamente especificadas en una instrucción y otras estas implícitas en una instrucción. Un operando pude ser localizado en alguno de los siguientes lugares:

- La instrucción itself (un operando inmediato).
- Un registro.
- Una localización de memoria.
- Un puerto de entrada / salida.

4.3.1 Operandos inmediatos

Algunas instrucciones utilizan datos codificados en la instrucción itself como un operando fuente. Estos operandos son llamados operandos inmediatos (o simplemente inmediatos). Por ejemplo, la siguiente instrucción ADD agrega un valor inmediato de 14 para el contenido del registro EAX:

```
ADD EAX, 14
```

Todas las instrucciones aritméticas (excepto las instrucciones DIV e IDIV) permiten el operando fuente para ser un valor inmediato. El valor máximo permitido por un operando inmediato varia entre instrucciones, pero nunca pude ser más grande que el valor máximo de una palabra doble entera sin signo (2^{32}).

4.3.2 Operandos de registro

Los operandos fuente y destino pueden ser localizados en algunos de los siguientes registros, dependiendo de la instrucción que se este ejecutando:

- Registro de propósito general de 32 bits (EAX, EBX, ECX, EDX, EDI, ESP, o EBP).
- Registro de propósito general de 16 bits (AX, BX, CX, DX, SI, DI, SP, o BP).
- Registros de propósito general de 8 bits (AH, BH, CH, DH, AL, BL, CL, o DL).
- Registros de segmento (CS, DS, SS, ES, FS, y GS).
- Registro de banderas.

- Registros de sistema, tal como la tabla del registro descriptor global (GDTR) o la tabla del registro del descriptor de interrupción (IDTR).

Algunas instrucciones (tal como la instrucción DIV y MUL) utilizan operandos de palabra cuadrada contenidos en un par de registros de 32 bits. El par de registros esta representado con un colon separatingn them. Por ejemplo en el registro par EDX: EAX, EDX contiene los bits de orden alto y el EAX contiene los bits de orden bajo de un operando de palabra cuadrada.

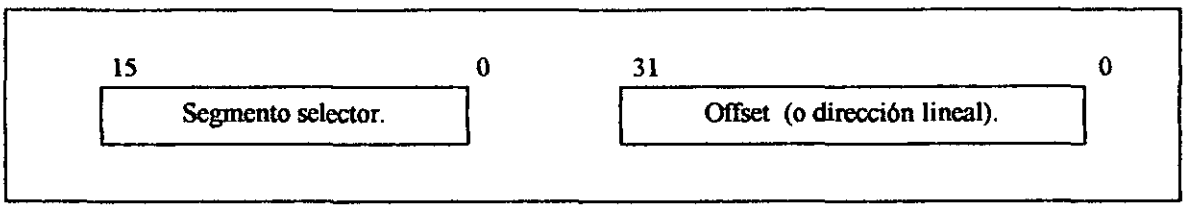
Varias instrucciones (tal como las instrucciones PUSHFD y POPFD) son provistas para carga y almacén del contenido del registro de banderas o para activar o limpiar banderas individuales en estos registros. Otras instrucciones (tal como la instrucción Jcc) utilizan el estado del estado banderas en el registro de banderas como condición de código por ramas u otra decisión en vías de formación de operaciones.

El procesador contiene una selección de registro de sistema que son utilizados para controlar la administración de memoria, interrupción y excepción, administración de tareas, administración del procesador, actividades debugging. Algunos de estos registros del sistema son accesados por un programa de aplicación , el sistema operativo, o para ejecutar una instrucción del sistema. Cuando se accesa a un registro del sistema con una instrucción del sistema, el registro es generalmente un operando sobre entendido de la instrucción.

4.3.3 Operandos de memoria

Los operandos fuente y destino en memoria están referenciados por medio de un segmento selector un offset, como se muestra en la figura 4–4. El segmento selector especifica el segmento que contiene el operando y el offset especifica la dirección lineal o efectiva del operando.

Figura 4-4. Dirección de operandos de memoria[4]



4.3.3.1 Especificación de un segmento selector

El segmento selector puede ser especificado ya sea implícitamente o explícitamente. El método más común de especificación de un segmento selector es para cargar este en un segmento de registro y entonces dejar al procesador seleccionar el registro implícitamente, dependiendo del tipo de operación que se este ejecutando. El procesador cambia automáticamente un segmento de acuerdo a las reglas que se muestran en la tabla 4-1.

Tabla 4-1 Reglas default en la selección de un registro[5]

Tipo de referencia	Registro usado.	Segmento usado.	Regla de selección default.
Instrucciones	CS	Segmento de código.	Todas las instrucciones fetches.
Pila	SS	Segmento de pila.	Todos los pushes y pops de la pila. Alguna referencia del memoria la cual utiliza el registro ESP o EBP como registro base.
Dato local.	DS	Segmento de dato.	Todas las referencias de datos excepto cuando la pila es relativa o el destino es cadena.
Destino de cadenas.	ES	Segmento de datos apuntado con el registro ES.	Destino de instrucciones cadena

Cuando se están cargando o almacenando datos desde la memoria, el segmento default DS puede ser sobrepasado (overridden) para dejar a otros segmentos que sean accesados. Dentro del ensamblador el segmento sobre pasado es generalmente manejado con un operador dos puntos " : " . por ejemplo, la instrucción MOV, mueve un valor del registro EAX en el segmento apuntado por el registro ES. El offset en el segmento esta contenido en el registro EBX.

MOV ES : [EBX], EAX;

A nivel de máquina un segmento sobrepasado esta especificado con un prefijo en el segmento override, el cual es colocado por un bit al comienzo de una instrucción. Los siguientes segmentos de selección default no pueden ser overridden:

- Las instrucciones fetches deben ser realizadas desde el segmento de código.
- Los destinos cadena en instrucciones cadena deben ser almacenados en el segmento de datos apuntado por el registro ES.
- Las operaciones push y pop deben referenciarse siempre en el segmento S.

Algunas instrucciones requieren un segmento selector para ser especificados explícitamente. En estos casos los 16 bits del segmento selector pueden ser localizados en una localidad de memoria o en un registro de 16 bits. Por ejemplo, la siguiente instrucción MOV mueve un segmento selector en el registro BX hacia el segmento del registro DS:

MOV DS, BX

Los segmentos selectores pueden también ser especificados explícitamente como parte de un apuntador lejano de 48 bits en memoria. Aquí, la primer palabra doble en memoria contiene el offset y la siguiente palabra contiene el segmento selector.

4.3.3.2 Especificación de un offset

El offset parte de una dirección de memoria que puede ser especificada ya sea directamente como un valor estático (llamada a un desplazamiento) o bien de una dirección de computación inventada de uno o más de los siguientes componentes:

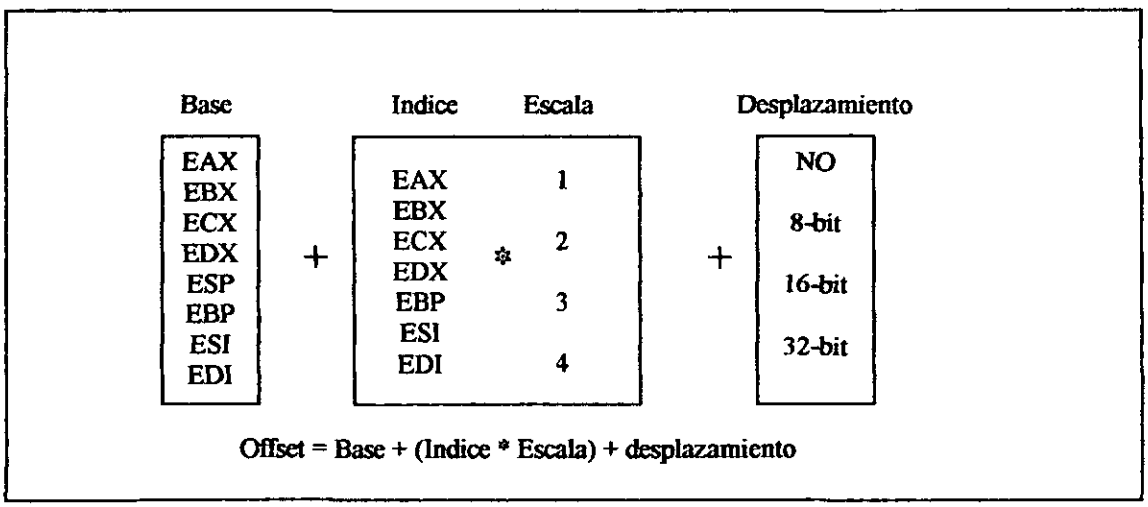
- Desplazamiento – en valores de 8, 16, o 32 bits.
- Base – valor en un registro de propósito general.

- Indice – valor en un registro de propósito genera.
- Factor de escala – un valor de 2, 4, u 8 que es multiplicado por un valor índice.

El offset que resulta al agregar a estos componentes es llamada "dirección efectiva".

Cada uno de estos componentes puede tener un valor positivo o negativo (complemento a dos), con excepción del factor de escala. En la figura 4-5 se muestran todos los posibles modos con que estos componentes pueden combinarse para crear una dirección efectiva en el segmento seleccionado.

Figura 4-5 Computación offset (dirección efectiva)[6]



Los usos de los registros de propósito general como componentes base o índice estas restringidos de la siguiente manera:

- El registro ESP no puede ser usado como un registro índice.

- Cuando el registro ESP o EBP son usados como la base, el segmento SS es el segmento default. En todos los demás casos el segmento DS es el segmento default.

Los componentes base índice y desplazamiento pueden ser utilizados en alguna combinación, y alguno de estos componentes pueden ser nulos. El factor escala puede ser utilizado solo cuando el índice también lo es. Cada una de las posibles combinaciones es utilizada para estructuras de datos, común ¡mente es usada por lenguajes de programación de alto nivel y lenguaje ensamblador los siguientes modos de direccionamiento sugieren usos de combinaciones comunes de componentes de dirección.

Desplazamiento

Un desplazamiento solo representa un offset directo para el operando. Porque el desplazamiento es codificado en la instrucción, esta forma de dirección es algunas veces llamada dirección absoluta o estática. Este es comúnmente utilizado para accesar a localidades estáticas de operandos escalares.

Base

Una base solo representa un offset indirecto del operando. El valor en el registro base puede cambiarse, este puede ser utilizado por un almacén dinámico de variables y estructuras de datos.

Base + desplazamiento

Un registro base y un desplazamiento pueden ser utilizados juntos para dos propósitos distintos:

- Como un índice en un arreglo cuando el tamaño del elemento no es 2, 4 u 8 bytes. El componente del desplazamiento codifica el offset estático en el comienzo del arreglo. El registro base tiene los resultados de un calculo para determinar el offset de un elemento especifico dentro del offset.
- Para accesar al archivo de un registro. El registro base tiene la dirección del comienzo de un registro, mientras el desplazamiento es un offset para el archivo.

Un importante caso especial de esta combinación es accesar por parámetros en un procedimiento para activar un registro. En el procedimiento que activa el registro esta la estructura de la pila creada por un procedimiento a entrado. Aquí, el registro EBP es la mejor elección para el registro base, porque este automáticamente selecciona el registro base. Esto es una compacta codificación para esta función común.

(Indice X escala) + desplazamiento

Este modo de dirección ofrece un eficiente camino para indexar en un arreglo estático cuando el tamaño del elemento es 2, 4, u 8 bytes.

El desplazamiento localiza el comienzo del arreglo, el registro índice el subscriptor del elemento deseado del arreglo, y el procesador

automáticamente convierte el subscriptor en un índice para aplicar el factor de escalación.

Base + índice + desplazamiento

Usando dos registros juntos, soporta ya sea un arreglo dimensional (el desplazamiento tiene la dirección del comienzo del arreglo) o una de varias instancias de un arreglo de registros (el desplazamiento es un offset para un archivo dentro del registro).

Base + (índice X escala) + desplazamiento

Utilizando todos los componentes juntos, nos permite un indexado eficiente de un arreglo dimensional cuando el tamaño de los elementos es de 2, 4, u 8 bytes .

4.3.3.3 Modos de direccionamiento en el ensamblador y compilador

A nivel de código de maquina la selección de combinación de desplazamiento, registro base, registro índice, y factor de escala es codificada en una instrucción. Todos los ensambladores permiten un programa para usar algunas de las combinaciones posibles para estos componentes de direccionamiento para direccionar operandos. Los compiladores de lenguaje de alto nivel (HLL) pueden seleccionar una combinación apropiada de estos componentes basados en lenguajes de bajo nivel (HHL).

4.3.4 Direccionamiento de puerto I/O

El procesador soporta un espacio de dirección I/O que contiene por encima de los 65, 536, 8 bits puertos de I/O. Los puertos de 16 bits y 32 bits pueden ser definidos en este espacio de dirección.

Un puerto I/O puede ser direccionado ya sea como un operando inmediato o como un valor en el registro DX.

---

[1] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 5-1

[2] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 5-1

[3] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 5-4

[4] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 5-7

[5] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 5-8

[6] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 5-9

CAPITULO V
LLAMADA A PROCEDIMIENTOS, INTERRUPCIONES Y EXCEPCIONES

Objetivo

Explicar el funcionamiento de las interrupciones, llamadas y excepciones a procedimientos.

Introducción

En el siguiente capítulo se describen las facilidades que ofrece la Arquitectura Intel (IA) para ejecutar llamadas a procedimientos o subrutinas. También se describe la manera en como funcionan las interrupciones y excepciones desde la perspectiva de un programa de aplicación.

5.1 Tipos de llamada de procedimientos

El procesador soporta llamadas a procedimientos de dos diferentes maneras:

- Instrucciones CALL y RET.
- Instrucciones ENTER y LEAVE, en conjunto con las instrucciones CALL y RET.

Estos dos mecanismos de llamadas de procedimientos utilizan el procedimientos stack (pila); este procedimiento es utilizado para salvar el estado de la llamada de un procedimiento, pase de parámetros para la llamada de un procedimiento, y para almacenar variables locales para el procedimiento que se esta ejecutando.

Las facilidades del procesador para manejar interrupciones y excepciones son similares a las utilizadas por las instrucciones CALL y RET.

5.2 El stack (pila)

La pila es un continuo arreglo de localidades de memoria. Esta está contenida en un segmento e identificado por el segmento selector en el registro SS. Cuando se está utilizando el modelo de memoria "flat" la pila puede ser localizada en algún lugar del espacio de dirección lineal por el programa. Una pila puede estar por encima de los 4 gigabytes de longitud del tamaño máximo de un segmento.

La siguiente localidad de memoria disponible sobre la pila es llamada tope de la pila.

En algún momento la pila tiene que entregar la dirección guardada, el apuntador de pila (contenido en el registro ESP) da la dirección (que es el offset procedente de la base del segmento SS) del tope de la pila.

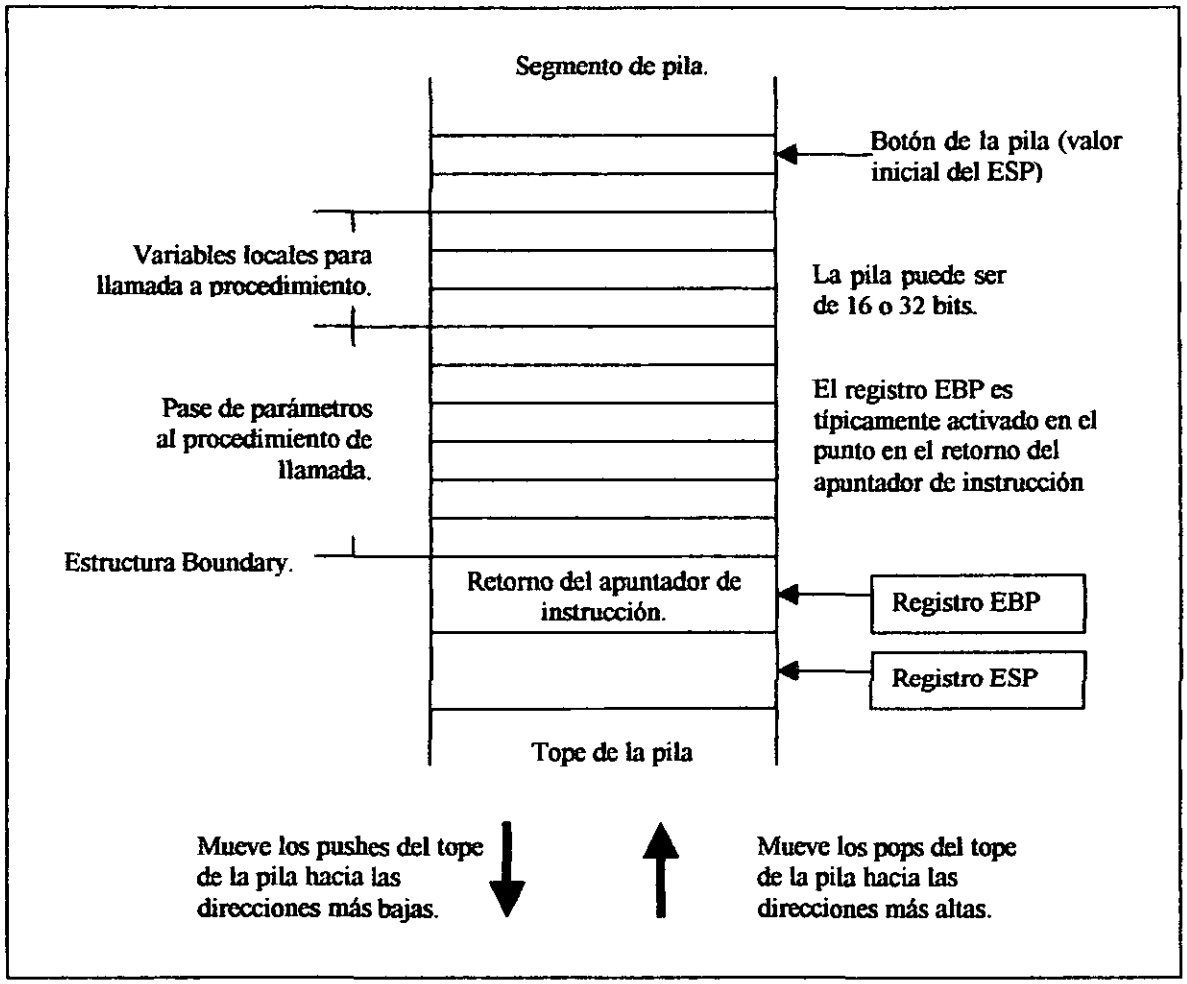
Un dato es situado sobre la pila usando la instrucción *push* y removiendo de la pila mediante la instrucción *pop*. Cuando un dato es puesto sobre la pila, el procesador decrementa el registro ESP, entonces escribe el dato en el nuevo tope de la pila. Cuando un dato es sacado de la pila, entonces se incrementa el registro ESP. De esta manera, la pila aumenta en memoria cuando los datos son puestos sobre la pila y decrementa la memoria cuando los datos son sacados de la pila.

Un programa u operación de sistema/ejecución puede activar muchas pilas. Por ejemplo, un sistema multitareas, cada tarea puede estar dado por su propia

pila. El número de pilas es un sistema limitado por el número máximo de segmentos y la memoria física disponible. Cuando un sistema pone en práctica muchas pilas, únicamente una pila esta disponible en un dato. La pila actual es la contenida en el segmento referenciado por el registro SS.

El procesador referencia el registro SS automáticamente para todos los componentes de la pila. Por ejemplo, cuando el registro ESP es utilizado como una dirección de memoria este automáticamente apunta a la dirección de la pila actual. Además, las instrucciones CALL, RET, PUSH, POP, ENTER y LEAVE realizan operaciones sobre la pila actual. En la figura 5-1 se muestra la estructura de la pila.

Figura 5-1. Estructura de la pila[1]



5.2.1. Activación de una pila

Para activar una pila y establecerla como una pila actual, el programa o sistema operativo puede realizar lo siguiente:

1. Establecer un segmento de pila.
2. Cargar el segmento selector para el segmento de pila en el registro SS usando una instrucción MOV, POP, o LSS.

3. Carga el apuntador de pila por la pila en el registro ESP utilizando una instrucción MOV, POP o LSS. La instrucción LSS puede ser utilizada para cargar los registros SS y ESP en una operación.

5.2.2 Alineación de la pila

El apuntador de la pila para un segmento de la pila debe ser alineado en 16 bits (palabra) o 32 bits (palabra doble) como límite, dependiendo del ancho del segmento de pila. Las instrucciones POP y PUSH utilizan la bandera D para determinar que tanto deben decrementar o incrementar el apuntador de pila sobre una operación POP o PUSH, respectivamente. Cuando el ancho de la pila es de 16 bits, el apuntador de pila es incrementado o decrementado en incrementos de 16 bits; cuando el ancho es de 32 bits, el apuntador de pila es incrementado o decrementado en incrementos de 32 bits.

El procesador no checa la alineación del apuntador de pila. Esta es responsabilidad de los programas, tareas, y procedimientos del sistema que están corriendo sobre el procesador para mantener una alineación apropiada del apuntador de la pila. La desalineación de un apuntador de pila puede causar serias degradaciones en la ejecución de un programa y en algunas instancias del programa puede causar un fracaso.

5.2.3. Atributos de tamaño de dirección para accesar a la pila

Las instrucciones que utilizan la pila implícitamente (tal como las instrucciones PUSH y POP) tienen dos atributos para el tamaño de dirección, cada uno de estos puede ser de 16 o 32 bits. Esto se debe a que estas instrucciones siempre tienen la dirección implícita de el tope de la pila, y estos

pueden también tener una dirección de memoria explícita (por ejemplo, PUSH Array1[EBX]. El atributo de la dirección explícita es determinada por la bandera D del segmento de código actual y la presencia o ausencia del prefijo de tamaño de dirección 67H, como lo mas usual.

El atributo del tamaño de dirección de el tope de la pila determina si el SP o ESP es utilizado para accesar a la pila. Las operaciones de pila con un atributo de tamaño de dirección de 16 utiliza los 16 bits del SP del registro y puede utilizar un máximo de dirección de FFFFH; las operaciones de pila con un atributo de tamaño de dirección de 32 bits usan los 32 bits del ESP del registro y puede utilizar un máximo de dirección de FFFFFFFFH. El atributo de tamaño de dirección default por el segmento de datos utilizado como pilas, es controlado por la bandera B del descriptor del segmento. Cuando esta bandera se limpia, el atributo del tamaño de dirección default es 16. Cuando la bandera se activa, el atributo del tamaño de dirección es 32.

5.3. Llamada de procedimientos utilizando CALL y RET

Las instrucciones CALL permiten el control de transferencia a procedimientos dentro del segmento de código actual (llamada cercana) y en un segmento de código diferente (llamada lejana). Las llamadas cercanas generalmente proporcionan accesos a procedimientos locales dentro del programa o tarea que esta corriendo correctamente. Las llamadas lejanas son generalmente utilizadas para accesar a procedimientos de sistemas operativos o procedimientos en una tarea diferente.

La instrucción RET también permite retornos cercanos y lejanos igual que en las versiones de la instrucción CALL cercana y lejana. La instrucción RET

permite un programa para incrementar el apuntador de pila en retorno para liberar parámetros desde la pila. El número de bytes liberados desde la pila esta determinado por un argumento opcional (n) para la instrucción RET.

5.3.1. CALL cercano y operación RET

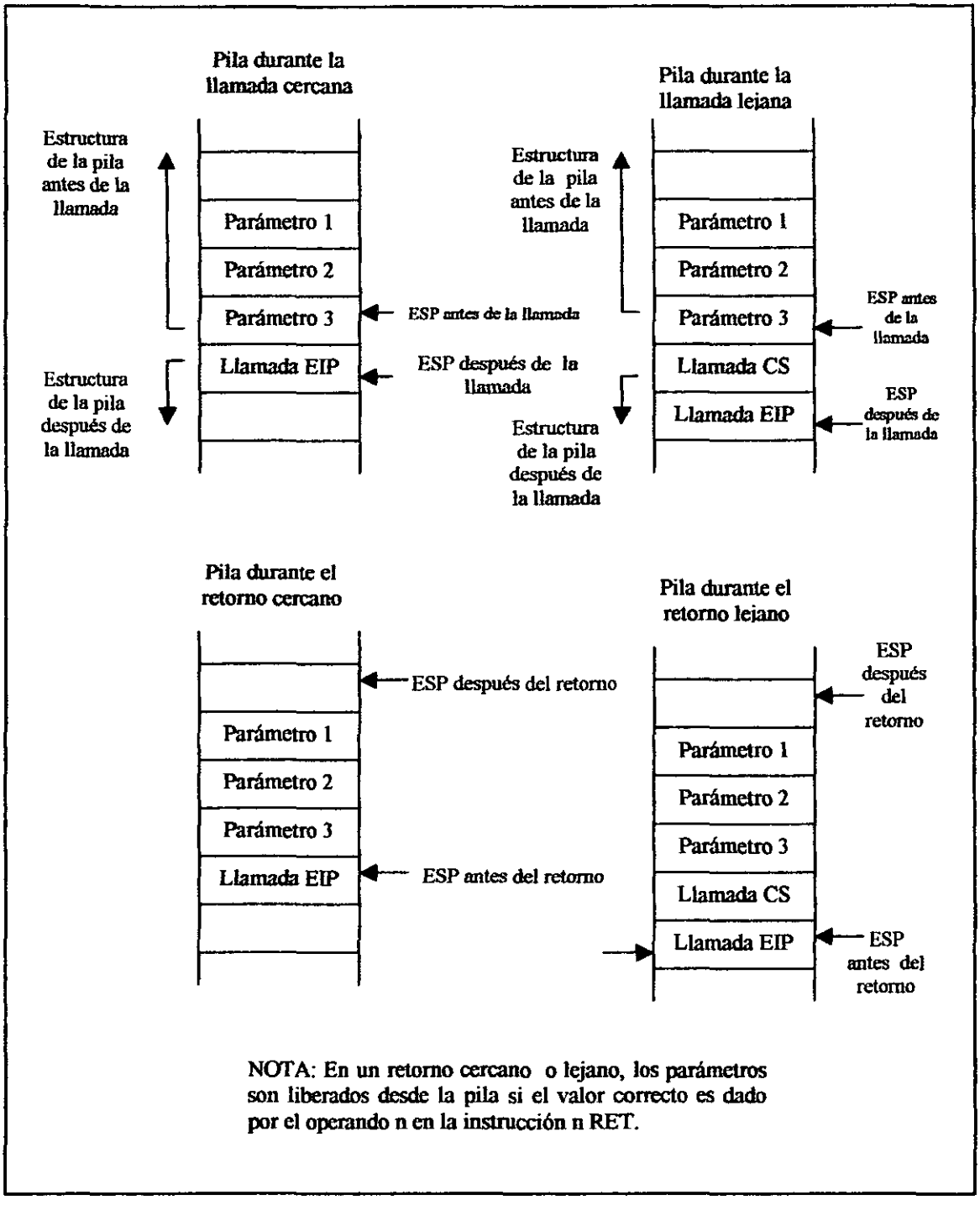
Cuando se esta ejecutando un call cercano, el procesador realiza lo siguiente (ver figura 5-2):

1. Coloca el valor actual del registro EIP en la pila.
2. Carga el offset del procedimiento llamado en el registro EIP.
3. Comienzo de la ejecución del procedimiento llamado.

Cuando se esta ejecutando un retorno cercano, el procesador realiza las siguientes acciones:

1. Saca el valor del tope de la pila (el retorno de la instrucción apuntada) en el registro EIP.
2. (si la instrucción RET tuvo un argumento opcional n). Incrementa el apuntador de pila por el número de bytes especificado con el operando n para liberar parámetros desde la pila.
3. Resume la ejecución de las llamadas del procedimiento.

Figura 5-2. Pila en llamadas cercana y lejana[2]



NOTA: En un retorno cercano o lejano, los parámetros son liberados desde la pila si el valor correcto es dado por el operando n en la instrucción n RET.

5.3.2. CALL lejano y operación RET

Cuando se está ejecutando un *call* lejano, el procesador realiza las siguientes acciones (ver figura 5-2):

1. Coloca el valor actual del registro CS en la pila.
2. Coloca el valor actual del registro EIP en la pila.
3. Carga el segmento selector del segmento que contiene la llamada del procedimiento en el registro CS.
4. Carga el offset de la llamada del procedimiento en el registro EIP.
5. Comienzo de la ejecución de la llamada del procedimiento.

Cuando se está ejecutando un retorno lejano, el procesador realiza lo siguiente:

1. Saca el valor del tope de la pila (el retorno de la instrucción apuntada) en el registro EIP:
2. Saca el valor del tope de la pila (el segmento selector por el código del segmento siendo retornado) en el registro CS.
3. (si la instrucción RET tuvo un argumento opcional n). Incrementa el apuntador de pila por el número de bytes especificado con el operando n para liberar parámetros desde la pila.
4. Resume ejecuciones de la llamada del procedimiento.

5.3.3 Pase de parámetros

Los parámetros pueden ser pasados entre los procedimientos en alguna de estas tres maneras: registros de propósito general, en una lista de argumentos, o en la pila.

5.3.3.1 Pase de parámetros a través de los registros de propósito general

El procesador no salva el estado de los registros de propósito general en las llamadas de procedimientos. Una llamada de procedimiento puede de este modo pasar seis parámetros en alguno de estos registros (excepto los registros ESP y EBP) previo a la ejecución de la instrucción CALL. La llamada a procedimiento puede igualmente pasar parámetros antes de la llamada de un procedimiento a través de los registros de propósito general.

5.3.3.2. Pase de parámetros en la pila

Para pasar un largo número de parámetros en la llamada de un procedimiento, los parámetros pueden ser situados sobre la pila, en la estructura de la pila por la llamada del procedimiento. Aquí, esto sirve para utilizar el apuntador base de la estructura de la pila (en el registro EBP para hacer una estructura límite para un fácil acceso a los parámetros.

La pila puede ser también utilizada para pasar parámetros de la llamada del procedimiento en un procedimiento de llamadas.

5.3.3.3 Pase de parámetros en una lista de argumentos

Un método alternativo de pasar un número largo de parámetros (o una estructura de datos) en la llamada de un procedimiento es colocando los parámetros en una lista de argumentos en uno de los segmentos de datos en memoria. Un apuntador para la lista de argumentos puede entonces ser pasado por la llamada de procedimiento a través de los registros de propósito general

o la pila. Los parámetros pueden ser también pasados antes en el procedimiento de llamada de esta misma manera.

5.3.4 Salvando la información del estado del procedimiento

El procesador no salva el contenido de los registros de propósito general, registros de segmento, o registro de banderas en un procedimiento de llamada. Una llamada de procedimiento debe explícitamente salvar los valores en alguno de los registros de propósito general que este necesitara cuando resuma la ejecución después de un retorno. Estos valores pueden ser salvados en la pila o en memoria en uno de los segmentos de datos.

La instrucción PUSHP y POPA salvando y restaurando los contenidos de los registros de propósito general. PUSHA coloca los valores en todos los registros de propósito general en la pila en el siguiente orden: EAX, ECX, EDX, EBX, ESP (el valor previo a la ejecución de la instrucción PUSHA), EBP, ESI, y EDI. La instrucción POPA saca todos los valores del registro salvados con una instrucción PUSHA (excepto el valor ESI) desde la pila a sus registros respectivos.

Si una llamada a procedimiento cambia el estado de alguno de los registros de segmento explícitamente, este de be restaurarlos a su valor anterior antes de ejecutar un retorno para la llamada de procedimiento.

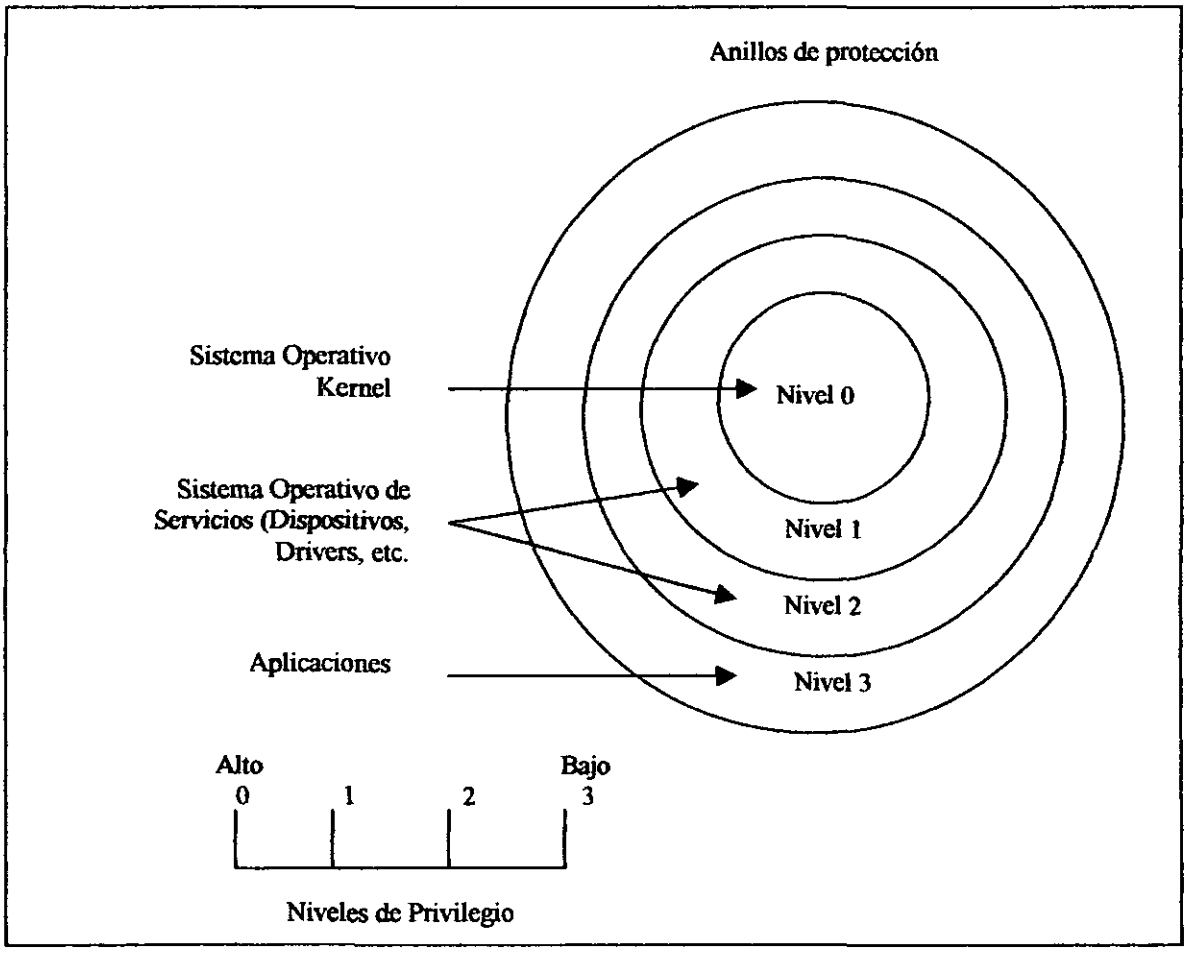
Si una llamada a procedimiento necesita mantener el estado de registro de banderas este puede salvar y restaurar todo o parte del registro usando las instrucciones PUSHF/PUSHFD y POPF/POPFD. La instrucción PUSHF coloca la palabra baja del registro de banderas sobre la pila, mientras la

instrucción PUSHF coloca el registro entero. La instrucción POPF saca una palabra desde la pila en la palabra baja del registro de banderas, mientras la instrucción POPFD saca una palabra doble desde la pila en el registro.

5.3.5 Llamadas a otros niveles de privilegio

El mecanismo de protección de la Arquitectura Intel reconoce cuatro niveles de privilegio numerados de 0 a 3. La primera razón para utilizar estos niveles de privilegio es para mejorar la reliability de los sistemas operativos. Por ejemplo, en la figura 5-3 se muestra como los niveles de privilegio pueden ser interpretados como anillos de protección.

Figura 5-3 Anillos de Protección[3]



En este ejemplo, el nivel de privilegio 0 más alto (en el centro del diagrama) es usado por segmentos que contiene los módulos de código más críticos en el sistema, generalmente el kernel (núcleo) de un sistema operativo. Los anillos externos (con privilegios progresivamente menores) son usados por segmentos que contienen módulos de código para software menor crítico.

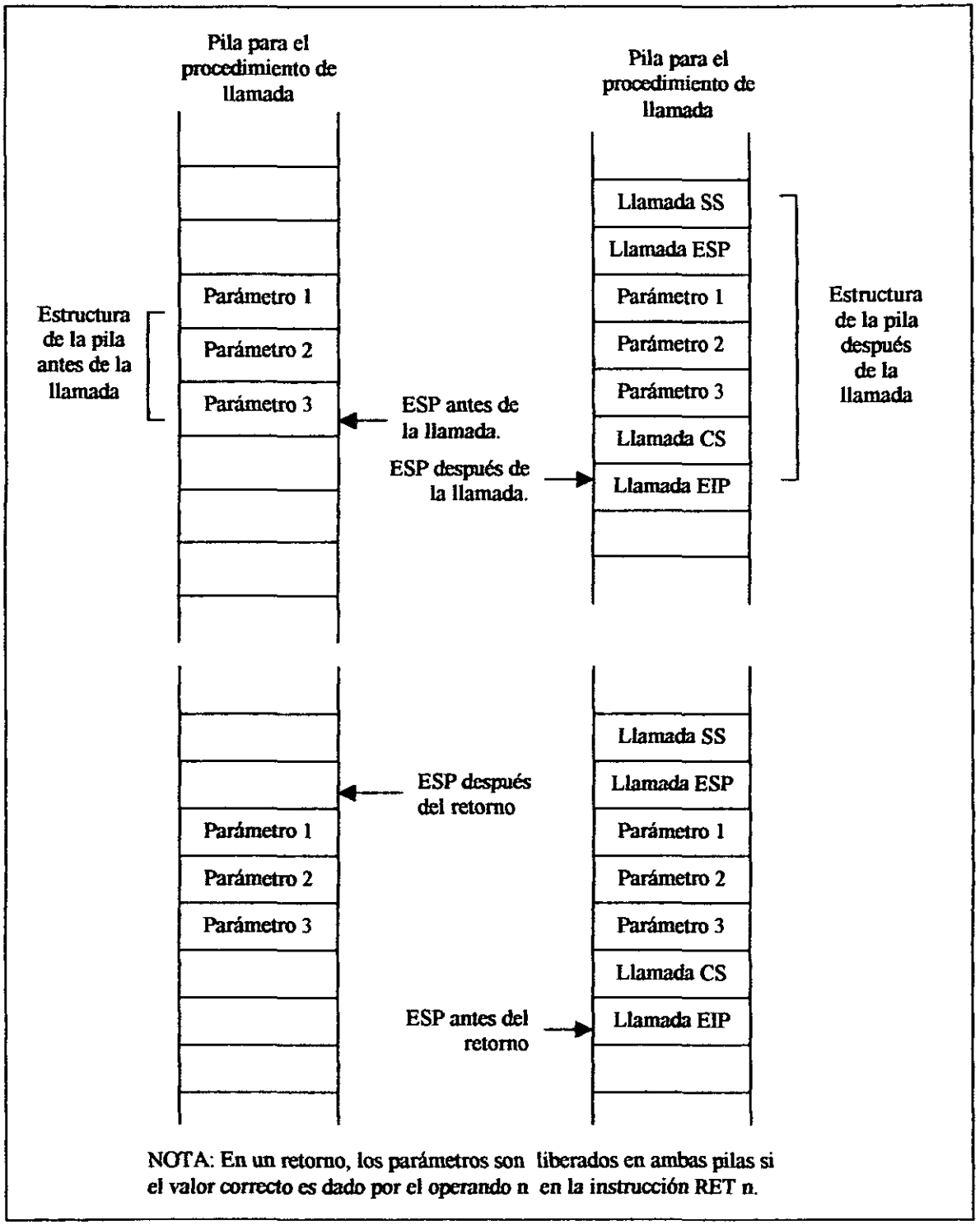
Los módulos de código en segmentos de privilegios menores pueden únicamente accesar a operaciones de módulos en segmentos de privilegio alto por medio de una interface muy fuerte controlada y protegida, llamada "gate". Tratar de accesar a segmentos de privilegios saltos sin ninguna puerta de protección y sin tener acceso suficiente a los anillos causa una excepción de protección general (#GP).

5.3.6 Operación CALL y RET entre los niveles de privilegio

Cuando se esta haciendo se esta haciendo una llamada a un nivel de protección más privilegiado, el procesador realiza lo siguiente, (ver figura 5-4):

1. Realiza un control del acceso de anillos (control de privilegio).
2. Temporalmente salva (internamente) el contenido actual de los registros SS, ESP, CS, y EIP.
3. Carga el segmento selector y apuntador de pila para la nueva pila (esto es, la pila por el nivel de privilegio siendo llamado) desde la TSS (estado del segmento tarea) en los registros SS y ESP e interruptores a la nueva pila.

Figura 5-4. Interrupción de pila en la llamada de un nivel de privilegio diferente[4]



4. Coloca los valores salvados temporalmente SS y ESP por la llamada de la pila del procedimiento en la nueva pila.
5. Copia los parámetros desde la llamada de la pila del procedimiento en la nueva pila. Un valor en el descriptor de la llamada de puerta determina cuantos parámetros copiar en la nueva pila.
6. Carga el segmento selector por el nuevo segmento de código y la nueve instrucción apuntada desde la llamada "gate" en los registros CS y EIP, respectivamente.
7. Comienza la ejecución de la llamada de procedimiento en el nuevo nivel de privilegio.

Cuando se esta ejecutando un retorno desde el procedimiento privilegiado, el procesador realiza las siguientes acciones:

1. Realiza un control de privilegio.
2. Restaura los registros CS y EIP a sus valores previos a la llamada.
3. (si la instrucción RET tenia un argumento opcional n). Incrementa el apuntador de pila por el número de bytes especificado con el operando n para liberar parámetros desde la pila. Si la llamada del descriptor "gate" especifica que uno o más parámetros son copiados desde una pila a otra, una instrucción RET n debe ser usada para liberar los parámetros de ambas pilas. Aquí, el operando n especifica el número de bytes ocupado en cada pila por los parámetros. En un retorno, el procesador incrementa el ESP por n por cada pila en sobre paso (efectivamente removida) de estos parámetros desde las pilas.

4. Restaura los registros SS y ESP a sus valores previos a la llamada, los cuales causan un interruptor antes en la pila de la llamada de procedimiento.
5. (si la instrucción RET tuvo un argumento opcional n). Incrementa el apuntador de pila por número de bytes especificado con el operando n para liberar parámetros desde la pila.
6. Resume la ejecución de la llamada de procedimiento.

5.4 Interrupciones y excepciones

El procesador proporciona dos mecanismos para interrumpir la ejecución de un programa:

Interrupciones y excepciones:

- Una interrupción es un evento asíncrono hasta que es típicamente engatillada por un mecanismo de I/O.
- Una excepción es un evento sincrono que es generado cuando el procesador detecta una o más condiciones predefinidas mientras se esta ejecutando una instrucción. La Arquitectura Intel especifica tres clases de excepciones: fault, trampas, y abortos.

El procesador responde a interrupciones y excepciones esencialmente en algunos modos. Cuando una interrupción o excepción es señalada, el procesador detiene la ejecución de el programa o tarea actual e interrumpe al procedimiento manejador que ha sido escrito específicamente para manejar la condición de interrupción o excepción. El procesador accesa al procedimiento manejador a través de la entrada en la tabla del descriptor de interrupción

(IDT). Cuando el manejador a completado la interrupción o excepción el control del programa es regresado al programa o tarea interrumpida.

El sistema operativo, ejecución, y/o mecanismo de manejo normalmente maneja interrupciones y excepciones independientemente desde programas o tareas de aplicación. Los programas de aplicación pueden, de alguna manera, accesar la interrupción y excepción manejados, incorporando en un sistema operativo o ejecución a través de llamadas de lenguaje ensamblador.

La Arquitectura Intel define 17 interrupciones y excepciones predeterminadas y 224 interrupciones definidas por el usuario, las cuales son asociadas con entradas en la IDT. Cada interrupción y excepción en la IDT es identificada con un número, llamada vector. En la tabla 5-1 se listan las interrupciones y excepciones con entradas en la IDT y sus números respectivos de vector. Vectores de 0 a 8, 10 a 14, 16 a19, son las interrupciones y excepciones predefinidas, y vectores de 32 a 255 son las interrupciones definidas por el usuario, llamadas interrupciones mascarables.

El procesador define algunas interrupciones adicionales que no están apuntadas en la IDT; la interrupción más notables es la SMI.

Tabla 5-1 Interrupciones y Excepciones[5]

No. de Vector.	Mnemonico	Descripción	Fuente
0	#DE	División de error.	Instrucciones DIV e IDIV.
1	#DB	Debug.	Algún código o referencia de dato.
2		Interrupción NMI.	Interrupción externa no mascarable.
3	#BP	Punto roto.	Instrucción INT 3.
4	#OF	Sobre flujo.	Instrucción INTO.
5	#BR	Rango excedido BOUND.	Instrucción BOUND.

6	#UD	Código de operación invalido (código de operación indefinido).	Instrucción UD2 o código de operación reservado.[1]
7	#NM	Dispositivo no disponible.	Punto flotante o instrucción WAIT/FWAIT.
8	#DF	Fault doble.	Alguna instrucción que puede generar una excepción, una NMI, o una INTR.
9		Segmento del coprocesador sobre corriendo (reservado).	Instrucción de punto flotante.[2]
10	#TS	TSS invalido.	Acceso al interruptor de tarea o TSS.
11	#NP	Segmento no presente.	Registros del segmento de carga o acceso a segmentos de sistema .
12	#SS	Segmento de pila por defecto.	Operaciones de pila y carga del registro SS.
13	#GP	Protección general.	Alguna referencia de memoria y otros chequeos de protección.
14	#PF	Pagina por defecto.	Alguna referencia de memoria.
15		(reservado por Intel. No esta en uso).	
16	#MF	Error de punto flotante.	Punto flotante o instrucción WAIT/FWAIT.
17	#AC	Chequeo de alineación.	Alguna referencia de dato en memoria.[3]
18	#MC	Chequeo de máquina.	Códigos de error y fuentes son modelo dependiente.[4]
19	#XF	Extensiones SIMD.	Excepciones numéricas de punto flotante SIMD.[5]
20-31		(reservado por Intel. No esta en uso).	
32-255		Interrupciones mascarables.	Interrupción externa desde la instrucción INTR pin o INT n.

[1] La instrucción UD2 fue introducida en el procesador Pentium Pro.
[2] Los procesadores de Arquitectura Intel después del procesador Intel 386 no generaron esta excepción.
[3] Esta excepción fue introducida en el procesador Intel 486.
[4] Esta excepción fue introducida en el procesador Pentium y mejorada en el procesador Pentium Pro.
[5] Esta excepción fue introducida en el procesador Pentium III.

Cuando el procesador detecta una interrupción o excepción, este realiza una de las siguientes acciones:

- Ejecuta una llamada implícita a un procedimiento manejado.
- Ejecuta una llamada implícita a una tarea manejadora.

El procesador Pentium III puede generar dos tipos de excepciones:

- Excepciones numéricas.
- Excepciones no numéricas.

Cuando las excepciones numéricas ocurren, soporta extensiones SIMD tomando una de dos posibles recursos de acción:

- El procesador puede manejar la excepción por si mismo, produciendo el más razonable resultado y permitiendo la ejecución de un programa numérico para continuar sin interrupción.
- Un software manejador de excepción puede ser invocado para manejar la excepción.

Cada una de las condiciones de excepción numérica tuvo correspondencia de bandera y bits mascarables en la MXCSR (control del estado del registro con extensión SIMD). si una excepción es mascarada (la correspondiente mascara de bit en MXCSR = 1), el procesador toma una acción apropiada default y continua con la computación. Si la excepción es demascarada (mascara bit = 0) y la OS soporta excepciones punto flotante SIMD (CR4.OSXMMEXCPT=1), un software manejador de excepción de interrupción vector 19 punto flotante SIMD. si la excepción es desmascarada (mascara bit = 0) y la OS no soporta excepciones de punto flotante SIMD

(CR4.OSXMMEXCPT=0), un código de operación de excepción invalida es señalado en lugar de una excepción de punto flotante.

5.4.1 Operación CALL y RETURN por manejador de procedimientos de interrupción o excepción

Una llamada a una interrupción o excepción en un manejador de procedimiento es similar a una llamada de procedimiento. Aquí, el vector de interrupción se refiere a uno de los dos tipos de "gates": una interrupción "gate" o una trampa "gate". La interrupción y trampa "gates" son similares a las llamadas "gates", las cuales proporcionan la siguiente información:

- Acceso a los anillos de información.
- El segmento selector por el código de segmento contiene el manejador del procedimiento.
- Un offset en el código de segmento a la primera instrucción del manejador del procedimiento.

La diferencia entre una interrupción "gate" y una trampa "gate" es como siguen: si una interrupción o excepción es llamada a través de una interrupción "gate", el procesador limpia la bandera de interrupción habilitada (IF) en el registro de banderas para prevenir subsecuentes interrupciones que estén interfiriendo con la ejecución del manejador. Cuando un manejador es llamado a través de la trampa "gate", el estado de la bandera IF no es cambiada.

Si en código del segmento para el manejador de procedimiento tuvo el mismo nivel de privilegio como la tarea o programa actualmente ejecutado, el

manejador de procedimiento utiliza la tarea actual; si el manejador se ejecuta en un nivel más privilegiado, el procesador interrumpe a la pila por el nivel de privilegio del manejador.

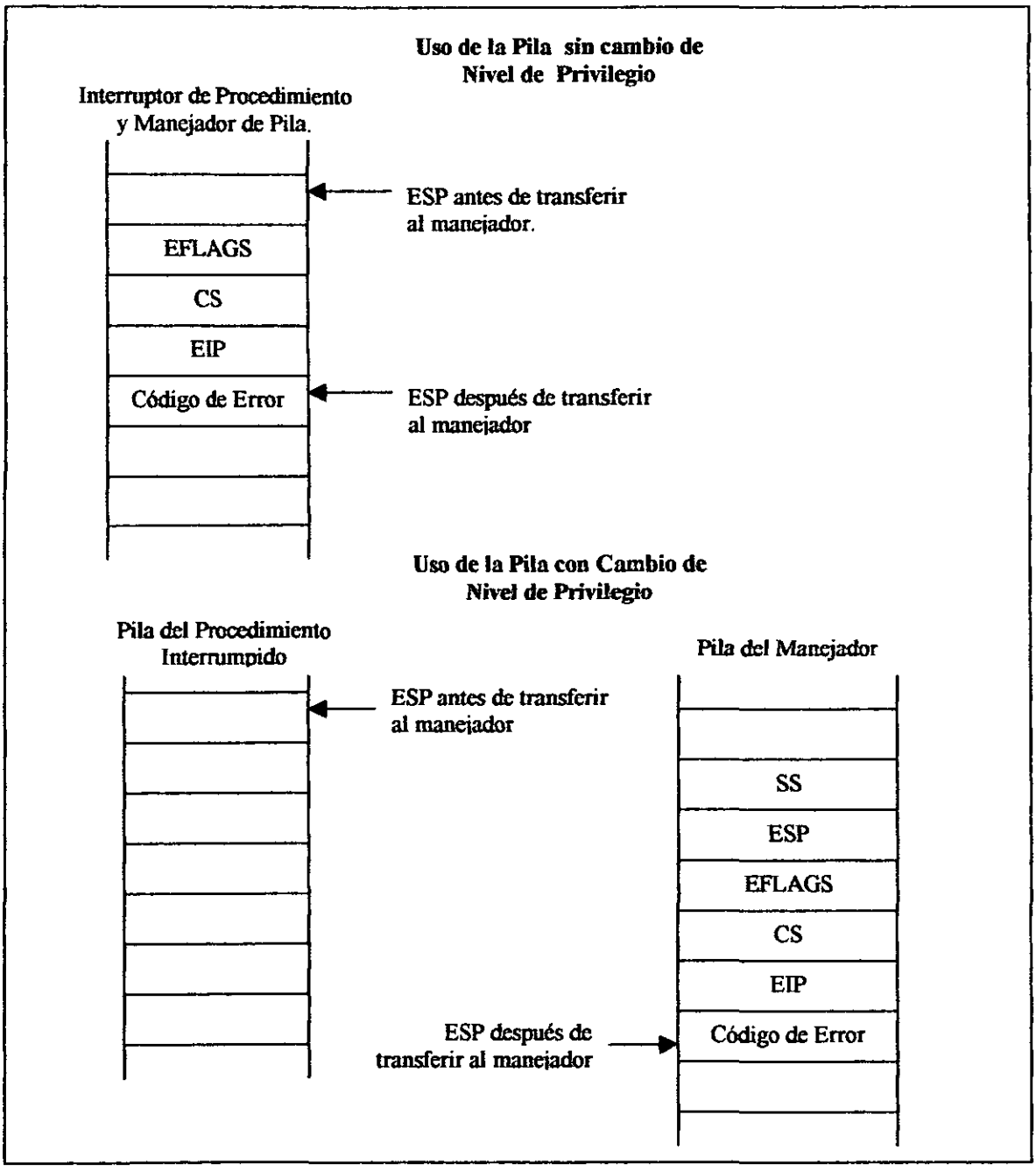
Si no ocurre una interrupción en la pila, el procesador realiza los siguiente cuando llama un manejador de interrupción o excepción, (ver figura 5-5):

1. Coloca el contenido actual de los registros EFLAGS, CS y EIP (en este orden) en la pila.
2. Coloca un código de error (si es apropiado) en la pila.
3. Carga el segmento selector para el nuevo código de segmento y el nuevo apuntador de instrucción (desde la interrupción "gate" o trampa "gate") en los registros CS y EIP, respectivamente.
4. Si la llamada es a través de una interrupción "gate", limpia la bandera IF en el registro EFLAGS.
5. Comienza la ejecución del manejador del procedimiento en el nuevo nivel de privilegio.

Si ocurre una interrupción en la pila, el procesador realiza lo siguiente:

1. Salva temporalmente (interrumpe) los contenidos actuales de los registros SS, ESP, EFLAGS, CS y EIP.
2. Carga el segmento selector y apuntador de pila por la nueva pila (esto es, la pila por el nivel de privilegio que esta siendo llamado) desde la TSS en los registros SS y ESP e interrumpe la nueva pila.
3. Coloca los valores salvados temporalmente en SS, ESP, EFLAGS y EIP de la pila del procedimiento interrumpido de la nueva pila.
4. Coloca un código de error en la nueva pila (si es apropiado).

Figura 5-5 Uso de la pila en transferencias de interrupción y excepción manejando rutinas[6]



5. Carga el segmento selector para el nuevo código de segmento y el nuevo apuntador de instrucción (desde la interrupción "gate" o trampa "gate") en los registros CS y EIP, respectivamente.
6. Si la llamada es a través de una interrupción "gate", limpia la bandera IF en el registro EFLAGS.
7. Comienza la ejecución del manejador de procedimiento en el nuevo nivel de privilegio.

Un retorno desde un manejador de interrupción o excepción es inicializado con la instrucción IRET. La instrucción IRET es similar a la instrucción lejana RET, excepto que esta también restaura el contenido del registro EFLAGS por el procedimiento interrumpido.

Cuando se esta ejecutando un retorno mediante un manejador de interrupción o excepción en un mismo nivel de privilegio tal como el del procedimiento interrumpido, el procesador realiza lo siguiente:

1. Restaura los registros CS y EIP a sus valores previos a la interrupción o excepción.
2. Restaura el registro EFLAGS.
3. Restaura los registros SS y ESP a sus valores previos a la interrupción o excepción.
4. Resume la ejecución del procedimiento interrumpido.

5.4.2 Llamada a interrupción o excepción por un manejador de tareas

La interrupción y excepción maneja rutinas que pueden ser ejecutadas en una tarea separada. Aquí, una interrupción o excepción causa una interrupción

de tarea o un manejador de tarea. El manejador de tarea entrega el espacio de dirección que tiene y (opcionalmente) puede ejecutarse en un nivel de protección más alto que los programas de aplicación o tareas.

El interruptor del manejador de tarea es acompañado con una llamada de tarea implícita que se refiere a un descriptor de tarea "gate". La tarea "gate" proporciona acceso al espacio de dirección para el manejador de tarea. Como parte del interruptor de tarea, el procesador salva el estado completo de la información para el programa o tarea interrumpido. Sobre el retorno desde el manejador de tarea, el estado del programa o tarea interrumpida es restaurado y continua la ejecución.

5.4.3 Interrupción y excepción manejando un modo de dirección real

Cuando se esta operando se esta operando en modo de dirección real, el procesador responde a la interrupción o excepción con una llamada lejana implícita en el manejador de interrupción o excepción. El procesador utiliza el número de vector de interrupción o excepción como un índice en una tabla de interrupción. La tabla de interrupción contiene apuntadores de instrucción en los procedimientos manejedores de interrupción y excepción.

El procesador salva el estado de los registros EFLAGS, EIP, CS y un código de error opcional en la pila antes de interrumpir a el procedimiento manejador.

Un retorno desde un manejador de interrupción o excepción es llevado a cabo con la instrucción IRET.

5.4.4 Instrucciones INT n, INTO, INT 3, y BOUND

En las instrucciones INT n, INTO, INT 3, y BOUND permiten a un programa o tarea llamar explícitamente un manejador de interrupción o excepción. La instrucción INT n usa un vector de interrupción como un argumento, el cual permite a un programa llamar algún manejador de interrupción.

La instrucción INTO llama explícitamente el manejador de excepción de sobre flujo (#OF) si la bandera de sobre flujo (OF) en el registro EFLAGS esta activada. La bandera OF indica sobre flujo o instrucciones aritméticas, pero esta no aumenta automáticamente un sobre flujo de excepción. Un sobre flujo de excepción solo puede ser aumentado explícitamente en alguno de los siguientes métodos:

- Ejecución de la instrucción INTO.
- Prueba de la bandera OF y ejecución de la instrucción INT n con un argumento de 4 (el número de vector del sobre flujo de excepción) si la bandera esta activada.

La instrucción INT 3 llamada explícitamente el manejador de excepción de punto roto (#BP).

La instrucción BOUND llama explícitamente el manejador de excepción excedido de rango BOUND (#BR) si un operando es creado para no estar dentro de los limites predefinidos en memoria. Esta instrucción es proporcionada para controlar referencias o arreglos y otras estructuras de datos. Al igual que la excepción de sobre flujo, la excepción excedida en rango BOUND solo puede ser aumentada explícitamente con la instrucción

BOUND o la instrucción INT n con un argumento de 5 (el número de vector de la excepción bounds-check). El procesador no permite implícitamente bounds-check y aumenta la excepción excedida de rango BOUND.

5.5. Llamada de procedimientos en lenguaje de bloque estructurado

La Arquitectura Intel soporta un método alternativo de realizar llamadas a procedimientos con las instrucciones ENTER (procedimiento enter) y LEAVE (procedimiento leave). Estas instrucciones automáticamente crean y liberan, respectivamente, estructuras de pila por llamadas de procedimiento. Las estructuras de pila tiene espacios predefinidos por variables locales y apuntadores necesarios para permitir regresar coherentes a las llamadas de procedimientos.

Las instrucciones ENTER y LEAVE ofrecen los siguientes beneficios:

- Proporcionan soporte de lenguaje – máquina para implementar lenguajes de bloque estructurado, tal como C y Pascal.
- Simplifican procedimientos de entrada y salida en el código generado por el compilador.

5.5.1 Instrucción ENTER

La instrucción ENTER crea una estructura de pila compatible del ámbito de reglas típicamente utilizados en lenguajes de bloque estructurado. El lenguaje de bloque estructurado el ámbito de un procedimiento es el set de variables en el cual este a accesado. Las reglas de ámbito varían entre los lenguajes.

En la instrucción ENTER tiene dos operandos. El primero especifica el número de bytes para ser reservados en la pila por un almacén dinámico para el procedimiento que esta siendo llamado. El almacén dinámico es la memoria localizada por variables creadas cuando el procedimiento es llamado, también conocidos como variables automáticas. El segundo parámetro es el léxico nivel anidado (desde 0 a 3) del procedimiento. La anidación del nivel es la profundidad de un procedimiento en una jerarquía de llamadas a procedimiento. El nivel léxico esta sin relación en algún nivel de privilegio de protección o en el nivel de privilegio I/O del programa o tarea que actualmente están corriendo.

5.5.2 Instrucción LEAVE

La instrucción LEAVE, la cual no tiene ningún operando, realiza lo contrario de la instrucción ENTER. La instrucción LEAVE copia el contenido del registro EBP en el registro ESP todo el espacio de la pila distribuida en un procedimiento. Entonces esta restaura el valor anterior del registro EBP hacia la pila. Este restaura simultáneamente el registro ESP a su valor original. Una instrucción subsecuente RET puede entonces remover algún argumento y colocar el regreso de la dirección en la pila por el programa llamado por el procedimiento.

---

[1] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 4-2

[2] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 4-6

[3] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 4-9

[4] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 4-11

[5] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 4-14

[6] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 4 –15

CAPITULO VI
UNIDAD DE PUNTO FLOTANTE (FPU)

Objetivo

Explicar el funcionamiento de la Unidad de Punto Flotante

Introducción

La unidad de punto flotante (FPU) en la Arquitectura Intel proporciona un alto funcionamiento en la capacidad de procesamiento de punto flotante. Esta soporta datos de tipo real, entero, entero BCD, procesamiento de algoritmos de punto flotante, Arquitectura de excepción definida en los estándares IEEE 754 y 854 de la aritmética de punto flotante. La FPU ejecuta instrucciones desde la instrucción normal "stream" del procesador y mejora la eficiencia de los procesos de la Arquitectura Intel en el tipo de precisión alta de punto flotante, procesando operaciones comúnmente fundadas en aplicaciones de ciencia, ingeniería y negocios.

6.1 Formatos de números reales y punto flotante

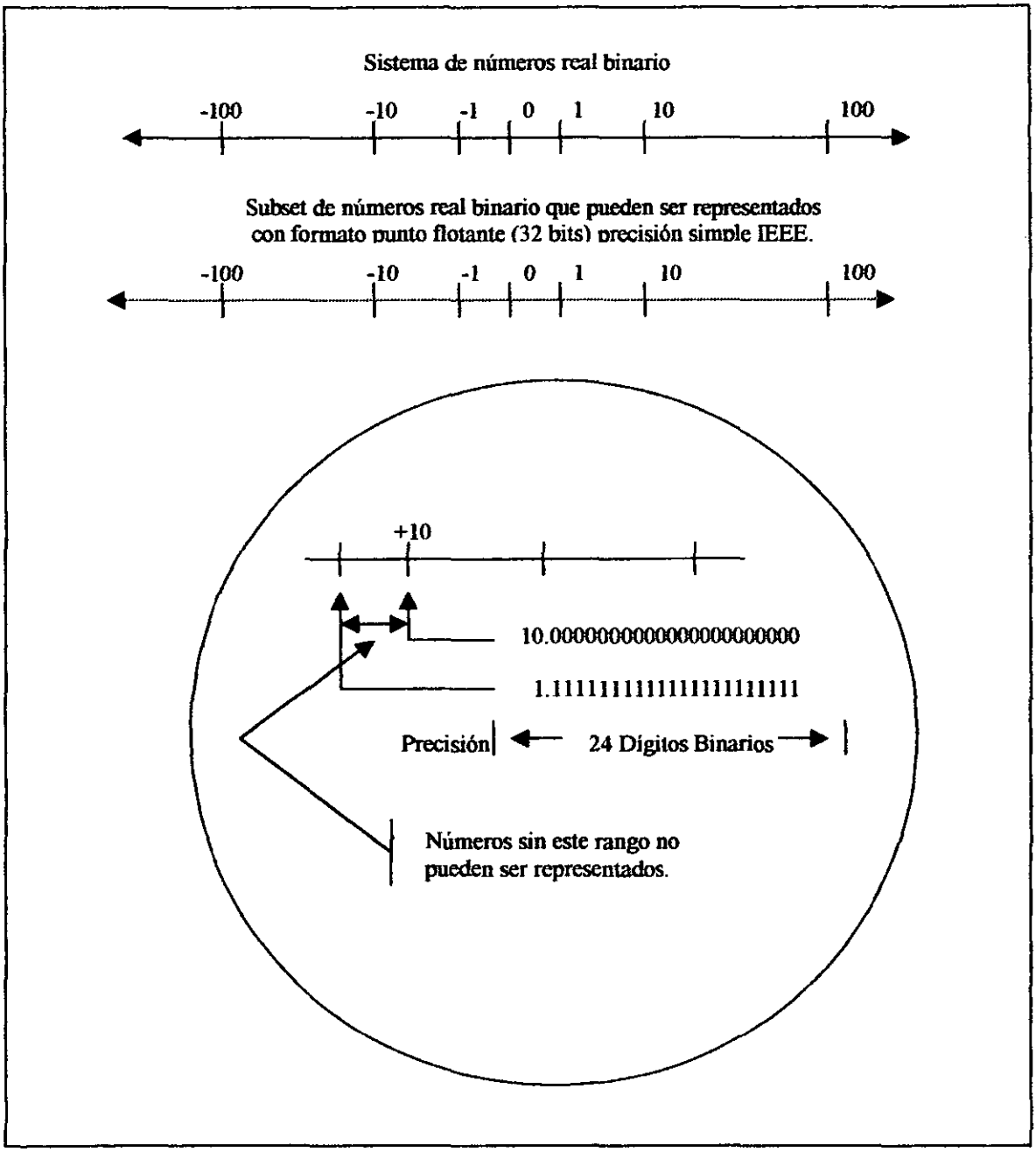
En la siguiente sección se describe como son representados los números reales en formato de punto flotante dentro de la Arquitectura Intel en la FPU, por otro lado se describen los términos como números normalizados, números desnormalizados, exponentes base, ceros con signo y NaNs

6.2.1. Sistema de números reales

En la figura 6-1 se muestra el sistema de números reales que esta comprendido de menos infinito hasta más infinito.

El tamaño y número de registros que algún computador puede tener esta limitado, solo un subset de números real continuo puede ser usado el calculo de números reales. Como se muestra en el circulo de la figura 6-1, el subset de números reales que una particular FPU soporta, representa una aproximación del sistema de números reales. El rango y precisión de este subset de números reales esta determinado por el formato que la FPU utiliza para representar números reales.

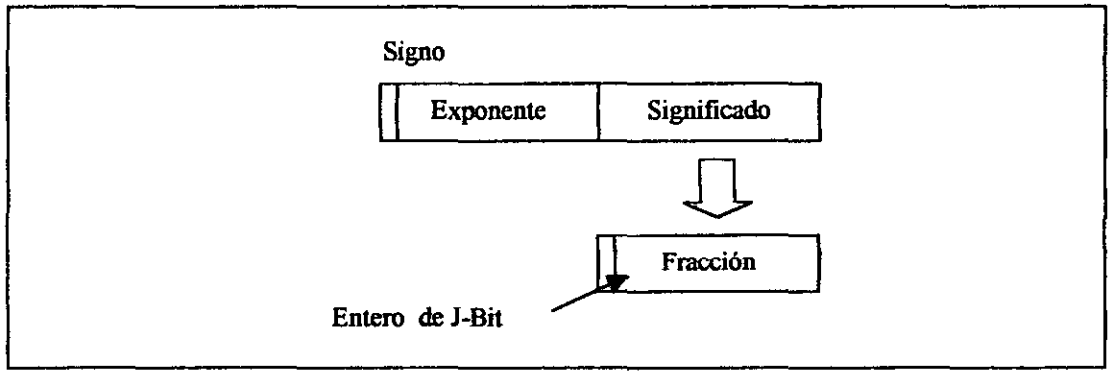
Figura 6-1 Sistema de números real binario[1]



6.1.2 Formato de punto flotante

Para incrementar la rapidez y eficiencia de computación de números reales, las computadoras o FPUs típicamente representan números reales en un formato de punto flotante. En este formato, un número real tiene tres partes: un signo, un significado y un exponente. La figura 6-2 muestra el formato binario de punto flotante que la FPU de la Arquitectura Intel utiliza. Este formato conforma el estándar IEEE.

Figura 6-2 Formato de punto flotante binario[2]



El signo es un valor binario que indica si el número es positivo (0) o negativo (1) el significado tiene dos partes: 1 bit binario entero (también llamado como J bit) y una fracción binaria. El J bit esta con frecuencia no representada, pero en cambio es un valor sobre entendido. El exponente es un entero binario que representa el poder de la base 2 que el significado a aumentado.

La tabla 6-1 muestra como el número real 178.125 (en formato de decimal ordinario) es almacenado en formato de punto flotante.

La tabla lista una progresión de rotaciones de número real que aventaja al real simple, formato punto flotante de 32 bits (el cual es uno de los formatos

de punto flotante que soporta la FPU). En este formato el significado es normalizado y el exponente es predisponido. Para el formato de real simple la constante predispuesta es + 127.

Tabla 6-1 Notación de número real[3]

Notación	Valor		
Decimal ordinario	178.125		
Decimal científico	$1.78125E_{10}$ 2		
Binario científico	$1.0110010001E_{2}$ 111		
Binario científico (base exponente)	$1.0110010001E_{2}$ 10000110		
Formato real simple	Signo	Exponente base	Significado normalizado
	0	10000110	01100100010000000000000 1.

6.1.2.1 Números normalizados

En muchos casos, la FPU representa números reales en forma normalizada. De esta manera, excepto para cero, el significado esta siempre hecho de un entero de 1 y la fracción: 1.fff..ff Para valores menores que 1, los ceros principales son eliminados. Por cada cero principal eliminado, el exponente es decrementado por uno. La representación de números en forma normalizada maximiza el número de dígitos significantes que pueden ser acomodados en un significado de un ancho dado. Un número real normalizado consiste de un significado normalizado que representa un número real entre 1 y 2 en el exponente que especifica el punto binario del número.

6.1.2.2 Exponente predispuesto

La FPU representa exponentes en forma predispuesta. De esta manera una constante es agregada al exponente actual de manera que el exponente predispuesto es siempre un número positivo. El valor de la constante predispuesta depende del número de bits disponible para representar exponentes en el formato de punto flotante que esta siendo utilizado. La constante predispuesta es seleccionada de manera que el pequeñisimo número normalizado puede ser reciproco sin sobre flujo.

6.1.3 Codificación de número real y no número

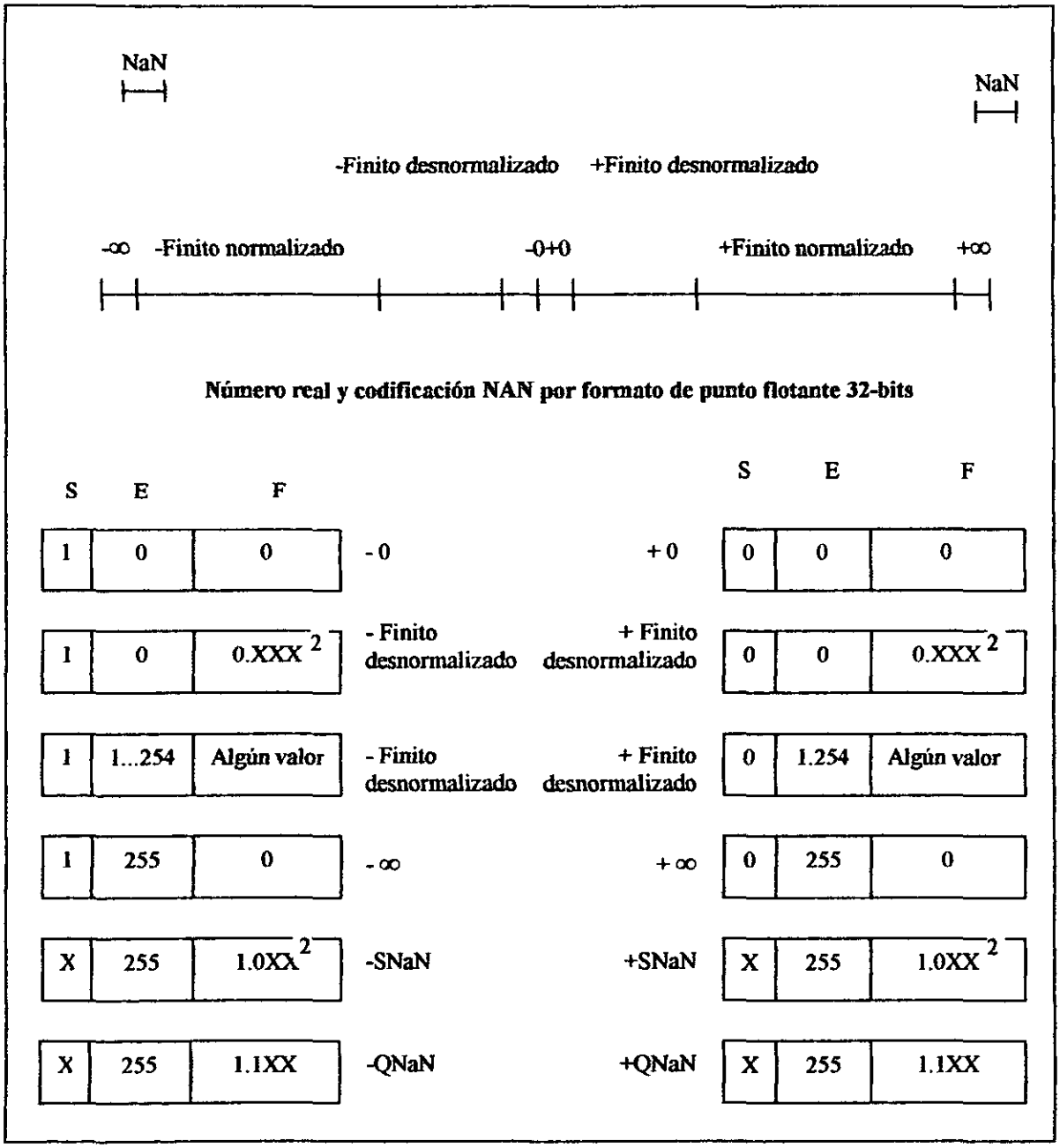
Una variedad de números reales y valores especiales pueden ser codificados en el formato de punto flotante de la FPU. Estos números y valores están generalmente divididos en las siguientes clases:

- Ceros con signo.
- Números finitos desnormalizados.
- Números finitos normalizados.
- Infinitos con signo.
- NaNs (el termino NaN es utilizado para un no número ("not a number")).
- Números indefinidos.

La figura 6-3 muestra la codificación para estos números y no números en el número real continuo. La codificación muestra el formato de precisión simple IEEE (32 bits), donde el termino "S" indica el bit de signo, "E" exponente predispuesto y "F" la fracción. El valor del exponente esta dado en decimal.

La FPU puede operar sobre y/o el regreso de alguno de estos valores, dependiendo del tipo de computación que este siendo realizado.

Figura 6-3 Números reales y NaNs[4]



6.1.3.1 Ceros con signo

El cero puede ser representado como un +0 ó un –0 dependiendo del bit de signo. Ambas codificaciones son iguales en valor. El signo resultado de un cero depende de la operación que fue realizada y el modo de redondeo que esta siendo usado. Los ceros con signo deben proporcionar ayuda en la implementación de intervalo. El signo de un cero puede indicar la dirección desde la cual ocurrió un menor flujo (under flow), o puede indicar el signo de un ∞ que tiene que ser reciproco.

6.1.3.2 Números finitos normalizados y desnormalizados

No - cero, números finitos son derivados en dos clases: normalizados y desnormalizados. Los números finitos normalizados comprenden todos los valores finitos no - cero que pueden ser codificados en forma de número real normalizado entre cero y ∞. La figura 6-3 muestra el formato de real simple, este grupo de números incluye todos los números con exponentes predispuestos en un rango de 1 a 254 (base 10).

Cuando los números reales llegan a ser muy cerrados a cero, el formato de números normalizados no puede alargarse para representar los números. Esto es debido a que el rango del exponente no es una compensación muy larga para el cambio del punto binario a la derecha para eliminar los ceros adelantados.

Cuando el exponente predispuesto es cero, números pequeñisimos solo pueden ser representados para hacer el bit entero del cero significante. Los números en este rango son llamados desnormalizados. El uso de ceros

adelantados con números desnormalizados permiten representar números pequeñisimos. De cualquier manera esto causa desnormalización en perdida de precisión (el número de significado de bits en la fracción es reducida por los ceros adelantados).

Cuando se están realizando computaciones de punto flotante normalizado, una FPU opera normalmente en números normalizados y producen números normalizados como resultado. Los números desnormalizados representan una condición de "underflow".

Un número desnormalizado es computado a través de una técnica gradual llamada "underflow". La tabla 6-2 muestra un ejemplo de "underflow" gradual en el proceso de desnormalización. El formato de real simple esta siendo utilizado, así el exponente mínimo es – 126 (base 10). El verdadero resultado en este ejemplo requiere un exponente de – 129 (base 10) en el orden que tiene un número normalizado.

En el caso extremo, todos los bits significantes son trasladados a la derecha por ceros adelantados, creando un resultado de cero.

Tabla 6-2 Proceso de desnormalización[5]

Operación	Signo	Exponente	Significado
Resultado verdadero	0	-129	1.01011100000...00
Desnormalizado	0	-128	0.10101110000...00
Desnormalizado	0	-127	0.01010111000...00
Desnormalizado	0	-126	0.00101011100...00
Resultado desnormal	0	-126	0.00101011100...00

6.1.3.3 Infinitos con signo

Los infinitos, + ∞ y - ∞, representan los números reales positivo y negativo, respectivamente, estos pueden ser representados en formato de punto flotante. El infinito es representado siempre por un cero significante (fracción y entero bit) y el máximo exponente predispuesto permitido en el formato especificado (por ejemplo, 255_{10} para el formato real simple).

La aritmética de infinitos es siempre exacta. Las excepciones son generadas solo cuando el uso de un infinito es como un operando fuente, lo que constituye una operación invalida.

Mientras que los números desnormalizados representan una condición del "underflow", los dos número infinitos representa el resultado de una condición de sobre flujo. Aquí, el resultado de la normalización de una computación tiene un exponente predispuesto muy largo permitible por el resultado del formato seleccionado.

6.1.3.4 NANS

Desde las NaNs hasta los no - números, no forman parte del número real en línea. En la figura 6-3, el espacio de codificación para las NaNs en los formatos de punto flotante FPU muestra los fines del número real en línea. Este espacio incluye algún valor con el máximo exponente predispuesto permitible y una fracción no – cero (el signo del bit es ignorado por las NaNs).

El estándar IEEE define dos clases de NaN: "quiet" NaNs (QNaNs) y "signaling" NaNs (SNaNs).

Una QNaN, es un NaN con el bit más significante activado de la fracción; un SNaN es una NaN con el bit más significante desactivado de la fracción. La QNaN esta permitida para propagarse a través de más operaciones aritméticas sin señalar una excepción. La SNaN generalmente señala una excepción de operación invalida donde quiera que estas aparezcan como operandos en operaciones aritméticas.

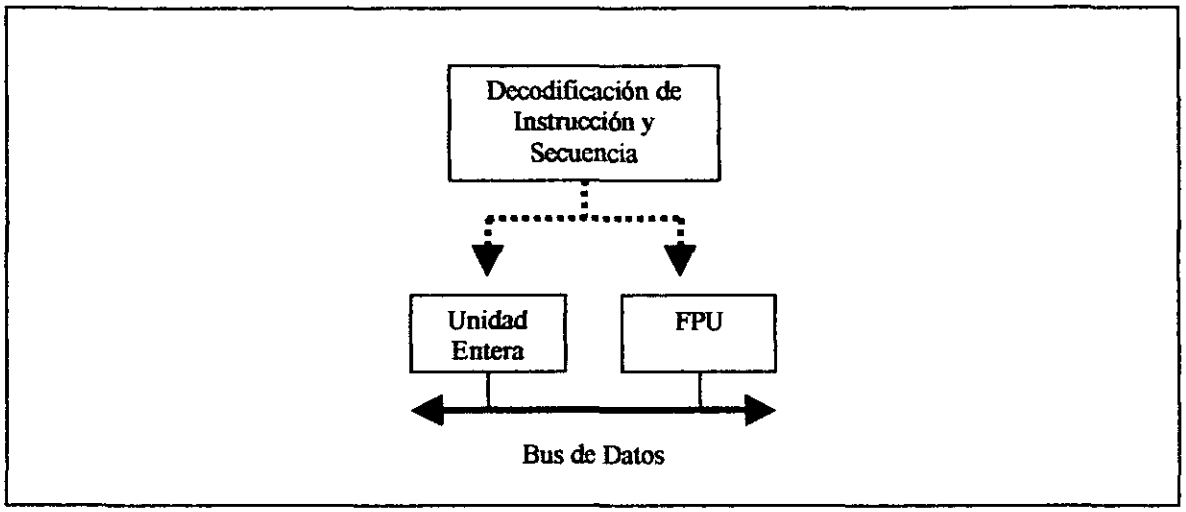
6.1.4 Indefinidos

Para cada tipo de dato FPU, un único código es reservado para representar el valor especial indefinido. Por ejemplo, cuando operan valores reales, el valor real indefinido es una QNaN. La FPU produce valores indefinidos como respuestas para hacer excepciones de punto flotante.

6.2 Arquitectura FPU

La FPU es un coprocesador que opera en paralelo con la unidad de enteros del procesador (ver figura 6-4). La FPU obtiene sus instrucciones por medio de la instrucción de decodificación y secuencia como la unidad d entero y "Shares" del bus del sistema con la unidad de entero. La unidad de entero y la FPU operan independientemente y en paralelo. (La microarquitectura de un procesador de Arquitectura Intel varia según las distintas familias de procesadores. Por ejemplo, el procesador Pentium Pro tiene dos unidades de enteros y dos FPUs; el procesador Pentium tiene dos unidades de enteros y una FPU; y el procesador Intel 486 tiene una unidad de enteros y una FPU).

Figura 6-4 Relación entre la unidad entera y la FPU[6]



La ejecución de instrucción de la FPU consiste en 8 registros de datos, como se puede ver en la figura 6-5, (llamados registros de datos de la FPU) y los siguientes registros de propósito especial:

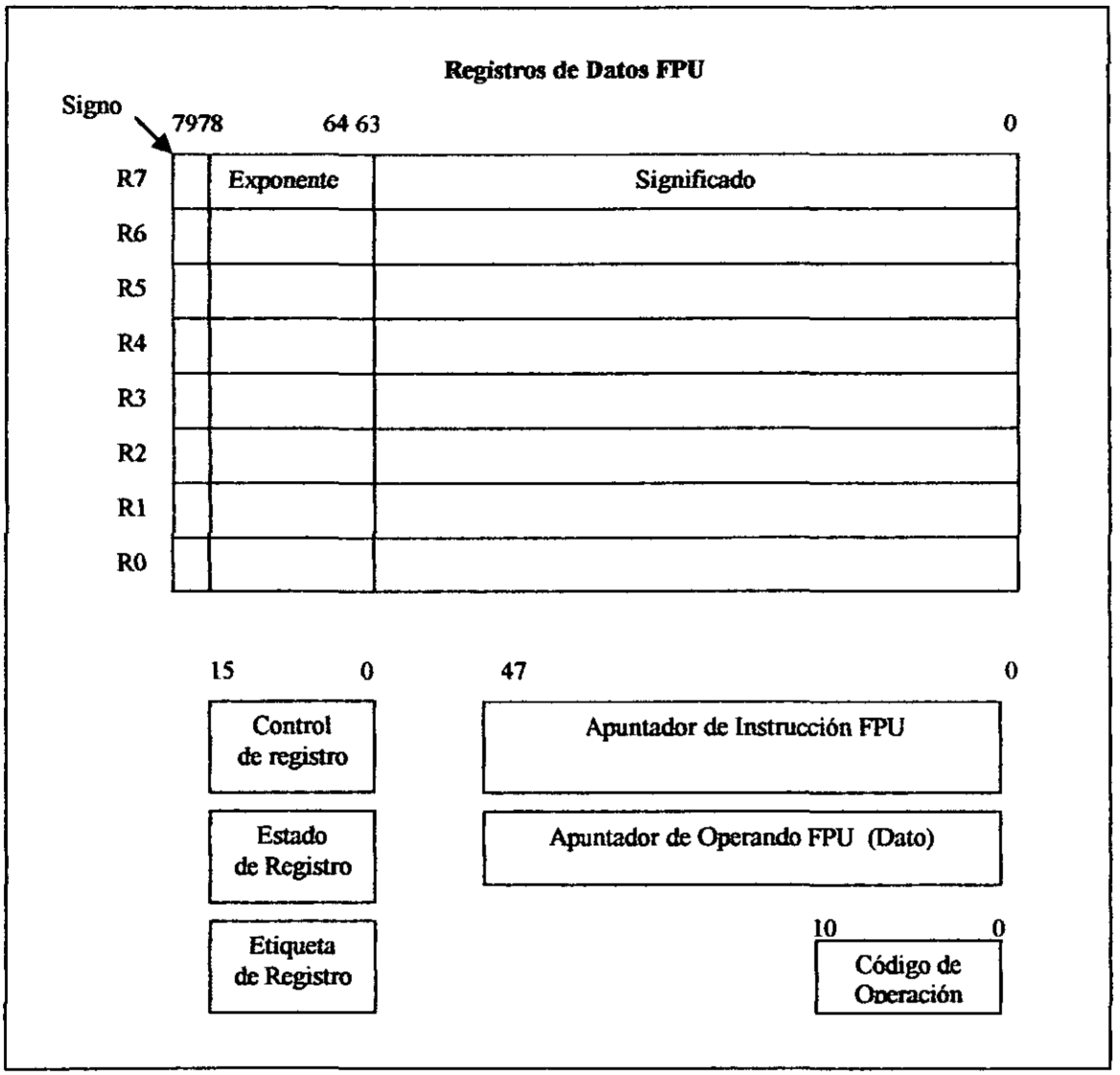
- El registro de estado.
- Registro de control.
- La etiqueta del registro de palabra.
- Registro de apuntador de instrucción.
- Registro del siguiente operando (apuntador de dato).
- Registro de código de operación.

Estos registros son descritos a continuación.

6.2.1 Registros de datos de la FPU

El registro de datos de la FPU , como se muestra en la figura 6-5, consiste de 8 registros de 80 bits. Los valores son almacenados en estos registros en el formato de real extendido. Los valores real, entero, o entero BCD empacado

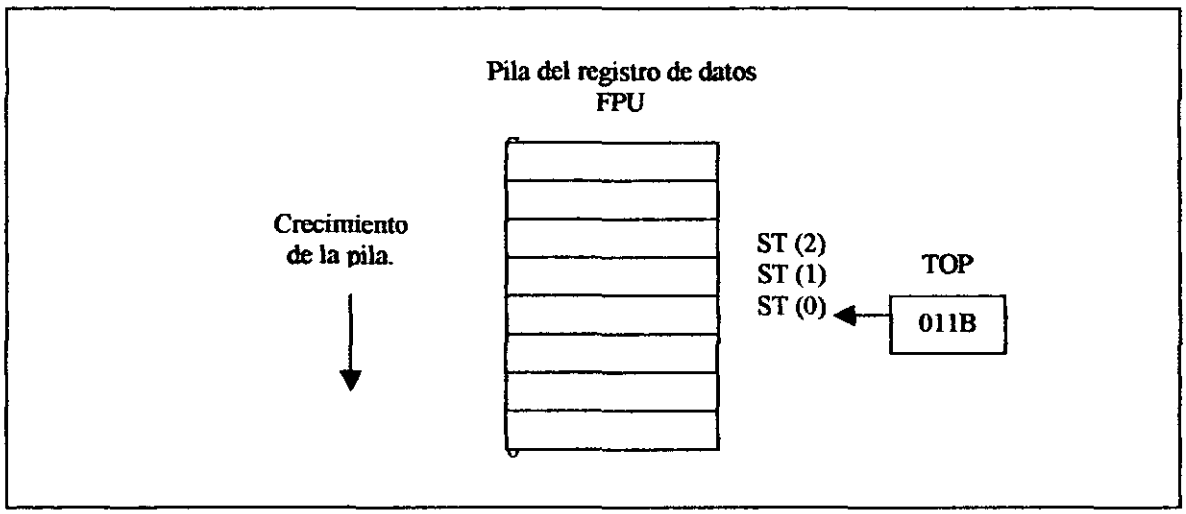
Figura 6-5 Medio ambiente de Ejecución de la FPU[7]



son cargados desde la memoria en algunos de los registros de datos, los valores son automáticamente convertidos dentro del formato de real extendido (si estos no están ya en este formato). Cuando los resultados de la computación son transferidos hacia la memoria en algunos de los registros de la FPU, los resultados pueden estar a la izquierda en el formato de real extendido o convertidos en algún otro formato FPU (real, entero o enteros BCD empacado).

Las instrucciones tratan los ocho registros de datos de la FPU como un registro de pila (ver figura 6-6). Todas las direcciones de los registros de datos son relativas al registro en el tope de la pila. El número de registro es almacenado en el archivo TOP (tope de la pila) en el estado de palabra FPU. Carga las operaciones decrementando el TOP en uno y carga un valor en el nuevo tope de la pila del registro, y almacena operaciones almacenando el valor desde el actual TOP del registro en memoria y entonces incrementa el TOP en uno. Para la FPU, una carga de operación es equivalente a un push y un almacén de operación es equivalente a un pop.

Figura 6-6 Pila del registro de datos de la FPU[8]

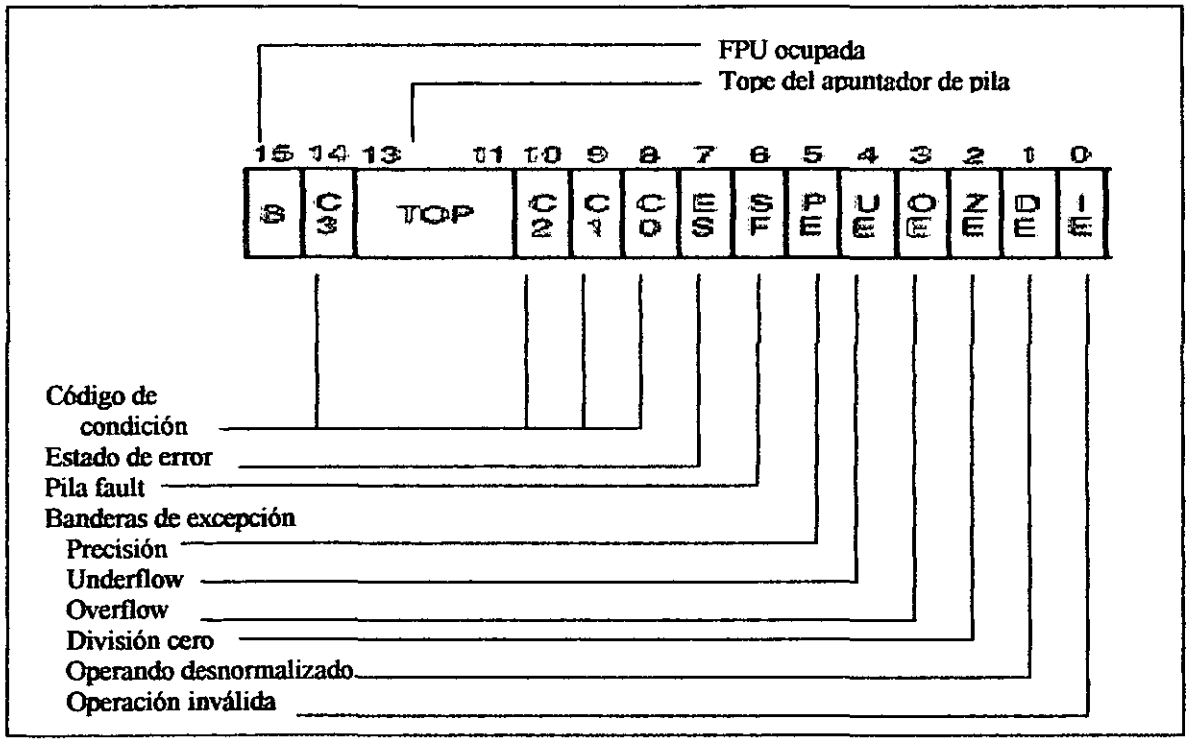


6.2.2 Registro de estado de la FPU

Los 16 bits del registro de estado de la FPU, ver figura 6-7, indican el estado actual de la FPU. Las banderas en el registro de estado de la FPU incluyen la bandera de "busy" FPU, apuntador del tope de la pila (TOP), banderas de condición de código, bandera del estado de error, bandera "fault" de la pila y las banderas de excepción. La activación de las banderas en la FPU en estos registros muestran los resultados de las operaciones.

El contenido de los registros de estado de la FPU pueden ser almacenados en memoria utilizando las instrucciones FSTSW/FNSTSW, FSTENV/FNSTENV y FSAVE/FNSAVE. También pueden ser almacenados en el registro AX de la unidad de entero, utilizando las instrucciones FSTSW/FNSTSW.

Figura 6-7 Estado de la palabra FPU[9]



6.2.3 Ramas y movimientos condicional en los códigos de condición de la FPU

La FPU de la Arquitectura Intel soporta dos mecanismos para ramas y desarrollo de movimientos condicionales de acuerdo a las comparaciones de dos valores de punto flotante. Estos mecanismos son llamados: "mecanismo viejo" y "mecanismo nuevo".

El mecanismo viejo esta disponible en las FPU 's de los procesadores anteriores al procesador Pentium Pro y en el mismo Pentium Pro. Este mecanismo utiliza instrucciones de comparación de punto flotante (FCOM, FCOMP, FCOMPP, FTST, FUCOMPP, FICOM y FICOMP) para comparar dos valores de punto flotante y el set de las banderas del código de condición (C0 hasta C3) de acuerdo a los resultados. El contenido de la condición de las banderas de código son entonces copiados en las banderas de estado de le registro EFLAGS utilizando un procedimiento "step":

1. La instrucción FSTSW AX mueve el estado de la palabra FPU a el registro AX.

2. La instrucción SAHF copia los 8 bits altos del registro AX, el cual incluye las banderas de código de condición, en los 8 bits bajos del registro EFLAGS.

Cuando la condición de las banderas de código han sido cargadas en el registro EFLAGS, los saltos o movimientos condicionales pueden ser realizados en el nuevo set de la bandera de estado en el registro EFLAGS.

El nuevo mecanismo esta disponible únicamente en el procesador Pentium Pro. Este mecanismo se utiliza para la comparación del nuevo punto flotante y set de instrucciones EFLAGS (FCOMI, FCOMIP, FUCOMI y FUCOMIP) compara dos valores de punto flotante y el set de banderas ZF, PF y CF directamente en el registro EFLAGS. Una instrucción simple de este modo reemplaza las tres instrucciones requeridas por el mecanismo viejo.

6.2.4 Control de palabra FPU

Los 16 bits del control de palabra (ver figura 6-8) controlan la precisión de la FPU y el método de redondeo utilizado. Esta también contiene los bits de la mascara de la bandera de excepción. El control de palabra esta cacheada en el control de registro FPU. El contenido de los registros pueden ser cargados con la instrucción FLDCW y almacenados en memoria con las instrucciones FSTCW/FNSTCW.

Cuando la FPU es inicializada con cualquiera de las instrucciones FINIT/FNINIT, el control de palabra FPU es activada por 03FH, las cuales enmascaran todas las excepciones de punto flotante, activa el redondeo mas cercano y activa la precisión FPU en 64 bits.

6.2.5 Etiqueta de palabra FPU

Los 16 bits de la etiqueta de palabra (ver palabra 6-9) indican el contenido de cada uno de los 8 registros en la pila del registro de datos de la FPU (una etiqueta de 2 bits por registro). El código de etiqueta indica si un registro contiene un número válido, cero o un número especial punto flotante (NaN, infinito, desnormal o formato no soportado), o si este esta vacío. La etiqueta

Figura 6-8 Control de palabra FPU[10]

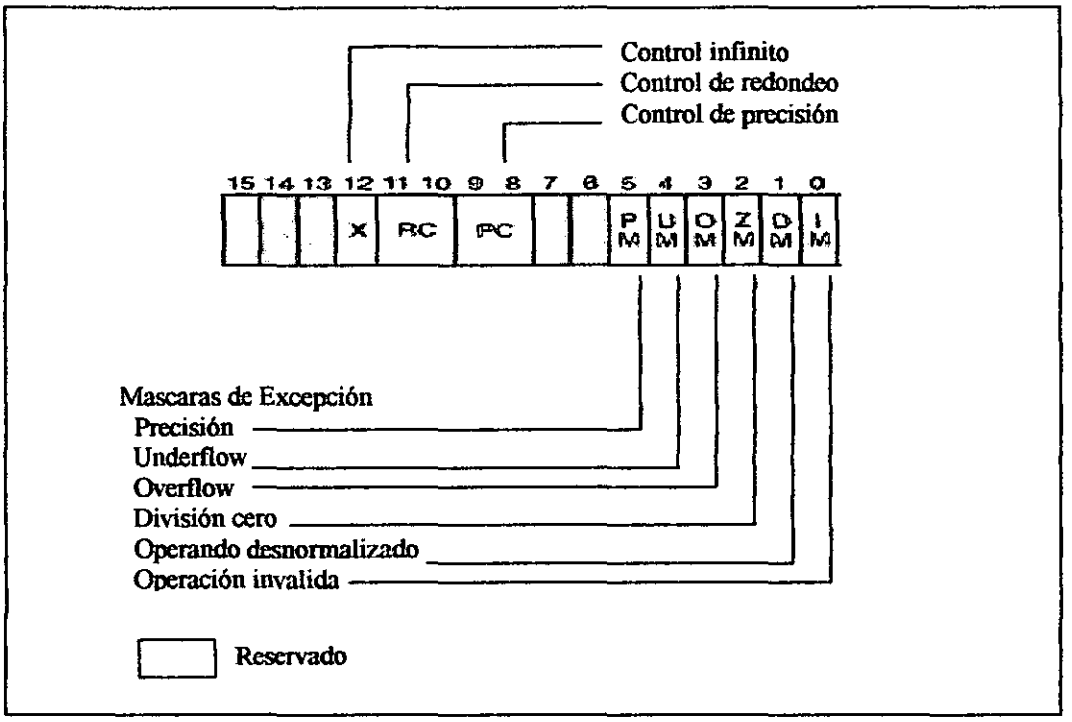
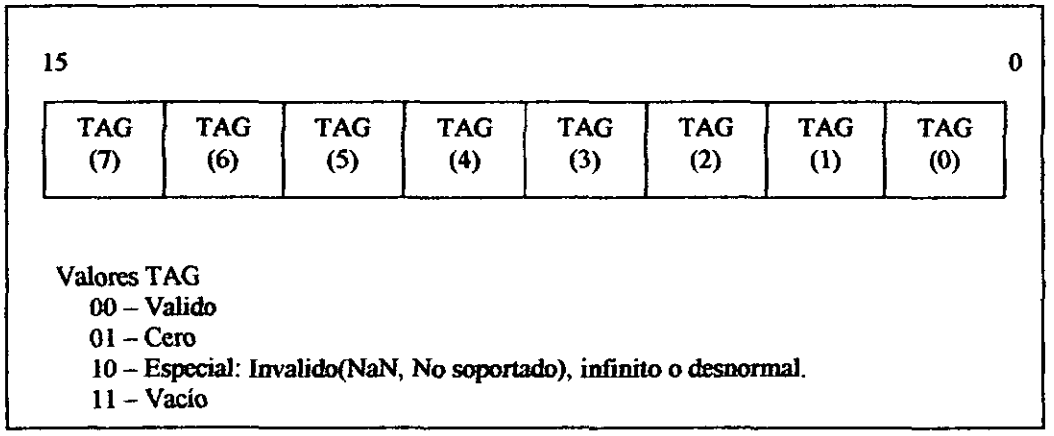


Figura 6-9 Etiqueta de palabra FPU[11]



de palabra esta cacheada en la FPU en el registro de la etiqueta de palabra FPU. Cuando la FPU es inicializada con cualquiera de las instrucciones FINIT/FNINIT o FSAVE/FNSAVE, etiqueta de palabra FPU es puesta en FFFFH, la cual enmascara todos los registros de datos FPU como vacíos.

6.2.6 Instrucción FPU y apuntadores de operando

La FPU almacena apuntadores en la instrucción y operando (dato) por la siguiente instrucción de no control ejecutada en dos registros de 48 bits: el apuntador de instrucción FPU y el apuntador de operando (dato) FPU (ver figura 6-5).

Los contenidos de la instrucción FPU y registros de apuntador de operando permanece sin cambios cuando alguna de las instrucciones de control (FINIT/FNINIT, FCLEX/FNCLEX, FLDCW, FSTCW/FNSTCW, FSTSW/FNSTSW, FSTENV/FNSTENV, FLDENV, FSAVE/FNSAVE, FRSTOR Y WAIT/FWAIT) son ejecutadas. Los contenidos del registro de operando FPU son indefinidos si la anterior instrucción de no control no tuvo un operando en memoria.

Los apuntadores almacenados en la instrucción FPU y operandos del registro de apuntador consisten de un offset (almacenado en bits de 0 a 31) y un segmento selector (almacenado en bits de 32 a 47).

Estos registros pueden ser accesados por las instrucciones FSTEN/FNSTENV, FLDENV, FINIT/FNINIT, FSAVE/FNSAVE y FRSTOR. Las instrucciones FINIT/FNINIT y FSAVE/FNSAVE limpian estos registros.

Para todas FPU's y NPX's de la Arquitectura Intel excepto la 8087, el apuntador de instrucción FPU apunta a algún prefijo que precede a la instrucción. Para el 8087, el apuntador de instrucción FPU apunta únicamente al código de operación actual.

---

[1] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-3

[2] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-4

[3] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-5

[4] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-6

[5] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-7

[6] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-9

[7] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-10

[8] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-10

[9] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-13

[10] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-17

[11] En Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 7-20

CAPITULO VII

EL MACROENSAMBLADOR (MASM 6.x) Y LA PROGRAMACIÓN

Objetivo.

Dar a conocer cómo introducir programas en el Macroensamblador (MASM 6.x).

Introducción.

En este capítulo se explicarán los requisitos básicos para desarrollar un programa en ensamblador: el uso de comentarios, el formato general de codificación, las directivas, algunos ejemplos de programas en ensamblador. También se describirá la organización general de un programa, incluyendo la inicialización y la terminación de su ejecución.

7.1 Ensambladores y compiladores.

Primero identificaremos dos clases de lenguajes de programación: de alto nivel y de bajo nivel. Los lenguajes de alto nivel son aquellos como C y Pascal, los cuales codifican comandos poderosos, cada uno de los cuales puede generar muchas instrucciones en lenguaje máquina. Los lenguajes de bajo nivel como el ensamblador, codifican instrucciones simbólicas, cada una de las cuales genera una instrucción en lenguaje máquina. A pesar de que codificar en lenguajes de alto nivel es más productivo, algunas ventajas de codificar en lenguaje ensamblador son:

- Proporciona más control sobre el manejo particular de los requerimientos de hardware.

- Genera módulos ejecutables y más compactos.
- Con mayor probabilidad tiene una ejecución más rápida.

Sin importar el lenguaje de programación que se utilice, es un lenguaje simbólico que tiene que traducirse a una forma que la computadora pueda ejecutar. Un lenguaje de alto nivel utiliza un *compilador* para traducir el código fuente a lenguaje de máquina (técnicamente código objeto). Un lenguaje de bajo nivel utiliza un *ensamblador* para realizar la traducción. Un lenguaje *enlazador* para ambos niveles, alto y bajo, completa el proceso al convertir el código objeto ejecutable de máquina.

7.2 Las palabras reservadas.

Ciertas palabras en lenguaje ensamblador están reservadas para sus propósitos propios, y son usadas sólo bajo condiciones especiales. Por categorías, las palabras reservadas incluyen:

- Instrucciones, como MOV y ADD, que son operaciones que la computadora puede ejecutar.
- Directivas, como END o SEGMENT, que se emplean para proporcionar comandos al ensamblador.
- Operadores, como FAR y SIZE, que se utilizan en expresiones.
- Símbolos predefinidos, como @Data y @Model, que regresan información a su programa.

El uso de una palabra reservada para un propósito equivocado provoca que el ensamblador genere un mensaje de error.

7.3 Identificadores.

Un identificador es un nombre que se aplica a elementos en el programa. Los dos tipos de identificadores son: nombre, que se refiere a la dirección de un elemento de dato, y etiqueta, que se refiere a la dirección de una instrucción. Las mismas reglas se aplican tanto para los nombres como para las etiquetas. Un identificador puede utilizar los siguientes caracteres:

- Letras del alfabeto: desde la A hasta la Z
- Dígitos: desde el 0 hasta 9 (no puede ser el primer carácter)
- Caracteres especiales: signo de interrogación (?)
  subrayado (_)
  signo de pesos ($)
  arroba (@)
  punto (.) (no puede ser el primer carácter)

El primer carácter de un identificador debe ser una letra o un carácter especial, excepto el punto. Ya que el ensamblador utiliza algunos símbolos especiales en palabras que inician con el símbolo @, debe evitarse usarlo en las definiciones.

El ensamblador trata las letras mayúsculas y minúsculas como iguales. La longitud máxima de un identificador es de 247 caracteres (desde el MASM 6.0). Se recomienda que los nombres sean descriptivos y con significado. Los nombres de registros, como AL, DI y AL, están reservados para hacer referencia a esos mismos registros. Por ejemplo en una instrucción como:

ADD AX, BX

El ensamblador sabe de forma automática que AX y BX se refieren a los registros. Por otro lado en una instrucción como:

MOV REGSAVE,AX

El ensamblador puede reconocer el nombre REGSAVE sólo si se define en algún lugar del programa.

7.4 Instrucciones

Un programa en lenguaje ensamblador consiste en un conjunto de enunciados. Los dos tipos de enunciados son:

1. Instrucciones, tal como MOV y ADD, que el ensamblador traduce a código objeto.
2. Directivas, que indican al ensamblador que realice una acción especifica, como definir un elemento de dato.

A continuación está el formato general de un enunciado, en donde los corchetes indican una entrada opcional:

[identificador]	operación	[operando (s)] [;comentario]

Un identificador (si existe), una operación y un operando (si existe) están separados por al menos un espacio en blanco o un carácter de tabulador. Existe un máximo de 512 caracteres en una línea (desde el MASM 6.0).

Identificador, operación y operando pueden empezar en cualquier columna. Sin embargo, si de manera consistente se inicia en la misma columna para estas entradas se hace un programa más legible.

7.5 Comentarios en lenguaje ensamblador

El uso de comentarios a lo largo de un programa puede mejorar su claridad, en especial en lenguaje ensamblador, donde el propósito de un conjunto de instrucciones no es claro. Un comentario empieza con punto y como (;) y, en donde quiera que lo codifique, el ensamblador supone que todos los caracteres a la derecha de esa línea son comentarios. Un comentario puede contener cualquier carácter imprimible, incluyendo el espacio en blanco.

Un comentario puede aparecer sólo en una línea o a continuación de una instrucción en la misma línea. Por ejemplo:

```
; toda esta línea es un comentario

ADD AX,BX  ;Comentario en la misma línea que la instrucción
```

7.6 Directivas

El lenguaje ensamblador permite usar diferentes enunciados que permiten controlar la manera en que un programa ensambla y lista. Estos enunciados son llamados como *directivas*. Actúan sólo durante el ensamblado de un programa y no generan código ejecutable de máquina (ver el Anexo 1).

7.7 Definición de datos

El ensamblador permite la definición de elementos de varias longitudes de acuerdo con el conjunto de directivas que defina datos. A continuación se listan las longitudes más comunes:

1. DB (define byte)
2. DW (define palabra)
3. DD (define palabra doble)
4. DF (define palabra larga)
5. DQ (define palabra cuádruple)
6. DT (define diez bytes)

Un elemento de datos puede contener un valor indefinido (esto es, no inicializado) o una constante, definida como una cadena de caracteres o como un valor numérico. A continuación se muestra el formato general para la definición de datos:

[nombre] Dn expresión

En donde nombre es el *nombre* del elemento de dato, el cual puede ser opcional.

Dn es la directiva que define el elemento de datos, el cual puede ser: DB, DW, DD, DF, DQ o DT.

La expresión es un operando que puede contener un signo de interrogación para indicar un elemento no utilizado como:

FLD1 DB ? ;Elemento no inicializado

También se puede utilizar el operando para definir una constante, como:

FLD2 DB 25 ;Elemento inicializado

7.7.1 Cadenas de caracteres

las cadenas de caracteres son usadas para datos descriptivos como nombres de personas y títulos de páginas. La cadena está definida dentro de los apóstrofos, como 'PC', o dentro de comillas, como "PC". El ensamblador traduce las cadenas de caracteres en código objeto en formato ASCII normal.

DB es el único formato que define una cadena de caracteres que excede a dos caracteres y los almacena en la secuencia normal de izquierda a derecha. En consecuencia, DB es el formato convencional para la definición de datos de caracteres de cualquier longitud. Por ejemplo:

DB "cadena de caracteres"

7.8 Cómo preparar un programa para su ejecución.

Cualquier programa que se quiera realizar deberá ser tecleado en un editor de textos, el cual puede ser el editor del DOS, y guardado con un nombre y la extensión ASM, la cual será reconocida por el ensamblador. El programa tal como esta escrito no puede ejecutarse, pues es solo un archivo de texto. Para poderlo ejecutar es necesario ensamblarlo y enlazarlo.

El paso de ensamble consiste en la traducción del código fuente en código objeto y la generación de un archivo intermedio .OBJ (objeto), o modulo. Una de las tareas del ensamblador es calcular el desplazamiento de cada elemento

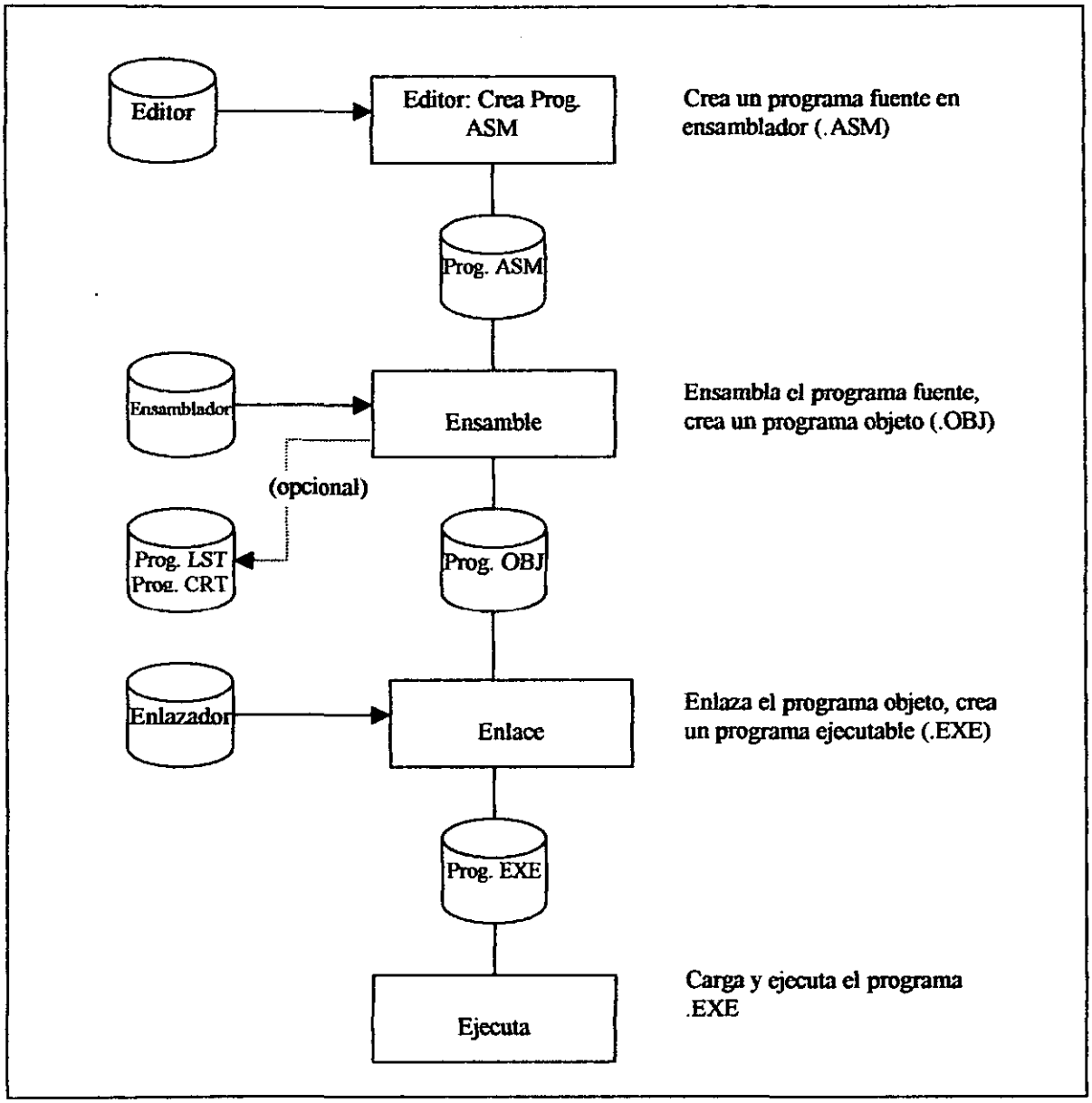
en el segmento de datos y de cada instrucción en el segmento de código. El ensamblador también crea un encabezado al frente del módulo .OBJ generado; parte del encabezado tiene información acerca de direcciones incompletas. El módulo .OBJ aún no esta en forma ejecutable.

El paso de enlace implica convertir el módulo .OBJ en un módulo de código de máquina .EXE (ejecutable). Una de las tareas del enlazador es combinar los programas ensamblados en forma separada en un módulo ejecutable.

El último paso es cargar el programa para su ejecución. Ya que el cargador conoce en dónde está el programa a punto de ser cargado, puede completar las direcciones indicadas en el encabezado que estaban incompletas. El cargador desecha el encabezado y crea un PSP inmediatamente antes del programa cargado en memoria.

En la figura 7-1 se muestran los pasos implicados al ensamblar, enlazar y ejecutar un programa.

Figura 7-1 Pasos para ensamblar, enlazar y ejecutar[1]



7.9 Cómo ensamblar un programa fuente

El programa ensamblador de Microsoft MASM 6.x por lo general utiliza el comando ML para ensamblar, pero también acepta MASM por compatibilidad con versiones anteriores. El formato general para un comando de línea para ensamblar un programa es:

MASM [opciones] fuente[,objeto] [listado] [,refcruzadas]

7.10 Cómo enlazar un programa objeto

Una vez que el programa queda sin mensajes de error, el siguiente paso es enlazar el módulo objeto, ejemplo .OBJ, Que fue producido por el ensamblador y que contiene sólo código de máquina. En enlazador realiza las funciones siguientes:

- Si se pide, combina más de un módulo ensamblado de forma separada en un programa ejecutable, como dos o más programas en ensamblador o un programa en ensamblador con un programa en C.

- Genera un módulo .EXE y lo inicializa con instrucciones especiales para facilitar su subsecuente carga para ejecución.

Una vez que ha enlazado uno o más módulos .OBJ en un módulo .EXE, puede ejecutar el módulo .EXE cualquier número de veces. El comando ML de Microsoft MASM 6.x proporciona tanto el ensamble como el enlace.

7.11 Ejemplos de programas en ensamblador

En la figura 7-2 se ilustran instrucciones de movimiento y de suma, así como la estructura general de cualquier programa en ensamblador.

El STACKSG contiene una entrada, DW (definir palabra), que define 32 palabras inicializadas a cero, un tamaño adecuado para la mayoría de los programas.

El DATASG define tres palabras de datos llamadas FLDA, FLDB y FLDC

El CODESG contiene las instrucciones ejecutables para el programa, aunque el primer enunciado, ASSUME, no genera código ejecutable.

La directiva ASSUME realiza las siguientes operaciones:

- Asigna STACKSG al registro SS, de forma que el sistema utilice la dirección en el registro SS para direccionamiento de STACKSG.
- Asigna DATASG al registro DS, de modo que el sistema utilice la dirección en el registro DS para direccionamiento de DATASG.
- Asigna CODESG al registro CS, de modo que el sistema utilice la dirección en el registro CS para direccionamiento de CODESG.

Figura 7-2 Programa que realiza operaciones de movimiento y de suma

```
PROGASM1 (EXE) Operaciones de mover y sumar

                page 60,132
TITLE           PROGASM1 (EXE)  Operaciones de mover y sumar
;-------------------------------------------------------------------------
STACKSG         SEGMENT         PARA STACK 'Stack'
                DW              32 DUP (0)

STACKSG         ENDS
;-------------------------------------------------------------------------
DATASG          SEGMENT         PARA 'Data'
FLDA            DW              250
FLDB            DW              125
FLDC            DW              ?
DATASG          DATASG          ENDS
;-------------------------------------------------------------------------
CODESG          SEGMENT         PARA 'Code'
BEGIN           PROC            FAR
                ASSUME          SS:STACKSG, DS:DATASG, CS:CODESG
                MOV             AX,DATASG       ;Establecer la dirección
                MOV             DS,AX           ;de DATASG en el
                                                ;registro DS
                MOV             AX,FLDA         ;Mover 0250 a AX
                ADD             AX,FLDB         ;Sumar 0125 a AX
                MOV             FLDC,AX         ;Almacenar suma en FLDC
                MOV             AX,4C00H        ;Salida al DOS
                INT             21H             ;Fin de procedimiento
BEGIN           ENDP
CODESG          ENDS                            ;Fin de segmento
                END             BEGIN           ;Fin de programa
```

7.11.1 Ejemplo de un programa con operaciones de movimiento extendido.

En el programa de la figura 7-3, el segmento de datos contiene dos campos de nueve bytes definidos como NOM1 y NOM2. El objetivo del programa es mover el contenido de NOM1 a NOM2:

NOM1: A B C D E F G H I

NOM2: J K L M N O P Q R

Ya que cada uno de los campos es de 9 bytes, se necesita más de una instrucción MOV. A fin de pasar NOM1 a NOM22, la rutina inicializa el registro CX a 9 (la longitud de los dos campos) y utiliza los registros índice SI y DI. Dos instrucciones LEA cargan las direcciones de desplazamiento de NOM1 a NOM2 en SI y DI.

El programa utiliza las direcciones de los registros SI y DI para mover el primer byte de NOM1 al primer byte de NOM2. Los corchetes alrededor de SI y DI en los operandos de MOV significan que la instrucción es para usar el desplazamiento en el registro dado, a fin de accesar la localidad de memoria.

Dos instrucciones INC incrementan los registros SI y DI en 1, y DEC decrementa el CX en 1. DEC también pone a 1 o a 0 la bandera de cero (ZF), dependiendo del resultado en CX; si el contenido no es cero, aún existen caracteres por mover, y JNZ regresa a la etiqueta B20 para repetir las instrucciones MOV. Y como el SI y DI han sido incrementados en 1, el siguiente MOV hace referencia a NOM1+1 y NOM2+1. El ciclo continua de esta manera hasta que ha movido nueve caracteres en total, hasta mover NOM1+8 a NOM2+8.

Figura 7-3 Programa que mueve el contenido de NOM1 a NOM2

```
                Page 60,132
TITLE           PROGASM2 (EXE)      Operaciones de movimiento extendido
;-------------------------------------------------------------------------
                .MODEL SAMALL
                .STACK 64
;-------------------------------------------------------------------------
                .DATA
NOM1            DB                  'ABCDEFGHI'
NOM2            DB                  'JKLMNOPQR'
;-------------------------------------------------------------------------
                .CODE
BEGIN           PROC                FAR
                MOV                 AX,@data        ;Inicia registros
                MOV                 DS,AX           ;de segmento
                MOV                 ES,AX

                MOV                 CX,09           ;Iniciación para mover
                                                    ;9 caracteres
                LEA                 SI,NOM1         ;Iniciación de direcciones
                                                    ;para  NOM1 y NOM2
                LEA                 DI,NOM2
B20:
                MOV                 AL,[SI]         ;Obtener carácter de NOM1
                MOV                 [DI], AL        ;Moverlo a NOM2
                INC                 SI              ;Incrementar sig. Carácter en
                                                    ;NOM1
                INC                 DI              ;Incrementar, a sig. posición,
                                                    ;en NOM2
                DEC                 CX              ;Decrementar contador de
                                                    ;iteraciones
                JNZ                 B20             ;¿Contador diferente de
                                                    ;cero? Si, iterar

                MOV                 AX,4C00H        ;Salida al DOS
                INT                 21H
BEGIN           ENDP
                END                 BEGIN
```

7.11.2 Ejemplo de un programa que utiliza JMP

El programa de la figura 7-4 se ilustra el uso de la instrucción JMP. El programa inicializa los registros AX, BX y CX con el valor de 1, y un ciclo realiza lo siguiente:

- Suma 1 a AX
- Suma AX a BX
- Duplica el valor en CX

Al final del ciclo, la instrucción JMP A20 transfiere el control a la instrucción etiquetada con A20. El efecto de repetir el ciclo hace que AX se incremente como 1,2,3,4,...; BX aumente de acuerdo a la suma de los primeros números naturales, obteniéndose 1,3,6,10,...; y CX se duplique como 1,2,4,8,... Ya que este ciclo no tiene salida, el procesamiento es infinito (por lo común no es una buena idea).

Figura 7-4 Uso de la instrucción JMP

```
                Page 60, 132
TITLE           PROG3ASM (COM)      Uso de JMP para iterar
                .MODEL SMALL
                .CODE
                ORG                 100H
MAIN            PROC                NEAR
                MOV                 AX,01           ;Iniciación de AX,
                MOV                 BX,01           ;BX Y
                MOV                 CX,01           ;CX a 01
A20
                ADD                 AX,01           ;Sumar 01 a AX
                ADD                 BX,AX           ;Sumar AX a BX
                SHL                 CX,1            ;Multiplicar por dos a CX
                JMP                 A20             ;Saltar a la etiqueta A20
MAIN            ENDP
                END                 MAIN
```

7.11.3 Ejemplo de un programa que utiliza la instrucción LOOP.

El programa de la figura 7-5 ilustra el uso de LOOP y realiza la misma operación que la del programa del ejemplo 7-3, excepto que termina después de 10 vueltas. Una instrucción MOV inicializa el CX con el valor 10. Como LOOP utiliza el CX, este programa usa ahora DX en lugar de CX para duplicar el valor inicial de 1. La instrucción LOOP reemplaza JMP A20 y, para procesamiento más rápido, INC AX (incrementa el AX en 1) reemplaza ADD AX,01.

Igual que para JMP, el operando del código de máquina contiene la distancia desde el final de la instrucción LOOP a la dirección de A20, la cual es sumada al IP.

Figura 7-5 Uso de la instrucción LOOP

```
                Page 60.132
TITLE           PROG4ASM (COM)   Ilustración de LOOP
                .MODEL SAMALL
                .CODE
                ORG              100H
BEGIN           PROC             NEAR
                MOV              AX,01           ;Iniciar AX,
                MOV              BX,01           ;BX, y
                MOV              DX,01           ;DX con 01
                MOV              CX,10           ;Iniciar número de
A20:                                             ;iteraciones
                INC              AX              ;Sumar 01 a AX
                ADD              BX,AX           ;Sumar AX a BX
                SHL              DX,1            ;Multiplicar por 2 a DX
                LOOP             A20             ;Decrementar CX
                                                 ;iterar si es <> de cero
                MOV              AX,4C00H        ;Salida al DOS
                INT              21H
BEGIN           ENDP
                END              BEGIN
```

7.11.4 Ejemplo de un programa con llamada y regreso cercano.

En la figura 7-6 se muestra un programa con llamadas y regresos cercanos, el cual muestra las siguientes características:

- El programa esta dividido en un procedimiento lejano, BEGIN y dos procedimientos cercanos, B10 y C10. Cada procedimiento tiene un nombre único y contiene su propio ENDP para finalizar su definición.
- Las directivas PROC para B10 y C10 tienen el atributo NEAR para indicar que estos procedimientos están dentro del segmento de código actual.
- En el procedimiento BEGIN, la instrucción CALL transfiere el control del programa al procedimiento B10 e inicia su ejecución.
- En el procedimiento B10, la instrucción CALL transfiere el control al procedimiento C10 e inicia su ejecución.
- En el procedimiento C10, la instrucción RET hace que el control regrese la instrucción que sigue a CALL B10.
- Entonces el procedimiento BEGIN reasume el procedimiento desde ese punto.
- RET siempre regresa la rutina que llama. Si B10 no termina con una instrucción RET, las instrucciones se ejecutarían pasando B10 e irían directamente a C10. De hecho, si C10 no contiene un RET el programa ejecutaría, pasando el final de C10, todas las instrucciones (si hay) que estuvieran ahí, con resultados impredecibles.

Figura 7-6 Programa que realiza llamadas a procedimientos

```
                Page 60,132
TITLE           PROG5ASM (EXE)      Llamada a procedimientos
                .MODEL SMALL
                .STACK 64
                .DATA
;---------------------------------------------------------------------
                .CODE
BEGIN           PROC                FAR
                CALL                B10             ;Llamada a B10
                ...
                MOV                 AX,4C00H        ;Salida al DOS
                INT                 21H
BEGIN           ENDP
;---------------------------------------------------------------------
B10             PROC                NEAR
                CALL                C10             ;Llamada a C10
;               ...
                RET                                 ;De regreso a quien
B10             ENDP                                ;llama
;---------------------------------------------------------------------
C10             PROC NEAR
;               ...
                RET                                 ;De regreso a quien
                ENDP                                ;llama
;---------------------------------------------------------------------
                END                 BEGIN
```

7.11.5 Ejemplo de un programa que acepta y despliega nombres.

El programa de la figura 7-7 pide al usuario que introduzca un nombre y después lo despliega en el centro de la pantalla y emite un sonido de bocina. Por ejemplo, si el usuario introduce el nombre Pat Brown, el programa realiza lo siguiente:

1. Divide la longitud 09 entre dos: 9/2 =4, ignorando la fracción.
2. Resta este resultado de 40: 40-4 =36

En F10CENT, la instrucción SHR corre la longitud 09 un bit a la derecha dividiendo la longitud entre 2. Los bits 00001001 se convierten en 00000100, o 4. La instrucción NEG invierte el signo, cambiando +4 a –4. ADD suma el valor 40, dando en el registro DL la posición inicial de la columna, 36. Con el cursor colocado en el renglón 12, columna 36, el nombre aparece en la pantalla como sigue:

```
Renglón 12: Pat    Brown
              |      |
  Columna: 36      40
```

Observe que la instrucción en E10CODE que inserta el carácter campana (07H) en el área de entrada sigue de manera inmediata al nombre:

```
MOV  BH,00                ;Reemplaza el carácter Enter (0DH)
MOV  BL,NAMELEN           ;con el carácter campana (07H)
MOV  NAMEFLD [BX],07H
```

Los dos primeros MOV establecen el BX con la longitud. El tercer MOV hace referencia a un especificador de índice en corchetes, que significa que el BX actúa como un registro especial de índice para facilitar el direccionamiento extendido. El MOV combina la longitud en el BX con la dirección de NAMEFLD y mueve el 07H a la dirección calculada. Así, para una longitud de 05 la instrucción inserta 07H en NAMEFLD+5 (reemplazando el carácter Enter) a continuación del nombre. La última instrucción en E10CODE inserta un delimitador '$' después del 07H, de manera que la función 09H del DOS pueda desplegar el nombre y sonar la bocina.

El programa continua aceptando y desplegando nombres hasta que el usuario presione sólo la tecla Enter como respuesta a una petición. La función 09H del DOS la acepta e inserta una longitud de 00H en la lista de parámetros.

Si la longitud es cero, el programa determina que la entrada ha finalizado, como lo muestra por la instrucción CMP NAMELEN,00 en A20LOOP.

Figura 7-7 Programa que acepta y muestra nombres

```
                Page 60,132
TITLE           PROG6 (EXE)     Acepta nombres y los centra en la pantalla
;-------------------------------------------------------------------------
                .MODEL SAMALL
                .STACK 64
;-------------------------------------------------------------------------
                .DATA
NAMEPAR         LABEL           BYTE            ;Lista de parámetros
MAXNLEN         DB              20              ;nombre: long. Max. de nombre
NAMELEN         DB              ?               ;# de caracteres introducidos
NAMEFLD         DB              21 DUP(' ')     ;nombre introducido
PROMPT          DB              'Name? ', '$'
;-------------------------------------------------------------------------
                .CODE
BEGIN           PROC            FAR
                MOV             AX,@data        ;Iniciar registros de segmento
                MOV             DS,AX
                MOV             ES,AX
                CALL            Q10CLR          ;Despejar pantalla
A20LOOP:
                MOV             DX,0000         ;Fijar cursor en 00,00
                CALL            Q20CURS
                CALL            B10PRMP         ;Exhibir indicación
                CALL            D10INPT         ;Proporciona entrada / nombre
                CALL            Q10CLR          ;Despejar pantalla
                CMP             NAMELEN,00      ;¿Se ingresó el nombre?
                JE              A30             ;no, salida
                CALL            E10CODE         ;Fijar campana y '$'
                CALL            F10CENT         ;Centra y exhibe el nombre
                JMP             A20LOOP
A30:
                MOV             AX,4C00H        ;Salir a DOS
                INT             21H
BEGIN           ENDP
;               Exhibe indicador:
;               ----------------
B10PRMP         PROC            NEAR
                MOV             AH,09H          ;Petición de exhibición
                LEA             DX,PROMPT
                INT             21H
                RET
B10PRMP         ENDP
;               Acepta nombre
;               -------------
```

```
D10INPT     PROC    NEAR
            MOV     AH,0AH          ;Petición de teclado
            LEA     DX,NAMEPAR      ;entrada
            INT     21H
            RET
D10INPT     ENDP
;           Fijar pantalla y delimitador '$'
;           ------------------------------
E10CODE     PROC    NEAR
            MOV     BH,00           ;Reemplaza carácter Enter
            MOV     BL,NAMELEN      ;(OD) con el de campana (07)
            MOV     NAMEFLD [BX],07
            MOV     NAMEFLD [BX+1],'$' ;Pone el delimitador de
            RET                     ;exhibición
E10CODE     ENDP
;           Centrar y exhibir nombre
;           ------------------------
F10CENT     PROC    NEAR
            MOV     DL,NAMELEN      ;Localiza columna central:
            SHR     DL,1            ;divide longitud en 2,
            NEG     DL              ;invierte el seguro
            ADD     DL,40           ;suma 40
            MOV     DH,12           ;Centra hilera
            CALL    Q20CURS         ;Fija cursor
            MOV     AH,09H
            LEA     DX,NAMEFLD      ;Exhibe nombre
            INT     21H
            RET
F10CENT     ENDP
;           Despejar pantalla
;           -----------------
Q10CLR      PROC    NEAR
            MOV     AX,0600H        ;Petición de recorrido de panta.
            MOV     BH,30           ,Color (07 para blanco y negro)
            MOV     CX,0000         ;De 00,00
            MOV     DX,184FH        ;A 24,79
            INT     10H
            RET
Q10CLR      ENDP
;           Fijar hilera/columna de cursor
;           ------------------------------
Q20CURS     PROC    NEAR            ;DX fija en entrada
            MOV     AH,02H          ;Petición de ubicar cursor
            MOV     BH,00           ;Pagina 0
            INT     10H
            RET
Q20CURS     ENDP

            END     BEGIN
```

[1] En PETER A., ***Prentice Hall***, LENGUAJE ENSAMBLADOR Y PROGRAMACIÓN PARA PC IBM® Y COMPATIBLES, 3a. ed., México, D.F., PRENTICE HALL, 1996, p.74

Conclusiones

Tras el estudio realizado en éste trabajo puedo concluir lo siguiente:

1. El procesador Pentium III es una evolución del Pentium II. No es un procesador radicalmente nuevo, sino que avanza y mejora algunas de las ideas que se emplearon en el Pentium II y en el Celerón.

2. El Pentium III esta diseñado para Internet, incluye extensiones SSE y cada micro tiene un número de serie único.

3. El procesador Pentium III tiene un proceso de 0,25 micras (la cuarta parte de una millonésima de metro), para obtener velocidades de más de 450 MHz y disipar menos calor que la generación anterior de micros.

4. Tanto el procesador Pentium II como el Pentium III son predecesores del procesador Pentium Pro.

5. El Pentium III agrega 70 nuevas instrucciones, lo cual lo hace más poderoso, sin embargo sigue utilizando todas las instrucciones que en los procesadores anteriores se incluyeron.

6. La forma de programación de un Pentium III es muy similar a la de cualquier procesador anterior, la diferencia radica básicamente en las nuevas instrucciones.

Bibliografía:

1. http://developer.Intel.com/design/PentiumIII/manuals

2. Intel Corporation, Intel Architecture Software Developer's, Volume 1: Basic Architecture (Order Number 243190), USA, 1999, p. 369

3. Intel Corporation, Intel Architecture Software Developer's, Volume 2: Instructions Set Reference (Order Number 243191), USA, 1999, p. 854

4. Intel Corporation, Intel Architecture Software Developer's, Volume 3: System Programming Guide (Order Number 243192), USA, 1999, p. 658

5. PETER ALLEN., ***Prentice Hall***, LENGUAJE ENSAMBLADOR Y PROGRAMACIÓN PARA PC IBM® Y COMPATIBLES, 3a. ed., México, D.F., PRENTICE HALL, 1996, p.555

6. ROGER L. Tokheim, ***Mc Graw Hill***, FUNDAMENTOS DE LOS MICROPROCESADORES, 2a. ed., México, D.F., MC GRAW HILL, 1996, p.594

7. TITO KLEIN, *"Todo sobre el Pentium III"* en ***Computer***, año II, núm. XII, México, marzo de 1999, p.96

ANEXO 1

Conjunto de instrucciones

En el siguiente anexo se describen cada una de las instrucciones del Procesador Pentium III, dicha descripción es realizada en su idioma original, el ingles.

Instrucciones de movimiento de datos

INSTRUCCIÓN	DESCRIPCIÓN
MOV	Move
CMOVE/CMOVZ	Conditional move if equal/Conditional move if zero
CMOVNE/CMOVNZ	Conditional move if not equal/Conditional move if not zero
CMOVA/CMOVNBE	Conditional move if above/Conditional move if not below or equal
CMOVAE/CMOVNB	Conditional move if above or equal/Conditional move if not below
CMOVB/CMOVNAE	Conditional move if below/Conditional move if not above or equal
CMOVBE/CMOVNA	Conditional move if below or equal/Conditional move if not above
CMOVG/CMOVNLE	Conditional move if greater/Conditional move if not less or equal
CMOVGE/CMOVNL	Conditional move if greater or equal/Conditional move if not less
CMOVL/CMOVNGE	Conditional move if less/Conditional move if not greater or equal
CMOVLE/CMOVNG	Conditional move if less or equal/Conditional move if not greater
CMOVC	Conditional move if carry
CMOVNC	Conditional move if not carry
CMOVO	Conditional move if overflow
CMOVNO	Conditional move if not overflow
CMOVS	Conditional move if sign (negative)
CMOVNS	Conditional move if not sign (non-negative)
CMOVP/CMOVPE	Conditional move if parity/Conditional move if parity even

CMOVNP/CMOVPO	Conditional move if not parity/Conditional move if parity odd
XCHG	Exchange
BSWAP	Byte swap
XADD	Exchange and add
CMPXCHG	Compare and exchange
CMPXCHG8B	Compare and exchange 8 bytes
PUSH	Push onto stack
POP	Pop off of stack
PUSHA/PUSHAD	Push general-purpose registers onto stack
POPA/POPAD	Pop general-purpose registers from stack
IN	Read from a port
OUT	Write to a port
CWD/CDQ	Convert word to doubleword/Convert doubleword to quadword
CBW/CWDE	Convert byte to word/Convert word to doubleword in EAX register
MOVSX	Move and sign extend
MOVZX	Move and zero extend

Instrucciones de aritmética binaria

INSTRUCCIÓN	DESCRIPCIÓN
ADD	Integer add
ADC	Add with carry
SUB	Subtract
SBB	Subtract with borrow
IMUL	Signed multiply
MUL	Unsigned multiply
IDIV	Signed divide
DIV	Unsigned divide
INC	Increment
DEC	Decrement
NEG	Negate
CMP	Compare

Instrucciones de aritmética decimal

INSTRUCCIÓN	DESCRIPCIÓN
DAA	Decimal adjust after addition
DAS	Decimal adjust after subtraction
AAA	ASCII adjust after addition
AAS	ASCII adjust after subtraction
AAM	ASCII adjust after multiplication
AAD	ASCII adjust before division

Instrucciones lógicas

INSTRUCCIÓN	DESCRIPCIÓN
AND	And
OR	Or
XOR	Exclusive or
NOT	Not

Cambio y traslado de instrucciones

INSTRUCCIÓN	DESCRIPCIÓN
SAR	Shift arithmetic right
SHR	Shift logical right
SAL/SHL	Shift arithmetic left/Shift logical left
SHRD	Shift right double
SHLD	Shift left double
ROR	Rotate right
ROL	Rotate left
RCR	Rotate through carry right
RCL	Rotate through carry left

Instrucciones bit y byte

INSTRUCCIÓN	DESCRIPCIÓN
BT	Bit test
BTS	Bit test and set
BTR	Bit test and reset
BTC	Bit test and complement

BSF	Bit scan forward
BSR	Bit scan reverse
SETE/SETZ	Set byte if equal/Set byte if zero
SETNE/SETNZ	Set byte if not equal/Set byte if not zero
SETA/SETNBE	Set byte if above/Set byte if not below or equal
SETAE/SETNB/SETNC	Set byte if above or equal/Set byte if not below/Set byte if not carry
SETB/SETNAE/SETC	Set byte if below/Set byte if not above or equal/Set byte if carry
SETBE/SETNA	Set byte if below or equal/Set byte if not above
SETG/SETNLE	Set byte if greater/Set byte if not less or equal
SETGE/SETNL	Set byte if greater or equal/Set byte if not less
SETL/SETNGE	Set byte if less/Set byte if not greater or equal
SETLE/SETNG	Set byte if less or equal/Set byte if not greater
SETS	Set byte if sign (negative)
SETNS	Set byte if not sign (non-negative)
SETO	Set byte if overflow
SETNO	Set byte if not overflow
SETPE/SETP	Set byte if parity even/Set byte if parity
SETPO/SETNP	Set byte if parity odd/Set byte if not parity
TEST	Logical compare

Instrucciones del control de transferencia

INSTRUCCIÓN	DESCRIPCIÓN
JMP	Jump
JE/JZ	Jump if equal/Jump if zero
JNE/JNZ	Jump if not equal/Jump if not zero
JA/JNBE	Jump if above/Jump if not below or equal
JAE/JNB	Jump if above or equal/Jump if not below
JB/JNAE	Jump if below/Jump if not above or equal
JBE/JNA	Jump if below or equal/Jump if not above
JG/JNLE	Jump if greater/Jump if not less or equal
JGE/JNL	Jump if greater or equal/Jump if not less
JL/JNGE	Jump if less/Jump if not greater or equal
JLE/JNG	Jump if less or equal/Jump if not greater
JC	Jump if carry
JNC	Jump if not carry
JO	Jump if overflow
JNO	Jump if not overflow

JS	Jump if sign (negative)
JNS	Jump if not sign (non-negative)
JPO/JNP	Jump if parity odd/Jump if not parity
JPE/JP	Jump if parity even/Jump if parity
JCXZ/JECXZ	Jump register CX zero/Jump register ECX zero
LOOP	Loop with ECX counter
LOOPZ/LOOPE	Loop with ECX and zero/Loop with ECX and equal
LOOPNZ/LOOPNE	Loop with ECX and not zero/Loop with ECX and not equal
CALL	Call procedure
RET	Return
IRET	Return from interrupt
INT	Software interrupt
INTO	Interrupt on overflow
BOUND	Detect value out of range
ENTER	High-level procedure entry
LEAVE	High-level procedure exit

Instrucciones cadena

INSTRUCCIÓN	**DESCRIPCIÓN**
MOVS/MOVSB	Move string/Move byte string
MOVS/MOVSW	Move string/Move word string
MOVS/MOVSD	Move string/Move doubleword string
CMPS/CMPSB	Compare string/Compare byte string
CMPS/CMPSW	Compare string/Compare word string
CMPS/CMPSD	Compare string/Compare doubleword string
SCAS/SCASB	Scan string/Scan byte string
SCAS/SCASW	Scan string/Scan word string
SCAS/SCASD	Scan string/Scan doubleword string
LODS/LODSB	Load string/Load byte string
LODS/LODSW	Load string/Load word string
LODS/LODSD	Load string/Load doubleword string
STOS/STOSB	Store string/Store byte string
STOS/STOSW	Store string/Store word string
STOS/STOSD	Store string/Store doubleword string
REP	Repeat while ECX not zero
REPE/REPZ	Repeat while equal/Repeat while zero
REPNE/REPNZ	Repeat while not equal/Repeat while not zero
INS/INSB	Input string from port/Input byte string from port

INS/INSW	Input string from port/Input word string from port
INS/INSD	Input string from port/Input doubleword string from port
OUTS/OUTSB	Output string to port/Output byte string to port
OUTS/OUTSW	Output string to port/Output word string to port
OUTS/OUTSD	Output string to port/Output doubleword string to port

Instrucción para el control de banderas

INSTRUCCIÓN	DESCRIPCIÓN
STC	Set carry flag
CLC	Clear the carry flag
CMC	Complement the carry flag
CLD	Clear the direction flag
STD	Set direction flag
LAHF	Load flags into AH register
SAHF	Store AH register into flags
PUSHF/PUSHFD	Push EFLAGS onto stack
POPF/POPFD	Pop EFLAGS from stack
STI	Set interrupt flag
CLI	Clear the interrupt flag

Instrucciones para los segmentos de registro

INSTRUCCIÓN	DESCRIPCIÓN
LDS	Load far pointer using DS
LES	Load far pointer using ES
LFS	Load far pointer using FS
LGS	Load far pointer using GS
LSS	Load far pointer using SS

Instrucciones diversas

INSTRUCCIÓN	DESCRIPCIÓN
LEA	Load effective address
NOP	No operation
UB2	Undefined instruction
XLAT/XLATB	Table lookup translation
CPUID	Processor Identification

Instrucciones de transferencias de datos MMX

INSTRUCCIÓN	DESCRIPCIÓN
MOVD	Move doubleword
MOVQ	Move quadword

Instrucciones de conversión MMX

INSTRUCCIÓN	DESCRIPCIÓN
PACKSSWB	Pack words into bytes with signed saturation
PACKSSDW	Pack doublewords into words with signed Saturation
PACKUSWB	Pack words into bytes with unsigned saturation
PUNPCKHBW	Unpack high-order bytes from words
PUNPCKHWD	Unpack high-order words from doublewords
PUNPCKHDQ	Unpack high-order doublewords from quadword
PUNPCKLBW	Unpack low-order bytes from words
PUNPCKLWD	Unpack low-order words from doublewords
PUNPCKLDQ	Unpack low-order doublewords from quadword

Instrucciones de empacado de aritmética MMX

INSTRUCCIÓN	DESCRIPCIÓN
PADDB	Add packed bytes
PADDW	Add packed words
PADDD	Add packed doublewords
PADDSB	Add packed bytes with saturation
PADDSW	Add packed words with saturation
PADDUSB	Add packed unsigned bytes with saturation
PADDUSW	Add packed unsigned words with saturation
PSUBB	Subtract packed bytes
PSUBW	Subtract packed words
PSUBD	Subtract packed doublewords
PSUBSB	Subtract packed bytes with saturation
PSUBSW	Subtract packed words with saturation
PSUBUSB	Subtract packed unsigned bytes with saturation
PSUBUSW	Subtract packed unsigned words with saturation
PMULHW	Multiply packed words and store high result
PMULLW	Multiply packed words and store low result
PMADDWD	Multiply and add packed words

Instrucciones de comparación MMX

INSTRUCCIÓN	DESCRIPCIÓN
PCMPEQB	Compare packed bytes for equal
PCMPEQW	Compare packed words for equal
PCMPEQD	Compare packed doublewords for equal
PCMPGTB	Compare packed bytes for greater than
PCMPGTW	Compare packed words for greater than
PCMPGTD	Compare packed doublewords for greater than

Instrucciones de lógica MMX

INSTRUCCIÓN	DESCRIPCIÓN
PAND	Bitwise logical and
PANDN	Bitwise logical and not
POR	Bitwise logical or
PXOR	Bitwise logical exclusive or

Instrucciones de translación y rotación MMX

INSTRUCCIÓN	DESCRIPCIÓN
PSLLW	Shift packed words left logical
PSLLD	Shift packed doublewords left logical
PSLLQ	Shift packed quadword left logical
PSRLW	Shift packed words right logical
PSRLD	Shift packed doublewords right logical
PSRLQ	Shift packed quadword right logical
PSRAW	Shift packed words right arithmetic
PSRAD	Shift packed doublewords right arithmetic

Administración del estado MMX

INSTRUCCIÓN	DESCRIPCIÓN
EMMS	Empty MMX™ state

Transferencias de datos

INSTRUCCIÓN	DESCRIPCIÓN

FLD	Load real
FST	Store real
FSTP	Store real and pop
FILD	Load integer
FIST	Store integer
FISTP	Store integer and pop
FBLD	Load BCD
FBSTP	Store BCD and pop
FXCH	Exchange registers
FCMOVE	Floating-point conditional move if equal
FCMOVNE	Floating-point conditional move if not equal
FCMOVB	Floating-point conditional move if below
FCMOVBE	Floating-point conditional move if below or equal
FCMOVNB	Floating-point conditional move if not below
FCMOVNBE	Floating-point conditional move if not below or equal
FCMOVU	Floating-point conditional move if unordered
FCMOVNU	Floating-point conditional move if not unordered

Aritmética básica

INSTRUCCIÓN	DESCRIPCIÓN
FADD	Add real
FADDP	Add real and pop
FIADD	Add integer
FSUB	Subtract real
FSUBP	Subtract real and pop
FISUB	Subtract integer
FSUBR	Subtract real reverse
FSUBRP	Subtract real reverse and pop
FISUBR	Subtract integer reverse
FMUL	Multiply real
FMULP	Multiply real and pop
FIMUL	Multiply integer
FDIV	Divide real
FDIVP	Divide real and pop
FIDIV	Divide integer
FDIVR	Divide real reverse
FDIVRP	Divide real reverse and pop
FIDIVR	Divide integer reverse

FPREM	Partial remainder
FPREMI	IEEE Partial remainder
FABS	Absolute value
FCHS	Change sign
FRNDINT	Round to integer
FSCALE	Scale by power of two
FSQRT	Square root
FXTRACT	Extract exponent and significand

Comparación

INSTRUCCIÓN	DESCRIPCIÓN
FCOM	Compare real
FCOMP	Compare real and pop
FCOMPP	Compare real and pop twice
FUCOM	Unordered compare real
FUCOMP	Unordered compare real and pop
FUCOMPP	Unordered compare real and pop twice
FICOM	Compare integer
FICOMP	Compare integer and pop
FCOMI	Compare real and set EFLAGS
FUCOMI	Unordered compare real and set EFLAGS
FCOMIP	Compare real; set EFLAGS; and pop
FUCOMIP	Unordered compare real; set EFLAGS; and pop
FTST	Test real
FXAM	Examine real

Trascendental

INSTRUCCIÓN	DESCRIPCIÓN
FSIN	Sine
FCOS	Cosine
FSINCOS	Sine and cosine
FPTAN	Partial tangent
FPATAN	Partial arctangent
F2XM1	2 x -1
FYL2X	$y * \log_2 x$
FYL2XP1	$y * \log_2 (x+1)$

Carga de constantes

INSTRUCCIÓN	DESCRIPCIÓN
FLD1	Load +1.0
FLDZ	Load +0.0
FLDPI	Load pi
FLDL2E	Load $\log_2 e$
FLDLN2	Load $\log_e 2$
FLDL2T	Load $\log_2 10$
FLDLG2	Load $\log_{10} 2$

Control de la unidad de punto flotante (FPU)

INSTRUCCIÓN	DESCRIPCIÓN
FINCSTP	Increment FPU register stack pointer
FDECSTP	Decrement FPU register stack pointer
FFREE	Free floating-point register
FINIT	Initialize FPU after checking error conditions
FNINIT	Initialize FPU without checking error conditions
FCLEX	Clear floating-point exception flags after checking for error conditions
FNCLEX	Clear floating-point exception flags without checking for error conditions
FSTCW	Store FPU control word after checking error conditions
FNSTCW	Store FPU control word without checking error conditions

Instrucciones del sistema

INSTRUCCIÓN	DESCRIPCIÓN
LGDT	Load global descriptor table (GDT) register
SGDT	Store global descriptor table (GDT) register
LLDT	Load local descriptor table (LDT) register
SLDT	Store local descriptor table (LDT) register
LTR	Load task register
STR	Store task register

LIDT	Load interrupt descriptor table (IDT) register
SIDT	Store interrupt descriptor table (IDT) register
MOV	Load and store control registers
LMSW	Load machine status word
SMSW	Store machine status word
CLTS	Clear the task-switched flag
ARPL	Adjust requested privilege level
LAR	Load access rights
LSL	Load segment limit
VERR	Verify segment for reading
VERW	Verify segment for writing
MOV	Load and store debug registers
INVD	Invalidate cache; no writeback
WBINVD	Invalidate cache; with writeback
INVLPG	Invalidate TLB Entry
LOCK (prefix)	Lock Bus
HLT	Halt processor
RSM	Return from system management mode (SSM)
RDMSR	Read model-specific register
WRMSR	Write model-specific register
RDPMC	Read performance monitoring counters
RDTSC	Read time stamp counter
SYSENTER	Fast System Call, transfers to a flat protected mode kernel at CPL=0
SYSEXIT	Fast System Call, transfers to a flat protected mode kernel at CPL=3

Instrucciones de transferencia de datos con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
MOVAPS	Move aligned packed single-precision floating-Point
MOVUPS	Move unaligned packed single-precision floating-Point
MOVHPS	Move unaligned high packed single-precision floating-point
MOVHLPS	Move aligned high packed single-precision floating-point to low packed single-precision floating-point
MOVLPS	Move unaligned low packed single-precision

	floating-point
MOVLHPS	Move aligned low packed single-precision floating-point to high packed single-precision floating-point
MOVMSKPS	Move mask packed single-precision floating-point
MOVSS	Move scalar single-precision floating-point

Instrucciones para la conversión con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
CVTPI2PS	Convert packed 32-bit integer to packed single-precision floating-point
CVTSI2SS	Convert scalar 32-bit integer to scalar single-precision floating-point
CVTPS2PI	Convert packed single-precision floating-point to packed 32-bit integer
CVTTPS2PI	Convert truncate packed single-precision floating-point to packed 32-bit integer
CVTSS2SI	Convert scalar single-precision floating-point to a 32-bit integer
CVTTSS2SI	Convert truncate scalar single-precision floating-point to scalar 32-bit Integer

Instrucciones de empacado de aritmética con extensión SIMD

ADDPS	Add packed single-precision floating-point
SUBPS	Subtract packed single-precision floating-point
ADDSS	Add scalar single-precision floating-point
SUBSS	Subtract scalar single-precision floating-point
MULPS	Multiply packed single-precision floating-point
MULSS	Multiply scalar single-precision floating-point
DIVPS	Divide packed single-precision floating-point
DIVSS	Divide scalar single-precision floating-point
SQRTPS	Square root packed single-precision floating-point
SQRTSS	Square root scalar single-precision floating-point
MAXPS	Maximum packed single-precision floating-point
MAXSS	Maximum scalar single-precision floating-point
MINPS	Minimum packed single-precision floating-point
MINSS	Minimum scalar single-precision floating-point

Instrucciones de comparación con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
CMPPS	Compare packed single-precision floating-point
CMPSS	Compare scalar single-precision floating-point
COMISS	Compare scalar single-precision floating-point ordered and set EFLAGS
UCOMISS	Unordered compare scalar single-precision floating-point ordered and set EFLAGS

Instrucciones de lógica con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
ANDPS	Bit-wise packed logical AND for single-precision floating-point
ANDNPS	Bit-wise packed logical AND NOT for single-precision floating-point
ORPS	Bit-wise packed logical OR for single-precision floating-point
XORPS	Bit-wise packed logical XOR for single-precision floating-point

Instrucciones de shufle de datos con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
SHUFPS	Shuffle packed single-precision floating-point
UNPCKHPS	Unpacked high packed single-precision floating-Point
UNPCKLPS	Unpacked low packed single-precision floating-Point

Instrucciones adicionales entero-SIMD con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
PAVGB/PAVGW	Average unsigned source sub-operands; without incurring a loss in precision
PEXTRW	Extract 16-bit word from MMX™ register
PINSRW	Insert 16-bit word into MMX™ register
PMAXUB/PMAXSW	Maximum of packed unsigned integer bytes or signed integer words

PMINUB/PMINSW	Minimum of packed unsigned integer bytes or signed integer words
PMOVMSKB	Move Byte Mask from MMX™ register
PMULHUW	Unsigned high packed integer word multiply in MMX™ register
PSADBW	Sum of absolute differences
PSHUFW	Shuffle packed integer word in MMX™ register

Instrucciones del control del habilitador de cache con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
PAVGB/PAVGW	Average unsigned source sub-operands; without incurring a loss in precision
PEXTRW	Extract 16-bit word from MMX™ register
PINSRW	Insert 16-bit word into MMX™ register
PMAXUB/PMAXSW	Maximum of packed unsigned integer bytes or signed integer words
PMINUB/PMINSW	Minimum of packed unsigned integer bytes or signed integer words
PMOVMSKB	Move Byte Mask from MMX™ register
PMULHUW	Unsigned high packed integer word multiply in MMX™ register
PSADBW	Sum of absolute differences
PSHUFW	Shuffle packed integer word in MMX™ register

Instrucciones de administración del estado con extensión SIMD

INSTRUCCIÓN	DESCRIPCIÓN
LDMXCSR	Load SIMD Floating-Point Control and Status Register
STMXCSR	Store SIMD Floating-Point Control and Status Register
FXSAVE	Saves floating-point and MMX™ state and SIMD Floating-Point state to memory
FXRSTOR	Loads FP and MMX™ state and SIMD Floating-Point state from memory

ANEXO 2

Directivas del ensamblador

En este anexo se describe la mayoría de las directivas del lenguaje ensamblador. Las directivas están divididas en varias categorías:

- Etiquetas de código: ALIGN, EVEN, LABEL Y PROC.
- Ensamblado condicional: IF, ELSE y otras.
- Errores condicionales.
- Asignación de datos: ALIGN, EQU, EVEN, LABEL, ORG, DB, DB, DW, DD, DF, DQ y DT.
- Control de listado: .CRT, .LIST, .PAGE, SUBTTL, TITLE, XCREF, .XLIST, .LALL, .LFCONT, .SALL, .SFCONT, .TFCOND y .XALL.
- Macros: ENDM, EXITM, LOCAL, MACRO y PURGE.
- Varias: COMMENT, INCLUDE, INCLUDELIB, NAME, &OUT y .RADIX.
- Procesador: .8086, .286, .285P, .386, .386P, .486, .486P, .586, .586P, etcétera.
- Bloques repetidos: IRP, IRPC y REPT.
- Alcance: COMM, EXTRN y PUBLIC.
- Segmento: .ALPHA, ASSUME, .DOSSEG, END, ENDS, GROUP, SEGMENT y .SEQ.
- Segmento simplificado: .CODE, .CONST, .DATA, .DATA?, DOSSEG, .EXIT, .FARDATA, .FARDATA?, .MODEL, .STACK.
- Estructura/registro: ENDS, RECORD, STRUCT, TYPEDEF, UNION.

A continuación se da una breve descripción de algunas de estas directivas.

Directiva ALIGN

MASM 5.0 introduce ésta directiva para forzar al ensamblador alinee el siguiente elemento de datos o instrucción de acuerdo a un valor dado. El formato general es:

ALIGN número

El número debe ser una potencia de 2, tal como 2,4,8 o 16.

Directiva .ALPHA

La directiva .ALPHA, colocada en o cerca del inicio de un programa, le indica al ensamblador que acomode los segmentos en orden alfabético. Pasa por encima de la opción de ensamblador /S.

Directiva ASSUME

ASSUME le indica al ensamblador que asocie nombres de segmento con los registros de segmento CS, DS, ES y SS. El formato general es:

ASSUME reg-seg:nom-reg[...]

Directiva .CODE

Esta directiva simplificada de segmento define el segmento de código. Su formato general es:

.CODE [nombre]

Todo código ejecutable debe ser colocado en éste segmento.

Directiva COMM

Al definir una variable como COMM le da los atributos PUBLIC y EXTRN. De esta manera, no tendría que definir la variable como PUBLIC en un módulo y EXTRN en otro. El formato general es:

COMM [NEAR/FAR] etiqueta:tamaño[:contador]

Directiva COMMENT

Esta directiva es útil para múltiples líneas de comentarios. El formato general es:

COMMENT delimitador [comentarios]
[comentarios]
delimitador [comentarios]

El delimitador es el primer carácter diferente de espacio en blanco que sigue a COMMENT, tal como % o +. El comentario termina en la línea en la que aparece un segundo delimitador.

Directiva .CONST

Esta directiva simplificada de segmento define un segmento de datos (o constante) con la clase 'const'.

Directiva .CREF

Esta directiva (que se tiene por omisión) le indica al ensamblador que genere una tabla de referencias cruzadas. Sería usada a continuación una directiva .XCREF que causa la supresión de la tabla.

Directivas .DATA y .DATA?

Estas directivas simplificadas de segmento definen segmentos de datos. .DATA define un segmento para inicializar datos cercanos; .DATA? define un

segmento para datos cercanos no inicializados, por lo común usada cuando se enlaza con lenguajes de alto nivel.

Directiva DOSSEG

Existen varias maneras de controlar la secuencia en la que el ensamblador acomoda los segmentos. Se pueden codificar las directivas .SEQ o .ALPHA al inicio de un programa, o se pueden ingresar las opciones del ensamblador /S o /A en el momento del ensamblado. La directiva DOSSEG (.DOSSEG desde el MASM 6.x) le indica al ensamblador ignorar todas las demás peticiones y adoptar la secuencia de segmentos DOS –básicamente, código, datos y pila.

Directiva END

La directiva END es colocada al final de un programa fuente. El formato general es:

END [dirección-inicial]

La dirección inicial opcional indica la localidad en el segmento de código (por lo regular la primer instrucción) en donde la ejecución empieza.

Directiva ENDP

Esta directiva indica el final de un procedimiento, definido por PROC. El formato general es:

etiqueta ENDP

La etiqueta es la misma como la que define el procedimiento.

Directiva ENDS

Esta directiva indica el final de un segmento o de una estructura. Su formato general es:

etiqueta ENDS

La etiqueta es la misma como la que define el segmento o estructura.

Directiva EQU

La directiva EQU es usada para definir un nombre de dato o una variable con otro nombre de dato, variable o valor inmediato. La directiva debe ser definida en un programa antes de ser referenciada. Los formatos para datos numéricos y de cadena de caracteres difieren:

Equivalencia numérica: nombre EQU expresión

Equivalencia de cadena: nombre EQU <cadena>

El ensamblador reemplaza cada ocurrencia del nombre con el operando. Ya que EQU es utilizada para reemplazo simple, no ocupa espacio de almacenamiento adicional en el programa objeto generado.

Directiva .ERR

Directiva de error condicional que puede ser utilizada para ayudar a probar errores durante un ensamblado.

Directiva EVEN

EVEN le indica al ensamblador avanzar su contador de localidad si es necesario, de modo que el siguiente dato o etiqueta definido esté alineado con una frontera par de almacenamiento. Esta técnica hace el procesamiento más eficiente en procesadores que accesan 16 o 32 bits a un tiempo.

Directiva EXTRN

La directiva EXTRN informa al ensamblador y al enlazador acerca de las variables de datos y de las etiquetas que el actual ensamblado hace referencia, pero que otro módulo (enlazado al actual) define. El formato general es:

EXTRN nombre:tipo [, ...]

Directiva .FARDATA y .FARDATA?

Estas directivas simplificadas de segmento definen segmentos de datos. .FARDATA define un segmento para datos lejanos inicializados, y .FARDATA? define un segmento para datos lejanos no inicializados.

Directiva GROUP

Un programa puede contener varios segmentos del mismo tipo (código, datos y pila). El objetivo de la directiva GROUP es juntarlos bajo un mismo nombre, de modo que residan dentro de un segmento, por lo regular un segmento de datos. El formato general es:

Nombre GROUP nom-seg [, nom-seg], ...

Directiva INCLUDE

Se pueden tener secciones de código ensamblado o macroinstrucciones que varios programas utilicen. Si es así, se pueden almacenar éstas en archivos de disco separados disponibles para usarse por cualquier programa.

Directiva LABEL

La directiva LABEL permite redefinir el atributo de una variable de datos o de una etiqueta de instrucción. El formato general es:

Nombre LABEL especificador-de-tipo

Directiva .LIST

La directiva .LIST (que es por omisión) causa que el ensamblador liste el programa fuente. Se puede utilizar la directiva .LIST en cualquier parte en un programa fuente ensamblador para discontinuar el listado.

Directiva .MODEL

Esta directiva simplificada de segmento crea segmentos por omisión y los enunciados ASSUME y GROUP necesarios. Su formato general es:

.MODEL modelo-memoria

Los modelos de memoria son:

- TINY (Para programas .COM)
- SMALL
- MEDIUM
- COMPACT
- LARGE
- HUGE

Directiva ORG

El ensamblador utiliza un contador de localidades para llevar la cuenta de su posición relativa en un segmento de datos o de código. La directiva ORG puede ser utilizada para cambiar el contenido del contador de localidades. El formato general es:

ORG expresión

Directiva PAGE

La directiva PAGE al inicio de un programa fuente especifica el número máximo de líneas a lista en una página y el número de caracteres en una línea. Su formato general es:

PAGE [[longitud],ancho]

Directiva PROC

Un procedimiento es un bloque de código que inicia con PROC y termina con ENDP. Un uso común es para una subrutina dentro de un segmento de código.

Directiva del procesador.

Estas directivas definen los procesadores que el ensamblador va a reconocer. La colocación normal de las directivas de procesador es al inicio de un programa fuente, aunque se podrían codificar dentro de un programa en un punto en donde se quiera habilitar o deshabilitar un procesador.

Directiva PUBLIC

El objetivo de la directiva PUBLIC es para informar al ensamblador y al enlazador que los símbolos identificados en un ensamblado serán referenciados por otros módulos enlazados con el actual. El formato general es:

PUBLIC símbolo [,...]

Directiva RECORD

Permite definir patrones de bits. Un objetivo es definir indicadores de conmutación como un bit o como un multibits. El formato general es:

Nom-reg RECORD nom-campo:ancho[=exp] [, ...]

Directiva SEGMENT

Un módulo ensamblado consiste en uno o más segmentos, parte de un segmento o aun partes de varios segmentos. El formato general es:

```
nom-seg  SEGMENT     [alinear] [combinar] ['clase']
          ...
nom-seg  ENDS
```

Directiva .SEQ

Esta directiva (que es por omisión), colocada cerca del inicio de un programa, le indica al ensamblador que deje segmentos en su secuencia origina. Pasa por alto la opción del ensamblador /A.

Directiva STACK

Esta directiva simplificada de segmento define la pila. Su formato general es:

.STACK [tamaño]

Directiva TITLE

La directiva TITLE hace que un titulo de hasta 60 caracteres se imprima en la línea 2 de cada página de un listado fuente en ensamblador. El formato general es:

TITLE texto

GLOSARIO

Apuntador En una hoja de cálculo [spreadsheet], la barra resaltada [highlighted bar] que indica qué celda [cell] está activa actualmente. Cuando usted abre una nueva hoja de cálculo, el apuntador o puntero de celda [cell pointer] siempre aparece ubicado en la celda A1, en la esquina superior izquierda de la hoja de cálculo. También se le conoce como selector de celda [cell selector].

Archivo Colección de datos [data] almacenada en el disco bajo un mismo nombre, de manera que se le presente al usuario como una sola entidad.

Archivo De Procesamiento Por Lotes Archivo [file] de texto ASCII que contiene los comandos del sistema operativo y, posiblemente, otros comandos a los cuales les da apoyo el procesador por lotes [batch processor]. Los comandos del archivo se ejecutan línea por línea, lo mismo que si se hubieran mecanografiado en el indicador del sistema [system prompt].

BCD (Decimal Codificado A Binario) Sistema sencillo para convertir los números decimales a forma binaria [binary], donde cada dígito decimal se convierte a binario y luego se almacena como un carácter único.

Bios Acrónimo de basic input/output system [sistema básico de entrada/salida]. En una computadora personal, el BIOS es un conjunto de instrucciones almacenadas en la memoria de sólo lectura [read-only memory] que le permite al hardware de la computadora, y al sistema operativo, comunicarse con los programas de aplicación y con los dispositivos periféricos [peripheral devices], tales como los discos duros, las impresoras y los adaptadores de vídeo [hard disks, printers y video adapters].

Bit Contracción de BInary digiT [dígito binario]. Un bit es la unidad básica de información del sistema de numeración binario [binary], representando el 0 (para apagado [Off]) o el 1 (para encendido [On])

Bus Ruta [pathway] electrónica por la cual se envían las señales desde una parte de la computadora a otra. Una computadora personal contiene varios buses (entre dispositivos), cada uno de los cuales se usa para un propósito diferente:

Byte Contracción de BinarY digiT Eight [dígito binario ocho]. Grupo de 8 bits, también conocido como octeto. En términos de almacenamiento de la computadora, un byte contiene generalmente un solo carácter, tal como un número, un símbolo, una letra o un espacio en blanco.

Cache Area especial de la memoria, manejada por un controlador de caché [cache controller]. El caché (antememoria) [cache] permite un mejor rendimiento de la computadora, almacenando el contenido de las ubicaciones de la memoria [memory locations] a las cuales se tiene acceso con mayor frecuencia, así como las direcciones [addresses] donde se encuentra dicho contenido. Cuando el procesador [microprocessor] hace referencia a una dirección de memoria [memory address], el caché verifica si tiene almacenada esa dirección; si la tiene, la información se pasa de inmediato al procesador, de modo que no sea necesario recurrir a la memoria de acceso aleatorio [random access memory (RAM)].

Call Comando del lenguaje de procesamiento por lotes [batch language command] del DOS y el OS/2 que hace arrancar la ejecución de un segundo archivo de procesamiento por lotes [batch file] desde el que se está ejecutando. Cuando la ejecución del segundo archivo termina, el control de la ejecución regresa al archivo de procesamiento por lotes original.

Circuito Integrado Abreviado IC, también llamado chip. Pequeño circuito de semiconductores que contiene muchos componentes electrónicos.

Codificación Proceso de codificar la información como intento de impedir el acceso no autorizado a ella. El reverso de este proceso se conoce como decodificación [decryption].

Código Fuente Versión original de un programa, escrita en un lenguaje de programación [programming language] en particular --legible por los humanos--, antes de que sea compilada o interpretada [compiled o interpreted] en una versión que sólo sea legible por la máquina.

Compilador Programa que traduce un lenguaje de alto nivel [high-level language] tal como C o Pascal a un programa de lenguaje de máquina [machine language program].

CPU Parte de cómputo y de control de la computadora. La unidad central de procesamiento [central processing unit (CPU)] en una maxicomputadora [mainframe] puede estar contenida en muchas tarjetas de circuito impreso [printed-circuit boards]; en una minicomputadora [minicomputer], puede estar contenida en varias tarjetas [boards]; en una computadora personal, está contenida en un solo microprocesador extremadamente poderoso.

Debug Programa de servicio de apoyo [utility program] del DOS que usted puede utilizar para ver o cambiar el contenido de los archivos [files] ejecutables.

Desensamblador Programa, generalmente parte de un compilador [compiler] de un lenguaje de computación, que convierte de nuevo un programa de lenguaje de máquina [machine language] al código fuente de lenguaje ensamblador [assembly language source code].

Dirección De Memoria Ubicación exacta en la memoria, que almacena un elemento de datos [data] en particular o una instrucción de un programa.

Direccionamiento Segmentado Esquema de direccionamiento [addressing] utilizado en los procesadores [microprocessors] de Intel, el cual divide el espacio de direccionamiento [address space] en trozos lógicos llamados segmentos [segments].

Para tener acceso a cualquier dirección [address] dada, el programa debe especificar el segmento [segment] así como el desplazamiento [offset] dentro de ese segmento

Direccionar Ubicación exacta en la memoria [memory] o en el disco, donde está almacenada determinada información. Cada byte en la memoria y cada sector en un disco tienen su propia dirección [addresses] exclusiva.

Disipador Dispositivo que elimina gran parte del calor generado por las elevadas frecuencias de trabajo de los procesadores.

Dispositivo Término genérico que se utiliza para describir cualquier periférico [peripheral] o elemento de hardware de una computadora, el cual puede enviar o recibir datos [data].

Encapsulado Empaquetado o la forma externa que tiene un chip.

Encriptación Método matemático para convertir unos datos en números incomprensibles y para devolverlos a su estado original más tarde.

Ensamblador Programa que convierte un programa de lenguaje ensamblador a lenguaje de máquina [assembly language a machine language], de manera que la computadora pueda ejecutar el programa.

Estructura De Datos. Cualquier método de organizar datos [data], el cual los ordena de una manera lógica de modo que puedan ser interpretados o procesados por una aplicación [application]. Esto puede variar desde un registro [record] en un archivo de base de datos [database file] a una matriz en un programa de análisis estadístico.

Firmware Cualquier software almacenado como memoria de sólo lectura [read-only memory] ROM, EPROM, o EEPROM, la cual mantiene su contenido aun después de que la corriente le ha sido cortada. Los sistemas básicos de entrada/salida o BIOS [Basic Input/Output System]) utilizados en las computadoras compatibles con las fabricadas por IBM [IBM-compatible computers] son un ejemplo de firmware.

Interprete Traductor incluido en un lenguaje de programación [programming language], el cual convierte el código fuente [source code] de un programa de alto nivel --línea por línea-- en enunciados [statements] del lenguaje de máquina [machine language].

Interrupción Señal que interrumpe el procesamiento normal, la cual se envía al procesador [microprocessor] y que la genera un dispositivo [device] bajo su control, tal como el reloj del sistema. Una interrupción [interrupt] indica que ocurrió un evento que requiere la atención del procesador, haciendo que el procesador suspenda la ejecución y grabe su actividad actual, para luego saltar a una rutina de servicio de interrupción [interrupt service routine (ISR)]. Esta rutina de servicio [service routine] procesa la interrupción, incluso si ésta fue generada por el reloj del sistema, una pulsación de tecla, o un clic del ratón [mouse]; y cuando termina, retorna el control al proceso suspendido.

Lenguaje Ensamblador	Lenguaje de bajo nivel [low-level language] en el que cada enunciado del programa [program statement] debe corresponder a una instrucción única de lenguaje de máquina [machine language] que el procesador [microprocessor] pueda ejecutar
Lenguaje Máquina	Lenguaje binario nativo [native binary language] (el cual consta de ceros y unos), utilizado internamente por la computadora; también conocido como código de máquina [machine code]. El lenguaje de máquina [machine language] es difícil de leer y entender por los humanos.
Memoria Virtual	Técnica de administración o gestión de la memoria [memory-management] que permite que la información en la memoria física [physical memory] sea intercambiada con un disco duro [hard disk]. Esta técnica proporciona a los programas de aplicación [application] más espacio de memoria de la que en realidad está disponible en la computadora.
Microcode	Instrucciones de muy bajo nivel [low-level] que definen cómo funciona un procesador [microprocessor] en particular, especificando lo que éste debe hacer cuando ejecuta una instrucción de lenguaje de máquina [machine-language].
Microprocesador	Abreviado processor [procesador]. Unidad central de procesamiento [central processing unit (CPU)] en un solo chip.
MMX	De las siglas en ingles "Multimedia eXtensions", es decir, instrucciones especiales que aceleran los procesos multimedia.
Mnemónico	Nombre o abreviatura utilizada para ayudar a recordar una instrucción larga o muy compleja. Los lenguajes de programación utilizan muchos mnemotécnicos [mnemonics] diferentes para representar las instrucciones complejas.
Modo Protegido	En los procesadores 80286 (o más avanzados) de Intel, estado de operación que da apoyo a características avanzadas. El modo protegido [protected mode] en estos procesadores [microprocessors] da apoyo al hardware para la gestión o administración de tareas múltiples y de la memoria virtual [multitasking y virtual memory], y evita que los programas tengan acceso a los bloques de memoria que pertenecen a los otros programas que están en ejecución.
Modo Real	Estado de operación [operating state] al que dan apoyo [supported] todos los procesadores [microprocessors] de la familia 80x86 de Intel, y el único modo de operación al cual da apoyo el DOS. En el modo real [real mode], el procesador puede direccionar 1 Megabyte (1024 K) de memoria directamente.
Modo virtual 8086	Modalidad de operación que se encuentra en los procesadores [microprocessors] 80386 (y más avanzados), la cual permite al procesador emular simultáneamente varios entornos [environments] separados 8086.
Multiprocesamiento	Habilidad que posee un sistema operativo [operating system] para utilizar más de una unidad central de procesamiento [central processing unit (CPU)] en una misma computadora.
Pagina De Códigos.	En el OS/2, y en el DOS a partir de la Versión 3, conjunto de 256 caracteres que el teclado puede generar, los que la pantalla puede mostrar, y los que la impresora puede imprimir. Solamente una página de códigos [code page] puede estar activa a la vez. Existen dos tipos de página de códigos [code page]: las páginas de códigos para hardware, incorporadas a los chips de ROM en su computadora y las páginas de códigos para software, también conocidas como páginas de códigos preparadas [prepared code pages], las cuales se pueden cargar en la memoria para anular [override] los valores iniciales por definición [defaults] del hardware.
Paridad.	Forma simple de verificación de errores [error checking], la cual utiliza un bit adicional o bit redundante [extra bit o redundant bit] a continuación de los bits de datos [data bits], pero antes del bit o bits de parada [stop bit(s)].

Pila Area reservada de la memoria [memory] utilizada para seguir el rastro a las operaciones internas de un programa, incluyendo las funciones de direcciones de los remitentes [return addresses], los parámetros pasados [passed parameters], y así sucesivamente. Por lo general, la pila [stack] se mantiene como una estructura de los datos [data structure] del tipo último en entrar, primero en salir [last in, first out (LIFO)], de modo que el último elemento en ingresar a la estructura sea el primero que se utiliza.

Pipelining
1. En la arquitectura de procesadores [processor architecture], método de ir a por y de decodificar instrucciones, el cual garantiza que el procesador nunca tiene que esperar, pues tan pronto como una instrucción se ejecuta, otra está en espera.
2. En el procesamiento en paralelo [parallel processing], método utilizado para pasar instrucciones de una unidad de procesamiento [processing unit] a otra.

Procesador De Punto Flotante. Procesador secundario, de propósito especial, diseñado para realizar los cálculos de punto flotante [floating-point calculations] mucho más rápidamente que el procesador principal [main processor]. Muchos procesadores, tal como el 80386 también tienen un procesador de punto flotante [floating-point processor] que funciona concertadamente con él; en este caso, el 80387. Sin embargo, la tendencia modernista para el diseño de procesadores consiste en integrar la unidad de punto flotante al procesador principal, como sucede con los 80486 y Pentium.

Puerto Una conexión física, tal como un puerto serial o un puerto paralelo [serial port o parallel port].

Reubicar un programa o un sistema operativo [operating system] desde una plataforma de hardware a otra.

Puerto Paralelo Puerto de entrada y (o) salida de datos [input/output (I/O)] que maneja información a razón de 8 bits a la vez; se utiliza frecuentemente para conectar una impresora en paralelo [parallel printer].

Puerto Serial Puerto [port] de entrada/salida (de datos) de una computadora, el cual da apoyo a la comunicación en serie [serial communications], en la que la información se procesa a razón de un bit a la vez.

Punto Flotante Sistema que permite trabajar con toda la precisión necesaria sin emplear inmensas cantidades de memoria para almacenar tantos decimales.

Ram Acrónimo de random access memory [memoria de acceso aleatorio]. Memoria del sistema principal [system memory] de la computadora, utilizada por el sistema operativo [operating system], los programas de aplicación y los datos [application, data].

Registro Los registros del procesador se emplean para controlar instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética. Los registros son direccionables por un nombre.

Rom Acrónimo de read-only memory [memoria de sólo lectura]. Sistema de memoria [memory] basado en un semiconductor, el cual almacena permanentemente la información, reteniendo su contenido aún después de que la corriente se desconecta. Las memorias de sólo lectura [read-only memory (ROMs)] se utilizan en el firmware tal como el BIOS (Basic Input/Output System) de las computadoras personales.

SIMD De las siglas en ingles "Single Instruction-Multiple Data", esto es, que una única instrucción puede operar varios datos distintos al mismo tiempo.

SIMM Chips de RAM [RAM chips] individuales, soldados o montados en pequeñas y estrechas tarjetas de circuito impreso [circuit boards], llamadas módulos portadores [carrier modules], los cuales se conectan en zócates [sockets] en la tarjeta (circuito impreso) madre [motherboard].

Slot El zócalo en el cual se conecta un procesador en formato SECC

SSE Estas siglas quieren decir "Streaming SIMD", es decir, extensiones SIMD de flujo, o que fluyen.

Tareas Múltiples	Ejecución simultánea de dos o más programas en una computadora.
Tarjeta	Tarjeta de circuito impreso o adaptador [printed circuit board o adapter] que usted enchufa en una ranura de expansión [expansion slot] de una computadora para dar apoyo a un tipo específico de hardware que normalmente no se encuentra en la computadora.
Tarjeta De Circuito Impreso	Abreviado PCB. Cualquier tarjeta plana, hecha de plástico o de fibra de vidrio, que contiene chips y otros componentes electrónicos. Muchas tarjetas de circuito impreso [printed-circuit boards (PCBs)] son tarjetas de capas múltiples [multi-layer boards] con varios conjuntos diferentes de pistas de cobre [copper traces] que conectan a los componentes.
Transistor	Componente semiconductor que actúa como un interruptor [switch], para controlar el flujo de una corriente eléctrica. Los transistores [transistors] han sido incorporados a los modernos microprocesadores [microprocessors] en cantidades industriales (por millones).