ARQUITECTURA DE COMPUTADORES

LAS SOLUCIONES INTEL
* 80086 - 80186 - 80286 - 80386 - 80486
* PENTIUM I - PRO - II - III - IV
* ITANIUM

JOSU BILBAO EGUIA

INDICE del TEMARIO

*** Capítulo 24: Pentium 4.**

*** Capítulo 25: Itanium.**

*** Anexo A: Diseño de procesadores RISC.**

*** Anexo B: Ejercicios.**

ARQUITECTURA DE COMPUTADORES: INTRODUCCIÓN Y CLASIFICACIÓN

1.1- INTRODUCCIÓN A LOS COMPUTADORES.

Las computadoras no han nacido hace poco, en realidad el hombre siempre buscó tener dispositivos que le ayudaran a efectuar cálculos precisos y rápidos; una breve reseña histórica nos permitirá, comprender cómo llegamos a las computadoras actuales.

1.1.1- Un poco de historia: Las cinco generaciones

Aunque los antecedentes del computador se remontan al ábaco (que en su presente forma, fue introducido en China sobre el 1200 d. C.), se puede afirmar que no existieron los computadores, tal como hoy se les considera, hasta la aparición de la Electrónica a mediados del siglo XX y desde ese momento se han sucedido cinco generaciones.

1.1.1.1- La primera generación (1938 – 1952)

Esta generación ocupó la década de los cincuenta. Y se conoce como la primera generación. Estas máquinas tenían las siguientes características:

- Estas máquinas estaban construidas por medio de tubos de vacío.
- Eran programadas en lenguaje de máquina.

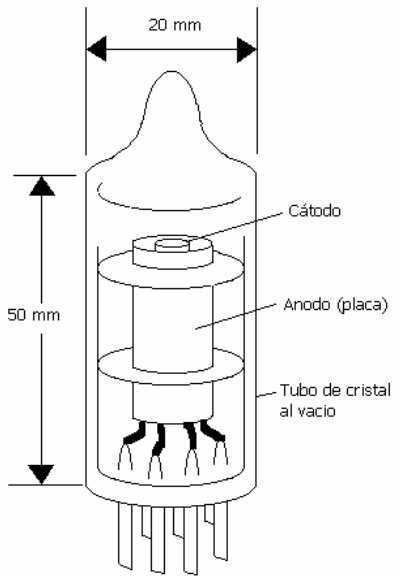


Figura 1.1 – Dibujo simplificado de un tubo de vacío.

En esta generación las máquinas son grandes y costosas, de elevado consumo, gran disipación de calor y una limitada vida de funcionamiento.

En 1.946 aparece el primer computador fabricado con Electrónica Digital, el computador ENIAC. Éste soportaba una estructura de 20 registros de 10 dígitos, tenía tres tipos de tablas de funciones y la entrada y salida de datos y resultados se realizaban mediante tarjetas perforadas.

Tenía unos 18.000 tubos de vacío, pesaba 30 toneladas, ocupaba 1.500 pies cuadrados y realizaba 5.000 sumas por segundo.

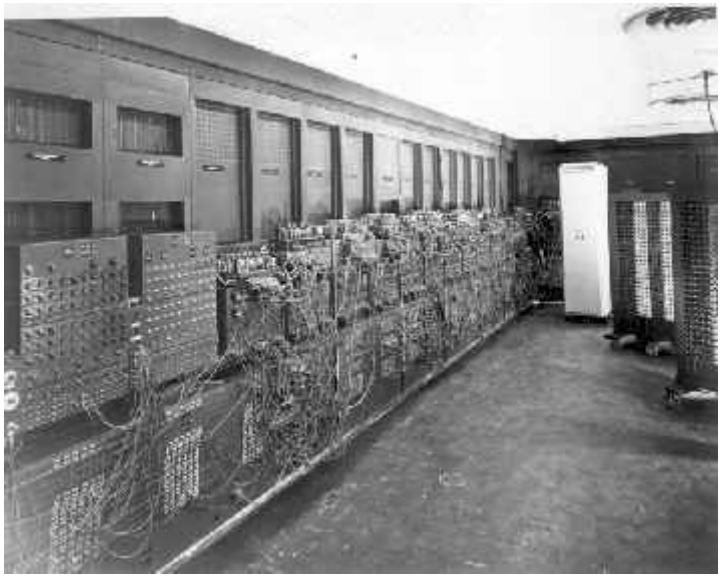


Figura 1.2 – ENIAC

Años posteriores el científico matemático Von Neumann propuso modificar el ENIAC en dos importantes aspectos, que dieron lugar al EDVAC. Estos aspectos fueron el programa almacenado en sustitución del programa cableado y la utilización de la aritmética binaria codificada en lugar de la decimal.

En 1951 aparece la primera computadora comercial: la UNIVAC I. Esta máquina, que disponía de mil palabras de memoria central y podía leer cintas magnéticas, fué usada para procesar los datos del censo de 1950 en los Estados Unidos.

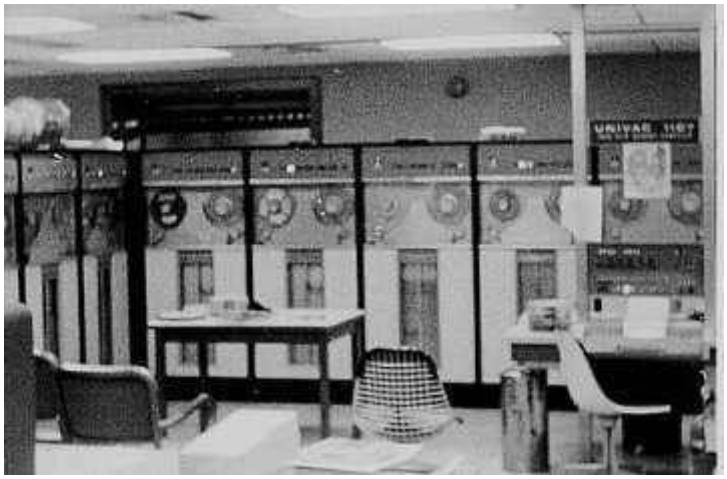


Figura 1.3 **–** 1951 UNIVAC (Universal Automatic Computer) de John Mauchly y J. Presper Eckert

Además del UNIVAC I se considera modelo representativo de esta época el IBM 701, al que siguieron el 704 y 709.

1.1.1.2- La segunda generación (1953 – 1962)

La tecnología de esta generación esta caracterizada por el descubrimiento del transistor, que se utilizan en la construcción de las Unidades Centrales de Proceso. El transistor, al ser más pequeño, más barato y de menor consumo que la válvula, hizo a los computadores mas asequibles en tamaño y precio. Las memorias empezaron a construirse con núcleos de ferrita.

Cerca de la década de 1960, las computadoras seguían evolucionando. Se reduce su tamaño y crece su capacidad de procesamiento. También en esta época se empezó a definir la forma de comunicarse con las computadoras, que se denominó programación de sistemas.

Las características de la segunda generación son las siguientes:

- Están construidas con circuitos de transistores.
- Se programan en nuevos lenguajes llamados lenguajes de alto nivel.:Aparecen lenguajes como el FORTRAN (FORmula TRANslation), ALGOL (ALGOrithimic Language) y COBOL (COmmon Business Oriented Language).

Algunas de estas computadoras se programaban con cintas perforadas y otras más por medio de cableado en un tablero. Los programas eran hechos a la medida por un equipo de expertos: analistas, diseñadores, programadores y operadores que se manejaban como una orquesta para resolver los problemas y cálculos solicitados por la administración.

Las computadoras de esta generación fueron: la Philco 212 (esta compañía se retiró del mercado en 1964), la UNIVAC M460, la Control Data Corporation modelo 1604, seguida por la serie 3000, IBM mejoró la 709 y sacó al mercado la 7090, National Cash Register empezó a producir máquinas para proceso de datos de tipo comercial introduciendo el modelo NCR 315.

1.1.1.3- La tercera generación (1963 – 1971)

Con los progresos de la electrónica y los avances de comunicación con las computadoras en la década de los 1960, surge la tercera generación de las computadoras. Se inaugura con la IBM 360 en abril de 1964.

Las características de esta generación fueron las siguientes:

- Su fabricación electrónica esta basada en circuitos integrados.
- Su manejo se realiza mediante los lenguajes de control de los sistemas operativos.

IBM fabrica la serie 360 con los modelos 20, 22, 30, 40, 50, 65, 67, 75, 85, 90, 195 que utilizaban técnicas especiales del procesador, unidades de cinta de nueve canales, paquetes de discos magnéticos y otras características que ahora son estándares (no todos los modelos usaban estas técnicas, sino que estaba dividido por aplicaciones).

El sistema operativo de la serie 360, se llamó OS que contaba con varias configuraciones, incluía un conjunto de técnicas de manejo de memoria y del procesador que pronto se convirtieron en estándares.

A finales de esta década IBM de su serie 370 produce los modelos 3031, 3033, 4341. Burroughs con su serie 6000 produce los modelos 6500 y 6700 de avanzado diseño, que se reemplazaron por su serie 7000. Honey-Well participa con su computadora DPS con varios modelos.

1.1.1.4- La cuarta generación (1972 – 1987)

Los constantes progresos en el incremento de la densidad de integración nos llevan a al tecnología LSI (Alta Escala de Integración) y posteriormente a la VLSI (Muy Alta Escala de Integración). Lo que permitió la comercialización de circuitos integrados de memoria conteniendo 1 Gbits, con tiempo de acceso de 35 ns.

Aparecen los microprocesadore***s*** que son circuitos integrados de alta densidad y con una velocidad impresionante. Las microcomputadoras basadas en estos circuitos son extremadamente pequeñas y baratas, por lo que su uso se extiende al mercado industrial. Surgen las computadoras personales que han adquirido proporciones enormes y que han influido en la sociedad en general sobre la llamada "revolución informática".

En 1976 Steve Wozniak y Steve Jobs diseñan la primera microcomputadora de uso masivo y más tarde forman la compañía conocida como Apple que fué la segunda compañía más grande del mundo, detrás de IBM.

Figura 1.4 – Computador Apple II

En 1981 se vendieron 80.000 computadores personales, al siguiente subió a 1.400.000. Entre 1984 y 1987 se vendieron alrededor de 60 millones de computadores personales, por lo que no queda duda que su impacto y penetración han sido enormes.

Con el surgimiento de los computadores personales, el software y los sistemas que con ellas de manejan han tenido un considerable avance, porque han hecho más interactiva la comunicación con el usuario. Surgen otras aplicaciones como los procesadores de texto, las hojas electrónicas de cálculo, paquetes gráficos, etc. También las industrias del software de las computadoras personales crece con gran rapidez, Gary Kildall y William Gates se dedicaron durante años a la creación de sistemas operativos y métodos para lograr una utilización sencilla de las microcomputadoras (son los creadores de CP/M y de los productos de Microsoft).

No todo son microcomputadoras, por su puesto, las minicomputadoras y los grandes sistemas continúan en desarrollo. De hecho las máquinas pequeñas rebasaban por mucho la capacidad de los grandes sistemas de 10 o 15 años antes, que requerían de instalaciones costosas y especiales. Sin embargo, los grandes computadores no desaparecen; por el contrario, su presencia es imprecindible en prácticamente todas las esferas de control gubernamental, militar y de la gran industria. Las enormes computadoras de las series CDC, CRAY, Hitachi o IBM por ejemplo, son capaces de atender a varios cientos de millones de operaciones por segundo.

1.1.1.5- La quinta generación (1987 – …..)

En vista de la acelerada marcha de la Microelectrónica, la sociedad industrial ha procurado poner a esa altura el desarrollo del software y los sistemas con que se manejan las computadoras. Surge la competencia internacional por el dominio del mercado de la computación, en la que se perfilan dos líderes que, sin embargo, no han podido alcanzar el nivel que se desea: la capacidad de comunicarse con la computadora en un lenguaje más cotidiano y no a través de códigos o lenguajes de control especializados.

Japón lanzó en 1983 el llamado "programa de la quinta generación de computadoras", con los objetivos explícitos de producir máquinas con innovaciones reales en los criterios mencionados. Y en los Estados Unidos ya está en actividad un programa en desarrollo que persigue objetivos semejantes, que pueden resumirse de la siguiente manera:

- Procesamiento en paralelo mediante arquitecturas y diseños especiales y circuitos de gran velocidad.
- Manejo de lenguaje natural y sistemas de inteligencia artificial.

El futuro previsible de la computación es muy interesante, y se puede esperar que esta ciencia siga siendo objeto de atención prioritaria de gobiernos y de la sociedad en conjunto.

En los microprocesadores actuales para aumentar la velocidad de procesamiento se utilizan técnicas de segmentación y paralelización En la segmentación se descompone la ejecución de las instrucciones máquina en pasos aislados. Con esto se consigue reducir los tiempos medios de ciclo y se consigue ejecutar varias instrucciones en paralelo. Los computadores superescalares utilizan microprocesadores que pueden ejecutar varias instrucciones en paralelo.

También en la actualidad se han extendido bastante los computadores paralelos de memoria distribuida formados por un conjunto de procesadores con memoria local conectados por una rápida red de interconexión que cooperan entre sí para resolver la misma tarea. Al principio estos computadores paralelos llevaban procesadores especializados pero el enorme tiempo de diseño y depuración de estos procesadores hace que la relación coste rendimiento disminuya si se diseñan con microprocesadores comerciales. Ejemplos son el CRAY 3TX y el IBM SP2.

Por otro lado la idea de computador vectorial no se ha abandonado, sólo que se tiende a utilizar tecnología CMOS en lugar de la ECL. También se utilizan los microprocesadores como elemento básico para el diseño de computadores paralelos de memoria compartida. Los microprocesadores se conectan a la memoria por medio de un bus como en el caso de las arquitecturas SG Power Challenge, Sun sparcserver, HP DEC8000. El número de procesadores de estos computadores suele ser inferior a 20.

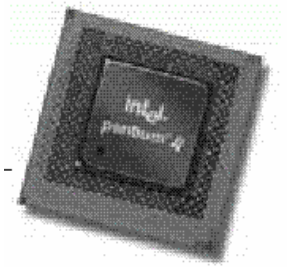


Figura 1.5. Foto de un Pentium IV a 2GHz.

1.1.2- Clasificación de los computadores.

Existen diversos métodos de clasificación de los computadores según la tecnología de su fabricación, de las aplicaciones y de otras diversas circunstancias, etc.

1.1.2.1- Clasificación según su funcionamiento.

- **Computador digital**: recibe este nombre porque procesa datos cuya representación responde a valores discretos como 0, 1, 2, etc., operando con ellos en etapas diversas y sucesivas.

- **Computador analógico**: tienen semejanza con instrumentos de medida tales como amperímetro, voltímetro, termómetro, etc.; los cuales controlan la medida de una magnitud de forma continua.

- **Computador híbrido**: posee características de los dos anteriores, habitualmente, los cálculos los realizan de forma analógica mientras que la entrada y salida de datos se hace de modo digital. La utilización del computador híbrido es frecuente en el control de procesos industriales, en ellos las funciones principales son controladas por un computador digital, mientras que la entrada de datos y salida de resultados se hace empleando computadores analógicos conectados con el digital.

1.1.2.2- Clasificación según la finalidad.

- **De propósito general:** cuando están dedicados a resolver cualquier tipo de aplicación dependiendo del programa que se utilice, como por ejemplo los computadores de las grandes empresas.

- **De propósito especial:** cuando únicamente pueden resolver un tipo concreto y particular de problemas como el computador de un coche o de una lavadora.

1.1.2.3- Clasificación comercial.

Habitualmente se han dividido en cuatro tipos en los cuales, a mayor tamaño, mayor coste, complejidad, potencia y necesidad de mantenimiento.

- Los **Supercomputadores o maxicomputadores** son las máquinas más potentes, complejas, grandes y costosas. Son utilizados por científicos y técnicos para resolver problemas complejos, como por ejemplo, serían los utilizados para la previsión del tiempo, desarrollos económicos a escala mundial, para estudios de medio ambiente, etc.. Un ejemplo de supercomputador es el "CRAY 2".

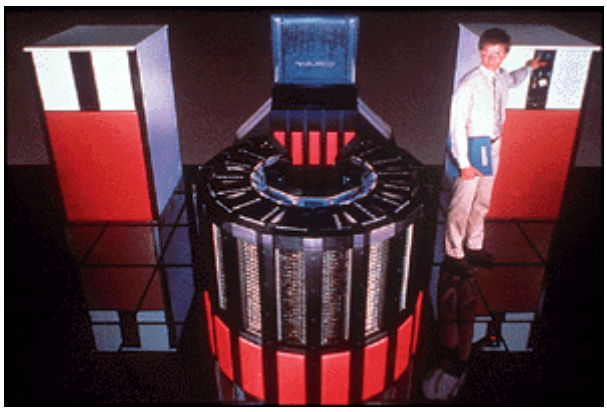


Figura 1.6. Supercomputador CRAY 2. Este computador tenia la memoria central más grande del mundo en 1985 con una capacidad de 2048 megabytes.

- Los **Mainframes o Computadores Grandes** tienen una alta velocidad de proceso y capacidad de memoria.. Se destinan a operaciones diarias en las grandes empresas u organizaciones como son la gestión de cuentas en bancos, facturaciones de fábricas. Podemos citar como ejemplo IBM 9370 Y 9000. Estos computadores suelen ser muy caros.

- Los **Minicomputadores** son máquinas de tamaño medio. Su costo es más reducido. Son máquinas con una capacidad de proceso y de memoria bastante elevados. Un ejemplo de minicomputador es el Vax 11/780.

- **Microcomputadores** : que son los de menor tamaño, capacidad de proceso, memoria y los más baratos. Se utilizan tanto en empresas como a nivel doméstico.

Podemos concluir diciendo que el avance de la tecnología se introdujo inicialmente en los supercomputadores, incrementándose así su rendimiento en relación con los demás tipos, pero ha medida que ha ido pasando el tiempo, los computadores de gamas más bajas han incrementado su rendimiento incluso más rápidamente. Así, los microcomputadores nos ofrecen el crecimiento de rendimiento más rápido.

En la figura 1.7 se muestran los diferentes crecimientos del rendimiento de los diferentes tipos de ordenadores.

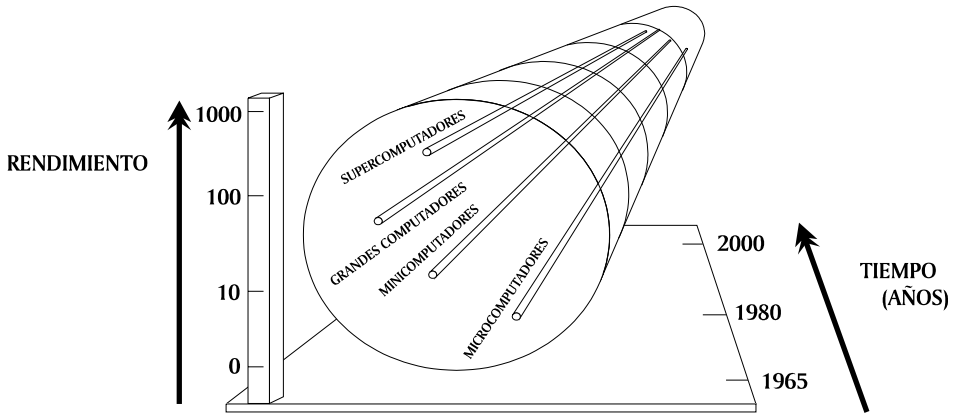


Figura 1.7. Crecimiento del rendimiento de los diferentes computadores

Cuando los supercomputadores han incrementado su rendimiento en un 8% anual, los computadores grandes lo han conseguido en un 15%, los minicomputadores en un 25% y los microcomputadores han alcanza cotas de hasta un 35%. Esta información permite deducir que en un futuro no muy lejano, los diferentes tipos de computadores se unirán en uno sólo.

1.2-FUNDAMENTOS DE LA ARQUITECTURA DE COMPUTADORES.

1.2.1- La arquitectura y el arquitecto de computadores

Actualmente el uso de computadoras esta muy extendido en nuestras actividades cotidianas, nos proporcionan mayor rapidez en nuestras tareas así como la posibilidad de manejar gran cantidad de datos. Así podemos percibir como de un tiempo a esta parte las computadoras ganan terreno en las actividades humanas, su capacidad de cálculo, de procesar datos, generar información e incluso simular procesos las convierten en herramientas indispensables únicamente limitadas por su propia capacidad.

Debemos tener en cuenta que el uso de computadores esta muy extendido en actividades que requieren el manejo de gran cantidad de datos a una gran velocidad como, por ejemplo, en diagnósticos médicos, investigaciones nucleares y de energía, inteligencia artificial, etc... Es en estos casos, que demandan mayor capacidad y rapidez, cuando se sobrepasan las características de los computadores actuales. La solución pasa entonces por la construcción de mejores computadores que logren avances substanciales en su rendimiento.

Para lograr un aumento en el rendimiento se necesita mejorar la "arquitectura" de los computadores y desarrollar nuevas técnicas de procesamiento. El concepto de "arquitectura de un computador" se refiere a la integración de su estructura física con su estructura lógica. Se utiliza el término "arquitectura" para enfatizar la síntesis de elementos de ingeniería y ciencias exactas con elementos estéticos y de funcionalidad práctica, de la misma manera en que un arquitecto combinará las técnicas y conocimientos de la Ingeniería con la apreciación artística e integración de su obra con su entorno.

Esta "arquitectura de computadores" abarca las siguientes fases:

- Definición de las necesidades que pretende cubrir el computador.
- Planificación general del computador.
- Diseño del computador y sus componentes.
- Análisis del sistema obtenido.
- Especificación del sistema: características del sistema, de sus componentes y de las instrucciones ejecutables.

Así el arquitecto de computadores deberá tener un gran conocimiento del equipo físico y del equipo lógico, para poder obtener un buen rendimiento de la máquina, el cual depende principalmente del:

- Lenguaje de programación.
- Compilador.
- Sistema operativo.

- Arquitectura de la máquina.

Cada una de estas áreas dependerá de sí misma y de las siguientes, por lo que hay una gran incidencia en el buen desarrollo de unas sobre las otras. Así, el arquitecto tendrá que realizar las siguientes funciones:

1. **Seleccionar y organizar el hardware**: Implementación, estructura e interconexión de la CPU, subsistema de memoria, subsistema de entradas o salidas y redes de conexión.
2. **Seleccionar y estructurar el software:** Diseño del repertorio de instrucciones a nivel del lenguaje máquina, sistema operativo y compiladores.
3. **Elegir el lenguaje de programación** de alto nivel que mayor rendimiento pueda obtener del sistema diseñado.

Con todo lo expuesto hasta ahora podemos decir que para obtener el máximo provecho del computador, tan importante es conocer su estructura lógica como la física sólo de esta manera podremos obtener las máximas prestaciones de la estructura física aprovechando los recursos de la lógica.

1.2.2- Problemática en la arquitectura de computadores

El desarrollo de los computadores y en concreto de la Informática, está ligada al desarrollo de la electrónica. El avance de las tecnologías y el uso de los computadores ha conseguido que su diseño pase de un arte, en los primeros computadores, a una disciplina de Ingeniería que plantea gran dificultad, pero se basa en una metodología.

Al principio, las dos causas principales de la problemática en la arquitectura de computadores fueron:

1. La independencia entre el hardware y el software, y la falta de definición de las funciones de cada uno de ellos. Hasta hace pocos años, los arquitectos de computadores procedían del campo de la Ingeniería Electrónica, y potenciaban el equipo físico aplicando los constantes avances de la Microelectrónica, sin tener en consideración las prestaciones del sistema lógico.
2. El seguimiento a ultranza de la arquitectura de Von Neumann, no diseñada para soportar los nuevos sistemas operativos, lenguajes y aplicaciones.

Pero hoy en día, los inconvenientes ante los que nos enfrentamos son entre otros:

- **El factor tiempo**. En el diseño del equipo físico, el tiempo es un factor de gran relevancia. Se cita como ejemplo que cada tres años por el mismo precio y calidad obtenemos el doble del número de transistores contenidos en un chip, frecuencia de trabajo y densidad de discos magnéticos y la cuatriplicidad de la densidad de la memoria RAM.
  La miniaturización es la constante en la evolución de los dispositivos electrónicos.

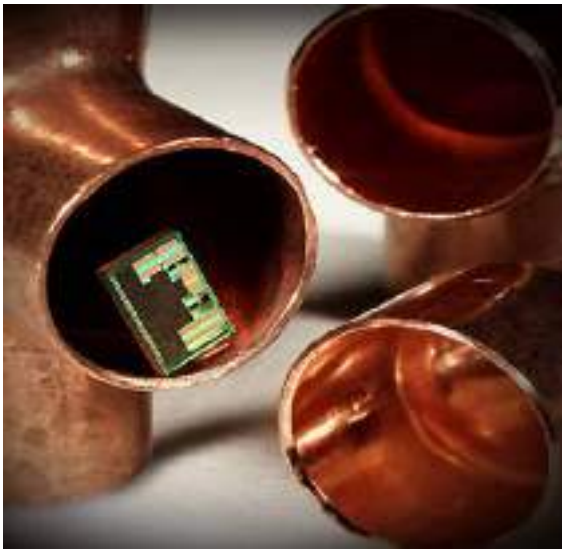


Figura 1.9 Microcomputador de IBM.

  Este fenómeno de desarrollo acelerado conduce a la ley de Moore (1964), que asegura que el número de transistores que incorpora un circuito integrado se multiplica por cuatro cada aproximadamente año y medio. Se piensa que esta ley, vigente durante 40 años, todavía tiene por lo menos otros 10 ó 20 años de vida. Los últimos chips fabricados tienen más de 1.000 millones de transistores en poco más de un centímetro cuadrado.

- **El alcance de límites difícilmente superables de la tecnología hardware**. La búsqueda de mayor rendimiento, se basará en nuevas arquitecturas que exploten en mayor grado las posibilidades del hardware. Un ejemplo es la utilización de computadores paralelos.
- **Límite de coste y ventas.** El aprovechamiento de la compatibilidad con equipos anteriores disminuye la potencia en los nuevos equipos físicos. Un caso es la arquitectura de la familia de los procesadores 80x86 de Intel, en la que los nuevos modelos son compatibles con los anteriores. El rendimiento conseguido es mucho menor que aprovechando todos los recursos del nuevo sistema (Ejemplo: Pentium trabajando con un sistema operativo MS-DOS de 16 bits), pero el decremento de rendimiento se ve subsanado por los bajos costes de los equipos lógicos.

En conclusión, el alto nivel de conocimiento requerido del equipo físico y lógico, el factor tiempo, la tecnología hardware insuperable y las limitaciones de coste-ventas, hacen que la labor del arquitecto de computadores adquiera un alto grado de complejidad.

1.3- ARQUITECTURA CLÁSICA Y MODERNA.

1.3.1- Arquitectura clásica (1950 - 1990).

La arquitectura de un computador es la que fué definida por uno de los mejores matemáticos de la historia John Von Neumann, que propuso es una arquitectura en la cual la CPU (Unidad Central de proceso) está conectada a una única memoria donde se guardan conjuntamente instrucciones (programas) y datos (con los cuales operan estos programas). Además existe un módulo de entradas y salidas para permitir la comunicación de la máquina con los periféricos externos que maneja el usuario.

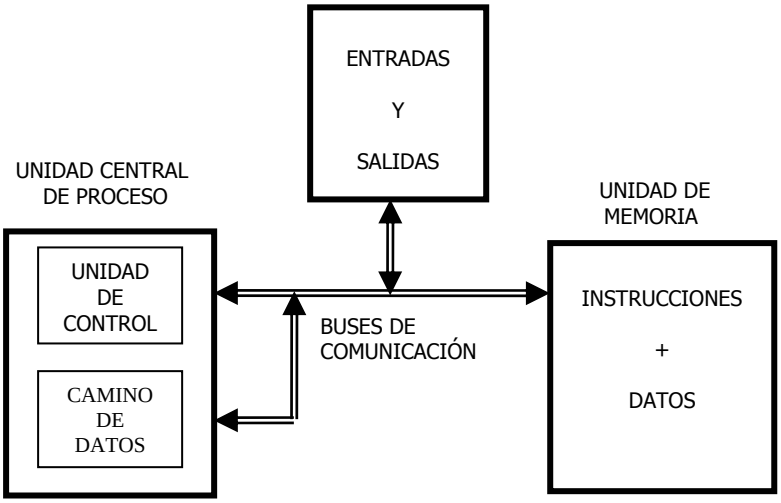


Figura 1.10. Arquitectura de J. Von Neumann.

Si se dispone de un microprocesador que maneja palabras de 8 bits, conectado a un bus de 8 bits de ancho, que lo conecta con la memoria, deberá manejar instrucciones de una o más unidades de 8 bits (1 byte), con lo que se obliga a la memoria a estar internamente dividida en unidades de 8 bits. En esta arquitectura si debemos acceder a una instrucción y/o dato de mas de 8 bits deberemos hacer de dos a mas accesos a memoria seguidos.

Esta arquitectura se denomina de tipo CISC " Computador de Juego de Instrucciones Complejas". Las instrucciones complejas exigen mucho tiempo de CPU para ejecutarlas y sólo un acceso a la memoria que era lenta.

También destaca el hecho de que compartir el bus ralentiza los tiempos de operación ya que no se puede hacer la búsqueda de un nueva instrucción antes de terminar de realizar la transferencia de datos resultante de los resultados obtenidos por la operación anterior.

Por tanto, esta arquitectura tiene dos principales desventajas:

- La longitud de las instrucciones está limitada por la longitud de los datos, por lo tanto el procesador se ve obligado a hacer varios accesos a memoria para buscar instrucciones complejas.

- La velocidad de operación está limitada por el efecto cuello de botella, que significa que un bus único para datos e instrucciones impide superponer ambos tipos de acceso.

1.3.2- Arquitectura moderna (1990 - hoy).

Propone modificaciones en la arquitectura del equipo físico y mejoras y nuevas prestaciones en el tiempo lógico. Un ejemplo en el primer aspecto es la arquitectura Harvard, que está especialmente diseñada para atacar las debilidades de la arquitectura Von Neumann, la solución, conceptualmente, es harto sencilla, se construye un procesador que está unido a dos tipos de memoria diferentes por medio de dos buses independientes.

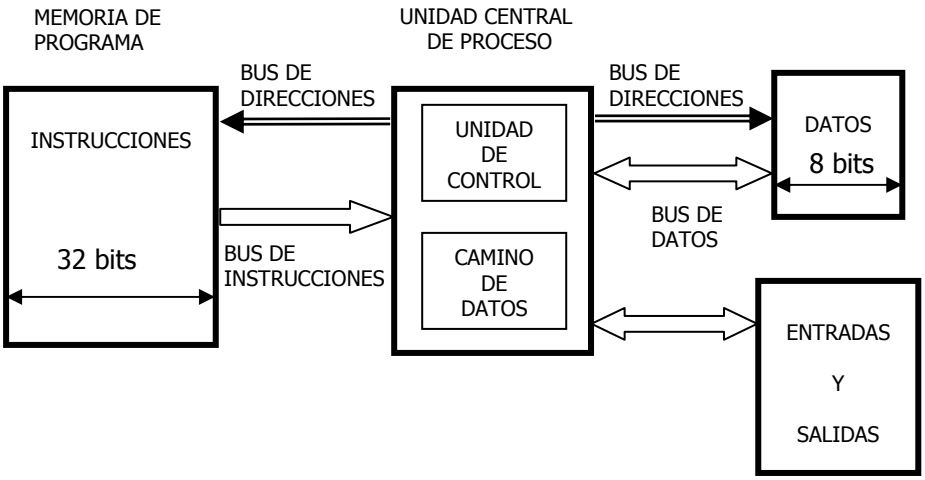


Figura 1.11. Arquitectura Harvard.

La memoria de datos y la memoria de instrucciones son independientes, almacenándose en ellas los datos y el programa, respectivamente.

Para un procesador de tipo RISC "Computador de Juego de Instrucciones Reducido", el conjunto de instrucciones y el bus de la memoria de programa pueden diseñarse de manera tal que todas las instrucciones tengan la misma longitud que la posición de la memoria y lo mismo con los datos. Además, como los buses de ambas memorias son independientes, la CPU puede estar accediendo a los datos para completar la ejecución de una instrucción, y al mismo tiempo estar leyendo la próxima instrucción a ejecutar.

Una forma de potenciar el aislamiento entre las instrucciones y los datos es la incorporación de memorias caché ultrarápidas, que como sucede en los últimos modelos Pentium, una se encarga de guardar los datos que va a precisar la CPU y otra las instrucciones.

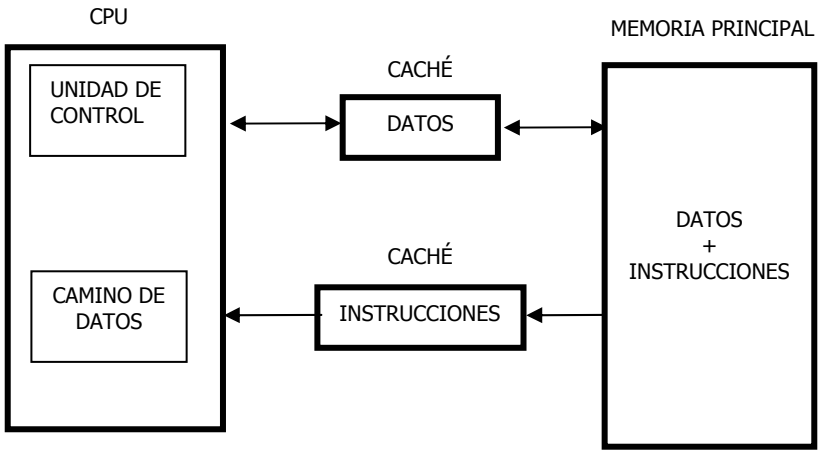


Figura 1.11. Arquitectura actuales Pentium.

1.4- INFLUENCIA DE LA TECNOLOGÍA EN LA EVOLUCIÓN DE LA ARQUITECTURA DE COMPUTADORES.

Desde los tiempos en que se estableció la arquitectura de von Neumann, el desarrollo de los computadores se ha realizado a un ritmo inimaginable, la causa del crecimiento de los computadores se debe fundamentalmente a la mejora en la tecnología. Esta evolución de la arquitectura de los computadores puede contemplarse en cuatro etapas sucesivas en las que se aprecia su relación con la tecnología.

1.4.1- Primera etapa

Esta primera etapa contempla la época situada en la mitad del siglo XX. La tecnología en aquellos momentos se basaba en las válvulas de vacío. Los computadores seguían plenamente el modelo de von Neumann, con un módulo de proceso (CPU), un módulo de E / S y una memoria

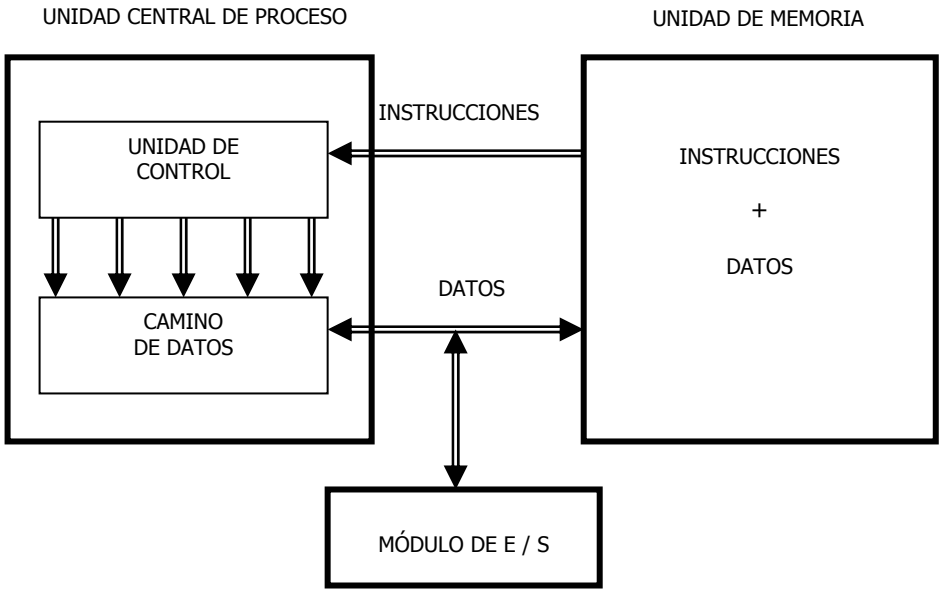


Figura 1.12. Diagrama general por bloques de los computadores de la primera etapa.

Tanto la CPU (Unidad Central de Procesos) como la memoria estaban construidas con válvulas de vacío con lo que la velocidad de funcionamiento en ambos bloques era igual, aprovechando por tanto el rendimiento de ambos por igual.

La sencillez de la CPU y el pequeño tamaño de la memoria obligaban a utilizar un conjunto reducido y elemental de instrucciones. La CPU tenía que acceder frecuentemente a memoria, pero al poseer ambos módulos la misma tecnología el rendimiento conjunto era aceptable. A este tipo de computadores se les denominó RISC (Computadores de Set de Instrucciones Reducidos).

1.4.2- Segunda etapa

En esta etapa aparecen los primeros circuitos integrados de pequeña y media escala de integración (SSI y MSI), que se aplican a la construcción de la CPU, mientras que la memoria principal es construida con núcleos de ferrita, cuyos tiempos de accesos son elevados.

La velocidad de la memoria principal es 10 veces menor que la CPU, lo que provoca largos períodos de inactividad de la CPU. A este fenómeno se le conoce como el "Cuello de Botella de Von Neumann".

Para solucionar esta pérdida de rendimiento de la CPU, se utilizan juegos de instrucciones complejos, en los que cada instrucción equivale a varias operaciones elementales, evitándose así accesos a memoria principal. A este tipo de computadores se les denomina CISC (Computadores de Set de Instrucciones Complejo). La CPU contendrá una Memoria de Control, que se trata de una memoria rápida en la que se almacenan las operaciones elementales (microinstrucciones) correspondientes a cada instrucción compleja (macroinstrucción).

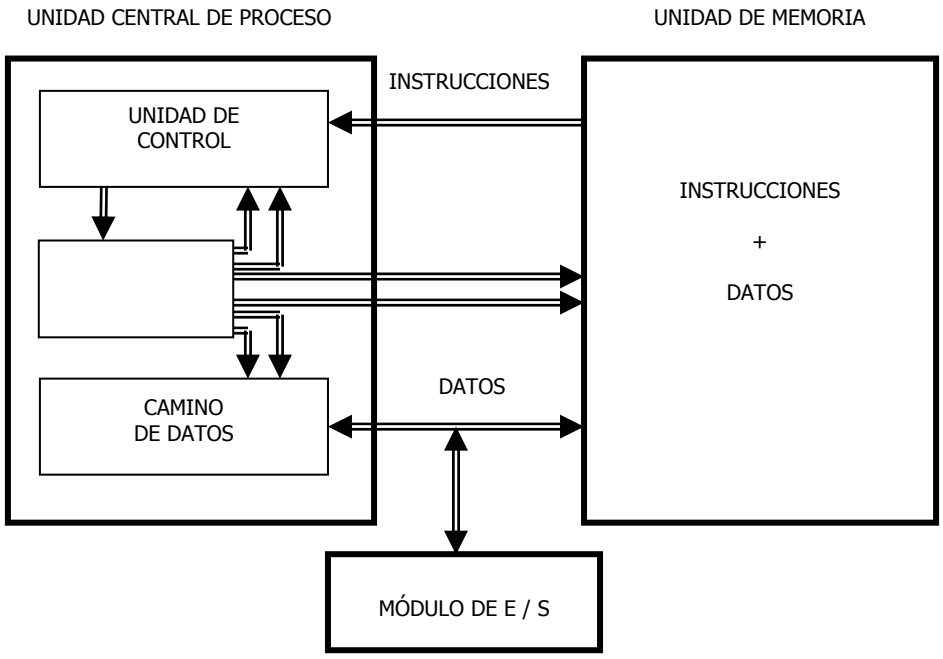


Figura 1.13. Diagrama general por bloques de la segunda etapa.

Como conclusión, se incrementa el proceso de decodificación de la macroinstrucción pero se reduce el número de accesos a memoria principal, solucionándose el problema de las diferentes velocidades de la memoria y la CPU, y acercándonos al modo de programar de los lenguajes de alto nivel, que comienzan a desarrollarse por esta época.

1.4.3- Tercera etapa

La tecnología de circuitos integrados siguió evolucionando, alcanzando la alta escala de integración (LSI), que permitía la fabricación de memorias rápidas, pero que seguían siendo lentas con respecto a la CPU.

La decodificación de instrucciones complejas requería más tiempo que el acceso a estas memorias, por lo que los computadores CISC disminuyen la complejidad de las instrucciones, reduciéndose así el número de microinstrucciones correspondientes a cada instrucción. Este nuevo tipo de computadores entre RISC y CISC obtiene el mayor rendimiento posible de la CPU y estas memorias.

Aparece la memoria caché que se encarga de guardar la información de uso más frecuente de la memoria principal, para disminuir el número de accesos a esta última. Estas memorias son más rápidas, más caras y se usan con pequeñas capacidades. Como se ve en la figura 1.14 la Unidad de Control accede a la memoria caché, y sólo cuando no está la información que necesita en ella, a la memoria principal. El contenido de la caché está optimizado para que sea el de más uso, obteniendo de 5 a 10 veces velocidades mayores que la memoria principal.

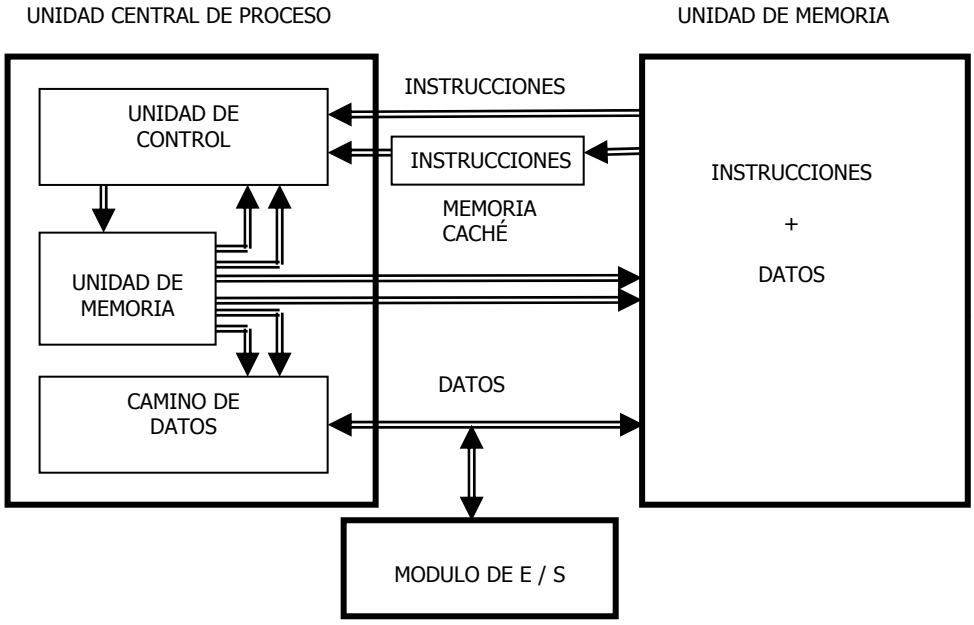


Figura 1.14. Diagrama general por bloques de la tercera etapa.

1.4.4- Cuarta etapa

Se desarrolla la tecnología VLSI (Muy alta escala de integración) que se aplica tanto a la CPU como a la memoria caché, por lo que se vuelve a los computadores RISC.

Las instrucciones vuelven a ser pocas y básicas, desapareciendo así la Memoria de Control.

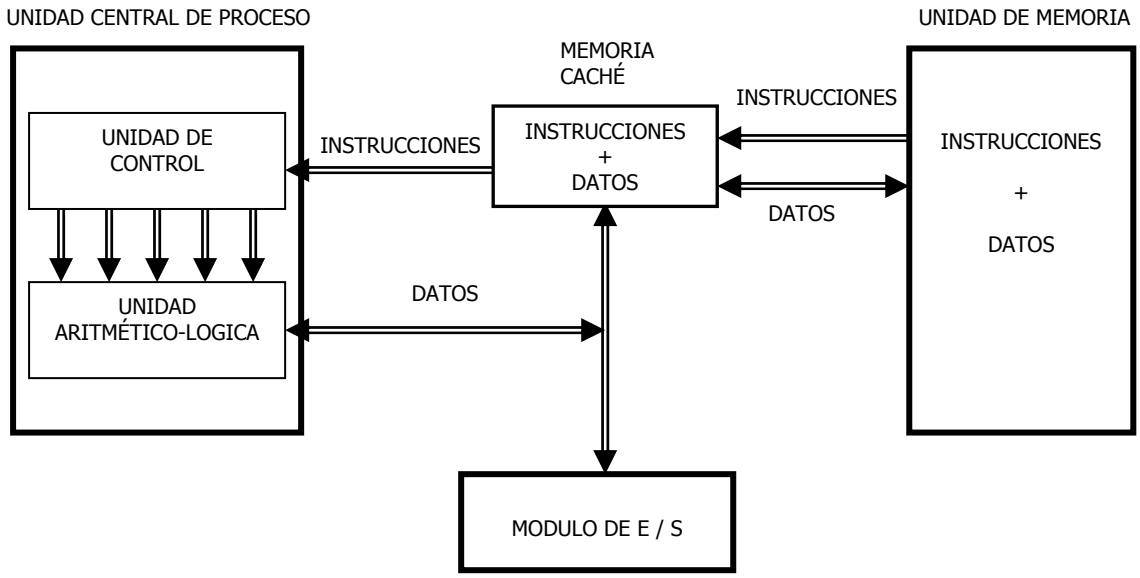


Figura 1.15. Diagrama general por bloques de la cuarta etapa.

En esta etapa el acceso a memoria se hará siempre a la caché, que contendrá la parte de la memoria principal a utilizar en la mayoría de los accesos.

1.5-TAXONOMÍAS O CLASIFICACIONES DE COMPUTADORES SEGÚN SU ARQUITECTURA

Según las diferentes arquitecturas desarrolladas pueden clasificarse los computadores de diferentes puntos de vista. Una de las clasificaciones más extendida es la denominada taxonomía de Flynn (1966), que se detalla a continuación.

1.5.1- Taxonomía de Flynn

Esta taxonomía se basa en el número de flujos de instrucciones y flujos de datos que posee cada sistema computador.

El proceso computacional consiste en la ejecución de una secuencia de instrucciones sobre un conjunto de datos. Flujo de instrucciones es la secuencia sobre la que opera un procesador, y el flujo de datos comprende la secuencia de datos de entrada y los resultados parciales y totales.

Las arquitecturas de computadores se caracterizan por el hardware que destinan a atender a los flujos de instrucciones y datos.

Flynn propuso 4 categorias:

- **SISD:** Simple flujo de instrucciones, simple flujo de datos.
- **MISD:** Múltiple flujo de instrucciones, simple flujo de datos.
- **SIMD:** Simple flujo de instrucciones, múltiple flujo de datos.
- **MIMD:** Múltiple flujo de instrucciones, múltiple flujo de datos.

Después introdujo una quinta clasificación separada un poco de las cuatro anteriores:

- **MTMD:** Múltiple tareas, múltiple flujo de datos.

1.5.1.1- Computadores SISD

Responden a los monoprocesadores convencionales (tipo Von Neumann) que más se usan. Al disponer de una única Unidad de Proceso (Camino de Datos) sólo existe un Flujo de Instrucciones y un Flujo de Datos.

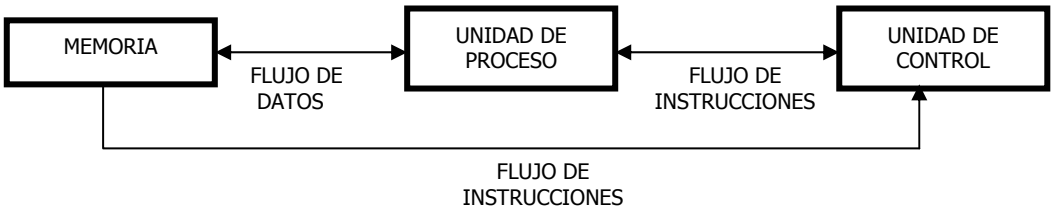


Figura 1.16. Estructura básica de los ordenadores SISD.

1.5.1.2- Computadores MISD

Existen n Unidades de Proceso, cada una con su propia Unidad de Control y sus propias instrucciones, pero operando sobre el mismo flujo de datos, de forma que la salida de un procesador pasa a ser la entrada (operandos) del siguiente en el macrocauce de los datos. Se hacen diferentes operaciones con los mismos datos.

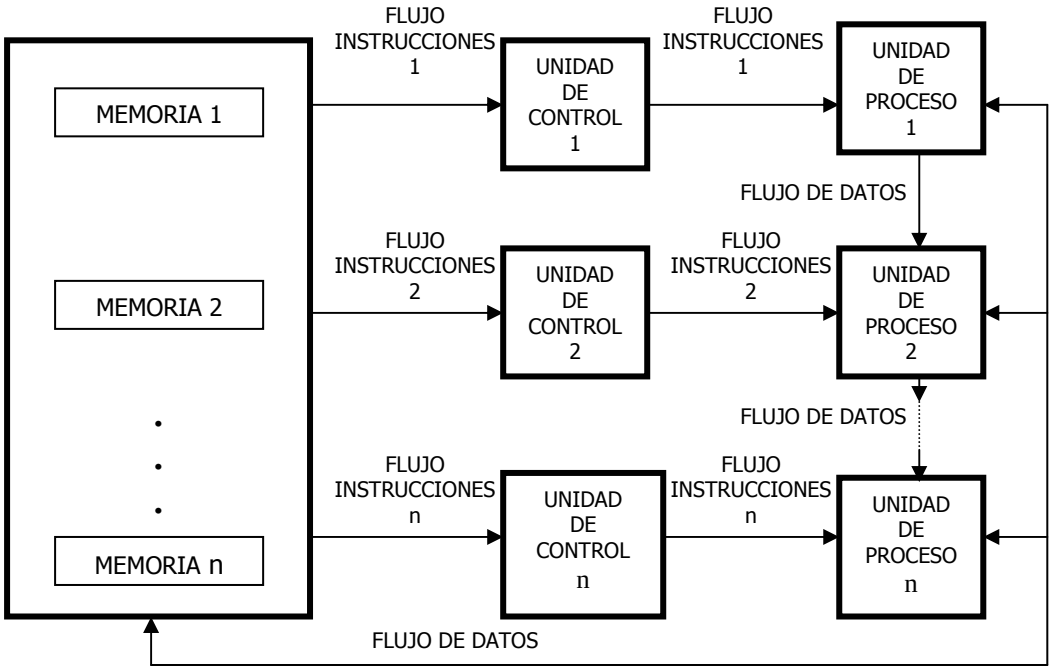


Figura 1.17. Arquitectura de los computadores MISD.

Los arquitectos de computadores han menospreciado esta organización y de hecho no existe ninguna materialización real de este tipo.

1.5.1.3- Computadores SIMD

Flujo único de instrucciones y Flujo múltiple de Datos. Sólo hay una Unidad de Control que controla las diferentes Unidades de Proceso. Todas la Unidades de Proceso reciben la misma instrucción, pero operan sobre los diferentes datos procedentes de la memoria compartida.

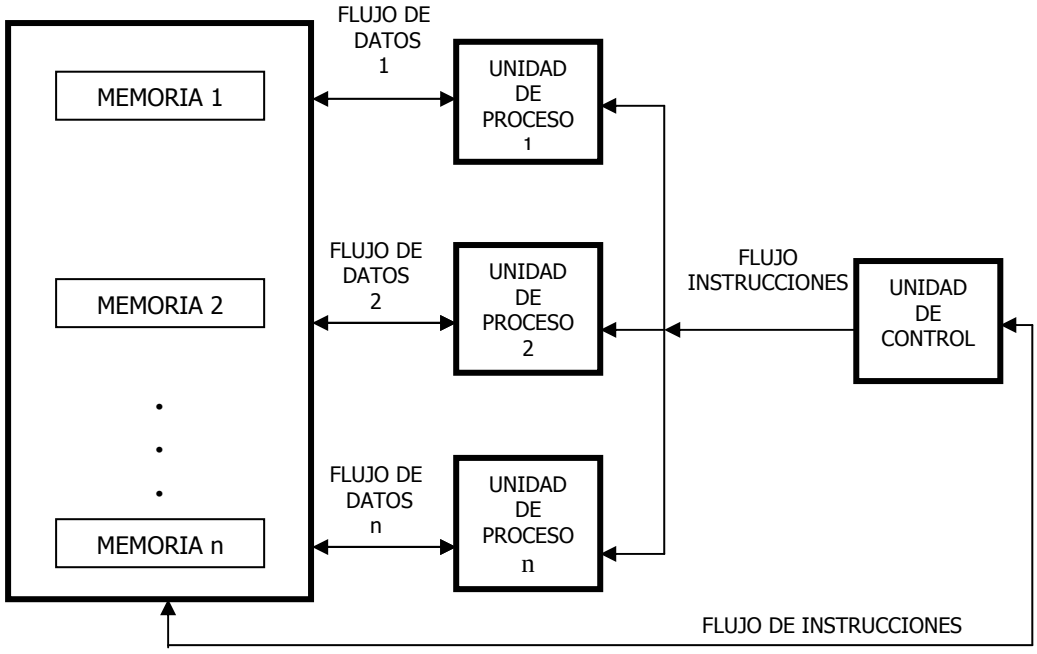


Figura 1.18. Arquitectura de los computadores SIMD.

La misma instrucción la reciben todas las Unidades de Proceso, pero a veces no todas la realizan porque la instrucción lleva codificado los procesadores que intervienen y los que están inactivos.

La mayoría de los computadores SIMD necesitan que exista intercomunicación entre las Unidades de Proceso, para compartir datos y resultados intermedios. Hay dos formas de lograrlo:

1. **Memoria Compartida**: Todas las Unidades de Proceso utilizan una memoria común y cuando una quiere enviar un dato a otra, primero lo escribe en una posición que la otra conoce y luego ésta lee dicha posición. Es como un tablón de anuncios que puede usar todo el mundo.

2. **Red de Interconexión**: Las M posiciones de la memoria se reparten entre los N procesadores a razón de M/N posiciones de memoria local para cada uno, además cada procesador se une con los demás mediante una línea Full-Duplex de forma que en un momento determinado un procesador puede recibir datos de otro y al mismo tiempo mandar otros datos a un tercer procesador. Ver Figura 1.19.

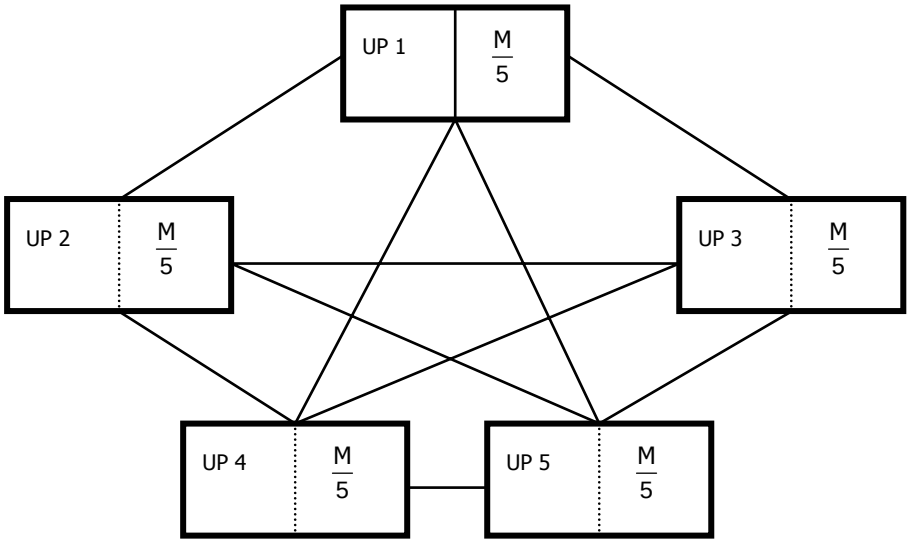


Figura 1.19. Arquitectura con red de interconexión.

En este caso particular los 5 procesadores se reparten las M posiciones de memoria. Permite la comunicación instantánea entre cualquier pareja de procesadores y de varias parejas entre sí (sólo un procesador se comunica con otro). Además existen varios tipos de interconexión de redes como la conexión serie o lineal, bidimensional o malla, en árbol, etc...

Los SIMD son mucho más útiles y comerciales en el mercado que los MISD y también más flexibles. Además, es más fácil hacer algoritmos para los SIMD que para los MISD.

El caso ideal de los SIMD es cuando un problema se puede dividir en subproblemas idénticos y además éstos tienen las mismas instrucciones.

Algunos ejemplos de esta arquitectura fueron: Thinking Machines CM-2, MassPar computers , Procesador MMX

1.5.1.4- Computadores MIMD

Este tipo de computadora se basa en el paralelismo como las SIMD, la diferencia es que la arquitectura MIMD es asíncrona. No tiene un reloj central. Cada procesador en un sistema MIMD puede ejecutar su propia secuencia de instrucciones y tener sus propios datos. Esta característica es la más general y poderosa de esta clasificación.

Es una agrupación de monoprocesadores convencionales, cada uno con su Unidad de Control, su Unidad de Proceso y su memoria local. Cada uno dispone de su Flujo de Instrucciones y de su Flujo de Datos, trabajan en paralelo y de forma asíncrona y están comunicados entre ellos igual que los SIMD. Usan la memoria compartida o bien la red de interconexión.

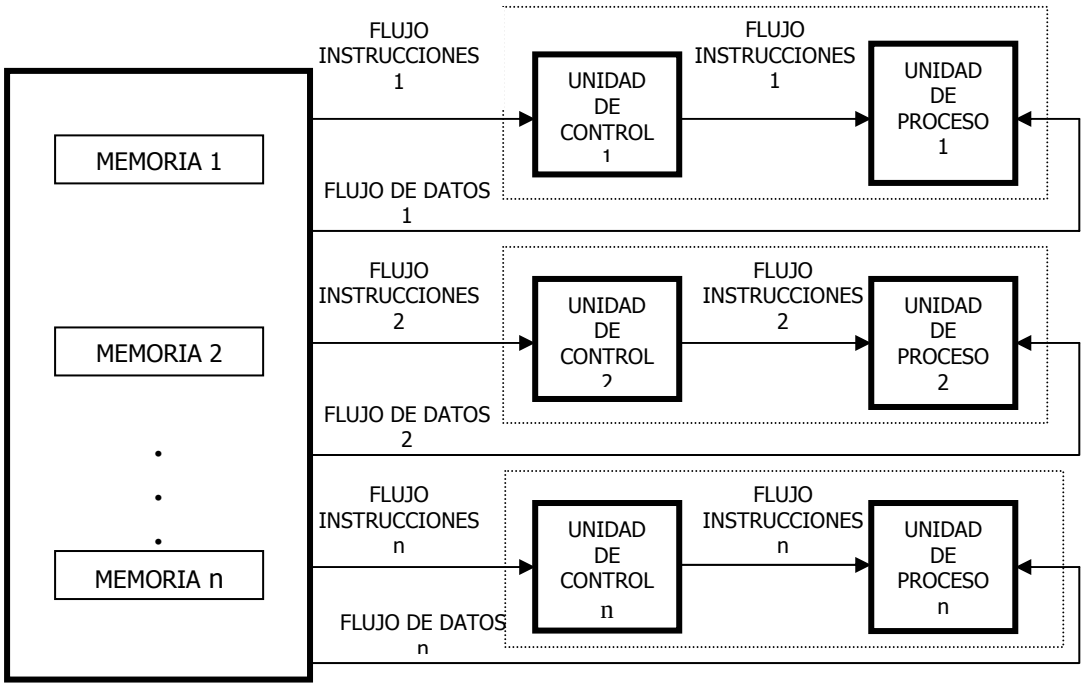


Figura 1.21. Arquitectura de los computadores MIMD.

Se supone que son los sistemas más perfectamente paralelos, ya que el paralelismo es total, pero también son los más caros.

Los algoritmos para los MIMD deben tener un factor claro de paralelismo, aunque pueden ser totalmente asíncronos, y además se necesita intercomunicación. Normalmente comienzan cargando una tarea básica a uno cualquiera de los procesadores, y éste va descomponiendo tareas y lanzándolas a los demás, así como creando dos colas, una de proceso y otra de procesadores. En la primera se van introduciendo los procesos pendientes de ejecutar, y en la segunda se van metiendo los procesadores que van quedando libres después de terminar su tarea.

Ejemplos de computadores con arquitectura MIMD son BURR D-85, Cmmp, CRAY/2, CRAY-MP y IBM 370/168MP..

1.5.1.5- Computadores MTMD

Estos computadores surgen como una extensión a la clasificación de Flynn, algo restringida al contemplar la ejecución sólo a nivel de instrucciones. Múltiples Tareas con Múltiples Flujos de Datos.

Son como los computadores MIMD, la única diferencia es la tarea que se aplica a cada Unidad de Proceso. Estos computadores son capaces de ejecutar concurrentemente un número determinado de tareas, cada una con su propio conjunto de datos.

1.5.2- Otras taxonomías

Existen otras taxonomías que no son tan populares como la de Flynn entre las que destaca la taxonomía de Shore que al igual que la de Flynn, clasifica los computadores en función del número de elementos; pero mientras que la taxonomía de Flynn pretende clasificarlos por la organización del software (Instrucciones y Datos), la clasificación de Shore lo hace por la estructura del hardware (Unidad de Control, Unidad de Proceso y Memoria de Datos e Instrucciones).Por lo tanto la aparición de paralelismo dentro de cada uno de estos componentes no se valora.

La arquitectura Shore se representa seis tipos:

1. **Tipo 1**: Formada por una UC (Unidad de Control) conectada a una UP (Unidad de Proceso) y a una Memoria de Instrucciones.
2. **Tipo 2**: Similar a la anterior, con la salvedad de que las lecturas de memoria se realizan de forma paralela, es decir, un bit de cada palabra por cada acceso. Así la unidad de proceso está preparada para realizar operaciones con los datos leídos de esta forma.
3. **Tipo 3**: Es una combinación de las arquitecturas anteriores; está formada por una memoria bidimensional a la que acceden dos UP que operan en consecuencia a la lectura que realizan, horizontal o vertical. La UC supervisa las dos UP.
4. **Tipo 4**: Existen múltiples UP conectadas a una sola UC, que recibe órdenes de una Memoria de Instrucciones. Cada UP trabaja con una Memoria local de Datos. No existe ninguna comunicación entre ellas.
5. **Tipo 5:** Es similar a la anterior, pero las UP se encuentran interconectadas entre ellas, pudiendo así la misma UP acceder a varios módulos de memoria.
6. **Tipo 6:** En esta arquitectura se integran la UP y la Memoria local en un solo componente, que lógicamente estará conectado a una UC.

Esta clasificación es bastante incompleta, y queda poco determinado el tipo de paralelismo que se puede encontrar en un sistema. Además, tampoco diferencia, por ejemplo, sistemas que implementen la segmentación dentro de la Unidad de Proceso y/o Unidad de Control, de los que no las utilicen.

Otra taxonomía es la "Estructural" que no se basan sólo en el paralelismo para clasificar los computadores, sino que estudian también el modo de tratar los datos, la existencia de segmentación y su tipo. Es una aproximación a la clasificación global y en la que intervienen varios criterios, incluyendo la aplicación o no de técnicas de paralelismo en distintos niveles.

EL RENDIMIENTO EN LOS COMPUTADORES 2

2.1- INTRODUCCIÓN

En este capítulo se analiza el concepto y las unidades de medida del rendimiento de los computadores.

Definimos en general rendimiento de un elemento como una medida de productividad o capacidad de trabajo de dicho elemento en función de unos parámetros o características.

El rendimiento no es una medida única, sino una relación entre varias medidas, la valoración de varios parámetros, y depende del objetivo al que se oriente el trabajo.

EJEMPLO 1

Para entender mejor el concepto de rendimiento analizaremos el siguiente ejemplo, se trata de determinar cuál de los dos automóviles tiene mejor rendimiento.

Coche	Consumo	Velocidad	Plazas	C.V	Precio
A	4.5	145	4	90	13.400€
B	7.1	185	7	110	15.300€

Figura 2.1. Prestaciones de los vehículos.

Como hemos comentado anteriormente, el rendimiento dependerá de la finalidad a la que se quiera destinar el vehículo y de unos parámetros, en este caso consumo, velocidad, etc **:**

- Si se utiliza para transportar 2 personas diariamente teniendo en cuenta el coste, el rendimiento mejor será el del vehículo A.
- Si el vehículo es requerido para recorrer 100 km y transportar 6 personas, el rendimiento mejor será el del vehículo B.
- Si lo que prima es el tiempo en un determinado recorrido la elección acertada es el B.

2.2-RENDIMIENTO EN LOS COMPUTADORES

Se define ***rendimiento de un sistema*** como la capacidad que tiene dicho sistema para realizar un trabajo en un determinado tiempo. Es inversamente proporcional al tiempo, es decir, cuanto mayor sea el tiempo que necesite, menor será el rendimiento.

Los computadores ejecutan las instrucciones que componen los programas, por lo tanto el ***rendimiento de un computador*** está relacionado con el tiempo que tarda en ejecutar los programas. De esto se deduce que el tiempo es la medida del rendimiento de un computador.

El ***rendimiento del procesador*** depende de los siguientes parámetros:

1. Frecuencia de la CPU (f_{CPU}) : es el número de ciclos por segundo al que trabaja el procesador o CPU. No confundir la frecuencia de la CPU con la frecuencia del sistema, el bus del sistema trabaja a menor frecuencia que la CPU.

$$f_{CPU} = \frac{\text{n° ciclos}}{\text{segundo}}$$

2. Periodo de la CPU (T_{CPU}) : es el tiempo que dura un ciclo y es la inversa de la frecuencia de la CPU.

$$T_{CPU} = \frac{1}{f_{CPU}}$$

3. Ciclos por instrucción (CPI) : las instrucciones se descomponen en microinstrucciones, que son operaciones básicas que se hacen en un ciclo de reloj. En un programa se llama CPI al promedio de microinstrucciones que tienen las instrucciones del programa, es decir, los ciclos de reloj que se tarda de media en ejecutar una instrucción.

$$CPI = \frac{\sum_{i=1}^{n} N^{o}\ Instruc_{i} * CPI_{i}}{N^{o}\ InstrucTot}$$

4. Número de instrucciones del programa : cuantas más instrucciones haya en el programa más tiempo se tarda en ejecutarlo luego baja el rendimiento. El que tengamos un número reducido de instrucciones dependerá del programador y de que dispongamos de un buen compilador.

5. Multitarea : hace referencia a la capacidad que tiene un computador de atender simultáneamente varias tareas.

Como anteriormente hemos comentado, el ***rendimiento de un procesador*** para un programa concreto es un factor inversamente proporcional al tiempo que tarda en ejecutar dicho programa.

$$\eta_{prog} = \frac{1}{T_{prog}}$$

El ***tiempo de programa*** depende a su vez del número de instrucciones del programa y del tiempo que se tarda en ejecutar cada instrucción.

$$T_{Prog} = N°\text{instruc.} * T_{\text{instruccio nes}}$$

El ***tiempo*** que tarda en ejecutarse cada ***instrucción*** depende del número de microinstrucciones o ciclos en los que se descompone. Cada microinstrucción tarda distintos ciclos de reloj en ejecutarse, por lo que se hace un promedio ponderado de ciclos de instrucción.

$$T_{instrucción} = CPI * T_{CPU}$$

$$T_{prog} = N° \text{ instrucciones} * CPI * T_{CPU}$$

$$\eta_{prog} = \frac{1}{T_{prog}} = \frac{1}{N°instruc * CPI * T_{CPU}} = \frac{f_{CPU}}{N°instruc * CPI}\ s^{-1}$$

En resumen, el rendimiento de un procesador para un determinado programa queda en función de tres factores:

- ***Frecuencia de la CPU*** la cual depende fundamentalmente de la tecnología de fabricación del procesador. Cuanto mayor sea la frecuencia de la CPU, mejor será el rendimiento.
- ***Número de instrucciones del programa*** el cual depende del programador, del lenguaje de programación y del compilador. Cuanto mayor sea el número de instrucciones del programa peor rendimiento tendrá.
- ***CPI*** que depende de diseño interno o arquitectura del computador y del software o instrucciones que se hayan elegido. Es importante optimizar el programa con instrucciones que tengan pocos ciclos. Cuanto mayor sea el CPI, peor será el rendimiento.

EJEMPLO 2

Dadas las características de un programa calcular el tiempo que tarda en ejecutarse el programa y su rendimiento. Se sabe que la CPU trabaja a 1GHz. El programa se compone de 19 instrucciones de tres tipos: A, B y C y cada tipo de instrucción tiene un número de microinstrucciones diferente según indica la tabla:

Tipo de instrucción	Instrucciones/Prog.	CPI_i
A	**5**	**2**
B	**6**	**3**
C	**8**	**1**

Tiempo de programa:

$$CPI = \frac{\sum N°instruc.*CPI_i}{N°instruc.\ TOTAL} = \frac{(\ 5*2+6*3+8*1\ )}{19} = 1.89$$

$$T_{prog} = N°instruc.*CPI*T_{CPU} = 19*1.89*\frac{1}{1000*10^6} = 36ns$$

Rendimiento:

$$\eta_{prog} = \frac{1}{T_{prog}} = \frac{1}{36ns} = 27777777.78\ s^{-1}$$

EJEMPLO 3

Se dispone de un computador basado en un Pentium IV a 2GHz. Se ejecuta un programa compuesto por 1000 instrucciones. Según los datos de la siguiente tabla, calcular el rendimiento:

Tipo de instrucción	Instrucciones/Prog.	CPI_i
Lógico-Aritmética	20%	4
Salto incondicional	10%	3
Salto condicional	10%/10%	7/5
Carga (Reg<-Mem)	30%	3
Almacenamiento (Mem<-Reg)	20%	4

Tiempo de programa:

$$CPI_{Pentium} = \frac{\left(200*4+100*3+\left(100*7+100*5\ \right)+300*3+200*4\ \right)}{1000} = 4$$

$$T_{prog} = N°instruc.*CPI*T_{CPU} = 1000*4*\frac{1}{2000*10^6} = 2000ns$$

Rendimiento:

$$\eta_{prog} = \frac{1}{T_{prog}} = \frac{1}{2000ns} = \frac{10^9}{2*10^3} = 500000\ s^{-1}$$

2.3.- COMPARACION DEL RENDIMIENTO

El objetivo principal de los compradores, diseñadores y otros usuarios de computadores, es conseguir el mayor rendimiento posible con el menor coste.

El rendimiento, por tanto, ofrece una medida cuantitativa que permite comparar unos computadores frente a otros y decidir cual nos aporta mayores ventajas.

Cuando comparemos 2 máquinas hablaremos de ***aceleración :***

1.- *Aceleración del rendimiento* : Es la relación que existe entre el rendimiento del computador A y el rendimiento del computador B. Indica que el computador A es un n% mejor que el B.

$$A_\eta = \frac{\eta_A}{\eta_B}$$

2.- *Aceleración del tiempo* : Es la relación entre el tiempo que tarda el computador A en ejecutar un programa y el tiempo que tarda el computador B, y es inversamente proporcional al rendimiento.

$$A_T = \frac{T_A}{T_B} = \frac{\eta_B}{\eta_A}$$

EJEMPLO 4

Tenemos un PentiumIV a 500 MHz y ejecuta un programa de 1000 instrucciones, según la siguiente tabla calcular el rendimiento. Si sustituimos el procesador por un PentiumIV a 1GHz, ¿cómo varia el rendimiento?.

Tipo de instrucción	Instrucciones/Prog.	CPI_i
Carga y Almacenamiento	30%	3
Lógico Aritméticas	50%	4
Salto	20%	2

a) $$CPI = \frac{(\ 300*3+500*4+200*2\ )}{1000} = 3{,}3$$

$$\eta_{prog} = \frac{f_{cpu}}{n^o inst.*cpi} = \frac{500*10^6}{1000*3{,}3} = 151515{,}15\ s^{-1}$$

b) El rendimiento se multiplicará por 2 ya que es directamente proporcional a la frecuencia.

EJEMPLO 5

Tenemos un PentiumIV a 2GHz y ejecuta un programa de 1000 instrucciones, según la siguiente tabla calcular el rendimiento. Para aumentar el rendimiento se añade posteriormente una caché que ahorra 1 ciclo en accesos di el dato está en la caché y si no está dura lo mismo. El porcentaje de acierto es del 90%. Calcular el rendimiento y la aceleración del rendimiento con respecto del anterior sin caché.

Tipo de instrucción	Instrucciones/Prog.	CPI_i
Lógico-Aritmética	300	5
Llamadas a subrutinas	100	9
Salto incondicional	100	6
Salto condicional	50/50	9/3
Carga (Reg<-Mem)	100	2
Almacenamiento (Mem<-Reg)	100	4
Mov (Reg<-Reg)	200	2

$$CPI_{\text{sin } caché} = \frac{\left(300*5+100*9+100*6+\left(50*9+50*3\right)+100*2+100*4+200*2 \right)}{1000} = 4{,}6$$

$$\eta_{\text{sin } caché} = \frac{2*10^9}{1000*4{,}6} = 434708{,}6\ s^{-1}$$

$$CPI_{concaché} = \frac{(..... + 100*6+\left(50*9+50*3\right)+(90*1+10*2)+(90*3+10*4)+200*2)}{1000} = 4{,}42$$

$$\eta_{concaché} = \frac{2*10^9}{1000*4{,}42} = 452488\ s^{-1}$$

$$CPI_{\text{sin } caché} = \frac{\left(300*5+100*9+100*6+\left(50*9+50*3\right)+100*2+100*4+200*2 \right)}{1000} = 4{,}6$$

$$A\eta = \frac{rend.concaché}{rend.\text{sin } caché} = \frac{452488}{434782} = 1{,}0407\ s^{-1}$$

Hemos conseguido un aumento en el rendimiento de un 1% al incorporar caché.

2.4- UNIDADES DE MEDIDA DEL RENDIMIENTO GLOBAL

En la búsqueda de una medida estándar del rendimiento de los computadores, se han desarrollado una serie de métricas populares como alternativa al uso del tiempo, el cual ha conducido en alguna ocasión a resultados distorsionados o interpretaciones incorrectas. En esta sección se explican algunas de las métricas más comúnmente usadas.

2.4.1- MIPS

Los MIPS son los millones de instrucciones por segundo que ejecuta un procesador para un programa determinado.

$$MIPS = \frac{N° \text{ instruc prog}}{T_{Prog} * 10^6} = \frac{N° \text{ instruc prog}}{N° \text{ instruc prog} * CPI * T_{CPU} * 10^6} = \frac{f_{CPU}}{CPI * 10^6}$$

La ventaja de esta unidad de medida es su fácil comprensión ya que un mayor número de MIPS indicará una mayor velocidad de la máquina. Sin embargo, no es una medida buena ya que puede producir resultados erróneos, un mismo programa en un mismo computador puede dar resultados diferentes según el programador o el repertorio de instrucciones utilizadas, además los MIPS pueden variar inversamente al rendimiento. Por ese motivo, se pasó a utilizar los modelo de referencia el VAX11/780.

EJEMPLO 6

Se dispone de una máquina con dos clases de instrucciones, la instrucción 1 tendrá un CPI de 3 y la instrucción 2 un CPI de 2. Al medir el código para el mismo programa para dos compiladores diferentes se obtienen los siguientes datos.

Código	Instrucción 1	Instrucción 2
Compilador A	6	2
Compilador B	4	2

Se supone que la frecuencia de reloj de la máquina es de 200MHz. ¿Qué secuencia de código se ejecuta con más rapidez de acuerdo con los MIPS?. Realizar lo mismo de acuerdo al tiempo de ejecución. Explicar los resultados.

Para calcular los MIPS utilizamos la ecuación:

$$MIPS = \frac{f_{CPU}}{CPI * 10^6}$$

$$CPI = \frac{\sum_{i=1}^{N}(CPI_i * I_i)}{N° instruc\ TOTAL}$$

$$CPI_A = \frac{(6*3+2*2)}{(6+2)} = \frac{22}{8} = 2.75$$

$$MIPS_A = \frac{200*10^6}{2.75*10^6} = 72.72$$

$$CPI_B = \frac{(4*3+2*2)}{(4+2)} = \frac{16}{6} = 2.66$$

$$MIPS_B = \frac{200*10^6}{2.66*10^6} = 75.18$$

Por consiguiente, el código producido por el compilador B tiene una frecuencia en MIPS más elevada.

Calcularemos ahora el tiempo de ejecución y para ello utilizaremos la formula:

$$T_{Prog} = \frac{CPI * N°Instruc}{f_{CPU}}$$

$$T_{PROG\ A} = \frac{2.75*(6+2)}{200*10^6} = 1{,}1^{-7}\ s^{-1}$$

$$T_{PROG\ B} = \frac{2.66*(4+2)}{200*10^6} = 79{,}8^{-7}\ s^{-1}$$

Por tanto, el compilador A es claramente más rápido lo cual contradice lo que habíamos observado con los MIPS.

2.4.2- MIPS VAX

Los MIPS VAX son la relación entre el tiempo que tarda un computador en realizar un programa y el tiempo que tarda en realizarlo el computador VAX11/780. Por ejemplo, un valor de MIPS VAX = 1 que toma el 80286 significa que dicho computador ejecuta un programa en el mismo tiempo que el VAX11/780.

$$MIPS\ VAX = \frac{T_{Prog}\ VAX11/780}{T_{Prog\ CPU}}$$

2.4.3- MEGAFLOPS NATIVOS Y RELATIVOS

Surgen ya que los MIPS no hacen distinción entre operaciones normales y operaciones en coma flotante.

Los ***Megaflops nativos*** indican los de millones de instrucciones en coma flotante por segundo que se realizan para un determinado programa.

$$MFLOPS_{nativos} = \frac{Millones\ instruc}{T_{Prog} * 10^6}$$

Los Megaflops nativos tampoco son fiables, ya que dependen mucho de programas y computadores. Tuvieron muchas críticas de los fabricantes de computadores, porque, hay computadores que solo tienen operaciones sencillas (ADD, SUB, MUL) y generan pocas operaciones en coma flotante, en cambio hay otros computadores que tienen más operaciones en coma flotante (DIV, SQRT, EXP, SIN, ...).Las operaciones sencillas tardan menos en ejecutarse que las operaciones complejas, por lo que apareció el concepto de Megaflops relativos.

Los ***Meglaflops relativos o normalizados*** indican los millones de operaciones en coma flotante por segundo pero teniendo en cuenta la equivalencia que tienen las operaciones complejas con las simples.

$$MFLOPS_{Relativos} = \sum (MillonesInstruc._i * Peso_i)$$

Instrucciones	Peso
ADD, SUB, MULT	1 operación en coma flotante
DIV, SQRT...	4 operación en coma flotante
EXP, SIN,...	8 operación en coma flotante

Figura 2.2. Equivalencias.

EJEMPLO 7

Hallar los MFLOPS nativos y relativos que se obtienen al utilizar un computador que ejecuta un programa en 66 segundos y que consta de las siguientes instrucciones en coma flotante.

Operación	Millones de instrucciones	Peso
ADD	20	1
SUB	10	1
MUL	10	1
DIV	12	4
EXP	2	8
SIN	99	8
SQRT	1	4

$$MFLOPS_{nativos} = \frac{(20+10+10+12+2+99+1)*10^6}{66*10^6} = 2{,}33$$

$$MFLOPS_{relativos} = \frac{(20*1+10*1+10*1+12*4+2*8+99*8+1*4)*10^6}{66*10^6} = 13{,}64$$

El resultado de los Mflops relativos es mayor porque utiliza muchas instrucciones de alta complejidad.

Tabla de rendimiento comparativa en Megaflops (millones de operaciones por segundo), obtenida de la revista PC Today para índices PC y estimación Apple para Mac:

Equipo	Rendimiento
Pentium III a 800 MHz	234,9 Megaflops
Pentium IV a 1.400 MHz	362,4 Megaflops
AMD Athlon a 1.000 MHz	486,4 Megaflops
PowerPC G4 a 450 MHz	3.000 Megaflops
PowerPC G4 a 500 MHz	7.000 Megaflops

Figura 2.3.Rendimiento en Megaflops.

Comparando:

PowerPC G4 a 450 MHz es:

- 12 veces superior al de un Pentium III a 800 MHz
- 8 veces superior al de un Pentium IV a 1,4 GHz
- 6 veces superior al de un AMD Athlon a 1 GHz

PowerPC G4 a 500 MHz es:

- 29 veces superior al de un Pentium III a 800 MHz
- 19 veces superior al de un Pentium IV a 1,4 GHz
- 14 veces superior al de un AMD Athlon a 1 GHz

2.5- PROGRAMAS DE PRUEBA O BENCHMARKS

Las medidas de rendimiento vistas hasta ahora no son válidas hoy en día dado que algunas como los MIPS tienden a dar resultados erróneos y a que los computadores actuales tienen una elevada velocidad. La mejor y mas fiable forma de calcular el rendimiento es medir el tiempo que los diversos computadores tardan en ejecutar los programas que realmente el usuario va a utilizar posteriormente. Ese será el mejor rendimiento para ese usuario, pero no para todos los usuarios, ya que el rendimiento es un valor relativo de acuerdo con la aplicación que se va a hacer.

El rendimiento de una estación de trabajo se mide analizando una serie de componentes físicos que determinan el rendimiento completo del sistema. A la hora de determinar el rendimiento global de un sistema, también hay que evaluar el sistema operativo, los equipos lógicos de red, los compiladores y las librerías gráficas, etc.

Para la evaluación del rendimiento de los sistemas se utilizan pruebas de rendimiento o benchmarks, que son programas modelo que efectúa la industria para comparar factores de rendimiento y relaciones rendimiento / precio de los diferentes modelos de computadores. No obstante, estas evaluaciones no son siempre directamente comparables, y en ocasiones ofrecen poca información, porque las configuraciones con las que se realizan las evaluaciones no son expuestas con claridad.

Hay multitud de programas de prueba o benchmarks. Estos programas se dividen principalmente en 4 grupos, los tres primeros tipos han quedado en desuso.

- BENCHMARK SINTÉTICOS
- BENCHMARKS REDUCIDOS
- BENCHMARK KERNEL O DE NÚCLEO
- PROGRAMAS REALES

2.5.1- Benchmarks Sinteticos

El objeto de este tipo de programas de prueba es simular el comportamiento de aplicaciones del mundo real. Para elaborar estas pruebas sintéticas se estudian una serie de aplicaciones y se desarrolla un código artificial que mezcla los cálculos matemáticos, bucles, llamadas a funciones, etc. Las series de programas de prueba sintéticos más conocidas son Whetstone y Dhrystone. Los Benchmark Sintéticos están formados por las rutinas más repetitivas de los programas más utilizados.

- **Dhrystone Benchmark (MIPS) :** miden la eficiencia del procesador y del compilador en un entorno de desarrollo de sistemas con lenguajes de alto nivel. Su valor es expresado en instrucciones Dhrystone por segundo (Dhrystone MIPS, millones de instrucciones Dhrystone por segundo). No realiza operaciones en coma flotante, por lo que muchos fabricantes no lo consideran como una medida adecuada para definir el rendimiento de hoy en día. Los resultados se relativizan respecto al número de instrucciones Dhrystone por segundo que son alcanzadas en un VAX 11/780.
- **Whetstone Benchmark :** predecesora del Dhrystone, es una medida desarrollada para evaluar sistemas que se vayan a destinar a la ejecución de pequeños programas científicos y de ingeniería. Sus programas se han implementado en FORTRAN e incluyen cálculos con enteros y en coma flotante, manipulación de arrays y saltos condicionales. Esta prueba predice cómo serán ejecutadas aplicaciones que hacen un uso intensivo de la unidad central de proceso. Los resultados son expresados en Kwips (miles de instrucciones Whetstone por segundo

Estos dos tipos de Benchmarks, a partir de los 80, cayeron en desuso y aparecieron los Benchmark reducidos.

2.5.2- Benchmarks Reducidos o de juguete

Los programas reducidos tienen entre 10 y 100 líneas de código y producen un resultado que el usuario conoce antes de ejecutarlo. Algunos ejemplos de este tipo de Benchmarks serían: el Towers, que resuelve el problema de las torres de Hanoi con muchas llamadas recursivas; el Perm, que calcula permutaciones de 7 tornadas de 5 en 5 y los programas Criba de Eratóstenes, Puzzle y Quicksort, que son los más populares porque son pequeños, fáciles de introducir y de ejecutar en cualquier computador.

Estos programas, al ser tan pequeños y sencillos, eran muy vulnerables, era muy fácil mejorar el rendimiento para un programa concreto, por lo que se pasó a los Benchmark Kernel o de núcleo.

2.5.3- Kernel Benchmarks

Son programas de pruebas formados por pequeñas piezas clave de programas reales que evalúan el rendimiento y lo aíslan de las características individuales de una máquina, permitiendo explicar las razones de las diferencias en los rendimientos de programas reales.

Los ejemplos más conocidos son el "Livermore Loops", una serie de 21 fragmentos de bucles pequeños, y el "Linpack", formado por un paquete de subrutinas de álgebra lineal. Sólo tratan algunos aspectos y son antiguos. No existen núcleos para evaluar prestaciones gráficas.

2.5.4- Programas Reales

Son programas hechos con partes de programas que realmente se utilizan mucho (procesadores de texto, compiladores, herramientas CAD, etc). Los problemas que presentan son que dependen mucho de los datos de entrada, suelen ser complejos de usar, los S.O. sobre los que se prueban suelen ser incompatibles y no son ni estándar ni de libre distribución.

2.5.4.1 Programa de pruebas SPEC (System Performance Evaluation Cooperative)

Para evitar el caos que había al medir el rendimiento, en 1986 se asociaron una serie de empresas (IBM, SUN, INTEL, APPLE, ...) y constituyeron una organización sin ánimo de lucro para la evaluación del rendimiento de los computadores. Esta organización tenía la función de diseñar y proponer los programas y pruebas que debían cumplir los computadores para la evaluación de su rendimiento. La mayoría de los fabricantes de ordenadores incluyen en sus páginas Web las medidas SPEC de sus equipos, eso sí, con las críticas correspondientes y poniendo claramente de relieve donde sobresalen. Incluso se basan en los resultados dados en estos benchmarks para tomar decisiones de diseño con respecto a sus máquinas.

En 1986 apareció la primera normativa y cada 3 años aproximadamente se modifican los programas teniendo en cuenta partes de programas actuales. SPEC consiste en realidad en tres grupos diferentes:

- OSG (Open Systems Group), que crea benchmarks para procesadores y sistemas que ejecutan UNIX, Windows NT y VMS.
- HPC (High Performance Group), que mide prestaciones de ordenadores dedicados a cálculo intensivo.
- GPC (Graphics Performance Characterization Group), que mide prestaciones de subsistemas gráficos, OpenGL y XWindows.

La evolución de los SPEC ha sido :

Primera versión 1989

La primera normativa se basó en 10 programas de los cuales 4 eran con números enteros y 6 con números en coma flotante. Se tomó como referencia el VAX11/780 y teniendo en cuenta lo que tarda en ejecutar los 10 programas se obtuvo una medida SPEC. Inicialmente se escogieron tres índices:

- **SPECint86** : es la media geométrica de los 4 programas de números enteros.
- **SPECfp89** : es la medida geométrica de los 6 programas para números en coma flotante.
- **SPECmark89** : es la media de las dos anteriores.

En 1989 se cambió la normativa porque se daba demasiada importancia a las operaciones en coma flotante.

Segunda versión 1992

En 1992 se redefinieron, incrementando el número de Benchmarks a 20, de los cuales 6 eran de números enteros y 14 con números en coma flotante. Se eliminó el concepto de SPECmark ya que favorecía a maquinas con alto rendimiento de punto flotante e introdujo el concepto de SPECrate basado en la ejecución multitarea. Estos programas tenían una variedad y un tamaño tal que era muy difícil que el fabricante mejorara el índice SPEC variando una parte de su computador.

- **SPECint92** : es un indicador del rendimiento de la UCP en un entorno comercial gcc, compress, espresso, etc.
- **SPECfp92** : es una buena medida del comportamiento en entornos de operación en coma flotante, como pueden ser entornos de ingeniería y científicos spice2g6, programas de astrofísica, partículas, procesamiento de señales, etc.

Tercera versión 1995

- **SPECint95** : 8 programas enteros.
- **SPECfp95** : 10 programas F.P..

Cuarta versión 2000, SPEC CPU2000

- **CINT2000** : 11 programas enteros.
- **CFP2000** : 14 programas F.P..

Hoy en día nos basamos en los SPEC de los años 95 y 2000. A continuación se ofrecen comparativas para los SPECint95, SPECcfp95 y los SPECint2000, SPECcfp2000 de distintos tipos de procesadores:

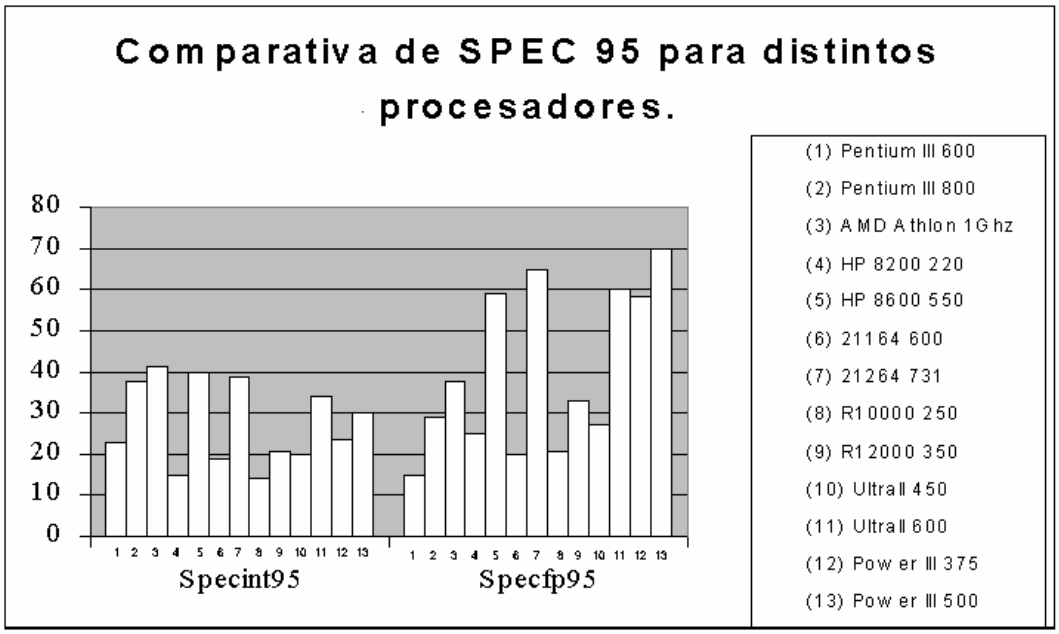


Figura 2.4. Spec95

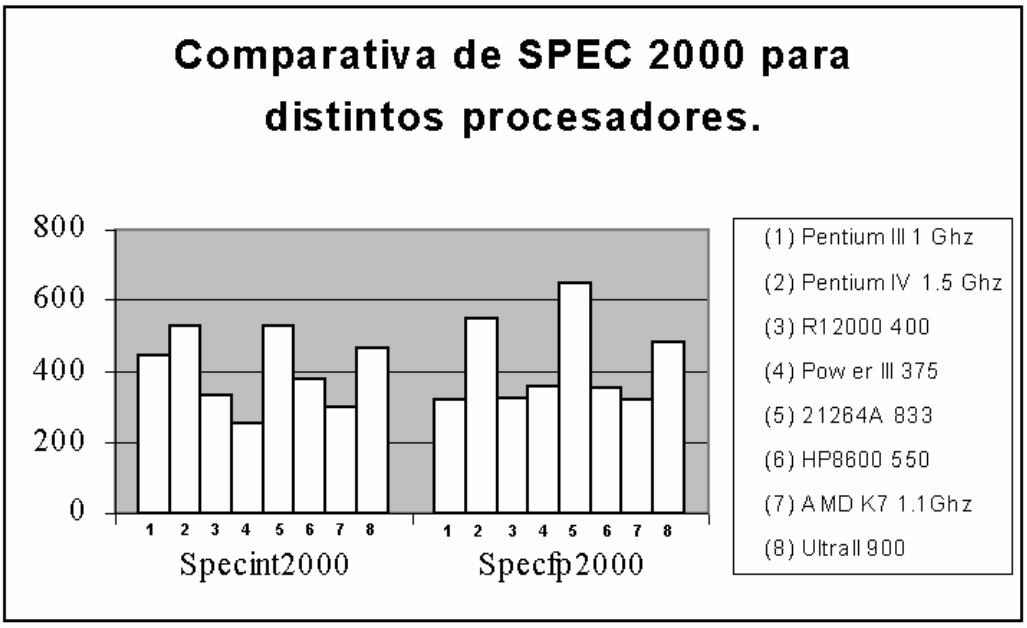


Figura 2.5

2.5.4.2 Otros benchmarks

- **AIM Suite III :** es un Benchmark que mide la eficiencia de sistemas multiusuario, en entornos servidores, UNIX, para atender varios usuarios ejecutando cada uno de ellos un proceso diferente.
- **Xmark :** es un Benchmark de dominio público basado en UNIX y disponible bajo el sistema Xwindow, para evaluar el rendimiento de aplicaciones propias de entornos gráficos y de autoedición.
- **Graphstone :** es una prueba gráfica que evalúa el rendimiento del subsistema de vídeo. El resultado se expresa en número de operaciones ejecutadas por segundo.
- **Khornerstone :** es una medida que prueba tanto el subsistema gráfico, como el rendimiento de la UCP y del disco duro, así como la capacidad para la ejecución de operaciones en coma flotante.
- **SYSmark :** se trata de un benchmark para evaluar CPU's bajo el sistema Windows. Hay diferentes versiones como la SYSmark98 y la SYSmark2002.
- **CPUmark99** : mide el rendimiento de los Pentium bajo Windows.
- **Wintune98** : dedicado a procesamientos de texto y hojas de calculo.
- **Winstone99, Multimediummark99, J.Mark 2.0, etc.**

Pentium® 4 processor	SYSmark 2002
1.5GHz	159
1.6GHz	166
1.7GHz	174
1.8GHz	179
1.9GHz	186
2GHz	193
2AGHz	212
2.2GHz	227
2.4GHz	242

Figura 2.6 Comparativa sobre diferentes frecuencias del Pentium IV del SYSmark2002.

2.5.5- Programas propios de fabricantes

Dentro de este tipo de programas está por ejemplo el índice iCOMP. Se crea en 1992 y proporciona una medición sencilla y relativa del rendimiento del microprocesador. No es un programa de pruebas, sino un conjunto de ellos que se utiliza para calcular un índice de rendimiento relativo que ayude a aquellos que vayan a comprar un PC a decidir qué microprocesador Intel satisface mejor sus necesidades.

Para elaborar el índice iCOMP, Intel toma uno de sus procesadores como referencia y le asigna el índice iCOMP 100. El procesador que se toma como referencia es el 486 SX a 25MHz. Los índices iCOMP de los demás procesadores se obtienen comparándoles con el microprocesador que se toma como referencia.

A continuación se presenta una comparativa el índice iCOMP 3.0, que incluye software y tecnología 3D, multimedia e Internet, para algunos de los procesadores de la familia Intel.

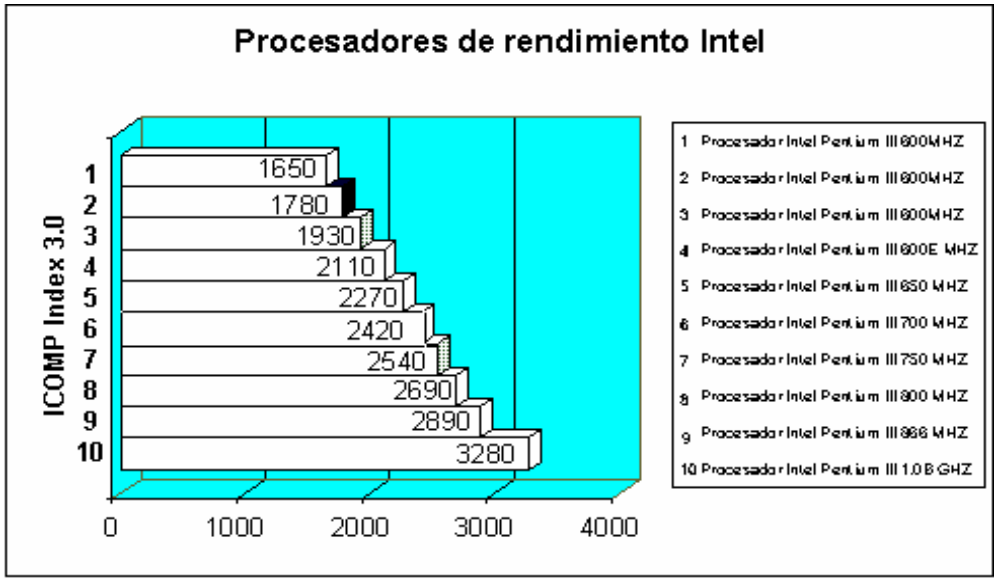


Figura 2.6 El índice iCOMP 3.0 basado en productividad, multimedia, 3D e internet.

2.6- LEYES DE AMDAHL

Las leyes de Amdahl evalúan las modificaciones en el rendimiento de un computador cuando se introducen mejoras o más recursos. El criterio fundamental de Amdahl es que lo que hay que mejorar o modificar siempre es lo que se usa más frecuentemente, ya que es lo que más afecta al rendimiento. En el estudio de las leyes de Amdahl se utilizan dos formulas:

$Aceleración_{\eta} = \frac{\eta_{sist.nuevo}}{\eta_{sist.antiguo}}$	$\text{Ganancia Velocidad} = \frac{\text{Tiempo}_{\text{sist.nuevo}}}{\text{Tiempo}_{\text{sist.antig uo}}}$

2.6.1- Primera ley de amdahl

La primera ley de Amdahl dice que el aumento del rendimiento debido a la inclusión de una mejora con un nuevo recurso en el sistema está limitado por el tiempo que se utiliza dicha mejora en la ejecución de la tarea.

$$T_{nuevo} = T_{antiguo} * \left(\frac{Fracción\ de\ tiempo\ mejorada}{Ganancia\ Velocidad} + Fracción\ de\ tiempo \text{ sin } mejora \right)$$

2.6.2- Segunda ley de amdahl

La segunda ley de Amdahl dice que cuando se introduce una mejora a un computador previamente mejorado, el incremento del rendimiento es menor que si se introduce la mejora sobre el sistema sin mejorar. Dicho de otra forma, la mejora incremental en la aceleración conseguida con la mejora de una parte se va reduciendo a medida que se van introduciendo nuevas mejoras.

EJEMPLO 11

Se cambia la ALU de un procesador dedicado a una tarea en la cual el 50% del tiempo lo hace la ALU. La nueva ALU es dos veces más rápida que la anterior. Calcular el tiempo de mejora.

$$\text{T con mejora} = \text{T sin mejojora}\left(\frac{0{,}5}{2} + 0{,}5\right)$$

T con mejora = T sin mejora * 0.75

Se ha mejorado el tiempo en un 75%.

Sustituimos la ALU por una nueva un 30% mayor. Si inicialmente un computador tardaba 2sg. en hacer un programa, ¿cuánto tarda ahora sabiendo que la ALU para este programa está funcionando el 50% del tiempo?

$$T_{nuevo} = T_{antiguo} * \left(\frac{Fracción\ de\ tiempo\ mejorada}{Ganancia\ Velocidad} + Fracción\ de\ tiempo \sin mejora\right)$$

$$T_{nuevo} = 2 * \left(\frac{0{,}5}{1{,}3} + 0{,}5\right) = 1{,}76 sg.$$

EJEMPLO 12

Calcular la aceleración del rendimiento de un sistema computador cuando se cambia la ALU del procesador por otra 10 veces más rápida, sabiendo que la ALU participa durante el 40% del tiempo que dura la tarea.

$$\text{Aceleración}_{\eta} = \frac{\eta_{\text{con mejora}}}{\eta_{\text{sin mejora}}} = \frac{T_{\text{sin mejora}}}{T_{\text{con mejora}}}$$

$$T_{\text{con mejora}} = T_{\text{sin mejora}} * (0.4/10 + 0.6) = 0.64 * T_{\text{sin mejora}}$$

$$\text{Aceleración}_{\eta} = \frac{T_{\text{sin mejora}}}{0.64 * T_{\text{sin mejora}}} = 1.56$$

Luego el rendimiento aumenta en un 56%.

2.7- MEDIA ARITMÉTICA Y GEOMETRICA

En la mayoría de las evaluaciones de rendimiento de computadores se mide el tiempo que tarda en ejecutar diversos programas y después se calcula la media. Hay dos tipos de media:

- ***Media aritmética***: se suman los tiempos de las diversas pruebas y se divide por el número de pruebas.

$$M_{\text{aritmética}} = \frac{\sum t_i}{n}$$

- ***Media geométrica***: se calcula la raíz del producto de todos los tiempos.

$$M_{\text{geométrica}} = \sqrt{\prod_{i=1}^{n} t_i}$$

La media aritmética no interesa en el calculo del rendimiento porque aunque proporciona una medida de lo que tarda el computador en ejecutar los programas, los resultados son contradictorios, por lo que se debe usar la media geométrica.

2.8- RELACION COSTE RENDIMIENTO

Aunque típicamente el rendimiento es un parámetro fundamental por el que se selecciona un computador, en la práctica el coste tiene una importancia decisiva. A la hora de elegir un computador, lo que determina su compra es la relación coste-rendimiento, es decir, el coste por unidad de rendimiento. La tendencia del mercado es construir computadores con mayor rendimiento y menor coste.

2.8.1- Análisis de la distribución de los costes de un computador

Vamos a analizar el coste de los componentes de un computador poniendo como ejemplo Pentium IV:

Placa DFI Pentium IV ATX NB72-SC	**94,00€**
CPU INTEL PIV 2,2GHZ	**228,00€**
Ventilador Extra PIV Superball	**9,00€**
DIMM 128 MB PC133 MHZ SDRAM	**24,50€**
Tarjeta Gráfica VGA TNT2 32MB AGP	**31,50€**
Disketera 3 1/2 HD 1,44 MB	**9,62€**
Caja Semitorre ATX LUJO P IV L2000-300W	**38,50€**
Monitor Color 15" Digital KFC (3 años de Garantía)	**105,00€**
Teclado 106 Teclas Win96 Multimedia PS2	**5,71€**
Tarjeta de sonido SOUND BLASTER 128 Bits PNP	**14,90€**
MODEM 56KB Interno V90 MOTOROLA	**13,25€**
CDROM 52x LG IDE OEM	**25,50€**
Disco Duro 40GB SEAGATE ATA100	**63,00€**
Altavoces 180 Watios AZUL 3D	**4,81€**
TOTAL	**667,29€**

Figura 2.7. Costes de un Pentium IV a 2,2 GHZ.

Porcentualmente:

Placa Base, CPU y Ventilador	50 %
Disketera, Teclado, Modem y Monitor	22 %
Caja semitorre	6 %
Tarjeta de sonido, Tarjeta Gráfica y Altavoces	8 %
Disco Duro y Memoria SDRAM	14 %

Figura 2.8 Análisis porcentual de los costes del computador.

Como conclusión observamos que la mitad del coste del computador recae en la CPU, su placa Base y el ventilador para refrigerarla. En los establecimientos informáticos a los precios de fabrica de los componentes deberemos incluir los costes directos de los distribuidores como son mano de obra, stocks y garantías. Asimismo hay que incluir los márgenes de beneficio que se lleva el distribuidor que están entre el 25 y el 40 % del coste del computador.

2.9- EJERCICIOS PROPUESTOS

EJERCICIO 1

Se propone a un 386 añadirle una memoria caché con una tasa de acierto de un 90%, deforma que, cuando el acceso se haga en la caché, el CPI de las instrucciones que afectan a la memoria se decrementa en una unidad. Al poner la caché, en los fallos se pierde un ciclo, es decir, se le suma al CPI 1 unidad. Se pide:

- Calcular el rendimiento del 386 sin caché.

- Calcular el rendimiento del 486 con caché.

- Calcular la relación de rendimientos entre el 386 con caché y el 386 sin caché.

Las diferentes instrucciones tienen la frecuencia y el CPI que se refleja en la tabla:

N° de instrucciones	Tipo de instrucciones	CPI386
20	**Carga**	**2**
10	**Almacenar**	**4**
15	**Reg/Reg**	**2**
8/7	**Salto condicional**	**9/3**
10	**Call**	**9**
30	**Operaciones Aritméticas**	**5**

EJERCICIO 2

Suponiendo que tenemos 2 maquinas con las siguientes características para un determinado programa R:

- Maquina A : Duración del ciclo de reloj de 23 ns. Con un CPI de 3,2

- Maquina B : Duración del ciclo de reloj de 15 ns. Con un CPI de 4

¿Cuál de las dos maquinas tiene mayor rendimiento para el programa R?

EJERCICIO 3

Estamos interesados en dos implementaciones de una máquina. Una con hardware especial de punto flotante y otra sin él. Considerar un programa P, con la siguiente mezcla de operaciones:

Multiplicación en punto flotante	10%
Suma en punto flotante	15%
División en punto flotante	5%
Instrucciones enteras	70%

La maquina MFP (máquinas con punto flotante), tiene hardware de punto flotante y además puede implementar directamente las operaciones en punto flotante.

Necesita el siguiente número de ciclos para cada clase de instrucción:

Multiplicación en punto flotante	6
Suma en punto flotante	4
División en punto flotante	20
Instrucciones enteras	2

La máquina MNFP (máquina sin puntos flotante) no tiene hardware de punto flotante y por ello debe las operaciones en punto flotante utilizando instrucciones enteras. Todas las instrucciones enteras necesitan dos ciclos de reloj. El número de instrucciones enteras necesarias para implementar cada una de las operaciones en punto flotante es como sigue:

Multiplicación en punto flotante	30
Suma en punto flotante	20
División en punto flotante	50

Ambas máquinas tienen una frecuencia de reloj de 100 MHz. Calcular las frecuencias en MIPS nativos para ambas máquinas.

EJERCICIO 4

Se va a mejorar una máquina y se barajan dos opciones: hacer que la ejecución de las instrucciones de multiplicación tarden 4 veces menos, o que la ejecución de las de acceso a memoria tarde 2 veces menos. Se ejecuta un programa de prueba antes de realizar la mejora y se obtienen las siguientes medidas de tiempo de uso de la CPU: el 20% del tiempo es utilizado para multiplicar, el 50% para instrucciones de acceso a memoria y el 30% restante para otras tareas.

¿Cuál será el incremento de velocidad si sólo se mejora la multiplicación? ¿Y si sólo se mejoran los accesos a memoria? ¿Y si se realizan ambas mejoras?

EJERCICIO 5

Se ejecutan sobre una máquina dos programas A y B utilizados como test para medir su rendimiento. Los recuentos de instrucciones tienen la siguiente distribución en ambos:

	PROGRAMA A	PROGRAMA B
Instrucciones de proceso	37%	48%
Instrucciones de transferencia	45%	36%
Instrucciones de salto	18%	16%

La máquina presenta los siguientes CPI (ciclos por instrucción) medios para cada grupo de instrucciones sin memoria caché de 2º nivel y con ella.

	CPI MEDIO	
	SIN CACHE DE 2º NIVEL	CON CACHE DE 2º NIVEL
Instrucciones de proceso	1.0	1.0
Instrucciones de transferencia	5.2	2.4
Instrucciones de salto	1.1	1.0

Determinar la ganancia de rendimiento (aceleración o speed up) que presenta la mejora de la jerarquía de memoria introducida en la máquina con respecto a la situación sin mejora.

EJERCICIO 6

Una vez graduado, el lector se preguntará cómo llegar a ser un líder en el diseño de computadores. Su estudio sobre la utilización de construcciones de los lenguajes de alto nivel sugiere que las llamadas a los procedimientos son una de las operaciones más caras. Suponga que ha inventado un esquema que reduce las operaciones de carga y almacenamiento normalmente asociadas con las llamadas y vueltas de procedimientos. Lo primero que hace es ejecutar algunos experimentos con y sin esta optimización. Sus experimentos utilizan el mismo compilador optimizador en ambas versiones del computador.

Los experimentos realizados revelan lo siguiente:

• La duración del ciclo de reloj de la versión no optimizada es el 5% más rápido.

• El 30% de las instrucciones de la versión no optimizada son operaciones de carga o almacenamiento.

• La versión optimizada ejecuta 1/3 menos de operaciones de carga y almacenamiento que la versión no optimizada. Para las demás instrucciones, el recuento de ejecución dinámica es inalterable.

• Todas las instrucciones (incluyendo las de carga y almacenamiento) emplean un ciclo de reloj.

¿Qué versión es más rápida? Justificar cuantitativamente la decisión.

LA FAMILIA X86. 3

3.1. DIFERENCIAS ENTRE MICROPROCESADOR Y MICROCONTROLADOR.

Las diferencias básicas entre un Microcontrolador y un Microprocesador están en los componentes de la arquitectura interna y el tipo de arquitectura. Como a continuación se describe en el punto 3.1.1.

Esto ha generado que gracias a los microprocesadores existan grandes computadores que nos ayuden a predecir los fenómenos metereologicos o realizar simulaciones y que gracias a los microcontroladores formando parte de sistemas embebidos, conducir un vehículo sea mas seguro o incluso sea muy sencillo lavar la ropa en una lavadora.

Y gracias al gran desarrollo que los microprocesadores han experimentado, hoy es posible que casi todo el mundo pueda disfrutar su propio ordenador personal, y que incluso los ordenadores personales de hoy tengan mas potencia que los primeros grandes ordenadores.

3.1.1. Arquitectura interna:

El **microprocesador** es un circuito integrado formado por transistores. Su arquitectura interna contiene la unidad central de proceso (CPU), que a su vez esta compuesta por la unidad de control y el camino de datos.

La unidad de control traduce las instrucciones que hay en memoria y el camino de datos las ejecuta.

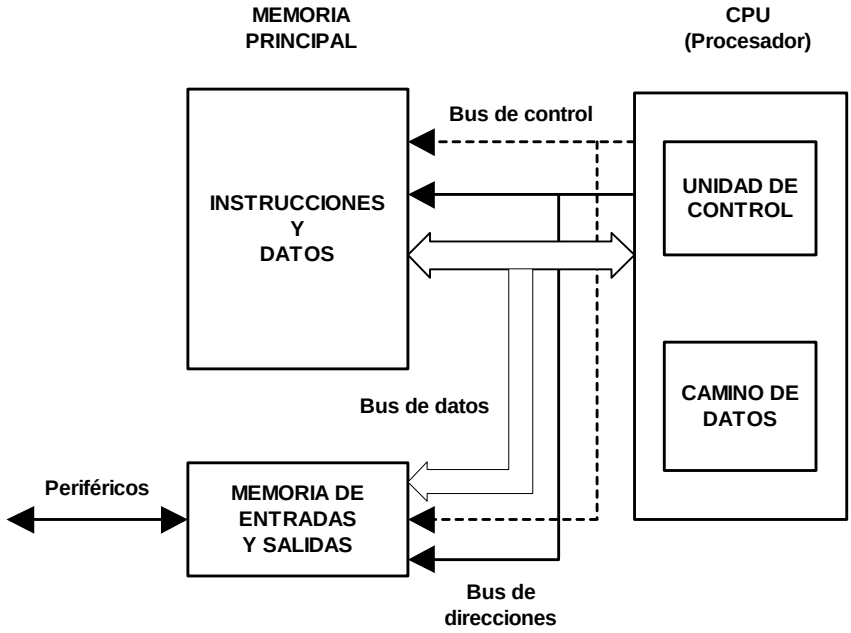


Figura 3.1 - Bloques que componen un microprocesador.

El **microcontrolador** es un circuito integrado programable que contiene todos los componentes de un procesador. En su arquitectura interna además de la CPU, esta la memoria, los módulos de entrada salida y todos los recursos complementarios.

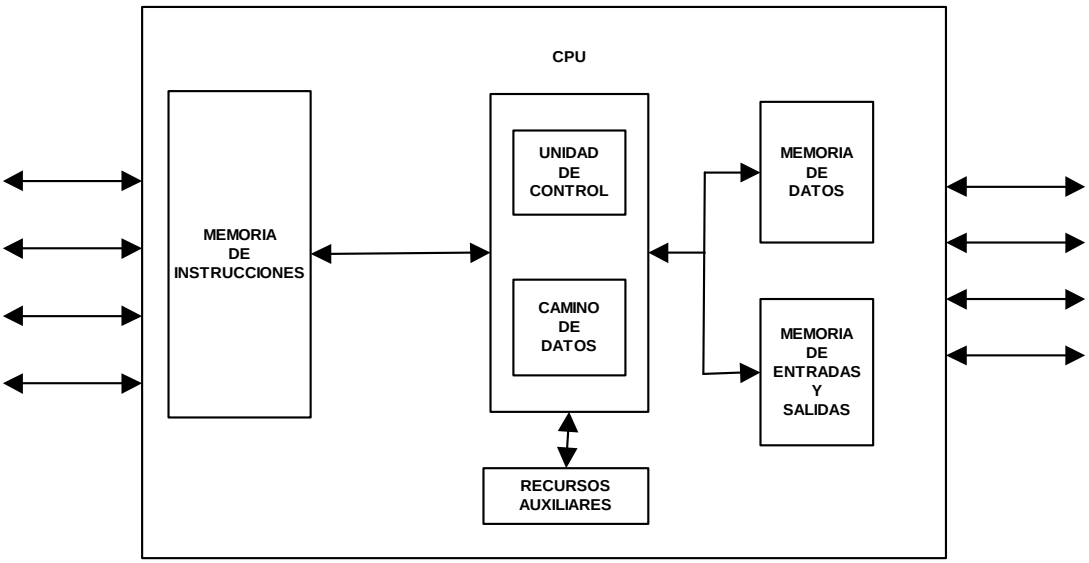


Figura 3.2 - Bloques que forman un microcontrolador.

3.1.2. Arquitectura Abierta Versus Arquitectura Cerrada

El microprocesador forma una arquitectura abierta porque el computador es configurable por el usuario y puede realizar varias tareas. El microprocesador ofrece al sistema los buses de direcciones, datos y control.

Con microprocesadores se pueden formar sistemas de uso general. Por ejemplo el computador que utilizamos normalmente encima de la mesa.

El microcontrolador forma una arquitectura cerrada porque no es configurable por el usuario (es un computador completo) y realiza una sola tarea debido a que sus prestaciones son limitadas. El microcontrolador ofrece al sistema las líneas que controlan a los periféricos.

Con el microcontrolador se pueden formar sistemas embebidos (van dentro del dispositivo al que controlan).

Si nos paramos por un momento a pensar, podemos ver microcontroladores por todas partes. Cualquier electrodoméstico que realice varias tareas, como si siguiera varios programas lleva uno dentro. Por ejemplo una lavadora, en función de lo que queremos lavar elegimos un "programa" u otro, eso es gracias a que el microcontrolador que tiene la lavadora es el encargado de controlar que la lavadora utilice una determinada cantidad de agua, a una temperatura determinada, etc.

Y hay sistemas que no solo llevan uno. En un coche puede haber más de 150. Desde los encargados de controlar el motor, pasando por los encargados del sistema de frenos antibloqueo, o incluso la climatización.

3.2. IMPORTANCIA DEL AVANCE DE LA TECNOLOGÍA:

Han pasado más de 25 años desde que Intel diseñara el primer microprocesador, siendo la compañía pionera en el campo de la fabricación de estos productos, y que actualmente cuenta con más del 90 por ciento del mercado. Y cada vez más, hay otras compañías, como AMD, que están creando productos que rivalizan con los de Intel.

El avance en el mundo de la informática se produjo gracias a la tecnología electrónica, ya que mediante ella se mejoro el software de los primeros procesadores y se incremento su rendimiento. Y en un futuro, puede que gracias a la nanotecnología se fabriquen ordenadores de orden molecular, con mucha mayor potencia y un menor consumo que los actuales.

3.2.1. Ley de Moore.

El Dr. Gordon Moore, uno de los fundadores de Intel Corporation, formuló en el año 1965 una ley que se ha venido a conocer como la "Ley de Moore" para establecer los avances tecnológicos.

Esta ley establece que cada dieciocho meses se dobla aproximadamente el número de transistores que hay en un circuito integrado. Esto va unido a un aumento de potencia y rendimiento significativos.

Según esta ley, se predijo que para el año 2011 existiría un circuito integrado compuesto por 1,5 Gigas de transistores con una frecuencia de trabajo de 10 Ghz, tecnología de 0,07 micras y un rendimiento de 10000 Mips. Esto supone u gran aumento en el número de transistores teniendo en cuenta que en el año 1999 el Pentium III contaba con 9,5 millones.

Hoy en día las recientes innovaciones tecnológicas introducidas por IBM e Intel han provocado una serie de especulaciones sobre la posibilidad de que la tecnología de la computación se desarrolle a una velocidad mucho mayor de lo que predice esta ley. Esto será debido a la utilización de una tecnología distinta de la actual.

3.2.2. Limitaciones en el avance de la tecnología.

La tecnología actual se ve limitada por:

- **La temperatura:** la miniaturización y la frecuencia actual de los procesadores hace que se genere mucho calor. La solución a este problema son sistemas de refrigeración más potentes y eficaces.

- **El espacio**: Al aumentar el número de transistores, aumenta la superficie del procesador. También, al aumentar la escala de integración, los transistores están más juntos. Debido a esto se producen problemas, como:

  - **Fenómenos de inversión**: Al comunicarse transistores relativamente lejanos a frecuencias elevadas, puede ocurrir que el transistor emisor envíe un 1 y el transistor receptor reciba un 0.

  - **Electromigración**: debido a la miniaturización de los transistores pueden migrar los electrones de un material al otro del que están compuestos los mismos.

3.3. CONCEPTO DE FAMILIA:

Una familia de microprocesadores es un conjunto de modelos ligados por algunas características comunes.

Las dos características comunes de la familia X86 son:

- Compatibilidad con el software descendente.
- Aumento de prestaciones en los nuevos modelos con respecto a los anteriores.

Estas características ofrecen una ventaja comercial, al actualizar el procesador se puede seguir usando el software adquirido y desarrollado para los procesadores anteriores. Es la clave principal del éxito de esta familia de procesadores.

Pero también una desventaja tecnológica, ya que se ha tenido que mantener la arquitectura y el repertorio de instrucciones básico de los modelos anteriores, y al tener que mantener el núcleo, los fallos existentes también se han ido arrastrando.

3.3.1. Familia x86.

Los procesadores de esta familia son del fabricante Intel, y su arquitectura responde al nombre IA-32. El núcleo de esta arquitectura es común para todos los microprocesadores y cada nuevo modelo añade extensiones y recursos a dicho núcleo.

El juego de instrucciones de esta arquitectura es de tipo CISC (repertorio amplio de instrucciones), con lo que cada instrucción es de tipo complejo y se ejecuta en varios ciclos de reloj.

Esta familia ha tenido un éxito comercial enorme y también una gran aceptación, gracias a su alto rendimiento. En cuanto aparece una gran novedad Intel la incorpora a sus nuevos modelos.

La razón por la cual se estudian estos procesadores es porque son los más usados actualmente y parece que lo van a seguir siendo en el futuro ya que la de Intel es una arquitectura progresiva y moderna. Pero el hecho de que sean los más utilizados no quiere decir que sean superiores, existen otros procesadores que tecnológicamente no envidian en nada a los de Intel o incluso son superiores. Sin embargo, el software de estos computadores no es compatible con el tipo de instrucciones de los 80x86 utilizados por Intel.

3.4. GENERACIONES:

La familia x86 esta compuesta por seis generaciones. Dentro de cada generación hay diferentes modelos que varían en la relación precio/prestaciones y el consumo. Cada modelo esta orientado a cubrir el sector de mercado donde ese factor sea critico. Todos son compatibles entre sí.

3.4.1. La primera generación: 8086

Aparecen en los meses de Junio de 1978-79 los procesadores 8086 y 8088. Actualmente se utiliza el 8086 para sistemas embebidos.

Para permitir la compatibilidad y la creación de sistemas informáticos integrados en esta familia se disponen de diversos coprocesadores:

- 8089 coprocesador de E/S.
- 8087 coprocesador matemático de coma flotante.

Por eso también el 8088 se diseña con un bus de 8 bits en vez de 16 bits como el que tiene el 8086. Para resolver esta diferencia Intel divide cada procesador 8088 y 8086 en dos Sub-procesadores:

- Unidad de Ejecución (EU): realiza todas las operaciones.
- Unidad de Interfaz con el Bus (BIU): accede a datos e instrucciones del mundo exterior.

En ambos procesadores las Unidades de Ejecución son idénticas pero varia la Unidad de Interfaz con el bus. Así se consiguió un ahorro de esfuerzo para producir el 8088.

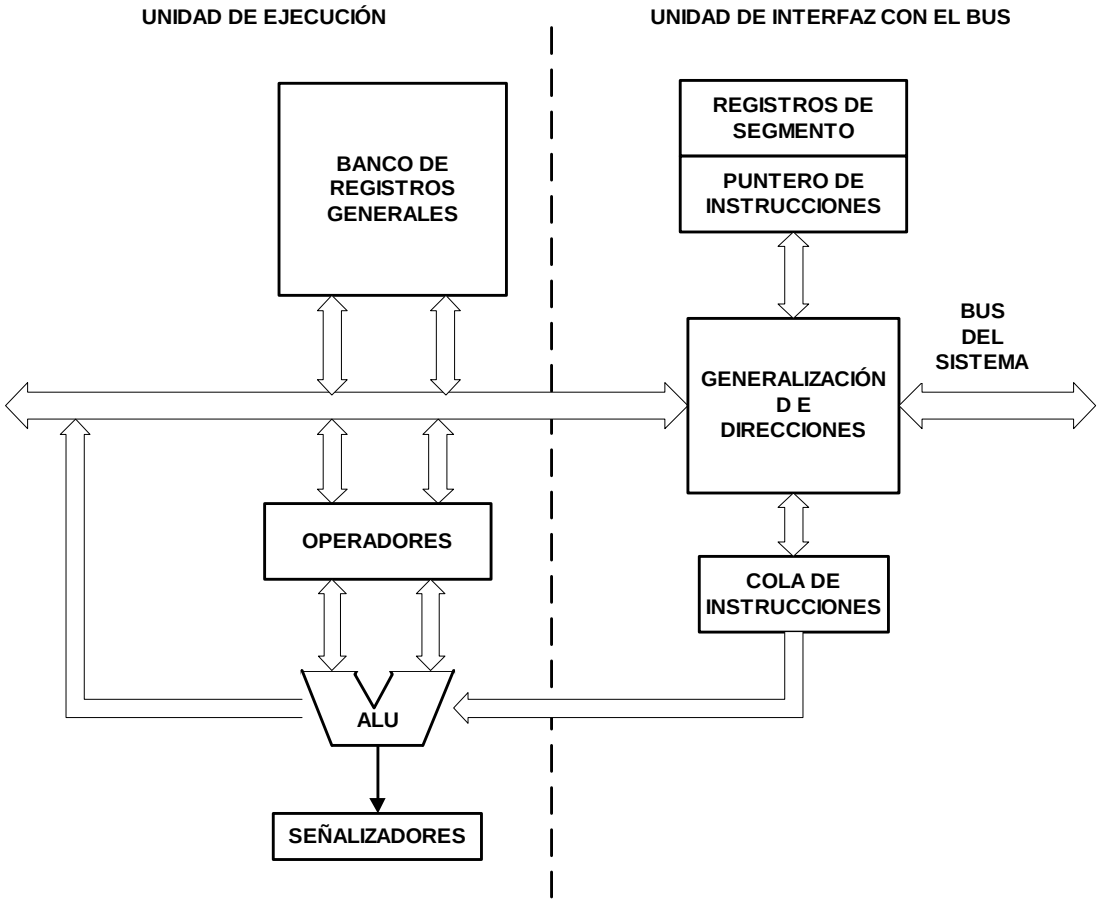


Figura 3.3 - Arquitectura interna del 8086

Estos procesadores tienen 27 modos de direccionamiento para localizar un operando de una instrucción.

Las principales aportaciones son:

- Gestión de memoria
- Segmentación de 2 etapas: Buscar instrucción y ejecutar.
- Interrupciones sectorizadas multinivel.

3.4.2. La segunda generación: 80286

Aparecen a principios de febrero de 1982. Se caracterizan por poseer dos modos de funcionamiento completamente diferenciados:

- **Modo Real:** Se comporta igual que un 8086 pero con mayor velocidad, al ser conectado a la alimentación arranca en este modo.

- **Modo Protegido:** funciona con capacidad de proceso multitarea y memoria virtual. Este modo es propio del 286 donde todas las extensiones se ponen en marcha, por lo que pierde la compatibilidad con los procesadores anteriores.

  Cuando la CPU esta en modo protegido, los programas de usuario tienen un acceso limitado al juego de instrucciones; solo el proceso supervisor esta capacitado para realizar ciertas tareas. Así se evitan posibles conflictos entre los distintos programas de usuario, con lo que el fallo de un proceso no afecta al resto.

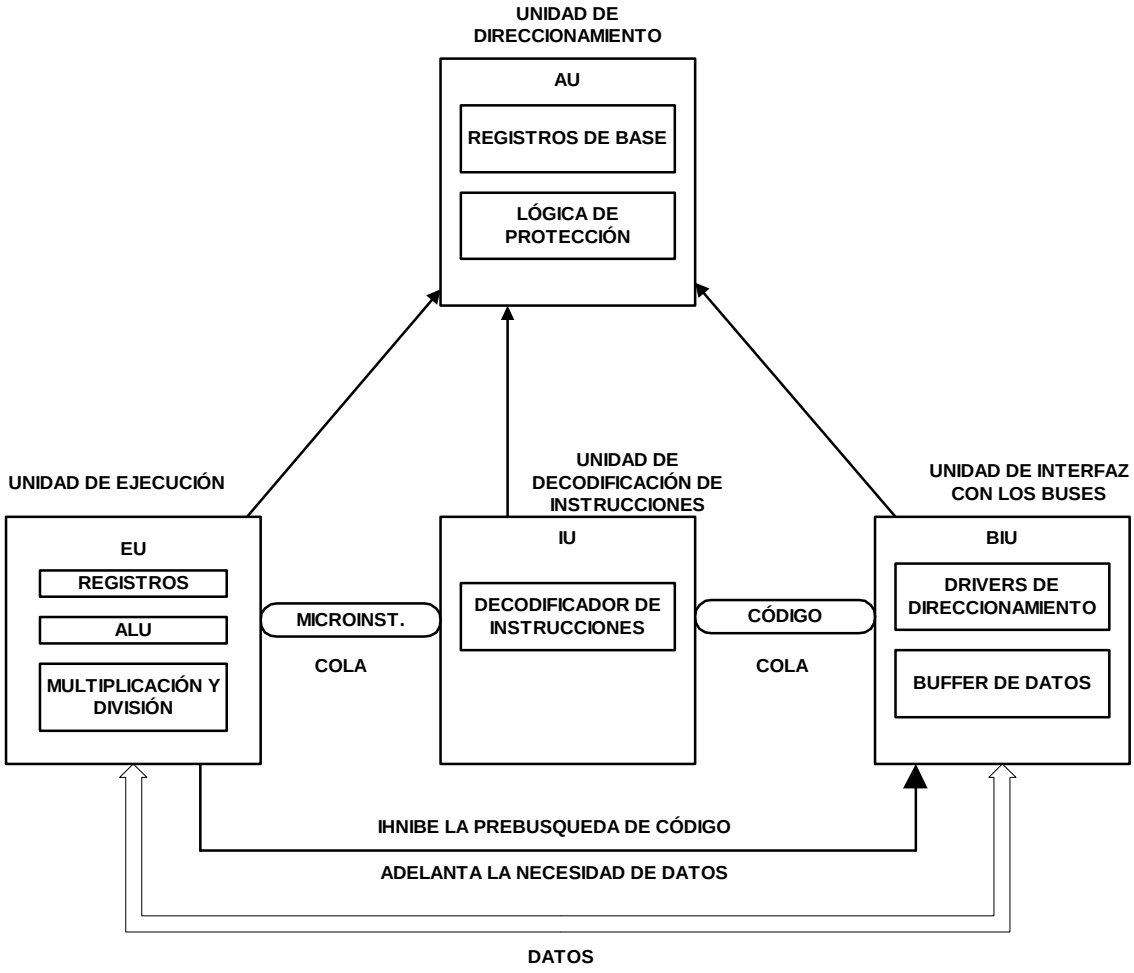


Figura 3.4 - Arquitectura interna del 80286

Las principales aportaciones son:

- Memoria Virtual hasta 1 GB y Memoria Física hasta 16 MB.
- Admitía multitarea e introdujo sistemas de protección.
- Cuatro niveles de privilegio.
- Aumenta la segmentación a cuatro capas.

3.4.3. La tercera generación: 80386

Es el primer procesador de 32 bits del mundo y ha llegado a ser un estándar en la industria.

Tiene tres modos posibles de funcionamiento:

- **Modo Real:** Compatible con el 8086.
- **Modo Protegido:** propio que le permite romper la barrera de los tradicionales segmentos.
- **Modo Virtual 86:** puede emular el funcionamiento simultáneo de varios 8086.

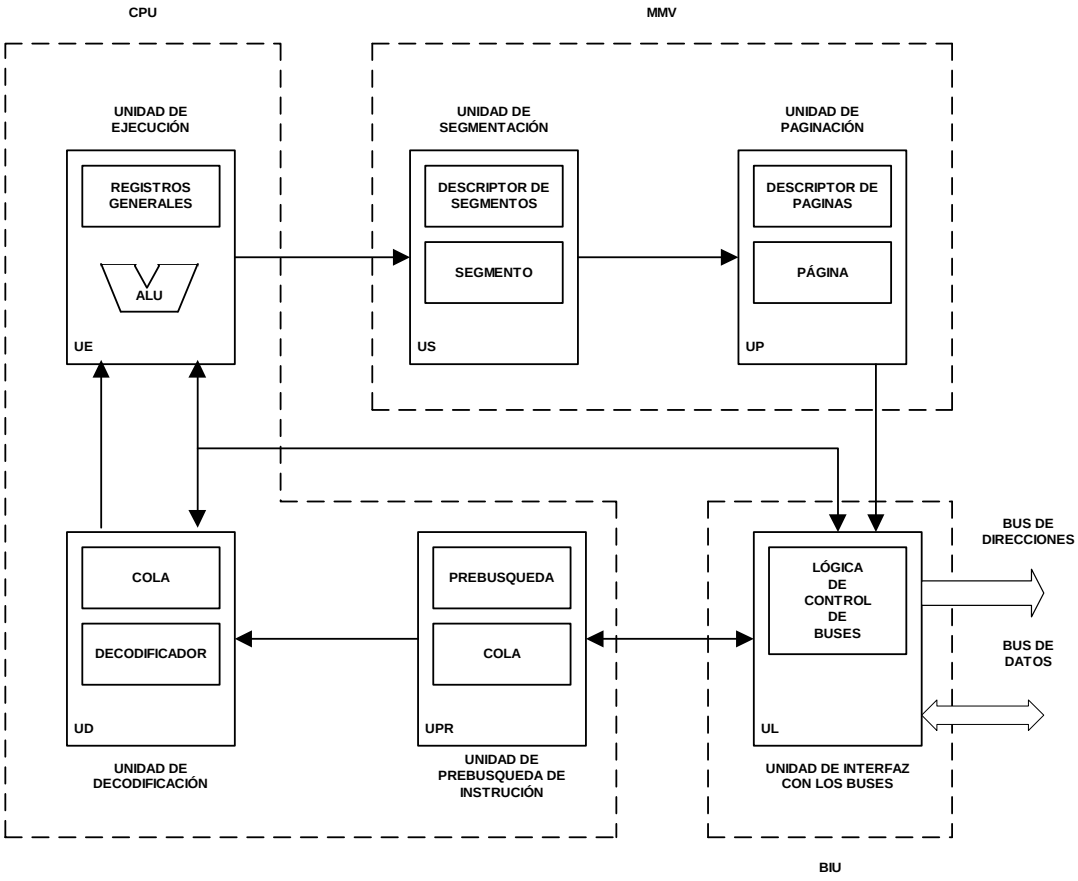


Figura 3.5 - Arquitectura interna del 80386.

El sistema de paginación es transparente a la segmentación y permite el manejo de direcciones físicas. Cada segmento se divide en una o más paginas de cuatro kilobytes.

La unidad de segmentación provee cuatro niveles de protección para aislar y proteger aplicaciones y el sistema operativo.

Para facilitar diseños de hardware de alto rendimiento, la interfaz con el bus (BIU) del 80386 ofrece pipelining de direcciones, tamaño dinámico del ancho del bus de datos (16 o 32 bits) y señales de habilitación de bytes por cada byte del bus de datos.

3.4.3.1. Versiones:

Además de las diferentes frecuencias de funcionamiento, Intel saco dos versiones especiales:

- **386 SX:** Versión intermedia de precio y prestaciones entre el 286 y el 386 DX. El bus de datos tiene sólo 16 bits.

- **386 SL:** Para uso en ordenadores portátiles, incluye recursos para minimizar el ahorro de energía.

3.4.4. La cuarta generación: 80486

El 80486 es una versión mejorada del 80386 que además tiene integrada una cache de 8 Kbytes y un coprocesador matemático 80387, con lo que se consigue que casi la mitad de las instrucciones del 486 se ejecuten en un periodo de reloj en vez de los dos periodos que requiere el 386.

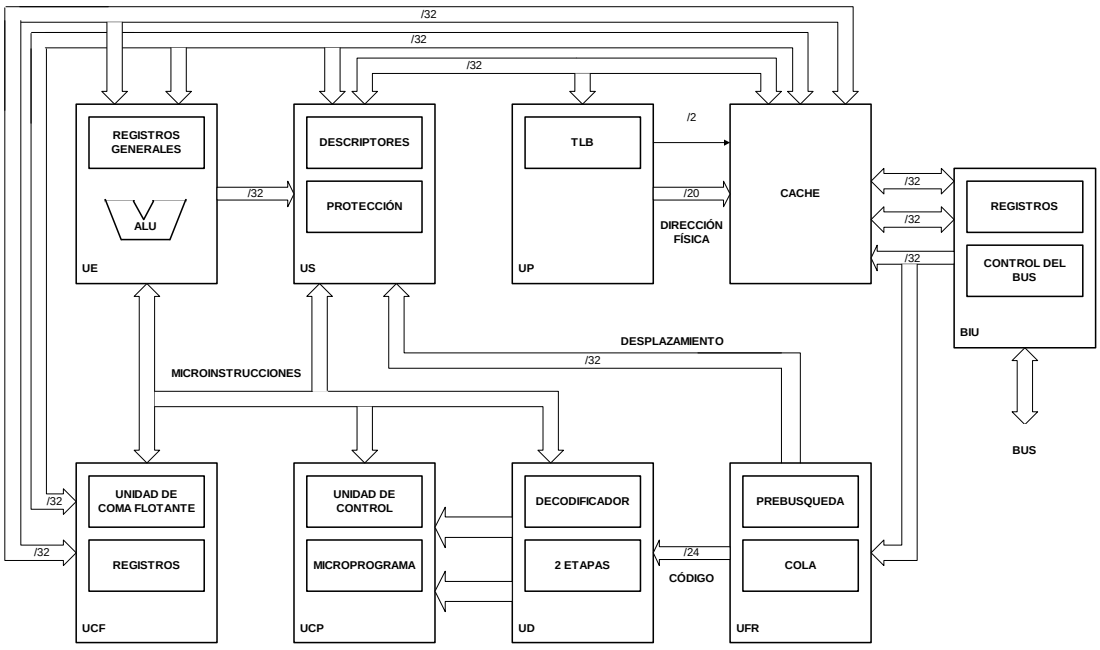


Figura 3.6 - Arquitectura interna del 80486.

Esta compuesto de nueve unidades funcionales:

- Unidad de interconexión con el bus.
- Unidad de prebúsqueda.
- Unidad de decodificación.
- Unidad de control.
- Unidades de enteros y coma flotante.
- Unidades de segmentación y paginación.
- Unidad de caché.

3.4.4.1. Versiones

Hay cinco versiones:

- **DX:** Doble de la velocidad del 80386, manteniendo la compatibilidad con los procesadores precedentes. Máximo consumo de 5W.
- **SX:** Solución de menor coste que no incluye el coprocesador matemático.
- **DX2:** Posee duplicador de frecuencia interno, el procesador funciona al doble de velocidad.
- **SL:** Reduce la tensión de trabajo del procesador para ahorro de energía.
- **DX4:** Triplica la frecuencia de reloj y aumenta el tamaño del cache interno a 16 Kbytes.

3.4.5. La quinta generación: Pentium

Las ventajas que aporta son:

- **Supersegmentación con 14 etapas.** Técnicas de predicción de saltos condicionales para evitar introducir demasiadas burbujas.

- **Arquitectura superescalar.** Dos cauces de datos, en un ciclo se ejecutan más de una instrucción.

- **Aumento de cache.** Caché de primer nivel (L1) y cache de segundo nivel (L2). La caché L1 se divide en dos partes independientes para datos para instrucciones de 8 KB cada una, con lo que es posible acceder a un dato y una instrucción en paralelo.

- El chip se empaqueta en **formato PGA** (Pin Gris Array) de 273 pines.

Las 7 unidades funcionales que aportan características específicas e innovadoras al procesador consiguiendo altas prestaciones, compatibilidad y mantenimiento de la integridad de los datos son:

- **Unidad de enteros Superescalar:** consiste en dos unidades de enteros de 32 bits que operan en paralelo.

- **Unidades de memoria caché:** están subdivididas en dos memorias caché independientes, una para datos y otra para instrucciones.

- **Unidad de interconexión con el bus:** presenta un bus de datos de 64 bits con lo que se obtiene una velocidad de transferencia de 538 Mbytes/s.

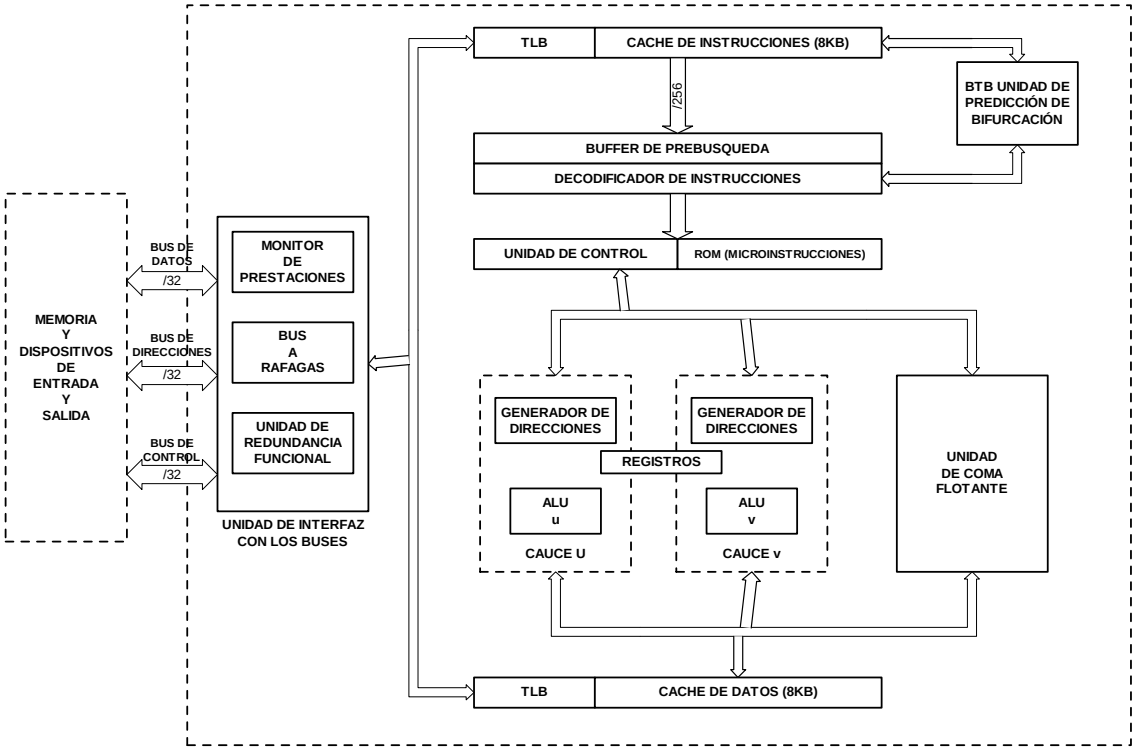


Fig. 3.7 - Arquitectura interna del Pentium.

- **Monitor de prestaciones:** consta de una serie de controladores internos y unidades de rastreo para evitar que se pierda gran cantidad de tiempo en ciertas rutinas o secciones de código.

- **Unidad de redundancia funcional:** consiste en una serie de técnicas para asegurar la integridad de los datos.

- **Unidad de predicción de bifurcaciones:** consta de una caché específica encargada de hacer una predicción dinámica de los saltos condicionales.

- **Unidad de coma flotante:** ha sido mejorada respecto del 486 incorporando un cauce segmentado de instrucciones de 8 etapas.

- **Vías de acceso múltiple:** proporcionan una arquitectura superescalar que tiene la habilidad de ejecutar más de una instrucción por cada ciclo de reloj.

3.4.6. La sexta generación: Itanium

Nace de la colaboración de Intel y HP. Es el primer procesador con arquitectura de 64 bits. Es compatible con la familia x86, pero va a ejecutar más lentamente procesos de 32 bits.

Su arquitectura se denomina EPIC (Explicity Parallel Instruction Computing, o proceso de instrucciones explícitamente paralelo). Esta arquitectura le permite ejecutar hasta 6 instrucciones en paralelo por ciclo de reloj. En tiempo de compilación, el compilador decide cuales son las instrucciones que se pueden ejecutar en paralelo sin conflictos.

Esto es posible gracias a la duplicidad de unidades funcionales dentro del mismo procesador. En el Itanium hay 6 unidades de suma, dos de coma flotante y cuatro de enteros.

La arquitectura del procesador Itanium incluye también características únicas de confiabilidad a través de Enhanced Machine Check Architecture, que permite detección, corrección y registro de errores, además de características Error Correcting Code (ECC) y de comprobación de paridad.

Este procesador tiene memoria cache de 3 niveles, L1 y L2 dentro del procesador, y L3 en el encapsulado. Esta cache L3 es de 2 o 4 Megas. Tiene tecnología BSB (Back side Bus) a 12,8Gb/s. El tiempo de latencia se reduce a solo 15 ciclos de reloj, en vez de los 100 -150 ciclos de un procesador normal.

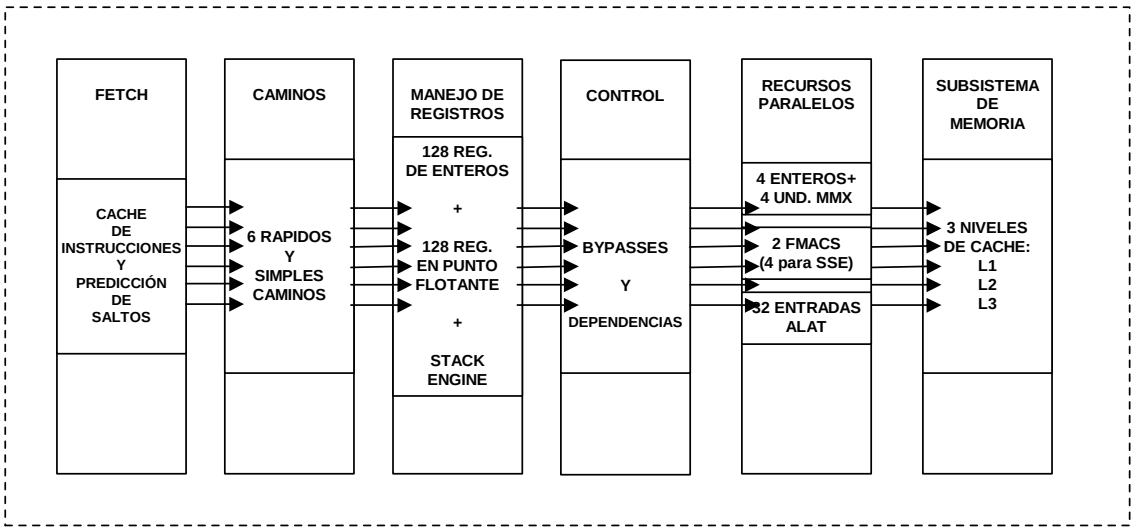


Fig. 3.8 - Arquitectura interna del Itanium.

Este procesador tiene como aplicaciones fundamentales aquellas que necesiten cómputo empresarial y de alto desempeño más exigentes. Gracias a la arquitectura de 64 bits ofrece grandes recursos para el procesamiento de Terabytes de datos, la aceleración de compras y transacciones en línea protegidas, y el procesamiento de cálculos complejos.

Estas características contemplan las necesidades crecientes de comunicaciones, almacenamiento, análisis y seguridad de datos al tiempo que ofrecen ventajas de desempeño, escalabilidad y confiabilidad a costos significativamente más bajos que las ofertas propietarias. Entre los segmentos de las aplicaciones se cuentan bases de datos grandes, obtención de datos, transacciones de seguridad de e-Commerce e ingeniería mecánica asistida por ordenador, además de computación científica.

Cuatro sistemas operativos ofrecerán compatibilidad con sistemas basados en el procesador Itanium, entre otros la plataforma Microsoft Windows* (Edición de 64 bits* para estaciones de trabajo y Windows Advanced Server Limited Edition 2002* para servidores); HP-UX 11i v1.5* de Hewlett-Packard, AIX-5L* de IBM y Linux. Caldera International*, Red Hat*, SuSE Linux* y Turbolinux* planean ofrecer versiones de 64 bits del sistema operativo Linux.

3.5. Otros procesadores:

A pesar de ser Intel el fabricante dominante en el mercado de procesadores para PC, no se debe olvidar que también hay otros fabricantes en el mismo. Como principal competidor actual de Intel en el mercado del PC, podemos resaltar a AMD (Advanced Micro Devices).

Por ejemplo podemos citar la serie K7, con sus versiones Duron y Athlon como competidores del Pentium III celeron y Pentium III respectivamente.

Transmeta ha sacado un procesador orientado al mercado de los ordenadores portátiles, distinguiéndose frente a los procesadores de Intel por su bajo consumo. Y que también se va a integrar en dispositivos como los Tablet PC.

VIA Technologies, fabricante conocido en nuestro mercado por fabricar placas base, ha desarrollado también otro procesador a 800 MHz en el año 2001.

También tenemos otros fabricantes como Cyrix o Motorola, que han intentado copar parte del mercado de Intel, introduciéndose en nichos de mercado como el de los microcontroladores o los equipos de gama baja.

3.6. Anexo (tablas de datos)

GENERACIÓN	PROCESADOR	AÑO	Nº TRANSISTORES	TECNOLOGÍA	FRECUENCIA	ALIMENTACIÓN	POTENCIA	RENDIMIENTO	CARACTERÍSTICAS RELEVANTES
1º	8086	1978	29.000	1.5 μm	5 - 8 - 12 MHz	+5v	20w	0.33-0.75 MIPS-VAX	Gestión de memoria. Interrupciones vectorizadas. Segmentación a 2 etapas.
2º	80286	1982	34.000	1.5 μm	6 - 12 MHz	+5v	25w	0.9-2.6 MIPS-VAX	Modo Real: 8086. Modo Protegido: Men Virtual. Niveles privilegio. Multitarea.
3º	80386	1885	275.000	1 μm	16 - 32 MHz	+5v	2.5w	5-11.4 MIPS-VAX	1º microprocesador de 32 bits. Segmentación Paginación Modo Protegido Memoria Virtual 64 TB
4º	80486	1989	$1_1$200.000	1 μm	100 MHz DX2 DX4	+5v +3.3v	3w	20-41 MIPS-VAX	Coprocesador integrado. Cache integrado. Bus a ráfagas. L1, asociativa de 4 vías y escritura inmediata.
5º	Pentium	1993	$3_1$100.000	0.6 μm	60 - 133 MHz	+5v +3.3v +2.9v	13w	112 MIPS-VAX 64 SPEC int_{92}	Bus de datos de 64 bits. Caches independientes, 2 vías y escritura obligatoria. Predic. de saltos BTB-256. Superescalar.

GENERACIÓN	PROCESADOR	AÑO	Nº TRANSISTORES	TECNOLOGÍA	FRECUENCIA	ALIMENTACIÓN	POTENCIA	RENDIMIENTO	CARACTERÍSTICAS RELEVANTES
5º	Pentium PRO	1995	5$_1$500.000	0.35 µm	200 MHz	+2.9v	20w	220 SPEC int$_{92}$	Incluye cache L2 interna. Supersegmentación: 14 etapas. BTB-512 L1-2 vías (inst) y 4 vías (datos) Ejecución fuera de orden.
	Pentium MXX	1997	4$_1$500.000	0.35 µm	233 MHz	+2.8v CPU +3.3v EXT	14w	7$_1$12 SPEC int$_{95}$	57 instrucciones MMX. Bus de direcciones 32 bits. Técnicas SIMD. Registros MMX MM0-MM7 (64)
	Pentium II	1997	7$_1$500.000	0.25 µm	>300 Mhz	+2.8v CPU +3.3v EXT	37w	11$_1$6 SPEC int$_{95}$	Rediseño del PCB. Hasta 4 Pentium paralelos Bus del sistema a 100 MHz
	Pentium III	1999	9$_1$500.000	0.13 µm	1000 MHz	+2v CPU +3.3v EXT	18w	410 SPEC int$_{2000}$	70 nuevas instrucciones MMX 17 modelos en 1999. Bus del sistema 100-133 MHz. Chipset 810/20/40
	Pentium IV	2000	42$_1$000.000	0.13 µm	2.4 GHz	+1.7v	>50w	Igual al P.III	144 nuevas inst. SSE2. Apli. Internet y multimedia. Chipset 850. Bus del sistema 400 MHz Supersegmentación: 20 etapas.
6º	Itanium	2001	25 M CPU 300 M Caches	0.13 µm	800 MHz	+1.7 v	>100w	En análisis hoy en día	Primer procesador de 64 bits. Puede realizar hasta 6 inst. Dispone de cache de 3º nivel.

CARACTERÍSTICAS DE LOS SISTEMAS OPERATIVOS PARA LOS MICRPROCESADORES AVANZADOS

4

4.1- CONCEPTOS BÁSICOS SOBRE SISTEMAS OPERATIVOS.

Definimos sistema operativo como un conjunto de programas implementados tanto en software como en firmware que hacen asequible el hardware, de forma que lo hacen disponible de la manera más adecuada al usuario y **aumentan su rendimiento total**, es decir, su rendimiento específico o **throughput** (volumen de trabajo por unidad de tiempo) y su disponibilidad (tiempo de respuesta mínimo).

Un ordenador no es una máquina dedicada sino que esta construido para un propósito general con el fin de ejecutar aplicaciones de diversa índole. En líneas generales el ordenador dispone además de una unidad central de proceso y de la memoria, de diversos periféricos (pantallas, impresoras, teclados, etc) que son compartidos en gran medida por todas las aplicaciones porque todas las aplicaciones de un computador utilizan los mismos recursos físicos y los mismos recursos lógicos.

Los **sistemas operativos nacen** porque sería extremadamente complejo que cada una de las aplicaciones soportadas por el ordenador contuviese todos los programas necesarios para manejar el hardware y el software que comparten muchas aplicaciones.

Además el sistema operativo hace de intermediario entre el hardware y el usuario creando un entorno adecuado que se adapte a los diferentes usuarios y aplicaciones.

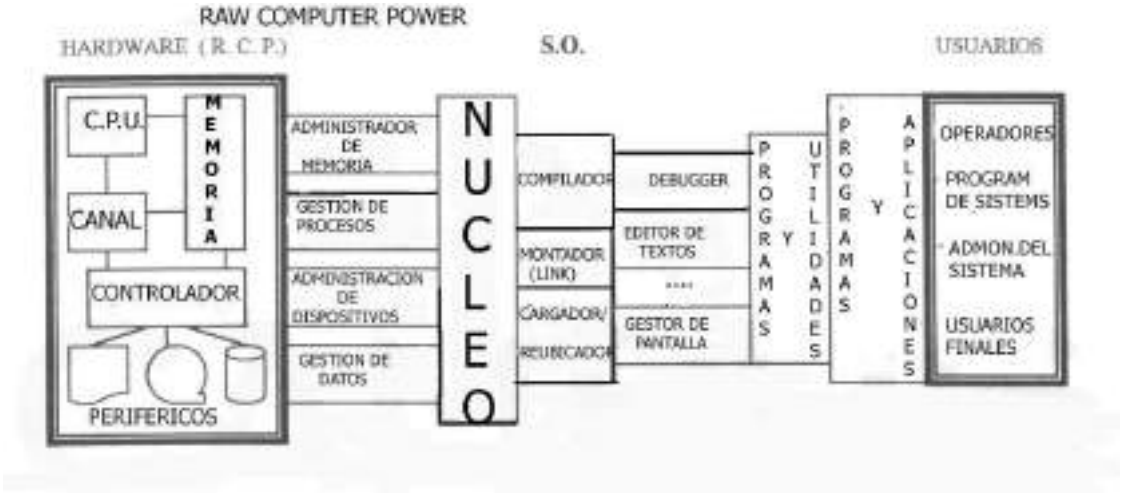


Figura 4.1. Descripción interna del S.O.

Por tanto **el sistema operativo** es un **gestor de recursos**, tratando de sacar de ellos el mayor rendimiento posible, por lo que debe poseer las siguientes **funciones**; **organiza** los archivos en diversos dispositivos de almacenamiento; **supervisa y gestiona** ejecuciones, **errores** hardware y la **pérdida** de datos; **coordina y manipula** el hardware de la computadora y tiene que **evitar** usos inadecuados además de **responder** a cualquier evento que se produzca.

4.1.1- Clasificación de los sistemas operativos.

Existen muchas clasificaciones diferentes de sistemas operativos dependiendo del criterio en el que se basen. Nosotros los clasificaremos basándonos en dos criterios:

1º- Según el punto de vista del usuario:

Existen los **SISTEMAS MONOUSUARIO** en los cuales la máquina virtual (la presentada por el S.O. al usuario) tiene un solo usuario y generalmente está dedicada a una sola función. Por tanto el interface del S.O. constará básicamente de un **gestor de ficheros** sencillo, utilidades que proporcionen **facilidades de E/S** y un intérprete de comandos sencillo, es decir un **OSCL**(Lenguaje de Control del S.O.); y sus **cualidades** son fiabilidad, eficacia, sencillez de uso y facilidad de extensión.

Se puede representar de las siguientes maneras según el modo en que se trabaje:

Figura 4.2. Representaciones Batch y On-Line de sistemas monousuario.

También existen los **SISTEMAS DE TIEMPO REAL** que tienen en común la necesidad de dar respuestas a unas entradas con un tiempo de proceso informático limitado**.** Por los que sus **funciones** principales serán interactuar con dispositivos externos (sensores, válvulas,...), reacción inmediata ante cualquier suceso externo, registrar la información (fichero registro), tener noción de tratamientos prioritarios y realizar una planificación eficiente, siempre teniendo en cuenta el tiempo físico (tiempo real)

Su **cualidad** principal debe ser la fiabilidad de forma que garantice la seguridad del sistema en cualquier caso (tolerancia a fallos) debiendo ofrecer unos servicios mínimos (disponibilidad) Los sistemas en tiempo real se utilizan en muchos campos como gestión de centralitas telefónicas, pilotaje de aviones, seguimiento de trayectorias, vigilancia médica, control de robots, ..., y control de procesos en general..

En general responden al siguiente esquema:

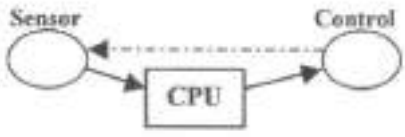


Figura 4.3. Esquema de un sistema de tiempo real.

Como evolución de los sistemas operativos aparecieron los **SISTEMAS TRANSACCIONALES** que se caracterizan por poseer las siguientes **funciones**; gestionar un gran volumen de información (Bases de Datos) desde distintos y numerosos puntos de acceso y de transacciones desarrollándose simultáneamente (de forma paralela → concurrencia) y por la ejecución de operaciones predefinidas, muy a menudo interactivas.

Sus **cualidades** son fiabilidad y disponibilidad.(servicios mínimos)

Son ejemplos de sistemas transaccionales los sistemas de reservas de plazas en líneas aéreas, gestión de cuentas bancarias, consulta de documentos, etc...

Más tarde aparecieron los **SISTEMAS TS/MULTIPROGRAMADOS**, donde **los sistemas Time-Sharing** se caracterizan por prestar servicio a un conjunto de usuarios simultáneamente, dividiendo el tiempo de utilización de la máquina en quantums para cada usuario.

Mientras que **los sistemas multiprogramados** se caracterizan por la ejecución simultanea de varios programas (procesamiento concurrente). En ambos sistemas el S.O. será el encargado de conmutar los recursos entre los diversos procesos y/o usuarios.

Por tanto las **funciones** que debe proporcionar el S.O. serán todas las de los sistemas monousuario y las de los sistemas transaccionales: definición de la máquina virtual a presentar a cada usuario, ofrecer la utilización compartida y asignación de los recursos físicos comunes (conmutación), gestión de la información compartida. Sus **cualidades** serán: disponibilidad, fiabilidad y seguridad, facilidad de extensión y adaptación a los diferentes usuarios, facilidad de uso y eficacia.

También existen los **SISTEMAS MULTIPROCESADOR** que **s**e caracterizan por la existencia de varios procesadores centrales compartiendo a veces memoria y periféricos en la ejecución de instrucciones en paralelo (concurrencia real).

Sus **funciones** son todas las de los sistemas TS/Mutiprogramados, las de los sistemas de tiempo real y además las de gestión de los 'n' procesadores: asignación, conmutación, comunicación, etc... Por otro lado, si estos procesadores están a cierta distancia y se comunican por las líneas de transmisión de datos (sistemas distribuidos) tendrán como función la gestión teleinformática.

Sus **cualidades** serán todas las de los tipos anteriores, pero haciendo especial hincapié en la fiabilidad.

2º- Según la arquitectura del computador:

Existen los **SISTEMAS MONOLÍTICOS** que están diseñados como un conjunto de rutinas (rutinas de servicio y rutinas de utilidad o auxiliares) compiladas por separado, que pueden llamarse entre sí y que se montan para formar el S.O., que será la rutina principal (gestor de interrupciones).

El **principal problema** de los sistemas operativos monolíticos es su diseño poco apropiado para sistemas operativos grandes, ya que es muy difícil de especificar, codificar, probar y depurar. La solución sería introducir estructura en el S.O.

Como ejemplos de S.O. monolíticos están DOS/VS 360 y UNIX.

También se desarrollaron los **SISTEMAS POR NIVELES O CAPAS** en los cuales los sistemas operativos están diseñados como una jerarquía de niveles, en la que cada nivel hace uso de las facilidades que le proporcionan el hard y los niveles inferiores a él, sin necesidad de conocer los detalles de operación de dichos niveles.

Por tanto **el S.O**. está constituido por una serie de **módulos pequeños** y fáciles de manejar, cada uno de ellos especializado en una única función y con interfaces con otros módulos.(**ocultamiento de información**) Son ejemplos de S.O. por niveles MINIX y THE :

La principal ventaja de estos sistemas es su facilidad de mantenimiento que incrementa la fiabilidad del sistema, y su principal problema reside en la dificultad de elegir las funciones que deben asignarse a cada nivel.

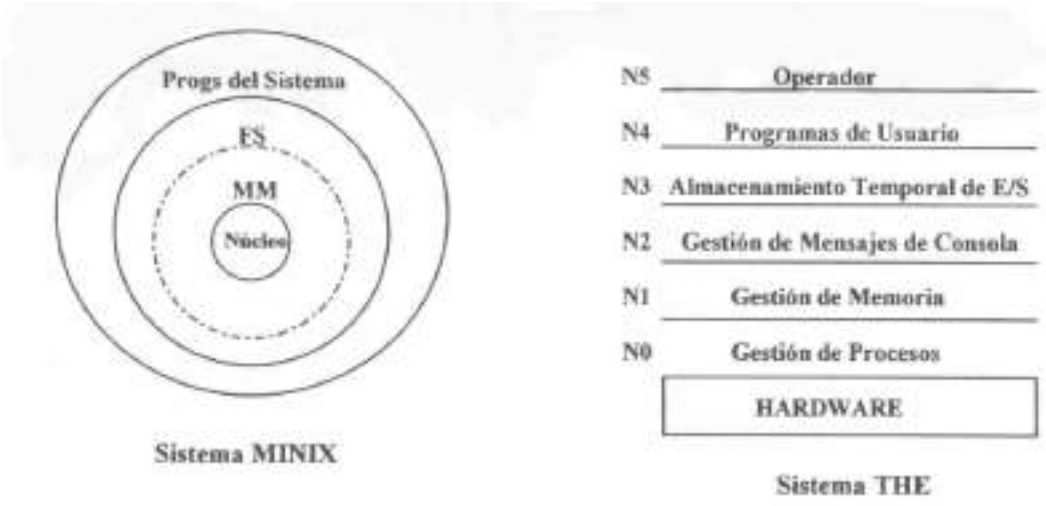


Figura 4.4. Representación gráfica de un sistema por capas

Como extensión del enfoque de capas surgieron los **SISTEMAS DE MÁQUINA VIRTUAL** en los cuales se separa la parte del S.O. que proporciona la máquina virtual (la máquina adaptada al usuario) de la parte que posibilita la compartición de recursos.

Se pueden instalar diferentes S.O.s sobre las máquinas virtuales, lo cual proporciona gran flexibilidad al sistema y la posibilidad de experimentar y reducir tiempos de desarrollo con nuevo S.O.s sin interferir con el trabajo habitual.

Ejemplo : VM de IBM

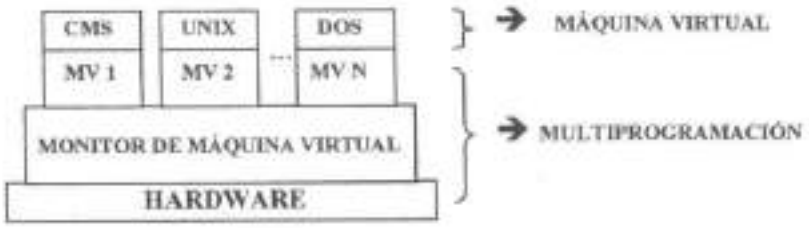


Figura 4.5. Representación gráfica de un sistema máquina Virtual VM de IBM

El **principal problema** de estos sistemas es la dificultad de implementación.(proporcionar duplicados exactos de las máquinas)

Por último también existen los **SISTEMAS BASADOS EN MICROKERNELS** que incluyen en el núcleo sólo las funciones esenciales del S.O., estando el resto de funciones implementadas en módulos a nivel de usuario sobre el núcleo, que hace de base común para los procesos de usuario y el resto de servicios del S.O.

Por tanto estos sistemas tienen un núcleo mínimo pero con un número suficiente de mecanismos eficientes, que le capacitan básicamente para gestionar mensajes y controlar los accesos al hardware.

Las **principales ventajas de este tipo de S.O.s** son su fácil adaptabilidad a los sistemas distribuidos; su alta fiabilidad (por tener un núcleo pequeño); su facilidad de idear; implementar y mantener; además de su facilidad con la que los servicios implementados pueden modificarse, recompilarse y cargarse sin afectar al resto.

La **principal dificultad** consiste en decidir qué funciones deben incluirse en el núcleo: las que necesiten rapidez o deban ejecutarse muy a menudo. En general habrá que lograr un equilibrio entre servicios rápidos e inflexibles, ya que cuanto menos se sobrecargue el núcleo más fácil será su mantenimiento.

4.1.2-Requisitos del sistema operativo multitarea.

Los sistemas operativos que soportan la multitarea manejan una lista de tareas pendientes de ser procesadas, que se ordena y organiza en función de diversos criterios, o bien, como sucede en los sistemas operativos de tiempo real, de acuerdo con la aparición de factores externos.

Para ello debe poseer **ciertos requisitos**. Un requisito imprescindible para la multitarea es la asignación de una zona exclusiva de memoria a cada tarea, en la que se almacenan su código y sus datos. Dicha zona de memoria recibe el nombre de **área local.**

Por otra parte, también deberá existir una zona de memoria común y compartida por todas las tareas, a la cual se denomina **área global.**

Cada tarea tendrá **derecho a acceder** a su propia área local y al área global; pero tendrá prohibido el acceso a las áreas locales de las restantes tareas.

En cada momento, el sistema operativo pone a disposición de la tarea en curso los recursos de la máquina, incluyendo los espacios de memoria adecuados.

Como se ha mencionado anteriormente, cada tarea tiene un **tiempo asociado para su ejecución ("quantum")**, pasado este tiempo, el sistema operativo provoca una conmutación de tareas poniendo la máquina a disposición de otra tarea.

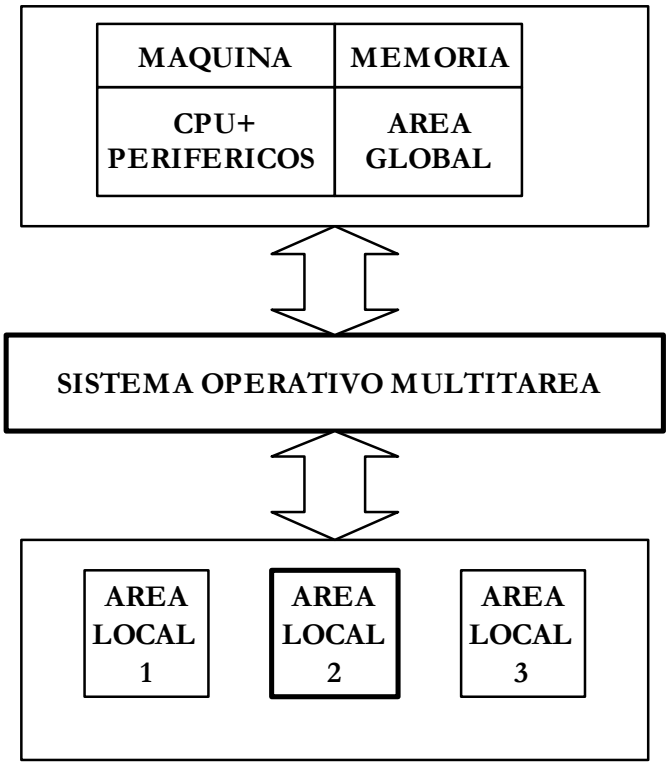


Figura 4.6. Representa un sistema operativo multitarea.

4.2-DÉSARROLLO HISTÓRICO DEL HARDWARE DE LOS SISTEMAS OPERATIVOS

4.2.1- FASE 0: AMBIENTE MANUAL E INTERACTIVO DIRECTO

Inicialmente los ordenadores no tenían S.O. por lo que cada vez que se quería que el hardware hiciese algo, el "usuario" debía reservar previamente su tiempo de uso de ordenador (**método de reservas de solicitud**), luego tenía que codificar todas las instrucciones del programa (en forma de 0s y 1s), cargarlo mediante interruptores y seguir la ejecución mediante lucecitas en la consola **(ejecución paso a paso**).

Evidentemente esto suponía muchísimo trabajo y una altísima probabilidad de cometer errores, por lo que se hacía necesario reducir las diferencias entre el lenguaje máquina y el lenguaje humano. La solución vino con el desarrollo de un software básico (componentes) y de nuevos dispositivos.

4.2.2- FASE 1: DESARROLLO DE COMPONENTES

Esta fase se caracteriza por el desarrollo de componentes como ensambladores, compiladores y herramientas que faciliten la puesta a punto de los programas (depuradores, etc...). Así los programas eran más inteligibles y el número de errores disminuyó considerablemente.

Luego, para evitar tener que codificar u montón de instrucciones similares cada vez que había que hacer una operación de E/S, codificaron unas rutinas de E/S (*device/driver*) con parámetros que se cargaban en memoria con los programas, los cuales sólo tenían que llamarlas pasándolas los parámetros adecuados. Estas rutinas de E/S se encargaban de sincronizar la E/S de datos con la CPU, de la conmutación automática de Ioáreas y detectaban algunos tipos de errores; Por tanto gestionaban buffers, flags, registros, bits de control, bits de estado, etc...

Otro componente que se desarrolló en esta fase fueron los programas de ayuda, como generadores de listados, cargadores de programas, programas de volcado de memoria (DUMP), combinadores de programas,(MERGE)... . Además surgió la idea de utilizar una biblioteca central de programas.

Con todo esto se consiguió aumentar un poco el rendimiento y facilitar la programación, pero la operación era muy compleja (**modo operativo de puerta abierta**) y se seguía haciendo un uso antieconómico de un material muy costoso, por lo que la siguiente fase tendría como principal objetivo aumentar el rendimiento reduciendo el tiempo de preparación.("Set-up time")

4.2.3.- FASE 2: OPERADOR / MONITORES RESIDENTES

Lo primero que se hizo para **aumentar el rendimiento** fue separar los tipos de trabajos naciendo así la figura del **operador**, que era un profesional especializado en operar con el sistema, por lo que era más rápido y cometía menos errores, desapareciendo así la puerta abierta y la inactividad por método de reserva de ordenador. Además, el operador solía clasificar los trabajos para aumentar la productividad (proceso por lotes (*BATCH*)).

No obstante se observó que la diferencia entre el tiempo de ejecución y el de preparación era abismal, así que se desarrolló **un programa que hiciera el trabajo del operador** (el secuenciamiento automático de trabajos) llamado **programa de control**, para el cual se desarrollo un lenguaje de control de trabajos (*OSCL*) con su correspondiente intérprete.

En torno al programa de control se agruparon los componentes desarrollados en la fase anterior, dando lugar a los *primeros sistemas operativos* conocidos como **monitores de encadenamiento o residentes**, responsables de gestionar y proteger los recursos máquina, y cuyas funciones eran supervisar las operaciones de E/S, proteger la memoria reservada al S.O., limitar el tiempo de ocupación de la CPU por los programas (par lo cual tenían que gestionar un reloj), asegurar el flujo continuo de trabajo, etc...

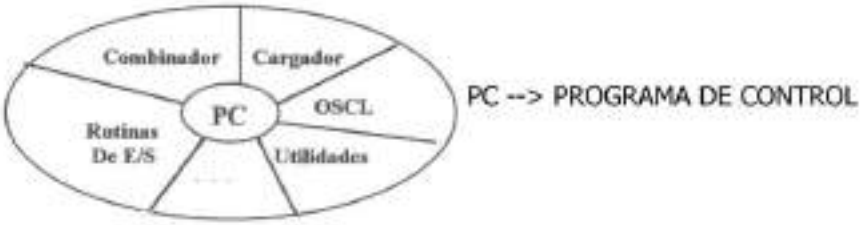


Figura 4.7. Representación gráfica de un monitor de encadenamiento

4.2.4.-FASE 3: DESARROLLO DE LA TECNOLOGÍA HARDWARE

El objetivo inicial en esta fase es disminuir el tiempo que la CPU emplea en las operaciones de E/S, ya que los periféricos son mucho más lentos al trabajar a velocidades electromecánicas.

- **FASE 3a**

Se desarrollan procesadores satélites u OFF-LINE, que eran unos dispositivos específicos (aparte del procesador principal) que permitían la **operación en OFF-LINE** al reemplazar las lectoras de tarjetas y las impresoras de líneas por unidades de cinta magnética.

Para conseguir una cierta **independencia de los dispositivos**, los programas trabajaban con dispositivos lógicos, siendo el S.O. el encargado de determinar los dispositivos físicos mediante el OSCL. Surge así la posibilidad de multiprocesamiento y la posibilidad de usar múltiples sistemas de grabación en cinta y de descarga a impresora.

Además se desarrollaron unos procesadores autónomos especializados en la transferencia de información entre memoria y periféricos, llamados **canales**, que se ocupaban de las operaciones de E/S. Así la CPU queda libre durante las operaciones de E/S, por lo que pensaron en cómo ocuparla, y así surgieron las E/Ss con **memorias intermedias** (buffers) de los que la CPU cogería los datos (leídos masivamente de antemano) para depositarlos en memoria intermedia también tras su proceso. De esta manera se conseguía solapar la E/S de los datos con su procesamiento.

Pero el *buffering* presentaba una serie de problemas como la dificultad de programarlo, la posibilidad de colapsar la memoria, no servía para trabajos intensivos de E/S y seguía sin poderse mantener ocupados a la CPU y a los dispositivos permanentemente, en parte por las limitaciones de la época (cintas) que eran secuenciales.

Aprovechando el nacimiento de los discos (**DAAD**) se encontró la solución a estos problemas trabajando con una memoria a dos niveles: técnica de SPOOL, que consistía en utilizar memoria secundaria como un gran buffer, evitando así que se colapsara la memoria central y consiguiendo un solapamiento de actividades.

Dado que quien tenía que gestionar esta memoria a dos niveles era el S.O., los monitores de encadenamiento tuvieron que extenderse dando lugar a los **Ejecutores** o **Ejecutivos**, los cuales, además de gestionar varios tipos de memoria, facilitaban el uso eficiente de los canales e incluían mecanismos completos de sincronización y manejo de interrupciones.

- **FASE 3b**

El objetivo de esta fase sería reducir los tiempos muertos de la CPU para aprovecharla al máximo y aumentar el rendimiento, y la solución vino con la **multiprogramación**: se reparte la memoria entre varios procesos, de forma que cuando el proceso en ejecución no podía seguir se conmutaba a otro proceso y así sucesivamente. Esta labor de conmutar el procesador central entre los diversos procesos la llevaba a cabo el **planificador de bajo nivel** o "dispatcher".

De esta manera se incrementaba considerablemente el rendimiento específico e incluso la disponibilidad, pero hubo que aumentar también la memoria y ampliar el S.O. para que incluyera al dispatcher y se ocupara de la gestión, protección y compartición de la memoria y de la asignación y ubicación de recursos que hicieran posible la concurrencia.

Paralelamente a la multiprogramación, surgieron las compañías de líneas aéreas con unas necesidades de tratamiento informático nuevas: básicamente necesitaban tratar un fichero maestro de plazas de forma ON-LINE desde varios puntos y con un tiempo de respuesta aceptable. La solución a este problema vino dada por una combinación de hard y soft: En cuanto al hardware se desarrollaron CPUs más rápidas, nuevos periféricos (terminales) y sistemas de comunicación; y en cuanto al software se desarrolló un programa maestro capaz de atender a varias peticiones de información simultáneas (procesado concurrente).

Y así nacieron los **Sistemas Transaccionales**:

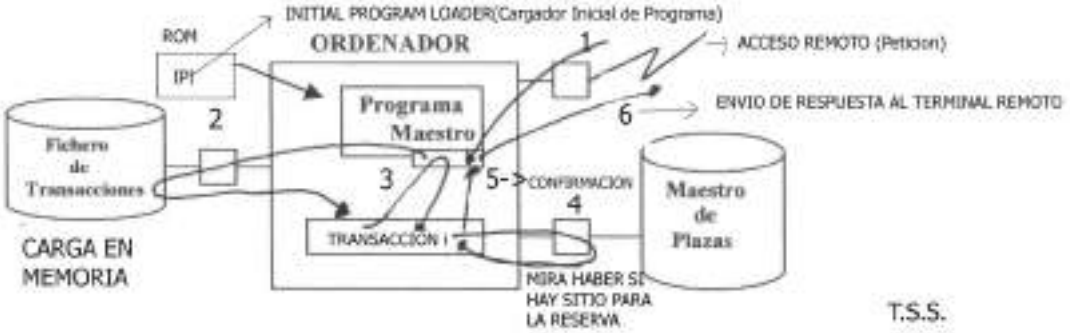


Figura 4.8. Representación de un sistema transaccional y su funcionamiento

4.2.5.-FASE 4: CRECIMIENTO DE LAS PRESTACIONES DE LOS MICROPROCESADORES

En esta fase se desarrollaron técnicas de transmisión de datos (*teleinformática*) y la progresiva integración de la función de comunicación en los sistemas (*telemática*).

De esta manera surgen las necesidades de compartir recursos y de adecuar la estructura de los sistemas a las aplicaciones que tratan (*descentralización*). La solución vino con el desarrollo de las **redes de teleinformática** (S.Distribuidos), de las **redes locales** y de los **sistemas multiprocesador** que permiten el procesamiento en paralelo (concurrencia real y aparente).

Para soportar las redes se desarrollaron los Servidores como módulos incluidos en el S.O. encargados de controlar las comunicaciones, obteniéndose S.O.s multiformes de propósito general.

DIAGRAMA DE LA EVOLUCIÓN HISTÓRICA

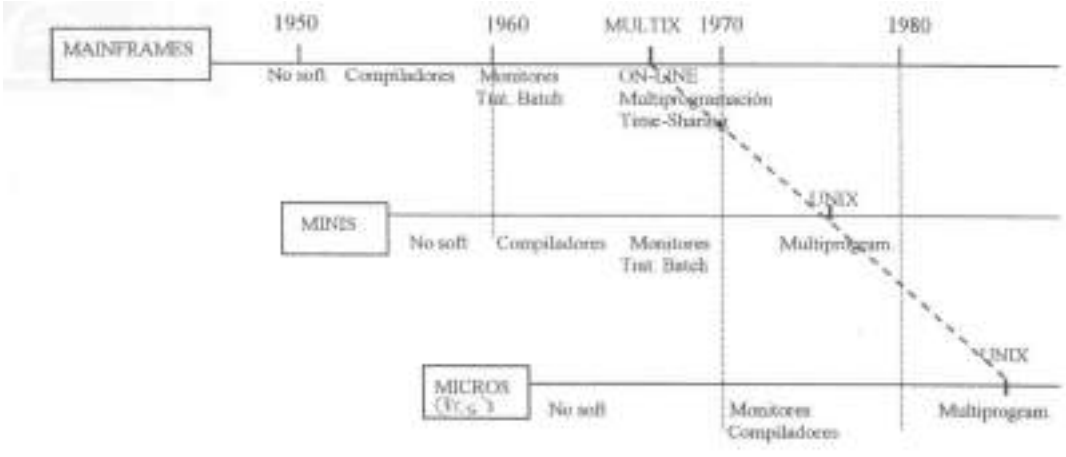


Figura 4.9. Esquema de la evolución histórica de los S.O.

Ahora las tendencias actuales son las siguientes:

• **USER-FRIENDLY**: Se tiende a que la máquina sea lo más "amistosa" posible, por lo que el interface presentado por el S.O. proporciona un acceso fácil y un manejo sencillo y guiado, vía menús, mensajes de ayuda e iconos, unido a una homogeneidad en el aspecto de las diferentes aplicaciones (*look & feel*).

• **MÁQUINA VIRTUAL**: El usuario no necesita conocer la máquina física sino la que le presenta el S.O.

• **PROCESAMIENTO DISTRIBUIDO**: Varios computadores independientes interconectados entre sí, cooperando e intercambiando información y con su propio procesamiento local (tolerancia a fallos, redundancia,...), S.O.s dispersos, disminución del coste de comunicación y mayor velocidad de transmisión.

• **PROCESAMIENTO PARALELO**: Aumento de la escala de integración y de la capacidad de procesamiento y almacenamiento, máquinas masivamente paralelas, lenguajes concurrentes y arquitecturas que distribuyen el control entre varias CPUs.

4.3 CARACTERÍSTICAS ESPECÍFICAS DE LOS MICROPROCESADORES AVANZADOS

Los microprocesadores avanzados provocaron un aumento del rendimiento en el procesamiento que se debe en gran parte a su configuración de su estructura interna también influyó la base de registros y a buses de 32 bits, además a que la mayoría de la cascada de instrucciones (pipeline) y por supuesto al aumento de las frecuencias de trabajo y el uso de memorias ultrarrápidas.

Estas **mejora**s se relegan a un segundo plano debido a los avances que se producen en la arquitectura de microprocesadores avanzados y la utilización de multitarea; aunque exigiendo la multitarea **aportaciones** de ciertos requisitos como que la CPU sea muy veloz sumándole una gran capacidad de memoria para soportar la multitarea y por último pero muy importante un **sistema de protección** que restringa los accesos privilegiados a las tareas y evite errores de protección de tareas tanto en área global como local.

Para obtener memoria de gran capacidad se emplea la técnica de memoria virtual. Que explicaremos a continuación pero que hace referencia a que el programador o el usuario piense que posee más memoria de la que realmente posee y puede utilizar.

El sistema de protección para soporte multitarea exige la inclusión en la arquitectura del procesador de dispositivos capaces de proporcionar al sistema operativo multitarea los requerimientos que precisa de la máquina.

Por lo que esto se refiere a que es más importante que el aumento de potencias, velocidad,..el trabajar en entornos multitarea, multiusuario y en tiempo real, basándose estas prestaciones en cuatro aspectos:

1. **Memoria Virtual:** Método de organización y gestión de la memoria que proporciona al programador de aplicaciones un espacio mucho mayor de que dispone físicamente, da la ilusión a la CPU de que puede manejar mucha más memoria de la que la DRAM tiene a su disposición.

2. **Sistemas Multitarea:** Dispone de los recursos físicos(registro de tarea, TSS) para producir un cambio o conmutación de tareas rápido y seguro.

3. **Sistemas de protección:** Es un sistema que controla el acceso de las tareas a memoria. Que los segmentos de datos y de código sólo puedan acceder a segmentos accesibles por ellos, SÓLO de la misma tarea y del área global.

4. **Subsistemas de memoria caché:** Es una memoria ultrarrápida para las CPUs que trabajan a frecuencias medidas en GHz, normalmente oculta.

4.4 MEMORIA VIRTUAL

Definimos memoria virtual como conjunto de programas que tienen el sistema operativo que hacen creer a las CPU y por consiguiente, a los usuarios/programadores que pueden manejar directamente los discos aunque en la realidad sólo pueda acceder a la memoria electrónica (DRAM).

Nace la memoria virtual porque la CPU sólo es capaz de acceder directamente a una memoria principal o física cuya capacidad está limitada por el tamaño del bus de direcciones, y por ello cuando el procesador intenta acceder a memoria es realmente poco para el procesador.

Hay que tener en cuenta que en la memoria principal residen los datos y las instrucciones que manipulan la CPU directamente, a través de los buses, entre los dos elementos básicos del ordenador y que dicha memoria debe estar constituida con una tecnología similar a la de la CPU para que los tiempos de trabajo(**búsqueda de instrucciones y ejecución de CPU**) sean similares, (circuitos integrados CMOS rápidos)siendo el rendimiento de la máquina óptimo.

Las memorias RAM y ROM siguen siendo caras si las comparamos con dispositivos de almacenamiento magnéticos.

La CPU está limitada por el tamaño del bus de direcciones aunque éste haya evolucionado como aquí se indica:

	8085	Pentium
Número de bits	8	32
Líneas de bus direcciones	16	32(mínimo)
Tamaño memoria ppal.	64KB	4GB

Para **utilizar la memoria virtual** el procesador deberá realizar una serie de comprobaciones y pasos, que son los siguientes:

1. Genera la dirección del objeto que necesita, un mecanismo de gestión de memoria, denominado Unidad de Gestión de Memoria (MMU) comprueba si el objeto se encuentra en memoria principal

2. Si está en memoria principal accede a él normalmente.

3. En caso contrario, comunica el hecho al sistema operativo que pone en marcha la rutina encargada de localizar el objeto en memoria virtual (disco) y transferirlo a la memoria principal para que la CPU acceda normalmente a él.

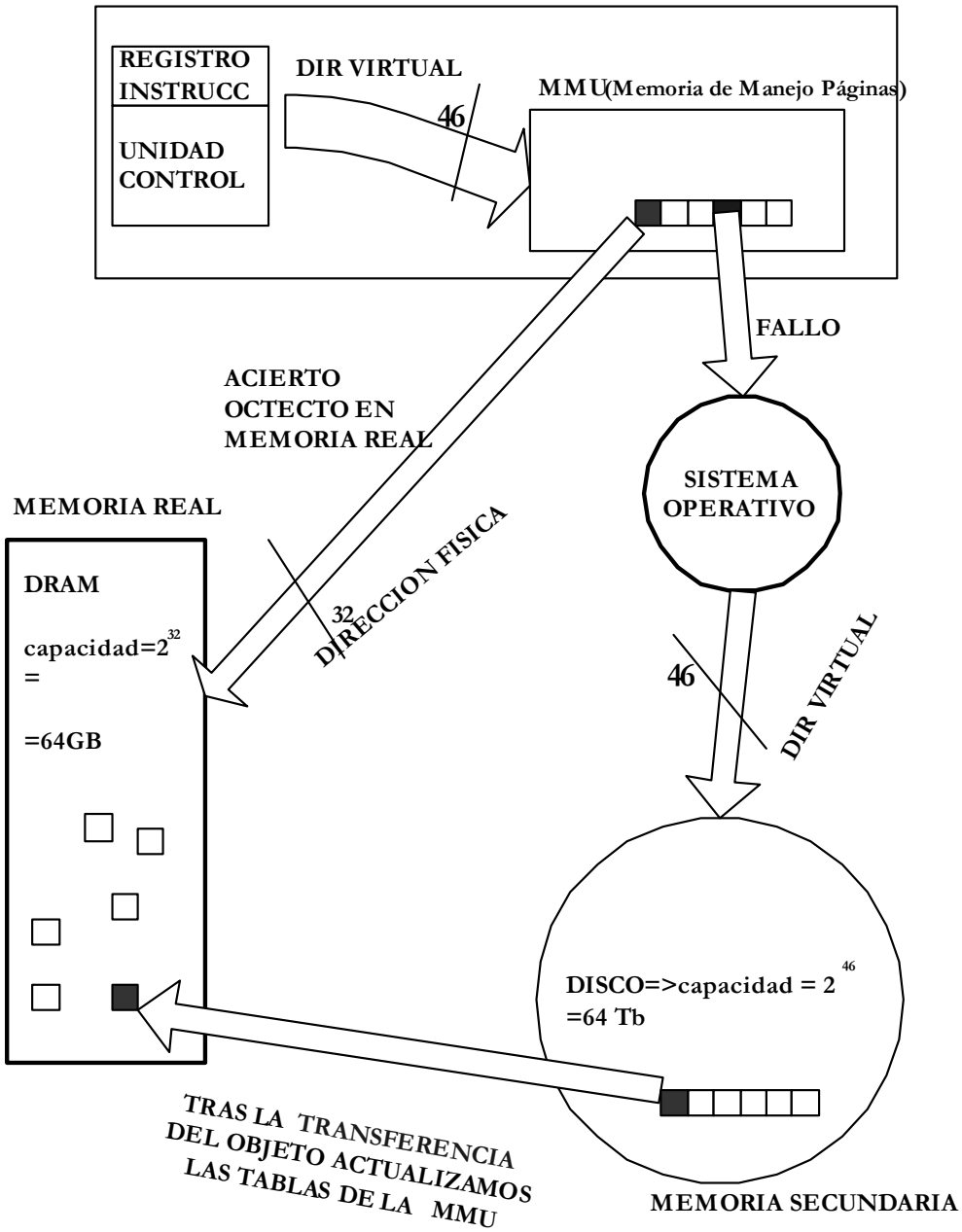


Figura 4.10. Representación de una traducción en memoria virtual

Como la memoria virtual es mucho mayor que la principal, existirá un constante flujo de transferencia entre ambas.

La memoria virtual permite que las aplicaciones ocupen mucho más espacio que el disponible en la memoria principal. Además si los **algoritmos de transferencias** son buenos esto implicará un aumento de accesos directos a la CPU implica **aumento de velocidad**; debido a que las posiciones que precisa la CPU se anticiparán.

La implantación de la memoria virtual conlleva la incorporación de hardware que configura la MMU que **facilita la comunicación con el sistema operativo** que debe disponer de las siguientes **tres funciones**.

La **primera** de ellas la posesión de una lista completa con la descripción de todos los objetos residentes tanto en memoria virtual como en la principal. La **segunda** de ellas es la carga de objetos en al memoria física en tiempo de ejecución.

Y por último, la **tercera**, el manejo de los espacios libres de la memoria física si no hay espacios libres decidir con un algoritmo cuál sustituimos en la memoria principal, para insertar la posición necesitada. Además al sacar la posición de memoria principal habrá que comprobar si ha sido modificada para antes de sustituirla modificarla en la memoria virtual o si no sustituirla directamente.

Existen dos formas de organizar la memoria virtual.(**Segmentación** y **Paginación**)

Una de ellas, la **segmentación** que es la división de la memoria virtual en "trozos"(segmentos) de tamaños variables. Los distintos segmentos pueden crecer o reducirse en forma independiente sin afectar a los demás, esto hace más sencilla la administración de las estructuras de datos que crecen ó se reducen, si cada procedimiento ocupa un segmento independiente con la posición inicial cero el ligado independiente de los procesos compilados es mucho más sencillo.

Otra forma de división es la **paginación** que es la división del espacio de direcciones de cada proceso en bloques de tamaño uniforme llamados páginas(1K,2K,4K ó4MB), los cuales se pueden colocar dentro de cualquier página marco disponible en memoria.(Se desaprovecha memoria), cuando las tablas de páginas son muy grandes se puede utilizar un esquema de paginación de varios niveles que las páginas se paginen a sí mismas.

En Intel la memoria virtual es de 64 Tb.

4.5 TIPOS DE MEMORIA VIRTUAL

Los distintos modelos de memoria virtual se diferencian por sus políticas de solape y por métodos que emplean en a organización de la memoria:

Como son la Memoria paginada, la Memoria segmentada, y la Memoria virtual con segmentos paginados. En estos tres casos la fracción de memoria que debe asignarse a un programa es variable en cada caso.

La **política de solape y compartición** debe tener en cuenta ciertas características internas de los programas que, determinan la construcción modular y estructurada de los mismos.

El sistema operativo debe tener en cuenta la fracción de memoria principal que se va a sustituir o cargar en disco así como las modificaciones que existan de lectura y escritura. Los criterios usados son los siguientes

Regla FIFO(First In – First Out): Se sustituye la fracción que más tiempo lleva en la memoria principal para dejar hueco a otra.

Regla LRU(Last Recently Used): La porción que lleva en la memoria más tiempo sin haber sido usada.

Regla LIFO(Last In – First Out): Se sustituye la fracción que menos tiempo lleva en la memoria principal para dejar hueco a otra.

Regla LFU(Last Frecuently Used): La porción que se accedido menos veces desde que se inició el proceso.

Regla RAND(Random): Se elige una porción al azar.

Regla CLOCK: Cuando se coloca un bit de uso en cada entrada de una cola FIFO y se establece un puntero que se convierte en circular. Es una aproximación al algoritmo LRU con una simple cola FIFO.

4.5.1-MEMORIA PAGINADA

Este modelo organiza el espacio(memoria) virtual y el físico en bloques de tamaño fijo llamados páginas.(de 1, 2 4 KB ó 4MB en los últimos Pentium). Los **S.O**. son muy **sencillos** porque los algoritmos de transferencia son muy sencillos, sabemos el tamaño de la sustitución, es decir el tamaño de la página. Pero esto implica que en una página no siempre entra toda la estructura del objeto y a veces el objeto es muy pequeño y la página no se llena, **desaprovechamiento de la memoria** es **clara** si hay **pocos datos** en la memoria. Está claro **no** se **optimiza la memoria.**

Ya que en la realidad como se trabaja con la memoria principal en lugar de con el disco, y se deben usar direcciones físicas se precisan de dos métodos para realizar la traducción virtual a física que son los siguientes:

4.5.1.1- MÉTODO DE CORRESPONDENCIA DIRECTA

El primero de los métodos para traducir la dirección física es el que se denomina método de correspondencia directa.

La dirección virtual para una página se divide en dos campos:

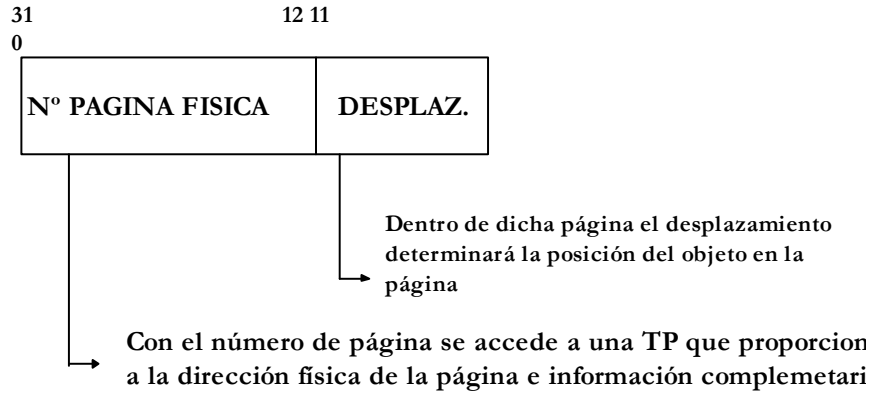


Figura 4.11. Representación de una dirección virtual de página.

La tabla de páginas (TP) tiene tantas posiciones como páginas de memoria virtual hay. La configuración de las **entradas de TP** se muestran en la figura y consta de los siguientes campos:

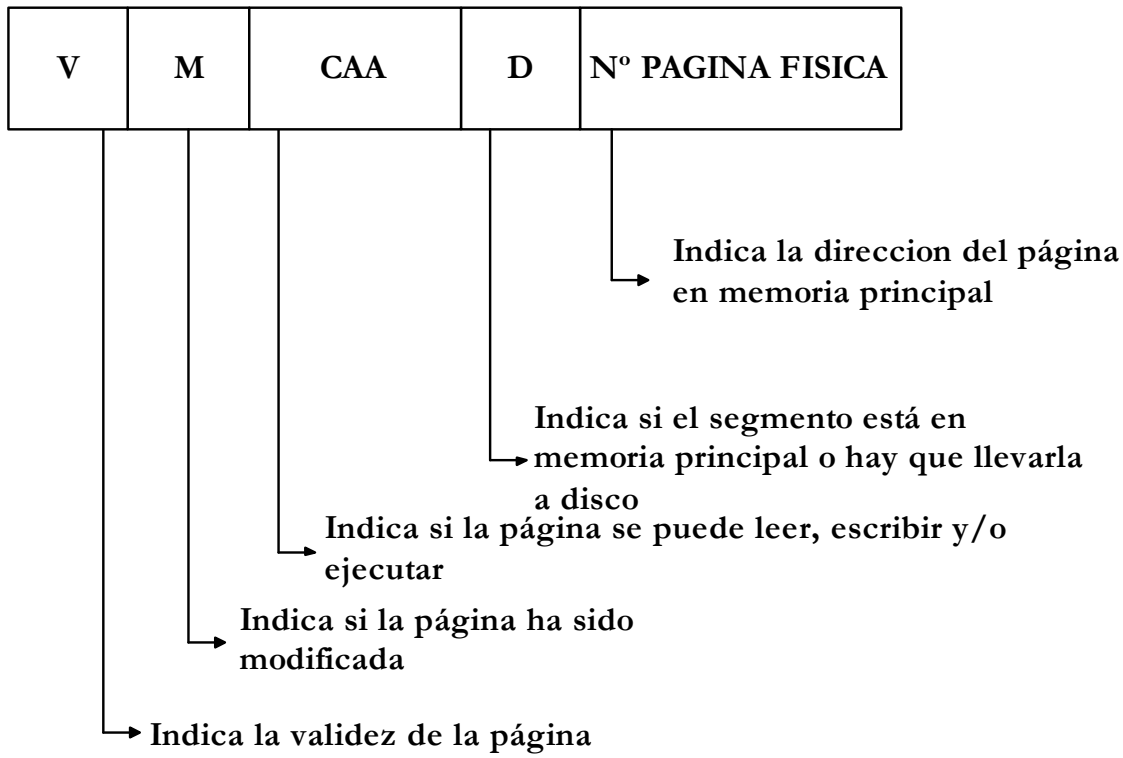


Figura 4.12. Representación de una entrada de la Tabla de Páginas.

La dirección de comienzo de la TP está almacenada en el Registro Base de la Tabla de Páginas(RBTP). Para acceder, a las entradas de la TP, se incrementa el valor correspondiente al número de página virtual (npv). Para calcular la dirección física en memoria se concatena DP con el desplazamiento (d) cuando la página en la memoria principal (Figura 4.13)

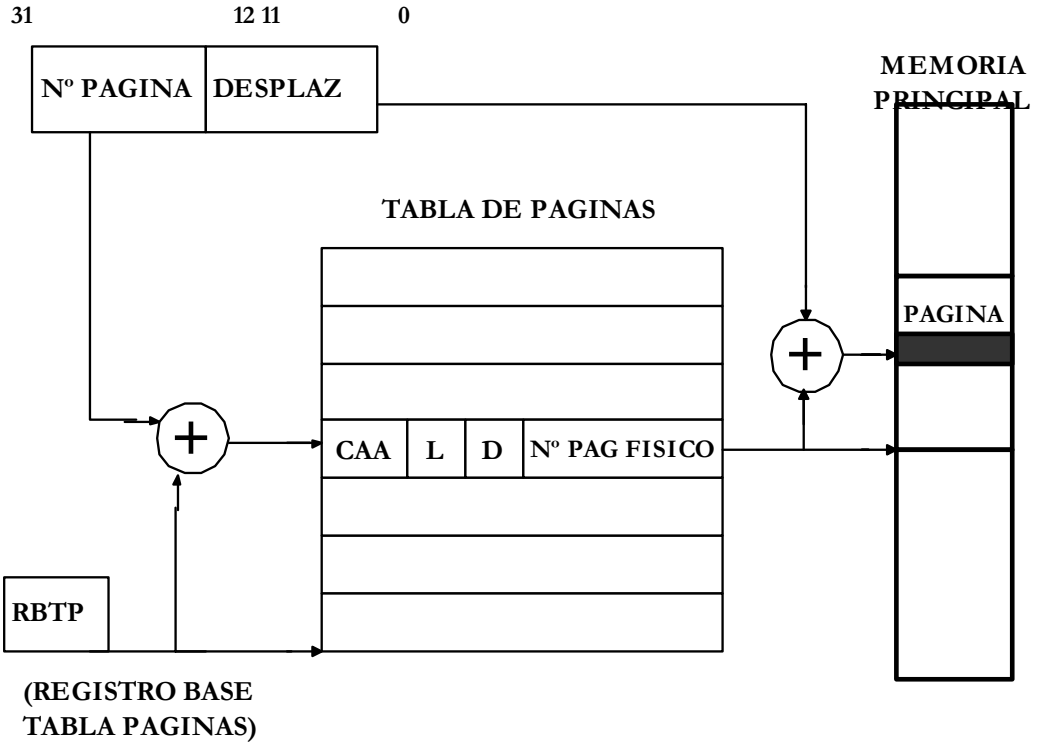


Figura 4.13. Representación del funcionamiento de búsqueda de una página con una Tabla de Páginas

Esta forma de trabaja plantea un **inconveniente** que es que el número de entradas de la TP debe coincidir con el de páginas virtuales, que es muy grande. Para **solucionarlo** se hace uso de la **tabla inversa**, ordenada por el número de página física, que reduce el número de entradas necesarias, pero requiere una traducción con una función de búsqueda. El proceso es muy lento.

4.5.1.2- MÉTODO DE CORRESPONDENCIA ASOCIATIVA

En este caso se dispone de una tabla inversa realizada con tecnología asociativa, esto es, memoria tipo CAM, mucho más rápida porque en lugar de tener la TP en memoria principal se carga en esa memoria especial llamada CAM, que se encarga del proceso de búsqueda a muy alta velocidad, suministrando el número de página física o indicación de que la dirección lógica no se encuentra en memoria. En este último se elimina una página de la memoria principal y se trae la nueva al hueco que deja.

En una memoria asociativa se puede acceder a la vez a todas las posiciones para identificar la que satisface cierto criterio de selección. Dado el elevado coste de las memorias asociativas, la tabla en CAM suele ser incompleta, albergando el conjunto de páginas activas en un momento determinado. Si se origina una falta en la CAM, hay que acudir a la TP para comprobar si está en la memoria principal y, en tal caso, actualizar la CAM. De lo contrario se procederá a un cambio de página.

La memoria CAM se divide en dos partes: Etiqueta y Datos.

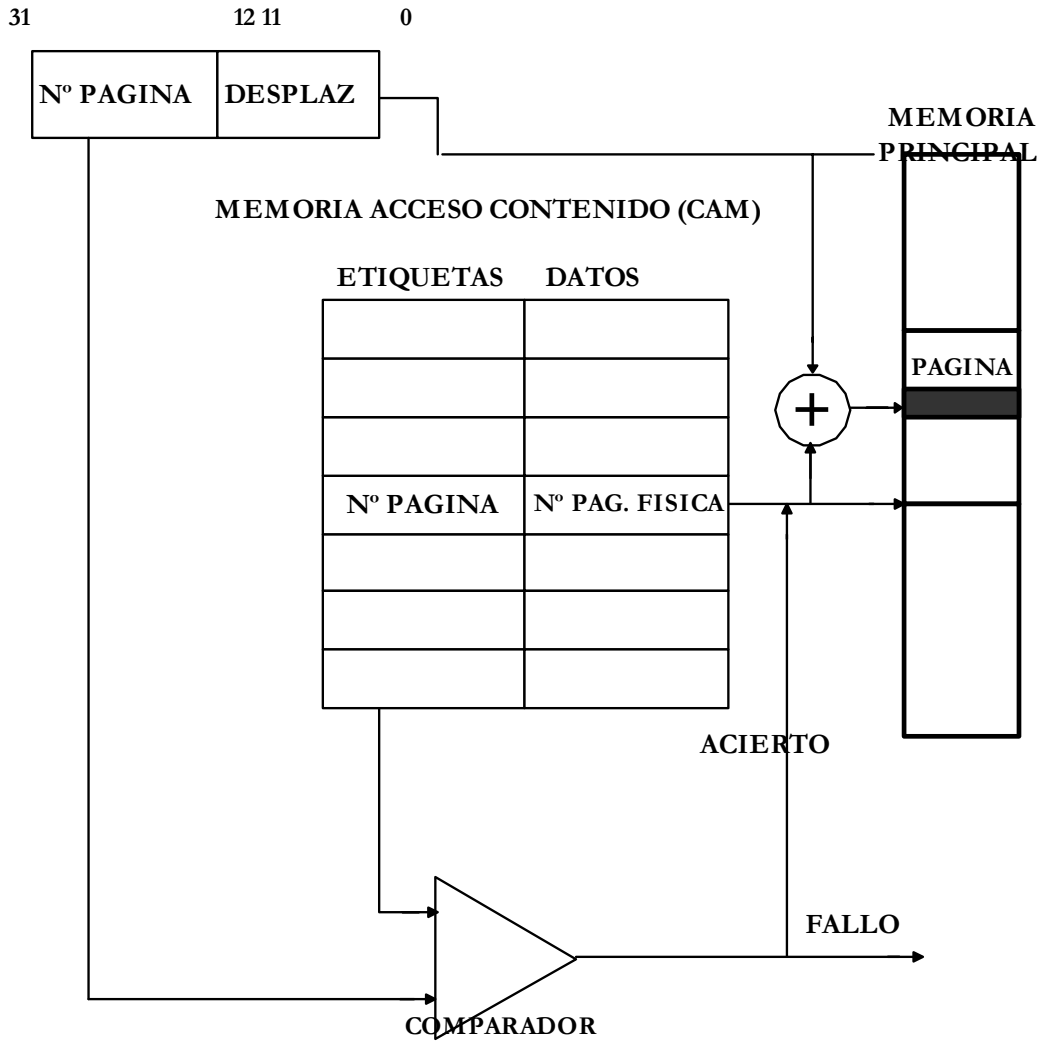


Figura 4.14. Búsqueda de una página mediante una memoria de acceso por contenido.

4.5.2- MEMORIA SEGMENTADA

La memoria se divide en trozos de tamaño variable llamados segmentos. Éstos tienen un tamaño diferente y se adaptan al tamaño del elemento siempre del mismo tipo para cada segmento que deben tener cada segmento; lo cual implica que la memoria virtual está muy bien estructurada.

Ello implica algoritmos de transferencia muy complicados por que no sabe le tamaño del segmento. Por ello necesita saber el tamaño (LIMITE) del segmento para saber si lo podemos introducir o no en la memoria real

Cada TS contiene los siguientes campos:

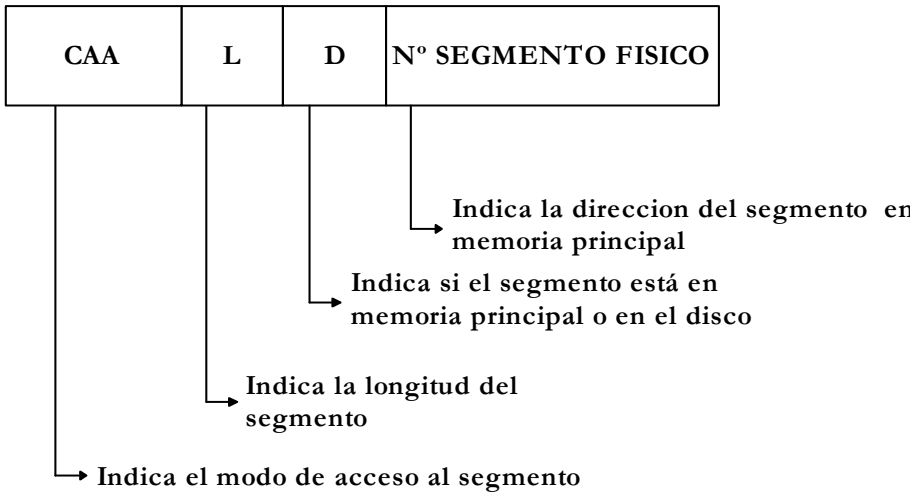


Figura 4.15. Disposición de los campos de una entrada de la TS.

Los elementos de un segmento se identifican con la dirección del segmento al que pertenecen y un desplazamiento dentro del mismo.

A semejanza con el modelo anterior existe una RBTS que direcciona el comienzo el comienzo de la Tabla de Segmentos (TS), de las que existe una por cada proceso activo.

Las TS funcionan de forma similar a las tablas de páginas (TP) antes descritas. Es decir con el número de segmento sumado al RTBS nos da la posición dentro de la tabla de segmentos.

Mientras no haya errores, en esa posición tendremos toda la información de ese segmento gracias a los campos de la tabla de segmentos.

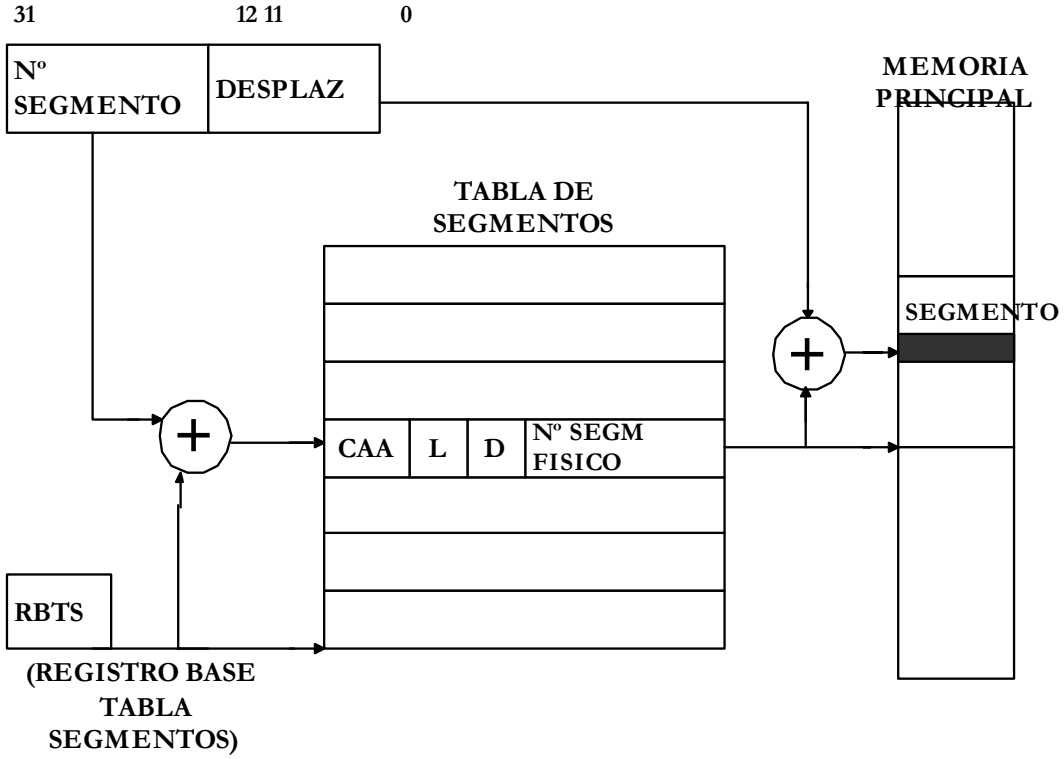


Figura 4.16. . Representación del funcionamiento de búsqueda de una página con una Tabla de Segmentos.

Siendo uno de ellos la dirección absoluta de comienzo de segmento que sumada al desplazamiento nos dará el segmento total y conseguiremos la información.

La utilización de la memoria segmentada implica mejor aprovechamiento de memoria ya que puede ser tan grande o tan pequeña como se quiera, lo que hace que no se desperdicie memoria y se estructure mejor y además proporciona un ajuste a la programación estructurada, utilizada por Intel.

Aunque esto también implica una complicación del sistema operativo, es más lento y con problemas.

Dado que la longitud de los segmentos es variable se precisa algún algoritmo que localice un espacio libre para que resida el segmento apropiado, ya que no es corriente encontrar un bloque continuo y vacío en memoria, para colocarlo por completo.

Estos algoritmos forman parte de un mecanismo que se pone en marcha cuando se detecta una falta de segmento, siendo los más relevantes: **De mejor ajuste:** Minimiza el espacio desaprovechado seleccionando el mejor hueco. **De peor ajuste:** Localiza el hueco maximiza el desperdicio. **De primer ajuste:** Localiza el primer hueco donde cabe el segmento empezando por la dirección más baja **Algoritmo Buddy:** Localiza técnicas de compactación de memoria, fusionando espacios útiles.

4.5.3- MEMORIA CON SEGMENTOS PAGINADOS

Este tipo de memoria implica que primero la memoria virtual se divide en segmentos que a su vez luego se paginan. Y quiere decir que para acceder a nuestro objeto primero accederemos a una TS y cada segmento de la TS tendrá su propia TP que será en ella donde encontraremos la página donde estará ubicado el objeto.

Por ello la dirección virtual se divide en tres como indica la siguiente figura:

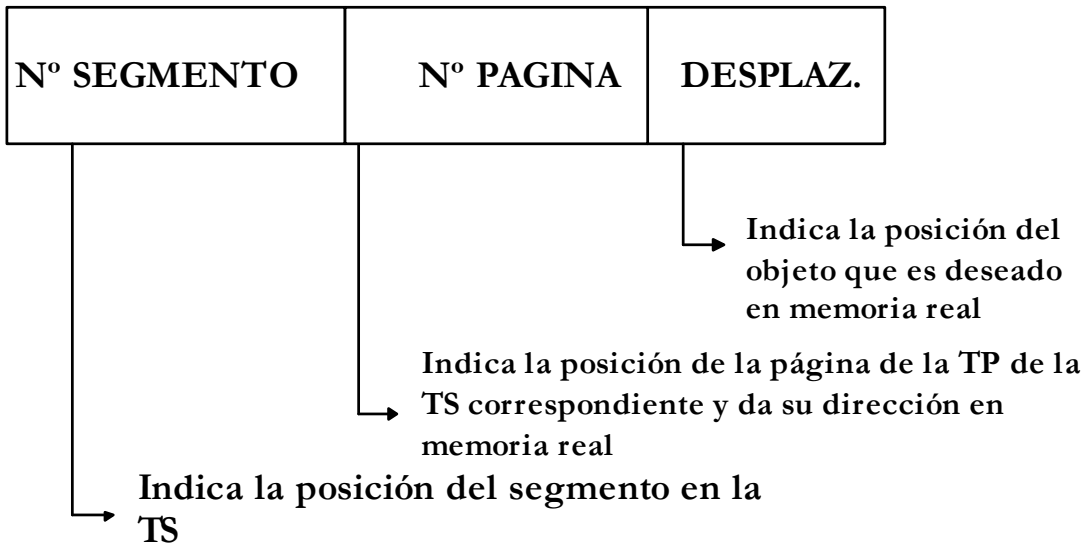


Figura 4.17. Representación de los campos de una dirección virtual

Ahora sólo se mueve páginas por lo que los algoritmos de transferencia son sencillos y además se aprovecha de la buena estructura de los segmentos.

Lo que en definitiva aprovecha las ventajas tanto de la segmentación y de la paginación en un intento de mejorar el rendimiento.

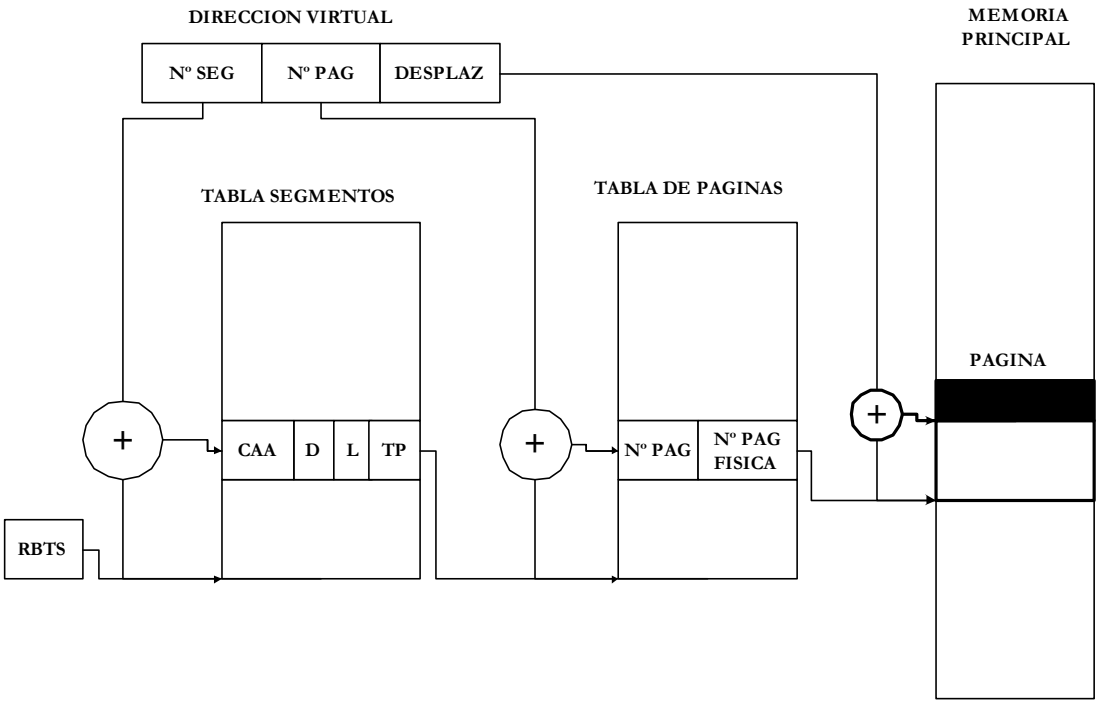


Figura 4.18. Representación del funcionamiento de búsqueda de una página con una Tabla de Segmentos que contiene una Tabla de Páginas.

4.6- MULTITAREA

La multitarea es una característica que poseen los sistemas operativos avnzados. Permite que varios procesos sean ejecutados al mismo tiempo compartiendo uno o más recursos y/o procesadores.

4.6.1- TIPOS DE MULTITAREA

- **Nula**: El sistema operativo carece de multitarea. Aún así puede lograrse a veces algo parecido a una multitarea implementándola en espacio de usuario, ó usando trucos como los TSR de MS-DOS. Un ejemplo sería Windows, hasta la versión 3.11

- **Cooperativa**: Los procesos de usuario son quienes ceden la CPU al sistema operativo a intervalos regulares. Resulta muy problemática, puesto que si el proceso se cuelga y no cede la CPU al sistema operativo, todo el sistema estará colgado. Da lugar también a latencias muy irregulares, y la imposibilidad de tener en cuenta este esquema en sistemas operativos de tiempo real. Un ejemplo sería Windows hasta la versión 3.11.

- **Peemtiva**: El sistema operativo es el encargado de administrar el procesador(es). Repartiendo el tiempo de uso de éste entre los procesos que estén esperando para utilizarlo. Cada proceso utiliza el procesador durante cortos periodos de tiempo, pero el resultado final es prácticamente igual que si estuviese ejecutándose al mismo tiempo. Ejemplo de sistemas de este tipo serían Unix y clones(FreeBSD, Linux,..), VMS y derivados, AmigaOS, etc,....

- **Real**: Sólo se da en sistemas multiprocesador. Es aquella en la que varios procesos se ejecutan realmente al mismo tiempo en distintos microprocesadores. Suele ser también preemptiva. Ejemplo de sistemas operativos con esa capacidad: Linux.

4.7-MECANISMOS DE PROTECCIÓN

En un sistema multitarea el sistema operativo divide a la memoria en "trozos"; tantos como tareas haya en el sistema más una: la global. Esto implicará que la memoria se divide en dos grupos de áreas:

1.Global: Área compartida por todas las tareas.
2.Local: Área propia y sólo propia de cada tarea.(indica **privacidad**)

Para evitar accesos indeseados en un espacio de memoria protegida deben estar especificados las áreas locales de cada tarea como el área global, con sus propios objetos.

Dentro de la zona de memoria existen distintos niveles de privilegio.(Indicando si el objeto debe estar muy **protegido** o no)

El esquema de protección de Intel dispone de cuatro niveles de privilegio, lo que implica una división de la memoria en cuatro porciones diferenciadas entre sí por. Siendo el valor de prioridad de cada nivel:

	Grado prioridad/seguridad	Programas de cada nivel
Nivel 0	El mayor/máximo	Funciones del núcleo
Nivel 1	Buena	Extensión del núcleo o E/S
Nivel 2	Aceptable	Programas comerciales
Nivel 3	Mínima	Programas del usuario

4.8- REGLAS DE ACCESO

Los microprocesadores del Pentium por hardware controlan un conjunto de reglas de acceso a objetos situados en diversas zonas de la memoria, y dentro de ellas, el acceso según los niveles de privilegio, que haya asignado el programador de sistemas.

Esto implica que Intel tiene mayor control de seguridad y mayor precisión y a su vez evita interferencias en el sistema multitarea, por lo que los accesos de un objeto a otro están más controlados.

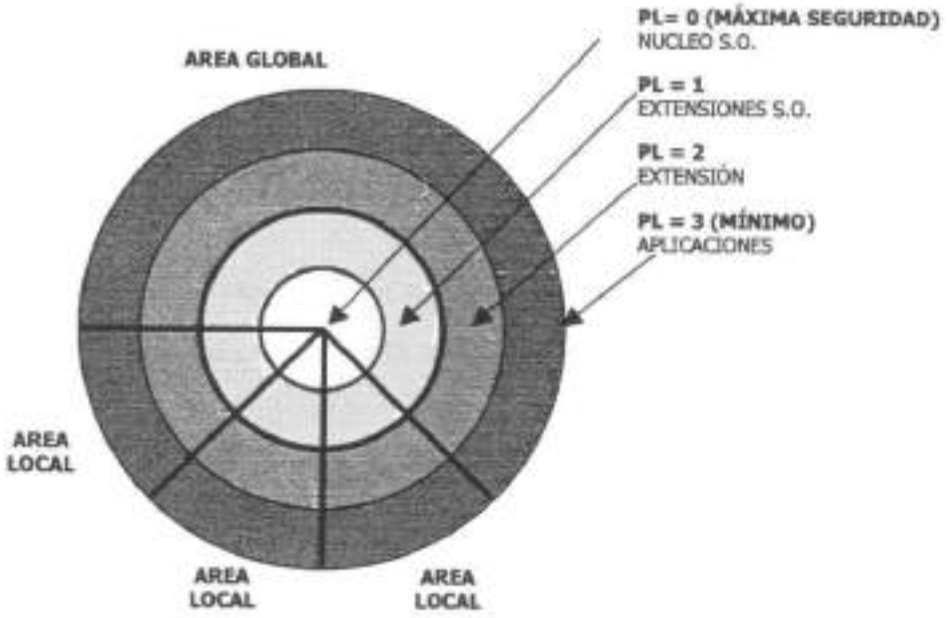


Figura 4.19. Espacio de memoria dividido en 4 niveles de privilegio. Utilizado por INTEL en los Pentium

1. PROTECCIÓN ENTRE TAREAS

La posibilidad que desde una Tarea se acceda a otra Tarea es NULA. Por el contrario está permitido el acceso de una Tarea al espacio global.

2. ACCESO DESDE OBJETOS DE CÓDIGO A OTROS OBJETOS DE LA MEMORIA

Desde un **objeto de código** se puede acceder de forma **JERÁRQUICA**:

Tipo de objeto	**Instrucción(es) utilizada(s)**	**Niveles de acceso permitidos**	**Permiso de acceso entre Tareas**
De código-código	JMP/CALL	Sólo igual	Sólo la misma tarea en curso y Área Global
De código-datos	MOV	Igual ó inferior	Sólo la misma tarea en curso y Área Global
De código-pila (**asociado al objeto de código**)	PUSH/POP	Sólo igual	Sólo la misma tarea en curso

3. EJECUCIÓN DE INSTRUCCIONES ESPECIALES

Dentro del repertorio de instrucciones de los procesadores con esquema de protección. Existen instrucciones **privilegiadas** que sólo pueden ser ejecutadas por el nivel máximo de privilegio (**jerárquicamente hablando***)*

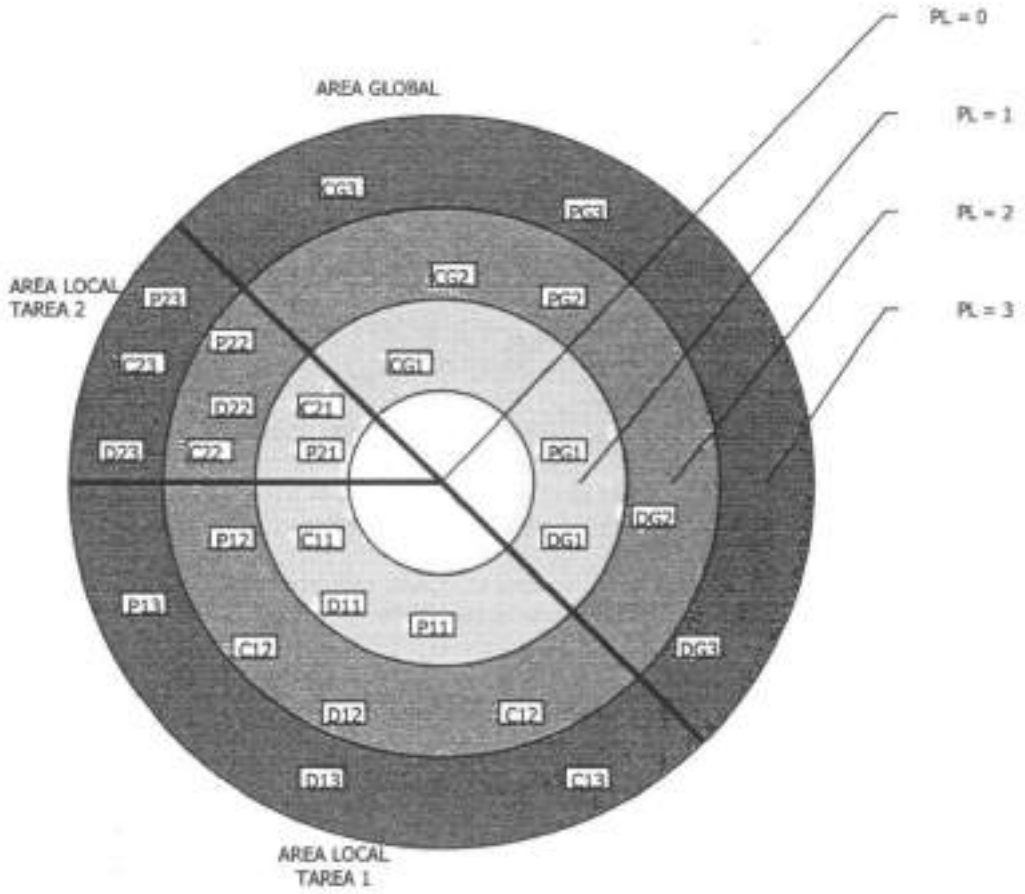


Figura 4.20. Reglas de acceso entre tareas.

Viendo la división de la memoria según la figura 4.12 aquí tenemos algunos ejemplos:

- **C11** podrá acceder a los objetos de datos D11, D12, D13 de su propia tarea y también podría acceder a cualquier objeto de datos del área global con PL =< al suyo; pero en este caso no hay objetos de datos en el área global.

- **C22** podrá acceder a los objetos de código C23 en su propia tarea, y además en el área global podrá acceder a los objetos de código CG2 y CG3.

C23 podrá acceder al objeto de pila P23, que es el segmento de pila asociado intrínsicamente al objeto de código siendo único y exclusivo en cada momento para cada objeto de código.

4.9- SISTEMAS OPERATIVOS ACTUALES

4.9.1- WINDOWS XP

Descripción:

- Sistema **multitarea**.
- Sistema **monousuario**.
- Sistema **multiprocesador**.
- *Windows XP Home Edition* : actualización de Windows 2000 Professional/Me.
- *Windows XP Professional* : actualización de Windows 2000 Server/Me.
- Basado en tecnología NT.
- Presenta notables mejoras: interfaz de usuario más sencilla y agradable, saca mayor partido al hardware de nuestro equipo, etc.

Requisitos mínimos y recomendados:

- *Windows XP Home Edition*

  Es la más compatible y poderosa que ninguna otra estación de trabajo anterior ya que consigue que el ordenador sea más fácil de manejar.

  - **Pentium II a 233 MHz mínimo / 300 MHz ó más recomendado**.
  - **64 Mb de RAM mínimo / 128 Mb recomendado** (4 Gb como máximo).
  - Disco duro de 3 Gb con un mínimo de **1'5 Gb de espacio libre**.
  - Unidad de **CD-ROM** o **DVD-ROM** (32x-12x o más rápido recomendado)
  - **Teclado** y **Ratón** Microsoft o compatible.

- *Windows XP Professional*

  Es un sistema operativo de red para negocios de todos los tamaños.

  - **Pentium II a 350 MHz mínimo / 500 MHz o más recomendado**.
  - **64 Mb de RAM mínimo / 256 Mb recomendado** (4 Gb como máximo).
  - Disco duro de 3 Gb con un mínimo de **1'5 Gb de espacio libre**.
  - Unidad de **CD-ROM** o **DVD-ROM** (32x-12x o más rápido recomendado)
  - **Teclado** y **Ratón** Microsoft o compatible.

Novedades:

- Se reducen significativamente el número de reinicializaciones, con lo cual, las instalaciones pueden realizarse en menor tiempo.

- Incluye también el llamado **Instalador de Microsoft** (MSI); Es la tecnología de Windows 2000 que permite la auto reparación de las aplicaciones (problemas causados por instalaciones / desinstalaciones mal realizadas, etc).

Para que Windows XP funcione debe de ser activado. Durante los 30 primeros días de uso tendremos la opción de hacerlo, y tras finalizar el periodo será obligatorio, ya que de lo contrario el sistema no permitirá iniciar sesión. **Windows Product Activation** se encarga de asociar la clave de instalación del sistema con el hardware el que se ha instalado, de forma que si realizamos más de 4 cambios en el hardware del equipo, tendremos que reactivar el producto para que éste siga funcionando.

4.9.2- WINDOWS 2000

Descripción:

- Sistema **multitarea**.
- Sistema **monousuario**.
- Sistema **multiprocesador**.

- *Windows 2000 Professional* : actualización de Windows NT WorkStation.
- *Windows 2000 Server* : actualización de Windows NT Server.
- Basado en tecnología NT.
- Presenta notables mejoras: al contrario que NT es **Plug & Play**, incorpora directorio activo que centraliza la información sobre recursos y usuarios, incluye mejoras en los servicios de impresión, etc.

Requisitos mínimos y recomendados:

- *Windows 2000 Professional*

Es la más compatible y poderosa que ninguna otra estación de trabajo anterior ya que consigue que el ordenador sea más fácil de manejar.

- **Pentium II**.
- **32 Mb de RAM mínimo / 64 Mb recomendado** (4 Gb como máximo).
- Disco duro de 2 Gb con un mínimo de **1 Gb de espacio libre**.
- Unidad de **CD-ROM**.
- **Ratón** Microsoft o compatible.

- *Windows 2000 Server*

Es un sistema operativo de red para negocios de todos los tamaños. La versión más actual de este sistema operativo permite:

1. Compartir ficheros e impresiones con seguridad.
2. Elegir entre cientos de aplicaciones de negocios compatibles ejecutables.
3. Construir aplicaciones de red y conexiones a Internet.

- **Pentium II**.
- **128 Mb de RAM mínimo / 256 Mb recomendado** (4 Gb como máximo).
- Disco duro de 2 Gb con un mínimo de **1 Gb de espacio libre**.
- Unidad de **CD-ROM**.
- **Ratón** Microsoft o compatible.

Novedades:

- Incluye una protección llamada Protección Windows de Archivos, que no permite que se borren ficheros del sistema.
- Incluye también el llamado Instalador de Microsoft (MSI); Es la tecnología de Windows 2000 que permite la auto reparación de las aplicaciones (problemas causados por instalaciones / desinstalaciones mal realizadas, etc).
- Se reducen significativamente el número de reinicializaciones, con lo cual, las instalaciones pueden realizarse en menor tiempo**.**

4.9.3- LINUX

Descripción:

- Sistema operativo libre (desarrollado por voluntarios). "*FREE OPEN CODE*"
- Sistema **multitarea real**.
- Sistema **multiusuario**.
- Sistema **multiprocesador**.
- Basado en **UNIX**.
- Puede coexistir con otros sistemas operativos (en un mismo procesador).

Requisitos mínimos y recomendados:

- **Pentium**.
- **32 Mb de RAM mínimo / 64 Mb recomendado** (4 Gb como máximo).
- Disco duro con un mínimo de **500 Mb de espacio libre**.
- Unidad de **CD-ROM** o **adaptador de red**.
- **Teclado** y **Ratón**.

Características principales:

- Sistema operativo compatible con Unix.
- El sistema lo forman el núcleo del sistema (*kernel*) y un gran número de librerías
- Se distribuye bajo la GNU Public License, por lo que el código fuente es accesible libremente.
- Carga de ejecutables por demanda: Linux sólo lee del disco aquellas partes de un programa que están siendo usadas actualmente.
- Protección de la memoria entre procesos, de manera que uno de ellos no pueda colgar el sistema.

4.9.4- LINDOWS

Descripción:

- Sistema **multitarea real**.
- Sistema **multiusuario**.
- Sistema **multiprocesador**.
- Basado en **Xandros OS**.
- Puede coexistir con otros sistemas operativos (en un mismo procesador).

Requisitos mínimos:

- **Pentium/AMD** a 90MHz.
- **64 Mb de RAM** (4 Gb como máximo).
- Disco duro con un mínimo de **1 GB de espacio libre** o superior.
- Unidad de **CD-ROM** o **adaptador de red**.
- **Teclado** y **Ratón**.

Características:

Lindows es un sistema operativo construido para aprovechar al máximo, en un futuro próximo, el ancho de banda donde la conectitividad con Internet será más frecuente y barata. Precisamente cada día los equipos requieren soluciones más beneficiosas, fáciles de utilizar y altamente personalizables, y en esos aspectos es donde Lindows destaca, pudiendo personalizar los puestos individualmente. Lindows se aprovecha de un interface similar al de Windows, de apariencia más gráfica e intuitiva y conocido por un 80% de los usuarios por la seguridad del sistema Linux. Con esto pretende extenderse salvando los obstáculos causados por la escasez de aplicaciones creadas para Linux. Precisamente es en este aspecto en donde Lindows quiere mejorar respecto a Linux para poder extenderse. **Basado en Xandros OS** y con **Wine,** un emulador de Windows para Linux, Lindows hace posible ejecutar muchas de las aplicaciones diseñadas para Windows.

El nuevo Lindows se puede instalar sobre **Windows 98/NT/XP**, siguiendo el proceso de actualización tradicional de sistema operativo y será compatible con las aplicaciones Windows, principal problema de los usuarios a la hora de funcionar con sistemas puros Linux. Windows está tan extendido a pesar de sus fallos, que las aplicaciones que todos usamos a diario están diseñadas para funcionar con este sistema, por lo que pasarnos a Linux resulta un tanto complicado. Sin embargo, este traumático cambio se ve completamente erradicado gracias al Lindows.

LA MEMORIA CACHÉ 5

5.1- NECESIDAD DE LA CACHÉ

La arquitectura de los últimos microprocesadores está orientada a mejorar el rendimiento, de manera que se ejecuten más instrucciones por unidad de tiempo.

Para llevar a cabo este objetivo los nuevos microprocesadores se apoyan en tres recursos fundamentales:

- Arquitectura superescalar, característica de la arquitectura RISC. Paralelismo explícito. Se compone de instrucciones sencillas.
- Supersegmentación, es decir, segmentación con elevado número de etapas. Esta técnica precisa de técnicas sofisticadas para eliminar riesgos, especialmente en las instrucciones que contengan saltos condicionales.
- Potenciamiento de la memoria caché, para aumentar la velocidad de la memoria.

Un procesador segmentado ejecuta cada instrucción en cinco etapas que son:

1. Búsqueda de la instrucción. (*Fetch*): se accede a la memoria para buscar la instrucción.
2. Decodificación: es trabajo de la CPU y su ciclo es de 10 ns cuando trabaja a 100 MHz.
3. Búsqueda de los operandos: se accede.
4. Ejecución de la instrucción: realizada por la CPU.
5. Escritura del resultado: se accede de nuevo a la memoria .

De estas cinco etapas hay tres que consisten en un acceso a la memoria principal (DRAM), donde están las instrucciones y los datos (búsqueda de la instrucción, búsqueda de los operandos, escritura del resultado) y las otras dos son propias del procesador. Las dos etapas que afectan al procesador se realizan en un ciclo cada una. Las otras tres etapas ocupan un tiempo equivalente al de acceso a la memoria principal.

Si suponemos que la memoria principal DRAM trabaja con un tiempo de acceso de 50 ns, los tres accesos a la memoria principal suman un total de 150 ns . Si además el microprocesador trabaja a 1Ghz, su periodo es de 1 ns, por lo que el resultado anterior representa un desequilibrio notable entre etapas. Este hecho es la causa fundamental de un mal rendimiento.

Hay una laguna entre la tecnología de construcción de procesadores y la de la memoria.

Para que el rendimiento sea óptimo, el objetivo es que todas las etapas duren lo mismo. Debemos disponer de una memoria más rápida. Las cachés son ultrarápidas pero de poca capacidad y muy caras, por lo que no se pueden sustituir las DRAMs por cachés. Por lo que debemos emplear la jerarquía de memoria, la cuál consiste en interponer entre la CPU y la memoria DRAM una memoria ultrarápida (caché).

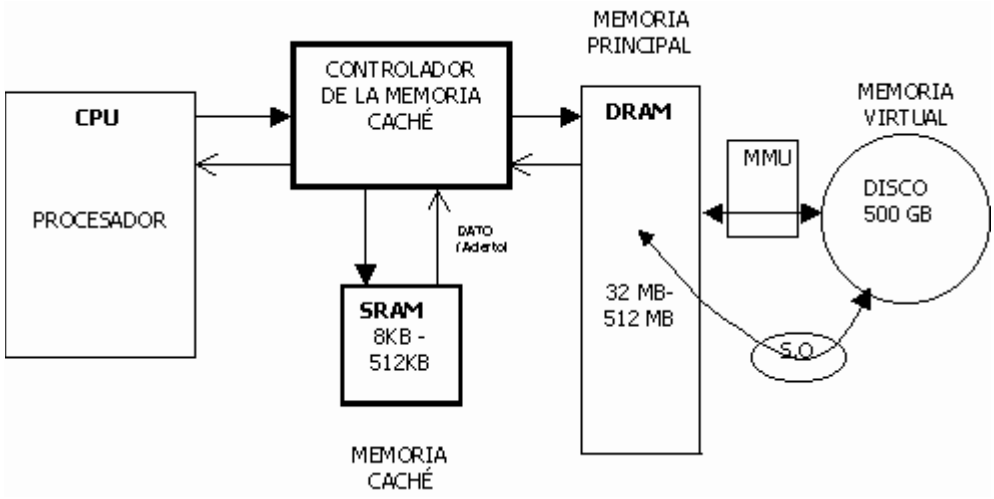


Figura 5.1. Jerarquía de memorias

La memoria caché es una SRAM (RAM estática) que tiene un tamaño comprendido entre 8 KB y 512 KB mientras que la Memoria Principal puede alcanzar cientos de MB.

Como se estudió en el tema de la Memoria Virtual, cuando la CPU solicita una información no presente en la Memoria Principal, la MMU trae el bloque necesario de la Memoria Virtual. Este mismo procedimiento también es empleado para el comportamiento entre la Memoria Principal y la memoria Caché.

La CPU se relaciona con la Memoria Caché y si ésta contiene lo solicitado se tardan unos pocos ns en el acceso. Si hay fallo se debe acceder a la Memoria Principal.

El movimiento de datos se genera de forma que produciéndose una ausencia, la caché recibe de la Memoria Principal el dato pedido y otros contiguos, que previsiblemente va a pedir la CPU. Esto es predecible dependiendo de la programación, ya que puede estar construida mediante las reglas clásicas de la contigüidad espacial y temporal. Es fácil prever la siguiente información que solicitará la CPU debido a la localidad espacial, la CPU tiende a requerir datos que estén en posiciones cercanas físicamente.

Si la caché tiene el dato, solo penaliza el tiempo de acceso a la misma, pero cuando la caché no dispone del dato solicitado, el tiempo empleado es el de acceso a la Memoria Caché más el de acceso a la Memoria Principal.

Así por ejemplo, si empleamos una memoria caché con un tiempo de acceso *tc* , con una tasa de acierto del 90% y una memoria principal con un tiempo de acceso *tm*, la formula que refleja el tiempo medio de acceso al sistema de memoria es:

$$\overline{t} = 0.9 \times tc + 0.1 \times (tc + tm)$$

Si suponemos que el tiempo de acceso a la memoria caché es de 5ns y el tiempo de acceso a la Memoria Principal es de 5 y 50 ns respectivamente, el resultado es:

$$\overline{t} = 0.9 \times 5ns + 0.1 \times (5ns + 50ns) = 10\ ns$$

Un sistema con estas características tiene como tiempo de acceso medio 10 ns , por lo que se ve reflejada la importancia del algoritmo de intercambio de información, es decir, la capacidad que tiene el sistema de predecir las siguientes peticiones de información.

Hay dos factores destacables en la memoria que proporciona la caché:

1. **Factor de velocidad:** Es la relación entre el tiempo de acceso a la Memoria Principal y el tiempo de acceso a la Memoria Caché:

$$\gamma = \frac{tp}{tc}$$

2. **Factor de eficacia:** La eficacia depende en gran medida, del programa que se esté ejecutando, es decir, de cómo está escrito y estructurado el software:

$$\text{Eficacia} = \frac{tc}{t}$$

5.2- PRINCIPIO DE FUNCIONAMIENTO DE LA CACHÉ

Una caché está estructurada en tres bloques importantes:

- **BLOQUE DE ETIQUETAS: RAM-CAM:** Es una memoria de acceso por contenido. No se accede por dirección de memoria, sino que, se compara el valor o dato con los que hay dentro de la memoria y así sabemos si se encuentra o no contenido en ella.
- **BLOQUE DE DATOS ASOCIADOS: SRAM:** Es un conjunto de datos de forma que a cada dato le corresponde una etiqueta. Por ejemplo: dato 0 → etiqueta 0.
  Si hay acierto, la etiqueta que coincide con el dato introducido, devuelve el dato asociado a la misma.
- **LÓGICA DE CONTROL:** Comparadores de "n" bits, tantos como tenga la etiqueta.

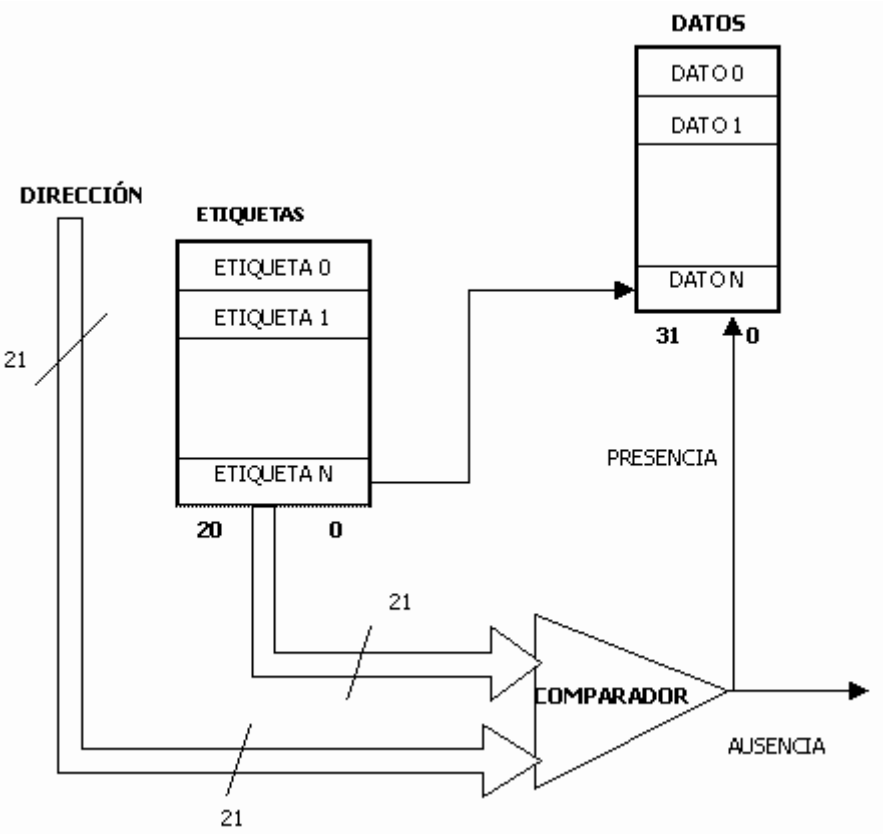


Figura 5.2. Principio de funcionamiento de la caché

El bus de direcciones va a la caché, que tiene dos partes: la de la etiquetas y la de datos. Hay N posiciones de etiquetas y cada etiqueta tiene 21 bits. Los 21 bits de la etiqueta se corresponden con los 21 bits de más peso de la dirección, por eso, el comparador coge las N etiquetas y las compara con los 21 bits de más peso de la dirección. Si alguna coincide, el comparador devuelve un uno, lo que implica presencia. En cambio, si no coincide ninguna, el comparador devuelve un cero lo que implica ausencia , lo que significa que esa dirección no está contenida en la caché.

Aunque hay múltiples organizaciones en la caché, desarrolladas más adelante, concretamente en el ejemplo de la figura 5.2 hay un dato por etiqueta, por lo tanto N datos, existiendo una correspondencia etiqueta-dato. Como por ejemplo: ETIQUETA1 →DATO1.

Si el comparador devuelve un uno (presencia), el dato correspondiente a la etiqueta es transferido a la CPU reflejando los bits restantes, la posición en la que se encuentra el dato. Este sistema es más rápido que el de localizar una dirección en memoria principal.

Si el comparador devuelve un cero (ausencia) se accede a la memoria principal.

5.3- TIPOS DE CONEXIONADO DE LAS MEMORIAS CACHÉ

La Memoria Caché se puede conectar a la CPU en serie o en paralelo.

5.3.1 – Conexión en serie

La CPU sólo se conecta con la caché por lo que todas las peticiones que hace la CPU al bus del sistema son a través de la memoria caché.

Por lo tanto todo lo que necesita la CPU del sistema se lo proporciona la memoria caché y cómo es de tamaño y tiempo de acceso reducido cada vez que el dato está almacenado en la caché, el tiempo de acceso es muy reducido y evita manejar el bus del sistema.

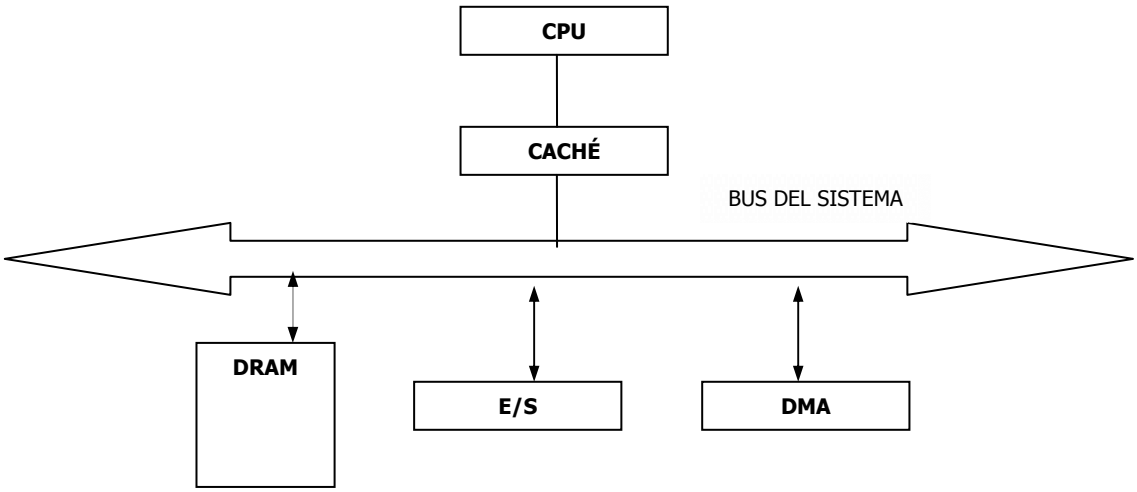


Figura 5.3. Conexión en serie

Si la memoria caché contiene el dato solicitado por la CPU, la parte superior de la figura funcionará de manera independiente y en pocos nanosegundos proporcionará la información, liberando de la búsqueda a la parte inferior de la figura.

De esta forma si el bus del sistema está desocupado, simultáneamente que la CPU ejecuta instrucciones, los módulos de entrada/salida pueden estar trabajando con la memoria principal. Se permite el paralelismo.

El inconveniente de la conexión en serie es la penalización de tiempo ya que cuando la información que necesita la CPU no está en la caché tiene que trasladar la petición al bus del sistema para poder acceder a la memoria principal.

Otro inconveniente es que la caché es de uso obligatorio ; no se puede desconectar la caché y conectar la CPU al sistema.

Cuando el bus del sistema queda libre, puede ser utilizado por todos los elementos que dependan de él.

5.3.2 – Conexión en paralelo

En la conexión en paralelo, todo depende del bus del sistema:

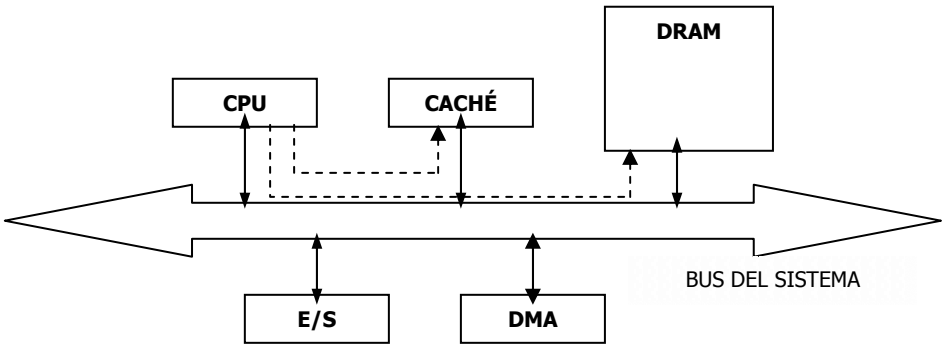


Figura 5.4. Conexión en paralelo

Cada vez que la CPU realiza una petición, la envía simultáneamente a la caché y a la Memoria Principal . Si se encuentra la información contenida en la caché es entregada al bus en pocos ns avisando a la memoria principal para que no continúe con la búsqueda de la información. De esta forma se parará el ciclo. Si la memoria caché no tiene el dato pedido, la memoria principal sigue trabajando, con lo cual no hay penalización de tiempo, lo que es una gran ventaja.

Este sistema se utiliza mucho porque cuenta con otra ventaja, y es que la caché es opcional, es decir, se puede añadir o eliminar sin alterar el funcionamiento del sistema.

El inconveniente es que la CPU realiza todas las peticiones por el bus del sistema, quedando sobrecargado por el continuo flujo de información, lo cual deja muy poco espacio libre para E/S y DMA, ya que el bus del sistema se libera muy pocas veces.

5.4 ARQUITECTURA DEL SUBSISTEMA DE MEMORIA CACHÉ

Las características fundamentales que determinan un subsistema caché son las siguientes:

- Tamaño de la caché.
- Organización.
- Estructura física de una caché.
- Actualización de la caché.
- Actualización de la memoria principal.

5.4.1 – El tamaño de la caché

El tamaño de la caché suele oscilar entre 8KB y 512KB. En muchas ocasiones, no se consigue aumentar el rendimiento mediante cachés de mayor capacidad. Lo que influye son los algoritmos de transferencia de la memoria caché.

Aumentos importantes del tamaño de la caché suponen variaciones pequeñas en el porcentaje de acierto, porque el porcentaje de aciertos se basa en el algoritmo. Según los algoritmos, la mejor capacidad de la caché está entre 32K y256K.

El precio sube de forma exponencial: aumentando el tamaño x8, la tasa de aciertos sólo aumenta un 4%. Se reflejá en la siguiente gráfica.

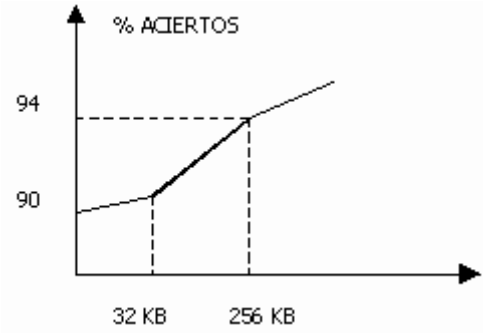


Figura 5.5.Gráfica número de aciertos respecto tamaño de la caché

5.4.2 – Tipos de organización

Existen tres tipos de organización de las memorias caché:

1. **Totalmente asociativa:** Se caracteriza porque cualquier posición de la memoria principal se puede ubicar en cualquier posición de la memoria caché.

   Se usan comparadores de 32 bits, muy caros, con buenos algoritmos.

   Todas las direcciones de la memoria principal que quepan se pueden guardar sin ningún orden preestablecido, no hay normas, hay una flexibilidad total.
   Esta flexibilidad es un problema grave, puesto que la caché necesita los 32 bits de la dirección para compartirla con la etiqueta.

   Cada partición de la caché se puede ubicar en cualquier parte de la memoria principal. Como hay flexibilidad total, la etiqueta ha de contener TODOS los bits de la dirección de la memoria principal a los que corresponden los datos.

Luego el inconveniente de este tipo de organización de la memoria caché es que tiene que tener apuntadas todas las direcciones en la zona de etiquetas y por lo tanto éstas serán muy largas Consecuentemente el comparador va a ser muy lento y caro ya que va a necesitar muchos bits.

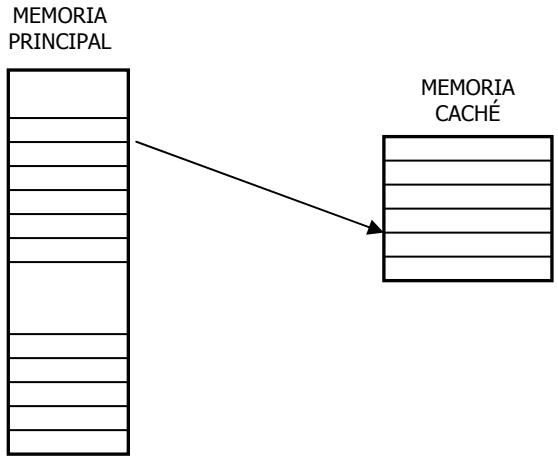


Figura 5.7. Organización de memoria caché asociativa

2. **Asociativa de una vía:** Suponemos que la memoria principal tiene 256 KB y la caché 256 Bytes. La memoria principal se divide en bloques de 256 bytes. Habrá 1K bloques, es decir, 1024 bloques.

   Cada posición de 1 bloque de la memoria principal sólo puede ir a la misma posición de la caché.

   Solo es necesario precisar cuál es el bloque, porque sabiendo el bloque ya se conoce la posición de la caché debido a que todas las direcciones de un bloque están en la misma posición que en la caché.

   La ventaja es que para conexionar la memoria principal necesitamos que el bus de direcciones tenga 18 líneass. La etiqueta, por tanto, sólo necesita 10 bits: 2^{10} = 1K es decir, lo que ocupa un bloque. Sólo necesito los bits necesarios para definir el bloque y me ahorro así los 8 bits.

   Como sólo cabe una dirección de cada bloque en una posición , tendremos que machacar la dirección anterior, cada vez que queramos modificarla. La caché estará sufriendo continuamente modificaciones.

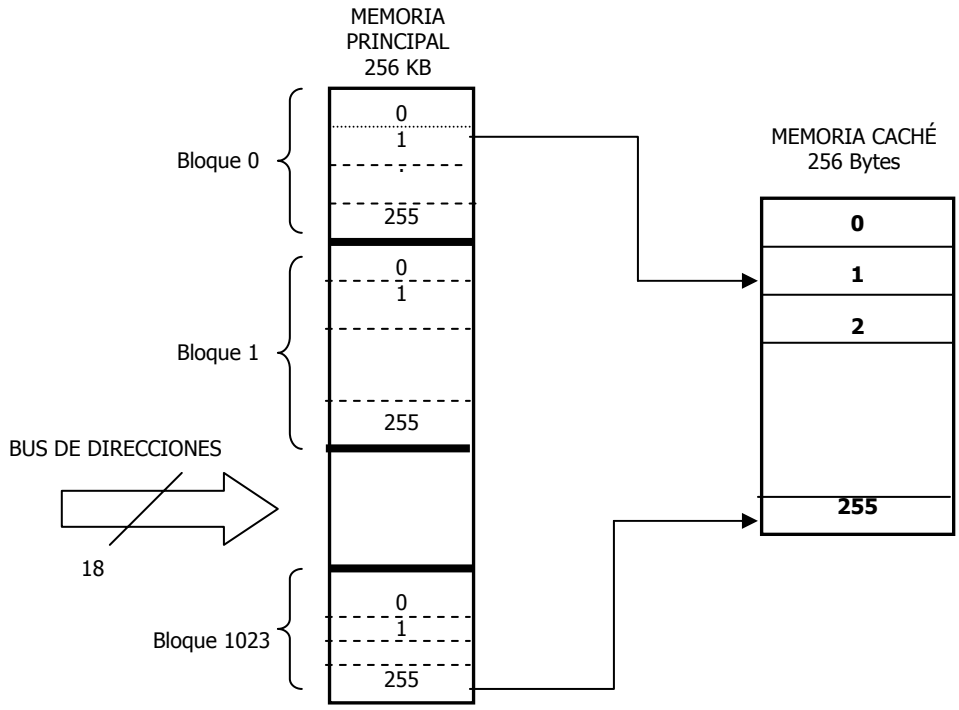


Figura 5.8. Organización asociativa de una sóla vía de la memoria caché

3. **Asociativa de "n" vías:** Su funcionamiento es similar al de una vía pero la caché se va a descomponer en varias vías, no en una sola

La memoria principal se divide en fragmentos iguales a cada uno le corresponde una de las vías funcionando de la misma manera que en el caso anterior, incorporando la ventaja de que no es necesario machacar las posiciones de memoria inmediatamente que surja una modificación .

Con la caché asociativa de N vías, se ahorra gran cantidad de bits debido a que hay correspondencia entre las posiciones de las páginas de la memoria principal y las posiciones de la caché.

Esta organización de la memoria ofrece un mayor rendimiento pero las vías son de menor tamaño. Dependiendo de la aplicación será mejor utilizar la memoria asociativa de 1 vía o la de n vías.

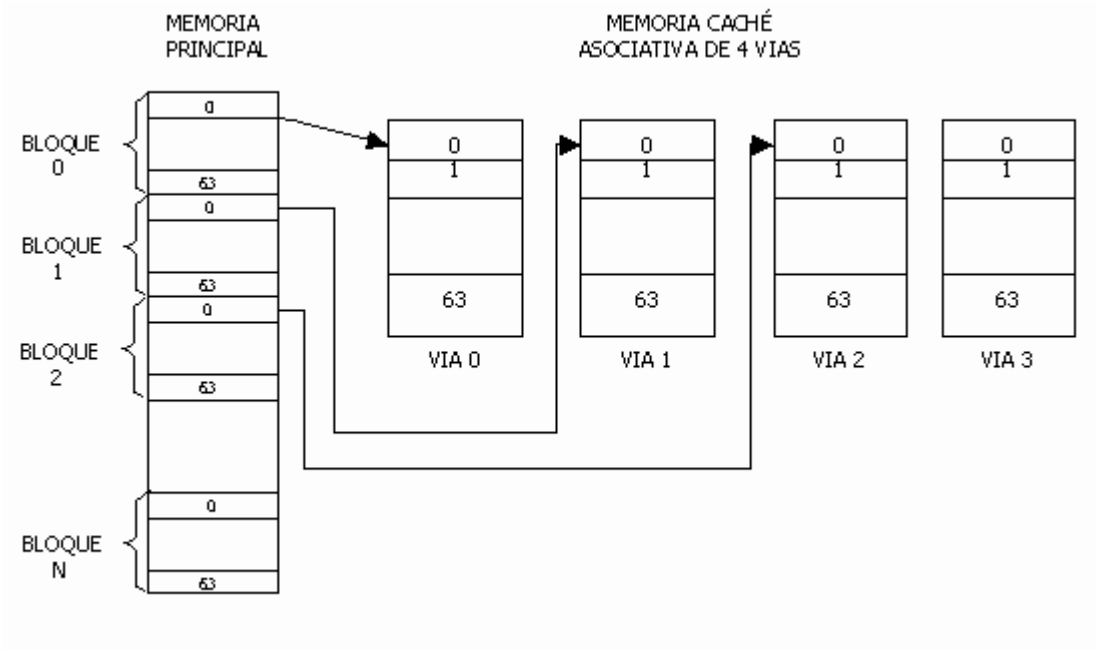


Figura 5.9. Organización asociativa de "N" vías de la memoria caché

5.4.3 – La estructura física de una caché

Las cachés son el elemento diferencial que tienen los nuevos procesadores pentium con respecto a los anteriores. Para referirnos a una caché real presentaremos como ejemplo la del procesador Pentium.

El Pentium tiene 2 memorias cachés independientes: una para datos y otra para instrucciones, permitiendo el paralelismo entre instrucciones y datos. Cada una de ellas es de 8 KB.

Son cachés asociativas de 2 vías. Las etiquetas contienen los 20 bits de más peso de la dirección además de dos bits de código: el WP protege contraescritura y V predice si es línea váilda o no.

Los datos asociados a la etiqueta tiene cada uno 32 bytes (= una línea), es decir, cada vez que la caché se actualiza se le introducen 32 bytes. Cada vía tiene 128 etiquetas. Cada vía guarda 128 x 32 bytes = 4 Kbytes de datos, luego la memoria la dividimos en 4KB.

La sustitución de direcciones se realiza mediante el algoritmo LRU que es aquel que selecciona la dirección que menos se ha utilizado.

Cuando dicha dirección se va a incorporar en una línea y estas están ocupadas, se consultan los dos bits del LRU, y la dirección menos utilizada será la que se machaque.

La estructura interna de la caché del Pentium es la siguiente:

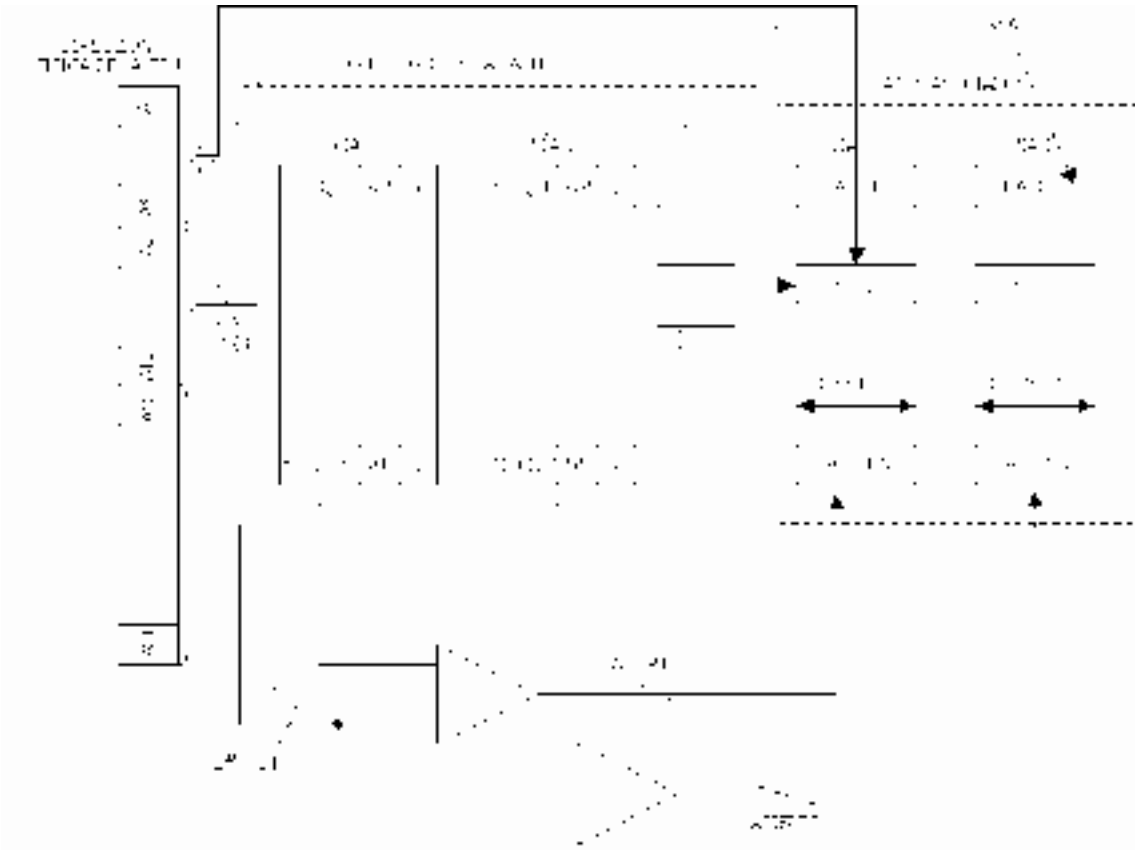


Figura 5.6 Estructura interna de la caché del Pentium

5.4.4 – Actualización de la memoria caché

En el pentium cuando se produce una ausencia en la caché se busca la información de una línea completa en la memoria principal y se carga en la caché, en una vía que esté libre.

Si fuera totalmente asociativa, cualquier línea se podría almacenar en cualquier posición de la caché. Si es asociativa de una vía sólo hay una posibilidad. Si es de varias vías, como el Pentium, hay varias posibilidades.

Si se necesita introducir una posición y hay alguna vía vacía, ésta se ocupa. Si hay una posición que las cuatro vías tengan ocupada, se aplica uno de estos dos algoritmos para extraer una de ellas y machacarla:

- **RANDOM:** Aleatoriamente se elige y se machaca una de las posiciones ocupadas de una de las cuatro vías. El inconveniente es que quizá se machaque la información que a continuación se necesite. Se dice que es el algoritmo de las cachés de bajo coste.
- **LRU:** Se elimina la posición de la vía que menos se haya empleado últimamente, ya que suponemos que es la que menos se va a seguir utilizando. Su funcionamiento se basa en dos bits que apuntan a la vía que menos se ha empleado. Hay dos alternativas para actualizarla:

1) EL DATO PEDIDO VA EN ÚLTIMO LUGAR: El dato último es el que se ha direccionado y se traen los 31 bytes que van delante. Tiene un inconveniente, y es que la CPU hasta que no se carga el último byte no puede trabajar, , ya que los 31 anteriores bytes no le sirven. Tiene que esperar a que termine de cargarse la línea para transferir la información a la CPU. Es un método simple y rápido.

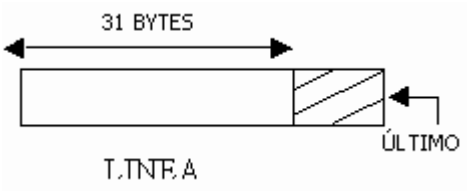


Figura 5.10. Dato pedido en el último lugar

2) EL DATO PEDIDO VA EN PRIMER LUGAR.: La caché, desde el primer dato que llega, ya tiene la información que ha solicitado la CPU. El problema es que es un algoritmo más complejo y por tanto, se necesita más circuitería interna, más transistores de silicio.

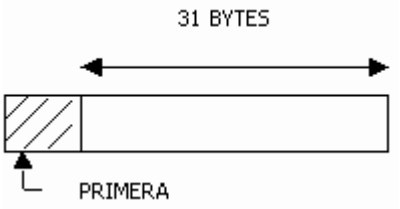


Figura 5.11. Dato pedido en primer lugar

Cuando la caché está en serie la CPU funciona de la siguiente forma:

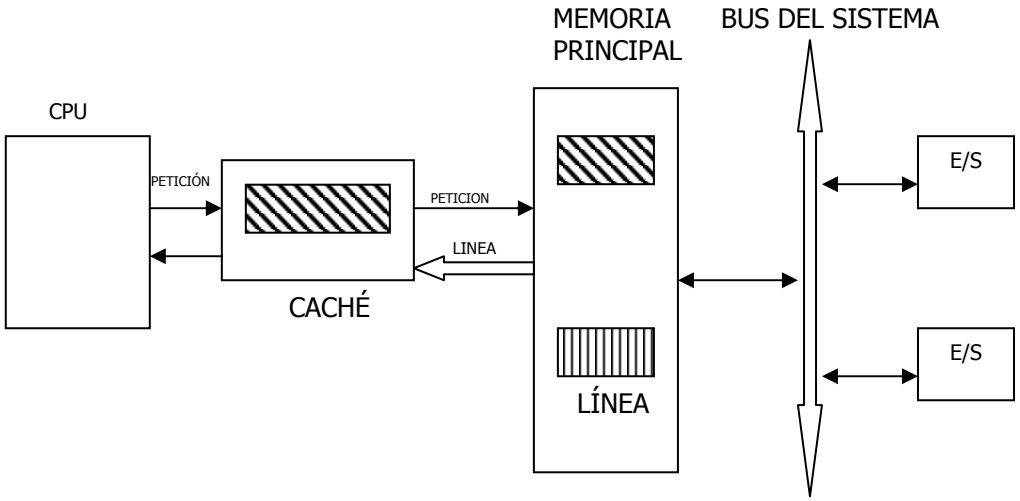


Figura 5.12. Gráfico de la actualización de la memoria principal

La CPU siempre se dirige a la caché. Si va a leer un dato a la caché no tiene ningún inconveniente, pero si la CPU necesita escribir en la caché y modifica una de sus posiciones, la CPU da por concluida su operación resultando que esa posición de la caché tiene una imagen en memoria principal que no tiene constancia de esta modificación, por lo que hay una discordancia. Esto puede dar lugar a errores graves. Por lo tanto se debe escribir lo que hay modificado en la caché en la memoria principal.

La memoria principal se actualiza mediante uno de estos tres métodos:

- **Actualización por escritura inmediata**: Cada vez que la CPU modifica la caché, ésta última manda una orden al bus del sistema y se transfiere la información a la CPU, consiguiendo que no haya errores de coherencia y actualizando así la memoria principal. Muchas veces la escritura de la CPU es repetitiva, ya que la escritura en la caché es continua, y bloquea el bus del sistema. Es anormalmente seguro, pero baja el rendimiento. De esta forma se evitan muchos errores.

- **Actualización por escritura diferida:** La caché dispone de registros intermedios donde cargas temporalmente las modificaciones que ha habido en la caché. Actualiza la memoria principal cuando el bus del sistema está libre, sólo hay que esperar a que el bus del sistema está inactivo. Puede existir falta de coherencia mientras se espera y los periféricos pueden leer datos erróneos de la memoria principal. Es más rápido, pero hay un pequeño riesgo de fallo.

- **Actualización por escritura obligada:** La actualización de memoria principal se produce cuando no queda otro remedio, por lo que no hay nunca fallo.

Hay que actualizar la memoria obligatoriamente cuando:

1. Se accede a una posición de la memoria principal modificada en la caché por la CPU. Antes de dar paso a esa lectura, hay que escribir el dato modificado.
2. Cuando hay que eliminar una línea en la caché porque está llena, en la cual hay un dato modificado. Antes de borrarlo hay que enviar dicho dato a la memoria principal.

5.5 PROTOCOLO MESI

En los sistemas multiprocesador puede haber varias cachés, por tanto, puede suceder que una misma posición de la memoria principal la están empleando dos CPU´s y como consecuencia, permanecer en las dos cachés. Cuando se plantea esta situación, surge la necesidad de asegurar que cualquier acceso a la memoria lea el dato más actualizado. Para evitar esta inconsistencia de la caché Intel desarrolló el protocolo MESI (modified exclusive shared invalid).

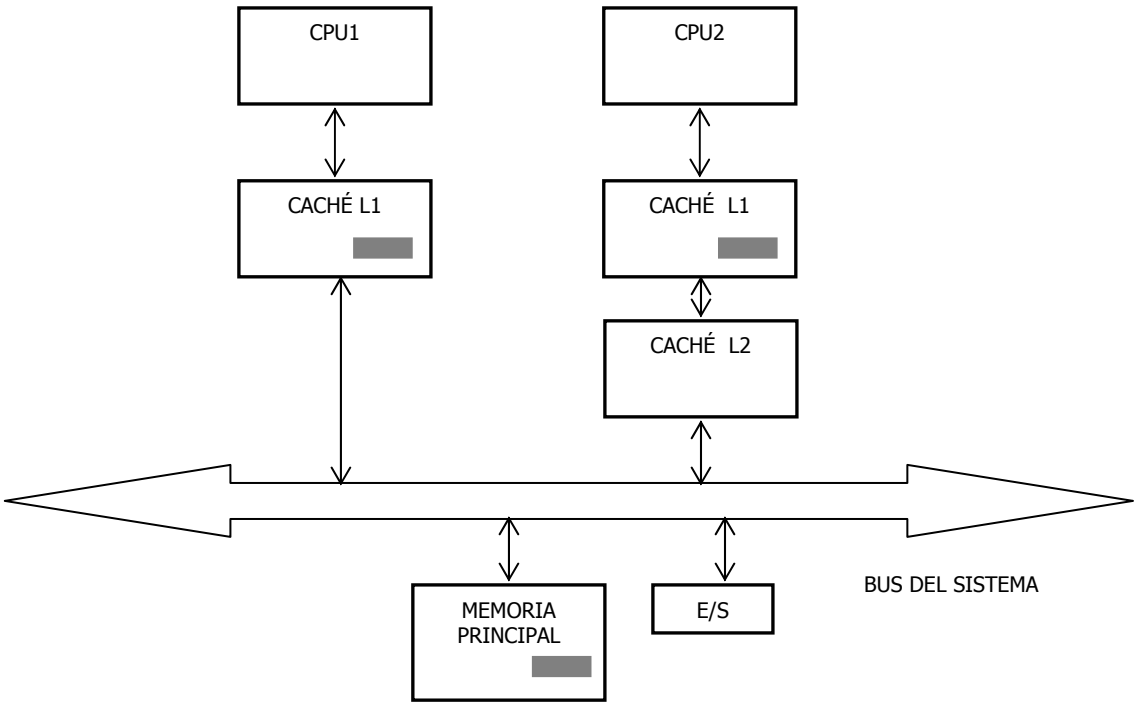


Figura 5.13. Multiprocesador con más de una caché

El protocolo MESI asigna cuatro estados diferentes a cada línea (gestionados por los bits MESI), que definen si una línea es válida (es decir, si existe presencia o no). Si está disponible para otras cachés, o ha sido modificada. Estos estados pueden ser modificados bien por el propio procesador, o bien por unidades lógicas externas tales como otros procesadores o el controlador caché L2.

Los posibles estados de la línea son:

- **M:Modificado:** La línea que lleva dicha M está modificada por 1 escritura del procesador y a la espera de actualizar la memoria principal si es preciso.
- **E:Exclusiva:** La línea sólo la tiene una caché, y está sin modificar. La memoria principal tiene una copia .
- **S:Simultáneo:** Esta línea está repetida en otras cachés. Si se escribe en una de ellas, las demás se invalidan automáticamente.
- **I:Inválido:** Esta línea está invalidada. La lectura de esta posición por parte del procesador genera una ausencia y por tanto un llenado con los datos que provienen de la memoria principal. Si el procesador escribe en esta posición, se procede a actualizar la memoria principal de inmediato.

5.6 NIVELES DE JERARQUÍA EN LA CACHÉ

Una de las técnicas de aumentar el rendimiento, es disminuir el tiempo de acceso a las cachés y otra aumentar la tasa de aciertos.

Si queremos mejorar el rendimiento disminuyendo el tiempo de acceso a las cachés nos resultará complicado ya que el tiempo de acceso a las cachés es una característica tecnológica.

Sin embargo, podremos aumentar dicho rendimiento aumentando la tasa de aciertos.

Para realizar dicho aumento de aciertos debemos:

- **Mejorar los algoritmos de carga en la caché:** Si ese algoritmo es inteligente y se adapta al programa tendrá que tener en cuenta la vecindad temporal y la espacial.
- **Aumentar el tamaño de la caché:** El tiempo de acceso aumenta si aumenta el tamaño, así que tendrá que haber un equilibrio. Como la tecnología no se puede forzar, utilizaremos la jerarquía de memoria, utilizando cachés de primer, segundo nivel, etc...

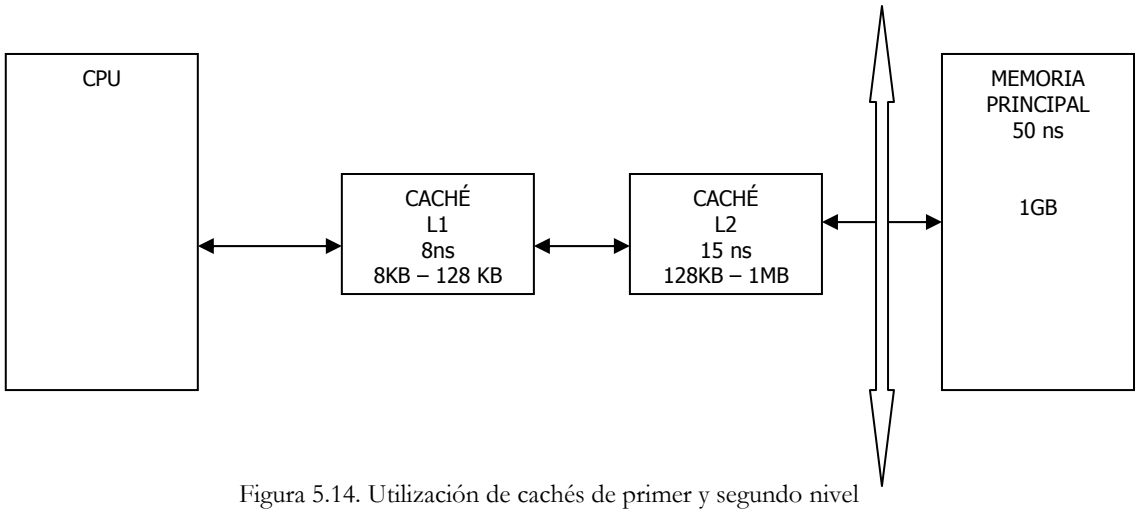


Figura 5.14. Utilización de cachés de primer y segundo nivel

La caché L1 es pequeña y rápida y los algoritmos que se cargan son los algoritmos clásicos (vecindad temporal y espacial). Si la caché L1 no tiene el dato, se transfiere la petición a L2. La caché L2 tendrá todo lo que tiene la caché L1 y algo más. Por lo tanto la caché L2 tiene mayor capacidad y es algo más lenta.

Los algoritmos que cargan a L2 son especiales. Intentan ajustarse al programa.

En la actualidad, en el ITANIUM, la jerarquía de niveles ya llega a tres niveles para lo cual destinan 300 millones de transistores para los niveles de caché y 25 millones de transistores para la CPU.

5.6.1 – Conexionado de cachés de varios niveles:

Hay 2 tipos de conexionado: conexionado en paralelo y conexionado en serie:

- **Conexionado en paralelo:**

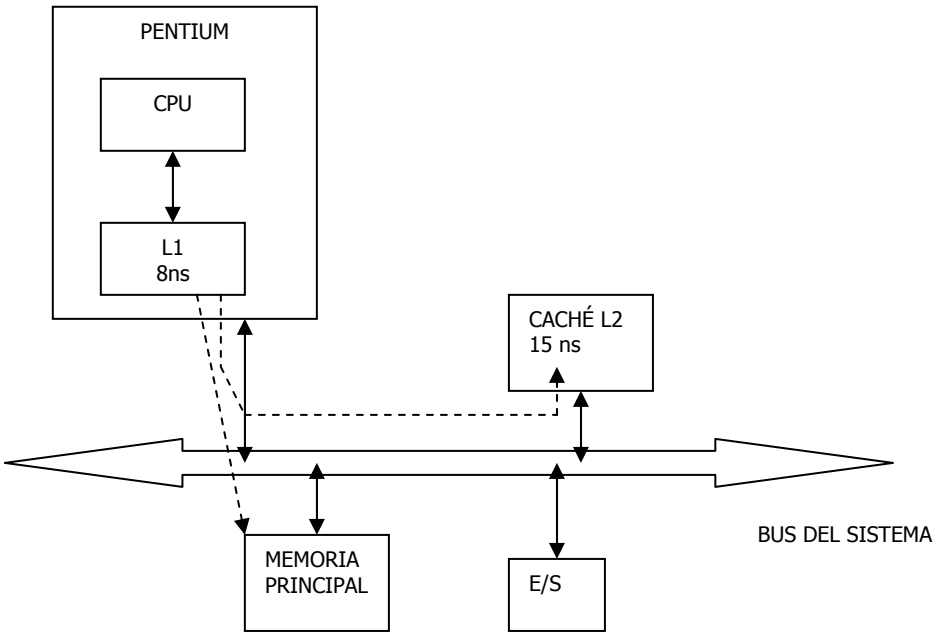


Figura 5.15. Conexionado paralelo

Si la caché L1 da fallo, no importa porque a través del bus del sistema se envía la petición a la caché L2 y a la memoria principal. Si la caché L2 no la tiene en su memoria, no habrá penalización de tiempo puesto que la memoria principal y la caché L2 reciben al mismo tiempo la petición. La caché L2 es optativa, y podremos añadirla o eliminarla cuando queramos sin dañar ni alterar el conexionado.

El bus del sistema recibe todos los fallos de la caché L1 y por lo tanto siempre se está empleando, lo cual es un inconveniente a tener en cuenta.

- **Conexionado en serie:**

Desventajas del conexionado en serie:

1. Hay penalización de tiempo: Si la caché L2 da fallo transcurrirá tiempo hasta que la caché L1 reciba la petición.
2. La caché L2 es obligatoria porque la caché L1 no se puede conectar directamente al bus del sistema.

Ventajas del conexionado en serie:

1. El tráfico de peticiones a la memoria principal disminuye considerablemente y por lo tanto el bus del sistema está desocupado la mayor parte del tiempo y puede hacer frente a los sistemas adheridos.

El conexionado en serie es el siguiente:

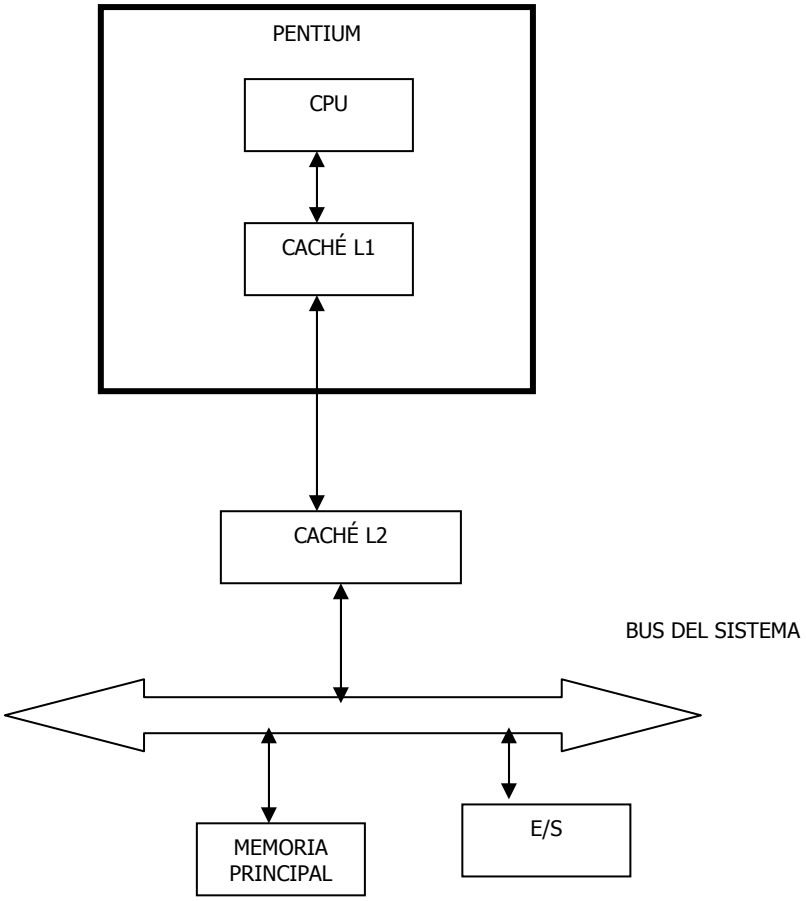


Figura 5.16. Conexionado serie

ARQUITECTURA DEL PENTIUM 6

6.1. INTRODUCCIÓN

El 17 de mayo 1993 Intel lanzó al mercado el primer microprocesador Pentium (5ª generación), y bastó solo un lustro para que el mundo entero se inundara de millones de computadoras que soportaban diferentes versiones de este microprocesador. En el segundo semestre de 1993 se consiguieron vender 100.000 Pentium a un precio de 958 dólares cada uno, cifra poco relevante si comparamos con los 30 millones de 80486 que vendieron ese mismo año y muchísimo menos si se tiene en cuenta los miles de millones de diversos Pentium que se han fabricado. Hoy en día el computador se ha convertido en una herramienta imprescindible en la vida de los seres humanos.

El microprocesador Pentium sigue teniendo una arquitectura interna IA-32 de 32 bits, similar a la de sus predecesores 80386 y 80486. Además mantiene la compatibilidad binaria de software de manera que se ejecutan en los nuevos todo lo que se desarrolló a partir del 8088. Esta política comercial ha proporcionado a Intel su indiscutible éxito de ventas dejando atrás así a sus competidores más directos como Motorola. El Pentium es básicamente evolutivo por lo que se ha denominado como un diseño "80486+".

El primer modelo de Pentium, el Pentium I fue construido con 3,1 millones de transistores (muchos más que el 8086 que poseía 29.000 transistores aproximadamente), fabricado con una tecnología BICMOS de 0,8 micras y funcionaba con una frecuencia de 60 MHz. Utilizaba una tensión de alimentación de 5 voltios y su rendimiento era de 100 MIPS-VAX.
En el pasado año, en el 2002, el Pentium IV ya contenía 42 millones de transistores (se puede apreciar un gran aumento en el número de éstos). Su tecnología ya era de 0,3 micras, usando una frecuencia de 2,4 GHz y utilizando un voltaje inferior a 3 voltios, consiguiendo un rendimiento de 500 MIPS-VAX.

En la siguiente tabla podemos apreciar la evolución de las primeras versiones del Pentium:

PROCESADOR	FRECUENCIA	TECNOLOGÍA	VOLTAJE	BUS	MULTIPLICADOR
P60	60 MHz	0,8 µ	5 V	60 MHz	-
P66	66 MHz	0,8 µ	5 V	66 MHz	-
P75	75 MHz	0,6 µ	3,52 V	50 MHz	1,5
P90	90 MHz	0,6 µ	3,52 V	60 MHz	1,5
P100	100 MHz	0,6 µ	3,52 V	66 MHz	1,5
P120	120 MHz	0,35 µ	3,52 V	60 MHz	2
P133	133 MHz	0,35 µ	3,52 V	66 MHz	2
P150	150 MHz	0,35 µ	3,52 V	60 MHz	2,5
P166	166 MHz	0,35 µ	3,52 V	66 MHz	2,5
P200	200 MHz	0,35 µ	3,52 V	66 MHz	3

Tabla 6.1 – Tabla con las características relevantes de las primeras versiones de Pentium

El incremento del rendimiento y la frecuencia de trabajo supusieron un problema para el primer Pentium. El inconveniente consistía en la alta temperatura que podía llegar a alcanzar debido al aumento de transistores y a la miniaturización de los mismos. Había que disipar 16 watios cuando se trabajaba a la máxima potencia (5 Voltios x 3,2 Amperios = 16 Watios). Por este motivo, se disminuyó la tensión de alimentación hasta 3,3 VDC. La solución era clara, conseguir un mejor sistema de refrigeración.

Existen dos segmentos de mercado para los cuales un ordenador basado en el procesador de la familia Pentium puede ser una solución muy aconsejable. Uno de ellos es el de los ordenadores personales con altas prestaciones, y el otro para el uso como servidores de redes de área local y sistemas multiprocesador. Como ordenador de sobremesa, los procesadores de la familia Pentium tienen la enorme ventaje de que sobre él funcionan todos los principales sistemas operativos, como pueden ser UNIX, Windows 95, Windows NT, OS/2, Solares... Todas la aplicaciones actuales funcionan mucho más rápido bajo un Pentium gracias a sus grandes prestaciones y especialmente se desarrollan estos procesadores para aumentar al máximo las posibilidades de las aplicaciones 3D, de tratamiento de imágenes, de vídeo, sonido y de reconocimiento de la voz.

Por último, mencionar que una novedad que incorporó el Pentium fue la división de la memoria caché de primer nivel L1 en dos secciones independientes de la misma capacidad (8 KB), una dedicada a las instrucciones y otra a los datos. También destaca la ampliación del bus de datos externos a 64 líneas bidireccionales y la potenciación de la Unidad de Coma Flotante, que hasta este modelo, Intel no había prestado mucho interés.

6.2. ARQUITECTURA INTERNA DEL PENTIUM

El procesador Pentium es un miembro de la familia Intel de microprocesadores de propósito general de 32 bits. Al igual que los demás miembros de su familia, el 80386 y el 80486, su rango de direccionamiento es de 4 GBytes de memoria física o principal, ya que sus direcciones físicas son de 32 bits (2^{32} = 4GB) y 64 TBytes de memoria virtual, ya que sus direcciones virtuales son de 46 bits (2^{46} = 64 TB). Proporciona unas prestaciones más elevadas gracias a una arquitectura mucho más optimizada. Su bus de datos es de 64 bits, el de direcciones es de 32 bits y el de control de 1 bit.

A continuación vemos en la figura 6.1, un diagrama de bloques simplificado. En él, podemos apreciar los cinco bloques (uno de los cuales se subdivide en tres partes) en los que se divide la arquitectura interna del pentium. Estos bloques son:

- Subsistema de memoria caché
- Unidad de enteros superescalar
- FPU: coprocesador matemático
- Sistema de predicción de saltos condicionales
- BIU: unidad de interfaz con el bus
    - Monitor de presentaciones
    - Bus a ráfagas
    - Redundancia funcional

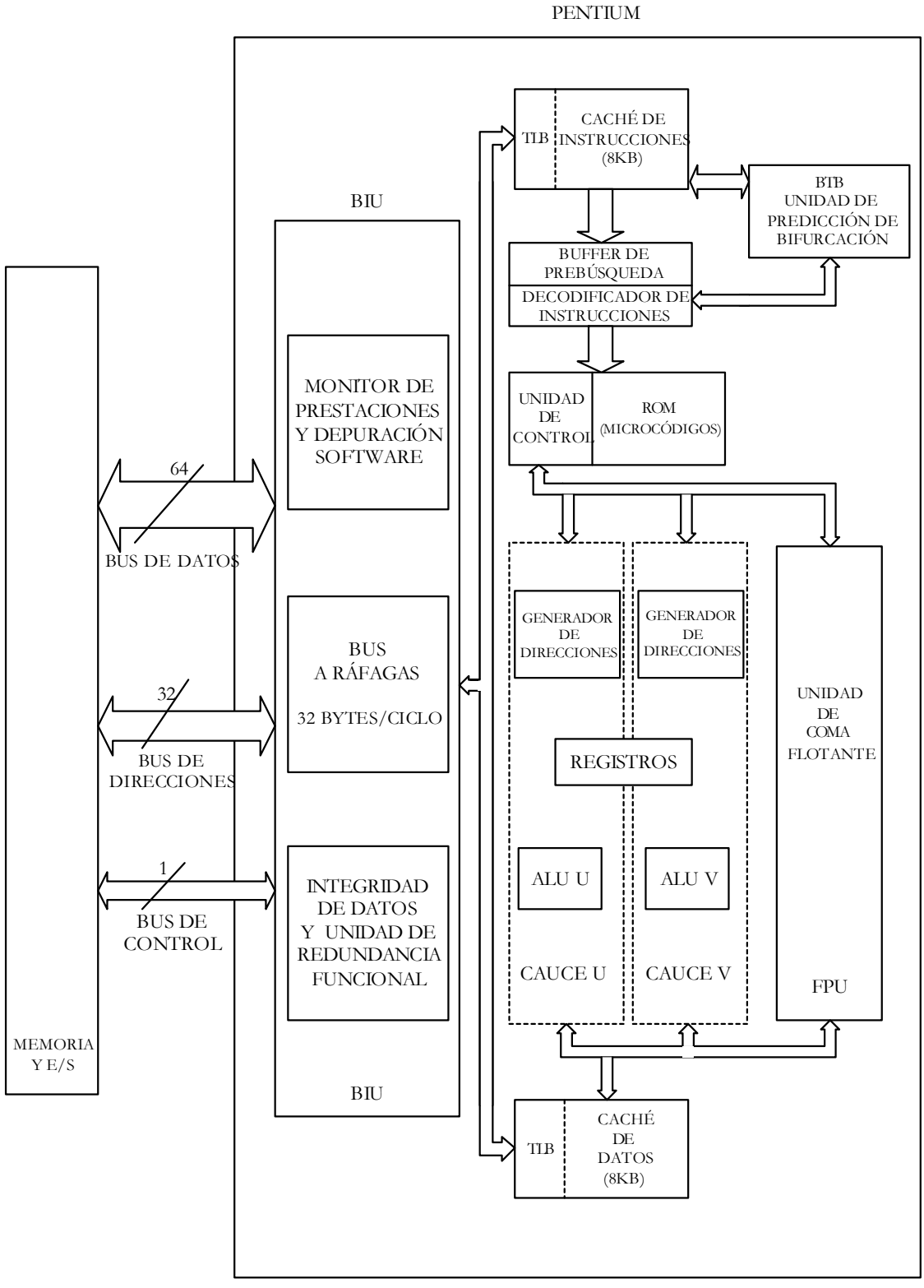


Figura 6.1 – Arquitectura del Pentium

6.2.1. Subsistema de memoria caché

El subsistema de memoria caché se compone de dos memorias caché independientes de 8KB cada una: una para almacenar instrucciones (código) y otra para almacenar datos. Este esquema acelera las prestaciones y la capacidad de transferencia del procesador. Por ejemplo, durante la prebúsqueda las instrucciones se obtienen de la memoria caché de instrucciones. Si hubiese una única memoria caché, no podría realizarse un acceso a un dato al mismo tiempo. Con memorias caché independientes para instrucciones y datos ambas operaciones, de prebúsqueda y acceso a datos, pueden realizarse simultáneamente.

Las dos memorias cachés son memorias asociativas de dos vías (ver figura 6.2) que utilizan como unidad de información una línea que es de 32 Bytes; 8 dobles palabras, es decir 8 x 2 x 16 bits = 256 bits = 32 Bytes (el doble que en su antecesor 80486), ya que el bus externo del Pentium es de 64 bits. De este forma, en un acceso de tipo ráfaga se puede llenar una línea completa de caché, igual que ocurría con el 80486. Los buses independientes que abastecen a las cachés internas desde la unidad de bus externo son de 64 bits cada uno.

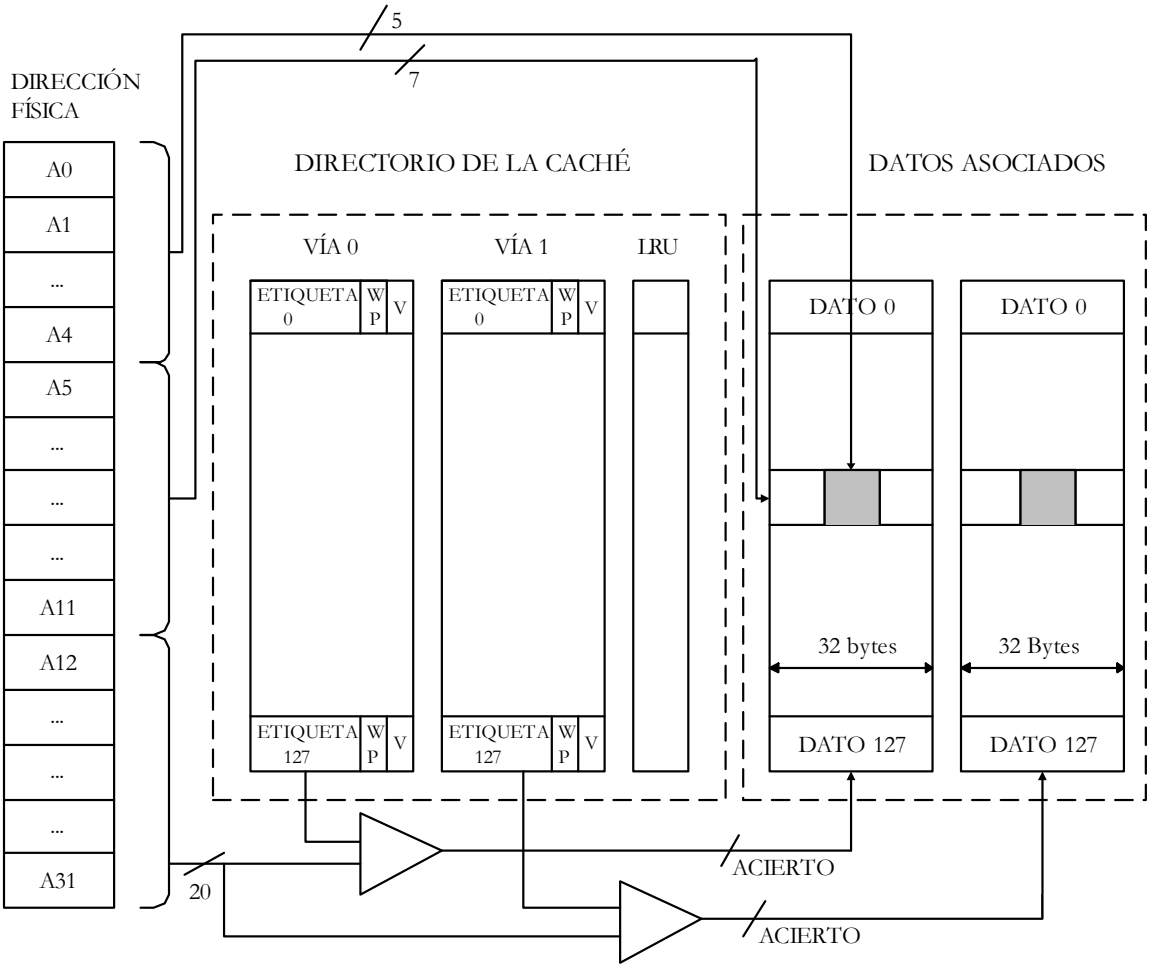


Figura 6.2 – Arquitectura del Pentium

Como podemos observar en la figura 6.3, el bus que parte de la caché de datos es de 64 bits, mientras que el que conecta la caché de instrucciones con los registros de prebúsqueda de instrucciones es de 256 bits.

Figura 6.3 – Caché de datos y caché de instrucciones del Pentium

La caché de datos utiliza el protocolo MESI (modified exclusive shared invalid) para asegurar la consistencia de datos entre la memoria principal y los cachés de todos los procesadores que integren el sistema multiprocesador, es decir, puede haber varias cachés y puede darse que dos CPU´s estén empleando la misma posición de la memoria principal. Así dicha posición se encontrará en ambas cachés. Por este motivo, dicho protocolo asegura que se lea el dato que este más actualizado. Por ello, cada línea de la caché puede tener cuatro estados diferentes: M (modificado, por lo que la línea no está actualizada en la memoria principal), E (exclusivo, si únicamente se encuentra en esa caché), S (simultaneo, si la línea está repetida en varias cachés) e I (inválida, si su contenido no sirve).

Cuando se precisa almacenar instrucciones o datos en la caché correspondiente y está totalmente ocupada con valores válidos, se usa el algoritmo de sustitución de líneas LRU (Last Recently Used). Se reemplaza la línea menos recientemente usada, es decir, la que hace más tiempo que no se usa. Por ejemplo, cuando se accede a una página, el bit accedido A, que se encuentra en el registro de control CR3, se pone a 1. Este bit lo maneja el sistema operativo para llevar la cuenta del número de accesos que tiene cada página. De este modo, el algoritmo LRU consulta este número de accesos para eliminar de la memoria, cuando ésta esté llena, la página que menos se hay usado recientemente.

Las caches son de escritura obligada (Write Back), mientras que su antecesor 80486 era de escritura inmediata, es decir, se almacenaban en la caché y en la memoria principal al mismo tiempo. Que una caché sea de escritura obligada implica que los resultados de las operaciones no se transfieren a memoria principal sino que se quedan dentro de la caché hasta que sea preciso actualizarla, porque no queda otro remedio. Su principal ventaja por tanto es que nunca da fallo.

Existen dos situaciones que obligan a actualizar la memoria principal:

- Cuando se va a machacar una línea de la caché porque está llena y ésta no ha sido transferida a memoria principal.
- Cuando algún otro procesador, por ejemplo DMA intenta acceder a una posición de memoria principal cuyo dato está en la caché y ha sido modificado por la CPU. Por tanto, habrá que detener este acceso hasta que se actualice la memoria principal. Una vez actualizada, se prosigue con el acceso.

En el tipo de escritura obligada, al no existir transferencias de resultados a la memoria principal, las operaciones se terminan antes.

Por último, cabe destacar que dos cachés que integran el subsistema de memoria caché son de primer nivel (L1) y admiten una interconexión fácil con caches de segundo nivel (L2) con lo que se aumentan significativamente las prestaciones.

6.2.2. Unidad de enteros superescalar

Primeramente, aclararemos el significado de la palabra superescalar. Este término hace referencia a que en su interior existe más de una unidad de ejecución dedicadas a realizar las mismas funciones. Así queda reflejado en la siguiente figura 6.4.

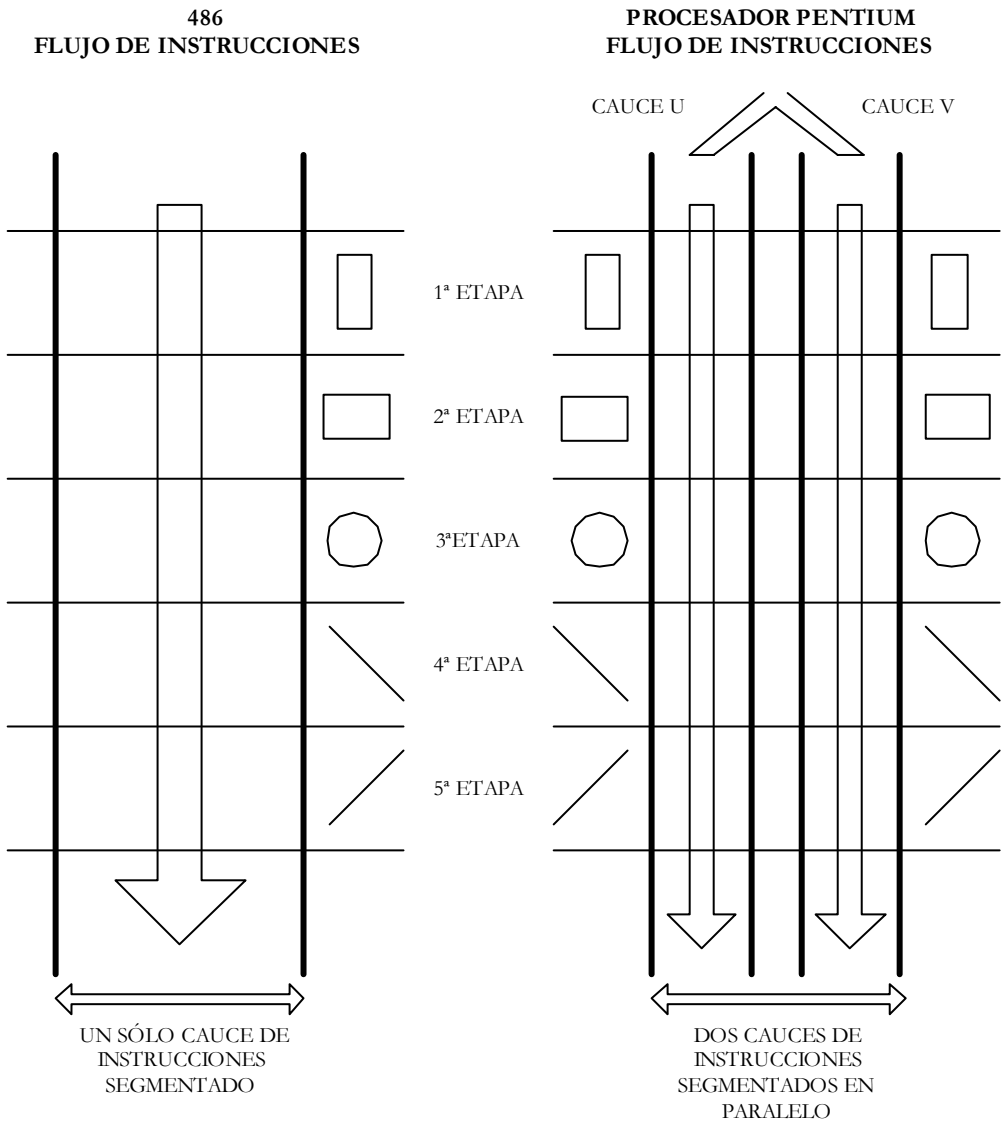


Figura 6.4 – Arquitectura superescalar del Pentium

En este caso, existen dos cauces (U y V) que operan en paralelo para ejecutar las instrucciones de números enteros. Son independientes entre sí, ya que son capaces de funcionar uno independientemente del otro. Es como si existieran dos procesadores del tipo 80486 trabajando al mismo tiempo, por lo que el Pentium podría proporcionar dos resultados enteros por

ciclo de reloj, es decir, ejecutamos dos instrucciones en cada ciclo de reloj. Cada unidad de enteros tiene un cauce segmentado de instrucciones de cinco etapas:

- Prebúsqueda de instrucciones
- Decodificación
- Cálculo de la dirección efectiva (búsqueda de operandos)
- Ejecución
- Escritura de los resultados

Estas etapas del cauce segmentado siguen el orden de la siguiente figura 6.5.

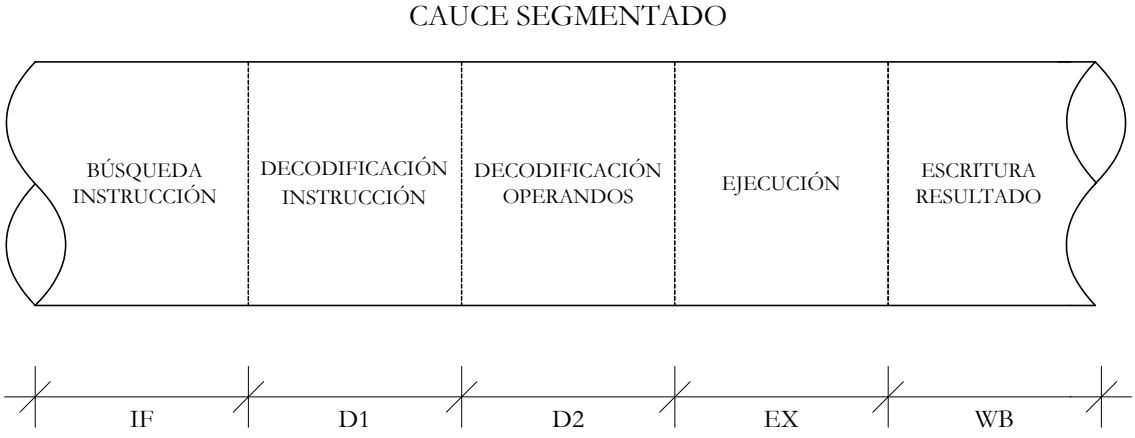


Figura 6.5 – Etapas del cauce segmentado

Cada unidad de proceso interno tiene su propia unidad aritmético lógica (ALU) con un circuito de generación de direcciones exclusivo y un interfaz específico con la memoria caché de datos. Los resultados de las operaciones se almacenan en la caché interna y no se transfieren a la memoria principal a no ser que sea necesario. Estas dos ALUs tienen características diferentes.

La ALU A que pertenece al cauce U es más simple que la del cauce V. El cauce U sólo ejecuta instrucciones simples y del núcleo RISC. Por otra parte la ALU B correspondiente al cauce V es más potente y utiliza instrucciones complejas de tipo CISC.

El bloqueo en la ejecución paralela de instrucciones se realiza de forma transparente para el software y el usuario, y también cuando existe dependencia entre los operandos de las instrucciones. Por ejemplo, si una instrucción realiza una operación que deja el resultado en el registro EDX, la siguiente si utiliza el registro EDX como uno de los operandos origen para cualquier otra operación.

El Pentium intenta paralizar al máximo la ejecución de instrucción, sólo si se garantiza la integridad de los datos, y es capaz de ejecutar alrededor de 1,3 instrucciones por cada ciclo de reloj, rompiendo por tanto la mítica barrera de conseguir la ejecución de la instrucción en cada ciclo de reloj.

Ahora pasaremos a explicar el funcionamiento de la segmentación. La unidad de prebúsqueda manda una dirección a la caché de instrucciones. Si la caché tiene dicha dirección manda una línea de información (32 bytes) a uno de los buffer de prebúsqueda que a su vez pasará la dirección en cuestión a la unidad decodificadora donde decodificará la información. Inicialmente las instrucciones son decodificadas para ver si pueden ser ejecutadas a la vez. En caso afirmativo,

una instrucción irá al cauce U y otra al V para realizar simultáneamente, ya que no existen dependencias entre ellas. En caso contrario, es decir, que existen dependencias entre ellas, la primera deberá completar su ejecución antes de que comience la segunda.

Cuando se predice un salto, la dirección de esta instrucción es demandada por la caché de instrucciones. Si se encuentra allí, una línea de código se manda al otro buffer de prebúsqueda de tal manera que se impide cualquier retraso cuando la instrucción branch se ejecute. Si no hay instrucciones de este tipo ambos cauces son tratados conjuntamente, realizando las prebúsquedas linealmente.

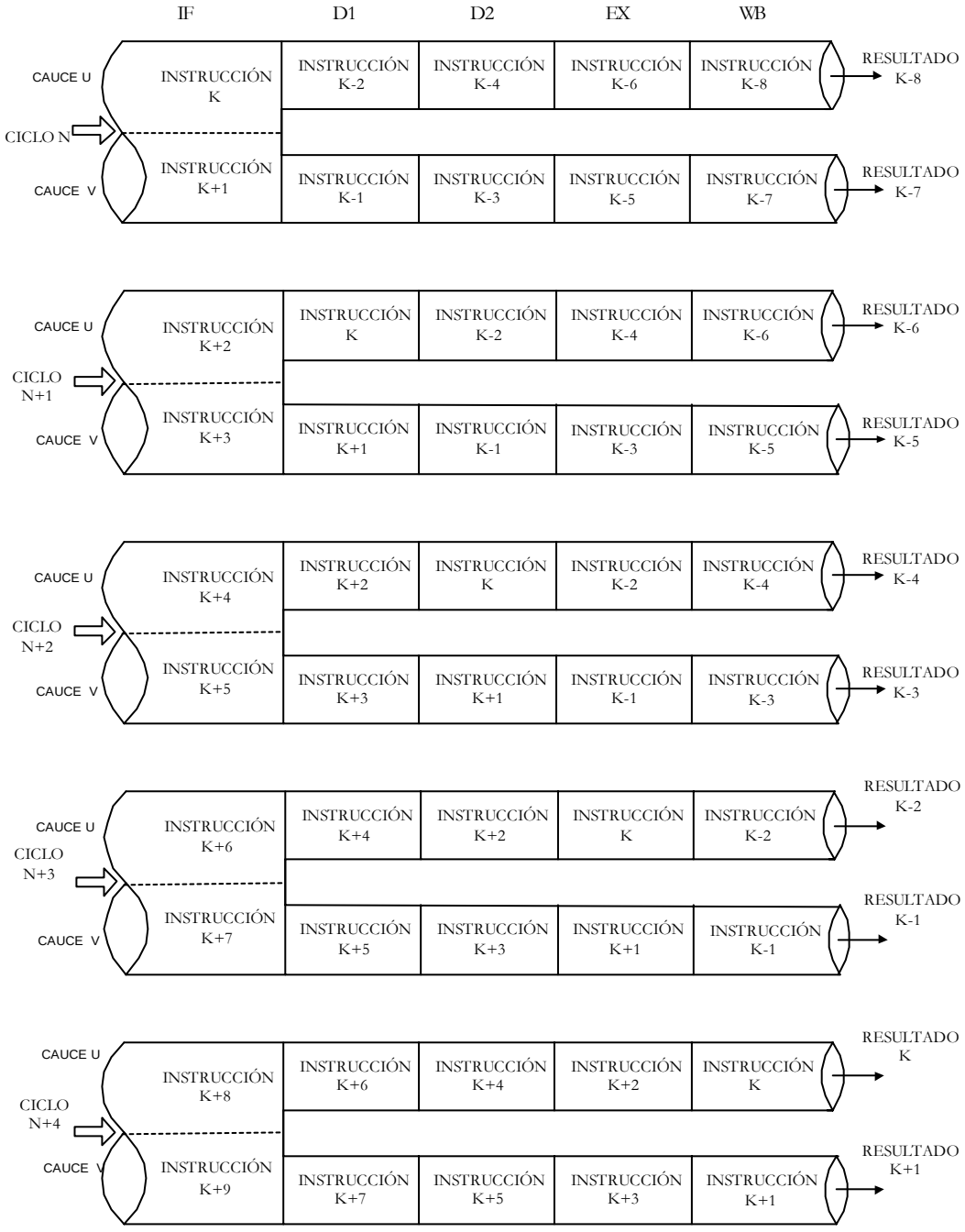


Figura 6.6 – Paso de las instrucciones a través de los dos cauces

Con esta arquitectura se pueden introducir y obtener dos instrucciones en cada etapa del cauce (ver figura 6.6). A esto en inglés se le denomina "pairing" o emparejamiento. Durante el primer ciclo de reloj un par de instrucciones realizan la prebúsqueda; en el segundo ciclo de reloj las dos instrucciones se tratan en paralelo, una en el cauce U y otra en el cauce V , claro está, sino existen dependencias entre ambas instrucciones. En un tercer ciclo de reloj se decodificarán para q en el último se ejecuten. Por todo ello se deduce que el máximo número de instrucciones que puede ejecutar el Pentium son dos.

6.2.3. FPU: Coprocesador matemático

La FPU es un coprocesador que opera con unidades enteras de otros procesadores, de los cuales coge sus instrucciones desde el mismo decodificador y secuenciador que la unidad de enteros, compartiendo con esta última el bus del sistema. Cabe destacar que la unidad de enteros y la FPU operan independientemente y en paralelo, como podemos observar en la figura 6.7.

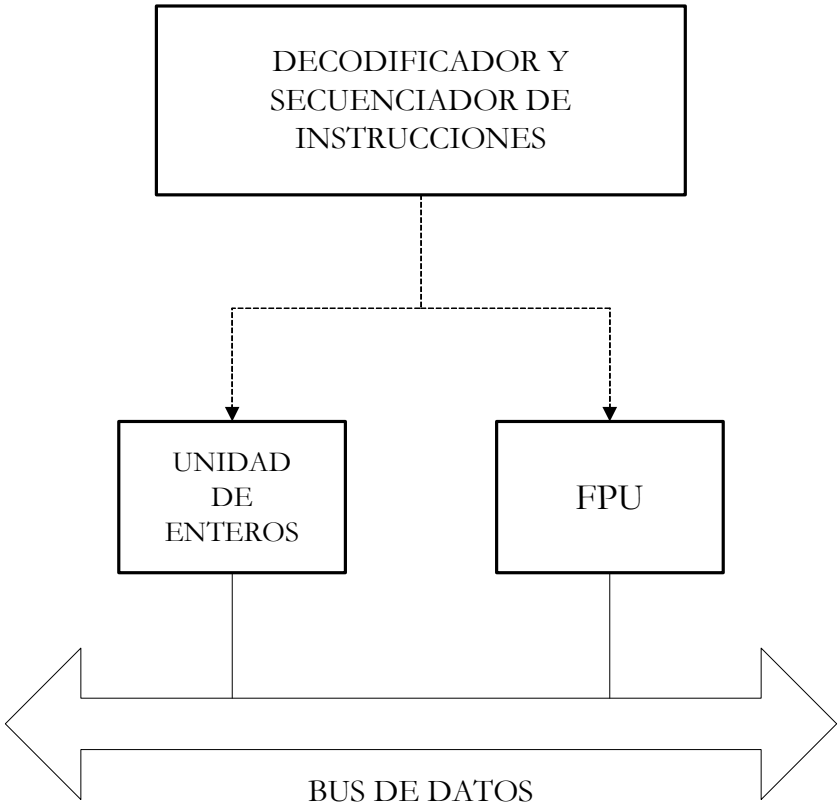


Figura 6.7 – Relación de la FPU con la unidad de enteros

Este bloque se ha rediseñado totalmente respecto al que se utilizaba en el 80486. Sin embargo sigue siendo compatible 100%. Utiliza nuevos algoritmos que aceleran la ejecución de las operaciones (hasta tres veces más rápido que con el 80486) e incluye elementos de hardware dedicados (un multiplicador, un sumador y un divisor).

La FPU consta de un cauce segmentado de instrucciones de 8 etapas que permite obtener resultados partiendo de instrucciones de coma flotante en cada ciclo de reloj. Las etapas son las siguientes:

1º : Prebúsqueda de instrucciones
2º : Decodificación
3º : Cálculos de la dirección efectiva
4º : Ejecución
5º : Ejecución de las instrucciones de coma flotante
6º : Ejecución de las instrucciones de coma flotante
7º : Escritura de los resultados
8º : Informe de posibles errores

Para llevar a cabo estas instrucciones el coprocesador matemático internamente posee registros. Son 8 registros de datos y los siguientes registros especiales:

- Registros de datos, que es donde se guardan los operandos y los resultados. Consta de 8 registros (de R0-R7) de 80 bits de longitud cada uno. Un bit es para el signo, 15 bits para el exponente y los 64 restantes para la mantisa.

- Registros de estado, son registros de 16 bits que indican la situación actual de la FPU, los flags incluidos en estos registros son activados por la FPU para mostrar el resultado de sus operaciones.

- Registros de control, estos registros controlan la precisión de la FPU y los métodos de redondeo usados.

- Registros de palabra, tienen 16 bits y están divididos en campos de 2 bits cada uno (se le denomina tag). Así contendrá 8 tag cada uno del 0 al 7. Cada tag hace referencia a un registro de datos, numerados del R0 a R7.

- Registro puntero de instrucciones, se trata de un registro de 48 bits que guarda la dirección RS (16 bits) y el desplazamiento (32 bits) de las direcciones virtuales de las últimas instrucciones que hemos usado.

- Registro puntero al último operando, también denominado puntero dato, se trata de un registro de 48 bits que guarda la dirección RS (16 bits) y el desplazamiento (32 bits) de las direcciones virtuales de los últimos datos que hemos usado.

- Registro de código, se trata de un registro que consta de 11 bits. La FPU almacena el código de las últimas instrucciones ejecutadas que no sean de control.

El coprocesador puede obtener y escribir datos en memoria de los siguientes tipos:

- Entero: Words de 16 bits, Dword de 32 bits y Qwords de 64 bits.
- Real: Words de 16 bits, Dword de 32 bits, Qwords de 64 bits y Twords de 80 bits.
- Simple precisión en coma flotante.
- Doble precisión en coma flotante.
- Doble precisión expandida en coma flotante.
- Entero con signo.
- BCD.

6.2.4. Sistema de predicción de saltos condicionales

Una de las razones que más influyen en el bajo rendimiento del procesador son los saltos condicionales, que obligatoriamente introducen tres burbujas en el cauce (tres ciclos del cauce sin nada) ya que no se sabe la siguiente instrucción a ejecutar por lo que hay que esperar a ver si se cumple la condición para saber cuál es dicha instrucción.

Los fabricantes intentan eliminar estas burbujas del cauce. Intel lo hace prediciéndolo en un solo ciclo.

Para ello utiliza dos elementos:

- Software: Consiste en un potente algoritmo estadístico.

- Hardware: (BTB) Buffer de destino de las bifurcaciones. ("Branch Target Buffer"). Es una caché ultrarrápida que tiene 256 posiciones donde se guardan los resultados de las 256 últimas instrucción de salto BRANCH (salto condicional).

  Cuando una instrucción supone un salto la BTB recuerda dicha instrucción y la dirección de salto efectuada y predice, aplicando ciertos algoritmos en qué dirección se va a producir el salto la próxima vez que se ejecute. Hay una tasa de acierto del 90%. En este caso, si la predicción es correcta, la bifurcación se realizará en 0 ciclos de reloj, puesto ésta ya se realizó, y se siguieron buscando instrucciones en dicha dirección. Por otro lado, si la predicción falla (en un 10% de los casos) habrá una penalización de tiempo debido a que se han metido en el cauce tres instrucciones erróneas por lo que hay que deshacer todas las operaciones realizadas con dichas instrucciones.

En la figura 6.8 se muestra como está conectada la BTB a la caché de instrucciones y al decodificador de las mismas, de tal forma que si la BTB acierta existe una recompensa de tiempo ya que no hay que recoger la instrucción siguiente de la caché.

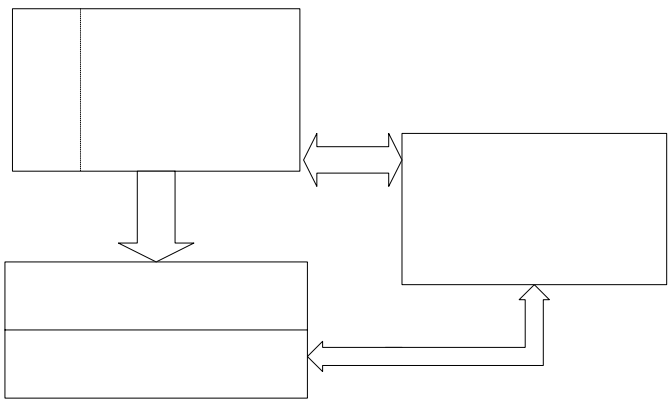


Figura 6.8 – Conexionado de la BTB

6.2.5. BIU: unidad de interfaz con el bus

La BIU ("Bus Interface Unit") es el bloque encargado de soportar todas las transferencias con el mundo exterior. Controla los ciclos del bus que acceden a la memoria y a las E/S. En la siguiente figura 6.9, podemos ver las distintas partes en las que se divide la unidad de interfaz con el bus.

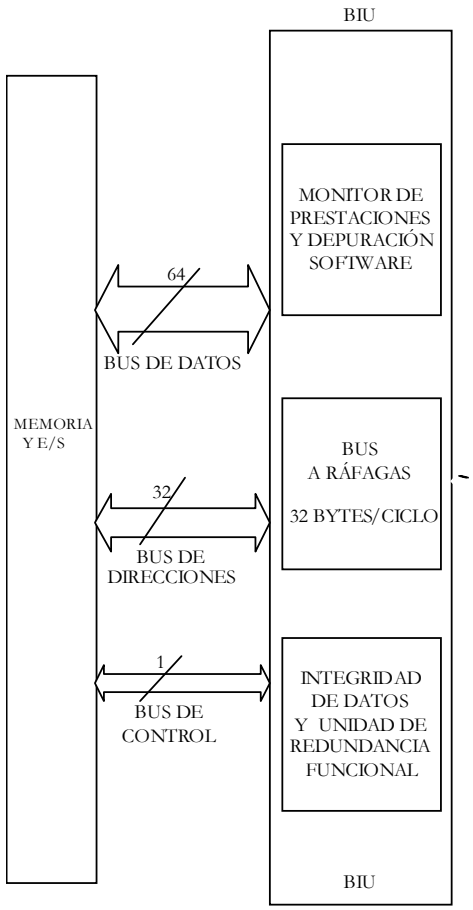


Figura 6.9 – Conexionado de la BIU

El Pentium tiene el mismo rango de direccionamiento que los 80386 y los 80486. Esto es debido a que también usa un bus de direcciones de 32 bits. Sin embargo, el bus de datos externo es de 64 bits, el doble que sus predecesores. El subsistema de memoria debe por tanto, estar organizado en ocho grupos de 8 bits cada uno, es decir, un total de 64 bits para adecuarse al bus del procesador.

El bus externo de 64 bits, es capaz de transferir datos entre la memoria y el procesador a velocidades que pueden llegar hasta 584 MBytes/sg.

La BIU consta de tres partes:

- Monitor de prestaciones
- Bus a ráfagas
- Unidad de redundancia funcional

A continuación se explican cada una de ellas.

6.2.5.1. Monitor de prestaciones

Desarrollar aplicaciones es cada vez más complejo y precisa de una cuidadosa realización para evitar que la mayor parte del tiempo se pierdan ciertas rutinas o selecciones del código que no son excesivamente importantes. Para facilitar el trabajo de los desarrolladores de software, el procesador Pentium incorpora un monitor de prestaciones y una unidad de depuración software.

El procesador posee un conjunto de contadores y unidades de rastreo y traza, que exploran y archivan todos los acontecimientos significativos del flujo de control. Es una herramienta hardware que la puede usar el programador para saber cómo funciona el programa. Se creó para facilitar el trabajo de los desarrolladores de software.

Estos contadores junto a las unidades de rastreo y traza, permiten conocer el estado del procesador, el tiempo que se emplea en la realización de operaciones y las instrucciones que se ejecutan. Desde el exterior del procesador, por medio del puerto serie, se puede interactuar con esta unidad.

A parte de todo esto, se puede obtener el número de ciclos que el procesador emplea en operaciones internas que de algún modo afecten a:

- La lectura y escritura de datos.
- La ausencia o presencia de datos o código en las memorias caché internas del Pentium.
- Las interrupciones.
- La utilización del bus.

También se puede saber cuanto tiempo tiene que esperar el procesador para conseguir el control del bus externo. Gracias a todo lo anterior, se consigue sistema más rápido debido a la fácil optimización del mismo.

La unidad de traza cuando se produce cierta condición de bifurcación o los saltos a subrutinas o si la ejecución se produce en determinada sección de código o que instrucción ha provocado una interrupción, etc. Por tanto, se pueden detectar los cuellos de botella, es decir, dónde el sistema se ralentiza o la aplicación pierde mucho tiempo inútilmente. Por ello, se optimiza consiguiendo mejores prestaciones y mejor tiempo de respuesta.

6.2.5.2. Bus a ráfagas

Es una circuitería de silicio que permite cargar 256 bits (32 Bytes que es igual a la línea de caché) en la caché de datos de una sola vez, es decir en un ciclo.

En cuanto al tipo de ciclos del bus, los valores medios corresponden a:

- 36%: Prebúsqueda de instrucciones
- 21%: Lecturas de datos
- 36%: Escritura de datos
- 7%: Escrituras obligadas de datos (L1)

Como se acaba de comentar, el tipo de ciclo de bus de ráfaga puede cargar 256 bits de una sola vez. El bus externo de 64 bits es capaz de transferir datos entre la memoria y el procesador a una velocidad que puede llegar a ser de 528 MBytes por segundo. Esto significa que, por ejemplo, el contenido completo de un disco fijo de 100 MBytes pasaría por este bus en menos de un quinto de segundo. Esta velocidad de transferencia es superior en más de tres veces al ancho de banda del bus de un 80486 a 50 MHz.

6.2.5.3. Unidad de redundancia funcional

Es un recurso que emplea diferentes técnicas para la detección de errores tanto externa como internamente, para asegurar la integridad de los datos. Cada octeto del bus de datos, lleva asociado un bit de paridad lo que hace un total de 8 bits de paridad para todo el bus de datos. Los bits de paridad son comprobados por el procesador en cada lectura. A su vez el Pentium genera un bit de paridad por cada octeto de los 64 bits que componen cada escritura hacia el exterior, lo que hace un total de 8 bits de paridad. También el bus de direcciones añade un bit de paridad por octeto; de este modo hay 4 bits de paridad para las direcciones que se generan y comprueban en cada acceso de escritura o lectura respectivamente. Con todo ello, el procesador Pentium es capaz de detectar errores en el bus de direcciones y en el de datos. Por tanto, el Pentium, no sólo detecta que el dato leído o escrito es correcto sino que también es capaz de saber si la dirección de memoria es correcta.

Internamente, también se hacen controles de paridad en la caché interna, en los registros internos y en la memoria ROM que almacena el microcódigo. Hay otro tipo de recursos que aseguran la fiabilidad del procesador. Siempre después de una inicialización se realiza un autodiagnóstico interno que comprueba que al menos un 70% de los dispositivos internos funcionan adecuadamente.

El Pentium implementa un sistema de redundancia funcional de una forma muy sencilla. Basta con poner dos procesadores Pentium en el mismo bus, uno trabajando en modo maestro y otro como comprobador. Los dos procesadores ejecutan las mismas instrucciones en el mismo tiempo. El que hace de comprobador chequea cada resultado obtenido por el maestro con el suyo propio. Si existe diferencia se produce una interrupción de máxima prioridad que detiene el sistema y avisa que los dos procesadores no están de acuerdo en los resultados de la ejecución del programa.

6.3. MODOS DE FUNCIONAMIENTO DEL PENTIUM

Las tres formas de trabajo que tiene el Pentium son el modo real, el modo protegido y el modo de manejo del sistema. En cada una el Pentium funciona de manera diferente. Para seguir correctamente el funcionamiento se aconseja consultar la figura 6.10

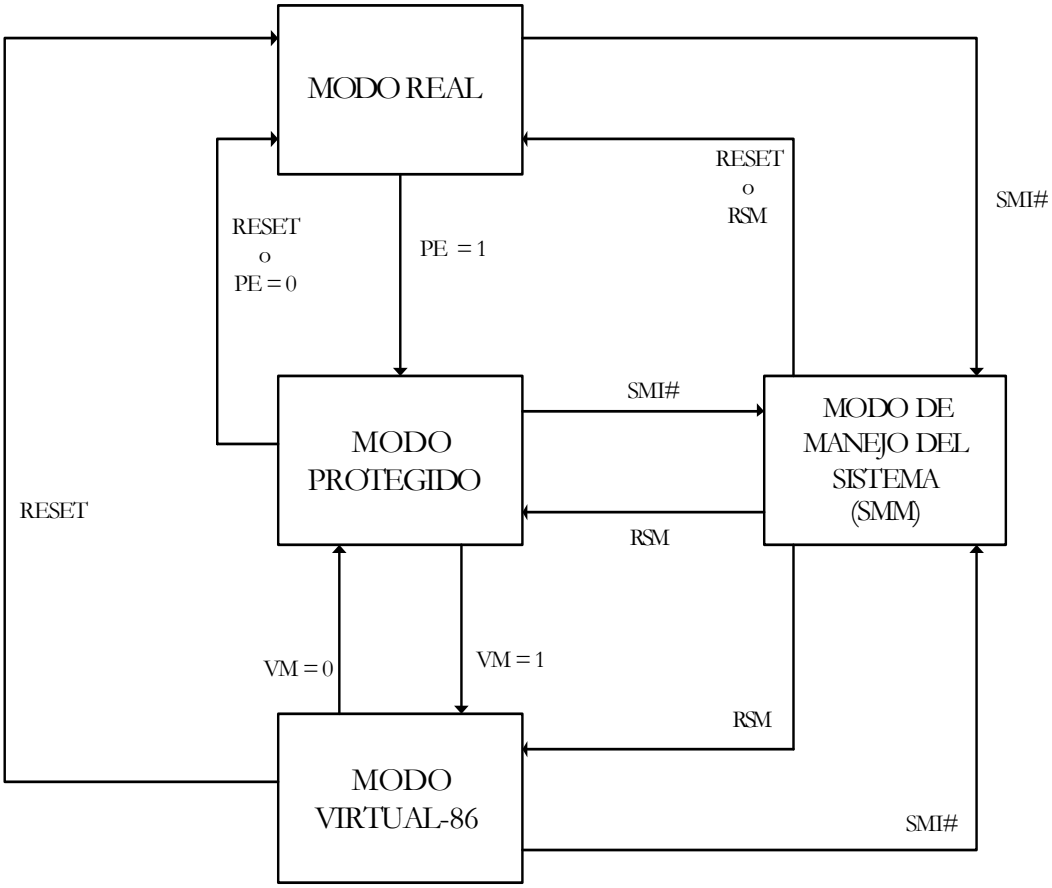


Figura 6.10 – Modo de funcionamiento del Pentium

- Modo real:

Funciona como si fuese un 8086 aunque posee algunas extensiones, por ejemplo la frecuencia es mayor (es la propia de trabajo del Pentium). En este modo entra cuando se produce un Reset (se usa la segmentación y funciona con 16 bits). En este modo el Pentium solo trabaja con el primer MByte de la memoria, siendo 64 KBytes la capacidad máxima que puede ocupar un segmento. Por último citar que el bus de direcciones utiliza los 20 bits de menos peso. Este modo es monotarea y los registros de propósito general (AX, BX, CX, DX, SI, DI, SP, BP), IP (Puntero de Instrucciones) son de 16 bits; lo mismo que el registro de flags. La tabla IDT es 256 entradas y en cada una de ellas hay un vector de interrupciones que apunta al inicio de la rutina que atiende esa entrada.

En el modo real se empieza a trabajar con un reset o cuando se conecta la alimentación por primera vez, es decir cuando se produce un reset automático.

- Modo protegido:

Este es el modo nativo con toda su potencia. Los registros ya tienen una extensión de 32 bits. Es un modo opcional. Puede trabajar con memoria virtual y con una memoria física de hasta 32 GBytes, pudiéndose dar la segmentación y la paginación. Se trabaja ya

con multitarea y por ello funciona el sistema de protección y aislamiento entre tareas, pudiendo alcanzar los segmentos un tamaño de 32 GBytes.

Estando en modo real para pasar a modo protegido hay que poner el bit PE a 1 ("Protection Enabled") que se encuentra en el CR0 (doble palabra de estado de la máquina) del registro de control.

Al pasar al modo protegido hay que tener en cuenta que el flag IF puede estar a 1 , también hay que crear una nueva IDT ya que ésta cambia en contenido y en capacidad respecto a la del modo real. El sistema operativo debe crear la IDT justo al empezar el modo protegido.

Además para funcionar, se necesitan las tablas de descriptores, GDT y LDT, para localizar un segmento. También hay que actualizar los registros de segmento.

Como resumen de lo comentado hasta ahora, los pasos que se deben seguir para pasar a este modo son los siguientes:

1. Poner el bit IF a 0.

2. Instalar en la memoria la tabla de descriptores global (GDT), mediante el registro GDTR que apunta a la base y al límite de la GDT.

3. Poner el bit PE a 1. Esto se realiza cargando el registro EAX con 00....01 (32 bits) y moviendo el contenido de este a CR0.

4. Hacer una instrucción JUMP para eliminar así las instrucciones que se habían cargado en la cola de prebúsqueda. En el modo real hay un elemento que es la cola de prebúsqueda que va guardando las instrucciones que se van a ejecutar a continuación. Así al pasar a modo protegido hay que limpiar esta cola ya que contiene instrucciones de modo real.

5. Cargar la LDT y el registro LDTR pasa a contener las características de la tabla en curso.

6. Cargar el registro de tarea (TR).

7. Actualizar los registros de segmentos, el CS, SS.....GS para que apunten a los segmentos de modo protegido.

8. Poner en marcha la IDT que una vez cargada se cargará en el registro LDTR, la base y el límite de la misma.

9. Permitir las interrupciones mascarables.

- Modo Virtual 8086

Es una mezcla de modo real y modo protegido. Trabajamos en un ambiente igual que en modo protegido, en un ambiente multitarea y con sistema de protección entre las tareas, pero se permiten ejecutar tareas del 8086 (del modo real). Así algunas tareas son por tanto del modo real y como tienen unicamente 20 bits, las convierte a 32 bits para que así puedan apuntar a más del primer MByte de memoria.

Para pasar del modo protegido al modo virtual, basta con poner el bit VM a 1 del registro de estado, en caso contrario habrá que poner el bit VM a 0.

- Modo de manejo del sistema (también conocido como SMM)

  En este modo el Pentium proporciona un sistema operativo que es transparente para el programador y que implementa dos funciones muy importantes: la primera función está relacionada con la seguridad de todo el sistema y mejora dicha seguridad; la otra función es un sistema de control de la alimentación que controla el consumo del procesador del sistema y lo mejora.

  El procesador aisla completamente, cuando se trabaja en este modo, un espacio de memoria reservado para él donde se salva todo el contenido de la tarea que se va a ejecutar. Para pasar de cualquiera de los otros tres modos a SSM hay que activar por hardware una patita del Pentium. Se trata de la patita SMI# que se activa por nivel bajo. Hay otra forma que no es por hardware y que consiste en provocar una interrupción desde el controlador programable de interrupciones SMI. Dicha interrupción, la atiende una entrada específica de la IDT.

  Para pasar a modo real desde este modo, basta con que se produzca un reset. Además existe una instrucción del repertorio de instrucciones del Pentium, la RSM mediante la cual se puede pasar al modo real, al modo protegido o al modo virtual 86. Para saber a cual de los tres modos se ha pasado, se consultarán los bits PE y VM. Si el bit PE es 0, haremos pasado al modo real y no es necesario consultar el VM. Si el bit PE y VM están a 1, habremos pasado al modo virtual 86. Por último, habremos pasado a modo protegido si el bit PE está a 1 y el VM a 0.

MODELO DEL PENTIUM PARA EL PROGRAMADOR DE APLICACIONES

7.1. PROGRAMADOR DE SISTEMAS Y PROGRAMADOR DE APLICACIONES

En los procesadores avanzados es muy importante saber apreciar la diferencia existente entre los dos tipos de programadores habituales que participan en la construcción del software.

7.1.1. Programador de aplicaciones

Se encarga de crear el sistema lógico que soportan las aplicaciones del usuario. Ve la CPU como un conjunto de registros de trabajo que le permiten confeccionar dichas aplicaciones del usuario. Con estos registros puede manipular instrucciones, datos y direcciones además de otros elementos que le permitirán desarrollar sus programas.

Para este programador resultan transparentes los recursos de la CPU empleados para llevar a cabo la tarea de aplicación junto con otras distintas de acuerdo con un mecanismo de protección que controla los accesos entre las tareas entre sí, entro los objetos de la tarea y entre los accesos de las tareas y el sistema operativo incluso en la ejecución de determinadas instrucciones.

Los conocimientos que el programador de aplicaciones debe tener sobre la máquina son los imprescindibles para obtener el máximo rendimiento de las instrucciones usadas para resolver las aplicaciones. En el caso de emplear el lenguaje máquina, deberá conocer los registros internos accesibles para la manipulación de datos y direcciones, el repertorio básico de instrucciones y los modos de direccionamiento. También deberá manejar el modelo de programación del coprocesador matemático auxiliar.

7.1.2. Programador de sistemas

Es necesario que conozca profundamente la arquitectura detallada de la CPU para así optimizar todos los recursos, obteniendo en su funcionamiento la máxima potencia, seguridad y rendimiento. Debe conocer también las prestaciones de la memoria virtual, las características de protección del entorno, los mecanismos que posibilitan la conmutación de tareas, el tratamiento de interrupciones y excepciones, etc.

La misión de este programador es construir un sistema de explotación óptimo que sea capaz de soportar todas las aplicaciones previstas. Entre sus funciones más destacadas están:

- Organizar el sistema para el correcto tratamiento de las tareas pertenecientes a los diferentes usuarios.
- Confección de objetos para sistemas operativos, depuradores, compiladores ...
- Asignar a cada tarea su nivel de privilegio y un sistema de protección adecuado.
- Organizar toda la memoria y el procesador para que de este forma las tareas consigan un mejor rendimiento.

Los microprocesadores Pentium disponen de una serie de registros y recursos especiales, denominados “del sistema”, que se encargan de gestionar el funcionamiento general, aprovechando todas las prestaciones.

La complejidad de estos microprocesadores ha hecho necesaria la construcción de herramientas para el desarrollo de software. Estas herramientas varían dependiendo del tipo de progre

mador. Así, Intel inicialmente diseñó una herramienta lógica, denominada BINDER, encargada de la generación de aplicaciones, sencilla de manejo, pero que no permite el acceso a los mecanismos del sistema, haciéndola idónea para el programador de aplicaciones. El BUILDER por el contrario, es otra poderosa herramienta destinada a la construcción de sistemas, que posibilita el control de todos los recursos de la CPU, por lo que requiere ser usada por los programadores de sistemas, capaces de describir con detalle la estructura global que se precisa implantar.

7.2. REGISTROS INTERNOS PARA EL PROGRAMADOR DE APLICACIONES

El Pentium dispone de 32 registros en su arquitectura interna de los cuales la mitad, es decir 16, son para uso del programador de aplicaciones, tal y como se pueden apreciar en la figura 7.1

Estos últimos se clasifican en cuatro grandes grupos:

- Registros de propósito general
- Registro Puntero de Instrucciones (EIP)
- Registro de Estado (o de Señalizadores)
- Registros de Segmento

Antes de pasar a explicar estos cuatro grupos, recordaremos muy brevemente las tres formas de trabajo existentes en el Pentium.

- Modo real: trabaja como el 8086 es decir, se usa la segmentación y al iniciar el sistema funciona de esta manera (16 bits).
- Modo protegido: este es el modo nativo con toda su potencia (32 bits). Puede trabajar con memoria real, memoria virtual, sistemas de protección, multitarea, paginación ...
- Modo de manejo del sistema.

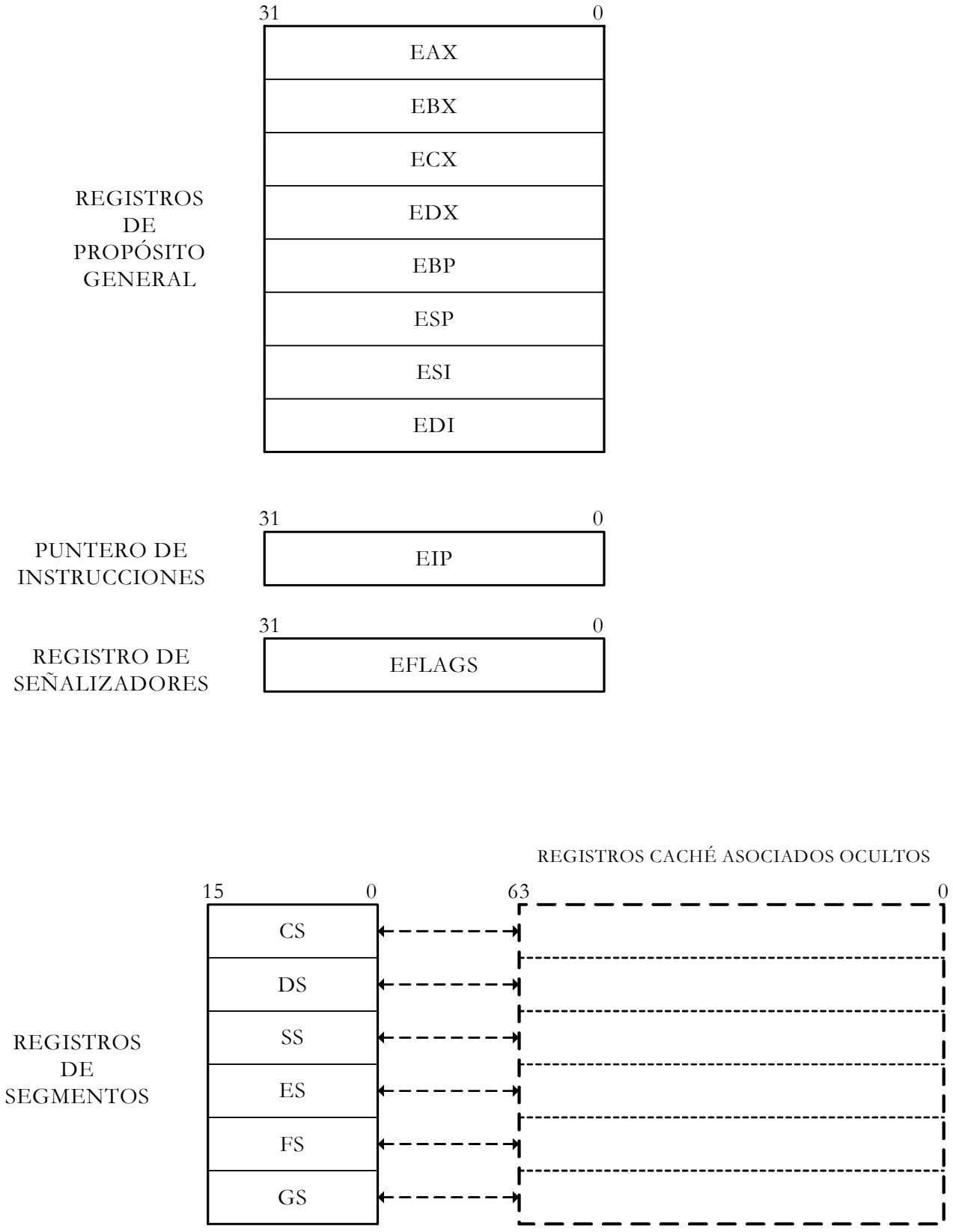


Figura 7.1 – Esquema general de los 16 registros disponibles para el programador de aplicaciones.

7.2.1. Registros de propósito general

Son los mismos registros que tenía el 8086 de 16 bits pero ampliados a 32 bits. Este grupo consta de ocho registros capaces de trabajar con información de 32 bits cuando utilizan todo su tamaño aunque también pueden manejar 16 bits e incluso, 4 de estos registros pueden manejar 8 bits.

Cuando se hace referencia a uno de los registros que trabaja con 32 bits, se pone la letra "E" de extendido precediendo al nombre habitual del registro de 16 bits de los microprocesadores 8086 y 80286. Los nombres de cada uno de estos ocho registros son:

- **EAX**: Acumulador
- **EBX**: Base
- **ECX**: Contador
- **EDX**: Datos
- **ESP**: Puntero de pila
- **EBP**: Puntero de base (Base Pointer)
- **ESI**: Índice fuente (Snack Pointer)
- **EDI**: Índice destino

Cuando se accede únicamente a los 16 bits de menos peso de estos registros, se designan por AX, BX, CX, DX, SP, BP, SI, DI, respectivamente.

También son accesibles, de forma independiente, los dos bytes de menos peso de los registros AX, BX, CX y DX. Dentro de estos 16 bits de menos peso, si accedemos al byte de menos peso se le asigna con AL, BL, CL y DL, respectivamente, mientras que si accedemos al de más peso se le referencia con AH, BH, CH y DH, tal y como se representa en la figura 7.2

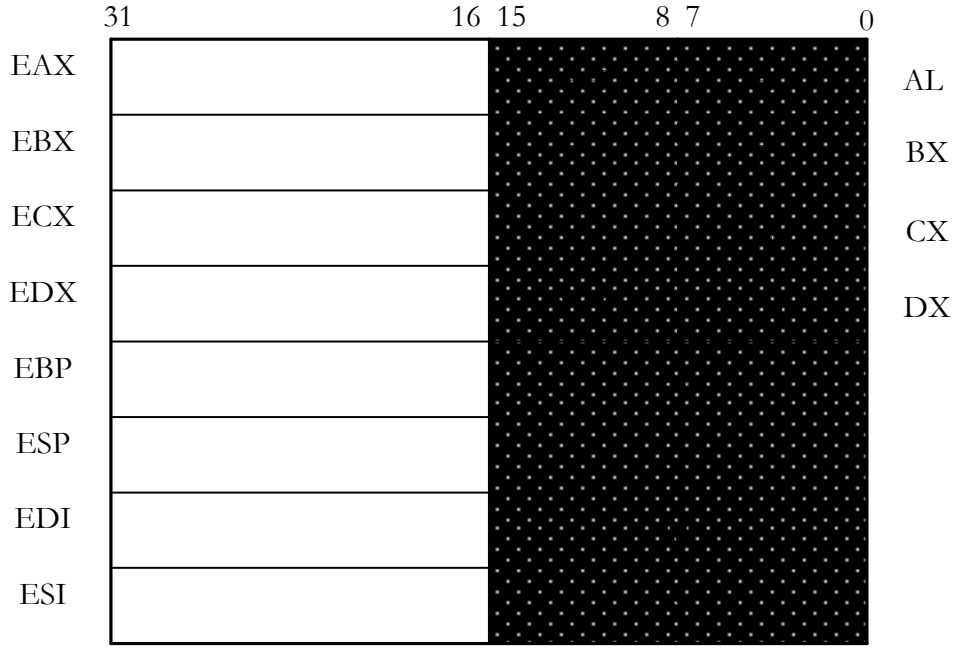


Figura 7.2 – Formatos que pueden adoptar los registros que corresponden al grupo destinado a propósitos generales.

Los registros de propósito general pueden usarse tanto para almacenar datos como instrucciones. En este último caso, el contenido del registro es un desplazamiento que apunta a una dirección de memoria.

Teniendo en cuenta todos los posibles formatos de los registros de este grupo, quedan disponibles ocho registros de 32 bits, otros ocho de 16 bits y otros ocho de tamaño byte. Por tanto, cuando se emplean para contener datos, posibilita al Pentium operar indistintamente sobre bytes, palabras, dobles palabras y cuádruples palabras.

Los registros EAX, EBX, ECX y EDX, se emplean fundamentalmente en operaciones generales como las lógicas, las aritméticas... Es importante mencionar en este punto, que Intel ha procurado mejorar la simetría de los registros en los microprocesadores avanzados, es decir, favorecer la flexibilidad de usar cualquiera de ellos en la mayoría de las instrucciones. En microprocesadores anteriores, bastantes instrucciones requerían el uso específico de ciertos registros para contener operandos o el resultado.

Por lo general el registro EAX, muy utilizado en los procesadores Intel, se emplea como Acumulador en instrucciones lógico-aritméticas y las relacionadas con las transferencias entre la memoria y la CPU.

EJEMPLO 1

MOV AH, AL

El byte de menos peso del Acumulador se mueve al byte AH, que es el de más peso

---

El registro EBX contiene la base de la dirección donde empieza una estructura datos, bien sea un array, una pila...

El registro ECX es habitual utilizarlo como contador en instrucciones que contengan iteraciones, es decir, que se repitan distintas veces, por ejemplo en la operación de notación ROR.

El registro EDX se utiliza como apoyo al EAX. Si por ejemplo el valor del producto no cabe en EAX se agranda con este registro. También para operaciones de E/S, posiciones de E/S que van periféricos o vienen de periféricos.

EJEMPLO 2

```
MOV ECX, 08h.
Start:
   IN AL, 70h.
   MOV [EBX], AL.
   INC EBX.
   LOOP Start.
```

Se carga ECX con 08h, se define el comienzo del bucle, se lee AL vía puerto 70h, se escribe el contenido de AL en el lugar de memoria [EBX], se incrementa en uno la dirección EBX y por último se llevan a cabo 8 repeticiones (hasta que ECX valga 0).

Los registros apuntadores ESP y EBP, sirven para controlar el direccionamiento de la pila y almacenar desplazamientos relativos a la pila en curso.

Las operaciones en la pila las soportan tres registros diferentes que son los que exponemos a continuación:

1. Registro de segmento de pila (SS). Especifica las características del segmento de pila que reside en memoria. El número de pilas en un sistema, está limitado sólo por el máximo número de segmentos. Por tanto, una pila puede tener hasta 4 GBytes de longitud que es el máximo tamaño de un segmento. El registro SS, del que hablaremos en el apartado 7.2.4., lo usa el procesador para todas las operaciones de la pila.

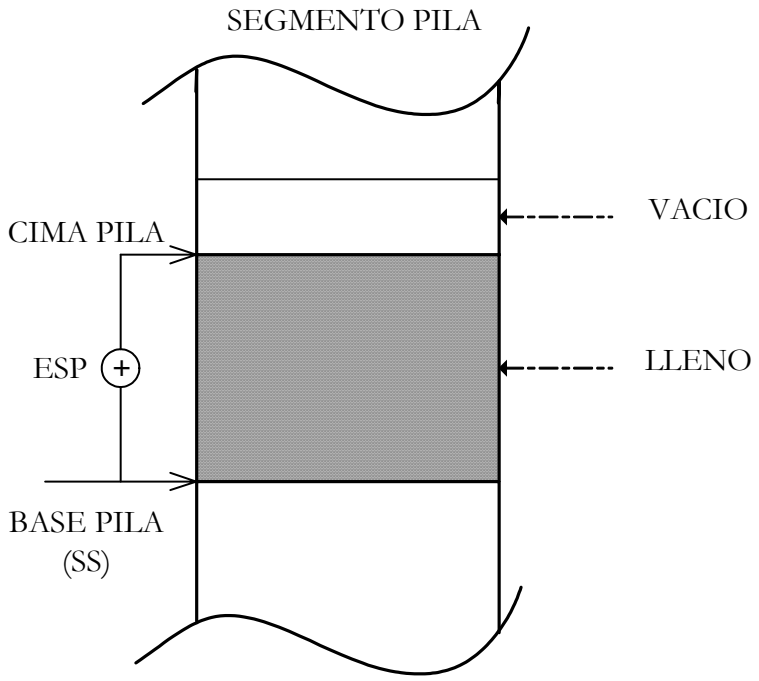


Figura 7.3 – Figura del funcionamiento de la pila SS

2. Registro puntero de pila (ESP). Contiene el desplazamiento de la cima de la pila en el segmento de la pila actual. Lo usan las operaciones PUSH y POP, las llamadas a subrutinas, el retorno, las excepciones y las interrupciones. Cuando se introduce un elemento a la pila, el procesador decrementa el puntero ESP, y escribe el elemento en la cima de la pila. Cuando se saca un elemento se hace la operación contraria, es decir, incrementar ESP.

3. Registro puntero base de la pila (EBP). Se usa normalmente par a acceder a estructuras de datos pasadas en la pila. Cuando el registro EBP se usa para direccional memoria, el segmento de pila en curso es referenciado. Este registro apunta a la base de la pila y cuando existen rutinas hace el papel de ESP para así no modificar el valor de este último.

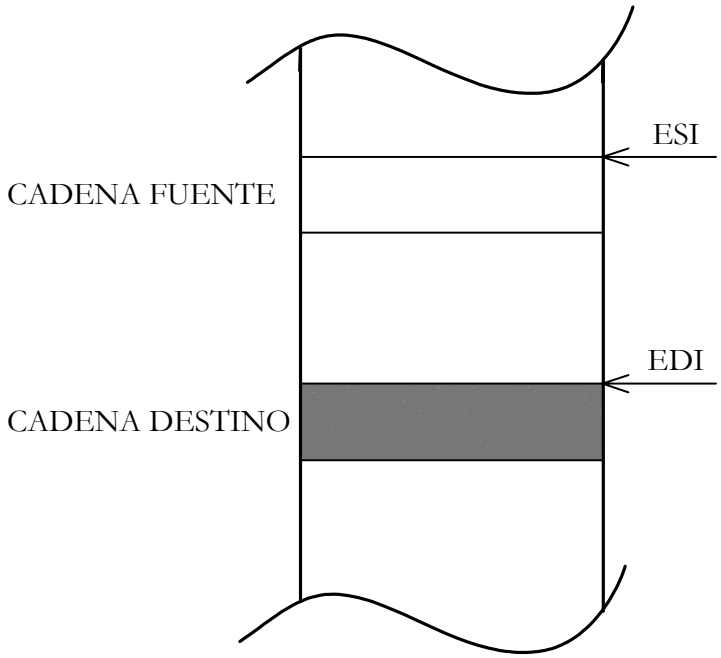


Figura 7.4 – Figura del funcionamiento de los registros ESI y EDI

Por último, los registros ESI y EDI, contienen valores índice usados en la exploración de grandes conjuntos de datos, cadenas, arrays, etc...y admiten la posibilidad de incremento y decremento automático de su valor para relaciones fuente y destino, como se puede apreciar en la figura 7.4.

7.2.2. EIP: Registro puntero de instrucciones

Es un registro de 32 bits que almacena el desplazamiento que hay que añadir a la base del segmento de código para obtener la dirección donde está la siguiente instrucción, es decir, la dirección que el procesador tiene que ejecutar a continuación. Por tanto, el registro EIP contiene el valor de dicho desplazamiento, mientras que la base se obtiene a partir del contenido del registro de segmento de código CS.

El EIP no está directamente disponible para el programador , lo gobierna implícitamente el control de transferencias de las instrucciones, las interrupciones y las excepciones.

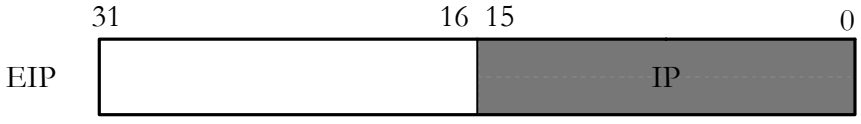


Figura 7.5 – El Puntero de Instrucciones admite el tamaño de 32 bits (EIP) y el de 16 bits (IP), cuando soporta el direccionamiento "reducido".

Este registro puede trabajar en dos modos:

- En modo nativo, que se acaba de describir, el cual recibe el nombre de EIP y posee 32 bits.
- En modo real, cuando se emplea este tipo de direccionamiento reducido, compatible con los procesadores 8086 y 80286, sólo se precisan de 16 bits para especificar el desplazamiento, que son los dos bytes de menos peso de EIP, y que se denominan IP. (figura 7.5)

En la memoria segmentada la dirección de la instrucción en curso se halla sumando el desplazamiento a la base donde comienza el segmento del código, como podemos ver en la figura 7.6

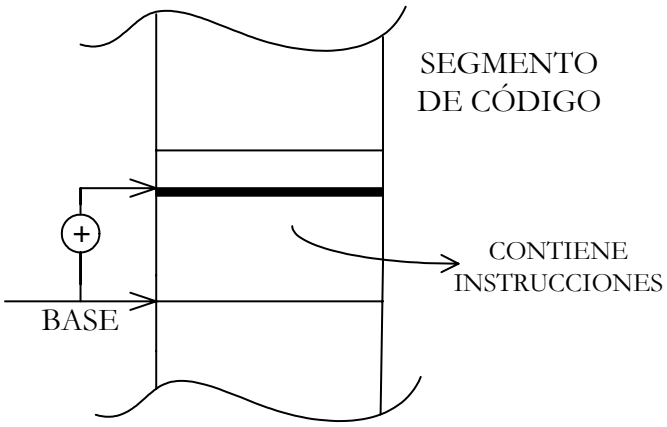


Figura 7.6 – Funcionamiento de la memoria segmentada

7.2.3. Registro de estado o señalizadores

Este registro también conocido como el registro EFLAGS consta de 32 bits de los cuales la mayoría son señalizadotes de estado, controlados por la ALU (acarreo, paridad, acarreo auxiliar, cero, signo y sobrepasamiento), actuando los restantes como señalizadotes del sistema, ligados al mecanismo de protección y a otros recursos de que dispone el sistema de explotación cuya misión será mejor interpretada a medida que se profundice en el estudio del procesador (figura 7.7)

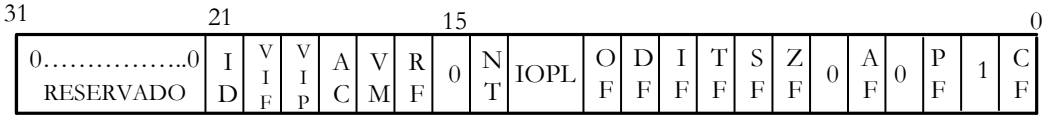


Figura 7.7 – Registro EFLAGS

A continuación, pasaremos a explicar cada uno de los bits que forman parte del registro de estado siguiendo el orden de menos peso a más peso:

- **CF**: Señalizador de acarreo en el bit más peso (en el más significativo). Si trabajamos en modo real es en el bit 16 y si trabajamos en modo protegido es en el bit 32.

    - 1: Si hay acarreo en las operaciones de suma aritmética.
    - 0: En las restas cuando hay llevada en el bit de más peso. También se utiliza este bit en operaciones de desplazamiento y rotación.

- **PF**: Bit de paridad impar.

    - 1: Para generar la paridad impar con los bits que conforman el resultado de una operación
    - 0: En caso de no producirse paridad impar.

- **AF**: Señalizador de acarreo auxiliar (o intermedio).

    - 1: Si ha habido acarreo en el bit 3 del resultado (contando sería el 4º bit). Se utiliza en las operaciones BCD.
    - 0: En caso de no haber acarreo auxiliar.

- **ZF:** Señalizador de cero.

    - 1: Si todos los bits del resultados son cero. Es muy útil al comparar dos registros, ya que si tienen el mismo contenido ZF pasa a valer 1
    - 0: En caso de que el resultado no sea 0

- **SF**: Señalizador de signo.

    - 1: Si el bit de más peso del resultado de la operación es 1.
    - 0: Si el bit de mas peso del resultado de la operación en 0.

- **TF**: Excepción al terminar la ejecución de la instrucción.

    - 1: Provoca una excepción al completarse la ejecución de la instrucción en curso. Posibilita la depuración de los programas al permitirse la ejecución paso a paso, es decir instrucción por instrucción.
    - 0: No hay excepción

    Si un programa de aplicación dispone de un flag TF utilizando una instrucción POPF, POPFD, o IRET, se genera una "limpieza de excepción" después de la instrucción que sigue a la de estas.

- **IF**: Flag de habilitación de interrupciones.

    - 1: Permite el reconocimiento de las peticiones de interrupción mascarables provocadas por la activación de la patita INTR del Pentium. Las no mascarables se atienden siempre ya que son de máxima prioridad.
    - 0: Prohíbe el reconocimiento de la interrupción externa y no la atiende.

    La señal IF no afecta a la generación de excepciones o interrupciones no mascarables como por ejemplo la interrupción NMI. El CPL, IOPL y el estado del flag VME del registro de control CR4 determina si el flag IF puede ser modificado por las instrucciones CLI, STI, POPF, POPFD e IRET.

- **DF**: Flag de dirección de exploración de las cadenas de caracteres o strings.

    - 1: Postdecremento automático de los registros ESI, EDI (registros de propósito general), que direccional la cadena.
    - 0: Postincremento automático en ESI, EDI.

- **OF**: Flag de sobrepasamiento (u overflow)

    - 1: En operaciones con números enteros con signo se activa si el resultado es muy grande (es positivo) o muy pequeño (si es negativo). Indica resultados erróneos por ejemplo si existe acarreo en el penúltimo bit.
    - 0: Si no existe sobrepasamiento

- **IOPL**: Nivel de privilegio de las entradas y salidas (input/output). Es un campo de 2 bits que se emplean en modo protegido y determina:

    1. El nivel de privilegio a partir del cual se pueden ejecutar las instrucciones de E/S sin generar error. Las instrucciones de E/S son las siguientes:
        - IN, entrada
        - OUT, salida
        - INS, string como entrada
        - OUTS, string como salida
        - CLI, pone a 0 el bit IF
        - STI, pone a 1 el bit IF

        Únicamente las instrucciones input y output pueden usarse para acceder a esta memoria.

    2. El nivel de privilegio de privilegio que permite la alteración del señalizador IF al cargar el registro EFLAGS.

    Los valores que se pueden tomar son:
    - 11: nivel 3 (pueden acceder todos)
    - 10: nivel 2
    - 01: nivel 1
    - 00: nivel 0 (únicamente puede acceder el S0)

    POPF e IRET pueden modificar el campo IOPL solamente cuando se ejecutan con el máximo nivel de privilegio, o sea, el 0. Una conmutación de tarea puede alterar siempre el IOPL al cargarse el nuevo EFLAGS desde el segmento de estado de la nueva tarea como se explicará posteriormente.

    El IOPL es igualmente uno de los mecanismos que controla la modificación del flag IF y el manejo de interrupción en la función virtual 8086 cuando las extensiones de la función virtual están vigentes (el flag VME en registro de control CR4 está dispuesto).

- **NT**: Tarea anidada. Este señalizador se activa y desactiva automáticamente al producirse una conmutación de una tarea.

    - 1: La tarea en curso está anidada con la anterior, es decir, luego hay que volver a ésta.
    - 0: La conmutación de tareas es libre

El flag puede ser explícitamente activado o desactivado mediante las instrucciones POPF/POPFD. Sin embargo, la alteración del estado de este flag puede originar inesperadas excepciones en programas de aplicación.

- **RF**: Flag de reanudación. Su activación provoca la ejecución de la siguiente instrucción cuando se produce un fallo de depuración en una instrucción, es decir, se ignora el fallo producido en la depuración.

    - 1: Se ignoran los puntos de parada
    - 0: No se ignoran los puntos de parada

- **VM**: Modo virtual – 86. Trabajando en modo protegido, se permite que algunas tareas trabajen en Modo Real.

    - 1: Estando el procesador en Modo Protegido, se pasa a Modo Virtual 86
    - 0: No se hace nada

    Este bit puede activar por medio de la instrucción IRET, cuando se está en Modo Protegido si el nivel de privilegio en curso CPL = 0. También por medio de una conmutación de tarea en cualquier nivel de privilegio.

    El señalizador VM no queda afectado por la instrucción POPF. Sin embargo, cuando se ejecuta PUSHF, se pone a 0 este bit, incluso cuando se está trabajando en Modo Virtual-86.

- **AC**: Bit de chequeo de alineamiento.

    - 1: Se produce una excepción para alinear una palabra
    - 0: No hay excepción

    Las direcciones de palabra de palabras de 2 bytes deben ser múltiplos de 2, las de 4 bytes deben serlo de 4 y las de 8 bytes, múltiplos de 8. Si un programa de aplicación con el mínimo nivel de privilegio, es decir, nivel de privilegio 3, tiene una palabra desalineada y este flag AC = 1, se produce una excepción en concreto la excepción número 17. Las referencias de memoria que por defecto tienen un nivel de privilegio 0 no generan esta excepción, incluso cuando son ejecutadas en modo usuario.

    Si el programador no pone los bits de datos alineados, el bus trabajará más, es decir, necesitará más ciclos para leer lo mismo que si estuviesen alineados. Como hemos dicho anteriormente, si AC = 1 y se produce un fallo de alineamiento, se genera una excepción. Por el contrario, si AC = 0, el acceso se realiza en más ciclos, es decir, no se produce excepción alguna.

    Al ponerse a 1 los flags AC y AM en el registro de control CR0 se habilita el chequeo de alineamiento de las referencias de memoria.

    La excepción por chequeo de alineamiento puede ser empleada para chequear el alineamiento de datos. Esto es útil cuando se intercambian datos con otros procesadores, los cuales requieren todos los datos para ser alineados.

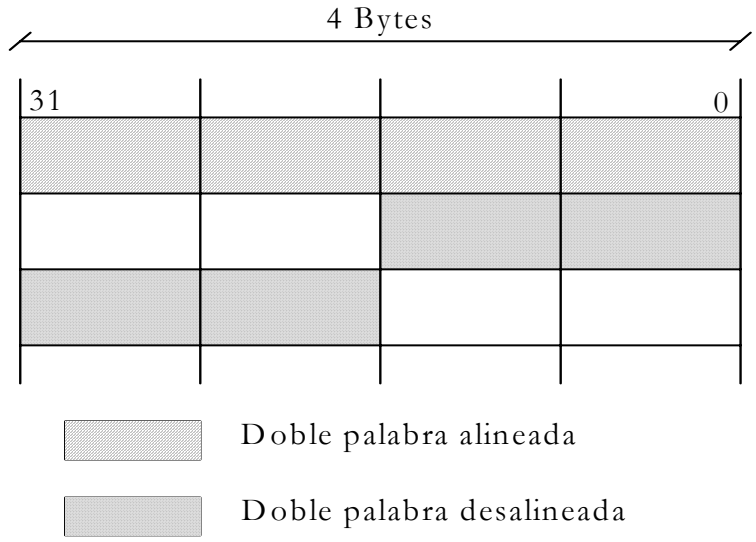


Figura 7.8 – Alineamiento de la memoria

En la figura 7.8, podemos observar una doble palabra alineada y una doble palabra desalineada. En caso de que la doble palabra estuviera alineada, se cogen los 4 Bytes y se colocan en el bus de datos consiguiendo así una lectura rápida. En el caso opuesto, es decir, si la doble palabra se encuentra desalineada, habrá que leer en dos accesos por lo que se produce una pérdida considerable en el rendimiento.

Por último, citar que Intel permite que existan las palabras desalineadas pero Motorola lo prohíbe.

- **VIP**: Interrupción (mascarable) virtual pendiente. Trabaja en combinación con el flag VIF para que cada tarea en modo virtual disponga se su flag IF. Así se aceleran notablemente las interrupciones y las instrucciones CLI y STI no provocan excepción. Se activa por software para indicar que una interrupción está pendiente. El procesador lee este flag pero nunca lo modifica. El procesador sólo reconoce el flag VIP cuando el flag VME o el PVI en el registro de control CR4 están activados y el IOPL es menor que 3.

    - 1: Interrupción pendiente
    - 0: No hay interrupción pendiente

- **VIF**: Interrupción virtual. Es un flag equivalente a IF cuando se trabaja en modo virtual 8086. Este flag se utiliza con el flag VIP. El procesador sólo reconoce el flan VIF cuando bien el flag VME o el PVI en el registro de control CR4 están activados y el IOPL es menor que 3.

- **ID**: Bit de identificación. El flag ID indica si el Pentium soporta la instrucción CPUID que sirve para su identificación. Mediante CPUID se informa sobre las características más importantes del procesador. Le informa al software acerca del modelo de microprocesador en que se está ejecutando. Un valor cargado en EAX antes de ejecutar esta instrucción deberá retornar CPUID. Si EAX = 0, se cargará en dicho registro el máximo valor de EAX que se podrá utilizar en CPUID (para el Pentium este valor es 1). Además, en la salida aparece la cadena de identificación del fabricante contenido en EBX, ECX y EDX. EBX contiene los primeros cuatro caracteres , EDX los siguientes

cuatro, y ECX los últimos cuatro. Para los procesadores Intel la cadena es "GenuineIntel".

- 1: El Pentium soporta la instrucción CPUID que sirve para la identificación de la CPU (número de serie, frecuencia con la que se trabaja ...)
- En caso contrario.

7.2.4. Registro de segmento

Intel ha incorporado en la arquitectura IA-32 la segmentación como sistema principal en la organización de la memoria. La segmentación favorece la programación estructurada y la modularidad, siendo una técnica apropiada para permitir la reubicación y soportar una estructura de protección muy segura y flexible.

Los segmentos son zonas de la memoria de tamaño variable que contienen el mismo tipo de información. Así, hay tres tipos de segmentos principalmente:

- De pila
- De código
- De datos

El Pentium controla en cada instante 6 segmentos a los que direcciona a través de los Registros de Segmento (RS). Esto no significa que sólo pueda manejar 6 segmentos, sino que, si desea acceder a otro que no está referenciado por dichos registros, deberá cargarse previamente, en uno de ellos, el valor correspondiente al nuevo segmento.

Para direccionar la ubicación de los segmentos, el Pentium dispone de 6 registros de 16 bits que se muestra a continuación en la figura 7.9.

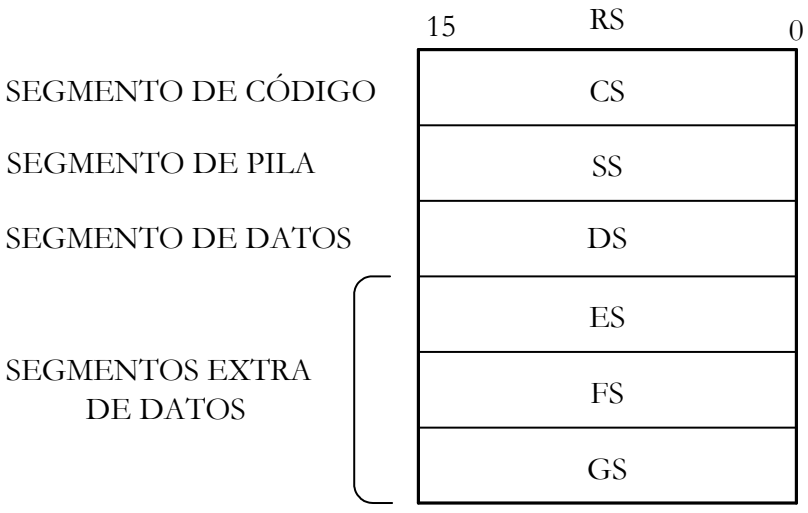


Figura 7.9– Registro de segmento en el Pentium

Desde el punto de vista del programador de aplicaciones, los 6 registros de segmento materializan, en cada momento, los segmentos que es capaz de identificar y manipular la CPU.

La dirección lógica o virtual de todo elemento accesible en la memoria, está formada por un puntero que consta de los siguientes campos:

- **Selector**: Es un valor de 14 bits, contenido en cada uno de los 6 registros de segmento que identifica al segmento y en concreto la dirección de su base. Los dos bits de menos peso de los registros de segmento no intervienen en el selector ya que contienen el campo RPL que indica el nivel de privilegio del peticionario, como se puede apreciar en la figura 7.10. El bit de menos peso de los 14 bits del campo selector es el TI (Índice de Tabla), que indica que la Tabla de Descriptor es local (TI=1) o global (TI=0), ayudando a localizar el segmento determinado en la memoria.

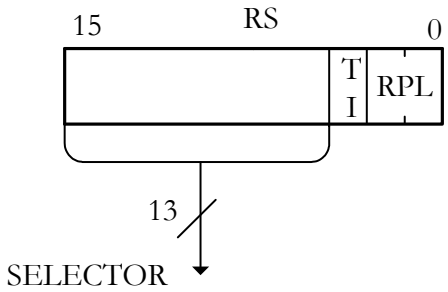


Figura 7.10 – Contenido del registro de segmento

- **Desplazamiento**: Es un valor que se añade a la base del segmento para localizar la dirección que hay que acceder en él. El tamaño de desplazamiento determina su longitud máxima que, en el caso del Pentium, es de 2^{32} = 4 GBytes, en el modo protegido, mientras que en el modo real son 2^{16} = 64 KBytes.

Cada vez que el procesador ejecuta un programa, utiliza estos dos campos. En concreto emplea el campo selector para calcular la base del segmento, es decir , la dirección donde comienza y el campo desplazamiento es el valor que suma a la base para determinar la posición a acceder tal y como se muestra en la figura 7.11.

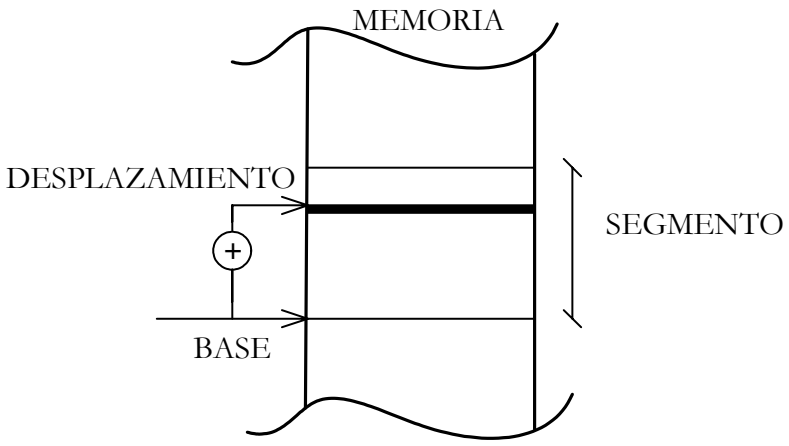


Figura 7.11 – Dirección lógica de un elemento (desplazamiento)

A continuación, después de ver el funcionamiento general para todos ellos, explicamos cada uno por separado:

- **CS**: El registro CS (Segmento de Código), contiene en cada momento el segmento de código que está ejecutando la CPU, es decir el segmento en curso. El desplazamiento que hay que añadir a la base del CS reside el Registro Puntero de Instrucciones EIP, explicado detalladamente en el apartado 7.2.2.

- **SS**: El registro SS (Segmento de Pila), guarda el valor del selector del segmento de pila en curso. El registro ESP contiene el desplazamiento que debe añadirse a la base del SS para determinar la cima, donde se cargan y descargan los datos.

- **DS**: El registro DS (Segmentos de Datos), soporta el valor del selector del DS y el desplazamiento viene especificado en el modo de direccionamiento usado en la instrucción para expresar operandos y el resultado.

EJEMPLO 3

MOV EAX, (5555).

Se carga EAX con el contenido de las posiciones de memoria del segmento DS que comienza en la dirección formada por su base más el desplazamiento 5555. Cuando no se explicita en el segmento de Datos se sobreentiende que es DS. Si fuese el segmento de Datos GS, la instrucción sería MOV EAX, GS : 5555.

Sobre este último punto, mencionar que el Pentium dispone de otros tres segmentos de datos activos, además del referenciado por DS, que reciben el nombre de segmentos "extra" y se denominan ES, FS y GS. De esta forma, se puede acceder a cuatro segmentos de datos sin alterar el valor de los registros de segmento, lo que redunda considerablemente en la velocidad de procesamiento de la CPU, como se analizará posteriormente.

En la figura 7.12, se muestra el mapa de memoria en el que se han especificado los seis segmentos activos para la CPU, en un momento determinado.

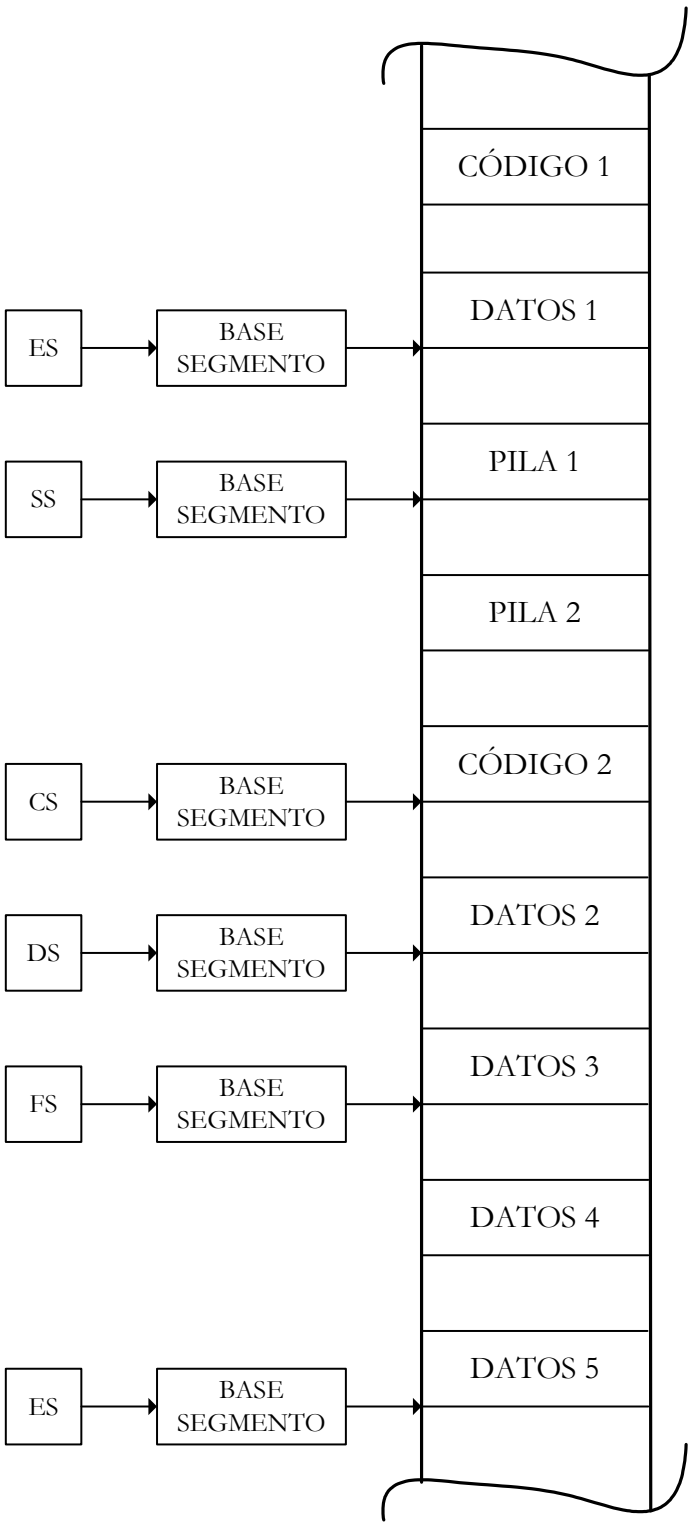


Figura 7.12 – A través de los registros de segmento, la CPU tiene activos a seis segmentos en cada instante. Para utilizar otros segmentos hay que modificar los selectores correspondientes contenidos en los registros de segmento.

7.3. SEGMENTACIÓN EN MODO REAL

Cuando el Pentium funciona en modo real (monotarea), compatible con el 8086, y sin tipo alguno de protección ni posibilidad de manejo de memoria virtual, un segmento queda definido básicamente por los siguientes elementos:

1. **Base** o dirección de comienzo de 20 bits.

2. **Desplazamiento** o tamaño de 16 bits. El tamaño máximo que se admite en este modo para mantener la compatibilidad con el 8086, es de 64 KBytes y la capacidad máxima de la memoria principal sólo es de 1 MByte.

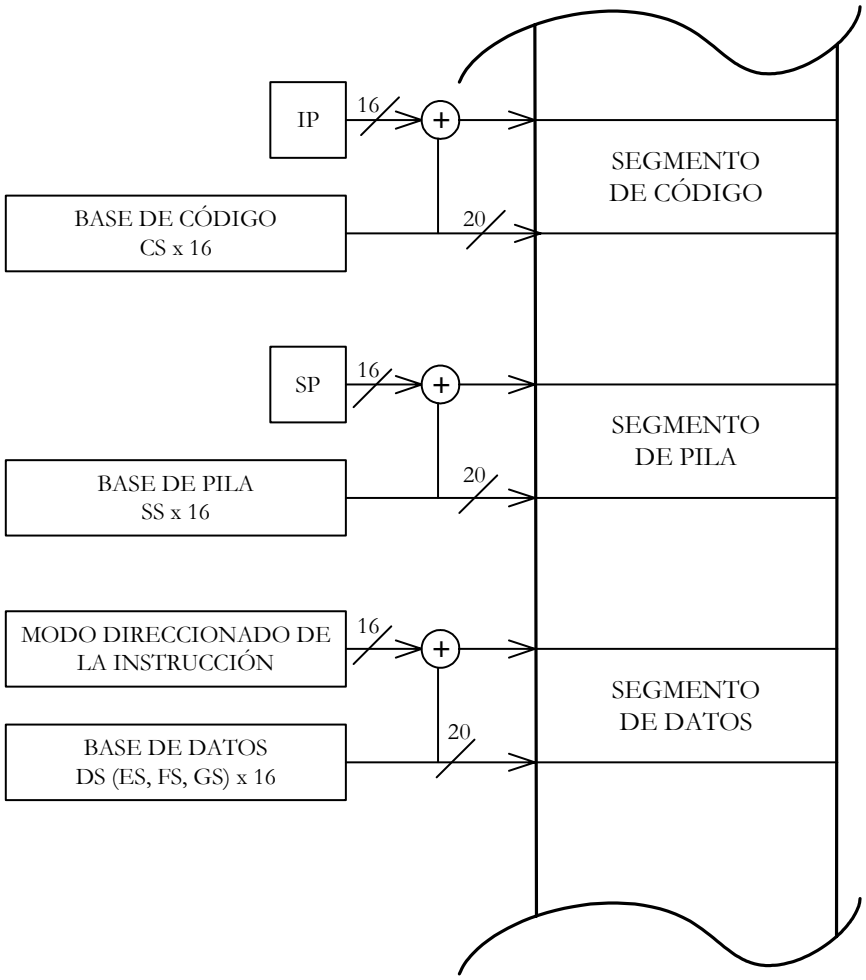


Figura 7.13 – Segmentación en modo real. En este modo se accede a elementos de memoria multiplicando por 16 el valor del registro de segmento y añadiendo un desplazamiento de 16 bits al resultado.

En modo real todo segmento de código, datos o pila está especificado por una dirección lógica, compuesta por dos campos de 16 bits cada uno.

- Selector: Referencia la base del segmento, la cual se deduce a partir del valor contenido en el registro de segmento apropiado. Como el 8086 sólo maneja una memoria de 1 MByte (2^{20}) de capacidad máxima, para obtener la base del segmento se añaden cuatro ceros a los 16 bits del registro de segmento o selector, es decir, se multiplica dicho valor binario por 16.

- Desplazamiento: El tamaño máximo del segmento en el 8086 es de 64 KBytes , por lo tanto bastan 16 bits para expresar el desplazamiento que hay que añadir a la base. En el caso del segmento de código, el desplazamiento lo almacena IP, que está formado por los 16 bits de menos peso de EIP. El desplazamiento correspondiente al segmento de pila está guardado en SP, y el del segmento de datos lo expresa el modo de direccionamiento de los operandos o del resultado en la instrucción en curso, por ejemplo (MOV AX, ES : 555 hex).

Por tanto una dirección quedaría: Dirección efectiva = RS x 16 + Desplazamiento. Esto queda especificado en la siguiente tabla. Dependiendo de si es un segmento de código, de pila o de datos, se conseguirá la base utilizando CS, SS, DS... tal y como se muestra a continuación.

	BASE	**DESPLAZAMIENTO**
CÓDIGO	CS X 16	IP
PILA	SS X 16	SP
DATOS	DS X 16 ES X 16 FS X 16 GS X 16	DESPLAZAMIENTO

Tabla 7.1. – Tabla para el modo real

7.4. SEGMENTACION EN MODO PROTEGIDO

Cuando el Pentium trabaja en modo protegido (multitarea), un segmento queda caracterizado por tres parámetros fundamentales que son comprobados automáticamente por el sistema de protección cada vez que se utiliza. Dichos parámetros son:

1. **Base**, es la dirección lineal donde comienza el segmento. Esta formada por 32 bits, que es la longitud de la dirección de la memoria física que puede alcanzar un tamaño máximo de 2^{32} = 4 GBytes.

2. **Límite**, consta de 20 bits que determinan con exactitud el tamaño del segmento usado por el programador y en el que residen informaciones válidas. Si está expresado en bytes, el lí

mite máximo sería de 2^{20} = 1 MByte y si está expresado en páginas de 4 KByte, un segmento puede ser tan grande como la memoria principal, es decir 4 GBytes.

3. **Atributos** o derechos de acceso, se trata de un campo de 12 bits, que proporciona las características relevantes del segmento como:

    - Tipo de segmento, admitiendo las variantes de legible, escribible, ejecutable o una combinación de estos.

    - Nivel de privilegio, que oscila entre 0 y 3. Es el grado de seguridad que tiene el contenido del segmento en el sistema.

    - Indicadores sobre aspectos relacionados con la gestión de la memoria virtual, como el que indica si el segmento se halla cargado o no en la memoria física.

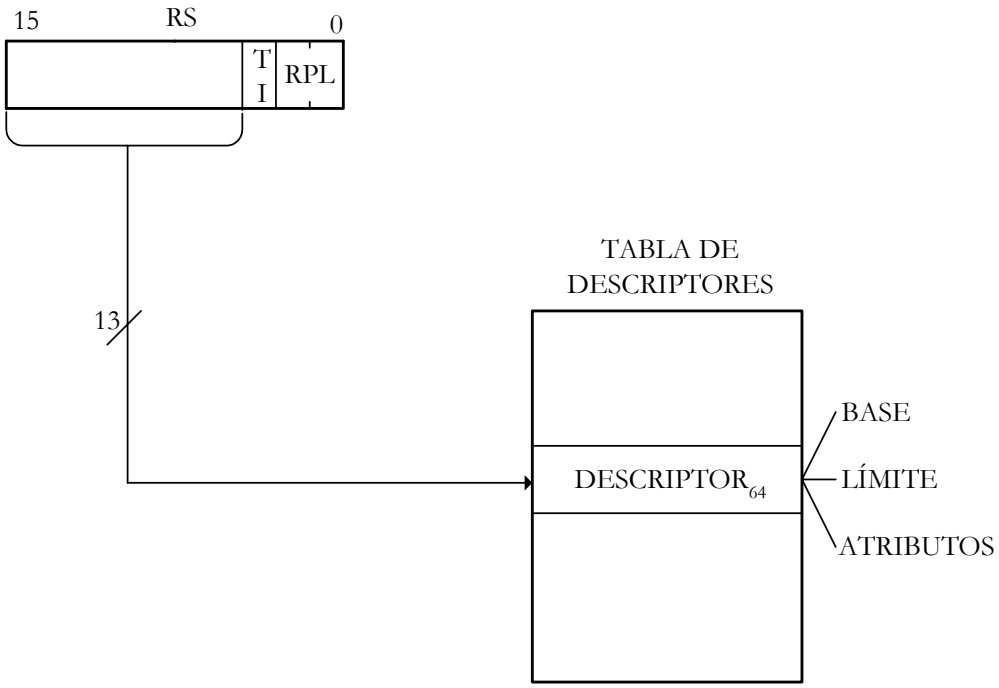


Figura 7.14 – Se accede a la tabla de descriptores consultado el selector y comprobando el índice de tabla.

Haciendo mención a lo comentado en la figura anterior, la figura 7.14, si **TI** (Indice de Tabla) es 0, se accederá a la tabla de descriptores global (GDT); en caso contraio, si es 1: se accederá a la tabla de descriptores local (LDT).

Como el índice es de 13 bits el número máximo de descriptores en la tabla de descriptores es de 2^{13} = 8 KByte. Por lo tanto se puede apuntar a 16 segmentos distintos. El máximo de memoria virtual que puede utilizar el Pentium sería 16 K descripores X 4 GBytes = 64 TBytes.

Al conjunto de los parámetros base (32 bits), límite (20 bits) y atributos (12 bits) que se utilizan para definir un segmento se denomina **descriptor de segmento** y tiene una longitud de 64 bits.

En modo protegido para obtener el descriptor de segmento, el Pentium utiliza el valor de registro de segmento, para acceder a unas tablas residentes en memoria principal. Como necesita disponer directamente de estas características, a cada segmento se le asocia un registro caché ultrarrápido (que se les denomina "ocultos" o "invisibles") que no son accesibles ni en lectura ni en escritura por el programador de sistemas, ni por el de aplicaciones. Es la CPU quien se encarga de gestionarlos automáticamente cada vez que se modifica el contenido de un registro o segmento.

Cuando se carga un registro segmento, la CPU busca automáticamente en las tabla de descriptores residentes en memoria principal, la base, el limite y los atributos del segmento referenciado y los carga en el registro caché asociado. A partir de aquí, todo acceso a dicho segmento, se realizará utilizando los datos que se encuentran en el registro caché, de acceso muy rápido. Cuando se modifica el valor de algún registro de segmento, surge una penalización en el tiempo ya que se tiene que actualizar la caché. Por eso hay cuatro registros de datos (ES, DS, FS, y GS) ya que lo normal es cambiar los segmentos de datos y no los de código. Todo ello se muestra gráficamente en la figura 7.15.

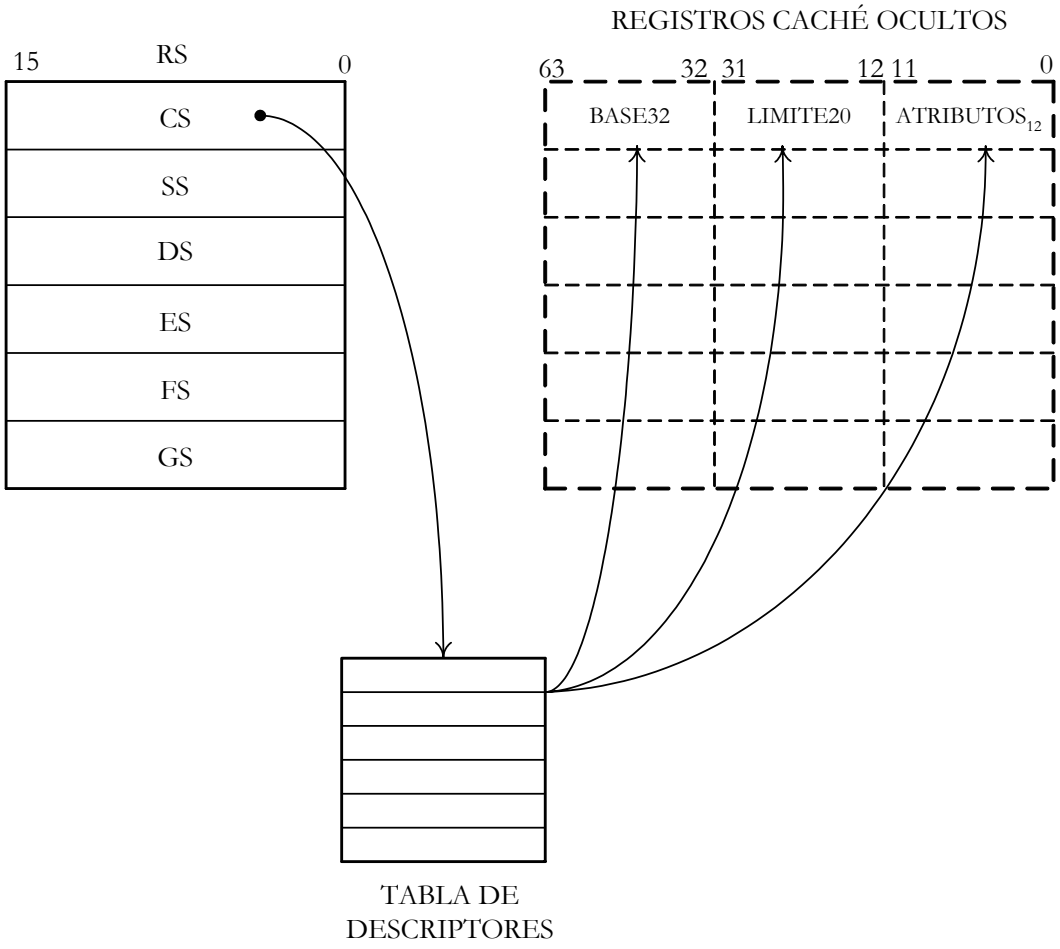


Figura 7.15 – En modo protegido, los registros de segmento actuan como selectores "visibles", con los que se accede a tablas en las que se hallan los parámetros que definen al segmento, es decir, la base, el límite y los atributos. Estos parámetros los carga automáticamente la CPU en los registros caché asociados, que son invisibles, es decir, inaccesibles al programador de sistemas y al de aplicaciones.

Como la carga de los registros caché invisibles consume un tiempo extra, se comprende que la instrucción PUSH DS se ejecute más rápidamente que POP DS, ya que la primera modifica el valor de DS y la CPU automáticamente busca en las tablas los valores que definen el nuevo segmento y los carga en el registro caché asociado a DS.

En modo real, también se usan los registros caché invisibles, siendo el valor de la base el del registro de segmento con cuatro ceros añadidos, el límite siempre es el máximo, o sea, 64 KBytes y todos los derechos de acceso están permitidos.

En capítulos posteriores, se estudiará en profundidad el mecanismo de direccionamiento en modo protegido . En este capítulo sólo se ha presentado la estructura de los registros disponibles por el programador de aplicaciones y su función básica.

7.5. JUEGO DE REGISTROS DE LA UNIDAD EN COMA FLOTANTE

Esta Unidad se ha rediseñado completamente respecto a la que se usa en el 486. Sin embargo, mantiene compatibilidad 100% binaria con ella. Incorpora un cauce segmentado de instrucciones de ocho etapas, que permite obtener resultados partiendo de instrucciones de coma flotante en cada ciclo de reloj. Las cuatro primeras etapas son las mismas que poseen las unidades de enteros. La quinta y la sexta, corresponden a la ejecución de las instrucciones de coma flotante. La séptima etapa se encarga de escribir el resultado en los registros adecuados y la octava realiza el informe de posibles errores que se hayan producido.

Esta unidad hace uso de nuevos algoritmos que aceleran la ejecución de las operaciones e incluye elementos de hardware dedicados, como son: un multiplicador, un sumador y un divisor. Las instrucciones de suma, multiplicación y carga de datos se ejecutan tres veces más rápido que en un 486.

La Unidad en Coma Flotante o FPU integrada en el Pentium amplía el número de registros que puede manejar el programador de aplicaciones, así como el repertorio de instrucciones. Los nuevos registros que proporciona el coprocesador son ocho generales de 80 bits cada uno y tres de 16 bits, siendo éstos últimos empleados en labores de control y presentación de estado (ver Figura siguiente).

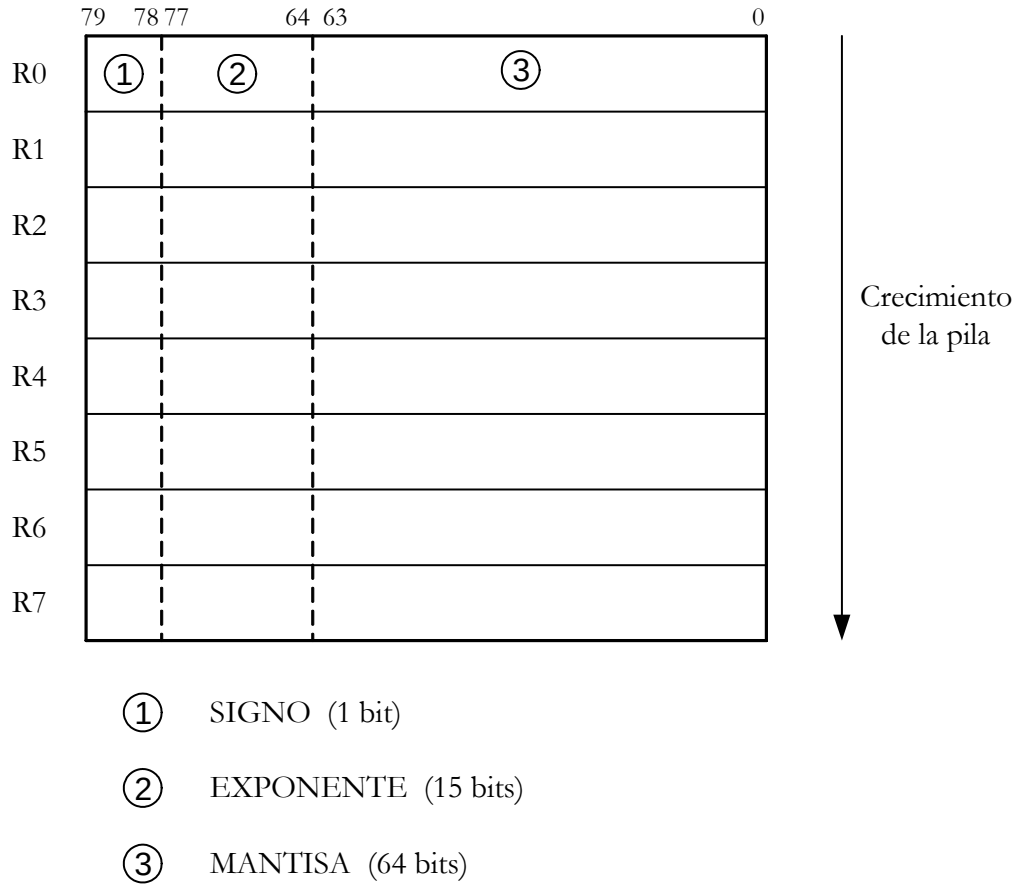


Figura 7.16 – Registros del coprocesador, accesibles al programador de aplicaciones.

Los registros generales R1 a R8 soportan datos con el formato normalizado de doble precisión con 80 bits. Como han de ser manejados en formato de pila, el coprocesador tiene un puntero de control. De pila llamado "St". Toda interacción que tengamos que hacer con los registros del coprocesador se realiza a través del puntero de pila St, donde el último valor introducido es St o St(0) y si hubiéramos rellenado todos los registros el último sería St(7).

EJEMPLO 4

St(0)=1345.
St(0)=5431.

El primer lugar, se carga en el coprocesador el dato 1345 y a continuación el 5431. Por lo tanto, tendremos en St(0) el dato 5431 y en St(1) el 1345.

Podemos observar cómo dichos registros están divididos en tres secciones: signo, exponente de 15 bits y mantisa de 64 bits.

Todas las operaciones del coprocesador se realizan usando los ocho registros generales, que están organizados en forma de pila, no pudiéndose acceder a ellos de forma arbitraria.

El coprocesador soporta los siguientes tipos de datos:

1. Enteros de 16, 32, 64 bits.
2. BCD empaquetados de 80 bits.
3. Números en coma flotante de 32, 64 y 80 bits.

En la figura 7.17 se muestra la distribución de los 16 bits de Palabra de Estado del coprocesador matemático.

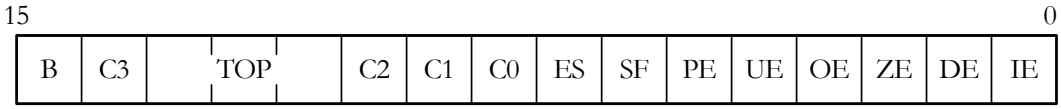


Figura 7.17 – Distribución de los bits en la Palabra de Estado del coprocesador matemático del Pentium

- **TOP**: Este campo de 3 bits de la Palabra de Estado muestra el primer registro. Realiza labores similares al puntero de pila ESP. Estos bits pueden, mediante instrucciones en coma flotante tales como FLD y FSTP (éstas son similares a las instrucciones de tipo entero PUSH y POP), decrementar el TOP en 1 y colocar un valor en su respectivo registro o incrementar el TOP en 1 y retirar el registro pertinente. La pila se incrementa de manera descendente para registrar números menores. Implícitamente, la mayor parte de instrucciones direccionan el registro cima de la pila, es decir, el registro con el número almacenado en el campo TOP del registro de estado. También podremos especificar explícitamente un registro con diversas instrucciones en coma flotante. Fíjese que el registro especificado no es absoluto sino relativo a TOP.

- **B y ES**: Siempre tienen el mismo valor e informa del estado de error. Si ES = 1 se produce una excepción no mascarable cuyo motivo lo indican los bits SP e IE. Bajo ningún concepto el bit B produce ningún dato con respecto al estado de la unidad numérica y en consecuencia al pin BUSY.

- **SF**: Diferencia entre las operaciones inválidas causadas por el desbordamiento de la pila y por otras causas. Si SF está activado el bit C1 diferencia entre un overflow (C1 = 1) y un underflow (C1 = 0). Para una interpretación del código C3-C0 de una comparación u operaciones similares adjuntamos la tabla 7.2 que se muestra a continuación.

INSTRUCCIÓN	C0	C3	C2	C1
FCOM, FCOMP, FCOMPP, FICOM, FICOMP, FTST, FUCOM, FUCOMP, FUCOMPP	Resultados de comparaciones		Operandos no comparables	0 Ó #IS
FCOMI, FCOMIP, FUCOMI, FUCOMIP	No definidas			#IS
FXAM	Clase de operando			SIGNO
FPREM, FPREM1	Q2	Q1	0 = reducción completa 1 = reducción incompleta	Q0 Ó #IS
F2XM1, FADD, FADDP, FBSTP, FCMOVCC, FIADD, FDIV, FDIVP, FDIVR, FDIRVP, FIDIV, FIDIVR, FIMUL, FIST, FISTP, FISUB, FSUBR, FMUL, FPATAN, FRNDINT, FSCALE, FST, FSTP, FSUB, FSUBP, FSUBR, FSUBRP, PSQRT, FYL2XP1	No definido			Roundup ó #IS
FCOS, FSIN, FSINCOS, FPTAN	No definido		1 = operando fuente fuera de rango	Roundup ó #IS (no definido si C2 =1)
FAB, FBLD, FCHS, FDECSTP, FILD, FINCSTP, FLD, CONSTANTES DE CARGA, FSTO (EXT. REAL), FSCH,. FXTRACT	No definido			0 ó #is
FLDENV, FRSTOR	Cada bit se carga de la memoria			
FFREE, FLDCW, FCLEX/FNCLEX, FNOP, FSTCW/FNSTCW, FSTENV/FNSTENV, FSTSW/FNSTSW	No definido			
FINIT/FNINIT, FSAVE/FNSAVE	0	0	0	0

Tabla 7.2 – Códigos en coma flotante para el Pentium.

- Los cuatro **flags de condición de código** (desde C0 a C3) indican el resultado de operaciones aritméticas y de comparación en coma flotante. Estos flags se usan principalmente para el almacenamiento de información usada en excepciones. El flag de condición de estado C1 es usado para gran variedad de funciones. Cuando los bits IE y SF de la CPU están activados, indicando desbordamiento de la excepción overflow o underflow (#IS), el flan C1 lo distingue de la siguiente manera: si está a 1 overflow, sino underflow. Cuando el flag PE de palabra de estado está activado indica que se ha producido un resultado inexacto (redondeo), el flag C1 Es puesto a 1 si los últimos redondeos han sido descendentes. Las instrucciones FXAM activa C1 para examinar el valor del signo.

  El bit de condición de código C2 es usado por las instrucciones FPREM y FPREM1 para indicar una reducción correctamente (o resto parcial). Cuando se completa una reducción completamente, los flags de condición de estado C0, C3 y C1 son activados haciendo referencia a los tres bits menos significativos del cociente (Q2, Q1, y Q0 respectivamente).

  Las instrucciones FPTAN; FSIN, FCOS y FSINCOS ponen el flag C2 a 1 para indicar que el operando fuente está fuera de rango permitido (más/menos 2^{63}).

- **PE**: Precisión.
- **UE**: Underflow.
- **OE**: Overflow.
- **ZE**: División por cero.
- **DE**: Operando desnormalizado.
- **IE**: Operación inválida.

Estos 6 últimos bits (cada uno de ellos por separado) si ponen su bit a 0 es para indicar que no se produce excepción y 1 para indicar que se ha generado la condición de excepción.

Bajo ciertas circunstancias, el Pentium genera una excepción de coprocesador. Estas excepciones pueden ser individualmente enmascaradas. Lo que es más, podemos determinar diferentes modos para precisión y redondeo. La Palabra de Control se emplea para este propósito, y la estructura la mostramos en la figura 7.20:

Figura 7.18 – Distribución de los bits en la Palabra de Control del coprocesador matemático del Pentium

- **IC**: No tiene significado cuando no manejan valores infinitos porque el Pentium aplica el estándar IEEE a las operaciones con coma flotante. En los terrenos de compatibilidad con el 8087 y el 80287 el bit IC está disponible pero no tiene efecto. El Pentium siempre maneja cantidades infinitas en el sentido de más menos infinito., incluso si ponemos IC = 0.

- **RC**: Los dos bits RC controlan el redondeo en el modo definido. En la siguiente tabla se muestran los valores que puede contener el campo RC y el significado de los mismos.

MODO DE REDONDEO	CAMPO RC
Redondeo al más cercano	00B
Redondeo hacia abajo	01B
Redondeo hacia arriba	10B
Redondeo a cero	11B

Tabla 7.3 – Códigos del campo RC para el Pentium

- **PC**: El campo de control de precisión (bits 8 y 9 de la Palabra de Control de la FPU) determina la precisión (64, 53 o 24 bits) de los cálculos en coma flotante realizados por la FPU. Controla el redondeo en las instrucciones ADD, SUB, MUL, DIV y SQRT. En la tabla que se adjunta a continuación, se muestran los valores que puede contener en campo PC y el significado de los mismos.

PRECISIÓN	CAMPO PC
Precisión simple (24 bits)	00B
Reservado	01B
Doble precisión (53 bits)	10B
Precisión extendida (64 bits)	11B

Tabla 7.4 – Códigos del campo PC para el Pentium

- **PM, UM, OM, ZM, DM e IM**: Controlan la generación de una excepción cada uno y su interrupción resultante. Los Pentium provocan seis excepciones diferentes en sus operaciones con coma flotante. Podemos enmascarar las excepciones individualmente empleando los bits PM, UM, OM, ZM, DM e IM. Entonces el Pentium ejecuta una rutina estándar para tratar los respectivos errores utilizando un supuesto estándar. Esto es una parte integral del chip.

Hay disponible un registro de estado más, la Palabra Tag. La FPU usa los valores de los tag para detectar desbordamientos en condiciones de overflow o underflow. Los desbordamientos de overflow ocurren cuando el puntero del campo TOP es decrementado para apuntar a un registro no vacío. Los desbordamientos de underflow se producen cuando el puntero del campo TOP es incrementado para apuntar a un registro vacío, o cuando un registro vacío es referenciado como un operando fuente. Un registro no vacío viene definido como un registro que contiene valor cero (01), valor válido (00) o valor especial (10). Su estructura la mostramos en la figura 7.21 Tag_7 –Tag_0 contiene información, que identifica los contenidos de los ocho registros de datos R7-R0. El coprocesador utiliza esta información para realizar ciertas operaciones a una velocidad superior. Empleando este proceso, el Pentium puede determinar muy rápidamente ciertos valores como NAN, infinito y sin la necesidad de decodificar el valor del registro correspondiente. Mediante las instrucciones de FSTENV/FNSTENV podemos almacenar la Palabra Tag en memoria y examinarla.

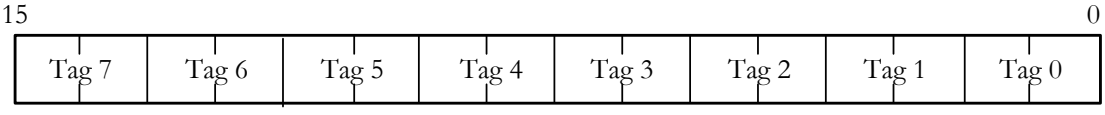
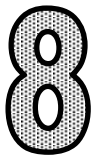


Figura 7.19 – Distribución de los bits en la Palabra Tag del coprocesador matemático del Pentium

MEMORIA SEGMENTADA



8.1- ORGANIZACIÓN DE LA MEMORIA.

La memoria que controla en Pentium está organizada en bytes, palabras, dobles palabras y cuádruples palabras. El byte es el elemento básico de la memoria del Pentium. Las palabras se almacenan en dos bytes, quedando el byte de menor peso en la dirección inferior. Las dobles palabras ocupan cuatro bytes consecutivos, depositándose el byte inferior en la dirección más baja y el más significativo en la dirección superior (numérica). Una cuádruple palabra consta de ocho bytes de direcciones consecutivas.

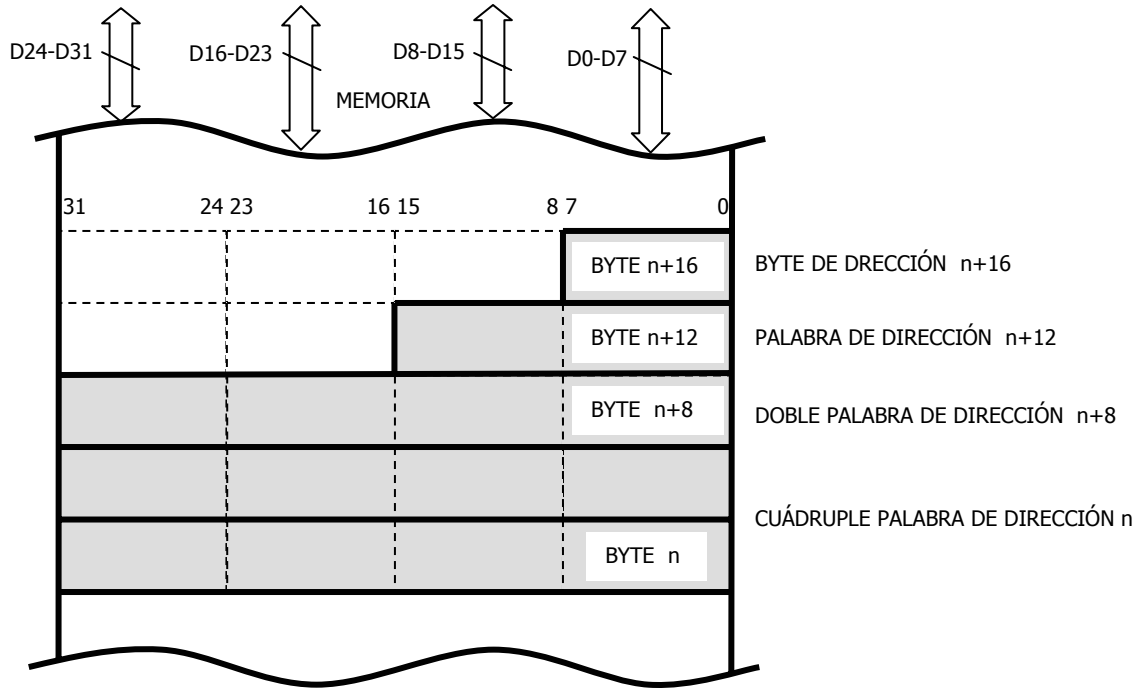


Figura 8.1. Distribución en la memoria de bytes, palabras, dobles palabras y cuádruples palabras.

Dado que el bus de datos es de 32 líneas divididas en 4 grupos de 8, los de dobles palabras conviene que comiencen en una dirección múltiple de 4, para que un ciclo en un ciclo los 32 bits que se componen se transfieran por el bus de datos. Si no cumple esta condición de transferencia de la doble palabra se realiza en dos ciclos. Para evitar estas penalizaciones de tiempo las cuádruples palabras deben ocupar direcciones de múltiplo de 4 y las palabras las direcciones pares

Para solucionar este problema la memoria debe estar "alineadas", es decir que las palabras se encuentren en dirección par y las dobles palabras en direcciones múltiplos de cuatro.

También existen instrucciones que reconocen los siguientes tipos de datos: enteros, ordinales, enteros BCD empaquetados y sin empaquetar, punteros de direcciones, campos, cadenas y números en coma flotante.

Además de los tipos básicos de estructuras de datos mencionadas, el Pentium maneja otros dos mucho más complejos:

1. Segmentos
2. Páginas.

Los segmentos son bloques de memoria de tamaño variable, que contienen información de la misma clase y constituyen el objeto principal sobre el que se basa el mecanismo de protección. La

segmentación es eficaz para organizar la memoria en módulos lógicos de similares características, que son el soporte de la programación estructurada. Es posible compartir recursos entre todas las tareas si se sitúan en el espacio global, o bien, ser exclusivos de una tarea concreta si están ubicados en el área local de la misma. Se puede asignar a cada segmento un determinado nivel de privilegio. Intel basa el control de la memoria en la segmentación por eso siempre está activada.

La paginación divide el espacio de memoria en trozos de longitud fija, llamados páginas, que, en el caso del Pentium, tienen un tamaño de 4 KB ó 4 MB. La paginación simplifica la labor del programador de sistemas al simplificar los algoritmos de intercambio entre objetos de la memoria física y la memoria virtual, al manejar elementos del mismo tamaño lo que supone un incremento de la velocidad en la localización y relocalización de páginas. La paginación es optativa y sólo funciona cuando la activa el programador.

La paginación y la segmentación son técnicas complementarias y ambas introducen ventajas particulares en la gestión de la memoria virtual. Los sistemas operativos UNIX y DOS son, respectivamente, ejemplos representativos de dichas técnicas.

El Pentium es capaz de combinar ambos métodos cuando funciona con segmentación paginada. De esta forma, el programador de aplicaciones estructura la memoria lógica en segmentos que soportan la multitarea, mientras el programador de sistemas emplea la paginación en el manejo y transferencia de bloques en la memoria física.

Cuando sólo se precisa trabajar con la paginación, al estar activa siempre la segmentación, es preciso que toda la memoria física se considere como un único segmento, que ocupa un espacio continuo o lineal. El modelo "plano" permite esta posibilidad.

En la segmentación paginada, el Pentium utiliza la paginación para descomponer los segmentos en páginas de 4 KB ó de 4 MB de tamaño, que se distribuyen aleatoriamente en la memoria física. No todas las que conforman un segmento tienen que residir en ella, sino sólo las que van a ser procesadas.

8.2- LA MEMORIA EN MODO REAL.

Cuando el procesador trabaja en Modo Real, lo hace de forma similar al 8086 con el fin de ser compatible con el software creado para dicho microprocesador de 16 bits.

Siempre que se realiza la operación RESET o de puesta en marcha, el Pentium comienza trabajando en Modo Real.

En la memoria en Modo Real, se encuentran 8 registros de propósito general de 16 bits (AX, BX, CX, DX, SP, BP, SI y DI), aunque también se pueden encontrar los registros extendidos de 32 bits (EAX, EBX, ECX, EDX, ESP, EBP, ESI y EDI) que son accesibles para programas que utilizan una operación para ignorar el tamaño de los operandos. Además de estos registros generales se encuentran otros que son los denominados registros de segmento (CS, DS, SS, ES, FS y GS). Aunque los registros FS y GS son para programas que explícitamente acceden a ellos.

En este modo se contempla un espacio de direcciones directamente accesibles por la CPU de 1MB (solo usa 20 líneas de direcciones), en el que sólo es posible aplicar la técnica de la segmentación. Se multiplica el contenido del registro segmento por 16 (se le añaden cuatro ceros) para hallar la base del mismo y luego para calcular la dirección a acceder, se suma al resultado obtenido, un desplazamiento de 16 bits, lo que implica que el tamaño máximo del segmento es de 64 KB. Como se ve en Modo Real los selectores de segmento son utilizados para formar la dirección lineal.

```
DIRECCIÓN EFECTIVA = RS x 16 + DESPLAZAMIENTO
```

Para direccionar una instrucción en el segmento de código, se usa como registro de segmento a CS y como desplazamiento el contenido de IP, que es la parte baja (16 bits) del Registro Puntero de Instrucciones (EPI).

Si se hace referencia a la pila, SS actúa como registro de segmento y SP como desplazamiento (palabra baja de ESP). Finalmente, si se desea direccionar un dato, se toma como registro a DS por defecto, ya que también puede actuar ES, FS o GS, si se explicita en la instrucción. Como desplazamiento se usa el que se indique en la propia instrucción del operando. Ejemplo:

```
MOV EBX, ES: DESPLAZAMIENTO
```

Esta instrucción mueve al registro EBX una información de 32 bits situada en la memoria en la dirección ES x 16 + DESPLAZAMIENTO.

En la figura 8.2 se muestra un ejemplo de direccionamiento de la memoria en Modo Real. En cada momento, la CPU tiene activados a un segmento de código (referenciado por CS), otro de pila (que selecciona con SS) y hasta cuatro datos, referenciados por DS, ES, FS y GS.

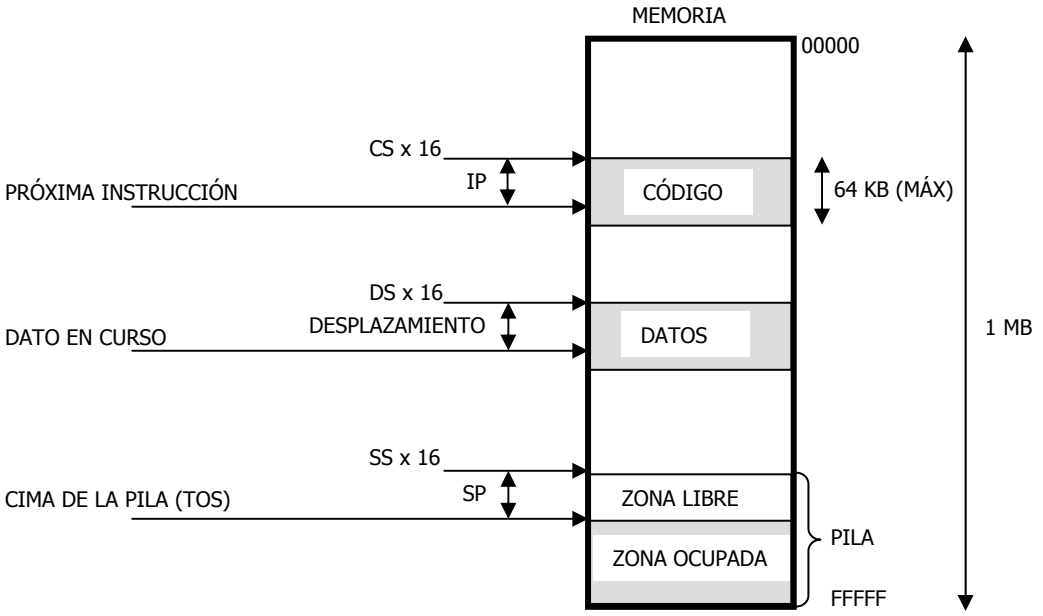


Figura 8.2. La manipulación de los registros de segmento y los desplazamientos permite referenciar en cada momento a las posiciones de memoria que contienen la próxima instrucción, el dato en curso y la cima de la pila.

Aunque en Modo Real el Pentium manipula, por defecto, operandos y desplazamientos de 16 bits, es posible usar elementos de 32 bits de longitud. Además, en este modo existen nuevas instrucciones respecto al 8086 y se han potenciado otras. Todo ello implica el uso de un "compilador" de lenguaje Ensamblador más complejo si se desea alcanzar el máximo rendimiento. Por otra parte, el Pentium precisa menos ciclos de reloj que el 8086 para ejecutar la mayoría de las instrucciones, de esta manera el tiempo de ejecución del software para el 8086 se reduce notablemente cuando se usa el Pentium.

En Modo Real, todas de las instrucciones específicas de protección de modo como LLDT (load local descriptor table) son tan inútiles como los códigos innecesarios.

Como el acceso a las posiciones de los segmentos se hace mediante direccionamiento relativo (desplazamiento) se obtiene como ventaja adicional la facilidad en la relocalización, característica esencial para soportar la portabilidad de los programas sobre diferentes configuraciones físicas.

8.3- LA MEMORIA EN MODO PROTEGIDO.

El Modo Protegido fue originalmente implementado en el 80826 para proteger a las diferentes tareas cuando el sistema está operando en multitarea, contra los posibles accesos incorrectos o inválidos.

Dentro de la memoria contemplada por el Pentium, se pueden distinguir tres espacios:

- Espacio virtual o lógico.
- Espacio lineal.
- Espacio físico.

El espacio virtual abarca toda la dimensión de la memoria virtual y es el que maneja el programador de aplicaciones. Como la dirección virtual es de 46 bits el tamaño del espacio virtual (memoria de masa o disco) es de 2^{46} =64 TB. La Unidad de Segmentación, cuya activación siempre es obligada, traduce las direcciones virtuales a lineales, que reciben este nombre porque hacen referencia a segmentos que, al situarse sobre la memoria física, tienen dispuestas todas sus posiciones en orden consecutivo o lineal. Cuando la Unidad de Paginación, optativa, no está activada, la dirección lineal coincide con la dirección física, es decir, la de acceso a la memoria principal ligada directamente a la CPU. Si la Unidad de Paginación está activada, cada segmento se descompone en un número variable de páginas del mismo tamaño (4KB ó 4MB), y la Unidad de Paginación deposita a dichas páginas sobre la memoria física en los huecos que encuentra libres, no una detrás de otra, lo que significa que la dirección lineal se debe traducir a física de acuerdo con esta distribución aleatoria de las páginas (figura 8.3). La memoria física o principal (DRAM) al direccionarse con 32 bits admite un tamaño de 2^{32} = 4 GB.

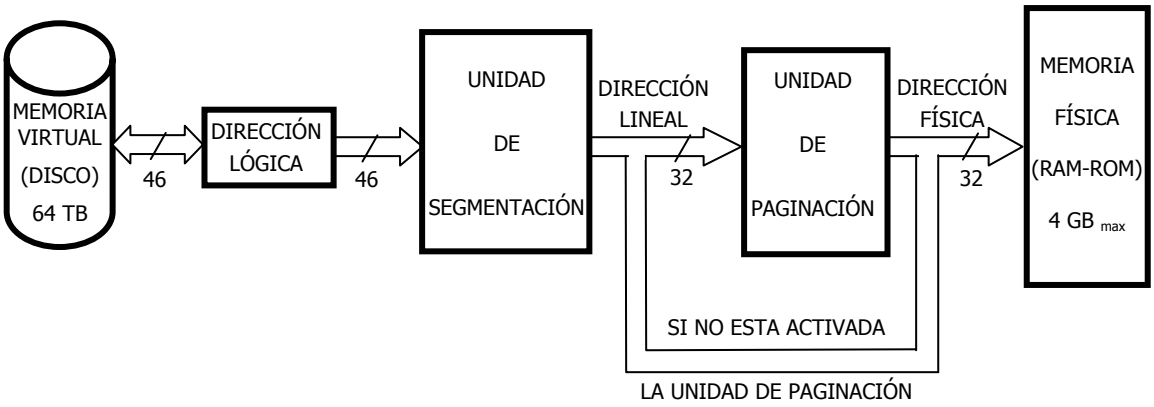


Figura 8.3.Traducción de la dirección lógica en dirección lineal y física.

Los programas de aplicaciones que procesa el computador sólo hacen referencia a direcciones virtuales y es la MMU (integrada en el hardware del Pentium) la que se encarga de traducirlas a direcciones físicas. Como la memoria física es mucho más pequeña que la virtual, la MMU también

deberá detectar la ausencia de los elementos que no estén cargados en la memoria física, para comunicárselo mediante una excepción al Sistema Operativo, el cual realiza el traslado de dichos elementos desde la memoria virtual a la física (figura 8.4).

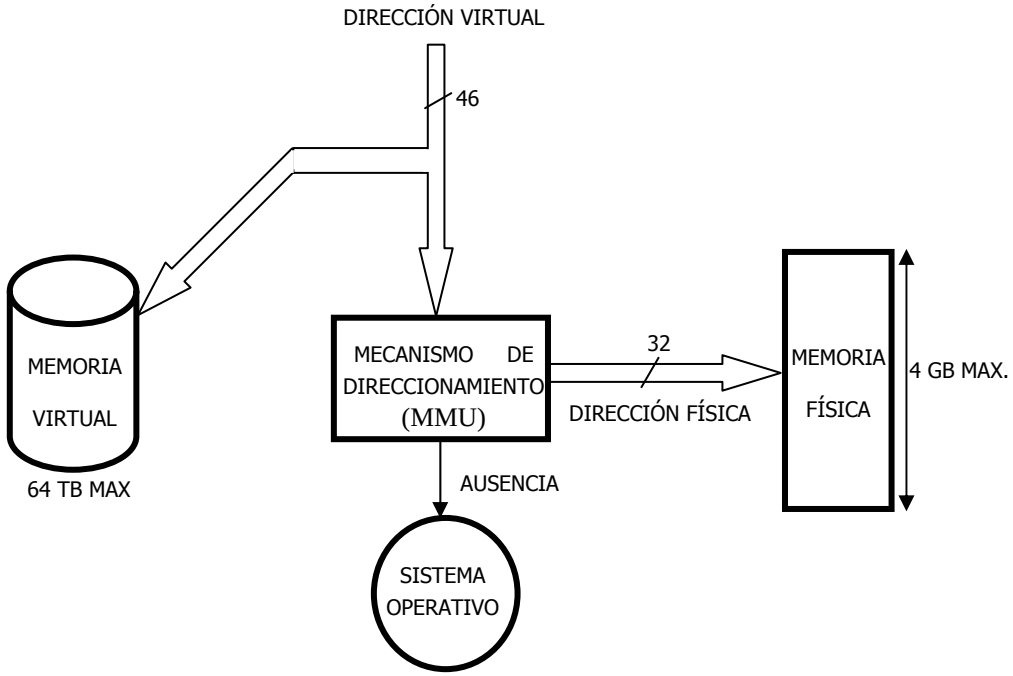


Figura 8.4. El mecanismo de direccionamiento traduce la dirección virtual a física, y, caso de no hallarse el elemento en la memoria, se detecta su 'ausencia' y se genera una excepción encargada de hacer la transferencia oportuna que es soportada por el Sistema Operativo.

8.4- EL ESPACIO VIRTUAL O LÓGICO.

La dirección lógica que usan los programadores de aplicaciones, consta de dos partes:

1. **Selector:** Es un campo de 14 bits, que selecciona un determinado segmento del espacio virtual.
2. **Desplazamiento:** Es un campo reducido de 32 bits que determina una posición de un segmento cuya capacidad máxima es de 4 GB. Este campo puede tener una longitud de 16 bits para mantener la compatibilidad con el software procedente de los microprocesadores de 16 bits (8086-80286), en cuyo caso los segmentos admiten un tamaño máximo de 64 KB.

En realidad, el puntero de direcciones virtuales tiene el siguiente formato:

```
SELECTOR(14): DESPLAZAMIENTO(32),
```

El campo selector está contenido en los 14 bits de más peso del registro de segmento y los dos restantes sirven para referenciar al nivel de privilegio del peticionario del segmento y no se utilizan en el direccionamiento propiamente dicho. Dicha pareja de bits forman el campo RPL. El campo RPL puede tener 4 niveles, teniendo un valor máximo cuando vale 00 y un valor mínimo cuando vale 11.

En el campo selector está contenido también el TI (Indicador de Tabla), que indica que la Tabla de Descriptor es local (TI=1) o global (TI=0), ayudando a localizar el segmento determinado en la memoria.

Como selector, actúa uno de los registros de segmento de 16 bits, en donde los 14 bits de más peso son discriminantes y direccionan un máximo de 16 K descriptores de segmento.

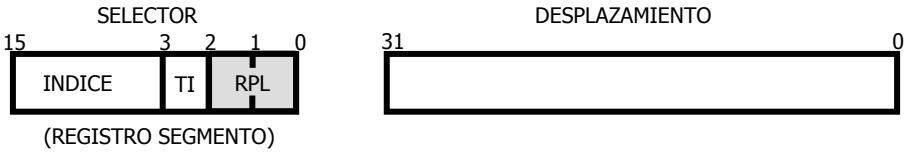


Figura 8.5. Una dirección lógica consta de un campo SELECTOR de 16 bits contenido en un registro segmento, y un campo DESPLAZAMIENTO de 32 bits.

El desplazamiento es un campo de 32 bits cuyo valor se suma a la base del segmento para hallar la dirección a acceder. Cuando se accede a código y CS actúa como selector el desplazamiento lo conforma el contenido de EIP. Si se accede a la pila, SS hace de selector y ESP de desplazamiento. Finalmente, cuando se accede a datos, uno de los registros de segmento DS, ES, FS o GS, actúa como selector, y el desplazamiento se calcula de acuerdo con el modo de direccionamiento utilizando en la instrucción en curso.

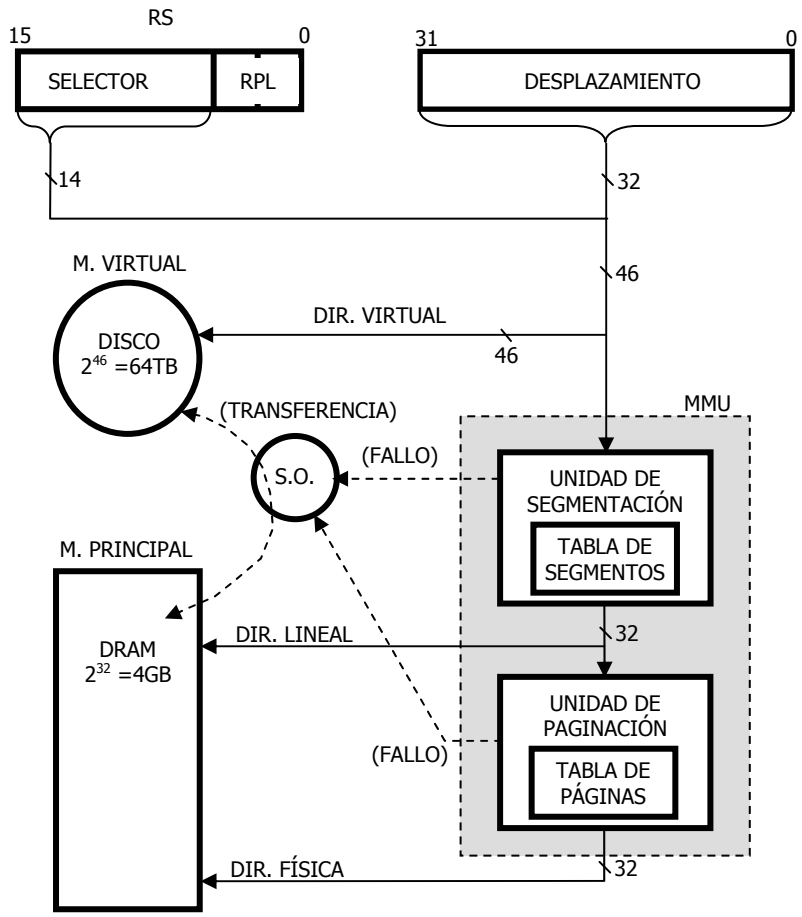


Figura 8.6.

Los 14 bits de más peso del Registro de Segmento (RS) determinan el selector del segmento en la memoria virtual y sus dos últimos bits indican el nivel de privilegio del peticionario (RPL), es decir, el nivel de privilegio del segmento que ha solicitado usar ese selector.

El Pentium dispone de un mecanismo hardware llamado MMU, Unidad de Manejo de Memoria, por el cual convierte la dirección virtual de 46 bits a dirección física de 32 bits. La MMU recoge la dirección virtual de 46 bits y lo introduce en la Unidad de Segmentación, que funciona siempre. La Unidad de Segmentación contiene la Tabla de Segmentos, que determina y registra los segmentos que están en la memoria principal y qué posición ocupan. Si el segmento solicitado está en la memoria principal, se traduce la dirección virtual a dirección lineal de 32 bits y se accede a dicha posición de memoria. Si por el contrario, no está presente (fallo), el Sistema Operativo realiza una transferencia que consiste en coger el segmento del disco y llevarlo a la memoria principal, actualizando también la Tabla de Segmentos, para dar curso a la traducción de dirección virtual a dirección lineal.

En el caso de que funcione la paginación, la dirección lineal que sale de la Unidad de Segmentación pasa a la Unidad de Paginación. La Unidad de Paginación contiene la Tabla de Páginas, que soporta la ubicación de las páginas en que se han dividido los segmentos y que están en memoria principal. Si la página está presente en memoria, la dirección lineal se traduce a dirección física de 32 bits. Si no está presente (fallo), avisa al Sistema Operativo para que este realice la transferencia correspondiente y actualice las tablas.

8.5- EL ESPACIO LINEAL.

La Unidad de Segmentación siempre se halla activada en el Pentium y se encarga de traducir la dirección lógica de 46 bits en dirección lineal de 32 bits. La dirección lineal coincide con la física correspondiente a la memoria principal si la Unidad de Paginación no está activada.

Se supone que no funciona la Unidad de Paginación dejando para el siguiente capítulo la explicación de su actuación.

Los objetos con los que opera la Unidad de Segmentación son los segmentos, por lo tanto en la memoria principal sitúa y mueve segmentos completos. De ahí proviene el nombre de dirección lineal, que significa que la segmentación referencia a bloques (segmentos) que tienen todas sus posiciones ordenadas consecutiva o linealmente.

Aunque la segmentación sea beneficiosa dada la adaptación de la memoria a las modernas técnicas de programación estructurada, tiene el inconveniente de trabajar con segmentos de tamaño variable, lo que complica el mecanismo de transferencia entre las memorias virtual y física, así como la optimización en el comportamiento del espacio de esta última (figura 8.7).

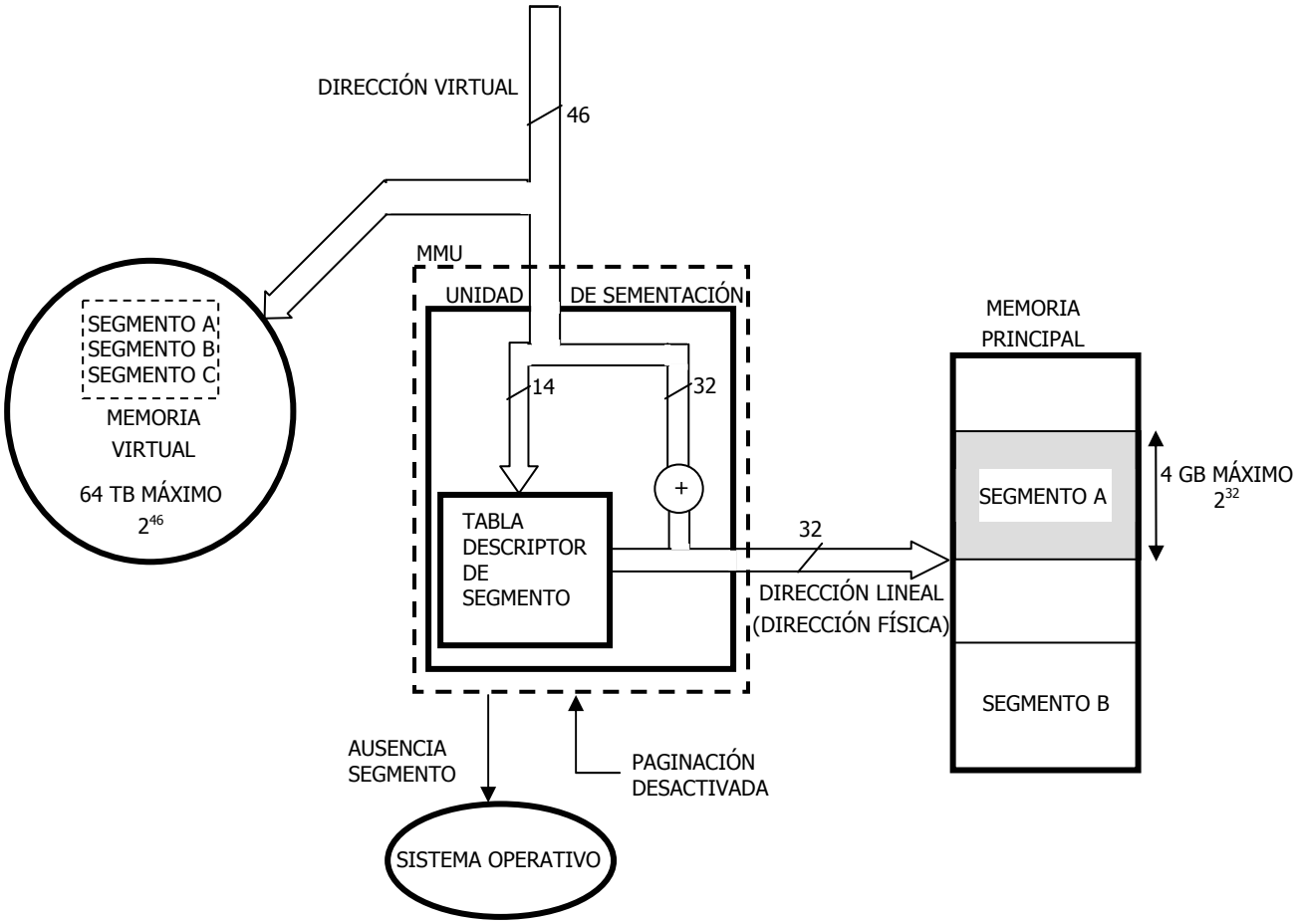


Figura 8.7. La Unidad de Segmentación, cuando está desactivada la Paginación, traduce las direcciones virtuales en lineales (físicas) y opera con segmentos completos, de forma que en todo momento la memoria principal contiene una parte de los segmentos existentes en la memoria virtual.

Cuando la dirección virtual hace referencia a un segmento que no está cargado en la memoria principal (puesto que no caben todos los existentes en la memoria virtual) la Unidad de Segmentación detecta su ausencia, o sea, el fallo del segmento. En esta situación, provoca una excepción, que pone en marcha una rutina del Sistema Operativo que se encarga de trasladar dicho segmento desde la memoria virtual a la memoria física. Al tener que manejar esta rutina transferencias de bloques de tamaño variable los algoritmos que la configuran son largos y complejos. Tras cargar el segmento en la memoria principal la rutina actualiza la tabla de segmentos y se da curso al acceso.

8.5.1- Descriptores de segmento.

En el modo protegido un segmento queda especificado por tres parámetros: Base de 32 bits, Límite o tamaño de 20 bits y Atributos de 12 bits. Al conjunto de estas informaciones se llama "Descriptor" que es una estructura de 8 bytes.

Los 14 bits de más peso del Registro Segmento (RS) conforman el Selector del Segmento que actúa como una entrada en la Tabla de Descriptores, que referencia a los segmentos que maneja la MMU en donde se encuentra la información necesaria para definir al segmento referenciado.

Un descriptor de segmento es una estructura de datos compuesta por ocho bytes, que contiene los parámetros que definen completamente el segmento referenciado (base, límite y derechos de acceso o atributos). Los Descriptores son típicamente creados por compiladores, vínculos, cargadores, o por las operaciones del sistema, pero no aplicaciones de programa.

En la figura 8.8 se muestra el formato que se utiliza en la memoria para contener un descriptor de segmento.

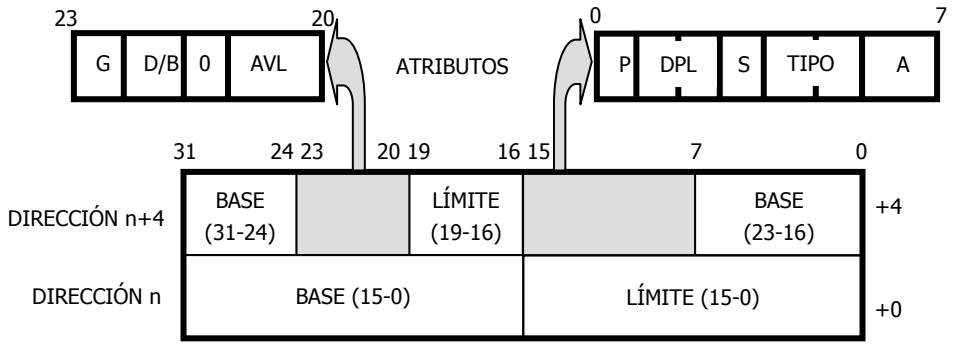


Figura 8.8. Formato de un descriptor de segmento compuesto por ocho bytes, que ocupan dos posiciones de memoria de 32 bits cada una.

Al analizar la estructura del descriptor, se comprueba que en ella residen los tres parámetros que determinan un segmento:

1. **Base**: Es un campo de 32 bits que contiene la dirección lineal donde comienza el segmento.
2. **Límite**: Campo de 20 bits que expresa el tamaño del segmento. Como con 20 bits el tamaño máximo es de 1 MB, hay otro bit complementario en el campo de atributos, llamado de granularidad, G, que indica si el límite está expresado en bytes (G=0) o en páginas de 4 KB (G=1). En este último caso, el tamaño máximo del segmento sería:

$$1\ M \times 4\ KB = 4\ GB$$

3. **Atributo o derechos de acceso**: Se trata de un campo de 12 bits, que proporciona las características relevantes del segmento.

A continuación se describe la función de diversos bits que componen el campo de atributos.

- **Bit de presencia (P):** Indica si el segmento al que referencia el descriptor está cargado, o sea, se halla presente en la memoria principal (P=1), o bien, está ausente (P=0).

  Si P=1, la Unidad de Segmentación accede a la tabla de los Descriptores de Segmento de la MMU y carga la base, el límite y los atributos en el registro caché "invisible" asociado al registro de segmento correspondiente procediendo a continuación al acceso en la memoria principal.

  Si P=0, la Unidad de Segmentación genera una excepción que pone en marcha una rutina del Sistema Operativo encargada de transferir el segmento referenciado desde la memoria virtual a la física. Luego, actualiza el valor del descriptor en la Tabla de Segmentos y da paso a la carga de los parámetros en el registro caché y su posterior acceso.

- **Nivel de privilegio (DPL):** Indica el nivel de privilegio del segmento al que referencia el descriptor. Su valor puede variar entre el 0 y el 3 y es un campo que consta de dos bits.

- **Tipo de segmento (S):** Si S=1, el segmento correspondiente al selector es "normal", o sea, un segmento de código, de datos o de pila.

  Si S=0, se refiere a un segmento del sistema, que referencia un recurso especial del sistema, como puede ser una puerta de llamada, un segmento TSS, etc. Estos segmentos los maneja el programador de sistemas.

- **Tipo:** Los tres bits de este campo distinguen en los normales si se trata de uno de código, de datos o de pila. Además, determinan el acceso permitido: lectura/escritura/ejecución. Posteriormente se estudian los bits de este campo.

- **Accedido (A):** Este bit se pone automáticamente a 1 cada vez que el procesador accede al segmento. Este bit también podría considerarse dentro del Tipo de segmento.

  Muchos S.O. se encargan de borrar periódicamente a este bit y llevar la cuenta del número de veces que es accedido cada segmento, con el fin de implementar adecuadamente los algoritmos que controlan las transferencias entre la memoria virtual y física. En muchos casos, al estar llena la memoria física, hay que establecer qué segmento se ha de eliminar para dejar sitio a uno nuevo. Para esta labor se suele aplicar el algoritmo LRU, que selecciona al segmento que menos veces se haya usado recientemente. Para conocer este dato, el S.O. lleva la cuenta del número de accesos a cada segmento.

- **Granularidad (G):** Los 20 bits del campo LÍMITE del descriptor indican el tamaño del segmento, que estará expresado en bytes si G=0, y en páginas de 4 KB si G=1. En el primer caso, el tamaño máximo del segmento es de 1 MB y en el segundo, de 4 GB.

- **Defecto/grande (D/B):** En los segmentos de código el bit D (Defecto) y en los segmentos de datos de este mismo bit, llamado B (Grande), permiten distinguir los segmentos nativos de 32 bits para el Pentium. Así se mantiene una compatibilidad total con el software creado para el 80286, sin penalizar las nuevas instrucciones

que aporta el Pentium. Si D=1 tanto las direcciones efectivas como los operandos son de 32 bits. Si D=0 se toman sólo 16.

Por ejemplo, la instrucción DEC CX, que decrementa el registro CX, tendrá el mismo código en un segmento perteneciente al 80286, que la instrucción DEC ECX de un segmento del Pentium. Además, la instrucción DEC ECX en un segmento del 80286 y DEC CX en otro del Pentium, estarán precedidas por un prefijo de tamaño byte, que indica que el tamaño del operando es el contrario al tamaño por defecto del tipo de segmento que lo utiliza.

En resumen, con este bit es posible manejar conjuntamente segmentos pertenecientes al 80286 con otros del Pentium.

- **Disponible (AVL):** Este bit está en disposición del usuario para poder diferenciar ciertos segmentos que contengan un tipo determinado de información o que cubran alguna función específica.

Todos los componentes del descriptor son empleados por la Unidad de Segmentación para comprobar si se satisface el conjunto de reglas que protegen a los segmentos, tales como:

1. Si la dirección está dentro del tamaño del segmento.
2. Si el segmento está presente en la memoria.
3. Si el nivel de privilegio del segmento hace posible su acceso.
4. Si se puede leer, escribir o ejecutar.

8.5.2- Tipos de segmentos normales.

Dentro de los segmentos normales, con el bit S =1 en el campo de los atributos, hay dos grandes tipos, cada uno con diversas posibilidades.

- **Segmento de código:**
  1. Sólo ejecutable.
  2. Ejecutable y leíble.
  3. Ajustable o conforming.

- **Segmento de datos:**
  1. Se puede leer y escribir.
  2. Sólo se puede leer.
  3. Es un segmento de pila.

De los tres bits que componen el campo TIPO, dentro de los atributos del descriptor, el bit de más peso E (Ejecutable), diferencia los segmentos de código (E=1), de los segmentos de datos que no se pueden ejecutar (E=0). Figura 8.9

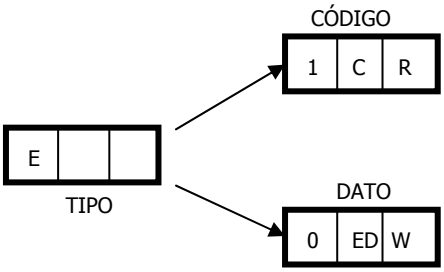


Figura 8.9. El bit E del campo TIPO, distingue entre segmentos de código (E=1) y segmento de datos (E=0). Los otros dos bits toman distinto significado según el tipo de segmento que se trate.

En caso de que E=1, o sea, se referencia a un segmento de código, los otros dos bits de este campo tienen el siguiente cometido:

- **Ajustable o Conforming (C):** Si C=0, al ser accedido el segmento no cambia su nivel de privilegio.

  Si C=1, se llama "segmento ajustable", porque, cuando se accede a él, su nivel de privilegio toma el del segmento que lo ha pedido. Es decir pueden ejecutarse y compartirse por programas con diferentes niveles de privilegio.

- **Leíble (R):** El segmento de código se puede leer si R=1. En ningún caso se puede escribir un segmento de código.

Cuando E=0 y se hace referencia a un segmento de datos, los otros dos bits del campo TIPO tienen el siguiente significado:

- **Expansión decreciente (ED):** Si ED=0, se trata de un segmento de datos normal, datos puros, lo que supone que el crecimiento del mismo se realiza incrementando el valor de la dirección.

  Cuando ED=1, se trata de un segmento de pila pues su crecimiento se efectúa decrementando el valor de la dirección que apunta a su cima, es decir, disminuyendo el valor de la dirección(figura 8.10).

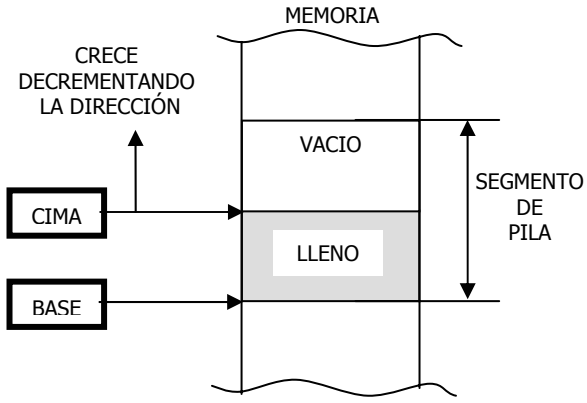


Figura 8.10. Los segmentos de datos con el bit ED = 1, referencian a los segmentos de pila, los cuales crecen disminuyendo el valor de la dirección que apunta su cima.

- **Escribible (W):** Si W=1, el segmento de datos se puede leer y escribir, mientras que, si W=0 sólo se puede leer.

8.6- MANEJO DE LOS DESCRIPTORES

A los descriptores de los segmentos sólo los maneja el procesador automáticamente. Ni el programador de aplicaciones ni el de sistemas tienen acceso a ellos. Los descriptores determinan las características del segmento.

Los descriptores están agrupados en Tablas disponibles en la memoria principal. Cuando en un programa se desea acceder a un nuevo segmento se ejecuta una instrucción. Por ejemplo si se quiere cambiar de segmento de código se ejecuta una " jmp CS':DESPLAZAMIENTO ". Dicha instrucción carga el valor de CS' en el registro de segmento CS. Inmediatamente que se modifica un valor de cualquiera de los seis registros de segmento (CS, SS, DS, ES, FS y GS) el procesador toma como puntero los 14 bits de más peso de dicho registro y direcciona una entrada de la Tabla de descriptores de segmentos. El contenido de dicha entrada que son los 8 bytes de un descriptor los carga en un registro caché ultrarápido de 64 bits asociado al registro de segmento modificado y a partir de ese momento el procesador toma los valores de la Base, Límite y Atributos del descriptor cargado en el registro oculto para direccionar el segmento. En el caso de ser un segmento de código el desplazamiento se carga en el puntero de instrucciones EIP y su valor se suma al de la Base para determinar la siguiente instrucción a ejecutar (ver figura 8.11). Cuando se cambie segmento de código se cambia el de pila en los que el desplazamiento lo determina el valor de ESP.

Como se ve cada registro de segmento tiene:

- Una parte visible, que se puede manejar con una instrucción que puede ser directa (MOV, POP, LDS,…) o implícita, que carga el contenido del registro CS,
- y otra parte invisible que es cargada por el propio procesador y que a la que no se puede acceder .

La carga de los registros de segmento de datos se producen cuando se ejecute en una instrucción un operando ubicado en un nuevo segmento de datos.

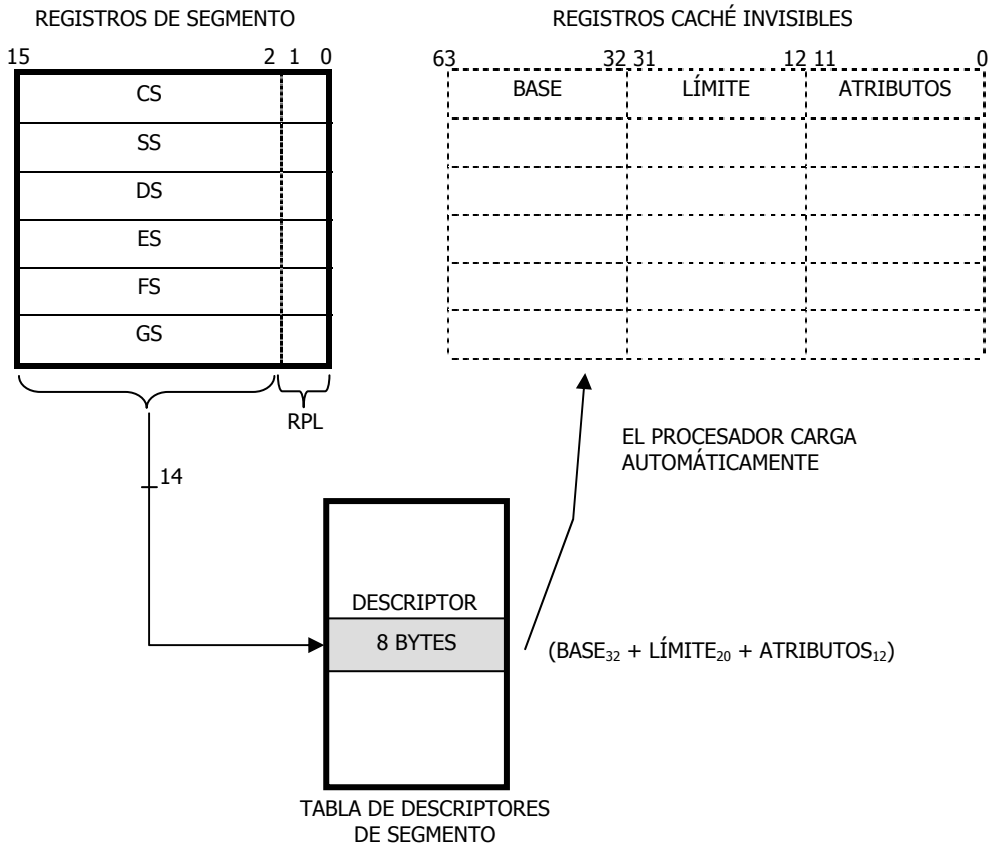


Figura 8.11. Con los 14 bits de más peso del registro de segmento modificado se accede a un descriptor de la Tabla y su contenido (Base, Límite y Atributo) lo carga automáticamente el procesador en el registro caché asociado.

8.7. - TABLAS DE DESCRIPTORES.

Al entrar al Modo Protegido, deben estar residiendo en la memoria principal las tablas de descriptores, que contienen las referencias precisas para los segmentos que va a usar el procesador.

En los Pentium se trabaja en un ambiente multitarea, en el que hay varias tareas atendidas por la CPU, y cada una de estas tareas está formada por segmentos. Un sistema multitarea se compone de un área global, en la que residen los segmentos comunes a todas las tareas y de un área local exclusiva de cada tarea, con los segmentos propios de cada una (figura 8.12).

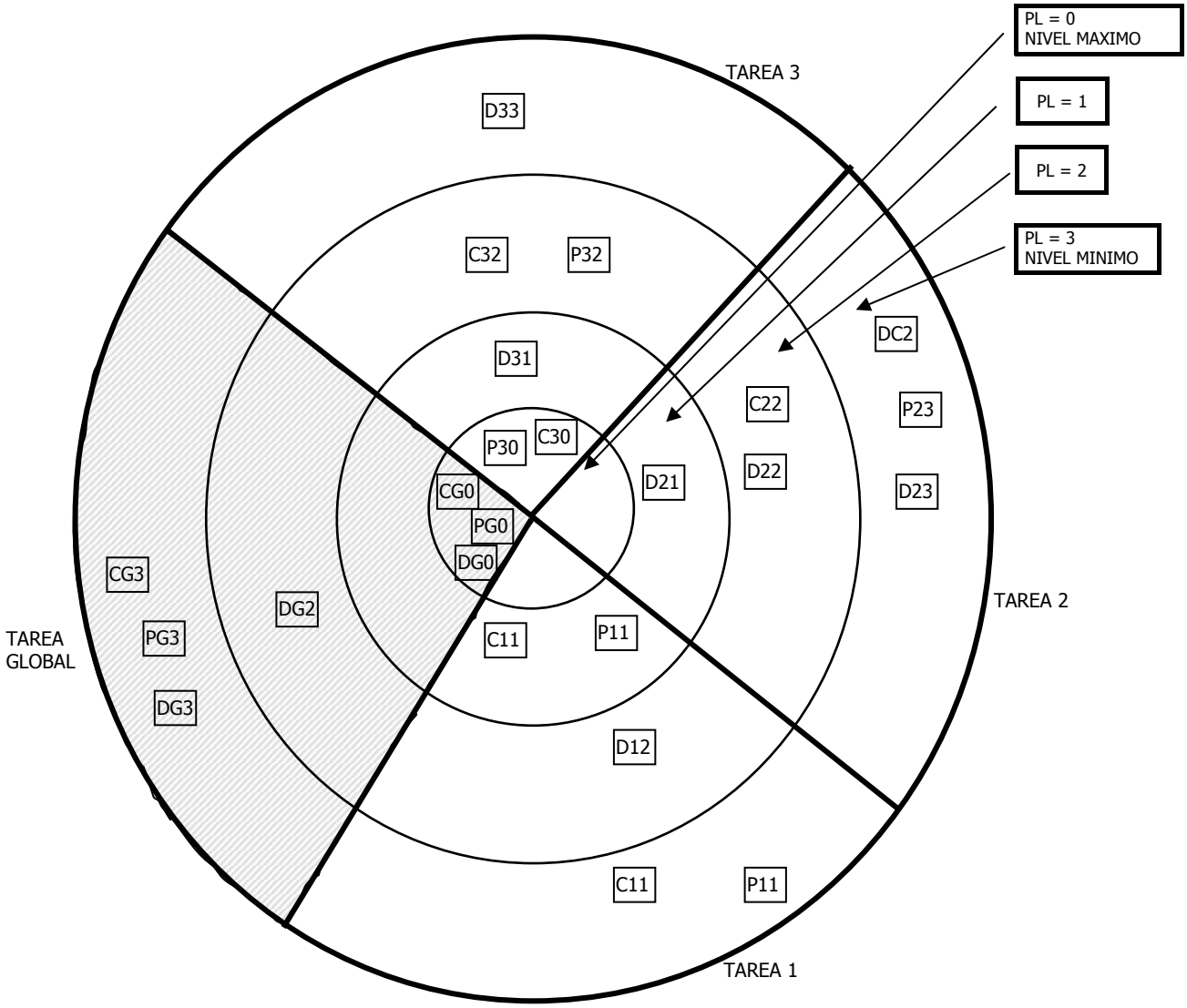


Figura 8.12. Ambiente multitarea en los Pentium.

Cada segmento del área global está definido por un descriptor, existiendo una tabla, llamada Tabla de descriptores globales y también Tabla Global de Descriptores, (GDT), que contiene todos los descriptores del área global.

Asimismo, existe una tabla para cada tarea, que recoge todos los descriptores de los segmentos de cada una de ellas. Se trata de las Tablas de descriptores locales o Tablas Locales de Descriptores, (LDT). Existirán tantas LDT como tareas soporte el sistema.

En un momento determinado el Pentium estará ejecutando una tarea concreta y tendrá activas a la GDT y a la LDT correspondiente a la tarea en curso. Dos registros internos de la CPU, manejados por el programador de sistemas, apuntan a la base de la GDT y a la base de LDT activa, denominándose GDTR y LDTR, respectivamente (figura 8.13).

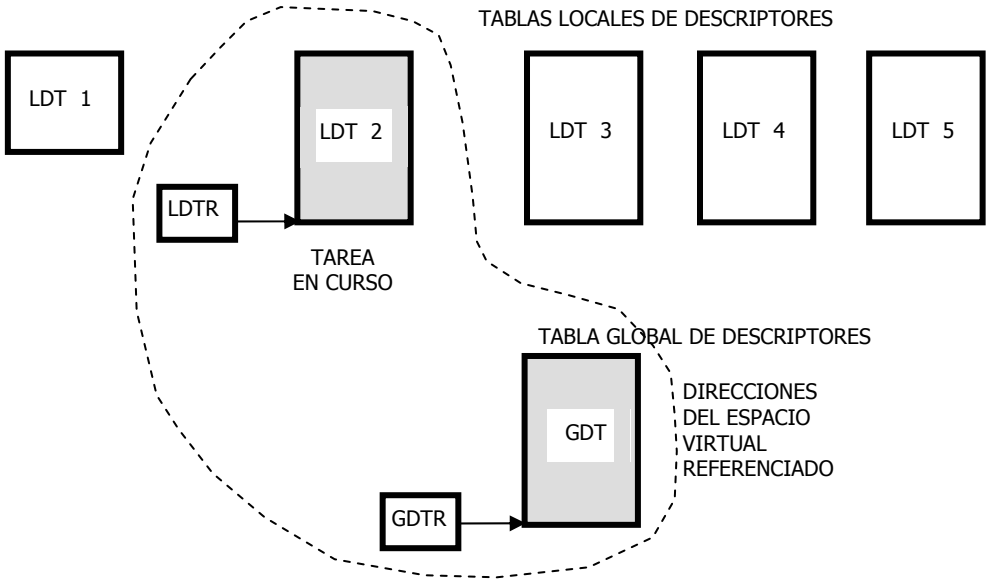


Figura 8.13. En cada instante el Pentium tiene activados a la GDT y a una LDT, la correspondiente a la tarea en curso de procesamiento.

La GDT y la LDT actúan como segmentos del sistema y sólo son accesibles por el sistema de explotación.

La estructura interna de una tabla de descriptores se muestra en la figura 8.14 pudiendo contener un máximo de 8 K descriptores de ocho bytes de tamaño cada una. La LDT es una tabla local propia de la tarea en curso y una conmutación de tarea provoca automáticamente el cambio de la LDT a través de la modificación del valor en el registro LDTR.

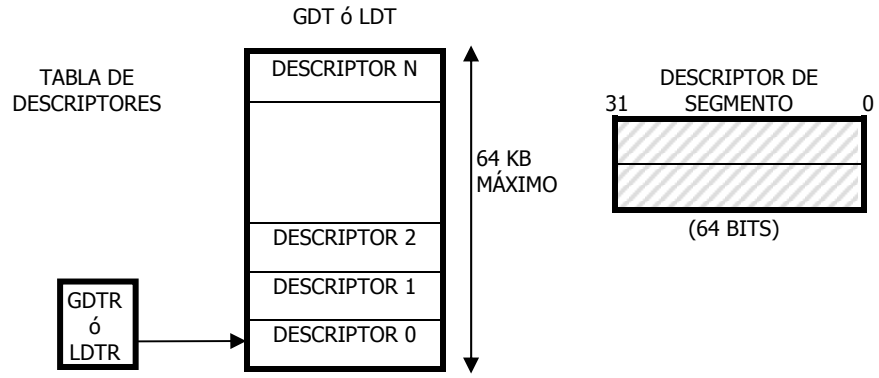


Figura 8.14. Estructura de tablas de descriptores.

Como la CPU tiene activadas dos tablas de descriptores, GDT y LDTn, hay un bit en el selector de segmento del registro de segmento, que indica a cual de ellas se refiere. Es el bit TI: Indicador de Tabla. Cuando TI=1, se selecciona la LDTn, mientras que si TI=0, se hace referencia a la GDT. Los 13 bits más significativos del selector, a los que se denomina ÍNDICE, apuntan una de las entradas a la tabla de descriptores seleccionada con TI. En realidad, el valor de ÍNDICE se multiplica por ocho para apuntar la dirección concreta de inicio del descriptor, puesto que cada uno consta de ocho bytes (figura 8.15). Las Tablas de Descriptores tienen un tamaño máximo de 64 KB y contiene un máximo de 8 K descriptores.

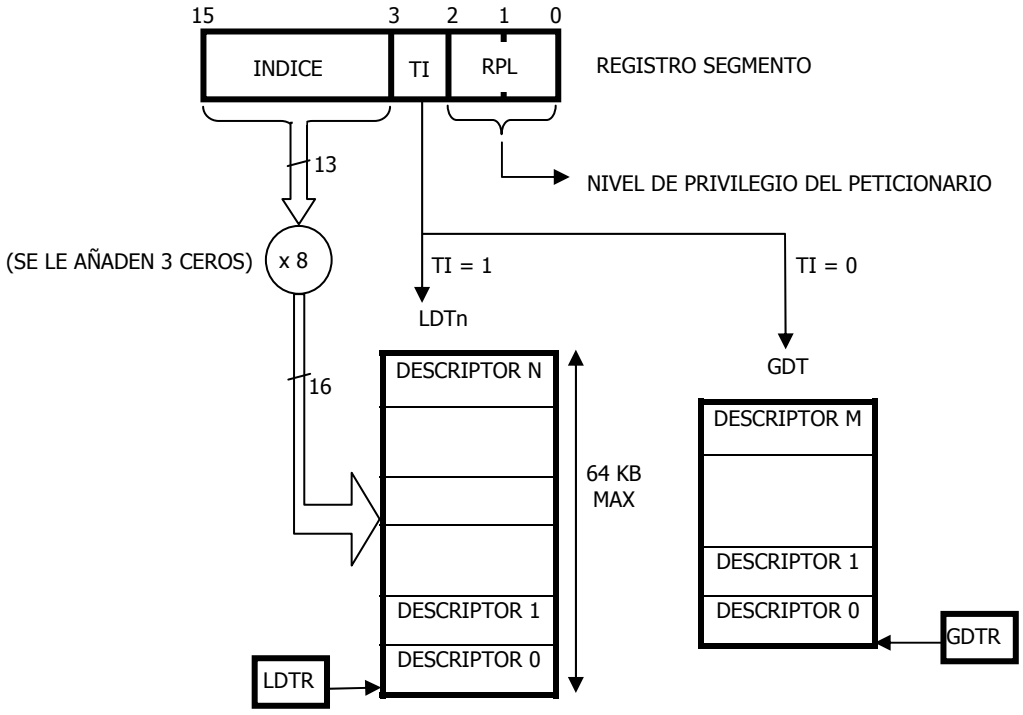


Figura 8.15. El bit TI selecciona la tabla de descriptores, mientras que el campo INDICE, multiplicado por 8, la dirección del descriptor seleccionado.

El Pentium no usa al descriptor de la entrada 0 de la GDT, que es un selector nulo, o sea, con todos los bits a 0 (INDICE =0 y TI =0). Esta circunstancia posibilita que el sistema cargue en cualquier registro de segmento un selector nulo sin que se origine una condición de excepción. De esta forma se puede borrar el contenido de un registro de segmento cargándolo con ceros.

El descriptor 0 de las LDT puede usarse, puesto que no responde a un selector nulo, ya que el bit TI=1.

A partir del selector y a través de las tablas de descriptores, la Unidad de Segmentación localiza la base del segmento a la que suma el desplazamiento para obtener la dirección lineal, que se convierte en la dirección física cuando se halla desactivada la Unidad de Paginación (figura 8.16).

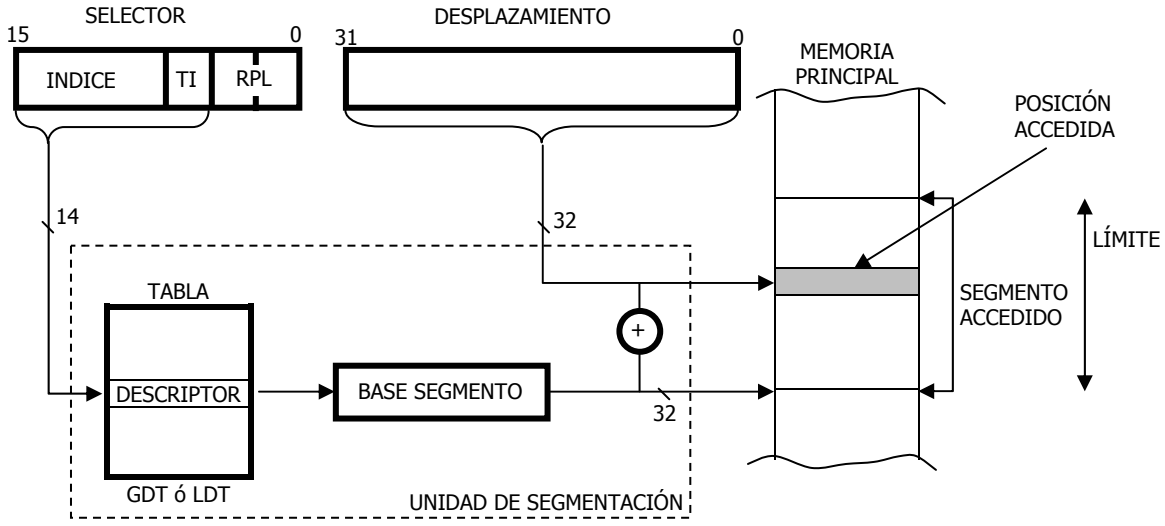


Figura 8.16. Obtención de la dirección lineal o física por parte de la Unidad de Segmentación, cuando está inhibida la Paginación

Como desplazamiento del segmento de código se utiliza EIP. ESP actúa como desplazamiento del segmento de pila. Cuando se accede a datos y se emplea como selector a uno de los registros DS, ES, FS ó GS el desplazamiento viene dado por la definición del operando y su modo de direccionamiento dentro de la instrucción a la que pertenece. En el caso más complejo, este desplazamiento puede venir formado por cuatro parámetros:

1. Un registro base.
2. Un registro desplazamiento de hasta 32 bits, incluido en la propia instrucción.
3. Un registro índice.
4. Un factor de escala (x1, x2, x4 o x8), según el número de bytes del operando.

El organigrama de la figura 8.17 muestra la obtención de la dirección física correspondiente a una posición de un segmento de datos, con un modo de direccionamiento en el que participan un registro base, un desplazamiento, un registro índice y un factor de escala.

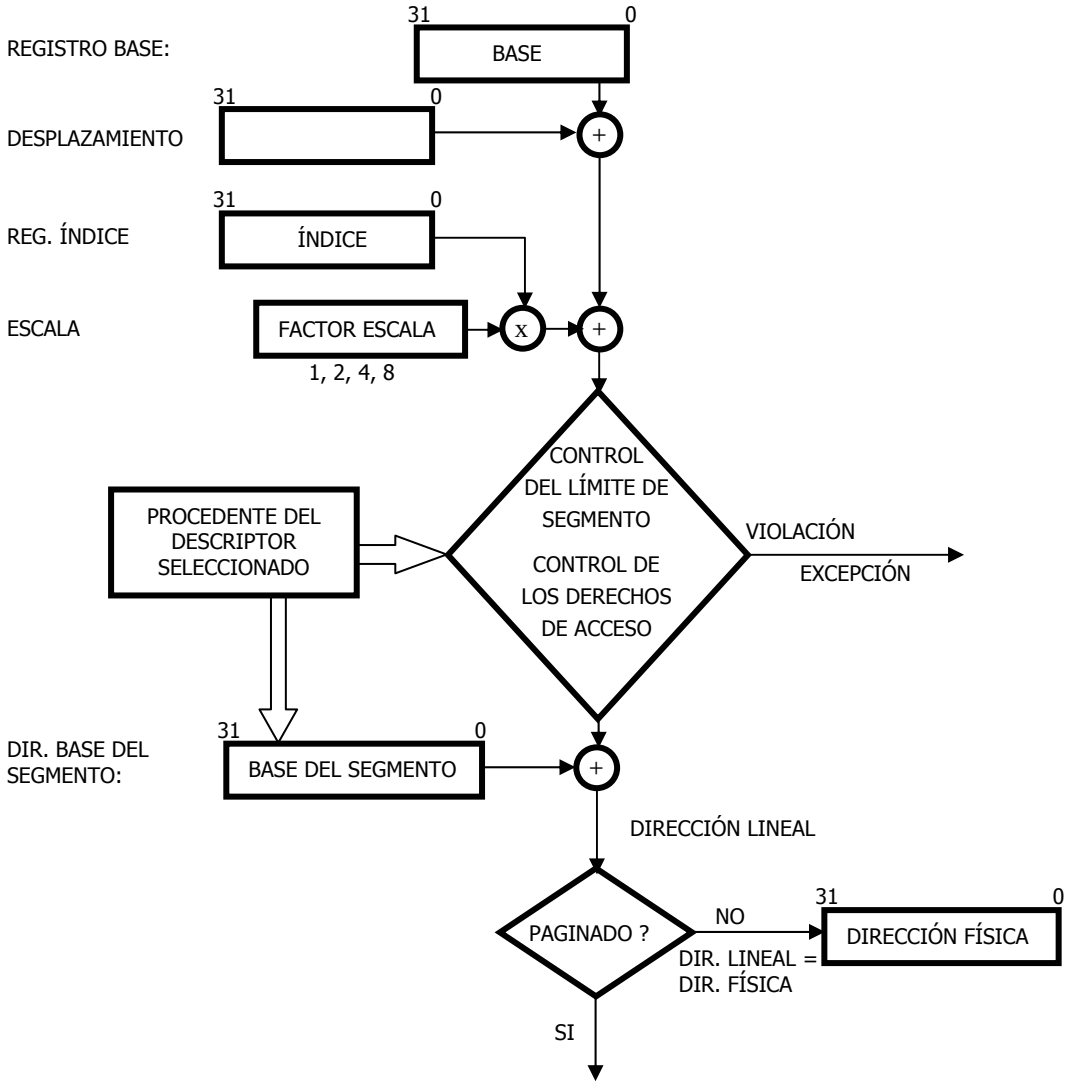


Figura 8.17. Procedimiento por el que se obtiene la dirección física de una posición en un segmento de datos, definida por un direccionamiento complejo.

8.8- EL MODELO PLANO

El mecanismo de segmentación es intrínseco al Pentium y no puede desactivarse. En aquellas aplicaciones y sistemas en los que no se use la segmentación hay un procedimiento para simular su inhabilitación, llamándose plano al modelo de memoria al que se tiene acceso en esta situación.

Se cargan todos los registros de segmento con selectores que apuntan en las tablas a descriptores caracterizados porque el valor de su base es 00000000H y el límite FFFFFFFFH. De esta manera, la CPU sólo maneja un único segmento, que abarca todo el espacio lineal, o sea, los 4 GB (figura 8.18). No alterando el valor de los registros de segmento, el procesador maneja seis segmentos que se solapan en el espacio lineal, en base a los diferentes desplazamientos, que puesto que admiten 32 bits, pueden alcanzar los 4 GB.

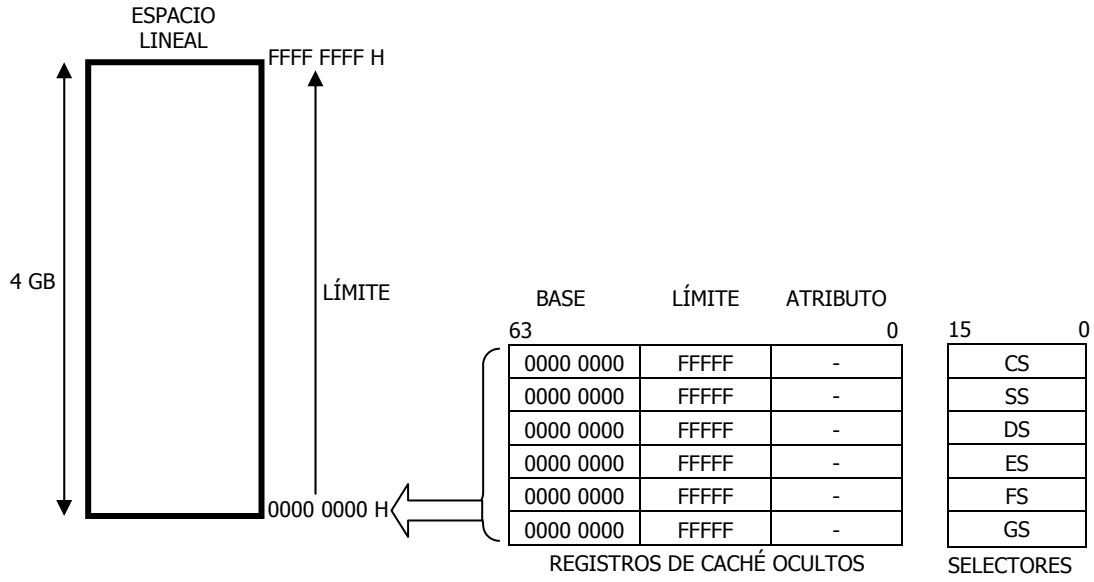


Figura 8.18. En el modelo "plano" el procesador maneja un único segmento en el que se solapan los segmentos de código, datos y pila, mediante los desplazamientos de 32 bits correspondientes a cada uno.

Usando el modelo plano se puede activar la Paginación con objeto de conseguir un entorno protegido a nivel de páginas.

EL MECANISMO DE PAGINACIÓN 9

9.1. INTRODUCCION

La paginación es un procedimiento de gestión de la memoria muy eficaz en los sistemas operativos multitarea que manejan memoria virtual. Divide y manipula los programas y los datos en trozos de tamaño fijo, llamados páginas.

A diferencia con los segmentos, las páginas no guardan relación con la estructura lógica con la que se ha construido el software.

La mayor ventaja de la paginación se obtiene en la transferencia e intercambio de elementos entre la memoria virtual y la física. El hecho de que las páginas tengan siempre el mismo tamaño facilita la ocupación de la memoria así como el rendimiento en su explotación.

Esto implica que los sistemas operativos que manejan la paginación sean muy simples, sencillos y rápidos debido a que los algoritmos de transferencia son muy simples ya que mueven cantidades de datos de igual tamaño.

El mayor inconveniente que presenta la paginación es el mal aprovechamiento de la memoria, puesto que las páginas tienen un tamaño fijo y puede que un objeto de código sea muy grande y requiere varias páginas o que un objeto de los datos ocupe menos espacio que una página, por lo tanto la paginación no se adapta a las necesidades de espacio que precisa el software.

Por otra parte, y puesto que la mayoría de los sistemas lógicos basan su flujo de procesamiento en el principio de vecindad, sólo es necesario que un reducido número de páginas de la tarea en curso esté ubicado en la memoria principal en cada momento.

El procesador Pentium siempre trabaja con segmentación y optativamente puede trabajar además con paginación.

9.2. MECANISMO DE PAGINACION

La Unidad de Paginación esta implantada en hardware dentro del Pentium.

El funcionamiento de la paginación es optativo y para su habilitación basta con poner a 1 un bit (PG) de uno de los registros de control (CR0), de los que maneja el programador de sistemas, para ello se utiliza la instrucción: MOV CR0, FFFF. Como a dicho bit solo se le puede modificar en Modo Protegido, la paginación solo opera en dicho modo.

Cuando está habilitada la paginación, se divide a cada segmento del espacio lineal creado por la Unidad de Segmentación, en páginas sucesivas de 4 KB de tamaño cada una. El Pentium también puede manejar páginas de hasta 4 MB, Luego, la Unidad de Paginación carga y distribuye, de forma aleatoria, las páginas que se precisan en cada momento, sobre el espacio de la memoria física.

Los algoritmos usados en la transferencia de bloques desde/hacia la memoria principal, son mucho más sencillos y efectivos que en la segmentación, puesto que manipular bloques de tamaño fijo y reducido, optimiza el aprovechamiento del espacio de memoria. Además, puesto que el Pentium dispone de una memoria caché de alta velocidad para guardar la traducción de direcciones lineales a físicas, dentro de la paginación, se puede acelerar extraordinariamente la velocidad de acceso.

En los procesadores Intel386 e Intel486 era necesario ejecutar una instrucción JMP después de activar la paginación para que no se ejecutaran las instrucciones que previamente ya habían sido buscadas y decodificadas. El Pentium utiliza un buffer (BTB) de predicción de bifurcaciones que evita la necesidad de utilizar instrucciones para eliminar las instrucciones que ya han sido buscadas y decodificadas como ocurría en los procesadores anteriores.

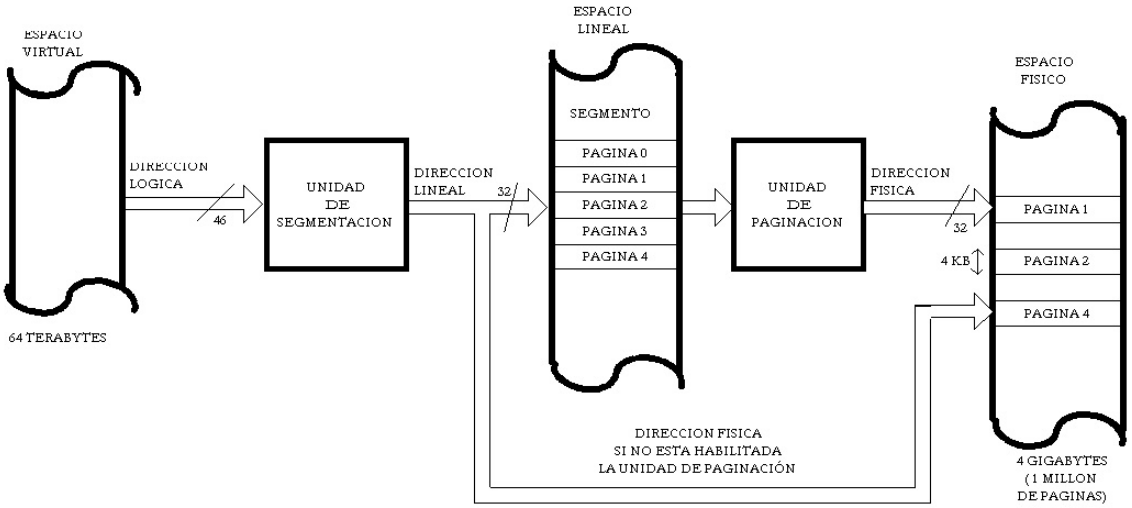


Figura 9.1 – Cuando funciona la Unidad de Paginación, los segmentos del espacio lineal se dividen en páginas y se trasladan al espacio físico solo aquellas páginas que se precisan en cada momento

De la figura 9.1 se deduce que la Unidad de Paginación traduce la dirección lineal a física. Teniendo en cuenta el espacio físico que puede alcanzar un máximo de 4 GB, la paginación lo descompone en un millón de páginas de 4KB, o sea, actúa como una gran tabla de un millón de entradas, una por página. En cada entrada se guardaría la dirección base de comienzo de la página y los atributos de la misma, algo similar a un descriptor de segmento, pero sin el campo LIMITE.

Para referenciar la base de la página bastan 20 bits, puesto que, como cada página tiene 4 KB, los 12 bits menos significativos de la dirección de 32 bits, serán siempre 0. De esta forma el descriptor de página proporciona los 20 bits de más peso de la dirección de la base, a los cuales se añaden los 12 bits de menos peso, que serán 0 si se referencia el comienzo y otro valor cuando se intenta acceder a una posición cualquiera de dicha página.

Con esta organización de la memoria, la Unidad de Paginación manejaría una tabla con un millón de entradas, conteniendo cada una la base (20 bits) y los derechos de acceso (12 bits) de cada página de la memoria principal. Luego si cada entrada consta de 32 bits de Tabla de Páginas tendría un tamaño de 1 M x 4 bytes = 4 MB (figura 9.2).

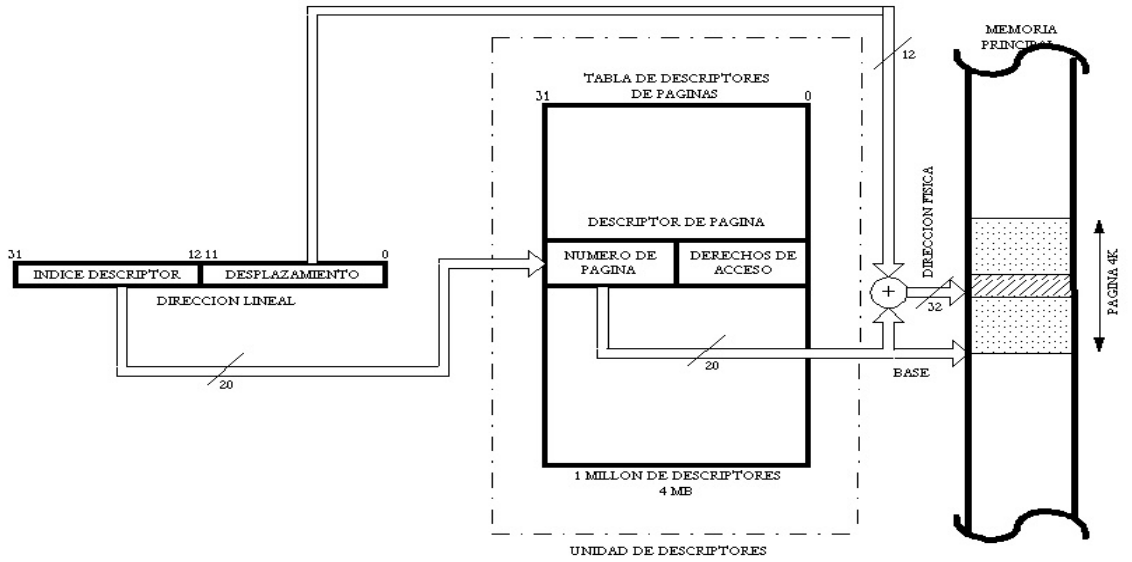


Figura 9.2 – "Aparentemente", la Unidad de Paginación se comporta como una tabla con un millón de entradas, que contienen los descriptores de las páginas en la memoria principal. La tabla traduce la dirección lineal en física.

Cada vez que la Unidad de Paginación detecta que la página no reside en la memoria principal, genera un fallo de pagina, que origina una excepción que llama a una rutina del S.O. que se encarga de trasladar dicha página desde la memoria virtual a disco hasta la memoria física o RAM.

Como la Tabla de descriptores de Páginas tiene un tamaño de 4 MB y debe residir en la memoria principal para poder ser manejada por la CPU, puesta en marcha del mecanismo de paginación hipotecaría un amplio espacio de memoria física. Para solucionar este problema, en vez de emplear una sola tabla con un millón de entradas, INTEL introdujo una traducción de direcciones a dos niveles, pero ahora se pueden producir dos fallos de página, uno por cada tabla.

En cada tarea, un primer nivel de traducción lo soporta la tabla denominada DIRECTORIO DE TABLAS DE PAGINAS, que consta de 1 K entradas de 32 bits cada una, es decir, ocupa una página de 4 KB. Esta tabla o directorio es fija para cada tarea y su base está cargada en el registro de control de la CPU, designado como CR3. Para posibilitar la paginación es imprescindible que el Directorio de Tabla de Páginas esté ubicado en la memoria principal.

El acceso a una entrada del Directorio de Tablas de Páginas se calcula sumando a la base (CR3) el valor de los 10 bits de más peso de la dirección lineal. La entrada seleccionada del Directorio contiene la dirección de la base de una página, que actúa como una segunda Tabla de Páginas, formada por 1 K entradas de 32 bits cada una. Se accede a una de estas entradas sumando a la dirección base de la Tabla de Páginas los 10 bits centrales de la dirección lineal. En la entrada seleccionada se almacena la dirección de la base de la página a acceder junto a sus atributos y para elegir, dentro de ella, la posición concreta, se suma a la base el valor de los 12 bits de menos peso de la dirección lineal. Véase la figura 9.3.

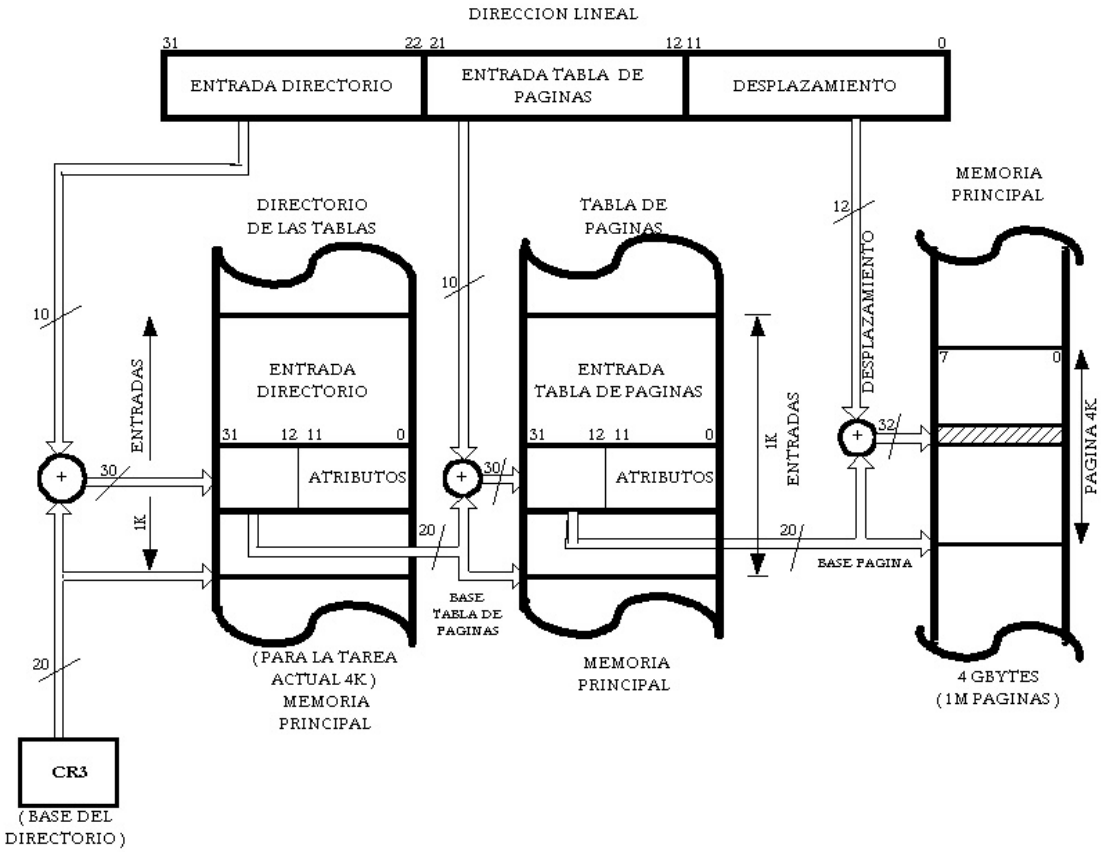


Figura 9.3 – Esquema sobre el mecanismo de paginación. El directorio de las Tablas de Páginas es propio de ada tarea y su base es apuntada por el registro de control CR3, el cual se modifica en cada conmutación de tarea.

Apréciese que, para seleccionar entradas en el Directorio de Tablas de Páginas y en las Tablas de Páginas de segundo nivel, solo se usan 10 bits, para cada caso, aunque el tamaño de ambos elementos es el de una página de 4 KB. La razón estriba en que cada entrada posee cuatro bytes y, por tanto, los dos bits de menos peso de la dirección de cada entrada serán 0, lo que equivale a multiplicar por cuatro el valor de 10 bits usado en la selección de la entrada (figura 9.4).

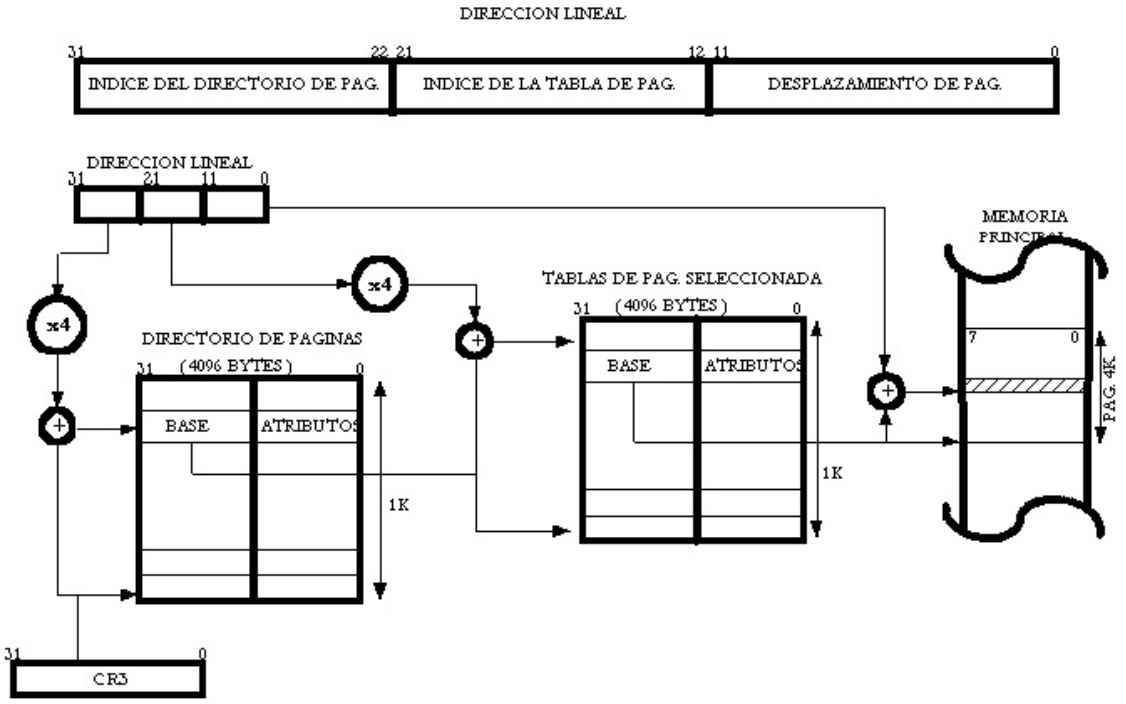


Figura 9.4 – Traducción de una dirección lineal a física.

En el caso de referenciar la dirección lineal una página de 4MB, cosa posible en el Pentium, los 10 bits de más peso direccional el directorio de páginas y los restantes 22 actúan como desplazamiento en la página.

Como las anteriores alternativas planteadas son lentas y degradan el rendimiento del computador, INTEL introdujo en el hardware, dentro del propio chip del procesador Pentium, una caché ultrarrápida de acceso por contenido (CAM) denominado TLB para optimizar el rendimiento del sistema, y que será explicado en los siguientes apartados.

Otra posible solución para el mecanismo de paginación es la Tabla de Páginas Inversa, que es utilizada por los equipos AS/400 de IBM y los RISC PowerPC (figura 9.5).

Esta técnica consiste, en que, en lugar de una entrada por cada página virtual tiene una entrada por cada marco de páginas de la memoria física asignada al proceso. Es decir la tabla inversa solo da información de las páginas guardadas en los marcos. Luego si se produce un fallo debe existir un mecanismo de traducción de la información a memoria secundaria.

Se utiliza una tabla hash para implementar la tabla. El número de página virtual se mapea a esta tabla mediante una función hashing sencilla. Recordamos que una función hash hace corresponder números que se encuentran en un rango 0-m a números que se encuentran en el rango 0-n siendo m>n. La salida de esta función se utiliza como índice de la tabla hash.

Es posible que más de una entrada apunte a la misma posición de la memoria principal. Cuando una posición está ocupada se crea un puntero a otra posición. Las cadenas que se crean con las técnicas hashing son habitualmente cortas. En estos casos para saber qué marco contiene una página hay que recorrer todos los enlaces hasta el final. Con lo que se demuestra que las cadenas que se forman son pequeñas. La tabla hash o tabla inversa, contiene la siguiente información:

- Página: indica la página almacenada en esa dirección.

- ETP: entrada da tabla de página que contiene el número de marco. Esta entrada de tabla es equivalente a la ya vista y contiene información similar.
- Puntero: a otra dirección de la tabla.

En las tablas de páginas normales se tienen tantas entradas como páginas virtuales luego pueden crecer enormemente. En la tabla de páginas inversa hay tantas entradas como marcos de página asignados al proceso el sistema operativo. Su principal ventaja es que el tamaño de la tabla es constante, independientemente del tamaño del proceso. Como inconveniente tiene que la paginación es un poco más compleja pues hay que aplicar previamente la función hash.

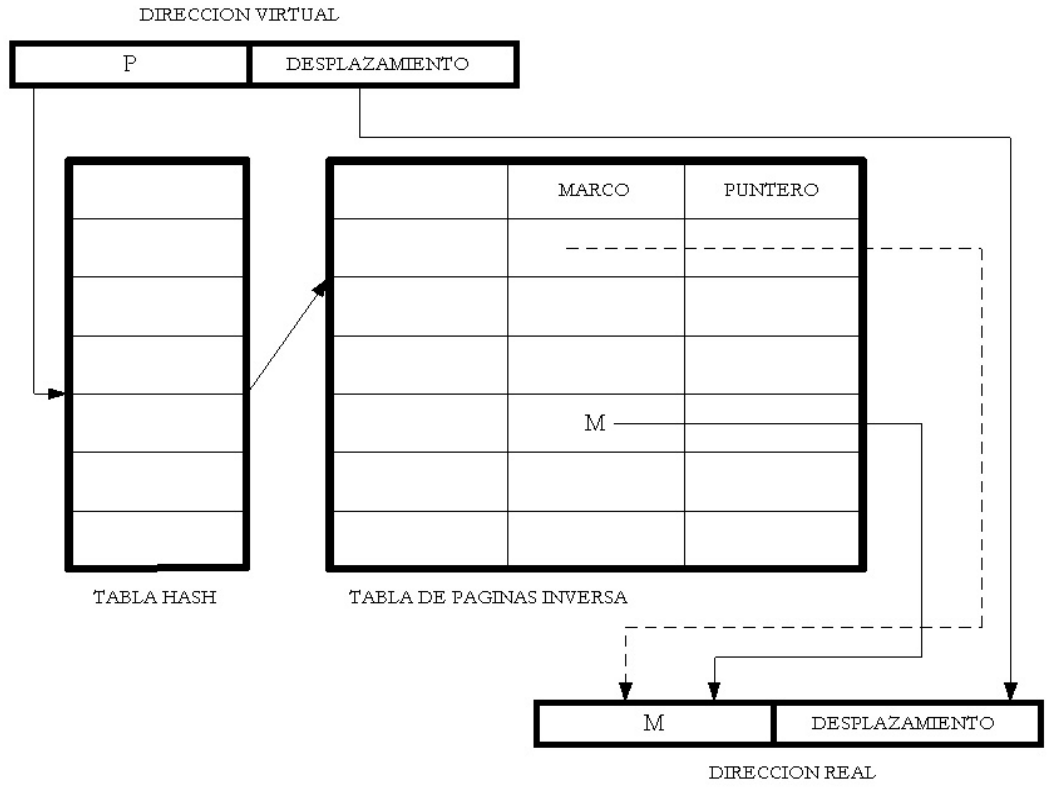


Figura 9.5. Paginación mediante la Tabla de Páginas Inversa.

9.3. FORMATO DE LAS ENTRADAS AL DIRECTORIO Y A LAS TABLAS DE PÁGINAS

El formato de una entrada al Directorio es similar al de una entrada a una Tabla de Páginas. Ambas constan de 32 bits de los cuales los 20 de más peso proporcionan los 20 bits más significativos de la dirección de la base de la página de la siguiente estructura a la que hacen referencia. Los 12 bits de menos peso de la dirección de la base son ceros y por eso en la entrada se utilizan para definir los atributos.

En cuanto a los 12 bits restantes de las entradas que se comentan, sirven para definir los atributos tal como se refleja en la figura 9.6.

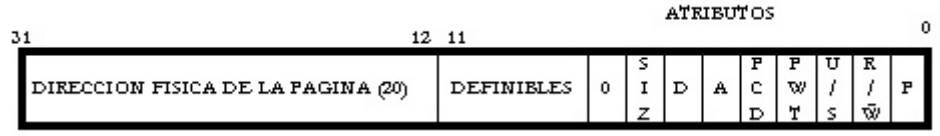


Figura 9.6. Estructura de una entrada del Directorio o de la Tabla de Páginas. Hay tres bits a disposición del S.O. que pueden usarse para guardar información auxiliar sobre la página.

A continuación se describe la función de los bits significativos del campo de los atributos de una entrada del Directorio o de las Tablas de Páginas.

- **P** (BIT DE PRESENCIA): Es el bit de menos peso de una entrada y, si está a 1, indica que la página está cargada en la memoria física y, en consecuencia, los restantes bits de dicha entrada son operativos.
  Si P = 0, significa que la página no está cargada en la memoria física. La CPU genera una excepción de fallo de página, que activa una rutina del S.O. que trae la página desde la unidad de almacenamiento externo (disco) a la RAM. Una vez cargada la página en la memoria física, el S.O. pone P = 1, escribe los 20 bits de más peso de la entrada con el valor correspondiente a la dirección de la base de la página y actualiza los restantes bits de los atributos. Posteriormente, la CPU vuelve a acceder a la entrada y como P = 1, procede a la lectura o escritura de la posición accedida.
- **A** (BIT ACCEDIDO): Se pone a 1 cada vez que se accede a dicha página. Este bit lo maneja el S.O. para llevar la cuenta del número de accesos que tiene cada página. Cada poco tiempo el S.O. lee este bit y, si vale 1, lo pasa a 0 e incrementa el contador que tiene asociada la página. El número de accesos es empleado por el algoritmo LRU, que sirve para eliminar de la memoria la página que menos se haya usado recientemente.
- **SIZ**: Este bit sólo existe en CR3 y si esta activado el Pentium trabajará con páginas de tamaño 4MB. En las entradas del directorio de p´sginas y de las tablas de páginas este bit vale 0 ya que no tiene ningún valor. Los anteriores procesadores al Pentium sólo podían manejar páginas de 4 KB.
- **D** (BIT SUCIO): Indica si se ha escrito en la página. Si D = 1, significa que se ha escrito y, cuando se quiera eliminar esta página de la memoria principal, será preciso salvarla previamente en la memoria virtual para mantenerla actualizada.
  Si D = 0, se puede sobrescribir en esta página cuando se decide sustituirla por otra, ya que no ha sido modificada durante su estancia en la memoria principal.

- **R/W** (BIT DE LECTURA/ESCRITURA): Si R/W = 1, la página es accesible en lectura y escritura, mientras que si vale cero, sólo se puede leer.
- **U/S** (BIT USUARIO/SUPERVISOR): Indica el nivel de privilegio correspondiente a dicha página. La paginación también aplica reglas de protección, aunque menos potentes al existir solo dos niveles de privilegio: Nivel Supervisor, equivalente al nivel 0 de la segmentación y Nivel Usuario, equivalente al nivel 3.

  Si U/S = 1, la página tiene nivel Supervisor y, por tanto, el mayor grado de confianza y seguridad. En dichas páginas pueden existir todo tipo de instrucciones, mientras que las páginas con el nivel de Usuario (U/S = 0) tienen ciertas restricciones.
  Al activarse la paginación, cada acceso a memoria soporta tres controles por parte del sistema de protección:

  1. A nivel de descriptor de segmento.
  2. A nivel de entrada al Directorio.
  3. A nivel de entrada a la Tabla de Páginas.

  El nivel resultante en la paginación se considera como el más restrictivo entre el correspondiente a la entrada del Directorio y el de la entrada de la Tabla de Páginas.

- **PCD** (BIT DE ACTIVACION DE LA CACHE): Indica si la página es cacheable, si se puede meter o no en la memoria caché
- **PWT** (BIT DE ESCRITURA OBLIGADA): Indica que la página además de ser cacheable funciona en modo de Escritura Obligada.
- **Definibles**: Son tres bits a disposición del S.O. que pueden usarse para guardar información auxiliar sobre la página.

Cualquier fallo en la protección que detecta la Unidad de Paginación origina una excepción, que, entre otras cosas, guarda el valor de la dirección lineal que lo ha provocado en uno de los registros de control (CR2).

9.4. TABLA DE TRADUCCION DE DIRECCIONES LINEALES (TLB)

El mecanismo de traducción de direcciones en la paginación es bastante lento porque debe realizar dos accesos a memoria, uno al Directorio y otro a la Tabla de Páginas, para localizar la página y, en un tercer acceso, efectuar la lectura o escritura solicitada por la CPU. Además, en dichos accesos se debe realizar la suma del valor de la base y el desplazamiento en la página referenciada, con lo cual la CPU tarda cierto tiempo en realizar la traducción de una dirección lineal a física.

Para acelerar el proceso de traducción, INTEL, introdujo dentro del propio chip una pequeña memoria caché ultrarrápida, que se denomina TLB: "TRANSLATION LOOKSI DE BUFFER".

La TLB guarda la traducción de las direcciones lineales a físicas, correspondientes a las 32 últimas páginas que se han manejado. Se trata de un mecanismo parecido al empleado en la segmentación, donde a cada registro segmento se asociaba un registro caché invisible, que guardaba el valor del descriptor con el fin de evitar tener que acceder a las tablas de descriptores en cada acceso a memoria.

Con la paginación en marcha, la CPU consulta a la TLB en cada acceso a memoria y, en el caso de que la página referenciada se halle almacenada junto a su traducción (de dirección lineal a física), se tardan un tiempo despreciable en obtener la dirección física. Si la página no esta en la

TLB, el mecanismo de la paginación accede al Directorio y, luego, a la Tabla de Páginas adecuada, cargando el valor de la dirección física hallada en la TLB. Después, la CPU vuelve a buscar en la TLB para efectuar el acceso, con lo que se originará un retardo suplementario, que no es mucho si se compara con el ahorro que se logra si, como es de esperar, la mayor parte de los accesos se encuentran en la TLB al seguir las reglas de la vecindad de la construcción del software.

Cada vez que la dirección lineal está contenida y traducida en la TLB, se obtiene la dirección física en escasos nanosegundos.

9.5. ESTRUCTURA Y FUNCIONAMIENTO DE LA TLB

Con las 32 entradas residentes en la TLB se controlan 32 páginas, que suponen un espacio total de 128 KB de memoria, que, en muchas ocasiones, es suficiente para contener el área de trabajo de un proceso. Se ha comprobado experimentalmente que, para programas de proposito general, una TLB de 32 entradas proporciona "ACIERTO" en más del 97% de los accesos a la memoria. También hay que considerar que en cada cambio de contexto, hay que limpiar la TLB, lo que puede ser barato, pero hay que considerar un costo indirecto, pues si los cambios de contexto son muy frecuentes, la tasa de aciertos se puede reducir.

Las 32 entradas de la TLB están organizadas en cuatro grupos de ocho entradas cada uno, que operan en paralelo. La gran velocidad en este tipo de memorias se alcanza por su método de acceso, que es por contenido (CAM: Memoria de Acceso por Contenido).

Cada entrada en una CAM se compone de una etiqueta y un dato asociado. Cuando se quiere obtener una información se suministra un valor, que se compara con los campos de etiqueta de todas las posiciones. La comparación se hace en paralelo y a gran velocidad. En el caso de que el circuito comparador hardware detecte ACIERTO o PRESENCIA, es decir, que el valor suministrado a la CAM coincida con alguna etiqueta, la información asociada a la misma se obtiene como salida. En la TLB, se proporciona como entrada la dirección lineal y, si la contiene, el resultado es la dirección física correspondiente.

Si el comparador no encuentra una etiqueta igual a la información suministrada, la CAM no contiene la traducción que se busca y señaliza AUSENCIA o FALLO.

Como la TLB está estructurada en cuatro grupos de ocho entradas cada uno, la comparación con el campo etiqueta se hace de la siguiente forma: Con los bits 12, 13 y 14 de la dirección lineal a traducir, se selecciona una de las ocho entradas en los cuatro grupos, simultáneamente. Dichas entradas o etiquetas constan de 20 bits: los 16 bits de más peso de la dirección lineal, un bit de VALIDEZ y tres más de atributos (D: Sucio, U: Usuario y W: Escritura). Después, mediante cuatro comparadores que trabajan en paralelo, se comparan las cuatro entradas o etiquetas seleccionadas con los bits 15 al 31 de la dirección lineal. En caso de que algún comparador detecte PRESENCIA (HIT), significa que la dirección lineal está traducida en la TLB. Entonces la CPU procede a extraer la información ligada con la etiqueta del acierto y saca los 20 bits de más peso de la dirección física que carga en la líneas 12-31 del bus de direcciones. Añadiendo los 12 bits de menos peso de la dirección lineal a las líneas 0-11 del bus de direcciones, se obtiene la dirección física completa.

Si los circuitos comparadores señalan AUSENCIA (MISS), se pone en marcha el mecanismo de paginación y, tras acceder al Directorio y a una Tabla de Páginas, se obtiene los 20 bits de más peso de la dirección física, que se cargan en una de las entradas da la TLB. Después, la CPU reintenta el acceso por la TLB, que dará ACIERTO, procediendo a su acceso (figura 9.6).

Cada vez que se modifican las Tablas de Páginas al cambiar de Directorio como consecuencia de modificarse el contenido del registro CR3. Hay que borrar la TLB, puesto que el 386 no hace esta operación automáticamente. El procesador asume la traducción de dirección lineal a física, sin considerar las conmutaciones de tarea. El sistema de explotación debe encargarse de inicializar las tablas de paginación, soportar las rutinas ante fallos de páginas y reinicializar la TLB después de toda modificación de las tablas de páginas.

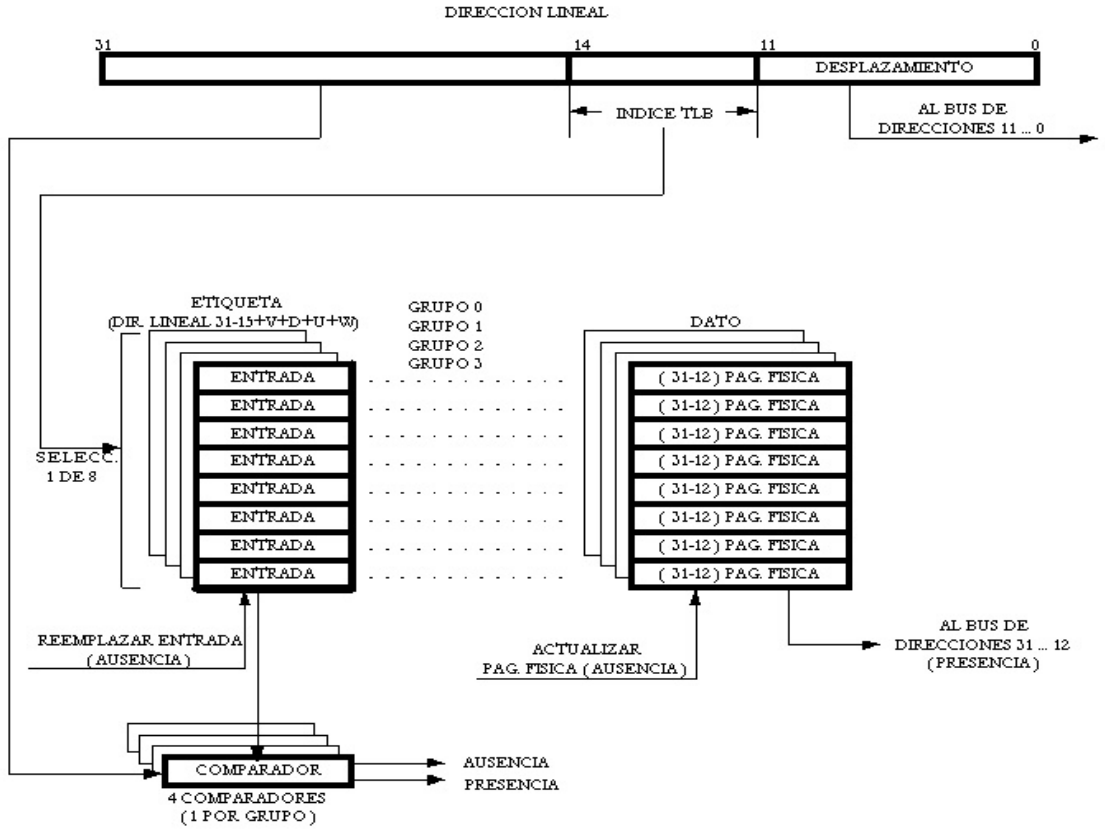


Figura 9.7 – Organización y esquema simplificado sobre el comportamiento de la TLB.

El Pentium tiene dos TLB independientes, una para la caché de instrucciones y otra para la de datos. La caché de datos tiene 8 entradas exclusivas para páginas de 4MB y 64 para las de 4 KB.

Las TLB son invisibles para todos los programas con excepción de los del Sistema Operativo con nivel de prioridad PL = 0.

El programador del Sistema Operativo debe invalidar las tablas de TLB en cuanto se produzcan cambios en las entradas de las mismas. Para realizar dicha invalidación puede usar una instrucción MOV para cargar CR3 o provocar una conmutación de tareas. Esto se debe a que el programador del sistema operativo posee dos registros de prueba de la TLB (TR6 y TR7, que se explicarán en los siguientes capítulos), con los que puede leer y escribir el contenido de una entrada de la TLB.

Existe una instrucción en el Pentium, INVLPG, que permite invalidar una entrada concreta de la TLB, generando una dirección virtual a partir del operando dado e invalidando la correspondiente entrada de la caché de la tabla de páginas, la TLB.

MECANISMOS DE PROTECCIÓN 10

10.1. ASPECTOS GENERALES

10.1.1 Necesidad de protección

Cuando se trabaja en Modo Real, el Pentium atiende a una sola tarea y el programa de instrucciones, confeccionado por el diseñador del software, se encarga de la problemática originada por el desarrollo de la aplicación.

En cambio, en Modo Protegido, el procesador atiende a varias tareas simultáneamente. El Pentium dispone de un hardware auxiliar, integrado en el chip, que se encarga de comprobar el cumplimiento de unas reglas que conforman el llamado mecanismo de protección y así permitir la ejecución de todas las tareas sin interferencias.

El objetivo fundamental del mecanismo de protección es caracterizar y defender las funciones vitales del sistema de explotación ante las posibles intrusiones procedentes de las aplicaciones, pero sin excluir una comunicación controlada.

En la construcción de un sistema lógico existen programas que son notablemente seguros por estar muy experimentados y probados; tienen un nivel de seguridad muy alto es decir, un elevado nivel de privilegio (PL). También existen programas, como los de aplicación del usuario, que son poco fiables al estar en la fase de evaluación y depuración; consecuentemente su nivel de privilegio es bajo.

Se debe evitar que programas poco seguros accedan a los datos o al código situado en niveles de privilegio superiores, porque podrían ocasionar la corrupción de estos últimos rebajando su nivel de privilegio al del programa que les accedió.

El mecanismo de protección hardware persigue la integridad del sistema, y las funciones vitales, sobre todo las intrusiones no permitidas y las comunicaciones entre las tareas.

Cuando el mecanismo de protección detecta una violación de las reglas escritas en el silicio, genera una excepción, deteniendo el procesamiento normal de la CPU. Como podría suceder al ejecutar una instrucción de división en la que el valor del divisor sea cero.

El núcleo del S.O. posee el grado de seguridad máximo, también son muy seguros los módulos de los servicios del S.O. Por el contrario, son poco fiables los programas de aplicación que tienen un notable riesgo de generar errores y fallos, además existen programas que necesitan de un nivel de seguridad intermedio.

Como se muestra en la figura 10.1, en el caso de la segmentación, se distinguen cuatro niveles de privilegio, numerados del 0 al 3 (el 0 el más privilegiado o de nivel jerárquico superior). Hay que tener en cuenta que el nivel numérico es inverso al jerárquico. Por ejemplo, un segmento con PL =2 es jerárquicamente mayor que otro con PL = 3, pero numéricamente es menor. Al comparar niveles de privilegio en la segmentación, se debe precisar si se hace referencia a su valor numérico o al jerárquico.

En el caso de la paginación, solamente se distinguen dos niveles de privilegio: el del supervisor, que es el que proporciona la máxima seguridad, y el del usuario, en el que el grado de seguridad es mínimo.

En la siguiente tabla se refleja la relación entre los niveles de protección en la segmentación y en la paginación:

SEGMENTACIÓN	PAGINACIÓN
PL = 0 Segmentos y programas del núcleo del S.O. : Seguridad alta	**Supervisor** Nivel máximo
PL = 1 Programas que necesitan seguridad media	
PL = 2 Programas que necsitan seguriadad media	
PL = 3 Programas de usuario : Seguridad baja.	**Usuario** Nivel mínimo

Figura 10.1 -- Niveles de protección en la segmentación y en la paginación.

La Unidad de Segmentación evalúa el cumplimiento de las reglas de acceso y manejo de los segmentos en primer lugar, y si está habilitada la paginación, son examinadas las reglas que afectan a las páginas en la Unidad de Paginación seguidamente.

En la figura 10.2, se incluye un ejemplo de un sistema que utiliza los cuatro niveles de privilegio. Maneja dos tareas, un espacio global compartido, existiendo segmentos de datos y de código en todos los niveles de privilegio. Hay que tener en cuenta que, siempre que exista un segmento de código debe haber uno de pila.

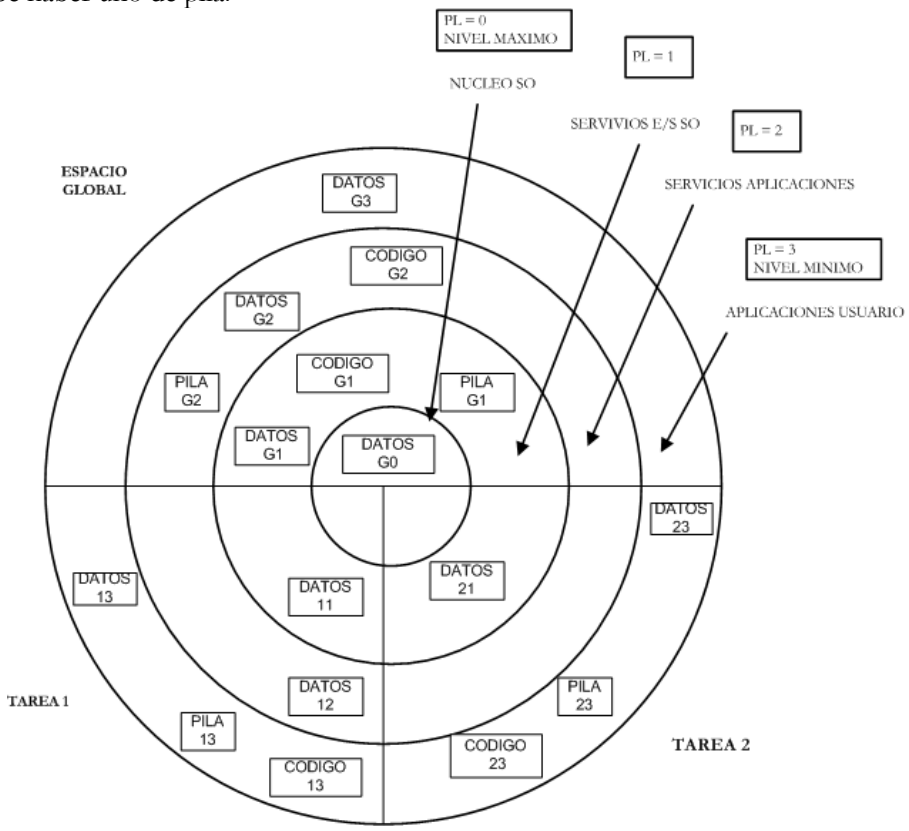


Figura 10.2 – Esquema de distribución de los segmentos que conforman un sistema lógico, que utiliza los cuatro niveles de privilegio.

En la figura 10.3, puede observarse que todos los segmentos de las tareas tienen asignado un nivel de privilegio. El campo DPL del descriptor que referencia al segmento, es el que indica dicho nivel de privilegio.

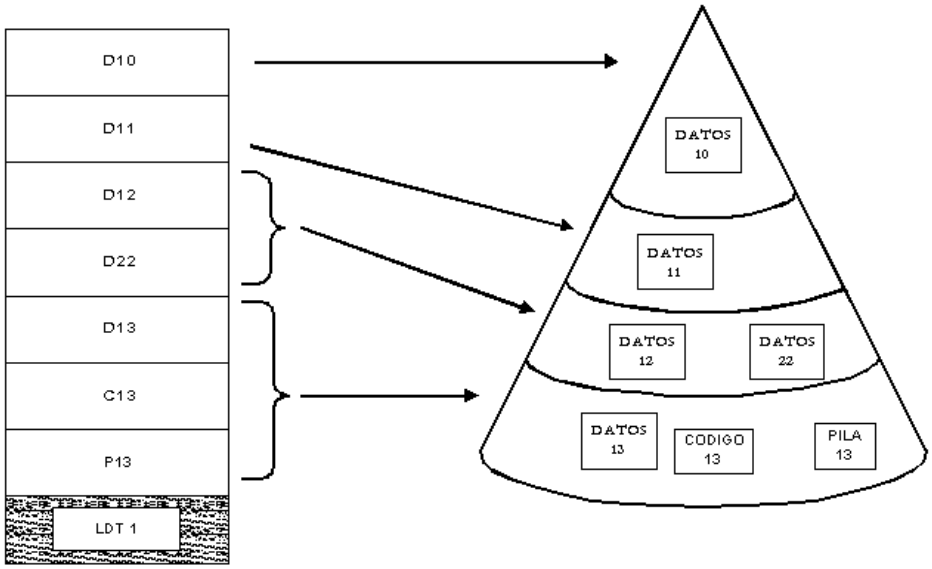


Figura 10.3 -- Cada segmento de tarea tiene asignado un nivel de privilegio, indicado en el campo DPL del descriptor que referencia.

El mecanismo ideal de protección es el proporcionado conjuntamente por la segmentación y la paginación: segmentación para la protección y paginación para manejar la memoria.

Como resultado de esta combinación ciertas páginas pueden ser escritas (las páginas son una parte del segmento) y otras leídas.

El mecanismo de protección del Pentium cubre los siguientes niveles:

1. Protección de tareas
2. Protección de los segmentos
3. Protección de las páginas
4. Protección de las instrucciones

10.1.2. Activación y desactivación de la protección de segmentos y páginas

Para que el procesador cambie a Modo Protegido y éste active el mecanismo de protección de segmentos, se debe activar el flag EP (enable protection) del registro de control CR0.

No existe ningún bit de control para desactivar este mecanismo de protección estando en Modo Protegido, pero asignando el nivel de protección 0 (mayor privilegio) a todos los segmentos y descriptores de segmento se pueden desactivar las reglas de protección de segmentos basadas en los niveles de privilegio. Así se deshabilitan las barreras de protección del nivel de privilegio entre los segmentos, pero todavía siguen activas las reglas de protección del límite y del acceso permitido a lectura/escritura.

Al activar la paginación, poniendo el flag PG del registro CR0 a 1, se activa automáticamente la protección en la paginación.

Si se quiere desactivar el mecanismo de protección en la paginación, se debe poner el flag WP del registro CR0 a 0. Posteriormente, se ponen a 1 los flags R/W (read/write) y U/S (usuario/supervisor) en el Directorio de Páginas y en la Tabla de Páginas. Así las páginas pasan a ser escribibles y pertenecientes al nivel Supervisor, quedando desactivado el mecanismo de protección en la paginación.

10.1.3. Campos y flags usados para la protección entre segmentos y páginas

El mecanismo de protección debe tener en cuenta ciertos campos y señalizadores para controlar el acceso a las páginas y segmentos.

A continuación, se describen los campos necesarios para realizar dicho control.

1. **Flag tipo de segmento (S)**: bit 12 en la segunda doble palabra de un descriptor de segmento. Determina si el descriptor de segmento describe un segmento del sistema o un segmento normal de código o datos.
2. **Campo TIPO**: bits del 8 al 11 en la segunda doble palabra de un descriptor de segmento. Determina si el segmento es de código, de datos o del sistema.
3. **Campo del límite**: bits del 0 al 15 de la primera doble palabra y bits del 16 al 19 de la segunda doble palabra del descriptor de segmento. Determina el tamaño del segmento, estando relacionado con el flag G y el flag E (para segmentos de datos).
4. **Flag G**: bit 32 en la segunda doble palabra del descriptor de segmento. Define el tamaño del segmento, estando relacionado con el límite del segmento y el flag E (para segmentos de datos).
5. **Flag E**: bit 10 en la segunda doble palabra del descriptor de segmento. Define el tamaño del segmento, estando relacionado con el límite del segmento y el flag G.
   Si está a 1 será un segmento de código (ejecutable o de sólo lectura), sin embargo si está a 0 se trata de un segmento de datos.
   Junto con el bit G se puede saber de que tamaño se trata si de 4MB ó 4KB y con el campo límite se observa cual es el límite de dicho segmento.
6. **Campo descriptor del nivel de privilegio (DPL)**: bits 13 y 14 en la segunda doble palabra del descriptor de segmento. Determina el nivel de privilegio del segmento.
7. **Campo del nivel de privilegio del peticionario (RPL)**: bits 0 y 1 de cualquier selector de segmento.
8. **Campo del nivel de privilegio de la tarea en curso (CPL)**: bits 0 y 1 del registro de segmento CS. Se refiere al procedimiento o programa que se está ejecutando en ese momento.
9. **Bit de Tamaño (SIZ)**: Bit 7 de una entrada directa del Directorio de Páginas o de la Tabla de Páginas. Indica si se trata de páginas de 4KB si es 0 ó sin embargo si se tarta de páginas de 4MB si es 1. No se debe olvidar que el Pentium acepta páginas de 4MB.
10. **Bit Sucio (D):** Bit 6 de una entrada directa del Directorio de Páginas o de la Tabla de Páginas. Si D=1 se ha escrito a la página. Sirve para avisar al Sistema Operativo que antes de machacar la página hay que actualizarla en la Memoria Virtual.

11. **Bit A (Accedido):** Bit 5 de una entrada directa del Directorio de Páginas o de la Tabla de Páginas. El Sistema Operativo pone un contador para cada página. Cada vez que se accede a esa página el bit A se pone automáticamente a 1. El Sistema Operativo cada cierto tiempo mira los bits A incrementando los respectivos contadores. Al final cuando se llena la memoria con páginas el S.O sustituye el que menos tiene en su contador.
12. **Bit PCD:** Bit 4 de una entrada directa del Directorio de Páginas o de la Tabla de Páginas. Indica si está a 1 que se trata e una página cacheable (se puede meter en la caché).
13. **Bit PWT:** Bit 3 de una entrada directa del Directorio de Páginas o de la Tabla de Páginas. Indica si está a 1 que la página es de escritura obligada y cacheable.
14. **Usuario/Supervisor (U/S)**: bit 2 de una entrada directa del Directorio de Páginas o de la Tabla de Páginas. Indica el tipo de página: Usuario o Supervisor.
15. **Lectura/escritura (R/W)**: bit 1 de una entrada del Directorio de Páginas o de la Tabla de Páginas. Indica el tipo de acceso permitido a esta página: sólo lectura o lectura y escritura.
16. **Bit de Presencia (P)**: Bit 0 de una entrada directa del Directorio de Páginas o de la Tabla de Páginas. Indica si está a 1 que la página está presente en la memoria principal y si está a 0 que no lo está.

En las figuras 10.4 y 10.5 se muestran los campos de un Descriptor de Segmento y de una entrada de la Tabla de Páginas, respectivamente.

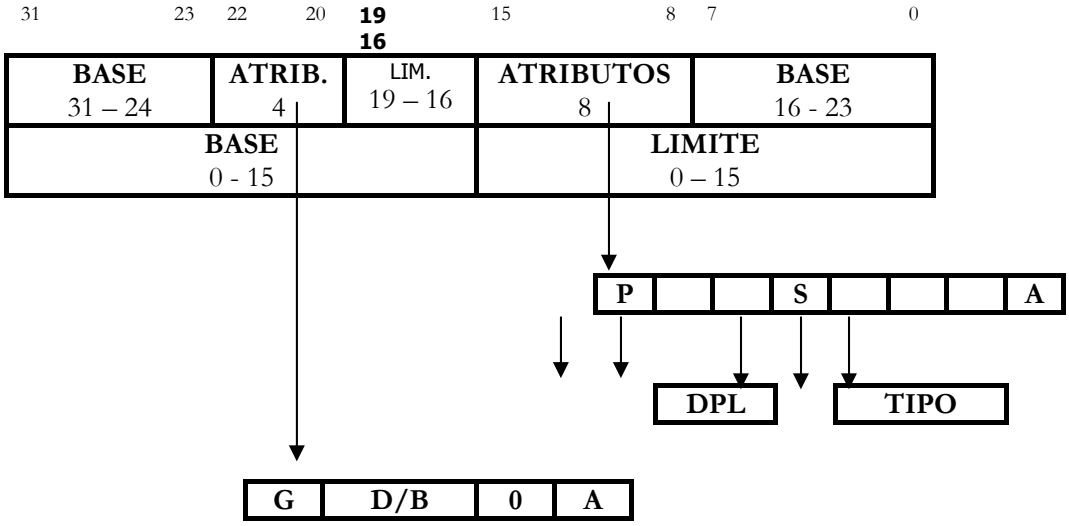


Figura 10.4 Estructura del descriptor de segmento

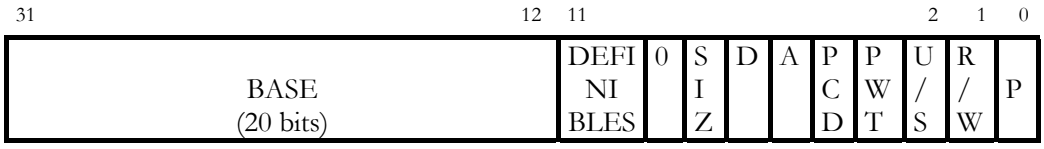


Figura 10.5 Formato de las entradas a las Tablas de Páginas

10.2. NIVELES DE PROTECCIÓN

10.2.1. Protección entre tareas

Los descriptores de segmento de la GDT, en todo momento, hacen referencia a objetos del espacio global (comunes a todas las tareas). Y los descriptores de segmento de la LDT, hacen referencia a los segmentos de la tarea en curso.

Nunca es posible que, ejecutando una tarea se realice un acceso prohibido a un segmento de otra tarea, ya que a la CPU sólo se le permite el acceso a los descriptores de la LDT activa, ésta de forma indirecta, es apuntada por el registro LDTR.

Lo que sí es posible, es acceder desde una tarea a segmentos referenciados por los descriptores de la GDT.

Para ejecutar una tarea nueva, se cambia el contenido del registro de tabla LDTR y así se referencia a la LDT de la tarea nueva y se activan los descriptores de los segmentos de la nueva tarea.

10.2.2. Protección de segmentos

Cuando se carga el contenido de un descriptor en el registro caché ultrarrápido asociado a un registro de segmento, se almacena la base (32 bits), el límite (20 bits) y los atributos (12 bits).

Cada vez que se accede a memoria, la Unidad de Segmentación es la encargada de comprobar las normas de protección y en caso de fallo, se genera una excepción.

10.2.2.1. Protección del límite

Para llevar a cabo la comprobación del límite de los segmentos se debe tener en cuenta el flag de granularidad (G). Para segmentos de datos, también depende del flag ED (dirección de expansión descendente) y del B (tamaño por defecto del puntero a pila).

Si G = 0 (granularidad de byte), el límite efectivo, medido en bytes, viene marcado por los 20 bits del límite indicado en el descriptor de segmentos. En este caso el límite está situado en el rango de 0 a 1MB (FFFFF H). En cambio, si G = 1 (granularidad de páginas de 4KB), el procesador escala el valor del campo límite con un factor de 2^{12}, el rango límite efectivo se encuentra comprendido entre 4KB (FFF H) y 4GB (FFFFFFFF H). En el caso de G = 1, los 12 bits menos significativos del desplazamiento no se comprueban con respecto al límite.

Para todos los segmentos (menos para los segmentos de datos de expansión decreciente), la última dirección a la que se puede acceder (límite efectivo), es el tamaño del segmento (en bytes) menos 1. Cada vez que se intente acceder a las siguientes direcciones en un segmento, el procesador genera una excepción de protección general:

1. Un byte más arriba del límite efectivo
2. Una palabra más arriba del límite efectivo (-1)
3. Una doble palabra más arriba del límite efectivo (-3)

4. Una cuádruple palabra más arriba del límite efectivo (-7)

Para segmentos de Pila (segmentos de datos de expansión decreciente), el límite depende del valor de B. Si B=0, el rango va de (límite efectivo + 1) a 4KB. Si B=1, el rango válido va de (límite efectivo + 1) a 4GB. Cuando el límite es 0, un segmento de expansión decreciente tiene el máximo tamaño.

Esta comprobación genera errores, que sino se detectan se puede llegar a sobrescribir el código o datos de otro segmento.

El procesador también debe comprobar los límites de las tablas de descriptores. Para evitar que los programas seleccionen un descriptor de segmento de fuera de sus tablas de descriptores, se usan los valores de los registros GDTR e IDTR (16 bits). Para evitar que se acceda fuera de las actuales LDT y TSS, se utilizan los valores del límite de segmento (32 bits) de los registros LDTR y los de tareas TR.

10.2.2.2. Protección del tipo

Para evitar el uso de un segmento o puerta de forma incorrecta se tiene en cuenta el tipo de los descriptores de segmento que viene determinado por el campo tipo y el flag S, tal y como puede observarse en la figura 10.6.

<table>
<tr><td rowspan="2">S = 1</td><td>E = 1</td><td>C</td><td>R</td><td rowspan="2">A</td></tr>
<tr><td>E = 0</td><td>ED</td><td>W</td></tr>
</table>

Figura 10.6. Formato del campo TIPO

Si S = 1, se trata de un segmento normal de código, datos o pila creado por el programa. Si S = 0, es un segmento especial creado por el programador de sistemas.

Si el bit E = 1, el segmento es ejecutable o de código. En este caso los bits restantes que componen el campo tipo son C, que indica si es ajustable (1) o no (0), y el bit R, que indica si el segmento es leíble (1) o no (0).

En caso de que E = 0, se trata de un segmento de datos no ejecutable, siendo los bits ED y W los que forman el campo tipo. ED indica si el segmento tiene o no expansión de direcciones decreciente. Si ED = 1 se trata de un segmento de pila, en caso contrario es un segmento de datos. El flan W indica si el segmento es escribible.

En cada acceso, se comprueban todas las características del segmento relativas al campo Tipo. Las diferentes excepciones generadas por el mecanismo de protección, son las siguientes:

1. Si se intenta escribir en un segmento de código (E=1)
2. Si se intenta leer un segmento de código con R=0 (prohibición de lectura)
3. Si se intenta cargar CS (registro de segmento de código) con el valor de un selector que corresponda a un descriptor con E=0. Se estaría intentando ejecutar un segmento de datos
4. Si se intenta escribir un segmento de datos con W=0 (prohibición de escritura)
5. Si se intenta cargar SS (registro del segmento de pila) con el valor de un selector, cuyo descriptor asociado esté definido como no escribible (W=0).

La información es examinada por el procesador en varias ocasiones según se opere con segmentos selectores o segmentos descriptores. A continuación se muestran algunos ejemplos de operaciones típicas en las que se comprueba el campo TIPO:

1. Cuando un segmento selector es cargado en un registro de segmento: ciertos registros de segmento sólo pueden contener ciertos tipos de descriptores, por ejemplo:

- El registro CS solo se puede cargar con el selector de un segmento de código.
- Selectores de segmento de segmentos de código que no son leíbles o de segmentos del sistema no pueden ser cargados en registros de segmentos de datos (DS, ES, FS, GS).
- Sólo selectores de segmentos de datos escribibles pueden ser cargados en registros de pila (SS).

2. Cuando un selector de segmento es cargado en LDTR o un registro de tareas:

- En LDTR sólo se puede cargar el selector de una LDT.
- En el registro de tareas TR sólo se puede cargar un selector de segmento de un TSS.

3. Cuando instrucciones acceden a segmentos cuyos descriptores ya han sido cargados en registros de segmento. Ciertos segmentos pueden ser utilizados por instrucciones sólo en ciertas predefinidas formas, por ejemplo:

- Ninguna instrucción debería escribir en un segmento ejecutable
- Ninguna instrucción debería escribir en un segmento de datos si éste no es escribible.
- Ninguna instrucción debería leer un segmento ejecutable a menos que el flag de permiso de lectura esté a 1.

4. Cuando un operando de una instrucción contiene un selector de segmento: ciertas instrucciones pueden acceder a segmentos o puertas de tan solo un tipo en particular. Por ejemplo:

- Una instrucción JMP o CALL sólo puede acceder a un descriptor de segmento de un segmento de código conforming , no conforming, puerta de llamada, puerta de tareas o TSS.
- La instrucción LLDT debe referenciar a un descriptor de segmento de una LDT
- La instrucción LTR debe referenciar a un descriptor de segmento de una TSS
- La instrucción LAR debe referenciar a un descriptor de segmento o puerta de una LDT, TSS, puerta de llamada, CS o segmento de datos.
- La instrucción LST debe referenciar a un descriptor de segmento de una LDT, TSS, segmento de código o segmento de datos.
- Las entradas IDT deben ser interrupciones, excepciones o puertas de llamada.

5. Durante una determinada operación interna. Por ejemplo:

- En una llamada (CALL) o salto (JMP) lejano, el procesador determina el tipo de transferencia de control que debe llevarse a cabo (call o jump a otro segmento de código, a través de una puerta o un cambio de tareas) comprobando el campo TIPO en el descriptor de segmento (o puerta) apuntado por el selector de segmento (o puerta) dado como operando en la CALL o JMP. Si el descriptor es para un CS o puerta de llamada, la llamada o salto a otro CS está indicado. Si el descriptor es para una TSS o puerta de tarea, es indicado un cambio de tareas.
- En una llamada o salto a través de una puerta de llamada , el procesador comprueba automáticamente que el descriptor de segmento al que apunta por medio de la puerta es un segmento de código.
- En una llamada o salto a una nueva tarea a través de la puerta de tareas, el procesador comprueba automáticamente que el descriptor de segmento apuntado por la puerta de rateas es una TSS.
- En una llamada o salto a una nueva tarea mediante una referencia directa a una TSS, el procesador comprueba automáticamente que el descriptor de segmento al que se apunta por medio de la instrucción CALL o JUMP es una TSS.
- Volviendo de una tarea anidada (iniciada por una instrucción IRET),el procesador comprueba que el campo de unión de la tarea precedente en la actual TSS apunta a una TSS.

Para los segmentos del sistema en los que el bit S = 0, los otros cuatro bits del campo tipo definen la clase de segmento de que se trata, de acuerdo con la siguiente relación de códigos:

CÓDIGO	TIPO
0	No definido
1	TSS disponible del 80286
2	LDT
3	TSS ocupado del 80286
4	Puerta de llamada del 80286
5	Puerta de tarea
6	Puerta de interrupción del 80286
7	Puerta de excepción del 80286
8	No definido
9	TSS disponible del 386
A	No definido
B	TSS ocupado del 386
C	Puerta de llamada del 386
D	No definido
E	Puerta de interrupción del 386
F	Puerta de excepción del 386

Figura 10.7. Los códigos del campo tipo cuando S es 1

Comprobación del selector del segmento nulo:

Si se intenta cargar un selector de segmento nulo en CS o SS se provoca una excepción.

Este selector sólo puede cargarse en los registros DS, ES, FS o GS, pero su acceso, una vez cargados con valor cero, genera una excepción.

Éste es un método para detectar accesos a registros de segmentos que no se usan y evitar accesos no deseados a segmentos de información.

10.2.2.3. Protección según el nivel de privilegio

El campo DPL de los atributos de un descriptor, consta de dos bits que expresan el nivel de privilegio del descriptor, o sea, del segmento al que selecciona. Para identificar el nivel de privilegio de un segmento, hay que estudiar el campo DPL de los atributos de un descriptor.

Los niveles de privilegio son usados por el procesador para evitar accesos indebidos a los mismos, como puede ser que un programa de un nivel bajo de seguridad acceda a los segmentos de niveles superiores. En caso de que esto ocurra, el procesador genera una excepción de protección general (GP).

La mezcla de segmentos de mayor nivel de seguridad con otros segmentos de menor nivel de seguridad por parte del usuario no se puede permitir ya que, si sucediera, se degradaría al nivel de privilegio del más seguro.

Siempre que se quiera acceder a un segmento debe ser a través de una instrucción ejecutada en el segmento de código en curso, su nivel de privilegio se denomina Nivel de Privilegio en Curso (CPL).

Mediante la ejecución de una instrucción, es decir, desde el segmento de código en curso de procesamiento se realiza el acceso a cualquier segmento. Es imposible acceder a un segmento desde uno que no sea de código. Para realizar la selección del segmento en curso se hace mediante el selector cargado en CS: sus 13 bits de más peso actúan como índice de la tabla GDT o LDT, pudiendo así encontrar el descriptor del segmento de código cuyo nivel de privilegio es definido por el campo DPL.

El nivel de privilegio del segmento de código en curso recibe el nombre de Nivel de Privilegio en Curso, CPL y a partir del valor del CPL se determinan las reglas de acceso a otros segmentos, ya que son las instrucciones que contiene las que determinan los mismos.

Para localizar un operando, una instrucción puede necesitar acceder a un segmento de datos, por ejemplo, MOV EAX, 5FFFH. Esta instrucción accede al dato localizado en la posición 5FFFH del segmento de datos DS (DS:5FFFH) para cargarlo en EAX.

También se puede acceder a un segmento de pila con una instrucción como PUSH EAX. Esta instrucción lo que hace es introducir un nuevo dato en la pila.

Las instrucciones del tipo JMP y CALL. La JMP: realiza un salto incondicional a otro segmento de código, por otro lado, la CALL: llama a una rutina que al terminar (RET) vuelve a la posición en la que se encontraba al hacer la CALL. Ésta última guarda en una pila parámetros del segmento de código en curso, por lo que siempre debe haber un segmento de pila en el mismo PL que haya un segmento de código.

En el caso de Pentium, siempre que haya un acceso a un segmento, se comprueba la relación entre el CPL y el DPL del segmento que será accedido.

Para controlar el acceso a los distintos tipos de segmentos existen unas reglas básicas que se describen a continuación:

1ª REGLA: ACCESO A SEGMENTOS DE CÓDIGO

Sólo se puede acceder a segmentos de código que tengan el mismo PL que el segmento de curso peticionario.

Si se quisiera acceder a otro de mayor PL, no está permitido porque éste se "rebajaría" a nuestro nivel, que es menos seguro. Si por el contrario, se desease acceder a uno de menor PL, el segmento peticionario sería el que se "rebajara" a dicho nivel inferior.

Las instrucciones que permiten estas llamadas o saltos directos son JMP, CALL y RET. El CPL del segmento de código peticionario debe ser igual al DPL del segmento a acceder directamente con la instrucción JMP,CALL o RET.

En la figura 10.8. se muestran los accesos permitidos entre segmentos de código y accesos no permitidos entre estos. Para que se puedan realizar dichos accesos los segmentos de código tienen que tener igual nivel de privilegio.

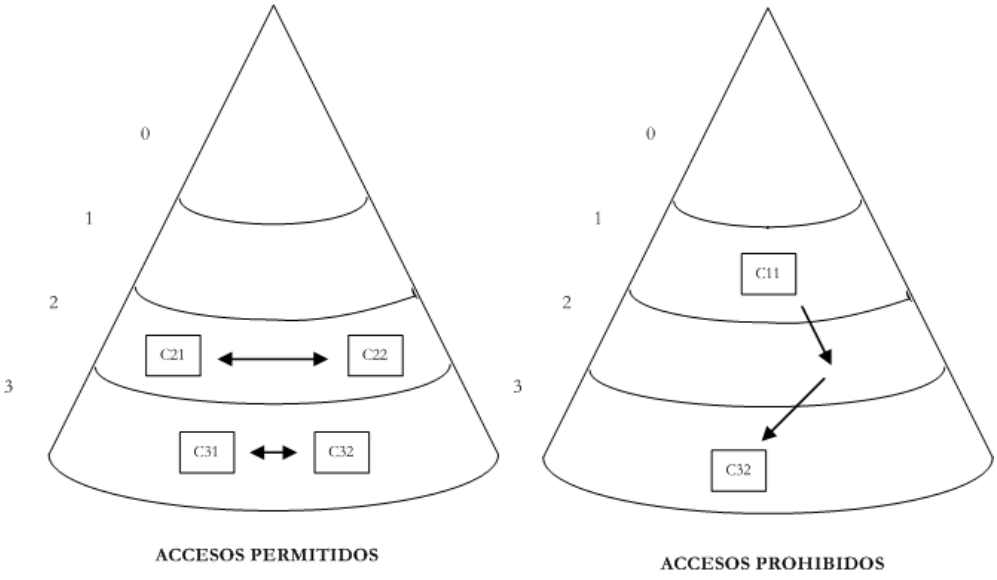


Figura 10.8. El acceso entre segmentos de código

El selector de segmento del destinatario debe ser cargado en el CS para realizar la transferencia del control del programa de un segmento de código a otro. Las comprobaciones en cuanto al límite, tipo y privilegio son examinadas por parte del procesador como parte de este proceso de carga. Si son satisfactorias, lo que implica que el registro CS es cargado, el control del programa es transferido al nuevo segmento de código y empiezan a ejecutarse las instrucciones apuntadas por el registro EIP.

Las transferencia del control del programa las llevan a cabo las instrucciones JMP, CALL, RET, INTn, IRET, SYSENTER y SYSEXIT, al igual que lo hacen los mecanismos de excepción e interrupción. Las dos últimas instrucciones son especiales y se usan para ejecutar procedimientos ó para realizar llamadas rápidas o para regresar del S.O.

El Pentium dispone de un recurso llamado Puerta de Llamada creada por el programador de sistema ya que es normal que desde un segmento de código de bajo PL se quiera acceder a rutinas del S.O. (PL superior).

2ª REGLA DE ACCESO A SEGMENTOS DE DATOS

Sólo está permitido el acceso desde segmentos de código con un PL a otros de datos de igual o menor PL. Para ello se usan instrucciones tipo MOV o similares.

Un segmento de datos se puede leer y escribir. Para evitar corromper la seguridad de los segmentos de datos se usa esta regla, ya que obliga a ser igual o mayor el nivel de privilegio del segmento de código que les intenta acceder. La escritura se realiza con la seguridad correspondiente al segmento de código. Se produciría una degradación del nivel de privilegio del segmento de datos si se accediese desde un segmento de código con menor PL (Figura 10.9).

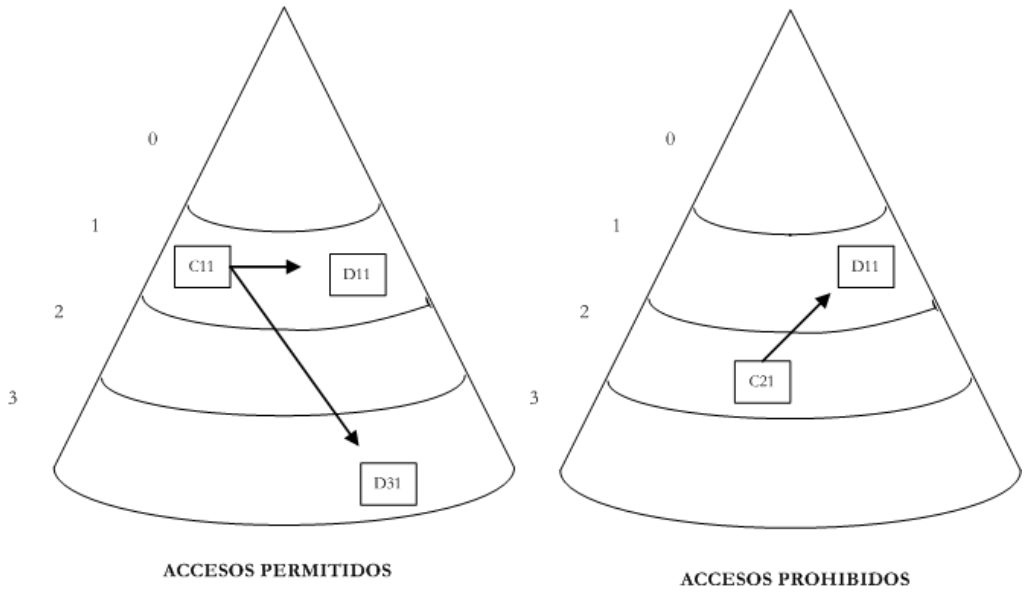


Figura 10.9. El acceso entre segmentos de datos

A través de una puerta de llamada se puede aumentar el PL de un segmento de código, pudiendo así, pasar a la pila un parámetro que sería usado como selector al ser cargado en un registro de segmento.

Esto ocurre si se comparan los valores, solamente, de CPL y DPL; para solucionarlo se compara el DPL del segmento al que se quiere acceder con el EPL (Nivel de Privilegio Efectivo).

3ª REGLA DE ACCESO A SEGMENTOS DE PILA

Sólo se permite acceder a segmentos de pila con el mismo PL que el del segmento de código que los solicita.

El CPL, el RPL del selector de segmento y el DPL del selector del segmento de pila deben ser iguales. En caso de restricción se genera una excepción de protección general (GP).

El procesador debe tener en cuenta los siguientes tipos de niveles de privilegio para llevar a cabo todas las comprobaciones:

1. Nivel de privilegio actual (CPL): PL del programa o tarea en curso, es el mismo que del segmento de código del que se realizan actualmente las instrucciones.

Al realizar una transferencia de control internivel, el procesador cambia el CPL.

El CPL puede encontrarse en los dos bits de menos peso de los registros CS y SS.

2. Nivel de privilegio del descriptor (DPL): PL de un segmento o puerta. Se almacena en el campo DPL del descriptor del segmento ó de la puerta.

Dependiendo del tipo de puerta o segmento al que se quiere acceder, se puede interpretar el DPL de diferentes maneras:

- Segmento de datos: el DPL indica el menor nivel de privilegio que un programa o tarea puede tener para que pueda acceder al segmento.
- Segmento de código, que no puede ser accedido a través de una puerta de llamada: indica el PL que se debe tener para poder acceder al segmento.
- Segmento de código accedido a través de una puerta de llamada: indica el mayor PL que se debe tener para acceder al segmento.
- Puerta de llamada: el DPL es el menor PL que se le permite tener a u segmento para acceder a la puerta. Esta norma es igual que la de los segmentos de datos.
- TSS: indica el menor PL que el programa o tarea, en estado de ejecución, puede tener para realizar el acceso. Esta norma es igual que la de los segmentos de datos.

3. Nivel de privilegio del peticionario (RPL): es el PL más importante y es almacenado en los bits 0 y 1 del selector de segmento.

   El CPL y el RPL son comprobados por el procesador para ver si se permite el acceso ó, por el contrario, es un acceso no permitido.

   Para asegurarse que las instrucciones privilegiadas no son accedidas por segmentos de un programador de aplicaciones a menos que el programa en sí tenga privilegios de acceso a este segmento se usa el RPL .

10.2.2.4. Acceso a segmentos con el bit ajustables (conforming) a cero

Para accesos a segmentos de código con C = 0, el CPL del segmento que quiere hacer el acceso, igual del DPL del segmento de código destino, sino se daría excepción de protección general (GP).

Al cargar un selector de segmento de un segmento de código en un registro CS, el campo del nivel de privilegio no varía aún cuando el RPL del segmento selector sea diferente del CPL.

El RPL debe ser jerárquicamente mayor o igual que el CPL de la rutina, por lo que se dice que el RPL tiene un efecto limitado en la comprobación de privilegio.

10.2.2.5. Acceso a segmentos con el bit ajustables (conforming) a uno

Cuando se accede a segmentos de código con C = 0, el CPL de la rutina que hace la llamada, puede ser igual o menor que el DPL del segmento de código destino, sino se daría excepción de protección general (GP).

En el caso de C = 1, no se tiene en cuenta el RPL del selector de segmento del destino. En estos casos el DPL representa numéricamente el de menor PL que puede tener una rutina para poder hacer una llamada.

El CPL no cambia al transferir el control del programa a un segmento de código conforming (aunque el DPL del destino sea mayor que el CPL), por lo que no hay intercambio de pilas. Es la única situación en la que el CPL puede ser distinto del DPL del segmento de código actual.

Los segmentos ajustables se usan para módulos de código como por ejemplo, librerías matemáticas y manejadores de excepciones, que aunque formando parte del S.O. o del software ejecutable pueden ser ejecutados en niveles de menos privilegio.

Para los segmentos de código nonconforming, que son la mayoría, a menos que la transferencia se lleve a cabo mediante una puerta de llamada, el programa de control puede ser transferido sólo a segmentos de código con el mismo nivel de privilegio.

10.2.3.Protección de las páginas

Los niveles existentes en la paginación son el Usuario y el Supervisor como se muestra en la figura 10.10. Las reglas de paginación se aplican tras aplicar las reglas en segmentación.

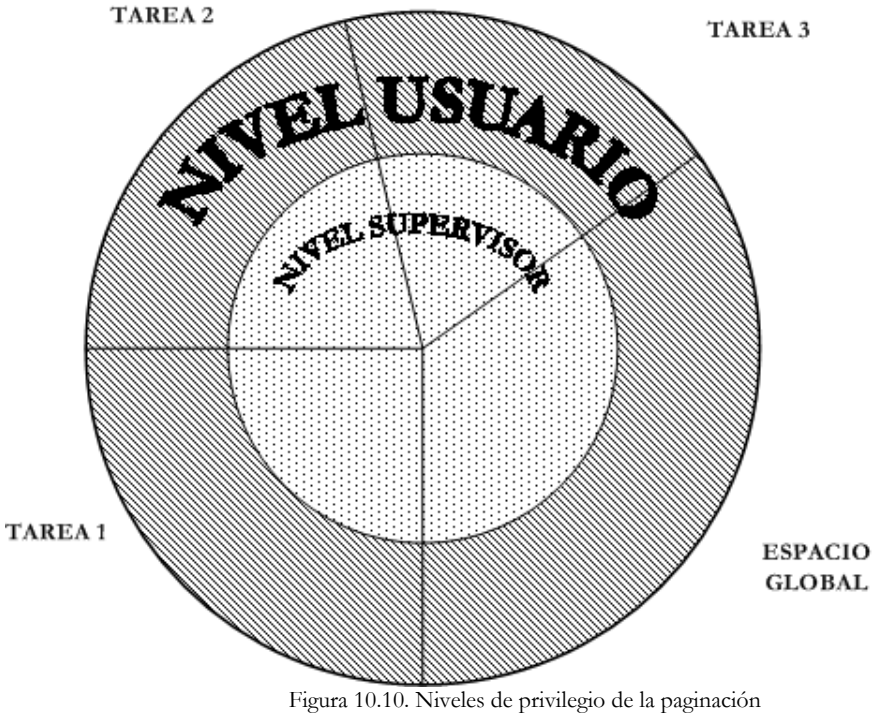


Figura 10.10. Niveles de privilegio de la paginación

Para definir el acceso el único bit que se usa el W/R, el cual se encuentra en las entradas de las Tablas de Páginas y en las del Directorio de Páginas. En caso que éste sea 0, puede leerse, si es 1 puede tanto leerse como escribirse.

Puede establecerse una relación entre estos niveles y los de la segmentación: en lo referente al nivel supervisor, éste equivale al los niveles 0, 1 o 2 en los que se realizan la ejecución de instrucciones y programador del sistema.

En cuanto al nivel usuario, que equivale al nivel 3 se realizan las aplicaciones de usuario y del programador de aplicaciones. En este tipo de protección se tienen en cuenta dos reglas:

1. Desde una página ubicada en nivel Usuario sólo se puede acceder a páginas de dicho nivel.

2. Desde una página del nivel de Supervisor se puede acceder a todas.

Este mecanismo de protección se usa al nivel de Directorio de Páginas y al nivel de Tabla de Páginas, es decir en la traducción de direcciones lineales a físicas. Puede darse que haya diferencias en ambas entradas respecto a la especificación de página, por lo que se coge como nivel de privilegio y tipo de acceso el más restrictivo.

Las distintas combinaciones quedan reflejadas en la tabla de la figura 10.11:

Entrada al Directorio de Páginas		**Entrada a la Tabal de Páginas**		**Efecto Combinado**	
Privilegio	Tipo de acceso	Privilegio	Tipo de acceso	Privilegio	Tipo de acceso
Usuario	Solo lectura	Usuario	Solo lectura	Usuario	Solo lectura
Usuario	Solo lectura	Usuario	lectura-escritura	Usuario	Solo lectura
Usuario	lectura-escritura	Usuario	Solo lectura	Usuario	Solo lectura
Usuario	lectura-escritura	Usuario	lectura-escritura	Usuario	lectura/escritura
Usuario	Solo lectura	Supervisor	Solo lectura	Supervisor	lectura/escritura*
Usuario	Solo lectura	Supervisor	lectura-escritura	Supervisor	lectura/escritura*
Usuario	lectura-escritura	Supervisor	Solo lectura	Supervisor	lectura/escritura*
Usuario	lectura-escritura	Supervisor	lectura-escritura	Supervisor	lectura/escritura
Supervisor	Solo lectura	Usuario	Solo lectura	Supervisor	lectura/escritura*
Supervisor	Solo lectura	Usuario	lectura-escritura	Supervisor	lectura/escritura*
Supervisor	lectura-escritura	Usuario	Solo lectura	Supervisor	lectura/escritura*
Supervisor	lectura-escritura	Usuario	lectura-escritura	Supervisor	lectura/escritura
Supervisor	Solo lectura	Supervisor	Solo lectura	Supervisor	lectura/escritura*
Supervisor	Solo lectura	Supervisor	lectura-escritura	Supervisor	lectura/escritura*
Supervisor	lectura-escritura	Supervisor	Solo lectura	Supervisor	lectura/escritura*
Supervisor	lectura-escritura	Supervisor	lectura-escritura	Supervisor	lectura/escritura

Figura 10.11. Tabla de niveles de privilegio

(*) si el flag WP de CR0 está a 1, el tipo de acceso lo determina R/W de las entradas del Directorio de Páginas y de la Tabla de páginas

Se debe tener en cuenta que no todas las instrucciones pueden ejecutarse desde cualquier nivel de privilegio, el sistema de protección el Pentium resuelve este problema.

Cabe destacar la existencia de dos grupos de instrucciones especiales que exigen ser ejecutadas con un CPL mínimo: el de instrucciones protegidas y el de instrucciones privilegiadas

10.2.4.1.Instrucciones protegidas

Son aquéllas que sólo pueden ejecutarse desde objetos de código situados en un nivel de privilegio igual o mayor que el indicado en el campo IOPL (Nivel de privilegio de E/S) del registro EFLAGS, éste es determinado por el programador del sistema.

El campo IOPL puede ser controlado dinámicamente por el S.O. para decidir los módulos que tienen acceso a los periféricos, este hecho se muestra en la figura 10.12.

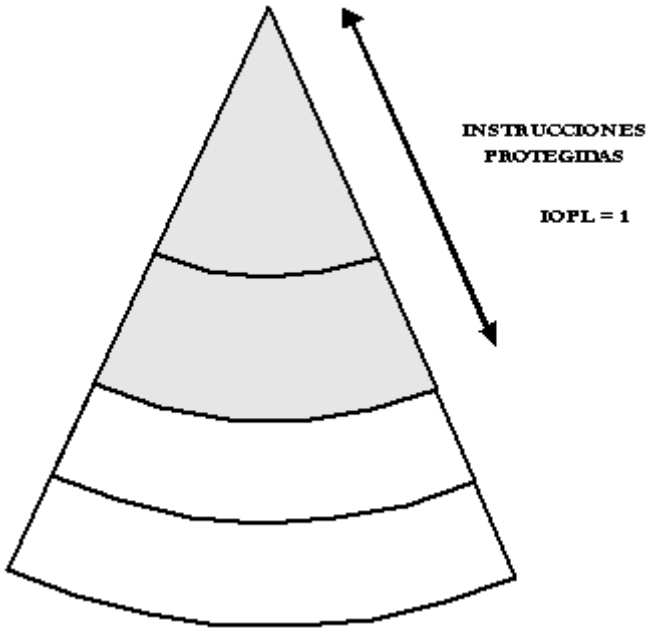


Figura 10.12. Si IOPL = 1, el grupo de instrucciones protegidas sólo se puede ejecutar desde segmentos de código con nivel de privilegio 0 y 1

Se incluyen las siguientes instrucciones para manejar las operaciones del espacio de E/S periféricos:

1. IN: puerta por la que toma un valor para cargar el acumulador
2. OUT: puerta en la que se deposita el valor del acumulador (EAX)
3. INS, OUTS: como las anteriores, pero manejan una cadena de caracteres
4. CLI: pone el flan de interrupción IF del registro de señalizadotes a 0, no permitiendo instrucciones mascarables
5. STI: pone el flan de interrupción IF del registro de señalizadotes a 1, permitiendo instrucciones mascarables.

10.2.4.2.Instrucciones privilegiadas

Están protegidas de los programas de aplicaciones, estas instrucciones controlan las funciones del sistema (como por ejemplo la carga de los registros del sistema).

Sólo se pueden ejecutar desde el nivel de mayor de privilegio (CPL=0), sino se da una excepción de protección general (GP).

Grupos en los que se clasifican:

1. Instrucciones que pueden modificar el IOPL

   POPF: carga los 32 bits de la cima de la pila en el registro de flags afectando al IOPL

   RET: retorno de interrupción, además de retornar al programa original, carga una parte en el registro de estado

2. Instrucciones que escriben los registros que controlan las tablas del sistema al operar en modo protegido

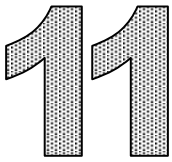
3. Instrucciones que afectan al contenido de la palabra de estado

4. Instrucción de paro HTL.

A continuación se muestra una lista de las instrucciones privilegiadas:

- LGDT: carga el registro GDT
- LLDT: carga el registro LDT
- LTR: carga el registro de tareas
- LIDT: carga el registro IDT
- MOV (registros de control):carga y almacena los registros de control
- LMSW: carga la palabra de estado de la máquina
- CLTS: pone a cero el flan de conmutación de tareas del registro CR0
- MOV (registro debug): carga y almacena registros de debug
- INVD: invalida la caché, sin actualizar la memoria principal
- WBINVD: invalida la caché, actualizando la memoria principal
- INVLPG: invalida la TLB
- HLT: para temporalmente el procesador.
- RDMSR: Read Mode-Specific Registers : instrucción de leer modelo y registro específico.
- WDMSR: Write Mode-Specific Registers : instrucción de escribir modelo y registro específico.
- RDPMC: Read Preformase-Monitoring Counter: leer contadores de monitoreo de desempeño.
- RDTSC: Read Time-Stamp Counter: leer contador de tiempo estampado

MODELO DEL PENTIUM PARA EL PROGRAMADOR DE SISTEMAS



11.1- REGISTROS DEL SISTEMA

En un entorno protegido y multitarea como son el Modo Protegido y el Modo Virtual 86, es indispensable que el programador del sistema posea un conocimiento profundo de todos los mecanismos de protección de la memoria. Además de saber manejar los recursos de la CPU destinados al desarrollo de aplicaciones, dicho programador debe controlar los que tienen un carácter especial y permiten obtener el máximo rendimiento del Pentium en el Modo Protegido. Dichos recursos son destinados a manejar:

- Memoria Virtual (Segmentación y Paginación).
- Entorno Protegido (Reglas de acceso).
- Multitarea (Conmutación de tarea).
- Control del flujo de instrucciones.
- Manejo de interrupciones y excepciones.
- Gestión de la caché.
- Pruebas y autochequeos.

En la figura 11.1 se muestra un esquema que abarca el conjunto de registros del sistema, usados por el programador de sistemas. Obsérvese en dicha figura que aparecen los registros de segmento (CS, DS, SS, ES, FS y GS), cuya misión también se describió al analizar los registros del programador de aplicaciones. El motivo de su presencia en ambos conjuntos de registros se debe a que, si bien el programador de aplicaciones puede modificar en sus programas el contenido de los mismos, cuando la CPU opera en Modo Protegido, se accede a través de ellos a las tablas de descriptores de segmento y se obtienen los parámetros que se cargan, de forma automática y transparente, en los registros caché ultrarápidos e invisibles, sirviendo de base para el direccionamiento y gestión de la memoria virtual.

Igualmente, se ha incluido en la figura 11.1 otro registro presentado dentro del conjunto que maneja en programador de aplicaciones. Se trata del registro de señalizadores EFLAGS también llamado registro de estado. La razón se debe a que en él hay varios bits que son propios en el control del sistema:

- **IOPL:** Nivel de privilegio de E/S. Controla el acceso al espacio de E/S.
- **NT:** Tarea anidada.
- **RF:** Flag de reanudación. Invalida temporalmente los fallos de depuración.
- **VM:** Modo Virtual-86.
- **AC:** Chequeo de alineamiento.
- **VIF:** Flag de interrupción en modo virtual.
- **VIP:** Flag de interrupción pendiente en modo virtual.
- **ID:** Flag de permiso de identificación del procesador.

Los restantes bits del registro de estado (EFLAGS) se usan en el desarrollo de las aplicaciones de usuario.

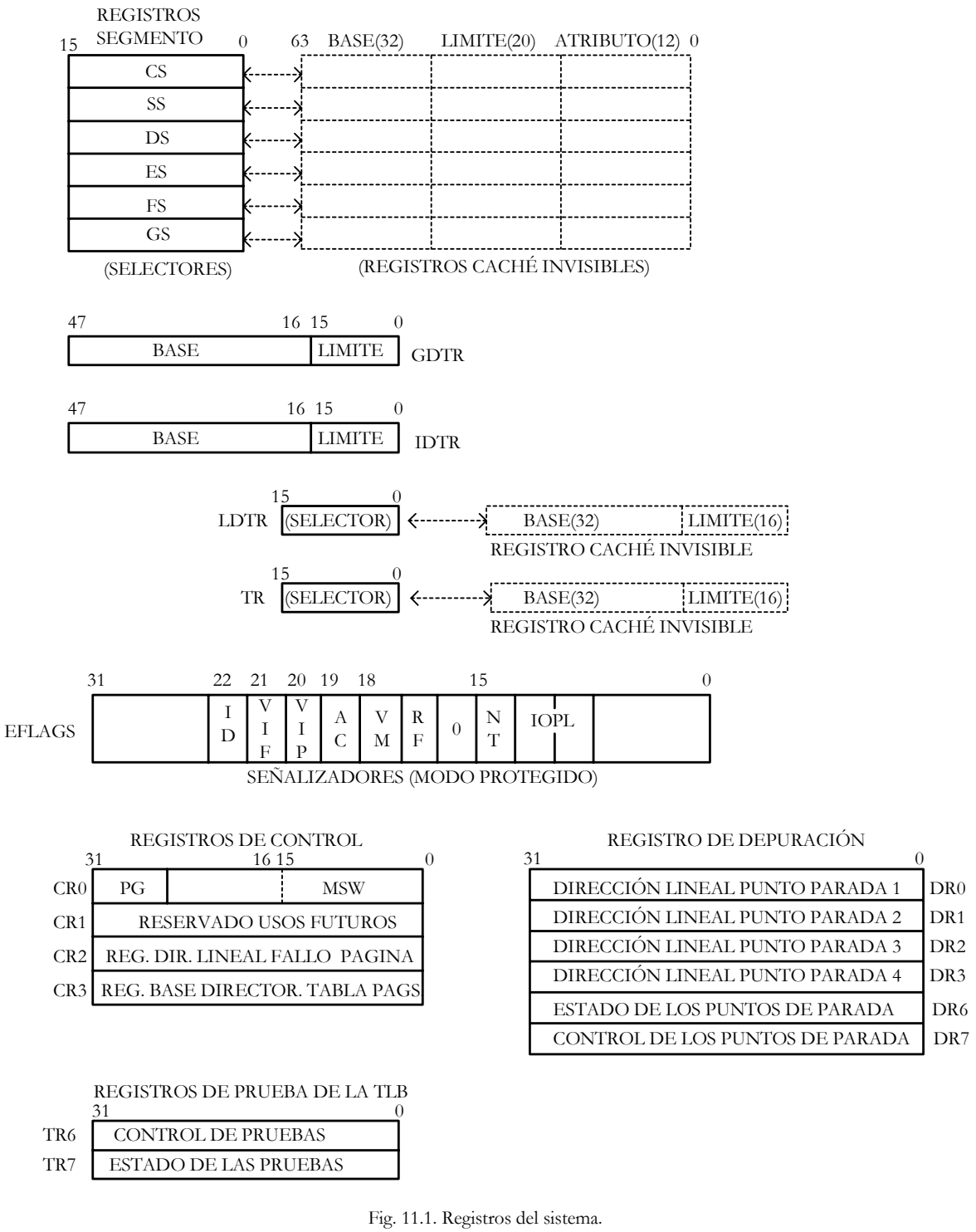


Fig. 11.1. Registros del sistema.

11.2- REGISTROS DE SEGMENTO

Los procesadores Intel utilizan la segmentación como método principal y obligatorio en la gestión de la memoria, mientras que la paginación es optativa.

Los segmentos quedan definidos por su dirección de inicio o base, su tamaño o limite y sus atributos o derechos de acceso.

En Modo Real, la base se calcula multiplicando por 16 (en binario multiplicar por 16 significa añadirle 4 ceros por la derecha) el contenido del registro correspondiente, ya que los registros de segmento son de 16 bits y dado que en Modo Real solo se utiliza el primer MB de la memoria, solo se necesitan 20 bits para direccionar la memoria (2^{20}= 1 MB). El límite es fijo y de 64 KB de valor, y los atributos no se utilizan en este modo de trabajo, porque en Modo Real no hay sistema de protección y todos los accesos están permitidos.

En Modo Protegido, los segmentos se especifican mediante estructuras de datos de 8 bytes (64 bits), denominadas descriptores, que están agrupados en tablas. Cada descriptor consta de los siguientes campos:

- Base del segmento (32 bits).
- Límite del segmento (20 bits).
- Atributos (12 bits).

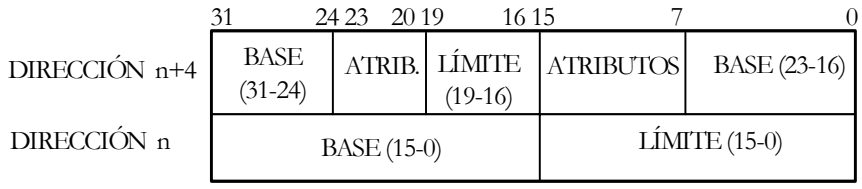


Fig. 11.2. Formato de un descriptor de segmento compuesto por ocho bytes, que ocupan dos posiciones de memoria cada una.

En cada instante, la CPU maneja los segmentos propios de la tarea en curso, que se hallan situados en el área local de dicha tarea. También puede manejar los segmentos compartidos por todas las tareas que se ubican en el área global.

La tabla GDT (Tabla de Descriptores Globales) dispone de tantas entradas como descriptores de segmentos existan en el área global. Dicha tabla tiene un tamaño de 64 KB, por lo que admite hasta 8 K descriptores. Cada descriptor de dicha tabla tiene 64 bits.

Los descriptores de los segmentos específicos de la tarea en curso están reunidos en otra tabla, llamada LDT, Tabla de Descriptores Locales, de la misma capacidad que la GDT, y que al igual que esta última, debe residir en la memoria principal desde el preciso instante en que se pasa al Modo Protegido, para que la CPU pueda localizar el descriptor del segmento a ejecutar.

Entre la GDT y LDT, la CPU puede controlar una memoria virtual de 16 K descriptores de segmento. Puesto que cada segmento puede alcanzar un tamaño máximo de 4 GB, ya que el límite de los segmentos tiene 32 bits y 2^{32} = 4 GB, teóricamente la capacidad máxima de la memoria virtual que puede manipular un Pentium llegaría a los 64 TB.

En la figura 11.3 se recuerda, de forma gráfica, la actuación del contenido de los registros de segmento, esto es, es el selector del descriptor de un segmento en la GDT o en la LDT.

En el campo RPL del registro de segmento viene el valor del nivel de privilegio del segmento que ha solicitado la intervención del segmento bajo análisis, es decir, es el nivel de privilegio (PL) del segmento que se encuentra en el registro CS. TI, es el Indicador de Tabla, de forma que si TI=0, el descriptor se encuentra en la GDT, y si TI=1, en la LDT.

Los 13 bits de mas peso del selector actúan como índice que referencia una entrada en la tabla seleccionada por TI. Como cada descriptor consta de 8 bytes, el valor del índice se multiplica por 8 para obtener la dirección de inicio del descriptor seleccionado.

Una vez que se ha seleccionado la entrada en la tabla de descriptores, la información contenida en esta se carga en un registro caché oculto ultrarápido que esta asociado al registro de segmento, esta información constará de los 64 bits que hacen falta para direccionar en Modo Protegido, 32 bits para la base, 20 bits para el límite y 12 bits para los atributos, con lo que el registro de segmento correspondiente ya podrá direccionar la memoria.

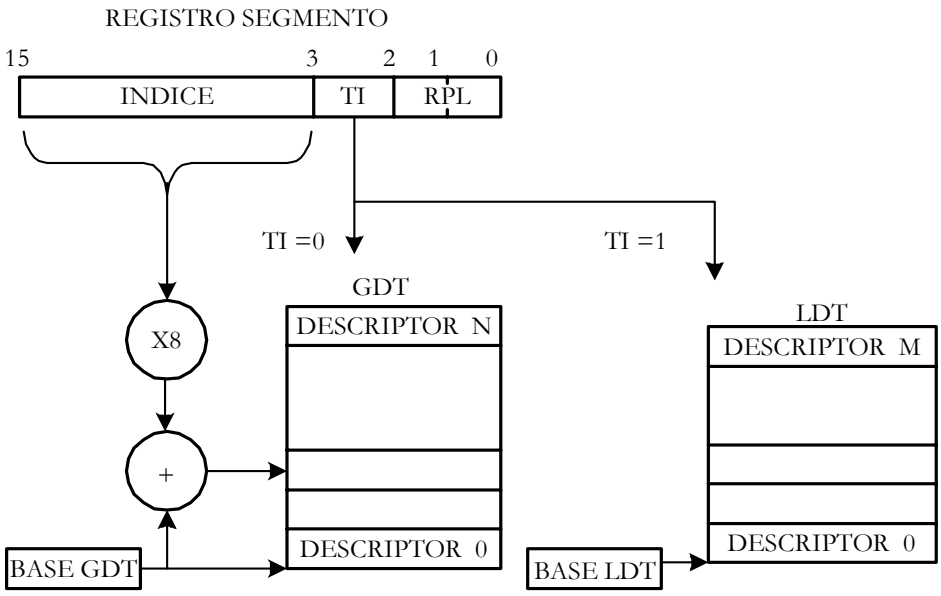


Fig. 11.3. El contenido del registro segmento actúa, en Modo Protegido, como un selector de una entrada en la GDT o en la LDT.

11.3- REGISTROS DE TABLAS

Existen tres registros dedicados a la manipulación de las tablas de descriptores:

1. **GDTR:** Registro de la base de la GDT.
2. **IDTR:** Registro de la base de la IDT (Tabla de Descriptores de Interrupción).
3. **LDTR:** Registro del selector del descriptor de la LDT.

Los registros GDTR e IDTR tienen un tamaño de 48 bits, 32 de ellos están destinados a contener la dirección inicial de la tabla correspondiente (base), y los restantes 16, los de menos peso, para indicar el tamaño, que puede alcanzar un máximo de 64 KB.(2^{16}=64 KB)

El GDTR contiene la dirección de la base de la GDT y el tamaño de la misma. Las direcciones de las entradas de las tablas de descriptores deben ser múltiplos de 8. Para manipular este registro se utilizan las instrucciones LGDT y SGDT, para cargarlo y almacenarlo, respectivamente.

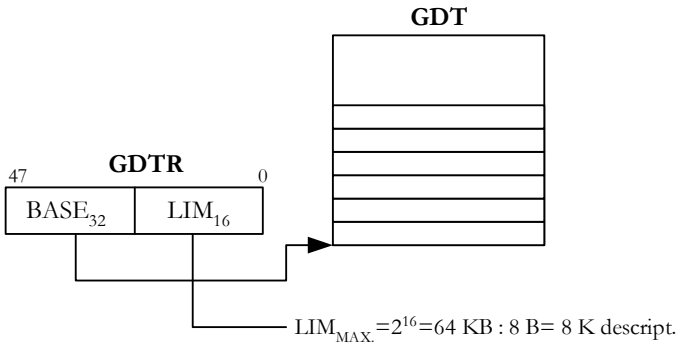


Fig. 11.4. Funcionamiento del registro GDTR.

El IDTR almacena el valor de la base de la IDT, así como su límite. Los descriptores contenidos en la tabla IDT se utilizan cuando se producen interrupciones y excepciones. La tabla IDT, como se estudiará mas adelante, puede contener hasta 256 descriptores de 8 bytes, por lo que su tamaño en Modo Protegido, alcanza los 2 KB. Las entradas de esta tabla en Modo Real no contienen lo mismo que en Modo Protegido.

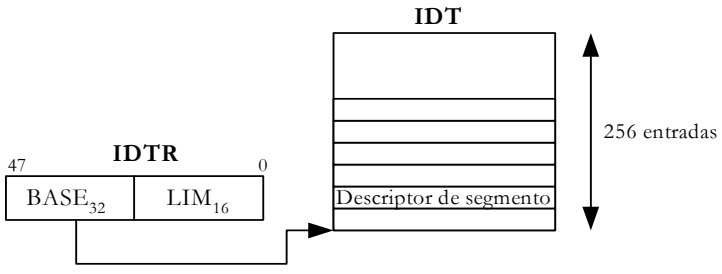


Fig. 11.5. Funcionamiento del registro IDTR.

El registro LDTR solo consta de 16 bits y actúa como un selector de un descriptor de segmento de la GDT. El segmento al que referencia el descriptor contiene la Tabla de Descriptores Locales, LDT, que guarda los descriptores de los segmentos de una tarea. Habrá, por tanto, tantas tablas de descriptores LDT como tareas haya ejecutándose en el sistema. Sin embargo, solo hay un registro LDTR, el cual apuntará siempre a la base de la LDT de la tarea en curso. Para cargar y almacenar este registro se usan las instrucciones LLDT y SLDT respectivamente.

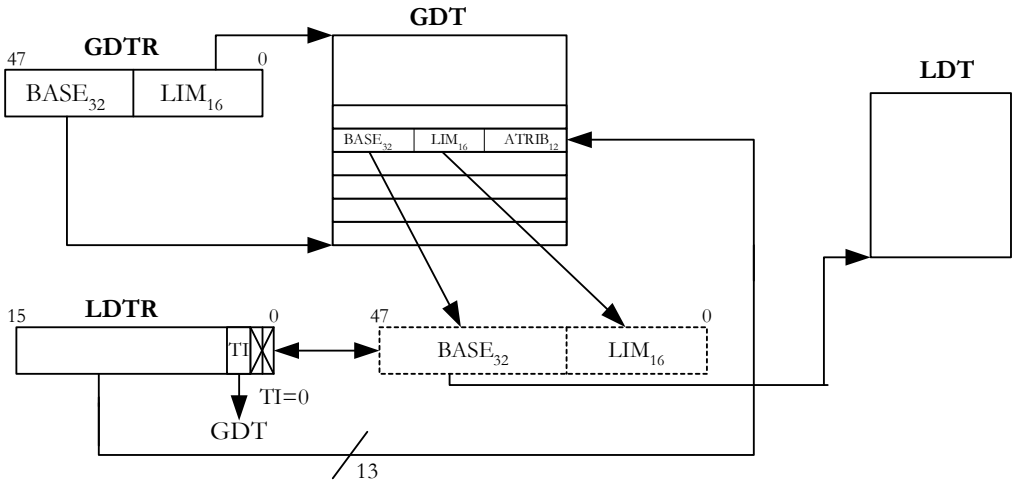


Fig. 11.6. Funcionamiento del registro LDTR.

Con el valor que hay en LDTR, tratándolo como un selector, se localiza uno de los descriptores de la GDT, cuya información (base, límite y atributos), carga la CPU automáticamente en el registro caché invisible asociado a LDTR, que se encarga de determinar la LDT de la tarea en curso. Para que esto se pueda llevar a cabo, en el bit TI del registro LDTR, el cual es el índice de tabla hay un cero, con lo que se indica que el selector que contiene ese registro en sus 13 bits de mayor peso apunta a la GDT.

Los segmentos que contienen las LDT de cada tarea deben tener un descriptor de segmento, cada uno, en la GDT.

Antes de entrar al Modo Protegido hay que asegurarse de que en la memoria principal residen la GDT y la IDT, para que la CPU pueda localizar los segmentos que precisa y atender las excepciones que se produzcan.

11.4- REGISTRO DE TAREA

El registro de tarea (TR), posee 16 bits y actúa, al igual que el LDTR, como selector de un descriptor de un segmento en la GDT. Dicho segmento recibe el nombre de Segmento de Estado de la Tarea (TSS) y guarda el contexto del procesador necesario para reanudar esa tarea.

Se entiende por contexto de la CPU para una tarea, la información que existe en todos los registros del procesador y que se precisa para poder reanudar la tarea en la misma situación en que se abandonó cuando se salvo el contexto.

El contexto de una tarea del Pentium está formado por el valor de los registros de propósito general, el del registro de estado (EFLAGS), EIP, registros de segmento, etc. En la figura 11.4 se muestra la actuación de TR y el contenido del segmento TSS, al que referencia indirectamente TR, a través de la GDT. Apréciese como TR dispone de un registro caché invisible donde la CPU carga los parámetros que definen al segmento TSS.

Para que un sistema pueda llevar a cabo la multitarea, debe poder ejecutar una operación llamada conmutación de tareas, la cual consiste en salvar el contexto de la tarea en curso, guardándola para poder ejecutarla posteriormente, y cargar el contexto de la nueva tarea que se quiera ejecutar, esto es, cada vez que la CPU abandona una tarea para iniciar la ejecución de otra, se salva automáticamente, el contexto actual del procesador en el TSS de la tarea saliente. Después el contenido del TSS de la tarea entrante se carga en los registros de la CPU, conformando el nuevo contexto de trabajo.

Los segmentos que contienen los TSS de cada tarea en curso tienen que tener un descriptor, cada uno, en la GDT.

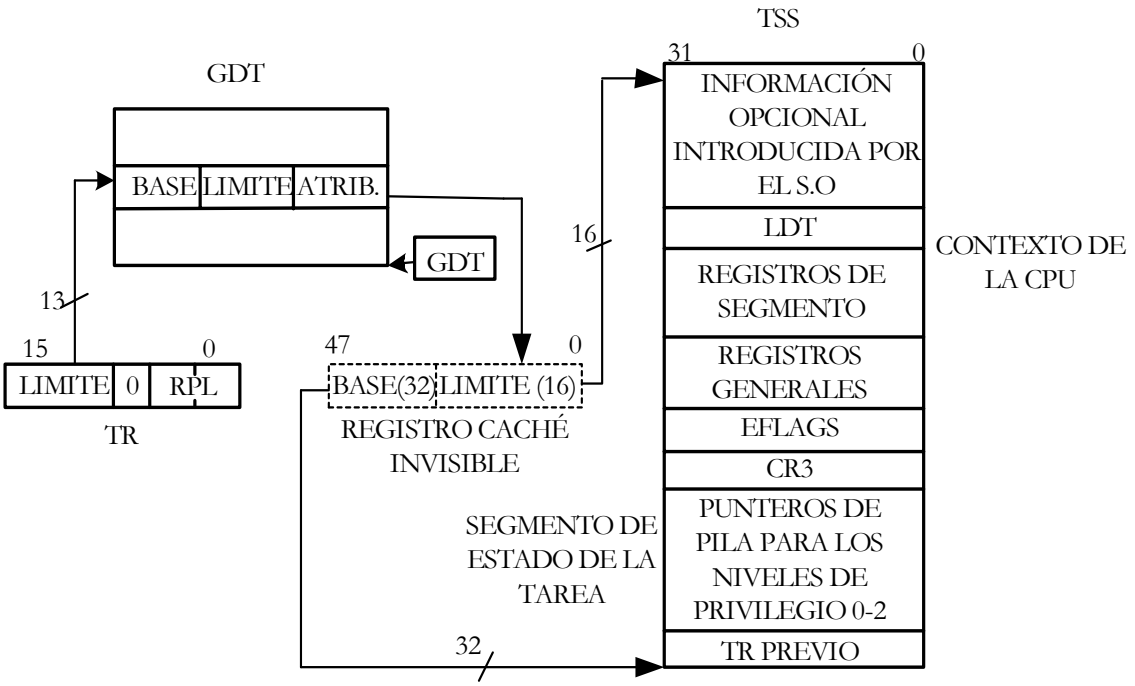


Fig. 11.7. El registro TR actúa como un selector de un descriptor de segmento de la GDT, que apunta al segmento TSS, que es el Segmento de Estado de la Tarea. Tiene igual comportamiento que el registro LDTR.

11.5- REGISTRO DE SEÑALIZADORES (EFLAGS)

El registro EFLAGS, como ya se ha visto anteriormente, se compone de 32 bits, de los cuales 21 son significativos. No obstante, los que a continuación se explicaran son específicos para el programador de sistemas, siendo el resto usados por el programador de aplicaciones:

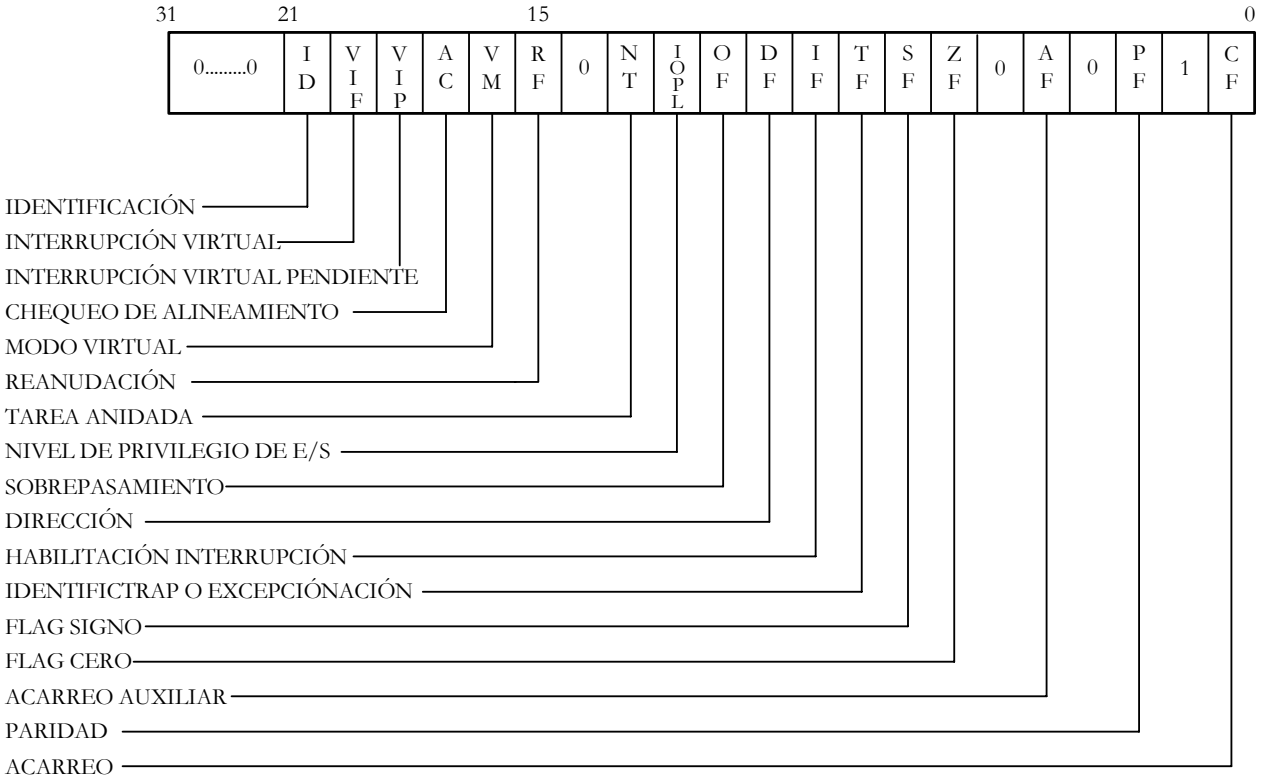


Figura 11.8. Distribución de los bits señalizadores del registro EFLAGS.

- **VM: Modo virtual-86:** Sirve para pasar desde el Modo Protegido al Modo Virtual 86, para lo cual basta con poner a 1 este bit, dentro del Modo Protegido. En este modo de funcionamiento, se pueden ejecutar programas del 8086 en un entorno protegido, esto es, con sistemas de protección, multitarea...

- **RF: Flag de reanudación del punto de parada:** El Pentium dispone de un conjunto de registros de depuración para programar puntos de parada. Poniendo RF=1, se permite ignorar los puntos de parada.

- **IOPL: Nivel de privilegio de las E/S:** Tiene 2 bits. Contiene el menor nivel de privilegio que deben tener los segmentos de código, en Modo Protegido, para poder ejecutar las instrucciones protegidas relacionadas con las E/S.

- **NT: Tarea anidada:** La tarea en fase de ejecución ha sido llamada por la otra a la que hay que retornar en el caso de que el bit NT=1. Controla el encadenamiento de las interrupciones y las llamadas de las tareas. El procesador pone este flag a 1 cuando se hace una llamada a una tarea mediante una instrucción CALL, una interrupción o una excepción, y examina y modifica este flag cuando se retorna de una tarea mediante la instrucción IRET. De cualquier modo, cambiar el valor de este flag puede generar excepciones inesperadas en las aplicaciones.

- **AC: Chequeo de alineamiento:** Si AC=1 y AM=1 en el registro de control CR0 la CPU genera una excepción cuando localiza un operando cuya dirección no sea múltiplo de 4. Normalmente las excepciones provocadas por el chequeo de alineamiento se producen solo en el modo usuario (PL=3). Las excepciones por chequeo de alineamiento pueden ser usadas para comprobar el alineamiento de los datos. Esto es útil cuando se intercambia información con otros procesadores, lo que requiere que los datos estén alineados.

- **VIF: Flag de interrupción en modo virtual:** Es un bit equivalente al bit IF del modo protegido, que se utiliza cuando trabajamos en modo virtual-86, es decir, es la imagen virtual del flag IF. El procesador solo reconoce este flag cuando el bit VME o el bit PVI en el registro CR4 están activados (1) y el nivel de privilegio (PL) es inferior a 3.

- **VIP: Flag de interrupción pendiente en modo virtual:** Este bit nos indica que una interrupción esta pendiente cuando esta a 1, por el contrario no esta pendiente cuando es 0. El software activa y desactiva este flag, mientras que el procesador solamente lo lee, nunca lo modifica. Al igual que pasa en el flag anteriormente visto, el procesador solo reconoce este flag cuando el bit VME o el bit PVI en el registro CR4 están activados (1) y el nivel de privilegio (PL) es inferior a 3.

- **ID: Identificación del procesador:** Si se activa este flag se habilita el uso de la instrucción CPUID, la cual identifica el procesador devolviendo muchas de las características del mismo.

11.6- REGISTROS DE CONTROL

Son 5 registros de 32 bits cada uno, llamados CR0-CR4, de los cuales el CR1 no esta definido en el funcionamiento del Pentium y su uso está previsto desarrollar en nuevos procesadores.

CR0, CR2, CR3 y CR4 son accesibles al programador de sistemas y pueden ser leídos y escritos mediante instrucciones del tipo MOV desde los registros de propósito general. Estos registros determinan el modo de operación del procesador y las características de la tarea en ejecución.

11.6.1- CR0: Doble palabra de estado de la máquina

Como puede apreciarse en la figura 11.6, este registro solo tiene asignadas funciones a once de sus bits.

Los bits definidos en CR0 son:

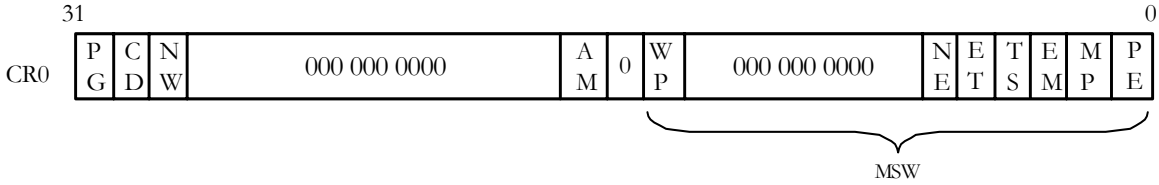


Figura 11.9. Estructura del registro CR0, cuyos 16 bits de menos peso forman la "palabra de estado de la maquina" (MSW).

- **PG: Paginación:** Cuando este bit vale 1 significa que está habilitada la Unidad de Paginación, que traduce las direcciones lineales a físicas. Cuando la paginación esta desactivada, todas las direcciones lineales son tratadas como direcciones físicas. El flag PG no tiene efecto si el flag PE (bit 0 en CR0) no esta tambien activo. De hecho, la activación de PG estando PE inactivo provocaría que se generase una excepción de protección general.
- **CD: Caché desactivada:** Cuando CD=0 activa la caché interna. Si CD=1 la desactiva.
- **NW: No escritura obligada:** Si NW=0 pone en marcha la escritura obligada y los ciclos de invalidación de la caché. Si NW=1 los desactiva.
- **ET: Tarea de coprocesador:** En el Pentium no se utiliza. Se usaba en los procesadores X86, 286 y 386 para indicar si el coprocesador externo era el 287 ó el 387. Cuando valía 1 el coprocesador era el 387.
- **TS: Tarea conmutada:** La CPU pone a 1 este bit al producirse una conmutación de tarea. Luego, durante la ejecución de instrucciones, lo examina cuando encuentra alguna instrucción para el coprocesador, en cuyo caso salva el contexto de la anterior tarea que tenía el coprocesador y carga el nuevo poniendo TS=0.
- **EM: Emulación:** Cuando EM=1 la ejecución de una instrucción numérica genera una excepción, por lo que debe tener dicho valor en caso de que el procesador no disponga de FPU (Unidad de Coma Flotante). Es decir, este bit se usa para indicar si se utiliza el emulador del coprocesador. Poner este bit a 1 fuerza a todas las instrucciones en coma flotante a ser ejecutadas por emulación software.
- **PE: Habilitación del modo protegido:** Cuando PE=1, el Pentium opera en Modo Protegido, mientras que si PE=0, lo hace en Modo Real.
- **AM: Mascara de alineamiento:** Cuando AM=1 y el bit AC(EFLAGS)=1 si el nivel de privilegio (PL)=3, estando en modo protegido o modo virtual-86, comprueba si la dirección de los operandos es múltiplo de 4 y si no lo es genera una excepción.
- **WP: Protección de escritura:** Si WP=1 impide escribir una pagina de nivel de usuario definida de solo lectura o ejecución, incluso desde el modo supervisor.
- **NE: Error del coprocesador:** Cuando NE=1 se activa el mecanismo normal de devolver errores en la x87 FPU. Si NE=0 y IGNE#=0 se ignoran los errores numéricos. Si NE=0 y IGNE#=1 un error numérico causa la parada del procesador y espera una interrupción. Dicha interrupción llega por la patita FERR# que provoca una entrada al controlador de interrupciones y detiene inmediatamente la ejecución de instrucciones antes de ejecutar la siguiente instrucción en coma flotante o las instrucciones WAIT/FWAIT.
- **MP: Monitor de coprocesador:** En el Pentium MP=1. Se usaba en procesadores anteriores para controlar la función de la instrucción WAIT, que se usa para sincronizar el procesador con el coprocesador.

11.6.2- CR2: Dirección lineal de fallo de página

En este registro se almacena la dirección lineal que se introdujo en la Unidad de Paginación para traducirla a dirección física, y que ocasionó un error o fallo de pagina.

El código de error que especifica la causa de fallo se almacena en la pila del manipulador de los fallos de página.

Figura 11.10. Estructura del registro CR2.

11.6.3- CR3: Base del directorio de las tablas de páginas

Guarda la dirección física en la que comienza el Directorio de las Tablas de Páginas de la tarea en curso. Como dicho Directorio tiene formato de una página de 4 KB, los 12 bits de menos peso de CR3 (como deberían ser todo 0) se ignoran en la escritura y solo dos tienen significado en el Pentium: el PCD y el PWT. Se le conoce como Registro Base del Directorio de Páginas (PDBR).

Figura 11.11. Estructura del registro CR3.

El bit PCD tiene el valor de la patita del Pentium PCD en los ciclos que no hay paginación. Se usa para controlar la caché. Para que un acceso a memoria pueda ser llenado con una línea de la caché, la patita KEN# debe estar a cero. Es decir, para permitirlo PCE y KEN# deben valer cero. El bit PWT toma el valor de la patita del Pentium del mismo nombre en los ciclos que no hay paginación.

Cuando hay paginación, estos dos flags indican lo siguiente:

- PWT: Pagina de escritura obligada.
- PCD: Pagina cacheable.

Indican lo mismo que los flags del mismo nombre en las entradas de las tablas de páginas.

Tanto la conmutación de tareas, como la carga de CR3, invalidan todas las entradas de la TLB.

11.6.4- CR4: Extensiones de la arquitectura

Es un nuevo registro que se incorporó en el Pentium y que contiene varios bits que soportan diversas extensiones de la arquitectura.

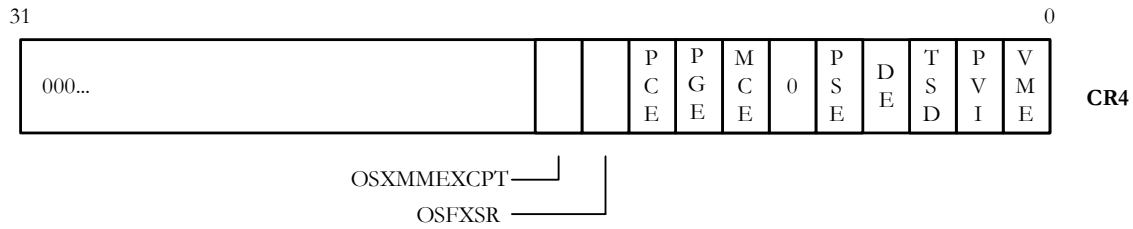


Figura 11.12. Distribución de los bits mas significativos del registro CR4.

- **VME: Extensión del modo virtual 8086:** Si VME=1 se activa el soporte para que funcione el flag VIF para las interrupciones en Modo Virtual.
- **PVI: Interrupciones virtuales en modo protegido:** Si PVI=1 se activa el soporte para que funcione el flag VIF para las interrupciones en Modo Protegido. Así algunos programas diseñados para ejecutarse con PL=0, se pueden ejecutar con PL=3.
- **TSD: Desactivación del "time-stamp":** Si TSD=1 la instrucción RDTSC pasa a ser de tipo privilegiada. Esto es, no se puede ejecutar para los niveles de prioridad distintos a 0 (PL != 0).
- **DE: Depuración de extensiones:** Con DE=1 se activan los puntos de ruptura en el espacio de E/S.
- **PSE: Extensión del tamaño de página:** Si PSE=1, se activan páginas de 4 MB. Por el contrario si PSE=0 las páginas son de 4 KB.
- **MCE: Activación del chequeo de la máquina:** Con MCE=1 se activa la excepción correspondiente al chequeo de la máquina.
- **PGE:** Cuando este bit esta a 0, las paginas globales están desabilitadas, mientras que si esta a 1 están habilitadas.
- **PCE:** Este bit permite, cuando esta activado (PCE=1), ejecutar la instrucción RDPMC. Sin embargo, esta instrucción solo se podrá ejecutar desde el nivel de prioridad 0 (RPL=0) cuando este bit valga 0.

11.7- REGISTROS DE DEPURACION

Forman un conjunto de ocho registros de 32 bits, dos de los cuales (DR4-DR5) son de uso reservado por el fabricante. Son accesibles al programador y constituyen una potente herramienta para la puesta a punto del sistema lógico, puesto que permiten especificar hasta cuatro puntos de parada o ruptura, que se controlan de forma muy flexible. (Figura 11.9).

Cuando se produce un punto de parada o se da una condición especial en la depuración, el procesador genera un tipo concreto de interrupción cuya rutina de servicio se encarga de atender el origen que la ocasionó.

El señalizador de reanudación, RF, si esta a 1, permite volver a ejecutar una instrucción después de producirse una excepción de depurado, de manera que no se reproduzca la misma excepción a causa de las mismas condiciones.

Los registros DR0-DR3 contienen las direcciones asociadas a una de las cuatro condiciones de punto de ruptura, definidas por ciertos bits en el registro DR7. Estos cuatro registros contienen direcciones lineales, que son direcciones físicas cuando se halla inhabilitada la paginación, o bien hay que traducirlas a físicas si está habilitada. En este último caso, en DR7 residen los bits de habilitación local y global, que determinan que direcciones de los registros DR0-DR3 son relevantes para cada tarea.

A DR7 se le llama registro de control, pues define y habilita o no, selectivamente, las condiciones de depuración. Cada registro de direcciones de depuración, DR0-DR3, tiene en DR7 sus propios campos de dos bits para la habilitación de puntos de ruptura de Lectura/Escritura (RWx), Longitud (LENx) y Local/Global (Lx y Gx).

En DR7, los bits 16:17, 20:21 y 24:25 contienen los campos R/W. Estos campos R/W se interpretan de la siguiente forma:

00: Ruptura en ejecución de una instrucción.
01: Ruptura en escritura de datos.
10: Operaciones L/E (DE debe ser 1).
11: Ruptura en escritura o lectura de datos, pero no en la búsqueda de instrucciones.

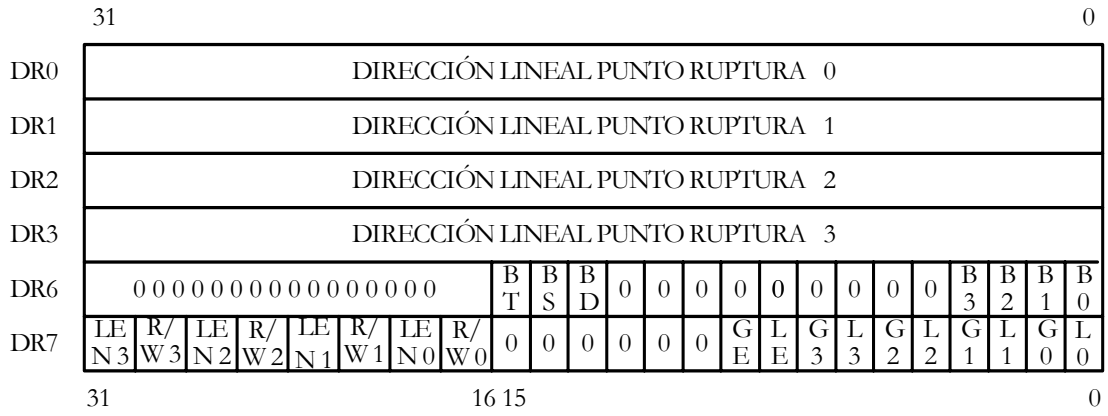


Figura 11.13. Formato de los seis registros de depuración que tienen definidas funciones en el Pentium.

Cada registro de direcciones DR0-DR3, tiene asociado en DR7 un campo LENx que especifica la longitud de los datos que se están vigilando, para esto los datos deben estar alineados. Estos campos LEN se encuentran en los bits 18:19, 22:23, 26:27, 30:31 de DR7. LENx se interpreta:

00: Longitud de un byte.
01: Longitud de dos bytes.
10: No utilizado.
11: Longitud de cuatro bytes.

Cuando el campo R/W n esta a cero, el campo Ln también debe estar a cero.

Cada registro de direcciones también tiene asociados dos campos, llamados Lx y Gx. Los Lx, que se encuentran en los bits 0, 2, 4 y 6 de DR7, representan las habilitaciones locales de las condiciones x de los puntos de ruptura. Los Gx, sin embargo, se encuentran en los bits 1, 3, 5 y 7 de DR7, y representan las habilitaciones globales de las condiciones x de los puntos de ruptura. En los dos casos anteriores, un 1 habilita local o globalmente la condición x del punto de ruptura al que este asociado (x), mientras que un 0 la deshabilita. La diferencia entre ellos es que Ln habilita su puesta a cero (desactivación) por el microprocesador en cada conmutación de tarea para evitar condiciones indeseables de puntos de ruptura en la nueva tarea.

El registro DR7, además de todos los anteriores, tiene tres bits llamados LE, GE y GD, los cuales se encuentran en los bits 8, 9 y 13, respectivamente. Los dos primeros se ignoran en los procesadores superiores al 486. Sin embargo, GD si tiene un significado en el Pentium: si GD vale 0 se deshabilita la protección de los registros de depuración, mientras que si vale 1 se habilita esta protección. Este bit es puesto a 0 cuando se entra al manipulador de excepciones.

DR6 contiene varios señalizadores de condición de depuración, que permiten determinar al depurador, las condiciones que se han producido con el error. Cuando se detecta una excepción de depuración habilitada, el bit asociado Bn se pone a 1. El campo BT de DR6 funciona con un bit de excepción en depuración que se guarda en la primera posición del TSS. BT se pone a 1 antes de introducir al manipulador de depuraciones, si se ha producido una excepción de depuración por una conmutación de tareas y el bit de excepción del TSS esta a 1. El bit señalizador BS funciona con el bit del señalizador de excepción TF, del registro EFLAGS. BS se pone a 1 cuando el manipulador de depuraciones actúa como resultado de una ejecución paso a paso. El señalizador BD indica si la siguiente instrucción leerá o escribirá uno de los ocho registros de depuración.

El microprocesador nunca pone a cero los señalizadores de DR6. Los ceros se desplazan a DR6 antes de intentar identificar la siguiente excepción de depuración.

Los registros de depuración mejoran considerablemente las características y el control sobre puntos de ruptura propios de los microprocesadores convencionales. Así, por ejemplo, añaden la posibilidad de forzar puntos de ruptura sobre acceso a datos, de forma que si una variable se intenta rescribir accidentalmente, se puede activar un punto de ruptura que detenga la ejecución, siempre que se vaya a modificar el contenido de la variable.

Otras herramientas de las que dispone el Pentium, complementarias a los registros de depuración son:

1. **Puntos de ruptura por software:** Usando la instrucción de un byte INT3, se provoca una excepción al ser ejecutada, la cual es tratada mediante el recurso que halla para su atención en la IDT.
2. **Paso a paso:** Cuando el señalizador TF=1, se produce una excepción cada vez que se ejecuta una instrucción. La rutina que atiende esta excepción puede usarse para visualizar el estado de la CPU, o bien, examinar alguna característica del procesador.

11.8- REGISTROS DE PRUEBA DE LA TLB

Son dos registros de 32 bits, TR6 y TR7, con los que se puede leer y escribir el contenido de una entrada de la TLB, que es una pequeña memoria caché ultrarápida que contiene la traducción de dirección lineal a dirección física de las 32 páginas que se han usado últimamente.

La TLB consta de 32 entradas agrupadas en cuatro bloques de ocho entradas cada uno. Cada entrada se compone de dos campos, uno con la etiqueta y otro, asociado con la información correspondiente. En el campo de etiqueta residen los 20 bits de más peso de la dirección lineal, junto con 4 bits de atributos (V: Validez, D: Sucio, U: Usuario, W: Escritura). En el campo de datos asociado al de la etiqueta se almacenan los 20 bits de mas peso de la dirección física correspondiente a la lineal de la etiqueta.

El programador de sistemas puede leer y escribir los registros de prueba TR6 y TR7 mediante instrucciones privilegiadas que solo se pueden ejecutar en el nivel de privilegio 0 (tipo MOV).

Las dos operaciones básicas que se pueden realizar con los registros de prueba son:

1. Leer una entrada de la TLB, comprobando la dirección física que contiene, correspondiente a la dirección lineal que se proporciona.
2. Escribir una entrada de la TLB, introduciendo la dirección lineal y la física que corresponde.

Al registro TR6 se le denomina de comando o de control, y fundamentalmente contiene el campo Etiqueta de la TLB y el bit de comando C, que si vale 1, indica que se va a realizar una lectura en la TLB, y si vale 0, una escritura.

Como se refleja en la figura 11.10, el campo de la etiqueta se compone de los 20 bits más significativos de la dirección lineal y de otros cuatro para los atributos.

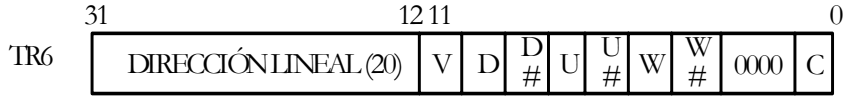


Figura 11.4. Formato del registro de comando TR6, utilizado en el chequeo de la TLB.

TR7 actúa como registro de datos de la prueba y contiene los 20 bits de mas peso de la dirección física, un campo de dos bits llamado REP y el bit HT, que en lectura expresa si hubo presencia (1) o ausencia (0), mientras que en escritura siempre HT=1.

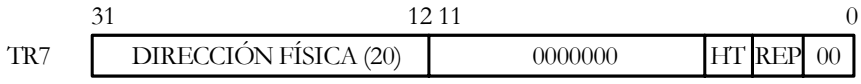


Figura 11.15. Formato del registro de datos TR7, para la comprobación de la TLB.

REP es un campo de dos bits que identifica cuál de los cuatro bloques de ocho entradas, que funcionan en paralelo, es el seleccionado.

Poder conocer la traducción de lineal a física de algunas páginas puede reducir el tiempo de inicio de tareas o procedimientos, esta reducción se produce al escribir la traducción de las páginas mas usadas al inicio de las tareas.

11.9- REGISTROS ESPECÍFICOS

El Pentium incorporó algunos registros específicos como el registro de chequeo de dirección de la máquina, el registro de chequeo del tipo de máquina, los registros de chequeo del TR1 al TR12, el contador de "time-stamp", el registro de selección de control/eventos y los dos contadores 0 y 1.

Paralelamente se incorporaron al repertorio del Pentium las instrucciones que permitían leer (RDMSR) y escribir (WRMSR) los registros específicos.

- **RDMSR:** (Read Model-Specific Register): El valor en ECX especifica uno de los registros de 64 bits específicos del modelo del procesador. El contenido de ese registro se carga en EDX:EAX. EDX se carga con los 32 bits más significativos, mientras que EAX se carga con los 32 bits menos significativos.
- **WRMSR:** (Write Model-Specific Register): El valor en ECX especifica uno de los registros de 64 bits específicos del modelo del procesador. El contenido de EDX:EAX se carga en ese registro. EDX debe contener los 32 bits más significativos, mientras que EAX debe contener los 32 bits menos significativos.

Para referenciar a estos registros se les asigna un numero. Así el 00 es para el registro de chequeo de direcciones de maquina y se usa para conocer las direcciones físicas que provocan un ciclo de bus erróneo. El número 01 es para el registro de chequeo del tipo de máquina y se usa para conocer el tipo de ciclo de bus erróneo. El número 0E Hex es para TR12 y actúa como un registro de control para salto en ciclos especiales. Finalmente el número 10 Hex se asigna al contador "time-stamp" que se usa para leer o escribir el contador interno de 64 bits.

La instrucción RDMSR sólo es ejecutable en Modo Protegido con nivel de privilegio 0.

PUERTAS DE LLAMADA 12

12.1- TRANSFERENCIAS DE CONTROL

Es manifiesta la necesidad de solicitar algunas rutinas o procedimientos del Sistema Operativo desde las aplicaciones. Sin embargo, no se debe permitir que desde una aplicación se pueda bifurcar directamente a un procedimiento del sistema, porque no todos los procedimientos deben quedar accesibles a las aplicaciones; algunos están reservados exclusivamente al sistema. Además, si se pudiera acceder directamente a cualquier punto de entrada del código privilegiado, el resultado y la seguridad del procedimiento sería aleatorio y se resentiría la seguridad del sistema.

Las transferencias de control son los cambios entre segmentos de código, es decir, el paso de un segmento de código, que se está ejecutando por la CPU, a otro segmento de código. Estos cambios se realizan a través de las instrucciones de salto: JMP y CALL/RET. Estas instrucciones, normalmente, tienen 7 bytes:

```
JMP  CS': DESPLAZAMIENTO
```

Figura 12.1.Estructura de las instrucciones JMP, CALL y RET.

Las transferencias de control no pueden incumplir las reglas de acceso entre segmentos de código. La regla que forma parte del sistema de protección entre segmentos se encuentra implementada en hardware del Pentium y de otros procesadores de Intel.

Dicha regla establece que sólo se puede acceder a un segmento de código desde otro segmento de código que pertenezca al mismo nivel de privilegio (PL), es decir, sólo podría ser, en principio, posible una transferencia intranivel.

En el Modo Protegido, hay dos tipos de transferencias de control: Intranivel y Internivel.

TRANSFERENCIAS DE CONTROL INTRANIVEL

El segmento peticionario y el segmento solicitado tienen el mismo nivel de privilegio, ya sea dentro de su área local o de la global.

Mientras una tarea esté activa, la CPU maneja sólo dos tablas de descriptores, la GDT y la LDT de dicha tarea. En las transferencias de control intranivel no hay problemas de corrupción, ya que ambos segmentos tienen el mismo nivel de privilegio. Asimismo el Pentium sólo necesita cargar el selector CS para acceder al segmento de código solicitado. En este caso, el procesador sólo comprueba las reglas referidas al tamaño y al tipo:

- que el desplazamiento que implica el valor de EIP no supere el tamaño del segmento, recogido en el campo LIMITE del descriptor, y
- que el bit E del campo TIPO valga 1.

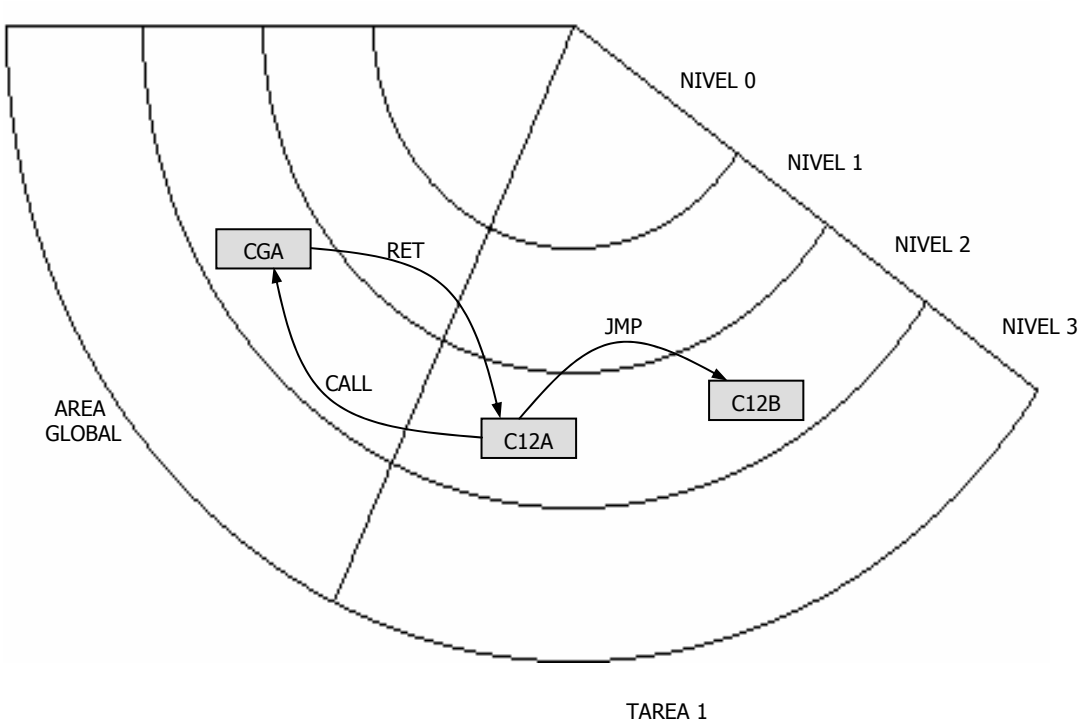


Figura 12.2. Ejemplos de transferencias de control intranivel permitidas.

El campo selector de la figura 12.1 apunta a un descriptor de la tabla GDT o LDT, correspondiente al segmento de código al que se transfiere el control del procesamiento y cuyo DPL debe ser igual al PL del segmento que ha ejecutado la instrucción de transferencia de control (figura 12.2).

Si, al efectuarse la transferencia intranivel, no aparecen errores el contenido del descriptor se carga en el registro caché oculto, asociado al CS.

TRANSFERENCIAS DE CONTROL INTERNIVEL

Se producen cuando el nivel de privilegio del segmento peticionario y el del segmento solicitado son distintos. En principio, es una transferencia de control que no es permitida según las reglas de acceso. Ahora bien, frecuentemente las aplicaciones de bajo nivel de privilegio necesitan con utilizar rutinas del S.O. y para ello se debe acceder a un nivel de privilegio mayor.

Se producen cuando la instrucción de transferencia implica el salto a un código ejecutable de diferente nivel de privilegio que el que hay en curso. Esta posibilidad entraña un alto riesgo de corrupción del sistema y está soportada mediante un recurso denominado Puerta de Llamada, que lo crea el programador del sistema cuando lo juzga conveniente.

La transferencia internivel sólo se puede realizar desde un segmento de código a otro con mayor nivel de privilegio.

En las transferencias de control internivel existe riesgo de corrupción del sistema y para evitarlo, la arquitectura IA-32 tiene grabada en silicio unas reglas que permiten realizar transferencias desde un segmento de código a otro de mayor nivel de privilegio a través de un recurso llamado "Puertas de Llamada" (PLL), que construye a medida el programador de sistemas.

12.2- DEFINICIÓN Y COMPORTAMIENTO DE LAS PUERTAS DE LLAMADA

Para poder acceder desde un segmento de código a otro de mayor nivel de privilegio, los procesadores Pentium utilizan un recurso llamado Puerta de Llamada.

Físicamente, la puerta de llamada consiste en un descriptor local o global, al que se accede a través de una instrucción de formato CALL NOMBRE_PUERTA. El descriptor referenciado por la puerta de llamada permite acceder a un punto de entrada concreto de un segmento de código. La instrucción CALL va asociada con la instrucción RET situada al final del procedimiento accedido. Con RET se devuelve el control al segmento de código original (figura 12.3).

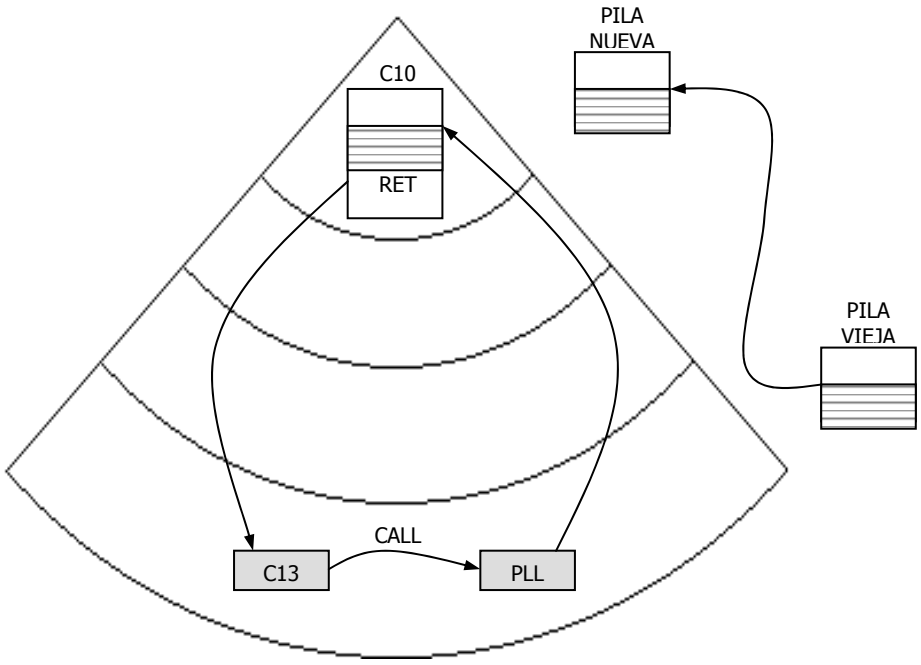


Figura 12.3. Desde un segmento de código ejecutable del nivel 3 se puede acceder aun punto de entrada determinado de un segmento de código del nivel 0, a través de una Puerta de Llamada PLL.

Las Puertas de Llamada, sólo las puede crear el programador de sistemas y es el que otorga esta llave al programador de aplicaciones. Con una Puerta de Llamada se permite acceder a un segmento de código con mayor nivel de privilegio que el del segmento solicitante, pero únicamente por un sitio, es decir, sólo se permite el acceso a un punto de entrada concreto del segmento.

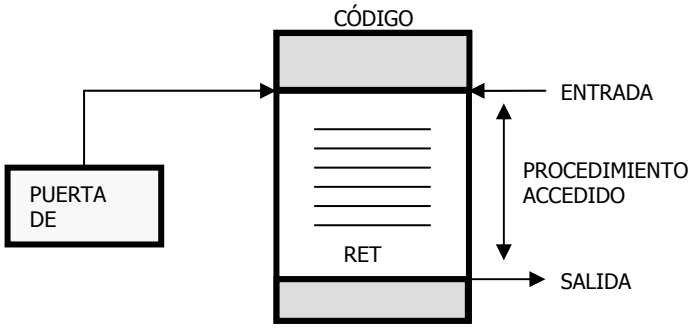


Figura 12.4. Funcionamiento de una puerta de llamada.

En la figura 12.5 se muestra la estructura de un descriptor correspondiente a una puerta de llamada. Obsérvese que en ella, además de los atributos de la puerta, hay un selector y un desplazamiento. Mediante el selector, que se cargará en CS, se determina el nuevo segmento de código, y mediante el desplazamiento se determina el punto de entrada fijo en dicho segmento. El segmento de código peticionario no tiene posibilidad de fijar el punto de entrada, puesto que viene fijado en la propia Puerta de Llamada.

Una Puerta de Llamada (PLL) es un descriptor de 64 bits que puede estar situada en el área global o local, es decir, puede estar en la GDT, o bien, en la LDT.

El descriptor de una Puerta de Llamada tiene básicamente las siguientes funciones:

1. Especifica el segmento de código a acceder.
2. Define un punto de entrada fijo a una rutina determinada del segmento de código al que se accede.
3. Determina el número de parámetros que se copiarán automáticamente de la pila del peticionario a la pila del segmento solicitado.
4. Define los atributos de la Puerta de Llamada.

La estructura del descriptor de una Puerta de Llamada es la siguiente:

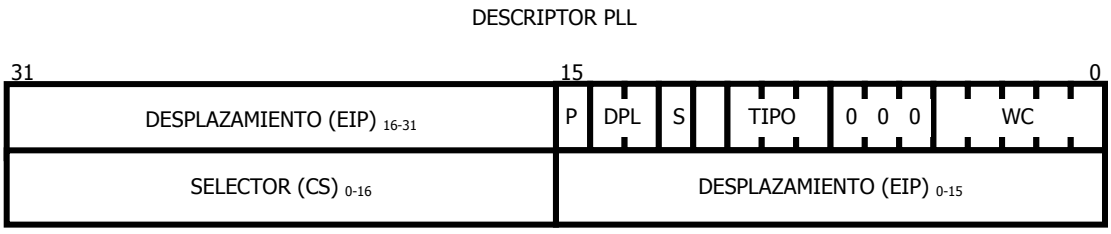


Figura 12.5. Estructura de un descriptor correspondiente a una puerta de llamada. El campo TIPO, para el Pentium, tiene el valor C en hexadecimal.

- **Selector:** El Selector es un valor de 16 bits que se carga en CS y determina el nuevo selector del segmento de código.

- **Desplazamiento:** Determina el punto de entrada fijo en el segmento. Dicha entrada es, generalmente, la primera instrucción de una rutina.

- **Atributos:**

  - **P**: BIT DE PRESENCIA: Indica si el segmento al que referencia el descriptor está cargado en la memoria principal (P=1), o bien está ausente (P=0). Es decir, si dicho descriptor es válido o no.
  - **DPL**: NIVEL DE PRIVILEGIO DEL DESCRIPTOR: Indica el nivel de privilegio de la PLL a la que referencia.
  - **S**: SEGMENTO DEL SISTEMA: Si S=0 indica que se trata de un segmento del sistema. Por el contrario si S=1 se trata de un segmento de código.
  - **TIPO**: Con 4 bits (1100) indica que se trata de un descriptor de Puerta de Llamada. Para el Pentium tiene el valor C en hexadecimal.
  - **WC**: CONTADOR DE PALABRAS: Son 5 bits que indican el número de parámetros que se transfieren desde la pila del nivel del segmento peticionario a la del segmento solicitado. El número máximo de parámetros que se pueden pasar es de 32 dobles palabras (2^5).

Debido a la compatibilidad descendente de todos los procesadores Intel, el Pentium tiene la posibilidad de usar Puertas de Llamada del 80286, que se diferencian en el formato descriptor: por: El campo de $DESPLAZAMIENTO_{16\text{-}31}$ que está reservado, es decir, sus bits están puestos a 0. Además, del campo WC, que indica el número de palabras de 16 bits, no de palabras dobles de 32 bits, que se transfieren como parámetros a través de la puerta de llamada.

FASES DE UNA TRANSFERENCIA POR PLL

La operatividad de una Puerta de Llamada, viene expresada de forma gráfica en la figura 12.6 y consta de las siguientes fases:

- **1ª FASE:** En el programa en curso aparece una instrucción de llamada a una puerta: CALL PLL.

- **2ª FASE:** Se accede al descriptor de la puerta de llamada, en donde se obtiene, junto con los atributos de la puerta, un selector y un desplazamiento.

- **3ª FASE**: El valor del selector de la puerta se carga en CS, con el que se accede a un descriptor de segmento de código cuyo contenido (base, límite y atributos) se carga en el registro caché oculto asociado a CS.

- **4ª FASE:** Mediante los valores de la base y el límite situados en el registro caché de CS se determina el segmento a acceder. Luego la CPU comienza la ejecución de la instrucción situada en la dirección obtenida al sumar a la base el desplazamiento existente en la Puerta de Llamada.

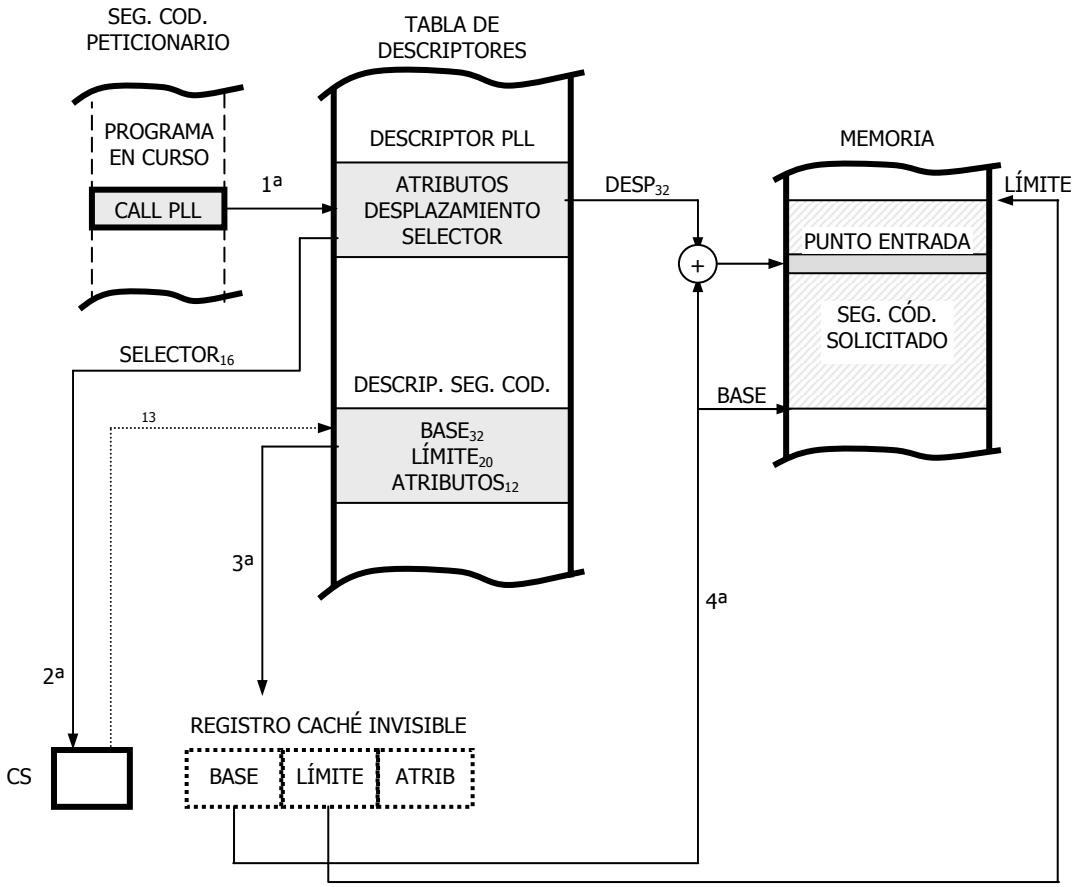


Figura 12.6. Mecanismo para la transferencia de control internivel, utilizando una Puerta de Llamada, PLL.

El acceso a un segmento de código mediante una Puerta de Llamada supone la comprobación de los siguientes niveles de privilegio:

- CPL: nivel de privilegio del segmento de código en curso.
- RPL: nivel de privilegio del segmento peticionario.
- DPL: nivel de privilegio del descriptor de la puerta de llamada.
- DPL del descriptor del segmento solicitado.

También es verificado el bit C (Conforming) del segmento de código destino.

REGLAS DE MANEJO DE LAS PLL

En el manejo de las Puertas de Llamada se debe cumplir tres reglas:

1. Para acceder a una Puerta de Llamada desde un segmento de código, el nivel de privilegio de este último ha de ser igual o mayor que el de la puerta. Esta regla es similar a la expuesta para realizar el acceso a los segmentos de datos.

2. Mediante una Puerta de Llamada sólo se puede transferir el flujo de control a un segmento de código de mayor nivel de privilegio que el peticionario.

3. Cuando se ejecuta una instrucción CALL PLL se guarda en la pila del nivel correspondiente a la PLL (pila vieja) la dirección de retorno CS:EIP al segmento

4. peticionario. También se guardan los parámetros que referencian a la pila vieja SS:ESP y los que indica el WL. Asimismo se realiza una transferencia de datos de la pila vieja a la nueva, es decir, la correspondiente al PL del segmento destino. Efectuada la transferencia a un segmento de código con mayor PL, el retorno al segmento de código primitivo se realiza con la instrucción RET, que recupera de la pila el selector y el desplazamiento iniciales. El retorno sólo se puede llevar a cabo a un segmento con igual o menor PL.

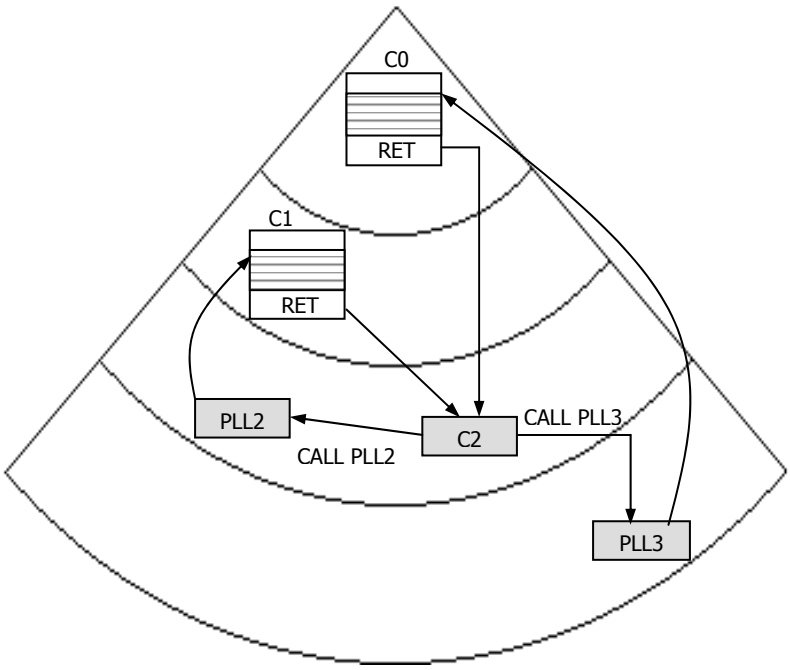


Figura 12.7. Posibles ejemplos de transferencia de control internivel, usando Puertas de Llamada.

Las Puertas de Llamada proporcionan una rápida entrada a puntos concretos del sistema operativo, considerando a éste último como parte de una tarea.

12.3-COMPORTAMIENTO DE LA PILA EN LAS TRANSFERENCIAS INTERNIVEL

En cada PL en donde existe un segmento de código debe existir también un segmento de pila.

Si se compartiese el mismo segmento de pila en una tarea, o sea, la utilizasen todos los segmentos de código situados en los diferentes niveles de privilegio, la pila del sistema operativo podría ser corrompida por el código de aplicación de la tarea.

La regla de acceso a los segmentos de pila exige que el nivel de privilegio del segmento de código coincida con el de la pila.

Cada tarea dispone de un segmento de pila independiente para cada nivel de privilegio en el que residan segmentos de código. Cuando se cambia de nivel de privilegio, se cambia automáticamente de pila, modificándose el contenido de SS y ESP. Este es un recurso suplementario de protección, que evita la mezcla de parámetros procedentes de diversos PL en la pila (figura 12.8).

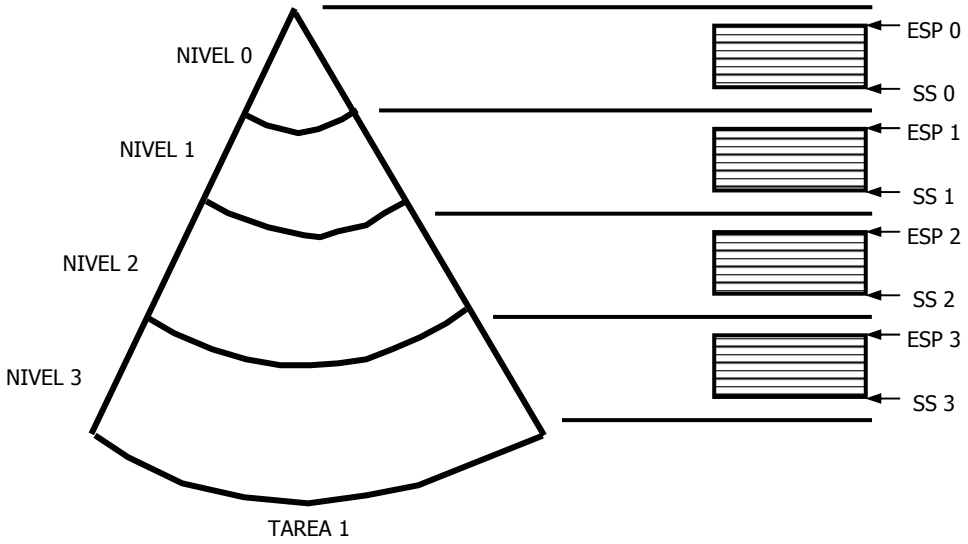


Figura 12.8. Cada tarea dispone de un segmento de pila para cada nivel de privilegio en el que exista código ejecutable.

Los parámetros que definen a los segmentos de pila correspondientes a los diversos niveles de privilegio se hallan guardados en el Segmento de Estado de la Tarea, TSS.

En cuanto se produce una transferencia de control mediante una Puerta de Llamada el procesador realiza un traspaso de parámetros a la pila del segmento accedido. El cambio de pila se ha de realizar para prevenir la mezcla de datos de diferente PL entre rutinas de diferentes niveles de privilegio en el caso en que compartieran la misma pila.

Cuando se produce una transferencia de control internivel, es muy frecuente tener que traspasar o copiar un conjunto de parámetros desde la pila primitiva a la nueva. Mediante el uso de Puertas de Llamada se consigue una autocopia automática de parámetros entre las dos pilas afectadas.

Apréciese en la figura 12.5, que en la estructura del descriptor de puerta de llamada hay un campo de cinco bits denominado Contador de Palabras (WC: Word Counter). En dicho campo se

carga un valor que especifica el número de palabras (Dword), con un máximo de 31, que se van a transferir, desde la cima de la pila en curso a la nueva, situada en otro PL.

En el caso en que se necesite transferir más de 31 parámetros a la rutina destino, uno de dichos parámetros puede ser un puntero a un bloque de datos en memoria.

En realidad, además de copiarse en la pila nueva el número de palabras de la pila vieja indicado en WC, también se salvan:

A) El SS y el ESP de la pila en curso.
B) La dirección de retorno (CS y EIP) del segmento de código peticionario.

En la figura 12.9 se presenta la estructura del segmento de la pila nueva, al que se ha pasado mediante el uso de una puerta de llamada.

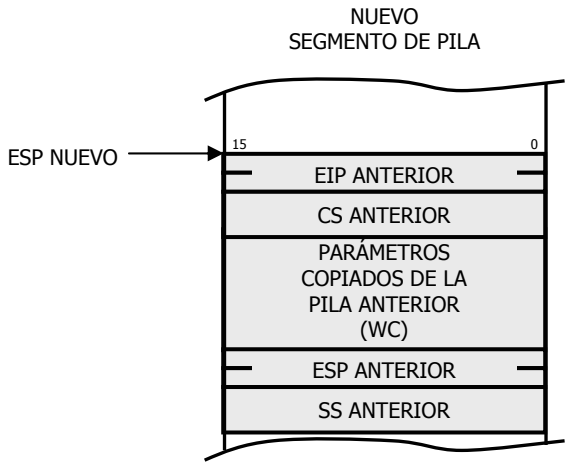


Figura 12.9. Estructura del contenido que se salva en el segmento de pila, al cual se ha accedido por una Puerta de Llamada.

Debe ser el programador de sistemas el encargado de confeccionar las rutinas o procedimientos que pueden ser usados por los módulos de aplicación, así como diseñar las Puertas de Llamada necesarias para acceder a ellas.

La Puerta de Llamada es el mecanismo más seguro y flexible para propiciar a las aplicaciones el acceso al sistema por un punto concreto sin riesgo de degradar los niveles de seguridad. La puerta de llamada puede residir en el espacio local o en el global y, según la diseñe el programador de sistemas, conseguirá:

a) Determinar los módulos de las aplicaciones capaces de acceder a las rutinas del sistema. Para ello debe situar a la PLL en el nivel de privilegio y espacio adecuados.

b) Especificar el punto concreto por el que se permite la entrada a una rutina del sistema, estableciendo el valor del desplazamiento en el descriptor de la PLL.

12.4- EL ESCENARIO DEL CABALLO DE TROYA

Un caballo de Troya es un programa destructivo que aparenta ser legítimo y que con frecuencia se adjuntan en correos, en programas o en juegos de software gratis. Los caballos de Troya pueden encontrar información de contraseñas, hacer a los sistemas más vulnerables a invasiones o sabotear datos en el disco duro de un usuario.

El escenario del caballo de Troya se origina cuando se envía a través de la Puerta de Llamada parámetros prohibidos, de forma que la rutina del nivel 0 permite el acceso a segmentos de datos a los que el segmento peticionario desde su PL no puede acceder.

Las Puertas de Llamada, como se ve, plantean un grave problema al sistema de protección, puesto que su empleo permite aumentar el nivel de privilegio del segmento de código en ejecución (peticionario). Hay que evitar que desde un segmento de código se pueda acceder a un segmento de datos de mayor nivel de privilegio, hecho factible cuando, como paso intermedio, se sube el nivel del código usando una Puerta de Llamada.

Supóngase el caso mostrado en la figura 12.10:

Se ve que hay un segmento de código, perteneciente a una aplicación con PL=3, que a través del mecanismo de transferencia de la Puerta de Llamada PLL, permite acceder al segmento de código del S.O. con PL=0. Un programador podría enviar por medio de la transferencia de parámetros a la pila, utilizando WC (Contador de Palabras), el selector del segmento de datos del sistema y algún dato para escribir en él, con lo que se estaría violando la seguridad del sistema.

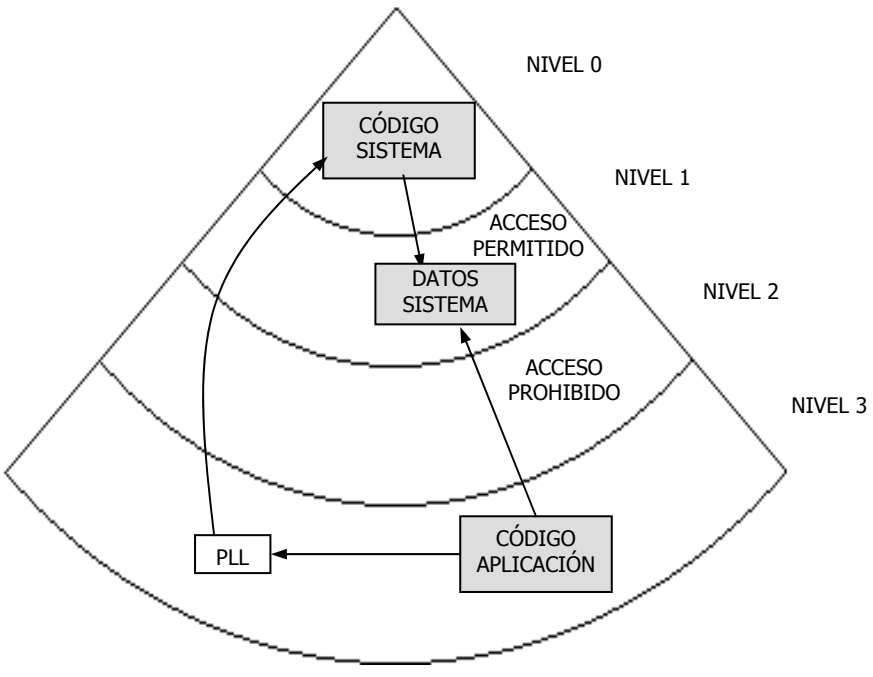


Figura 12.10. Hay que evitar que el segmento "Código Aplicación" pueda acceder en escritura al segmento "Datos Sistema", a través de una Puerta de Llamada.

Para evitar el escenario del caballo de Troya, existe dos recursos fundamentales que se describen en apartados posteriores. Se trata de los "segmentos ajustables" y "la instrucción ARPL".

Si en el establecimiento de las reglas de acceso sólo se comparase el CPL (Current Privilege Level) del segmento de código peticionario con el DPL (Data Privilege Level) del segmento a acceder, podría reconstruirse el escenario del caballo de Troya, que ya se ha explicado.

Para realizarlo se eleva el nivel de privilegio mediante una Puerta de Llamada y se pasan a la pila del nuevo nivel un parámetro, que posteriormente se utiliza como selector, para acceder a un segmento de datos prohibido para el segmento de código inicial, es decir peticionario. Para evitar esta posible transgresión, la comparación del CPL del segmento a acceder se lleva a cabo con el Nivel de Privilegio Efectivo, EPL, cuyo valor es:

$$EPL = Máx\ (CPL,\ RPL)$$

Recuérdese que el campo RPL (Request Privilege Level) ocupa los dos bits de menos peso del selector cargado en el registro de segmento. El RPL es el PL del segmento de código que inicialmente se había solicitado al selector. Es una especie de recordatorio del CPL peticionario del selector. En el caso que se comenta, el valor del RPL del selector de la Puerta de Llamada tendrá de valor tres, al ser éste el CPL del segmento Código Aplicación. Por lo tanto, aunque se accede al segmento "Código sistema" de CPL=0, el EPL resultante será el máximo valor numérico entre RPL y CPL, o sea, tres. Así se evita el acceso al segmento de Datos Sistema con DPL=1.

Con el empleo del EPL se tiene en cuenta el nivel de privilegio del segmento que llama a la Puerta de Llamada, evitando que ningún elemento pueda aumentar su nivel de privilegio, de forma indirecta.

Las instrucciones de retorno RET, sólo pueden disminuir el nivel de privilegio en la transferencia de control y durante su ejecución se examinan los registros de segmento de datos y se ajusta el nivel de privilegio al del código al que ha retornado.

12.4.1- Segmentos ajustables

Reciben el nombre de segmentos ajustables, aquellos segmentos de código que tienen el bit C (conforming o ajustable) a 1. Su característica fundamental es que pueden ser llamados directamente desde otro segmento de código con igual o menor nivel, ejecutándose con el mismo nivel que el del segmento peticionario.

Un segmento ajustable ajusta su PL al del segmento de código desde el que se transfiere el control de la CPU.

Mediante los segmentos ajustables, muchas funciones generales, como transformaciones matemáticas, pueden ser accedidas desde cualquier nivel de privilegio directamente (sin Puerta de Llamada) y ser ejecutados con el mismo nivel que el del segmento que lo ha solicitado, evitando que se reproduzca el escenario del caballo de Troya (figura 12.11).

Todos los segmentos que tiene el bit C=1, al ser accedidos desde otro segmento de código, cambian su PL a un nivel que se llama "efectivo" (EPL). El EPL es igual al mayor numérico entre el nivel del segmento peticionario y el nivel del segmento solicitado.

$$EPL = (CPL,\ RPL)_{\text{MAYOR NUMÉRICO}}$$

El problema de hacer ajustables los segmentos es la degeneración del nivel de privilegio dando lugar a que, durante el intervalo de tiempo en que la puerta de llamada esté activa, el segmento de código degenerado de nivel pueda quedar corrompido.

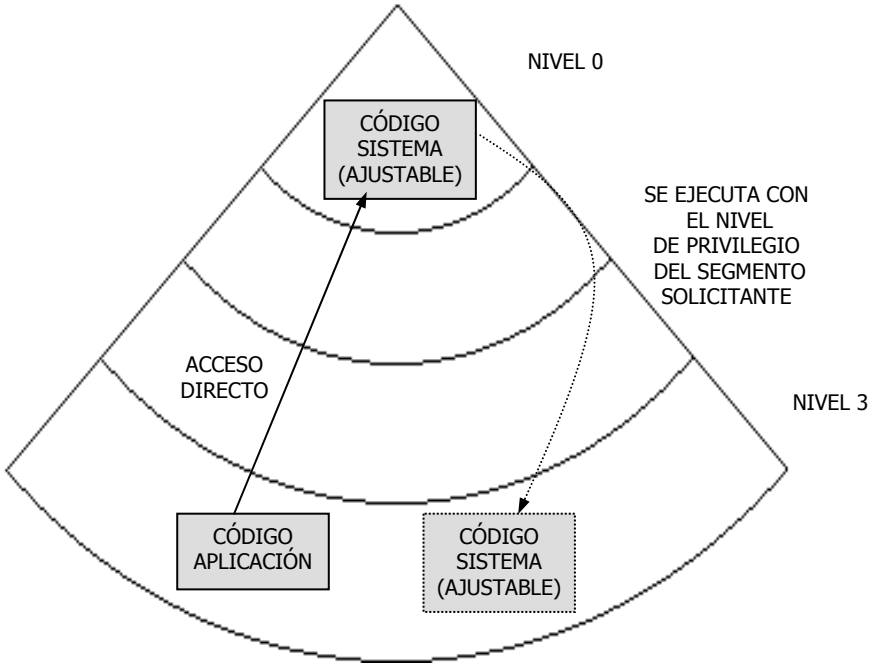


Figura 12.11. Cuando un segmento es ajustable, se ejecuta con el mismo nivel de privilegio que el que tiene el segmento peticionario.

Para transformar un segmento de código en ajustable, basta con poner a 1 el bit C ("Conforming") de los atributos de su descriptor, como se indica en la figura 12.12.

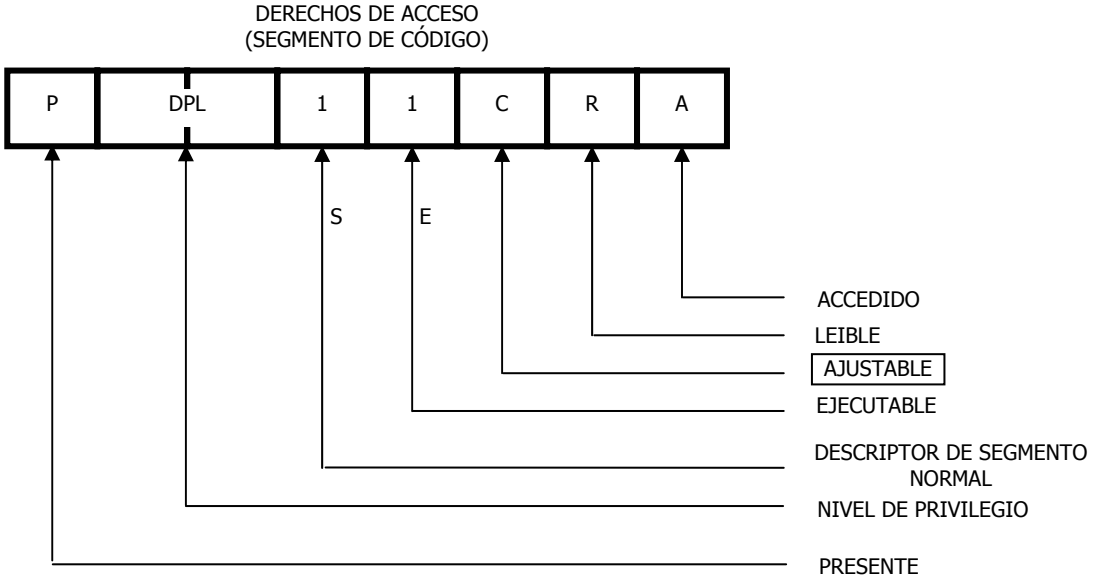


Figura 12.12. El bit C de los derechos de acceso de un descriptor de segmento de código, indica si dicho segmento es ajustable (C=1).

12.4.2- Instrucción ARPL

La instrucción ARPL (Adjust Requested Privilege Level) sirve para cargar en el campo RPL de un selector, el valor EPL, que es el máximo valor numérico entre el CPL peticionario y el RPL del selector a ajustar.

```
EPL= Máx (CPL, RPL).
```

Es una instrucción muy eficaz para aquellos casos en los que un mismo selector se utiliza en diversos niveles de privilegio, evitando que se pueda acceder a segmentos de datos prohibidos.

El formato de la instrucción es:

```
ARPL (SELECTOR A AJUSTAR),(SELECTOR CS DEL PETICIONARIO)
```

Los dos selectores que manipula ARPL están ubicados en los registros de la CPU, por ejemplo, ARPL CX, DX.

En la figura 12.13 se muestra el contenido de CS y DX antes de realizar la instrucción y la forma en que se altera el contenido de CX, en donde los dos bits que conforman su campo RPL, toman el valor del EPL calculado en la instrucción.

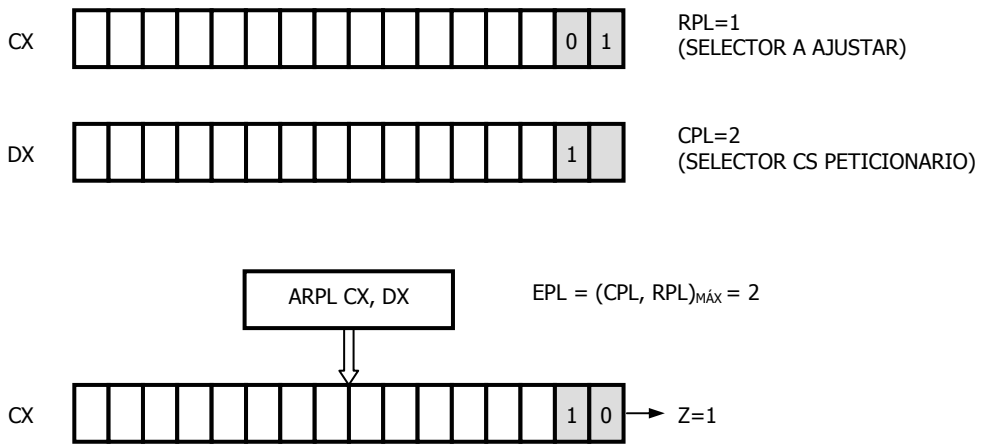


Figura 12.13. Comportamiento de la instrucción ARPL que modifica el valor RPL del selector a ajustar, con el del EPL calculado.

Cuando al ejecutase la instrucción ARPL, se produce un cambio en el campo RPL del selector a ajustar, el señalizador Z pasa a valer 1, lo que indica que el RPL del selector a ajustar es menor que el CPL del selector CS peticionario, de esta forma el sistema se da cuenta de que se quería hacer trampa.

Siempre que sea llamado un procedimiento, el CPL del segmento que llama se almacenará en el campo RPL del selector CS de la dirección de retorno (figura 12.14).

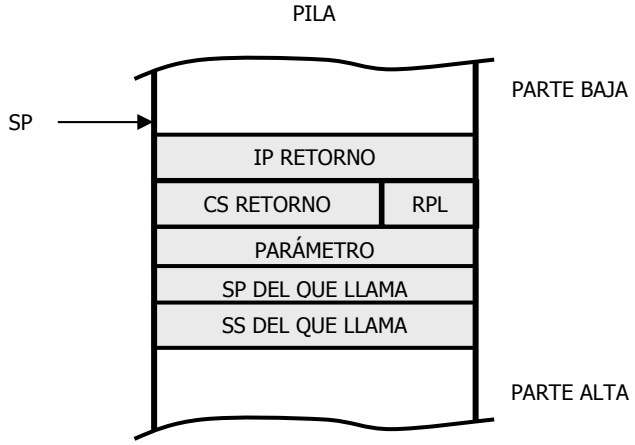


Figura 12.14. El CPL del segmento que llama es salvado en el campo RPL del selector CS del retorno.

Se propone un ejemplo de aplicación de la instrucción ARPL en la posible validación de la carga de un selector en un segmento de datos DS en donde se manejan parámetros de 16 bits. Véase la figura 12.15.

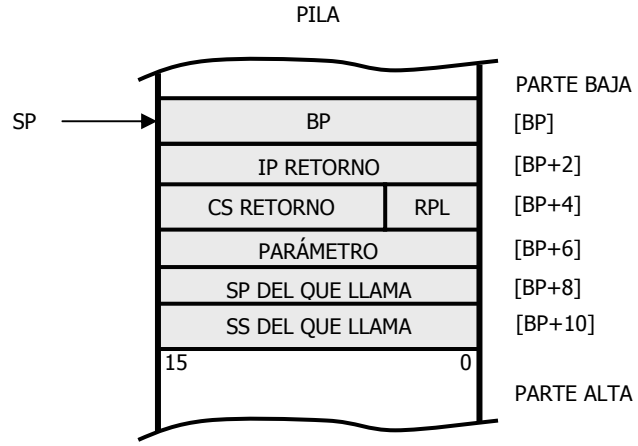


Figura 12.15. Estructura de la pila cuando se emplea la instrucción ARPL para la validación.

Obsérvese en el programa confeccionado en lenguaje Ensamblador, cómo se trata de cargar un parámetro pasado por la Puerta de Llamada correspondiente a un segmento de datos que se quiere cargar en DS. La instrucción ARPL previa evita que se plantee el escenario del caballo de Troya.

```
PROCED_A    PROC  FAR    WC (1)
                                    PUSH        BP
                                    MOV         BP, SP
                                    MOV         AX, (BP+6)
                                    ARPL  AX, (BP+4)
                                    MOV         DS, AX
(si Z=1, ha cambiado el PL)
                                    -------
            -------
            POPBP
            RET2
PROCED_A    ENDP
```

12.5- DESCRIPTORES ALIAS

Las tablas de descriptores GDT, LDT, IDT, etc, son consideradas por el sistema de protección del Pentium como si se tratase de segmentos de código, lo que significa que su contenido no se puede modificar, pues en ese tipo de segmento siempre está prohibida la escritura.

El S.O. debe actualizar los contenidos de los descriptores de forma que, si un descriptor marca como "no presente" al segmento que referencia, cuando se direcciona, el S.O. lo carga en la memoria principal y, luego, deberá actualizar el bit P poniéndolo a 1; además, debe introducir en los campos base, límite y atributos los valores correspondientes a la zona de memoria usada. Esto significa que el S.O. debe escribir en la tabla de descriptores.

También al bit A (Accedido) del campo de atributos del descriptor debe ponerlo a 0, periódicamente, el S.O., para controlar el número de veces que ha sido usado y poder aplicar los algoritmos de intercambio de segmentos.

Para superar la prohibición de escritura en las tablas de descriptores, el S.O. crea dinámicamente descriptores alias, que son "copias" de los segmentos para los que se desean modificar sus derechos de acceso. El descriptor alias asigna al segmento el carácter de "datos", para poder escribir en ellos. Para limitar el acceso a estos descriptores al S.O., se les sitúa en el nivel de privilegio 0 (figura 12.16).

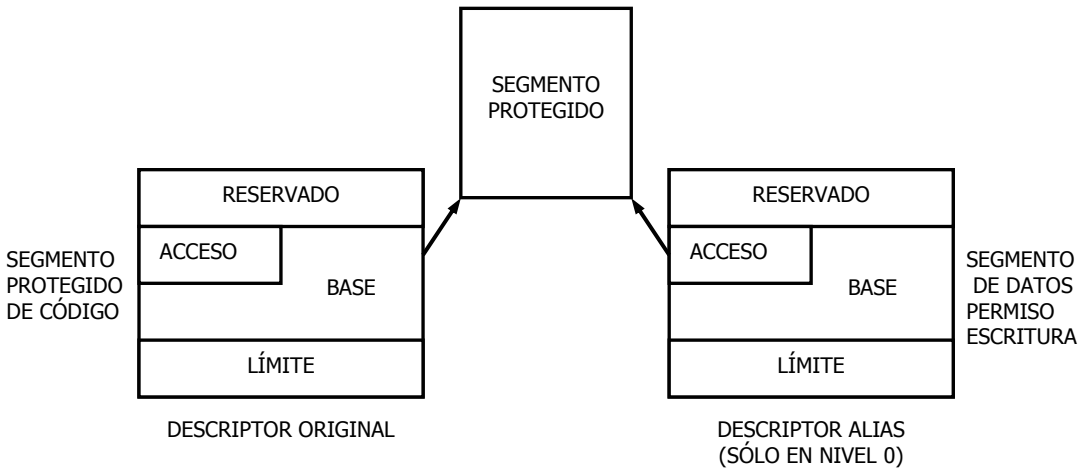


Figura 12.16. El S.O. puede crear descriptores alias, que dan al segmento que referencian el carácter de segmento de datos, en los que es posible escribir y modificar su contenido.

El empleo de los descriptores alias permite el manejo de un segmento como si fuese de código contemplado desde la aplicación, mientras que desde el S.O. se manipula como si se tratase de un segmento de datos (figura 12.17).

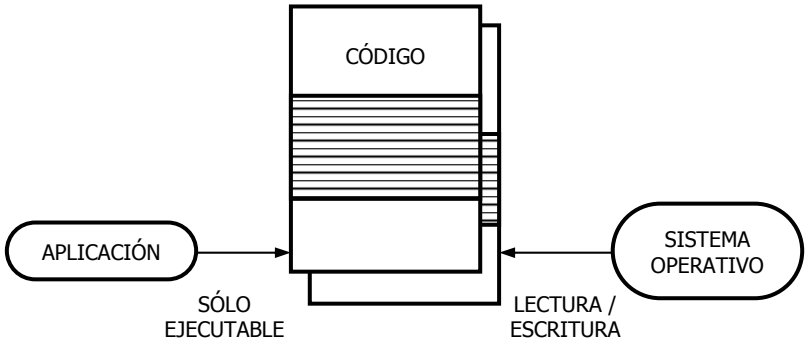


Figura 12.17. Utilización del descriptor alias.

También es posible asignar derechos de acceso diferentes a distintos usuarios empleando este tipo de descriptor (figura 12.18).

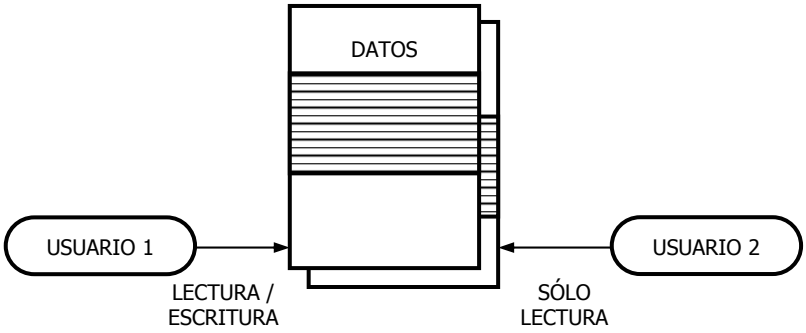


Figura 12.18. Según el usuario que maneja el segmento, cambian los derechos de acceso.

Finalmente, otra interesante aplicación de los descriptores alias es cuando se colocan en la GDT y permiten acceder a todos los elementos que referencian desde cualquier parte del sistema (figura 12.19).

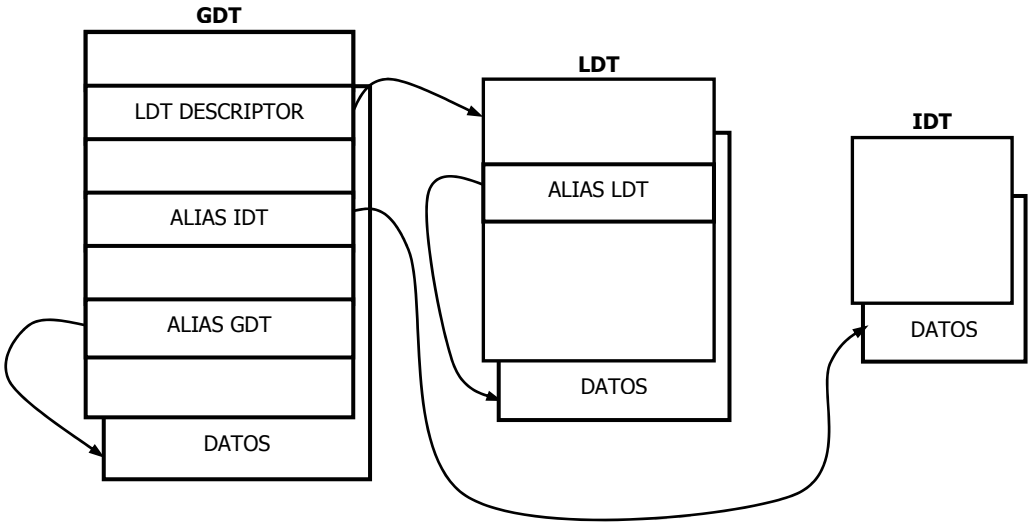


Figura 12.19. Los descriptores alias en la GDT son siempre accesibles.

12.6- PARTICULARIDADES DE LOS SEGMENTOS DE PILA

La pila es una zona cualquiera de la memoria, con estructura LIFO. El control de la pila es soportado directamente por el procesador y se emplea para:

- Almacenar la dirección de retorno en subrutinas e interrupciones.
- Espacio de trabajo de un procedimiento.
- Para paso de parámetros.

El mecanismo de direccionamiento de la pila se muestra en la figura 12.20, donde se aprecia que el puntero ESP, crece hacia la base del segmento de pila direccionada por SS. Se trata de un crecimiento hacia abajo, es decir, hacia direcciones inferiores.

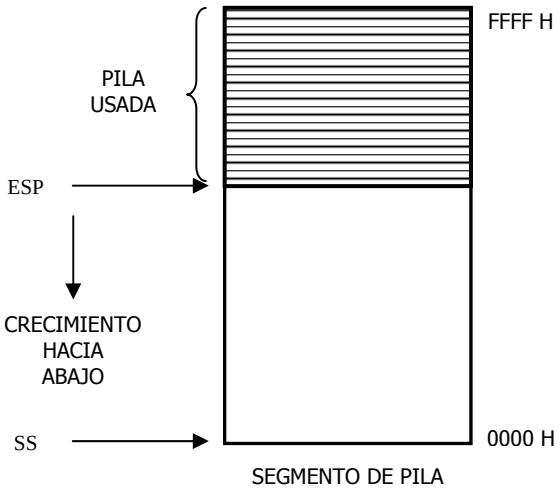


Figura 12.20. El ESP crece hacia el SS, o sea, la pila va creciendo hacia direcciones numéricamente inferiores o decrecientes.

La manipulación de la pila puede dar lugar a dos tipos de fallos o errores:

1. **Desbordamiento positivo de la pila**.
   Cuando se excede el límite máximo asignado al segmento de pila.

2. **Desbordamiento negativo de la pila.**
   Suele consistir en un error de programa, en el que se intercambian instrucciones PUSH por POP, dando lugar a que la pila se desborde por el límite opuesto al caso anterior.

Cualquiera de los dos errores típicos de pila compromete la integridad del sistema.

El vector de excepción número 12 se dedica a resolver los fallos provenientes del manejo de la pila. En caso de desbordamiento, la rutina de excepción puede encargarse de aumentar el límite del segmento. También se utiliza este vector cuando el segmento no está presente. En ambas situaciones, se salva en la pila de la rutina de excepción el selector del código de error, que contiene el segmento que ha producido el fallo.

En la figura 12.21 se muestra la diferencia fundamental entre el funcionamiento de un segmento de pila convencional y otro con expansión hacia abajo (expand down). En este último caso, detrás del límite del segmento de pila existe una zona de memoria en la que es posible el crecimiento de la pila sin necesidad de modificar el valor de SS.

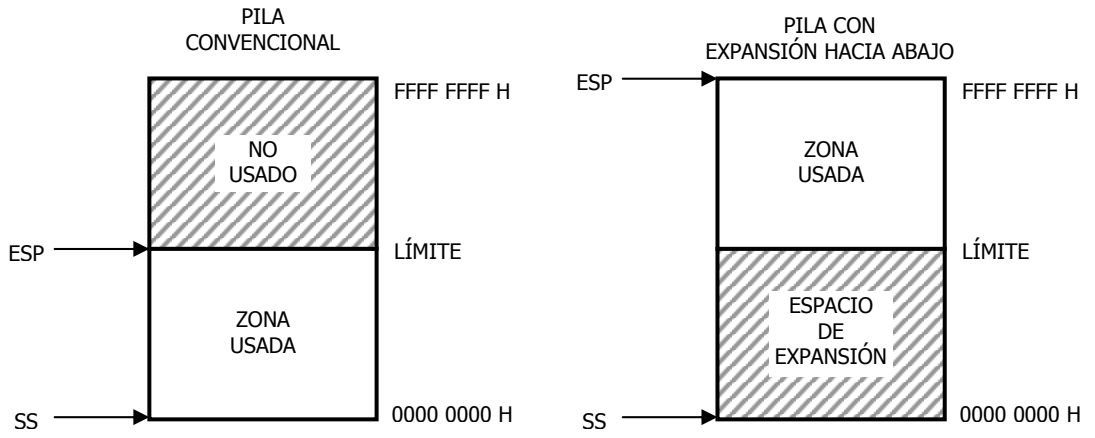


Figura 12.21. Diferencia entre el comportamiento de una pila convencional y otra con expansión hacia abajo.

En los atributos de los descriptores de segmentos de datos existe un bit DE (Expand Down), que indica si el crecimiento de las direcciones implica el aumento de su valor (ED=0), o por el contrario, su disminución (ED=1).

Si ED=0, se debe cumplir que el DESPLAZAMIENTO ha de ser igual o menor que el LIMITE.

Si ED=1, el desplazamiento puede tener un valor mayor que el límite contenido en el descriptor.

Cuando se trata de un segmento de datos, con ED=1, el S.O. sólo debe cambiar el valor del límite (actualizando el descriptor) para ampliar la capacidad de la pila. Los desplazamientos no se alteran como consecuencia de dicha ampliación. Con esta posibilidad, es posible usar pilas pequeñas pero ampliables cuando se haga necesario (figura 12.22).

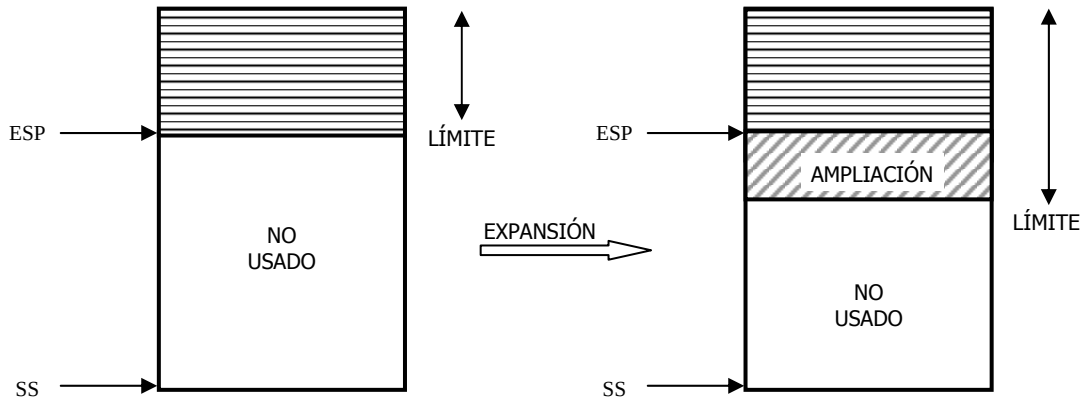


Figura 12.22. El sistema operativo sólo debe cambiar el valor del límite cuando se desea ampliar la capacidad de la pila.

CONMUTACIÓN DE TAREAS 13

13.1- IMPLANTACIÓN DE LA MULTITAREA

Una tarea es un conjunto de segmentos u objetos ubicados en memoria, que, al ser procesados por la CPU, producen un resultado final, que es el objetivo de la tarea.

Un sistema multitarea permite al procesador atender simultáneamente a varias tareas independientes, de tal forma que, cuando abandona una tarea, pasa al procesamiento de otra. Si la CPU es muy rápida y la operación de conmutar tareas dura poco, parece como si cada tarea tuviese permanentemente a su disposición la CPU.

Para un programador, la multitarea es el procesamiento discontinuo de varias tareas que aparentemente se ejecutan de forma paralela.

Una conmutación de tarea consiste en abandonar el procesamiento de la tarea en curso (vieja) para reanudar otra tarea (nueva).

Para reanudar una tarea en cualquier momento, hay que disponer de algún objeto que almacene el estado completo del procesador cuando abandonó anteriormente dicha tarea. Dicho estado se compone, fundamentalmente, del contenido de la mayor parte de los registros de la CPU, y recibe el nombre de "contexto". Debe contener la LDT específica, el registro que apunta al Directorio de Páginas, los punteros de pila para los diversos niveles de privilegio, los restantes registros del procesador, parámetros del S.O., etc.

Cada tarea tiene guardado el contexto de la CPU que le corresponde en un segmento que se denomina Segmento de Estado de la Tarea, TSS.

El procesador tiene cuatro registros para la gestión de memoria:

- GDTR (*global descriptor table register*): contiene los 32 bits de la dirección base y los 16 bits del límite de tabla de la GDT.
- LDTR (*local descriptor table register*): contiene los 16 bits del selector de segmento, 32 bits de la dirección base, 16 bits del límite de segmento y atributos del LDT.
- IDTR (*interrupt descriptor table register*): contiene los 32 bits de dirección base y 16 bits del límite de tabla del IDT (tabla de interrupciones).
- TR (*task register*): contiene 16 bits del selector de segmento, 32 bits de la dirección base, 16 bits del límite de segmento y atributos del TSS (segmento del estado-tarea) de la tarea actual.

13.2- SEGMENTO DE ESTADO DE LA TAREA, TSS.

El segmento TSS, como es específico para cada tarea, se define mediante valores de selector almacenado en los registros de segmento del sistema.

En la figura 13.1 se muestra la estructura mínima del TSS, para una CPU de 32 bits, compatible con tareas de procesadores de 16 bits. Además, puede contener otras informaciones auxiliares complementarias, que precisa el S.O. para reanudar la tarea.

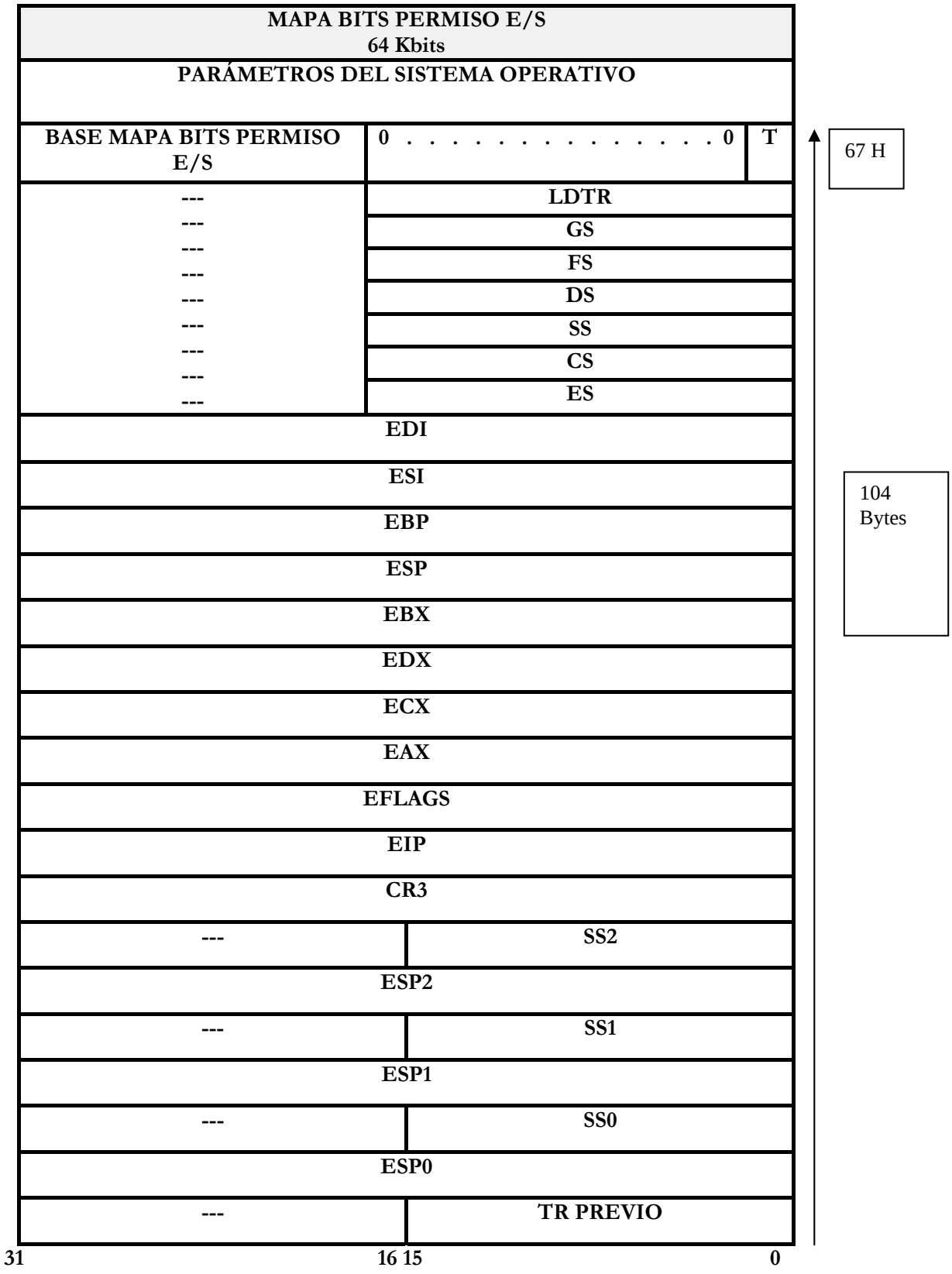


Figura 13.1. Estructura del segmento TSS. Dispone de una capacidad mínima de 104 bytes, que se precisan para guardar el contexto de la CPU.

Los campos son:

1. Registros de propósito general. El estado de los registros EAX, ECX, EDX, EBX, ESP, EBP, ESI y EDI antes de la conmutación de tarea.

2. Selectores de segmento. Los selectores de segmento almacenados en los registros ES, CS, SS, DS, FS y GS previos a la conmutación de tarea.

3. El registro EFLAGS. Estado del registro EFLAGS antes de la conmutación de tarea.

4. Campo EIP. Estado de EIP antes de la conmutación.

5. El TR previo. El selector de segmento TSS de la tarea previa. (Actualiza la conmutación de tarea que comenzó por una instrucción CALL, una interrupción, o una excepción). Este campo permite la conmutación a la tarea previa mediante una instrucción IRET.

6. Selector de segmento LDTR.

7. Registro de Control CR3. Contiene la base del directorio de páginas a ser usada por la tarea. El registro de control CR3 también es conocido como el registro base del directorio de páginas.

8. Punteros de la pila de nivel de privilegio 0, 1 y 2. Los punteros de la pila consisten en una dirección lógica compuesta por el selector de segmento para el segmento de pila (SS0, SS1, y SS2) y un desplazamiento en la pila (ESP0, ESP1, y ESP2). Los valores en estos campos son estáticos para una tarea particular, teniendo en cuenta que los registros SS y ESP cambiarán si la conmutación de la pila ocurre dentro de la tarea.

9. Flag T (Flag de depuración) (byte 100, bit 0). Cuando esta activo y se produce una conmutación de tarea, el flag T causa en la CPU una excepción de depuración.

10. Base del mapa de bits de E/S. Contiene el desplazamiento de 16 bits que hay que sumar a 67H para apuntar al mapa de bits de permiso de E/S.

11. Parámetros del S.O.

En la figura 13.1 no están incluidos en el contexto de la CPU, los valores que referencian al segmento de pila de nivel de privilegio 3, o sea, el valor de los registros SS3 y ESP3. La razón es la siguiente:

1. Si cuando se está trabajando en el nivel de privilegio 3, se guarda el contexto del procesador, los valores SS3 y ESP3 son los que están en curso, y por tanto, están salvados en los registros generales SS y ESP.

2. Si el momento en el que se salva el contexto de la CPU tiene un nivel de privilegio diferente del 3, no es necesario conocer los valores de SS3 y ESP3, pues resulta imposible rebajar el nivel de privilegio del código.

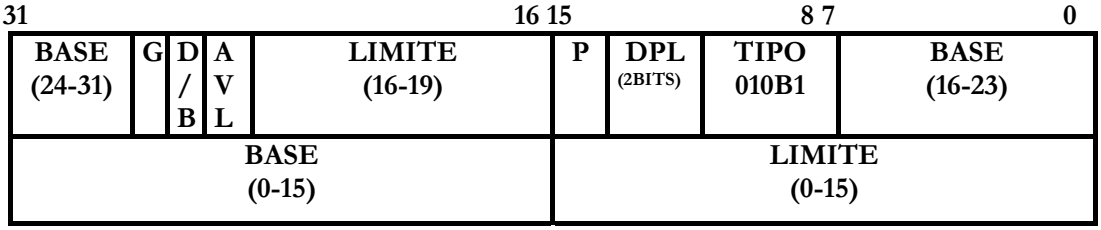
La conmutación de tarea se llevará a cabo de una manera más eficiente si las páginas que contienen estas estructuras también están presentes en la memoria antes de que la conmutación comience.

La CPU hace referencia al TSS por medio del registro TR, Registro de Tarea, que actúa como un selector de la GDT, seleccionando uno de sus descriptores que corresponden con TSS. Se trata de uno de los llamados segmentos del sistema.

El flag BUSY (B) en el campo TIPO indica si la tarea está ocupada. El campo TIPO de los atributos tienen el código 010B1. El bit B indica que la tarea está libre (B = 0) u ocupada (B = 1) (figura 13.2).

El procesador usa el flag BUSY para detectar una llamada a una tarea cuya ejecución se ha interrumpido. Para asegurar que hay sólo un flag BUSY asociado con una tarea, cada TSS debe tener solamente un descriptor TSS que apunte a él.

DESCRIPTOR DE UN SEGMENTO DE ESTADO DE TAREA.



AVL	Libre para ser usado por el software del sistema.
B	Flag BUSY.
BASE	Base del segmento.
DPL	Nivel de privilegio del descriptor.
G	Granularidad.
D/G	Defecto/Grande.
LIMITE	Límite.
P	Presencia
TIPO	010B1

Figura 13.2. Estructura de un descriptor de TSS situado en la GDT. Si el campo TIPO tiene de valor 9, se trata de una tarea libre y si es B, es una tarea ocupada.

La base, el límite, los campos DPL, la granularidad y el flag de presencia tienen funciones similares de uso en descriptores de segmentos de datos. Cuando el flag G es 0 en un descriptor de un TSS de 32 bits, el límite debe tener un valor igual o mayor que 67H, un byte menos del tamaño mínimo de un TSS.

El hecho de intentar conmutar a una tarea cuyo descriptor TSS tiene un límite menor de 67H genera una excepción, TSS no válido. Se requiere un límite más grande si el mapa de bits de permiso de E/S es incluido en el TSS. Se requeriría un límite aun más grande si el sistema operativo guardara datos adicionales en el TSS. La CPU en una conmutación de tarea no verifica límites mayores de 67H, sin embargo, lo hace cuando accede al mapa de bits de permiso de E/S o redirecciona la interrupción del mapa.

Cualquier programa o proceso que tenga acceso a un descriptor TSS (cuando su CPL es numéricamente igual o menor al DPL del descriptor de TSS) puede avisar a la tarea con una CALL o una JMP.

En la mayoría de los sistemas, el DPL de los descriptores TSS deben ponerse con valores menores de 3 para que solamente el software con privilegios pueda realizar la conmutación de tarea. Sin embargo, en las aplicaciones multitarea, el DPL de algunos descriptores TSS puede ponerse a 3 para permitir la conmutación de tarea del nivel de privilegio de la aplicación.

13.3- EL REGISTRO DE TAREA, TR.

El registro de tarea es un segmento de 16 bits que apunta de forma indirecta al TSS. Con esos 16 bits apunta a un descriptor de la GDT del cual se obtiene la base y el límite del TSS, que se cargan en un registro caché oculto asociado al TR. Guardar estos valores en un registro caché hace que la ejecución de la tarea sea más eficiente, porque el procesador no necesita ir a la memoria a por estos valores para referirse al TSS de la tarea actual.

Las instrucciones LTR (carga del registro TR) y STR (almacenamiento de TR) cargan y leen la parte visible del registro de la tarea. La instrucción LTR carga un selector de segmento (operando fuente) en el registro de tarea que apunta a un descriptor TSS en la GDT, y entonces se carga la parte invisible del registro de tarea con información del descriptor TSS. Esta instrucción es una instrucción privilegiada que sólo puede ejecutarse cuando el CPL es 0. La instrucción LTR generalmente se usa durante la inicialización del sistema para poner un valor inicial en el registro de tarea. Después, los contenidos del registro de tarea son cambiados implícitamente cuando ocurre una conmutación de tarea.

La instrucción STR (almacenamiento de TR) guarda la parte visible del registro de tarea en un registro de propósito general o en memoria. Esta instrucción puede ejecutarse desde cualquier nivel de privilegio para identificar la tarea que está corriendo actualmente, sin embargo, normalmente se usada sólo por el software del sistema operativo. Al arrancar o resetear el procesador, el selector de segmento y la base un cargan con el valor por defecto (predefinido 0) y el límite es puesto a FFFFH.

En la figura 13.3 se representa gráficamente el mecanismo que permite a la CPU direccionar al TSS a través del Registro de Tarea (TR).

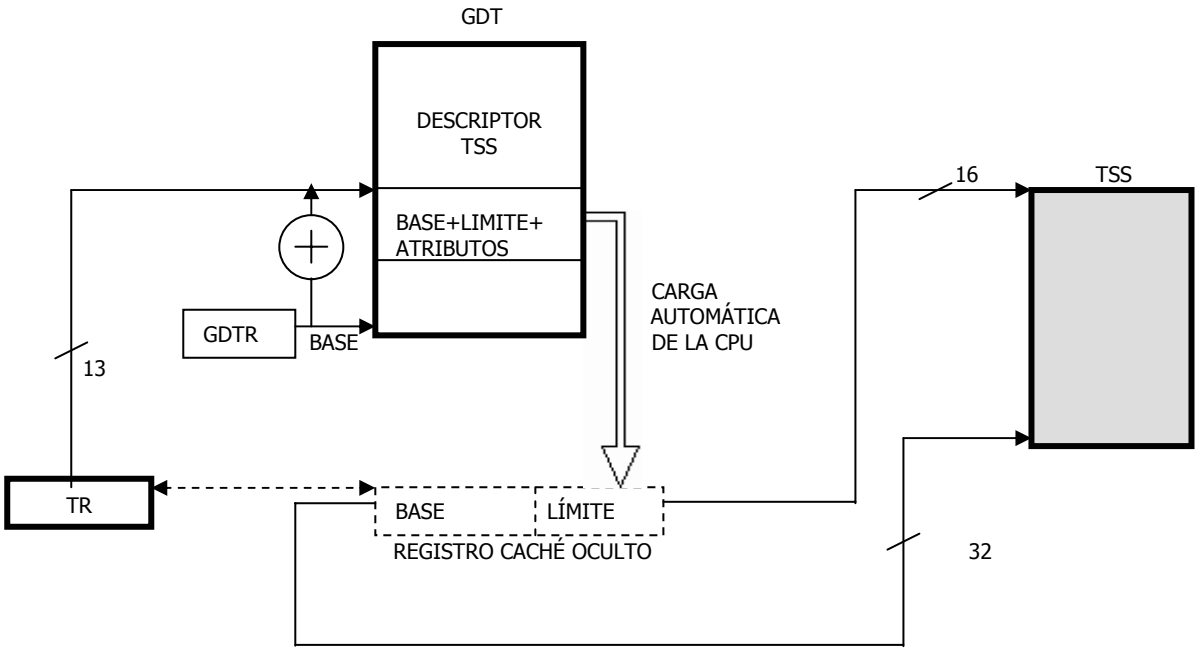


Figura 13.3. Descripción gráfica del mecanismo de direccionamiento del segmento TSS.

13.4- CONMUTACIÓN DE TAREA

El Pentium no dispone de instrucciones específicas para provocar la conmutación de tarea. Utiliza JMP y CALL referidos al nuevo TR desde segmentos de código que deben poseer el mismo nivel de privilegio que el TSS, lo cual supone la restricción de poder realizar únicamente la conmutación desde el máximo nivel de privilegio. Como se verá más adelante, es posible realizar una conmutación de tarea mediante una interrupción.

La CPU transfiere la ejecución a otra tarea en cualquiera de estos cuatro casos:

1. El programa actual, la tarea, o el proceso ejecuta una JMP o instrucción CALL a un descriptor de TSS en la GDT.

2. El programa actual, la tarea, o el proceso ejecuta un JMP o instrucción CALL a un descriptor de puerta de tarea en la GDT o la LDT actual.

3. Una interrupción o el vector de la excepción apunta a un descriptor de la puerta de tarea en el IDT.

4. La tarea actual ejecuta un IRET cuando el flag NT situado en el registro EFLAGS se pone a uno.

La conmutación de tarea se compone de cuatro etapas fundamentales:

1. En la tarea vieja se ejecuta una instrucción JMP o CALL con el TR_NUEVO. Las dos instrucciones hacen lo mismo, la diferencia está en la CALL que incluye anidamiento de tarea.

2. Se carga en TR el valor del TR_NUEVO.

3. Se salva el contexto de la CPU en el TSS_VIEJO.

4. Se carga el TSS_NUEVO como contexto de la CPU.

En la figura 13.4 se muestra gráficamente el mecanismo simplificado de la conmutación de tarea.

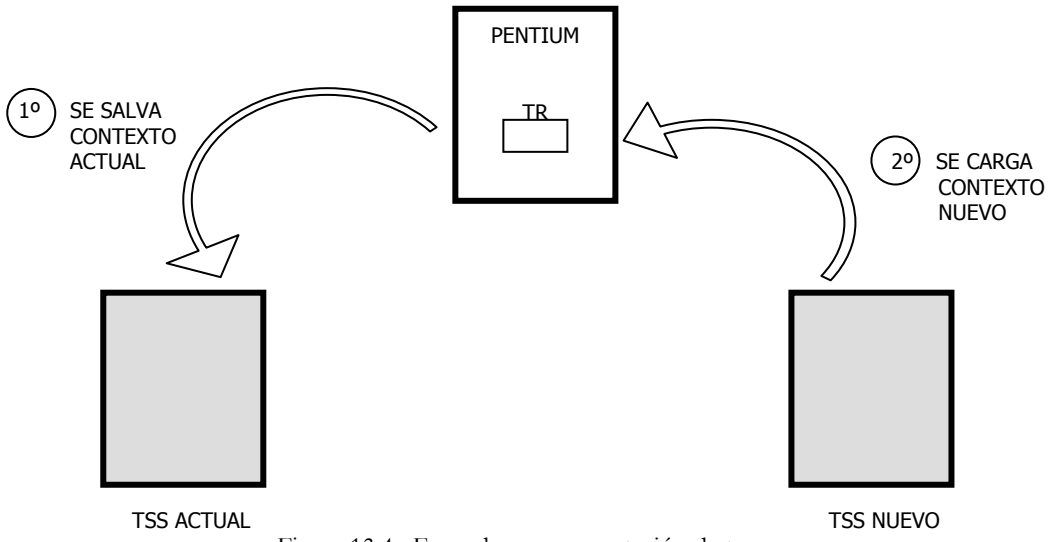


Figura 13.4. Fases de una conmutación de tarea.

Las instrucciones JMP, CALL, e IRET, así como las interrupciones y excepciones, son mecanismos generales para redirecionar un programa. La referencia a un descriptor TSS o una

puerta de tarea (cuando llama o salta a una tarea) o el estado del flag NT (al ejecutar una instrucción IRET) determina si ocurre una conmutación de tarea. La CPU realiza las siguientes operaciones al cambiar a una nueva tarea:

1. Obtiene el selector del TSS nuevo del operando de la instrucción, de una puerta de tarea, o del TSS previo, para una conmutación de tarea que comienza con una instrucción IRET.

2. Comprueba que la tarea actual (vieja) permite conmutar a la tarea nueva. de. El CPL de la tarea actual (vieja) y el RPL del selector de segmento para la tarea nueva deben ser menor o igual al DPL del descriptor TSS o puerta de tarea a la que nos referimos.

3. Comprueba que el descriptor TSS de la tarea nueva está presente y tiene el límite válido (mayor que o igual a 67H) , que la nueva tarea está disponible y que el TSS actual (viejo), el TSS nuevo, y todos los descriptores de segmento usados en la conmutación de tarea están dentro la memoria del sistema.

4. Guarda el estado de la tarea actual (vieja) en su TSS asociado. La CPU encuentra la base del TSS actual mediante el registro de tarea y entonces copia en el TSS el estado de los siguientes registros: registros de propósito general, selectores de los registros de segmento, la imagen del registro EFLAGS, y el registro de instrucción del puntero (EIP).

5. Carga el registro de tarea con el selector de segmento y descriptor para el TSS de la tarea nueva.

6. Carga el estado de la tarea nueva en la CPU. Cualquier error asociado con la carga y verificación del descriptor de segmento ocurre en el contexto de la tarea nueva. La información del estado de la tarea incluye el registro LDTR, registro de control CR3, registro EFLAGS, registro EIP, registros de propósito general, y las partes del descriptor de segmento de los registros de segmento.

7. Empieza a ejecutar la nueva tarea.

El estado de la tarea en curso siempre es guardado cuando la conmutación de tarea ocurre satisfactoriamente. Si la tarea se reanuda, la ejecución empieza con la instrucción apuntada por el valor EIP guardado, y los registros se restauran a los valores que tenían antes de que se suspendiera.

En la conmutación de tareas, el nivel de privilegio de la nueva tarea no hereda su nivel de privilegio de la tarea suspendida. La nueva tarea empieza ejecutándose en el nivel de privilegio especifico en el CPL del registro CS que está cargado desde el TSS. El software no necesita realizar verificaciones de privilegios explícitos en un conmutación de tarea debido a que las leyes de privilegio controlan el acceso al TSS.

La duración de una conmutación de tarea es de pocos microsegundos, y, en el caso de ser originada mediante una instrucción CALL, se guarda en el TSS nuevo el TR viejo (TR PREVIO en la figura 13.1) para regresar a la tarea peticionaria de la conmutación, con la instrucción de retorno.

La localización en memoria de los recursos destinados a cada tarea se consigue con el cambio automático del contenido de los registros LDTR y CR3, que apuntan a la tabla de descriptores locales y a la base del Directorio de las Tablas de Páginas.

13.5- PUERTAS DE TAREA.

La conmutación de tarea mediante la ejecución de instrucciones JMP o CALL en un segmento de código de la tarea vieja, exigen que el nivel de privilegio del mismo sea el máximo, es decir, el 0 para igualarlo al del TSS.

Para poder acceder a un TSS desde un segmento de código con menor nivel de privilegio, se usan las puertas de tarea, que son un tipo especial de descriptor del sistema ubicados en la GDT, en la LDT o en la IDT.

La puerta de tarea realiza una indirección sobre el TSS, comportándose de forma similar a la puerta de llamada (figura 13.6).

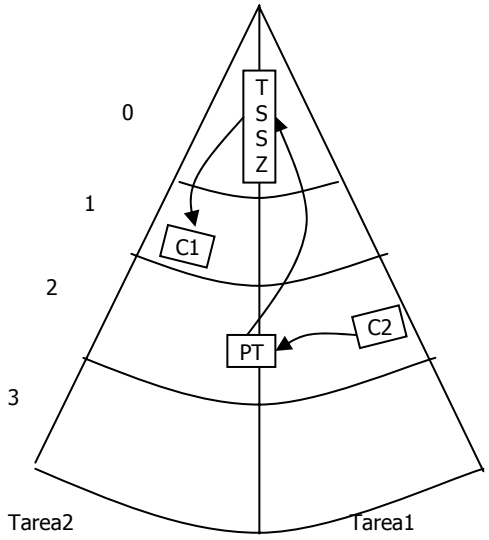


Figura 13.5. Mediante la puerta de tarea (PT) con PL = 2, desde C2 de la tarea 1 se puede realizar la conmutación a la tarea 2.

La estructura de un descriptor correspondiente a una puerta de tarea se ofrece en la figura 13.7 y sólo tiene dos campos útiles:

1. **El selector del TSS de la tarea,** que se carga en el TR del Pentium.

2. **Los atributos**, con los siguientes bits válidos: el bit P de presencia, el campo de dos bits DPL que indica el PL en el que se ha situado la puerta, y el campo TIPO, que identifica el descriptor como puerta de tarea y que tiene el código 5 en hexadecimal.

3. **El DPL (Nivel de privilegio del Descriptor)** es el encargado de producir la conmutación de tarea, puesto que para acceder a la puerta de tarea es preciso poseer un nivel de privilegio igual o mayor.

DESCRIPTOR DE LA PUERTA DE TAREA.

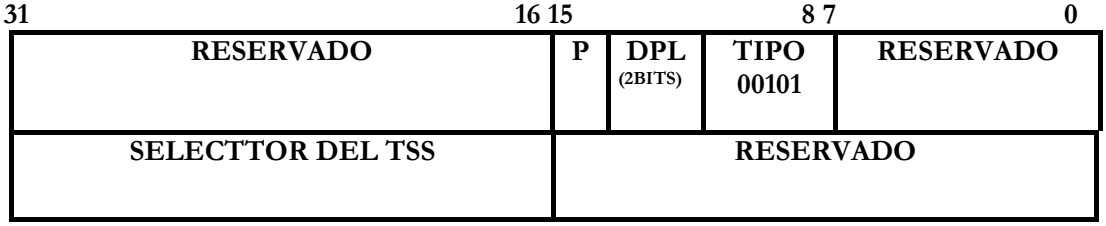


Figura 13.6. En un descriptor de una puerta de tarea, sólo están definidos el campo del selector del TSS y los atributos.

Se puede acceder a una tarea a través de un descriptor de puerta de tarea o a través de un TSS. Ambas estructuras se proporcionan para satisfacer las siguientes necesidades:

1. La necesidad de una tarea de tener sólo un flag BUSY. Como el flag BUSY se guarda para una tarea en el descriptor TSS, cada tarea debe tener sólo un descriptor TSS. Sin embargo, puede haber varias puertas de tareas refiriéndose al mismo descriptor TSS.

2. La necesidad de proporcionar el acceso selectivo a las tareas. Las puertas de tarea llenan esta necesidad, porque éstas pueden residir en una LDT y pueden tener un DPL que es diferente al DPL del descriptor de TSS. Un programa o proceso que no tiene el suficiente privilegio para acceder al descriptor TSS para una tarea en el GDT (qué normalmente tiene un DPL de 0) puede permitirse el acceso a la tarea a través de la puerta de tarea con un DPL más alto. Las puertas de tarea dan al sistema operativo el nivel mayor para limitar el acceso a las tareas específicas.

3. La necesidad de que una interrupción o excepción sea manejada por una tarea independiente. Las puertas de tarea también pueden residir en el IDT que permite manejar las interrupciones y excepciones por las tareas del "handler". Cuando una interrupción o el vector de excepciones apunta a una puerta de tarea, la CPU cambia a la tarea especificada a través de la citada puerta de tarea. En la figura 13.8 se ilustra cómo una puerta de tarea en un LDT, una puerta de tarea en un GDT, y una puerta de tarea en un IDT, pueden apuntar a la misma tarea.

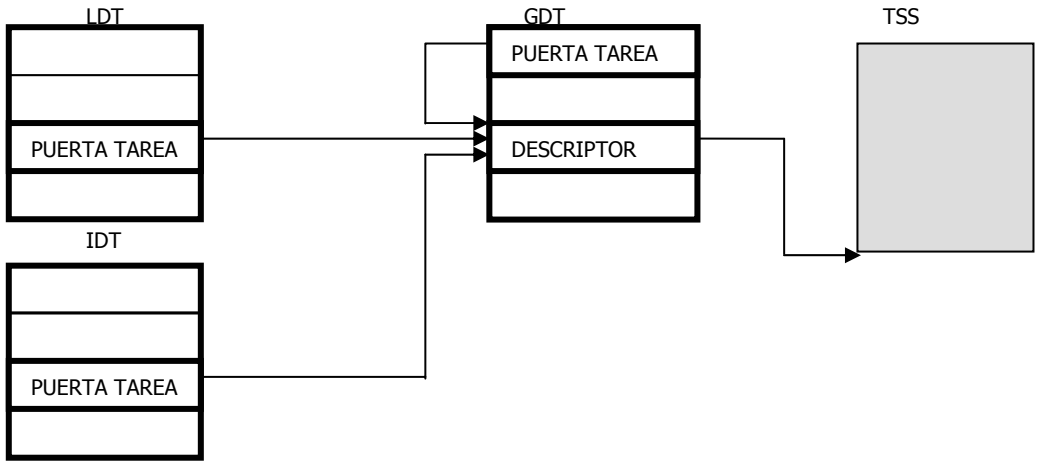


Figura 13.8. Puertas de tarea referenciando a una misma tarea

Cuando, desde el programa en curso, se ejecuta una instrucción JMP o CALL sobre una puerta de tarea en principio se comprueban las reglas de acceso comparando el CPL del programa en ejecución con el DPL de la puerta. Si la regla de acceso relativa a los PL se cumple, se salva el contexto de la CPU en el TSS de la tarea actual, referenciado por el contenido de la TR. Después, el descriptor de la puerta de tarea proporciona el selector del nuevo TSS que se carga en TR, procediendo a cargar la información almacenada en el TSS nuevo en la CPU, es decir, a configurar un nuevo contexto. A partir de este momento, el proceso inicia la nueva tarea con la instrucción direccionada por la LDT y CS:EIP, que ha suministrado el TSS nuevo (figura 13.9).

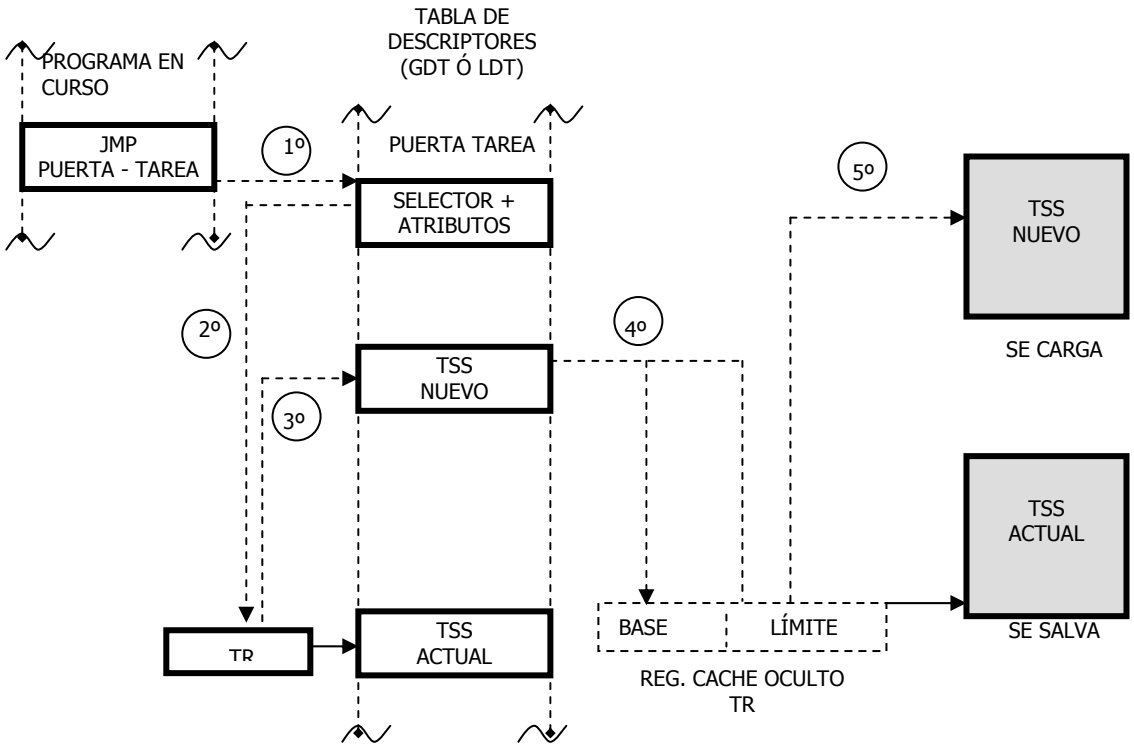


Figura 13.9. Expresión gráfica del mecanismo de conmutación de tarea a través de una puerta de tarea.

Al poder residir los descriptores de las puertas de tarea, indistintamente, en la GDT o en la LDT, es posible solicitar la conmutación de tarea desde cualquier punto del sistema.

En la conmutación de tarea el mecanismo de protección comprueba el cumplimiento de diversas reglas:

1. El nivel de privilegio de la puerta de tarea ha de ser igual o menor que el EPL, que es el máximo entre el CPL y el RPL del selector de la puerta.

2. El descriptor del TSS nuevo debe estar presente (P) en la memoria y con el límite correcto.

3. Cuando se carga el selector del TSS nuevo en el TR, hay que marcarlo como ocupado.

4. Para indicar que se ha producido una conmutación de tarea, se pone a 1 el bit TS de la Palabra de Estado.

Para soportar adecuadamente los anidamientos entre tareas, realizados mediante instrucciones CALL, el Pentium maneja el bit B (ocupado o BUSY) existente en el descriptor del TSS.

En general, en una conmutación de tareas el Pentium pone a 1 el bit B del descriptor del TSS de la tarea nueva, que comienza a procesarse, y pone a 0 el bit B del descriptor del TSS viejo. Esto último sólo sucede si la conmutación se ha originado como consecuencia de algún hecho que no sea la ejecución de una instrucción CALL, pues, en tal caso, permanece a 1 el bit B de la tarea anterior.

De esta forma, en los anidamientos de tareas, al aparecer la instrucción IRET de retorno, la CPU utiliza el valor del TR previo guardado en el TSS activo, para conmutar con la tarea que antes la enlazó o anidó.

El empleo de las puertas de tarea se presenta muy eficaz cuando se desea que una determinada subrutina proporcione servicio a un conjunto de varias tareas. Ejecutando dicha subrutina como una tarea independiente, se asegura que el contexto de la tarea en curso no sea corrompido al manejar la tarea común, pues en la conmutación se salva el estado de la tarea en curso (figura 13.10).

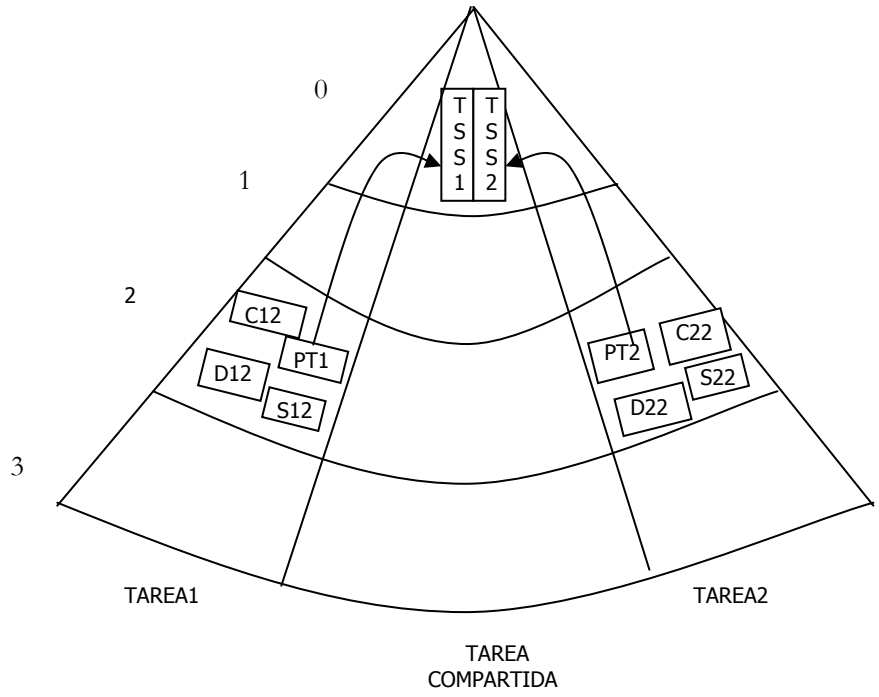


Figura 13.10. Las puertas de tarea permiten crear un entorno seguro en ambientes multitarea en los que existe una tarea compartida.

13.6- MAPA DE BITS DE PERMISO DE E/S.

El segmento TSS tiene una longitud mínima de 67H bytes y en él se guarda el contexto fundamental de la CPU. Los cuatro bytes situados en su cima, cuyas direcciones en hexadecimal serán las 64, 65, 66 y 67, en estas posiciones se encuentra, ocupando los bytes 66 y 67, el campo llamado "base del mapa de permiso de E/S".

Los 16 bits del campo base del mapa de permiso de E/S, conforman el desplazamiento que hay que añadir a la posición que ocupa este campo, o sea, 67 H, para apuntar el comienzo del "mapa de bits de permiso de E/S". Dicho mapa ocupa desde la posición anteriormente citada hasta el límite del segmento TSS.

El mapa de bits de permiso de E/S está compuesto por una serie de bits 0 y 1, cada uno de los cuales se corresponde con la dirección de una puerta de ocho bits del mapa de E/S. Si el bit del mapa es 0, la puerta correspondiente del mapa de E/S es accesible, pero si el bit es 1, la puerta no es accesible y se produce una excepción de violación de la protección en el caso de que intente acceder a ella.

En el caso de que una instrucción de E/S afecte a un operando de tamaño mayor que el byte es decir, se quiera realizar una operación de E/S con una información de más de ocho bits (una puerta de E/S), todos los bits que referencien las puertas a las que afecta dicha instrucción deben estar a cero.

14 INTERRUPCIONES Y EXCEPCIONES

14.1. CONCEPTOS GENERALES

Las interrupciones y excepciones son acontecimientos externos o internos al procesador, que provocan una desviación en el flujo de control de la CPU. Estas anomalías son asíncronas, ya que no se saben cuando van a suceder.

Las **interrupciones** se originan por acontecimientos externos, como la activación en alguna patita del procesador o bien por la ejecución de alguna de las instrucciones especificas que tiene el procesador para generarlas.

Las **excepciones** se generan automáticamente como consecuencia de algún acontecimiento anormal, producido y detectado en el desarrollo del programa en curso de ejecución.

Para manejar las interrupciones y excepciones la CPU del Pentium dispone de una tabla de interrupciones, llamada **IDT** de 256 entradas, cada una de las cuales atiende a un tipo de interrupción o excepción diferente, y que mediante un mecanismo se especifica la dirección de comienzo de la rutina que atiende la causa que la ha provocado, solucionando de esta manera el problema.

La tabla IDT ocupa un segmento cuya base y límite están contenidos en el Registro de Tablas de Descriptores de Interrupciones **IDTR**, si el Pentium trabaja en Modo Protegido. En Modo Real, la operatividad de dicha tabla es diferente, aunque su objetivo es el mismo.

14.2. TIPOS DE INTERRUPCIONES

Las interrupciones que trata el procesador, pueden ser de dos tipos:

14.2.1. Interrupciones externas

Se trata de interrupciones activadas por componentes hardware externos, que provocan la activación de una de las patitas del procesador Pentium. Esta activación la detecta el Controlador de Interrupciones Programable Avanzado local (APIC) que dispone el procesador Pentium y sucesores

Cuando el APIC local no esta habilitada, las patitas de estas, se configuran como INTR y NMI. Cuando el APIC esta habilitado, las patitas se pueden configurar a través de la tabla de vectores para asociarlo con cualquier vector de interrupción o excepción del procesador.

Hay que tener en cuenta que la activación de otras patitas del procesador, son capaces de provocar una interrupción del proceso de la CPU. Sin embargo estas interrupciones no son manipuladas por el mecanismo de interrupción y excepción descritos por este capitulo. Esas patitas son: RESET#, FLUSH#, STPCLK#, SMI, R/S# e INIT#.

Las patitas que provocan estas interrupciones externas, son la **INTR** y **NMI**.

- **NMI**: es una interrupción NO enmascarable y que por lo tanto es siempre atendida por la CPU. esta interrupción es el resultado de un problema hardware serio, como el error de paridad de la memoria o un error del bus.

  Esta interrupción se activa por flanco ascendente de su patita y tras la su activación, se atiende mediante la entrada numero 2 de la tabla IDT.

- **INTR**: es una interrupción que se origina por la activación de su patita INTR en el procesador, por parte de un componente externo al procesador.

  Se trata de una interrupción enmascarable y su aceptación y puesta en marcha depende del estado del señalizador IF del registro E-FLAGS. Si IF=1, se atiende la interrupción enmascarable, pero si IF=0, no se tiene en cuenta la petición.

  Cuando la CPU acepta la petición de INTR, responde con dos ciclos de reconocimiento de interrupción, similares a los de lectura. Durante dichos ciclos, el componente externo que haya solicitado la interrupción del proceso del procesador, deberá introducir el valor de la entrada de la tabla IDT, que quiere que la interrupción procese. El valor de la entrada de la tabla IDT, vendrá dada por las 8 líneas de menos peso del bus de datos (D0-D7)

14.2.2. Interrupciones internas

Este grupo de interrupciones se origina como consecuencia de la ejecución de alguna instrucción especial, es decir, son interrupciones que se provocan a través del software. Su desarrollo funcional es igual al que corresponde a una excepción.

Las instrucciones que pueden provocar este tipo de interrupciones son **INT n** y **INTO**

- **INT n** es una interrupción NO enmascarable, generada por software. Siempre que se ejecuta esta instrucción, se salta a la rutina de la interrupción que indique el valor 'n' que apunta a la tabla de la IDT.

- **INTO** es una intrusión que salta al vector 4 de la IDT, siempre que el valor del bit OF del registro E-FLAGS valga 1.

14.3. TIPOS DE EXCEPCIONES

Las excepciones son provocadas automáticamente por el procesador al detectar alguna anomalía en el flujo de control.

Los tipos de excepciones son las siguientes:

- **Excepciones faltas o errores**: son aquellas excepciones que se encargan de corregir el error o la falta al intentar ejecutar una instrucción, retornando al lugar donde la CPU la dejo, tras la finalización de la excepción. De esta forma la instrucción que la había provocado, se puede realizar. Un ejemplo de este tipo de excepción es cuando la CPU ejecuta una operación matemática y aun no tiene todos los operandos que están involucrados en la instrucción.

- **Trampa**: son aquellas excepciones que se generan tras la finalización de la instrucción. Un ejemplo de este tipo de excepción son las interrupciones definidas por el usuario e incluidas en el programa.

- **Aborto**: son aquellas excepciones que no permiten la localización exacta de la instrucción que la origino. Se suele usar para indicar errores muy graves, como los que se originan del comportamiento del equipo físico o valores en las tablas que manipulan el sistema.

14.4. TABLA DE DESCRIPTORES DE INTERRUPCIONES (IDT)

En Modo Protegido, el procesador dispone de la Tabla de Descriptores de Interrupciones (IDT), que contiene los descriptores que asocia cada vector de excepción o interrupción con la puerta de descriptores para cada proceso de interrupción o excepción. Esta tabla consta de 256 entradas, de las cuales las veinte primeras (0 - 19) están reservadas exclusivamente para Intel. De las 20 a la 31 están reservados exclusivamente para la CPU y las demás entradas están a disposición del usuario.

La IDT puede ubicarse en cualquier sitio dentro del espacio de direcciones lineales. Ya que el procesador localiza la IDT a través del registro IDTR. Este registro contiene 32 bits para marcar la dirección base y 16 bits para indicar el límite de la IDT.

Para usar la IDTR, el procesador utiliza dos instrucciones la LIDT y SIDT. La instrucción LIDT carga en la IDTR la dirección base y el límite. Esta instrucción puede ser ejecutada únicamente cuando el CPL=0. Este normalmente se emplea para la inicialización del código de un sistema operativo cuando se crea una IDT o también lo puede usar para pasar de una IDT a otra. La instrucción SIDT hace una copia en la memoria de los valores de la base y el límite almacenados en la IDTR. Esta instrucción se puede ejecutar en cualquier nivel de privilegio.

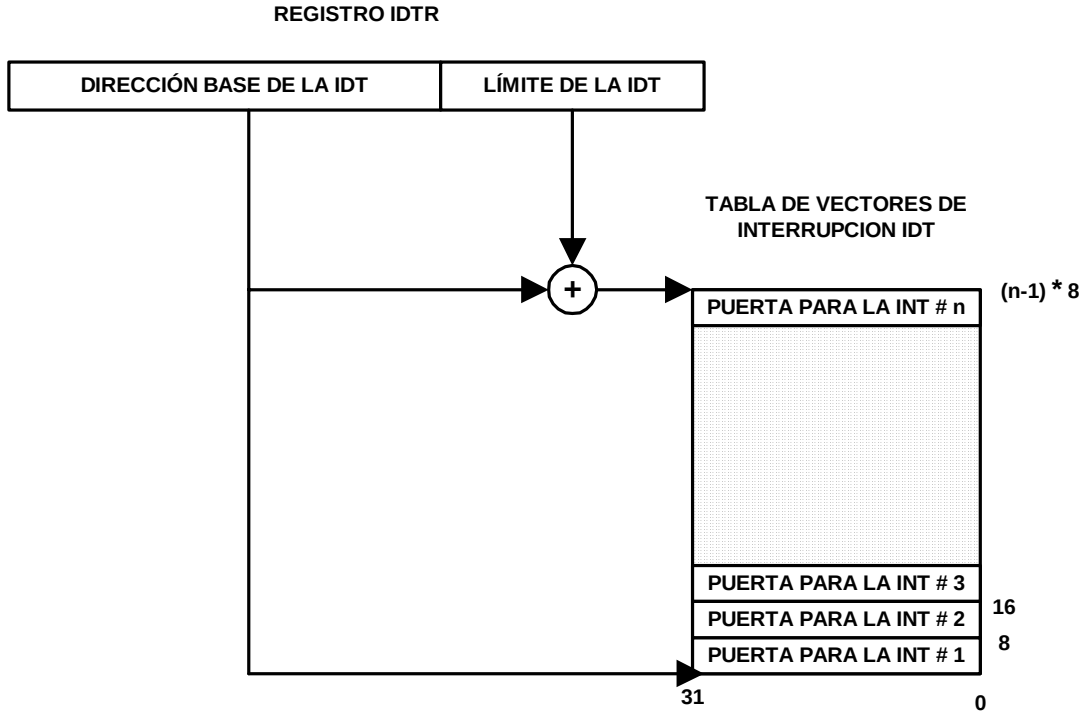


Figura 14.1 – Relaciones entre la IDTR y la IDT

En la tabla 14.2 se explican las entradas de la IDT, dando su vector de interrupción, descripción, instrucciones que puede causar, el código de error y su clase.

VECTOR	DESCRIPCIÓN	INSTRUCCIONES QUE PUEDE CAUSAR	CÓDIGO DE ERROR	CLASE
0	Error de división #DE	DIV IDIV	No	Falta
1	Excepción de depuración #DB	Cualquier código o dato de referencia	No	Falta Trampa
2	Interrupción no enmascarable NMI	Interrupción externa no enmascarable	No	NMI interrupción externa
3	Punto de ruptura #BP	INT 3	No	Trampa
4	Sobrepasamiento #OF	INT 0	No	Trampa
5	Comprobación de limites #BR	BOUND	No	Falta
6	Código OP no valido #UD	UD2 o código OP reservado	No	Falta
7	Coprocesador no disponible #NM	WAIT/FWAIT o coma flotante	No	Falta
8	Doble falta #DF	Instrucciones que origine una excepción, NMI o INTR	Si (cero)	Aborto
9	Sobrepasamiento del segmento por el coprocesador	Instrucciones de coma flotante	No	Falta
10	TSS no valida #TS	Acceso TSS o conmutación de tareas	Si	Falta
11	Segmento no presente #NP	Carga de registros del segmento o acceso a segmentos	Si	Falta
12	Excepción en la pila #SS	Operaciones de pila y carga registros SS	Si	Falta
13	Protección general #GP	Referencias a memoria y comprobación de protección	Si	Falta
14	Fallo de página #PF	Referencia a memoria	Si	Falta
15	Reservas de Intel	----	No	----
16	Error de coma flotante x87 FPU #MF	WAIT/FWAIT o coma flotante	No	Falta
17	Comprobación de alineamiento #AC	Cualquier dato referenciado en memoria	Si	Falta
18	Comprobación de la maquina #MC	Códigos de error y fuentes son modelos dependientes	No	Aborto
19	Excepción de coma flotante SIMD #XF	SSE SSE2	No	Falta
20-31	Reservas por Intel	----	----	----
32-255	Interrupción definidas por el usuario	Interrupción externa o INT n	----	Interrupción

Tabla 14.2 – Entradas de la tabla IDT

A continuación se describe cada vector de interrupción:

Vector 0: Error de división

Este error lo generan las instrucciones **DIV** o **IDIV** cuando tienen el divisor con valor 0 o cuando el cociente es muy grande y no cabe en el operando destino.

Los contenidos guardados en los registros CS y EIP apuntan a la instrucción que ha generado la excepción.

Vector 1: Excepción de depuración

Esta interrupción se produce por causa de que se haya detectado una o más de las condiciones de excepción de depuración. En función de si es una falta o una depuración la condición será diferente, como se puede apreciar en la figura 14.2:

Condiciones de excepción	Clase de condición
Instrucción de Captura del punto de ruptura	Falta
Lectura de datos o escritura del punto de ruptura	Trampa
Lectura de E/S o escritura del punto de ruptura	Trampa
Condición de defecto general	Falta
Paso único	Trampa
Cambio de tarea	Trampa
Ejecución de la instrucción INT 1	Trampa

Tabla 14.3 – Condiciones de excepción de depuración y sus clases de excepción correspondientes.

Si se trata de una falta, los contenidos guardados en los registros CS y EIP apuntan a la instrucción que ha generado la excepción, mientras que si es una depuración, apuntan a la **siguiente** instrucción a la que ha generado la excepción.

Vector 2: NMI

Esta interrupción la provoca la patita de entrada del **NMI** del procesador y es siempre atendida. Esta interrupción provoca la llamada al manipulador de interrupciones de la **NMI**.

Vector 3: Punto de ruptura (#BP)

Esta interrupción salta cuando de ha ejecutado una instrucción de punto de ruptura, **INT 3**, saltando a una rutina de depuración. Además la excepción de punto de ruptura puede también generarse ejecutando la instrucción **INT n** con el valor 3 en 'n'. La acción de esta instrucción (INT n) es ligeramente diferente que la instrucción **INT 3.**

Los contenidos guardados de los registros CS y EIP apuntan a la instrucción siguiente a la instrucción **INT 3**.

Vector 4: Desbordamiento (#OF)

Esta interrupción salta cuando se ejecuta una instrucción **INTO** y el valor de OF del registro E-FLAGS es igual a 1. Hace que se produzca una depuración de desbordamiento.

Este tipo de interrupciones se ejecutan cuando se comprueba directamente el señalizador OF del registro E-FLAGS o mediante la instrucción INTO, tras la ejecución de una instrucción aritmética como la ADD o SUB. La ventaja de utilizar la instrucción INTO, consiste en que si la excepción por desbordamiento es detectada, un manipulador de excepciones puede ser llamado automáticamente para manipular la condición de desbordamiento.

Los contenidos guardados de los registros CS y EIP apuntan a la instrucción siguiente a la instrucción INTO.

Vector 5: Excepción por sobrepasamiento del rango (Bound #BR)

Esta excepción surge cuando se ha intentado leer o escribir fuera de los límites de un segmento. La genera la instrucción BOUND al comprobar que un índice de array con signo no está entre el límite inferior y superior de un array ubicado en memoria.

Los contenidos guardados de los registros CS y EIP apuntan a ka instrucción BOUND que generó la excepción.

Vector 6: Excepción por cogido de operación no valido

Esta excepción surge cuando el procesador:

- Intenta ejecutar un código de operación no valido o reservado.
- Intenta ejecutar un código de operación con tipos de operandos no validos para la instrucción.
- Intenta ejecutar una instrucción MMX, SSE o SS2 en un procesador IA-32 que no soporta esa tecnología o el señalizador EM del registro de control CR0 esta activo.
- Intenta ejecutar una instrucción SSE o SSE2 en el procesador IA-32 que produce una excepción de coma flotante SIMD cuando el bit OSXMMEXCPT del registro de control CR4 esta inactivo o cuando el bit OSFXSR del registro de control CR4 está inactivo.
- Intenta ejecutar una instrucción LLDT, SLDT, LTR, LSL, LAR, VERR, VERW o ARPL mientras se está en Modo Real o en Modo Virtual 8086.
- Intenta ejecutar la instrucción RSM cuando no está en Modo SMM.
- Ha detectado un prefijo LOCK que precede a una instrucción que puede no estar cerrada o una que puede estar cerrada pero que el operando destino no esta localizado en memoria.
- Ha ejecutado la instrucción UD2, que garantiza la generación de un código de operación no valido.

Los contenidos guardados de los registros CS y EIP apuntan a la instrucción que generó la excepción.

Vector 7: Excepción por coprocesador no disponible (#NM)

Esta excepción salta cuando el procesador detecta que no tiene disponible el coprocesador para realizar operaciones con **coma flotante** o **WAIT** / **FWAIT**.

Los contenidos guardados de los registros CS y EIP apuntan a la instrucción de coma flotante o a la instrucción WAIT/FWAIT que generó la excepción.

Vector 8: Excepción por doble fallo (#DF)

Indica que el procesador detecta una excepción secundaria mientras está llamando a un manipulador de excepciones para una excepción anterior. Normalmente, cuando el procesador detecta otra excepción mientras está intentando llamar al manipulador de excepciones, las dos excepciones pueden ser manipuladas seriamente. Si en cambio el procesador no puede manipularlas, señaliza una excepción de doble falta. Para determinar cuando dos faltas necesitan ser señaladas como doble falta, el procesador divide las excepciones en tres clases: excepciones benignas, excepciones contribuyentes y faltas de página.

Clase	Numero de vector	Descripción
Excepciones benignas	1	Depuración
	2	Interrupción NMI
	3	Punto de ruptura.
	4	Desbordamiento
	5	Rango BOUDED excedido
	6	Código OP no valido.
	7	Coprocesador no disponible
	9	Segmento de estado de tarea no valido
	16	Error en coma flotante.
	17	Comprobación de alineamiento.
	18	Comprobación de maquina.
	19	Coma flotante SIMD
	All	INT n
	All	INTR
Excepciones contribuyentes	0	Error de división.
	10	TSS o valida.
	11	Segmento no presente
	12	Falta de pila
	13	Protección gerenal
Falta de páginas	14	Falta de página

Tabla 14.4 – Clases de interrupciones y excepciones.

Cualquier tipo de faltas generadas mientras el procesador intenta transferir el control a un manipulador de faltas adecuado podría dejar secuencias de doble falta.

Primera excepción	Segunda excepción		
	Benigna	**Contribuyente**	**Falta de página**
Benigna	Manipula Excepciones en serie	Manipula Excepciones en serie	Manipula Excepciones en serie
Contribuyente	Manipula Excepciones en serie	Genera una doble falta	Manipula Excepciones en serie
Falta de página	Manipula Excepciones en serie	Genera una doble falta	Genera una doble falta

Tabla 14.5 – Condiciones para generar una doble falta

Los contenidos guardados de los registros CS y EIP están indefinidos.

Vector 9: Sobrepasamiento del segmento por el procesador

Indica que un sistema basado en el Intel386 con procesador matemático del Intel387 detecta la violación de una página o segmento mientras se transfería una parte del operando del coprocesador matemático del Intel387.

Vector 10: Excepción del segmento de estado de tarea no valido

Indica que se ha intentado realizar una conmutación de tareas y que la información no valida es detectada por la TSS para la tarea correspondiente.

Índice del código de error	Condición no valida
Índice del selector del segmento TSS.	El límite del segmento TSS menor que 67H para 32 bits o menor que 2CH para 16 bits.
Índice del selector del segmento LDT.	LDT no valida o LDT no presente.
Índice del selector del segmento de pila.	El selector del segmento de pila excede del límite de la tabla de descriptores.
Índice del selector del segmento de pila.	El segmento de pila no se puede escribir.
Índice del selector del segmento de pila.	El segmento de pila DPL ≠ CPL.
Índice del selector del segmento de pila.	El selector de segmento de pila RPL ≠ CPL.
Índice del selector del segmento de código.	El selector del segmento de código excede del límite de la tabla de descriptores.
Índice del selector del segmento de código.	Segmento de código no es ejecutable.
Índice del selector del segmento de código.	Segmento de código no ajustable.
Índice del selector del segmento de código.	Segmento de código ajustable DPL mejor que CPL.
Índice del selector del segmento de datos.	El selector de segmento de datos excede del límite de la tabla de descriptores.
Índice del selector del segmento de datos.	Segmento de datos no leíble.

Tabla 14.6 – Condiciones TSS no válidas.

Si la condición de excepción se detecta antes del cambio de tarea, los contenidos guardados de los registros CS y EIP apuntan a la instrucción que invocó el cambio de tarea. Si la condición de excepción se detecta después del cambio de tarea, los contenidos guardados de los registros CS y EIP apuntan a la primera instrucción de la nueva tarea.

Vector 11: Segmento no presente

Esta excepción surge cuando el procesador intenta acceder a un descriptor de segmento o una puerta y está inactivo. El procesador puede generar esta excepción durante cualquiera de las siguientes operaciones:

- Mientras se intenta cargar los registros CS, DS, ES, FS o GS. Esta situación puede ocurrir mientras se realiza el cambio de tarea.
- Mientras se intenta cargar el LDTR empleando la instrucción LLDT.
- Mientras se ejecuta la instrucción LTR y la TSS es señalada como no presente.
- Mientras se intenta usar un descriptor de puertas o la TSS es señalada como segmento no presente, pero a pesar de todo es valida.

Los contenidos guardados de los registros CS y EIP normalmente apuntan a la instrucción que ha generado el cambio de tarea. Si la excepción se produce mientras se carga los descriptores del segmento para los selectores del segmento de la nueva TSS, los registros CS y EIP apuntan a la primera instrucción en la nueva tarea. Si la excepción se produce mientras se accede al descriptor de la puerta, los registros CS y EIP apuntan a la instrucción que invoca el acceso.

Vector 12: Excepción por falta de pila

Esta excepción surge cuando hay algún problema al manejar la pila. Estos problemas pueden ser:

- Se detecta una violación de límite durante una operación que se refiere al registro SS. Las provocan las instrucciones orientadas al manejo de la pila como puedes ser POP, PUSH, CALL, RET, IRET, ENTER y LEAVE, así como otras referenciadas a memoria que implícita el empleo del registro SS.

- Se detecta un segmento de pila no presente cuando se intenta cargar el registro SS. Esto puede ser debido al cambio de tarea, la ejecución de la instrucción CALL o un retorno a un nivel de privilegio diferente, la ejecución de la instrucción LSS, MOV o POP al registro SS.
- Es posible la recuperación de esta falta extendiendo el límite del segmento de pila o cargando el segmento de pila perdido en memoria.

Los contenidos guardados de los registros CS y EIP normalmente apuntan a la instrucción que ha generado la excepción. Sin embargo cuando la excepción resulta del intento de carga el segmento de pila no presente durante un cambio de tarea, los contenidos guardados de los registros CS y EIP apuntan a la primera instrucción de la nueva tarea.

Vector 13: Excepción por protección general

Esta excepción surge cuando hay alguna clase de violación de protección, como pueden ser:

- Exceso del segmento límite cuando se accede a los segmentos CS, DS, ES, FS o GS.
- Exceso del segmento límite cuando se hace referencia a la tabla de descriptores.
- Transferencia de la ejecución a un segmento que no es ejecutable.
- Escritura de un segmento de código o de datos que es solo de lectura.
- Lectura de un segmento de solo ejecución.
- Carga del registro SS con un selector de segmento para un segmento de solo lectura.
- Carga de los registros SS, DS, ES, FS o GS con un selector de segmento para un segmento del sistema.
- Carga de los registros DS, ES, FS o GS con un selector de segmento para un segmento de código de sólo ejecución.
- Carga del registro SS con un selector de segmento de un segmento ejecutable o un selector de segmento nulo.
- Carga del registro CS con un selector de segmento para un segmento de datos o un selector de segmento nulo.
- Acceso a memoria empleando los registros DS, ES, FS o GS cuando contiene un selector se segmento nulo.
- Cambia de una tarea ocupada durante una llamada o salto al TSS.
- Cambiar a una tarea no ocupada durante la ejecución de una instrucción IRET.
- Empleo de un selector de segmento en un cambio de tarea que apunta a un descriptor TSS en la LDT actual. Los descriptotes TSS pueden únicamente estar en la GDT.
- Violación de cualquiera de las reglas de privilegio.
- Exceso del límite de la longitud de ka instrucción (15 bytes).
- Carga del registro CR0 con un señalizador de página PG activo y un señalizador PE inactivo.
- Carga del registro CR0 con un señalizador activo y un señalizador CD inactivo.
- Referencia a una entrada en la IDT que no es ni una interrupción, trampa ni puerta de tarea.
- Intento de acceso a un interrupción o excepción a través de una interrupción o puerta de depuración del Modo Virtual 8086 cuando el segmento de código DPL del manipulador es mejor que 0.
- Intento de escritura de '1' en un bit reservado del registro CR4.
- Intento de ejecución de una instrucción privilegiada cuando el CPL no es igual a 0.
- Escritura de un bit reservado en una MSR.
- Acceso a una puerta que contiene un selector de segmento nulo.
- Ejecución de la instrucción INT n cuando el CPL es mejor que el DPL de la interrupción, trampa o puerta de tarea correspondiente.

- El selector de segmento en una llamada, interrupción o puerta de depuración no apunta a un código de segmento.
- El operando del selector del segmento en la instrucción LLDT es un tipo local o no apunta al descriptor del segmento del tipo LDT.
- El operando del selector del segmento en la instrucción LTR es local o apunta a una TSS que no es valida.
- El selector del segmento de código preciso para una llamada, salto o retorno es nulo.
- Si el señalizador PAE y/o PSE del registro de control está activo y el procesador detecta bits reservados en una entrada de la tabla de punteros de directorio de página, se activa a 1. Esos bits son comprobados durante una escritura de los registros de control CR0, CR3 y CR4 que hace que la entrada a la tabla de punteros del directorio de páginas se vuelva a cargar.
- Intento de escritura con un valor que no sea cero en los bit reservados del registro MXCSR.
- Ejecución de la instrucción SE o SSE2 que intenta acceder a una posición de memoria de 128 bits que no está alineada en un límite de 16 bytes cuando la instrucción se requiere un alineamiento de 16 bytes. Esta condición también se aplica al segmento de pila.

Los contenidos guardados de los registros CS y EIP normalmente apuntan a la instrucción que ha generado la excepción.

Vector 14: Excepción de falta de página

Indica que, con la paginación habilitada, el procesador detecta una de las siguientes condiciones mientras se emplea el mecanismo de traslación de página para trasladar la dirección lineal a física:

- El señalizador P (presencia) en un dirección de página o en la entrada de la tabla de página necesitado para la traslación de la dirección, esta puesto a 0, indicando que una tabla de página o la página que contiene el operando no esta presente en memoria física.
- El proceso no tiene suficiente privilegio para acceder a la página indicada.
- El código de ejecución en modo usuario intenta escribir en una página de solo lectura.
- Uno o más bits reservados en la entrada de directorio de páginas son activados a 1.

Vector 16: Error en coma flotante x87 FPU

Indica que el x87 FPU ha detectado un error en coma flotante. El señalizador NE en el registro CR0 debe ser activado por una interrupción 16.

NOTA: las excepciones en coma flotante (#XF) están señaladas a través de la interrupción 19.

Vector 17: Excepción de comprobación de alineamiento

Esta excepción surge cuando el procesador antes de acceder a un segmento, detecta cuando una palabra almacenada en dirección de byte más antiguo, o una doble palabra almacenada en la dirección que no es un entero múltiplo de 4.

Vector 18: Excepción de comprobación de maquina (#MC)

Esta excepción surge cuando el procesador detecta un error interno de la maquina o un error del bus o que un agente externo detecta un error del bus, indicando dicho error por las patitas BIBIT# y MCERR#.

Vector 19: Excepción de coma flotante SIMD (#XF)

Esta excepción salta cuando el procesador detecta algún error en las instrucciones **SSE** o **SSE2** al ejecutar operaciones flotantes SIMD. Las razones que se produzcan la ejecución de esta excepción puede ser alguna de las siguientes:

- Operación invalida (#I)
- Divisor por cero (#Z)
- Operandos desnormalizado (#D)
- Desbordamientos numéricos (#O)
- Desbordamiento inferior numérico (#U)
- Resultado inexacto (precisión) (#P)

Vectores 32-255: Interrupciones definidas por el usuario

Indica que el procesador hace alguna de las siguientes cosas:

- Se ejecuta una instrucción **INT n** donde el operando de la instrucción es uno de los números se los vectores, del 32 al 255.
- Se devuelve la petición de interrupción a la patita INTR o del APIC local cuando el numero del vector de interrupción asociado con la petición es del 32 al 255.

14.5. CÓDIGO DE ERROR

Cuando el procesador esta atendiendo una interrupción o excepción y se detecta una nueva interrupción o excepción, genera un código de error que es apilado en la pila del gestor de la excepción. De esta forma se marca que se ha producido un error y antes de terminar el proceso de la interrupción o excepción con al IRET, se deberá desapilar los errores que se han producido y atenderlos. Después de hacerlo ejecutara la instrucción IRET para volver a la tarea principal.

El formato del código de error es el siguiente:

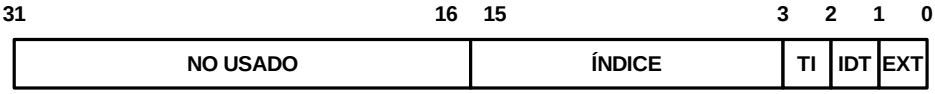


Figura 14.7 – Formato del código de error que se introduce en la pila del gestor de la excepción

EXT es un bit que indica el origen del error. Si vale 1, indica que el error viene desde el exterior y por lo tanto es una interrupción. Si por lo contrario es 0, indica que el procesador ha detectado una anomalía y por lo tanto se trata de una excepción.

IDT es el bit que indica que el error se ha producido en la tabla IDT, si vale 1 y en caso de que valga 0, el error se ha producido en la GDT o LDT.

TI es el bit que indica si el error proviene de la GDT, si tiene valor '0' o de la LDT si tiene valor '1'.

ÍNDICE, indica el selector donde se produjo el error.

14.6. INTERRUPCIONES Y EXCEPCIONES EN MODO REAL

Los Pentium como los demás procesadores de Intel, cuando arrancan por primera vez, trabajan en Modo Real y el sistema operativo debe tener ya en la memoria principal la tabla IDT, para funcionar. En la inicialización o Reset del Pentium, hay que ubicar la tabla de vectores de interrupción en el mismo sitio que el 8086 y para ello se carga la base del IDTR con el valor 0000 0000H.

Como en Modo Protegido, la IDT del Modo Real dispone de 256 entradas para las interrupciones. La diferencia entre el como protegido y el Modo Real, reside en el tipo de entras que dispone cada uno. En Modo Real, la tabla contiene descriptores, que son de 64 bits. En cambio las entradas en Modo Real son de 32 bits cada una. En cada entrada de la IDT, se encuentra la dirección del segmento de código al que a de saltar para ejecutar la interrupción, dado por el registro CS y el desplazamiento dentro de ese segmento de código, que viene dado por el registro IP.

Al ser las entradas de la IDT de 32 bits, el tamaño máximo que puede tener la tabla de interrupciones y excepciones es de un 1KB (256 entradas x 4bytes/entrada).

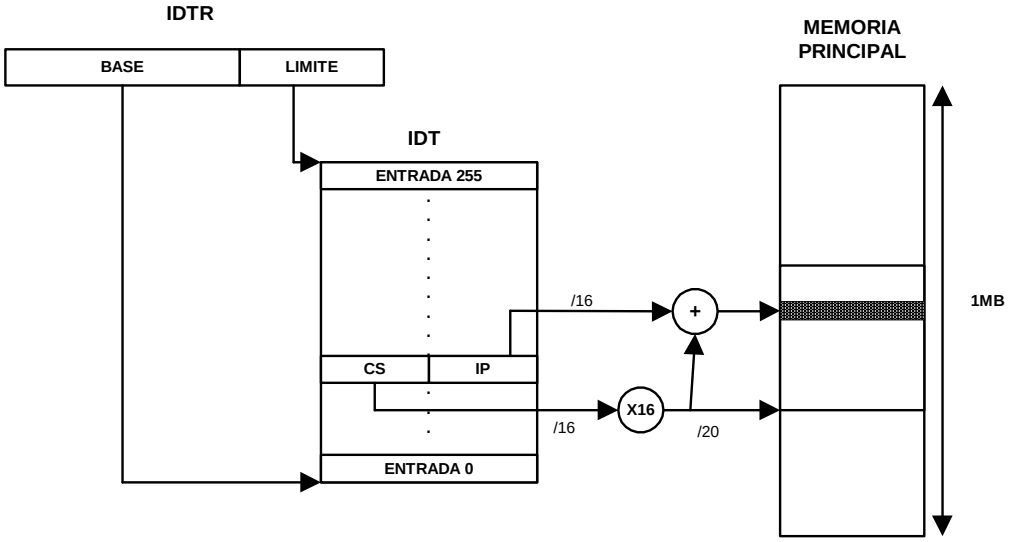


Figura 14.8 – Acceso a las interrupciones y excepciones en Modo Real.

En Modo Real y en Modo Protegido, además de las diferencias la tabla IDT, tienen algunas diferencias mas. En Modo Real dispone menos vectores de interrupción ya que no dispone de los mecanismos de protección y funcionamiento de la memoria virtual. Los vectores comunes son los siguientes:

Función	Numero de vector
Error de división.	0
Interrupción paso a paso.	1
Interrupción NMI.	2
Interrupción de puno de parada.	3
Excepción por sobrepasamiento.	4
Sobrepasamiento de los límites.	5
Código OP invalido.	6
Coprocesador no disponible.	7
Reservadas (Modo Protegido)	8 – 15
Error de coprocesador.	16
Reservadas INTEL	17 – 31
Definidas por el usuario	32 – 255

Tabla 14.9 – Relación de vectores comunes entre el Modo Real y Modo Protegido.

14.7. INTERRUPCIONES Y EXCEPCIONES EN MODO PROTEGIDO

Los Pentium como los demás procesadores de Intel, cuando arrancan por primera vez, trabajan en Modo Real. Para pasar a Modo Protegido solo hay que cambiar el valor del bit PE de registro de estados CR0 a 1. Pero antes el sistema operativo debe haber creado la tabla IDT en memoria principal nada mas empezar el Modo Protegido.

Esta tabla IDT dispone de 256 entradas para el tratamiento de las posibles interrupciones. Sin embargo a diferencia del Modo Real, la tabla está cargada por descriptores de puertas, que ocupan 8 Bytes cada uno, haciendo que el tamaño máximo de la IDT sea de 2K

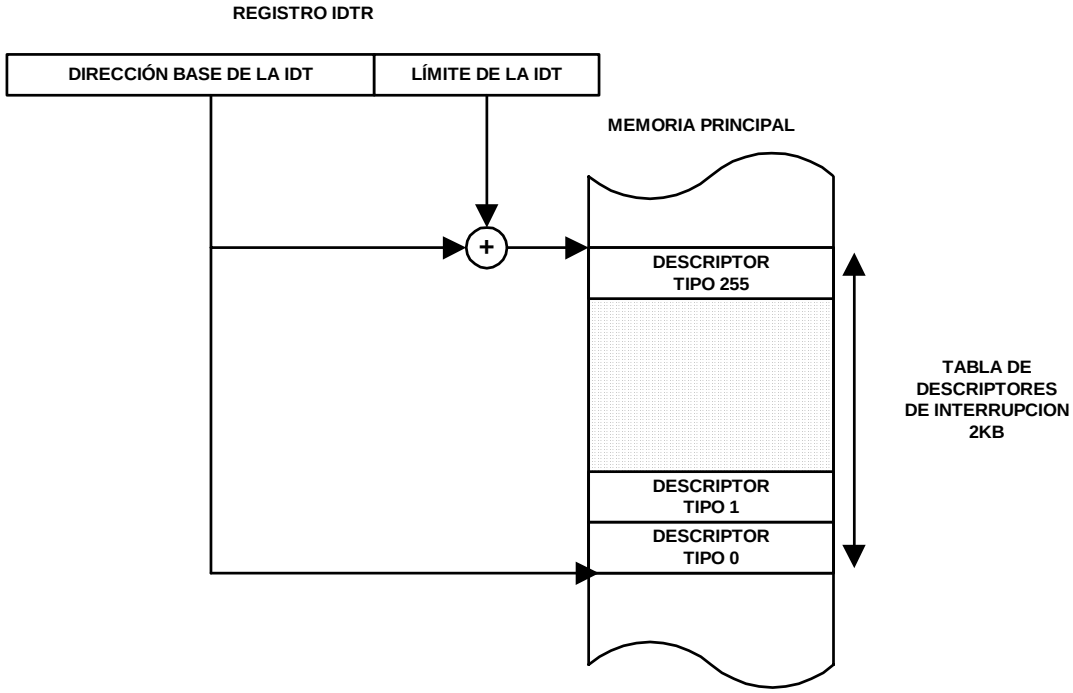


Figura 14.10 – Acceso a las interrupciones y excepciones en Modo Protegido

Los descriptores de la IDT responden a tres tipos de puertas, que son los siguientes:

- **Puertas de Tarea:** en la entrada de la IDT puedes encontrar puertas de tareas (PT) y eso quiere decir que la rutina donde esta localizada la interrupción está en otra tarea distinta a la tarea en curso. Esto tiene la ventaja de que hay una independencia entre tareas, pero como inconveniente hay que decir que una conmutación de tarea consume mucho tiempo en realizarla.

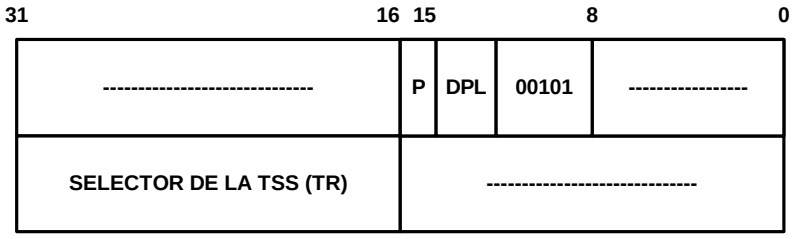


Figura 14.11 – Puerta de Tarea.

Al producirse la interrupción, se direcciona una entrada de la IDT, en la que reside el descriptor de una puerta de tarea que la va a atender. Este tipo de descriptor da lugar a una conmutación de tarea, que implica un total aislamiento de la nueva tarea en curso con respecto a la tarea que estaba ejecutando el procesador antes de la interrupción, dejando el señalizador NT con valor a 1. La tarea de la interrupción finalizará con la ejecución de la instrucción IRET, que devolverá el control a la tarea previa.

Si durante la ejecución de la tarea de la interrupción, se producen nuevas excepciones, se introducirán en la pila de la tarea de la interrupción y antes de terminar la tarea de la interrupción hay que eliminar las excepciones apiladas de la tarea.

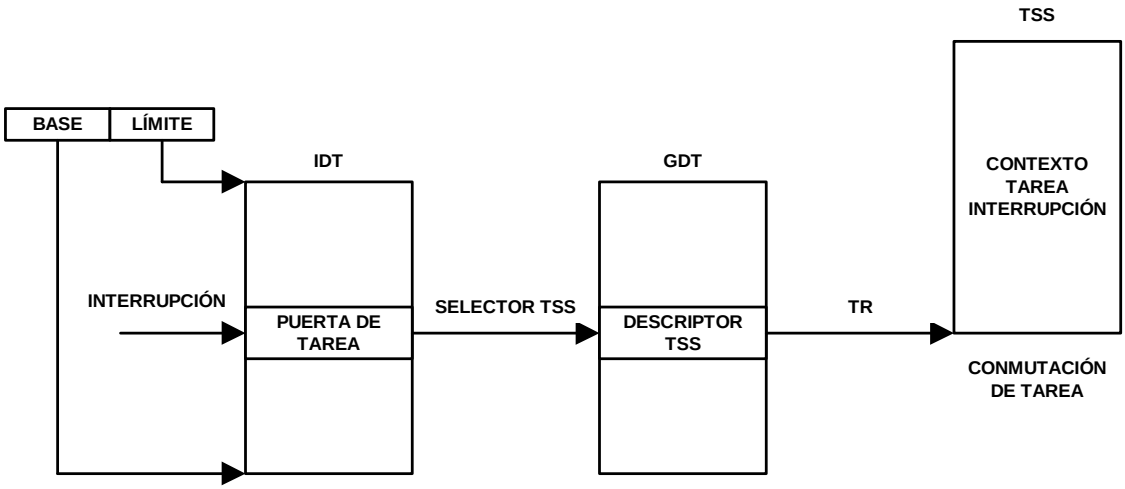


Figura 14.12 – Mecanismo de funcionamiento de una puerta de tarea.

- **Puertas de Interrupciones y Excepciones:** actúan de la misma forma que las puertas de llamadas. Estas puertas no hacen una conmutación de tareas, sino que lo único que se hace es cambiar de segmento de código dentro de una tarea.

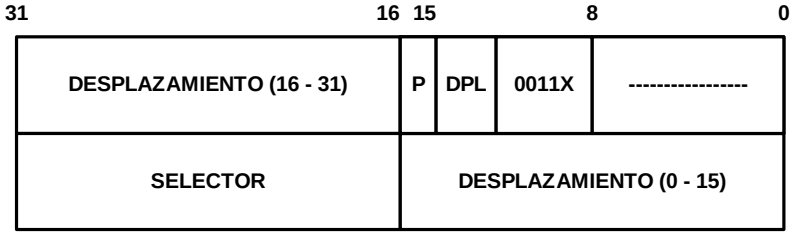


Figura 14.13 – Puerta de Interrupción o Excepción.

El bit '**X**' representa el tipo de puerta. Si adquiere el valor 1, indica que la puerta es del tipo excepción. Esto conlleva a poner el bit IF con valor 1, permitiendo que la subrutina a la que accede pueda ser parada por otra interrupción, como por un dispositivo de entrada y salida. Si en cambio el bit 'X' adquiere el valor 0, indica de que se trata de una interrupción y por lo tanto pone el bit IF con valor 0.

Al producirse la interrupción o excepción, se direcciona una entrada de la IDT, en la que reside el descriptor de una puerta de interrupción o de excepción que la va a atender. Este tipo de descriptor funciona como la puerta de llamada, atendiendo a los periféricos externos y usando los mecanismos de protección.

La puerta realiza una indirección dentro del espacio virtual de la tarea, guardando en la pila las direcciones de retorno (CS y EIP), los valores SS y ESP en caso de menor nivel de privilegio y el registro E-FLAGS. Una vez salvados estos datos, si la puerta es de interrupción, los señalizadotes TF e IF, se pondrán a cero, prohibiendo la aceptación de nuevas interrupciones enmascarables, hasta que se complete la que se halla en curso de procesamiento. Si la puerta es de excepción, el señalizador IF no se pone a 0, para poder atender a los periféricos.

Con la instrucción IRET, ultima de la rutina de servicio, se recupera CS, EIP, SS, ESP y el registros E-FLAGS, para continuar con lo que el procesador estaba haciendo antes de la interrupción o excepción.

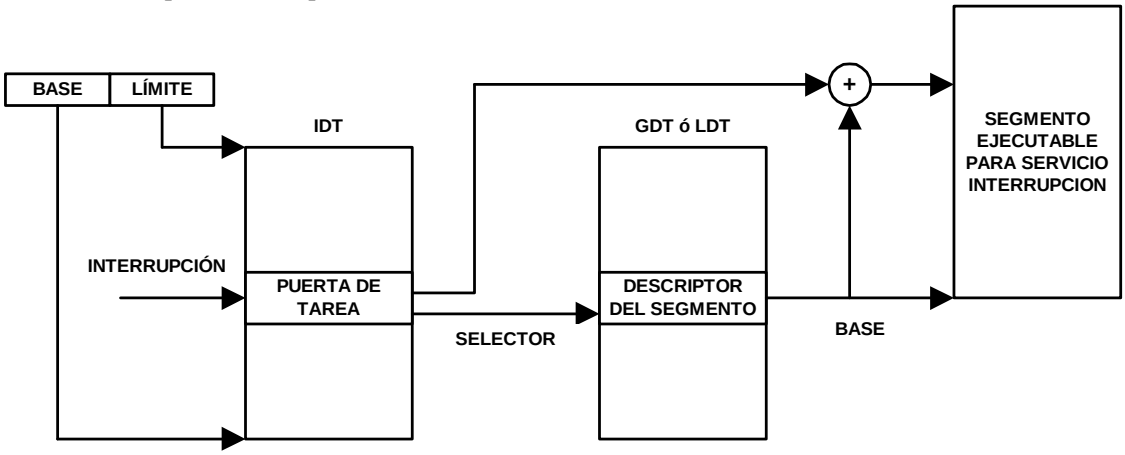


Figura 14.14 – Mecanismo de funcionamiento de una puerta de interrupción o excepción.

14.8. REGLAS DE ATENCIÓN DE UNA INTERRUPCIÓN O EXCEPCIÓN

Cuando se produce una interrupción o una excepción. Los pasos que hace la CPU para tratarla son los siguientes:

- Analizar si hay más de una interrupción pendiente. En caso de haberla, se selecciona aquella solicitud con mayor prioridad. Las prioridades son las siguientes:

Prioridad	Descripciones
1 (Mayor prioridad)	Reset del hardware y comprobación de la maquina. • RESET • Comprobación de la maquina.
2	Depuración en el cambio de tarea. • Se activa el señalizador T en TSS.
3	Intervenciones de hardware externos. • FLUSH • STOPCLK • SMI • INIT
4	Depuración de la instrucción previa. • Punto de ruptura. • Eliminación de fallos en la depuración de excepciones.
5	Interrupciones externas. • Interrupciones NMI. • Interrupciones hardware enmascarables.

6	Faltas procedentes de la captura de las instrucciones siguientes • Falta del código de punto de ruptura. • Violación límite del código de segmento. • Falta de código de página.
7	Falta procedente de la codificación de las instrucciones siguientes: • Longitud de instrucción > 15 bytes. • Código OP ilegal. • Coprocesador no valido.
8 (Menor prioridad)	Faltas en la ejecución de una instrucción • Desbordamiento. • Error bound • TSS invalida • Segmento no presente • Falta de la pila • Protección general • Falta de la página de datos • Comprobación del alineamiento. • Excepción de coma flotante x87 FPU • Excepción de coma flotante SIMD

Tabla 14.15 – Prioridad entre interrupciones y excepciones simultaneas.

- Se salva en la pila el contenido del CS, IP y el registro de estado E-FLAGS. Además se pone a 0 los bits TF e IF.

- Busca el vector predefinido, mirando en la IDT. Si no viene predefinido, viene dado por la instrucción INTR que esta en los bits de menos peso del bus de datos (D0 - D7).

- Al finalizar la rutina de la interrupción, con la instrucción IRET, se saca de la pila los datos antes salvados, CS, IP y E-FLAGS para continuar con lo que antes estaba haciendo le CPU.

DIAGRAMA DE CONEXIONADO 15

15.1. INTRODUCCIÓN

El microprocesador es un sistema abierto, se comunica con el subsistema de memoria, elementos periféricos de muy diversa índole, dispositivos de E/S e incluso, con otros procesadores. Esta comunicación se realiza a través de las señales que salen y entran desde y hacia el procesador por las patitas de este. El conjunto de todas las señales constituye el bus propio del microprocesador. La disposición e identificación de dichas señales en un encapsulado constituyen su esquema de conexionado.

A lo largo del tiempo Intel ha presentado diferentes tipos de encapsulados, variando su presentación , el número y la disposición de las patitas. A la vez que los procesadores se hicieron más potentes, más rápidos y más complejos la necesidad de mejorar el encapsulado se ha ido incrementando.

El encapsulado PGA (Pin Grid Array) es la forma más común de presentación de los microprocesadores Intel y el que más éxito ha tenido a lo largo de los años. Este tipo de encapsulado está caracterizado por su forma rectangular y por la disposición de las patitas ordenadas en filas y columnas como si de una matriz se tratase.

El primer Pentium utiliza el encapsulado PGA 273, denominado así por los 273 pines que lo conforman.

Figura 15.1 – Encapsulado PGA 273

A partir del Pentium II Intel lanza un nuevo tipo de encapsulado el S.E.C.C. (Single Edge Contact Cartridge). Está formado por un cartucho que reúne: al procesador, la caché L2, otros componentes y un área de ventilación.

En el Pentium III se presentó la cápsula S.E.C.C-2 además del PGA 370.

El lanzamiento por parte de Intel del cartucho S.E.C.C. es una forma de continuar con su compromiso de proporcionar soluciones innovadoras y de calidad. Además este tipo de encapsulado fue registrado de tal forma que los competidores de Intel no pudieran usarlo.

Los encapsulados son el soporte físico del diagrama de conexionado. Las 273 patitas del encapsulado PGA 273 se agrupan en una matriz de 21 filas x 21columnas. A las filas se les asigna una letra y a las columnas un número, así por ejemplo la patita A17 (*Address 17),* que es la línea 17 del bus de direcciones, se determina con la fila T y columna 11.

No todas las patitas del diagrama de conexionado están ocupadas y hay una asimetría producida por la patita 273.

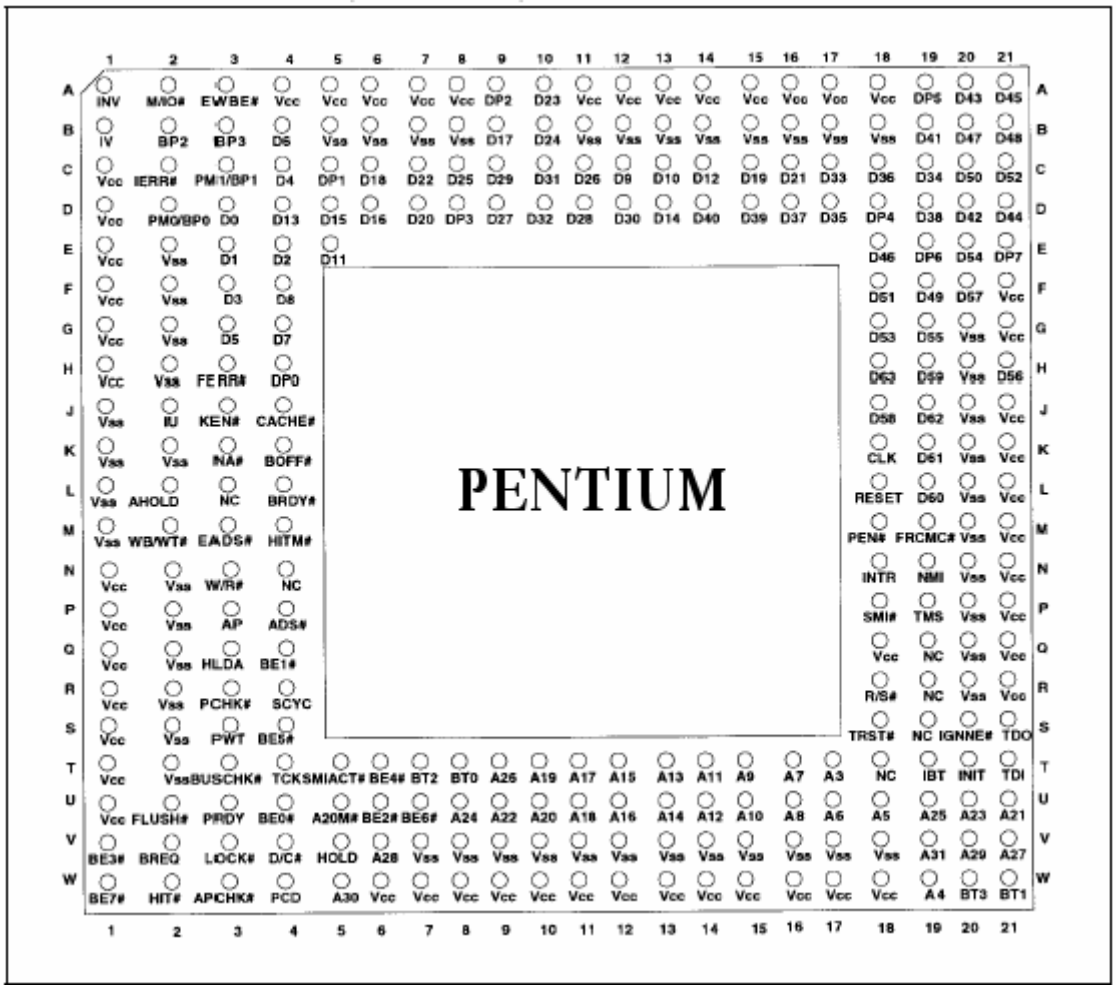


Figura 15.2 – Diagrama de conexionado del Pentium

A continuación se describe el diagrama de conexionado del Pentium. Para una mejor comprensión del mismo mostramos la siguiente figura en la cual se agrupan las señales por funciones.

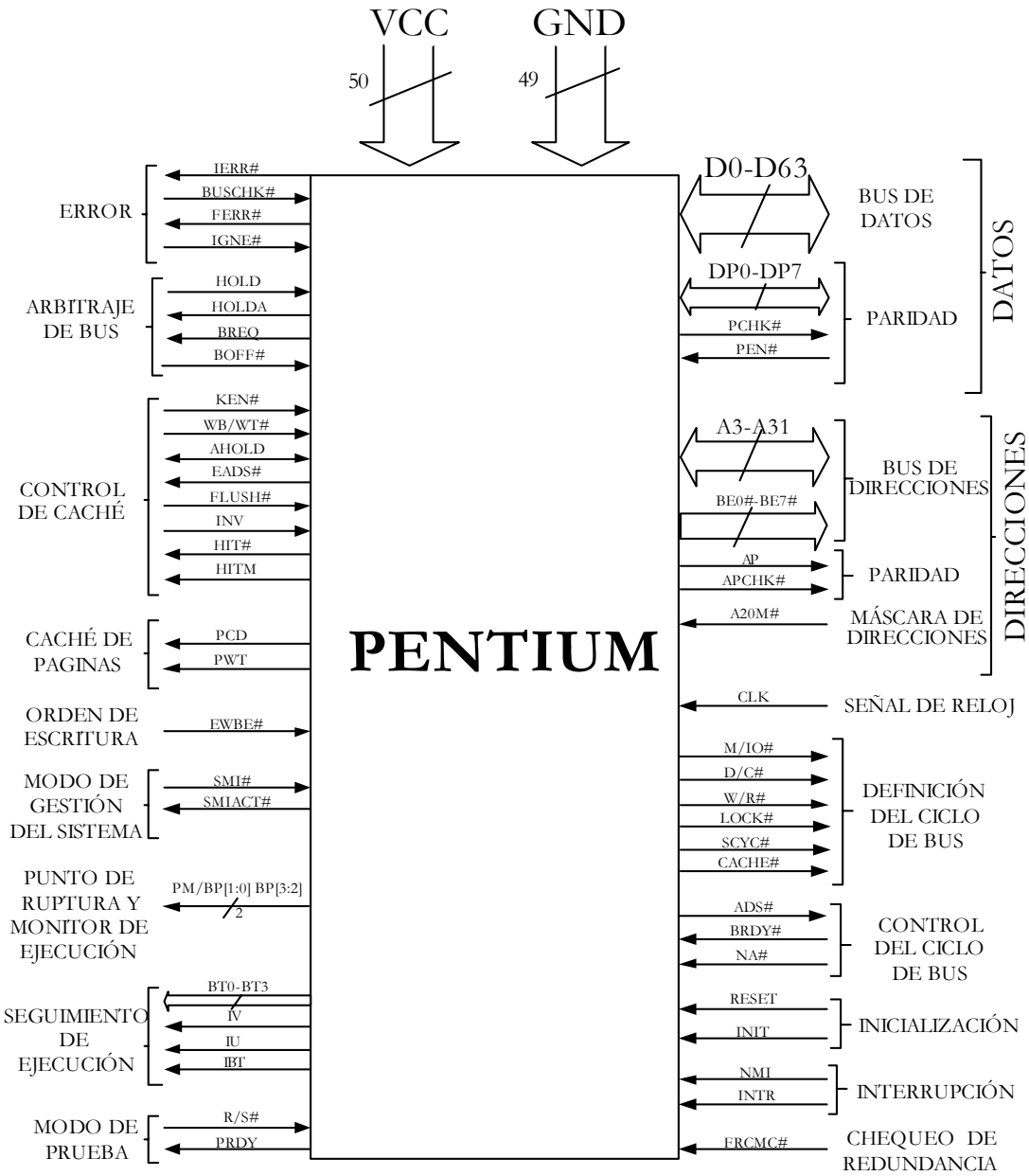


Figura 15.3 – Esquema de las señales del Pentium

15.2. ALIMENTACIÓN

El número de patitas del Pentium aumenta debido en parte a la alimentación. Dentro del procesador existen cada vez más circuitos, los cuales necesitan su propia alimentación para funcionar.

En el Pentium hay 50 patitas de alimentación conectadas a VCC y 49 patitas conectadas a GND.

El consumo normal del procesador empezó siendo 5V, pero como la disipación de potencia (13 W) y el calor (70 º C) eran muy altos, la tensión se ha ido rebajando continuamente. Incluso se utilizan refrigeradores de cualquier tipo para que no se queme el procesador.

15.3. SEÑALES DE DATOS

Agrupan las señales del bus de datos y las señales de paridad de datos.

15.3.1. Señales del bus de datos

Las señales del bus de datos son líneas bidireccionales triestado: tienen nivel alto, nivel bajo y alta impedancia. Las líneas del bus de datos se dejan en estado de alta impedancia para ser controladas por el exterior.

- **D0-D63 (E/S):** *Data lines*. Son 64 líneas de datos de entrada/salida del procesador. Las líneas D0-D7 definen el byte menos significativo del bus de datos y las líneas D56-D63 definen el byte de mayor peso del bus de datos.
  Como entradas se activan después de recibir la señal BRDY#.
  Como salidas están activas durante los estados T2, T12 y T2P del bus.

15.3.2. Señales de paridad de datos

El Pentium utiliza paridad par cuando envía información al exterior. Además tiene un sistema de detección de paridad cuando recibe información, en el caso de que el elemento exterior sea capaz de generar paridad.

Las señales de paridad de datos son las siguientes:

- **DP0-DP7 (E/S):** *Data parity*. Son 8 líneas de entrada/salida, una por cada byte del bus de datos, para indicar la paridad de los datos. Son enviadas en la misma señal de reloj que las líneas del bus de datos. DP0 acompaña al byte D0-D7, y DP7 acompaña al byte D56-D63.

- **PCHK# (S):** *Parity check*. Es una línea de salida que indica si se ha producido error en la paridad de los datos en el proceso de lectura. Se envía dos señales de reloj después de recibir la señal BRDY# y se mantiene activa durante una señal de reloj por cada error de paridad detectado.

- **PEN# (E):** *Parity enable*. Es una patita de entrada que se activa cuando el Pentium ha sacado información al exterior y el dispositivo externo detecta un error de paridad en los datos.

15.4. SEÑALES DE DIRECCIONES

Agrupa las señales del bus de direcciones, de máscara de direcciones y de paridad de dirección.

15.4.1. Señales del bus de direcciones

Las líneas del bus de direcciones son lineas bidireccionales triestado: tienen nivel alto, nivel bajo y alta impedancia. Se dejan en alta impedancia cuando son controladas por el exterior.

Observando cómo se agrupa se puede comprobar que la memoria se agrupa en posiciones de ocho bytes, por lo que el bus de direcciones no apunta a un byte sino a ocho. Se necesita por tanto 29 líneas para apuntar una posición de memoria.

- **A3-A31 (E/S):** *Address.* Son 29 líneas de entrada/salida. De salida porque el Pentium deposita en ellas la dirección física de la memoria donde se quiere acceder, y de entrada porque se debe poder acceder a la caché.
  Existen dos razones para acceder a la caché:

  -Para actualizar la caché:
  Cuando se produce un fallo en la caché, hay que acceder a la caché para actualizarla.

  -Para invalidar una línea de la caché:
  Cuando se escribe en la memoria principal hay que acceder a la caché para invalidar el contenido que ésta tiene.

- **BE7#-BE0# (S):** *Byte enable.* Son 8 líneas que se activan en función de los bytes del bus de datos que participan en la operación. BE7# corresponde al byte de mayor peso del bus de datos y BE0# al de menor peso.

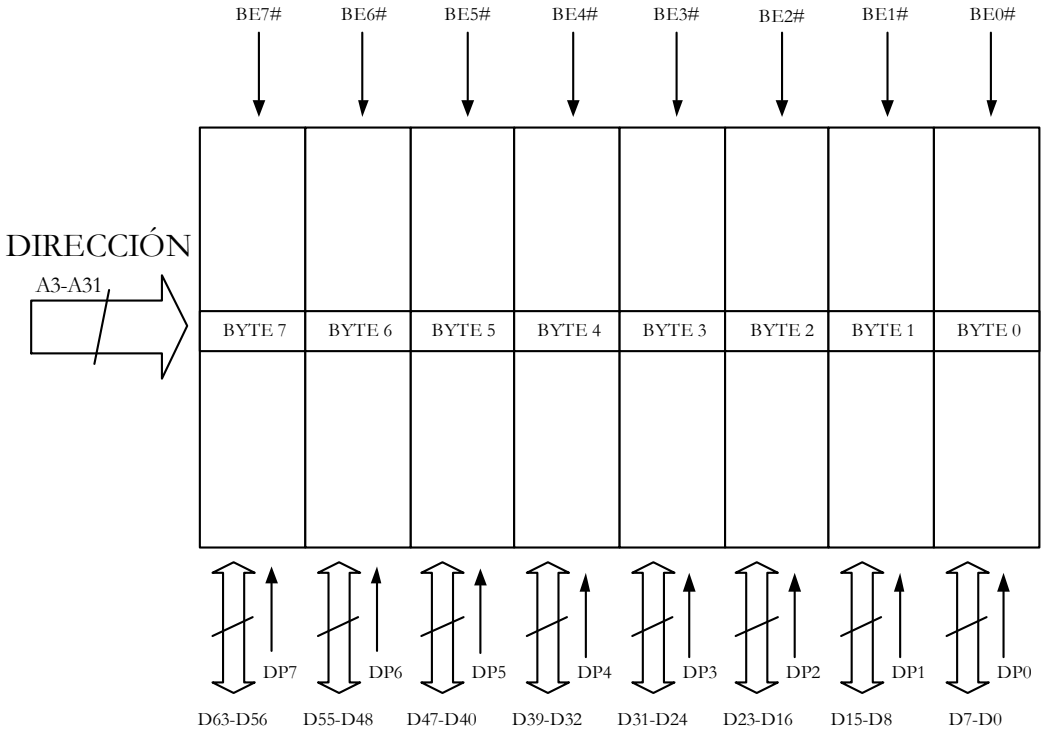


Figura 15.4 – Agrupación del bus de datos de 8 en 8 para cada byte de memoria.

15.4.2. Señal de máscara de direcciones

- **A20M# (E):** *Address bit 20 mask*. Es una línea de entrada que activa el elemento externo para indicar al Pentium que el bus de direcciones trabaja con el primer Mbyte del espacio de direcciones de la memoria. Cuando se activa sólo se utilizan las primeras 20 patitas del bus de direcciones.
  Esta patita sólo puede utilizarse cuando el procesador trabaja en modo real.

15.4.3. Señales de paridad de direcciones

- **AP (S):** *Address parity*. Es una línea de salida que activa el Pentium para generar paridad en las líneas del bus de direcciones.

- **APCHK# (S):** *Address parity check*. Es una línea de salida que indica si se ha producido error en la paridad del bus de direcciones.

15.5. SEÑAL DE RELOJ

La señal de reloj CLK es una línea de entrada que proporciona la frecuencia a la que trabaja internamente el Pentium.

15.6. SEÑALES DE DEFINICIÓN DEL CICLO DE BUS

Un ciclo de bus es el tiempo en el que la CPU realiza una transferencia de datos completa con el exterior, esto es, con la memoria o con los periféricos de E/S.

Las señales de definición del ciclo de bus son 6 líneas de salida:

- **M/IO# (S):** *Memory / Input-Output*. Es una patita que toma el valor 1 ó 0 si en el ciclo actual se accede a la memoria principal o al espacio de posiciones de E/S. Es válida en el mismo ciclo de reloj que la señales ADS#.

- **D/C# (S):** *Data / Code*. Esta patita toma valor 1 si es un dato y 0 si es el código de una instrucción ó se trata de un ciclo especial. Se activa en el mismo ciclo de reloj que ADS#.

- **W/R# (S):** *Write / Read*. Esta patita tomar valor 1 si es un ciclo de escritura y valor 0 si es un ciclo de lectura. Se activa en el mismo ciclo de reloj que ADS#.

- **LOCK# (S):** *Bus Lock*. El Pentium activa por nivel bajo esta patita para indicar al exterior que no va a ceder los buses hasta que finalice la instrucción que está ejecutando. Esto lo hace el Pentium porque está ejecutando una instrucción especial protegida, normalmente de lectura, escritura o modificación.

- **SCYC#(S):** *Split cycle*. Esta patita se activa para indicar que existen más de dos ciclos de LOCK# consecutivas.

- **CACHE# (S):** En un ciclo de lectura, se activa esta patita para avisar al exterior que hay espacio libre en la caché. Por tanto, si se lee de la memoria principal se puede guardar luego en la caché. Si no se activa, indica que no hay espacio en la caché.
  En un ciclo de escritura, el Pentium la activa si el ciclo es de escritura obligada.

15.7. SEÑALES DE CONTROL DEL CICLO DE BUS

- **ADS# (S):** *Address status.* Esta línea se activa cuando comienza un nuevo ciclo de bus válido.

- **BRDY# (E):** *Burst ready.* La activa el elemento externo para indicarle al Pentium que ha cogido el dato depositado por el Pentium en el bus de datos cuando se dispone a escribir o bien que el elemento externo ha depositado el dato en el bus de datos cuando el Pentium se dispone a leer. Esta señal indica el final del ciclo de bus y es activa en los estados T1, T2 y T2P del bus.

- **NA# (E):** *Next address.* Esta línea la activa el sistema exterior para avisar al Pentium que, aunque el ciclo de bus actual no haya finalizado, está preparado para recibir un nuevo ciclo de bus.

15.8. SEÑALES DE INICIALIZACIÓN

- **RESET (E):** *Reset.* La provoca un sistema externo cuando quiere inicializar el Pentium. Fuerza al procesador a comenzar la ejecución en un estado conocido. Todos las caches internas del procesador serán invalidadas. Las líneas modificadas en la caché de datos no se pueden deshacer.

- **INIT (E):** *Initialization.* El pin de entrada de *inicialización* del procesador fuerza al procesador Pentium a comenzar la ejecución en un estado conocido. El estado del procesador después de INIT es igual al el estado después del RESET excepto que los caches internas, buffers de escritura y los registros de la FPU conservan los valores que tenían antes de INIT. INIT no se puede utilizar en lugar de RESET después de encender el ordenador.

15.9. SEÑALES DE INTERRUPCIÓN

- **NMI (E):** *Non-maskable interrupt.* Señal de petición de interrupción no mascarable, a esta interrupción la CPU la atiende siempre con la entrada dos de la IDT. Indica que se ha generado una interrupción no mascarable externa.

- **INTR (E):** *Maskable interrupt.* Señal de petición de interrupción mascarable. Indica que se ha generado una interrupción externa. Si el bit IF en el registro EFLAGS está activo, el procesador atiende la interrupción, con el vector definido por los ocho bits de menos peso del bus de datos (D0-D7).Si por el contrario el bit IF = 0 se ignora la interrupción.
  INTR debe seguir activo hasta que el primer ciclo de reconocimiento de interrupción es generado para asegurar que la interrupción es reconocida.

15.10. SEÑALES DE ERROR:

- **IERR# (S):** *Internal error.* El pin interno de error se utiliza para indicar dos tipos de errores, errores de paridad internos y errores de comprobación de redundancia. Este segundo se debe a la capacidad de actuar en paralelo con otro procesador, uno maestro y otro esclavo comprobando posteriormente los resultados. Si ocurre un error entre el valor muestreado en los pins y el valor computado internamente, el procesador activará IERR# dos relojes después de que el valor erróneo se devuelva.
  Si un error de paridad ocurre en una lectura de un array interno, el procesador activará el pin IERR# en un ciclo de reloj y después parar.

- **BUSCHK# (E):** *Bus check.* La comprobación del bus de entrada, el sistema indica al Pentium que el ciclo de bus que estaba desarrollando no se ha completado con éxito.
  Si este pin es muestreado como activo, el procesador guardará la dirección y las señales de control en los registros de comprobación de la máquina. Si además, el bit MCE en CR4 está activo, el procesador Pentium se dirigirá a la excepción de comprobación de la máquina.

- **FERR# (S):** *Floating point error.* El pin del error de coma flotante, indica que ha ocurrido un error no mascarable en el coprocesador matemático FPU.
  FERR# es similar al pin ERROR# en el coprocesador matemático de Intel387.

- **IGNNE# (E):** *Ignore numeric error.* Entrada numérica del error ignore. Permite ignorar errores numéricos, se le indica al Pentium que ignore los errores de la FPU.
  Si se activa IGNNE# y el bit NE del registro de control CR0 está a 0, el procesador ignorará cualquier excepción numérica desenmascarada pendiente y continuará ejecutando las instrucciones en coma flotante. Si por el contrario NE está a 1, este pin no tendrá ningún efecto.
  Cuando el bit NE está a 0 e IGNNE# no está activo, existe una excepción numérica desenmascarada pendiente, y si la instrucción en coma flotante es una FINIT, FCLEX, FSTENV, FSAVE, FSTSW, FSTCW, FENI, FDISI, o FSETPM, el Pentium ejecutará la instrucción a pesar de tener una excepción pendiente. Si por el contrario la instrucción en coma flotante no es una de las anteriores el Pentium parará la ejecución y esperará una interrupción externa.

15.11. SEÑALES DE ARBITRAJE DE BUS

- **HOLD (E):** *Hold request.* Petición de los buses. El Pentium después de completar los ciclos de bus pendientes activa HLDA y deja en triestado el bus; se desconecta de él para que el elemento que realiza la petición lo utilice. El Pentium mantendrá su bus en este estado hasta que HOLD sea desactivado. Esta señal no se reconocerá durante ciclos de LOCK. El procesador reconocerá la señal HOLD durante el reset.

- **HLDA (S):** *Bus hold acknowledge.* Pin de reconocimiento del estado hold. Se activa en respuesta a una petición del bus, dirigida al procesador para el pin HOLD. Indica que el Pentium ha cedido el bus. Al salir de HOLD, HLDA se desactiva y el Pentium reasumirá el control del bus y si tiene algún ciclo del bus pendiente, será completado en el mismo ciclo de reloj en el que HLDA se desactive.

- **BREQ (S):** *Bus request.* Señal de petición del bus. Indica al sistema externo que el Pentium internamente ha generado una petición del bus. Esta señal se controlará siempre aunque el Pentium no esté controlando el bus.

- **BOFF# (E):** *Backoff.* Entrada que se emplea para abortar todos los ciclos de bus que todavía no se han completado, quedando congelados. Supone una parada brusca de la CPU. El Pentium permanece en este estado hasta que este pin no se desactive. Cuando se activa reanuda y completa los ciclos que no finalizó.

15.12. SEÑAL DE CHEQUEO DE REDUNDANCIA

- **FRCMC# (E):** *Functional redundancy checking master /Checker.* Entrada que se utiliza para determinar si el Pentium está configurado en modo maestro o en modo esclavo. Cuando está configurado como maestro, el procesador pone sus pins de salida según es requerido por el protocolo de bus. Cuando es configurado como esclavo, el procesador pone en triestado todas las salidas (excepto IERR# y TDO) y muestrea los pins de salida.
  La configuración como maestro / esclavo se fija después de RESET y no se podrá cambiar hasta otro RESET.

15.13. SEÑALES DE CONTROL DE CACHÉ

- **KEN# (E):** *Cache enable.* El pin que se utiliza para determinar si el ciclo actual es cacheable o no y por lo tanto determinar la longitud del ciclo.
  Cuando el Pentium genera un ciclo que puede ser cacheado (CACHE# activado) y KEN# es activo, el ciclo será transformado en un ciclo de relleno de línea.

- **WB/WT# (E):** *Writeback / Writethrough.* Entrada que permite que una línea de la caché de datos sea definida de escritura obligada o escritura diferida.
  Consecuentemente, se determina si la línea de la caché de datos está inicialmente en el estado S o E (protocolo MESI).

- **AHOLD (E/S):** *Assertion* of *address hold.* En respuesta a la activación de HOLD del bus de direcciones, el Pentium dejará de tratar las líneas de dirección (A31-A3), y del AP en el ciclo de reloj siguiente. El resto del bus seguirá siendo activo así que los datos se pueden volver o llevar a los ciclos previamente ocurridos del bus.
  Si es de entrada, le indicamos al Pentium que active la hold del bus de direcciones porque hay que invalidar una línea de la caché del Pentium, por ello el controlador de la caché le pide a la CPU active el hold del bus de direcciones, ya que el controlador es el que conoce la dirección de la línea a invalidar, que introducirá por el bus de direcciones.
  Si es de salida, el Pentium solicita el bus de direcciones para sacar la dirección de una línea de la caché de segundo nivel L2 para invalidarla o modificarla.

- **EADS# (E):** *Valid external address.* Esta señal indica que una dirección externa válida se ha llevado a los pins de dirección del Pentium. Indica que se va a iniciar un ciclo de bus especial, esta dirección corresponde a una línea de la caché externa que se va a invalidar.

- **FLUSH (E):** *Cache flush*. La activa un elemento externo para invalidar la caché interna. Se invalidan todas las líneas de la caché, dejan de tener validez, y esto fuerza la escritura obligada de todas las líneas modificadas de la caché de datos.
  El Pentium generará un ciclo especial indicando la finalización de la escritura obligada y de la invalidación.

- **INV (E):** *Invalidation*. Entrada de invalidación que determina el estado final de la línea de caché (S o I) en caso de que sea un ciclo de investigación. Si es muestreado junto con la dirección para el ciclo de investigación, EADS# será activado.

- **HIT# (S):** *Hit*. Se utiliza para reflejar el resultado de un ciclo de investigación. Si un ciclo de la investigación da una línea válida en cualquiera de los datos del Pentium o caché de instrucción, este pin es activado dos ciclos de reloj después de que se active EADS#.
  Si el ciclo de la investigación pierde la caché del Pentium, este pin es negado dos ciclos después que EADS#. Este pin cambia su valor solamente como resultado de un ciclo de la investigación y conserva su valor entre los ciclos.

- **HITM# (S):** *Hit to a modified line*. Se utiliza para reflejar el resultado de un ciclo de investigación. Se activa después de los ciclos que dieron lugar a un hit sobre una línea modificada en la caché de datos.
  Es utilizada para inhibir otro bus maestro de tener acceso a los datos hasta que la línea ha sido totalmente escrita.

15.14. SEÑALES DE CACHE DE PAGINAS

- **PCD (S):** *Page cache disable*. Pin que refleja el estado del bit PCD del registro de control CR3 de la entrada del directorio de páginas, o de la entrada de la tabla de páginas que apunta. El propósito de PCD es indicar si la página correspondiente es cacheable o no.

- **PWT (S):** *Page write through*. El pin que refleja el estado del bit PWT en CR3 de la entrada en el directorio de páginas, o de la entrada de la tabla de páginas que apunta. El propósito de PWT es indicar si la página correspondiente es de escritura obligada.

15.15. SEÑAL DE ORDEN DE ESCRITURA

- **EWBE# (S):** *External write buffer empty*. Es una línea que entra al Pentium durante un ciclo de escritura de la caché. Si es activa indica que para terminar la operación todavía queda un ciclo de escritura pendiente.

15.16. SEÑALES DEL SMM (MODO DE GESTIÓN DEL SISTEMA)

Las señales de modo de gestión del sistema son dos líneas, una de entrada y otra de salida.

- **SMI# (E):** *System management interrupt.* Es una línea de entrada. Cuando se activa, el procesador entra en el modo de gestión del sistema.

- **SMIACT# (S):** *System management interrupt active.* Es una señal de salida que indica al exterior que el procesador está trabajando en el modo de gestión del sistema.

15.17. SEÑALES DE PUNTO DE RUPTURA (BP) Y MONITOR DE EJECUCIÓN (PM)

- **PM/BP[1:0] BP[3:2]:** *Performance monitoring /Breakpoint.* Los puntos de ruptura BP [1:0] están multiplexados con las lineas del monitor de ejecución PM [1:0]. Los bits PB0 y PB1 del registro de control de depuración determinan si las lineas están configuradas como puntos de ruptura o como líneas del monitor de ejecución.
  Cuando se produce un reset, dichas líneas quedan configuradas como líneas del monitor de ejecución.
  Los puntos de ruptura BP0-BP3 corresponden con los registros de depuración DR0-DR3. Indican un punto de ruptura cuando los registros de depuración están programados para testear puntos de ruptura.

15.18. SEÑALES DE SEGUIMIENTO DE EJECUCIÓN

- **BT3-BT0 (S):** *Branch trace.* Proporciona los bits 2-0 de la dirección lineal de la bifurcación (BT-BT0) y el tamaño del operando por defecto (BT3) durante un ciclo especial. Si el bit BT3 está activo el tamaño será 32 bits, si no lo está el tamaño será de 16 bits.

- **IV (S):** *V_pipe instruction complete.* Se activa por nivel alto en un ciclo de reloj para indicar que la instrucción en el cauce V ha finalizado su ejecución. Este pin es manejado siempre por el procesador.

- **IU (S):** *U_pipe instruction complete.* Se activa por nivel alto en un ciclo de reloj para indicar que la instrucción en el cauce U ha finalizado su ejecución. Este pin es manejado siempre por el procesador.
  IV e IU se utilizan para el control de la segmentación de forma externa.

- **IBT (S):** *Instruction branch taken.* Se activa por nivel alto en un ciclo de reloj para indicar que ha ocurrido una instrucción de salto. Esta salida es manejada siempre por el procesador.

15.19. SEÑALES DEL MODO DE PRUEBA

- **R/S# (E):** La entrada R/S# es una interrupción asíncrona, usada para parar la ejecución normal del procesador y para ponerla en estado IDLE. Un flanco descendente en el pin R/S# interrumpirá el procesador y hará parar la ejecución en el límite con la siguiente instrucción.

- **PRDY (E):** Indica que el procesador ha parado la ejecución normal en respuesta a la puesta a 1 del pin R/S#, o a la activación de modo de prueba *"probe mode"*.

EL BUS Y LOS CICLOS DE BUS 16

16.1- INTRODUCCIÓN

En este tema se estudiarán los ciclos de bus y los diferentes términos en relación con éstos.

Para leer y escribir datos, el Pentium debe ser capaz de transmitir y recibir varias señales de control y por supuesto también necesita energía para su circuitería.

El bus del procesador Pentium puede funcionar en un modo llamado *pipeline*. En este modo, se puede comenzar un segundo ciclo de bus antes de haber terminado el primero. Frecuentemente las instrucciones y datos requeridos se encuentran en cualquiera de las dos cachés, (en algunos casos tres, como en el Pentium 4 Xenon con una L3 de 1MB) que pueden ser leídas y escritas en un único ciclo de reloj. Para la conexión con la caché de segundo nivel (caché nivel L2), el bus de datos ha de ser ampliado a 64 bits, con lo que las cachés pueden ser leídas y escritas lo suficientemente rápido: 32 bits son insuficientes si los tres 'pipelines' ejecutan una instrucción por cada ciclo de reloj (sin bloqueos) y adicionalmente se ejecuta un acceso a memoria cuando sea posible. El Pentium intenta ejecutar accesos a memoria cuando sea posible. El Pentium intenta ejecutar accesos a memoria lo más rápido posible, tanto en llenado de líneas caché como en recuperaciones.

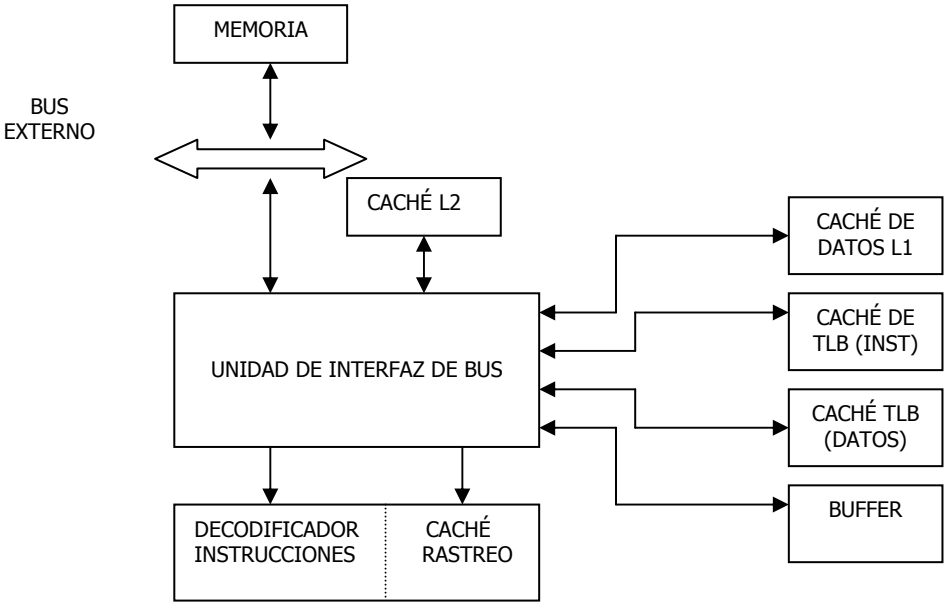


Figura 16.1 Ejemplo de estructura con tres cachés para el Pentium 4 Xenon

Se conoce como ciclo de bus al tiempo en el que la CPU realiza una transferencia de datos completa con el exterior, es decir, con memoria o con los periféricos de entrada/salida. Como mínimo se compone de dos estados T1 y T2.

El procesador realiza las siguientes acciones:

- Coloca en el bus la dirección a acceder
- Activa las señales de control que indican el tipo de transacción
- Transfiere o recibe el dato
- Las acciones se realizan de forma sincronizada y controladas por una unidad interna del procesador: la Unidad de Bus

16.2- CARACTERÍSTICAS DEL CICLO DE BUS

La frecuencia de funcionamiento del Pentium se denomina ciclo de reloj interno (CLK). También existe el ciclo de reloj externo (CLK2) que no es otro que el que se introduce por la patita CLK2 y que tiene el doble de la frecuencia a la que funciona la CPU.

Ejemplo: CLK f=20 Mhz
CLK2 f=40 Mhz

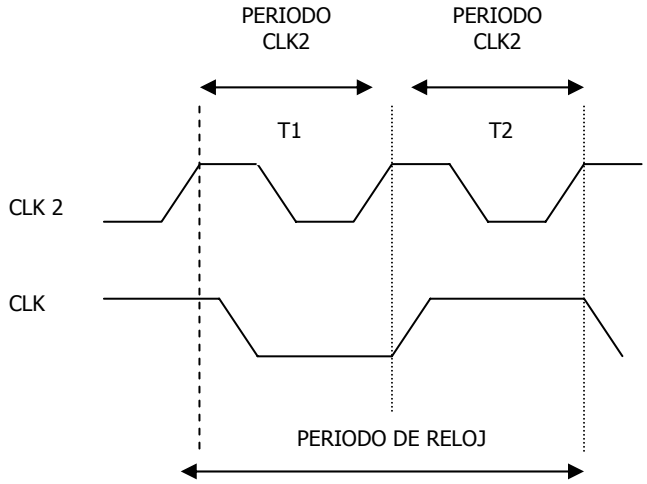


Figura 16.2 Frecuencia de trabajo del Pentium

Se denomina periodo de la CPU , y se representa por T_{CPU} , al tiempo que dura un ciclo de trabajo.

Un estado de bus (T) es el tiempo mínimo en el que la CPU puede realizar una operación elemental y dura un periodo de CLK, o lo que es lo mismo, dos ciclos de CLK2.

Un ciclo de bus es el tiempo en el que la CPU realiza una transferencia de datos completa con el exterior, o sea, con la memoria o con los periféricos de E/S. Como mínimo se compone de dos estados: T1 y T2.

Existen diferentes tipos de ciclos de bus en el Pentium. A continuación se indican las tres principales categorías en las que pueden ser divididos y mas adelante se profundizará mas en cada uno de ellos:

–Ciclo de bus en modo Sencillo o Simples (no burst):
–Ciclo de bus en modo ráfaga(Burst Bus Cycle): Para transferir grandes cantidades de datos, por ejemplo una línea de la caché. Si la línea es de 256 bits (32 bytes) y el ancho del bus es de 64 bits (8 bytes), en una transferencia normal necesitaríamos 4 ciclos de bus (8 ciclos de procesador). De esta forma, la transferencia se realiza en 5 ciclos de procesador.
–Ciclos especiales: Reconocimiento de interrupción, "shutdown",etc.

Cada vez que se activa ADS# comienza un nuevo ciclo de bus y al mismo tiempo aparece una dirección válida por las líneas A2-A31 / BE0#0- BE3#, quedando definido el tipo de ciclo de bus con las señales de control M/IO, D/C# y W/R#.

TIPOS DE CICLO DE BUS	M/IO#	D/C#	W/R#
Lectura de datos E/S	0	1	0
Escritura de datos E/S	0	1	1
Lectura de datos de memoria	1	1	0
Escritura de datos de memoria	1	1	1
Lectura de código de memoria	1	0	0
Reconocimiento de interrupción	0	0	0
No válido	0	0	1
Tipo Especial	1	0	1

Tabla 16.1. Tipos de Ciclos de Bus

Cuando se produce la combinación 101 pueden ocurrir varios tipos de ciclos especiales. Estos a su vez se recogen en la tabla 16.2., la cual dirá el Byte que se ha de habilitar en cada caso.

De este modo, por ejemplo, el ciclo de bus tipo HALT se produce al ejecutarse la instrucción HLT. Cuando se da este tipo de bus, por el bus de direcciones sale A2-A31 y BE2#=0.

El ciclo SHUTDOWN se genera al estar generándose una excepción de "doble fallo" y producirse otra excepción de "fallo de protección", es decir, una violación de las reglas de protección. En este caso, por el bus de direcciones sale A2-A31=0 y BE0#=0.

TIPO ESPECIAL DE CICLO	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#
REPOSO (SHUTDOWN)	1	1	1	1	1	0
INVALIDAR CACHÉ(FLUSH)	1	1	1	1	0	1
PARADA(HALT)	1	1	1	0	1	1
WRITE BACK (Escritura obligada)	1	1	0	1	1	1
Reconocimiento de Flush	1	0	1	1	1	1
Mensaje de Rastreo de Etiqueta	0	1	1	1	1	1

Tabla 16.2. Tipos de Ciclos de Bus Especiales

Cuando el procesador no está realizando ninguno de los ciclos de bus existentes, significa que está en estado de reposo (Ti) o en estado de reconocimiento de HOLD (Tn). El estado de reposo queda identificado porque desde el último ciclo de bus no se ha activado la señal ADS#. El estado de reconocimiento de HOLD se da cuando otro maestro pide los buses a la CPU y este le cede los buses, dejándolos en triestado, y se identifica comprobando la habilitación de la señal de salida HLDA.

16.3- CARACTERÍSTICAS DE LA MEMORIA Y SUBSISTEMAS DE E/S

El Pentium contiene un bus externo de 64 bits.

Internamente el Pentium es un procesador de 32 bits.

Tanto la caché externa L2 como la memoria principal están organizadas como una memoria de 64 bits. A pesar de esto el Pentium puede direccionar bytes individuales, palabras o dobles palabras por medio de las señales de activación de byte BE7#-BE0#. Cada ciclo de bus direcciona memoria por medio de A31-A3, en múltiplos de 8.

El espacio de memoria principal del Pentium comprende 4 Gigabytes. Por supuesto, la memoria también puede configurarse como una memoria de 32, 16 u 8 bits. Los bits de direccionamiento A2,A2 y A1,A2-A0, respectivamente deben ser decodificados de las señales de validación de octetos (BEx).

El Pentium incluye un modo burst o por ráfagas para un llenado y salvado más rápido de las líneas caché. Ha sido mejorado para poder leer y escribir el doble de bytes, usando la duplicación del ancho de bus. El Pentium intenta realizar todas las lecturas y escrituras como si fueran ciclos burst o por ráfagas. Solo los accesos al área de memoria de E/S son ejecutadas como ciclos sencillos de transferencia.

Al contrario que los accesos de memoria, los accesos al área de direcciones de E/S no han sido ampliados a 64 bits. La anchura máxima es de 32bits. La razón de esto es que en principio los accesos a E/S no pasan por caché. Así la caché sólo afecta a memoria y no a los puertos de E/S. El Pentium puede también direccionar puertos de 8 y también de 16 o 32 bits siempre que las direcciones de memoria sean contiguas. El subsistema de E/S debe decodificar los bits A2-A0 de las señales de activación de bits BE7#-BE0#.

16.4- SEÑALES ENTRE LA CPU Y LA MEMORIA

Para leer o escribir datos en memoria el Pentium tiene que direccionar físicamente la memoria (usando el bus de direcciones – A:A32 y BE0#BE7#) y transferir los datos (por el bus de datos). Así, un ciclo de bus en el que los datos son leídos o escritos, se sigue un a secuencia definida y estricta de señales de direcciones, datos y control. Se dispone de un controlador de bus adicional, que hace que todas las señales estén disponible para el dispositivo e memoria. Hoy en día, el controlador de bus es, junto con los buffers de direcciones y de datos y circuitería adicional de control, la parte principal de un controlador de sistema altamente integrado.

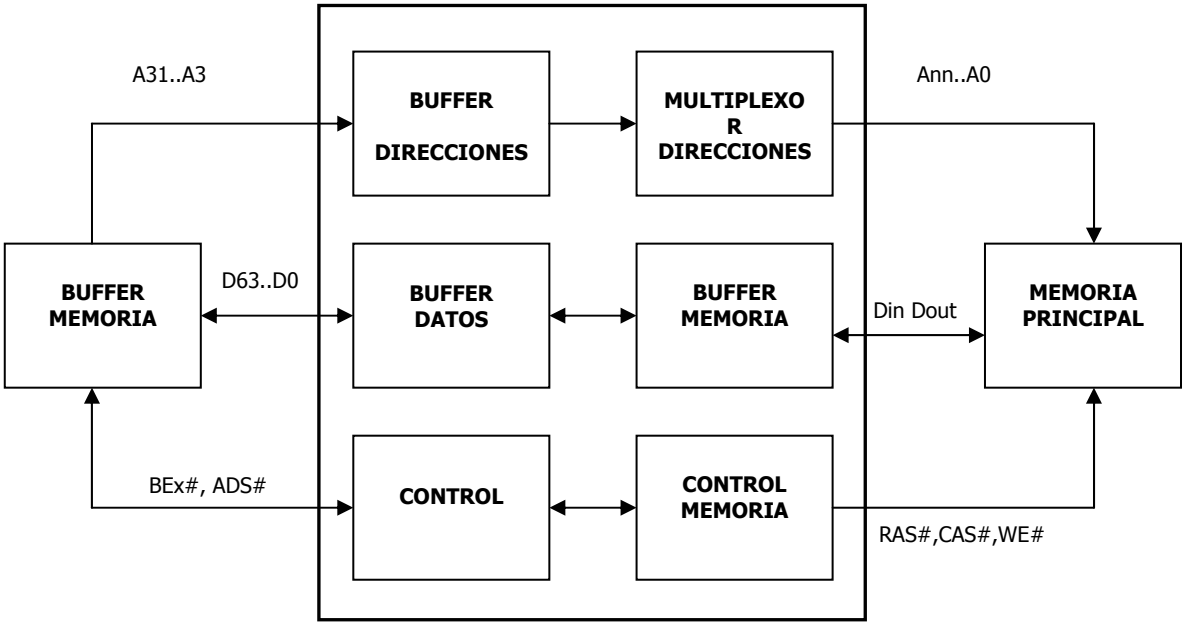


Figura 16.3 Controlador del Sistema

Si el Pentium desea leer o escribir datos de memoria, se transmite una dirección al buffer de direcciones por medio de su bus de direcciones.

Si la señal ADS# está activa, el buffer de direcciones acepta los bits de la dirección y los almacena.

BE0#-BE7# definen qué línea del bus de datos debe transferir los datos.

Para direccionar la palabra correcta en la unidad principal de almacenamiento, se dispone de un multiplexador de direcciones (que selecciona la palabra seleccionada de la memoria principal con la asistencia del controlador de memoria).

Los datos que han sido leídos son entonces transferidos de la memoria principal al buffer de memoria.

Posteriormente, el buffer de memoria transfiere los datos al buffer de datos, del cual el Pentium puede leer los datos. La transferencia de datos sólo es posible por medio del uso de señales eléctricas, no mediante instrucciones software.

16.5- CICLOS DE MEMORIA

El Pentium realiza los mismos ciclos independientemente de cuando el sistema de memoria externa represente una rápida memoria caché de segundo nivel (L2) o una lenta memoria principal DRAM. La única diferencia es que en el último caso, la señal BRDY# se devuelve más tarde así el procesador inserta más estados más estados de espera.

16.5.1- Posibles estados de un ciclo

El bus del procesador Pentium tiene seis estados de bus. Estos estados son los mostrados a continuación.

Ti Estado de Reposo o Inactividad. Este estado indica que no se está ejecutando ningún ciclo de bus. Ti es la condición de espera o preparado.

T1 Tiempo de direccionado. La condición de bus correspondiente al primer ciclo de reloj de un ciclo de transferencia de bus. La dirección y definición del ciclo de bus se dan durante T1 cuando ADS# es impuesta y con esto proporciona la dirección y el tipo de ciclo. T1 indica que este es el único bus de ciclo en ejecución.

T2 Tiempo de datos. Corresponde al segundo estado del ciclo de reloj de un ciclo de transferencia de bus, que puede ser el último si se activa #READY, o que puede repetirse (estado inactivo) hasta que se produzca dicha activación. Durante T2, se realiza la escritura o los datos leídos son transferidos. Si se ha dado un ciclo de lectura, los datos son protegidos del bus de datos cuando BRDY# es impuesta al final de T2.

T12 Tiempo de direccionado (segundo ciclo en modo pipeline) y Tiempo de datos (primer ciclo ya en progreso). En esta condición de bus, hay dos ciclos de reloj pendientes. El primer ciclo aun no está completo (y esta en la condición T2), y el segundo está recién enviado (que está en condición T1).T12 sólo ocurre si el Pentium ya ha enviado la dirección para la transferencia posterior antes de que el ciclo de bus actual haya sido completado. Resumiendo, el procesador está todavía en el estado T2 durante este ciclo y ha comenzado el estado T1 para el próximo ciclo en modo 'pipeline', de aquí que este sea el estado compuesto T12.

T2P Tiempo de datos(primer ciclo) y Tiempo de datos (segundo ciclo en modo 'pipeline'). Este estado representa dos ciclos de reloj que están pendientes en el bus y que además ambos están en el estado T2. El primer ciclo aún no está completo (y está en la condición T2), y el segundo está en el segundo, o posterior ciclo de reloj (también en condición T2). T2P sigue a T12 si, T12 ha sido completado, pero el primer ciclo aún no ha finalizado.

TD Estado Muerto (Dead State). Indica un ciclo pendiente, donde el Pentium , debe primero permitir transcurrir a un ciclo de reloj, para ejecutar una operación de escritura después de una operación de lectura y viceversa. Este estado ocurre únicamente durante transferencias en modo 'pipeline'.

Las transiciones descritas ocurren entre estados según se muestra en el siguiente diagrama de estados.

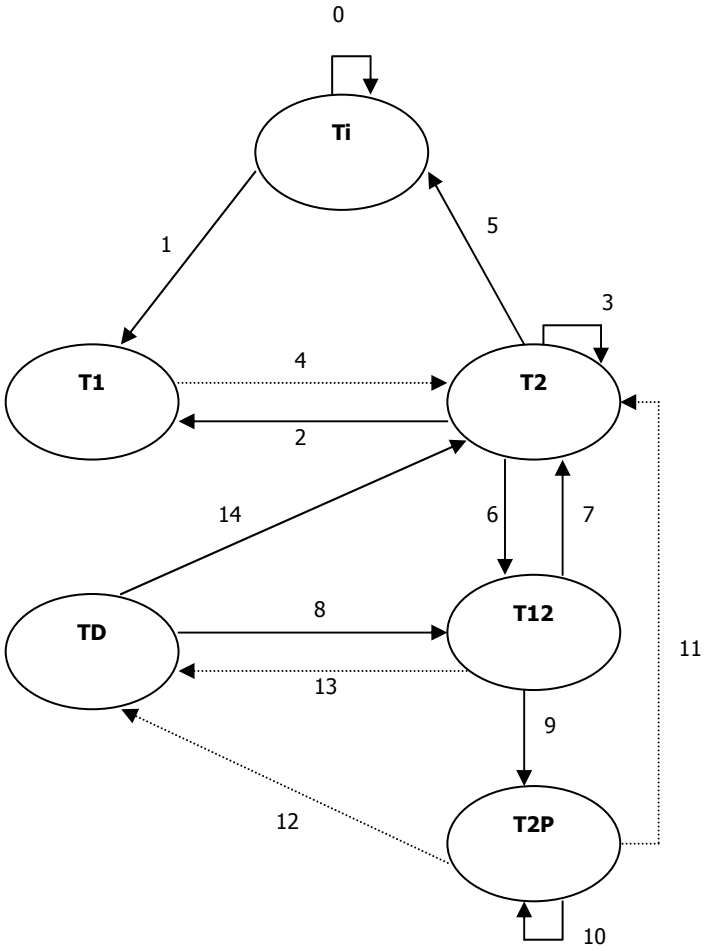
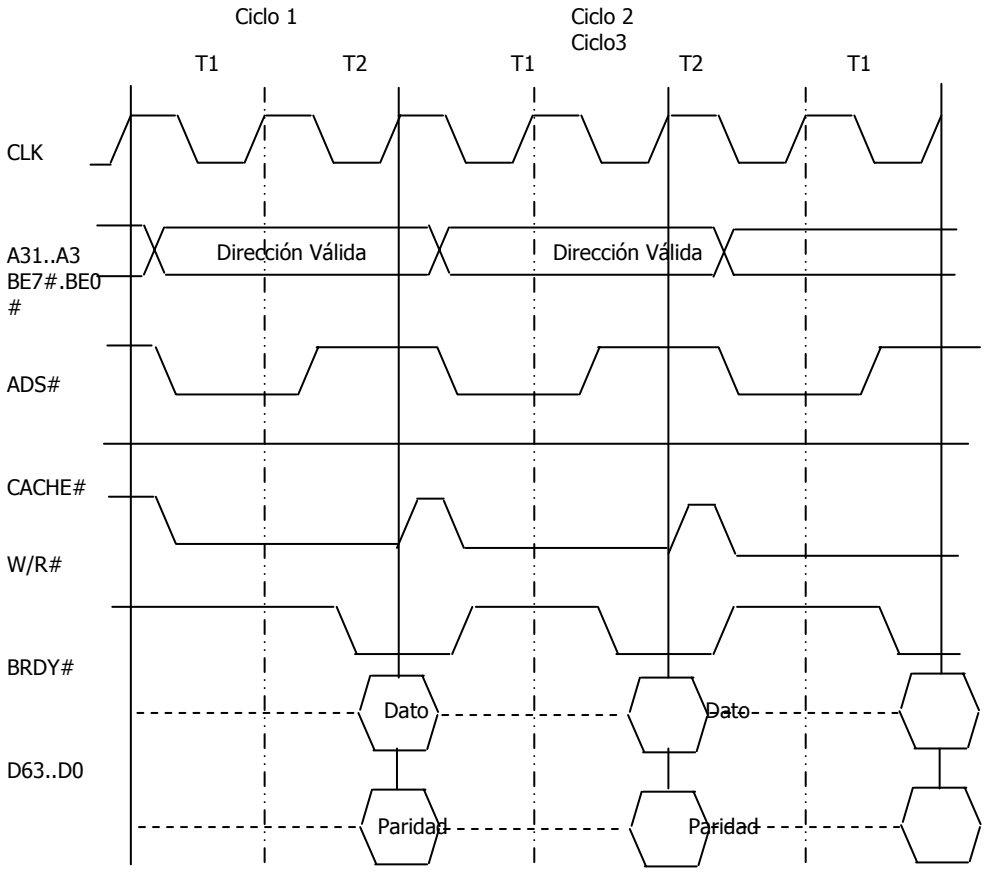


Figura 164 Diagrama de estado del procesador

A continuación se procederá a la descripción de los estados de Transición.

0: No hay petición pendiente.
1: Se comienza el ciclo de bus cuando el procesador lleva a los demás buses la dirección y definición del ciclo de bus.
2: De T1 a T2 siempre , a menos que BOFF# sea afirmada, en cuyo caso el ciclo se termina para retomarse cuando la señal sea liberada.
3: Al añadir estados de Reposo, el Pentium se queda en T2 hasta que la ultima BRDY# sea muestreada , hasta que NA# sea afirmada y otro ciclo esté pendiente.
4: De T2 a T1 cuando termine el ciclo de bus . si NA# no estaba afirmada el estado de transición vueve a Ti



5: De T2a Ti cuando acaba el ciclo de bus, si NA# estaba afirmada o si el otro ciclo se queda pendiente.
6: De T2a T12 si NA#estaba afirmada y esta pendiente otro ciclo de bus.
7: T12 pasa a T2. El ciclo ha terminado.
8: De t12 a TD. _El ciclo de bus ha terminado, pero en este caso se necesita un reloj que indique el fin.
9: De t12 a T2P, dos ciclos están pendientes coincidiendo ambos con la fase de datos.
10: Se espera en la etapa T2P hasta que se complete la primera transferencia.
11: De T2P a T2 cuando la primera transferencia se completa si no se necesita reloj que indique el final.
12: De T2P a TD cuando la primera transferencia (una lectura) se ha completado y la segunda es una escritura , o viceversa.
13: El estado de bus deja Td para volver T12 cuando NA# esté afirmada.
14: TD a T2 si no hay ciclo de bus pendiente.

16.5.2- Ciclos simples de transferencia

Los ciclos simples de transferencia en lectura y escritura son los dos ciclos de acceso de memoria más simples del Pentium. Durante su ejecución, los datos de 8, 16, 32 ó 64 bits son transferidos de la memoria al Pentium o viceversa.

En el modo de transferencia sencillo (escritura o lectura) una transferencia de datos sin estados de espera requiere al menos dos ciclos de reloj.

16.5.2.1- Ciclo de Lectura

Figura 16.5 Ciclo de lectura sin estado de espera

Durante el primer ciclo de reloj T1, el Pentium primero envía:

- Las direcciones A31:A3
- Las señales BE0:BE7 correspondientes al tamaño del dato en el acceso
- Las señales de control W/R# y CACHE# (inactiva para este ciclo para indicar que es un ciclo de transferencia simple)

Finalmente, activa la señal ASD# para indicar la validez de la dirección y de las señales de control del bus.

Después de un período de tiempo, dependiendo de la velocidad del subsistema de memoria, el subsistema transmite los datos direccionados y activa la señal de preparado BRDY#. El subsistema puede, mas o menos, opcionalmente transmitir los bits de paridad correspondientes a las señales activas BE0:BE7.

Para el ciclo de lectura de memoria descrito, los bits de datos y paridad están inmediatamente disponibles, por lo que no se requieren estados de espera de espera. Así BRDY# es activado por el subsistema de memoria antes del final del ciclo T2.

16.5.2.2- Ciclo de escritura (sin estados de espera)

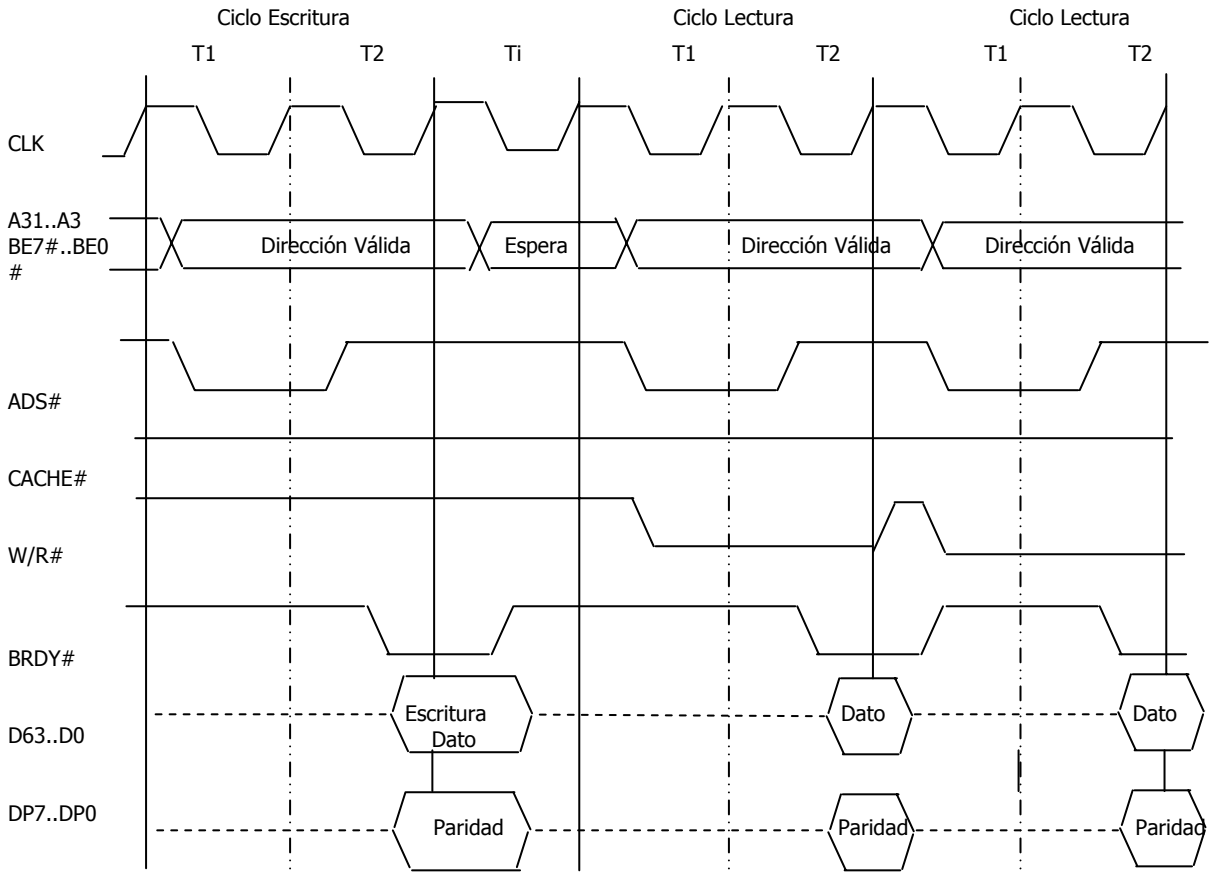


Figura 16.6 Ciclo de escritura sin estado de espera

En este caso el Pentium primero envía:

- Las direcciones A31-A3,
- Las señales BE0:BE7 correspondientes al tamaño del dato en el acceso,
- Las señales de control W/R#(nivel alto: acceso a escritura) y CACHE# (inactiva para indicar ciclo de transferencia).

Finalmente el Pentium activa la señal ASD#, para indicar la validez de las direcciones y las señales de control del bus.

Durante T2, el Pentium también transmite los datos a escribir y los bits de paridad necesarios. El subsistema toma los datos y entonces devuelve una señal activa BRDY#. Ahora el Pentium está listo para el siguiente ciclo.

16.5.3- Estados o ciclos de espera

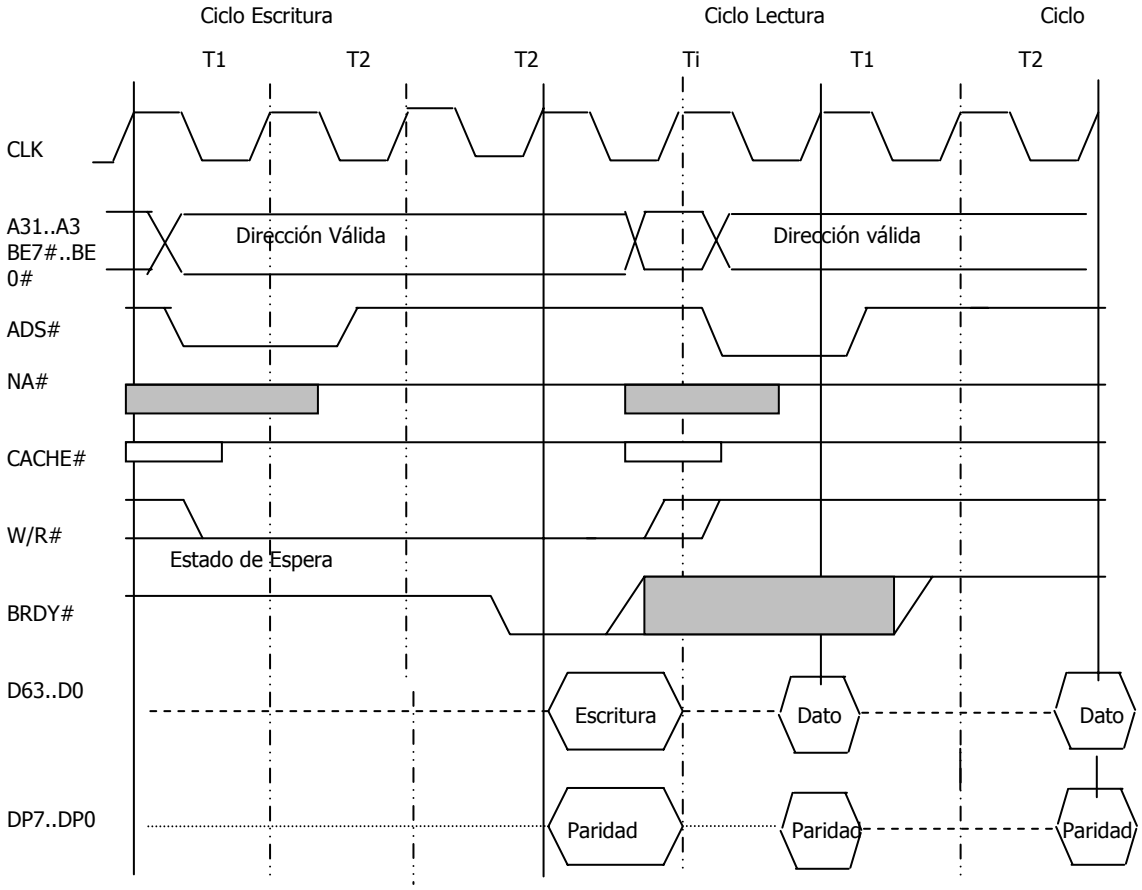


Figura 16.7. Ciclo de escritura básico

Si la memoria o el dispositivo periférico no puede terminar una petición de escritura o lectura en los dos ciclos T1 y T2, entonces el controlador de memoria cambia la señal BRDY# a nivel alto. Esto indica al Pentium que debería implementar otro ciclo de instrucción T2, para dar a la memora o al periférico más tiempo para ajustarse a la respuesta. Esto es conocido como ciclo de espera (Ti)

o un estado de espera. Si, complementariamente al ciclo adicional T2, la señal BRDY# continúa a nivel alto, entonces el procesador inserta otro ciclo de espera (y otro más mientras la señal continúe en nivel alto).

Por supuesto, el número de ciclos de espera necesarios para escribir datos puede ser distinto de los necesarios para realizar una lectura. Las memorias DRA pueden de hecho escribir más rápido de lo que leen. Puesto que cuando se escribe sólo es necesario transferir la dirección del controlador de memoria y el valor de byte de datos.

El controlador de memoria ejecuta el proceso de escritura independientemente usando los datos almacenados o el buffer de memoria, mientras el Pentium puede dedicarse a otro proceso y no tiene que esperar a que concluya el proceso de escritura. Por el contrario, cuando lee, la CPU no tiene otra opción más que esperar a la terminación del proceso interno de lectura en el área principal de almacenamiento.

Antes, los ciclos de espera venían determinados por el diseño de la placa o podían ser determinados con un jumper, dependiendo de lo rápidos que fueran los chips de memoria instalados. Hoy en día, el controlador de memoria reacciona de una manera standard ante un determinado retardo de la señal BRDY#.

16.5.4- Límites para las cuádruples palabras

El Pentium es un procesador de 32 bits con un bus de datos de 64 bits. Así la memoria principal está normalmente representada como una memoria física de 32bits. En realidad, las cuádruples palabras (64 bits) pueden comenzar en direcciones de memoria que no son múltiplo de 8. Pero en este caso no serían leídas en un único acceso a memoria.

Si una cuádruple palabra es almacenada o leída en una dirección en la cual no cabe, el interfaz de bus del Pentium divide el acceso en dos: el primer acceso lee o escribe la parte menos significativa y el segundo acceso la más significativa.

Por tanto que la escritura o lectura de una cuádruple palabra que no es múltiplo de 8, siempre requiere de 2 ciclos de bus, mientras que el acceso a una cuádruple palabra en una dirección múltiplo de 8 requiere únicamente de un ciclo de bus.

En ambos casos se direcciona una cuádruple palabra, pero sólo los bytes del 1 al 7 son combinados como una única cuádruple palabra de 64 bits.

Este procedimiento sólo se aplica a datos porque la selección de código se realiza por medio de la caché de instrucciones. La transferencia de los códigos de operación de la caché de instrucciones al preselector puede comenzar en cualquier dirección. Para leer una dirección de la cuádruple palabra, si es necesario, el procesador lee de uno a siete bytes primero, antes de comenzar en los límites de la cuádruple palabra.

16.5.5- Ciclos por ráfagas (burst)

Para la transferencia de grandes cantidades de datos, el Pentium implementa un modo de bus conocido como modo burst o por ráfagas.

El tiempo para una transferencia de un valor es reducido a un ciclo de reloj simple. El Pentium usa el modo burst o por ráfagas para ciclos de recuperación y lectura de la caché.

Se usan cuatro ciclos de bus para transferir 32 bytes (recordemos que un ciclo tiene 8 bytes).

Para las señales BRDY#, CACHE# y KEN#, el Pentium decide independientemente cuando debe ejecutarse un ciclo de bus burst o por ráfagas, o si sólo un ciclo de transferencia simples es suficiente.

Un ciclo de burst o por ráfagas generalmente comienza con un acceso normal a memoria (que requiere de ciclos de reloj).

16.5.5.1- Ciclo de Lectura

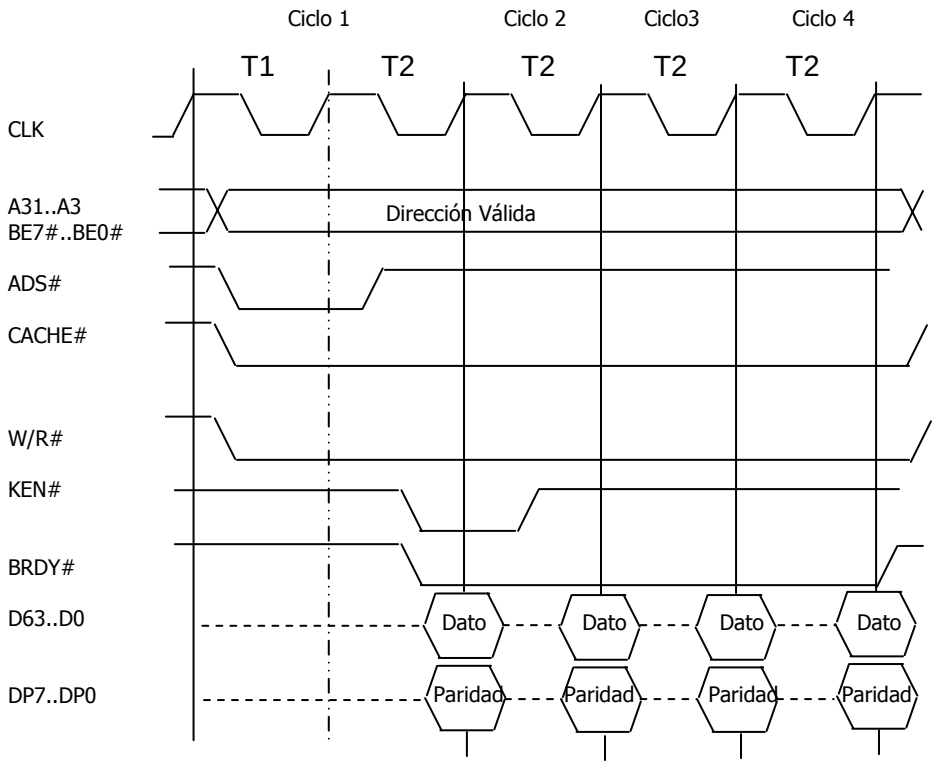


Figura 16.8. Ciclo de lectura por ráfagas

Si el Pentium inicia un ciclo de lectura , primero envía la dirección de memoria del objeto a ser leído por medio del bus de direcciones, y activa la señal ASD# para indicar que la dirección y la señal de control son válidas.

Para un ciclo burst o por ráfagas, la señal CACHE# del Pentium y la señal KEN# constituyen un papel importante. La señal CACHE# a nivel bajo, que el objeto direccionado puede ser transferido a la caché. Si la señal KEN# está activa, el Pentium de forma independiente y automática convierte la transferencia simple a un llenado de fila de caché, para almacenar el objeto direccionado en la caché. El llenado de la fila de caché es necesario para esto porque ningún byte individual, palabras, dobles palabras o cuádruples palabras es almacenado en la caché; por el contrario se utilizan siempre líneas completas de 32 bytes. El subsistema de memoria sólo activa la señal KEN# si el objeto referenciado no se encuentra en el área bloqueada. Un ejemplo de área bloqueada es el mapa de registros en memoria, que es direccionado por medio del espacio normal de direcciones de memoria pero que realiza funciones de control.

Como los ciclos burst o por ráfagas están limitados a un área de memoria que comienza en el límite de 2 bytes, después de recibir la dirección, el subsistema de memoria puede independientemente calcular las otras tres direcciones burst, sin la necesidad de decodificar ninguna

señal del Pentium. El Pentium sólo envía la dirección y las señales BEx durante el primer ciclo; estos valores no se cambian durante los otros tres ciclos posteriores. Para satisfacer las limitaciones y ajustarse a las particiones de 32 bytes, se ha definido una secuencia específica de direcciones en el modo burst o por ráfagas.

Las señales no son muy distintas a las del ciclo de lectura. Sólo es diferente la señal W/R#, que está a nivel alto para indicar la transferencia de escritura.

Adicionalmente, la señal KEN# se ignora porque todos los datos disponibles en la caché son obviamente cacheables y se vaciaría una entrada en la caché y no sería rellenada.

16.5.5.2- Ciclo de Escritura

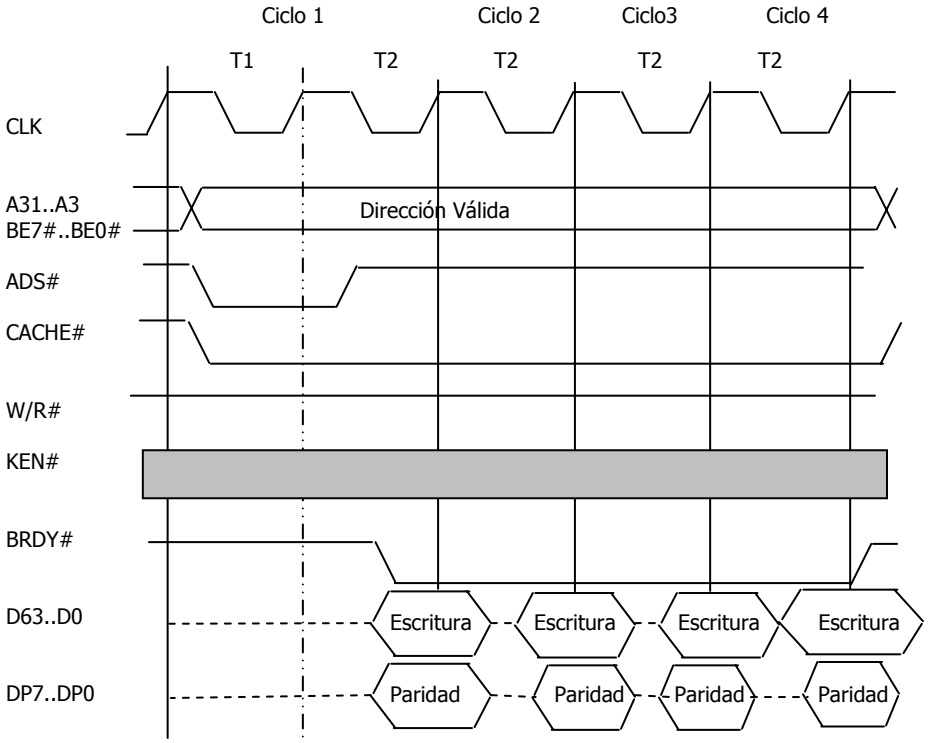


Figura 16.9 Ciclo de ecritura por ráfagas

Un **ciclo de escritura** en modo burst o por ráfagas es siempre un ciclo de recuperación de una línea modificada en la caché de datos. El Pentium indica esto con con una señal activa CACHE a nivel bajo.

Para la caché L2, que contiene datos e instrucciones, no se pueden producir recuperaciones de instrucciones, porque la información que almacena no debería ser nunca modificada. La caché no contiene instrucciones a diferencia de sus predecesoras.

Así, el ciclo de escritura burst siempre afecta a la caché de datos; los ciclos de lectura burst por otro lado, afectan tanto a la caché de datos como a la de código.

Las recuperaciones ocurren en los siguientes casos:

1º) El reemplazo de una línea de caché modificada por una línea de memoria (caché L2 o memoria principal); esto es resultado de un error de caché en la caché de datos.

2ª) En un ciclo de petición de acuerdo con el protocolo MESI ("Modified, Exclusive, Shared, Invalid" o "Modificado, Exclusivo, Compartido, Inválido"), se produce un acierto para una línea modificada en la caché de datos.

3º) Un ciclo snoop (rastreo) interno del Pentium ha encontrado una línea modificada en la caché de datos.

4ª) Una activación externa de la señal FLUSH# (en la caché).

5º) El Pentium ejecuta una instrucción WBINV (recuperación e invalidación).

Se emplean 4 ciclos de reloj para escribir 32 bytes.

En este modo, también se pueden insertar estados de espera si el subsistema de memoria no pueden continuar lo suficientemente rápido (ciclo de burst o por ráfagas lento). Esto ocurre, si el subsistema de memoria tarda en activar la señal BRDY#. Entonces el ciclo de bus se extiende con un ciclo de reloj T2.

16.6- CICLOS "INQUIRY" (PETICIÓN) Y "SNOOPING" (RASTREO INTERNO)

16.6.1- Ciclos "Inquiry"

Los ciclos "inquiry" son utilizados para el protocolo MESI para sistemas multiprocesador. De este modo es posible para una unidad externa determinar cuando los datos en una dirección específica está almacenada en la caché del Pentium.

Adicionalmente, la unidad externa puede invalidar los datos almacenados, así mismo la línea de caché correspondiente. Esto es necesario para que los datos en un sistema multiprocesador sean siempre consistentes. Si un procesador cambia un dato en su caché que por otra parte está además almacenado en la caché de otra CPU, y esta última también modifica el valor, no está claro del todo cual de los dos cambios es el correcto ni el actual.

Para un ciclo "inquiry" la unidad externa debe transferir la dirección física de memoria del dato al Pentium. Esto se consigue con la ayuda de la señal AHOLD. Después de que el sistema externo ha enviado una señal AHOLD activa con un nivel alto al Pentium y además ha esperado dos ciclos CLK, transfiere la dirección solicitada al Pentium a través de A31-A5,A4 y A3 son ignoradas porque una línea completa de caché es siempre direccionada en un ciclo "inquiry" (sólo se pueden leer y escribir unidades de líneas en las caché). La activación final de la señal EADS indica al Pentium que obtenga la dirección dada y que las use como una dirección "inquiry". Así, para ciclos "inquiry" EADS tiene el mismo propósito que ADS# para ciclos de bus normales. La señal INV debe también enviarse con las direcciones A31-A5.

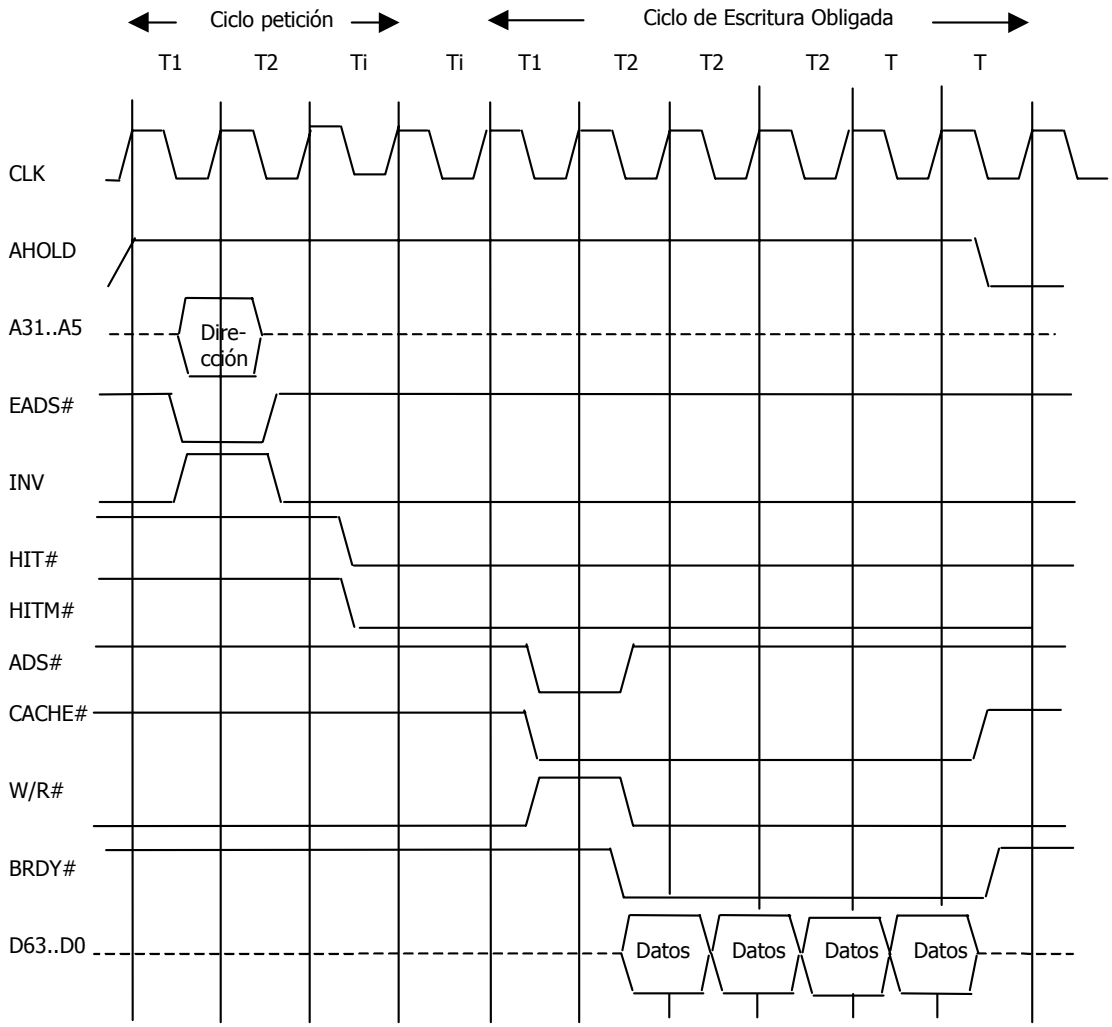


Figura 16.10. Ciclo Inquiry

En un ciclo "inquiry" iniciado externamente, como en un acceso normal a memoria, la lógica "tag" (etiqueta) de la caché chequea cuando la dirección está localizada en la memoria, y así cuando el valor correspondiente está disponible en la caché. En un ciclo "inquiry" la dirección se origina en la lógica externa; en un ciclo de memoria normal, por el contrario en la Kernel de la CPU. En ambos casos, el comparador de direcciones analiza el contenido de la caché y compara las direcciones transferidas direcciones transferidas para determinar un error o un acierto. Si se produce un acierto en la caché de datos o de código el comparador de direcciones, envía una señal activa de nivel bajo desde el pin HIT#, si esto no ocurre, se produce un error y la salida HIT envía una señal con nivel alto. El Pentium requiere dos ciclos de CLK después de la activación de EADS. Si se produce un acierto en una línea de caché de datos el Pentium envía una señal HITM (acierto modificado) con un nivel bajo. Esto es importante para la armonización en sistemas multiprocesador, en los cuales las CPUs individuales se comunican por medio de la memoria principal y además acceden a ella.

Si durante un "ciclo inquiry" ocurre un acierto en una línea de caché modificada (no hay líneas modificadas en la caché de código)entonces el Pentium recupera la línea en memoria. Así se envía un ciclo de recuperación"burst", de este modo se asegura que el sistema contiene los últimos datos.

El Pentium realiza el ciclo de recuperación "burst" con la activación de ADS# dos ciclos de reloj después de HIT# se haya enviado. En principio HIT# no cambia su valor entre los dos ciclos "inquiry"; así permanece en nivel bajo. HITM# permanece a nivel bajo hasta dos ciclos de reloj después de la última BRDY# el ciclo de recuperación "burst". Si AHOLD permanece activo al comienzo de una recuperación, y así el Pentium está prevenido de usar su bus de direcciones, entonces el subsistema de memoria utiliza las direcciones A31-A5. De este modo, A31-A5 siempre proporciona la dirección de comienzo de una línea de caché. El sistema externo también puede desactivar AHOLD antes de que el Pentium comience con la recuperación de la línea de caché. En este caso el Pentium envía la dirección de recuperación. Un ciclo "inquiry" puede suceder como máximo cada dos ciclos de reloj. Si se obtiene una línea de caché, entonces el P recupera los 32 bytes y sólo entonces acepta el siguiente ciclo "inquiry".

16.6.2- Ciclo de "snooping" o rastreo interno

Adicionalmente a esta iniciación externa de ciclo "inquiry" (por medio de EADS) existe también un ciclo snooping o de rastreo interno, usado principalmente para chequear la consistencia de los dos cachés independientes, la de instrucciones y la de datos. Los ciclos de rastreo pueden ser inicializados con o sin petición de invalidación. (INV=1 ó 0) En modo real la misma información puede estar disponible tanto en la caché de datos como en el código, como resultado indirecto de una preselección.

Puede producirse en tres casos:

- El Kernel de la CPU del Pentium accede a la caché de instrucciones, y este acceso resulta un error. Si la línea de caché con el valor direccionado está situada en la caché de datos pero la caché de instrucciones está cargada con la misma línea de memoria, es posible que las dos líneas tengan diferentes valores. La entrada en la caché de datos es modificada, pero no ha sido recuperada. En este caso, el ciclo snooping o de rastreo provoca que la línea de la caché de datos se recupere antes de cargar la caché de código. Si la línea está localizada en la caché de datos, pero está marcada como compartida o exclusiva, entonces sólo invalida la línea.

- El kernel de la CPU accede a la caché de datos y este acceso origina un error. Si la línea de caché correspondiente está también almacenada en la caché de instrucciones, es invalidada por el ciclo snooping o de rastreo. La razón de esto es que en el primer caso (acceso de lectura) una unidad externa puede haber cambiado la entrada en la memoria externa sin que el Pentium tenga conocimiento de esto. El llenado de la línea de caché siguiente al error, carga la versión correcta del valor en la caché de datos; la entrada correspondiente en la caché de código es invalidada. En el segundo caso (acceso de escritura) el Pentium cambiará el valor en la memoria externa con una probabilidad del 100%. De este modo, las entradas en la línea de caché correspondiente de la caché de código no son corregidos.

- Si el bit A o D (bit de accedido y de "dirty"(sucio), respectivamente) de una tabla de página o entrada de directorio de página es sobrescrita como el resultado de un acceso o un cambio de la página correspondiente, el Pentium realiza un ciclo snooping o de rastreo interno. Si la línea de caché correspondiente está disponible en una de las dos cachés y está marcada como válida. Entonces el Pentium invalida la entrada. Si la línea está disponible y además ha sido modificada, primero se recupera, y sólo entonces se invalida. Esto es necesario para permitir una administración centralizada de las páginas que afectan a la memoria principal compartida y que son además usadas por el procesador en el sistema. El almacenamiento externo de una página, posiblemente con un almacenamiento explícito a un sistema externo de almacenamiento masivo, es

decidido por la frecuencia y el tipo de accesos a la página. Se podría considerar la memoria principal como una 'caché' entre la CPU y el sistema de almacenamiento masivo.

ESTADO PRESENTE	PRÓXIMO ESTADO INV=1	PRÓXIMO ESTADO INV=0	DESCRIPCIÓN
M	I	S	El rastreo accede a una línea modificada indicada por los pines HIT# y HITM# a nivel bajo.
E	I	S	El rastreo es indicado por el pin HIT# a nivel bajo; no por el ciclo de bus generado.
S	I	S	El rastreo es indicado por el pin HIT# a nivel bajo; no por el ciclo de bus generado
I	I	I	No hay dirección en la caché:El pin HIT# está a nivel alto.

Tabla 16.3 Transiciones de estado de la caché durante los ciclos de rastreo

16.7- ESPACIO DE DIRECCIONES DE E/S Y PERIFÉRICOS

Adicionalmente al área de memoria, el Pentium tiene también el llamado espacio de memoria E/S, que puede ser accedido mediante las instrucciones IN/OUT y por medio de los puertos. Este espacio de memoria, al igual que sus predecesores, incluye puertos de 8 bits (64K), 16 bits (32K) y 32 bits (16K) o una equivalencia mixta de todos ellos (sólo es posible combinar puertos cuya dirección de memoria sea contigua). Las direcciones asignadas para la memoria E/S son desde la 0000H hasta la FFFFH, estando reservadas las direcciones comprendidas entre la 0F8H hasta la 0FFH Así el Pentium tiene dos espacios de memoria totalmente separados (memoria y espacio de E/S), ambos direccionados por medio de los buses de datos y direcciones.

Los accesos a direcciones de E/S se permiten con un máximo de 32 bits; por tanto los restantes 32 bits del bus de datos del Pentium quedan sin uso.

El Pentium identifica un ciclo de bus de E/S por medio de una señal de nivel bajo M/IO#. Como en otros procesadores 80x86 los puertos 0f8h a 0ffh están reservados. Estos puertos reservados son utilizados para las comunicaciones entre la CPU y el coprocesador.

16.7.1- Direccionamiento de E/S

Existen dos modos de acceso a los puertos E/S:

1.- Usando las instrucciones E/S
2.- Usando las instrucciones de propósito general, lo cual requiere de mapear (copiar) la memoria E/S

El uso de instrucciones E/S es recomendable porque acceden directamente a la memoria E/S y así se asegura que la instrucción se ejecuta antes de que el siguiente flujo de instrucciones sea ejecutado.

16.7.1.1- Acceso directo a E/S

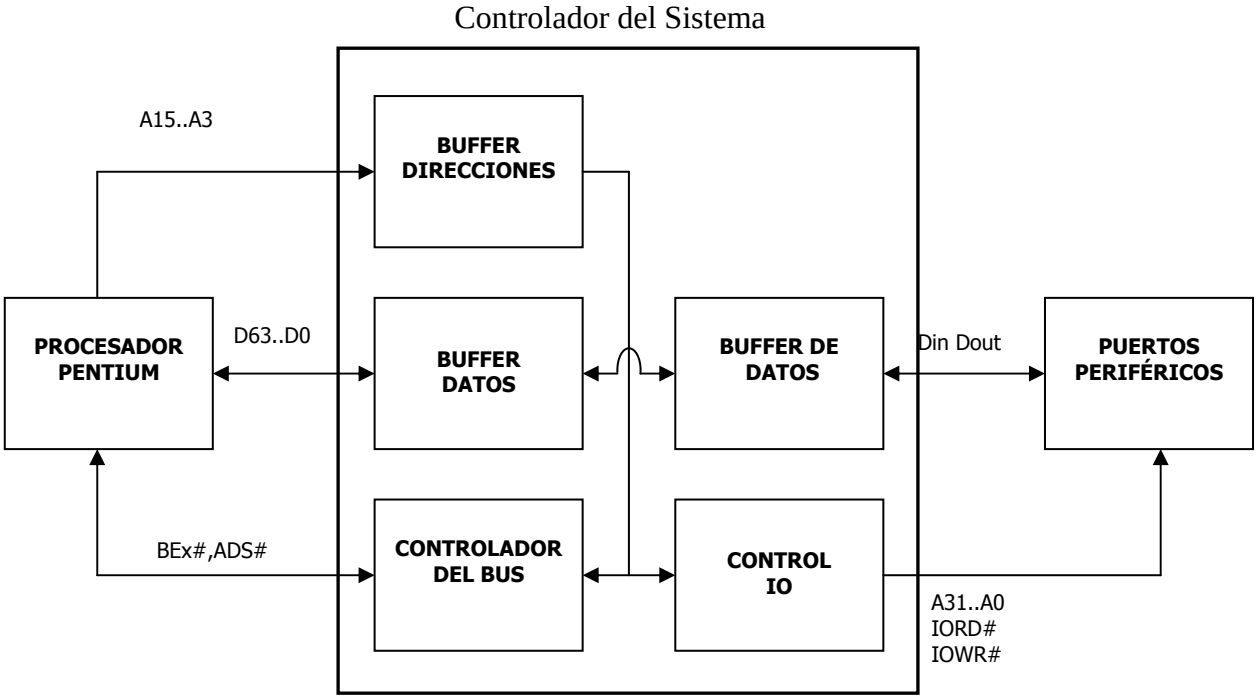


Figura 16.8 Direccionamiento E/S mediante instrucciones específicas de forma directa

Si el Pentium desea acceder a un puerto, entonces produce una señal de salida de nivel bajo M/IO#. El controlador del sistema reconoce que se debe ejecutar un acceso al espacio de direcciones de E/S. Adicionalmente, por medio de WR# se determina la dirección de la transferencia como Pentium $\rightarrow$ puerto (W/R# = 1) ó puerto $\rightarrow$ Pentium (W/R# = 0). La escritura o lectura de datos en un puerto se realiza de la misma manera que si se tratara de memoria

Como en lectura y escritura de la memoria principal, la CPU proporciona una dirección al buffer de direcciones y controla el controlador del bus usando las señales de estado M/IO#, D/C#, y W/R#. El controlador lógico del bus reconoce a partir de la señal de control M/IO# que se va a realizar un acceso a las direcciones de E/S y no a memoria. Así, por medio del controlador de memoria, se activa el controlador de E/S, que decodifica la dirección de la señal del buffer de direcciones y direcciona el puerto adecuado. Así los datos pueden ser transferidos entre el procesador y un registro en el espacio de memoria de E/S del PC. El Pentium 4 puede direccionar un máximo de 64K puertos de 8 bits, donde las 16 líneas mas significativas A31-A16 están siempre a nivel bajo para el acceso a un puerto.

16.7.1.2- Acceso mediante memoria E/S mapeada

Se puede usar todas las instrucciones que permita acceder a memoria (MOV etc).

No es deseable cachear la memoria E/S mapeada. En el caso del Pentium 4, se utiliza los registros de memoria MTRH, que no están presentes en anteriores versiones del pentium ni en el i486.

16.8- LOS BUFFERS DEL BUS INTERNO

16.8.1- Buffers de escritura

El interface de bus del Pentium contiene dos buffers de escritura de 64 bits; cada buffer es asignado a uno o dos pipelines (cauces) U o V. Deberían prevenir el frenado o detección de los pipelines del buffer si el bus externo no está disponible inmediatamente. Si se ha producido un llenado de la caché y el bus externo está ocupado, sin los buffers de escritura, los pipelines podrían tener que detenerse por un mínimo de ciclos de reloj (el mínimo para un llenado en la línea caché en modo burst).

El kernel de la CPU del Pentium puede escribir en ambos buffers simultáneamente.

Todos los accesos de escritura que están esperando a ejecutarse en los buffers de escritura deben ser complementados antes de que se produzca un acceso posterior de lectura.

Sólo los accesos de escritura a memoria son almacenados temporalmente en los buffers. Las escrituras en el espacio de memoria de E/S por medio de los buffers de escritura son directamente realizados por medio del bus externo, pero sólo después de que la espera de los accesos de escritura en los buffers se haya complementado.

16.8.2- Buffers write-back ("escritura obligada")

Adicionalmente a los buffers de escritura de 64 bits, el Pentium contiene tres buffers de 'write-back' de 32 bytes. Cada uno de esos buffers incluye una línea completa de la caché y soporta al Pentium durante la recuperación de líneas de caché en la caché de datos.

El **primer buffer** almacena la línea de caché a recuperar, si durante el reemplazo de una línea de caché modificada debe ser recuperada en la memoria externa.

El **segundo buffer** se provee para el ciclo “snooping” (rastreo) externo, y almacena la línea de caché a ser recuperada, si el ciclo ha alcanzado una entrada modificada en la caché de datos.

El **tercer buffer** soporta la recuperación de una línea de caché si un proceso interno de 'snooping' (rastreo) alcanza una entrada modificada en la caché de datos.

Los ‘snoops’ o rastreos internos y externos son usados para la implementación del protocolo MESI y para asegurar la consistencia de la caché en un sistema multiprocesador con múltiples cachés.

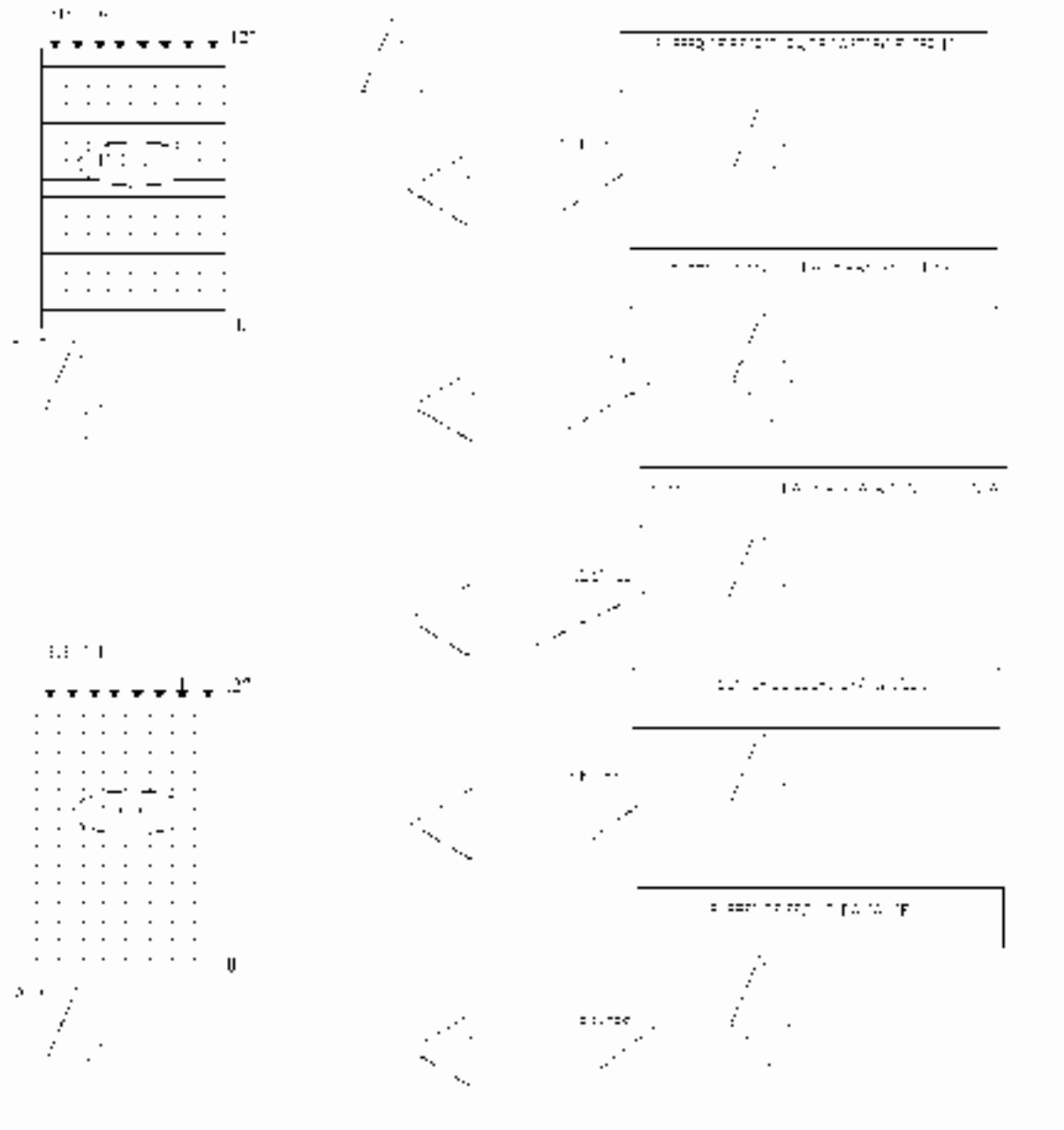


Figura 16.11 Buffers de Escritura

16.8.3- Buffers de llenado de línea

Adicionalmente a los buffers de escritura descritos previamente, el Pentium contiene dos buffers adicionales de llenado de línea que soportan llenados de líneas de caché. Cada uno de ellos tiene una anchura de 32 bytes (una línea de caché).

El primer buffer se asigna a la caché de datos, y el segundo a la caché de instrucciones.

Si el Pentium lee una línea completa de caché durante un ciclo burst o por ráfagas en la caché de datos o de instrucciones, dicha línea no se introduce en la caché. En lugar de esto, se almacenan en el buffer en fragmentos de 8 bytes, y cuando se tiene la línea completa, ésta es transferida a la caché de datos o de código.

Un llenado de línea de caché suele ser consecuencia de un error en el acceso a la caché. Así el Pentium obtiene los datos loo más rápido posible. Los bytes requeridos son transferidos desde el subsistema de memoria durante el primer ciclo burst y son inmediatamente pasados al Kernel de la CPU. Así el Kernel de la CPU los usa para la ejecución de la instrucción antes de que el ciclo de lectura 'burst' haya sido completado y la línea del buffer haya sido llenada.

Esto también se aplica cuando durante un error de caché, una línea de caché modificada es cambiada por otra y almacenada externamente. Sólo después de que la línea de buffer ha sido llenada, la línea modificada a salvar es transferida al buffer de recuperación para la línea de caché reemplazada, y la nueva línea de caché se carga desde el buffer de llenado de líneas. Esto es necesario porque el llenado de la línea de caché para la nueva línea de caché y también la recuperación de la línea de caché modificada, debe ser ejecutada por ciclos 'burst', que no deben ser interrumpidos.

REPERTORIO DE INSTRUCCIONES 17

17.1- TIPO DE DATOS

El Pentium soporta todos los tipos de datos que manejan los lenguajes evolucionados, aunque hay cuatro fundamentales:

1º) *byte* u octeto de ocho bits.

2º) *Palabra,* compuesta por dos *bytes*.

3º) *Doble palabra* de cuatro *bytes*.

4º) *Cuádruple palabra* de ocho *bytes*.

Cuando la memoria está organizada con tamaño byte, tanto las palabras como las dobles palabras, ocupan dos o cuatro posiciones sucesivas, respectivamente. Sin embargo, la dirección de inicio de palabras y dobles palabras puede ser cualquiera y no se precisa (como sucede en otros procesadores) que estén *alineadas,* ocupando direcciones pares y múltiplos de cuatro.

Esta característica mejora la flexibilidad en la ubicación de datos y libera al programador de vigilar el direccionado, aunque se recomienda alinear dichos tipos de datos para que su acceso sea más rápido. Como se aprecia en la figura 17.1, los bytes de más peso ocupan las direcciones más altas en la memoria.

Además de los tres tipos de datos básicos, el Pentium admite los tipos que se muestran en la figura 17.2 y que se pueden clasificar, en función de las instrucciones que las manejan, de la siguiente forma:

1º) Enteros y ordinales

Son los números con y sin signo, respectivamente, que son reconocidos por todas las instrucciones aritméticas. Tienen tamaño byte, palabra y doble palabra.

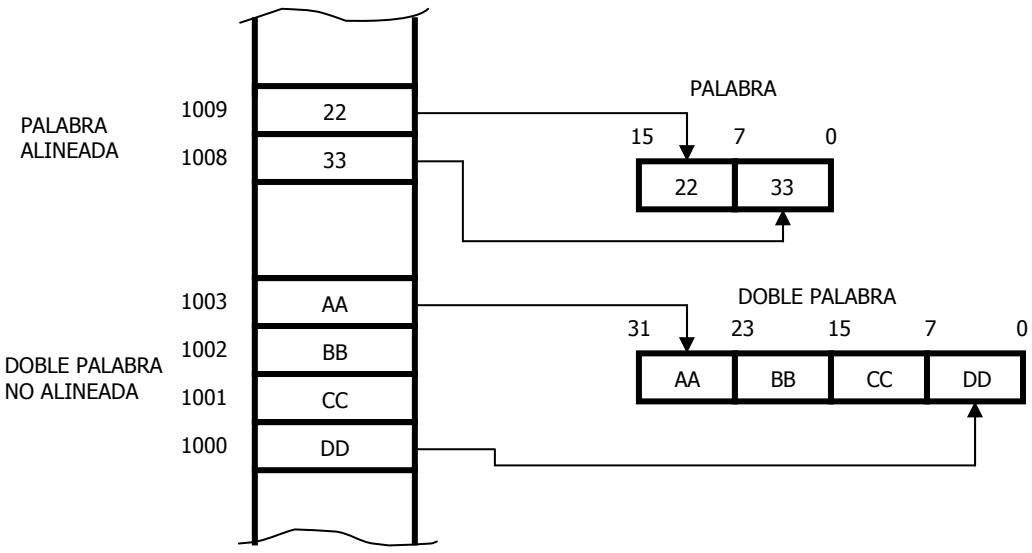


Figura 17.1. Distribución de palabras y doble palabras en una memoria organizada con tamaño byte.

2º) Números BCD (Decimal Codificado en Binario)

Es la representación de los números decimales mediante cuatro bits. Este tipo de datos lo soportan las instrucciones de ajuste, que complementan a las instrucciones aritméticas.

Los datos BCD pueden estar desempaquetados o empaquetados. En el primer caso, en cada byte hay un dígito BCD en los cuatro bits de menos peso, mientras que en los empaquetados se representan dos dígitos BCD en un byte.

3º) Cadenas

Hay instrucciones especializadas para manipular cadenas de bits, de bytes, de palabras y de dobles palabras. Las cadenas pueden alcanzar un tamaño de 4
GB.

4º) Campo de bits

Este formato permite a ciertas instrucciones manejar ciertos bits particulares en un campo, cuyo tamaño máximo puede alcanzar los 32 bits.

Las instrucciones capaces de trabajar con campos de bits y cadenas son muy interesantes en aplicaciones gráficas y de comunicaciones.

5º) Punteros de direcciones

Este tipo de datos está referenciado en los modos de direccionamiento de los operandos y en las instrucciones de salto.

El *puntero corto* o cercano, dispone de un desplazamiento de 32 bits que permite un salto dentro del segmento que se trabaja.

El *puntero largo* o lejano, además del desplazamiento, contiene el valor de un selector para uno de los seis registros de segmento. Se puede realizar un salto a otro segmento. El tamaño de este puntero es de 48 bits.

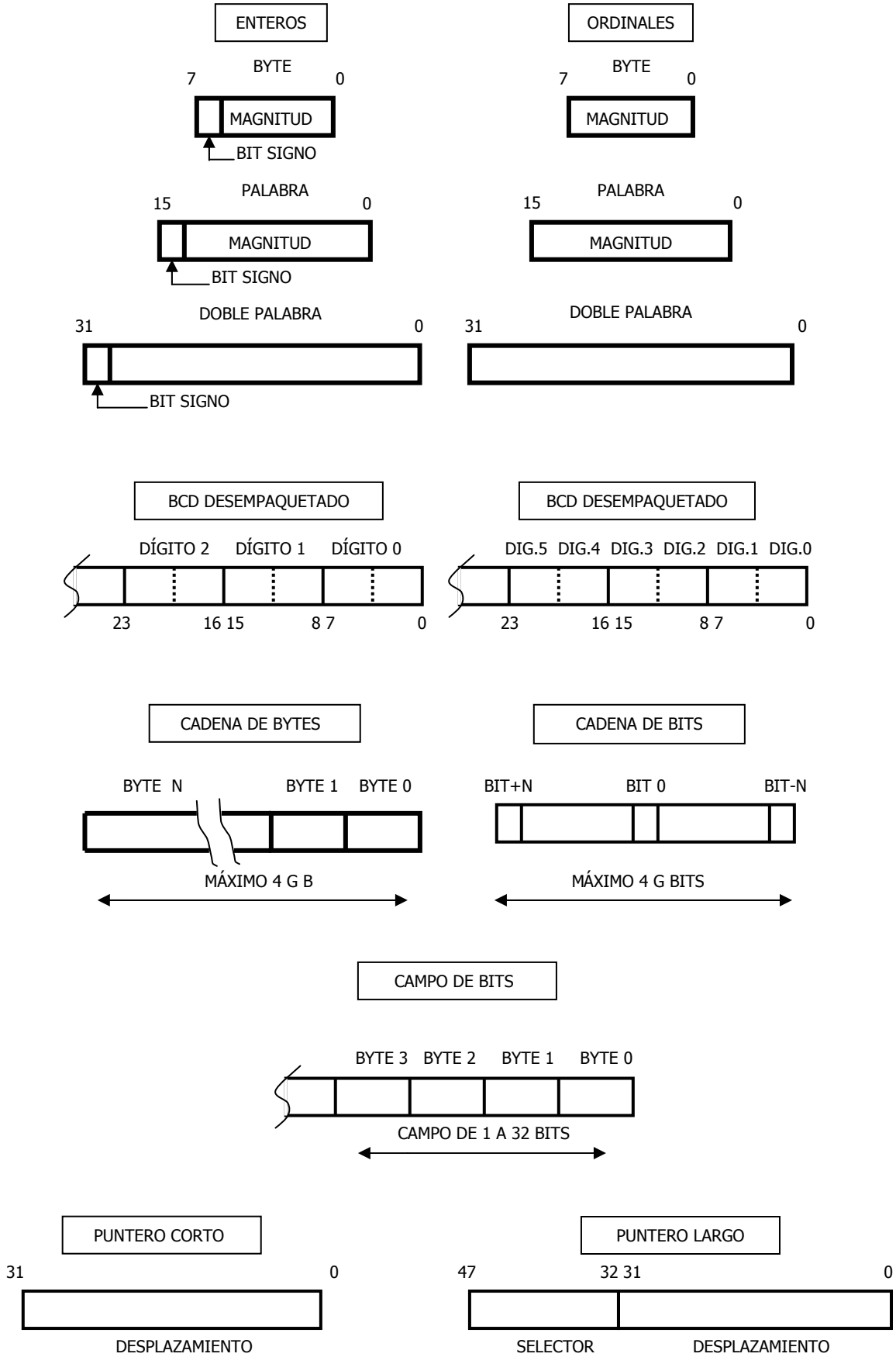


Figura 17.2. Tipos de datos que utiliza el Pentium.

17.1.1-TIPOS DE DATOS DEL COPROCESADOR MATEMÁTICO

El coprocesador matemático soporta siete tipos de datos diferentes, aunque internamente guarda toda la información en el formato de coma flotante y precisión extendida, es decir, en una representación que ocupa 80 bits, igual a la del tamaño de sus registros internos.

Como la memoria de datos tiene almacenados los siete tipos de datos en cualquier formato, son las instrucciones las que se encargan de hacer la conversión en los dos sentidos. Si una instrucción toma un dato de la memoria, lo convierte en formato de coma flotante con precisión extendida de 80 bits, al introducirlo dentro del coprocesador y viceversa.

Los siete tipos de datos se clasifican en tres grandes grupos:

1º. Enteros
En este grupo hay tres tipos: largo, corto y palabra, de 64, 32 y 16 bits, respectivamente.

2º. Decimal empaquetado
Guarda 18 dígitos BCD empaquetados, en formato de 80 bits.

3º.Coma flotante
En este apartado se distinguen los datos en simple precisión, en doble precisión y de precisión extendida, que es el formato con el que opera internamente el coprocesador.

En la figura 16.3 se muestran los sirte tipos de datos para el coprocesador matemático.

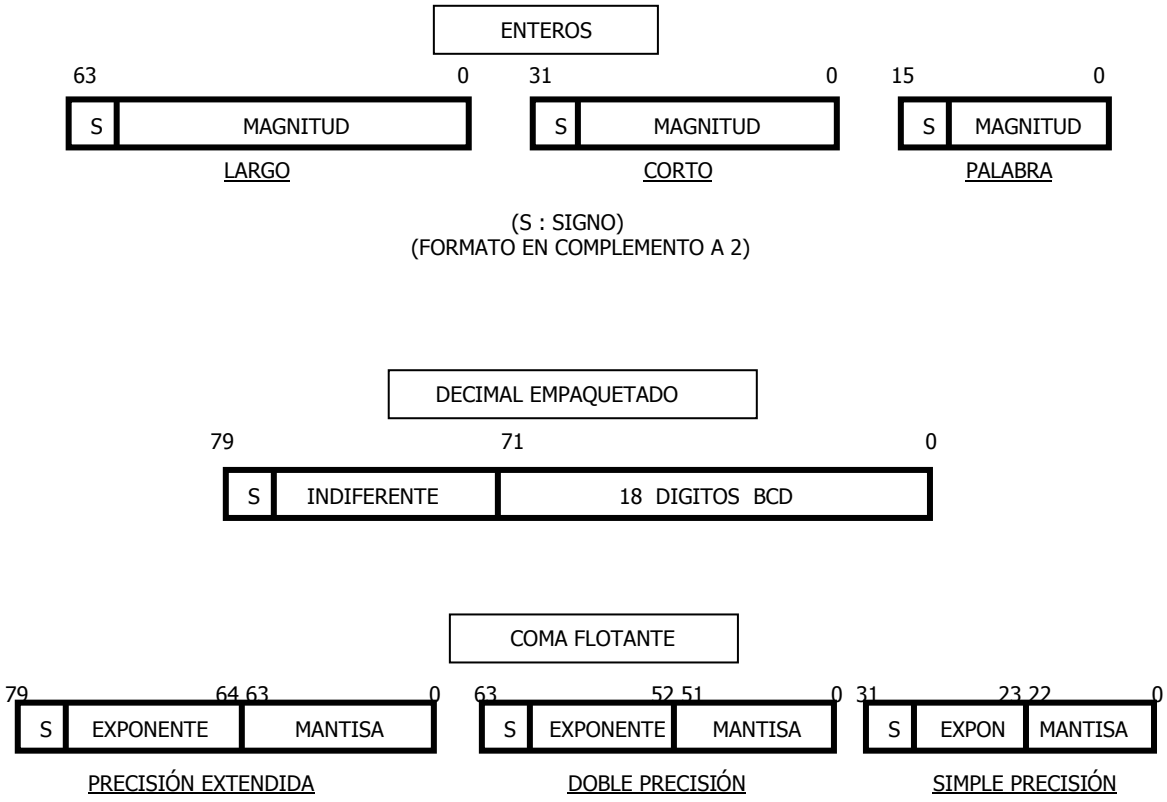


Figura 17.3. Descripción de los siete tipos de datos que admite el coprocesador matemático

17.2- MODOS DE DIRECCIONAMIENTO

Las formas que dispone el Pentium para referenciar los operandos de sus instrucciones, se pueden dividir en tres grandes categorías:

1º. Modo de direccionamiento inmediato

El operando reside en la propia instrucción.

Ejemplo

IMUL ECX, -4

Multiplica el entero contenido en el registro ECX por el operando inmediato, situado en la propia instrucción, que es -4.

2ª. Por registro

El operando reside en un registro interno del procesador.

Ejemplo

INC EBX

Incrementa el operando, que es el valor contenido en el registro EBX.

3ª.Modo de direccionado en memoria

El operando reside en la memoria y admite múltiples variantes para expresar la posición donde se encuentra.

El algoritmo fundamental que aplica el Pentium para calcular la dirección efectiva, cuando trabaja con operandos de 32 bits, es el siguiente:

Dirección = *Variable* + *Reg. Base* + *Reg. índice* X *Factor de Escala* + *Desplazamiento*

Los diferentes campos del algoritmo pueden tomar un valor de acuerdo con el registro interno de la CPU que tenga capacidad para almacenarlo, según el siguiente criterio:

Reg. Base ={ EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI }

Reg. Índice = {EAX, EBX, ECX, EDX, EBP, ESI, EDI }

Factor de Escala = {1, 2, 4, 8 }

A)El primer término posible que admite el algoritmo se denomina *variable y* permite precisar un identificador o etiqueta que representa el desplazamiento del principio de una variable, como puede serlo una tabla.

B)El *registro Base* apunta al comienzo de una estructura de datos de dirección variable, por ejemplo, el comienzo de una matriz de registros, que no tiene una dirección fija en la memoria.

C)El *registro Índice* mueve el apuntador dentro de las estructuras de datos para acceder a los diferentes campos de que constan. El *factor de escala* tiene en cuenta el tamaño del elemento (octeto, palabra, etc.).

D)Finalmente, el *desplazamiento* de valor constante se usa para seleccionar un dato de dirección conocida, o bien, la base de una estructura de datos que responde a una dirección fija.

Este completo y potente sistema de direccionamiento, proporciona al Pentium un acceso cómodo a gran variedad de datos en la memoria, que son tan frecuentes en los lenguajes de alto nivel, como el C, o el FORTRAN.

Aunque se ha indicado que todos los registros generales pueden actuar como Base y como Índice, hay que matizar, que, como sucedía en el 8086, cada uno está dedicado preferentemente a ciertas misiones. Por ejemplo, el registro EAX es el más solicitado para la implementación de instrucciones de tipo aritmético. Los registros ESI y EDI se emplean comúnmente como registros Índice. También es habitual encontrarse con instrucciones que seleccionan implícitamente uno de los registros, como es el caso de las instrucciones con repetición de bucles (LOOP), en el que como registro contador se usa ECX. No obstante, el Pentium admite una eficaz flexibilidad de empleo de los registros.

Ejemplo:

ADD EAX, TABLA[EBP + ESP4 +8]

Se deposita en EAX el resultado de la suma del contenido de EAX con el dato de 32 bits ubicado en la posición de memoria de TABLA, a la que se suma EBP, ESI*4 y 8.

El algoritmo empleado para calcular la dirección efectiva cuando se trabaja con operandos de 16 bits, es el siguiente:

DIRECCION = VARIABLE + REG. BASE + REG. INDICE + DESPLAZAMIENTO

El registro Base puede ser *BX* o *BP* y el registro Índice el *SI* o el *DI*.

Con este modo de direccionamiento se trabaja sobre segmentos de 64 KB de tamaño máximo.

El manejo de las puertas de E/S se realiza mediante instrucciones especializadas, dedicadas a esta finalidad. En el Pentium se dedica un tamaño de 64 KB no segmentado al espacio de E/S.

Las instrucciones que introducen o sacan datos por las puertas de E/S, disponen de un operando que especifica la posición de éstas en el espacio de E/S, bien directamente, si el valor de la puerta es inferior a *255,* o bien colocando en DX su valor, cualquiera que sea el mismo.

17.3-CLASIFICACIÓN Y CARACTERÍSTICAS GENERALES DEL REPERTORIO DE INSTRUCCIONES

En este apartado se describen de forma muy sucinta todas las instrucciones del Pentium, clasificándolas según la función que realizan, para dar una idea de todas las existentes y su misión principal.

Para resaltar la aportación del Pentium, se separan las instrucciones que se han incorporado en este procesador de las convencionales conocidas en modelos precedentes. Junto con las nuevas instrucciones, otras ventajas inherentes al Pentium son:

La posibilidad de tratar operandos de 32 bits, también implica una mayor rapidez de ejecución y también los nuevos tipos de datos y posibilidad de combinar varios de ellos en ciertas instrucciones.

El siguiente capítulo está destinado a proporcionar un resumen detallado de todas las instrucciones del Pentium, ordenadas alfabéticamente e incluyendo todas las características de interés para el programador que las va a aplicar.

17.3.1-INSTRUCCIONES PRIVILEGIADAS

Cuando trabaja el Pentium en Modo Protegido se debe asegurar la protección de la memoria y de las tareas para hacer inaccesible el paso de una LDT a otra, así como la modificación no regulada de los registros del sistema (GDTR, LDTR, TR, IDTR, etc).

La mayoría de las instrucciones del Pentium se pueden ejecutar en cualquier nivel de privilegio, pero hay unas pocas que pueden comprometer la integridad del sistema cuyo uso está restringido. Se clasifican en dos grandes grupos:

INSTRUCCIONES PROTEGIDAS

Se pueden ejecutar únicamente desde niveles de privilegio que sean jerárquicamente superiores al campo de Nivel de privilegio de E/S (IOPL) del registro EFLAGS.

INSTRUCCIONES PRIVILEGIADAS

Sólo pueden ejecutarse en el máximo nivel de privilegio, o sea, con segmentos que tengan CPL = 0.

Dentro de las instrucciones privilegiadas, que sólo se usan en el 5.0. y nunca en las aplicaciones, se distinguen cuatro grupos:

1º)Cualquier instrucción que pueda modificar el campo IOPL, como IRET, POPF y las relacionadas con la conmutación de tarea (TS).

2º)Instrucciones que afectan a los registros que hacen referencia a tablas que controlan el sistema en Modo Protegido.

3º) Instrucciones que afecten a la MSW, que es la palabra baja del CRO.

4º) Instrucción HLT.

Dentro del segundo grupo, se comentan algunas peculiaridades de las instrucciones que lo componen.

LGDT – LIDT

Sirven para cargar los registros GDTR e IDTR, respectivamente. Tanto GDTR como IDTR, sólo se deben cargar en la inicialización del sistema.

SGDT – SIDT

Almacenan los contenidos de GDTR e IDTR, respectivamente.

Las cuatro instrucciones de carga y almacenamiento de los registros GDTR e IDTR, hacen uso de operandos de memoria de ocho bytes de longitud.

LLDT- LTR

Cargan a LDT y a IR, respectivamente. El operando que manejan es el selector de un descriptor apropiado.

LDTR y IR deben cargarse de forma explícita durante la inicialización del sistema.

SLDT - STR

Almacenan el valor de LDTR y TR, respectivamente y pueden ejecutarse en cualquier nivel de privilegio.

Las tres instrucciones que afectan a MSW, son:

LMSW

Carga en la MSW el valor del operando. Instrucción privilegiada.

SMSW

Almacena el valor de MSW, pudiéndose ejecutar desde cualquier nivel de privilegio.

CLTS

Borra o pone a 0 el señalizador TS de tarea conmutada. Instrucción privilegiada.

Finalmente, la instrucción *HLT* también es privilegiada, pues pasa al procesador a un estado de parada bloqueando la ejecución de instrucciones. Del estado de parada se sale mediante un RESET o una interrupción.

17.3.2- INSTRUCCIONES PROTEGIDAS

Desde las aplicaciones hay que acceder a las puertas de E/S para utilizar los periféricos, pero el acceso ha de estar bajo control del sistema.

En Modo Protegido, el sistema controla el campo de dos bits IOPL del registro EFLAGS, que determina a partir de qué nivel de privilegio se puede acceder a las E/S. Será necesario que el CPL del segmento que intenta usar E/S sea igual o mayor que IOPL para poder ejecutar instrucciones de E/S.

Recuérdese que cada tarea disponía de otro mecanismo para restringir el uso de las E/S. Se trataba del mapa de bits de permiso de E/S, que ocupaba una zona del segmento TSS y que, poniendo a 1 ó a 0 los bits del mapa, prohibían o permitían el acceso *;* cada una de las puertas de E/S.

Mediante el control del campo IOPL (con la instrucción privilegiada POPF) el sistema puede establecer dinámicamente a cada tarea, un control riguroso a las E/S. Así, cuando el sistema pasa a realizar una tarea y coloca el campo IOPL 1, sólo se podrán ejecutar instrucciones de E/S desde segmentos de código que tengan un CPL cuyo valor sea 0 ó 1.

Las **instrucciones protegidas** son:

IN, OUT, INS y OUTS

Corresponden a instrucciones de E/S de operandos y cadenas, respectivamente.

SEI y CLI

Permiso y prohibición de las interrupciones enmascarables.

17.4- INSTRUCCIONES ARITMÉTICAS

AAA

Después de realizarse una suma, ajusta el último byte del resultado al formato BCD.

AAD

Ajusta el dividendo que va a participar en una operación de división para que el resultado, después de efectuada la división, lo proporcione en formato cociente-resto.

AAM

Ajusta el resultado de una operación de dos números BCD a dicho formato.

AAS

Tras realizar una resta, ajusta el último byte del resultado a BCD.

DAA

Tras sumar dos números BCD, dejando el resultado en AL, transforma dicho resultado en dos dígitos BCD.

DAS

Tras efectuar una resta de dos números BCD, dejando el resultado en AL, transforma dicho resultado en dos dígitos BCD.

ADD

Realiza la suma de dos operandos

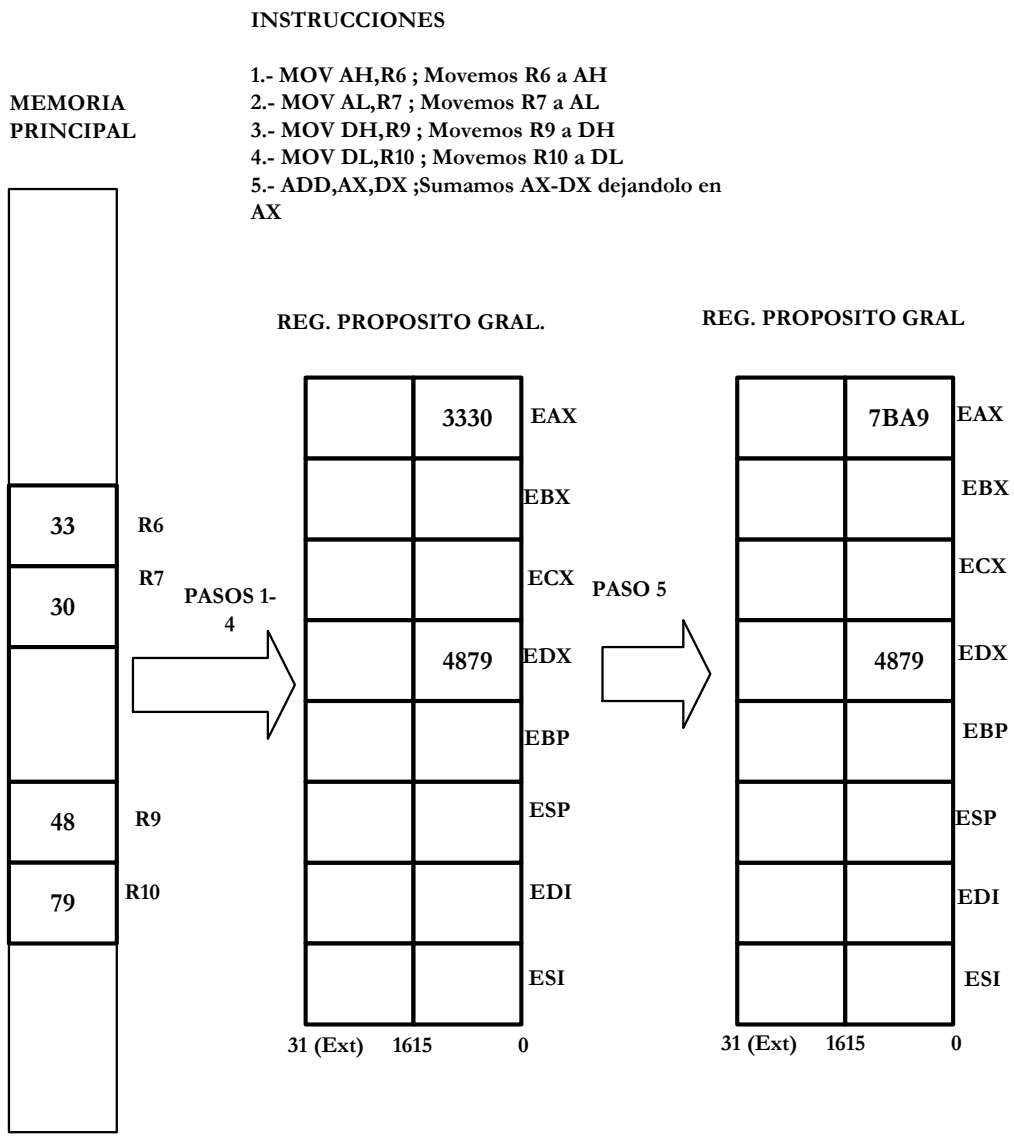


Figura 17.4. Resultado de la operación suma.

ADC

Suma dos operandos y el acarreo**.**

SUB

Efectúa la resta de dos operandos:

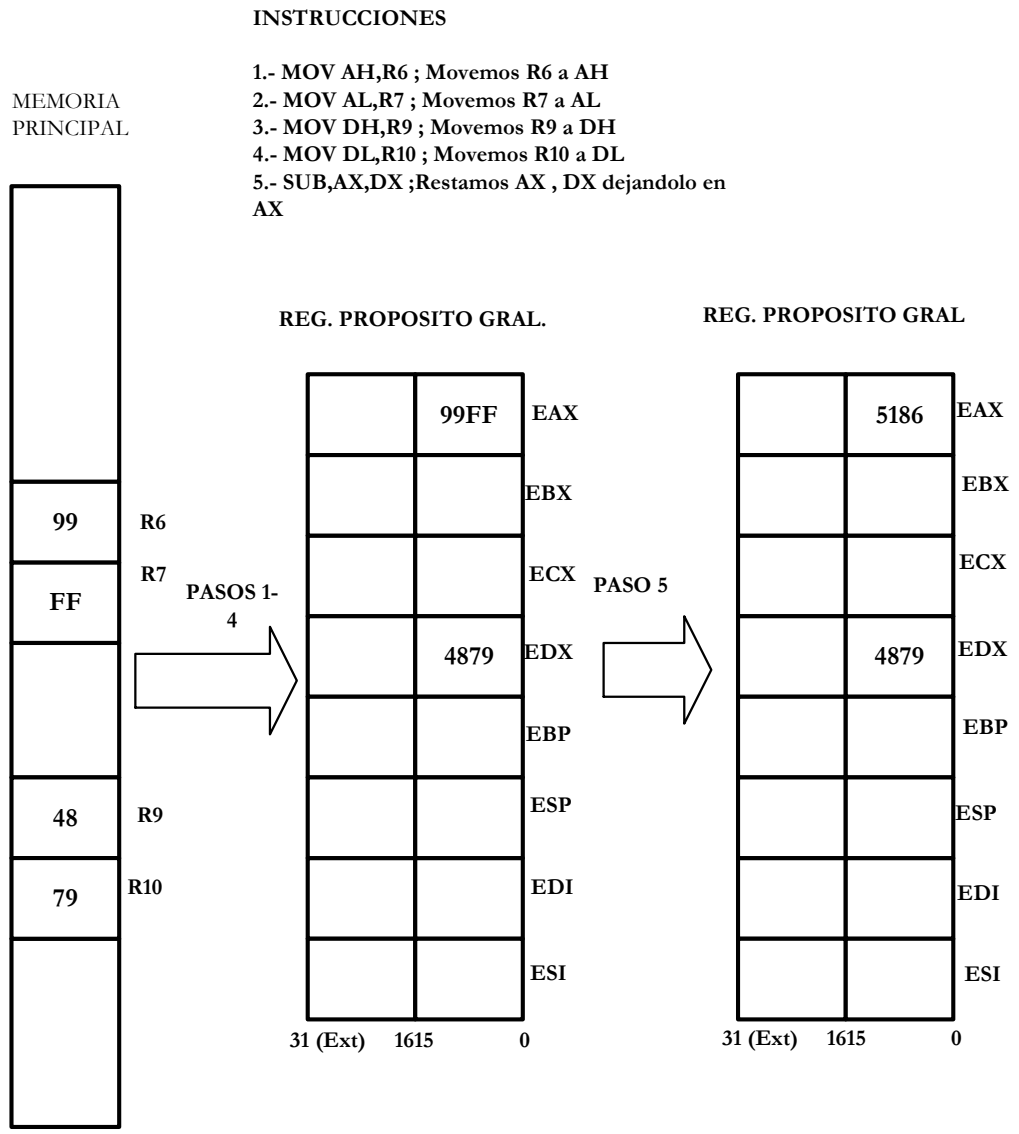


Figura 17.5. Resultado de la operación resta.

SBB

Resta el minuendo del sustraendo más el acarreo o llevada**.**

DEC

Decrementa el operando.

INC

Incrementa el operando.

MUL

Multiplica el operando, sin signo, por AL, AX ó EAX, según el tamaño del operando, dejando el resultado en AX, EAX ó EDX:EAX, respectivamente.

IMUL

Multiplicación de operandos con signo.

DIV

Divide a AL, AX o EAX por el operando especificado en la instrucción, dejando el resultado en dos registros, en forma de Resto:Cociente.

IDIV

División con signo, semejante a la anterior.

CBW/CWD

Convierten AL y AX a doble tamaño, respectivamente.

NEG

Realiza el complemento a dos del operando.

CMP

Compara dos operandos, reflejando el resultado únicamente en los señalizadores.

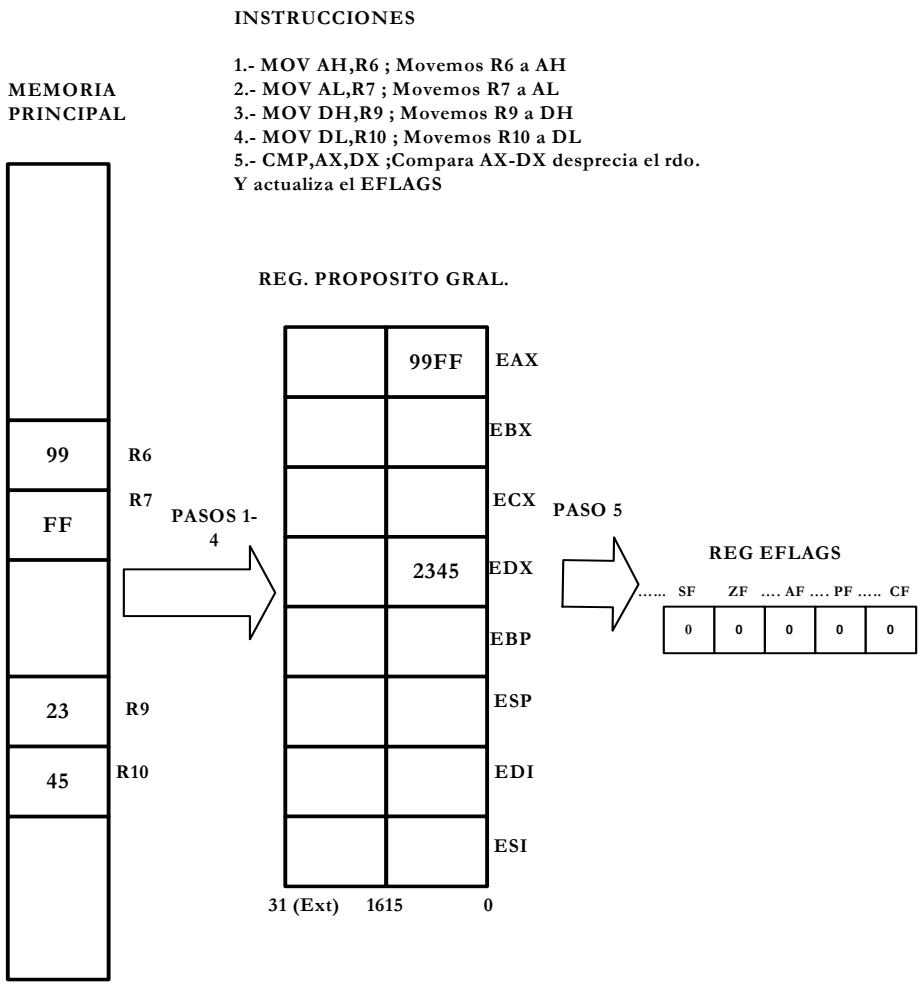


Figura 17.6. Resultado de comparar dos posiciones de memoria.

17.4.1- NUEVAS INSTRUCCIONES ARITMÉTICAS DEL PENTIUM.

CWDE

Convierte una palabra a doble palabra.

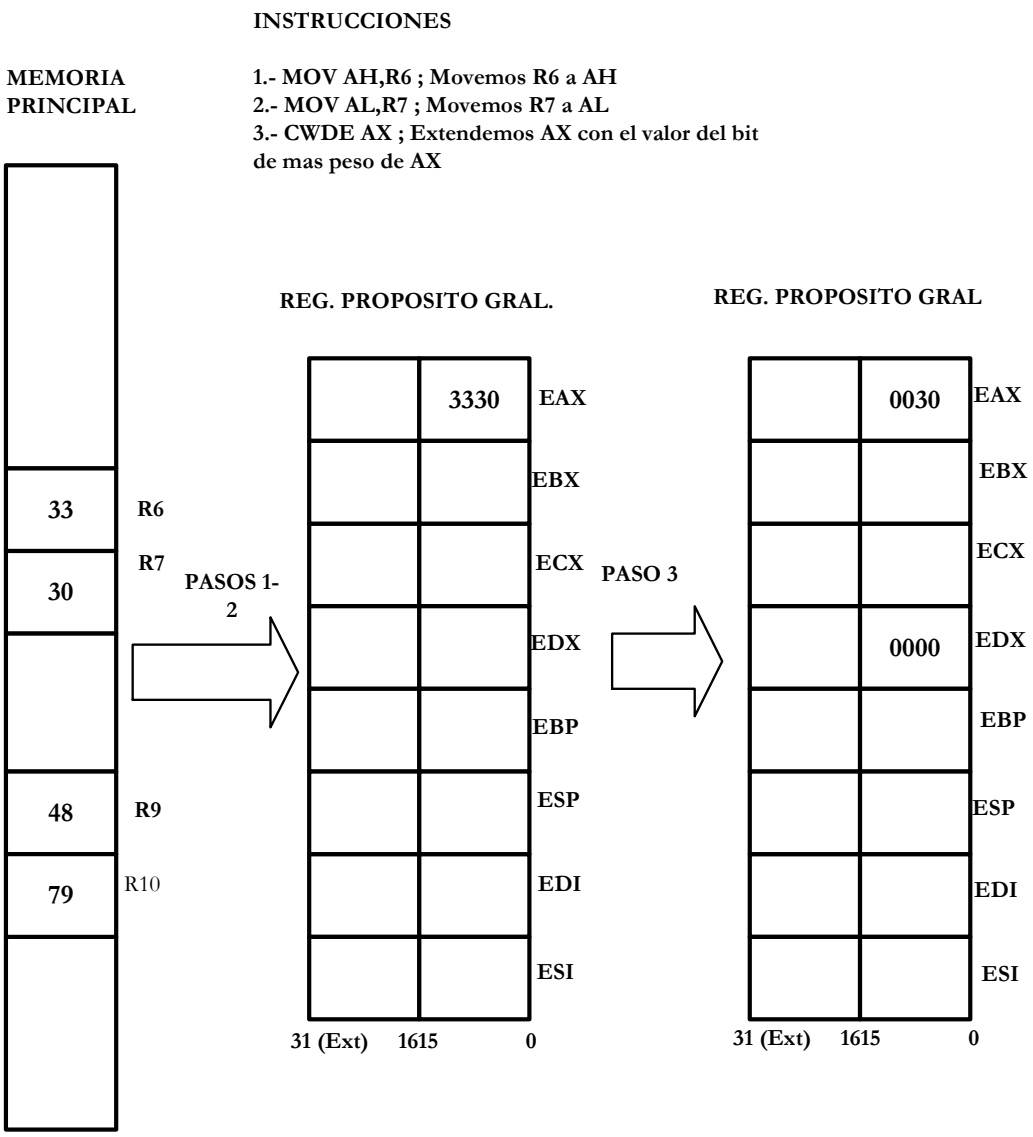


Figura 17.7. Convierte una palabra en doble palabra

CDQ

Convierte una doble palabra en cuádruple.

17.5- INSTRUCCIONES LÓGICAS

AND-OR-XOR-NOT

Realizan las operaciones lógicas AND, OR, EOR y NOT, respectivamente.

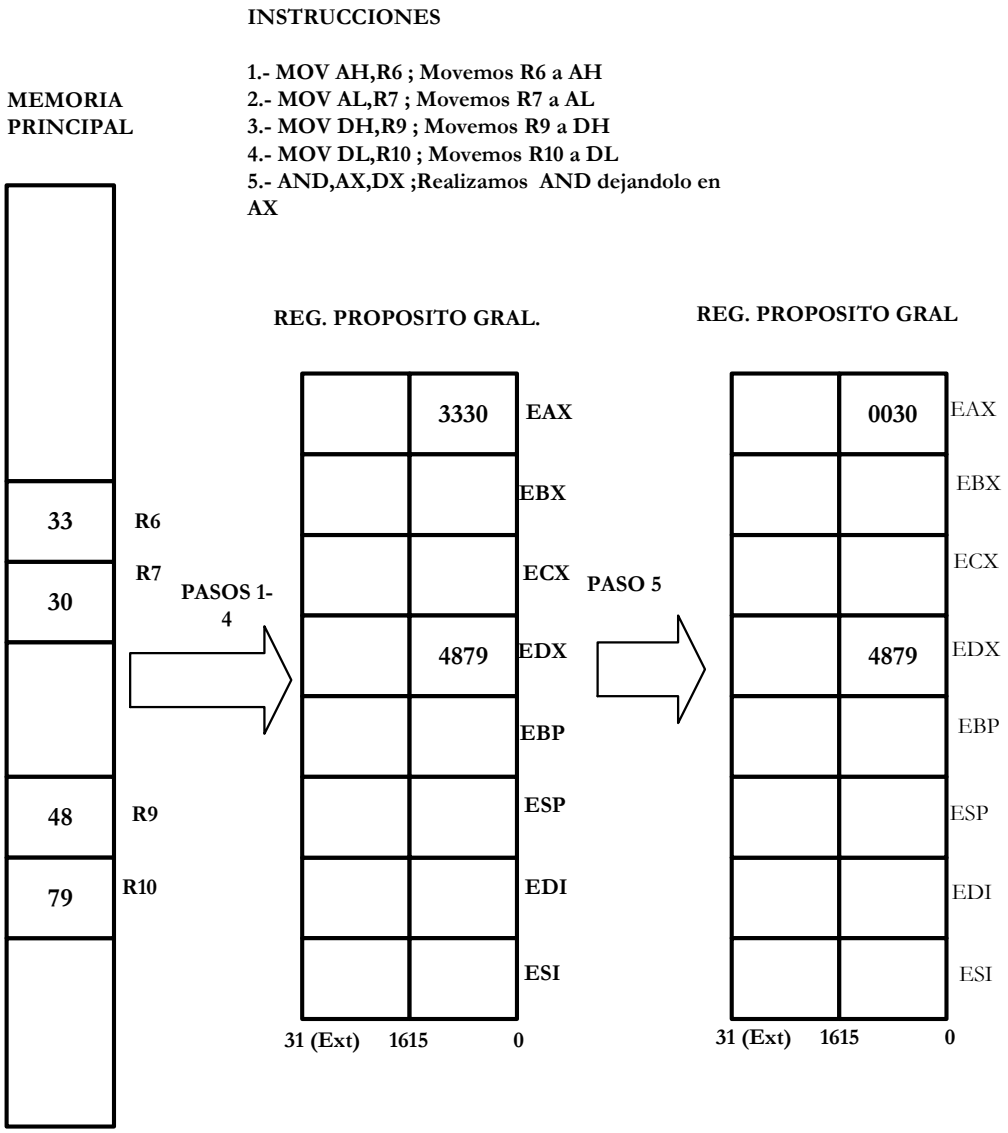


Figura 17.8. Resultado de la operación AND

ROL/ROR

Rotación a la izquierda o derecha de los bits del primer operando, tantas veces como lo indique CL u otro operando.

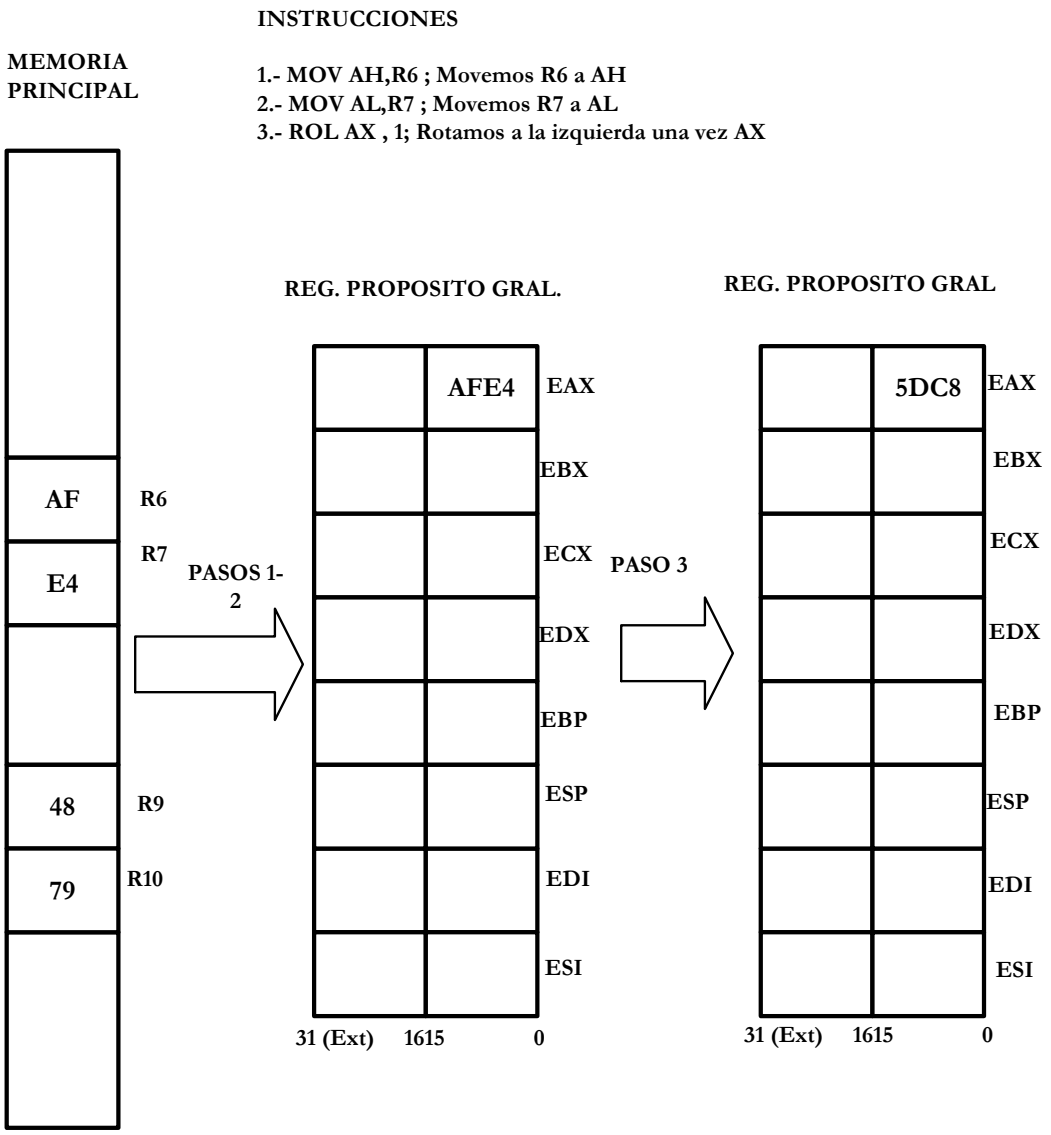


Figura 17.9. Resultado de rotar a la izquierda el operando

RCL/RCR

Rotación a izquierda o derecha, respectivamente, de los bits del primer operando, junto con el acarreo, el número de veces especificado por CL.

TEST

Realiza la operación lógica AND de los operandos, sin resultado. Sólo afecta a los señalizadores.

SAL/SAR

Desplazamiento aritmético a izquierda o derecha.

SHL/SHR

Desplazamiento lógico (sin preservar el valor del bit de signo) a izquierda o derecha.

17.5.1- Nuevas instrucciones lógicas del Pentium

SHLD

Desplaza el contenido del primer y segundo operando, conjuntamente, a la izquierda, el número de veces especificado en el tercer, operando.

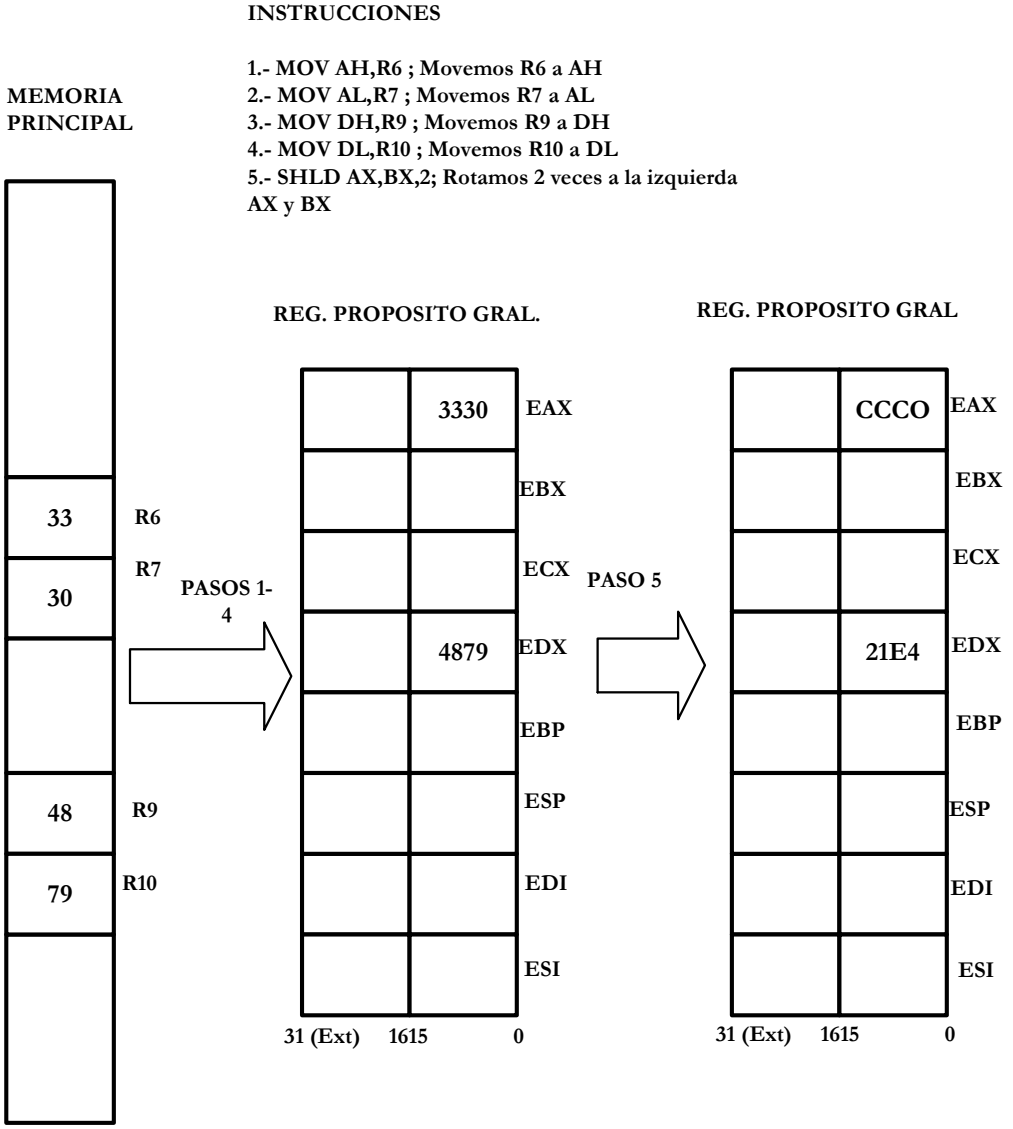


Figura 17.10. Resultado de la operación de rotar dos veces a la izquierda dos operandos distintos

SHRD

Igual formato y comportamiento que SHRD, pero realizando el desplazamiento a la derecha.

17.6- INSTRUCCIONES DE CADENAS

CMPS - CMPSB - CMPSW

Sirven para comparar cadenas, restando el contenido del elemento apuntado por ES:EDI, con el de DS:ESI, afectando sólo a los señalizadores.

INSTRUCCIONES

1.- CMPS ;Compara ESI de DS con EDI de CS.
Desprecia el rdo. Y actualiza el EFLAGS

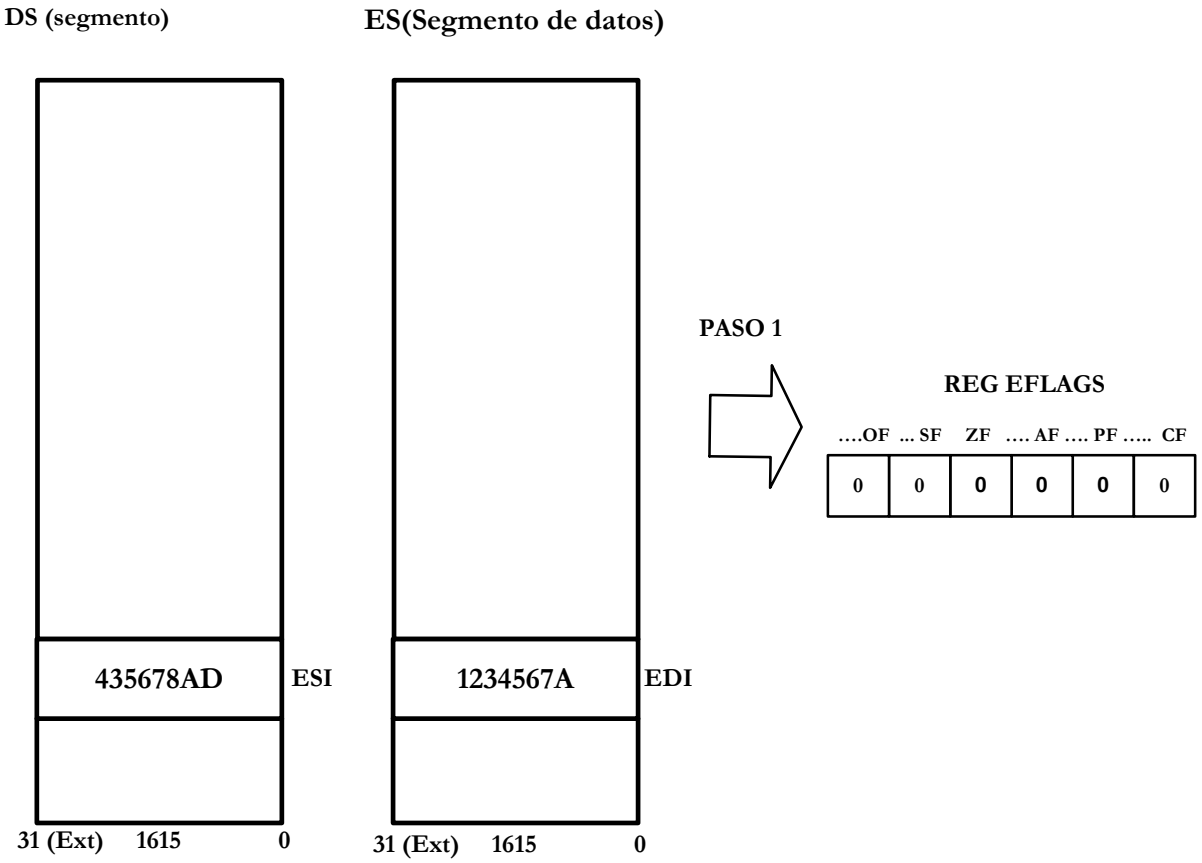


Figura 17.11. Resultado de comparar el contenido de dos segmentos y la representación de la actualización de EFLAGS

MOVS

Mueve el contenido de la dirección apuntada por DS:ESI a la referenciada por el puntero ES:EDI, incrementando el valor de los registros índice, según el tamaño en bytes de los operandos.

LODS -ODSB - LODSW

Transfiere al Acumulador el elemento apuntado por DS:ESI.

STOS - STOSB - STOSW

Transfiere el contenido del Acumulador a la dirección apuntada por ES:EDI.

SCAS - SCASB - SCASW

Resta el contenido direccionado por ES:EDI al Acumulador, afectando exclusivamente a los señalizadores.

INS - INSB - INSW

Carga en la dirección apuntada por ES:EDI el valor de la puerta de E/S apuntada por DX.

OUTS - OUTSB - OUTSW

Saca por la puerta apuntada por DX, uno o dos bytes a partir de la dirección señalada por ES:EDI.

REP

Es un prefijo que se antepone a ciertas instrucciones de cadenas. Repite la ejecución de la instrucción a la que antecede tantas veces como indica el valor del contenido de ECX.

REPE - REPZ

Se repite la instrucción a la que antecede hasta que ECX valga 0, o bien, el señalizador Z = 0.

REPNE - REPNZ

Es igual que la anterior, excepto que termina la repetición cuando Z = 1, o bien, ECX = 0.

XLAT-XLATB

Depositan en AL el contenido de la dirección obtenida al sumar EBX + AL. Sirve principalmente para la creación de ficheros

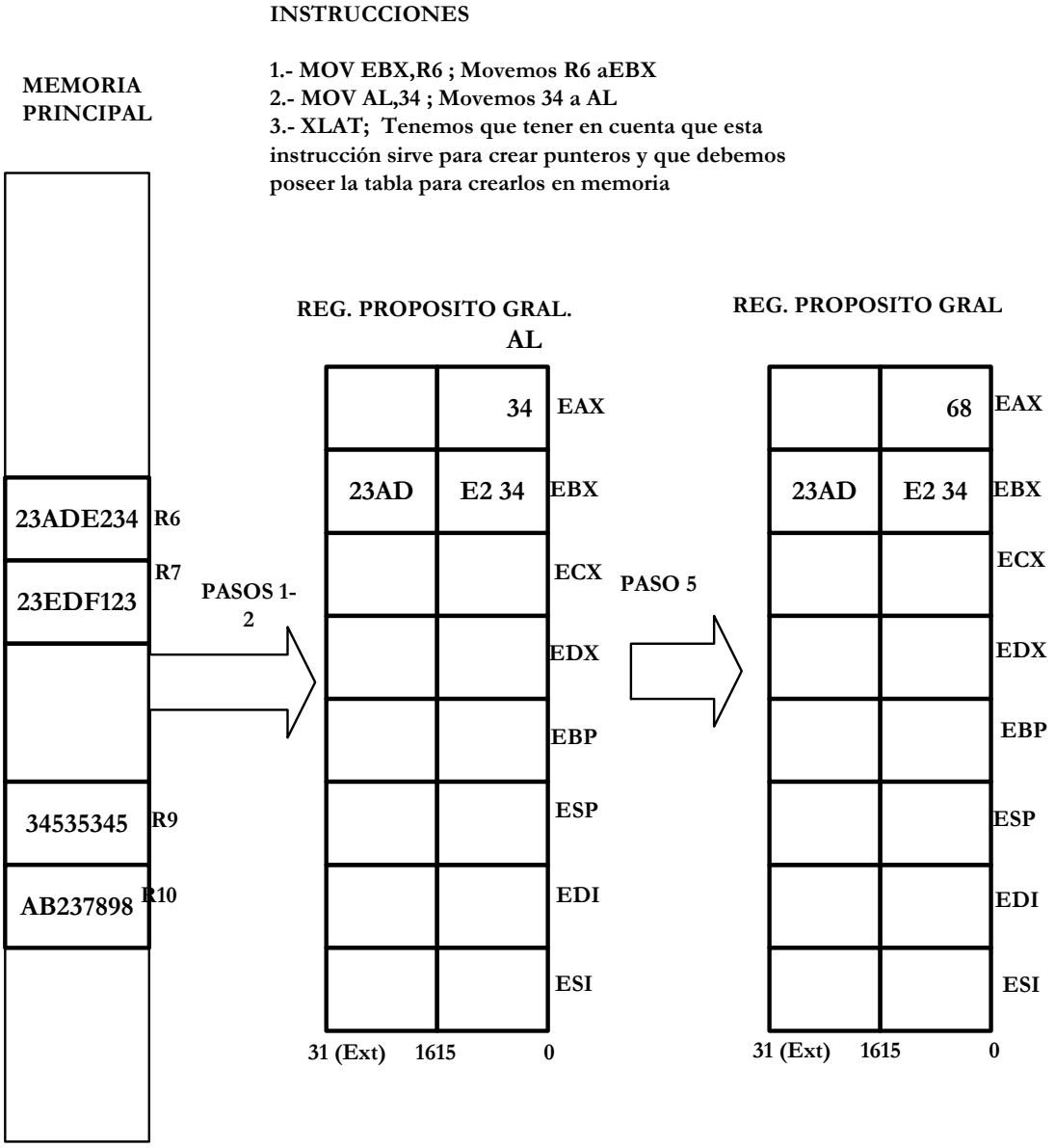


Figura 17.12.Representación de la creación de punteros en memoria mediante XLAT.

17.7- INSTRUCCIONES DE TRANSFERENCIA DE CONTROL

JMP

Realiza un salto a la dirección que se indica en el descriptor referenciado.

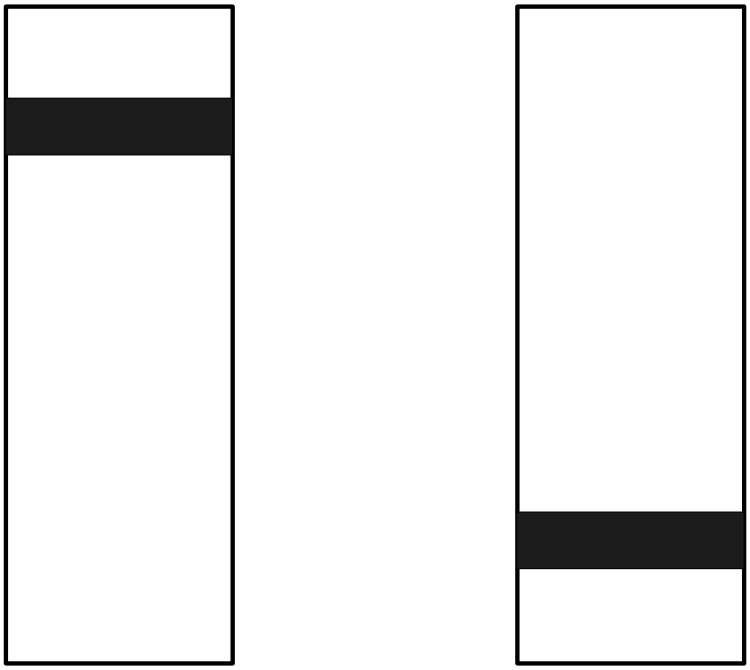


Figura 17.13.Representación de un salto incondicional dentro de un segmento de código.

LOOP

Provoca un salto a una etiqueta corta (situada entre -128 y + 127 posiciones).

LOOPZ - LOOPE - LOOPNZ - LOOPNE

Salto corto mientras ECX no sea 0 y Z =0 ó 1, según sean las dos primeras o las dos últimas.

CALL

Llamada o salto a una rutina.

RET

Retorno de una rutina.

17.7.1- INSTRUCCIONES DE TRANSFERENCIA DE CONTROL ESPECIALES

*J*cc son instrucciones de salto condicional en las que CC representa el código de condición y son las siguientes.

JA

Salta si el primer operando es mayor que el Segundo. Sin signo

JAE

Salta si el primer operando es menor que el Segundo. Sin signo

JB

Salta si el primer operando es menor igual que el Segundo. Sin signo

JBE

Salta si el primer operando es mayor igual que el Segundo. Sin signo

JE

Salta si el bit Z del EFLAGS es igual a 1

JO

Salta si el bit de OVERFLOW del EFLAGS es igual a 1.

JNO

Salta si el bit de OVERFLOW del EFLAGS es igual a 0.

JCXZ

Salta si el registro de CX es distinto a 0.

JS

Salta si el bit S del EFLAGS es igual a 1.

JNS

Salta si el bit S del EFLAGS es igual a 0.

JNA

Salta si el primer operando es mayor que el Segundo. Con signo

JNAE

Salta si el primer operando es menor que el Segundo. Con signo

JNB

Salta si el primer operando es menor igual que el Segundo. Con signo

JNBE

Salta si el primer operando es mayor igual que el Segundo. Con signo

JC

Salta si el bit de CARRY del EFLAGS es igual a 1

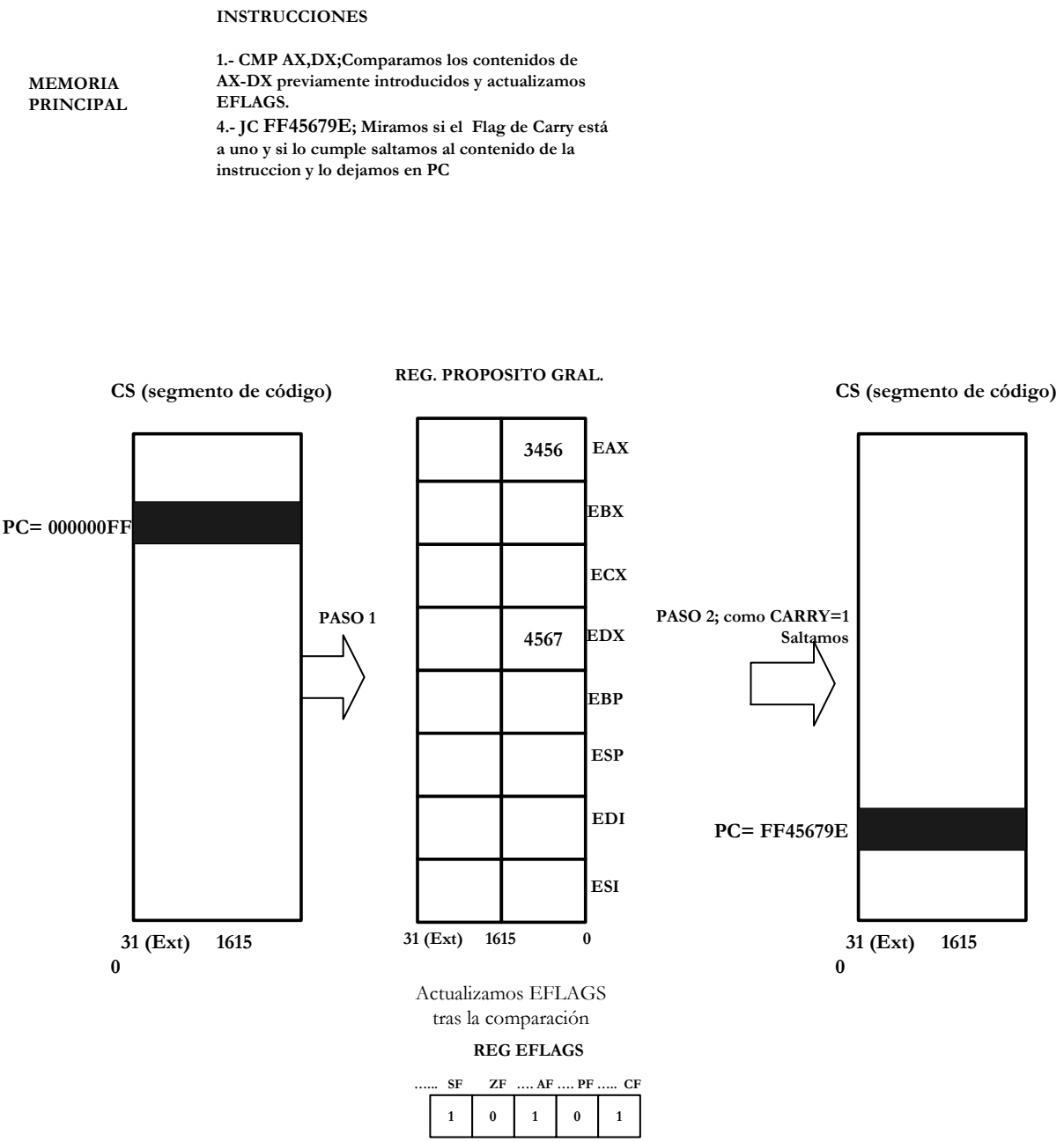


Figura 17.14.Representación de un salto condicional (CARRY=1?) dentro de un segmento de código.

JNC

Salta si el bit de CARRY del EFLAGS es igual a 1

JP

Salta si la paridad es impar o no-paridad

JNP

Salta si la paridad es par o hay paridad

17.8- INSTRUCCIONES DE TRANSFERENCIA DE DATOS

IN - OUT

Entrada y salida de información desde las puertas de E/S.

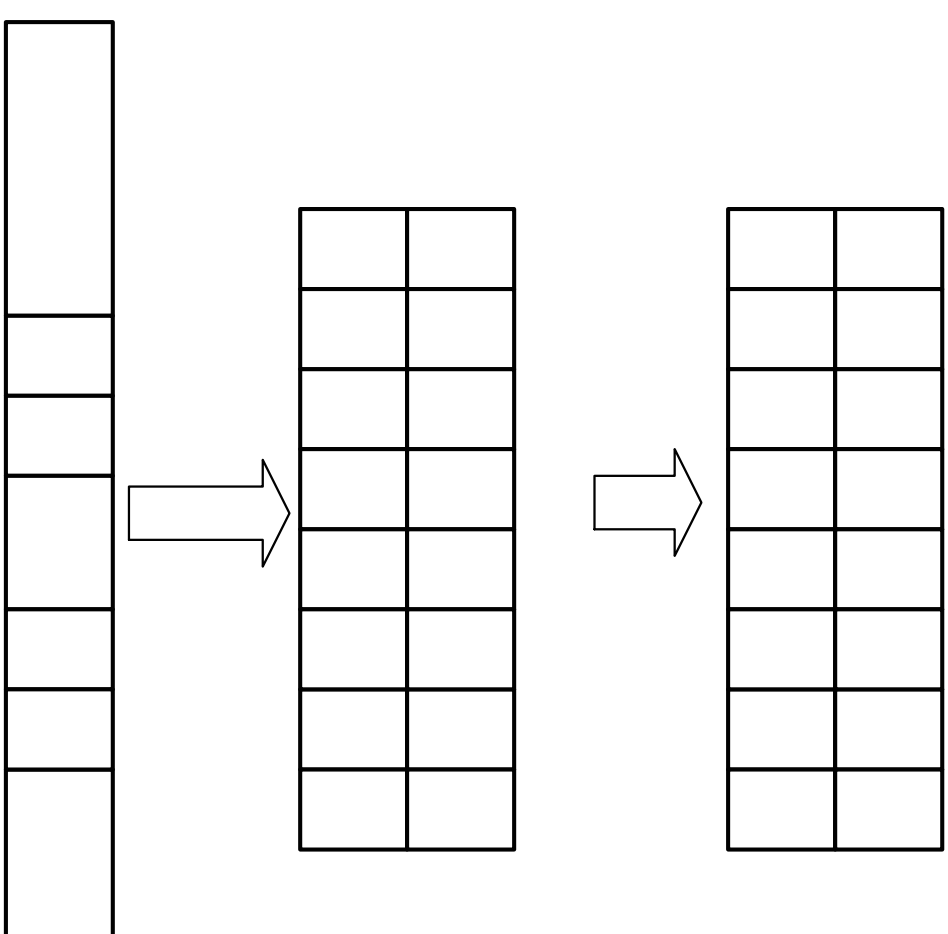


Figura 17.15. Introducimos un valor de una posición de memoria en AL mediante la instrucción IN

POP

Se transfiere desde la cima de la pila una información a un registro.

MEMORIA
PRINCIPAL

POPA

Transfiere desde la pila ocho palabras que se cargan en los ocho registros generales.

POPF

Transfiere desde la cima de la pila una palabra al registro de señalizadores.

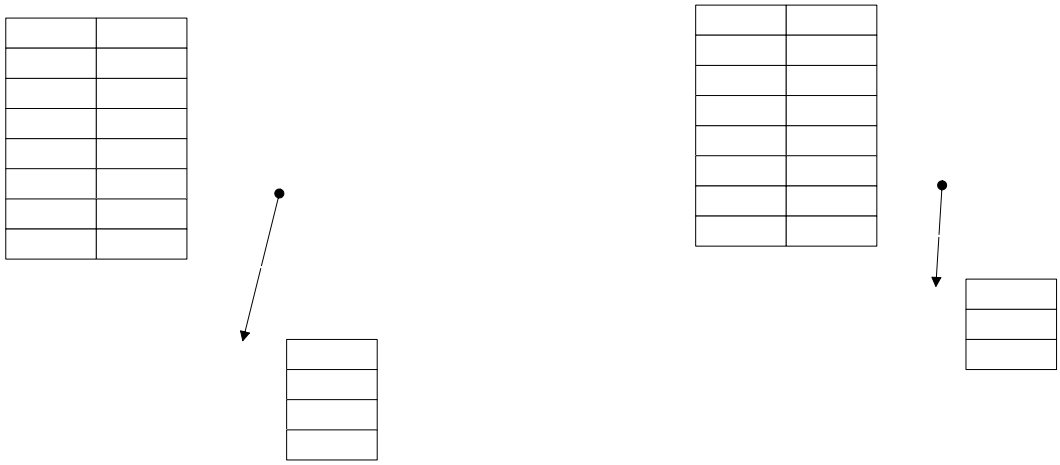


Figura 17.16. Representación del antes y el después de realizar la instrucción POPF.

PUSH

Transfiere un registro a la pila.

PUSHA

Transfiere a la cima de la pila ocho registros generales de 16 bits (AX, BX, CX, DX, SP, BP, SI y DI).

PUSHF

Transfiere a la pila el registro de los señalizadores.

MOV

Transfiere al primer operando el valor del segundo.

REG. PROPOSITO GRAL.

EAX

EBX

ECX

EDX

EBP

3569 ESP(Cima de la

EDI

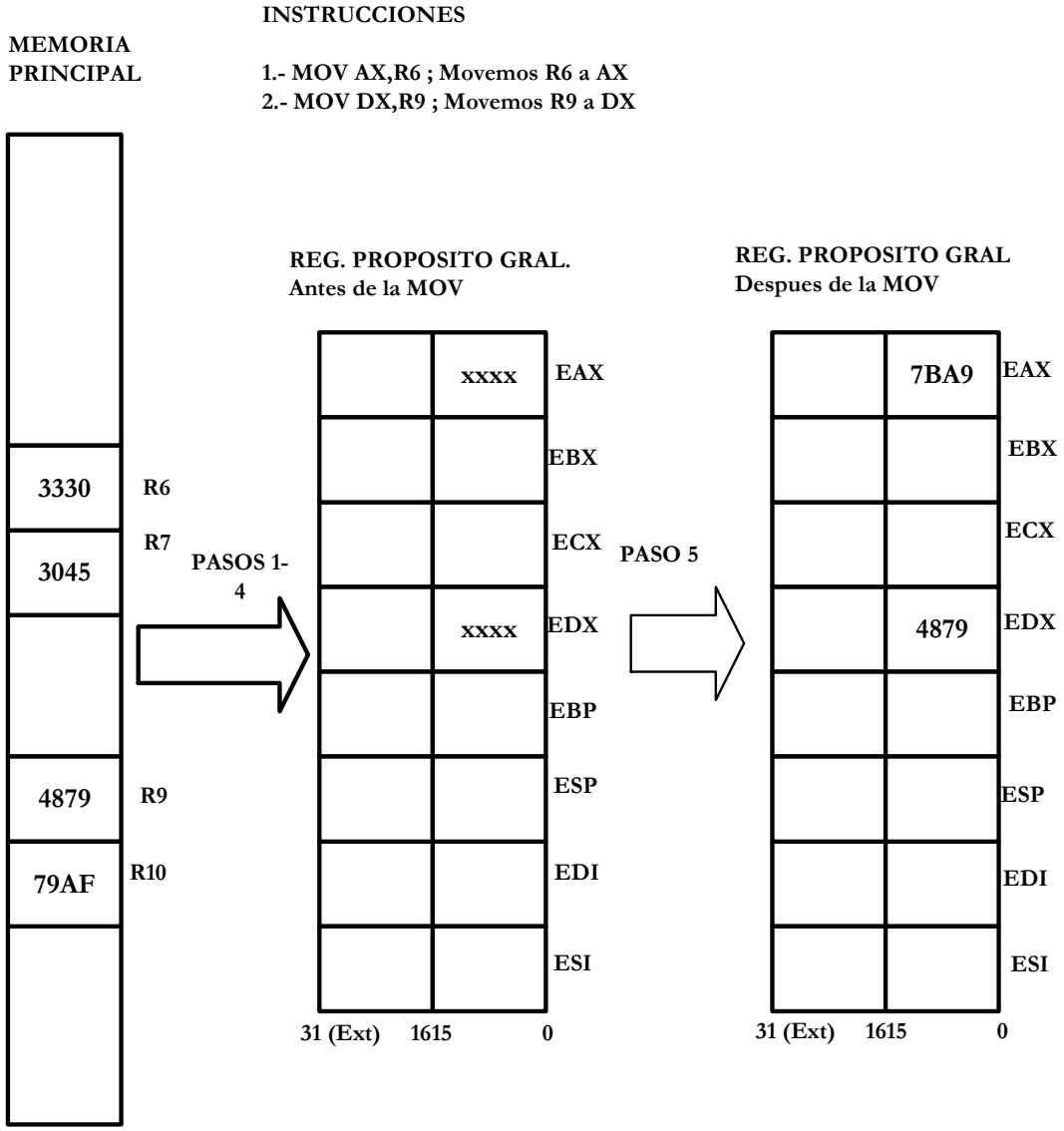


Figura 17.17. Representación gráfica de movimiento de operandos mediante MOV.

XCHG

Intercambia el contenido de los operandos.

LEA

Transfiere a un registro el valor de la dirección que corresponde al segundo operando.

17.8.1- NUEVAS INSTRUCCIONES DE TRANSFERENCIA DE DATOS

El Pentium admite nuevas instrucciones de transferencia para operandos de 32 bits, como *POPAD, POPFD, PUSHFD* y *PUSHAD*. También está la instrucción *MOVSX,* que extiende el signo de un byte ó una palabra a 16 ó 32 bits, respectivamente.

17.9- INSTRUCCIONES DE CONTROL DE LOS SEÑALIZADORES

CLC - STC

Ponen a 0 ó a 1 el señalizador de acarreo, respectivamente.

Reservado **CF (Antes CLC)**

00……..0	ID	VIF	VIP	AC	VM	RF	0	NT	IO	PL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	X
31	**21**																					**0**

REALIZAMOS LA INSTRUCCIÓN

……. CLC; …….

Reservado **CF(Despues CLC)**

00……..0	ID	VIF	VIP	AC	VM	RF	0	NT	IO	PL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	0
31	**21**																					**0**

Figura 17.18. Representación de cómo quedaría EFLAGS tras realizarse la operación CLC

CLD - STD

Ponen a 0 ó a 1 al señalizador D, respectivamente.

CMC

Complementa el valor del señalizador C.

CLI

Pone a 0 el señalizador 1. Inhabilita interrupciones enmascarables.

LAHF

Transfiere al registro AH el byte de menos peso del registro de señalizadores.

Estamos ante el primer byte del registro EFLAGS que va a ser transferido a AH (registro de proposito general)

ZF	1	0	AF	0	PF	1	x

Para un valor del primer byte igual a

1	0	0	0	1	0	1	1

⇨ **8B**

Este valor implica que tras realizar la instrucción LAHF el valor de AH quedara de la siguiente forma

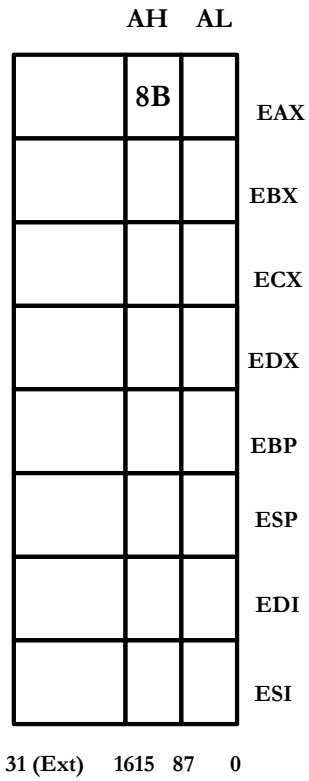


Figura 17.19. Carga del registro AH con el byte de menos peso de EFLAGS

SAHF

Transfiere el contenido de AH al byte de menos peso del registro de señalizadores.

17.10- INSTRUCCIONES DE ASIGNACIÓN CONDICIONAL

Es un grupo de instrucciones nuevas en el Pentium.

Si se cumple la condición implícita en la instrucción, se pone a 1 el operando y, en caso contrario, se pone a 0. Las instrucciones son las siguientes:

SETA

Pone a uno si el primer operando es mayor que el Segundo. Sin signo

SETAE

Pone a uno si el primer operando es menor que el Segundo. Sin signo

SETB

Pone a uno si el primer operando es menor igual que el Segundo. Sin signo

SETBE

Pone a uno si el primer operando es mayor igual que el Segundo. Sin signo

SETC

Pone a uno si el bit de CARRY del EFLAGS es igual a 1

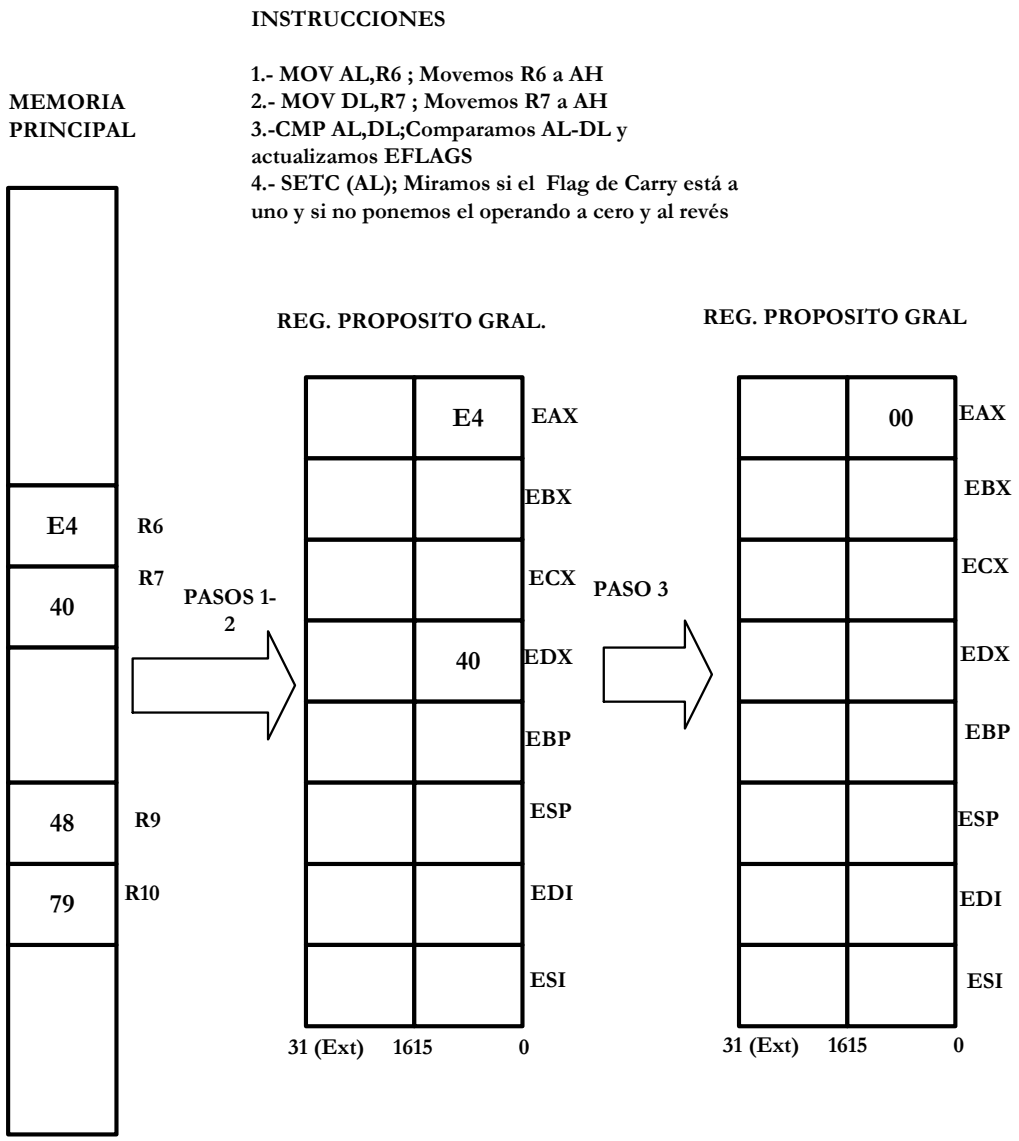


Figura 17.20. La condición no se cumple por lo que el registro se pone a cero

SETNC

Pone a uno si el bit de CARRY del EFLAGS es igual a 1

SETE

Pone a uno si el bit Z del EFLAGS es igual a 1

SETO

Pone a uno si el bit de OVERFLOW del EFLAGS es igual a 1.

SETNO

Pone a uno si el bit de OVERFLOW del EFLAGS es igual a 0.

SETCXZ

Pone a uno si el registro de CX es distinto a 0.

SETS

Pone a uno si el bit S del EFLAGS es igual a 1.

SETNS

Pone a uno si el bit S del EFLAGS es igual a 0.

SETNA

Pone a uno si el primer operando es mayor que el Segundo. Con signo

SETNAE

Pone a uno si el primer operando es menor que el Segundo. Con signo

SETNB

Pone a uno si el primer operando es menor igual que el Segundo. Con signo

SETNBE

Pone a uno si el primer operando es mayor igual que el Segundo. Con signo

SETP

Pone a uno si la paridad es impar o no paridad

SETNP

Pone a uno si la paridad es par o hay paridad

17.11- INSTRUCCIONES DE BIT

Todas estas instrucciones son nuevas.

BT

Asigna al señalizador CF el valor del bit del primer operando, quedando especificada su posición por el segundo operando.

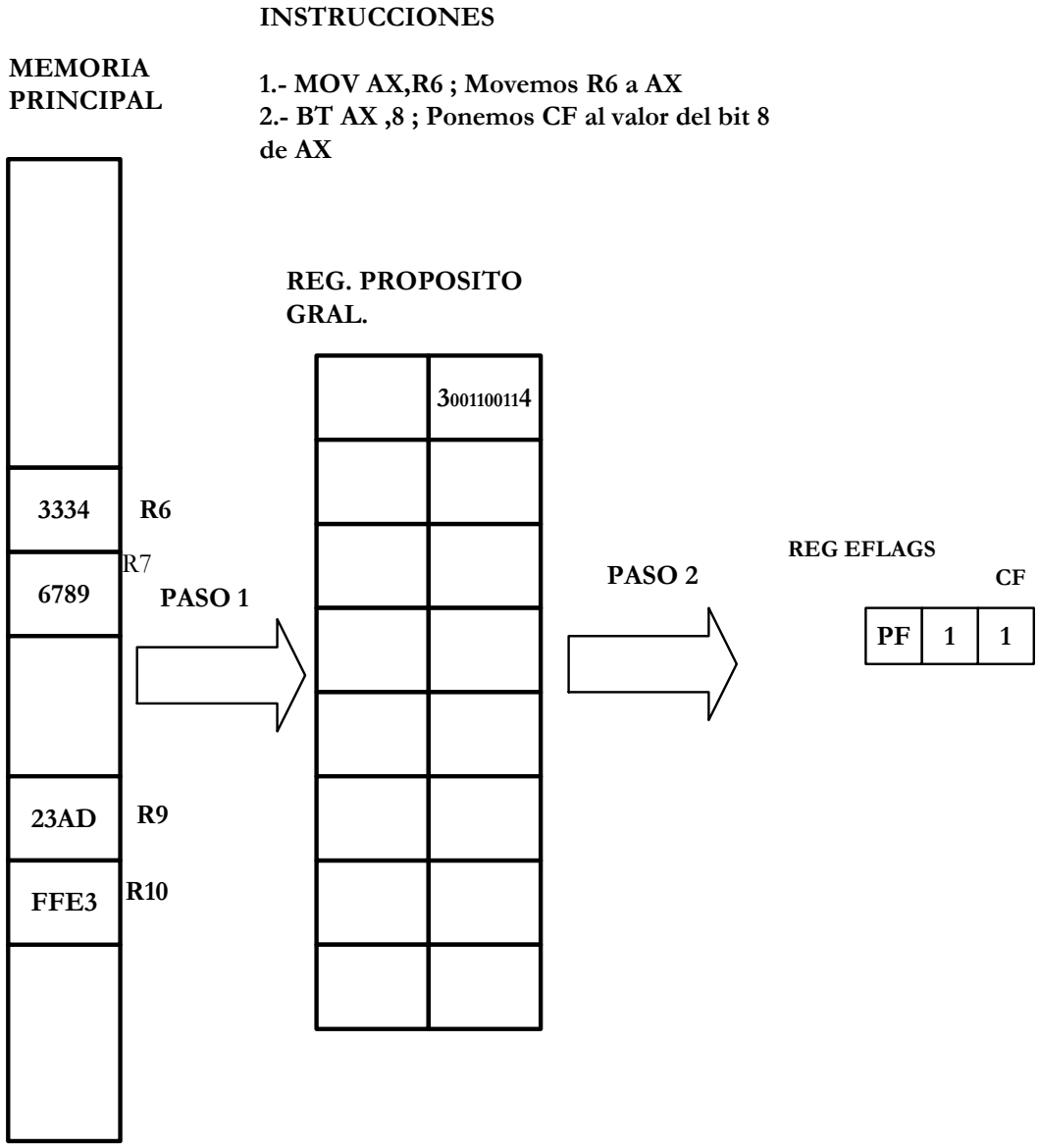


Figura 17.21. El octavo bit de AX es igual a UNO por lo que ponemos CF de EFLAGS a UNO.

BTC

Realiza la misma operación que BT, pero también complementa el bit especificado en La instrucción.

BTR

Igual que BT, pero pone a 0 el bit especificado en la instrucción.

BTS

Igual que la anterior, pero pone a 1 el bit especificado en la instrucción.

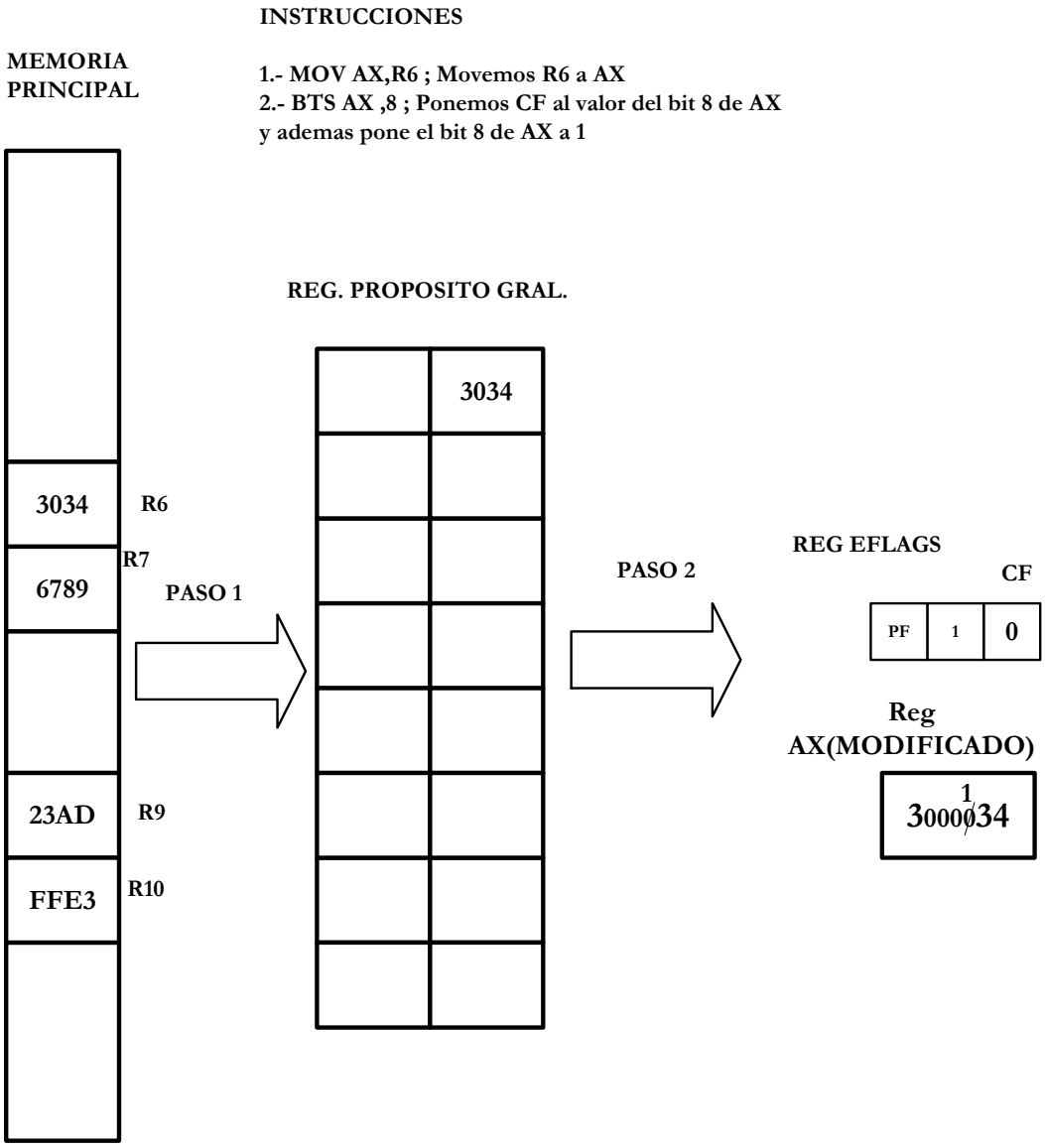


Figura 17.22. El octavo bit de AX es igual a CERO por lo que ponemos CF de EFLAGS a CERO y además complementamos ese bit.

BSF

Si todos los bits, recorridos de menor a mayor peso, del primer operando, especificados por el segundo, son ceros, Z = 1. Si se encuentra algún bit a 1, Z = 0 guardando en el primer operando la posición del primer bit a 1 que se haya encontrado.

BSR

Igual que la anterior instrucción, pero la exploración se realiza desde el bit de más peso al de menos peso.

17.12-INSTRUCCIONES DE ALTO NIVEL

BOUND

Comprueba que un operando está comprendido entre dos límites, y, en caso contrario, genera la excepción *5*.

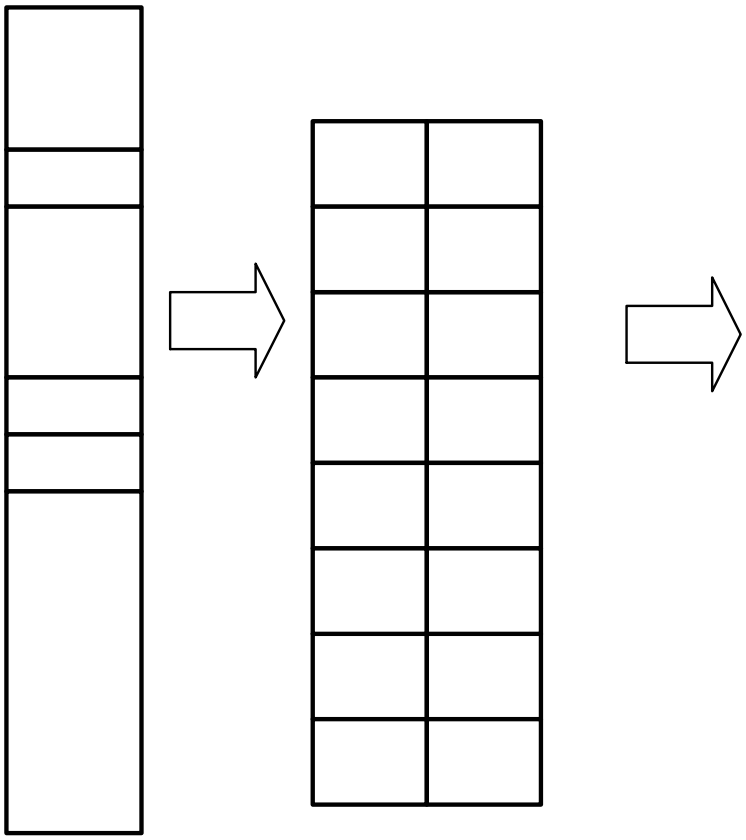


Figura 17.23. BOUND

ENTER

Sirve para crear en la nueva pila que se usa, tras una llamada a un procedimiento, un espacio reservado para el paso de parámetros. El primer operando expresa el número de bytes que se reservan en la nueva pila. El segundo operando expresa el grado de anidamiento.

LEA VE

Elimina el espacio reservado en la pila del procedimiento saliente, asignando como nueva pila la del procedimiento anterior en anidamiento.

17.13- INSTRUCCIONES ESPECIALES

NOP

No efectúa operación alguna.

LOCK

Es un prefijo de algunas instrucciones que, al ser ejecutadas, activan la patilla de salida LOCK# del procesador, impidiendo la cesión del *bus* hasta la finalización de la instrucción con LOCK.

17.14- INSTRUCCIONES MULTISEGMENTO

Implican la utilización de más de un segmento por procedimiento.

CALL

Llamada a una rutina o procedimiento.

RET

Retorno de un procedimiento.

INT

Llamada a un programa de manejo de una interrupción.

INTO

Llamada a la entrada 4 de la tabla de interrupciones, si OF = 1.

IRET

Retorno de un programa de interrupción.

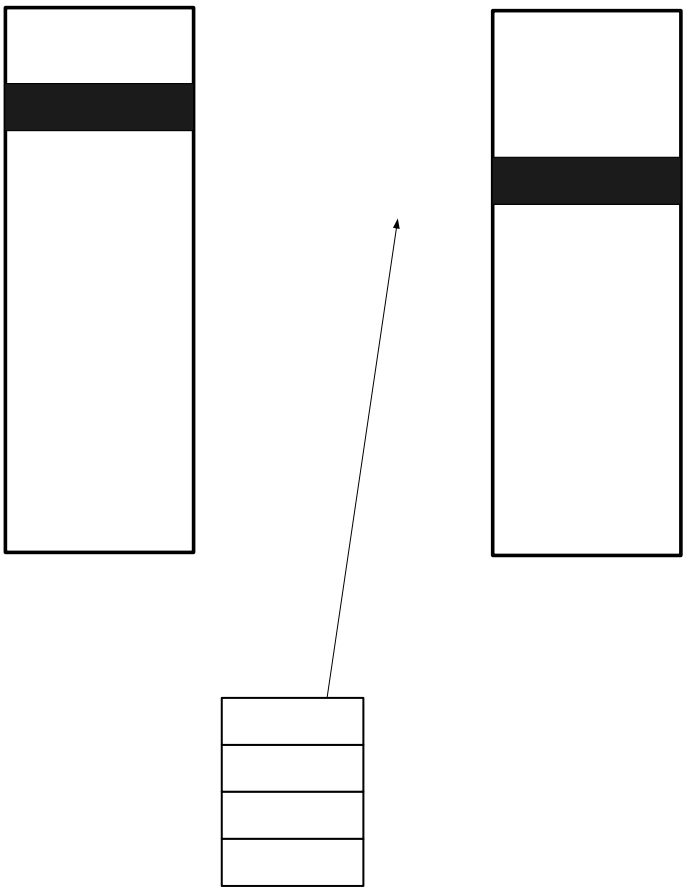


Figura 17.24 Recuperamos de la pila el valor de PC para continuar nuestro programa, después actualizaremos ESI

LDS

Carga simultáneamente el registro DS y otro general con seis bytes de memoria.

LES

Igual que la anterior, pero cargando el registro ES.

JMP

Salto a un segmento de código.

17.14.1- NUEVAS INSTRUCCIONES MULTISEGMENTO

LFS - LGS - LSS

Igual que LDS y LES, pero afectando a los registros FS, GS y SS.

MOV- POP- PUSH

Permiten realizar transferencias de registros de segmentos a otros registros o a la pila.

17.15- INSTRUCCIONES DEL SISTEMA OPERATIVO

ARPL

Comprueba el RPL del primer operando con el del segundo. Si es menor, Z = 1 e iguala los RPL al valor del segundo. (Explicado en el Tema 12 (Puertas de Llamada))

CLTS

Pone a 0 el bit TS de la MSW situada en CR0.

TS(bit de conmutacion de tareas)

PE	CD	NW	RESERVADO	AM	__	WP	RESERVADO	NE	ET	1	EM	MP	PE

31 0

REALIZAMOS LA INSTRUCCIÓN

……. CLTS; …….

TS(bit de conmutacion de tareas)

PE	CD	NW	RESERVADO	AM	__	WP	RESERVADO	NE	ET	0	EM	MP	PE

31 0

Figura 17.25. Representación de EFLAGS tras la realización de la operación CLTS.

HLT

Detiene la ejecución de un programa.

LAR

Carga en el primer operando los derechos de acceso del descriptor al que hace referencia el segundo operando.

LGDT

Carga el GDTR desde una posición de memoria.

LIDT

Carga el IDTR desde una posición de memoria.

LLDT

Carga el registro LDTR.

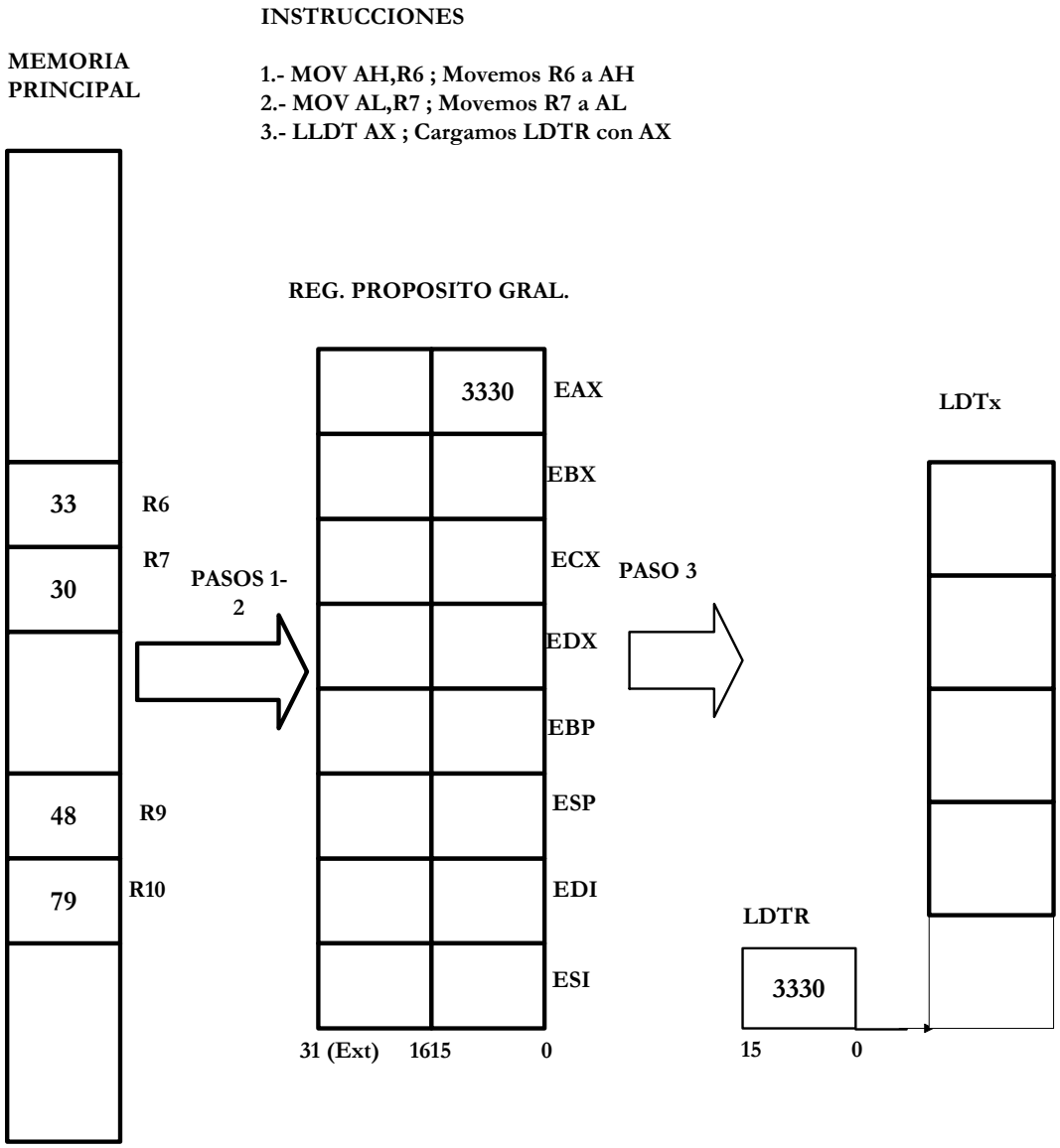


Figura 17.26. Carga de LLDT desde un registro de memoria previamente cargado en AX.

LMSW

Carga a MSW.

LSL

Carga el primer operando con el límite del segmento especificado por el segundo operando.

LTR

Carga el registro TR.

SGDT - SIDT - SLDT- SMSW - STR

Almacenan en la memoria a los registros GDTR, IDTR, LDTR, MSW y TR.

VERR

Verifica si puede ser leído el segmento definido por el selector que se proporciona en el operando de la instrucción. En caso afirmativo, Z = 1.

VERW

Verifica si puede ser escrito un segmento.

17.16- INSTRUCCIONES PARA EL COPROCESADOR

ESC

Precede a las instrucciones que debe procesar el coprocesador matemático. Indica al Pentium que lo que sigue a este prefijo lo envíe al coprocesador.

WAIT

Se detiene el Pentium hasta que la patilla BUSY# se desactive. Así el procesador espera al coprocesador.

17.17.NUEVAS INSTRUCCIONES DEL MICROPROCESADOR PENTIUM

CMPXCHG8B reg, mem64 (Compare and Exchange 8 Bytes)

Compara el valor de 64 bits ubicado en EDX:EAX con un valor de 64 bits situado en memoria. Si son iguales, el valor en memoria se reemplaza por el contenido de ECX:EBX y el indicador ZF se pone a uno. En caso contrario, el valor en memoria se carga en EDX:EAX y el indicador ZF se pone a cero.

CPUID (CPU Identification)

Le informa al software acerca del modelo de microprocesador en que está ejecutando. Un valor cargado en EAX antes de ejecutar esta instrucción indica qué información deberá retornar CPUID. Si EAX = 0, se cargará en dicho registro el máximo valor de EAX que se podrá utilizar en CPUID (para el Pentium este valor es 1). Además, en la salida aparece la cadena de identificación del fabricante contenido en EBX, ECX y EDX. EBX contiene los primeros cuatro caracteres, EDX los siguientes cuatro, y ECX los últimos cuatro. Para los procesadores Intel la cadena es "GenuineIntel".

Luego de la ejecución de CPUID con EAX = 1, EAX[3:0] contiene la identificación de la revisión del microprocesador, EAX[7:4] contiene el modelo (el primer modelo está indicado como 0001b) y EAX[11:8] contiene la familia (5 para el Pentium). EAX[31:12], EBX y ECX están reservados. El procesador pone el registro de características en EDX a 1BFh, indicando las características que soporta el Pentium. Un bit puesto a uno indica que esa característica está soportada. La instrucción no afecta los indicadores.

RDMSR (Read from Model-Specific Register)

El valor en ECX especifica uno de los registros de 64 bits específicos del modelo del procesador. El contenido de ese registro se carga en EDX:EAX. EDX se carga con los 32 bits más significativos, mientras que EAX se carga con los 32 bits menos significativos.

RDTSC (Read from Time Stamp Counter)

Copia el contenido del contador de tiempo (TSC) en EDX:EAX (el Pentium mantiene un contador de 64 bits que se incrementa por cada ciclo de reloj). Cuando el nivel de privilegio actual es cero el estado del bit TSD en el registro de control CR4 no afecta la operación de esta instrucción. En los anillos 1, 2 ó 3, el TSC se puede leer sólo si el bit TSD de CR4 vale cero.

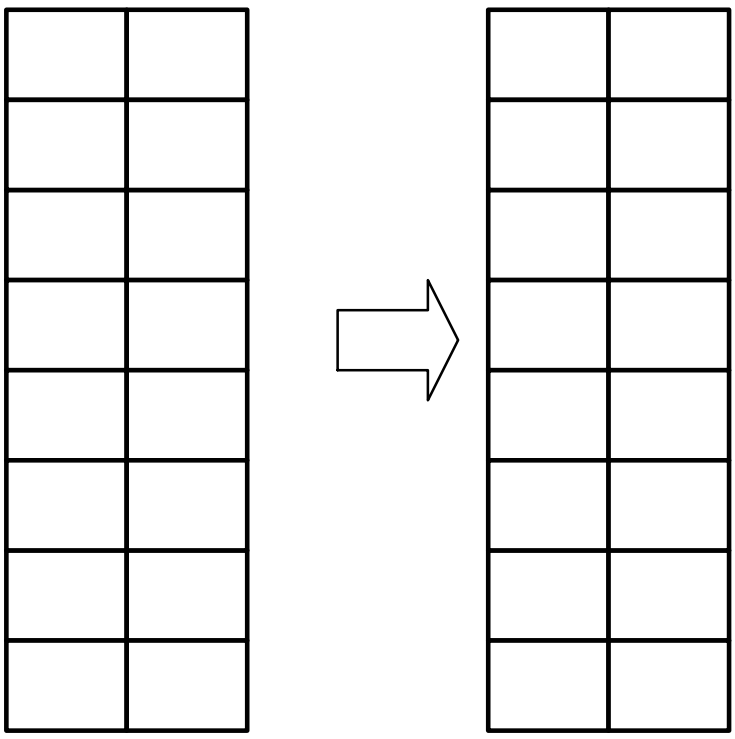


Figura 17.27. Carga del contador del Pentium en los registros de propósito general EAD:EDX

INSTRUCCIONES

1.- RDTSC;Carga el

RSM (Resume from System Management Mode)

El estado del procesador se restaura utilizando la copia que se creó al entrar al modo de manejo del sistema (SMM). Sin embargo, los contenidos de los registros específicos del modelo no se afectan. El procesador sale del SMM y retorna el control a la aplicación o sistema operativo interrumpido. Si el procesador detecta alguna información inválida, entra en el estado de apagado (shutdown).

WRMSR (Write to Model-Specific Register)

El valor en ECX especifica uno de los registros de 64 bits específicos del modelo del procesador. El contenido de EDX:EAX se carga en ese registro. EDX debe contener los 32 bits más significativos, mientras que EAX debe contener los 32 bits menos significativos.

COPROCESADOR MATEMÁTICO (FPU) 18

18.1- INTRODUCCIÓN.

Los coprocesadores matemáticos eran circuitos integrados que se añadían, opcionalmente, a los sistemas procesadores para ampliar el número de instrucciones capaces de interpretar y ejecutar, especialmente de tipo matemático. Un coprocesador se encarga de realizar operaciones con números reales representados según algún estándar, habitualmente el IEEE 754. Los coprocesadores funcionan de dos formas diferentes:

1. Los *memory-mapped* funcionan de forma que sus operandos e instrucciones son colocadas en unas posiciones de memoria determinadas por parte del microprocesador. Cuando tiene el resultado, genera una interrupción y el procesador recoge el resultado de otra posición de memoria. Tienen la ventaja de que son independientes del microprocesador, como por ejemplo los *Weitek*. Evidentemente, se trata de coprocesdores externos. Como coprocesadores matemáticos están prácticamente en desuso; sin embargo, algún otro coprocesador funciona también de esta forma.

2. Otros, sin embargo, comparten el flujo de instrucciones con el microprocesador, de forma que cuando el microprocesador detecta una instrucción que no es suya, la envía al coprocesador, que la ejecuta y devuelve los resultados al microprocesador. Este es el caso de los coprocesadores internos como ocurre en el Pentium en el que las FPU que van incluidas dentro de los microprocesadores

Cada vez es mayor el número de programas y aplicaciones que requieren la actuación de un coprocesador matemático. Se citan algunas populares:

- Programas de simulación.
- Hojas electrónicas.
- CAD-CAM.
- Gráficos.
- Control numérico.
- Robótica.
- Visión computerizada.
- Inteligencia artificial.

18.2- ESTRUCTURA INTERNA.

El coprocesador trabaja internamente sólo en formato real, por lo que cualquier carga en los registros del coprocesador provocará que dicho valor sea convertido a coma flotante.

Sus registros están estructurados en forma de pila y se accede a ellos por el número de entrada que ocupan en la pila.

La FPU es un coprocesador que opera con unidades enteras de otros procesadores que coge sus instrucciones desde el mismo decodificador y secuenciador que la unidad de enteros compartiendo el bus del sistema con esta.. A pesar de este hecho la unidad de enteros y la FPU operan independientemente y en paralelo. (La actual microarquitectura de los procesadores Intel varía entre la variedad de familia de procesadores. Por ejemplo el procesador Pentium Pro posee dos unidades de enteros y dos FPUs mientras que el procesador Pentium tiene dos unidades de enteros y una FPU y el prcesador 486 dispone de una unidad de enteros y una FPU)

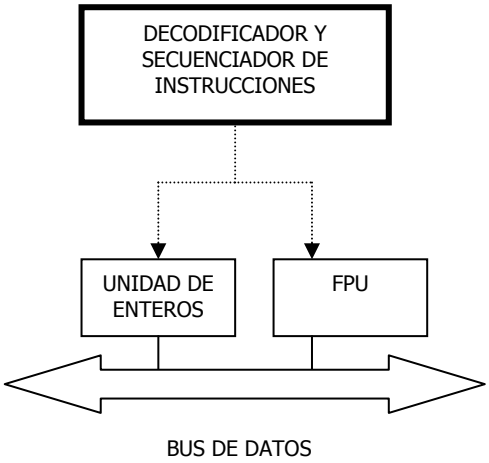


Figura 18.1 Relación entre la unidad de enteros y la FPU

Las instrucciones de ejecucción de la FPU consisten en 8 registros de datos (llamados registros de datos de la FPU) y los siguientes registros especiales:

- Registros de estado
- Registros de control
- Registros de palabra
- Registro puntero de instrucciones
- Registro puntero al último operando (puntero dato)
- Registro códigos

18.2.1 – Registros de datos

Está formada por 8 registros de 80 bits. Los registros generales R1 a R8 soportan datos con el formato normalizado de doble precisión con 80 bits. Como han de ser manejados en formato de pila, el coprocesador tiene un puntero de control de pila llamado "St". Toda interacción que tengamos que hacer con los registros del coprocesador se realiza a través del puntero de pila St, donde el ultimo valor introducido es St o St(0) y si hubiéramos rellenado todos los registros el ultimo seria St(7).

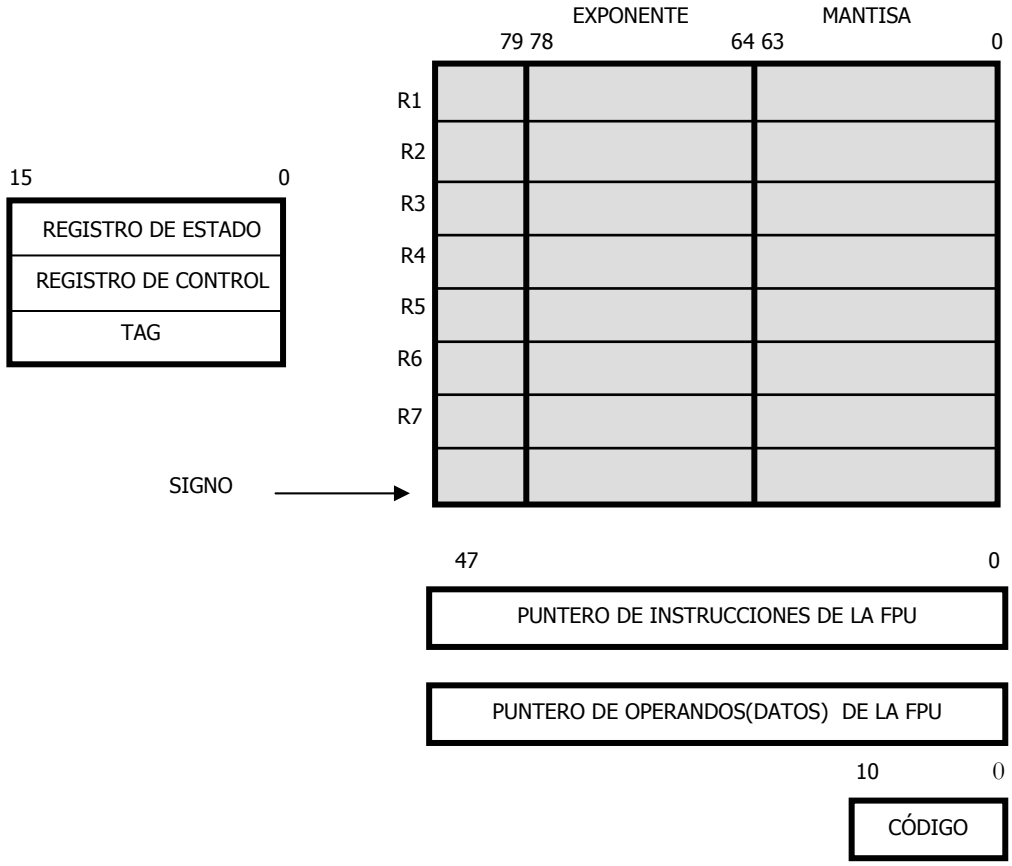


Figura 18.2 Registros del coprocesador

Podemos observar cómo dichos registros están divididos en tres secciones: signo, exponente de 15 bits y mantisa de 64 bits.

Todas las operaciones del coprocesador se realizan usando los ocho registros generales, que están organizados en forma de pila, no pudiéndose acceder a ellos de forma arbitraria.

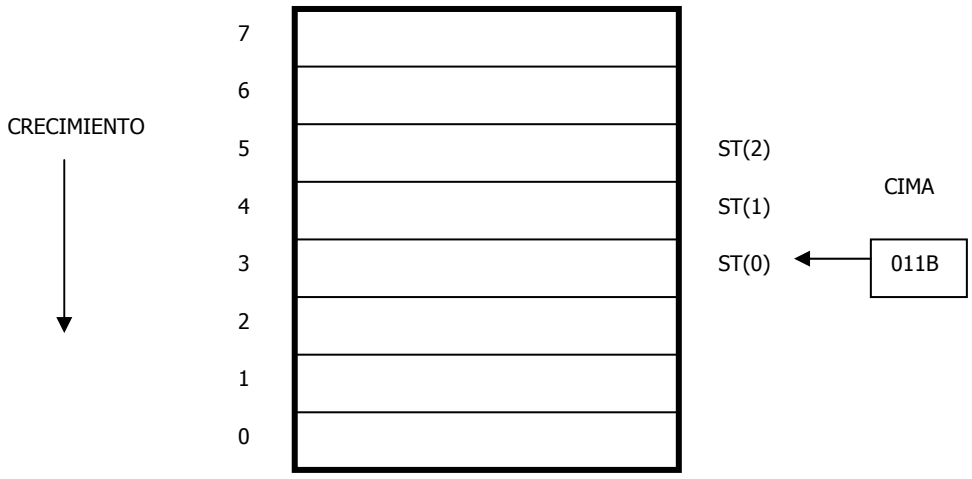


Figura 18.3 Registro de pila de datos de la FPU

Todas las direcciones de los registros de datos están relativas al registro de la cima de la pila. El número de registro de la cima de la pila activo es almacenado en el campo CIMA en la palabra de estado de la CPU. Las operaciones de carga decrementan la cima en unidad en unidad y cargan un valor en la nueva cima de la pila, cargan operaciones, cargan el valor del registro cima actual en memoria y posteriormente incrementan la cima una a una.. (Para la FPU una operación de carga es equivalente a un apilamiento, lo que se conoce como "push" y una operación de almacenamiento equivlae a una extracción o "pop").

Si una operación de carga es realizada cuando la cima está a 0, se crea una devolución del registro y el nuevo valor de la cima es puesto a 7. La excepción de desbordamiento de pila de la unidad de coma flotante, indica cuando es posible que ocurra esta devolución y borran el valor para que sea reescrito. (La excepción que indica desbordamiento de pila es la #IS).

Muchas instrucciones en coma flotante tienen varios modos de direccionamiento para permitir al programador operar implícitamente en la cima de la pila, u operar explícitamente en un registro específico relativo a la cima. Los ensambladores llevan a cabo este modo de direccionamiento, usando la expresión ST(0) o simplemente ST, para representar la cima de la pila actual ST(i) para especificar el registro "i"avo desde la cima en la cima ($0 \leq i \leq 7$). Por ejemplo si la cima contiene 011B (registro 3 en la cima de la pila), la siguiente instrucción debería añadir el contenido de dos registros en la pila (registros 3 y 5).

Como ocurre en los registros de propósito general de en los procesadores de unidades de enteros, el contenido de los registros de datos de la FPU no se ve afectado por procedimientos de llamadas, en otras palabras, los valores se mantienen a través de procedimientos limitantes. Una llamada la pueden usar los registros de datos de la FPU para pasar parámetros entre procedimientos. Los procedimientos de llamada pueden hacer referencia a los parámetros pasados a través de los registros pila usados en el actual puntero de la pila (TOP) y en las nomenclaturas ST(0) y ST(i). Suele ser habitual para un procedimiento de llamada dejar un valor de retorno o resultado en el registro ST(0) cuando se devuelve la ejecución de una llamada o programa.

El coprocesador soporta los siguientes tipos de datos:

1. Enteros de 16, 32 y 64 bits.
2. BCD empaquetados de 80 bits.
3. Números en coma flotante de 32, 64 y 80 bits.

18.2.2 – Registros de estado

Estos 16 bits de registro de estado indican la situación actual de al FPU. Los flags incluidos son los siguientes: flag-acupado o "busy-flag", puntero de la cima de la pila (TOP), flags de condición de código, de estado de error y de excepción. La FPU activa estos flags para mostrar el resultado de sus operaciones.
El contenido de los registros de estado de la FPU (referidos al estado de palabra)pueden ser almacenados en memoria usando la instrucciones FTSW/ FSTSW , FSTENV/FNSTENV, Y FSAVE/FNSAVE. Pueden ser almacenados también en el registro AX de la unidad de enteros usando las instrucciones FSTSW/FNSTSW

En la figura18.4 se muestra la distribución de los 16 bits de Palabra de Estado del coprocesador matemático.

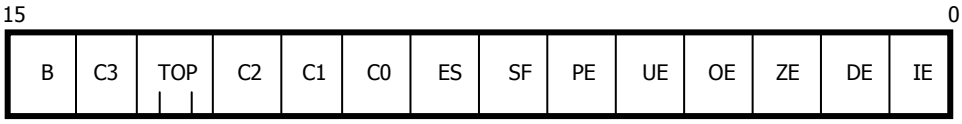


Figura 18.4. Distribución de los bits en la Palabra de Estado del coprocesador matemático.

- **TOP**: Este campo de 3 bits de la Palabra de Estado muestra el primer registro. Realiza labores similares al puntero de pila ESP. Estos bits pueden, mediante instrucciones en coma flotante tales como FLD y FSTP (éstas son similares a las instrucciones de tipo entero PUSH y POP), decrementar el TOP en 1 y colocar un valor en su respectivo registro o incrementar el TOP en 1 y retirar el registro pertinente. La pila se incrementa de manera descendente para registrar números menores. Implícitamente, la mayor parte de instrucciones direccionan el registro cima de la pila, es decir, el registro con el número almacenado en el campo TOP del registro de estado. También podremos especificar explícitamente un registro con diversas instrucciones en coma flotante. Fíjese que el registro especificado explícitamente no es absoluto sino relativo a TOP.

- **B y ES:** Siempre tienen el mismo valor e informa del estado de error. Si ES = 1 se produce una excepción no mascarable cuyo motivo lo indican los bits SP e IE. Bajo ningún concepto el bit B produce ningún dato con respecto al estado de la unidad numérica y en consecuencia al pin BUSY.

- **SF:** Diferencia entre las operaciones inválidas causadas por el desbordamiento de la pila y por otras causas. Si SF está activado el bit C1 diferencia entre un overflow (C1=1) y un underflow (C1=0). Para una interpretación del código C3-C0 de una comparación u operaciones similares adjuntamos la tabla 7.1.

- Los cuatro **flags de condición de código** (desde C0 a C3) indican el resultado de operaciones aritméticas y de comparación en coma flotante. Estos flags se usan principalmente para almacenamiento de información usada en excepciones. El flag de condición de estado C1 es usado para gran variedad de funciones. Cuando, los bits IE y SF de la FPU están activados, indicando desbordamiento de la excepción overflow o underflow (#IS), el flag C1 lo distingue de la siguiente manera: si esta a 1 overflow , sino underflow. Cuando el flag PE de palabra de estado está activado indica que se ha producido un resultado inexacto (redondeo), el flag C1 es puesto a 1 si los últimos redondeos han sido descendentes. La instrucción FXAM activa C1 para examinar el valor del signo.

El bit de condición de código C2 es usado por las instrucciones FPREM y FPREM1 para indicar una reducción correctamente (o resto parcial). Cundo se completa una reducción completamente, los flags de condición de estado C0, C3 y C1 son activados haciendo referencia a los tres bits menos significativos del cociente (Q2, Q1 y Q0 respectivamente).
Las instrucciones FPTAN; FSIN,FCOS y FSINCOS ponen el flag C2 a 1 para indicar que el operando fuente está fuera del rango permitido ($\pm 2^{63}$)

INSTRUCCIÓN	C0	C3	C2	C1
FCOM, FCOMP, FCOMPP, FICOM, FICOMP, FTST, FUCOM, FUCOMP, FUCOMPP	Resultados de comparaciones		Operandos no Comparables	0 ó #IS
FCOMI, FCOMIP, FUCOMI, FUCOMIP	No definidas			#IS
FXAM	Clase de operando			Signo
FPREM, FPREM1	Q2	Q1	0 = reducción completa 1 = reducción incompleta	Q0 ó #IS
F2XM1, FADD, FADDP, FBSTP, FCMOVcc, FIADD, FDIV, FDIVP, FDIVR, FDIVRP, FIDIV, FIDIVR, FIMUL, FIST, FISTP, FISUB, FISUBR,FMUL, FMULP, FPATAN, FRNDINT, FSCALE, FST, FSTP, FSUB, FSUBP, FSUBR, FSUBRP,FSQRT, FYL2X, FYL2XP1	No definido			Roundup ó #IS
FCOS, FSIN, FSINCOS, FPTAN	No definido		1 = operando fuente fuera de rango	Roundup ó #IS (No definido si C2=1)
FABS, FBLD, FCHS, FDECSTP, FILD, FINCSTP, FLD, Constantes de Carga, FSTP (ext. real), FXCH, FXTRACT	No definido			0 ó #IS
FLDENV, FRSTOR	Cada bit se carga de la memoria			
FFREE, FLDCW, FCLEX/FNCLEX, FNOP, FSTCW/FNSTCW, FSTENV/FNSTENV, FSTSW/FNSTSW,	No definido			
FINIT/FNINIT, FSAVE/FNSAVE	0	0	0	0

Tabla 18.1. Códigos en coma flotante para el Pentium.

- **PE:** Precisión.
- **UE:** Underflow.
- **OE:** Overflow.
- **ZE:** División por cero.
- **DE:** Operando desnormalizado.
- **IE:** Operación inválida.

} 0 = No se produce excepción
1 = Condición de excepción generada

Estos 6 flags indican que una o más excepciones en coma flotante han sido detectadas desde que los bits fueron limpiados por última vez.
Estos flags de excepcion se pueden considerar un poco pegajosos, es decir, permanecen activados hasta que son explícitamente limpiados. Para ello se utilizan las siguintes instrucciones:

- FCLEX/FNCLEX (excepciones de limpieza)
- FINIT/FNINIT (reinilializan la FPU)

- FSAVE/FNSAVE
- FRSTOR o FLDENV (para sobreescribir los flags)

18.2.3 – Saltos y movimientos condicionales en la condición de código de la FPU

La arquitectura de la FPU de Intel (comenzó con el procesador Pentium PRO) posee dos mecanismos para saltar y efectuar movimientos condicionales de acuerdo con las comparaciones de dos valores en coma flotante. Estos mecanismos se refieren a "nuevo mecanismo" y "viejo mecanismo".

El mecanismo viejo está disponible a priori en la FPU de los procesadores Pentium Pro. Este mecanismo usa instrucciones de comparación en coma flotante (FCOM, FCOMP, FCOMPP, FTST, FUCOMPP, FICOM Y FICOMP) para comparar dos valores en coma flotante y activar los flag de condición de código (C0, C3) de acuerdo con los resultados. Los contenidos de la condición de código son después copiados en los flags de estado del registro EFLAGS mediante dos pasos como se puede ver en la figura de la siguiente página.

1. La instrucción FSTSW AX mueve la palabra de estado de la FPU al registro AX.
2. La instrucción SAHF copia los 8 bits de más peso, en los cuales están incluidos los flags de condición de código, dentro de los 8 bits de menos peso de l registro EFLAGS

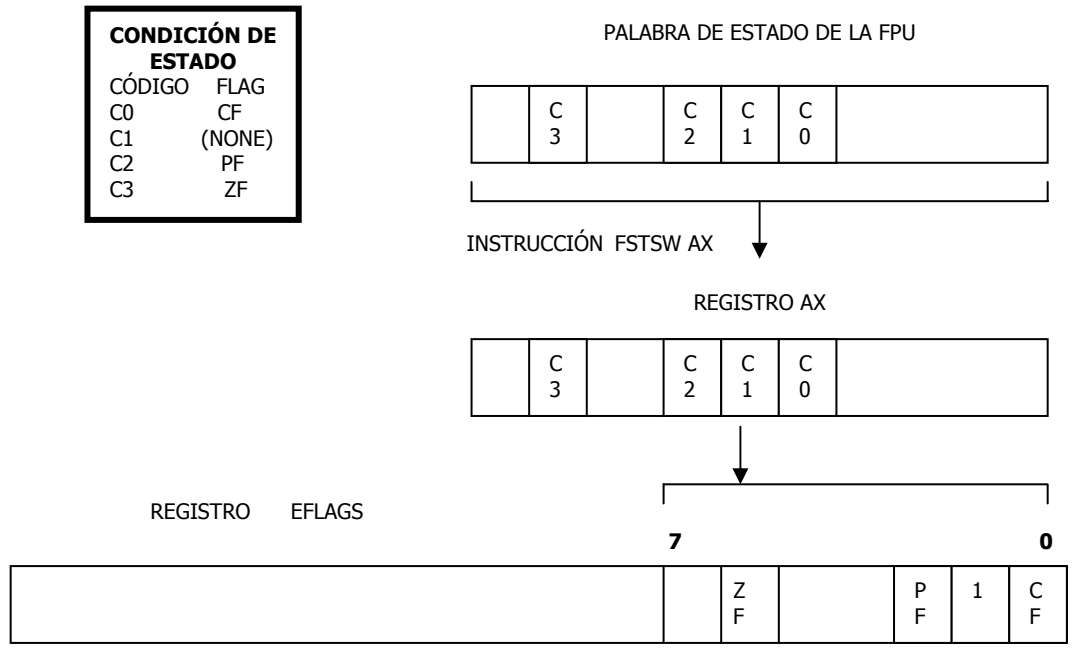


Figura 18.5 Movimiento de la condición de código al registro EFLAGS

El mecanismo nuevo está solamente disponible en el procesador Pentium Pro. Usando este mecanismo la nueva comparación en coma flotante compara y activa las instrucciones EFLAGS (FCOMI, FCOMP. FUCOMI, Y FUCOMIP) que comparan dos valores en coma flotante y activan directamente los registros EFLGS ZF, y PF. Una única instrucción sustituye tres instrucciones requeridas en el viejo mecanismo.

18.2.4 – Registros de control

Estos registros controlan la precisión de la FPU y los métodos de redondeo usados. También existen unos bits que hacen posible el enmascaramiento de excepciones. El contenido de estos registros puede ser cargado con la instrucción FLDCW y almacenado en memoria con la instrucción FSTCW/FNSTCW.
Cundo la FPU es inicializada con las instrucciones FINIT/FNINIT o FSAVE/FNSAVE, la palabra de control es puesta a 037FH, la cual enmascara todas las excepciones en coma flotante, redondea hacia el más próximo y se activa la FPU con una precisión de 64 bits.

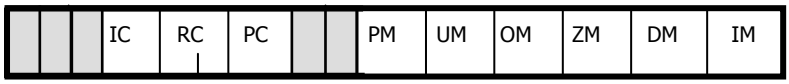


Figura 18.6. Distribución de los bits en la Palabra de Control del coprocesador matemático.

Bajo ciertas circunstancias, el Pentium genera una excepción de coprocesador. Estas excepciones pueden ser individualmente enmascaradas. Lo que es más, podemos determinar diferentes modos para precisión y redondeo. La Palabra de Control se emplea para este propósito, y su estructura la mostramos en la figura 18.4

- **IC:** No tiene significado cuando no manejan valores infinitos porque el Pentium aplica el estándar IEEE a las operaciones con coma flotante. En los terrenos de compatibilidad con el 8087 y el 80287 el bit IC está disponible pero no tiene efecto.

  El Pentium siempre maneja cantidades infinitas en el sentido de $\pm\infty$, incluso si ponemos IC a 0.

- **RC:** Los dos bits RC controlan el redondeo en el modo definido.

MODO DE REDONDEO	CAMPO RC
Redondeo al más cercano	00B
Redondeo hacia abajo	01B
Redondeo hacia arriba	10B
Redondeo a cero	11B

Tabla 18.2. Códigos en coma flotante para el Pentium.

- **PC:** El campo de control de precisión (bits 8 y 9 de la Palabra de Control de la FPU) determina la precisión (64, 53, o 24 bits) de los cálculos en coma flotante realizados por la FPU. Controla el redondeo en las instrucciones ADD, SUB, MUL, DIV y SQRT.

PRECISIÓN	CAMPO PC
Precisión simple (24 bits)	00B
Reservado	01B
Doble precision (53-Bits)	10B
Precisión extendida (64-Bits)	11B

Tabla 18.3. Tabla del campo PC en relación a la precisión.

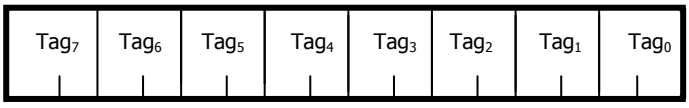
- **PM, UM, OM, ZM, DM e IM:** Controlan la generación de una excepción cada uno y su interrupción resultante. Los Pentium provocan 6 excepciones diferentes en sus operaciones con coma flotante. Podemos enmascarar las excepciones individualmente empleando los bits PM, UM, OM ZM, DM e IM. Entonces el Pentium ejecuta una rutina estándar para tratar los respectivos errores utilizando un supuesto estándar. Esto es una parte integral del chip.

18.2.5 – Flag de control de infinito

Este flag proporciona compatibilidad con el coprocesador matemático 287 de Intel; no es muy significante para versiones posteriores de la arquitectura IA-32

18.2.6 – Registros de palabra

. La FPU usa los valores de los tag para detectar desbordamientos en condiciones de overflow o underflow . Los desbordamiento de overflow ocurren cuando el puntero del campo TOP es decrementado para apuntar a un registro no vacío. Los desbordamientos de underflow se producen cuando el puntero del campo TOP es incrementado para apuntar a un registro vacío, o cuando un registro vació es referenciado como un operando fuente. Un registro no vacío viene definido como un registro que contiene valor cero (01), valor válido (00), o valor especial (10). Su estructura la mostramos en la figura 7.16. Tag_7 – Tag_0 contiene información, que identifica los contenidos de los ocho registros de datos R7-R0. El coprocesador utiliza esta información para realizar ciertas operaciones a una velocidad superior. Empleando este proceso, el Pentium puede determinar muy rápidamente ciertos valores como NAN, infinito y sin la necesidad de decodificar el valor del registro correspondiente. Mediante las instrucciones de FSTENV/FNSTENV podemos almacenar la Palabra Tag en memoria y examinarla.



00 = VÁLIDO.
01 = CERO.
10 = NAN, INFINITO, VALOR DESNORMALIZADO O FORMATO DE NÚMERO INVÁLIDO.
11 = LLENO.

Figura 18.7 Distribución de los bits en la Palabra Tag del coprocesador matemático.

18.2.7 – Punteros de instrucciones y datos

La FPU almacena instrucciones a datos (operandos) e instrucciones para las últimas instrucciones ejecutadas que no son de control. Estos punteros son almacenados en dos registros de 48 bits.

Hay que tener en cuenta que el valor de un registro puntero de dato es siempre un puntero a un operando de memoria. Si las últimas instrucciones no controladas que fueron ejecutadas no contenían un operando en memoria, el valor del registro puntero de datos es indefinido.

El contenido de los registros de los punteros de instrucciones y datos permanecen invariables cuando no se ejecuta ninguna instrucción de control. (FINIT/FNINIT, FCLEX/FNCLEX... etc). Los punteros almacenados en los registros de los punteros de instrucciones y datos contienen un offset (almacenado en los bits del 0 al 31) y un selector de segmento (almacenado en los bits del 32 al 47)

18.2.8 – Registros de código

La FPU almacena el código de las últimas instrucciones ejecutadas que no sean de control en un registro de código de 11 bits. Sólo el primer y el segundo byte de código están almacenados en el registro de código de la FPU. La figura muestra el almacenamientos de estos 2 bytes. Desde que los 5 bits de más peso del primer byte de código son los mismos para todos los códigos de unidad en coma flotante (11011B), solo los tres bits de menos peso son almacenados en este registro de código.

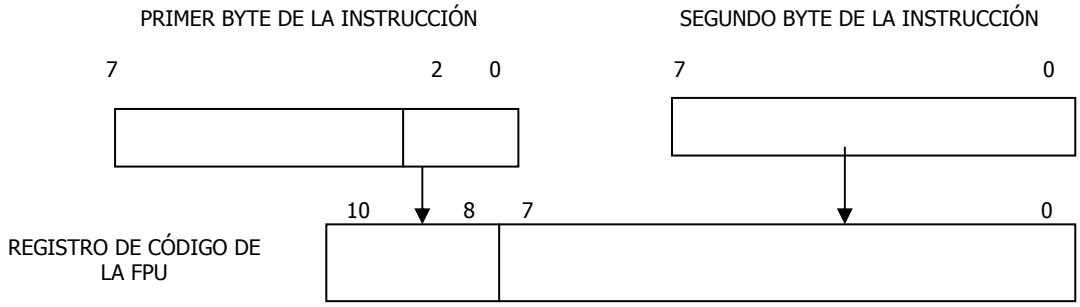


Figura 18.8 Contenido del registro código

18.3- TIPOS DE DATOS

El coprocesador puede obtener y escribir datos en memoria de los siguientes tipos.

- **Entero:** Words(16 bits), Dword(32 bits), Qwords(64 bits)
- **Real:** Words(16 bits), Dword(32 bits), Qwords(64 bits), Twords(80 bits)
- **Simple precisión en coma flotante**
- **Doble precisión en coma flotante**
- **Doble precisión extendida en coma flotante**
- **Entero con signo**
- **BCD**

A continuación se muestra una tabla de las excepciones que existen en doble precisión en coma flotante.

Excepción	Causa	Acción por defecto (si la excepción está enmascarada)
Invalid Op.	Operación sobre un Signaling NaN, formato no soportado, operación indeterminada (0*infinito, 0/0, infinito-infinito, etc.), o underflow/overflow de pila	El resultado es un Quiet NaN, un entero indefinido o un BCD indefinido.
Denormalized Op.	Al menos uno de los operandos es un Denormal, o sea, tiene el menor exponente pero una mantisa no-cero.	Continua el proceso normal
Zero Divisor	El divisor es cero mientras el dividendo es un número no cero y no infinito.	El resultado es Infinito.
Overflow	El resultado es demasiado grande para caber en el formato especificado.	El resultado es el mayor número posible o infinito.
Underflow	El resultado es no-cero pero es demasiado pequeño para caber en el formato especificado, y si está enmascarada la excepción de Underflow, la denormalización causa pérdida de precisión.	El resultado es un denormal o cero.
Resultado Inexacto (Precisión)	El resultado EXACTO no es Representable en el formato Especificado (p.ej. 1/3); el resultado es redondeado de acuerdo con el modo de redondeo.	Continua el proceso normal.

Tabla 18.4 - Tabla de excepciones

18.4- ORÍGENES DEL COPROCESADOR MATEMÁTICO EN EL PENTIUM

Esta unidad se ha rediseñado totalmente respecto a la que se usa el 486. Sin embargo, mantiene compatibilidad 100% binaria con ella. Incorpora un cauce segmentado de instrucciones de ocho etapas, que permite obtener resultados partiendo de instrucciones de coma flotante en cada ciclo de reloj. Las cuatro primeras etapas son las mismas que poseen las unidades de enteros. La quinta y la sexta, corresponden a la ejecución de las instrucciones de coma flotante. La séptima etapa se encarga de escribir el resultado en los registros adecuados y la octava realiza el informe de posibles errores que se hayan producido.

Hace uso de nuevos algoritmos que aceleran la ejecución de las operaciones e incluye elementos de hardware dedicados, como son : un multiplicador, un sumador y un divisor. Instrucciones de suma, multiplicación y carga de datos se ejecutan tres veces más rápido que en un 486.

La Unidad en Coma Flotante o FPU integrada en el Pentium amplía el número de, así como el repertorio de instrucciones. Los nuevos registros que proporciona el coprocesador son ocho generales de 80 bits cada uno y tres de 16 bits, siendo éstos últimos empleados en labores de control y presentación de estado.

18.5- INSTRUCCIONES

Estas son las instrucciones más comunes, se omiten algunas, por ser poco comunes en el mundo del tiempo-real. Los ciclos de reloj de las instrucciones, son los declarados por Intel para el Pentium básico, esto puede verse alterado en las CPU's MMX, PRO y Pentium II. Cualquier instrucción de cálculo (fadd, fsub, etc.) toman por defecto si no se le especifica otros operandos St(0) y St(1).

Leyenda: rm=Modo Real, vm=Modo Virtual, pm=Modo Protegido

INSTRUCCIÓN	EJEMPLO	CICLOS DE RELOJ
FABS	fabs	1
FADD [reg,reg]	fadd	3, 1
FADD memreal	fadd shortreal	3, 1
FADDP reg,ST	faddp st(6),st	3, 1
FIADD memint	fiadd int16	7, 4
FCHS	fchs	1
FCLEX	fclex	9+
FNCLEX	fnclex	9
FCOM	fcom	4, 1

FCOMP	fcomp	4, 1
FCOMPP	fcompp	4, 1
FICOM memint	ficom double	8, 4
FICOMP memint	ficomp darray[di]	8, 4
FCOS	fcos	18-124
FDECSTP	fdecstp	1
FDIV [reg,reg]	fdiv st(5),st	39
FDIV memreal	fdiv longreal	39
FDIVP reg,ST	fdivp st(6),st	39
FIDIV memint	fidiv warray[di]	42
FDIVR [reg,reg]	fdivr st(5),st	39
FDIVR memreal	fdivr longreal	39
FDIVRP reg,ST	fdivrp st(6),st	39
FIDIVR memint	fidivr warray[di]	42
FFREE ST(i)	ffree st(3)	1
FILD memint	fild quads[si]	3, 1
FINCSTP	fincstp	1
FINIT	finit	16
FNINIT	fninit	12
FIST memint	fist doubles[8]	6
FISTP memint	fistp longint	6
FLD reg	fld st(3)	1
FLD mem32real	fld longreal	1
FLD mem64real		1
FLD mem80real		3
FLD1	fld1	2

FLDZ	fldz	2
FLDPI	fldpi	5, 3
FLDL2E	fldl2e	5, 3
FLDL2T	fldl2t	5, 2
FLDLG2	fldlg2	5, 3
FLDLN2	fldln2	5, 3
FLDCW mem16	fldcw ctrlword	7
FLDENV mem	fldenv [bp+10]	37, 16-bit pm=32, 32-bit pm=33
FMUL [reg,reg]	fmul st(5),st	3, 1
FMULP reg,ST	fmulp st(6),st	3, 1
FIMUL memint	fimul warray[di]	7, 4
FNOP	fnop	1
FPATAN	fpatan	17-173
FPREM	fprem	16-64
FPREM1	fprem1	20-70
FPTAN	fptan	17-173
FRNDINT	frndint	9-20
FRSTOR mem	frstor [bp-94]	16-bit rm or vm=75; 32-bit rm or vm=95; pm=70
FSAVE mem	fsave [bp-94]	16-bit rm or vm=127+; 32-bit rm or vm=151+; pm=124+
FNSAVE mem	fnsave [bp-94]	16-bit rm or vm=127; 32-bit rm or vm=151; pm=124
FSCALE	fscale	20-31
FSIN	fsin	16-126
FSINCOS	fsincos	17-137
FSQRT	fsqrt	70
FST reg	fst st	1

FST memreal	fst longs[bx]	2
FSTP reg	fstp st(3)	1
FSTP mem32real	fstp longreal	2
FSTP mem64real		2
FSTP mem80real		3
FSTCW mem16	fstcw ctrlword	2+
FNSTCW mem16	fnstcw ctrlword	2
FSTENV mem	fstenv [bp-14]	16-bit rm or vm=50+; 32-bit rm or vm=48+; 16-bit pm=49+; 32-bit pm=50+
FNSTENV mem	fnstenv [bp-14]	16-bit rm or vm=50; 32-bit rm or vm=48; 16-bit pm=49; 32-bit pm=50
FSTSW mem16	fstsw statword	2+
FSTSW AX	fstsw ax	2+
FNSTSW mem16	fnstsw statword	2
FNSTSW AX	fnstsw ax	2
FSUB [reg,reg]	fsub st,st(2)	3, 1
FSUB memreal	fsub longreal	3, 1
FSUBP reg,ST	fsubp st(6),st	3, 1
FISUB memint	fisub double	7, 4
FSUBR [reg,reg]	fsubr st,st(2)	3, 1
FSUBR memreal	fsubr longreal	3, 1
FSUBRP reg,ST	fsubrp st(6),st	3, 1
FISUBR memint	fisubr double	7, 4
FTST	ftst	4, 1
FUCOM [reg]	fucom st(2)	4, 1
FUCOMP [reg	fucomp st(7)	4, 1

FUCOMPP	fucompp	4, 1
FWAIT	fwait	1-3
FXAM	fxam	21
FXCH [reg]	fxchg st(3)	1
FXTRACT	fxtract	13
FYL2X	fyl2x	22-111
FYL2XP1	fyl2xp1	22-103

Tabla 18.5. Instrucciones

18.6- HIOSTORIA DEL ERROR DE LA FPU DEL PENTIUM

El 30 de octubre de 1.994, el doctor Thomas R. Nicely, profesor de Matemáticas del Lynchburg College del Estado de Virginia (U.S.A.), enviaba un mensaje mediante correo electrónico a Andrew Schulman, coautor del libro Undocumented DOS y autor de Undocumented Windows (bien conocidos por los programadores de sistemas, por tratar de interrupciones y opciones diversas no documentadas de estos sistemas) como uno más de los comentarios o informaciones sobre este tipo de asuntos que Andrew solicitaba que le enviaran por este procedimiento.

En el mensaje, el doctor Nicely le comunicaba que mientras investigaba con el programa Matlab sobre números primos elevados con un Pentium, observó que éste le dio un resultado erróneo a una determinada operación de división con punto flotante, lo que inducía a pensar en un error en la FPU (Floating Point Unit, Unidad de punto flotante o coprocesador) del Pentium, por resultar correcta idéntica operación en los 486. Andrew envió copia del mensaje a Richard Smith, de la casa de software Phar Lap, para que pudiese comprobar el fallo en algún equipo Pentium, de los que él no disponía. Hecho esto, y a la vista de la importancia que podría tener el asunto, lo notificó públicamente la noche del 1 de noviembre en el fórum de discusión Canopus de Compuserve. Las pruebas se multiplicaron, y sencillamente desde la calculadora de Windows, resultaba fácil constatar que las matemáticas no eran exactas, al menos para el Pentium. Poco después Alex Wolfe, de la revista Electronic Engeneering Times, se interesaba por la nota que aparecía en Canopus y, comprendiendo su importancia, publicó el primer artículo escrito sobre el error, en la primera plana de la edición del 7 de noviembre de la revista. Antes, notificó lo que estaba investigando a Terje Mathisen, de Noruega, quien estaba escribiendo en USENET sobre las instrucciones del Pentium, y éste hizo público el error el 3 de noviembre en el grupo comp.sys.intel de USENET. Una vez aquí, Tim Coe, de Vitesse Semiconductor, logró describir correctamente cómo se producía el error y efectuó predicciones correctas sobre qué pares de números lo ocasionarían. El error siguió siendo discutido en este foro, y desde ahí se conoció en toda Internet, la mayor red de ordenadores que llega a grandes partes del mundo, y precursora de la no muy lejana Autopista de la Información, lo cual supone hablar de una audiencia de veinticinco millones de personas en la actualidad (en constante crecimiento). He aquí cómo la información se ha transmitido, siempre en forma de bytes y a través de los cables, hasta llegar a nuestros oídos. Pero qué es exactamente lo que falla? En términos prácticos, el error consiste en que se redondean mal los dígitos decimales desde el cuarto al decimonoveno en determinadas divisiones en coma flotante: la precisión de la respuesta es mucho menor de la esperada. Esto se produce independientemente de la precisión, simple, doble o extendida, de los números; tampoco depende de la velocidad del chip, ni de las instrucciones previas, ni del modo de redondeo. El fallo reside en la unidad de división de la FPU, y por tanto

cualquier instrucción que la utilice puede mostrar el error; no sólo la conocida FDIV, sino otras indicadas por Intel, como FPTAN, FPATAN (tangente y arcotangente en punto flotante), FYL2X, FYL2XP1 (logaritmos) y la instrucción de resto; en cambio, no fallan el seno ni el coseno, entre otros.

La causa final del error es, obviamente, humana: el olvido de unas cuantas entradas en una tabla de consulta empleada por la unidad de división. La división en punto flotante en la mayoría de los microprocesadores, como el Pentium, se realiza usando un algoritmo similar al de la caja que todos aprendimos en el colegio, y que, como aquél, produce un bit en cada ciclo de la operación. Adicionalmente, el Pentium posee un sistema de optimización que le permite deducir dos bits en cada ciclo, usando en un determinado punto de la división los restos del dividendo y del divisor como índices a una tabla de consulta de donde obtiene los siguientes dos bits. Si desafortunádamente esos restos llevan a una de las entradas que faltan en la tabla, se obtienen los dos bits incorrectos, y el resto ya se sabe... dos bits que han dado la vuelta al mundo informático. Concretamente, IBM ha indicado que las secuencias de bits peligrosas son éstas: 10001, 10100, 10111, 11010 y 11101, seguidas por una cadena de al menos 20 unos.

De lo anterior se deduce que el error sucede siempre al dividir pares de números concretos. Si dos números fallan, lo harán siempre que se repita la prueba; si no es así, nunca darán problemas. En total, considerando sólo los casos de números de simple precisión con una exactitud en el resultado inferior a la simple precisión, sólo parecen existir 1.738 pares que producen el error, y sólo 87 con una precisión de cuatro dígitos decimales. Dada la naturaleza del error, hay que añadir a estos números todos los obtenidos al multiplicar o dividir a uno o ambos números del par por una potencia de dos, y/o cambiarle el signo, ya que el error lo determinan las secuencias de bits en las mantisas, con lo que los números de igual mantisa han de ser considerados un único caso. Para la doble precisión, los casos se multiplican, y más aún si aumentamos la precisión requerida al resultado, pero al mismo tiempo la posibilidad de encontrar un error con esos números disminuye rápidamente.

18.6.1 – Mantisas, exponentes y precisiones

El Pentium, como el resto de los microprocesadores, emplea el standard IEEE 754 para representar números en punto flotante. Este tipo de almacenamiento transforma los números reales al formato signo*1.XXX*2^YYY, es decir, uno más una fracción, multiplicado por una potencia de dos, junto con el signo. El XXX se denomina mantisa; YYY es el exponente.

El standard IEEE define el número de simple precisión como aquél que tiene una mantisa de 23 bits. Esto proporciona una precisión de 24 bits (contando el primer bit) lo que equivale a aproximadamente 7 digitos decimales. Un número de doble precisión tiene una mantisa de 52 bits, unos 15 digitos decimales. La mantisa en la precisión extendida tiene 63 bits, es decir, se puede obtener una precisión de aproximadamente 18 decimales.

Para los programadores en C, una variable float es de simple precisión, mientras que un double tiene doble precisión y un long double suele tener precisión extendida (aunque no todos los compiladores soportan esta última opción).

18.6.2 – Continuando con el error

Para comprobar si un Pentium tiene el error, basta efectuar con cualquier software de calculadora u hoja de cálculo, la división de alguno de los pares de números afectados (véase ejemplo adjunto). Comparando los resultados con los reales, veremos diferencias a partir del cuarto dígito decimal. Sin embargo, es posible obtener un resultado más escandaloso al efectuar la operación x - (x/y) * y con los números afectados. Esta operación, como es fácil ver, resulta cero para cualquier x e y tomados; sin embargo, una operación efectuada en un Pentium con los x e y adecuados, devolverá un número distinto de 0. Si el resultado es erróneo, está claro que tenemos el ; si no lo es, deberemos asegurarnos de que el programa empleado utiliza las instrucciones FDIV y

similares para obtener el resultado, lo cual no es siempre seguro, aunque se sabe que lo es para la Calculadora de Windows y la hoja de cálculo Excel; por tanto el resultado de la prueba efectuada en estos programas es infalible con respecto al . Por otra parte, aunque se han desarrollado ya numerosas soluciones por software para corregirlo, no existen parches para actualizar los binarios de los programas que pudieran efectuar las operaciones prohibidas ; sólo cabe esperar que las respectivas casas de software introduzcan en el mercado una nueva versión con la leve corrección pertinente, y de momento utilizar dichos programas residentes para la corrección.
Si además de este método casero queremos información adicional a la simple verificación del error, podemos emplear el programa P87TEST.ZIP del antes mencionado Terje Mathisen, disponible en el G.U.I. Es muy de esperar que el resultado sea positivo, ya que sólo un ínfimo porcentaje de entre los primeros Pentium aparecidos carecían del error.

```
/*Este programa dará cero en cualquier procesador que no sea
Pentium, que dará 256.*/
main() double x,y,z;
x =4195835,0; y = 3145727.0;
z =x-(x/y)*y;
printf("Resultado: %f\n", z);
}
```

Intel, tras efectuar siete billones de divisiones aleatorias, indicó que el error se producía en una de cada nueve mil millones, partiendo del número de casos erróneos encontrados. El tipo de cálculos que realicemos afectará, sin duda, a la probabilidad de toparnos con un error, pero, con este dato, podríamos efectuar el siguiente razonamiento: Si ejecutamos una instrucción FDIV constantemente en un Pentium-90, al durar cada instrucción 39 ciclos (según documenta Intel) y disponer de un reloj de 90 MHz, esto es, 90 millones de ciclos, se efectuarían 2.307.692 divisiones cada segundo, y más de ocho mil millones cada hora. De esta forma, al cabo de poco más de una hora se obtendría un error. Naturalmente, estas circunstancias son imposibles. La estimación realista de Intel es la siguiente: realizando, con una hoja de cálculo, 1.000 divisiones de promedio al día, sólo obtendremos un error a los 27.000 años.

Pero sin tener en cuenta la frecuencia del error, lo importante aquí es la respuesta que Intel pueda dar a sus usuarios suspicaces. En esto Intel no puede ser alabada. Sus propios técnicos descubrieron el error ya en Junio, pero lo minimizaron, pensando que defectos los hay en todo procesador, aunque de mayor o menor importancia, y que éste no llegaría a salir a la luz, por lo que no creyó conveniente alertar a los casi dos millones de usuarios de aquel entonces (fue un grave error por su parte no tener en cuenta las leyes de Murphy). Después, a la vista de las circunstancias, trató de enmendarse, haciendo pública una descripción técnica del error y ofreciéndose a sustituir los chips erróneos: envió un fax a distribuidores de todo el mundo, cuyo contenido reproducimos en el recuadro adjunto, y que resulta cuanto menos poco serio para con los usuarios, pues indica que decide sustituir el procesador defectuoso debido a un error que sucede cada 27.000 años. Intel asegura haber reservado una partida importante de sus beneficios a lo largo del cuarto trimestre del año para la operación de sustitución, a la vez que afirma haber corregido ya el error en todos los nuevos Pentium s fabricados desde la última semana de Diciembre. A pesar de esto, los comercios (al menos en Valladolid) no están recibiendo Pentium s corregidos con regularidad, dado que a pesar de lo que se fabrique en Taiwan, lo que traiga el distribuidor a España puede ser bien diferente.

La trascendencia del error, a pesar de su difusión a través de Internet, no hubiera llegado a tanto de no haberse producido la reacción de IBM. El gigante azul sigue siendo el vendedor más importante de PCs, y por ello el hecho de que decidiera, a mediados de diciembre, retirar sus equipos con Pentium del mercado, supuso un fuerte impacto a nivel empresarial. IBM adujo que su División de Investigación, una vez al corriente del fallo, indicó que, aún siendo ciertos los datos técnicos sobre la naturaleza del error, las situaciones de riesgo podían ser mucho más numerosas que las indicadas por Intel. De hecho, según IBM, las estimaciones de Intel fallaban en sus bases, ya que una hoja de cálculo puede efectuar 5.000 divisiones cada segundo. Trabajando una media de 15 minutos diarios en una hoja de cálculo, ello supondría un error cada 24 días, o lo que es igual, a diario en una empresa con 24 ordenadores. Pero IBM no convenció al resto de fabricantes, que se han agrupado en torno a Intel, restando importancia al error y confiando en la marca. Todos

afirman haber continuado con normalidad sus ventas tras explicar la situación a sus clientes, e incluso algunas marcas dicen haber rebasado sus expectativas sobre el Pentium.

Hay realmente segundas intenciones en la decisión de IBM? El hecho innegable es que IBM está inmersa en la estrategia PowerPC. Para quienes no tengan muy claro de qué se trata, diré rápidamente que PowerPC significa Performance Optimized With Enhanced Risc-Personal Computer (está claro que este tipo de acrónimos son del agrado de todos los anglosajones), y se trata de una nueva familia de microprocesadores, no compatibles con los de Intel, que está siendo desarrollada por IBM, Apple y Motorola, y que Apple e IBM incluirán en equipos que, hasta la fecha, no son compatibles en software. Apple ya ha comercializado equipos PowerMac, además de ofrecer a todos los usuarios de los procesadores 680x0 de Motorola una fácil actualización a PowerPC. Una de sus características más importantes es que ofrecen una computación RISC (Reduced Instruction Set Computing) frente a la CISC (Complex Instruction Set Computing) de los equipos Intel hasta el 80486, siendo Pentium una solución intermedia entre ambos métodos..

FUNCIONES ESPECIALES, ENTORNOS DE TRABAJO Y COMPATIBILIDAD

19

19.1 CAPACIDAD DE AUTOCOMPROBACIÓN.

El Pentium es capaz de auto chequearse hasta el 70% del silicio que posee, comprendiendo toda la memoria ROM de control y la mayoría de la lógica no aleatoria. Este auto chequeo se produce mediante la operación BIST (Built-In Self-Test) del Pentium. Para que esta operación pueda ejecutarse se debe conectar:

- La alimentación.
- La señal de entrada Reset debe subir a nivel alto.
- La patita Busy # debe de conectarse a Tierra.

El auto chequeo es un programa de autocomprobación cuyo tiempo de ejecución varía según el tipo de procesador.

Tras ejecutarse dicho programa deja en EAX un código binario que se trata del resultado del auto chequeo. Si todos los bits son 0 entonces la operación ha sido ejecutada con éxito, si por el contrario se encuentra un valor distinto de cero en el registro significa que se ha dado un fallo en el procesador y hay que acudir a las características técnicas del fabricante para saber que anomalía se ha producido.

En EDX queda depositada una información sobre las características técnicas del Pentium que se ha chequeado, así como su número de versión.

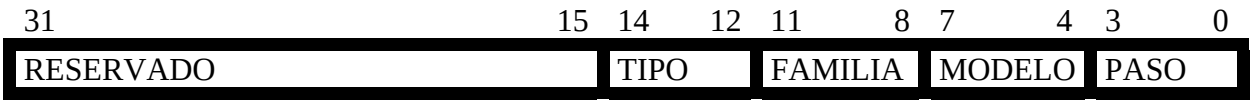


Figura 19.1-Formato del registro EDX

19.1.1 Comprobación de la TLB

La TLB es una memoria tipo CAM (acceso por contenido) que está realizada con una memoria caché ultrarrápida, dónde hay etiquetas y datos. En la etiqueta pone una dirección lineal y en los datos la dirección física. La TLB guarda las 256 últimas traducciones de las últimas páginas utilizadas.

La arquitectura interna del Pentium dispone de 2 registros TR6 y TR7 mediante los cuales se puede leer y escribir cualquier posición de la TLB.

Para realizar el auto chequeo se precisa un programa en lenguaje Ensamblador, que controla un mapa de pruebas. También es recomendable desactivar la paginación para impedir que los resultados de las pruebas eliminen a los datos obtenidos del manejo de la memoria con la paginación.

TR6 actúa como registro de comandos para el control de las pruebas, mientras que TR7 es el que se carga con los resultados de las pruebas, una vez efectuadas.

19.2 SOPORTE A LA DEPURACIÓN

La depuración es la operación que se realiza para comprobar y ajustar el funcionamiento, tanto del equipo físico como del sistema lógico.

El Pentium dispone de tres recursos eficaces para el soporte de la depuración:

- **Punto de Parada por Software**: Se consigue detener el procesamiento de un programa en un punto determinado del mismo, introduciendo allí una instrucción de interrupción. Se trata de INT3, que está codificada en un sólo byte y su ejecución origina interrupción, que es atendida por la entrada 3 de la IDT. Allí, el programador del sistema puede situar la referencia a un programa que visualice los elementos deseados de la arquitectura del procesador y del estado del sistema.

- **Excepción paso a paso:** Cuando se habilita el señalizador TF del registro de EFLAGS (TF=1), cada vez que se ejecuta una instrucción se produce una excepción que es atendida por la entrada 1 de la IDT. La excepción pasa a paso almacena el registro de señalizadores en la pila, con el bit TF=1. Después, se desactiva TF para poder procesar con normalidad la rutina de servicio a dicha excepción, que al concluir con la instrucción IRET, se vuelve a colocar (desapilar) el registro de señalizadores como estaba, con TF=1, transfiriéndose el control a la siguiente instrucción del programa principal.

- **Registros de Depuración (DR0-DR7):** Mediante estos registros se pueden programar cuatro puntos de parada independientes. Los registros DR0-DR3 contienen la dirección lineal de los cuatro posibles puntos de parada. DR7 sirve para especificar la condición que se desea detectar par originar la parada del procesamiento. También se puede determinar la granularidad, así como enmascarar cada condición por separado y definir la entrada correspondiente: local para una tarea, o bien, global para todas las que hay en el sistema. DR6 guarda la información necesaria para conocer la causa que ha provocado una parada, al detectarse una condición de punto de parada.

19.3 EL RESET Ó LA INICIALIZACIÓN

Manteniendo activa la patilla RSET durante un cierto tiempo que viene determinado y que no es el mismo para cada Pentium, el procesador del sistema comienza un hardware de inicialización del procesador (conocido como RESET del hardware) y una autocomprobación opcional llamada BIST.

Cuando se activa la patita Reset se deja a los registros en un estado conocido y coloca al procesador en modo Real. Borra y limpia la caché interna, la TLB y también la BTB(otra caché especial en la que se guarda el resultado de los últimos 256 saltos condicionales, y que se utiliza para prever el resultado de un salto condicional. En un 97% de los casos acierta, si hay fallo hay que retirar las instrucciones que hemos metido en el cauce y limpiar los registros).

La activación de la patilla INIT# en el procesador tiene un efecto similar al RESET del hardware. La diferencia está en que durante un INIT# los estados de las cachés internas y e FPU permanecen inalterados. La activación del INIT# permite cambiar de modo protegido a modo real manteniendo el contenido de las cahés internas.

El estado de los registros es el siguiente:

EFLAGS	0000 0002 hex	
EIP	0000 FFF0 hex	
CS	Base	FFFF 0000
	Límite	FFFF
EAX=EBX=ECX=EDX=EDI=ESI=EBP=ESP	0	
CR0	6000 0010 hex	

1ª Conclusión: Si el Registro CR0 tiene un 0 en el bit de menos peso , el bit PE se encontraría a 0 por lo que el Pentium siempre comenzaría trabajando en modo real . Por otro lado también es 0 el bit de mas peso , que se trata del bit PG de manera que al comenzar a trabajar el Pentium lo hace con la paginación desactivada.
(Dibujo del registro cr0)

2ª Conclusión: En modo Real la Memoria Principal sólo utiliza el primer Mbyte de la memoria física. Por lo que en Modo Real las 12 líneas de menos peso no son utilizadas, sin embargo la primera dirección que vaya después del Reset entra por las 32 líneas.

(Dibujo de la Memoria Principal)

La primera instrucción que se ejecuta después de un Reset se hallará sumando la base más el EIP de modo que queda la dirección física : FFFF FFF0 en hexadecimal.

Base:	FFFF 0000
EIP :	FFF0
Dirección física:	FFFF FFF0

La 1ª dirección no está dentro del mapa de memoria que le corresponde al modo Real por lo que en esa posición hay una instrucción de salto a la 000F FFF0 que se trata de la 1ª dirección real de nuestro programa.

3ª Conclusión: Nada mas empezar a trabajar se puede generar una interrupción ó una excepción, por lo tanto antes de empezar a trabajar la IDT tiene que estar residente en memoria principal. Por defecto, la dirección física de la IDT es 0000 0000 hexadecimal.

19.4 MODO REAL

Este modo de operación implementa el entorno de programación del procesador Intel 8086 , imprimiéndole mas velocidad a la hora de ejecutar programas.

En Modo Real se empieza a trabajar siempre que hay un Reset ó cuando se conecta por 1ª vez la alimentación.

Características del entorno de ejecución del Modo Real:

1-La Memoria Principal sólo es de 1Mbyte. Este espacio está dividido en segmentos , cada uno de los cuales puede tener una longitud de 64 Kbytes .

2-El bus de direcciones sólo usa las 20 líneas de menos peso.

3-Es monotarea .

4-Los 8 registros de propósito general(AX, BX, CX , DX, SI, DI, SP y BP) son de 16 bits.

5-El puntero de instrucciones (IP) del 8086 está manejado en los 16 bits de menos peso del registro EIP.

6-El registro de 16 bits FLAGS contiene los flags de control y estado. (Este registro está mapeado en los 16 bits menos significantes del registro de 32 bits EFLAGS).

7-La tabla IDT es de 256 entradas, con entradas de 8 bytes y en cada una hay un vector de interrupciones que apunta a la tabal de interrupciones que tiene entradas de 4 bytes.

(Dibujo de la IDT en Modo Real)

19.5 MODO PROTEGIDO

Es el modo natural que tiene de trabajar el Pentium, los registros tienen ya una extensión de 32 bits, se trabaja con multitarea, se puede poner en marcha la paginación y funciona el modo de protección.

Estando en Modo Real para pasar a Modo Protegido se pone el flag PE del CR0 a 1, que habilita el mecanismo de protección de segmentos, pasando automáticamente a Modo Protegido.

Al pasar a Modo Protegido hay unas cuántas cosas que hay que tener en cuenta:

- IF=1 (Interrupciones mascarables prohibidas).

- La IDT en Modo Protegido es completamente distinta de la IDT en Modo Real. La IDT del Modo Protegido no contiene vector de interrupciones , contiene descriptor de Interrupciones. La IDT en Modo Protegido es completamente distinta en tamaño y contenido de la IDT en Modo Real. La IDT cuando entramos en Modo Protegido tiene que estar ya cargada, de manera que justo antes de entrar en Modo Protegido tenemos que crear la IDT en memoria Principal. La IDT debe estar situada en el primer kilobyte de la memoria física.

- Estando aún en Modo Real, hay que construir una imagen básica de la GDT e inicializar el registro GDTR.

Hay que inicializar el segmento de pila, direccionando una zona de RAM. Simplemente, será preciso cargar los valores SS y ESP con los correspondientes a la zona deseada, lo cual se hace a base de instrucciones MOV.

Entonces hay una serie de pasos que hay que dar antes de pasar a Modo Protegido:

1º Deshabilitar las interrupciones . Una instrucción CLI (IF=0) deshabilita las interrupciones mascarables.

2º Instalar en memoria la GDT y apuntarla mediante LGDTR (instrucción que nos apunta a la GDT.

3º Ejecutar la instrucción MOV CR0 que pone el flag Pe a1 (Meter un 1 en CR0, por lo que cargo en EAX 0000 0001 hex y de ahí movemos a CR0)

4º Hacer una instrucción Jump. Cuando estamos en Modo Real hay un elemento que se denomina Cola de Prebúsqueda que tiene cargados códigos de Instrucciones de Modo Real, por lo que antes de pasar a Modo Protegido hay que limpiar la pila. Esto se hace con la Jump que elimina las instrucciones que quedan por detrás de la posición a la que salto.

5º Ejecutar la instrucción LLDT (Load LDT). Carga en LDTR la dirección de la LDT de la tarea que hay en curso.

6º Ejecutar la instrucción LTR. Carga el registro de tareas con un selector de segmentos a la tarea inicial del modo protegido o a un área escribible de memoria que pueda ser usada para cargar la información del TSS en un cambio de tarea.

7º Actualizar los registros de segmentos para que apunten a los segmentos de Modo Protegido.

8º Ejecutar la instrucción LIDT para cargar el registro IDTR con la dirección y el límite del IDT del modo Protegido.

9º Ejecutar la instrucción STI (Permitimos las interrupciones mascarables. Hasta ahora no hemos permitido las interrupciones de los periféricos.)

19.6 MODO VIRTUAL 86

Se trata de un modo que es mezcla del Modo Real y del Modo Protegido. En el Modo Protegido, el procesador puede trabajar en un ambiente conocido como Modo Virtual_86 o modo de tarea especial. Este modo permite al procesador ejecutar software del 8086 en un entorno protegido y multitarea.

Cuando el sistema operativo cambia a una tarea en Modo Virtual_86, el procesador simula un procesador 8086 donde el entorno de ejecución es igual al Modo Real con la diferencia de que este modo puede utilizar servicios del Modo Protegido.

Un aspecto a tener en cuenta es que el Modo Virtual_86 puede ejecutar algunas tareas que son del Modo Real y como sabemos este modo trabaja con direcciones de 16 bits, mientras que el Modo Virtual_86 trabaja con direcciones de 32 bits. Por esto el Modo Virtual_86 tendrá que convertir las direcciones de 16 bits a 32 bits.

Para entrar en este modo, el procesador tiene que estar dentro de cualquiera de estas situaciones:

- Cuando el flag VM del registro E_FLAGS es puesto a 1 en la TSS de la tarea.
- Cuando se retorna de una interrupción o excepción en modo protegido, cuando el flag VM es puesto a 1 en el registro EFLAGS

El procesador sólo podrá abandonar el Modo Virtual_86 mediante una interrupción o una excepción.

El procesador sólo testeará el flag VM en los siguientes casos:

- Cuando se cargan los registros del segmento, para determinar si hay que usar la traducción de dirección en estilo 8086.
- Cuando decodifica las instrucciones para determinar qué instrucciones son las no válidas en Modo Virtual_86.

Al comprobar instrucciones privilegiadas, en accesos a páginas, o cuando se realiza cualquier otra comprobación de permiso.

19.7 MODO MANEJO DEL SISTEMA (SSM)

En este modo, el procesador proporciona un sistema operativo que es transparente para el programador y que implementa dos funciones muy importantes:

- Mejora de la seguridad de todo el sistema.
- Un sistema de control de la alimentación. Mediante este sistema controla el consumo de energía y realiza un consumo lo más óptimo posible.

Cuando se pasa a Modo SMM, el procesador aísla completamente un espacio de memoria reservado para él donde se conmuta mientras guarda todo el contexto de la tarea que va a ejecutar. Es entonces cuando ejecuta el código específico del SMM. Cuando se retorna del modo SMM, el procesador vuelve a su estado anterior a la interrupción del manejo del sistema.

Para pasar de cualquiera de los modos a modo SMM, habrá que activar por hardware una patita del Pentium. Esa patita es la SMI, que se activa por nivel bajo (SMI#). La interrupción que provoca esta patita tiene mayor prioridad que las interrupciones externas o las excepciones de depuración. Lo primero que realizara el procesador será salvar el estado en el que se encuentra el procesador y cambiar a un entorno operativo reservado por él y contenido en l a RAM de manejo de sistema.

Mientras el procesador esté en este modo, ejecutará el código del manipulador del SMI para realizar ciertas operaciones como desactivar unidades de disco o monitores, poniendo todo el sistema en un modo de reposo. Cuando el manipulador SMI a completado sus operaciones, ejecuta una instrucción de salida (RSM) que hace que el procesador cargue de nuevo el contexto de la tarea que estaba ejecutando antes de iniciar este modo y por último vuelve al Modo Real o al Modo Protegido dejando de ejecutar la aplicación de interrupción o el programa del sistema operativo.

Para pasar a SMM a los otros tres modos existentes, existen tres formas:

- RSM + PE = 0→ Modo Real.
- RSM + PE = 0→ Modo Protegido.
- RSM + PE = 1 + VM = 1 → Modo Virtual 86.

El modo SMM es similar al Modo Real en que no hay niveles de privilegio ni mapas de direcciones. Un programa en modo SMM puede tener hasta 4Gbytes de memoria y puede ejecutar todas las instrucciones del sistema así como todas las instrucciones de E/S.

PENTIUM-PRO 20

20.1- CARACTERÍSTICAS GENERALES.

20.1.1- Introducción.

La aparición, a finales de marzo de 1995, del procesador Pentium Pro supuso para los servidores de red y las estaciones de trabajo un importante avance, tal y como ocurriera con el Pentium en el ámbito doméstico. Este nuevo dispositivo constituía la sexta generación de procesadores de la familia x86.

La potencia de este procesador había aumentado de forma notable, gracias a la arquitectura de 64 bits y el empleo de una tecnología revolucionaria como es la de 0,6 micras, lo que permitía la inclusión de cinco millones y medio de transistores en su interior. Para la fabricación de dicho procesador se había usado la tecnología BICMOS de 4 niveles.

Una de las novedades que aportaba el pentium pro era su aspecto físico, que se presentaba en un encapsulado con 387 patitas, bastante grande debido a que incorpora dos chips: uno de ellos contiene el procesador y el otro una caché de segundo nivel, que se encargaba de mejorar la velocidad de la memoria caché, lo que resultaba en un incremento del rendimiento sustancioso. Dicha caché se comunicaba con la CPU a la misma velocidad del procesador. Las frecuencias de reloj se mantenían como límite superior en 200 MHz, partiendo de un mínimo de 150 MHz.

En la arquitectura de este procesador se encuentra un bus de direcciones de 36 líneas, por lo que es capaz de direccionar hasta 64 Gigabytes.

Como desventajas tecnológicas, es reseñable el alto precio de fabricación del conjunto multichip. La microarquitectura utilizada está optimizada para software de 32 bits, que tienen rendimiento pobre con código de 16 bits. Se aprecia un consumo de energía y disipación de calor totalmente inapropiadas para ordenadores portátiles

Todas estas características se encuentran resumidas en la tabla 20.1. que corresponden al primer modelo que fue comercializado.

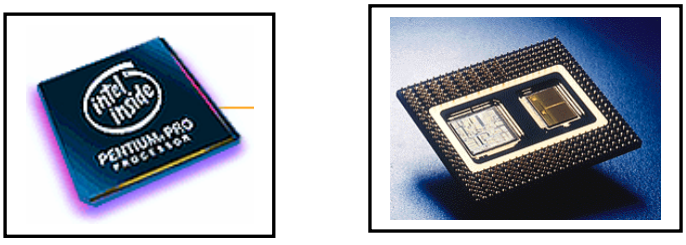


Figura 20.1 . Fotografía de la cápsula del Pentium-Pro

	CARACTERÍSTICAS GENERALES
AÑO	1995
MILLONES DE TRANSISTORES	5,5 (CPU) 15,5 (L2)
TECNOLOGÍA (MICRÓMETROS)	0,6
FRECUENCIA (MHZ)	150
PRECIO ($)	974
RENDIMIENTO	220 $SPEC_{INT92}$
CAPSULA	PGA 273
ALIMENTACIÓN (VOLTIOS)	2,9
POTENCIA (WATIOS)	20
CACHE L1 (KBYTES)	8 (INSTRUCCIONES) 8 (DATOS)
CACHE L2	256 Kbytes(INTERNA)

Tabla 20.1. Resumen Características

En el Pentium Pro los siguientes puntos de su arquitectura, son de vital relevancia:

1. Integración total de una caché de segundo nivel con el procesador.
2. Potenciación del paralelismo mediante la ejecución dinámica de las instrucciones, es decir, fuera de orden.
3. Potenciación del paralelismo mediante la incorporación de más unidades de ejecución que trabajen simultáneamente, lo que recibe el nombre de superescalar.
4. Adaptación a la línea de los procesadores RISC, intentando traducir las instrucciones complejas de la familia x86, con un formato muy irregular y operandos inmediatos de longitud variable, a microprocesadores RISC con simetría absoluta y posibilidad de ejecución fuera de orden.
5. Incremento de la velocidad mediante la disminución del ciclo de reloj, aumentando así el número de etapas del cauce, que se eleva a 14 y da lugar a la supersegmentación.

20.1.1- Arquitectura Interna Básica.

El microprocesador del Pentium Pro está constituido principalmente por los siguientes elementos principales:

1. **Unidad de Bus Externo:** Realiza transacciones de bus cuando es requerido para ello por la caché L2 o núcleo de microprocesador.
2. **Unidad de Bus Trasero:** Interfaz entre el núcleo y la caché L2.
3. **Caché L2 Unificada:** Actúa cuando la caché L1 falla, tanto en datos como en código. Cuando sea necesario hará peticiones a la Unidad de Bus Externo.
4. **Caché de Datos L1:** Da servicio a las peticiones de carga y almacenamiento hechas por las unidades de carga y almacenamiento. Cuando no pueda dar servicio, pasará la petición a la caché L2.
5. **Caché de Código L1:** Da servicio a las peticiones de búsqueda de instrucciones formuladas por el prebuscador de instrucciones.
6. **Microprocesador:** Responsable de las siguientes operaciones primordiales:
    - Búsqueda de instrucciones.
    - Predicción de saltos.
    - Análisis del flujo de instrucciones.
    - Traducción de las instrucciones CISC a RISC.
    - Despacho, ejecución y retirada de microoperaciones.

7. **Unidad Local Apic:** Responsable de recibir las peticiones de interrupción de otros procesadores, de los pines locales de interrupción , del temporizador APIC. Estas peticiones prioritarias, son enviadas al microprocesador para su ejecución.

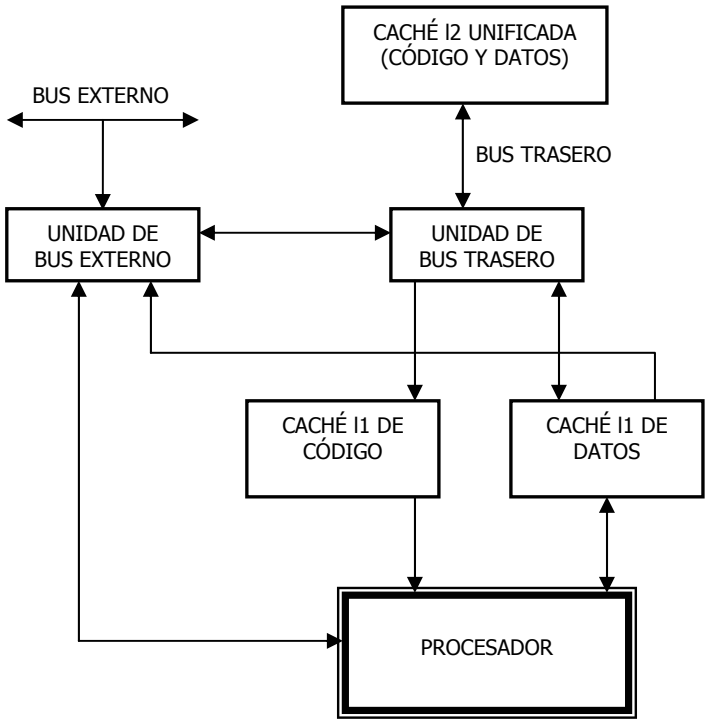


Figura 20.2. Diagrama simplificado de bloques del Pentium Pro

20.2- APORTACIONES Y NUEVOS RECURSOS ARQUITECTÓNICOS.

20.2.1- Registros de Propósito General

Los registros de los anteriores microprocesadores x86 eran demasiado pequeños. Esta restricción solo permitía al microprocesador mantener un pequeño numero de datos fáciles de acceder por las unidades de ejecución. A menudo, el programador se veía obligado a guardar en la memoria los contenidos de uno o más registros, cuando necesitaba leer nuevos operandos para operar con ellos. Más adelante puede necesitar el registro original, que guardó previamente en la memoria, por lo que tendrá que volver a leerlo de la memoria, lo cual es una perdida de tiempo e introduce una penalización en el rendimiento del programa.

El Pentium Pro soluciona este problema con 40 registros nuevos. Las instrucciones son traducidas a microoperaciones antes de ser ejecutadas. Cuando se ejecutan se pueden dar las siguientes circunstancias.

1. La microoperación tiene que colocar un valor en unos de los registros de propósito general.
2. La microoperación tiene que leer un valor que fue colocado en el registro por otra instrucción ejecutada con anterioridad.
3. La microoperación cuando es ejecutada, cambia los contenidos del registro EFLAGS.

Menos mal que el microprocesador permite la ejecución fuera de orden, ya que si el resultado de la ejecución fuera inmediatamente reflejado en el set de registros del microprocesador, los valores en los registros se cambiarían, los bits de condición de condición de los registros de estado serían actualizados, probablemente de forma incorrecta y en un orden inadecuado.

En vez de una carga inmediata de los resultados de la instrucción en los registros reales, el microprocesador almacena el resultado. Si una vez ejecutada, otra microoperación requiere el resultado producido por las microoperaciones que le preceden, los resultados son enviados a la microoperación peticionaria. A esto se le conoce como Feed Forwarding.

Esta técnica de almacenamiento de registros es necesaria para pode realizar la ejecución fuera de orden, y es conocida también como Aliasing de Registros.

20.2.1.1- Eliminación de Dependencia de Falsos Registros.

Si examinamos el ejemplo que tenemos a continuación:

EJEMPLO 1

Supongamos que se está ejecutando el siguiente programa:

```
mov eax,17
add mem,eax
mov eax,3
add eax,ebx
```

En el antiguo microoprocesador x86, estas instrucciones hubieran tenido que ser ejecutadas de una en una para proporcionar el resultado correcto. El microprocesador no podría ejecutar las instrucciones 3ª y 4ª antes de haber ejecutado la 1ª la 2ª. La 2ª instrucción debe usar el valor de EAX antes de que la 3ª instrucción coloque un nuevo valor en EAX.

En el Pentium Pro el microprocesador es capaz de darse cuenta de que el registro EAX no tiene porque ser cargado con el valor 17. Se puede obtener el mismo resultado sumando 17 a la dirección de memoria. Del mismo modo, EAX no necesita ser cargado con el valor 3. El valor 3 se puede sumar a EBX y el resultado se carga en EAX. El microprocesador pude ejecutar de forma simultanea las instrucciones 1ª y 3ª, disponiendo de los valores 17 y 3 para cada una de las microoperaciones que lo necesiten. De igual manera las instrucciones 2 y 4 pueden ser ejecutadas en paralelo, utilizando los valores 17 y 3 asociados a las instrucciones 1 y 3.

20.2.2- Caché L2 y L1

La caché interna de primer nivel L1 del Pentium Pro dispone de una zona de 8kb destinada a contener instrucciones y otra del mismo tamaño para contener datos. La caché L1 de instrucciones es del tipo de asociativa de dos vías y la de datos es de 4 vías.

A un procesador que pase de los 100Mhz, no le basta con una caché L1 de capacidad reducida, ya que el porcentaje de fallos es lo suficientemente importante como para que afecte al rendimiento de la CPU al tener que acceder al la Memoria Principal DRAM (dinamic RAM). Se hace imprescindible potenciar la caché SRAM (static RAM), lo que motivó a INTEL a integrar junto con

el procesador una caché de segundo nivel de capacidad notable, con la que se redujesen drásticamente los accesos a la DRAM.

La caché de segundo nivel L2 del Pentium Pro tiene un tamaño de 256 kb y una estructura asociativa de 4 vías, siendo lo más importante de esta memoria la conexión mediante el bus trasero con el procesador caracterizándose por funcionar a la misma frecuencia que el procesador pudiendo transferir 64 bits por el bus de datos en un ciclo de reloj.
El procesador se comunica con la Memoria Principal con los módulos de E/S mediante otro bus, llamado frontal (figura 20.2.).

El procesador se empaqueto conjuntamente con la caché L2 por dos razones:

1. Construcción de sistemas de altas prestaciones a los fabricantes de equipos.
2. Mejorar el interfaz directo entre el procesador y la caché L2.

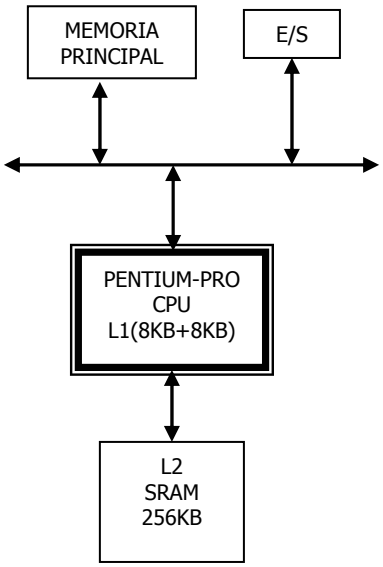


Figura 20.3. Arquitectura de la caché L2 en el Pentium-Pro

Cuando la CPU no encuentra las instrucciones ó datos en las caché L1 pasa a consultar la caché L2, lo que supone un retraso en el ciclo de reloj. Si tampoco se encuentra lo que se busca en la caché de 2º nivel se pasa a buscar en la Memoria Principal, lo que significa una gran penalización de tiempo y una degradación de rendimiento.

Para mejorar lo anteriormente comentado el Pentium Pro opera de forma independiente y simultanea con el bus frontal y trasero. Mientras se espera que se complete un acceso a la memoria, se puede iniciar otro. Para realizar dicho fin existe un Buffer de Ordenamiento de Accesos a Memoria (MOB), que guarda hasta 8 accesos o "transacciones", pudiendo iniciarse un acceso aunque existan 8 pendientes de realizarse (figura 20.3.)

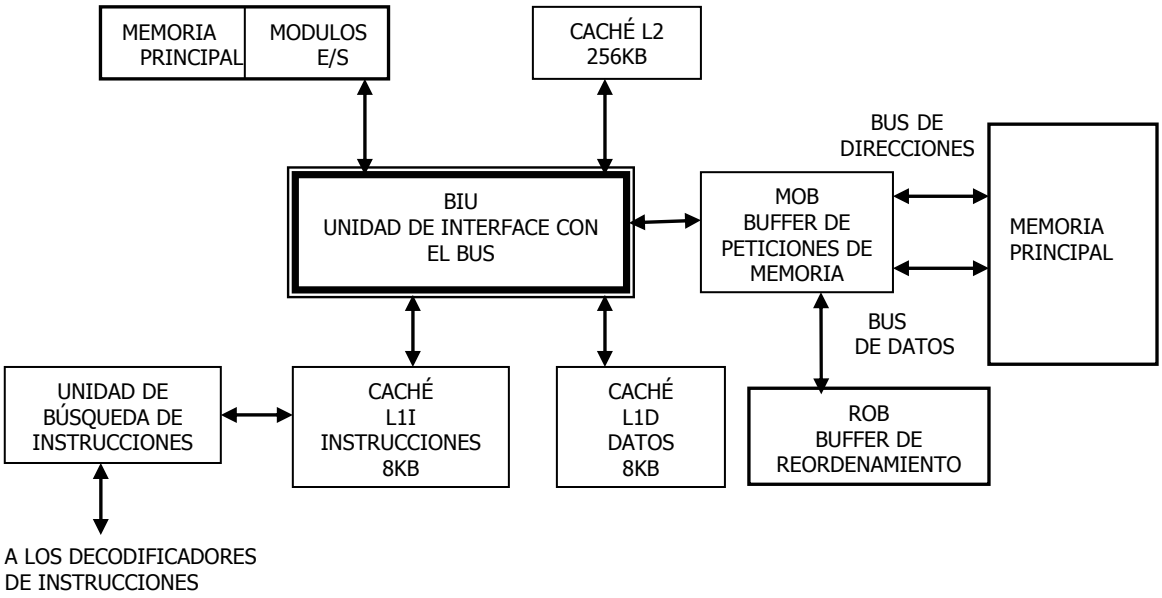


Figura 20.4 Arquitectura de la unidad de interface con el bus (BIU).

El MOB actúa como depósito de reserva y regulación, que permite a determinados accesos de lectura de memoria anticiparse a otras lecturas y escrituras, consiguiendo así elevar notablemente el rendimiento del sistema. El MOB actúa en coordinación con el ROB (Buffer de reordenamiento) que almacena las peticiones de lectura y escritura pendientes de ejecutarse hasta que desaparecen las causas que las impiden, bien procedan de una dependencia de datos o bien de la utilización de un recurso necesario.

Para evitar problemas entre las caches y la Memoria Principal, este procesador, al igual que sucedía con el 486, emplea el protocolo MESI, qué está diseñado para evitar los errores generados cuando se usa código automodificable, típico en algunos programas de la familia x86.

20.2.3- Aplicación de la técnica RISC

Una de las características que INTEL arrastra desde sus comienzos, es que el software de los nuevos procesadores, es compatible con el software de todas las versiones anteriores. Por lo que el set de instrucciones de la familia x86 de tipo CISC (Computadores de juego de instrucciones complejo).

Muchas de estas instrucciones son complejas, requiriendo una extensiva decodificación y lógica de ejecución. Algunas de ellas necesitan un numero de recursos elevados para completarse. Una instrucción puede suponer, una o más lecturas de memoria, una o más o escrituras de memoria. Además de estos inconvenientes, una instrucción puede ser desde 1 hasta 15 bytes de longitud. Lo cual es un gran obstáculo con respecto a la ejecución de más de una instrucción simultáneamente. El microprocesador tiene que saber donde acaba una instrucción y donde comienza la siguiente. Solamente entonces la lógica del procesador decodifica la instrucción y determina si pueden ser o no ejecutadas al mismo tiempo.

Debido a todo esto, los fabricantes de los procesadores se deciden por la técnica Risc (computadores de juego de instrucciones reducido), que manejan un grupo pequeño de instrucciones muy sencillas que se ejecutan por hardware en un solo ciclo de reloj.

La conversión de instrucciones Cisc a microprocesadores Risc se lleva a cabo en el Pentium Pro en la etapa **Unidad de decodificación** que consta de 4 bloques de los cuales dos son decodificadores básicos, encargados de la mayoría de las instrucciones.

Las instrucciones son prebuscadas en la memoria y almacenadas en la caché L2 y en la caché de código L1. Como de la caché se obtienen bloques de instrucciones, el núcleo del microprocesador los analiza para distinguir los limites entre las instrucciones. Entonces decodifica las instrucciones, variables en longitud, en instrucciones RISC, de longitud fija, llamadas microoperaciones. Estas se almacenan y esperan a ser despachadas y ejecutadas.

Se puede obtener hasta un máximo de seis microoperaciones por ciclo de reloj, dado que los decodificadores trabajan de forma independiente y en paralelo.

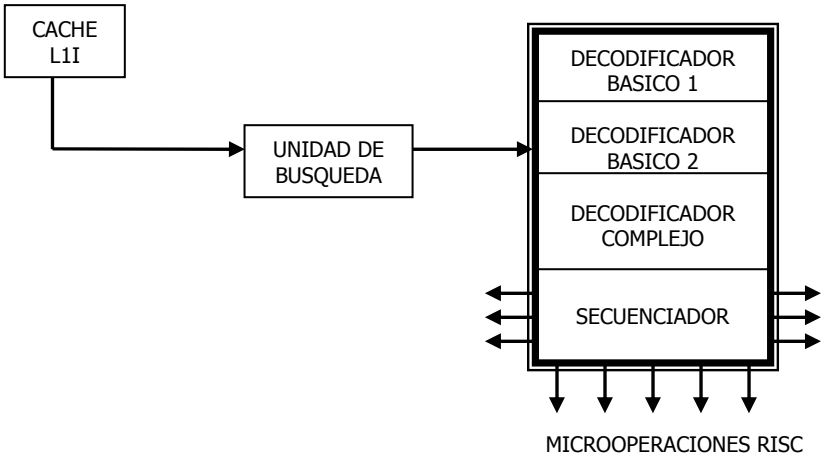


Figura 20.5 Los decodificadores que convierten las instrucciones de la familia x86 en microoperaciones RISC.

20.2.4- Supersegmentación

Se dice que el cauce de procesamiento de instrucciones del Pentium Pro es supersegmentado dado que consta de 14 etapas y cuantas más etapas tenga el cauce más elementales son las funciones que realiza. Sin embargo la existencia de tantas etapas complica los riesgos debidos a las dependencias de datos y a las bifurcaciones condicionales.

La primera etapa del cauce: calcula el valor que hay que cargar en el contador de programa (EIP) para apuntar a la dirección de la siguiente instrucción. Como tenemos que tener en cuenta las instrucciones de bifurcación, se dispone de una memoria CAM de 512 entradas, llamada BTB (Buffer de Destinos de Bifurcación), que almacena las direcciones que se han utilizado en las bifurcaciones anteriormente.

Las penalizaciones son bastante significativas, oscilando entre 4 y 5 ciclos de reloj, para mejorar los aciertos se emplean los métodos clásicos de predicción: estático y dinámico.

El método estático para predecir si existe ó no bifurcación se ayuda de un compilador que analiza el comportamiento del programa, sin embargo el comportamiento dinámico evalúa las ramificaciones. Para ello la BTB dispone de bits de historia de la ramificación a dos niveles: a nivel de ramificación individual y a nivel de grupo de ramificaciones.

Las tres siguientes etapas del cauce se destinan a la búsqueda de las instrucciones. Estas proceden de la caché L1 de Instrucciones, de donde son leídas a razón de 2 líneas de 32 bytes cada una por ciclo de reloj. La delimitación de las instrucciones x86 es necesaria al tener un formato variable comprendido entre 8 y 120 bits. Para delimitar las instrucciones se toman los 16 bytes al que apunta el EIP, los cuales se introducen a uno de los decodificadores. Es posible que el byte apuntado por el EIP se encuentre al final de una línea; por dicho motivo se leen dos líneas, evitando tener que realizar otro acceso para completar los 16 bytes que delimitan cada instrucción.

Una vez se hayan delimitadas las instrucciones x86 se introducen al decodificador de instrucciones que transforma los códigos binarios en secuencias de microprocesadores Risc.

La séptima etapa se encarga de realizar un renombrado de los registros de propósito general que emplean las instrucciones para los numero enteros y de coma flotante. 8 de estos registros se utilizan para número enteros y otros 8 para números en coma flotante. La existencia de pocos registros da lugar a muchos problemas de dependencias de datos, ya que todas las instrucciones se disputan los pocos registros disponibles. Como hay que mantener la compatibilidad y admitir dichos registros x86, el Pentium Pro dispone en esta etapa de una Tabla de Alias de Registros (RAT), que permite representar a cualquiera de los registros que emplean las instrucciones por medio de un registro físico del banco de registros ampliado. Esta ampliación alcanza a 40 registros implementados en el Buffer de Reordenamiento.

La octava etapa, se encuentra el ROB (Buffer de Reordenamiento), uno de los elementos clave en la estructura de este microprocesador. Se trata de una memoria CAM con 40 entradas de 254 bits que puede contener cada una una microoperación, los operandos que maneja y los bits de estado que señaliza. En esta etapa interviene la Unidad de Predicción de destinos de las bifurcaciones (BTB). Un fallo de la BTB significa una penalización promedio de 8 ciclos, necesarios para la recuperación y recomposición de todos los registros que han afectado a la predicción errónea.

Tras el ROB se encuentra (RS) que es la estación de reserva. Tiene como objetivo planificar el orden con que se van entregando las microoperaciones a las Unidades de Ejecución, que comprenden las etapas 10ª y 11ª. Admite hasta 20 microoperaciones.

20.2.5- Arquitectura superescalar

Este concepto corresponde a la existencia de 5 unidades de ejecución las cuales trabajan de forma simultanea é independiente, lo que significa que se puede alcanzar un flujo máximo de hasta 5 microoperaciones ejecutadas por cada ciclo de reloj. Habitualmente se reduce a 3 por ciclo de reloj.

La ejecución dinámica permite la ejecución fuera de orden y con varias microoperaciones en paralelo. Es posible gracias a la intervención de la ROB y de la RS. Dos de las unidades de ejecución (IU1 e IU2) trabajan sobre microoperaciones que manejan número enteros, otras dos (AGU1 y AGU2) calculan las direcciones de los operandos y las instrucciones de carga/almacenamiento y, además, hay una quinta (FPU) que trata las instrucciones con números en coma flotante. (figura 20.5).

En el caso que los bits de estado de una determinada microoperación no detecten dependencias, si se dispone de los operandos y existe una Unidad de Ejecución libre, se lanza dicha microoperación a la fase de ejecución.

Por la compleja red de interconexión y realimentación, los resultados producidos sobre las microoperaciones por las unidades de ejecución se devuelven a la estación de reserva y al ROB. De esta forma, la Estación de Reserva puede introducir los resultados de algunas microoperaciones en otra Unidad de Ejecución que los puede necesitar como operandos, evitando retrasos. Con los resultados recibidos en la ROB se determina cuando se ha completado una microoperación y está lista para ser retirada. La máxima velocidad de retirada de microoperaciones es de 3 ciclos de reloj.

Ejecutada la microoperación, se actualizan los bits de estado y se rescribe en el ROB para dar información a la estación de reserva y a la unidad de Extracción o Estación de retiro, que almacena los resultados en el Banco de Registros de Retiro (RRF) cuando dicha microoperación se halle completamente terminada. La Unidad de Extracción puede tratar en cada ciclo de reloj hasta la terminación de 3 microoperaciones, por lo que se dice que el Pentium Pro es un "procesador superescalar de 3 vías".

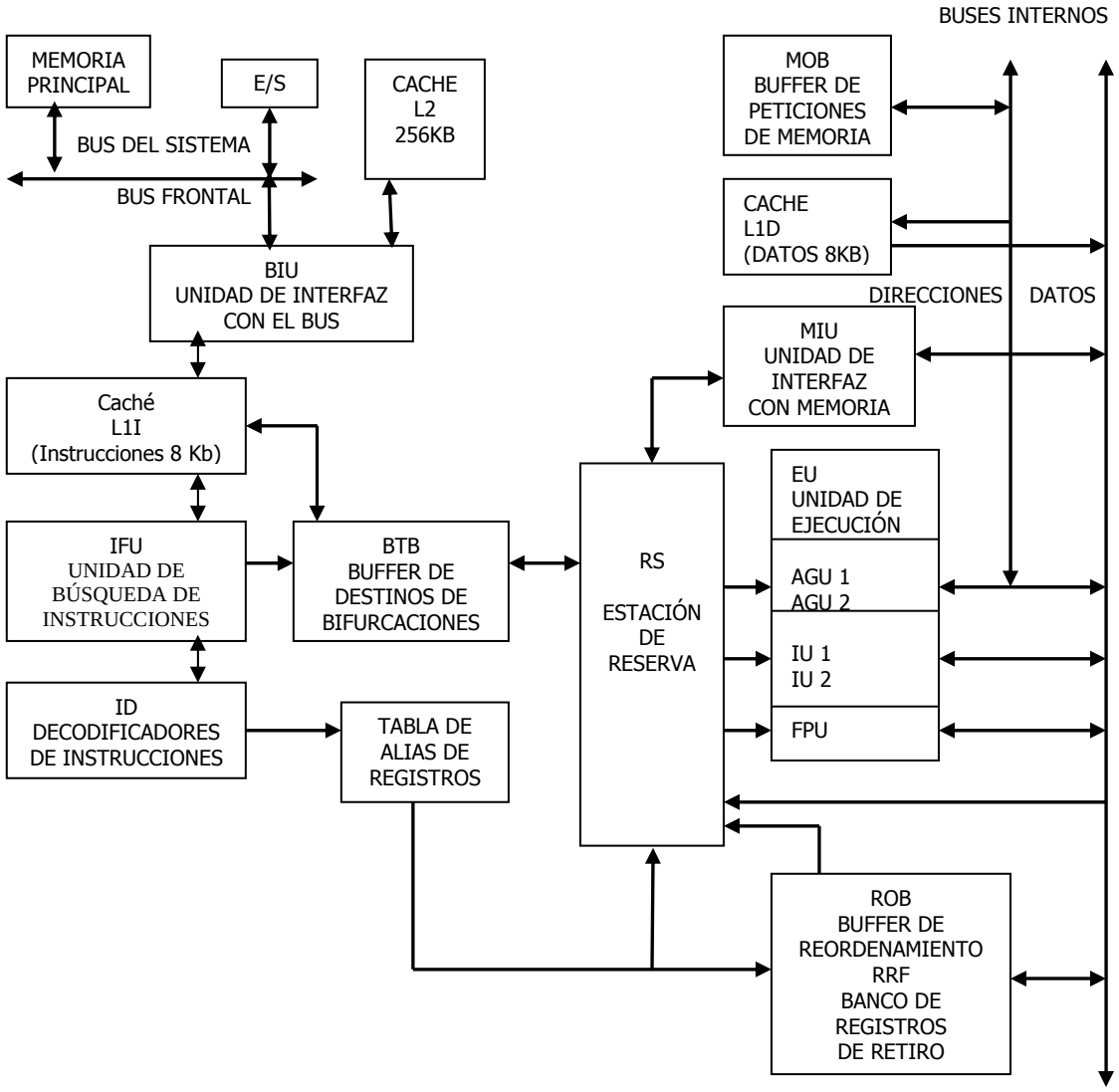


Figura 20.6 Arquitectura interna del Pentium Pro

20.3-.NUEVAS INSTRUCCIONES.

El repertorio de instrucciones de Pentium Pro solo se diferencia del Pentium, en una instrucción nueva, la de mover datos condicionalmente que supone una mayor predicción de ramificaciones en la ejecución.

La nueva instrucción es una instrucción de movimiento condicional que puede ser usada por los escritores de compilación como alternativa a una construcción "test and set" (testear y poner a 1). Esto permite que los bifurcaciones de datos dependientes sean eliminados. El código de resultado será más predecible por el procesador y tendrá por tanto mayor rendimiento.

20.4- ANÁLISIS DEL RENDIMIENTO.

Con la llegada del Pentium Pro se produjo un gran salto en el rendimiento de los computadores.

Para observar la evolución en el rendimiento del Pentium Pro se ofrece el siguiente gráfico con la comparativa del rendimiento en la medida SPECint92.

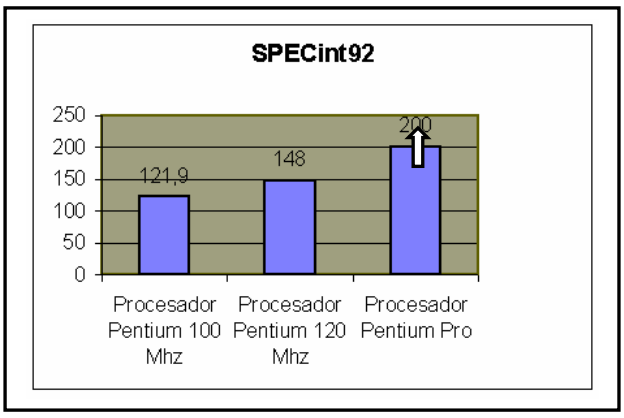


Figura 20.7 Comparativa de rendimiento con SPECint92.

La mejora en el rendimiento se dió debido a la incorporación nuevas mejoras, de las cuales destacaron la ejecución dinámica y la inclusión de una memoria cache secundaria integrada en el encapsulado del chip.

20.4.1- Ejecución dinámica

Para incrementar el rendimiento del Pentium Pro, se intentó aprovechar los tiempos improductivos que se generan en los accesos a la Memoria Principal, es decir cuando se producen fallos en las caches; y usando esos fallos para ejecutar otras instrucciones posteriores que no dependan del resultado de la que se está ejecutando. El Pentium Pro ejecuta las instrucciones fuera del orden, que establece el programa y decidiendo si es posible realizarlas ó no.

EJEMPLO 2

Supongamos que se está ejecutando el siguiente programa:

```
mov r1,(r0)
add r2,r1
inc r6
sub r7,r4
```

La 1ª instrucción carga en r1 el contenido de la posición de memoria apuntada por r0. Dicha posición de memoria, por ser totalmente aleatoria, causará fallo en las caches L1 y L2, por lo que se deberá acceder a la Memoria Principal durante vario ciclos de reloj. En este procesador, para evitar que quede inactivo durante este proceso, se exploran las siguientes instrucciones del programa. La 2ª instrucción no se puede ejecutar porque depende del resultado de la 1ª, que se está llevando a cabo. Pero la 3ª y la 4ª instrucción son independientes de las dos anteriores y pasa a ejecutarlas la CPU originando resultados temporales, que no se transforman en permanentes, hasta que no se realicen las anteriores y se restablezca el orden del programa. Los resultados de la 3ª y la 4ª instrucción se guardan en el Buffer o Depósito de Reordenamiento (ROB).

El Pentium Pro ejecuta las instrucciones fuera de orden. Lo consigue explorando de 20 a 30 instrucciones por delante de la que se halla en curso, y la apunta el contador de programa (CS:EIP). Esta labor es efectuada por La Unidad de Búsqueda, que recoge las instrucciones de L1 y las envía a la unidad de decodificación, la cual convierte las instrucciones x86 en microoperaciones tipo a Risc, que se envían al ROB. En la (figura 20.6) se puede apreciar como la Unidad de Búsqueda y Decodificación introduce al ROB las microoperaciones generadas.

El Buffer de Reordenamiento o ROB comunica con las tres unidades operativas del procesador. La Unidad de selección y Ejecución explora en las microoperaciones del ROB las dependencias de datos y las necesidades de recursos que precisan y seleccionan aquellas que son posibles de ejecutar, las ejecuta y los resultados temporales obtenidos, vuelve a depositarlos en el ROB. Una vez los resultados temporales pasan a ser definitivos al restablecerse el orden original del programa, la Unidad de Extracción, también llamada Unidad de Retiro, los recoge y los almacena.

El comportamiento de la ejecución dinámica de las instrucciones en el Pentium Pro se implementa mediante el uso conjunto de estos tres recursos:

1. Predicción de ramificaciones múltiples.
2. Análisis del flujo de dato.
3. Ejecución especulativa.

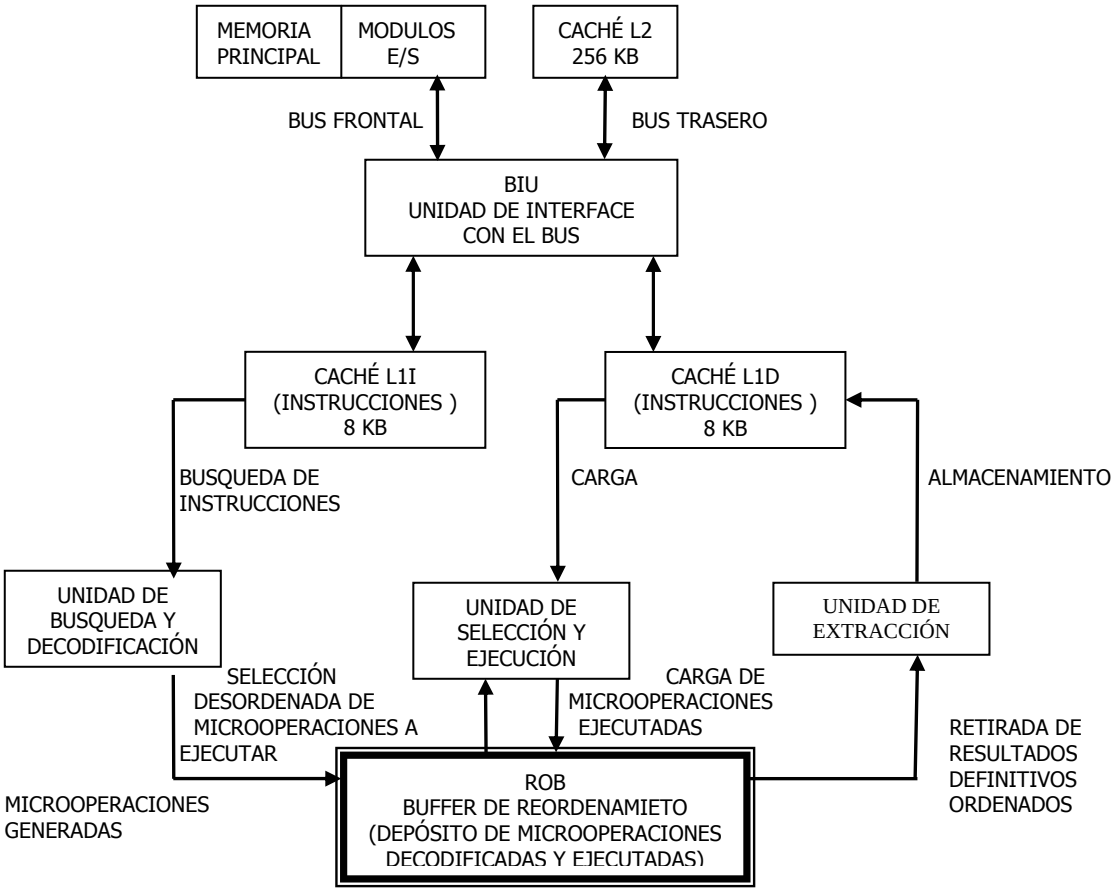


Figura 20.8 El ROB actúa como depósito de microoperaciones y de coordinación entre los tres bloques fundamentales

20.4.2- Orientación software del Pentium Pro

Como ya hemos comentado anteriormente, el Pentium Pro está diseñado con una arquitectura de 32 bits, por lo que su rendimiento es inferior en programas escritos en código de 16 bits. Debido a esto no se eliminaron los bloques de registros parciales, muy comunes en el código de 16 bits.

EJEMPLO 3

Para estudiar esta característica se estudia el siguiente programa:

```
mov a1,c1
mov bx,ax
```

Como la primera instrucción se escribe a1 y con la segunda se lee ax, que es un registro ampliado que contiene a1. Esta secuencia de instrucciones, típica en el código de 16 bits, puede producir un bloqueo de muchos ciclos de reloj en el Pentium Pro, que serán los necesarios hasta que no se retire del ROB la microoperación que provocó la escritura de a1. En el diseño de código de 32 bits debe tenerse cuidado de evitar esta secuencia de instrucciones.

El Pentium Pro para incrementar fuertemente su rendimiento ejecutando las instrucciones fuera de orden, para tener ocupados al máximo todos sus recursos. Las situaciones de bloqueo de los registros parciales implican importantes retrasos, que no están contemplados en el código de 16 bits.

Este microprocesador está orientado a Sistemas Operativos de 32 bits como: Microsoft Windows NT, OS/2 y UNIX. Windows 95 sólo mejora de un 20 a un 30% sobre el Pentium.

Las normas más importantes que se deben seguir en la programación del Pentium Pro:

1. Evitar leer un registro extendido después de haber escrito una parte del mismo.
2. Evitar las bifurcaciones condicionales, usando lo máximo posible las instrucciones JMP, CALL y RET.
3. Alinear los datos.
4. Evitar el uso de código automodificable, que ocasionaría la eliminación del código residente en las caches cuando se mezclasen los canales de ejecución del procesador.

PENTIUM-MMX 21

21.1. CARACTERÍSTICAS GENERALES

21.1.1. Introducción

El 8 de enero de 1997 apareció la última versión de la 5ª generación de los microprocesadores de Intel, el procesador Pentium con tecnología MMX (P55C). Dicho microprocesador incorporaba una tecnología adecuada para aplicaciones multimedia y que como los procesadores anteriores seguía manteniendo la compatibilidad software con toda la familia x86.

La tecnología MMX (MultiMedia eXtensions) fue desarrollada hace varios años como respuesta al crecimiento de la computación basada en multimedia y la fuerte demanda de la generación de gráficos, audio y video de alta calidad, por esta razón se considera el avance más significativo de la arquitectura de INTEL desde el punto de vista del programador. La utilización del PC sería mejorada a través de una nueva generación de sistemas y software que ofrecían recursos con colores de la vida real en toda la pantalla, animación y manipulación de imágenes en tiempo real, sonido 3D, etc.

Este procesador se ofrecía a 166 y 200 Mhz para sistemas "desktop", que estaban orientados hacia el segmento de mercado de consumo. Las dos velocidades de procesadores también se ofertaban como productos embalados individualmente para integradores de sistemas y VARs, a través de distribuidores autorizados INTEL. Los procesadores de 150 y 166 Mhz para computadores portátiles estaban destinados al segmento de mercado de negocios.

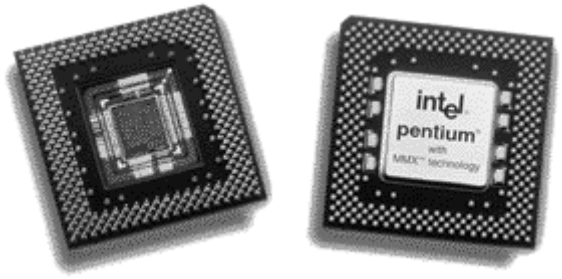


Figura 21.1- Fotografía de la cápsula del Pentium-MMX

El amplio soporte de la tecnología MMX en la comunidad de software proporcionó el desarrollo de nuevas aplicaciones educacionales, de referencia, de juegos y de comunicaciones. Muchos de ellos soportaban capacidades híbridas, combinando las ventajas del procesamiento local de alto desempeño y del almacenamiento de media con los beneficios de una conexión a la Internet, para acceder al contenido multimedia en la Web, interactuando con otras personas, o actualizando aplicaciones.

CARACTERÍSTICAS GENERALES	
AÑO	1997
MILLONES DE TRANSISTORES	4,5
TECNOLOGÍA	0,35
FRECUENCIA (MHZ)	233
PRECIO ($)	550
RENDIMIENTO	7,12 SPECINT95
CÁPSULA	PGA 296
ALIMENTACIÓN (VOLTIOS)	2,8 (CPU) 3,3 (EXTERNA)
POTENCIA (WATIOS)	14
CACHÉ L1 (KBYTES)	16 (INSTRUCCIONES) 16 (DATOS)
CACHÉ L2	NO TIENE (SERÁ EXTERIOR)

Tabla 21.1- Resumen de las características

Comparando el Pentium MMX con sus predecesores cabe destacar el mayor rendimiento de este último debido a la incursión de nuevas caches y las nuevas instrucciones MMX, mayor integración de transistores con una tecnología mas desarrollada (0.35 μm) y la consecución de mayores frecuencias de trabajo, además de otras ventajas entre la que se encuentra la reducción del coste. El Pentium MMX logra mayores frecuencias de trabajo lo que en un principio fue un problema ya que el calor que se disipaba era elevado y generaba errores, por lo que hubo que reducir el voltaje del procesador con la consiguiente reducción en la potencia del mismo.

21.1.2. Versiones del procesador

21.1.2.1. Pentium MMX y Pentium Overdrive

Del procesador Pentium MMX existió una versión OverDrive (P54CTB) para computadoras cuyas tarjetas madres soportaban un zócalo para el procesador tipo ZIF. Se podían cambiar los siguientes procesadores: Pentium de 100MHz por uno Pentium MMX para tener un rendimiento de 166MHz, Pentium de 90MHz por uno Pentium MMX con un rendimiento de 150MHz y finalmente el Pentium de 75MHz por un Pentium MMX con un rendimiento de 125MHz. En todos los casos el factor de aceleración es de 1.6.

Sin embargo para mejorar estos procesadores Pentium que oscilaban entre 120MHz y 200MHz, se tendría que esperar hasta que pasado medio año (justo cuando hizo su debut el Pentium II), se lanzaran al mercado procesadores Pentium MMX Overdrive para estas velocidades.

Existe una ligera pero importante diferencia entre un Pentium MMX y un Pentium MMX Overdrive: el voltaje.

El primer Pentium MMX trabajaba en una tarjeta madre que se alimentaba con 2.8 voltios. Insertar dicho procesador al zócalo de un Pentium no era una tarea muy difícil, ya que ambos operaban sobre un zócalo número 7 de 321 pines con control VRM (Voltaje Regulator Module - Módulo de Regulación de Voltaje), mas todo el trabajo debería detenerse, ya que encender la computadora podría dañar seriamente e inutilizar por completo el nuevo procesador Pentium MMX. Por esta razón, y para todos los usuarios estaban disponibles los Pentium MMX OverDrive

que incorporaban un módulo de regulación de voltaje. Además de ello, el ventilador o enfriador venía fabricado con el mismo procesador. Ha de transcurrir un poco más de tiempo hasta que salieran al mercado nuevas tarjetas madres que soporten a este primer Pentium MMX.

Finalmente cabe destacar que solo existen procesadores Pentium MMX OverDrive para CPUs 486 de 100MHz en adelante. Todos los que posean un procesador anterior pueden lamentablemente estar perdiendo cualquier ilusión. Prácticamente MMX es una tecnología disponible solo para procesadores Pentium.

21.1.2.2. Pentium MMX de bajo consumo

Del procesador Pentium MMX existió otra versión llamada de **bajo consumo** que era prácticamente idéntica, salvo que estaba construido con una tecnología de 0.25 µm y que consumía menos (entre 1'8-2'0 V para la CPU y 2'5 V para el exterior, dependiendo del modelo) y por tanto disipaba menos energía (entre 4'1-7'6 W). Este procesador era operativo a mayores rangos de temperatura, ya que por ejemplo la versión del procesador a 166 Mhz era operativo dentro del rango de –40/+115 ºC. Además de esto esta versión del Pentium MMX introdujo un nuevo encapsulamiento de 352 patitas llamado HL-PBGA (High-termal, low-profile plastic ball grid array).

21.1.3. Orientación y mercado a los que se dirige

Mientras el mercado no estuvo saturado y las marcas rivales no hagan la competencia, INTEL no saca al mercado un nuevo procesador. Cuando la competencia (CYRIX y AMI)) están a punto de sacar un chip equivalente, entonces es cuando INTEL comercializa un nuevo modelo. Ha pasado por ejemplo, con el Pentium 200MHz. Cuando ha visto que la competencia sacaba procesadores de 150MHz y 166MHz (o de prestaciones más o menos equivalentes), entonces INTEL comercializa el Pentium de 200MHz. Y ahora, que ve que la competencia se acerca, saca el Pentium MMX. Ahora bien, licencia la tecnología MMX para poderla imponer como un estándar en el mercado. Esto quiere decir que el secreto de futuro de INTEL no es el MMX, ya que todos los procesadores, tanto los INTEL como los de la competencia, la tendrán, es decir que INTEL prepara algo diferente que las otras marcas no tengan.

Esta tecnología MMX mantiene compatibilidad completa con la arquitectura INTEL y también es totalmente compatible con los sistemas operacionales y las aplicaciones más utilizadas del mercado. Esta tecnología se incluiría en futuros procesadores, incluyendo procesadores Pentium OverDrive®.

21.2. APORTACIONES Y NUEVOS RECURSOS ARQUITECTÓNICOS

21.2.1. Recursos

El Pentium MMX contiene todas las características del procesador Pentium, y además incorpora una caché interna más amplia de 32Kb, de los cuales l6Kb son para datos y los otros 16Kb para instrucciones, frente a los 16Kb de caché interna que tenía el Pentium. Esta memoria caché adicional permite que el procesador tenga un acceso más rápido a la información que utiliza con más frecuencia. Por eso el Pentium MMX es más rápido que el Pentium.

Otro aspecto a destacar del Pentium MMX es la configuración superescalar capaz de, al igual que el Pentium, ejecutar dos instrucciones simultáneamente. Pero en el caso del Pentium MMX, no puede ejecutar simultáneamente dos instrucciones MMX. Lo que sí puede hacer es ejecutar dos instrucciones no MMX simultáneamente o una instrucción MMX y una instrucción no MMX simultáneamente.

Un ejemplo sobre las ventajas del Pentium MMX sobre el Pentium es el siguiente.

EJEMPLO 1

Operación	Numero de instrucciones
Multiplicación de una matriz 4x4 por un vector de 4 elementos	72 instrucciones con MMX 1400 millones de instrucciones sin MMX
Transición de una imagen de 24 bits a otra imagen de 24 bits	28 instrucciones con MMX 528 millones de instrucciones sin MMX

Otra mejora arquitectónica es la predicción mejorada de bifurcaciones, que incrementa el rendimiento ayudando al procesador a determinar con mayor precisión qué instrucción ejecutará la aplicación siguiente.

21.2.2- Arquitectura Pentium MMX

La arquitectura del Pentium MMX, presenta dos características fundamentales como son: la arquitectura SIMD (una sola instrucción, datos múltiples), que explicaremos más adelante y la configuración superescalar capaz de ejecutar dos instrucciones a la vez, con números enteros, en un solo ciclo de reloj. La unidad FPU (Floating-Point Unit, Unidad de Coma Flotante) es la que realiza dichas operaciones y utiliza los formatos especificados en el estándar 754 IEEE de 32 y 64 bits, como también un formato de 80bits.

Las caches inmediatas de l6Kb de código y de datos, como observamos en la figura 21.3, reducen el tiempo de acceso medio a memoria y proporcionan acceso rápido a las instrucciones y a los datos usados recientemente.

Entre las aplicaciones típicas de este procesador se utilizan mucho las imágenes 3D, en las cuales cuando se hacen operaciones de aceleración de Rendering 3D, INTEL recomienda utilizar los registros MMX, en cambio, en cálculos geométricos, recomienda utilizar los registros FP de coprocesador. A pesar de todo, en gráficos 3D, el Pentium MMX solo es un 3'7% más rápido que el Pentium.

Si hablarnos de los registros MMX, el procesador indica que los registros FP están Busy (Ocupados), para que ningún software pueda utilizar el coprocesador en el mismo ciclo o conjunto de ciclos de reloj.

Por software, se puede testear el procesador para ver si es un procesador Pentium MMX o un Pentium. Si después de ejecutar la instrucción CPUID, el bit 23 es "1", quiere decir que es un procesador Pentium MMX, si es "0" quiere decir que es un procesador Pentium.

En lo que se refiere a software, en Windows 95, el Pentium MMX puede llegar a ser un 16% más rápido que el Pentium, debido a su caché más grande. En otros programas, el Pentium MMX solo es un 5% más rápido.

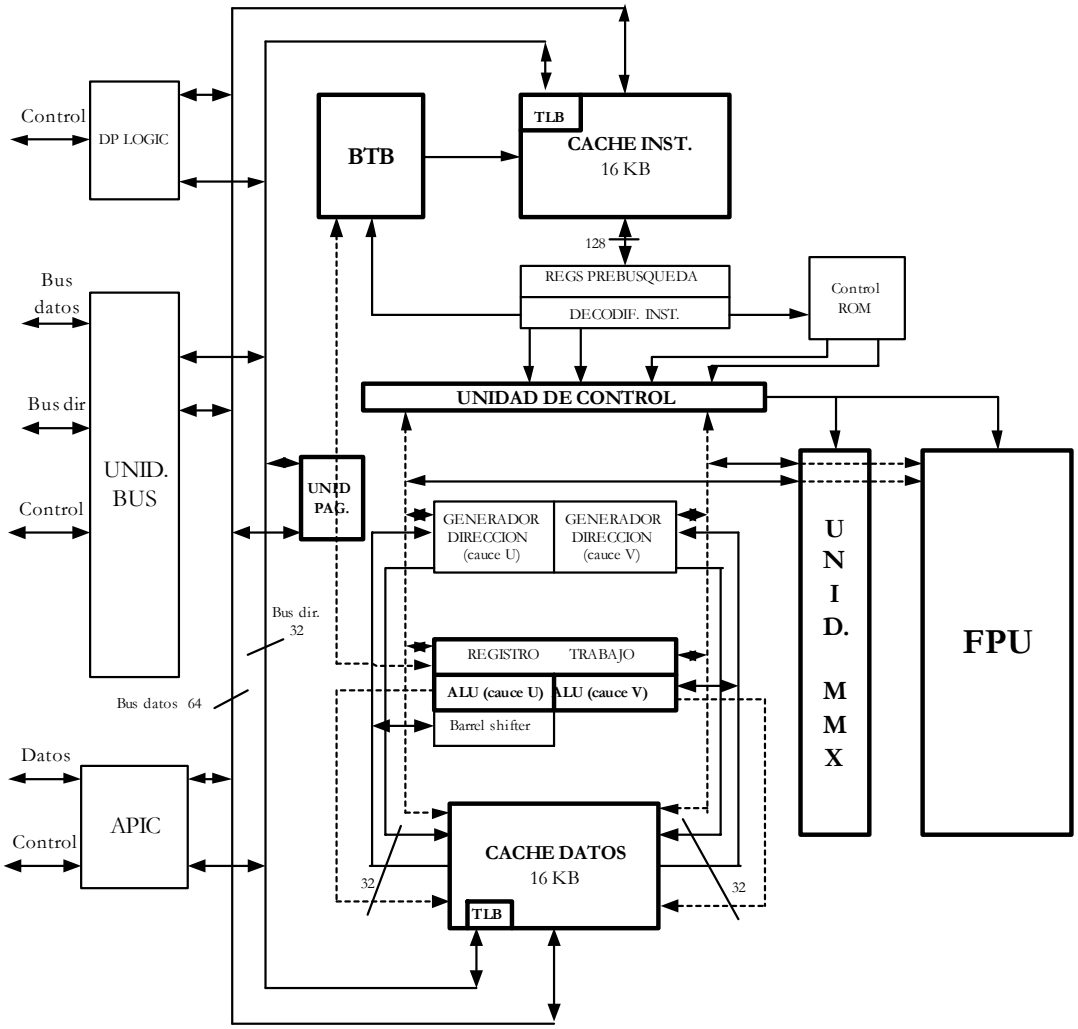


Figura 21.2- Arquitectura del Pentium MMX

Utilizando el DirectX (el API del Windows 95 que se encarga de acelerar los gráficos), el Pentium MMX no es mucho más rápido que el Pentium, es un contrasentido y una cosa extraña, pero parece similar al problema del cuello de botella que tiene el Pentium Pro ejecutando programas de 16bits. En cambio, cuando utiliza el Direct3D (el API del Windows 95 que se encarga de acelerar los gráficos en 3D), si funciona bastante más rápido que el Pentium. Cuando el Pentium MMX funciona con MS-DOS, solo va un 2'5% más rápido que el Pentium, hecho que no significa mucha mejora.

Un aspecto grave es que en el Pentium MMX, al igual que en el Pentium, la multiplicación de enteros con 32bits, en una operación estándar, no es una instrucción del coprocesador ni una instrucción MMX, y gasta 4 veces más ciclos de reloj que la multiplicación normal de l6bits. Un Pentium (tanto el Pentium, como el Pentium MMX) no utiliza el Pipeline, y tarda 10 ciclos de reloj, el Pentium Pro en cambio, utiliza el Pipeline y solo gasta 4 ciclos de reloj. En este aspecto, tanto el Pentium como el Pentium MMX, podrían mejorar el rendimiento.

21.3. NUEVAS INSTRUCCIONES

21.3.1. Juego de instrucciones MMX

MMX quiere decir Multimedia Extensions, e incorpora 57 nuevas instrucciones, además de 4 nuevos tipos de paquetes de datos, todos de 64bits. Las instrucciones MMX son similares a las instrucciones utilizadas por los procesadores Motorola 88110, Hewlett-Packard HP PA-7100 LC y Sun UltraSparc (VIS, Visual Instructions Set o Conjunto de Instrucciones Visual).

El Pentium MMX, al igual que el Pentium, utiliza registros de 80bits (64bits de mantisa y l6bits de exponente), que son 8 registros del copro R0-R7 como se observa en la (figura 21.3.) Las instrucciones MMX y los registros MMX solo utilizan los 64bits de la mantisa, los 16 bits de más peso no son validos.

El Pentium MMX incorpora 8 nuevos registros de 64bits (MMO a MM7), además de los ya conocidos de toda la familia INTEL, y una nueva unidad de predicción tomada del Pentium Pro.

Las instrucciones MMX (MMO a MM7) son, de forma igual a las instrucciones del coprocesador (RO a R7), adicionales. Esto quiere decir que al activar las instrucciones, se activa una parte del procesador. Por esto, a pesar de que las instrucciones MMX y las del coprocesador son compatibles, se molestan, y es mejor utilizar instrucciones MMX o instrucciones del coprocesador, por separado, no mezclarlas porque entonces hay ciclos de reloj de penalización, y se pierde en rendimiento. De hecho, los registros MMX son, físicamente, los mismos registros que los del coprocesador (Floating Point).

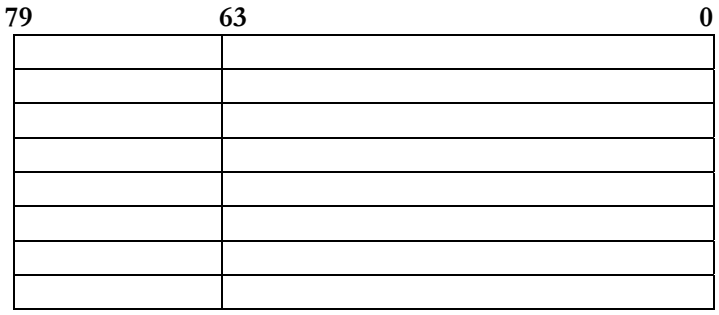


Figura 21.3 Registros del Pentium MMX

La instrucción EMMS (permiso para poner en marcha el set de instrucciones multimedia) cambian el juego de instrucciones MMX a copro.

Las instrucciones MMX manejan los registros SIMD (múltiple flujo de datos, simple flujo de instrucciones), es decir se permite que el procesador ejecute un solo cálculo en elementos múltiples de datos. Todas las UP reciben la misma instrucción, pero operan sobre los diferentes datos procedentes de la memoria compartida como muestra la (figura 21.3.).

Esta técnica se utiliza debido a que las aplicaciones multimedia y comunicaciones, con frecuencia usan ciclos repetitivos que, aunque ocupan el l0% o menos del código total de la aplicación, pueden ser responsables de hasta del 90% del tiempo de ejecución. SIMD permite al chip reducir los ciclos intensos de computación comunes al video, gráfica y animación.

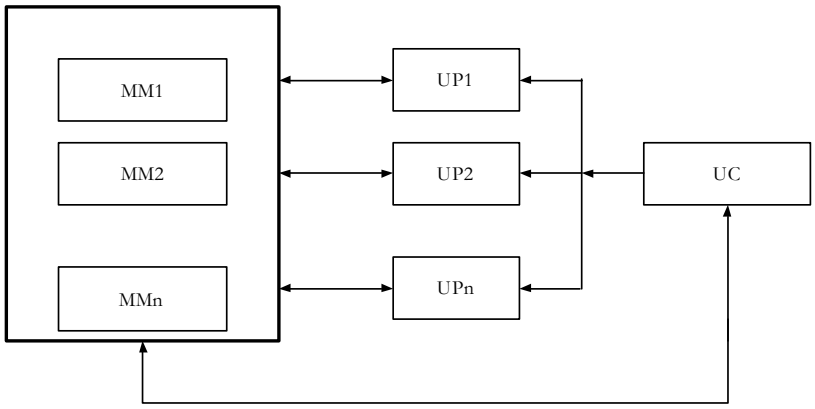


Figura 21.4 Arquitectura de los computadores SIMD

21.3.1.1- Formato de las Instrucciones MMX

Como se ha dicho anteriormente la extensión MMX define un set adicional de 57 nuevas instrucciones. En realidad, no son más que las clásicas instrucciones de suma, multiplicación y desplazamiento, adaptadas para manejar registros de 64 bits.

El formato de las instrucciones MMX es el siguiente:

1. P: Todas las instrucciones MMX comienzan por la letra P, para indicar que trabajan con datos empaquetados.
2. Nemónico clásico, como puede ser: add, and, sub, or, etc.
3. S: Resultado saturación con bit de signo. US: Resultado de la saturación sin signo. Si el resultado sobrepasa el máximo, se utiliza el máximo. Ídem para el mínimo.
4. Tamaño dato: B (byte), W (word), D (doble), Q (cuádruple).

MMX está diseñado para tratar tipos de datos pequeños, sobre los que se realizan operaciones de cálculo de forma intensiva y con gran paralelismo. La operación se realiza sobre un registro de 64 bits (figura 21.5.).

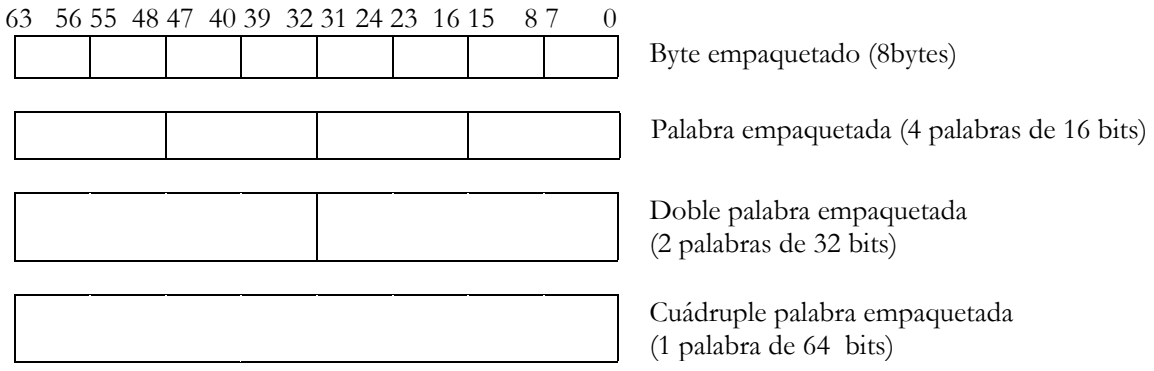


Figura 21.5 Tipos de datos MMX

EJEMPLO 2

Operación aritmética de resta

PSUBUSD MM2,MM3

(MM2)-(MM3) → MM2

DW1	DW2

DW1'	DW2'

Como hemos dicho la primera letra, la P indica que estamos trabajando con datos empaquetados, el nemónico SUB indica que es una operación de resta. Después tenemos la abreviatura US, es decir que si el resultado sobrepasa el máximo nos quedamos con dicho máximo no con otro valor. La última letra es para que sepamos que estamos trabajando con palabras empaquetadas, fijarse en la figura.

EJEMPLO 3

Multiplicar en doble precisión 4 words en MM1 por 4 words en MM2, dejando el resultado en MM1 y MM2

```
MOVQ MM0,MM1 ;       Realiza una copia de MM1
PMULHW MM0,MM2 ;   Calcula los bits de más peso en MM0
PMULLW MM1,MM2 ;   Calcula los bits de menos peso en MM1
MOVQ MM2,MM1 ;       Realiza una copia de los bits de menos peso que deja en MM2
PUNPCKHWD MM1,MM0 ; Mezcla las 2 primeras dobles palabras
PUNPCKLWD MM2,MM0 ;  Mezcla las 2 últimas dobles palabras
```

Este código necesita 6 ciclos para calcular 4 productos. En Pentium, no MMX, se necesitan 10 ciclos para una sola multiplicación de 16x16->32 bits

21.3.1.2. Repertorio de Instrucciones MMX

CATEGORÍA	INSTRUCCIÓN	USO
Aritmética	PADD [B,W,D]	Suma con redondeo
	PADDS [B,W]	Suma con acarreo
	PSUB [B,W,D]	Suma sin acarreo
	PSUBS [B,W]	Resta con redondeo
	PSUBUS [B,W]	Resta sin acarreo
	PMULHW	Multiplicación del byte más significativo
	PMULLW	Multiplicación del byte menos significativo
	PMADDWD	Multiplicación y suma del resultado
Comparación	PCMPEQ[B,W,D]	Comparación (igualdad) entre dos datos
	PCMIPGT[B,W,D]	Comparación (mayor o igual) entre dos datos
Conversión	PACKUS WB	Empaqueta. palabras en bytes, sin acarreo
	PACKSS [WB,DW]	Empaqueta palabras en bytes, y dobles palabras en palabras, con acarreo
	PUNPCKH[BW,WD,DQ]	Desempaqueta los bytes más significativos del registro MMX
	PUNPCKL[BW,WD,DQ]	Desempaqueta los bytes menos significativos del registro MMX
Lógica	PAND	AND empaquetado
	PANDN	AND NOT empaquetado
	POR	OR empaquetado
	PXOR	XOR empaquetado
Desplazamiento	PSLL[W,D,Q]	Desplazamiento lógico a la izquierda
	PSRL[W,D,Q]	Desplazamiento lógico a la derecha
	PSRA[W,D]	Desplazamiento aritmético a la derecha
Movimiento	MOV [D,Q]	Mueve una doble o cuádruple palabra desde o hacia un registro MMX
Estado	EMMS	Vacía el byte de estado MMX

Tabla 21.2 Resumen de instrucciones MMX

Cuando en el final de una instrucción aparecen dos o más letras dentro de corchetes, significa que en realidad se trata de más de una instrucción. Por ejemplo la instrucción PADD [B,W,D] serían en realidad tres instrucciones: PADDB realiza una suma con redondeo entre dos bytes, PADDW realiza una suma con redondeo entre dos palabras y PADDD realiza una suma con redondeo entre dos dobles palabras.

También puede aparecer en otras instrucciones la expresión DQ, cuyo significado es el de cuádruple palabra.

A parte de las instrucciones de la (tabla 21.2) también hay otros juegos de instrucciones multimedia:

- VIS (Visual Instruction Set) para UltraSparc, Añade estimación de movimiento para codificación MPEG. Este set es más potente que el MMX ya que dispone de 32 registros.
- MVI (Motion Video Instructions) para Alpha.
- MDMX (Mips Digital Media Extensions) para MIPS.

INTEL desarrolla MMX2: aceleración de operaciones de procesado de imágenes en 3D. Aquí se subsana el problema de MMX1 con los cálculos en punto flotante

21.4. ANÁLISIS DEL RENDIMIENTO

21.4.1. Introducción

Los incrementos de rendimiento en el Pentium MMX varían dependiendo del tipo de aplicación y el punto hasta el cual el software aprovecha la tecnología MMX de INTEL, los usuarios de aplicaciones de software diseñadas para el nuevo procesador pueden disfrutar de más colores, imágenes de video más uniforme, imágenes enriquecidas y sonido estereofónico en 3D.

Basado en pruebas de referencia estándares de la industria, el software actual se ejecuta a una velocidad promedio 10 a 20% mayor[1] en un procesador Pentium con tecnología MMX de INTEL. El uso de software diseñado para aprovechar la tecnología MMX de INTEL da lugar a un mayor número de mejoras de velocidad y calidad. Según la INTEL Media Benchmark, que mide específicamente el rendimiento de la tecnología MMX en ambientes multimedia, el procesador Pentium con tecnología MMX de INTEL opera a una velocidad superior a 60% mayor.

21.4.2. Índice icomp® 2.0 para los procesadores Pentium MMX

El índice iCOMP® 2.0 refleja aplicaciones y pruebas de referencia de 32 bits y combina cinco pruebas de referencia: CPUMark32, Norton SI-32, SPECint_base95, SPECfp_base95 y la INTEL Media Benchmark. La evaluación de cada procesador se calcula en una computadora de escritorio en el momento en que se introduce el procesador. El rendimiento en sistemas portátiles variará, y otras diferencias en la configuración del hardware y el software, incluido el software diseñado para la tecnología MMX de INTEL, afectarán también el rendimiento real. Las evaluaciones de procesadores aparecidos antes del índice iCOMP® 2.0 se calcularon al aparecer la versión 2.0,

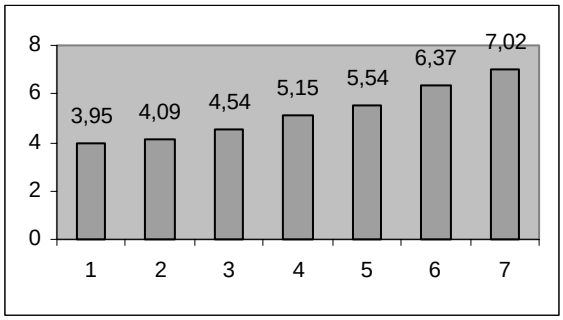


Figura 21.6 SPECint 95 en Windows NT 4.0

[1] Todas las comparaciones se han realizado entre un procesador Pentium con tecnología MMX de INTEL y un procesador Pentium que opera a la misma frecuencia.

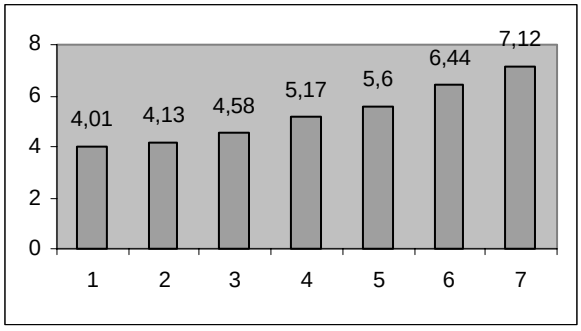


Figura 21.7 SPECint 95 en UNIX

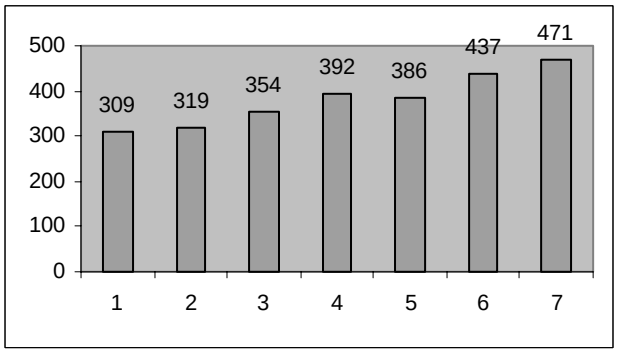


Figura 21.8 CPUmark32 en Windows 95

1. Pentium 133 Mhz
2. Pentium 150 Mhz
3. Pentium 166 Mhz
4. Pentium 200 Mhz
5. Pentium MMX 166 Mhz
6. Pentium MMX 200 Mhz
7. Pentium MMX 233 Mhz

Figura 21.9 Procesadores analizados

El procesador Pentium con tecnología MMX de INTEL contiene mejoras arquitectónicas y también derivadas de la tecnología MMX de INTEL. Los incrementos de rendimiento en el intervalo de 10 a 20% en el software actual provienen principalmente de mejoras arquitectónicas.

Los incrementos adicionales obtenidos de la tecnología MMX de INTEL en los rangos de rendimiento, riqueza y calidad de aplicaciones, dependerán de la cantidad de código de software que se haya diseñado para aprovechar la tecnología MMX de INTEL Los incrementos de rendimiento superiores al 60% según la INTEL Media Benchmark son el resultado de mejoras arquitectónicas y del código de software diseñado para la tecnología MMX de INTEL.

21.4.3. INTEL Media Benchmark

Ésta es una prueba de referencia desarrollada por INTEL que prueba la capacidad de un sistema para ejecutar tipos de datos multimedia como video MPEG, imágenes, sonido y geometría en 3D. Esta prueba de referencia compara resultados en sistemas basados en procesadores Pentium con o sin tecnología MMX de INTEL

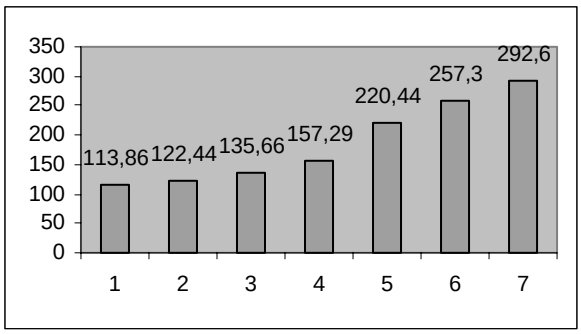


Figura 21.10. INTEL Media Benchmark

1. Pentium 133 Mhz
2. Pentium 150 Mhz
3. Pentium 166 Mhz
4. Pentium 200 Mhz
5. Pentium MMX 166 Mhz
6. Pentium MMX 200 Mhz
7. Pentium MMX 233 Mhz

Figura 21.11 Procesadores analizados

21.4.4. Rendimiento entre diferentes aplicaciones

Las exigentes aplicaciones multimedia repletas de imágenes, video y sonido se prestan bien para aprovechar las mejoras de la tecnología MMX de INTEL. Las aplicaciones como los procesadores de palabras no observan esta ventaja. Incluso en una misma aplicación, diferentes partes del código pueden estar diseñadas para aprovechar la tecnología MMX de INTEL en diferentes niveles. Por ejemplo, algunos filtros de Adobe PhotoDeluxe[2] han sido altamente optimizados, mientras que funciones administrativas más básicas, como abrir y cerrar archivos, no han pasado por este proceso. Además, muchos fabricantes de software han aprovechado el rendimiento adicional del procesador que ofrece la tecnología MMX de INTEL no sólo para hacer que la aplicación se ejecute a mayor velocidad, sino también para mejorar la calidad de la aplicación agregando más colores, gráficos más detallados, sonido de mayor calidad, imágenes de video más uniformes, etc. Para cualquier usuario la mejora observada estará basada en el punto hasta el cual el código haya sido diseñado para la tecnología MMX de INTEL. En general, se puede esperar que las aplicaciones operen de 10 a 20% más rápido en un procesador Pentium con tecnología MMX de INTEL incluso con software no diseñado específicamente para esta tecnología.

[2] Todas las comparaciones se han realizado entre un procesador Pentium con tecnología MMX de INTEL y un procesador Pentium que opera a la misma frecuencia.

PENTIUM II 22

22.1. INTRODUCCIÓN

En mayo de 1997 Intel lanza al mercado el nuevo procesador Pentium II, con el objetivo básico de cubrir las marcadas deficiencias del Pentium Pro manejando código de 16 bits. Todos los que han manejado computadores dotados de procesadores Pentium Pro saben que aplicaciones con código de 16 bits como lo son todas las del DOS y Windows 3.11, corren efectivamente muy lentamente. Esto ha asociado siempre al Pentium Pro con Windows NT particularmente, y con algunos otros sistemas operativos nativos de 32 bits. Por cierto que Pentium II no intenta sustituir al Pentium Pro. Este último tiene muy bien ganado su lugar entre equipos servidores, incluso existe una versión mejorada del Pentium Pro, que tiene una memoria de caché L2 gigantesca.

Figura 22.1 – Vista anterior Slot One

La arquitectura del Pentium II se basa en el nuevo "**Slot One**" denominado Ranura Uno. Es la nueva arquitectura del Pentium II, esto significa que ya no cabe en la ranura Nº 7 del Pentium y Pentium MMX, como tampoco cabe en la ranura Nº 8 del Pentium Pro (las ranuras Nº 7 y Nº8 son estandarizaciones, tanto en tamaño como en cantidad de orificios empleadas por Intel para las ranuras que sostienen sus procesadores como los clones).

El Pentium II viene junto con la memoria caché L2 y algunos elementos de soporte en una pequeña tarjeta de circuito, que tiene una ranura única y muy particular, lo que ha de constituirse en el mayor obstáculo para todos los que estén acostumbrados a remover un procesador y a incorporar otro a la misma ranura de la tarjeta madre. El Pentium II no trae incorporado dentro del mismo chip la caché de nivel 2 o L2 como sucedía con el Pentium Pro, aspecto que elevaba considerablemente el precio de fabricación. En vez, tanto el microprocesador como los chips de memoria de caché vienen en una pequeña tarjeta de 242 contactos, que es la que en definitiva se inserta a la tarjeta madre.

En la figura 1 se puede apreciar una tarjeta "Slot One". En la parte central de la misma se halla la ranura para el microprocesador, es decir el Pentium II, y a los lados las memorias caché L2 más el tradicional conjunto de circuitos y dispositivos electrónicos de apoyo. Ciertamente que el Pentium II no ha de tener el mismo rendimiento que el Pentium Pro, nada como tener la caché L2 dentro del mismo procesador y corriendo a la misma velocidad, pero el hecho de que estén tan cerca incrementa notablemente el rendimiento, mucho más que cualquiera de los procesadores de la línea del Pentium. La velocidad de reloj de la caché L2 del Pentium II será la mitad de la velocidad del procesador, y el tipo de memoria es BSRAM ("Burst Static RAM" - RAM Estática de Estallido), con un tamaño de 256 ó 512KB. Por otra parte, la comunicación del procesador con el bus seguirá siendo a 66.6MHz.

Pentium II es un procesador que incorpora aproximadamente 7.5 millones de transistores basado en la arquitectura P6, lo que no significa que sea un Pentium Pro en esencia, sino que incorpora algunas de las características más importantes de ese procesador.

Adicionalmente la tecnología que soporta su fabricación es de 0.35 micrones. Las versiones iniciales de este procesador funcionan a una frecuencia de 233 y 266 MHz.

Las cualidades por las que destaca este procesador son: su arquitectura de bus dual e independiente, la tecnología MMX, tecnología de ejecución dinámica y cartucho de contactos de contacto simple. Estos aspectos mejoran el potencial del procesador en tres grandes aspectos: el cálculo de la coma flotante, operaciones multimedia, y mejora el cálculo de enteros.

22.2. APORTACIONES Y NUEVOS RECURSOS ARQUITECTÓNICOS

22.2.1. Consumo y Alimentación

Para los usuarios lo único apreciable es el mayor o menor rendimiento de un procesador, pero para los diseñadores e ingenieros, existe y existirá siempre un problema crítico: el manejo de los voltajes eléctricos de un procesador, no solamente en cuanto a su distribución dentro del procesador, sino también al enfriamiento que se debe propinar al mismo a fin de que no se sobrecaliente. Cuanto mayor sea la velocidad de procesamiento de la CPU, más enfriamiento debe tener.

Los tres voltajes que requiere el Pentium Pro desde la tarjeta madre fue un problema en su momento: 5 voltios para el manejo del bus, 3.3 voltios para la lógica interna del procesador y 2.45 voltios para el intercambio de información entre la CPU y la caché. Estos tres voltajes son suministrados por una unidad especial reguladora situada muy cerca al Pentium Pro. El Pentium II va un poco más allá de estos tres voltajes.

Intel ha optimizado el Pentium II a fin de que pueda regular sus propios voltajes hasta alcanzar sus especificaciones necesarias. Requiere de una unidad de suministro de energía capaz de aceptar una señal identificadora de voltaje compuesta por 5 bits. Este código le indicará a la unidad de suministro, el voltaje requerido por el procesador. Este código será emitido por el procesador a través de 5 pines del mismo, contra los 4 pines que el Pentium Pro emplea para especificar sus demandas. Por su parte y en respuesta al código, la unidad de suministro de energía debe ser capaz de devolver un voltaje entre 2.1 y 3.5 voltios regulada dentro de un rango de +-100mV. El procesador no debe recibir voltajes superiores a los indicados ya que el recalentamiento sería inmediato, y los circuitos podrían verse seriamente dañados. Este punto ha sido un gran problema a resolver para los fabricantes de unidades de alimentación eléctrica para el Pentium II.

Figura 22.2 – Ventilador de enfriamiento del Pentium II

A fin de atacar el problema de la disipación masiva de calor, el procesador puede automáticamente bajar su consumo a fin de reducir el recalentamiento, esto en períodos de baja actividad. Por supuesto, el ventilador (figura 2) permanecerá siempre disponible, construido en un chasis especial, denominado **SEC**, que recubre todo el "Slot One" del Pentium II.

22.2.2. Arquitectura del bus dual independiente

Para satisfacer las demandas de las aplicaciones y anticipar las necesidades de las generaciones futuras de procesadores, Intel ha desarrollado la arquitectura "Dual Independent Bus" (Bus Dual Independiente) para resolver las limitaciones en el ancho de banda de la arquitectura de la plataforma actual del PC.

La arquitectura Bus Dual Independiente fue implementada por primera vez en el procesador Pentium Pro y tendrá disponibilidad más amplia con el procesador Pentium II.

Intel creó esta arquitectura para ayudar al ancho de banda del bus del procesador. Al tener dos buses independientes el procesador Pentium II está habilitado para acceder a datos desde cualesquiera de sus buses simultáneamente y en paralelo, en lugar de hacerlo en forma sencilla y secuencial como ocurre en un sistema de bus simple.

La capacidad de bus es uno de los parámetros más interesantes para comprender y medir el potencial de transferencia de datos de un computador. Esta capacidad puede ser obtenida multiplicando la velocidad del bus o frecuencia de operación del mismo por el número de bytes que el procesador puede mover en cada pulso de reloj. Así, el Pentium II opera a una velocidad de bus de sistema de 66.6MHz y su ancho de bus es de 8 bytes, para hacer un total de 533MB/seg.

Cuando el procesador trabaja a una velocidad mayor a la del bus o bien ejecuta más de una instrucción por ciclo de reloj, los datos alimentados por el bus le resultarán insuficientes para procesarlos, generándose un cuello de botella. De esta forma se requiere, o bien incrementar la velocidad del bus, o incrementar la capacidad de transferencia del bus, que es lo mismo que su ancho en número de bits.

PROCESADOR	CAPACIDAD DE TRANSFERENCIA
Procesador Pentium Estándar	533 MB/seg
Procesador Pentium con bus a 75 MHz	600 MB/seg
Procesador Pentium II de 233 MHz	1466 MB/seg
Procesador Pentium II de 266 MHz	1600 MB/seg
Procesador Pentium II de 300 MHz	1733 MB/seg

Tabla 22.1 – Comparativa de rendimiento de buses

Dos buses conforman la Arquitectura de Bus Dual Independiente: El **Bus de la Caché L2** y el **Bus del Sistema**. Cada uno tiene un ancho de 8 bytes, es decir 64 bits. De esta forma, se puede decir que se doblan los canales disponibles para el movimiento de información. El primero de los buses, el bus de la caché L2 está integrado en el mismo **SEC**, y su velocidad no se halla limitada a la velocidad del reloj de la tarjeta madre, es decir, la caché L2 del Pentium II trabaja a la mitad de frecuencia con la que lo hace el mismo procesador. Este esquema puede ser apreciado en la figura 3. Al tener una frecuencia de operación superior a la de la tarjeta madre, su rendimiento se incrementa notablemente. La tabla 1 muestra una comparativa de las capacidades de movimiento de información de los buses Pentium y Pentium II.

Esta velocidad extra le permite al Pentium II obtener la información que requiere procesar de la caché L2 tan pronto como la necesite. Por su parte, la caché L2 puede manejar una velocidad menor para comunicarse con la memoria principal. Esta es quien sabe uno de los puntos más interesantes e importantes que le dan al Pentium II la ventaja extra en cuanto a procesamiento de información se refiere respecto a sus competidores: los Pentium MMX.

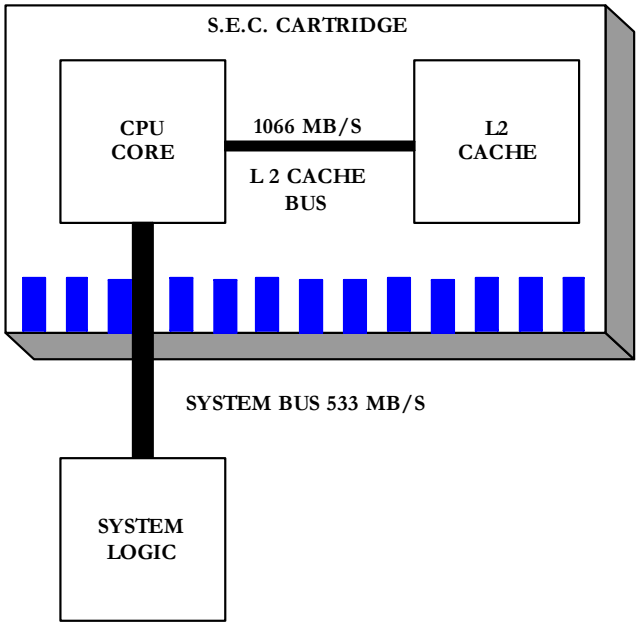


Figura 22.3 – Arquitectura de Bus dual independiente

22.2.3. Ejecución dinámica

Utilizada por primera vez en el procesador Pentium Pro, la Ejecución Dinámica es una innovadora combinación de tres técnicas de procesamiento diseñada para ayudar al procesador a manipular los datos más eficientemente. Éstas son la predicción de ramificaciones múltiples, el análisis del flujo de datos y la ejecución especulativa. La ejecución dinámica hace que el procesador sea más eficiente manipulando datos en lugar de sólo procesar una lista de instrucciones.

La forma cómo los programas de software están escritos puede afectar el rendimiento de un procesador. Por ejemplo, el rendimiento del software será afectado adversamente si con frecuencia se requiere suspender lo que se está haciendo y **saltar** o **ramificarse** a otra parte en el programa. También pueden ocurrir retardos cuando el procesador no puede procesar una nueva instrucción hasta completar la instrucción original. La ejecución dinámica permite al procesador alterar y predecir el orden de las instrucciones.

22.2.3.1. Predicción de ramificaciones múltiples

Predice el flujo del programa a través de varias ramificaciones, mediante un algoritmo de predicción de ramificaciones múltiples, el procesador puede anticipar los saltos en el flujo de las instrucciones. Éste predice dónde pueden encontrarse las siguientes instrucciones en la memoria con una increíble precisión del 90% o mayor. Esto es posible porque mientras el procesador está buscando y trayendo instrucciones, también busca las instrucciones que están más adelante en el programa. Esta técnica acelera el flujo de trabajo enviado al procesador.

22.2.3.2. Análisis del flujo de datos

Analiza y ordena las instrucciones a ejecutar en una sucesión óptima, independiente del orden original en el programa. Mediante el análisis del flujo de datos, el procesador observa las instrucciones de software decodificadas y decide si están listas para ser procesadas o si dependen de otras instrucciones. Entonces el procesador determina la sucesión óptima para el procesamiento y ejecuta las instrucciones en la forma más eficiente.

22.2.3.3. Ejecución Especulativa

Aumenta la velocidad de ejecución observando el contador del programa y ejecutando las instrucciones que posiblemente van a necesitarse. Cuando el procesador ejecuta las instrucciones (hasta cinco a la vez), lo hace mediante la ejecución especulativa. Esto aprovecha la capacidad de procesamiento superescalar del procesador Pentium II tanto como es posible para aumentar el rendimiento del software. Como las instrucciones del software que se procesan con base en predicción de ramificaciones, los resultados se guardan como **resultados especulativos**. Una vez que su estado final puede determinarse, las instrucciones se regresan a su orden propio y formalmente se les asigna un estado de máquina.

22.2.4. Memorias caché

El Pentium incluye dos cahés, una para datos y otra para instrucciones. Cada caché es de 8 KBytes, utilizando un tamaño de línea de 32 bytes y una organización asociativa por conjuntos de dos vías. El Pentium Pro y el Pentium II incluyen también dos cachés L1. Las primeras versiones del procesador incluye una caché de instrucciones de 8 KBytes, asociativa por conjuntos de cuatro vías, y una caché de datos, también de 8 KBytes, asociativa por conjuntos de dos vías. Ambos incluyen además una caché L2 que alimenta a las dos cachés L1. La caché L2 es asociativa por conjuntos de cuatro vías, y con tamaños que oscilan entre 256 KBytes y 1MByte.

El núcleo del procesador consta de cuatro componentes principales:

- **Unidad de captación / decodificación:** capta instrucciones en su orden de la caché de instrucciones L1, las decodifica en una serie de micro operaciones, y memoriza los resultados en el depósito ("pool") de instrucciones.
- **Depósito de instrucciones:** contiene el conjunto de instrucciones actualmente disponibles para ejecución.
- **Unidad de envio/ejecución:** planifica la ejecución de las micro operaciones sujetas a dependencias de datos y disponibilidad de recursos, de manera que las micro operaciones pueden planificarse para su ejecución en un orden distinto al que fueron captadas. Cuando hay tiempo disponible esta unidad realiza una ejecución especulativa de micro operaciones que pueden ser necesarias en el futuro. La unidad ejecuta micro operaciones, captando los datos necesarios de la caché de datos L1, y almacenando los resultados temporalmente en registros.
- **Unidad de retirada:** determina cuándo los resultados provisionales, especulativos, deben retirarse (unificarse) para establecerse como permanentes en registros o en la caché de datos L1. Esta unidad también elimina instrucciones del depósito tras haber unificado los resultados.

La Figura 4 proporciona una visión simplificada de la estructura del Pentium II, resaltando la ubicación de las tres cachés.

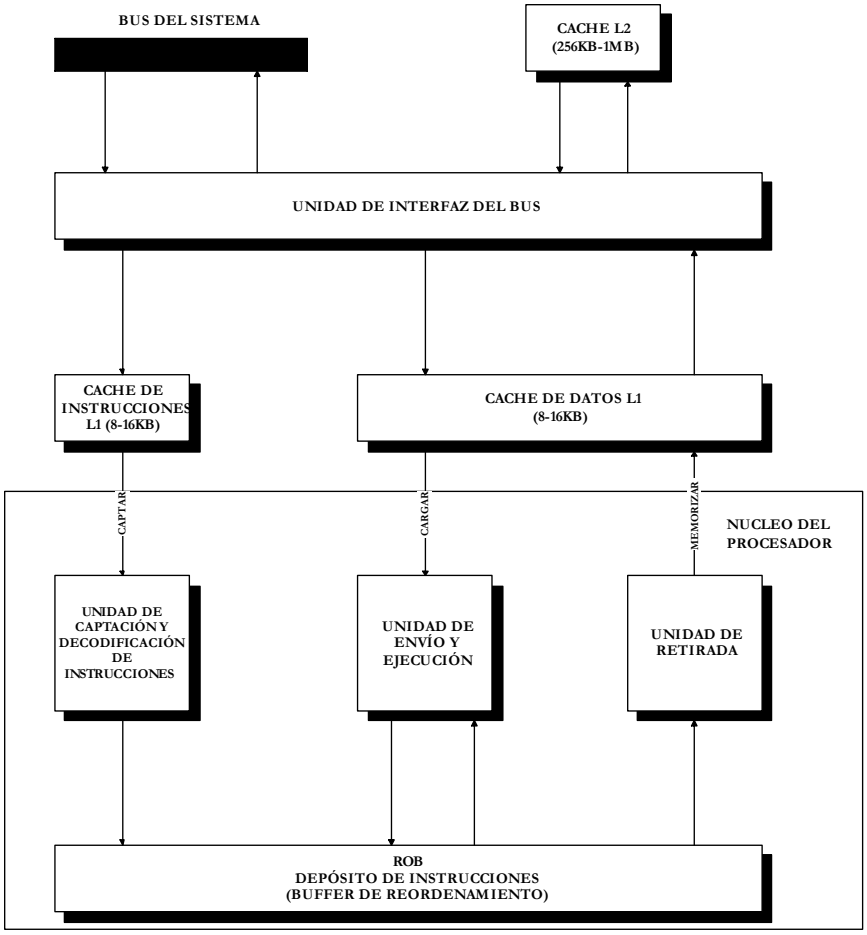


Figura 22.4 – Diagrama de bloques del Pentium II

22.2.5. Unidad de predicción de saltos

El Pentium II usa una estrategia de predicción dinámica de saltos basada en la historia de las ejecuciones recientes de instrucciones de bifurcación. Se utiliza un buffer de destino de saltos ("branch target buffer", **BTB**) que guarda información sobre las instrucciones de bifurcación encontradas recientemente. Siempre que aparece una instrucción de bifurcación en el flujo de instrucciones, se comprueba el BTB. Si ya existe una entrada en el BTB, la unidad de instrucciones se guía por la información de historia guardada en esa entrada, para determinar si predice que se producirá el salto. Si se predice un salto, la dirección destino del salto asociada con esta entrada, se utiliza para precaptar la instrucción destino del salto.

Una vez que se ejecuta la instrucción, la parte de historia de la entrada adecuada se actualiza, para que refleje el resultado de la instrucción de bifurcación. Si la instrucción no está representada en el BTB, se carga su dirección en una entrada del BTB; si es preciso, se borra una entrada más antigua.

El BTB del Pentium II está organizado como una caché asociativa por conjuntos de cuatro vías con 512 líneas. Cada entrada utiliza la dirección de la instrucción de salto como etiqueta. La entrada también incluye la dirección destino del salto de la última vez que se produjo, y un campo de historia de cuatro bits.

22.2.6. Ejecución fuera de orden

Los Pentium Pro/II pueden ejecutar las instrucciones no en el orden que lleguen, sino reordenarlas para que no haya dependencias entre ellas. Esta reordenación está hecha en tiempo de ejecución utilizando el algoritmo de Tomasulo. Este algoritmo también contempla el renombramiento de registros. Pentium II tiene 40 registros, aparte de los tradicionales del 80x8, que se utilizan para la ejecución de los comandos. Los 16 registros de enteros y los 12 de la unidad de coma flotante tradicionales del 80x86/87 para la escritura del resultado de las instrucciones 80x86.

22.3. NUEVAS INSTRUCCIONES

Intel introdujo en 1996 la tecnología MMX en su línea de procesadores Pentium. MMX comprende instrucciones muy optimizadas para tareas multimedia. Hay 57 instrucciones nuevas que tratan los datos en un modo SIMD ("single-instruction", "multiple-data"), es decir una secuencia de instrucciones y múltiples secuencias de datos, que posibilita efectuar la misma operación, tal como una suma o una multiplicación, con varios elementos de datos a la vez.

Cada instrucción suele ejecutarse en un ciclo de reloj. Para ciertas aplicaciones, estas operaciones rápidas en paralelo pueden conducir a una velocidad entre dos y ocho veces superior a la de algoritmos equiparables que no utilicen las instrucciones MMX.

El conjunto de instrucciones MMX está orientado a programación multimedia. Los datos de vídeo y audio suelen representarse mediante vectores o matrices grandes, compuestos por datos de longitud reducida (8 ó 16 bits), mientras que las instrucciones convencionales operan normalmente con datos de 32 ó 64 bits. Estos son algunos ejemplos: En gráficos y en vídeo, cada escena consiste en una matriz de puntos de imagen 1, y hay 8 bits para cada punto de imagen, u 8 bits para cada componente de color (rojo, verde, azul) del punto de imagen. Las muestras de audio suelen estar cuantizadas con 16 bits. Para algunos algoritmos de gráficos 3D es común emplear 32 bits para los tipos de datos básicos. Para posibilitar el procesamiento paralelo con estos tamaños de datos, en MMX se definen tres nuevos tipos de datos. Tienen una longitud de 64 bits, y constan de varios campos de datos más pequeños, cada uno de los cuales contiene un entero en coma fija.

22.4. ANÁLISIS DEL RENDIMIENTO

El estudio del rendimiento se hace en base a diversos programas de prueba a partir de los cuales obtendremos diferentes resultados. Estos programas los podemos clasificar de acuerdo a características en las que coincidan. En concreto, en esta ocasión nos fijaremos en los programas de pruebas que trabajan bajo DOS, WINDOWS, y un grupo adicional para programas que trabajen con las instrucciones MMX.

22.4.1. Rendimiento bajo DOS

Observando la tabla 2, se puede apreciar que el fuerte del Pentium II no son las aplicaciones del DOS con resoluciones bajas, más bien el Pentium MMX tiene un mejor desempeño en varias pruebas, y aunque el Pentium Pro no es un procesador optimizado para aplicaciones de 16 bits, tiene la delantera en varias pruebas. La instrucción adicional que el Pentium Pro puede ejecutar resulta siempre una ventaja extra para el computador.

Claramente se puede apreciar que los famosos juegos que corren bajo DOS no han de ser de lo más adecuado para el Pentium II, sí para el Pentium MMX, y aunque resulta mucho para tales aplicaciones, también para el Pentium Pro.

Pero en la actualidad algunos juegos deben ser lo único que permanece en el mercado del viejo DOS, el fuerte del software está disponible para 32 bits, es decir Windows 95 y sistemas operativos superiores.

PRUEBA	PENTIUM II - 233 MHz 512KB Caché L2	PENTIUM PRO 233MHz 256KB Caché L2	PENTIUM MMX 233MHz 512KB Caché L2
Quake Timedemo2@ 320x200	45.9	47	49.1
Quake Timedemo2@ 480x360	22.3	23.8	25.2
Quake Timedemo2@ 640x480	…	22.7	18.3
PCPBench@ 640x480	33.4	35	27.6
3Dbench	200	500	200
CDBench	53.1	53.8	46.7

Tabla 22.2 – Rendimiento bajo DOS

22.4.2. Rendimiento bajo WINDOWS

22.4.2.1. Rendimiento bajo WINDOWS 95

Particularmente vale la pena analizar el rendimiento de Pentium II bajo un sistema operativo como Windows 95. La tabla 3 muestra algunas de las pruebas clásicas sobre el Pentium II en un ambiente 16/32 bits. Viendo los resultados no cabe duda en que este procesador tiene un desempeño importante en todo lo que se refiere a operación gráfica y multimedia, particularmente con código de 32 bits. Sin embargo su rendimiento no es destacable con código de 16 bits, contra su más directo opositor, el Pentium MMX. Si existe un punto importante a favor del Pentium II es su rendimiento de operación interna. Esto significa que algunas aplicaciones que hacen uso intensivo de la CPU, como cálculos matemáticos o aplicaciones gráficas, y que no emplean demasiados accesos a discos duros o a memoria, sacarán partido del procesador mejor que ninguna otra aplicación de software estándar.

PRUEBA	PENTIUM II - 233 MHz 512KB Caché L2	PENTIUM PRO 233MHz 256KB Caché L2	PENTIUM MMX 233MHz 512KB Caché L2
Business Winstone 97	54.8	54.3	53.2
High End Winstone 97	25.2	26.2	24.2
Winstone 96	100.8	93.6	109
CPUMark 16	442	418	473
CPUMark 32	605	622	464
Business Graphics Winmark 97	91.6	86.1	90.5
High End Graphics Winmark 97	37.7	35.4	40.4

Tabla 22.3 – Rendimiento bajo WINDOWS

22.4.2.2. Rendimiento bajo WINDOWS NT

Windows NT ha sido siempre del dominio del Pentium Pro, mucho más cuando consideramos que en las pruebas anteriores no se ha empleado el más poderoso de los Pentium Pro, como el que tiene 512KB de caché. No cabe duda que esos 512KB construidos con el mismo procesador es la clave del alto rendimiento de estos procesadores.

22.4.3. Rendimiento MMX

La demanda del software por procesadores con capacidades mejoradas para el manejo de video y sonido, es decir multimedia, se ha incrementado considerablemente, y en este punto es donde el Pentium II reúne la mayor puntuación.

De forma general y como se aprecia en la tabla 4, el Pentium II en aplicaciones que hacen uso intensivo del conjunto de instrucciones MMX, es superior a sus opositores, en puntos específicos como pueden ser la velocidad de video, el procesamiento de imágenes, gráficos tridimensionales y audio, el Pentium II tiene un claro desempeño mejorado. Esta ha de constituir una buena noticia para todos los que emplean aplicaciones multimedia, como también para todos los que desean observar Internet en sus computadoras. Sin embargo, y para ser sinceros, es importante destacar el rendimiento del Pentium MMX, que sigue demostrando una capacidad mejorada gracias a la tecnología MMX. Desde ese punto de vista, ambos procesadores son relativamente similares, no tanto en estructura, ya que el Pentium II se asemeja estructuralmente más al Pentium Pro, pero sí en el tamaño de las cachés y en el rendimiento. Cabe notar que el Pentium II en definitiva, siempre ha de tener ventaja sobre el Pentium MMX por su nueva estructura interna.

PRUEBA	PENTIUM II – 233 MHz 512KB Caché L2	PENTIUM PRO 233MHz 256KB Caché L2	PENTIUM MMX 233MHz 512KB Caché L2
Intel Media Benchmark Overall	301.64	235.46	287.64
Intel Media Benchmark Video	309.6	203.94	315.35
Intel Media Benchmark Image Processing	960.56	255.33	836.76
Intel Media Benchmark 3D Graphics	215.41	242.43	182.4
Intel Media Benchmark Audio	343.74	281.54	346.41
Monster Truck Madness Benchmark (Plus Patch)	9	9	15

Tabla 22.4 – Rendimiento MMX

22.5. VERSIONES ESPECÍFICAS DEL PROCESADOR

22.5.1. Celeron

Debemos distinguir entre dos empaquetados distintos. El primero es el SEPP que es compatible con el Slot 1 y que viene a ser parecido el empaquetado típico de los Pentium II (el SEC) pero sin la carcasa de plástico.

El segundo y más moderno es el PPGA que es el mismo empaquetado que utilizan los Pentium y Pentium Pro, pero con distinto zócalo. En este caso se utiliza el Socket 370, incompatible con los anteriores socket 7 y 8 y con los actuales Slot 1. Por suerte existen unos adaptadores que permiten montar procesadores Socket 370 en placas Slot 1 (aunque no al revés).

También debemos distinguir entre los modelos que llevan caché y los que no, ya que las diferencias en prestaciones son realmente importantes. Justamente los modelos sin caché L2 fueron muy criticados porque ofrecían unas prestaciones que en algunos casos eran peores que las de los Pentium MMX a 233.

El procesador Celeron está optimizado para aplicaciones de 32 bits. Se comercializa en versiones que van desde los 266 hasta los 466 MHz. La caché L2 trabaja a la misma velocidad que el procesador (en los modelos en los que la incorpora).

Posee 32 Kbytes de caché L1 de primer nivel repartidos en 16 Kbytes para datos y los otros 16 Kbytes para instrucciones. Los modelos 266-300 no poseen caché de nivel dos L2, pero el resto de los modelos poseen una caché L2 de 128 Kbytes.

La velocidad a la que se comunica con el bus sigue siendo de 66 MHz. Posee un juego de instrucciones MMX.

Por último, este procesador incorpora 7,5 millones de transistores en los modelos 266-300 y 9,1 millones a partir del 300ª (por la memoria caché integrada).

En la tabla 5 se pueden ver las especificaciones de la gama Celeron.

PROCESADOR	FRECUENCIA	CACHÉ L2	TECNO-LOGIA	VOLTAJE I/O	BUS	MULTIPLI-CADOR	ZOCALO
Celeron 266	266Mhz.	0	0,25 µ	3,3	66Mhz	4	Slot1
Celeron 300	300Mhz	0	0,25 µ	3,3	66Mhz	4,5	Slot1
Celeron 300A	300Mhz.	128 KB	0,25 µ	3,3	66Mhz	4,5	Slot1-S.370
Celeron 333	333Mhz	128 KB	0,25 µ	3,3	66Mhz	5	Slot1-S.370
Celeron 366	366Mhz.	128 KB	0,25 µ	3,3	66Mhz	5,5	Slot1-S.370
Celeron 400	400Mhz	128 KB	0,25 µ	3,3	66Mhz	6	Slot1-S.370
Celeron 433	433Mhz.	128 KB	0,25 µ	3,3	66Mhz	6,5	Slot1-S.370
Celeron 466	466Mhz	128 KB	0,25 µ	3,3	66Mhz	7	S.370

Tabla 22.5 – Especificaciones de la gama Celeron

22.5.2. Xeon

Basado en la arquitectura del procesador Pentium II, el procesador Pentium II Xeon agrega el rendimiento, facilidad de uso y confiabilidad en misión crítica superiores que exigen sus servidores y estaciones de trabajo basados en Intel.

Este procesador está disponible con memorias caché grandes y rápidas que procesan los datos a velocidades muy elevadas a través del núcleo del procesador. Además, características superiores de facilidad de uso como protección térmica, comprobación y corrección de errores, comprobación de redundancia funcional y el bus de administración del sistema ayudan a garantizar confiabilidad y tiempo de actividad máximos.

El procesador Pentium II Xeon ha sido mejorado para ofrecer un alto nivel de rendimiento para realizar tareas con grandes exigencias de cómputo en una arquitectura que ofrece escalabilidad y facilidad de uso.

- Incorpora una memoria caché L2 de 512 KB o 1 MB. La memoria caché L2 opera a la misma velocidad que el núcleo del procesador (400 MHz), lo que pone a disposición del núcleo del procesador una cantidad de datos sin precedentes.
- Comparte datos con el resto del sistema a través de un bus de sistema multitransacciones de alta capacidad de 100 MHz, otra tecnología de vanguardia que extiende el potencial de velocidad de procesamiento superior al resto del sistema.
- Se puede direccionar y asignar a caché un máximo de 64 GB de memoria para incrementar el rendimiento con las aplicaciones más avanzadas.
- El bus del sistema permite múltiples transacciones pendientes de ejecución para incrementar la disponibilidad de ancho de banda. También ofrece compatibilidad sin "suplementos" con un máximo de 8 procesadores. Esto hace posible el multiprocesamiento simétrico con cuatro y ocho procesadores a un bajo costo y ofrece un incremento de rendimiento significativo para sistemas operativos multitareas y aplicaciones con múltiples subprocesos.
- PSE36 - Es una expansión de la compatibilidad con memoria de 36 bits que permite a los sistemas operativos utilizar memoria por arriba de los 4 GB, lo cual incrementa el rendimiento del sistema para aplicaciones con grandes exigencias de lectura y espacio de trabajo grande.

PENTIUM III 23

23.1.- CARACTERÍSTICAS GENERALES

23.1.1.- Introducción.

Tras varios meses de retraso, el 26 de febrero de 1999, Intel sacó a la venta un nuevo procesador con dos velocidades: 450 y 500 Mhz. Ya se anunciaba el término **Katmai** como el nuevo microprocesador, pero al final decidió no mantener este nombre y lo bautizó como Pentium III. Es un procesador prácticamente igual al Pentium II pero se diferencia de él en que incorpora 70 nuevas instrucciones cuyo fin principal es mejorar la experiencia en las aplicaciones referentes a Internet y los sistemas 3D.

Se pueden distinguir tres modelos de Pentium III:

- **Katmai**: funciona con un bus de 100 Mhz y a una velocidad inferior por su caché de 512 Kb de segundo no integrada. Tecnología de 0'25 micras.
- **Coppermine**: caché se segundo nivel de 256 Kb integrada, con un bus de 100 a 133 Mhz y tecnología de 0'18 micras.
- **Tualatin**: tecnología de 0'13 micras con lo que se puede integrar en el chip una caché de segundo nivel de 512 Kb. Está protegido con una chapa metálica.

La principales características del nuevo procesador son:

- Optimizado para aplicaciones de 32 bits.
- Versiones desde los 400 Mhz hasta los 600 Mhz.
- Posee 32 KB de caché L1 repartidos en 16 KB para datos y los otros 16 para instrucciones.
- La caché L2 es de 512 Kb y trabaja a la mitad de la frecuencia del procesador.
- Velocidad a la que se comunica con la placa es de 100 Mhz.
- Incorpora 9'5 millones de transistores.
- Puede cachear hasta 4 Gb.
- Tecnología de fabricación de 0'25 micras.
- Totalmente compatible con el software actual basado en la arquitectura Intel.

23.1.2.- Arquitectura básica.

En la arquitectura del Pentium III podemos distinguir, al igual que en los modelos anteriores varias unidades o bloques físicos. Entre las que destacan por su novedad, distinguiremos:

1. Memoria del sistema.
2. Unidad de lectura de instrucciones.
3. Unidad de decodificación de instrucciones.
4. Unidad de predicción de saltos (BTB).

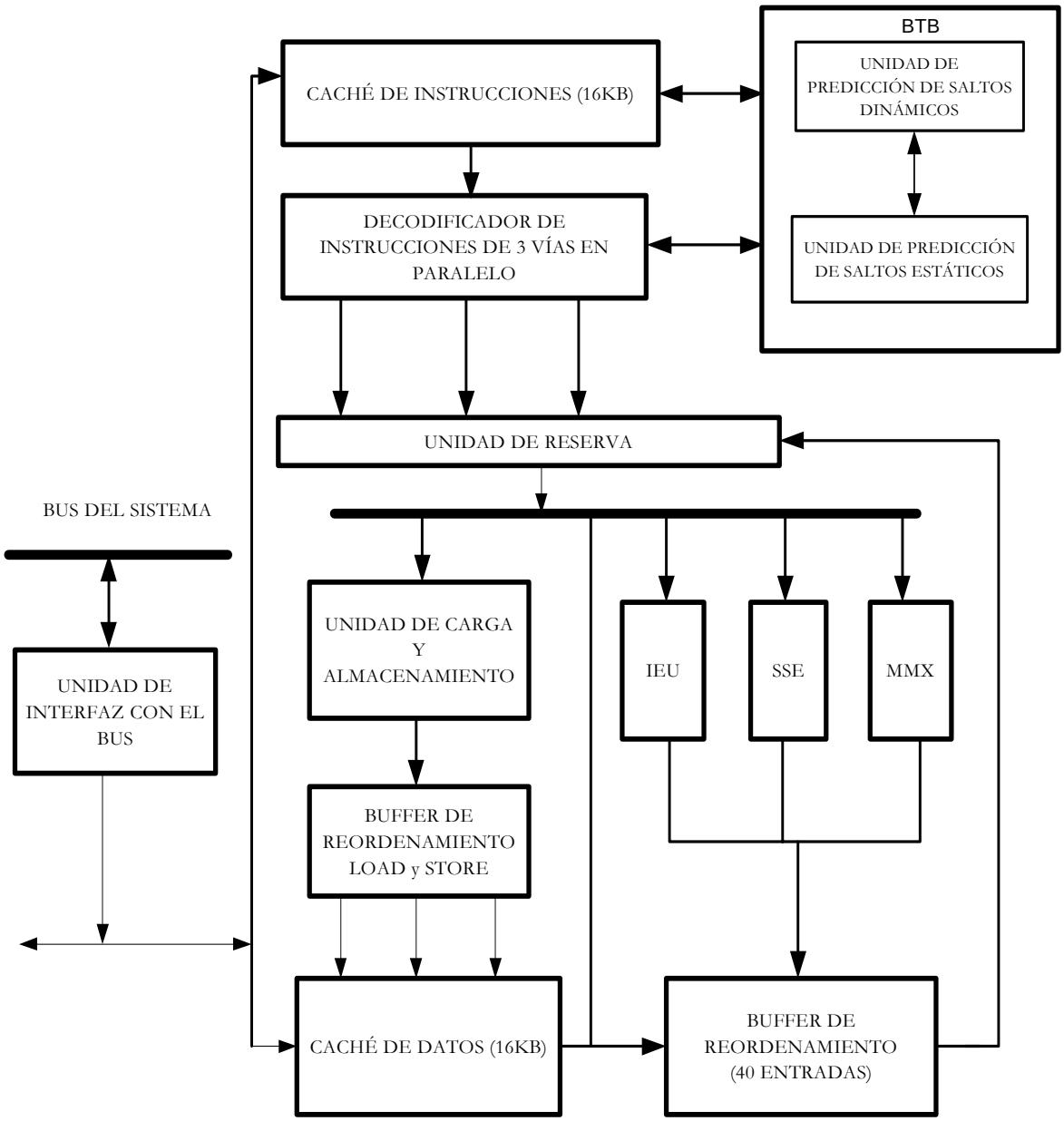


Figura 23.1 – Arquitectura del Pentium III

A continuación, detallaremos algunos de los bloques anteriormente citados de manera individual.

1.- MEMORIA DEL SISTEMA.

1.1.- Accesos a memoria: se controlan a través del bus del sistema y de la caché. Nos encontramos con un bus externo de memoria de 64 bits que maneja por separado cada una de las peticiones de acceso a memoria que recibe del microprocesador. Por otro lado tenemos el bus que conecta la caché de segundo

nivel con el microprocesador, de igual tamaño y funcionamiento que el anterior pudiendo procesar hasta cuatro accesos a la vez.

1.2.- Memoria Caché : El Pentium III posee dos cachés independientes de primer nivel integradas, una para instrucciones y otra para datos y una caché de segundo nivel también integrada. Ambas se detallarán más concretamente en apartados posteriores.

2.- UNIDAD DE LECTURA DE LAS INSTRUCCIONES.

La unidad de lectura de instrucciones está diseñada para leer una línea de caché de 32 bytes de código en cada ciclo, aunque hay documentos referentes a optimizaciones que contradicen esto y exponen que son 16 los bytes de código que se leen en cada ciclo. Esta unidad envía la lectura al decodificador de datos en bloques de 16B. Dependiendo del orden y longitud de las instrucciones que contenga ese bloque, la unidad de lectura podrá leer instrucciones adelantándose a la lectura de la unidad de decodificación lo que provocará una bajada del rendimiento al permitir la colocación de instrucciones erróneas y/o demasiado largas.

3.- UNIDAD DE DECODIFICACIÓN DE INSTRUCCIONES.

Es la encargada de traducir cada una de las instrucciones de la arquitectura Intel estándar en un conjunto de microoperaciones. Para ello dispone de tres componentes diferentes que se denominan decodificadores D0,D1 y D2. El primero de ellos puede decodificar instrucciones que generen hasta cuatro microoperaciones, mientras que los dos restantes solo son capaces de decodificar instrucciones que generen una única micro operación.

4.- UNIDAD DE PREDICCIÓN DE SALTOS.

4.1.- Predicción dinámica de saltos: la predicción dinámica de saltos se basa en el historial de los últimos saltos ejecutados en el microprocesador que se encuentran en el Branch Target Buffer (BTB). El BTB posee un buffer con 512 entradas agrupadas, en 32 conjuntos o set de 16 entradas (way), parecido a lo que ocurre en la caché.
Cuando se agota el espacio disponible en el BTB y es necesario introducir un nuevo salto, se descarta una de las entradas.
La primera vez que se ejecuta una instrucción de salto que no está almacenada en el BTB, se reserva espacio para mantener el historial de la misma. A menos que sea descartada por necesidad de espacio en el conjunto correspondiente de entradas en el BTB, permanecerá en el mismo de forma indefinida.
La predicción dinámica de saltos no predice donde va a saltar una instrucción, si no que si ya existe una entrada en el BTB para la instrucción de salto, se predecirá el comportamiento de la misma dependiendo de sus ejecuciones anteriores.
Este método de predicción de saltos posee muchas ventajas, como por ejemplo el poco tamaño que se necesita para almacenar cada una de las entradas del BTB (36 bits) o la gran cantidad de patrones que el algoritmo es capaz de predecir de forma correcta.

4.2.- Predicción estática de saltos: para esta función el microprocesador se fijará si existe una entrada en el BTB de una instrucción de salto. Esta unidad predice que se llevarán a cabo los saltos incondicionales, los saltos condicionales cuyo destino sea anterior a la instrucción de salto (bucles) y los saltos condicionales que no se llevarán a cabo por tener el destino posterior a la

instrucción de salto. Esta predicción actúa después de predicción dinámica por lo que se introduce un retardo adicional. Es decir, si la predicción es correcta se produce un retardo de seis ciclos y si es superior el retardo puede ser de hasta doce ciclos. El microprocesador dispone de una pila donde almacena hasta 16 direcciones de retorno en las llamadas a subrutinas.

23.1.3.- Orientación y mercado al que se dirige.

El Pentium III se centra principalmente en 2 campos: el de multimedia y el de Internet. Por ello esta complementado por un nuevo set de instrucciones que mejoran el desempeño de operaciones como la compresión de video para Internet y el reconocimiento de voz, al igual que el procesamiento geométrico 3D y la edición de imágenes 2D.

Las nuevas instrucciones, un total de 70, van orientadas hacia los usuarios que interactúan con Internet o que trabajan con aplicaciones multimedia con muchos datos, ya que incrementan notablemente el rendimiento y las posibilidades de las aplicaciones 3D, de tratamiento de imágenes, de vídeo, sonido y de reconocimiento de la voz.

El procesador Intel Pentium III ofrece excelentes prestaciones para todo el software del PC y es totalmente compatible con el software existente basado en la arquitectura Intel. El procesador Pentium III a 450 y 500 MHz amplía aún más la potencia de procesamiento al dejar un margen mayor en previsión de una futura exigencia de rendimiento en funciones relacionadas con Internet, comunicaciones y medios comerciales.

Dirigiéndonos a pequeñas y grandes empresas, los sistemas basados en el procesador Pentium III también incluyen las últimas funciones para simplificar la gestión del sistema y reducir el coste total de propiedad.

En conclusión, el software diseñado para este procesador posibilita por ejemplo el vídeo de pantalla completa y movimiento pleno, gráficos realistas y la posibilidad de disfrutar al máximo de Internet.

23.2.- APORTACIONES Y NUEVOS RECURSOS ARQUITECTÓNICOS

23.2.1.- Aportaciones generales.

Con el paso del Pentium II al Pentium III se suponía una mejora en algunas de las características básicas del procesador, pero como veremos a continuación las nuevas aportaciones con respecto a los anteriores modelos son escasas, incluso algunos expertos han llegado a asegurar que "simplemente, se trata de un Pentium II modificado para emplear un conjunto de instrucciones 3D".

En cuanto a su arquitectura interna, podemos decir que el microprocesador Pentium III es muy parecido a los anteriores procesadores (Pentium Pro, y Pentium II, la sexta generación de la familia 80x86). No obstante, se introducen algunas características nuevas que una vez integradas en la arquitectura interna común y con el cambio de la misma serán quienes proporcionen el aumento de velocidad respecto a los modelos anteriores.

Entre las principales características que se introducen y que comentaremos en apartados posteriores, podemos destacar:

- Nuevo conjunto de instrucciones MMX2.
- Nuevos registros internos que serán, renombrados para suplantar a los registros estándar.
- Extensiones Streaming.
- Introducción del número de serie.
- Predicción de saltos mejorada y optimizada.
- Ejecución especulativa de los saltos que se predicen.
- Ejecución no secuencial de las instrucciones.

Además de lo anterior, cabe destacar que proporcionan nuevos niveles de prestaciones para los servidores de gama alta basados en Intel con multiproceso de 4 y 8 vías. Estos modelos incrementan las prestaciones de las plataformas a 550 MHz y en el caso del modelo a 700 MHz consta de una amplia caché basada en la tecnología de fabricación de 0.18 micras de Intel, y ofrecen 1 ó 2 MB de memoria caché de segundo nivel (L2) incorporada en el chip (frente a los 256 Kb del Pentium Pro o los 512 Kb del Pentium II).

23.2.2.- S.S.E. (Extensiones Streaming SIMD)

A este procesador se le ha añadido las llamadas S.S.E. o "*Streaming SIMD Extensions"* (extensiones SIMD de flujo), durante mucho tiempo se conocieron como KNI ("*Katmai New Instructions"*) y posteriormente mucha gente ha preferido llamarlas, más comercialmente, MMX-2. El proceso que siguen estas instrucciones lo describe la propia palabra SIMD: "*Single Instruction, Múltiple Data";* instrucción única, datos múltiples. Es decir, permite realizar una única operación compleja con varios datos en vez de realizar varias operaciones más simples, pudiendo hacer hasta 4 operaciones en coma flotante por cada ciclo de reloj.

Son 70 nuevas instrucciones orientadas hacia tareas multimedia, especialmente en 3D equivalentes a las 3DNow que implementa AMD. Algunas de las ventajas que se obtienen de las S.S.E son:

- Posibilidad de utilizar simultáneamente las nuevas instrucciones con la FPU o junto con las MMX sin verse penalizado por ello.
- Trabajar con operandos de coma flotante de simple precisión.
- Aceleración de gráficos 3D.
- Reproducción de vídeo y sonido digital de alta calidad, codificación y decodificación.
- Tratamiento de imágenes de mayor resolución y calidad.
- Reconocimiento de voz.
- Posibilidad para los programadores de lenguaje ensamblador de realizar manipulaciones directas tanto en la caché de primer nivel como en la de segundo nivel.

Además de estas ventajas también tenemos inconvenientes. El principal es que para conseguir un aumento de rendimiento, las aplicaciones deberán estar optimizadas para las nuevas instrucciones.

La introducción de la tecnología S.S.E. supone la aparición de 8 nuevos registros de 128 bits capaces de contener cada uno de ellos cuatro operandos en formato de coma flotante de simple precisión. (Estos nuevos registros se nombrarán más adelante).

23.2.3.- Número de serie del procesador Intel.

Una de las novedades antes citadas ha sido la introducción de un número de serie vía software en cada PC. Este número es programado durante el proceso de fabricación del procesador Pentium III, con lo que se obtiene una especie de "carné de identidad" único para cada PC. Su objetivo principal es Internet y lo que esto abarca:

- Realizar transacciones más seguras a través de Internet.
- Facilitar el control a los administradores de redes.
- Disminuir al máximo el riesgo de error y de fraude.
- Manejar aplicaciones que utilicen funciones de seguridad: acceso gestionado a nuevos contenidos y servicios de Internet, intercambio de documentos electrónicos, etc.
- Manejar aplicaciones de gestión: de activos, carga y configuración remotas del sistema.

Esta característica del procesador ha sido duramente criticada por un amplio colectivo que la considera un atentado contra el derecho a la intimidad y una invasión de la privacidad. Por esta razón Intel se vio obligado a ofrecer alguna forma de desactivar el número de serie al usuario del ordenador.

Según Intel, el código de serie viene activado por defecto pero una vez desactivado por software es necesario un RESET físico de la máquina para que vuelva a activarse. El programa para la desactivación ya existe en Internet y es ofrecido directamente por Intel.

La forma de leer el número de serie consiste en ejecutar la instrucción CPUID con el valor 3 en el campo MX y así obtenemos el número de 96 bits que identifica al procesador. Más concretamente, los pasos exactos para conocer el número de serie del procesador serían:

- Ejecutar CPUID con EAX=0 sirve para verificar si el procesador es Intel.
- Ejecutar CPUID con EAX=1 informa si el usuario ha desactivado el número de serie y verifica si el bit 18 del registro EDX está activado.
- Si todo ha salido bien, ejecutar CPUID con EAX=3 y el número de serie estará en los registros ECX y EDX.

23.2.4.- Caché de transferencia avanzado.

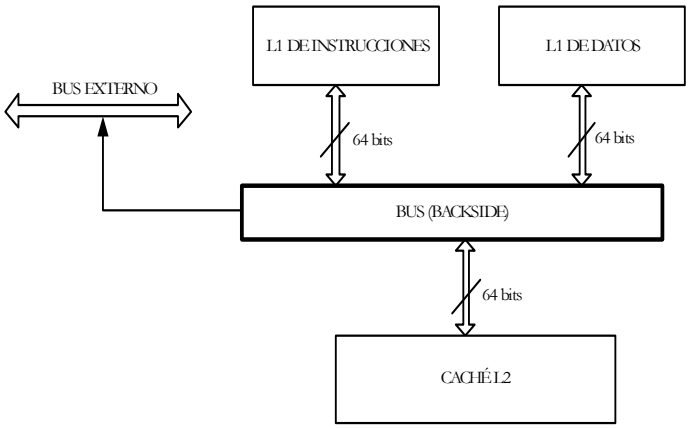


Figura 23.2 – Arquitectura de la caché

El procesador Pentium III incorpora 32 K de caché de nivel 1 sin bloqueo y 512 K de caché de nivel 2 integrada, también sin bloqueo y que funciona a la mitad de la velocidad interna del micro (como ocurría en el Pentium II). Así mismo dispone de un bus dedicado denominado Backside Bus.

La caché de primer nivel L1, está dividida en dos cachés separadas, una para código o instrucciones y otra para datos. El tamaño depende del modelo del microprocesador pudiendo ser de 8 Kb o de 16 Kb, siendo más común la caché de 16 Kb.

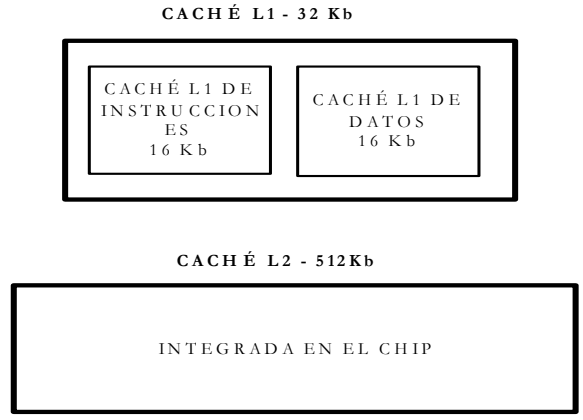


Figura 23.3 – Estructura de la caché de primer nivel y de segundo nivel.

En los ejemplos nos referiremos a la caché de 16Kb teniendo en cuenta que los cálculos serán similares para la de 8 Kb (la mitad).

El comportamiento de la caché de instrucciones o de código respecto a la memoria es del tipo asociativa de 4 vías y el comportamiento de la caché de datos es del tipo asociativa de 2 vías.

El Pentium III transforma cada instrucción en un conjunto de microoperaciones simples. Los 16 Kb de caché de datos, se dividen en 512 grupos de 32 bytes cada uno. Cada uno de estos grupos se denominan "líneas de caché" ("Cache lines"). Cada vez que se introduzcan datos en la caché desde la memoria principal del sistema, se rellenará una de estas líneas por lo tanto la caché sólo puede recibir datos en bloques de 32 bytes cada uno.

Estas 512 líneas de caché se emparejan en la caché de datos de dos en dos (asociativa de 2 vías), formando 256 bancos de caché. Cada dirección de memoria, múltiplo de 32 bytes, posee asociado un banco, y entre toda la memoria disponible en el sistema, existen varias direcciones de memoria que poseen el mismo banco. De todas las direcciones de memoria que tengan asociado el mismo banco, sólo dos de ellas podrán estar simultáneamente en la memoria caché de primer nivel de datos.

Los 16 Kb de caché de código disponibles se dividen también en 512 líneas de caché. La diferencia es que dichas líneas de caché se agrupan de cuatro en cuatro (asociativa de 4 vías).

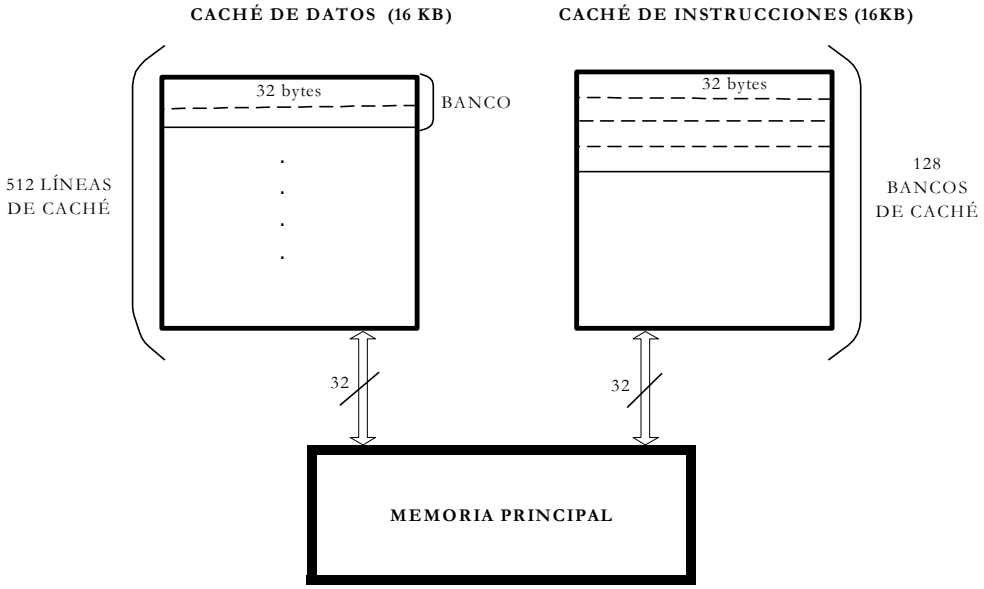


Figura 23.4 – Arquitectura de la caché (L1).

Como es normal, en la caché de datos de primer nivel no se puede albergar toda la memoria del ordenador. Por ello, cuando se lee una línea de caché en un banco que ya posee sus dos líneas ocupadas, se descarta el contenido de una de las dos líneas, que es reemplazado con el contenido de la nueva posición de memoria. La línea de caché que se descarta es aquella que haya permanecido más tiempo sin ser usada (se usa el algoritmo de reemplazo LRU: Least Recently Used).

Existen varias formas de controlar la asignación de las líneas de caché a la memoria:

- **No cacheable o Uncacheable (UC) :** los datos no se introducen directamente en la memoria caché. Todos los accesos a la memoria se realizan directamente sobre el bus externo. Este tipo de memoria es adecuada para los dispositivos E/S.

- **Escritura Diferida o Write-Combining (WC)**: los datos no se introducen en la caché sino que en unos registros intermedios y no se permiten consultas. Este tipo de memoria es adecuada para memoria de video.

- **Escritura directa, inmediata o Write-Through (WT):** se introducen en la memoria caché tanto las lecturas como las escrituras a la memoria del sistema. Cuando se intenta leer una zona de memoria que no se encuentra en la caché, se rellena una nueva línea de la misma. Cuando se realizan escrituras, se actualizan tanto la memoria caché, como la memoria del sistema. Se permite emplear también Write-Combining.

- **Escritura obligada o Write-Back (WB):** Se introducen en la memoria caché tanto las escrituras como las lecturas a la memoria del sistema. Cuando se intenta leer una zona de memoria que no se encuentra en caché, se rellena una nueva línea de la misma. Cuando se realizan escrituras, si la zona de memoria no se encuentra actualmente en la caché, se rellena una nueva línea. Las escrituras siempre se envían a la memoria caché y cuando una línea de la misma que ha sido modificada es descartada, se rescribe la línea en la memoria del sistema. Se permite emplear Write-Combining. Este tipo de memoria es el que tiene el mejor rendimiento.

- **Escritura protegida o Write-Protected (WP)**: las lecturas de la memoria del sistema se llevan a cabo desde la memoria caché. Cuando se intenta leer una zona de memoria que no se encuentra en la caché, se rellena una nueva línea de la misma. Cuando se realizan escrituras, se envían directamente a la memoria del sistema y, si existe, se invalida la línea de caché correspondiente.

23.3.- NUEVAS INSTRUCCIONES.

La incorporación de 70 nuevas instrucciones es lo que distingue al Pentium III del Pentium II. Como se ha dicho anteriormente, estas instrucciones van dirigidas a mejorar la velocidad en el procesamiento de imágenes 3D y 2D, reconocimiento de voz... es decir multimedia. Estas 70 instrucciones se pueden dividir en 3 grupos:

- 8 instrucciones que mejoran el acceso a memoria, para cachear memoria, especialmente para manejar muchos datos , como el reconocimiento de voz o los vectores de datos 3D.

- 12 instrucciones específicas para multimedia, para tareas como optimizar el proceso de datos de audio o para mejorar las representaciones MPEG-2. Estas instrucciones complementan a las 59 MMX ya existentes.

- 50 instrucciones para el manejo de datos en coma flotante. Especialmente diseñadas para el proceso de datos tridimensionales. Pueden producir hasta 4 resultados por ciclo de reloj.

Algunas de estas instrucciones van encaminadas a realizar accesos comunes durante la descompresión de vídeo digital en formato MPEG-2 como puede ser el poder solicitar que el contenido de una determinada posición de memoria se deposite en la caché del procesador antes de que vaya a utilizarse, mejorándose de esta forma los accesos a bloques de memoria que se usen con mucha frecuencia

.

Ejemplo de alguna de las nuevas instrucciones pueden ser:

- PMAX, PMIN: instrucciones para reconocimiento de habla: "Viterbi-Search algorithm"
- PAVG: instrucción destinada a la decodificación de video.

- PSADBW (Suma de Diferencias Absolutas) (para codificar MPEG): Calcula el valor absoluto de la resta entre cada uno de los bytes empaquetados en el registro MMX origen y el registro MMX destino. Después, calcula la suma de todos estos valores y la almacena en la palabra empaquetada más baja del registro MMX destino, poniendo los demás bits a cero.

Para el correcto funcionamiento de estas instrucciones es conveniente asegurarnos que el microprocesador soporte este juego de instrucciones, así como de que el sistema operativo bajo el que estamos ejecutando nuestras aplicaciones las haya habilitado.

23.4.- NUEVOS REGISTROS.

Pentium III introduce 8 nuevos registros de 128 bits (frente por ejemplo los 64 bits del MMX) y tiene la posibilidad de empaquetar cuatro números en coma flotante.

Lo primero que necesitamos es añadir un conjunto de registros que trabaje con datos empaquetados en coma flotante. Podría haberse optado por la misma solución que en el MMX que recordando, empleaba la pila de registros del coprocesador matemático. Sin embargo, hay algunas razones para optar por introducir un nuevo conjunto de registros y la principal es que los registros del coprocesador son de 64 bits (MMX) y como sabemos, si queremos representar números en coma flotante necesitamos al menos 32 bits (simple precisión) con lo que solo podríamos empaquetar 2 números por registro en lugar de los 4 que se obtienen con este método.

Los 8 nuevos registros tienen nombres que van desde XMMO hasta XMM7. Hay que destacar que estos nuevos registros sólo se utilizarán para realizar cálculos, no para realizar direccionamientos, para estos casos se emplean los registros de la arquitectura estándar. Estos nuevos registros también reciben el nombre de registros de extensión.

Además de los anteriores se incluye un registro de estado: el MXCSR de 32 bits. Este registro de estado se encarga de controlar la máscara de excepciones, de notificar las excepciones ocurridas y de controlar los métodos de redondeo que se emplean.

23.5.- ANÁLISIS DEL RENDIMIENTO.

Intel esperaba que con este nuevo Pentium el rendimiento fuese mucho mayor del que se ha obtenido. El problema ha sido que al aumentar la velocidad de reloj han tenido que reducir la velocidad de la caché. En realidad se puede apreciar un mayor rendimiento cuando se activan las nuevas instrucciones (Extensiones Streaming SIMD).

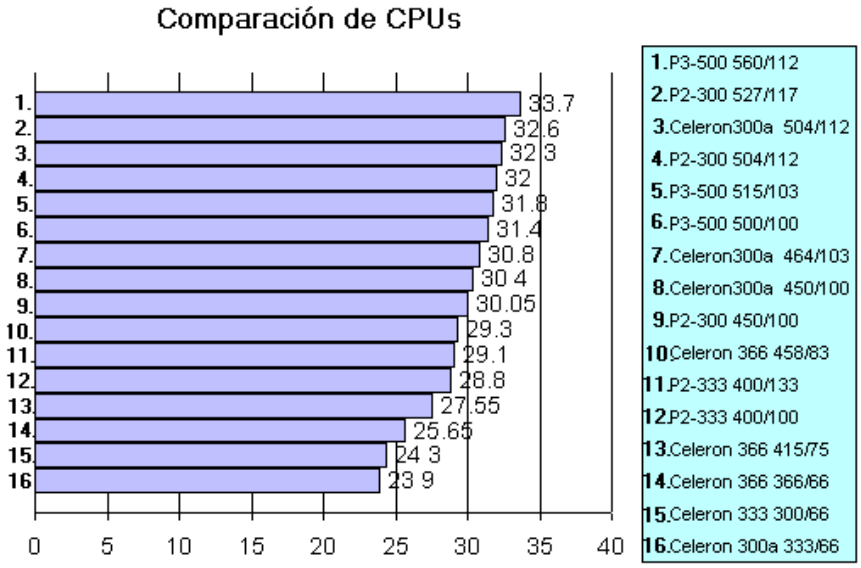


Figura 23.5 – Análisis de CPUs

En determinados programas de prueba de 3D y multimedia ha sido donde el Pentium III ha demostrado ventajas sustanciales en cuanto a rendimiento. Algunas de los programas donde se han mostrado este aumento de rendimiento frente al Pentium II han sido tales como MultimediaMark o Winbench.

Por lo tanto, gracias a esto y a las nuevas instrucciones que incorpora el Pentium III, usuarios software como pueden ser Adobe Photoshop, Corel PhotoDraw o Microsoft Office 2000, obtendrán, como bien se muestra en el gráfico (Figura 23.4) una sustancial mejora en los tiempos de ejecución de sus trabajos. Y en lo que respecta a los fabricantes hardware, han diseñado nuevas líneas compatibles basadas en estos microprocesadores. Entre otros mencionaremos a Hewlett-Packard, IBM, Dell Computer, NEC ...

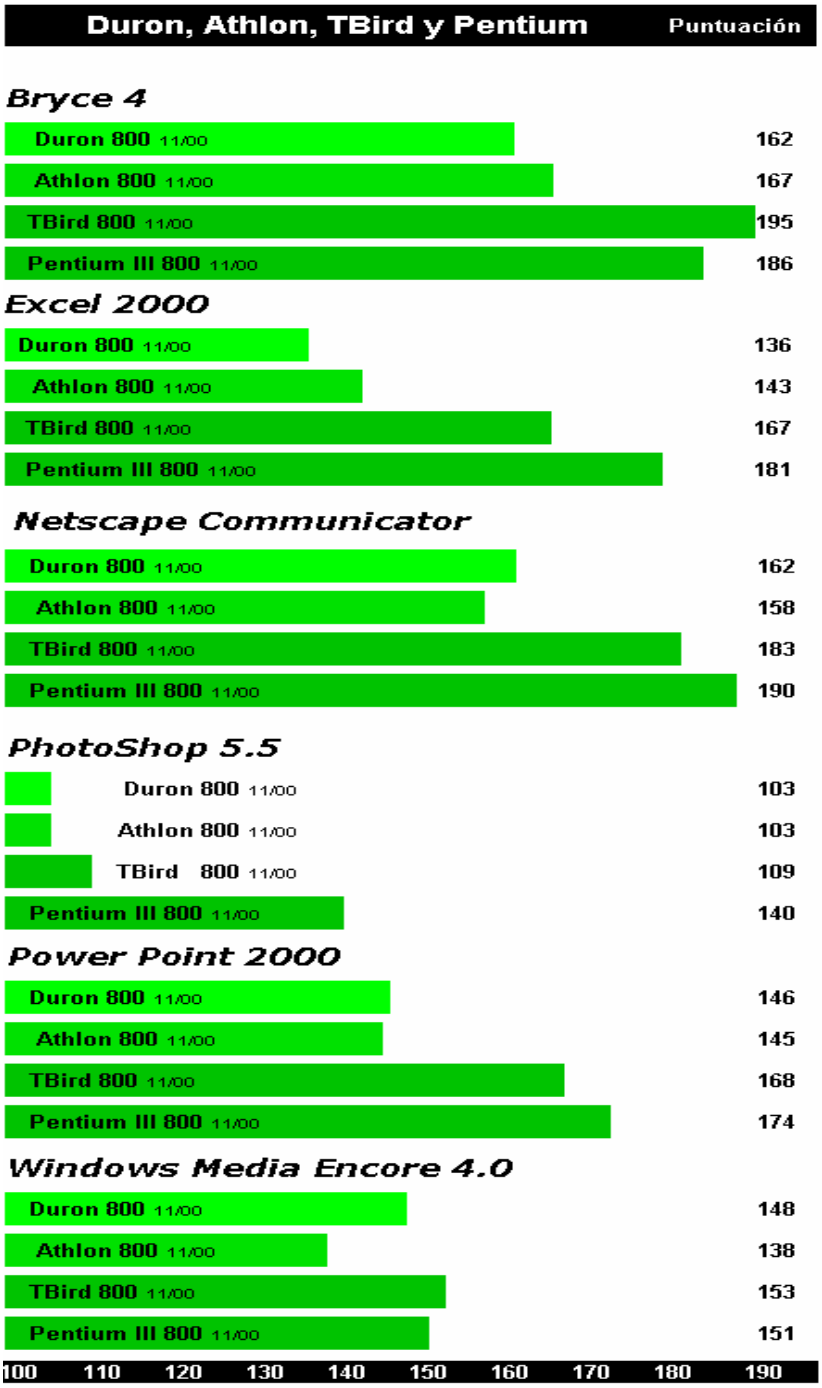


Figura 23.6 – Análisis de Rendimiento

23.6.- K7 ATHLON DE AMD.

Durante mucho tiempo el mercado de los microprocesadores estaba copado en su mayoría por los microprocesadores de Intel, pero con el tiempo AMD ha ido ganando terreno. Toda la competencia que daba AMD a Intel antes del K5 se basaba en simplemente ofrecer lo mismo pero a menor precio, y además con un retraso considerable de tiempo del chip, aunque ya entonces comenzaba a plantar cara directamente al Pentium, aunque sin excesivo éxito.

Cuando el K6 hizo su aparición, el microprocesador contaba con unas instrucciones específicamente diseñadas para acelerar aplicaciones multimedia, pero en este caso no eran como las MMX de Intel sino que eran realmente unas instrucciones muy potentes, basadas en arquitecturas vectoriales. Estas instrucciones, llamadas 3DNow! pusieron por primera vez en la historia a AMD por delante de Intel en el mercado, y también por primera vez los fabricantes de software dieron su apoyo prácticamente masivo a este nuevo set de instrucciones, que actualmente se ha convertido prácticamente en un estándar.

La novedad que se produjo con la aparición del Pentium III fue que por primera vez era Intel quien imitaba a AMD, con unas instrucciones similares a 3DNow! (las ya comentadas instrucciones SSE). Entonces era el turno de AMD que planteó una verdadera revolución con la aparición del K7 Athlon. Este micro fue la gran apuesta de AMD ya que es un micro con una arquitectura totalmente nueva, que le permitía ser el más rápido en todo tipo de aplicaciones.

Podemos considerar por tanto al procesador K7 Athlon de AMD como el primer procesador capaz de plantar cara al Pentium III no solo en prestaciones incorporadas sino que también en precio y rendimiento.

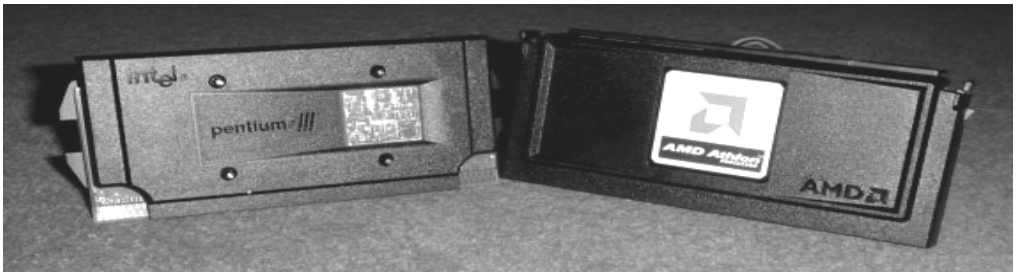
El primer modelo K7 Athlon apareció a finales de Mayo de 1999 e incorpora algunas novedades respecto al Pentium III y a los anteriores modelos de AMD, de las que destacaremos las principales:

- La caché de primer nivel (L1) tendrá 128 KB, repartidos entre 64 KB para datos y 64 KB para instrucciones.
- La caché de segundo nivel (L2) será muy flexible. En principio viene con una caché de 512 KB, pero AMD planeó versiones del K7 con 2 MB, pudiendo llegar a los 8 MB.
- La velocidad de esta cache (L2) variará entre 1/3 de la frecuencia del micro hasta la misma frecuencia. Esta flexibilidad en la caché de segundo nivel permitirá a AMD ofrecer varias líneas de su K7, para diferentes tipos de usuarios variando desde el nivel domestico hasta servidores de altas prestaciones.
- El proceso en coma flotante ha sido siempre la asignatura pendiente de AMD pero esto cambió con el K7. El AMD K7 dispone de 3 Líneas de ejecución o pipelines paralelas lo que supone una gran mejora en este tipo de operaciones.

Realizando una breve comparación entre el Pentium III y el AMD K7 Athlon podemos destacar entre las diferencias:

	Pentium III	**AMD K7 Athlon**
Velocidad máxima de reloj	600 MHz	650 MHz
Caché L1	16 KB + 16 KB	64 KB + 64 KB
Caché L2	Integrada 512 KB	Externa 512 KB → 8 MB
Bus escritura Caché	133 MHz – 1GB/seg	133 MHz – 2.1 GB/seg
Bus del sistema	100 MHz	200 MHz
Instrucciones SIMD	MMX II	3DNow!

Su único y mínimo inconveniente radica en que el K7 de Athlon necesita placas base específicamente diseñadas para él, debido a su novedoso bus de 200 MHz y a su método de conexión, el "Slot A". El "Slot A" es físicamente igual al Slot 1 de Intel, pero incompatible con él, debido, entre otras cosas, a que Intel no quiso dar licencia a AMD para utilizarlo. Es por ello que si queremos cambiar el microprocesador, deberemos cambiar también la placa base.



23.7 – Imagen de los procesadores Pentium III y AMD K7 Athlon

23.7.- PENTIUM III XEON

23.7.1.- Aportaciones y nuevos recursos arquitectónicos

Tras la aparición del Pentium III, y tras la realización de unas pequeñas modificaciones que comentaremos a continuación, Intel comercializó El Pentium III Xeon.

Este modelo fue el primer modelo de procesador a 700 MHz con amplia caché basado en la tecnología de fabricación de 0.18 micras.

Ofrece como novedad más importante 1 o 2 MB de caché de segundo nivel (L2) incorporada en el chip.

23.7.2.- Orientación y mercado al que se dirige

Estos procesadores han sido diseñados para satisfacer las necesidades de escalabilidad, disponibilidad y administrabilidad del mercado de los servidores.

Actualmente existen en el mercado las denominadas empresas de "tercera generación" que están revolucionando el mercado tecnológico tradicional (Internet). Este tipo de empresas se caracterizarán por su capacidad de suministrar información detallada y personalizada tanto a clientes corporativos como individuales, debiendo integrar eficazmente su información con la de los clientes y proveedores con el fin de crear una experiencia comercial positiva.

Lo que Intel pretende con este modelo es que estas empresas sean capaces de acelerar, automatizar y optimizar sus sistemas comerciales, además de entregar información que ayude a los clientes y proveedores a acelerar, automatizar y optimizar sus procesos de toma de decisiones.

PENTIUM 4 24

24.1- CARACTERÍSTICAS GENERALES

24.1.1- Introducción

Intel tras algunos retrasos lanzó el 20 de noviembre del año 2000 el Pentium 4, antes denominado Willamette. Este procesador proporciona altas prestaciones para procesar vídeo y audio, explotando las últimas tecnologías de Internet, visualizado de gráficos en 3-D, videojuegos, CAD, ...

24.1.2- Arquitectura básica

La microarquitectura NetBurst le permite al Pentium 4 funcionar a velocidades extremadamente altas, aportando grandes prestaciones a usuarios de ordenadores, además esta tecnología dará potencia a los más avanzados procesadores de 32 bits de Intel en los próximos años. El Pentium 4 es el primero en incorporar un diseño totalmente nuevo, desde que Intel introdujera el Pentium Pro, con su microarquitectura P6.

El Pentium 4 se conecta a placa a través de Socket 423, con lo que abandona la conexión Slot de anteriores procesadores.

La memoria RAM utilizada es ahora de tipo RIMM que trabaja a elevada velocidad llegando a los 400 MHz en el bus del sistema, sin embargo tiene como inconveniente el alto coste. La memoria caché tiene 20 KB de primer nivel donde 12 KB son para instrucciones y 8 KB para datos, en el Pentium III la caché L1 era de 32 KB, por lo que en el Pentium 4 se ve reducida. La caché de segundo nivel también se reduce pasando de 512 KB a 256 KB.

La tecnología de fabricación utilizada es de 0,18 y 0,13 μm. El núcleo del procesador integra 42 millones de transistores.

El rango de frecuencias en el que esta disponible este procesador va desde 1,7 GHz hasta 2,8 GHz, siendo las velocidades en que puede trabajar el bus del sistema de 400 ó 533 MHz dependiendo de la velocidad del procesador.

24.1.3- Orientación y mercado al que se dirige

El Pentium 4 está orientado hacia un mercado doméstico altamente exigente, por lo que no pretende introducirse en otros campos que ya están cubiertos con procesadores como Xeon o Itanium, prueba de ello es que no hay planes de establecimiento de plataformas multiprocesador con Pentium 4.

24.2- APORTACIONES Y NUEVOS RECURSOS ARQUITECTÓNICOS

La microarquitectura del Pentium 4 se ha diseñado partiendo casi de cero. En concreto, se basa en la nueva arquitectura NetBurst cuyos pilares se describen a continuación.

24.2.1- Tecnología hipersegmentada.

Dentro de un microprocesador, los datos pasan por "pipelines" (canales de datos), de un número determinado de etapas. En un Pentium con arquitectura P6 (Pentium Pro, Pentium II, Pentium III y Celeron), el pipeline tiene 10 etapas; en el Pentium 4 hay 20 etapas. Cuantas más etapas, más se tarda en "liberar" los datos, por lo que un número excesivo de etapas puede llegar a bajar el rendimiento del ordenador. Sin embargo, esto tiene una ventaja, al Pentium 4 le permite alcanzar mayores velocidades de reloj (más MHz), que es lo que busca Intel, a costa de perder parte del rendimiento para poder recuperarlo a fuerza de GHz.

24.2.2- Bus de Sistema de 400 MHz.

Es una de las mejores características de esta arquitectura. En realidad el bus del sistema no funciona a 400 MHz "físicos" (reales), sino a 100 MHz cuádruplemente aprovechados con una especie de "doble DDR", como se realiza con la tecnología AGP 4X; por ello, el multiplicador a seleccionar en la placa para el modelo de 1,4 GHz es 14x y no 3,5x.

Estos 400 MHz mejorarán el rendimiento de aplicaciones profesionales y multimedia (como renderizado y edición de vídeo), y de muchos juegos 3D.

La tasa de trasferencia que se alcanza son 3,2 GB/s, que es significativamente superior a los modelos anteriores de Intel. El Pentium III con bus a 133 MHz ofrece una tasa de 1 GB/s y el Celeron con su bus a 66 MHz ofrece 0,5 GB/s.

24.2.3- Rapid Execution Engine.

Otra de las novedades de esta arquitectura del Pentium 4 es la capacidad de dos unidades aritmético-lógicas de números enteros (ALUs) que consiguen tiempos de espera iguales a un semiciclo de reloj en la ejecución de algunas instrucciones con lo que el procesador estaría funcionando al doble de velocidad. Aunque esta capacidad parece muy atractiva para aplicaciones no matemáticas, no consigue plenamente sus objetivos debido a problemas con el exceso de etapas.

24.2.4- Caché y otras características.

La caché L2 está integrada en el micro y tiene un bus de datos de 256 bits, esto forma parte de la mejora de la tecnología "Advanced Transfer Caché" estrenada con el Pentium III pudiéndose alcanzar 48 GB/s de tasa de transferencia en el modelo de 1,5 GHz. Esto representa el doble de lo que puede hacer un Pentium III a la misma velocidad, y es mucho más de lo que puede alcanzar un AMD Athlon, sobre todo porque en éste la caché L2 tiene un bus de sólo 64 bits.

En cuanto a Execution Trace Caché y Advanced Dynamic Execution, son técnicas que mejoran la ejecución especulativa, se puede romper el orden de las instrucciones para acelerar su procesamiento y refuerzan la predicción de saltos y ramificaciones (branch prediction).

En la siguiente figura se muestra la distribución dela memoria caché en la micorarquitectura NetBurst:

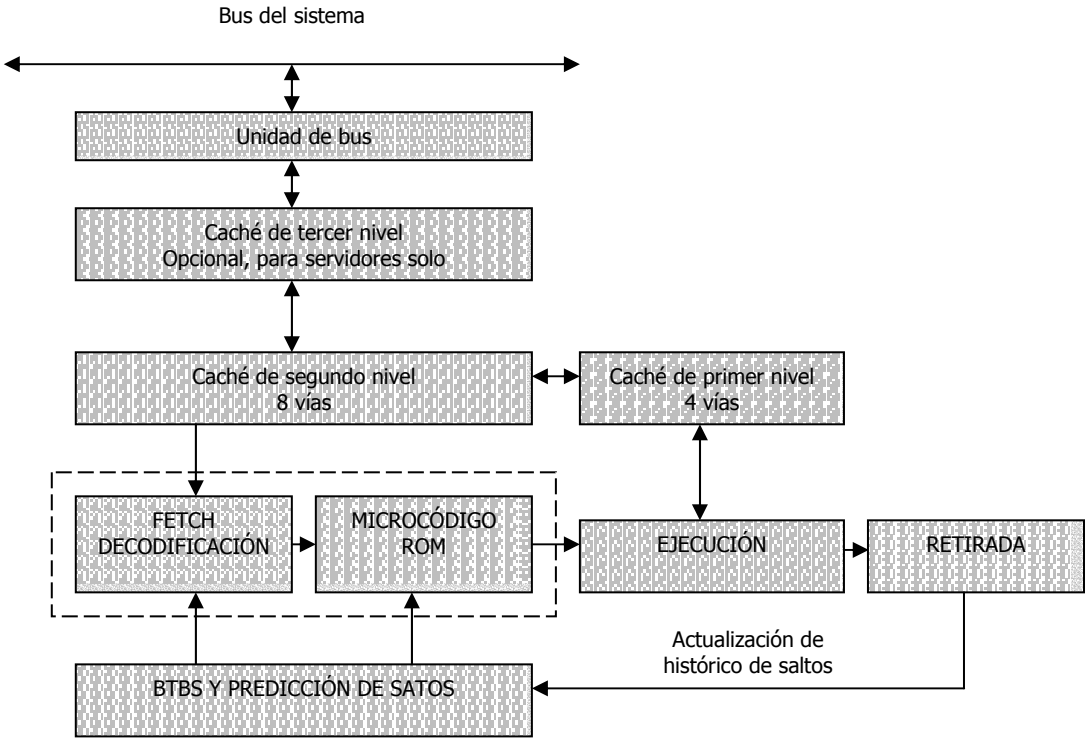


Figura 25.1. Distribución de la caché en el Pentium 4

24.2.5- Descripción de la arquitectura.

Sobre los componentes del Pentium 4 destaca el **BTB** (Branch Target Buffer) que es la parte encargada de guardar las direcciones de los saltos y de predecirlos. Tras pasar éste módulo la instrucción va al **Decodificador** que la convierte de formato x86 en varias microinstrucciones. Más tarde se pasa la instrucción a la zona de **Renombramiento/Reposicionamiento** que sirve para ejecutar varias instrucciones simultáneamente, siendo necesario que sean minimamente independientes.

Más tarde se pasa a las **Colas de microinstrucciones**: las cuales almacenan las mini-instrucciones pendientes de ejecutar. Para finalizar, se pasa a la zona **Store/Load AGU** compuesta de dos unidades que se encargan de guardar (Store) y cargar (Load) datos, desde y hacia, la memoria o en su defecto la caché. (AGU significa Adress Generation Unit, unidad generadora de direcciones de memoria).

La arquitectura del Pentium 4 se detalla en la siguiente figura:

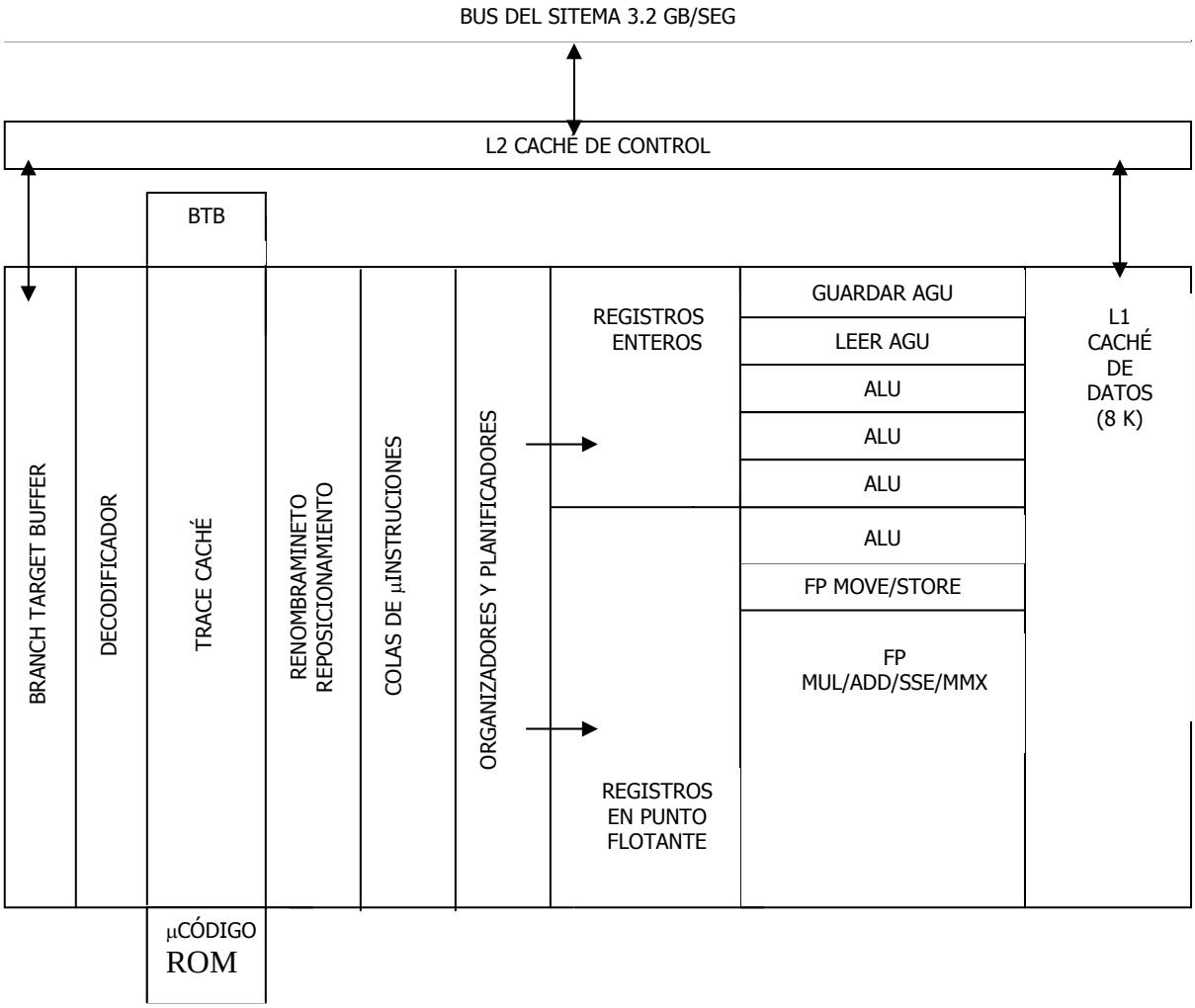


Figura 25.2. Esquema de la arquitectura del Pentium 4.

24.3- NUEVAS INSTRUCCIONES

Las nuevas instrucciones del Pentium 4 pueden llegar a ejecutarse en programas, sobre todo de carácter matemático, entendiendo por esto a renderizados, juegos 3D, compresión y descompresión de audio y vídeo y cálculos matemáticos con funciones complejas.

Para enfrentarse a ello el Pentium 4 tiene cuatro posibilidades: utilizar la unidad de coma flotante (la FPU), utilizar las instrucciones MMX, utilizar las SSE (Streaming SIMD Extensions, introducidas con el Pentium III), o la gran novedad del Pentium 4: las instrucciones SSE2 (Streaming SIMD Extensions 2).

Las SSE2 constan de 144 nuevas instrucciones de tratamiento de datos enteros y reales en simple y doble precisión. Además, el tamaño de los operandos es de 128 bits, duplicando la longitud de palabra, y teóricamente el rendimiento de las operaciones con MMX o SSE., algunas capaces de manejar cálculos de doble precisión de 128 bits en coma flotante. La idea de estas instrucciones, es reducir el número de operaciones necesarias para realizar las tareas.

Las ventajas que se obtienen con SSE2 son un aumento del rendimiento en la reproducción de audio y video. Además, facilita la codificación y cifrado de información con claves de gran longitud, así como la representación y el modelado en tiempo real de técnicas geométricas avanzadas en tres dimensiones.

El inconveniente del juego de instrucciones SSE2 es que sólo puede ser utilizado mediante software específicamente preparado para ello, requiere software optimizado y la mayoría de las aplicaciones no están preparadas para ello.

En la siguiente figura se muestra una típica operación SIMD, cuyo funcionamiento es igual al SSE2. En esta instrucción hay dos paquetes de cuatro datos con los siguientes elementos (X1, X2, X3, y X4, y Y1, Y2, Y3, y Y4) que se operan en paralelo. Los resultados la operación en paralelo se ordena como un paquete de cuatro elementos.

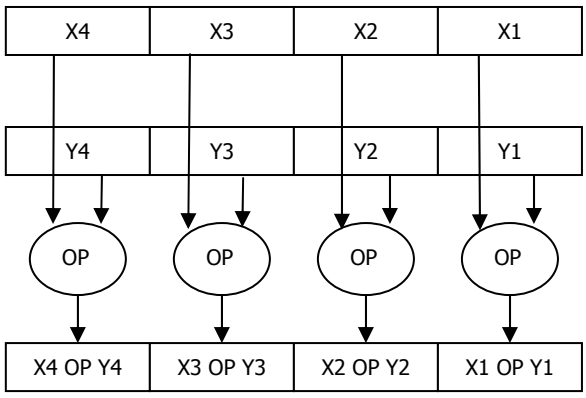


Figura 25.3. Esquema de operación SIMD.

Los registros sobre los que se opera con tecnología SIMD (la tecnología de MMX, SSE, y SSE2) da al programador la habilidad de desarrollar algoritmos en que pueden combinar los funcionamientos con números de 64 bits y del 128.

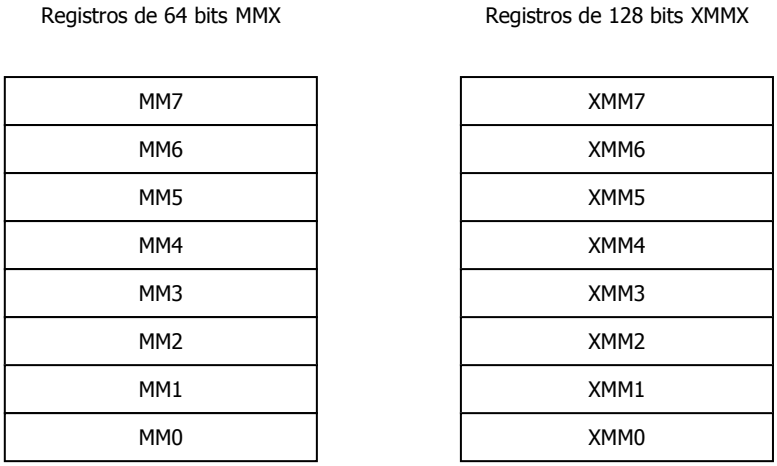


Figura 25.4. Registros para operaciones SIMD.

24.4- ANÁLISIS DEL RENDIMIENTO

Al evaluar la actuación de un microprocesador, se deben tener en cuenta sus principales características y también para lo que va a ser utilizado (Productividad, Calculo intensivo, Internet y Multimedia).

Habiendo estudiado las características técnicas del Pentium 4, se puede conocer hasta donde llegan sus prestaciones observando su rendimiento con el resto de procesadores del mercado.

Para observar el rendimiento del Pentium 4, ha sido comparado con los procesadores Pentium III a 1 GHz y AMD Athlon a 1,2 GHz. Además el Pentium 4 se ha ejecutado con los sistemas operativos Windows 2000 y Windows 98. Los programas de prueba utilizados han sido el Sysmark 2000 y paquetes de creación de contenidos de Internet y ofimática.

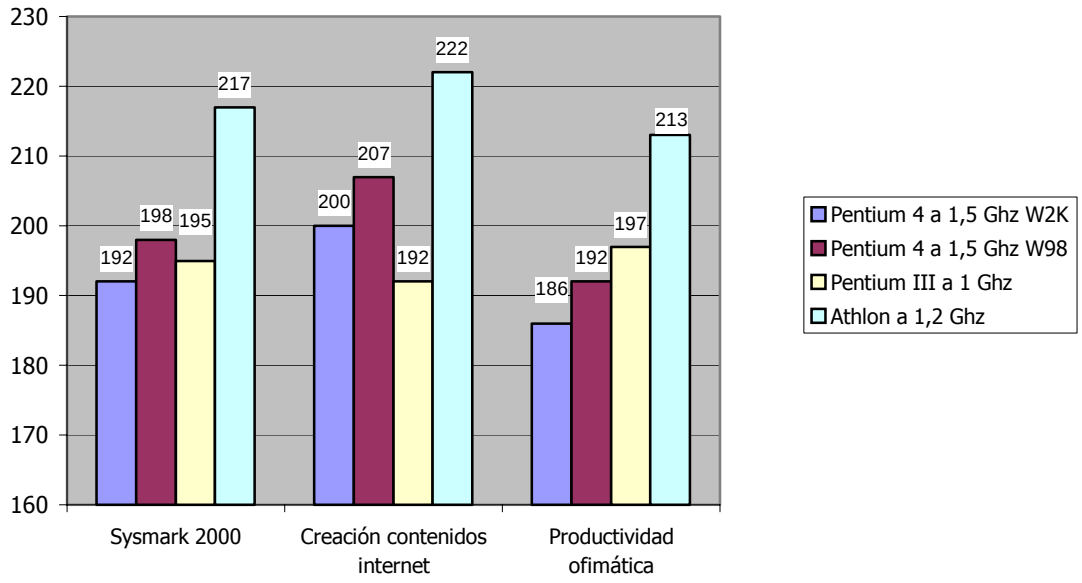


Figura 25.5. Comparativa de procesadores.

Atendiendo al gráfico mostrado se llega rápidamente a la conclusión de que el Pentium 4 es un fracaso. La inferioridad que muestra en aspectos tan importantes como la productividad ofimática con respecto a sus competidores le deja en una clara desventaja en el mercado. Así mismo los excelentes resultados que ha conseguido su gran competidor, el AMD Athlon, le otorgan una gran ventaja competitiva.

Sin embargo, el Pentium 4 ha sido desarrollado para ocuparse de un tipo de procesamiento cada vez más habitual como es la multitarea y el tratamiento masivo de datos, con lo que las aplicaciones futuras correrán, según Intel, a mayor velocidad que en los procesadores actuales.

ITANIUM 25

25.1- INTRODUCCIÓN

IA-64 (Itanium) es la familia de procesadores con la que Intel se introduce al mercado de los procesadores de 64 bits. El primer producto de esta familia de procesadores es, precisamente, el Itanium que se introduce en el mercado en mayor del 2001.

Se ha diseñado para ser utilizado en servidores y workstations de alto rendimiento. Intel afirma que su diseño va más allá de conceptos como CISC o RISC mediante la incorporación de una cantidad masiva de recursos de procesamiento con compiladores inteligentes que permiten la producción de código objeto que hace explícito el paralelismo.

Una de las distinciones más importantes entre el Itanium y los microprocesadores de 32 bits es el uso de técnicas mejoradas de procesamiento, de las cuales estos últimos están pobremente equipados. La principal técnica se refiere al procesamiento de más de una instrucción al tiempo en una misma unidad. El término que usa Intel para esto es EPIC ("Explicitly Parallel Instruction Computing" o Cálculo de Instrucciones Estrictamente en Paralelo). Cómo de bien funcione esta técnica hará que ésta sea independiente de la calidad de los compiladores desarrolladas para ella, así como la optimización para el procesamiento paralelo implementada en el software.

Las diferencias entre 32 y 64 bits son muy grandes. Olvidando por el momento el proceso en paralelo y otras mejoras inteligentes, propias de la arquitectura del Itanium, la ventaja más directa es la cantidad de memoria que se puede direccionar. Hace catorce años, direccionar una memoria de 4 GB con las plataformas de 32 bits era más que suficiente. Hoy en día, las grandes bases de datos sobrepasan con creces este tamaño. El tiempo empleado en cargar de nuevo los datos en la memoria virtual, junto con el acceso a los dispositivos de almacenamiento, tiene un efecto negativo en el rendimiento. Las plataformas de 64 bits son capaces de direccionar 16 terabytes de memoria (cuatro mil millones de veces más que en una plataforma de 32 bits).

Otra ventaja de la CPU de 64 bits sobre la de 32 bits es que procesa el doble de instrucciones por ciclo. Si se trabaja con registros de 16 bits en paralelo para encriptado, por ejemplo, la CPU de 64 bits procesa cuatro registros por cada ciclo de reloj, mientras que la de 16 bits lo hará de dos en dos. Una instrucción de 64 bits puede procesarse en un único ciclo de reloj, mientras que en una plataforma de 32 bits necesita dos ciclos de reloj, más uno de limpieza. Por tanto, un sistema de 64 bits puede direccionar de forma directa mucha más memoria. Cada una de las plataformas de 64 bits y las arquitecturas futuras explotan estas ventajas, además de las mejoras específicas en la arquitectura de cada marca.

El problema surge debido a que Intel se encuentra con jugadores con mayor experiencia y más sólidos en el terreno de los 64 bits, como UltraSPARC de Sun, PowerPC de IBM y Alpha de Compaq, que actualmente se encuentran en el corazón de gran cantidad de servidores y estaciones de trabajo de alto rendimiento.

Ya terminó su fase final de producción y se comercializa en pequeñas cantidades. Cuenta con Sistemas Operativos como Windows Advanced Server, Windows XP 64-bit Edition, Linux Trillian, HP-UX y otros. Pero para realizar el cambio con mayor comodidad, el Itanium ejecuta código IA-32 en hardware.

También se tiene una batalla en el software usado en las plataformas de 64 bits. Microsoft, al igual que Intel, domina en el terreno de los sobremesa, pero en el terreno corporativo tenemos algo totalmente diferente. Su versión de Windows de 64 bits tiene que luchar contra una innumerable cantidad de versiones de Unix que también se renovarán. Un factor decisivo es el momento en el que las empresas tengan que elegir cuál de las plataformas de 64 bits van a adoptar.

El resto de los competidores en el campo de los 64 bits aseguran tener una superioridad técnica frente a Itanium, además de tener una mayor estabilidad y disponibilidad de software que Windows 2000. El lanzamiento de Itanium con la experiencia de Intel en el "mercado de consumo" y la versión de 64 bits de Windows 2000 con el dominio de software del que dispone Microsoft han hecho tambalearse los cimientos del mundo de 64 bits tal como lo conocemos, y tienen que haber algunas bajas. Sólo el tiempo puede decir quiénes sobreviven y de qué forma, pero sin duda, ahora que Intel y Microsoft van a involucrarse, el mundo de los 64 bits no es el mismo.

En cuanto al mercado al que se dirige, en las tareas informáticas, un aumento en el rendimiento siempre es bienvenido, pero en un primer momento, los chips de 64 bits dominarán únicamente en los entornos de alto rendimiento. Las aplicaciones de Internet de hoy día, ven cómo millones de usuarios en todo el mundo acceden, a través de ellas, a gigabytes o incluso terabytes de información en tiempo real. Debido a esta gran capacidad manejar de datos, los sistemas de 64 bits son bienvenidos por Internet, ISPs y ASPs, además de cualquiera que tenga servidores Web multimedia y OLTP ("On-Line Transaction Proceses"). La consolidación de las bases de datos de propiedad en un único "almacén" de datos, es también una de los principales atractivos de las aplicaciones de 64 bits.

Esta tecnología proporciona un gran empuje a cualquiera que desee manejar grandes cantidades de datos, como visualizaciones científicas, simulaciones y efectos especiales de renderizado.

Si finalmente llega a los PC de sobremesa algún procesador descendiente del Itanium ya no se podrá medir el rendimiento de un chip por los Mhz ya que su velocidad dependerá más del IPC (Instrucciones Por Ciclo) que de otros aspectos.

En la Figura 25.1 puede observarse la evolución de la arquitectura de ordenadores. En ella se percibe que al final de la evolución de las arquitecturas se sitúa el Itanium, el cuál desarrolla la tecnología EPIC a 0.13 μm. Se aprecia en la gráfica que a medida que se ha aumentado el número de transistores por milímetro cuadrado se han mejorado las características arquitectónicas.

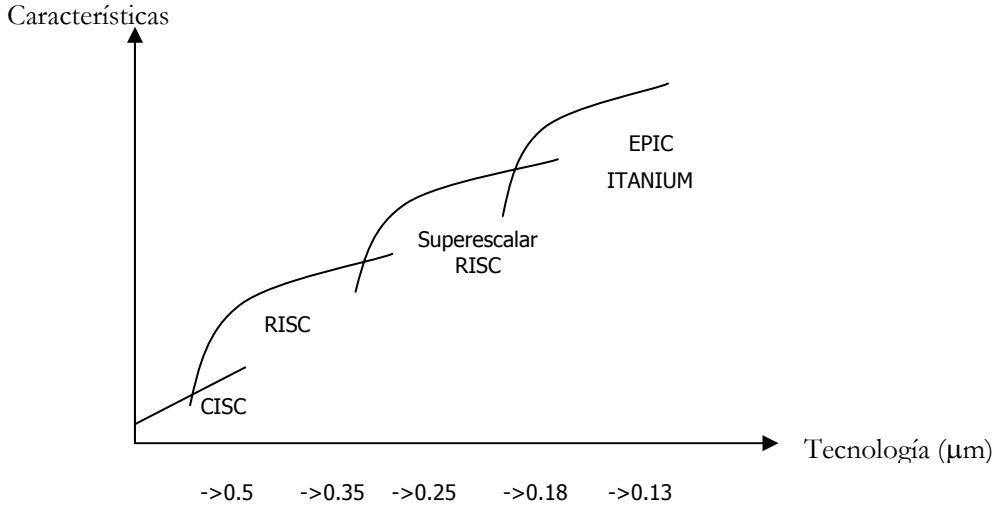


Figura 25.1. Evolución de la Arquitectura de los computadores

Tal como avanza la gráfica, Intel espera ir vendiendo más procesadores de 64 bits que con el tiempo coparán el mercado de los procesadores. Por esta razón Intel ya se ha puesto manos a la obra y ha realizado una segunda versión de este procesador, Itanium2.

25.2- CARACTERÍSTICAS

25.2.1- Características Generales

ITANIUM	CARACTERÍSTICAS GENERALES
AÑO	2001
TECNOLOGÍA	0,13 μm
MILLONES DE TRANSISTORES	25 (CPU) 300 (CACHE)
PROCESADOR	64 bits
FRECUENCIA	800 MHz
CACHE L1	32 KB
CACHE L2	96 KB
CACHE L3	2-4 MB
ALIMENTACIÓN	3.3V
RENDIMIENTO	370 SPECint_base2000
FRECUENCIA BUS SISTEMA	266MHz
ANCHO DE BANDA DE E/S	PCI-66MHz
TRANSFERENCIA	2,1GB/s

Figura 25.2 . Características generales.

25.2.2- Motivación para una arquitectura de 64-bit

La necesidad de procesadores de 64 bits es clara en aplicaciones que necesitan direccionar grandes cantidades de memoria ya sea esta virtual o física. Como ejemplos de dichas aplicaciones podemos citar:

- Bases de Datos: actualmente existen muchas bases de datos con más de 4Gb usados en registros y otra información, esta cantidad constituye el direccionamiento directo máximo posible con 32 bits. Aunque es posible y de hecho a menudo puesto en práctica, el direccionamiento de bases de datos muy grandes con procesadores de 32 bits recurriendo a distintas técnicas, es más fácil y más eficiente hacerlo si el procesador soporta operaciones y direcciones de 64 bits.
- Creación de contenido Digital: el DVD ("Digital Versatile Disk") y el HDTV ("High Definition TV") cada vez se vuelvan más comunes y el contenido de estos, por los grandes volúmenes de información que requieren, es el principal candidato para la edición en PCs avanzadas. TIVO (dispositivos semi inteligentes de grabación digital) y otros dispositivos similares podrían eventualmente volver obsoleto a las videograbadoras convencionales. En general toda la creación de contenido digital será beneficiada por el direccionamiento y procesamiento de 64 bits.
- CAD & Simulaciones: estos a menudo tratan con modelos enormes(edificios, aviones, explosiones nucleares, etc.) y como es de imaginar dichos modelos pueden fácilmente exceder los 4Gb de tamaño. Como ya se explicó antes cuando el rango direccionable natural del procesador es superado, se deben usar varios trucos para direccionar mas datos, esto implica a menudo una sobrecarga significativa al procesador.

- Criptografía: esta involucra grandes cantidades de cómputos con grandes enteros, y se beneficiaría inmensamente gracias a la disponibilidad de registros de 64 bit lo que le permitiría realizar cálculos mucho más rápidamente.
- Otros: las aplicaciones computacionalmente intensas también se verán favorecidas, por ejemplo:
    - Automatización de diseño electrónico
    - Servidores de alto rendimiento
    - Efectos gráficos como el Transform and Lighting

Todas estas aplicaciones se beneficiarán tanto de las direcciones de 64 bit como de un número mayor de registros.

25.2.3- Características de la arquitectura

- EPIC - Explicitly Parallel Instruction-Set Computing.
- Mejora el ILP ("Instruction Level Parallelism") maximizando la complementación de soluciones hardware-software. Provee mecanismos tales como "branch-hints", "cache-hints" (pistas de salto, de caché) para el compilador, para que este indique al procesador de tales eventos y permita que maneje el hardware más eficientemente. Estos mecanismos de aviso minimizan el costo computacional de los saltos y reducen fallos de consulta a la memoria caché. Además de eso, el conjunto de instrucciones en sí cuenta con "opcodes" que explícitamente permiten su ejecución en paralelo ("codeblock-hint").
- Otros conceptos que se manejan con EPIC son:
- Especulación: mejora el rendimiento permitiendo que el compilador adelante las instrucciones de carga debidas a saltos reduciendo el retardo debido a la memoria ("memory latency").
- Predicación: elimina muchos saltos y las penalizacioness debidas a las fallos en la predicción de los saltos ("branch prediction").
- Pila de registros: reduce el coste introducido por las secuencias de llamada y retorno mediante un modelo eficiente de registros enteros administrados por el RSE ("Register Stack Engine").
- Rotación de registros: renombra registros en hardware de manera automática para mejorar el rendimiento de los bucles, sin el coste debido a los métodos tradicionales ("loop unrolling").
- Instrucciones SIMD: mejoran la ejecución de aplicaciones multimedia al operar en varios datos enteros o de coma flotante en una sola instrucción.
- Cantidad masiva de registros: 128 registros de enteros, 128 registros de coma flotante, 8 registros de salto, 64 de instrucciones.
- Escalabilidad para 32 y más CPUs en paralelo.
- Uso optimizado de memoria:
    - tamaños de página de hasta 256 MB
    - tres niveles de caché
- Compatibilidad con IA-32 en hardware.

- Detección avanzada de errores (MCA – "Machine Check Architecture"), y ECC ("Error Correcting Code") en caché y bus de sistema.

Los principales objetivos que persigue Itanium son:

- Superar las limitaciones de las arquitecturas actuales.
- Aumentar la eficacia de las operaciones en coma flotante.
- Dar soporte a direccionamiento de 64 bits para memoria.
- Mantener la compatibilidad con la arquitectura IA-32.

25.3- ARQUITECTURA

Para cumplir todos estos objetivos, la arquitectura Itanium se basa en el uso de mejoras sobre el código, en tiempo de compilación, para así simplificar el proceso de ejecución de las instrucciones. Entre estas técnicas se encuentran:

- Incrementar el ILP (paralelismo a nivel de instrucciones)
- Mejorar el tratamiento de los saltos
- Evitar los tiempos de latencia en las operaciones de acceso a memoria
- Soportar la modularidad en el código

25.3.1- Paralelismo a nivel de instrucciones (ILP)

La arquitectura Itanium soporta ILP mediante:

- Soporte para que el programador especifique directamente el paralelismo .
- Una estructura denominada paquete ("bundle"), que agrupa tres instrucciones y que facilita el procesado en paralelo de las mismas .
- Un gran número de registros, para facilitar que diferentes variables hagan uso de diferentes registros.

La predicación ("predication") mejora el paralelismo.

Los saltos son un problema importante de las arquitecturas RISC que ejecutan código fuera de orden. Estas arquitecturas tradicionales usan una técnica llamada "branch prediction" (predicción de saltos) para predecir el camino correcto.

Fallos en la predicción son de ocurrencia común, alrededor del 5-10% de las veces, resultando en grandes penalidades en la ejecución, de hasta 30-40%. IA-64 usa una técnica llamada Predicación para ejecutar ambos caminos en paralelo y evitar las limitaciones de las arquitecturas tradicionales.

Se utilizan 64 registros de 1 bit para poner en práctica esta técnica, descartando aquellas ramas en la ejecución que no son parte del camino correcto. IA-64 también provee soporte eficiente para saltos múltiples y comparaciones paralelas, ambas características permiten realizar el salto múltiple en 1 ciclo de máquina, acelerando la ejecución significativamente.

En todos los casos, el registro de predicación indica si una rama en particular y sus datos asociados están activos o no. Las ramas activas continúan su ejecución y si se desactivan su ejecución cesa.

Los predicados, que son parte de toda condicional de la IA-64, proveen un mecanismo muy eficiente para terminar la ejecución de ramas que no deben seguirse, para establecer bucles y terminarlos, para administrar la predicción dinámica y realizar comparaciones de n caminos.

Los predicados ayudan a manejar la compleja tarea del flujo de control, compleja debido a un agresivo paralelismo a nivel de instrucciones (ILP – "Instruction Level Paralelism") que se realiza en tiempo de compilación.

De esta manera, la predicación reduce los fallos de predicción en los saltos, incrementando el rendimiento. Es particularmente útil en aplicaciones donde es difícil predecir los saltos como en grandes bases de datos, "data mining", "warehousing", etc.

Como no cuentan con predicación, las arquitecturas RISC tradicionales han tenido poco éxito con ILP.

25.3.2- Evitar tiempos de latencia en accesos a memoria

La especulación ("speculation") minimiza el efecto del retardo de la memoria.

En el contexto actual, las CPUs van incrementando su velocidad a una tasa mucho mayor que la memoria. Se espera que esta tendencia continúe en el futuro, y posiblemente se acentúe. Entonces estamos ante un gran cuello de botella que aqueja a las arquitecturas de hoy.

Para reducir el impacto de esta limitación, las arquitecturas tradicionales permiten que el compilador y el procesador realicen la carga de datos tiempo antes de que estos sean necesarios, pero los saltos hacen el papel de barreras, de límites para el entorno en el que se aplica esta técnica.

IA-64 emplea una técnica llamada especulación para iniciar la carga desde memoria de forma anticipada, inclusive anticipando al uso en porciones de código más allá de un salto.

La especulación es una característica muy utilizada en IA-64. Las arquitecturas RISC tradicionales pueden utilizar una técnica llamada 'carga a prueba de fallos' ("non-faulting load") para evitar el costoso manejo de errores cuando la carga anticipada podría no ser válida.

IA-64 evita este problema ofreciendo una solución en la arquitectura para manejar la carga anticipada. La idea consiste en que cada dato va asociado a un bit que dice si un error se ha producido o no cuando se cargó dicho dato. Este bit de error (llamado bit "Nat") acompaña al dato a través de todas las operaciones aritméticas, moves y condicionales hasta que se realiza sobre el dato una instrucción de check, que detecta si hubo un error, y dispara la rutina manejadora del error.

Sin embargo, un 99% de las veces una carga anticipada que produce un error resulta pertenecer a una rama que nunca se debió ejecutar, por lo que simplemente se abandona su ejecución, sin saltar a la rutina manejadora de errores.

Esto significa que con IA-64, el compilador puede realizar carga anticipada en forma agresiva ("speculation") sin prestar atención a penalizaciones en caso de errores.

La especulación de datos permite al compilador realizar la carga aún antes de un store que podría escribir sobre el mismo registro para el cual se realiza la carga anticipada.

IA-64 mantiene un registro de todas las cargas anticipadas en una tabla llamada ALAT ("Advanced Load Address Table"). Cuando se encuentra una instrucción store que causa conflictos con una carga anticipada contenida en la tabla, la entrada correspondiente se elimina, y cuando la carga se chequea en el ALAT, la ausencia de su entrada indica que el valor debe ser recuperado nuevamente y se obtiene el resultado correcto.

- *Especulación de datos*: resuelve el problema de que el contenido de la posición de memoria referenciada por la instrucción de carga sea modificado entre el momento de lanzar la instrucción de carga adelantada y el momento de la verificación. En el momento de la verificación se realiza una comprobación sobre el contenido de la dirección de memoria referenciada por la instrucción de carga. Si el contenido ha cambiado, se realiza de nuevo la consulta a memoria, si no, se devuelve el resultado. Este tipo de load adelantado no retrasa el lanzamiento de las excepciones, sino que se atienden en el momento de producirse, por ejemplo, por fallo de página.
- *Especulación de control*: esta técnica intenta resolver el problema del orden correcto de ejecución de las instrucciones, en concreto, el problema del lanzamiento de excepciones y el problema de los saltos. Ambos problemas se resuelven del mismo modo. Mediante el mecanismo de load especulativo. Cuando se lanza un load especulativo las excepciones que se puedan producir de su ejecución quedan pospuestas hasta el momento de su verificación.
- *Especulación Combinada* : en algunos casos es necesario combinar las dos técnicas, por ejemplo cuando se quiere adelantar la ejecución de una instrucción de carga que está dentro de una subrutina fuera de ésta. En estos casos, se debe comprobar que los datos no varían, pero posponiendo el tratamiento de las excepciones hasta el punto donde se realiza la verificación. En caso de que los datos hayan variado, o se haya producido una excepción, se lanzará una excepción en el momento de la verificación.

Una barrera de código es una instrucción por encima de la cual no se debería adelantar la ejecución de una instrucción de carga. Hay varias instrucciones que actúan de barrera de código:

- Instrucciones de almacenamiento (store): puede hacer referencia a la misma dirección de memoria que la instrucción de carga, que debería ejecutar después de dicha instrucción de almacenamiento.
- Instrucciones de salto (br): no se sabe si realmente la instrucción de carga se debe ejecutar .

La ejecución especulativa permite, evitar las barreras de código.

25.3.3- Desdoblamiento y rotación de bucles (“Loop unrolling and rotation”)

El manejo eficiente de bucles es uno de los factores clave para determinar el rendimiento de todas las arquitecturas, y IA-64 provee características especiales para ello.

Una técnica común utilizada es el desdoblamiento. Consiste en evitar los saltos al principio del bucle duplicando el código del bucle dos, tres o más veces para reducir el número de saltos. Esta técnica está disponible en IA-64 y en las arquitecturas RISC.

El problema de esta técnica es que conlleva un aumento del código. IA-64 sin embargo provee características especiales para explotar esta técnica. Cuenta con la rotación de registros que permite la ejecución de bucles sin el aumento de código.

Optimización en la Ejecución de Bucles

Para ello hace uso de:

- Instrucciones especiales de salto.
- Dos registros de propósito específico denominados LC ("Loop Count") y EC ("Epilogue Count").
- Mecanismo de rotación de registros: consiste en que el registro lógico X se convierte en la siguiente iteración en el X+1. Esto se puede aplicar a los registros de predicado, a los de enteros y a los de coma flotante.

En IA-64, los registros y los predicados pueden rotar bajo el control del compilador. Esto significa que con cada paso en la rotación, los valores que estaban en, por ejemplo, el registro 40, avanzarían al registro 41. El valor en el último registro se mueve al primer registro, o sea, los valores de los registros rotan. Lo mismo sucede con los registros que contienen predicados.

Con esta rotación, el compilador puede correr una copia del código dentro del bucle y los valores se asignan a distintos registros automáticamente a medida que el código itera. Cuando el bucle finaliza, con la ayuda de los registros "Loop count" y "Epilog count" se vuelve al estado anterior a la rotación y el bucle termina.

De esta manera se administran los bucles sin la necesidad de prólogos o epílogos al principio y al final de cada bucle para manejar la inicialización y la restauración, y sin usar predicción de saltos que podría llevar a penalizaciones en el rendimiento en caso de fallar.

Para realizar estos saltos, se utiliza la instrucción de salto(br) con el complemento cloop. Los "counted loops" hacen uso del registro de propósito específico LC ("Loop Count"). Este registro se chequea en cada salto, si es distinto de cero, se decrementa su contenido y se efectúa el salto. Cuando es igual a cero no se realiza el salto. La ventaja de esta técnica es que, mirando el contenido del registro LC, se puede saber de antemano si el salto se va a tomar o no.

25.3.4- Unidades funcionales

Las instrucciones de IA-64 están divididas en cuatro categorías:

- I: Operaciones con enteros.
- F: Operaciones con números en punto flotante.
- M: Operaciones de acceso a memoria.
- B: Saltos.

El Itanium maneja grupos de tres instrucciones, lo que Intel llama "bundle" (ternas de operaciones), que pueden ser despachadas al mismo tiempo por diferentes unidades funcionales.. Cada palabra de instrucción o "bundle" consiste de tres operaciones elementales de algunas de las categorías mencionadas.

Ejemplo de bundles:

- Memoria-entero-entero, memoria-salto-salto (MII/MBB)
- Memoria-punto flotante-entero, memoria-punto flotante-entero (MFI/MFI)

Como consecuencia del manejo de bundles a través de la ventana de ejecución el Itanium tiene un throughput máximo de 6 instrucciones por ciclo de reloj.

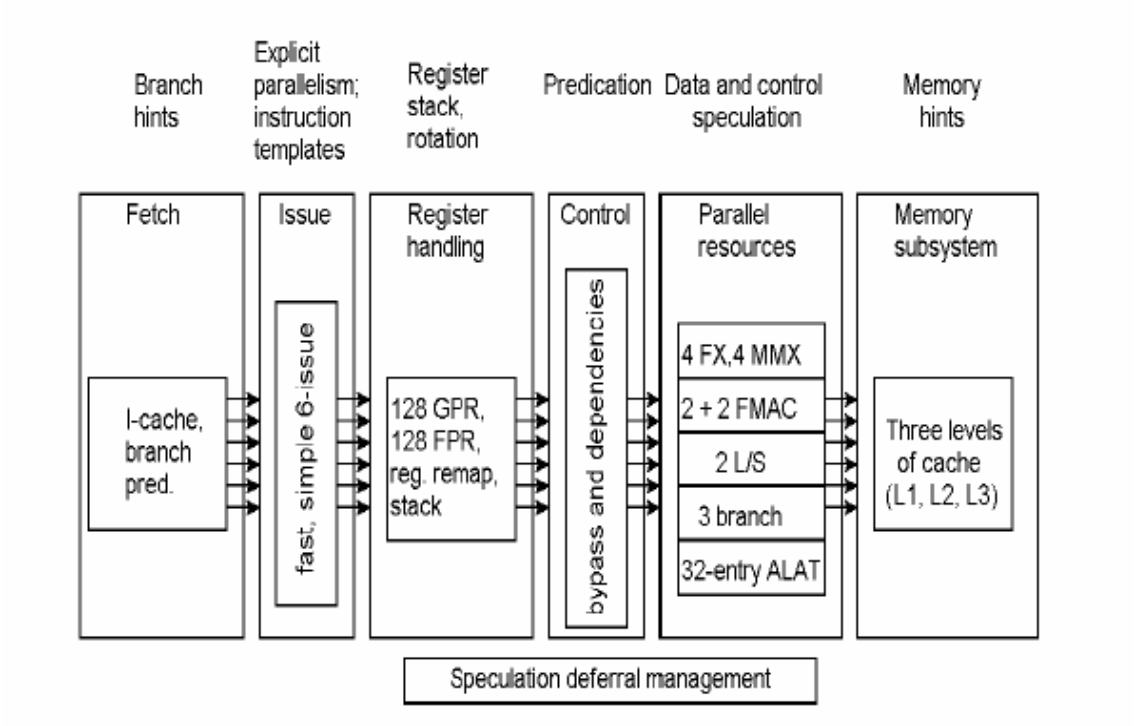


Figura 25.3 Flujo de instrucciones (Dispersión de bundles)

La principal característica que ofrece el Itanium es la de realizar hasta seis instrucciones a la vez, está particularidad esta representada a lo largo de toda la figura de su arquitectura (dispersión de bundles).

En el primer paso se realiza la búsqueda de instrucciones en la caché L1 de instrucciones y se aplican las técnicas de predicción de saltos, para ello se utilizan las tablas:

- BPT ("Branch Prediction Table") y MBPT ("Multiway Branch Prediction Table").

  Estas se usan para predecir dinámicamente si serán o no tomados los saltos, cada tabla se encarga de diferentes tipos de "bundles". Por ejemplo la BPT se ocupa de los saltos contenidos en ternas del tipo MMB mientras que la MBPT se encarga de los MBB o BBB (ternas con más de un salto en general).

- TAR ("Target Address Register"): tabla de cuatro entradas manipulada directamente por el compilador para especificar la dirección de saltos predichos estáticamente.

- TAC ("Target Address Cache") tabla de 64 entradas responsable de proporcionar la dirección de salto para los predichos dinámicamente.

- Una pila de direcciones de retorno.

La zona registros consta de 128 registros para números enteros (de 64 bits cada uno), 128 registros de números en coma flotante (de 82 bits cada uno), 8 registros para saltos, y varios registros más con diversas funciones, además hay otra serie de registros para la compatibilidad x86.

Así mismo en esta zona se encuentra la "Stack Engine", sistema de pila mediante registros que permite reducir el número de accesos a memoria en las llamadas y retornos de subrutinas (optimización en las llamadas a procedimientos). Desde el punto de vista de la pila el banco de registros de propósito general queda dividido de la siguiente manera:

- Registros r0 - r31: registros de propósito general.
- Registros r32 - r127: registros de pila.

Después las instrucciones se dirigen a una zona donde hay una serie de recursos cuya principal misión es ejecutar las instrucciones.

El Itanium tiene 17 unidades de ejecución paralelas (superescalar), todas ellas segmentadas ("fully pipelinet") con 10 etapas. Por tanto, cada unidad funcional puede aceptar una nueva instrucción por ciclo de reloj o, en su caso, un parón. Estas unidades son:

- 4 unidades para ejecutar operaciones con números enteros y accesos a memoria .
- 1 unidad de punto flotante que contiene 2 unidades FMAC ("Floating-point Multiply Acumulate" u operaciones acumuladas en punto flotante) que operan con operandos de 82 bits.
- 4 unidades para las instrucciones multimedia (MMX).
- 2 unidades punto flotante de precisión simple
- 2 unidades de carga /almacenamiento
- La Tabla de carga de direcciones adelantadas ALAT("Advanced Load Address Table"), que proporciona apoyo para especulación y minimización de latencia (burbujas de espera).
- 3 unidades de saltos

25.3.5- Microarquitectura

- Búsqueda /Emisión y ejecución desacopladas
- Prebúsqueda
- L1 Instrucciones, 16 KB
- Jerarquía de predictores de saltos
- Estática
- Dinámica en dos niveles (2,2)
- Multivía
- Corrección en la predicción asociada lazos
- Número de iteraciones conocido en tiempo de compilación
- Renombre de registros
- 9 puertas de emisión a unidades de ejecución
- 2 Memoria
- 2 enteros
- 2 punto flotante
- 3 saltos
- 6 instrucciones por ciclo
- Instrucciones independientes (template)
- Disponibilidad de recursos (template)
- 17 unidades de ejecución

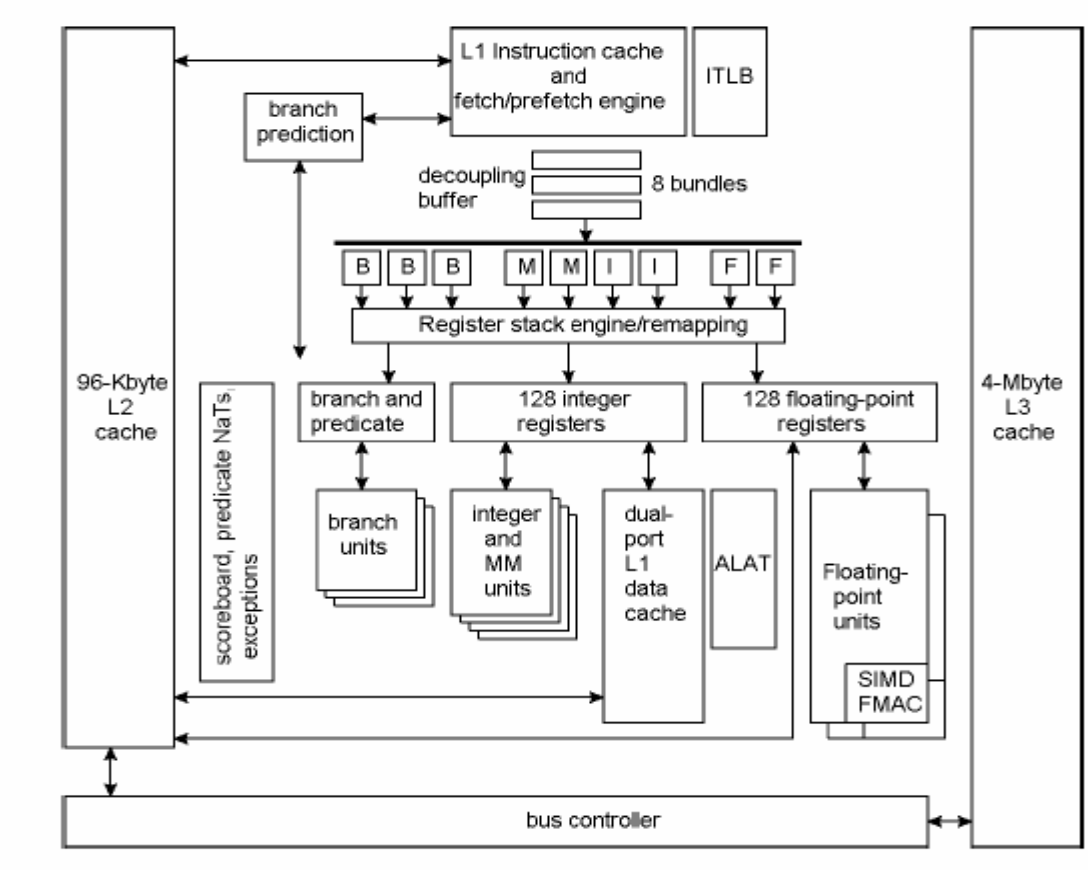


Figura 25.4 Microarquitectura del Itanium.

25.3.6- Segmentación

Las 10 etapas de las que dispone Itanium están dispuestas según la figura 25.3, éstas se han reducido de las 20 que contiene el Pentium 4 para evitar conflictos por dependencia de datos.

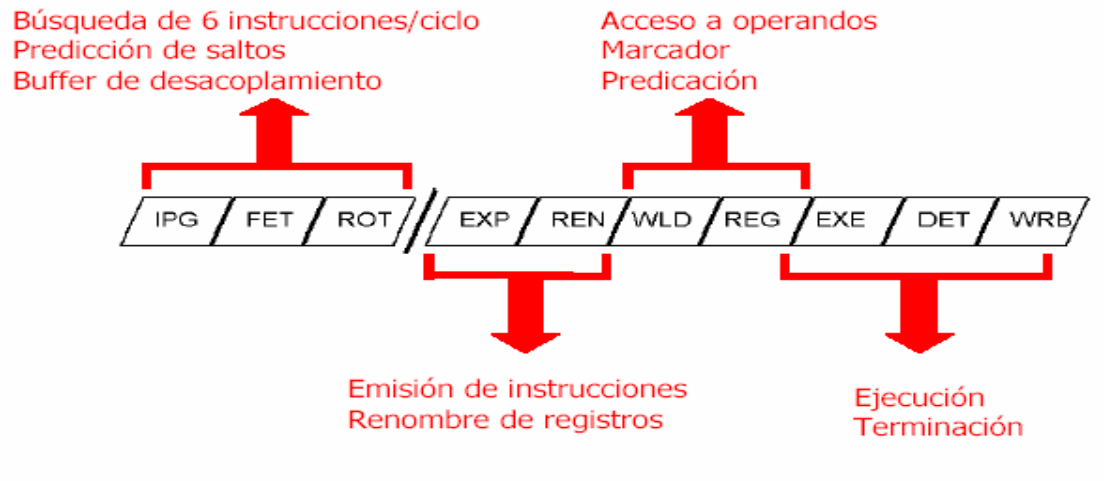


Figura 25.5. Segmentación de la unidad de enteros.

- IPG: generación de instrucciones.
- FET: búsqueda de instrucciones.
- ROT (rotación): rotación de registros para generar la burbujas necesarias para evitar las dependencias de datos .

- EXP (expansión): tiene como misión repartir las seis instrucciones a los diferentes puertos, en esta etapa además existe la posibilidad de parar la instrucción a través de los bits de parada.
- REN (renombramiento) :acomoda los registros en los lugares adecuados de la pila, para optimizar su tamaño.
- WLD: decodificación.
- REG: lectura de operandos.
- EXE: ejecución.
- DET: detección de excepciones.
- WRB: post-escritura.

25.4- APORTACIONES Y NUEVOS RECURSOS ARQUITECTONICOS

Las arquitecturas tradicionales, aparte de tener muchas restricciones para llevar a cabo ILP, también cuentan con pocos recursos en hardware para permitir la ejecución en paralelo. En contraste, los procesadores basados en IA-64 cuentan con 128 registros enteros de propósito general, 128 registros de punto flotante, 64 registros de predicación y muchas unidades de ejecución para asegurar el buen funcionamiento con altas cargas de trabajo. La arquitectura se diseñó para ser escalable, permitiendo la fácil adición de unidades de ejecución.

25.4.1- Procesador de 64 bits

Hay dos implicaciones en el término de procesador de 64 bits. Uno es la habilidad de acceder a 64 bits (8 bytes) de información a la vez, esta es una función de la estructura del bus. La otra es la habilidad de usar 64 bits para definir una dirección de memoria. Un procesador de 32 bits puede acceder a 2^{32} bits de datos, aproximadamente 4,3 billones de bits. Comparando esto con 2^{64} bits, aproximadamente 18.4 quintillones de bits (alrededor de 2,1 billones de gigabytes), a los que puede acceder un procesador de 64 bits

25.4.2- EPIC (Explicitly Parallel Instruction Computing)

EPIC como una nueva forma de concebir el código. La implementación de la filosofía de diseño EPIC permite alcanzar nuevos niveles de paralelismo y rompe el paradigma de ejecución secuencial que existe en las arquitecturas tradicionales. El uso de mecanismos como la predicación y la especulación combinadas con el paralelismo explícito permiten que IA-64 logre ganancias de rendimiento donde otras arquitecturas no han podido hacerlo (retardo de memoria, fallos en la predicción de ramas).

Es una nueva arquitectura que dota al Itanium de 17 unidades de ejecución. Utilizando este paralelismo de forma eficaz, estos microprocesadores son capaces de ejecutar hasta 20 instrucciones por ciclo de reloj.

EPIC tiene la singularidad de poder indicar al microprocesador, qué partes integrantes del programa pueden ser ejecutas con total paralelismo, algo que las herramientas de generaciones anteriores no podían.

Intel y HP acentúan las bondades de la predicación y la especulación más allá de simples características que mejoran la ejecución, y las sitúan al nivel de adelantos revolucionarios que permiten diseñar todo un nuevo paradigma para lograr verdaderamente la ejecución paralela más allá de su simple aplicación a casos especiales.

25.4.3- Un gran número de registros

El Itanium ofrecen un enorme número de estos registros. Para un buen uso de ellos, es necesario que el compilador aproveche las nuevas características de procesado en paralelo.

La arquitectura Itanium pone a disposición del programador gran cantidad de registros:

- 128 registros de enteros (64 bits)
- 128 registro de coma flotante (82 bits)
- 64 registros de predicado (1 bit), para guardar predicciones (formalmente hablando las evaluaciones de predicados) .
- 8 registros para direccionamiento en saltos (64 bits)
- 128 registros de propósito específico para aplicaciones (64 bits)
- 1 registro de puntero a la instrucción en curso (IP): no puede ser accedido ni modificado directamente, apunta a la dirección del paquete de la instrucción en curso y cambia cuando entra a ejecutar la siguiente instrucción.

Adicionalmente, el Itanium contiene varios registros dedicados a la monitorización de las prestaciones de la CPU. Información tal como el número de instrucciones ejecutadas, que pueden ser seguidas por software, haciendo posible al software de administración de sistemas la escritura en tiempo real, teniendo un impacto mínimo en las prestaciones del sistema y registros de identificación de la versión de IA-64 implementada que identifican al procesador (CPUID).

Validación de registros

La ejecución especulativa, para el reordenamiento de instrucciones, obliga a comprobar que el contenido de un registro es un dato válido en el momento de la consulta. Para ello, todos los registros de enteros y de coma flotante llevan asociado un bit NaT ("Not a Thing"), para indicar si el contenido del registro es válido en ese momento o no. En el caso de los registro de coma flotante, se llama NaTVal.

25.4.4- Organización de Memoria

Se define un solo espacio de direcciones lineal y uniforme, con un tamaño de 264 bytes. Esto quiere decir que, tanto datos como código, comparten el mismo espacio de memoria. Este espacio de memoria está sin segmentar, y no hay definidas zonas de uso específico.

El Itanium tiene 3 niveles de caché, L1 y L2 van integradas, es decir, residen en la cpu y trabajan a la misma velocidad que el núcleo. La memoria caché L3 a pesar de estar fuera del encapsulado del procesador también trabaja a su misma velocidad.

Adicionalmente el Itanium posee buffers de traducción de direcciones lineales a direcciones reales TLB's, basándose en el principio de localidad. El bus del sistema presenta un ancho de banda de 2.1 GB/s. La jerarquía de memoria está organizada en los siguientes niveles:

- Cache de datos de nivel 1 (L1D), de 16 Kbytes.
- Cache de instrucciones de nivel 1 (L1I), de 16 Kbytes.
- Cache de nivel 2 (L2), de 96 Kbytes.
- Cache de nivel 3 (L3), el tamaño puede variar de 2 a 4 Mbytes.
- TLB de datos de nivel 1 (L1-DTLB), con 32 entradas.
- TLB de datos de nivel 2 (L2-DTLB), con 96 entradas.
- TLB de instrucciones (ITLB), con 64 entradas.

25.4.5- Compatibilidad con las instrucciones de 32 bits

Uno de los requisitos de Itanium es conservar la compatibilidad con el juego de instrucciones de la arquitectura IA-32 (procesadores actuales Pentium). Itanium puede ejecutar aplicaciones de la arquitectura IA-32, así como aplicaciones que tengan mezcladas instrucciones de la arquitectura IA-32 e instrucciones de la arquitectura Itanium. Esto significa que los programas escritos para las máquinas actuales deberían funcionar sin modificaciones. Objetivamente parece que dichas aplicaciones funcionarán a una velocidad más lenta de lo normal debido a la compatibilidad.

25.4.6- Optimización operaciones de coma flotante y multimedia

Un gran porcentaje de la cpu del Itanium, aproximadamente el 10%, está dedicado a la unidad de coma flotante (FPU). La arquitectura implementa las siguientes optimizaciones para las operaciones en coma flotante:

- Banco de registros de coma flotante de 128 registros de 82 bits.
- Instrucciones especiales como multiplica y acumula (fma).
- Instrucciones especiales de load y store para números en coma flotante.
- Se permite la transferencia de datos entre el banco de registros de enteros y el de coma flotante.
- Fácil conversión entre enteros y números en coma flotante.
- Soporte para técnicas como la especulación o la rotación de registros.

25.4.7- Optimización en la Ejecución de Saltos

Hoy en día, la optimización de los saltos es uno de los grandes problemas con los que se enfrentan los diseñadores de hardware. La técnica más usada actualmente es la predicción de salto. Si la predicción es correcta, no se producen parones en el pipeline. Sin embargo, si es errónea, el coste es muy alto.

La arquitectura define dos tipos de saltos:

- Saltos relativos al registro IP
- Saltos indirectos, que hacen uso de los registros de salto.

Se permite la ejecución en paralelo de varios saltos a la vez. La forma de decidir qué salto se toma primero es mediante los registros de predicado. El salto cuyo registro de predicado se haga cierto antes, es el que primero salta.

Itanium optimiza el manejo de los saltos mediante dos Técnicas:

- Instrucciones con predicados

Se puede condicionar la ejecución de operaciones al contenido de los registros de predicado. Esta técnica nos permite convertir muchas de las dependencias de control (saltos) del código en dependencias de datos. Para ello, se lanzan a la vez las dos ramas del salto, condicionando la ejecución de las instrucciones de cada rama al contenido de un registro de predicado que se ha evaluado previamente. Así, la rama cuyo registro de predicado esté a uno, completará su ejecución, mientras que las instrucciones de la otra rama serán tratadas como instrucciones NOP.

- Instrucciones de salto especiales

Un ejemplo son los "counted loops", utilizados para realizar bucles de tipo for (cuya condición de parada es que un contador alcance un determinado valor). Este tipo de saltos no se puede eliminar con la ejecución con predicados.

25.5.- MODELO DE PROGRAMACIÓN

Como ya se dijo antes, las prestaciones del Itanium dependerán de la correcta relación entre el software y el hardware. Los compiladores, a través de instrucciones específicas para ello, tendrán control total, si así lo desean, sobre la predicción de saltos, el orden de las operaciones, la predicación y la especulación. En particular el compilador puede decidir, si un salto debe ser predicho estáticamente o dinámicamente con base en su historial. También puede decidir si un salto debe ser predicho tomado o no tomado, en caso de ser estática la predicción. El Itanium posee el hardware necesario para poder ofrecer estas facilidades al compilador.

El Itanium, tiene un nuevo conjunto de instrucciones, no es ya una máquina CISC, para ejecutar el código que podrían ejecutar sus ancestros tendrá que traducir las instrucciones mediante hardware específico para tal propósito. Esto hará, en principio, la ejecución de programas viejos más lenta de lo que será su ejecución en computadores anteriores de la arquitectura de Intel. Tampoco es RISC, más bien es del tipo VLIW ("Very Large Instruction Word") o EPIC ("Explicitly Parallel Instruction Computing").

VLIW o EPIC, es el paradigma idóneo para incrementar el paralelismo a nivel de instrucciones (ILP). Hacer muchas instrucciones o trozos de ellas al mismo tiempo.

La IA-64 representa el nuevo modelo ISA ("Instruction Set Architecture" o Conjunto de Instrucciones de Arquitectura) basado en la tecnología EPIC, siendo totalmente compatible por hardware con las instrucciones de su predecesora, la IA-32.

Las instrucciones se organizan en paquetes o grupos de 3 instrucciones máximo("bundle") que no poseen dependencias entre ellas y que, por lo tanto, se pueden ejecutar en paralelo. En un momento determinado, el procesador intentará procesar en paralelo tantas instrucciones de un grupo de instrucciones como le sea posible, dependiendo de los recursos disponibles.

Los paquetes de instrucciones definidos para este concepto están formados por 128 bits, a diferencia de lo que ocurría en la arquitectura IA-32, cuya longitud de instrucción no era fija en un principio.

Las tres instrucciones EPIC del paquete consumen 123 bits (41 para cada una), dejando los cinco restantes (bits de template) para ser utilizados como almacén extra de información, de cara a asistir al procesador en la utilización más eficiente de recursos.

El mecanismo template, deja al compilador la decisión de qué unidad funcional será la que ejecute la instrucción cada instrucción del paquete. En los templates se puede indicar qué paquete termina el grupo. Un paquete puede indicar que un grupo se acaba en la primera, segunda o tercera instrucción de dicho paquete. Para terminar un grupo, se indica añadiendo un stop. Además, también permite especificar si todas las instrucciones pertenecen al mismo grupo o no.

Los distintos tipos de templates disponibles son los siguientes:

MII, MIIs, MIsI, MIsIs, MLX*, MLXs*, MMI, MMIs,

MsMI, MsMIs, MFI, MFIs, MMF, MMFs, MIB, MIBs,

MBB, MBBs, BBB, BBBs, MMB, MMBs, MFB, MFBs

Donde:

M : Instrucción de memoria (ld o st)
I : Instrucción de enteros
F : Instrucción de coma flotante
B : Instrucción de salto
L : Instrucción que trabaja con datos inmediatos largos
s : Indica que se realice un stop
(*) LX : Tipo especial, que se procesa en la unidad funcional de enteros

25.5.1.- Tipos de datos

Los tipos de datos que soporta Itanium son los siguientes:

- Enteros : de 1, 2, 4 y 8 bytes.
- Coma flotante : formatos simple, doble y doble-extendido.
- Punteros a memoria: 8 bytes .

El formato estándar para los enteros es 8 bytes. Los registros también poseen 8 bytes (64 bits) de longitud. Cuando se trabaja con operandos enteros de 1, 2 ó 4 bytes, se rellenan con ceros hasta alcanzar la longitud de 8 bytes.

25.5.2.- Formato de instrucciones

El formato de instrucción es el siguiente:

[(preg)]	cop	[.comp1]	[.comp2]	destino =	fuente	[,fuente]
6 bits	14 bits			7 bits	7bits	7bits

Donde:

- (preg) : registro de predicado.
- cop (codigo de operación) : identifica a la instrucción.
- .comp1, .comp2: algunas instrucciones pueden llevar complementos, que indican una variación sobre la instrucción de base.
- destino, fuente : casi todas las operaciones tienen al menos dos operandos fuente y un destino.

El modelo de ejecución es registro-registro.

Las únicas instrucciones de acceso a memoria son ld (carga) y st (almacenamiento). Ejemplos de formato de instrucción son:

Instrucción simple: add r1 = r2, r3

Instrucción con predicado: (p4)add r1 = r2, r3

Instrucción con dato inmediato: add r1 = r2, r3, 1

Instrucción con complemento: cmp.eq p3 = r2, r4

25.6- ANÁLISIS DEL RENDIMIENTO

El Itanium representa el principal proyecto de desarrollo de Intel. Este procesador representa un cambio significativo en un mercado basado en instrucciones RISC. Intel se arriesga con su nueva arquitectura EPIC, la cual no puede ser efectiva fácilmente dentro de la arquitectura RISC que predomina en estos momentos.

Las implicaciones del Itanium en el mundo del PC son muchas:

- La velocidad de un ordenador, actualmente medida en megahercios en el mundo del PC, puede tener menos importancia con la llegada del Itanium.

- Para el desarrollo efectivo del Itanium se necesita una cooperación sin precedentes con las empresas de compiladores para que desarrollen software específico de 64 bits.

El Itanium es capaz ejecutar billones de operaciones de coma flotante por segundo, esto es debido a su habilidad para realizar cálculos de 6 instrucciones por cada ciclo de reloj y eso le da al Itanium un potencial impresionante. Considerando que el software aprovecha éstas cualidades vamos a analizar su rendimiento.

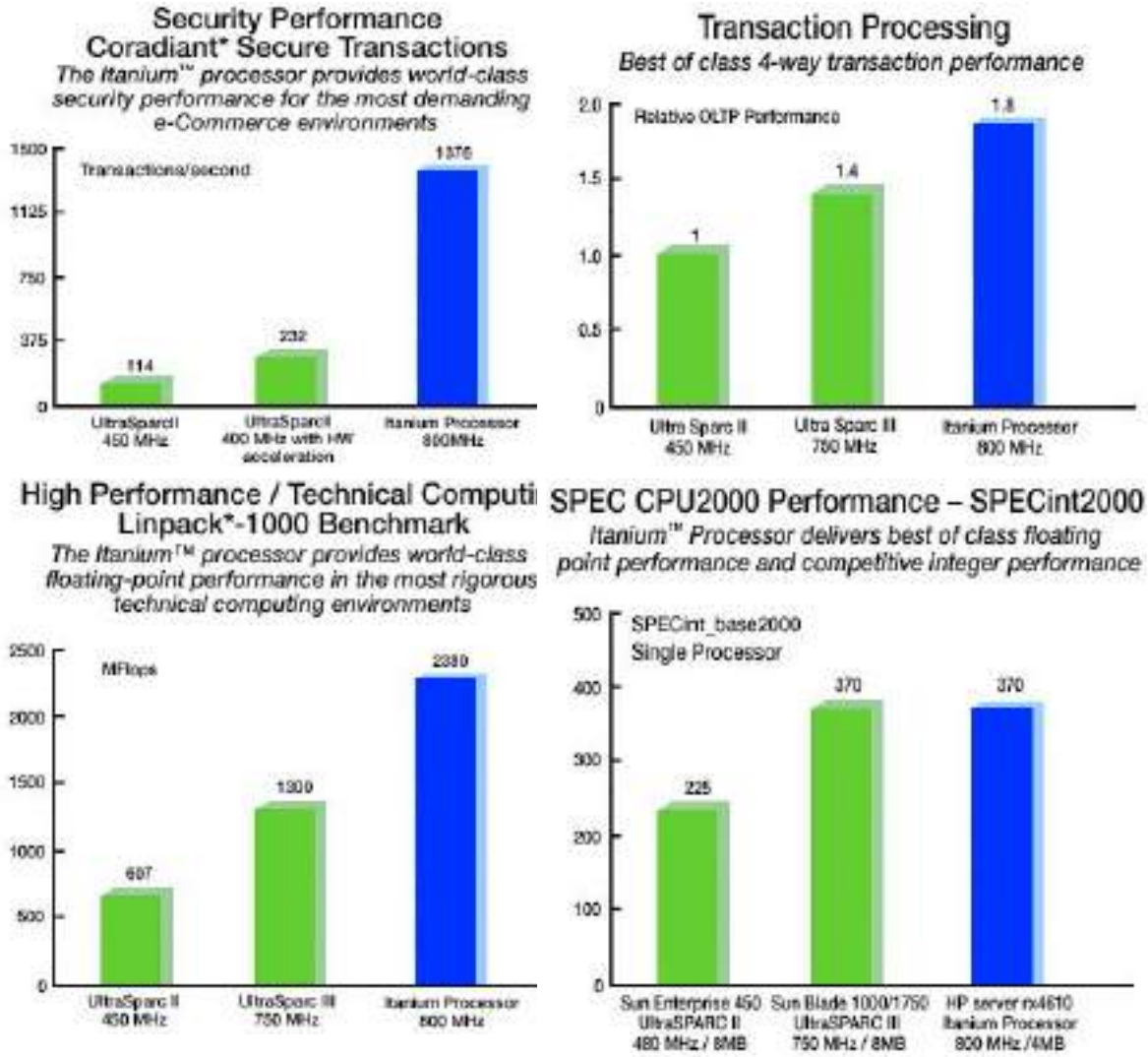


Figura 25.6. Benchmarks del Itanium

ANEXO A

DISEÑO DE PROCESADORES RISC

1. LA ARQUITECTURA "RISC"

1.1. Procesadores RISC y CISC

Hasta finales de la década de los 70 se pretendía reducir el coste del "hardware", mediante el uso de complejos juegos de instrucciones basados en la microprogramación. En esta época el tiempo de acceso a Memoria Principal era muy superior al de decodificación y procesamiento de las instrucciones, debido al uso de los circuitos integrados. Para paliar ese desequilibrio se disminuyó el empleo de la Memoria Principal y se aumentó el del procesador, creando así los juegos de instrucciones complejos o **CISC**.

Otros aspectos ventajosos son: el abaratamiento del "hardware", la facilitación del diseño de los compiladores, y la disminución del tamaño de los programas.

Los avances tecnológicos han permitido disminuir el desequilibrio antes mencionado con nuevas memorias más rápidas y el uso de las memorias caché. Esto ha supuesto la pérdida de interés por la microprogramación y la aparición, por su parte, de las tecnologías **RISC** en la década de los 80. El principal objetivo de esta tecnología es disponer de instrucciones muy simples, con el mínimo número de microinstrucciones.

En la actualidad se admite la superioridad de la arquitectura **RISC**, respecto a la arquitectura **CISC**, si bien **CISC** todavía persiste.

1.2. Generaciones de los procesadores RISC

1.2.1. Primera generación

A ella pertenecen los tres primeros modelos de la primera mitad de la década de los 80:

- IBM 801.
- RISC (diseñado por Patterson).
- MIPS (diseñado por Hennessy).

1.2.2. Segunda generación

Nace en la segunda mitad de la década de los años 80, con los siguientes modelos representativos:

- SPARC.
- CLIPPER C100.
- HPPA.
- M88100.
- MIPS R2000.
- AMD 29000.

1.2.3. Tercera generación

Integrado por los siguientes modelos de la primera mitad de la década de los 90:

- i860.
- M88110.
- IBM RS6000.
- PA7100.
- C400.
- Super SPARC.
- ALPHA.

1.3. Metodología de diseño

El profesor Hennessy inició en 1981 el proyecto **MIPS**, que culminó en 1984 con el desarrollo de la primera arquitectura de la familia de procesadores **RISC**.

Basándonos en este modelo resumiremos los pasos a desarrollar como método de diseño de procesadores:

- Definición del Repertorio de **Instrucciones Máquina**. Modos de direccionamiento y formato.
- Especificaciones que debe cumplir la **ALU** y su diseño correspondiente.
- Diseño del **Camino de Datos** "**monociclo**". ALU, Banco de Registros, Memoria y componentes auxiliares.
- Diseño del **Camino de Datos** "**multiciclo**".
- Diseño de la **Unidad de Control**. Técnicas cableada y microprogramada.
- Diseño de un procesador "**segmentado**". La técnica de la segmentación.

1.4. Fórmulas para la evaluación del rendimiento de los procesadores

Las fórmulas y magnitudes más importantes que definen el rendimiento de un procesador se describen aquí:

- El **CPI** especifica los ciclos que tarda de promedio en ejecutarse cada instrucción del programa de prueba al que se hace referencia. Para calcular el **CPI** hay que tener en cuenta los ciclos de cada instrucción, así como el porcentaje de dicha instrucción que se ejecuta en el programa en cuestión.
- El **rendimiento** del procesador se evalúa como el inverso del tiempo que necesita para ejecutar el programa de referencia.
- Para calcular los **MIPS** nativos se hallan los millones de instrucciones que se pueden realizar en un segundo
- Los **MIPSVAX** se calculan teniendo en cuenta el tiempo que tarda en ejecutarse un programa en el **VAX 11/780** y en el procesador a evaluar.

2. REPERTORIO DE INSTRUCCIONES MÁQUINA

Nos referiremos a una versión reducida de los procesadores **MIPS** para el estudio del repertorio de instrucciones. La palabra de estado de este procesador es de 32 bits, como el código máquina de cada instrucción.

Frente a la arquitectura memoria-memoria, que necesita utilizar demasiado espacio de memoria, el **MIPS** responde a la arquitectura carga-almacenamiento, en los que únicamente los registros internos soportan los operandos de la instrucción y sus contenidos se cargan en memoria ("load") y se almacenan ("store").

A la hora de desarrollar las instrucciones únicamente hacemos referencia a los componentes destinados a números enteros.

El **MIPS** dispone de un Banco de 32 registros, cada uno de 32 bits de tamaño. La designación de los registros para números enteros es \$0, \$1,.....,\$31, tal como se representa.

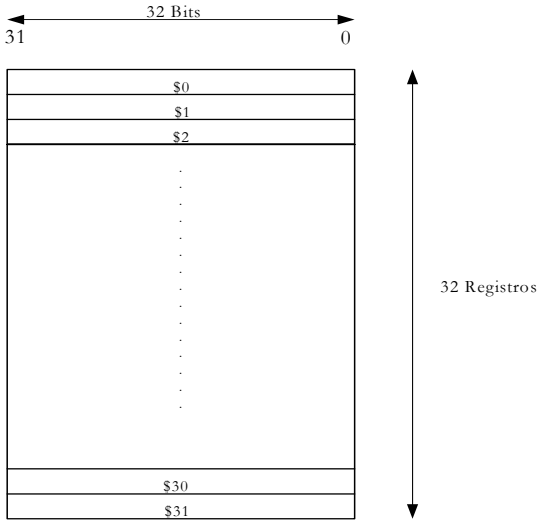


Figura 1—Representación del Banco de Registros para números enteros del MIPS.

2.1. Repertorio de instrucciones

Tenemos nueve instrucciones distribuidas de la siguiente forma:

2.1.1 Instrucciones aritméticas y lógicas

Tanto los dos operandos fuente, como destino de este grupo son registros. Explicamos la nomenclatura usada en Ensamblador y su funcionalidad.

- **add**

Realiza la suma binaria de dos registros dejando el resultado en el registro destino. En la nomenclatura se comienza designando el registro destino, seguido de los dos fuentes.

add $1, $2, $3 #$1 = $2 **+** $3

Una variante es el **addi**, que admite que uno de los opèrandos sea un valor inmediato.

addi $3, $5, 9 #$3 = $5 **+** 9

Las restantes instrucciones del grupo son similares a **add** y sólo varía la operación que realizan.

add $1, $2, $3 #$1 = $2 **+** $3
sub $1, $2, $3 #$1 = $2 **--** $3
and $1, $2, $3 #$1 = $2 **AND** $3
or $1, $2, $3 #$1 = $2 **OR** $3

- **slt**

Compara el contenido de los dos registros que actuan como operandos fuentes y pone un tercero a 1, si el primero es menor que el segundo, sino a 0.

slt $4, $6, $8 #$4 = 1 Si $6 < $8
#$4 = 0 Si $6 > $8

Esta tiene una variante **slti**, que compara el valor de un registro con un inmediato

slti $2, $5, 100 #$2 = 1 Si $5 < 100

2.1.2 Instrucciones de transferencia de datos con la memoria

Para realizar transferencias de datos entre la Memoria y los registros en ambas direcciones existen dos instrucciones.

- **lw** (load word)

Carga el contenido de una posición de Memoria, cuya dirección viene expresada en la instrucción por la suma del contenido de un registro y un valor inmediato de 16 bits, en un registro. Detrás del
nemónico se exprésale registro que se va a cargar y después la dirección de inicio en la memoria de un array y el índice del elemento del array en el que está depositado el dato a transferir.

lw $5, 5000($3) #$5 = M[5000 + ($3)]

Esta instrucción carga en el registro $5 el contenido de la posición de la Memoria expresada por la suma de una dirección que indica el comienzo de un array de datos y un índice, contenido en un registro, que se denomina "registro índice". En el ejemplo la dirección de inicio del array se expresa directamente mediante el valor 5000 y el registro índice $3.

- **sw** (store load)

Almacena la palabra contenida en un registro en una dirección de la Memoria expresada de la misma forma que la instrucción **lw**.

```
sw  $7, Array($1)                    #M[Array + ($1)] = $7
```

En este ejemplo la dirección de inicio del array de datos viene expresada mediante la etiqueta Array, que corresponde a un valor inmediato de 16 bits.

2.1.3 Instrucciones de salto condicional

- **beq**

Si el contenido de los dos registros fuente son iguales, se salta a la instrucción cuya dirección se halla sumando el valor del PC el inmediato de 16 bits que se incluye en la instrucción.

```
beq  $3, $5, 1000                    # Si $3 = 45, PC = PC+1
```

2.1.4 Instrucciones de bifurcación incondicional

En el núcleo básico de instrucciones del **MIPS** sólo se estudia en este grupo la instrucción **jump**, de mnemónico **j**. Esta instrucción bifurca incondicionalmente el programa a la dirección de destino que acompaña a la instrucción.

```
j  14600                             #PC = 14600
```

2.2 Modos de direccionamiento

El **MIPS** simplificado que se estudia dispone de cuatro modos de direccionamiento.

- **1º Direccionamiento por registro**

El operando es el contenido de un registro, como en la instrucción **add**.

- **2º Direccionamiento base o desplazamiento**

El operando reside en una posición de la memoria cuya dirección se obtiene sumando el contenido de un registro y un valor inmediato de 16 bits que incluye la instrucción **lw**

- **3º Direccionamiento inmediato**

El operando es una constante expresada en la propia instrucción mediante un valor de 16 bits, esto sucede con la instrucción **addi**

- **4º Direccionamiento relativo al PC**

El PC se carga con un valor resultante del valor que tenía con uno inmediato de 16 bits que acompaña a la instrucción. Este direccionamiento se usa en la instrucción **beq**.

2.3 Formato de las Instrucciones

Todas las instrucciones del **MIPS** tienen una longitud de 32 bits y su formato responde a tres modelos que reciben los nombres de **formato R**, **formato I**, y **formato J**.

2.3.1 Formato de las instrucciones lógicas y aritméticas

Este grupo de instrucciones responden al **formato R**, en el cual la longitud de 32 bits de la instrucción se descompone en 6 campos:

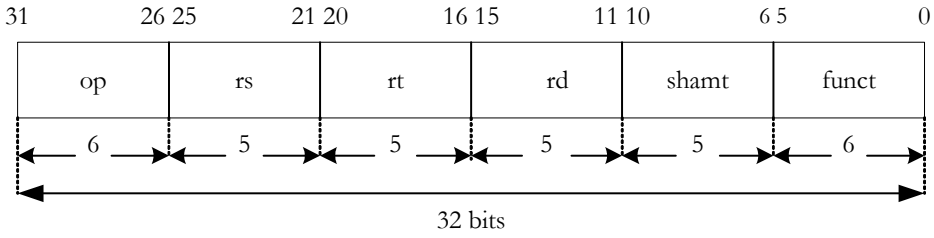


Figura 2 – Formato R, correspondiente a las instrucciones lógicas y aritméticas.

El campo **op** sirve para discriminar la instrucción. El campo **rs** expresa en binario el registro que contiene un operando fuente. El campo **rt** expresa el registro que contiene el segundo operando fuente. El campo **rd** indica el registro que actúa como destino. El campo **shamt** expresa un desplazamiento cuya misión se comentará más adelante. Y el campo **funct** sirve para discriminar las instrucciones que tienen el mismo campo **op**.

2.3.2 Formato de las instrucciones de transferencia

Responden al **formato I**, que descomponen los 32 bits de la instrucción en los campos que se muestran a continuación:

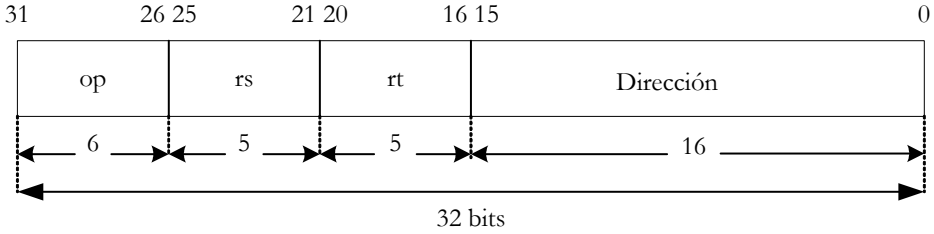


Figura 3—Campos en los que se descomponen las instrucciones de formato I

op: Código de la operación de la instrucción.

- **rs**: registro que contiene el Índice que apunta un elemento del "array" de datos de la Memoria.
- **rt**: registro que recibe el resultado de la operación de transferencia. En **lw** se le carga con el contenido de la dirección de la Memoria y en **sw** su valor se almacena en la dirección de la memoria.
- **dirección**: es un valor inmediato que representa la dirección de comienzo del "array" de datos.

Las instrucciones de direccionamiento inmediato, como **addi** y **slti**, también responden al **formato I**.

2.3.3 Formato de las instrucciones de salto condicional

La instrucción **beq** responde al **formato I**.

2.3.4 Formato de las instrucciones de bifurcación incondicional

La instrucción de bifurcación incondicional tiene el **formato J**, caracterizado por dividir la longitud de la instrucción en sólo dos campos, el del código **op** y el de **dirección**.

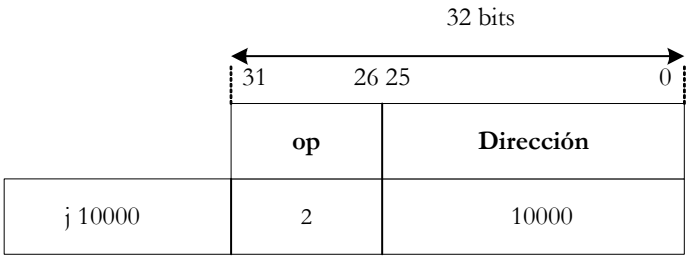


Figura 4—Formato J al que responde la instrucción de bifurcación incondicional.

La **Unidad de Control** cuando reciba el código **op** de cada instrucción determinará a qué formato corresponde y los campos en los que se descompone.

3. DISEÑO DE LA ALU

Las instrucciones lógicas y aritméticas son las principales responsables de las funciones que debe desarrollar la **ALU**, que son las siguientes:

- Suma binaria (**add**).
- Resta lógica (**sub**).
- AND lógica (**and**).
- OR lógica (**or**).
- Poner a 1 ó a 0 (**slt**).

Otras instrucciones también requieren realizar algunas operaciones lógicas o aritméticas. Así, por ejemplo, las instrucciones **lw** y **sw** precisan sumar el contenido de un registro a un inmediato de 16 bits extendido en signo y convertido en un valor de 32 bits.

En este apartado nos ocuparemos esencialmente de la **ALU** del procesador principal de **MIPS** que opera con números enteros.

3.1 Una ALU básica

La **ALU** del **MIPS** funciona con operandos de 32 bits, pero su implementación se consigue mediante el conectado en paralelo de 32 **ALU** elementales de 1 bit. En el siguiente dibujo se muestra una **ALU** básica de 1 bit que es capaz de efectuar las tres operaciones básicas: **suma binaria**, **and** y **or**.

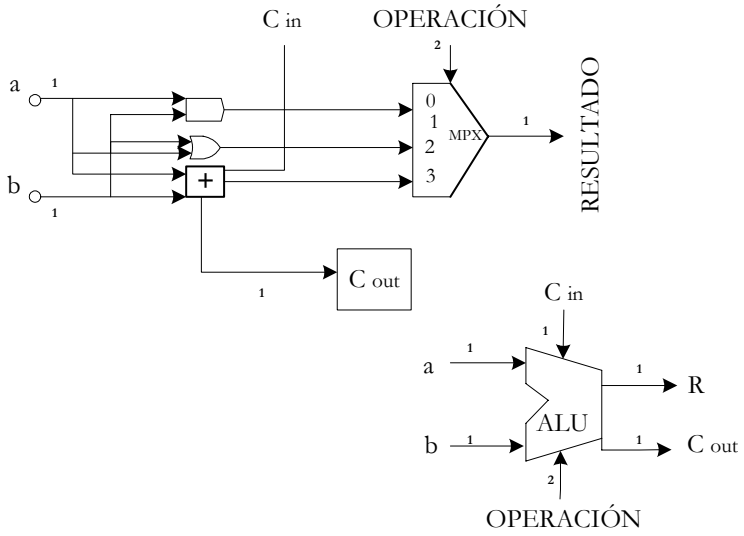


Figura 5—Esquema de la ALU básica de 1 bit, capaz de efectuar las 3 operaciones básicas. Incluyendo el esquema simplificado de este componente.

Por lo tanto, una **ALU** con operandos de 32 bits estará formada por 32 **ALU** básicas conectadas en paralelo.

3.2 Implementación de la resta en la ALU

Para posibilitar la resta binaria en la **ALU** básica sin introducir un restador, se utiliza la técnica del complemento a 2. La resta se realiza mediante una suma utilizando el sumador que ya se disponía en el circuito.

Funcionamiento del complemento a 2:

$$a + \overline{b} + 1 = a + (\overline{b}+1) = a + (-b) = a - b,$$

teniendo en cuenta que (b + 1) es el complemento a 2 de b, o sea, -b.

En la siguiente figura se muestra el esquema modificado de la **ALU** básica para permitir realizar la resta de los dos operandos de 1 bit. Apréciese que cuando se desea efectuar la resta de "a – b", las señales auxiliares **Binv** y **Cin** tienen que valer 1.

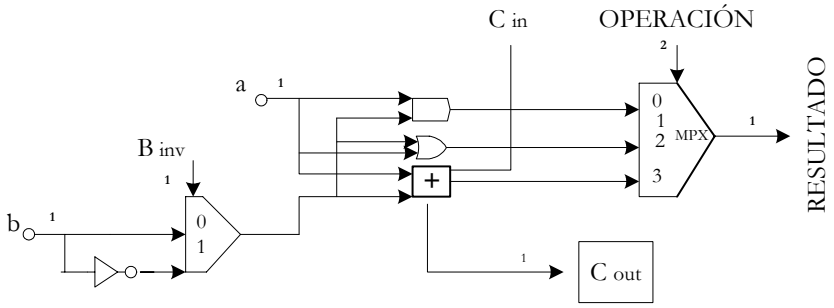


Figura 6—La posibilidad de poder introducir el sumador inverso de b posibilita la resta mediante dicho sumador.

3.3 La operación de detección de menor para inicializar a 1

La función de la instrucción **slt** es comprobar si **rs** es menor que **rt**, en cuyo caso pone a 1 el registro destino **rd**. En caso contrario **rd** se pone a 0. Al final de esta instrucción todos los bits de **rd** son 0 con excepción del bit de menos peso que valdrá 1, únicamente en el caso que **rs<rt** (que ocurrirá cuando el resultado de la resta sea negativo). Para soportar la instrucción **slt** se utiliza la entrada 3 libre del multiplexor de la **ALU** de la figura anterior. A dicha entrada se le aplica la señal **MENOR**. Dicha señal debe valer 0 en todas las **ALU s** elementales de bit, excepto en la que proporciona el bit de menos peso (**ALU0**), cuyo valor depende de la resta de **rs** y **rt**.

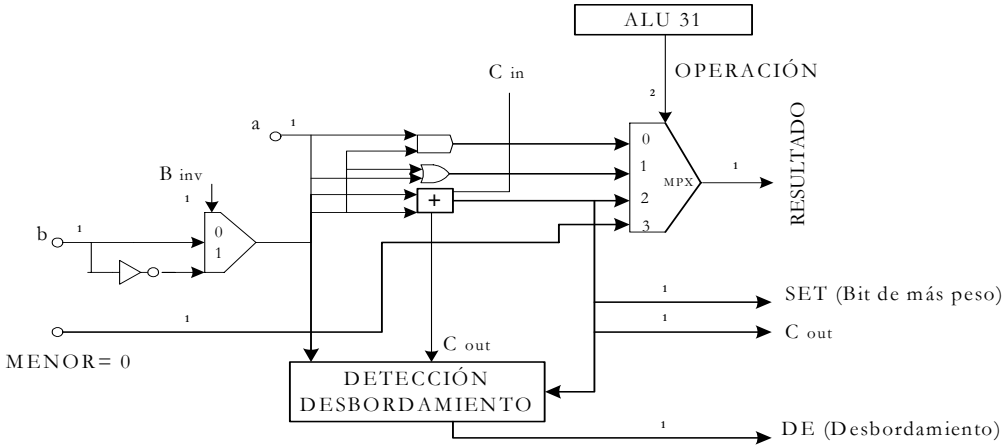


Figura 7—Esquema de la ALU correspondiente a los bits 31, de más peso, con las modificaciones precisas para poder realizar la instrucción slt. Las ALU de los restantes bits no tienen la lógica de "detección del desbordamiento". la entrada MENOR de la ALU0 está conectada a la salida SET de la ALU31, en lugar de ir a tierra.

3.4 La ALU completa del MIPS

En la figura de abajo, se muestra el esquema interno de la **ALU** del **MIPS**, compuesta por 32 **ALU** de 1 bit interconectadas entre sí. Las dos señales **Binv** y **Cin** se constituyen en una sola puesto que siempre tienen el mismo valor. Cuando se efectúa una suma ambas valen 0 y con una resta valen 1. A esta señal única se la denomina **BNEG**.

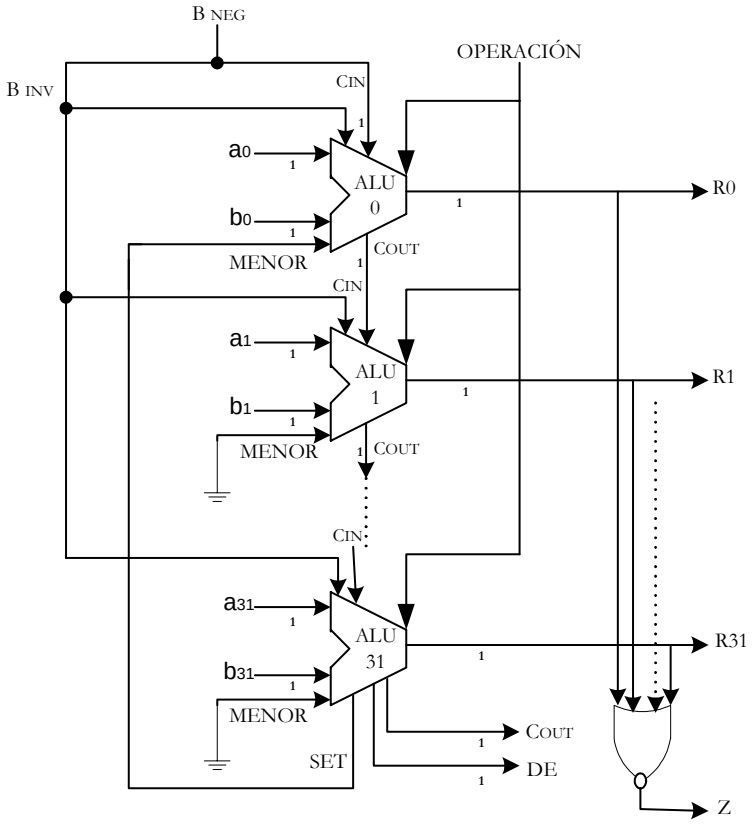
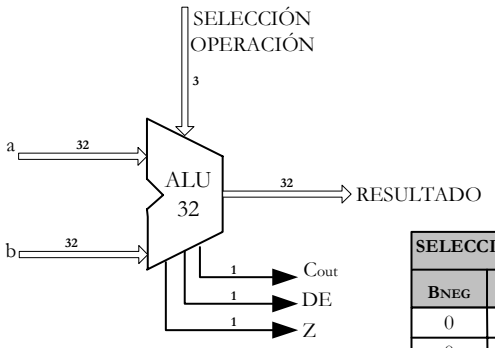


Figura 8—Esquema interno de la ALU del MIPS.

La siguiente figura representa el símbolo simplificado de la **ALU** de 32 bits, junto a la tabla de la verdad que selecciona las operaciones en función de las 3 señales de control.



SELECCIÓN OPERACIÓN			INSTRUCCIÓN
BNEG	OP1	OP0	
0	0	0	AND
0	0	1	OR
0	1	0	ADD
1	1	0	SUB
1	1	1	SLT

Figura 9—Símbolo de la ALU del MIPS y la tabla de la verdad para seleccionar la operación que realiza

3.5 Importancia de la coma flotante

Los modelos comerciales de procesadores **MIPS** disponen de un coprocesador matemático que les permite efectuar numerosas operaciones sobre datos expresados en coma flotante. La principal aportación que supone utilizar números en coma flotante (nota científica) es la de poder manejar con pocos dígitos tanto números muy grandes como pequeños.

MIPS también incorpora en su interior un coprocesador matemático que sigue el formato propuesto por la **norma IEEE 754**, la cual expresa a los números (en binario).

La **norma IEEE 754** dispone de 2 opciones según el tamaño en bits de los operandos. Simple precisión cuando el tamaño es de 32 bits y doble precisión cuando el tamaño alcanza los 64 bits.

4 "DISEÑO DEL CAMINO DE DATOS DEL MIPS MONOCICLO"

4.1.Fundamentos y fases operativas

En este apartado describiremos el funcionamiento del Camino de Datos básico de un procesador **MIPS**, también llamado "**monociclo**".

Se entiende por Camino de Datos la parte del procesador que se encarga de ejecutar las instrucciones del repertorio. Sus principales componentes son los siguientes:

- La **ALU**.
- El Banco de Datos.
- La Memoria de instrucciones y la Memoria de datos. Separadas en el caso del **MIPS monociclo**.

Además, el Camino de Datos dispone de un conjunto de componentes auxiliares (registros, sumadores, etc…), que dirigen la información.

En el Camino de Datos **monociclo** cada instrucción se ejecuta en un solo ciclo de reloj, por lo tanto la duración del ciclo de reloj viene determinada por la duración de la instrucción más larga. Esto provoca que haya una gran perdida de rendimiento en las instrucciones de menor duración ya que consume un ciclo completo sin ser necesario.

Por otra parte, no se puede utilizar más de una vez un recurso con cada instrucción. Por este motivo es necesario disponer de una memoria de instrucciones independiente de la de datos, puesto que en muchas instrucciones hay que acceder a buscar instrucciones y a leer o escribir datos al mismo tiempo.

Las fases en las que se desarrollan las instrucciones pueden agruparse de la siguiente forma:

- 1ª FASE : BUSQUEDA DE LA INSTRUCCIÖN

  Consiste en leer de la Memoria de instrucciones el código de 32 bits correspondiente a la instrucción en curso. La dirección donde se encuentra la instrucción la proporciona el registro

llamado Contador de Programa (PC). Posteriormente el PC se debe incrementar en 4 unidades para que pueda apuntar a la siguiente instrucción de la secuencia.

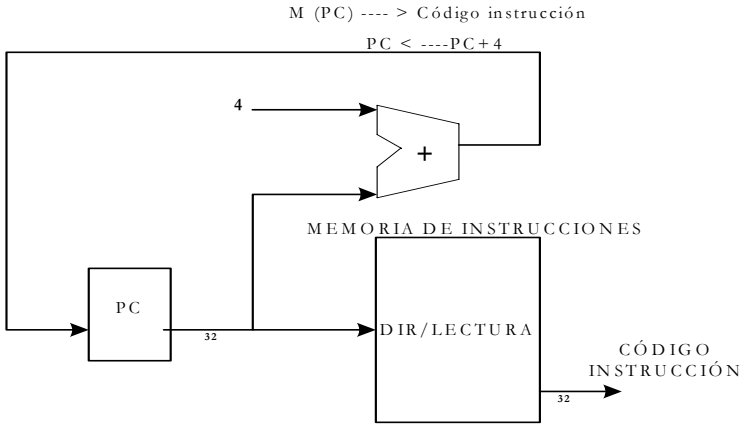


Figura 10—En la fase de búsqueda de la instrucción se lee la posición de la memoria de instrucciones direccionada por el PC y se incrementa en 4 unidades el contenido del PC.

- 2ª FASE : A/ EJECUCIÓN DE LA INSTRUCCIÓN LOGICO_ARITMETICAS

Las operaciones que se realizan en la segunda fase de cada instrucción dependen del tipo que se trate. Las instrucciones lógico-aritméticas las desarrolla la **ALU** utilizando como operandos fuente y destino elementos del Banco de Registros.

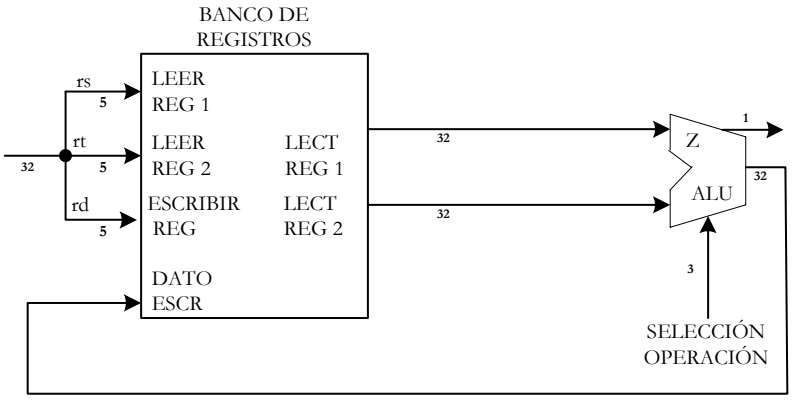


Fig. 11—La fase de ejecución de las instrucciones lógicas y aritméticas consiste en leer los registros rs y rt, y efectuar por medio de la ALU, la operación correspondiente con sus contenidos y el resultado escribirlo en el registro destino rd.

- 2ª FASE : B/ REALIZACIÓN DE LAS INSTRUCCIONES DE TRANSFERENCIA

En este grupo existen dos instrucciones (**lw** y **sw**), ambas responden al formato I.

Por ejemplo la instrucción **lw** $1, DESPL16($3) carga en el registro $1 el contenido de la posición de la Memoria de datos cuya dirección se obtiene mediante la suma del contenido del registro $3 y el valor inmediato de 16 bits DESPL16 extendido el signo hasta alcanzar el tamaño de 32 bits. Para llevar a cabo esta instrucción la **ALU** debe obtener la dirección a acceder en la memoria de Datos mediante la suma de **rs** y el valor inmediato extendido el

signo. El dato obtenido de la Memoria se aplica por la entrada DATO ESCR y se graba en el registro **rd**.

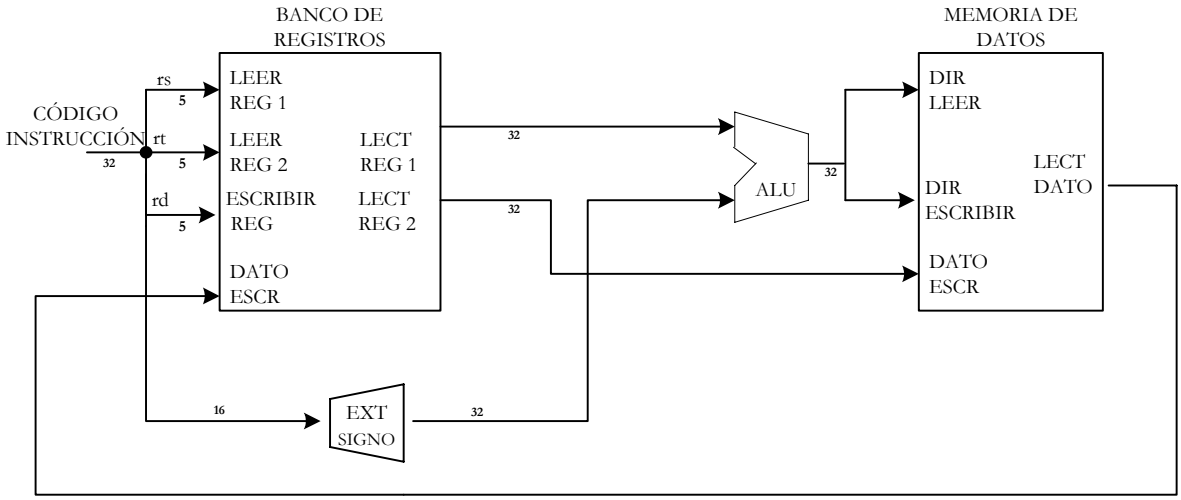


Figura 12—Interconexión precisa entre la ALU, el Banco de Registros y la Memoria de Datos para implementar la ejecución de las instrucciones de transferencia sw y lw.

- 2ª FASE : C/ REALIZACIÓN DE LAS INSTRUCCIONES DE SALTO CONDICIONAL

La instrucción **beq** responde al formato I y en su expresión se utilizan dos registros y un valor inmediato de 16 bits **beq** $3, $4, DESPL16. A este valor inmediato se le extiende el signo hasta completar los 32 bits y se le realiza un desplazamiento de 2 posiciones a la izquierda para hacerlo múltiplo de 4, sumándolo al PC+4 para obtener la dirección de la siguiente instrucción. Teniendo en cuenta que es necesario que el contenido de los dos registros sea igual o lo que es lo mismo que el flag Z=1, al efectuar su resta. Como la **ALU** se emplea para restar los registros operandos (**rs** y **rt**) se necesita un sumador complementario que se encargue de sumar al valor del PC el valor del salto.

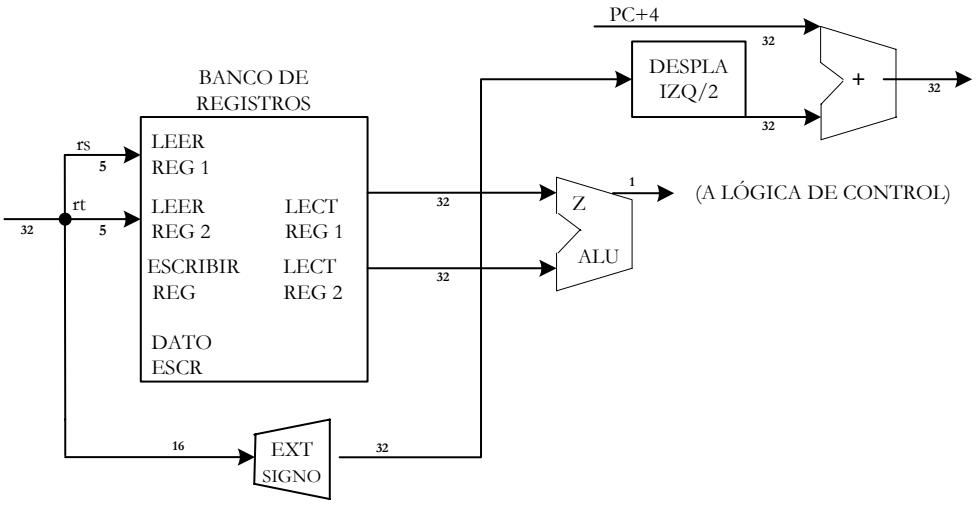


Figura 13—En la instrucción de salto condicional beq la ALU resta el contenido de los registros rs y rt y si Z=1, el sumador deposita su resultado en el PC. El DESPL IZQ/2 desprecia los bits procedentes de la extensión de signo e introduce dos ceros por la derecha para múltiplo de 4.

4.2. Esquema general del camino de datos del monociclo

Solapando los esquemas anteriores correspondientes a las distintas fases se obtiene el esquema general del camino de datos **monociclo**.

Como el CPI en los procesadores es igual a 1, cada vez que se precise utilizar un recurso más de una vez en una instrucción hay que duplicarlo. De aquí surge la necesidad de disponer de una Memoria de Instrucciones y otra de Datos.

Lo que sí es posible es compartir un mismo recurso por diferentes instrucciones. Para ello se requiere seleccionar entre diversas entradas al recurso, con este fin se utilizan frecuentemente los multiplexores.

Combinamos los dos esquemas de los caminos de datos para las instrucciones lógico-aritméticas y las de transferencia, añadiendo los recursos precisos para soportar, la fase de búsqueda de las instrucciones.

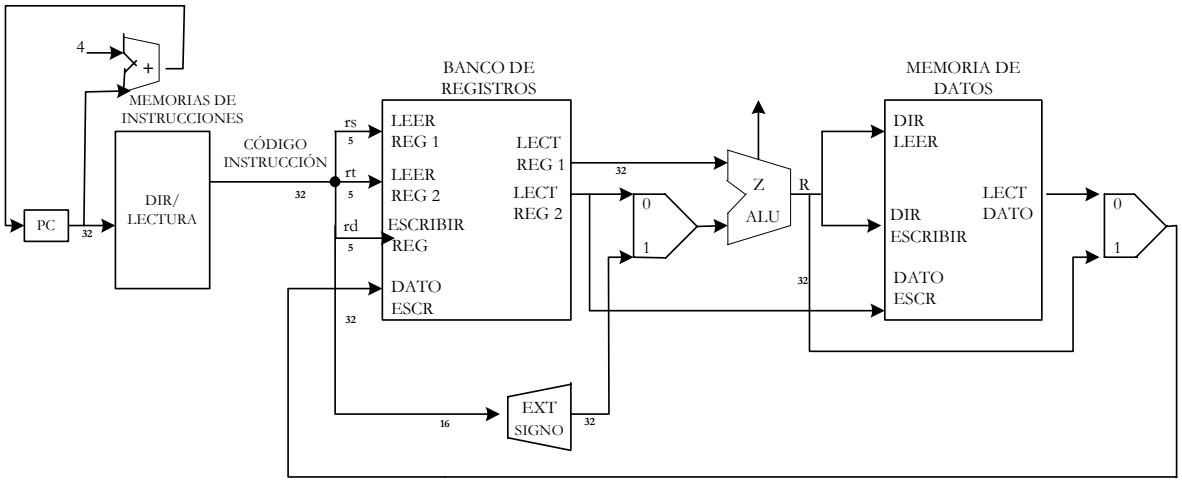


Figura 14—Esquema completo del Camino de Datos monociclo capaz de ejecutar íntegramente las instrucciones lógico-aritméticas y las transferencias.

Añadiendo a la figura anterior los recursos necesarios para resolver la ejecución de la instrucción de salto condicional se llega al siguiente Camino de Datos del **monociclo** del **MIPS**.

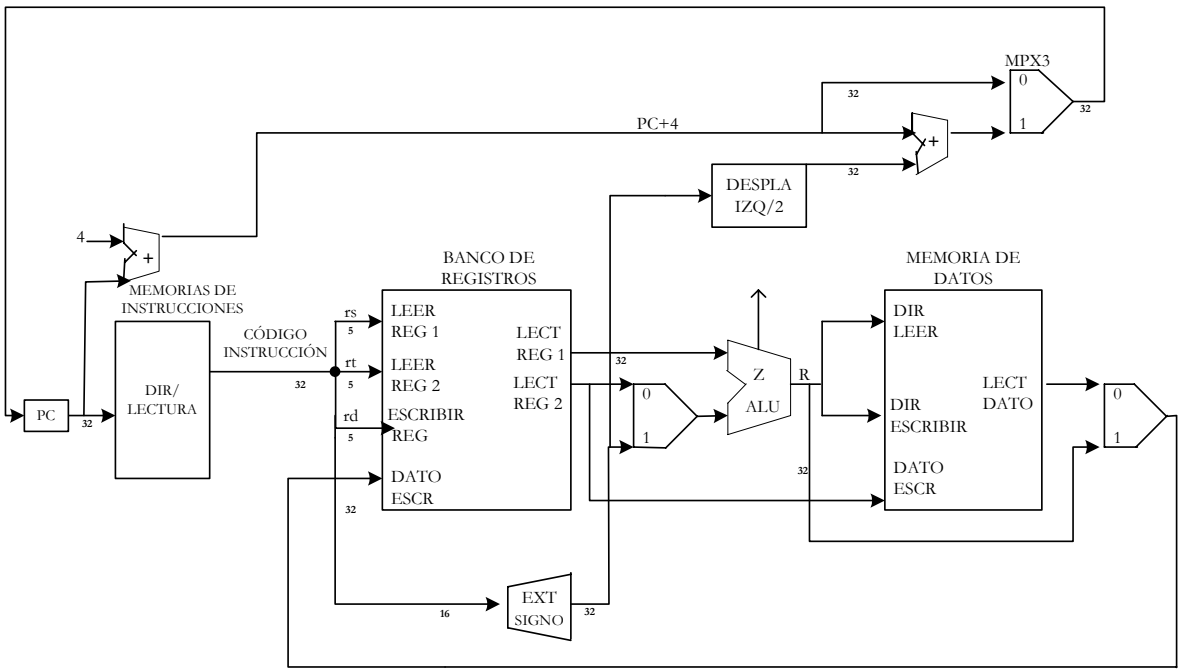


Figura 15—Esquema general del Camino de Datos del MIPS monociclo, que soporta la ejecución de instrucciones lógico-aritméticas, transferencias y salto condicional.

5. "LA UNIDAD DE CONTROL PARA EL PROCESADOR MONOCICLO"

5.1. El papel de la Unidad de Control

La Unidad de Control es el bloque más complejo del procesador. Su misión es la de gobernar las señales de control que regulan la actuación del Camino de Datos en la ejecución de las instrucciones. Tiene que interpretar o decodificar el código máquina de la instrucción en curso para activar aquellas señales que sean precisas para el desarrollo de la misma mediante los recursos existentes en el Camino de Datos del procesador.

En la siguiente figura se muestra el diagrama de bloque de la Unidad de Control del procesador **MIPS monociclo**

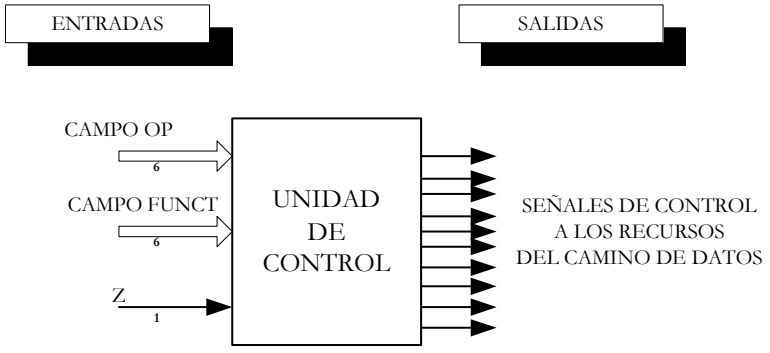


Figura 16—5ª unidad de control del MIPS monociclo.

En cada ciclo se ejecuta una instrucción (**MIPS monociclo**) y en dicho tiempo la Unidad de Control debe activar con el estado adecuado las señales de control que aplicará al Camino de Datos. En el procesador **monociclo** la Unidad de Control responde a un circuito combinacional que sigue una tabla de verdad que tiene como entradas las señales que determinan el tipo de instrucción que se trata y cuyas salidas son los estados correspondientes que deben soportar las señales de control aplicadas a los componentes del Camino de Datos.

Uno de los recursos principales del Camino de Datos a la hora de realizar una instrucción es la **ALU**.

El Banco de Registros siempre dispone de dos entradas que indican los registros que se leen y cuyos contenidos aparecen por las salidas y una tercera entrada que indica el registro que se desea escribir con la información que se aplica por DATO ESCR. Este registro solo se carga cuando se activa una señal de control llamada "Reg Write".

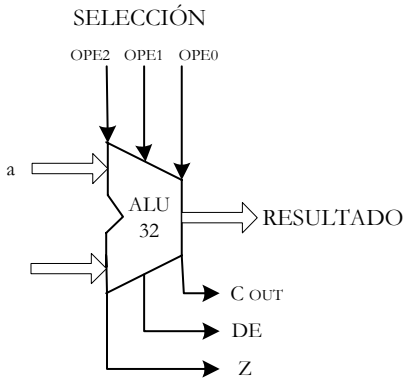
La Memoria dispone de dos secciones independientes, una encargada de contener las instrucciones y otra los datos. La memoria de instrucciones sólo se puede leer, mientras que la de datos se puede leer y escribir. Para controlar la Memora de datos existen dos señales de control denominadas "Mem Read" y "Mem Write".

El Camino de Datos dispone de unos componentes auxiliares que determinan la información que se aplica a los recursos principales. Se trata de los multiplexores

Para poder diseñar la Unidad de Control de un procesador hay que conocer el contenido de las diversas señales de control que gobiernan los recursos del Camino de Datos y el estado que deben tomar en cada instrucción.

5.2. Señales de control de la ALU

La **ALU** del **MIPS** soporta la realización de cinco operaciones diferente de tipo lógico y aritmético, que se seleccionan mediante tres señales de control que llamaremos OPERACIÓN1, OPERACIÓN2, y OPERACIÓN0. Mostramos el signo de la **ALU** y la tabla de verdad que selecciona la operación que realiza.



OPE2	OPE1	OPE0	OPERACIÓN
0	0	0	AND
0	0	1	OR
0	1	0	ADD
1	1	0	SUB
1	1	1	MENOR

Figura 17—Símbolo de la ALU y la tabla de la verdad que muestra la operación que realiza en la función del valor lógico que adoptan sus tres señales de control.

De acuerdo con el valor de 6 bits de más peso del código de la instrucción, que corresponde al campo del código OP, la Unidad de Control genera dos bits que constituyen las señales de control

designadas como **ALU op** y que discriminan entre los tres tipos principales de instrucciones: lógico-aritméticas, de transferencia y el salto condicional.

Para cualquier instrucción de tipo lógico-aritmética la Unidad de Control genera el mismo código del campo **ALU op** (10). Por lo tanto, para identificar la operación que debe realizar la **ALU** y obtener el valor de las tres señales que la controlan, hay que tener en cuenta el campo auxiliar func.

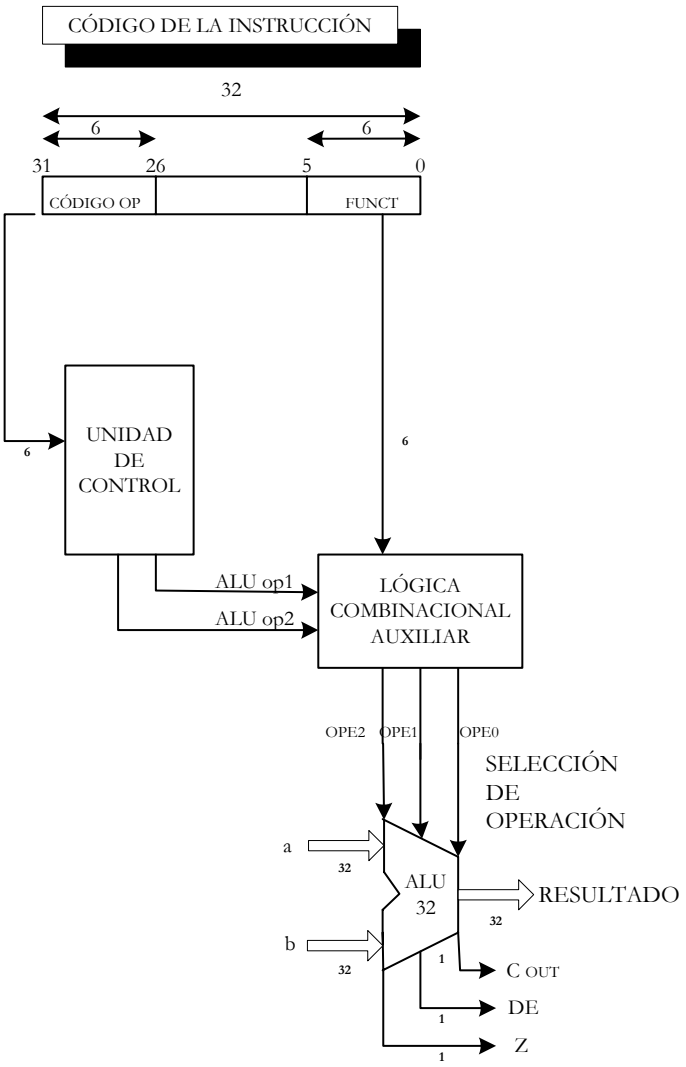


Figura 18—Una lógica combinacional complementaria se encarga de generar las tres señales de control que seleccionan la operación ALU y discriminan las diversas alternativas de las instrucciones de tipo R, mediante el análisis de los bits del campo funct.

En la figura siguiente se muestran los valores de las diversas operaciones que se manejan en los elementos mostrados en el esquema de la figura 18.

Código OP	ALU op	Instrucción	Funct F5-F0	Operación de la ALU	Ope2	Ope1	Ope0
lw	0 0	Cargar	xxxxxx	Sumar	0	1	0
sw	0 0	Almacenar	xxxxxx	Sumar	0	1	0
beq	0 1	Salto condi.	xxxxxx	Restar	1	1	0
tipo R	1 0	Sumar	100 000	Sumar	0	1	0
tipo R	1 0	Restar	100 010	Restar	1	1	0
tipo R	1 0	And	100 100	And	0	0	0
tipo R	1 0	Or	100 101	Or	0	0	1
tipo R	1 0	Menor	101 010	Menor	1	1	1

Tabla 1—Tabla en la que se indican los valores que toman las diversas señales que participan en el funcionamiento de los elementos mostrados en la figura anterior.

5.3. Diseño de la Unidad de Control cableada

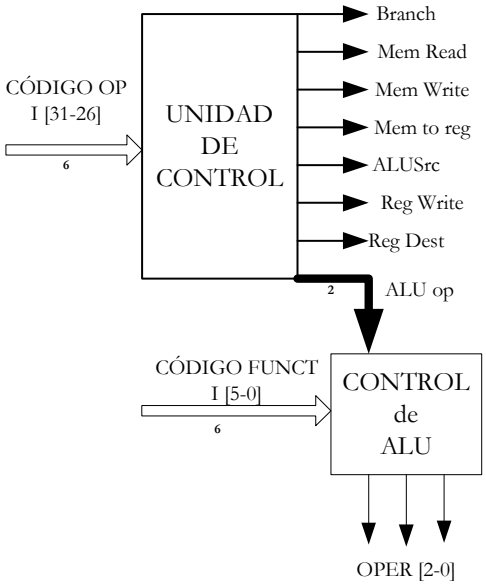


Figura 19—Señales de entrada y salida de la Unidad de Control

Como se conoce el código **op** correspondiente a cada tipo de instrucción se puede generar la tabla de verdad, en la cual las entradas serán los 6 bits del código y las salidas las 9 señales que debe generar la Unidad de Control.

Instrucción	op5	op4	op3	op2	op1	op0	Reg Dest	ALU Src	Mem ro Reg	Reg Write	Mem read	Mem Write	Branch	ALU op1	ALU op0
Formato R	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
lw	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
sw	1	0	1	0	1	1	x	1	x	0	0	1	0	0	0
beq	0	0	0	1	0	0	x	0	x	0	0	0	1	0	1

Tabla 2—Tabla de la verdad que establece las salidas que debe generar la Unidad de Control en función del estado que se recibe por sus entradas y que corresponde al código OP de la instrucción a ejecutar.

Implementación física de la Unidad de Control y tabla de control del **MIPS monociclo.**

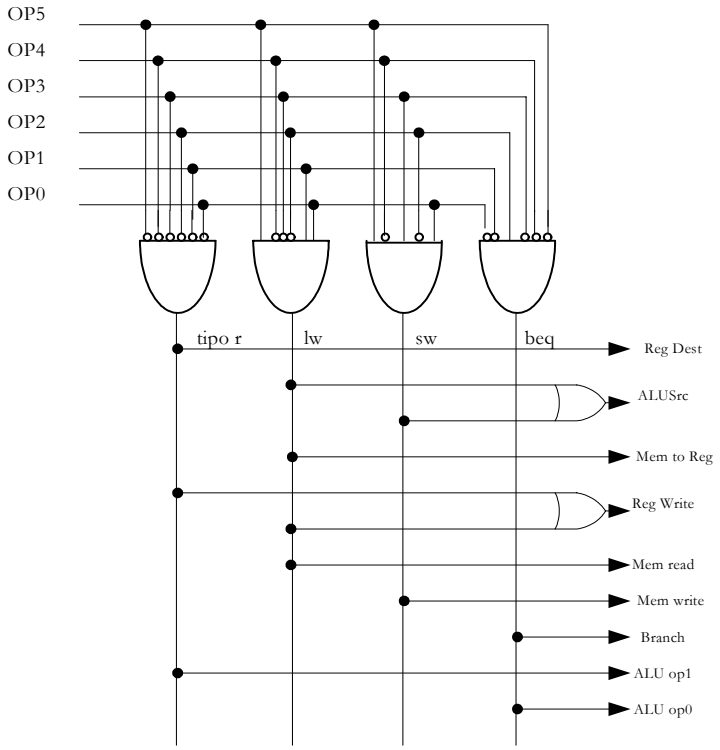


Figura 20—Implementación física de la Unidad de Control, mediante una PLA que resuelve las ecuaciones lógicas de las señales de control en función de los estados del código OP de cada instrucción.

5.4. Implementación de la instrucción de salto incondicional

La instrucción **j A** ocasiona un salto incondicional a una dirección de la Memoria de Instrucciones que se obtiene con el dato inmediato **A**. Su formato:

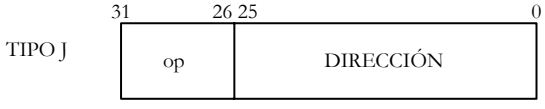


Figura 21—Formato J correspondiente a la instrucción j Dirección.

La dirección a la que se salta con esta instrucción y que se carga en el PC, se calcula de la siguiente forma:

- Se mantiene el valor de los 4 bits del PC.
- Detrás de los 4 bits de más peso del PC que no varían se colocan los 26 bits del valor inmediato que viene en la instrucción.
- Se añaden dos bits cero para ocupar las dos posiciones de menos peso haciendo que el valor sea múltiplo de 4.

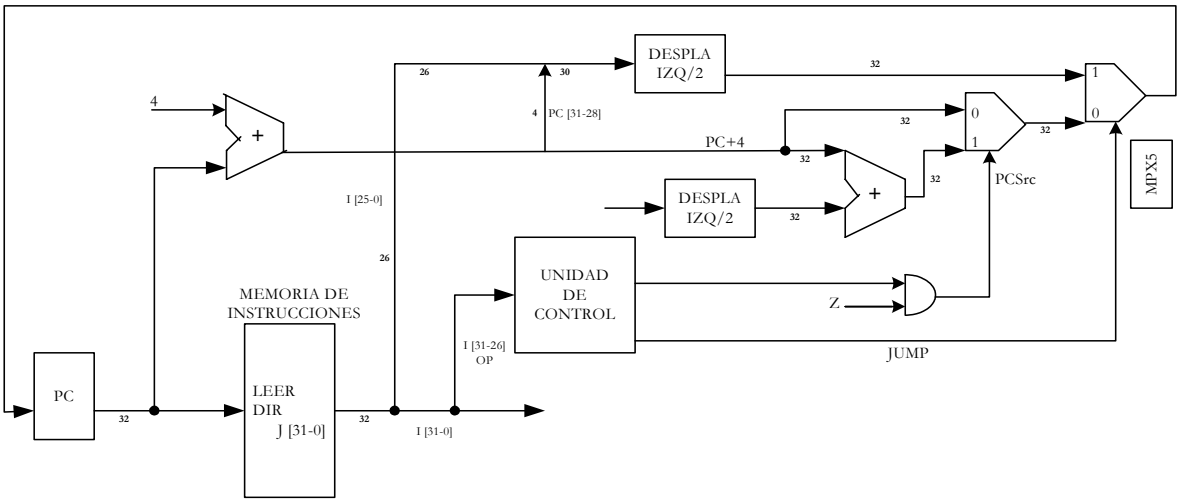


Figura 22—Elementos que hay que añadir al esquema del procesador monociclo para implementar la instrucción de bifurcación incondicional

5.5. Esquema general del procesador monociclo

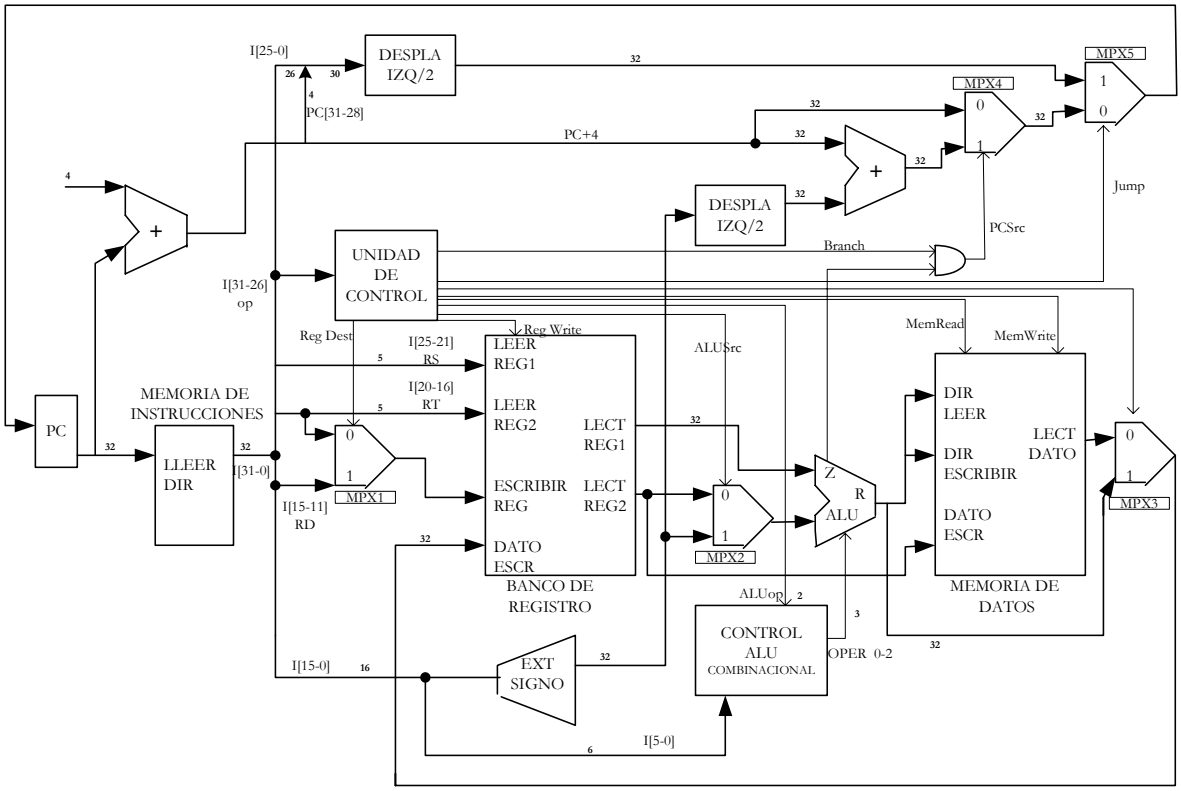


Figura 23—Esquema general del procesador MIPS

5.6. Problemática del procesador monociclo

La característica principal del procesador **monociclo** es su capacidad de realizar todas las instrucciones en un solo ciclo de reloj. Esto conlleva que la duración del ciclo del reloj sea la correspondiente a la instrucción más larga, lo que supone una gran pérdida del rendimiento del computador, pues muchas instrucciones simples tardarán en ejecutarse el mismo tiempo que la instrucción más compleja del repertorio. La instrucción más larga del **MIPS monociclo** es la **lw**, pues lee la Memoria de Instrucciones, lee el Banco de Registros, realiza una suma la **ALU**, se lee la Memoria de Datos y finalmente se escribe en el Banco de Registros. Para mejorar el Camino de Datos **monociclo** pasamos al **multiciclo**.

6- CAMINO DE DATOS DEL MIPS MULTICICLO

La aparición del MIPS multiciclo viene como solución a la gran pérdida de rendimiento del MIPS monociclo, ya que en este todas las instrucciones se ejecutaban en un mismo ciclo. En una implementación multiciclo, cada paso de la ejecución, empleará un ciclo de reloj.

La implementación multiciclo permite que una unidad funcional sea utilizada más de una vez por instrucción, mientras se utilice en diferentes ciclos de reloj. La posibilidad de que las instrucciones

empleen diferentes números de ciclos de reloj y la posibilidad de compartir unidades funcionales en la ejecución de una única instrucción son las mayores ventajas del diseño multiciclo. Esto supone una gran ventaja a nivel hardware, reflejándose en su arquitectura en dos aspectos:

1º. Utilización de una única memoria para instrucciones y datos.

2º. Se utiliza un registro para guardar la instrucción una vez que se lee. Este **Registro de Instrucciones (RI)** se necesita porque la memoria puede ser reutilizada para acceder más tarde al dato en la ejecución de la instrucción.

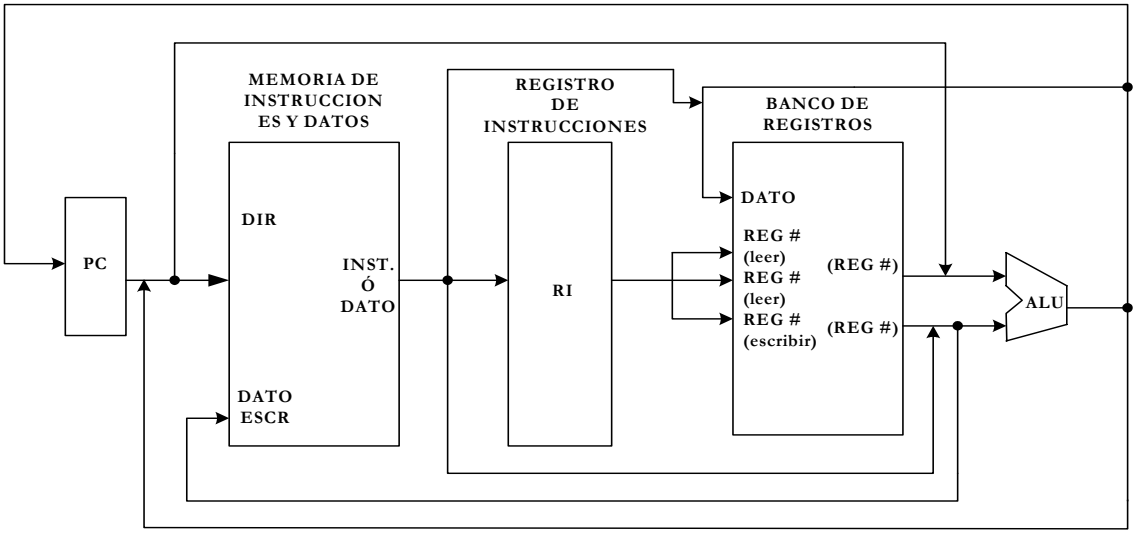


Figura 24.-Diagrama general del camino de datos multiciclo del MIPS.

Hay una sola ALU, en lugar de una ALU y dos sumadores.

Como en una instrucción varios recursos pueden ser utilizados para diversos propósitos, es necesario añadir algunos multiplexores:

1º. Como utilizamos una memoria tanto para datos como para instrucciones necesita un multiplexor de dos entradas para seleccionar entre las dos posibles fuentes de una dirección de memoria, a saber, el PC (para acceder a la instrucción) y el resultado de la ALU (para acceder a los datos).

2º. El multiplexor de la segunda entrada de la ALU hay que ampliarlo con 4 entradas ya que vamos a eliminar todos los sumadores auxiliares (la ALU realizará todas las operaciones).

3º. Se añade otro de 2 entradas a la primera entrada de la ALU, para seleccionar entre introducir a la ALU el contenido del registro **rs** o el del **PC**.

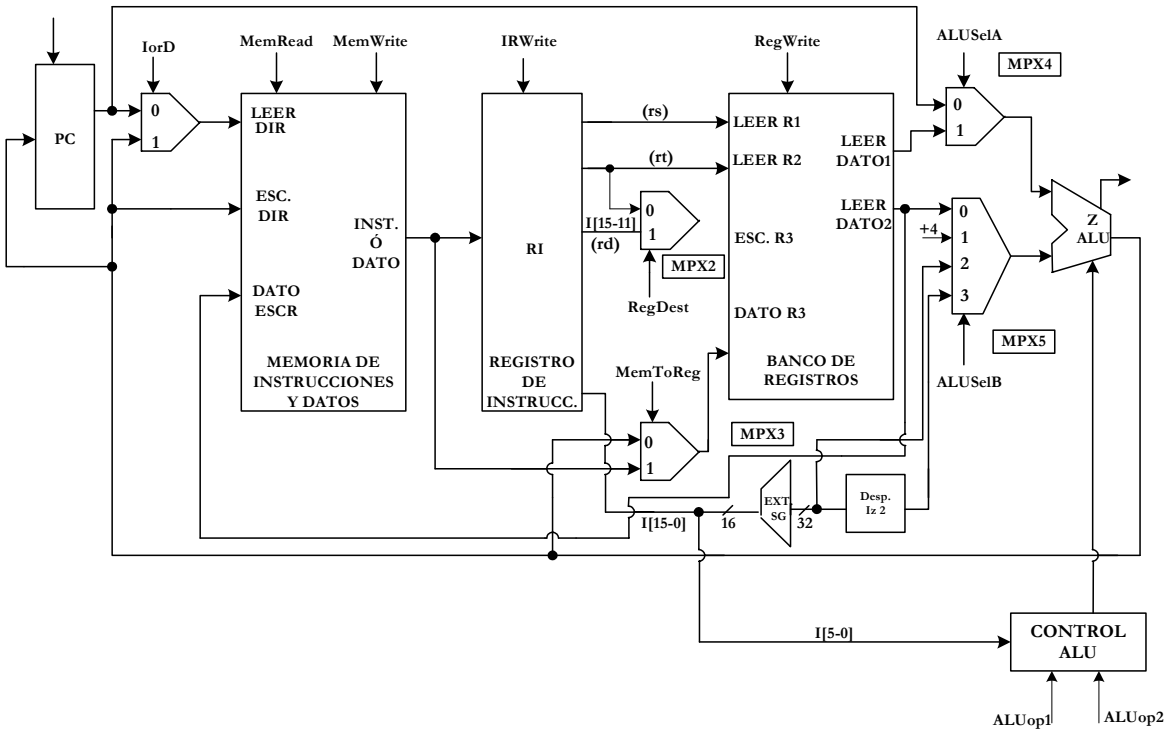


Figura 25.- Esquema del camino de datos del multiciclo de MIPS.

6.1- Descomposición de las instrucciones en operaciones elementales

Al realizarse en cada ciclo una instrucción elemental de las que configuran la instrucción en ejecución , se debe conseguir equilibrar los tiempos de las operaciones elementales, es decir, que duren más o menos lo mismo. Para ello, solo se puede dejar que en un mismo ciclo se utilice en secuencia un único recurso de los más lentos: ALU, Memoria y Banco de Registros. Puesto que la instrucción que más tarde en ejecutarse será la que determine el ciclo de trabajo del procesador.

Se precisa el uso de registros en los siguientes dos casos:

1º. La señal se calcula en un ciclo de reloj y se utiliza en otro.

2º. Las entradas al bloque funcional de donde sale esta señal pueden cambiar antes que la señal se escriba en un elemento de estado.

Necesitamos almacenar el valor en RI por que la memoria, que produce el valor cambia su salida antes que completemos todos los usos de los campos de la instrucción.

No se precisa un registro en la salida de la ALU, en instrucciones de formato R, porque el código de registros a leer (rs, y rt) no cambia en toda la instrucción, sino que provienen del registro RI.

Al descomponer una instrucción en operaciones más elementales, algunos recursos pueden operar en **paralelo** solapándose sus tiempos, mientras que otros trabajan en **serie** acumulando tiempos.

La operación elemental más larga determina la duración del ciclo.

Las instrucciones se dividen en cinco operaciones elementales o pasos:

1º. Paso de búsqueda de instrucción:

Busca la instrucción en memoria e incrementa el contador de programa. También se le

denomina fase de búsqueda.

IR = Memoria[PC];

PC = PC + 4;

En esta operación se realizan dos instrucciones en paralelo. Por un lado se lee la memoria para recoger el código de la instrucción y almacenarlo en el registro **RI**. Por otro lado y puesto que está inactiva la ALU se procederá a sumar 4 unidades al PC, para que apunte a la dirección de la siguiente instrucción a ejecutar.

Las señales de control toman los siguientes valores:

MemRead = 1; IRWrite = 1; IorD = 0 [M(PC) = RI]

ALUSelA = 0; ALUSelB = 01; ALUop = 00 (suma) [PC = PC + 4]

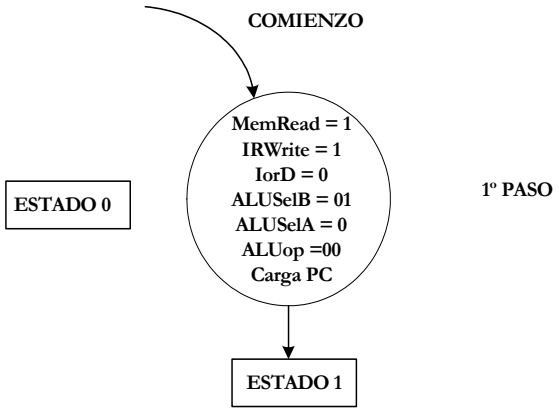


Figura 26.- Representación de la operación elemental de búsqueda de la instrucción. Se denomina estado 0.

2º. Paso de decodificación de la instrucción y búsqueda de registros :

El segundo paso sería la decodificación que es común a todas las instrucciones y se encuentra en el registro RI. Se denomina **estado 1**.

Durante la decodificación se lleva a cabo la lectura de los registros especificados por rs y rt, obteniéndose los valores de A y de B.

A = [rs]; rs =I[25-21]

B = [rt]; rt = I[20-16]

También calcularemos la dirección de salto, porque no sabemos si esta instrucción va a realizar el salto. La almacenaremos en un registro auxiliar (Target), por si la debemos utilizar más adelante.

Target (destino) = PC + IR[15-0]EXT-SIGNO

Para calcular la dirección Destino de salto condicional, se accede al banco de registros, el cual proporcionará por sus dos salidas de lectura el contenido de los registros especificados por sus entradas, sin necesidad de activar señales de control. Se calcula la dirección destino y se almacena en Target. Para ello se debe inicializar **ALUSelB** al **valor 11** (para que el campo de desplazamiento, que tiene los 16 bits de menos peso, esté con el signo extendido y desplazado), **ALUSelA a 0** y **ALUOp a 00**. Para almacenar en Target la señal de control **Target Write tiene que valer 1**.

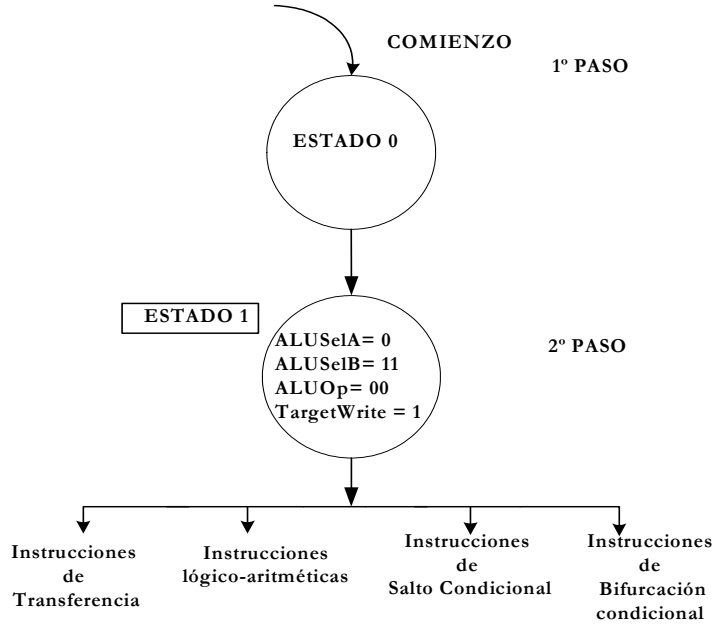


Figura 27.- El estado 1 es común para todas las instrucciones y requiere la activación de 4 grupos de señales diferentes.

Después de este ciclo de reloj, el siguiente estado depende de la siguiente instrucción a ejecutar.

3º. Paso: Selección de estado según el tipo de instrucción :

La salida de la ALU no es necesario guardarla en un registro temporal, porque las entradas van a ser estables e invariables durante el transcurso de la instrucción, al seleccionarse con los campos rs y rt provenientes de RI.

A) Ejecución de la operación en las instrucciones lógico-aritméticas (ESTADO 6).

Se obtiene el resultado por la ALU de la operación que hay que hacer con el contenido de los dos registros fuentes:

ALU = ALU = A opB

Las señales de control a activar son:

ALUSelA=1; ALUSelB=00; ALUOp=10

B) Cálculo de la dirección de memoria en las instrucciones de transferencia (ESTADO 2).

El tercer estado en las instrucciones de transferencia de información con la Memoria de Datos, la ALU calcula la dirección de acceso.

ALU_{out}= A + I[15-0]EXT-SIGNO

Para realizar esta suma hay que activar la siguientes señales de control:

ALUSelA=1; ALUSelB=10; ALUOp=00

C) Terminación del salto condicional BEQ (ESTADO 2).

Su tercer y último estado los dedica a comparar el contenido de los dos registros, mediante su resta por parte de la ALU (el resultado no se guarda) y la activación consecuente del flan Z. Si **Z=1** el contenido de Target hay que llevarlo a PC para que se pueda producir el salto:

ALU_{out}=A – B; si Z=1, PC = Target

Las señales de control son:

ALUSelA=1; ALUSelB=00; ALUOp= 01(resta)

Si **Z=0**, el PC apunta a la siguiente instrucción en secuencia.

6.1.1- Control de escritura del PC

El PC se puede cargar desde tres puntos diferentes:

1º. Puede cargarse con **PC + 4** cuando se sigue la secuencia normal del programa.

2º.Si se trata de un salto condicional y **Z=1**, se carga desde el registro auxiliar **Target**.

3º.Si se trata de una bifurcación incondicional se carga con los 4 bits de más peso del PC, seguido por los 26 bits de menos peso del código de la instrucción I[25-0] y finalmente, por 2 ceros(la extensión de signo).

Para seleccionar este punto se usa un multiplexor de 4 entradas, aunque una se invalida. Se controla con dos bits del campo **PCSorce**. Si vale **00** selecciona PC + 4, si vale **01** el Target y si es **10** la dirección de la bifurcación incondicional.

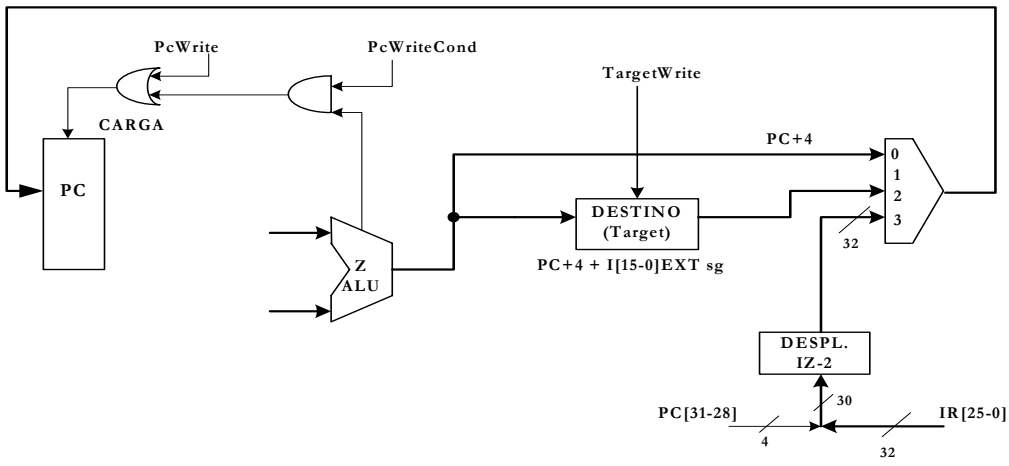


Figura 28.-Control de la carga del PC.

Si no es una instrucción beq se activa **PCWrite** y el PC se carga con lo que procede del multiplexor. Si se trata de una beq, **PCWrite =0** y **PCWrteCond=1**, pero el PC no se carga con lo que le llega del multiplexor, a menos que Z=1, sino permanece con el valor que se había cargado.

D) Terminación de las instrucciones de bifurcación incondicional (ESTADO 9).

La instrucción de bifurcación condicional, j, se completa en el tercer estado, cargando al PC con la dirección del salto.

Las señales de control que se deben activar son:

PCWrite = 1 y PCSrc = 10

Tras esta operación se inicia la siguiente instrucción, que será la que ha determinado la bifurcación.

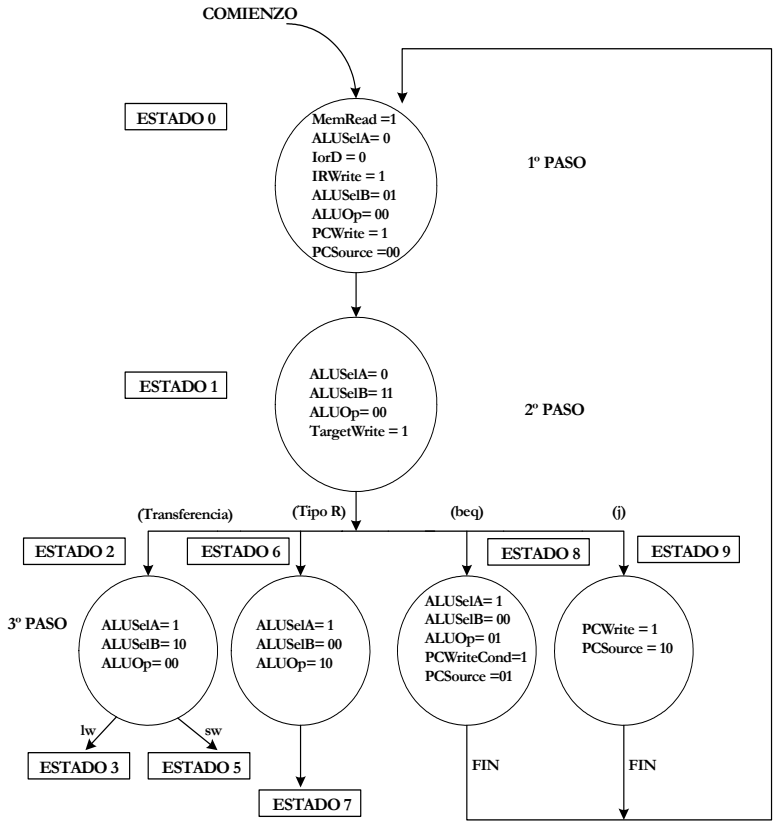


Figura 29.-Diagrama con los tres primeros estados de los posibles tipos de instrucciones del MIPS.

Las instrucciones de bifurcación y salto terminan en el tercer estado.

4º. Paso: Instrucciones tipo r y de transferencia :

Sólo lo realizan las instrucciones lógico-artméticas, que finalizan en este estado, y las de transferencia.

- **Instrucciones lógico-artiméticas(Estado7)**

  Se escribe el resultado de la ALU en el registro destino(rd).

  ALU_{out}=rd(I[15-11])

  Se activan las siguientes señales: ResDest= 1; Reg Write=1; MentoReg=0

  Las señales que gobiernan la ALU no se modifican.

- **Instrucciones de transferencia**

  Son dos las instrucciones de este grupo:

  1.- La instrucción de carga (lw) que lee una dirección de la Memoria de Datos y su contenido lo carga en el registro principal rt.

  2.- La instrucción de almacenamiento (sw) el contenido del registro rt lo escribe en la dirección de acceso a la Memoria de Datos. El cuarto paso es diferente según se trate de carga o almacenamiento.

- **Instrucción de carga (ESTADO 3)**

  En el cuarto paso, se lee la dirección de la Memoria de Daros que se ha calculado en el paso anterior, obteniéndose un paso para el quinto.

M(ALU$_{out}$)= Dato

Se lee la Memoria de Datos y se activan **MemRead = 1** y **IorD = 1**. Las señales de la ALU permanecen invariables.

- **Instrucciones de almacenamiento (ESTADO 5)**

Se escribe el contenido del registro rt en la posición direccionada por la salida de la ALU.

(rt) = M(ALU$_{out}$)

Se activan **Mem Write=1** y **IorD = 1**. Las señales de la ALU no varían.

5º. Paso: Terminación de la instrucción (ESTADO4):

La instrucción de carga (LW) es la unica que tiene cinco pasos. Es la más larga, y se encarga de escribir el dato leído de la Memoria de Datos en el registro rt.

Dato = (rt); rt = I[20-16]

Se activan **MemToReg = 1**, **Reg Write= 1**, **Reg Dest = 0**. Las señales de la ALU no varían.

La figura 1B.30 muestra el diagrama de estados completo al que responde el repertorio básico de instrucciones del MIPS.

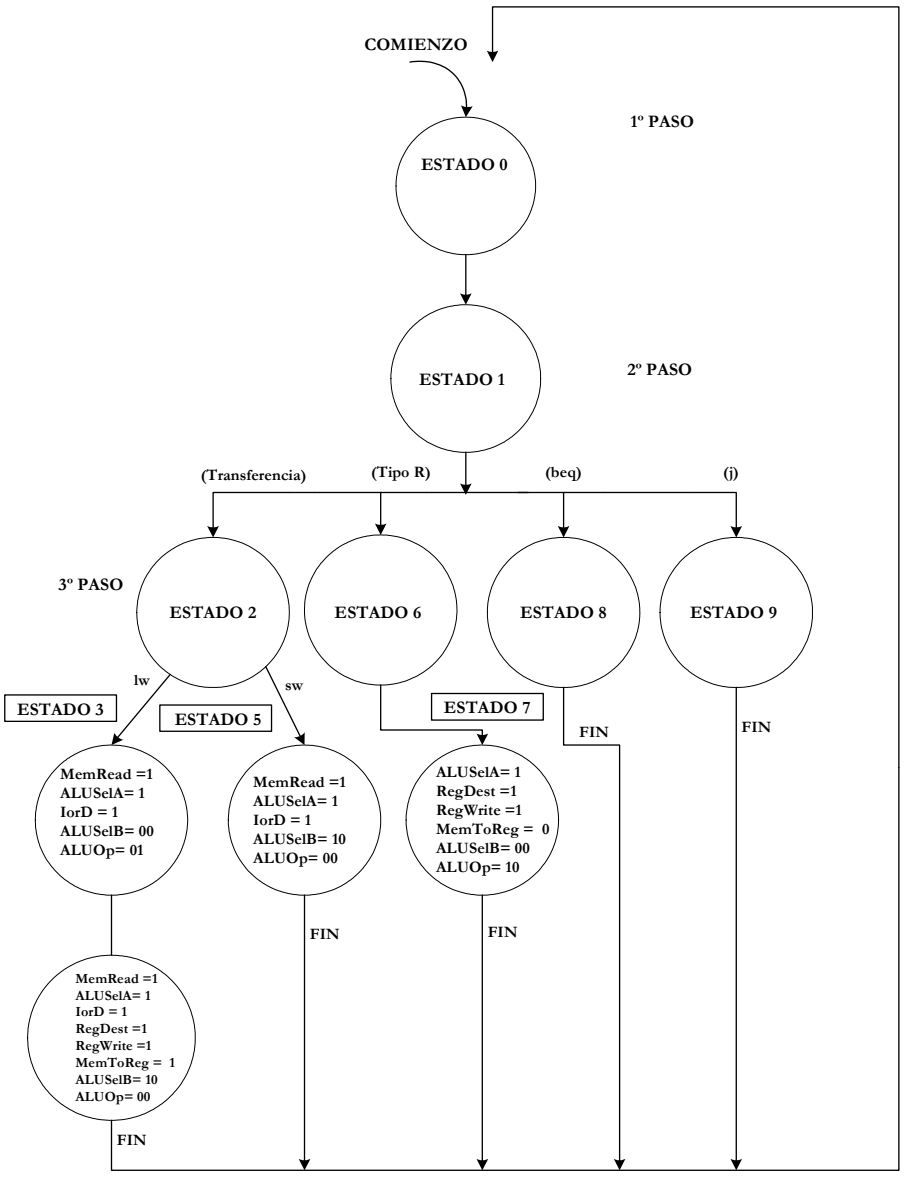


Figura 30.-Diagrama con los tres primeros estados de los posibles tipos de instrucciones del MIPS.

7- LA UNIDAD DE CONTROL PARA EL MIPS MULTICICLO

Se distinguen dos modelos de diseño o construcción de Unidades de Control:

1º. La Unidad de Control Cableada

2º. La Unidad de Control microprogramaza

7.1- Unidad de Control Cableada

Es un circuito, basado en hardware, encargado de generar las señales de control de cada estado. Presenta como una ventaja la rapidez de respuesta, pero su gran inconveniente consiste en su elevado coste. Por ello este tipo de Unidad de Control solo se emplea para procesadores de elevadas prestaciones.

Para su diseño se parte de la tabla de la verdad a la que responde el sistema. Las entradas de dicha tabla están formadas por los 6 bits del código Op (OP5-0) de la instrucción y los 4 bits (S´-0) que definen el estado actual dentro de los 10 posibles que existen en el grafo de estados del repertorio de instrucciones. Las salidas están formadas por los valores lógicos que toman las señales de control y por los 4 bits (S´3-0) que indican el estado siguiente que corresponde.

	ENTRADAS		**SALIDAS**				
	CÓDIGO OP	**ESTADO ACTUAL**				**ESTADO SIGTE.**	
ESTADO ACTUAL	**OP5 OP4 OP3 OP2 OP1 OP0**	**S3 S2 S1 S0**	**ALUop**	**ALUSelA**	**PCSource**	**S3´ S2' S1' S0'**	**ESTADO FIGURADO**
0	X X X X X X	0 0 0 0	0 0	0	0 0	0 0 0 1	1
1	CÓDIGO OP = lw y sw	0 0 0	0 0	0	- -	0 0 1 0	2
1	CÓDIGO OP = beq	0 0 0	0 0	0	- -	1 0 0 0	8
1	CÓDIGO OP = tipo R	0 0 0	0 0	0	- -	0 1 1 0	6
1	CÓDIGO OP = j	0 0 0	0 0	0	- -	1 0 0 1	9

Tabla 3.- Representación de una parte de la tabla de la verdad a que corresponde el grafo de estados del repertorio de instrucciones del MIPS multiciclo.

Si el número de instrucciones del repertorio y los modos de diseccionado son elevados, el grafo de estados se hace muy complejo y la tabla de la verdad admite miles de combinaciones. Debido a su elevado coste, en estos casos no es posible realizar un diseño cableado de la Unidad de Control.

Una vez obtenida la tabla de la verdad se pueden deducir las ecuaciones lógicas a las que responden las salidas en función de las entradas.

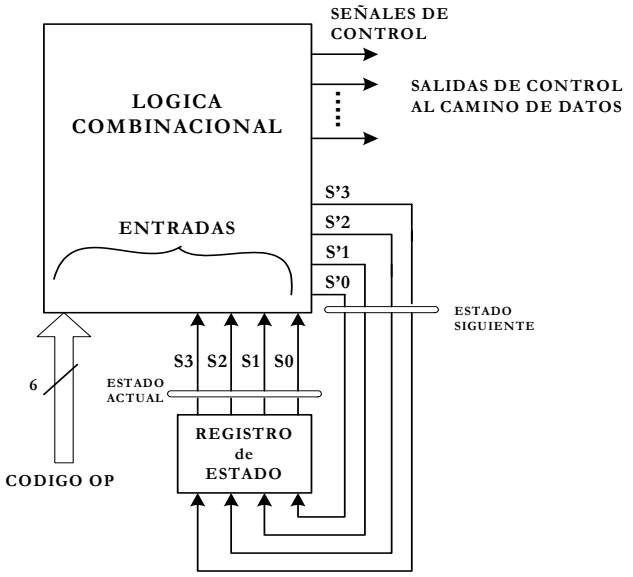


Figura 31.- Esquema general de la Unidad de Control cableada del MIPS multiciclo.

7.2- Unidad de Control Microprogramada

Cuando el número de instrucciones de repertorio y los modos de direccionado son elevados, se debe diseñar mediante **microprogramación**. Esta técnica es mucho más lenta que la Unidad de Control cableada pero tiene una gran ventaja ya que es barata y sencilla de implementar montones de instrucciones.

Está basada en la grabación de una memoria especial, denominada **memoria de control**. Cada posición de dicha memoria guarda el valor lógico de las señales de control que debe generarse para un estado determinado de una instrucción. Por lo tanto una instrucción está implementada en la memoria de control por un conjunto deposiciones que representan las diversas microinstrucciones o estados de la máquina. A este pequeño programa se le llama **microprograma**.

Según el secuenciamiento de las microinstrucciones existen dos tipos de Unidades de Control microprogramadas. Las de **Secuenciamiento Implícito**, en que todas las microinstrucciones de cada instrucción están grabadas en la memoria de control de forma secuencial y las de **Secuenciamiento Explícito**, en la que la dirección de la siguiente microinstrucción se especifica en uno de los campos de la microinstrucción anterior.

7.3- Formato de las microinstrucciones

En el formato de las microinstrucciones del MIPS multiciclo se pueden distinguir los siguientes campos:

CONTROL ALU: Especifica la operación de la ALU.
SCR1: Especifica la fuente para el primer operando de la ALU.
SCR2: Especifica la fuente para el segundo operando de la ALU.
DESTINO ALU: Indica el registro donde se guardará el resultado.
MEMORIA: Especifica lectura o escritura y la fuente de dirección.

REGISTRO MEMORIA: Especifica el registro que se carga con el dato leído en la memoria o que se escribe en ella.
CONTROL PCWRITE: Especifica como coger la siguiente instrucción que se va a ejecutar.

Existe otro campo llamado **Secuenciamiento,** que especifica la dirección donde se encuentra la siguiente microinstrucción a ejecutar. La Unidad de Control con Secuenciamiento Explícito tiene 3 posibilidades de determinar la siguiente microinstrucción a ejecutar:

1º. El campo secuenciamiento , toma el valor **seq** cuando la microinstrucción siguiente ocupa la dirección consecutiva actual.

2º. Cuando en el microprograma correspondiente a una instrucción se llega a la última microinstrucción, la siguiente es la primera de la siguiente instrucción, que siempre es una de tipo **fetch**. En el campo Secuenciamiento se indica con la opción **fetch**.

3º. Cuando la siguiente instrucción no es fija, sino que hay varias alternativas, según alguna entrada a la Unidad de Control, que generalmente suele ser el código OP. Para estos casos se hace una tabla para cada uno de los estados con esta posibilidad. En el caso de los MIPS esto pasa en el estado 1 y en el 2. Indicamos que la microinstrucción siguiente debería escogerse por una operación de distribución colocando **Dispatch i**, donde **i** es el número de la tabla de distribución en el campo Secuenciamiento ("Sequencing").

En la siguiente tabla se muestran los campos de las microinstrucciones del MIPS multiciclo, con los valores que admiten y la función que realizan.

NOMBRE DEL CAMPO	VALORES QUE ADMITE	FUNCIÓN QUE REALIZA
CONTROL ALU	Add Function case Sub	Suma la ALU (Transferencia) Depende del código de función (logic-arit) Resta (beq)
SRC1	PC Rs	1ª Entrada ALU PC 1ª Entrada ALU el rs
SRC2	4 Extend Extshift Rt	2ª Entrada ALU 4 I[15-0] 2ª Entrada ALU ext signo I[15-0] 2ª Entrada ALU ext signo y des.2 izq 2ª Entrada ALU rd
DESTINO ALU	Target (Destino) Rd	La salida de ALU se escribe en TARGET La salida de ALU se escribe en rd
MEMORIA	Read PC Read ALU Write ALU	Lee memoria con dirección PC Lee memoria con dirección ALU_{out} Escribe memoria con dirección ALU_{out}
REGISTRO MEMORIA	IR Write rt (lw) Read rt (sw)	El dato leido en mem se escribe en IR El dato leido en mem se escribe en rt El dato escrito en mem proviene de rt
CONTROL PCWRITE	ALU Target-Cond Jump Address	Escribe en PC Si Z= 1 PC TARGET Escribe PC con dirección bifurcación
SECUENCIAMIENTO	Seq Fetch Dispatch i	Elige secuencialmente la siguiente μinstrc. Va a 1ª μinstrc. De una nueva instrucción Distribuye utilizando la tabla i Dispatch

Tabla 4.- Tabla con los campos que componen las microinstrucciones del MIPS multiciclo y sus principales características.

En la tabla 5 se muestran los valores que toman los diversos campos correspondientes a las dos primeras microinstrucciones de cada instrucción. La primera microinstrucción para las 9 instrucciones del repertorio básico, que siempre son las mismas, es la **fetch**, que reailza la fase de búsqueda de instrucción, M(PC) = IR, y PC = PC + 4. La segunda, es la que realiza la fase de decodificación, por lo

que se denomina **decodif,** y además busca los dos operandos rs yrt y calcula la dirección de salto para las beq.

µInstrcción rotulo o etq.	Contol ALU.	SRC1	SRC2	Destino ALU	Memoria	Registro Memoria	Control PCWrite	Secuenciamiento
Fetch (1ª)	Add	PC	4		ReadPC	IR	ALU	Seq
Decodif.(2ª)	Add	PC	Extshift	Target				Dispatch 1

Tabla 5.- Tabla que muestra el valor que toman los campos de las dos primeras microinstrucciones , que son comunes a todas las instrucciones del MIPS multiciclo.

Para la primera microinstrucción los campos CONTROL ALU, SCR1 y SCR2 determinan la operación PC = PC + 4. Los campos relativos a memoria y registro-memoria, determinan la lectura de la memoria y la carga en IR el código de la instrucción M(PC) = IR, mientras que el campo PCWRITE permite que el PC se cargue con la salida de la ALU y el secuenciamiento será de tipo seq, ya que de la primera microinstrucción siempre se pasa a la segunda.

En cuanto a la segunda microinstrucción los campos que controlan la actuación de la ALU consiguen que se realice la operación PC + (EXTSG IR[15-0]x 4) = Target. Para el campo de Secuenciamiento hay que acudir a la Tabla 1 que determina según el código OP de la instrucción el siguiente estado o microinstrucción que se debe ejecutar.

En la tabla 6 se muestran las microinstrucciones correspondientes a las instrucciones de transferencia. La primera microinstrucción de la tabla, LWSW1, es igual para las dos instrucciones de transferencia lw y sw. Corresponde al estado 2 del grafo de estados del MIPS multiciclo. EN cuanto a LW2 y LW3 son las dos microinstrucciones que completan la instrucción lw, mientras que la microinstrucción SW" es la que completa la instrucción sw. La microinstrucción LWSW1 es la encargada de calcular la dirección de la memoria que hay que acceder. LW2 lee dicha posición y obtiene un dato para que en lw3 dicho dato se escribe en rt. En SW se escribe el valor de rt en la dirección calculada anteriormente.

µInstrcción rotulo o etq.	Contol ALU.	SRC1	SRC2	Destino ALU	Memoria	Registro Memoria	Control PCWrite	Secuencia-miento
LWSW	Add	rs	Extend					Dispatch 2
LW2	Add	rs	Extend		ReadALU			Seq
LW3	Add	rs	Extend		ReadALU	Write rt		Fetch
SW	Add	rs	Extend		WriteALU	Read rt		Fetch

Figura 6.- Tabla que recoge las microinstrucciones correspondientes a las instrucciones de transferencia lw y sw.

En la tabla 7, se proponen las dos microinstrucciones que finalizan la ejecución de las instrucciones lógico-aritméticas, de tipo R.

µInstrcción rotulo o etq.	Contol ALU	SRC1	SRC2	Destino ALU	Memoria	Registro Memoria	Control PCWrite	Secuencia-miento
Rformat 1	Fun code	rs	rt					Seq
	Fun code	rs	rt	Rd				Fetch

Tabla 7.- Tabla con los valores que toman los campos de las microinstrucciones que terminan la ejecución de las instrucciones lógico-aritméticas o de tipo R.

En la siguiente tabla se presentan los campos de todas las microinstrucciones que sirven para implementar el repertorio básico, que son 10 microinstrucciones, tantas como estados (0-9).

µInstrcción rotulo o etq.	Contol ALU	SRC1	SRC2	Destino ALU	Memoria	Registro Memoria	Control PCWrite	Secuencia-miento
Fetch	Add	PC	4		ReadPC	IR	ALU	Seq
	Add	PC	Extshf	Target				Dispatch 1
LSW1	Add	rs	Extend					Dispatch 2
LW2	Add	rs	Extend		ReadALU			Seq
	Add	rs	Extend		ReadALU	Write rt		Fetch
SW2	Add	rs	Extend		WriteALU	Read rt		Fetch
Rformat 1	Codefunc	rs	rt					Seq
	Codefunc	rs	rt	Rd				Fetch
BEQ 1	Sub	rs	rt				Target-cond	Fetch
JUMP 1							Jump adder.	Fetch

Tabla 8 - Representación de los valores que toman los diferentes campos de las 10 microinstrucciones básicas.

La siguiente figura muestra el esquema general de la Unidad de Control Microprogramaza del MIPS multiciclo. EL bloque principal es la Memoria de Control que estará formada por 10 posiciones con los bits precisos para contener todos los campos de las instrucciones. También existe una lógica auxiliar para configurar el Secuenciador , disponiendo como entradas los dos bits que eligen una de las tres alternativas del campo Secuenciamiento, y los 6 bits del código OP.

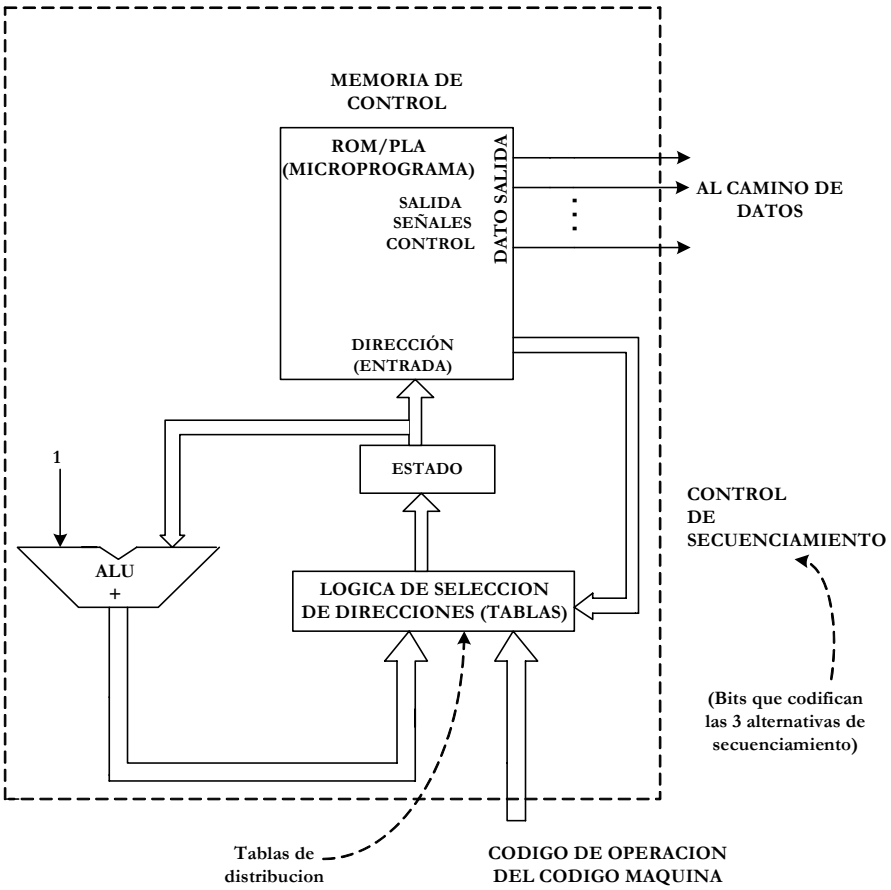


Figura 32.- Esquema general de la Unidad de Control microprogramada del MIPS multiciclo.

7.4- Excepciones e interrupciones

Se llama **excepción** a un acontecimiento inesperado o anómalo del procesador (por ejemplo: Desbordamiento, Uso del código no definido de instrucción, etc), que origina una ruptura automática del flujo de control.

Una interrupción es una excepción provocada desde el exterior al procesador. Para soportar las interrupciones el procesador dispone de patitas que detectan las peticiones externas cuando son activadas por el hardware externo.

Por lo tanto la Unidad de Control debe ser capaz de detectar las situaciones especiales que causan las excepciones y producir una bifurcación del control a la rutina que resuelve la anomalía. Después de resolverla, hay que regresar al punto de partida del programa principal.

La unidad de control del MIPS es capaz de detectar dos excepciones:

1ª : Código OP de instrucción no definido.

2ª : Desbordamiento de una operación de la ALU.

Cuando hay varias causas que provocan excepciones, existen también diferentes maneras de atenderlas. A dichos procedimientos se accede de dos maneras:

a) Mediante interrupciones vectorizadas (80x86)

Se trata de hacer corresponder a cada excepción una dirección concreta de la memoria donde debe comenzar la rutina que la atienda.

b) Mediante un registro de estado cuyos bits especifican el tipo de excepción (MIPS)

En el MIPS hay un registro, llamado **CAUSE**, cuyo último bit determina una de las dos posibles causas de excepción. Si vale 1, significa que se trata de un código OP no definido y si vale 0, de un desbordamiento. Figura 33.

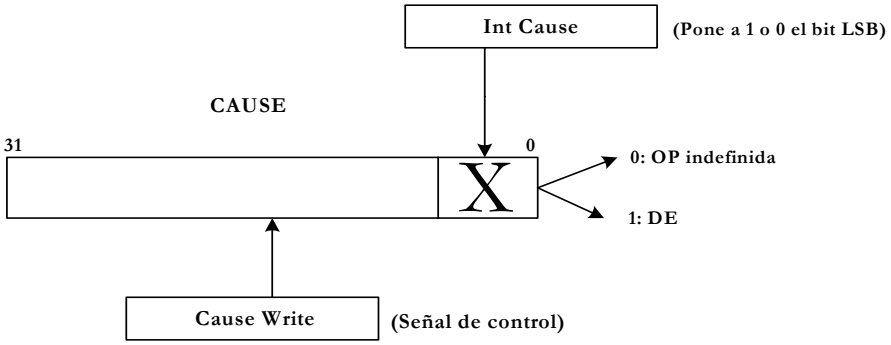


Figura 33.- El último bit del registro CAUSE determina el tipo de excepción que ha sucedido en el MIPS.

Una vez atendida la excepción se debe retornar al programa principal, para ello se necesita un registro auxiliar para guardar el valor de PC, este registro es el EPC. Figura 34.

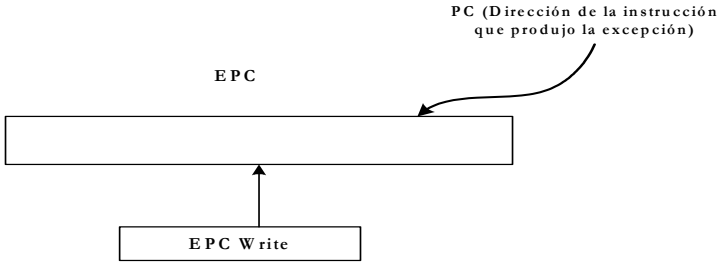


Figura 34.- Registro EPC del MIPS.

En el MIPS, cada vez que se produce una excepción, se salta a una rutina común que comienza en la dirección que se carga en el PC cuyo valor **es 0100 0000 0000 0000 0000 0000 0000 0000**. Luego la propia rutina leyendo el bit de menos peso de CAUSE determina cual de las dos posibles excepciones ha sucedido y atiende en consecuencia.

El multiplexor de 4 entradas que cargaba el PC, tenía una entrada libre que se emplea para dejar pasar con la señal de control PCSource = 11, el valor del PC indicado en el párrafo anterior que indica la rutina de atención a la excepción.

Además el registro EPC se debe cargar con el valor del PC donde ha sucedido la excepción y no con el del PC + 4 obtenido del PC desde la fase de búsqueda de la instrucción. Para conseguir dicho valor la ALU debe restar 4 al valor de PC + 4. El resultado obtenido se carga en el registro EPC.

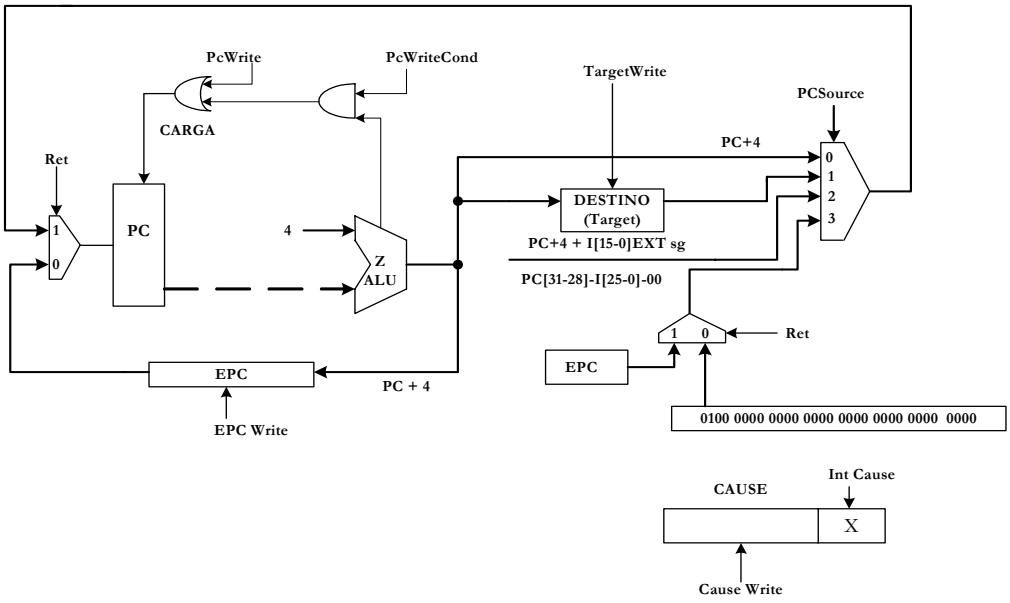


Figura 35.- Esquema de los elementos complementarios que hay que añadir al MIPS multiciclo para que sea capaz de soportar el tratamiento de las excepciones.

En la figura 36 se muestra el grafo de estados del MIPS multiciclo, con capacidad para el tratamiento de las dos excepciones, la del código OP no definido y la del desbordamiento. La primera se detecta después del estado 1, cuando se comprueba que el código OP no es ninguno de los admitidos. En cuanto a la segunda, ésta se detecta una vez acabado el estado 7, ultimo de las instrucciones lógicas y aritméticas, que es cuando se reconoce si se ha producido desbordamiento.

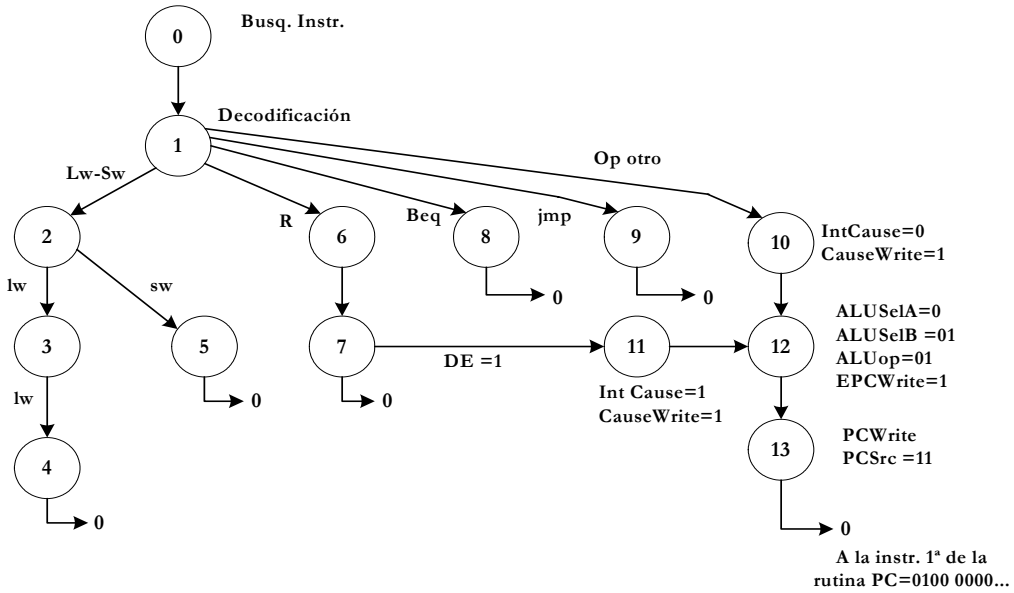


Figura 36.-Grafo de estados del MIPS multiciclo con posibilidad de tratar tanto la excepción de código OP no definido y la de desbordamiento de la ALU.

8- EL MIPS SEGMENTADO

En la actualidad la mayoría de los procesadores alcanzan rendimientos espectaculares debido a tres aspectos fundamentales:

1º. Funcionan a una frecuencia superior a los 100 Mhz.

2º. Disponen millones de transistores, que les permiten implementar:

a) Una potente y sofisticada arquitectura dotada de todos los recursos de los grandes procesadores.

b) Una memoria caché de gran capacidad y varios niveles jerarquicos.

3º. Se integran en el mismo procesador las dos técnicas más representativas y eficaces para elevar la potencia de procesamiento:

a)SUPERSEGMENTACIÓN

Se consigue un número muy elevado en las etapas de segmentación, así por ejemplo el Pentium Pro alcanza un nivel 16.

b)SUPERESCALAR

Disponen de más de un cauce segmentado y son capaces de iniciar más de una instrucción en cada ciclo de reloj.

La segmentación fue planteada por primera vez al comienzo de la década de los años 60 en el procesador IBM 703. Hitos importantes de esta técnica fueron los procesadores CDC 6600 en 1964 y el IBM 360/91, en 1966.

Los compiladores deben ser capaces de evitar introducir al mismo tiempo a los cauces de los procesadores superescalares pares o conjuntos de instrucciones que sean dependientes entre sí.

Los compiladores deben deshacer las dependencias que surgen entre instrucciones consecutivas de un programa, cuando se introducen a un cauce segmentado. Si se trata de una instrucción de bifurcación condicional (beq), la siguiente instrucción deberá introducirse al cauce cuando se haya resuelto la condición de bifurcación y la instrucción de salto se haya completado.

8.1- La técnica de la segmentación

Mediante la segmentación se pueden ejecutar varias instrucciones a la vez en un mismo procesador multiciclo, con lo que se pretende que cada instrucción utilice en cada ciclo un recurso diferente, consiguiendo un cierto paralelismo, que recibe el nombre de **paralelismo implícito**.

Todos los procesadores modernos utilizan la segmentación de forma exclusiva.

Una línea o cauce bien equilibrado es el que tiene sus etapas de la misma duración, por lo que cada ciclo o tiempo empleado en la operación de cada célula, una instrucción pasa a la célula siguiente y de la última sale una acabada. Estas células reciben el nombre de **etapas de segmentación** o **etapas** en el procesador.

Con la segmentación no se reduce el tiempo, pero se mejora la productividad.

Si las etapas están perfectamente equilibradas, es decir, duran lo mismo:

$$\text{Tiempo tarda en salir Instrucción Segmentada (una vez este el cauce lleno)} = \frac{\text{Tiempo instrucción sin segmentar}}{\text{NUMERO DE ETAPAS}}$$

Por lo tanto la mejora de velocidad en la terminación de instrucciones es igual al número de etapas, si están bien equilibradas. En caso contrario, existe una pérdida de rendimiento al existir perdida de rendimiento en algunas.

En la práctica, las etapas nunca llegan a estar bien equilibradas.

EJEMPLO:

Analicemos el siguiente supuesto en el que la tecnología de fabricación utilizada en la construcción de MIPS proporciona los siguientes retardos significativos:

T Acceso a memoria = 80 ns

T Acceso a banco = 50 ns

T Retardo ALU = 60 ns

Comparemos el aumento de la velocidad en la ejecución de instrucciones del MIPS monociclo y del segmentado.

A continuación calculamos los tiempos que dura cada instrucción en el monociclo:

INSTRUCCIÓN	**T_{acc} M_{ins}**	**T_{acc} Banco Lectura**	**T ALU**	**T_{acc} M_{datos}**	**T_{acc} Banco Escritura**	**TOTAL**
Lw	80	50	60	80	50	320 ns
Sw	80	50	60	80		270 ns
R	80	50	60		50	240 ns
Beq	80	50	60			190 ns

Tabla 9.- Cálculo de la duración de las instrucciones en el MIPS monociclo.

El ciclo que se utiliza en este tipo de procesador el da la instrucción más lenta. Por lo tanto, el ciclo será de 320 ns.

En el MIPS segmentado, a cada etapa se le da la duración de la que más dura, que en nuestro caso será la de acceso a la memoria que son 80 ns. De este modo, las cinco etapas en las que se descompone la instrucción duran 80 ns cada una.

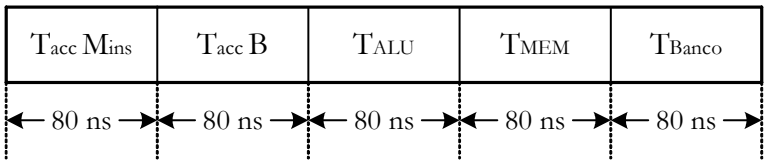


Figura 37.- Duración de los ciclos de una instrucción.

Una vez que se llena el cauce de las 5 etapas, transcurridos 80 ns sale de él una instrucción terminada. En algunas etapas, existe una pérdida de rendimiento, debido a que algunas se podían haber realizado en menos tiempo.

Por lo tanto, el aumento de velocidad entre un MIPS monociclo y uno segmentado es de: 320 / 80 = 4. Es decir el resultado sería 4, aunque el teórico sería 5, debido a que hay 5 cauces, esto es debido al desequilibrio de tiempos entre las 5 etapas.

8.2- MIPS con camino de datos segmentado

Para determinar las etapas del cauce segmentado se parte del MIPS monociclo, y de los pasos en los que se descomponen las instrucciones. Obteniéndose de este modo 5 etapas:

1ª: IF : Búsqueda de la instrucción.

2ª: ID : Decodificación y Búsqueda de operandos.

3ª: EX : Ejecución y Cálculo Dirección Efectiva.

4ª: MEM : Acceso a Memoria de Datos.

5ª: WB : Postescritura.

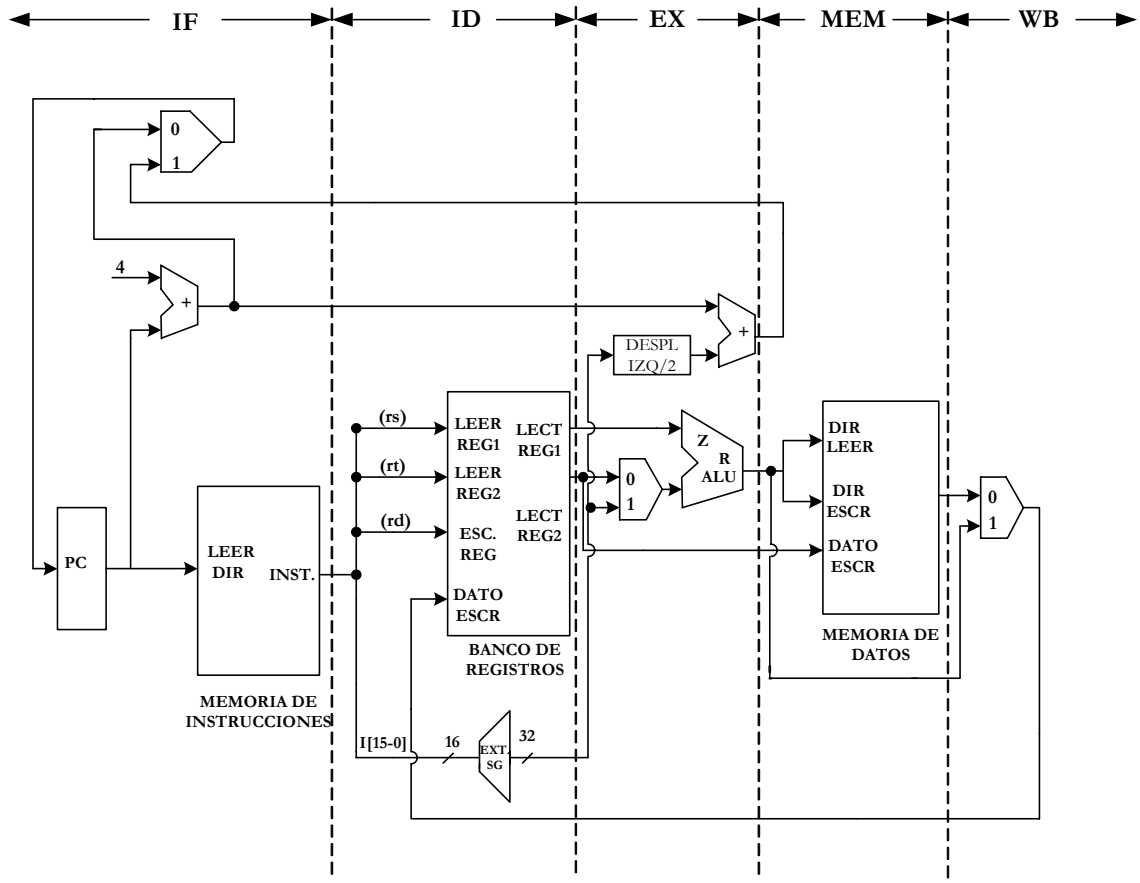


Figura 38.- Cauce sementado del Camino de Datos del MIPS

El flujo de datos e instrucciones va recorriendo las 5 etapas de izquierda a derecha, aunque hay dos excepciones:

- En la etapa MEM existe una realimentación a la etapa anterior IF con el valor de salto que se carga en PC para la instrucción beq.
- En la etapa de Postescritura, el resultado que genera se escribe en la etapa anterior ID, en el Banco de Registros.

Para poder compartir un mismo recurso por varias instrucciones se colocan registros intermedios entre las etapas, que almacenan el resultado que produce cada una.

8.3- Representación gráfica de la segmentación

Mediante este gráfico se pretende mostrar la ejecución de varias instrucciones, como si cada una tuviese su propio camino.

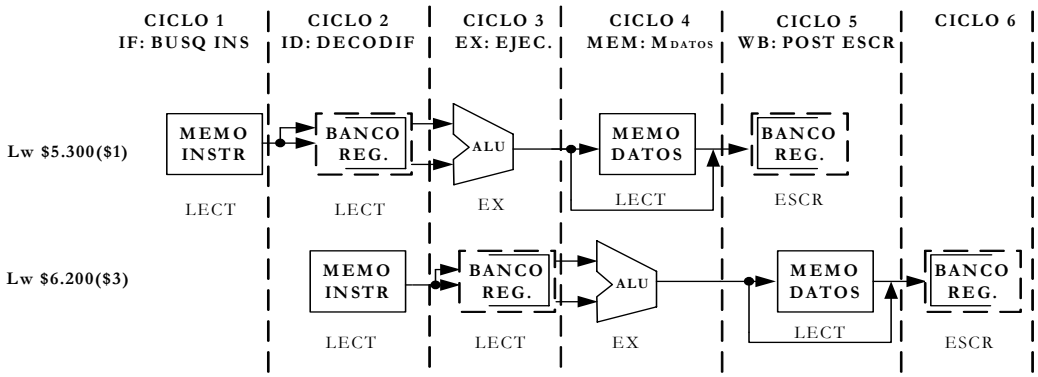


Figura 39.- Representación gráfica de la segmentación que muestra la ejecución de varias instrucciones.

En cada ciclo de reloj sólo se utiliza un recurso. El **Banco de Registros** se descompone en dos módulos independientes, uno se encargará de la lectura de una pareja de registros y el otro de la escritura del que se especifica por la entrada correspondiente. Para su correcta representación, se marca con línea gruesa la mitad derecha del banco de registros cuando se lee (la izquierda a trazos), y cuando se escribe se intercambia la representación de las dos mitades.

La memoria de instrucciones solo se usa en la fase IF después queda libre pudiendo ser accedida por otras instrucciones, pero para ello es necesario almacenar en in registro (RI) el código leído.

Las instrucciones avanzan en su ejecución de una etapa a la siguiente en cada ciclo, pasando la información que les corresponde desde un registro de segmentación al siguiente.

Los registros deben de ser lo suficientemente grandes para almacenar toda la información que genera cada etapa y que hay que pasar a la siguiente.

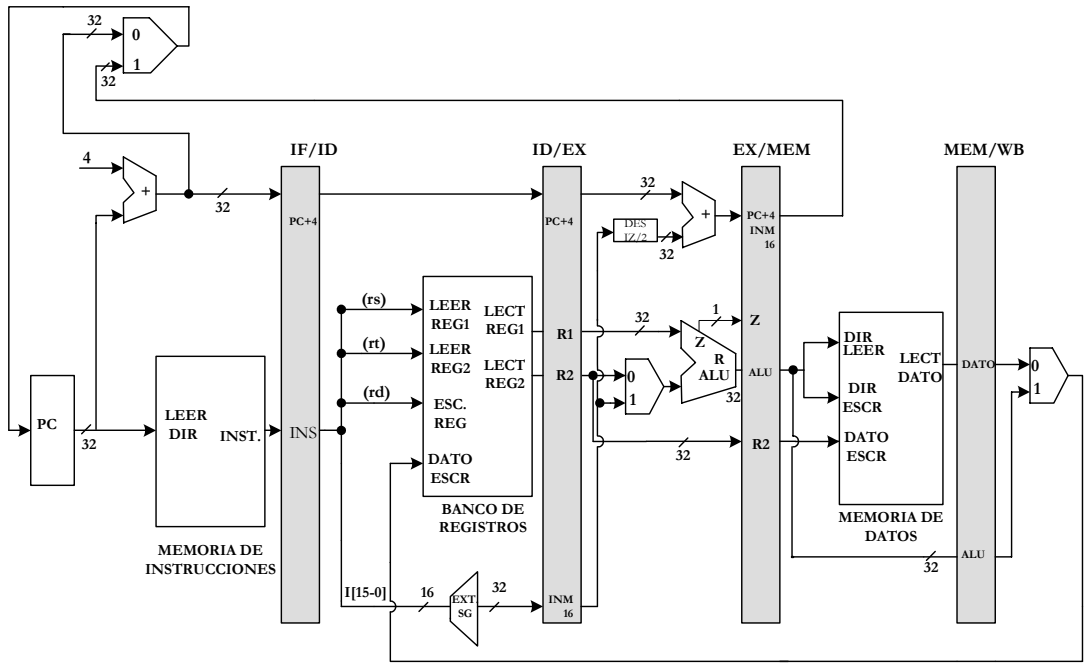


Figura 40.- Los registros de segmentación existentes entre etapas, sirven para almacenar información correspondiente a la instrucción que utiliza esa etapa, proporcionándosela a la siguiente.

Para saber que información debe almacenarse, inicialmente se hace el estudio de la segmentación referida a la arquitectura del procesador monociclo

- Primera etapa **IF**

  Operaciones elementales de esta etapa:

  M(PC)= INS ; PC + 4= PC

  En el registro IF/ID se guardan 64 bits, 32 del PC y los restantes 32 del código INS.

- Segunda etapa **ID**

  En el MIPS monociclo, se lleva a cabo la **decodificación**, la lectura automática de los registros R1 y R2 y la extensión de signo de valor INMEDIATO.

  En ID/EX se guardan 128 bits : (PC +4), (R1), (R2), y (INM32).

- Tercera etapa **EX**

  La ALU realiza la suma de R1 (rs) con INM32 para calcular la dirección de acceso a la **memoria de datos**.

  EN el registro EX/MEM se guardan 97 bits: ALU_{out}, Z, [PC + 4 + INM32] y R2.

- Cuarta etapa **MEM**

  Se lee la memoria de datos con la dirección que sale por la ALU.

  En el registro MEM/WB se guardan 64 bits: DATO32 y ALU_{out}.

- Quinta etapa **WB**

  El dato leído en la memoria de datos se escribe en el registro R3 (rt). En esta última etapa no hay registro de segmentación, puesto que en esta etapa se escribe en el Banco, bien el dato de la memoria o bien el ALU_{out}.

En esta última etapa surge un problema: hay que escribir el dato en R3 y en consecuencia , se debería haber guardado el valor de dicho registro que venía en el código de INS y se obtenía desde la etapa ID, a través de las distintas etapas hasta la WB.

Para resolver este problema, el valor de R3 (rt), que se obtiene a la salida de ID, no se aplica al Banco de Registros, sino que se guarda en el registro ID/EX y se va pasando hasta la etapa WB, donde se usa para especificar donde hay que escribir. Este procedimiento también se puede realizar en todas las instrucciones aritmético-lógicas.

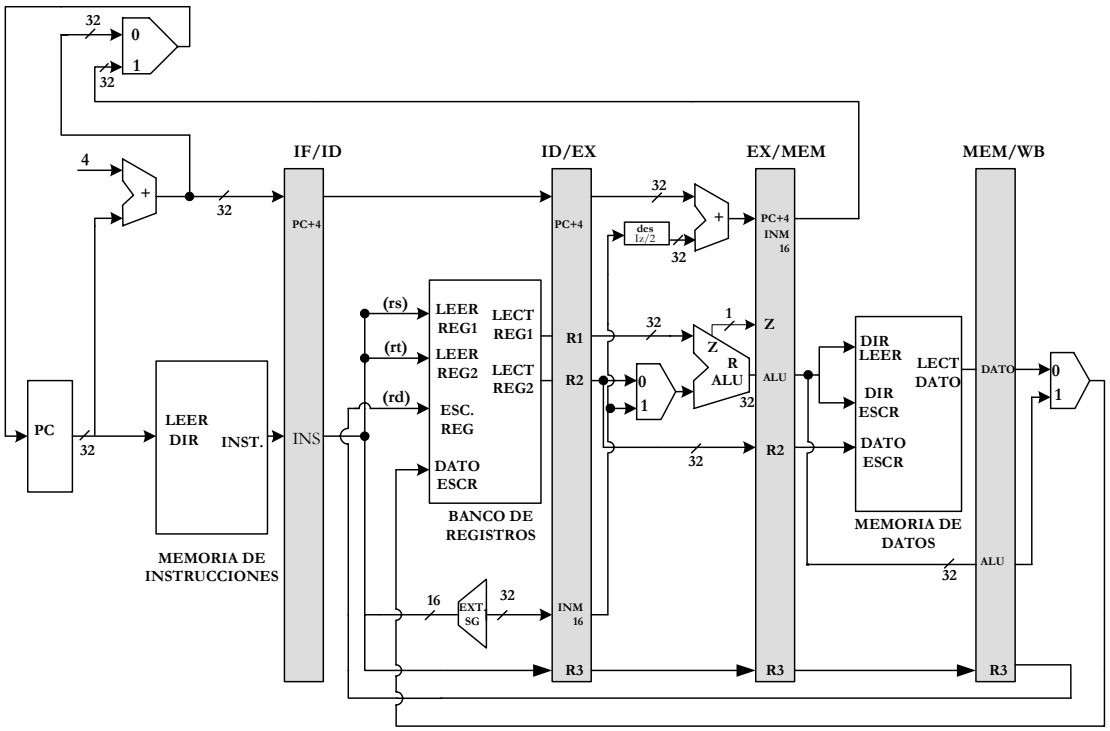


Figura 41.- El valor del registro donde hay que escribir, se va traspasando registro a registro, hasta la etapa WB.

En la instrucción de almacenamiento (sw), en la etapa EX se usa el registro R2 para pasarlo a la etapa MEM para ser escrito en la dirección que se obtiene por ALUout. Después de escribir la memoria de datos no pasa nada más.

8.4- Unidad de control segmentada

Se trata de definir el valor lógico de las líneas que controlan los recursos del Camino de Datos segmentado y calcular sus valores para cada ciclo de reloj de cada instrucción.

Las señales de control de cada etapa son:

- Etapa **IF :** Las señales de control para leer la memoria de instrucciones y para escribir el PC están activadas siempre, así que no hay control especial en esta etapa.
- Etapa **ID :** Ocurre lo mismo que en la etapa anterior, por lo que no hay líneas de control opcionales que inicializar.

- Etapa **EX :** Las señales que se van a inicializar son **RegDst**, **ALUop** y **ALUSrc**. Las señales seleccionan el registro Result, la operación de la ALU, y/o un registro o un inmediato con signo extendido para la ALU.
- Etapa **MEM :** Las líneas de control que se inicializan son **Branch**, **MemRead** y **MemWrite**. Estas señales las inicializan las instrucciones saltar sobre igual, cargar y almacenar, respectivamente.
- Etapa **WB :** Las dos líneas de control son **MemtoReg**, que decide entre enviar el resultado de la ALU o el valor de memoria a los registros, y **RegWrite**, que escribe el valor escgido.

En la siguiente tabla se muestran las diversas señales de control que regulan el comportamiento del Camino de Datos.

Instrucción	ETAPA EX				ETAPA MEM			ETAPA WB	
	Reg Dest	ALU op1	op op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem to Reg
TIPO R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Tabla 10.- Tabla que muestra las señales de control que regulan el funcionamiento del Camino de Datos.

A continuación se muestra el Camino de Datos segmentado con sus correspondientes señales de control.

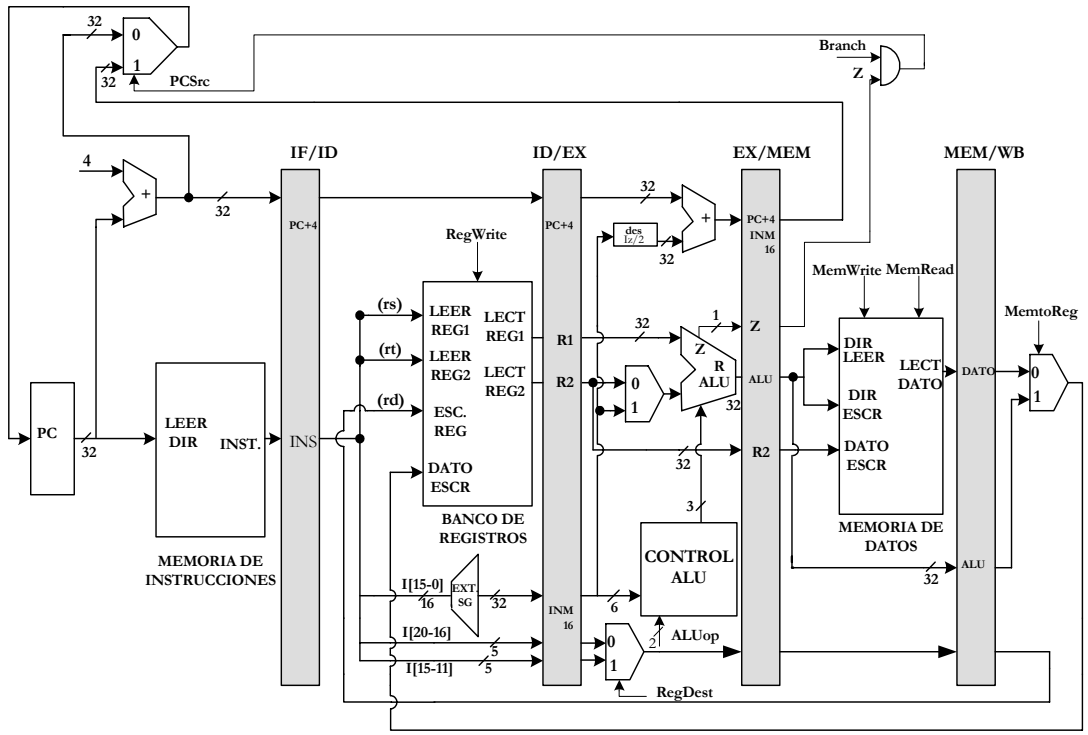


Figura 42.- Esquema del Camino de Datos segmentado con sus señales de control.

Existen 9 señales de control, los cuales sólo se emplean en las tres últimas etapas de segmentación. Tras la segunda etapa ID se calculan para cada instrucción el valor de las 9 señales de control y se van traspasando por los registros interetapas , hasta alcanzar cada señal de control la etapa donde debe actuar.

8.5- Los conflictos en la segmentación y su incidencia en el rendimiento del procesador

Se denominan **conflictos** y también **riesgos** ("hazards") a las causas o situaciones que impiden introducir instrucciones al cauce segmentado. Se puede interrumpir la entrada de interrupciones al cauce durante uno o varios ciclos de reloj, ciclos vacíos en contenido que reciben el nombre de **burbujas** o **ciclos de detección**.

Por ello, al detenerse la entrada de instrucciones en el cauce se disminuye notoriamente el rendimiento del procesador.

Debido a los conflictos, el CPI$_{IDEAL}$ con el que se parte teóricamente se convierten el CPI$_{IRREAL}$, de mayor valor que el primero al incrementarse a causa de los ciclos de detención del cauce. Este CPI$_{IDEAL}$ se obtiene al dividir el CPI sin segmentación por el número de etapas de segmentación, que se denomina **profundidad**.

$$\mathbf{CPI}_{IDEAL} = \frac{\text{CPI SIN SEGMENTACIÓN}}{\text{PROFUNDIDAD}}$$

CPI$_{SS}$ = CPI$_{IDEAL}$ x PROFUNDIDAD

De donde la aceleración con segmentación,

ACELERACIÓN$_{segm}$= Tiempo de una instrucción sin segm / Tiempo de una instrucción con segm

CIP$_{SS}$ x Ciclo reloj$_{ss}$ = CPI$_{IDEAL}$ x PROFUNDIDAD x CICLOS$_{SS}$
CIP$_{CS}$ x Ciclo reloj$_{cs}$ = CPICS x CICLOS$_{CS}$

Al tenerse en cuenta los ciclos de reloj de detención que se producen por promedio por cada instrucción a causa de los conflictos, la aceleración con segmentación queda como:

CPI$_{CS}$ = CPI$_{IDEAL}$ + Ciclos$_{Detención\ instrucción}$

ACELERACIÓN$_{segm}$= Ciclo reloj$_{ss}$ x CPI$_{IDEAL}$ x PROFUNDIDAD / Ciclo reloj$_{cs}$ x CPI$_{IDEAL}$ + Ciclos Detención

Debido a los gastos de la segmentación el ciclo de reloj con segmentación llega a ser muy parecido al ciclo sin segmentación, por lo que el primer término de la última fórmula se supone igual a la unidad, quedando, la expresión simplificada como:

ACELERACIÓN$_{segm}$= CPI$_{IDEAL}$ x PROFUNDIDAD / CPI$_{IDEAL}$ + Ciclos Detención

8.5.1- Tipos de conflictos

Se pueden producir cuatro tipos de conflictos según las causas que los provocan:

1º.Conflictos estructurales: Se producen cuando el cauce segmentado soporta varias instrucciones a la vez y algunas de ellas necesitan utilizar a la vez un mismo recurso del Camino de Datos. También se les suele denominar **colisiones**.

Una colisión se produce, por ejemplo, cuando en el mismo ciclo de reloj una etapa de una instrucción tiene que acceder a la memoria para buscar la instrucción y otra al mismo tiempo debe acceder a la memoria para leer o escribir un dato.

Este conflicto se resuelve dotando al procesador de dos memorias tipo caché, independientes, una para datos y otra para instrucciones.

2º.Conflictos por dependencia de datos: Se producen cuando una instrucción utiliza como operando el resultado de una instrucción previa que se está ejecutando en el cauce y todavía no ha producido el valor del resultado:

```
add  $7, $1, $2
or   $3, $7, $4
```

En este caso cuando la instrucción se encuentra en la segunda etapa del cauce (ID), debe proceder a leer el valor de uno de los operandos que es el registro $7, pero como la instrucción anterior se encuentra en la etapa de ejecución todavía no se actualizado su valor, por ello la instrucción or deberá esperar tres ciclos, que den tiempo a la instrucción add a escribir el resultado en el registro $7.

3º.Conflictos por saltos condicionales: Hasta que no se complete la instrucción condicional (beq), no puede introducirse otra instrucción en el cauce, ya que no se sabe cual será la siguiente.

Para solucionar este tipo de conflictos, se emplean métodos de predicción de salto.

4º.Conflictos por excepciones e interrupciones: Cuando se produce una interrupción o una excepción durante la ejecución de una instrucción de un programa se genera un salto a una rutina de atención, debiéndose eliminar las instrucciones del programa que se hayan introducido al cauce después de la que origina la excepción o interrupción.

8.6- Conflictos estructurales

Para poder introducir al cauce cualquier combinación posible de instrucciones hay que duplicar los recursos, ya que sino dos instrucciones pueden requerir un mismo recurso, produciéndose así un conflicto.

Una solución para los conflictos estructurales sería duplicar la memoria para poder realizar así accesos simultáneos a datos e instrucciones.

En nuestro caso si solo existiese una memoria para datos e instrucciones, cada vez que se introdujese en el cauce una instrucción de transferencia, habría que impedir la entrada de una nueva instrucción en el cauce en el ciclo en el que las dos accediesen simultáneamente a memoria a leer instrucciones o a leer o escribir datos. Por lo que habría que añadir un ciclo de reloj para que se pudiesen realizar correctamente estas instrucciones.

Otro recurso a tener en cuenta sería el Banco de Registros , que solo tiene un puerto de escritura y que podría dar lugar a un conflicto si sucediese que dos instrucciones del cauce tendrían que escribir en el mismo ciclo en el banco. Para evitar esta posibilidad es necesario seleccionar cuidadosamente el repertorio de instrucciones y su división en etapas.

Como la duplicación aumenta el coste, se buscan otras soluciones, como la segmentación o división en etapas de recursos, como es el caso de la ALU. Se pueden tomar dos medidas, una añadir dos sumadores, lo que aumentaría el coste u otra el segmentar o dividir en etapas la ALU con lo que se evitaría el aumento de coste, pero aumentaría la profundidad, lo que puede provocar una disminución del rendimiento.

8.7- Conflictos por dependencia de datos

Se produce cuando la segmentación cambia el orden de acceso a los operandos, con relación al que sigue a la secuencia normal de instrucciones. Ejemplo:

```
add $7, $1, $2
or $3, $7, $4
and $5, $6, $7
add $8, $7, $7
lw $9, 8($7)
```

Todas las instrucciones que siguen a la primera del programa utilizan como operando el contenido del registro $7, que es el resultado de la primera add. Por lo tanto hasta que no se termine la primera instrucción, el valor de dicho registro no se actualizará y las instrucciones que lean este valor y que se realicen antes de que se actualice dicho registro leerán un valor erróneo.

En la siguiente figura mostramos un ejemplo:

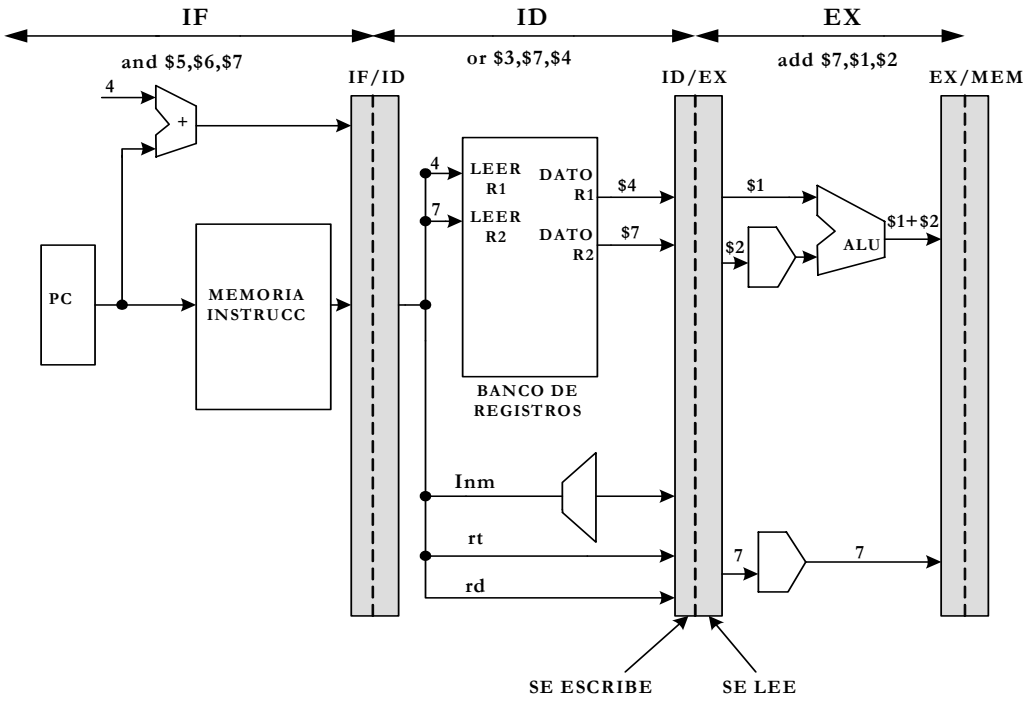


Figura 43.- La instrucción or se halla en la etapa ID y procede a leer sus dos operandos que son el registro $7 y $4. La instrucción anterior se halla en la etapa EX y debe actualizar el valor de $7, pero aún debe pasar por las etapas MEM y WB para hacerlo.

Del estudio del esquema de la figura 43, se deduce que va a producirse un conflicto por dependencia de datos en el cauce, ya que mientras se está generando la suma $1+$2, a la vez que se está determinando que el registro destino rd, será el $7, que se halla aún sin actualizar.

Este conflicto se representa de la siguiente manera:

ID/EX.Escribir Registro = IF/ID.Leer Registro1=$7

A continuación se analizan los distintos conflictos que surgen con las instrucciones del ejemplo:

1.-Conflicto add-or

Se detecta cuando add está en la etapa EX y la or en la ID donde el registro causante del conflicto es el $7, que tiene que ser escrito por add y leido por or.

ID/EX.Escribir Registro=IF/ID.Leer Registro1 = $7

2.-Conflicto add-and

Se detecta cuando add está en la etapa MEM y la and en ID.

EX/MEM.Escribir Registro = IF/ID.Leer Registro2 = $7

3.-Conflicto add-add (primero)

La primera add está en la etapa WB, y la segunda se encuentra en la etapa ID.

MEM/WB.Escribir Registro = IF/ID.Leer Registro1 = $7

4.-Conflicto add-add (segundo)

El conflicto se debe a que la segunda utiliza el registro $7 para sus dos operandos.

MEM/WB.escribir Registro = IF/ID.Leer Registro2 = $7

No existe conflicto entre la primera instrucción y la última, ya que para cuando entra en el cauce la última instrucción la primera ya ha acabado.

8.7.1- Detección y eliminación de las dependencias de datos

Para eliminar estos conflictos distinguimos dos alternativas. Una emplea un hardware auxiliar del procesador, mientras que la otra lo realiza mediante software, concretamente mediante el compilador.

La solución hardware consiste en diseñar una lógica combinacional capaz de detectar los conflictos de dependencias de datos, a esta lógica se le llama Unidad de detección de conflictos.

Un conflicto de dependencia de datos ocurre cuando se cumplen estas condiciones:

1. La señal Reg. Write = 1, en las etapas EX, MEM ó WB. Esto significa que una instrucción previa del cauce tiene que escribir un registro y aún no lo ha hecho.
2. Que el registro o registros que hay que leer en la etapa ID coinciden con el que hay que escribir en cualquiera de las etapas siguientes.

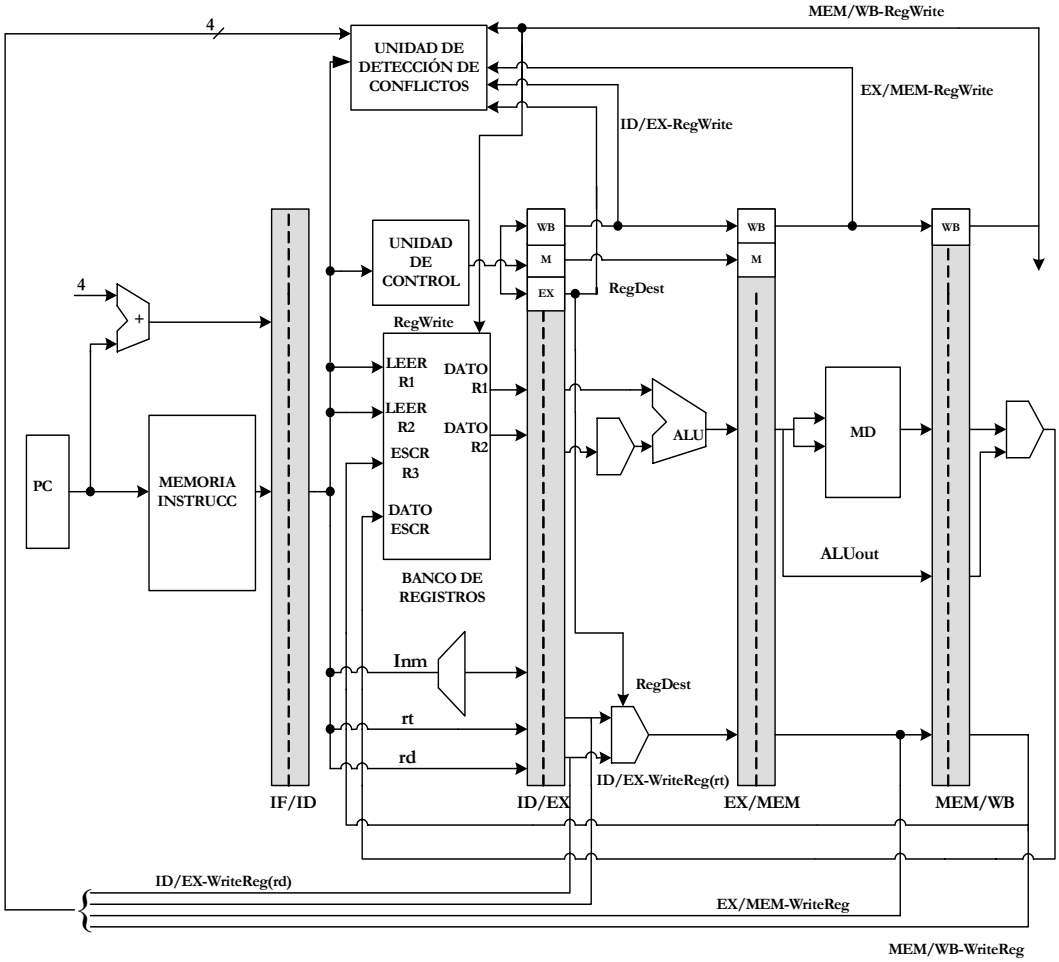


Figura 44.- Representación de las líneas principales que determinan cuando se debe escribir un registro y su selección mediante la señal de control Reg.Dst.

La **Unidad de detección de conflictos** de la figura 44, considera las señales de entrada:

- Señales Reg.Write de las etapas EX, MEM y WB.
- Señal Reg.Dst de la etapa EX.
- Señal Esch.Reg de las etapas EX, MEM y WB.
- Señales de lectura de registros R1 y R2 etapa ID.

La Unidad de Detección de Conflictos produce una salida **S = 1**, si es que se ha producido un conflicto por dependencia de datos, si no es así, tomara el valor **S = 0**.

Si se produce conflicto, se debe detener el traspaso de instrucciones de las etapas IF e ID, hasta que se solucione el conflicto. Esto se puede realizar mediante un compilador introduciendo instrucciones NOP en el cauce.

Por hardware también se pude conseguir, si la Unidad de Control genera como valor de sus nueve señales de control un "cero"(desconexión). De este modo no se modifica el contenido de PC ni de IF/ID.

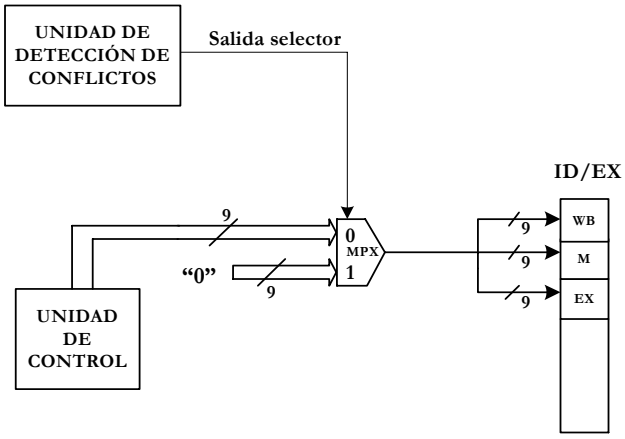


Figura 45.- Cuando la Unidad de Detección de Conflictos detecta uno y genera una salida de nivel alto, las nueve señales de control se transforman en ceros mediante el multiplexor.

Tanto por software como por hardware se consigue evitar el conflicto introduciendo burbujas en el cauce, lo que degrada el rendimiento del procesador.

8.7.2- Técnica de anticipación para reducir los conflictos por dependencias de datos en registros

También se le denomina **desvío** o "bypassing". Consiste en almacenar temporalmente los resultados de la ALU en un par de registros e implementar una lógica de desvío que examine si el resultado producido por la ALU y que más tarde se escribirá en un registro, se usa como operando en alguna de las instrucciones posteriores, si es así se introduce el resultado de la ALU, mediante un multiplexor a una de las entradas de la ALU, evitando la introducción de burbujas. Figura 46.

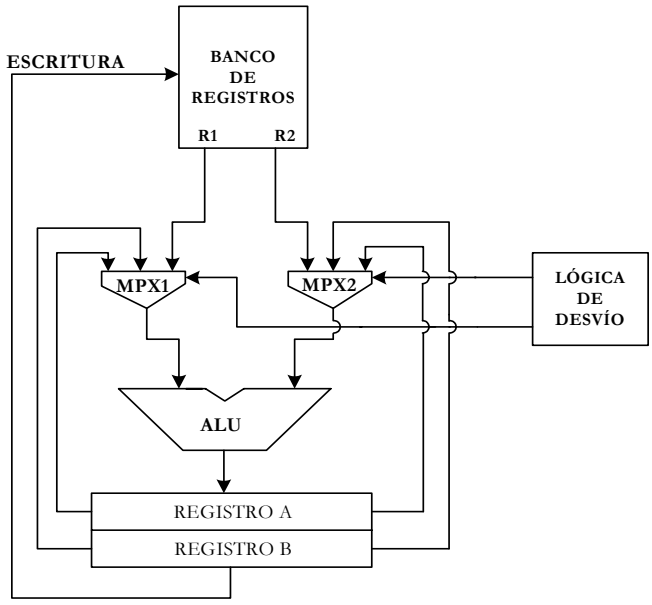


Figura 46.- Funcionamiento de la técnica de anticipación.

Para implementar la técnica de la anticipación en el MIPS se realizan las siguientes hipótesis:

Los resultados que hay que cargar en los registros afectados, se encuentran en los registros interetapa EX/MEM y MEM/WB.

- Para poder introducir en la ALU los resultados obtenidos en ella, se utilizan multiplexores que realimentan la salida con las entradas.
- Al control de estos multiplexores lo efectúa una **lógica auxiliar de desvío,** que analiza si los resultados van a ser leídos por instrucciones posteriores en el cauce.

La lógica de desvío compara los registros que actúan como operandos en la etapa ID, con los registros que actúan como destino. Si hay coincidencia, se introduce como entrada de la ALU, mediante multiplexores de realimentación, el valor de la salida de la ALU procedente del registro EX/MEM o del MEM/WB. Figura 47.

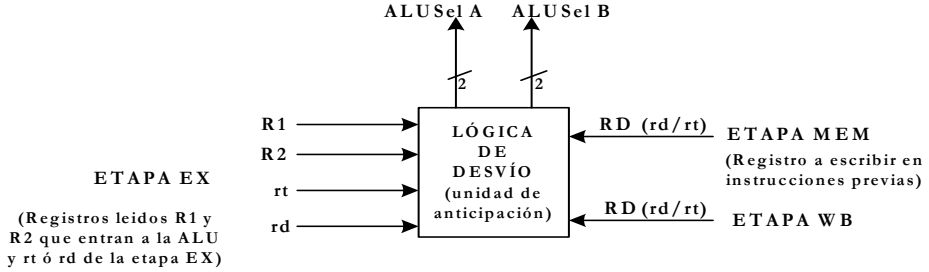


Figura 47.- La lógica de desvío analiza los registros R1 y R2 que se introducen como operandos a la ALU, con los valores de los posibles registros donde hay que escribir en las etapas MEM y WB.

Se pueden producir dos combinaciones de conflicto:

Si los registros que se introducen como operandos a la ALU en la etapa EX coinciden con alguno que hay que escribir en la etapa MEM

Conflicto EX

- Si EX/MEM.Reg Write

  y EX/MEM.Escribir Registro = ID/EX.Leer Registro1 ALUSelA=01
- Si EX/MEM.Reg Write

  y EX/MEM.Escribir Registro = ID/EX.Leer Registro2 ALUSelB=01

Conflicto MEM

- Si MEM/WB.Reg Write

  y MEM/WB.Escribir Registro = ID/EX.Leer Registro1 ALUSelA=01
- Si MEM/WB.Reg Write

  y MEM/WB.Escribir Registro = ID/EX.Leer Registro2 ALUSelB=01

No se considera el tercer conflicto WB, porque se supone que en la etapa ID se lee el resultado correcto si en dicho ciclo la instrucción en la etapa WB escribe alguno de los registros.

8.7.3- Técnica de anticipación para evitar conflictos por dependencias de datos en accesos a memoria

Estudiemos la siguiente secuencia de instrucciones:

lw $7, 50($2)

add $3, $7, $1

Durante el mismo ciclo de reloj add lee el valor del registro $7 que actúa como operando de la ALU, y la instrucción lw está accediendo a Memoria de Datos para leer el valor que tiene que cargar en el registro $7.

Como el dato que se va a cargar mediante la instrucción lw no va a estar disponible cuando lo necesita la add, la única solución para este conflicto es detener la segmentación. Por lo tanto habrá que introducir una burbuja para dar tiempo a disponer del dato que sale de la memoria.

Siempre que exista una instrucción lw seguida de otra que utilice como operando el registro que carga la lw, será preciso introducir una burbuja en el cauce.

Para detectar este tipo de conflicto se usa la Unidad de Detección de Conflictos, que deberá introducir nueve ceros como señales de control cuando detecte este conflicto, que se expresa de la siguiente manera:

Si ID/EX.Reg Write e ID/EX.RegDst = 0 e

ID/EX.Escribir Reg rt = IF/ID.Leer Registro1 ó

ID/EX.Escribir Reg rt = IF/ID.Leer Registro2

detener la segmentación

La figura 48 muestra la actuación de la Unidad de Detección de Conflictos.

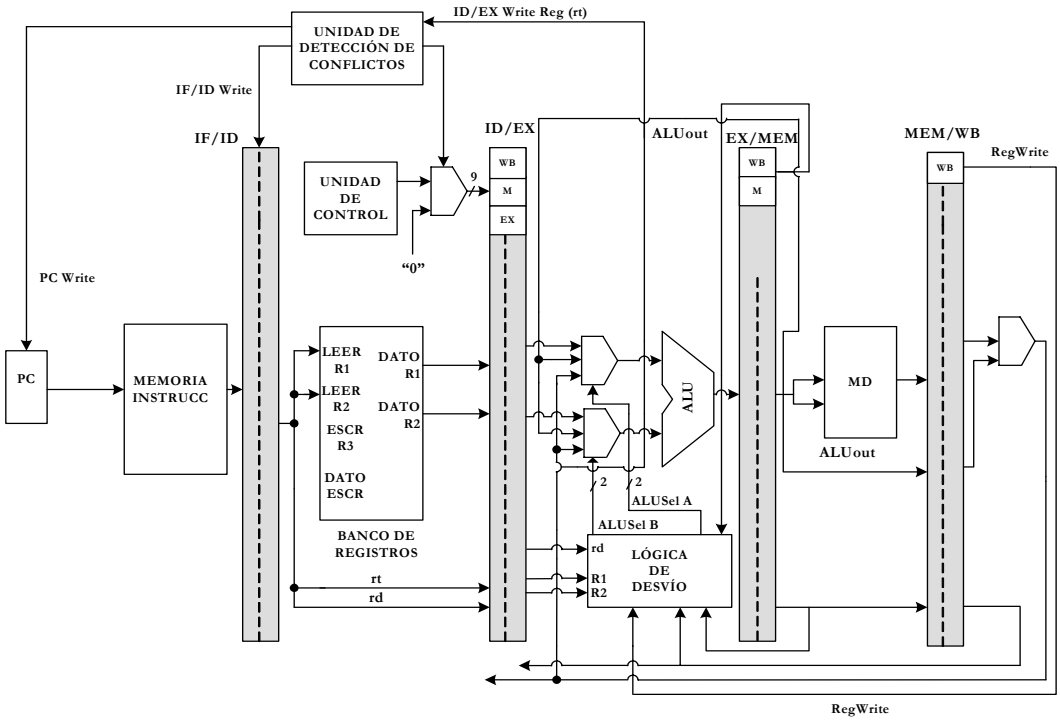


Figura 48.- Actuación de la Unidad de Detección de conflictos cuando trata los procedentes de las dependencias de datos con instrucciones de transferencia con memoria.

8.8- Conflictos de control por saltos condicionales

Los conflictos de control se producen cuando entran al cauce instrucciones de salto condicional (beq). Hasta que no sean completadas dichas instrucciones en la etapa MEM no se va a poder conocer la siguiente instrucción que ha de introducirse al cauce.

La técnica más sencilla y eficaz para solucionar este conflicto es introducir tres burbujas detrás de cada instrucción condicional. Aunque este procedimiento degrada mucho el rendimiento, debido a la gran cantidad de instrucciones condicionales que existen en los programas.

8.8.1- Técnicas para reducir los conflictos de control

Las técnicas suelen emplear conjuntamente un hardware específico y la actuación de un compilador, que deberá ser específico para el procesador segmentado que se utilice.

- Diseñar una lógica auxiliar específica que cuando detecta en la etapa ID una instrucción beq, calcule el valor que tomará el flan Z y halle el valor del PC correspondiente. Este hardware resulta muy caro.
- Diseñar un compilador específico para el procesador que analice las instrucciones y actúe en los saltos condicionales realizando alguna de las siguientes funciones:
    1. Introduciendo instrucciones NOP o burbujas al detectar las instrucciones condicionales. Esta solución disminuye el rendimiento.
    2. Eligiendo las instrucciones alternativas del programa, previas a la condicional, tales que no influyan en el desarrollo del salto. El rendimiento dependerá de las instrucciones que se introduzcan.
    3. Suponer que todos los saltos van a ser **no efectivos.** Se realiza un estudio de probabilidades. Cuando se falla en la predicción hay que deshacer todos los cambios que se han realizado. Para esta técnica se necesita un hardware adicional que lleve a cabo la **limpieza del cauce** ("flushing"). El MIPS emplea esta técnica.
    4. Suponer que todos los saltos van a ser **efectivos.** Esta solución es interesante en programas con instrucciones complejas.

8.8.2- Predicción de bifurcaciones

Consiste en suponer que no se va a realizar el salto y proseguir introduciendo instrucciones en el cauce.

Si se falla en la predicción habrá que deshacer todos los cambios que ha ocasionado las instrucciones posteriores a beq.

En el MIPS se añade un hardware auxiliar que cuando detecta que el salto es efectivo, limpia el cauce. La Unidad de Control comprueba si FZ = 1 para cada instrucción beq, en caso afirmativo genera una señal de **flush** a las etapas que se citan:

- **Etapa IF**: Envía una señal **IF. Flush** que pone a cero el campo de la instrucción del registro IF/ID.
- **Etapa ID**: La Unidad de Control genera una señal **ID. Flush** que realiza una operación OR con la señal de detección de conflictos. Ver figura 39.
- **Etapa EX**: Se genera una señal **EX. Flush** que a través de dos multiplexores pone a cero las señales de control que pasan a la siguiente etapa.

Cuando la Unidad de Control detecta que FZ = 0 para una instrucción beq que está en la etapa MEM, activa las señales **IF. Flush, ID. Flush, EX. Flush** que deshacen todo lo originado por las tres instrucciones introducidas en el cauce detrás de beq. Además carga en PC la dirección del salto para que la siguiente instrucción que se meta en el cauce sea correcta.

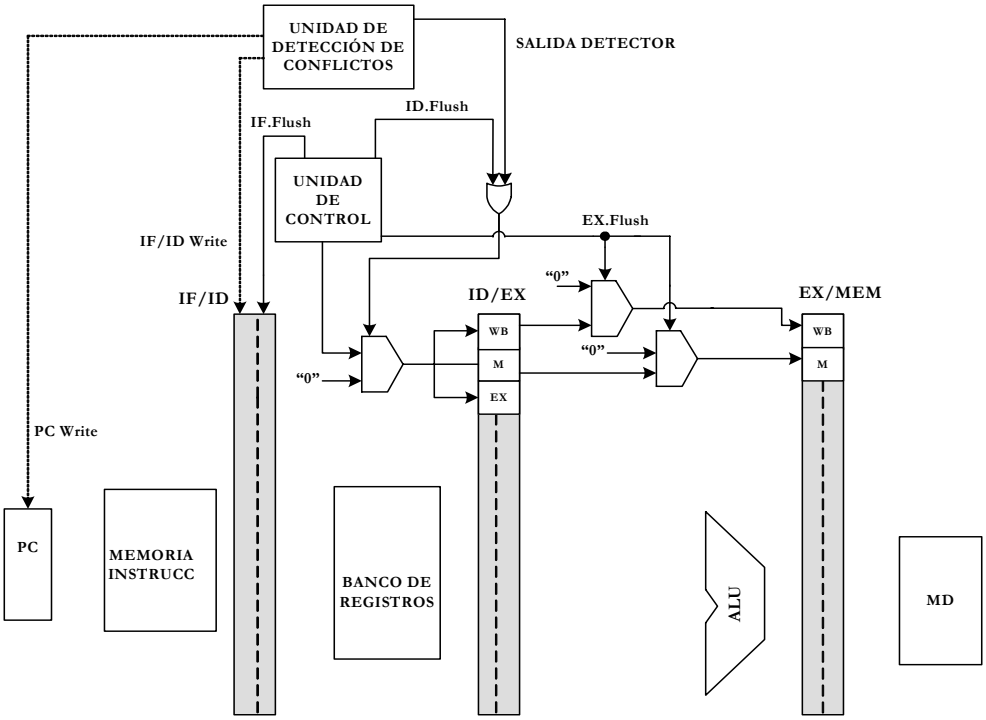


Figura 49.- Esquema del hardware necesario para la implementación de la técnica de predicción de bifurcaciones.

8.9- Conflictos por excepciones e interrupciones

Cuando se produce una excepción o interrupción sucede algo parecido a una instrucción de salto condicional. En el caso del MIPS se aceptan dos tipos de excepciones: Código OP no válido y desbordamiento en una instrucción aritmética. Las fases para el tratamiento de estas excepciones:

- Se detiene la instrucción en curso y se invalidan los resultados de las siguientes instrucciones introducidas al cauce.
- Se guarda la dirección del PC de la instrucción que ha provocado la excepción en EPC.
- Se salta q la rutina de tratamiento de la excepción, que comienza en la dirección 4000 0004 H. Aquí se analiza la causa (último bit del registro cause).

- Al finalizar la rutina de atención a la excepción se retorna a la instrucción que la produjo.

Para la desactivación de las señales de control en el tratamiento de las excepciones se utiliza la misma técnica que en las de salto condicional. Además se introduce desde un multiplexor al PC el valor 4000 0004 H. Figura 50.

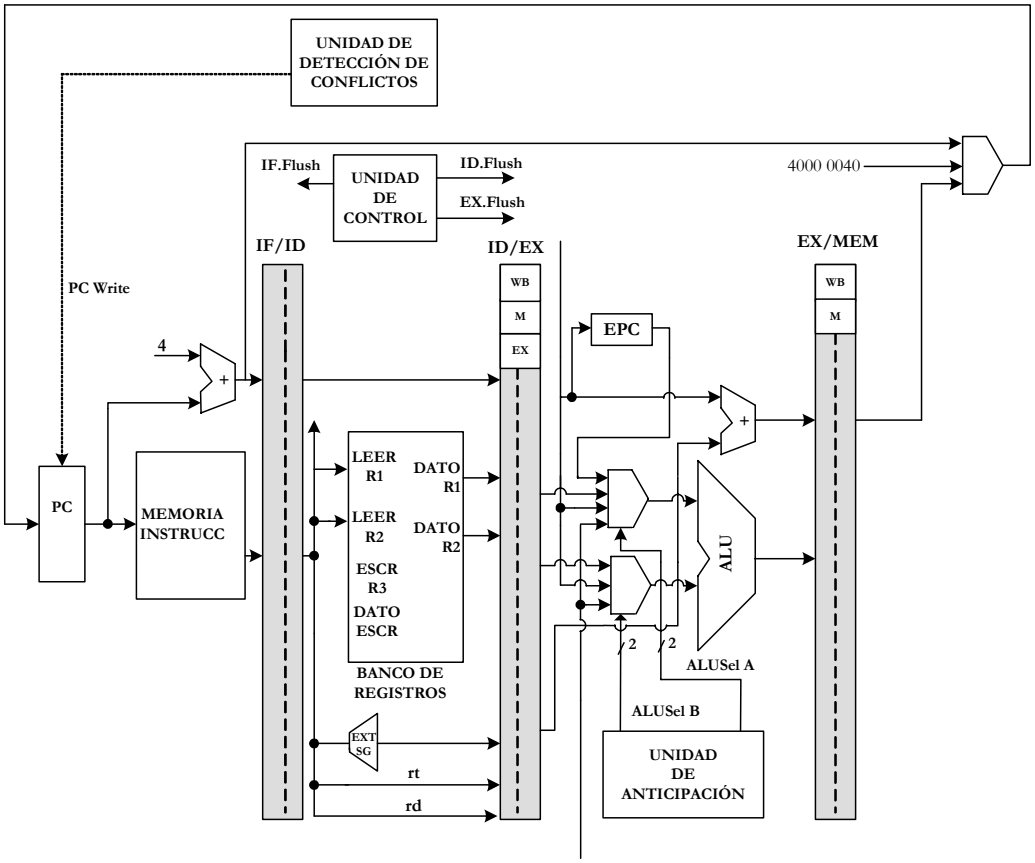


Figura 50.-Esquema de los elementos que actúan en el tratamiento de los conflictos por excepciones.

Si se trata de una excepción por desbordamiento se impide que se escriba el resultado de la ALU en la etapa WB para evitar que si un registro actúa como operando y destino, se borre su valor inicial y no se pueda conocer lo que ha ocurrido.

8.10- Incidencia de la profundidad de la segmentación en el rendimiento

El aumento de la profundidad en la segmentación no siempre incrementa el rendimiento del procesador.

Al aumentar la profundidad se incrementan las detenciones ocasionadas por los conflictos por dependencias de datos. También se elevan las burbujas necesarias para resolver conflictos de control. Por último, al elevar el número de etapas tiene mayor incidencia los tiempos de carga de los registros interetapas y el de los flancos de la señal de reloj.

En la siguiente figura se muestra un gráfico que relaciona la profundidad con el rendimiento relativo. Si el nivel de segmentación supera a 8, el rendimiento disminuye.

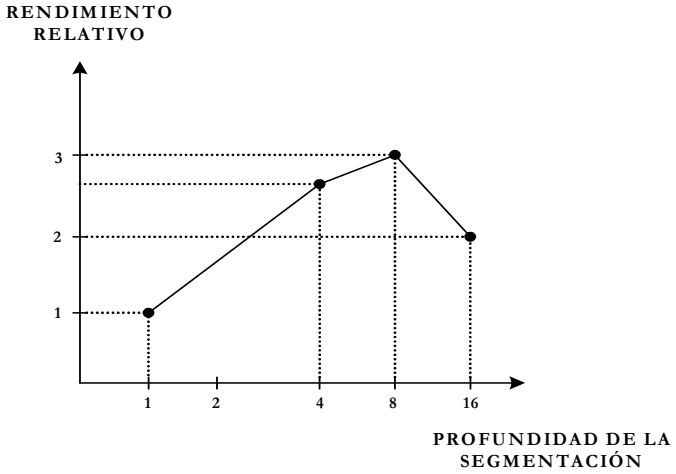


Figura 51.- Al aumentar la profundidad de la segmentación se eleva el rendimiento del procesador. Sin embargo si supera a 8 el rendimiento disminuye como consecuencia de los conflictos por dependencia de datos y de control, así como por la incidencia del tiempo de carga de los registros interetapas y del flanco de la señal de reloj.

ANEXO B

EJERCICIOS

1. TEMA 8: SEGMENTACIÓN

1.1. Ejercicio 1

En un entorno multitarea hay tres tareas en memoria (T1, T2 y T3), estando activa la tarea T2. En un instante dado, genera la dirección virtual (en binario):

00 0000 0000 0100 0000 0000 0000 0000 0000 0000 0000 0111

Sabiendo que la situación de la tabla de descriptores global y de la propia de la tarea, estando los valores expresados en hexadecimal, es la indicada en la figura 26.1, responder a las siguientes preguntas:

a) ¿A qué descriptor de segmento se accede?
b) ¿Dónde se encuentra el segmento accedido? Realizar un diagrama de la memoria que muestre claramente dónde se encuentra.
c) ¿Cuál es la dirección de la base del segmento?
d) ¿Cuál es el tamaño del segmento?
e) ¿Está presente el segmento en memoria?
f) ¿De qué tipo de segmento se trata?
g) ¿Cuál es la dirección física a la que se accede?

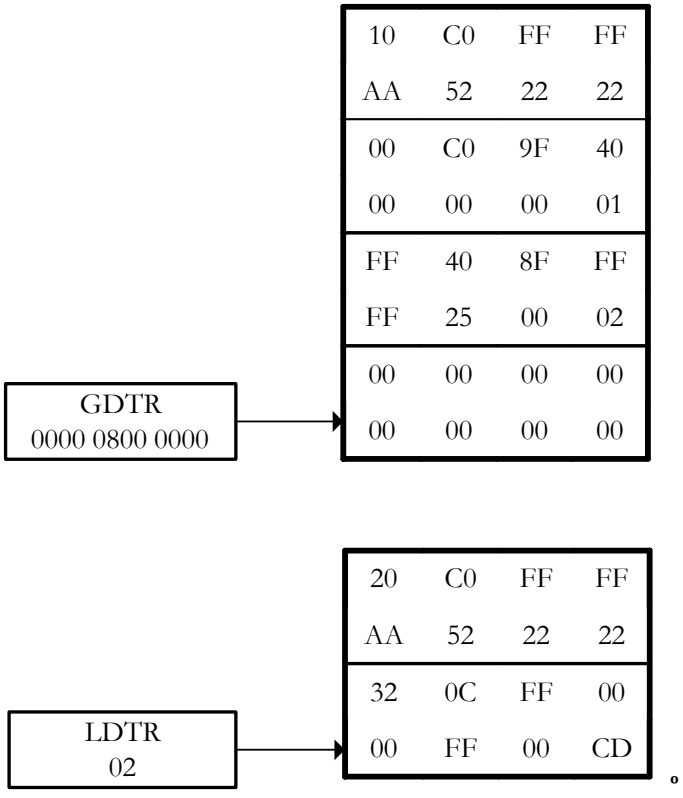


Figura 1 - Contenido de las tablas de descriptores

Resolución

a) ¿A qué descriptor de segmento se accede?

Lo primero que necesitamos saber es a qué descriptor se accede. Para ello, tomamos los primeros 16 bits de la dirección virtual, como vemos en la figura 2. De esos 16 primeros bits, los dos de menos peso serán el CPL (nivel de privilegio), el siguiente será el bit TI, que indicará si la tabla a la que se accede es la GDT o la LDT de la tarea en curso, y los 14 restantes serán el selector de segmento, que indicará a que selector se accede dentro de la tabla que corresponda.

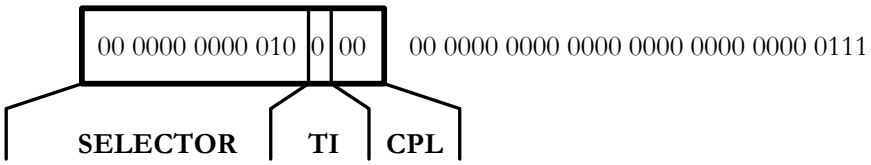


Figura 2 - Separación de los 16 bits de más peso de la dirección virtual

Como TI = 0, se utiliza la GDT. Vemos que el valor del campo Selector es 1, por lo que se accede al segundo descriptor de la GDT (recordemos que el primero es el selector nulo y no se utiliza).El selector accedido es por tanto, el que se indica en la figura 3

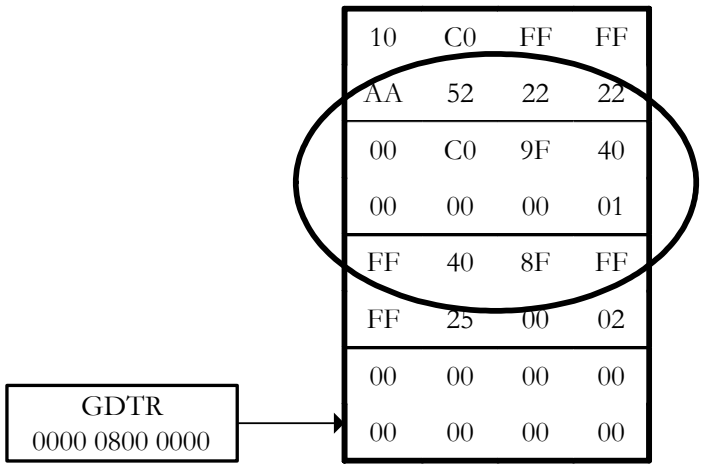


Figura 3 - Selector al que se accede

b) ¿Dónde se encuentra el segmento accedido? Realizar un diagrama de la memoria que muestre claramente dónde se encuentra.

Dado que su descriptor se encuentra en la GDT, el segmento se estará en el área global de memoria. Para saber en qué nivel de privilegio, es necesario examinar los atributos del descriptor. Para eso es conveniente recordar cuál es la estructura de un descriptor de segmento (figura 4), lo que nos va a permitir ubicarlos fácilmente.

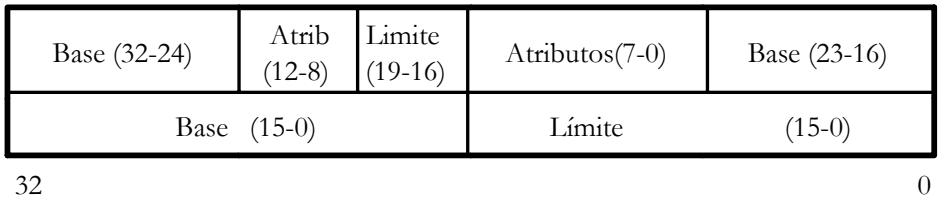


Figura 4 - Estructura de un descriptor de segmento

Examinamos ahora los doce bits que indican los atributos.

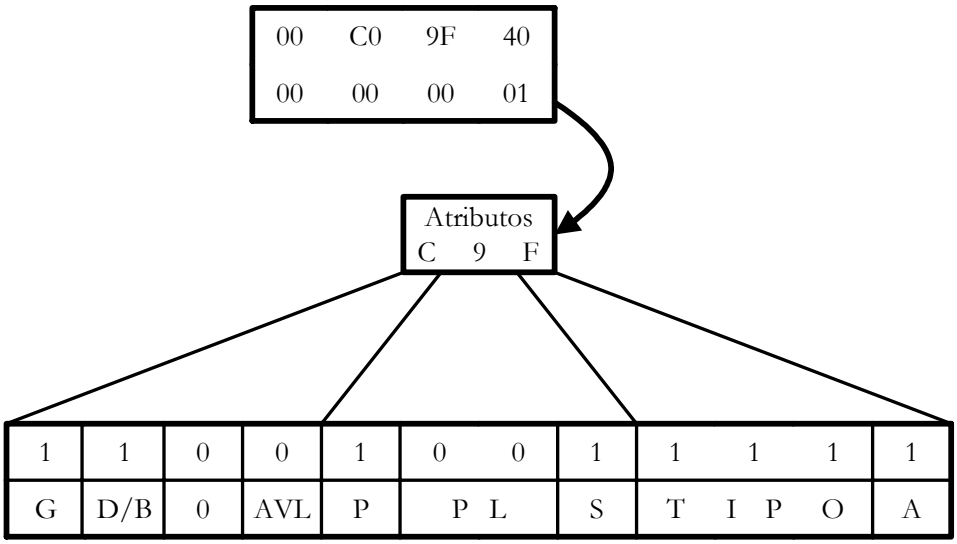


Figura 5 - Atributos de un descriptor

Vemos que el nivel de privilegio del segmento es 00. Por lo tanto, se encontrará en el área global, en la zona de PL = 0, como se recoge en el siguiente diagrama.

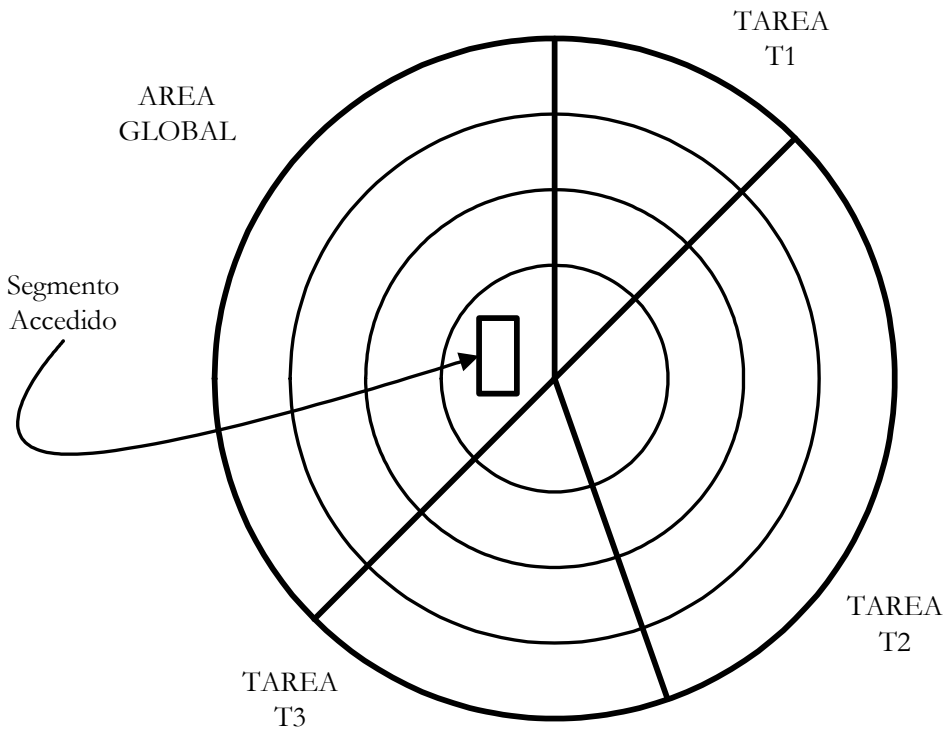


Figura 6 - Ubicación del segmento

c) ¿Cuál es la dirección de la base del segmento?
Teniendo en cuenta la figura 4, vemos claramente que la base de este segmento es:

00 40 00 07

d) ¿Cuál es el tamaño del segmento?

Si observamos el descriptor de segmento, vemos que su limite es 0 00 01. Sin embargo, al estar el bit G a 1 (activo) este límite no viene expresado en bits, sino en páginas de 4Kb. El tamaño del segmento es, por tanto, de 4Kb.

e) ¿Está presente el segmento en memoria?

El bit P está a 1 (activo), con lo que el segmento sí está presente en memoria.

f) ¿De qué tipo de segmento se trata?

Como el primer bit del campo 'tipo' de los atributos está a 1, se trata de un segmento de código, y los dos bits siguientes indican, respectivamente, que se trata de un segmento Conforming (o ajustable) y que puede leerse.

g) ¿Cuál es la dirección física a la que se accede?

Para obtener la dirección física, tomamos la base del segmento accedido, D0 00 92 A0, y le sumamos el desplazamiento que indica la dirección virtual, es decir:

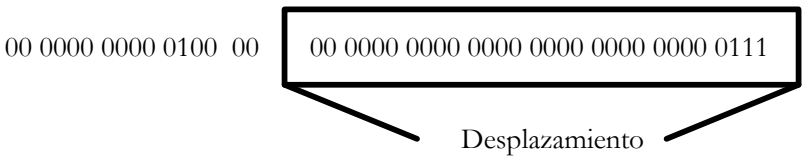


Figura 7 - Desplazamiento en una dirección virtual

Que en hexadecimal queda 00 00 00 07.

Por lo tanto, la dirección física accedida es 00 40 00 00 + 00 00 00 07 = 00 40 00 07

1.2. Ejercicio 2

Se proporcionan a continuación los datos de cuatro segmentos de un entorno multitarea en el que conviven tres tareas (T1, T2 y T3, siendo T1 la activa), además de una dirección virtual para cada uno de ellos. Se pide calcular sus descriptores y construir, en base a ellos, las tablas de descriptores (GDT y LDT de la tarea en curso), expresando para todos los segmentos el límite en unidades de 4Kb, y dibujar asimismo la situación de memoria correspondiente.

a) Segmento A: Es un segmento de datos normal, asociado a software de 32 bits, cuyo nivel de privilegio es 2. Ocupa 64 Kb, y la última vez que estuvo en memoria, comenzaba en la dirección AA AA 00 00. Es un segmento escribible. Se corresponde con la dirección virtual

00 0000 0000 0100 1000 0000 0000 0000 0000 0000 0000 0111

b) Segmento B: Es un segmento de código normal de software de 32 bits, de máximo nivel de privilegio. Comienza en la dirección 01 00 80 00 y ocupa 64Kb. Es un segmento ajustable y leíble. Se corresponde con la dirección virtual

00 0000 0000 0011 0000 0000 0000 0000 0000 0000 0000 0111

c) Segmento C: Es el segmento de pila correspondiente al segmento C. Ocupa 32Kb, comenzando en la dirección en que acaba el segmento B. Se corresponde con la dirección virtual

00 0000 0000 0101 0000 0000 0000 0000 0000 0000 0000 0111

d) Segmento D: Es un segmento de código que contiene rutinas de sistema. Comienza en la dirección 02 00 00 00 y ocupa 128Kb. Es un segmento ajustable pero no puede ser leído. Se corresponde con la dirección virtual

00 0000 0000 0010 0000 0000 0000 0000 0000 0000 0000 0111

Resolución

Primeramente, conviene recordar la estructura de los descriptores de segmento (ver figura 4) y la colocación de los bits de atributo (figura 5), así como la división en campos de una dirección virtual (figuras 2 y 26). Una vez hecho esto, resolver este ejercicio es tan sencillo como cambiar los datos de esas figuras por los del enunciado.

a) Segmento A
Es un segmento de datos normal, y es escribible: Por ser un segmento de datos, los dos primeros bits del campo tipo (C/D# y ED) estarán a cero; por ser escribible, el tercero estará a 1. Por ser un segmento normal, el bit S estará a 1.

Asociado a software de 32 bits: Esto indica que el bit D/B estará a 1, ya que es un segmento nativo.

Su nivel de privilegio es 2, por lo que el campo DPL contendrá 10

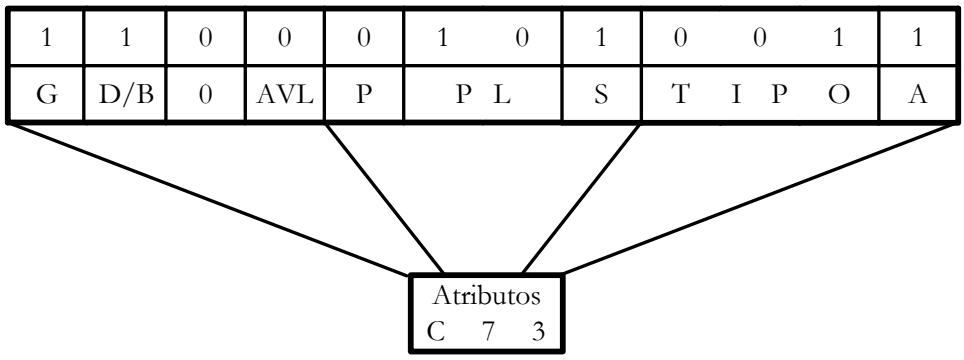
Ocupa 64 Kb, lo cual equivale a 16 unidades de 4Kb, por lo que el límite contendrá $00010_{)h}$

La última vez que estuvo en memoria, comenzaba en la dirección AA AA 00 00, así que esto será lo que contenga el campo base. Al no residir actualmente en memoria, el bit P estará a 0

Se corresponde con la dirección virtual
00 0000 0000 0100 1000 0000 0000 0000 0000 0000 0000 0111

Si separamos esta dirección virtual en sus campos, descubrimos que accede al tercer selector (selector número 2) de la GDT, por lo que este segmento estará ubicado en el área global.

Con estos datos, ya podemos construir establecer sus atributos y su descriptor (figura 8)



AA	C0	73	AA
00	00	00	10

Figura 8 - Atributos y descriptor del segmento A

b) Segmento B

Es un segmento de código normal, ajustable y leíble: Los tres bits del campo tipo estarán a uno, el primero por ser un segmento de código, el segundo, por ser ajustable; y el tercero, por ser leíble. Por ser un segmento normal, el bit S estará a 1.

Asociado a software de 32 bits Esto indica que el bit D/B estará a 1, ya que es un segmento nativo.

De máximo nivel de privilegio, es decir, nivel 0. Por lo tanto, el campo DPL contendrá 00.

Comienza en la dirección 01 00 80 00, que será lo que contenga el campo base. Además, el bit P estará a uno.

Ocupa 64Kb, lo cual equivale a 16 unidades de 4Kb, con lo cual el campo límite contendrá $00010_{)h}$

Se corresponde con la dirección virtual

00 0000 0000 0011 0000 0000 0000 0000 0000 0000 0000 0111

Si separamos esta dirección virtual en sus campos, descubrimos que accede al segundo selector (selector 1) de la LDT, por lo que este segmento estará ubicado en el área local de la Tarea 1.

Con estos datos, ya podemos construir establecer sus bits de atributos y su descriptor (figura 9).

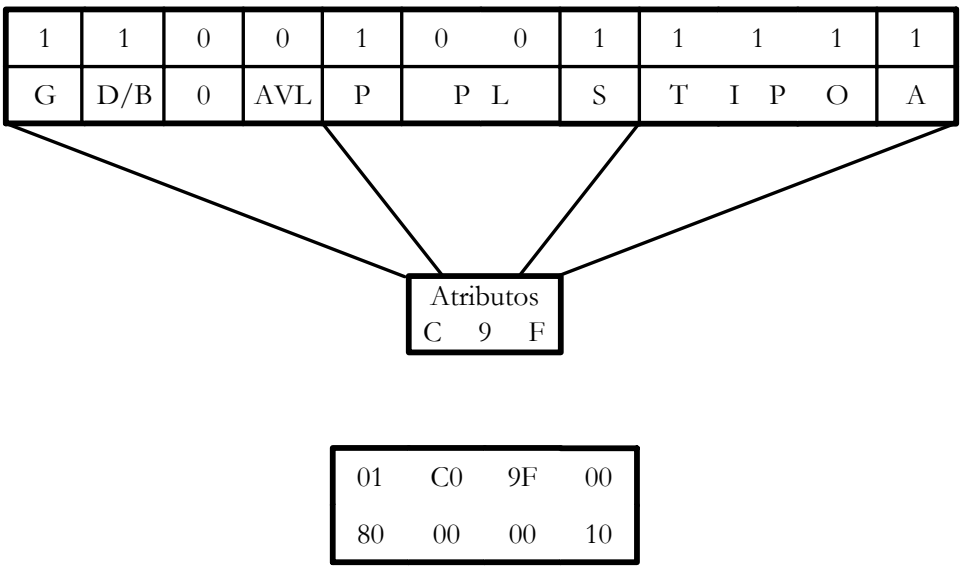


Figura 9 - Atributos y descriptor del segmento B

c) Segmento C

Es el segmento de pila correspondiente al segmento C. Dado que es un segmento de pila, su campo tipo tendrá el bit C/D# a 0, el bit ED a 1 y el bit W a 1, ya que los segmentos de pila crecen hacia abajo y pueden modificarse. Al estar asociado al segmento B, tendrá su mismo nivel de privilegio, es decir, 00.

Ocupa 32Kb, es decir, 8 unidades de 4Kb, por lo que su campo límite contendrá 00008

Comienza en la dirección en que acaba el segmento B. Para hallarla sumamos a la base del segmento B, 01 00 80 00, su tamaño, 00010, con lo que obtenemos 01 00 80 10, que será la dirección de comienzo del segmento C

Se corresponde con la dirección virtual

00 0000 0000 0101 0000 0000 0000 0000 0000 0000 0000 0111

Es decir, corresponde al tercer descriptor de la LDT, por lo que estará ubicado en el área local de la tarea T1.

Sus atributos y su descriptor quedan reflejados en la figura 10

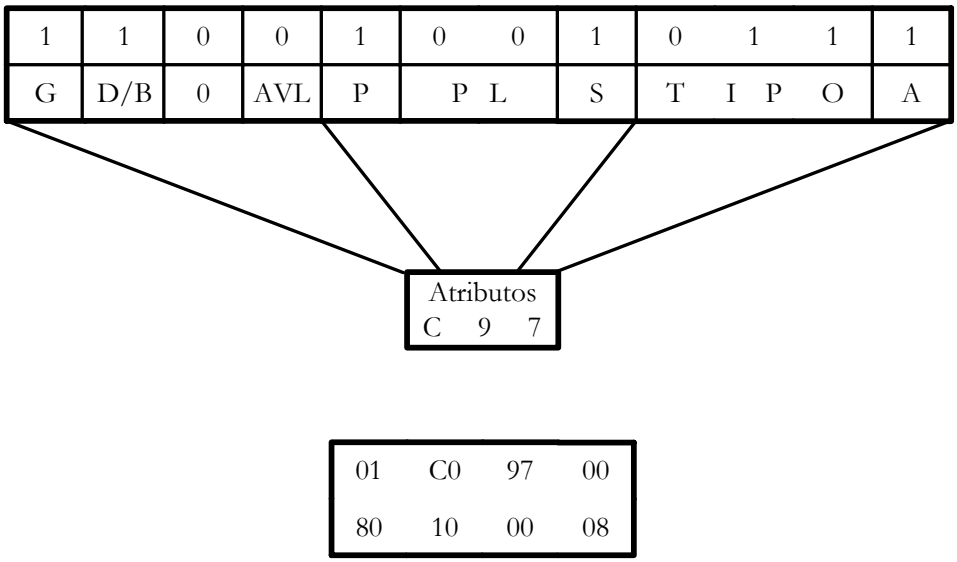


Figura 10 - Atributos y descriptor del segmento C

d) Segmento D

Es un segmento de código que contiene rutinas de sistema, ajustable pero que no puede ser leído. Al ser un segmento de rutinas del sistema, podemos deducir que su nivel de privilegio será el máximo (00). Además, al ser de código, su bit C/D# será uno; al ser ajustable, el bit Conforming también será uno, pero el bit de lectura estará a cero.

Comienza en la dirección 02 00 00 00, que será lo que contenga el campo base.

Ocupa 128Kb, es decir, 32 unidades de 4Kb, con lo que el campo límite contendrá $00020_{)h}$

Se corresponde con la dirección virtual

00 0000 0000 0010 0000 0000 0000 0000 0000 0000 0000 0111

Observamos que accede al segundo selector (selector número 1) de la GDT, por lo que el segmento estará ubicado en el área global.

En base a esto, construimos el descriptor reflejado en la figura 11

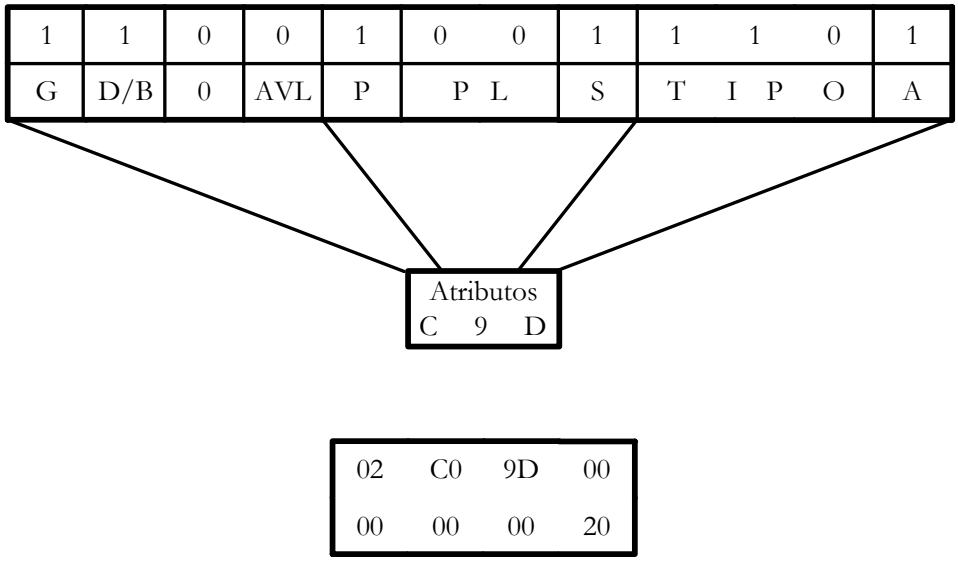


Figura 11 - Atributos y descriptor del segmento D

Una vez construidos los descriptores de los cuatro segmentos, podemos rellenar las tablas (GDT y LDT de la tarea T1), recordando incluir el descriptor nulo en la primera posición de la GDT. Así, las tablas quedan como indica la figura 12 (los guiones indican campos para los que no disponemos de datos)

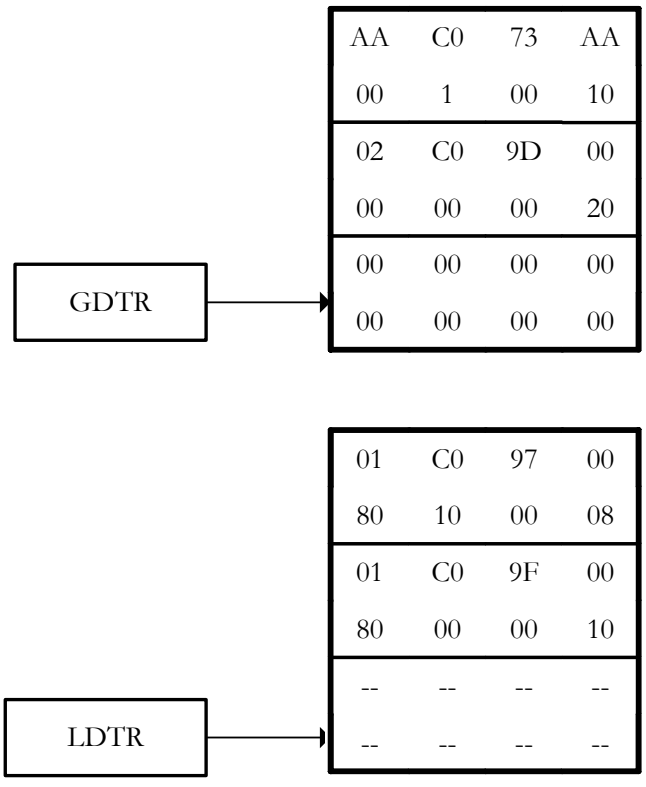


Figura 12 - GDT y LDT para la tarea T1

Por último, dibujamos la situación de memoria que reflejan estas tablas (figura 13)

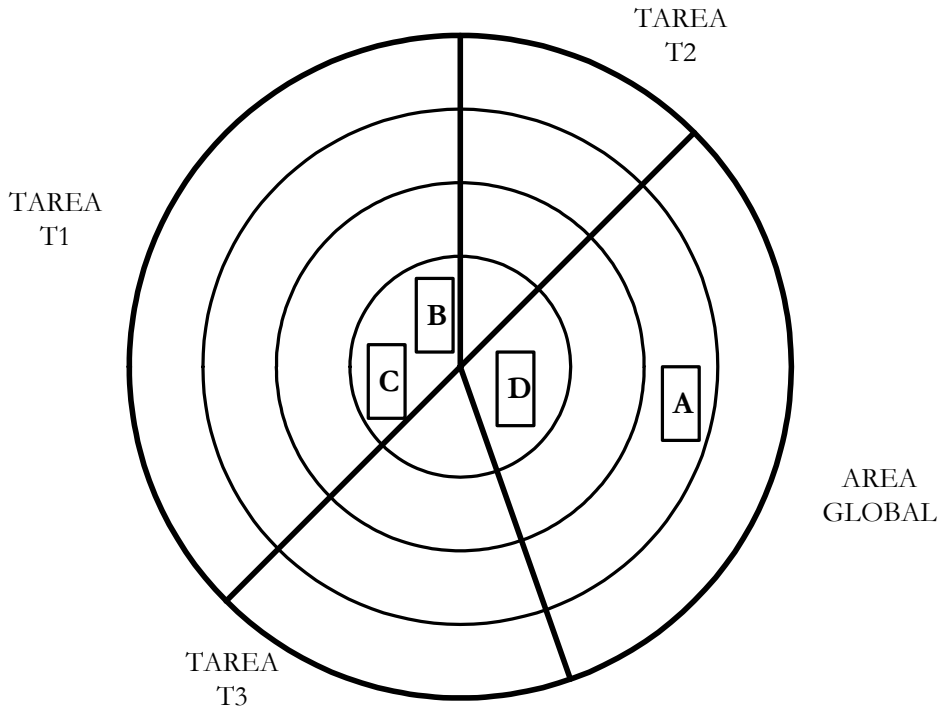


Figura 13 - Situación de memoria

1.3. Ejercicio 3

Se trata ahora de realizar el proceso contrario: en lugar de disponer de una dirección virtual y hallar la dirección física a la que corresponde, partiremos de la dirección física accedida y será necesario deducir cuál fue la dirección virtual que provocó ese acceso, además de rellenar, en la medida de lo posible, el descriptor del segmento accedido. Para ello disponemos de los siguientes datos:

- **Se accedió a la instrucción que ocupa la dirección FF 00 00 FF, que pertenece a una rutina del sistema operativo a la que tienen acceso todas las tareas.**
- **El segmento accedido puede ajustar su nivel de privilegio y no puede ser leído.**
- **La base del segmento es múltiplo de 64Kb (consideraremos que es el múltiplo más cercano), que es también el tamaño del mismo. Se trabaja con unidades de 4Kb para expresar los límites.**
- **Se accede al primer descriptor válido de la tabla que se utilice.**
- **No es un segmento de sistema, y está asociado a código de 32 bits**

Resolución

Antes de resolver este ejercicio, sería conveniente recordar brevemente el funcionamiento de la segmentación (figura 14)

Partimos de la dirección accedida: FF 00 00 FF Sabemos que la dirección de la base del segmento es múltiplo de 64Kb = $10000_{)h}$, por lo que debe acabar en cuatro ceros hexadecimales Por lo tanto, consideraremos que la base es FF 00 00 00, y por tanto el desplazamiento será, en hexadecimal, 00 00 00 FF.

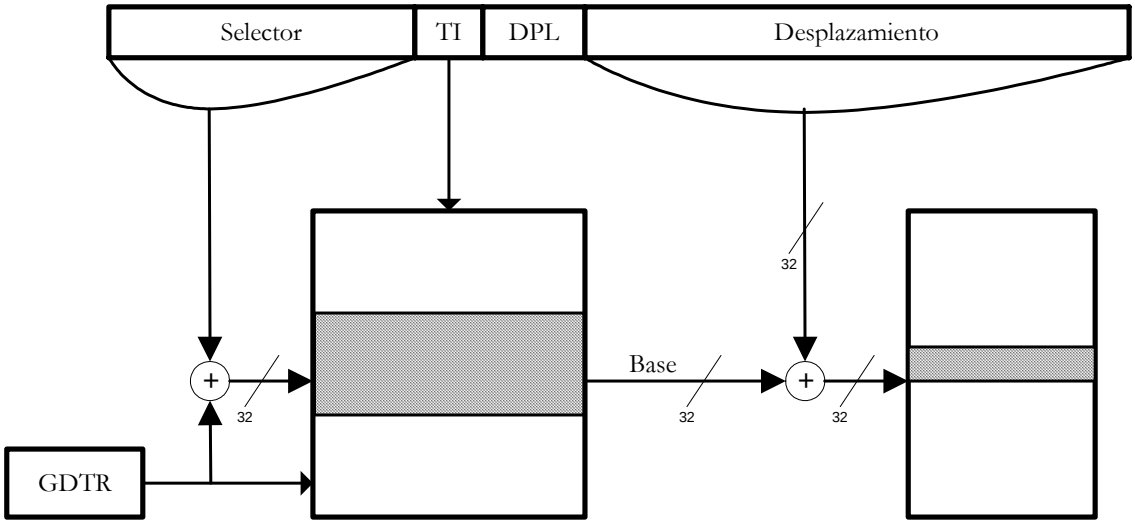


Figura 14 - Funcionamiento de la segmentación

Dado que la rutina de la que forma parte esta instrucción es accesible por todas las tareas, deducimos que está en el área global de memoria, por lo que su descriptor estará en la GDT, y el bit TI de la dirección virtual será 0. Además, por el momento consideraremos que tan sólo un segmento de código de nivel 0 puede acceder a otro de ese nivel (más adelante veremos que existen recursos que permiten saltarse esta regla de protección), por lo que el RPL que aparece en la dirección virtual será 00

El primer descriptor válido de la GDT es el segundo (el número 1), con lo que el selector que le corresponde es el $00\ 0000\ 0000\ 0001_{)2}$

Podemos ahora, por tanto, rellenar el esquema con los datos que hemos obtenido:

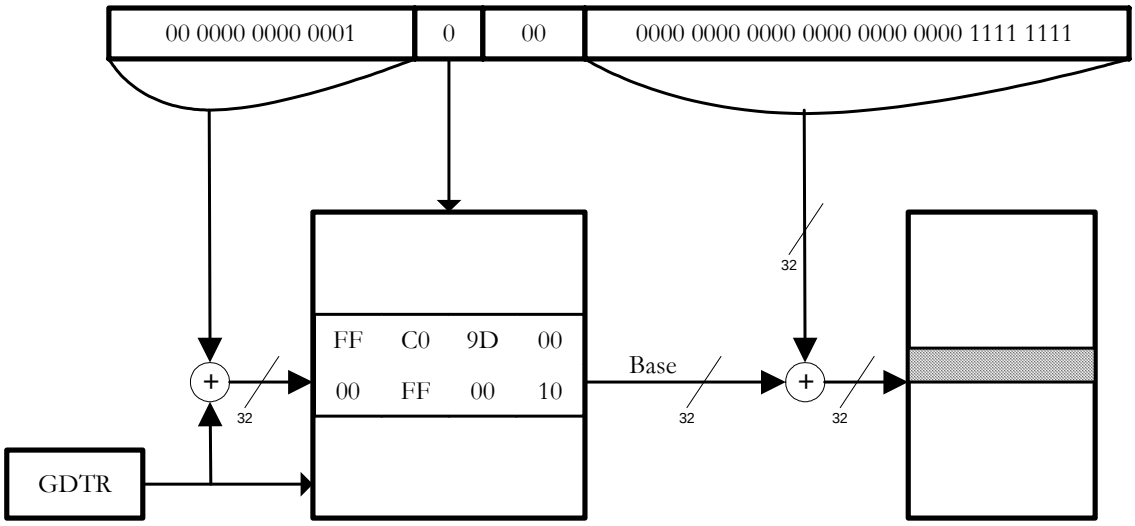


Figura 15 - Dirección virtual y descriptor correspondiente

1.4. Ejercicio 4

En un entorno multitarea hay tres tareas en memoria (T1, T2 y T3), estando activa la tarea T2. En un instante dado, genera la dirección virtual (en binario):

00 0000 0000 0010 0000 0000 0000 0000 0000 0000 0000 0111

En la primera posición útil de las tablas de segmentos hay información relativa a los dos segmentos que se describen a continuación, pudiendo existir además descriptores relativos a otros.

Segmento 1: Es un segmento de código normal de software de 32 bits, correspondiente a software de usuario. Comienza en la dirección AA AA 00 00 y ocupa 64Kb. Es un segmento no ajustable y no leíble, exclusivo de la tarea T2.

Segmento 2: Es un segmento de código que contiene rutinas de sistema. ocupa 128Kb y acaba donde comienza el segmento 1. Es un segmento ajustable pero no puede ser leído. Puede ser accedido por todas las tareas.

a) ¿Al descriptor de qué segmento se accede?
b) ¿Dónde se encuentra el segmento accedido? Realizar un diagrama de la memoria que muestre claramente dónde se encuentran ambos segmentos y a cuál de ellos se accede.
c) ¿Cuál es el tamaño del segmento 2?
d) ¿Cuál es la dirección física a la que se accede? Realizar un diagrama que explique el funcionamiento de la segmentación en este caso concreto.

2. TEMA 9: PAGINACIÓN

2.1. Ejercicio 1

Supongamos la situación del Directorio y las Tablas de Páginas que muestra la figura 26.16. Teniendo en cuenta los datos que proporciona el diagrama, responder a las preguntas siguientes:

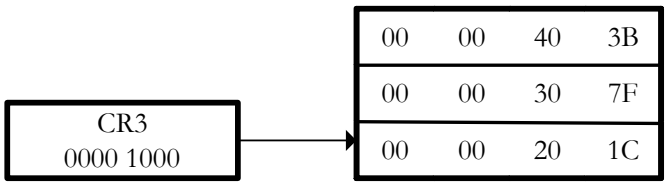


Tabla de páginas 0

00	00	60	7C
00	00	50	1C

Tabla de páginas 1

00	00	80	1C
00	00	70	7F

Tabla de páginas 2

00	00	A0	7C
00	00	90	1F

Figura 16 - Directorio y Tablas de Páginas

a) ¿Cuáles de las páginas están presentes en memoria? ¿Cuáles han sido accedidas? ¿Cuáles han sido modificadas?
b) ¿Cuál es el nivel de privilegio de las páginas de la tabla de paginas 0?
c) ¿Cuál es la dirección de comienzo de cada una de las tablas de páginas?

d) ¿Cuáles de las páginas son cacheables?
e) ¿Qué tamaño tiene cada una de las páginas?

Resolución

Antes de resolver las preguntas, conviene recordar cuál es la estructura de las entradas del directorio y de las tablas de páginas (figura 17).

Direccion (20 bits)	0000	SIZ/0	D	A	PCD	PWT	U/S	R/W	P

Figura 17 - Estructura de una entrada de directorio

a) ¿Cuáles de las páginas están presentes en memoria? ¿Cuáles han sido accedidas? ¿Cuáles han sido modificadas?.

Para responder a estas preguntas, analizaremos los atributos de cada entrada, que aparecen reflejados en la figura 18

Direccion (20 bits)	0000	SIZ/0	D	A	PCD	PWT	U/S	R/W	P
Entrada 0 del Directorio		0	0	0	1	1	1	0	0
Entrada 1 del Directorio		0	1	1	1	1	1	1	1
Entrada 2 del Directorio		0	0	1	1	1	0	1	1
Entrada 0 de Tabla 0		0	0	0	1	1	1	0	0
Entrada 1 de Tabla 0		0	1	1	1	1	1	0	0
Entrada 0 de Tabla 1		0	0	0	1	1	1	0	0
Entrada 1 de Tabla 1		0	1	1	1	1	1	1	1
Entrada 0 de Tabla 2		0	0	0	1	1	1	1	1
Entrada 1 de Tabla 2		0	1	1	1	1	1	0	0

Figura 18 - Atributos en las entradas del Directorio y las Tablas de Páginas

Están presentes, por tanto, las páginas referenciadas por la entrada 1 de la tabla de páginas 1 y la entrada 0 de la tabla 2, así como las entradas 1 y 2 del directorio, es decir, las que contienen las tablas de páginas 1 y 2.

Han sido accedidas las páginas referenciadas por la entrada 1 de las Tablas 0, 1 y 2, así como las entradas 1 y 2 del directorio. Todas esas páginas, salvo la entrada 2 del Directorio, han sido, además, modificadas.

b) ¿Cuál es el nivel de privilegio de las páginas de la tabla de paginas 0?

Como vemos en la figura 18, para ambas páginas el nivel de privilegio es uno, es decir, son páginas de usuario.

c) ¿Cuál es la dirección de comienzo de cada una de las tablas de páginas?

Para responder a esta pregunta es necesario observar la parte de la entrada que no hemos contemplado en la figura 18, que nos da los 20 bits más significativos de la dirección. Dado que todas las páginas son de 4Kb, al estar el bit SIZ de CR3 a cero, los 12 bits restantes serán ceros. Por lo tanto, la dirección de comienzo de la tabla 0 será la 00 00 20 00, la de la tabla 1 será 00 00 30 00 y la de la tabla 2 será 00 00 40 00. Se observa que están colocadas consecutivamente en memoria

d) ¿Cuáles de las páginas son cacheables?

Observando el bit PCD de los atributos de las entradas (figura 18) , podemos ver que son cacheables todas las páginas.

e) ¿Qué tamaño tiene cada una de las páginas?

Como ya se ha dicho, dado que el bit SIZ de CR3 está a 0, todas las páginas son de 4Kb.

2.2. Ejercicio 2

Retomemos el ejercicio del apartado 1.1. Supongamos que, con la misma situación para las tablas de descriptores (figura 1), se activa la paginación. En base a la información relativa al Directorio de Tablas de Páginas y las Tablas de Páginas que se proporciona en la figura 16,responder a las siguientes preguntas.

a) ¿ Cuál es la dirección lineal que se obtiene a partir de la dirección virtual?
b) ¿ A qué entrada del Directorio se accede?
c) ¿ A qué tabla de páginas se accede?
d) ¿ Cuál es la dirección física que se corresponde con la dirección virtual inicial?

Resolución

a) ¿ Cuál es la dirección lineal que se obtiene a partir de la dirección virtual?

Recordemos que, cuando se activa la paginación, el Pentium trabaja con Segmentación Paginada. Por lo tanto, el primer paso de la traducción de la dirección virtual es exactamente el que se hizo en el ejercicio 1.1. La diferencia es que, utilizando sólo segmentación, el valor obtenido en dicha traducción (00 40 00 07) es la dirección física a la que se accede. En este caso, sin embargo, es la dirección lineal, que se utiliza para acceder al directorio de páginas y desde él a la tabla que corresponda, hasta obtener la dirección física.

b) ¿ A qué entrada del Directorio se accede?

Para saberlo, basta descomponer la dirección lineal en sus campos, como indica la figura 19. Vemos que se accede a la segunda entrada (entrada numero 1) del directorio, y, por tanto, a la Tabla de Páginas 1.

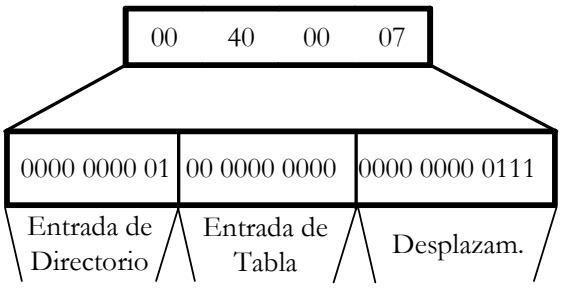


Figura 19 - División de una dirección lineal

c) ¿ A qué tabla de páginas se accede?

A la vista de la figura anterior, vemos que se accede a la primera entrada (numero 0) de la Tabla de Páginas 1.

d) ¿ Cuál es la dirección física que se corresponde con la dirección virtual inicial?

Para obtener la dirección física, habría que tomar la dirección base que indica la entrada a la que se ha accedido, 00007, y añadirle los doce ceros que no se almacenan en las tablas al ser todas las páginas múltiplos de 4Kb. Obtenemos, por lo tanto, 00 00 70 00. Si a esta dirección base le sumamos el desplazamiento, $007_{)h}$, obtenemos las dirección física a la que se accede, que en este caso es 00 00 70 07. También puede obtenerse directamente concatenando el desplazamiento a los bits de la dirección que proporciona la tabla.

2.3. Ejercicio 3

Se trata ahora de realizar el proceso contrario: conociendo la dirección a la que se accede, la FF 00 00 FF, hallar la dirección lineal de la que se partió. Rellenar también, en la medida de lo posible, las entradas del Directorio y de las Tablas de Páginas por las que se pase. Para ello sabemos que:

- **Se está trabajando con páginas de 4Kb**
- **El desplazamiento dentro de la Tabla de Páginas y del Directorio para acceder a la entrada que corresponde es de 256 bites.**

Resolución

Para completar la dirección virtual necesitamos conocer el número de entrada del Directorio y de la Tabla de Paginas que se utilizan, además del desplazamiento.

Comenzaremos por éste último. Como las páginas ocupan 4Kb, sus direcciones de comienzo serán múltiplos de ese valor, es decir, acabarán en doce ceros binarios. Por lo tanto, deducimos que la dirección de la base de la página a la que se accede es FF 00 00 00, y el desplazamiento será, por tanto, la diferencia entre la dirección física y la base, es decir,

$$\text{FF 00 00 FF} - \text{FF 00 00 00} = \text{00 00 00 FF}$$

Dado que solo tomamos los doce bits menos significativos, el valor a colocar en el campo desplazamiento de la dirección lineal será 0FF

Para hallar el número de entrada de la Tabla de Páginas y del Directorio conocemos el desplazamiento en bits, 256. Dado que cada entrada ocupa 32 bits, en ambos casos será la novena entrada, es decir, la número 8. Por lo tanto, en ambos campos de la dirección lineal colocaremos el valor $00\ 0000\ 1000_{)b}$. Así, la dirección lineal completa queda:

$$0000\ 0010\ 0000\ 0000\ 1000\ 0000\ 1111\ 1111_{)2} = 02\ 00\ 80\ FF_{)h}$$

En cuanto a la entrada de la Tabla de Páginas, con los datos de los que disponemos podemos rellenar el campo base, pero no los atributos (figura 20). De la entrada del Directorio, sin embargo, no podemos dar ningún valor.

FF	00	0 -	- -
- -	- -	- -	- -
- -	- -	- -	- -
- -	- -	- -	- -
- -	- -	- -	- -
- -	- -	- -	- -
- -	- -	- -	- -
- -	- -	- -	- -
- -	- -	- -	- -

Figura 20 - Tabla de páginas a la que se accede

2.4. Ejercicio 4

Suponiendo la misma situación de memoria que en el ejercicio 2.1, responder a las siguientes preguntas:

a) ¿Qué habría que modificar en la GDT para que se accediera a la primera página de la tercera tabla de páginas?

b) Realizar un diagrama que muestre el funcionamiento tanto de la segmentación como de la paginación en este caso, rellenando todos los campos posibles

c) ¿Qué dirección lineal se corresponde con ese acceso?

3. TEMA 11: MECANISMO DE PROTECCIÓN

3.1. Ejercicio 1

Dado el esquema de memoria de la figura 21, en el que C denota un segmento de código, D de datos y P de pila, determinar:

a) A qué segmentos de código se puede acceder desde C0
b) A qué segmentos de pila se puede acceder desde C0
c) A qué segmentos de datos se puede acceder desde C0
d) A qué segmentos de código se pude acceder desde C1
e) A qué segmentos de datos se puede acceder desde C1

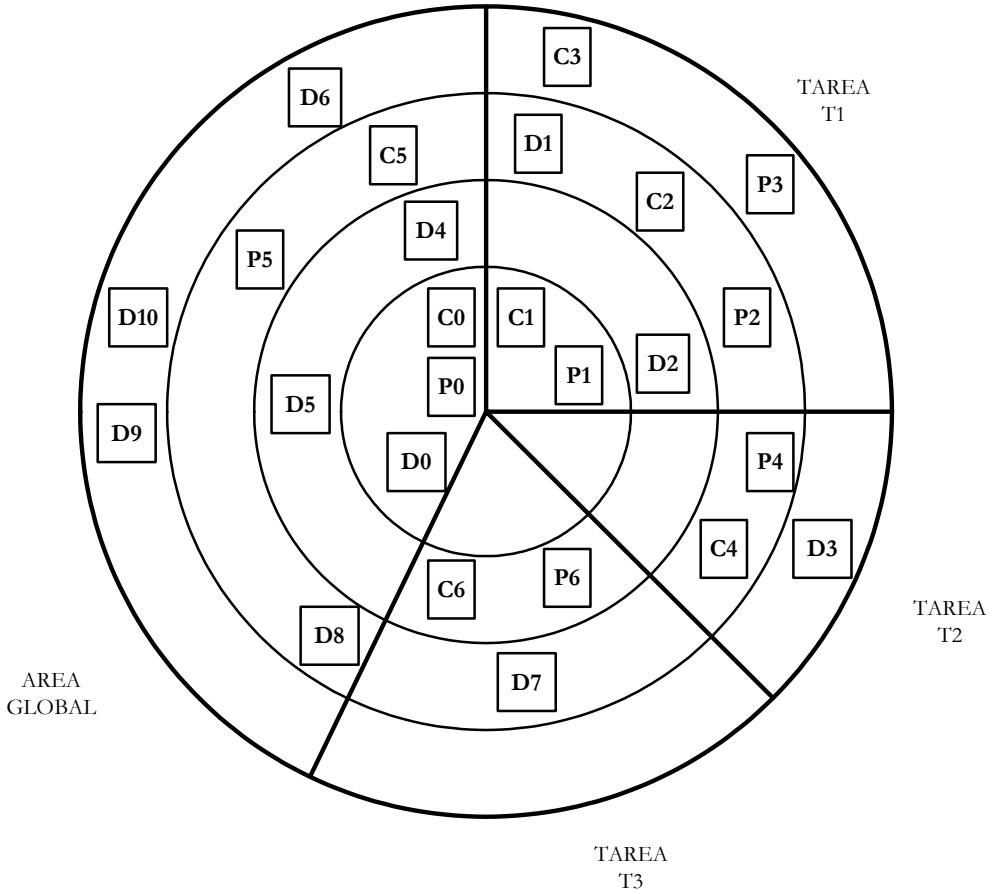


Figura 21 - Situación de memoria en un entorno multitarea

Resolución

a) A qué segmentos de código se puede acceder desde C0

Observamos que C0 se encuentra en el nivel de privilegio 0 del área local. Podría acceder a otros segmentos de código del área global y del nivel 0. Dado que no hay ninguno más, no puede acceder a ningún segmento de código. (Conviene recordar que desde el área global no puede accederse al área local de una tarea)

b) A qué segmentos de pila se puede acceder desde C0

Los segmentos de pila siguen la misma regla de acceso que los de código. Así, el segmento C0 puede acceder al segmento P0, que está en su misma área de memoria y mismo nivel de privilegio.

c) A qué segmentos de datos se puede acceder desde C0

Un segmento de código puede acceder a segmentos de datos de su misma área o del área global de nivel de privilegio igual o menor al suyo propio. En este caso, el segmento C0 podrá acceder a todos los segmentos de datos del área global, es decir: D0, D4, D5, D8, D6, D9 y D10.

d) A qué segmentos de código se pude acceder desde C1

C1 puede acceder al segmento C0, ya que se encuentra en su mismo nivel de privilegio y en el área global.

e) A qué segmentos de datos se puede acceder desde C1

C1 puede acceder a todos los segmentos de datos tanto del área local como de la suya propia, es decir, D0, D4, D5, D8, D6, D9 y D10 en el área global, y D1 y D2 en su área local.

3.2. Ejercicio 2

Dado el esquema de memoria de la figura 20, determinar desde qué segmentos se puede acceder a:

a) C1, estando activa la tarea T1
b) D4, estando activa la tarea T2
c) C5, estando activa la tarea T3
d) P4, estando activa la tarea T2
e) D0, estando activa la tarea T1
f) D0, estando activa la tarea T2

Resolución

a) C1, estando activa la tarea T1

Para poder acceder a T1, dado que se encuentra en el nivel de privilegio 0 de su área local, debería haber otro segmento de código en la misma área y nivel. Por lo tanto, no hay ningún segmento en la figura que pueda acceder a C1

b) D4, estando activa la tarea T2

D4 puede ser referenciado por el segmento C0 del área global. Ningún segmento de código de la tarea T2 puede acceder a él.

c) C5, estando activa la tarea T3

Dado que no hay ningún segmento de código al mismo nivel ni en el área global ni en la local a la tarea T3, ninguno de los segmentos de la figura puede acceder a C5 en esta situación.

d) P4, estando activa la tarea T2
P4 sólo puede ser accedido desde C4, ya que está en el área local a T2.

e) D0, estando activa la tarea T1
Puede ser accedido desde C1, que está en el área local de T1 y en el mismo nivel de privilegio.

f) D0, estando activa la tarea T2
No puede ser accedido desde ningún segmento, ni del área global ni de la tarea T2.

3.3. Ejercicio 3

¿Dónde habría que colocar un segmento de código para que pudiera acceder simultáneamente a los siguientes conjuntos de segmentos? Puede suceder que haya varias ubicaciones diferentes para el mismo segmento, o que no exista ninguna.

a) D10, D5 y D7
b) D10, D5 y P6
c) P2 y P4
d) C0 y C6

Resolución

a) D10, D5 y D7
D10 y D5 están en el área global, y D7 en la de la Tarea T3, por lo que el segmento pedido debe encontrarse en el área de T3. Además, dado que D5 es el segmento de mayor nivel de privilegio (1), deberá estar bien en el nivel 1 o en el 0 de la tarea T3.

b) D10, D5 y P6
Al igual que antes, debe estar en el área de memoria reservada para T3 para poder acceder a P6. Además, dado que P6 es un segmento de pila, obliga a que esté situado en su mismo nivel de privilegio, es decir, en el nivel 1 de la tarea T3

c) P2 y P4
Dado que P2 y P4 pertenecen a tareas distintas, es imposible que un mismo segmento pueda acceder a ambas.

d) C0 y C6
C0 está en el área global y C6 en la de la tarea T3, por lo que, de existir, el segmento debería esta en esta última zona de memoria. Sin embargo, C0 está en el nivel de privilegio 0 y C6 en el 1, y dado que los segmentos de código sólo pueden ser accedidos por otros de su mismo nivel de privilegio, es imposible que un mismo segmento pueda acceder a ambos.

3.4. Ejercicio 4

Dada la situación de memoria de la figura 21, determinar

a) A qué segmentos de datos puede accederse desde C6
b) A qué segmentos puede accederse desde D0
c) Desde qué segmentos puede accederse a P5
d) ¿Hay algún segmento que pueda acceder a todos los demás presentes en memoria?

4. TEMAS 12 y 13: PUERTAS DE LLAMADA Y DE TAREA

4.1. Ejercicio 1

Dado el esquema de memoria de la figura 22, responder a las siguientes preguntas

a) ¿Qué segmentos pueden usar DG2?
b) ¿Desde qué segmentos se puede acceder a CG1?
c) ¿Qué segmentos de código pueden utilizar la PLLG3?
d) ¿Puede acceder el segmento C10 al CG1?
e) ¿Desde qué segmentos de la tarea T2 se puede utilizar la PLL2?

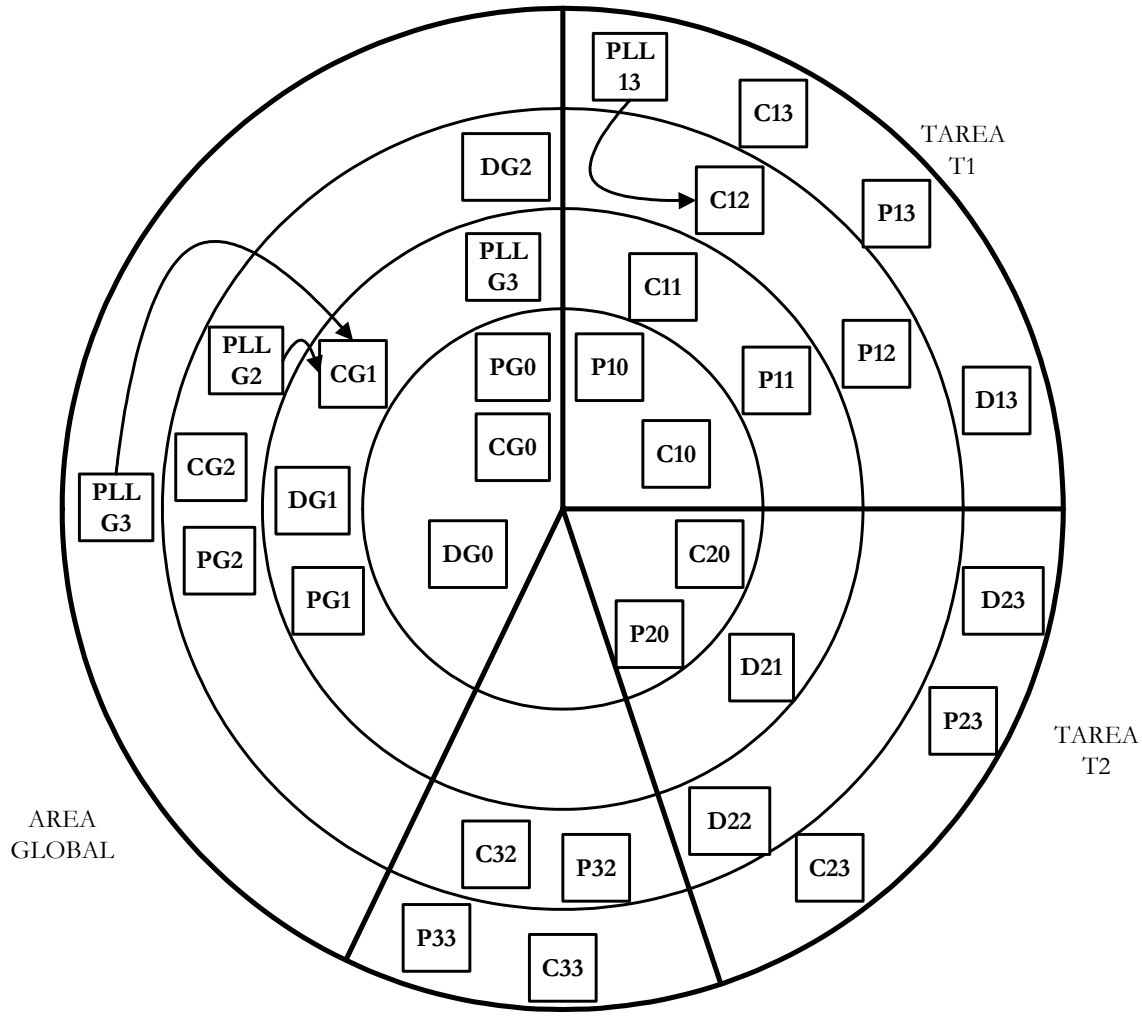


Figura 22 - Situación de memoria

Resolución

a) ¿Qué segmentos pueden usar DG2?

Dado que es un segmento de datos, pueden acceder a él todos aquellos segmentos cuyo nivel de privilegio sea igual o superior a 2, es decir:

- Del área global, CG2, CG1, CG0.
- De la tarea T1, C12, C11, C10.
- De la tarea T2, C20
- De la tarea T3, C32.

b) ¿Desde que segmentos se puede acceder a CG1?

Dado que es un segmento de código, pueden acceder a él aquellos que se encuentren en su mismo nivel de privilegio. Dado que PLLG3 y PLLG2 proporcionan acceso a CG1, también habrá que tener en cuenta aquellos segmentos que puedan llamar a las PLLs y cuyo nivel de privilegio se vea aumentado al acceder a CG1.Teniendo esto en cuenta, la lista de segmentos que pueden acceder a CG1 es.:

- Directamente: C11
- A través de PLLG3: CG2, C32, C33, C23, C12 C13
- A través de PLLG2: CG2, C12,

En conclusión: C11, CG2, C32, C33, C23, C12 y C13

c) ¿Qué segmentos de código pueden utilizar la PLLG3?

Como ya hemos dicho en la pregunta anterior: CG2, C32, C33, C23, C12 C13

d) ¿Puede acceder el segmento C10 al CG1?

No puede. Directamente el acceso no está permitido, ya que el nivel de privilegio de CG1 es inferior al de C10. C10 puede llamar a PLLG3 y PLLG2., que le permitiría acceder a CG1; sin embargo, dado que con dicho acceso su nivel de privilegio no resulta aumentado, tampoco es posible.

e) ¿Desde qué segmentos de la tarea T2 se puede utilizar la PLLG2?

Desde ninguno, ya que no hay ningún segmento que tenga el suficiente nivel de privilegio para acceder a ella y que a su vez resulte aumentado por el acceso al segmento CG1 desde la puerta.

4.2. Ejercicio 2

Dados los descriptores de la figura 23, indicar, para cada uno de ellos, si corresponde a una puerta de llamada o de tarea, y el valor de cada uno de sus campos.

Puerta A

00	80	EC	1F
00	01	00	07

Puerta B

00	80	E9	1F
00	01	00	07

Puerta C

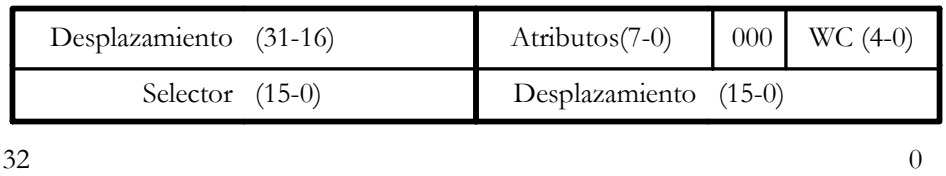
00	80	EB	1F
00	01	00	07

Figura 23 - Descriptores de puertas

Resolución

Antes de resolver este ejercicio, convendría recordar cuál es la estructura de una PLL y de una PT (figura 24), así como que el campo tipo tiene el valor $1100_{)b}$ para las puertas de llamada y $010B1_{)b}$ para las puertas de tarea. Una vez hecho esto, pasamos a analizar cada descriptor.

Puerta de Llamada



Puerta de Tarea

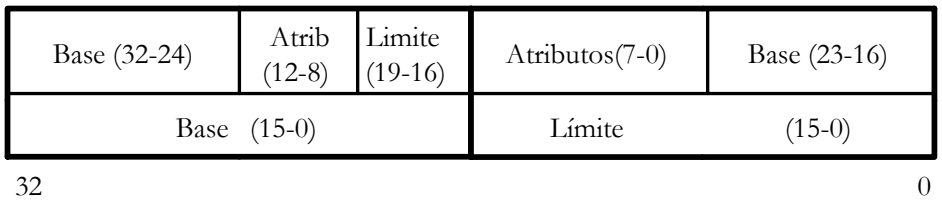


Figura 24 - Estructura de las puertas

- Puerta A

Si observamos los bits 0 a 7 del campo Atributos, observamos que es $EC_{)h} = 1110\ 1100_{)2}$, y dado que el tipo lo indican los cuatro bits de menos peso, se trata de una puerta de llamada. Por lo tanto, los demás campos del descriptor son:

- Selector: 00 01
- Desplazamiento: 00 80 00 07
- Word Counter (WC): es $11111_{)b}$, es decir, 31
- Atributos: Los atributos de un descriptor de puerta de tarea son el bit P, dos bits para el nivel de privilegio DPL, otro para indicar si es un segmento de sistema y el tipo. Según esto, se trata de una puerta de llamada de sistema que se encuentra en memoria cuyo nivel de privilegio es 3.

- Puerta B

Si observamos los bits 0 a 7 del campo Atributos, observamos que $E9_{)h} = 1110\ 1001_{)2}$, y dado que el tipo lo indican los cuatro bits de menos peso, se trata de una puerta de Tarea.. Por lo tanto, los demás campos del descriptor son:

- Base: 00 1F 00 01
- Límite: 00007
- Atributos: Son los mismos que en un descriptor normal. Observamos que el bit de granularidad está activo, por l oque el límite está expresado en unidades de 4Kb. El bit D/B está también a uno, indicando que se trata de un segmento asociado a código de 32 bits. Podemos decir además que está presente en memoria y que su nivel de privilegio es 3.
- Bit Busy: El cuarto bi del campo tipo indica si la tarea está o no ocupada. En este caso, está libre.

- Puerta C

Si comparamos este descriptor con el anterior, vemos que sólo difiere en un bit, que es precisamente el bit Busy. Todo lo dicho en el apartado anterior es válido para esta tarea, que sólo se diferencia en que está ocupada.

4.3. Ejercicio 3

Dado el esquema de memoria de la figura 22, responder a las siguientes preguntas:
a) ¿Qué segmentos deberían ser "ajustables" para evitar que se produzca el escenario del Caballo de Troya?
b) ¿Dónde habría que colocar una PLL para que todos los segmentos de código presentes en memoria pudieran acceder a CG0?
c) Explicar razonadamente por qué no tiene sentido colocar una PLL en el máximo nivel de privilegio

4.4. Ejercicio 4

Dado el esquema de memoria de la figura 25, responder a las siguientes preguntas:
a) ¿Existe algún error en el gráfico? En caso afirmativo, razonarlo.
b)¿ Desde qué segmentos de código se puede realizar una conmutación a la tarea T1?
c) ¿ Qué segmentos deberían hacerse "ajustables" para evitar que se produzca el escenario del caballo de Troya?
d) ¿Desde qué segmentos se puede acceder a C11?
e) ¿Qué segmentos pueden acceder a D12?
f) ¿Qué segmentos de la tarea T2 pueden acceder a C11?

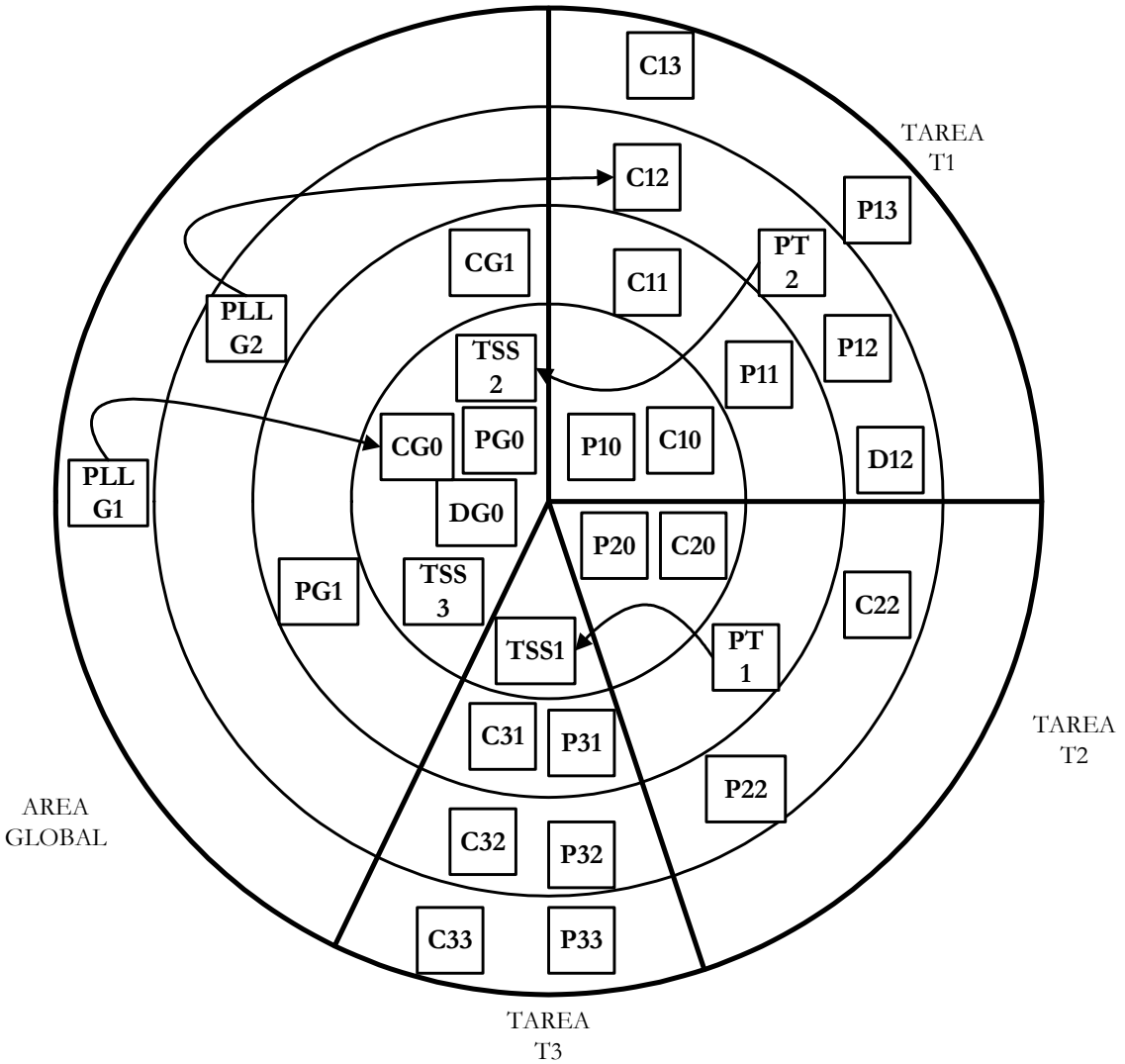


Figura 25 - Situación de memoria

5. TEMA 16: CICLOS DE BUS

5.1. Ejercicio 1

Dibujar y explicar el diagrama de un ciclo de bus simple de lectura de memoria no cacheable, con un solo estado T2, representando también la línea de comprobación de paridad.

Resolución

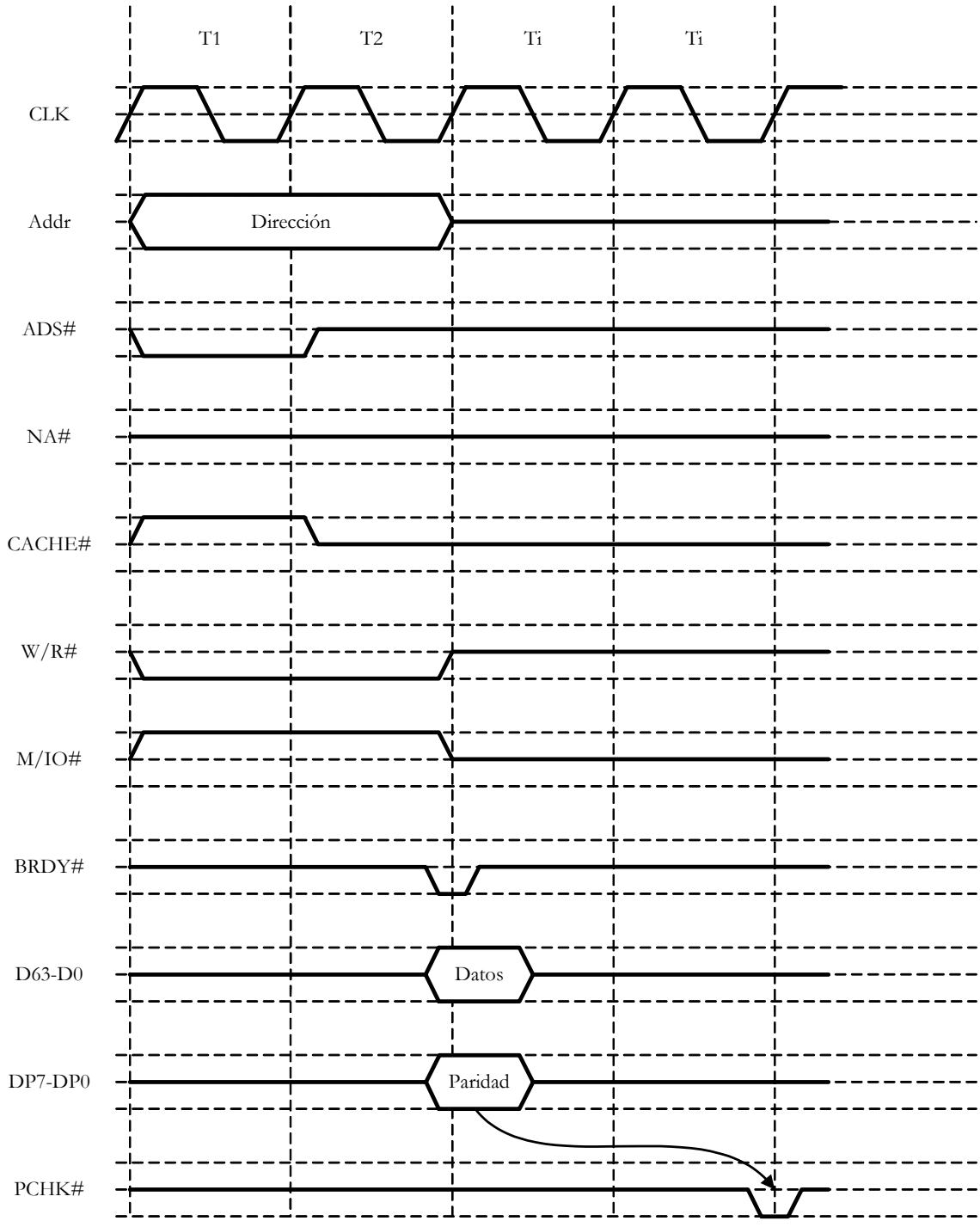


Figura 26- Ciclo de bus simple de lectura

La figura 26 muestra las líneas de control que definen el ciclo de bus. En el momento en que se activa ADS#, aparece en el bus de direcciones una dirección válida, se marca el ciclo como no cacheable poniendo a uno la línea CACHE#, se indica que es una lectura y que el acceso se realizará a memoria. Una vez completado el ciclo, la memoria activa la señal BRDY# para indicar que en el bus de datos ya están disponibles el dato pedido y su paridad. Dos ciclos de reloj después, el procesador activará PCHK# para indicar que ha realizado la comprobación de paridad.

5.2. Ejercicio 2

Dibujar y explicar el diagrama de un ciclo de bus a ráfagas de escritura, representando también la línea de comprobación de paridad

Resolución

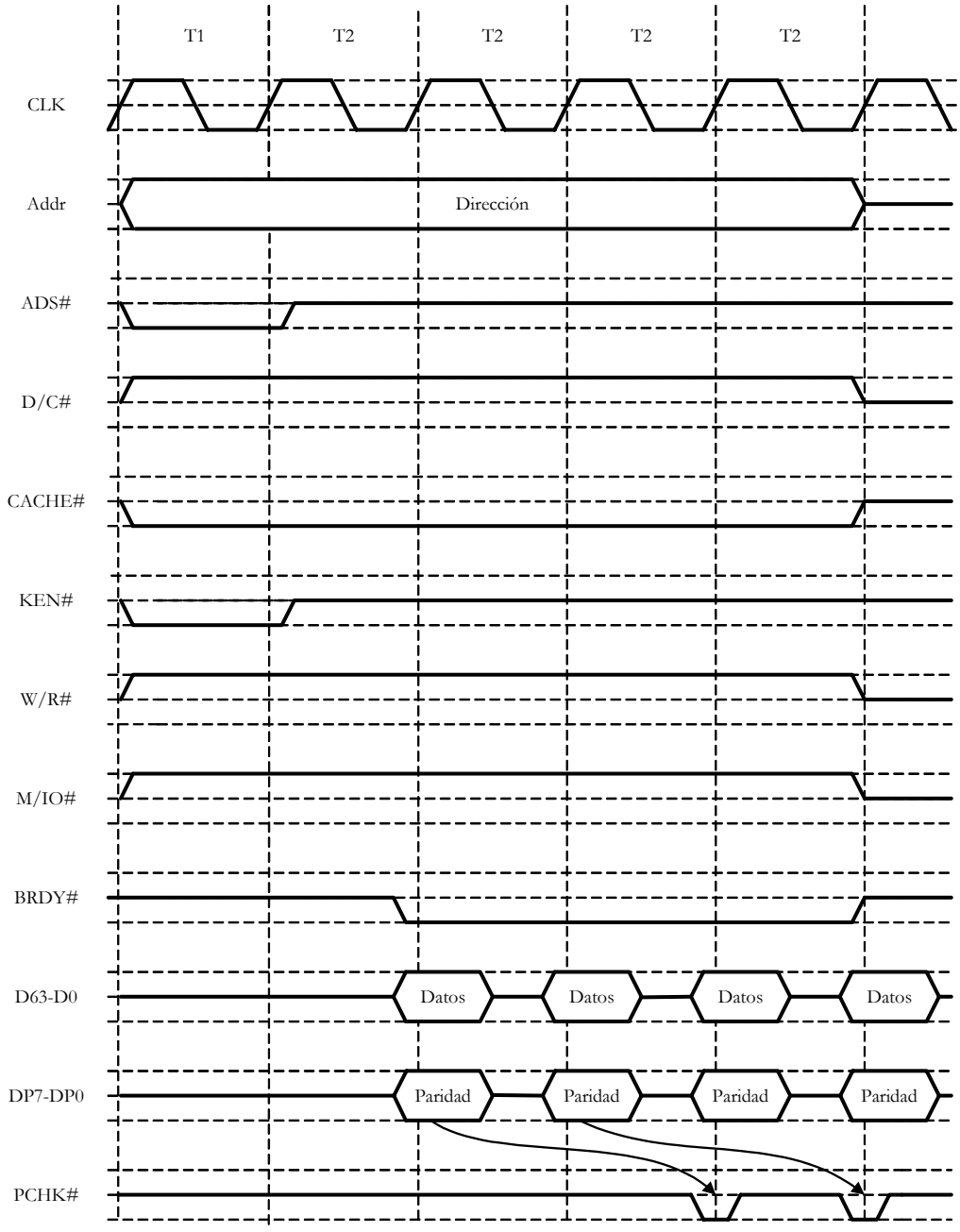


Figura 27 - Ciclo de bus a ráfagas

La figura 27 muestra las líneas de control que definen el ciclo de bus. A diferencia de un ciclo normal, uno a ráfagas consta de cinco estados, un T1 y cuatro T2 a continuación. En cada uno de ellos se genera un conjunto de datos con su paridad asociada, y la señal PCHK# se va generando dos ciclos después de ser recogido cada dato.

Como se trata de un ciclo de caché, tanto la señal CACHE# como KEN# deben estar activas. Recuérdese que los intercambios con caché siempre son con memoria, nunca con E/S, por lo que M/IO# debe estar a uno.

5.3. Ejercicio 3

Dibujar y explicar el diagrama de un ciclo de bus simple de escritura de memoria no cacheable, con tres estados T2, representando también la línea de comprobación de paridad.

5.4. Ejercicio 4

Dibujar y explicar el diagrama de un ciclo de bus a ráfagas de escritura, representando también la línea de comprobación de paridad.