CAPÍTULO 1

EVOLUCIÓN DE LOS MICROPROCESADORES

1.1 PROYECTO DE LA UNIDAD DE CONTROL

1.1.1 Lógica al azar

A mediados de los años 70', la técnica preponderante en el diseño de los microprocesadores de 8 bits era la de "lógica al azar". Esta consistía en definir la arquitectura del procesador, lo cual incluye cantidad, tamaño y función de los registros, el conjunto de instrucciones y los servicios de entrada/salida. Luego, aplicando el álgebra de circuitos lógicos y las técnicas de diseño de circuitos combinacionales y secuenciales y algoritmos de minimización se obtenía el circuito electrónico que cumplía las especificaciones.

1.1.2 Microprogramación

La microprogramación, fue desarrollada por Maurice Wilkes en la década de 1950 mientras trabajaba en la Universidad de Cambridge. La primera implementación de la unidad de control microprogramada se realizó en el EDSAC 2.

El concepto de control microprogramado se popularizó rápidamente y se convirtió en un elemento fundamental en el diseño de muchos sistemas informáticos posteriores. Varias empresas adoptaron esta tecnología en sus computadoras, pero una de las primeras en hacerlo fue IBM, quien la utilizó en sus computadoras System/360, lanzadas en la década de 1960, lo que permitió una mayor flexibilidad y capacidad de actualización en comparación con los sistemas informáticos anteriores. Digital Equipment Corporation incorporó en 1980 a Wilkes a su equipo de ingeniería a los 67 años.

La microprogramación consiste en descomponer las instrucciones en operaciones elementales, llamadas microinstrucciones e implementar el hardware partiendo de una base común (véase la figura 1.1.1) donde todas las instrucciones utilizan los mismos circuitos.

1.1.3 Unidad de control micro programada

Se analizará el diagrama esquemático de la figura 1.1.1, que muestra una unidad de control micro programada, basada en un diseño de Digital Equipment Corporation (DEC PDP11/44). La línea de puntos divide el procesador en dos bloques con funciones bien diferenciadas: la lógica de control y el camino de los datos.

Lógica de control: está compuesta por la memoria de microprograma, el secuenciador y el registro de micro palabra.

Memoria de microprograma: es una memoria de alta velocidad que contiene la secuencia de operaciones elementales que componen cada instrucción. Es el núcleo del procesador, ya que de su contenido depende como responderá el mismo a los distintos códigos de instrucciones. La cantidad de locaciones oscila entre 500 y 4000 y el ancho de palabra entre 24 y 100 bits.

Secuenciador: es el encargado de direccionar la memoria de microprograma (control store) basándose en el contador de microprograma MPC (Micro Program Counter). El MPC es un registro que se incrementa con un reloj interno, pero que depende de los registros de estado e instrucción y de algunos bits del registro de micro palabra. Ante determinadas situaciones toman valores preestablecidos (RESET, INTERRUPT, HOLD, etc.)

Registro de estado: proviene de la unidad aritmética y lógica y refleja el estado (status) de los indicadores (flags) luego realizar una operación (CARRY, ZERO, OVERFLOW, etc.). Su contenido es tenido en cuenta por el secuenciador cuando la instrucción en curso es un salto condicional.

Registro de instrucción: es el código de operación de la instrucción actual. Interviene en la fijación del valor del contador de microprograma dependiendo de la Decodificación de la instrucción y del modo de direccionamiento.

Registro de micro palabra: retiene la microinstrucción que el secuenciador direccionó sobre la memoria de microprograma y asigna bits en posiciones preestablecidas para dos objetivos principales: el control del "camino de los datos" (data path) para habilitar multiplexores, buffers y seleccionar la operación lógica en la ALU y por otra parte el control del secuenciador en función de los bits de micro bifurcación, que intervienen en las instrucciones de salto condicional e interrupciones.

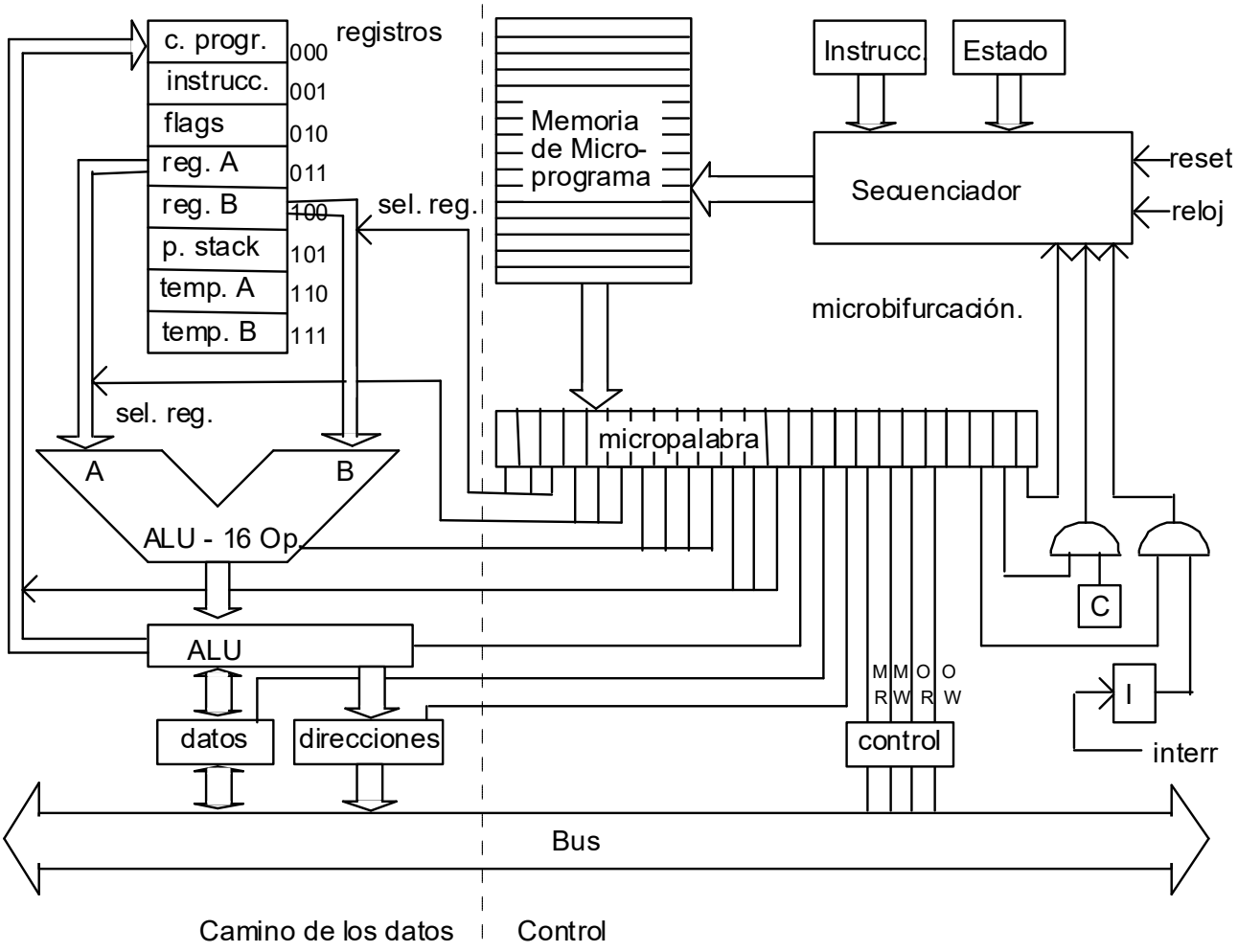


Fig. 1.1.1 - Unidad de control microprogramada

Camino de los datos: está compuesto por los registros de uso general, la unidad aritmética y lógica y la interfaz al bus.

Registros de uso general: incluyen los registros del modelo programable (los que ve el programador en assembly) y otros internos que intervienen como temporarios (temporary register) en operaciones más complejas, como por ejemplo la multiplicación.

Unidad aritmética y lógica: en ella reside la capacidad de procesador para efectuar operaciones lógicas elementales (AND, OR, INCREMENTO, COMPLEMENTO (/), NEGACION (-), etc.) sobre los datos que los multiplexores de registros colocan en cada una de sus "piernas", almacenando el resultado en otro registro. La cantidad de bits que la ALU puede manejar a la vez determina el ancho de palabra de la unidad de control que se está diseñando, por ejemplo, una ALU de 16 bits podrá sumar dos datos de 16 bits cada uno.

Interfaz al bus: con este registro, habilitado por la micro palabra, se representa en forma esquemática la interfaz entre el procesador y el mundo exterior y a partir de aquí se formarán los buses de datos, direcciones y control.

Para controlar el camino de los datos, los bits de la micro palabra están agrupados como indica la Tabla 1.1.1.

Dirección	*ALUB*	*ALUA*	*ALUFUN*	*ALUR*	*ALUCTL*	*BUSCTL*	*TEST*	*Total*
Bits	3	3	4	3	3	4	4	**24**

Tabla 1.1.1 – Organización de la memoria de microprograma (control store)

ALUB agrupa los 3 bits que seleccionan, a través de un multiplexor, el registro que se enviará a la rama B de la ALU. De la misma forma **ALUA** lo hace para la rama A.

ALUFUN selecciona la función lógica, según la codificación que muestra la Tabla 1.1.2.

ALUFUN	*Función*	*ALUFUN*	*Función*	*ALUFUN*	*Función*	*ALUFUN*	*Función*
0000	NOP	0100	A + B	1000	A XOR B	1100	A AND B
0001	A	0101	B	1001	- A	1101	A < B
0010	A=B	0110	A+1	1010	A-1	1110	SH IZQ A
0011	/A	0111	A-B	1011	A OR B	1111	SH DER A

Tabla 1.1.2 – Funciones de la ALU

ALUR controla el multiplexor que envía el resultado de la ALU o el registro de datos hacia uno de los 8 registros de uso general.

ALUCTL posee un bit que, estando en 0 habilita la escritura hacia arriba, al registro determinado por ALUR. En cambio, en 1 lo hace hacia abajo, al registro de datos. Los otros dos bits controlan la habilitación de los registros de datos y direcciones respectivamente.

BUSCTL agrupa los cuatro bits que habilitan la escritura o lectura de posiciones de memoria o E/S externas.

TEST son los bits de test para las bifurcaciones condicionadas por eventos externos (ej. Interrupciones), internos (ej. Excepciones), saltos condicionales o para indicar al secuenciador el fin de una secuencia de microinstrucciones, cuando el último bit está en 1.

Nótese que, ante una interrupción externa, aún en el caso en que el bit de habilitación de la palabra de estado lo permita, el microprograma, es decir, la secuencia de microinstrucciones que implementan una instrucción no será interrumpido. Por el contrario, la existencia o no de una interrupción externa será tratada como otro caso de bifurcación condicional mediante la selección de los bits apropiados del campo “Test”.

En los casos mencionados, la señal enviada al secuenciador tendrá dos estados posibles: **activado**, en el cual se modifica el contador de microprograma para que realice las operaciones

(ejecutar el salto condicional o la interrupción) y **desactivado**, en el que el contador no es afectado y por tanto se continúa la secuencia de instrucciones.

Ejemplo de microprogramación – Instrucción INC A:

La dirección de memoria 82h contiene 02h, que corresponde a incrementar el reg. A

El valor inicial del reg. A es 20h y el PC contiene 82h.

El microprograma debe hacer lo siguiente:

1- Decodificarla y ejecutarla (la instrucción fue traída por la instrucción previa)

2- Incrementar el PC

3- Traer la próxima instrucción de la memoria

Para eso, hay que armar una tabla que representa el contenido de la memoria de microprograma (MP):

Dir. MP	*ALU B*	*ALU A*	*ALU FUN*	*ALU R*	*ALU CTL*	*BUS CTL*	*TEST*	*Comentario*	*PC*	*ALU*	*DIR*	*A*
(*) 20	xxx	011	0110	111	000	0000	0000	Inc A y almac. en Temp B	82	21	82	20
21	xxx	111	0001	011	000	0000	0000	Almacena temp B en reg. A	82	21	82	21
22	xxx	000	0110	111	000	0000	0000	Inc PC y pone en temp B	82	82	82	21
23	xxx	111	0001	000	000	0000	0000	Poner temp B en PC	83	83	82	21
24	xxx	000	0001	xxx	101	0000	0000	Poner PC en bus direcc	83	83	83	21
25	xxx	000	0001	001	000	1000	0000	Lee bus de datos y pone en reg Instrucción				
26	xxx	xxx	0000	111	000	0000	0001	Decodifica la próxima instr. Fin de secuencia	83	xx	83	21

Tabla 1.1.3 – Ejemplo de microprogramación

(*) Cuando 02 pasa al registro de instrucción, el contador de microprograma pasa a contener 020, es decir la primera microinstrucción de la secuencia que incrementa A.

Para la realización de los trabajos prácticos de microprogramación, se desarrolló el simulador **UC1-TD3**, que se puede descargar de la página web de la materia. El mismo se implementó mediante un software de diseño de circuitos digitales, llamado Multimedia Logic

1.6.1, de Softronics Inc. El objetivo de diseño del circuito fue respetar lo más fielmente posible el funcionamiento de la unidad de control microprogramada explicada en clase.

1.1.4 Comparación de ambas técnicas

El gráfico de la figura 1.1.2, representa en abscisas la complejidad de un circuito (cantidad de instrucciones, modos de direccionamiento, ancho de palabra, etc.) y en ordenadas el costo de desarrollo y producción.

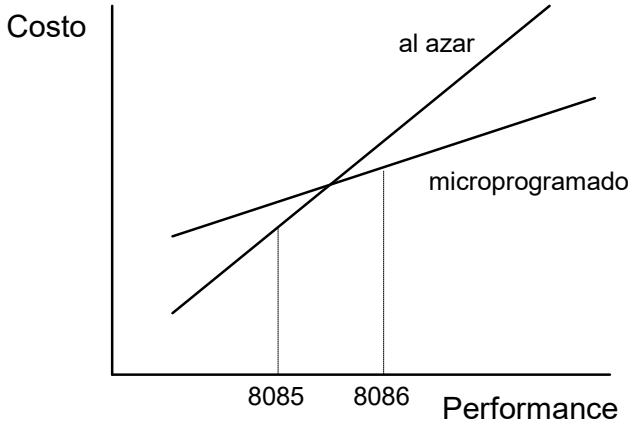


Fig. 1.1.2 – Comparación de ambas técnicas

Se observa que la pendiente que corresponde al micro programado es más suave, por lo tanto, a partir de cierta complejidad es más conveniente el diseño con esta técnica. Como ejemplo se indican los microprocesadores 8085 y 8086 de Intel.

1.2 PROCESAMIENTO PARALELO

1.2.1 Procesador microprogramado

En un procesador microprogramado genérico como el descripto en 1.1.3., la secuencia de operaciones de una instrucción típica puede representarse como sigue:

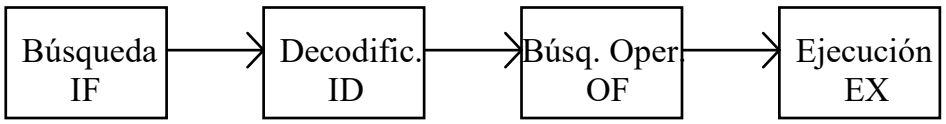


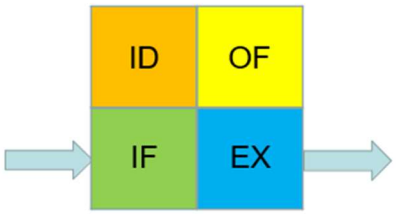
Fig. 1.2.1 – secuencia de operaciones

En un microprocesador de segunda generación como el 8085 de Intel, esta secuencia, expresada en función del tiempo y para un programa que contiene la sucesión de instrucciones I1, I2, I3, se vería así:

EX	.	.	.	I1	.	.	.	I2	.	.	.	I3
OF	.	.	I1	.	.	.	I2	.	.	.	I3	.
ID	.	I1	.	.	.	I2	.	.	.	I3	.	.
IF	I1	.	.	.	I2	.	.	.	I3	.	.	.
T	1	2	3	4	5	6	7	8	9	10	11	12

Fig. 1.2.2 – secuencia de operaciones – diagrama temporal

Si se diseña un procesador en el cual los circuitos que realizan la búsqueda de instrucciones (IF), la Decodificación de instrucciones (ID), la búsqueda de operandos (OF) y la ejecución (EX) no tienen elementos en común, es decir que pueden operar independientemente uno de otro, se obtiene un procesador paralelo de cuatro unidades.



1.2.2 Pipelining

El diagrama de tiempos, para la misma secuencia de instrucciones del caso precedente es como se muestra:

Esta técnica se denomina también "pipelining" porque asocia las cuatro fases de la ejecución de instrucciones a cuatro "caños" por donde fluyen las fases de cada instrucción.

EX	.	.	.	I1	I2	I3	I4	I5	I6	I7	I8	I9
OF	.	.	I1	I2	I3	I4	I5	I6	I7	I8	I9	.
ID	.	I1	I2	I3	I4	I5	I6	I7	I8	I9	.	.
IF	I1	I2	I3	I4	I5	I6	I7	I8	I9	.	.	.
T	1	2	3	4	5	6	7	8	9	10	11	12

Fig. 1.2.3 – secuencia de operaciones con pipeline – diagrama temporal

Cañería de ejecución
Cañería de búsqueda de operandos
Cañería de Decodificación de instrucción
Cañería de búsqueda de instrucción

Fig. 1.2.4 – pipeline de 4 etapas

La mejora en el tiempo de procesamiento, una vez que se llena la cañería, se calcula así:

El tiempo para ejecutar n instrucciones es:

Tk = k+n–1 (procesador paralelo de k unidades)

Ts = n.k (procesador tradicional)

Speedup = Ts / Tk = n.k / (k+n-1) (mejora en el tiempo)

Como n >> k … **Speedup ≈ k**

La mejora está dada por la cantidad de unidades que operan en paralelo.

K	.	.	.	I1	.	.	.	.	.	.	.	In
.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.
1	I1	.	.	Ik	.	.	.	.	.	.	.	.
T	1	.	.	.	.	.	.	.	.	.	.	k

Fig. 1.2.5 – mejora en la performance

Este análisis simplificado considera que los tiempos involucrados en cada una de las operaciones son los mismos, cosa que en un procesador tradicional no es exactamente cierta. Los tiempos de espera introducidos degradan la performance alejándola de la ideal.

Otro factor indeseable son las rupturas de secuencia (saltos) que invalidan todas las instrucciones buscadas con anticipación y obligan a reiniciar el ciclo de llenado de la cañería.

En la figura siguiente consideramos que I2 corresponde a la ejecución de un salto a I10, entonces todas las actividades resaltadas en rojo no son de utilidad y las resaltadas en azul serán canceladas. El hueco que se produce por esta situación se conoce como burbuja.

EX	.	.	.	I1	I2	I3	I4	I5	I10	I11	I12	I13
OF	.	.	I1	I2	I3	I4	I5	I10	I11	I12	I13	.
ID	.	I1	I2	I3	I4	I5	I10	I11	I12	I13	.	.
IF	I1	I2	I3	I4	I5	I10	I11	I12	I13	.	.	.
T	1	2	3	4	5	6	7	8	9	10	11	12

Fig. 1.2.6 – problema de la burbuja

En el procesador 80486 de Intel se distinguían 5 operaciones:

WB	.	.	.	.	I1	I2	I3	I4	I5	I6	I7	I8
EX	.	.	.	I1	I2	I3	I4	I5	I6	I7	I8	.
AG	.	.	I1	I2	I3	I4	I5	I6	I7	I8	.	.
ID	.	I1	I2	I3	I4	I5	I6	I7	I8	.	.	.
IF	I1	I2	I3	I4	I5	I6	I7	I8	.	.	.	.
T	1	2	3	4	5	6	7	8	9	10	11	12

Fig. 1.2.7 – secuencia de operaciones en el procesador 80486

Donde:

IF: Instruction fetch – Búsqueda de instrucción

ID: Instruction decode – de instrucción

AG: Address generation - Generación de direcciones

EX: Execution - Ejecución

WB: Write back – Escritura en la caché interna

En el procesador Pentium se incluyó una segunda unidad de ejecución, por lo que pudo manejar en paralelo dos instrucciones, quedando el diagrama de la figura 1.2.7.

Donde:

IF1: Instruction fetch del procesador 1

IF2: Instruction fetch del procesador 2

WB2	.	.	.	.	I2	I4	I6	I8	I10	I12	I14	I16
EX2	.	.	.	I2	I4	I6	I8	I10	I12	I14	I16	.
AG2	.	.	I2	I4	I6	I8	I10	I12	I14	I16	.	.
ID2	.	I2	I4	I6	I8	I10	I12	I14	I16	.	.	.
IF2	I2	I4	I6	I8	I10	I12	I14	I16	.	.	.	.
WB1	.	.	.	.	I1	I3	I5	I7	I9	I11	I13	I15
EX1	.	.	.	I1	I3	I5	I7	I9	I11	I13	I15	.
AG1	.	.	I1	I3	I5	I7	I9	I11	I13	I15	.	.
ID1	.	I1	I3	I5	I7	I9	I11	I13	I15	.	.	.
IF1	I1	I3	I5	I7	I9	I11	I13	I15	.	.	.	.
T	1	2	3	4	5	6	7	8	9	10	11	12

Fig. 1.2.8 – secuencia de operaciones en el procesador Pentium

1.3 PROCESADORES RISC

Los procesadores RISC (Reduced Instruction Set Computer, o Computadora de Conjunto de Instrucciones Reducido) fueron diseñados para sacar provecho del procesamiento paralelo.

El conjunto de instrucciones se redujo a las operaciones elementales, con el objeto de que cada una de ellas pueda ejecutarse en un solo ciclo de reloj.

Dado que este diseño requiere menos transistores que un procesador tradicional, la superficie de chip que sobra se destina a registros (poseen muchos) y a memoria de alta velocidad (caché). Esto reduce la necesidad de acceder al bus para buscar operandos y registros intermedios aumentando la performance.

Como desventaja, puede citarse que los programas (el código objeto más precisamente) requieren más espacio por tener instrucciones menos potentes. En cuanto a los compiladores, se exige que sean optimizados para el procesador en cuestión.

Para evitar el problema de las burbujas, es conveniente el uso de lenguajes de programación estructurados, que reducen la cantidad de saltos.

1.3.1 Procesadores actuales

Debido a que en la actualidad es posible integrar miles de millones de transistores en el mismo chip, los pros de ambas técnicas se pueden aprovechar para lograr procesadores RISC con alto paralelismo, que también ejecutan instrucciones CISC, las que favorecen la programación y son compatibles con las CPU de generación anterior.

Predicción de trayectoria

A partir del Pentium, los procesadores tienen una función denominada “predicción de trayectoria”. La misma consiste en establecer un “mapa” de los lugares a donde es posible que se dirija la instrucción para adelantar la ejecución próxima. El método se basa en un buffer de decisión de destino (BTB), que incluye tres elementos por cada entrada: la dirección de la instrucción de salto, la dirección de destino de la instrucción y los bits de historia. Se usa una tabla de hasta 256 entradas para predecir los resultados de las decisiones. Los bits de historia indican si la decisión se tomó o no. Esta mejora tiende a evitar las burbujas que se producen ante saltos condicionales e Intel declara que esta característica incrementa el rendimiento en un 25%.

Hyper-Threading

Es una marca registrada de la empresa Intel para promover la implementación de la tecnología multihilo simultáneo, también conocido como SMT. Permite que los programas que estén preparados para ello ejecuten tareas usando múltiples hilos, lo cual es un procesamiento en paralelo dentro de un único procesador, incrementando así el uso de las unidades de ejecución (ALUs, FPs, etc.) del procesador.

La tecnología Hyper-Threading consiste en simular dos procesadores lógicos dentro de un único procesador físico. El resultado es una mejora en el rendimiento del procesador, puesto que al simular dos procesadores se pueden aprovechar mejor las unidades de cálculo manteniéndolas ocupadas durante un porcentaje mayor de tiempo. Esto conlleva una mejora en la velocidad de las aplicaciones que según Intel es aproximadamente de un 60 %.

El equivalente de AMD se denomina SMT.

Ref: Hyper-threading - Wikipedia

En el apéndice E se detallan algunas de las características principales de los procesadores de Intel, AMD e IBM, desde los primeros chips hasta la actualidad.

1.4 ORGANIZACION DE MEMORIAS

1.4.1 Paginación

En este ejemplo, se explicará la técnica que utilizaban los procesadores DIGITAL PDP-11 para direccionar más de 64 kB con registros de 16 bits.

Los tres primeros bits de la dirección virtual se utilizan para determinar cuál será el registro de página (PAR) seleccionado, el cual se sumará con un desplazamiento de seis bits a la dirección virtual para obtener un resultado de 22 bits (dirección física).

De esta forma, un procesador que sólo contaba con registros de 16 bits podía direccionar 4 MB de memoria física, aunque solamente 64 kB a la vez, en “páginas” de 8 kB.

Cambiando el contenido de un PAR se podía acceder a otra parte de la memoria física y así se facilitaba la conmutación de tareas en sistemas multitarea.

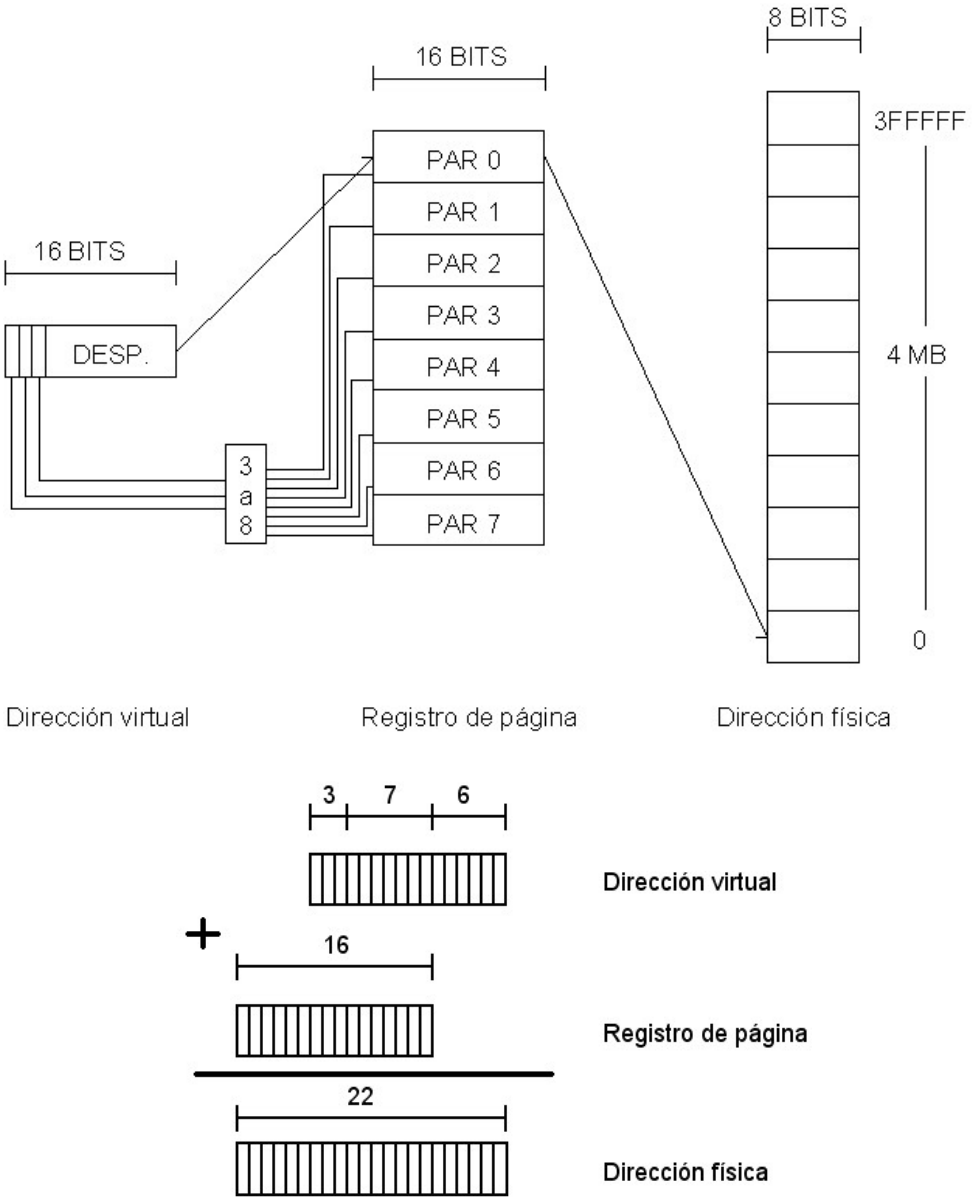


Fig. 1.4.1 - Paginación

1.4.2 Segmentación

En este ejemplo, se explicará la técnica que utilizaban los procesadores Intel 8086/80286 para direccionar más de 64 kB con registros de 16 bits.

Los registros de segmento eran CS (code), DS (data), ES (extra) y SS (stack).

La forma de sumar los registros para obtener la dirección física era la misma que en el caso de la paginación. La diferencia aquí es el modo de seleccionar el registro de segmento, que consiste en determinar el tipo de operación a realizar. Por ejemplo, si se trata de una instrucción MOV, se usará el registro DS, si es un JMP, el registro CS, si es PUSH, el registro SS.

Con un prefijo, el 8086 podía forzar un registro en particular.

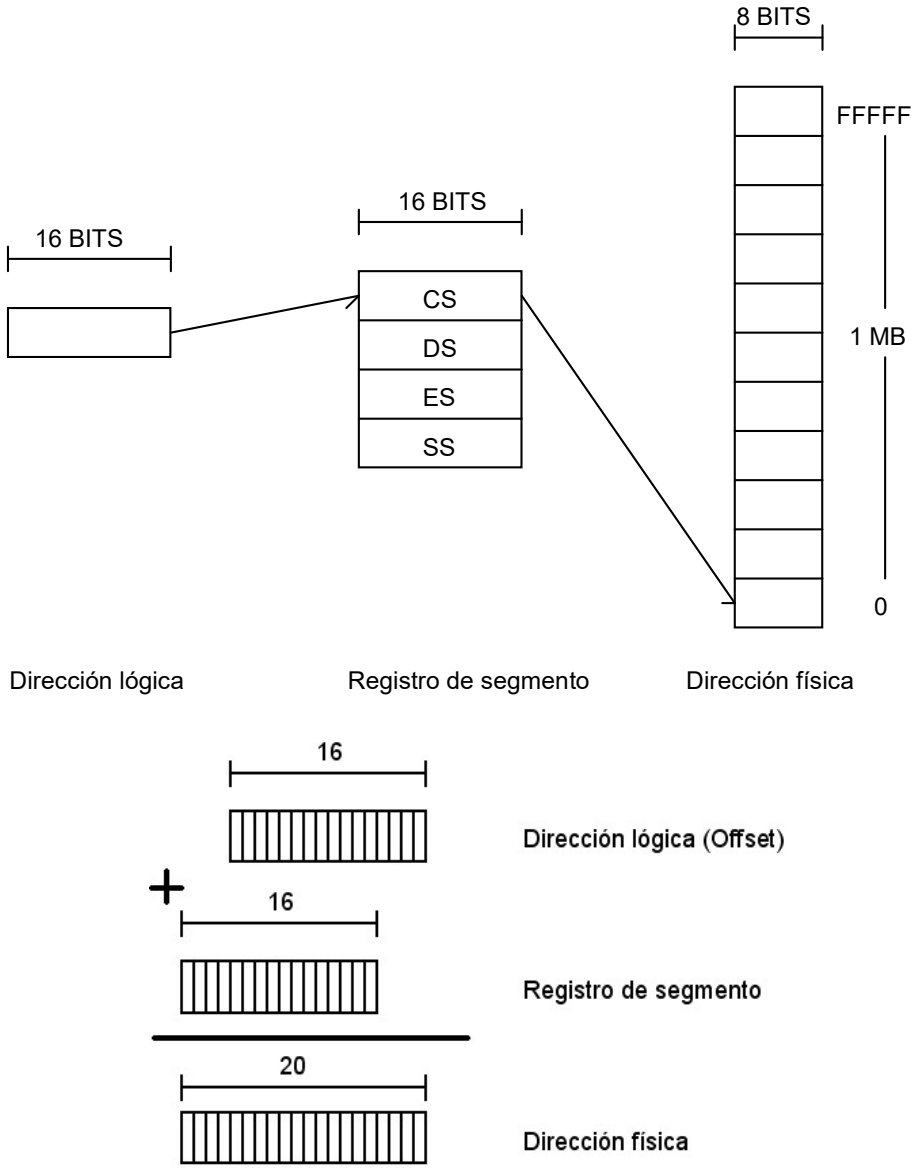


Fig. 1.4.2 - Segmentación

Aplicando segmentación (S), paginación (P) o ambas se logra:

- Aumentar el espacio de direcciones físico con registros existentes (S)(P)
- Aprovechar el espacio de direcciones virtual e implementar memoria virtual (P)

- Establecer protecciones a espacios de memoria (P)
- Aumentar el rendimiento de la cache separando en código y datos (S)
- Ejecutar código compartido entre programas (S)(P)
- Ejecutar rutinas reentrantes con distintos datos (S)(P)
- Reubicar código con facilidad (S)
- Favorecer el cambio de contexto en sistemas multitarea (S)(P)

1.5 MEMORIAS CACHE

1.5.1 Introducción

La demanda permanente de performance en las computadoras exigió siempre mejoras en cada uno de sus componentes: procesador, memoria y entrada/salida. En lo que atañe a las memorias, la reducción del precio por byte y la creciente densidad de integración permitieron aumentar la complejidad de los programas, pero como contra-partida, se requería mayor velocidad de ejecución para mantener la interactividad con el usuario.

El tiempo que demandan la búsqueda de instrucciones, operandos y el almacenamiento de resultados en memoria dependen esencialmente del *tiempo de acceso* de las memorias. A mediados de la década de los 70', existían las memorias bipolares, de baja densidad de integración, alta disipación, alto costo por bit y bajo tiempo de acceso. Estas memorias eran utilizadas principalmente para registros. También existían las memorias MOS, de mayor densidad de integración, baja disipación, bajo costo por bit y elevado tiempo de acceso.

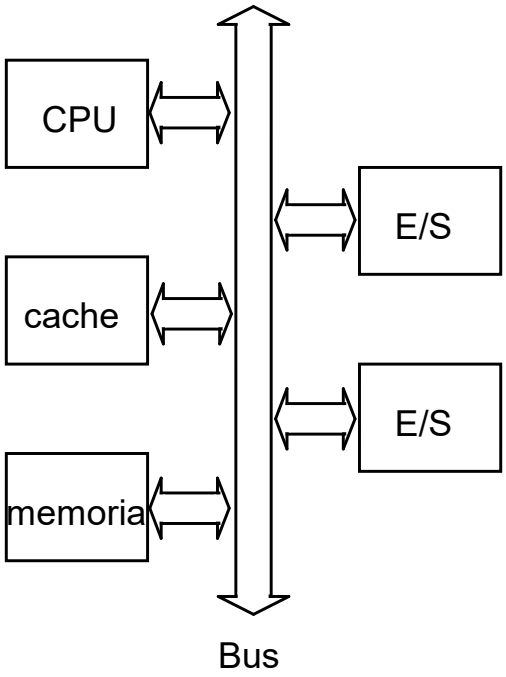


Fig. 1.5.1 - Diagrama en bloques

Esta situación obligó a los diseñadores de computadoras, ya en la década de los 70, a pensar en algo para resolver la disyuntiva: memoria veloz, pero de elevado costo (bipolar) o baja velocidad con mayor capacidad (MOS). En la figura 1.5.1 se puede observar la disposición circuital original de la memoria cache.

Las características de esta memoria eran las siguientes: tecnología bipolar (estática), capacidad 8 kB y tiempo de acceso 70 ns. La memoria convencional, de tecnología MOS (dinámica) tenía una capacidad de 4 MB y tiempo de acceso 600 ns.

1.5.2 Configuración básica

El circuito opera de la siguiente forma. Tanto la memoria cache como la convencional están conectadas al bus del sistema y por lo tanto ambas reciben la dirección solicitada por la CPU. Esta configuración se denomina: memoria curiosa.

Al principio, cuando la memoria cache está vacía, el dato pedido por el procesador será suministrado por la memoria convencional, en 600 ns, pero la memoria cache guarda tanto el dato como la dirección y un bit adicional que indica la validez del dato (véase la organización interna de la memoria cache en la figura 1.5.2).

Valid.	Dirección	Dato

Fig. 1.5.2 - Organización interna

La próxima vez que la CPU requiera el contenido de esa posición de memoria, la memoria cache podrá suministrarlo en 70 ns. Si se da el caso, como frecuentemente ocurre durante la ejecución de un programa real, que el acceso a esa posición de memoria hace falta muchas veces más, por ejemplo, en una iteración, todas las lecturas posteriores demandarán 70 ns. Si las iteraciones fueron 100, el tiempo promedio será:

$$Tp = (600 + 70 \times 99) / 100 = 75 \text{ ns}$$

Y la mejora en el tiempo de ejecución será:

$$Sp = 600 \text{ ns} / 75 \text{ ns} = 8$$

En estas condiciones la computadora se desempeña con una velocidad 8 veces mayor, casi como si toda su memoria fuese de 70 ns. Esto es válido sólo si el programa que itera cabe completamente en la memoria cache. El tamaño de la cache debe estar relacionado con el de las rutinas o programas que se pretende acelerar. El software más moderno, de mayores dimensiones, lógicamente requiere tamaños de cache superiores a los mencionados anteriormente (por ejemplo: es común usar varios MB en la actualidad).

Hace muchos años que no se usan las memorias bipolares. En la actualidad las memorias son todas MOS tanto la principal como la cache. Pero dado que la primera es dinámica para lograr alta capacidad, usando un solo transistor y un capacitor, la segunda es estática, usa 6 transistores sin capacitor y por tanto es mucho más rápida, aunque más costosa por bit.

1.5.3 Configuración avanzada

A medida que el tiempo de acceso de las memorias disminuía, la configuración circuital de la figura 1.5.1 tropezó con un inconveniente físico: la línea de transmisión que constituye el bus del sistema.

Para reducir los efectos de la capacidad e inductancia de las conexiones, se diseñó un bus privado, sólo para la memoria principal y la cache, que además *duplica* el número de líneas de datos (véase la figura 1.5.3). En 1996 la memoria principal de una PC de oficina tenía un tiempo de acceso de 60 ns y la memoria cache uno de 15 ns.

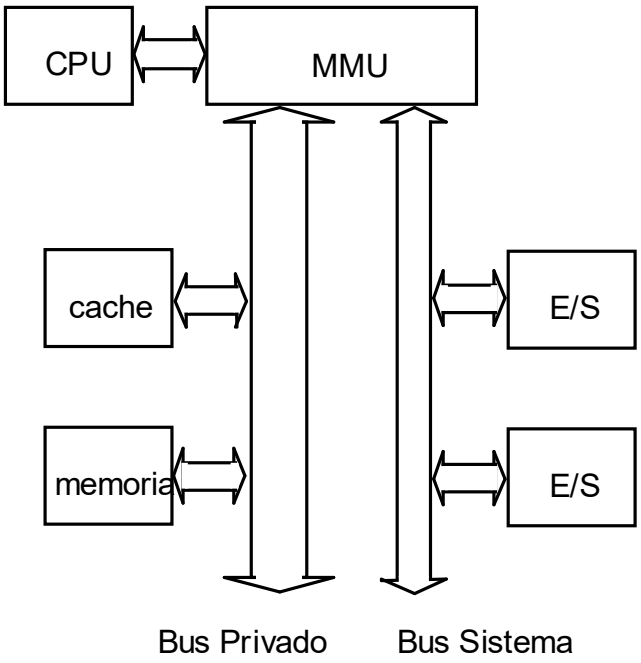


Fig. 1.5.3 - Bus privado

En los procesadores iAPX486 de (1989), Pentium (1992), Pentium Pro (1995) y posteriores, la unidad de manejo de memoria (MMU), la memoria cache y el bus privado se encuentran alojados en el chip del procesador, incrementando notablemente la performance.

Actualmente las caches se organizan en distintos niveles y además se distinguen según estén asociadas a datos o direcciones. Esto último facilita el acceso a los pipelines de búsqueda. Las más próximas al procesador se denominan L1 (Level 1: Nivel 1) y si existe otra a continuación, como en los procesadores actuales, se denomina L2 (Level 2: Nivel 2). Eventualmente puede existir un nivel L3 en los procesadores de múltiples núcleos.

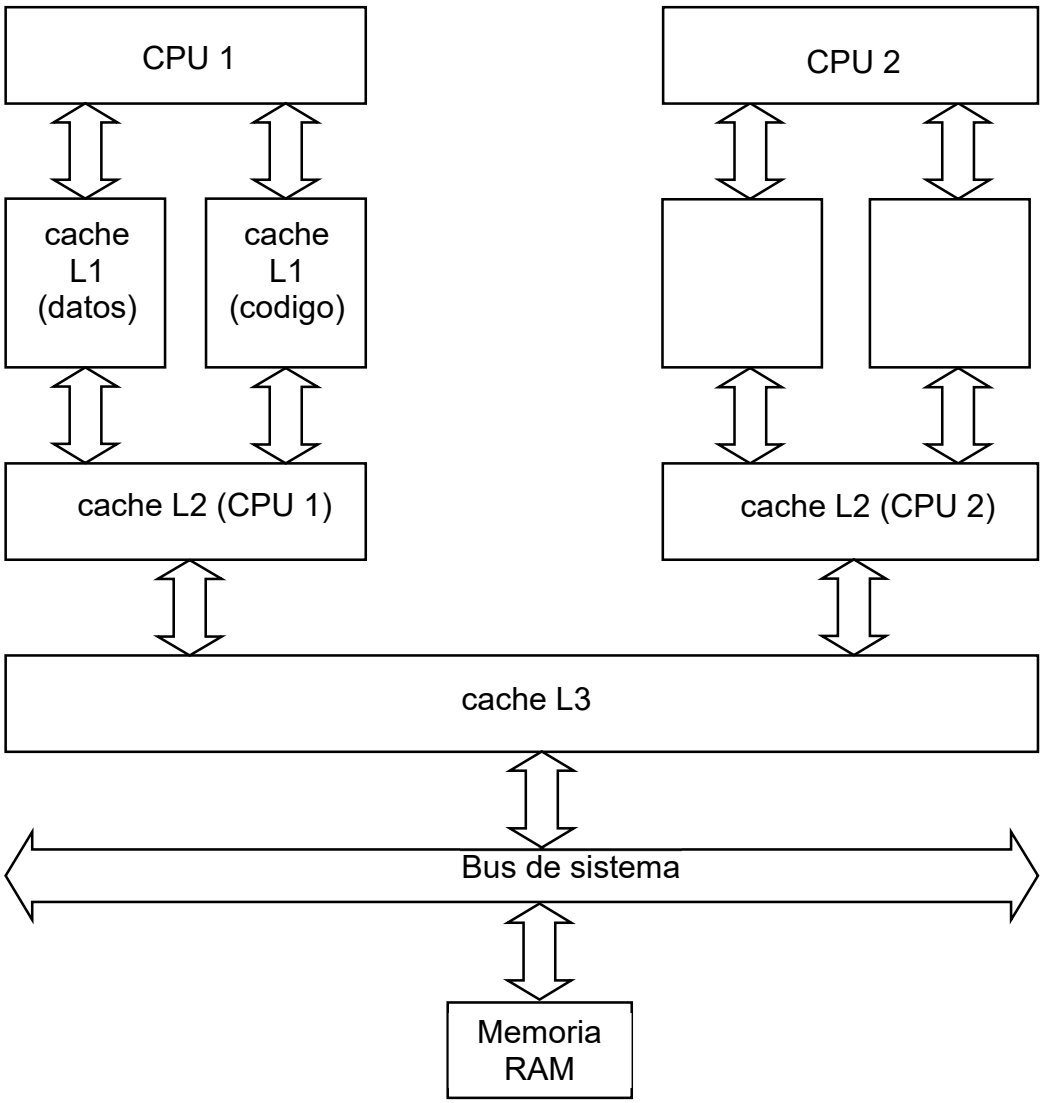


Fig. 1.5.4 – Niveles de cache

El Core I7 de Intel tiene 4 núcleos, cada uno de los cuales posee 32 kB de cache L1 para datos y 32 kB para código, 256 kB de cache L2 por núcleo y 8000 kB de cache L3 compartida por los 4 núcleos. Esto suma: 8x32 + 4x256 + 8000 = 9280 kB. Esta cantidad de cache utiliza más de 450 millones de transistores, de los 731 millones que posee este procesador.

El Phenom de AMD tiene también 4 núcleos, cada uno de los cuales posee 64 kB de cache L1 para datos y 64 kB para código, 512 kB de cache L2 por núcleo y 2000 kB de cache L3 compartida por los 4 núcleos. Esto suma: 8x64 + 4x512 + 2000 = 4560 kB. Esta cantidad de cache utiliza 224 millones de transistores, de los 450 millones que posee este procesador.

Los procesadores actuales de Intel pueden tener hasta 18 núcleos y 40 MB de caché L3 y los de AMD pueden tener hasta 64 núcleos y 256 MB de caché L3.

En el apéndice E se pueden ver las características de los principales procesadores Intel, AMD e IBM.

1.6 SISTEMAS MULTIUSUARIO

1.6.1 Sistema multitarea

Es un sistema que da la sensación de ejecutar varias tareas (programas) al mismo tiempo. Requiere una implementación de hardware: sistema de interrupciones y una de software: sistema operativo multitarea.

Las tareas se ejecutan basándose en un sistema de prioridades, donde la tarea más prioritaria se ejecuta hasta que requiera un recurso no disponible o hasta que finalice. En ese momento se transfiere el control a la que sigue y así sucesivamente hasta agotar la lista. Como todos los programas requieren de algún acceso a disco, impresora, pantalla, etc. y estos periféricos son más lentos que la CPU, queda tiempo de máquina durante el cual avanza la ejecución de las tareas menos prioritarias. Este esquema de planificación (scheduling) se conoce como "FCFS" (First Come, First Served).

En los procesadores actuales, para asegurar que se respeten las prioridades con una asignación de tiempo de CPU proporcional a la prioridad establecida, el RTC (reloj de tiempo real) genera una interrupción no enmascarable (NMI) cada 1 ms lo que obliga al sistema operativo a evaluar la lista de programas en espera de ejecución y los tiempos que les va asignando. En la figura 1.6.1 se ve la asignación de tiempos para 3 programas (#A #B #C) cuyas prioridades fueron fijadas en 5, 2 y 1 respectivamente. En Windows las prioridades tienen un rango de 1 a 255, siendo 255 la más alta.

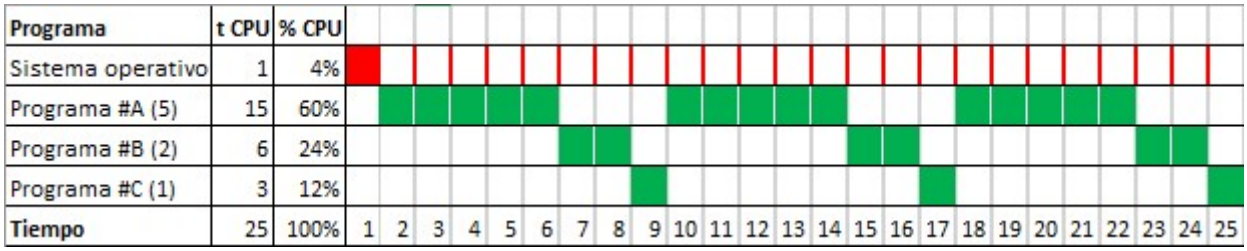


Fig. 1.6.1 – Sistema multitarea – diagrama de tiempos

Un algoritmo de planificación más avanzado es el denominado "round-robin", donde se agrega al esquema anterior la posibilidad de cambiar de tarea luego de agotado un tiempo preestablecido, denominado "time quantum" (entre 10 y 100 ms). En este caso, se programa un temporizador, que al cumplir el tiempo fijado genera una interrupción de HW, la que restituye el control al sistema operativo. La tarea en curso es enviada al final de la lista de pendientes de ejecución, pasándose el control a la primera de la lista. Si la tarea se ejecuta dentro del tiempo fijado, se sigue usando FCFS.

1.6.2 Sistema multiusuario

Es un sistema en el cual los usuarios tienen la sensación de disponer la computadora para ellos, pero en realidad el SO administra los recursos y los usuarios compiten por ellos. En estos sistemas, se asignan a los usuarios privilegios para tener acceso a los recursos.

El administrador del sistema es un usuario con los máximos privilegios. Él tiene la función de asignar a los usuarios los recursos del sistema (cuentas, espacio en disco, tiempo de CPU, dispositivos de impresión, mensajes, nivel de seguridad, recursos de software, etc.).

Para poder ser multiusuario, el sistema debe ser multitarea. Además, requiere una implementación en el hardware: los modos protegido y usuario/supervisor y en el software: el sistema operativo multiusuario.

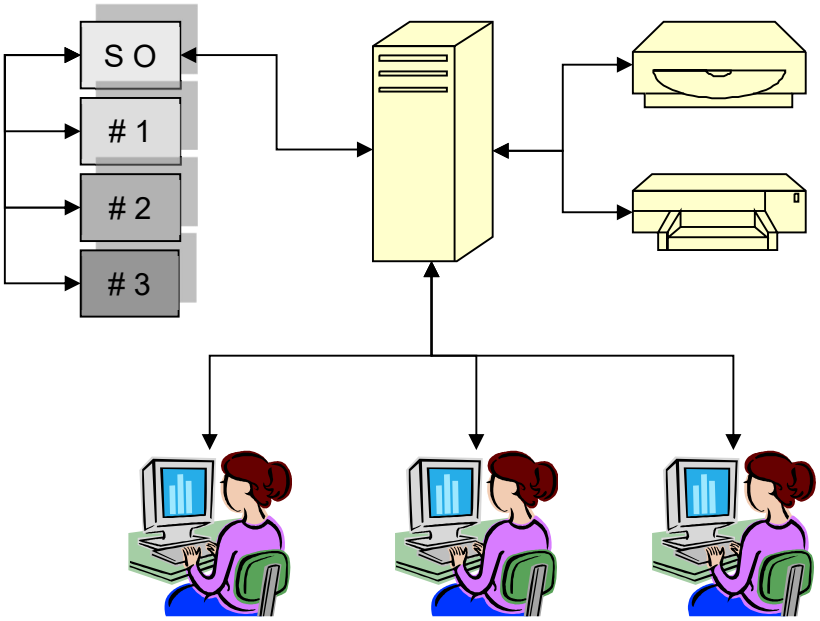


Fig. 1.6.2 – Sistema multiusuario

El sistema debe tomar los recaudos para que un usuario no afecte (intencionalmente o no) a otros usuarios o a la integridad del sistema. Por ejemplo, si un usuario escribe un programa en assembly que incluya la instrucción HLT (Halt Processor), hasta el momento de la ejecución nadie se daría cuenta de lo que pretende hacer. Si esta instrucción se ejecutase, se detendría la CPU y todos los usuarios se verían perjudicados.

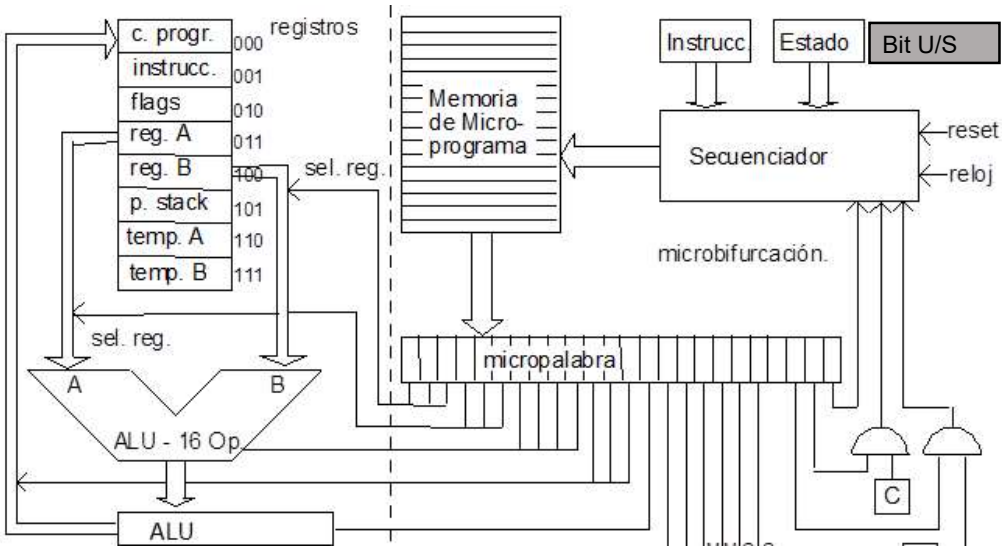
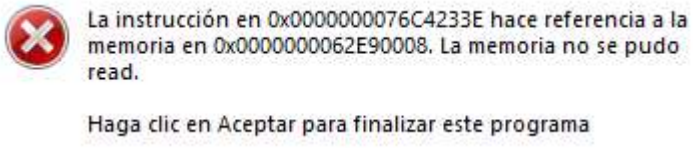


Fig. 1.6.3 – Modo U/S en la arquitectura microprogramada

En un procesador de estas características, la CPU toma las instrucciones de memoria de la manera habitual, las descodifica y en caso de ser HLT, se fija en el bit de usuario/supervisor del estado del procesador (PSW). Si está en supervisor, ejecuta HLT de la manera normal, deteniendo la CPU. Si está en modo usuario, genera una excepción de instrucción inválida, conmuta la CPU a modo supervisor y da la posibilidad al SO de suspender la tarea en cuestión sin que los demás usuarios se percaten de lo que ha ocurrido. Otras instrucciones que generan esta excepción son: RESET y modificación del PSW.

Para prevenir la situación anterior, el hardware provee dos modos de trabajo, usuario y supervisor y se asigna un bit en la palabra de estado del procesador PSW para indicar en qué modo se encuentra la CPU en todo momento. Los programas de usuario se ejecutarán en este modo y el SO en el modo supervisor.

Por último, en los sistemas multiusuario existen registros que determinan los derechos de acceso a las zonas de memoria. Esta implementación de hardware impide a un usuario escribir en la memoria de otro o en la del SO, generando una excepción de dirección inválida y permitiendo al SO suspender la tarea transgresora, como muestra esta alerta de Windows.



Los archivos en medios magnéticos están protegidos por software, pero un usuario no puede cambiar los derechos de acceso de los archivos de otro y por ende si el otro determinó que no puedan ser leídos por terceros, no lo serán.

De la misma forma, el SO maneja el servicio de impresión, para evitar que se mezclen las páginas que desean imprimir los distintos usuarios.

A continuación, se puede ver una tabla que resume lo visto precedentemente.

Recurso compartido	Implementación de HW	Implementación de SW	8086	80386
	¿Se requiere un HW adicional para administrar los permisos de acceso al recurso? ¿Cuál?	¿Cómo implementa el Sistema operativo los permisos de acceso al recurso?	¿se puede administrar completamente el recurso con este procesador?	¿se puede administrar completamente el recurso con este procesador?
Disco rígido	no, es suficiente con el sist. de interrupciones	con el administrador de archivos	si	si
Impresora	no, es suficiente con el sist. de interrupciones	con el administrador de impresión	si	si
Red	no, es suficiente con el sist. de interrupciones	con el administrador de red	si	si
Memoria	sí, el modo protegido	Sistema operativo multiusuario	no (no tiene MP)	si
Procesador	sí, el modo usuario/supervisor	Sistema operativo multiusuario	no (no tiene U/S)	si

Fig. 1.6.4 – Resumen de requisitos de HW y SW

1.7 MODO PROTEGIDO

1.7.1 Microprocesador provisional

Cuando Intel presentó por primera vez el 8086, este fue conocido internamente como "interim microprocessor" (la CPU que usarían los usuarios de 8080, 8085 y Z80 para actualizarse hasta tanto el iAPX 432, de 32 bits estuviese listo para la venta. Así, la primera consideración de diseño del 8086 fue que los programas escritos para los viejos microprocesadores de 8 bits fueran fáciles de transportar al 8086. Los diseñadores dijeron que tanto los juegos de registros como el conjunto de instrucciones permitían traducir *mecánicamente* las aplicaciones de 8080 en aplicaciones para el nuevo microprocesador.

Los pares de registros, que podían ser manipulados como una unidad en la arquitectura anterior, ahora eran de 16 bits de ancho. Ya que el 8086 tenía 20 bits de direcciones, las aplicaciones transportadas deberían haberse tenido que rescribir extensivamente si los cálculos de direcciones se hubiesen hecho en registros mayores. Esta es una de las razones de por qué Intel eligió dividir las direcciones en dos partes: un *registro de segmento* de 16 bits y un *desplazamiento de dirección efectivo* (o dirección lógica).

Como se vio en el ejemplo del capítulo 1.4, el 8086 tiene cuatro registros de segmento, los cuales fueron usados para mapear las direcciones de 16 bits utilizadas por la pila, datos y código del programa en cuatro regiones potencialmente disjuntas en el espacio de direcciones de 1 Mbyte.

El registro de segmento es desplazado cuatro bits a la izquierda y sumado a la dirección lógica para completar la dirección física de 20 bits.

Aunque esta arquitectura no hace fácil manipular estructuras de datos mayores que 64K bytes, poca gente tuvo esa cantidad de RAM en esa época.

Fue fácil para muchos programas (y virtualmente todas las aplicaciones transportadas desde CP/M) fijar los registros de segmento por única vez al principio del programa y olvidarse de ellos. Podría decirse que el contenido de los registros de segmento provee valores por defecto para los bits superiores de cada dirección. Desde entonces, el programa trató principalmente con desplazamientos de dirección efectivos de 16 bits, la mitad de tamaño de las direcciones manejadas por el 68000. El conjunto de instrucciones era también más compacto y no requería que las instrucciones o datos estuviesen alineados en el límite de palabras (de 16 bits). Como resultado, el código y los datos inmediatos demandaron cerca del 30% menos espacio. En aquellos días, la RAM era costosa, una buena razón para IBM para elegir la arquitectura Intel sobre la de MOTOROLA para la IBM PC.

1.7.2 El 80286: Primer Intento de Modo Protegido

Desafortunadamente, el iAPX 432, un microprocesador de 32 bits en dos chips diseñado para multiprocesamiento, multitarea, programación orientada a objetos y tolerancia a fallas de hardware y software, no fue un éxito en el mercado. Esto puede haber sido debido a que fue demasiado adelantado para su tiempo, o debido a que fue concebido para correr ADA (el cual no logró la penetración en el mercado que Intel pensó que tendría). O tal vez, simplemente, los chips fueron demasiado costosos.

Pero el IBM PC y el 8088 realmente tuvieron éxito y los usuarios clamaron por más memoria y por más potencia de procesamiento para correr su software existente. Con el 8086 Intel les dio todo eso y más. Tal como el resto de este artículo mostrará, el 80286 incluyó muchas de las características implementadas originalmente para el iAPX 432.

Dos características son esenciales para cualquier sistema que provea protección razonable contra programas errantes y usuarios maliciosos: *aislación de tareas* y *protección de los recursos del sistema*. El sistema necesita prevenir que las tareas corrompan recíprocamente sus memorias o acaparen el sistema. Los recursos del sistema (p/ej. hardware y sistema de archivos) requieren permanecer bajo el control exclusivo del software del sistema operativo. Puesto que la mayoría de los recursos son accesibles tanto a través de direcciones de memoria como de E/S, el sistema debe mantener control sobre la manera en que una tarea accede a ellos.

1.7.3 Direccionamiento Virtual

El 80286, luego de una condición de Reset, arranca en modo real. Una rutina de inicialización, la cual se ejecuta en modo real, define *tablas de descriptores*, que gobiernan como es direccionada la memoria y luego conmutan la CPU al modo protegido fijando un bit en un registro de control. Intel hizo esta transición casi irreversible en el 80286. El 80386, sin embargo, puede ser retrotraído al modo real reponiendo el mismo bit. Esto permite a la sesión en modo DOS del OS2 conmutar modos literalmente sin detener el microprocesador.

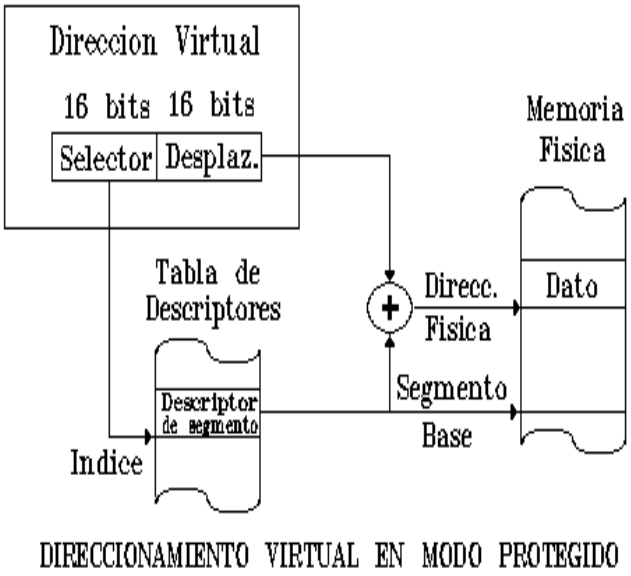


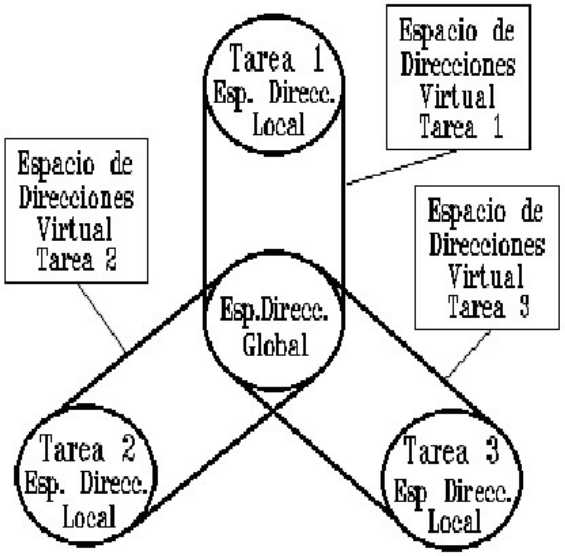
Fig. 1.7.1

En modo protegido, no hay más una simple relación aritmética entre un selector de segmento, el desplazamiento y la dirección física a la que ellos refieren. En lugar de eso, el microprocesador interpone una operación de mapeo especial entre una *dirección virtual* (el selector de segmento del modo protegido y el desplazamiento, tomados juntos) y la dirección física, como se muestra en la figura 1.7.1.

En modo protegido, un selector de segmento no es un agregado de una dirección; en vez de ello es una "receta mágica". El selector refiere a un *descriptor de segmento*, un registro en memoria conteniendo información acerca del segmento, incluyendo la *dirección base del segmento*, en la que el segmento comienza. El desplazamiento de dirección efectivo es sumado a la dirección base para producir una dirección física. El puntero de *selección* para todos los chips 80x86 es de 16 bits de ancho. El puntero de *desplazamiento* del 80286 es de 16 bits de ancho. Sin embargo, el puntero de desplazamiento para el 80386 y 80486 es de 32 bits de ancho.

1.7.4 Espacios de Direccionamiento Local y Global

Cada tarea que corre en modo protegido puede acceder segmentos dentro de un espacio de direccionamiento virtual que consiste en dos espacios menores: su *espacio de direcciones local* y su *espacio de direcciones global* (ver la figura 1.7.2). Los objetos de código y datos necesitados por toda tarea en el sistema, son puestos en el espacio de direcciones global, mientras que los objetos usados por una sola tarea van en el espacio de direcciones local. Ya que una tarea no puede acceder a memoria para la cual no tiene un selector, los espacios de direcciones locales están garantizados de ser privados a menos que un segmento sea mapeado intencionalmente en dos o más de ellos.



ESPACIO DE DIRECCIONES VIRTUAL Y AISLACION DE TAREAS

Fig. 1.7.2

La figura 1.7.3 muestra el formato de un selector de modo protegido y la forma en que este es usado para encontrar el descriptor de un segmento en memoria. Los 13 bits superiores del selector dan la ubicación del descriptor en un arreglo de descriptores llamado tabla de descriptores. Una tabla de descriptores es de 8 bytes de longitud, de forma que enmascarando los tres bits inferiores del selector convenientemente se forma el desplazamiento en la tabla del byte del descriptor.

Los tres bits inferiores del selector son bits de control. El bit 2, llamado el *indicador de tabla* (TI), toma una de las dos tablas de descriptores posibles: la *tabla de descriptores global*, la cual mantiene descriptores para segmentos en el espacio de direcciones global, o la *tabla de descriptores local* de la tarea, que hace lo mismo para el espacio de direcciones local.

Los bits restantes, 1 y 0, son el campo de *nivel de privilegio requerido (RPL).* Estos bits describen el nivel de privilegio que una tarea desea que se conceda cuando se accede al segmento.

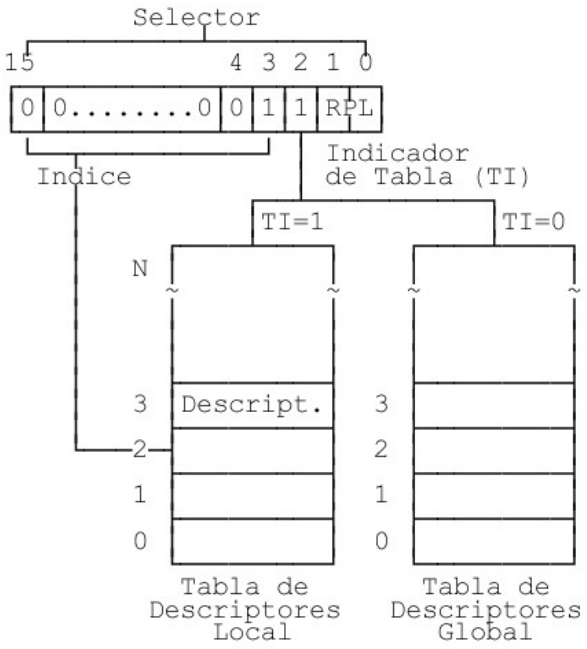


Fig. 1.7.3

1.7.5 Manejo de Memoria Sin Preocupaciones

Los selectores de segmento y las tablas de descriptores toman gran parte de la preocupación del manejo de memoria. Debido a que un selector de segmento es, en esencia, un "handle" que aparece como doble referenciado durante cada acceso a memoria, los segmentos pueden ser movidos sobre la memoria física en forma invisible. Y ya que un descriptor de segmento es automáticamente capturado dentro del microprocesador cuando su selector es puesto en un registro de segmento, no se necesitan dos registros de direcciones para acceder al segmento potencialmente movible. La carga de un selector de 16 bits lo hace todo.

Un segmento en modo protegido no sólo puede ser movido de un lugar a otro dentro de la RAM física, también puede ser permutado enteramente fuera de la RAM. El mecanismo de selector/descriptor de segmento permite al 80286 y sus parientes proveer memoria virtual sin una unidad de manejo de memoria separada. Cada descriptor de segmento contiene un bit que indica si el segmento está o no presente físicamente en RAM, y el microprocesador genera una excepción si una aplicación intenta acceder a un segmento que ha sido purgado o permutado hacia afuera.

1.7.6 Protección: el Modelo de la Cebolla

La jerarquía de privilegios y protección en los microprocesadores Intel sigue el modelo del *anillo*, o *cebolla*, mostrado en la figura 1.7.4. Hay cuatro niveles de privilegio posibles, numerados de 0 a 3, donde el nivel 0 es el más privilegiado.

Toda tarea en el sistema tiene un *nivel de privilegio actual (CPL)*, el cual es usado para determinar si puede hacer E/S, ejecutar instrucciones privilegiadas (aquellas que hacen cosas tales como parar la CPU), y/o acceder segmentos. Cada selector de segmento tiene también un campo de RPL, el cual es cambiado por una tarea si ella desea tener menor privilegio del que su CPL normalmente permitiría cuando accede al segmento. El *nivel de privilegio efectivo (EPL)* es menos privilegiado que el RPL y el CPL, y el determina si la tarea puede acceder a un segmento con un selector dado.

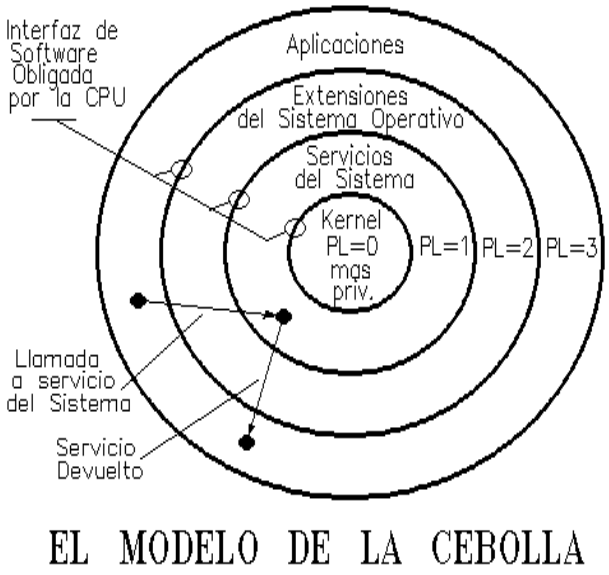


Fig. 1.7.4

¿Por qué querría una tarea darse a sí misma menos privilegio del que podría tener en otra ocasión al acceder a un segmento?. La respuesta se logrará con algo llamado: el problema del caballo de Troya. Suponga que un sistema operativo de modo protegido tiene una función llamada *DOSREAD*, la cual lee información de un archivo abierto (el OS2 tiene una función tal como ésta). La definición (encabezamiento) de esta función en C habrá de verse así:

```
unsigned int pascal far DOSREAD
(unsigned short FileHandle,
        void far*PtrBuffer,
    unsigned int BufferLength,
  unsigned int far*PtrBytesRead);
```

Usted no tiene que ser hábil en C para el propósito de este ejemplo; lo importante es que, entre los argumentos, hay un puntero llamado *PtrBuffer* (declarado como *void far*PtrBuffer*). La palabra *void* indica que el puntero es indefinido (ej.: puede apuntar a un objeto de cualquier clase), y la palabra *far* indica que el puntero consiste tanto de selector de segmento como de desplazamiento. *PtrBuffer* da la dirección de la locación que va a recibir los datos de la operación de lectura.

Ahora, suponga que una aplicación llamó a la función PtrBuffer apuntando a un segmento para el que a la aplicación se le ha permitido acceder (porque ésta no ha tenido el privilegio suficiente). El sistema operativo, que presumiblemente correría en un nivel de privilegio mayor, tendría acceso a ese segmento, así la aplicación causaría, en teoría, que el sistema operativo lea arbitrariamente datos en ese lugar de la memoria.

Para prevenir que esto ocurra, el sistema operativo ajusta el campo RPL dentro de la parte del selector de *PrtBuffer* y así él no tiene más privilegio que la aplicación cuando accede al segmento. Entonces, si el puntero se torna un caballo de Troya, la violación de privilegio será capturada por los mecanismos de protección habituales.

Toda la información acerca de que puede ser hecho con un segmento, y por quien, está contenida en el descriptor de segmento. Cada descriptor de segmento tiene un byte *de derechos*

de acceso, el cual a su vez contiene un campo de dos bits: el *descriptor de nivel de privilegio*. El EPL de la tarea debe darle igual o mayor privilegio que el DPL para que un acceso sea permitido. El byte de derechos de acceso también contiene otra información acerca de que puede ser hecho con el segmento, si puede ser leído o escrito, por ejemplo, y si este contiene código, datos, o una pila de una tarea.

Bibliografía:
L. Brett Glass – BYTE (Diciembre 1989) – Traducción y adaptación: FFP
Consulte también el texto de Tanenbaum "Organización de Computadoras".

1.7.7 Protección: el 80386

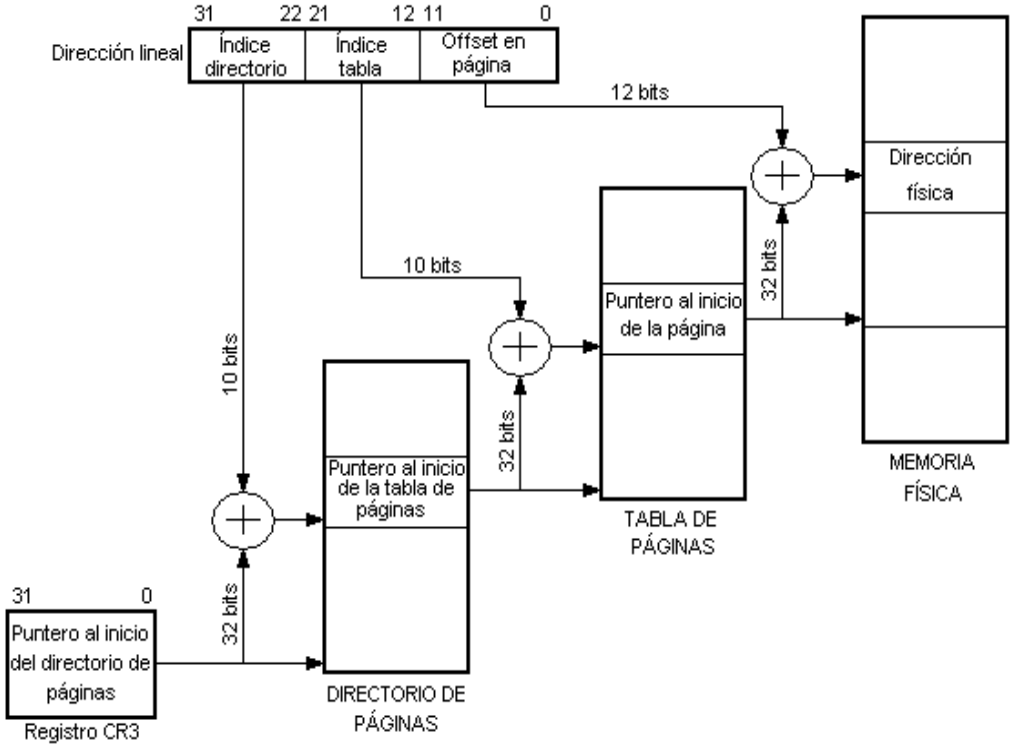


Fig. 1.7.5 – Paginación en el 80386

En el procesador 80386 la protección de la memoria basada en la paginación se implementa de acuerdo con el esquema de la figura 1.7.5.

El registro CR3 contiene la dirección física de donde comienza el directorio de páginas. Los 12 bits menos significativos del registro están en cero para que el directorio de páginas esté en una misma página.

El tamaño del directorio de páginas es de 4 kB y permite hasta 1024 entradas. Estas entradas se seleccionan mediante los bits 31-22 de la dirección lineal.

Cada entrada del directorio contiene la dirección de la tabla de páginas y varios bits de atributos de la página, como se muestra a continuación:

31 – 12	*11 - 9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*

Dirección tabla de páginas (31-12)	**Libre**	**0**	**0**	**D**	**A**	**0**	**0**	**U/S**	**R/W**	**P**

El tamaño de las tablas de páginas es de 4 kB y permite hasta 1024 entradas. Estas entradas se seleccionan mediante los bits 21-12 de la dirección lineal. Cada entrada de la tabla contiene la dirección inicial de la página y varios bits de atributos, como se muestra a continuación:

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

Los 20 bits más significativos de la dirección de la página se suman con los 12 bits menos significativos de la dirección lineal para formar la dirección física.

El bit 0 (Presente), cuando está en 1 indica que la página está en memoria y por lo tanto la entrada se puede utilizar para traducir de dirección lineal a física. Cuando P = 0 los otros 31 bits quedan libres a disposición del programador. Por ejemplo, si se usa memoria virtual, estos bits podrían utilizarse para indicar dónde se encuentra la página en el disco.

El bit 5 (Accedido), es puesto a uno por el 80386 en ambos tipos de entradas cuando ocurre un acceso de lectura o escritura en una dirección que esté dentro de una página cubierta por estas entradas.

El bit 6 (Dirty), es puesto a uno por el microprocesador cuando ocurre un acceso de escritura.

Los tres bits marcados como libres pueden ser utilizados por el programador.

Los bits 1 (Lectura/Escritura) y 2 (Usuario/Supervisor) se utilizan para proteger páginas individuales.

Para la paginación existen dos niveles de protección: usuario que corresponde al nivel de privilegio 3 y supervisor, que corresponde a los otros niveles: 0, 1 y 2.

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

Bibliografía:
Darío Alpern – Adaptación: FFP

1.8 MEMORIA Y MÁQUINA VIRTUALES

Los MC68010/MC68012 introdujeron el concepto de memoria y máquina virtual en la arquitectura de la familia 68000 de Motorola. En la familia 80x86 de Intel los primeros procesadores en incorporar estas características fueron el 80286 y el 80386.

En la mayoría de los sistemas de micro cómputo, sólo una fracción del espacio de direcciones corresponderán realmente a memoria física. Con las técnicas de **memoria virtual**, el sistema puede ser hecho de manera que para el usuario aparezca como teniendo más memoria física accesible. Estas técnicas han sido usadas durante muchos años en grandes computadoras, luego en minicomputadoras y desde la década del 1980 está disponible en los microprocesadores.

En un sistema de memoria virtual, un programa de usuario puede ser escrito pensando que hay una gran cantidad de memoria disponible para él, cuando sólo una pequeña cantidad de memoria está presente físicamente en el sistema. En una forma similar, un sistema puede ser diseñado de tal manera que permita a los usuarios acceder a otros tipos de dispositivos que no están físicamente presentes en el sistema, tales como unidades de cinta, unidades de disco, impresoras o CRTs.

Con una apropiada emulación de software, un sistema físico puede ser hecho de manera que aparezca al programa de usuario como algún otro sistema de computadora y el programa puede tener acceso a la totalidad de los recursos del sistema emulado. Tal sistema emulado es llamado una **máquina virtual**.

1.8.1 Memoria virtual

El mecanismo básico para soportar memoria virtual en las computadoras es proveer una cantidad limitada de memoria física de alta velocidad que puede ser accedida directamente por el procesador, mientras se mantiene una imagen de memoria "virtual" mucho mayor en dispositivos de almacenamiento secundarios tales como unidades de disco de alta capacidad. Cuando el procesador intenta acceder a una locación en el mapa de memoria virtual que actualmente no se encuentra en la memoria física (referido como falla de página), el acceso a esa locación es suspendido temporariamente mientras son buscados los datos necesarios desde el medio de almacenamiento secundario y puestos en memoria física. El acceso que había sido suspendido es entonces completado.

Los MC68010/MC68012 proveen soporte de hardware para memoria virtual con la capacidad de suspender la ejecución de una instrucción cuando se señaliza un error de bus y luego completar la instrucción, luego que la memoria física haya sido actualizada como era necesario.

Los MC68010/MC68012 usan continuación de instrucción en vez de reiteración de instrucción para soportar memoria virtual. Con reiteración de instrucción, el procesador debe recordar el estado exacto del sistema antes de que comience cada instrucción a los fines de restaurar ese estado en el caso de que ocurra una falla de página durante su ejecución. Entonces, después que la falla de página haya sido reparada, la instrucción que ha causado la falla se ejecutará completa. Con continuación de instrucción, cuando ocurre una falla de página, el procesador almacena su estado interno y luego de que la falla de página es reparada, restablece ese estado interno y continúa la ejecución de la instrucción. Para implementar la continuación de instrucción, los MC68010/MC68012 almacenan su estado interno en la pila del modo supervisor cuando un ciclo de bus termina con una señal de error de bus. Este carga luego el contador de

programa desde el vector número dos de la tabla de interrupciones (desplazamiento $008) y continúa la ejecución del programa a partir de esa nueva dirección. Cuando la rutina de manejo de excepciones correspondiente al error de bus ha completado su ejecución, se ejecuta una instrucción RTE, la cual carga el MC68010/MC68012 con el estado interno almacenado en la pila, ejecuta nuevamente el ciclo de bus que falló y continúa la instrucción suspendida. La continuación de instrucción tiene la ventaja adicional de permitir soporte de hardware para dispositivos virtuales de E/S. Como los registros virtuales pueden ser simulados en el mapa de memoria, un acceso a tales registros causará una falla de bus y la función de ese registro podrá simularse por software.

Ejemplo de memoria virtual con un procesador genérico

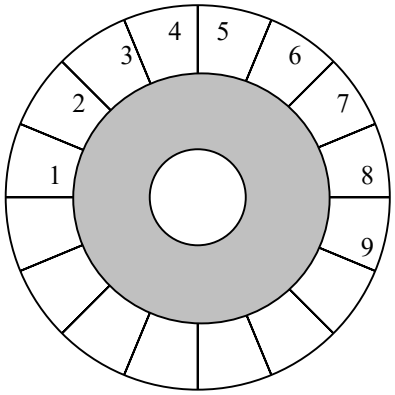
Se pretende cargar un programa que ocupa 9 bloques desde la unidad de disco. La memoria RAM disponible (L) es de 6 bloques. Dos bloques están ocupados por el sistema operativo (SO).

0	0	N
1	0	N
2	0	N
3	0	N
4	0	N
5	0	N
6	0	N
7	0	N
8	0	N
9	0	N

Tabla de Mem. Virtual

00	SO
10	SO
20	L
30	L
40	L
50	L
60	L
70	L

Memoria RAM



Unidad de disco

Fig. 1.8.1 – Memoria virtual - problema

Si el sistema no posee HW para soportar memoria virtual, el SO informará “memoria insuficiente” y el programa no podrá ejecutarse. En cambio, si el HW está disponible, el procesador contará con registros para “cargar” el programa completo como se ve en la figura 1.8.1. Primero el SO lee el bloque 1 del disco, donde está la longitud total del archivo (cantidad de bloques). Luego verifica cuantos bloques libres de memoria hay y copia del disco los primeros 6 bloques hasta completar la memoria. Finalmente, completa la tabla de la memoria virtual indicando los bloques presentes (P) y los que no se pudieron traer (N).

La tabla y la memoria, luego de la carga inicial, quedan como muestra la figura 1.8.2. Se consideró una tabla con 10 posiciones, donde se cargó la dirección de cada bloque, ya sea en memoria, si está presente o en disco, si no lo está.

Ahora el SO transfiere el control al programa, en la dirección de memoria 20. Para acceder a esta dirección el procesador determina la dirección física basado en la tabla de direcciones TMV, y si está presente comienza a ejecutar normalmente. Si en algún momento se pretende un acceso a una dirección no presente, por ejemplo, el bloque 7 del disco, se genera una excepción (interrupción interna) cuya rutina de atención deberá traer del disco el bloque 7, actualizar la tabla TMV y retomar el programa, ahora con el bloque 7 en RAM.

Hay distintas estrategias para determinar que bloque de RAM se libera para hacer lugar al nuevo bloque a traer de disco. Una de ellas, FIFO, consiste liberar el bloque más antiguo en RAM.

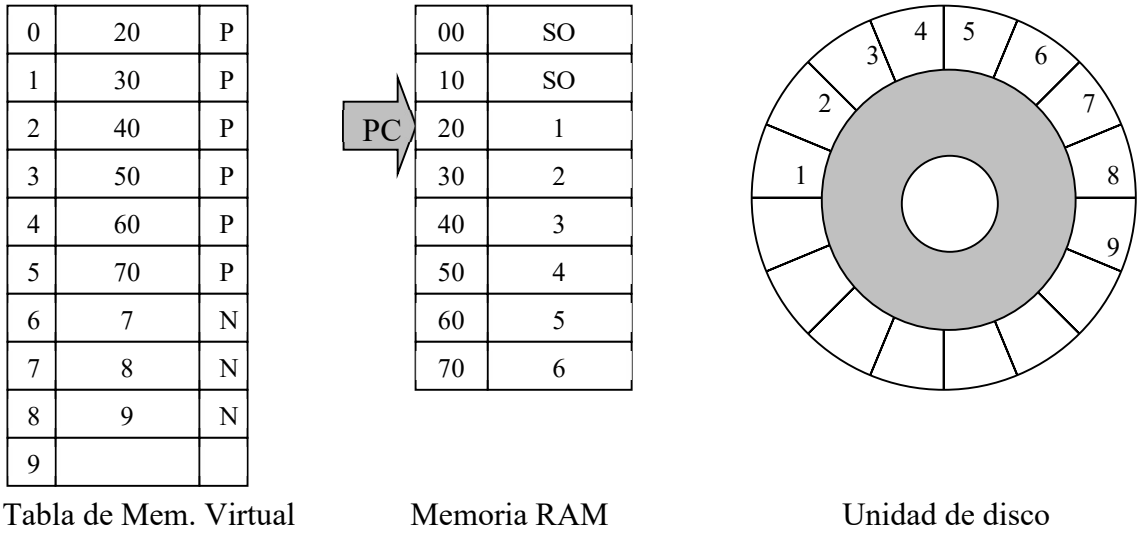


Fig. 1.8.2 – Memoria virtual - carga inicial

1.8.2 Máquina virtual

Un uso típico de un sistema de máquina virtual es en el desarrollo de software, como puede ser un sistema operativo, para otra máquina con hardware aún en desarrollo y por ende no accesible para el programador. En tal sistema, el sistema operativo director emula el hardware del nuevo sistema operativo sea ejecutado y depurado tal como si hubiese estado corriendo en el nuevo hardware. Como el nuevo sistema operativo está controlado por el sistema operativo director, el nuevo debe ejecutarse con un nivel de privilegio menor que el sistema operativo director, de forma tal que cualquier intento del nuevo sistema operativo de usar recursos virtuales que no estén físicamente presentes, y que deberían ser emulados, será capturado por el sistema operativo director y manejado por software.

En los MC68010/MC68012 puede ser completamente soportada una nueva máquina virtual, corriendo el nuevo sistema operativo en el modo usuario y el sistema operativo director en el modo supervisor, de forma que cualquier intento de acceder a los recursos del supervisor o ejecutar instrucciones privilegiadas por el nuevo sistema operativo será capturado (trap) por el sistema operativo director.

A los efectos de soportar completamente una máquina virtual el MC68010/MC68012 debe proteger los recursos del supervisor del acceso de por parte de programas de usuario. El único recurso de supervisor que no está completamente protegido en el MC68000 es el byte del sistema del registro de estado. En el MC68000 y el MC68008, la instrucción MOVE desde el SR permite a los programas de usuario verificar el estado del bit S (además de los bits T y la máscara de interrupciones) y así determinar que ellos están corriendo en el modo usuario.

Para un completo soporte de máquina virtual, un sistema operativo nuevo no debe estar enterado del hecho que correrá en el modo usuario y por lo tanto no debería permitirse que lea el bit S. Por esta razón se ha agregado la instrucción MOVE desde el SR para permitir al programa de usuario un libre acceso a los códigos de condición. Si la instrucción MOVE desde el SR

hubiese sido privilegiada, cuando el sistema operativo nuevo intente acceder al bit S, será capturado por el sistema operativo director y la imagen del SR que éste le pasará contendrá el bit S fijado (verdadero).

Bibliografía:
MC68010/MC68012 16/32 Virtual Memory Microprocessor. (traducción: FFP)

1.9 SOFTWARE PARA EJECUTAR MÁQUINAS VIRTUALES

INTRODUCCIÓN:

Existen programas que, ejecutándose en un procesador que soporte modo protegido y con el sistema operativo apropiado, pueden ejecutar múltiples máquinas virtuales en una única máquina real. Tal es el caso de VMware, Microsoft Virtual PC y VirtualBox.

Además de su interés como medio de desarrollo, experimentación y estudio, actualmente la virtualización permite también la racionalización del hardware de servidores, instalando en una sola máquina anfitrión lo que antes se hacía con varias, cada una de ellas con su diferente HW y sistema operativo. Finalmente, esto ayuda también a la racionalización energética por tener menos nodos físicos activos y al menor costo de mantenimiento por la estandarización del HW.

1.9.1 VirtualBox - Arquitectura

La virtualización es, por naturaleza, extraordinariamente compleja, especialmente en hardware basado en x86. La comprensión del código fuente de VirtualBox por lo tanto requiere al menos para algunos componentes, un gran entendimiento acerca de los detalles de la arquitectura x86 como así también un gran conocimiento acerca de la implementación de la plataformas anfitrión e invitado involucradas.

Los procesos VirtualBox:

Cuando usted inicia la interfaz gráfica VirtualBox, al menos un proceso adicional se inicia (VBoxSVC: el proceso de “servicio” VirtualBox). Una vez que usted ha iniciado la máquina virtual (VM) desde de la interfaz gráfica, usted tiene dos ventanas (y la ventana principal y de la VM), pero habrá tres procesos corriendo. Observando su sistema desde el administrador de tareas (en Windows) o el monitor de sistema (en Linux) usted verá esto:

1. VirtualBox, la interfaz gráfica para la pantalla principal.
2. Otro proceso VirtualBox que fue iniciado con el parámetro –startvm, lo cual significa que su interfaz gráfica actuará como un entorno para la máquina virtual.
3. VBoxSVC, el servicio mencionado previamente, el cual está corriendo en segundo plano para seguir la pista de todos los procesos involucrados. Esto fue automáticamente iniciado por el primer proceso de interfaz gráfico.

Nombre de imagen	Nombre de us...	ID. DE ...	CPU	Uso de ...
VBoxSVC.exe	Fernando	0	00	34.148 KB
VirtualBox.exe	Fernando	0	00	51.228 KB
VirtualBox.exe	Fernando	0	64	100.312...

Fig. 1.9.1 – Procesos de VirtualBox

Para el sistema operativo anfitrión (SO), la VM que corre "dentro" de la segunda ventana se ve como un programa ordinario. VirtualBox tiene muy buen comportamiento en ese aspecto: él toma el control sobre una gran parte de su computadora, ejecutando un sistema operativo completo con su propio conjunto de procesos invitados, drivers, y dispositivos adentro de su proceso de máquina virtual, pero el sistema operativo anfitrión no tiene demasiada noticia acerca de esto. Lo que sea que la máquina virtual haga, es sólo un proceso más en su sistema operativo anfitrión. Tenemos entonces dos tipos de encapsulado que tiene lugar en los diversos archivos y procesos VirtualBox:

1. Arquitectura cliente/servidor. Todos los aspectos de VirtualBox y las máquinas virtuales que están corriendo pueden ser controlados con una simple y aun así poderosa, COM/XPCOM API. Por ejemplo, existe una utilidad de línea de comando llamada VBoxManage que permite controlar las máquinas virtuales tal como lo hace la interfaz gráfica.
2. Arquitectura Frontend/Backend. Las entrañas de VirtualBox (todo eso que hace la virtualización x86 complicada) están ocultos en una librería compartida, VBoxVMM.dll, o VBoxVMM.so en Linux. Esto pude ser considerado un "backend" o caja negra, la cual es estática y es relativamente fácil escribir otro "frontend" sin tener que lidiar con la complejidad de la virtualización x86.

Dentro de una máquina virtual

Desde la perspectiva del SO anfitrión, una máquina virtual es simplemente otro proceso. El SO anfitrión no necesita muchas modificaciones para soportar la virtualización. Aun pensando que hay un driver corriendo en el anillo 0 (recuerde el modelo de la cebolla) que debe cargarse en el SO anfitrión para que funcione VirtualBox, este driver hace menos de lo que usted puede pensar. Sólo se necesita para unas pocas tareas específicas, tales como:

- Asignación de la memoria física para la VM;
- Guardar y recuperar los registros del CPU y tablas de descriptores cuando ocurre una interrupción del anfitrión mientras se ejecuta código de un invitado en el anillo 3 (por ejemplo, cuando el SO anfitrión OS quiere reorganizar la planificación de tareas);
- Al cambiar de contexto desde el anfitrión en anillo 3 al invitado;
- Habilitar o deshabilitar el soporte para VT-x etc. (procesadores avanzados).

Lo más importante, el driver de anillo 0 del anfitrión no interfiere con el planificador de tareas y prioridades de su SO. El SO invitado entero, incluyendo sus propios procesos, es programado para ejecutarse (sheduled) sólo cuando el SO anfitrión le asigna al proceso de la VM un tiempo de ejecución (timeslice) de acuerdo con su prioridad como una tarea más.

Luego de que una VM se ha iniciado, desde el punto de vista de su procesador, su computadora puede estar en uno entre varios estados (lo siguiente requerirá una buena comprensión de la arquitectura de anillo del x86):

1. Su CPU puede estar **ejecutando código de anfitrión de anillo 3** el (por ejemplo de otros procesos del anfitrión), o **código de anfitrión de anillo 0**, como sería si VirtualBox no estuviese ejecutándose.
2. Su CPU puede estar **emulando el código del invitado** (dentro del proceso VM de anillo 3 del anfitrión). Básicamente, VirtualBox intenta ejecutar nativamente tanto código del invitado como posible. Pero puede (más lentamente) emular el código del invitado como recurso cuando no está seguro de lo que el sistema del invitado está haciendo, o cuando la penalización de performance de la emulación no es demasiado alta. Nuestro emulador (en src/emulator /) está basado en QEMU y típicamente funciona cuando:
    - el código del invitado desactiva las interrupciones y VirtualBox no pueden deducir cuando se habilitarán nuevamente (en estas situaciones, VirtualBox

analiza el código del invitado usando su propio desensamblador en src/VBox/Disassembler/);
    - para la ejecución de ciertas instrucciones individuales; esto pasa típicamente cuando una instrucción privilegiada del invitado como LIDT ha causado una excepción y ha necesitado ser emulada;
    - para cualquier código de modo real (por ejemplo, código de BIOS, un invitado de DOS, o cualquier inicio del sistema operativo).
3. Su CPU puede estar **ejecutando nativamente código de invitado de anillo 3** (dentro del proceso VM de anillo 3 del anfitrión). Con VirtualBox, nosotros llamamos a esto "anillo 3 crudo". Esto es, claro, la manera más eficaz de ejecutar al invitado, y en lo posible nosotros no dejamos a menudo este modo.
4. Su CPU puede estar **ejecutando nativamente código de invitado de anillo 0**. Aquí es donde las cosas se ponen difíciles: el invitado sólo piensa que está corriendo código en anillo 0, pero VirtualBox ha engañado al OS invitado para entrar en anillo 1 (que normalmente no es usado en los sistemas operativos x86).

También, en el código fuente de VirtualBox, usted encontrará las muchas referencias a "contexto del anfitrión" o "contexto del invitado". Esencialmente, éstos significan:

- **Contexto del anfitrión (HC**) significa que el OS anfitrión tiene el control de todo, incluyendo la memoria virtual. En las fuentes de VirtualBox, el término "HC" normalmente se referirá sólo al contexto de anillo-3 del anfitrión. Nosotros sólo usamos al contexto de anillo-0 del anfitrión (R0) con nuestro nuevo soporte para Intel VT-x (Vanderpool), por lo que dejaremos el tema para después.
- **Contexto del invitado (GC)** significa básicamente que OS invitado tiene el control, pero con VirtualBox observando en las cosas en segundo plano. Aquí, VirtualBox ha preparado el CPU y la memoria exactamente la manera que el invitado espera, pero se ha insertado "al fondo de la foto". Puede asumir el mando entonces cuando ocurren cosas no permitidas. (una instrucción privilegiada se ejecuta, las excepciones del invitado o las interrupciones externas ocurren). VirtualBox puede delegar entonces el manejo las tales cosas al SO anfitrión. Así que, en el contexto del invitado, nosotros tenemos:
    - anillo 3 (es de esperar que sea ejecutado en "modo crudo" todo el tiempo);
    - anillo 1 (el invitado piensa que es anillo 0, vea lo anterior), y
    - anillo 0 (el cual es código VirtualBox). Este contexto de código de invitado de anillo 0, también se llama a menudo "**hypervisor**".

Soporte de Intel VT-x (" Vanderpool") y AMD-V (SVM)

Con sus últimos procesadores, Intel ha introducido el soporte de virtualización por hardware, llamado "Vanderpool", "IVT", "VT-x", o "VMX" (para las "extensiones de máquina virtual"). Cuando VirtualBox comenzó hace bastante con esto, usaba internamente el término "VMX". Una explicación completa de esta arquitectura puede encontrarse en las páginas de Ref (1), pero con estas extensiones, un procesador opera siempre en uno de los dos modos siguientes:

- En el **modo raíz**, su comportamiento es muy similar al modo normal de funcionamiento (sin VMX), y éste es el contexto que se ejecuta un monitor de la máquina virtual (VMM).
- El modo **no-raíz** (o contexto de invitado) se diseñó para ejecutar una máquina virtual.

Una novedad notable es que los cuatro niveles de privilegio (anillos) son soportados en cualquier modo, de manera que el software de invitado puede correr teóricamente en cualquiera de ellos. VT-x define entonces las transiciones del modo raíz al modo no-raíz (y viceversa) y llama a éstos "entrada de VM" (VM entry) y "salida de VM" (VM exit).

En el modo no-raíz, el procesador causará automáticamente VM exits para ciertas instrucciones privilegiadas y eventos. Para algunas de estas instrucciones, es configurable de qué manera deben ocurrir las salidas de VM.

Como dijimos anteriormente, nuestro hypervisor, en máquinas no-VT-x, está en el anillo 0 del contexto de invitado, debajo del código de invitado anillo-0 (que realmente se ejecuta en anillo 1). Cuando VT-x se habilita, el hypervisor puede estar seguro en el contexto de anillo 0 del anfitrión y se pueden activar automáticamente por el uso de las nuevas salidas de VM.

VirtualBox también soporta totalmente AMD-V, SVM de AMD y VT-x de Intel.

Referencia (1):
http://www.virtualbox.org/wiki/VirtualBox_architecture (traducción: FFP)

VirtualBox – Ejemplo de aplicación

A continuación, se puede ver una sesión de VirtualBox ejecutándose en Ubuntu Linux. La configuración del programa es muy simple. En este caso se creó un disco rígido virtual de 30 MB para instalarle DOS 6.22.

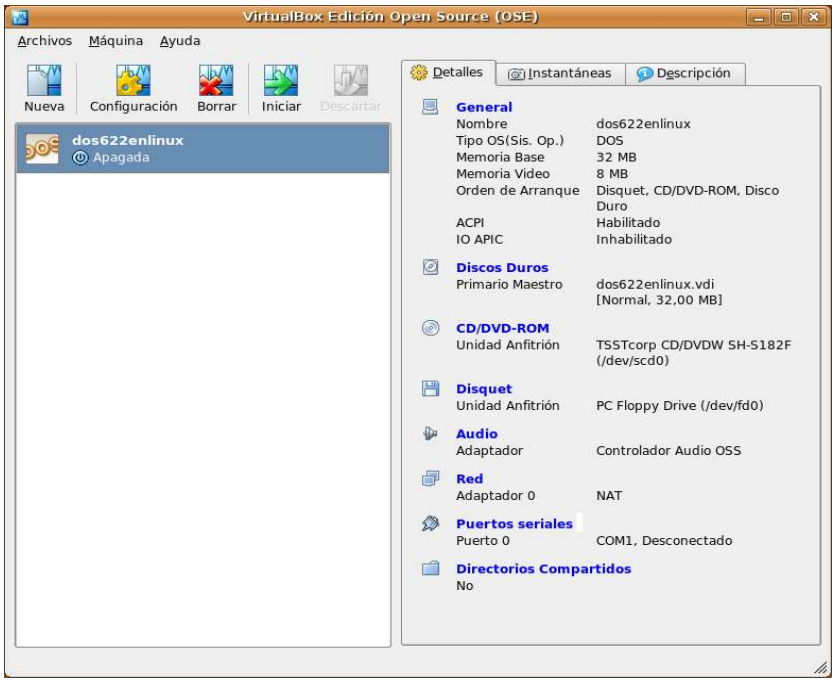


Fig. 1.9.2 – Sesión de VirtualBox

La máquina virtual se arranca con el botón Iniciar. Aparece una nueva ventana que muestra el proceso de "booteo". En este caso se inició desde el CD ROM de Técnicas Digitales III, mostrando su menú principal.

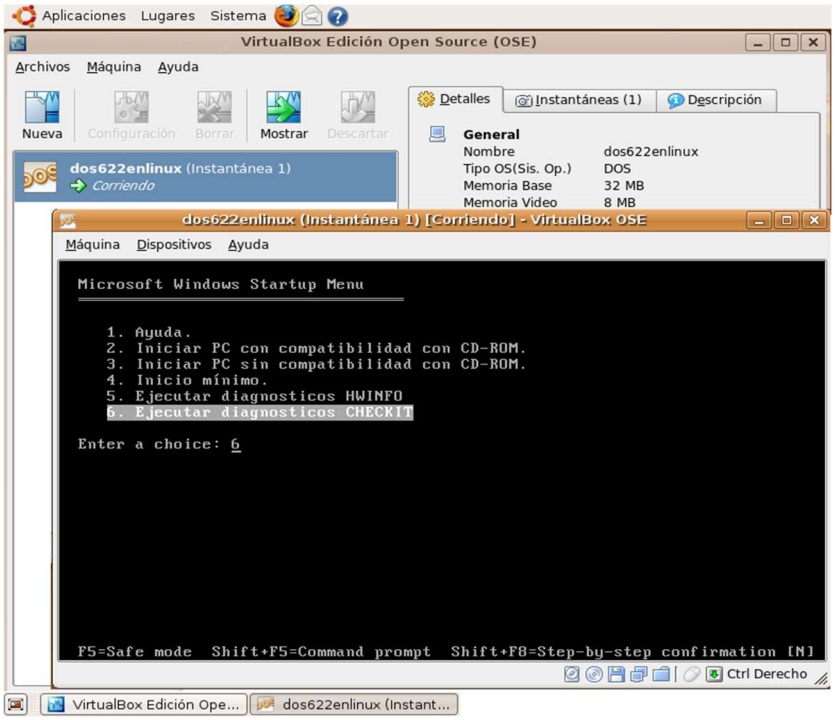


Fig. 1.9.3 – Booteo de VirtualBox

1.9.2 VMware - características

Cuando se desarrolla software que se ejecutará con otro sistema operativo, por ejemplo Windows, Linux o Novel Netware, las distintas máquinas virtuales ven la CPU, la memoria, la placa de red y los discos como si fuesen propios.

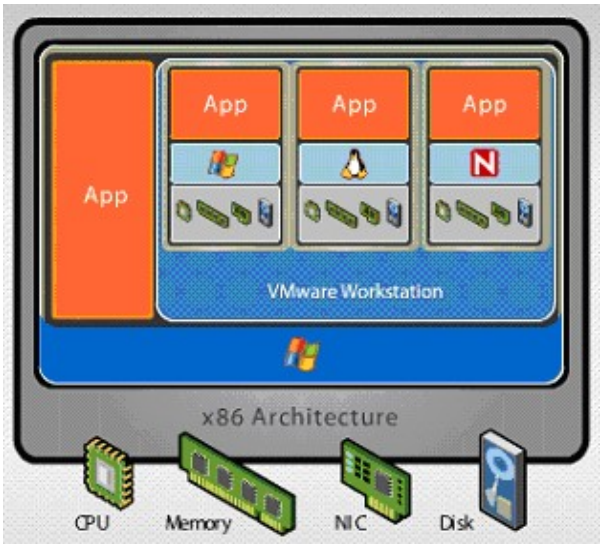


Fig. 1.9.4 – Arquitectura de VMware – Sistemas operativos

También cuando se está desarrollando una aplicación que se ejecutará en red. En este caso, las distintas máquinas virtuales se podrán comunicar entre sí como si fuesen reales.

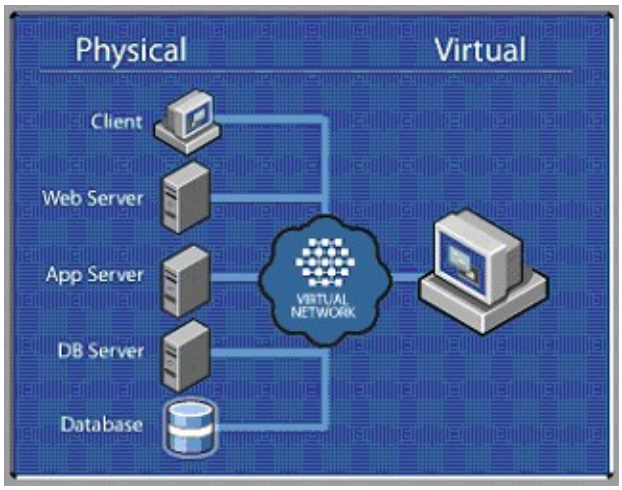


Fig. 1.9.5 – Arquitectura de VMware

Por último, por tratarse de una máquina virtual, se la puede iniciar (CTL-ALT-DEL), se puede particionar su disco rígido, se lo puede formatear, etc., sin afectar a la máquina ni al sistema operativo anfitrión ni a las demás máquinas virtuales.

Si la máquina virtual se ve infectada por virus informáticos, la máquina anfitrión no será afectada.

Desde los sitios Web de Microsoft, VMware y VirtualBox se pueden descargar versiones de estos programas según el SO que actúe como anfitrión (host).

1.10 MULTIPROCESAMIENTO

1.10.1 Introducción

Multiprocesamiento significa usar en un sistema dos o más procesadores coordinados entre sí. El multiprocesamiento se ha ido tornando más atractivo a medida que el costo de los procesadores ha ido bajando. La performance se puede incrementar substancialmente distribuyendo las tareas del sistema entre procesadores separados que las ejecuten concurrentemente. Además, el multiprocesamiento incita a un diseño modular, lo que redunda usualmente en sistemas que son más fáciles de mantener y ampliar.

En general, el uso de procesadores múltiples ofrece varias ventajas significativas respecto del método centralizado, que depende de una sola CPU y una memoria extremadamente rápida:

- Las tareas del sistema pueden ser asignadas a procesadores específicos cuyos diseños están optimizados para realizar dichas tareas de manera fácil y eficiente.
- Se pueden lograr muy altos niveles de performance cuando los procesadores pueden ejecutar simultáneamente (procesamiento distribuido/paralelo).
- La subdivisión del sistema estimula el desarrollo en paralelo de los subsistemas, divide la aplicación en tareas más pequeñas y por ende más manejables, y ayuda a aislar los efectos de modificaciones en el sistema.

Existen dos tipos de procesadores: los procesadores independientes y las extensiones del procesador, también llamadas coprocesadores. Un procesador independiente ejecuta su propia secuencia de instrucciones. El 8086, el 8088 y el 8089 son ejemplos de procesadores independientes. Un segundo tipo de procesador, llamado "extensión del procesador", tal como el 8087 NPX, agrega registros, tipos de datos y nuevas instrucciones al sistema. Una extensión del procesador, en efecto, extiende el conjunto de instrucciones (y la arquitectura) de su procesador principal.

En los sistemas de multiprocesamiento existen dos problemas de coordinación clásicos: arbitraje del bus y exclusión mutua. El arbitraje del bus puede ser realizado por la lógica de pedido/concesión del bus contenida en cada uno de los procesadores (arbitraje de bus local), por árbitros de bus (arbitraje del bus del sistema) o por una combinación de ambos cuando los procesadores comparten múltiples buses.

A medida que cada CPU requiere asincrónicamente el acceso al bus compartido, la lógica de arbitraje resuelve prioridades y concede acceso al bus, la CPU completa su transferencia y luego cede el bus o espera ser forzada a ceder el bus. En todos los casos, el arbitraje opera invisible al software. Para la exclusión mutua, cada procesador tiene una señal de bloqueo para el bus (bus LOCK), que puede ser activada por un programa para prevenir que otros procesadores obtengan acceso al bus compartido. Cada procesador tiene una instrucción que puede examinar un byte de memoria mientras el bus está bloqueado, permitiendo de esta manera implementar un mecanismo de señalización (semáforo) para controlar el acceso a los recursos compartidos (tablas, buffers, etc.).

Nota: el desarrollo precedente corresponde al inicio del multiprocesamiento por parte de Intel y su implementación a fines de los 90. Los procesadores actuales continúan con el mismo concepto con la salvedad de que el aumento de la integración posibilitó incorporar en un mismo chip varios procesadores, las unidades de punto flotante y el I/O necesario. El aumento de la cantidad de pines evita el multiplexado con el consiguiente aumento de performance en el bus.

1.10.2 Arquitectura Multibus

Para soportar la interfaz de múltiple maestro para la familia 8086/88, Intel incluyó el árbitro de bus 8289. El 8289 es compatible con el bus local del 8086 y en conjunto con el controlador de bus 8288 implementa el arbitraje del bus con el protocolo MULTIBUS.

Véase que en la figura se representan 2 procesadores idénticos que se comunican entre sí sólo por el bus y los recursos compartidos, de manera que si fuese necesario colocar mas procesadores se pueden agregar en paralelo sin mayores inconvenientes.

La mayoría de los procesadores actuales no se pueden conectar de esta forma, ya que la estrategia de los fabricantes ha cambiado, promoviendo el cambio de modelo y no la ampliación mediante multiprocesamiento.

Los procesadores Intel aptos para multiprocesamiento son los de la línea Xeon, destinados principalmente a servidores.

Adicionalmente, los procesadores actuales son multinúcleos, con lo que el multiprocesamiento se realiza en el interior del chip y el SO distribuye las tareas a los distintos núcleos

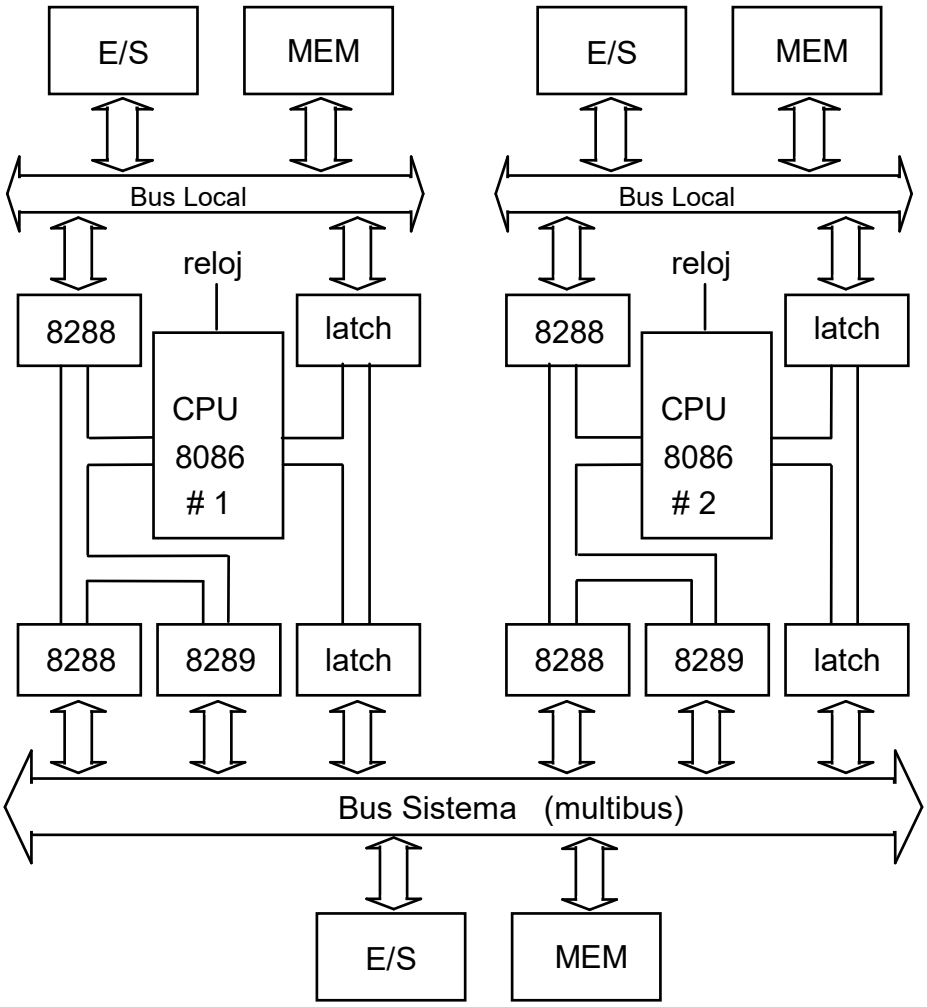


Fig. 1.10.1 - Sistema de dos procesadores usando Intel 8086

1.10.3 Ejemplo – problema del semáforo

Se trata de un sistema de dos procesadores conectados como indica la figura 1.10.1. El primero de ellos (P1) adquiere datos de 5 temperaturas de un proceso cada 30 segundos. Luego de realizar la conversión a ºC los debe enviar al otro procesador (P2) para realice los cálculos apropiados de acuerdo con un modelo matemático y evalúe las correcciones necesarias.

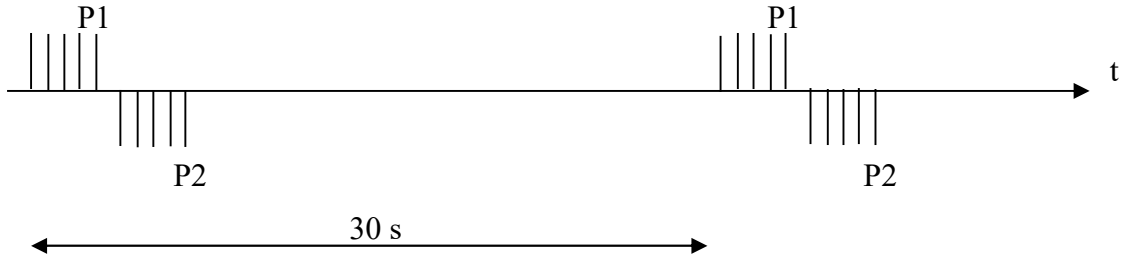


Fig. 1.10.2 – Diagrama de tiempos del ejemplo

Para comunicar los procesadores se desea utilizar una tabla de 5 valores en la memoria compartida o de sistema (ver Tabla 1.10.1). T1 (0) indica el valor de temperatura 1 adquirido en el muestreo 0. La dirección del primer dato es acordada entre los dos procesadores y las de los restantes siguen en orden consecutivo. El procesador P2 debe poder leer la tabla en forma asincrónica, es decir, cuando lo estime conveniente, sin previo aviso de P1. Los 5 valores deben ser considerados como un "conjunto" de datos, que sólo es válido cuando hayan sido obtenidos en el mismo ciclo de muestreo.

El inconveniente que se puede presentar es que P2 lea la tabla de datos mientras P1 la está actualizando. En este caso, el conjunto de valores será una mezcla de datos nuevos, adquiridos en el muestreo (1) y viejos (0) (ver Tabla 1.10.2).

T1 (0)
T2 (0)
T3 (0)
T4 (0)
T5 (0)

Tabla 1.10.1

T1 (1)
T2 (1)
T3 (1)
T4 (0)
T5 (0)

Tabla 1.10.2

Para intentar la solución de este problema aprovecharemos las facilidades de HW y SW de ambos procesadores y agregaremos una posición más en la tabla, que será una bandera o semáforo para controlar el acceso a la tabla. La misma en 00 indica que el conjunto de datos es coherente y corresponde al mismo muestreo. En este estado puede ser leído por P2. Con criterio más amplio, la tabla está "habilitada". La bandera en FF indica que la tabla está siendo actualizada por P1 o leída por P2. En este estado, podemos decir que la tabla está "bloqueada" por el procesador que la está accediendo.

Partimos de la situación en que los datos son coherentes y FLAG=00. Cuando P2 decide leer, verifica primero si FLAG=00. Si es así, pone FLAG=FF y luego lee los 5 datos. Si en el

primer intento FLAG=FF, espera e intenta nuevamente. Luego de leer todos los datos, P2 pone FLAG=00. Cuando P1 necesita actualizar los datos, verifica primero si FLAG=00. Si es así, pone FLAG=FF, escribe los 5 datos y pone FLAG=00.

FLAG (00)
T1 (0)
T2 (0)
T3 (0)
T4 (0)
T5 (0)

Tabla 1.10.3 (habilitada)

FLAG (FF)
T1 (1)
T2 (1)
T3 (1)
T4 (0)
T5 (0)

Tabla 1.10.4 (bloqueada)

Parecería que con este simple artilugio de SW es suficiente para solucionar nuestro problema, sin embargo, no es así. En la Tabla 1.10.5 analizamos las actividades de P1 y P2 en el tiempo (secuencia de ciclos de bus: 1, 2, 3...). En la columna FLAG está el valor "verdadero" de esa posición de memoria de sistema. En P1 está la "copia" que hace P1 cuando lee la memoria de sistema y la lleva a un registro interno donde determinará su valor (00 o FF) y las instrucciones ejecutadas. Ídem P2 para el procesador P2. Obsérvese que P1 y P2 no pueden utilizar el bus al mismo tiempo.

Tiempo	FLAG	P1	P2
1	00	CMP, FLAG=00	-
2	00	JMPE	CMP, FLAG=00
3	FF	MOV, FLAG=FF	JMPE
4	FF	Conversión T1	MOV, FLAG=FF
5	FF	ídem	Preparación lectura
6	FF	ídem	ídem
7	FF	MOV, P1→T1	ídem
8	FF	Conversión T2	MOV, T1→P2
9	FF	ídem	-
10	FF	ídem	MOV, T2→P2
11	FF	MOV, P1→T2	-
12	FF	Conversión T3	MOV, T3→P2
13	FF	ídem	-
14	FF	ídem	MOV, T4→P2
15	FF	MOV, P1→T3	-
16	FF	Conversión T4	MOV, T5→P2

Tabla 1.10.5

Como se puede apreciar, en el tiempo 16 el procesador P2 leyó los 5 valores, pero T1 corresponde a una muestra distinta (más nueva) que los restantes (T2 a T5). Esto ocurrió porque los dos procesadores "vieron" el FLAG en 00 y por lo tanto habilitada para operar.

Tiempo	**FLAG**	**P1**	**P2**
1	00	MOV AL,FF	-
2	FF	LOCK XCHG FLAG=FF	MOV AL,FF
3	FF	TEST AL,AL FLAG=00	LOCK XCHG FLAG=FF
4	FF	Habilitado para escribir	TEST AL,AL FLAG=FF
5	FF	Conversión T1	Bloqueado para leer
6	FF	-	-
7	FF	-	-
8	FF	MOV, P1→T1	-
9	FF	Conversión T2	-
10	FF	-	-
-	-	-	-
24	FF	MOV, P5→T5	-
25	00	MOV FLAG,00	-
26	00	-	MOV AL,FF
27	FF	-	LOCK XCHG FLAG=00
28	FF	-	TEST AL,AL FLAG=00
29	FF	MOV AL,FF	Habilitado para leer
30	FF	LOCK XCHG FLAG=FF	-
31	FF	TEST AL,AL FLAG=FF	MOV, T1→P2
32	FF	Bloqueado para escribir	-
33	FF	-	MOV, T2→P2
-	-	-	-
39	FF	-	MOV, T5→P2
40	FF	-	MOV, 00→FLAG

Tabla 1.10.6

Para solucionar esto, los procesadores como el Intel 8086, diseñado para trabajar en sistemas de procesadores múltiples, cuenta con un prefijo (instrucción LOCK) para bloquear el bus con la señal LOCK mientras determina el estado de la bandera y una instrucción XCHG para intercambiar un registro y memoria. De esta manera, el otro procesador no puede acceder a la tabla ni a la bandera hasta que el primero no haya fijado "su" bandera y por tanto asegurado la integridad de los datos. El Pentium agrega la instrucción CMPXCHG8B *reg*, *mem64* (*Compare and Exchange 8 Bytes*) que hace lo siguiente: 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 fija en 1. En caso contrario, el valor en memoria se carga en EDX:EAX y el indicador ZF se fija en cero.

Volviendo al problema, la tabla de tiempos quedaría modificada como muestra la Tabla 1.10.6, donde se ve que ahora funciona correctamente.

Nota: se puede ver una animación paso a paso de este ejemplo, en la página web de Técnicas Digitales III (http://www.frsn.utn.edu.ar/tecnicas3).

1.11 COPROCESADORES

1.11.1 Introducción

En la figura se puede ver el esquema de interconexión entre la CPU Intel 8086, el coprocesador matemático 8087 y los circuitos auxiliares para su correcta operación.

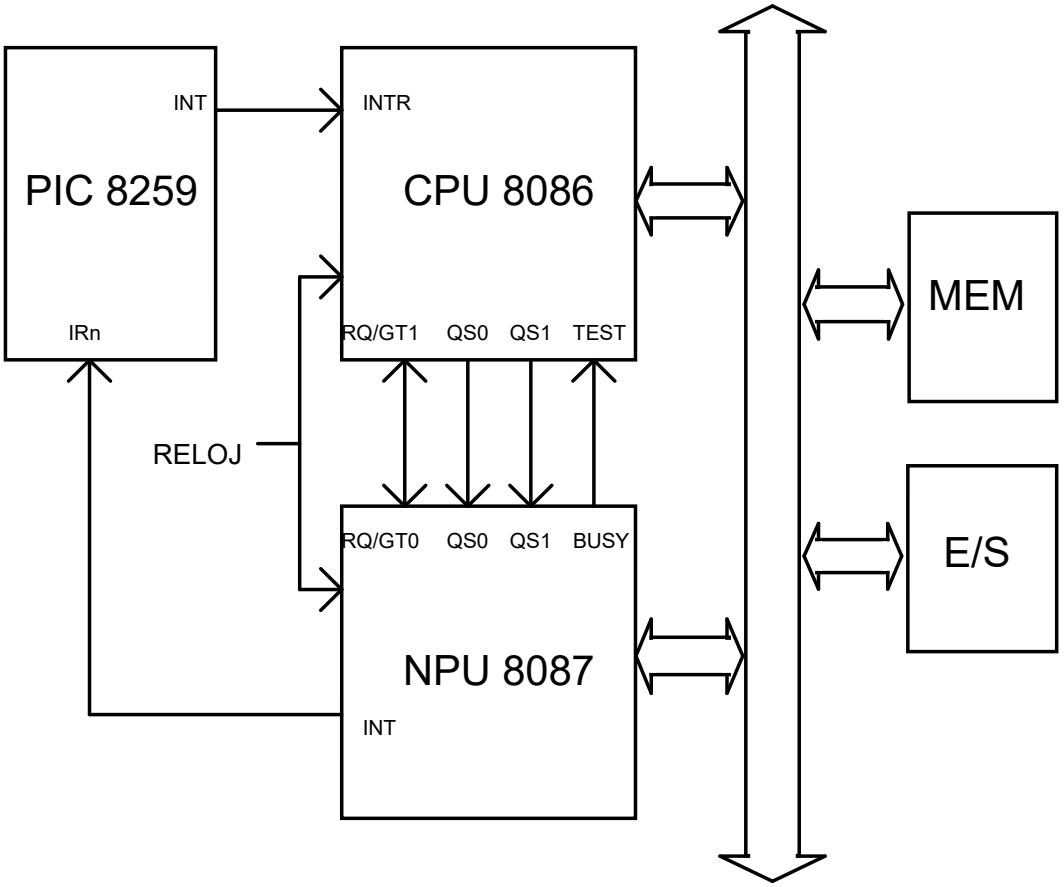


Fig. 1.11.1 - Coprocesador matemático Intel 8087

El 8087 NPX (extensión de procesamiento numérico) realiza operaciones aritméticas y de comparación sobre una variedad de tipos de datos; también ejecuta numerosas funciones trascendentes (ej.: tangente y logaritmo). El programador generalmente no percibe al 8087 como un dispositivo separado; en cambio, la capacidad de cálculo de la CPU aparece enormemente expandida.

A partir del 80486 y Pentium todos los procesadores ya incorporan el coprocesador en el mismo encapsulado, no obstante, todos los conceptos mencionados se mantienen por la compatibilidad hacia atrás que mantienen Intel y AMD en sus productos.

Para hacerse una idea de la mejora que puede se obtener, en el siguiente cuadro se comparan los tiempos de ejecución (en µS) del 8087 y el 8086 para realizar las mismas funciones.

Instrucción	*8087*	*8086*
Multiplicación	19	1600
Suma	17	1600
División	39	3200
Comparación	9	1300
Raíz cuadrada	36	19600
Tangente	90	13000

Tabla 1.11.1 - Coprocesador matemático 8087

1.11.2 Aplicaciones

- *Procesamiento de datos comercial.*
- *Robótica.*
- *Navegación.*
- *Terminales gráficas.*
- *Adquisición de datos.*
- *Control de procesos.*
- *Control numérico.*

Una instrucción numérica para el 8087 aparece como una instrucción ESCAPE para el 8086; ambos descodifican y ejecutan la misma instrucción simultáneamente. Si la instrucción ESCAPE hace referencia a memoria, la CPU direcciona y luego hace una lectura simulada, ignorando el dato, pero el NDP toma el dato y la dirección. Si el NDP requiere más bytes, gestiona el bus vía RQ/GT y con la dirección capturada previamente lee todos los datos que necesita. En siguiente ejemplo, la CPU (bloques inferiores) detecta ESCAPE simultáneamente al NDP, el cual comienza el cálculo apenas tiene los datos. La CPU continúa ejecutando instrucciones en paralelo hasta que necesita los resultados del NDP. En este punto espera (WAIT) chequeando la línea TEST el final del cálculo.

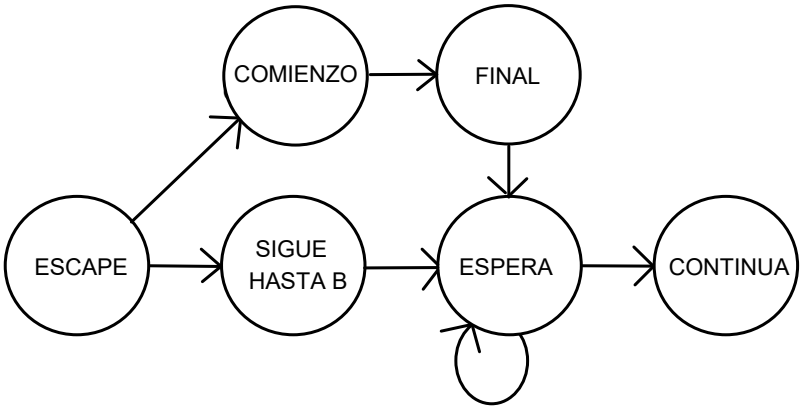


Fig. 1.11.2 – Flujo de operaciones del coprocesador matemático

En los procesadores 80486, Pentium y posteriores el coprocesador matemático se encuentra en el mismo encapsulado que el procesador y para el programador resulta simplemente un procesador con más instrucciones.