

Arquitectura de Sistemas

Soporte hardware

Gustavo Romero López

Updated: 14 de febrero de 2019

Arquitectura y Tecnología de Computadores

Índice

1. Motivación

2. Clasificación

3. Componentes

3.1 Procesador

3.2 Memoria

4. Interacción

Lecturas recomendadas

Bacon Operating Systems (1-3)

Silberschatz Fundamentos de Sistemas Operativos (1)

Stallings Sistemas Operativos (1)

Tanuenbaum Sistemas Operativos Modernos (1)

Fenómeno: gestión de memoria

matriz

```
int matriz[10000][10000];
```

inicialización a cero: **bien**

```
for (unsigned i = 0; i < 10000; ++i)
    for (unsigned j = 0; j < 10000; ++j)
        matriz[i][j] = 0;
```

inicialización a cero: **mal**

```
for (unsigned i = 0; i < 10000; ++i)
    for (unsigned j = 0; j < 10000; ++j)
        matriz[j][i] = 0;
```

- ⑤ ¿Habrá alguna diferencia al ejecutar estos bucles?

Comportamiento en tiempo de ejecución

- ◎ Para una matriz 10000 x 10000:

	matriz[i][j]		matriz[j][i]	
Mac OS X 1.25GHz PPC G4 512MB RAM	user	0.45	user	17.413
	system	1.15	system	2.037
	real	1.84	real	20.097
Linux 2 x 3GHz P4 Xeon 4 GB RAM	user	0.42	user	12.25
	system	0.73	system	0.733
	real	1.2	real	12.99
Sun OS 2 x 1GHz UltraSparc 8 GB RAM	user	1.645	user	45.495
	system	0.89	system	0.885
	real	2.87	real	47.725
Windows XP 2 x 3GHz P4 Xeon 4 GB RAM	user	0.843	user	9.937
	system	0.25	system	0.250
	real	1.125	real	10.344
Linux (casa) 2GHz AMD Athlon64 1 GB RAM	user	0.224	user	13.317
	system	0.524	system	0.660
	real	0.799	real	14.554

Causa: gestión de memoria en C

- C/C++ almacenan las matrices por filas:
ejemplo: int m[4][4];

m[0][0]	m[0][1]	m[0][2]	m[0][3]
m[1][0]	m[1][1]	m[1][2]	m[1][3]
m[2][0]	m[2][1]	m[2][2]	m[2][3]
m[3][0]	m[3][1]	m[3][2]	m[3][3]

- Tras acceder a m[0][0], ¿cuánto se tarda en acceder a m[0][1]? ¿Y a m[1][0]?
- En teoría mientras haya suficiente memoria RAM disponible el resultado debería ser el mismo.
- ¿Pasaría lo mismo en el 8086 original que un procesador actual?
- ¿Y si la memoria no es suficiente o está ocupada?

n + 0	m[0][0]
n + 1	m[0][1]
n + 2	m[0][2]
n + 3	m[0][3]
n + 4	m[1][0]
n + 5	m[1][1]
n + 6	m[1][2]
n + 7	m[1][3]
n + 8	m[2][0]
n + 9	m[2][1]
n + 10	m[2][2]
n + 11	m[2][3]
n + 12	m[3][0]
n + 13	m[3][1]
n + 14	m[3][2]
n + 15	m[3][3]

Memoria virtual

- ◎ ¿Cómo ejecutar grandes programas en memorias pequeñas?
- ◎ Truco:
 - Enviar partes no utilizadas de aplicaciones a disco.
 - Se cargan en RAM bajo demanda.
 - Esta carga bajo demanda es **muy lenta**.
- ◎ \Rightarrow Evitar los saltos lejanos.
- ◎ Observación:
 - **m[j][i]** no evita los saltos lejanos.
 - **m[i][j]** sigue el principio de localidad espacial.

¿Por qué recordar como es el hardware?

- ◎ El SO está bajo el control del hardware.
 - Parte del núcleo del SO depende del hardware.
- ◎ El hardware ayuda a controlar el SO.
- ◎ El hardware moderno es paralelo \implies el SO debe gestionar el paralelismo.
- ◎ Aplicaciones de tiempo real:
 - Necesitan soporte hardware adicional.
 - Controlan hardware especial.

Clasificación de arquitecturas paralelas (Flynn)

	flujo de instrucciones simple	flujo de instrucciones múltiple
flujo de datos simple	SISD (PC)	MISD (poco frecuente)
flujo de datos múltiple	SIMD (procesador vectorial)	MIMD (PC)

Clasificación “práctica” de arquitecturas paralelas

- ◎ Multiprocesadores de memoria compartida:
 - SMT → 2 ··· 8 hebras
 - SMP → 2 ··· 64 procesadores
 - UMA/ccNUMA → 2 ··· 64 procesadores
- ◎ Multiprocesadores masivamente paralelos:
 - NUMA/ccNUMA
 - Paso de mensajes/NORMA
- ◎ Cluster → +10M procesadores
 - GPUs

Arquitectura de computadores

◎ Sistemas monoprocesador.

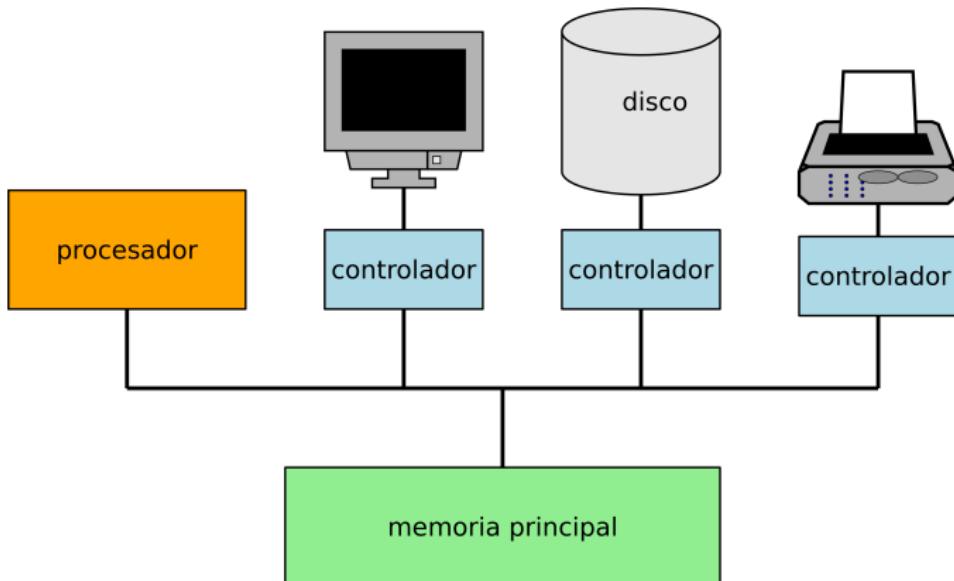
- Bus único.
- Buses separados/especializados.

◎ Sistemas multiprocesador.

- Multiproceso simétrico (SMP).
- Multihebra simultánea (SMT).
- Multinúcleo (SMP).

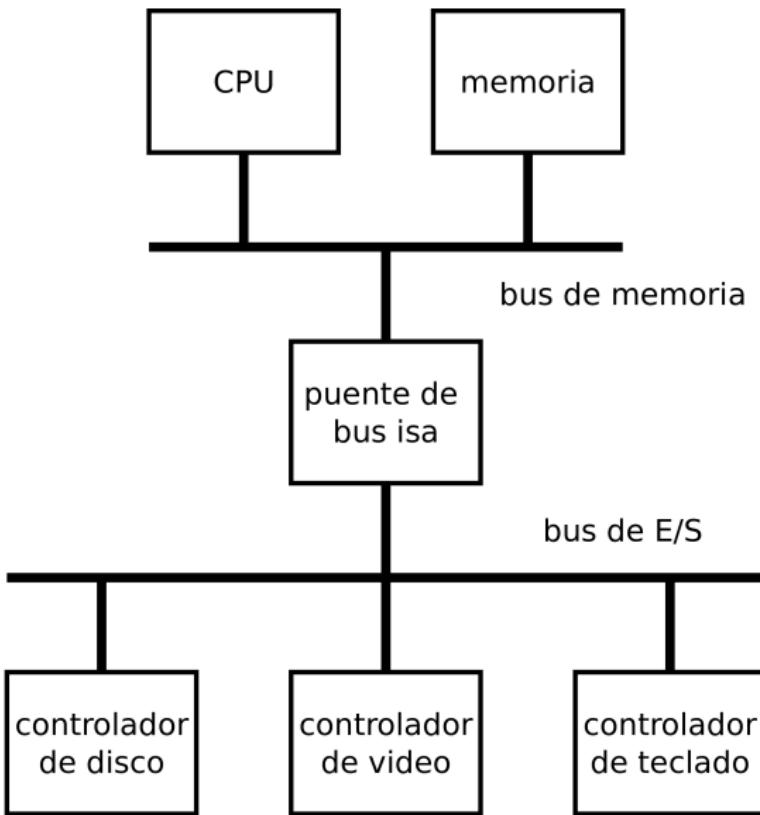
◎ Sistemas distribuidos.

Sistemas monoprocesador

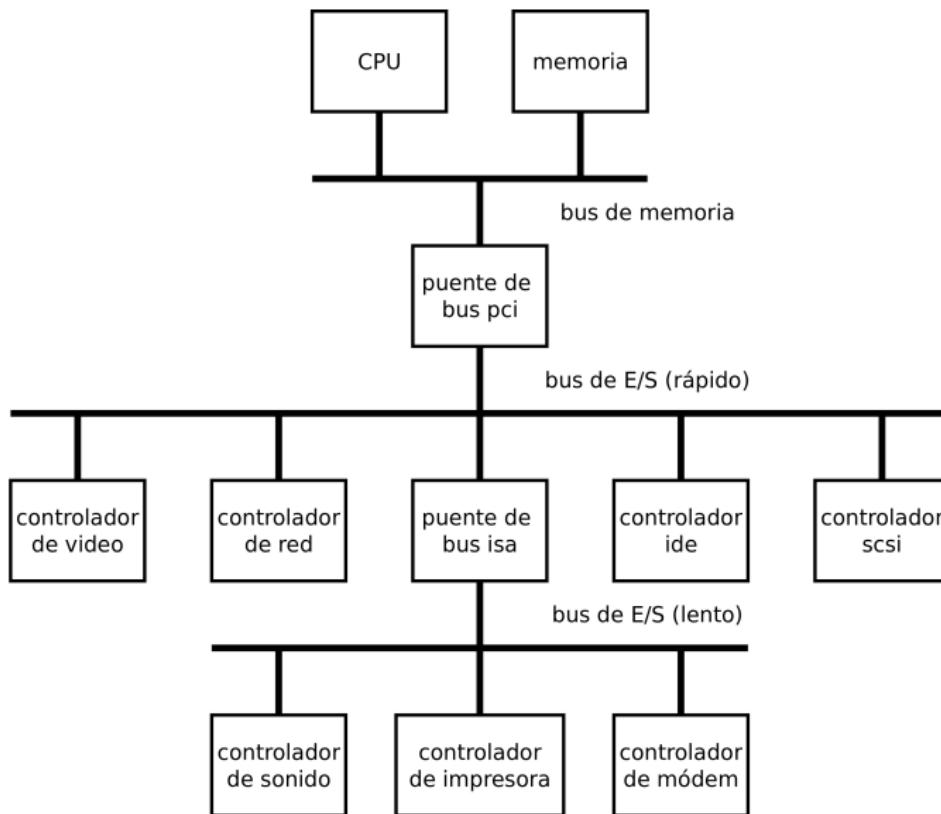


- ◎ El modelo más simple conecta todos los elementos del ordenador a un bus común.
- ◎ ¿Puede esto acarrear algún inconveniente?

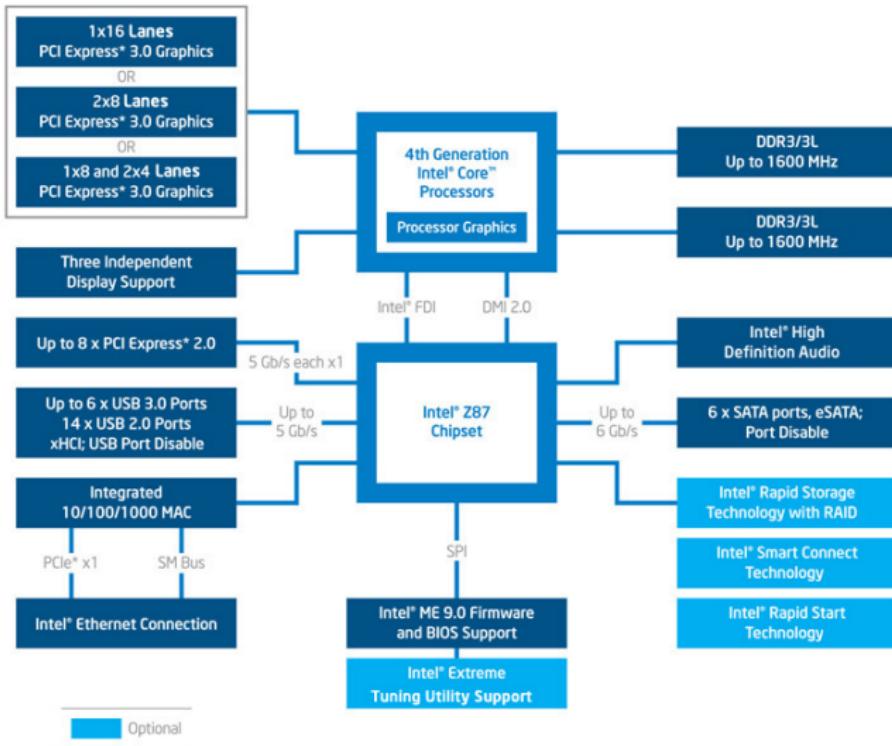
Sistemas monoprocesador



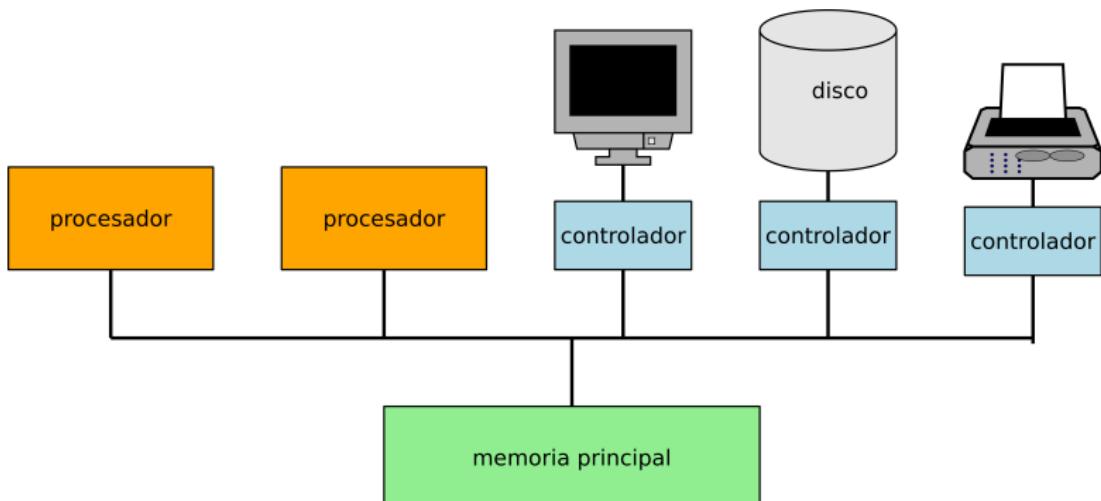
Sistemas monoprocesador



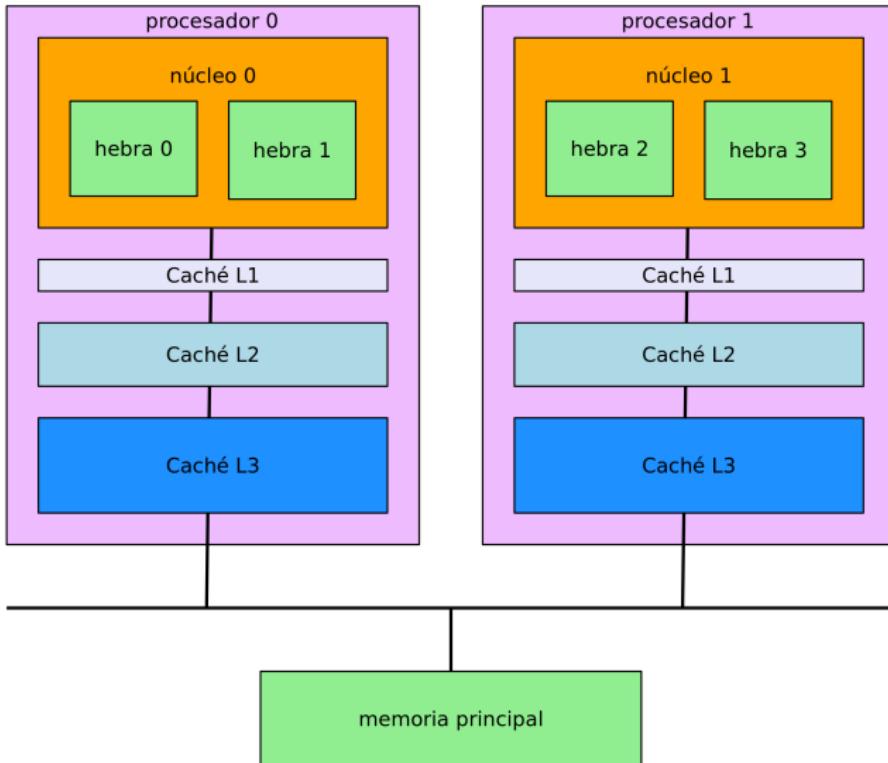
Sistemas monoprocesador: chipset Intel Z87



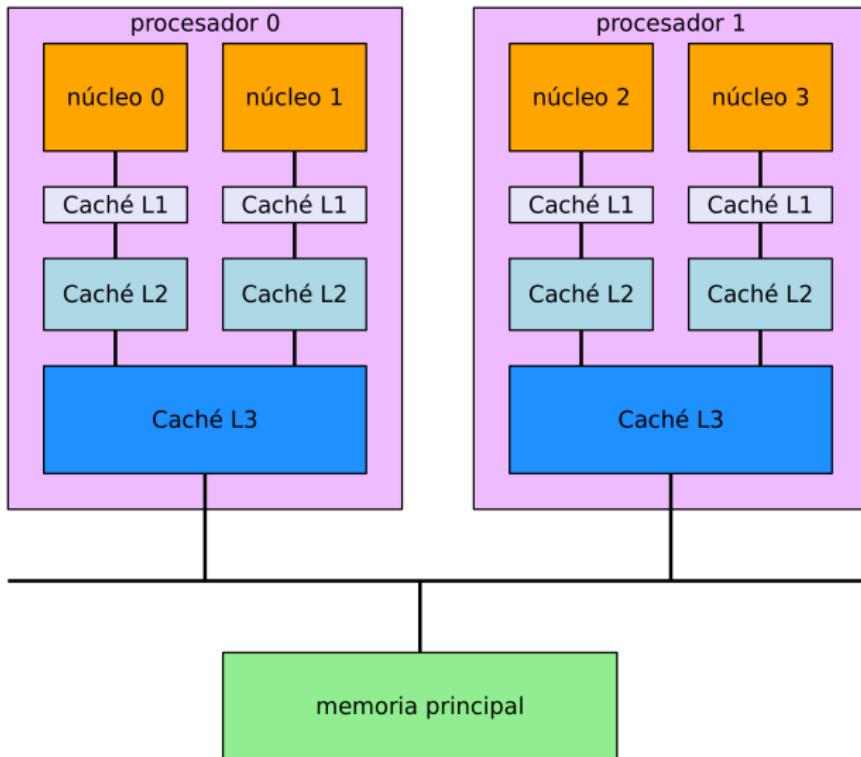
Sistemas multiprocesador: multiproceso simétrico



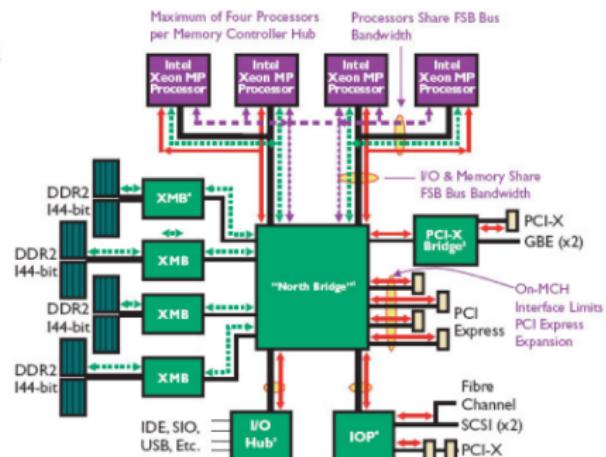
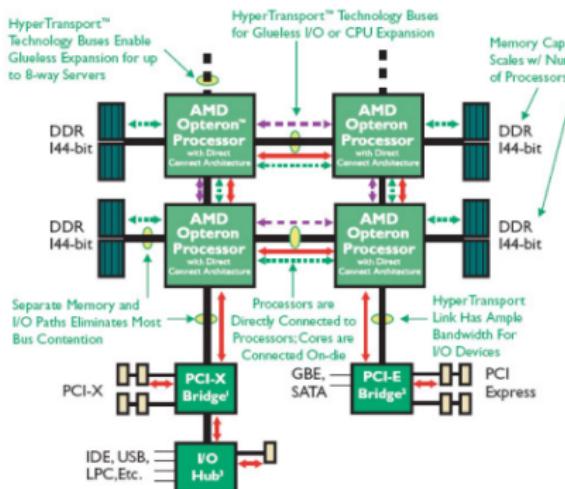
Sistemas multiprocesador: multihebra simultánea



Sistemas multiprocesador: multiproceso simétrico



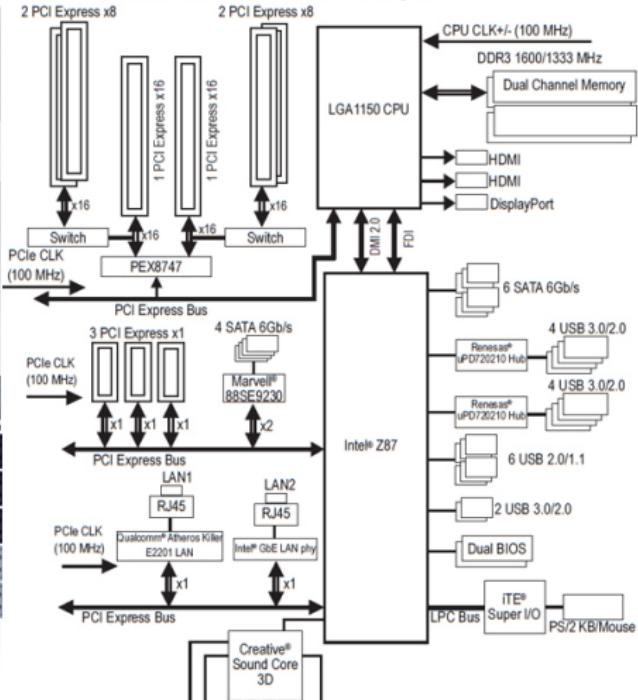
Sistemas multiprocesador



Arquitectura de un sistema actual



G1.Sniper 5 Motherboard Block Diagram



Componentes básicos

- ◎ Procesadores
- ◎ Jerarquía de memoria: caches, RAM, discos,...
- ◎ Buses de interconexión: AGP, Hypertransport, IDE, IEEE 1394, ISA, PCI, PCIe, SATA, SCSI, USB,...
- ◎ Entrada/Salida: controladores, canales de DMA, procesadores de E/S,...
- ◎ Periféricos: altavoz, disco, impresora, micrófono, monitor, ratón, teclado,...

Interfaz del procesador

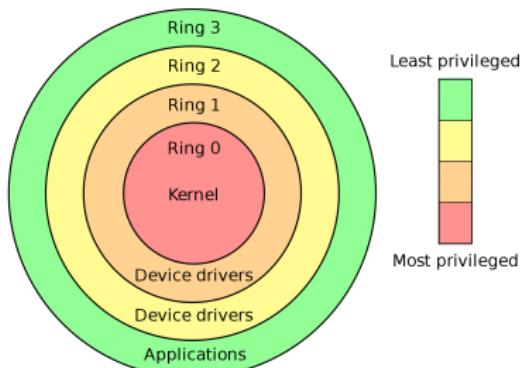
◎ Conjunto de instrucciones

- transferencia: in, mov, out,...
- modificación: add, and, div, mul, or, sub,...
- control: cli, sti,...

◎ Registros generales y especiales

◎ Al menos dos modos de ejecución con diferentes privilegios:

- privilegiado: acceso completo
- no privilegiado: acceso limitado ⇒ **excepción**



Registros del procesador: familia 80x86

ZMM0	YMM0	XMM0	ZMM1	YMM1	XMM1	ST(0)	MM0	ST(1)	MM1	EAX	RAX	R8	RBW	RBD	R12	R12W	R12D	CR0	CR4		
ZMM2	YMM2	XMM2	ZMM3	YMM3	XMM3	ST(2)	MM2	ST(3)	MM3	EBX	RBX	R9	RBW	RSD	R13	R13W	R13D	CR1	CR5		
ZMM4	YMM4	XMM4	ZMM5	YMM5	XMM5	ST(4)	MM4	ST(5)	MM5	ECX	RCX	R10	RBW	R10D	R14	R14W	R14D	CR2	CR6		
ZMM6	YMM6	XMM6	ZMM7	YMM7	XMM7	ST(6)	MM6	ST(7)	MM7	EDX	RDX	R11	RBW	R11D	R15	R15W	R15D	CR3	CR7		
ZMM8	YMM8	XMM8	ZMM9	YMM9	XMM9	CW	FP_IP	FP_DP	FP_CS	SI	ESI	RSI	SP	ESP	RSP	MSW	CR9	CR3	CR8		
ZMM10	YMM10	XMM10	ZMM11	YMM11	XMM11	SW											CR10				
ZMM12	YMM12	XMM12	ZMM13	YMM13	XMM13	TW											CR11				
ZMM14	YMM14	XMM14	ZMM15	YMM15	XMM15	FP_DS											CR12				
ZMM16	ZMM17	ZMM18	ZMM19	ZMM20	ZMM21	ZMM22	ZMM23	FP_OPC	FP_DP	FP_IP	CS	SS	DS	GDTR	IDTR	DR0	DR6	CR13			
ZMM24	ZMM25	ZMM26	ZMM27	ZMM28	ZMM29	ZMM30	ZMM31				ES	FS	GS	TR	LDTR	DR1	DR7	CR14			
															RFLAGS	EFLAGS	FLAGS	DR2	DR8	CR15	MXCSR
																DR3	DR9				
																DR4	DR10	DR12	DR14		
																DR5	DR11	DR13	DR15		

Necesidad del los niveles de privilegio

- ◎ ¿Qué hace diferente al núcleo del SO?
 - Sólo el núcleo puede ejecutar instrucciones privilegiadas.
- ◎ Ejemplos de instrucciones privilegiadas:
 - Acceso a los dispositivos de E/S: consultar el estados de los dispositivos de E/S, llevar a cabo DMA, atrapar interrupciones.
 - Manipular la unidad de gestión de memoria (MMU): manipular las tablas de segmentos y páginas, cargar y vaciar el búfer de traducción anticipada (TLB).
 - Configurar varios modos de funcionamiento: nivel de prioridad de interrupciones, alterar el vector de interrupción.
 - Utilizar la instrucción halt para activar el modo de ahorro de energía.
- ◎ El procesador comprueba el nivel de privilegio en la ejecución de cada instrucción.

Como cambiar de nivel de privilegio

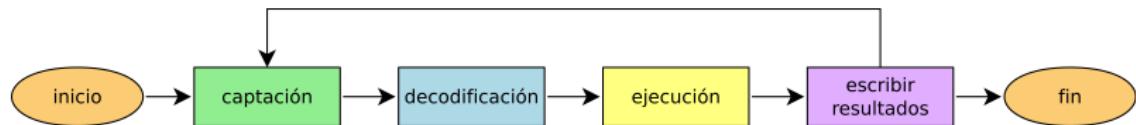
◎ **Usuario ⇒ Núcleo:** ¿Cómo obtiene el control el núcleo?

- Al arrancar.
- Llamada al sistema.
- Interrupción hardware.
- Excepción.

◎ **Núcleo ⇒ Usuario:** ¿Cómo obtiene el control una aplicación?

- EL SO prepara el entorno necesario para que la aplicación comience su ejecución.
- El SO termina alguna de sus actividades y devuelve el control a la aplicación.

Ciclo de instrucción básico



- ◎ El procesador capta una instrucción desde la memoria.
- ◎ La instrucción debe ser decodificada para averiguar su tipo.
- ◎ Conocido el tipo puede ser necesario captar nuevos operandos.
- ◎ Se ejecuta la instrucción.
- ◎ Se almacenan los resultados de la ejecución.
- ◎ El proceso se repite hasta que finaliza el programa.

Tendencias en el diseño de procesadores

- ◎ CISC \Rightarrow RISC \Rightarrow VLIW \Rightarrow ?
- ◎ Ejecución concurrente sobre procesadores: intento de explotación del **paralelismo entre instrucciones** (ILP).
- ◎ ILP es una medida de cuántas operaciones de un programa pueden ejecutarse simultáneamente sin afectar al resultado.
- ◎ Ejemplo: ¿Qué operaciones son independientes?

```
e = a + b;  
f = c - d;  
g = a * c;  
h = e / f;
```

- ◎ h **depende** de variables cuyo cálculo puede no haber finalizado.
- ◎ Las operaciones 1, 2 y 3 **no dependen** de ninguna otra de forma que pueden realizarse en **paralelo** y en **cualquier orden**.

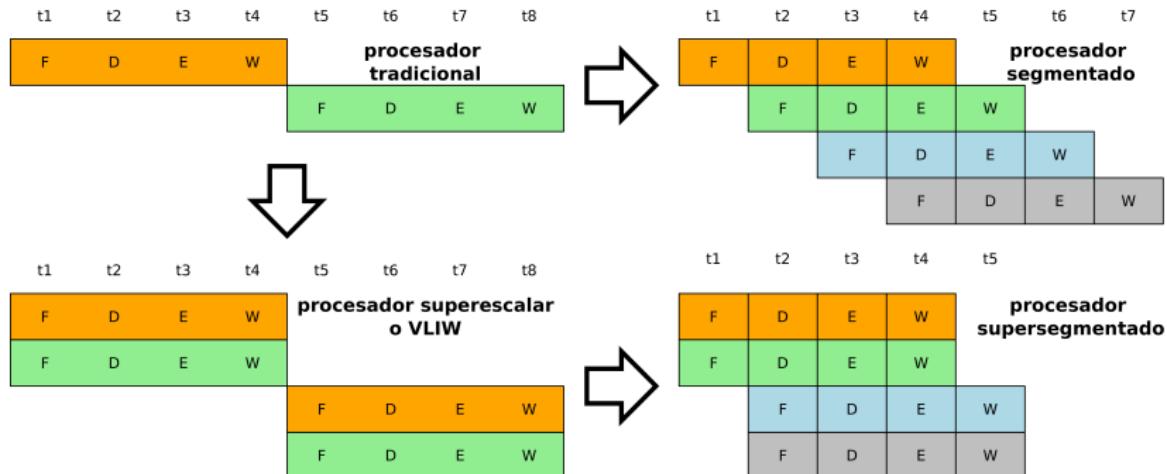
Técnicas de explotación del ILP (1)

- ◎ **Segmentación de cauce:** la ejecución de múltiples instrucciones puede solaparse total o parcialmente en el tiempo.
- ◎ **Ejecución superescalar:** múltiples unidades de ejecución se utilizan para ejecutar múltiples instrucciones en paralelo.
- ◎ **Computación con instrucciones explícitamente paralelas (EPIC):** uso del compilador en lugar de complejos circuitos para identificar y explotar el ILP.
- ◎ **Ejecución fuera de orden:** ejecución de instrucciones en cualquier orden que no viole las dependencias entre instrucciones.

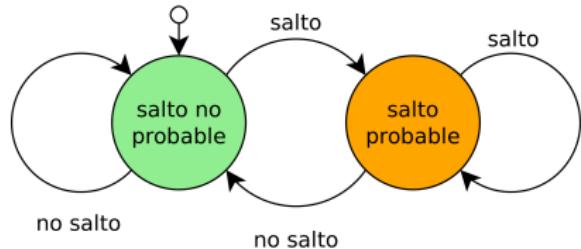
Técnicas de explotación del ILP (2)

- ◎ **Renombrado de registros:** técnica para evitar la innecesaria serialización de instrucciones por la reutilización de registros.
- ◎ **Ejecución especulativa:** permitir la ejecución de instrucciones completas, o partes, antes de conocer con seguridad si su ejecución debe tener lugar.
- ◎ **Predicción de salto:** se utiliza para evitar quedar parado antes de que se resuelvan las dependencias de control. Se utiliza en conjunción con la ejecución especulativa.
- ◎ **Multihebra simultánea (SMT):** técnica que permite la ejecución de múltiples hebras de ejecución para aprovechar mejor las unidades funcionales de los procesadores superescalares.

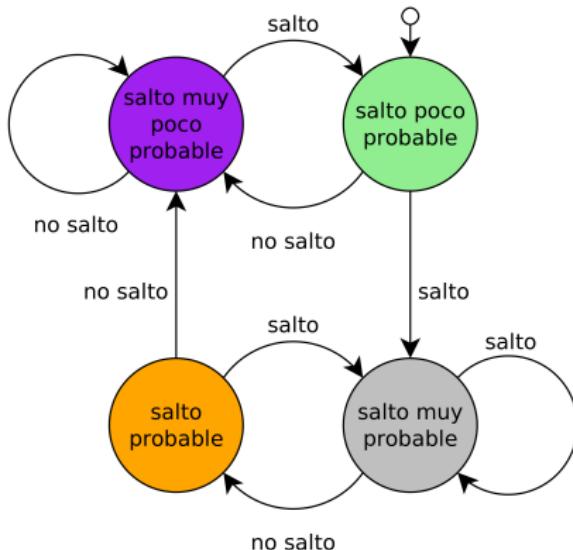
Técnicas de explotación del ILP (3)



Técnicas de explotación del ILP (4)



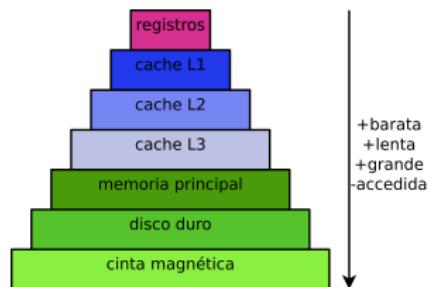
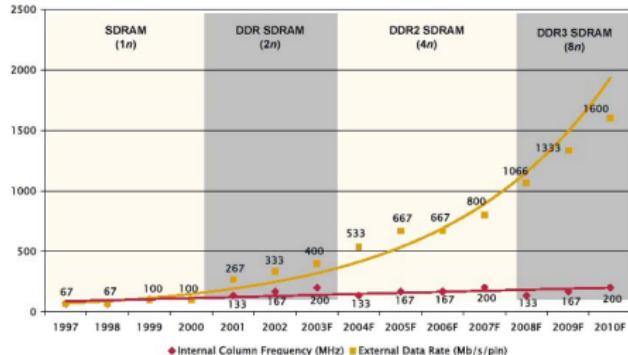
(a) Algoritmo de predicción de dos estados:
funciona bien en bucles salvo entrada y salida.



(b) Algoritmo de predicción de cuatro estados:
permite recordar estado previo de un bucle.

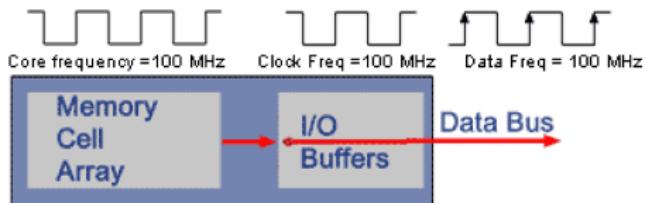
¿Por qué usar una jerarquía de memoria?

- Por la gran **diferencia de velocidad** entre procesador y memoria.
- La jerarquía de memoria puede aliviar este problema gracias a los **principios de localidad** y la **regla 90/10**:
 - Localidad **espacial**: la información a la que se accede suelen estar próxima a la que ha sido accedida con anterioridad.
 - Localidad **temporal**: la información a la que se accede una vez suele volver a ser utilizada.
 - Regla 90/10: el 10 % del código realiza el 90 % del trabajo.

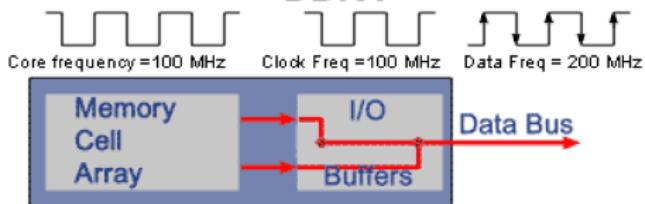


Tipos de memoria RAM

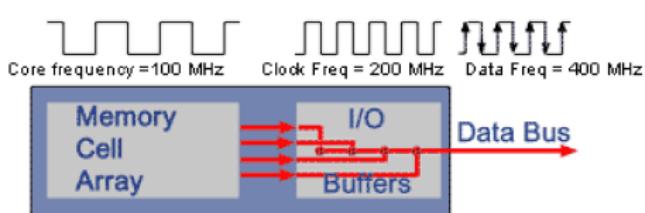
SDRAM



DDR I



DDR II



Cantidad de memoria en registros

- ◎ Tipos de registros:
 - propósito general
 - punto flotante
 - multimedia
 - especiales (ip, flags)
- ◎ RISC:
 - 32 de propósito general (32 ó 64 bits)
 - 32 de punto flotante (64 bits IEEE 754)
 - multimedia (64 ··· 256 bits)
- ◎ CISC:
 - IA32: 8 de propósito general, 8 de punto flotante, 8 multimedia
 - IA64: 128 de propósito general, 128 de punto flotante
- ◎ Algunos procesadores tiene varios conjuntos de estos registros (ventanas de registros).

Análisis de una jerarquía de 2 niveles

$$\bar{T}_{acceso} = (1 - T_{fallos}) \times T_{cache} + T_{fallos} \times (T_{cache} + T_{ram})$$

$$T_{cache} = 1\text{ns}$$

$$T_{ram} = 100\text{ns}$$

$$T_{fallos} = 5\% \implies \bar{T}_{acceso} = 0,95 \times 1 + 0,05 \times 101 = 6\text{ns}$$

$$T_{fallos} = 2\% \implies \bar{T}_{acceso} = 0,98 \times 1 + 0,02 \times 101 = 3\text{ns}$$

Parámetros de diseño de memorias caché

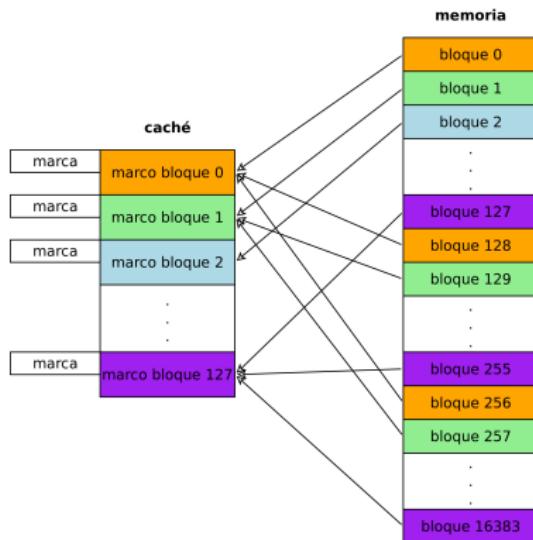
- ◎ Tamaño:
 - L1: 8 … 256KB
 - L2: 64KB … 8MB
 - L3: 1 … 16MB
- ◎ Tamaño de bloque: 32 … 128B
- ◎ Tiempo de acceso: 1 … 10ns
- ◎ Política de búsqueda:
 - bajo demanda
 - anticipativas
- ◎ Política de colocación:
 - correspondencia directa
 - asociativa por conjuntos
- completamente asociativa
- ◎ Política de reemplazo:
 - LRU
 - FIFO
 - aleatoria
- ◎ Política de actualización:
 - escritura directa
 - post-escritura
- ◎ Otras características importantes:
 - caché de víctimas
 - inclusiva/exclusiva
 - unificada/separada

Políticas de colocación: ejemplo

Ejemplo:

- ◎ memoria principal: 256K palabras → 16K bloques
- ◎ 16 palabras por bloque
- ◎ caché: 2K palabras → 128 marcos de bloque
- ◎ Bus de direcciones → ≥ 18 bits

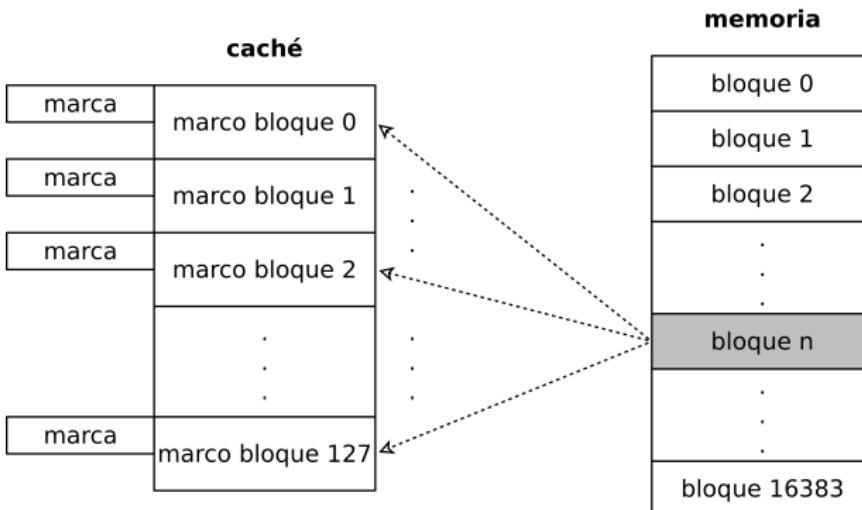
Políticas de colocación: correspondencia directa



dirección de memoria (18 bits)

17	11	10	4	3	0
marca (7 bits)		marco de bloque (7 bits)		palabra (4 bits)	

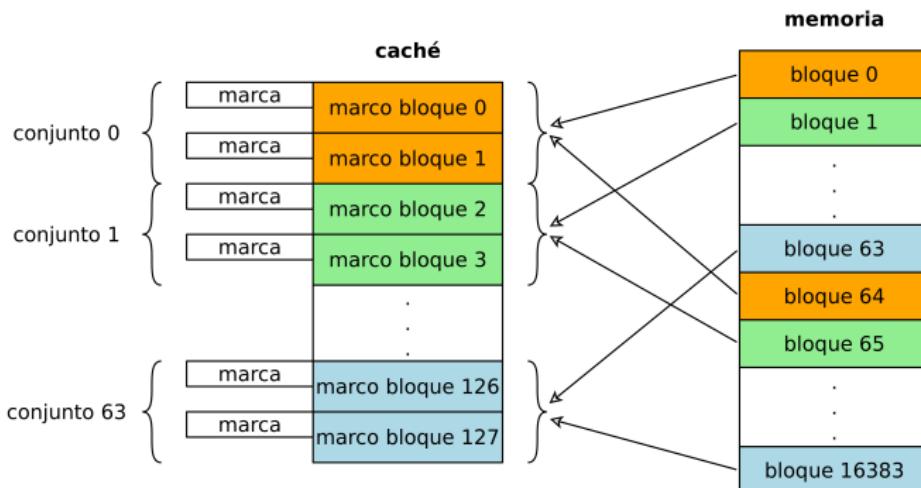
Políticas de colocación: correspondencia totalmente asociativa



dirección de memoria (18 bits)



Políticas de colocación: correspondencia asociativa por conjuntos (2 M/C)



dirección de memoria (18 bits)

17	10 9	4 3	0
marca (8 bits)	conjunto (6 bits)	palabra (4 bits)	

Ejemplos de memorias caché modernas

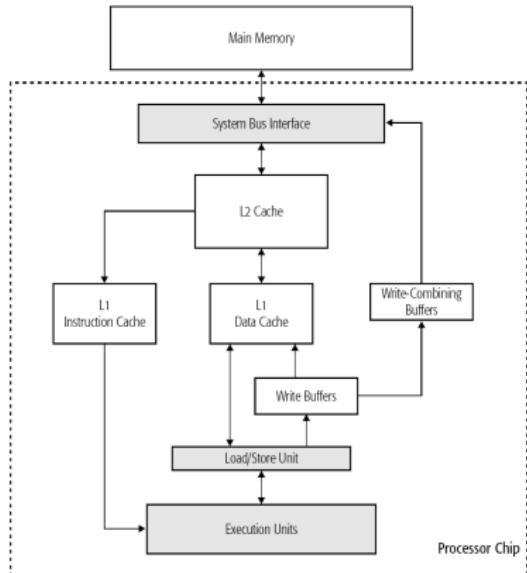


Figura: AMD K8

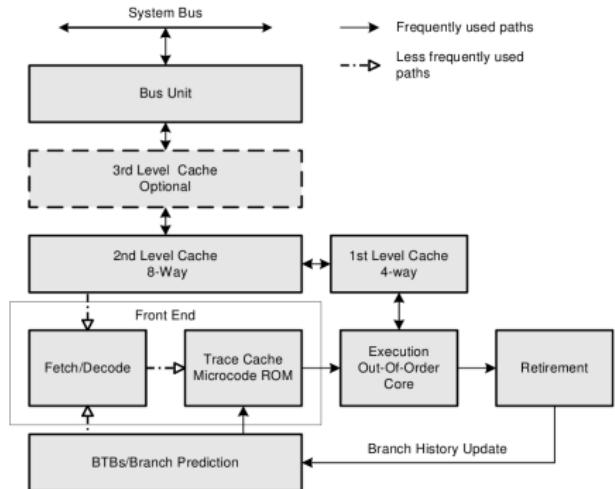


Figura: Intel P6

Protección de memoria

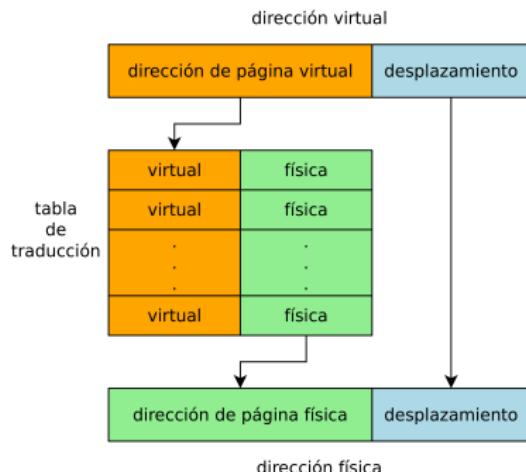
- ◎ El SO debe proteger a los programas entre sí y a él mismo de programas maliciosos o erróneos.
- ◎ Solución: **Espacios de direcciones (AS)**.
- ◎ El procesador dispone de recursos para establecer límites al espacio de direcciones de memoria accesibles por los programas: **memoria virtual**.
- ◎ Tipos (en función de cómo se representa el espacio de direcciones):
 - **Segmentación**: tamaño variable y arbitrario.
 - registro base: principio del espacio de direcciones.
 - registro límite: tamaño del espacio de direcciones.
 - **Paginación**: tamaño variable en múltiplos de página.
 - tabla de páginas: lista de páginas de un proceso.

Memoria virtual

- ◎ La mayoría de los SO asocian un espacio de direcciones diferente para cada proceso (salvo **SASOS**):
 - Ventaja: \Rightarrow **protección automática**.
 - Inconveniente: \Rightarrow dificulta la compartición.
- ◎ Las partes de un espacio de direcciones pueden colocarse en cualquier parte de la memoria física.
 - Algunos espacios de direcciones deben ocupar lugares específicos de la memoria física, ej: dispositivos de E/S.
- ◎ El procesador debe tener una **unidad de gestión de memoria (MMU)** extremadamente eficiente porque la traducción dirección **virtual \Rightarrow física** puede realizarse una o más veces para cada instrucción ($\times n$).
- ◎ Se requiere **una traducción por cada acceso** a memoria.

Traducción dirección virtual \Rightarrow dirección física

- La ejecución de una instrucción puede requerir varias traducciones, ej: `mov ax, ds:[bx + si*4 + 0x8000]`.
 - captación de la instrucción.
 - captación del dato.
 - Cada traducción puede requerir de varios accesos a memoria:
 - segmentación + paginación.
 - tablas de páginas multínivel.
- Se necesita una tabla de traducción por cada espacio de direcciones.
 - ¿Cuál es el tamaño de las tablas de páginas?



Funcionamiento de la segmentación (x86)

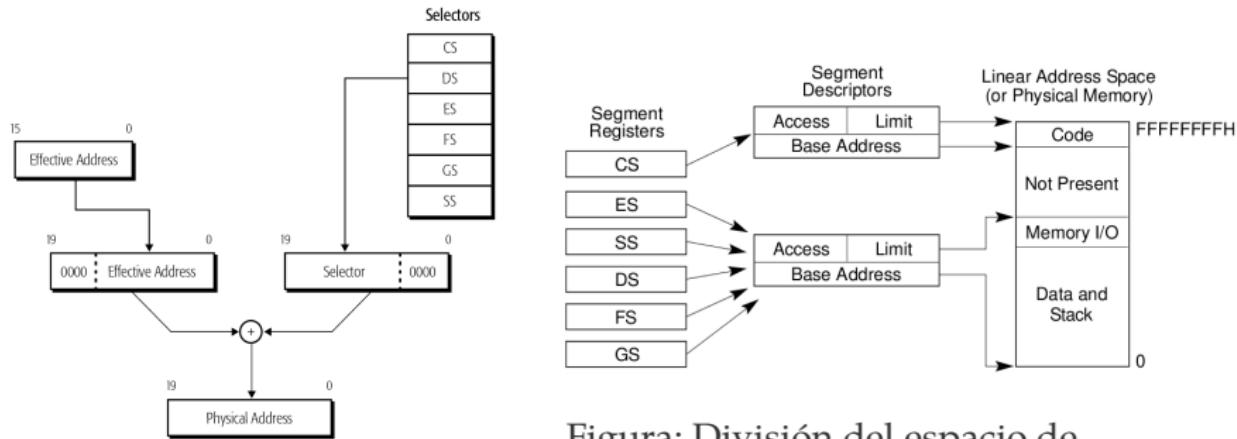


Figura: División del espacio de direcciones de un proceso.

Figura: Cálculo de la dirección física.

Segmentación + Paginación (x86)

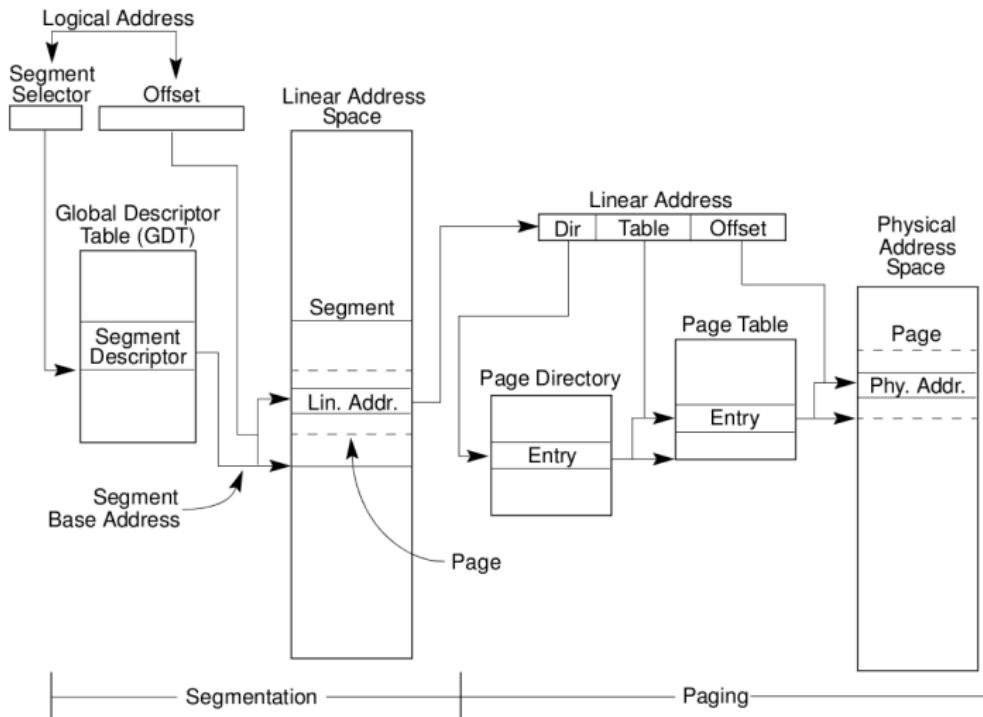
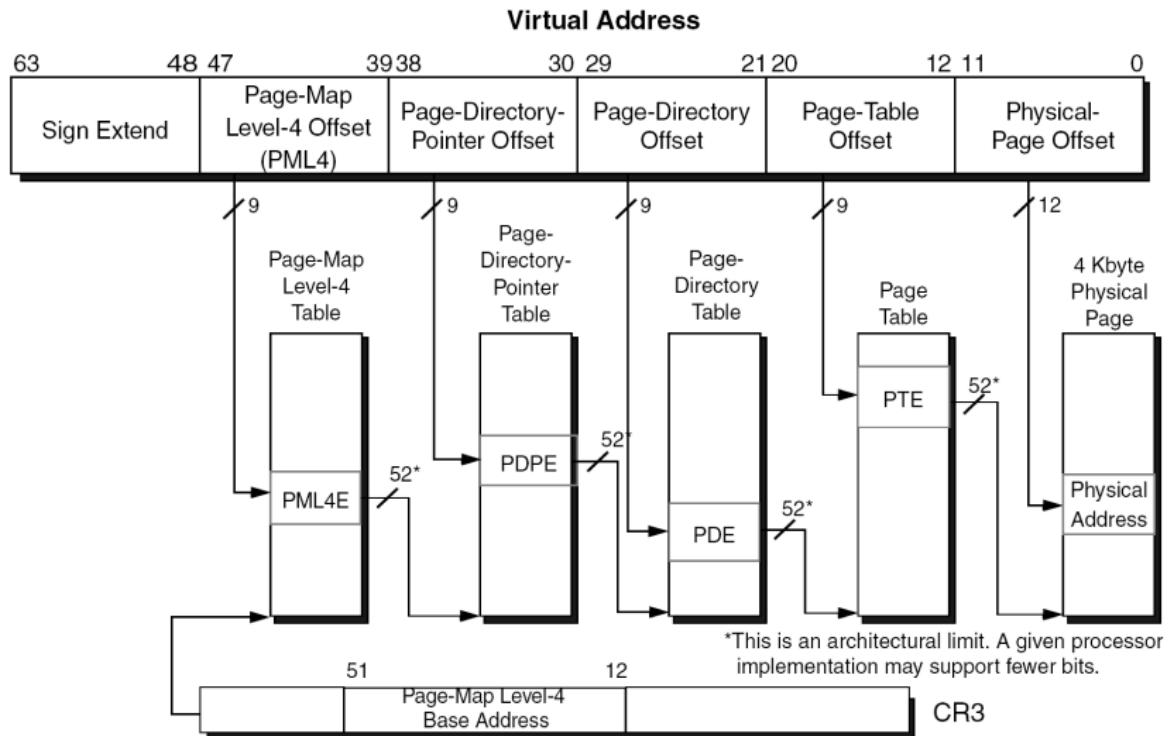


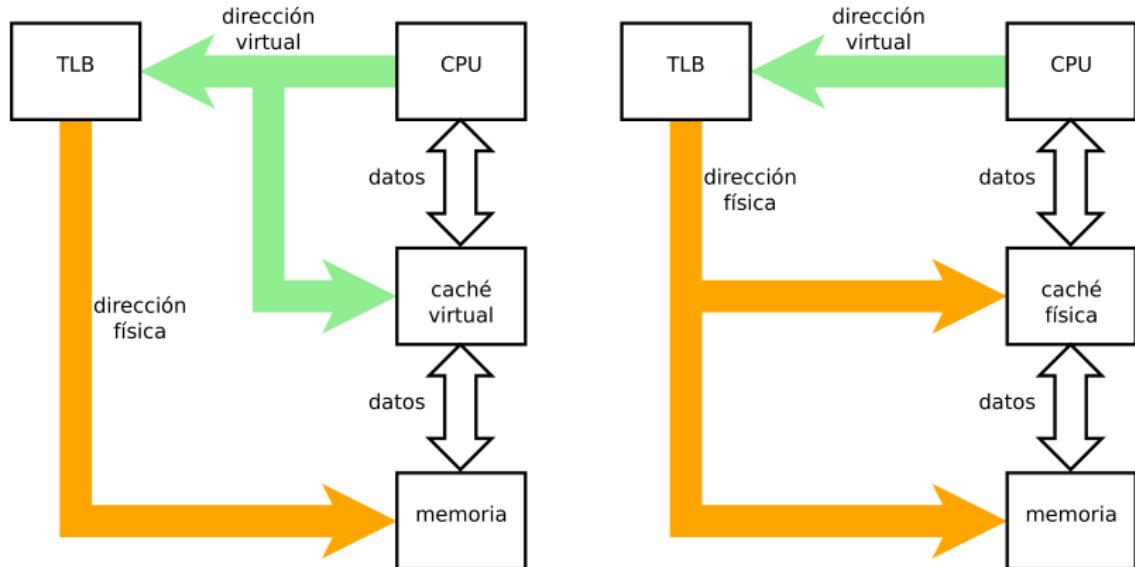
Tabla de páginas multinivel (x86_64)



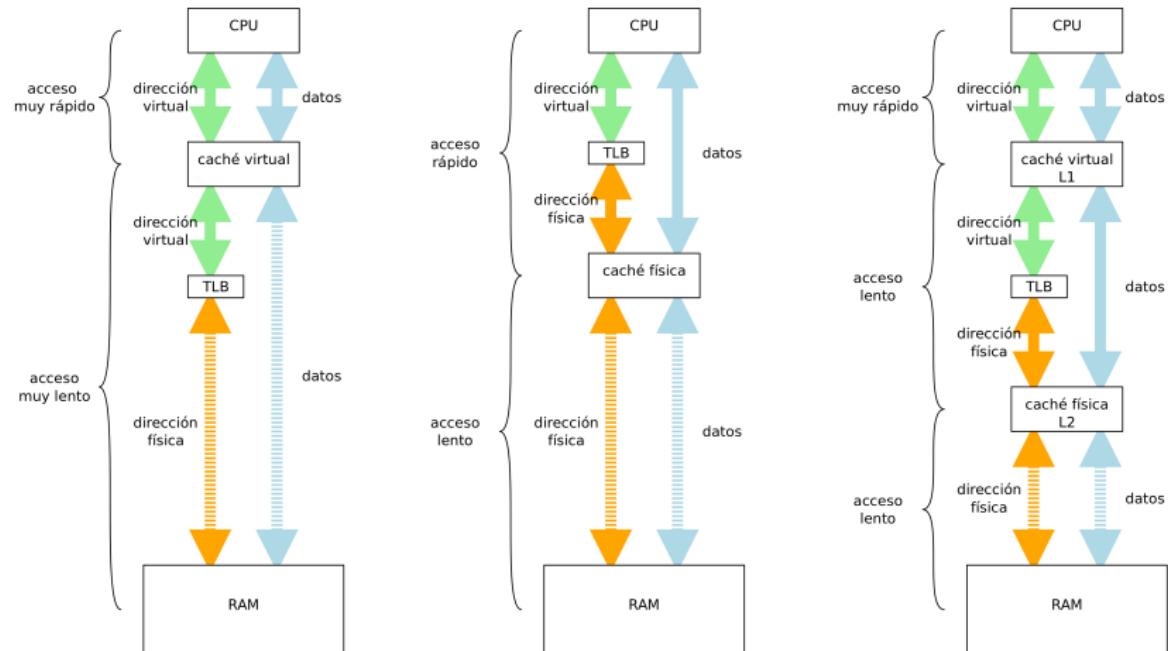
Búfer de traducción anticipada (TLB)

- ◎ Traducir direcciones por software es muy lento.
- ◎ Mediante hardware puede hacerse más rápido y además podemos añadir una caché de traducciones (TLB):
 - contenido: pares de direcciones virtual/física.
 - implementado mediante memorias asociativas.
 - tamaño típico: de 32 a 128 entradas.
- ◎ Hay dos tipos fundamentales:
 - etiquetadas: añaden una marca del espacio de direcciones al que pertenece.
 - no etiquetadas: no poseen la capacidad anterior (la mayoría).
- ◎ Funcionamiento:
 - acierto \Rightarrow se devuelve la traducción correcta.
 - fallo \Rightarrow es necesario calcular la traducción.
 - hardware: rápido pero poco flexible (x86).
 - software: más lento pero más flexible (PowerPC).
- ◎ Es fundamental conocer la interacción TLB/caché.

Esquemas de direccionamiento de caché



Esquemas de direccionamiento de caché (2)



Control de Entrada/Salida

- ◎ ¿Cómo inicia el SO una operación de E/S?
 - Instrucciones especiales: **in/out** (x86)
 - E/S en memoria: **mov** (x86)
 - acceso al hardware como direcciones de memoria
 - requiere de una MMU capaz de traducir al bus de E/S
 - Normalmente escribiendo una orden en un puerto de E/S
- ◎ ¿Como averigua el SO cuando una orden de E/S **finaliza**?
 - **sondeo**: leer el puerto de estado hasta encontrar el valor adecuado
 - **interrupción**: existe una linea física mediante la cual se avisa del fin de la operación y se cede el control al SO

Ejemplo de E/S: controlador de disco

0x1	sector
0x2	count
0x3	operation
0x4	status
0x5	data

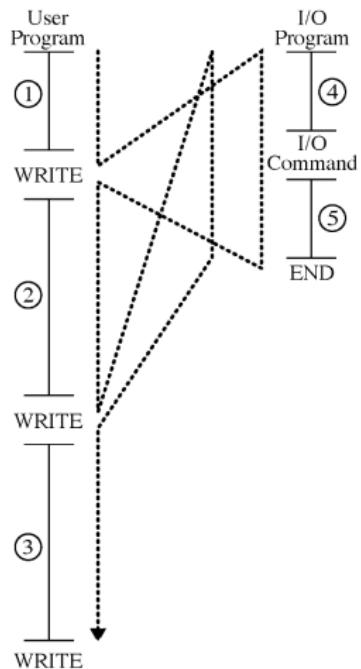
E/S independiente y lenguaje ensamblador

```
out 0x101, 23;           /* comenzar en el sector 23 */
out 0x102, 5;            /* transferir 5 sectores */
out 0x103, 1;            /* iniciar la lectura */
in ax, 0x104;            /* leer el estado */
```

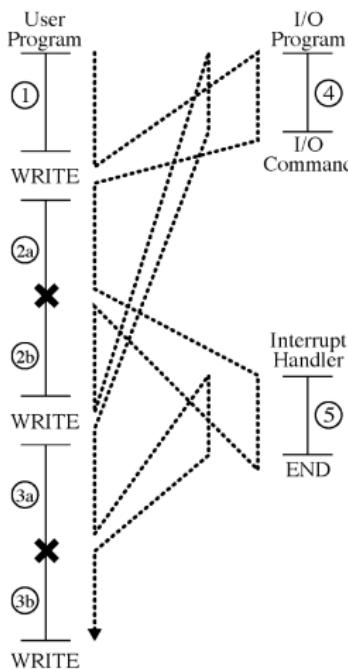
E/S mapeada en memoria y lenguaje C

```
*(char*) 0x4001 = 23;      /* comenzar en el sector 23 */
*(char*) 0x4002 = 5;       /* transferir 5 sectores */
*(char*) 0x4003 = 1;       /* iniciar la lectura */
status = *(char*) 0x4004;  /* leer el estado */
```

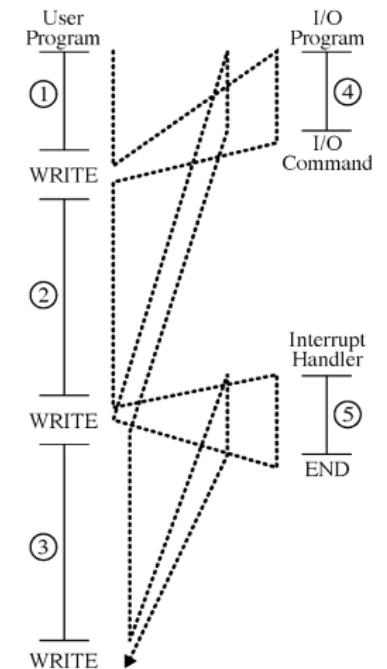
Espera del fin de la operación de E/S



(a) sondeo



(b) interrupciones: espera breve



(c) interrupciones: espera prolongada

Eventos: excepciones e interrupciones

- ◎ Existen dos clases principales de “situaciones especiales” (**eventos**):
 - **excepciones síncronas:** llamada al sistema.
 - **interrupciones asíncronas:** fin de operación de DMA.
- ◎ Hay otras diferencias como...
 - el origen.
 - la predictibilidad.
 - la reproducibilidad.
- ◎ El manejo de excepciones e interrupciones...
 - debe producirse a tiempo.
 - es específico de cada tipo de procesador.

Eventos: excepciones e interrupciones (2)

Excepciones síncronas e internas al procesador:

- ◎ origen: la instrucción actual.
- ◎ excepciones erróneas:
 - puntero inválido.
 - división entre 0.
- ◎ normalmente el programa es abortado.
- ◎ excepciones no erróneas:
 - fallo de página.
 - punto de ruptura.
- ◎ SO gestiona de forma transparente al usuario.

Interrupciones asíncronas y externas al procesador:

- ◎ origen: dispositivos de E/S, temporizador,...
- ◎ no están relacionadas con el flujo de instrucciones.
- ◎ la mayoría de las interrupciones están causadas por la finalización de operaciones de E/S.
- ◎ algunas interrupciones están causadas por fallos en dispositivos (atasco de papel).

Interrupciones software (int/trap)

- ◎ Mecanismo para ceder el control al SO elevando el nivel de privilegio.
- ◎ Síncronas, predecibles, reproducibles.
- ◎ Un mismo programa aislado siempre provoca las mismas:
 - instrucción desconocida
 - instrucción errónea (división entre 0).
 - modo de direccionamiento erróneo.
 - violación del espacio de direcciones.
 - llamada al sistema.
 - fallo de página (sólo políticas de paginación locales!).
- ◎ No puede evitarse sin haber gestionado la causa ⇒ comportamiento erróneo.
- ◎ Requiere tiempo y espacio que son una sobrecarga para el sistema ya que no realizan trabajo útil.

Interrupciones

- ◎ Mecanismo para solicitar la atención de la CPU.
- ◎ Asíncronas, no predecibles, no reproducibles.
- ◎ Un periférico puede solicitar una interrupción independientemente del estado de la CPU o el proceso actual.
- ◎ Ejemplos:
 - eventos externos: sensores.
 - final de una operación de DMA.
 - final de una operación de E/S.

Vector de interrupciones

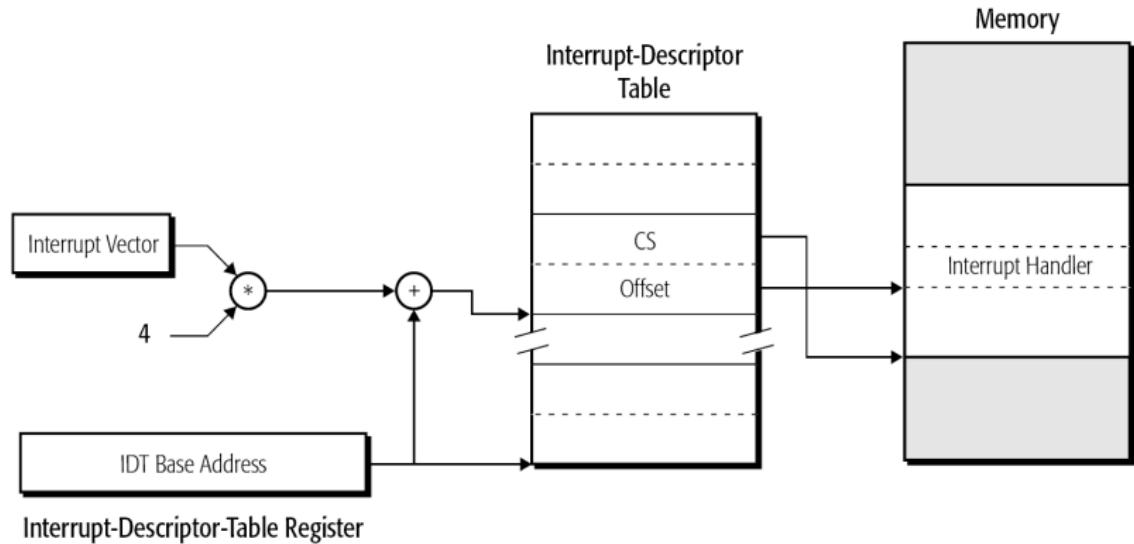


Figura: Vector de interrupciones (procesadores 80x86).

Manejo de excepciones e interrupciones

- ◎ Eventos que podría lanzar una excepción o una interrupción:
 - Señales de periféricos:
 - final de una lectura de disco.
 - atasco de papel.
 - Cambio del dominio de protección
(segmentación/paginación).
 - Errores de programación (acceso a dirección inválida).
 - Desbordamiento de memoria (desbordamiento de pila).
 - Paginación bajo demanda.
 - Fallos hardware (paridad de memoria).
- ◎ El manejo del evento se hace durante la ejecución del proceso actualmente en ejecución.

Modelos de manejo de eventos

- ◎ Modelo de **reanudación**:
 - una vez manejado el evento el programa interrumpido puede reanudar su ejecución desde la misma instrucción.
- ◎ Modelo de **terminación**:
 - en el caso de eventos que no pueden ser manejados el programa interrumpido es abortado.
- ◎ Lanzar un evento provocará un **cambio de contexto**
 - programa en ejecución \implies manejador del evento.
 - se debe almacenar el contexto del programa interrumpido para poder reanudarlo.
 - objetivo: reducir la **sobrecarga** del almacenamiento y restauración del contexto.

Efectos sobre la temporización

- ◎ El manejo de cada evento **ralentiza** la ejecución del programa interrumpido.
- ◎ El resultado de los programas no se pone en peligro, salvo...
- ◎ ...las aplicaciones de **tiempo real** que si pueden **fallar**.
- ◎ Esta es una de las razones por las que los programadores no deben confiar en las condiciones de temporización (ej: sincronización).

Control de interrupciones (1)

- ◎ Las interrupciones pueden llegar en cualquier momento:
 - usuario modificando datos compartidos.
 - SO realizando una operación no interruptible.
- ◎ SO y hardware permiten sincronizar actividades concurrentes mediante **operaciones atómicas**: secuencias de instrucciones que no pueden ser interrumpidas.
- ◎ Soluciones:
 - deshabilitar las interrupciones ¹.
 - instrucciones atómicas basadas en leer/modificar/almacenar:
 - **tas**: test and set.
 - **cmpxchg**: comparar e intercambiar.
 - **ldl/stc**: carga enlazada y almacenamiento condicional.

¹¿Funcionaría en un multiprocesador?

Control de interrupciones (2)

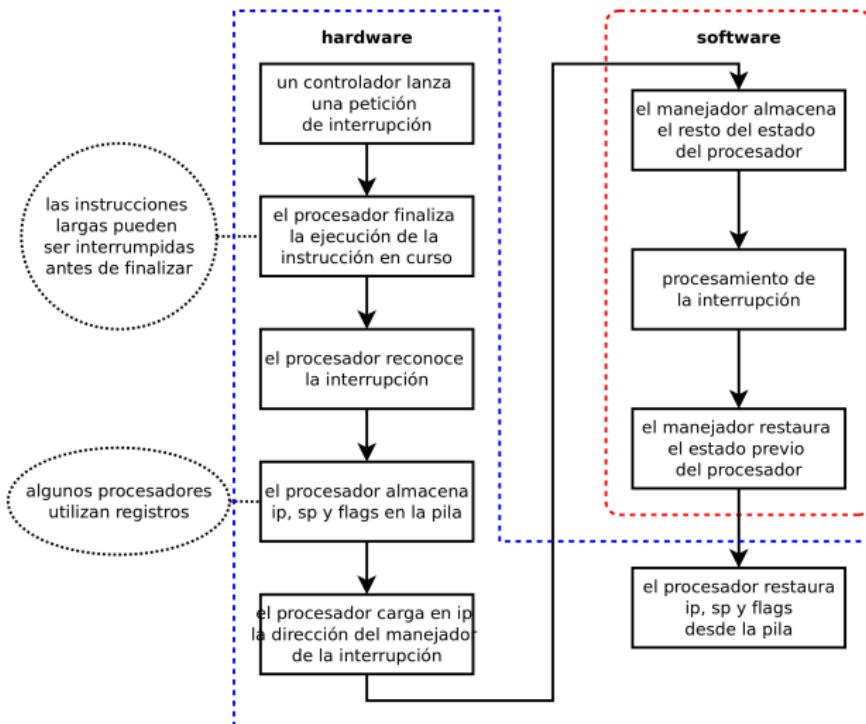
- ◎ La mayoría de los ordenadores permiten que los controladores de E/S interrumpan la actividad del procesador.
- ◎ La CPU transfiere el control a un **manejador de interrupción** que normalmente es parte del SO (*excepción: controladores en espacio de usuario*).
- ◎ El procesador puede prevenir las interrupciones...
 - enmascarándolas.
 - deshabilitándolas.
 - estableciendo un **nivel mínimo de prioridad**.

Manejo de interrupciones

- ◎ Una interrupción provoca la detención del programa en ejecución y la ejecución de su correspondiente manejador de interrupción.
- ◎ Al finalizar el manejador de interrupción *podría*² devolver el control al programa interrumpido
- ◎ Como la interrupción puede llegar en cualquier momento de la ejecución de un programa el hardware o el manejador de la interrupción debe guardar el estado del programa interrumpido (estado del procesador) para poder restaurarlo posteriormente.

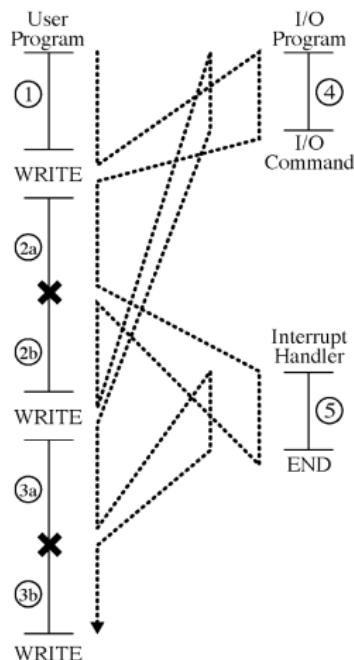
²¿Qué otro programa podría ejecutarse?

Procesamiento de interrupciones



Ventajas del uso de interrupciones

- ◎ Los eventos son atendidos más rápidamente.
- ◎ No se consume tiempo del procesador para descubrir el final de un evento.
- ◎ Los programas pueden ejecutarse más rápidamente.
- ◎ Se mejora el aprovechamiento del procesador.
- ◎ => pero... aún es el procesador el encargado de realizar las transferencias de información :(
- ◎ => mejor utilizar DMA para evitarlo :)



Múltiples interrupciones

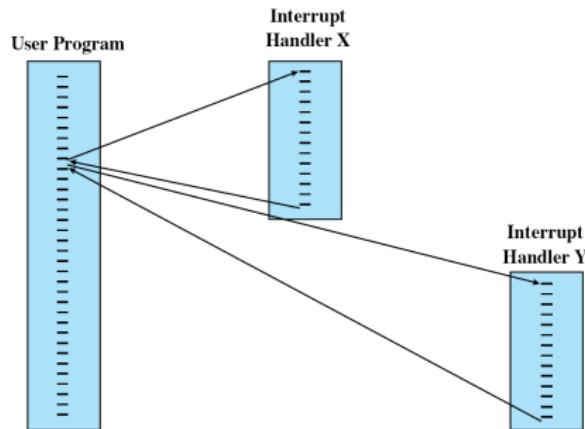


Figura: Procesamiento secuencial
(interrupciones deshabilitadas).

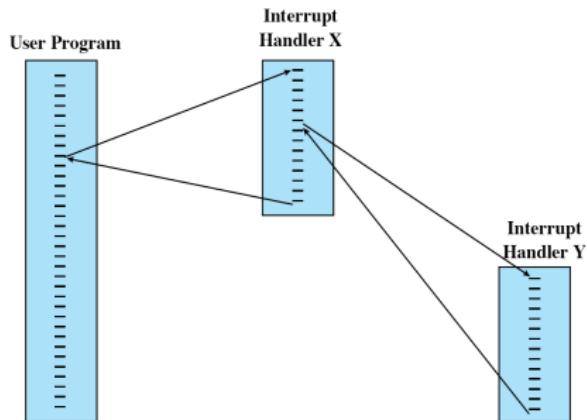


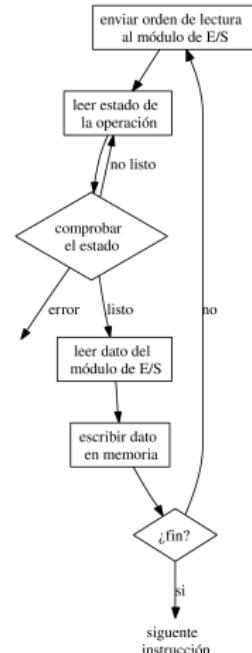
Figura: Procesamiento anidado
(interrupciones con prioridades).

Técnicas de Entrada/Salida

- ◎ E/S programada (sondeo).
- ◎ E/S dirigida mediante interrupción.
- ◎ Acceso directo a memoria (DMA).

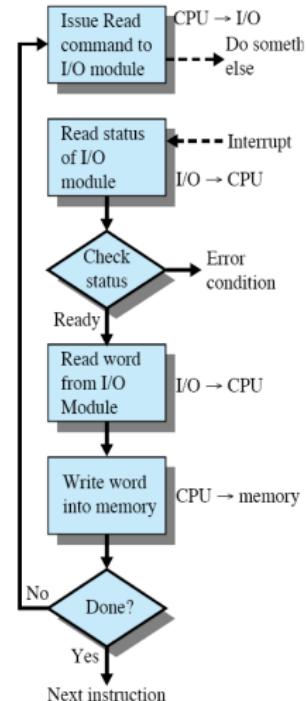
E/S programada

- ◎ El módulo de E/S realiza su trabajo en mitad del proceso en ejecución.
- ◎ El módulo de E/S no necesita interrumpir al procesador al finalizar ⇒ en realidad no lo deja libre.
- ◎ El procesador se mantiene doblemente ocupado:
 - comprobando el estado del módulo de E/S.
 - realizando la transferencia de información.



E/S dirigida mediante interrupción

- ⑤ El procesador es interrumpido cuando el módulo de E/S está preparado para realizar la transferencia.
 - ejemplo: al finalizar una operación de lectura.
- ⑥ EL procesador es libre de realizar cualquier otra tarea mientras tanto \Rightarrow no se produce una **espera ocupada**.
 - Apropiado para eventos esporádicos o de grano medio.
 - Inapropiado para eventos de grano fino por el alto coste de interrupción:
 - Una interrupción por cada paquete de red puede ser aceptable.
 - Una interrupción por byte sería demasiado costoso.

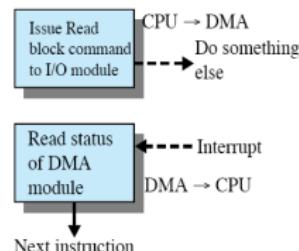


E/S mediante acceso directo a memoria

◎ Funcionamiento:

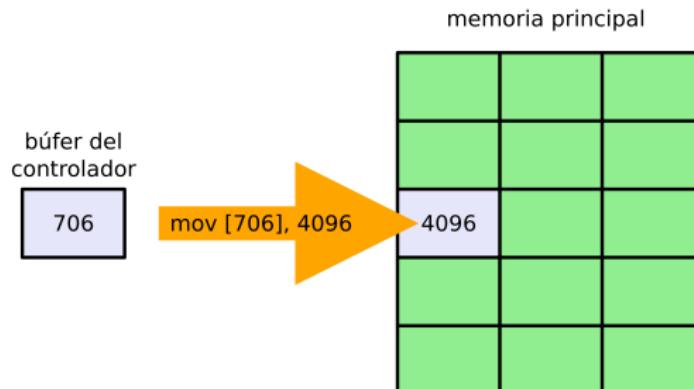
- El procesador envía una petición de DMA al módulo de DMA.
- El módulo de DMA realiza la transferencia solicitada.
- Cuando acaba el módulo de DMA avisa al procesador mediante el envío de una interrupción.

- El procesador sólo trabaja al principio y al final del proceso y es libre de realizar otra tarea durante la transferencia.
- El DMA puede poner una carga elevada en el bus → problema del robo de ciclo.



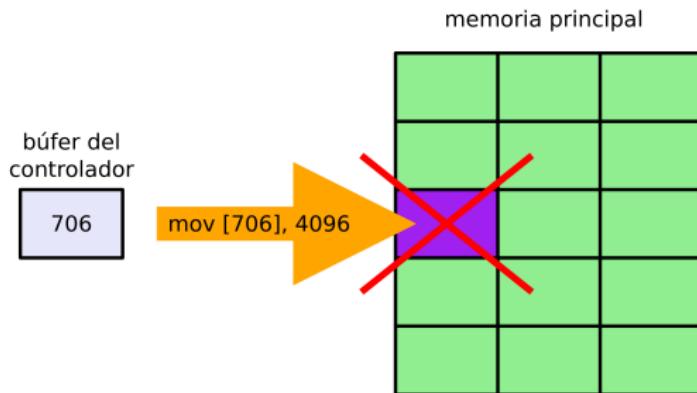
Direccionamiento físico de la memoria principal

- El direccionamiento virtual es algo que sólo existe en el **interior del procesador**.
- Para acceder a memoria los controladores de dispositivos y DMA sólo emplean **direcciones físicas** → ¿dónde ejecutar cada uno?
- Ejemplo: un controlador necesita transferir un bloque a memoria principal.



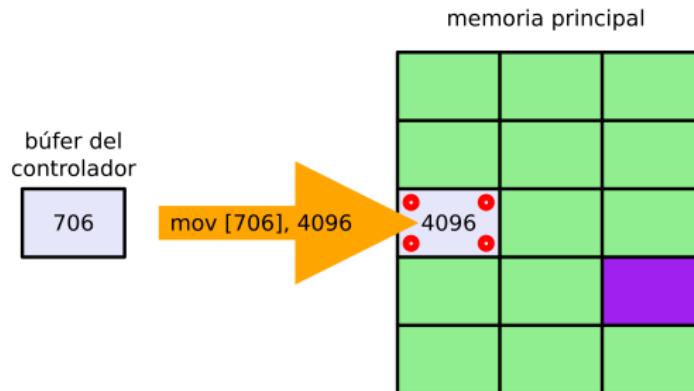
Direccionamiento físico de la memoria principal

- ◎ Un retraso en la E/S podría provocar que el marco 4096 fuese asignado a otro proceso a través de los mecanismos de segmentación o paginación.
- ◎ ¿Cómo resolver este problema?



Direccionamiento físico de la memoria principal

- ◎ Solución: fijar (**pinning**) el marco de página durante el tiempo que dure el proceso de transferencia.
- ◎ Fijar (**pinning**) y liberar (**unpinning**) es un *mecanismo*...
¿Qué *políticas* podrían establecerse?
- ◎ Usos: soporte de DMA y procesos de tiempo real.
- ◎ Problema: el abuso para acaparar recursos.



Temporizador

- ◎ Los niveles de privilegio son un mecanismo de protección útil para el SO.
- ◎ Los eventos pueden transferir el control a SO.
- ◎ ¿Que ocurre si un proceso no genera eventos? \implies nunca cede el control del procesador.
- ◎ Soluciones:
 - más habitual: generar una interrupción de forma periódica mediante el temporizador, ej: cada 10ms.
 - más conveniente: reprogramar el temporizador tras cada evento para que sólo genere interrupciones cuando sea necesario.

Glosario

AS	Address Space.	RISC	Reduced Instruction Set Computer.
ccNUMA	Cache Coherent NUMA.	SASOS	Single Address Space Operating System.
CISC	Complex Instruction Set Computer.		
DMA	Direct Memory Access.	SMP	Simetric MultiProcessing.
EPIC	Explicit Parallel Instrucction Computing.	SMT	Simultaneous MultiThreading.
MMU	Memory Management Unit.	UMA	Unified Memory Architecture.
NORMA	NO Remote Memory Access.	VLIW	Very Long Instruction Word.
NUMA	Non-Uniform Memory Access.		