Unidad 5 - Componentes del computador

· Memoria

● Componente complejo (Sistema de memoria)
Un sistema de memoria es el conjunto de componentes de hardware y software encargados de almacenar y gestionar datos en un sistema informático. Su propósito es proporcionar acceso rápido y eficiente a la información necesaria para la ejecución de programas.

● Formado por elementos con distintas cualidades:
○ Tecnología
○ Organización
○ Performance
○ Costo

● Jerarquía de subsistemas de memoria
La jerarquía de subsistemas de memoria es un modelo organizativo que clasifica los distintos tipos de memoria en un sistema informático según su velocidad, capacidad, costo y proximidad al procesador. El objetivo de esta jerarquía es optimizar el rendimiento y la eficiencia en el acceso a los datos.

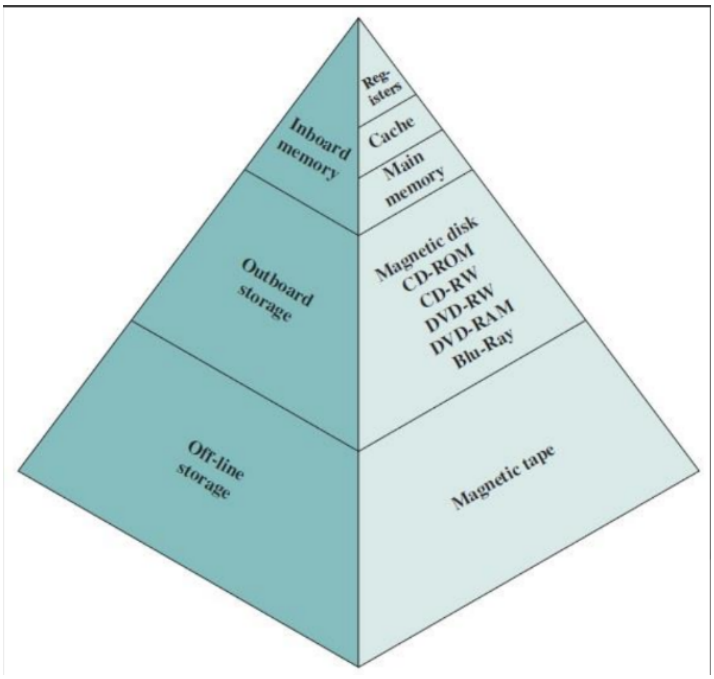
○ Internos al sistema (accedidos directamente por el procesador)

○ Externos al sistema (accedidos por el procesador a través de un módulo de E/S)

○ Tres características a tener en cuenta
* Capacidad
* Tiempo de acceso
* Costo

○ Subsistemas de memoria con relación de compromiso entre estas características ⇒ No se usa un solo componente de memoria:
Esto significa que mejorar una de estas caracteristicas generalmente implica sacrificar otras.

Se organiza en varios niveles, donde las memorias más rápidas y costosas están cerca del procesador, y las más lentas y baratas tienen mayor capacidad pero están más lejos.



○ A medida que se baja de la pirámide:

* Costo por bit decreciente
* Capacidad creciente
* Tiempo de acceso creciente
* Frecuencia de acceso de la memoria por parte de procesador decreciente: Se refiere a la necesidad del CPU de acceder cada cierto tiempo a los datos contenidos en la memoria correspondiente

● Locación
○ Interna
- Registros
- Memoria interna para unidad de control
- Memoria Cache
○ Externa
- Dispositivos de almacenamiento periféricos (discos, cintas, etc.)

● Capacidad:
Es la cantidad maxima de datos que una memoria puede almacenar. Se mide en:
○ Bytes / Palabras (memoria interna)
○ Bytes (memoria externa)

● Unidad de transferencia
Cantidad minima de datos que pueden leerse o escribirse en una sola operacion. Esta cantidad varía dependiendo del tipo de memoria y de su configuración.
○ Número de líneas eléctricas del módulo de memoria, típicamente el tamaño de palabra o 64, 128 o 256 bits (memoria interna)
○ Bloques (memoria externa)

● Métodos de acceso de unidades de datos
○ Acceso secuencial
* Unidades de datos: registros (records)
* Acceso lineal en secuencia
* Se deben pasar y descartar todos los registros intermedios antes de acceder al registro deseado
* Tiempo de acceso variable
* Ejemplo: Cintas magnéticas

○ Acceso directo
* Dirección única para bloques o registros basada en su posición física
* Tiempo de acceso variable
* Ej. discos magnéticos

○ Acceso aleatorio
* Cada posición direccionable de memoria tiene un mecanismo de direccionamiento cableado físicamente:
Cuando la CPU necesita acceder a un dato, envía la dirección a través del bus de direcciones, activando las líneas electricas específicas que llevan a esa celda. Esto permite un acceso aleatorio, es decir, la CPU puede ir directamente a cualquier posición sin necesidad de recorrer toda la memoria
* Tiempo de acceso constante, independiente de la secuencia de accesos anteriores
- Ej. memoria principal y algunas memorias cache

○ Acceso asociativo
* Tipo de acceso aleatorio por comparación de patrón de bits:
Cuando se busca un dato, se compara directamente con los valores almacenados en todas las celdas de la memoria.
* La palabra se busca por una porción de su contenido en vez de por su dirección:
A la memoria se le da una parte del contenido del dato. La memoria revisa todas sus celdas en paralelo y devuelve la celda que contiene el dato solicitado.
* Cada posición de memoria tiene un mecanismo de direccionamiento propio:
Cada celda de memoria tiene su propio mecanismo para identificar si contiene la información solicitada.
* Tiempo de acceso constante, independiente de la secuencia de accesos anteriores o su ubicación:
Debido a que la memoria asociativa busca en paralelo, el tiempo de acceso es constante, sin importar cuántos datos haya almacenados o en qué posición se encuentren
- Ej. memorias cache

● Parámetros de performance
○ Tiempo de acceso (latencia)
* Memorias de acceso aleatorio: tiempo necesario para hacer una operación de lectura o escritura
* Memorias sin acceso aleatorio: tiempo necesario para posicionar el mecanismo de lectura/escritura en la posición deseada
○ Tiempo de ciclo de memoria:
Tiempo total que tarda en completarse una operación de lectura o escritura
* Memorias de acceso aleatorio: tiempo de acceso más el tiempo adicional necesario para que una nueva operación pueda comenzar

○ Tasa de transferencia
* Tasa con la cual los datos son transferidos dentro o fuera de la unidad de memoria
* Memorias de acceso aleatorio: 1/Tiempo de ciclo de memoria
* Memorias sin acceso aleatorio:
Tn = TA + n/R ; donde:
Tn = Tiempo promedio para leer o escribir n bits
TA = Tiempo promedio de acceso
n = Número de bits
R = Tasa de transferencia, en bits por segundo (bps)

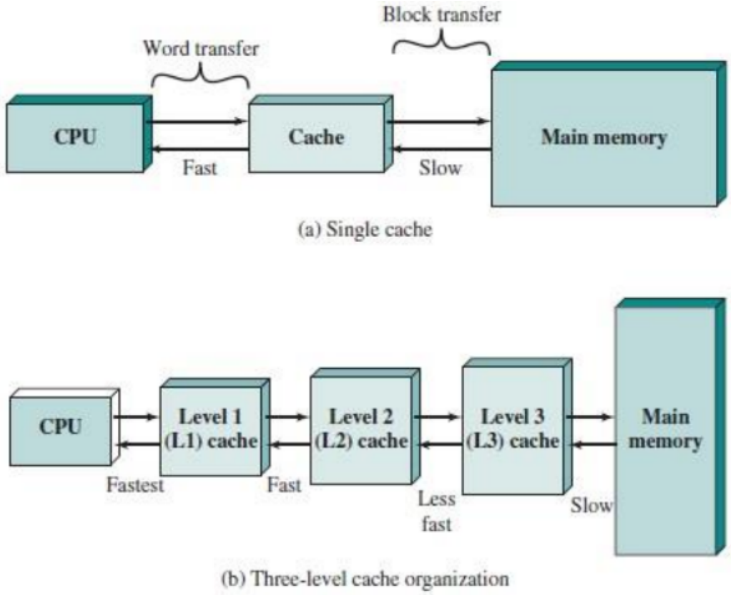
- Tipos físicos
  - Memorias semiconductoras (memoria principal y cache)
  - Memorias de superficie magnética (discos y cintas)
  - Memorias ópticas (medios ópticos)

- Características físicas
  - Memorias volátiles: se pierde su contenido ante la falta de energía eléctrica (Ej. algunas memorias semiconductoras)
  - Memorias no volátiles: no se necesita de energía eléctrica para mantener su contenido (Ej. memorias de superficie magnéticas y algunas memorias semiconductoras)
  - Memorias de solo lectura: (ROM – Read Only Memory) no se puede borrar su contenido (Ej. algunas memorias semiconductoras)

- Principio de localidad de referencia:

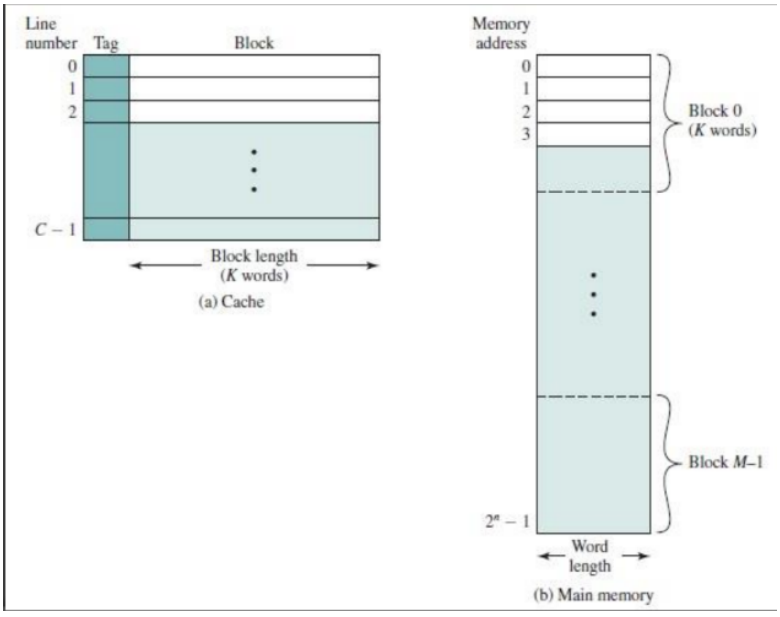
Durante la ejecución de un programa, las referencias a memoria que hace el procesador tanto para instrucciones como datos tienden a estar agrupadas. (Ej. loops, subrutinas, tablas, vectores)

- Memoria Cache
  - Memoria semiconductora más rápida (y costosa) que la principal
  - Se ubica entre el procesador y la memoria principal
  - Permite mejorar la performance general de acceso a memoria principal
  - Contiene una copia de porciones de memoria principal



  - Cómo funciona:

1. CPU trata de leer una palabra de la memoria principal
2. Se chequea primero si existe en la memoria cache.
3. Si es así se la entrega al CPU
4. Sino se lee un bloque de memoria principal (número fijo de palabras), se incorpora a la cache y la palabra buscada se entrega al CPU
5. Por el principio de localidad de referencia es probable que próximas palabras buscadas estén dentro del bloque de memoria subido a la cache



  - Estructura sistema cache/memoria principal
    * Memoria principal
    - 2^n palabras direccionables (dirección única de n-bits para cada una)
    - Bloques fijos de K palabras cada uno (M bloques)
  - Cache
    * m bloques llamados líneas
    * Cada línea contiene:
    - K palabras
    - Tag (conjunto de bits para indicar qué bloque está almacenado, usualmente una porción de la dirección de memoria principal)
    - Bits de control (Ej. bit para indicar si la línea se modificó desde la última vez que se cargó en la cache)

· Administración de Memoria

La memoria es un recurso limitado que debe administrarse eficientemente para optimizar el rendimiento del sistema.
El Sistema Operativo se encarga de gestionar la memoria y los procesos, pero esta administración depende del soporte del hardware.

● Sistema Operativo

Software que administra los recursos del computador, provee servicios y controla la ejecución de otros programas

○ Algunos servicios que provee

* Schedule de procesos: Administra qué procesos se ejecutan y en qué orden, asignando tiempo de CPU de manera eficiente.
* Administración de memoria: Maneja la memoria disponible y asigna espacios a los procesos, evitando que interfieran entre sí.

○ Monitor:

* Es la parte del sistema operativo que siempre está en ejecución

● Uniprogramación

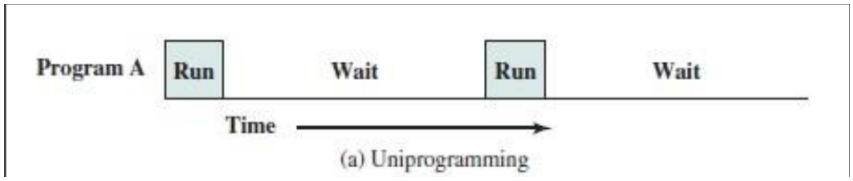
Es un esquema en el que solo se ejecuta un único programa en la memoria principal en un momento dado. Este enfoque es sencillo, pero tiene algunas limitaciones en términos de eficiencia en la utilización de la memoria y el rendimiento.

○ Un solo proceso de usuario en ejecución a la vez

○ La memoria de usuario está completamente disponible para ese único proceso:

Esto significa que cuando un programa se está ejecutando, no hay otros programas en la memoria esperando ser ejecutados.

○ Uso del procesador a lo largo del tiempo:



* Run: El proceso está en ejecución y la CPU está ocupada procesando sus instrucciones.
* Wait: El proceso está esperando un evento o recurso (como una entrada/salida, una respuesta del usuario, etc.) y no está ejecutando instrucciones activas. En este estado, no hay otro proceso que ocupe la CPU en sistemas de uniprogramación.

● Administración de memoria simple

○ Sistema con uniprogramación

○ Se divide la memoria en dos partes

* Monitor del S.O.
* Programa en ejecución en ese momento

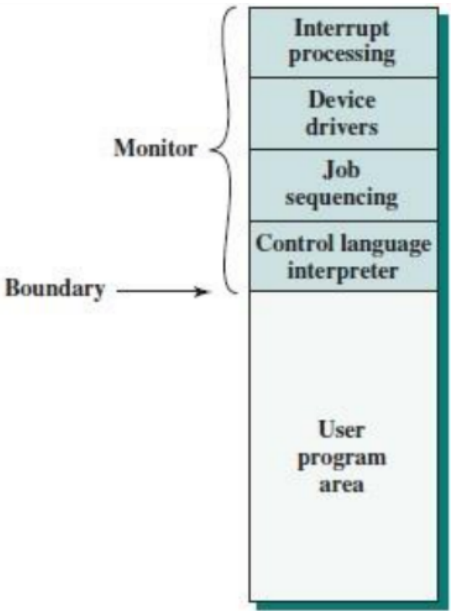
○ Ventajas:

* Simplicidad

○ Desventajas:

* Desperdicio de memoria
* Desaprovechamiento de los recursos del computador

○ Ej. MS-DOS, iPhone OS v1-3, IBM OS/PCP (Primary Control Program)

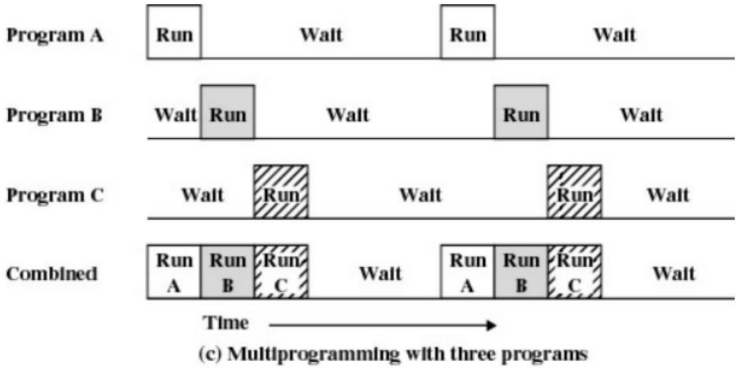


● Multiprogramación

○ Varios procesos de usuario en ejecución a la vez

○ Se divide la memoria de usuario entre los procesos en ejecución

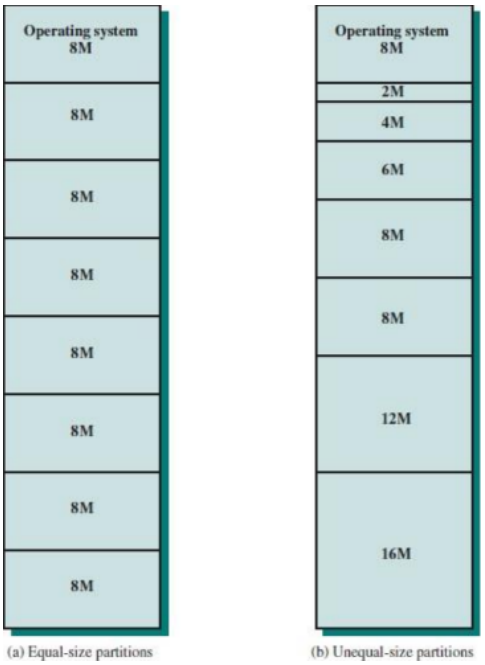
○ Se comparte el tiempo de procesador entre los procesos en ejecución (timeslice)



Aclaracion: El timeslice (o "quantum de tiempo") es el término que se utiliza para describir el intervalo de tiempo durante el cual un proceso tiene acceso exclusivo a la CPU

○ Condiciones de finalización de los procesos:
* Termina el trabajo
* Se detecta un error y se cancela
* Requiere una operación de E/S (suspensión)
* Termina el timeslice (suspensión)

● Administración de memoria por asignación particionada
○ Sistema con multiprogramación
○ La memoria de usuario se divide en particiones de tamaño fijo:
* Iguales
* Distintas
○ Ventajas:
* Permite compartir la memoria entre varios procesos
○ Desventajas:
* Desperdicio de memoria
- Fragmentación interna (dentro de una partición):
Ocurre cuando se asigna más memoria de la necesaria a un proceso. Esto sucede típicamente cuando la memoria se divide en bloques de tamaño fijo o predeterminado (en el caso de particiones fijas)
- Fragmentación externa (particiones no usadas):
Ocurre cuando hay suficiente memoria total libre en el sistema, pero está distribuida en pequeños bloques dispersos que no son lo suficientemente grandes como para acomodar un nuevo proceso
○ Ej. IBM OS/MFT (Multiprogramming with a Fixed number of Tasks)



(a) Equal-size partitions (b) Unequal-size partitions

● Administración de memoria por asignación particionada reasignable
○ Sistema con multiprogramación

○ Swapping:
Es una técnica que implica mover procesos entre la memoria principal y el almacenamiento secundario (generalmente en disco, como un disco duro o SSD) para optimizar el uso de la memoria y permitir la ejecución de más procesos de los que físicamente cabrían en la memoria principal al mismo tiempo.

○ La memoria de usuario se divide en particiones de tamaño variable:
Las particiones que se crean para los procesos pueden tener tamaños diferentes según las necesidades de cada proceso. Cada vez que un proceso requiere memoria, se asigna una partición del tamaño exacto que necesita (o lo más cercano posible, en algunos casos)

○ Compactación para eliminar la fragmentación:
Es una tecnica que reorganiza los procesos en la memoria para crear bloques de memoria contiguos, lo que facilita la asignación de memoria para nuevos procesos.

○ Se usa un recurso de hardware (registro de reasignación) para la realocación:
Este registro guarda la información sobre dónde se movió el proceso dentro de la memoria física. El procesador utiliza este registro para ajustar las direcciones de memoria utilizadas por el proceso, de manera que los punteros dentro del proceso se actualicen automáticamente y apunten a las nuevas ubicaciones de memoria, sin que el proceso pierda acceso a sus datos.

○ Realocación dinámica en tiempo de ejecución

○ Ventajas:
* Permite compartir la memoria entre varios procesos
* Elimina el desperdicio por fragmentación interna. Con la compactación se elimina además la fragmentación externa

○ Desventajas:
* La tarea de compactación es costosa
○ Ej. IBM OS/MVT (Multiprogramming with a Variable number of Tasks)

○ Ejemplo:

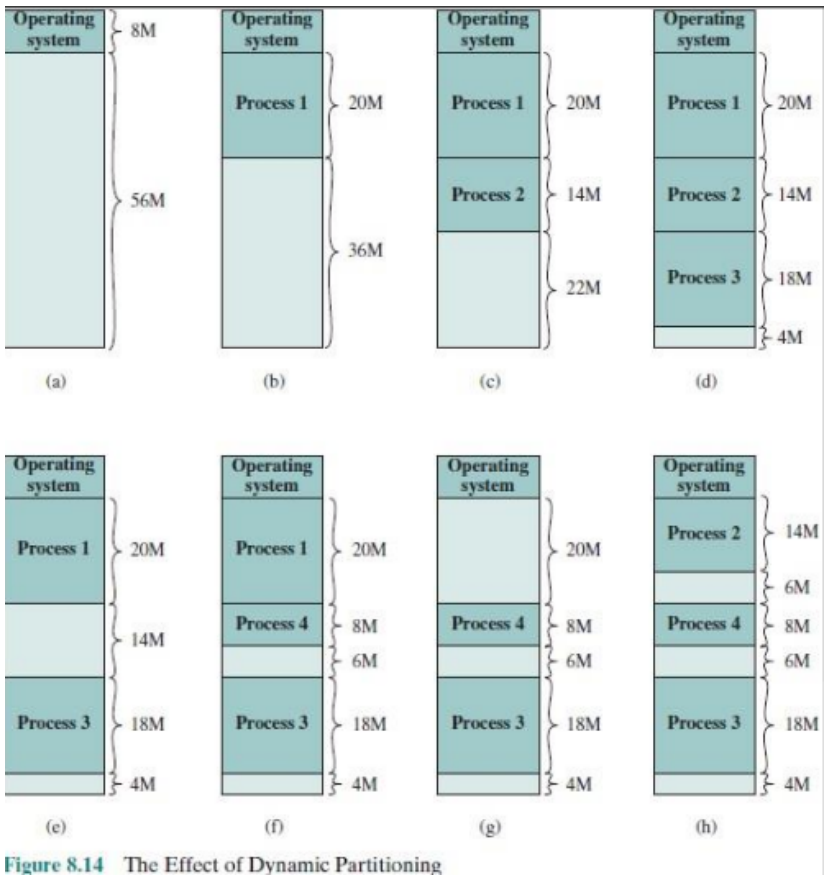


Figure 8.14 The Effect of Dynamic Partitioning

● Administración de memoria paginada simple
○ Sistema con multiprogramación

○ Se divide el address space del proceso en partes iguales (páginas) (ej. IA-32 4KB c/u):
El espacio de direcciones de un proceso es el rango de memoria que puede usar durante su ejecución. Dividirlo en páginas significa fragmentarlo en bloques de tamaño fijo (ej. 4 KB u 8 KB)

○ Se divide la memoria principal en partes iguales (frames):
Es un bloque de memoria de tamaño fijo en la memoria física (RAM), que corresponde a una página del address space de un proceso.

○ Hay una tabla de páginas por proceso:
Cada proceso tiene su propia tabla de páginas, que el sistema operativo usa para mapear páginas a frames en la memoria física. Cuando un proceso accede a una página, la tabla traduce su número de página a su ubicación en la RAM.

○ Hay una lista de frames disponibles:
Mantiene un registro de cuáles frames de la memoria física están libres y cuáles están ocupados.

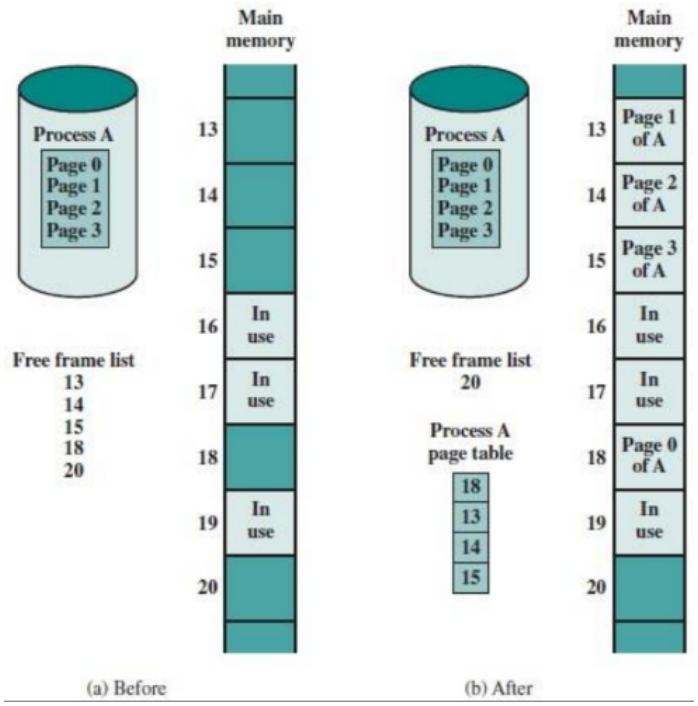
○ Se cargan a memoria las páginas del proceso en los frames disponibles (no es necesario que sean contiguos):
Cuando un proceso necesita cargar una página en la memoria, el sistema operativo consulta la lista de frames disponibles para encontrar un frame libre donde colocar la página

○ Las direcciones lógicas se ven como número de página y un offset:
El número de página identifica el frame en la tabla de páginas, y el offset indica la posición exacta dentro de ese frame.

○ Se traducen las direcciones lógicas en físicas (address translation) con soporte del hardware (MMU – Memory Management Unit)

○ La paginación es transparente para el programador:
No es necesario que el programador tenga conocimiento del proceso de paginacion, no requiere saberlo para programar.

○ Ejemplo:
* Asignacion de paginas a frames



* Traduccion de las direcciones logicas a reales.

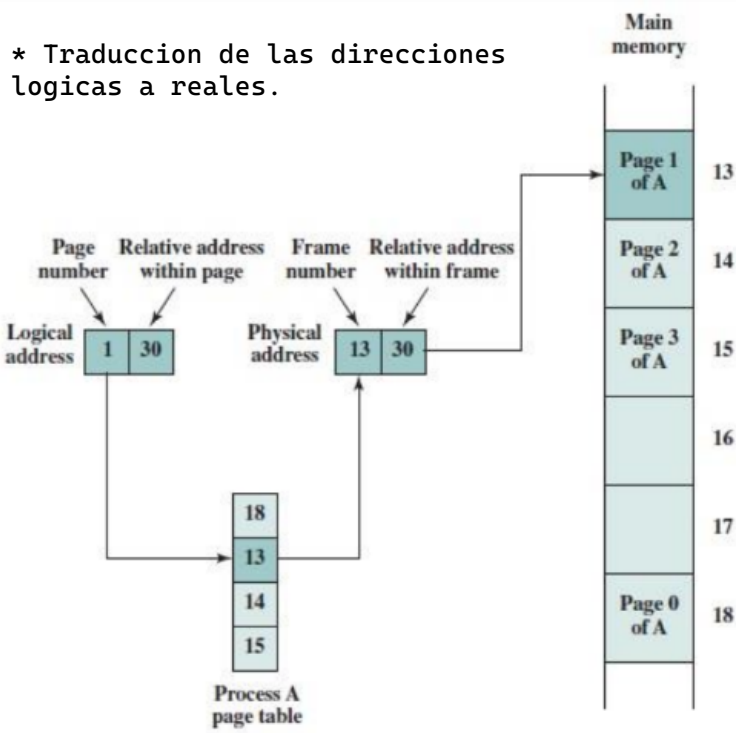


Figure 8.16 Logical and Physical Addresses

○ Ventajas:
* Permite compartir la memoria entre varios procesos
* Permite el uso no contiguo de la memoria
* Minimiza la fragmentación interna (solo existe dentro de la última página de cada proceso)
* Elimina la fragmentación externa

○ Desventajas:
* Se requiere subir todas las páginas del proceso a memoria
* Se requieren estructuras de datos adicionales para mantener información de páginas y frames

● Administración de memoria paginada por demanda
La memoria paginada por demanda es un mecanismo que permite ejecutar programas más grandes que la memoria física disponible al cargar solo las partes necesarias en la RAM. Esto mejora el uso de la memoria y permite que varios procesos compartan recursos sin cargar todo el programa en la RAM desde el inicio.

○ ¿Cómo funciona?
1 Cargar solo lo necesario
No se carga todo el programa en la RAM, solo las páginas necesarias.

2 Page Fault (Fallo de Página)
Si un proceso intenta acceder a una página que no está en RAM, ocurre un page fault.
Esto genera una interrupción en el sistema, que pausa el proceso hasta resolverlo.

3 Manejo del Page Fault
El sistema operativo (SO) revisa dónde está la página en memoria secundaria (ej. disco duro).
Luego, la carga en RAM para que el proceso pueda continuar.

4 Page Swapping (Intercambio de páginas)
Si la RAM está llena, el SO necesita liberar espacio.
Se reemplazan páginas antiguas por las nuevas necesarias.
Existen algoritmos para decidir qué páginas eliminar, como:
-FIFO (First In First Out): La primera página en entrar es la primera en salir.
-LRU (Least Recently Used): Se elimina la página menos usada recientemente.

5 Thrashing (Exceso de intercambio de páginas)
Si un sistema tiene poca RAM y los procesos requieren muchas páginas, se pasa más tiempo cargando y sacando páginas que ejecutando instrucciones.
Esto ralentiza el sistema y lo hace ineficiente.

● Administración de memoria por segmentación
La segmentación es una técnica de gestión de memoria utilizada en sistemas operativos con multiprogramación. En lugar de tratar la memoria como un espacio de direcciones contiguo y lineal (como en la paginación), se divide en segmentos lógicos que pueden crecer o reducirse dinámicamente. Si se necesita más memoria para datos, el segmento de datos crece.
Si se liberan datos, el segmento de datos se reduce.

○ Características Claves:
1 Visible al programador
A diferencia de la paginación, donde la memoria se divide en bloques fijos, la segmentación es más intuitiva porque se organiza en segmentos lógicos, como código, datos y pila.

2 Estructura de la memoria
La memoria del programa se divide en segmentos de diferentes tamaños.
Cada segmento tiene su propio espacio de direcciones, en lugar de un único espacio contiguo.

3 Tabla de segmentos
Cada proceso tiene una tabla de segmentos que almacena:
-Número del segmento
-Dirección base del segmento
-Tamaño del segmento
El sistema operativo usa esta tabla para asignar y gestionar memoria.

4 Separación de datos e instrucciones
Permite que los programas tengan segmentos separados para código, datos y pila, evitando que una parte del código sobrescriba otra.

5 Protección y privilegios
Se pueden asignar permisos de lectura, escritura o ejecución a cada segmento.
Si un programa accede a un segmento fuera de su rango permitido, se genera un segmentation fault (fallo de segmentación).

6 Traducción de direcciones
Cada referencia a memoria se compone de:
-Número de segmento
-Offset (desplazamiento) dentro del segmento
La Unidad de Gestión de Memoria (MMU - Memory Management Unit) convierte estas direcciones lógicas en direcciones físicas.

7 Memoria Virtual
La segmentación se puede usar con memoria virtual, donde solo algunos segmentos del proceso están en memoria física y el resto permanece en disco.

○ Ventajas:
1 Simplifica el manejo de estructuras de datos con crecimiento: Facilita la gestión de memoria para datos que pueden cambiar de tamaño.

2 Permite compartir información entre procesos dentro de un segmento: Los procesos pueden acceder a la misma región de memoria de manera eficiente.

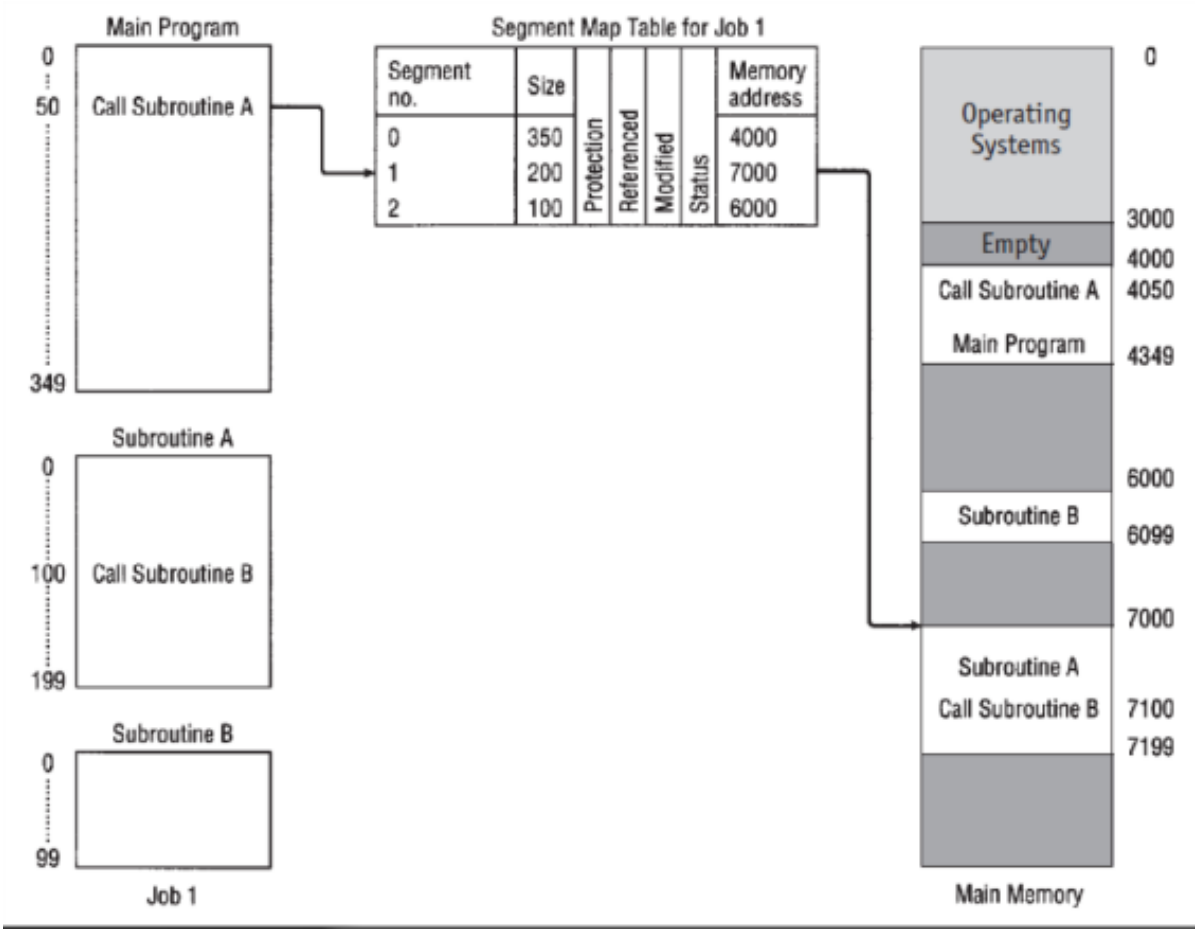
3 Permite aplicar protección/privilegios sobre un segmento fácilmente: Se puede restringir el acceso a diferentes partes de un segmento según sea necesario.

○ Desventajas:
1 Fragmentación externa en la memoria principal: Se puede generar espacio desperdiciado si no hay suficiente memoria contigua para alojar un segmento.

2 Hardware más complejo que memoria paginada para la traducción de direcciones: Requiere un hardware más avanzado para traducir direcciones debido a la gestión de segmentos.

○ Ejemplos de arquitecturas que la aplican:
- Burroughs Corporation B5000-B6500 - IBM AS/400 - Intel x86 (por compatibilidad hacia atrás).

- Ejemplo visual:



- Distintas combinaciones (Ej. Intel Pentium)
  - Sin segmentación y sin paginación
    * Direcciones lógicas iguales a las físicas. No es útil para multiprogramación. Usado en controladores de alta performance

  - Paginación sin segmentación
    * La protección y administración de la memoria se hace a través de las páginas.
    * Ej. Berkeley UNIX

  - Segmentación sin paginación
    * La memoria se ve como una colección de espacios lógicos, con protección a nivel segmentos.
  - Segmentación con paginación
    * Segmentos para controlar el acceso a particiones de memoria.
    * Páginas para administrar la locación dentro de los segmentos
    * Ej. UNIX System V

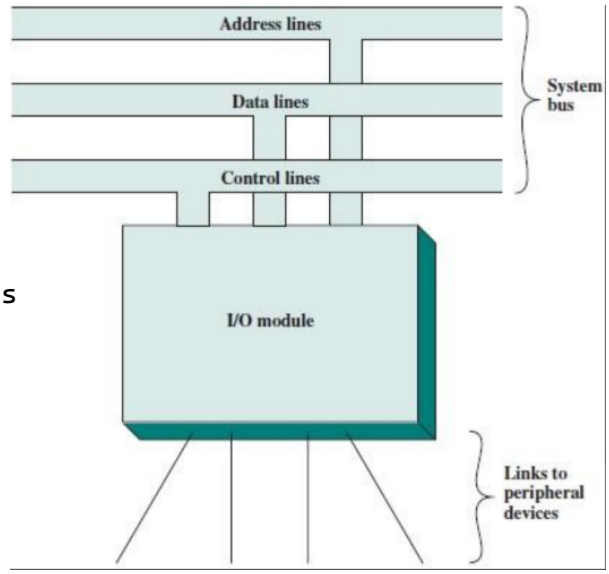
· Módulo de E/S

- ¿Qué hace?
  - Conecta a los periféricos con la CPU y la memoria a través del bus del sistema o switch central y permite la comunicación entre ellos

- ¿Por qué existe?
  - Amplia variedad de periféricos con distintos métodos de operación
  - La tasa de transferencia de los periféricos es generalmente mucho más lenta que la de la memoria y procesador
  - Los periféricos usan distintos formatos de datos y tamaños de palabra

- ¿Para qué sirve?
  - Oculta detalles de timing, formatos y electro mecánica de los dispositivos periféricos

● Intefaz interna / System bus / Bus del sistema
Conecta la CPU y la memoria con el módulo de E/S.
Se compone de tres tipos de líneas:
○ Address lines (Líneas de dirección): Son un conjunto de cables o señales que la CPU usa para indicar qué periférico debe responder a una operación. Son esenciales para que la CPU pueda gestionar múltiples dispositivos correctamente.



○ Data lines (Líneas de datos): Transportan los datos entre la CPU y los dispositivos.

○ Control lines (Líneas de control): Señales que coordinan la comunicación, como lectura/escritura, activación de dispositivos y generación de interrupciones.

● Interface externa (periféricos)
○ Datos: Canal para transferir información entre los periféricos y el sistema.
○ Control: Señales que gestionan las operaciones entre los periféricos y el sistema.
○ Estado: Proporciona información sobre el estado del periférico, como si está listo para operar o si está ocupado.

● Funciones
○ 1. Control & Timing
Coordina el flujo de datos entre la CPU, la memoria y los dispositivos de entrada/salida.
Se asegura de que los datos se envíen y reciban en el momento adecuado.
* Ejemplo:
Cuando la CPU quiere leer datos de un disco duro, el módulo de E/S controla el tiempo de acceso y la transferencia.

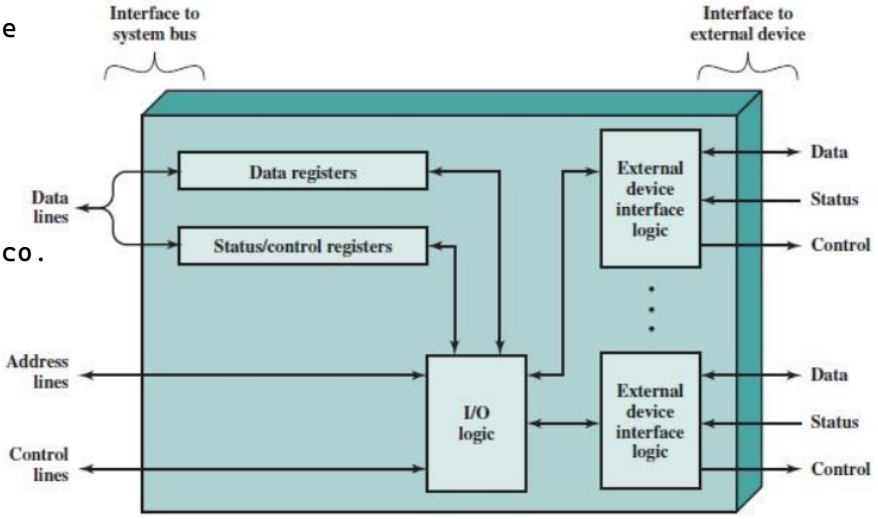
○ 2. Comunicación con el Procesador (Intefaz interna)
Decodificación de comandos → Traduce las órdenes de la CPU para que el dispositivo las entienda.
Datos → Transfiere los datos entre la CPU y los dispositivos.
Información de estado → Indica si el dispositivo está ocupado, libre o ha ocurrido un error.
Reconocimiento de direcciones → Determina qué periférico debe responder a la CPU.
* Ejemplo:
Si la CPU quiere imprimir un documento, envía un comando al módulo de E/S, que lo traduce y lo envía a la impresora.

○ 3. Comunicación con el Dispositivo (Intefaz externa)
Comandos → Envía órdenes al periférico (ejemplo: "lee datos", "imprime", "mueve el cursor").
Información de estado → Recibe datos del dispositivo sobre su estado actual (ocupado, error, listo).
Datos → Gestiona la transferencia de información entre el periférico y la CPU/memoria.
* Ejemplo:
Si un teclado envía una tecla presionada, el módulo de E/S recibe la señal y la pasa a la CPU.

○ 4. Buffering de Datos
Almacena temporalmente los datos antes de enviarlos a la CPU o al periférico.
Ayuda a compensar diferencias de velocidad entre la CPU y los periféricos (que son más lentos).
* Ejemplo:
Cuando descargas un video en streaming, se almacena temporalmente en un buffer para evitar interrupciones si la conexión es lenta.

○ 5. Detección de Errores
Verifica que los datos transferidos no contengan errores.
Puede utilizar códigos de detección de errores, como paridad o CRC (Cyclic Redundancy Check).
* Ejemplo:
Si descargas un archivo y se detecta un error en la transmisión, el módulo de E/S solicita que se reenvíen los datos dañados.

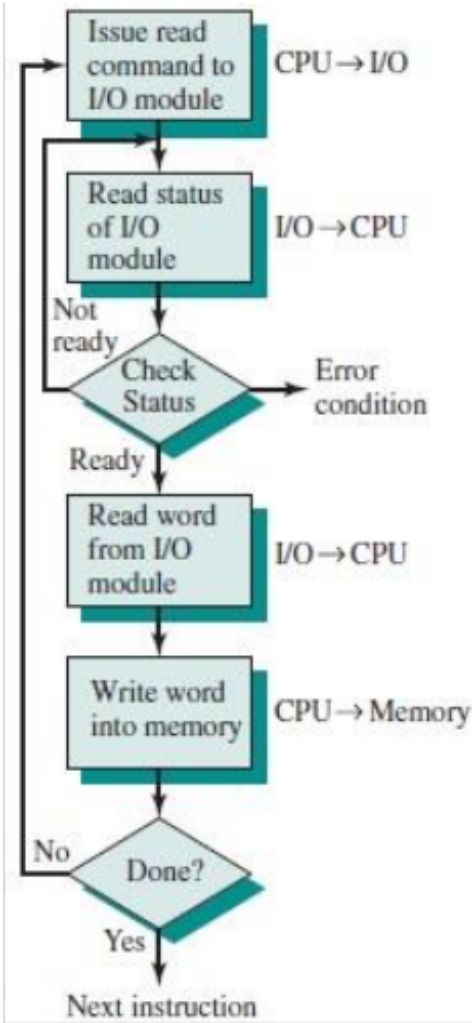
● Estructura del modulo E/S
○ Data Registers: Almacenan los datos que se transfieren entre la CPU y los periféricos.
○ Status Registers: Indican el estado del dispositivo (ocupado, listo, error, etc.).
○ Control Registers: Permiten a la CPU configurar y controlar el módulo de E/S. Puede contener banderas (bits) para activar o desactivar funciones del periférico.
Ejemplo:
-Configurar si una impresora trabaja en modo texto o gráfico.
-Habilitar o deshabilitar interrupciones de un dispositivo.



○ Diferencia entre Control Registers y Control lines:
En resumen, un control register es un lugar donde se almacenan configuraciones que controlan el funcionamiento del módulo de E/S, mientras que una control line es una señal de control en tiempo real que influye en las operaciones del sistema.

● Técnicas para operaciones de E/S
○ E/S Programada
○ E/S manejada por interrupciones
○ Acceso directo a memoria (DMA)

● E/S Programada
El diagrama describe el proceso de Entrada/Salida (E/S) Programada, un método en el que la CPU gestiona directamente la transferencia de datos con un dispositivo de E/S.



○ Issue read command to CPU→I/O:
La CPU envía una orden de lectura al módulo de E/S (por ejemplo, para leer datos de un disco duro o un teclado).

○ I/O module reads status:
* Si el estado es "Ready" → Transfiere los datos del dispositivo a la CPU.
* Si el estado es "Not Ready" → Vuelve a "Read status of I/O" (bucle de espera)
* Si hay error → Maneja la excepción (no se muestra en el diagrama).

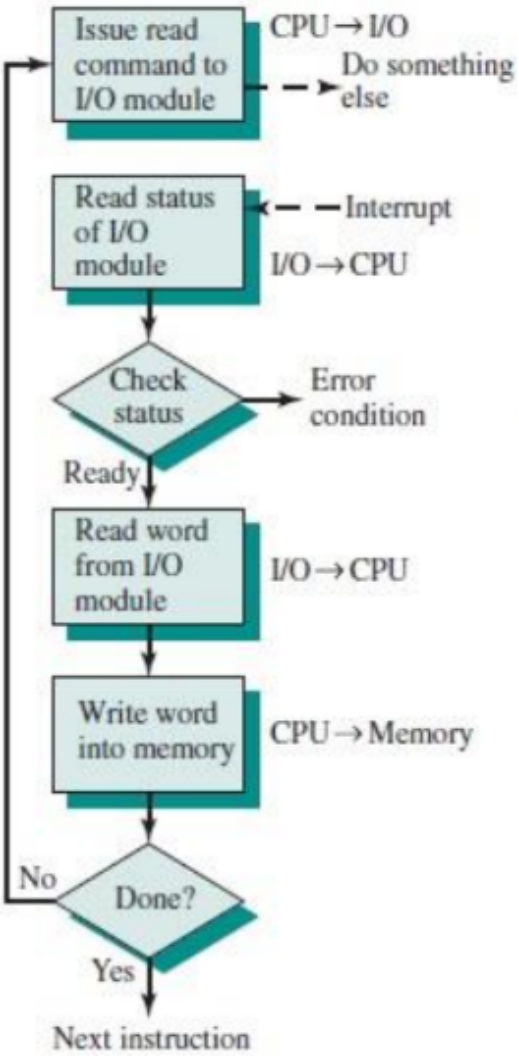
○ Read word from I/O module
Si el dispositivo está listo, se transfiere la palabra de datos desde el módulo de E/S a la CPU. En el diagrama no se especifica como el modulo de E/S lee los datos del dispositivo

○ Write word into memory (CPU→Memory):
La CPU escribe los datos recibidos en la memoria principal.

○ Done?:
Se verifica si se completó la operación (por ejemplo, si se leyó todo el bloque de datos requerido). Si no (No), se repite el proceso. Si sí (Yes), la CPU continúa con la Next instruction.

○ Desventajas:
* Ineficiencia de la CPU (Espera Activa o Polling):
La CPU queda ocupada verificando constantemente el estado del dispositivo en un bucle (polling), sin poder realizar otras tareas.
- Ejemplo: Si un disco duro tarda 10 ms en preparar datos, la CPU desperdicia miles de ciclos de reloj en esperar, en lugar de ejecutar procesos útiles.

- E/S manejada por interrupciones



- Issue read command to CPU→I/O:
La CPU envía un comando de lectura al módulo de E/S (ej: leer datos de un disco).

- Do something else:
La CPU no espera a que el dispositivo esté listo. En cambio, ejecuta otras instrucciones (procesa tareas en paralelo).

- I/O module reads status:
* Si el estado es "Ready" → envía una señal de interrupción a la CPU.
* Si hay error → Maneja la excepción

- Read word from I/O module:
Si el dispositivo está listo y no hay errores, la CPU lee los datos desde el módulo de E/S

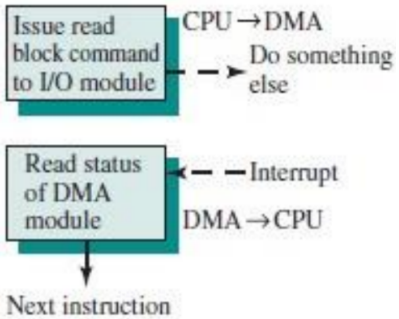
- Write word into memory:
Los datos leídos se escriben en la memoria principal.

- Done?:
Si la operación está completa (Yes), la CPU retoma la ejecución de la siguiente instrucción. Si no (No), repite el proceso.

- ¿Cuando el CPU procesa una interrupcion?
La CPU verifica si hay interrupciones pendientes después de ejecutar cada instrucción de máquina, específicamente al final del ciclo de instrucción (después de las fases de busqueda y ejecucion).

- Desventajas
Es ineficiente que la CPU se encargue directamente de leer los datos del módulo de E/S y escribirlos en memoria, especialmente cuando se manejan grandes volúmenes de datos.
- ¿Por qué es ineficiente?
* Consumo de ciclos de CPU:
Cada operación de lectura/escritura requiere que la CPU ejecute instrucciones para mover los datos (por ejemplo, Read word from I/O module y Write word into memory en el diagrama).
Esto roba ciclos de procesamiento que podrían usarse para tareas más críticas, como ejecutar programas o algoritmos.

- Acceso directo a memoria (DMA)
- Issue read block command (CPU→DMA):
La CPU envía un comando al controlador de DMA para iniciar una transferencia de datos en bloque (por ejemplo, leer un archivo completo de un disco).
* En este comando, la CPU especifica:
- Dirección de memoria donde se almacenarán los datos.
- Dispositivo de E/S involucrado (ej: disco duro).
- Tamaño de los datos a transferir (ej: 1 MB).



- Do something else:
La CPU libera sus recursos y continúa ejecutando otras tareas (ej: procesar programas en segundo plano), ya que el DMA se encarga de toda la transferencia.

- Read status of DMA module:
* El controlador de DMA gestiona la comunicación con el módulo de E/S:
- Verifica si el dispositivo está listo.
- Coordina la transferencia directa de datos entre el dispositivo y la memoria.

- Interrupt (DMA→CPU):
Cuando la transferencia se completa o ocurre un error, el DMA envía una única interrupción a la CPU para notificar el resultado.

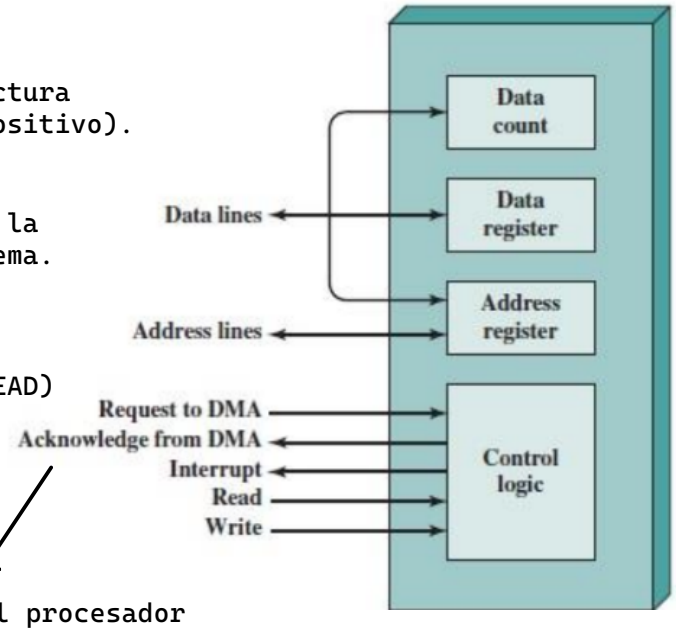
- Next instruction:
La CPU, al recibir la interrupción, puede manejar el evento (ej: confirmar que los datos están disponibles) y retomar su flujo normal de ejecución.

○ Informacion enviada por el CPU al DMA:
1 Operación (READ o WRITE) – Línea de Control
La CPU indica al DMA si la transferencia será de lectura (dispositivo → memoria) o escritura (memoria → dispositivo).

2 Dirección del dispositivo – Línea de Dirección
La CPU señala qué dispositivo de E/S participará en la transferencia, usando su dirección única en el sistema.

3 Dirección inicial de memoria – Address Register
La CPU proporciona la dirección de memoria donde iniciará la transferencia, ya sea para almacenar (READ) o extraer datos (WRITE).

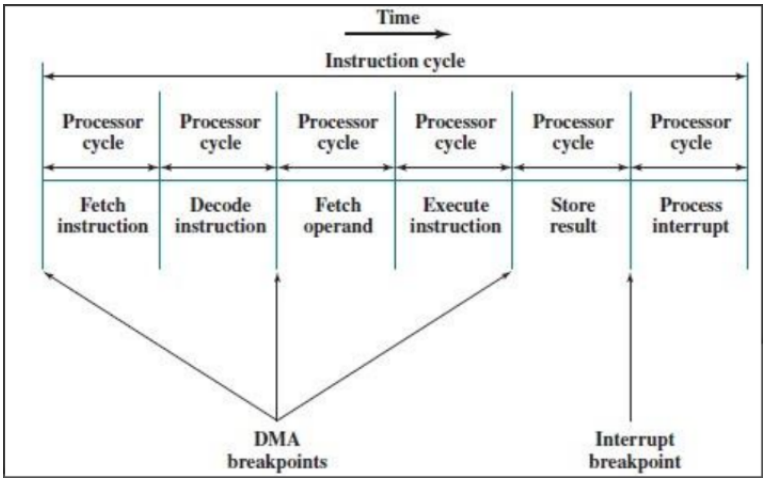
4 Cantidad de palabras – Data Count Register
Se especifica el número de datos a transferir, medido en palabras.



El dma le dice al procesador que esta preparado para tomar la solicitud

○ Flujo de una transferencia DMA
* Configuración: La CPU envía al DMA los parámetros de la operación.
* Inicio: El DMA toma el control del bus y realiza la transferencia sin intervención de la CPU.
* Finalización: El DMA envía una interrupción al procesador cuando termina la transferencia.

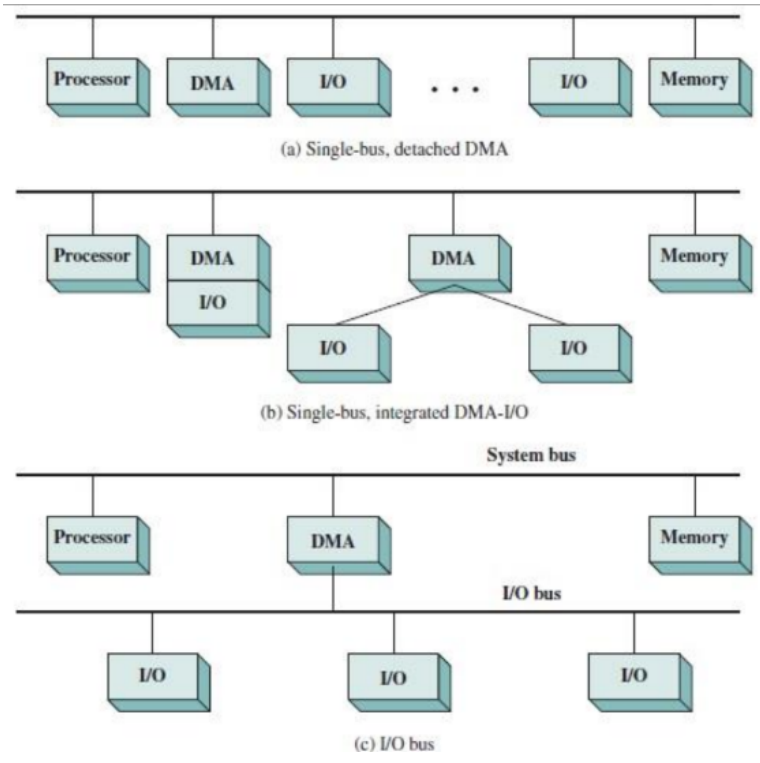
○ DMA – "Robo de ciclos":
Para que se implemente el DMA, el procesador tiene que compartir su bus de sistema con el módulo DMA. Por lo tanto, el módulo DMA debe utilizar el bus solo cuando el procesador no lo necesita, o debe obligar al procesador a suspender la operación temporalmente. La última técnica es la más común de utilizar y se conoce como robo de ciclos. Como se puede ver en el grafico, el unico momento donde el DMA puede suspender, es antes de que se realice cualquier microinstruccion que acceda a la memoria (como un fetch, o un store)



○ DMA – Topologías de configuración
1 Bus único, DMA independiente
Todos los módulos comparten el mismo bus.
El DMA actúa como un procesador sustituto, usando E/S programadas para transferencias.
Económico, pero ineficiente: Cada palabra transferida usa dos ciclos de bus.

2 DMA integrado de bus único
Hay una ruta directa entre el DMA y los módulos de E/S, sin usar el bus del sistema.
Menos ciclos de bus necesarios.
El bus del sistema solo se usa para intercambiar datos con la memoria.

3 Bus de E/S
Extiende la configuración anterior conectando los módulos de E/S al DMA mediante un bus propio.
Reduce las interfaces de E/S necesarias y facilita la expansión.
El bus del sistema se usa solo para comunicación con la memoria.



● Canales y procesadores de E/S
○ Canales de E/S
1️⃣ Tienen la habilidad de ejecutar instrucciones de E/S
Son dispositivos especializados que gestionan las operaciones de entrada/salida sin intervención constante de la CPU. Pueden ejecutar una serie de instrucciones de E/S de manera autónoma.

2️⃣ La CPU principal no ejecuta instrucciones de E/S
La CPU solo inicia la operación, pero el canal se encarga de realizar la transferencia de datos entre los dispositivos y la memoria. Esto libera a la CPU para que pueda realizar otras tareas mientras el canal maneja la E/S.

3️⃣ Las instrucciones de E/S se almacenan en memoria principal
En lugar de que la CPU controle directamente los dispositivos, las instrucciones de E/S se guardan en la memoria RAM y el canal las ejecuta cuando recibe la orden.

4️⃣ La CPU le indica al canal de E/S que inicie un programa de canal
La CPU no gestiona cada operación de E/S manualmente, sino que solo envía una orden inicial al canal, especificando:
- Dispositivo: Qué periférico se usará (impresora, disco duro, etc.).
- Área de memoria para storage: Dónde almacenar los datos transferidos en la memoria principal.
- Prioridad: Indica la urgencia de la operación respecto a otras tareas.
- Acciones ante errores: Define cómo manejar fallos (reintento, notificación a la CPU, etc.).

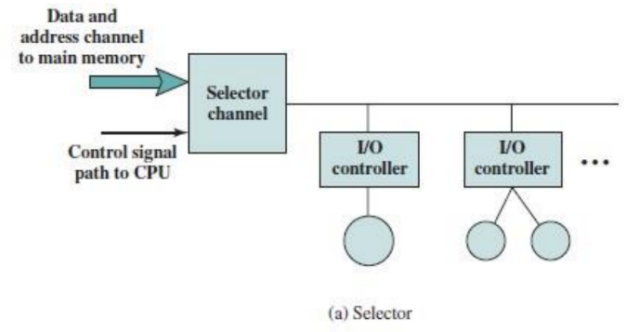
○ Procesadores de E/S
Agregan a los canales memoria propia en vez de usar la memoria principal
* Características:
- Tienen memoria local para almacenar programas y datos.
- Ejecutan programas de E/S almacenados en memoria.
- Pueden manejar operaciones complejas: cifrado, compresión, etc.
- Ejemplo: En mainframes, un procesador de E/S gestiona transferencias de datos entre discos, redes y la CPU principal.

○ Tipos de Canales de E/S

* Canal Selector:
- Diseñado para dispositivos de alta velocidad (ej: discos duros).

- Dedica toda su capacidad a un solo dispositivo durante una transferencia.

- Ejemplo: Transferir un archivo grande desde un disco SSD.

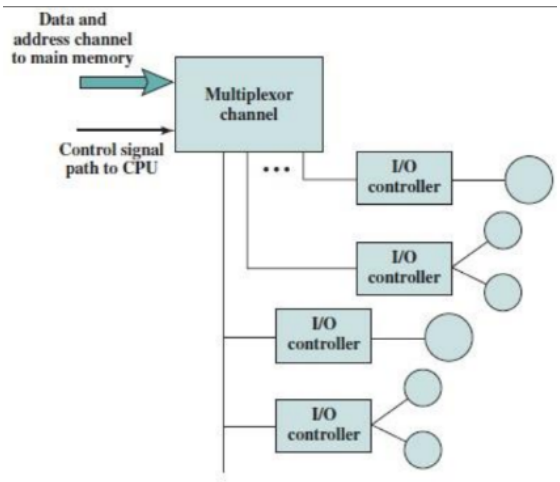


(a) Selector

* Canal Multiplexor:
- Maneja múltiples dispositivos al mismo tiempo.

- Byte Multiplexor: Para dispositivos lentos (ej: teclados), intercala bytes de diferentes dispositivos.

- Bloque Multiplexor: Para dispositivos rápidos (ej: tarjetas de red), intercala bloques de datos.
Ejemplo: Un servidor gestionando solicitudes de varios clientes simultáneamente.



(b) Multiplexor

· Interrupciones

Mecanismos por los cuales otros módulos (E/S, memoria, etc.) interrumpen el normal procesamiento del CPU para mejorar la eficiencia de procesamiento de un computador.

- Clases de interrupciones
  - Hardware (asincrónicas):
    * E/S: Se producen cuando un dispositivo (teclado, disco, red, etc.) necesita la atención del procesador, por ejemplo, cuando se presiona una tecla.
    * Reloj (timer): Provocadas por un temporizador para administrar el tiempo de ejecución de los procesos en un sistema de multiprogramacion.
    * Fallas de hardware: Ocurren cuando hay problemas físicos en el sistema, como un fallo en la memoria o el procesador.

  - Software:

Generadas por el propio procesador en respuesta a ciertas condiciones dentro del software en ejecución.

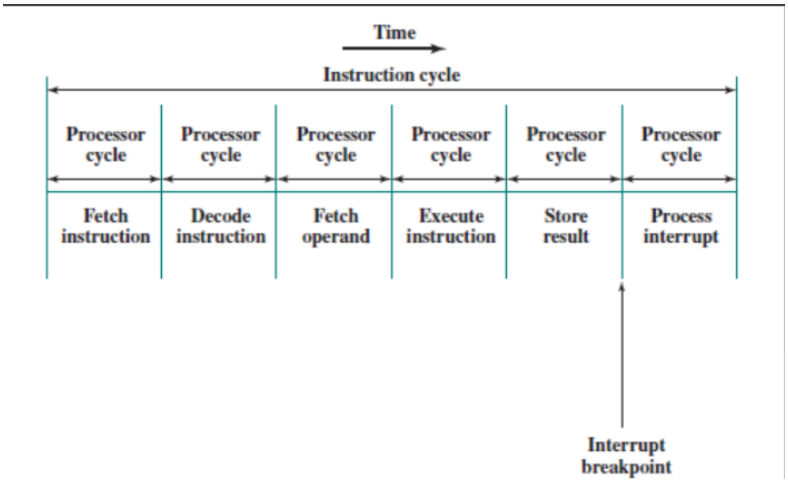
* Excepciones de programa
- División por cero: Se activa cuando un programa intenta dividir un número entre cero.
- Acceso indebido a memoria: Se genera al intentar acceder a una dirección de memoria no permitida.
- Overflow: Ocurre cuando un resultado excede la capacidad de almacenamiento del procesador.
- Instrucción inválida: Se produce al ejecutar una instrucción desconocida o ilegal para la CPU.

* Instrucciones privilegiadas

Son interrupciones que ocurren cuando un programa de usuario intenta ejecutar una instrucción reservada solo para el sistema operativo (como cambiar el modo de ejecución o gestionar dispositivos).

- Momento en que se atiende una interrupción

El procesador solo revisa si hay interrupciones pendientes al final de la ejecución de cada instrucción de máquina. Esto sucede en el pequeño intervalo entre que termina una instrucción y comienza la siguiente.



- Transferencia de control al S.O. (Handler)

Esta imagen ilustra cómo el procesador detiene momentáneamente la ejecución de un programa cuando ocurre una interrupción, ejecuta un manejador del sistema operativo y luego retorna al programa original sin perder el progreso.

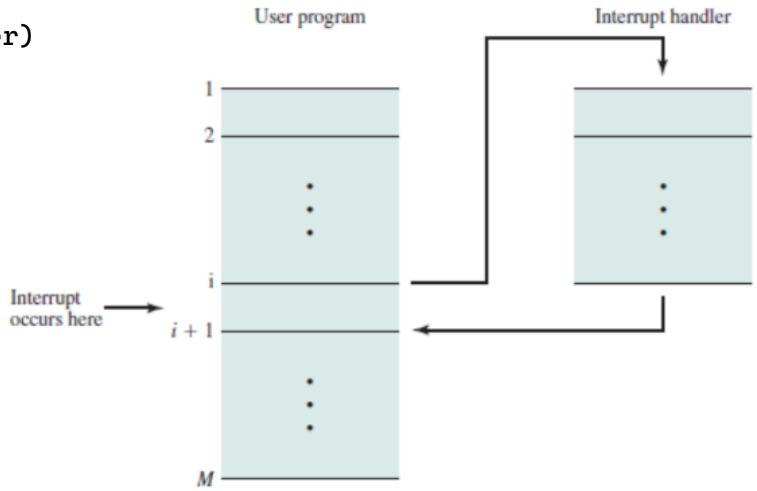


Figure 3.8 Transfer of Control via Interrupts

- Procesamiento de interrupciones

  - Parte de Hardware (lado izquierdo)

1. Un dispositivo o hardware del sistema genera una interrupción.
2. El procesador termina la ejecución de la instrucción actual antes de atender la interrupción.
3. La CPU reconoce la interrupción y envía una señal de confirmación.
4. La CPU guarda el estado actual (PSW – Program Status Word y PC – Program Counter) en la pila de control.
5. Carga la dirección del manejador de interrupciones en el PC para transferir el control al sistema operativo.

  - Parte de Software (lado derecho)

1. Se guarda el estado restante del proceso para poder retomarlo después.
2. Se ejecuta el manejador de interrupciones, resolviendo el evento.
3. Se restaura el estado del proceso interrumpido.
4. Se recuperan el PSW y PC antiguos, reanudando la ejecución del programa donde se interrumpió.

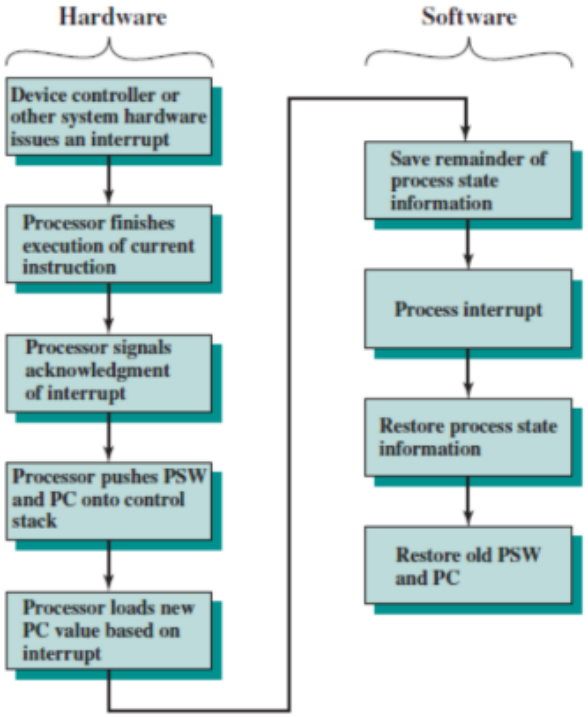
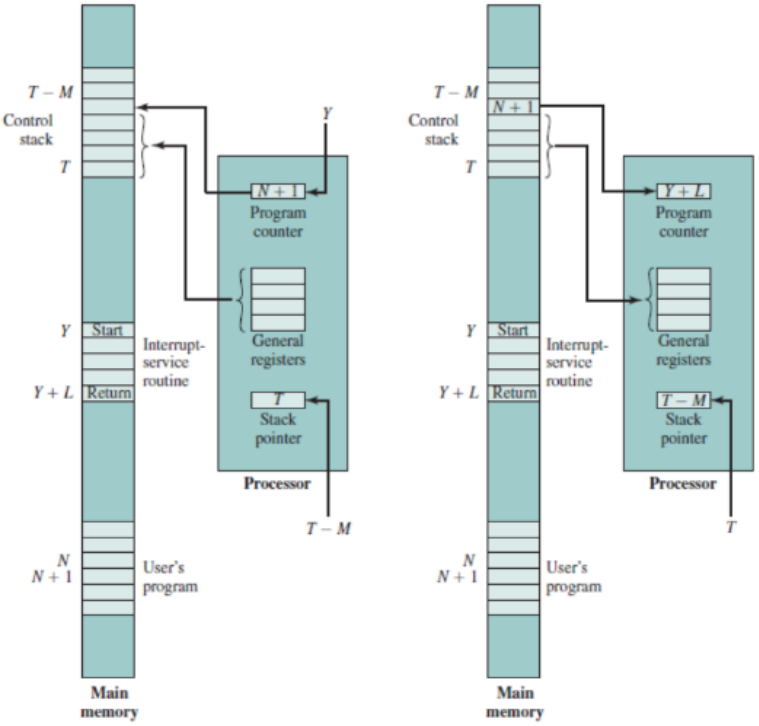


Figure 7.6 Simple Interrupt Processing

  - Ejemplo:

(a) Ocurre una interrupción después de la instrucción en N
1. El programa de usuario está ejecutando una instrucción en la dirección N.
2. Se genera una interrupción.
3. El procesador guarda el estado actual (valores de los registros y el contador de programa PC = N+1) en la pila de control.
4. La CPU cambia el PC para ejecutar la rutina de servicio de interrupción (ISR) en la memoria.

(b) Retorno de la interrupción
1. Una vez que la ISR finaliza su trabajo, se recupera el estado guardado de la pila de control.
2. Se restaura el program counter (PC = N+1) y los registros.
3. El programa de usuario continúa su ejecución donde se detuvo.

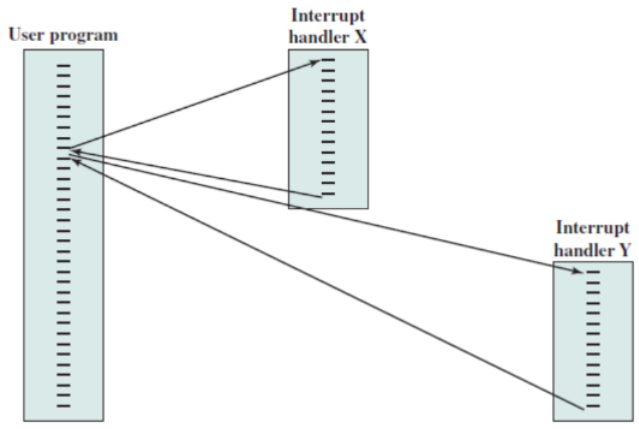


- Múltiples interrupciones

  - Deshabilitar interrupciones (secuencia):

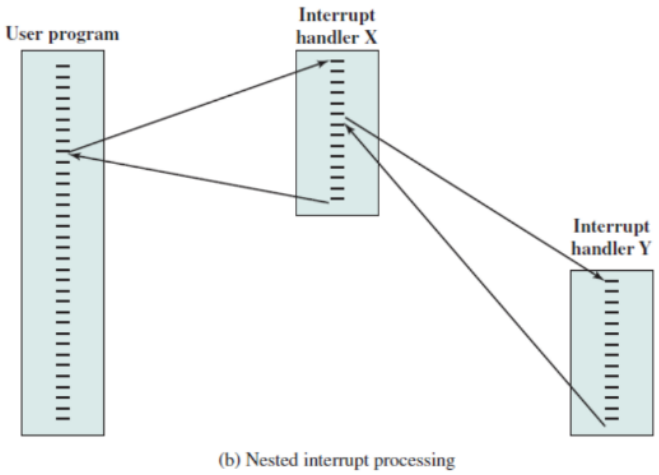
Funciona manejando una interrupción completamente antes de atender otra. Es decir, el sistema no permite interrupciones anidadas; en su lugar, cada interrupción debe finalizar antes de que se procese la siguiente.

Al no permitir interrupciones de mayor prioridad, las tareas urgentes, como recibir datos de la red, pueden no ser procesadas a tiempo y perderse si el sistema está ocupado con una tarea menos importante.



○ Priorizar interrupciones (anidadas)

Este enfoque resuelve el problema anterior al permitir que, si ocurre una interrupción de mayor prioridad mientras se está manejando una de menor prioridad, el sistema suspenda temporalmente la interrupción menos prioritaria para atender la nueva interrupción urgente.



(b) Nested interrupt processing

Figure 3.13 Transfer of Control with Multiple Interrupts

○ Ejemplo:

* Tres dispositivos de E/S
1. Línea de comunicación (Prioridad 1)
2. Disco (Prioridad 2)
3. Impresora (Prioridad 3)

* Eventos
1. T=10 – Interrupción de Impresora
2. T=15 – Interrupción de línea de comunicación
3. T=20 – Interrupción de disco

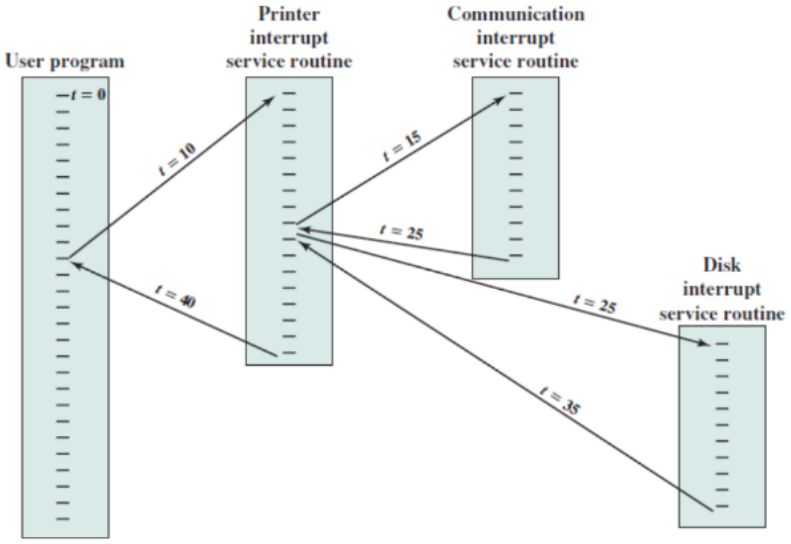


Figure 3.14 Example Time Sequence of Multiple Interrupts

· Procesador

● Arquitectura de procesadores
○ Historia
* Primero orientado al hardware (hasta los ‘70)
* Luego orientado al software (a partir de los ‘80)
○ CISC vs RISC

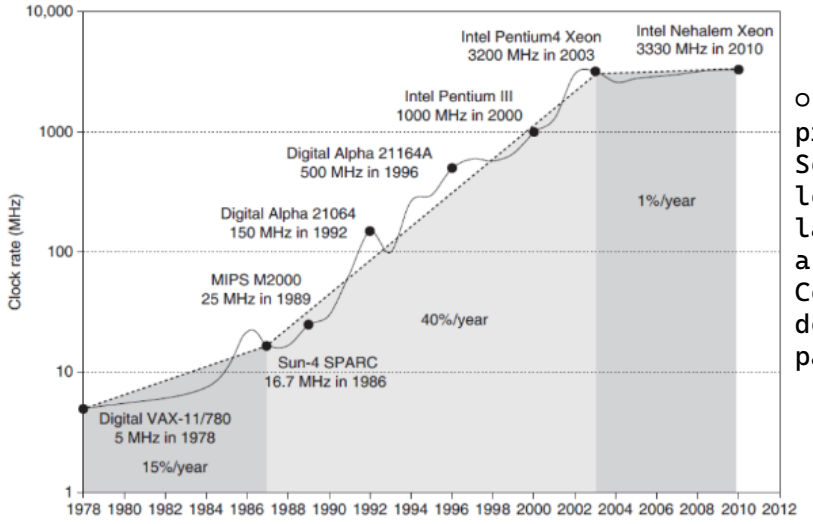
● CISC (Complex Instruction Set Computer)
○ Pocos registros de procesador (especializados)
○ Set de Instrucciones amplio
○ Muchas instrucciones para trabaja con memoria
○ Microarquitectura en sofware/hardware compleja
○ Instrucciones complejas (más de un ciclo de reloj)
○ Varios modos de direccionamiento
○ Muchos tipos de datos
○ Muchos formatos de instrucción (variables o híbridos)
○ Orientado al hardware, compiladores relativamente simples (tamaño de código pequeño)
○ Ejemplos: VAX, Intel x86 (hasta IA -32), Intel-64, IBM Mainframe, Motorola 68k

● RISC (Reduced Instruction Set Computer)
○ Muchos registros de procesador de uso general
○ Set de Instrucciones pequeño
○ Solo acceso a memoria a través de LOAD/STORE
○ Microarquitectura en hardware simple
○ Instrucciones simples (un ciclo de reloj)
○ Pocos modos de direccionamiento
○ Pocos tipos de datos
○ Pocos formatos de instrucción (fijos)
○ Orientado al software, compiladores relativamente complejos (tamaño de código largo)
○ Ejemplos: SPARC, MIPS, ARM, Intel Itanium (IA-64)

● Ecuación de Performance:
Mide el rendimiento de un procesador, calculando el MIPS rate como el producto de la frecuencia del reloj (f) y las instrucciones por ciclo (IPC).

● Uniprocesadores:
Sistemas con un solo procesador ejecutando instrucciones secuenciales.
○ Taxonomía de Flynn:
Es una clasificación de arquitecturas de computadoras basada en la cantidad de flujos de instrucciones y de datos que pueden ser procesados simultáneamente.
* SISD: Un solo procesador y un solo flujo de datos (uniprocesadores).
* SIMD: Un solo flujo de instrucciones operando sobre múltiples datos.
* MISD: Varios flujos de instrucciones para un solo dato (no utilizado comercialmente).
* MIMD: Múltiples flujos de instrucciones operando sobre múltiples datos.

○ Limitación de la velocidad del reloj (calor):
A medida que aumenta la frecuencia del reloj, el procesador genera más calor, lo que limita la velocidad de operación



○ Grafico de velocidad de reloj en un procesador:
Se observa que el aumento en la velocidad de los procesadores se ha ralentizado debido a la generación de calor, que se intensifica a medida que la frecuencia del reloj aumenta. Como resultado, surgió la necesidad de desarrollar tecnologías de refrigeración para seguir mejorando el rendimiento.

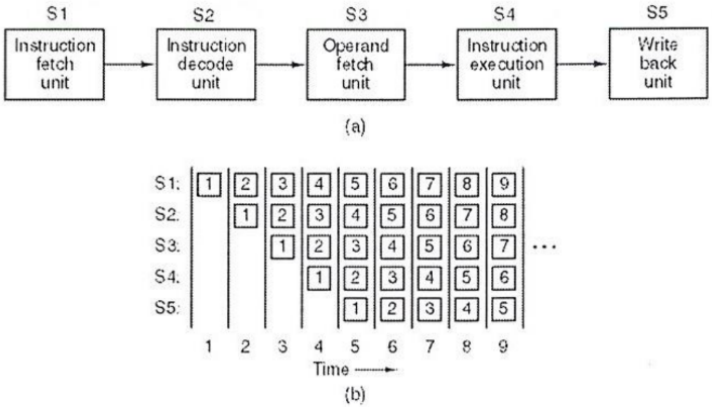
● Paralelismo
Debido a las limitaciones en el aumento de la frecuencia del reloj, los fabricantes comenzaron a centrarse en mejorar el rendimiento mediante técnicas que aumentan la cantidad de instrucciones procesadas por ciclo.
El paralelismo es una estrategia que permite la ejecución simultánea de múltiples instrucciones, con el objetivo de incrementar el número de instrucciones procesadas por segundo sin depender exclusivamente del aumento de la frecuencia del reloj.

○ Técnicas
* A nivel instrucción
- Pipelining
- Dual pipelining
- Superscalar
- Multithreading
* A nivel procesador
- Procesadores paralelos de datos
- Multiprocesadores
- Multicomputadores

● Pipelining
El pipelining (o "tubería de instrucciones") es una técnica utilizada en los procesadores para optimizar la ejecución de instrucciones al permitir que múltiples etapas del procesamiento de una instrucción se solapen. En lugar de ejecutar una instrucción completa antes de comenzar la siguiente, las instrucciones se dividen en diferentes etapas de ejecución, y cada etapa puede procesar diferentes instrucciones al mismo tiempo.

○ Cómo funciona el Pipelining:
Imagina una línea de producción en una fábrica donde diferentes trabajadores realizan tareas consecutivas. Cada trabajador (o etapa) realiza una parte del trabajo, y cuando un trabajador termina su parte, pasa la tarea al siguiente, mientras que él mismo comienza a trabajar en una nueva tarea. De manera similar, el procesador utiliza el pipelining para que cada etapa de procesamiento esté ocupada con diferentes instrucciones.

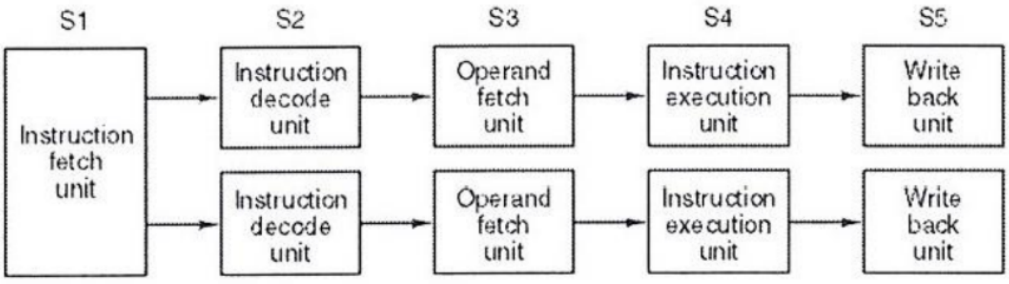
En un procesador, el pipelining divide la ejecución de una instrucción en varias etapas, como las siguientes (dependiendo del procesador):
* Fetch (Búsqueda): Obtener la instrucción desde la memoria.
* Decode (Decodificación): Decodificar la instrucción y determinar qué operación realizar.
* Execute (Ejecución): Realizar la operación especificada por la instrucción (por ejemplo, una suma).
* Memory (Memoria): Acceder a la memoria, si es necesario, para leer o escribir datos.
* Writeback (Escritura de resultados): Escribir los resultados de la operación en los registros o en la memoria.



○ Ventajas del Pipelining:
* Mayor rendimiento: Al solapar las etapas de las instrucciones, el procesador puede ejecutar más instrucciones en un menor período de tiempo. Esto aumenta el throughput del procesador, es decir, la cantidad de instrucciones ejecutadas por unidad de tiempo.
* Uso eficiente de recursos: Cada etapa del pipelining puede estar ocupada con una parte diferente de la ejecución de las instrucciones, lo que maximiza el uso del hardware.

○ Control de dependencia de instrucciones
Estas dependencias son situaciones en las que una instrucción depende de los datos o el resultado de una instrucción anterior, lo que puede generar conflictos o riesgos en la ejecución de instrucciones en un pipeline.
Cuando varias instrucciones se solapan en un pipeline, el procesador debe tener cuidado de que las instrucciones que están en diferentes etapas del pipeline no interfieran entre sí, especialmente si una instrucción necesita los resultados de una instrucción anterior que aún no ha terminado. Si no se gestionan correctamente estas dependencias, puede haber errores o bloqueos en el pipeline, lo que afectaría el rendimiento.

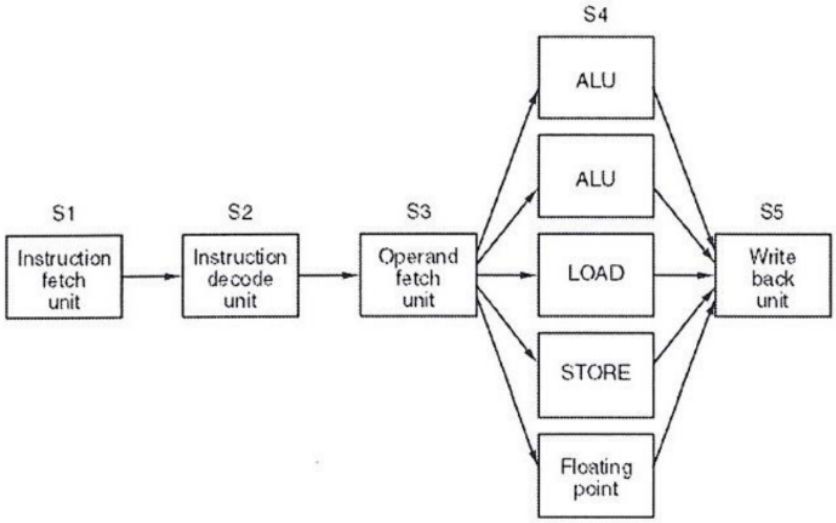
● Dual Pipelining:
El dual pipelining se refiere a la utilización de dos pipelines independientes dentro de un procesador. En lugar de tener un solo pipeline para procesar las instrucciones secuencialmente, el procesador tiene dos pipelines que permiten la ejecución paralela de dos instrucciones al mismo tiempo. Cada pipeline maneja un conjunto específico de instrucciones o diferentes etapas de ejecución, lo que permite una mayor eficiencia y utilización de los recursos del procesador.
○ Cómo funciona:
El procesador divide su pipeline en dos canales.
Un canal puede estar ejecutando instrucciones relacionadas con operaciones aritméticas, mientras que el otro puede manejar instrucciones de carga/almacenamiento, por ejemplo.
Esto permite que dos instrucciones se procesen simultáneamente, aumentando el rendimiento sin necesidad de aumentar la frecuencia del reloj.



○ Ventajas:
* Incremento de la velocidad de ejecución sin necesidad de cambiar tanto el diseño del procesador. Mejora en la utilización de los recursos del procesador.

○ Desventajas:
* Complejidad adicional para gestionar la coordinación de los dos pipelines.

● Superscalar:
Un procesador superscalar es un tipo de arquitectura que puede ejecutar varias instrucciones en paralelo en un solo ciclo de reloj. La clave de la arquitectura superscalar es que tiene múltiples unidades de ejecución (ALUs, FPU, unidades de carga/almacenamiento, etc.), lo que permite que el procesador ejecute más de una instrucción por ciclo.
○ Cómo funciona:
Un procesador superscalar tiene múltiples pipelines y unidades funcionales (como unidades aritmético-lógicas, unidades de punto flotante, unidades de acceso a memoria, etc.).
Cada ciclo de reloj, el procesador puede enviar instrucciones a diferentes unidades funcionales, procesando varias instrucciones simultáneamente.
Si no hay dependencias entre las instrucciones, estas pueden ser enviadas a diferentes unidades funcionales para su ejecución en paralelo.



○ Ventajas:
* Aumento significativo del rendimiento, ya que varias instrucciones se pueden ejecutar al mismo tiempo.
* Mejor aprovechamiento de los recursos del procesador.

○ Desventajas:
* Complejidad en la planificación y el enrutamiento de instrucciones.
* Contención de recursos si hay demasiadas instrucciones para las unidades funcionales disponibles
* Necesidad de más recursos de hardware (más unidades funcionales, más registros, etc.).

● Hardware multithreading
El hardware multithreading es una técnica utilizada para mejorar la eficiencia del procesador al permitir la ejecución de múltiples hilos (threads) en un mismo núcleo.
○ ¿Cómo funciona?
Cuando un hilo de ejecución se detiene temporalmente (por ejemplo, porque está esperando datos de memoria o una operación de entrada/salida), en lugar de dejar el procesador inactivo, este cambia rápidamente a otro hilo disponible. Así, se aprovecha mejor el tiempo de procesamiento, evitando que el núcleo quede inactivo mientras espera datos. Aclarar que el propio CPU detecta cuándo cambiar de hilo sin intervención del S.O.

○ Thread (Hilo de ejecución)
Es la unidad más pequeña de ejecución dentro de un proceso.
Contiene su propio contador de programa (PC), registros y pila (stack).
Todos los hilos de un proceso comparten el mismo espacio de direcciones (address space), lo que facilita la comunicación entre ellos.
También se les llama "lightweight processes" (procesos ligeros) porque tienen menos sobrecarga que un proceso completo.

○ Proceso
Es una instancia de un programa en ejecución.
Puede contener uno o más hilos de ejecución.
Cada proceso tiene su propio espacio de direcciones y su estado, que son gestionados por el sistema operativo.

○ Cambio de contexto entre threads vs procesos
Cambiar la ejecución entre hilos dentro de un mismo proceso es rápido y eficiente porque comparten el mismo espacio de direcciones y solo se necesita modificar los registros y la pila.
Cambiar entre procesos diferentes es más costoso porque implica cambiar todo el espacio de memoria y estado del CPU, lo que requiere intervención del sistema operativo.

○ Fine-Grained Multithreading (Multithreading de grano fino)
El procesador cambia de hilo después de ejecutar cada instrucción. Se hace de forma constante, sin importar si el hilo está esperando datos o no.
* Ventaja: Se evita que un solo hilo monopolice el procesador, distribuyendo equitativamente el uso del CPU entre todos los hilos.
* Desventaja: Introduce una pequeña latencia porque el cambio de hilo ocurre con cada instrucción, lo que puede reducir el rendimiento en tareas secuenciales.
* Ejemplo: Algunos procesadores Intel IA-32 usaban este método.

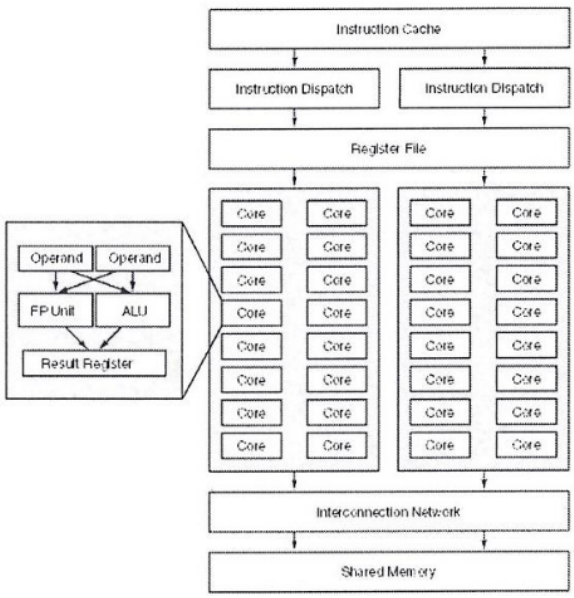
○ Coarse-Grained Multithreading (Multithreading de grano grueso)
El procesador solo cambia de hilo cuando ocurre un evento significativo, como un cache miss (cuando el procesador no encuentra los datos en la caché y debe esperarlos desde la memoria principal, lo que es más lento) o un page fault.
* Ventaja: Reduce la sobrecarga de cambiar de hilo constantemente, ya que solo lo hace cuando realmente es necesario.
* Desventaja: Si no ocurre un evento que cause el cambio de hilo, un solo hilo puede monopolizar el procesador.
* Ejemplo: Procesadores Intel Itanium 2 utilizaban este enfoque.

● Procesadores paralelos de datos
Estos procesadores están diseñados para procesar grandes cantidades de datos en paralelo.
○ Una sola unidad de control:
Un solo controlador dirige la ejecución de instrucciones para varios procesadores.

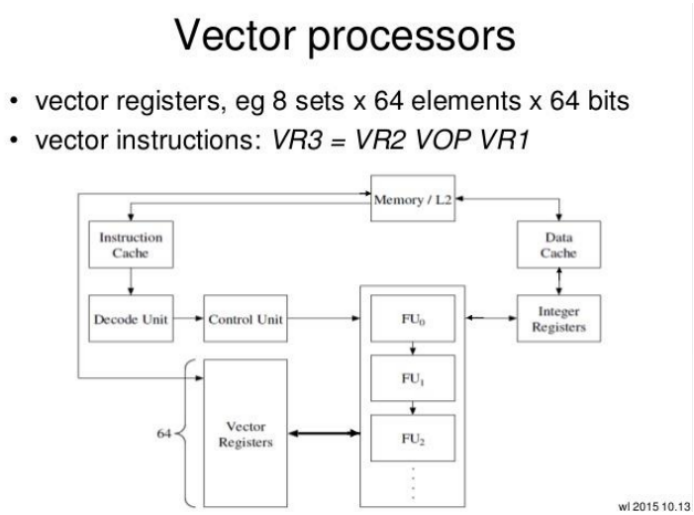
○ Múltiples procesadores:
Varias unidades de procesamiento trabajan en paralelo sobre diferentes datos para acelerar el rendimiento.

○ Métodos:

* SIMD (Single Instruction Multiple Data)
Un solo conjunto de instrucciones se ejecuta sobre múltiples datos simultáneamente. Ideal para operaciones matemáticas y gráficas.
- Ejemplo: GPU (Nvidia Fermi GPU). Las GPU tienen cientos o miles de núcleos pequeños optimizados para ejecutar SIMD, ideal para gráficos y cálculos científicos.



* Vectoriales
Similar a SIMD, pero en vez de trabajar directamente con datos en memoria, usan registros vectoriales.
- Registro vectorial:
Conjunto de registros convencionales que se cargan desde memoria con una sola instrucción, acelerando el procesamiento.
- Opera por pipelining:
Se divide la ejecución en etapas para procesar múltiples instrucciones al mismo tiempo.
- Ejemplo: Intel Core (SSE – Streaming SIMD Extensions). Tecnología de Intel que optimiza operaciones SIMD para mejorar el rendimiento en multimedia, procesamiento de imágenes y cálculos científicos.
- Ejemplo de una instruccion:
*Registro A: [10, 20, 30, 40]
*Registro B: [5, 5, 5, 5]
*Instrucción: A = A + B → Resultado: [15, 25, 35, 45] (4 sumas en paralelo).



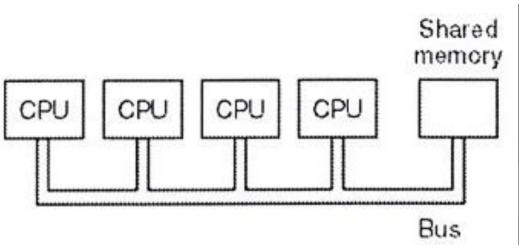
● Multiprocesadores:
Son sistemas con múltiples CPUs que trabajan juntas y comparten memoria para mejorar el rendimiento en tareas complejas.

○ Múltiples CPUs que comparten memoria común:
Varios procesadores acceden a una memoria compartida, lo que permite comunicación y coordinación entre ellos.

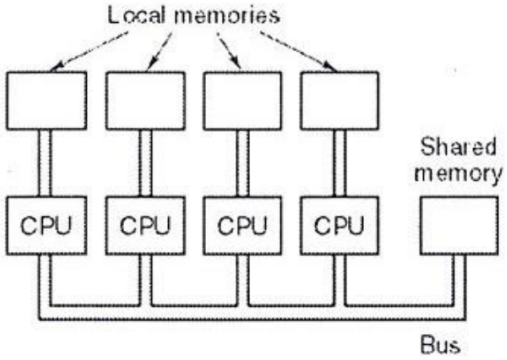
○ MIMD (Multiple Instruction Multiple Data):
Cada CPU ejecuta instrucciones diferentes en conjuntos de datos distintos, permitiendo verdadero paralelismo en tareas diversas.

○ CPUs fuertemente acoplados:
Los procesadores están conectados entre sí y comparten memoria en una arquitectura bien integrada para optimizar el acceso a datos.

○ Diferentes implementaciones de multiprocesadores:
* Single bus y memoria compartida (centralizada) (UMA – Uniform memory access) (SMP – Symmetric multiprocessor):
Todas las CPUs acceden a la memoria a través de un único bus de comunicación.
- Ejemplo: Intel Core i7



* CPUs con memoria local y memoria compartida (NUMA – Non-Uniform Memory Access)
Cada procesador tiene su propia memoria local, pero también puede acceder a una memoria compartida.
- El acceso a la memoria no es uniforme: la memoria local es más rápida que la compartida.
- Ejemplos de arquitecturas NUMA: BBN Butterfly, SGI Origin 2000, Compaq AlphaServer GS320, Intel Itanium 2



● Multicomputadores
- Computadores interconectados con memoria local (memoria distribuida)
- No hay memoria compartida
- CPUs ligeramente acoplados - Clusters
- MIMD (Multiple Instruction Multiple data)
- Intercambio de mensajes
- Topologías de grillas, árboles o anillos
- Ej. IBM Blue Gene/P