

Componentes de un computador

Memoria

Se le denomina memoria al conjunto de elementos de una computadora destinados al almacenamiento de información. Dentro de las características de la misma entra la temporalidad, que puede ser volátil o persistente (ram o disco duro). Luego hay todo un mundo que es el almacenamiento secundario.

Se pueden diferenciar cualidades como la tecnología, la organización, el performance y el costo. Se identifica una jerarquía en memoria que se divide entre internos y externos al sistema:

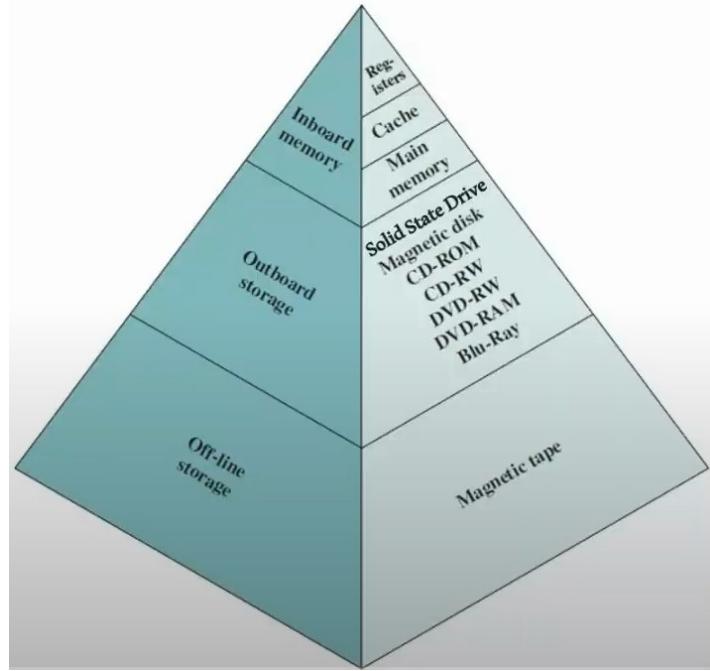
○ Memoria

- Componente complejo (Sistema de memoria)
- Formado por elementos con distintas cualidades:
 - Tecnología
 - Organización
 - Performance
 - Costo
- Jerarquía de subsistemas de memoria
 - Internos al sistema (accedidos directamente por el procesador)
 - Externos al sistema (accedidos por el procesador a través de un módulo de E/S)

Dentro de estos subsistemas se mide la **capacidad, el tiempo de acceso y el costo**. De acuerdo a que tanta capacidad, que tan veloz es y que tan alto es el costo, el elemento estará más alto en la “pirámide” de jerarquía. Los subsistemas de memoria se ven comprometidos al hacer un balance entre estas características.

○ Jerarquía de memoria

- 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



Memoria interna (guardado de información volatil / temporal)

El primer elemento que figura en la piramide son los registros y los tomamos como un almacenamiento interno. Se los reconoce como un conjunto de biestables, donde se permite almacenar pequeñas cantidades de información en forma de bits . Almacena los elementos de forma temporal, no se persiste. Las instrucciones cargadas en los mismos suceden millones de veces por segundo. Se encuentran por encima de la piramide ya que son los elementos más rápidos gracias a su definición tecnológica y esta dentro de la UAL. También se destaca que su costo es elevado y su capacidad es pequeña.

El segundo elemento, la memoria caché existe para mejorar la performance de la pc, a partir del guardado de datos de cierta información dentro de un proceso. Se la reconoce como una memoria intermedia entre el cpu y la memoria principal. La memoria cache es la segunda más rápida y de menor capacidad, aún así son muchisimo más grandes que los registros. Son más económicas que los registros también.

El tercer escalón es la memoria principal o memoria RAM, en forma de chips constituidos por circuitos semiconductores, donde se persiste la información de forma electrónica. Constituye un circuito un poco más lento que la memoria cache, pero si es bastante más barato.

Hasta acá llegamos al límite de la memoria interna. Todos estos elementos están interconectados a través de un bus.

Memoria externa (Persistencia de datos)

Lo que le sigue se consideran almacenamientos externos que se conectan con el cpu con la entrada y salida del mismo.

Los dispositivos de estado sólido son muy similares a la memoria ram en cuanto a lo tecnológico (la información se guarda bajo una configuración electrónica), su principal diferencia es que la información persiste y que su capacidad de almacenamiento es mucho mayor.

El disco magnético es el siguiente en jerarquía, siendo aún común en el mercado. Este es un medio con elementos mecánicos. Tienen capacidades muchísimas más grandes que lo anterior (no así respecto a un disco de estado sólido), el costo es menor y su velocidad es menor dada sus características tecnológicas. Tanto un disco sólido como uno magnético se utilizan para la transacción de datos que requerirán los softwares de las computadoras para realizar dicho proceso, no se suelen utilizar para persistir datos en muchísimo tiempo (en el caso hipotético de una empresa por ejemplo). Para esto se utilizan cintas / cartuchos magnéticos o dispositivos ópticos.

La cinta magnética es un medio de almacenamiento histórico de información que mantiene más usabilidad que los medios ópticos. Tiene capacidad similar a un dispositivo de estado sólido pero en cuanto a velocidad es muy lento ya que lee de forma secuencial la información. No se puede saltar a una posición particular de la cinta, si no que hay que recorrer toda la información anterior (como en un cassette).

● Jerarquía de memoria

- 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

Accesos en memoria

Entre los distintos tipos de memoria es importante destacar la diferencia entre un acceso secuencial y directo. El secuencial ya se mencionó y el directo es aquel que a partir de una dirección única para bloques, basada en su posición física se puede acceder directamente. El tiempo de acceso es variable.

Luego se encuentra el acceso aleatorio, donde cada posición direccionable en memoria se accede con la misma velocidad dado el mecanismo de direccionamiento cableado físicamente. El tiempo de acceso es constante e independiente de la secuencia de accesos anteriores. Lo aleatorio acá es que no importa cuando pidas acceso, la velocidad es la misma. Distinto es en un disco magnético, que depende de donde este la lectora o la degradación del material donde está grabado, por ejemplo.

Por último tengo accesos asociativos que a partir de una comparación de patrón de bits se accede o no, esto se ve en la memoria caché. En vez de buscar una dirección, busca un patrón de bits particular. La velocidad también es constante.

Parámetros de performance

Sabemos que para cualquier opción elegida voy a tener que realizar el proceso de acceder a la información. A esto se lo reconoce como tiempo de acceso o latencia. Esto no es igual en todos los elementos en memoria. En la ram se refiere al tiempo que se necesita para ejecutar una operación completa de lectura o escritura, siendo indistinto a que celda de memoria voy. Me interesa cuánto tarda en llegar y persistirse. Ahora, cuando uno se refiere a un disco rígido nos referimos a dejarle lista la lectora/grabadora para ser utilizada, pero no incluye el tiempo en si de la lectura / escritura:

○ Sistema de memoria

● Características

○ 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
 - Memorias de acceso aleatorio: tiempo de acceso más el tiempo adicional necesario para que una nueva operación pueda comenzar

Luego se mide como parámetro de performance el **tiempo de ciclo** en memoria. Este se lo define como el tiempo de latencia más el tiempo adicional que se requiere para estar lista la memoria para hacer otra operación.

Lo que le sigue en parámetro de performance es la tasa de transferencia:

- **Sistema de memoria**
 - Características
 - Parámetros de performance
 - Tasa de transferencia
 - Tasa con la cual los datos son transferidos dentro o fuera de la unidad de memoria
 - Memorias de acceso aleatorio: $1/\text{Tiempo de ciclo de memoria}$
 - Memorias sin acceso aleatorio:
$$T_n = T_A + n/R$$

donde

T_n = Tiempo promedio para leer o escribir n bits

T_A = Tiempo promedio de acceso

n = Número de bits

R = Tasa de transferencia, en bits por segundo (bps)

Esto refiere cuantas unidades de información por unidad de tiempo pueden ser transferidas desde o hacia esa memoria. Esto varia según si es una memoria de acceso aleatorio (ram) tengo que de forma genérica será $1/\text{tiempo de ciclo de memoria}$. Para cualquier otro tipo de memoria es la formula indicada: El tiempo promedio de escritura / lectura es el tiempo de acceso sumado a la cantidad de información que quiero guardar (n) dividido la tasa de transferencia (R).

Fisicamente, se separan las memorias de la siguiente manera:

Características

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

La volatidad se define de como se comporta con la falta de electricidad. Luego tengo memorias que pueden estar en uno periférico o circuito interno (algunas memorias semiconductoras) que se definen como ROM (Read Only Memory).

Características

○ 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)

Hay una categorización de todas las memorias semiconductoras:

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level	Nonvolatile	
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

Las memorias RAM tienen distintas características tecnológicas. Tengo rams estáticas y dinámicas que varía su electrónica más que nada.

Memoria caché

El concepto de caché responde al **Principio de localidad de referencia**. Esto ocurre para todo software. Cuando se ejecuta un programa, los accesos a la memoria ram a las instrucciones y datos tienden a agruparse, tienden a estar juntos por ejemplo en un loop, subrutina, tabla, vectores, etc:

● Jerarquía de memoria

- 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)

La idea es guardarme cosas en la memoria caché ya que es más rápida que la memoria ram. La idea de guardarme ciertas porciones de memoria es para que la siguiente vez que yo ejecute lo que le sigue a dicho bloque, empiece justo desde ahí. Cuando hago un loop por ejemplo, voy momentaneamente a la caché para hacerlo más rápido. No así sucede cuando hago una bifurcación. Lo que pasa es que a nivel probabilístico tiendo a que esto de guardarme en bloques me sirva para acceder a la siguiente instrucción más rápido, además de darme más opciones de acceso en general.

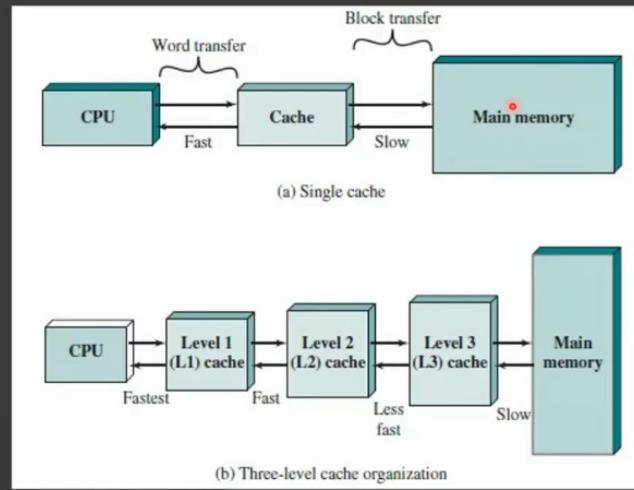
● Memoria Cache

- 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

En general entonces, sabemos que la memoria cache es semiconductora al igual que la ram pero es más rápida y costosa que esta.

Ahora el procesador no le va a pedir directamente a la memoria principal, si no que primero se lo pide a la cache y esta es la que va a buscar a la ram lo que se solicita, es por eso que se encuentra entre dos elementos.

○ Memoria Cache

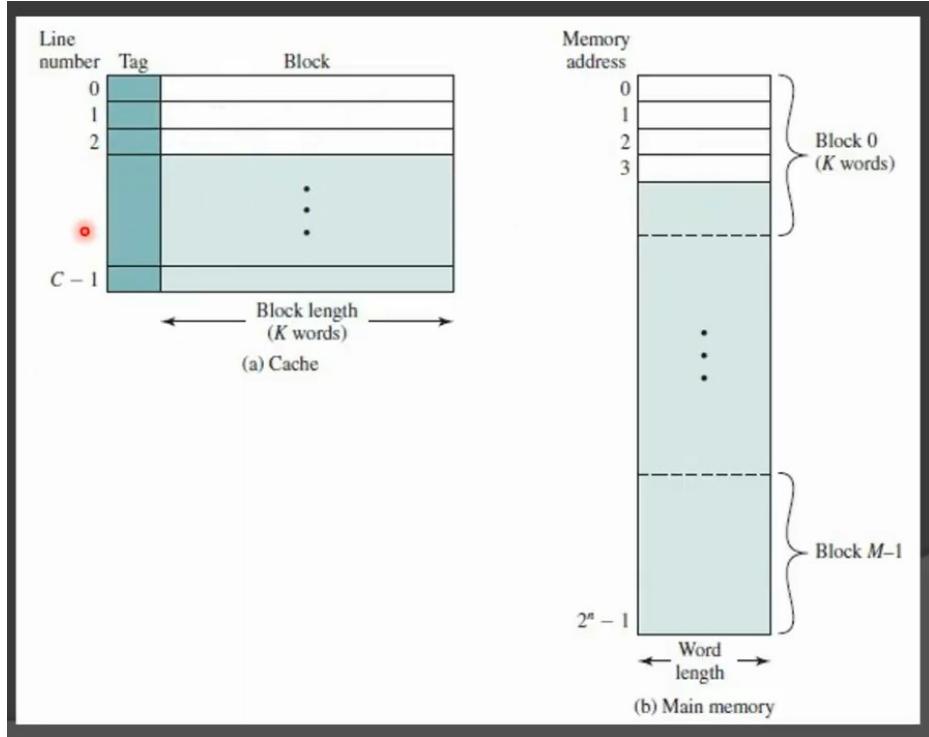


Este gráfico me dice que el cpu con la caché tiene una velocidad más rápida de transferencia y de la memoria principal a la caché es más lenta. A partir de este guardado de bloques es que ahorro tiempo. Cuanto menor sea el nivel, más rápida, menos capacidad y más costosa es.

○ Memoria Cache

- Cómo funciona
 - CPU trata de leer una palabra de la memoria principal
 - Se chequea primero si existe en la memoria cache.
 - Si es así se la entrega al CPU
 - 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
 - Por el principio de localidad de referencia es probable que próximas palabras buscadas estén dentro del bloque de memoria subido a la cache

Mediante un sistema de referencia por tags es que se enlazan los bloques guardados en la cache referenciando a porciones que se encuentran en la memoria principal.



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)

Los bits de control me pueden advertir si hay una modificación de un dato dentro de la ram, referenciado en la cache, se modificó de modo que me provoque una inconsistencia.

Se suele subdividir la memoria cache para guardar datos y otra parte para instrucciones de modo que la memoria principal en si suele tener separado estas partes, por ende tiene sentido que así sea también en la caché.

Processor	Type	Year of Introduction	L1 Cache ^a	L2 Cache	L3 Cache
Pentium 4	PC/server	2000	8 kB/8 kB	256 kB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA ^b	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	2002	32 kB	256 kB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB
Intel Core i7 EE 990	Workstation/ server	2011	6 × 32 kB/ 32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ server	2011	24 × 64 kB/ 128 kB	24 × 1.5 MB	24 MB L3 192 MB L4

Memoria principal

A la hora de como administrar la memoria RAM toma rol fundamental el sistema operativo. Este es un software que está presente en todos los ámbitos computacionales, salvo en excepciones como microprocesadores.

En si el sistema operativo es el software encargado de administrar los recursos del hardware, provee servicios y controla la ejecución de los programas.

Uno de los servicios principales es el **schedule de procesos** que monitorea todos los procesos en ejecución en ese momento en la memoria principal de la computadora. Lo que hace el sistema operativo es asignarle una porción de tiempo de cpu para cada uno de esos procesos. Se muestra bajo que criterio se administra el tiempo del cpu el sistema operativo.

Otro servicio es la administración de la memoria. El sistema operativo actua de gobierno ante el uso de la misma al ser un recurso escaso. El servicio de la administración de la memoria ayuda a entender como se interconecta el hardware para funcionar en su conjunto.

○ Administración de Memoria

- 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
 - Administración de memoria
 - Monitor
 - Parte residente del Sistema Operativo

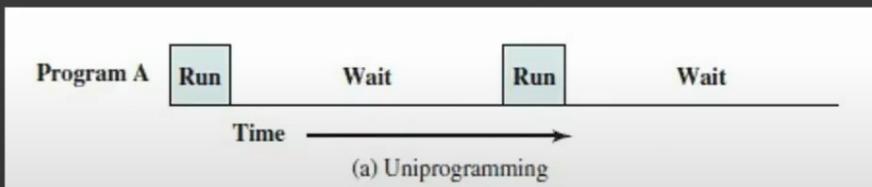
Uniprogramación

Hace referencia a aquellos cpus que tienen un unico software ejecutando sin necesidad que haya un sistema operativo por encima que lo gestione. Hay un unico proceso a la vez ejecutandose. No es el caso masivo pero es una solución para ciertos casos para la administracion de memoria.

Cuando hay un unico proceso sucede que toda la memoria esta empleada por este. En este caso el cpu alterna entre un tiempo de espera del proceso en cuestion hasta que vuelve a haber otra petición. El tiempo “ocioso” del cpu es un desperdicio de energia.

○ Uniprogramación

- Un solo proceso de usuario en ejecución a la vez
- La memoria de usuario está completamente disponible para ese único proceso
- Uso del procesador a lo largo del tiempo



- Run: Tiempo efectivo de uso del CPU
- Wait: Tiempo ocioso del CPU esperando E/S (*Idle time*)

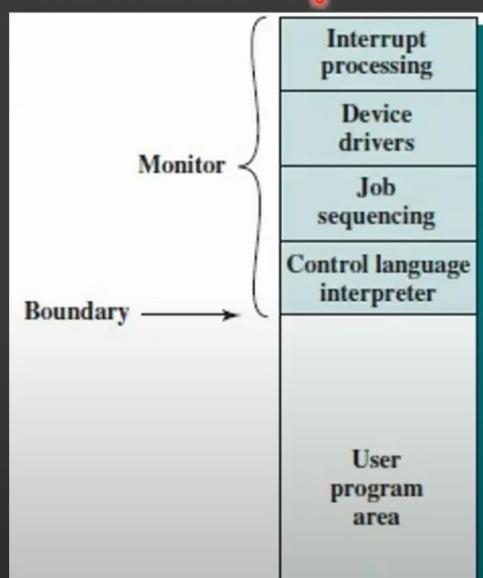
○ Administración de memoria simple

- 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)

Hay una pequeña parte llamada monitor que el sistema operativo gestiona para su propia ejecución. Lo único que hace es subir y bajar el proceso cuando lo necesita.

○ Administración de memoria simple

- Memoria



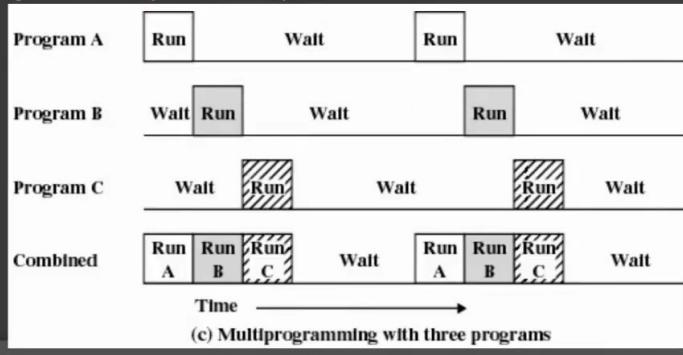
Multiprogramación

Este es el contexto universal, donde varios procesos de usuario se ejecutan a la vez. Sucede que hasta hay varios procesos de un mismo software, por ejemplo un navegador maneja los múltiples procesos.

En este contexto el sistema operativo debe gestionar el espacio que se tenga de ram. Cada vez que un proceso usa el cpu, el resto se encuentra en espera. Esa porción de tiempo que se le da al cpu para cada programa se le llama timeslice (porción de tiempo).

● 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)



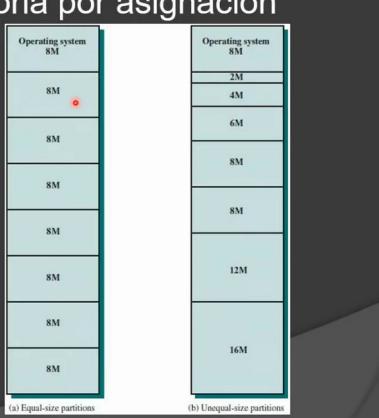
Un proceso puede finalizar por distintas condiciones. Una puede ser porque el usuario lo termino o el proceso en si llego a su fin. También puede terminar por un error o por una suspensión. Esta suspensión es cuando el sistema operativo lo “frizza”. Un proceso puede suspenderse porque ese lapso de tiempo que le dio el sistema operativo termino y para concluir el proceso hay que esperar a que el so le vuelva a dar el tiempo para terminarlo (termina el timeslice). Otra razón muy común es cuando el proceso inicia alguna operación de entrada o salida (cualquier interacción con un periférico). La capacidad de que un proceso pueda hacer esto son soluciones para maximizar el rendimiento del cpu.

Administración de memoria por asignación particionada

En estos contextos, la memoria ram se dividía en porciones iguales o de distinto tamaño pero siempre fijas. El sistema operativo a cada proceso le asigna un bloque. No se podía compartir con otros procesos las asignaciones, si había procesos que requerían menor tamaño quedaba sin usar una parte.

● Admin. de memoria por asignación particionada

- Particiones Fijas
 - Iguales
 - Distintas

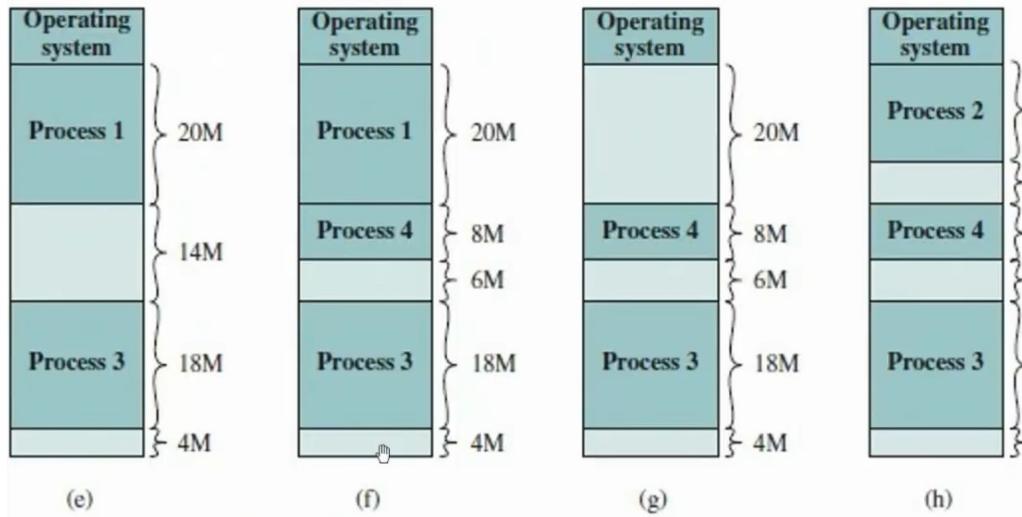


Esta era una forma relativamente sencilla de gestionar multiprocesos, como ventaja. Como desventaja había estos desperdicios de memoria en el sentido de una fragmentación interna (cuando te quedan partes parciales de un bloque sin usar) o cuando te quedaba memoria disponible pero no bloques del tamaño que se precisa para un proceso en particular.

- 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)
 - Fragmentación externa (particiones no usadas)
 - Ej. IBM OS/MFT (Multiprogramming with a Fixed number of Tasks)

Asignación particionada reasignable

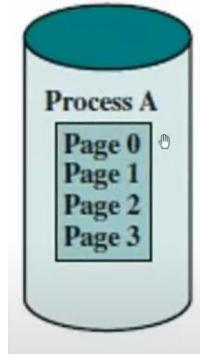
Esta en vez de asignar a los bloques tamaños fijos, se le asignaba a cada proceso la cantidad de memoria que requería. A medida que le voy asignando procesos a la memoria, va quedando ciertas partes libres. El problema estaba que a medida que sacaba o ponía procesos me quedan porciones sin usar de la memoria:



Mediante un recurso de hardware lo que se hacía es tomar todos estos pequeños bloques vacíos y unificarlos para poder aprovechar el espacio y poder alojar otro proceso potencial. Esta relocación de procesos consumía recursos y esa es una de las principales desventajas.

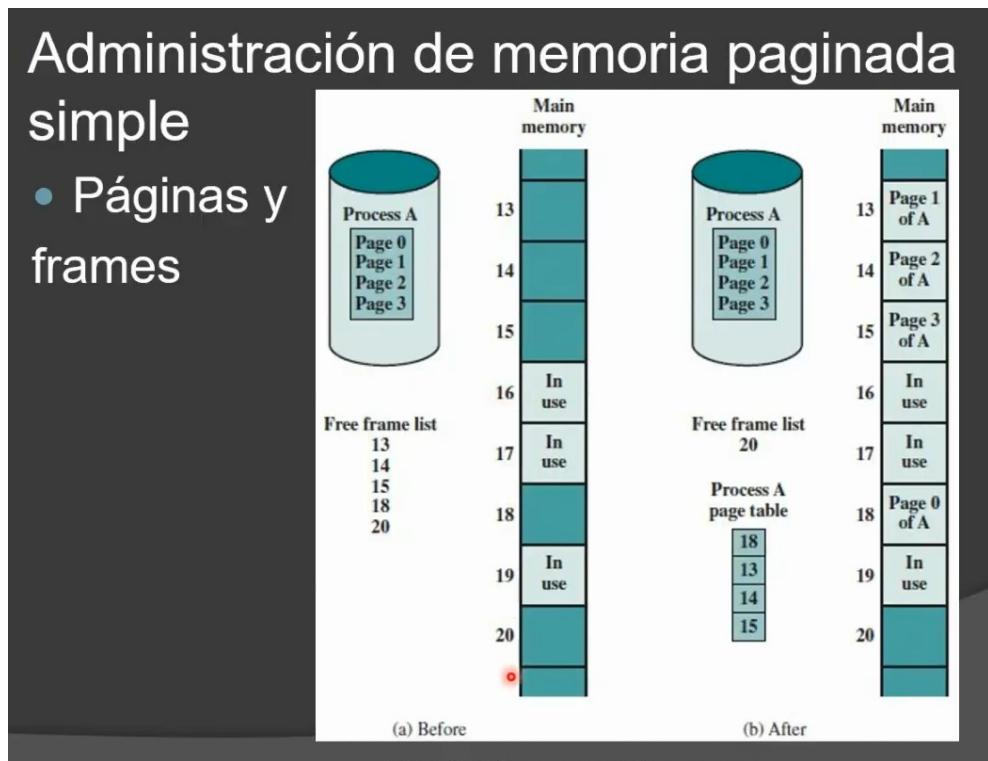
Administración de memoria paginada simple

Este es casi igual al que se usa hoy en día. Mediante esta forma de subdividir se obtiene mucha versatilidad a partir de, en forma lógica subdividir a cada proceso en partes iguales a las que llama pagina. Todo el espacio de direcciones se mapea en páginas:



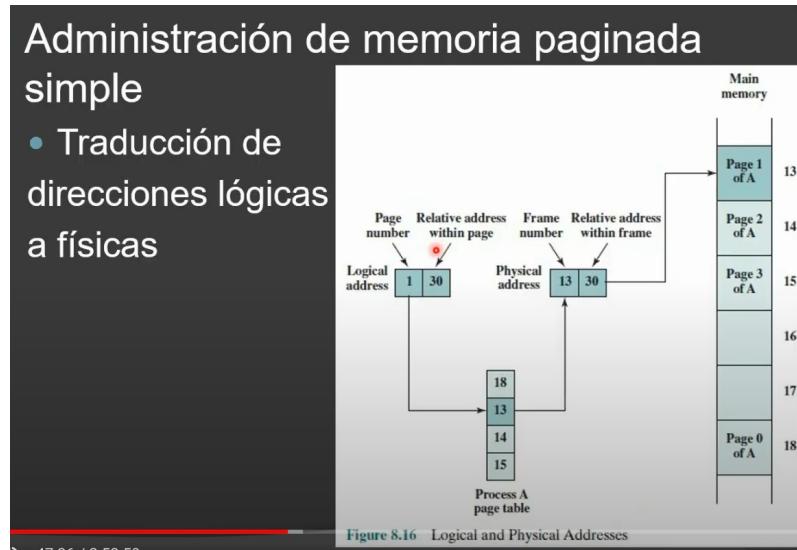
Por ejemplo, en arquitectura intel de 32 bits, cada página ocupa 4kb.

A su vez, la memoria ram tambien se subdivide en partes iguales pero se llaman frames. Allí se van subiendo cada una de las páginas del proceso. En vez de meterlo todo junto como un bloque se partitiona y se va metiendo de una forma más dinámica y versatil.



De esta manera el sistema operativo identifica dos estados: Libre y en uso. El so tiene una lista de los frames que estan libres que se pueden utilizar para subir paginas de los procesos. De esta forma los procesos no necesariamente estan todos juntos y ni siquiera estan en orden las paginas.

Para poder acceder a cada página distribuida por la memoria tengo que poder traducir la posición relativa lógica mediante la que yo programe a la dirección real donde se va a guardar en la memoria. Esto se hace a partir de un hardware dedicado:



A partir de un número de página y una dirección relativa desde esa página hacia la dirección que quiero traducir, como el sistema operativo sabe en qué frame se guarda (sabiendo el sistema operativo cuál es la dirección real de este frame) hay que realizar la traducción lógica a una traducción física. Se traduce la página y el offset en un frame y el offset. A cada página se le asigna un frame.

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)
- Se divide la memoria principal en partes iguales (frames)
- Hay una tabla de páginas por proceso
- Hay una lista de frames disponibles
- Se cargan a memoria las páginas del proceso en los frames disponibles (no es necesario que sean contiguos)
- Las direcciones lógicas se ven como número de página y un offset
- Se traducen las direcciones lógicas en físicas (*address translation*) con soporte del hardware (MMU – Memory Management Unit)
- La paginación es transparente para el programador

Este modo de gestionar la memoria permite un uso más dinámico y eficiente y se evita la fragmentación interna y no existe fragmentación externa: cualquier frame libre puede usarse para cualquier página. Entre otras ventajas:

Administración de memoria paginada simple

- 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

La gran desventaja es la que se indica, hay que subir todas las páginas a memoria.

Administración de memoria paginada por demanda

Es la forma por defecto en la que hoy en dia se administra la memoria, salvo en algún nicho de mercado. El hardware debe estar construido para soportar esta forma de administración (debe tener el MMU, por ejemplo).

Lo que cambia a diferencia del anterior es que el sistema operativo no necesita que esten todas las páginas subidas a memoria. Se cargan algunas y quedan esperando las otras en una memoria secundaria.

Cuando tengo subida una página en específico a la memoria y quiero hacer una bifurcación hacia la siguiente página que no está en memoria, se acude a un evento que se llama **page fault (fallo de página)**. Este es un evento habitual y no necesariamente es un error. Lo que hace es disparar una **interrupción** (evento que ocurre en la computadora para que el cpu deje de hacer lo que está haciendo y resuelva una solicitud) por hardware. En este caso alerta de que se suba la página no subida en memoria. El cpu usa el mmu y a través de una rutina del sistema operativo que a través de tablas internas da cuenta de la página que tiene que subir, levanta la página que desea, cambiando el frame en el que está. Dicha página esta en la memoria virtual / memoria secundaria (disco rígido). Esta pequeña porción de disco rígido simula ser una extensión de la ram, solamente para guardar las páginas siguientes a la página cargada en la memoria principal. Este proceso implica hacer la traducción de dirección lógica a dirección física.

Si se llega a subir muchos procesos a la memoria principal puede ocurrir que no haya más frames libres en la memoria. Para resolver un pagefault ante esta situación el sistema operativo realiza un **page swapping** consistente en bajar páginas a memoria secundaria y reemplazarlas.

Cuando sucede que se ralentiza una pc/celular muy frecuentemente es porque se están cargando más procesos de los que soporta la memoria de dicho equipo y comienza a ser muy frecuente el swapping.

Puede llegar un momento que el cpu se use más para hacer swappings que las instrucciones en sí de máquina para ser ejecutadas. A este proceso se le denomina **Trashing**

Administración de memoria paginada por demanda (memoria virtual)

- Sistema con multiprogramación
- Solo se cargan a memoria principal las páginas necesarias para la ejecución de un proceso
- Cuando se quiere acceder a una posición de memoria de una página no cargada se produce un *page fault*
- El *page fault* dispara una interrupción por hardware (MMU) atendida por el sistema operativo
- El sistema operativo (*page fault handler*) levanta la página solicitada desde memoria secundaria (memoria virtual)
- Si no hay frames libres es necesario bajar páginas a memoria secundaria y reemplazarlas (*page swapping*)
- Algoritmos para reemplazo de páginas (por ejemplo FIFO, First In First Out o LRU, Least Recently Used)
- Thrashing: el CPU pasa más tiempo reemplazando páginas que ejecutando instrucciones

Administración de memoria paginada por demanda

- Ventajas:
 - No es necesario cargar todas las páginas de un proceso a la vez
 - Maximiza el uso de la memoria al permitir cargar más procesos a la vez
 - Un proceso puede ocupar más memoria de la efectivamente instalada en el computador
- Desventajas:
 - Mayor complejidad por la necesidad de implementar el reemplazo de páginas
- Ej. Windows 3.x en adelante, Linux

Administración de memoria por segmentación

Esto se usaba antes en Intel pero por retrocompatibilidad podría implementarse una arquitectura con esta lógica. Es similar a la paginación dado que el espacio de direcciones de los procesos se divide en segmentos al igual que la memoria ram. No obstante, la gestión es diferente.

En la segmentación uno como programador podía intervenir como se dividía los segmentos, que tamaños, que privilegios, etc. Hoy en día quedó en desuso ya que la paginación es más eficiente.

Los segmentos en general son mucho más grandes que las páginas y se podían definir segmentos de tamaño variable.

El sistema operativo también administraba a partir de una tabla el mapeo.

● Administración de memoria por segmentación

- Sistemas con multiprogramación
- Generalmente visible al programador
- La memoria del programa se ve como un conjunto de segmentos (múltiples espacios de direcciones)
- Los segmentos son de tamaño variable y dinámico
- El sistema operativo administra una tabla de segmentos por proceso
- Permite separar datos e instrucciones
- Permite dar privilegios y protección de memoria como por ej. lectura, escritura, ejecución. (segmentation faults como mecanismos de excepción de hardware para accesos indebidos)
- Las referencias a memoria se forman con un número de segmento y un offset dentro de él. Con ayuda de hardware (MMU – Memory Management Unit) se hacen las traducciones de las direcciones lógicas a físicas
- Se pueden usar para implementar memoria virtual (solo se suben a memoria física algunos segmentos por proceso)

● Administración de memoria por segmentación

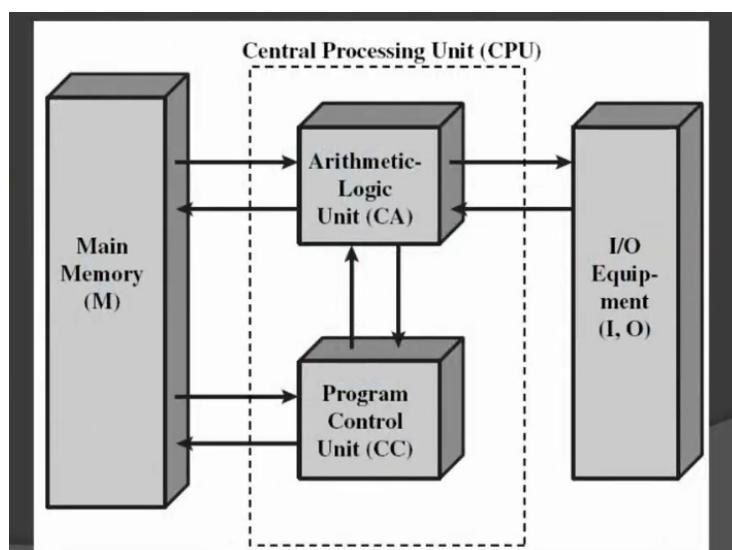
- Ventajas:
 - Simplifica el manejo de estructuras de datos con crecimiento
 - Permite compartir información entre procesos dentro de un segmento
 - Permite aplicar protección/privilegios sobre un segmento fácilmente
- Desventajas:
 - Fragmentación externa en la memoria principal por no poder alojar un segmento
 - Hardware más complejo que memoria paginada para la traducción de direcciones
- Ej. Burroughs Corporation B5000-B6500, IBM AS/400, Intel x86 (por compatibilidad hacia atrás)

○ Administración de memoria

- 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

Entrada / Salida

Siguiendo el esquema básico de Von Neumann, sabemos que los tres elementos principales en una computadora son la memoria, el cpu y la entrada/salida.

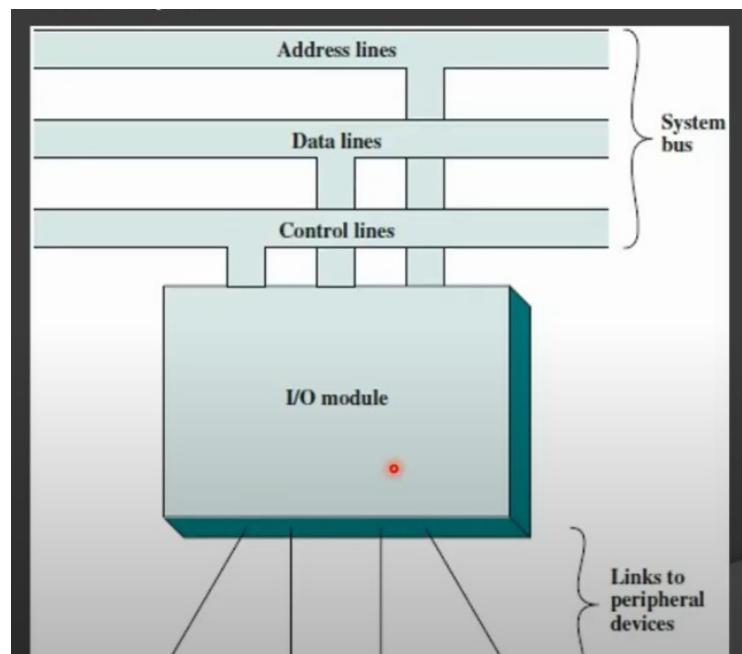


El modulo de entrada y salida permite la conexión de la cpu con los periféricos. Ya sea un teclado, un mouse, un touchpad o bien un disco rígido dentro de la misma carcasa. La razón de que este modulo funcione como intermediario ya que cada periférico tiene una forma de operar, una taza de transferencia distinta y formatos / tamaño de palabras distintos. Permite al cpu no tener que lidiar con toda esta complejidad.

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

El modulo de entrada / salida se conecta a partir de un bus con 3 carriles con información de direcciones, de datos y de control respectivamente. La interfaz en sí del modulo de entrada y salida luego se conecta con los distintos periféricos. La proporción entre cantidad de modulos y periféricos es variable. El modulo también se encarga de interpretar los estados de cada periférico para hacerse saber a la pc, a partir de un medio físico ya sea un cable usb o wifi, por ejemplo.



Se pueden diferenciar una interface interna y externa. Esta última refiere al enchufe en sí, mientras que la interface interna refiere a que cosas se transfieren por dentro del sistema en sí.

Funciones del modulo

En primera instancia este se encarga de controlar el flujo de información entre el CPU/memoria y los periféricos. Esto es algo que sucede de forma constante. Hay una gestión de la información de entrada para obtener una salida.

Las dos grandes funciones del modulo, por un lado es el intercambio de información con el cpu. La entrada/salida en primera instancia tiene la tarea de decodificar los comandos que le envía el CPU para luego mandarlos al periféricos. La decodificación en sí de comandos entonces es un comando genérico para traducir una solicitud a los periféricos.

Luego, obviamente se tiene que transmitir datos, por ejemplo cuando se envían los bytes que representan una imagen a una impresora.

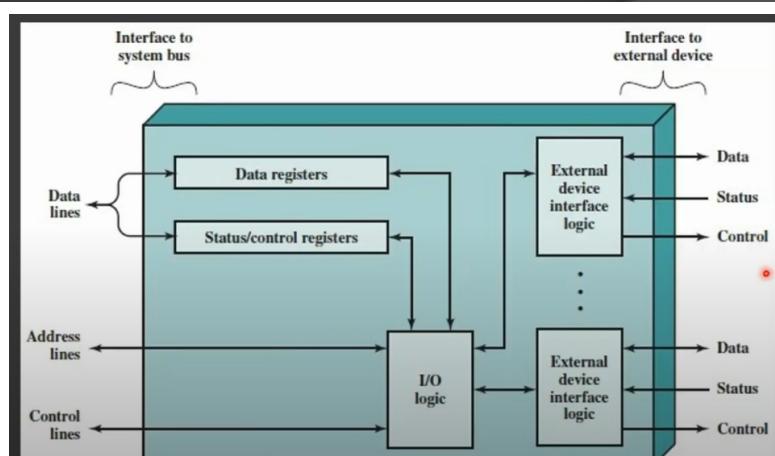
Todos los periféricos conectados a una computadora tienen un ID único lo que permite hacer un reconocimiento de direcciones al módulo, para decirle al cpu que tiene conectado y para que el mismo sepa cuando quiere hacer algo a que dispositivo con que ID mandarlo.

Luego tenemos la comunicación en sí del modulo con el dispositivo periférico. El modulo le puede decir que grabe datos en el disco, le envía comandos, información de estado, datos etc.

El buffering de datos refiere a que como los periféricos tienen distintas velocidades, ocurre dentro del modulo que tiene algunos registros para guardar información momentaneamente para después en el momento adecuado, transmitirlos.

La detección de errores viene enlazado a la detección y envío de información de Estado.

- **Funciones**
 - Control & Timing
 - Controla flujo de tráfico entre CPU/Memoria y periféricos
 - Comunicación con el procesador
 - Decodificación de comandos
 - Datos
 - Información de estado
 - Reconocimiento de direcciones
 - Comunicación con el dispositivo
 - Comandos
 - Información de estado
 - Datos
 - Buffering de datos
 - Detección de errores



Modos de operación de una entrada/salida

○ Módulo de E/S

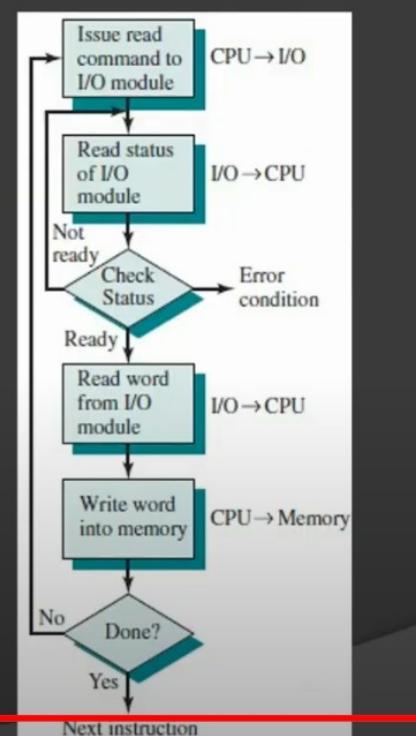
- Técnicas para operaciones de E/S
 - E/S Programada
 - E/S manejada por interrupciones
 - Acceso directo a memoria (DMA)

La entrada/salida programada se puede esquematizar como un envío de un programa de un comando genérico de lectura hacia un periférico determinado, que recibe el módulo de entrada/salida. Esto viaja a partir del bus de sistema. De forma secuencial, lo siguiente que ocurre es leer el Estado que devuelve al CPU el módulo de entrada/salida. Esta respuesta puede ser un rechazo del comando, que puede ser interpretado como un error o como la habilitación de un estado de espera que se desataba por el error en sí o el cambio a un estado de “listo”, que surge a partir de que el periférico tome la solicitud y devuelva una respuesta en forma de bytes al módulo de entrada/salida, el cual tendrá que ubicar en alguna dirección de memoria a partir de un buffer de lectura. Cuando esto se resuelve se pasa a la siguiente instrucción.

Lo que no es eficiente es que el CPU se quede esperando a que la operación de entrada/salida ocurra. Esto es un desperdicio enorme de potencial que tiene el mismo tal que en el tiempo que tarda una respuesta de periférico se pueden ejecutar cientos de miles de instrucciones.

○ Módulo de E/S

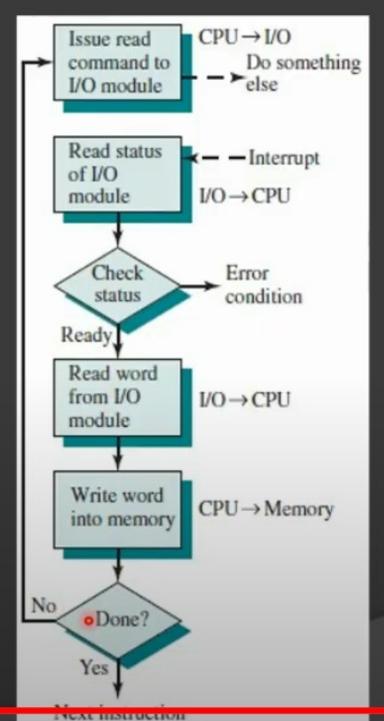
- E/S Programada



Lo que aparece para resolver esto último es el concepto de interrupción, a partir de la **E/S manejada por interrupciones**. Acá lo que cambia es que entre la acción de envío del comando al módulo se deja de esperar una respuesta y al CPU se le asigna otro proceso. Cuando se obtiene una respuesta el CPU la recibe a partir de las interrupciones, que permite hacer que el CPU deje de hacer lo que esté haciendo y evaluar qué hacer con ella. Cada instrucción de máquina que se ejecuta dedica un tiempo a ver si hay alguna instrucción de interrupción vigente. Cuando vuelve a leer la interrupción el CPU sucede lo mismo de chequear el estado para tirar un error o no. Cuando se leen los datos enviados por el periférico se escribe en memoria y cuando termina se pasa a la siguiente instrucción.

○ Módulo de E/S

- E/S manejada por interrupciones

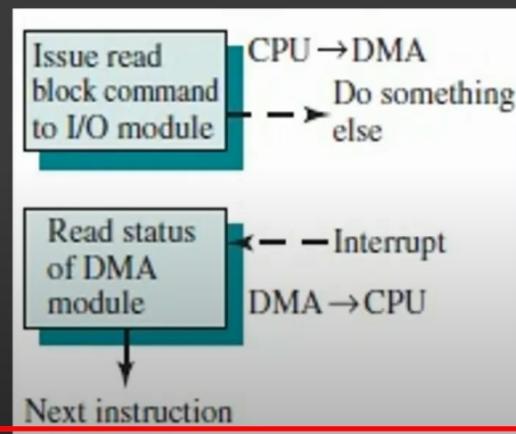


La inefficiencia acá es escribir la información que deja el módulo de entrada/salida en el CPU para que se escriba en la memoria principal. Hay que tener en cuenta que el CPU es un recurso escaso con una velocidad mucho mayor a cualquier otro componente. Cualquier cosa que el CPU haga que no sea manejo de instrucciones es un uso inefficiente.

Es por esto último que existen manejos más sofisticados para que el módulo tenga un acceso directo a la memoria para resolver lo de la entrada/salida sin tener que ocupar el CPU:

○ Módulo de E/S

- Acceso directo a memoria (DMA)

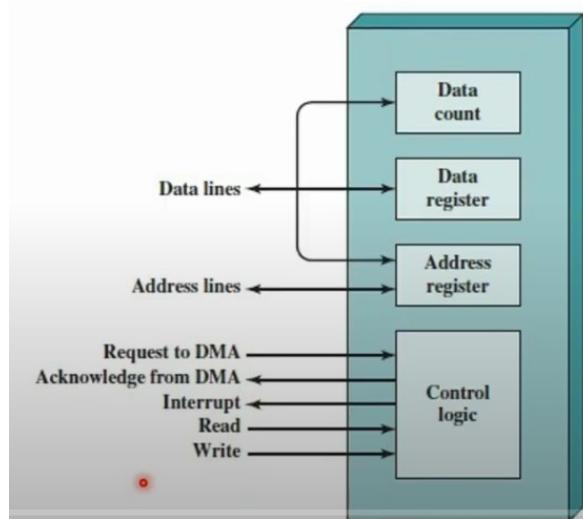


Acá el diagrama de bloque cambia rotundamente, de modo que el DMA (Direct Memory Access) es otro componente que se encarga de resolver de que la info que se lee, se escriba en la memoria. Y acá aparece el dilema de tener un componente adicional que puede interactuar con la memoria. Hasta este momento el único que interactuaba con la misma es el CPU.

Con la presencia del DMA, es posible leer bloques enteros de datos a leer y no palabra por palabra. Tiene la inteligencia para capturar el estado del entrada/salida, ver si está listo, hacer la escritura de la información a la memoria para finalmente emitir la interrupción al CPU.

Interacción del DMA con el CPU

Para desentenderse del tipo de acciones ligadas a la escritura en memoria, la relación que tiene el DMA con el CPU conlleva que el CPU envíe al DMA el tipo de operación (**Read o Write**) que se enviará al dispositivo conectado vía Línea de Control (Control Logic):



Luego se debe indicar al DMA con el cual se quiere interactuar, y esto lo hace a partir de su ID. Esto se hace vía Línea de Dirección (Address lines). Luego el CPU debe decirle al DMA con qué dirección en memoria va a interactuar, ya sea para leer los datos e enviarlos al dispositivo (**Read**) o escribir datos recibidos del dispositivo en memoria (**Write**). Esto era algo que resolvía el CPU y no se enviaba al modulo de entrada / salida, pero si es algo que se envía al DMA. Esta dirección inicial de memoria para Read O Write se envía vía Línea de Datos (Data lines) y lo guarda en Data register.

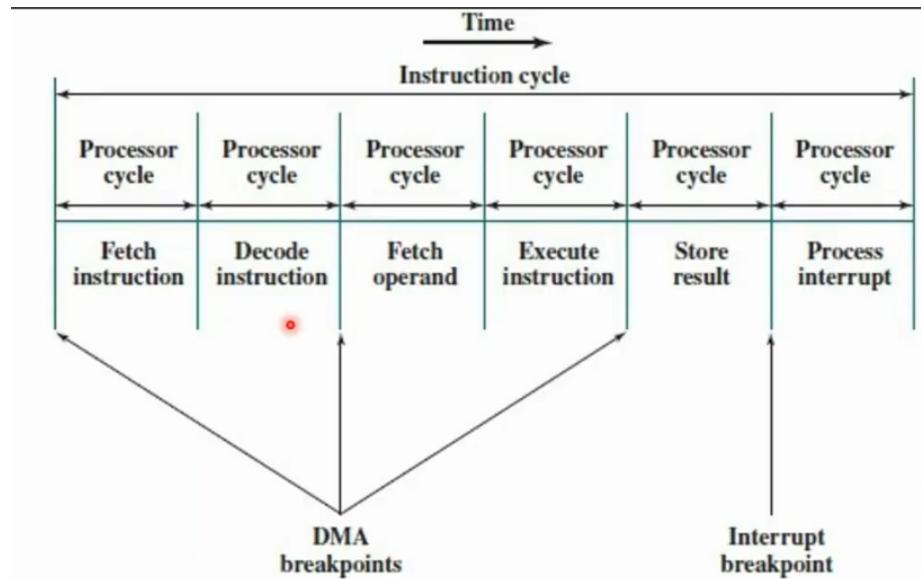
Por último, el CPU debe decirle cuantas palabras para READ o WRITE se utilizarán a partir de la dirección inicial y vía Línea de Datos se almacenará en el data count register.

El componente Control Logic es el componente que decodifica el comando y manda información tanto al modulo de entrada y salida para que este se lo envíe al periférico.

El DMA es el que tiene la capacidad de enviar la señal de Interrupt al CPU.

DMA – “Robo de ciclos”

Al tener un nuevo actor que accede a la memoria, cuando antes era únicamente el CPU el que podía hacerlo es necesario arbitrar cuando accede el DMA y cuando accede el CPU a dicha memoria. El robo de ciclo consiste en lapsos dentro de un ciclo de instrucción en los que el DMA le dice al CPU que va a acceder a la memoria y este tiene que esperar a que la acción del DMA finalice:



Hay tres momentos en específico en los que el DMA pueda acceder a la memoria, que son momentos claves en los que se necesitan ir a buscar algo a la memoria. Potencialmente en cada breakpoint el CPU requiera datos de la memoria. Cuando se hace el fetch de la instrucción, potencialmente haya que ir a buscar donde esta se encuentra alojada para luego traerla al registro de instrucción (RI) dentro del CPU.

Dentro de lo que es la ejecución de la instrucción en primera instancia tenemos la decodificación de la misma (interpretación de cada uno de los bits dentro del registro de instrucción para entender que es lo que se le pide hacer. Código de operación, operandos, etc). Antes del fetch del operando se abre un DMA breakpoint dada la potencialidad de que el operando se vaya a buscar a la memoria.

Finalmente, luego de que se haya ejecutado la instrucción tengo un breakpoint en el store del resultado, para por ejemplo guardar la suma de dos registros en memoria.

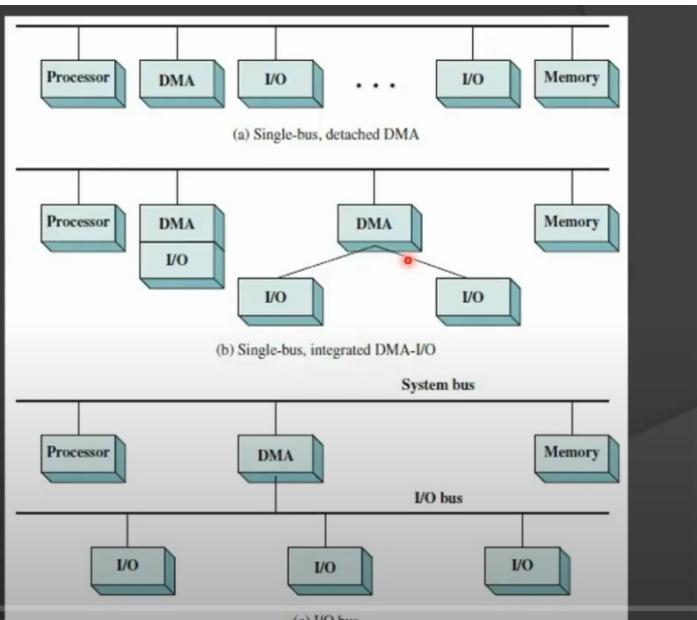
Luego lo que es el Interrupt breakpoint, es el momento en específico en el que el cpu se pone a escuchar si hay algún evento que lo quiera interrumpir. La interrupción se lee solo cuando se termino de ejecutar una instrucción. Si se intenta hacer una interrupción en el medio de la ejecución de una instrucción, esta interrupción queda “encolada” hasta que esta termine de ejecutarse.

Los esquemas de como los componentes se interconectan responden a distintas topologías de configuración, donde se entienden las líneas como los bus que interconectan a cada componente:

○ Módulo de E/S

- DMA –

Topologías de configuración



Canales y procesadores de Entrada y Salida

Ante grandes demandas de este tipo de interacciones como en servidores, aparecen los conceptos de canales y procesadores de entrada y salida. El concepto es como el del modulo de entrada y salida pero tienen una complejidad tecnológica mayor, dando más versatilidad.

Un canal tiene una CPU propia, de modo que ejecuta las instrucciones de entrada/salida sin ningún tipo de intervención del CPU principal de la máquina.

La diferencia entre un canal y un procesador es que este CPU secundario del canal sigue interactuando con la memoria principal. Tiene que leer las instrucciones guardadas allí. En el caso del procesador, estos tienen una memoria ram propia, donde se ejecutan las operaciones de entrada y salida, el propio cpu del procesador, yendo a buscar dentro de su propia memoria:

○ Módulo de E/S

- Canales y procesadores de E/S

- Canales

- Tienen la habilidad de ejecutar instrucciones de E/S
 - La CPU principal no ejecuta instrucciones de E/S
 - Las instrucciones de E/S se almacenan en memoria principal
 - La CPU le indica al canal de E/S que inicie un programa de canal
 - Dispositivo
 - Área de memoria para storage
 - Prioridad
 - Acciones ante errores

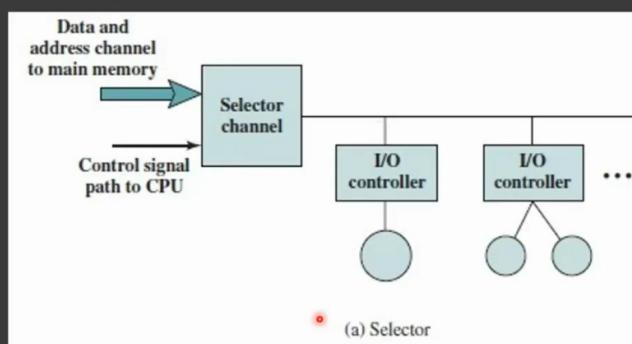
- Procesadores

- Agregan a los canales memoria propia en vez de usar la memoria principal

En caso de los canales, el CPU principal lo único que hace es indicar que se debe ejecutar una **operación de canal**, que indica información relevante para que este canal lo procese. Esto es el dispositivo, el área de memoria para storage, la prioridad (en contextos en el que haya miles de operaciones de entrada y salida al tiempo) y acciones ante errores.

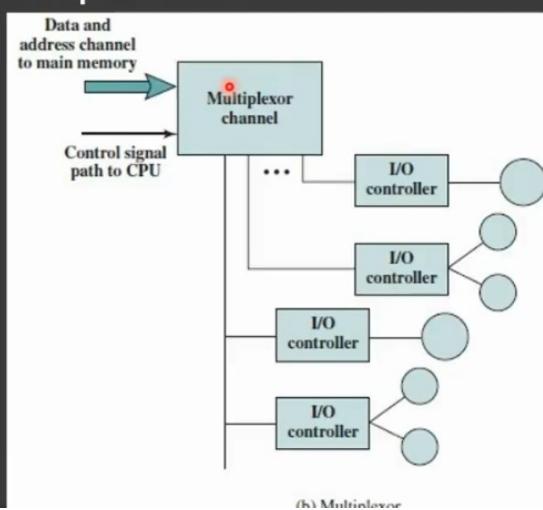
Los canales pueden operar de dos formas. La forma más simple es el canal selector, que es aquel que usa un dispositivo a la vez para operar. Se conecta con los distintos módulos de entrada y salida y puede operar con uno solo a la vez:

- Módulo de E/S
 - Canales y procesadores de E/S
 - Canales selectores



Luego están los multiplexores son aquellos que pueden trabajar con múltiples pedidos a la vez. La conectividad es en paralelo, de modo que puede recibir peticiones de todos los controladores a la vez:

- Módulo de E/S
 - Canales y procesadores de E/S
 - Canales multiplexores



Interrupciones

Son mecanismos mediante los cuales otros módulos pueden interrumpir el normal procesamiento del CPU. Los eventos en paralelo que suceden tienen que congeniar con el habitual procesamiento de instrucciones del CPU. Es mediante las interrupciones que este es capaz de registrar eventos de cualquier tipo y que dichos eventos sucedan al tiempo de los procesos que ya de por sí se están ejecutando.

● Interrupciones

- ¿Qué son?

“Mecanismos por los cuales otros módulos (E/S, memoria, etc.) interrumpen el normal procesamiento del CPU”

- ¿Para qué existen?

“Para mejorar la eficiencia de procesamiento de un computador”

- Clases de interrupciones

- Hardware
- Software

Es por esto que las interrupciones existen para mejorar la eficiencia del procesamiento de un computador.

En general hay dos tipos de interrupciones. Las más generales son aquellas que son producidas por algún elemento del hardware en particular. El otro tipo de interrupciones son las de software, aquellas que se producen porque algo ocurrió a nivel programación dentro de algún proceso que se está ejecutando

Las de hardware se suele enlazar con el concepto de asincronías tal que son eventos que no están alineados con el proceso que está corriendo, por lo contrario se hace de forma no sincronizada.

Otros componentes además de los componentes de entrada/salida que pueden generar las interrupciones son el reloj: Este se encuentra dedicado a cronometrar los ciclos de instrucción y dedicarle un tiempo a cada proceso. Por último tenemos las fallas de hardware, cuando algún periférico falla.

Luego para las interrupciones de Software se pueden identificar excepciones de programa: eventos que en general son sincrónicos, ya que ocurren dentro del proceso que el cpu atiende. Cuando de repente sucede algo que hace que se corte la normal ejecución del programa:

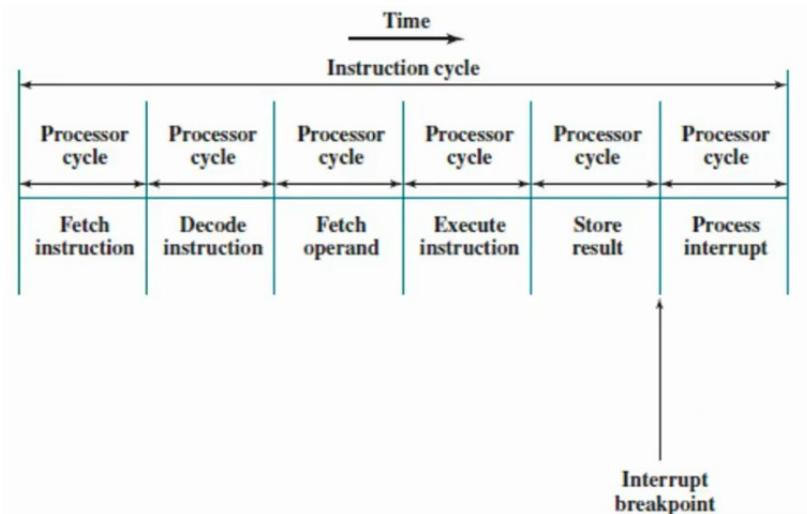
○ Interrupciones

- Clases de interrupciones
 - Hardware (asincrónicas)
 - E/S
 - Reloj (timer)
 - Fallas de hardware
 - Software
 - Excepciones de programa
 - División por cero
 - Acceso indebido a memoria
 - Overflow
 - Instrucción inválida
 - Instrucciones privilegiadas

Las instrucciones privilegiadas son aquellas excepciones generadas por el propio código del programa (como el swi en ARM para acceder a un servicio del sistema operativo).

Interrupciones dentro del ciclo de instrucción

Dentro del propio ciclo de instrucción, el momento en el cual se permite leer interrupciones es tras ejecutar la instrucción máquina completa (tras haber guardado el resultado). Luego entra el procesamiento de dicha interrupción. Estas interrupciones suceden millones de veces por segundo.



El comportamiento asincrónico de una interrupción puede verse de la siguiente manera:

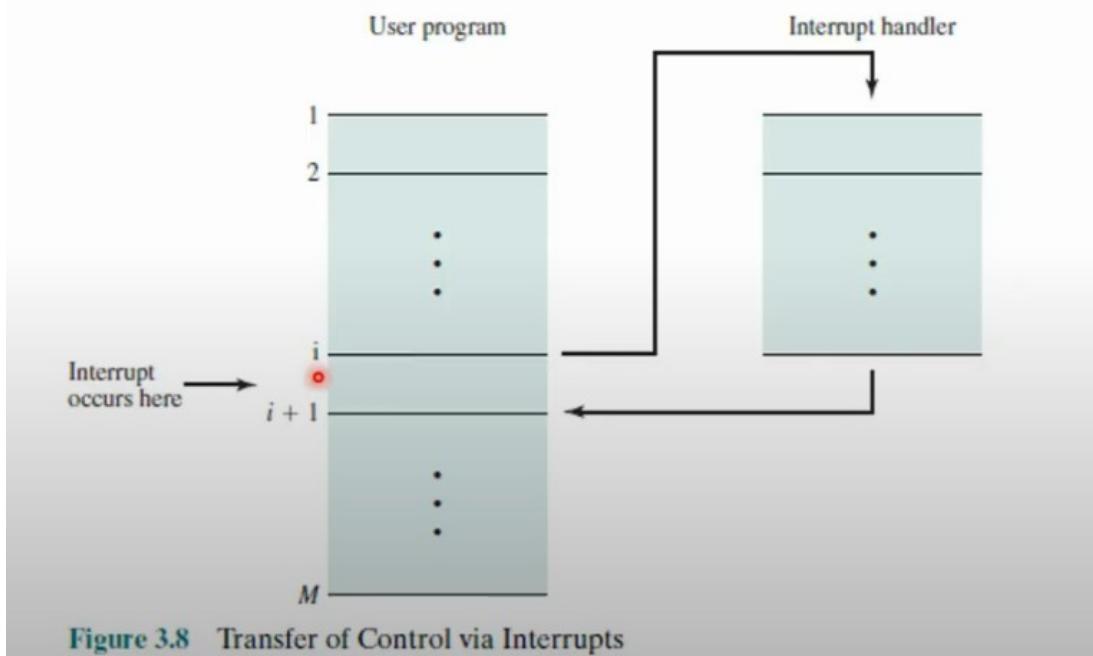


Figure 3.8 Transfer of Control via Interrupts

Si en el medio de la interrupción-ísema ocurre que un hardware lanza otro interrupción, esta se ejecutará entre la isema y la isema + 1. Atender una interrupción significa que algún servicio del sistema operativo escuchará dicha instrucción y hará algo.

Los handler son rutinas de software encargadas de atender las posibles interrupciones que pueden ocurrir de cada hardware que sea capas de lanzar una interrupción.

Primeramente en una interrupción, el elemento del hardware genera la interrupción. El CPU finaliza la ejecución de la instrucción actual para escuchar y confirma al modulo de entrada/salida que esa interrupción será atendida.

El CPU cuando recibe la interrupción resguarda la PSW (Program Status Word) que es un área de memoria donde se guarda para cada proceso información de control (estado de algunos bits, etc). Luego también resguarda el Program Counter, es decir, el valor de la próxima instrucción que se debería haber ejecutado del proceso actual. Estos datos tiene que guardarlo en una pila de control ya que se van a pisar con los que van a obtener de la interrupción.

Finalmente para efectivizar la interrupción se carga en el program counter, en el rpi, el valor de la rutina que va a entrar a partir de la interrupción:

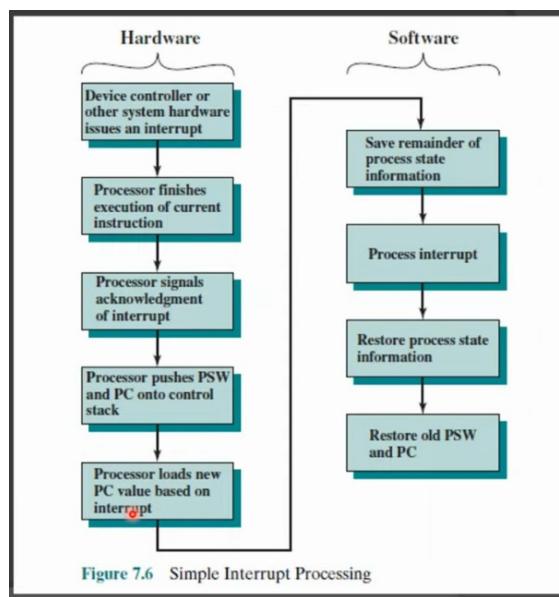
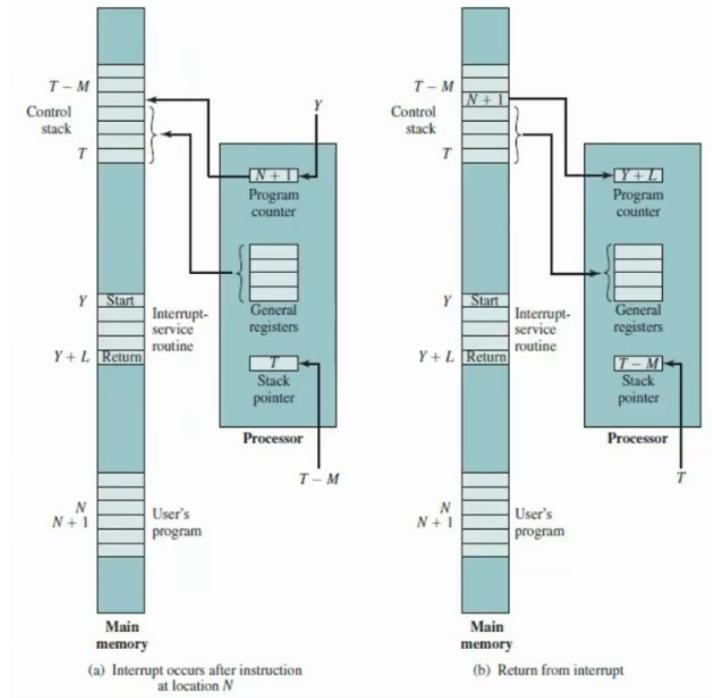


Figure 7.6 Simple Interrupt Processing

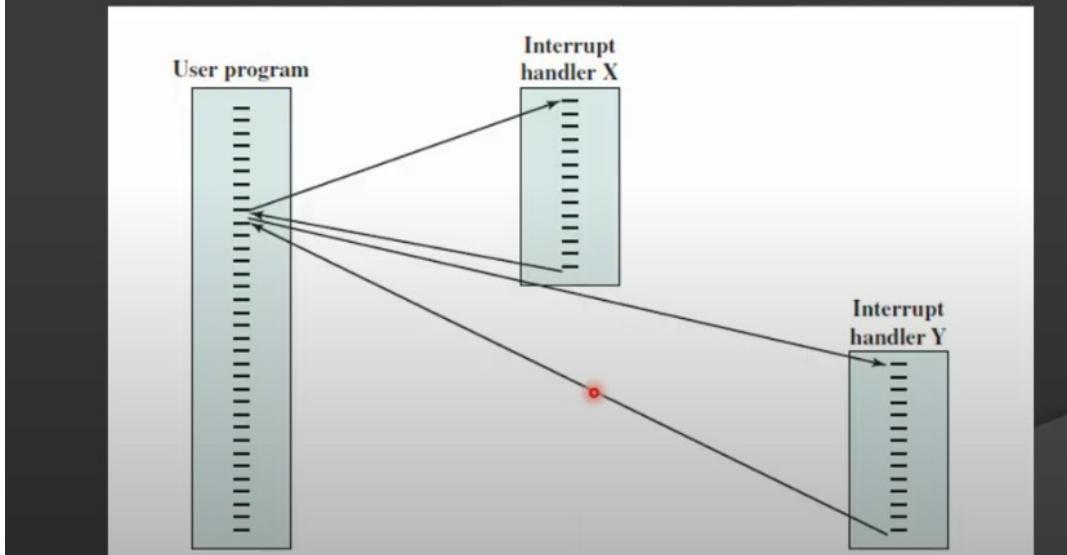
La rutina de software que atiende la interrupción lo primero que hace es resguardar el estado del proceso anterior en memoria interna tal que la interrupción potencialmente va a utilizar los registros que estaba usando este. Luego se ejecuta la interrupción, se restauran los valores anteriores de los registros y finalmente se busca el anterior PSW y PC de la pila de control.



Multiples interrupciones

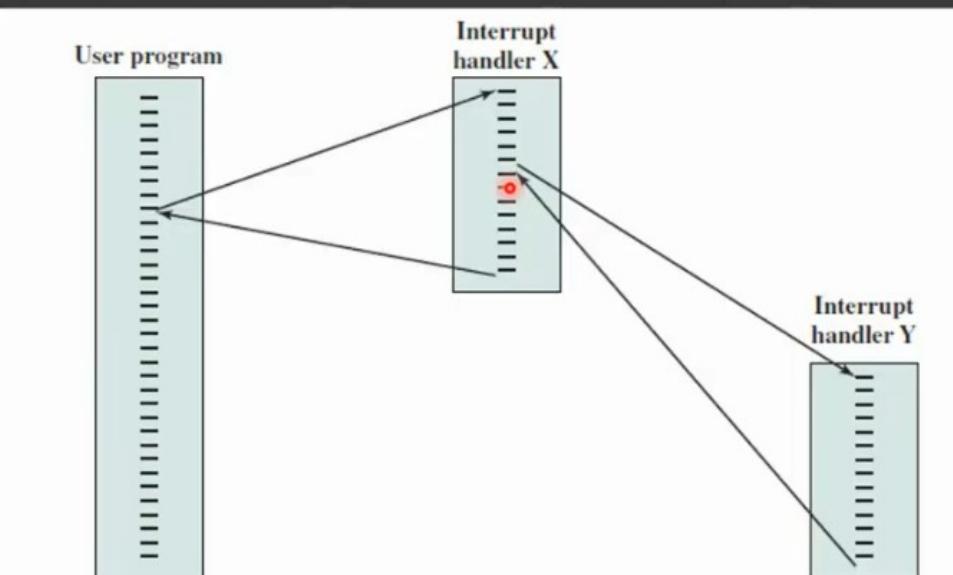
En la vida real suceden multiples interrupciones al mismo tiempo. Para atender eficientemente cada interrupción. Una opción es hacerlo de forma secuencial, pero esto trae algunos problemas en el sentido que puede haber interrupciones con más prioridad que se posterguen mucho por una cuestión de que llegaron antes interrupciones de menor prioridad.

○ Deshabilitar interrupciones (secuencia)



Lo que habitualmente ocurre con el hardware es que las interrupciones se manejan por prioridad, es decir, de forma anidada:

○ Priorizar interrupciones (anidadas)



(b) Nested interrupt processing

Figure 3.13 Transfer of Control with Multiple Interrupts

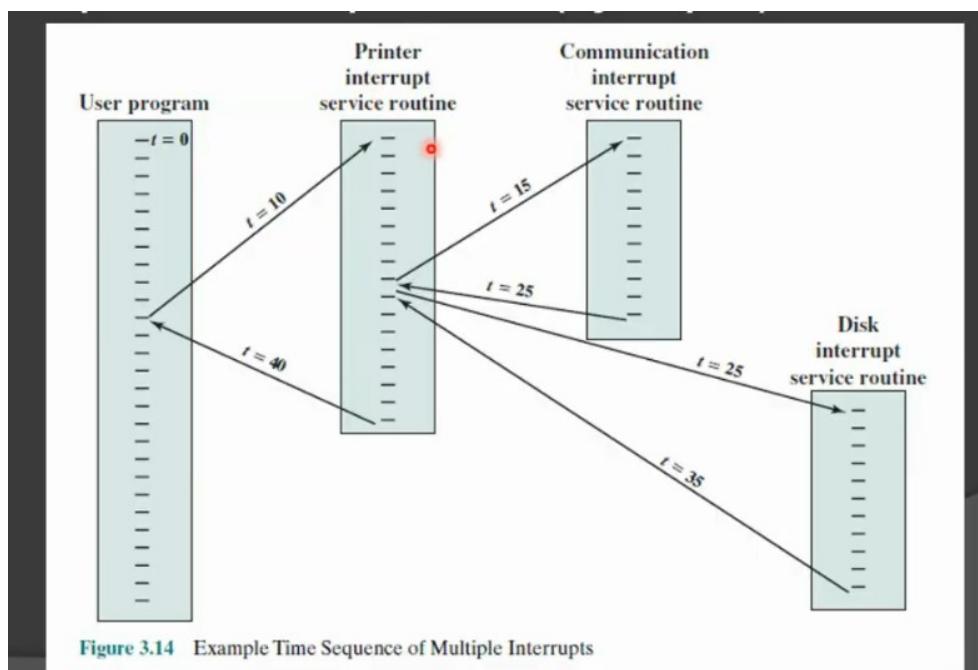
Múltiples interrupciones (ejemplo)

- Tres dispositivos de E/S

- Línea de comunicación (Prioridad 1)
- Disco (Prioridad 2)
- Impresora (Prioridad 3)

- Eventos

- T=10 – Interrupción de Impresora
- T=15 – Interrupción de línea de comunicación
- T=20 – Interrupción de disco



Procesador (CPU)

Se pueden diferenciar 2 tipos de arquitecturas principalmente: **CISC (Complex Instruction Set Computer)** y **RISC (Reduced Instruction Set Computer)**.

El CISC son los que predenominaron hasta fine los 70'.

El RISC es un diseño de arquitectura de computadores más minimizado, simple o elemental. Aparece a principios de los 80'. Todo el diseño de hoy en dia tiende a ser de este tipo. Esta arquitectura está orientada a Software.

A principios de los 70' no estaba comercializada tanto, ni había computación hogareña. El software se programaba para cada arquitectura en específico, hasta principio de los 70'.

En el diseño de procesadores se buscaba que estos tengan una arquitectura robusta que pueda resolver muchas cosas tal que no había tanto software desarrollado aún. Las instrucciones de máquina ocupaban muchos ciclos de reloj, se hacía mucho uso de la memoria, etc.

A partir de la evolución de la industria del software y la masificación del uso de la computación empieza a haber en el mercado la tendencia del RISC, de modo que el hardware se va simplificando más y más tal que la industria del software más avanzada venía a resolver un montón de las complejidades que antes necesitaba resolver el hardware. Los lenguajes de programación son cada vez más difundidos y resuelven más problemas, de tal manera que se advierte que es más performante una arquitectura minimalista. No obstante aún hay en el mercado arquitecturas como Intel x86 que por su retrocompatibilidad mantiene la arquitectura CISC.

CISC

No tiene muchos registros accesibles para un programador y por lo general son especializados. El conjunto de instrucciones es muy amplio. Tiene muchas instrucciones que trabajan directamente con memoria.

Lo que es la microarquitectura compleja muestra múltiples caminos de datos. Las instrucciones son complejas y pueden tardar más de un ciclo de reloj. Posee varios tipos de direccionamiento, datos y formatos de instrucción. Está orientado al hardware de modo que es el hardware el que principalmente resuelve las instrucciones.

RISC

Tiene muchos registros de uso general e intercambiables, a diferencia de CISC. A nivel de circuito no tiene ninguna restricción si quizás a nivel programación. Posee un set de instrucciones pequeño y se accede a la memoria solo a través de LOAD/STORE. La microarquitectura en hardware es simple porque las instrucciones son simples. Esto último indica que dichas instrucciones no durarán más de un ciclo de reloj. Posee pocos modos de direccionamiento, pocos tipos de datos y pocos formatos de instrucción. El formato de instrucción es fijo lo que es clave para la performance. La búsqueda de las instrucciones es más simple gracias a esto, el tamaño será siempre el mismo.

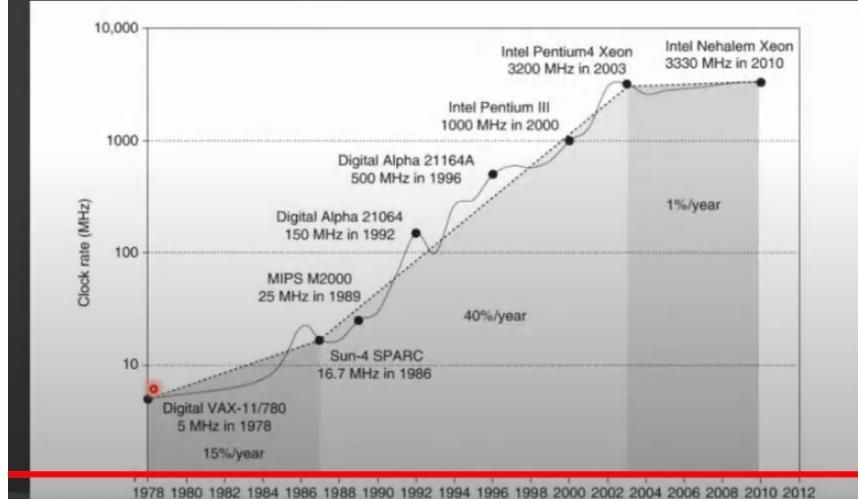
Está orientada al software y el hardware es mínimo. Aprovecha los lenguajes de programación cada vez más eficientes y complejos.

Ecuación de performance

MIPS = Frecuencia del reloj en Mhz (f) * Instrucciones por Ciclo (IPC) → Esto me da un ratio de performance. Esto calcula cuantos millones de instrucciones puede ejecutar el cpu por segundo.

Los primeros CPUS de un solo núcleo tenían una frecuencia de reloj baja y fue subiendo con el tiempo:

Velocidad de reloj (uniprocesadores)



En los últimos años el aumento de la frecuencia con los nuevos procesadores es cada vez más lento tal que el aumentar la frecuencia de un procesador implica una generación de calor que efectúa como limitante.

Ante estas limitaciones para maximizar el uso que se le puede dar al procesador nace el concepto de **parallelismo**. No se le exige al cpu que ejecute más instrucciones por frecuencia, si no que se lleva al CPU que ejecute más instrucciones por ciclo de reloj a partir de técnicas de procesamiento en paralelo:

Parallelismo

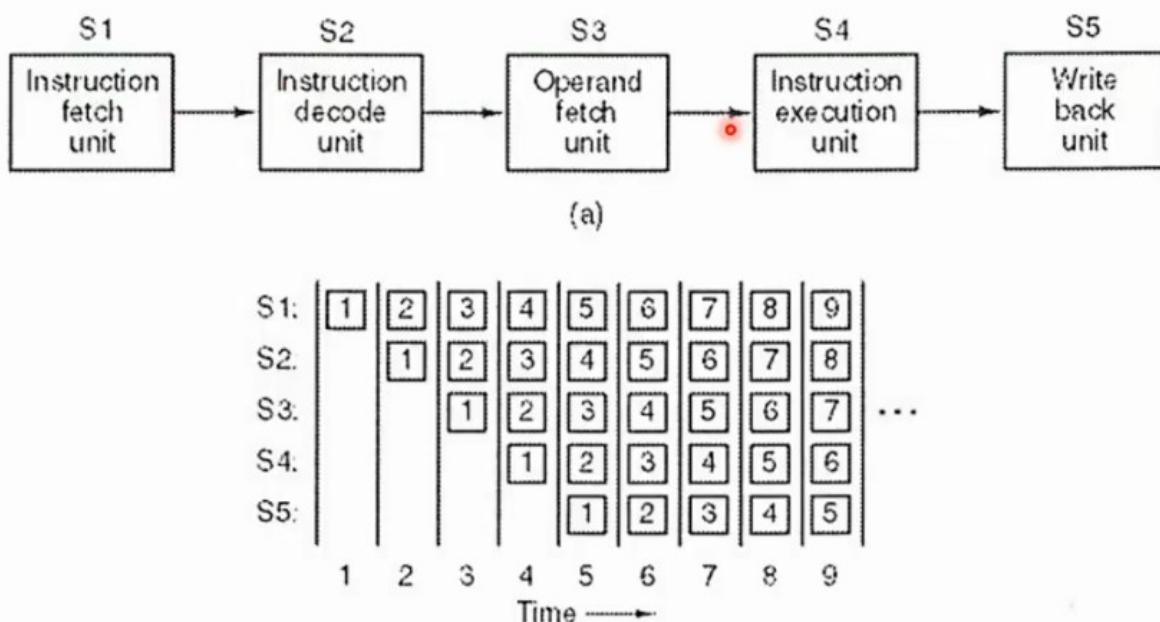
- Técnicas
 - A nivel instrucción
 - Pipelining
 - Dual pipelining
 - Superscalar
 - Multithreading
 - A nivel procesador
 - Procesadores paralelos de datos
 - Multiprocesadores
 - Multicomputadoras

Las primeras parallelizan procesos en dentro de un solo core. Luego las técnicas a nivel procesador se busca que el procesamiento se haga en varios procesadores al tiempo, obteniendo rendimientos altísimos con los multiprocesadores y/o multicomputadoras.

Técnicas a nivel instrucción

Pipelining

A partir de un CPU, el mismo trata de ejecutar en paralelo más de una instrucción a partir del solapamiento de ejecuciones de instrucción, dividiendo a los ciclos de instrucción en etapas:



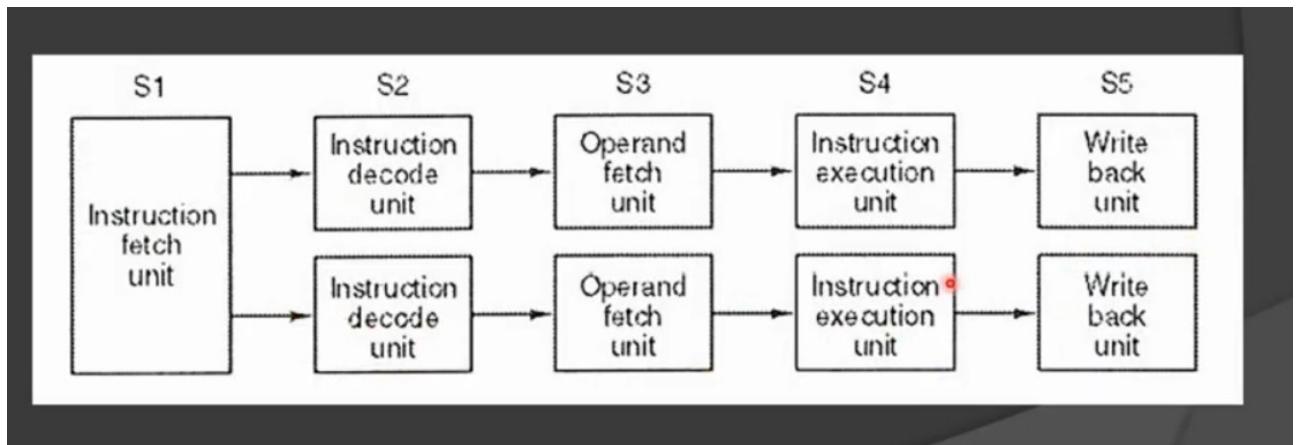
A medida que pasan las instrucciones a las siguientes unidades de máquina, otras se van procesando en las anteriores. Por ciclo de reloj sigue ejecutando una a la vez.

Uno de los problemas de ir ejecutando instrucciones de esta manera, teniendo en distintas secciones del cpu a cada instrucción, puede ser si alguna de estas instrucciones es una bifurcación, tal que dicha bifurcación me llevará a otras instrucciones y no se ejecutarían en el orden en el que estaban encoladas las demás instrucciones.

También puede suceder que una instrucción modifique algún operando de una instrucción por detrás procesándose. Para evitar esto se debe hacer un **control de dependencia** entre las instrucciones, ya sea a nivel de hardware o a nivel de software (compilador / hardware). Para solucionar esto dicho control de dependencia lo que hace para dos instrucciones donde se comparta un mismo operando es dejar un “tiempo muerto” (Not Operation) para que la instrucción termine de ejecutarse y de esta forma la que le sigue antes obtiene el valor esperado.

Dual pipelining

A partir del diseño del pipeline, algunos fabricantes implementan el Dual Pipelining. Acá si hay una genuina ejecución en paralelo de dos instrucciones.



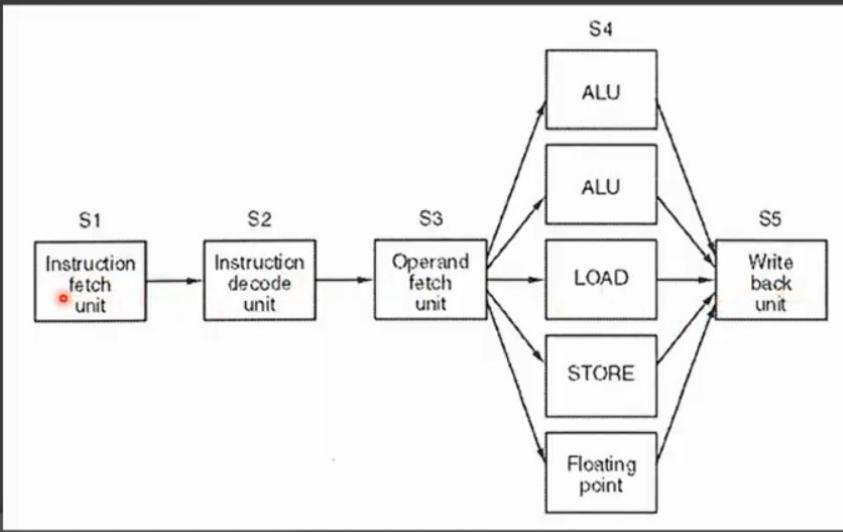
El acceso a la memoria es único pero el resto se paralleliza, ejecutando dos instrucciones por ciclo de reloj. Esto implica lugar físico del CPU, un costo de fabricación y generación de calor, no se escala más la cantidad de Pipelines por estos limitantes. Acá se requiere capacidad de ejecución de dos instrucciones al tiempo.

Superscalar (Múltiples unidades funcionales)

Esta se da a nivel instrucción y es complementaria (al igual que todas las otras) a las demás técnicas. Es decir, pueden combinarse entre sí, no son excluyentes.

Esta técnica genera un paralelismo en una de las etapas, la más lenta, se busca paralelizar en múltiples unidades funcionales (múltiples circuitos) para que puedan ejecutar varias instrucciones en paralelo y mejorar la performance:

- Superescalar



Se paraleliza la etapa 4 (la de ejecución) al ser la más lenta. Cuando no se tiene esta técnica hay un “cuello de botella” en esta etapa tal que para seguir procesando instrucciones hay que terminar de ejecutar la que estaba delante y suele tardar mucho en esta etapa.

Estas múltiples unidades funcionales (piezas de hardware) se dedican a ejecutar una acción en particular. Por ejemplo las ALU (Arithmetical Logic Unit) se dedican a hacer operaciones lógicas y matemáticas. Por ende, cuando entra más de una instrucción que se requiera una ALU, ambas se pueden ejecutar al tiempo tal que dispongo de dos. Si tuviera una instrucción luego para hacer un LOAD o STORE, se puede hacer tal que va a esa unidad libre. Suelen tener entre 3 y 6 de estas unidades.

○ Procesador

- Paralelismo

- A nivel instrucción

- Superscalar (múltiples unidades funcionales)
 - Ejecuta más de una instrucción por ciclo de reloj
 - N-way / N-issue (N entre 3 y 6)
 - Ej. Intel Core

Hardware multithreading / multiframe

Acá es cuando aparecen los famosos “hilos” (thread) y “núcleos” (core). Acá cada core paralleliza la ejecución de una instrucción en hilos diferentes. Cuando un hilo se frena por alguna causa, la instrucción se sigue ejecutando sobre otro hilo. De esta forma se maximiza el uso del CPU y este no se queda esperando a que finalice una determinada ejecución.

Un hilo es una línea de ejecución con cierta autonomía y dependencia. Cada hilo tiene un program counter (el valor del RPI). Cada hilo puede dejar ejecutando en una posición del programa distinta, es decir en un RPI distinto. La autonomía la consigue de tener registros de instrucciones propios para no hacer llamados a la memoria y una pila propia para hacer stacking de contextos.

Lo que comparten todos los hilos es el mismo espacio de direcciones. Los procesos se parallelizan pero todos usan la misma memoria. No usan una memoria independiente. Por eso se los conoce como procesos “livianos”.

Un proceso pesado es el proceso en sí tal que este tiene su propio espacio de direcciones (único) y un estado gestionado por el S.O. El cambio de contexto entre procesos es una gestión pesada, el cambio de procesos entre hilos es una gestión liviana:

○ A nivel instrucción

- Hardware multithreading
 - Busca incrementar el uso del CPU intercambiando la ejecución entre threads (hilos de ejecución) cuando uno está frenado por alguna causa
 - Thread: Contiene un PC, un conjunto de registros y la pila (stack). Comparten un mismo address space. Se los conoce como “lightweight processes”
 - Proceso: Puede tener uno o más threads, contiene un address space y un estado gestionado por el S.O.
 - El cambio de contexto entre threads es “liviano” (en un mismo ciclo de reloj) en comparación con los procesos, que requieren del S.O.

El hardware multihilo puede operar bajo varias técnicas.

Fine-Grained multitherading: intercambiando el uso del proceso entre hilos luego de la ejecución de cada instrucción. Es decir hace un barrido de los mismos.

Coarse-grained multithreading: El intercambio de hilos solo ocurre si hay algún evento significativo. Algo que frene la ejecución de un hilo en particular y valga la pena hacer el cambio de contexto. Esto sucede por ejemplo en el page fault (querer acceder a una página en memoria no mapeada, lo que dispara dicho evento y genera una interrupción). Cuando se llega a un proceso como este en un hilo, el procesador cambia a otro hilo para ejecutar otra instrucción de máquina.

El uso de esta técnica también puede verse presente cuando hay un “cache miss”. Cuando el CPU le pide a la caché determinada palabra y no la encuentra, esta memoria cache tiene que ir a pedirla a otro nivel de cache o a la memoria ram. Como esto implica quedarse esperando, se puede resolver que no se quede esperando pasando a otro hilo. Este tipo de técnicas está más orientada a servidores.

Técnicas a nivel procesador

Acá se paraleliza el uso de multiples cores, multiples cpus o multiples computadoras.

Procesadores paralelos de datos

SIMD

Son arquitecturas que trabajan una única instrucción de máquina con multiples procesadores. De ahí viene la sigla SIMD – Single Instruction Multiple Data. Son un subconjunto de instrucciones dentro de la ISA. En estos casos el CPU tiene múltiples core y una única unidad de control (CPU). Los multiples nucleos me permiten paralelizar operaciones.

Hay dos formas de operar, una es esta de tener multiples procesadores que ejecutan la misma secuencia de pasos sobre un conjunto diferente de datos. Esto sirve por ejemplo para hacer la misma operación para distintos operandos, cada core realiza la misma operación pero varia los elementos con los que trabaja. Esto es común en los procesadores dedicados para la gráfica. El calculo gráfico implica hacer muchas operaciones aritmeticas con muchos datos en paralelo.

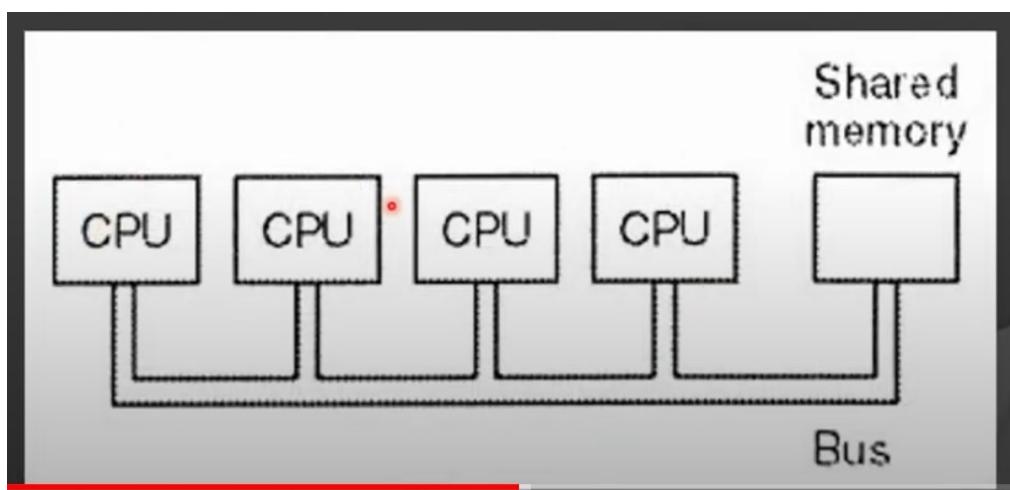
La otra técnica también se tiene una única instrucción para multiples datos, pero se cambia como se implementa en el hardware. Acá hay un registro vectorial: Un conjunto de registros que trabajan como un único bloque y hay una instrucción vectorial para trabajar con ese bloque. Esto opera bajo la técnica SSE (Streaming SIMD Extension).

MIMD

(Multiple Instruction Multiple Data)

Multiprocesadores: Esta técnica utiliza CPUs diferentes, completos y trabajando en paralelo. Estos CPUs están fuertemente acoplados tal que siguen utilizando una memoria compartida.

Dentro de las posibles implementaciones de multiprocesadores se puede tener un único BUS y una memoria compartida centralizada (Intel Core I7):



- Paralelismo

- A nivel procesador

- Multiprocesadores

- Múltiples CPUs que comparten memoria común

- MIMD (Multiple Instruction Multiple data)

- CPUs fuertemente acoplados

- Diferentes implementaciones

- Single bus y memoria compartida (centralizada) (UMA – Uniform memory access) (SMP – Symmetric multiprocessor)

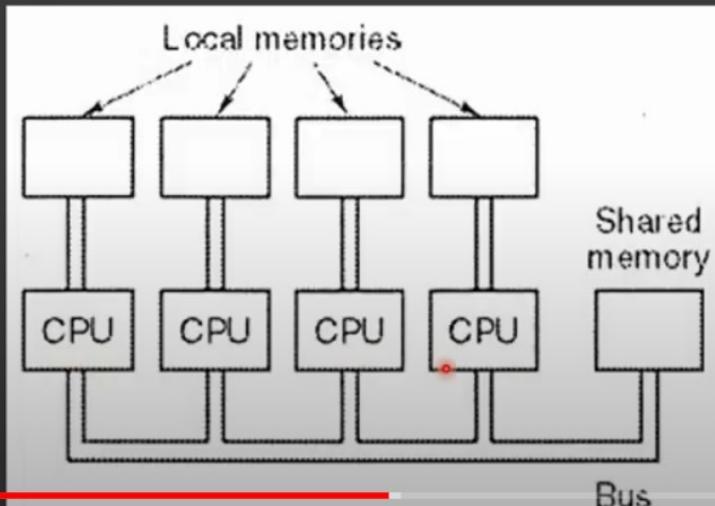
- Ej. Intel Core i7

- CPUs con memoria local y memoria compartida (NUMA – non-uniform memory access)

- Ej. BBN Butterfly, SGI Origin 2000, Compaq AlphaServer GS320, Intel Itanium 2

Después tenés implementaciones donde hay una mezcla de memoria local con memoria compartida (más común en servidores):

- CPUs con memoria local y memoria compartida



Multicomputadores

Ahora no solo se tienen CPUs independientes, si no que se tienen PCs interconectadas con su propia memoria local y se integran mediante mensajería entre computadoras. Así funcionan las supercomputadoras. Es una técnica de MIMD

- Paralelismo
 - A nivel procesador
 - Multiprocesadores
 - Múltiples CPUs que comparten memoria común
 - MIMD (Multiple Instruction Multiple data)
 - CPUs fuertemente acoplados
 - Diferentes implementaciones
 - Single bus y memoria compartida (centralizada) (UMA – Uniform memory access) (SMP – Symmetric multiprocessor)
 - Ej. Intel Core i7
 - CPUs con memoria local y memoria compartida (NUMA – non-uniform memory access)
 - Ej. BBN Butterfly, SGI Origin 2000, Compaq AlphaServer GS320, Intel Itanium 2