Sistemas Operativos
Memoria

Memoria

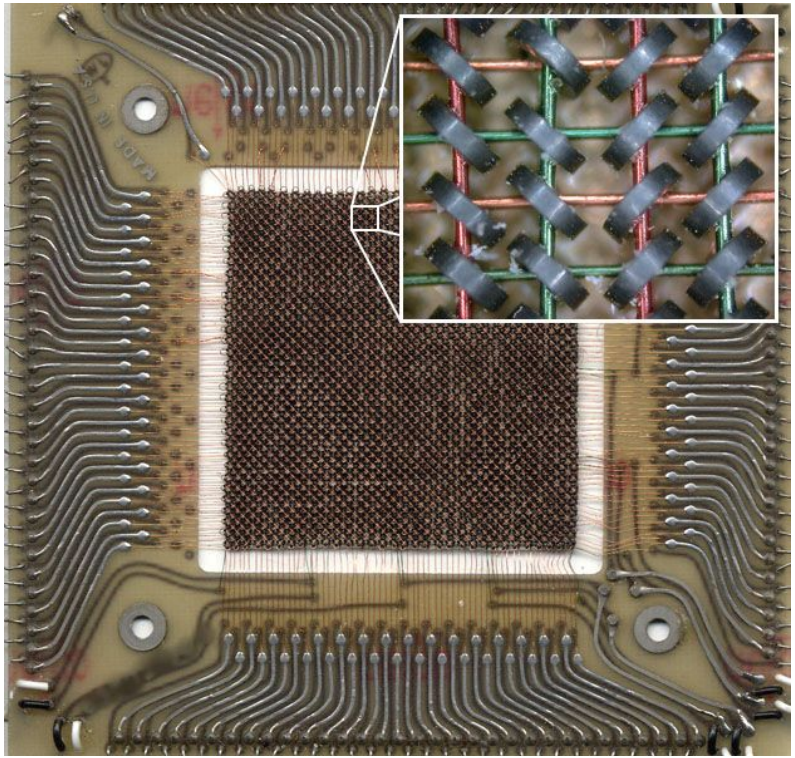
¿ Que es la Memoria?

El tema fundamental sobre la administración de memoria se encuentra en la forma adecuada de representarla.

La memoria física puede ser imaginada como un arreglo de direcciones de memoria una detrás de otra.

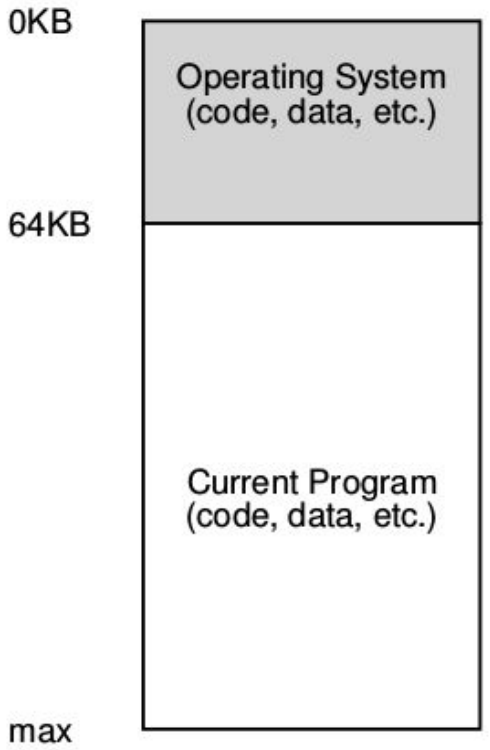
¿ Que es la Memoria?

Esta “abstracción” es manejable mientras la cantidad de memoria necesitada está en un rango “manejable” y cual es ese rango. En DOS (Disk Operating System) ese rango lo daba la elección de una arquitectura determinada, la del 8086, que permite tener direcciones de memoria de hasta 220 bits, es decir 1 MegaByte de memoria.



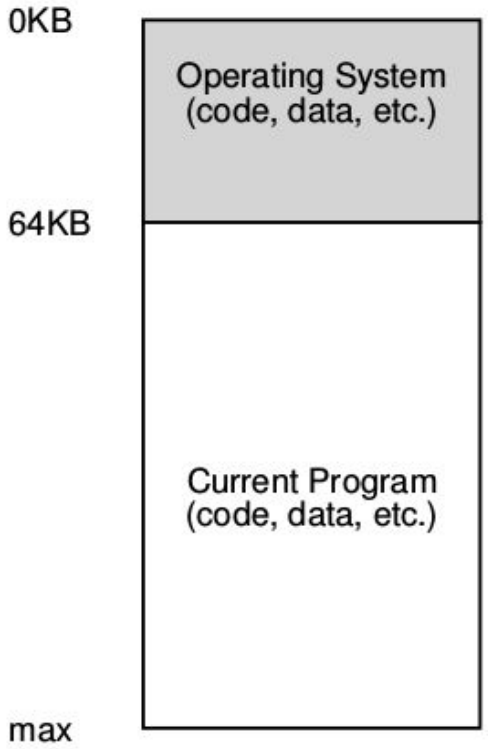
Los Early Days

En los primeros Sistemas Operativos, en “the Early Days”, en los cuales el mismo S.O. se desplegaba en solo ***64 KiloBytes*** de memoria, ***el resto de la misma estaba disponible para un único proceso ejecutándose a partir de los 64Kb de memoria física***,



Los Early Days

El sistema operativo era más o menos un conjunto de funciones o rutinas (específicamente una biblioteca) que se alojaba en la memoria empezando en la dirección física 0, y después existía un único programa en ejecución (un proceso) que se encontraba en la memoria física de la computadora, esta memoria era el resto no utilizado por el sistema operativo.

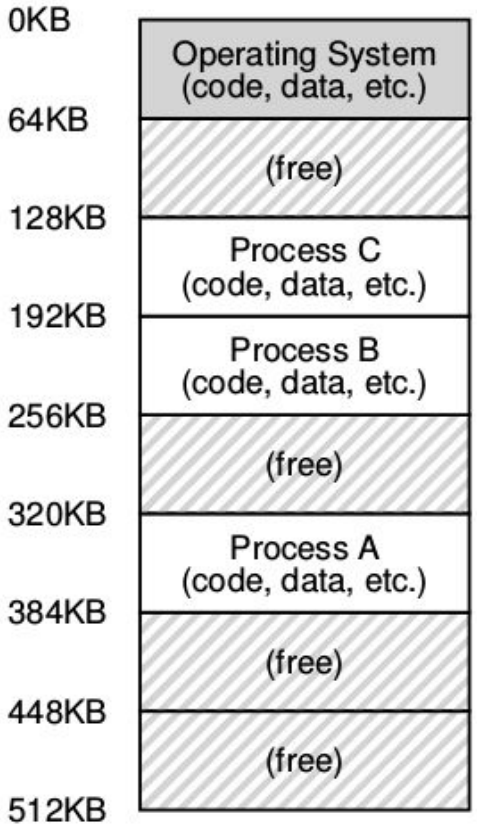


It used a paging mechanism to map the virtual addresses available to the programmer on to the real memory that consisted of 16,384 words of primary core memory with an additional 98,304 words of secondary drum memory.

Atlas Univ. Manchester 1961

Multiprogramación

Multiprogramming is a rudimentary form of parallel processing in which several programs are run at the same time on a uniprocessor. Since there is only one processor, t**here can be no true simultaneous execution of different programs**. Instead, the operating system executes part of one program, then part of another, and so on. To the user it appears that all programs are executing at the same time.



Multiprogramación

Más de un proceso estaba preparado para ser ejecutado en algún determinado momento, y el sistema operativo intercalaba dicha ejecución según la circunstancia. Haciendo esto se mejoró efectivamente el uso de la CPU, tal mejora en la eficiencia fue particularmente decisiva en esos días en la cual una computadora costaba cientos de miles o tal vez millones de dólares.

En esta era, múltiples procesos están listos para ser ejecutados un determinado tiempo según el S.O. lo decidiese en base a ciertas políticas de planificación o scheduling.

Time Sharing

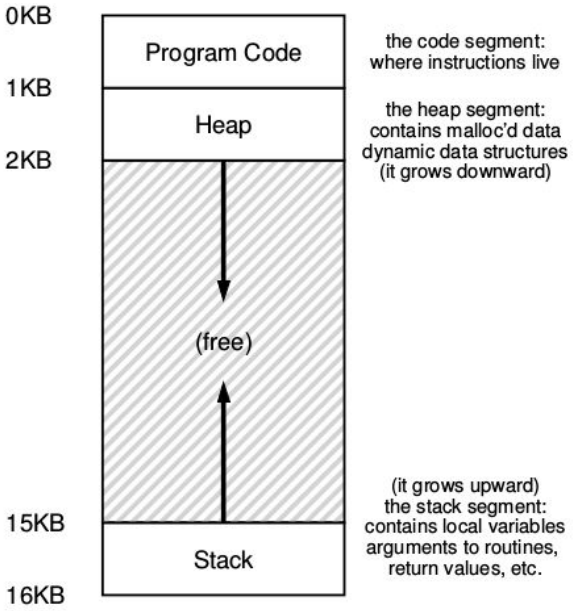
Tiempo compartido se refiere a compartir de forma concurrente un recurso computacional (tiempo de ejecución en la CPU, uso de la memoria, etc.) entre muchos usuarios por medio de las tecnologías de multiprogramación y la inclusión de interrupciones de reloj por parte del sistema operativo, permitiendo a este último acotar el tiempo de respuesta del computador y limitar el uso de la CPU por parte de un proceso dado.

Time Sharing - IBM 704 - primer sistema Time Sharing

El Espacio de Direcciones o Address Space

Crear un mecanismo que permita que la memoria física de una computadora sea utilizada de forma fácil y eficiente llevó paulatinamente a concebir el concepto de espacio de direcciones, la abstracción para la memoria.

El Address Space de un proceso contiene todo el estado de la memoria de un programa en ejecución.



Virtualización de Memoria

El sistema operativo tiene que virtualizar memoria… pero lo tiene que hacer con estilo, para ello una de las metas principales de la virtualización es la transparencia.

El sistema operativo debería implementar la virtualización de memoria de forma tal que sea invisible al programa que se está ejecutando; es más, el programa debe comportarse como si él estuviera alojado en su propia área de memoria física privada.

Pero detrás de escena, el sistema operativo y el hardware hacen todo el trabajo para multiplexar memoria a lo largo de los diferentes procesos y por ende implementa la ilusión.

Virtualización de Memoria

Otra meta importante de la virtualización es la eficiencia. El sistema operativo debe esforzarse para hacer que la virtualización sea lo más eficiente posible en términos de tiempo (no hacer que los programas corran más lentos) y espacio (no usar demasiada memoria para las estructuras necesarias para soportar la virtualización).

Por último, una tercera meta de la virtualización de memoria es la protección. El sistema operativo tiene que asegurarse de proteger a los procesos unos de otros como también proteger al sistema operativo de los procesos. Cuando un proceso realiza un load, un store, o un fetch de una instrucción este no tiene que ser capaz de hacerlo o afectar de ninguna forma al contenido de la memoria del proceso o del sistema operativo.

Virtualización de Memoria

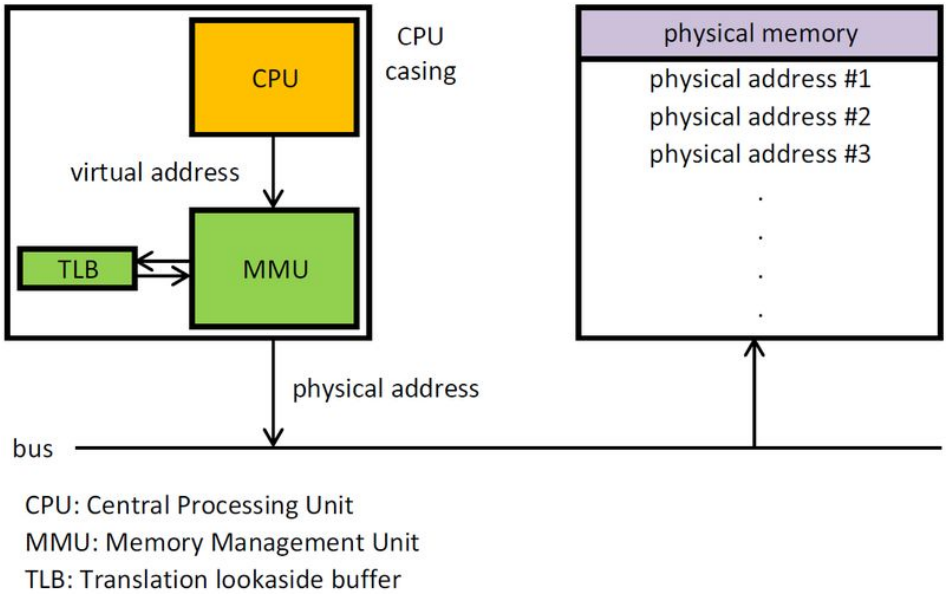
Existen dos puntos importantes a la hora de virtualizar memoria:

- la flexibilidad
- la eficiencia

Para llegar a ellos un buen mecanismo de virtualización de memoria debe ser lo más flexible y eficiente posible.

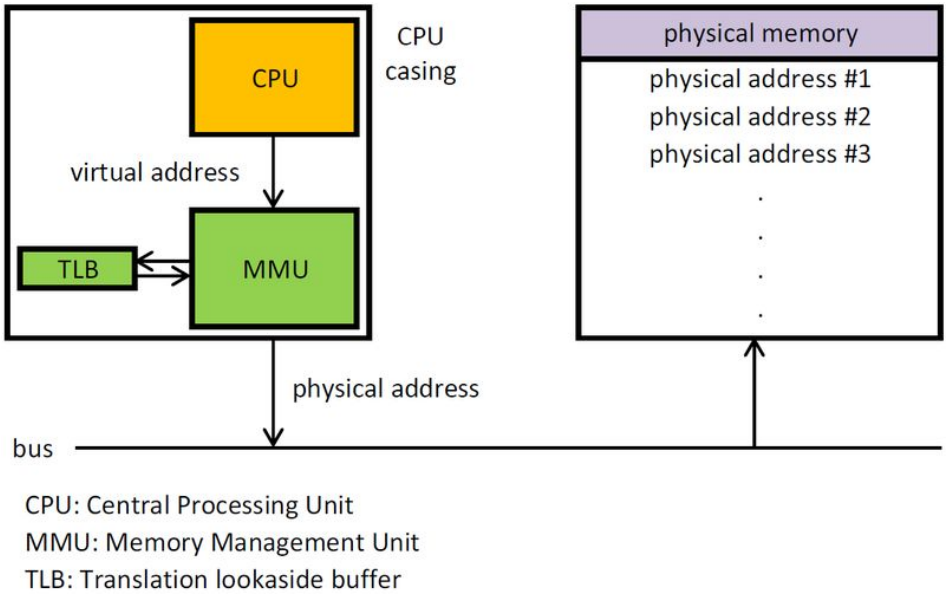
Address Translation

Con esta técnica el hardware transforma cada acceso a memoria, transformando la virtual address que es provista desde dentro del espacio de direcciones en una physical address en la cual la información deseada se encuentra realmente almacenada.

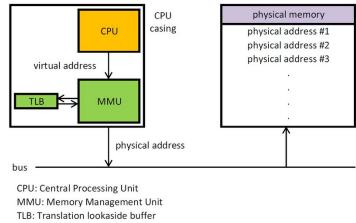


Address Translation

Entonces, en todos y por cada una de las referencias a memoria un address translation es realizada por el hardware para redireccionar las referencias a la memoria de la aplicación hacia las posiciones reales en la memoria física.



Address Translation



Es evidente, que el hardware por sí solo no puede virtualizar la memoria. Éste, sólo provee un mecanismo de bajo nivel para poder hacerlo eficientemente.

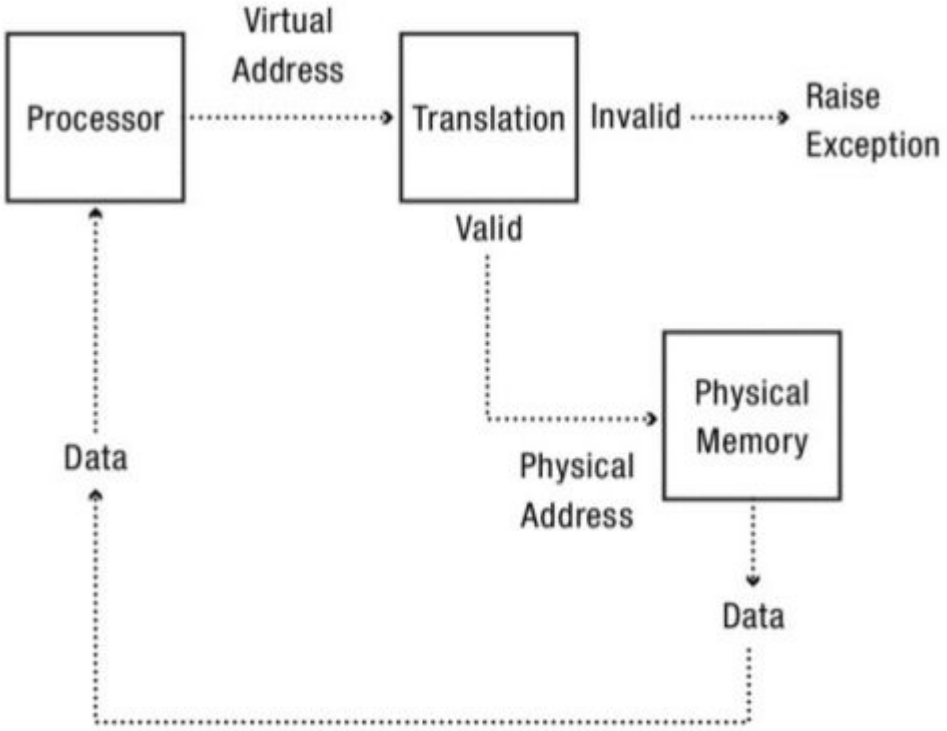
El sistema operativo tiene que involucrarse en los puntos claves para:

- setear al hardware de forma correcta para que esta traducción se de lugar;
- por eso debe entonces gerenciar la memoria, manteniendo información de en que lugar hay áreas libres y en que lugar hay áreas en uso;
- y por último intervenir de forma criteriosa como mantener el control sobre toda la memoria usada.

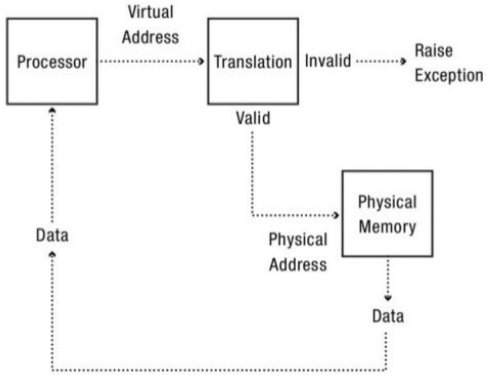
Address Translation

El S.O. está en el medio de ese proceso determina si el mismo se ha realizad correctamente. Lo que hace es gerencia el manejo de la memoria:

- Manteniendo registro de qué part está libre.
- Qué parte está en uso.
- Manteniendo el control de la form en la cual la memoria está siend utilizada.



Address Translation



Formalmente, el proceso de address translation es un mapeo entre los elementos de un espacio de direcciones virtuales de N-elementos (VAS) y un espacio de direcciones físicas de M-elementos (PAS):

$$MAP: VAS \Rightarrow PAS \cup \varnothing$$

$$MAP(A) = \begin{cases} A' \text{ si los datos en la dir. virtual } A \text{ están presentes la dir. física } A' \text{ de } PAS \\ \varnothing \text{ si los datos de la dir. virtual } A \text{ no están presentes en la memoria física.} \end{cases}$$

Hacia una Flexible Address Translation

Para ir ganando comprensión sobre el ***hardware-based address translation***, se verá su primera implementación, introducida en las primera máquinas que utilizaban time-sharing hacia el fin de los años 50, es una idea simple llamada base y segmento también puede ser vista como ***dynamic realocation***.

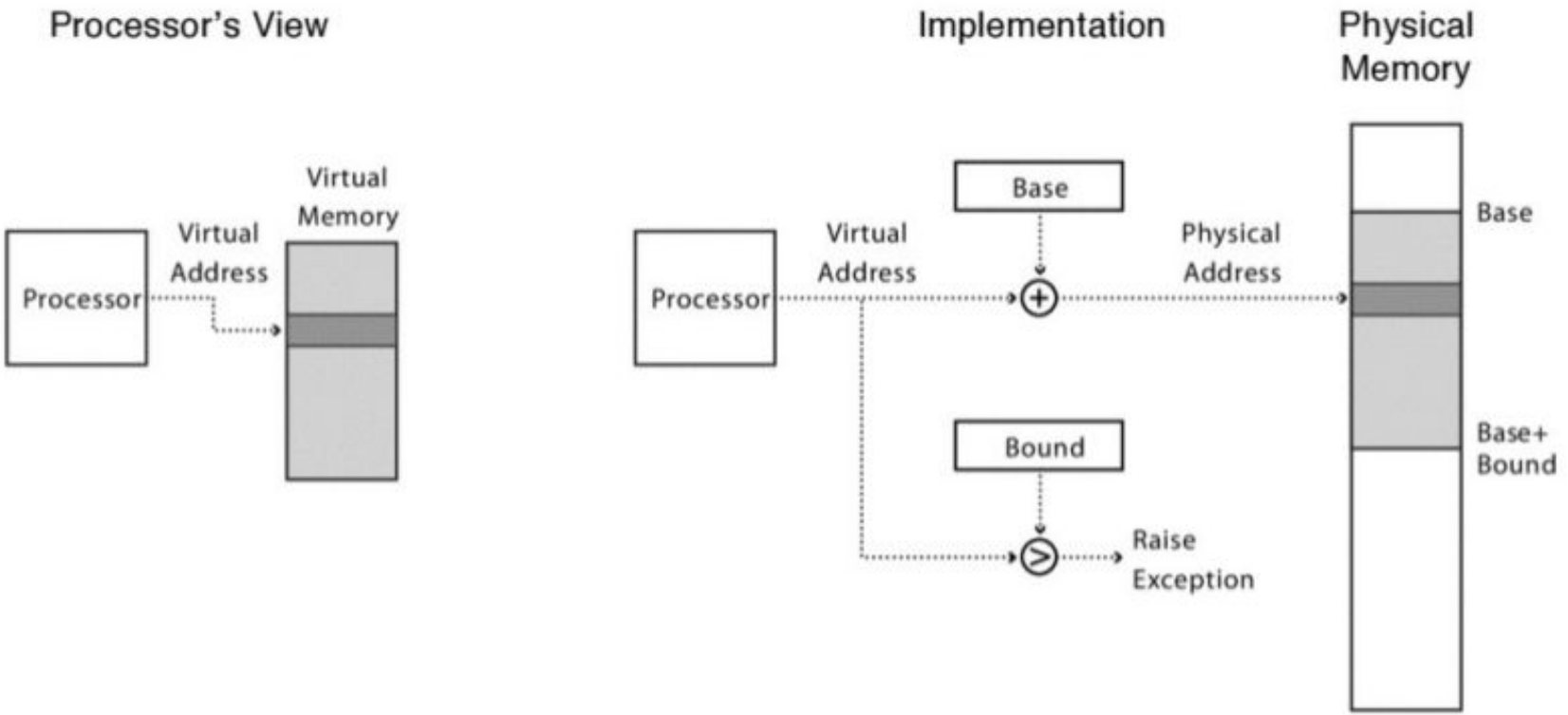
Específicamente solo se necesitan dos registros de hardware dentro de cada cpu: Uno llamado registro base y el otro registro límite o Segmento.

Base and Bound

Específicamente solo se necesitan dos registros de hardware dentro de cada cpu:

- registro base
- registro límite o Segmento.

Este par base-límite va a permitir que el address space pueda ser ubicado en cualquier lugar deseado de la memoria física, y se hará mientras el sistema operativo se asegura que el proceso solo puede acceder a su address space.



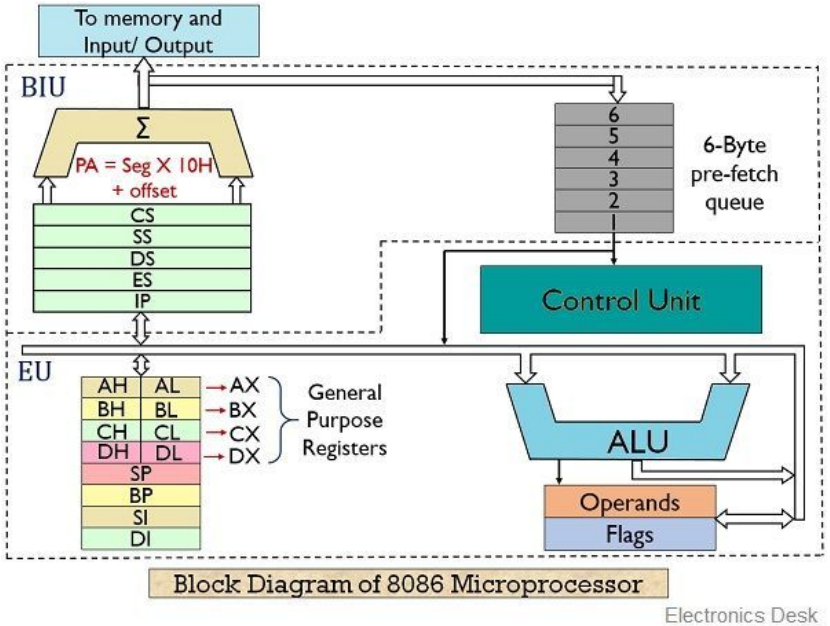
Base and Bound (segmentación)

En X86

AX, BX, CX, DX : Registros Generales

CS, DS, SS, ES : Registros de Segmentos, manejan la traducción en modo real.(Code Segment, Data Segment, Extra Data Segment)

SI, DI, BP, SP, IP: Registros de Punteros y Registro de Índices. (Base Pointer, Stack Pointer y Instruction Pointer)



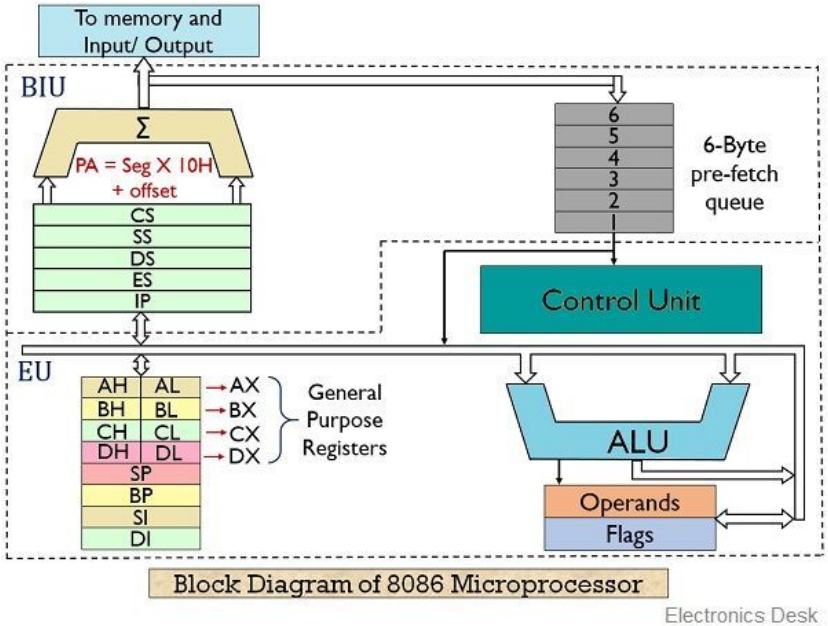
En X86

CS:IP localiza la próxima instrucción a ser ejecutada en modo real.

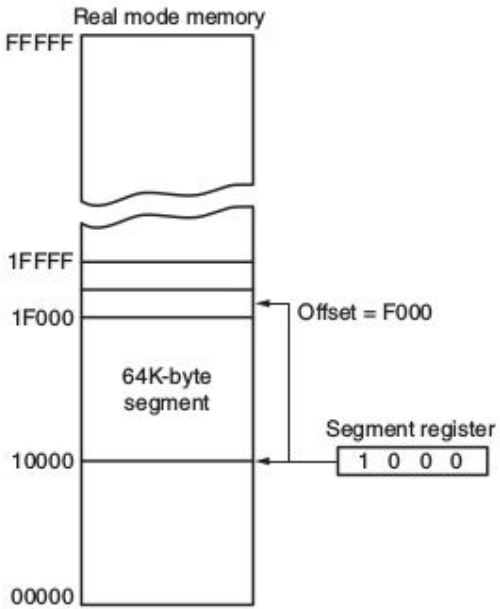
SS:SP localiza la dirección del puntero al stack, a veces también puede ser SS:BP.

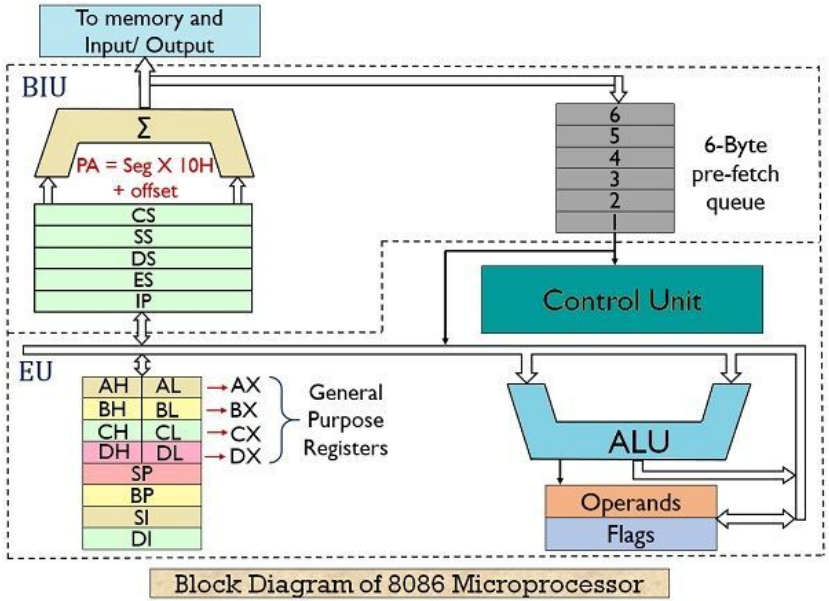
DS: BX,DI,SI localizan el puntero a una dirección de memoria dentro del data address.

ES:DI puntero al extra data address donde van los strings.



En X86 Real Mode

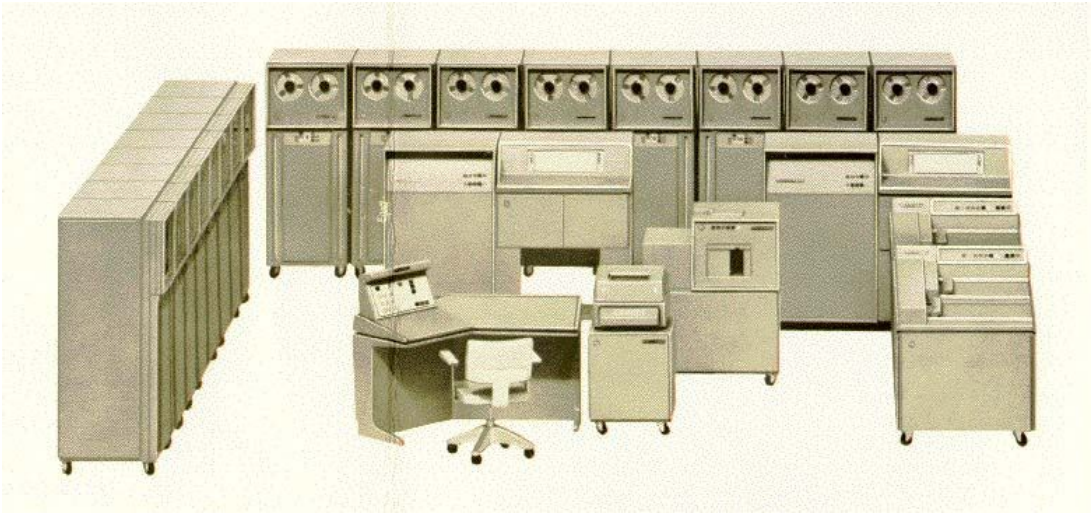




Block Diagram of 8086 Microprocessor

Electronics Desk

Segmentación B5000



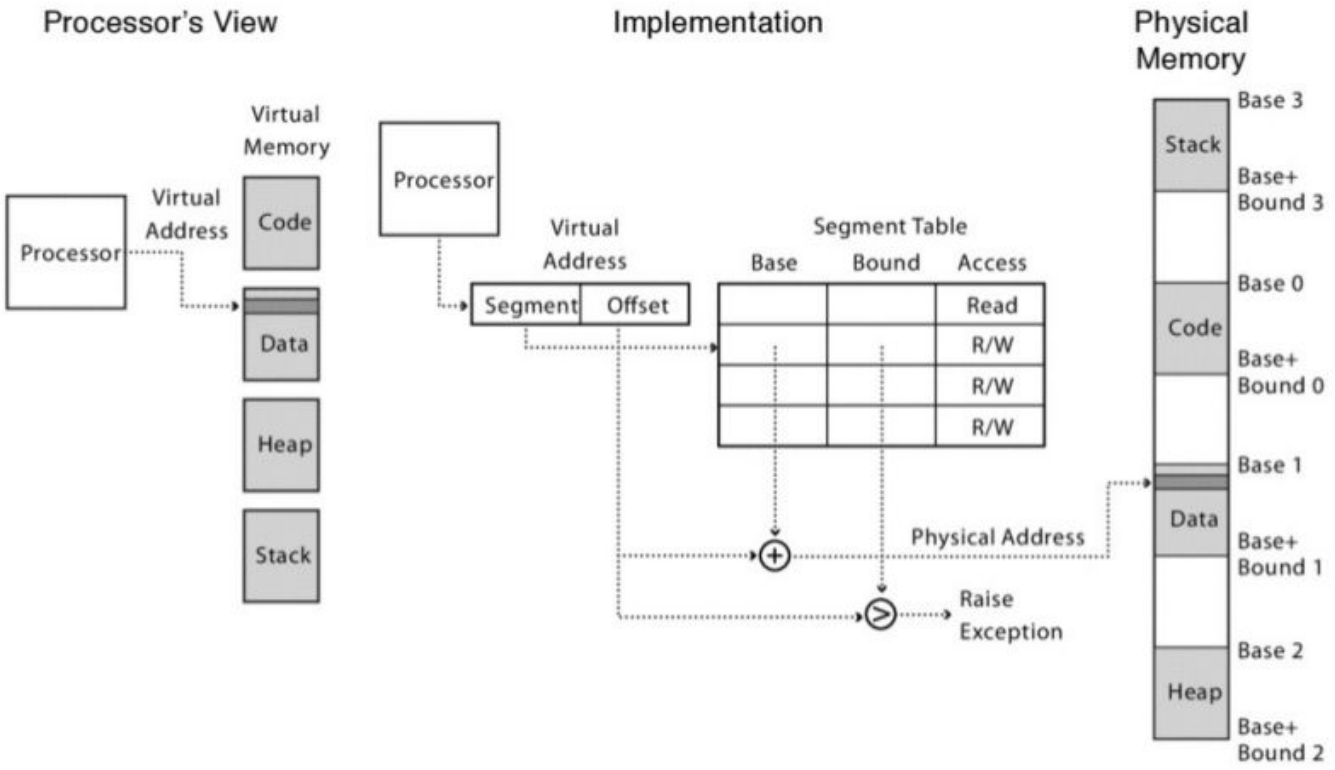
Address Translation con Tabla de Segmentos

El problema de la técnica anterior es que se tiene un solo registro base y solo un segmento. La mejora a este método es mediante la aplicación de un pequeño cambio: en vez de tener un solo registro límite, se tiene un arreglo de pares de (registro base, segmento) por cada proceso.

Una dirección virtual tiene dos componentes:

un número de segmento:un offset de segmento

El número de segmento es el índice de la tabla para ubicar el inicio del segmento en la memoria física. El registro bound es chequeado contra la suma del registro base+offset para prevenir que el proceso lea o escriba fuera de su región de memoria.



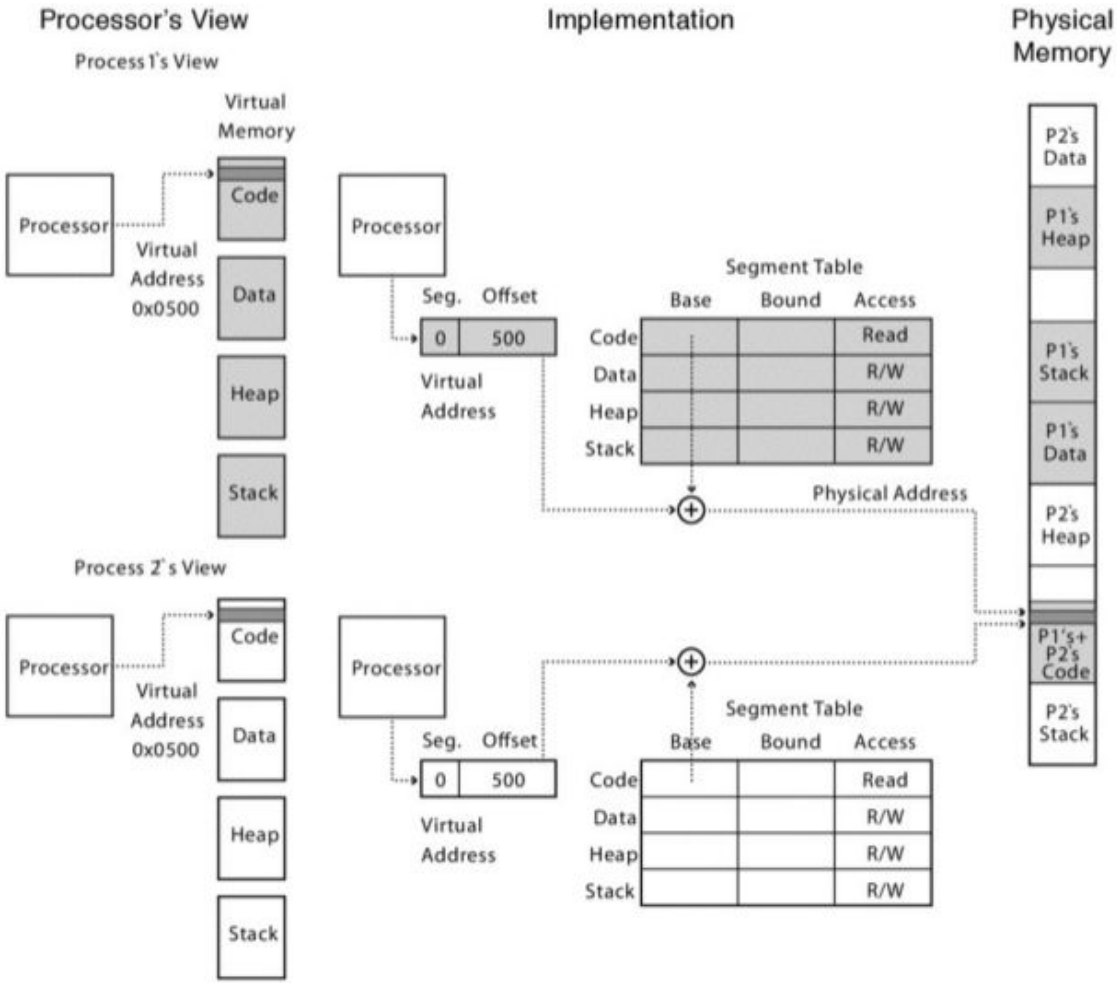
Address Translation con Tabla de Segmentos

Address Translation con Tabla de Segmentos

En una dirección virtual utilizando esta técnica, los bit de más alto orden son utilizados como índice en la tabla de segmentos. El resto se toma como offset y es sumado al registro base y comparado contra el registro bound.

El número de segmentos depende de la cantidad de bits que se utilizan como índice.

El error de **Segmentation Fault** era un error que se daba cuando, en las máquinas que implementan segmentación, se quería direccionar una posición fuera del espacio direccionable. Increíblemente este error aún se utiliza en sistemas operativos que no usan segmentación.

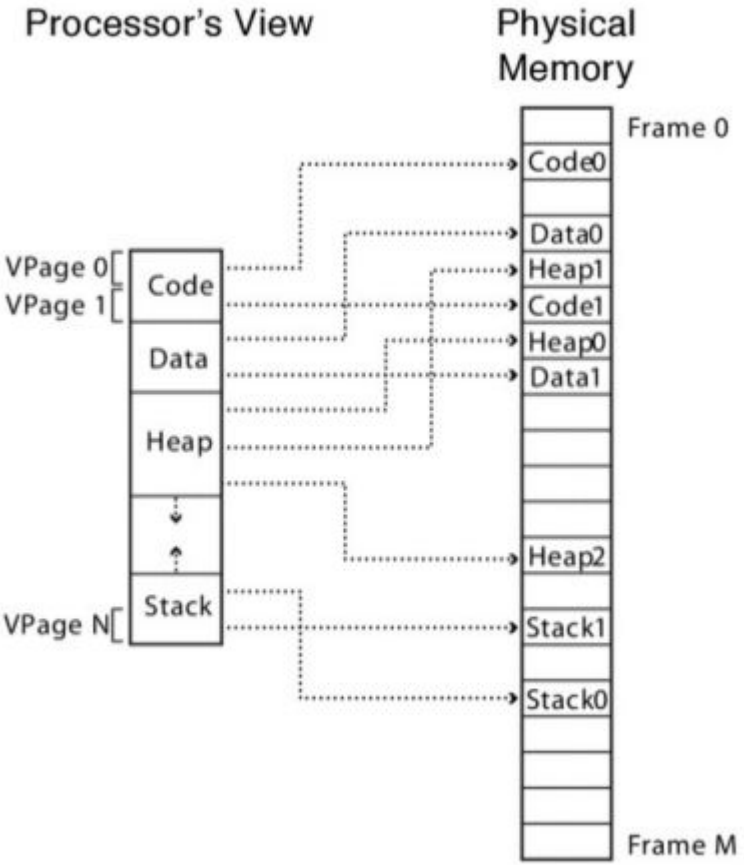

Processor's View
Implementation
Physical Memory
Process 1's View
Virtual Memory
Processor
Code
Data
Heap
Stack
Virtual Address 0x0500
Processor
Seg.
Offset
0
500
Virtual Address
Segment Table
Base
Bound
Access
Code
Data
Heap
Stack
Read
R/W
R/W
R/W
Physical Address
P2's Data
P1's Heap
P1's Stack
P1's Data
P2's Heap
P1's+ P2's Code
P2's Stack
Process 2's View
Processor
Code
Data
Heap
Stack
Virtual Address 0x0500
Processor
Seg.
Offset
0
500
Virtual Address
Segment Table
Base
Bound
Access
Code
Data
Heap
Stack
Read
R/W
R/W
R/W

Memoria Paginada

- Una alternativa a la memoria segmentada es la memoria paginada.
- El address translation es similar a como se trabaja con la segmentación.
- En vez de tener una página de segmentos cuyas entradas contienen punteros a segmentos, hay una tabla de páginas por cada proceso cuyas entradas contienen punteros a las page frames.
- Teniendo en cuenta que los page frames tienen un tamaño fijo, y son potencia de 2,las entradas en la page table sólo tienen que proveer los bit superiores de la dirección de la page frame. De esta forma van a ser más compactos. No es necesario tener un límite; la página entera se reserva como una unidad.

Memoria Paginada

Con la paginación, la memoria es reservada en pedazos de tamaño fijo llamados page frames.

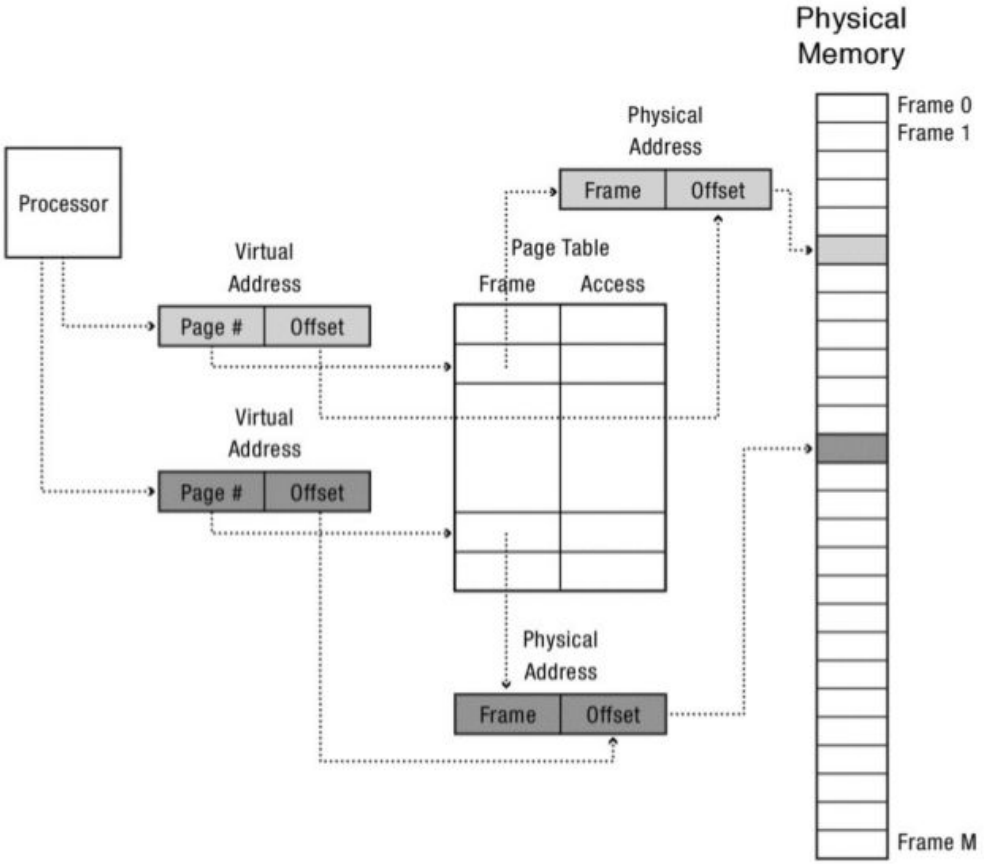


Memoria Paginada

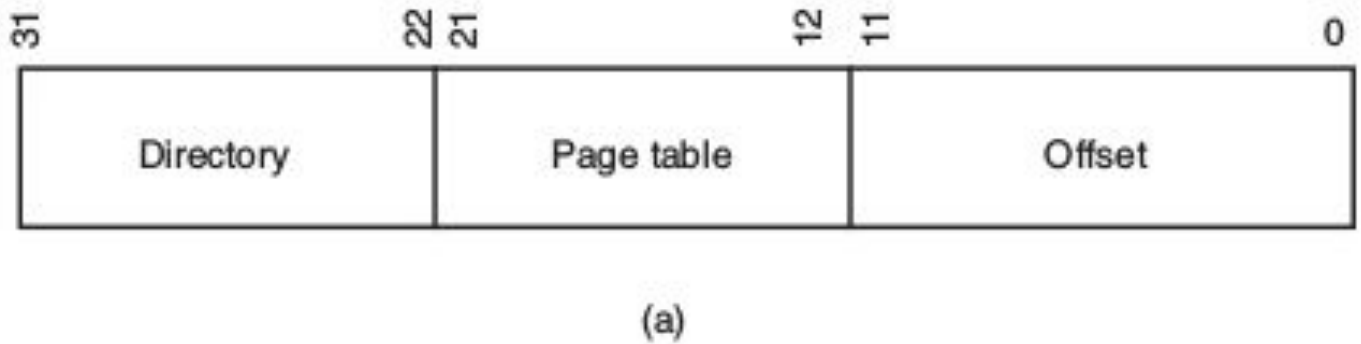
El número de la página virtual es el índice en la **page table** para obtener el page frame en la memoria física.

La dirección física está compuesta por la dirección física de**l Frame Page** que se obtiene de la page table concatenado con el offset de la página que se obtiene de la virtual address. El sistema operativo maneja los accesos a la memoria.

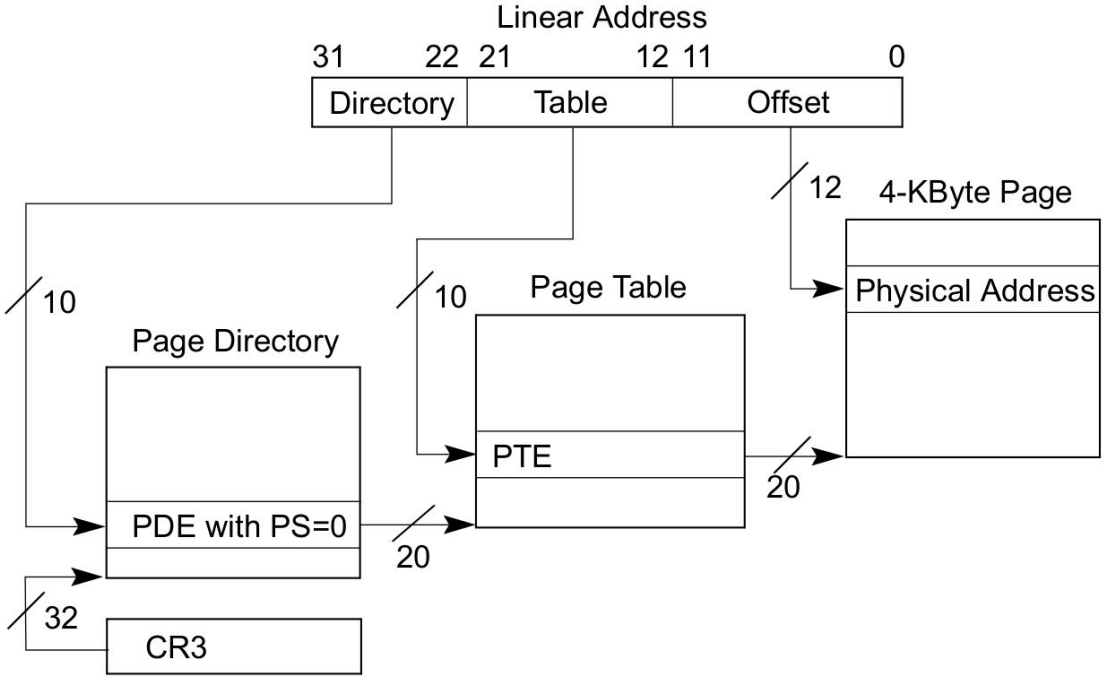
Una de las cosas que pueden parecer raras sobre la paginación es que si bien el programa cree que su memoria es lineal, de hecho, su memoria está desparramada por toda la memoria física como si fuera un mosaico.

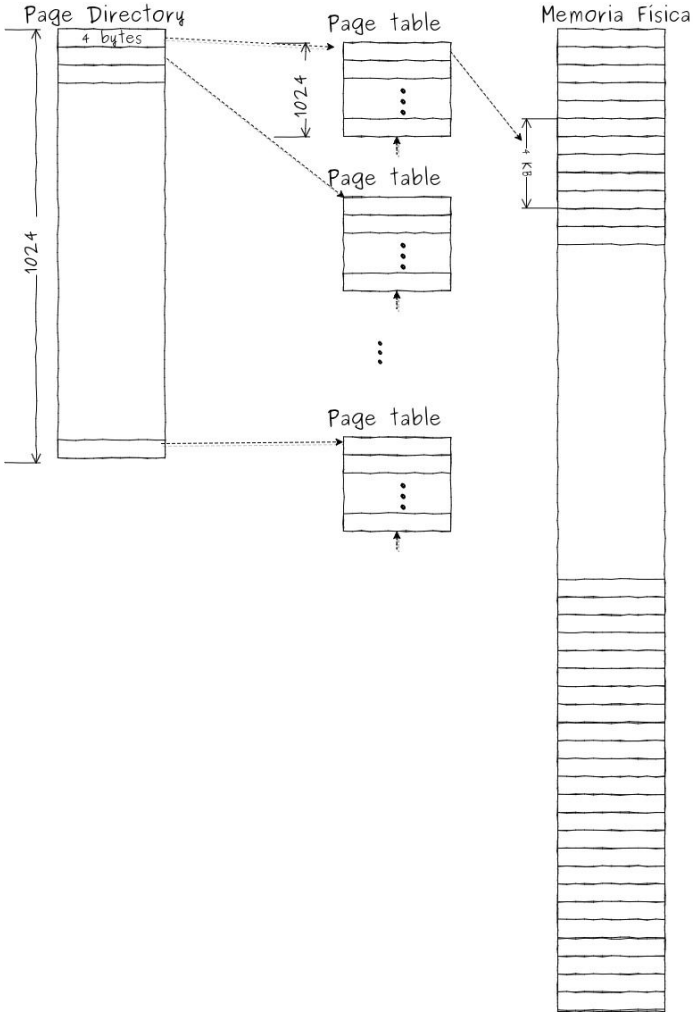


Memoria Paginada x86: Virtual Address



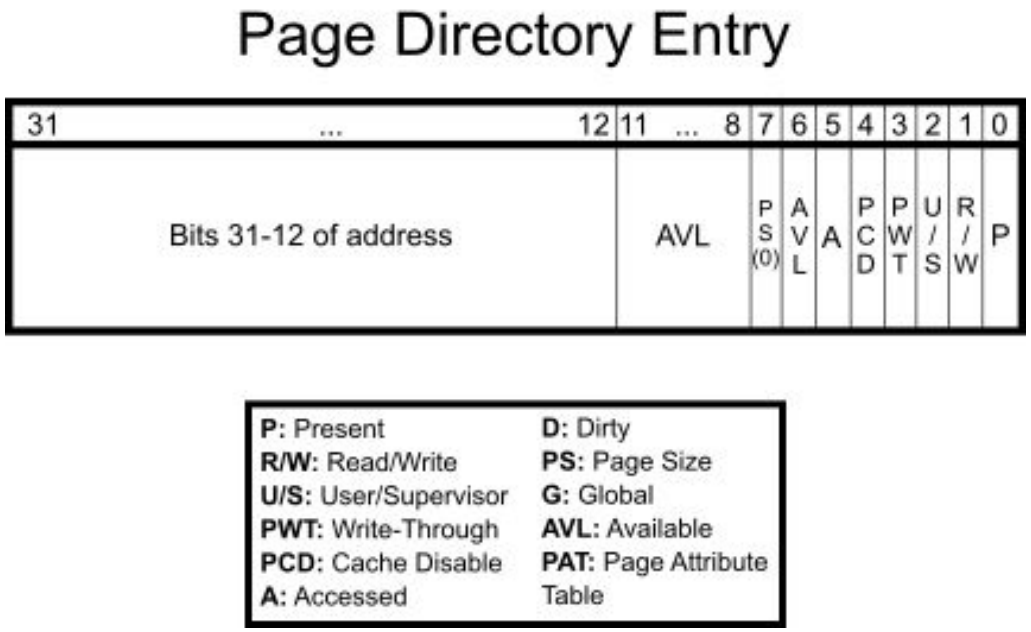
Page Directory bits 31-22 (10 bits)
Page Table Entry bits 21-12 (10 bits)
Memory Page offset address bits 11-0 (12 bits)



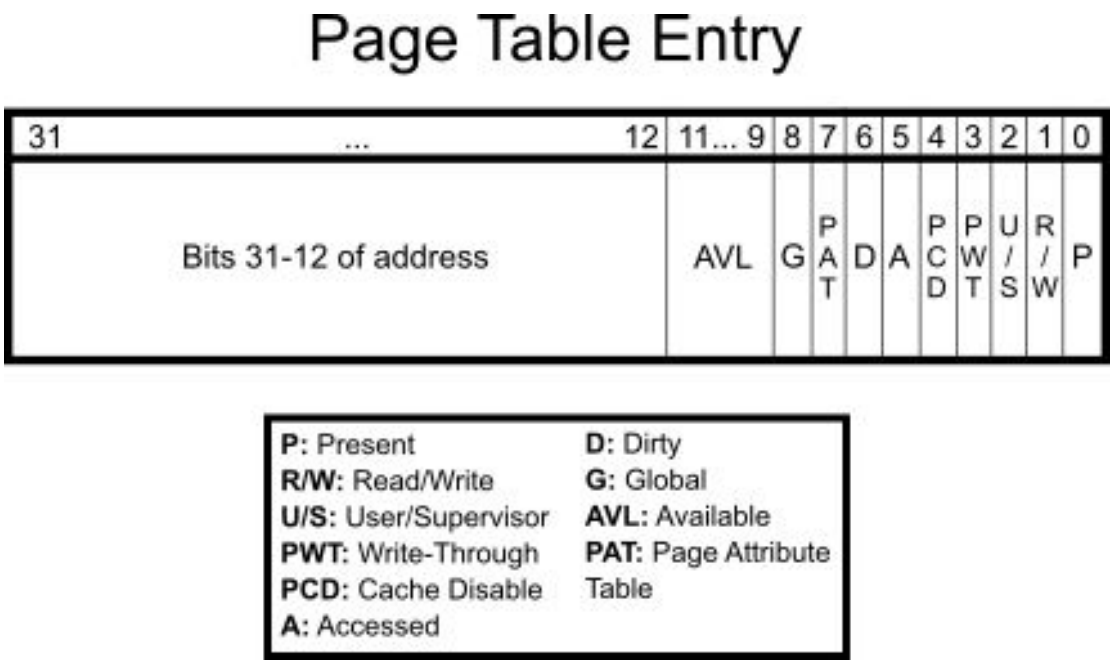


Memoria Paginada x86

Memoria Paginada x86 : Page Directory Entry



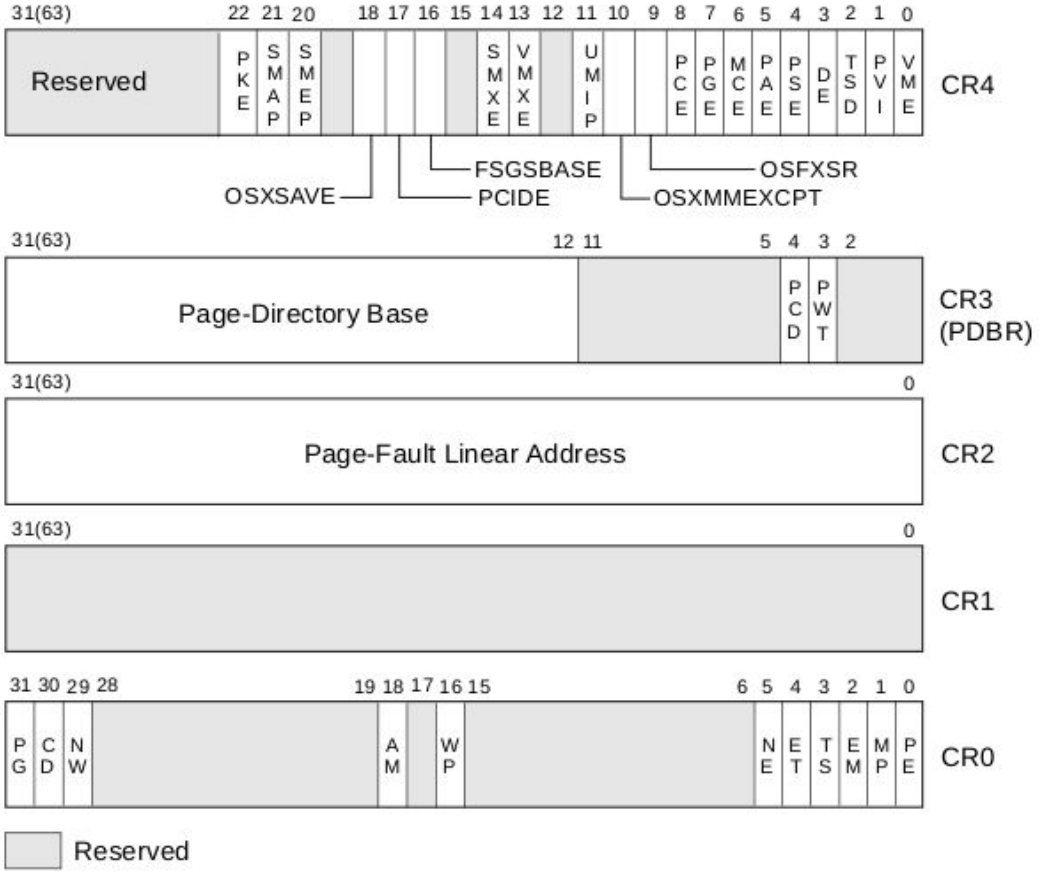
Memoria Paginada x86 : Page Table Entry



Estos registros se encuentran a partir del microprocesador intel 80386 hasta Core2. A partir del Pentium4 un registro llamado CR4 extiende la arquitectura x86.
Los registros importantes para la paginación son: CR0 y CR3.

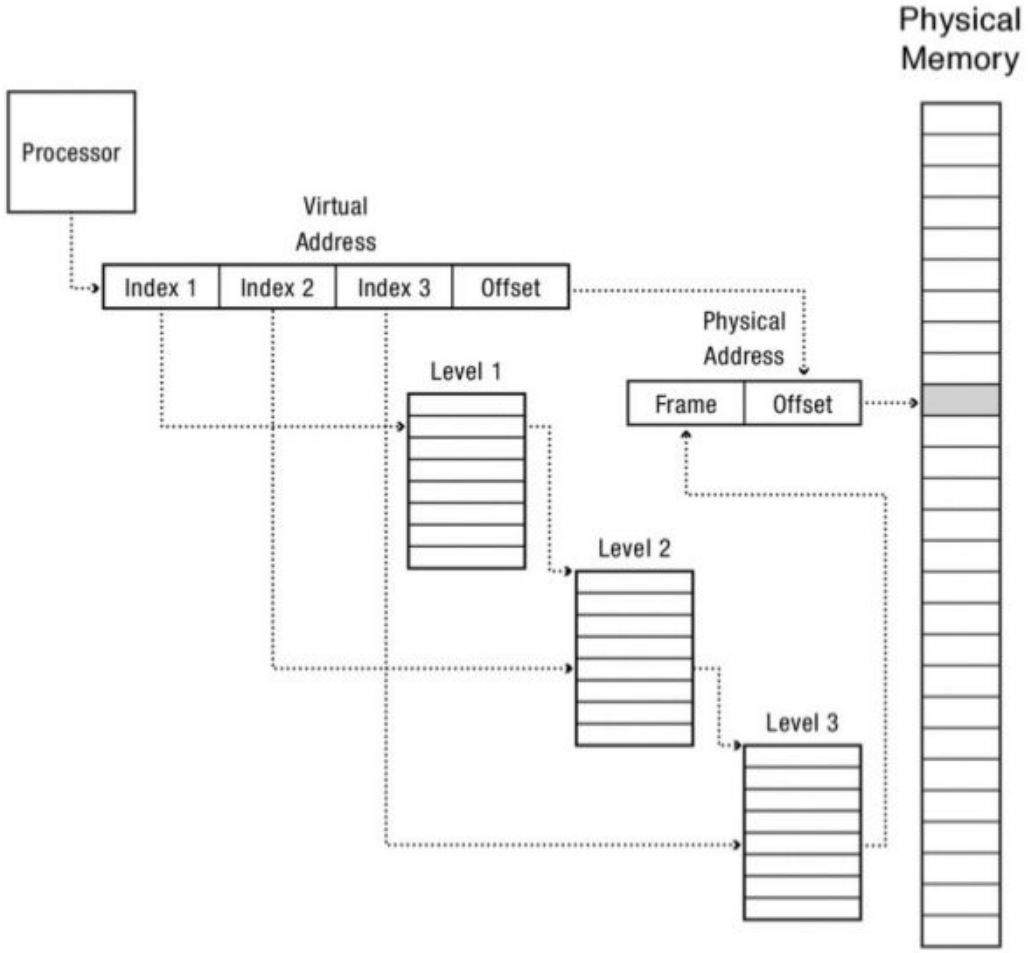
El bit más a la izquierda de el registro CR0 si está en 0 determina que la lineal address se convierte directamente en physical address para acceder a la memoria. Si PG está en 1 la lineal address debe ser convertida en physical address a través del mecanismo de paginación.

CR3 contiene page directory base que contiene 1024 entradas de 4 bytes cada una, cada entrada en el page directory ocupa 4 bytes y direcciona a una page table que contiene 1024 entradas.



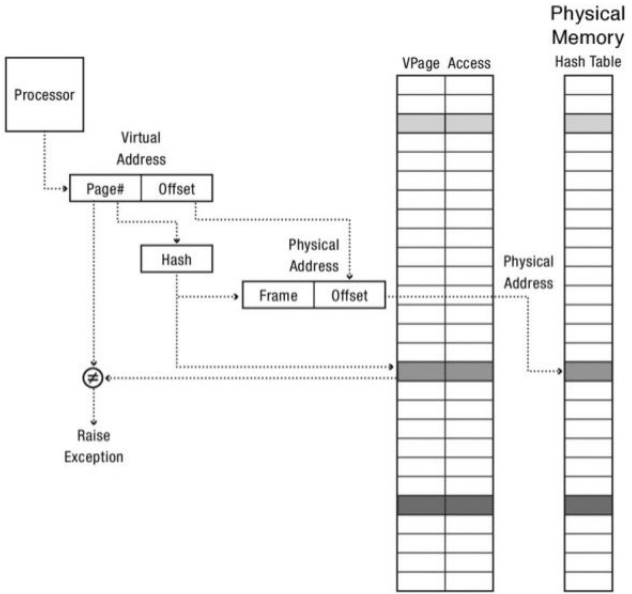
Memoria Paginada x86

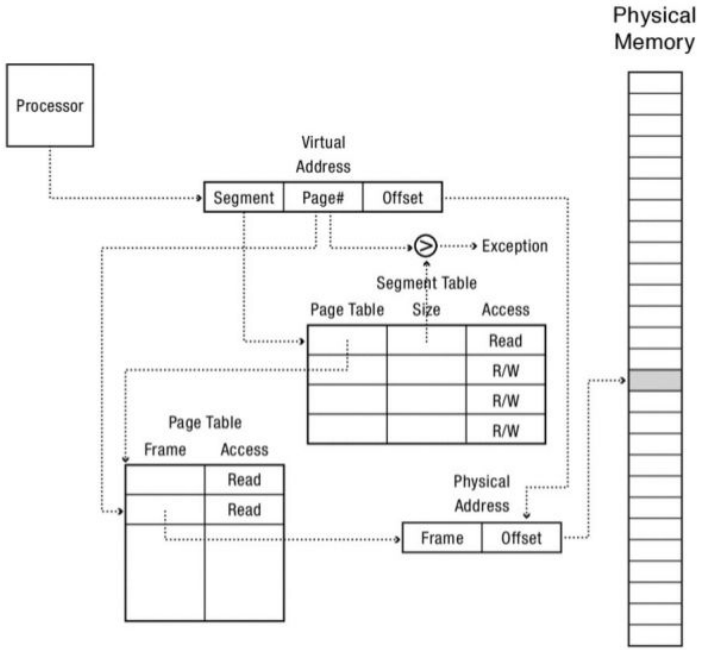
Memoria Paginada



Listo se termino?

Listo se termino?





Leer Dahlin!!!

Hacia una Eficiente Address Translation

Hasta aquí ya uno puede estar un poco cansado de todos los mecanismos de hardware para realizar la traducción de direcciones de memoria.

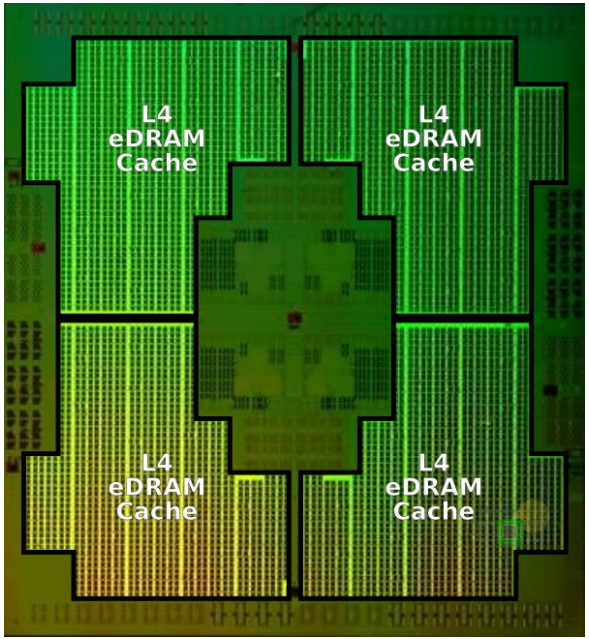
El tema es que muchos de estos métodos tienen varios niveles, hasta 4 en algunos casos, para alcanzar una dirección física, entonces eso lo hace realmente poco práctico para el procesador.

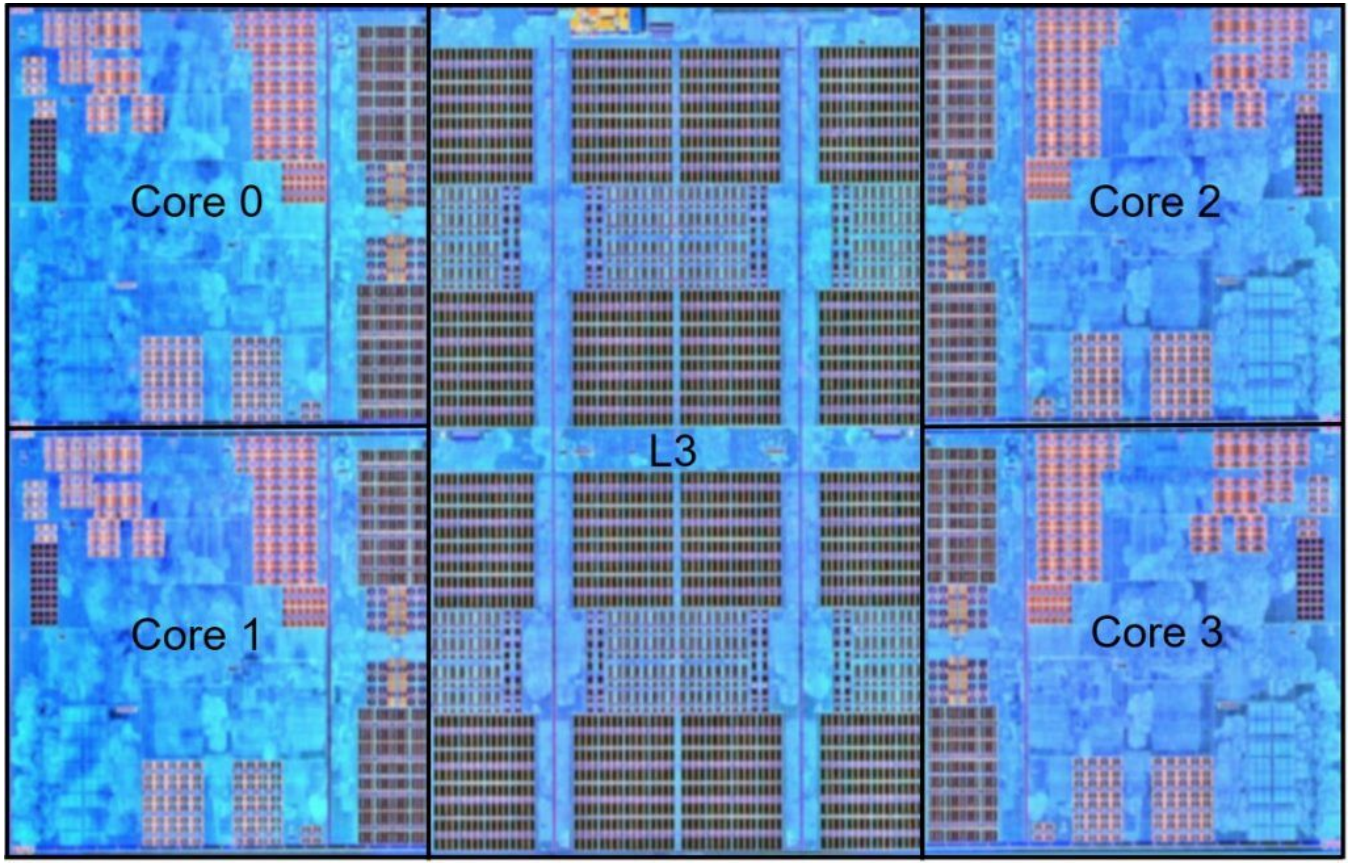
A continuación se mostrarán mecanismos para mejorar el rendimiento de la traducción de las direcciones.

Hacia una Eficiente Address Translation

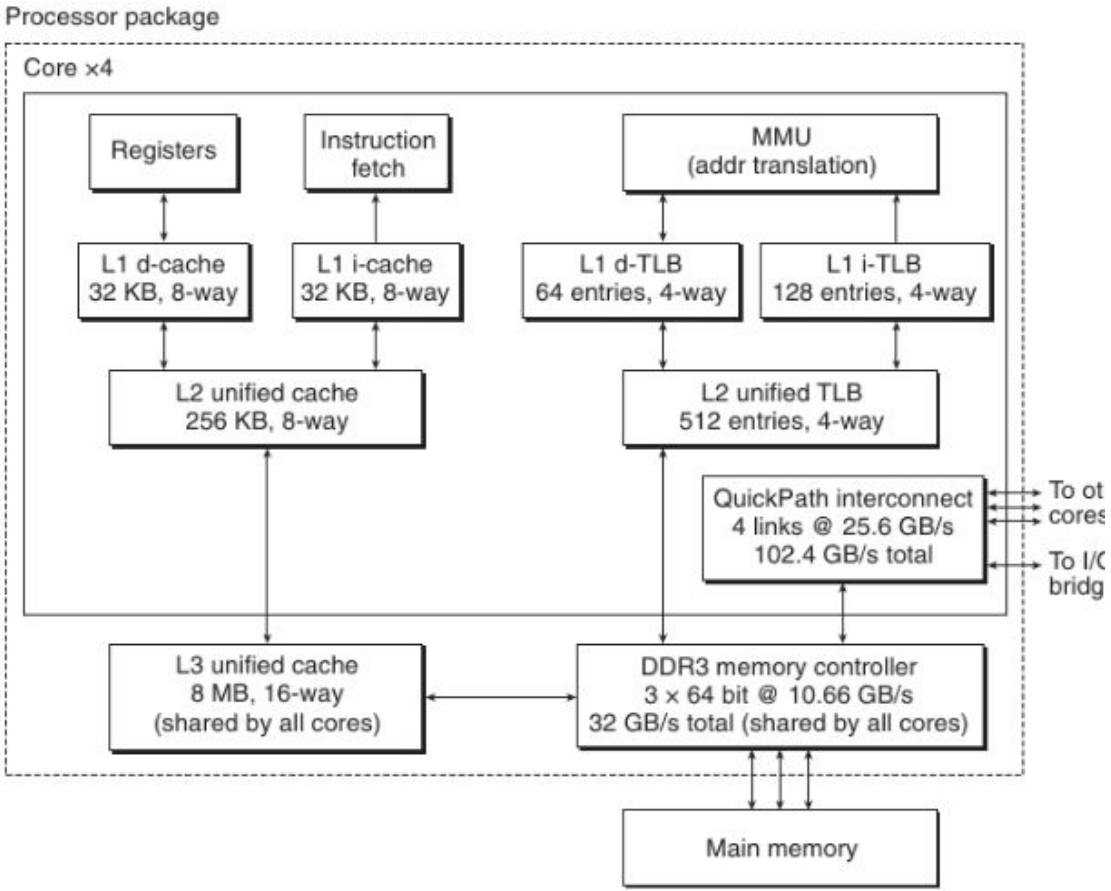
Este nuevo mecanismo usará un caché (o escondrijo), que consiste en una copia de ciertos datos que pueden ser accedidos más de una vez más rápidamente.

El concepto de Cache es ampliamente utilizado en muchas ramas de las ciencias de la computación: arquitectura de computadoras, sistemas operativos, sistemas distribuidos.





Hacia una Eficiente Address Translation



Hacia una Eficiente Address Translation

Hacia una Eficiente Address Translation: TLB

Uno de los problemas del address translation reside en la velocidad de la traducción para ello se utilizan técnicas que mejoran la velocidad de esta traducción.

Para mejorar el address translation se utiliza un mecanismo de hardware llamado **Translation-Lookaside Buffer**; o también conocido como TLB.

La TLB es parte de la MMU y es simplemente un mecanismo de cache de las traducciones más utilizadas entre los pares virtual to physical address. Por ende un mejor nombre para este mecanismo podría ser address translation cache.

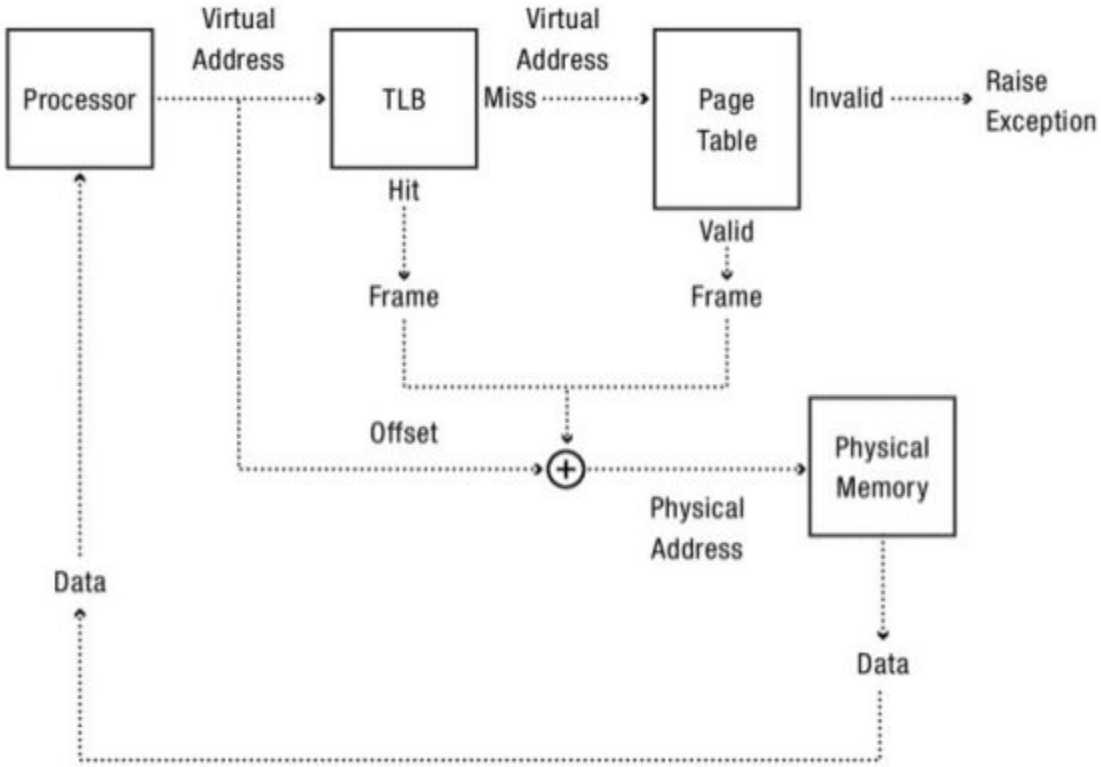
Por cada referencia a la memoria virtual, el hardware primero chequea la TLB para ver si esa traducción esta guardada ahí; si es así la traducción se hace rápidamente sin tener que consultar a la page table (la cual tiene todas las traducciones).

Translation-Lookaside Buffer

```
...
mov r1,r2
mult r1,r2
...
```

```
TLB entry = {
        virtual page number,
        physical page frame number,
        access permitions
}
```

- Secuencia y localización
- El hardware tiene que hacer el fetch de la instrucción add, después pasear por todos los multiniveles de las tablas de traducción para encontrar la dirección física de la instrucción add; ejecutar la instrucción , incrementar el contador de programa y volver a hacer todo esto otra vez para la próxima instrucción y además para sus datos pero esto es muy ineficiente.
- La Translation Lookaside Buffer (TLB) es una pequeña tabla a nivel hardware que contiene los resultados de la recientes traducciones de memorias realizadas. Cada entrada de la tabla mapea una virtual page a una physical page



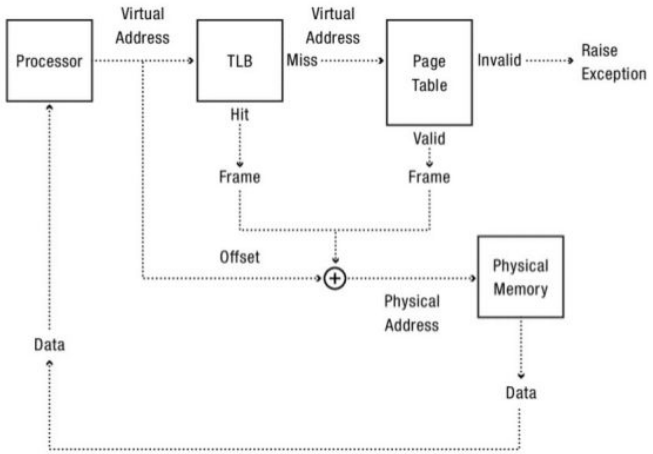
Translation-Lookaside Buffer

Translation-Lookaside Buffer

Normalmente se chequean todas las entradas de la TLB contra la virtual page, si existe matcheo el procesador utiliza ese matcheo para formar la physical address, ahorrándose todos los pasos de la traducción.

Esto se llama un TLB hit

Cuando del proceso anterior no existe matcheo en la TLB , se dice que se tiene un TLB miss



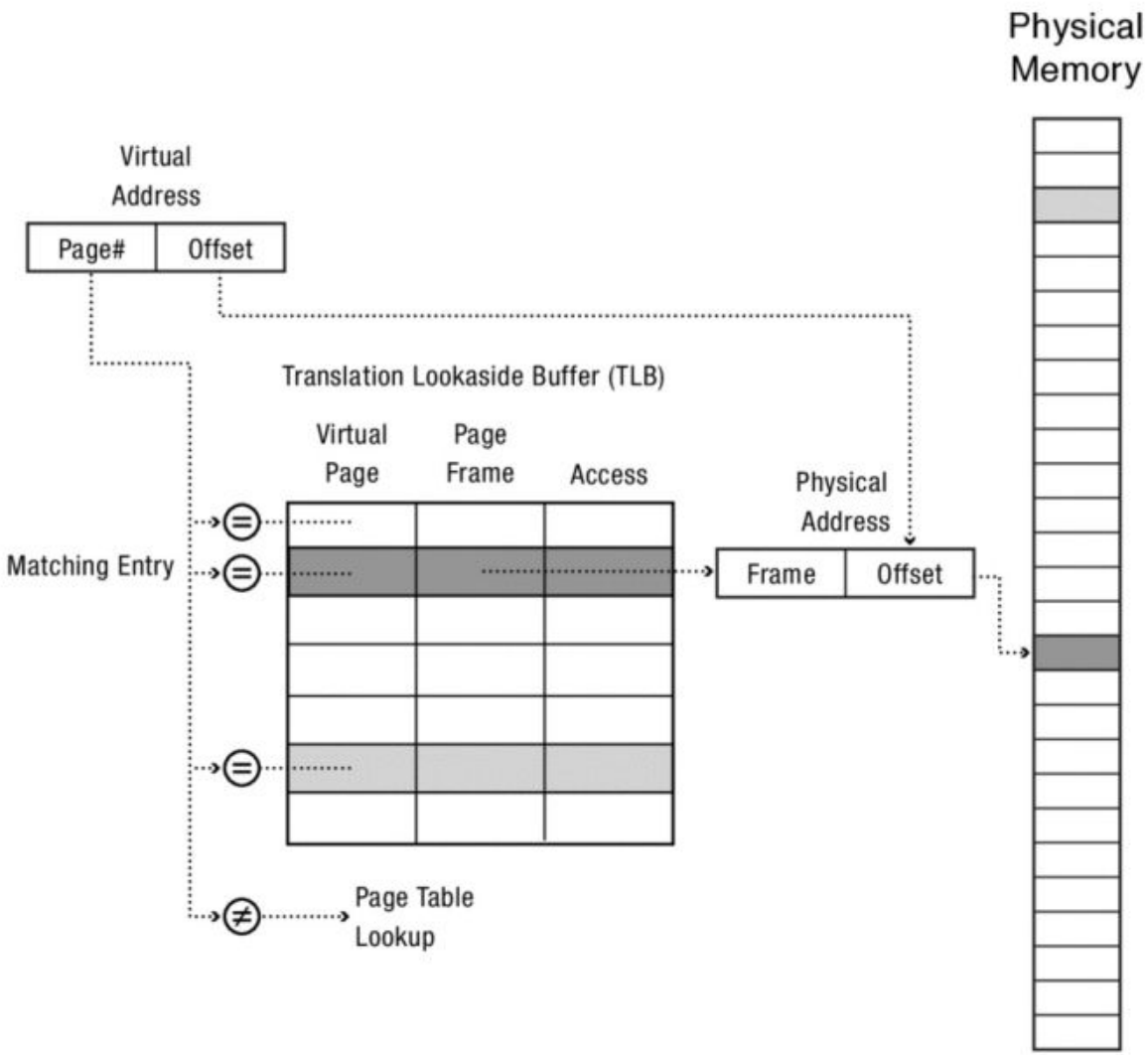
Translation-Lookaside Buffer

Para que sea útil, la búsqueda de la TLB necesita ser mucho más rápido que realizar una traducción completa de una dirección de memoria.

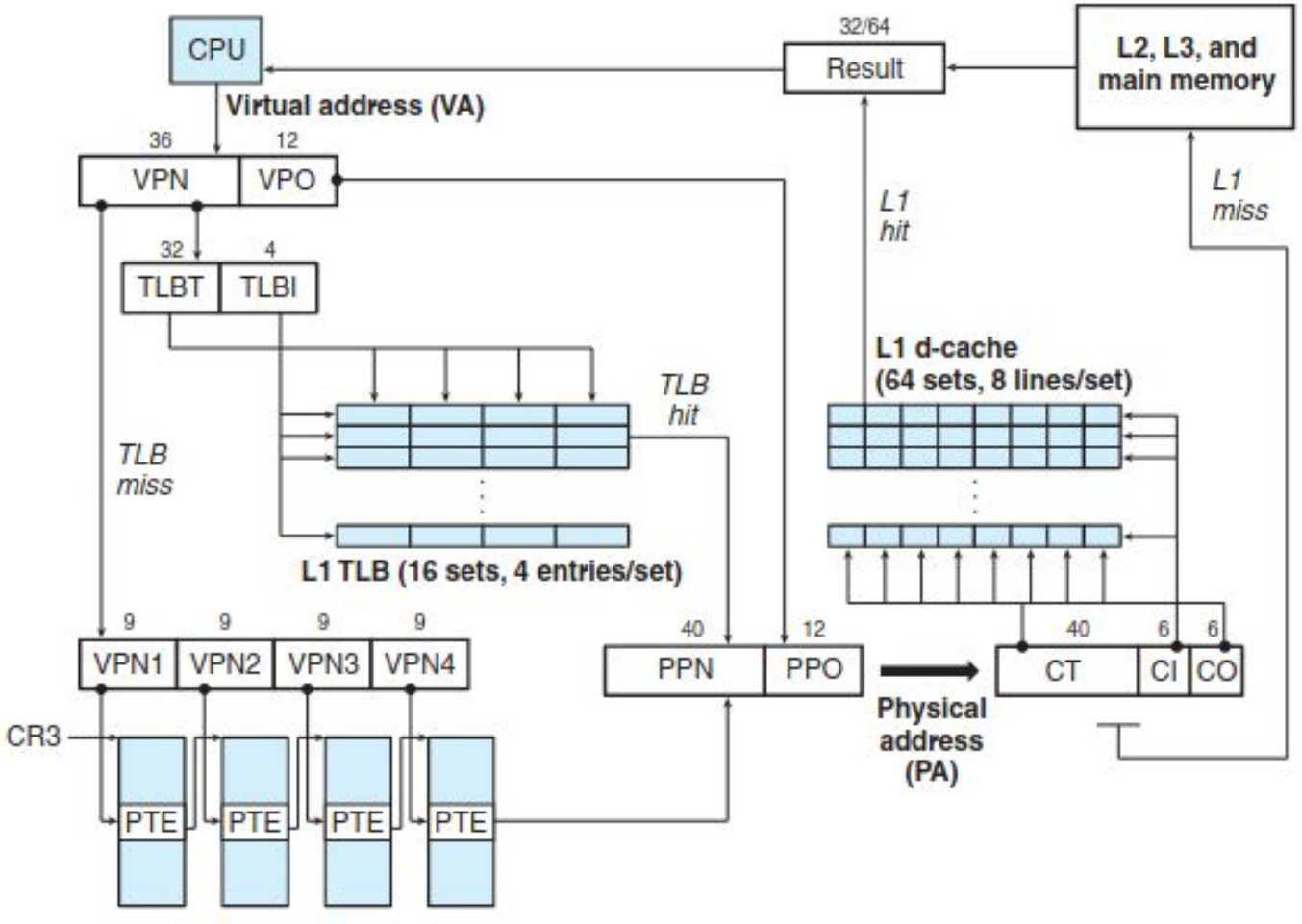
Por ende, las entradas de la tabla de la TLB son implementadas en una memoria muy rápida, memoria estática on-chip, situada muy cerca del procesador. De hecho, para mantener esta búsqueda rápida, muchos sistemas en la actualidad incluyen múltiples niveles de TLB. En general, cuanto más pequeña es la memoria, más rápida es la búsqueda.

Si el primer nivel de la TLB produce un TLB miss existe otro nivel, que guarda más datos, y este es consultado y la traducción se realiza si se falla en ambos niveles

La TLB


Physical Memory
Virtual Address
Page#
Offset
Translation Lookaside Buffer (TLB)
Virtual Page
Page Frame
Access
Physical Address
Frame
Offset
Matching Entry
Page Table Lookup

Todo completo



Consistencia de la TLB

Cada vez que se introduce un cache en el sistema, se necesita considerar la forma de asegurar la consistencia del cache con los datos originales cuando las entradas en el mismo son modificadas. Una TLB no es la excepción.

Para una ejecución correcta y segura de un programa, el sistema operativo tiene que asegurarse que cada programa ve su propia memoria y la de nadie más.

Consistencia de la TLB

Context switch: Las direcciones virtuales del viejo proceso ya no son más válidas, y no deben ser válidas, para el nuevo proceso.

De otra forma, el nuevo proceso sería capaz de leer las direcciones del viejo proceso. Frente a un context switch, se necesita descartar el contenido de TLB en cada context switch.

Este approach se denomina flush de TLB. Debido a que este proceso acarrearía una penalidad, los procesadores taguean la TLB de forma tal que la misma contenga el id del proceso que produce cada transacción.

Consistencia de la TLB

Reducción de Permiso: Qué sucede cuando el sistema operativo modifica una entrada en una page table?Normalmente no se provee consistencia por hardware para la TLB; mantener la TLB consistente con la page table es responsabilidad del sistema operativo.

Consistencia de la TLB

TLB shutdown: En un sistema multiprocesador cada uno puede tener cacheada una copia de una transacción en su TLB. Por ende, para seguridad y correctitud, cada vez que una entrada en la page table es modificada, la correspondiente entrada en todas las TLB de los procesadores tiene que ser descartada antes que los cambios tomen efecto.

Típicamente sólo el procesador actual puede invalidar su propia TLB, por ello, para eliminar una entrada en todos los procesadores del sistema, se requiere que el sistema operativo mande una interrupción a cada procesador y pida que esa entrada de la TLB sea eliminada.

Esta es una operación muy costosa y por ende tiene su propio nombre y se denomina TLB shutdown.