Organización y gestión de la memoria

Erica Vidal
Diego Feroldi*

Arquitectura del Computador
Departamento de Ciencias de la Computación
FCEIA-UNR

*Actualizado 10/11/2023 (D. Feroldi, feroldi@fceia.unr.edu.ar)

Índice

Nota general
Este apunte no es para nada una referencia completa del tema **Organización y gestión de memoria** sino que debe ser utilizado como material complementario a lo visto en las clases teóricas.

1. Introducción

Las primeras computadoras contaban con muy poca memoria física. Uno de los primeros sistemas de tiempo compartido trabajaba en una computadora PDP-1 con un tamaño de memoria principal de hasta 144 KB para almacenar el sistema operativo y los programas de usuario. Si un programa necesitaba más espacio de memoria había que usar memoria secundaria, pero era el programador el que debía dividir el programa en varios fragmentos que entraran en memoria. Luego, para ejecutar su programa, debía hacer que se lea el primer fragmento, se ejecutara un tiempo y al terminar se cargara el siguiente fragmento y así sucesivamente sin ayuda del sistema. En 1961 se propuso un método para realizar automáticamente la gestión entre la memoria principal y secundaria. Este método que ahora se conoce como **memoria virtual** se usó por primera vez en los años 70 en varias computadoras.

Observación

- *La **memoria principal**, también llamada **memoria primaria**, es una memoria utilizada para almacenar programas mientras se ejecutan. Típicamente consiste en DRAM en las computadoras actuales.*
- *La **memoria secundaria** es una memoria no volátil utilizada para almacenar programas y datos con mayor capacidad que la memoria principal. Normalmente se utilizan discos magnéticos pero hay otros ejemplos tales como discos de estado sólido, cintas magnéticas, CDs, DVDs, memorias USB, etc.*

2. Memoria virtual

Para administrar la memoria de manera más eficiente y con menos errores, los sistemas modernos proporcionan una abstracción de la memoria principal conocida como **memoria virtual**. La memoria virtual es un mecanismo eficiente que proporciona a cada proceso un espacio de direcciones grande, uniforme y privado. La memoria virtual proporciona tres capacidades importantes:

1. Utiliza la memoria principal de manera eficiente manteniendo solo las áreas activas en la memoria principal y transfiriendo datos entre el disco y la memoria según sea necesario.
2. Simplifica la gestión de la memoria al proporcionar a cada proceso un espacio de direcciones uniforme.
3. Protege el espacio de direcciones.

En base a lo anterior, la memoria virtual se puede definir como una técnica que utiliza la memoria principal como un "*buffer*" de almacenamiento para el almacenamiento secundario. La idea consiste en separar los conceptos de **espacio de direcciones** y **posiciones de memoria**.

Ejemplo

Consideremos una computadora de los años 60 con una memoria de 4 KB y palabras de 16 bits. Un programa para esta computadora puede direccionar 65536 (2^{16}) palabras de memoria. Eso es porque existen 65536 direcciones de 16 bits, cada una de las cuales corresponde a una palabra de memoria diferente.

Entonces, el espacio de direcciones (virtual) de esta computadora corresponde con el conjunto de números $0, 1, 2, \ldots, 65535$, porque ese es el conjunto posible de direcciones virtuales. Por otro lado, tenemos las direcciones de memoria física cuya cantidad depende de la memoria física de la máquina. Si tenemos 4 KB de memoria RAM podemos almacenar 2048 palabras de 16 bits, con direcciones de memoria físicas de la 0 a la 2047.

Observación

El número de palabras direccionables virtualmente depende únicamente del número de bits que tiene una dirección y es independiente de la cantidad de memoria física que tiene la máquina.

Antes de que existiera el concepto de memoria virtual no era necesario distinguir entre el espacio de direcciones y las direcciones de memoria, porque había una correspondencia uno a uno entre ellos. La idea de separar el espacio de direcciones y las direcciones de memoria es la siguiente. En cualquier momento se puede acceder a una posición de memoria, pero no es necesario utilizar directamente la dirección de memoria física. Se podría utilizar una dirección de memoria virtual que tenga una correspondencia con una dirección de memoria física. En otras palabras, se puede definir un "mapeo" del espacio de direcciones virtuales a las direcciones de memoria reales (físicas).

Observaciones

- *La motivación principal para utilizar memoria virtual es permitir el uso compartido eficiente y seguro de la memoria entre varios programas.*
- *La segunda motivación es permitir que un solo programa de usuario exceda el tamaño de la memoria primaria. Estas cuestiones se irán profundizando a lo largo de este apunte.*

En la Fig. 1 se ve un esquema de un mapeo en que las direcciones lógica (o virtuales) 4096 a 8191 se hacen corresponder con las direcciones 0 a 4095 de la memoria principal. Todas las implementaciones de memoria virtual, con excepción de los emuladores, requieren soporte en hardware. Esto se hace comúnmente mediante la unidad de administración de memoria (**MMU**, *Memory Management Unit*) construida actualmente dentro de la CPU.

Observación

*Una **dirección lógica** es una dirección de memoria tal como la ve una aplicación. En la instrucción* `movq 0x404028, %rax` *hay una dirección lógica:* `0x404028`*. Un programador tiene la ilusión de que él es el único usuario de la memoria. Cualquiera que sea la celda de memoria a la que se dirija, nunca ve datos o instrucciones de otros programas que se ejecutan con el suyo en paralelo. Sin embargo, la memoria física contiene varios programas a la vez. De hecho,*

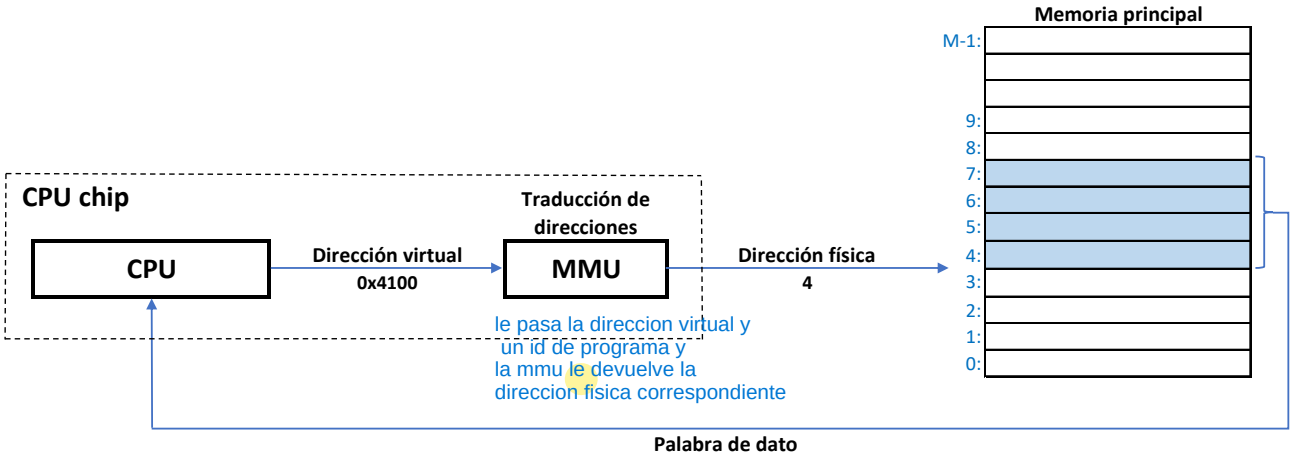


Figura 1: Esquema básico de memoria virtual.

otro programa podría ejecutar la misma instrucción pero no accedería a las mismas celdas en memoria. Esto es debido al mecanismo de memoria virtual que iremos viendo a continuación. En este contexto, ***dirección virtual*** *es sinónimo de* ***dirección lógica****.*

3. Segmentación

Un método para implementar memoria virtual es el denominado **segmentación**, el cual se basa en proporcionar muchos espacios de direcciones totalmente independientes, llamados **segmentos**. Los segmentos son una forma de organizar y administrar el espacio de direcciones de la memoria virtual. Representan divisiones lógicas dentro de ese espacio de direcciones y cada segmento puede tener sus propios permisos, tamaño y características. Así, los segmentos se definen como bloques de datos de diferentes longitudes que residen en la memoria secundaria. Un segmento completo puede copiarse temporalmente en una región disponible de la memoria principal. Cada segmento consiste en una sucesión lineal de direcciones, de la 0 a alguna dirección máxima y por lo tanto la longitud de cada segmento puede ser cualquiera desde 0 hasta el máximo permitido. Diferentes segmentos pueden tener diferentes longitudes, y generalmente así es. Además, la longitud de los segmentos podría cambiar durante la ejecución. Por ejemplo, la longitud de un segmento de pila podría incrementarse cada vez que algo se mete en la pila y reducirse cada vez que algo se "desapila".

Observación

En segmentación ocurre un fenómeno denominado ***fragmentación externa****, el cual ocurre cuando hay suficiente espacio de memoria total para satisfacer una solicitud, pero los espacios disponibles no son contiguos. Es decir, el almacenamiento queda fragmentado en una gran cantidad de pequeños "huecos". Este problema de fragmentación puede ser grave. Una técnica para superar la fragmentación externa es la* ***compactación****: de vez en cuando, el sistema operativo mueve los procesos para que sean contiguos y para que toda la memoria libre esté junta en un bloque.*

En segmentación una dirección lógica se considera que está formada por dos partes: un número de segmento, que se asocia a una dirección física, y un desplazamiento de segmento. Una consecuencia de que los segmentos sean de tamaño desigual es que no existe una relación

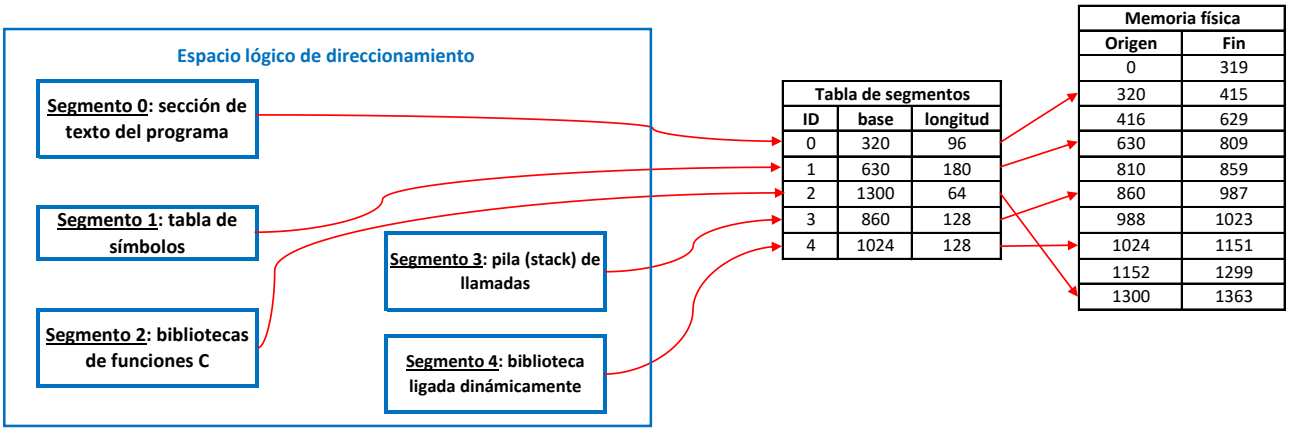


Figura 2: Esquema de segmentación (ID es el identificador del segmento).

simple entre las direcciones lógicas y las direcciones físicas. Un esquema de segmentación simple hace uso de una tabla de segmentos para cada proceso[1] e información sobre los bloques libres de memoria principal. Cada entrada de la tabla de segmentos tiene que dar la dirección inicial en la memoria principal del segmento correspondiente. La entrada también debe proporcionar la longitud del segmento, para garantizar que no se utilicen direcciones no válidas. Cuando un proceso ingresa al estado "En ejecución", la dirección de su tabla de segmentos se carga en un registro especial utilizado por el hardware de administración de memoria[2].

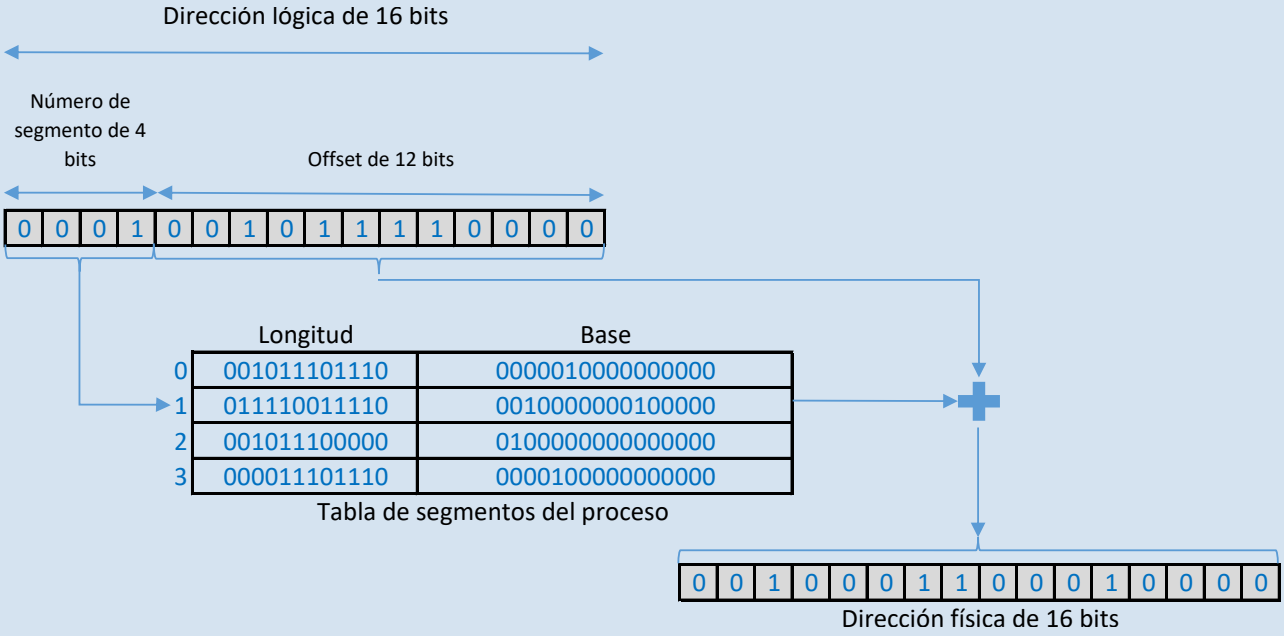
La Fig. 2 muestra de manera esquemática cómo funciona segmentación. En este ejemplo vemos 5 segmentos, su correspondiente tabla de segmentos y cómo se ubican estos segmento en la memoria física en base a la información de la tabla.

Ejemplo

Consideremos una dirección de $n+m$ bits, donde los n bits más a la izquierda son el número de segmento y los m bits más a la derecha son el desplazamiento. En nuestro ejemplo, $n = 4$ y $m = 12$. Por lo tanto, el tamaño máximo del segmento es $2^{12} = 4096$. Este ejemplo se ilustra en la siguiente figura:

[1] Un proceso, en informática, puede entenderse informalmente como una instancia de un programa en ejecución. Este tema se verá en detalle en Sistemas Operativos II.

[2] Este mecanismo depende de la arquitectura.



Debemos realizar los siguientes pasos para la traducción de la dirección:

- *Extraer el número de segmento como los n bits más a la izquierda de la dirección lógica.*
- *Utilizar el número de segmento como índice en la tabla de segmentos del proceso para encontrar la dirección física inicial del segmento. En este ejemplo hay 4 segmentos.*
- *Comparar el desplazamiento, expresado en los m bits más a la derecha, con la longitud del segmento. Si el desplazamiento es mayor o igual que la longitud, la dirección no es válida.*
- *Finalmente, la dirección física deseada es la suma de la dirección física inicial del segmento más el desplazamiento.*

En nuestro ejemplo tenemos la dirección lógica `0x12f0`*, que de acuerdo a lo anterior corresponde al segmento número 1 y desplazamiento* `0x2f0`*. Supongamos que este segmento reside en la memoria principal y comienza en la dirección física* `0x2020`*. Entonces, la dirección física es* `0x2020 + 0x2f0 = 0x2310`*.*

4. Paginación

En el método de paginación, el espacio de direcciones virtuales se divide en **páginas** (conjuntos de bytes) de tamaño uniforme. El tamaño de la página siempre es una potencia de 2. El espacio de direcciones físico se divide en fragmentos de la misma manera y cada uno es del mismo tamaño de una página. Los fragmentos de la RAM en la que entra una página se llama **marco de página física**. En la memoria virtual, la dirección se divide en un **número de página virtual** y en un **desplazamiento de página**. La cantidad de bits del campo de la dirección correspondiente al desplazamiento de la página determina el **tamaño de la página**. En la Fig. 3 vemos un esquema de cómo se compone la dirección virtual para un caso con direcciones virtuales de 32 bits y tamaño de página de 4 KB (4096 bytes = 2^{12} bytes).

El procesador genera una **dirección virtual**, la cual se convierte mediante una combinación de hardware y software en una dirección física, la cual a su vez puede ser utilizada para

primer paso: entrar a la tabla de paginas con esta direccion. luego con el numero de pagina (id de la pagina) obtenemos la direccion fisica concatenandole el desplazamiento (offset)

por lo tanto requerimos 2 entradas a memoria, la primera para entrar a la tabla de paginas la segunda para entrar a la direccion fisica

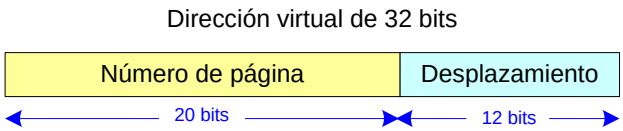


Figura 3: Esquema de dirección virtual utilizando paginación.

acceder a la memoria principal. La Fig. 4 muestra la memoria direccionada de forma virtual con páginas asignadas a la memoria principal. Este proceso se denomina conversión de direcciones o **traducción de direcciones**.

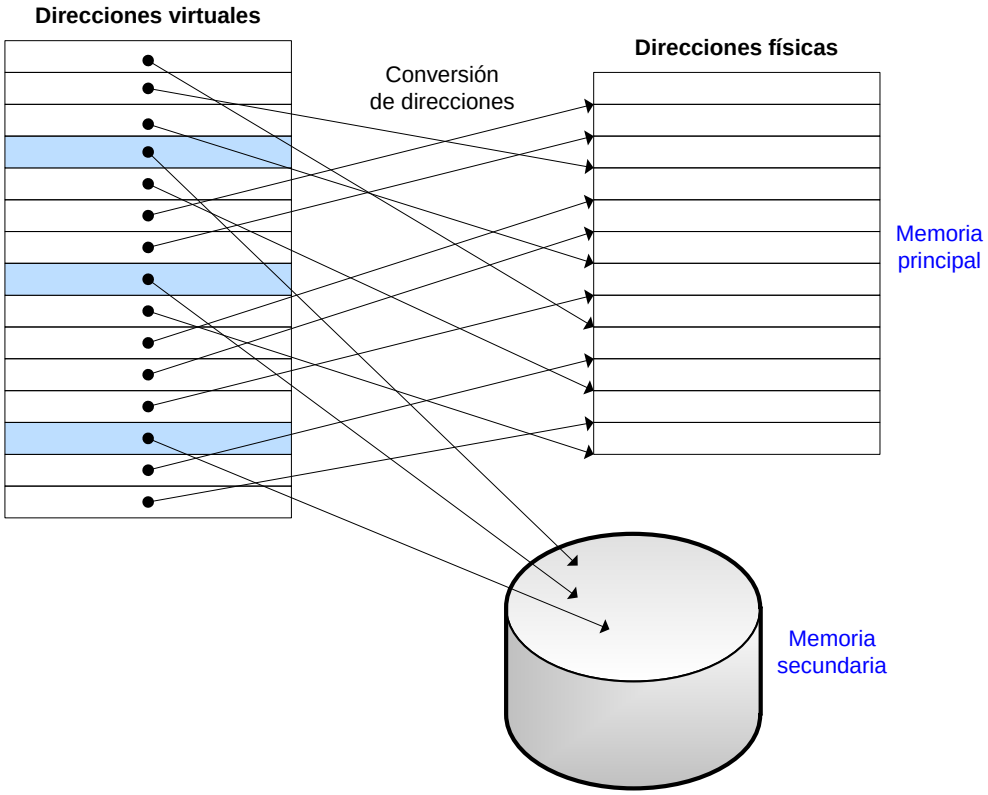


Figura 4: Conversión de direcciones.

5. La tabla de paginación

El procesador usa una tabla de páginas para traducir direcciones virtuales a direcciones físicas. La tabla de páginas (o tabla de paginación) contiene una entrada para cada página virtual. Esta entrada contiene al menos un número de página física y un **bit de validez** (V). Si $V = 1$, la página virtual se asigna a la página física especificada en la entrada. De lo contrario, la página virtual se encuentra en la memoria secundaria.

Debido a que la tabla de páginas es muy grande, se almacena en la memoria física. Supongamos por ahora que se almacena como un arreglo contiguo, como se muestra en la Fig. 5. La tabla de páginas está indexada con el número de página virtual (VPN). Por ejemplo, la entrada 5 especifica que la página virtual 5 se asigna a la página física 1. La entrada 6 no es válida ($V = 0$), por lo que la página virtual 6 se encuentra en la memoria secundaria.

Para realizar una operación de memoria, el procesador primero debe traducir la dirección virtual a una dirección física y luego acceder a los datos en esa dirección física. Luego, el procesador lee esta entrada de la tabla de páginas de la memoria física para obtener el número

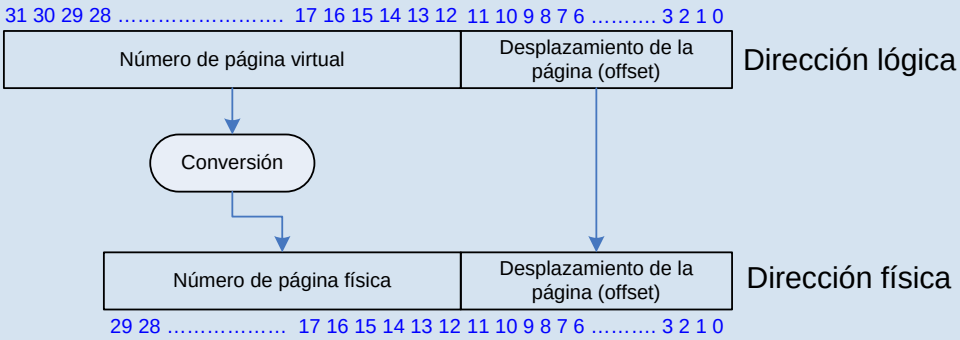
Número de página virtual	V	Dirección de página física
0x7ffff	0	
0x7fffe	0	
0x7fffd	0	
0x7fffc	0	
0x7fffb	1	0x0001
0x7fffa	0	
⋮		⋮
0x0004	0	
0x0003	1	0x7ffe
0x0002	0	
0x0001	0	
0x0000	1	0x0000

Figura 5: Tabla de páginas.

de página física. Si la entrada es válida, junta este número de página física con el desplazamiento de página para crear la dirección física. Finalmente, lee o escribe datos en esta dirección física. Debido a que la tabla de páginas se almacena en la memoria física, cada carga o almacenamiento implica dos accesos a la memoria física.

Ejemplo

Supongamos que tenemos direcciones virtuales de 32 bits y páginas de 4 KB = 4096 bytes = 2^{12} bytes, por lo tanto la cantidad de bits para el desplazamiento dentro de la página es igual a 12. Además, supongamos que tenemos una memoria física de 1 GB = 2^{30} bytes. Por lo tanto, la cantidad de bits para especificar el número de marco de página es 30 - 12; eso es 2^{18} páginas físicas. La memoria virtual para cada proceso cubre 4 GB (direcciones de memoria de 32 bits). Para especificar el número de página virtual la cantidad de bits es 32 – 12 = 20, porque 12 es la cantidad de bits para el desplazamiento. El siguiente esquema ilustra este ejemplo:



*Luego, la conversión (o traducción) entre el número de página lógica (o virtual) y el número de página física se realiza utilizando la **tabla de paginación** (o **tabla de páginas**). Supongamos la siguiente tabla de paginación (en realidad, solo mostramos un fragmento por cuestión de espacio):*

	V	*Número de página físico*
0xfffff0	*0*	*0x3aa00*
0xfffff1	*1*	*0x31800*
0xfffff2	*0*	*0x320a0*
0xfffff3	*0*	*0x3021f*
0xfffff4	*0*	*0x31c09*
0xfffff5	*1*	*0x36020*
0xfffff6	*0*	*0x32056*
0xfffff7	*1*	*0x3ff00*
0xfffff8	*1*	*0x3ff0c*
0xfffff9	*0*	*0x3fe00*

Veamos ahora cómo hacer una traducción de direcciones. Es decir, dada una dirección virtual determinar a cuál dirección física corresponde. Supongamos que partimos de la dirección virtual `0xffff5a06`*. Si tenemos páginas de 4 kB, los 12 bits menos significativos corresponden al offset. En este caso,* `0xa06`*. Luego, los restantes bits determinan el número de página virtual. En este caso,* `0xffff5`*. Con este número de página virtual podemos entrar a la tabla de páginas (o tabla de paginación) anterior y obtener el número de página física. En este caso,* `0x36020`*. Finalmente, la dirección física se obtiene concatenando el número de página física y el offset (o desplazamiento). Por lo tanto, la dirección física es* `0x36020a06`*.*

Observación

Como se ve en el ejemplo anterior, el número de bits para especificar una dirección física puede diferir del número de bits para especificar una dirección virtual. Por el contrario, el número de bits para especificar el desplazamiento de la dirección virtual y física es el mismo.

Observación

¿Por qué el tamaño de página es potencia de 2? Recordando que la paginación se implementa dividiendo una dirección en un número de página y un número de desplazamiento, y debido a que cada posición de bit representa una potencia de 2, da como resultado un tamaño de página que es una potencia de 2.

Por lo tanto, la traducción de direcciones (o conversión de direcciones) es un proceso por el cual una dirección virtual se convierte en una dirección que se utiliza para acceder a la memoria principal. Sin embargo, también es posible que una página virtual se encuentre ausente de la memoria principal y no se le haya asignado una dirección física, por lo que se encuentra en el disco. Se denomina **fallo de página** al suceso que ocurre cuando se quiere acceder a una página que no se encuentra en la memoria principal. El sistema operativo maneja la falla seleccionando una página "víctima", intercambiando la página víctima si está sucia, intercambiando la página nueva y actualizando la tabla de páginas. Este mecanismo se denomina ***swapping***.

Estas capacidades son proporcionadas por una combinación de software del sistema operativo, hardware de traducción de direcciones en la MMU (unidad de administración de memoria) y una estructura de datos almacenada en la memoria física conocida como **tabla de páginas** que asigna páginas virtuales a páginas físicas. El hardware de traducción de direcciones lee la tabla de páginas cada vez que convierte una dirección virtual en una dirección física. El sistema operativo es responsable de mantener el contenido de la tabla de páginas y transferir páginas de un lado a otro entre el disco y la DRAM.

6. Tamaño de página

El tamaño de la página es a menudo un parámetro que puede elegir el sistema operativo entre varios tamaños soportables por la arquitectura. Determinar el mejor tamaño de página requiere equilibrar varios factores en competencia. Como resultado, no existe un óptimo general. Por un lado, un determinado proceso no llenará un número entero de páginas. En promedio, la mitad de la página final estará vacía. El espacio adicional en esa página se desperdicia. Este desperdicio se llama **fragmentación interna**. Con n procesos y un tamaño de página de p bytes, se desperdiciarán $np/2$ bytes debido a la fragmentación interna. Esto aboga por un tamaño de página pequeño. Por otro lado, páginas pequeñas significan que los programas necesitarán muchas páginas, por lo tanto, una tabla de páginas grande. Este tema se verá en la Sección 5.

7. La TLB

Teniendo en cuenta que cada carga o almacenamiento implica dos accesos a la memoria física, esto tendría un impacto severo en el rendimiento. Afortunadamente, los accesos a la tabla de páginas tienen una gran localidad temporal. La localidad temporal y espacial de los accesos a los datos implica que es probable que muchas operaciones de memoria hagan referencia a una página que ya fue traducida recientemente. Por lo tanto, si la MMU recuerda la última entrada de la tabla de páginas que leyó, probablemente pueda reutilizar esta traducción sin volver a leer la tabla de páginas. En general, el procesador puede mantener las últimas entradas de la tabla de páginas en una pequeña tabla llamada ***Translation Lookaside Buffer*** **(TLB)** dentro de la memoria caché. El procesador primero trata encontrar la traducción en la TLB antes de tener que acceder a la tabla de páginas en la memoria física. En los programas reales, la gran mayoría de los accesos se realizan utilizando la TLB, lo que evita las lecturas de la tabla de páginas desde la memoria física que consumen mucho tiempo.

bufferde busqueda de traducciones

Una TLB se organiza como una memoria caché totalmente asociativa y, por lo general, contiene de 16 a 512 entradas. Cada entrada de TLB contiene un número de página virtual, su correspondiente número de página física y banderas. Se accede a la TLB utilizando el número de página virtual. Si la TLB "acierta", devuelve el número de página física correspondiente. De lo contrario, el procesador debe leer la tabla de páginas en la memoria física. La TLB está diseñada para ser lo suficientemente pequeña como para poder acceder a ella en menos de un ciclo. Aun así, las TLB suelen tener una tasa de aciertos superior al 99 %. La TLB reduce el número de accesos a la memoria necesarios para la mayoría de las instrucciones de carga o almacenamiento de dos a uno (y más si hay paginación multinivel).

Ejemplo

Continuando con el ejemplo anterior, supongamos ahora que además de la tabla de páginas tenemos una TLB con 4 entradas como se muestra a continuación:

V	***Número de página virtual***	***Número de página físico***
0	*0xefff0*	*0x0aa00*
1	*0xfffff5*	*0x36020*
1	*0x0fff4*	*0x4800a*
0	*0xafff3*	*0x7a001*

Supongamos que queremos traducir la dirección virtual `0xffff5a06` *como en el ejemplo anterior. La TLB recibe el número de página virtual de la dirección entrante,* `0xffff5`*, y lo compara con el número de página virtual de cada entrada. La entrada 1 coincide y es válida, por lo*

que la traducción se puede realizar y no es necesario acceder a memoria. La dirección física traducida es el número de página física de la entrada coincidente, `0x36020`*, concatenado con el desplazamiento de página de la dirección virtual. Como siempre, el desplazamiento de página no requiere traducción. Al contrario, la solicitud de la dirección virtual* `0xffff8` *no se encuentra en la TLB. Por lo tanto, la solicitud se envía a la tabla de páginas para su traducción y además luego se carga la traducción en la TLB.*

8. Protección de memoria

Hasta ahora, nos hemos centrado en el uso de la memoria virtual para proporcionar una memoria grande, rápida y económica. Una razón igualmente importante para usar la memoria virtual es brindar protección entre programas que se ejecutan simultáneamente. Las computadoras modernas normalmente ejecutan varios programas o procesos al mismo tiempo. Todos los programas están simultáneamente presentes en la memoria física. En un sistema informático bien diseñado, los programas deben estar protegidos entre sí para que ningún programa pueda interferir en otro programa. Específicamente, ningún programa debería poder acceder a la memoria de otro programa sin permiso. Esto se llama **protección de la memoria**.

Los sistemas de memoria virtual brindan protección a la memoria al otorgar a cada programa su propio **espacio de direcciones virtuales**. Cada programa puede usar tanta memoria como quiera en ese espacio de direcciones virtuales, pero puede que solo una parte del espacio de direcciones virtuales esté en la memoria física en un momento dado. Cada programa puede utilizar todo su espacio de direcciones virtuales sin tener que preocuparse por la ubicación física de otros programas. Sin embargo, un programa puede acceder solo a aquellas páginas físicas que están asignadas en su tabla de páginas. De esta forma, un programa no puede acceder accidental o maliciosamente a las páginas físicas de otro programa, porque no están mapeadas en su tabla de páginas. En algunos casos, varios programas acceden a instrucciones o datos comunes. El sistema operativo agrega bits de control a cada entrada de la tabla de páginas para determinar qué programas, si los hay, pueden escribir en las páginas físicas compartidas.

9. Tablas de páginas multinivel

Las tablas de páginas pueden ocupar una gran cantidad de memoria física. Por ejemplo, la tabla de páginas para una memoria virtual de 32 bits con páginas de 4 KB necesitaría 2^{32} bytes/2^{12} bytes = 2^{20} entradas. Si cada entrada tiene 4 bytes, la tabla de páginas ocupa $2^{20} \times 2^2$ bytes = 2^{22} bytes = 4 MB.

Para conservar la memoria física, las tablas de páginas se pueden dividir en múltiples niveles. Hoy en día, las estructuras de tablas de páginas más complejas comprenden cuatro niveles[3]. La tabla de páginas de primer nivel siempre se mantiene en la memoria física e indica dónde se almacenan las pequeñas tablas de páginas de segundo nivel en la memoria virtual. En un sistema con tabla de niveles, cada una de las tablas de páginas de segundo nivel contiene las traducciones de un rango de páginas virtuales. Si un rango particular de traducciones no se usa activamente, la tabla de páginas de segundo nivel correspondiente se puede paginar en el disco duro para que no se desperdicie memoria física.

En una tabla de páginas de dos niveles, el número de página virtual se divide en dos partes: el **número de la tabla de páginas** y el **desplazamiento de la tabla de páginas**, como se muestra en la Fig. 6. El número de tabla de páginas indexa la tabla de páginas de primer

[3]Existen procesadores con 5 niveles que hacen uso de una extensión (*Intel 5-level paging*).

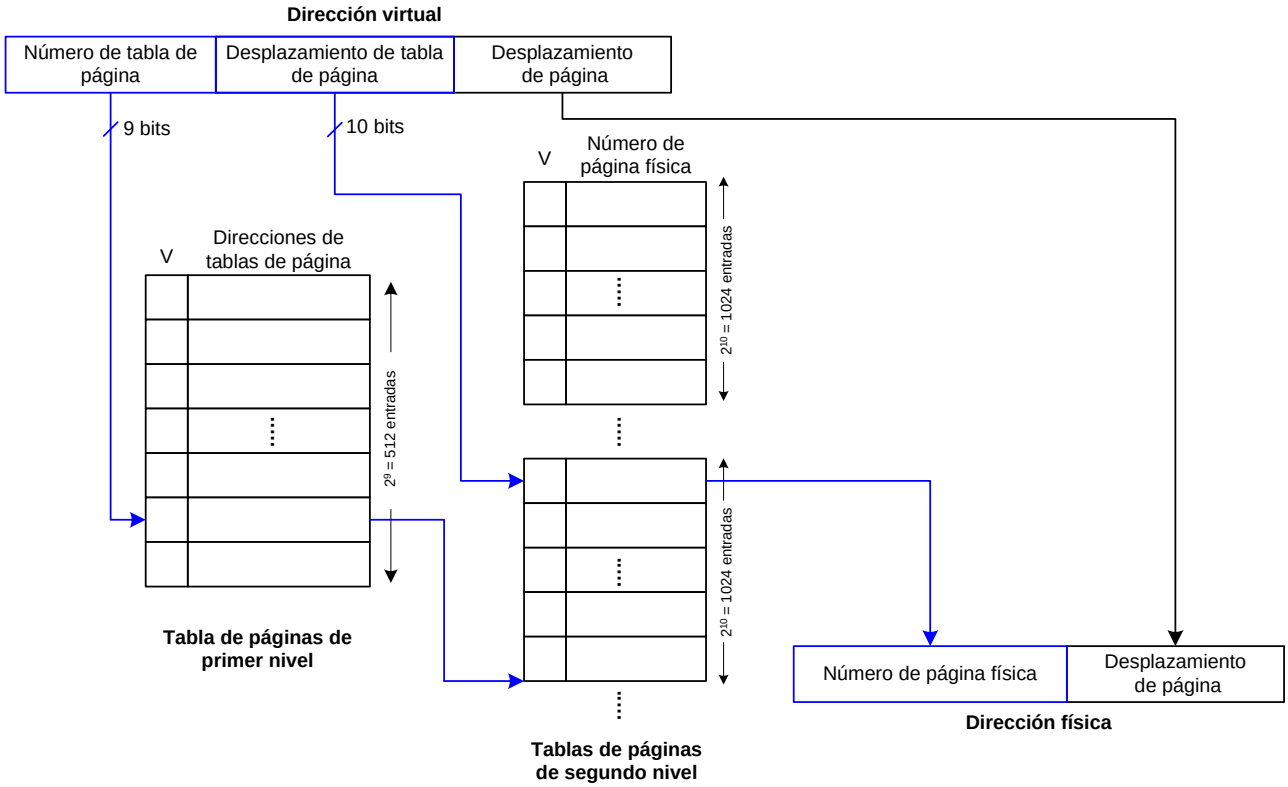


Figura 6: Ejemplo de paginación con tabla de páginas de dos niveles.

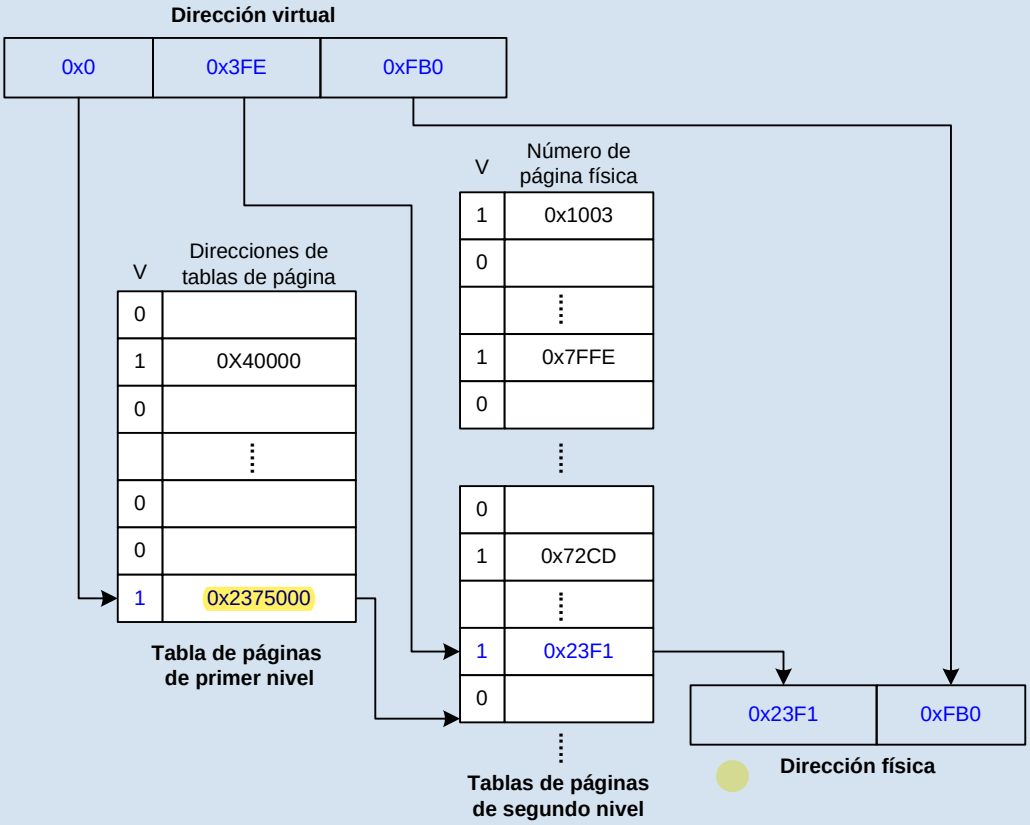
nivel, que debe residir en la memoria física. La entrada de la tabla de páginas de primer nivel proporciona la dirección base de la tabla de páginas de segundo nivel o indica que debe obtenerse del disco duro cuando $V = 0$. El desplazamiento de la tabla de páginas indexa la tabla de páginas de segundo nivel. Los 12 bits restantes de la dirección virtual son el desplazamiento de página, suponiendo un tamaño de página de 4 KB (2^{12} bytes).

El número de página virtual de 19 bits en la Fig. 6 se divide en 9 y 10 bits, para indicar el número de la tabla de páginas y el desplazamiento de la tabla de páginas, respectivamente. Así, la tabla de páginas de primer nivel tiene 2^9=512 entradas. Cada una de estas 512 entradas apunta a una tabla de segundo nivel. A su vez, cada una de las tablas de segundo nivel tiene $2^{10} = 1024$ entradas. Si cada una de las entradas de la tabla de páginas de primer y segundo nivel es de 32 bits (4 bytes) y solo dos tablas de páginas de segundo nivel están presentes en la memoria física a la vez, la tabla de páginas jerárquica usa solo $(512 \times 4$ bytes$) + 2 \times (1024 \times 4$ bytes$) =$ 10 KB de memoria física. Por lo tanto, la tabla de páginas de dos niveles requiere una fracción de la memoria física necesaria para almacenar la tabla de páginas completa (4 MB). Sin embargo, el inconveniente de una tabla de páginas de dos niveles es que agrega otro acceso a la memoria para la traducción cuando "falla" la TLB.

Ejemplo

La siguiente figura muestra los posibles contenidos de una tabla de páginas de dos niveles como la mostrada en la Fig. 6. Se muestra el contenido de una sola tabla de páginas de segundo nivel. Usando esta tabla de páginas de dos niveles, se describe cómo se realiza un acceso a la dirección virtual `0x003FEFB0`.

el gral el algoritmo para acceder es:
indexar primeros nueve bits en tabla de pagina primer nivel
acceder tabla de pagina segundo nivel con la direccion fisica encontrada en la de primer nivel
indexar segundos 10 bits en la tabla de pagina de segundo nivel encontrada
guardar la direccion de memoria incompleta que guarda la tabla de segundo nivel en la posicion dada
concatenar con los ultimos 12 bits para completar la direccion del dato buscado
end.



Como ya hemos visto, solamente el número de página virtual requiere traducción. Los nueve bits más significativos de la dirección virtual, `0x0`*, dan el número de la tabla de páginas. Es decir, el índice de la tabla de páginas de primer nivel. La tabla de páginas de primer nivel en la entrada* `0x0` *indica que la tabla de páginas de segundo nivel reside en la memoria (V=1) y su dirección física es* `0x2375000`*. Los siguientes diez bits de la dirección virtual,* `0x3FE`*, son el desplazamiento de la tabla de páginas, lo que proporciona el índice en la tabla de páginas de segundo nivel. La entrada* `0` *está en la parte inferior de la tabla de páginas de segundo nivel y la entrada* `0x3FF` *está en la parte superior. La entrada* `0x3FE` *en la tabla de páginas de segundo nivel indica que la página virtual reside en la memoria física (V=1) y que el número de página física es* `0x23F1`*. El número de página física se concatena con el desplazamiento de página para formar la dirección física, la cual resulta* `0x23F1FB0`*.*

10. Caso de estudio: Intel Core i7

Un caso de estudio real interesante para estudiar los mecanismos de memoria virtual es el procesador Intel Core i7 ejecutando Linux. Aunque la arquitectura x86-64 permite espacios completos de direcciones físicas y virtuales de 64 bits, las implementaciones actuales de Intel Core i7 (y las del futuro previsible) admiten un espacio de direcciones virtuales de 48 bits (256 TB) y un espacio de direcciones físicas de 52 bits (4 PB), junto con un modo de compatibilidad que admite espacios de direcciones físicas y virtuales de 32 bits (4 GB).

La Fig. 7 muestra los aspectos más destacados del sistema de memoria del procesador Intel Core i7. El chip incluye cuatro núcleos[4], una gran caché L3 compartida por todos los núcleos y un controlador de memoria DDR3, entre otros módulos. Cada núcleo contiene una jerarquía de

[4]Intel Core i7 es una familia de procesadores con diferentes modelos. El modelo mostrado en la Fig. 7 tiene 4 núcleos pero hay modelos con mayor cantidad de núcleos.

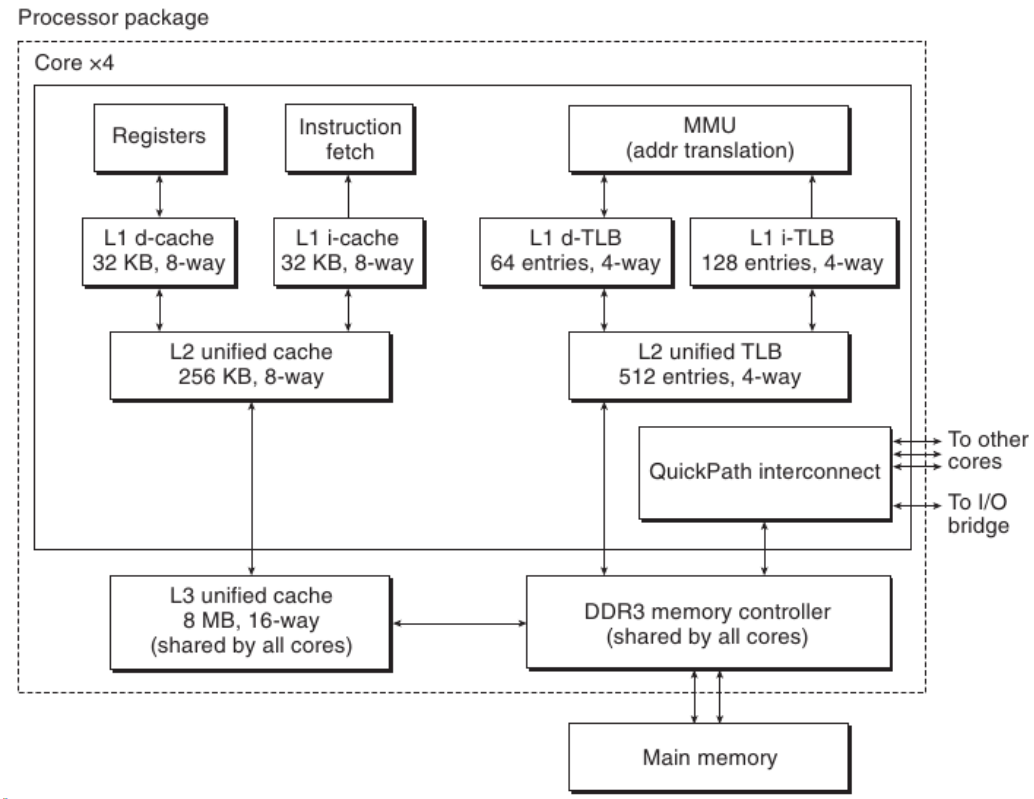


Figura 7: Sistema de memoria del procesador Intel Core i7 (fuente: [1]).

TLB y una jerarquía de cachés de datos e instrucciones. La TLB de cada núcleo es asociativa de 4 vías. L1 y L2 son asociativas de 8 vías, y L3 es asociativa de 16 vías. El tamaño de la página se puede configurar en el momento del inicio como 4 KB o 4 MB. Linux usa páginas de 4 KB.

El procesador Intel Core i7 utiliza una jerarquía de tablas de páginas de cuatro niveles. Cada proceso tiene su propia jerarquía de tablas de páginas privadas. Cuando se ejecuta un proceso de Linux, las tablas de páginas asociadas con las páginas asignadas residen todas en la memoria, aunque el procesador Intel Core i7 permite que estas tablas de páginas se intercambien y se desconecten. El registro de control CR3 contiene la dirección física del comienzo de la tabla de páginas de nivel 1 (L1). El valor de CR3 es parte de cada contexto de proceso y se restaura durante cada cambio de contexto[5]. Las direcciones virtuales son de 48 bits: 36 bits para el número de página virtual (9 bits para cada nivel) y 12 bits para el offset, mientras las direcciones físicas son de 52 bits (40 bits para el número de página física y 12 bits para el offset). El procedimiento de traducción de direcciones para el Intel Core i7 se puede ver en la Fig. 8.

El número de dirección virtual de 36 bits se divide en cuatro fragmentos de 9 bits, cada uno de los cuales se utiliza como un desplazamiento en una tabla de páginas. El registro CR3 contiene la dirección física de la tabla de páginas de primer nivel (L1 PT). El primer fragmento del número de la dirección virtual (VPN 1) proporciona un desplazamiento a una entrada de primer nivel (L1 PTE), que contiene la dirección base de la tabla de páginas de nivel 2 (L2 PT). El segundo fragmento (VPN 2) proporciona un desplazamiento a L2 PTE, y así sucesivamente.

El proceso descripto es lento, dado que involucra varios accesos a memoria. Sin embargo, gracias a la TLB se suele acceder a memoria en páginas ya traducidas y memorizadas. Por lo tanto, solamente se debe agregar el desplazamiento al número de página ya traducida, lo cual es muy rápido. Como ya se mencionó, la TLB es una memoria caché asociativa que rápidamente

[5]Este tema excede el alcance de esta asignatura y se verá en detalle en sistema Operativos II.

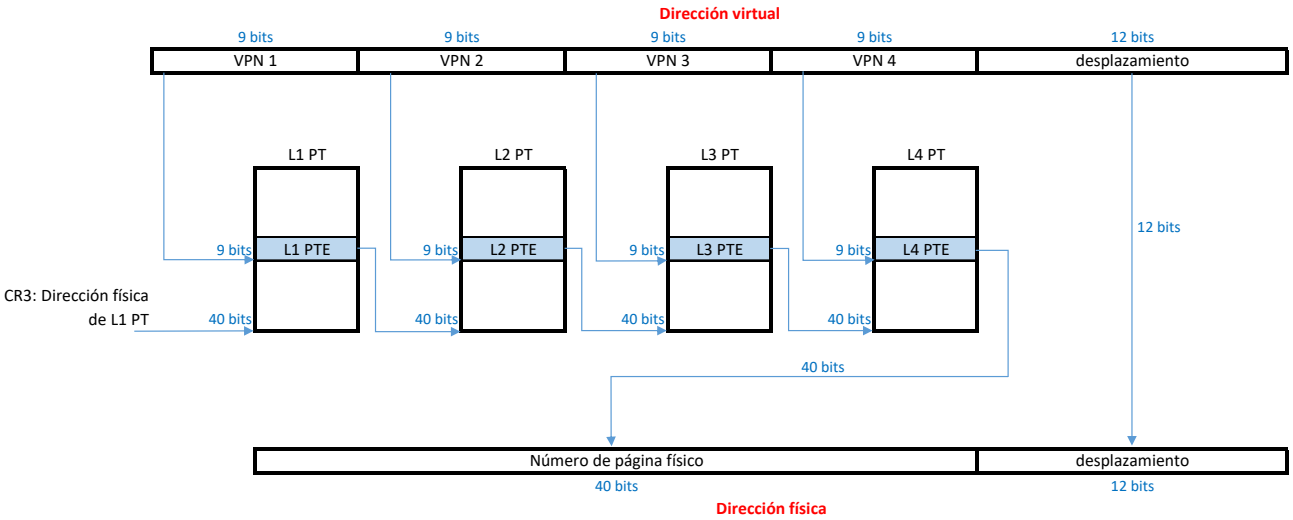


Figura 8: Esquema de traducción de direcciones virtuales del Intel Core I7 con tabla de páginas multinivel.

nos proporciona la información guardada.

A. Unidades de memoria

Un byte (B) se define como un conjunto de 8 bits. A su vez, hay múltiplos de esta cantidad utilizando diferentes prefijos como se muestra en la siguiente tabla:

Prefijo	**Unidad**	**Símbolo**	**Cantidad**
Kilo	Kilobyte	KB	1 KB = 2^{10} bytes = 1024 bytes $\approx 10^3$ bytes
Mega	Megabyte	MB	1 MB = 2^{20} bytes = 1024 Kilobytes $\approx 10^6$ bytes
Giga	Gigabyte	GB	1 GB = 2^{30} bytes = 1024 Megabytes $\approx 10^9$ bytes
Tera	Terabyte	TB	1 TB = 2^{40} bytes = 1024 Gigabytes $\approx 10^{12}$ bytes
Peta	Petabyte	PB	1 PB = 2^{50} bytes = 1024 Terabytes $\approx 10^{15}$ bytes

Referencias

[1] Randal E Bryant, O'Hallaron David Richard, and O'Hallaron David Richard. *Computer systems: a programmer's perspective*. Pearson Education Limited, third edition, 2015.

[2] Ulrich Drepper. What every programmer should know about memory. *Red Hat, Inc*, 11(2007):2007, 2007.

[3] David Money Harris and Sarah L. Harris. *Digital Design and Computer Architecture, Second Edition*. Morgan Kaufmann, 2012.

[4] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[5] David A Patterson, John L Hennessy, and Peter J Ashenden. *Computer Organization and Design: The Hardware/Software Interface*. Elsevier Science Limited, fifth edition edition, 2014.

[6] William Stallings and Goutam Kumar Paul. *Operating systems: internals and design principles*, volume 9. Pearson New York, 2012.

[7] Andrew S Tanenbaum. *Organización de computadoras: un enfoque estructurado.* Pearson educación, 2000.

[8] Gunnar Wolf, Esteban Ruiz, Federico Bergero, and Erwin Meza. *Fundamentos de sistemas operativos.* Universidad Nacional Autónoma de México, Instituto de Investigaciones Económicas: Facultad de Ingeniería, 2015.

[9] Igor Zhirkov. *Low-Level Programming: C, Assembly, and Program Execution on Intel 64 Architecture.* Apress, 2017.