Organización y gestión de la memoria

Diego Feroldi*
Erica Vidal
Arquitectura del Computador
Departamento de Ciencias de la Computación
FCEIA-UNR

*Actualizado 07/05/2025 (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**. A través del mecanismo de memoria virtual, cada proceso accede a su propio **espacio de direcciones virtuales** privado (también conocido simplemente como **espacio de direcciones**), que el sistema operativo mapea de algún modo en la memoria física de la máquina. Una referencia de memoria dentro de un programa en ejecución no afecta el espacio de direcciones de otros procesos (ni del propio sistema operativo); desde la perspectiva del programa en ejecución, parece tener toda la memoria física para sí mismo. Sin embargo, la realidad es que la memoria física es un recurso compartido, gestionado por el sistema operativo.

La memoria virtual es un mecanismo eficiente que ofrece a cada proceso un espacio de direcciones amplio, uniforme y privado. Este mecanismo proporciona tres capacidades clave:

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 contamos con 4 KB de memoria RAM, podemos almacenar 4096 palabras de 16 bits, utilizando direcciones de memoria físicas que van desde 0 hasta 4095.*

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 utiliza una dirección de memoria virtual que tiene una correspondencia con una dirección de memoria física. En otras palabras, se define 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.

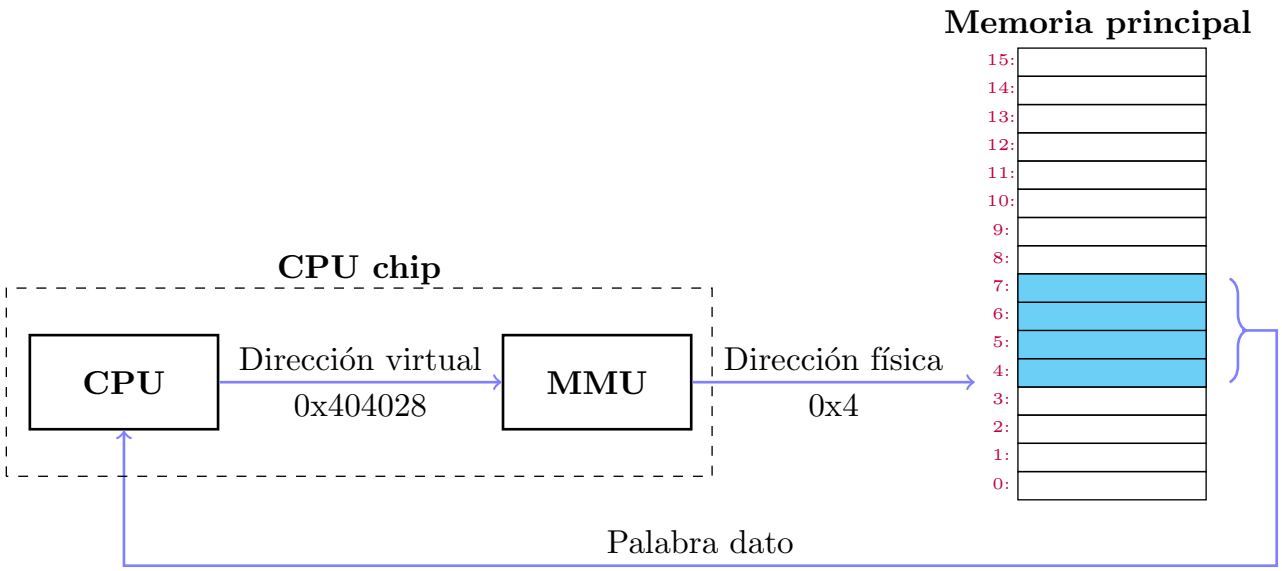


Figura 1: Esquema básico de memoria virtual.

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, 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. Espacio de direcciones virtuales

El **espacio de direcciones virtual** es una abstracción de la memoria física que representa la visión del programa en ejecución sobre la memoria del sistema. Comprender esta abstracción fundamental del sistema operativo es esencial para entender cómo se virtualiza la memoria.

El espacio de direcciones de un proceso incluye todo el estado de memoria del programa en ejecución. Por ejemplo, el código del programa (las instrucciones) debe residir en algún lugar de la memoria, por lo que forma parte del espacio de direcciones. Mientras se ejecuta, el programa utiliza una pila (*stack*) para registrar su posición en la cadena de llamadas a funciones, asignar variables locales, pasar parámetros y devolver valores a las rutinas. Por último, el *heap* se emplea para la memoria gestionada dinámicamente por el usuario, como la que se asigna mediante `malloc()` en C o `new` en lenguajes orientados a objetos como C++ o Java. Además, el espacio de direcciones puede contener otros elementos, como variables inicializadas de forma estática.

En la Figura 2 tenemos un ejemplo de espacio de direcciones. El código del programa se encuentra en la parte inferior del espacio de direcciones. Dado que el código es estático (y, por lo tanto, fácil de ubicar en memoria), podemos colocarlo en la parte inferior del espacio de direcciones y estar seguros de que no necesitará más espacio durante la ejecución del programa. De manera similar, a continuación se puede ubicar el segmento con las variables inicializadas y

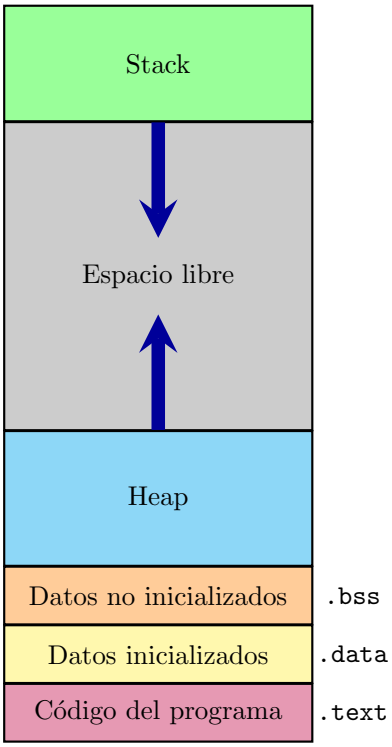


Figura 2: Ejemplo de espacio de direccionamiento virtual.

el segmento con las variables no inicializadas, que no crecerán durante la ejecución.

A continuación, tenemos las dos regiones del espacio de direcciones que pueden crecer (y reducirse) durante la ejecución del programa. Estas son el *heap* (ubicado después del segmento de datos inicializados) y la pila (en la parte superior). Se colocan de esta manera porque ambas requieren la capacidad de crecer, y al ubicarlas en extremos opuestos del espacio de direcciones, se facilita ese crecimiento: cada una puede expandirse en direcciones opuestas. Así, el *heap* comienza justo después de los datos no inicializados y crece hacia arriba (por ejemplo, cuando un usuario solicita más memoria mediante `malloc()`), mientras que la pila está ubicada en la parte superior y crece hacia abajo (por ejemplo, cuando un usuario realiza una llamada a procedimiento). Sin embargo, esta disposición de la pila y el *heap* es solo una convención; el espacio de direcciones podría organizarse de manera diferente.

Observación

El enfoque más simple para asignar el espacio de direcciones de cada proceso en memoria consiste en utilizar dos registros: uno para apuntar al inicio del espacio de direcciones y otro para almacenar su tamaño. Sin embargo, es importante señalar que existe un gran bloque de espacio "libre" entre la pila y el heap. Aunque este espacio no es utilizado por el proceso, aún ocupa memoria física. Por lo tanto, dicho enfoque resulta ineficiente y dificulta la ejecución de un programa cuando el espacio de direcciones completo no cabe en memoria. A continuación veremos métodos más eficientes.

4. 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

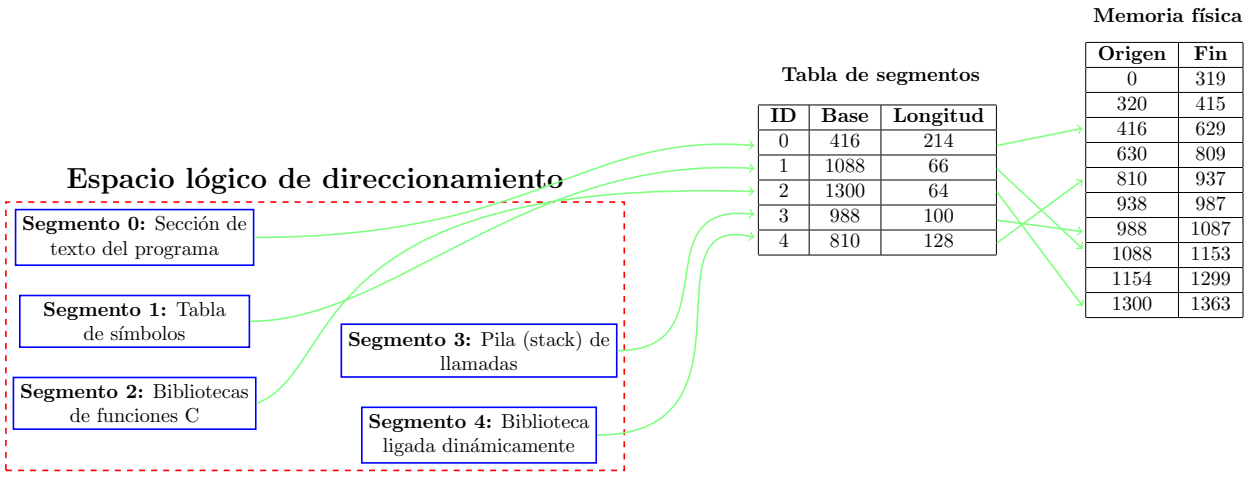


Figura 3: Ejemplo de esquema de segmentación (ID es el identificador del segmento).

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".

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 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].

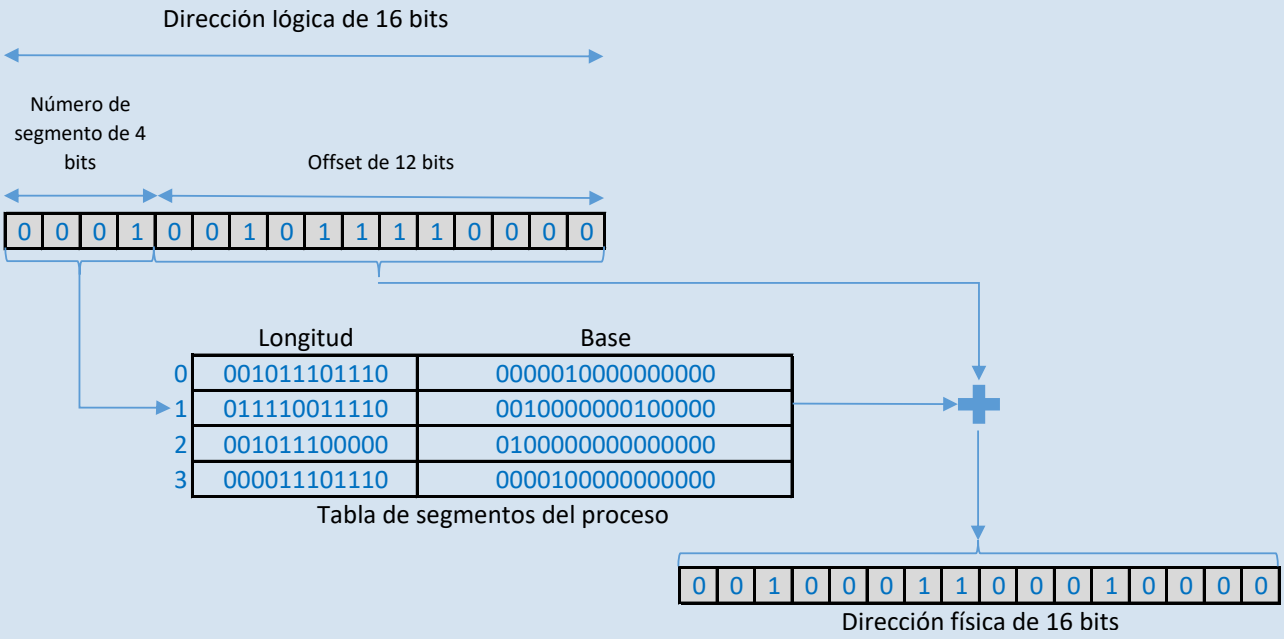
La Fig. 3 muestra de manera esquemática cómo funciona segmentación. En este ejemplo vemos 5 segmentos, su correspondiente tabla de segmentos y cómo se ubican estos segmento en la memoria física en base a la información de la tabla.

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:

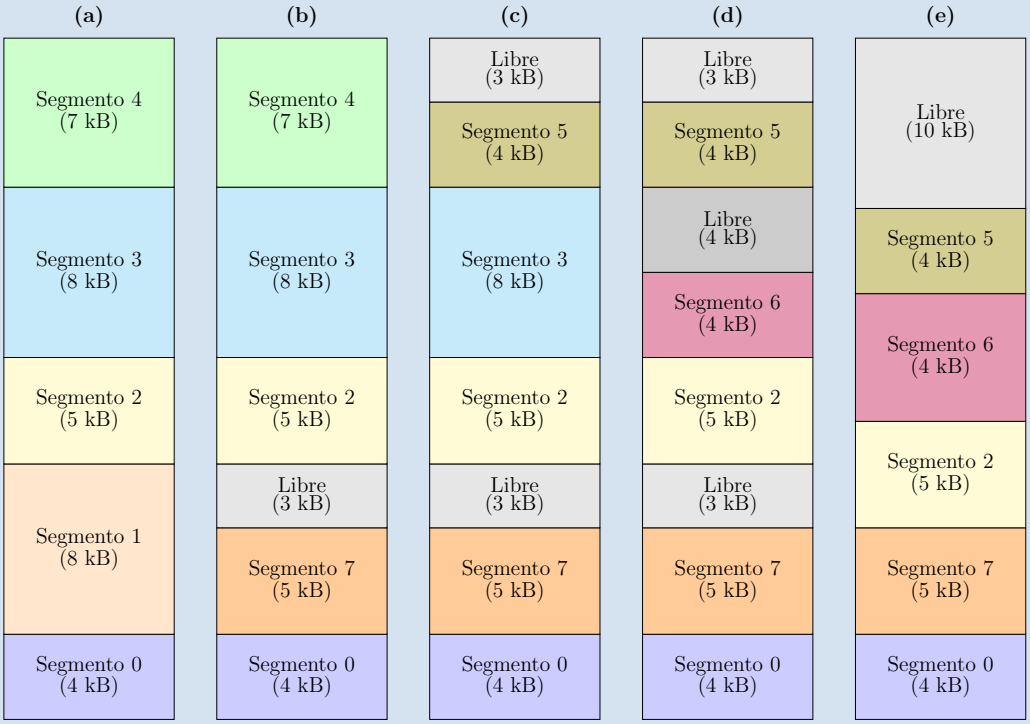
- *Extraer el número de segmento como los n bits más a la izquierda de la dirección lógica.*
- *Utilizar el número de segmento como índice en la tabla de segmentos del proceso para encontrar la dirección física inicial del segmento. En este ejemplo hay 4 segmentos.*
- *Comparar el desplazamiento, expresado 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`*.*

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.*

Los siguientes esquemas ilustran este concepto:



Supongamos que inicialmente hay cinco segmentos. Consideremos ahora qué ocurre si el segmento 1 es eliminado y en su lugar se coloca el segmento 7, que ocupa menos espacio. Esto genera la configuración de memoria representada en el esquema ***b****. En esta configuración, entre el segmento 7 y el segmento 2 aparece un área desocupada, es decir, un hueco.*

A continuación, el segmento 4 es reemplazado por el segmento 5, como se muestra en el esquema ***e****. Posteriormente, el segmento 3 es sustituido por el segmento 6, generando la configuración que se observa en el esquema* ***d****.*

Consideremos qué ocurriría si quisiéramos agregar un segmento de 8 kB en el esquema ***d****. Aunque el espacio total disponible en los huecos es de 10 kB, suficiente en términos absolutos, este espacio está fragmentado en pequeños bloques inútiles, lo que impide cargar directamente el segmento. Por lo tanto, sería necesario eliminar primero algún otro segmento.*

Para evitar la fragmentación externa, una posible solución es desplazar los segmentos que están después de cada hueco hacia posiciones más cercanas a la dirección de memoria 0, eliminando así los huecos y dejando un único espacio grande al final. Como alternativa, se puede optar por esperar hasta que la fragmentación externa sea significativa (por ejemplo, cuando un porcentaje considerable de la memoria total esté desperdiciado en huecos) antes de realizar una compactación (o defragmentación).

El esquema ***e*** *ilustra cómo se vería la memoria del esquema* ***d*** *tras una compactación. El objetivo de esta operación es consolidar los pequeños huecos en un único bloque grande, permitiendo acomodar uno o más segmentos. Sin embargo, la compactación tiene la desventaja evidente de consumir tiempo para llevarse a cabo. Por lo general, no es práctico compactar cada vez que se genera un hueco, ya que el tiempo invertido sería excesivo.*

5. 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. 4 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). En este caso, el espacio de direcciones virtuales se divide en 2^{20} páginas.

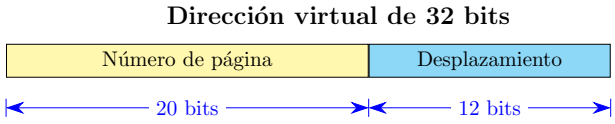


Figura 4: Ejemplo de esquema de dirección virtual utilizando paginación.

El procesador genera una **dirección virtual**, la cual se convierte, mediante una combinación de hardware y software, en una dirección física. Esta última puede ser utilizada para acceder a la memoria principal. La Fig. 5 muestra una memoria direccionada de forma virtual, con páginas asignadas a la memoria principal. Este proceso se denomina conversión o **traducción de direcciones**. Obsérvese también que algunas páginas no se encuentran en la memoria principal, sino que están almacenadas en la memoria secundaria.

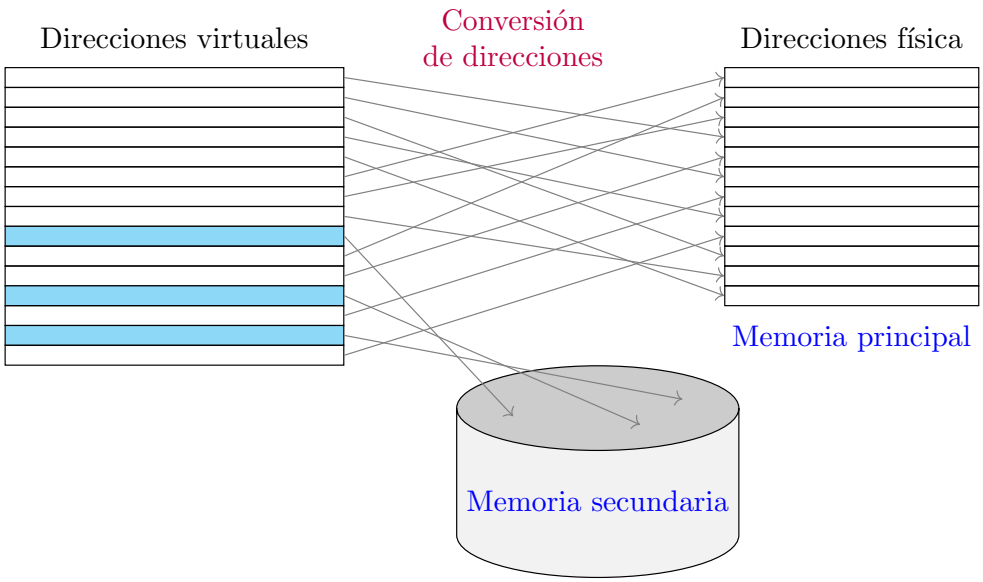


Figura 5: Conversión de direcciones.

6. La tabla de paginación

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

Debido a su gran tamaño, la tabla de páginas se almacena en la memoria física. Supongamos que está organizada como un arreglo contiguo, como se muestra en la Fig. 6. La tabla de páginas está indexada por el número de página virtual (VPN). Por ejemplo, la entrada 3 indica que la

página virtual 3 está asignada a la página física `0x7ffe`. En contraste, la entrada 4 no es válida ($V = 0$), lo que significa que la página virtual 4 no se encuentra en la memoria principal, sino en la memoria secundaria.

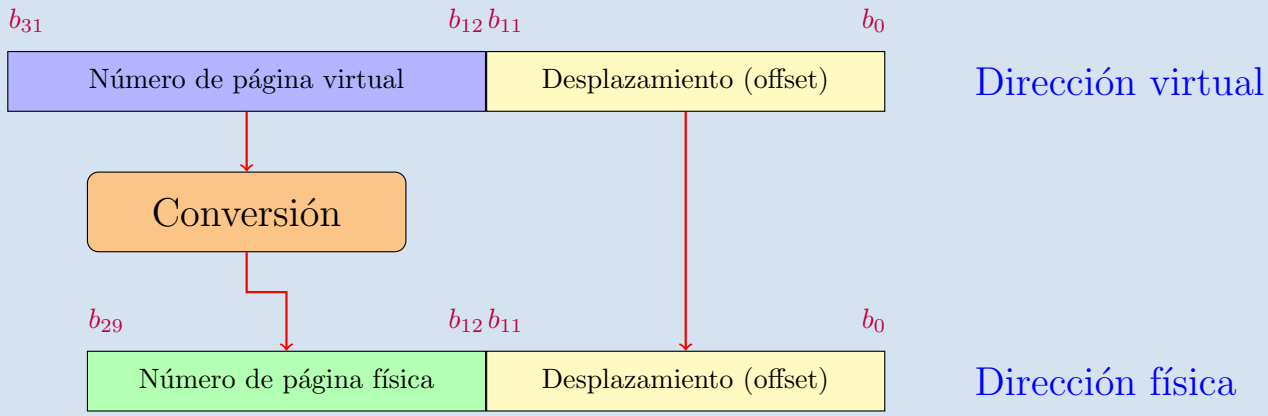
Para realizar una operación de memoria, el procesador primero debe traducir la dirección virtual en una dirección física y luego acceder a los datos en dicha dirección física. El procesador lee la entrada correspondiente en la tabla de páginas, que está almacenada en la memoria física, para obtener el número de página física. Si la entrada es válida, combina este número de página física con el desplazamiento dentro de la página para generar la dirección física. Finalmente, lee o escribe los datos en esta dirección física. Debido a que la tabla de páginas se almacena en la memoria física, cada operación de carga o almacenamiento requiere dos accesos a la memoria física.

Número de página virtual	V	Dirección de página física
0x7ffff	0	
0x7fffe	0	
0x7fffd	0	
0x7fffc	0	
0x7fffb	0	
0x7fffa	1	0x0001
⋮	⋮	⋮
0x00004	0	
0x00003	0	
0x00002	1	0x7ffe
0x00001	0	
0x00000	1	0x0000

Figura 6: Tabla de páginas.

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	Dirección de página física
0x7fff9	1	0x3fe00
0x7fff8	1	0x3ff0c
0x7fff7	0	0x32056
0x7fff6	1	0x36000
0x7fff5	1	0x36020
0x7fff4	0	0x31c09
0x7fff3	0	0x3021f
0x7fff2	0	0x320a0
0x7fff1	1	0x31800
0x7fff0	0	0x3aa00

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 `0x7fff5a06`*. 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.

7. Tamaño de página

El tamaño de la página es un parámetro que, en muchos casos, puede ser seleccionado por el sistema operativo entre varios tamaños compatibles con la arquitectura. Determinar el tamaño de página óptimo implica equilibrar diversos factores en competencia, lo que significa que no existe una solución única que sea ideal en todos los casos.

Por un lado, es común que un proceso no utilice completamente todas las páginas asignadas. En promedio, la mitad de la última página de un proceso quedará vacía, y ese espacio adicional se considera desperdiciado. Este tipo de desperdicio se conoce como **fragmentación interna**[3]. Si hay n procesos y el tamaño de la página es de p bytes, se estima que se desperdiciarán $n \times \frac{p}{2}$ bytes en promedio debido a la fragmentación interna. Esto sugiere que un tamaño de página más pequeño podría ser beneficioso.

Sin embargo, páginas más pequeñas implican que los programas requerirán un mayor número de páginas, lo que a su vez conduce a una tabla de páginas más grande. Este tema se abordará en detalle en la Sección 6.

8. Translation Lookaside Buffer (TLB)

Dado que cada operación de carga o almacenamiento implica dos accesos a la memoria física, esto podría tener un impacto severo en el rendimiento. Afortunadamente, los accesos a la tabla de páginas suelen tener una alta 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 Unidad de Manejo

[3]Se denomina **fragmentación interna** porque ocurre dentro de la página.

de Memoria (MMU) recuerda la última entrada de la tabla de páginas que consultó, es posible que pueda reutilizar esta traducción sin necesidad de volver a acceder a la tabla de páginas.

En general, el procesador almacena las últimas entradas de la tabla de páginas en una pequeña tabla llamada **Translation Lookaside Buffer (TLB)**, que está ubicada dentro de la memoria caché. El procesador primero intenta encontrar la traducción en la TLB antes de tener que acceder a la tabla de páginas en la memoria física. En la práctica, la gran mayoría de los accesos se realizan utilizando la TLB, lo que evita las costosas lecturas de la tabla de páginas desde la memoria física.

La TLB se organiza como una memoria caché totalmente asociativa y generalmente contiene entre 16 y 512 entradas. Cada entrada de la TLB incluye un número de página virtual, su correspondiente número de página física y algunas banderas de estado. Se accede a la TLB utilizando el número de página virtual. Si la TLB tiene un "acierto", 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 permitir su acceso en menos de un ciclo de reloj. A pesar de su tamaño compacto, las TLB suelen tener una tasa de aciertos superior al 99 %. Gracias a la TLB, se reduce el número de accesos a la memoria necesarios para la mayoría de las instrucciones de carga o almacenamiento de dos a uno (e incluso más en sistemas con paginación multinivel).

Ejemplo

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

V	Número de página virtual	Número de página físico
0	`0xefff0`	`0x0aa00`
1	`0xffff5`	`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.*

9. 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.

10. Políticas de reemplazo

Los sistemas de memoria virtual utilizan una política de escritura diferida (*write-back*) y un algoritmo de reemplazo aproximado de las páginas menos recientemente usadas (*Least Recently Used*, LRU). Una política de escritura directa (*write-through*), donde cada escritura en la memoria física inicia una escritura en el disco duro, sería poco práctica. Las instrucciones de almacenamiento operarían a la velocidad del disco duro en lugar de la velocidad del procesador (milisegundos en lugar de nanosegundos). Bajo la política de escritura diferida, la página física se escribe en el disco duro solo cuando es desalojada de la memoria física. Este proceso, que implica escribir la página física en el disco duro y recargarla con una página virtual diferente, se denomina *paginación*. En este contexto, el disco duro se conoce a veces como *área de intercambio* (*swap space*).

Cuando ocurre un fallo de página (*page fault*), el procesador expulsa una de las páginas físicas menos recientemente usadas y la reemplaza con la página virtual que falta. Para respaldar estas políticas de reemplazo, cada entrada en la tabla de páginas contiene dos bits adicionales de estado: un bit de modificación (*dirty bit*, D) y un bit de uso (*use bit*, U).

El *dirty bit* es igual a 1 si alguna instrucción de almacenamiento ha modificado la página física desde que se leyó del disco duro. Cuando una página física se expulsa, debe escribirse en el disco duro únicamente si su *dirty bit* es 1; de lo contrario, el disco duro ya contiene una copia exacta de la página.

El *use bit* es igual a 1 si la página física ha sido accedida recientemente. Al igual que en un sistema de caché, un reemplazo exacto de LRU sería excesivamente complicado. En su lugar, el sistema operativo aproxima el reemplazo LRU restableciendo periódicamente todos los *use bits* en la tabla de páginas. Cuando se accede a una página, su *use bit* se establece en 1. En caso de una falta de página, el sistema operativo busca una página con U = 0 para expulsarla de la memoria física. Por lo tanto, no necesariamente se reemplaza la página menos recientemente usada, sino una de las páginas menos recientemente usadas.

11. 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.

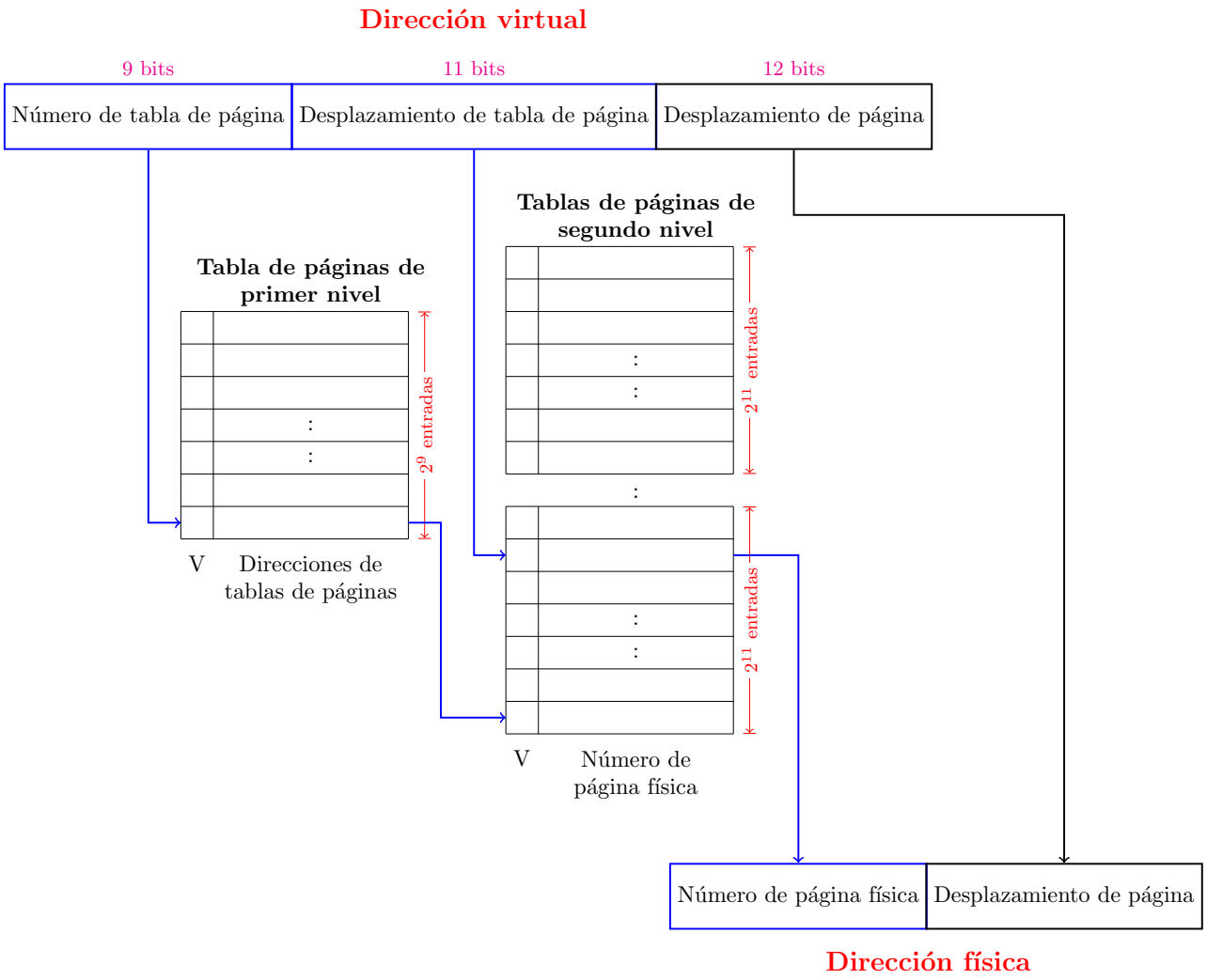


Figura 7: Ejemplo de paginación con tabla de páginas de dos niveles.

Hoy en día, las estructuras de tablas de páginas más complejas comprenden cuatro niveles[4]. 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. 7. El número de tabla de páginas indexa la tabla de páginas de primer 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 $V = 0$. A su vez, 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).

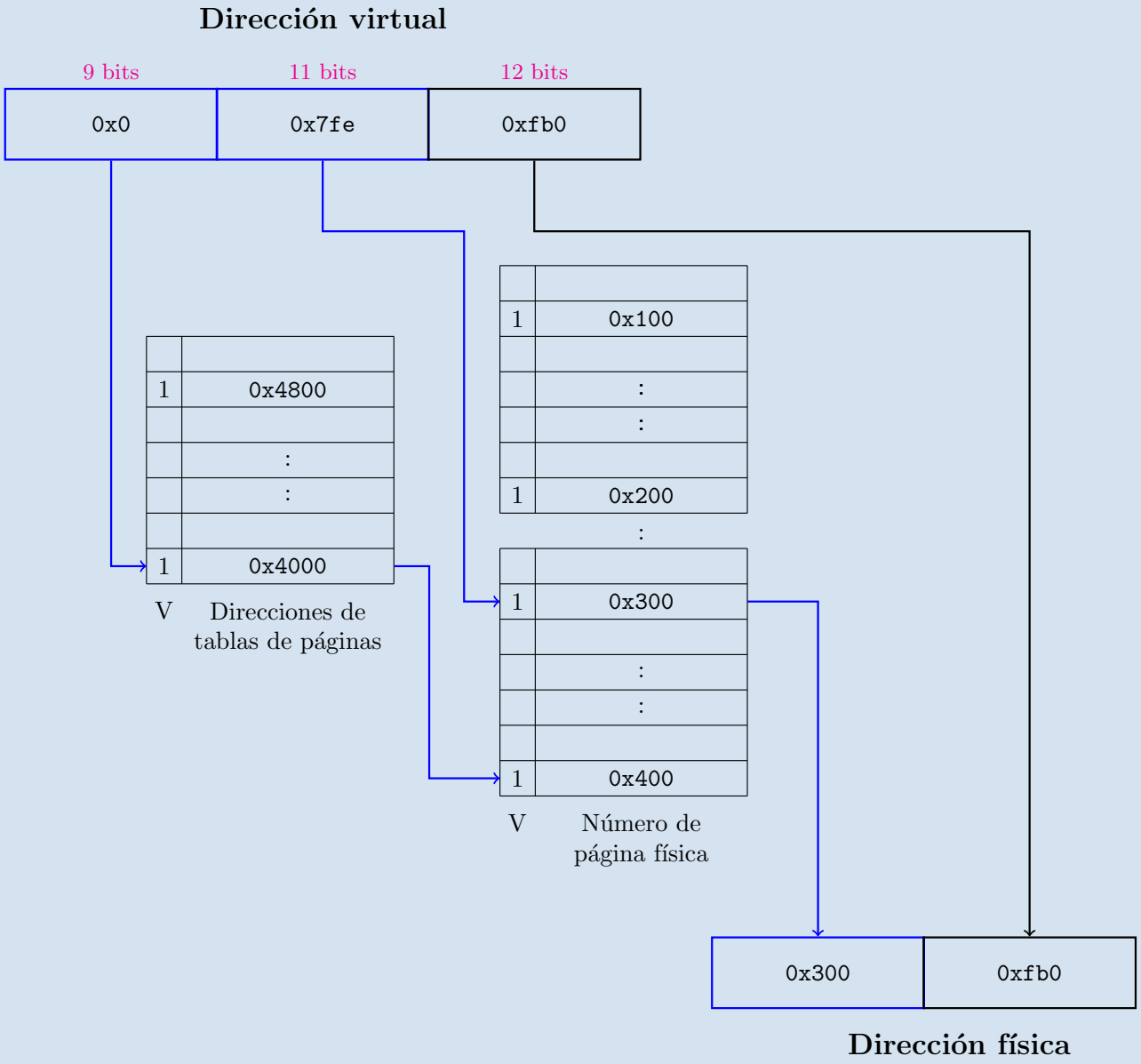
En el ejemplo de la Fig. 7, el número de página virtual de 20 bits en se divide en 9 y 11 bits, para indicar el número de la tabla de páginas y el desplazamiento de la tabla de páginas, respectivamente. Así, la tabla de páginas de primer nivel tiene $2^9 = 512$ entradas. Cada una

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

de estas 512 entradas apunta a una tabla de segundo nivel. A su vez, cada una de las tablas de segundo nivel tiene $2^{11} = 2048$ 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 512×4 bytes $+ 2 \times 2048 \times 4$ bytes $= 18$ 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, a modo de ejemplo, los posibles contenidos de una tabla de páginas de dos niveles, como la presentada en la Fig. 7. A partir de esta información, veremos cómo se realiza la traducción de la dirección virtual `0x007fefb0`.



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*

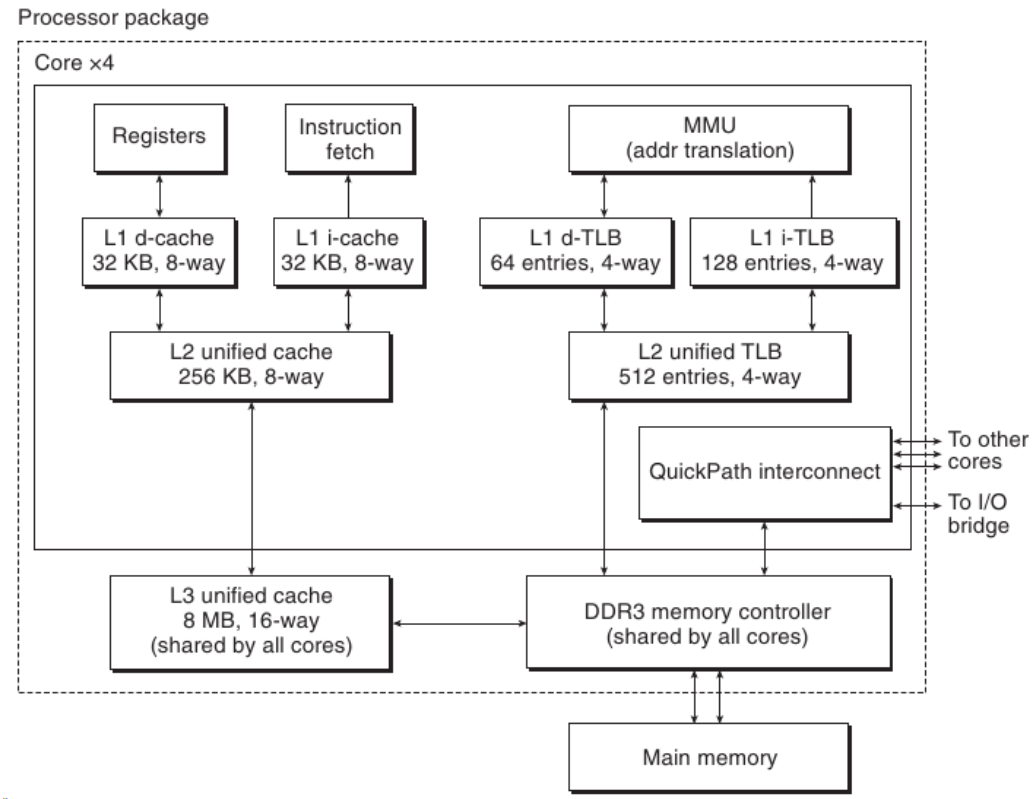


Figura 8: Sistema de memoria del procesador Intel Core i7 (fuente: [2]).

entrada `0x0` *indica que la tabla de páginas de segundo nivel reside en la memoria (porque V=1) y su dirección física es* `0x4000`*. Los siguientes once bits de la dirección virtual,* `0x7fe`*, son el desplazamiento de la tabla de páginas, lo que proporciona el índice en la tabla de páginas de segundo nivel. La entrada* `0x7fe` *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* `0x300`*. El número de página física se concatena con el desplazamiento de página para formar la dirección física, la cual resulta* `0x300fb0`*.*

12. 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. 8 muestra los aspectos más destacados del sistema de memoria del procesador Intel Core i7. El chip incluye cuatro núcleos[5], una gran caché L3 compartida por todos los núcleos y un controlador de memoria DDR3, entre otros módulos. Cada núcleo contiene una jerarquía de 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

[5]Intel Core i7 es una familia de procesadores con diferentes modelos. El modelo mostrado en la Fig. 8 tiene 4 núcleos pero hay modelos con mayor cantidad de núcleos.

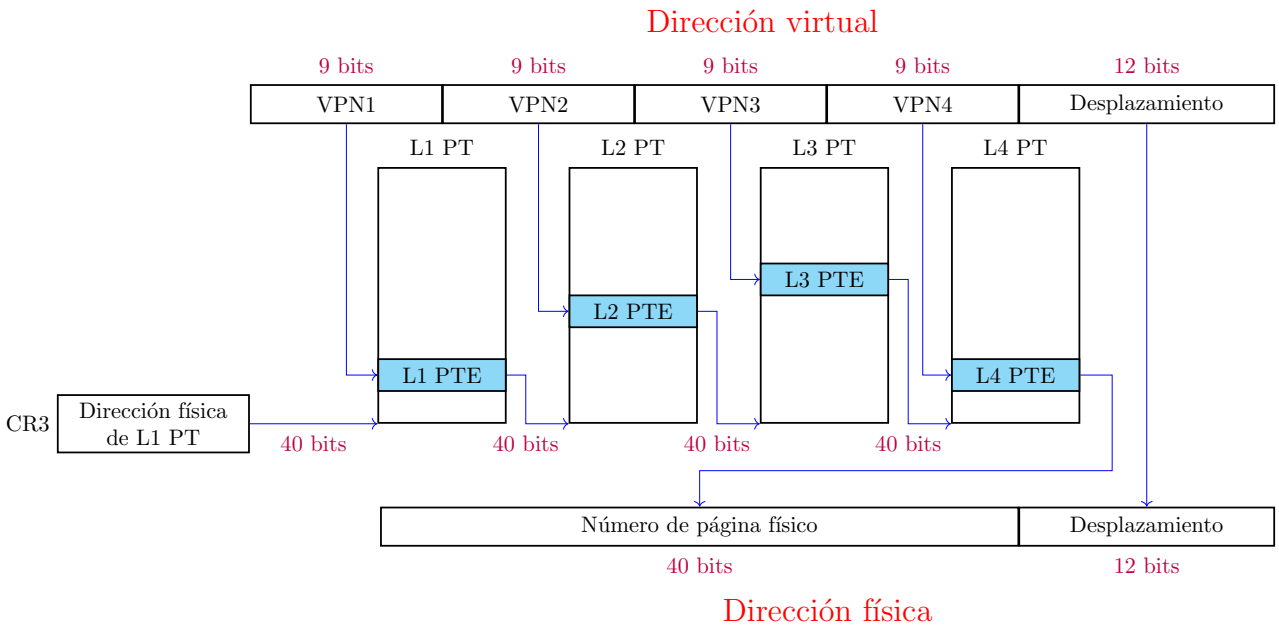


Figura 9: Esquema de traducción de direcciones virtuales del Intel Core I7 con tabla de páginas multinivel.

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[6]. 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. 9.

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 nos proporciona la información guardada.

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

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] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, first edition, 2018.

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

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

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

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

[6] 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.

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

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

[9] 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.

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