Ejercicios de Sistemas Operativos. Tema 3: Memorias
E.T.S. Ingeniería Informática. Universidad de Málaga
Profesor: Manuel Ujaldón
Dpto. de Arquitectura de Computadores

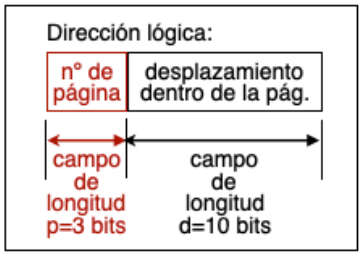
Resumen preliminar. Este capítulo tiene 8 conceptos clave, asociados en 4 parejas afines:

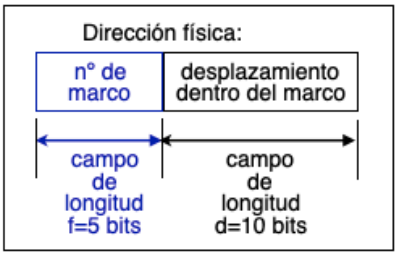
- Cada proceso que accede a memoria emite **direcciones virtuales** (también denominadas *direcciones lógicas*), creando antes su propia memoria o **espacio de direcciones virtuales**.
- El Sistema Operativo se encarga de traducir cada dirección virtual a una **dirección física**, ubicada dentro del **espacio de direcciones físicas**, que es la memoria real de que se dispone.
- Tanto el espacio virtual como el físico se descompone en bloques de tamaño fijo de entre 4 y 16 Kbytes, denominados **páginas** en el primer caso y **marcos** *(frames)* en el segundo.
- La traducción de cada dirección virtual a su dirección física asociada se realiza a nivel de página, que encontrará su marco asociado en una **tabla de páginas** mantenida por el Sistema Operativo. Para acelerar este proceso, se pueden guardar en una **TLB** (o caché) aquellas parejas de traducciones ya hechas (cada número de página y su número de marco asociado), evitando el acceso a la tabla de páginas cuando son referenciadas de forma frecuente.

---

1. Sea un espacio de direcciones lógico de 8 páginas de 1024 palabras de un byte cada una, mapeado sobre un espacio de direcciones físico de 32 marcos. ¿Cuántos bits de longitud tiene cada dirección lógica y cada dirección física?

   La dirección lógica tiene 2^3 páginas de 2^{10} palabras, por lo que debe tener una longitud de $3 + 10 = 13$ bits.
   Por otra parte, la dirección física tiene 2^5 marcos de 2^{10} palabras (la página y su marco asociado deben coincidir en tamaño para que éste pueda albergar a aquélla). Por lo tanto, cada dirección física debe tener una longitud de $5 + 10 = 15$ bits. En resumen, tendríamos lo siguiente:





   Aclaración: Lo habitual es que la dirección lógica sea más extensa que la dirección física, puesto que cada proceso suele utilizar más memoria virtual de la que se dispone físicamente en el sistema. En el caso que nos ocupa sucede lo contrario, pero aún así, el sistema necesitará memoria virtual en cuanto haya más de 4 procesos, pues en total se superarán las 32 páginas para alojar por completo su espacio de direcciones, y sólo tenemos 32 marcos para ellas en memoria física.

2. Una memoria virtual paginada cuenta con direcciones lógicas de 16 bits y 16 marcos de 1024 palabras de memoria. En un momento dado, su tabla de páginas tiene los siguientes contenidos específicos para sus 64 entradas (en azul):

N	**Valor**
0	**1001**
1	**0100**
2	**0000**
...	...
62	**0101**
63	**1100**

Realizar la traducción a dirección física de las siguientes direcciones de memoria virtual:

a) 1111111111111111.

b) 0000000000000000.

c) 0000100100010001.

d) 1111100000111111.

El tamaño del marco, 1024 palabras (2^{10}), nos indica que se utilizan 10 bits para direccionar el desplazamiento dentro de la página (y dentro del marco, pues ambos tienen el mismo tamaño). Estos 10 bits son los últimos de la dirección, al ser el campo numéricamente menos significativo. Por lo tanto, los 6 bits superiores de la dirección lógica corresponden a su número de página, y con estos dos campos de 6 y 10 bits debemos realizar la traducción a una dirección física de 14 bits (4 para el marco y 10 para el desplazamiento) de la siguiente forma:

- El número de página (6 bits) debe traducirse al número de marco que contiene dicha página en memoria física utilizando la tabla de páginas anterior, donde la página actúa como índice de acceso a las entradas de la tabla y el marco se corresponde con el valor de dicha entrada.
- El desplazamiento dentro del marco (10 bits) deberá agregarse al número de marco anterior para dar con la ubicación exacta de la dirección de memoria solicitada, que estará compuesta de ambos campos concatenados. Esta concatenación se produce por el peso numérico de cada uno de los campos: Los marcos direccionan en múltiplos de 1K (que por tanto comienzan en una dirección que tiene los 10 últimos bits a cero) y el desplazamiento dentro del marco es el que concreta el valor de estos 10 últimos bits. En decimal quizá lo veamos mejor: si tenemos 500 palabras de memoria física organizada en 50 marcos de 10 palabras cada uno, el marco 27 agrupa desde la dirección 270 a la 279, y así, la dirección 273 estaría en el marco 27, y dentro de él, en la dirección 3 (y concatenando 27 y 3, formamos el número 273).

Lo primero es localizar el marco que corresponde a la página según la tabla, y posteriormente el dato dentro del marco, que se encontrará en la dirección relativa indicada por su desplazamiento. Procedamos de esta forma para cada una de las referencias a memoria, coloreando en rojo el **número de página**, en azul su **número de marco** asociado, y en negro el **desplazamiento** dentro de ambos:

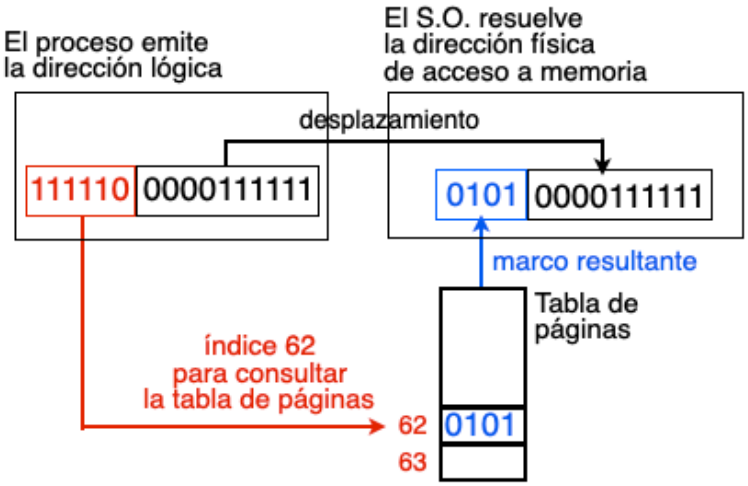
a) La dirección virtual **111111**1111111111 está en la página **63**, desplazamiento **1023** (pasando ambos de binario a decimal). La tabla de páginas indica que la página **63** se encuentra ubicada en el marco **1100** de memoria física (esto es, el **12**), por lo que la dirección física resultante es la concatenación de los campos marco y desplazamiento, esto es, **1100**1111111111.

b) La dirección virtual **000000**0000000000 está en la página **0**, desplazamiento **0**. La tabla de páginas indica que la página **0** se encuentra ubicada en el marco **1001** de memoria física (el **9** en numeración decimal), por lo que la dirección física resultante es **1001**0000000000. En representación decimal, diríamos que la dirección virtual 0 se ubica en la dirección física 9K, esto es, 9x1024 = 9216.

c) La dirección virtual **000010**0100010001 está en la página **2**, que según la tabla de páginas se ubica en el marco **0000** de memoria física, por lo que la dirección física resultante es **0000**0100010001.

d) Por último, la dirección virtual **111110**0000111111 está en la página **62**, que la tabla de páginas nos dice que se ubica en el marco **0101** de memoria física, resultando la dirección física **0101**0000111111.

Para esta última traducción, el esquema visto en clase de forma genérica para abordar las traducciones podría particularizarse de la siguiente forma:



---

3. Sobre un espacio de direcciones virtuales de 1 Giga-palabras de 1 byte montado sobre una memoria física de 256 Megabytes, calcular:

a) El tamaño de página que da lugar a una tabla de páginas de 2^{19} entradas.

b) El número de bits de que se compone cada dirección física.

c) El número de marcos que necesita la memoria física.

a) La tabla tiene una entrada por cada página de memoria virtual, por lo que debemos tener 2^{19} páginas de X palabras cada una para componer el espacio de direcciones virtuales de 1 Gigapalabra (2^{30} palabras). Despejando X, tenemos un tamaño de página de X= $\frac{2^{30}}{2^{19}} = 2^{11} =$ 2048 palabras.

b) La memoria física tiene 256 Mpalabras de 1 byte, y dado que 2^{28} = 256M, necesitamos 28 bits para direccionar todas esas palabras.

c) El marco y la página tienen el mismo tamaño, que ha resultado ser de 2048 palabras de 1 byte (2^{11} bytes). Para componer el total de la memoria física de 256 Mbytes (2^{28} bytes) son necesarios por tanto 2^{17} marcos de 2^{11} bytes cada uno.

4. Consideremos direcciones virtuales de 64 bits y páginas de 4 Kilobytes sobre una memoria física de 512 Megabytes que direcciona palabras de 32 bits. Calcular el espacio que ocupa en memoria la tabla de páginas considerando que se usan 7 bits de control en cada una de sus entradas.

La tabla de páginas contiene una entrada o fila por cada página de memoria virtual, que se encarga de darnos su traducción al marco que lo aloja en memoria física (y contendrá, adicionalmente, los 7 bits de control).
Cada página de 4 Kbytes alberga 1K palabras de 32 bits (4 bytes). El número de páginas necesario es el cociente entre las 2^{64} direcciones virtuales y las 2^{10} palabras que caben en cada página, es decir, 2^{54}. Ése es el número de filas de la tabla de páginas.
Por otro lado, la memoria física es de 512 Mbytes y sus marcos son de 4 Kbytes, existiendo en total $\frac{512Mbytes}{4Kbytes}=\frac{2^{29}}{2^{12}} = 2^{17}$ marcos de página, por lo que son necesarios 17 bits para direccionarlos. Estos 17 bits constituyen la longitud de cada fila de la tabla de páginas (puesto que contiene el número de marco en el que se aloja cada página), completados con los 7 bits de control para llegar a 24 bits (3 bytes) en cada entrada de la tabla de páginas.
La tabla de páginas tendrá por tanto 2^{54} filas de 3 bytes cada una, es decir, 48 Petabytes. La magnitud de este número nos dice que con una memoria virtual tan extensa es imposible utilizar un sistema paginado de un solo nivel, teniendo que recurrir a una tabla de páginas multinivel. El precio a pagar es la lentitud de las traducciones de dirección virtual a física, puesto que ahora serán necesarios múltiples accesos a memoria física para consultar la tabla de páginas, uno para cada nivel añadido.

---

5. Un sistema de memoria virtual con páginas de 8 Kilobytes, espacio lógico de 1 Terabyte y espacio físico de 32 Gigabytes, ambos direccionables con palabras de memoria de un byte, utiliza paginación multinivel para implementar su ingente tabla de páginas. Indica la estructura de dicha tabla de páginas (número de entradas que posee y tamaño que tiene cada una de ellas) según organicemos ésta en múltiples niveles. Partiendo de la tabla en un solo nivel, debes ir paginando la misma e incorporando nuevos niveles sucesivamente más externos para acceder a todas las páginas que resulten del nivel anterior, hasta llegar al nivel en el que todas sus entradas quepan en una sola página. Considera la presencia de 10 bits de control acompañando al número de marco en cada entrada de las tablas de páginas.

La tabla de páginas contiene una entrada por cada página de memoria virtual, que nos indica el marco que aloja sus datos en memoria física (junto con los 10 bits de control que maneja el Sistema Operativo).
Una memoria física de 32 Gigabytes se descompone en 4 Mega-marcos de 8 Kilobytes cada uno (el tamaño del marco debe coincidir con el de la página, puesto que alberga los datos de ésta). Estos 4 Mega-marcos pueden direccionarse usando $f = 22$ bits, ya que $2^{22} = 2^2 \times 2^{20} = 4$ Mega. Con este dato ya podemos concretar la **anchura de las tablas de páginas: 32 bits**, que corresponden con los 22 bits para el número de marco que aloja sus datos y los 10 bits de control para dicho marco (bit de presencia, bit de validación, bit *dirty*, etc).

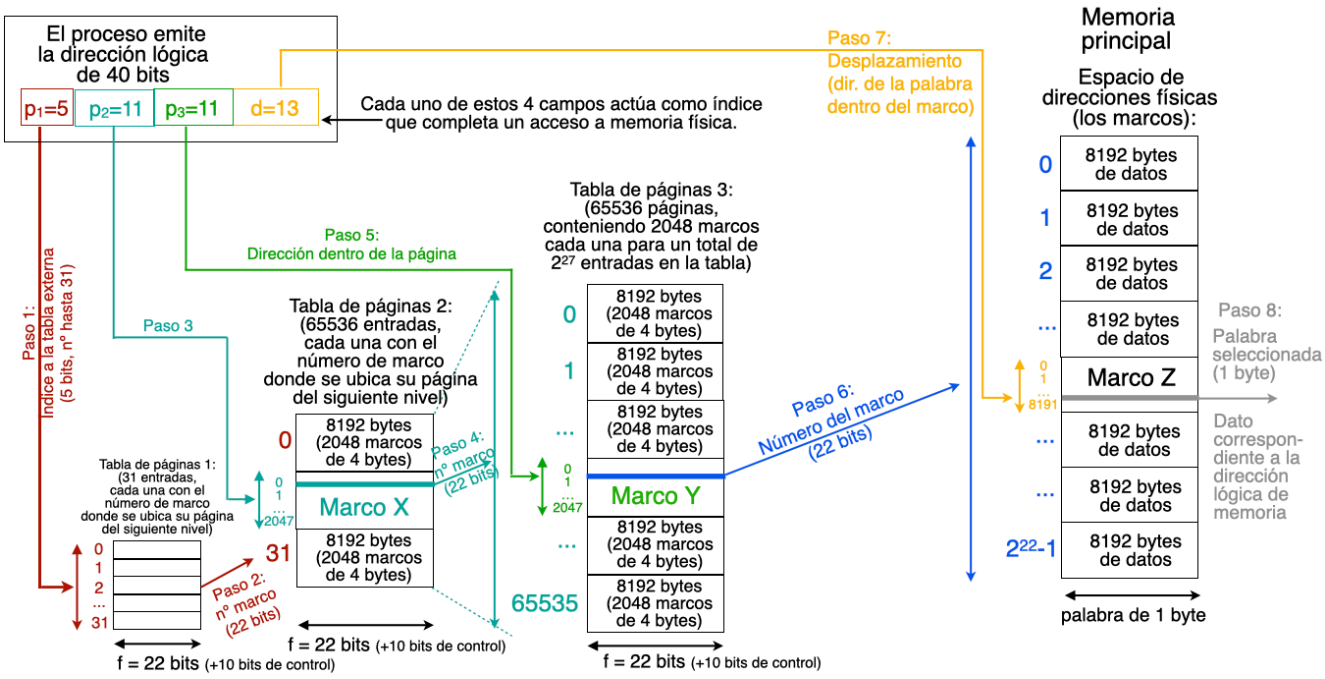
- La memoria virtual de 1 Terabyte tendrá 128 Mega-páginas de 8 Kilobytes cada una. Puesto que la tabla de páginas tiene una entrada para cada página, dispone de 128 Mega-entradas, que ya sabemos que son de 32 bits. En total, el espacio ocupado por **esta tabla será de 128 Mega-entradas $\times$ 4 bytes** si la implementamos en un solo nivel, ocupando un espacio de 512 Mbytes.

- Para incorporar el segundo nivel, procedemos a paginar la tabla anterior. En una página de 8 Kbytes caben 2048 entradas de la tabla (de 4 bytes cada una), por lo que el número de páginas necesarias para esta tabla es de $\frac{128\ Megaentradas}{2048\ entradas/pagina} = 65536$ páginas. También podemos llegar a este dato dividiendo el espacio ocupado por la tabla anterior (512 Mbytes), por el tamaño de cada página (8 Kbytes).
  La tabla del segundo nivel debe contener una entrada por cada página de la tabla anterior, en la que se ubicará el marco que la contiene. Esto nos lleva a una **segunda tabla que tiene 65536 entradas de 4 bytes** cada una (de nuevo, los 22 bits del marco y los 10 bits de control), ocupando un espacio total de 256 Kbytes.
- Para incorporar el tercer nivel, paginaremos la tabla anterior de 256 Kbytes. Dividiendo este tamaño por el de la página, 8 Kbytes, obtenemos que son necesarias 32 páginas para alojar la segunda tabla, por lo que **esta tercera tabla ubicada en el nivel más externo tendrá 32 entradas**. Una vez más, cada entrada de esta tabla contendrá 32 bits (el número de marco de 22 bits y los 10 bits de control adicionales). El tamaño de esta tercera tabla es de sólo 128 bytes, por lo que ya cabe dentro de una sola página de memoria, concluyendo aquí la construcción de todas las tablas suplementarias necesarias para acceder al último nivel.

Recopilando todo y renumerando los niveles para comenzar por el más externo, tenemos:

- El acceso al nivel 1 más externo se realiza con un índice p_1 de 5 bits que accede a una tabla de 32 entradas, donde encontramos el marco que ubica la página de la tabla del siguiente nivel a la que debemos acceder. Esto completa el primer acceso a la memoria física.
- El marco anterior nos sitúa en una página X dentro de la tabla del nivel 2, donde ahora utilizaremos el índice p_2 como desplazamiento desde la base de X para obtener a su vez el marco que ubica la página de la tabla del nivel 3. Así completamos el segundo acceso a memoria física.
- A su vez, el marco anterior nos ubica al comienzo de una página Y dentro de la tabla del nivel 3 (el más interno). Aquí usaremos el índice p_3 como desplazamiento dentro de Y para obtener la dirección física que accede a memoria física por tercera vez.
- Finalmente, el dato de la dirección anterior nos proporciona la dirección base del marco Z, que ya contiene los 8 Kbytes de datos en memoria física. Utilizando el campo `d` (desplazamiento) como dirección relativa a dicha base, obtendremos la dirección de memoria física donde se encuentra la palabra solicitada por la dirección virtual de partida, completando el cuarto y último acceso a memoria física.

La estructura de cada una de las tablas que componen los índices de acceso a los diferentes niveles se refleja en la siguiente figura conjunta:



---

6. Un sistema de memoria virtual con palabras de 32 bits habilita 2^{20} marcos de página en una memoria física de 64 Gbytes. La traducción de direcciones virtuales a físicas tiene lugar a través de una tabla de páginas estructurada en dos niveles, cuyos campos de acceso son los siguientes:

Índice al primer nivel	Índice al segundo nivel	Campo desplazamiento dentro de la página
$p_1 = 12$ bits	$p_2 = 14$ bits	d = ?

Teniendo en cuenta toda esta información, se pide determinar:

a) El tamaño de página.

b) La longitud del campo `d` (desplazamiento) que direcciona la palabra dentro del marco.

c) El espacio de direcciones virtuales de cada proceso.

d) Si se considera una sola tabla de páginas (un solo nivel), indica cuántas entradas serían necesarias en dicha tabla. Estas entradas tendrán que incorporar, adicionalmente, 12 bits de control cada una.

e) Incorpora ahora el segundo nivel de la tabla paginando el anterior e indicando el efecto que tendría en el rendimiento del proceso de traducción respecto al caso anterior.

En esta ocasión, las palabras de memoria no son de un byte como es habitual en un PC actual, sino de 32 bits (4 bytes). Esto supone que la memoria se direcciona en múltiplos de 4 si tomamos el byte como unidad de referencia (es decir, la CPU accede al byte 0, 4, 8, ..., ya que el valor del byte 2, por ejemplo, forma parte de la primera palabra de memoria junto a los valores de los bytes 0, 1 y 3). Más razonable y sencillo resulta, por tanto, adaptar nuestros bytes a palabras y usar ésta como magnitud de referencia, teniendo siempre presente que las cantidades expresadas en bytes se dividen por 4 para saber cuántas palabras de memoria contienen.

a) El tamaño del marco coincide con el de la página de memoria. Si tenemos 2^{20} marcos en una memoria física de 64 Gigabytes con palabras de 4 bytes, la memoria tendrá 16 Gigapalabras (2^{34} palabras). El cociente entre esta cantidad y el número de marcos, nos dirá cuántas palabras hay en cada marco, y su logaritmo binario será la longitud del campo `d` utilizado para direccionar la palabra dentro del marco:
$\frac{2^{34} palabras}{2^{20} marcos} = 2^{14}$ palabras/marco.
Por lo tanto, el tamaño de página es de 2^{14} palabras (64 Kbytes) y d=14.

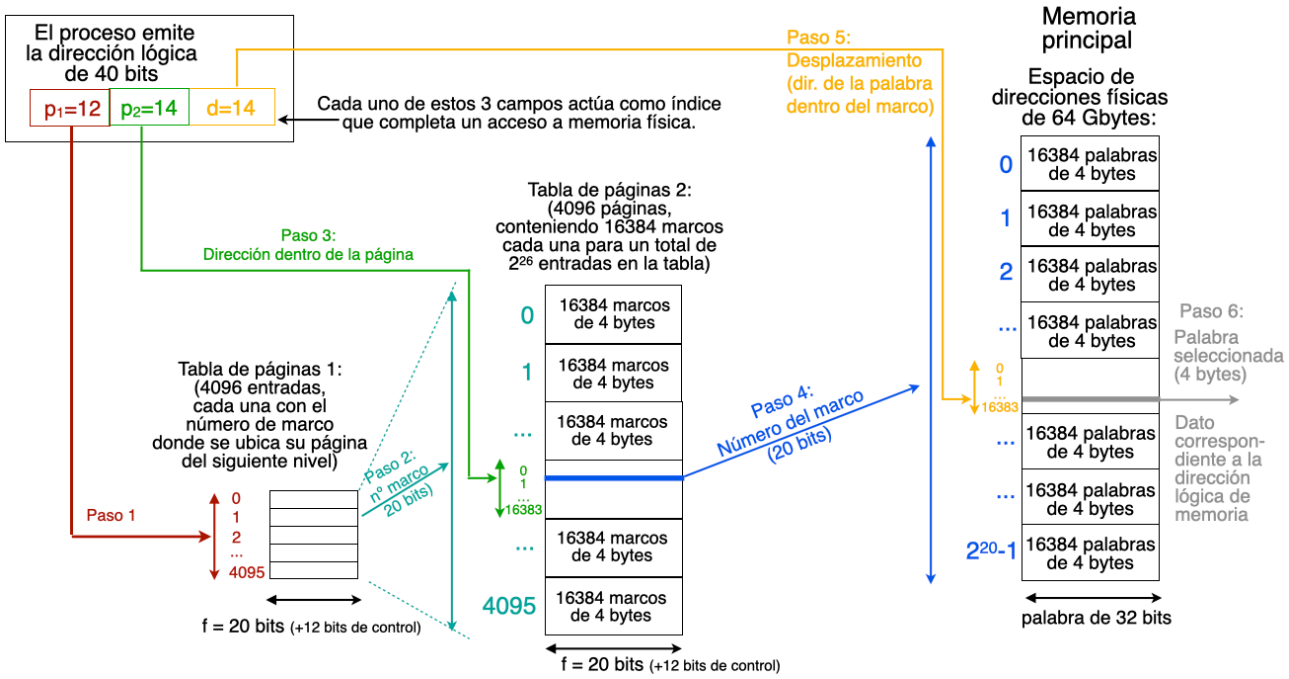
b) La longitud del campo `d` (desplazamiento), 14 en este caso, nos dice el número de bits necesarios para direccionar la palabra dentro de la página (o del marco). Este desplazamiento interviene en el último paso del proceso de traducción, una vez se conoce el marco que alberga los datos de la página que contiene la dirección de memoria virtual. El campo `d` permite localizar la dirección física dentro del marco que contendrá el dato de 32 bits que conforma la palabra de memoria solicitada.

c) La dirección de memoria virtual tiene un total de 12+14 = 26 bits para indicar la página (12 bits corresponden al primer nivel y 14 al segundo), y 14 bits adicionales para indicar el desplazamiento dentro de la página. Esto nos da un total de 40 bits de longitud. Por lo tanto, el espacio de memoria virtual se compone de 2^{40} palabras de memoria de 4 bytes, esto es, 4 Terabytes, organizados en 2^{26} páginas de 64 Kbytes.

d) Para el caso de una sola tabla de páginas, la tabla tiene una entrada para cada página de memoria virtual, esto es, 2^{26} entradas. Cada entrada alberga un número de marco y los 12 bits de control. Como el número de marco ocupa 20 bits, cada entrada de la tabla de páginas ocupará 32 bits (4 bytes). Recopilando todo, el espacio total ocupado por esta tabla es de $2^{26}\ entradas \times 4\ bytes/entrada = 256$ Mbytes.

e) La tabla anterior de 256 Mbytes se pagina en 4 Kpáginas de 64 Kbytes, y en cada una de esas páginas caben 16 Kmarcos, puesto que cada marco ocupa 32 bits. La segunda tabla de páginas que se incorpora direcciona a estas 4 Kpáginas con los 12 bits más significativos del campo dirección, esto es, p_1. Los 14 bits restantes, correspondientes a p_2, seleccionan la entrada dentro de dicha página, que nos dará la dirección base del número de marco que contiene los datos de la palabra en memoria física. Esta segunda tabla ocupa 4 Kentradas x 32 bits = 16 Kbytes, que ya nos cabe dentro de una sola página, y por lo tanto, no es necesario volver a paginar.

El siguiente diagrama ilustra la ubicación de las tablas que componen los índices de acceso a los marcos de página y su relación con los campos de la dirección virtual emitida desde el proceso:



---

7. El sistema de memoria de mi PC tiene las siguientes características:

- Sistema Operativo con espacio de direcciones virtuales de 64 Terabytes para cada proceso, direccionable por páginas de 4 Kbytes y palabras de un byte.
- Memoria física (DRAM) de 32 Gbytes.
- Microprocesador con TLB (Translation Look-Aside Buffer) de 4 entradas.

Se pide detallar:

a) La longitud de los campos `p` (página virtual), `f` (marco físico o *frame*), y `d` (desplazamiento) en las respectivas direcciones de memoria.

b) La estructura de la TLB, desglosando la información que contienen sus 4 entradas.

Los campos `p`, `f` y `d` de las direcciones de memoria indican el número de la página lógica, el marco físico y el desplazamiento dentro de éstos, respectivamente. La descomposición es la siguiente:

Dirección de memoria lógica o virtual: `p` + `d` bits

Número de página: Campo de longitud `p` (bits más significativos)	Desplazamiento dentro de la página: Campo de longitud `d` (bits menos significativos)

Dirección de memoria física: `f` + `d` bits

Número de marco (*frame*): Campo de longitud `f` (bits más significativos)	Desplazamiento dentro del marco: Campo de longitud `d` (bits menos significativos)

Numéricamente, los campos se corresponden con el logaritmo binario de los respectivos tamaños máximos que se pueden representar en cada caso, así que expresando las cantidades en potencias de dos veremos todo más claro:

- La página de memoria es de 2^{12} palabras de un byte, por lo que el campo desplazamiento dentro de la página tiene una longitud de `d` = **12 bits**. Este desplazamiento sirve también dentro del marco.
- La memoria física tiene 2^{35} palabras de un byte, por lo que sus direcciones físicas tienen una longitud de 35 bits. De ellos, los 12 bits menos significativos corresponden al desplazamiento `d`, y los 23 restantes indicarán el marco de memoria física, concluyendo que `f` = **23 bits**.
- La memoria virtual tiene 2^{46} palabras de un byte, por lo que sus direcciones lógicas se representan numéricamente con 46 bits. De forma similar al caso anterior, los 12 bits menos significativos son para el desplazamiento dentro de la página, quedando los `p` = **34 bits** superiores para indicar la página de memoria en la que se encuentra dicha dirección.

A partir de ahí, la traducción de cada dirección se realiza resolviendo el campo `f` que corresponde a cada valor de `p`, y conservando el mismo desplazamiento `d` en ambos casos, que se concatena a continuación si todo lo expresamos en bits.
Por otro lado, la TLB guarda las correspondencias entre `p` y `f` para las traducciones que han sido resueltas de forma más reciente y/o frecuente, por lo que sus cuatro entradas almacenarán cada una 57 bits en total: los 34 del campo `p` y los 23 del campo `f` que indican el marco de memoria física que aloja los datos de esa página. Esos campos no se registran como una tabla cualquiera dentro de la TLB: la columna correspondiente al campo `p` se implementa con una memoria asociativa en la que la *MMU (Memory Managemente Unit)* consulta en paralelo (esto es, simultáneamente) el contenido de todos los valores (cuatro en este caso), con objeto de resolver de forma instantánea la traducción en caso de que ésta se encuentre en una de las entradas de la TLB y devolver el valor del campo `f` de forma inmediata. Recordemos que el único objetivo de una TLB es acelerar el proceso de traducción, por lo que si éste no se optimiza, su concurso no aporta nada.

---

8. Sobre el PC del ejercicio anterior, en un instante determinado la TLB tiene los contenidos que se muestran a continuación:

TLB

entrada	página	marco
0	00010000000001000000000100000000001	01111111111111111111110
1	11111111111111111111111111111111111	00000000000000000000000
2	00000000000000000000000000000000000	11111111111111111111111
3	11101111111110111111111011111111110	10000000000000000000001

Las entradas de la TLB se van llenando de arriba a abajo, y se reemplazan con algoritmo FIFO. Si una entrada abandona la TLB, consideraremos que conserva sus valores si vuelve a entrar (es decir, la traducción de página a marco no ha cambiado mientras estuvo fuera de la TLB).

Sea la siguiente secuencia de direcciones de acceso a memoria solicitadas desde un proceso:

1. 11111111111111111111111111111111111111111111111
2. 00000000000000000000000000000000000000000000000
3. 00000000000000000000000000000000000001111111111
4. 11111111111111111111111111111111111110000000000

```
5. 10101010101010101010101010101010101010101010101010
6. 01010101010101010101010101010101010101010101010101
7. 11111111111111111111111111111111111111111111111110
8. 11111111111111111111111111111111111111111111111111
9. 00000000000000000000000000000000000000000000000000
```

Con todos estos elementos, se pide lo siguiente:

a) Indicar la secuencia de aciertos/fallos a TLB para cada una de las peticiones a memoria, mostrando la traducción de dirección virtual a física en el caso de los aciertos a TLB. Puedes ayudarte de la tabla adjunta para ilustrar todo el proceso.

	Petición de memoria (4 bits más altos)	¿Acierto en TLB? (Sí(*)/No)	¿A qué entrada de la TLB reemplaza? (posición y 4 bits más altos)	¿Cuál es la dirección física resultante al traducir? (colocar marco y desplazamiento)
1	**1111...**			
2	**0000...**			
3	**0000...**			
4	**1111...**			
5	**1010...**			
6	**0101...**			
7	**1111...**			
8	**1111...**			
9	**0000...**			

(*) En caso de acierto, indica entre paréntesis el número de entrada de la TLB en la que se produce.

b) Si la TLB utiliza el algoritmo LRU para reemplazar las traducciones menos usadas recientemente, muestra los resultados en otra tabla similar, comparando el resultado de esta opción frente a la anterior (en la que el reemplazo era FIFO).

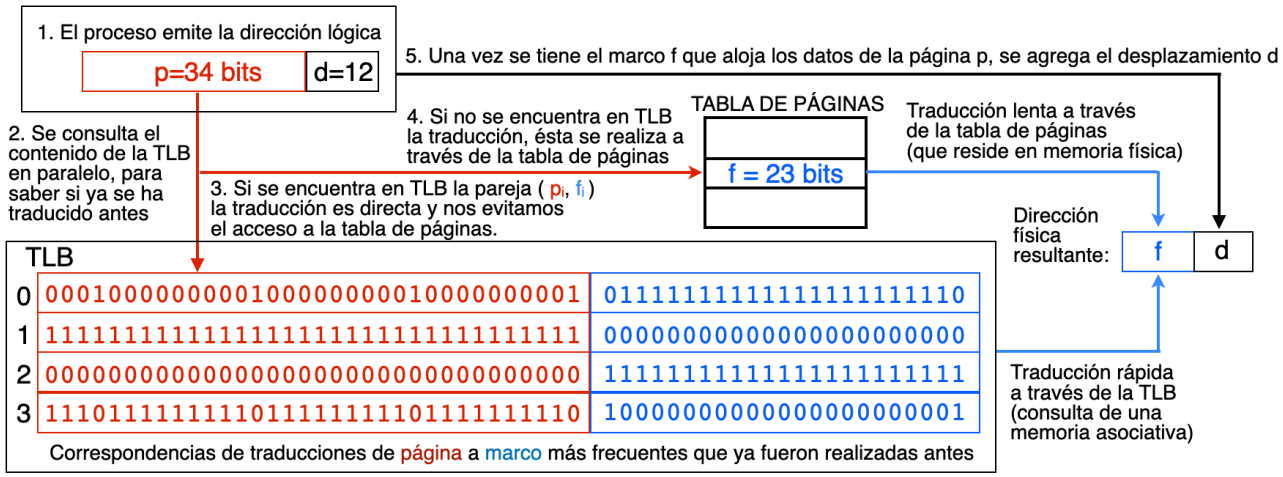
c) La TLB es tan importante para mejorar el proceso de traducción de direcciones, que cada CPU lleva una integrada. En el caso del procesador i9 de Intel, su TLB dispone de 32 entradas de 4 vías (4-way). Estas vías se corresponden con el número de entradas que hay en conjuntos (sets) predefinidos algebráicamente de forma disjunta a través de sistemas de congruencias (operaciones módulo de una división para los números de página). De esta manera, las consultas a TLB para conocer si una traducción ya se ha realizado antes y puede amortizarse, tienen lugar dentro del conjunto, lo que agiliza notablemente la respuesta de la TLB. Puesto que hay 32 entradas y 4 vías, resultan 8 conjuntos establecidos como sigue:

- Si la división del número de página por 8 devuelve resto 0, la traducción se guarda en el conjunto 0.
- Si la división por 8 devuelve resto 1, se guarda en el conjunto 1.
- ... y así sucesivamente hasta concluir con el conjunto 7.

Al direccionar al conjunto de forma directa mediante esa simple operación módulo y limitar la búsqueda dentro de él, no sólo aceleramos el proceso, sino también el coste de la memoria asociativa necesaria para consultar por contenido. Por el lado negativo, se asume el riesgo de que muchas peticiones pudieran coincidir en un mismo conjunto, aumentando el número de reemplazos (aunque esto es algo improbable según la ley estadística de los grandes números). Atrévete a experimentar aquí con esta idea, considerando una sencilla TLB de sólo 4 entradas organizadas en dos conjuntos de 2 vías: Los números de página par van al conjunto 0 y

rivalizan dentro de él (aplica reemplazo FIFO en caso de ser necesario), y los números de página impares van al conjunto 1, compitiendo por sus dos entradas de forma similar. Si nos fijamos bien en la figura anterior que mostraba los contenidos de nuestra TLB, el conjunto 1 estaría compuesto por la mitad superior de la TLB (sus dos entradas contienen números impares), y el conjunto 0 estaría en la mitad inferior (con los números de dirección par). Puedes construir una tabla similar a las anteriores, incorporando una nueva columna que refleje el conjunto al que pertenece cada petición para gestionar los posibles reemplazos dentro de él.

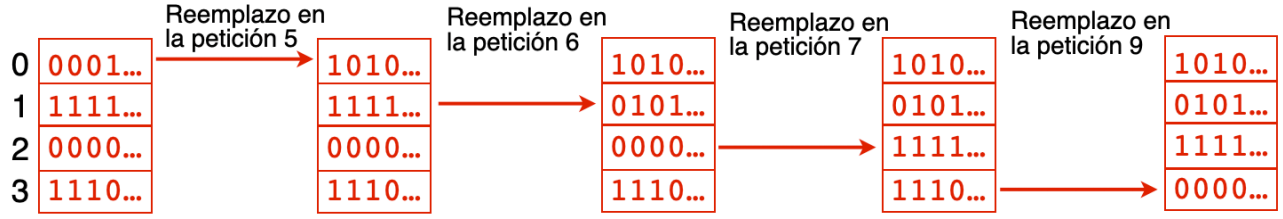
A continuación resumimos la forma de proceder para traducir las direcciones virtuales a físicas cuando la TLB apoya a la tabla de páginas:



Apartado a)
Aplicando el algoritmo de reemplazo FIFO dentro de la TLB cuando no se produce un acierto, la tabla se completa de la siguiente forma:

	Petición de memoria	¿Acierto en TLB?	¿A qué entrada de la TLB reemplaza?	Dirección física traducida si se acierta en la TLB (**marco** y **desplazamiento**)
1	**1111...**	Sí (en 1)	A ninguna	**000000000000000000000111111111111**
2	**0000...**	Sí (en 2)	A ninguna	**111111111111111111111000000000000**
3	**0000...**	Sí (en 2)	A ninguna	**111111111111111111111001111111111**
4	**1111...**	Sí (en 1)	A ninguna	**000000000000000000000110000000000**
5	**1010...**	No	A la 0, **0001...**	
6	**0101...**	No	A la 1, **1111...**	
7	**1111...**	No	A la 2, **0000...**	
8	**1111...**	Sí (en 2)	A ninguna	**000000000000000000000111111111111**
9	**0000...**	No	A la 3, **1110...**	

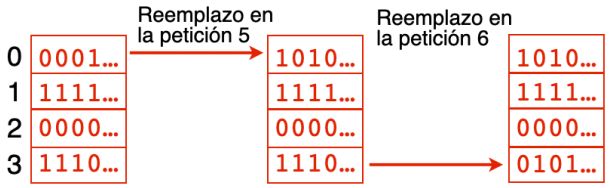
Y la secuencia de 4 reemplazos FIFO en TLB es la siguiente:



Apartado b)
Cuando la TLB reemplaza por LRU (en lugar de FIFO), la evolución es la siguiente:

	Petición de memoria	¿Acierto en TLB?	¿A qué entrada de la TLB reemplaza?	Dirección física traducida si se acierta en la TLB (**marco** y **desplazamiento**)
1	**1111...**	Sí (en la 1)	A ninguna	**000000000000000000000001111111111111**
2	**0000...**	Sí (en la 2)	A ninguna	**111111111111111111111111000000000000**
3	**0000...**	Sí (en la 2)	A ninguna	**111111111111111111111111001111111111**
4	**1111...**	Sí (en la 1)	A ninguna	**000000000000000000000001110000000000**
5	**1010...**	No	A la 0, **0001...**	
6	**0101...**	No	A la 3, **1110...**	
7	**1111...**	Sí (en la 1)	Ninguna	**000000000000000000000001111111111110**
8	**1111...**	Sí (en la 1)	Ninguna	**000000000000000000000001111111111111**
9	**0000...**	Sí (en la 2)	Ninguna	**111111111111111111111111000000000000**

Y la secuencia de reemplazos LRU en TLB es más breve, pues se producen sólo dos:

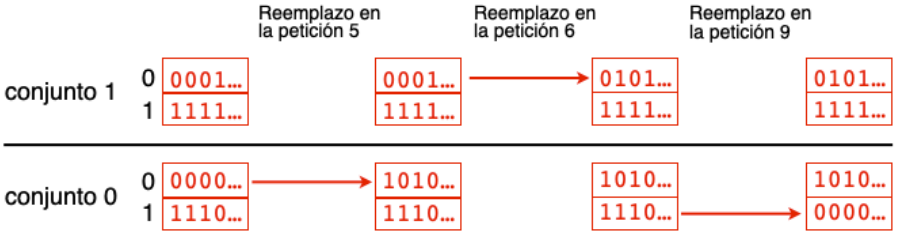


La participación de la TLB nos ahorra, por tanto, cinco accesos a la tabla de páginas, que suben a siete si adoptamos el algoritmo LRU en lugar del FIFO. En total, sólo necesitamos dos consultas a la tabla de páginas para traducir la secuencia de nueve referencias a memoria.

Apartado c)
Aquí la TLB dispone de dos conjuntos de dos vías y reemplaza por FIFO (siempre dentro del conjunto, por lo que tanto en caso de acierto como en caso de fallo esta vez indicaremos el número de la entrada dentro de éste). En total se producen tres reemplazos, tal y como indicamos a continuación:

	Petición de memoria	¿Conjunto?	¿Acierto en TLB?	Entrada TLB reemplazada	Dirección física traducida si se acierta en la TLB (**marco** y **desplazamiento**)
1	**1111...**	1	Sí (en la 1)	Ninguna	**000000000000000000000001111111111111**
2	**0000...**	0	Sí (en la 0)	Ninguna	**111111111111111111111111000000000000**
3	**0000...**	0	Sí (en la 0)	Ninguna	**111111111111111111111111001111111111**
4	**1111...**	1	Sí (en la 1)	Ninguna	**000000000000000000000001110000000000**
5	**1010...**	0	No	0. **0000...**	
6	**0101...**	1	No	0. **0001...**	
7	**1111...**	1	Sí (en la 1)	Ninguna	**000000000000000000000001111111111110**
8	**1111...**	1	Sí (en la 1)	Ninguna	**000000000000000000000001111111111111**
9	**0000...**	0	No	1. **1110...**	

La secuencia de 3 reemplazos FIFO para el caso de 2 conjuntos de 2 entradas cada uno es la siguiente:



Finalmente, y sólo como curiosidad, si la TLB de dos conjuntos de dos vías reemplazase por LRU en lugar de FIFO, puedes comprobar que se producen los mismos dos reemplazos que en la TLB convencional con reemplazo LRU (apartado b). Así que en este caso lograríamos una aceleración en las consultas a TLB sin reducir el número de aciertos. Con el reemplazo FIFO, hemos visto que la TLB de dos conjuntos sale mejor parada respecto a la TLB convencional, pues el número de reemplazos a TLB se reduce de cuatro a tres, y además los conjuntos agilizan las consultas a dicha TLB.

---

9. El sistema de memoria anterior, con los mismos valores iniciales ya expuestos para la TLB, tiene en ese mismo instante el contenido de la tabla de páginas y la memoria física que se indica a continuación:

Tabla de páginas

0	11111111111111111111111
1	11111111111111111111110
…	……
…	……
…	……
2^{34}-2	00000000000000000000001
2^{34}-1	00000000000000000000000

Memoria física (DRAM)

Marco	Desplaz.	Dato
0	0	a
	1	…
	…	…
	4095	v
1	0	b
	1	
	…	
	4095	x
…	0	…
	1	
	…	
	4095	…
2^{23}-2	0	c
	1	
	…	
	4095	y
2^{23}-1	0	d
	1	
	…	
	4095	z

Se detallan únicamente las dos primeras y las dos últimas entradas de la tabla de páginas, cuyo valor es un número de marco de 23 bits. Por otra parte, mostramos los valores de la memoria física a la derecha, indicando el dato almacenado en la primera y última posición de algunos marcos (no es necesario conocer más datos para resolver el ejercicio en su totalidad, lo cual es una pista que puede ayudarnos mucho en su resolución). Al ser una memoria direccionable a nivel de byte, los datos que contiene se han considerado valores de tipo `char` del lenguaje C, reflejando cada posición de memoria un carácter ASCII diferente (`a`, `v`, `b`, `x`, ...). Esto nos permite diferenciar el contenido de cada palabra de memoria.

En esta situación, un programa solicita la siguiente secuencia de cuatro direcciones de memoria virtual:

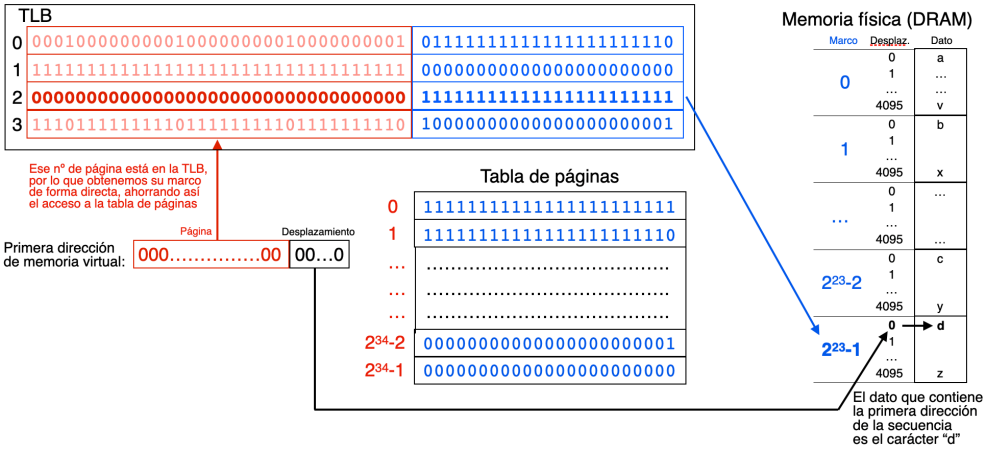
1. `00000000000000000000000000000000000000000000`
2. `00000000000000000000000000000000000111111111111`
3. `11111111111111111111111111111111110000000000000`
4. `11111111111111111111111111111111111111111111111`

Se pide encontrar el dato almacenado en cada una de esas cuatro posiciones de memoria, teniendo en cuenta tanto la consulta a la TLB como a la tabla de páginas cuando sea necesario en cada caso.

Para la primera dirección, su campo `p` que indica la página lógica está compuesto por 34 ceros, y su desplazamiento `d` por 12 ceros.

- Lo primero que hacemos es consultar en paralelo el valor de las 4 entradas rojas de la TLB, para saber si el campo `p` coincide con alguna de ellas.
- En este caso, coincide con la tercera entrada, por lo que el campo azul adyacente nos proporciona directamente el marco de memoria física donde se encuentran ubicados los datos correspondientes a esta página (que ya habría sido solicitada en un pasado muy reciente, y por ello la TLB retuvo aquí su traducción, lo que nos permite ahora reutilizarla).
- El marco almacenado en la tercera entrada de la TLB es un campo `f` de 23 bits cuyo valor resulta ser de 23 unos. Por lo tanto, esta página está en el marco numéricamente más alto, esto es, el último de todos.
- Dentro del marco anterior, usamos ahora el campo desplazamiento, `d`, para localizar el dato asociado a esa dirección. Este campo `d` tiene 12 bits, que resultan ser todos cero, por lo que se está solicitando el primer dato de ese marco, cuyo valor es el carácter "d".

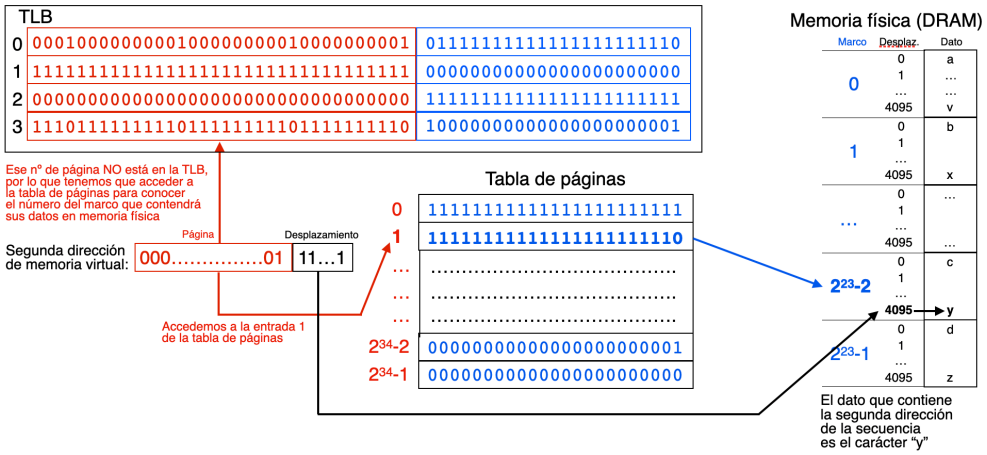
El proceso completo de traducción puede resumirse en el siguiente diagrama:



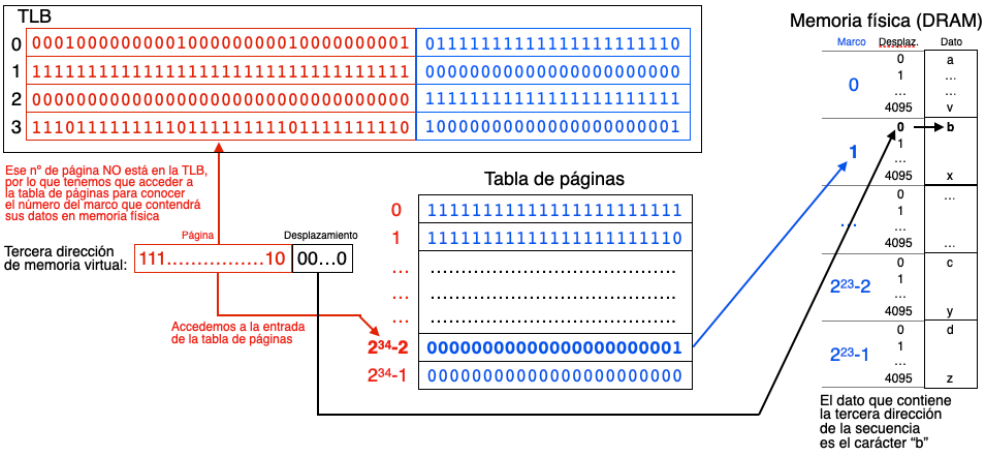
Para la segunda dirección, su campo `p` que indica la página lógica está compuesto por 33 ceros seguidos de un uno en la posición menos significativa, y su desplazamiento `d` por 12 unos.

- Consultamos la TLB, y en este segundo caso no tenemos tanta fortuna: Esa traducción no ha sido retenida por la TLB, así que hay que acceder a la tabla de páginas para conocer el marco asociado a esa página.
- A partir de ese marco, el campo desplazamiento, `d`, nos permite acceder a la dirección relativa al comienzo de ese marco. Los 12 unos de ese campo `d` indican que esta dirección es la última dentro del marco, en este caso la 4095, cuyo valor es el carácter "y".

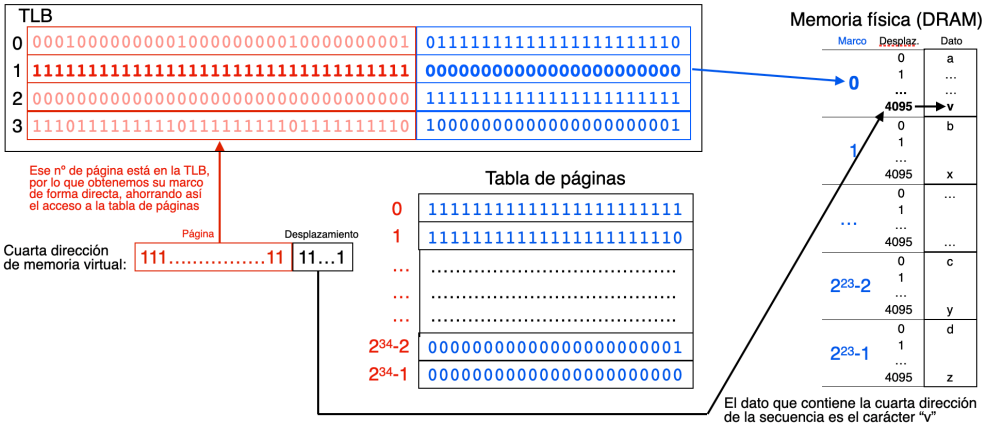
Para esta segunda dirección, el proceso de traducción se resume en el siguiente diagrama:



La tercera dirección virtual tampoco está en la TLB, por lo que el procedimiento para su traducción es análogo al de la segunda dirección virtual, que resumimos a continuación:



Por último, la cuarta dirección virtual sí acierta cuando consulta la TLB, por lo que el proceso de traducción se asemeja al de la primera dirección virtual, quedando como sigue:



Por lo tanto, los datos almacenados en las cuatro posiciones de memoria virtual indicadas en la secuencia que se nos solicita son los caracteres "d", "y", "b" y "v", por ese orden.

10. Hasta este punto hemos manejado volúmenes y direcciones de memoria realistas en el rango de los Terabytes, con la consiguiente dificultad para manejarnos con números que superan los 40 bits de longitud. En este último ejercicio usaremos direcciones más cortas que nos permitan estudiar con sencillez el alojamiento y la fragmentación de la memoria. Considera para ello un espacio físico de memoria de tan sólo 64 Kbytes con páginas de 4 Kbytes, y un espacio lógico para cada proceso de 128 Kbytes con segmentación paginada. En un momento dado, consideremos que sólo 40 Kbytes de esa memoria se encuentran ocupados por un único proceso, cuya tabla de páginas adjunta revela el uso que hace de la memoria.

	Tabla de páginas
0	0001
1	0110
2	0000
3	0100
4	1111
5	0010
6	0101
7	1011
8	0111
9	1100

Se pide lo siguiente:

a) Ilustra cómo asignaría el Sistema Operativo los restantes 24 Kbytes a un programa que ocupe 22 Kbytes divididos en sus 3 segmentos típicos:

- Segmento de código: 5 Kbytes.
- Segmento de datos: 10 Kbytes.
- Segmento de pila: 7 Kbytes.

Considera que se asigna primero el marco de memoria libre numéricamente más bajo, y que los segmentos se alojan en el orden (1) código, (2) datos y (3) pila.

b) En realidad, el programa sólo necesita alojar su segmento de código y su segmento de datos para echar a andar (los delimita con antelación el compilador), mientras que el segmento de pila arranca desde cero y va creciendo en tiempo de ejecución según se llama a funciones y librerías. Más realista resulta, por tanto, no asignar un tamaño fijo a este segmento de pila, y considerar que crecerá hasta donde le permita el Sistema Operativo. Olvídate por tanto del tamaño de 7 Kbytes dado para este último segmento e indica cuál sería el máximo tamaño que podría ocupar durante la ejecución del programa si el Sistema Operativo sólo le deja crecer mientras quede memoria física libre.

c) Indica cuánto espacio se desperdicia en el caso anterior por fragmentación interna y externa.

d) Compara la fragmentación anterior con el caso de que redujéramos el tamaño de página a la mitad (es decir, 2 Kbytes) para tu programa (el proceso que ya ocupaba la memoria conserva el mismo espacio que tenía asignado). ¿Qué tamaño máximo podría ocupar el segmento de pila en este caso?

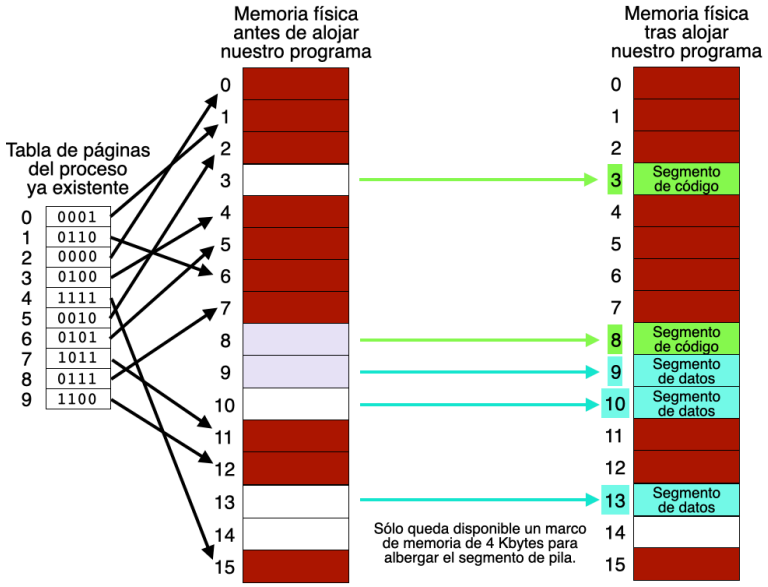
e) Supongamos que el sistema de memoria no está paginado, y que por tanto la asignación de memoria debe hacerse de forma contigua. Tomando el mismo punto de partida (40 Kbytes de memoria ocupados por el proceso inicial y 24 Kbytes libres que ahora no están organizados en marcos), indica cómo llenarían la memoria los tres segmentos alojándose en el orden código-datos-pila, con tamaños de 5, 10 y 4 Kbytes, respectivamente. Considera primero *best-fit* como algoritmo para el alojamiento de memoria no contigua, e ilustra después las diferencias respecto a *first-fit* y *worst-fit*.

f) ¿Hasta qué tamaño podría crecer la pila en tiempo de ejecución en este sistema de memoria no contigua? (toma como referencia el alojamiento con *best-fit*)

g) ¿Qué algoritmo(s) de los tres estudiados (*best-fit*, *first-fit* y *worst-fit*) no consigue(n) alojar completamente en memoria los tres segmentos de tu programa? ¿qué fenómeno adverso sufre el Sistema Operativo para no poder completar el alojamiento de tu programa aún existiendo memoria suficiente en los 24 Kbytes de memoria disponibles?

a) La tabla de páginas del proceso existente revela el uso de la memoria que mostramos a la izquierda en el diagrama anexo, y que corresponde al punto de partida para resolver el ejercicio. A partir de ahí, los tres segmentos de tu programa se alojan en el orden código-datos-pila, por lo que la secuencia de ocupación de los marcos de memoria es la siguiente:

1) El segmento de código necesita dos marcos de 4 Kbytes para alojar sus 5 Kbytes. Ocupará los marcos 3 y 8, que son los primeros disponibles.
2) El segmento de datos necesita tres marcos para alojar sus 10 Kbytes, que serán los números 9, 10 y 13.
3) El segmento de pila no puede alojarse en su totalidad, ya que tiene 7 Kbytes y sólo queda disponible el marco 14, cuyo tamaño es de 4 Kbytes.



b) El máximo espacio que puede ocupar el segmento de pila es de 4 Kbytes, una vez que el segmento de código ha ocupado 2 marcos y el de datos los 3 marcos siguientes, tal y como muestra la parte inferior derecha del diagrama anterior.

c) El segmento de código ocupa 8 Kbytes (2 marcos) pero sólo necesita 5 Kbytes, por lo que desperdicia 3 Kbytes por fragmentación interna de los 4 Kbytes que le proporciona su segundo marco. De forma similar, el segmento de datos ocupa 12 Kbytes (3 marcos) y desperdicia 2 Kbytes en su tercer marco por fragmentación interna. En total, se pierden 3+2=5 Kbytes de memoria, que son los que evitan que el segmento de pila de 7 Kbytes considerado inicialmente pueda alojarse en memoria. En este caso, no entra en juego la fragmentación externa, que tiene lugar cuando el sistema de memoria no está paginado.

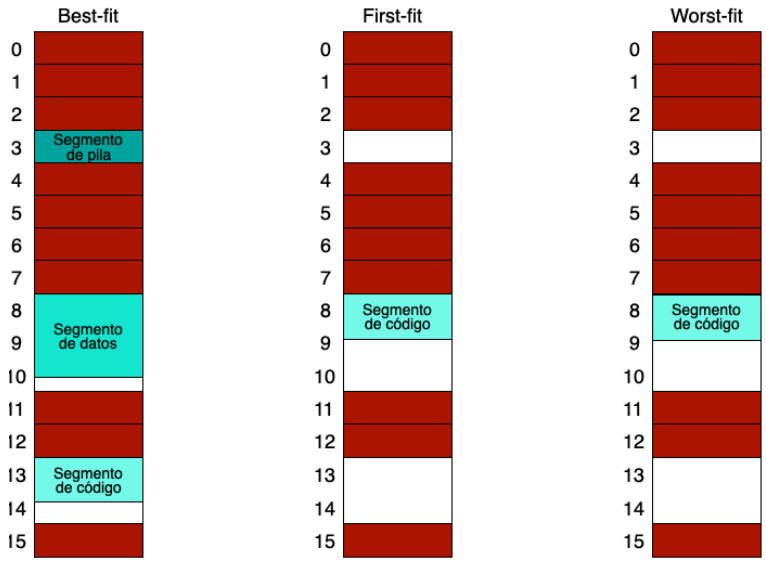
d) Si reducimos el tamaño de página de 4 Kbytes a 2 Kbytes:

- El segmento de código ocuparía 6 Kbytes (3 marcos, con pérdida de 1 Kbyte en el último marco por fragmentación interna).
- El segmento de datos ocuparía 10 Kbytes (5 marcos completos).

- El segmento de pila podría ocupar hasta 8 Kbytes (4 marcos de 2 Kbytes), puesto que el total disponible son 24 Kbytes (12 marcos, y sólo hemos ocupado 3+5=8).

En este caso, sólo se desperdiciaría 1 Kbyte por fragmentación interna, que se produce en el último marco del segmento de código. Los otros dos segmentos ocupan exactamente un múltiplo del tamaño de página, y por lo tanto, aprovechan íntegramente su memoria.

e) El único algoritmo que consigue alojar los tres segmentos en memoria es *best-fit*, que busca entre todos los huecos disponibles aquel que mejor se ajusta al tamaño de cada segmento. Tanto *first-fit* como *worst-fit* asignan al segmento de código el bloque de 12 Kbytes, y como es el único que puede albergar el segmento de datos de 10 Kbytes, cuando éste llega después, no encuentra un hueco suficientemente grande para alojarse. La siguiente ilustración ubica todos los alojamientos que pueden completarse en cada caso.



f) El segmento de pila no puede ocupar más espacio de los 4 Kbytes que ya tiene asignados con *best-fit*, pues corresponde exactamente al hueco que tiene asignado.

g) La fragmentación externa que sufre el alojamiento contiguo de memoria es lo que impide que, aunque exista más espacio disponible del que se solicita, no podamos alojar los tres segmentos del programa usando los algoritmos *first-fit* o *worst-fit*. Habría que realizar una compactación de los huecos (fusionar previamente todos ellos en un solo bloque) para que esto fuese posible.