✎ Sea una memoria caché en la que cada 2^{20} accesos se producen 2^{17} fallos. Calcule la tasa de fallos.

$$\text{Tasa aciertos } h = \frac{\text{Nº de aciertos}}{\text{Nº de peticiones}} \cdot 100\% = \frac{(\text{Nº de peticiones}) - (\text{Nº de fallos})}{\text{Nº de peticiones}} \cdot 100\% = \frac{2^{20} - 2^{17}}{2^{20}} \cdot 100\% = 87.5\%$$

$$\text{Tasa fallos} = 100\% - h = 12.5\%$$

✎ Sea una memoria caché en la que se realizan 2^{30} accesos con una tasa de aciertos de1 75%. Calcule el número de fallos.

$$75\% = \frac{\text{Nº de aciertos}}{\text{Nº de accesos}} \cdot 100\% \Rightarrow 75\% = \frac{(\text{Nº de accesos}) - (\text{Nº de fallos})}{\text{Nº de accesos}} \cdot 100\% \Rightarrow \text{Nº de fallos} = 0.25 \cdot (\text{Nº de accesos}) = 0.25 \cdot 2^{30} = 2^{-2} \cdot 2^{30} = 2^{28}$$

✎ Sea un sistema jerárquico de memoria caché con un tiempo de acceso de 10*ns*. y una memoria principal con un tiempo de acceso de 100*ns*. La tasa de acierto es del 90%. Calcule el tiempo de acceso medio del conjunto en cada uno de estos dos diseños:

a) Cuando se produce un fallo, primero se mueve el bloque completo a la caché y después se lee desde la caché.

b) Cuando se produce un fallo, se mueve el dato a la UCP y, simultáneamente, se mueve el bloque a la memoria caché.

Previamente a las posibles lecturas en la caché se comprueba si la dirección solicitada por la UCP ya está en la caché, mediante una búsqueda asociativa entre las etiquetas (muy rápida, de tiempo despreciable).

a) En el primer nivel se lee siempre: (Cantidad de accesos al nivel 1)≡ 100 de cada 100.

$$T_A = \frac{(\text{Nº de accesos al nivel } 1) \cdot t_{A1} + (\text{Nº de accesos al nivel } 2) \cdot t_{A2}}{100} = \frac{100 \cdot t_{A1} + (100 - T) \cdot t_{A2}}{100} = \frac{(100 \text{ veces} \cdot 10\ ns) + (10 \text{ veces} \cdot 100\ ns)}{100 \text{ veces}} = (1 \cdot 10\ ns) + (0.1 \cdot 100\ ns) = 20\ ns$$

b) En el primer nivel se lee sólo cuando hay acierto: (Cantidad de accesos al nivel 1)≡ *T* de cada 100.

$$T_A = \frac{(\text{Nº de accesos al nivel } 1) \cdot t_{A1} + (\text{Nº de accesos al nivel } 2) \cdot t_{A2}}{100} = \frac{90 \cdot t_{A1} + (100 - T) \cdot t_{A2}}{100} = \frac{(90 \text{ veces} * 10\ ns) + (10 \text{ veces} * 100\ ns)}{100 \text{ veces}} = (0.9 * 10\ ns) + (0.1 * 100\ ns) = 19\ ns$$

✎ Sea un sistema jerárquico de memoria está compuesto por una memoria caché dividida en particiones de 8 palabras y con un tiempo de acceso de 10 *nseg*; y por una memoria principal de con un tiempo de acceso de 100 *nseg*. La tasa de acierto de la caché es del 90%. Calcule el tiempo de acceso medio del conjunto en cada uno de estos dos diseños:

a) Cuando se produce un fallo, primero se mueve el bloque completo a la caché y después se lee desde la caché.

b) Cuando se produce un fallo, se mueve el dato a la UCP y, simultáneamente, se mueve el bloque a la memoria caché.

$$T_A = \frac{(\text{Nº de accesos al nivel } 1) \cdot t_{A1} + (\text{Nº de accesos al nivel } 2) \cdot t_{A2}}{100}$$

Para contar la cantidad de accesos al nivel 2 se ha de tener en cuenta que cuando se produce un fallo, se mueve un bloque de 8 palabras desde la memoria de segundo nivel hasta la de primer nivel.

En ambos diseños, en el segundo nivel siempre se lee 8 veces por cada fallo: (Cantidad de accesos al nivel 2)≡ (100-*T*)·8.

a) En el primer nivel se lee siempre: (Cantidad de accesos al nivel 1)≡ 100 de cada 100.

$$T_A = \frac{(\text{Nº de accesos al nivel } 1) \cdot t_{A1} + (\text{Nº de accesos al nivel } 2) \cdot t_{A2}}{100} = \frac{100 \cdot t_{A1} + (100 - T) \cdot 8 \cdot t_{A2}}{100} = \frac{100 \cdot 10 \cdot 10^{-9} + (100 - 90) \cdot 8 \cdot 100 \cdot 10^{-9}}{100} = 90 \cdot 10^{-9}\ seg$$

b) En el primer nivel se lee sólo cuando hay acierto: (Nº de accesos al nivel 1)≡ *T*.

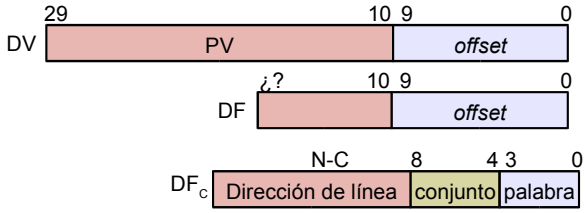
$$T_A = \frac{(\text{Cantidad de accesos al nivel } 1) \cdot t_{A1} + (\text{Cantidad de accesos al nivel } 2) \cdot t_{A2}}{100} = \frac{T \cdot t_{A1} + (100 - T) \cdot 8 \cdot t_{A2}}{100} = \frac{90 \cdot 10 \cdot 10^{-9} + (100 - 90) \cdot 8 \cdot 100 \cdot 10^{-9}}{100} = 89 \cdot 10^{-9}\ seg$$

Sea un computador con un mecanismo de memoria virtual paginada, con un espacio virtual de 1 Gbyte y con páginas de 1 Kbyte. El computador incluye una memoria caché que almacena líneas de 16 bytes. La caché está organizada en 32 conjuntos de 4 líneas cada uno. Describa cómo se realizaría la traducción virtual y la búsqueda en la caché de forma que la información solicitada por el procesador fuera recibida lo más pronto posible. Trace un esquema con el proceso total: desde que el procesador emite la dirección virtual hasta que recibe el dato.

| EV | = 1 Gbyte = 2^{30} bytes $\Rightarrow$ || DV || = 30 bits.
| P | = 1 Kbytes = 2^{10} bytes $\Rightarrow$ || offset || = 10 bits.

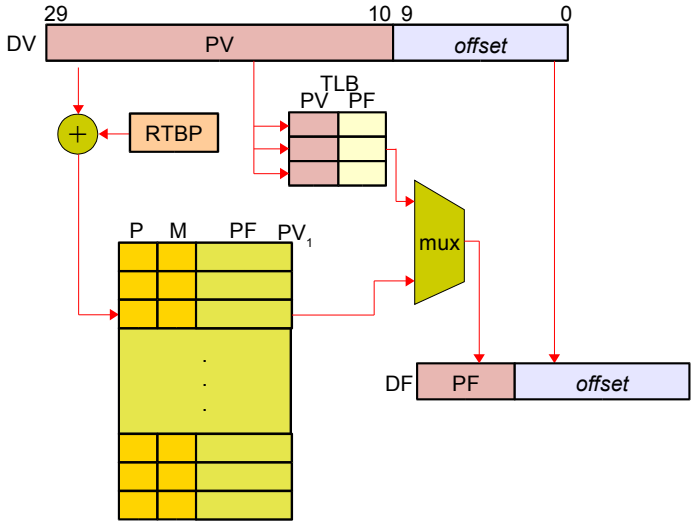
Memoria caché:

| líneas | = 16 bytes = 2^4 bytes $\Rightarrow$ 4 bits por palabra.
2^4 conjuntos $\Rightarrow$ 4 bits por conjunto, con 4 líneas en cada conjunto.



Para la traducción de dirección virtual a dirección física. Suponemos el algoritmo de ubicación asociativa (que es más rápida que la directa) o en dos niveles.

Mecanismo de traducción para obtener la DF:



Una vez obtenida la DF, la buscamos en la memoria caché de tal forma que en el directorio caché almacenamos *N-C*; y en la zona de almacenamiento registramos la línea completa *N*.

Sea un sistema con memoria virtual paginada con un espacio lógico de 1 Kpalabras. El sistema dispone de una memoria física de 64 palabras ocupada por un único proceso. En dicho espacio físico se pueden almacenar 4 páginas. El esquema de traducción es directo y está auxiliado por una *TLB* de 2 entradas. Inicialmente las páginas virtuales

`05`, `1A`, `1F` y `09` (en notación hexadecimal)

se encuentran en memoria principal ocupando las páginas físicas:

`0`, `1`, `2` y `3` respectivamente.

La TLB contiene los pares (dirección virtual, dirección física):

(`1F`, `02`) y (`05`, `00`).

El sistema utiliza un algoritmo de reemplazo LRU cuya cola es:

(`05`, `1F`, `09`, `1A`).

Además, el sistema dispone de una memoria caché de asignación asociativa por conjuntos (2 conjuntos) que permite almacenar 4 líneas (2 por conjunto) de 4 palabras cada una. Utiliza un algoritmo de reemplazo LRU para cada conjunto. La caché contiene en su directorio las siguientes entradas (Las dos primeras en el conjunto 0 y las dos últimas en el conjunto 1):

`0`, `7`, `3` y `4`.

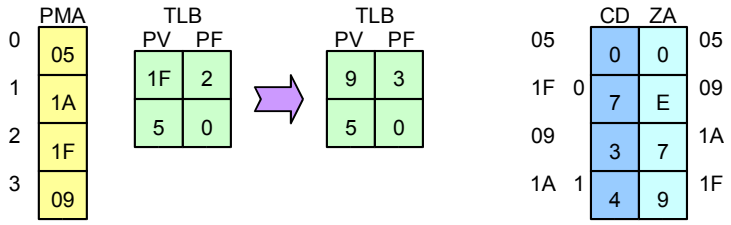
Supongamos que:

- ✓ El tiempo de búsqueda en la *TLB* es T_d unidades de tiempo.
- ✓ El acceso a la tabla de páginas es 50 veces más lento.
- ✓ El tiempo de acceso a la información de la caché (obtención de una palabra) es T_i unidades temporales.
- ✓ El tiempo de acceso a memoria principal (lectura de una palabra) es $20 \cdot T_i$.
- ✓ El tiempo de acceso a disco (lectura del disco y escritura en MP) es $20000 \cdot T_i$.
- ✓ El sistema permite el acceso simultáneo a la caché y a la memoria principal.

Dada la secuencia de direcciones virtuales (en notación hexadecimal) `53`, `98`, `58` y `BA`:

a) Muestre en cada referencia, el tiempo necesario para que el procesador reciba la palabra solicitada.
b) Muestre la evolución de la *TLB*, *PMA* y caché; así como las direcciones físicas traducidas.

| EV | = 1 K palabras
| EF | = 64 palabras
| líneas | = 4 palabras
| páginas | = 16 palabras

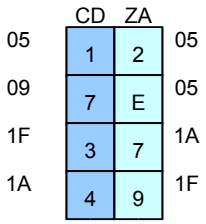


TLB → T_d
TP → $50 \cdot T_d$
Caché → T_i
MP → $20 \cdot T_i$
Disco → $20000 \cdot Ti$

1.- 53 DV 0101 0011 → DF 000000
Acierto TLB
Acierto caché
$T = T_d + T_i$

2.- 98 DV 1001 1000 → DF 001110
Fallo TLB
Está en MP
Acierto caché
$T = 50 \cdot T_d + T_i$

3.- 98 DV 0101 1000 → DF 000010
Acierto TLB
Fallo caché
$T = T_d + 20 \cdot T_i$



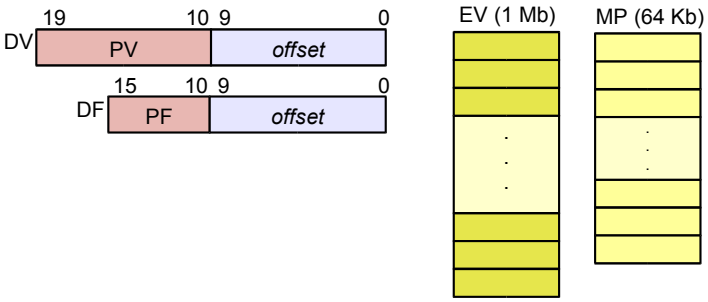
Sea un computador con una jerarquía de memoria de dos niveles gestionada mediante un sistema de memoria virtual paginado. El espacio físico es de 64 Kbytes, mientras que el espacio virtual es de 1 Mbyte. La memoria principal (*MP*) cubre todo el espacio físico. El tamaño de página es de 1 Kbyte y se utiliza un esquema de traducción virtual a físico híbrido, haciendo uso de una memoria asociativa (*TLB*) capaz de almacenar tres pares (*PV*, *PF*). Cada entrada de la tabla de páginas (TP) incluye, además de la PF, los dos bits siguientes: bit modificado (*M*) y bit *Mp/Ms* (si la página se encuentra en *MP* o no) (*P*). En caso de ser requerido un reemplazo, se utiliza el algoritmo *LRU*. Analice cómo evolucionan los contenidos de la memoria principal, la *TP* y la *TLB* cuando el procesador emite la siguiente secuencia de direcciones virtuales (Las letras `R` y `W` al lado de cada dirección virtual indican si el procesador pretende leer o escribir, respectivamente en dicha dirección virtual):

`2A307 (R), 2A75B (R), 2A85B (W), 2A09F (W), 2AC80 (W), 2A441 (R).`

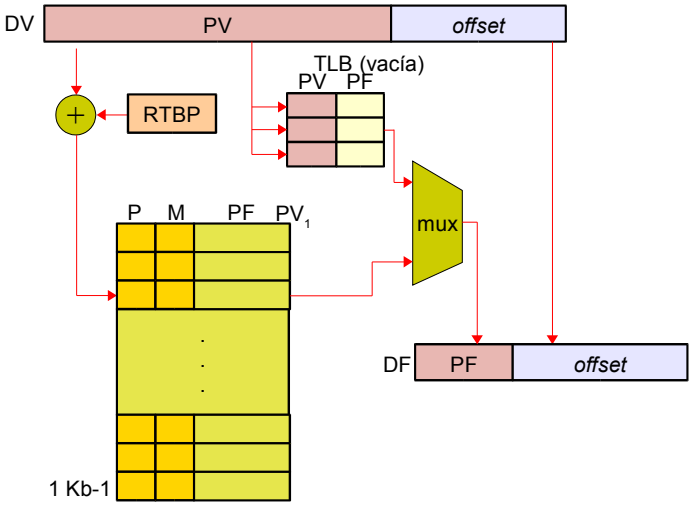
|EF| = |MP| = 64 Kb = 2^{16} bytes ⇒ || DF || 16 bits

|EV| = 1 Mb = 2^{20} ⇒ || DV || = 20 bits

|página| = 1 Kb = 2^{10} ⇒ || offset || = 10 bits



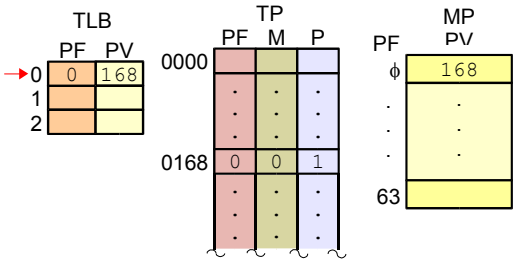
Traducción híbrida:



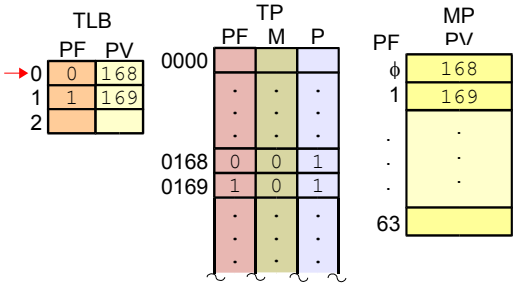
DV: 2A307 (R), 2A75B (R), 2A85B (W), 2A09F (W), 2AC80 (W), 2A441 (R).

DV (hex)	DV (binario)	PV		
	PV	offset	(decimal)	
2A307	0010 1010 00	11 0000 0111	168	(R)
2A75B	0010 1010 01	11 0101 1011	169	(R)
2A85B	0010 1010 10	00 0101 1011	170	(W)
2A09F	0010 1010 00	00 1011 1111	168	(W)
2AC80	0010 1010 11	00 1000 0000	171	(W)
2A441	0010 1010 01	00 0100 0001	169	(W)

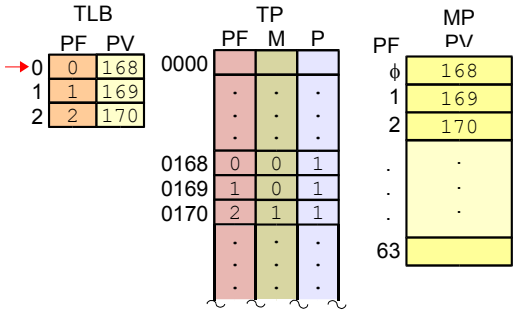
✔ 2A307 (R) →PV: 168 M=0, pues es (R). Hay fallo de página



✔ 2A75B (R) →PV: 169 Hay fallo de página



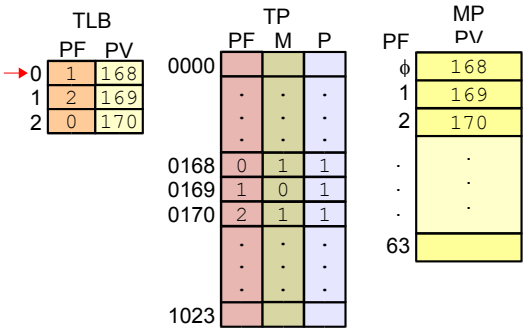
✔ 2A85B (W) →PV: 170 M=1, pues es (W). Hay fallo de página



✔ 2A09F

Número de página virtual (168_d) Offset dentro de la página
010 1010 00 00 1001 1111

La página 168 ya estaba almacenada en TP y en TLB, no hay fallo de página. Únicamente cambia el bit de modificación en la tabla de páginas, pues la página 168, que antes se referenció para leer en ella, ahora es para escribir. Otro cambio es en la TLB: tiene lugar una reordenación para reflejar que la 168 es hasta ahora la última referenciada.



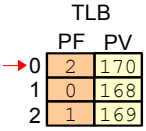
✔ 2A080 (W) No hay fallo de página

Número de página virtual (168_d) Offset dentro de la página
010 1010 00 00 1000 0000

Vuelve a ser pedida la misma página anterior; y además para la misma operación. Ninguna tabla es modificada.

✔ 2A441 (R) No hay fallo de página

Número de página virtual (169_d) Offset dentro de la página
010 1010 01 00 0100 0001



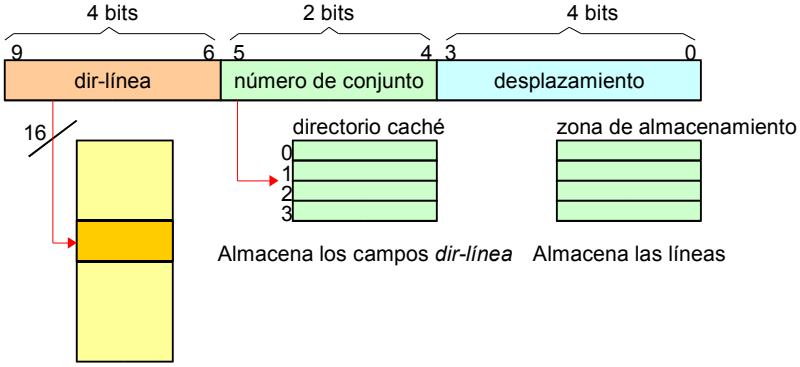
Suponga un computador con una memoria principal (MP) de 1 Kbyte, siendo ésta la máxima capacidad de memoria física que puede direccionar el ordenador. La memoria de este sistema está gestionada mediante un esquema de memoria virtual paginada, con un tamaño de página de 128 bytes y con direcciones virtuales de 16 bits. Además, consta de una memoria caché capaz de almacenar 8 líneas, cada una de ellas de 16 bits. El procesador, en un momento dado, empieza a ejecutar un programa que consta de 5 páginas virtuales (PV), encontrándose todas ellas en MP. La tabla de páginas consta de los siguientes pares :

`(PV, PF): (0, 3), (1, 7), (2, 0), (3, 2), (4, 1).`

Se usa un esquema de traducción directa. Muestre la evolución del contenido de la caché (tanto en su zona de almacenamiento como en su directorio), a nivel de línea, a medida que el procesador solicita la siguiente secuencia de direcciones virtuales (en notación hexadecimal):

`106, 132, 152, 117, 14B, 142, 79, 6B, 11, 26, 31, 250, 1B9, 1C4.`

Resuelva el ejercicio suponiendo que las líneas se ubican en la caché según el esquema asociativo por conjuntos (4 conjuntos). Suponga que si hay reemplazar alguna línea de la caché, será la más antigua de entre todas las candidatas (la primera en ser cargada, FIFO). Haga un recuento de los fallos de página.



Funcionamiento:

1 El procesador genera una DV.
2 El sistema de MV traduce la DV a DF.
3 El sistema caché busca en la caché. Si no la encuentra, manda la DF a la MP y recoge una copia de la línea correspondiente.

Traducción de dirección virtual a dirección física; y determinación de la línea:

DV	PV	offset
106	00010	000 0110
132	00010	011 0010
152	00010	101 0010
117	00010	001 0111
14B	00010	100 1011
142	00010	100 0010
79	00000	111 1001
6B	00000	110 1011
11	00000	001 0001
26	00000	010 0110
31	00000	011 0001
250	00100	101 0000
1B9	00011	011 1001
1C4	00011	100 0100

PV	PF
2	0
2	0
2	0
2	0
2	0
2	0
0	3
0	3
0	3
0	3
0	3
4	1
3	2
3	2

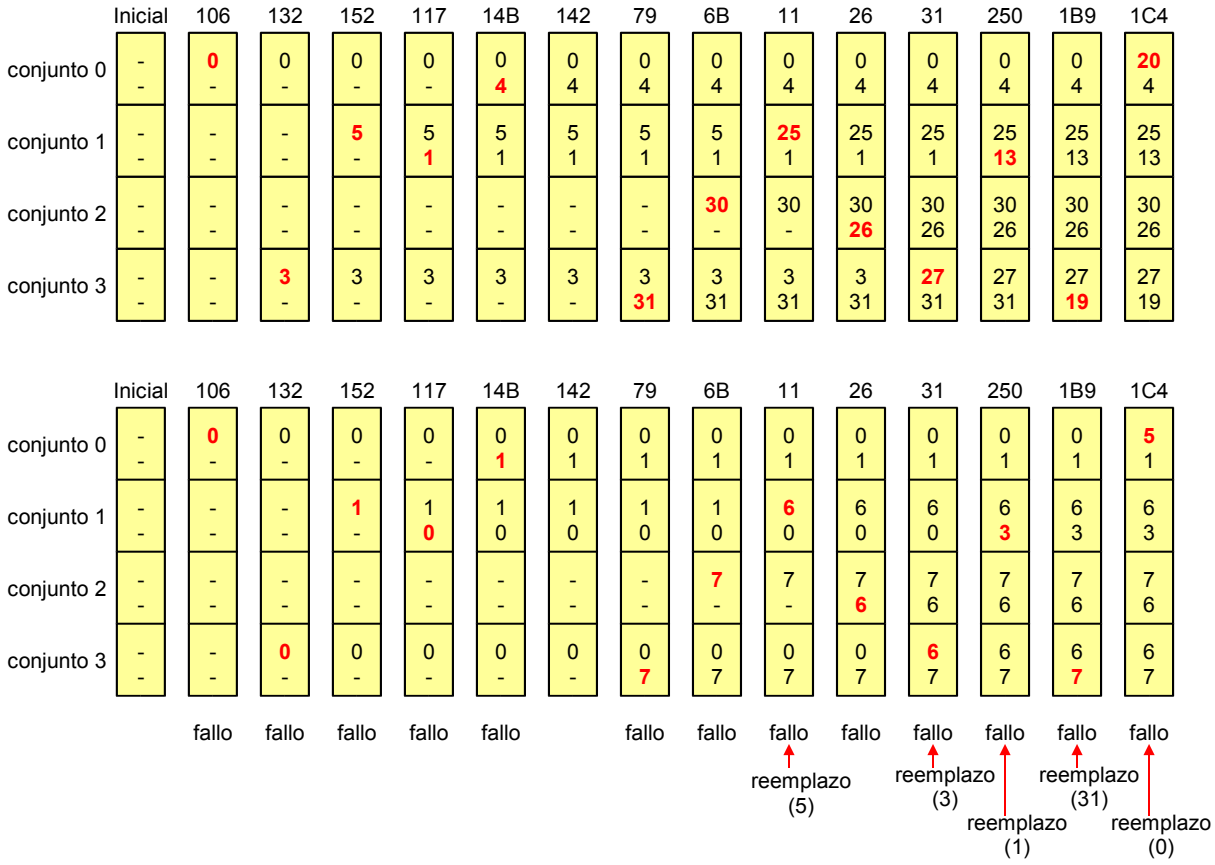
PF	offset	
0000	00	0110
0000	11	0010
0001	01	0010
0000	01	0111
0001	00	1011
0001	00	0010
0111	11	1001
0111	10	1011
0110	01	0001
0110	10	0110
0110	11	0001
0011	01	0000
0100	11	1001
0101	00	0100

DC
dir-línea C *offset*
ZA línea
DF

línea ZA	dir-línea DC	C
0	0	0
3	0	3
5	1	1
1	0	1
4	1	0
4	1	0
1F	7	3
1E	7	2
19	6	1
1A	6	2
1B	6	3
0D	3	1
13	4	3
14	5	0

Evolución de la caché.

Zona de almacenamiento. Línea física que contiene cada línea caché.



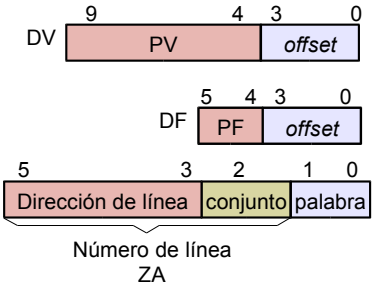
Sea un computador con un mecanismo de memoria virtual paginada, con un espacio lógico de 1 Kpalabras. El sistema dispone de una memoria física de 64 palabras ocupada por un único proceso. En dicho espacio físico se pueden almacenar 4 páginas. El esquema de traducción es directo y está auxiliado por una *TLB* de dos entradas. Inicialmente se encuentran en memoria principal las páginas virtuales `5`, `1A`, `1F` y `9` (en base hexadecimal) ocupando las páginas físicas `0`, `1`, `2` y `3` respectivamente. La *TLB* contiene los pares dirección virtual-dirección física (`1F`, `2`) y (`5`, `0`). El sistema utiliza un algoritmo de reemplazo *LRU* cuya cola es (`5`, `1F`, `9`, `1A`). Además, el sistema dispone de una memoria caché con ubicación asociativa por conjuntos (dos conjuntos) que permite almacenar 4 líneas (2 por conjunto) de 4 palabras cada una. Dicha caché utiliza un algoritmo de reemplazo LRU para cada conjunto. La caché contiene en su directorio las siguientes entradas: `0`, `7`, `3` y `4` (las dos primeras en el conjunto `0` y las dos últimas en el conjunto `1`). Supongamos que:

- ✔ El tiempo de búsqueda en la *TLB* es T_d unidades de tiempo.
- ✔ El acceso a la tabla de páginas es 50 veces más lento.
- ✔ El tiempo de acceso a la información de la caché (obtención de una palabra) es T_i unidades temporales.
- ✔ El tiempo de acceso a memoria principal (lectura de una palabra) es $20 \cdot T_i$.
- ✔ El tiempo de acceso a disco (lectura del disco y escritura en *MP*) es $2 \cdot 10^4 \cdot T_i$.
- ✔ El sistema permite el acceso simultáneo a la caché y a la memoria principal.

Dada la secuencia de direcciones virtuales (en base hexadecimal) `53`, `98`, `58` y `BA`: Analice la evolución de la *TLB*, *PMA* y caché; así como las direcciones físicas traducidas; calculando en cada referencia, el tiempo necesario para que el procesador reciba la palabra solicitada.

| EV | = 1 Kbyte = 2^{10} bytes $\Rightarrow$ || DV || = 10 bits.
| EF | = 64 bytes = 2^6 bytes $\Rightarrow$ || DF || = 6 bits.
Cantidad de conjuntos = 2^1 conjuntos $\Rightarrow$ 1 bit de conjunto.
Cantidad de palabras = 2^2 palabras $\Rightarrow$ 1 bit de palabra.
Cantidad de páginas = 4 páginas.

$$|P| = \frac{64\ bytes}{4\ páginas} = 16\ \frac{bytes}{páginas} \Rightarrow \|offset\| = 4\ bits$$



Estado inicial:

PV	PF
05	0
1A	1
1F	2
09	3

TLB

PV	PF
1F	2
05	0

caché

	0001	1011
Conjunto 0	0	0
	7	E
Conjunto 1	3	7
	4	9
	DC	ZA

Completamos la caché:

`DC C` `000, 0` $\Rightarrow$ `ZA=0` (ZA)

`DC C` `111, 0` $\Rightarrow$ `ZA=E` (ZA)

`DC C` `011, 1` $\Rightarrow$ `ZA=7` (ZA)

`DC C` `100, 1` $\Rightarrow$ `ZA=9` (ZA)

Dirección solicitada Nº 1: `53`.

`0101` `0011`
`PV5` *offset*

PV5 ↔ PF0. Está en la TLB $\Rightarrow$ No ocurre fallo de página. El tiempo consumido es T_d .

`DF0` *offset*
`00 0 0 11`
conjunto
`ZA=0`

Está en la caché $\Rightarrow$ No ocurre fallo de línea. El tiempo consumido es T_i .
Así pues, el tiempo total es $T = T_d + T_i$ (el de acceso a la TLB y a la caché para examinarla).

Dirección solicitada Nº 2: `98`.

`1001` `1000`
`PV9` *offset*

PV	PF
05	0
1A	1
1F	2
09	3

Sí está en la PMA: PV9 ↔ PF3.
No está en la TLB $\Rightarrow$ Ocurre un fallo de página.
Actualizamos la TLB.

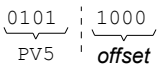
`PV3` *offset*
`11 1 0 00`
conjunto
`ZA=E`

Observamos que está presente en la caché, en la palabra 00 $\Rightarrow$ No hay fallo caché.

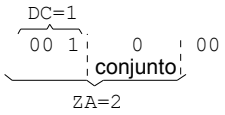
Tiempo total = Tiempo de acceso a la TP + Tiempo de acceso a la caché: $T = 50\,T_d + T_i$

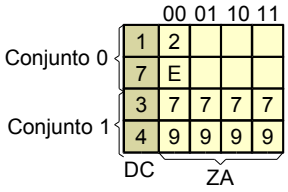
	00	01	10	11
Conjunto 0	0			0
	7	E		
Conjunto 1	3	7	7	7
	4	9	9	9
	DC	ZA		

Dirección solicitada Nº 3: `58`.



PV5 está en la TLB ⇒ No ocurre fallo de página.
Veamos en la caché.

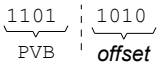




Fallo de línea. Vayamos a la MP. Reemplazo LRU.

Tiempo total = Tiempo de acceso a la TLB + Tiempo de acceso a MP: $T = T_d + 20\,T_i$

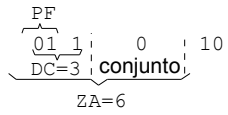
Dirección solicitada Nº 4: `BA`.



PVB no está en la TLB ⇒ Ocurre fallo de página en TLB.
PVB no está en la TP ⇒ Ocurre fallo de página en TP. Es necesario ir al disco (y usar el algoritmo LRU).

Cola: {5, 9, 1F, 1A}. Reemplazamos la 1A.
$T = 50\,T_d + 50\,T_i + 20000\,T_i$

$50\,T_d$: Actualización de la TP.
$50\,T_i$: Fallo de línea.
$20000\,T_i$: Fallo de página en la TLB y en TP; búsqueda en la memoria secundaria.

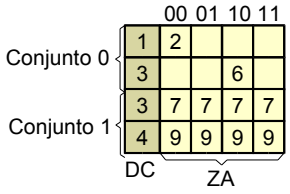


Fallo en la caché; actualizamos.

PV	PF
05	0
0B	1
1F	2
09	3

TLB

PV	PF
0B	1
05	0



Sea un computador con una memoria caché de una capacidad de 8 líneas, con un tamaño de línea de 32 bytes; y una memoria principal de 64 Kbytes. El computador incorpora un esquema de gestión virtual paginada entre la memoria principal y la secundaria, con un espacio virtual de 1 Mbytes y un tamaño de página de 128 bytes. En un instante determinado se encuentran residentes en la memoria principal las páginas virtuales 18, 21 y 23; ocupando las páginas físicas 3, 31 y 12, respectivamente. Tras ese instante, el procesador emite las dos direcciones virtuales (en base hexadecimal) `00AA9` y `00BE5`.

Denotemos por:

- *P* el tiempo que consume la traducción virtual.
- *Q* el tiempo de acceso al directorio caché.
- *R* el tiempo de comparación entre el contenido del directorio y lo que solicita el procesador

Tengamos los siguientes supuestos:

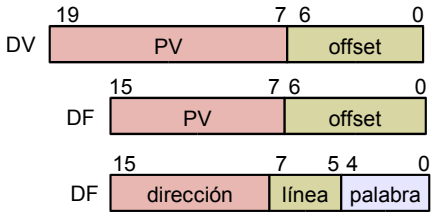
i. Algoritmo de ubicación directa.
ii. Algoritmo de ubicación totalmente asociativa.
iii. Algoritmo de ubicación asociativa por conjuntos con cuatro conjuntos.
iv. Algoritmo de ubicación asociativa por conjuntos con dos conjuntos.
v. Algoritmo de ubicación asociativa por sectores con cuatro sectores.

a) Para cada uno de los cinco supuestos:
a.1) Calcule el contenido del directorio caché tras esas dos direcciones (`00AA9` y `00BE5`).
a.2) Calcule el tiempo total consumido desde que el procesador solicita la primera dirección virtual (`00AA9`).
a.3) Indique cuántos comparadores contiene el directorio caché.

b) Si pretendemos minimizar el producto del retardo temporal por el número de comparadores, ¿cuál de los supuestos es el mejor?.

Se conoce por comparador el elemento necesario para encontrar la posición dentro de un sub-bloque cuando la ubicación en éste es al azar.

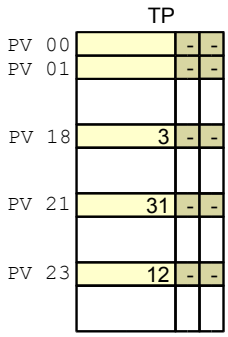
| MP | = 64 Kbytes = 2^{16} bytes ⇒ || DF || = 16 bits.
| EV | = 1 Mbyte = 2^{20} bytes ⇒ || DV || = 20 bits.
| PF | = | PV | = 128 bytes = 2^{7} bytes ⇒ || offset || = 7 bits.



Situación inicial:

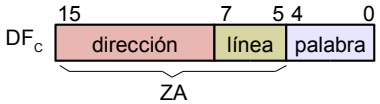
PV	DF
18	3
21	31
23	12

i. Asignación directa.

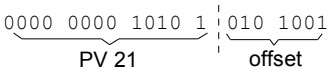


| caché | = 2^{3} líneas ⇒ 3 bits para direccionar una línea.
| línea | = 2^{5} bytes ⇒ 3 bits para indicar el desplazamiento dentro de una línea.

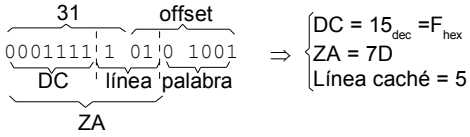
Ventaja: simplicidad y bajo coste.
Inconveniente: Reducido índice de aciertos.



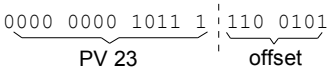
Primera DV: 00AA9.



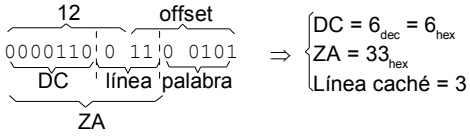
La PV 21 se corresponde con la PF 31, por lo que:



Segunda DV: 00BE5.



La PV 23 se encuentra residente en la MP ocupando la DF 12:



línea	DC	ZA
0		
1		
2		
3	6	33
4		
5	F	7D
6		
7		

DV = 00AA9 = 0000 0000 1010 1 | 010 1001

DF = 0000 1111 101 | 0 1001

Palabra dentro de la línea

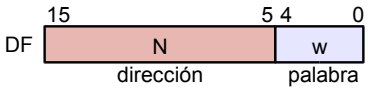
DF_C = 00 00111 1101 (0 | 7 | D)

$T_{AA9} = P + Q + R$

Se necesita un comparador por cada línea.

ii. Asignación totalmente asociativa.

Permite cualquier combinación de líneas de memoria física en la caché, elimienando cualquier conflicto entre ellas.
Ventaja: Mejor rendimiento.
Inconveniente: Mayor coste.



Primera DV: 00AA9.

Fallo de línea.

DF = 0000 1111 101 | 0 1001 (dirección | palabra)

DC = 7D

Segunda DV: 00BE5.

Fallo de línea.

DF = 0000 0110 011 | 0 0101 (DC | palabra)

DC = 33

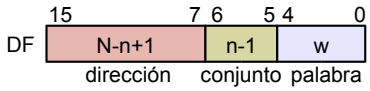
línea	DC	ZA
0	7D	7D
1	33	33
2		
3		
4		
5		
6		
7		

$T_{AA9} = P + 8 \cdot R$

Se necesitan 8 comparadores.

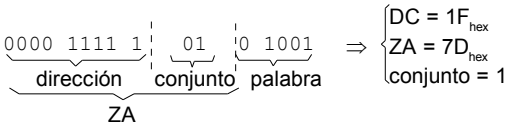
iii. Asignación asociativa por conjuntos (cuatro conjuntos).

Asignación directa para los conjuntos y totalmente asociativa para la línea caché dentro de cada conjunto.
Pretende reducir el coste para mantener un rendimiento similar a la ubicación totalmente asociativa.

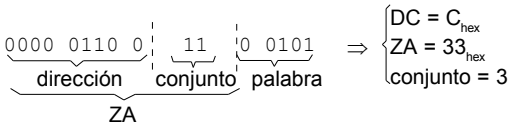


2^3 líneas $\Rightarrow$ 4 conjuntos de 2 líneas cada uno.
2^2 conjuntos $\Rightarrow$ 2 bits para codificar el número de conjunto.

Primera DV: 00AA9.



Segunda DV: 00BE5.

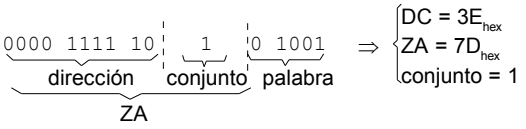


$T_{AA9} = P + 2 \cdot R$

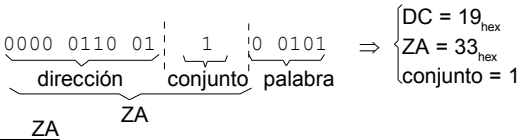
iv. Asignación asociativa por conjuntos (dos conjuntos).

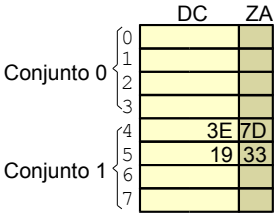
2^1 conjuntos $\Rightarrow$ 1 bit para codificar el número de conjunto.

Primera DV: 00AA9.



Segunda DV: 00BE5.

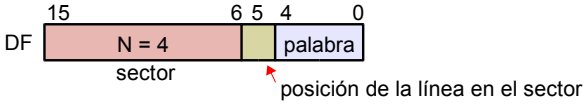




T = máx {P, Q} + R
Se necesitan 2 comparadores (los de las líneas).

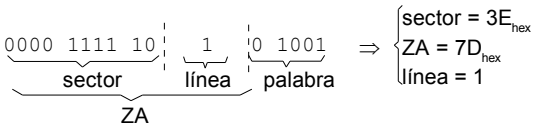
v. Asignación asociativa por sectores (cuatro sectores).

En cada sector hay 2 líneas.
Ventajas: Simple y de bajo coste, pues su directorio es más reducido.
Inconveniente: El número de combinaciones diferentes de líneas de memoria en la caché es muy inferior los esquemas anteriores.

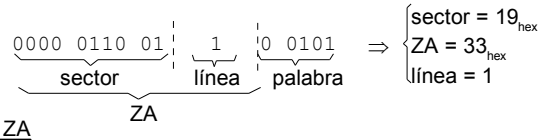


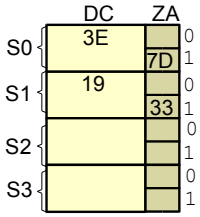
2^1 líneas por sector ⇒ 1 bit para codificar el número de línea.

Primera DV: 00AA9.



Segunda DV: 00BE5.





$T_{AA9} = P + 4 \cdot R$
Se necesitan 4 comparadores (los de los sectores).

Conclusión: Para minimizar el producto del retardo temporal por el número de comparadores, lo aconsejable es utilizar la **asignación directa**.

Sea un computador con una memoria principal de 32 bytes dividida en cuatro bloques iguales entrelazados según el esquema de orden inferior. Si numeramos los bloques de 0 a 3, entonces el bloque número M contiene el valor $16 \cdot M + 2 \cdot \alpha$ en la posición α (α =0, 1, 2, ..., 7). El computador también posee una memoria caché de 8 bytes. Todo el sistema de memoria está controlado mediante un esquema de memoria virtual paginada, donde las páginas virtuales (PV) contienen 8 bytes y las direcciones virtuales (DV) son de 8 bits de longitud. El esquema de traducción es totalmente asociativo y el sistema caché trabaja con líneas de 2 bytes siguiendo un esquema de ubicación asociativo por sectores (2 sectores) y un algoritmo de reemplazo aleatorio. La TLB contiene los pares (PV, PF) siguientes:

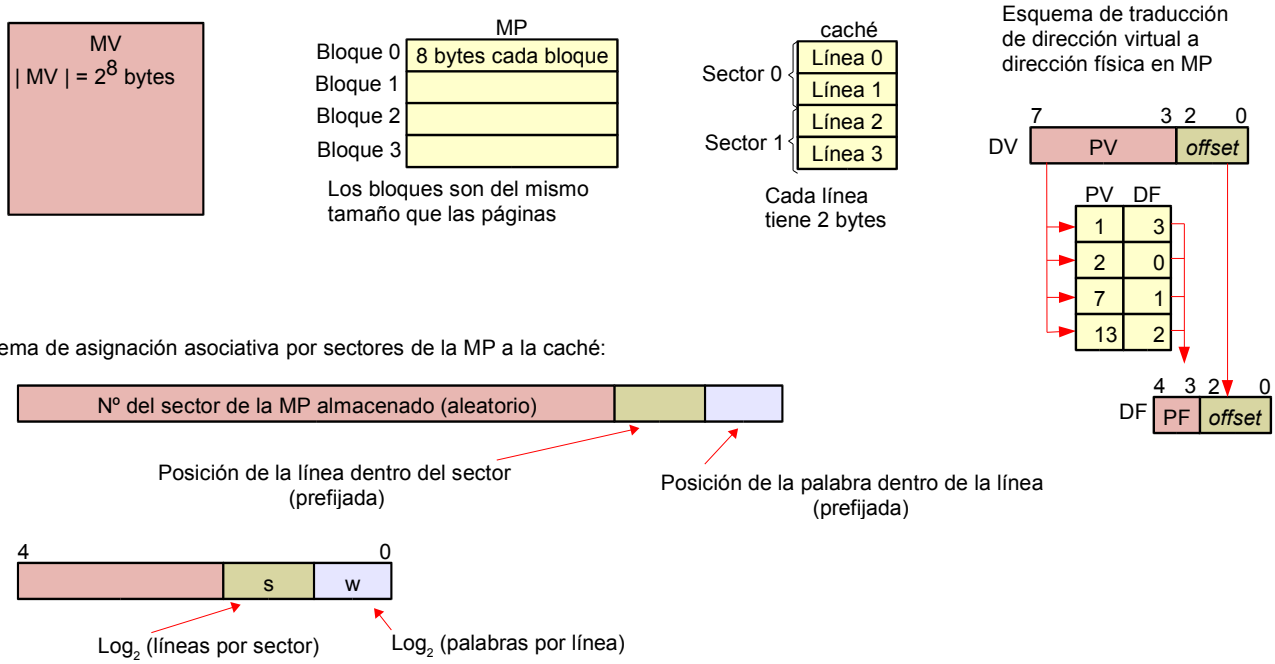
(1, 3), (2, 0), (7, 1) y (13, 2)

Y el procesador solicita la siguiente secuencia de DV (en base hexadecimal):

15, 16, 3C, 14, 0D, 0F y 68

Analice la evolución del contenido de la memoria caché (directorio caché y zona de almacenamiento).

| P | = 8 bytes ⇒ || offset || = Log_2 8 = 3 bits.
|| DV || = 8 bits
| MP | = 32 bytes = 2^5 bytes ⇒ || DF || = 5 bits.



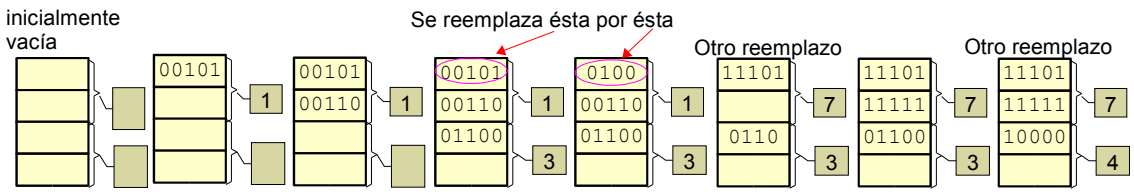
Esquema de asignación asociativa por sectores de la MP a la caché:

Secuencia de direcciones emitidas:

Dirección virtual en hexadecimal	***Dirección virtual en binario***		***Página virtual***	***Página física***	***Dirección física***
	Número de página virtual	offset			
15	0001 0	101	2	0	00101
16	0001 0	110	2	0	00110
3C	0011 1	100	7	1	01100
14	0001 0	100	2	0	00100
0D	0000 1	101	1	3	11101
0F	0000 1	111	1	3	11111
68	0110 1	000	13	2	10000

No ocurre ningún fallo de página, por lo cual no es necesario realizar reemplazos en la MP, ni cambia la TP.

Direcciones de las palabras de MP que hay en cada instante en la caché:

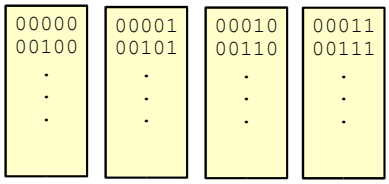


En realidad, en cada línea hay dos palabras (la acabada en 0 y la acabada en 1). Sólo se indica la referenciada.
Pero debe recordarse que están las dos.

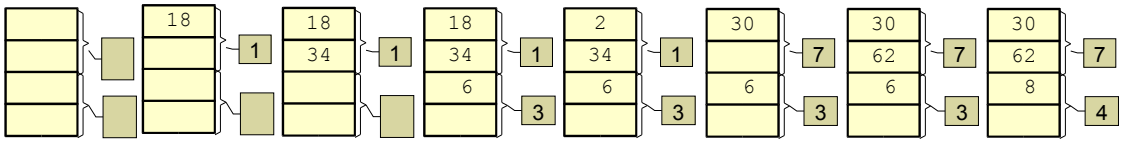
Contenidos correspondientes a esas direcciones en la caché.

En este ejercicio, para conocer el contenido de una dirección se usa la hipótesis: *Contenido=16·M+2·α*.
Donde:

- $x_4\, x_3 x_2 x_1\, x_{0\,(b}$: Dirección física en MP.
- M = $x_1\, x_{0\,(b}$: Número de bloque.
- α = $x_4\;\; x_3 x_2{}_{(b}$: Posición dentro de cada bloque.



$$\alpha \bmod 256 = \begin{cases} \alpha \;\; si \;\; \alpha < 256 \\ \alpha - 256 \;\; si \;\; 256 \le \alpha \le 512 \end{cases}$$



Sea un sistema de memoria virtual segmentada con un espacio virtual de 1 Kbyte. En el instante *t*, la lista de segmentos residentes es la siguiente:

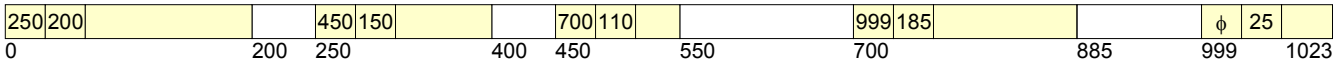
posición	0	100	250	450	700	999
tamaño	200	100	150	100	185	25

Se supone que el orden de entrada de los segmentos presentes en este instante *t* es el que tienen en la tabla. A continuación se reciben las siguientes peticiones de segmentos (todas ellas no residentes).

instante	*t*+1	*t*+2	*t*+3	*t*+4	*t*+5	*t*+6	*t*+7
tamaño	135	25	30	170	100	100	100

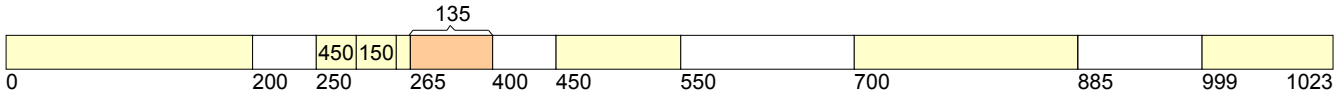
El algoritmo de reemplazo es FIFO hasta generar suficiente espacio (y sin compactación). Analice el estado de la memoria en el caso de que se aplique cada uno de los siguientes algoritmos:

i. *Best-fit*.
ii. *First-fit*.

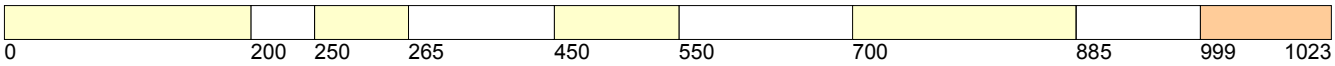


i. *Best-fit*.

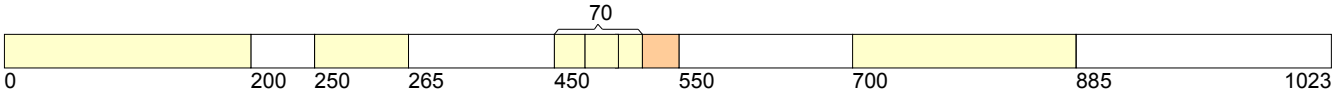
t+1



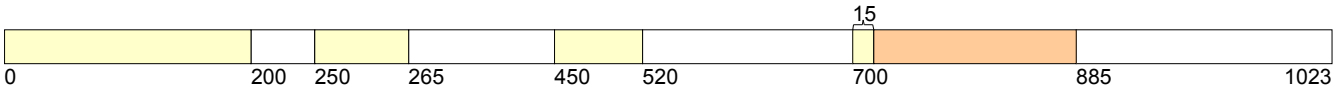
t+2



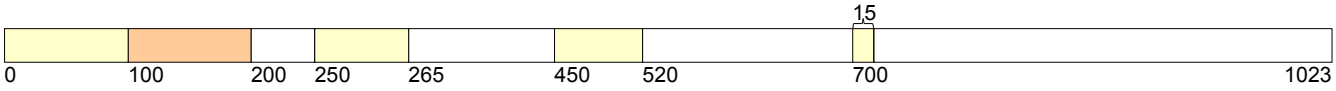
t+3



t+4

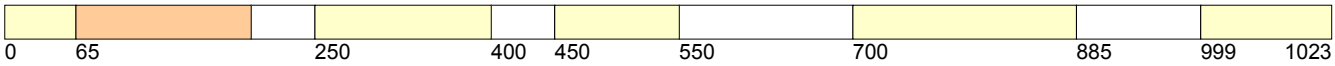


t+5

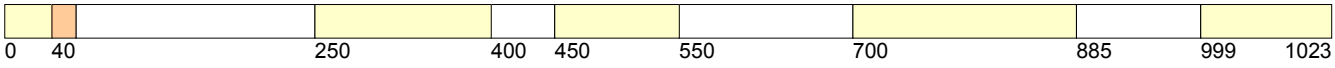


ii. *First-fit*.

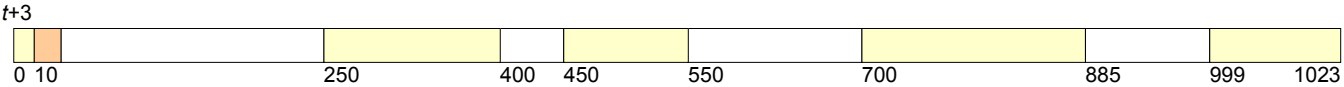
t+1



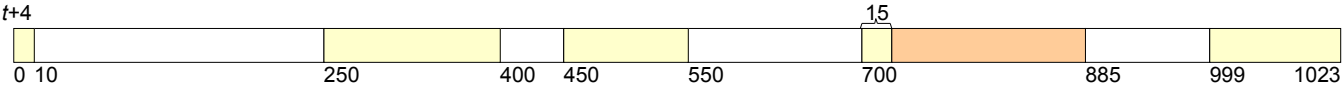
t+2



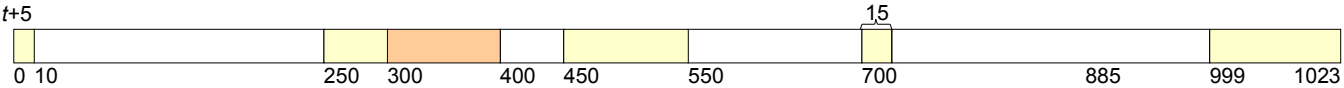
t+3



t+4



t+5



Sea un sistema de memoria virtual segmentada con un espacio virtual de 1 Mbytes y una memoria principal de 1 Kbytes. El sistema admite la definición de segmentos virtuales con un tamaño máximo de 2 Kbytes. Un programador redacta un programa en este sistema que consta de los diez primeros segmentos virtuales, de tamaños:

segmento	0	1	2	3	4	5	6	7	8	9
tamaño	160	50	170	70	114	150	155	135	60	25

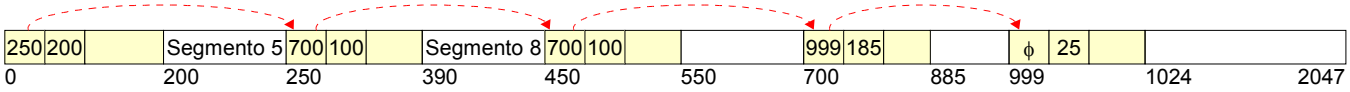
El computador ejecuta este programa; y tras un cierto intervalo temporal, la memoria principal contiene cuatro de estos segmentos, de manera que la lista LAVS contiene la siguiente información:

(0, 200), (250, 140), (450, 100), (700, 180) y (999, 25),

donde el primer valor de cada par indica la dirección del hueco y el segundo valor indica su tamaño. El sistema utiliza un algoritmo de ubicación del tipo *first-fit* y prefiere el reemplazo de un segmento antes de la compactación. Calcule el contenido final de LAVS después de que el procesador solicite la siguiente secuencia de direcciones virtuales (en base hexadecimal):

038A2, 04D01, 004BB, 030A9, 01773, 01A1C.

Situación inicial:

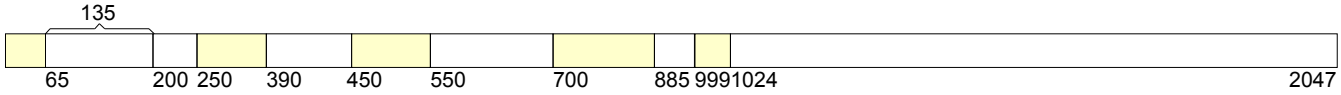


No se indica cuáles son los segmentos residentes. Sería necesario conocerlos para saber si se debe reemplazar o no tras cada petición. Supondremos que inicialmente, ninguno de los segmentos residentes pertenece al usuario de nuestro problema.

Dirección virtual en hexadecimal	*Dirección virtual en binario*		*Número de segmento*
038A2	0000 0011 1	000 1010 0010	7
04D01	0000 0100 1	101 0000 0001	9
004BB	0000 0000 0	100 1011 1011	0
030A9	0000 0011 0	000 1010 1001	6
01773	0000 0001 0	111 0111 0011	2
01A1C	0000 0001 1	010 0001 1100	3

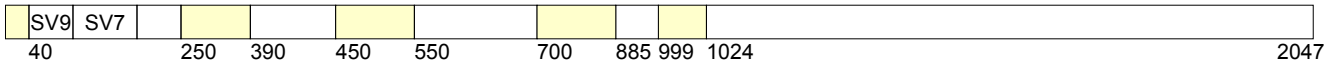
|| offset || = $\log_2$ |segmento|$_{máx}$ = 11 bits

Petición de segmento 1ª: SV7, de tamaño 135 bytes. Cabe en el primer hueco.



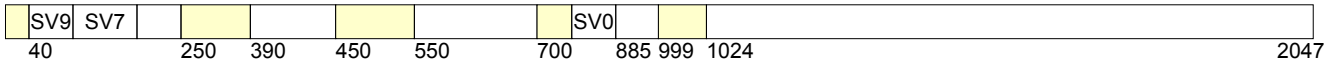
LAVS={(0, 65); (250, 140); (450, 100); (700, 185); (999, 25)}

Petición de segmento 2ª: SV9, de tamaño 25 bytes. También cabe en el primer hueco.



LAVS={(0, 40); (250, 140); (450, 100); (700, 185); (999, 25)}

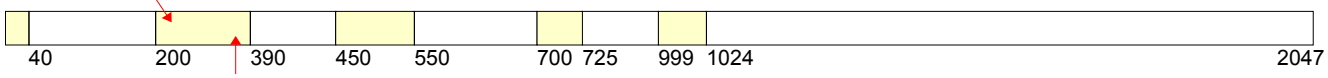
Petición de segmento 3ª: SV0, de tamaño 160 bytes. No cabe en ninguno de los tres primeros huecos, pero sí cabe en el cuarto hueco.



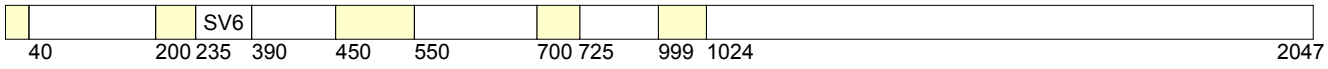
LAVS={(0, 40); (250, 140); (450, 100); (700, 25); (999, 25)}

Petición de segmento 4ª: SV6, de tamaño 155 bytes. No cabe en ningún hueco. Se utiliza el reemplazo LRU.

Elegimos el primero de los segmentos que había inicialmente, de tamaño 50

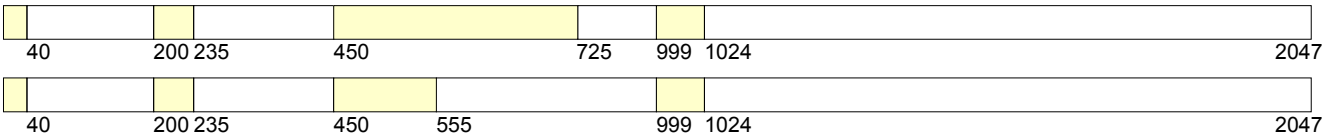


Ahora colocamos SV6 ahí



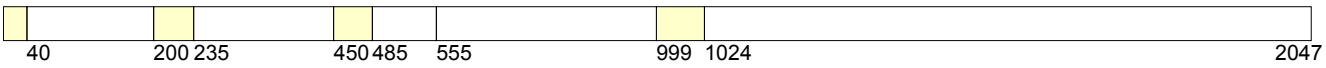
LAVS={(0, 40); (200, 35); (450, 100); (999, 25); (700, 25)}

Petición de segmento 5ª: SV2, de tamaño 170 bytes. No cabe en ningún hueco. No sirve con reemplazar el segmento original situado entre las posiciones 390 y 450 (que sería el siguiente). Debemos reemplazar el situado entre 550 y 700.



LAVS={(0, 40); (200, 35); (450, 105); (999, 25)}

Petición de segmento 6ª: SV3, de tamaño 70 bytes. No cabe en el primer hueco ni en el segundo, pero sí cabe en el tercer hueco.



LAVS={(0, 40); (200, 35); (450, 35); (999, 25)}

En cada ubicación y reeemplazo se iría modificando la tabla de segmentos.

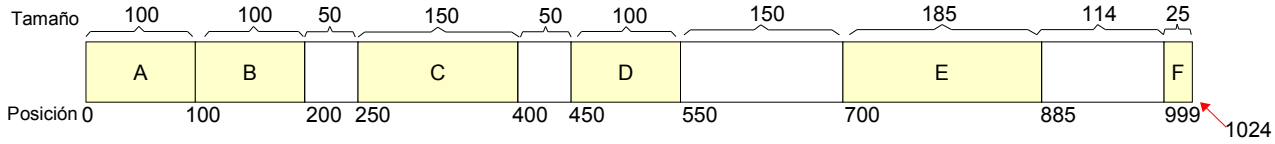
✎ Sea un sistema de memoria virtual segmentada con una memoria principal de 1 Kbyte. En un cierto instante, la lista de los segmentos residentes en la memoria física es la siguiente:

Posición	0	100	250	450	700	999
Tamaño	100	100	150	100	185	25

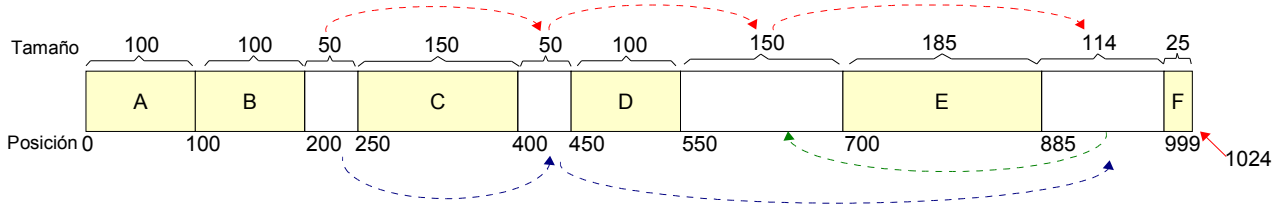
A continuación se recibe la petición de un segmento no residente de tamaño 120; y seguidamente, la petición de otro de tamaño 150. Analice el contenido de la memoria principal y qué acciones lleva a cabo la unidad de gestión de memoria (si hay varias posibilidades en cada acción, indíquelas todas) en los tres algoritmos siguientes:

i. *First fit.*
ii. *Best fit.*
iii. *Worst fit.*

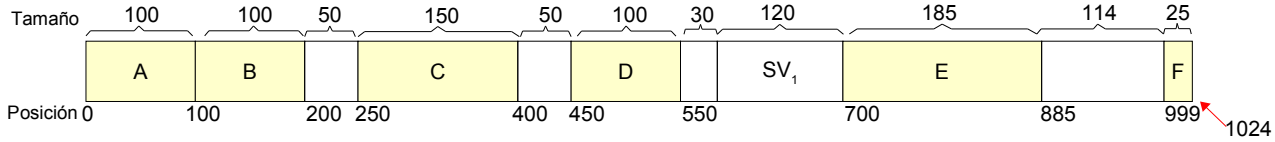
Estado inicial:



i. FIRST-FIT. El segmento virtual es introducido en el primer hueco que quepa. La búsqueda se lleva a cabo en orden ascendente de direcciones.



✔ SV_1
No se encuentra residente en la memoria principal ⇒ Fallo de segmento.
Tamaño de (SV_1) = 150-120=30 ⇒ Cabe en el tercer hueco

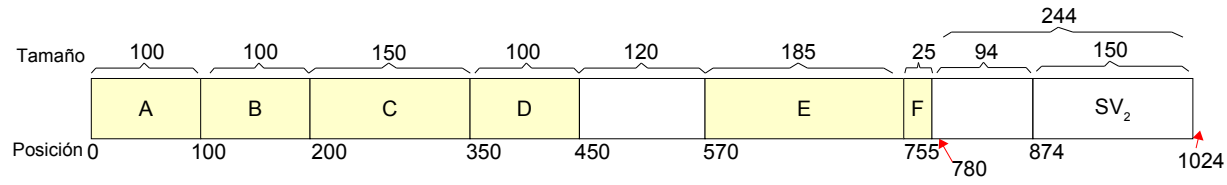


Lista LAVS: (200, 50), (400, 50), 550, 30), (885, 114).

✔ SV_2
No se encuentra residente en la memoria principal ⇒ Fallo de segmento.
Tamaño de (SV_2) = 150 ⇒ No cabe en ningún hueco ⇒ Es necesaria una compactación.

Trasladamos los segmentos libres desde sus respectivas posiciones hacia una zona de la memoria física de tal manera que cada hueco esté contiguo a otro, es decir, que los segmentos virtuales residentes en la memoria física pasen a encontrarse también contiguos en otra zona de la memoria.

La lista LAVS contendrá un único elemento cuyo tamaño será la suma de los tamaños de todos los huecos residentes. En este hueco podremos ubicar el segmento virtual que provocó el fallo.



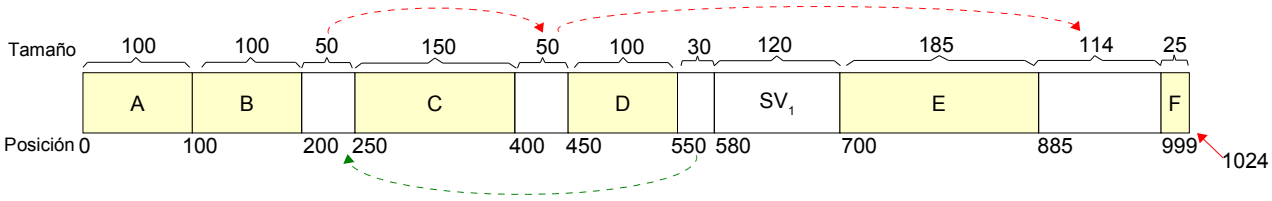
En el hueco resultante de tamaño 244 cabe el SV_2. Actualizando:
244-150 = 94
Lista LAVS: (780, 94).

Se podría haber elegido el reemplazo, es decir, elegir un segmento virtual residente de tamaño suficiente para dar cabida al segmento que provocó el fallo. Una vez que se ha seleccionado y reemplazado un segmento residente, debe actualizarse su entrada correspondiente de los TS. El consumo temporal asociado a la actualización es inferior que en el caso de la compactación.

Se podría haber reemplazado el bloque **C** (de tamaño 150) y en su lugar colocar el SV_2.
La lista LAVS resultaría: (350, 100), (550, 30), (885, 114). Adosamos el SV_2 al bloque **B**.

ii. BEST-FIT. Se recorre la lista LAVS en orden creciente de tamaños (con lo cual se tiende a generar segmentos de tamaño pequeño).

✔ SV_1
Tamaño de (SV_1) = 120.
Tamaño del hueco Nº 1 = 50 ⇒ Aquí no cabe.
Tamaño del hueco Nº 2 = ¿50? ⇒ Aquí no cabe.
Tamaño del hueco Nº 3 = 114 ⇒ Aquí no cabe.
Tamaño del hueco Nº 4 = 150 ⇒ Aquí sí cabe ⇒ 150-120=30.



Se reordena la lista LAVS: (550, 30), (200, 50), (400, 50), (885, 114).

✔ SV_2
Tamaño de (SV_2) = 150 ⇒ No cabe en ningún hueco ⇒ Es necesaria una compactación.
El resultado es idéntico al del FIRST-FIT, con la única diferencia en la lista LAVS: (930, 94).

iii. WORST-FIT. Se ordena la lista LAVS en orden inverso al del algoritmo BSET-FIT, es decir, en orden decreciente de tamaños. Los nuevos segmentos son ubicados en el mayor de los huecos libres.
El resultado es idéntico al del FIRST-FIT, con la única diferencia en la lista LAVS.

✔ SV_1
Tamaño de (SV_1) = 120 ⇒ LAVS = (885, 114), (400, 50), (200, 50), (550, 30).

✔ SV_2
Tamaño de (SV_2) = 150 ⇒ LAVS = (930, 94).

Sea un sistema segmentado con una memoria primcipal (*MP*) de 1 Kbyte (la máxima direccionable por el procesador) y un espacio virtual que admite un máximo de 16 segmentos, cada uno de ellos de un tamaño máximo de 512 bytes. Suponga que dicho sistema tiene cargado un programa que ocupa diez segmentos virtuales (*SV*), concretamente los diez primeros del espacio virtual cuyos tamaños respectivos son (en bytes):

50, 50, 150, 40, 135, 25, 400, 170, 100 y 110.

El procesador comienza a ejecutar el programa; y tras un cierto tiempo, nos encontramos con los cuatro primeros segmentos (del 0 al 3) en la *MP*, comenzando respectivamente, en las siguientes direcciones físicas (en notación decimal):

200, 400, 550 y 885.

Además, suponga que:

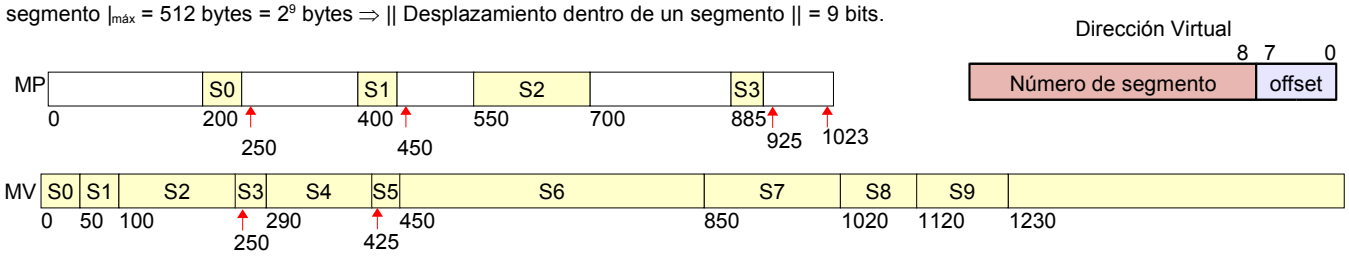
- ✔ Se utiliza el esquema de traducción híbrida, con una memoria asociativa capaz de almacenar sólo cuatro pares (*SV*, *SF*) (los cuatro más recientemente referenciados).
- ✔ La tabla de segmentos (*TS*) se encuentra en la *MP*, pero no debe tenerse en cuenta el espacio que ocupa.
- ✔ El gestor de memoria introducirá en la *MP* cada segmento en el primer hueco sificientemente grande que encuentre a partir de la dirección física 0.
- ✔ El gestor prefiere la compactación al reemplazo.
- ✔ En caso de reemplazo, se vacía el segmento más antiguamente referenciado cuyo tamaño sea igual o mayor que el tamaño del segmento solicitado.

a) Analice la evolución del contenido de la *MP*, de la memoria asociativa y de la *TS* a medida que el procesador emite las siguientes direcciones virtuales (en base hexadecimal):

82D, 6B6, AFD, C1C, 50B, E34, 1001, 127A.

b) Haga un recuento de los fallos de segmento.

| MP | = 1 Kbytes = 2^{10} bytes ⇒ || DF || = 10 bits.
| segmento |$_{máx}$ = 512 bytes = 2^9 bytes ⇒ || Desplazamiento dentro de un segmento || = 9 bits.



TS

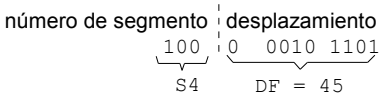
	segmento	longitud
0	200	50
1	400	50
2	550	150
3	885	40
4		
5		
6		
7		
8		
9		

TLB

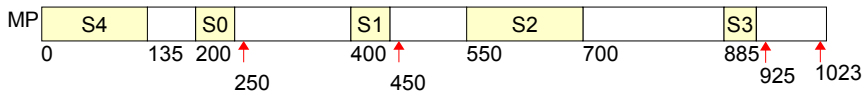
0	200
1	400
2	550
3	885

En el enunciado no se indica ningún orden.
Suponemos que el orden en que han tenido lugar las últimas referencias de cada segmento es el que hay en la tabla:
(más antiguo, ..., menos antiguo) = (0, 1, 2, 3)

Dirección virtual solicitada Nº 1: $082D_{16}$.



Ocurre un fallo de segmento en la TLB, pero no en la MP.
El segmento solicitado (S4) tiene un tamaño de 135 bytes. Cabe en el primer hueco.
La dirección física solicitada fue: $0\ 0010\ 1101_{16} = 45_{10}$



TLB

1	400
2	550
3	885
4	0

TS

	segmento	longitud
0	200	50
1	400	50
2	550	150
3	885	40
4	0	135
5		
6		
7		
8		
9		

Dirección virtual solicitada Nº 2: $06B6_{16}$. = 1718_{10}. Cae fuera del programa.

número de segmento | desplazamiento
011 | 0 1011 0110
S3

El segmento solicitado (S3) ya reside en la MP; y está anotado en la TLB.
No ocurre fallo de segmento. La MP no varía.
Únicamente es necesario actualizar el orden de entradas en la TLB.

TLB

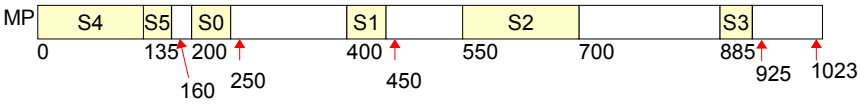
1	400
2	550
4	0
3	885

Dirección física referenciada: 885_d + 0 1011 0110_b = 1067_d. ¡Cae fuera de la zona de almacenamiento del programa en MP!.

Dirección virtual solicitada Nº 3: $0AFD_{16}$. = 2813_{10}.

número de segmento | desplazamiento
101 | 0 1111 1101
S5

El segmento solicitado (S5) provoca un fallo de segmento en la MP (y por tanto en la TLB).
Su tamaño es |S5|=25 bytes, es decir, cabe en el primer hueco.



TLB

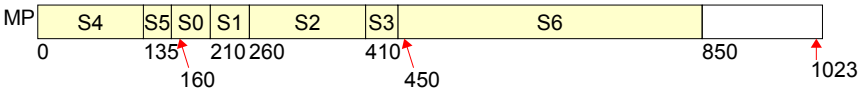
2	550
4	0
3	885
5	135

Dirección física referenciada: 135_d + 0 1111 1101_b = 388_d. También cae fuera del programa.

Dirección virtual solicitada Nº 4: $0C1C_{16}$. = 3100_{10}.

número de segmento | desplazamiento
110 | 0 0001 1100
S6

El segmento solicitado (S6) provoca un fallo de segmento en la MP.
Su tamaño es |S6|=400 bytes. No cabe en ninguno de los huecos existentes en este momento. Es necesaria una compactación.



TLB

4	0
3	410
5	135
6	450

TS

	segmento	longitud
0	160	50
1	210	50
2	260	150
3	410	40
4	0	135
5	135	25
6	450	400
7		
8		
9		

Dirección virtual solicitada Nº 5: $050B_{16}$. = 1291_{10}.

número de segmento | desplazamiento
010 | 1 0000 1011
S2

El segmento solicitado (S2) ya reside en la MP; pero no en la TLB. Actualizándola:

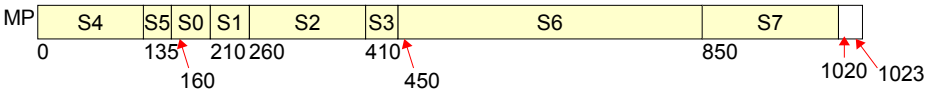
TLB

3	410
5	135
6	450
2	260

Dirección virtual solicitada Nº 6: $0E34_{16}$. = 3636_{10}.

número de segmento | desplazamiento
111 | 0 0011 0100
S7

El segmento solicitado (S7) provoca un fallo de segmento en la MP.
Su tamaño es |S7|=170 bytes.



TLB

5	135
6	450
2	260
7	850

TS

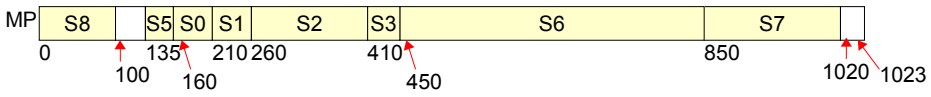
	segmento	longitud
0	160	50
1	210	50
2	260	150
3	410	40
4	0	135
5	135	25
6	450	400
7	850	170
8		
9		

Dirección virtual solicitada Nº 7: 1001_{16}. = 4097_{10}.

número de segmento	desplazamiento
0001 000	0 0000 0001
S8	

El segmento solicitado (S8) provoca un fallo de segmento en la MP.

Su tamaño es |S8|=100 bytes. No hay hueco suficiente. La MP está compactada, por lo que la única solición es un reemplazo.
Hay que buscar entre los segmentos de tamaño mayor o igual que 100 bytes, el más antiguamente referenciado.
Recordemos: *(más reciente, ..., menos reciente) = (7, 2, 6, 4)*.
Por tanto, hay que reemplazar el S4 por el S8.



TLB

6	450
2	260
7	850
8	0

TS

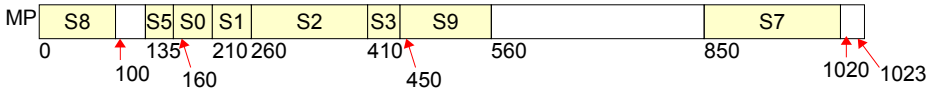
	segmento	longitud
0	160	50
1	210	50
2	260	150
3	410	40
4	-	-
5	135	25
6	450	400
7	850	170
8	0	100
9		

Dirección virtual solicitada Nº 8: $127A_{16}$. = 4730_{10}.

número de segmento	desplazamiento
0001 001	0 0111 1001
S9	

El segmento solicitado (S9) provoca un fallo de segmento en la MP.
Su tamaño es |S8|=110 bytes.

Sumando los tamaños de los huecos: (135-100) + (1024-1020) = 39 bytes < 110 bytes.
En caso de realizar una compactación, no se conseguiría hueco suficiente para ubicar el segmento S9. Por ello es necesario un reemplazo.
Segmentos candidatos (de tamaño mayor o igual que 110 bytes): S2, S6, S7 y S8.
Ordenándolos por el orden de más recientemente referenciados: *(más reciente, ..., menos reciente) = (8, 7, 2, 6)*.
Hay que reemplazar el S6 por el S9:



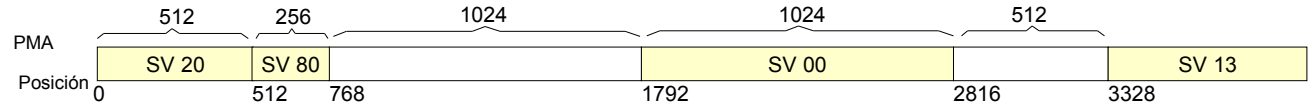
TLB

2	260
7	850
8	0
9	450

TS

	segmento	longitud
0	160	50
1	210	50
2	260	150
3	410	40
4	-	-
5	135	25
6	-	-
7	850	170
8	0	100
9	450	110

Sea un sistema de memoria virtual segmentada; y un espacio virtual de 1 Mbyte. El sistema dispone de una memoria física de 4 Kbytes reservada completamente para un único proceso (las tablas de traducción no ocupan espacio de memoria). Considere que el tamaño del mayor de los segmentos no supera 1 Kbyte. Además, el sistema dispone de una memoria caché capaz de almacenar 8 líneas de 4 bytes cada una, con un esquema de asignación directa. El estado inicial de la memoria física y de la caché es el siguiente:



línea	DC	ZA
0	12	090
1	01	009
2	7E	3F2
3	3A	1D3
4	3E	1F4
5	03	01D
6	76	3B6
7	38	1C7

El sistema utiliza un algoritmo de ubicación *Best-fit* (los segmentos libres son ordenados según orden creciente de los tamaños de sus huecos) y reemplazo LRU (se reemplaza el más antiguamente referenciado), prefiriendo la compactación al reeemplazo. La cola LRU es:

{0, 80, 20, 13}

Siendo el 0 el segmento más recientemente referenciado; y si se suprime un segmento de la PMA, se debe compactar antes de introducir el nuevo segmento. Además, cada compactación lleva asociada una actualización de la caché para contener las nuevas direcciones de las líneas. Dada la siguiente secuencia de direcciones virtuales:

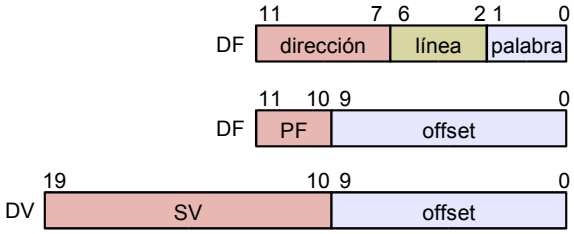
{08C11, 05480, 00407, 08A08}

correspondientes a segmentos de tamaños respectivos

256, 768, 512 y 1024 bytes

Analice la evolución de la memoria y de la caché, indicando las operaciones efectuadas (compactación, acceso a la memoria secundaria, etc) y las direcciones físicas que se generan.

| EV | = 1 Mbyte = 2^{20} bytes ⇒ || DV || = 20 bits.
| EF | = 4 Kbytes = 2^{12} bytes ⇒ || DF || = 12 bits.
| SV $|_{máx}$ = 1 Kbyte = 2^{10} bytes ⇒ Desplazamiento: || offset || = 10 bits.
| caché | = 2^3 líneas ⇒ || ubicación dentro de la caché || = 3 bits
| línea | = 4 bytes ⇒ || desplazamiento || = 2 bits



En primer lugar se han de traducir las direcciones virtuales a direcciones físicas; y posteriormente las direcciones físicas deberán ser traducidas en el directorio de la caché.

	DV	Línea de MP				SV	$|SV|_{máx}$
		SV		offset			
		Línea caché		ubicación	palabra		
1ª	08011	0000 1000 11	00 000	1 00	01	35	0256
2ª	05480	0000 0101 01	00 100	0 00	00	21	0768
3ª	00407	0000 0000 01	00 000	0 01	11	01	0512
4ª	08A08	0000 1000 10	10 000	0 10	00	34	1024

Línea caché dirección DC	Ubicación de la línea	Línea de MP ZA
C	0	60
4	4	24
0	0	00
A	0	50

Dirección virtual solicitada Nº 1: SV 35 de tamaño 256.

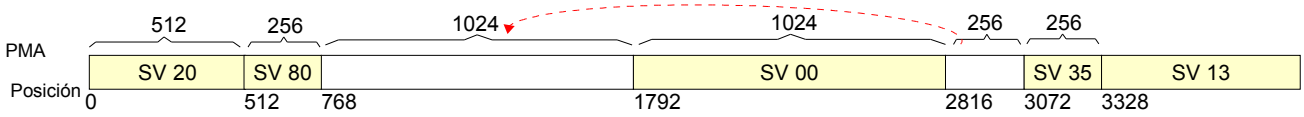
Según el algoritmo *Best-fit*, ha de ser colocada por encima de SV 13. Por tanto, la dirección física será: DF= 3328-256=3072:

```
DF = 1100 0000 0000+ offset
```

```
      1100 000 0 00 00
    +   00 000 1 00 01
    ------------------
DF ≡ 1100 000 1 00 01
     DC=60hex          línea=4
     ZA=304
```

Ocurre un fallo de segmento.

Ocurre un fallo de línea, pues donde debería ir la línea `4` ya está ocupada por otra línea.
Reemplazamos la dirección `3E` por la `60`; y en la zona de almacenamiento se colocará su contenido `ZA=304`.



línea	DC	ZA
0	12	090
1	01	009
2	7E	3F2
3	3A	1D3
4	60	304
5	03	01D
6	76	3B6
7	38	1C7

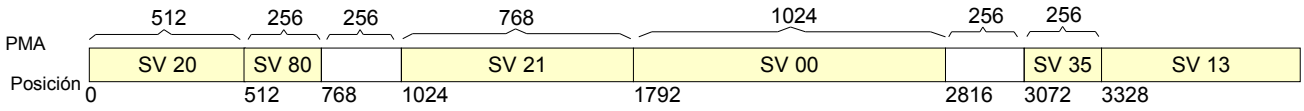
Dirección virtual solicitada Nº 2: SV 21 de tamaño 768.

Según el algoritmo *Best-fit*, SV 21 ha de ser colocada por encima de SV 00. Por tanto, la dirección física será: DF= 1792-768=1024:

Ocurre un fallo de segmento.

```
1024 = 1100 000 0 00 00
     +   00 100 0 00 00
     ------------------
DF ≡ 0100 100 0 00 00
     DC=24hex          línea=0
     ZA=120
```

Ocurre un fallo de línea, pues donde debería ir la línea `0` ya está ocupada por otra línea.
Reemplazamos la dirección `12` por la `24`; y en la zona de almacenamiento se colocará su contenido `ZA=120`.

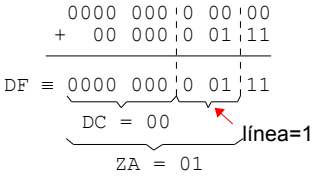


línea	DC	ZA
0	24	120
1	01	009
2	7E	3F2
3	3A	1D3
4	60	304
5	03	01D
6	76	3B6
7	38	1C7

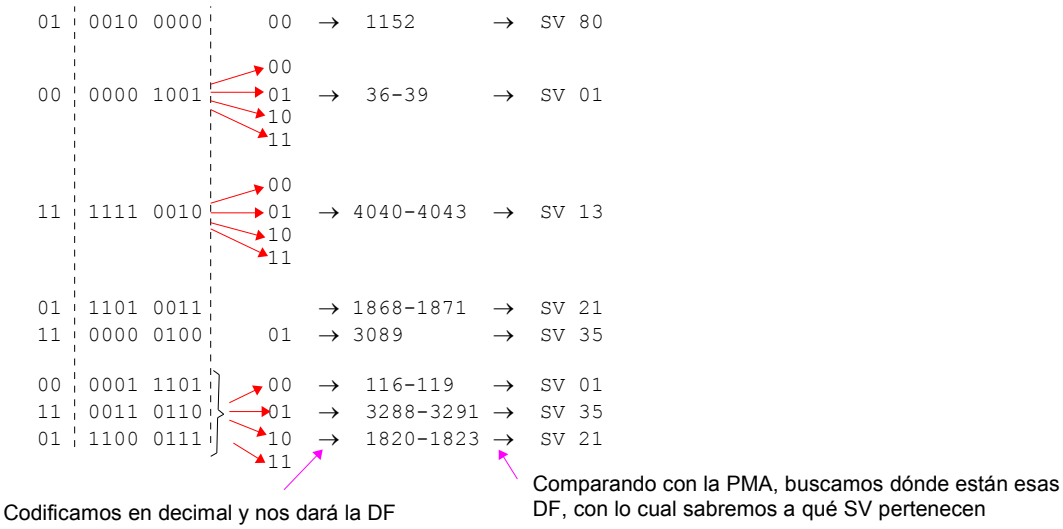
Dirección virtual solicitada Nº 3: SV 01 de tamaño 512.

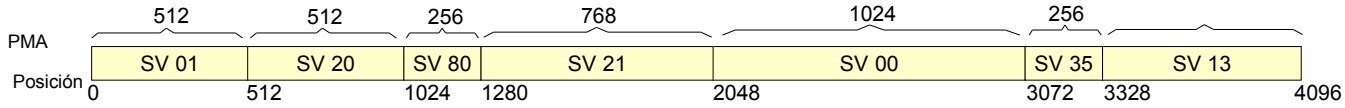
No cabe en ninguno de los dos huecos, por lo que es necesario compactar. Para ello desplazamos segmento a segmento hacia abajo.
SV 13 permanece donde estaba.
SV 35 permanece donde estaba.
SV 00 baja y se situará en la posición 3072-1024=2048.
SV 21 (de tamaño 768) quedará en la posición 2048-768=1280.
SV 80 (de tamaño 256) quedará en la posición 1280-256=1024.
SV 20 (de tamaño 512) quedará en la posición 1024-512=512.

El hueco resultante es de tamaño 512, por lo tanto ya cabe el SV 01. Ahora, para calcular la DF, como SV 01 está en la posición 0:



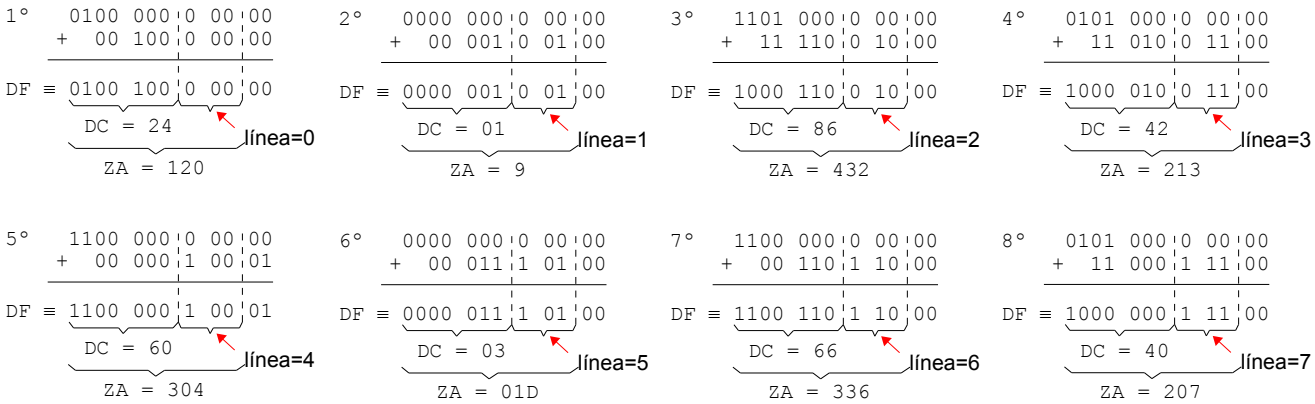
Al compactar hay que actualizar la caché. Se codifican en binario todas las ZA de la caché y se añaden los dos bits de palabra:





línea	DC	ZA
0	24	120
1	01	009
2	86	432
3	42	213
4	60	304
5	03	01D
6	66	3B6
7	40	207

Las DF están donde empiezan esos segmentos virtuales; sumamos a las DF los *offsets* de antes, con lo que obtenemos DC y ZA. Si coinciden se deja el mismo valor; y si no, los nuevos valores.

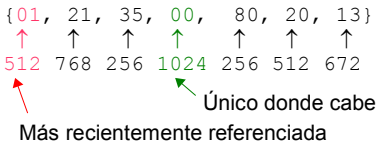


Ocurre un fallo de línea, pues en la línea 1 está situado un segmento. Actualizamos DC=00 y ZA=001:

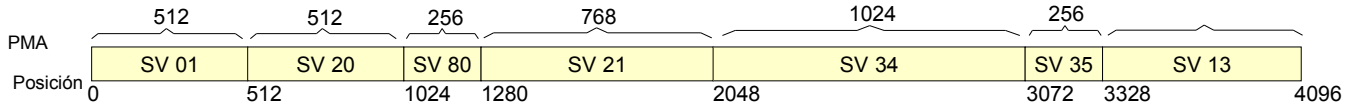
línea	DC	ZA
0	24	120
1	00	001
2	86	432
3	42	213
4	60	304
5	03	01D
6	66	3B6
7	40	207

Dirección virtual solicitada Nº 4: SV 34 de tamaño 1024.

Ocurre un fallo de segmento. Como ahora no es posible una compactación, se ha de utilizar el algoritmo de reemplazo LRU. Se busca en la PMA un segmento de tamaño mayor o igual que 1024, teniendo en cuenta la cola LRU. La cola será hasta este momento de la forma:

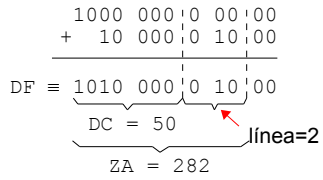


Reemplazamos el SV 00.



línea	DC	ZA
0	24	120
1	00	001
2	50	282
3	42	213
4	60	304
5	03	01D
6	66	3B6
7	40	207

Para calcular la dirección física:



Ocurre un fallo de línea, pues en la línea 2 está situada otra dirección: DC=86 y ZA=432. Por tanto, hay que sustituirla.

Sea un computador con la memoria jerarquizada en dos niveles y gestionada mediante un esquema de memoria virtual segmentada con paginación. El espacio virtual de es 1 Mbytes, totalmente privado para cada proceso. La memoria principal es de 64 Kbytes. El tamaño de las páginas es de 512 bytes y los segmentos están limitados a un tamaño máximo de 16 Kbytes. La memoria principal está organizada en bytes y el byte cuya dirección física es α contiene el valor (α `mod 256`). El esquema de traducción utilizado es directo en un único nivel. Cada entrada de la tabla de segmentos consta de 32 bits, de forma que el campo que especifica una dirección se encuentra en la parte menos significativa de dicha entrada (todos los bits de protección, etc., se encuentran en la parte más significativa). Cada entrada de las tablas de páginas consta de 16 bits, de manera que el campo que especifica la página física también se encuentra en la posición menos significativa de la entrada. Todas las tablas de segmentos y páginas están residentes en la memoria principal. Considere que el registro base de la tabla de segmentos contiene el valor `0`. Si el procesador solicita la dirección virtual `11D1C`, ¿qué dirección física se obtiene tras el proceso de traducción?.

| EV | = 1 Mbyte = 2^{20} bytes $\Rightarrow$ || DV || = 20 bits.
| EF | = 4 Kbytes = 2^{16} bytes $\Rightarrow$ || DF || = 16 bits.
| P | = 512 bytes $\Rightarrow$ || Posición dentro de la página || = 9 bits
| SV $|_{máx}$ = 16 Kbyte = 2^{14} bytes $\Rightarrow$ || Posición dentro del segmento (de cada palabra) || = 14 bits.

Número máximo de páginas por segmento:

$$\frac{Tamaño\ máximo\ de\ segmento}{Tamaño\ de\ página} = \frac{2^{14}}{2^{9}} = 2^{5} \Rightarrow$$ || Posición de la página dentro del segmento || = 5 bits