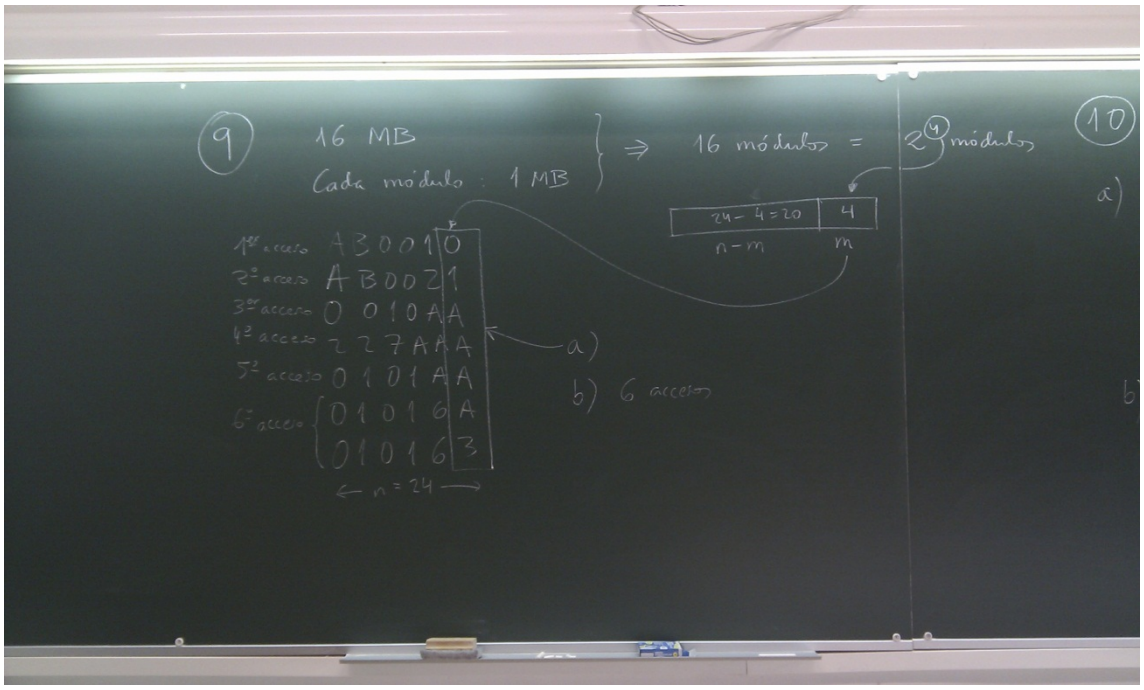
Problema Memoria 9

Aunque no lo hayamos explicado en clase, entrelazado **inferior** consiste en que los bits inferiores seleccionan el módulo de memoria (hemos visto que el 8086 usa entrelazado con el bit inferior A0, 80386DX usa entrelazado de 2bits A0-A1), y acceso Tipo S (**simultáneo**) consiste en que se hace *latch* a la salida de datos, no a la entrada de direcciones (se muestra la misma dirección a todos los módulos en cada acceso). Como sólo los dos últimos accesos coinciden en la dirección mostrada al módulo…

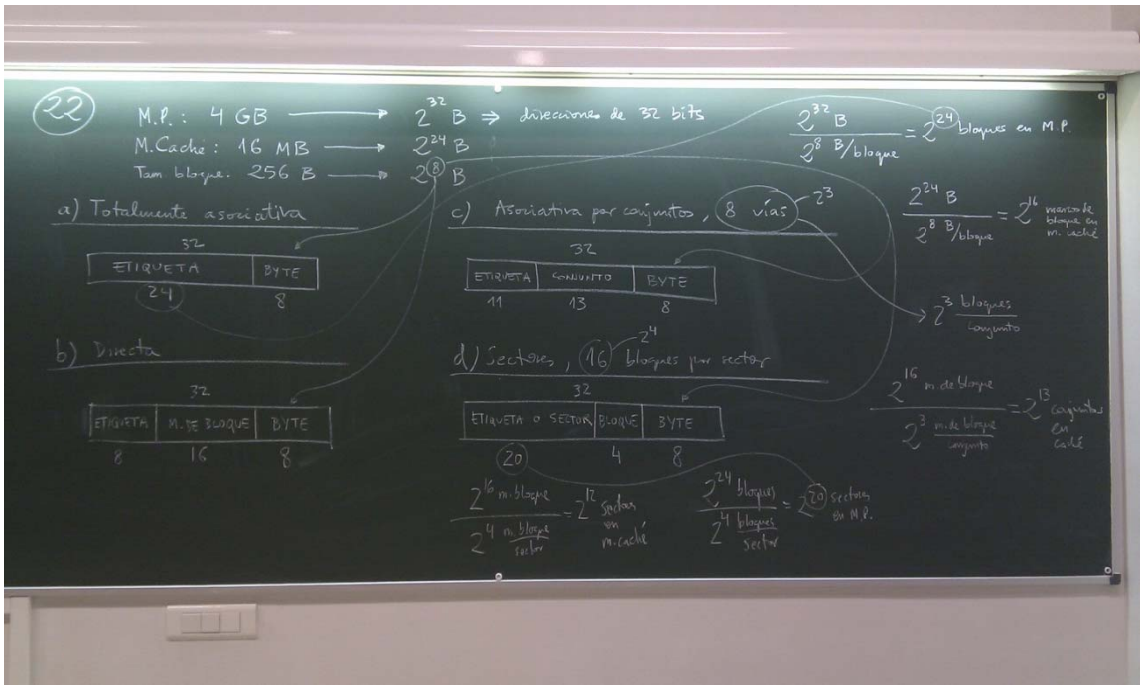


Problema Memoria 10

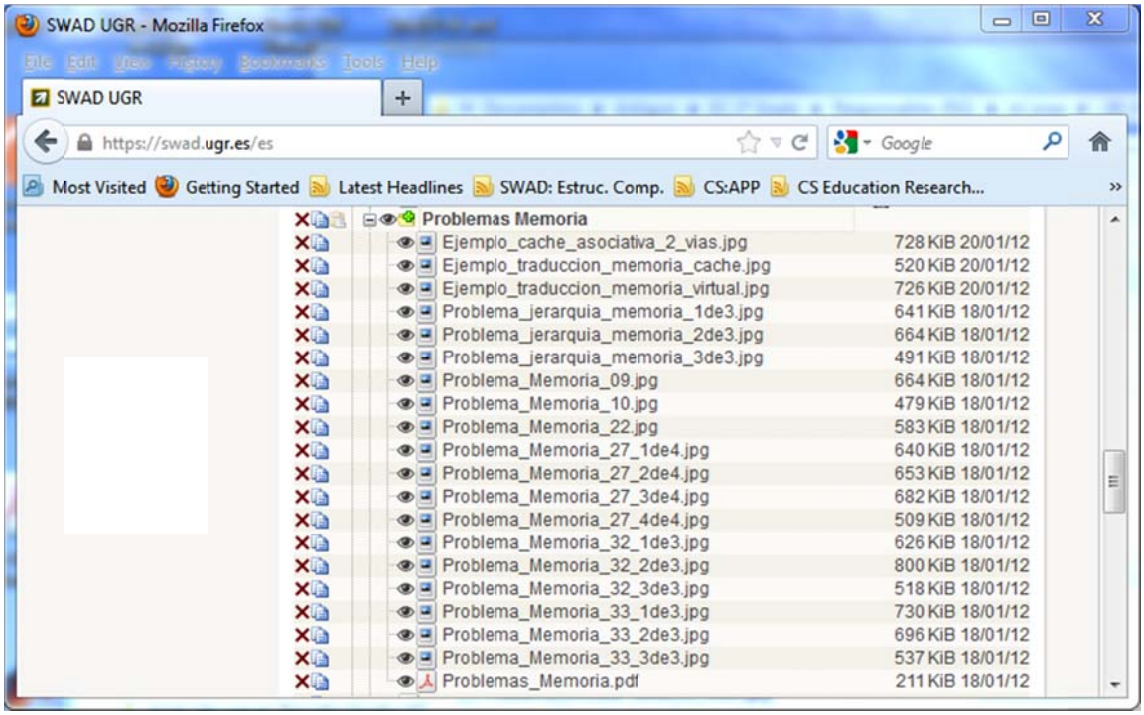
Aunque no lo hayamos explicado en clase, acceso Tipo C (**concurrente**) consiste en que se hace *latch* a la entrada de direcciones, no a la salida de datos (en cada acceso duración τ se muestra una dirección a un módulo, recogiendo el resultado T_a después). Sabiendo eso, como hay 4 módulos, sale 2x y 4x

Problema Memoria 22

Para a) y b) basta calcular cuántos bloques (y marcos) hay en MP (y cache). Para c) y d) hay que calcular cuántos conjuntos y sectores hay en MP y cache.

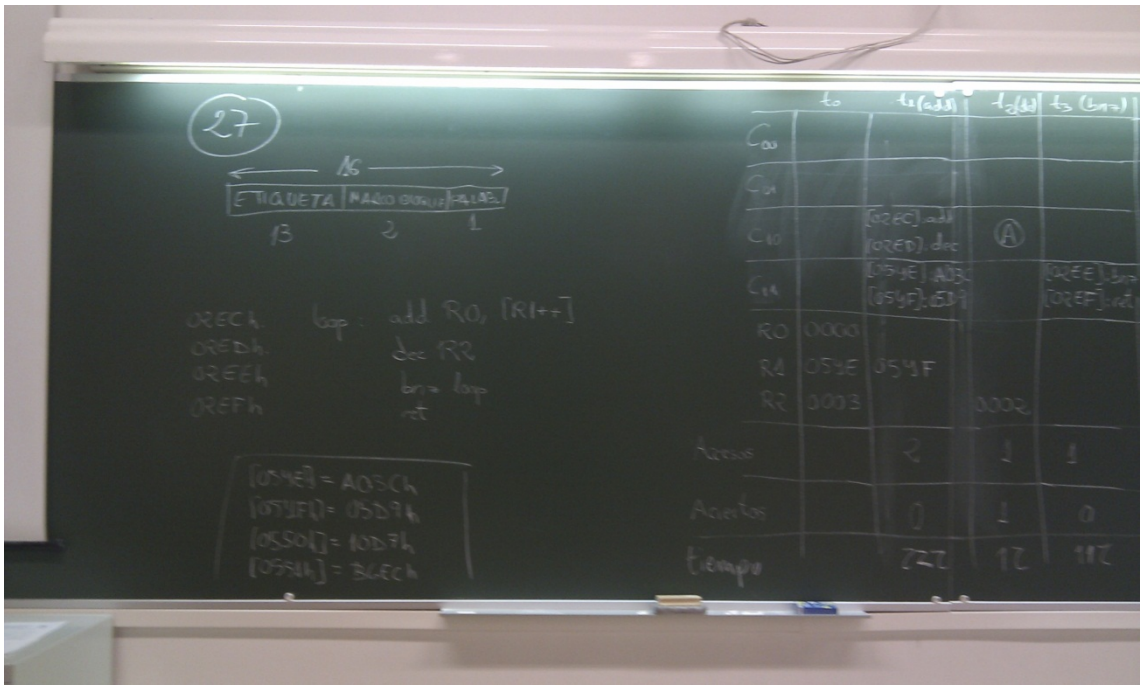


Lista de imágenes

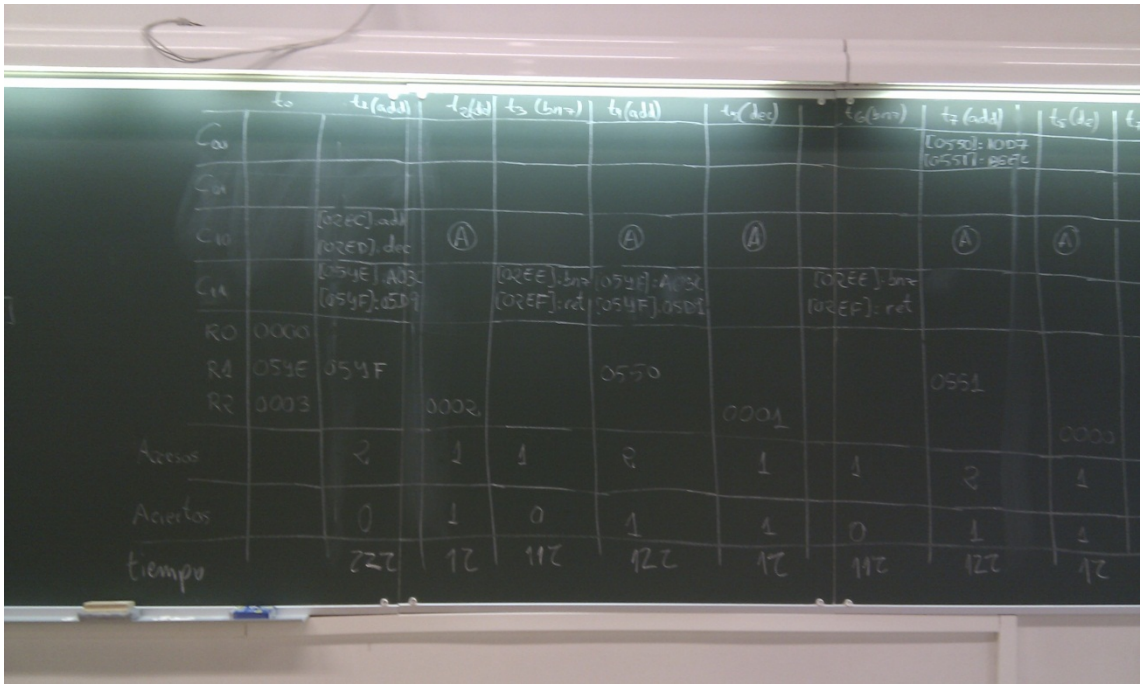


Problema Memoria 27

En la primera figura (a la izquierda) se resume y elabora el enunciado. Notar que en cada marco caben 2 palabras, así que las posiciones 0-7 irían a los marcos 0-3, y las posiciones 8-F también a marcos 0-3. En general, las posiciones que acaben en 0,8 van al marco 0, en 2,A al marco 1, en 4,C al 2, en 6,E al 3.

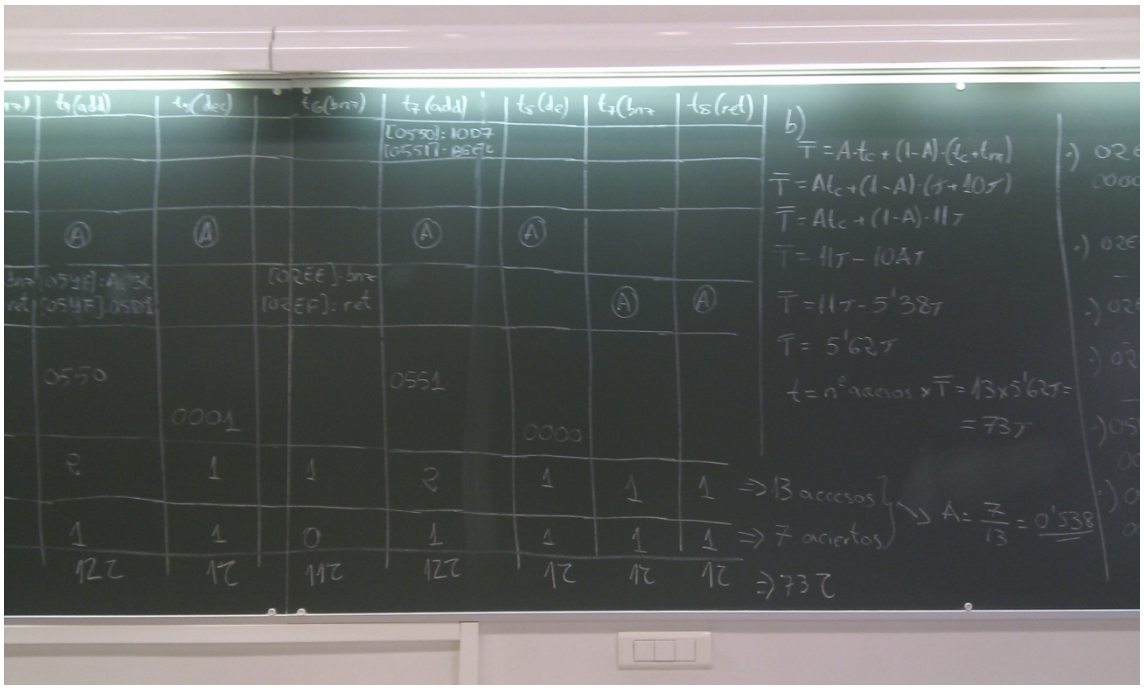


En la segunda y tercera figuras se hace una traza del programa en cache (a). Cada fila es una de las 4 líneas de cache, cada columna es un instante de tiempo (siguiente instrucción), y se van anotando los cambios en cache. También se hace traza de los valores de los registros (parece que a R0 no se le presta mucha atención), y se lleva contabilidad de los accesos, aciertos de cache y tiempo consumido (ver b).

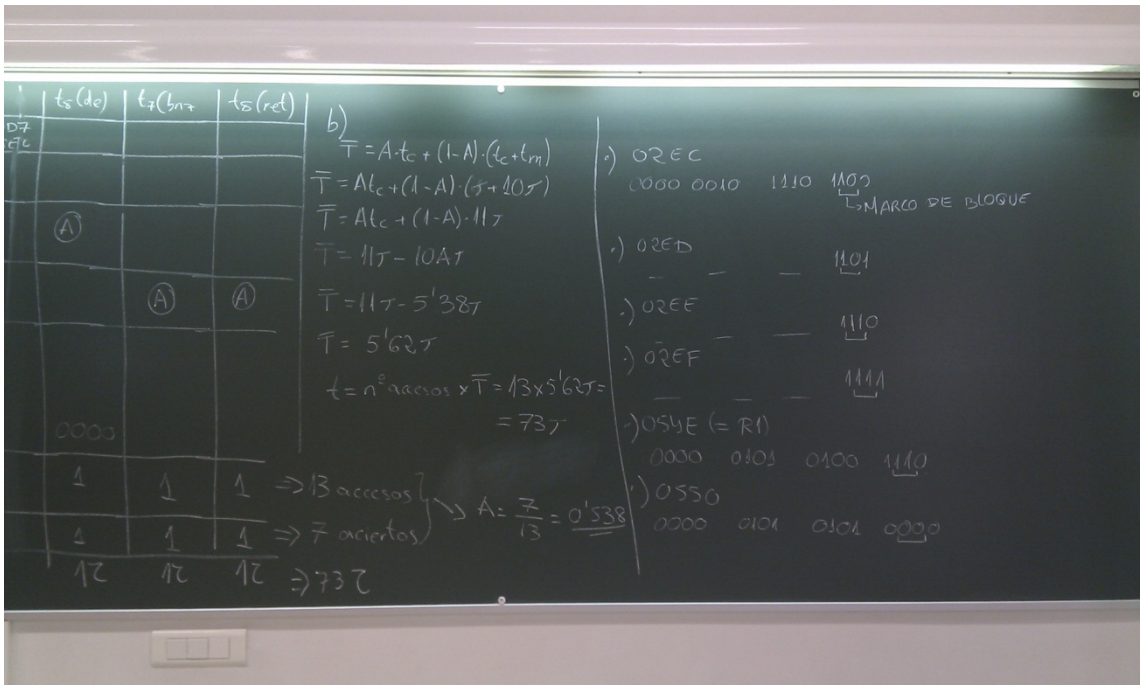


Sólo le faltan 2 columnas a la figura 2, mostradas después en la figura 3. Se capta la primera instrucción (dir. 02ECh se trae el marco C10), se accede a [R1++] (054Eh al marco C11), 2 fallos t1, dec R2 (acierto al captar la instrucción, t2), captar bnz (02EEh machaca C11, t3), acierto add, fallo [R1++] t4, etc…

Para anotar la fila “tiempo” se usa información del apartado b).

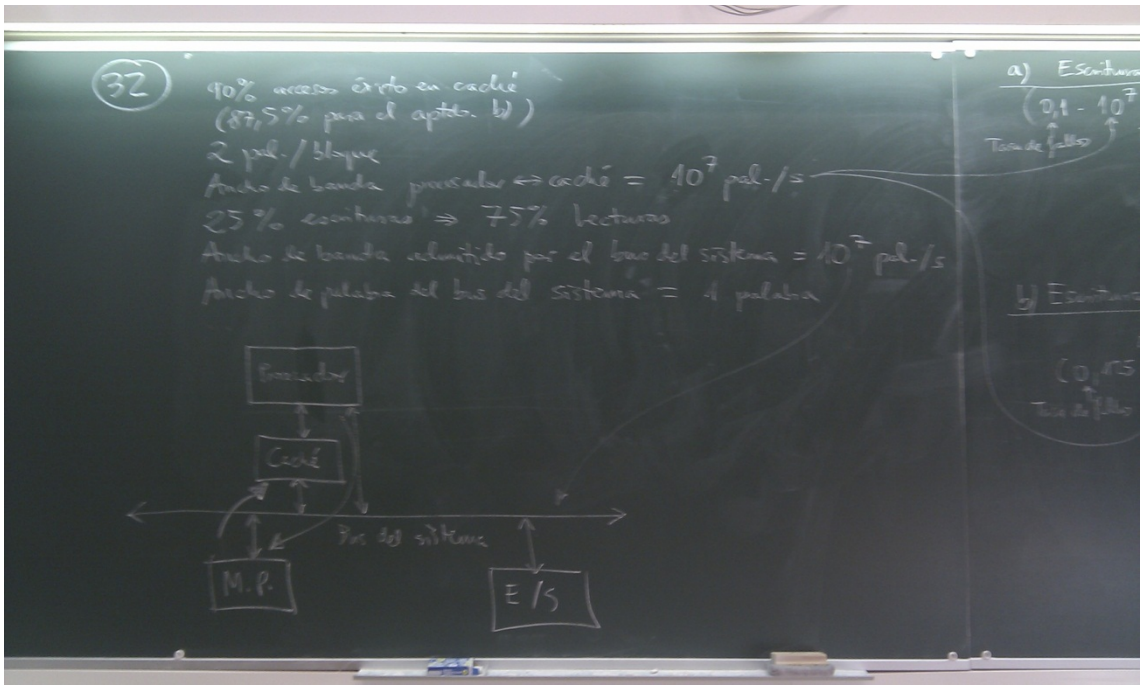


En la figura 4 se completa b), usando como tasa de aciertos la que tiene el programa de ejemplo (7/13). Sale $T_{medio}=5.62\tau$ (y por supuesto que $13*T_{medio} =73\tau$, como que la fórmula usada es justo lo que hemos ido anotando en "tiempo")

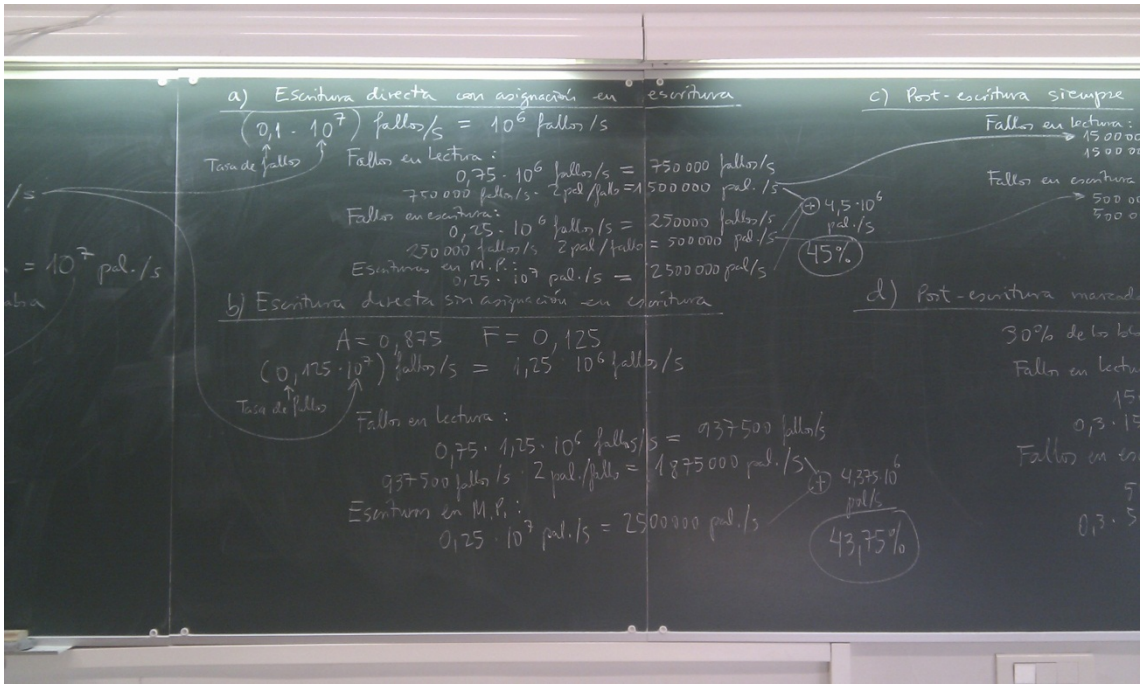


Problema Memoria 32

En la primera figura (a la izquierda) se resume y elabora el enunciado. (no queda claro por qué el ancho de banda es igual, 10^7pal/s, tanto para cache como para bus sistema MP… ¿es más cara la cache?)

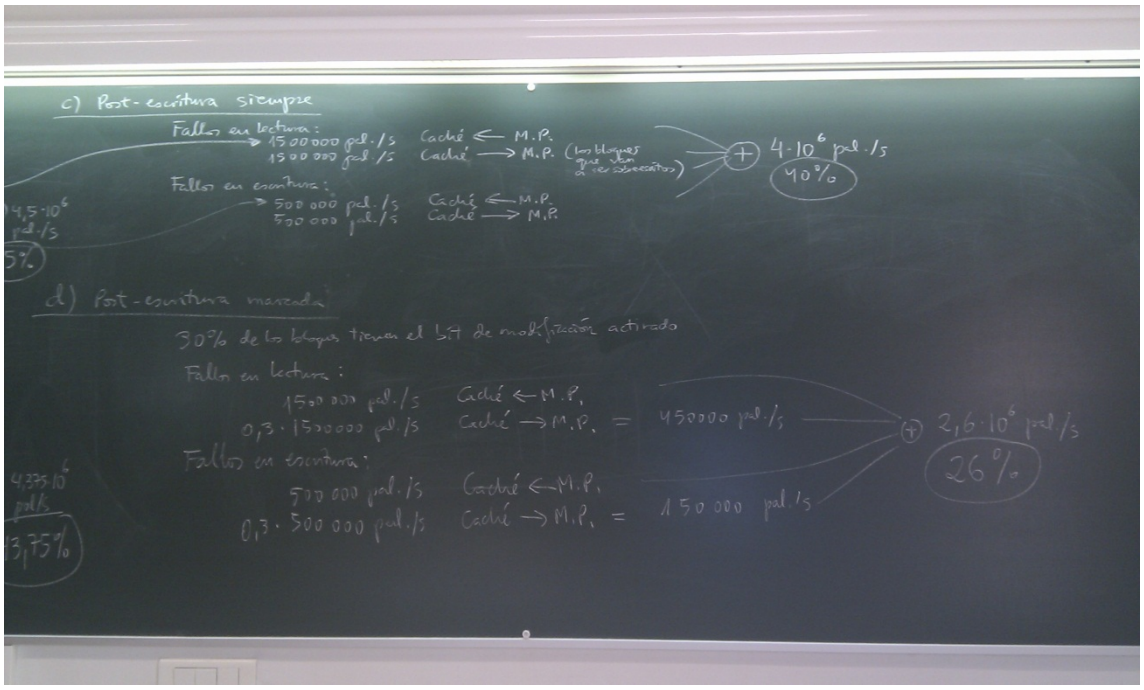


En la figura 2 se responden a) y b). Para a), recordar que “Con asignación” significa que se trae el bloque en fallo en escritura. Como cada bloque son 2 pal, cada fallo lectura son 2pal, cada fallo escritura también (por “con asignación”), y además escritura a memoria (“write-through”). Sale 45%



Para b), “Sin asignación” es que no se trae el bloque en fallo en escritura, sólo se hace write-through.

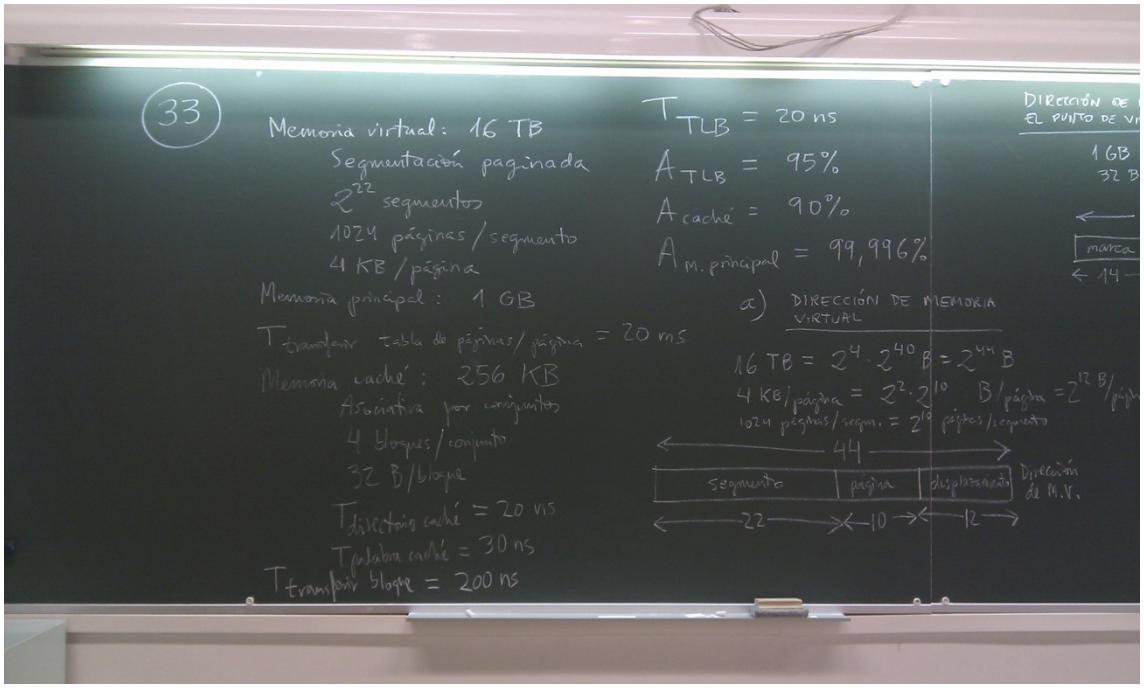
En la figura 3 se responden c) y d). Para c), se asume que la cache está llena, y cualquier fallo provoca que haya que post-escribir el bloque (además de traerse el que falla). En fallo en escritura hay que traerse el bloque para poder escribir (y al estar llena la cache, hay que post-escribir el marco que se desaloja).



Para d), también con toda la cache llena, si sólo el 30% de bloques está modificado, sólo hay que post-escribir ese 30%.

Problema Memoria 33

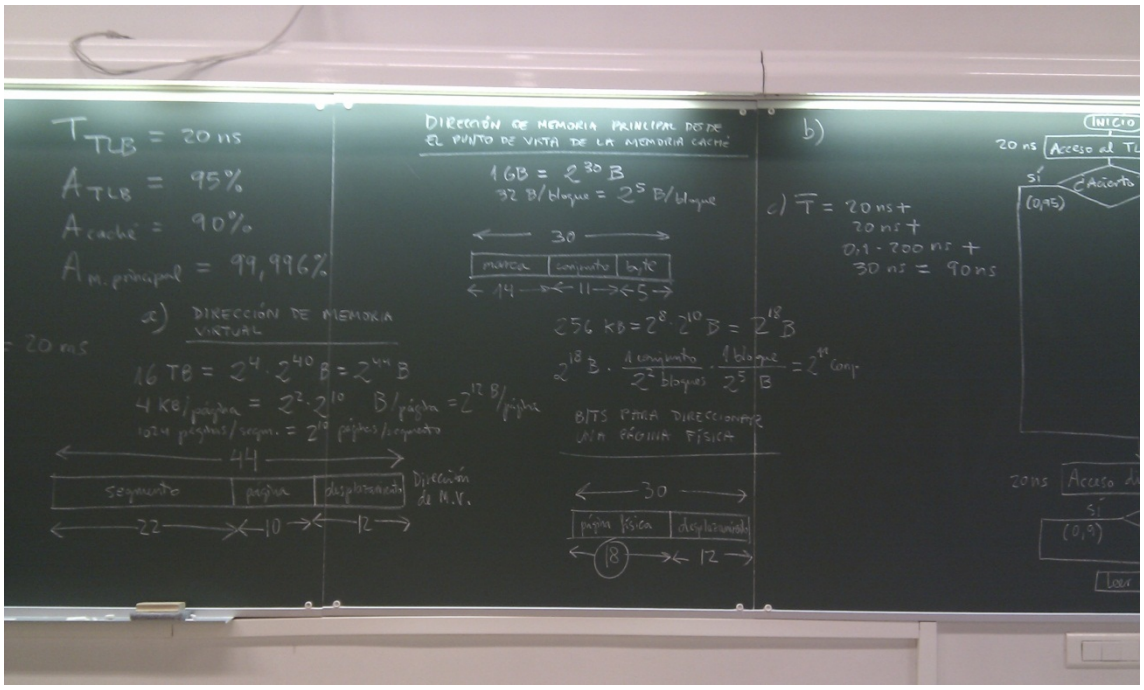
En la primera figura (a la izquierda) se resume y elabora el enunciado.



En la figura 2 se responde a) y c).

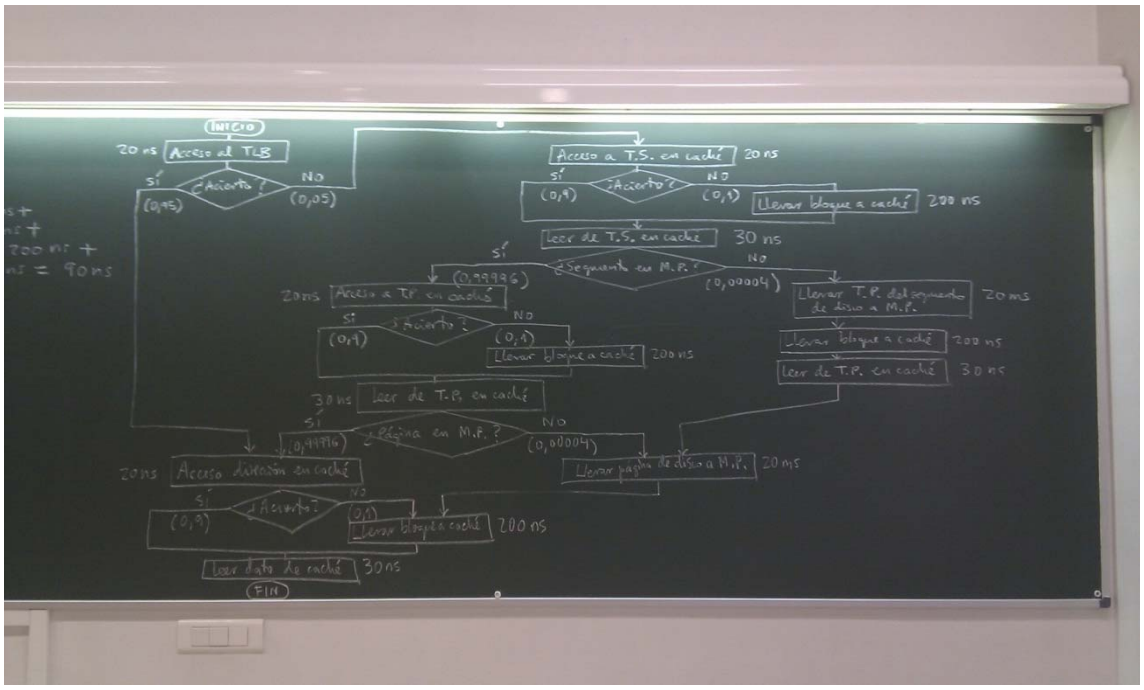
Para VM, $2^{44}=2^{22}x2^{10}x2^{12}$. Para cache, $2^{30}=2^{14}x2^{11}x2^{5}$, teniendo que sacar antes que hay 2^{11} conjuntos en cache. Para paginación, $230=2^{18}x2^{12}$.

Para c), se gastan 20ns en ver que hay acierto TLB, 20ns para ver si hay acierto cache, y dependiendo, si hay acierto (10%) sólo se gasta 30ns más, y si no (90%) además hay que traer el bloque 200ns. Ver b)

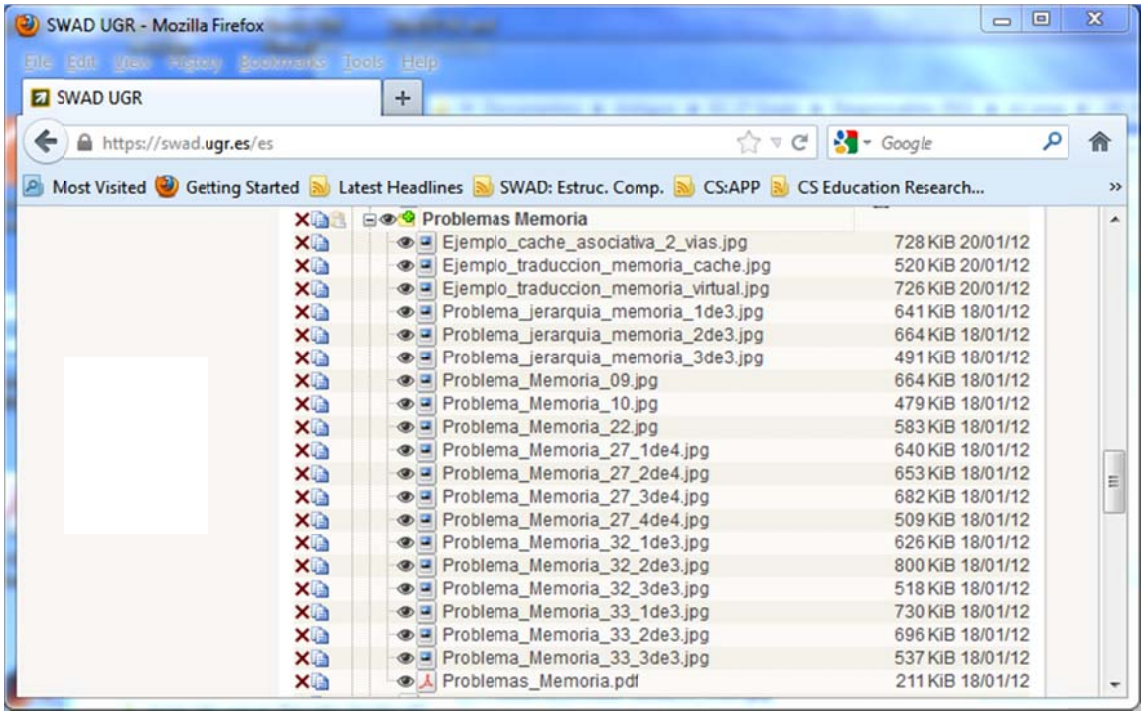


En la figura 2 se responde b). Dada una dirección VM=segm.pag.despl, para leer el dato en dirección MP=pag-fisica.despl la MMU intenta:

1) Buscar dir.MV en TLB (T_{TLB}=20ns, A_{TLB}=95%), en cuyo caso se obtiene la traducción VM→MP, y buscar dir.MP en cache (ver 11). Se ahorra (2-10) traducir otra vez usando tabla segms/págs.
2) Fallo TLB: VM=segm.pag.despl: Indexar con segm. en tabla segmentos a ver dónde está su tabla de páginas (bloques de la tabla segm. pueden estar en cache, $T_{dir.cach}$=20ns, suponer A_{cache}=90% también para tabla de segmentos), si no está, traer ese bloque a cache (T_{bloq}=200ns), y en cualquier caso indexar con segm. ($T_{pal.cach}$=30ns, tras T_{bloq} no se accede otra vez a directorio)
3) Bit "presente" indica si tabla páginas de segm. está en MP o en disco, y la dirección (MP/disco). Si no está en MP, (ver 4). Si está, (ver 8). Se ahorra (4-7) traer tabla de páginas a MP. (suponer A_{MP}=99.996% también para tablas páginas)
4) Traer tabla pag. de ese segm. de disco a MP (T_{disco}=20ms)
5) Para poder indexar con pag. en tabla páginas de ese segm.: Traer bloque a cache (T_{bloq}=200ns)
6) Indexar con pag. en tabla páginas ($T_{pal.cach}$=30ns, tras T_{bloq} no se accede a directorio)
7) Ya sabemos traducir VM→MP, seguir por (ver 10)
8) Para traducir VM=segm.pag.despl → MP=pag-fisica.despl : Indexar con pag. en tabla páginas de ese segm. (bloques de la tabla pueden estar en cache, $T_{dir.cach}$=20ns, suponer A_{cache}=90% también para tablas de páginas), si no está, traer esa parte a cache (T_{bloq}=200ns), y en cualquier caso indexar con pag. ($T_{pal.cach}$=30ns, tras T_{bloq} no se accede otra vez a directorio)
9) Bit "presente" indica si esa pag. de ese segm. está en MP o en disco, y la dirección (MP/disco). Si no está en MP, (ver 10). Si está, (ver 11). Se ahorra (10) traer página a MP. (suponer A_{MP}=99.996% también para tablas)
10) Traer pag. de ese segm. de disco a MP (T_{disco}=20ms). Seguir por (ver 11 "traer ese bloque")
11) Buscar MP=pag-fisica.despl en cache ($T_{dir.cach}$=20ns, A_{cache}=90%), si no está, traer ese bloque a cache (T_{bloq}=200ns), y en cualquier caso leer el dato. ($T_{pal.cach}$=30ns, tras T_{bloq} no se accede otra vez a directorio)



Lista de imágenes



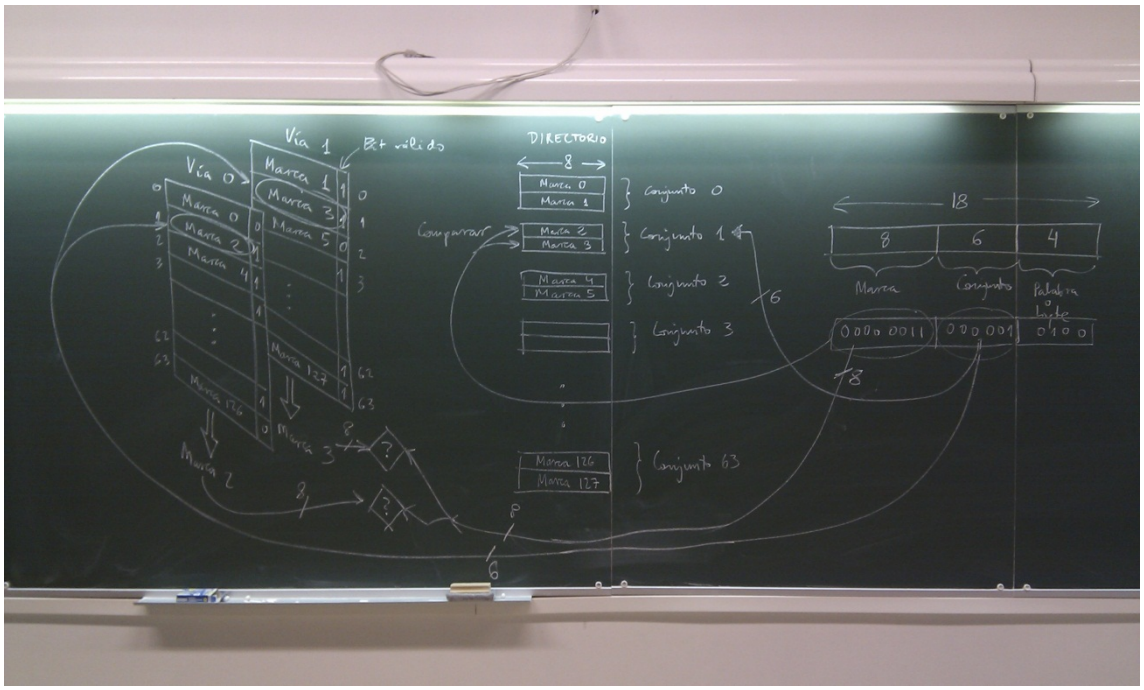
Ejemplo cache asociativa 2 vías

La figura muestra (parte derecha) un ejemplo con direcciones (físicas) de 18bits=8+6+4 ⇒ 256Kpal de espacio MP, etiquetas cache de 8bit, 64 conjuntos de 16palabras (bytes, si la memoria es de bytes, y entonces serían 256KB de MP).

Se indica que la cache es de 2 vías (parte izquierda), así que la cache sería de tamaño 2^6conj x 2^4 B/conj x 2vías = 2^{11} B = 2KB. Cada vía tiene 64 marcos para albergar hasta 64 bloques de MP. Los marcos rellenos se indican activando el bit "válido", y copiando la etiqueta ("marca") para poder indentificarlos después. Cada bloque MP puede ir al conjunto que indiquen sus bits A9-A4, y tiene 2 oportunidades (vía0 ó vía1). Para comprobar si un bloque está, basta mirar las 2 etiquetas de su conjunto (si ambas son válidas).

En la parte central se muestra el mismo ejemplo de relleno (las mismas etiquetas o “marcas” que en la parte izquierda), y se está comprobando si la dirección mostrada a la derecha, 0x00C14, está en cache. Es inmediato comprobar que esa posición es la 4-ésima (5ª) del bloque de MP 0x00C1, al cual le corresponde estar en el conjunto 1-ésimo (2º), con etiqueta ó “marca” 0x03. Efectivamente, de los 2 bloques que hay en el conjunto 1 (ambos válidos), en la vía 1 está la marca 0x03, así que queremos el byte 4 de ese marco de bloque.

Como la etiqueta 0x02 también está, es inmediato razonar que también están en caché todas las posiciones MP desde 0x00810-0x0081F, en conjunto 1 vía 0.



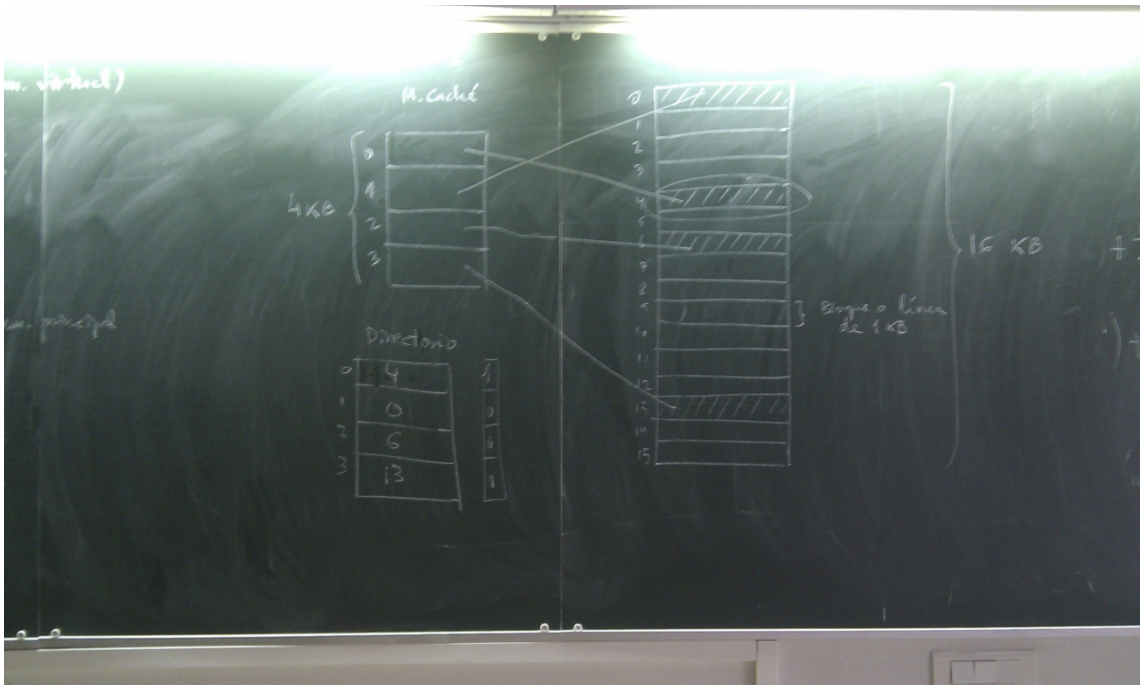
Ejemplo traducción memoria cache

La figura muestra una cache de 4KB (izq.) con 4 bloques (se deduce de 1KB), por tanto con un directorio de 4 etiquetas y 4 bits “válido”. La MP (der.) es de 16KB, de forma que aunque no lo ponga la figura, las direcciones son de 14bits=4(etiqueta)+10(byte en bloque), así que en el directorio cache las etiquetas son de 4bits (1 dígito hex). Se trata de una cache totalmente asociativa.

La memoria tiene por tanto 16bloques de 1KB, y en concreto los que están en cache ahora mismo son los bloques 0,4,6 y 13. Por algún motivo se han marcado como no válidos los marcos 1 y 2, que contenían los bloques 0 y 6. Puede que en el ejemplo se dijera que inicialmente la cache contenía los bloques 4 y 13, y se preguntara qué pasaría si el procesador accede a las direcciones 0x0000 y 0x1800 (primera dir. de los bloques 0 y 6). Sucedería que se traerían los bloques a cualquier marco libre (no hay que expulsar ningún bloque de cache, porque habría 2 libres), se rellenarían las etiquetas, y se marcarían como válidos (parece que este último paso se hubiera olvidado).

Es frecuente que haya otro bit “sucio” para saber si se ha modificado el bloque en cache, para hacer write-back cuando se expulsa para hacer sitio, y otros bits “LRU” para escoger qué marco se vacía para hacer sitio a un bloque nuevo.

Las posiciones de MP en el marco 2 son ahora las del bloque 6, esto es, 0x1800-0x1BFF. Como ejercicio adicional, se pueden calcular qué posiciones de MP están en cada uno de los 4 marcos.



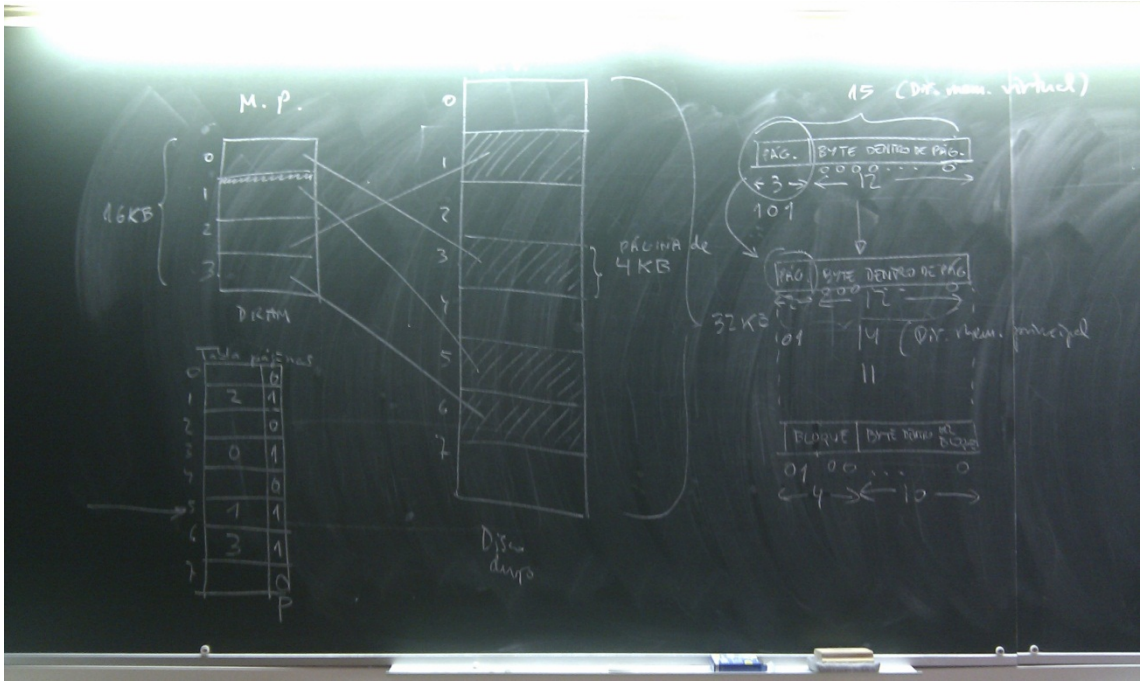
Ejemplo traducción memoria virtual

La figura muestra (izq.) una MP de 16KB gestionada como 4 páginas de 4KB (no se considera segmentación, sólo paginación, como en Linux), y una tabla de páginas con 8 entradas indicando la dirección MP donde está la página y un bit "presente". (Cuando no está presente, la dirección puede indicar en qué parte del fichero ejecutable o de swap está la página deseada). Con ese tamaño de tabla se puede aventurar que el espacio de memoria virtual VM va a ser de 8 páginas, $2^3 x 2^{12} = 2^{15}$B.

En el centro se muestra el fichero de swap en disco duro, con espacio para 8 páginas de VM. En la tabla de páginas se puede comprobar que las páginas rayadas en MV están en donde indican las flechas en MP (pág1VM → p2MP, p3VM → p0MP, p5VM → p1MP, p6VM → p3MP). Esas entradas tienen el bit "presente" activado.

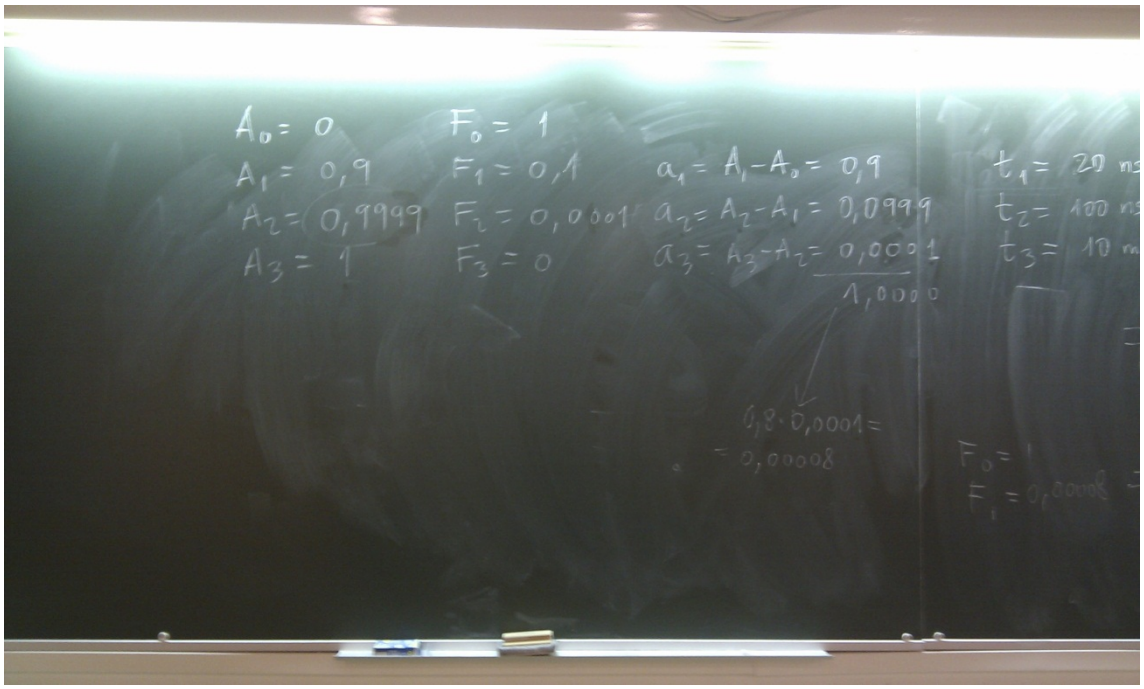
En la parte derecha se puede comprobar que el espacio de VM es en efecto 2^{15}B, repartiendo los bits 15=3+12, con 2^3páginas de 2^{12}bytes. La dirección VM 0x5000 es la dirección 0-ésima (1ª) de la página 5-ésima (6ª). La traducción VM=pag.offset (15bits)→MP=dir-fisica.offset (14bits) consiste en indexar con pag. (3bits) en la tabla de páginas, para obtener dir-fisica. (2bits). Se convierte entonces 0x5000 (15bits) → 0x1000 (14bits).

Por último, si a estos 16KB de MP se le pone una cache como la anterior (14bits=4+10), la posición 0x1000 es la 0-ésima (1ª) del 4-ésimo (5º) bloque de 1KB, que puede estar en cualquier marco de bloque.

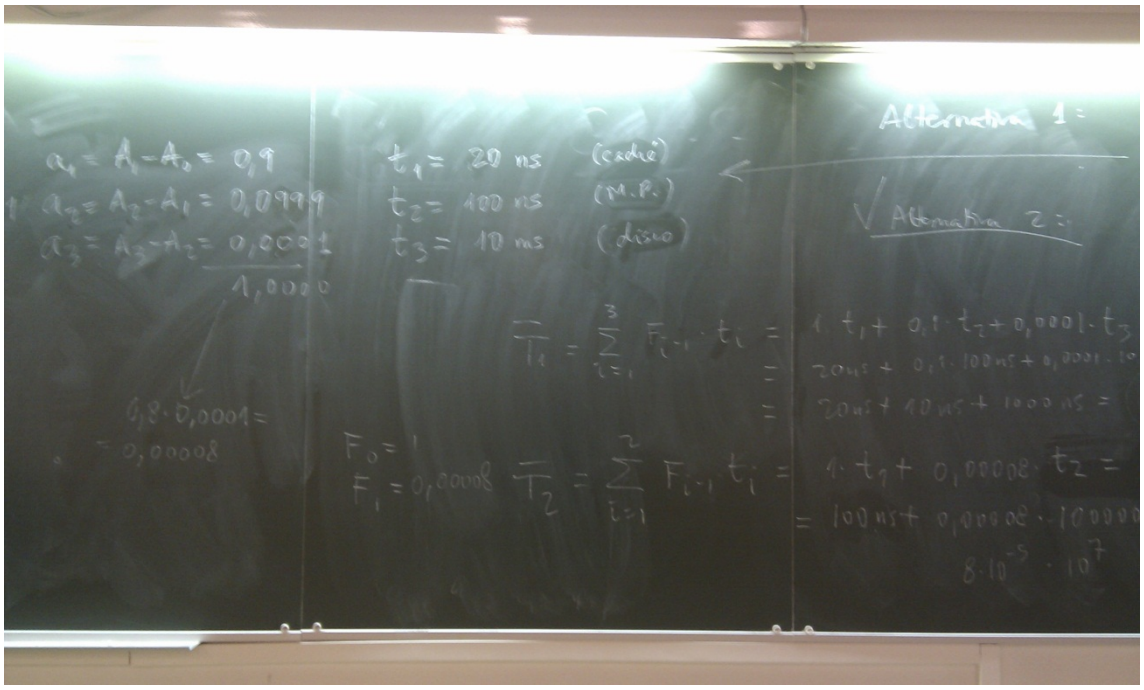


Problema jerarquía memoria

Basándose en el modelo de Chow, Tema6, tr.24-28, suponer estos valores concretos para tasas de acierto acumuladas (A_0-A_3) en una jerarquía de 3 niveles (por ejemplo, cache, MP y fichero swap)



Se pueden calcular inmediatamente las tasas de fallo F_0-F_3, y las tasas de acierto propias de cada nivel (a_1-a_3), que por supuesto deben sumar 1. Si nos proporcionan los tiempos de acceso propios de cada nivel (t_1-t_3)…



…nos basta con los F_i para calcular el tiempo medio de acceso con la fórmula al final de la tr.27. Sale 1030ns (Alternativa 1).

Nos dicen que si ponemos el doble de MP (Alternativa 2), el 20% de los fallos MP dejan de ser fallo. Nos cuestionamos entonces si esa mejora es más interesante que la propia cache. Sale 900ns, así que efectivamente, con esas proporciones de tiempos y fallos, es más eficiente duplicar la memoria que poner la cache (si nos dieran los costes por bit y de interconexión, se podría calcular si es también más barato).