

## 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...

9) 16 MB  
Cada módulo: 1 MB

$\Rightarrow 16 \text{ módulos} = 2^4 \text{ módulos}$

$\frac{24-4=20}{n-m} \quad m$

1º acceso A30010  
2º acceso AB0021  
3º acceso 0010AA  
4º acceso 227AAA  
5º acceso 0101AA  
6º acceso {01016A  
              010163  
← n=24 →

a) 6 accesos

10) a)

## 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  $\tau$  se muestra una dirección a un módulo, recogiendo el resultado  $T_a$  después). Sabiendo eso, como hay 4 módulos, sale 2x y 4x

10) 25K palabras - 4 módulos

Módulo 0	Módulo 1	Módulo 2	Módulo 3
00000	00001	00002	00003
00004	00005	00006	00007
00008	00009	00010	00011
00012	00013	00014	00015
00016	00017	00018	00019
00020	00021	00022	00023
00024	00025	00026	00027
00028	00029	00030	00031
00032	00033	00034	00035
00036	00037	00038	00039
00040	00041	00042	00043
00044	00045	00046	00047
00048	00049	00050	00051
00052	00053	00054	00055
00056	00057	00058	00059
00060	00061	00062	00063
00064	00065	00066	00067
00068	00069	00070	00071
00072	00073	00074	00075
00076	00077	00078	00079
00080	00081	00082	00083
00084	00085	00086	00087
00088	00089	00090	00091
00092	00093	00094	00095
00096	00097	00098	00099
00100	00101	00102	00103
00104	00105	00106	00107
00108	00109	00110	00111
00112	00113	00114	00115
00116	00117	00118	00119
00120	00121	00122	00123
00124	00125	00126	00127
00128	00129	00130	00131
00132	00133	00134	00135
00136	00137	00138	00139
00140	00141	00142	00143
00144	00145	00146	00147
00148	00149	00150	00151
00152	00153	00154	00155
00156	00157	00158	00159
00160	00161	00162	00163
00164	00165	00166	00167
00168	00169	00170	00171
00172	00173	00174	00175
00176	00177	00178	00179
00180	00181	00182	00183
00184	00185	00186	00187
00188	00189	00190	00191
00192	00193	00194	00195
00196	00197	00198	00199
00200	00201	00202	00203
00204	00205	00206	00207
00208	00209	00210	00211
00212	00213	00214	00215
00216	00217	00218	00219
00220	00221	00222	00223
00224	00225	00226	00227
00228	00229	00230	00231
00232	00233	00234	00235
00236	00237	00238	00239
00240	00241	00242	00243
00244	00245	00246	00247
00248	00249	00250	00251
00252	00253	00254	00255
00256	00257	00258	00259
00260	00261	00262	00263
00264	00265	00266	00267
00268	00269	00270	00271
00272	00273	00274	00275
00276	00277	00278	00279
00280	00281	00282	00283
00284	00285	00286	00287
00288	00289	00290	00291
00292	00293	00294	00295
00296	00297	00298	00299
00300	00301	00302	00303
00304	00305	00306	00307
00308	00309	00310	00311
00312	00313	00314	00315
00316	00317	00318	00319
00320	00321	00322	00323
00324	00325	00326	00327
00328	00329	00330	00331
00332	00333	00334	00335
00336	00337	00338	00339
00340	00341	00342	00343
00344	00345	00346	00347
00348	00349	00350	00351
00352	00353	00354	00355
00356	00357	00358	00359
00360	00361	00362	00363
00364	00365	00366	00367
00368	00369	00370	00371
00372	00373	00374	00375
00376	00377	00378	00379
00380	00381	00382	00383
00384	00385	00386	00387
00388	00389	00390	00391
00392	00393	00394	00395
00396	00397	00398	00399
00400	00401	00402	00403
00404	00405	00406	00407
00408	00409	00410	00411
00412	00413	00414	00415
00416	00417	00418	00419
00420	00421	00422	00423
00424	00425	00426	00427
00428	00429	00430	00431
00432	00433	00434	00435
00436	00437	00438	00439
00440	00441	00442	00443
00444	00445	00446	00447
00448	00449	00450	00451
00452	00453	00454	00455
00456	00457	00458	00459
00460	00461	00462	00463
00464	00465	00466	00467
00468	00469	00470	00471
00472	00473	00474	00475
00476	00477	00478	00479
00480	00481	00482	00483
00484	00485	00486	00487
00488	00489	00490	00491
00492	00493	00494	00495
00496	00497	00498	00499
00500	00501	00502	00503
00504	00505	00506	00507
00508	00509	00510	00511
00512	00513	00514	00515
00516	00517	00518	00519
00520	00521	00522	00523
00524	00525	00526	00527
00528	00529	00530	00531
00532	00533	00534	00535
00536	00537	00538	00539
00540	00541	00542	00543
00544	00545	00546	00547
00548	00549	00550	00551
00552	00553	00554	00555
00556	00557	00558	00559
00560	00561	00562	00563
00564	00565	00566	00567
00568	00569	00570	00571
00572	00573	00574	00575
00576	00577	00578	00579
00580	00581	00582	00583
00584	00585	00586	00587
00588	00589	00590	00591
00592	00593	00594	00595
00596	00597	00598	00599
00600	00601	00602	00603
00604	00605	00606	00607
00608	00609	00610	00611
00612	00613	00614	00615
00616	00617	00618	00619
00620	00621	00622	00623
00624	00625	00626	00627
00628	00629	00630	00631
00632	00633	00634	00635
00636	00637	00638	00639
00640	00641	00642	00643
00644	00645	00646	00647
00648	00649	00650	00651
00652	00653	00654	00655
00656	00657	00658	00659
00660	00661	00662	00663
00664	00665	00666	00667
00668	00669	00670	00671
00672	00673	00674	00675
00676	00677	00678	00679
00680	00681	00682	00683
00684	00685	00686	00687
00688	00689	00690	00691
00692	00693	00694	00695
00696	00697	00698	00699
00700	00701	00702	00703
00704	00705	00706	00707
00708	00709	00710	00711
00712	00713	00714	00715
00716	00717	00718	00719
00720	00721	00722	00723
00724	00725	00726	00727
00728	00729	00730	00731
00732	00733	00734	00735
00736	00737	00738	00739
00740	00741	00742	00743
00744	00745	00746	00747
00748	00749	00750	00751
00752	00753	00754	00755
00756	00757	00758	00759
00760	00761	00762	00763
00764	00765	00766	00767
00768	00769	00770	00771
00772	00773	00774	00775
00776	00777	00778	00779
00780	00781	00782	00783
00784	00785	00786	00787
00788	00789	00790	00791
00792	00793	00794	00795
00796	00797	00798	00799
00800	00801	00802	00803
00804	00805	00806	00807
00808	00809	00810	00811
00812	00813	00814	00815
00816	00817	00818	00819
00820	00821	00822	00823
00824	00825	00826	00827
00828	00829	00830	00831
00832	00833	00834	00835
00836	00837	00838	00839
00840	00841	00842	00843
00844	00845	00846	00847
00848	00849	00850	00851
00852	00853	00854	00855
00856	00857	00858	00859
00860	00861	00862	00863
00864	00865	00866	00867
00868	00869	00870	00871
00872	00873	00874	00875
00876	00877	00878	00879
00880	00881	00882	00883
00884	00885	00886	00887
00888	00889	00890	00891
00892	00893	00894	00895
00896	00897	00898	00899
00900	00901	00902	00903
00904	00905	00906	00907
00908	00909	00910	00911
00912	0091		

## 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.

Handwritten solution for Problema Memoria 22 on a chalkboard. The problem states: M.P.: 4 GB, M.Cache: 16 MB, Tam bloque: 256 B.

**a) Totalmente asociativa**

Diagram: ETIQUETA (24 bits) | BYTE (8 bits). Total 32 bits.

**b) Directa**

Diagram: ETIQUETA (8 bits) | N.º DE BLOQUE (16 bits) | BYTE (8 bits). Total 32 bits.

**c) Asociativa por conjuntos, 8 vías**

Diagram: ETIQUETA (11 bits) | CONJUNTO (13 bits) | BYTE (8 bits). Total 32 bits.

**d) Sectores, (16) bloques por sector**

Diagram: ETIQUETA O SECTOR (20 bits) | BLOQUE (4 bits) | BYTE (8 bits). Total 32 bits.

Calculations:

- $2^{32} \text{ B} \Rightarrow \text{direcciones de 32 bits}$
- $\frac{2^{32} \text{ B}}{2^8 \text{ B/bloque}} = 2^{24} \text{ bloques en M.P.}$
- $\frac{2^{24} \text{ B}}{2^8 \text{ B/bloque}} = 2^{16} \text{ conjuntos de bloques en cache}$
- $\frac{2^{16} \text{ m. de bloque}}{2^4 \text{ m. de bloque conjunto}} = 2^{12} \text{ conjuntos en cache}$
- $\frac{2^{16} \text{ m. bloque}}{2^4 \text{ m. bloque sector}} = 2^{12} \text{ sectores en cache}$
- $\frac{2^{24} \text{ bloques}}{2^4 \text{ bloques sector}} = 2^{20} \text{ sectores en M.P.}$

## Lista de imágenes

SWAD UGR - Mozilla Firefox

File Edit View History Bookmarks Tools Help

SWAD UGR

https://swad.ugr.es/es

Most Visited Getting Started Latest Headlines SWAD: Estruct. Comp. CS:APP CS Education Research...

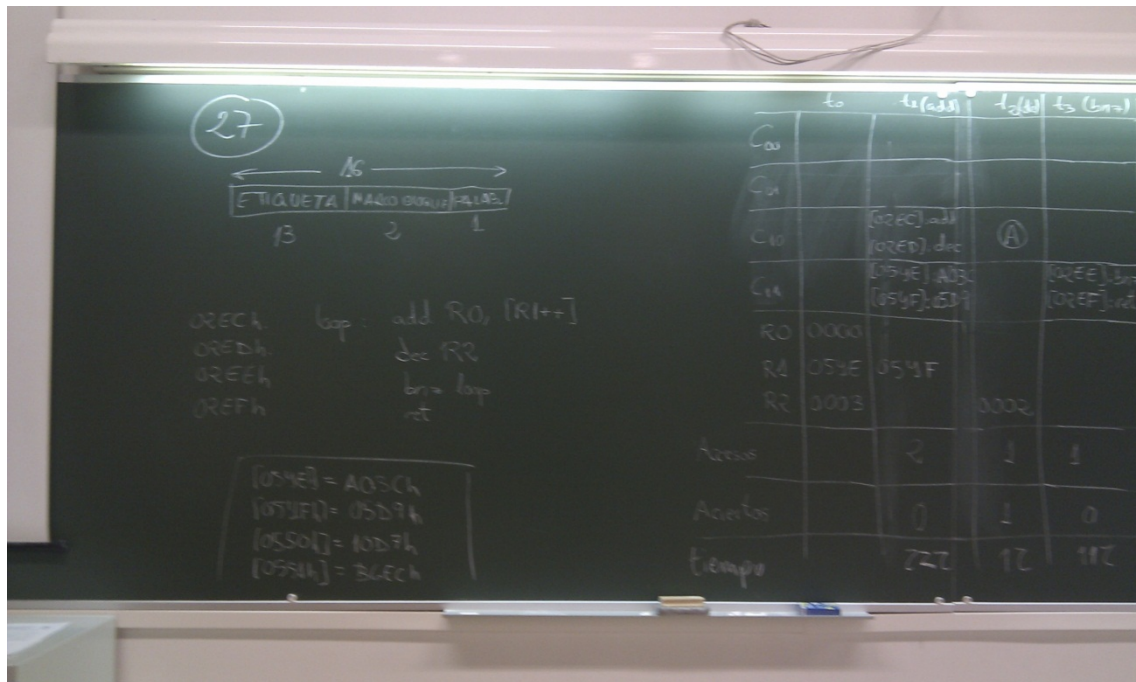
Problemas Memoria

File Name	Size	Date
Ejemplo_cache_asociativa_2_vias.jpg	728 KIB	20/01/12
Ejemplo_traduccion_memoria_cache.jpg	520 KIB	20/01/12
Ejemplo_traduccion_memoria_virtual.jpg	726 KIB	20/01/12
Problema_jerarquia_memoria_1de3.jpg	641 KIB	18/01/12
Problema_jerarquia_memoria_2de3.jpg	664 KIB	18/01/12
Problema_jerarquia_memoria_3de3.jpg	491 KIB	18/01/12
Problema_Memoria_09.jpg	664 KIB	18/01/12
Problema_Memoria_10.jpg	479 KIB	18/01/12
Problema_Memoria_22.jpg	583 KIB	18/01/12
Problema_Memoria_27_1de4.jpg	640 KIB	18/01/12
Problema_Memoria_27_2de4.jpg	653 KIB	18/01/12
Problema_Memoria_27_3de4.jpg	682 KIB	18/01/12
Problema_Memoria_27_4de4.jpg	509 KIB	18/01/12
Problema_Memoria_32_1de3.jpg	626 KIB	18/01/12
Problema_Memoria_32_2de3.jpg	800 KIB	18/01/12
Problema_Memoria_32_3de3.jpg	518 KIB	18/01/12
Problema_Memoria_33_1de3.jpg	730 KIB	18/01/12
Problema_Memoria_33_2de3.jpg	696 KIB	18/01/12
Problema_Memoria_33_3de3.jpg	537 KIB	18/01/12
Problemas_Memoria.pdf	211 KIB	18/01/12

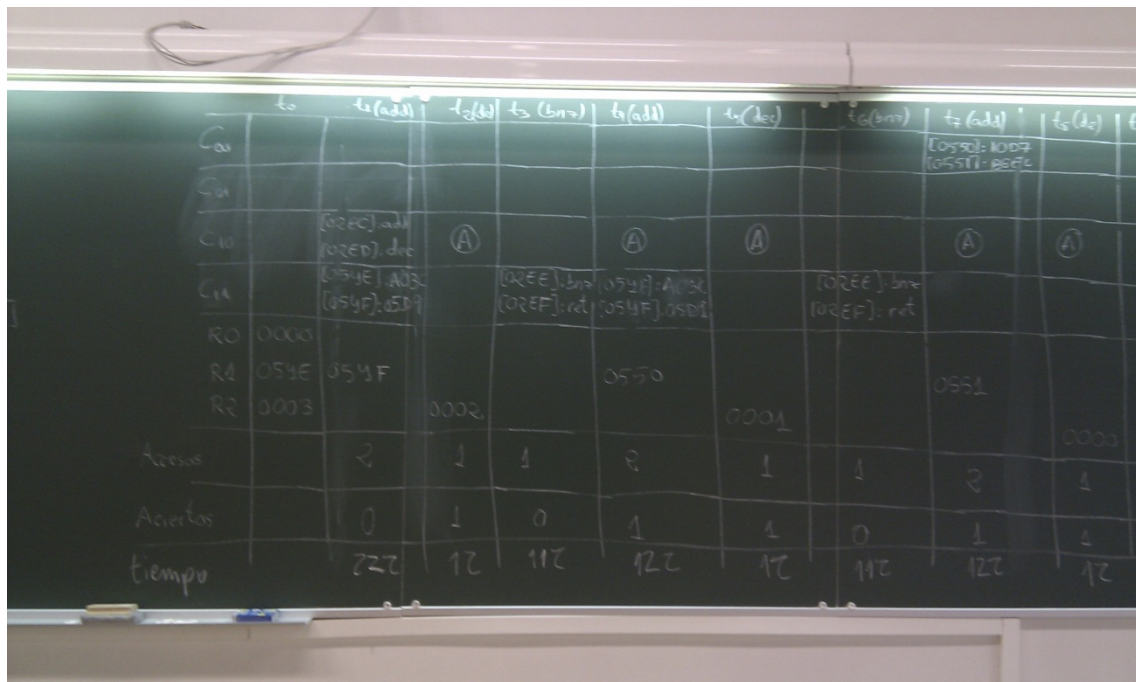
## Problema Memoria 27

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 t<sub>1</sub>, dec R2 (acierto al captar la instrucción, t<sub>2</sub>), captar bnz (02EEh machaca C11, t<sub>3</sub>), acierto add, fallo [R1++] t<sub>4</sub>, etc...

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

$t_1(Add)$	$t_1(Dec)$	$t_6(brr)$	$t_7(Add)$	$t_8(Dec)$	$t_7(brr)$	$t_8(ret)$
			$[0550]: 1007$ $[0551]: 88FC$			
(A)	(A)		(A)	(A)		
$brr[0548]: A'3C$ $ret[054F]: 05D1$		$[02EE]: brr$ $[02EF]: ret$			(A)	(A)
0550			0551			
	0001			0000		
0	1	1	2	4	1	1
1	1	0	1	4	1	1
127	17	17	127	17	17	17

b)

$$\bar{T} = A \cdot t_c + (1-A) \cdot (t_c + t_m)$$

$$\bar{T} = A t_c + (1-A) \cdot (5 + 107)$$

$$\bar{T} = A t_c + (1-A) \cdot 117$$

$$\bar{T} = 117 - 10A7$$

$$\bar{T} = 117 - 5'387$$

$$\bar{T} = 5'627$$

$$t = n^{\circ} \text{accesos} \times \bar{T} = 13 \times 5'627 = 737$$

$$\Rightarrow 13 \text{ accesos}$$

$$\Rightarrow 7 \text{ aciertos}$$

$$\Rightarrow 737$$

$$A = \frac{7}{13} = 0'538$$

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  $13 \cdot T_{medio} = 73\tau$ , como que la fórmula usada es justo lo que hemos ido anotando en "tiempo")

$t_8(Dec)$	$t_7(brr)$	$t_8(ret)$
$brr[0548]: A'3C$ $ret[054F]: 05D1$		
(A)		
	(A)	(A)
0000		
4	1	1
4	1	1
17	17	17

b)

$$\bar{T} = A \cdot t_c + (1-A) \cdot (t_c + t_m)$$

$$\bar{T} = A t_c + (1-A) \cdot (5 + 107)$$

$$\bar{T} = A t_c + (1-A) \cdot 117$$

$$\bar{T} = 117 - 10A7$$

$$\bar{T} = 117 - 5'387$$

$$\bar{T} = 5'627$$

$$t = n^{\circ} \text{accesos} \times \bar{T} = 13 \times 5'627 = 737$$

$$\Rightarrow 13 \text{ accesos}$$

$$\Rightarrow 7 \text{ aciertos}$$

$$\Rightarrow 737$$

$$A = \frac{7}{13} = 0'538$$

02EC 0000 0010 1110 1107  
MARCO DE BLOQUE

02ED — — — 1101

02EE — — — 1110

02EF — — — 1111

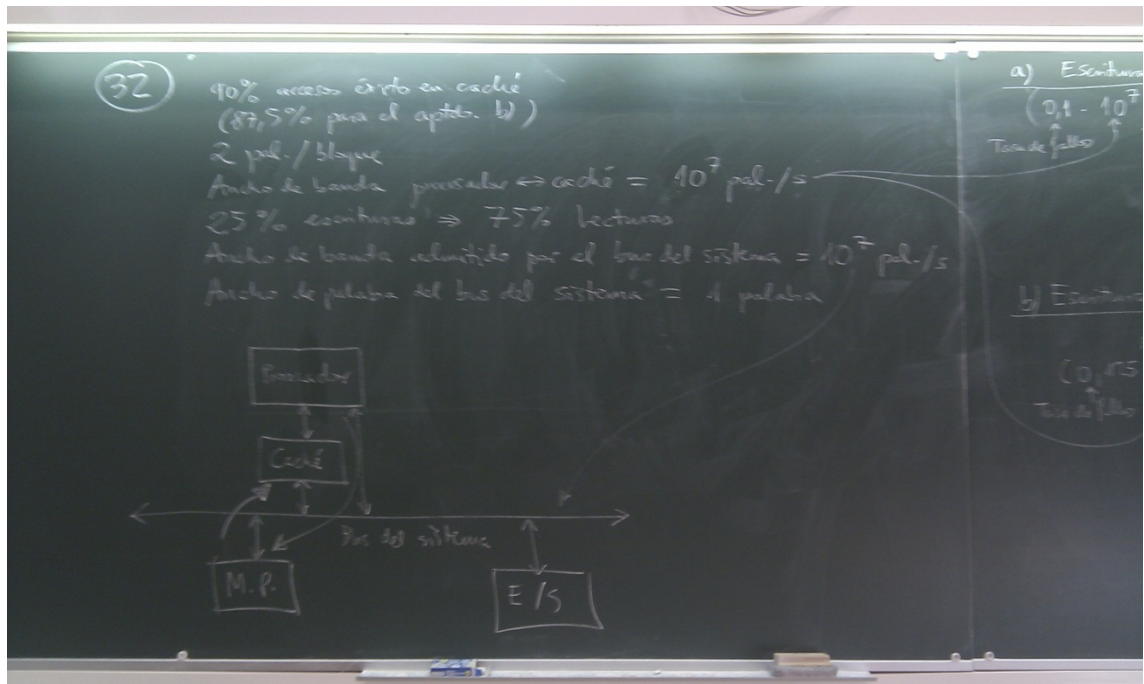
054E (= R1) 0000 0101 0100 1110

0550 0000 0101 0101 0000

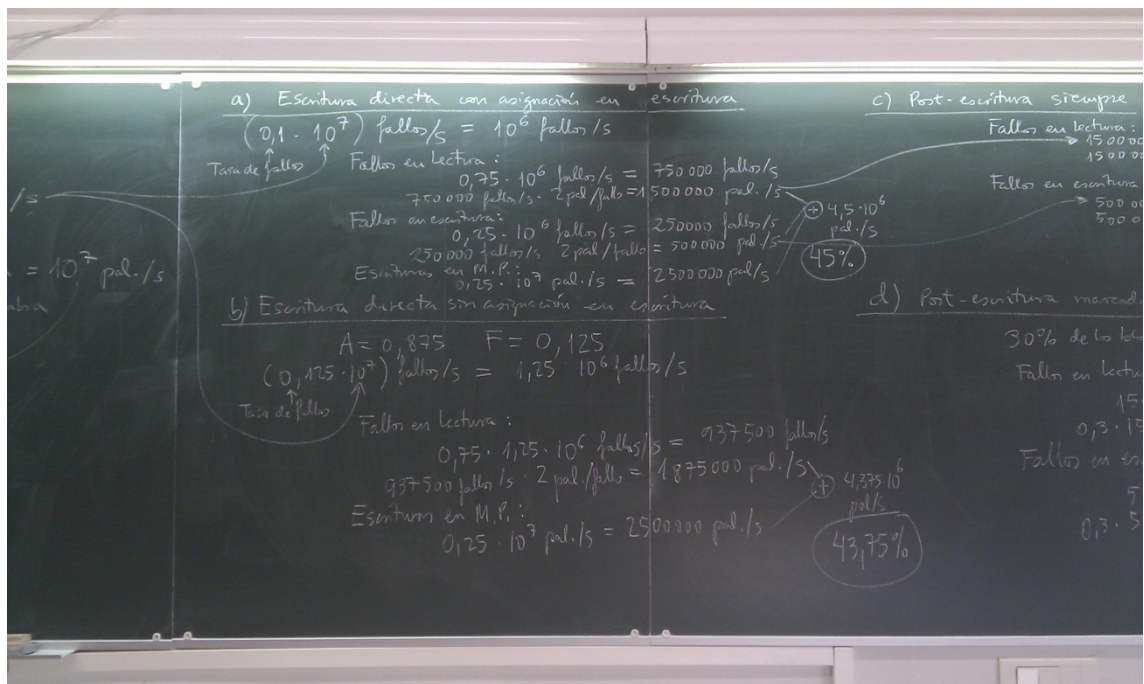
## Problema Memoria 32

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



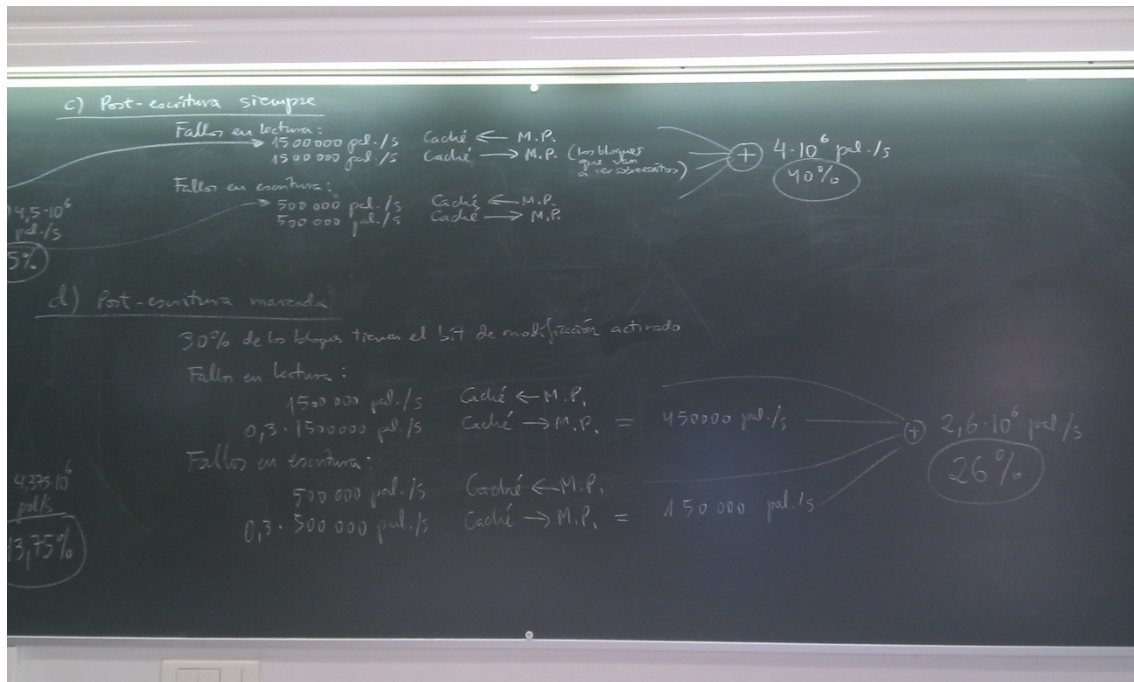


En la figura 2 se responden a) y b). Para a), recordar que “Con asignación” significa que se trae el bloque en fallo en escritura. Como cada bloque son 2 pal, cada fallo lectura son 2pal, cada fallo escritura también (por “con asignación”), y además escritura a memoria (“write-through”). Sale 45%



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

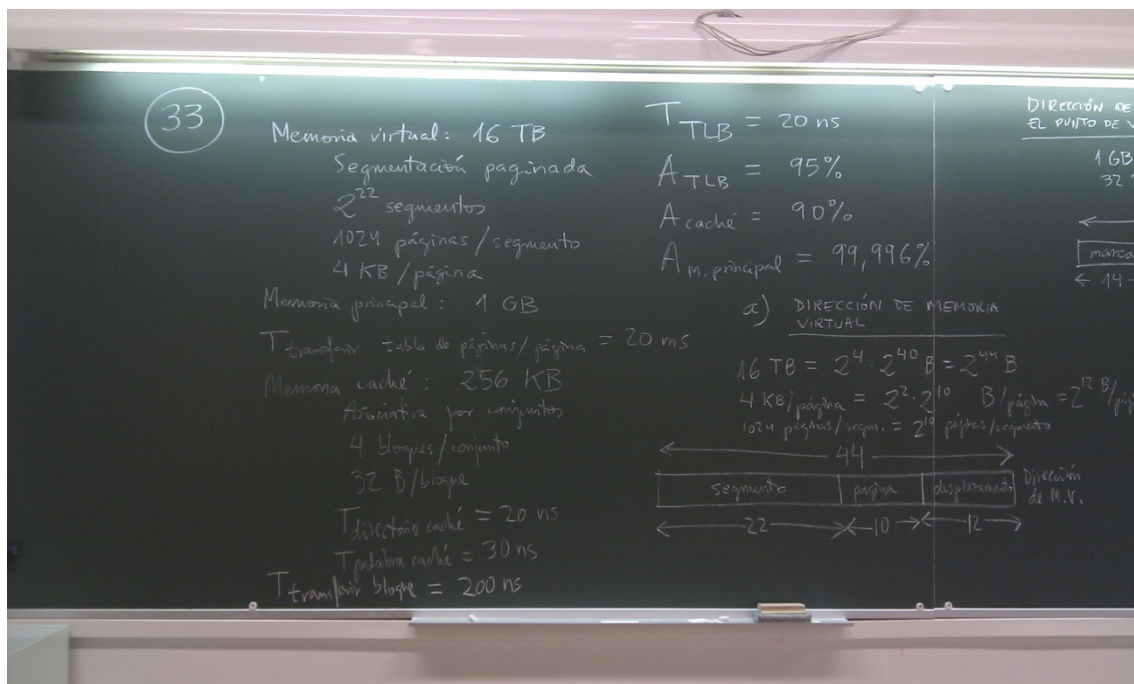
En la figura 3 se responden c) y d). Para c), se asume que la caché está llena, y cualquier fallo provoca que haya que post-escribir el bloque (además de traerse el que falla). En fallo en escritura hay que traerse el bloque para poder escribir (y al estar llena la caché, hay que post-escribir el marco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

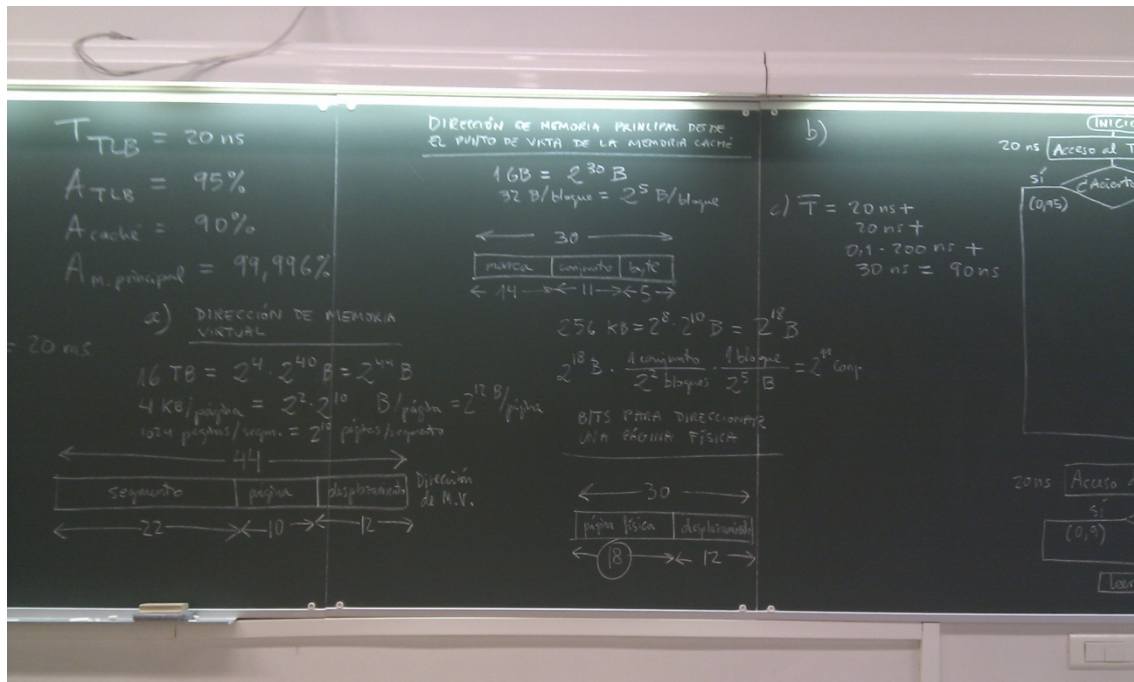


En la figura 2 se responde a) y c).

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

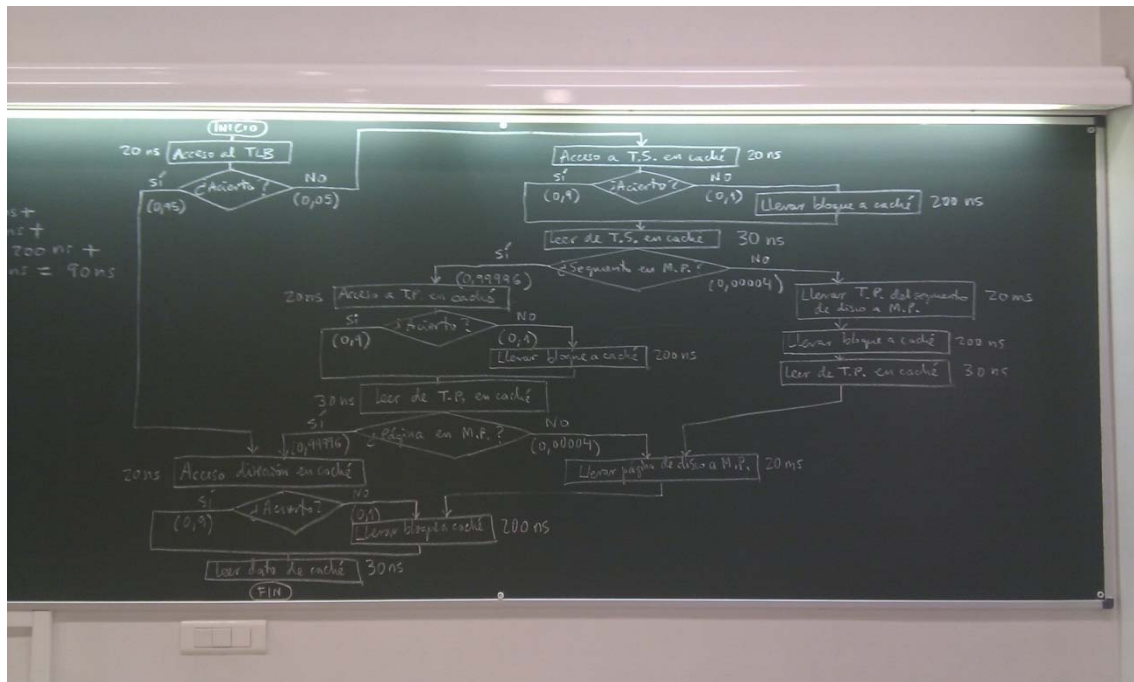
Para c), se gastan 20ns en ver que hay acierto TLB, 20ns para ver si hay acierto cache, y dependiendo, si hay acierto (10%) sólo se gasta 30ns más, y si no (90%) además hay que traer el bloque 200ns. Ver b)



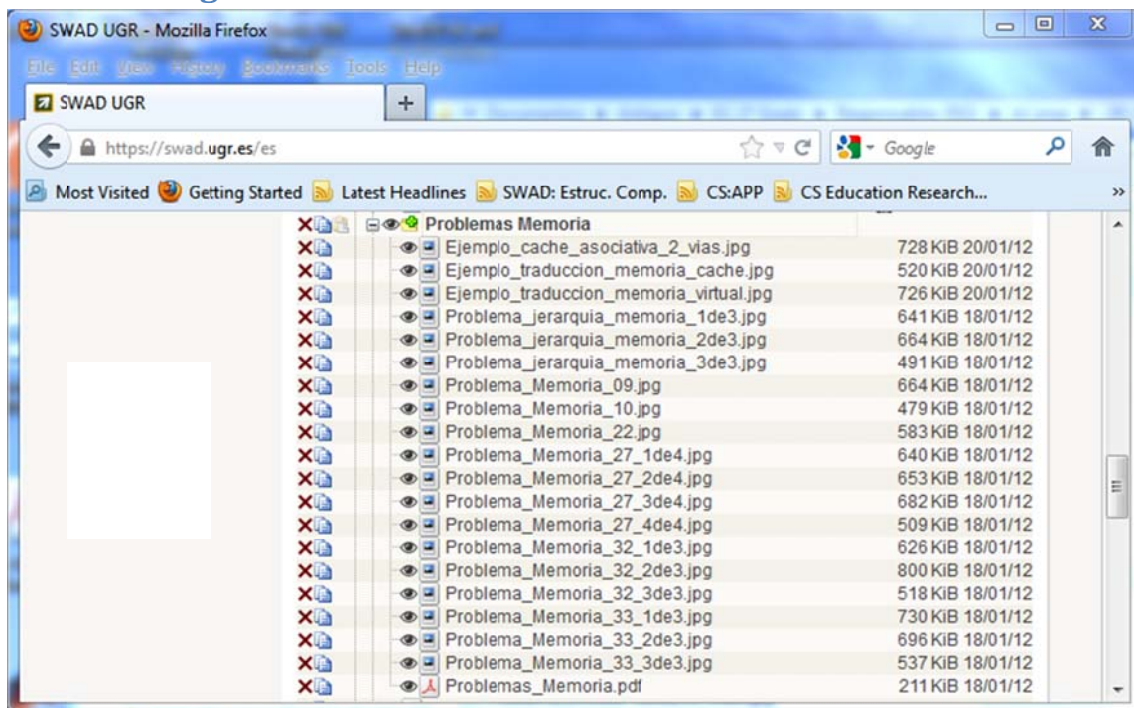


En la figura 2 se responde b). Dada una **dirección VM=segm.pag.despl**, para leer el dato en **dirección MP=pag-fisica.despl** la MMU intenta:

- 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  $A_{cach}=90\%$  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  $A_{cach}=90\%$  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 ( $T_{dir.cach}=20ns$ ,  $A_{cach}=90\%$ ), 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

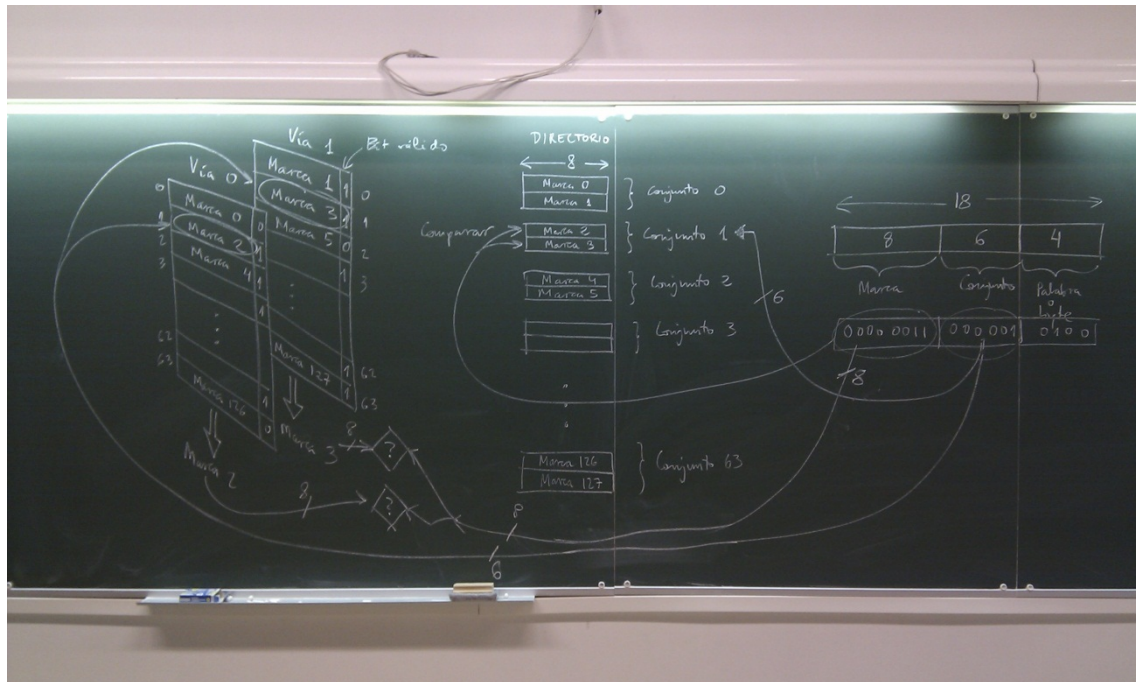
La figura muestra (parte derecha) un ejemplo con direcciones (físicas) de 18bits=8+6+4  $\Rightarrow$  256Kpal de espacio MP, etiquetas cache de 8bit, 64 conjuntos de 16palabras (bytes, si la memoria es de bytes, y entonces serían 256KB de MP).

Se indica que la cache es de 2 vías (parte izquierda), así que la cache sería de tamaño  $2^6 \text{conj} \times 2^4 \text{B/conj} \times 2 \text{vías} = 2^{11} \text{B} = 2\text{KB}$ . 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.



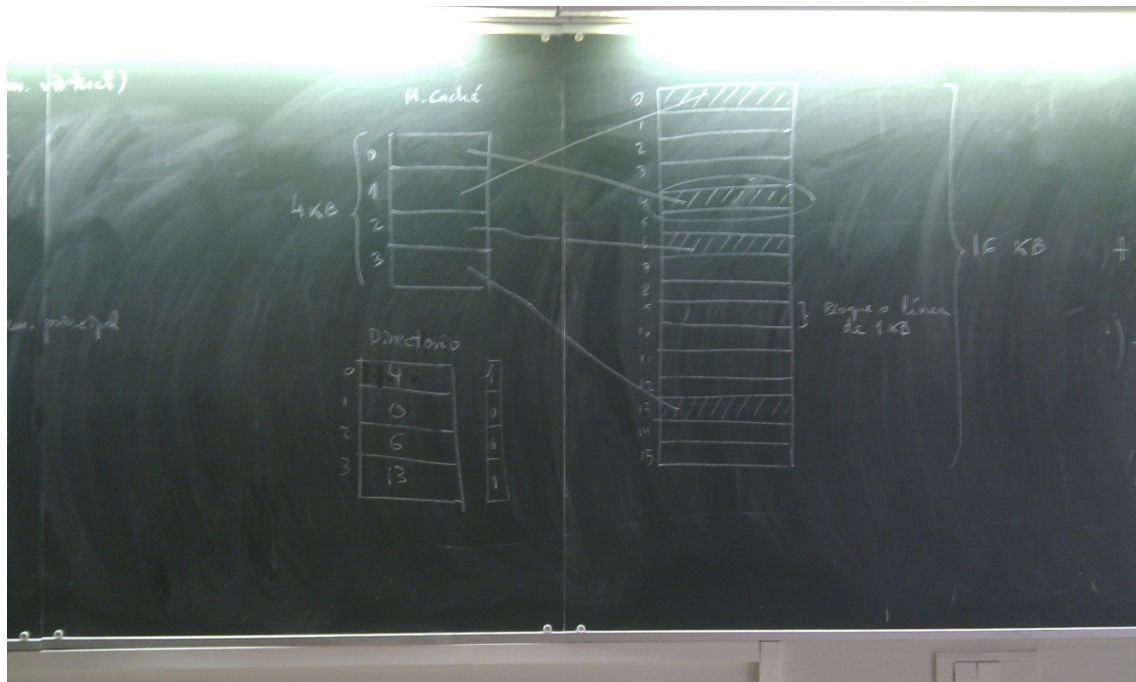
## Ejemplo traducción memoria cache

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 16 bloques 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

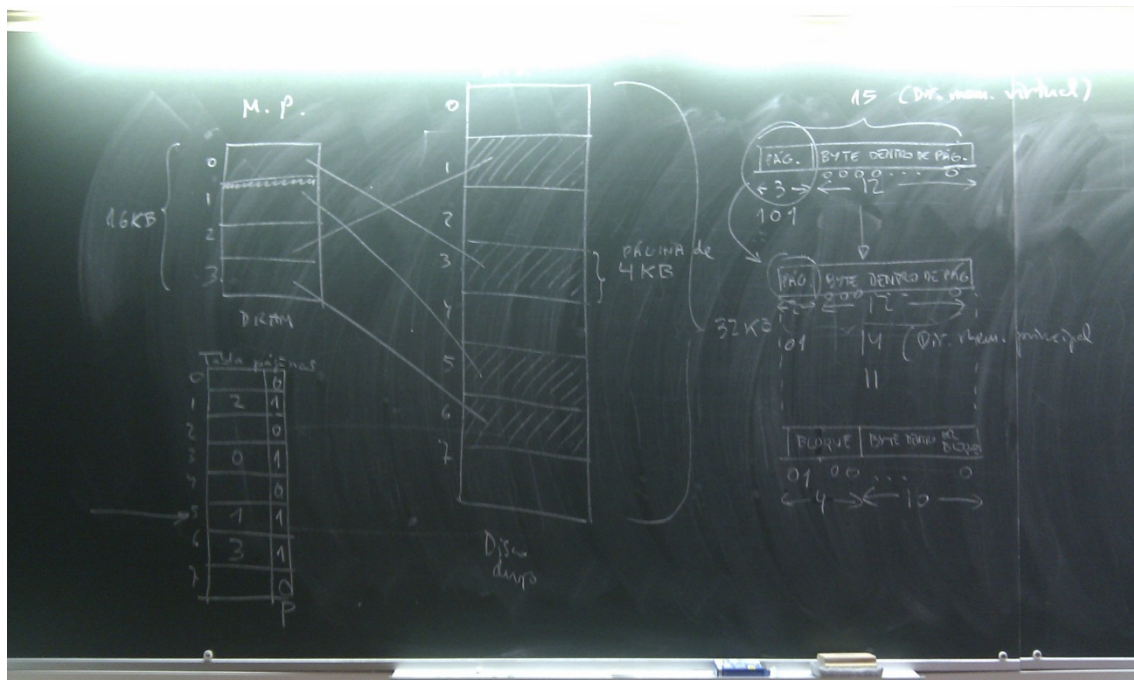
La figura muestra (izq.) una MP de 16KB gestionada como 4 páginas de 4KB (no se considera segmentación, sólo paginación, como en Linux), y una tabla de páginas con 8 entradas indicando la dirección MP donde está la página y un bit "presente". (Cuando no está presente, la dirección puede indicar en qué 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,  $2^3 \times 2^{12} = 2^{15}B$ .

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  $\rightarrow$  p2MP, p3VM  $\rightarrow$  p0MP, p5VM  $\rightarrow$  p1MP, p6VM  $\rightarrow$  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  $2^3$  páginas de  $2^{12}$  bytes. La dirección VM 0x5000 es la dirección 0-ésima ( $1^a$ ) de la página 5-ésima ( $6^a$ ). La traducción **VM=pag.offset** (15bits)  $\rightarrow$  **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)  $\rightarrow$  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 ( $1^a$ ) del 4-ésimo ( $5^o$ ) 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)

$$\begin{array}{llll}
 A_0 = 0 & F_0 = 1 & & \\
 A_1 = 0,9 & F_1 = 0,1 & a_1 = A_1 - A_0 = 0,9 & t_1 = 20 \text{ ns} \\
 A_2 = 0,9999 & F_2 = 0,0001 & a_2 = A_2 - A_1 = 0,0999 & t_2 = 100 \text{ ns} \\
 A_3 = 1 & F_3 = 0 & a_3 = A_3 - A_2 = 0,0001 & t_3 = 10 \text{ ms}
 \end{array}$$

$$\begin{array}{l}
 1,0000 \\
 \hline
 0,8 \cdot 0,0001 = \\
 = 0,00008
 \end{array}$$

$$\begin{array}{l}
 F_0 = 1 \\
 F_1 = 0,00008
 \end{array}$$

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 ( $t_1$ - $t_3$ )...

$\alpha_1 = A_1 - A_0 = 0,9$   
 $\alpha_2 = A_2 - A_1 = 0,0999$   
 $\alpha_3 = A_3 - A_2 = 0,0001$   
 $\frac{1,0000}{1,0000}$   
 $0,8 \cdot 0,0001 = 0,00008$

$t_1 = 20 \text{ ns}$  (cache)  
 $t_2 = 100 \text{ ns}$  (M.P.)  
 $t_3 = 10 \text{ ms}$  (disco)

Alternativa 1:  
 $\checkmark$  Alternativa 2:

$\bar{T}_1 = \sum_{i=1}^3 F_i \cdot t_i = 1 \cdot t_1 + 0,1 \cdot t_2 + 0,0001 \cdot t_3$   
 $= 20 \text{ ns} + 0,1 \cdot 100 \text{ ns} + 0,0001 \cdot 10$   
 $= 20 \text{ ns} + 10 \text{ ns} + 1000 \text{ ns} = 1030 \text{ ns}$

$F_0 = 1$   
 $F_1 = 0,00008$   
 $\bar{T}_2 = \sum_{i=1}^2 F_i \cdot t_i = 1 \cdot t_1 + 0,00008 \cdot t_2$   
 $= 100 \text{ ns} + 0,00008 \cdot 100000$   
 $= 8 \cdot 10^{-5} \cdot 10^7$

...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).

Alternativa 1:  
 Cache  
 M.P.  
 Disco

$\checkmark$  Alternativa 2:

$1 \cdot t_1 + 0,1 \cdot t_2 + 0,0001 \cdot t_3 =$   
 $20 \text{ ns} + 0,1 \cdot 100 \text{ ns} + 0,0001 \cdot 10000000 \text{ ns} =$   
 $20 \text{ ns} + 10 \text{ ns} + 1000 \text{ ns} = 1030 \text{ ns}$

$1 \cdot t_1 + 0,00008 \cdot t_2 =$   
 $100 \text{ ns} + 0,00008 \cdot 10000000 \text{ ns} = 900 \text{ ns}$   
 $8 \cdot 10^{-5} \cdot 10^7$

El doble de M.P. que en la alternativa 1.  
 $\downarrow$   
 El 20% de los accesos que se hacían a disco en la alternativa 1 ahora se encuentran en M.P., sin fallo.  
 $\downarrow$   
 Sigue el 80% de accesos a disco respecto a la alt. 1