

Quadern de laboratori Estructura de Computadors

Emilio Castillo
José María Cella
Montse Fernández
David López
Joan Manuel Parcerisa
Angel Toribio
Rubèn Tous
Jordi Tubella
Gladys Utrera
FIBerHub

Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Quadrimestre de Tardor - Curs 2015/16



Aquest document es troba sota una llicència Creative Commons

Licencia Creative Commons

Esta obra está bajo una licencia Reconocimiento-No comercial-Compartir bajo la misma licencia 2.5 España de Creative Commons. Para ver una copia de esta licencia, visite

<http://creativecommons.org/licenses/by-nc-sa/2.5/es/>

o envíe una carta a

Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Usted es libre de:

- copiar, distribuir y comunicar públicamente la obra
- hacer obras derivadas

Bajo las condiciones siguientes:

- **Reconocimiento.** Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- **No comercial.** No puede utilizar esta obra para fines comerciales.
- **Compartir bajo la misma licencia.** Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.
- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Advertencia: Este resumen no es una licencia. Es simplemente una referencia práctica para entender el Texto Legal (la licencia completa).

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.

Sessió 5: Memòria cache

Objectiu: Realitzar activitats relacionades amb la gestió del nivell de memòria cache del subsistema de memòria d'un computador que afavoreixin l'assimilació dels conceptes implicats.

Lectura prèvia

Considerem que el subsistema de memòria d'un computador consta d'una memòria principal (MP) i d'una memòria cache (MC). També considerem en aquest punt que la MP permet emmagatzemar tot l'espai adreçable pel processador mentre que la MC emmagatzema un subconjunt d'aquest espai adreçable. D'altra banda, sabem que el temps d'accés a MP és més elevat (més d'un ordre de magnitud) respecte al temps d'accés a MC. D'altra banda, també coneixem que el cost per byte de fabricar la MP és més baix que el cost de fabricar la MC. L'objectiu en incorporar el nivell de memòria cache en un computador és disminuir el temps mitjà d'accés a memòria.

A continuació es resumeixen els diferents aspectes arquitectònics que considerem en el disseny d'una memòria cache.

Mida de la MC. La MC s'organitza en blocs que contenen un cert nombre de paraules. El producte del nombre de blocs pel nombre de paraules per bloc estableix la capacitat de la MC quant a dades emmagatzemables de l'espai adreçable pel processador.

Algorisme d'emplaçament. Especifica en quina entrada de la MC s'ubica el bloc de MP que conté la paraula de memòria que accedeix el processador.

- **Correspondència directa:** A cada bloc de MP li correspon una entrada fixa de MC.
- **Completament associativa:** Cada bloc de MP pot anar a qualsevol entrada de MC.
- **Associativa per conjunts:** La MC s'organitza en conjunts, que contenen un cert nombre de blocs cada un. A cada bloc de MP li correspon un conjunt fix de MC, i el bloc pot anar a qualsevol de les entrades que té el conjunt.

Algorisme de reemplaçament. Especifica quin bloc s'ha de eliminar de la MC quan no hi ha lloc per col·locar-ne un de nou. Aquest algorisme només s'aplica en els emplaçaments totalment associatiu i associatiu per conjunts. En la correspondència directa el bloc a eliminar és únic i no hi ha alternativa. L'algorisme de reemplaçament que considerem és:

- **LRU:** S'elimina el bloc que fa més temps que no es referencia.

Política d'escriptura. Estableix com es gestionen les escriptures a memòria. D'una banda cal decidir si convé copiar a la MC el bloc de MP quan l'escriptura provoca una fallada i d'altra banda cal mantenir la coherència de les dues memòries (MC i MP). La combinació d'aquests dos aspectes ens dóna les següents alternatives:

- **Escriptura immediata amb assignació:** En cas de fallada en una escriptura es copia el bloc que s'accedeix a MC. Es realitza l'escriptura tant a MC com a MP. Aquesta política és l'única que es considera en el simulador MARS.
- **Escriptura immediata sense assignació:** En cas de fallada en una escriptura no es porta el bloc a MC i l'escriptura es realitza únicament a MP. Si l'escriptura genera un encert la paraula s'escriu tant a MC com a MP.
- **Escriptura retardada amb assignació:** En cas de fallada en una escriptura es porta el bloc a MC. L'escriptura es realitza únicament a MC. La MP s'actualitzarà quan el bloc on es realitza l'escriptura s'hagi de reemplaçar, en un moment posterior de l'execució del programa. Per a aquest efecte, s'utilitza el bit D (bit de bloc modificat).

Model de temps.

A continuació es descriu el model de temps que es considera per servir una referència a memòria, diferenciant si es tracta d'una lectura o d'una escriptura i si l'accés provoca un encert o una fallada a la MC. A més, el model contempla totes les polítiques d'escriptura estudiades.

A nivell general cal considerar que el temps de servei d'una referència a memòria és el temps en determinar si la referència és un encert o una fallada a memòria cache i servir la referència en cas d'encert (t_h) més el temps de penalització per resoldre la referència en accedir al següent nivell de la jerarquia de memòria (t_p):

$$t_{\text{accés}} = t_h + t_p$$

L'accés a les etiquetes per comprovar si la referència és un encert i el servei de la referència en cas d'encert es realitzen seqüencialment durant el temps t_h : en la primera meitat d'aquest temps es fa l'accés a les etiquetes i en la segona meitat es fa la lectura o escriptura a memòria cache.

Per aquelles configuracions de memòria cache amb política d'escriptura immediata es considera l'existència d'un buffer d'escriptura amb llargada ilimitada on queden

emmagatzemades les escriptures pendents de portar a MP. També es considera que cap referència a memòria entra en conflicte amb escriptures pendents en aquest buffer. Cal adonar-se que el contingut d'aquest buffer es porta a MP en paral·lel amb l'execució de les instruccions que vénen a continuació de l'accés a memòria.

A continuació es mostra una taula amb el temps de penalització (t_p) associat a una referència segons sigui lectura/escriptura; encert/fallada i segons la memòria cache contempli una política d'escriptura immediata sense assignació/retardada amb assignació:

t_p	Immediata amb assignació	Immediata sense assignació	Retardada amb assignació
Lectura - Encert	0	0	0
Lectura - Fallada	$t_{block} + t_h^1$	$t_{block} + t_h^1$	bloc modif.: $2 * t_{block} + t_h^2$
			bloc no mod.: $t_{block} + t_h^1$
Espectura - Encert	0 ³	0 ³	0 ⁵
Espectura - Fallada	$t_{block} + t_h^1$	0 ⁴	bloc modif.: $2 * t_{block} + t_h^2$
			bloc no mod.: $t_{block} + t_h^1$

¹Es porta el bloc de MP i es reinicia l'accés a memòria.

²Es porta el bloc modificat a MP; es porta el nou bloc de MP guardant-lo a l'entrada que es reemplaça i es reinicia l'accés a memòria.

³L'espectura es fa a MC i a MP. L'actualització de MP a partir del buffer d'espectura es fa de forma concurrent amb la continuació de la referència a memòria.

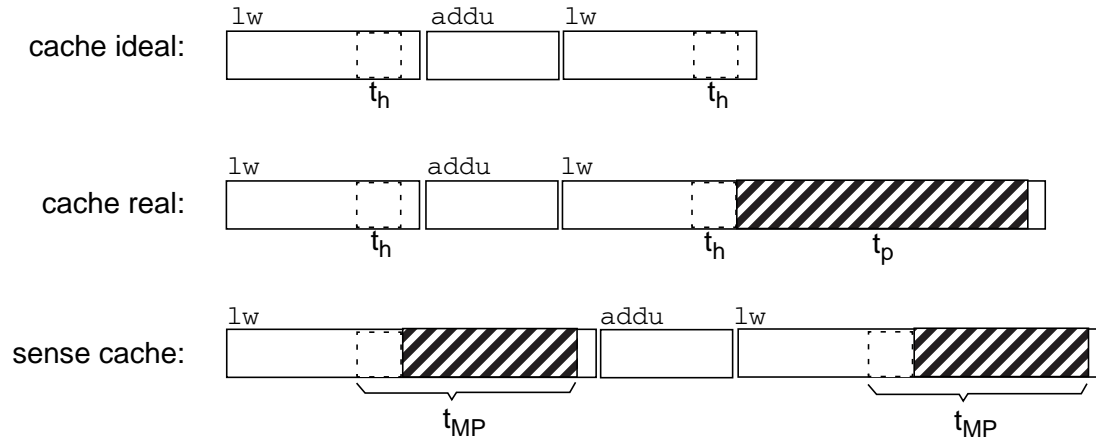
⁴L'espectura es fa únicament a MP. L'actualització de MP a partir del buffer d'espectura es fa de forma concurrent amb la continuació de la referència a memòria.

⁵L'espectura es fa únicament a MC.

Càlcul del temps d'execució en funció de t_p i CPI_{ideal}

Com ja sabem, el nombre de cicles d'execució es calcula $n_{cicles} = n_{ins} \cdot CPI$, essent n_{ins} el nombre d'instruccions executades i CPI el promig de cicles de rellotge que tarda cada instrucció. Sovint trobem problemes (com l'**exercici 5.5** d'aquesta pràctica) on es necessita calcular n_{cicles} per a un determinat programa sota una varietat de configuracions de memòria, on n_{ins} és constant però el CPI varia segons sigui el nombre de fallades de cache, ja que en cada fallada es tarden t_p cicles addicionals.

En els següents diagrames de temps s'executa un fragment de 3 instruccions `lw-addu-lw` sota tres supòsits: en el primer cas tenim una cache de dades ideal, on tots els accessos a memòria són encerts; en el segon cas tenim una cache real, on el primer accés és un encert però el segon sofreix una fallada de cache; i el tercer cas és un computador sense cache:



Cada rectangle simbolitza el temps d'execució d'una instrucció. Aquest temps inclou diverses activitats: llegir la instrucció de la cache d'instruccions, decodificar-la, llegir els operands-font del banc de registres, executar algun càlcul en la ALU, accedir a la cache de dades (sols en el cas dels `lw`), i finalment escriure el resultat en el banc de registres.

Considerem en primer lloc els dos primers diagrames. El primer diagrama ressalta amb línia de punts la fracció que correspon al temps d'accés a la cache de dades (t_h en els dos accessos, ja que són encerts). En el segon diagrama, el segon accés tarda un temps de penalització addicional t_p degut a la fallada (zona ratllada). Si anomenem $n_{\text{cicles_ideal}}$ el que tarda una execució ideal (parts en blanc) i $n_{\text{cicles_penal}}$ el total de cicles de penalització extra d'una execució real (parts ratllades), podem definir

$$\begin{aligned} \text{CPI}_{\text{ideal}} &= n_{\text{cicles_ideal}} / n_{\text{ins}} \\ \text{CPI}_{\text{real}} &= (n_{\text{cicles_ideal}} + n_{\text{cicles_penal}}) / n_{\text{ins}} \\ &= \text{CPI}_{\text{ideal}} + n_{\text{cicles_penal}} / n_{\text{ins}} \\ &= \text{CPI}_{\text{ideal}} + (n_{\text{fallades}} \cdot t_p) / n_{\text{ins}} \end{aligned}$$

I el temps d'execució (on t_c és el temps de cicle de rellotge) queda:

$$t_{\text{exe}} = t_c \cdot n_{\text{ins}} \cdot \text{CPI}_{\text{real}} = t_c \cdot (n_{\text{ins}} \cdot \text{CPI}_{\text{ideal}} + n_{\text{fallades}} \cdot t_p)$$

Aquest és justament el càlcul que demana l'**exercici 5.5** d'aquesta pràctica. Els paràmetres t_c i $\text{CPI}_{\text{ideal}}$ els dona l'enunciat i les variables n_{ins} i n_{fallades} es poden obtenir fàcilment amb el

simulador (Tools -> Instruction Statistics i Tools -> Data Cache Simulator). L'enunciat també dóna el paràmetre t_{block} , que ens permet calcular t_p . Efectivament, tenint en compte que la cache és de correspondència directa, i seguint les expressions de la taula de l'apartat anterior ("Model de Temps"), resulta: $t_p = t_{\text{block}} + t_h$.

En segon lloc, l'exercici 5.5 demana també calcular el temps d'execució en cas de no tenir cache de dades, on tots els accessos a memòria principal de les instruccions lw tarden un temps t_{MP} . Aquest és el cas il·lustrat al tercer diagrama. Per calcular el temps d'execució del cas sense cache podem sumar el temps d'execució del cas ideal més el temps extra del cas sense cache (porció ratllada). Cal observar que en cada load, aquest temps extra no és pas tot el temps d'accés t_{MP} sinó només la diferència entre el del cas sense cache i el del cas ideal, és a dir $t_{MP} - t_h$. Vist d'una altra manera, si considerem que el cas sense cache es comporta a la pràctica com el d'una cache "que sempre falla", llavors podrem considerar aquest temps extra com el "temps de penalització de fallada", i podrem fer servir la mateixa fórmula aplicada en el cas de la cache real. Sols cal considerar que

- n_{fallades} és igual al nombre total de loads
- el temps de penalització és $t_p = t_{MP} - t_h$

Enunciats de la sessió

En la realització d'aquesta sessió de laboratori considerarem un computador que disposa d'un processador de 32 bits, com és ara el MIPS, i que el seu sistema de memòria, adreçable a nivell de byte, té una memòria cache que gestiona de forma independent les instruccions i les dades. Només analitzarem les referències a paraules dins la memòria cache de dades. Aquesta memòria cache serà parametrizable pel que fa referència a l'algorisme d'emplaçament: correspondència directa, completament associatiu i associatiu per conjunts i pel que fa referència a la seva mida: nombre de blocs que conté i nombre de paraules per bloc (compte que no podem indicar el nombre de bytes per bloc). També cal considerar de forma general per totes les activitats d'aquesta sessió que les variables globals que contenen els programes analitzats estan sempre emmagatzemades a memòria a partir de l'adreça 0x10010000.

Activitat 5.A: Lectures a una cache de correspondència directa

En aquesta primera activitat analitzarem el comportament de la memòria cache amb correspondència directa en servir una seqüència de referències, que seran únicament de lectura.

En primer lloc identificarem els camps en què es descomponen les adreces de memòria, els quals intervenen en la gestió que s'ha de realitzar a la memòria cache.

Exercici 5.1: Considera que la memòria cache conté 32 blocs i cada bloc té 4 paraules. Si el processador genera una referència a memòria consistent en la lectura de la paraula que té com adreça 0x1001014C, contesta les següents preguntes:

- 1) Quin és el número (en hexadecimal) del bloc de memòria principal al qual pertany aquesta referència?

- 2) Quin és l'índex (en hexadecimal) del bloc de memòria cache que correspon a aquesta referència?

- 3) Quina és l'etiqueta (en hexadecimal) que correspon a aquesta referència?

Analitzarem com es comporta la memòria cache en el recorregut seqüencial d'un vector. El programa que fa aquest recorregut es mostra a la figura 5.1. Examina'l i contesta l'exercici 5.2 abans de continuar:

```
int vector[100];
```

```
main()
```

```
{
```

```
int i, sum=0;
```

```
    for (i=0; i<100; i++)
```

```
        sum += vector[i];
```

```
}
```

```
move    $t1, $zero    # sum=0
```

```
la      $t2, vector
```

```
move    $t0, $zero    # i=0
```

```
li      $t3, 100
```

```
for:
```

```
bge     $t0, $t3, fi
```

```
lw      $t4, 0($t2)
```

```
addu    $t1, $t1, $t4
```

```
addiu   $t2, $t2, 4
```

```
addiu   $t0, $t0, 1
```

```
b       for
```

```
fi:
```

Figura 5.1: Programa que fa el recorregut seqüencial d'un vector, en alt nivell i en assemblador MIPS.

Exercici 5.2: Considera la mateixa memòria cache de l'exercici 5.1, és a dir, de correspondència directa i que conté 32 blocs de 4 paraules cada un, i contesta les següents preguntes:

- 1) Indica l'adreça (en hexadecimal) de l'element del vector que s'accedeix en cada una de les 10 primeres iteracions del bucle. A més, per cada una d'aquestes referències indica el número de bloc de MC que s'accedeix i si es produeix un encert o una fallada.

Iteració	Adreça	Bloc MC	Encert / Fallada
1	0x10010000	0x00	MISS
2	0x10010004	0x00	HIT
3	0x10010008	0x00	HIT
4	0x1001000C	0x00	HIT
5	0x10010010	0x01	MISS
6	0x10010014	0x01	HIT
7	0x10010018	0x01	HIT
8	0x1001001C	0x01	HIT
9	0x10010020	0x02	MISS
10	0x10010024	0x02	HIT

- 2) Quina és la taxa d'encerts que s'obté en l'execució completa d'aquest programa?

Obté una taxa d'encerts del 75%.

Per **verificar** l'exercici 5.2, i d'aquí en endavant per analitzar el funcionament de la memòria cache, farem servir el mòdul "Data Cache Simulator" del MARS, que es troba a l'opció "Tools" del seu menú. Carrega aquest mòdul i observa que s'obre una nova finestra. En aquesta finestra es poden canviar els paràmetres de disseny a la part anomenada "Cache Organization". A la part anomenada "Cache Performance" obtindrem dades estadístiques del comportament de la memòria cache en l'execució d'un programa: les referències a memòria que es produeixen durant la seva execució, el nombre d'encerts, el nombre de fallades, i la taxa d'encerts. Inicialment, cal prémer el botó "Connect to MIPS" per poder usar aquest mòdul.

El fitxer **s5a.s** conté el programa de la figura 5.1. Carrega aquest programa al MARS, assembla'l i posa en marxa el mòdul "Data Cache Simulator". Executa completament el programa i verifica si la taxa d'encerts que has contestat a l'exercici 5.2 és la que indica el simulador.

Completa l'exercici 5.3 abans de continuar:

Exercici 5.3: Contesta les següents preguntes:

- 1) Si dupliquem la mida de la cache en base a duplicar el nombre de blocs que conté, hi haurà alguna variació en la taxa d'encerts obtinguda? Per què?

No, ja que el programa accedirà als 32 blocs disponibles originalment.

- 2) Si dupliquem la mida de la cache en base a duplicar el nombre de paraules per bloc, hi haurà alguna variació en la taxa d'encerts obtinguda? Per què?.

Augmentarà perquè els errors provenen dels canvis de bloc i ara hi haurà menys canvis.

Verifica la respostes de l'exercici 5.3 canviant la geometria de la cache en el mòdul "Data Cache Simulator" i executant novament de forma completa el programa. Tingues en compte d'inicialitzar la cache, prement el botó "Reset", cada cop que executis el programa.

L'accés als elements del vector també es pot efectuar en ordre contrari, atès que el que volem obtenir és la suma de tots els elements. Contesta l'exercici 5.4 abans de continuar:

Exercici 5.4: La taxa d'encerts que s'obtingria en l'execució del programa si el recorregut es fa des del darrer element del vector fins al primer és la mateixa o diferent respecte el recorregut en ordre creixent dels elements?

Hi haurà la mateixa taxa d'encerts.

Modifica el programa del fitxer **s5a.s** invertint l'ordre del recorregut i guarda'l amb el nom **s5a2.s**. **Verifica** amb el simulador la taxa d'encerts de l'exercici 5.4.

A continuació analitzarem l'eficiència de la cache comparant un processador amb i sense cache de dades. Completa el següent exercici:

Exercici 5.5: Suposem un sistema computador amb cache d'instruccions i dades separades. La d'instruccions és una cache ideal. La de dades és de correspondència directa amb 32 blocs i blocs de 4 paraules, i té el següent model de temps simplificat:

- Temps de cicle: $t_c = 10 \text{ ns}$.
- En cas d'encert de cache, temps d'accés a memòria : $t_h = 1 \text{ cicle}$
- En cas de fallada de cache el temps d'accés sofreix una penalització t_p que es calcula sabent que el temps per copiar un bloc de MP a cache és: $t_{\text{block}} = 10 \text{ cicles}$

Nota: Per calcular t_p , llegeix amb atenció l'apartat de la Lectura Prèvia "Model de temps"

a) Calcula el nombre d'instruccions executades pel programa de la figura 5.1

Nota: Tingues en compte que algunes línies d'assemblador són macros que s'expandeixen a més d'una instrucció; que el test del salt del bucle s'executa una vegada més que la resta d'instruccions; i que s'executen 3 instruccions del fitxer *startup.s*, 1 al principi del programa i 2 més al final del programa. Alternativament, pots usar les eines "Tools -> Instruction Statistics" o "Tools -> Instruction Counter" del simulador MARS.

$$\bullet \quad n_{\text{ins}} = 711$$

b) Calcula el temps d'execució del programa de la figura 5.1 suposant que hem mesurat que $\text{CPI}_{\text{ideal}} = 2$ (és el CPI si tots els accessos a cache són encerts).

Nota: Llegeix amb atenció el darrer apartat de la Lectura Prèvia "Càlcul del temps d'execució en funció de t_p i $\text{CPI}_{\text{ideal}}$ "

$$\bullet \quad t_{\text{exe}} \text{ (en ns.)} = 16970 \text{ ns } (10 \cdot (711 \cdot 2 + 25 \cdot (10 + 1)))$$

c) Calcula el temps d'execució del programa de la figura 5.1 suposant ara que el computador no té cache de dades (en aquest cas, el temps d'accés a memòria principal és $t_{\text{MP}} = 6 \text{ cicles}$). Calcula també el guany de rendiment que proporciona tenir aquesta cache de dades respecte de no tenir-la.

Nota: Llegeix amb atenció el darrer apartat de la Lectura Prèvia "Càlcul del temps d'execució en funció de t_p i $\text{CPI}_{\text{ideal}}$ ":

$$\bullet \quad t_{\text{exe}} \text{ (en ns.)} = 19220 \text{ ns } (10 \cdot (711 \cdot 2 + 100 \cdot (6 - 1)))$$

$$\bullet \quad \text{Guany} = 1,1326$$

Activitat 5.B: Escriitures en una cache de correspondència directa

Introduïrem ara les escriptures en les referències que genera el processador. Considerant el programa de la figura 5.2, que realitza l'ordenació dels elements d'un vector, completa l'exercici 5.6 abans de continuar.

```

int vec[5] = {20, -18, 45, 12, -1};
main() {
    int i, j, aux, max;
    for (i=0; i<4; i++) {
        max = i;
        for (j=i+1; j<5; j++)
            if (vec[j] > vec[max]) max = j;
        aux = vec[i];
        vec[i] = vec[max];
        vec[max] = aux;
    }
}

```

Figura 5.2: Programa d'ordenació dels elements d'un vector

Exercici 5.6: Considerant que la memòria cache aplica una política d'escriptura immediata amb assignació, que té 32 blocs, i que els blocs són de 4 paraules, digues:

- 1) Per a cada una de les 4 iteracions del bucle exterior omple la taula amb la següent informació: (a) una llista dels elements $vec[x]$ que s'accedeixen (anota sols l'índex); (b) per cada element, si és una referència de lectura (R) o d'escriptura (W) i si genera un encert (h) o a una fallada (m) a la cache.

Iteració i=0			Iteració i=1			Iteració i=2			Iteració i=3		
vec[]	R/W	h/m	vec[]	R/W	h/m	vec[]	R/W	h/m	vec[]	R/W	h/m
1	R	m	2	R	h	3	R	h	4	R	h
0	R	h	1	R	h	2	R	h	3	R	h
2	R	h	3	R	h	4	R	h	3	R	h
0	R	h	2	R	h	3	R	h	4	R	h
3	R	h	4	R	h	2	R	h	3	W	h
2	R	h	2	R	h	3	R	h	4	W	h
4	R	m	1	R	h	2	W	h			
2	R	h	2	R	h	3	W	h			
0	R	h	1	W	h						
2	R	h	2	W	h						
0	W	h									
2	W	h									

- 2) Compta el nombre total d'accessos a memòria i d'encerts que es realitzen en cada iteració del bucle extern.

Iteració i=0		Iteració i=1		Iteració i=2		Iteració i=3	
accessos	encerts	accessos	encerts	accessos	encerts	accessos	encerts
12	10	10	10	8	8	6	6

- 3) Quina és la taxa d'encerts que s'obté en l'execució del programa?

S'obté una taxa del 94,44%. (34/36)

El fitxer **s5b.s** conté la traducció del programa de la figura 5.2. **Verifica** la taxa d'encert calculada a l'exercici anterior carregant al simulador aquest fitxer, assemblant-lo, configurant correctament el mòdul "Data Cache Simulator", i executant el programa.

Completa el següent exercici abans de continuar:

Exercici 5.7: Indica com canviaria la taxa d'encerts si haguéssim considerat una política d'escriptura *immediata sense assignació*. Ídem per al cas d'una política d'escriptura *retardada amb assignació*? Raona les respostes.

Per immediata sense assignació no canviaria perquè després de les dues primeres fallades de lectura, de la mateixa manera que amb escriptura immediata amb assignació, tindríem disponibles a cache els únics dos blocs que necessitem al llarg de l'execució.

En el cas d'escriptura retardada amb assignació passarà el mateix.

Un cop vist el comportament de la memòria cache durant l'execució del programa, volem abstraure aquest comportament per entendre com es comportarà amb vectors més grans. Completa el següent exercici abans de continuar:

Exercici 5.8: Suposem que el vector a ordenar tingues 64 elements:

- 1) Considerant la mateixa geometria de la cache de l'exercici 5.6, quantes fallades es produirien?

Se'n produirien 16, una cada 4 elements (bloc) ja que no excedim la capacitat de la memòria cache.

- 2) Suposem que volem reduir al màxim la capacitat de la memòria cache però sense que augmenti el nombre de fallades del programa. Raona fins a quin nombre mínim de blocs la podem reduir.

La podem reduir fins a 16 blocs. Cada bloc conté 4 paraules, 64 ($16 * 4$) elements del vector. Si excedim aquest nombre, haurem de començar a sobreescriure espais de la memòria cache.

Modifica el programa del fitxer s5b.s (copiant-lo amb el nom **s5b2.s**) perquè el vector tingui 64 elements: canvia la declaració (per a aquest experiment no importa que no s'inicialitzin els elements del vector), i també el nombre d'iteracions dels dos bucles.

Verifica amb el simulador que les respostes donades a l'exercici 5.8 siguin correctes.

Activitat 5.C: Accés a múltiples dades estructurades

Analitzarem ara com es comporta la memòria cache en l'execució d'un programa que accedeix a múltiples estructures de dades, on es posa de manifest l'existència de conflictes per accedir als mateixos blocs de la cache. Considereu el programa de la figura 5.3, que realitza la suma de dos vectors i deixa el resultat en un tercer vector.

```
int A[128], B[128], C[128];
main() {
    int i;

    for (i=0; i<128; i++)
        C[i] = A[i] + B[i];
}
```

Figura 5.3: Programa que realitza una suma de vectors.

Completa el següent exercici abans de continuar:

Exercici 5.9: Considerant que la memòria cache, de correspondència directa, aplica una política d'escriptura immediata amb assignació, que té 16 blocs i que els blocs són de 4 paraules, indica quina és la taxa d'encerts en l'execució del programa de la figura 5.3.

La taxa d'encerts és del 0%. (0/384)

El fitxer **s5c.s** conté la traducció del programa de la figura 5.3 a assembleador MIPS. **Verifica** amb el simulador la resposta donada en l'exercici anterior.

Completa el següent exercici abans de continuar:

Exercici 5.10: Aquest exercici pretén que pensis una manera de millorar la taxa d'encerts de l'anterior programa fent petites modificacions a la declaració de dades: essent conscient de la geometria que té la cache (16 blocs de 4 paraules cadascun), omple les 2 caselles que hi ha a continuació, que formen part de la declaració dels vectors del programa de la figura 5.3, per tal que la taxa d'encerts sigui 0,75.

```
.data
A:      .space 512
        .space 16
B:      .space 512
        .space 16
C:      .space 512
```

Copia el fitxer s5c.s en un nou fitxer **s5c2.s**, i escriu-hi les modificacions que has proposat a l'exercici anterior. **Verifica** amb el simulador si s'obté la taxa d'encerts de 0,75.

Ara analitzarem quin seria el rendiment del programa de la figura 5.3 en una memòria cache associativa. En primer lloc ho farem per a una cache completament associativa. Completa el següent exercici:

Exercici 5.11: Considerant una memòria cache completament associativa (de 16 blocs de 4 paraules cadascun), calcula quina és la taxa d'encerts que obtindrem en l'execució del programa de la figura 5.3.

La taxa d'encerts és del 75%. (3/4)

Verifica-ho amb el simulador.

A continuació analitzarem el programa en una memòria cache associativa per conjunts. Completa el següent exercici:

Exercici 5.12: Considerant una memòria cache amb blocs de 4 paraules, associativa per conjunts amb 16 conjunts, raona quin ha de ser el grau mínim de l'associativitat (nombre de blocs per conjunt) perquè la taxa d'encerts del programa de la figura 5.3 sigui diferent de 0 (recorda que l'associativitat no és necessàriament potència de 2).

El grau mínim hauria de ser 3, un per vector.

Al simulador pots especificar el nombre de blocs que té cada conjunt (el grau de l'associativitat) al camp "Set size". Malauradament, el simulador només tracta graus d'associativitat que són potències de 2. **Verifica** la resposta amb el simulador, ajustant l'associativitat al valor que s'aproximi més a la teva resposta.

Activitat 5.D: Optimització de codi (opcional)

En aquesta darrera activitat l'objectiu és arribar a optimitzar l'execució d'un programa tenint en compte la memòria cache de què disposa el sistema. Considereu el programa de la figura 5.4, que realitza el producte d'una matriu per un vector:

```
int V1[16], M[16][16], V2[16];
main() {
    int i,j,tmp;

    for (i=0; i<16; i++) {
        tmp = 0;
        for (j=0; j<16; j++)
            tmp += M[i][j] * V2[j];
        V1[i] = tmp;
    }
}
```

Figura 5.4: Programa que realitza el producte d'una matriu per un vector.

El fitxer **s5d.s** conté aquest programa traduït a assembleador MIPS.

Determina mitjançant el simulador MARS quina és la taxa d'encerts que s'obté executant el programa de la figura 5.4. Considera que els blocs de la memòria cache són de 4 paraules. Calcula aquesta taxa per a diferents mides de la cache completant la següent taula (dins de cada fila suposem que la mida total no canvia, sols l'associativitat):

Capacitat de la cache	Correspondència directa	Associativa de grau 2	Associativa de grau 4	Associativa de grau 8	Completament Associativa
8 blocs	43%	81%	78%	73%	73%
16 blocs	65%	86%	86%	86%	86%
32 blocs	76%	86%	86%	86%	86%
64 blocs	82%	86%	86%	86%	86%
128 blocs	86%	86%	86%	86%	86%

L'optimització que volem aplicar està basada en aprofitar la localitat temporal de les referències al vector V2. Fixa't que aquest vector es llegeix tot sencer, una vegada per cada fila que tractem de la matriu M. El que pretenem és que els accessos al vector corresponguin únicament a aquells elements que caben en un bloc de la cache que, en el cas que considerem, són 4. Així, en lloc de multiplicar tota una fila de la matriu M per tot el vector V2 volem fer el tractament de 4 en 4 elements¹: és a dir, que dividirem la matriu en blocs de 4 columnes i farem primer totes les operacions involucrades amb els elements del primer bloc (iterant totes les seves files) abans de passar al bloc següent. Completa el següent exercici abans de continuar:

Exercici 5.13: El codi que hi ha a continuació correspon a una versió optimitzada del programa de la figura 5.4, on s'hi ha aplicat l'optimització anomenada "tiling". L'índex k itera els 4 blocs de la matriu (amb 4 columnes per bloc). Completa les caselles buides amb les sentències d'alt nivell que contenen els accessos a les estructures de dades.

```
int V1[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int M[16][16], V2[16];

main()
{
    int i,j,k,tmp;

    for (k=0; k<4; k++)
        for (i=0; i<16; i++)
        {
            tmp = 0;
            for (j=0; j<4; j++)
            {
                tmp += M[i][k*4+j] * V2[k*4+j] ;
            }
            V1[i] += tmp;
        }
}
```

1. Aquesta optimització rep el nom de "loop blocking" o "tiling", que consisteix en transformar un bucle en dos bucles de forma que l'espai d'iteracions de l'original (bucle j) passa a tenir menys iteracions (de 16 passa a 4) per aprofitar la localitat temporal de les estructures que s'hi accedeixen (en aquest cas, la localitat temporal del vector V2)

Completa el següent exercici abans de continuar:

Exercici 5.14: Tradueix a ensamblador MIPS el programa de l'exercici 5.13.

```

.data
V1: .space 64
M: .word 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
   .word 0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
   .word 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
   .word 0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
   .word 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
   .word 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
   .word 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1
V2: .word -5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10

.text
.globl main
main:
    move $t0, $zero
    li $t1, 4

fork: bge $t0, $t1, ffork
      li $t2, 16
      move $t3, $zero

fori:  bge $t3, $t2, ffori
      move $t5, $zero
      move $t6, $zero

forj:  bge $t6, $t1, fforj
      sll $t7, $t0, 2
      addu $t7, $t6, $t7
      la $s0, M
      la $s1, V2
      sll $s2, $t3, 4
      addu $s2, $s2, $t7
      sll $s2, $s2, 2
      addu $s0, $s0, $s2
      lw $s0, 0($s0)
      sll $s3, $t7, 2
      addu $s1, $s3, $s1
      lw $s1, 0($s1)
      mult $s0, $s1
      mflo $s0
      addu $t5, $t5, $s0

      addiu $t6, $t6, 1
      b forj

fforj: la $s4, V1
      sll $s5, $t3, 2
      addu $s4, $s5, $s4
      lw $s6, 0($s4)
      addu $s6, $s6, $t5
      sw $s6, 0($s4)
      addiu $t3, $t3, 1
      b fori

ffori: addiu $t0, $t0, 1
      b fork

ffork: jr $ra

```

Escriu el programa de l'exercici anterior en un fitxer nou, i anomena'l **s5d2.s**. Verifica amb el simulador que és correcte, comprovant que el resultat que s'obté al vector V1 coincideix amb l'obtingut al programa original del fitxer s5d.s.

Determina amb el simulador la taxa d'encerts que s'obté executant el programa optimitzat:

Capacitat de la cache	Correspondència directa	Associativa de grau 2	Associativa de grau 4	Associativa de grau 8	Completament Associativa
8 blocs	52%	84%	87%	87%	87%
16 blocs	70%	88%	88%	88%	87%
32 blocs	79%	88%	88%	88%	89%
64 blocs	84%	89%	89%	89%	89%
128 blocs	89%	89%	89%	89%	89%

Quina és la diferència entre la millor taxa d'encerts d'aquesta versió i la de la versió no optimitzada?

La diferència és d'un 3%