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 parametritzable 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 seqüencia 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?

   0x1001014

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

   0x14

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

   0x080080

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	0x1001000	Miss
2	0x10010004	0x1001000	Hit
3	0x10010008	0x1001000	Hit
4	0x1001000C	0x1001000	Hit
5	0x10010010	0x1001001	Miss
6	0x1001014	0x100101	Hit
7	0x10010018	0x100101	Hit
8	0x1001001C	0x100101	Hit
9	0x10010020	0x100102	Miss
10	0x10010024	0x100102	Hit

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

hit ratio = num_encerts/ num_ref = 0,75

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?

El numero de blocs marca la magnitud de l'index de cada bloc de MP, abans teniem 32 (2^5), per tant l'index tenia 5 bits, ara tindra 6 per tant sera menys probable el reemplaçar un bloc, per tant augmenta la taxa d'encerts.

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?.

El nombre de paraules en cada bloc marca l'offset, casa paraula es una cel·la de 4 bytes, si ho dupliquem serán 8 paraules, 32 bytes (2^5), 5 bits d'offset, per tant hi hauran més dades concentrades en un mateix bloc, això augmentarà la taxa d'encerts fins a un cert un punt, hi haura un moment on tindrás que treure linies per seguir amb la mateixa memoria.

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. Programa COMPLET

Actual = 75%, 64 blocs = 75%, 8 paraules = 87%

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'obtindria 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?

La mateixa

Modifica el programa del fitxer s5a.s invertint l'ordre del recorregut i guarda'l amb el nom **s5a2.s**. V**erifica** amb el simulador la taxa d'encerts de l'exercici 5.4. 74%

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 dé 4 paraules, i té el següent model de temps simplificat:

- Temps de cicle: t_c = 10 ns.
- En cas d'encert de cache, temps d'accés a memòria : t_h = 1cicle
- 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_{block} = 10 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.

- n_{ins} = 711

b) Calcula el temps d'execució del programa de la figura 5.1 suposant que hem mesurat que CPI_{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 CPI_{ideal}"

- t_{exe} (en ns.) = 10*(711*2 + 25*110) = 41720 ns

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_{MP}=6 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 CPI_{ideal}":

- t_{exe} (en ns.) = 10*(711*2 + 100*50) = 64220
- Guany = 64220/41720 = 1,54

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

Introduirem 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	2	10	10	8	8	6	6

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

hit_ratio = 34/36 = 0,9444%

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. 0,94%

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.

L'escriptura immediata sense assignació implica que hem de escriure a la MP en cas de miss, donat que no han hagut fallades en l'escriptura no canviara la taxa d'encerts.

En el cas de la retardada amb assignació la taxa d'encerts tampoc variarà, no hi ha ninguna fallada d'escriptura.

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 tingués 64 elements:

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

   64/4 = 16

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.

   16 blocs, cada bloc són 4 words per tant tindrem els 64 elements ja a MC al acabar la primera iteració i, fent que a les següents iteracions no produeixin més fallades.

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. Miss count = 16
16 blocs surt el mateix
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.

hit ratio = num_encerts / num_ref = 0/384 = 0

El fitxer **s5c.s** conté la traducció del programa de la figura 5.3 a assemblador MIPS. **Verifica** amb el simulador la resposta donada en l'exercici anterior. 0

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 80

```
B:      .space 512
```

.space 80

```
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.

hit ratio = num_encerts / num_ref = 288/384 = 0,75%

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).

3, un bloc de cada conjunt per cada 4 paraules de cada vector (son 3) així no es reemplaçarant mutuament

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 assemblador 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 elements[1]: 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 ][ j + 4*k ] * V2[ j  + 4*k] ];
            }
            [ 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 assemblador MIPS el programa de l'exercici 5.13.

```
main:
      move   $t0, $zero   # $t0  = k
      li     $t1, 4
fork:
 bge  $t0, $t1, fi_fork
 move  $t3, $zero   # $t3 = i
 li  $t2, 16
fori:
 bge  $t3, $t2, fi_fori
 move  $t5 $zero   # $t5 = temp
 move  $t6, $zero   # $t6 = j
forj:
 bge $t6, $t1, fi_forj
 sll  $t7, $t0, 2   # [4*k + j]
 addu  $t7, $t6, $t7

 la  $s0, M
 sll  $s2, $t3, 4   # [i]
 addu  $s2, $s2, $t7   # [i][4*k + j]
 sll  $s2, $s2, 2
 addu  $s0, $s0, $s2   # M[i][4*k + j]
 lw  $s0, 0($s0)

 la  $s1, V2
 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
fi_forj:
 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
fi_fori:
 addiu  $t0, $t0, 1
 b  fork
fi_fork:
```

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	80%	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?

3%