# Pràctica 5

## Preliminars

4 bits per fer un dígit d’hexadecimal. (0x10010000 són 8 dígits\*4b=32 bits=4 Bytes)

**Paraula** = Unitat estàndard de dades que utilitza el processador (Com que CPU és de 32 b 🡪Paraula 4B).

**Bloc** = Unitat mínima de dades transferides entre MC i MP. En aquest cas cada bloc conté 4 paraules. Si cada paraula són 4B 🡪 Mida\_Bloc = 16B

**Offset** = Desplaçament intern dins dels blocs de la cache. En aquest cas cada bloc conté 4 paraules. Ha de permetre especificar qualsevol byte dins dels 16 B (donat que podem fer un lb)

## Activitat 5.A

### Exercici 5.1

Tenim 32=2^5 blocs. Això significa que necessitarem accedir des de l’índex 0-31 llavors haurem d’agafar 5 bits.

Tenim Mida\_Bloc = 16=2^4 bytes llavors necessitem 4 bits per a poder identificar qualsevol byte dins del bloc.

| Número del bloc de MP 🡪 0x1001014C>>4 = 0x0**1001014** |
| --- |
| Índex 🡪 (0x1001014C>>4)%5 = 0x14 |
| Etiqueta 🡪0x1001014C>>(4+5) = 0x000**80080** |

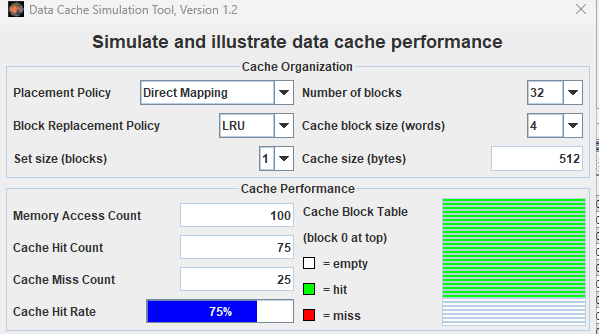
NOTA: | etiq (23 b) | línia (5 b) | offset (4 b) | llavors en fer mod5 “agafem literalment” els bits “línia”. Primer hem de fer el desplaçament perquè si no agafaríem els de offset també.

### Exercici 5.2

Recorda que cada bloc hi ha 4 words.

| **ITERACIÓ** | **ADREÇA** | **BLOC MC** | **H/M** |
| --- | --- | --- | --- |
| 1 | 0x10010000 | 0 | M |
| 2 | 0x10010004 | 0 | H |
| 3 | 0x10010008 | 0 | H |
| 4 | 0x1001000C | 0 | H |
| 5 | 0x10010010 | 1 | M |
| 6 | 0x10010014 | 1 | H |
| 7 | 0x10010018 | 1 | H |
| 8 | 0x1001001C | 1 | H |
| 9 | 0x10010020 | 2 | M |
| 10 | 0x10010024 | 2 | H |

| **Hits** | **Misses** | **Total** |
| --- | --- | --- |
| (100/4)\*3 = 75 | 100/4 = 25 | 75 % |



### Exercici 5.3

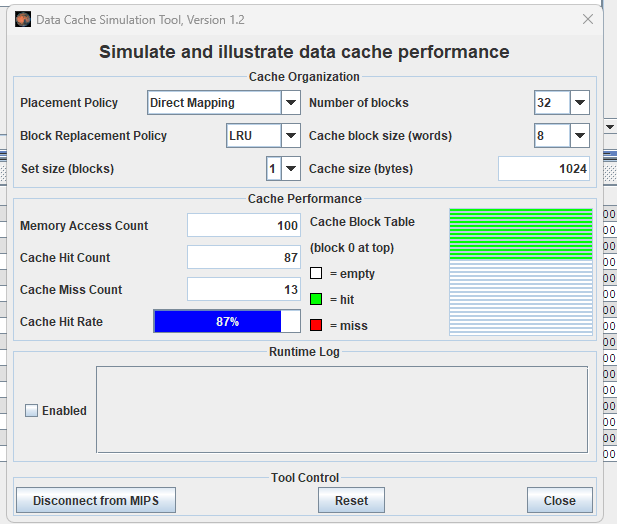
1. **Si dupliquem la mida de la cache en base a duplicar el nombre de blocs que conté, hi haurà alguna variació en la taxa d’encerts obtinguda? Per què?**

No perquè el problema no és el nombre de blocs sinó la quantitat de words per bloc. Realment només estem fent servir 25 blocs dels 32 que té.

1. **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?**

Sí que hi haurà variació perquè ara en poder contenir el doble de words per bloc, tindrem la meitat de misses donat que el miss que hi ha a la iteració 4 l’estalviarem però el miss de la iteració 8 ja no.

En aquest cas farem servir només 13 blocs així que hi haurà 13 misses (100/8)=12’6 🡪 13.



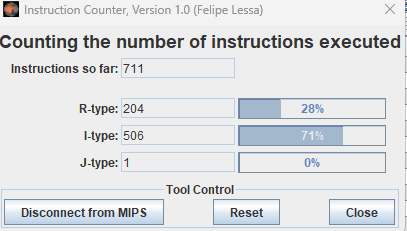
### Exercici 5.4

1. **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?**

No hi haurà cap variació perquè la divisió és en blocs és la mateixa (És a dir, la quantitat de peticions de lectura a la MP és la mateixa.

### Exercici 5.5

1. **Nombre d’instruccions.**



n\_ins = 711

1. **Temps execució amb CPI\_ideal = 2.**

t\_p = t\_block + t\_h

t\_exe = (n\_ins \* CPI\_ideal + n\_fallades \* t\_p) \* tc

t\_exe = ((711\*2)+(25\*(10+1))\*10e-9 = 16,97e-6 seg = 16970 ns

1. **Temps execució i guany si no PC no té dades i t\_MP = 6 cicles.**

tp = t\_MP – t\_h

t\_exe = (n\_ins \* CPI\_ideal + n\_fallades \* t\_p) \* tc

t\_exe = ((711\*2)+(100\*(6-1))\*10e-9 = 19,22e-6 seg = 19220 ns

guany = t\_exe\_new / t\_exe\_old = 1,13

## Activitat 5.B

### Exercici 5.6

1. **Escriptura immediata amb assignació. 32 Blocs. 4 paraules per bloc.**

Recorda que copia del 0-3 per cada miss.

|  | i=0 |  |  | i=1 |  |  | i=2 |  |  | 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 |  |  |  |  |  |  |  |  |  |

| accessos | hits | accessos | hits | accessos | hits | accessos | hits | **TOTAL** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12 | 10 | 10 | 10 | 8 | 8 | 6 | 6 | (10+10+8+6)/(12+10+8+6) |

### Exercici 5.7

1. **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.**

Escriptura immediata sense assignació:

No canviarà perquè els miss que hi ha amb la política anterior són de lectura fent que es copiï el bloc a la cache. Llavors posteriorment, en el codi, sí que hi ha modificacions, però aquestes es faran directament a la MC donat que ja ha estat el bloc carregat (No a la MP com diu la política si hagués sigut miss).

NOTA: Si hagués hagut un miss primer d’escriptura, posteriorment hi hauria un altre miss per fer lectura donat que aquesta política no copia el bloc a la MC sí és miss en W.

Escriptura retardada amb assignació:

En aquest cas succeeix el mateix. En la primera iteració del programa hi hauria els mateixos dos miss, però després funcionaria com pertoca a aquesta política i no afectaria la quantitat d’encerts o fallades.

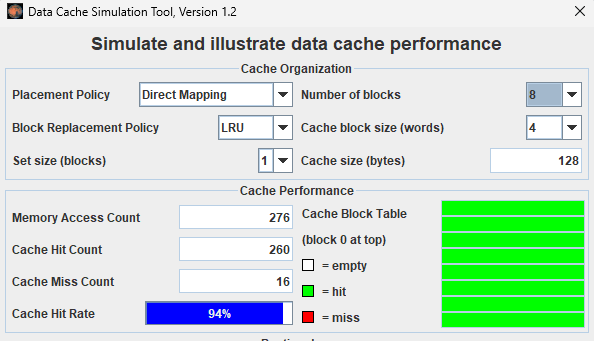
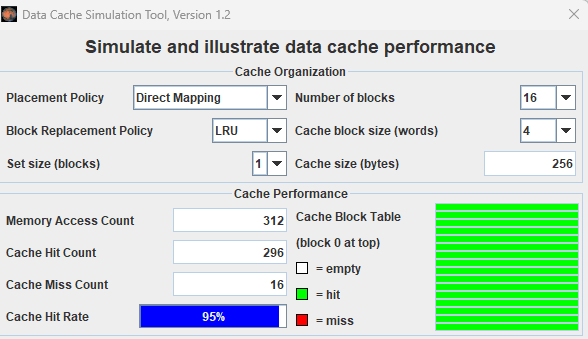
### Exercici 5.8

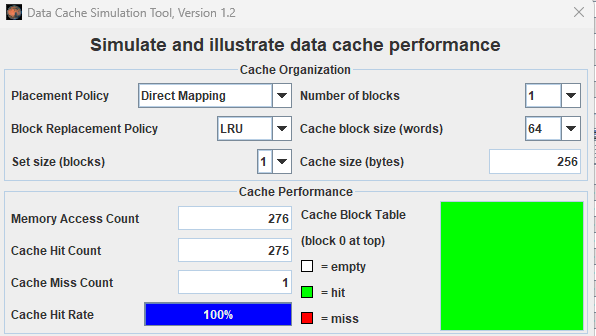
1. **Considerant la mateixa geometria de la cache de l’exercici 5.6, quantes fallades es produirien?**

Donat que només es poden guardar 4 words per bloc, si hi hagués 64 elements necessitaríem 64/4=16 blocs i ja en tenim suficient.

1. **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.**

Reduir la taxa de fallades va depenent de la mida del bloc. Si sabem que sempre tindrem 64 elements, amb 1 sol bloc amb la capacitat de 16 words només hi hauria 1 miss. Si no podem modificar la quantitat de words per bloc, llavors amb 16 blocs ja és suficient perquè no hi hagi cap reemplaçament.





## Activitat 5.C

### Exercici 5.9

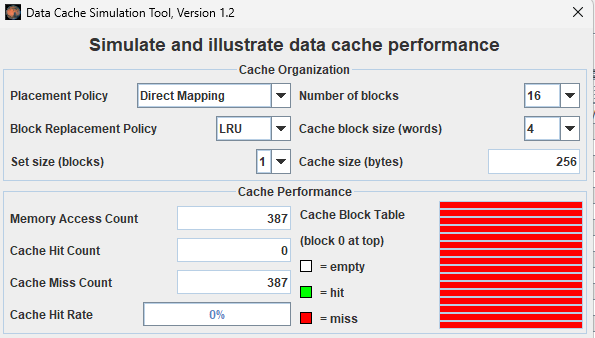
1. **Considerant que la memòria cache, de correspondència directa, aplica una política d’escriptura immediata amb assignació, que té 16 blocs i que els blocs són de 4 paraules, indica quina és la taxa d’encerts en l’execució del programa de la figura 5.3.**

A[128] ocupa des de 0x10010000 – 0x100101FC

B[128] ocupa des de 0x10010200 – 0x100103FC

C[128] ocupa des de 0x10010400 – 0x100105FC

El que succeeix és que A[0] -> 0x10010000 i el número de bloc (índex) corresponent és el 0x10010000>>4 = 0 i el mateix és per a B[0] i C[0] llavors s’estan tota l’estona sobreescrivint fent que mai hi hagi un hit. Llavors la taxa d’encerts és del 0%.



### Exercici 5.10

1. **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.**

| 1. .data 2. A: .space 512 3. .space 16 # 4\*4 4. B: .space 512 5. .space 16 # 4\*4 6. C: .space 512 |
| --- |

Això és així perquè ficant un .space de 16 bits fa que A[0], B[0] i C[0] no estiguin alineats i en aquest cas A[0] estigui en el bloc 0, B[0] en el bloc 1 i C[0] en el bloc 2.

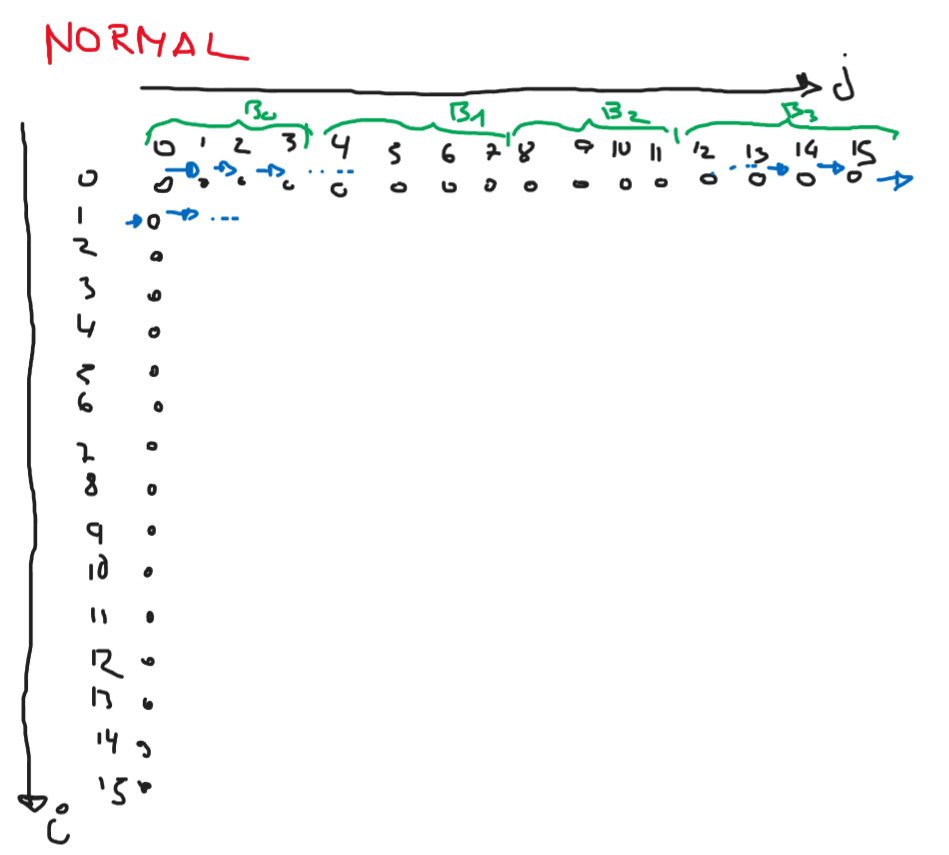
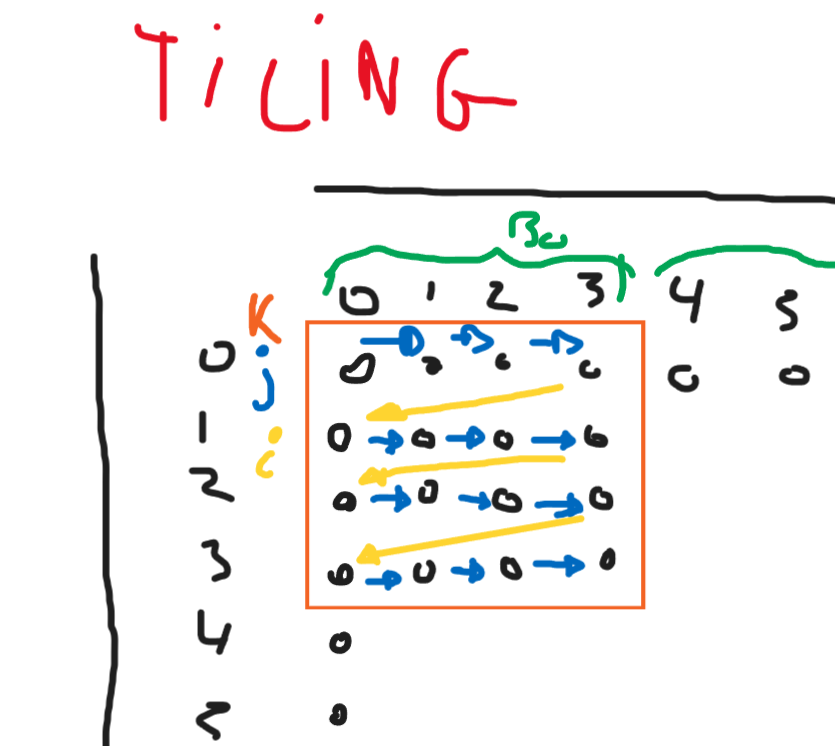
### Exercici 5.11 i 5.12

Són d’associativitat i encara no ho hem fet.

## Activitat 5.D

| Capacitat de la cache | Correspondència directa |
| --- | --- |
| 8 blocs |  |
| 16 blocs |  |
| 32 blocs |  |
| 64 blocs |  |
| 128 blocs |  |

### Exercici 5.13

| 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];  int main () {      int tmp;      for (int k = 0; k<4; k++) {          for (int i = 0; i<16; i++) {              tmp = 0;              for (int j = 0; j<4; j++) tmp += M[i][j+k\*4]\*V2[j+k\*4];              V1[i] += tmp;          }      }  } |
| --- |

| Capacitat de la cache | Correspondència directa |
| --- | --- |
| 8 blocs |  |
| 16 blocs |  |
| 32 blocs |  |
| 64 blocs |  |
| 128 blocs |  |

La millor diferència és en el cas de 8 blocs.

| .data  V1: .space 64  M:  .word   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      .word   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      .word   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      .word   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      .word   0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0      .word   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1      .word   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      .word   0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0      .word   1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0      .word   0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0      .word   0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1  V2: .word   -5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10        .text      .globl main  main:      li $t0,0  # k = 0      li $t9,4  # k\_max = j\_max = 4  fork:      bge $t0,$t9,end\_fork  # if (k>=k\_max) goto end\_fork      li $t1,0  # i = 0      li $t8,16  # i\_max = 16  fori:      bge $t1,$t8,end\_fori  # if (i>=i\_max) goto end\_fori      li $v0,0  # tmp = 0      li $t2,0  # j = 0  forj:      bge $t2,$t9,end\_forj  # if (j>=j\_max) goto end\_forj      # === MATRIU      # M[0][0] + ((i\*NC)+(j+k\*4))\*T      # Calcular Columna (j+k\*4)\*T      sll $t3,$t0,2  # k\*4      addu $t3,$t3,$t2  # j+k\*4      sll $t3,$t3,2  # (J+k\*4)\*4  ## int = 4 B      # Calcular fila (i\*NC)\*T      sll $t4,$t1,6  # i\*4\*2 ## 6 = 4+2 on 4 = NC i 2 = Tamany de int      # Calcular posició      la $t5,M      addu $t5,$t5,$t3      addu $t5,$t5,$t4      lw $t5,0($t5)  # M[i][j+k\*4]        # === VECTOR      la $t6,V2      addu $t6,$t6,$t3  # @V2[j+k\*4]      lw $t6,0($t6)  # V2[j+k\*4]      # === OPERAR      mult $t5,$t6  # M[i][j+k\*4]\*V2[j+k\*4]      mflo $t7      addu $v0,$v0,$t7  # tmp += M[i][j+k\*4]\*V2[j+k\*4]        addiu $t2,$t2,1  # j++      b forj  end\_forj:      # === VECTOR      la $t5,V2      sll $t6,$t1,2  # i\*4      addu $t5,$t5,$t6  # @V[i]      lw $t7,0($t5)  # V[i]      addu $t7,$t7,$v0  # V[i] += tmp      sw $t7,0($t5)      addiu $t1,$t1,1  # i++      b fori  end\_fori:      addiu $t0,$t0,1  #k++      b fork  end\_fork:      jr $ra |
| --- |