ESTRUCTURA DE COMPUTADORS
Grado de Ingeniería Informática

Pràctica 10

EL SISTEMA DE MEMÒRIA CAU EN EL MIPS R2000
LA CAU DE CODI

Introducció

En esta pràctica es treballa amb la memòria cau del MIPS R2000. Els conceptes estudiats en les pràctiques anteriors són fonamentals per a entendre el funcionament de la memòria cau i la seua influència en el temps d'execució dels programes. La ferramenta de treball és el simulador del processador MIPS R2000 denominat PCSpim-Cache.

Objectius

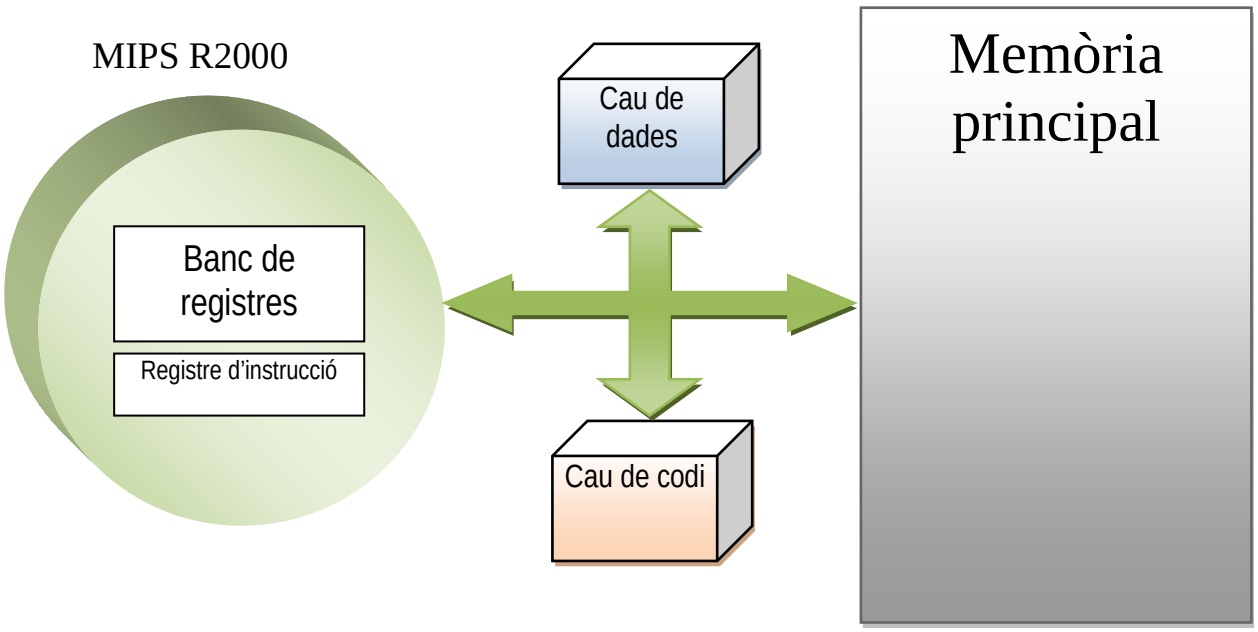
- Determinar com ajuda la memòria cau per a reduir el temps d'accés a la informació (instruccions).
- Conéixer com la memòria cau interpreta les adreces emeses pel processador.
- Analitzar com influïx l'organització de la memòria cau en la taxa d'encerts.

Material

El material requerit es trova a la carpeta de PoliformaT: Recursos -> Pràtiques -> P10

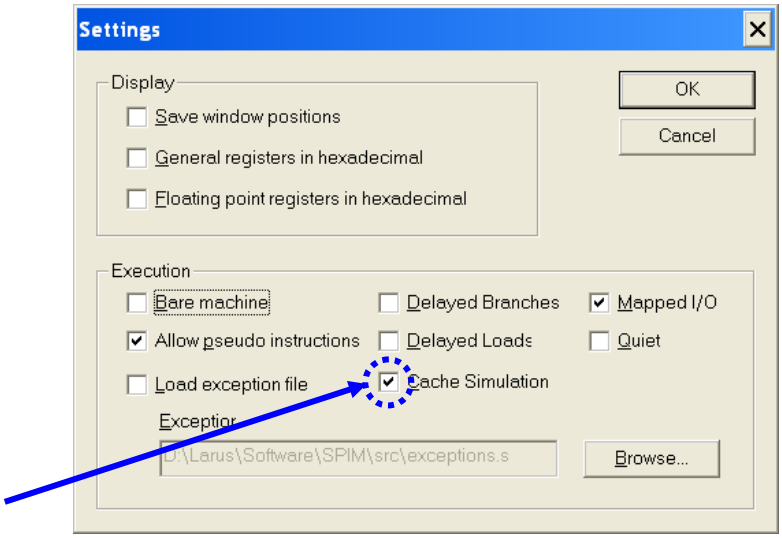
Configuració del simulador PCSpim-Cache

El simulador PCSpim-Cache és una extensió del simulador PCSpim bàsic que permet l'execució de programes escrits en ensamblador del processador MIPS R2000 tenint en compte el sistema memòria cau. El processador MIPS R2000 està dissenyat en la realitat per a utilitzar un únic sistema de memòria cau de primer nivell (L1) compost per una cau d'instruccions i una cau de dades (Il.lustració 1), les dues externes al processador. La cau de dades servix per a emmagatzemar informació ubicada en el segment de dades, és a dir, aquella que és llegida o escrita per mitjà de les instruccions de càrrega i emmagatzematge (`lb`, `lbu`, `lh`, `lhu`, `lw`, `lwc1`, `sb`, `sh`, `sw`, `swc1`); la cau de codi s'usa exclusivament per a emmagatzemar informació del segment de codi, açò és, instruccions, i per tant només s'accedix en operacions de lectura. En resum, la cau de dades és un intermediari entre la memòria principal i el banc de registres, mentre que la cau de codi ho és entre la memòria principal i el registre d'instrucció.



Il.lustració 1. La memòria cau en el processador MIPS R2000

El simulador amb que treballarem inclou un primer nivell de memòria cau organitzada en dos memòries, una de dades i una altra d'instruccions. Per a activar l'opció de simulació de memòries cau ha de seleccionar-se l'opció "*Cache Simulation*" dins de les opcions de configuració del simulador tal com es mostra en la Il.lustració 2. També pot accedir-se a través del menú *Simulator*|*Settings...*



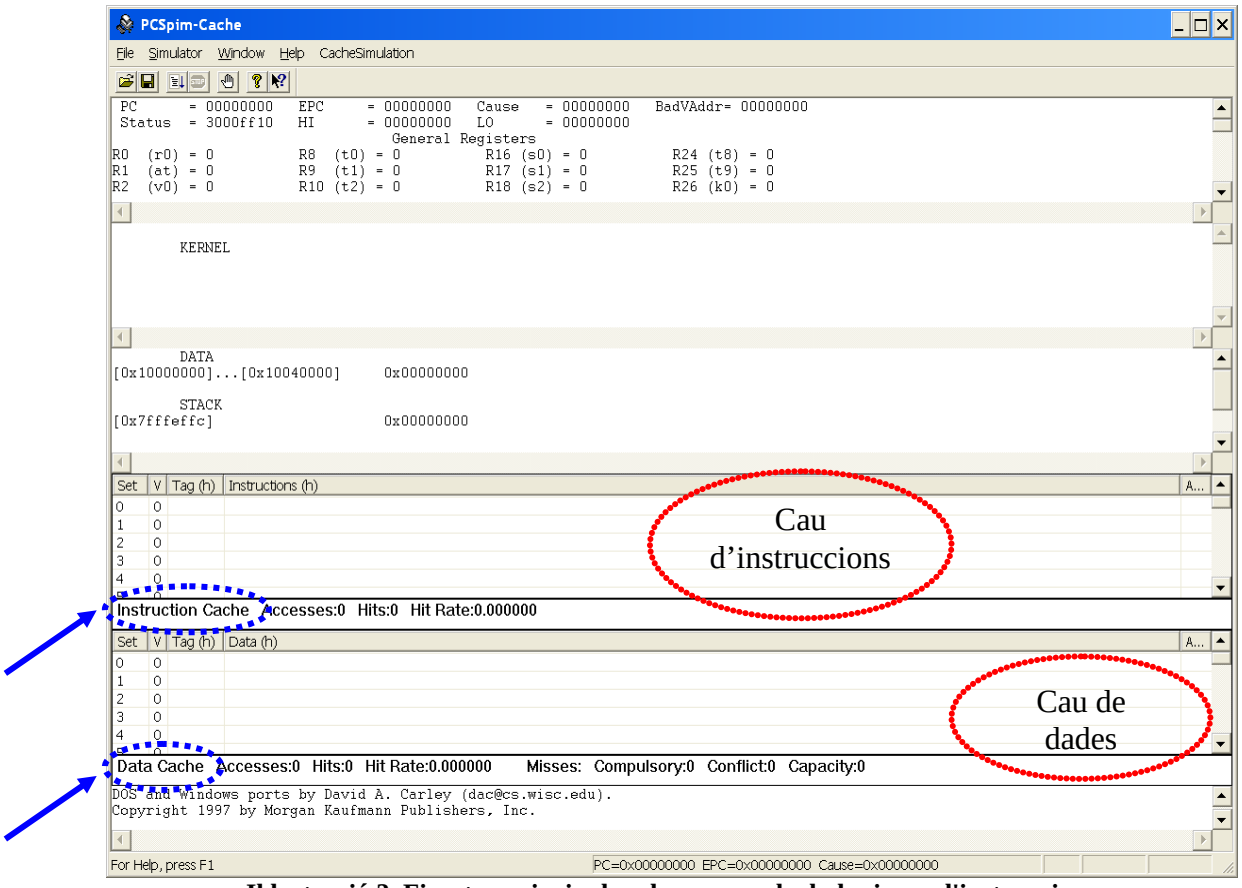
Il.lustració 2. Activació de Cache Simulation

Quan l´esmentada opció es troba activa, apareix un nou menú desplegable denominat *Cache Simulation* en la finestra principal del simulador. En el menú haurà de triar-se el tipus de cau a simular: dades, codi o ambdós (opció *Cache Configuration*) i l'organització i polítiques de la mateixa (opció *Cache Settings*).

Abans d'especificar l'organització de la cau ha de triar-se el tipus de cau a simular: de dades o d'instruccions. També és possible simular dos caus independents (dades i instruccions) al mateix

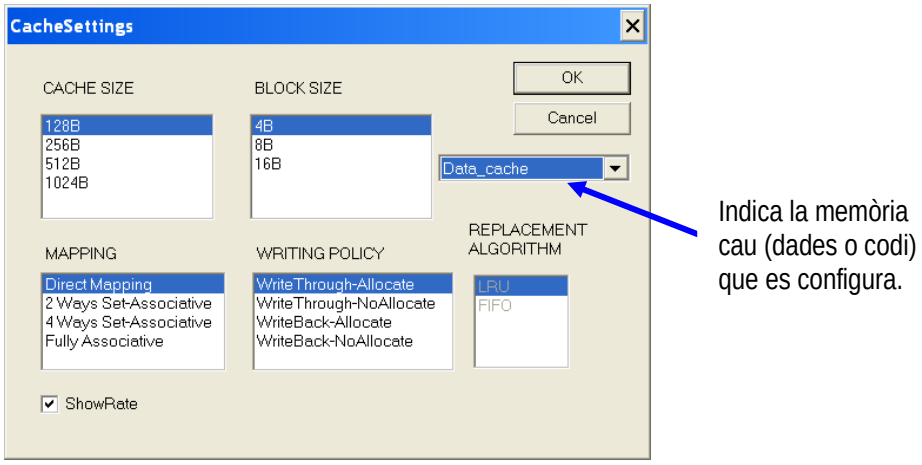
temps (arquitectura Harvard). Per a cada una de les memòries cau que se simula apareix un marc independent, tal com es mostra en la Il.lustració 3. Al peu de cada un d'estos marcs hi ha una línia de text que mostra les estadístiques associades a cada una de les caus (nombre d'accessos, encerts, etc.).

Respecte a la representació gràfica, cada conjunt de la cau apareix en una filera distinta. És a dir, la informació corresponent a totes les vies d'un mateix conjunt (etiquetes, bit vàlid, dades, valor LRU, etc.) es mostra en la mateixa filera de la pantalla. Esta disposició canvia només per a una cau de correspondència totalment associativa, i en este cas cada línia representa una via, ja que en este tipus de correspondència només hi ha un conjunt.



Il.lustració 3. Finestra principal amb una cau de dades i una d'instruccions

Com s'ha mencionat prèviament, una vegada seleccionat el tipus de memòria cau a simular han d'indicar-se els paràmetres que definixen la seua organització. La Il.lustració 4 mostra la finestra de diàleg corresponent. Com s'aprecia, ha d'especificar-se la geometria de la cau (capacitat de la memòria, grandària del bloc i nombre de vies), les polítiques d'escriptura, i l'algoritme de reemplaçament. També ha de seleccionar-se l'opció *ShowRate* si es desitja que es mostren estadístiques en pantalla.



Il.lustració 4. Finestra de diàleg de configuració de la cau

Una vegada realitzada la configuració ja es pot procedir amb la càrrega del programa escrit en assemblador i l'execució del mateix. Este procés es realitza exactament de la mateixa manera que en el simulador PCSpim bàsic.

Programa de treball: producte d'un vector per una constant

En la solució de molts problemes de càlcul numèric es requerix la multiplicació dels elements d'un vector per una constant: `I=k*X`. A continuació es presenta el codi d'un programa en ensamblador que du a terme esta operació. Els nombres enters manejats pel programa, codificats en complement a dos, són de 32 bits de longitud. El programa suposa que el resultat dels productes `k*X[i]` no excedix de 32 bits. Este programa és semblant a l'estudiat en una pràctica precedent encara que amb lleugeres modificacions. Atés que farem alguna modificació sobre este codi, al programa baix indicat ens referirem com a *programa original*.

```
                ##########################################################
                # Segment de dades
                ##########################################################

                .data 0x10000000
A:              .word 0,1,2,3,4,5,6,7   # Vector A
                .data 0x10001000
B:              .space 32               # Vector B (resultat)
                .data 0x1000A030
k:              .word 7                 # Constant escalar
dim:            .word 8                 # Dimensió dels vectors

                ##########################################################
                # Segment de codi
                ##########################################################

                .text 0x00400000
                .globl __start

__start:        la $a0, A               # $a0 = adreça de A
                la $a1, B               # $a1 = adreça de B
                la $a2, k               # $a1 = adreça de k
                la $a3, dim             # $a2 = adreça dimensió
                jal sax                 # Crida a subrutina

                ##########################################################
                # Fi d'execució per mitjà de crida al sistema
                ##########################################################

                addi $v0, $zero, 10     # Codi per a exit
                syscall                 # Fi de l'execució
```

```
                ########################################################
                # Subrutina que calcula Y <- k*X
                # $a0 = Adreça inici vector X
                # $a1 = Adreça inici vector Y
                # $a2 = Adreça constant escalar k
                # $a3 = Adreça dimensió dels vectors
                ########################################################

sax:            lw $a2, 0($a2)          # $a3 = constant k
                lw $a3, 0($a3)          # $a3 = dimensió
bucle:          lw $t0, 0($a0)          # Lectura de X[i] en $t0
                mult $a2, $t0           # Efectua k*X[i]
                mflo $t0                # $t0 <- k*X[i] (HI val 0)
                sw $t0, 0($a1)          # Escriptura de Y[i]
                addi $a0, $a0, 4        # Adreça de X[i+1]
                addi $a1, $a1, 4        # Adreça de Y[i+1]
                addi $a3, $a3, -1       # Disminució número elements
                bgtz $a3, bucle         # Bota si queden elements
                jr $ra                  # Retorn de subrutina

                .end
```

Abans de veure la relació entre l'execució d'este programa i el sistema de memòria cau és necessari analitzar la seua estructura i comportament. **No cal que ho carregues en el simulador**, es pot analitzar en paper i entendre com funciona. En este punt és important ser conscient que, durant l'execució del programa, cada instrucció executada ha sigut llegida del segment de codi i portada al Registre *d'Instrucció* per a la seua descodificació. Així mateix, cal tindre en compte que el vector A s'accedix en operacions de lectura (*load*) mentre que el vector B és accedit per mitjà d'operacions d'escriptura (*store*).

1. ►Quants elements tenen els vectors del programa? Quants bytes ocupa cada element?

   Hi ha 8 elements de 32 bits cada un, per tant 4 bytes ocupa cada element.

En primer lloc determinarem la grandària de les variables del programa en el segment de dades i la grandària de les instruccions del programa en el segment de codi.

2. ►Completa la següent informació del segment de dades. Utilitza el sistema hexadecimal per a expressar les adreces de memòria (utilitza la notació hexadecimal al llarg de tota la pràctica).

Adreça inicial del vector A	0x10000000
Bytes ocupats pel vector A	4 bytes x 8 elements
Adreça inicial del vector B	0x10001000
Bytes ocupats pel vector B	4 bytes x 8 elements
Adreça de la variable k	0x1000A030
Adreça de la variable `dim`	0x1000A034

3. ►Completa la següent informació del segment de codi. En este cas no oblides tindre en compte la traducció de les **pseudoinstruccions** del programa en instruccions màquina, ja que són estes últimes les úniques que cal considerar. En este cas és d'utilitat carregar el programa en el simulador (no cal executar-ho) per a veure la adreça on es troba l'última instrucció del programa.

Adreça de la primera instrucció	0x00400000
Adreça de l'última instrucció	0x00400050
Nombre d'instruccions del programa	21
Bytes ocupats pel codi del programa (instruccions)	21 x 4 bytes

A partir d'este moment ens anem a interessar pels aspectes dinàmics del programa i la seua relació amb la memòria. El més important ara és conéixer el nombre d'accessos a memòria efectuats pel programa. Els accessos es fan tant al segment de codi (insistim en el fet que cada instrucció s'ha de llegir de memòria per a portar-la al Registre *d'Instrucció*) com al segment de dades (per a llegir o escriure les variables del programa per mitjà de les instruccions de tipus *load* i *store*).

4. ►Determina el nombre d'accessos al sistema de memòria del programa. Estos valors són molt importants perquè ens serviran més tard per a conéixer quin nombre d'accessos del total són servits per la memòria cau, açò és, podrem distingir entre accessos que són encerts i accessos que són fallades.

Accessos al segment de dades	18
Accessos al segment de codi	77

Memòria cau de codi

Considerarem ara l'existència d'una memòria cau de codi amb les següents característiques:

Paràmetre	Valor
Capacitat	128 bytes
Correspondència	Directa
Bloc o línia	4 bytes

De moment **no cal que utilitzes el simulador**, ho farem un poc més tard. No oblides que una memòria cau de codi rep únicament operacions de lectura, ja que el processador es limita a llegir les instruccions per a executar-les. Així mateix, tin en compte que una línia només pot emmagatzemar una instrucció i que el programa sencer cap en la memòria cau.

5. ►Tenint en compte les característiques anteriors, indica quantes línies hi ha en la memòria cau.

   32 línies

6. ►Indica quina serà la interpretació que esta memòria cau farà de les adreces que reba (camps d'etiqueta, línia i desplaçament).

   etiqueta = 27 bits, línea = 5 bits, desplaçament = 0 bits

7. ►La instrucció del programa `jal sax` està emmagatzemada en la adreça `0x0040001C` del segment de dades. Indica a quina línia de la cau s'ubicarà i amb quina etiqueta.

   Etiqueta 0x0020000, en la línea 7

La memòria cau funciona amb la **informació de control** emmagatzemada en cada una de les línies. Esta informació de control inclou el bit de vàlid, els bits d'etiqueta i, segons el cas, el bit de modificat, els bits del comptador per a l'algoritme de reemplaçament, etc.

8. ►Calcula, per a este cas, quants bits de control s'emmagatzemen per línia. Així mateix, calcula el volum del directori, açò és, el nombre total de bits de control continguts en la memòria cau de codi.

Bits de control per línia	1 bit de validesa, 27 bits d'etiqueta
Volum del directori (bytes)	4 bytes (29 bits)

En estos moments ja estas en condicions d'utilitzar el simulador. Defineix un sistema de memòria cau tant per a dades com per a instruccions (arquitectura Harvard). No et preocupis de la cau de dades (l'estudiarem més tard), de moment centrat en la de codi i configura-la amb les característiques que hem definit anteriorment (capacitat de 128 bytes, correspondència directa i grandària de línia de 4 bytes).

9. ►Carrega el *programa original i* executa'l per mitjà de l'opció F10 (pas a pas) per a poder seguir amb detall l'efecte sobre la memòria cau de codi. Observa que la lectura de qualsevol instrucció afecta la cau de codi, però l'execució de les instruccions de memòria (`lw`, `sw`, etc.) afecta, a més, a la cau de dades que ara no considerem. A més, fixat en que el processament de les 18 primeres instruccions només originen fallades (missatge *miss* en el simulador) i que el primer encert es produïx en l'accés dinou a la cau d'instruccions. Completa la taula següent:

Accessos al segment de codi	77
Encerts	56
Fallades	21
Taxa d'encerts (H)	0.727273

10. ►Confirma que la instrucció `jal sax` s'emmagatzema en la línia prevista i amb l'etiqueta calculada anteriorment.

Si la linia es correcta

11. ►Suposa ara que la memòria principal del MIPS R2000 està implementada amb mòduls els xips de la qual funcionen a 50 MHz (període de 20 ns) i tenen com a paràmetres t_{CL}=2 (latència de CAS) i t_{RCD}=3; ambdós paràmetres estan expressats en cicles de rellotge. Suposa a més que el temps d'accés a la memòria cau és de 10 nanosegons. Recorda que este temps es pot calcular per mitjà de la fórmula:

100ns = (tcl + trcd) + 20ns

$$T = H \times T_{encert} + (1-H) \times T_{fallada}$$

T = 0.727273 x (56x100ns) + (1-0.727273) x (21x100ns)= 4645.4555ns

En este context, determina el temps mitjà d'accés al segment de codi experimentat pel programa.

Aprofitament del principi de localitat

Com has pogut comprovar, amb la configuració anterior de cau una línia de cau només pot contindre una instrucció. Si volem aprofitar el **principi de localitat** del programa per a reduir el temps d'accés a memòria podem fer més gran la grandària de bloc. D'esta manera aconseguim que en una mateixa línia càpien diverses instruccions.

12. ►Amb el simulador configura la memòria cau de codi amb una grandària de bloc de 16 bytes amb la resta de paràmetres anteriors sense canvis. Carrega i executa ara el *programa original* i completa la taula següent:

Accessos al segment de codi	77
Encerts	71
Fallades	6
Taxa d'encerts (H)	0.922078

13. ►Com s'aprecia, el nombre de fallades s'ha vist reduït de forma considerable. Quina és la raó?

Com caben 4 instruccions en un bloc simplement faran falta 6 per a carregar totes les instruccions en caché, per això ha decrementat considerablement