

Práctica 11

EL SISTEMA DE MEMORIA CACHE EN EL MIPS R2000 LA CACHE DE DATOS

Introducción

En esta práctica se trabaja se continua el trabajo empezado en la práctica anterior con la memoria cache del MIPS R2000. Incide en el funcionamiento de la memoria cache de datos y su influencia en el tiempo de ejecución de los programas y se utiliza como punto de partida el mismo programa de la práctica anterior. La herramienta de trabajo sigue siendo el simulador del procesador MIPS R2000 denominado PCSpim-Cache.

Objetivos

- Determinar cómo ayuda la memoria cache para reducir el tiempo de acceso a la información (datos).
- Conocer cómo la memoria cache interpreta las direcciones emitidas por el procesador.
- Analizar cómo influye la organización de la memoria cache en la tasa de aciertos.

Material

El material se puede obtener de la carpeta de recursos de PoliformaT.

- Simulador PCSpim-Cache del MIPS R2000.
- Archivos fuente (`programa.s`).

Programa de trabajo: producto de un vector por una constante

El *programa original* es el mismo de la anterior práctica de la memoria cache.

```
#####  
# Segmento de datos  
#####  
  
A:      .data 0x10000000  
        .word 0,1,2,3,4,5,6,7 # Vector A  
        .data 0x10001000  
B:      .space 32             # Vector B (resultado)  
        .data 0x1000A030  
k:      .word 7               # Constante escalar  
dim:    .word 8               # Dimensión de los vectores
```

```
#####
# Segmento de código
#####

.text 0x00400000
.globl __start

__start:    la $a0, A           # $a0 = dirección de A
            la $a1, B           # $a1 = dirección de B
            la $a2, k           # $a1 = dirección de k
            la $a3, dim         # $a2 = dirección dimensión
            jal sax             # Llamada a subrutina

#####
# Fin de ejecución mediante llamada al sistema
#####

addi $v0, $zero, 10           # Código para exit
syscall                               # Fin de la ejecución

#####
# Subrutina que calcula Y <- k*X
# $a0 = Dirección inicio vector X
# $a1 = Dirección inicio vector Y
# $a2 = Dirección constante escalar k
# $a3 = Dirección dimensión de los vectores
#####

sax:        lw $a2, 0($a2)       # $a3 = constante k
            lw $a3, 0($a3)       # $a3 = dimensión
bucle:      lw $t0, 0($a0)       # Lectura de X[i] en $t0
            mult $a2, $t0        # Efectúa k*X[i]
            mflo $t0             # $t0 <- k*X[i] (HI vale 0)
            sw $t0, 0($a1)       # Escritura de Y[i]
            addi $a0, $a0, 4      # Dirección de X[i+1]
            addi $a1, $a1, 4      # Dirección de Y[i+1]
            addi $a3, $a3, -1     # Decremento número elementos
            bgtz $a3, bucle       # Salta si quedan elementos
            jr $ra               # Retorno de subrutina

.end
```

Memoria cache de datos

Vamos a considerar ahora la existencia de una memoria cache de datos con las siguientes características:

Parámetro	Valor
Capacidad	256 bytes
Correspondencia	Directa
Bloque o línea	16 bytes
Política de escritura	Directa (<i>write through</i>) con ubicación (<i>write allocate</i>)

La correspondencia es directa, como en el caso anterior en el que hemos analizado el comportamiento de la memoria cache de código. En este apartado hemos de fijarnos en los accesos que se hacen en el programa mediante las instrucciones de *load* y *store*. En este caso, hay una lectura (instrucción *lw*) para la constante *k* y otra para el tamaño de los vectores *dim*, así como una lectura (*lw*) y una escritura (*sw*) por cada elemento de los vectores. **De momento no es necesario que use el simulador.**

- [illegible]

- | |
|------------------------------|
| <p>၆ ငွေဝင် နှင့် ငွေပုံ</p> |
|------------------------------|

3. ► Indique en qué líneas de la cache se ubican los vectores A y B.

[illegible]

4. ► Cargue el *programa original* en el simulador configurado con las características anteriores para la memoria cache de datos y ejecútelo mediante la opción F10 (paso a paso) a fin de analizar su comportamiento. Asegúrese de que los vectores se almacenan en las líneas previstas de la cache de datos y después complete la siguiente tabla:

5. ► ¿Cuál ha sido el acceso que ha provocado el único acierto en la memoria cache de datos?

[illegible]

Set	V	Tag (h)	Data (h)
1	0		
2	0		
3	1	1000a0	00000007 00000008 00000000 00000000

Como vemos tiene la etiqueta 1000a0
lo que significa que carga todo el bloque
como vemos en Data(h) , al ser de 16 bytes
cabén 4 elementos .
Carga la variable "K" y "dim" al ir seguida de "k".
K error pero "dim" acierto(en acceso), al cargar todo
el bloque.

- || 8æðãæåÁ^Áæ Á^△|^} 8æ Å^|Á|[*|æ æ
Ó| [~ ^• Å^Áæç FEEEEÁÁÁ
Ó| [~ ^• Å^Áæç FEEFÁÁÁ
Tã { æ|å^æ| [| Å ç Á| Å^ç 8æå| { [Á|Á ã { [Å| [~ ^Á

En los apartados siguientes vamos a tratar de aumentar la tasa de aciertos en los accesos al segmento de datos evitando que los dos vectores que maneja el programa se ubiquen en las mismas líneas de memoria cache. En resumen, vamos a experimentar con las siguientes acciones:

- Cambiar la política de escritura de la cache para evitar que haya ubicación en caso de fallo.
- Cambiar la ubicación de los vectores en el segmento de datos mediante las directivas del programa `.data`.
- Cambiar la correspondencia de la memoria cache para que haya una mayor asociatividad. En este caso probaremos con una política asociativa por conjuntos y con una política totalmente asociativa.

Primera alternativa: cambio en la política de escritura

El parámetro responsable de que el vector B se traiga a la memoria cache de datos es la utilización de una política de ubicación en escritura cuando se produce un fallo. En consecuencia, para evitar que el vector B colisione en la cache con el vector A hay que aplicar una política de no ubicación en escritura.

- Configure la memoria cache de datos con una política de escritura directa (*write through*) sin ubicación (*no write allocate*). Ejecute ahora el *programa original* mediante la opción F10 (paso a paso) a fin de analizar su comportamiento. Complete la siguiente tabla:

Accesos al segmento de datos	18
Aciertos	7
Fallos	
Tasa de aciertos (H)	0.38

Si examinamos la ejecución del programa veremos que hay un fallo por cada acceso al vector B, dos fallos en el acceso al vector A (lectura de los elementos `A[0]` y `A[4]`) y un fallo en la lectura de la variable `k`.

Ahora habrá podido comprobar que el vector B no llega a desplazar al vector A de la memoria cache y, por tanto, no hay fallos debido a la colisión entre los bloques de ambos vectores. Sin embargo, todavía se siguen produciendo fallos porque con la política de no ubicación hemos impedido que el vector B llegue a la memoria cache. De hecho, nunca podrá ubicarse en esta memoria porque este vector solamente se escribe pero nunca se lee.

Segunda alternativa: cambio en la ubicación de los vectores

Una manera distinta de mejorar la tasa de aciertos manteniendo la política de ubicación en escritura consiste en cambiar la ubicación de los vectores en el segmento de datos. En efecto, ya sabemos que las directivas `.data` del programa original previas a la declaración de ambos vectores hacen que compartan la misma línea de cache. Bastará, por tanto, con cambiar la dirección de inicio de uno de estos vectores con objeto de hacer que la correspondencia directa los sitúe en líneas diferentes.

- Cambie la directiva `.data 0x10001000` de manera que el vector B se ubique en las líneas 4 y 5 de la memoria cache. De este modo no habrá colisión con las líneas 0 y 1 (ubicación del vector A) ni con la línea 3 (ubicación de la constante `k` y de `dim`).

9. ► Ejecute el programa con el cambio anterior (recuerde que debe mantener la política de escritura **con ubicación**) y complete la siguiente tabla:

Set	V	Tag (h)	Data (h)
0	1	100000	00000000 00000001 00000002 00000003
1	1	100000	00000004 00000005 00000006 00000007
2	0		
3	1	1000a0	00000007 00000008 00000000 00000000
4	1	100010	00000000 00000007 0000000e 00000015
5	1	100010	0000001c 00000023 0000002a 00000031
6	0		

Data Cache Accesses:18 Hits:13 Hit Rate:0.722222 Misses: Compulsory:5 Conflict:0 Capacity:0

Tercera alternativa: aumento de la asociatividad de la cache

En este último apartado vamos a cambiar la configuración de la memoria cache con los siguientes parámetros:

Parámetro	Valor
Capacidad	256 bytes
Correspondencia	Asociativa por conjuntos
Número de vías	2
Bloque o línea	16 bytes
Política de escritura	Directa con ubicación
Política de reemplazo	LRU

En esta nueva situación hemos aumentado la asociatividad de la cache: un bloque proveniente de la memoria principal puede escoger entre las dos líneas del conjunto que le corresponde. Recuerde que estamos modificando la configuración de la memoria cache de datos para ejecutar el *programa original*. **De momento no hace falta que utilice el simulador.**

10. ► Indique cuál será la interpretación que esta memoria cache hará de las direcciones que reciba (campos de etiqueta, conjunto y desplazamiento).

Ö^•] KÁ Áã
Ô[} BÄVEG ÄFí ÁÄ Ä
Öã KÄ

11. ► Indique en qué conjuntos se ubicarán ahora los vectores A y B del *programa original*.

ÖããÄ^&f :Á[} ÁGä[~^• ÄÄ Ä [:äÄ[:Ää[~^ÄK
Ô[} B } ÄÄÄ

En vista de los valores anteriores, observamos que los bloques de los vectores A y B se mapean sobre los mismos conjuntos de la memoria cache. Recordemos que la correspondencia directa empleada hasta ahora mapeaba los bloques en las mismas líneas de memoria, originando reemplazos durante la ejecución del programa. ¿Estamos ahora en la misma situación? ¿Qué es lo que ha cambiado?

Ahora ya no existe problemas de reemplazo por esta en la misma linea ,ahora que son 2 vías desaparece dicho problema de conflictos y serán solo falladas de inicio.

12. ► Cargue en el simulador y ejecute el *programa original* con la nueva configuración de la cache y complete la siguiente tabla:

Set	V	LRU	Tag (h)	Data (h)	Way 0	Acc	V	LRU	Tag (h)	Data (h)	Way 1
0	1	1	200000	00000000	00000001 00000002 00000003		1	0	200020	00000000 00000007 0000000e 00000015	
1	1	1	200000	00000004	00000005 00000006 00000007		1	0	200020	0000001c 00000023 0000002a 00000031	
2	0	0					0	0			
3	1	0	200140	00000007	00000008 00000000 00000000		0	0			
4	0	0					0	0			
5	0	0					0	0			
6	0	0					0	0			
7	0	0					0	0			

Data Cache Accesses:18 Hits:13 Hit Rate:0.722222 Misses: Compulsory:5 Conflict:0 Capacity:0

Como se ha podido apreciar durante la ejecución del programa, en este caso no se produce ningún reemplazamiento. Todos los fallos son originados por el primer acceso al bloque seleccionado. Así pues, en este caso se ha producido un fallo en el acceso al bloque que contiene las variables k y dim, dos fallos en el acceso al vector A y otros dos fallos en el acceso al vector B.