ESTRUCTURA DE COMPUTADORES
Grado de Ingeniería Informática

Práctica 10

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

Introducción

En esta práctica se trabaja con la memoria cache del MIPS R2000. Los conceptos estudiados en las prácticas anteriores son fundamentales para entender el funcionamiento de la memoria cache y su influencia en el tiempo de ejecución de los programas. La herramienta de trabajo es 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 (instrucciones).
- 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 requerido está disponible en la carpeta de PoliformaT: Recursos -> Prácticas -> P10.

Configuración del simulador PCSpim-Cache

El simulador PCSpim-Cache es una extensión del simulador PCSpim básico que permite la ejecución de programas escritos en ensamblador del procesador MIPS R2000 teniendo en cuenta el sistema memoria cache. El procesador MIPS R2000 está diseñado en la realidad para utilizar un único sistema de memoria cache de primer nivel (L1) compuesto por una cache de instrucciones y una cache de datos (véase la Ilustración 1), ambas externas al procesador. La cache de datos sirve para almacenar información ubicada en el segmento de datos, es decir, aquella que es leída o escrita mediante las instrucciones de carga y almacenamiento (`lb`, `lbu`, `lh`, `lhu`, `lw`, `lwc1`, `sb`, `sh`, `sw`, `swc1`); la cache de código se usa exclusivamente para almacenar información del segmento de código, esto es, instrucciones, y por tanto solamente se accede en operaciones de lectura. En resumen, la cache de datos es un intermediario entre la memoria principal y el banco de registros, mientras que la cache de código lo es entre la memoria principal y el registro de instrucción.

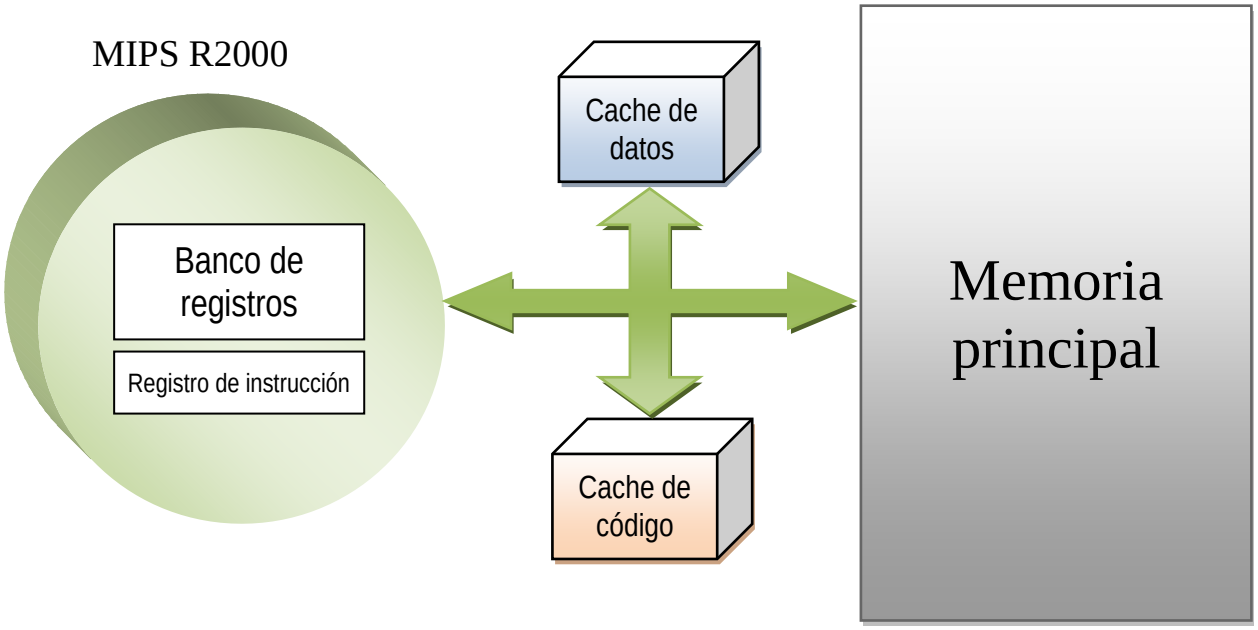

Ilustración 1 La memoria cache en el procesador MIPS R2000

El simulador con que vamos a trabajar incluye un primer nivel de memoria cache organizada en dos memorias, una de datos y otra de instrucciones. Para activar la opción de simulación de memorias cache debe seleccionarse la opción "*Cache Simulation*" dentro de las opciones de configuración del simulador tal y como se muestra en la Ilustración 2. También puede accederse a través del menú *Simulator*|*Settings...*

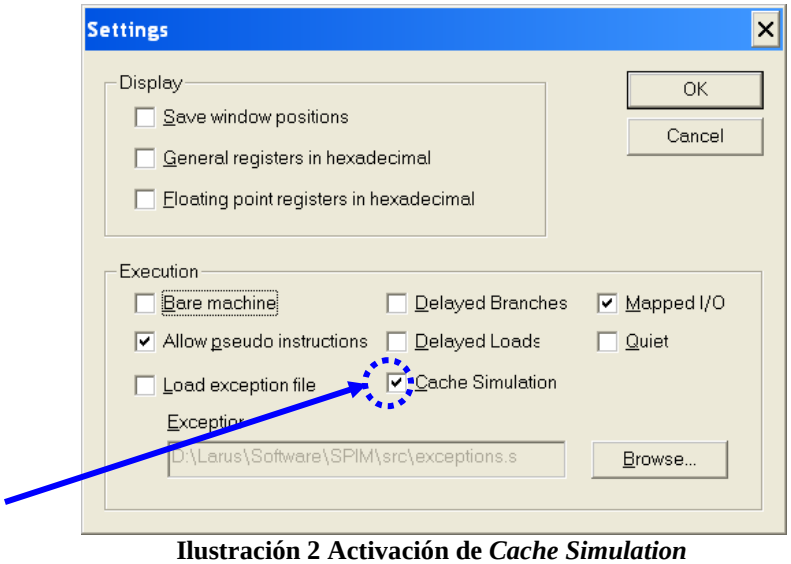

Ilustración 2 Activación de *Cache Simulation*

Cuando dicha opción se encuentra activa, aparece un nuevo menú desplegable denominado *Cache Simulation* en la ventana principal del simulador. En dicho menú deberá elegirse el tipo de cache a simular: datos, código o ambas (opción *Cache Configuration*) y la organización y políticas de la misma (opción *Cache Settings*).

Antes de especificar la organización de la cache debe elegirse el tipo de cache a simular: de datos o de instrucciones. También es posible simular dos caches independientes (datos e instrucciones) al mismo tiempo (arquitectura Harvard). Para cada una de las memorias cache que se simula aparece un marco independiente, tal y como se muestra en la Ilustración 3. Al pie de cada uno de estos marcos hay una línea de texto que muestra las estadísticas asociadas a cada una de las caches (número de accesos, aciertos, etc.).

Respecto a la representación gráfica, cada conjunto de la cache aparece en una fila distinta. Es decir, la información correspondiente a todas las vías de un mismo conjunto (etiquetas, bit válido, datos, valor LRU, etc.) se muestra en la misma fila de la pantalla. Esta disposición cambia solo para una cache de correspondencia totalmente asociativa, en cuyo caso cada línea representa una vía, ya que en este tipo de correspondencia solo hay un conjunto.

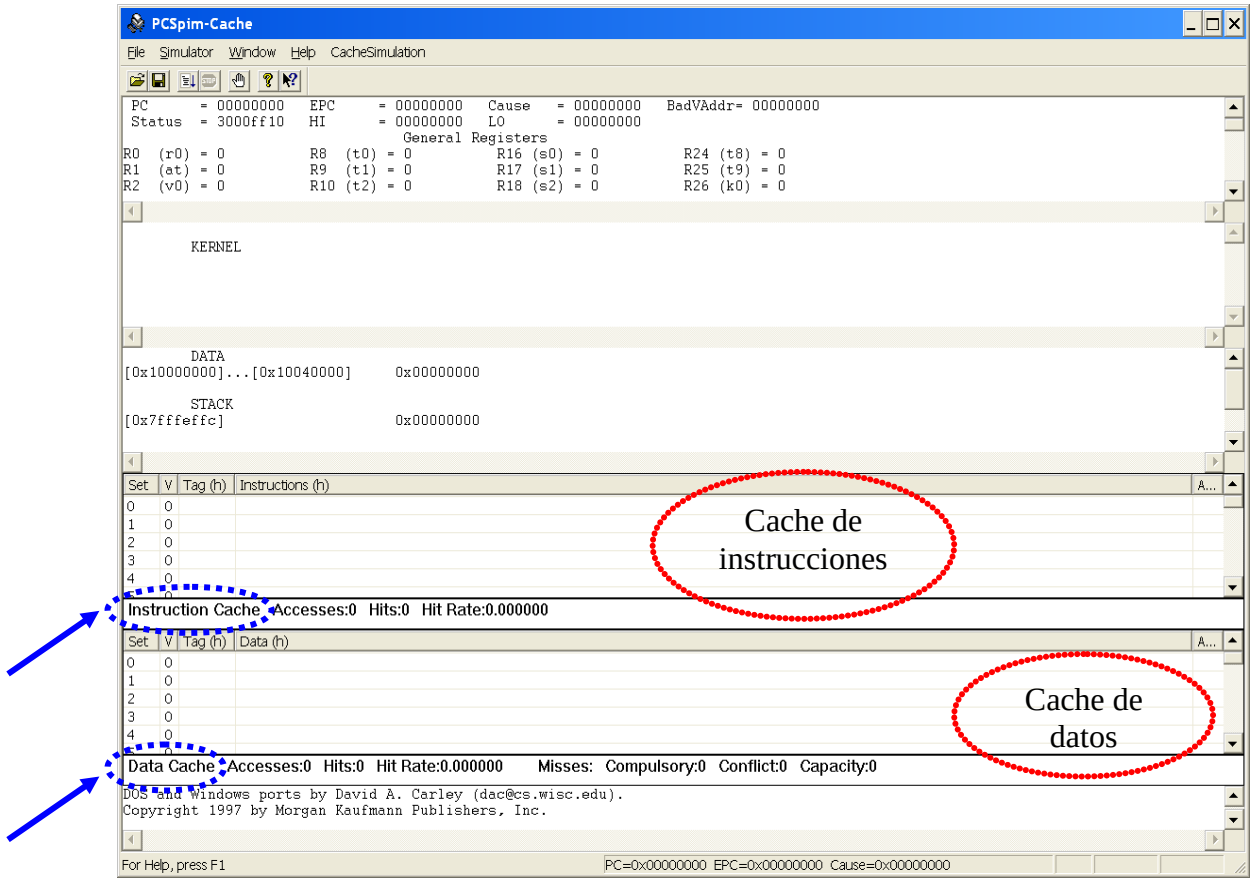

Ilustración 3 Ventana principal con una cache de datos y una de instrucciones

Como se ha mencionado previamente, una vez seleccionado el tipo o tipos de memoria cache a simular deben indicarse los parámetros que definen su organización. La Ilustración 4 muestra la ventana de diálogo correspondiente. Como se aprecia, debe especificarse la geometría de la cache (capacidad de la memoria, tamaño del bloque y número de vías), las políticas de escritura, y el algoritmo de reemplazo. También debe seleccionarse la opción *ShowRate* si se desea que se muestren estadísticas en pantalla.

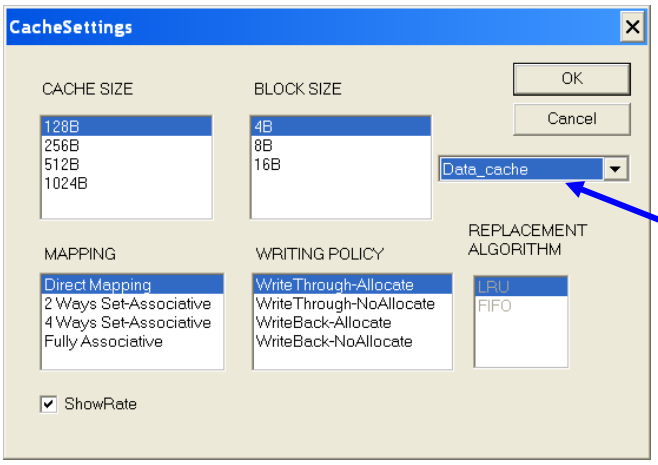

Ilustración 4 Ventana de diálogo de configuración de la cache

Una vez realizada la configuración ya se puede proceder con la carga del programa escrito en ensamblador y la ejecución del mismo. Este proceso se realiza exactamente de la misma forma que en el simulador PCSpim básico.

Programa de trabajo: producto de un vector por una constante

En la solución de muchos problemas de cálculo numérico se requiere la multiplicación de los elementos de un vector por una constante: `Y=k*X`. A continuación se presenta el código de un programa en ensamblador que lleva a cabo esta operación. Los números enteros manejados por el programa, codificados en complemento a dos, son de 32 bits de longitud. El programa supone que el resultado de los productos `k*X[i]` no excede de 32 bits. Este programa es similar al estudiado en una práctica precedente aunque con ligeras modificaciones. Dado que haremos alguna modificación sobre este código, al programa abajo indicado nos referiremos como *programa original*.

```
                ##########################################################
                # Segmento de datos
                ##########################################################

                .data 0x10000000
A:              .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
```

Antes de ver la relación entre la ejecución de este programa y el sistema de memoria cache es necesario analizar su estructura y comportamiento. **No hace falta que lo cargue en el simulador**, basta con que lo analice sobre el papel y entienda cómo funciona. En este punto es importante ser consciente de que, durante la ejecución del programa, cada instrucción ejecutada ha sido leída del segmento de código y llevada al *Registro de Instrucción* para su decodificación. Así mismo, hay que tener en cuenta que el vector A se accede en operaciones de lectura (*load*) mientras que el B es accedido mediante operaciones de escritura (*store*).

1. ► ¿Cuántos elementos tienen los vectores del programa? ¿Cuántos bytes ocupa cada elemento?

En primer lugar vamos a determinar el tamaño ocupado por las variables del programa en el segmento de datos y el tamaño ocupado por las instrucciones del programa en el segmento de código.

2. ► Complete la siguiente información del segmento de datos. Utilice el sistema hexadecimal para expresar las direcciones de memoria (haga igual a lo largo de toda la práctica).

Dirección inicial del vector A	
Bytes ocupados por el vector A	
Dirección inicial del vector B	
Bytes ocupados por el vector B	
Dirección de la variable k	
Dirección de la variable dim	

3. ► Complete la siguiente información del segmento de código. En este caso no olvide tener en cuenta la traducción de las **pseudoinstrucciones** del programa en instrucciones máquina, ya que son estas últimas las únicas que hay que considerar. En este caso le será de utilidad

cargar el programa en el simulador (no hace falta ejecutarlo) para ver la dirección donde se encuentra la última instrucción del programa.

Dirección de la primera instrucción	
Dirección de la última instrucción	
Número de instrucciones del programa	
Bytes ocupados por el código del programa (instrucciones)	

A partir de este momento nos vamos a interesar por los aspectos dinámicos del programa y su relación con la memoria. Lo más importante ahora es conocer el número de accesos a memoria efectuados por el programa. Dichos accesos se hacen tanto al segmento de código (insistimos en que cada instrucción se ha de leer de memoria para llevarla al *Registro de Instrucción*) como al segmento de datos (para leer o escribir las variables del programa mediante las instrucciones de tipo *load* y *store*).

4. ► Determine el número de accesos al sistema de memoria del programa. Estos valores son muy importantes porque nos servirán más tarde para conocer qué número de accesos del total son servidos por la memoria cache, esto es, podremos distinguir entre accesos que son aciertos y accesos que son fallos.

Accesos al segmento de datos	
Accesos al segmento de código	

Memoria cache de código

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

Parámetro	Valor
Capacidad	128 bytes
Correspondencia	Directa
Bloque o línea	4 bytes

De momento **no hace falta que utilice el simulador**, lo haremos un poco más tarde. No olvide que una memoria cache de código recibe únicamente operaciones de lectura, ya que el procesador se limita a leer las instrucciones para ejecutarlas. Así mismo, nótese que una línea solamente puede almacenar una instrucción y que el programa entero cabe en la memoria cache.

5. ► Teniendo en cuenta las características anteriores, indique cuántas líneas hay en la memoria cache.

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

7. ► La instrucción del programa `jal sax` está almacenada en la dirección `0x0040001C` del segmento de código. Indique en qué línea de la cache se ubicará y con qué etiqueta.

El funcionamiento de la memoria cache se basa en la **información de control** almacenada en cada una de las líneas. Esta información de control incluye el bit de válido, los bits de etiqueta y, según el caso, el bit de modificado, los bits del contador para el algoritmo de reemplazo, etc.

8. ► Calcule, para este caso, cuántos bits de control se almacenan por línea. Así mismo, calcule el volumen del directorio, esto es, el número total de bits de control contenidos en la memoria cache de código.

Bits de control por línea	
Volumen del directorio (bytes)	

En estos momentos ya está en condiciones de utilizar el simulador. Defina un sistema de memoria cache tanto para datos como para instrucciones (arquitectura Harvard). No se preocupe de la cache de datos (la estudiaremos más tarde), por el momento céntrese en la de código y configúrela con las características que hemos definido anteriormente (capacidad de 128 bytes, correspondencia directa y tamaño de línea de 4 bytes).

9. ► Cargue el *programa original* y ejecútelo mediante la opción F10 (paso a paso) para poder seguir con detalle el efecto sobre la memoria cache de código. Observe que la lectura de cualquier instrucción afecta a la cache de código, pero la ejecución de las instrucciones de memoria (`lw`, `sw`, etc.) afecta, además, a la cache de datos que ahora no consideramos. Además, fíjese en que el procesamiento de las 18 primeras instrucciones solamente originan fallos (mensaje *miss* en el simulador) y que el primer acierto se produce en el decimonoveno acceso a la cache de instrucciones. Complete la siguiente tabla:

Accesos al segmento de código	
Aciertos	
Fallos	
Tasa de aciertos (H)	

10. ► Confirme que la instrucción `jal sax` se almacena en la línea prevista y con la etiqueta calculada anteriormente.

11. ► Suponga ahora que la memoria principal del MIPS R2000 está implementada con módulos cuyos chips funcionan a 50 MHz (periodo de 20 ns) y tienen como parámetros t_{CL}=2 (latencia de CAS) y t_{RCD}=3 (tiempo entre RAS y CAS); ambos parámetros están expresados en ciclos de reloj. Suponga además que el tiempo de acceso a la memoria cache es de 10 nanosegundos. Recuerde que este tiempo se puede calcular mediante la fórmula:

$$T = H \times \text{Tacierto} + (1 - H) \times \text{Tfallo}$$

En este contexto, determine el tiempo medio de acceso al segmento de código experimentado por el programa.

Aprovechamiento del principio de localidad

Como ha podido comprobar, con la configuración anterior de cache una línea de cache solamente puede contener una instrucción. Si queremos aprovechar el **principio de localidad** del programa para reducir el tiempo de acceso a memoria podemos hacer más grande el tamaño de bloque. De esta forma conseguimos que en una misma línea quepan varias instrucciones.

12. ► Use el simulador y configure la memoria cache de código con un tamaño de bloque de 16 bytes y manteniendo el resto de parámetros como estaban. Cargue y ejecute ahora el *programa original* y complete la siguiente tabla:

Accesos al segmento de código	
Aciertos	
Fallos	
Tasa de aciertos (H)	

13. ► Como se aprecia, el número de fallos se ha visto reducido de forma considerable. ¿Cuál es la razón?