ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

ORGANIZACIÓN DE COMPUTADORES

GRADO EN INGENIERÍA DE COMPUTADORES

ACTIVIDADES PRÁCTICAS

ACTIVIDAD 1.2:
RESOLUCIÓN DE EJERCICIOS DE MEMORIA CACHÉ

CURSO 2019-20

Luis Rincón Córcoles
luis.rincon@urjc.es
Ángel Serrano Sánchez de León

INTRODUCCIÓN

En este cuaderno de actividades prácticas se proponen varios ejercicios de memoria caché, aprovechando la herramienta **“Data Cache Simulator”** del simulador de MIPS **MARS**. Los ejercicios se realizarán sobre diferentes configuraciones de memoria caché. Después de realizar cada ejercicio, se podrá cumplimentar un cuestionario asociado. Los cuestionarios se responderán a través del aula virtual. La nota obtenida en los mismos computa para el cálculo de la nota final de la asignatura, dentro del apartado de tests sobre actividades de laboratorio. Esta actividad es **optativa**.

EJERCICIO 1.2.1: CACHÉ ASOCIATIVA POR CONJUNTOS

Sea un computador con un procesador MIPS dotado de memoria caché. Se pretende ejecutar un programa del cual se muestra un fragmento de código fuente en C:

```
int m1[1024], m2[1024], m3[1024];
int n;
   …
int main (void) {
   …
   n = 0;
   do {
      m3[n] = m1[n] + m2[n];
      n = n+1;
   } while (n != 1024);
   …
}
```

El compilador empleado para traducir este programa a lenguaje ensamblador genera un código con las siguientes características:

- La variable **n** queda ubicada en un registro, por lo que no se reserva espacio en memoria para ella.
- El vector **m1** comienza en la dirección hexadecimal 0x10010000 y termina en la 0x10010FFF, y su tamaño en memoria es de 4 KBytes (1024 elementos de 4 bytes cada uno).
- El vector **m2** se sitúa justo a continuación del vector **m1**, por tanto comienza en la dirección hexadecimal 0x10011000 y termina en la 0x10011FFF. Su tamaño es exactamente igual al de **m1**.
- El vector **m3** se sitúa justo a continuación del vector **m2**, así que comienza en la dirección hexadecimal 0x10012000 y termina en la 0x10012FFF. Su tamaño coincide con el de **m1** y **m2**.
- Se accede a los vectores mediante punteros, que se incrementan dentro del bucle.

El código ensamblador resultante tras la traducción sería el siguiente:

```
      .data
m1:   .space      4096
m2:   .space      4096
m3:   .space      4096

      .text
      la    $s0, m1
      la    $s1, m2
      la    $s2, m3
# n = 0;
      li    $s3,0
# do {
do:
# m3[n] = m1[n] + m2[n]
      lw    $t0,0($s0)
      lw    $t1,0($s1)
      add   $t2,$t0,$t1
      sw    $t2,0($s2)
```

```
# n = n+1
      addi  $s3,$s3,1
# Incrementamos punteros
      addi  $s0,$s0,4
      addi  $s1,$s1,4
      addi  $s2,$s2,4
# while (n != 1024);
      li    $t3,1024
      bne   $s3,$t3,repeat
end_repeat:
# END
      li    $v0,10
      syscall
```

El código se incluye en el archivo **actividad1_2_1.asm**.

Se observa que en cada iteración hay 3 accesos a datos residentes en memoria:

- Lectura del elemento **m1[n]**.
- Lectura del elemento **m2[n]**.
- Escritura del elemento **m3[n]**.

Teniendo en cuenta que:

- En MIPS las direcciones son de 32 bits.
- Se direcciona a nivel de byte.
- El ancho de palabra (*word*) es de 32 bits.
- Suponemos que la caché de datos de nuestro procesador tiene un tamaño de 2 KBytes, es asociativa por conjuntos de 2 vías y el tamaño de cada bloque es de 4 palabras (16 bytes); se emplea política de escritura de *copy-back* y el algoritmo de reemplazo es LRU.
- El tiempo de acceso cuando hay acierto es de 1 ciclo.
- El tiempo de acceso cuando hay fallo es de 5 ciclos * número de palabras del bloque.

Responder a las siguientes cuestiones:

a) Indicar en qué campos se divide una dirección de un dato de memoria desde el punto de vista del acceso a la caché de datos, en qué orden se encuentran y cuál es el tamaño de cada uno de ellos. ¿Cuántos bloques hay en la caché de datos?

b) Rellenar la siguiente tabla, indicando la secuencia de accesos a la caché de datos, para las tres primeras iteraciones del algoritmo, suponiendo que la caché de datos se halla inicialmente vacía. Dado que en cada iteración hay 3 accesos a datos en memoria, será preciso rellenar 9 filas en la tabla. Se ofrece un pequeño modelo de cómo debería rellenarse la misma, donde i, j, k, m, p, q, r, U, V, W, X, Y y Z son los números que corresponda.

Elemento accedido	Dirección de memoria	Tipo de acceso	Conjunto y bloque	Evento (indicar bloque y conjunto involucrado en cada caso)
m1[i]	0x10010000	Lectura	Conjunto U Bloque V	Fallo. Se expulsa el bloque V del conjunto U y entran en él los elementos m1[j] … m1[k]
…	…	…	…	…
m2[m]	0x…	Lectura	Conjunto W Bloque X	Acierto. Está en el bloque X de dicho conjunto W.
…	…	…	…	…
m3[p]	0x…	Escritura	Conjunto Y Bloque Z	Fallo. Se expulsa al bloque Z del conjunto Y y entran en él los elementos m3[q] … m3[r].
…	…	…	…	…

c) Calcular razonadamente la tasa de fallos durante estas tres primeras iteraciones, y comprobarla mediante el simulador de caché de datos. Calcular razonadamente el tiempo medio de acceso para la caché de datos obtenido hasta el momento.

d) De acuerdo con lo expuesto en el apartado c), calcular razonadamente cuál sería la tasa de fallos para el programa completo, y comprobarla mediante el simulador. Calcular razonadamente tiempo medio de acceso final para la caché de datos.

e) Realizar un estudio de las prestaciones que se obtendrían con una caché de datos asociativa por conjuntos en función del grado de asociatividad de la misma. Mantener fijo el tamaño de caché (2 Kbytes) y el tamaño de bloque (4 palabras). Calcular en cada caso la tasa de fallos y el tiempo medio de acceso a memoria.

f) Realizar un estudio de las prestaciones que se obtendrían con una caché de datos asociativa por conjuntos en función del tamaño de cada bloque. Mantener fijo el tamaño de caché (2 Kbytes) y elegir el grado de asociatividad en función de los resultados del apartado e). Calcular en cada caso la tasa de fallos y el tiempo medio de acceso a memoria.

g) Elegir la mejor configuración posible para la memoria caché, y explicar por qué esta nueva configuración mejora el rendimiento. Comprobar la tasa de fallos mediante el simulador, y dar el tiempo medio de acceso final obtenido. En la elección, tener en cuenta la relación precio / prestaciones de la memoria.

CUESTIONARIO 1.2.1

Pregunta 1

La descomposición en campos de la dirección de memoria es:

a) 19 bits para la etiqueta, 7 para el conjunto, 4 para el desplazamiento de palabra y 2 para el desplazamiento de byte.
b) 20 bits para la etiqueta, 6 para el conjunto, 4 para el desplazamiento de palabra y 2 para el desplazamiento de byte.

c) 22 bits para la etiqueta, 6 para el conjunto, 2 para el desplazamiento de palabra y 2 para el desplazamiento de byte.
d) 21 bits para la etiqueta, 7 para el conjunto, 2 para el desplazamiento de palabra y 2 para el desplazamiento de byte.

Pregunta 2

¿Cuántos conjuntos hay en la caché de datos? (Responder en el cuadro destinado a tal efecto).

Pregunta 3

¿Cuántos ciclos consume la penalización de fallo? (Responder en el cuadro destinado a tal efecto).

Pregunta 4

¿Cuántos aciertos se consignan en la tabla de 9 filas de este ejercicio? (Responder en el cuadro destinado a tal efecto).

Pregunta 5

Para la configuración inicial de la caché, ¿cuál es el tiempo medio de acceso a memoria, expresado en ciclos? (Responder en el cuadro destinado a tal efecto). Escribir el resultado redondeado a dos cifras decimales.

Pregunta 6

Para este caso en concreto, manteniendo constante el tamaño total de la caché (2 kB) y poniendo un grado de asociatividad de 4 vías, al aumentar el tamaño de bloque se observa que:

a) Aumentan la tasa de aciertos y el tiempo medio de acceso a memoria.
b) Aumenta la tasa de aciertos, pero disminuye el tiempo medio de acceso a memoria.
c) Disminuye la tasa de aciertos, pero aumenta el tiempo medio de acceso a memoria.
d) Disminuyen la tasa de aciertos y el tiempo medio de acceso a memoria.

Pregunta 7

Al aumentar el grado de asociatividad de la caché a 4 vías por conjunto, ¿cuál pasa a ser el tiempo medio de acceso a memoria, expresado en ciclos? (Responder en el cuadro destinado a tal efecto). Escribir el resultado redondeado a dos cifras decimales.

Pregunta 8

Independientemente de la configuración de la caché, ¿cuál es el número total de acceso a memoria para este programa? (Responder en el cuadro destinado a tal efecto).

EJERCICIO 1.2.2: CACHÉ TOTALMENTE ASOCIATIVA

Sea un computador con ancho de palabra 32 bits cuya unidad direccionable de memoria es el byte, y que dispone de una memoria caché para instrucciones y una memoria caché para datos. Ambas memorias disponen de bloques de 2 palabras y una política totalmente asociativa. Considere los siguientes fragmentos de código (cuya versión en ensamblador se incluye en el archivo **actividad1_2_2.asm**):

Código 1:

```
for (i=0; i<512; i++) {
   x[i] = a[i];
   tmp = tmp + x[i];
}
```

Código 2:

```
for (i=511; i>=0; i--) {
   x[i] = i;
}
```

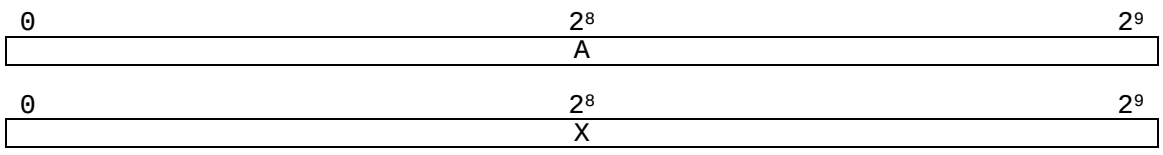
donde **a** y **x** representan vectores de 2^9 números enteros de 32 bits y las variables **i** y **tmp** se almacenan únicamente en registros. La caché se encuentra inicialmente vacía. El tamaño de la caché de datos es de 2 KB. Se pide:

a) Indicar razonadamente en qué campos se divide una dirección de memoria y cuántos bits ocupa cada uno.

b) Rellenar una tabla como la siguiente, considerando los accesos a los 4 primeros elementos de **a** y **x** al ejecutar el código 1.

Elemento	Tipo de acceso (R/W)	Acierto/Fallo
a[0]		
…		
…		
…		

c) Las siguientes figuras ilustran los vectores **a** y **x**. Deducir razonadamente qué parte de éstos quedan en la memoria caché después de ejecutar el código 1 si se utiliza una política de reemplazo LRU (*Least Recently Used*). Además, deducir la tasa de acierto de caché del código 2, si éste se ejecuta inmediatamente después del código 1.



CUESTIONARIO 1.2.2

Pregunta 1

La descomposición en campos de la dirección de memoria es:

a) 28 bits para la etiqueta, 2 para el desplazamiento de palabra y 2 para el desplazamiento de byte.
b) 29 bits para la etiqueta, 1 para el desplazamiento de palabra y 2 para el desplazamiento de byte.
c) 22 bits para la etiqueta, 8 para el desplazamiento de palabra y 2 para el desplazamiento de byte.
d) 27 bits para la etiqueta, 3 para el desplazamiento de palabra y 2 para el desplazamiento de byte.

Pregunta 2

¿Cuántas líneas tiene la caché de datos? (Responder en el cuadro destinado a tal efecto).

Pregunta 3

¿Cuál es la tasa de acierto conseguida tras ejecutar las 4 primeras iteraciones del código 1? (Responder en el cuadro destinado a tal efecto). Escribir el resultado redondeado a dos cifras decimales.

Pregunta 4

Para el código 1, cuando se accede a un elemento par del vector **a**, se obtiene siempre un:

a) Fallo.
b) Acierto.
c) Depende de si el índice del elemento es múltiplo de 2 ó de 4.
d) No se puede saber sin conocer los valores que contiene el vector.

Pregunta 5

Para el código 1, cuando se accede a un elemento impar del vector **x**, se obtiene siempre un:

a) Fallo.
b) Acierto.
c) Depende de si el índice del elemento es múltiplo de 2 ó de 4.
d) No se puede saber sin conocer los valores que contiene el vector.

Pregunta 6

Tras ejecutar el código 1 al completo, los elementos de los vectores **a** y **x** contenidos en la caché son:

a) El vector **a** al completo.

b) El vector **x** al completo.
c) Los elementos 0 a 255 tanto de **a** como de **x**.
d) Los elementos 256 a 511 tanto de **a** como de **x**.

Pregunta 7

¿Cuál es la tasa de acierto de caché del código 2, si éste se ejecuta inmediatamente después del código 1? (Responder en el cuadro destinado a tal efecto). Escribir el resultado redondeado a dos cifras decimales.

Pregunta 8

Tras ejecutar el código 2 al completo, los elementos de los vectores **a** y **x** contenidos en la caché son:

a) El vector **a** al completo.
b) El vector **x** al completo.
c) Los elementos 0 a 255 tanto de **a** como de **x**.
d) Los elementos 256 a 511 tanto de **a** como de **x**.

EJERCICIO 1.2.3: ARRAYS BIDIMENSIONALES

Primera parte: recorrido por columnas

Sea un computador con un procesador MIPS dotado de memorias caché separadas para instrucciones y datos. Analizaremos el comportamiento de la caché de datos ante la ejecución del programa codificado en el archivo **column-major.asm**, que recorre una matriz de enteros de 16 por 16 por columnas:

```
for (col = 0; col < 16; col++)
   for (row = 0; row < 16; row++)
      data[row][col] = value++;
```

a) La caché de datos tiene un tamaño de 128 bytes, y se gestiona mediante una política de ubicación por correspondencia directa. El tiempo de acceso cuando hay acierto es de 1 ciclo, y cuando hay fallo es de 5 * número de palabras del bloque.

   Ejecutar el código con diferentes configuraciones de memoria caché, y rellenar la siguiente tabla:

Tamaño de bloque (palabras)	Nº de bloques	Nº de aciertos	Nº de fallos	Tasa de fallo	Penalización de fallo	Tiempo medio de acceso a memoria
1	32					
2	16					
4	8					
8	4					

b) Repetir el paso anterior, suponiendo que la caché de datos se gestiona mediante una política de ubicación totalmente asociativa con política de reemplazamiento LRU.

Tamaño de bloque (palabras)	Nº de bloques	Nº de aciertos	Nº de fallos	Tasa de fallo	Penalización de fallo	Tiempo medio de acceso a memoria
1	32					
2	16					
4	8					
8	4					

c) Repetir el paso anterior, suponiendo que la caché de datos se gestiona mediante una política de ubicación asociativa por conjuntos de 2 vías, con política de reemplazamiento LRU.

Tamaño de bloque (palabras)	Nº de bloques	Nº de aciertos	Nº de fallos	Tasa de fallo	Penalización de fallo	Tiempo medio de acceso a memoria
1	32					
2	16					
4	8					
8	4					

Segunda parte: recorrido por filas

Repetir los pasos anteriores, utilizando el programa codificado en el archivo **row-major.asm**, que realiza un recorrido de la matriz por filas. El tamaño de caché se mantendrá en 128 bytes.

```
for (row = 0; row < 16; row++)
   for (col = 0; col < 16; col++)
      data[row][col] = value++;
```

a) Política de ubicación por correspondencia directa.

Tamaño de bloque (palabras)	Nº de bloques	Nº de aciertos	Nº de fallos	Tasa de fallo	Penalización de fallo	Tiempo medio de acceso a memoria
1	32					
2	16					
4	8					
8	4					

b) Política de ubicación totalmente asociativa y política de reemplazamiento LRU.

Tamaño de bloque (palabras)	Nº de bloques	Nº de aciertos	Nº de fallos	Tasa de fallo	Penalización de fallo	Tiempo medio de acceso a memoria
1	32					
2	16					
4	8					
8	4					

c) Política de ubicación asociativa por conjuntos de 2 vías, con política de reemplazamiento LRU.

Tamaño de bloque (palabras)	Nº de bloques	Nº de aciertos	Nº de fallos	Tasa de fallo	Penalización de fallo	Tiempo medio de acceso a memoria
1	32					
2	16					
4	8					
8	4					

Análisis

Explicar y comentar los resultados obtenidos.

CUESTIONARIO 1.2.3

Pregunta 1

En el recorrido por columnas con caché de correspondencia directa, ¿cuál es siempre el número total de aciertos, independientemente del tamaño de bloque? (Responder en el cuadro destinado a tal efecto).

Pregunta 2

En el recorrido por columnas con caché totalmente asociativa, ¿cuál es el único tamaño de bloque que consigue una tasa de acierto no nula?

a) 1 palabra por bloque.
b) 2 palabras por bloque.
c) 4 palabras por bloque.
d) 8 palabras por bloque.

Pregunta 3

En el recorrido por columnas con caché asociativa de 2 vías, ¿cuál es siempre el número total de aciertos, independientemente del tamaño de bloque? (Responder en el cuadro destinado a tal efecto).

Pregunta 4

En el recorrido por filas de la caché de correspondencia directa, ¿qué sucede al aumentar el tamaño de bloque?

a) Mejora la tasa de acierto.
b) Empeora la tasa de acierto.
c) Aumenta la tasa de fallo.
d) Permanece constante la tasa de penalización por fallo.

Pregunta 5

En el recorrido por filas de la caché de correspondencia directa, ¿qué sucede al aumentar el tamaño de bloque?

a) Disminuye el tiempo medio de acceso a memoria.
b) Aumenta el tiempo medio de acceso a memoria.
c) El tiempo medio de acceso a memoria aumenta o no dependiendo del tamaño de bloque.
d) Ninguna de las restantes respuestas es correcta.

Pregunta 6

¿Por qué se obtienen en general mejores tiempos medios de acceso a memoria cuando se recorre la matriz por filas?

a) No es verdad, se obtienen mejores resultados para el recorrido por columnas.

b) Es una casualidad para este ejercicio.
c) Porque los elementos de cada fila de la matriz se almacenan en memoria en posiciones consecutivas.
d) Porque los elementos de cada columna de la matriz se almacenan en memoria en posiciones consecutivas.

Pregunta 7

Indicar si la siguiente afirmación es verdadera o falsa: “En unos bucles anidados que recorren todos los elementos de un array, el orden en el que se colocan los bucles no tiene repercusión sobre el tiempo medio de acceso a memoria”.

Apéndice

Conceptos básicos sobre matrices

Matrices bidimensionales

Una matriz bidimensional es una colección de datos, todos ellos de igual tipo, organizados en forma rectangular por filas y columnas. A cada elemento se accede mediante sus coordenadas de fila y columna. Si empezamos a numerarlas a partir del 0, una matriz de 4 filas y 3 columnas llamada A se representará así:

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{02} \\ a_{20} & a_{21} & a_{02} \\ a_{30} & a_{31} & a_{02} \end{pmatrix}$$

En un programa, una variable de tipo matriz debe ser almacenada en memoria. Los elementos de las matrices se almacenan consecutivamente a partir de una dirección base. En la mayoría de los lenguajes de programación los elementos se almacenan por filas: los elementos de una misma fila se almacenan consecutivamente en memoria, y las sucesivas filas se almacenan una detrás de otra. Por tanto, una matriz **A** de dimensiones *numfil* x *numcol* se almacenaría así (para simplificar, supondremos que cada elemento ocupa 1 byte en memoria):

Dirección de memoria	**Contenido**
Dirección base de A: d(A)	$a_{0,0}$
d(A)+0*numcol+1	$a_{0,1}$
d(A)+ 0*numcol+2	$a_{0,2}$
	…
…	
d(A)+ 0*numcol+numcol-1	$a_{0,numcol-1}$
d(A)+ 1*numcol+0	$a_{1,0}$
d(A)+ 1*numcol+1	$a_{1,1}$
d(A)+ 1*numcol+2	$a_{1,2}$
	…
…	
d(A)+ 1*numcol+numcol-1	$a_{1,numcol-1}$
d(A)+ 2*numcol+0	$a_{2,0}$
	…
…	
d(A)+ i*numcol+j	$a_{i,j}$
	...
…	
d(A)+(numfil-1)*numcol+1	$a_{numfil-1,1}$
d(A)+ (numfil-1)*numcol+2	$a_{numfil-1,2}$
	…
…	
d(A)+ (numfil-1)*numcol+numcol-1	$a_{numfil-1,numcol-1}$

En lenguaje ensamblador también pueden manejarse matrices. Pero en MIPS no puede accederse directamente a un elemento de una matriz dando su fila y su columna, ya que la única forma posible de acceder a memoria es utilizando el direccionamiento indirecto a registro con desplazamiento:

desp($reg)

Para acceder a un elemento de posición *(i,j)*, lo más sencillo es cargar la dirección del elemento en el registro base y usar desplazamiento constante 0:

0($registro)

Por tanto, para acceder un elemento de posición *(i,j)*, será preciso calcular la dirección de dicho elemento y cargarla en un registro. Según la figura anterior, la dirección del elemento *(i,j)* de la matriz **A** viene dada por la expresión ***d(A)+ i*maxcol+j*** , donde:

- *d(A)* es la dirección base de la matriz,
- *maxcol* es el número de columnas de la matriz,
- *i* es la coordenada de fila del elemento buscado, y
- *j* es la coordenada de columna del elemento buscado.

Ahora, suponiendo que la dirección base de la matriz reside en el registro **$s0**, la coordenada *i* en el registro **$s1**, la coordenada *j* en el registro **$s2** y el número de columnas en el registro **$s3**, cargaremos el elemento **A(i,j)=a_{ij}** de la matriz en el registro **$s4** mediante la siguiente secuencia de código (supongamos que los elementos de la matriz son enteros positivos de tamaño byte):

```
# Calcular la distancia de A(i,j) al inicio de la matriz
            mult  $s1,$s3
            mflo  $t0
            addu  $t1,$t0,$s2
# Sumar distancia a la dirección base de la matriz
            addu  $t2,$s0,$t1
# Leer el elemento A(i,j)
            lbu   $s4,0($t2)
```

Si cada elemento ocupa más de un byte, es preciso escalar los accesos, multiplicando la distancia del elemento al origen de la matriz por el tamaño de cada elemento en bytes antes de sumárselo a la dirección base.

El archivo **ejemplo_matriz.asm** contiene ejemplos de acceso a una matriz de datos.