ARCO Pràctica 3 - Q2 21/22

Ixent Cornella

Ejercicio 1 y 2: Implementar en MIPS la caché directa a partir del fichero ya dado en Atenea.

Este ejercicio estará adjunto en la tarea de Atenea como “ejercicio2.s”.

Este es el resultado:

```
 .data

MemCacheDirect:

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0

        .word 0 0
```

```
        .word 0 0

X: .word 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Y: .word -2 0 3 6 -2 12 -3 14 5 6 -7 8 19 -20 1 0 -3 3 32 -3 34 -5 6 -7
Z: .space 96
.text
.align 2
.globl main
main:
        addi $sp, $sp, -4
        sw $ra, 0($sp)
        la $s0, X
        la $s1, Y
        la $s2, Z
        addi $s3, $zero, 0    # contador iteraciones
        addi $s4, $zero, 0    # contador hits
        addi $s5, $zero, 24   # contador del for
        addi $s6, $zero, 0    # representara el offset de la @. (4, 8, 12, ...)
for:    beq $s5, $zero, fi

        add $t0, $s0, $s6     # @X[i]
        lw $t1, 0($t0)        # X[i]
```

```
add $t2, $s1, $s6     # @Y[i]
lw $t3, 0($t2)        # Y[i]
add $t3, $t3, $t1     # Y[i] + X[i]
add $t4, $s2, $s6     # @Z[i]
sw $t3, 0($t4)                # Z[i] = Y[i] + X[i]

addi $sp, $sp, -8
sw $t2, 0($sp)
sw $t4, 4($sp)
addi $a0, $t0, 0
jal callCache
add $s4, $s4, $v0     # si es hit, s4 + 1
addi $s3, $s3, 1      # contador total
lw $t2, 0($sp)
addi $sp, $sp, 4

addi $a0, $t2, 0
jal callCache
add $s4, $s4, $v0     # si es hit, s4 + 1
addi $s3, $s3, 1      # contador total
lw $t4, 0($sp)
```

```
        addi $sp, $sp, 4


        addi $a0, $t4, 0
        jal callCache
        add $s4, $s4, $v0    # si es hit, s4 + 1
        addi $s3, $s3, 1     # contador total


        addi $s6, $s6, 4
        addi $s5, $s5, -1
        j for
fi:     sub $s7, $s3, $s4    # contador de misses en s7
        lw $ra, 0($sp)
        addi $sp, $sp, 4
        jr $ra
.end main


callCache:
        srl $t0, $a0, 4      # valid
        andi $t1, $t0, 0x0f  # line_cache
        srl $t2, $a0, 8              # tag
        la $t3, MemCacheDirect
```

```
        addi $t4, $zero, 8
        mul $t5, $t1, $t4       # en $t5 tenemos el la posicion mem[line_cache], falta sumar
        add $t5, $t5, $t3       # Ahora en $t5 tenemos @mem[line_cache]
        lw $t6, 0($t5)
        lw $t7, 4($t5)          # Tenemos en $t6 y $t7 valid y tag, respectivamente.
        andi $t4, $t6, 1        # VALID = 1?
        beq $t4, $zero, else
        bne $t7, $t2, else      # M[line].TAG = tag ?
        addi $v0, $zero, 1
        jr $ra
else:
        sw $t2, 4($t5)                  # Guardamos TAG
        addi $t4, $zero, 1
        sw $t4, 0($t5)                  # Guardamos VALID
        addi $v0, $zero, 0
        jr $ra
```

Ejercicio 3

a) ¿En qué dirección de memoria empieza la estructura MemCacheDirect?

Según QTSpim, MemCacheDirect empieza en 0x10010000.

b) ¿Cuántos bytes ocupa la estructura MemCacheDirect?

MemCacheDirect ocupa 128 bytes, lo calculamos en el informe anterior.

c) ¿En qué direcciones de memoria están los vectores X, Y, Z?

Es lógico pensar que X estará a MemCacheDirect + 128 bytes, Y estará a X + (tamaño de X), y lo mismo con Y. Obtenemos pues:

X: 0x10010080 (0x80 en hexa es 128, por lo que la asunción anterior está bien)

Y: 0x100100e0

Z: 0x10010140

d) ¿Cuándo se produce el primer acierto de cache?

El primer acierto se da cuando ya la cache ha sido inicializada, justo en la segunda iteración.

e) En la 10º iteración del bucle, a qué direcciones de los vectores X, Y y Z se acceder?

En la decima iteración se accederá a X + (9*4), Y + (9*4) e Z + (9*4). Quedaría así:

X: 0x10010080 + 0x24 = 0x100100a4 = X[9]

Y: 0x100100e0 + 0x24 = 0x10010104 = Y[9]

Z: 0x10010140 + 0x24 = 0x10010164 = Z[9]

e2) En esta misma 10ª iteración, cuando llamamos a callCache(@X[9]), en qué

posición de memoria está MemCacheDirect[line_cache(@X[9]).VALID y

MemCacheDirect[line_cache(@X[9]).TAG?

MemCacheDirect[line_cache] equivale a MemCacheDirect[10], ya que @X[9] equivale a 0x100100a4, y line_cache son los 4 bits de menor peso una vez ya quitamos los 4 primeros bits, es decir, los 4 bits de menor peso de 0x100100a. Estamos accediendo a la décima entrada de la caché.

Por tanto, la dirección de memoria de VALID y TAG será la posición de memoria de MemCacheDirect = 0x10010000 + 10*8 = **0x10010050** para el VALID y **0x10010054** para TAG.

e3) ¿Qué dirección de línea de cache es para line_cache(@X[9])?

La direcció de línea de cache es 10, como se calcula en el apartado anterior.

e4) ¿Se produce fallo o acierto al acceder a line_cache(@X[9])?

Se produce acierto, lo cual tiene sentido teniendo en cuenta el patrón:

1era iteración MISS

2nda, 3era y 4ta HIT

5 MISS

6, 7, 8 HIT

9 MISS

10 HIT.

f) ¿Cuántos fallos y aciertos en total se producen? ¿Cuántos de X, cuántos de Y y cuántos de Z?

Se producen 18 misses y 54 hits. Estos se dividen de forma equitativa, es decir, 18/3 misses y 54/3 misses por X, Y e Z.

X: 6 Misses, 18 Hits

Y: 6 Misses, 18 Hits

Z: 6 Misses, 18 Hits

g) Puedes observar algún patrón de comportamiento en estos accesos a

memoria.

Sí, hay un patrón. En la primera iteración siempre habrán misses, pero después, observamos un claro patrón: 3 Hits, y luego 1 Miss. Esto se debe a que accedemos a los 4 bits de menor peso de line_cache, que iran de 4 en 4, hasta llegar a 16, pero será módulo 16, por tanto 0, siendo esto un Miss, y luego los 3 Hits, etc.

Ejercicio 4

Suponemos que los aciertos en cache tienen una duración de 1 ciclo en caso de lectura y los fallos tanto en lectura como en escritura son de 10 ciclos. En el caso de escritura aunque acierte tarda 10 ciclos pues asumimos que si acierta escribe a la vez en memoria cache y memoria principal. Teniendo en cuenta los datos que has obtenido de la simulación: aciertos y fallos X (lecturas), y fallos Y (lecturas) y aciertos y fallos Z (escrituras). ¿Cuál sería el tiempo en realizar todos estos accesos?

Tenemos que se realizan 2 lecturas y una escritura por iteración, y son 24 iteraciones, por lo que tendremos 48 lecturas y 24 escrituras.

De estas 48 lecturas, hay 6 + 6 Misses y 18 + 18 Hits (calculado anteriormente), por lo que los ciclos de lectura son: 12*10 + 36 = 156 ciclos.

Por otro lado, las escrituras tardan siempre 10 ciclos, y hemos visto anteriormente que hay 6 misses y 18 hits, por tanto habrá: (6+18)*10 = 240 ciclos.

Sumando los resultados obtenemos que el tiempo en realizar todos estos accesos es de 240 + 156 = **396 ciclos.**