Arquitectura del Procesador II

Trabajo Práctico U3.2 **Jerarquía de Memoria**

Ejercicio 1: (Correspondencia Memoria-Cache) Considere una memoria de 32 bloques (identificados de 0 a 31) y una cache de 8 bloques (identificados de 0 a 7).
Resuelva los siguientes correspondencia directa ¿Qué bloques de memoria compiten por el bloque 2 de cache?.

A) ¿Cuáles son las entradas de la cache donde se puede alojar el bloque de memoria 31, si la asociatividad son conjuntos de 4 vías?

B) Considere la siguiente secuencia de referencias desde una CPU a bloques de memoria, comenzando con una cache vacía.

Orden de referencia	1	2	3	4	5	6	7	8
Bloque referenciado	0	15	18	5	1	13	15	26

¿En qué referencia ocurre la primer diferencia, si la cache es de correspondencia directa vs. conjunto asociativo de 4-vías?

D) Suponga que la cache es de correspondencia directa y está equipada con una entrada adicional, llamada V, para mantener a un bloque que debe abandonar la cache (víctima). Considere la siguiente secuencia de referencias a bloques de memoria desde la CPU (una R indica lectura o load, una W indica escritura)

Orden de referencia	1	2	3	4	5	6	7	8	9	10	11	12
Tipo de referencia	R	R	W	R	R	W	R	R	W	R	R	W
Bloque referenciado	0	15	18	5	1	13	15	26	6	8	15	3

Analice la secuencia y diga si algún bloque que estuvo en V debió ser sacado también de ese lugar.

Ejercicio 2: ¿Cuáles son las diferencias entre una política write-allocate y una política no-write-allocate en una cache?

Ejercicio 3:¿Puede ocurrir que una cahe de correspondencia directa tenga una tasa de hit mayor que una cache totalmente asociativa que usa el algoritmo LRU para el reemplazo de bloques ? (Misma secuencia de referencias y mismo tamaño de la cache)

Impacto de la cache en el desempeño

En los archivos adjuntos se muestran los diagramas[1] de tres tipos de caché:
(1) una caché de correspondencia directa con una palabra por línea,
(2) una memoria caché de correspondencia directa con cuatro palabras por línea y
(3) caché asociativa con conjuntos de dos vías con cuatro palabras por línea.
Los segmentos de código que se muestran a continuación dan dos posibles flujos de instrucciones, cada uno con un número variable de instrucciones. Vamos a analizar cómo cada una de estas caché se utilizan con cada una de las secuencias de instrucciones que se describen a continuación, con el fin de determinar cómo la caché afecta el desempeño cuando se ejecuta cada código.

Las cachés son todas del mismo tamaño, 128 Bytes o 32 palabras. Porque se consideran sólo caches de instrucciones, los dos bits menos significativos de cada dirección son siempre cero y no se utilizan en el tratamiento de bytes individuales dentro de las palabras (instrucciones) en la caché, como podría ser en el caso de una caché de datos.

[1] En estos diagramas se debe completar el tag y datos de los bloques de cache referenciados

Los caches tienen las siguientes características adicionales:

- ✳ Cada caché tiene un ciclo de reloj para un hit.
- ✳ Caché (1), la caché de mapeado directo con una palabra por línea, tiene 5 ciclos de reloj por un fallo de caché.
- ✳ Caché (2), la caché correspondencia directa con cuatro palabras por línea, tiene 7 ciclos de reloj por un fallo de caché.
- ✳ Caché (3), la cache de conjunto asociativo de dos vías, tiene 7 ciclos de reloj por un fallo de caché.
- ✳ El ciclo de reloj de la CPU con la caché asociativa por conjunto es 10% más largo que el ciclo de reloj para las dos cachés de correspondencia directa.

Problema 1: Comparación de Caches correspondencia directa

Considere la posibilidad de caché (1) con correspondencia directa y contiene 32 líneas de una palabra cada una. Cinco bits (bits 2-6) de la dirección de la instrucción se utilizan para indexar la línea de caché. Los 25 bits restantes se almacenan en el campo de la etiqueta de la memoria caché.

Supongamos que segmento de código 1 tiene k = 32 instrucciones en el bucle (las últimas tres instrucciones en el bucle entonces tendrían direcciones dadas por:

```
1001 1000 1111 0000 0000 1100 1101 1000    <instruction 30>
1001 1000 1111 0000 0000 1100 1101 1100    addi $s5, $s5, 1
1001 1000 1111 0000 0000 1100 1110 0000    bne $s5, $s6, loop
```

1) Utilice el diagrama de correspondencia directa con una sola palabra por línea de caché, para mostrar el contenido de la memoria caché y los valores de los campos de la etiqueta después de la primera iteración del bucle. Calcular el tiempo (en ciclos de reloj) para el bucle para completar 1001 iteraciones. Recuerde que debe incluir las instrucciones anteriores al loop en sus cálculos. (Nota: no se olvide de los fallos compulsivos de caché durante la primera ejecución del ciclo.)

2) Considere lo que sucede cuando se agrega una instrucción al cuerpo del bucle en el segmento de código 1, de modo que k = 33. Calcular el tiempo (en ciclos de reloj) para que el bucle complete 1001 iteraciones. A medida que el tamaño del bucle se incrementa una instrucción a la vez, ¿cuánto aumenta el tiempo de ejecución del bucle?
3) Haga la parte (1), pero utilizando caché (2), de correspondencia directa con cuatro palabras por línea. Recuerde, en un fallo de caché toda la línea de caché se sustituye!
4) Haga parte (2), pero utilizando caché (2).

Problema 2: Comparación de correspondencia directa vs. asociativa por conjunto

El segmento de código 2 se utiliza para este problema y contiene un bucle y una llamada de subrutina.
1) Usar la caché (2), de correspondencia directa con cuatro palabras por línea, y mostrar el contenido de la memoria caché después de la primera iteración del bucle. Calcular el tiempo (en ciclos de reloj) para que el bucle complete 1001 iteraciones.
2) Usar caché (3), asociativa con conjunto de dos vías, y mostrar el contenido de la memoria caché después de la primera iteración del bucle. Calcular el tiempo (en ciclos de reloj) para que el bucle complete 1001 iteraciones.
3) Recordar que la caché asociativa con conjunto de dos vías utilizada en la parte (2) necesita un ciclo de reloj que es 10% más largo que el ciclo de reloj para la caché de mapeado directo. Teniendo esto en cuenta, ¿cuánto más rápido o más lento es el código de caché (3), de conjunto asociativo, que con la caché (2), correspondencia directa?

Segmento de Código 1

Address	Instruction	Comment
1001 1000 1111 0000 0000 1100 0101 1100	addi $s6, $0, 1001	# initialize number of iterations
1001 1000 1111 0000 0000 1100 0110 0000	add $s5, $0, $0	# initialize loop counter
	loop:	
1001 1000 1111 0000 0000 1100 0110 0100	<instruction 1>	# beginning of loop body,
1001 1000 1111 0000 0000 1100 0110 1000	<instruction 2>	# which has a total
1001 1000 1111 0000 0000 1100 0110 1100	<instruction 3>	# of k instructions
...	...	
1001 1000 1111 0000 0000 1100 0101 1100+4k	addi $s5, $s5, 1	# instruction k-1
1001 1000 1111 0000 0000 1100 0110 0000+4k	bne $s5, $s6, loop	# instruction k
		# end of loop

Segmento de Codigo 2

Address	Instruction	Comment
1001 1000 1111 0000 0000 1100 0101 0100	add $s4, $0, $0	# initialize total
1001 1000 1111 0000 0000 1100 0101 1000	addi $s6, $0, 1001	# initialize number of iterations
1001 1000 1111 0000 0000 1100 0101 1100	add $s5, $0, $0	# initialize loop counter
	loop:	
1001 1000 1111 0000 0000 1100 0110 0000	add $a0, $s0, $0	# first parameter
1001 1000 1111 0000 0000 1100 0110 0100	add $a1, $s1, $0	# second parameter
1001 1000 1111 0000 0000 1100 0110 1000	add $a2, $s2, $0	# third parameter
1001 1000 1111 0000 0000 1100 0110 1100	add $a3, $s3, $0	# fourth parameter
1001 1000 1111 0000 0000 1100 0111 0000	jal function	# function call
1001 1000 1111 0000 0000 1100 0111 0100	add $s4, $s4, $v0	# add result to total
1001 1000 1111 0000 0000 1100 0111 1000	<instruction 7>	# remainder of loop
1001 1000 1111 0000 0000 1100 0111 1100	<instruction 8>	# which has a total
1001 1000 1111 0000 0000 1100 1000 0000	<instruction 9>	# of 16 instructions
...	...	
1001 1000 1111 0000 0000 1100 1001 0100	<instruction 14>	
1001 1000 1111 0000 0000 1100 1001 1000	addi $s5, $s5, 1	# instruction 15
1001 1000 1111 0000 0000 1100 1001 1100	bne $s5, $s6, loop	# instruction 16
		# end of loop
...	...	
	function:	
1001 1000 1111 0000 0000 1111 0110 1000	addi $sp, $sp, -16	# save state
1001 1000 1111 0000 0000 1111 0110 1100	sw $s0, 0($sp)	# instruction B
1001 1000 1111 0000 0000 1111 0111 0000	sw $s1, 4($sp)	# instruction C
1001 1000 1111 0000 0000 1111 0111 0100	sw $s2, 8($sp)	# instruction D
1001 1000 1111 0000 0000 1111 0111 1000	sw $ra, 12($sp)	# instruction E
1001 1000 1111 0000 0000 1111 0111 1100	<instruction F>	#body of subroutine
1001 1000 1111 0000 0000 1111 1000 0000	<instruction G>	
1001 1000 1111 0000 0000 1111 1000 0100	<instruction H>	
1001 1000 1111 0000 0000 1111 1000 1000	<instruction I>	
1001 1000 1111 0000 0000 1111 1000 1100	lw $s0, 0($sp)	# restore state
1001 1000 1111 0000 0000 1111 1001 0000	lw $s1, 4($sp)	# instruction K
1001 1000 1111 0000 0000 1111 1001 0100	lw $s2, 8($sp)	# instruction L
1001 1000 1111 0000 0000 1111 1001 1000	lw $ra, 12($sp)	# instruction M
1001 1000 1111 0000 0000 1111 1001 1100	addi $sp, $sp, 16	# instruction N
1001 1000 1111 0000 0000 1111 1010 0000	jr $ra	#return