Taller de Caché

Organización del Computador 1

Primer Cuatrimestre 2019 - Turno Mañana

Ejercicio 1 - Seguimiento de Caché

NOTA: completar la tabla manualmente, sin utilizar el simulador. Una vez completa, verificar los resultados con el mismo.

Caché de correspondencia directa

Dirección	Tag	Linea (*bits*/decimal)	Indice	Direcciones de la línea	Hit/Miss
0x0009					
0x001D					
0x000A					
0x0101					
0x0113					
0x000A					
0x001E					
0x0102					
0x0114					

Caché completamente asociativa

Dirección	Tag	Indice	# Linea	Direcciones de la línea	Hit/Miss
0x0009					
0x001D					
0x000A					
0x0101					
0x0113					
0x000A					
0x001E					
0x0102					
0x0114					

a) ¿En qué casos funciona mejor una memoria completamente asociativa frente a una de correspondencia directa? Dé un ejemplo.

b) ¿Qué pasa si sólo uso caché para los datos? ¿Y si sólo la uso para el código?

Ejercicio 2 - Políticas de desalojo

a) Medir el *hit rate* que se produce para ambos códigos, con las políticas FIFO, RANDOM y LRU.

- Iguales:
  - FIFO:
  - RANDOM:
  - LRU:
- Mix:
  - FIFO:
  - RANDOM:
  - LRU:

b) Explique la diferencia de performance de la cache encontrada entre ambos códigos, independientemente de la política utilizada.

c) Explique cuál es el beneficio que obtiene entre utilizar FIFO y LRU, tras analizar el hit rate en ambos casos.

Ejercicio 3 - Análisis de Caché

- A partir del resultado que se observa, ¿se puede decir que a mayor cantidad de lineas, mejor funcionamiento de la cache? ¿Para verificar su hipótesis, que pasa si tenemos más de líneas (**nota:** al menos debe quedar un bit para índice)? Explique qué sucede.

Ejercicio 4 - Seguimiento de código (opcional)

`mov r0`		**1**	`0x0003`		**H**	`mov r1`		**H**	`vector`		**H**
000	1	00	000	1	01	000	1	10	000	1	11
`add`			`suma`			`r1`			`[suma]`		
001	0	00	001	0	01	001	0	01	000	0	11
`[r1]`			`add r1`			`0x0001`			`sub r0`		
000	0	00	001	0	11	001	1	00	001	1	01
`0x0001`			`jne ciclo`			`add`			`suma`		
001	1	10	001	1	11	001	0	00	001	0	01
`r1`			`[suma]`			`[r1]`			`add r1`		
001	0	10	000	0	11	000	0	01	001	0	11
`0x0001`			`sub r0`			`0x001`			`jne ciclo`		
001	1	00	001	1	01	001	1	10	001	1	11

Seguimiento

1. Se carga TAG **000** en línea **1**
2. Se carga TAG ..... en línea ...
3. Se carga TAG ..... en línea ...
4. Se carga TAG ..... en línea ...

5. Se carga TAG ..... en línea ...
6. Se carga TAG ..... en línea ...
7. Se carga TAG ..... en línea ...

mov r0	**1**	0x0003	**H**	mov r1	**H**	vector	**H**
0001	00	0001	01	0001	10	0001	11
add		suma		r1		[suma]	
0010	00	0010	01	0010	01	0000	11
[r1]		add r1		0x0001		sub r0	
0000	00	0010	11	0011	00	0011	01
0x0001		jne ciclo		add		suma	
0011	10	0011	11	0010	00	0010	01
r1		[suma]		[r1]		add r1	
0010	10	0000	11	0000	01	0010	11
0x0001		sub r0		0x001		jne ciclo	
0011	00	0011	01	0011	10	0011	11

Seguimiento

1. Se carga TAG **0001** en línea **0**
2. Se carga TAG ..... en línea ...
3. Se carga TAG ..... en línea ...
4. Se carga TAG ..... en línea ...
5. Se carga TAG ..... en línea ...
6. Se carga TAG ..... en línea ...
7. Se carga TAG ..... en línea ...

a) Calcule el hit rate para cada memoria

b) Analice los comportamientos de ambos tipos de memorias para este programa. ¿Qué hubiera ocurrido con el hit rate entre ambas memorias si leyera un vector de más posiciones?

c) ¿Qué tiene de particular la remoción de las líneas en la caché asociativa? ¿Se te ocurre alguna manera simple de implementar (hardware) la política FIFO si y solo si no hubiera líneas inválidas?

Corrección

Para uso de los docentes

1	2	3