CS2100 Computer Organisation
Tutorial #11: Cache

(Week 13: 14 – 18 April 2025)

Discussion Question

D1. [CS2100 AY2007/8 Semester 2 Exam Question]
A machine with a word size of 16 bits and address width of 32 bits has a **direct-mapped cache** with 16 blocks and a block size of 2 words, initially empty.

(a) Given a sequence of memory references as shown below, where each reference is given as a byte address in both decimal and hexadecimal forms, indicate whether the reference is a hit (H) or a miss (M).

Memory address		Hit (H) or Miss (M)?	(For reference)
(in decimal)	(in hexadecimal)		
4	0x4	M	0000 … 0000 0100
92	0x5C		0000 … 0101 1100
7	0x7		0000 … 0000 0111
146	0x92		0000 … 1001 0010
30	0x1E		0000 … 0001 1110
95	0x5F		0000 … 0101 1111
176	0xB0		0000 … 1011 0000
93	0x5D		0000 … 0101 1101
145	0x91		0000 … 1001 0001
264	0x108		0000 … 1 0000 1000
6	0x6		0000 … 0000 0110

(b) Given the above sequence of memory references, fill in the final contents of the cache. Use the notation M[*i*] to denote the word starting at memory address *i*, where *i* is in hexadecimal. If a block is replaced, cross out the content in the cache and write the new content over it.

Index	Tag value	Word 0	Word 1
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

Tutorial Questions

1. Here is a series of address references in decimal: 4, 16, 32, 20, 80, 68, 76, 224, 36, 44, 16, 172, 20, 24, 36, and 68 in a MIPS machine. Assuming a **direct-mapped cache** with 16 one-word blocks that is initially empty, label each address reference as a hit or miss and show the content of the cache.

   You may write the data word starting at memory address X as M[X]. (For example, data word starting at memory address 12 is written as M[12]. This implies that the word includes the 4 bytes of data at addresses 12, 13, 14 and 15.) You may write the tag values as decimal numbers. If a block is replaced in the cache, cross out the corresponding content in the cache, and write the new content over it.

2. Use the series of references given in question 1 above: 4, 16, 32, 20, 80, 68, 76, 224, 36, 44, 16, 172, 20, 24, 36, and 68 in a MIPS machine. Assuming a **two-way set-associative cache** with two-word blocks and a total size of 16 words that is initially empty, label each address reference as a hit or miss and show the content of the cache. Assume **LRU** replacement policy.

   You may write the data word starting at memory address X as M[X]. (For example, data word starting at memory address 12 is written as M[12]. This implies that the word includes the 4 bytes of data at addresses 12, 13, 14 and 15.) You may write the tag values as decimal numbers. If a block is replaced in the cache, cross out the corresponding content in the cache, and write the new content over it.

3. Although we use only data memory as example in the cache lecture, the principle covered is equally applicable to the instruction memory. This question takes a look at both the instruction cache and data cache.

   The code below is from Tutorial 3 Question 1 (***palindrome checking***) with the following variable mappings:

   low → $s0, high→ $s1, matched → $s3, base of string[]→ $s4, size → $s5

#	Code	Comment
i0	`[some instruction]`	
i1	`addi $s0, $zero, 0`	`# low = 0`
i2	`addi $s1, $s5, -1`	`# high = size-1`
i3	`addi $s3, $zero, 1`	`# matched = 1`
	`loop:`	
i4	`slt $t0, $s0, $s1`	`# (low < high)?`
i5	`beq $t0, $zero, exit`	`# exit if (low >= high)`
i6	`beq $s3, $zero, exit`	`# exit if (matched == 0)`
i7	`add $t1, $s4, $s0`	`# address of string[low]`
i8	`lb $t2, 0($t1)`	`# t2 = string[low]`
i9	`addi $t3, $s4, $s1`	`# address of string[high]`
i10	`lb $t4, 0($t3)`	`# t4 = string[high]`
i11	`beq $t2, $t4, else`	
i12	`addi $s3, $zero, 0`	`# matched = 0`
i13	`j endW`	`# can be "j loop"`
	`else:`	
i14	`addi $s0, $s0, 1`	`# low++`
i15	`addi $s1, $s1, -1`	`# high–`
	`endW:`	
i16	`j loop`	`# end of while`
	`exit:`	
i17	`[some instruction]`	

Parts (a) to (d) assume that instruction i0 is stored at memory address 0x0.

(a) Instruction cache: **Direct mapped with 2 blocks of 16 bytes each** (i.e. each block can hold 4 consecutive instructions).

   Starting with an empty cache, the fetching of instruction i1 will cause a cache miss. After the cache miss is resolved, we now have the following instructions in the instruction cache:

Instruction Cache Block 0	[i0, **i1**, **i2**, **i3**]
Instruction Cache Block 1	[empty]

   Fetching of i2 and i3 are all cache hits as they can be found in the cache.

   Assuming the string being checked is a palindrome. Show the instruction cache block content **at the end of the 1st iteration (i.e. up to instruction i16).**

(b) If the loop is executed for a total of 10 iterations, what is the total number of cache hits (i.e. after the 10th "j loop" is fetched)?

(c) Suppose we change the instruction cache to:

- **Direct mapped with 4 blocks of 8 bytes each** (i.e. each block can hold 2 consecutive instructions).

Assuming the string being checked is a palindrome. Show the instruction cache block content **at the end of the 1st iteration (i.e. up to instruction i16).**

Instruction Cache Block 0	
Instruction Cache Block 1	
Instruction Cache Block 2	
Instruction Cache Block 3	

(d) If the loop is executed for a total of 10 iterations, what is the total number of cache hits (i.e. after the 10th "j loop" is fetched)?

Let us now turn to the study of **data cache**. We will assume the following scenario for parts (e) to (g):

- The string being checked is **64-character long**. The first character is located at location **0x1000**.
- The string is a palindrome (i.e. it will go through 32 iterations of the code).

(e) Given a **direct mapped data cache with 2 cache blocks, each block is 8 bytes**, what is the final content of the data cache at the end of the code execution (after the code failed the beq at i5)? Use **s[X..Y]** to indicate the data **string[X]** to **string[Y].**

Data Cache Block #0	
Data Cache Block #1	

(f) What is the hit rate of (e)? Give your answer in a fraction or a percentage correct to two decimal places.

(g) Suppose the string is now **72-character long**, the first character is still located at location **0x1000** and the string is still a palindrome, what is the hit rate at the end of the execution?