**CS2100 Computer Organisation**

**Tutorial #11: Cache**

**Answers**

***LumiNUS 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 | **M** | 0000 … 0101 1100 |
| 7 | 0x7 | **H** | 0000 … 0000 0111 |
| 146 | 0x92 | **M** | 0000 … 1001 0010 |
| 30 | 0x1E | **M** | 0000 … 0001 1110 |
| 95 | 0x5F | **M** | 0000 … 0101 1111 |
| 176 | 0xB0 | **M** | 0000 … 1011 0000 |
| 93 | 0x5D | **H** | 0000 … 0101 1101 |
| 145 | 0x91 | **H** | 0000 … 1001 0001 |
| 264 | 0x108 | **M** | 0000 … 1 0000 1000 |
| 6 | 0x6 | **H** | 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 | **0** | **M[4]** | **M[6]** |
| 2 | **4** | **M[108]** | **M[10A]** |
| 3 |  |  |  |
| 4 | **2** | **M[90]** | **M[92]** |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 | **~~1~~ ~~0~~ 1** | **~~M[5C]~~ ~~M[1C]~~ M[5C]** | **~~M[5E]~~ ~~M[1E]~~ M[5E]** |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |
| 12 | **2** | **M[B0]** | **M[B2]** |
| 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.

***Answer:***

Since this is a MIPS machine, a word consists of 4 bytes.

Should first work out the tag, index, and offset fields:

26 bits

4 bits

2

Tag

Index

Offset

4: 00…00 0001 00 🡨 Miss

|  |  |  |  |
| --- | --- | --- | --- |
| **Cache block** | **Valid bit** | **Tag** | **Word** |
| 0 | 0 |  |  |
| 1 | ~~0~~ 1 | ~~0~~ 1 | ~~M[4]~~ M[68] |
| 2 | 0 |  |  |
| 3 | ~~0~~ 1 | 1 | M[76] |
| 4 | ~~0~~ 1 | ~~0~~ ~~1~~ 0 | ~~M[16]~~ ~~M[80]~~ M[16] |
| 5 | ~~0~~ 1 | 0 | M[20] |
| 6 | ~~0~~ 1 | 0 | M[24] |
| 7 | 0 |  |  |
| 8 | ~~0~~ 1 | ~~0~~ 3 | ~~M[32]~~ M[224] |
| 9 | ~~0~~ 1 | 0 | M[36] |
| 10 | 0 |  |  |
| 11 | ~~0~~ 1 | ~~0~~ 2 | ~~M[44]~~ M[172] |
| 12 | 0 |  |  |
| 13 | 0 |  |  |
| 14 | 0 |  |  |
| 15 | 0 |  |  |

16: 00…00 0100 00 🡨 Miss

32: 00…00 1000 00 🡨 Miss

20: 00…00 0101 00 🡨 Miss

80: 00…01 0100 00 🡨 Miss

68: 00…01 0001 00 🡨 Miss

76: 00…01 0011 00 🡨 Miss

224: 00…11 1000 00 🡨 Miss

36: 00…00 1001 00 🡨 Miss

44: 00…00 1011 00 🡨 Miss

16: 00…00 0100 00 🡨 Miss

172: 00…10 1011 00 🡨 Miss

20: 00…00 0101 00 🡨 Hit

24: 00…00 0110 00 🡨 Miss

36: 00…00 1001 00 🡨 Hit

68: 00…01 0001 00 🡨 Hit

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.

***Answer:***

Since this is a MIPS machine, a word consists of 4 bytes or 32 bits.

Should first work out the tag, set index, and offset fields:

27 bits

2 bits

3

Tag

Set Index

Offset

4: 00…000 00 100 🡨 Miss

16: 00…000 10 000 🡨 Miss

32: 00…001 00 000 🡨 Miss

20: 00…000 10 100 🡨 Hit

80: 00…010 10 000 🡨 Miss

68: 00…010 00 100 🡨 Miss

76: 00…010 01 100 🡨 Miss

224: 00…111 00 000 🡨 Miss

36: 00…001 00 100 🡨 Miss

44: 00…001 01 100 🡨 Miss

16: 00…000 10 000 🡨 Hit

172: 00…101 01 100 🡨 Miss

20: 00…000 10 100 🡨 Hit

24: 00…000 11 000 🡨 Miss

36: 00…001 00 100 🡨 Hit

68: 00…010 00 100 🡨 Miss

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Cache set** | **Valid bit** | **Tag** | **Word0** | **Word1** | **Valid bit** | **Tag** | **Word0** | **Word1** |
| 0 | ~~0~~ 1 | ~~0~~  ~~2~~  1 | ~~M[0]~~  ~~M[64]~~  M[32] | ~~M[4]~~  ~~M[68]~~  M[36] | ~~0~~ 1 | ~~1~~  ~~7~~  2 | ~~M[32]~~  ~~M[224]~~  M[64] | ~~M[36]~~  ~~M[228]~~  M[68] |
| 1 | ~~0~~ 1 | ~~2~~  5 | ~~M[72]~~  M[168] | ~~M[76]~~  M[172] | ~~0~~ 1 | 1 | M[40] | M[44] |
| 2 | ~~0~~ 1 | 0 | M[16] | M[20] | ~~0~~ 1 | 2 | M[80] | M[84] |
| 3 | ~~0~~ 1 | 0 | M[24] | M[28] | 0 |  |  |  |

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 8 Question 1 (***palindrome checking***) with the following variable mappings:

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

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

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

1. 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).**

***Answer:***

|  |  |
| --- | --- |
| Instruction Cache Block 0 | **[i16, ……..]** |
| Instruction Cache Block 1 | **[i12, i13, i14, i15]** |

Working: Instructions executed = i1 to i11, i14 to i16

|  |  |
| --- | --- |
| Block #0, Cache index = 0 | [i0, i1, i2, i3] |
| Block #1, Cache index = 1 | [i4, i5, i6, i7] |
| Block #2, Cache index = 0 | [i8, i9, i10, i11] |
| Block #3, Cache index = 1 | [i12, i13, i14, i15] |
| Block #4, Cache index = 0 | [i16, other….] |

1. 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)?

***Answer:***

Working (1st Iteration):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i1 | i2 | i3 | i4 | i5 | i6 | i7 | i8 | i9 | i10 | i11 | i14 | i15 | i16 |
| M | H | H | M | H | H | H | M | H | H | H | M | H | M |

Working (2nd iteration onward):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i4 | i5 | i6 | i7 | i8 | i9 | i10 | i11 | i14 | i15 | i16 |
| M | H | H | H | M | H | H | H | M | H | M |

Total hits = 9 (1st iteration) + 7×9 (remaining 9 iterations) = **72**

1. 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).**

***Answer:***

|  |  |
| --- | --- |
| Instruction Cache Block 0 | **[i16, …]** |
| Instruction Cache Block 1 | **[i10, i11]** |
| Instruction Cache Block 2 | **[i4, i5]** |
| Instruction Cache Block 3 | **[i14, i15]** |

Working:

First, find out the block information for the full code:

|  |  |
| --- | --- |
| Block #0, Cache index = 0 | [i0, i1] |
| Block #1, Cache index = 1 | [i2, i3] |
| Block #2, Cache index = 2 | [i4, i5] |
| Block #3, Cache index = 3 | [i6, i7] |
| Block #4, Cache index = 0 | [i8, i9] |
| Block #5, Cache index = 1 | [i10, i11] |
| Block #6, Cache index = 2 | [i12, i13] |
| Block #7, Cache index = 3 | [i14, i15] |
| Block #8, Cache index = 0 | [i16, …] |

Second, use the execution pattern to find out what is accessed, since we execute i1 to i11 (Block #0 to Block #5) then i14 to i16 (Block #7 and Block #8), we get the final cache content as shown. You should note that Block #6 [i12, i13] is not accessed in this particular execution.

1. 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)?

***Answer:***

Working (1st Iteration):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i1 | i2 | i3 | i4 | i5 | i6 | i7 | i8 | i9 | i10 | i11 | i14 | i15 | i16 |
| M | M | H | M | H | M | H | M | H | M | H | M | H | M |

Working (2nd iteration onward):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i4 | i5 | i6 | i7 | i8 | i9 | i10 | i11 | i14 | i15 | i16 |
| H | H | M | H | M | H | H | H | M | H | M |

**Total hits** = 6 (1st iteration) + 7×9 (remaining 9 iterations) = **69**

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).

1. 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].**

***Answer:***

|  |  |
| --- | --- |
| Data Cache Block #0 | **s[32..39]** |
| Data Cache Block #1 | **s[24..31]** |

Access patterns = s[0], s[63], s[1], s[62], …, s[31], s[32]

Blocks information (blocks that can go into the same cache location are listed together):

|  |  |
| --- | --- |
| Cache index = 0 | s[0..7] [16..23] [32..39] [48..55] |
| Cache index = 1 | s[8..15] [24..31] [40..47] [56..63] |

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

***Answer:***

Observation: the access pattern nicely alternates between Block0-Block1 and Block1-Block0. So, in general, other than the first miss to bring in a block, the remaining 7 accesses on the block are all hits.

Hence, hit rate = **7/8** or **87.50%**

1. 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?

***Answer:***

Access patterns = s[0], s[71], s[1], s[70], …, s[35], s[36]

Blocks information (blocks that can go into the same cache location are listed together):

|  |  |
| --- | --- |
| Cache index = 0 | s[0..7] [16..23] [32..39] [48..55] [64..71] |
| Cache index = 1 | s[8..15] [24..31] [40..47] [56…63] |

Observation: the access pattern is either Block0-Block0 or Block1-Block1. So, every access is a miss, except the last block [32..39]! This is an example of ***cache thrashing*** (you can imagine the cache is “beaten up” pretty badly ☺).

Hence, hit rate = **7/72** (the last 7 accesses on block [32..39]) or **9.72%**