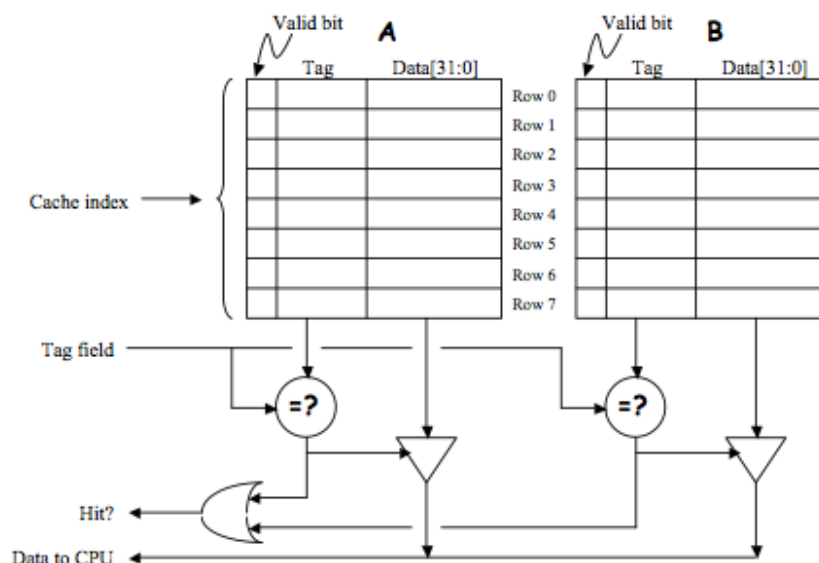


Video explanation of solution is provided below the problem.

Caches

10/10 points (ungraded)

Consider the diagram below for a 2-way set associative cache to be used with our Beta design. Each cache line holds a single 32-bit word of data along with its associated tag and valid bit (0 when the cache line is invalid, 1 when the cache line is valid).



1. The Beta produces 32-bit byte addresses, $A[31:0]$. To ensure the best cache performance, which address bits should be used for the cache index? For the tag field?

Address bits used for cache index: A[

4

✓ Answer: 4

:

2

✓ Answer: 2

]

Address bits used for tag field: A[

31

✓ Answer: 31

:

5

✓ Answer: 5

J

Explanation

There are 8 cache lines to index, so 3 bits are required for the index. Recall that bits 0 and 1 are used for word alignment, so bits 4:2 are used for the cache index. The remaining bits 31:5 are used for the tag.

2. Suppose the Beta does a read of location 0×5678 . Identify which cache location(s) would be checked to see if that location is in the cache. Select **all** locations that would be checked where 3A, for example, refers to row 3, section A. If there is a cache hit on this access what would be the contents of the tag data for the cache line that holds the data for this location?

☐ 1A☐ 2A☐ 3A☐ 4A☐ 5A☒ 6A☐ 7A☐ 1B☐ 2B☐ 3B☐ 4B☐ 5B☒ 6B☐ 7B



Cache tag data on hit for location 0x5678 (hex): 0x

✓ Answer: 2B3

Explanation

0x5678 = 0101 0110 0111 1000

tag = 01010110011 = 0x2B3

cache index = 110 = 6

byte offset = 00

3. Assume that checking the cache on each read takes 1 cycle and that refilling the cache on a miss takes an additional 8 cycles. If we wanted the average access time over many reads to be 1.1 cycles, what is the minimum hit ratio the cache must achieve during that period of time? You needn't simplify your answer.

Minimum hit ratio for 1.1 cycle average access time:

✓ Answer: 7.9/8

Explanation

Average cache access time = (hit_rate * hit_time) + ((1 - hit_rate) * miss_time)

1.1 = (hit_rate * 1) + ((1 - hit_rate) * 9) = 9 - (8 * hit_rate)

8 * hit_rate = 7.9

hit_rate = 7.9/8

4. Estimate the approximate cache hit ratio for the following program. Assume the cache is empty before execution begins (all the valid bits are 0) and that an LRU replacement strategy is used. Remember the cache is used for both instruction and data (LD) accesses.

```

        . = 0
        CMOVE(source,R0)
        CMOVE(0,R1)
        CMOVE(0x1000,R2)
loop:   LD(R0,0,R3)
        ADDC(R0,4,R0)
        ADD(R3,R1,R1)
        SUBC(R2,1,R2)
        BNE(R2,loop)
        ST(R1,source)
        HALT()

        . = 0x100
source:
        . = . + 0x4000 // Set source to 0x100, reserve 0x1000
words

```

Approximate hit ratio:

5/6

✓ Answer: 5/6

Explanation

Cache Line	Hex Addr	Code
-----	-----	-----
		. = 0
0	0	CMOVE(source, R0)
1	4	CMOVE(0, R1)
2	8	CMOVE(0x1000, R2)
3	C	loop: LD(R0, 0, R3)
4	10	ADDC(R0, 4, R0)
5	14	ADD(R3, R1, R1)
6	18	SUBC(R2, 1, R2)
7	1C	BNE(R2, loop)
0	20	ST(R1, source)
1	24	HALT()
		. = 0x100
		source:
		. = . + 0x4000 // Set source to
		0x100, reserve 0x1000 words

The loop has 5 instructions and one data fetch for each iteration through the loop. The 5 instruction fetches result in a cache hit, and the single data fetch results in a cache miss because each time a new piece of data is being fetched. Note that because the cache is 2-way set associative, and the LRU replacement policy is used, the data cache misses do not remove any of the instructions from the cache so the only misses that occur are for the data fetches. The resulting average cache hit ratio is 5/6.

5. After the program of part (D) has finished execution what information is stored in row 4 of the cache? Give the addresses for the instruction word and the data word that are cached (one in each of the sections).

Address of instruction that is cached in "Row 4": 0x

10

✓ Answer: 10

Address of data that is cached in "Row 4": 0x

40F0

✓ Answer: 40F0

Explanation

As we saw in the solution to the previous part of this question, the address of the instruction that is stored in row 4 is 0×10 .

The address of the data word that is stored in row 4 at the end of the program execution is that last piece of data that would go in row 4. Since there are 0×1000 total elements and we begin storing them at address 0×100 and each requires 4 bytes of storage, we use the address space 0×100 to 0×4100 to store all the data elements. Counting backwards from 0×4100 , address $0 \times 40FC$ would go in row 7, address $0 \times 40F8$ would go in row 6, $0 \times 40F4$ would be in row 5, and finally $0 \times 40F0$ would be the last element stored in row 4 of the cache.

Submit

i Answers are displayed within the problem

Caches

Mapping Data to Cache

. = $0 \times 40E0$	Address
D[$0 \times 0FF8$]	$0 \times 40E0$
D[$0 \times 0FF9$]	$0 \times 40E4$
D[$0 \times 0FFA$]	$0 \times 40E8$
D[$0 \times 0FFB$]	$0 \times 40EC$
D[$0 \times 0FFC$]	$0 \times 40F0$
D[$0 \times 0FFD$]	$0 \times 40F4$
D[$0 \times 0FFE$]	$0 \times 40F8$
D[$0 \times 0FFF$]	$0 \times 40FC$

line 0 of set A.

In a similar manner the next 2 CMOVE instructions and the LD instruction will be loaded into lines 1-3 of set A.

At this point, we begin loading data.

Since the cache is 2-way set associative, the data will be loaded into set B instead of removing the instructions that were loaded into set A.


The instructions that are outside the loop will end up getting taken out of set A in favor of loading a data item into those cache locations, but the instructions that make up the loop will not be displaced because every time something maps to cache lines 3-7, the least recently used location will correspond to a data value not to the instructions which


▶ 13:13 / 13:13 ▶ 1.0x 🔊 🔍 CC “

Video

📄 [Download video file](#)

Transcripts

 [Download SubRip \(.srt\) file](#)

 [Download Text \(.txt\) file](#)

Discussion

Hide Discussion

Topic: 14. Caches and the Memory Hierarchy / WE14.2

Add a Post

Show all posts



by recent activity



There are no posts in this topic yet.

