223_hwk8_zachary_hightower

CSCI 223 Computer Org. & Assembly Language

Homework #8

Name:

Due: 9:30am on Thursday 4/4/2024
Submission: Type your answer using any text editor (no handwriting) and upload the file to the Blackboard. No late submission will be accepted

Suppose we have a system with the following properties:

- The memory is byte addressable
- Memory accesses are to 1-byte words (not to 4-byte words)
- Addresses are 12 bits wide
- The cache is 2-way set associative with a 4-byte block size and 4 sets

The contents of the cache are as follows, with all addresses, tags, and values given in hexadecimal notation:

Set index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0	--	--	--	--
2	00	1	48	49	4A	4B
	40	0	--	--	--	--
3	FF	1	9A	C0	03	FF
	00	0	--	--	--	--

A. The following diagram shows the format of an address (one bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO The cache block offset
CI The cache set index
CT The cache tag

11	10	9	8	7	6	5	4	3	2	1	0

B. For each of the following memory accesses indicate if it will be a cache hit or miss when **carried out in sequence** as listed. Also give the value of a read if it can be inferred from the information in the cache. Show your work or no point will be given.

Operation	Address	Hit?	Read value (or unknown)
Read	0x834		
Write	0x836		
Read	0xFFD		

ANS-A

The cache block offset will be the final 2 bits at the right portion of the table. This is because the cache lines are 4 bytes long. So we have to allocate the b in this formula $4 = 2^b$ to represent it.

The set index requires more, and is placed in the middle, between the cache offset and the tag. The cache set index needs two bits as well, since it must have enough to represent 2×2 places in the memory, so it must be the amount of s in the following formula $4 = 2^s$ So we can see that it must be 2 bits as well.

The cache tag takes up the remaining bits. Since we have 12 bits total, and we've used 4, we can see that the tag takes up $12 - 4 = 8$ bits. It goes in the front.

11	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CT	CT	CT	CT	CT	CT	CT	CT	CI	CI	CO	CO

ANS-B

Now we want to send some instructions and check to see if they will function in our cache.

Operation	**Address**	**Hit?**	**Read value(or unknown)**
Read	0x834		
Write	0x836		
Read	0xFFD		

The first address we want to read is at
$0x834$

Which in binary is
100000110100

We can put it into our table and

11	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	0	0	0	0	0	1	1	0	1	0	0

See where all the bits fall.
CT= 10000011=83
CI =01=1
CO=00=0

Now we reference our table
Set index 1
Look inside for tag 83
We found it
It's valid
So we can record a **hit**
Next we want to read the byte at offset 0
It is non-null, we read the value of **FE**
We read only the value of FE, since out offset is 0, we only read only the first byte

The second address we want to read is at
$0x836$

Binary
100000110110

CT= 10000011=83
CI =01=1
CO=10=2

Now we reference our table
Set index 1
Look inside for tag 83
We found it
It is not valid
So we record a **miss**
And we can record unknown for our read value, since we aren't actually reading anything

The third address we want to read is at
$0xFFD$

Binary
111111111101

CT= 11111111=FF
CI =11=3
CO=01=1

Now we reference our table
Set index 3
Look inside for tag FF
We found it
It is valid

So we record a **hit**
And we want to read byte 1, because our offset value is 1
So the reader will return **C0**

Operation	Address	Hit?	Read value(or unknown)
Read	0x834	hit	FE
Write	0x836	miss	(unknown)
Read	0xFFD	hit	C0