91.305 ASSIGNMENT 9: CACHE MEMORY NAME______________________________

PROBLEM 1: 8-BYTE CACHE FOR 16-BYTE MEMORY.

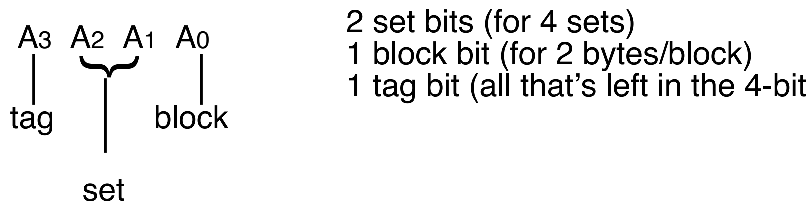
Assume we are building a cache for a memory system that's just 16 bytes big – 4 address bits.

We will make a direct mapped cache that has four set, so there are two set bits.

Each set has one line – because it's a direct mapped cache. That's the definition of direct-mapped – one line per set.

Each line has a block consisting of two bytes, so there is one block bit.

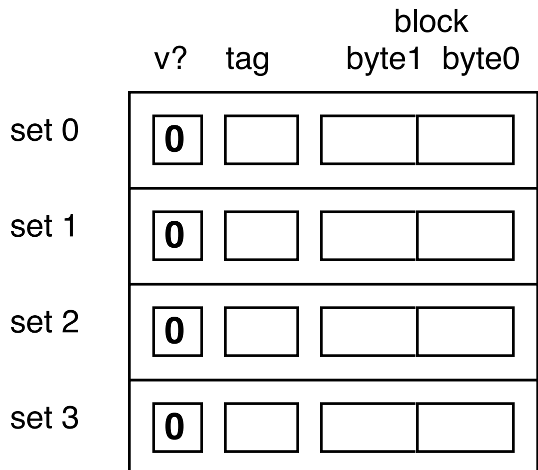
So for our 4-bit address, we have:



Your task is to show what happens over a series of memory reads. Assume that the main memory is as follows:

ADDR	VALUE	ADDR	VALUE
0	0x70	8	0x78
1	0x71	9	0x79
2	0x72	0xa	0x7a
3	0x73	0xb	0x7b
4	0x74	0xc	0x7c
5	0x75	0xd	0x7d
6	0x76	0xe	0x7e
7	0x77	0xf	0x7f

Here is an image of the cache in its initial state. As noted in the discussion, the cache is empty (a.k.a., "cold cache"), and all the valid bits are 0:



DISCUSSION

initial state ("cold cache")

- nothing is read in
- all valid bits are 0

Let’s start with reading from address 0. The set bits are ‘00,’ so set 0 is used. The tag bits are ‘0’ so this is saved with the two bytes of block data in the line, andthe corresponding valid bit is set. It is recorded as cache miss:

Read from address 0: HIT? ___ MISS? __X__

	v?	tag	block byte1	block byte0
set 0	1	0	71	70
set 1	0			
set 2	0			
set 3	0			

DISCUSSION

- set bits are ‘00’, set 0 is used
- tag bit is 0
- two bytes from main mem loaded into the line for set 0
- valid bit set

Read from address 1: HIT? ___ MISS? ___

	v?	tag	block byte1	block byte0
set 0				
set 1				
set 2				
set 3				

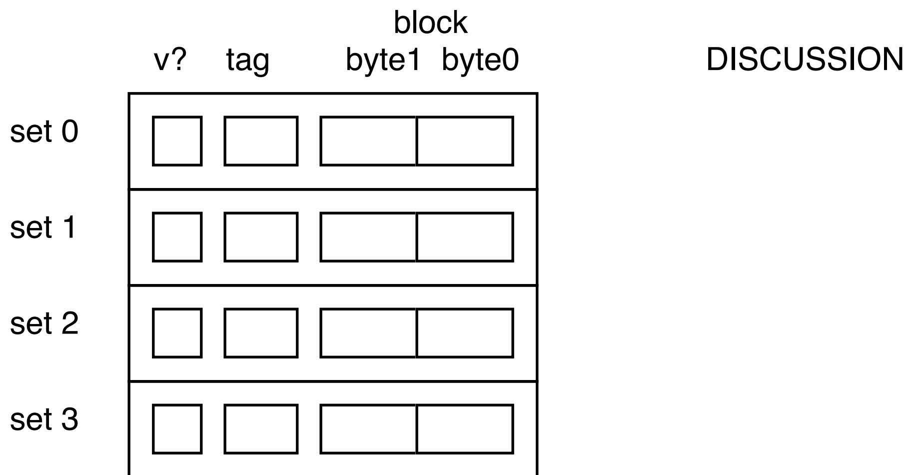
DISCUSSION

Read from address 4: HIT? ___ MISS? ___

	v?	tag	block byte1	block byte0
set 0				
set 1				
set 2				
set 3				

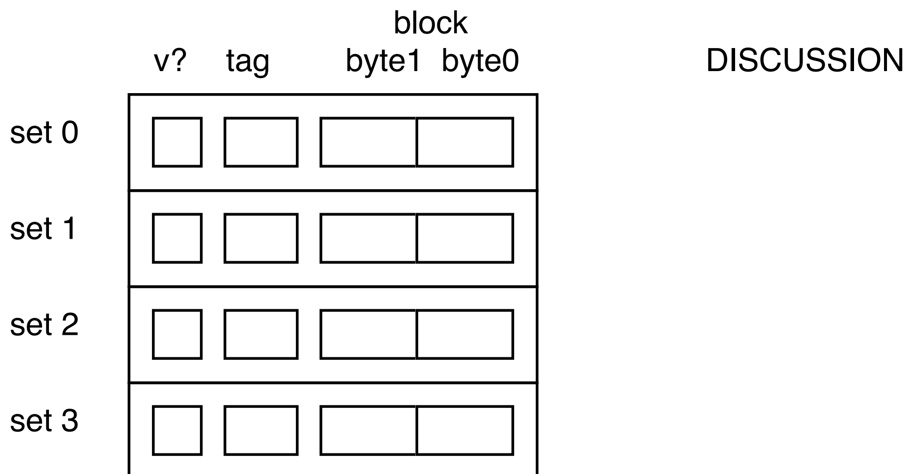
DISCUSSION

Read from address 8: **HIT?** ___ **MISS?** ___



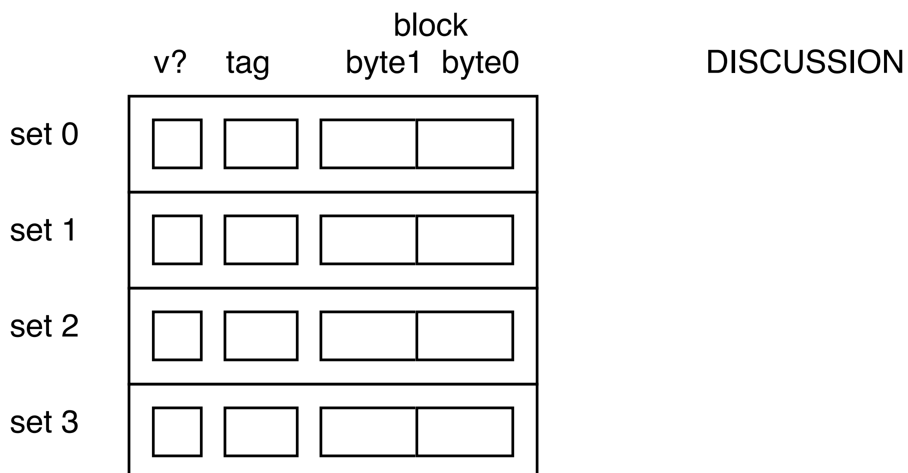
DISCUSSION

Read from address 9: **HIT?** ___ **MISS?** ___



DISCUSSION

Read from address 0xA: **HIT?** ___ **MISS?** ___

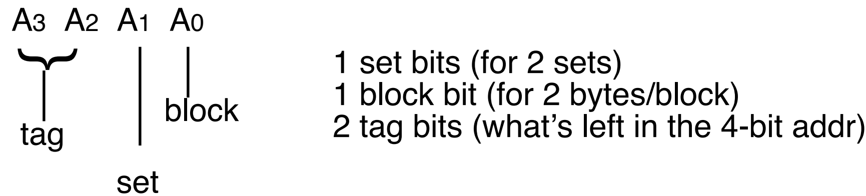


DISCUSSION

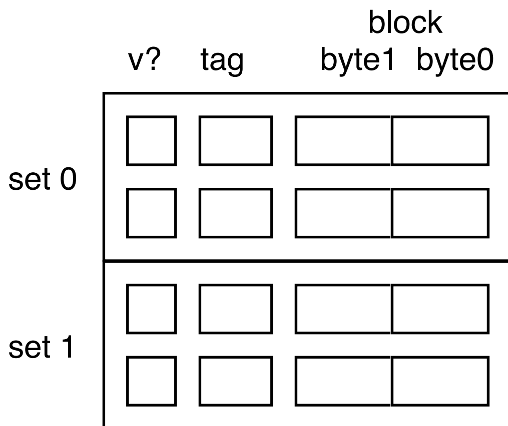
PROBLEM 2: TWO-WAY SET-ASSOCIATIVE CACHE.

Now, we'll go through the same series of memory accesses, but using a two-way set-associative cache.

The cache will still have 8 bytes of cache memory, but there will be two lines per set . Each set will hold 4 bytes, so we will have only two sets. Thus there are 1 set bit, 1 block bit, and 2 tag bits from our 4-bit address:



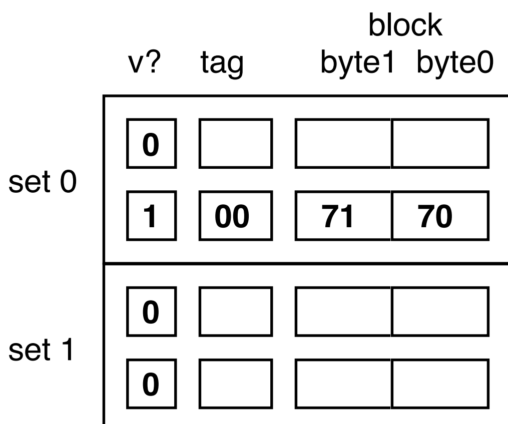
The cache looks like:



Note there is no bit indicating which line within the set should be used. That is the whole point of set associativity – any line in the set may be used. So even when multiple memory accesses map to the same set, there is the possibility that they can all be held in the cache (as long as there are enough lines per set). With the direct-mapped cache, if two different accesses map to the same set, the earlier one has to be ejected to make room for the newer one.

Let's start with reading from address 0. The set bits are '0,' so set 0 is used. The tag bits are '00' so this is saved with the two bytes of block data in the line. Either cache line in Set 0 may be used. The corresponding valid bit is set. It is recorded as cache miss:

Read from address 0: HIT? ___ MISS? __X__



DISCUSSION

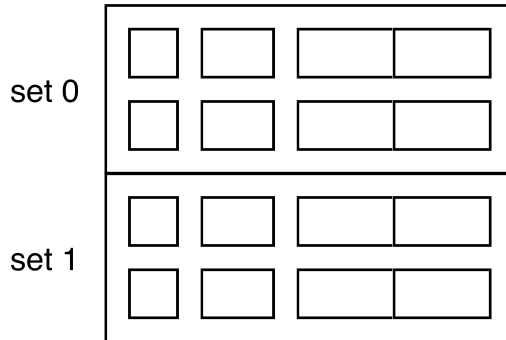
• contents of main memory are the same as with the earlier direct-mapped example

• I arbitrarily chose the 2nd line in set 0 to hold the data. The 1st one would have been just as good since both were empty at the start

Read from address 1: **HIT?** ___ **MISS?** ___

DISCUSSION

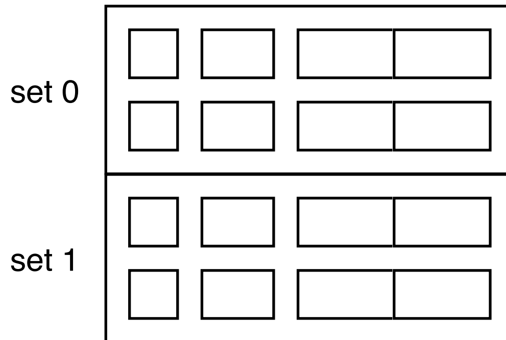
block
v? tag byte1 byte0



Read from address 4: **HIT?** ___ **MISS?** ___

DISCUSSION

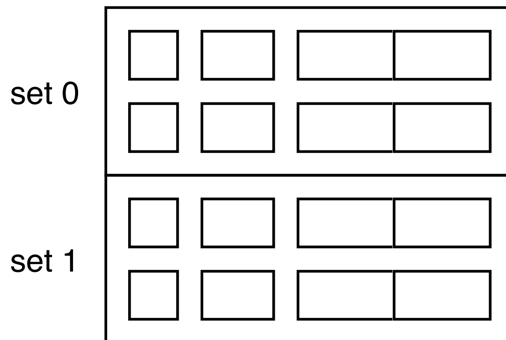
block
v? tag byte1 byte0



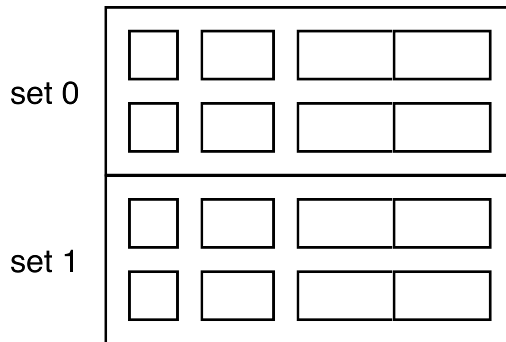
Read from address 8: **HIT?** ___ **MISS?** ___

DISCUSSION

block
v? tag byte1 byte0

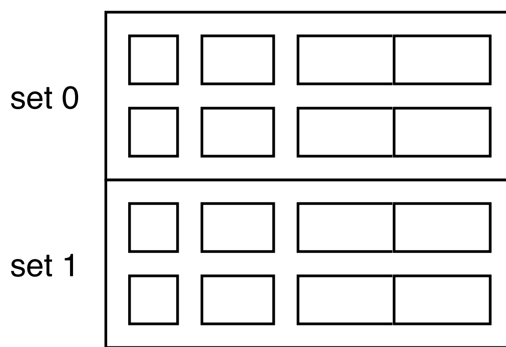


Read from address 9: HIT? ___ MISS? ___



DISCUSSION

Read from address 0xA: HIT? ___ MISS? ___



DISCUSSION

PROBLEM 3: DIRECT-MAPPED VS. SET-ASSOCIATIVE.

Which cache design did better in the two examples above—the direct-mapped cache, or the set-associate one?

If they both performed the same, construct a series of memory accesses that will cause the set-associative cache to “do a better job” (defined as having fewer cache misses) than the direct-mapped cache.

PROBLEM 4: DIRECT-MAPPED BETTER?

Is there a series of memory accesses for which the direct-mapped cache will outperform the set-associative one? If so, provide it. If not, explain why this is not possible.

PROBLEM 5: WRITING TO DIRECT-MAPPED CACHE.

For the next problem, we will use the direct-mapped cache (8 total bytes), 4 sets, 1 line per set, 2 bytes perblock),

but we will explore writing to the cache.

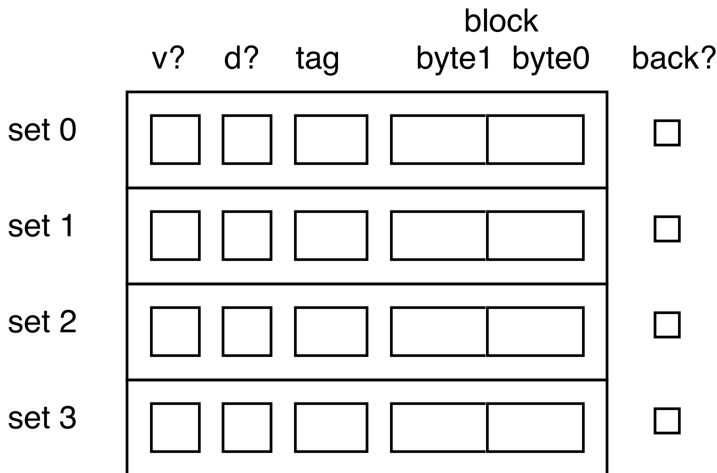
As discussed in Bryant/O'Hallaron (page 503), there are two strategies for write hits, and two for write misses. These are normally combined as follows:

• *write-through/write-no-allocate*: if already in cache, whole block is written to main memory when any one byte is written; if block is not in cache, it's not loaded on write. Simpler, but potentially causes more access to memory, because each time one byte of cache block is updated, the whole block is written to main memory.

• *write-back/write-allocate*: cache block is written to main memory when it's evicted from cache; block is read into cache from memory when any byte is written to. Requires use of "dirty" bit to indicate that cached block is different from memory, because after the block is read in, updates occur only to cache block until it's dumped.

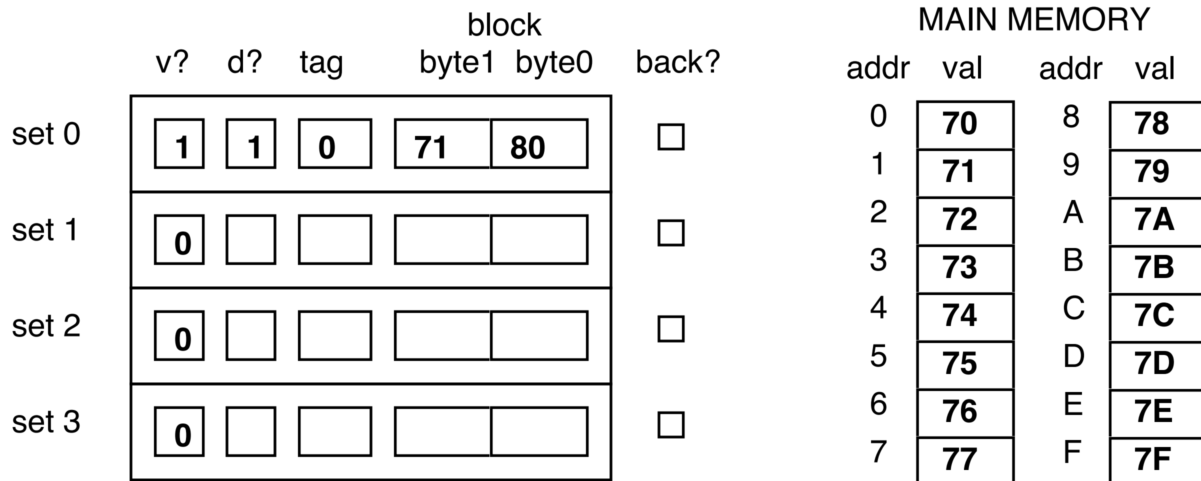
We will simulate the performance of the second of these choices—the write-back/write-allocate design.

Here is the cache. *Note the addition of the dirty bit "d?" per cache line, and the back? checkbox, which indicates the cache block was written to memory before being reloaded.*

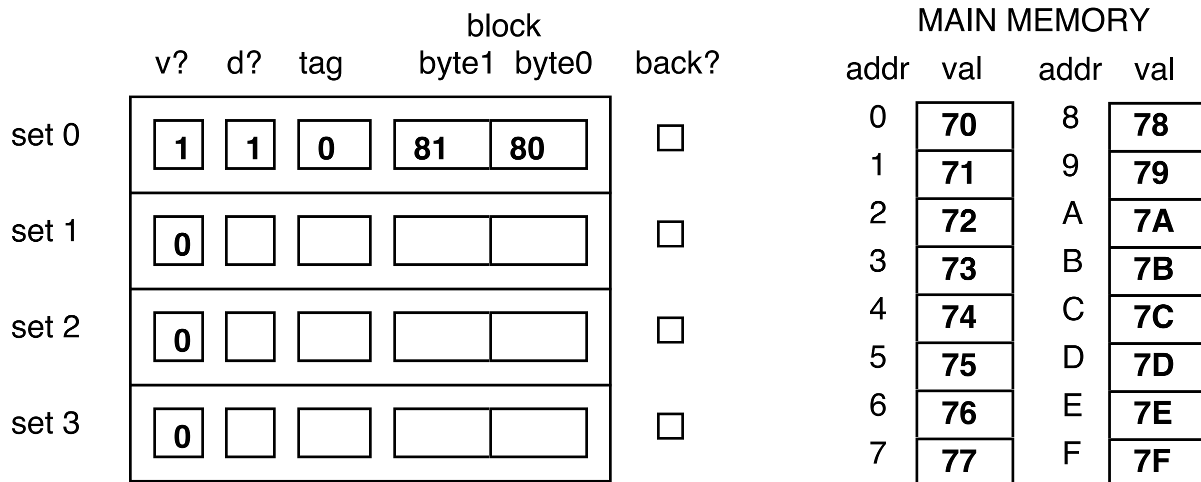


The initial state of memory is the same as before—each location from 0 to 0xF contains values from 0x70 to 0x7F.

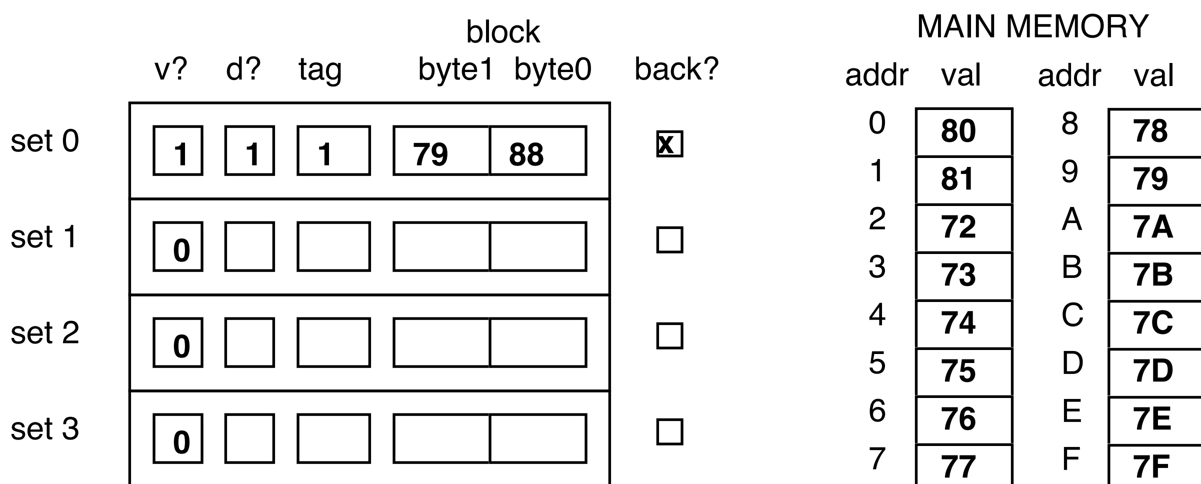
Let's get started by **writing an 0x80 to address 0**. The cache will load set 0 from addresses 0 and 1, and fill the cache with the new value for address 0. The dirty bit is set because at this point the cache block is changed from main memory:



Next we'll **write an 0x81 to address 1.** Only the cache block needs to be updated:

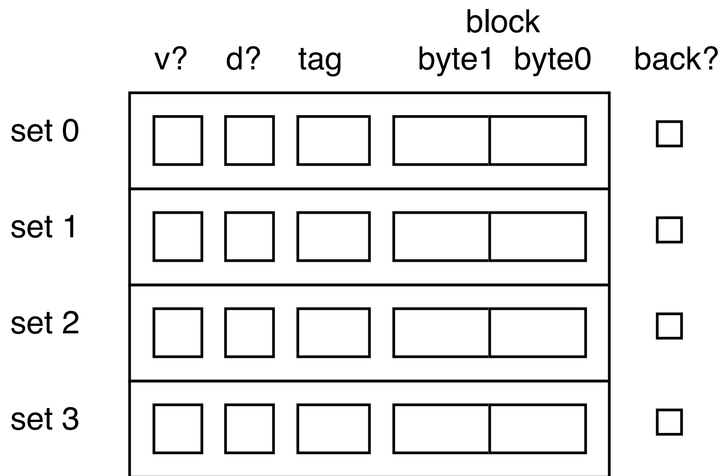


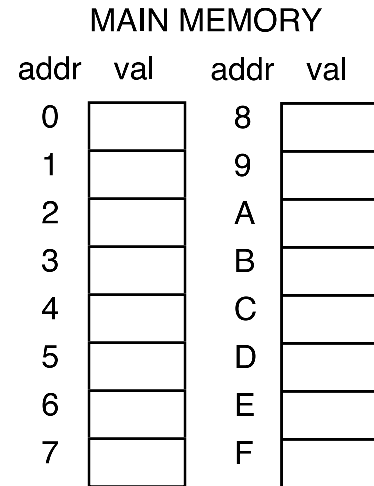
Next we'll **write an 0x88 to address 8.** Set 0 is written back to memory, and the cache is loaded from addresses 8 and 9:



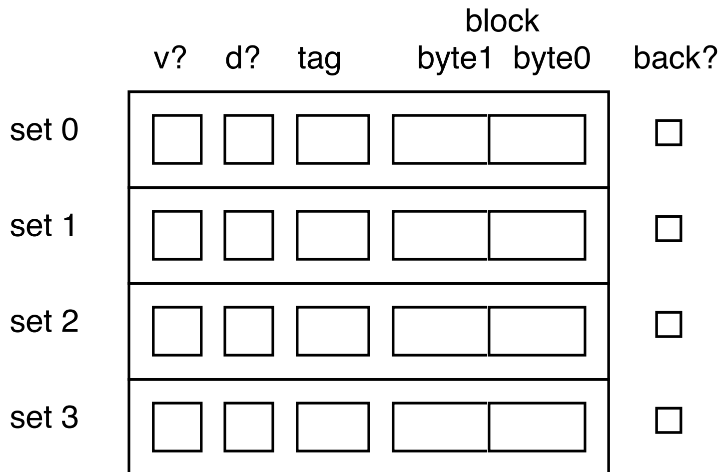
Now, you do it. Pick up from where we left off.

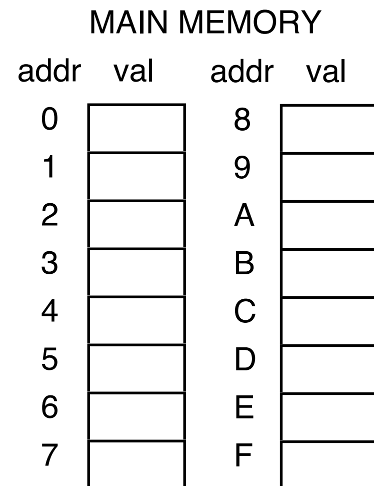
Write 0x89 to address 9:



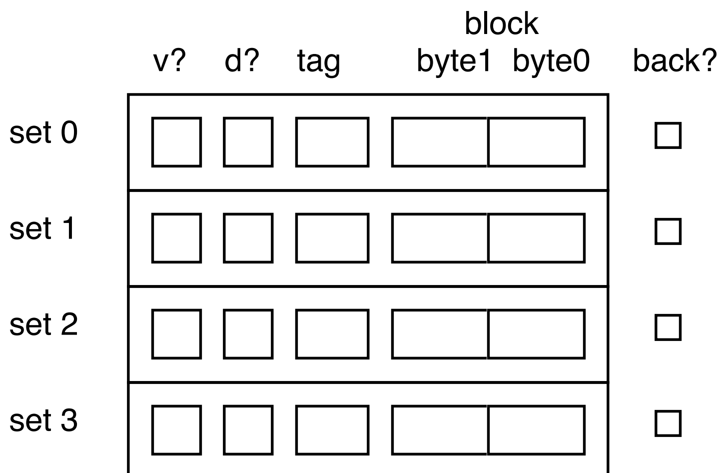


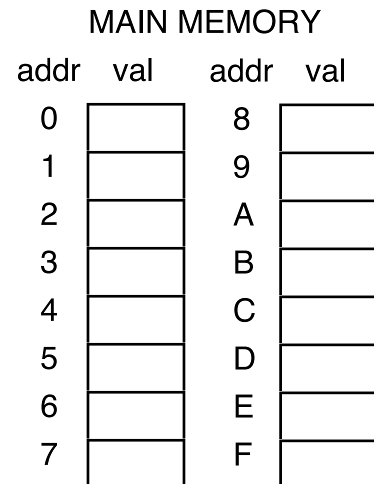
Read from address 3:



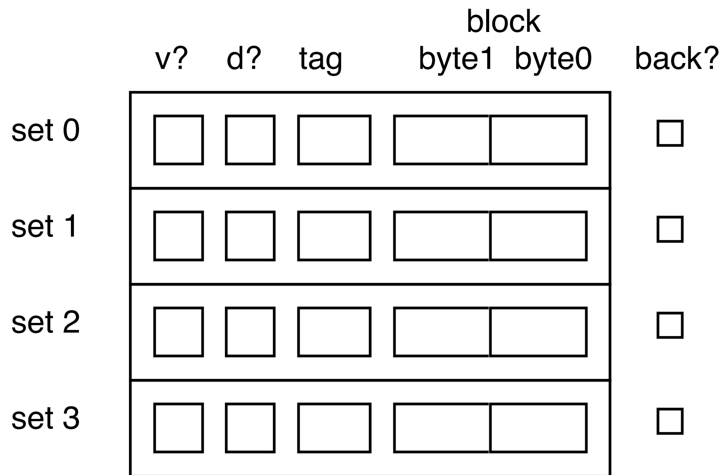


Write 0x84 to address 4:





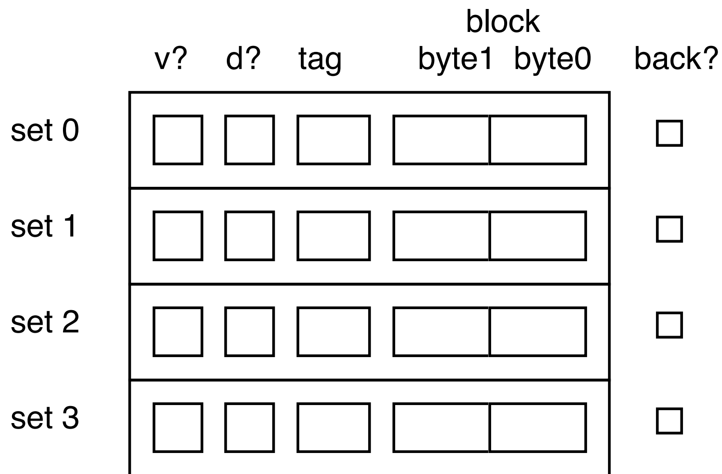
Read from address 1:



MAIN MEMORY

addr	val	addr	val
0		8	
1		9	
2		A	
3		B	
4		C	
5		D	
6		E	
7		F	

Write 0x89 to address 9:



MAIN MEMORY

addr	val	addr	val
0		8	
1		9	
2		A	
3		B	
4		C	
5		D	
6		E	
7		F	

PROBLEM 6: CACHE SIZES.

Do problem 6.21 on page 525. Turn in your answer on a separate piece of paper or fit it into the area below.