**TODO**

* Readme
* rename P3.py to simulate.py
* plots
* traces
* cntrl f all todo’s
* change output file name, or have an input cmd line argument (add input only at very end, if you feel like it
* zip necessary files up and submit to canvas. redownload and verify
* report

YUP. Also had a typo.

But now we’re overwriting things in FA that we shouldn’t be. Maybe we need to reset priority on write hit as well?

Jump until you get to 0x0300000

write 0x03000000: 0

write 0x04000002: 1

read 0x03000004: HIT!

read 0x01000003: 0… Ahh… Maybe we need to reset the bit to 0 on a hit as well, and increment the rest?

read 0x03000005

write 0x04000004

0: write 0x04000002

1: read 0x03000004

0: read 0x01000003

HIT! But it missed last time I think

So it’s more complex than that. The chain extends

Let’s see where 0x---04 gets stored and where 0x0100—3 gets stored and where 0x0300—05 get stored

Test 1 puts everything in the same index? At least for DM case. I imagine it would stay the same, although maybe I should verify this. Anyway, what this means is that once we move to 2W, 4W, FA we should see some big jumps in hit rate

That highlighted 49152 should be a hit. Check your LRU replacement strategy for read

1024 8 2W WB

Read. address: 0x02000006 Tag: 32768 index: 0 miss

Read. address: 0x04000004 Tag: 65536 index: 0 miss

Write. address: 0x02000007 Tag: 32768 index: 0 hit

Read. address: 0x06000004 Tag: 98304 index: 0 miss

Write. address: 0x01000001 Tag: 16384 index: 0 miss

Read. address: 0x01000002 Tag: 16384 index: 0 hit

Write. address: 0x02000000 Tag: 32768 index: 0 miss

Read. address: 0x06000007 Tag: 98304 index: 0 miss

Read. address: 0x05000003 Tag: 81920 index: 0 miss

Write. address: 0x02000001 Tag: 32768 index: 0 miss

Write. address: 0x03000000 Tag: 49152 index: 0 miss

Write. address: 0x04000002 Tag: 65536 index: 0 miss

Read. address: 0x03000004 Tag: 49152 index: 0 hit

Read. address: 0x01000003 Tag: 16384 index: 0 miss

Read. address: 0x03000005 Tag: 49152 index: 0 miss

Write. address: 0x04000004 Tag: 65536 index: 0 miss

Read. address: 0x04000007 Tag: 65536 index: 0 hit

Write. address: 0x05000003 Tag: 81920 index: 0 miss

Read. address: 0x02000000 Tag: 32768 index: 0 miss

Write. address: 0x02000003 Tag: 32768 index: 0 hit

Read. address: 0x03000002 Tag: 49152 index: 0 miss

Read. address: 0x00700002 Tag: 7168 index: 0 miss

Read. address: 0x02000005 Tag: 32768 index: 0 miss

Write. address: 0x01000001 Tag: 16384 index: 0 miss

Read. address: 0x01000006 Tag: 16384 index: 0 hit

Write. address: 0x01000005 Tag: 16384 index: 0 hit

Write. address: 0x07000006 Tag: 114688 index: 0 miss

Write. address: 0x02000003 Tag: 32768 index: 0 miss

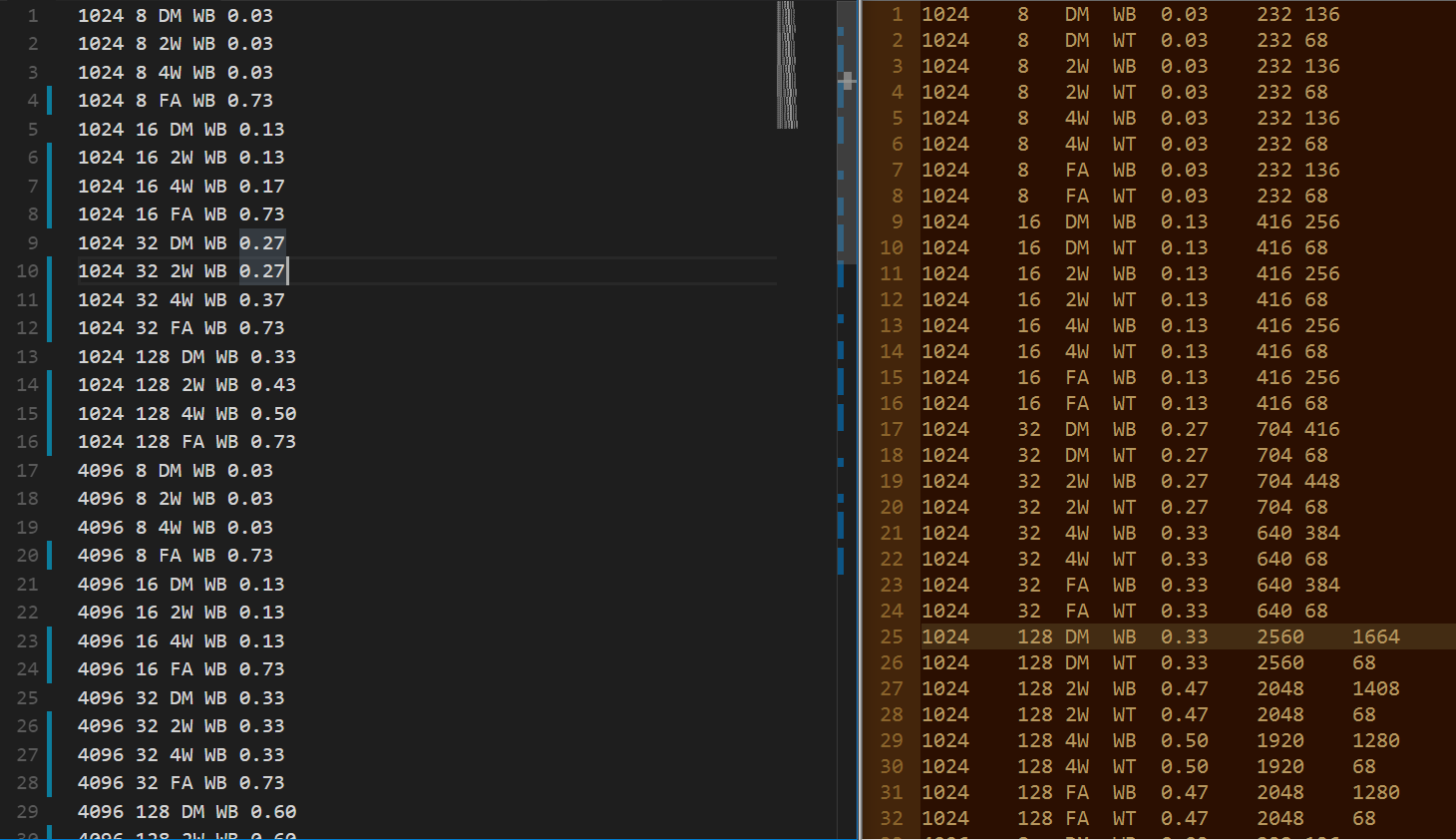
Read. address: 0x03000002 Tag: 49152 index: 0 miss

Read. address: 0x01000000 Tag: 16384 index: 0 miss

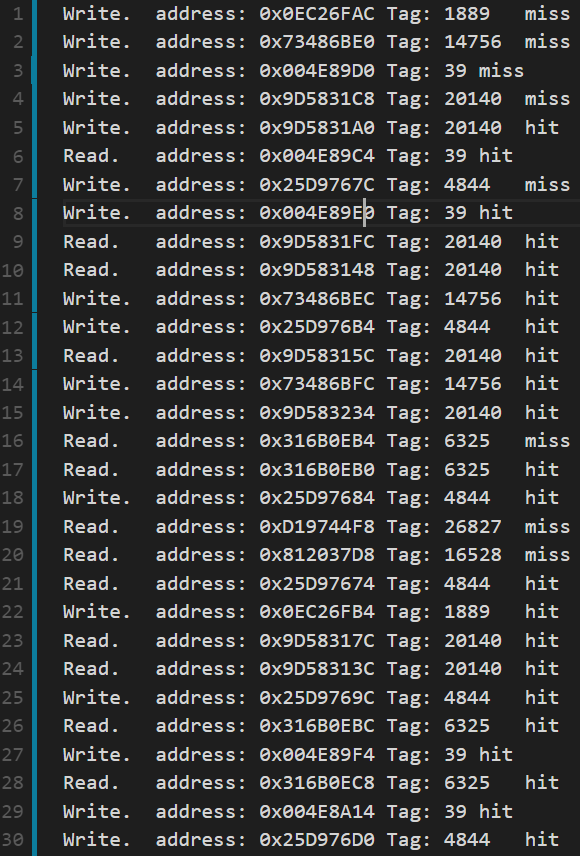
Read. address: 0x03000001 Tag: 49152 index: 0 hit

Read. address: 0x01000007 Tag: 16384 index: 0 hit

I'm still confused on this point... as you recommended, I've created a true LRU (not random). My hit rate matches for DM, but as I move towards FA my hit rate gets higher and higher. This makes sense to me based on my understanding of how I think this strategy should work. See my test 3 vs given below:



As you said, even a 1K cache is huge, so there won't be replacement. Based on my understanding, for fully associative this should mean that the hit rate is very high for our tests, because we reuse so many addresses that have been in a block read/written through previously. See below for a debug file for a FA cache:



As we continue, we start to get a lot of hits, because we're rereading/writing a bunch of address blocks, such as the block with Tag 39, the block with Tag 20140, the block with Tag 4844, etc. And unlike for the other cache placement strategies, we're not overwriting these blocks.

Going through this debug file, the hits and misses I'm getting all make logical sense to me.

I feel like I'm fundamentally misunderstanding something.

Any help would be greatly appreciated.

Thanks!

YUP! Could have gotten here easier by skimming through all the slides first, but whatever. Move slowly and carefully next time, like initially building the circuit in Analog Lab. It’ll save you time in the long run

Yeah, see below. 0x004E89D0 has tag 5026, but is in index 56. Meanwhile, 0x004E89C4 also has tag 5026, but is in index 58. Thus they’re in different blocks! But shouldn’t addresses in different blocks have unique tags? For the FA case this is a must. Should the tag computation differ

Maybe the block’s im bringing in aren’t the right size for FA?. I mean 0x004E89D0 may be a different block (index in DM) from 0x004E89E0, yet the tag stays the same.

Getting some misses which should be hits on DM? Nah probably outside of the block. different index most likely

1024 8 DM WB

Write. address: 0x0EC26FAC Tag: 241819 miss

Write. address: 0x73486BE0 Tag: 1888794 miss

Write. address: 0x004E89D0 Tag: 5026 miss

Write. address: 0x9D5831C8 Tag: 2577932 miss

Write. address: 0x9D5831A0 Tag: 2577932 miss

Read. address: 0x004E89C4 Tag: 5026 miss

Write. address: 0x25D9767C Tag: 620125 miss

Write. address: 0x004E89E0 Tag: 5026 miss

Read. address: 0x9D5831FC Tag: 2577932 miss

Read. address: 0x9D583148 Tag: 2577932 miss

Write. address: 0x73486BEC Tag: 1888794 miss

Write. address: 0x25D976B4 Tag: 620125 miss

Read. address: 0x9D58315C Tag: 2577932 miss

Write. address: 0x73486BFC Tag: 1888794 miss

Write. address: 0x9D583234 Tag: 2577932 miss

Read. address: 0x316B0EB4 Tag: 809667 miss

Read. address: 0x316B0EB0 Tag: 809667 hit

Write. address: 0x25D97684 Tag: 620125 miss

Read. address: 0xD19744F8 Tag: 3433937 miss

Read. address: 0x812037D8 Tag: 2115597 miss

Read. address: 0x25D97674 Tag: 620125 miss

Write. address: 0x0EC26FB4 Tag: 241819 miss

Read. address: 0x9D58317C Tag: 2577932 miss

Read. address: 0x9D58313C Tag: 2577932 miss

Write. address: 0x25D9769C Tag: 620125 miss

Read. address: 0x316B0EBC Tag: 809667 miss

Write. address: 0x004E89F4 Tag: 5026 miss

Read. address: 0x316B0EC8 Tag: 809667 miss

Write. address: 0x004E8A14 Tag: 5026 miss

Write. address: 0x25D976D0 Tag: 620125 miss

He’s saying that WB and WT will effect hit rate

For WB, on write miss don’t we need to send that block to m

Total number of bytes transferred from cache to memory is more than the number possible simply through reads, so there must be something else

Any time you miss on a read, you’re gonna overwrite what’s in the cache with what’s in memory. Therefore, memory and cache are going to be consistent. Thus, set dirty bit to 0.

Any time you write (in WB method), you need to set dirty bit to 1.

Only write the cache to memory if the dirty bit is 1 (cache not consistent with memory)

* On write back:
  + On read, if you miss on an index, you need to write the block from cache 2 mem (grow bcache2mem) before overwriting it
  + On read, if you hit then you’re good (you’re not overwriting it)
  + On write hit, just write to it and set dirty bit?
  + On write miss, send block to memory if dirty bit is set?
* On write through
  + On read, I think it’s fine regardless of what you do. The memory is written immediately on write
  + On write, I think you always write the BYTE (just a byte, so just increment bcache2mem? Or are we even writing directly to mem? Let’s increment it for now, it seems like that’s what the diagram indicates)

Ok, on write through (write back might be the same) on cache miss you first fetch the entire block from memory. Then you update the cache and memory (but only one byte, not the whole block). On hit it would make sense you don’t fetch anything form memory. Write back seems the same.

The only difference is the next time we read on write back, the cache write to memory.

TEST 3 HAS 232 BYTES MEM2CACHE!!! BUT THERE’S ONLY 13 GODAMN READS -> MAX -> BLOCKSIZE\*READ = 8\*13 = 104 BYTES MEM2CACHE WTFFFFF. 232 BYTES/8 = 29. THERES ONLY 30 GODAMN INSTRUCTIONS WHAT IS LIFE THIS IS THE JANKIEST SHIT OF ALL TIME

Track total bytes transferred from memory to cache:

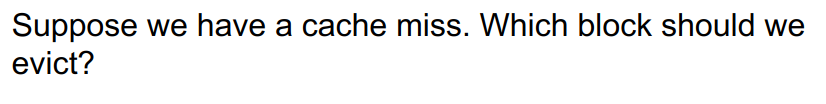
WB vs WT (dirty bit). When block evicted, transfer it to memory (

Every time there’s a memory read, and it misses, we must transfer a block from memory to cache. When it hits, no transfer (in this direction). I think this is regardless of memory/dirty bit. However, what happens when the block is evicted changes. This is verified by seeing that total bytes transferred from memory to cache doesn’t change for WB/WT

Every time there a memory write, we’re writing to cache. We’ll write to memory at some point depending on WB or WT, but I don’t think we’ll ever get a situation when transferring from memory to cache. So no worries with memory write, just modify/track memory read

block size?

Matches for DM, hit rate goes up as we move towards FA. This makes perfect sense to me but doesn’t match up with what they’ve got. Probably should just try to implement everything else next

ONLY EVICT LRU BLOCK ON MISS!

On a write, should I look through the matrix to see if that tag is there, and call that a hit if it is?

Pull in a whole block of memory addresses. Is this handled by the tag? (same tag for multiple addresses)

Check if Tag is correct, seems wrong in debug.txt

FOR FA, index is not always 0! Finding out why may solve my issues Only for debug.txt I think (no width being pulled their to adjust block\_count)

For LRU:

replace whichever piece has the maximum value. set that pieces counter to 0. increment the rest. Do I do this for read as well? Probably

Their DM, 2W, 4W, FA has no effect on the hit rate. This makes no sense…

2W: build the cache.

Then change write method.

Then change read method

For a 2 way cache, we want our width to be two and our block count to be halved (if 2W, width = 2,

elif 4W

FA)

block count = block count/width

Breaking when I up the size, “Index out of range”

Start by running through different sizes for direct mapped cache. Next, make the 2W, 4W, and FA. Finally, track bytes written and add in WB vs WT

Now writes and reads are hitting when the tag is the same, but index is different

test2: hitting on first read for some reason (valid bit should be 0, so hit should be impossible) Forgot to remove test line

Write: Miss if data not in cache. If not in cache, write it to cache?

WB and WT won’t change hit rate. However, you do need to track:

total bytes transferred from cache to memory

total byte transferred from memory to cache

both of which will change

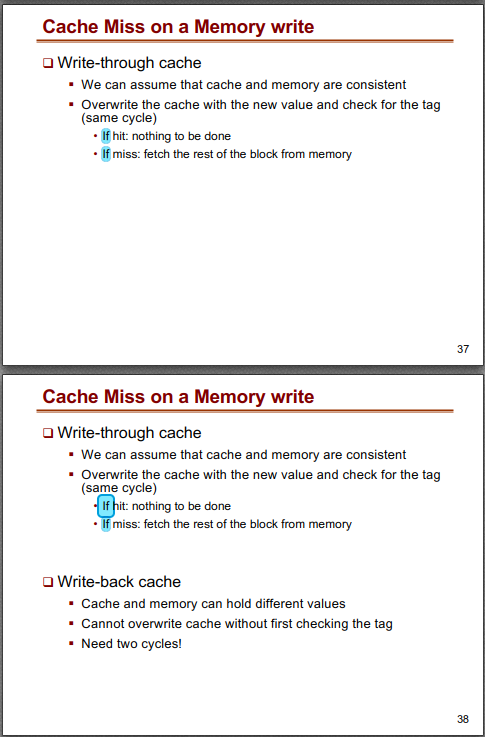
parse file: count number of lines. Set loop to number of lines.

if read

else if write

else:

do nothing



0x04000000 should be index 0, tag 65536

In main:

ParseFile()

{

RWvar;

address;

if “RWvar == read”

ReadFunction(numReads, numHits, address) TODO check if numHits or numMisses

else if “RWvar == write”

writeFunction(address) TODO track hits and misses here?

}

Have a function that builds and returns a CacheMatrix

Have a write Address function, which calculates where to store an element, set the tag, valid bit~~, and dirty bit~~.

Figure out what a dirty bit is and why it’s needed Not needed, that’s for virtual memory You actually do need a dirty bit, it’s used in the write back policy

Have a read address function, which takes in an address, calculates what the index and tag should be, checks the tag at that index. (If not direct mapped, checks multiple tags.) If valid bit is 0 or tag(s) doesn’t match, miss. Store total number of misses, also increment every read. Can probably pass in totalReads variable or something and just increment at the top of read method.

First, figure out how to simulate a cache of one type.

Then, make a class/method which will build the cache based on the 4 relevant variables.

Then, simulate reads/writes on the cache. Recall we don’t actually care what the data is

Finally, read in the file and interpret “read” for read method and “write” for write method, and interpret hex address.

Then we need to track certain things, plot those things, etc.