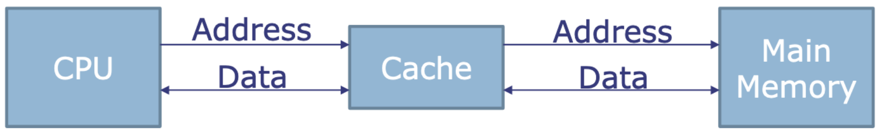
6.004 Worksheet Questions
L15 – Memory Hierarchy

Keep the most often-used data in a small, fast SRAM (often local to CPU chip). The reason this strategy works: LOCALITY.

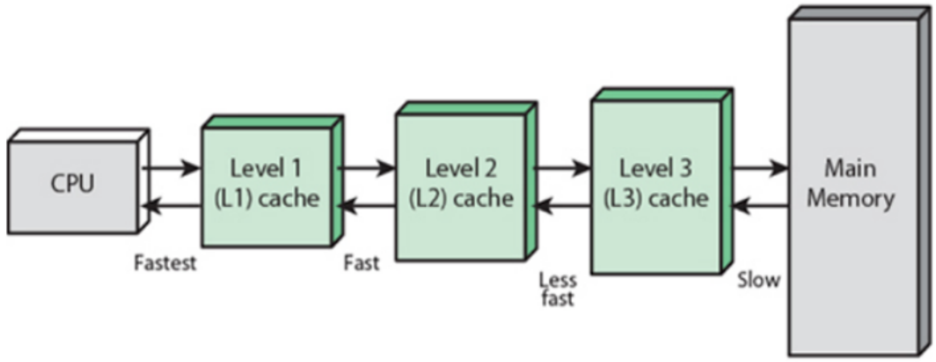
– *Temporal locality: If a location has been accessed recently, it is likely to be accessed (reused) soon*
– *Spatial locality: If a location has been accessed recently, it is likely that nearby locations will be accessed soon*

AMAT(Average Memory Access Time) = HitTime + MissRatio * MissPenalty



Problem 1 ★

Belly Eyelash is designing a processor and is analyzing the performance of different numbers of cache levels. Without any caches, Belly's main memory has an access time of 140 cycles.



(A) As a test, Belly adds a 32 KB Level 1 (L1) cache that has single-cycle reads/writes, and runs a long computation, during which she observes a new AMAT of 10. **What is the hit ratio for the Level 1 cache during this test?**

AMAT = 10 = 1 + (1 – HitRatio)*140
HitRatio = 131/140

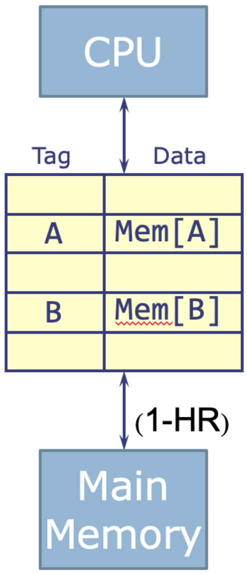
(B) Next, Belly adds a 256 KB Level 2 (L2) cache between the L1 cache and main memory. The L2 cache takes 4 cycles to decide if the a memory access is a hit or a miss. After running the same computation with this new memory hierarchy, Belly observes an improved AMAT of 1.45. Assume that the hit rate for the L1 cache is the same as in (A). **What is the hit ratio for the Level 2 cache during this test?**

AMAT = 1.45 = 1 + (9/140)*(4 + (1 – HitRatio)*140)
.45*(140/9) - 4
= 7 – 4 = (1- HitRatio)*140
HitRatio = 137/140

(C) Finally, Belly adds a 10 MB Level 3 (L3) cache between the L2 cache and main memory. After running the same test as before, she observes that the main memory was accessed once for every 140 accesses to the L3 cache. If Belly wants to achieve an AMAT of 1.3 for this computation, **what is the maximum number of clock cycles that the Level 3 cache can take to decide if a memory access is a hit or a miss?** Assume that the L1 and L2 caches can not be changed and they have the same hit rates observed in (A) and (B).

AMAT = 1.3 = 1 + (9/140)*(4 + (3/140)*(HitTime + (1/140)*140))
$.3*140^2$ = 9*140*(4 + 3*(HitTime + 1))
HitTime = ($.3*140^2$ – 36*140)/27 - 1
To achieve AMAT <= 1.3, **HitTime <= 30**

Basic Cache Algorithm (Reads)



On reference to Mem[X], look for X among cache tags

HIT: X = Tag(i) for some cache line i

Return Data(i)

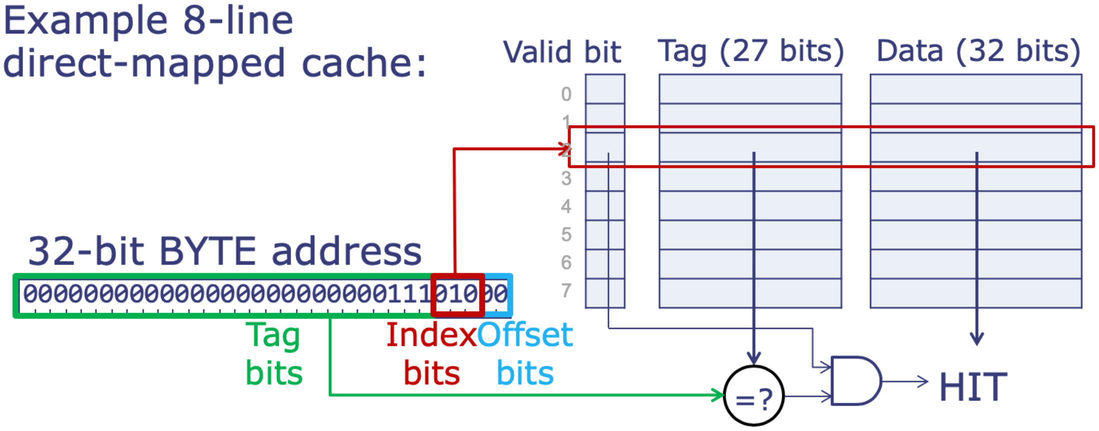
MISS: X not found in Tag of any cache line

Read Mem[X]
Return Mem[X]
Select a line k
to hold Mem[X]
Write Tag(k)=X,
Data(k) = Mem[X]

Q: How do we "search" the cache?

Direct-Mapped Caches

- Each word in memory maps into a single cache line
- Access (for cache with 2^W lines):
  - Index into cache with W address bits (the index bits)
  - Read out valid bit, tag, and data
  - If valid bit == 1 and tag matches upper address bits, HIT
- Example 8-line direct-mapped cache:



Example: Direct-Mapped Caches

64-line direct-mapped cache → 64 indexes → 6 index bits

Read Mem[0x400C]

0100 0000 0000 1100

TAG: 0x40
INDEX: 0x3
OFFSET: 0x0

HIT, DATA 0x42424242

Would 0x4008 hit?

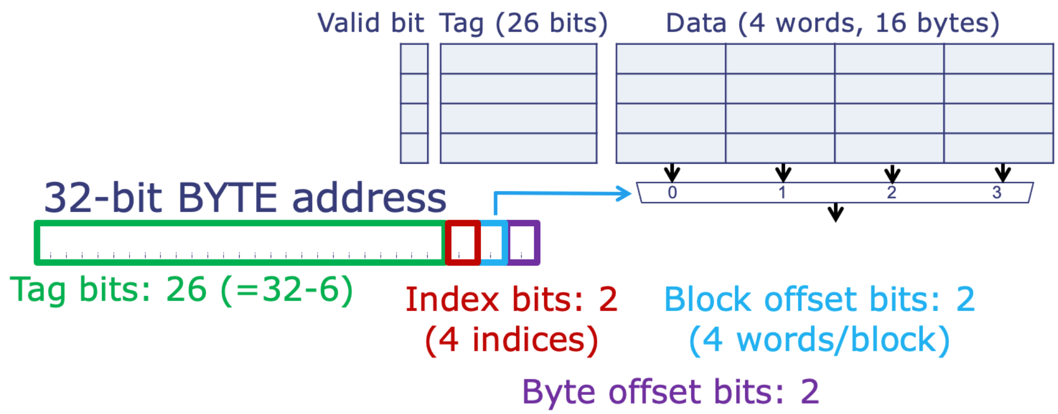
INDEX: 0x2 → tag mismatch
→ MISS

	Valid bit	Tag (24 bits)	Data (32 bits)
0	1	0x000058	0xDEADBEEF
1	1	0x000058	0x00000000
2	1	0x000058	0x00000007
3	1	0x000040	0x42424242
4	0	0x000007	0x6FBA2381
	⋮	⋮	⋮
63	1	0x000058	0xF7324A32

> Part of the address (index bits) is encoded in the location
> Tag + Index bits unambiguously identify the data's address

Block Size

- Take advantage of spatial locality: Store multiple words per data line
  - Always fetch entire block (multiple words) from memory
  - Another advantage: Reduces size of tag memory!
  - Potential disadvantage: Fewer indices in the cache
- Example: 4-block, 16-word direct-mapped cache



Problem 2

The RISC-V Engineering Team is working on the design of a cache. They've decided that the cache will have **a total of 2^{10} = 1024 data words**, but are still thinking about the other aspects of the cache architecture.

First assume the team chooses to build a direct-mapped cache with a block size of 4 words.

(A) Please answer the following questions:

Number of lines in the cache: 256

Tag: 20bit, Index: 8bit, Offset: 4bit

Number of bits in the tag field for each cache entry: 20

(B) This cache takes *2 clock cycles* to determine if a memory access is a hit or a miss and, if it's a hit, return data to the processor. If the access is a miss, the cache takes *20 additional clock cycles* to fill the cache line and return the requested word to the processor. If the hit rate is 90%, what is the processor's average memory access time in clock cycles?

Average memory access time assuming 90% hit rate (clock cycles): 4

HitTime = 2 clock cycles
MissRatio = 1 – HitRatio = 1 – 90%
MissPenalty = 20

$$AMAT = 2 + (1 - .90) \times 20 = 4$$

Problem 3 ★

(A) The timing for a particular cache is as follows: checking the cache takes 1 cycle. If there's a hit the data is returned to the CPU at the end of the first cycle. If there's a miss, it takes 10 *additional* cycles to retrieve the word from main memory, store it in the cache, and return it to the CPU. If we want an average memory access time of 1.4 cycles, what is the minimum possible value for the cache's hit ratio?

Minimum possible value of hit ratio: _0.96_

1.4=1+(1-HR)*10 => HR=0.96

(B) If the cache block size, i.e., words/cache line, is doubled but the total number of data words in the cache is unchanged, how will the following cache parameters change? Please circle the best answer.

# of offset bits: UNCHANGED … +1 … -1 … 2x … 0.5x … CAN'T TELL

# of tag bits: UNCHANGED … +1 … -1 … 2x … 0.5x … CAN'T TELL

# of cache lines: UNCHANGED … +1 … -1 … 2x … 0.5x … CAN'T TELL

Consider a direct-mapped cache with 64 total data words with 1 word/cache line. This cache architecture is used for parts (C) through (F).

(C) If cache line number 5 is valid and its tag field has the value 0x1234, what is the address in main memory of the data word currently residing in cache line 5?

Main memory address of data word in cache line 5: 0x123414

64 total data words with 1 word/cache line → 64 total cache lines → 64 cache lines in a direct-mapped cache → 6 line index bits. Block size of 1 word → 0 block offset bits. There are 2 byte offset bits (i.e., the last two bits are always 0). The remaining bits are tag bits.

Tag: 24 bits (0x001234), Line index: 6 bits (line 5 = 000101), Byte offset: 2 bits (00)

Problem 4 ★

Consider a direct-mapped cache with 64 total data words with 1 word/cache line.

The program shown on the right repeatedly executes an inner loop that sums the 16 elements of an array that is stored starting in location 0x310.

The program is executed for many iterations, then a measurement of the cache statistics is made during one iteration through all the code, i.e., starting with the execution of the instruction labeled `outer_loop:` until just before the next time that instruction is executed.

```
  . = 0               // tell assembler to start at
                      // address 0
outer_loop:
  addi x4, x0, 16     // initialize loop index J
  mv x1, x0           // x1 holds sum, initially 0

loop:                 // add up elements in array
  subi x4, x4, 1      // decrement index
  slli x2, x4, 2      // convert to byte offset
  lw x3, 0x310(x2)    // load value from A[J]
  add x1, x1, x3      // add to sum
  bne x4, x0, loop    // loop until all words are summed

  j outer_loop        // perform test again!
```

(A) In total, how many instruction fetches occur during one complete iteration of the outer loop? How many data reads?

Number of instruction fetches: 83

Instruction fetch = 2+5*16+1=83

2 instructions before inner loop + 5 instructions in inner loop * 16 iterations + 1 jump instruction

Number of data reads: 16

(B) How many instruction fetch misses occur during one complete iteration of the outer loop? How many data read misses? Hint: remember that the array starts at address 0x310.

Number of instruction fetch misses: 4

Number of data read misses: 4

All data reads and instruction fetches operate on main memory and therefore interact with the cache. The data array starts at address 0x310 (line index = 4). The next array element is at 0x314 (line index = 5). The entire 16-element data array maps to cache lines 4-19.

The instructions start at address 0x0 (denoted by the ". = 0") which maps to line 0. The next instruction (mv x1, x0) starts at address 0x4 (line index = 1). The next instruction (subi) maps to line 2 and so on. The instructions therefore map to cache lines 0-7.

Between the data and instructions there are conflicts in cache lines 4-7. As the inner loop iterates over the entire data array, each one of these lines will, one at a time, evict the instruction with the data and then, on the next iteration, evict the data with the instruction. Thus, each of the conflicting lines will see one instruction miss and one data miss. Note that the assembly code iterates over the array backwards, but this does not change the reasoning.

Data: **0x310, 0x314, 0x318, 0x31C**, 0x320, 0x324, 0x328 …
Instruction: 0x0 (addi), 0x4 (mv), 0x8 (subi), 0xC (slli), **0x10** (lw), **0x14** (add), **0x18** (bne), **0x1C** (j) …

Bold Blue addresses are cache misses / conflicts.

(C) What is the hit ratio measured after one complete iteration of the outer loop?
Total cache hits = (83 instr fetches – 4 instr miss) + (16 data reads – 4 data miss) = 91
Total memory ops = 83 instr fetches + 16 data reads = 99

Hit ratio: 91/99