CS110: Computer Architecture I

Spring 2026

Homework #5 Solution

Problem 1 (*Cache Geometry and Address Breakdown*)

Given a 32-bit byte-addressed system and an L1 data cache with total size 8 KiB (8192 B), block size 64 B, direct-mapped.

(a) Cache lines $= 8192/64 = 128 = 2^7$.

Block offset bits $= \log_2 64 = 6$.

Index bits $= \log_2 128 = 7$.

Tag bits $= 32 - 7 - 6 = 19$.

Final answer: cache lines = 128,tag bits = 19,index bits = 7,offset bits = 6

(b) Offset is the low 6 bits:

$$\text{offset} = 0x1234ABCD \bmod 64 = 13.$$

Index is the next 7 bits:

$$\text{index} = (0x1234ABCD \gg 6) \bmod 128 = 47.$$

Tag is the remaining upper bits:

$$\text{tag} = 0x1234ABCD \gg 13 = 0x91A5.$$

Final answer: tag = `0x91A5`,index = 47,offset = 13

(c) Under the new cache geometry:

$$\text{offset bits} = \log_2 32 = 5.$$

Number of cache lines:

$$16384/32 = 512 = 2^9,$$

so the index has 9 bits.

From part (b), reuse:

$$\text{tag} = 0x91A5, \quad \text{index} = 47, \quad \text{offset} = 13.$$

Reconstruct the address using the new field widths:

$$\text{address} = (\text{tag} \ll 14) + (\text{index} \ll 5) + \text{offset}.$$

Therefore,

$$\text{address} = (0x91A5 \ll 14) + (47 \ll 5) + 13 = 0x246945ED.$$

Final answer: `0x246945ED`

Problem 2 (*Direct-Mapped Mapping and Conflicts*)

Consider a 32-bit byte-addressed system with a 128 B direct-mapped cache and 16 B block size. Use the following block-aligned byte addresses:

$$B_1 = \texttt{0x00000000},\ B_2 = \texttt{0x00000080},\ B_3 = \texttt{0x00000100},$$
$$B_4 = \texttt{0x00000010},\ B_5 = \texttt{0x00000090}$$

(a) The cache has

$$128/16 = 8$$

lines, so the line index is

$$(\text{address} \gg 4) \bmod 8$$

and the tag is

$$\text{address} \gg 7.$$

Using the line index, the five addresses partition into two conflict classes:

$$\{B_1, B_2, B_3\}, \qquad \{B_4, B_5\}.$$

Final answer: class $1 = B1, B2, B3$,class $2 = B4, B5$

(b) For the class containing B_1:

$$B_1 : \text{line } 0,\ \text{tag } 0x0; \quad B_2 : \text{line } 0,\ \text{tag } 0x1; \quad B_3 : \text{line } 0,\ \text{tag } 0x2.$$

So this class has common line index 0 and tag values `0x0`,`0x1`,`0x2`.

For the class containing B_4:

$$B_4 : \text{line } 1,\ \text{tag } 0x0; \quad B_5 : \text{line } 1,\ \text{tag } 0x1.$$

So this class has common line index 1 and tag values `0x0`,`0x1`.

Final answer: class of B_1: line = 0,tags = `0x0`, `0x1`, `0x2`;class of B_4: line = 1,tags = `0x0`, `0x1`

(c) Starting from an empty cache, simulate the trace

$$B_1,\ B_4,\ B_1,\ B_2,\ B_4,\ B_5,\ B_2,\ B_3.$$

Using the direct-mapped conflicts from parts (a) and (b):

$$B_1, B_2, B_3 \text{ share line } 0, \qquad B_4, B_5 \text{ share line } 1.$$

The hit/miss sequence is:

$$M,\ M,\ H,\ M,\ H,\ M,\ H,\ M.$$

So there are 3 hits and 5 misses.

With hit time = 2 cycles and miss penalty = 18 cycles,

$$\text{total access time} = 8 \times 2 + 5 \times 18 = 16 + 90 = 106 \text{ cycles}.$$

Final answer: total access time = 106 cycles

(d) Let the number of direct-mapped lines be N, still a power of two.

The block numbers are:

$$0,\ 8,\ 16,\ 1,\ 9.$$

We need these to have distinct residues modulo N.

For $N = 16$, blocks 0 and 16 still collide.

For $N = 32$, the residues are

$$0,\ 8,\ 16,\ 1,\ 9,$$

which are all distinct.

So the minimum number of lines is 32, and the minimum total size is

$$32 \times 16 = 512 \text{ B}.$$

Final answer: 512 bytes

Problem 3 (*Associativity and LRU*)

Use the same five block-aligned byte addresses from Problem 2. Keep the total cache size 128 B and the block size 16 B, but now change the mapping to a 2-way set associative cache with LRU replacement.

(a) Total blocks = $128/16 = 8$.

For a 2-way cache, number of sets = $8/2 = 4$.

The set index is

$$(\text{address} \gg 4) \bmod 4.$$

The blocks that map to the same set as B_1 are exactly

$$B_1,\ B_2,\ B_3.$$

Final answer: number of sets = 4,same-set blocks = $B1, B2, B3$

(b) Set 0 can hold at most two of $\{B_1, B_2, B_3\}$.

Set 1 can hold both B_4 and B_5 simultaneously.

So the maximum number of listed blocks that can coexist is

$$2 + 2 = 4.$$

Final answer: 4

(c) Keeping total size fixed means there are always 8 total cache blocks.

With associativity 2, the three blocks B_1, B_2, B_3 still map to one set with room for only two of them.

With associativity 4, the cache has $8/4 = 2$ sets, and the blocks split as:

$$\{B_1, B_2, B_3\} \text{ in one set}, \qquad \{B_4, B_5\} \text{ in the other set.}$$

Each set then has enough capacity.

Final answer: 4

(d) Consider the trace

$$0x00000004, 0x00000088, 0x0000000C, 0x00000100, 0x00000084, 0x00000108, 0x00000008.$$

With 16 B blocks, these seven byte addresses fall into only three distinct cache blocks:

$$0x00000004,\ 0x0000000C,\ 0x00000008 \in B_1,$$

$$0x00000088,\ 0x00000084 \in B_2,$$

$$0x00000100,\ 0x00000108 \in B_3.$$

So the block-level access pattern is still

$$B_1,\ B_2,\ B_1,\ B_3,\ B_2,\ B_3,\ B_1.$$

In the 2-way cache, starting from empty:

$$M,\ M,\ H,\ M,\ M,\ H,\ M$$

so the total number of hits is 2.

In the 4-way cache, the same three blocks fit in one set simultaneously, so starting from empty:

$$M,\ M,\ H,\ M,\ H,\ H,\ H$$

and the total number of hits is 4.

Final answer: hits in 2-way cache = 2,hits in 4-way cache = 4

Problem 4 (*Memory Performance and Data Layout*)

Part I: Two-level cache and CPI model

Use this machine model for parts (a) and (b):

- L1 hit time = 1 cycle
- L1 miss rate = 12%
- On an L1 miss, the access goes to L2
- L2 hit time = 6 cycles
- L2 local miss rate = 20%
- On an L2 miss, the additional main-memory penalty = 90 cycles
- Base CPI (without memory stalls) = 1.0
- Memory accesses per instruction = 0.40

Part II: Data layout and streaming kernels

For parts (c) to (e), keep the same base CPI, memory-access rate, and two-level cache hierarchy. Also assume cache block size is 64 B, `double` is 8 B, and `Particle` has no extra padding:

```
typedef struct  double x; double y; double z; double vx; double vy; double vz;
double mass; double charge;  Particle; Particle p[N]; double x[N], mass[N];
double acc = 0;
```

```
Kernel 1:  for (i = 0; i < N; i++) acc += p[i].x * p[i].mass;
```

```
Kernel 2:  for (i = 0; i < N; i++) acc += x[i] * mass[i];
```

(a) The effective stall penalty of one L1 miss, excluding the 1-cycle L1 hit time, is the L2 hit time plus the chance of falling through to memory:

$$6 + 0.20 \times 90 = 24 \text{ cycles}$$

Then:

$$\text{final CPI} = 1.0 + 0.40 \times 0.12 \times 24 = 1.0 + 1.152 = 2.152$$

Final answer: effective miss penalty = 24 cycles,final CPI = 2.15

(b) Optimization 1 changes only the L1 miss rate:

$$\text{CPI}_{\text{opt 1}} = 1.0 + 0.40 \times 0.09 \times 24 = 1.864$$

Optimization 2 changes only the L2 local miss rate, so the new L1-miss penalty is:

$$6 + 0.10 \times 90 = 15 \text{ cycles}$$

$$\text{CPI}_{\text{opt 2}} = 1.0 + 0.40 \times 0.12 \times 15 = 1.72$$

Since $1.72 < 1.864$, optimization 2 is better.

Final answer: CPI with opt 1 = 1.86,CPI with opt 2 = 1.72,better optimization = 2

(c) Since `Particle` has no extra padding, the struct has 8 doubles:

$$8 \times 8 = 64 \text{ B}$$

In Kernel 1, consecutive `x` values come from consecutive structs, so:

$$\text{x-stride in Kernel 1} = 64 \text{ B}$$

In Kernel 2, `x[i]` is read from a plain double array, so:

$$\text{x-stride in Kernel 2} = 8 \text{ B}$$

A 64 B block therefore supplies:

$$64/64 = 1 \text{ iteration's x value in Kernel 1}$$

$$64/8 = 8 \text{ iterations' x values in Kernel 2}$$

Final answer: Particle size = 64 B,x-stride in Kernel 1 = 64 B,x-stride in Kernel 2 = 8 B,iterations/block in Kernel 1 = 1,iterations/block in Kernel 2 = 8

(d) Counting only compulsory data-block fetches, for Kernel 1 each `Particle` occupies one full 64 B block, so 8192 iterations fetch:

$$8192 \text{ blocks}$$

For Kernel 2, each array has:

$$8192 \times 8 = 65536 \text{ B}$$

$$65536/64 = 1024 \text{ blocks per array}$$

Since the loop reads both `x` and `mass`:

$$1024 + 1024 = 2048 \text{ blocks}$$

Final answer: blocks in Kernel 1 = 8192,blocks in Kernel 2 = 2048

(e) From part (a), the effective penalty of one L1 miss is 24 cycles, so:

$$\text{CPI} = 1.0 + 0.40 \times m \times 24$$

For Kernel 1, $m = 0.18$:

$$\text{CPI}_1 = 1.0 + 0.40 \times 0.18 \times 24 = 2.728$$

For Kernel 2, $m = 0.06$:

$$\text{CPI}_2 = 1.0 + 0.40 \times 0.06 \times 24 = 1.576$$

Since $1.576 < 2.728$, Kernel 2 is better.

Final answer: CPI of Kernel 1 = 2.73,CPI of Kernel 2 = 1.58,better kernel = 2