Cache Miss Types

**Compulsory Miss:** Starting with an empty cache, a cache line is first referenced (invalid)

**Capacity Miss:** The cache is not big enough to hold every block you want to access

**Conflict Miss:** Two different addresses (blocks) are mapped to the same location and there is not enough to room (way) to hold both

Problem 1.

Consider a 2-way set-associative cache where each way has 4 cache lines with a block size of 2 words. Each cache line includes a valid bit (V) and a dirty bit (D), which is used to implement a write-back strategy. The replacement policy is least-recently-used (LRU). The cache is used for both instruction fetch and data (LD,ST) accesses. Please use this cache when answering questions (A) through (D).

(A) Using this cache, a particular benchmark program experiences an average memory access time (AMAT) of 1.3 cycles. The access time on a cache hit is 1 cycle; the miss penalty (i.e., additional access time) is 10 cycles. What is the hit ratio when running the benchmark program? You can express your answer as a formula if you wish:

\[
\text{Hit ratio for benchmark program: 0.97}
\]

\[
\text{AMAT = hit_time + (1 - hit_ratio) * miss_penalty}
\]

\[
1.3 = 1 + (1 - HR) \times 10
\]

(B) The circuitry for this cache uses various address bits as the block offset, cache line index and tag field. Please indicate which address bits A[31:0] are used for each purpose by placing a “B” in each address bit used for the block offset, “L” in each address bit used for the cache line index, and “T” in each address bit used for the tag field.

Fill in each box with “B”, “L”, or “T”

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>L</td>
<td>L</td>
<td>B</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Block Size = 2Words = 8Byte: Offset 3bits (2bits are 2'b00)
4 lines/each way = 4 sets cache: Index 2bits
Remaning bits are all tag bits: 27 bits
(C) This cache needs room to store new data and based on the LRU replacement policy has chosen the cache line whose information is shown to the right for replacement. Since the current contents of that line are marked as dirty (D = 1), the cache must write some information back to main memory. What is the address of each memory location to be written? Please give each address in hex.

Addresses of each location to be written (in hex): 0x2478, 0x27C

Block offset: 3-bit value: 0b100 or 0b000
Index: 2-bit 3 = 0b11
Tag: 27-bit 0x123 = 0b000 0000 0000 0000 0001 0010 0011
1) Block Offset (0b000): 0b000…00_0100_0111_1000 = 0x2478
2) Block Offset (0b100): 0b000…00_0010_0100_0111_1100 = 0x27C

(D) This cache is used to run the following benchmark program. The code starts at memory address 0; the array referenced by the code has its first element at memory address 0x200. First determine the number of memory accesses (both instruction and data) made during each iteration through the loop. Then estimate the steady-state average hit ratio for the program, i.e., the average hit ratio after many iterations through the loop.

```
. = 0
    mv x3, x0, // byte index into array
    mv x1, x0, // initialize checksum accumulator
loop:
    lw x2, 0x200(x3) // load next element of array
    slli x1, x1, 1 // shift checksum
    addi x1, x1, 1 // increment checksum
    add x1, x1, x2 // include data value in checksum
    add x3, x3, 4 // byte index of next array element
    slti x2, x3, 1000 // process 250 entries
    bnez x2, loop
    unimp // halt

. = 0x200
array:
... array contents here ...
```

@Steady state
Each loop fetches 7 instructions and 1 data (lw).
First two inst in loop mappes to index “1”, next two in index “2”, next two in index “3” and bnez in index “0” → They can occupy Way 0 → No instruction cache misses!

The other way can be used for data array starting at 0x200.
1 Cache miss will fill in 2 elements from the array (Cache block size = 2Words)
So, data fetch (lw) misses every other cycle: 50%
Steady-state hit ratio = (8-0.5)/8 = 15/16

Number of memory accesses made during each iteration of the loop: 8

Estimated steady-state average hit ratio: 15/16
Problem 2.

Consider the diagram to the right for a 2-way set associative cache to be used with our RISC-V processor. Each cache line holds a single 32-bit word of data along with its associated tag and valid bit (0 when the cache line is invalid, 1 when the cache line is valid).

(A) The RISC-V produces 32-bit byte addresses, A[31:0]. To ensure the best cache performance, which address bits should be used for the cache index? For the tag field?

- address bits used for cache index: A[4:2]
- address bits used for tag field: A[31:5]

1 word block: Block offset 2bits (all 0s)
8 lines/way: Index 3bits
Remaining 27bits are for tags.

(B) Suppose the processor does a read of location 0x5678. Identify which cache location(s) would be checked to see if that location is in the cache. For each location specify the cache section (A or B) and row number (0 through 7). E.g., 3A for row 3, section A. If there is a cache hit on this access what would be the contents of the tag data for the cache line that holds the data for this location?

- cache location(s) checked on access to 0x5678: 6A and 6B
- cache tag data on hit for location 0x5678 (hex): 0x2B3

0x5678 in binary: 0b 0101 0110 0111 1000
Line: 3'b110 = 6
Tag: 0b0010_1011_0011 = 0x2B3

(C) Assume that checking the cache on each read takes 1 cycle and that refilling the cache on a miss takes an additional 8 cycles. If we wanted the average access time over many reads to be 1.1 cycles, what is the minimum hit ratio the cache must achieve during that period of time? You needn’t simplify your answer.

\[ 1.1 = 1 + (1-HR)\times 8 \]

minimum hit ratio for 1.1 cycle average access time: 79/80
(D) Estimate the approximate cache hit ratio for the following program. Assume the cache is empty before execution begins (all the valid bits are 0) and that an LRU replacement strategy is used. Remember the cache is used for both instruction and data (LD) accesses.

```
  . = 0
  addi x4, x0, 0x100
  mv x1, x0
  lui x2, 1       // x2 = 0x1000
loop: lw x3, 0(x4)
  addi x4, x4, 4
  add x1, x1, x3
  addi x2, x2, -1
  bnez x2, loop
  sw x1, 0x100(x0)
  unimp       // halt
  . = 0x100
source:
  . = . + 0x4000 // Set source to 0x100, reserve 0x1000 words
```

**approximate hit ratio: 5/6**

During the loop execution (lw ~ bnez), they take up lines 3-7 of one way of the cache. The data load from 0x100 0x104 0x108 …. (0x1000 times = 4096 times) will cause conflicts every time.

Each iteration: 5 instruction fetches (hit) and 1 data fetch (always miss)

(E) After the program of part (D) has finished execution what information is stored in row 4 of the cache? Give the addresses for the two locations that are cached (one in each of the sections) or briefly explain why that information can’t be determined.

**Addresses whose data is cached in “Row 4”:** 0x10 and 0x40F0

(@ 0x10) addi x4,x4,4: Mapped to Row 4

Data access ends at 0x4100 - 4 (Starts from 0x100 and does 0x1000 times: next element at +4) = 0x40FC @ Row 7

0x40F8 @ Row6
0x40F4 @ Row5
0x40F0 @ Row4
Problem 3.

A standard unpipelined RISC-V is connected to a 2-way set-associative cache containing 8 sets, with a block size of 4 32-bit words. The cache uses a LRU replacement strategy. At a particular point during execution, a snapshot is taken of the cache contents, which are shown below. All values are in hex; assume that any hex digits not shown are 0.

(A) The cache uses bits from the 32-bit byte address produced by the processor to select the appropriate set (L), as input to the tag comparisons (T) and to select the appropriate word from the data block (B). For correct and optimal performance what are the appropriate portions of the address to use for L, T and B? Express your answer in the form “A[N:M]” for N and M in the range 0 to 31, or write “CAN’T TELL”.

4 word (16 Byte) cache block: 4-bit block offset (Lower 2 bits are 0s)
8 rows per way (8-set cache): 3-bit index

Address bits to use for L: [6:4]
Address bits to use for T: [31:7]
Address bits to use for B: [3:2] or [3:0]

(B) For the following addresses, if the contents of the specified location appear in the cache, give the location’s 32-bit contents in hex (determined by using the appropriate value from the cache). If the contents of the specified location are NOT in the cache, write “MISS”.

0xA1100: Tag 0x1422; Line 0; Word 0 (B0 or A0)
Contents of location 0xA1100 (in hex) or “MISS”: MISS

0x548: Tag 0x00A; Line 4; Word 2 (B2 or A2)
Contents of location 0x548 (in hex) or “MISS”: 0xDC000000

(C) Ignoring the current contents of the cache, is it possible for the contents of locations 0x0 and 0x1000 to both be present in the cache simultaneously?

2-way

Locations 0x0 and 0x1000 present simultaneously (circle one): YES … NO
(D) (Give a one-sentence explanation of how the D bit got set to 1 for Line #7 of Way #1. One sentence explanation

ST updated a value of a word in that line in cache, but hasn’t yet been written back to memory (Dirty).

(E) The following code snippet sums the elements of the 32-element integer array X. Assume this code is executing on a RISC-V processor with a cache architecture as described above and that, initially, the cache is empty, i.e., all the V bits have been set to 0. Compute the hit ratio as this program runs until it executes the unimp instruction, a total of 2 + (6*32) + 1 = 195 instruction fetches and 32 data accesses.

hit ratio: 216/227

```assembly
  . = 0
  mv x4, x0     // loop counter
  mv x1, x0     // accumulated sum

loop:
  slli x2, x4, 2 // convert loop counter to byte offset
  lw x3, 0x100(x2) // load next value from array
  add x1, x1, x3 // add value to sum
  addi x4, x4, 1 // increment loop counter
  slti x2, x4, 32 // finished with all 32 elements?
  bnez x2, loop // nope, keep going

  unimp // all done, sum in x1

  . = 0x100
X: .word 1     // the 32-element integer array X
   .word 2
   ...
   .word 32
```

Cache stores 4 words per line: First four instructions (mv, mv, slli, lw) map to cache index “0” and the other 4 instructions (add, addi, slti, bnez) map to cache index “1”. Unimp is mapped to index “3”

For 32 words in the array, 4 are grouped into a cache block and stored in another way of the cache (total 8 lines)

3 instruction misses and 8 data misses (Compulsory – initially cache empty)
No more misses
Total instruction fetches: 2 + 6*(32) + 1 = 195, total data fetches: 32

\[
\frac{(195+32-3-8)}{(195+32)}
\]
Problem 4.

After his geek hit single *I Hit the Line*, renegade singer Johnny Cache has decided he’d better actually learn how a cache works. He bought three RISC-V processors, identical except for their cache architectures:

- *Proc1* has a 64-line direct-mapped cache
- *Proc2* has a 2-way set associative cache, LRU, with a total of 64 lines
- *Proc3* has a 4-way set associative cache, LRU, with a total of 64 lines

Note that each cache has the same total capacity: 64 lines, each holding a single 32-bit word of data or instruction. All three machines use the same cache for data and instructions fetched from main memory.

Johnny has written a simple test progr

```plaintext
// Try a little cache benchmark
// Assume x7 = 0x2000 (data region A)
// Assume x8 = 0x3000 (data region B)
// Assume x9 = 16 (size of data regions in BYTES!)
.
  = 0x1000 // start program here
P:  addi x6, x0, 1000 // outer loop count
Q:  mv x3, x9 // Loop index i (array offset)
R:  addi x3, x3, -4 // i = i-1
   addi x9, x3, x7 // x9 = address of A[i]
   addi x10, x3, x8 // x10 = address of B[i]
   lw x1, 0(x9) // read A[i]
   lw x2, 0(x10) // read B[i]
   bnez x3, R
   addi x6, x6, -1 // repeat many times
   bnez x6, Q
   unimp // halt
```

Johnny runs his program on each processor, and finds that one processor model outperforms the other two.

(A) (2 points) Which processor model gets the highest hit ratio on the above benchmark?
3 regions: 0x1000, 0x2000, 0x3000

Circle one: Proc1 Proc2 Proc3

(B) (2 points) Johnny changes the value of B in his program to 0x2000 (same as A), and finds a substantial improvement in the hit rate attained by one of the processor models (approaching 100%). Which model shows this marked improvement?
2 regions

Circle one: Proc1 Proc2 Proc3

(C) (3 points) Finally, Johnny moves the code region to 0x0 and the two data regions A, and B each to 0x0, and sets x9 to 64. What is the TOTAL number of cache misses that will occur executing this version of the program on each of the processor models?

One region (code data) needs to fetch 16 elements once in all caches (Compulsory miss)

TOTAL cache misses running on Proc1: 16; Proc2: 16; Proc3: 16