Computer Design — Cache Memory

Computer Science (2nd year B.Sc.) — Seminar #8

1. Pre-check

This part is designed as a check to help you determine whether you understand the concepts covered in class. Answer "True" or "False" to the following questions and include an explanation:

1.1. For the same cache size and block size, a 4-associative cache will have fewer index bits than a direct-mapped cache.
True! A direct-mapped cache must index every row in the cache, while a 4-associative cache must index every set of 4 rows. The 4-associative cache will therefore have 2 index bits less than the direct-mapped cache.

1.2. When the cache is full, any cache miss that occurs is a "capacity miss".
False! When the cache is full, you can still have compulsory misses (i.e. when a block of data is first cached) and conflict misses.

1.3. Increasing the size of the cache by adding more lines always improves the success rate.
False! It depends on the program being run. For example, a loop that traverses an array of integers with a step of four (i.e. 16-bytes) will perform poorly on a cache with a line size of 8 bytes. Cache failures will be compulsory misses in this case, and there is no way to reduce the number of these failures simply by adding more lines to the cache.

2. Understanding T/I/O

Manipulating data with caches relies on ingenious reading/decomposition of the memory address into three different bit fields:

- **I**ndex – The index of the cache line where the memory block will be placed
  $$Width\ (in\ bits) = log2(number\ of\ cache\ lines)$$
- **O**ffset – The offset position of the byte in the memory block
  $$Width\ (in\ bits) = log2(size\ in\ bytes\ of\ a\ line\ of\ cache)$$
- **T**ag – Used to distinguish between different blocks of memory that may use the same cache line (i.e., same Index number).
  $$\begin{aligned} Width\ (in\ bits) &= Width\ of\ memory\ address\ (in\ bits) \\ &\quad - Width(Index) - Width(Offset) \end{aligned}$$

Thus, one can verify the following identity:

$$\begin{aligned} log2\ (Memory\ size\ in\ bytes) &= Width\ of\ memory\ address\ in\ bit \\ &= Width(Tag) + Width(Index) + Width(Offset) \end{aligned}$$

Another useful equality to remember is:

$$Size\ of\ Cache = Size\ of\ Memory\ Block \times Number\ of\ Cache\ lines$$

Note: In the lecture, the "Offset" field is further broken down into two sub-parts: The "word" field indicating the position of the word in a memory block and the "byte" field giving the index of the byte in the current word. In this tutorial, we limit ourselves to the "Tag / Index / Offset" decomposition of the memory address.

2.1. Assume a direct-mapped cache with a capacity of 32 bytes and a cache line of 8 bytes. For a 32-bit memory address, which bits correspond to the "Offset" field?
The number of bits in the "Offset" field depends only on the block/line size, so since our blocks are 8 bytes in size, we need $\log_2(8) = 3$ bits for the "Offset" field. The bits of the "Offset" field are the least significant in a memory address (i.e. starting at bit 0). Therefore, bits 0, 1 and 2 are used as "Offset" in a memory block.

2.2. Which bits should we check to find the line to use in cache?
Since this is a direct-mapped cache (i.e., 1-associative), you only need to determine the number of rows in this cache to find the range of indices. Using the equality above, we can see that "$Number\ of\ Cache\ lines = Size\ of\ Cache \div Size\ of\ Memory\ Block$". Thus, our cache has $32 \div 8 = 4$ rows and we need $\log_2(4) = 2$ bits to index 4 cache lines.

To identify the location of the "Index" field in a memory address, we need to determine the number of bits associated with the "Offset" field. Indeed, the bits of the "Offset" field are the least significant in a memory address (i.e. starting at bit 0). The bits of the "Index" field are the next most significant set of bits. Having 3 bits for the "Offset" field (cf. answer to question 2.1) means that the bits of the "Index" field start at bit 3, and so we use bits 3 and 4 for indexing the cache lines.

2.3. What about the "Tag" field?
The "Tag" field is defined by the remaining MSB bits of the memory address: bits 5 – 31.

2.4. Indicate for each of the following memory accesses whether it is a cache hit (H), a cache miss due to invalid entry (M), or a cache miss due to Replacement (R). **Hint:** Drawing a sketch of the cache can help you see the overrides more clearly.

Address	**T**	**I**	**O**	**H**it, **M**iss, **R**eplacement	
`0x00000004`	0	0	4	M,	Compulsory
`0x00000005`	0	0	5	H	
`0x00000068`	3	1	0	M,	Compulsory
`0x000000C8`	6	1	0	R,	Compulsory
`0x00000068`	3	1	0	R,	Conflict
`0x000000DD`	6	3	5	M,	Compulsory
`0x00000045`	2	0	5	R,	Compulsory
`0x00000004`	0	0	4	R,	Capacity
`0x000000C8`	6	1	0	R,	Capacity

Note that the M and R distinction here is purely theoretical. The cache does not behave differently in these cases.

3. Cache Associativity

To minimise cache misses due to “insufficient capacity” in a direct-mapped cache, we could simply increase the size of the cache line. This will ensure a better exploitation of the spatial locality principle, but the number of cache misses due to, for example, repetitive function calls will not be reduced – increasing the size of the cache line in a direct-mapped cache does not ensure a better exploitation of the temporal locality principle.

To take this principle into account in the cache design, we allow to associate (hence the name associativity) to a memory block several possible locations on a cache line. Thus, a cache is said to be N-associative when each memory block can go to N distinct locations on a cache line. For an N-associative cache, we have the rule:

$$N \times \mathit{Number\ of\ cache\ lines} = \mathit{Size\ of\ cache}\ (\mathit{in\ number\ of\ memory\ blocks})$$

3.1 Assume a 2-associative cache with a Least Recently Used (LRU) replacement policy and an 8-bit addressable memory. The size of the cache is 32 bytes and the size of a cache line is 8 bytes. Indicate for each of the following memory accesses whether it is a cache hit (H), a cache miss (M) due to invalid entry, or a cache miss due to Replacement (R).
Since the cache is 2-associative, there are 2 blocks in a cache line. The size of a memory block and the total cache size are the same as in question 2.1, so the cache has 4 blocks organized in $4/2 = 2$ lines. Therefore, we need $\log_2(2) = 1$ bit to index the 2 lines (i.e. 1 bit for the "Index" field). Since the block size is 8 bytes, the "Offset" field occupies 3 bits and the remaining bits are used as “Tag”.

Address	**T**	**I**	**O**	**H**it, **M**iss, **R**eplacement
`0x04`	0	0	4	M, Compulsory
`0x05`	0	0	5	H,
`0x68`	6	1	0	M, Compulsory
`0xC8`	12	1	0	M, Compulsory
`0x68`	6	1	0	H,
`0xDD`	13	1	5	R, Compulsory
`0x45`	4	0	5	M, Compulsory
`0x04`	0	0	4	H,
`0xC8`	12	1	0	R, Capacity

Again, bits 0, 1 and 2 of the address define the bits of the "Offset" field, the "Index" field is defined by a single bit (bit 3), bits 4 to 7 being the “Tag” field.

3.2 What is the hit rate for the previous question?
$\frac{3\ hits}{9\ accesses} = \frac{1}{3}$ hit rate

4. The three causes of cache misses

Review questions 2.4 and 3.1 and classify each cache miss as one of the 3 types of misses described below:

- **Compulsory misses**: A miss that must occur when a memory block is referenced for the first time. We can reduce first reference misses by having longer cache lines (i.e. larger blocks). We can also prefetch blocks in advance by using a special circuit to predict the next blocks that are likely to be required.
- **Capacity misses**: These misses are caused by the fact that the cache cannot hold all the blocks referenced during the execution of the program. The number of these misses can be reduced by increasing the size of the cache.
- **Conflict misses**: These misses occur in addition to the two previous types. A block already loaded into the cache is ejected from the cache because another block with the same index number is required by the program. The number of these misses can be reduced by increasing the associativity of the cache.

5. Source Code Analysis

Consider the C code below, running on a system with 1MB of memory and a 16KB direct cache organized in 1KB blocks.

```
1  #define NUM_INTS 8192    // pow(2,13)
2  int A[NUM_INTS];         // Assume A is allocated at address 0x10000
3  int i, total = 0;
4  for (i = 0; i < NUM_INTS; i += 128) {
5     A[i] = i;
6  }
7  for (i = 0; i < NUM_INTS; i += 128) {
8     total += A[i];
9  }
```

5.1 What is the width (in bits) of this system's memory address?

$\log_2(1\text{MB}) = \log_2(2^{20}) = 20$ bits.

5.2 Give the "T/I/O" decomposition of the address

Offset = $\log_2(1\text{KB}) = \log_2(2^{10}) = 10$ bits.

Index = $\log_2(\frac{16\text{ KB}}{1\text{ KB}}) = \log_2(16) = 4$ bits.

Tag = $20 - 4 - 10 = 6$ bits

5.3 Calculate the hit rate for line 5 of the C code

The elements of the integer array A are accessed with a step size of $128 \times \text{sizeof(int)} = 128 \times 4 = 512$ bytes. Since the starting address of A coincides with the beginning of a block (block #64 in this case), this means that every two accesses to A are to the same memory block. The first of the accesses in each block is therefore a compulsory cache miss, but the second access will be a hit. Thus, we will have a success rate of 50%.

5.4 Calculate the hit rate for line 8 of the C code

The size of array A is 8192 $\times$ sizeof(int) = 2^{15} = 32 KB. That's twice the size of our cache! This means that at the end of the first loop, we will have the second half of array A stored in cache. However, since the second loop starts again from the beginning of A, we won't be able to reuse any of the cache data imported in the first loop and have to start over from the beginning. So, our hit rate will again be the same as for line 5 (i.e. 50%) since we access the memory in exactly the same way in the second loop.

6. Average Memory Access Time (AMAT)

The "Average Memory Access Time" is defined by the following rule:

$$\text{AMAT} = \textit{hit access time} + \textit{miss rate} \times \textit{miss penality}$$

Hit access time = access time to data found in cache
Miss rate = number of cache misses ÷ number of cache accesses

For a hierarchical cache system, there are two miss rate metrics for each cache level:

- **Global miss rate**: The number of accesses to RAM divided by the total number of accesses to the **entire** cache system.
- **Local miss rate** for level L: The number of cache misses at level L divided by the number of total accesses to **this** level.

6.1 In a two-level cache system, there were 20 misses out of a total number of 100 accesses. What is the global miss rate?

$\frac{20}{100} = 20\%$ global miss rate.

6.2 If the L1 (i.e. level 1) cache has a 50% miss rate, what is the local miss rate of the L2 cache?

$\frac{20}{50\% \times 100} = 40\%$ local miss rate. Recall that the cache at level n+1 is accessed only when the cache at level n suffers a miss. So, if the L1 cache has a miss rate of 50% this means that we access the L2 cache 50 times.

For the following questions, consider a system with the following characteristics:

- An L1 cache with a hit access time of 2 clock cycles and a local miss rate of 20%.
- An L2 cache with a hit access time of 15 clock cycles and a global miss rate of 5%.
- A main memory with an access time of 100 clock cycles.

6.3 What is the local miss rate for the L2 cache?

$\frac{\textit{global miss rate}}{\textit{local miss rate of L1}} = \frac{5\%}{20\%} = \frac{1}{4} = 25\%$

6.4 Give the Average Memory Access Time (AMAT) of the system

AMAT $= 2 + 20\% \times 15 + 5\% \times 100 = 10\ cycles$, using the global miss rate.
Alternatively, AMAT $= 2 + 20\% \times (15 + 25\% \times 100) = 10\ cycles$.

6.5 We would like to reduce the system's AMAT to 8 clock cycles or less by adding a level 3 cache. If the L3 cache has a local miss rate of 30%, what is the highest response time this cache should have?

if S = *L3 cache hit access time*. Using the AMAT equation,

$$2 + 20\% \times \big(15 + 25\% \times (S + 30\% \times 100)\big) \leq 8\ cycles$$

Solving the above inequality gives: $S \leq 30\ cycles$. Therefore, the longest response time of the L3 cache is 30 cycles.