Computer Architecture I

Spring, 2022

Homework 6

ShanghaiTech University

Due: May 7, 2022, 23:59

Instructions

This homework will help you review CPU caches. Remember to go through the textbook and all the lecture slides related.

Submission Guideline

- Both hand-writing and typesetting are accepted. You may find a LaTeX template helpful on our course website.
- Only PDF submissions to Gradescope will be counted. If you choose to write by hand, make sure it is transformed into a PDF document. Both scanning and taking photos are accepted.
- Please assign your answers properly on Gradescope. Any submission without proper assignment will result in a 25% point reduction.

1 Shut Up and Take My Cache!

In this section, we will review some basics of cache. A program is run on a byte-addressed system with a single-level cache. After a while, the entire cache has the state in Figure 1.

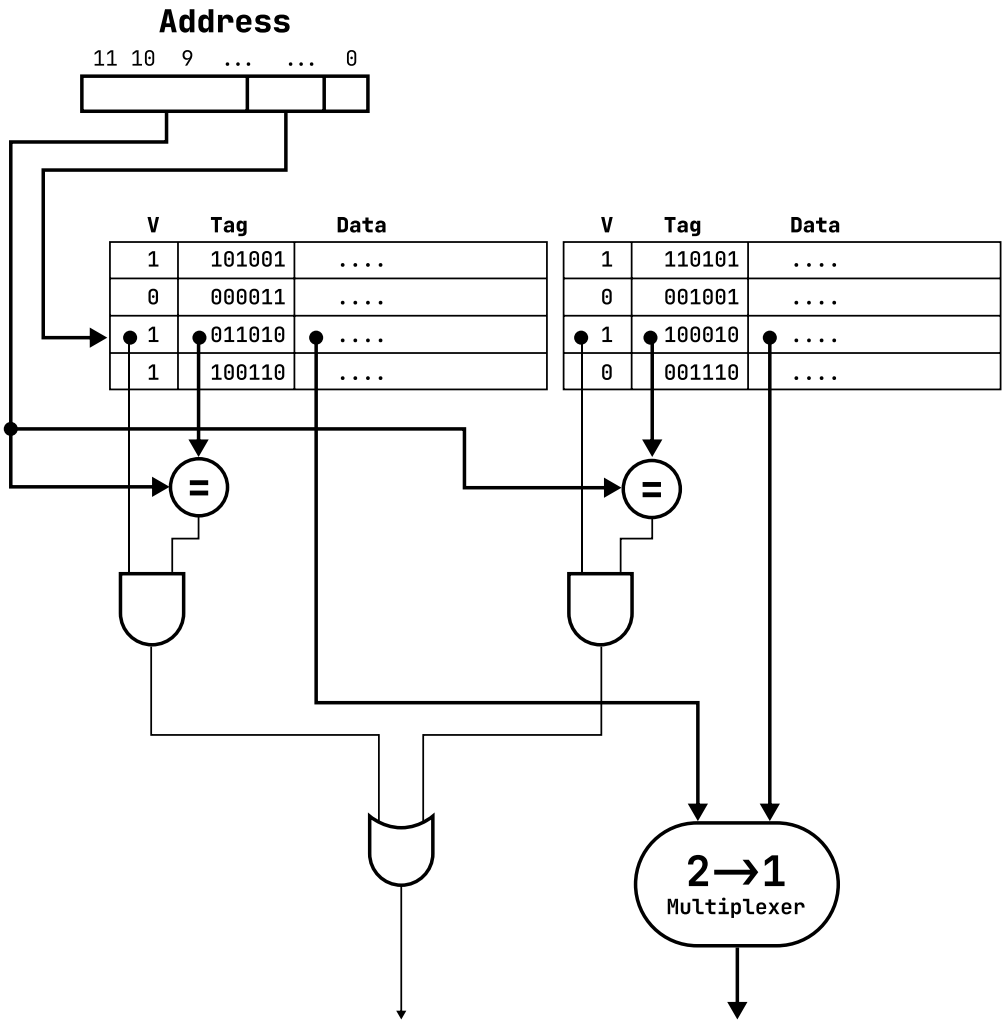


Figure 1: Layout for a cache implementation

1. (2 points) In Figure 1, what is a row stands for?
   A. One set. B. One cache block. C. One entire cache. D. Not listed in choices.

   Your answer: √ **A** ◯ B ◯ C ◯ D

2. (2 points) In Figure 1, what is `V` stands for?
   A. Valid bit. B. Vacant bit. C. Visited bit.

   Your answer: √ **A** ◯ B ◯ C

3. (2 points) What is the type of the cache described in Figure 1?
   A. Direct-Mapped cache.
   B. Set-Associative Cache (If you choose this, write down its associativity).
   C. Fully-Associative Cache (If you choose this, write down its associativity).

   Your answer:
   ◯ A
   √ **B (Associativity: 2)**
   ◯ C (Associativity: _____)

4. (3 points) What is the (Tag : Set Index : Byte Offset) breakdown of memory addresses in Figure 1?

   Your answer:
   1. Tag: **6**
   2. Index: **2**
   3. Byte Offset: **4**

5. (2 points) Is it TRUE that conflict misses cannot occur in fully-associated caches?
   A. Yes. B. No.

   Your answer: √ **A** ◯ B

6. (3 points) Tell the difference(s) between Conflict Miss and Capacity Miss.

Your answer:

- Conflict Miss is caused if multiple memory locations are mapped to the same cache location, while Capacity Miss is caused if cache cannot contain all blocks accessed by the program.
- If a cache miss occurs, we can go through the entire string of accesses with a fully associative cache with an LRU replacement policy. In this scenario, a cache hit indicates Conflict Miss, while a cache miss indicates Capacity Miss.

7. (12 points) For each of the following accesses to the cache described in Figure 1, determine if each access would be a hit or miss based on the cache state shown above. If it is a miss, classify the miss type(s).

   If multiple miss types may exist depending on prior memory accesses, select *all possible* miss types. For each access, you will get

   - 2 points if you choose all correct choice(s),
   - 1 point if you choose partial correct choice(s), and,
   - 0 point if you give no choice(s) or wrong choice(s).

   Each memory access should be considered *dependently*. In particular, *Do update* the cache status after each memory access. If a replacement happens, data in the first slot will always be evicted.

Order	Address	Access Outcome
1	`0b 101001 000100`	**1**
2	`0b 011010 110100`	**2**
3	`0b 111110 101000`	**3**
4	`0b 000011 111100`	**4**
5	`0b 000011 011001`	**5**
6	`0b 100110 101100`	**6**

Your answer:
Note: Each access may have one or more correct choice(s).

1. √ **Hit** ◯ Compulsory Miss ◯ Conflict Miss
2. ◯ Hit √ **Compulsory Miss** √ **Conflict Miss**
3. ◯ Hit √ **Compulsory Miss** √ **Conflict Miss**
4. ◯ Hit √ **Compulsory Miss** √ **Conflict Miss**
5. ◯ Hit √ **Compulsory Miss** ◯ Conflict Miss
6. ◯ Hit √ **Compulsory Miss** √ **Conflict Miss**

8. (6 points) The specification sheet of the system is given below. What is the Average Memory Access Time (AMAT) of memory accesses in Question 7? What is the AMAT if we remove this cache? Please give your answer in nanosecond (ns). You shall have the formula, the unit of results and the deriving procedure presented in your answer. (Note: A 1 gigahertz (GHz) processor ticks a cycle for each 1 nanosecond)

System Frequency	2 GHz
Cache Access Latency	2 Cycles
Main Memory Access Latency	600 Cycles

Table 1: Specification Sheet

Your answer:

- Clock Cycle = 1/System Frequency = 0.5ns
- Hit Time = Cache Access Latency = 2 Cycles = 1ns
- Miss Rate = 5/6
- Miss Penalty = Main Memory Access Latency = 600 Cycles = 300ns
- AMAT = Hit Time + Miss rate $\times$ Miss Penalty = $1 + 5/6 \times 300 = 251$ns

If cache is removed: AMAT = Main Memory Access Latency = 300ns

2 Oops ...Too many bites (bytes)

In this sections, we will review the implementation of caches and replacement policies.

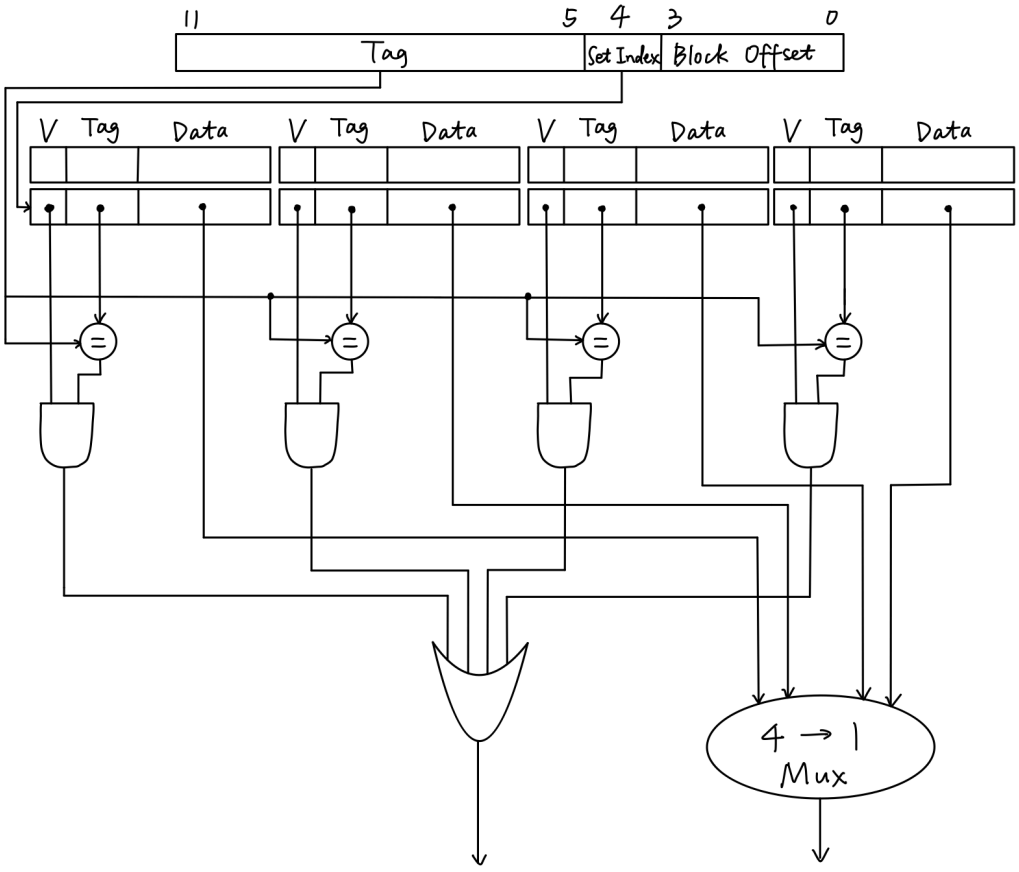
1. (15 points) Let's Draw It Out!

   Sketch the organization of a *four-way set associative* cache with a cache block size of 16 bytes and a total size of 128 bytes. Your sketch should have a style similar to Figure 1. The memory addresses are 12-bit long.

   In your sketch, the following components should be also presented.

   1. the width of set index, tag and data fields of memory addresses (3 points),
   2. logical components used for comparison and selection (2 points),
   3. the type of multiplexer (e.g. $2 \rightarrow 1$, $4 \rightarrow 1$, $8 \rightarrow 1$, $16 \rightarrow 1$, etc.) (1 point),
   4. the number of sets (1 point) , and,
   5. an implementation layout including wiring and placement of cache elements (8 points).

Your answer: The number of sets = 2



In the following questions, we will examine how replacement policies affect miss rate.

After the system is cold start, the address sequence of warm-up accesses is:
`0x2A, 0x3C, 0x7D, 0xCE, 0x5B, 0x01, 0x2C, 0x1D, 0x9B, 0x3E`.

After all warm-up accesses are done, the address sequence of follow-up accesses is:
`0x30, 0x40, 0x52, 0x44, 0x56, 0x48, 0x5A, 0x4C, 0x10, 0x3B, 0x5C, 0x30, 0x5E`.

2. (2 points) Which locality(s) can you observe in the follow-up accesses?

Your answer: Note: There may exist(s) one or more correct choice(s).
√ **Spatial locality** √ **Temporal locality**

3. (2 points) Assume *Least Recently Used* (LRU) replacement policy is applied. Circle out all access(es) with cache hit in the follow-up accesses.

Your answer: Circle the access(es) that meets a cache hit! (like this: (0xFF))

(0x30),0x40,(0x52),(0x44),(0x56),(0x48),(0x5A),(0x4C),(0x10),(0x3B),(0x5C),(0x30),(0x5E).

4. (2 points) Assume *Most Recently Used* (MRU) replacement policy is applied. Circle out all access(es) with cache hit in the follow-up accesses.

Your answer: Circle the access(es) that meets a cache hit! (like this: (0xFF))

(0x30),0x40,(0x52),(0x44),(0x56),(0x48),(0x5A),(0x4C),0x10,(0x3B),0x5C,0x30,0x5E.

5. (6 points) The specification sheet of the system is given below. What is the Average Memory Access Time (AMAT) of memory accesses in the question 3 and 4? Please give your answer in nanosecond (ns). You shall have the formula, the unit of results and the deriving procedure presented in your answer. (Note: A 1 gigahertz (GHz) processor ticks a cycle for each 1 nanosecond)

System Frequency	2 GHz
Cache Access Latency	2 Cycles
Main Memory Access Latency	130 Cycles

Table 2: Specification Sheet

Your answer:

- Clock Cycle = 1/System Frequency = 0.5ns
- Hit Time = Cache Access Latency = 2 Cycles = 1ns
- Miss Penalty = Main Memory Access Latency = 130 Cycles = 65ns

Q3:

- Miss Rate = 1/13
- AMAT = Hit Time + Miss rate × Miss Penalty = $1 + 1/13 \times 65 = 6$ns

Q4:

- Miss Rate = 5/13
- AMAT = Hit Time + Miss rate × Miss Penalty = $1 + 5/13 \times 65 = 26$ns

3 Let's See Some Real World Example

Each time when you access the course website, your activity will be recorded into our web server logs! This is the definition of the web server log for our Computer Architecture course website. Assume our web server is a 32-bit machine. In this question, we will examine code optimizations to improve log processing speed. The data structure for the log is defined below.
Note: Memory Alignment is considered in Problem 3

```
struct log_entry {
    int src_ip;    /* Remote IP address */
    char URL[128]; /* Request URL. You can consider 128 characters are enough. */
    // 4-bytes padding
    long reference_time; /* The time user referenced to our website. */
    char browser[64]; /* Client browser name */
    int status; /* HTTP response status code. (e.g. 404) */
    // 4-bytes padding
} log[NUM_ENTRIES];
```

Assume the following processing function for the log. This function determines the most frequently observed source IPs during the given hour that succeed to connect our website.

```
topK_success_sourceIP (int hour);
```

1. (2 points) Which field(s) in a log entry will be accessed for the given log processing function?

   Your answer: Note: There may exist(s) one or more correct choice(s).
   √ **src_ip** ◯ URL √ **reference_time** ◯ **browser** √ **status**

2. (1 point) Assuming 32-byte cache blocks and no prefetching, how many cache misses per entry does the given function incur on average?

   Your answer: **2.25 cache misses**

3. (3 points) How can you reorder the data structure to improve cache utilization and access locality? Justify your modification.

   Your answer:

   ```
   struct log_entry {
       int src_ip;    /* Remote IP address */
       int status; /* HTTP response status code. (e.g. 404) */
       long reference_time; /* The time user referenced to our website. */
       char URL[128]; /* Request URL. You can consider 128 characters are enough. */
       char browser[64]; /* Client browser name */
   } log[NUM_ENTRIES];
   ```

   Justification:
   Such layout guarantees that there is no padding among any entries and any member variables. For the first time, we access `src_ip`, `status`, `reference_time` and the first 16 bytes of `URL[128]` of the

1st entry. Next, we access the last 16 bytes of `browser[64]` of the 1st entry, `src_ip`, `status` and `reference_time` of the 2nd entry, with `URL[128]` and `browser[64]` of the 2nd entry being ignored. Next time the access starts at the beginning of the 3rd entry. After that, the procedure described above will be executed over and over again, with 2 entries being accessed within 2 misses. Therefore, **the average misses per entry is 1.**

4. (6 points) To mitigate the miss in the question 2, design a different data structure. How would you rewrite the program to improve the overall performance?

   Your answer shall include:

   - A new layout of data structure of our server logs.
   - A description of how your function would improve the overall performance.
   - How many cache misses per entry does your improved design incur on average?

Your answer:

Layout:

```
struct log_entry_useful {
    int src_ip;     /* Remote IP address */
    int status; /* HTTP response status code. (e.g. 404) */
    long reference_time; /* The time user referenced to our website. */
} log[NUM_ENTRIES];

struct log_entry_useless {
    char URL[128]; /* Request URL. You can consider 128 characters are enough. */
    char browser[64]; /* Client browser name */
} log[NUM_ENTRIES];
```

Description:
If we separate the accessed fields and unaccessed fields into two parts, the overall performance can be highly improved. Since the total size of `log_entry_useful` is only 16 bytes and the block size is 32 bytes, two entries can be loaded once there is a cache miss. Therefore, only consider the accessed fields, after the first cache miss 6 fields in total can be loaded to the cache, so all the following 5 references (the remaining accesses of two adjacent entries) will incur cache hits, which performs much better than the original layout or the reordered layout, whose data structure contains useless but large fields to prevent successive cache hits.

Calculation:
There is only 1 cache miss out of 6 consecutive cache references (in terms of field), namely two consecutive entry accesses. So **the average cache misses per entry is 0.5**.

One more thing...

Phew! That's all about cache! Tell us your feeling after finish this homework. Thank You! Don't worry if you did not assign this to Gradescope. This part is *optional*.

1. (0 points) How do you *feel* after you have done this homework?

   Your answer:

   √ **:) I feel good.**

   ◯ :(I feel bad.

2. (0 points) Do you think this homework is hard for you?

   Your answer:

   ◯ It's really difficult for me!

   ◯ I think it is a little bit challenging.

   √ **I am okay with that.**

   ◯ This is too easy for me.

3. (0 points) Your voice will be secretly heard by our team. Is there anything you want to feedback?

   Your answer:

   1. This homework is not problematic at all, good job!
   2. However, as for course projects, I think they should be more practical, complicated, integrated and well correlated to the real-world programs, rather than simply implementing a boring RV32I-RVC bidirectional translator or building a CPU. In fact if you really want us to do so, set them to be homework may be better.

Thank you! Now you are clear with cache! :)