

# CSCI 2510 Computer Organization 2020-21

## Assignment 3

**Deadline: December 1, 2020 (TUE) 14:30pm**

### Submission Notes:

- (1) For each of the following written exercises (*Questions 1~2*), please show your steps and explain in detail when needed to receive full credit.
- (2) Submit **four** files named **assignment3.pdf** (for *Questions 1~2*), **set\_associative\_ex1.asm** and **set\_associative\_ex2.asm** (for *Programming Exercise*), and **set\_associative\_report.pdf** (for *Programming Exercise Report*) to [Blackboard](#) before the deadline (**14:30pm on Dec 1**).
- (3) Late submission is **not** acceptable.

---

### Question 1 (30 pts)

---

Consider a computer uses 32-bit byte-addressable memory addresses. Assume it has a memory of 1G bytes and has a one-level cache of 1K bytes with 64 bytes per block and 4 bytes per word.

- (a) Derive the number of bits required by different fields (i.e., Tag, Block, Set, Word, and Byte) for the following types of cache mapping functions: 1) direct mapping, 2) associative mapping, and 3) two-way set-associative mapping. (15pts)
- (b) Suppose that the processor fetches 320 consecutive words starting at memory location 0, and repeats this fetch sequence for one more time. Assume that the cache is initially empty, derive the cache hit rate(s) for the following types of cache mapping functions: 1) direct mapping, 2) associative mapping (with the LRU replacement algorithm), and 3) two-way set-associative mapping (with the LRU replacement algorithm). (15pts)

---

### Question 2 (20 pts)

---

Suppose that a computer has a processor with two L1 caches, one for instructions and one for data, and one L2 cache. Let  $\tau$  be the access time for two L1 caches,  $10\tau$  be the access time for L2 cache, and  $100\tau$  be the access time for the main memory (i.e., transfer a block of data from main memory to the L2 cache). Assume the hit rate in L1 cache is 0.95 (for both instruction and data caches), the hit rate in L2 cache is 0.80, and the load through technique is not used.

- (a) What fraction of accesses miss in both the L1 and L2 caches, thus requiring access to the main memory? (5pts)
- (b) What is the average memory access time as seen by the processor? (5pts)
- (c) What hit rate of L1 Cache would be needed for reducing the average memory access time to  $1.5\tau$ , if all other parameters are the same? (5pts) Can the same the average memory access time (i.e.,  $1.5\tau$ ) be achieved by improving the hit rate of L2 cache? (5pts)

---

## Programming Exercise (50 pts)

---

*Set-associative mapping* is a mapping function that combines the designs of *direct mapping* and *associative mapping*. In this programming exercise, we are going to implement a **four-way set-associative mapping** using MASM IA-32 assembly language by simulating the arrays of cache tags (CacheTags) and timestamps (CacheTimes) in memory as variables. For the sake of simplicity, every element in these two arrays is defined to be 4 bytes (**dword**). In our implementation, there are a total of 128 cache blocks, where each cache block is of 16 words and each word is of 16 bits; moreover, the first-in-first-out (FIFO) replacement algorithm is adopted, which means that the first memory block cached in a set shall also be the first one to be replaced. In addition, each input value will be a 16-bit unsigned value (< 65536) and treated as the requested byte-address of memory.

*Note 1: It's not allowed to define additional variables in ".data".*

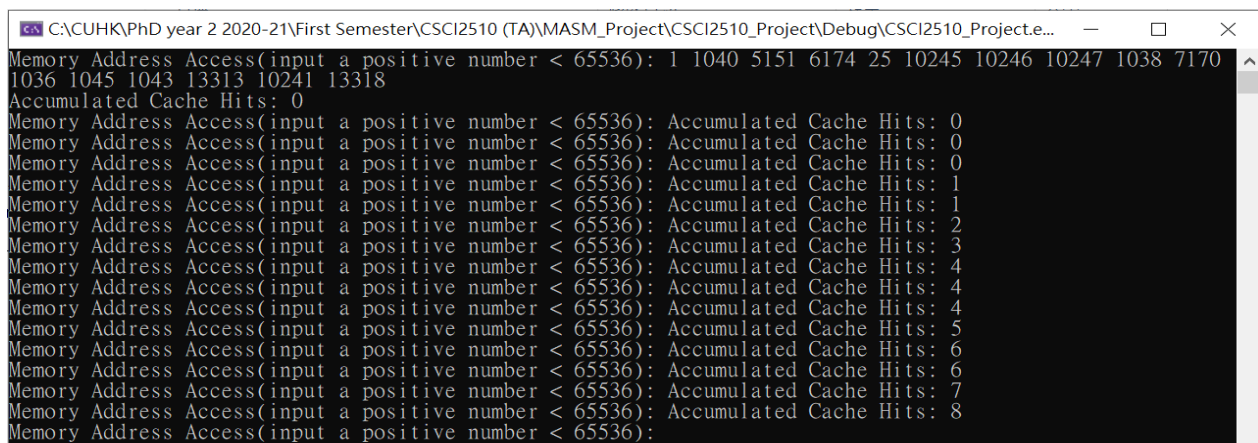
*Note 2: Submit two .asm files for Exercise 1 and Exercise 2 separately.*

*Note 3: Referencing the materials from Tutorials 07~09 is strongly advised.*

### Exercise 1 (40 pts)

Complete the provided MASM IA-32 assembly program named **set\_associative.asm** to implement a **four-way set-associative mapped cache**. To test the correctness of your program, you are required to record the total number of cache hits and print out the cumulative number of cache hits for every input value (as shown in the following screenshots). A file of testing input named **test\_input.txt** is also provided with the correct number of cache hits. Please paste the screenshots of your results in the report.

*Note: If you use arithmetic instruction(s) such as **div** and **mul** (including **idiv** and **imul**) for the implementation, you will receive a 15-point deduction. Instead, please use the bit-wise instructions (such as **and**, **shl** and **shr**) to receive the full marks.*



```
C:\CUHK\PhD year 2 2020-21\First Semester\CSCI2510 (TA)\MASM_Project\CSCI2510_Project\Debug\CSCI2510_Project.e...
Memory Address Access(input a positive number < 65536): 1 1040 5151 6174 25 10245 10246 10247 1038 7170
1036 1045 1043 13313 10241 13318
Accumulated Cache Hits: 0
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 0
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 0
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 0
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 1
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 1
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 2
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 3
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 4
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 4
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 4
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 5
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 6
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 6
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 7
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 8
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 8
```

Figure: Outputs of the first testing input.

```

C:\CUHK\PhD year 2 2020-21\First Semester\CSCI2510 (TA)\MASM_Project\CSCI2510_Project\Debug\CSCI251...
Memory Address Access(input a positive number < 65536): 997 2021 11238 21476 998 22500 22502 999
2024 21476 998 2047 22527
Accumulated Cache Hits: 0
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 0
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 0
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 0
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 1
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 1
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 2
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 2
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 2
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 3
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 4
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 5
Memory Address Access(input a positive number < 65536): Accumulated Cache Hits: 6

```

Figure: Outputs of the second testing input.

## Exercise 2 (10 pts)

There are four main functions defined in the `set_associative.asm`, such as `SearchSets`, `SearchTags`, `CacheHit`, and `CacheMiss`. However, in the current implementation, we are using `jmp` instruction to invoke them, making us hard to understand and maintain the program. To solve this issue, please convert the four main functions in `set_associative.asm` into subroutines and modify the “input” section of the program as the following screenshot. Please use the same file of testing input named `test_input.txt` to test the correctness of your program, and paste the screenshots of your results in the report.

input:

```

mov EAX, CurrentTimeStamp
inc EAX
mov CurrentTimeStamp, EAX ; update the time stamp
mov EBP, offset CacheTags ; Hint: EBP could be updated in the process, so initialize here
mov ESI, offset CacheTimes ; Hint: ESI could be updated in the process, so initialize here
invoke crt_printf, addr InputStatement
invoke crt_scanf, addr MemoryAddressFormat, addr MemoryAddress
mov EAX, MemoryAddress ; EAX stores the input, which is the requested memory address
; Passing the parameters to SearchSets
call SearchSets
; Getting the results from SearchSets
; Passing the parameters to SearchTags
call SearchTags
; Getting the results from SearchTags
; if cache hit
call CacheHit
; if cache miss
call CacheMiss
jmp PrintCacheHits

```

*Note 1: Exercise 2 will not be graded if Exercise 1 fails*

*Note 2: You are advised to define the region of a subroutine by `proc` and `endp`.*

*Note 3: You can use either registers or processor stack to pass the parameters.*