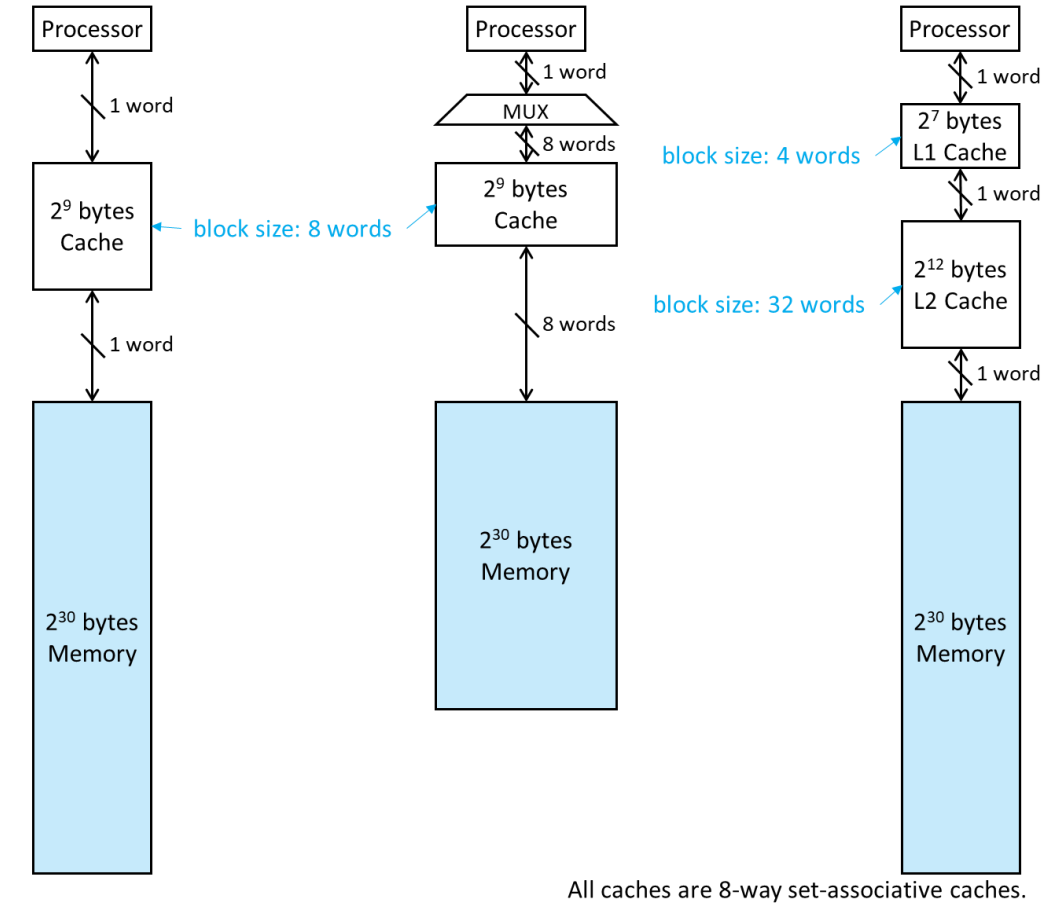
Computer Organization, Spring 2018

Lab 5: Cache Performance

Due: 2018/7/2

1. Goal

Cache Performance can be improved by two techniques: reducing the miss rate or reducing the miss penalty. Lab 4 (Cache Simulator) focuses on reducing the miss rate by reducing the probability that two different memory blocks will contend for the same cache location. In this lab, cache performance is going to be improved by reducing the miss penalty, and further measured by considering program execution cycles and memory stall cycles. You should simulate both CPU and cache behaviors with C/C++ style simulators with given timing assumptions. This Lab will help you understand the impact of cache performance.



a. One-word-wide memory organization b. Wider memory organization c. Two-level memory organization

Fig 1. Three different kinds of memory organization for Lab 5 implementation

2. Requirement (80%)

(1) Please attach your names and student IDs as comment at the top of each file.

(2) Please modify your "direct_mapped_cache_lru.cpp" in Lab 4 to complete this Lab, and provide Makefile to compile your source codes into executable file named **simulate_caches**.

(3) You are asked to simulate the single-cycle CPU and cache behaviors of a function for matrix multiplication: *matmul*(*A*, *B*), where both *A* and *B* are matrices with their dimensions to be $m \times n$ and $n \times p$, respectively. The assembly code of *matmul*(*A*, *B*) is given as matmul.txt. According to matmul.txt and following delay assumptions, you can calculate and analyze the performance.

Operation	Delay (cycle)
Send the address	1
Access single cache content (Fig. 1(a)(b))	2
Access L1 cache content	1
Access L2 cache content	10
Access memory content	100
Send a word of data	1

For example, considering the memory organization shown in Fig 1(c), if there is a miss in both L1 and L2, a 32-word block will be transferred from memory to L2, a 4-word block will be transferred from L2 to L1, and the required data will be sent to CPU with 1+32×(1+100+1+10)+4×(1+10+1+1)+1+1 = 3311 memory stall cycles.

3. Input/output

The first line contains three hexadecimal numbers $ADDR_0$, $ADDR_1$, and $ADDR_2$, which indicate the base addresses of input matrices *A*, *B* and output matrix *C*, and three decimal integers, *m*, *n*, and *p* (*m*, *n*, *p* are 2^x, $2 \leqq x \leqq 10$), which indicate the dimensions of matrices $A_{m\times n}$ and $B_{n\times p}$, respectively. In the following *m*+*n* lines are the elements of matrices *A* and *B*. Your output should contain the result matrix $C_{m\times p}$ and the simulated program execution cycles and the total memory stall cycles according to the calculation of miss penalty in p.33-34 of Chapter 5 slides.

Sample Input: a1xb1	Sample Output
ffff0040 ffff00c0 ffff0180 4 4 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ($A_{m\times n}$) 1 2 3 3 4 5 6 6 7 8 9 9 10 11 12 12 ($B_{n\times p}$)	70 80 90 90 158 184 210 210 246 288 330 330 334 392 450 450 ($C_{m\times p}$) 1553 6022 1648 11160 total memory stall cycles of cache in Fig 1(a), Fig 1(b), and Fig 1(c), respectively simulated program execution cycles

4. Report (20%)

Briefly write down how you calculate the memory stall cycles of different miss condition. Based on the results of CPU and cache simulation, compare and discuss the difference among the three memory organizations described in Fig 1.

5. Bonus: Performance improvement by software (15%)

The processor performance can be further improved by software, i.e., compiler. To get the bonus, you should rewrite the assembly code of *matmul*(*A*, *B*) as **bonus_matmul.txt** and calculate the new memory stall cycles with your C/C++ style simulator. The reasons why your *matmul*(*A*, *B*) is faster than the original one should be written down in the report. Note that the functionality of your assembly code should be the same as the original one, i.e., it should calculate the correct matrix multiplication result $C_{m\times p}$, which would be verified by TA's simple MIPS assembler (only includes **addi**, **addu**, **subu**, **mul**, **slt**, **beq**, **bne**, **j**, **lw**, **sw**, please see bonus_readme.txt) and single-cycle CPU. Besides, every instruction has its delay penalty and the total delay should be recalculated and written into the report!

6. Grade

(1) Total: 115 points, including 20 points for a report
(2) Late submission: 10 points off per day
(3) No plagiarism, or you will get 0 point.

7. Hand in

(1) Zip your folder and name it as "ID1_ID2.zip" (e.g., 0516001_0516002.zip) before uploading to e3. Other filenames and formats such as *.rar and *.7z are NOT accepted! Multiple submissions are accepted, and the version with the latest time stamp will be graded.
(2) Please include ALL source codes (*.c or *.cpp), Makefile and your report (*.doc or *.pdf) in the zipped folder.

8. How to test

Compile your source codes into executable file named simulate_caches: `make`
Test your program: ./simulate_caches [input_filename] [output_filename]
e.g., `./simulate_caches a1xb1 c1`

9. Q&A

For any questions regarding Lab 5, please contact 黃甯琪 (blackitty321@gmail.com).