CS211 Computer Architecture II

Multithreading and Memory Consistency

Assigned 10/11/2022 Homework #3 *Due 23:59:59 27/11/2022*

https://toast-lab.sist.shanghaitech.edu.cn/courses/CS211@ShanghaiTech/Fall-2022/

The homework is intended to help you learn the material, and we encourage you to collaborate with other students and to ask questions in discussion sections and office hours to understand the problems. However, each student must turn in their own solution to the problems.

The homework also provides essential background material for the mid-term and final exams. It will be graded primarily on an effort basis, but if you do not work through it, you are unlikely to succeed on the mid-term and final exams! We will distribute solutions to homework assignment. Note that each homework assignment is due at its respective due time, and all assignments are to be submitted **in English**. However, late submissions will **NOT** be accepted, except for extreme circumstances and/or with prior arrangement.

Name: 俞政宏

ID: 2020533156

Problem 1: Multithreading (1.8 Points)

Your old friend Li Hua has made a simple RISC-V-like program (we can call it "foo") below:

```
Loop:                       # x1 and x2 are already loaded
      LW x3, 0(x1)          # x1 is 0 in the beginning
      LW x4, 4(x1)
      LW x5, 8(x1)
      LW x6, 12(x1)
      ADD x5, x4, x3
      SUB x6, x4, x3
      ADDI x1, x1, 16
      BLT x1, x2, Loop      # x2 is a large integer
End:
```

Assume initially x1 is 0 and x2 is a large integer number. We run this program on a single-issue in-order processor. The processor can fetch and issue (dispatch) one instruction per cycle. If an instruction cannot be issued due to a data dependency, the processor stalls. **ADD, SUB, ADDI and BLT take one cycle to execute and the result can be used in the next cycle.** For example, if ADD and SUB are executed in cycle 1, ADDI can be executed in cycle 2. We also assume that the processor has a perfect branch predictor with no penalty for both taken and not-taken branches.

Problem 1.1 (0.4 Points)

Assume that our system does not have a cache. Each memory operation directly accesses main memory and takes 40 CPU clock cycles. The load/store unit is fully pipelined, and non-blocking. After the processor issues a memory operation, it can continue executing instructions until it reaches an instruction that is dependent on an outstanding memory operation. How many cycles does it take to execute one iteration of the loop in steady state? Fill out the following table and find the answer.

Instruction	Start Cycle	End Cycle
`LW x3, 0(x1)`	0	40
`LW x4, 4(x1)`	1	41
`LW x5, 8(x1)`	2	42
`LW x6, 12(x1)`	3	43
`ADD x5, x4, x3`	43	43
`SUB x6, x4, x3`	44	44
`ADDI x1, x1, 16`	45	45
`BLT x1, x2, Loop`	46	46

Total Cycle Number:	46

Problem 1.2 (0.7 Points)

Now we add zero-overhead multithreading to our pipeline. A processor executes multiple threads, each of which runs an independent "foo" program. Hardware mechanisms schedule a thread to execute each cycle. In our first implementation, the processor switches to a different thread every cycle using **fixed round robin scheduling**. Each of the N threads executes one instruction every N cycles. What is the minimum number of threads that we need to fully utilize the processor, i.e., execute one instruction per cycle?

If we have N threads, the first load execute at cycle 0. LW X3, 0(X1)
the second load execute at cycle 1. LW. X4, 4(X1)
the third load execute at cycle 2 LW. X3, 8(X1)

ADD which depend on the latest ins at cycle 2. excutes at $4 \cdot N$

To fully utilize the procesor, we need to hide 40-cycle memory lantency

$$\begin{cases} 4N \geq 40 \\ 4N-1 \geq 40 \\ 4N-2 \geq 40 \end{cases} \Rightarrow N_{min} = 11$$

the minimum number of thread needed is 11

Problem 1.2 (0.7 Points)

We change the processor to only switch (immediately) to a different thread when an instruction cannot execute due to data dependency. What is the minimum number of threads to fully utilize the processor now? Note that the processor issues instructions in-order in each thread.

If we have N threads, the first load execute at cycle 0 LW. X3, 0(X1)
then processor switch to another thread
next ins executes at cycle N
To fully utilize the processor, we need to hide 40-cycle memory lantency
We can start at add
$8N \geq 40$ the minimum number of thread needed is 5

Problem 2: Memory Consistency (1.2 points)

This time, Li Hua is trying to figure out the impact of different memory consistency. Assume there are two processes P1 and P2 running on two different processors, and the memory locations A, B, and C contains initial value 1. Please help Li Hua to solve the following questions with different memory consistency.

P1	P2
P1.1: ST(C) ← 0 P1.2: LD RB ← (B) P1.3: ST(A) ← 0	P2.1: ST(B) ← 0 P2.2: LD RC ← (C) P2.3: LD RA ← (A)

Problem 2.1 (0.4 Points)

Circle the results that could occur within the following possible final values of **(RA, RB, RC)** if the system is **Sequentially Consistent (SC)**. Please give a brief **explanation** for your answer.

P2.1, P1.1, P1.2, P2.2, P1.3, P2.3 | Possible. (0, 0, 0)

Possible (0, 1, 0)

P1.1, P1.2, P2.1, P2.2, P1.3, P2.3 ...

Possible (1, 0, 0)

P1.1, P2.1, P1.2, P2.2, P2.3, P1.3 ...

(1, 1, 0)

P1.1, P1.2, P2.1, P2.2, P2.3, P1.3

Possible

P2.1, P2.2, P1.1, P1.2, P1.3, P2.3

(0, 0, 1) Possible

(0, 1, 1) Not possible. No such sequence

P2.1, P2.2, P1.1, P1.2, P2.3, P1.3

(1, 0, 1) Possible

(1, 1, 1)

Not possible. No such sequence

A P1.3 P2.3 B. P1.2 P2.1 C P2.2 P1.1

Problem 2.2 (0.4 Point)

Circle the results that could occur within the following possible final values of **(RA, RB, RC)** if the system is **Total Store Order (TSO)**. Please give a brief **explanation** for your answer.

P2.1, P1.1, P1.2, P2.2, P1.3, P2.3 | Possible (0, 0, 0)

P1.1, P1.2, P2.1, P2.2, P1.3, P2.3 | Possible (0, 1, 0)

P1.1, P2.1, P1.2, P2.2, P2.3, P1.3 | Possible (1, 0, 0)

P1.1, P1.2, P2.1, P2.2, P2.3, P1.3 | Possible (1, 1, 0)

P2.1, P2.2, P1.1, P1.2, P1.3, P2.3

(0, 0, 1) Possible

P1.2, P2.2, P1.1, P2.1, P1.3, P2.3

relax

(0, 1, 1) Possible

(1, 0, 1) P2.1, P2.2, P1.1, P1.2, P1.3, P2.3 Possible

(1, 1, 1)

P1.2, P2.2, P1.1, P2.1, P1.3, P2.3

relax Possible

Problem 2.3 (0.4 Point)

Li Hua wants to make **TSO system** produces the same register values as the **SC system** by adding the **minimum** number of *barrier instruction*s. *barrier* is used to guarantee all the instructions before the *barrier* can be executed before it without being reordered. Please put *barrier(s)* in the following program and give a brief **explanation** for your answer.

P1	P2
P1.1: ST(C) ← 0 fence w,r // Read after write fence P1.2: LD RB ← (B) P1.3: ST(A) ← 0	P2.1: ST(B) ← 0 fence w,r // Read after write fence P2.2: LD RC ← (C) P2.3: LD RA ← (A)

This two fence will make sure P1.1, P1.2 / P2.1, P2.2 will not reorder due to TSO, which cause difference between TSO and SC