Last updated:
3/10/2020

Problem M10.1: Multithreading

Problem 10.1.A

Since there is no penalty for conditional branches, instructions take one cycle to execute unless there is a dependency problem. The following table summarizes the execution time for each instruction. From the table, the loop takes **104 cycles** to execute.

Instruction	Start Cycle	End Cycle
`LW R3, 0(R1)`	1	100
`LW R4, 4(R1)`	2	101
`SEQ R3, R3, R2`	101	101
`BNEZ R3, End`	102	102
`ADD R1, R0, R4`	103	103
`BNEZ R1, Loop`	104	104

Problem M10.1.B

If we have N threads and the first load executes in cycle 1, `SEQ`, which depends on the load, executes in cycle 2N + 1. To fully utilize the processor, we need to hide the 100-cycle memory latency, 2N + 1 101. The minimum number of thread needed is **50**.

Problem M10.1.C

	Throughput	**Latency**
Better	✓	
Same		
Worse		✓

Problem M10.1.D

In steady state, each thread can execute 6 instructions (`SEQ`, `BNEZ`, `ADD`, `BNEZ`, `LW`, `LW`). Therefore, to hide 99 cycles between the second `LW` and `SEQ`, a processor needs $\lceil 99/6 \rceil + 1 = 18$ threads.

Last updated:
3/10/2020

Problem M10.2: Multithreaded architectures

Problem M10.2.A

4, since the largest latency for any instruction is 4.

Problem M10.2.B

2/12 = 0.17 flops/cycle, on average we complete a loop every 12 cycles

Problem M10.2.C

Yes, we can hide the latency of the floating point instructions by moving the add instructions in between floating point and store instructions – we'd only need 3 threads. Moving the third load up to follow the second load would further reduce thread requirement to only 2.

Last updated:
3/10/2020

Problem M10.3: Multithreading

Problem M10.3.A

Fixed Switching: <u>6</u> **Thread(s)**

If we have N threads and L.D. executes in cycle 1, FADD, which depends on the load executes in cycle 2N + 1. To fully utilize the processor, we need to hide 12-cycle memory latency, 2N + 1 13. The minimum number of thread needed is 6.

Data-dependent Switching: <u>4</u> **Thread(s)**

In steady state, each thread can execute 4 instructions (FADD, BNE, LD, ADDI). Therefore, to hide 11 cycles between ADDI and FADD, a processor needs 11/4 + 1 = 4 threads.

Problem M10.3.B

Fixed Switching: <u>2</u> **Thread(s)**

Each FADD depends on the previous iteration's FADD. If we have N threads and the first FADD executes in cycle 1, the second FADD executes in cycle 4N + 1. To fully utilize the processor, we need to hide 5-cycle latency, 4N + 1 6. The minimum number of thread needed is 2.

Data-dependent Switching: <u>2</u> **Thread(s)**

In steady state, each thread can execute 4 instructions (FADD, BNE, LD, ADDI). Therefore, to hide 2 cycles between ADDI and FADD, a processor needs 2/4 + 1 = 2 threads.

Last updated:
3/10/2020

Problem M10.3.C

Consider a **Simultaneous Multithreading (SMT)** machine with limited hardware resources. **Circle** the following hardware constraints that can limit the total number of threads that the machine can support. For the item(s) that you circle, **briefly describe** the minimum requirement to support **N** threads.

(A) Number of Functional Units	Since not all the treads are executed in each cycle, the number of functional unit is not a constraint that limits the total number of threads that the machine can support.
[✓ **(B)** Number of Physical Registers]	We need at least [N (number of architecture registers) + 1] physical registers.
(C) Data Cache Size	This is for performance reasons.
(D) Data Cache Associatively	This is for performance reasons.

Problem M10.4: Multithreading (Spring 2015 Quiz 2, Part D)

Consider the following instruction sequence.

```
      addi   r3, r0, 256
loop: lw     f1, r1, #0
      lw     f2, r2, #0
      mul    f3, f1, f2
      sw     f3, r2, #0
      addi   r1, r1, #4
      addi   r2, r2, #4
      addi   r3, r3, #-1
      bnez   r3, loop
```

Assume that memory operations take 4 cycles (i.e., if instruction I1 starts execution at cycle N, then instructions that depend on the result of I1 can only start execution at or after cycle N+4); multiply instructions take 6 cycles; and all other operations take 1 cycle. Assume the multiplier and memory are pipelined (i.e., they can start a new request every cycle). Also assume perfect branch prediction.

Problem M10.4.A

Suppose the processor performs fine-grained multithreading with fixed round-robin switching: the processor switches to the next thread every cycle, and if the instruction of the next thread is not ready, it inserts a bubble into the pipeline. What is the minimum number of threads required to fully utilize the processor every cycle while running this code?

6 threads to cover the latency between `mul` and `sw`

Last updated:
3/10/2020

Problem M10.4.B

Suppose the processor performs coarse-grained multithreading, i.e. the processor only switches to another thread when there is a L2 cache miss. Will the following three metrics increase or decrease, compared to fixed round-robin switching? Use a couple of sentences to answer the following questions.

1) Compared to fixed round-robin switching, will the **number of threads needed for the highest achievable utilization** increase or decrease? Why?

It will decrease because the processor will switch less frequently and stall for instructions with long latency (e.g. mul).

2) Compared to fixed round-robin switching, will the **highest achievable pipeline utilization** increase or decrease? Why?

It will decrease because the processor will stall for instructions with long latency (e.g. mul) and insert bubbles into pipeline.

3) Compared to fixed round-robin switching, will **cache hit rate** increase or decrease? Why?

It will increase since there will be less threads competing the cache capacity.