Last updated:
10/17/2022

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.

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.

Problem M10.3: Multithreading

Problem M10.3.A

This is dictated by the CRC latency. Here is what issues look like:
Cycle 0: CRC, ADDI // source r3 will be ready in cycle 5
Cycle 1: BNE, LW // source r2 will be ready in cycle 4
Cycle 2: -
Cycle 3: -
Cycle 4: -
Cycle 5: CRC, ADDI
Cycle 6: BNE, LW

IPC = 4 instructions / 5 cycles = 0.8 instructions/cycle

Problem M10.3.B

No. The critical path is dictated by CRC latency.

Problem M10.3.C

5 threads. (Instructions from thread 1 are highlighted)
Cycle 0: BNE0, LW0, **CRC1, ADDI1**
Cycle 1: **BNE1, LW1**, CRC2, ADDI2
Cycle 2: BNE2, LW2, CRC3, ADDI3
Cycle 3: BNE3, LW3, CRC4, ADDI4
Cycle 4: BNE4, LW4, CRC0, ADDI0
Cycle 5: BNE0, LW0, **CRC1, ADDI1**
Cycle 6: **BNE1, LW1,** CRC2, ADDI2
Cycle 7: BNE2, LW2, CRC3, ADDI3
Cycle 8: BNE3, LW3, CRC4, ADDI4
Cycle 9: BNE4, LW4, CRC0, ADDI0

Another way to think about the question:
Maximum IPC = 4 instructions/cycle
Current IPC = 0.8 instructions/cycle
So if IPC = 4 is reachable, then at least 4 / 0.8 = 5 threads are needed.

Problem M10.3.D

IPC = 4 instructions / cycle

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`

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.