CIS 351 Practice Final Solutions

April 26, 2022

Monitor Piazza for details about the exam. None of these problems are due for credit.

The answers to the open-ended questions are deliberately terse and give just the basic information to check whether you are on the correct track. Your understanding should go beyond just what is written here.

1. Review the cache homework
2. Review the not-for-credit cache activities on PL
3. Describe the reasoning behind including cache memory in a system versus just a CPU, RAM, and hard drive.

A CPU is orders of magnitude faster than RAM. Although a small, fast memory (a cache) is not necessary for a program to run correctly, it prevents the program from being bogged down by the speed of RAM.

4. Caches are designed to realize temporal and spatial locality. Give context for each of the localities and describe how a cache is configured to make use of them.

Temporal locality – things used recently are likely to be used again. Spatial locality – things that will be used together are likely to be stored near one another.

A cache takes advantage of temporal locality by keeping recently-used values. A cache takes advantage of spatial locality by grabbing several nearby items from memory at once.

5. Explain the relationship between memory and a cache. How do we map memory to the cache? How do we check if an item is in the cache (What items are checked?). Be detailed.

A cache can hold only a subset of memory. We use mapping, as discussed in lecture, to determine where in the cache a block of memory should be stored.

Exactly where something is stored, and therefore what needs to be checked when looking for something, depends on the parameters of the cache.

6. Be prepared to calculate the miss rate for a given piece of assembly code and a specified cache.
7. What effect(s) can changing the block size have on a cache? (Hint: Pick one type of cache and explain for it.)

8. For pipelining, review the Week 14 activities from number 8 onwards, the Week 15 activities, and the Pipelining lab.

9. Explain the benefits of a pipelined CPU over a single cycle CPU. Explain possible drawbacks of pipelining.

Pipelining increases the number of instructions a processor can complete in a fixed amount of time. Pipelining increases the complexity of the circuit and may result in stalls or flushed instructions.

10. Consider playing a video game with two friends. The three of you decide you want to start a magic-sword business. One of you is a strong miner who can get ore, another is good at turning ore into refined metal, and the third can turn metal into magic swords. If the miner takes 5 minutes to gather enough ore, the refiner takes 10 minutes to purify the ore into metal, and the blacksmith takes 15 minutes to make a sword from metal, how long will it take you to make 1 sword? 5 swords? What is the long-term rate at which you can make swords?

- 1 sword: 30 minutes
- 5 swords: 90 minutes
- Long-term: 1 sword per 15 minutes

11. The code below does not have any data hazards. Make a small modification to the code so that it has a data hazard that can be solved without a stall. Add another hazard that can be solved only by stalling the pipeline.

```
add $t0 $t1 $t2
add $t3 $t4 $t5
add $t6 $t7 $t8
```

For example:

```
add $t0 $t1 $t2
add $t3 $t0 $t5
add $t6 $t7 $t8
```

For example:

```
lw $t0, 0($t2)
add $t3 $t0 $t5
add $t6 $t7 $t8
```

12. Consider the code below running on our five-stage MIPS pipeline. Assume the pipeline solves hazards by forwarding whenever possible and stalls only when absolutely necessary. Which instruction is in each stage of the pipeline during Cycle 7 (0-indexed)? What are the values on each of the wires of our microarchitecture diagram?

```
add $t0 $t1 $t2
addi $t3 $t4 $t5
sub $s0 $s1 $s2
lw $s4, 0($t3)
slt $a0 $0 $t2
and $s3 $s0 $t1
ori $v0 $t0 12
xori $v1 $t1 18
and $s1 $s2 $s3
```

- Fetch: xori
- Decode: ori
- Execute: and
- Memory: and
- Writeback: slt

The values on the wires in a particular stage are the same as they would be if the instruction in that stage were operating in a non-pipelined environment.

13. Explain why hazards caused by a load-word instruction cannot be solved without stalling.

The data is not available until the end of the Memory stage.

14. Given assembly code, identify hazards in the code running on a pipelined CPU. Explain why they are a problem and how they are mitigated.

15. For each instruction below, explain where the operands to the ALU come from when that instruction in in the Execute stage (i.e., whether they are forwarded and, if so, from which stage).

```
add $t0 $t1 $t2
add $t0 $t4 $t5
add $t6 $t1 $t8
add $t1 $t0 $t6
add $t3 $t0 $t6
```

(a) Registers read during previous cycle

(b) Registers read during previous cycle

(c) Registers read during previous cycle

(d) First register forwarded from Writeback; second forwarded from Memory

(e) First register read during previous cycle; second forwarded from Writeback

16. Explain what branch prediction is and why it is "safe" – that is, why it does not result in changing the outcome of a program if we guess wrong.

We make branch decisions early enough that we can flush incorrect instructions before they change program state.

17. Explain any benefits and drawbacks of branch prediction compared to stalling.

Branch prediction allows us to make use of cycles that would otherwise be wasted each time the prediction is correct.

18. Explain why branch prediction often has better than 50 percent accuracy.

Many of the branch statements in code are taken more than once, and they often result in the branch not being taken, so the odds are better than 50-50 if we simply guess that the branch will not be taken.

19. Explain the relationship between the stage in which a branch decision is made and the branch-misdirection penalty.

The later the decision is made, the larger the penalty will be because more incorrect instructions will have been fetched before the decision.

20. Consider the following code running on a machine that computes its branch decision in the Decode stage (such as the final version of our pipelined microarchitecture from the slides). If the branch instruction enters the pipeline during Cycle 0, what instruction, if any, is in the Decode stage during Cycle 2?

```
beq $0 $0 done
add $t6 $t1 $t8
add $t1 $t0 $t6
add $t3 $t0 $t6
done:
sub $t3 $t0 $t6
sub $t1 $t0 $t6
```

The add will be fetched during Cycle 1, but it will be flushed because the branch is taken. So, during Cycle 2, there will be nothing in the Decode stage.