

# EECS 370 Winter 2009

## Homework 5 SOLUTIONS

**Assigned:** Tuesday, March 17, 2009  
**Due:** Tuesday, March 31, 2009 (in class)

Name: \_\_\_\_\_ Uniqname: \_\_\_\_\_

### Instructions:

1. Please write your name and uniqname in the spaces provided above. *Attach this cover sheet* to your completed solutions to the problems listed and turn them in at class on the due date. Submissions without a completed, attached cover sheet cannot be graded.
2. Your answers should be neat, clear, and concise. Computer-written work is recommended (but not required). Show all your work, and state any special or non-obvious assumptions you make.
3. You may discuss your solution methods or your answers with other students, but the solutions you submit must be your own.

### Scores:

| Problem #    | Points     |
|--------------|------------|
| 1            | /4         |
| 2            | /8         |
| 3            | /10        |
| 4            | /10        |
| 5            | /8         |
| <b>Total</b> | <b>/40</b> |

**Problem 1** (4 points)

Suppose we are given a “magical” register file that you can read from and write to in negligible time. For everything else, assume your performance conforms to Figure 6.2 on page 373 of your textbook.

a) Compared to using the old register file (100 ps reads and writes) in Figure 6.2, will the magical register file affect the performance of a pipelined architecture? If yes, by how much? If not, why?

Improving the performance of the register file does not affect the clock cycle time, as there are multiple other stages that all take more time (200 ps) than the register read/register write stages did (100 ps). This means that the register read and register write stages were not the bottleneck for the pipeline. Thus, there is no speedup effect from this change.

b) What if, instead of a magical register file, you use a new register file that now takes 25% more time?

Register reads and writes will now take 125 ps, which does not exceed the runtimes of the other stages (200 ps). Therefore, this has no effect on the clock cycle time either.

**Problem 2** (8 points)

You have analyzed a program and discovered that 60% of branches are taken and 40% are not taken. You have also discovered that 35% of all branches branch forward, while the remaining 65% branch backward (this statistic holds for both taken and not taken branches). Evaluate the accuracy (i.e. % correct predictions on average) of the following branch prediction schemes when running your program. Show your work.

a) Random Prediction (i.e. 50/50)

$$0.5 * 0.6 + 0.5 * 0.4 = 50\% \text{ correct}$$

b) All branches predicted taken

$$1.0 * 0.6 + 0.0 * 0.4 = 60\% \text{ correct}$$

c) All branches prediction not taken

$$0.0 * 0.6 + 1.0 * 0.4 = 40\% \text{ correct}$$

d) Branches predicted taken if offset negative, not taken otherwise.

$$1 * 0.6 * 0.65 + 1 * 0.4 * 0.35 = 53\% \text{ correct}$$

**Problem 3** (10 points)

Consider the standard 5 stage LC-2K9 pipeline as presented in lectures. Assume all data dependencies can be handled by forwarding. Assume a speculated “never taken” policy and that branches are resolved after the EX cycle at stage 4.

a) What is the average CPI (clocks per instruction) for a program in which 20% of all instructions are branches (this is roughly true for most programs) and branches are predicted correctly 70% of the time?

Branch Penalty: 3 cycles

$$\text{CPI} = \text{Ideal CPI} + \text{branch\_misprediction\_penalty} = 1 + 0.2 * 0.3 * 3 = \mathbf{1.18}$$

b) You decide to implement a more accurate branch predictor in the standard LC2K7 pipeline. Simulations suggest that this branch predictor will be accurate 80% of the time. What would be the average CPI of your new pipeline?

$$\text{CPI} = \text{Ideal CPI} + \text{branch\_misprediction\_penalty} = 1 + 0.2 * 0.2 * 3 = \mathbf{1.12}$$

c) Now suppose that you were motivated by the above results and decided to evaluate an even more accurate branch predictor. The design under evaluation has an average branch prediction accuracy of 95.5%. However, this branch predictor is so large and complex that we must increase our fetch stage (IF) leading to a clock cycle increase of 15%. Is implementing this branch predictor in our design a good idea (compared to the design of part (b))? Why?

$$\text{CPI}_B = 1.12 \text{ (from part (b))}$$

$$\text{CPI}_C = \text{Ideal CPI} + \text{branch\_misprediction\_penalty} = 1 + 0.2 * 0.045 * 3 = \mathbf{1.027}$$

$$\text{Execution\_Time}_B = \text{Instructions} * \text{CPI}_B * \text{clock\_period}$$

$$\text{Execution\_Time}_C = \text{Instructions} * \text{CPI}_C * \text{clock\_period} * 1.15$$

$$\text{Execution\_Time}_C / \text{Execution\_Time}_B = (1.027 * 1.15) / 1.12 = 1.055$$

The pipeline of part (c) is 5.5% slower than the pipeline of part (b), so it's **not a good idea** to implement the new more accurate branch predictor.

**Problem 4** (10 points)

Consider a normal 5-stage LC-2K8 pipeline as discussed in class. Branches are resolved in the MEM stage and branches are predicted as **not taken**. 30% of load instructions incur a one cycle stall due to data hazards that cannot be resolved via forwarding. Assume the instruction memory has a 2% miss rate and the data memory has a 10% miss rate for both reads and writes. For an instruction or data memory read, a miss causes a 15 cycle stall. For a data memory write, a miss causes a 10 cycle stall. Analysis has shown the following dynamic instruction breakdown for program X: 10% not taken branches, 7% taken branches, 20% loads, 23% stores, 40% R-type instructions (add/nand).

(a) What is the CPI of this machine when running program X? (4 points)

$$\begin{aligned} \text{CPI} &= 1 + \text{branch penalty} + \text{load stall penalty} + \text{memory read penalty} + \text{instruction read} \\ &\text{penalty} + \text{memory write penalty} \\ \text{CPI} &= 1 + 0.07 * 3 + 0.2 * 0.3 * 1 + 0.2 * 0.1 * 15 + 1 * .02 * 15 + 0.23 * 0.1 * 10 \\ &= 2.1 \end{aligned}$$

(b) Now assume the pipeline was stretched out to 10 stages (numbered 1 to 10, with instruction fetch beginning in stage 1, and instructions finally completing in stage 10). Branches are resolved in stage 7. The clock frequency is increased by 50% than that of the original processor with the 5 stage pipeline. For an instruction or data memory read, a miss now causes a 35 cycle stall. 30% of load instructions still incur a one cycle stall due to data hazards that cannot be resolved via forwarding. What is the CPI for this new pipeline design? (3 points)

$$\begin{aligned} \text{CPI} &= 1 + 0.07 * 6 + 0.2 * 0.3 * 1 + 0.2 * 0.1 * 35 + 1 * .02 * 35 + 0.23 * 0.1 * 10 \\ &= 3.11 \end{aligned}$$

(c) Which of these machines performs better, the original 5 stage pipeline or the new 10 stage pipeline? How many times faster is it than the other one? (3 points)

Considering both the CPI change and the increased clock frequency, the new 10 stage pipeline is faster (that is, B is faster than A) by a factor of:  
 $A * 1.5 / B = 1.0128 \sim 1.3\%$  faster

**Problem 5** (8 points)

Consider a typical 5-stage pipelined processor (Fetch, Decode, Execute, Memory, and Write back stages) where:

- Branches are predicted not-taken and resolved in Memory stage.
- Forwarding paths are available. When forwarding cannot resolve the problem, the instruction stalls in Execute.
- All instructions take 1 cycle to execute other than the “multiply” instruction which takes 2 cycles (and thus causes a stall in the pipeline).
- Address calculations are done in the Execute stage.

For a given program, 20% of instructions are loads, 25% are adds, 10% are multiply, 20% are branches, 25% are stores; and 60% of branches are not taken.

Also, for a load instruction, *each operand* of the following instruction has a 40% chance of being dependent on the result of the load instruction. You may assume the probabilities are independent of each other.

What would be the CPI for this program? Show your work for full credit.

*Hint:* You may have to use the probability formula

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$
$$= P(A) + P(B) \quad \text{if A and B are mutually exclusive}$$

Let A = the event that regA depends on the result of the load instruction

Let B = the event that regB depends on the result of the load instruction

$$P(A) = 0.4$$

$$P(B) = 0.4$$

$$P(A \cup B) = P(A) + P(B) - P(A) * P(B)$$

$$P(A \cup B) = 0.4 + 0.4 - 0.4 * 0.4$$

$$P(A \cup B) = 0.64$$

(3 points)

$$\text{CPI} = 1 + \text{branch penalty} + \text{multiply penalty} + \text{load stall}$$

For the load stall, add, multiply and branch all have a 51% chance that they depend on the result of a load instruction immediately before them. For a load and store, since only 1 register (regA) can cause a data hazard, there is only a 30% chance that a load/store depends on the result of a load instruction immediately before it. The penalty for needing to stall for such a situation is still only 1 cycle.

$$\text{CPI} = 1 + 0.2 * 0.4 * 3 + 0.1 * 1 + [(0.25 + 0.1 + 0.2) * 0.64 + (0.2 + 0.25) * 0.4] * 1$$

$$\text{CPI} = 1.872$$

(5 points)