Introduction to Computer Organization – Fall 2024

Lab 9

Due: @11:55 PM, Wed Nov 6

The following assignment is intended to be completed during your assigned lab period. One member of your group must submit the assignment to Gradescope by the posted deadline and indicate your group members when submitting the assignment. **Each group member must be present during the scheduled lab period in order to receive credit.**

Group names and uniqnames

Pod Member Name	**Uniqname**
Chase Hadley	chasehad
Nicolas Fernandez	nicofern
Arnav Nikam	aunikam
Noah Del Valle	noahdelv

For each of the following problems, one person should act as the "scribe" and log the discussions of the group. You should rotate who is the scribe for each problem and indicate in the given space.

For all problems, assume we are starting with the 5-stage pipeline discussed in lecture, unless otherwise stated

Problem 1: Benchmarking *[32 points]*

Consider a 10000 instruction LC-2K benchmark with:

25% **lw**, 30% of which are followed by a dependent instruction
15% **sw**
30% **add**/**nor** 0% of which are immediately followed by a dependent instruction, and 50% of which are followed by a dependent instruction with a non-dependent instruction in between
20% **beq**, 60% of which are taken
10% **noop/halt**

Each use of the ALU costs **10ns**, reading memory costs **50ns**, writing memory costs **60ns**, register file reads and writes cost **5ns**, and everything else costs **0ns**. The pipelined design has an additional **10ns** delay due to pipeline registers and control logic.

a. What is the optimal achievable serial runtime of this benchmark? *[Sol'n given in lab slides]*

   877500ns

b. What is the benchmark runtime on the single cycle design?

   1250000ns

c. How many cycles are needed to complete this benchmark on the multi-cycle design?

   40500 cycles

d. What is the total execution time of this benchmark on the multi-cycle?

   2430000 ns

e. What is the average CPI (Cycles per Instruction) on the multi-cycle?

   4.05 cycles/instruction

f. Assuming we fix the code by inserting noops, what is the runtime and CPI on the pipelined design?

   cycles = 19004
   runtime = 19004 * 70ns = 1330280ns
   CPI = 19004 / 10000 = 1.9004 cycles/instruction

g. Assuming we use Detect-and-Stall to deal with hazards, what is the runtime and CPI on the pipeline?

   runtime = 19004 * 70ns = 1330280ns
   CPI = 19004 / 10000 = 1.9004 cycles/instruction

h. Assuming we use Detect-and-Forward for data hazards and Detect-and-stall for control hazards, what is the runtime CPI on the pipeline?

   cycles = 16754
   runtime = 1172780ns
   CPI = 16754 / 10000 = 1.6754 cycles/instruction

i. Assuming we use Detect-and-Forward and Speculate-and-squash predicting not taken instead, what is the runtime and CPI on the pipeline?

   cycles = 14354
   runtime = 14354 * 70ns = 1004780ns
   CPI = 14354 / 10000 = 1.4354 cycles/instruction

j. Assuming we use Detect-and-Forward and Speculate-and-squash predicting taken, what is the runtime and CPI on the pipeline?

   cycles = 13154
   runtime = 13154 * 70ns = 920780ns
   CPI = 13154 / 10000 = 1.3154 cycles/instruction

k. Assuming we use Detect-and-Forward and predict every branch correctly, what is the runtime and CPI on the pipeline?

   cycles = 10754
   runtime = 10754 * 70ns = 752780ns
   CPI = 10754 / 10000 = 1.0754 cycles/instruction

l. Compare all of the above results. In a few sentences, identify some comparisons with results that might be surprising. Why would these results be surprising, and what does that say about advancements in computer architecture as Moore's law is ending? *[10 points]*

   The results of the multicycle vs single-cycle shows that it takes double the time to do multi, which is extremely surprising. This could be due to the uneven latencies affecting the different cycles of the path. Even with hazard resolution, the pipeline is slower which led to increasing parallelism through forwarding logic which means that improving the memory's system would help time. Also, if we can get good branch prediction we can significantly speed up runtime.

Problem 2: Branch Prediction *[18 points]*

Consider the sample assembly code from the project 1 spec. What is the mispredict ratio for each of the following schemes?

- Predicted Not Taken 5:9 (total ratio)
- Predicted Taken 4:9 (total ratio)
- 2-bit predictor initialized to strongly not taken, with a separate predictor for each branch
  1:5 (first beq), 2:4 (second beq) -> 3:9 (total ratio)
- Forward branches predicted Not Taken, Backward branches predicted Taken
  1:5 (forward branch), 0:4 (backward branch) -> 1:9 (total ratio)
- Predict Taken for unconditional branches, 1-bit per branch for others, initialized to Taken
  2:5 (first beq) 0:4 (second beq) -> 2:9 (total ratio)

```
       lw     0       1       five      load reg1 with 5 (symbolic address)
       lw     1       2       3         load reg2 with -1 (numeric address)
start  add    1       2       1         decrement reg1
       beq    0       1       2         goto end of program when reg1==0
       beq    0       0       start     go back to the beginning of the loop
       noop
done   halt                             end of program
five   .fill  5
neg1   .fill  -1
stAddr .fill  start            will contain the address of start (2)
```