EE 4683/5683: Computer Architecture

Homework #2

Instructions:

Graduate Students: Answer all questions.

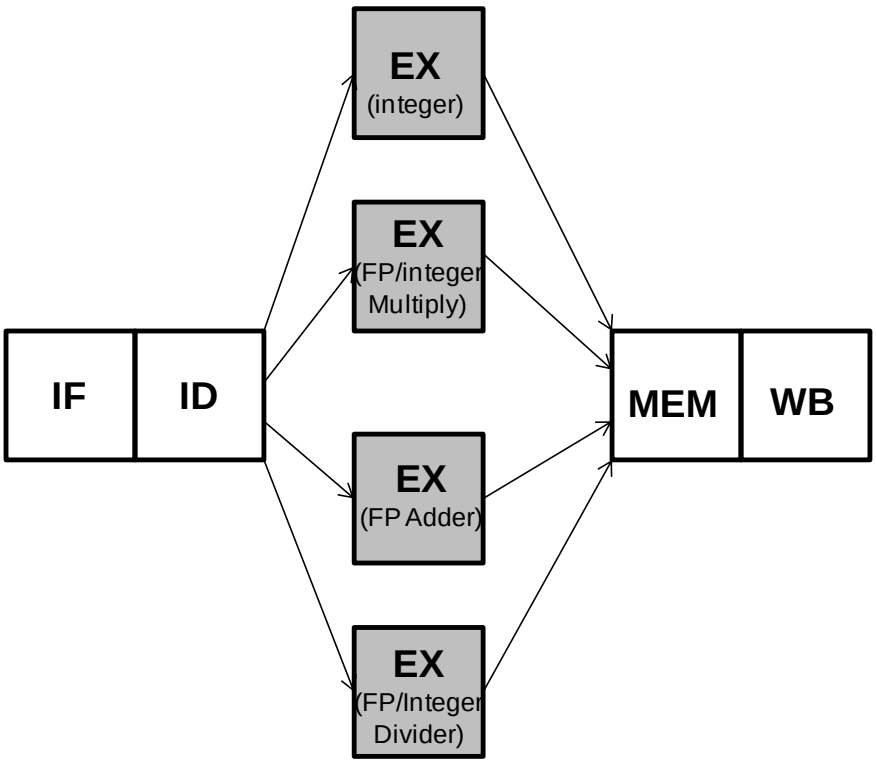
Undergraduate Students: The last question is bonus question.

Problem 1

Solve from text Problem C.1 [(a), (b), (c), (d), (e), (f) and (g)]

Problem 2

Consider a MIPS 5-stage pipeline with the execution stage consisting of 1-Integer, 1-FP multiply, 1-FP adder and 1-FP divide shown below. Integer operations takes 1 clock cycle, FP/integer multiply takes 7 clock cycles, FP adder takes 4 clock cycles and FP divide takes 25 clock cycles. FP load is similar to an integer load operation.




```
Loop: L.D       F0, 0(R2)
      L.D       F4, 0(R2)
      MULT.D    F0, F0, F4
      ADD.D     F2, F0, F2
      ADDI      R2, R2, #8
      ADDI      R3, R3, #8
      SUB       R5, R4, R2
      BNZ       R5, Loop
```

Assume that the initial value of R4 is R2+792.

(a) Show the timing of this instruction sequence for the 5-stage MIPS pipeline with forwarding. Assume that the branches are handled by flushing the pipeline. If all memory references hit in the cache, how many cycles does this loop take? (Hint: Make use of Excel spreadsheet where on one column show all the instructions and on multiple rows show the timing)

(b) Assuming the pipeline with a single cycle delayed branch slot, normal forwarding, schedule the instructions in the loop including the branch delay slot. You may reorder the instructions, and modify the individual instruction operands, but do not undertake loop transformations that change the number of opcode of the instructions in the loop. Compute the number of cycles needed to execute the entire loop.

(c) Can you now transform this loop by unrolling so that number of stalls can be reduced? Is it possible to eliminate all stalls? You may now re-order instructions, use forwarding, and the branch delay slot. Compute the total number of cycles needed to execute the loop.

Problem 3

Consider the following three pipeline processor implementations. Assume that all 3 pipelines can achieve a CPI of 1, except for mispredicted branches.

- **Implementation X**: A pipeline length of 20 stages with branches resolving in stage 7, running at 666MHz.
- **Implementation Y**: A pipeline length of 30 stages with branches resolving in stage 12, running at 1GHz.
- **Implementation Z**: A pipeline length of 25 stages with branches resolving in stage 5, running at 833MHz.

For all processors, the branch predictor is located in stage 1.

a) What are the cycle times for each of these pipelines, in nanoseconds?

b) Now consider that you are the CTO at Intel and you want to choose a pipeline implementation for your next generation processor. You know that the branch predictor in your architecture can achieve 95% prediction accuracy. If you were to select a pipeline design purely based upon on minimum branch misprediction penalty in nano-seconds, which architecture would you choose?

c) Now, consider pipeline Implementation X. Assume this is the only pipeline available to you, and as CTO you have two choices. The next generation of silicon technology has given you extra transistors to work with, and you can either use them to build a better branch predictor that will achieve 98% accuracy instead of 95%, OR to build extra hardware to allow branches to resolve in stage 5 instead of stage 7. Which approach will provide the best average CPI for a workload with 20% branches?

Problem 4

List all the dependencies (output, anti, and true) in the following code fragment. Indicate whether the true dependences are loop-carried or not. Show why the loop is not parallel.

```
for( i = 2; i < 100; i++ ) {
      a[i] = b[i] + a[i];              /* S1 */
      c[i-1] = a[i] + d[i];            /* S2 */
      a[i-1] = 2 * b[i];               /* S3 */
      b[i+1] = 2 * b[i];               /* S4 */
}
```

Problem 5 *For Graduate students, Undergrads get BONUS if solution is correct!*

Here is an unusual loop. First, list the dependencies and then re-write the loop so that it is parallel.

```
for( i = 1; i < 100; i++ ) {
      a[i] = b[i] + c[i];              /* S1 */
      b[i] = a[i] + d[i];              /* S2 */
      a[i+1] = a[i] + e[i];            /* S3 */
}
```