18-447
Computer Architecture
Recitation 2

Rachata Ausavarungnirun
Carnegie Mellon University
Spring 2015, 2/9/2015

Agenda for Today

- Quick recap on the previous lectures
- Practice questions
- Q&A on HW2, lab3, and lecture materials

- Important deadlines:
  - **HW2** due Wednesday (2/11)

Quick Review

- Microprogrammed
- Microcoded designs
- Pipelining
  - Handling stalls
  - Data dependences
  - Data Forwarding
  - Control Dependences
- Fine-grained multithreading
- Predicated Execution
- Branch Prediction

Practice Questions: Value Prediction

Assume the following piece of code, which has four load instructions in each loop iteration, loads to arrays x, y, z, t:

```
// initialize integer variables c, d, e, f to zeros
// initialize integer arrays x, y, z, t

for (i=0; i<1000; i++) {
    c += x[i];
    d += y[i];
    e += z[i];
    f += t[i];
}
```

Assume the following state of arrays before the loop starts executing:

- x consists of all 0's
- y consists of alternating 3's and 6's in consecutive elements
- z consists of random values between 0 and $2^{(32)} - 1$
- t consists of 0, 1, 2, 3, 4, ..., 999

Practice Questions: Pipelining

When handling dependent instructions, an alternative to data forwarding is *value prediction*, as we discussed in class. You are going to use value prediction to resolve flow dependences in a processor with a 10-stage pipeline (Fetch, two stages of Decode, five stages of Execute, one stage of Memory, one stage of Writeback). Assume that the processor never stalls for memory, that it assumes branches are not taken until they are resolved, and that it resolves branches in the last stage of Execute. The value predictor is placed in the Decode stage, and produces a value prediction for a register whose value is not available. The processor is designed so that it *never stalls due to data dependences*, but always value-predicts to resolve stalls. This works as follows:

- When an instruction passes through the second stage of Decode and *reads* a register, data dependence detection logic detects whether an older instruction later in the pipeline is *writing* to that register.
- When a data dependence is detected, the processor uses its *value predictor* to predict the value that the register will eventually have, and (i) feeds that value to the dependent instruction, (ii) records the value for later checking. Therefore, the dependent instruction can proceed without a stall using the predicted value.
- When an instruction that *writes* a register reaches the Writeback stage, it writes its value back, and also checks the value against predictions that were made for that value. If a prediction was incorrect, then (i) the entire pipeline is flushed, and (ii) fetch restarts in the subsequent cycle from the first instruction that received a mispredicted value.

Assume that the processor is designed so that it can handle multiple predictions "in flight" for a single register at different instructions in the pipeline, and that a new prediction is made each time a register is read.

Practice Questions: Branch Prediction

```
int array[1000] = { /* random values */ };
int sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0;

for (i = 0; i < 1000; i ++) // LOOP BRANCH
{
    if (i % 4 == 0) // IF CONDITION 1
        sum1 += array[i]; // TAKEN PATH
    else
        sum2 += array[i]; // NOT-TAKEN PATH

    if (i % 2 == 0) // IF CONDITION 2
        sum3 += array[i]; // TAKEN PATH
    else
        sum4 += array[i]; // NOT-TAKEN PATH
}
```

Q & A