Lecture 03 (Out of Order Pipelines)

1 Performance Considerations

1. Performance cannot be gauged with clock speed or RAM
2. Fasteness is comparable only when given a program

1.1 Performance Equation

$$\frac{programs}{time} = \frac{programs}{instructions} \cdot \frac{instructions}{cycles} \cdot \frac{cycles}{time} = \frac{IPC \times frequency}{instruction}$$

(assumes just 1 program)

1.1.1 Instructions

Depends on compiler

1.1.2 Frequency

1. Depends on transistor technology
2. If there are more (or smaller) pipeline stages, then frequency is higher
3. $\frac{1}{frequency} = time_{cycle} = \frac{T}{k} + L$ (k pipelines, L = latch delay)
4. Adding too many stages will increase forwarding logic, thus being counter productive after a point
5. $P \propto f^3 \implies \Delta T \propto P$ - thus pipeline stages which had gone to 27 came down to somewhere between 12-15

1.1.3 IPC

1. Depends on architecture and compiler
2. Will be primary focus of the book (course as well?)
3. Ideal IPC of in-order pipeline is 1, typically < 1

1.1.3.1 CPI

$$CPI = CPI_{ideal} + \text{stall\_rate} \times \text{stall\_penalty}$$

1. Stall rate will remain constant with pipeline stages (empirical)

2. Stall penalty increases with more pipeline stages

1.1.3.2 Increasing IPC

1. Issue more than one instructions per cycle
2. Make it a superscalar processor (it is hard to do this)
    i. very complex stall and forwarding paths
    ii. of the order of $O(n^2)$ for n-issue processor
    iii. if there are branches, then our party is over

2 Out of Order Pipelining

1. Much better solution
2. Consumer is executed after producer

2.1 Basic Principle

1. Create a pool if instructions
2. Find instructions that are mutually independent and have all their operands ready
3. Execute them OOO

ILP: Instruction Level Parallelism - number of ready and independent instructions we can simultaneously execute

2.2 Understanding OOO

1. Construct a dependency graph
2. Make directed edges from consumer to producer
3. Execute all nodes with no outgoing edges
4. Larger graph is always better so that all processor units are active

2.3 Issues

1. Need to figure out instruction window size
    - typical window size is 64 to 128
2. How do we handle branching and ensure that each instruction is on the correct branch
    - need a very accurate branch predictor
    - typically 1 in 5 instructions is a branch
    - need to predict the following:
        i. if it is a branch
        ii. if it is taken or not
        iii. where is the target of the branch

2.4 Math of Branch Prediction

Number of instructions: n Number of branches: $n/5$ Probability of incorrect prediction: p Probability of at least one mistake: $P_n = 1 - (1-p)^{n/5}$

For $n = 100$, if P_n has to be at most 10, p has to be < 0.5

Highest accuracy in A1 is observed to be 97, very poor :(