Lesson 3

Pipelining

Pipelining is crucial to improving performance in processors; it increases throughput and reduces cycle time. The downside of pipelines are the increase in hazards, both control and data.

Pipelining in a Processor

Five stages in a basic pipeline:
Fetch, Read/Decode, ALU, Memory Access, Write the Registers
F, D, E, M, W

Pipelining the instructions takes the same amount of time, but throughput is improved.

Pipelining CPI

If there an instruction has to wait at a pipeline stage, all the instructions ahead of it can proceed through the pipeline, but all the instructions behind it are also stalled. This is called a delay in the pipeline. The pipeline ahead of the delay will not have instructions to execute as the pipeline empties and the instructions behind the delay will be stalled.

B/c of stalls, steady-state pipeline CPI > 1

As the number of delays increase through the pipeline, the CPI will increase.

→ Stall = holding an instruction @ a stage in the pipeline (causing a backup) – "FORWARD" bubbles
→ Flush = clearing out previous stages in the pipeline – "BACKWARD" bubbles
a flush is a corrective action

Pipeline Stalls and Flushes

Pipeline Flush: Branches can cause bubbles when the incorrect branch is taken. When this happens all the incorrect instructions that were fetched must be flushed from the pipeline and replaced with NOPs. Then the correct instructions must be fetched.

Control Dependencies

Once we have a branch, all subsequent instructions have a CTRL DEP on that branch.

Control dependencies: When an instruction is dependent on the outcome of a branch decision, these instructions are said to have a control dependence.

20% of instructions are branches and jumps
50% of branch and jump instructions are taken

Overall CPI = CPI of program + % of instructions mispredicted * penalty for misprediction

↳ Penalty = how many cycles do we pay as a result of a flush?
Deeper pipeline = bigger penalty

Data Dependencies

Data dependence: When an instruction needs data from another instruction that is called a data dependence.

Type of data dependencies:

1. RAW - read after write. Also called Flow, True Dependence.

The following dependencies are False or Name dependencies.

2. WAW - write after write. Also called Output Dependence.
3. WAR - write after read. Also called Anti-Dependence

Can be "fixed" with register renaming

RAR -read after read is not a dependence.

We want to preserve the order of writes so that the right values are in the registers @ any point in time.

If an instruction is reading from a register that the next instruction is going to overwrite, the read needs to happen first so that we don't incorrectly feed it the wrong value.

Dependencies and Hazards

"TRUE"

Dependencies are caused by the program, not the pipeline.

Some dependences will not cause problems, but some, like RAW, can cause problems in pipelines.

Hazards are true dependencies that result in incorrect execution of the program, but not all true dependencies will result in a hazard. Hazards are caused by both the program and the pipeline.

Due to a particular pipeline the program is running on

Handling of Hazards

First step to handling hazards: Detect only those dependencies that will cause hazards.

Second Step to handling hazards: Remove the hazard.

Options for removing hazards are:

1. Flush dependent instructions out of the pipeline.
   This method is used with control dependencies. Since the wrong value is in the pipeline it needs to be flushed.
2. Stall dependent instructions in the pipeline.
   This method is used for data dependencies. This will give the data time to get written to registers before it is needed by a later instruction.
3. Fix the values read by dependent instructions.
   This method is also used for data dependencies. Instead of stalling the instruction until the correct values are in the registers, the pipeline can forward the required values. The values from an ALU for example can be used as soon as they computed, rather than waiting.
   This method does not always work.

If the required register value exists somewhere in the pipeline (for example, it was calculated but not written to the register yet), at the point BEFORE we actually used that value in a computation, we can "fix" an incorrectly read register value by forwarding the value.

How Many Stages

Every pipeline should be achieving the required CPI, it is different for every pipeline.

When more stages are added to a pipeline:

A true dependence is more likely to be a hazard on a pipeline with more stages (+ less likely on a smaller pipeline).

CPI ↑
1. There are more hazards introduced into the pipeline.
2. The penalty for hazards increases.

Cycle Time ↓
3. There is less work for each stage, so the cycle time can be smaller.

The Iron Law of Performance says: Exec Time = # Instructions · CPI · Cycle Time

The number of stages must be balanced between the CPI and the cycle time.

If only performance is considered, 30 - 40 stages is ideal. But if power consumption is also considered, the ideal pipeline is 10 - 15 stages.