25-pipeline-p1-notes

Pipelining: A Deeper Look

Agenda

- 0. Re-Orienting
- 1. Quick Review
- 2. Pipelining the Y86
- 3. Data Hazards
- 4. Solution: Stalling
- 5. Solution: Forwarding
- 6. Load Interlock
- 7. Branch Prediction
- 8. Putting it All Together

Reading: 4.5

---

the slides (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/25-pipeline-p1/slides.pdf)

---

0. Re-Orienting

Another big part of how the field progressed down the Mess/Speed path.

---

1. Quick Review

Basic idea:

- in one cycle, combo into register
- chop the combo up and add more registers

How can this ever be a good idea? That's more cycles to get anything done!

Definitions: delay (latency), throughput

Throughput = 1/clock cycle if...

- all instructions uniform
- everything is perfect

Why are things not perfect?

- Non-uniform delays
- Diminishing returns. (Overhead of the registers starts slowing things down
- Hazards! (Data, control... and others too)

Implications of pipelining: the map between

- your mental model of what the machine is doing
- and what the machine is actually doing

has become even messier...

---

2. Pipelining the Y86

SEQ+ to PIPE-

---

3. Data Hazards

Fig 4.43, prog1: it works!

Fig 4.44, prog2: it breaks!

(The **Y86 Tools (https://canvas.dartmouth.edu/courses/8347/pages/y86-tools)** package has a build director for a pipeline simulator, with various Makefile options to build various versions, including a broken one. Some set of these will be released for a future HW... but if you're impatient, go to the tarball! The sample programs used in Sec 4.5 are also there, as well as **here) (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/progs/)**

---

4. Solution: Stalling

Fig 4.47

5. Solution: Forwarding

Fig 4.49

6. Load Interlock

Gosh, forwarding isn't always enough!

Fig 4.53

7. Branch Prediction

When there's a conditional jump, what should we fetch next?

Oh heck: how about we assume the jump is taken.
But then what happens?

Fig 4.62

Instruction canceling

8. Putting it All Together