26-pipeline-p2-notes

Agenda

- 0. Re-Orienting
- 1. Pipeline Demos
- 2. Some Exemplar Mess
- 3. Missing Pieces in Pipelining
- 4. Missing Pieces Elsewhere

Reading: 4.5; 8.1

---

slides (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/26-pipeline-p2/slides.pdf) (with the bug in the superscalar RAW/WAW/WAR demo fixed :)

---

0. Re-Orienting

---

1. Pipeline Demos

On Friday, we took a deeper look at pipelining by looking at how to do it in the Y86.

Namely...

- slicing the datapath into stages
- inserting a register between each stage (saving each bit that we had to "snip" for the slicing)
- then using forwarding and stalls to handle the various hazards

---

To review, let's look at two examples again (but with a version of psim that uses GIANT fonts, so it's more readable when we project)

---

psim std on prog2, Fig 4.49.

- Note the bubbles!

- Where does the addl get valB from when it's it the D stage?

---

psim std on prog5, Fig 4.53

- when addl gets to D, a bubble gets inserted to let mrmovl get to M
- but we see that m_valM gets hijacked and forwarded then---the addl doesn't wait until the value gets to %eax

In the no-bypass version, notice the TWO errors!

---

psim std on prog8, Fig 4.62

Some Exemplar Mess

Consider: what should a sequential machine do if you give it an illegal instruction?

bad1.yo (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/26-pipeline-p2/bad1.yo)

```
                      | # prog1: Pad with 3 nop's
  0x000: 30f20a000000 |   irmovl $10,%edx
  0x006: F0f003000000 |   BAD  $3,%eax
  0x00c: 10           |   nop
  0x00d: 10           |   nop
  0x00e: 10           |   nop
  0x00f: 6020         |   addl %edx,%eax
  0x011: 00           |   halt
```

What if you give a pipelined machine an illegal instruction?

...but what if it were an illegal instruction fetched during a mis-predicted branch?

bad8.yo (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/26-pipeline-p2/bad8.yo)

```
                      | # Demonstrate branch cancellation
                      | # /* $begin prog8-ys */
                      | # prog8
  0x000: 6300         |    xorl %eax,%eax
  0x002: 740e000000   |    jne  target        # Not taken
  0x007: 30f001000000 |    irmovl $1, %eax    # Fall through
  0x00d: 00           |    halt
```

```
0x00e:                | target:
0x00e: F0f202000000 |    BAD $2, %edx    # Target
0x014: 30f303000000 |    irmovl $3, %ebx    # Target+1
                      | # /* $end prog8-ys */
0x01a: 00           |    halt
```

---

3. Missing Pieces in Pipelining

(Things not in the textbook, but which you should know something about)

ISAs and compilers and pipelining

Sometimes, these are aware of pipelining!

Branch Prediction can be non-trivial

Branch history

with an FSM

Q: pipeline-friendly code?

code1.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/26-pipeline-p2/code1.c)

Some CPUs can juggle microinstructions

---

particularly superscalar

(Remember what "superscalar" means?)

Being aggressive with microinstructions

- "issue"
- "retire"
- "scoreboard"
- dependence: RAW, WAR, WAW

---

out-of-order

---

secret registers/register renaming
