15-x86-dpath-notes

X86; the Datapath

Agenda

- 0. Re-O
- 1. X86
- 2. Stages of Processing
- 3. The Datapath (High-Level View)
- 4. Processing and the Datapath
- 5. How can this possibly work?

Reading:

- Chapter 3, up to and including 3.6.2
- 4.1.5, 4.1.6
- 4.3.1. 4.3.2

In case it's helpful.... here are the slides. (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/15-x86-dpath/slides.pdf)

---

0. Re-O

The layers, from the human down to the physics

The ISA

Using the stack

---

1. X86

Pages 157-8 show **some** of the history. But there's older history they don't show!

"Code museums"

Moore's Law (http://www.mercurynews.com/business/ci_27934824/silicon-valley-marks-50-years-moores-law)
(but it's more a self-fulfilling prophesy. "Hail Mary Pass" quip.)

To reiterate: the Y86 offers a simplified subset the functionality in the X86 family. We'll now go through some avenues where the X86 offers more power.

Data types.... bytes, 2-byte "words," 4-byte "double words," and more.

- why all the variations, and odd names?

---

registers

- why so many register "A"s?

---

data movement

- various sizes (including different sizes)
- more addressing modes
- but "IA32 imposes the restriction that a move instruction cannot have both operands refer to memory locations."

---

arithmetic and logic

---

condition codes

- additional flag: CF
- compare and test instructions (subtraction without subtracting!)
- direct access to flags

---

2. Stages of Processing

Suppose you were building hardware to actually implement an ISA. What might your machine do?

A traditional approach:

- Fetch
- Decode
- Execute
- Memory
- Write-back

- PC update

(Does it *have* to be this way?)

(I will sketch this out on the board)

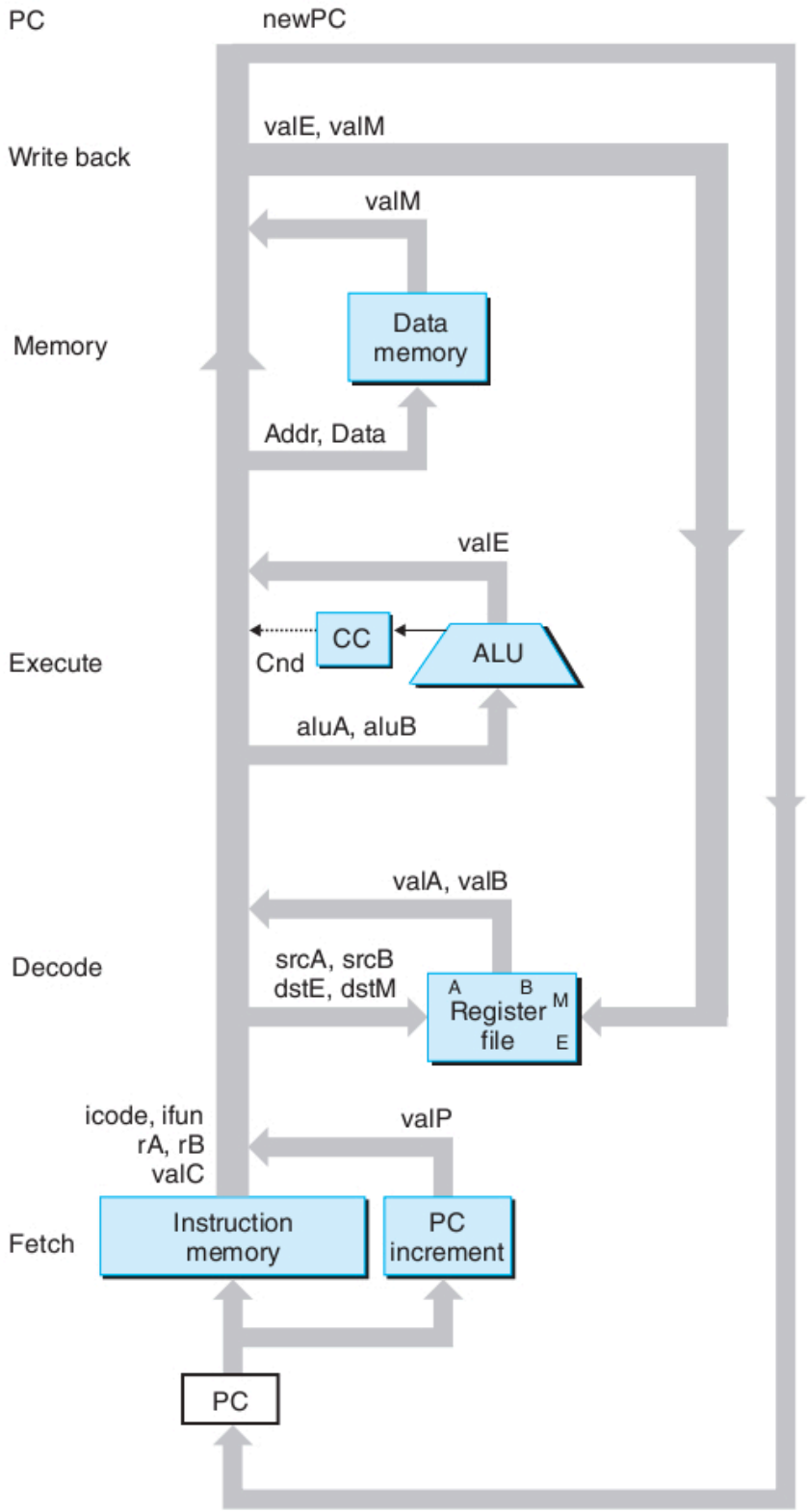
Fig 4.18

---

3. The Datapath (High-Level View)

Right now, we're talking about software instructions. But a few layers down, we have actual hardware.

Remember e.g. Fig 4.22:



4. Processing and the Datapath

Thinking about Fig 4.18 and 4.22 **together**

- The "variables" and such in the pseudocode map to storage elements
- assignment maps to wires
- operations, memory access map to operations on the larger elements

Consider for some examples from the textbook, 4.3.2:

```
opl
```

```
rrmovl
```

```
irmovl
```

```
rmmovl
```

```
mrmovl
```

```
pushl
```

5. How can this possibly work?