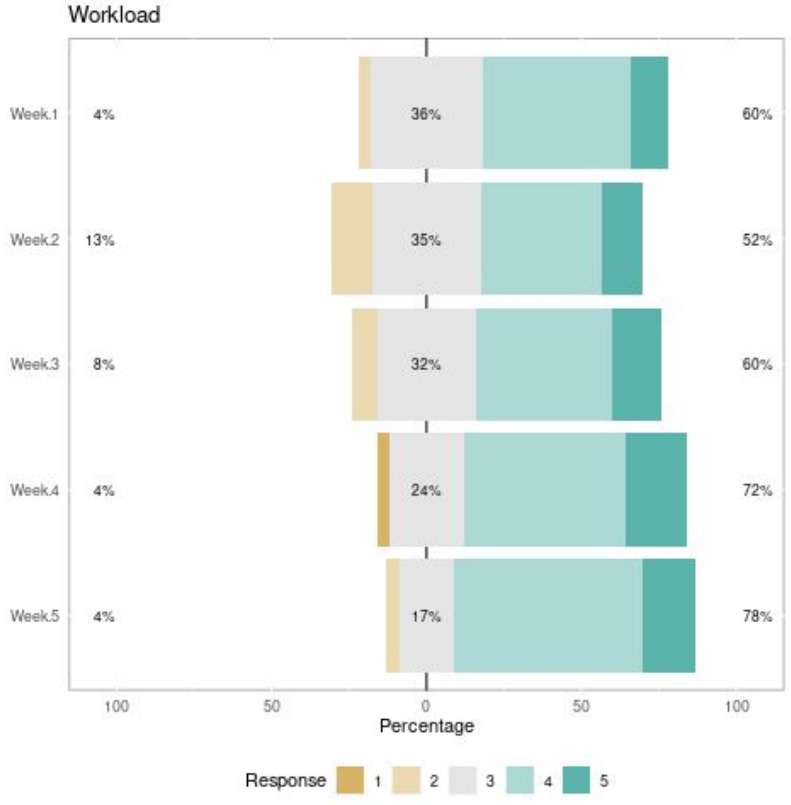
0x0A - Single Cycle CPU

ENGR 3410: Computer Architecture

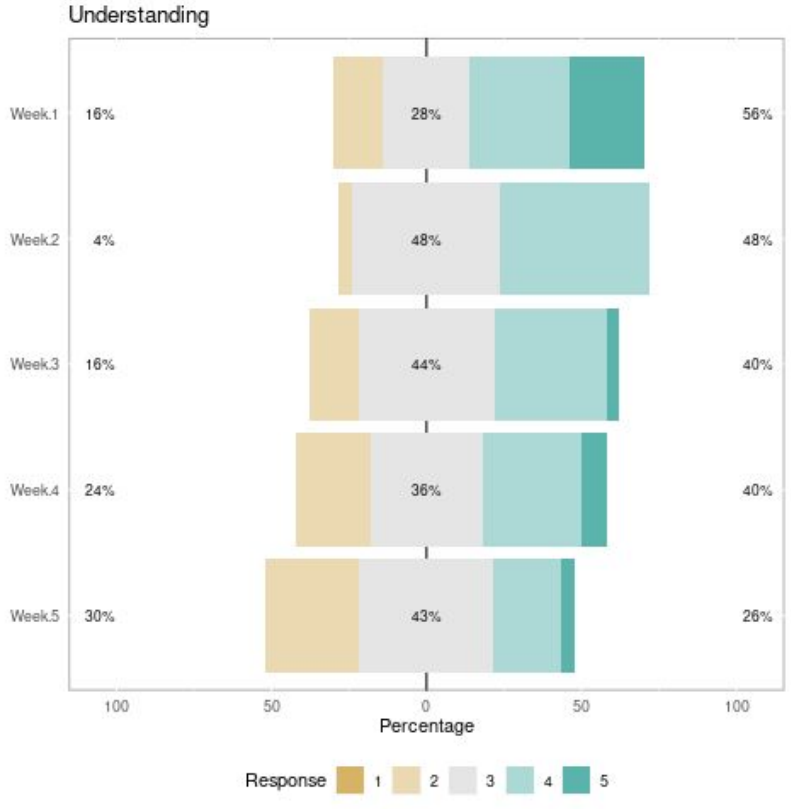
Jon Tse

Fall 2020

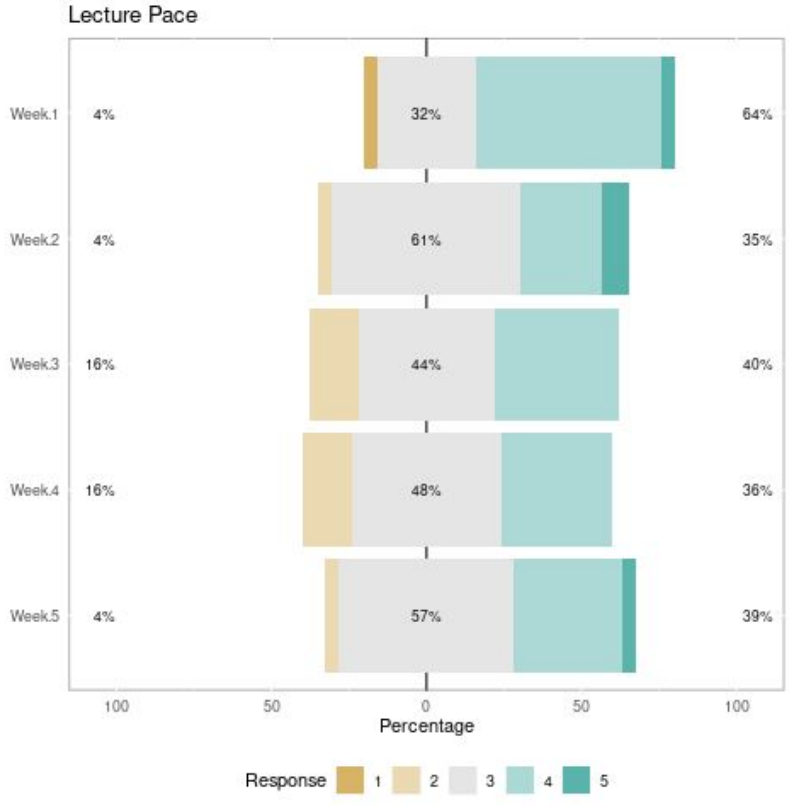
Feedback - Workload



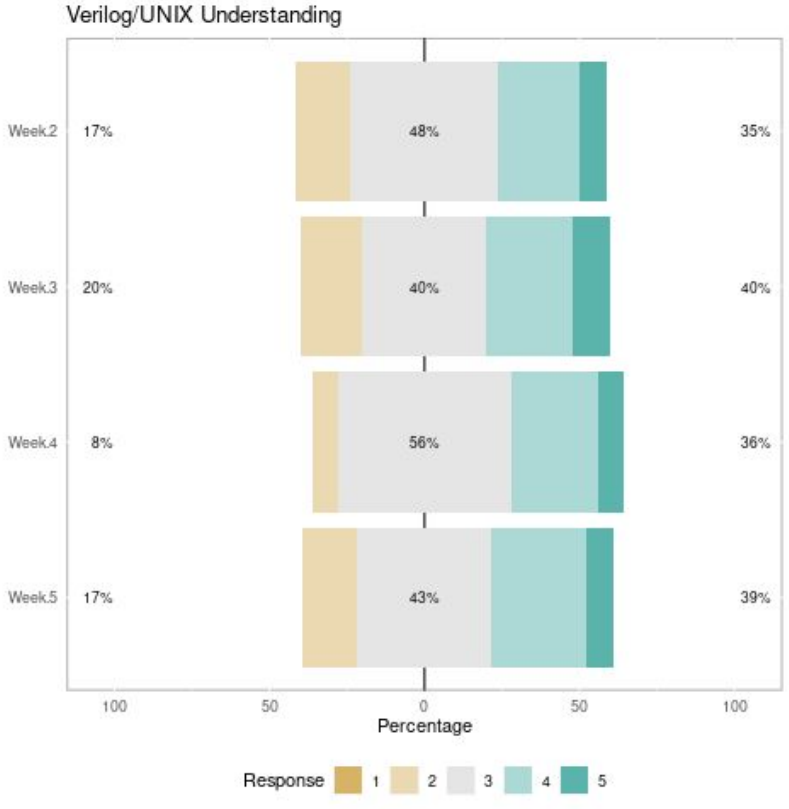
Feedback - Understanding



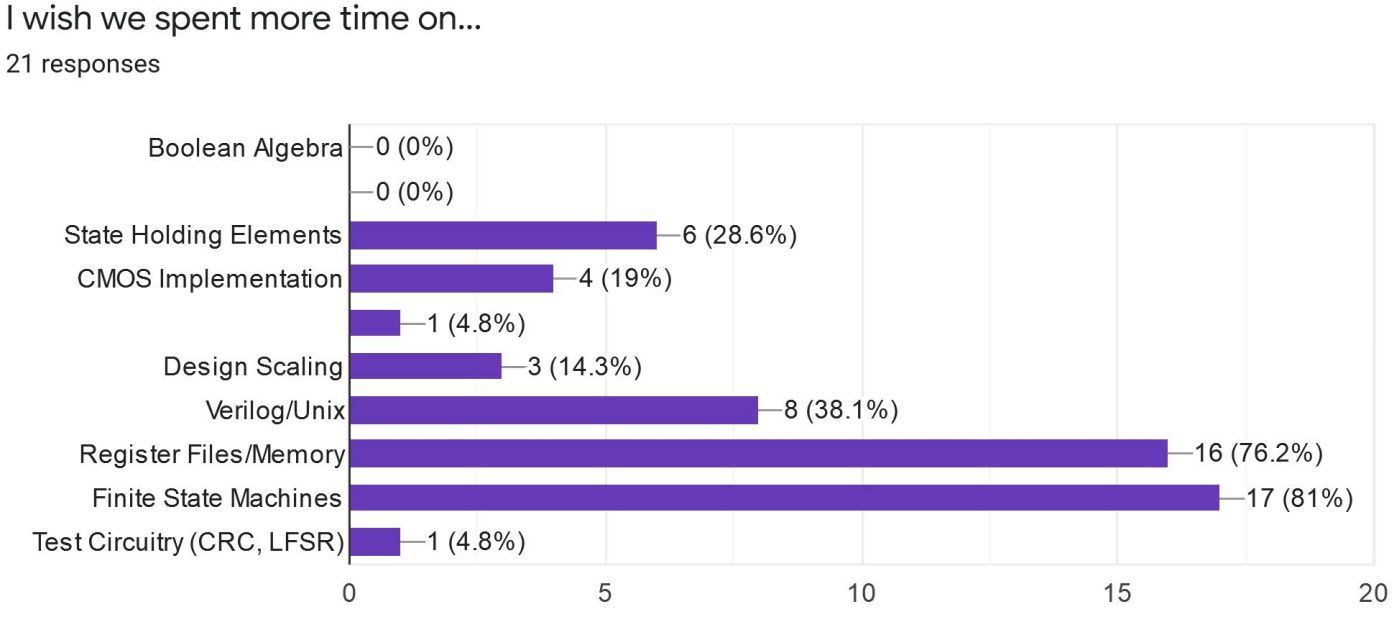
Feedback - Lecture Pace



Feedback - Verilog/UNIX



Feedback - More Time On...



Feedback - What's on your mind?

Why is verilog and unix joined? I feel comfy in UNIX but verilog is a lot

I know for the last lab I did kind of put off most of the work for the last 5 days. It was hard because NINJAs cancelled their hours over that weekend and our group would plan to go to those hours to ask for help but then find they weren't happening. So we had to stay up until 3am to finish the last lab and it was rough. If possible it would be good for NINJAs to not cancel their hours on the few days coming up to a lab.

Thinking ahead to the final project, I'm not really sure where to get started with finding project ideas. Would love to get some guidance about this. I also think we went over the one-hot encoding vs binary encoding part of the lecture reallllly fast and it was really hard to understand the one-hot encoding implementation.

FSM recitation was super helpful in understanding today. Otherwise I might have been completely lost.

Where We Are

- Done with first class: Digital Logic
- Now, Computer Organization + Design
- Final Project starts Computer Architecture

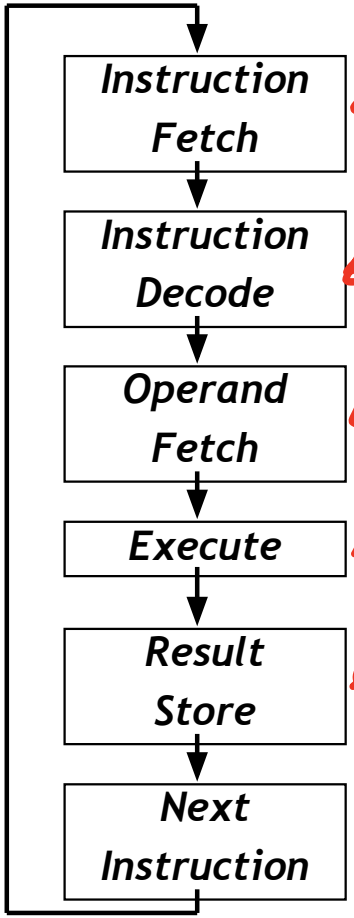
Context

- We now have almost all of the building blocks of a computer!
  - Nearing completion of our first “spiral”
- Today we will glance at the important ones
- Moving forward we will dive deeper into each of these building blocks to expand capabilities

Context

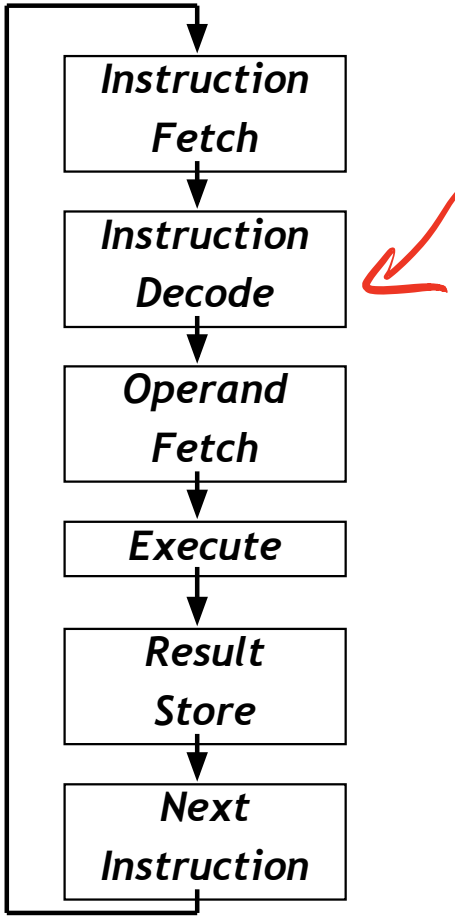
- Abstraction and box drawing
- Practice thinking hierarchically
- So… what does a CPU do?

CPU Execution Overview



- Fetch instruction from memory
- Decode instruction into actions/controls
- Fetch/Decode operands
- Compute result value or status
- Push result(s) to storage
- Determine next instruction

What boxes do we need?

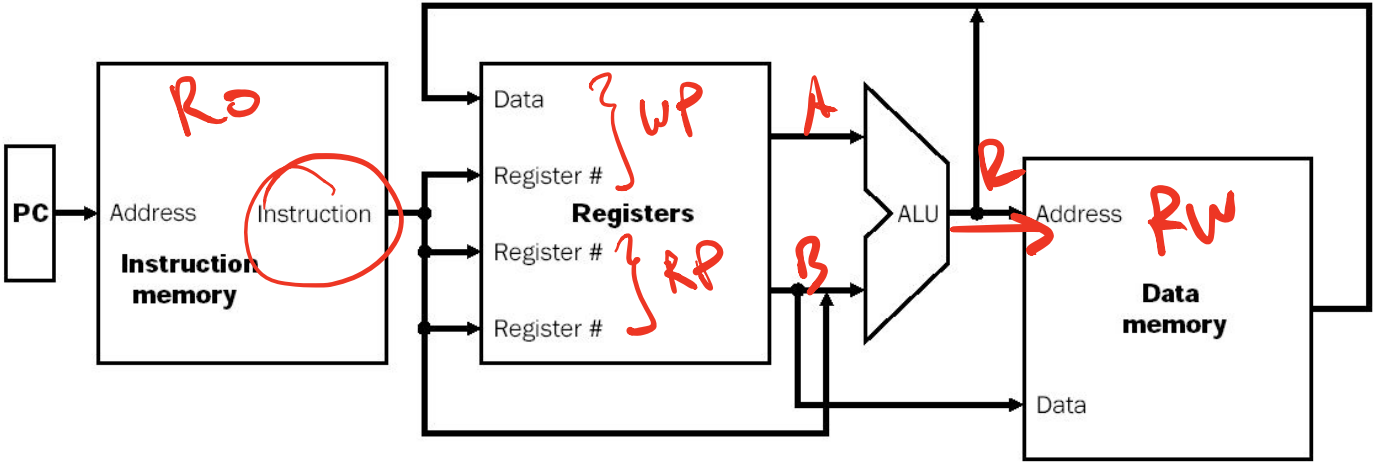


- Memory for Instructions
- Something to "unpack" instructions
- Memory for data
- Something to compute stuff

Processor Overview

Overall Dataflow

- PC fetches instructions
- Instructions select operand registers, ALU immediate values
- ALU computes values
- Load/Store addresses computed in ALU
- Result goes to register file or Data memory

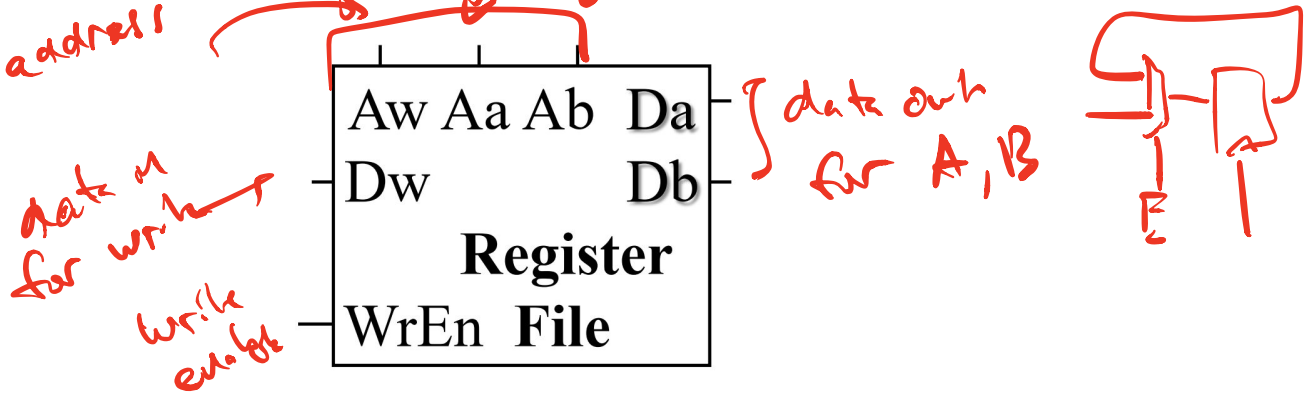


Program/Instruction Memory

- Large, single ported read-only memory
  - For today
- Stores the program the processor executes
- Instructions are 4 bytes wide
  - For today

Register File

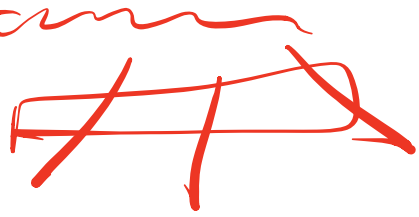
- Very small, *very* fast multi-ported memory
  - Small depth, large silicon area
- Each cycle reads 2 words, optionally writes 1
- This is the "scratch space" for the processor



Assembly

- The human readable version of the instructions the machine executes
- Closely related to “Op Codes” and Instructions
- We will visit this in more detail soon

Instruction Decode



- Today we are “black boxing” instructions

- Blue Wires come from the Instruction Decoder
  - Control signals that tell the CPU what to do
  - E.g. ALU Control Lines
  - Addresses

Instruction RTL

- RTL is Register Transfer Level

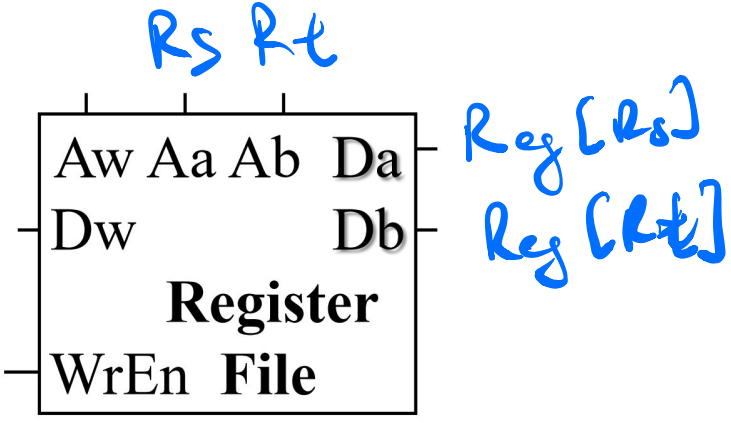
- Short Hand for what the instruction should do

```
Reg[Rd] = Reg[Rs] & Reg[Rt]
```

- "Set the memory at location Rd equal to the data at location Rs boolean anded with the data at location Rt."

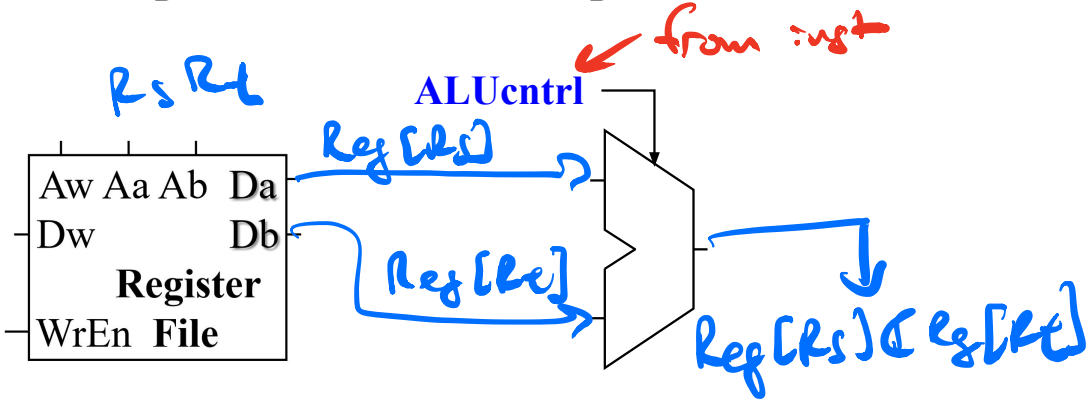
Fetch Operands

- Get Operand Addresses
- Pull from Register File
- `Reg[Rd] = Reg[Rs] & Reg[Rt]`



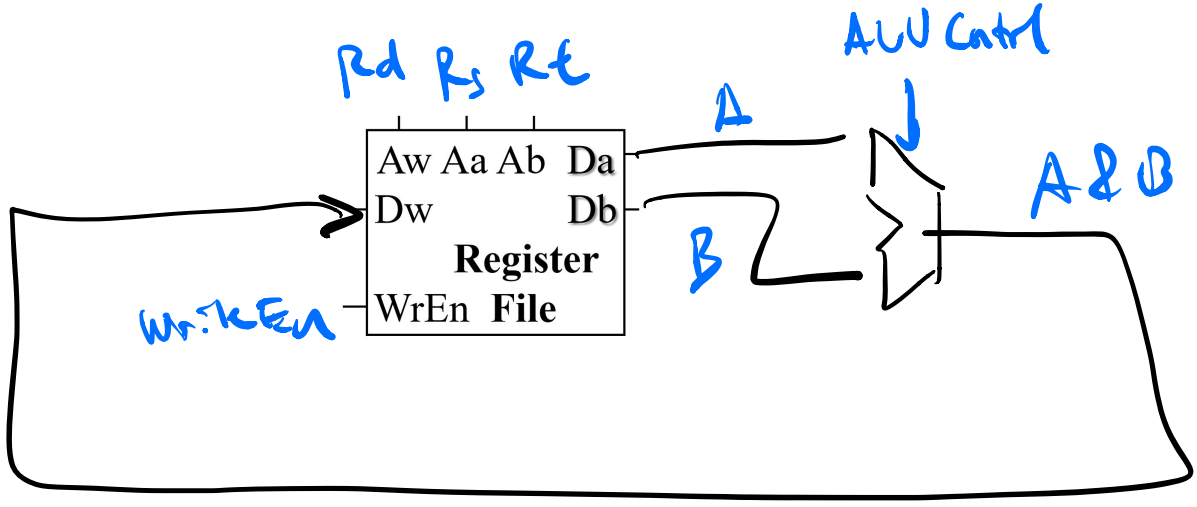
Execute

- ALU controls from Instruction Fetch/Decode
- Perform actual computation in ALU
- `Reg[Rd] = Reg[Rs] & Reg[Rt]`



Store Results

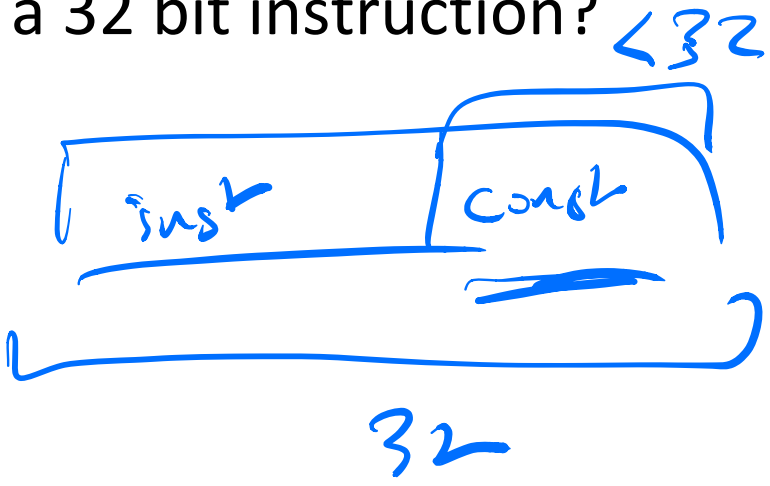
- Store the result back to the register file
- Others Instructions have other effects
- **Reg[Rd]** = Reg[Rs] & Reg[Rt]



Using Constants

- Sometimes we want to use a constant as an operand instead of a Register.

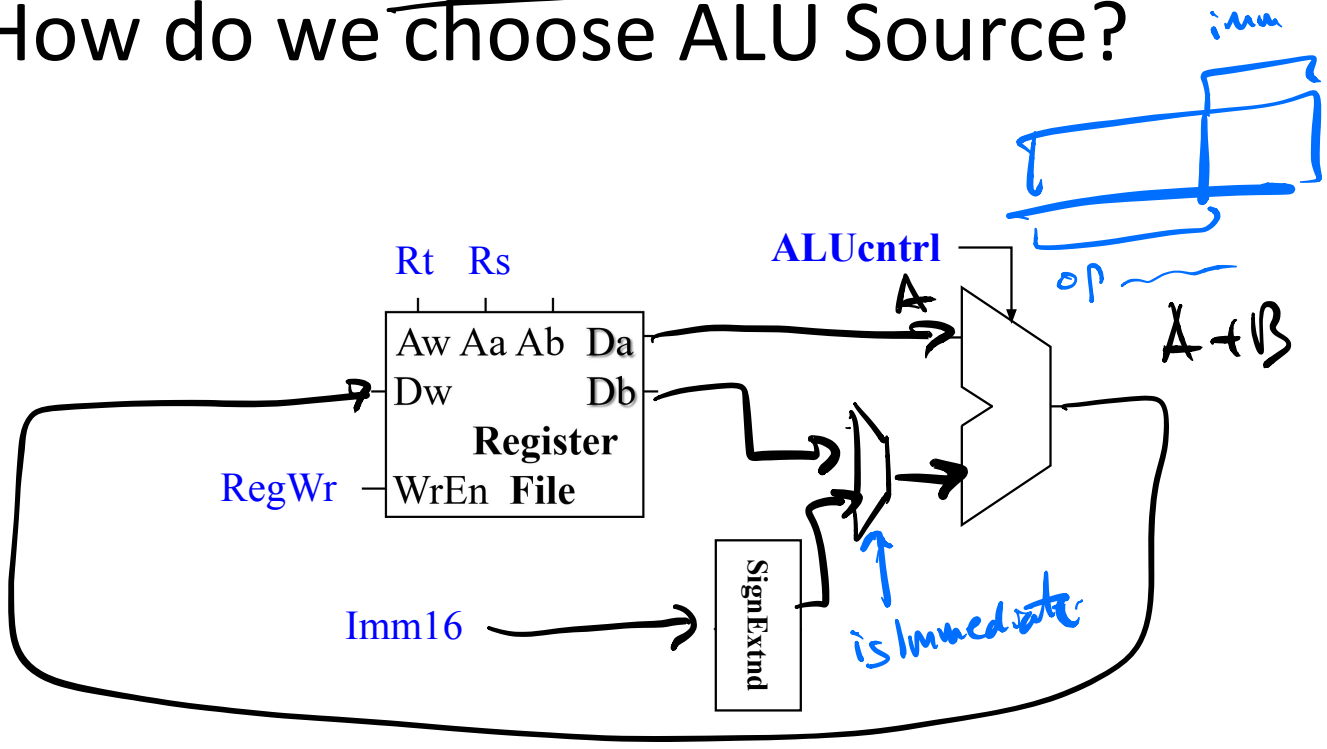
- How do you encode a 32 bit constant AND control signals into a 32 bit instruction?



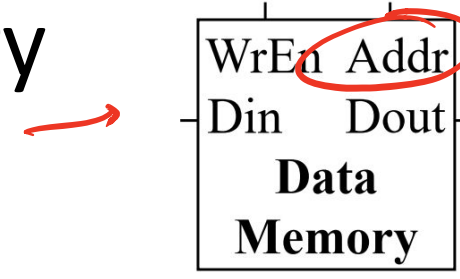
Using Constants (Immediates)

`Reg[rt]= Reg[rs]+`**`SignExtend(imm);`**

- How do we choose ALU Source?



Data Memory



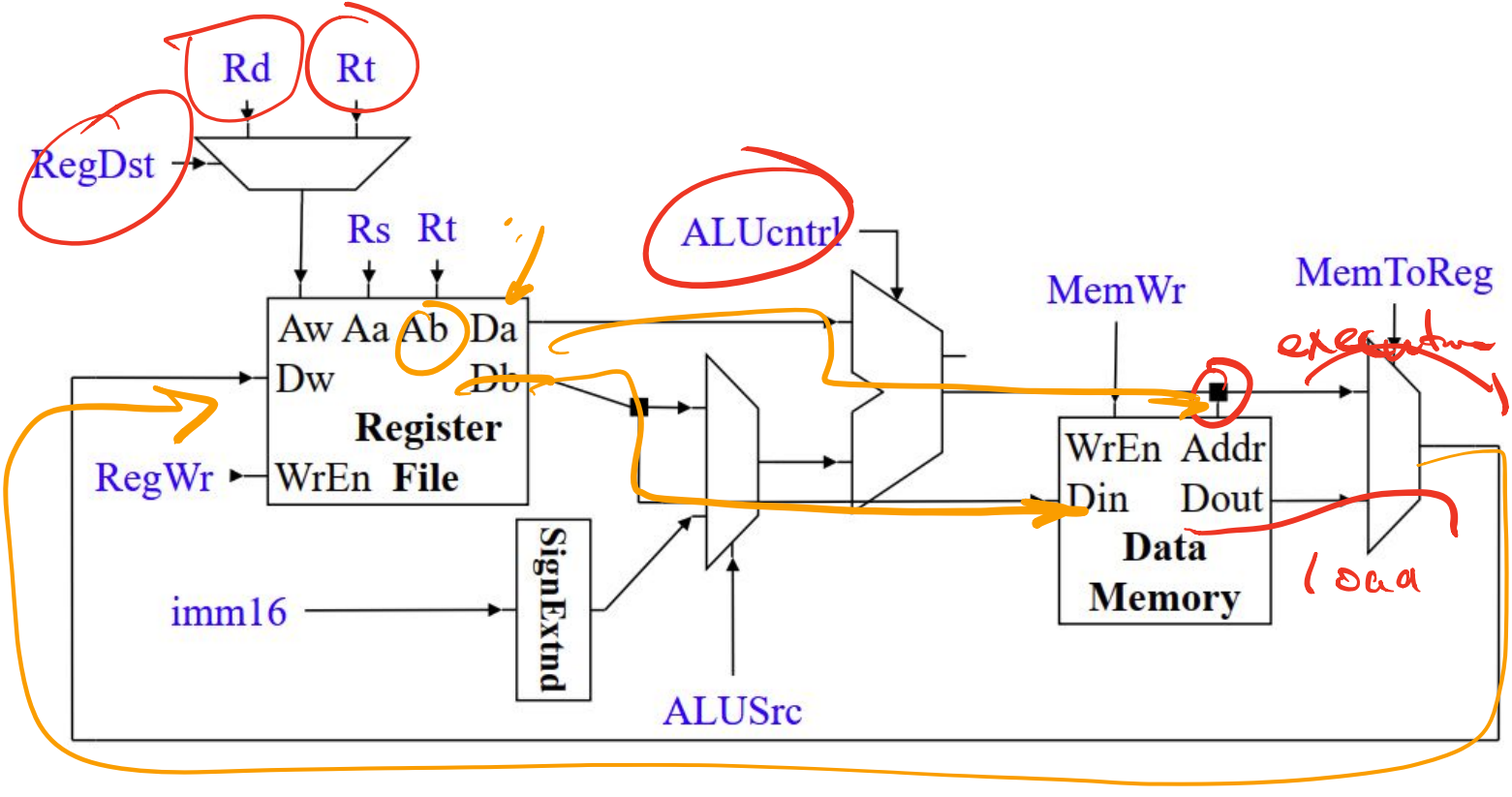
- Big Giant Slow Memory
  - Gigabytes!
- Read and Write, but only single port
  - Write Enable (WrEn) switches between the two modes
- Pulling from Data Memory to Register
  - Load
- Push from Register to Data Memory
  - Store

Quick Note on rd, rs, rt

- Aw is inconsistent in the MIPS standard
  - Sometimes rt
  - Sometimes rd
- This is ok!
  - Makes other things clearer later

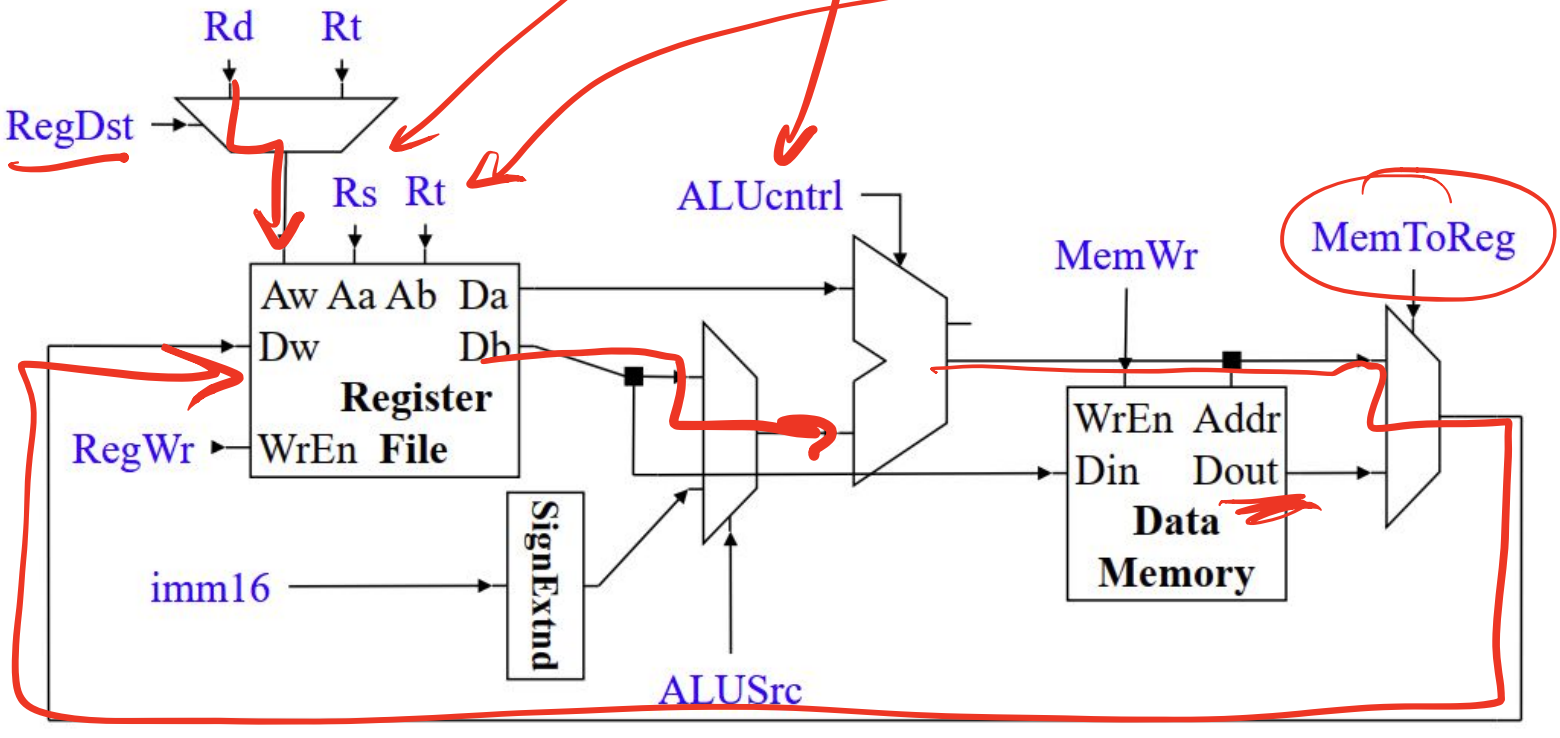
- Choosing between two signals? MUX!

One Possible Complete Datapath



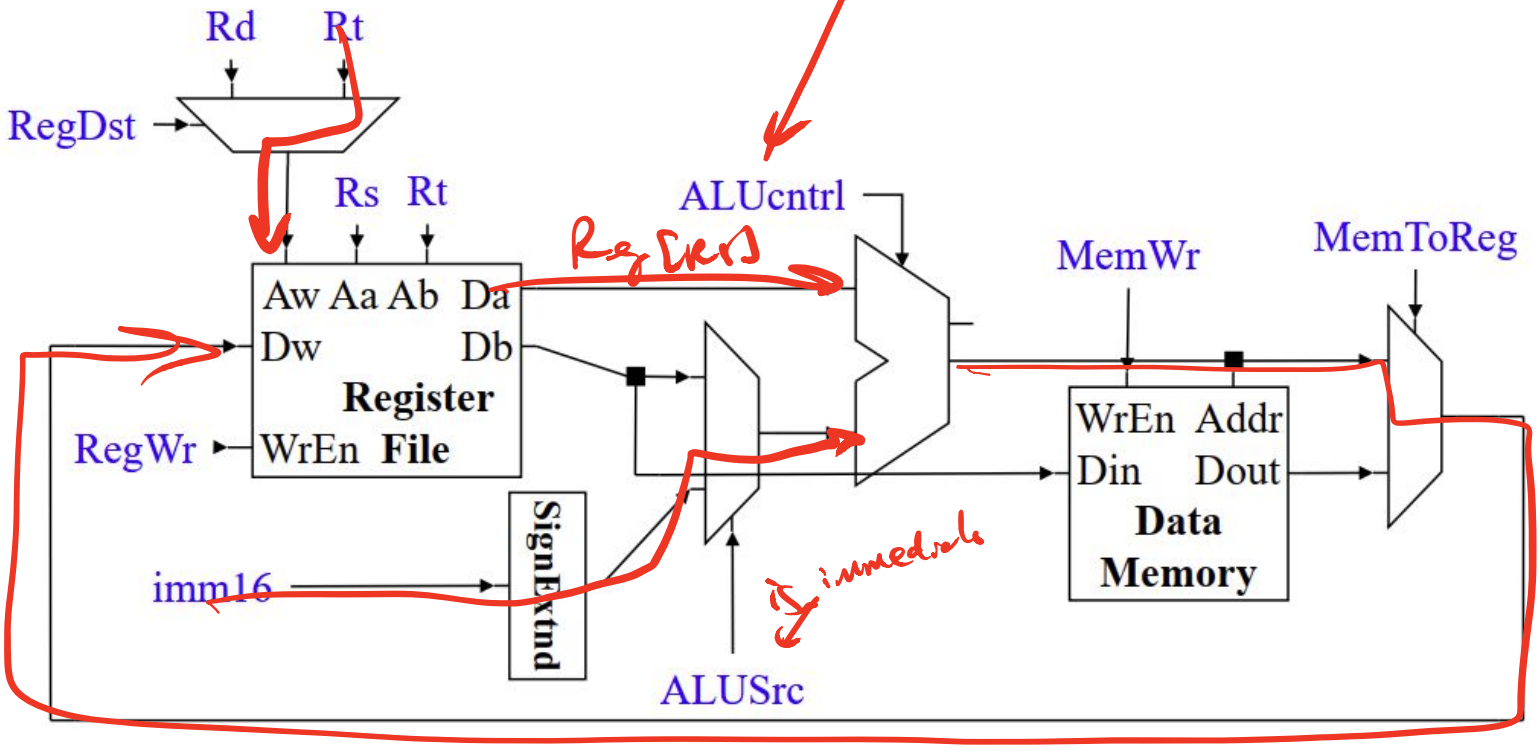
Math with 2 Variables

```
Reg[rd] = Reg[rs] op Reg[rt];
```



Math with 1 variable and 1 constant

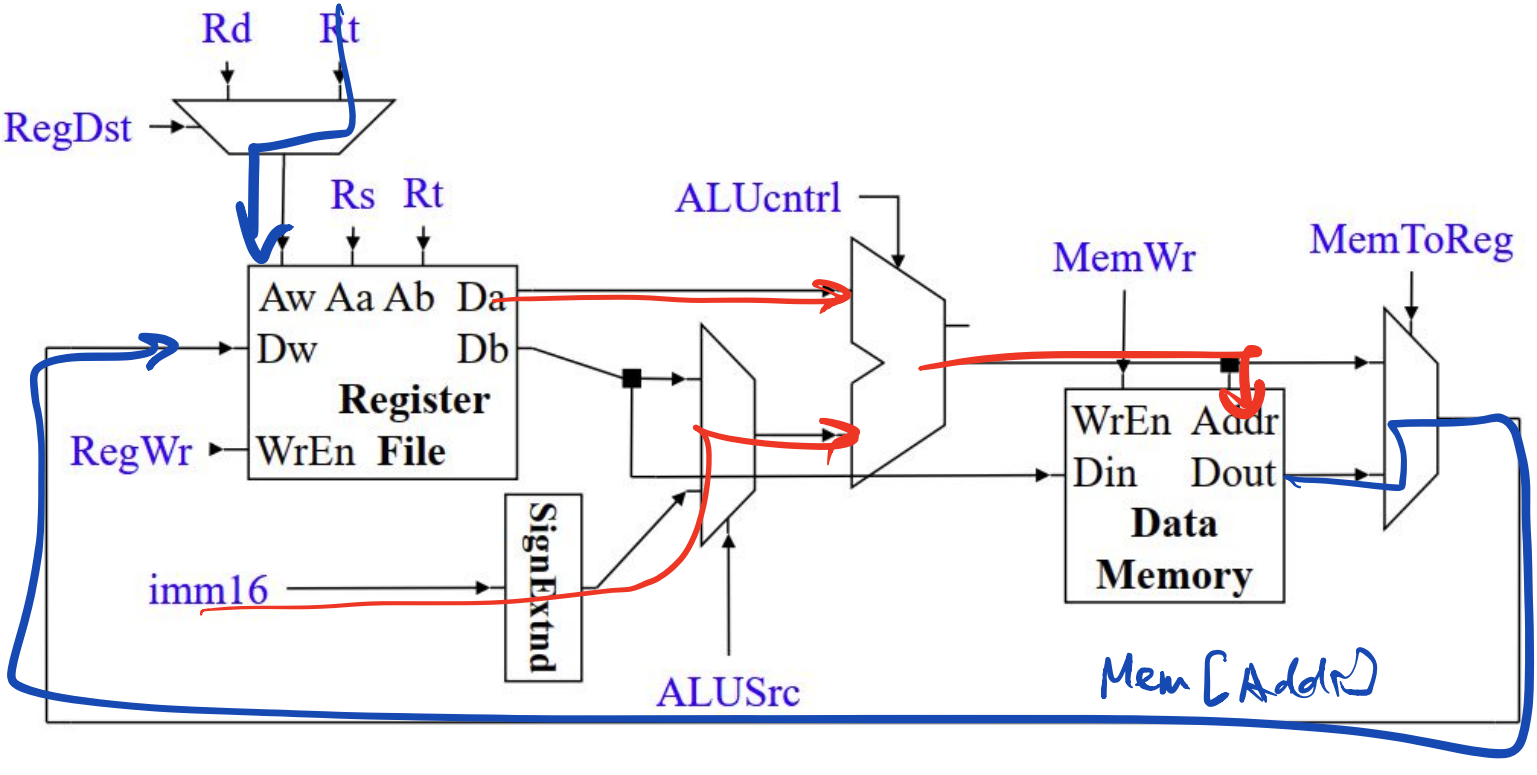
```
Reg[rt] = Reg[rs] op SignExtend(imm);
```



Load from data memory

```
Addr = Reg[rs] + SignExtend(imm);
Reg[rt] = Mem[Addr];
```

2 cycles



Store to data memory

```
Addr = Reg[rs] + SignExtend(imm);
Mem[Addr] = Reg[rt];
```