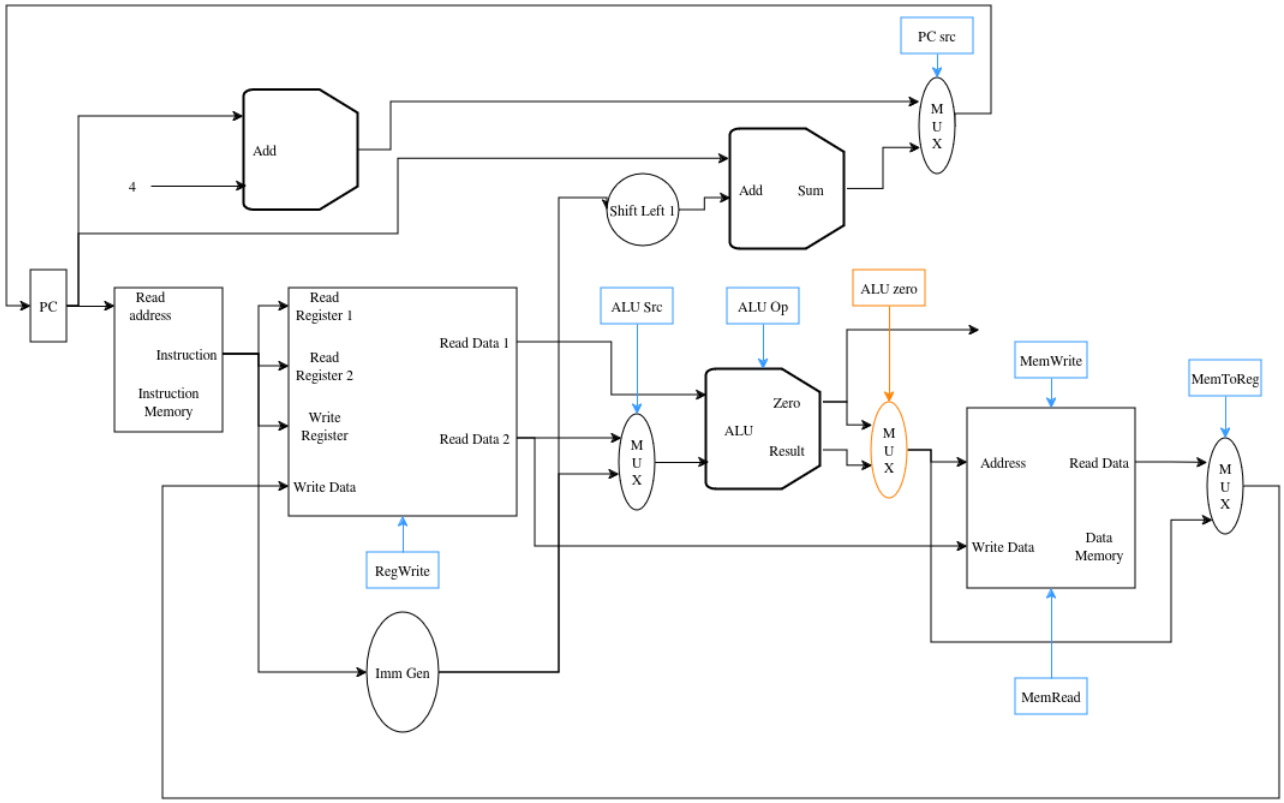
CSE 120
Day 9 Notes

Elijah Hantman

Recap Vanilla pipeline Five stages with pipelining between each stage.



Data Hazards

- One instruction has a source operand that is the result of a previous instruction in the pipeline.

Control Hazards

- Execution depends on a branch instruction.
- Becomes worse with deeper pipelines, ie: k is large.

> Creates a tradeoff. Deep pipelines can increase clock frequency, however deep pipelines can run into many control hazards which causes massive slow down.

Data dependency

- True Dependence, can cause RAW hazard

```
add t0, t1, t2
sub t3, t4, t0
```

- Output Dependence, can cause WAW hazard

```
add t0, t1, t2
sub t0, t4, t5
```

- Anti Dependence, can cause WAR hazard

```
add t0, t1, t2
sub t1, t3, t5
```

RAW example

5 Instructions fed into the pipeline.

```
sub x2, x1, x3  ;start of dependence
and x12, x2, x5 ;Hazard starts
or x13, x6, x2
add x14, x2, x2 ;Hazard ends, depends on when writes happen in cycle
sd x15, 100(x2) ;first instruction is readable
```

Fast register? A fast register will force writes to happen earlier in the clock cycle than the read, so it can be written to and read in the same cycle.

If a register is not fast when the read happens is undefined.

Solutions for RAW hazards

- One way is to stall the pipeline.
  - First you detect the hazard
  - pause execution
- We can delay the instruction in the sequence?
  - NOP can delay instructions
  - Advantage: simplifies hardware massively to not need to detect hazards.
  - Disadvantage: programs are pipeline specific. The number of NOPs are dependent on the pipeline.
- Hardware automatically inserts NOPS
  - Advantage: correct for all programs
  - Disadvantage: may miss some optimization opportunities
- Most modern machines use hardware NOPS

Manual NOP Insertion

One common NOP is

```
addi x0, x0, 0
;or you can use
00000000000000 ;this is an instruction with 0s in all 32 bits
```

By inserting NOPS we can prevent hazards. NOPS also tend to degrade performance, because they still force alignment to a clock cycle divisible by k, which will affect CPI.

Data Forwarding

- What if we can detect RAW conflict?
- We can generate the most recent value in the pipeline, then we can backpropagate it to the preceeding stage.
- When should we forward values to avoid RAW?
- Figure out when the RAW will take place.
- You can use a Mux alongside a dirty bit

  Set the bit when you have a value that needs to be forwarded, unset the bit when it gets written back.

  Maybe multiple bits to keep track of which registers are "dirty"
- forward unit that sits between ALU and previous pipeline registers to decide whether to data forward or not.

  All thats needed is to read the write signals as well and you can simple compare whether the write signals match, and if so you use data forwarding.
- When forwarding you have to make sure it only forwards from the most recent dependence.
- Forward pipelining can get complicated super fast

Forwarding for L/S

- Consider:

```
ld x10, 40(x1)
sd x10, 20(x3)
```

- Can forwarding save from unwanted NOPS?
- The address is avalible in time to write to write to memory, however we need to manage forwarding into Data memory stage.
- If a bottleneck stage, data forwarding can negatively affect the clock speed.

Hazard Detection for stalling the Pipeline

Forwarding can't always save the day.

```
ld x2, 20(x1)
and x4, x2, x5
```

If you don't get the value in time to use it data forwarding is physically unable to help.

The above scenario can only e solved via stalling.

Hazard detection condition for compulsory stall check textbook

How do we introduce a stall?

We insert a Nop directly into the pipeline and delay the previous stages by one.

How do we generate bubbles?

We add a control to the PC and IF/ID pipeline registers. The control lines prevent the PC from advancing, and the IF/ID registers from being written to.

It will also pass on a zero instruction to the stages after the Decode. This prevents smashing memory or registers.