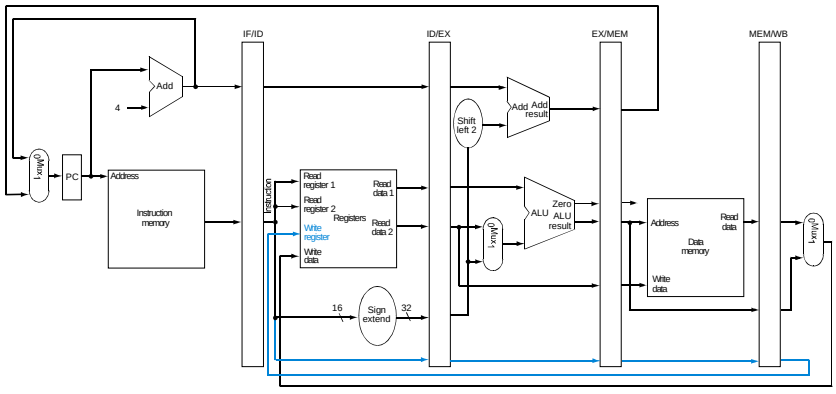
Lecture 12: Pipelining Hazards

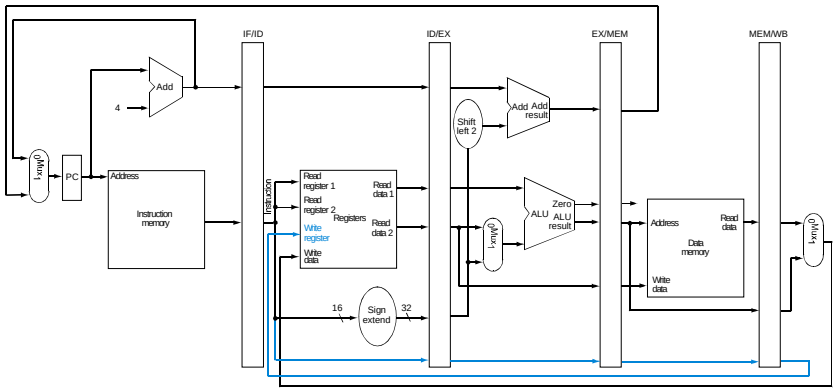
- Administrative
  - HW #3 due
  - HW #4 handed out, due Thursday after spring break
  - Exam #1 handed out
- Review pipelining
  - 5 stages
  - Data Hazards
    - Fixing them with forwarding
  - Memory Hazards
    - Fixing them with stalls
- Today
  - Control hazards
    - Branch delay slot, branch prediction
  - Branch prediction

Split machine into 5 pipeline stages

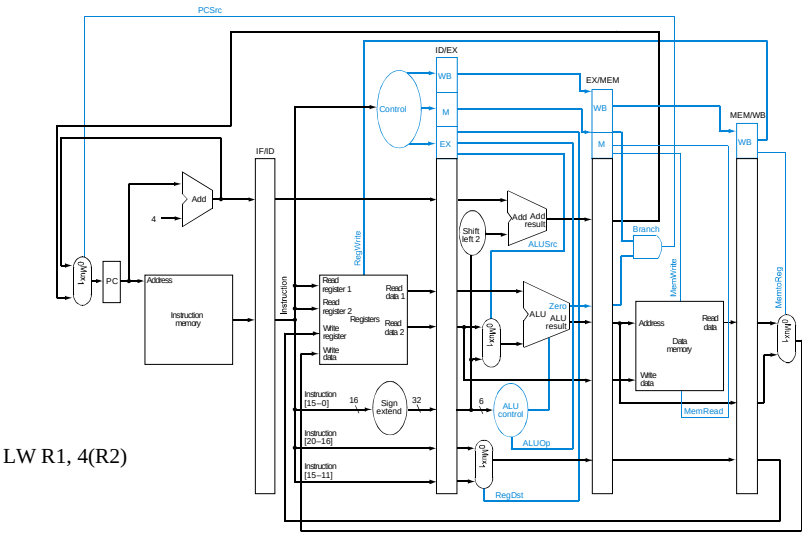


It's like attaching a note to the laundry basket used for a particular load

Several instructions in progress at once



Control signals are carried with each instruction



LW R1, 4(R2)

Three kinds of Pipeline Hazards

- Data hazards
  - an instruction uses the result of a previous instruction (RAW)

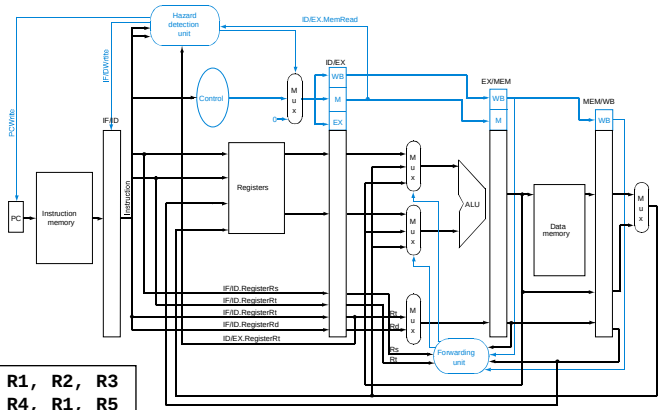
```
ADD   R1, R2, R3        or      SW    R1, 3(R2)
ADD   R4, R1, R5                LW    R3, 3(R2)
```

- Control hazards
  - the location of an instruction depends on a previous instruction

```
        JMP   LOOP
        ...
LOOP:   ADD   R1, R2, R3
```

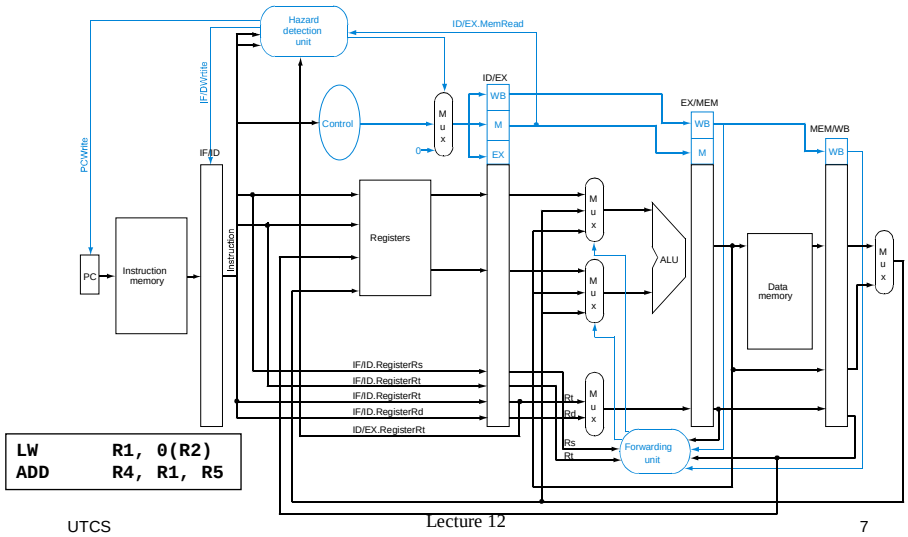
- Structural hazards
  - two instructions need access to the same resource
    - e.g., single memory shared for instruction fetch and load/store

Bypassing: send data directly to where it's needed




```
ADD     R1, R2, R3
ADD     R4, R1, R5
SUB     R5, R1, R6
XOR     R7, R8, R1
```

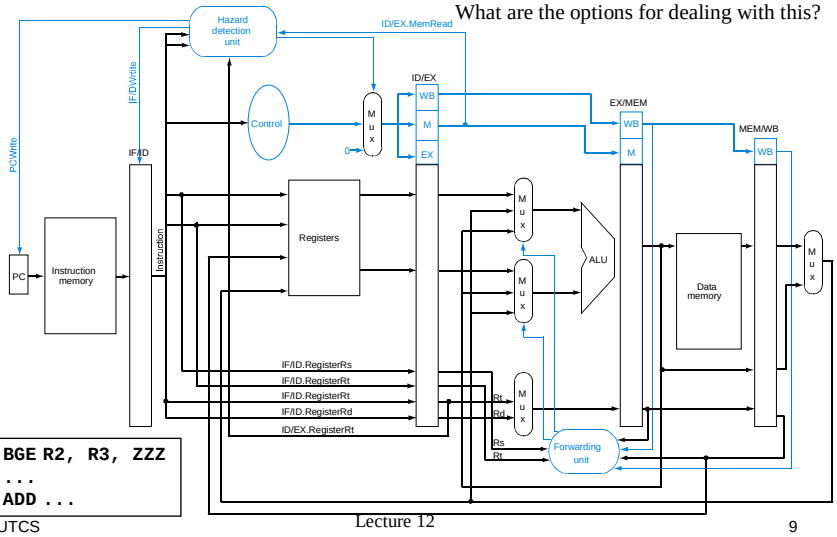
Memory hazards must stall: The "Hazard detection unit" inserts a NOP



Control Hazards

Control Hazard: Branch on condition

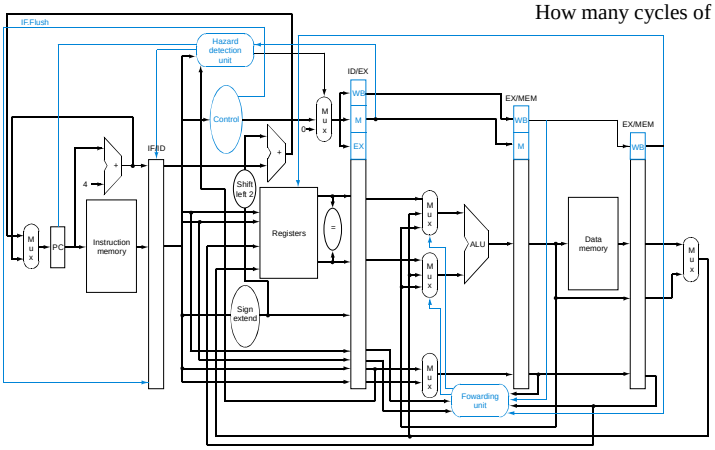
At what stage do we know if conditional is true?
What are the options for dealing with this?




```
      BGE R2, R3, ZZZ
      ...
XX:   ADD ...
```


Move branch logic to ID stage

How many cycles of stall now?




```
      BGE R2, R3, ZZZ
      ...
XX:   ADD ...
```

Branch Delay Slots

- Since we need to have a dead cycle anyway, let's put a useful instruction there

- Advantage:
  - Do more useful work
  - Potentially get rid of all stalls

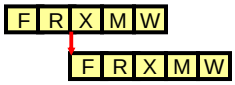
- Disadvantage:
  - Exposes microarchitecture to ISA
  - Deeper pipelines require more delay slots

```
ADD R2,R3,R4
BNEZ R5,_loop
NOP
```

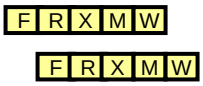
```
BNEZ R5,_loop
ADD R2,R3,R4
```


Speculating for control hazards

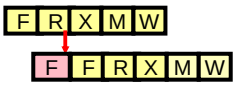
- Conservatively, the pipeline waits until the branch target is computed before fetching the next instruction.



- Alternatively, we can speculate which direction and to what address the branch will go.



- Need to confirm speculation and back up later.



How can we predict branch direction?

- Past history of this branch
  - Last time
  - Last two time (why is this useful?)
- Past history of last several branches
  - Why is this useful?

```
for (i=0; i<1000; i++) {
    for (j=0; j<1000; j++) {
          if (j == 0) {
                foo;
          }
          bar;
    }
}
```

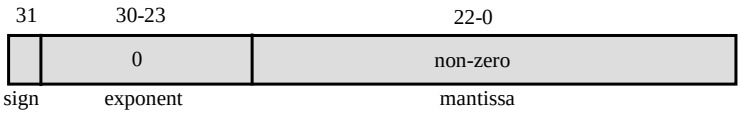

Control Hazards Summary

- Three approaches
  - Stall until new PC is known
  - Speculate that branch goes a particular way
    - If guess is right, great!
    - If guess is wrong, kill off speculated work
  - Delay slot
- Delay slot is only approach visible to programmer!
  - Unfortunately, MIPS picked this approach!

Exceptions – implicit conditional branches

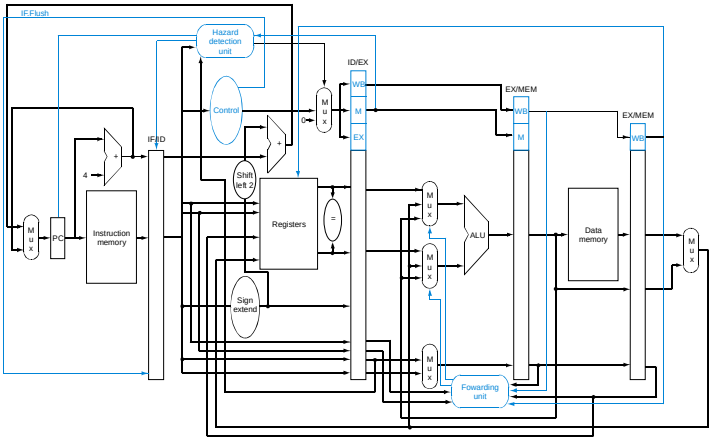
- Examples of exceptions
  - Overflow of result
  - Page fault on load
- On an exception, branch to some address
- But – no explicit branch instruction!
- Architectural issues:
  - What exceptions are supported?
  - Are they "precise"?
    - i.e. behavior is as-if there was no pipelining
  - Adds significant complexity to implementation!

Denormalized number



Often, the floating-point hardware cannot directly handle these.
The floating-point unit generates an exception.

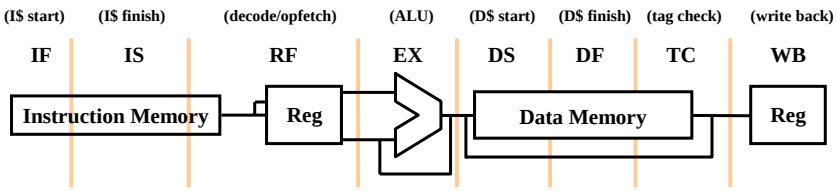
Exception example




```
ADD.S F1, F2, F3
MUL.S F4, F1, F5
```

What if F2 is a denormalized number?

R4000 Pipeline



- **How long is load delay?**
- **How long is branch delay?**
- **How many comparators are needed to implement the forwarding decisions?**
- **What instruction sequences will still cause stalls?**

How Do We Speed up the Pipeline?

- Pipeline too long ⇒ more ALUs (exploit ILP)
- WAR/WAW hazards ⇒ register renaming

```
ADD R1,R2,R3          ADD R1,R2,R3
SUB R1,R4,R5    ⇒     SUB R1',R4,R5
```

- Undetermined dependencies at compile time ⇒ dynamic scheduling
  - Object code compatibility
  - Simplify compiler
- Too many branches ⇒ better branch prediction
  - Or use predication to eliminate branches
- Unknown dependencies (control/data) ⇒ speculate
- Explicitly parallel architectures (VLIW / EPIC)

Summary

- Hazard detection and avoidance
- Improving Pipeline performance

- Next Time
  - Reading assignment: P&H 6.9 – 6.12