CSSE232
Computer Architecture I

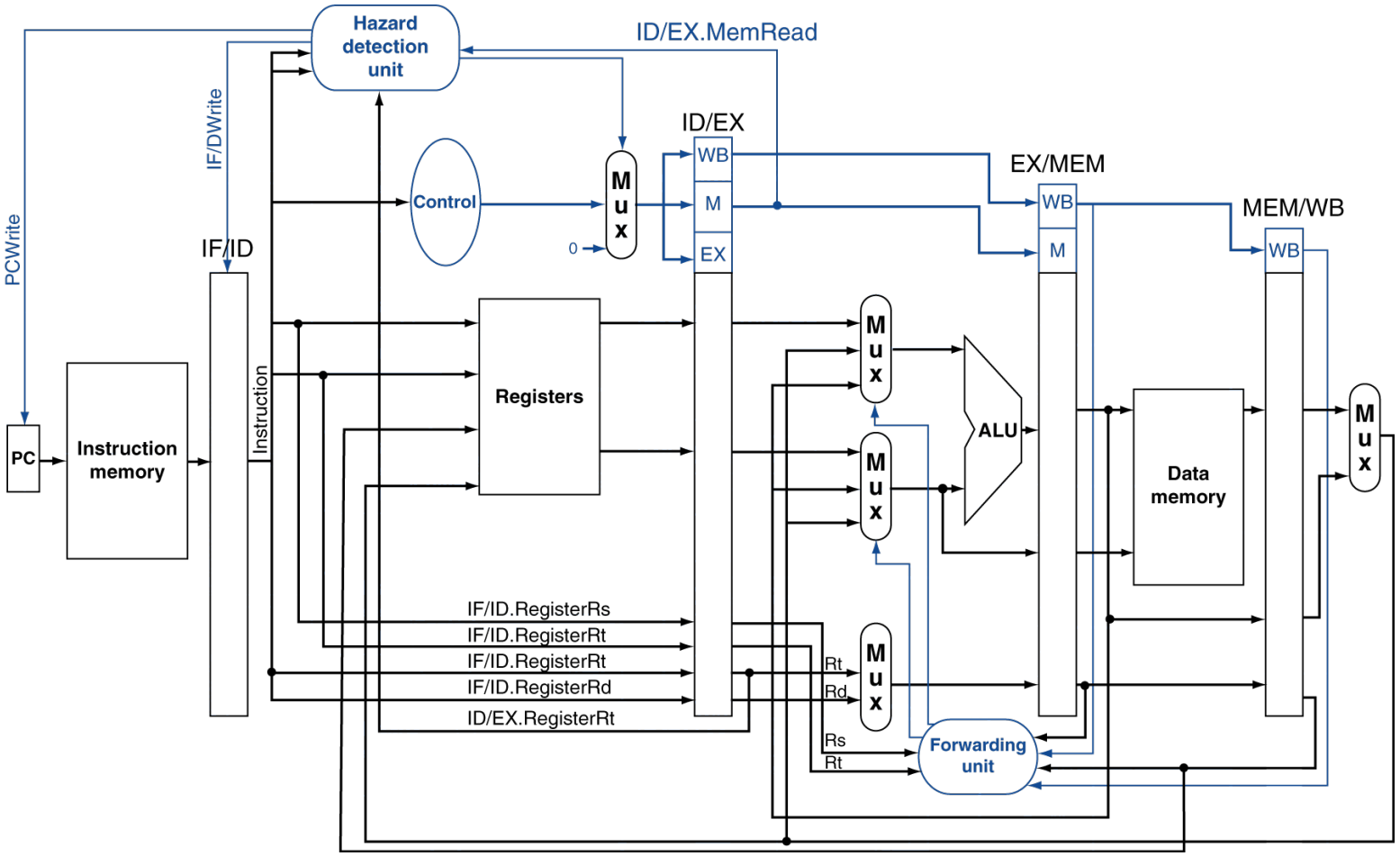
Control Hazards

Pipelining

- From last time…

- Data Hazards:
  - Create units to help with
    - Forward data
    - Detect when stalls are needed

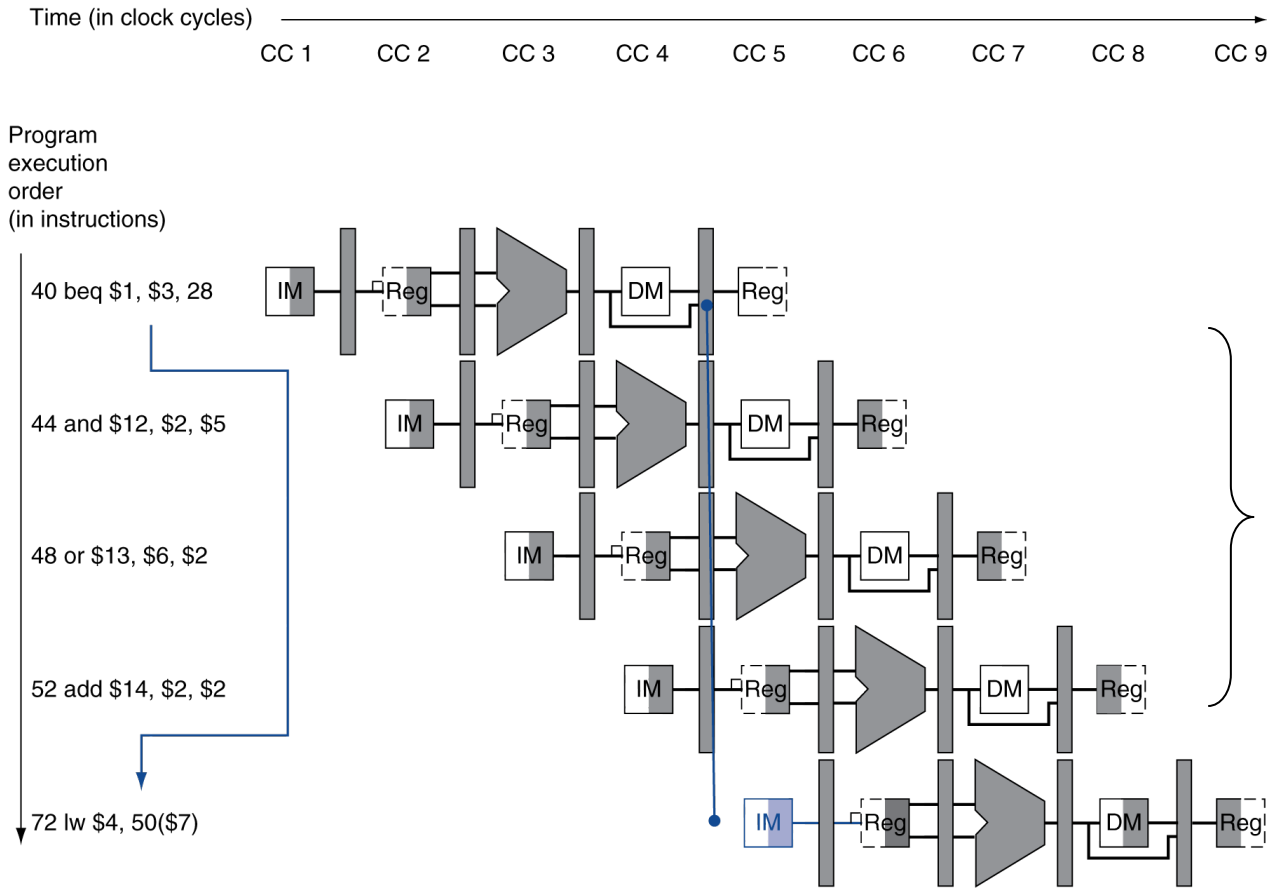
Datapath with Hazard Detection



Branching

- Previous datapaths had no issues with branching
- Pipeline datapath
  - Executing several instructions in parallel
  - Processing branch and subsequent instructions!

Branch Hazards



Flush these instructions (Set control values to 0)

Flushing

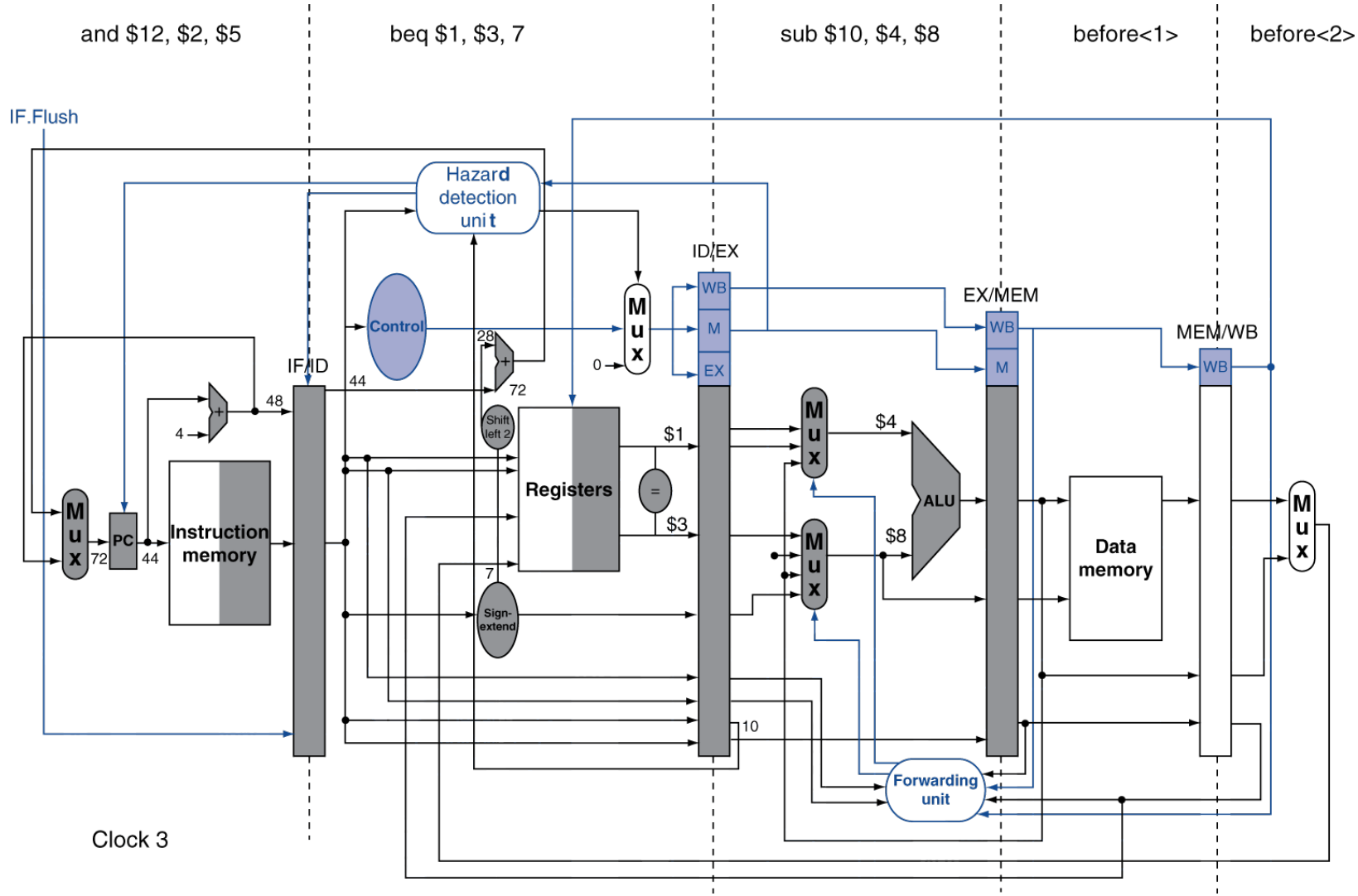
- Stall
  - Stopped updates to IF, ID
  - Cancelled EX for current instruction
- Flush
  - Cancel all instructions!
  - Set control to zero in for 3 instructions in flight

Reducing Flushed Instructions

- How to reduce wasted instructions?

Reducing Branch Flushing

- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- ALUs are slow
  - Subtaction is slow
- Equal is much easier
  - Just use XOR gates


and $12, $2, $5
beq $1, $3, 7
sub $10, $4, $8
before<1>
before<2>
IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
IF/ID
Mux
PC
Instruction memory
Registers
ALU
Data memory
Shift left 2
Sign-extend
Forwarding unit
$1
$3
$4
$8
48
44
72
28
4
7
10
Clock 3

Detecting Control Hazard

- Hazard unit can detect branch instructions, and automatically flush next instruction
- MIPS does not detect control hazards
  - Does not flush!
  - Will execute 1 instruction after branch
  - Require noop after all branches?
- Lets programmer choose to use the 'delay slot'
  - Can increment loop counter (or something else)

Other Control Hazards

- Jump and link
  - Greencard says PC+8
  - Assumes a delay slot
  - Return to instruction after delay slot

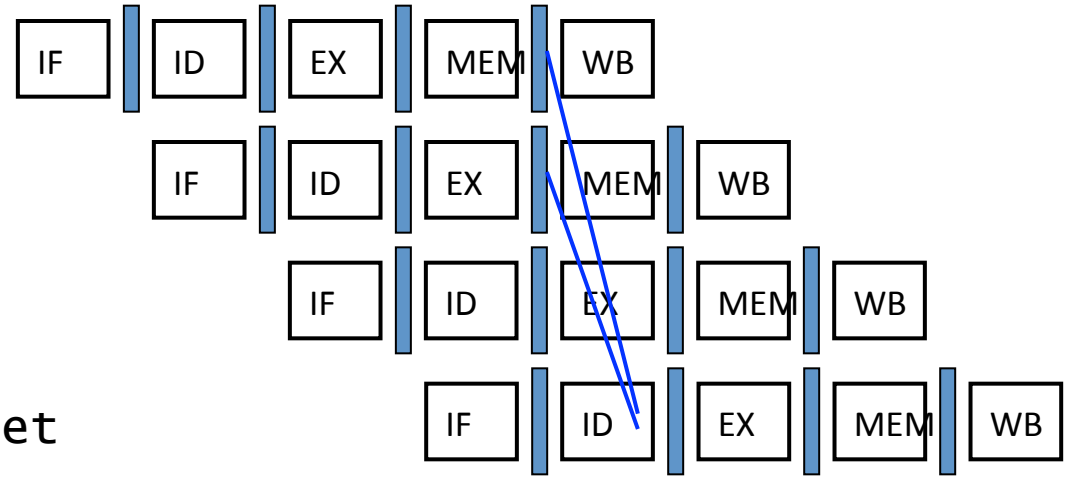
Other complications

- Flushing is one problem
  - MIPS solution is to reduce to single delay slot

- What about getting branch ready?

- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

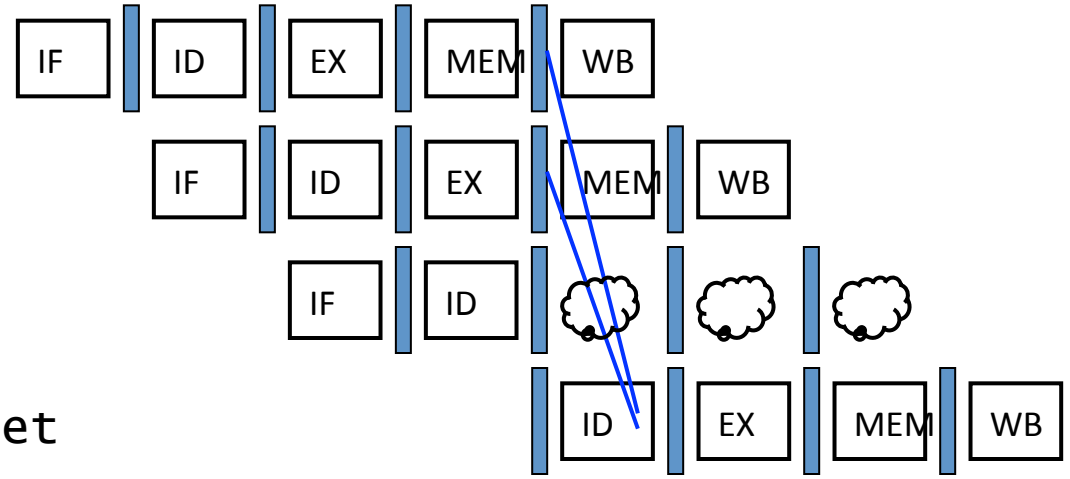
```
add $1, $2, $3
add $4, $5, $6
…
beq $1, $4, target
```



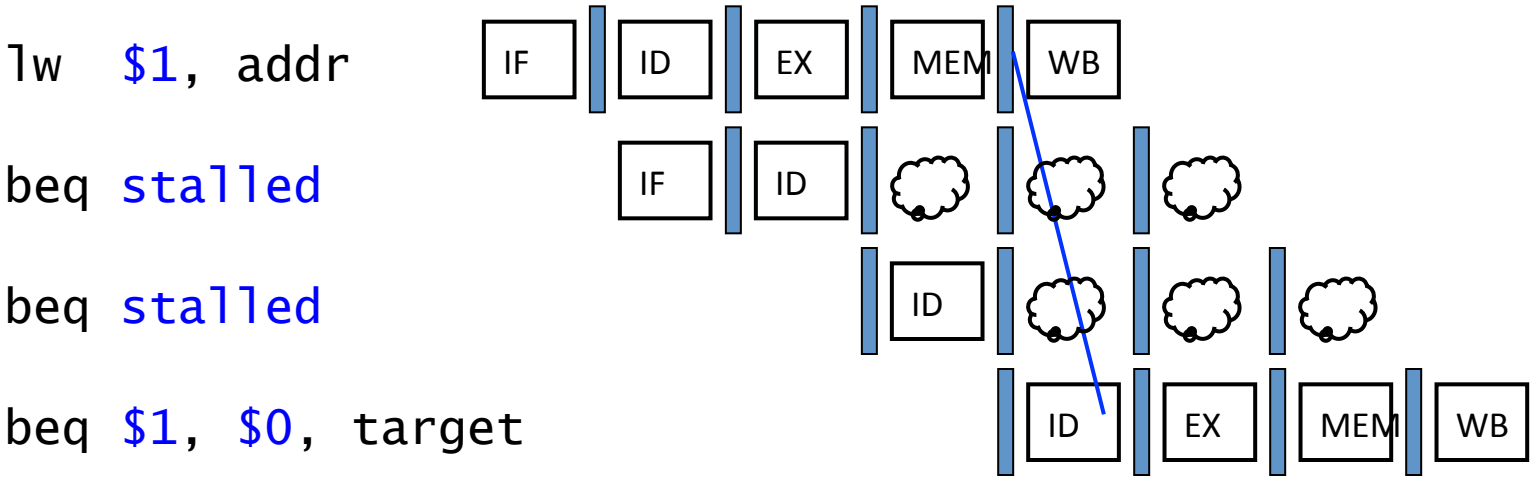
- Can resolve using forwarding

- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle

```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles



Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

Review and Questions

- Hazards
  - Data
  - Control
  - Structural
- Hazard detection
- Hazard prevention