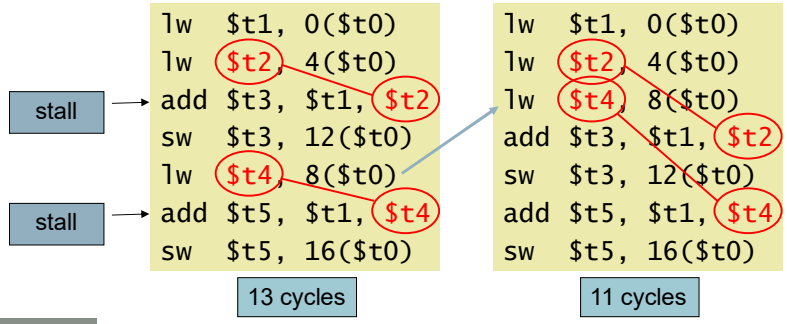
Stalls and Performance

The BIG Picture

- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

Code Scheduling to Avoid Stalls

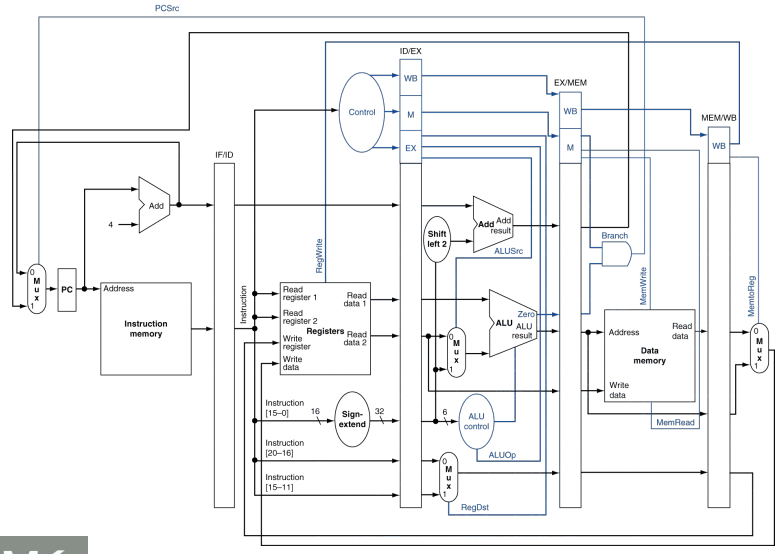
- Reorder code to avoid use of load result in the next instruction
- C code for `A = B + E; C = B + F;`



Control Hazards

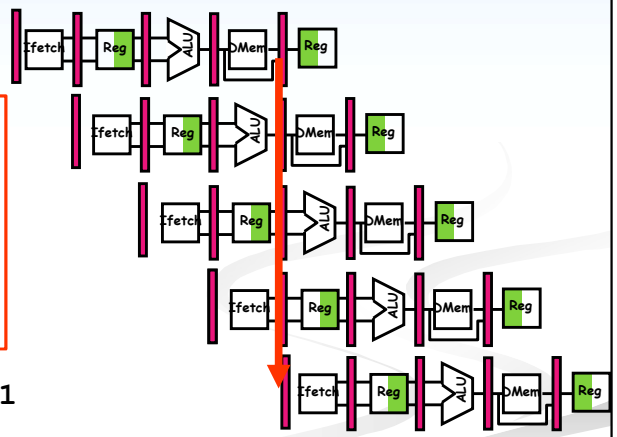
- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can't always fetch correct instruction
    - Still working on ID stage of branch
- In MIPS pipeline
  - Need to compare registers and compute target early in the pipeline
  - Add hardware to do it in ID stage

Pipelined Control



Control Hazard on Branches Three Stage Stall

```
40: beq r1,r3,28
44: and r2,r3,r5
48: or  r6,r1,r7
52: add r8,r1,r9
72: xor r10,r1,r11
```

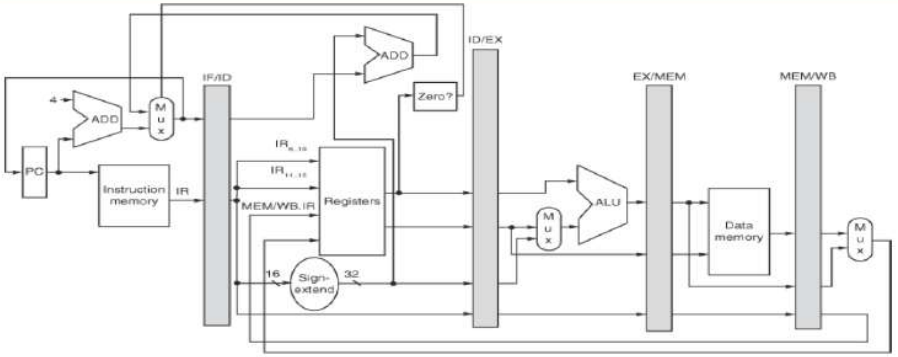


Assume branch decision made on MEM stage.
When branch is taken, the following 3 instructions should not complete execution.
Less frequent than data hazards, but
No effective solution like "forwarding". Better to decide branch earlier.

Example: Branch Stall Impact

- If 30% branch, stall of 3 cycles is significant
- Two part solution:
  - Determine if branch is taken or not sooner,
  - AND compute taken branch address earlier
- MIPS Solution:
  - Move Zero test to ID/EX stage
  - Adder to calculate new PC in ID/EX stage
  - Result: 1 clock cycle penalty for branch instead of 3

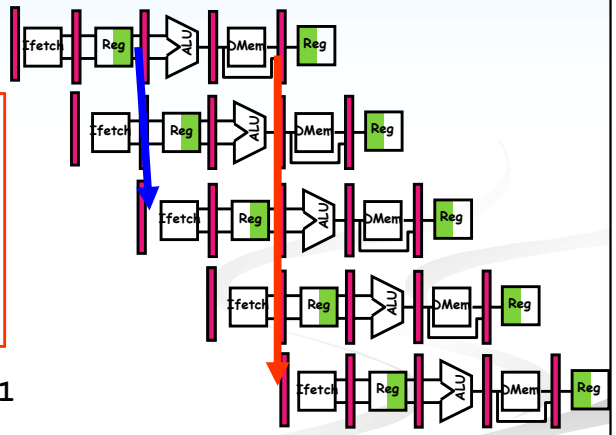
Reducing Branch Penalty by Anticipating the Branch Execution



- Branch-outcome resolution (zero test) and branch-target calculation moved into ID stage
  - PC is written in the IF stage using either the branch-target address computed during ID or the incremented PC computed during IF

Control Hazard on Branches One Stage Stall

```
40: beq r1,r3,28
44: and r2,r3,r5
48: or  r6,r1,r7
52: add r8,r1,r9
72: xor r10,r1,r11
```



Branch decision made on ID stage.
When branch is taken, the following 1 instructions should not complete execution.

Stall on Branch

- Wait until branch outcome determined before fetching next instruction