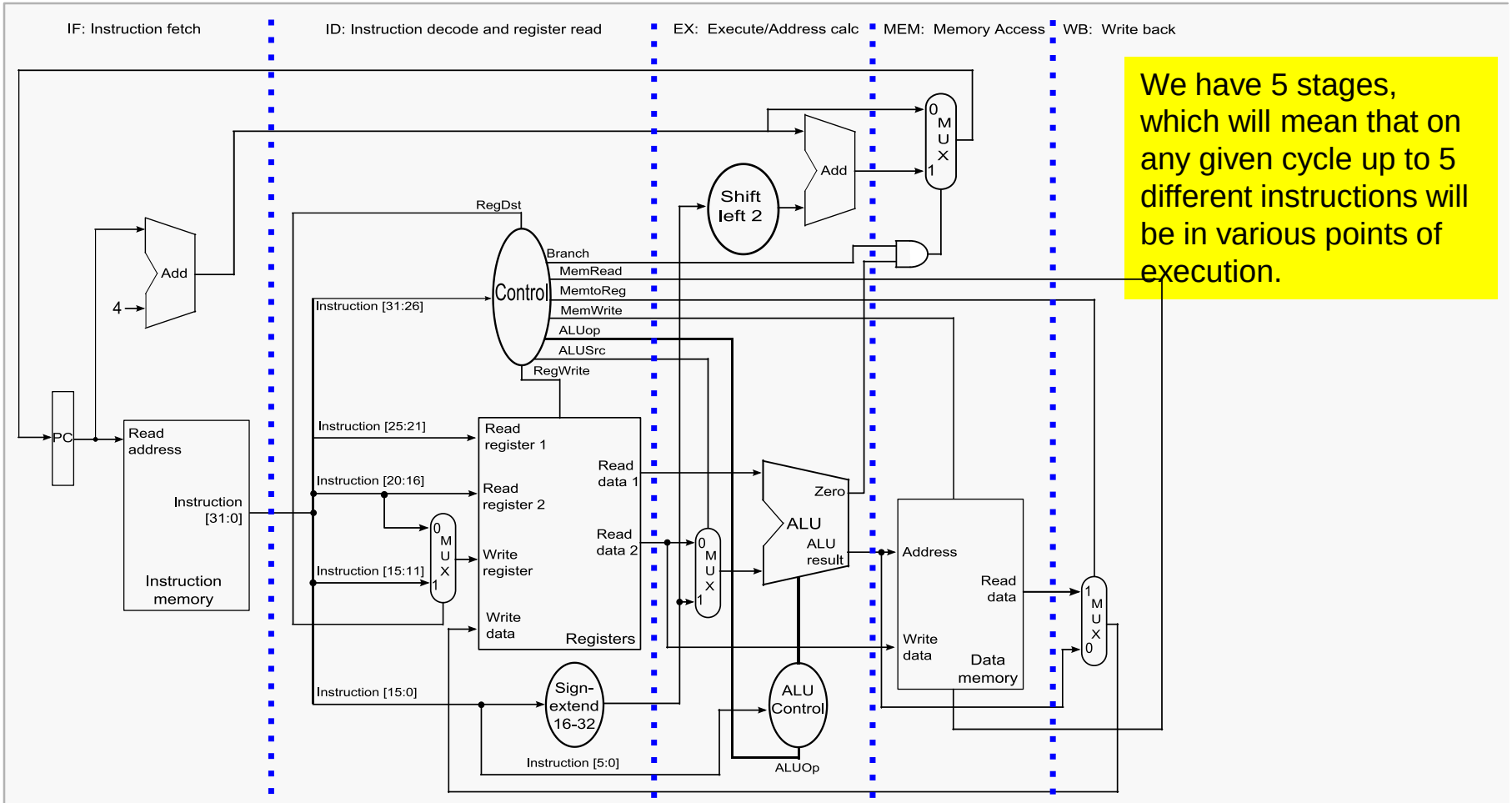
Recap: the Basic Idea



We have 5 stages, which will mean that on any given cycle up to 5 different instructions will be in various points of execution.

Can we operate the stages independently, using an earlier one to begin the next instruction before the previous instruction has completed?

Hazards

Situations that prevent starting the next instruction in the next cycle

Structural hazards

- A required resource is busy

Data hazard

- Need to wait for previous instruction to complete its data read/write

Control hazard

- Deciding on control action depends on previous instruction

Structural Hazards

Conflict for use of a resource

For example:

In MIPS pipeline with a single memory

- Load/store both require memory access
- So does instruction fetch
- Instruction fetch would have to stall for that cycle

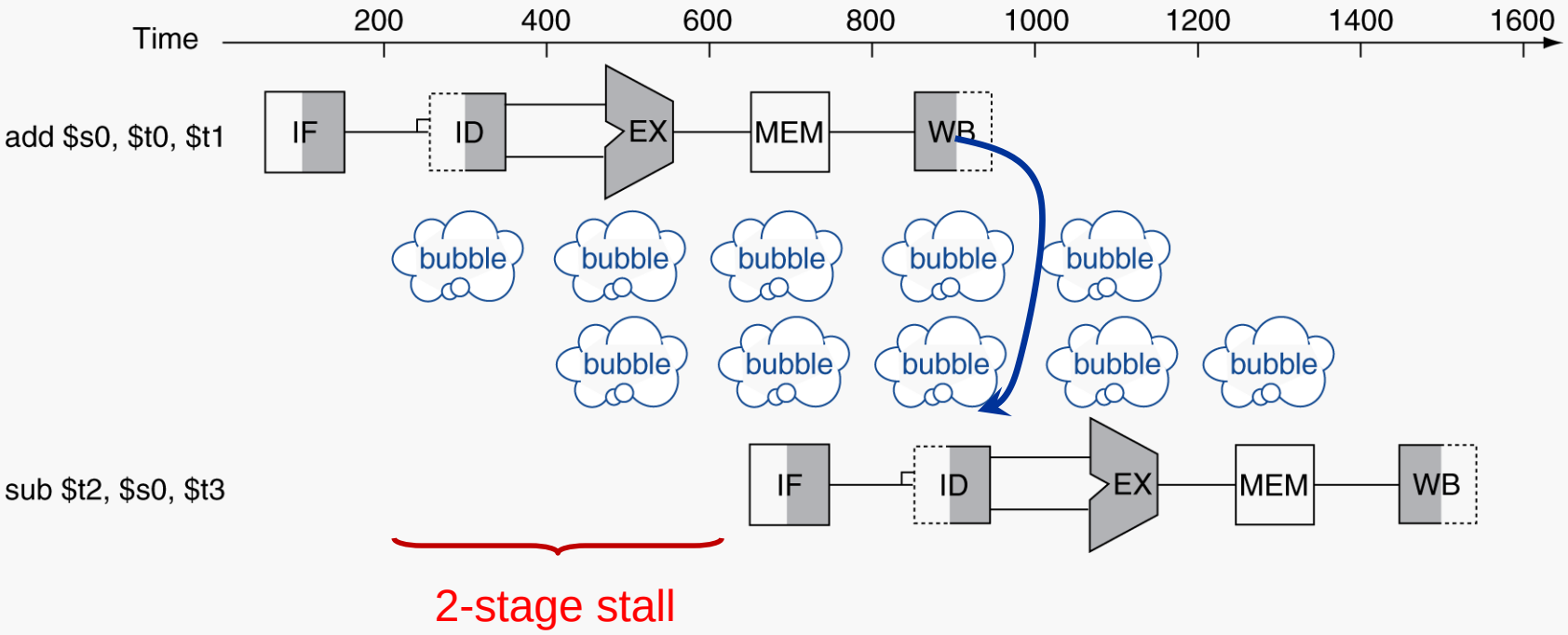
Hence, MIPS pipeline require separate instruction/data memories

- Or separate instruction/data caches
- Or dual-ported memories

Data Hazards

An instruction depends on completion of a write to a register by a previous instruction

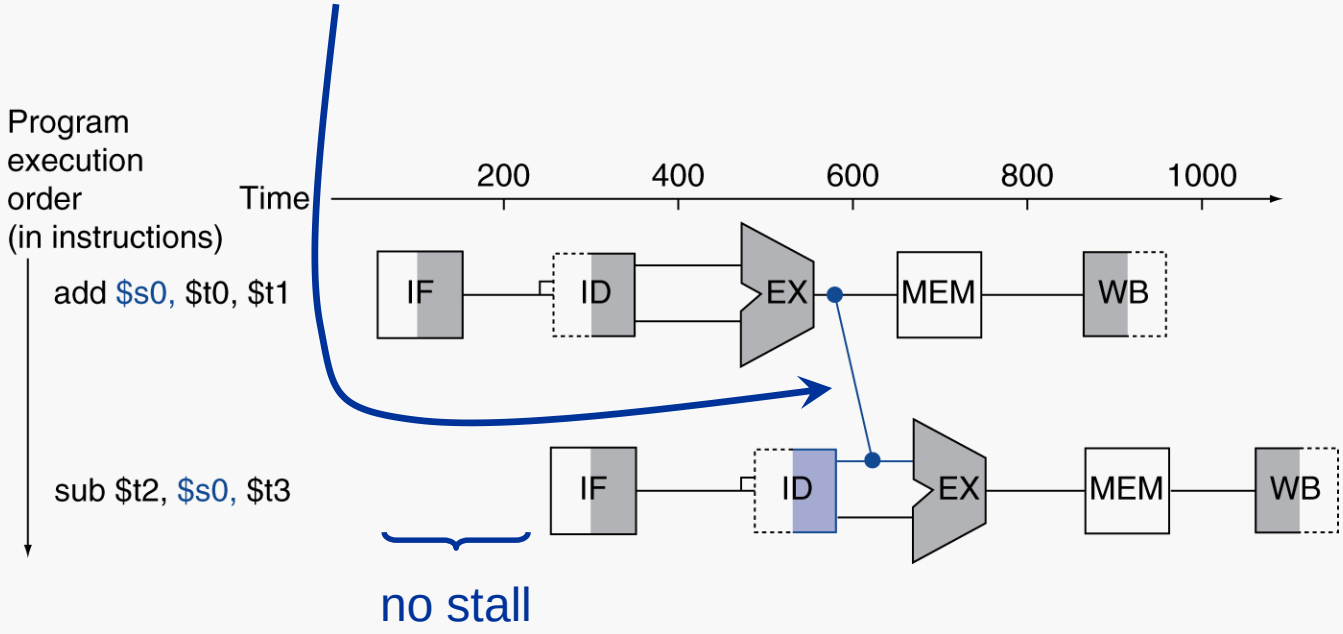
```
add    $s0, $t0, $t1   // writes $s0 from WB stage
sub    $t2, $s0, $t3   // needs value in ID stage
```



Forwarding (aka Bypassing)

Use result when it is computed

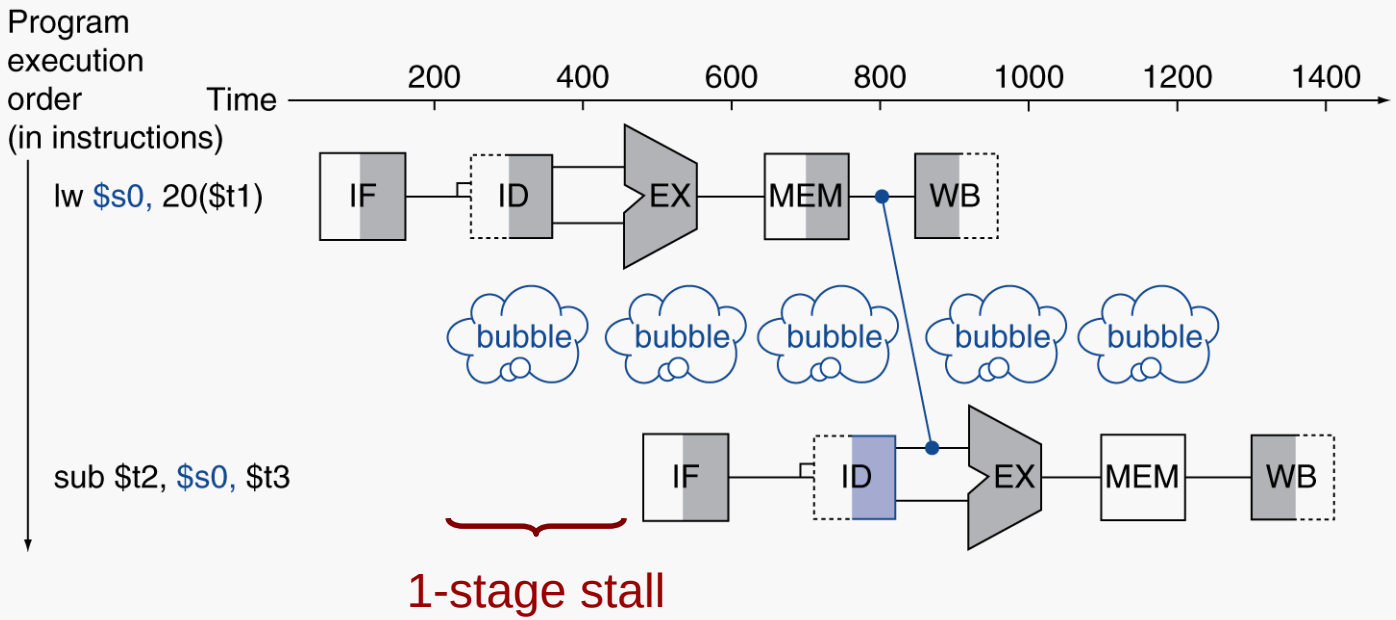
- Don't wait for it to be stored in a register
- Requires extra connections in the datapath (& more control logic?)



Load-Use Data Hazard

Can't always avoid stalls by forwarding

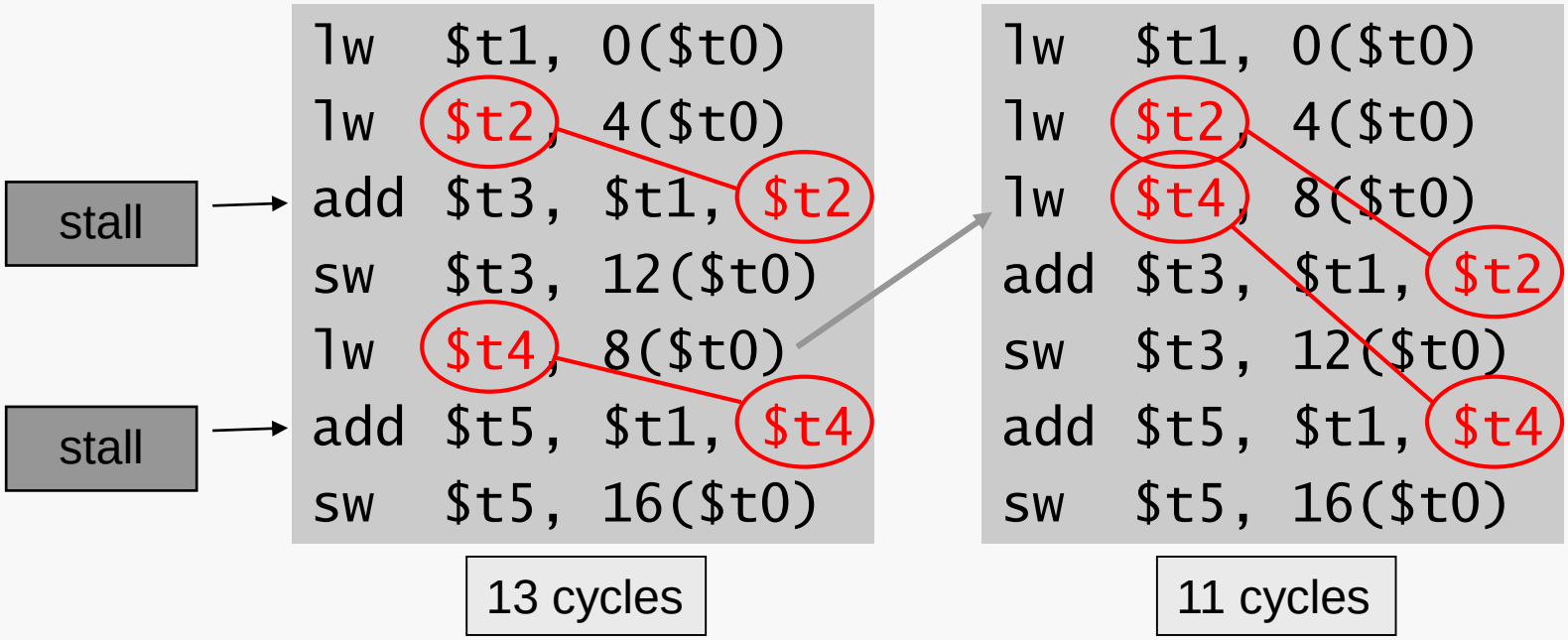
- If value not computed when needed, can't forward backward in time!



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;`



Who reorders the code?

Control Hazards: beq

Branch determines flow of control

Fetching next instruction depends on branch outcome

- Register comparison done in EX stage
- Branch target address computed in EX stage
- MUX selection done in MEM stage

What instruction do we fetch when BEQ is in ID stage?

Control Hazards: beq

How can we deal with BEQ?

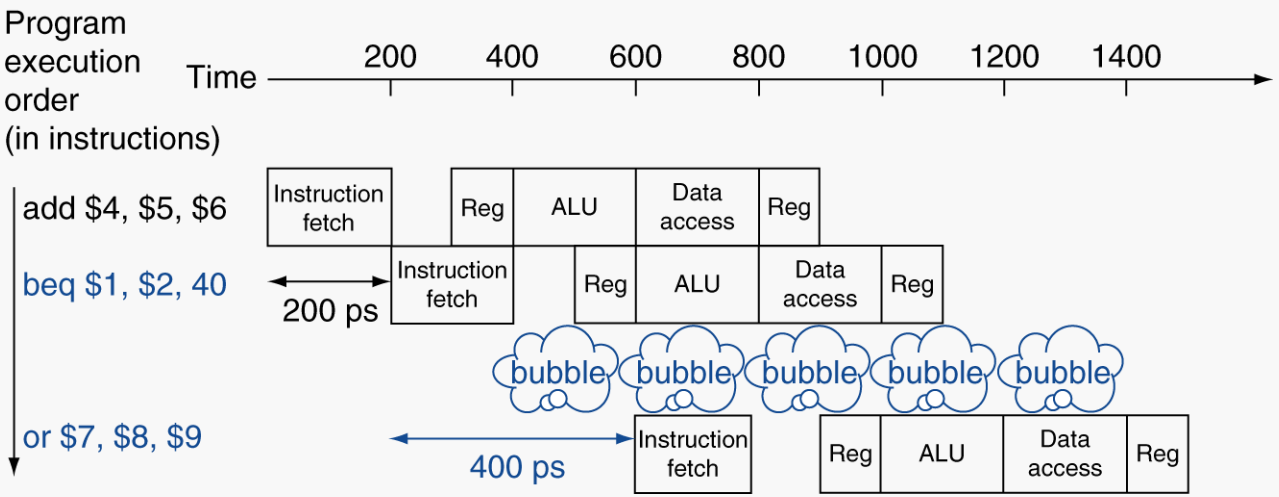
Stall until we know whether (& where) to branch?

Make the decision and calculate the address earlier?

Guess whether the branch will be taken?

Stall on Branch

Wait until branch outcome determined before fetching next instruction



How many cycles does this cost?

Early Decision

What new hardware would be needed to decide earlier?

Must compare the registers before the EX stage

Must compute the branch target address before the EX stage

Can we know what to do by the time BEQ enters the ID stage?

Branch Prediction

Longer pipelines can't readily determine branch outcome early

- Stall penalty becomes unacceptable

Predict outcome of branch

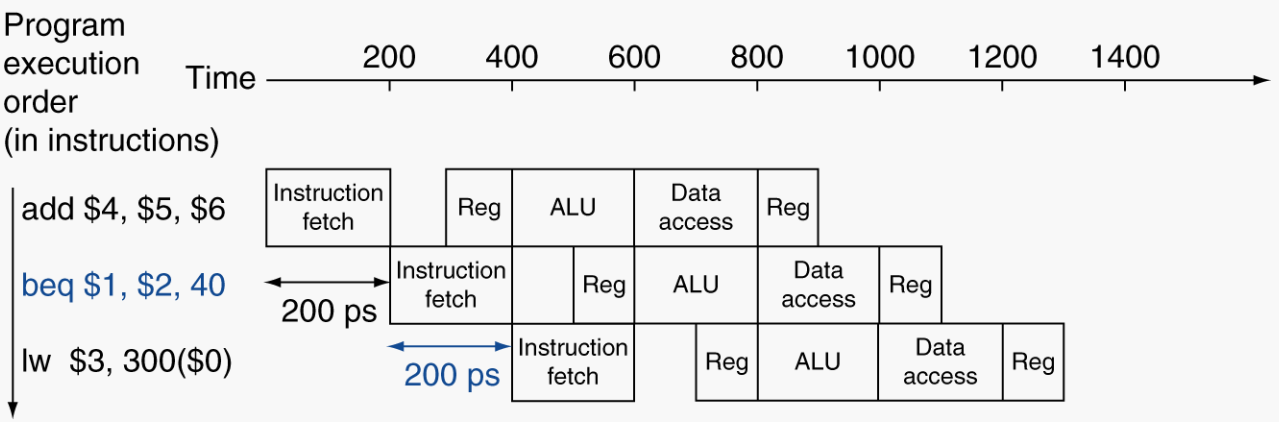
- Only stall if prediction is wrong

In MIPS pipeline

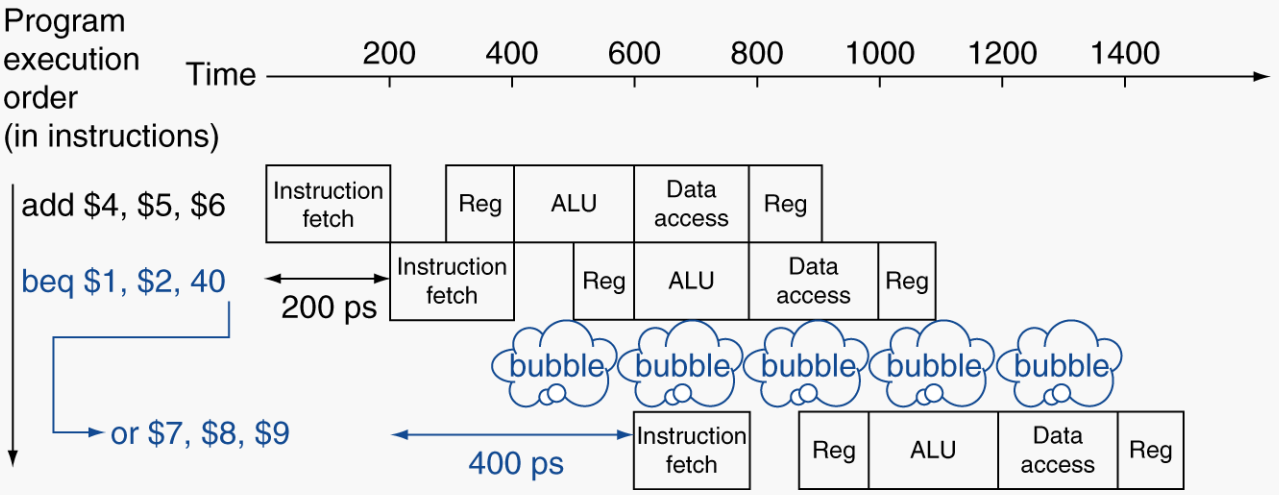
- Can predict branches will not be taken
- Fetch sequential instruction after branch, with no delay

MIPS with Predict-Not-Taken

Prediction correct



Prediction incorrect



Pipeline Summary

Pipelining improves performance by increasing instruction throughput

- Executes multiple instructions in parallel
- Each instruction has the same latency

Subject to hazards

- Structure, data, control

Instruction set design affects complexity of pipeline implementation