EE30342 – Digital Design with HDL (II)

Lecture 13

Dr. Ming Xu

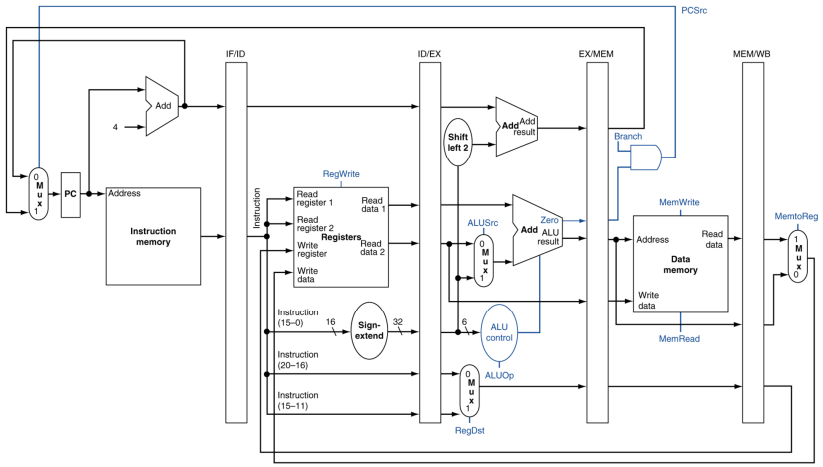
Dept of Electrical & Electronic Engineering

XJTLU

In This Session

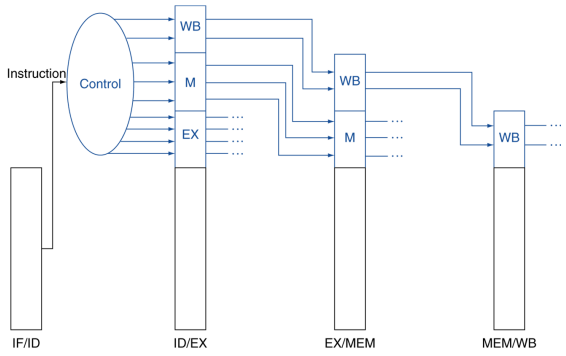
- Pipelined Control
- Pipeline Hazards

Pipelined Control (Simplified)

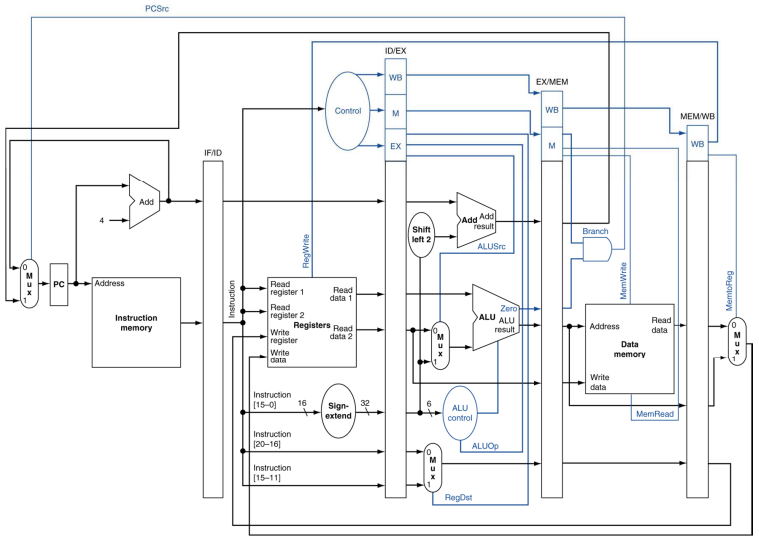


Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation



Pipelined Control



Pipeline Hazards

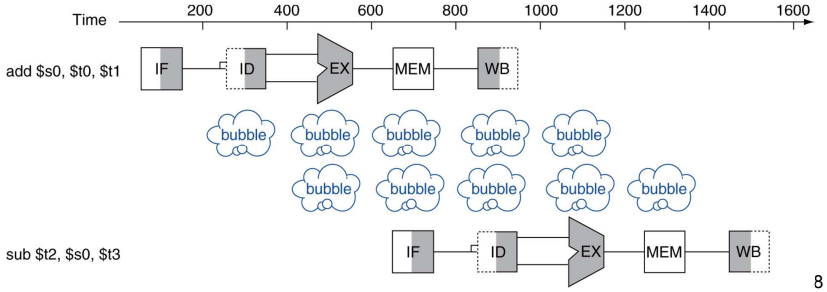
- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

Structure Hazards

- Conflict for use of a resource
- In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble”
- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches

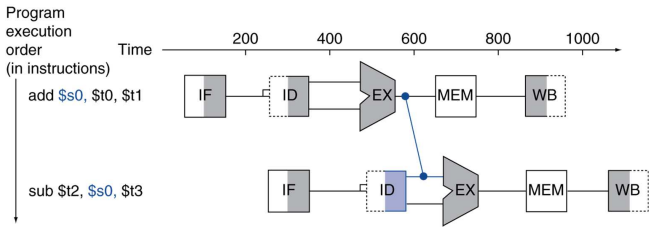
Data Hazards

- An instruction depends on completion of data access by a previous instruction
  - `add  $s0, $t0, $t1`
    `sub  $t2, $s0, $t3`



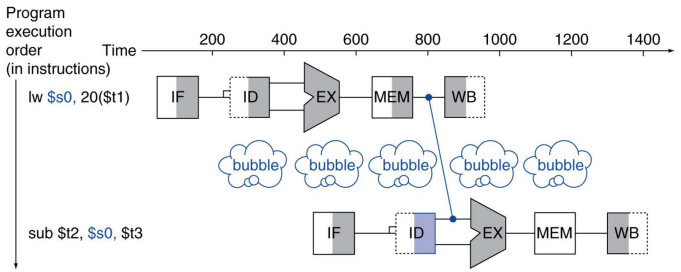
Forwarding (Bypassing)

- Use result when it is computed
  - Don't wait for it to be stored in a register
  - Requires extra connections in the datapath



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;`

```
stall  lw  $t1, 0($t0)          lw  $t1, 0($t0)
       lw  $t2, 4($t0)          lw  $t2, 4($t0)
       add $t3, $t1, $t2        lw  $t4, 8($t0)
       sw  $t3, 12($t0)         add $t3, $t1, $t2
       lw  $t4, 8($t0)          sw  $t3, 12($t0)
stall  add $t5, $t1, $t4        add $t5, $t1, $t4
       sw  $t5, 16($t0)         sw  $t5, 16($t0)
```

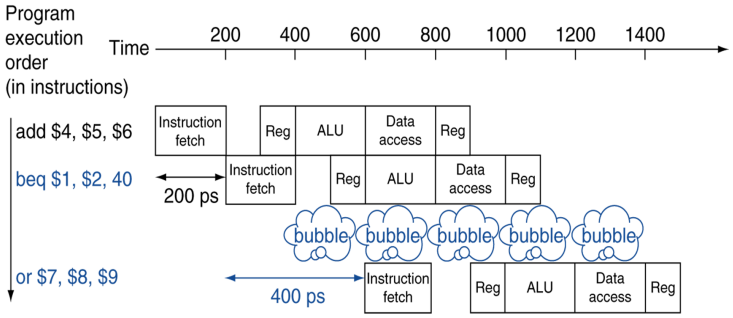
13 cycles | 11 cycles

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

Stall on Branch

- Wait until branch outcome determined before fetching next instruction

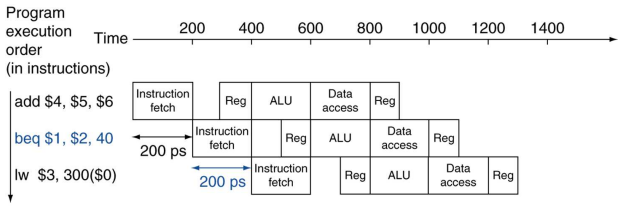


Branch Prediction

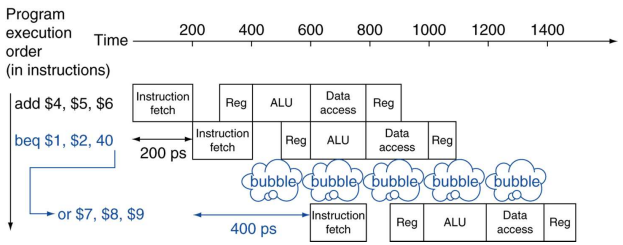
- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In MIPS pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay

MIPS with Predict Not Taken

Prediction correct



Prediction incorrect



More-Realistic Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history