CSC 411

Computer Organization (Spring 2022)
Lecture 17: Hazards, Branch Prediction

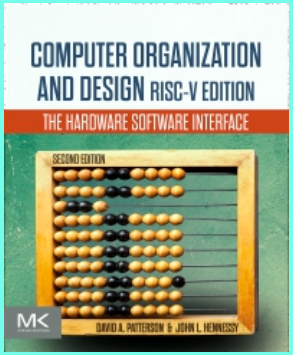
Prof. Marco Alvarez, University of Rhode Island

Disclaimer

Some of the following slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)

The Hardware/Software Interface



Hazards

Hazards

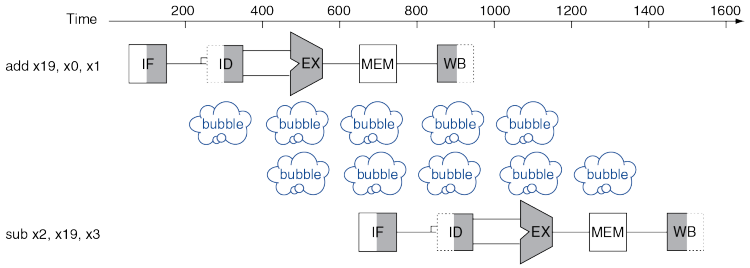
- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - a required resource is busy
- Data hazard
  - need to wait for previous instruction to complete its data read/write
- Control hazard
  - deciding on control action depends on previous instruction

Structural hazards

- Conflict for use of a resource
- In RISC-V 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

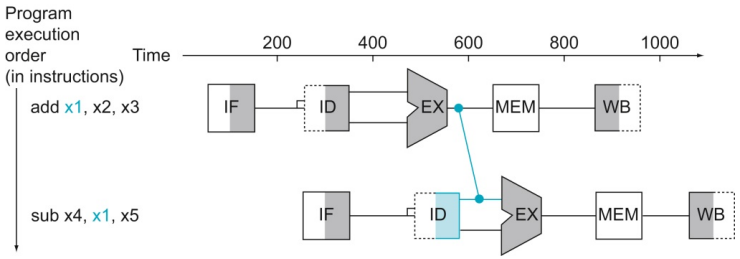
Data hazards

- An instruction depends on completion of data access by a previous instruction
- ```
  add  x19, x0, x1
  sub  x2, x19, x3
  ```



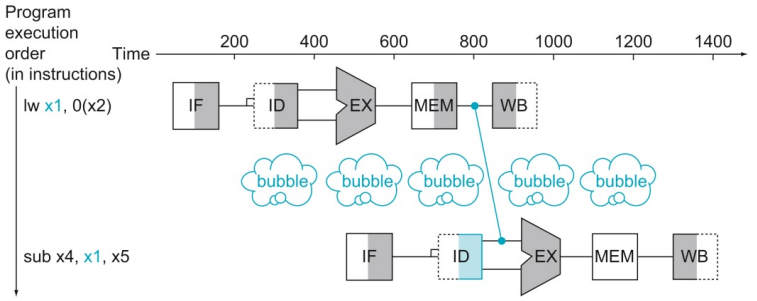
Forwarding (aka bypassing)

- Use result when it is computed
  - don’t wait for it to be stored in a register
  - requires extra connections in the datapath



Pipeline stall

- Can’t always avoid stalls by forwarding
  - if value not computed when needed


  ```

Code scheduling to avoid stalls

- Reorder code to avoid use of load result in the next instruction

```
A = B + E;
C = B + F;
```

```
  ld  x1, 0(x0)
  ld  x2, 8(x0)
★ add x3, x1, x2
  sd  x3, 24(x0)
  ld  x4, 16(x0)
★ add x5, x1, x4
  sd  x5, 32(x0)
```

13 cycles

```
ld  x1, 0(x0)
ld  x2, 8(x0)
ld  x4, 16(x0)
add x3, x1, x2
sd  x3, 24(x0)
add x5, x1, x4
sd  x5, 32(x0)
```

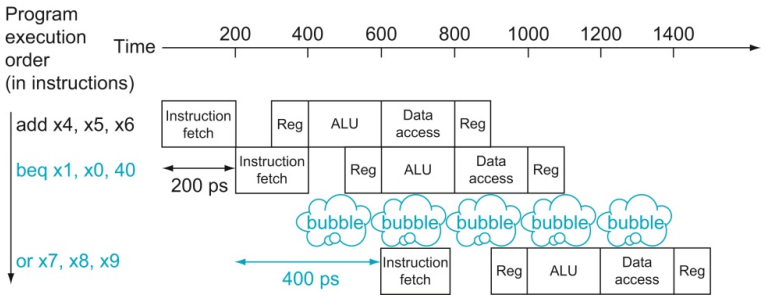
11 cycles

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 RISC-V pipeline
  - need to compare registers and compute target early in the pipeline
  - add hardware to do it in ID stage

Stall on branch

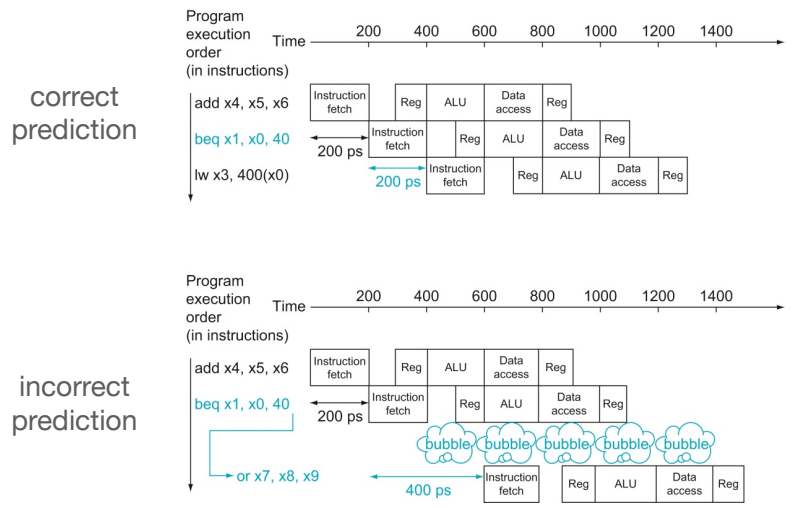
- Wait until branch outcome determined before fetching next instruction



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 RISC-V pipeline
  - can predict branches not taken
  - fetch instruction after branch, with no delay

Predicting branches are not taken



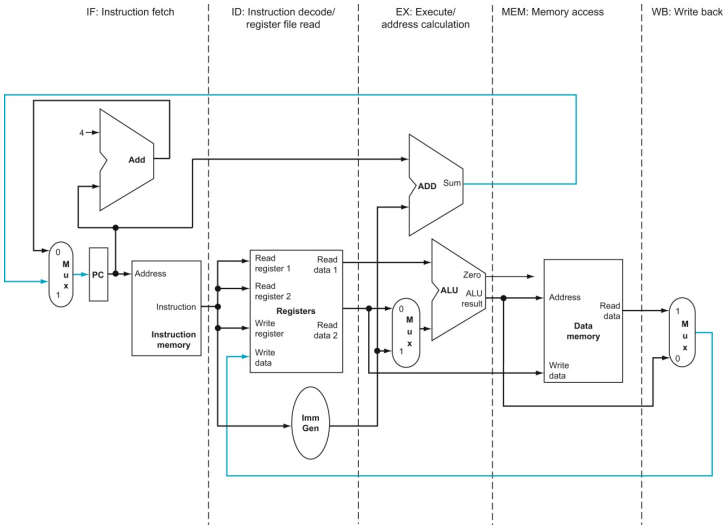
More realistic branch prediction

- Static branch prediction
  - based on typical branch behavior
  - example: **loop** and if-statement branches
    - predict backward branches taken (lower address)
    - 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

Pipeline summary

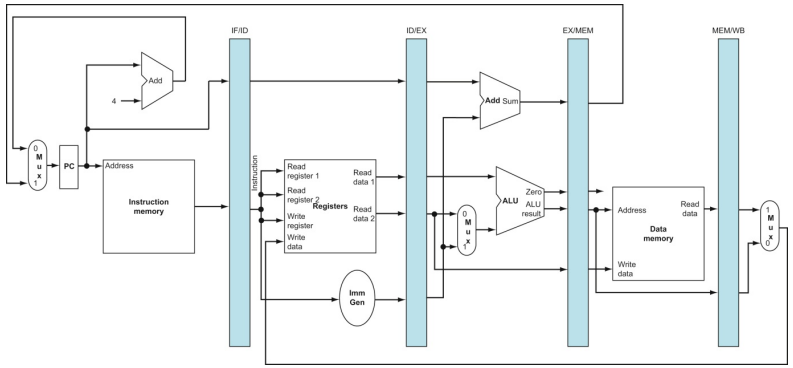
- Pipelining improves performance by increasing instruction throughput
  - executes multiple instructions in parallel
  - each instruction has the same latency
- Subject to hazards
  - structure, data, control
- Instruction set design affects complexity of pipeline implementation

RISC-V pipelined datapath

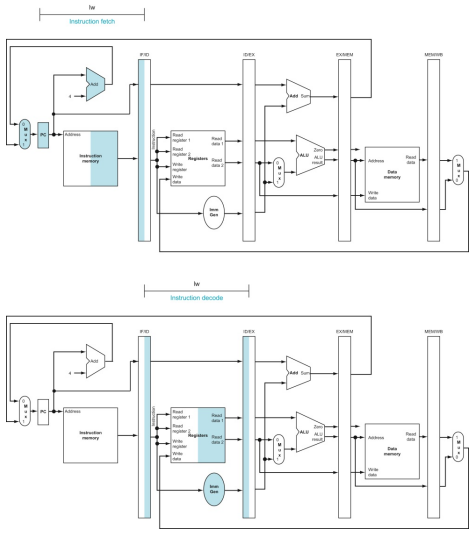


Pipeline registers

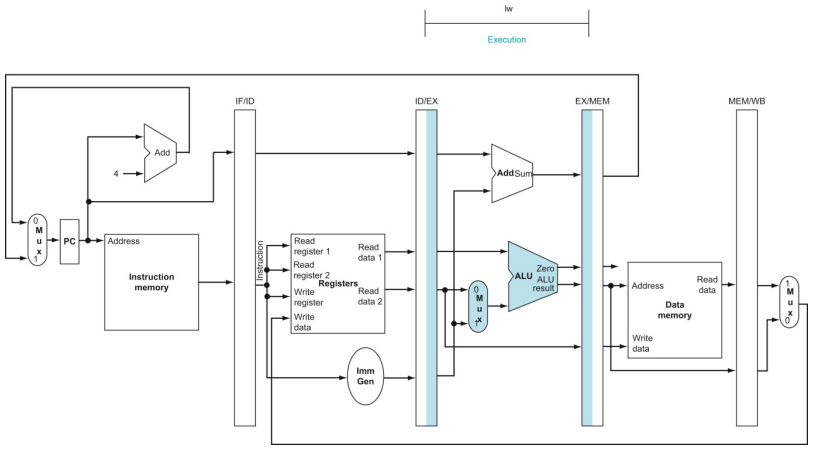
- Need registers between stages to hold information produced in previous cycle



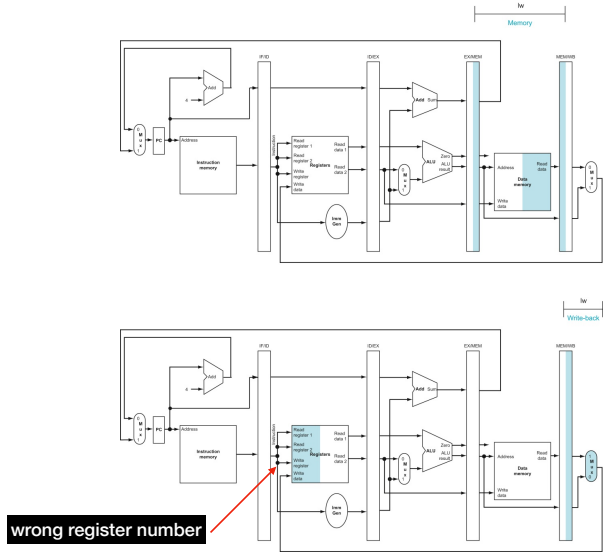
IF and ID for load, store



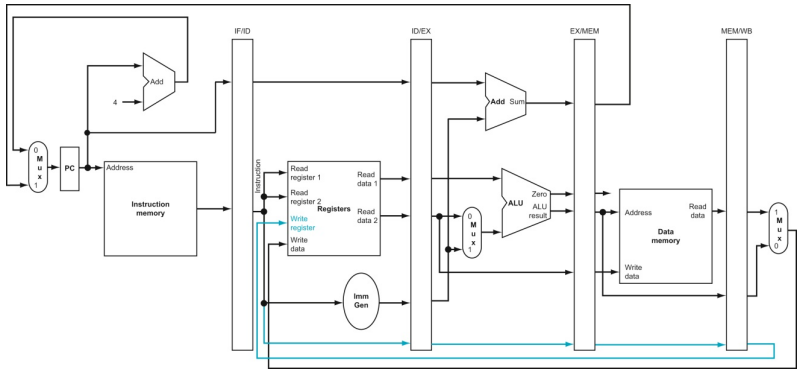
EX for load



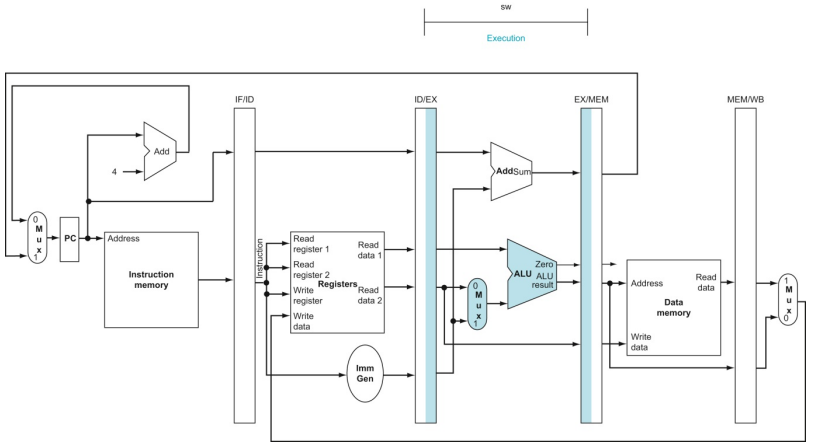
MEM and WB for load



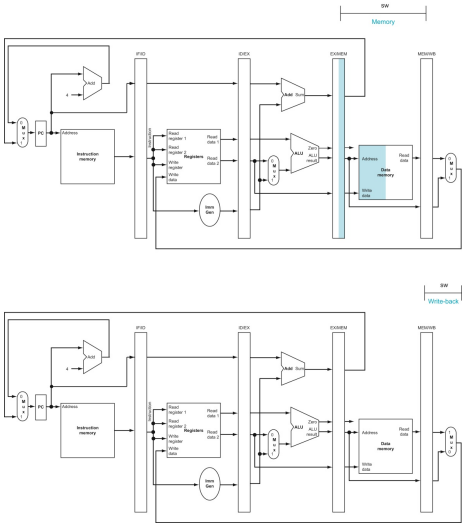
Corrected datapath



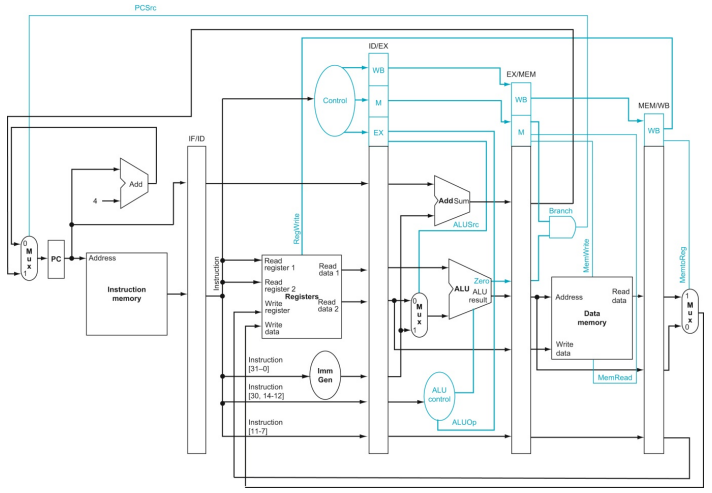
EX for store



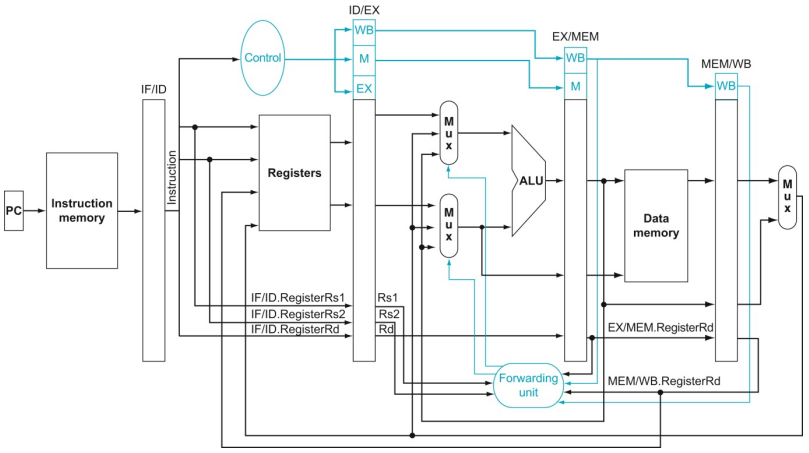
MEM and WB for store



Pipelined control



Datapath with forwarding



Datapath with hazard detection