

# CSC 411

Computer Organization (Fall 2024)

Lecture 23: Hazards and branch prediction

Prof. Marco Alvarez, University of Rhode Island

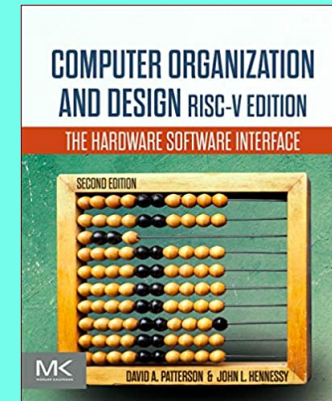
## Hazards

## Disclaimer

Some figures and slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)

The Hardware/Software Interface



## Hazards

- Pipeline hazards that prevent sequential instruction execution
  - **Structural hazards**
    - resource conflicts when multiple instructions require the same hardware component simultaneously, examples include memory ports or execution units
  - **Data hazard**
    - dependencies where an instruction requires data produced by a previous instruction that has not yet completed
  - **Control hazard**
    - occur when the instruction flow depends on a branch decision not yet resolved, pipeline must wait for branch condition evaluation before determining next instruction

## Stalls and flushes

### ▸ Pipeline stall (also called *bubble*)

- 1-cycle delay in pipeline execution where instructions are paused
- implementation:
  - holding current instruction values in pipeline registers
  - inserting “no operation” (NOP) instructions into the pipeline

### ▸ Pipeline flush

- discarding instructions from pipeline stages
  - clears all instructions, resets all control signals, and restarts instruction fetch

## Structural hazards

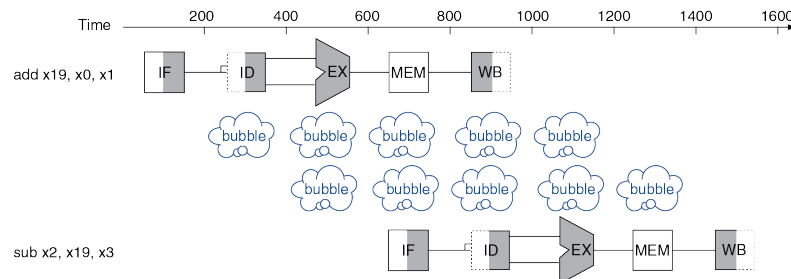
### ▸ Conflict for using a resource

### ▸ e.g., RISC-V pipeline with a single memory

- load/store requires data access
- instruction fetch would have to **stall** for that cycle, causing a pipeline “*bubble*”
- pipelined datapaths require separate instruction/data memories (separate instruction/data caches)

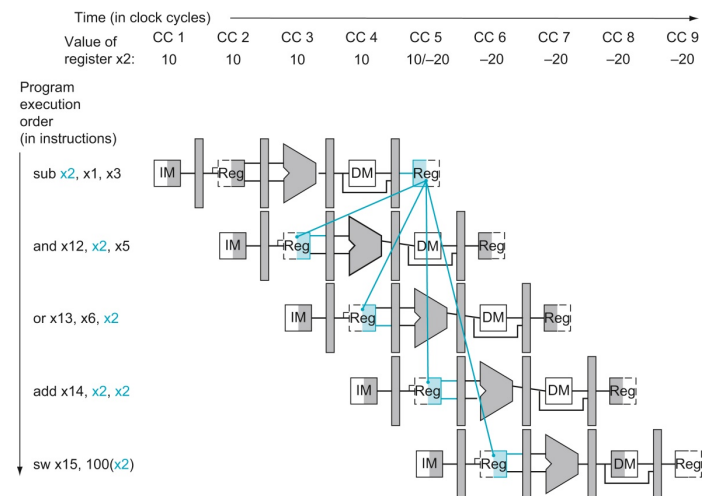
## Data hazards

### ▸ An instruction depends on completion of data access by a previous instruction



This figure is actually a simplification (bubbles/no-op instructions cause delays), section 4.7 shows the details of what really happens in the case of a hazard

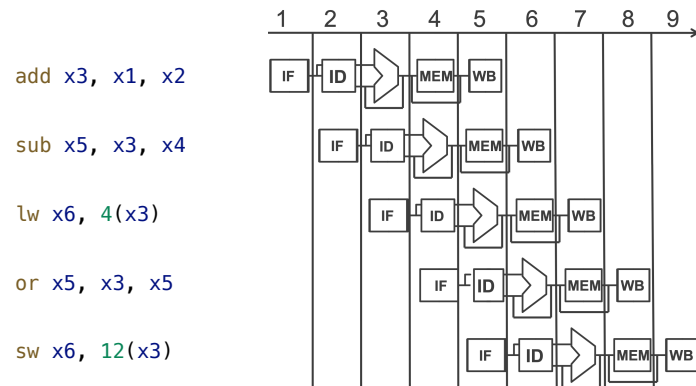
## Pipeline dependencies



## Practice

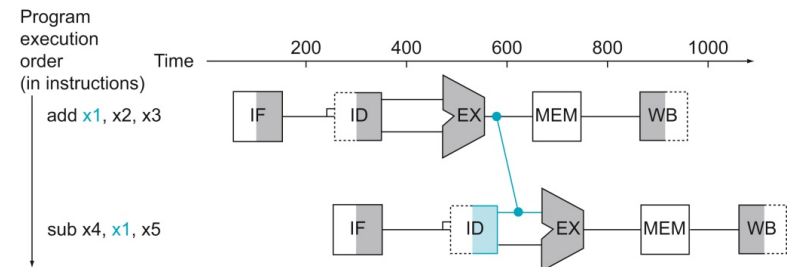
- Indicate all the data hazards?

- e.g. add → sub



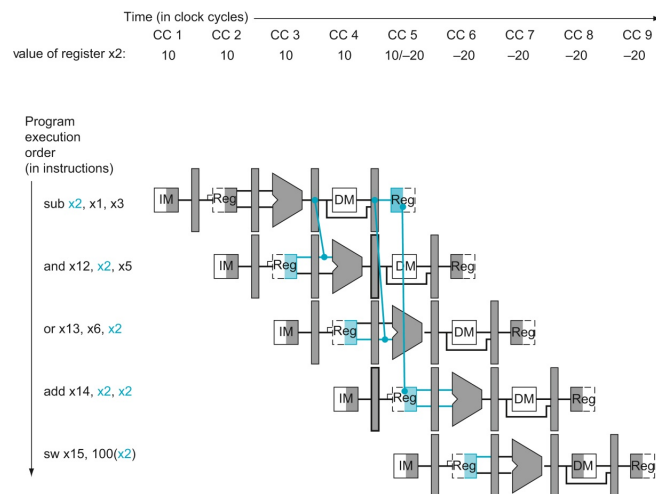
## Forwarding (a.k.a. bypassing)

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



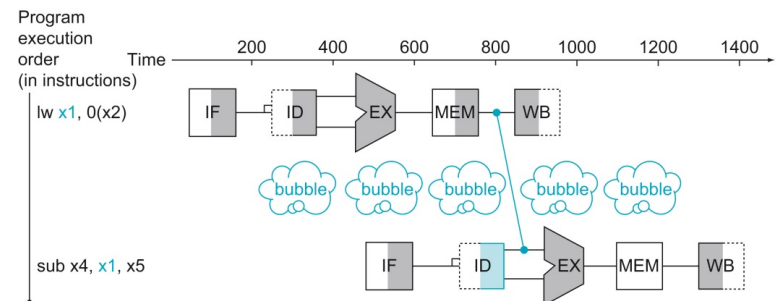
Forwarding the output of EX replaces the value from register x1 read in the second stage of sub

## Forwarding



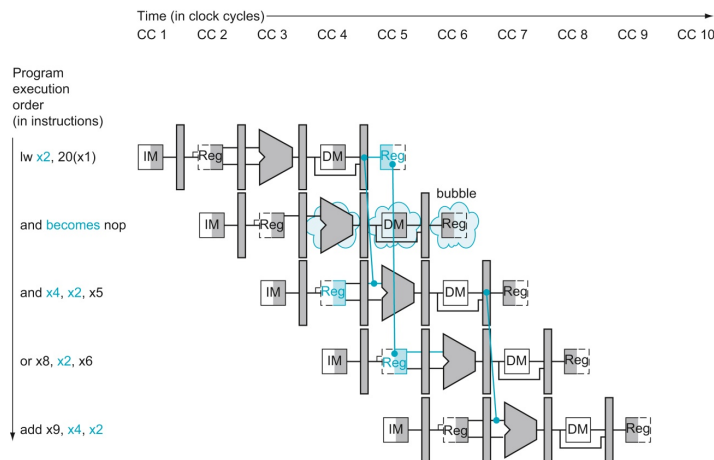
## Load-use data hazard

- Can't always avoid stalls by forwarding
  - if value not computed when needed, can't forward "backward" in time



We need a stall even with forwarding when an R-format instruction following a load tries to use the data (this figure is actually a simplification)

## Inserting stalls



## Code scheduling to avoid stalls

### Reorder instructions

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

lw x1, 0(x0)  
lw x2, 8(x0)  
add x3, x1, x2  
sw x3, 24(x0)  
lw x4, 16(x0)  
add x5, x1, x4  
sw x5, 32(x0)

Number of cycles?

lw x1, 0(x0)  
lw x2, 8(x0)  
lw x4, 16(x0)  
add x3, x1, x2  
sw x3, 24(x0)  
add x5, x1, x4  
sw x5, 32(x0)

Number of cycles?

## Control hazards

### Branch instructions impact control flow

- fetching next instruction depends on branch outcome
- pipeline can't always fetch correct instruction
  - still working on ID stage of branch

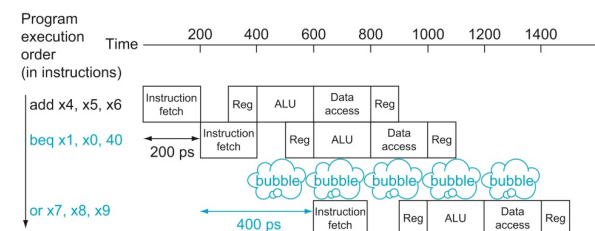
### In RISC-V pipeline

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

## Stall on branch

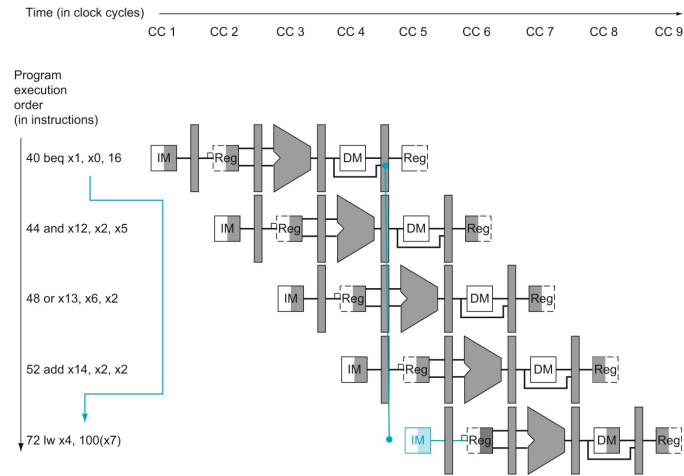
### Wait until branch outcome determined before fetching next instruction

- one cycle stall if branch condition is determined at ID
- two cycles stall if branch condition is determined at EXE



Pipeline showing stalling on every conditional branch as solution to control hazards. This example assumes the conditional branch is taken, and the instruction at the destination of the branch is the or instruction.

## Impact of the pipeline on branch instructions



## Branch prediction

## Branch prediction

### • Impact of pipeline depth

- modern deep pipelines increase branch resolution latency
- branch outcome determined several cycles after fetch
- stalling until resolution significantly impacts performance

### • Branch prediction strategies

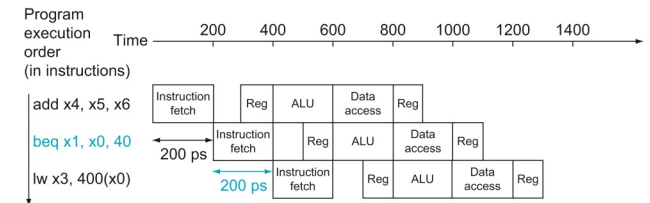
- predict branch outcome before resolution
- continue fetching along predicted path
- penalty occurs only on misprediction
  - pipeline flush required
  - restart fetch from correct path

### • RISC-V pipeline branch handling

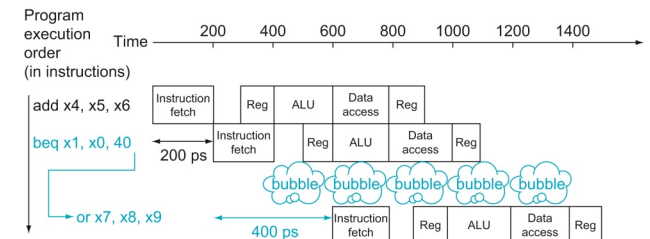
- static prediction assumes branches are not taken (fetch instruction after branch with no delay)
  - RISC-V ISA supports both static and dynamic prediction
- hardware includes branch prediction buffers, branch target buffers, and branch history tables
- need recovery mechanisms for misprediction

## Predicting branches are not taken

correct prediction



incorrect prediction



## More realistic branch prediction

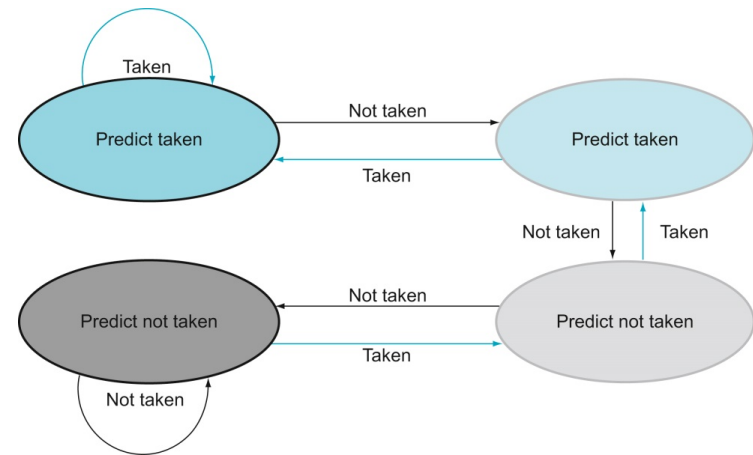
### ▸ Static branch prediction

- 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

## 2-bit prediction



The 2 bits are used to encode the four states in the system

## Pipeline summary

### ▸ Pipelining improves performance by increasing instruction throughput

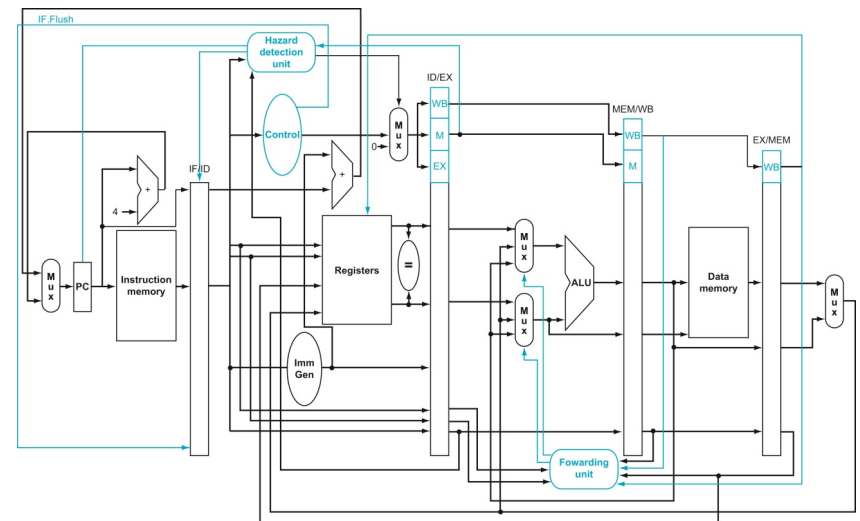
- executes multiple instructions in parallel
- each instruction has the same latency

### ▸ Subject to hazards

- structural, data, control

### ▸ Instruction set design affects complexity of pipeline implementation

## Final datapath



# Advanced techniques

## Instruction-level parallelism (ILP)

### • Basic pipelining

- multiple instructions execute simultaneously in different stages
- ideal CPI approaches 1
- limited by pipeline hazards

### • ILP techniques

- deeper pipeline
  - divide execution into more stages
  - reduced work per stage → higher clock frequencies
  - trade-off: increased hazard penalties
- multiple issue
  - multiple instructions issued per clock cycle
  - requires replicated functional units
  - performance, e.g., 4GHz 4-way multiple-issue
    - 16 BIPS, peak CPI = 0.25, peak IPC = 4
    - dependencies/hazards reduce this in practice

## Multiple issue

### • Static multiple issue

- compiler groups instructions to be issued together
- packages them into “issue slots”
- compiler detects and avoids hazards

### • Dynamic multiple issue

- CPU examines instruction stream and chooses instructions to issue each cycle
- compiler can help by reordering instructions
- CPU resolves hazards using advanced techniques at runtime

## Speculation

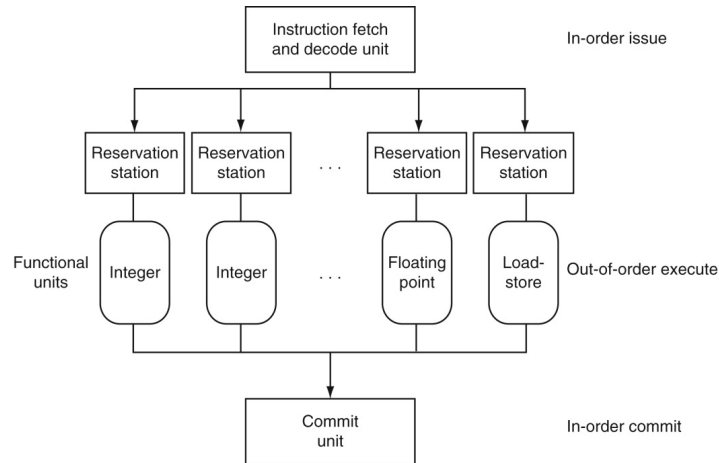
### • “Guess” what to do with an instruction

- start operation as soon as possible
- check whether guess was right
  - if so, complete the operation
  - if not, roll-back and do the right thing

### • Common to static and dynamic multiple issue

- examples
  - speculate on **branch outcome**
    - roll-back if path taken is different
  - speculate on **load**
    - roll-back if location is updated
- predict and continue issuing, commit after outcome determined
  - use buffers

# Dynamic pipeline scheduling

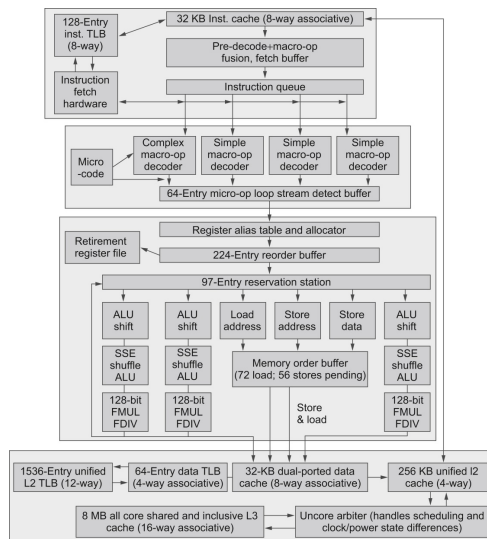


The three primary units of a dynamically scheduled pipeline. The final step of updating the state is also called retirement or graduation.

# Intel microprocessors

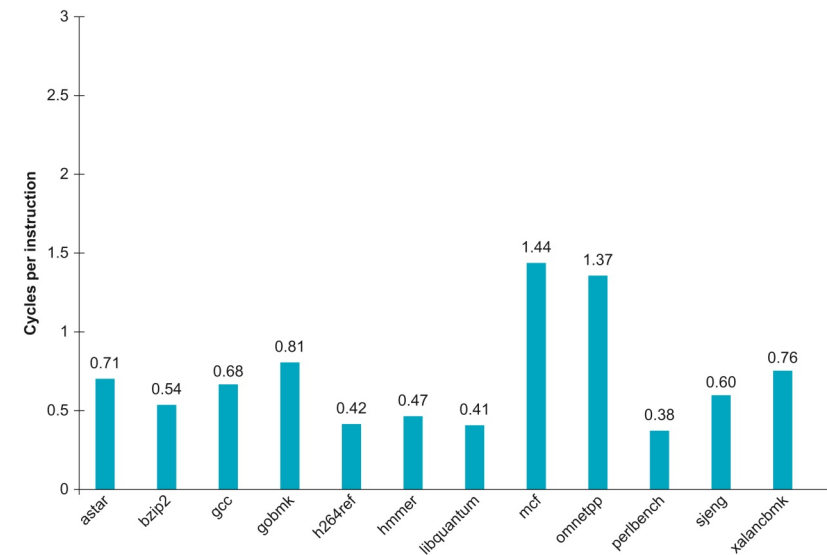
Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/Speculation	Cores/Chip	Power
Intel 486	1989	25 MHz	5	1	No	1	5W
Intel Pentium	1993	66 MHz	5	2	No	1	10W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103W
Intel Core	2006	3000 MHz	14	4	Yes	2	75W
Intel Core i7 Nehalem	2008	3600 MHz	14	4	Yes	2-4	87W
Intel Core Westmere	2010	3730 MHz	14	4	Yes	6	130W
Intel Core i7 Ivy Bridge	2012	3400 MHz	14	4	Yes	6	130W
Intel Core Broadwell	2014	3700 MHz	14	4	Yes	10	140W
Intel Core i9 Skylake	2016	3100 MHz	14	4	Yes	14	165W
Intel Ice Lake	2018	4200 MHz	14	4	Yes	16	185W

# Intel Core i7 pipeline



The total pipeline depth is 14 stages, with branch mispredictions typically costing 17 cycles. This design can buffer 72 loads and 56 stores. The six independent functional units (6-wide superscalar) can each begin execution of a ready micro-operation in the same cycle.

# CPI for the i7 6700 (SPEC CPUint2006)





## Misprediction rate for the i7 6700 (SPEC CPUint2006)

