Basic Pipelined Processor

Hung-Wei Tseng

Outline

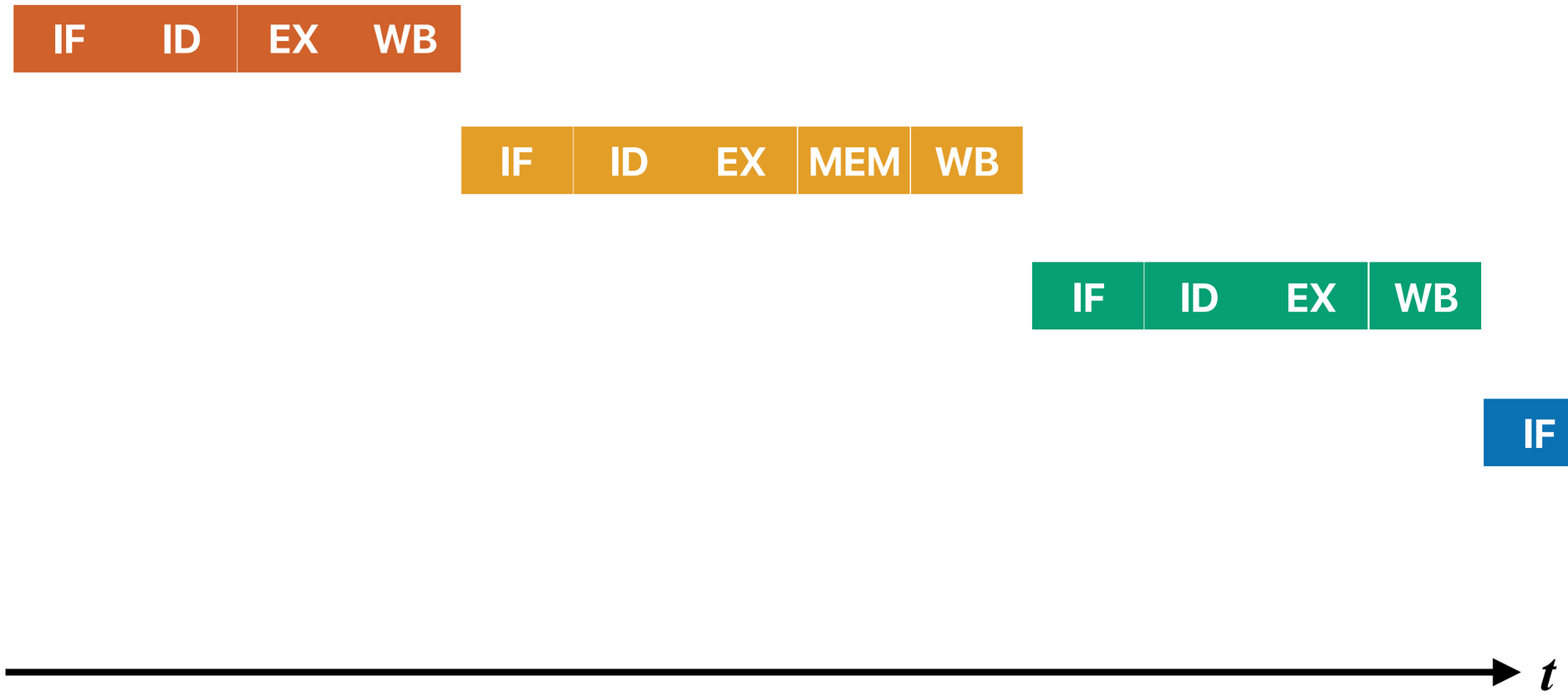
- Pipelining
- Pipeline Hazards
- Structural Hazards
- Control Hazards
- Dynamic Branch Predictions

Tasks in RISC-V ISA

- Instruction Fetch (**IF**) — fetch the instruction from memory
- Instruction Decode (**ID**)
  - Decode the instruction for the desired operation and operands
  - Reading source register values
- Execution (**EX**)
  - ALU instructions: Perform ALU operations
  - Conditional Branch: Determine the branch outcome (taken/not taken)
  - Memory instructions: Determine the effective address for data memory access
- Data Memory Access (**MEM**) — Read/write memory
- Write Back (**WB**) — Present ALU result/read value in the target register
- Update PC
  - If the branch is taken — set to the branch target address
  - Otherwise — advance to the next instruction — current PC + 4

Simple implementation w/o branch

```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9,x10,x11
sd  x1, 0(x12)
```



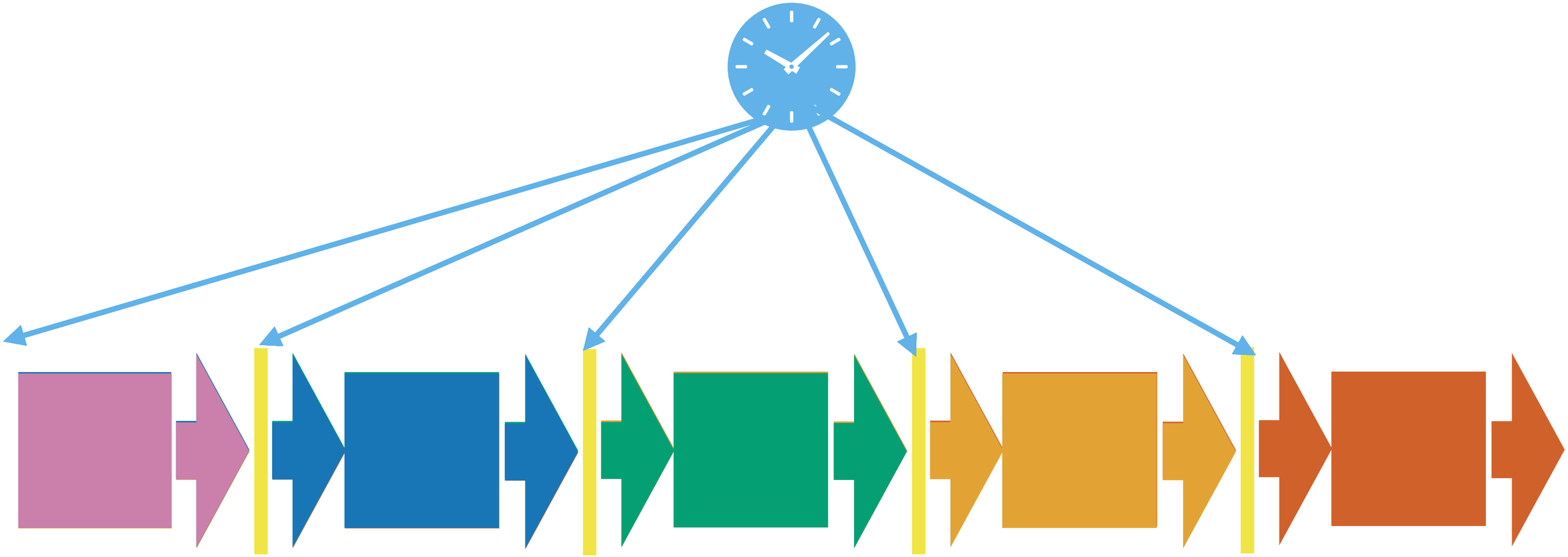
Pipelining

Pipelining

Pipelining

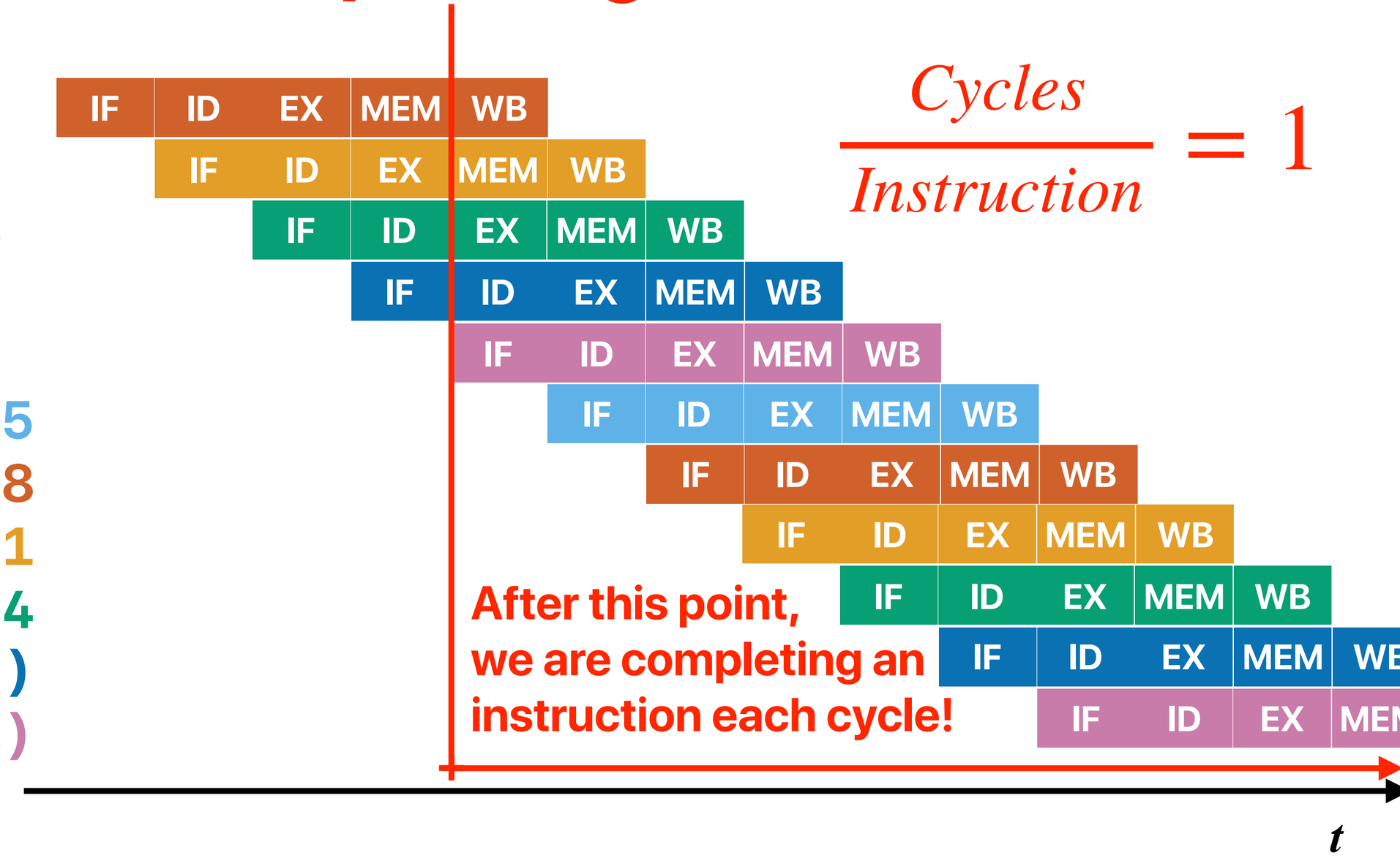
- Different parts of the processor works on different instructions simultaneously
- A clock signal controls and synchronize the beginning and the end of each part of the work
- A pipeline register between different parts of the processor to keep intermediate results necessary for the upcoming work

Pipelining



Pipelining

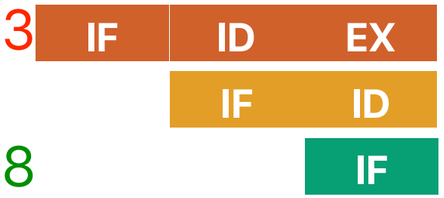
```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9,x10,x11
sd  x1, 0(x12)
xor x13,x14,x15
and x16,x17,x18
add x19,x20,x21
sub x22,x23,x24
ld  x25, 4(x26)
sd  x27, 0(x28)
```



$$\frac{Cycles}{Instruction} = 1$$

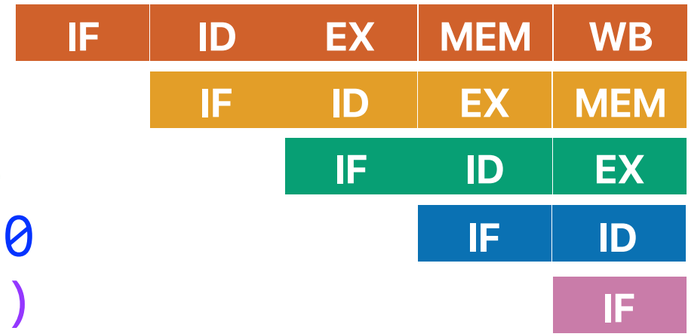
Draw the pipeline diagrams

```
add x1, x2, x3
ld  x4, 0(x1)
sub x6, x7, x8
sub x9,x10,x11
sd  x1, 0(x12)
```



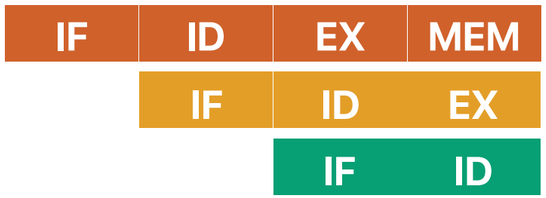
The desired value of x1 is not ready yet

```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9, x1, x10
sd  x11, 0(x12)
```

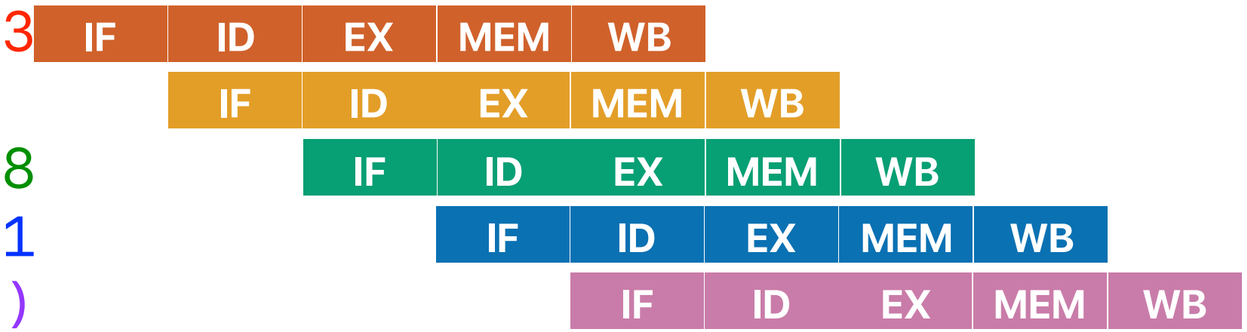


Both instructions need x1

```
add x1, x2, x3
ld  x4, 0(x5)
bne x0, x7, L
sub x9,x10,x11
sd  x1, 0(x12)
```



Doesn't know what to fetch at this moment



Can we get them right?

- Given a simple pipelined RISC-V processor that we discussed so far, how many of the following code snippets can be executed with expected outcome?

	I	II	III	IV
a	`add x1, x2, x3`	`add x1, x2, x3`	`add x1, x2, x3`	`add x1, x2, x3`
b	`ld x4, 0(x1)`	`ld x4, 0(x5)`	`ld x4, 0(x5)`	`ld x4, 0(x5)`
c	`sub x6, x7, x8`	`sub x6, x7, x8`	`bne x0, x7, L`	`sub x6, x7, x8`
d	`sub x9,x10,x11`	`sub x9, x1, x10`	`sub x9,x10,x11`	`sub x9,x10,x11`
e	`sd x1, 0(x12)`	`sd x11, 0(x12)`	`sd x1, 0(x12)`	`sd x1, 0(x12)`

A. 0

B. 1

C. 2

D. 3

E. 4

b cannot get x1 produced by a before WB

both a and d are accessing x1 at the 5th cycle

We don't know if d & e will be executed or not until c finishes

Pipeline hazards

Three pipeline hazards

- Structural hazards — resource conflicts cannot support simultaneous execution of instructions in the pipeline
- Control hazards — the PC can be changed by an instruction in the pipeline
- Data hazards — an instruction depending on a the result that's not yet generated or propagated when the instruction needs that

Can we get them right?

- Given a simple pipelined RISC-V processor that we discussed so far, how many of the following code snippets can be executed with expected outcome?

	I	II	III	IV
a	`add x1, x2, x3`	`add x1, x2, x3`	`add x1, x2, x3`	`add x1, x2, x3`
b	`ld x4, 0(x1)`	`ld x4, 0(x5)`	`ld x4, 0(x5)`	`ld x4, 0(x5)`
c	`sub x6, x7, x8`	`sub x6, x7, x8`	`bne x0, x7, L`	`sub x6, x7, x8`
d	`sub x9,x10,x11`	`sub x9, x1, x10`	`sub x9,x10,x11`	`sub x9,x10,x11`
e	`sd x1, 0(x12)`	`sd x11, 0(x12)`	`sd x1, 0(x12)`	`sd x1, 0(x12)`

A. 0

B. 1

C. 2

D. 3

E. 4

b cannot get x1 produced by a before WB

Data Hazard

both a and d are accessing x1 at the 5th cycle

Structural Hazard

We don't know if d & e will be executed or not until c finishes

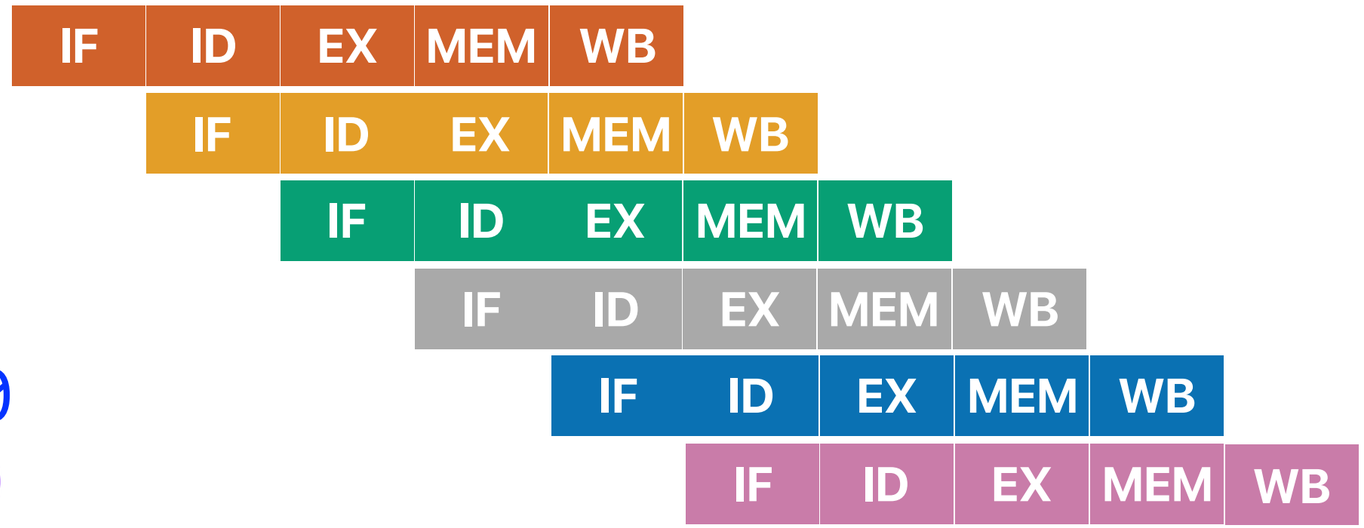
Control Hazard

Structural Hazards

Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Solution: insert no-ops (e.g, `add x0,x0,x0`) between them
- Drawback
  - If the number of pipeline stages changes, the code won't work
  - Slow

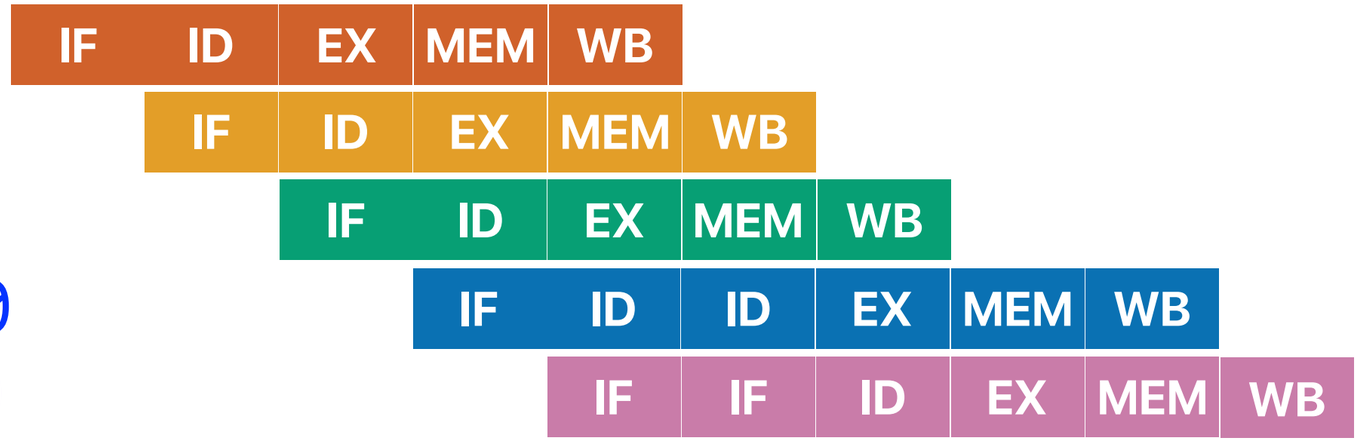
```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
add x0, x0, x0
sub x9, x1, x10
sd  x11, 0(x12)
```



Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Solution: stall the later instruction, allowing the write to present the change in the register and the later can get the desired value
- Drawback: slow

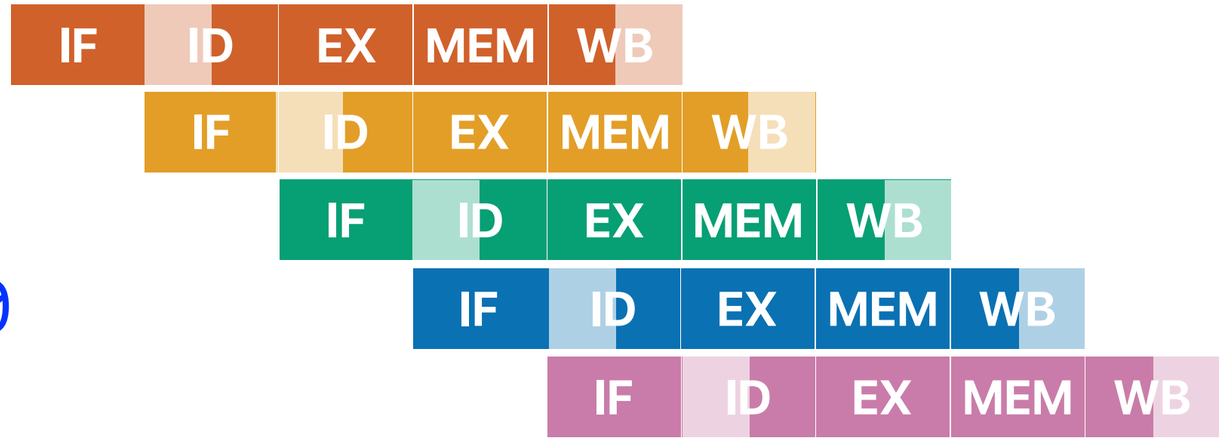
```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9, x1, x10
sd  x11, 0(x12)
```



Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Better solution: write early, read late
  - Writes occur at the clock edge and complete long enough before the end of the clock cycle.
  - This leaves enough time for outputs to settle for reads
  - The revised register file is the default one from now!

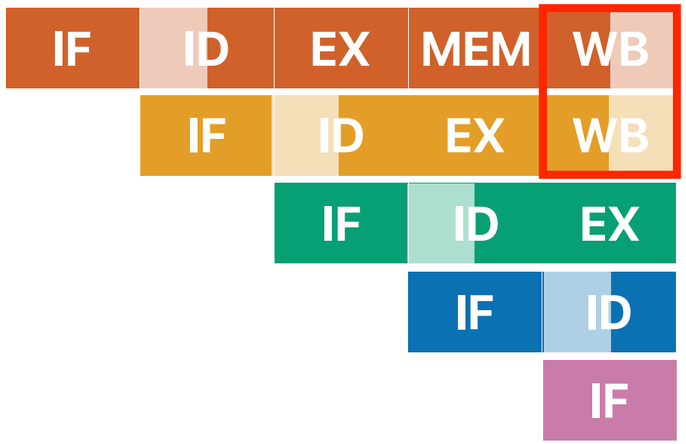
```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9, x1, x10
sd  x11, 0(x12)
```



Structural Hazards

- What pair of instructions will be problematic if we allow ALU instructions to skip the "MEM" stage?

```
a: ld  x1, 0(x2)
b: add x3, x4, x5
c: sub x6, x7, x8
d: sub x9,x10,x11
e: sd  x1, 0(x12)
```



A. a & b

B. a & c

C. b & e

D. c & e

E. None

Structural Hazards

- Stall can address the issue — but slow
- Improve the pipeline unit design to allow parallel execution

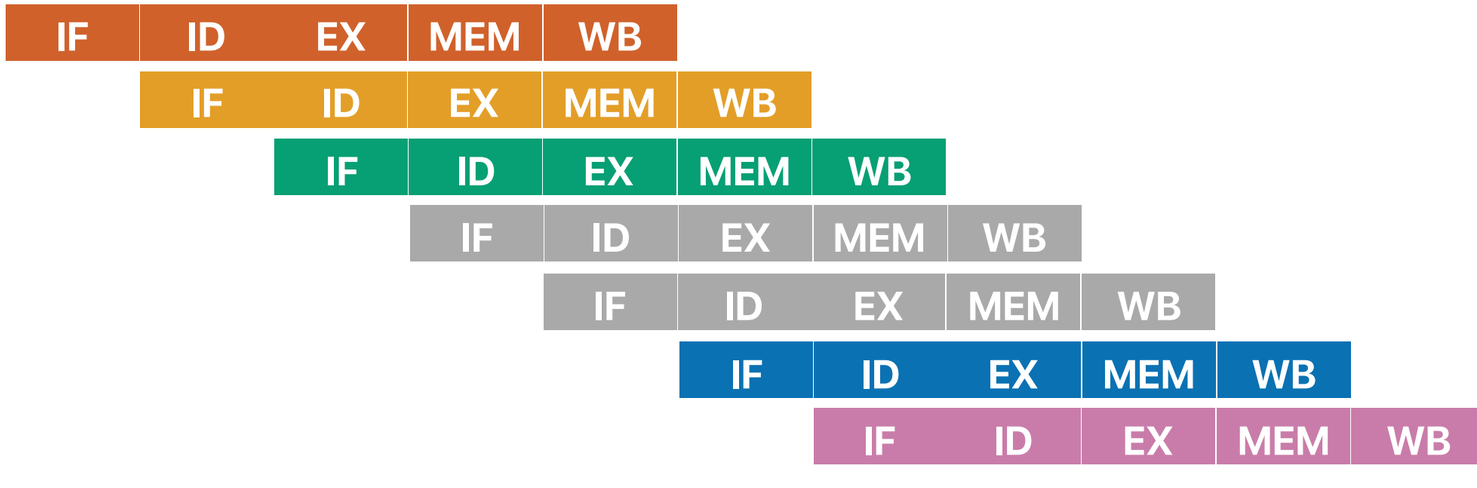
Control Hazards

The impact of control hazards

- Assuming that we have an application with 20% of branch instructions and the instruction stream incurs no data hazards. When there is a branch, we disable the instruction fetch and insert no-ops until we can determine the PC. What's the average CPI if we execute this program on the 5-stage RISC-V pipeline?

A. 1
B. 1.2
C. 1.4
D. 1.6
E. 1.8

```
add x1, x2, x3
ld  x4, 0(x5)
bne x0, x7, L
add x0, x0, x0
add x0, x0, x0
sub x9,x10,x11
sd  x1, 0(x12)
```



$$1 + 20\% \times 2 = 1.4$$

Why can’t we proceed without stalls/no-ops?

- How many of the following statements are true regarding why we have to stall for each branch in the current pipeline processor
  - ① The target address when branch is taken is not available for instruction fetch stage of the next cycle
  - ② The target address when branch is not-taken is not available for instruction fetch stage of the next cycle
  - ③ The branch outcome cannot be decided until the comparison result of ALU is not out
  - ④ The next instruction needs the branch instruction to write back its result

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Dynamic Branch Prediction

Why can’t we proceed without stalls/no-ops?

- How many of the following statements are true regarding why we have to stall for each branch in the current pipeline processor
  - ① The target address when branch is taken is not available for instruction fetch stage of the next cycle
  - ② The target address when branch is not-taken is not available for instruction fetch stage of the next cycle
  - ③ The branch outcome cannot be decided until the comparison result of ALU is not out
  - ④ The next instruction needs the branch instruction to write back its result

You need a cheatsheet for that — branch target buffer

You need to predict that — history/states

A. 0
B. 1
C. 2
D. 3
E. 4

Announcement

- Pickup midterm
  - Outside of Hung-Wei's office from 1p today
  - If you would like to regrade
    - Write a report on why should we.
    - We will regrade the whole test — might lower your grades
- Reading quiz due next Wednesday
- No class next Monday — Veterans' Day Holiday!
- Project
  - Watch carefully on piazza/iLearn/website, should be announced by this Friday