EECS151/251A
Introduction to Digital Design and ICs

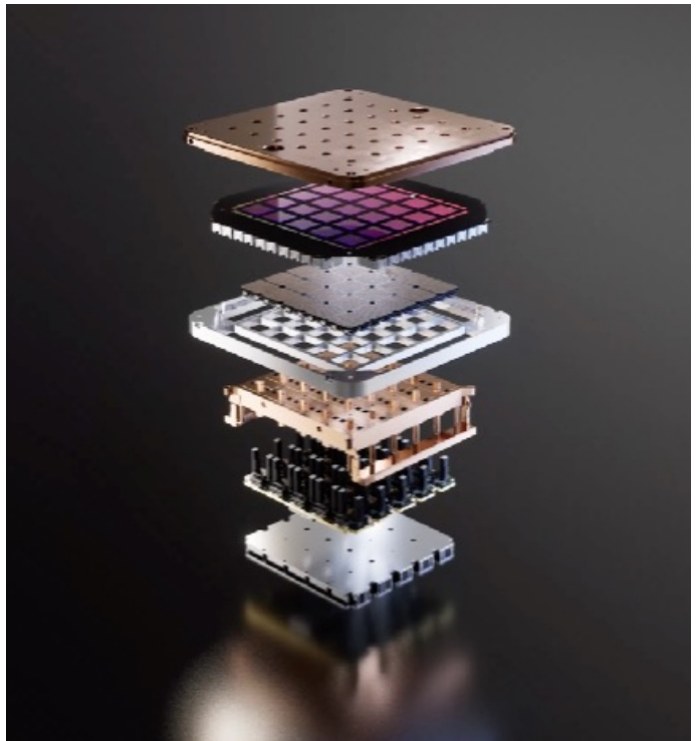
Lecture 10: RISC-V Pipelining

Sophia Shao

Tesla's Dojo

To say Tesla is merely interested in machine learning is an understatement. The electric car maker built an in-house supercomputer named Dojo, optimized for training its machine learning models. Unlike many other supercomputers, Dojo isn't using off-the-shelf CPUs and GPUs, such as from AMD, Intel, or Nvidia. Instead, Tesla designed their own microarchitecture tailored to their needs, letting them make tradeoffs that more general architectures can't make. In this article, we're going to take a look at that architecture, based on Tesla's presentations at Hot Chips. The architecture doesn't have a separate name, so for simplicity, whenever we mention Dojo further down we're talking about the architecture.

https://chipsandcheese.com/2022/09/01/hot-chips-34-teslas-dojo-microarchitecture/



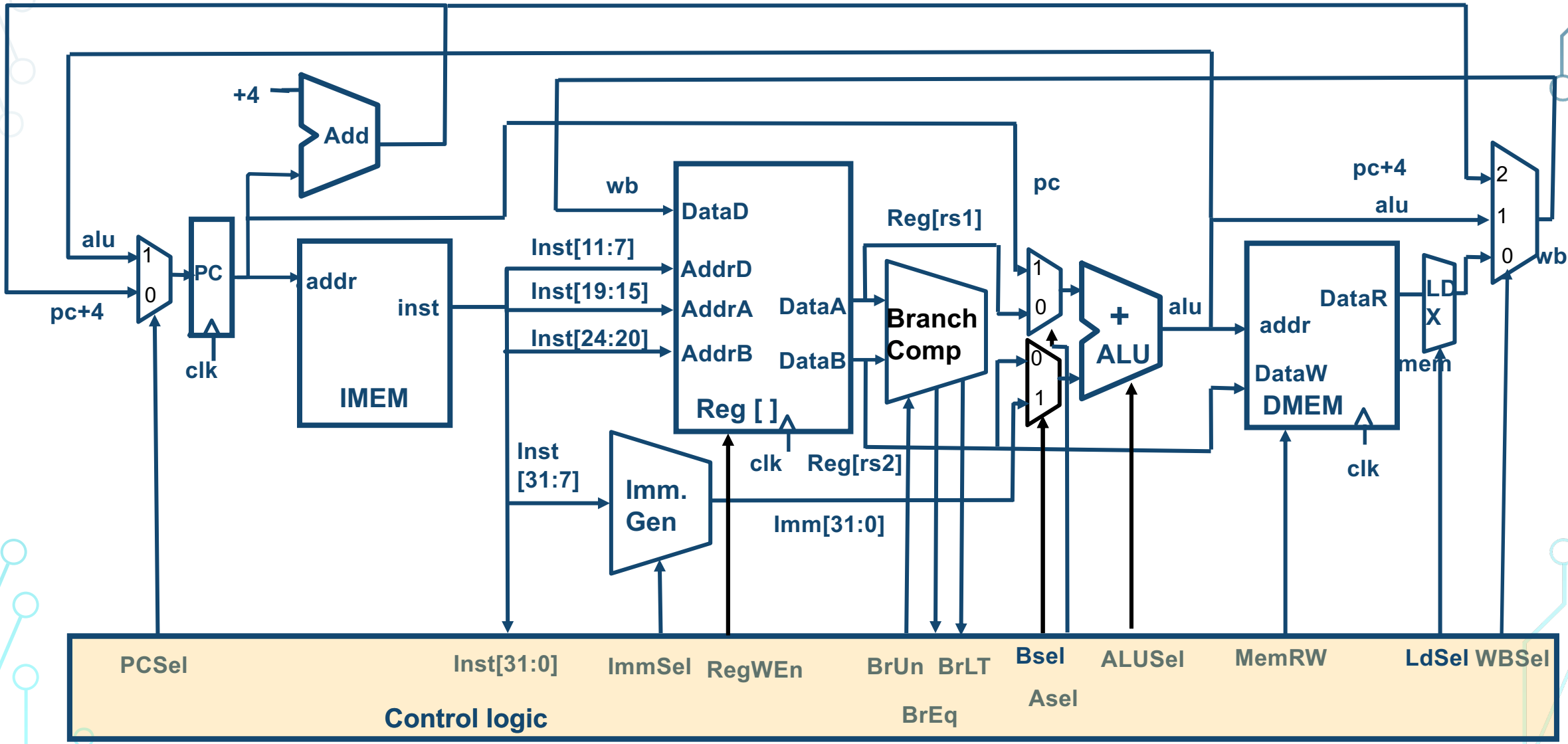
Review

- We have covered the implementation of the base ISA for RV32I!!!
  - Get yourself familiar with the ISA Spec.
- Instruction type:
  - R-type
  - I-type
  - S-type
  - B-type
  - J-type
  - U-type
- Implementation suggested is straightforward, yet there are modalities in how to implement it well at gate level.
- Single-cycle datapath is slow – need to pipeline it

- **RISC-V Datapath & Control**
  - R-type
  - I-type
  - S-type
  - B-type
  - J-type
  - U-type
  - **Control Logic**

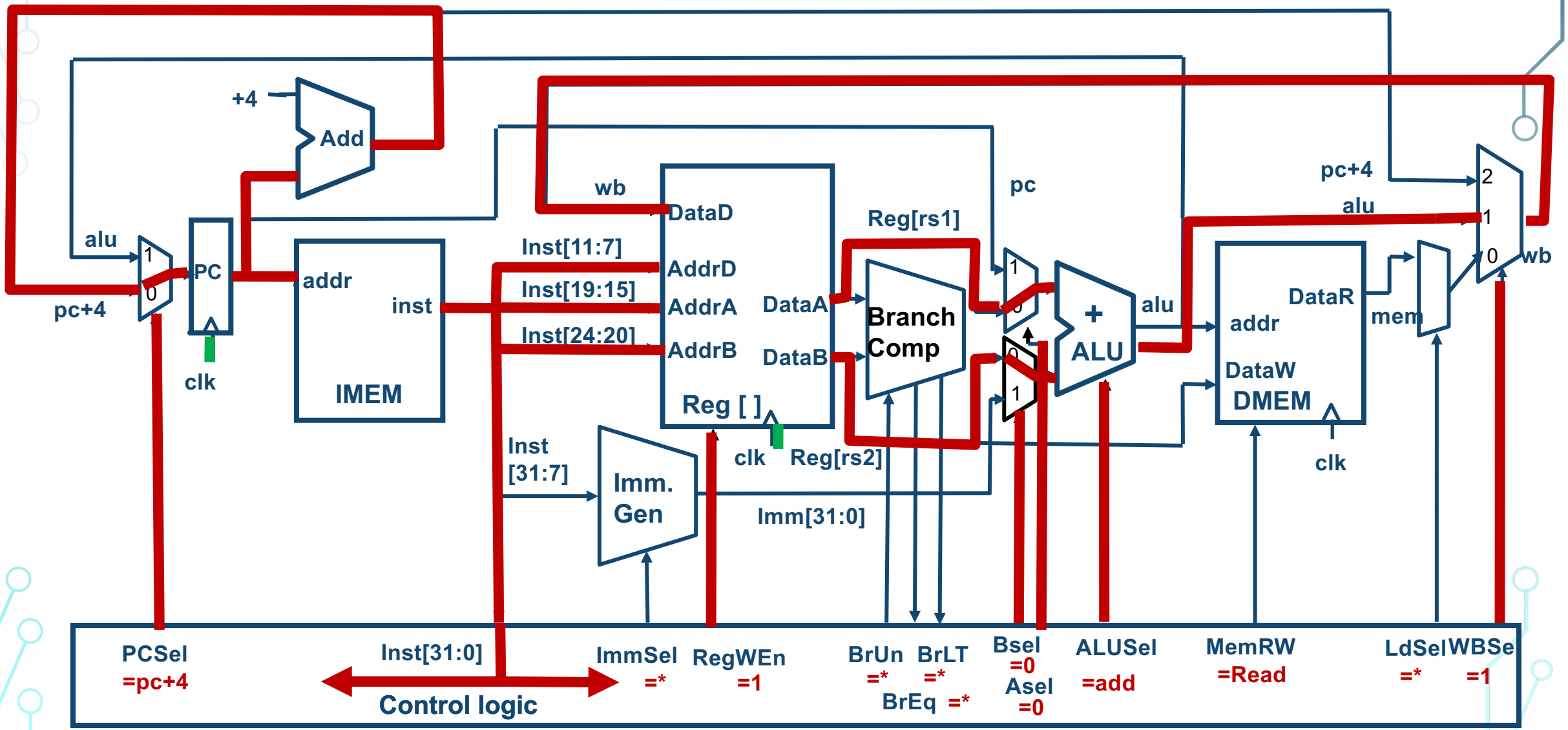
Shao Fall 2022 © UCB

Complete RV32I Datapath with Control



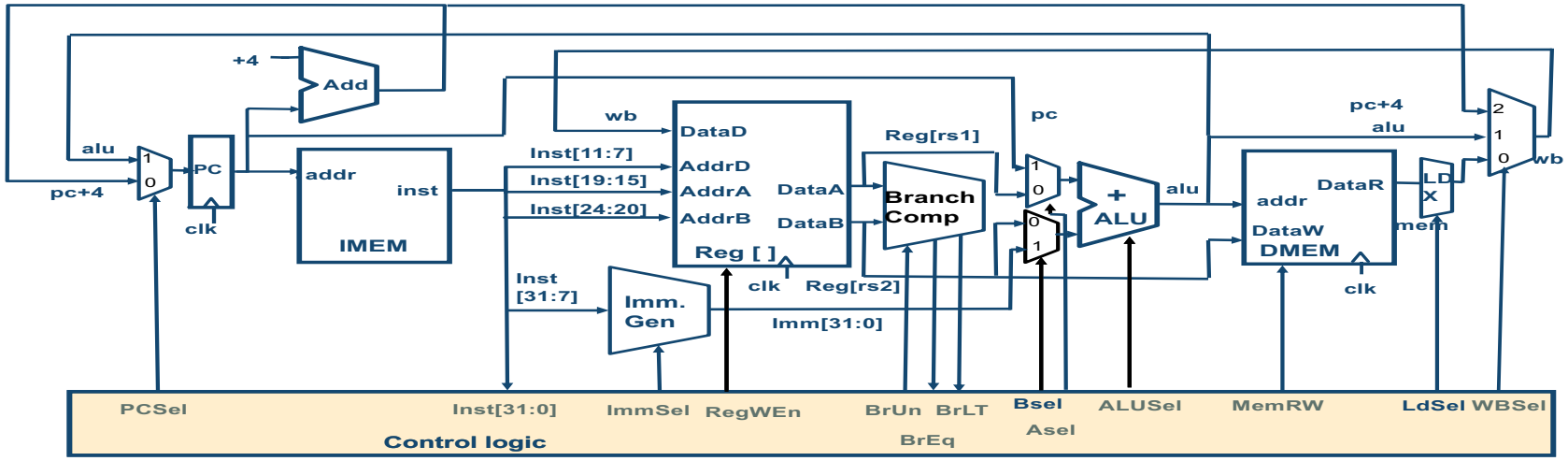
Shao Fall 2022 © UCB

Example: add


+4
Add
alu
pc+4
PC
clk
addr
inst
IMEM
wb
DataD
AddrD
AddrA
AddrB
Reg [ ]
clk
DataA
DataB
Inst[11:7]
Inst[19:15]
Inst[24:20]
Inst
[31:7]
Imm.
Gen
Imm[31:0]
Reg[rs1]
Reg[rs2]
pc
Branch
Comp
+
ALU
alu
addr
DataR
DataW
DMEM
clk
mem
pc+4
alu
wb
PCSel
=pc+4
Inst[31:0]
Control logic
ImmSel
=*
RegWEn
=1
BrUn
=*
BrLT
=*
BrEq
=*
Bsel
=0
Asel
=0
ALUSel
=add
MemRW
=Read
LdSel
=*
WBSel
=1


31
25
24
20
19
15
14
12
11
7
6
0
funct7
rs2
rs1
funct3
rd
opcode
7
5
5
3
5
7

add Execution



Clock			
PC	1000	1004	
PC+4	1004	1008	
inst[31:0]	`add x1,x2,x3`	`add x6,x7,x9`	
Control logic	`add control`	`add control`	
Reg[rs1]	`Reg[2]`	`Reg[7]`	
Reg[rs2]	`Reg[3]`	`Reg[9]`	
alu	`Reg[2]+Reg[3]`	`Reg[7]+Reg[9]`	
wb	`Reg[2]+Reg[3]`	`Reg[7]+Reg[9]`	
Reg[1]	???	`Reg[2]+Reg[3]`	

Control Logic Truth Table

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
`add`	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
`sub`	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
`addi`	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
`lw`	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
`sw`	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
`beq`	0	*	+4	B	*	PC	Imm	Add	Read	0	*
`beq`	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	1	*	+4	B	*	PC	Imm	Add	Read	0	*
`blt`	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
`bltu`	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
`jalr`	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
`jal`	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
`auipc`	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

RV32I, a nine-bit ISA!

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

inst[30]

inst[14:12]

inst[6:2]

Instruction type encoded using only 9 bits inst[30],inst[14:12], inst[6:2]

© UCB

Control Realization Options

- ROM
  - “Read-Only Memory”
  - Regular structure
  - Can be easily reprogrammed
    - fix errors
    - add instructions
- Combinatorial Logic
  - Decoder is typically hierarchical
    - First decode opcode, and figure out instruction type
    - E.g. branches are Inst[6:2] = 11000
    - Then determine the actual instruction
    - Inst[30] + Inst[14:12]
  - Modularity helps simplify and speed up logic
    - Narrow problem space for logic synthesis

Shao Fall 2022 © UCB
Nikolić, Shao Fall 2019 © UCB

Combinational Logic Control

- Simplest example: BrUn

inst[14:12] inst[6:2]

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

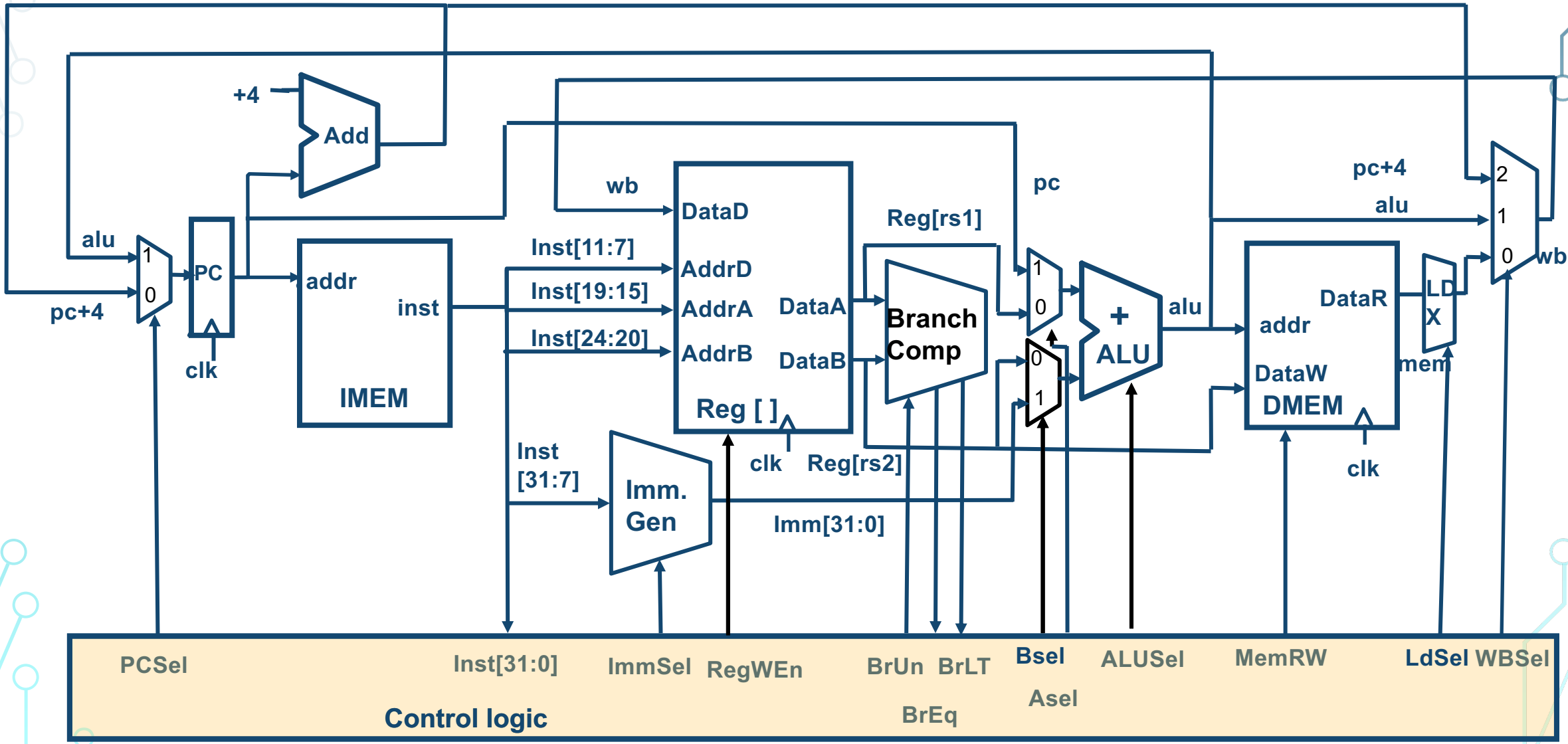
inst[14:13]

inst[12]	00	01	11	10
0				
1				

- **How to decode whether BrUn is 1?**
  - **Branch = Inst[6] • Inst[5] • !Inst[4] • !Inst[3] • !Inst[2]**
  - **BrUn = Inst [13] • Branch**

Shao Fall 2022 © UCB

Complete RV32I Datapath with Control



Shao Fall 2022 © UCB

- RISC-V Pipelining
  - 5-Stage Pipeline
  - Pipeline Hazards
    - Structural
    - Data
    - Control

Shao Fall 2022 © UCB

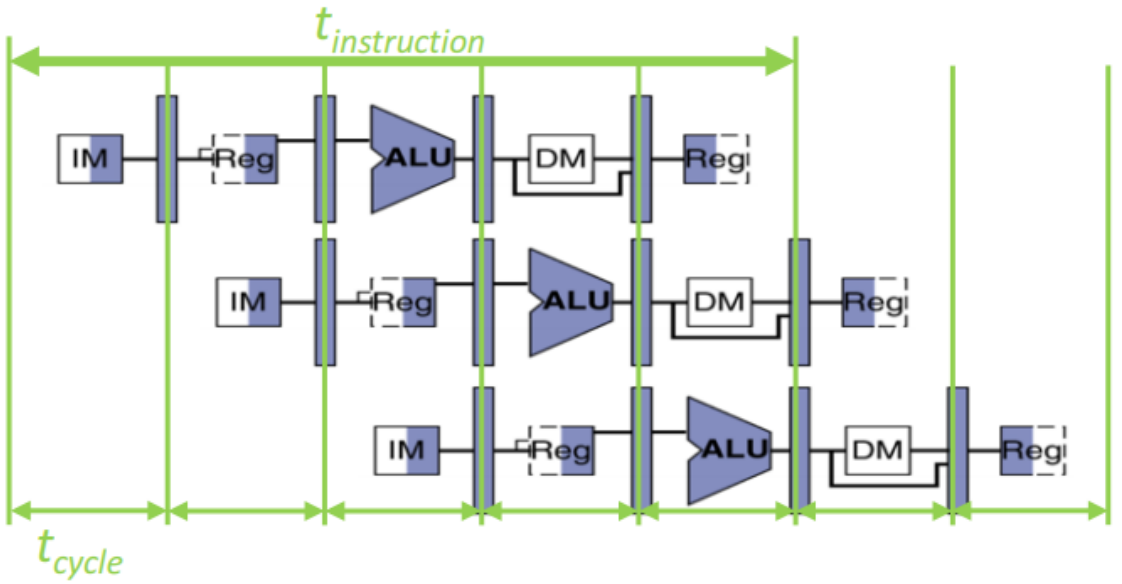
Pipelining with RISC-V

instruction sequence

add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3



	Single Cycle	Pipelining
Timing	t_{step} = 100 … 200 ps	t_{cycle} = 200 ps
	Register access only 100 ps	All cycles same length
Instruction time, $t_{instruction}$	= t_{cycle} = 800 ps	1000 ps
Clock rate, f_s	1/800 ps = 1.25 GHz	1/200 ps = 5 GHz
Relative speed	1 x	4 x

Pipelining with RISC-V

add t0, t1, t2

or t3, t4, t5

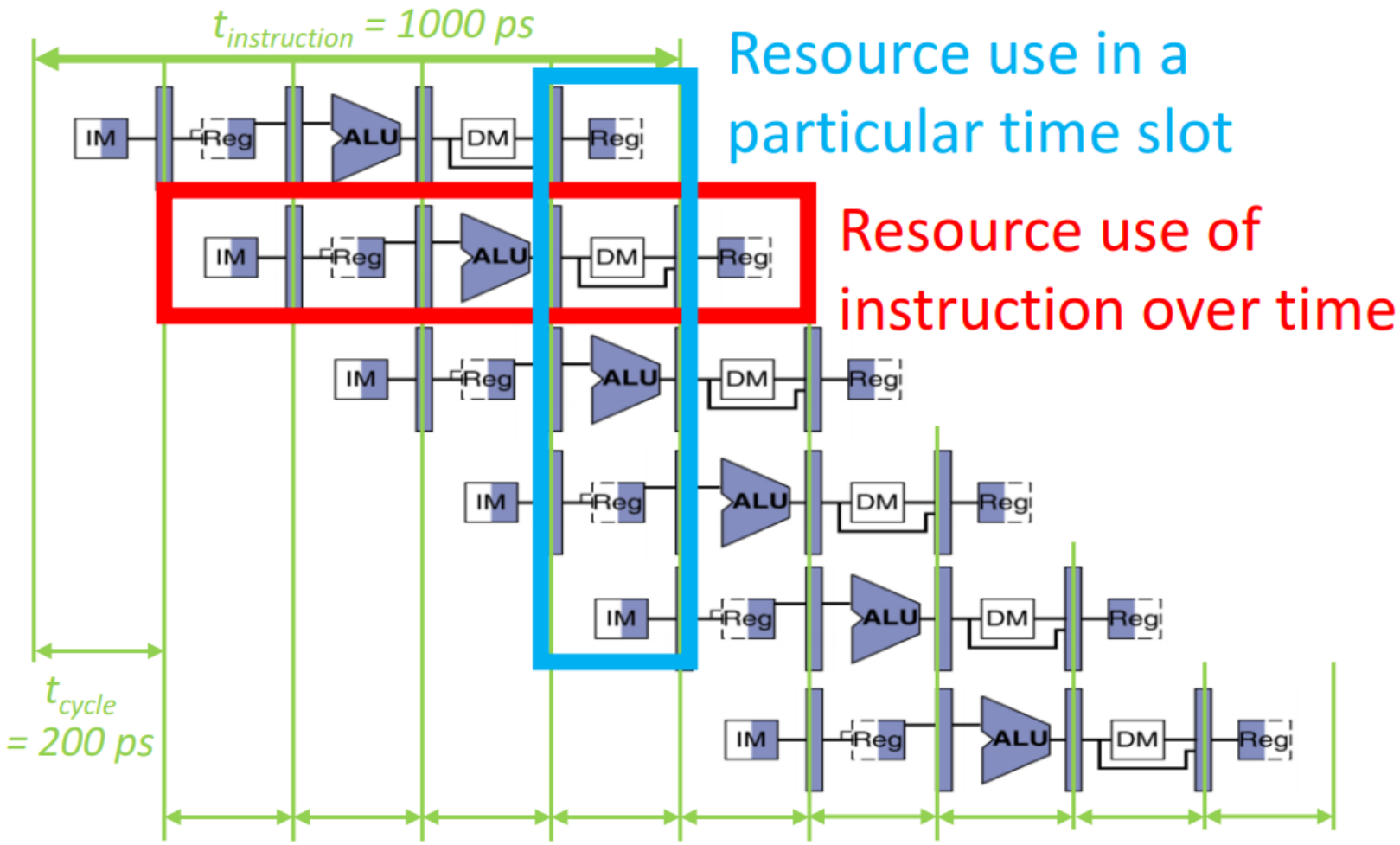
slt t6, t0, t3

sw t0, 4(t3)

lw t0, 8(t3)

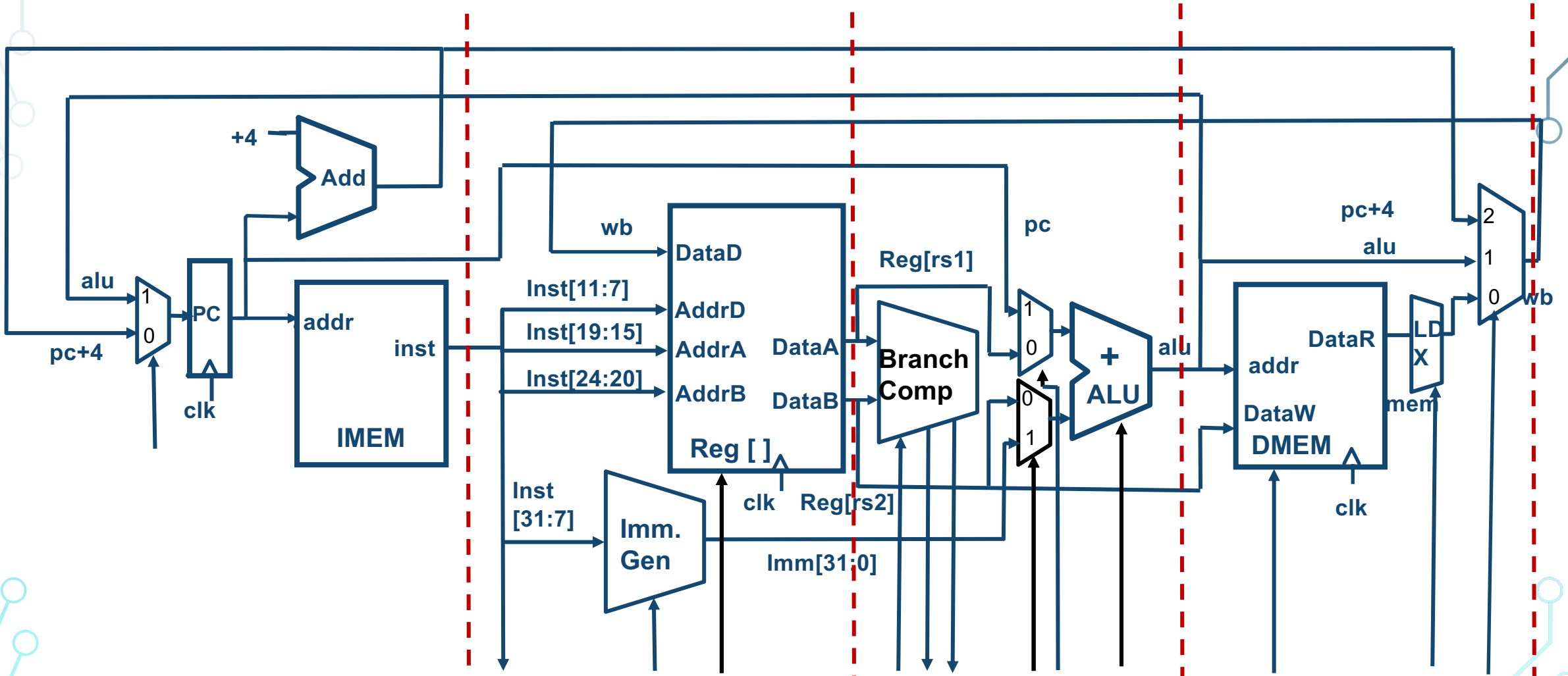
addi t2, t2, 1

instruction sequence



Shao Fall 2022 © UCB

Complete RV32I Datapath with Control



Instruction Fetch (F)

Instruction Decode/ Register Read (D)

ALU Execute (X)

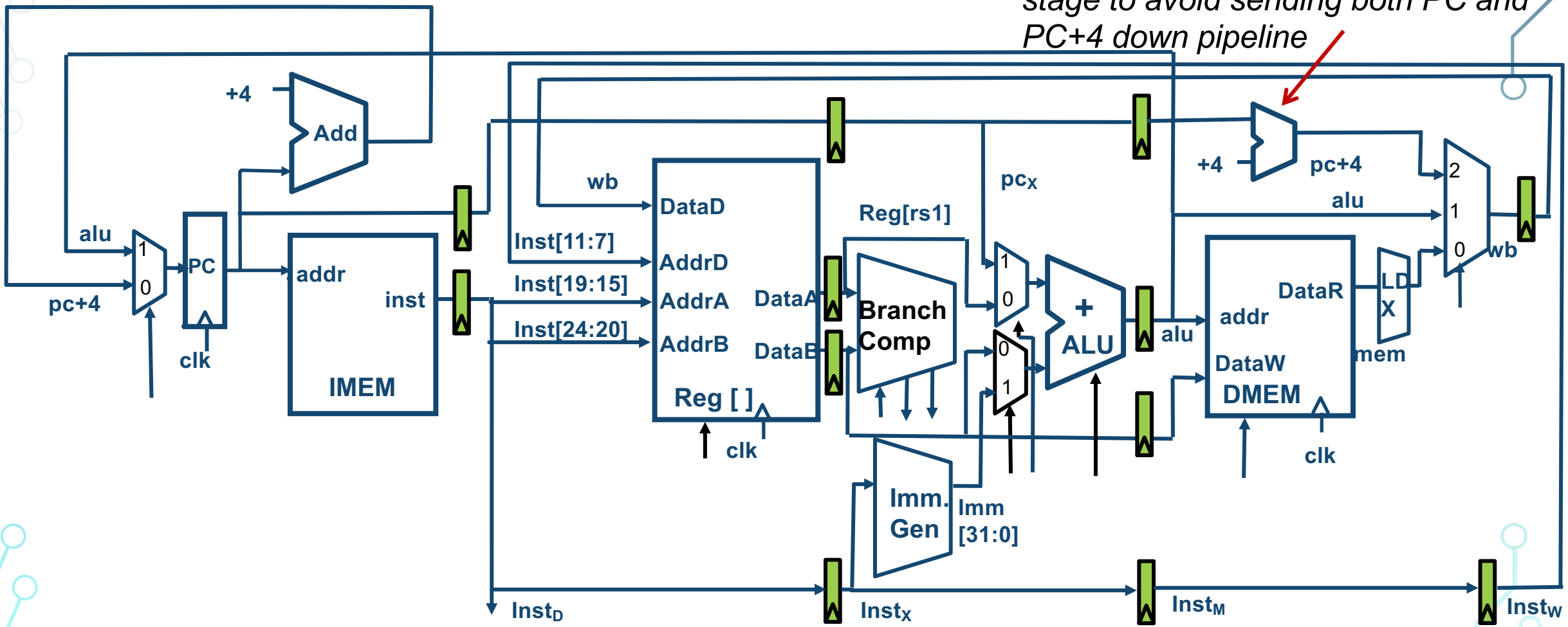
Memory Access (M)

Write Back (W)

Shao Fall 2022 © UCB

Pipelining RV32I Datapath

In this design, recalculate PC+4 in M stage to avoid sending both PC and PC+4 down pipeline



Instruction Fetch (F)

Instruction Decode/ Register Read (D)

ALU Execute (X)

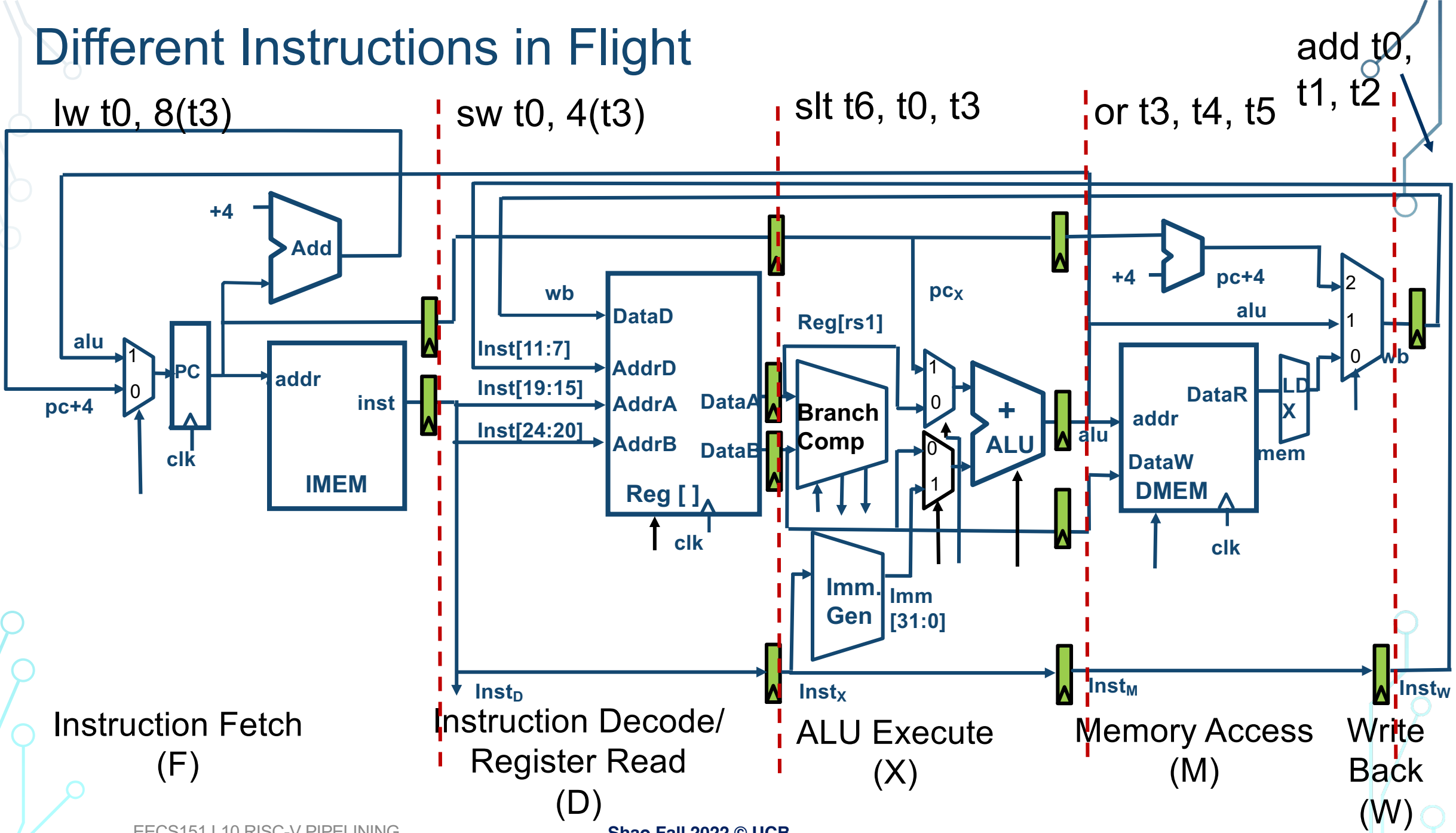
Memory Access (M)

Write Back (W)

Must pipeline instruction along with data, so control operates correctly in each stage

Shao Fall 2022 © UCB

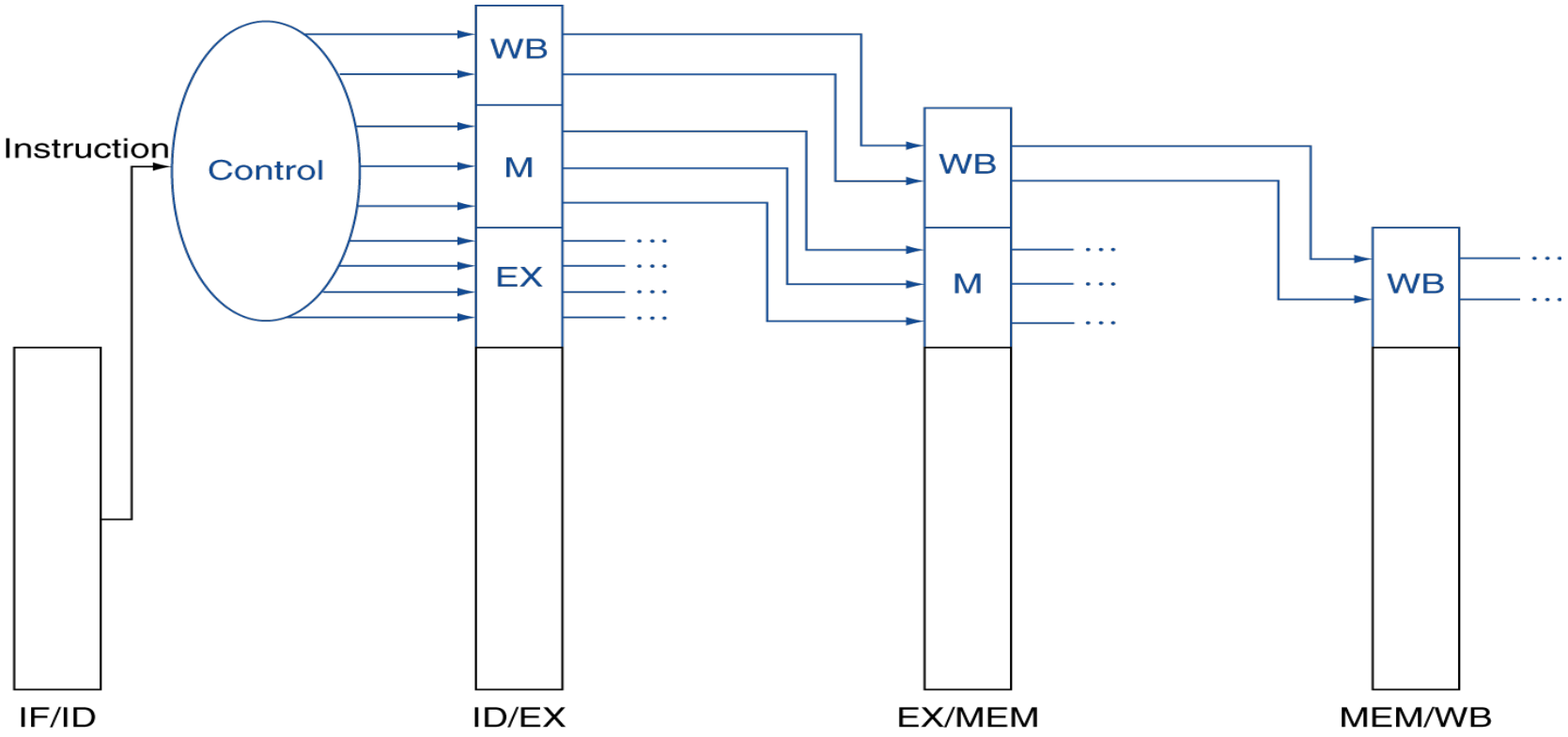
Different Instructions in Flight



Shao Fall 2022 © UCB

Pipelined Control

- Control signals derived from instruction
  - As in single-cycle implementation
  - Information is stored in pipeline registers for use by later stages



Administrivia

- FPGA Guest Lecture next week
- No new lab this week.
- New homework.

- RISC-V Pipelining
  - 5-Stage Pipeline
  - Pipeline Hazards
    - Structural
    - Data
    - Control

Shao Fall 2022 © UCB

Pipelining Hazards

A *hazard* is a situation that prevents starting the next instruction in the next clock cycle

1) *Structural hazard*
   - A required resource is busy (e.g. needed in multiple stages)
2) *Data hazard*
   - Data dependency between instructions
   - Need to wait for previous instruction to complete its data read/write
3) *Control hazard*
   - Flow of execution depends on previous instruction

Structural Hazard

- **Problem:** Two or more instructions in the pipeline compete for access to a single physical resource
- **Solution 1:** Instructions take it in turns to use resource, some instructions have to stall
- **Solution 2:** Add more hardware to machine
- *Can always solve a structural hazard by adding more hardware*

Structural Hazards I: Regfile

- Each instruction:
  - can read up to two operands in decode stage
  - can write one value in writeback stage
- Avoid structural hazard by having separate “ports”
  - two independent read ports and one independent write port
- Three accesses per cycle can happen simultaneously
  - We will see a circuit design for multi-ported register files later in the class
- Processors with multiple execution paths have larger number of ports

Shao Fall 2022 © UCB

Structural Hazard II : Memory Access

instruction sequence

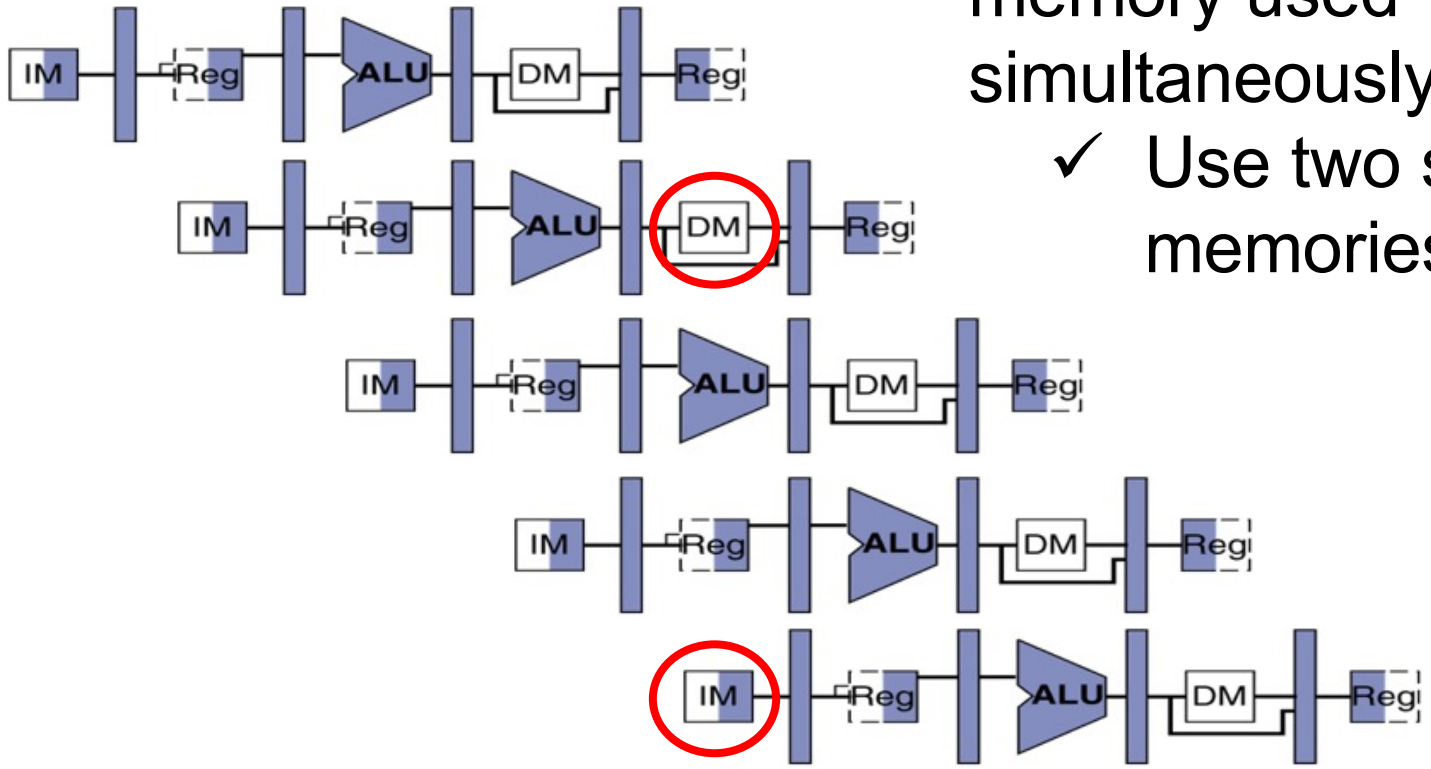
add t0, t1, t2

or t3, t4, t5

slt t6, t0, t3

sw t0, 4(t3)

lw t0, 8(t3)



- Instruction and data memory used simultaneously
  - ✓ Use two separate memories

Shao Fall 2022 © UCB

Structural Hazards – Summary

- Conflict for use of a resource
- In RISC-V pipeline with a single memory
  - Load/store requires data access
  - Without separate memories, instruction fetch would have to *stall* for that cycle
    - All other operations in pipeline would have to wait
- Pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches
- RISC ISAs (including RISC-V) designed to avoid structural hazards
  - e.g. at most one memory access/instruction

- RISC-V Pipelining
  - 5-Stage Pipeline
  - Pipeline Hazards
    - Structural
    - Data
    - Control

Shao Fall 2022 © UCB

Data Hazard: Register Access

- Separate ports, but what if write to same value as read?
- Does **`sw`** in the example fetch the old or new value?

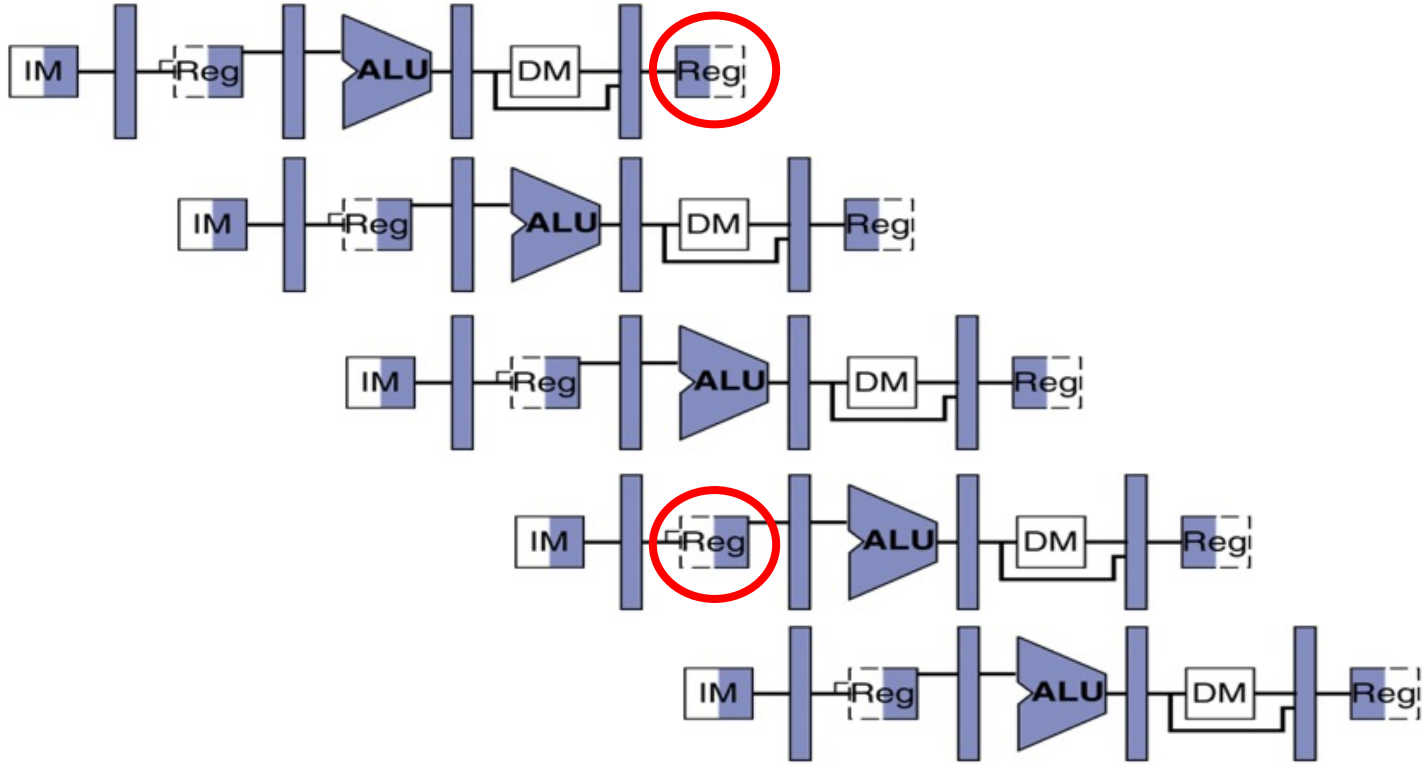
add t0, t1, t2

or t3, t4, t5

slt t6, t0, t3

sw t0, 4(t3)

lw t0, 8(t3)



instruction sequence

Register Access Policy

- Exploit high speed of register file (100 ps)
  1) WB updates value
  2) ID reads new value
- Indicated in diagram by shading

instruction sequence

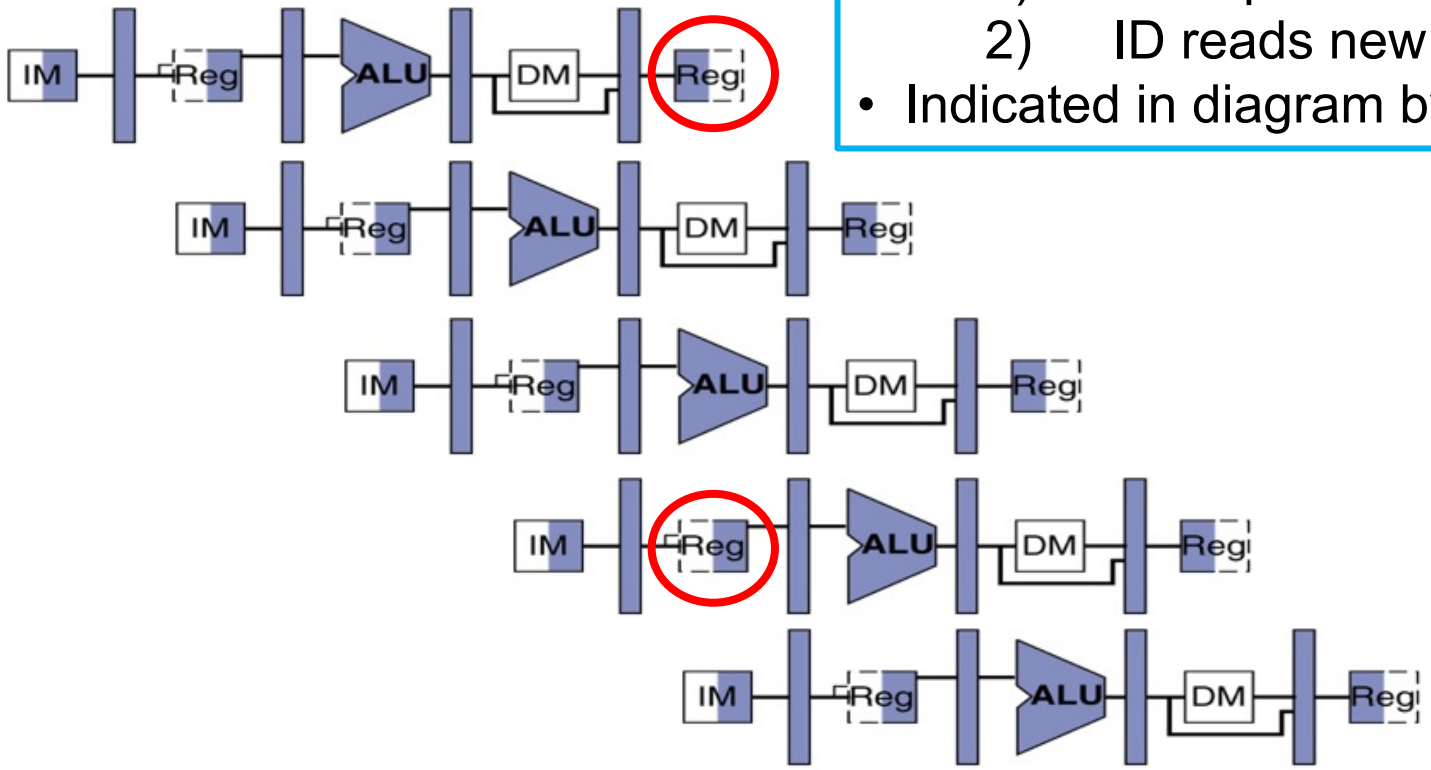
add t0, t1, t2

or t3, t4, t5

slt t6, t0, t3

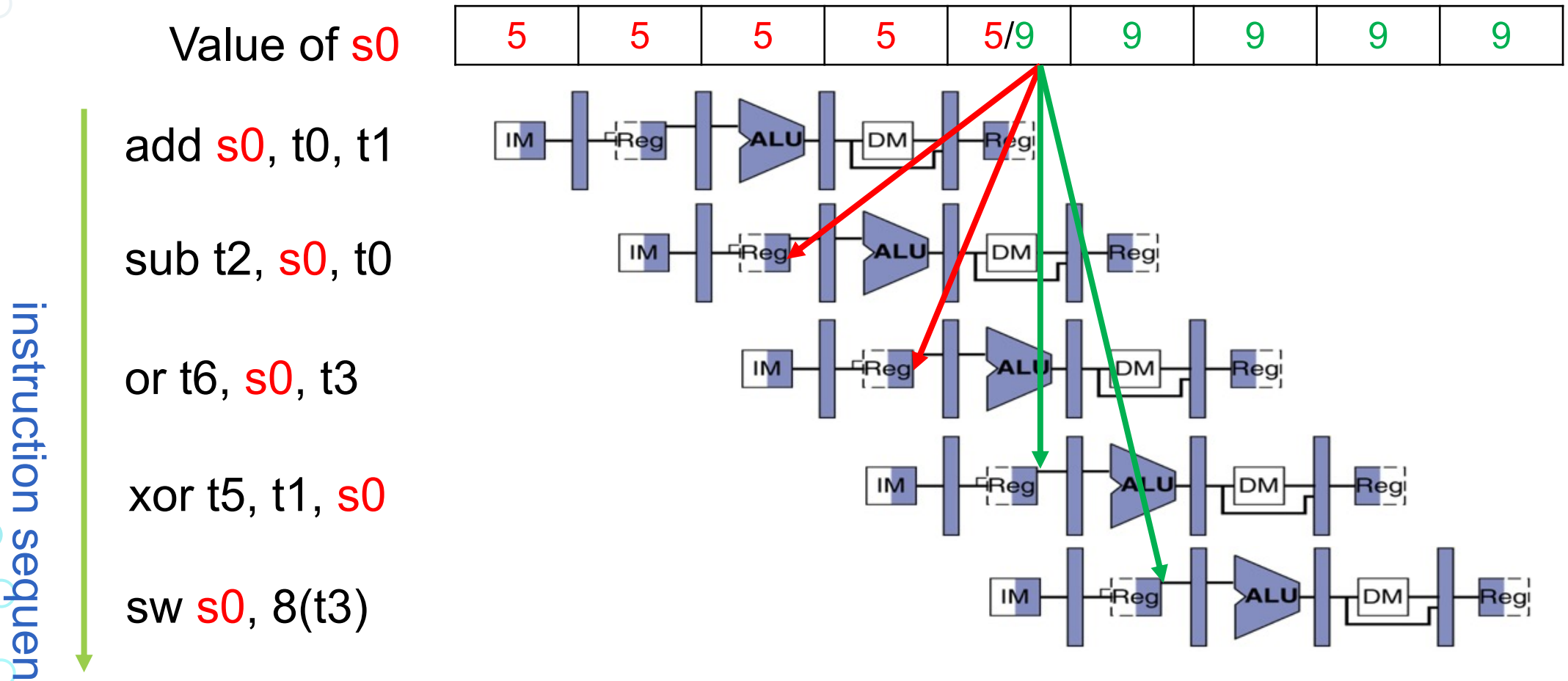
sw t0, 4(t3)

lw t0, 8(t3)



Might not always be possible to write then read in same cycle, especially in high-frequency designs. Check assumptions in any question.

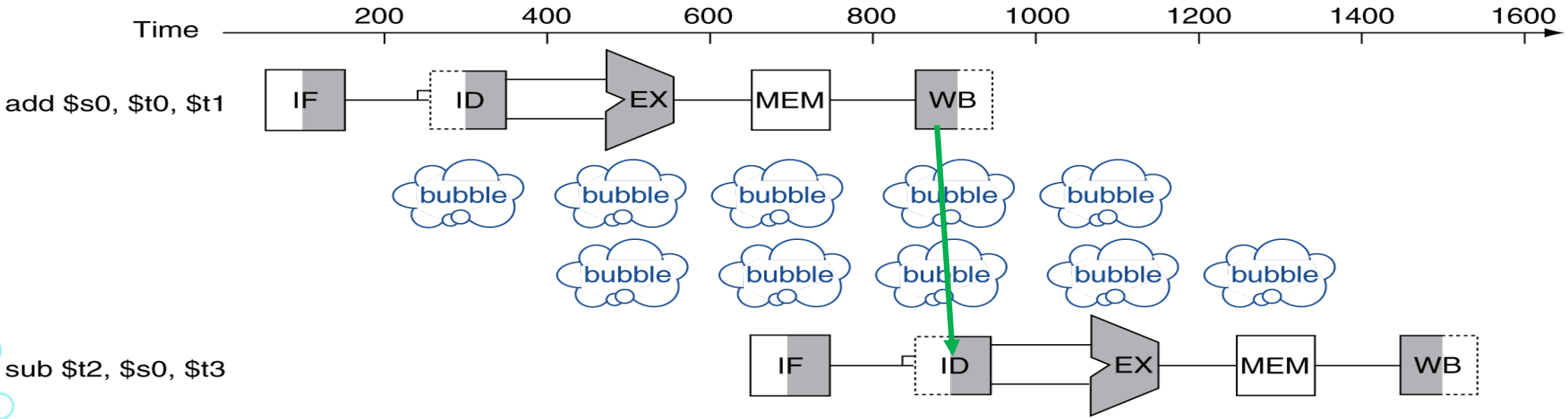
Data Hazard: ALU Result



Without some fix, **`sub`** and **`or`** will calculate wrong result!

Solution 1: Stalling

- Problem: Instruction depends on result from previous instruction
  - add s0, t0, t1
    sub t2, s0, t3



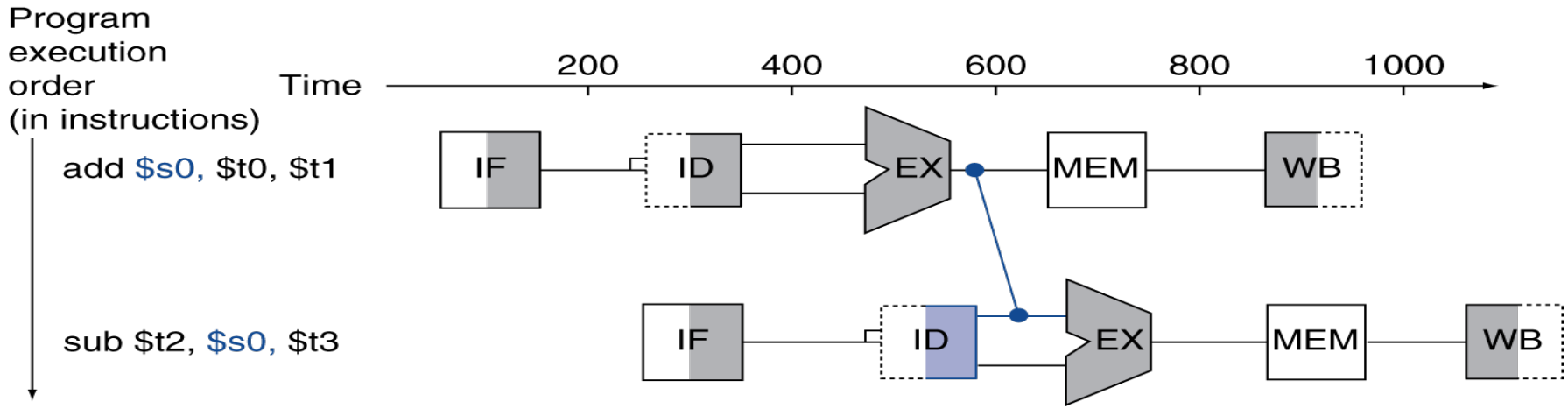
- Bubble:
  - effectively NOP: affected pipeline stages do “nothing”

Stalls and Performance

- Stalls reduce performance
  - But stalls are required to get correct results
- Compiler can arrange code or insert NOPs (writes to register x0) to avoid hazards and stalls
  - But requires knowledge of the pipeline structure

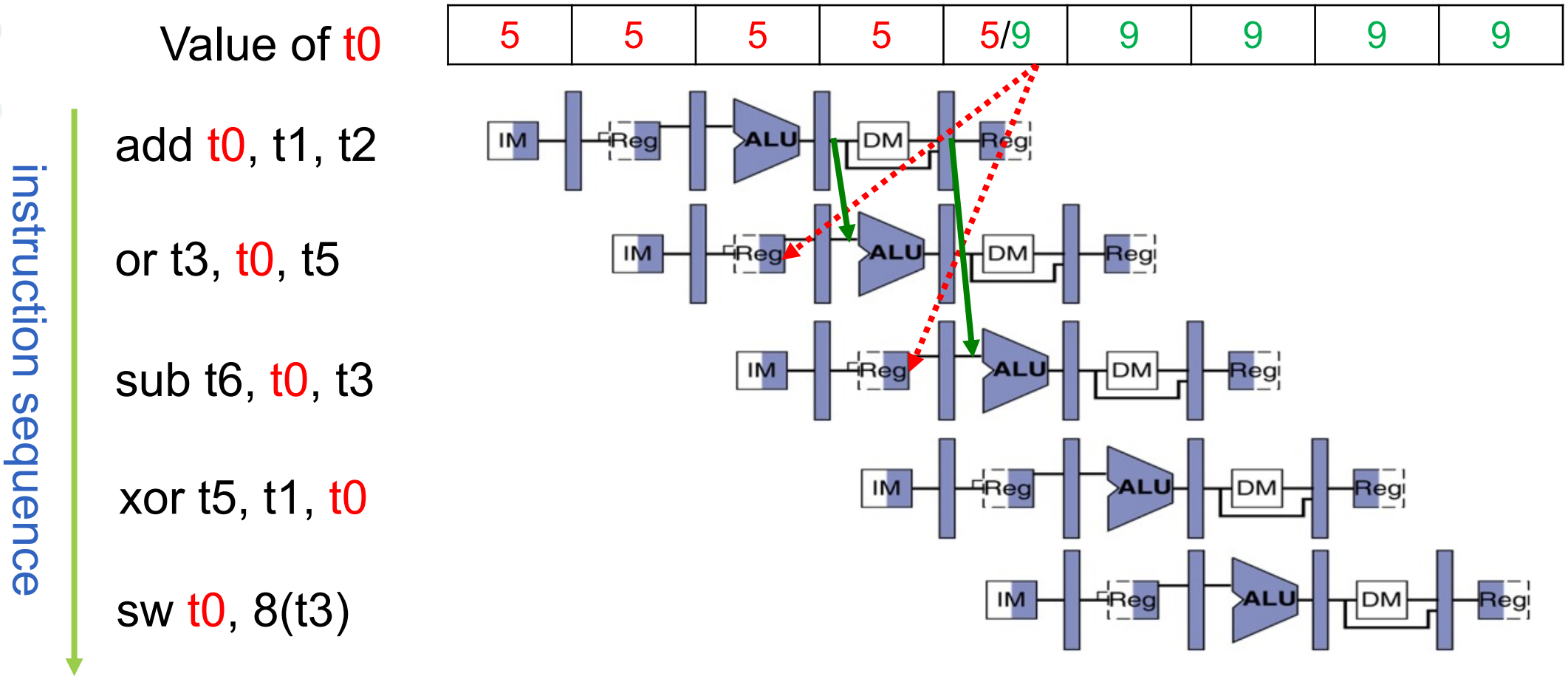
Solution 2: Forwarding

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



Shao Fall 2022 © UCB

Solution 2: Forwarding



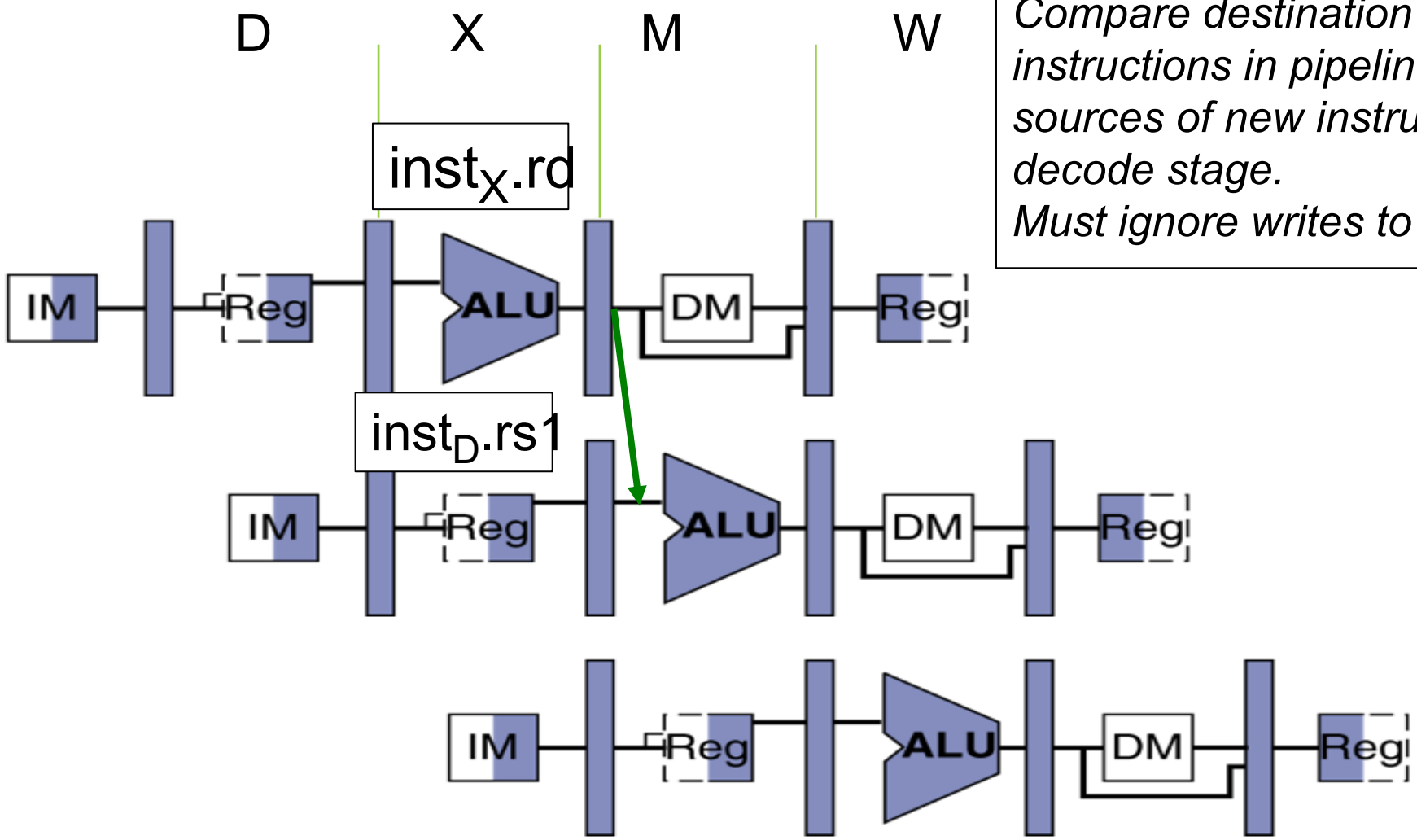
Forwarding: grab operand from pipeline stage, rather than register file

Detect Need for Forwarding (example)

add t0, t1, t2

or t3, t0, t5

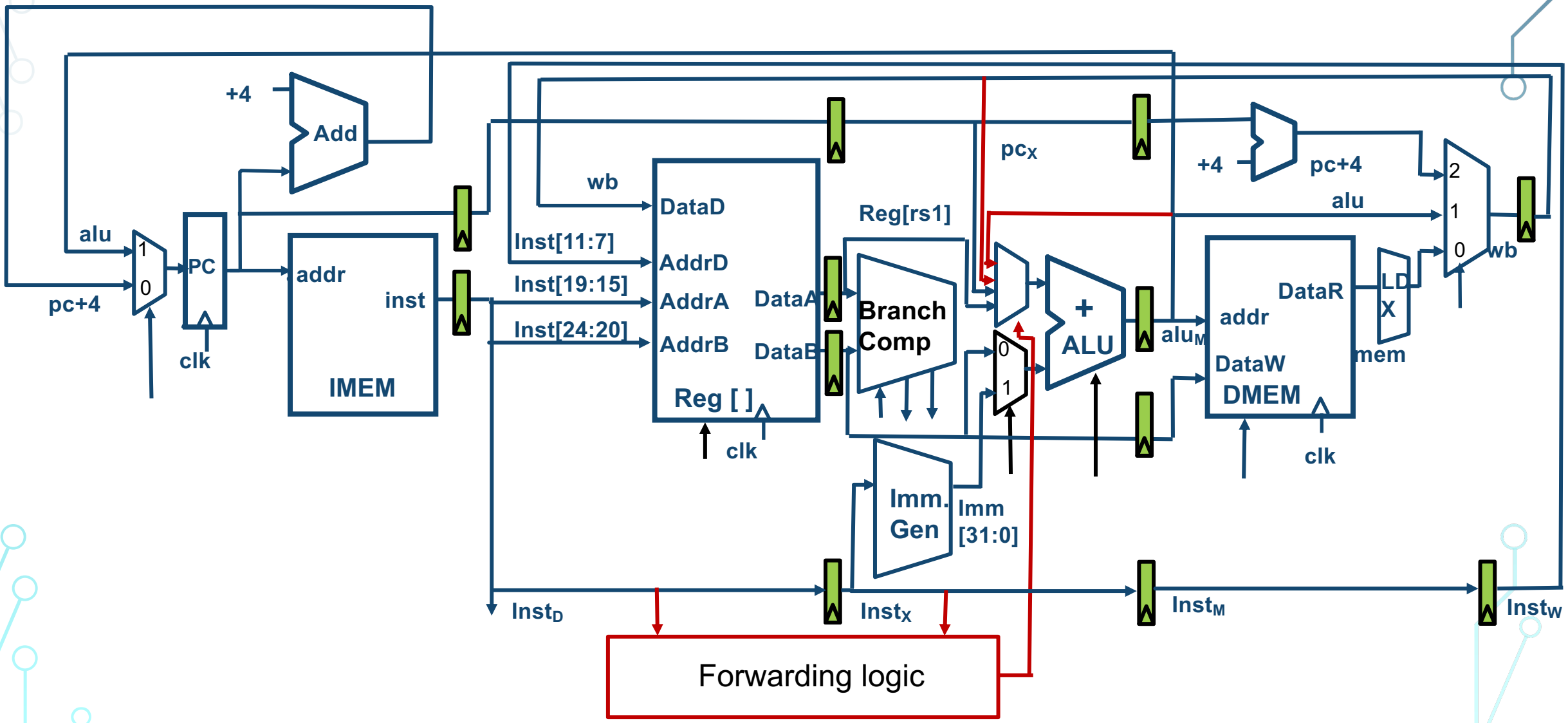
sub t6, t0, t3



Compare destination of older instructions in pipeline with sources of new instruction in decode stage.
Must ignore writes to x0!

Shao Fall 2022 © UCB

ForwardingA Path

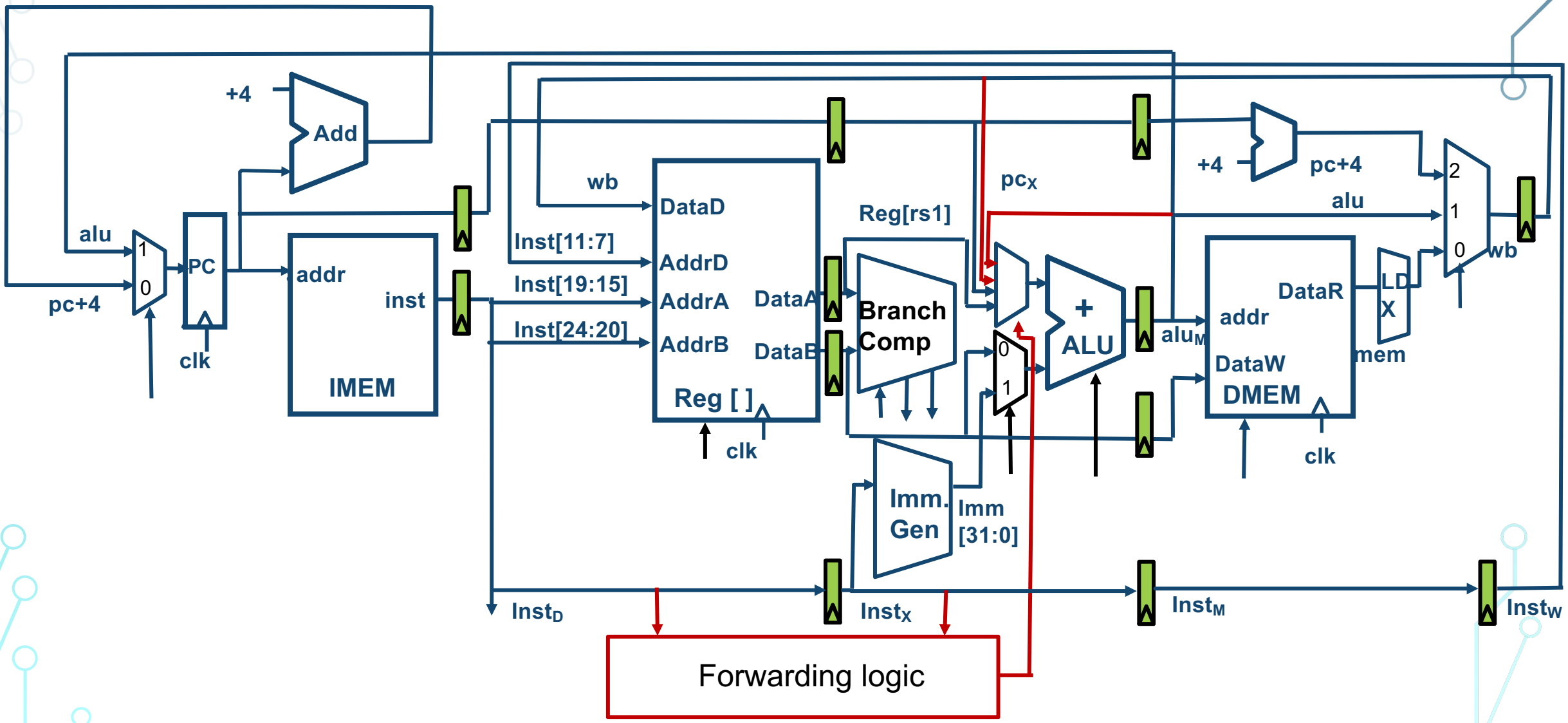


Shao Fall 2022 © UCB

- RISC-V Pipelining
  - 5-Stage Pipeline
  - Pipeline Hazards
    - Structural
    - Data
    - Control

Shao Fall 2022 © UCB

Pipelined Datapath: When is the target PC calculated?



Shao Fall 2022 © UCB

Control Hazards

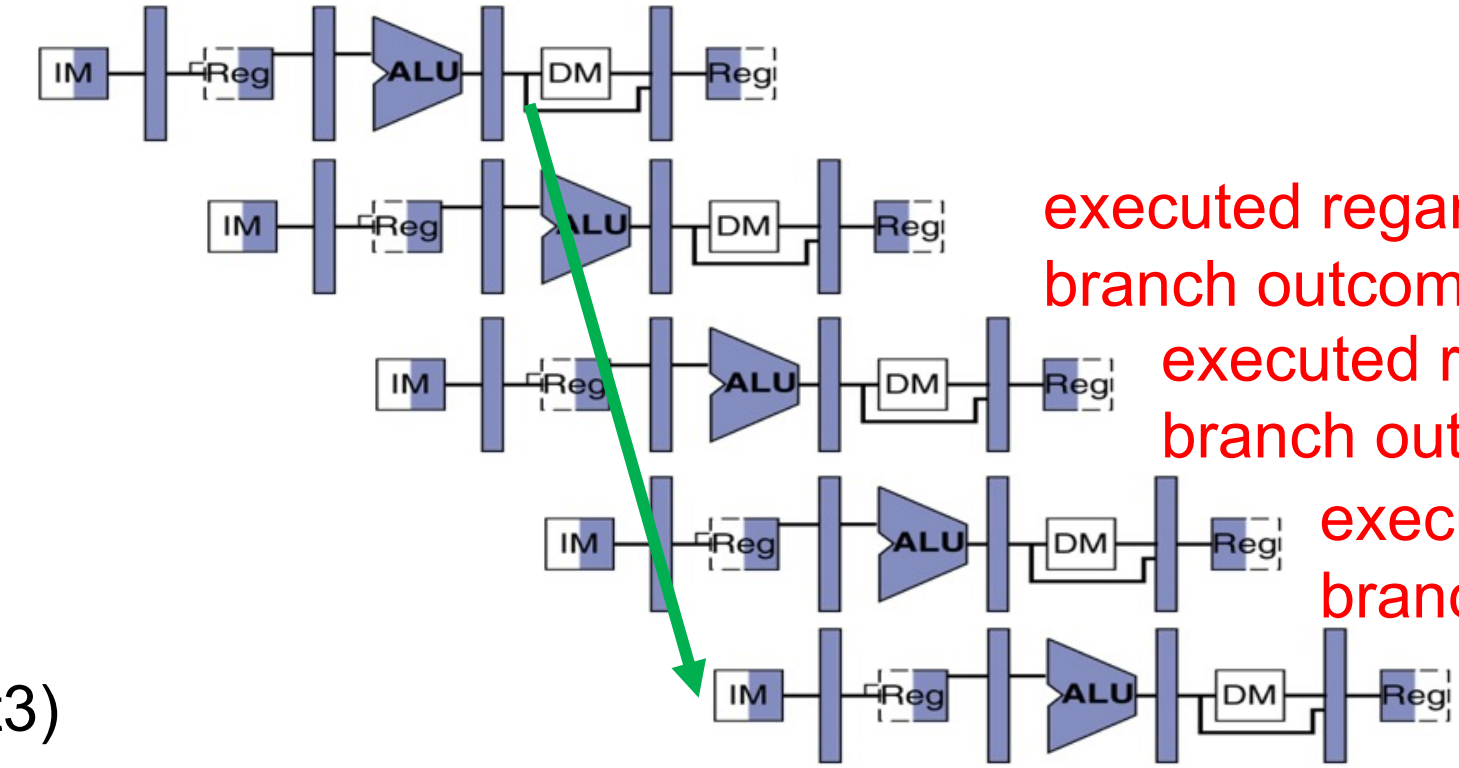
beq t0, t1, Label

sub t2, s0, t5

or t6, s0, t3

xor t5, t1, s0

Label: sw s0, 8(t3)



executed regardless of branch outcome!

executed regardless of branch outcome!!!

executed regardless of branch outcome!!!

PC updated reflecting branch outcome

Observation

- If branch not taken, then instructions fetched sequentially after branch are correct
- If branch or jump taken, then need to flush incorrect instructions from pipeline by converting to NOPs

Kill Instructions after Branch if Taken

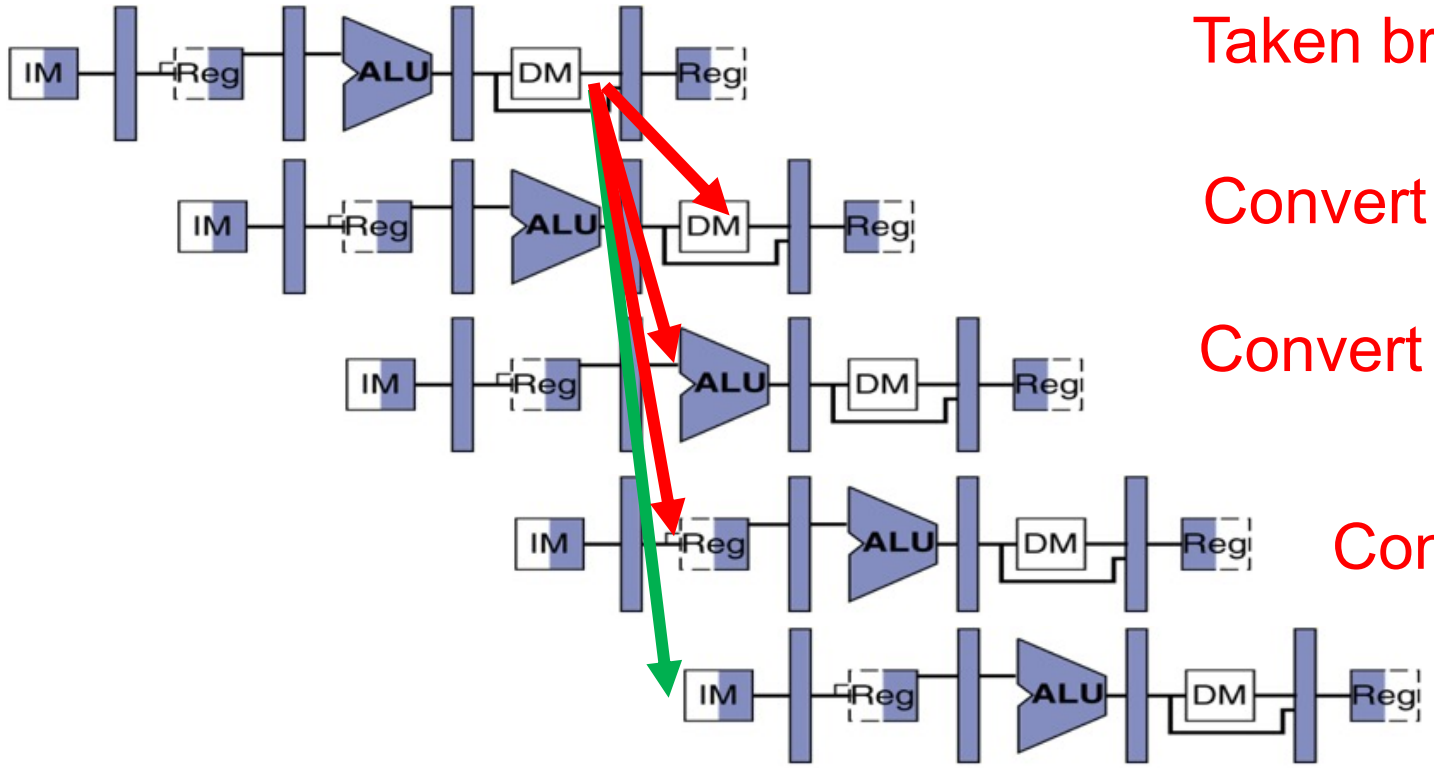
beq t0, t1, label

sub t2, s0, t5

or t6, s0, t3

Add t2, t3, s0

label: xxxxxx



Taken branch

Convert to NOP

Convert to NOP

Convert to NOP

PC updated reflecting branch outcome

Reducing Branch Penalties

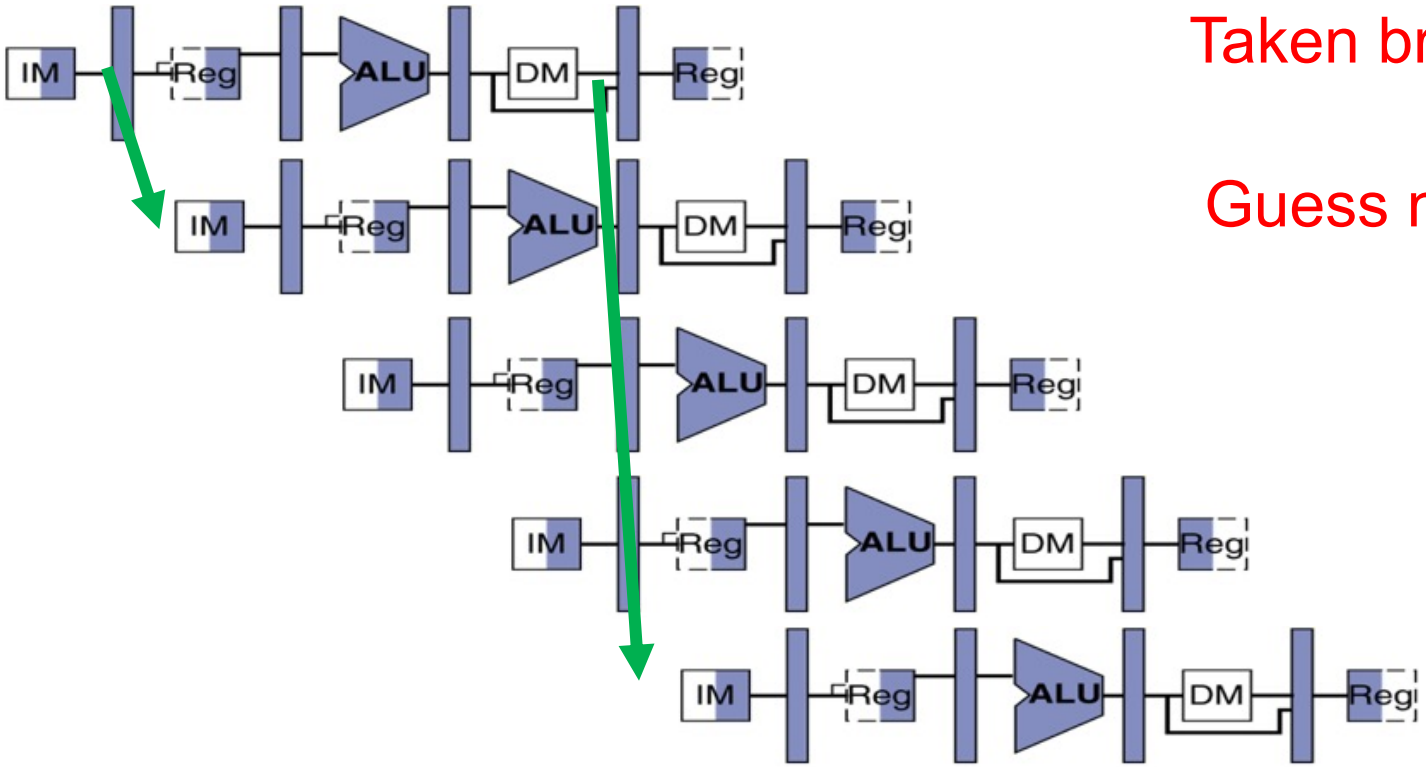
- In our datapath, every taken branch in simple pipeline costs 3 dead cycles
- To improve performance, use “branch prediction” to guess which way branch will go earlier in pipeline
- Only flush pipeline if branch prediction was incorrect

Branch Prediction

beq t0, t1, label

label: …..

…..



Taken branch

Guess next PC!

Check guess correct

Summary

- We have covered single-cycle, multi-cycle and pipelined datapaths
- Pipeline Hazards:
  - Structural
  - Data
  - Control hazards
- Need to be understood and handled appropriately
  - More resources
  - Forwarding
  - Stall