EECS151 : Introduction to Digital Design and ICs

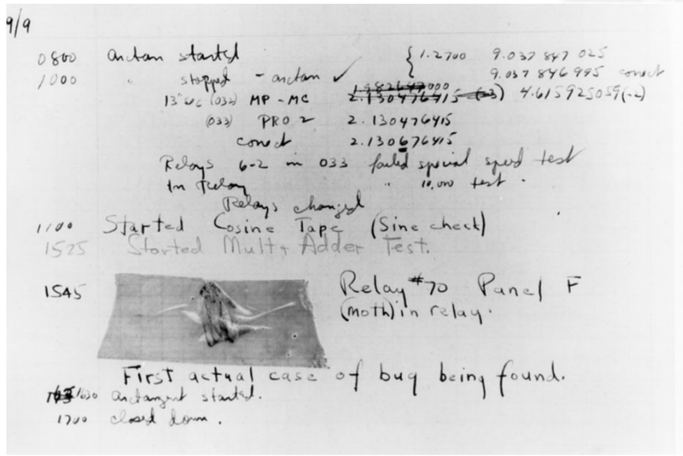
Lecture 9 – RISC-V Pipelining

Bora Nikolić and Sophia Shao

Stalking the Elusive Computer Bug

From at least the time of Thomas Edison, U.S. engineers have used the word "bug" to refer to flaws in the systems they developed. This short word conveniently covered a multitude of possible problems. It also suggested that difficulties were small and could be easily corrected. IBM engineers who installed the ASSC Mark I at Harvard University in 1944 taught the phrase to the staff there. Grace Murray Hopper used the word with particular enthusiasm in documents relating to her work. In 1947, when technicians building the Mark II computer at Harvard discovered a moth in one of the relays, they saved it as the first actual case of a bug being found. In the early 1950s, the terms "bug" and "debug," as applied to computers and computer programs, began to appear not only in computer documentation but even in the popular press.

Peggy Aldrich Kidwell,
IEEE Annals of the History of Computing , 1998.



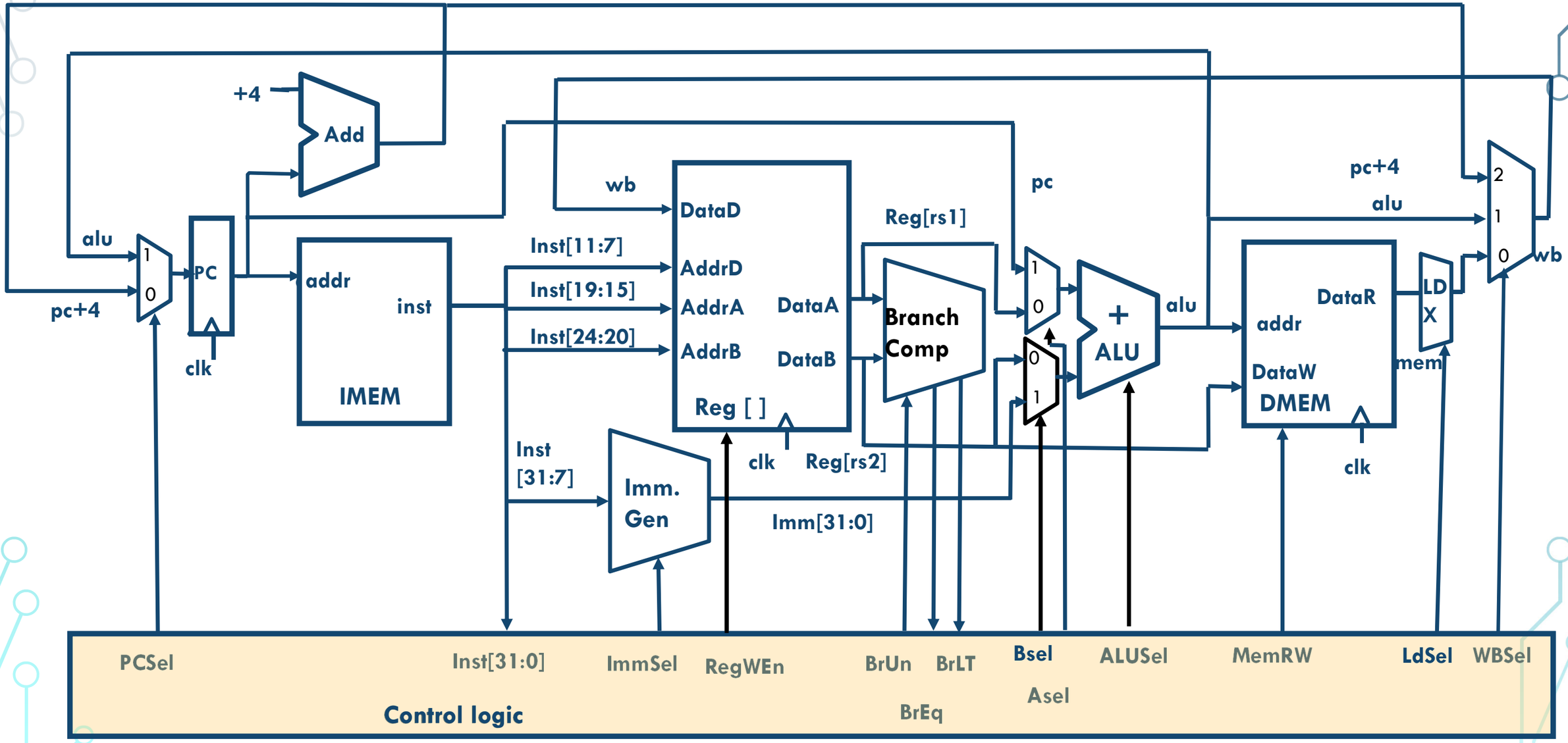
Grace Murray Hopper
Logbook of the Mark II for 9/9/1947

Review

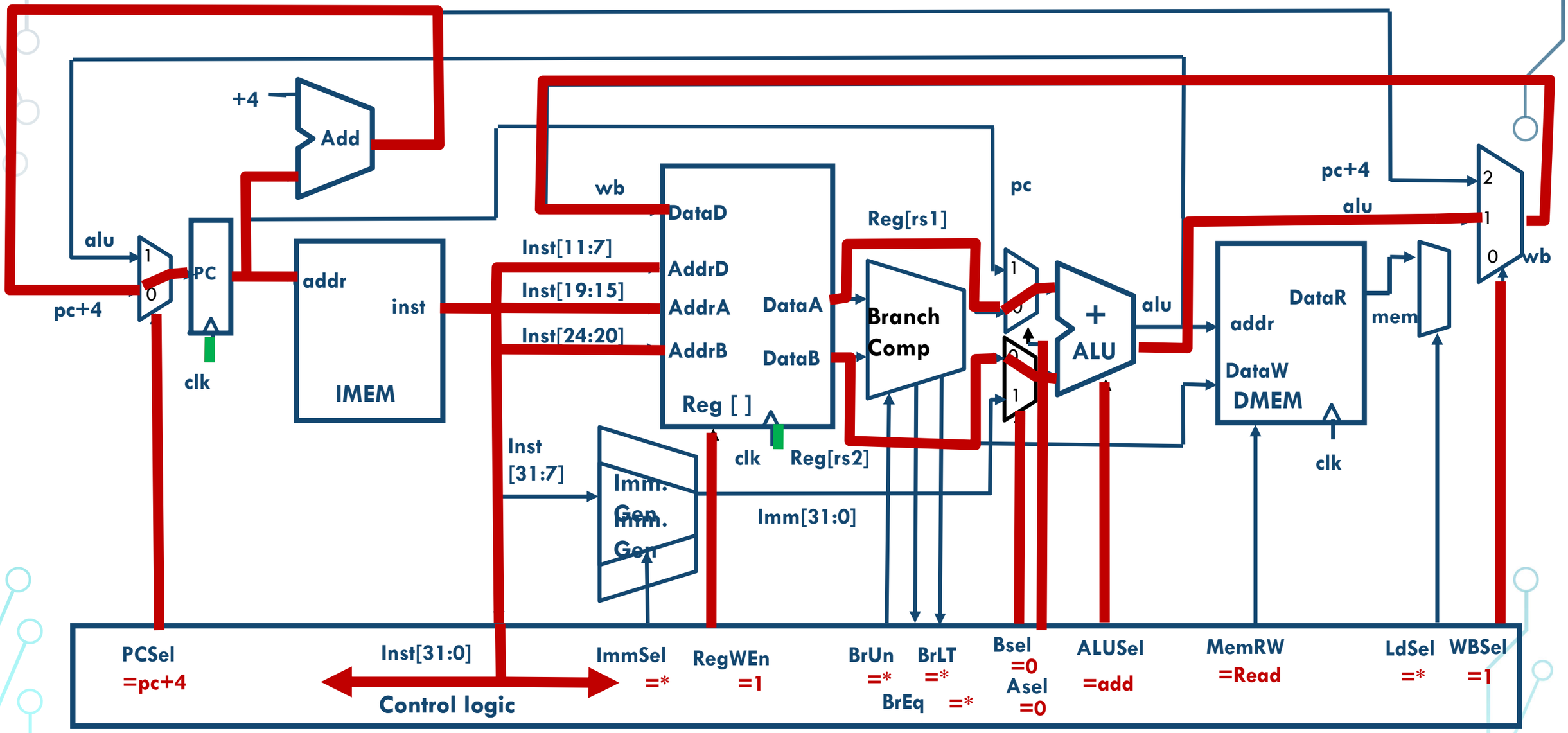
- We have covered the implementation of the base ISA for RV32I
- Implementation suggested is straightforward, yet there are modalities in how to implement it well at gate level.
- Still need control
- Single-cycle datapath is slow – need to pipeline it

RISC-V Control Logic

Complete RV32I Datapath with Control

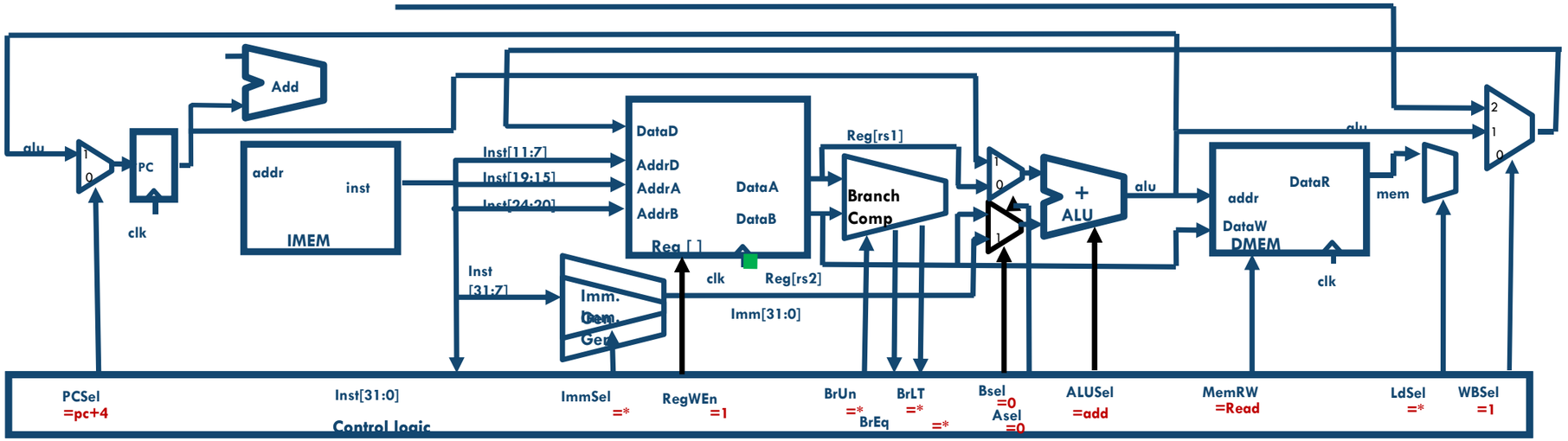


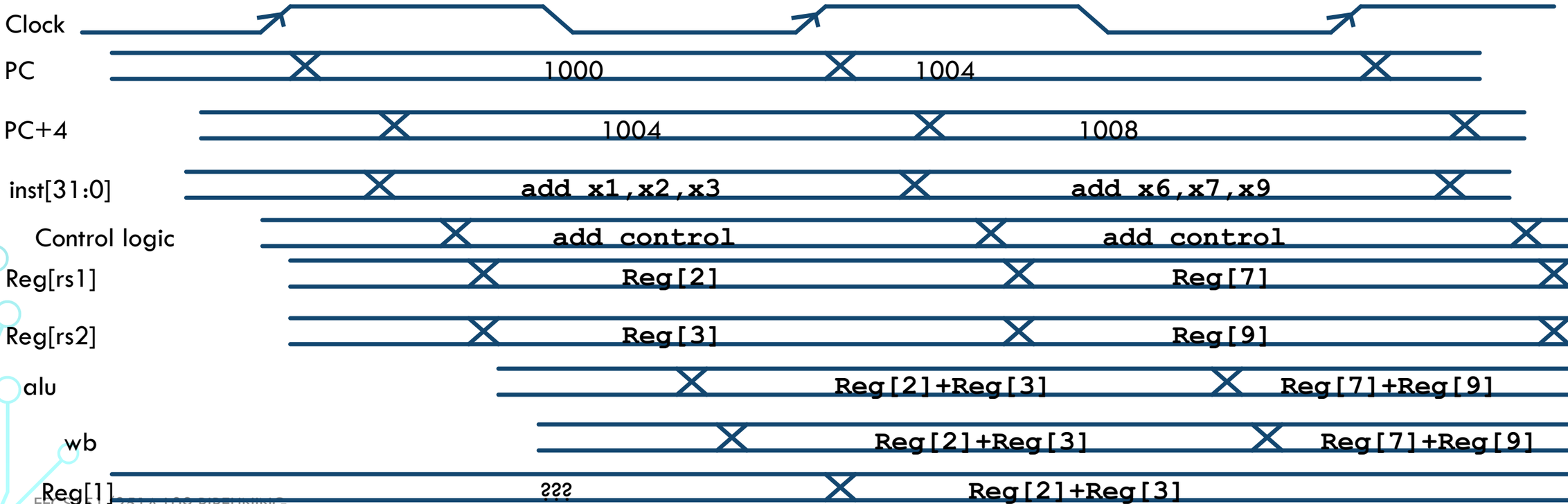
Example: add


+4
Add
alu
pc+4
PC
clk
addr
inst
IMEM
wb
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
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





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]

Control Realization Options

- ROM
  - "Read-Only Memory"
  - Regular structure
  - Can be easily reprogrammed
    - fix errors
    - add instructions
- Combinatorial Logic
  - More compact, faster
  - Use synthesis tools

Combinational Logic Control

- 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
  - Narrosw problem space for logic synthesis

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

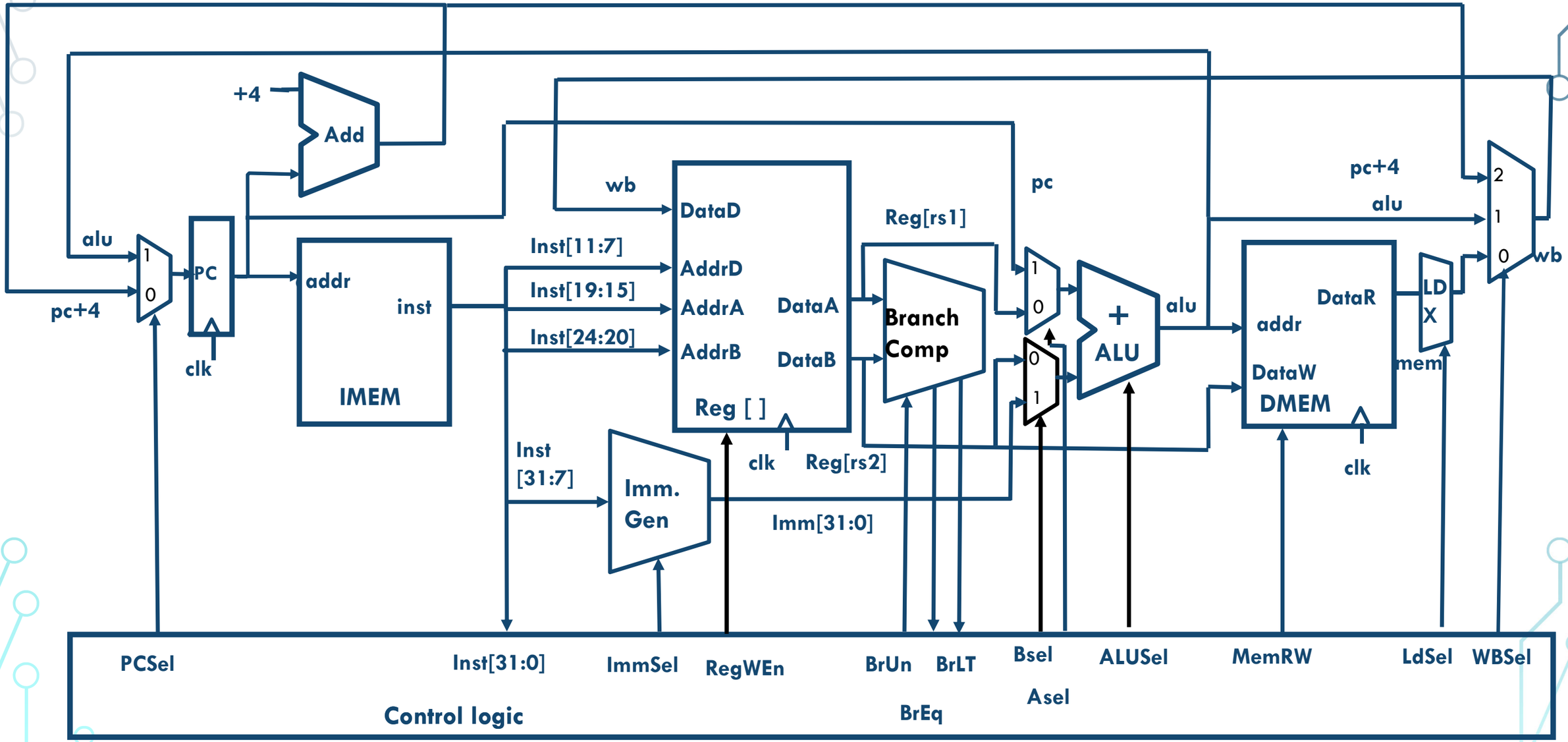
inst[14:13]

inst[12]

	00	01	11	10
0				
1				

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

Complete RV32I Datapath with Control



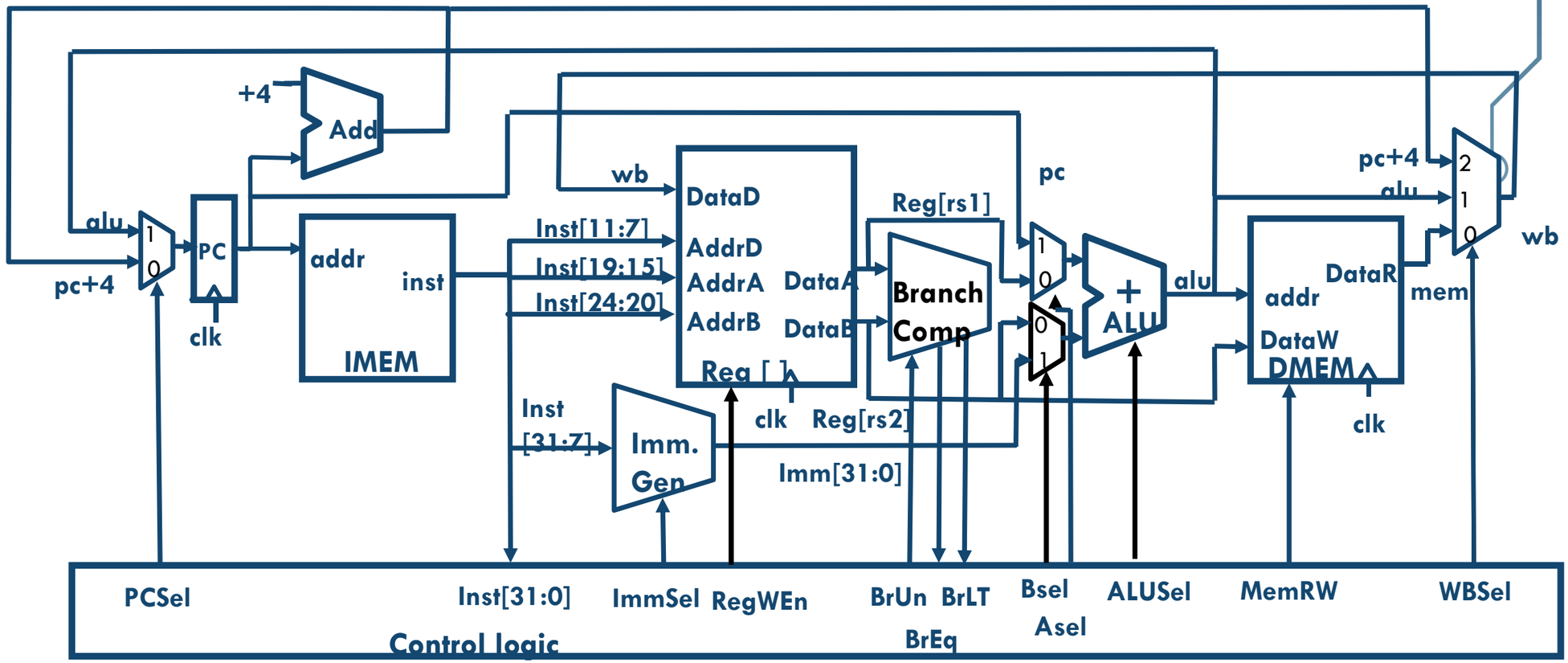
Multi-Cycle RV32I Datapath

- IMEM and DMEM model cache
- There is one, unified I+D memory in small processors w/o cache
- Can you draw a datapath with unifier I+D memory?
  - Requires multi-cycle operation

Peer Instruction(s): Critical Path

Critical
path for a addi

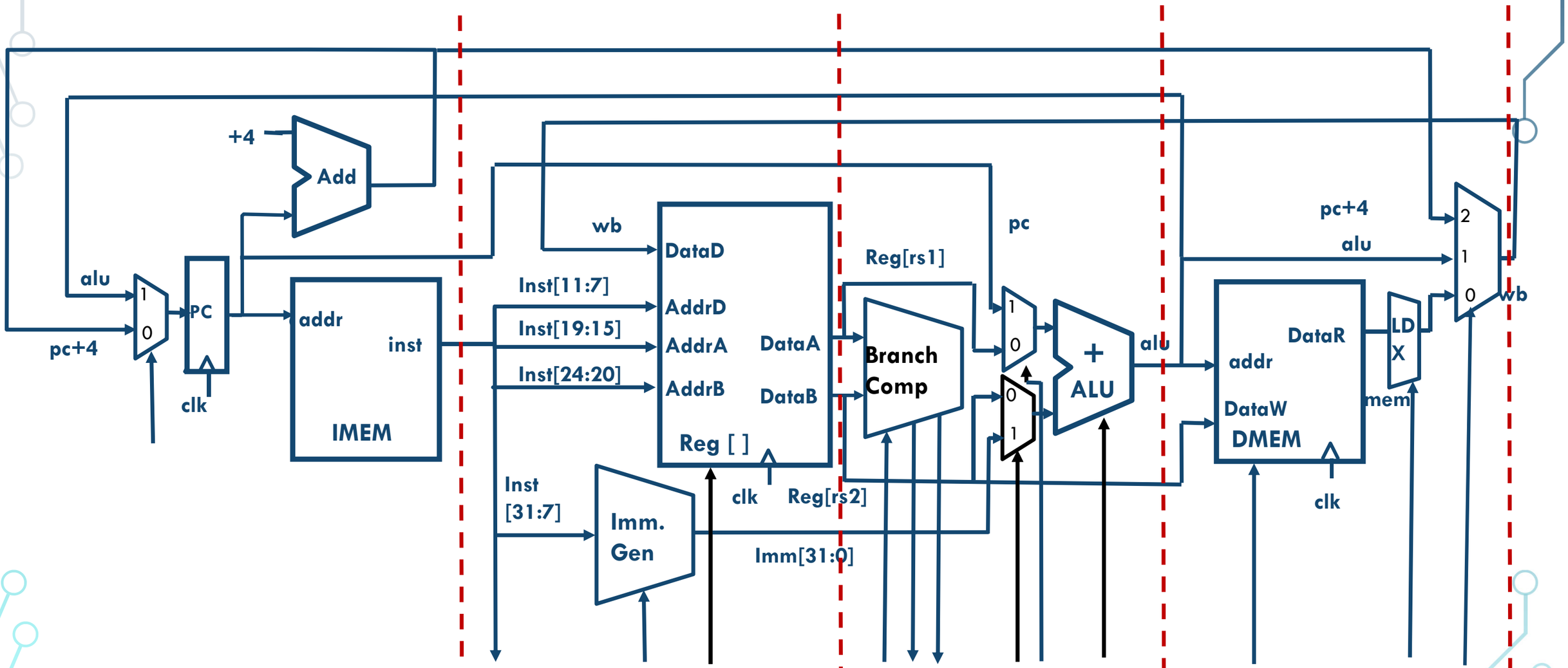
R[rd] = R[rs1]+imm



1) $t_{clk\text{-}q} + t_{Add} + t_{IMEM} + t_{Reg} + t_{BComp} + t_{ALU} + t_{DMEM} + t_{mux} + t_{Setup}$
2) $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 2t_{mux} + t_{Setup}$
3) $t_{clk\text{-}q} + t_{IMEM} + \max\{t_{Reg}, t_{Imm}\} + t_{ALU} + 3t_{mux} + t_{DMEM} + t_{Setup}$
4) None of the above

Pipelining

Complete RV32I Datapath with Control



Instruction Fetch (F)

Instruction Decode/ Register Read (D)

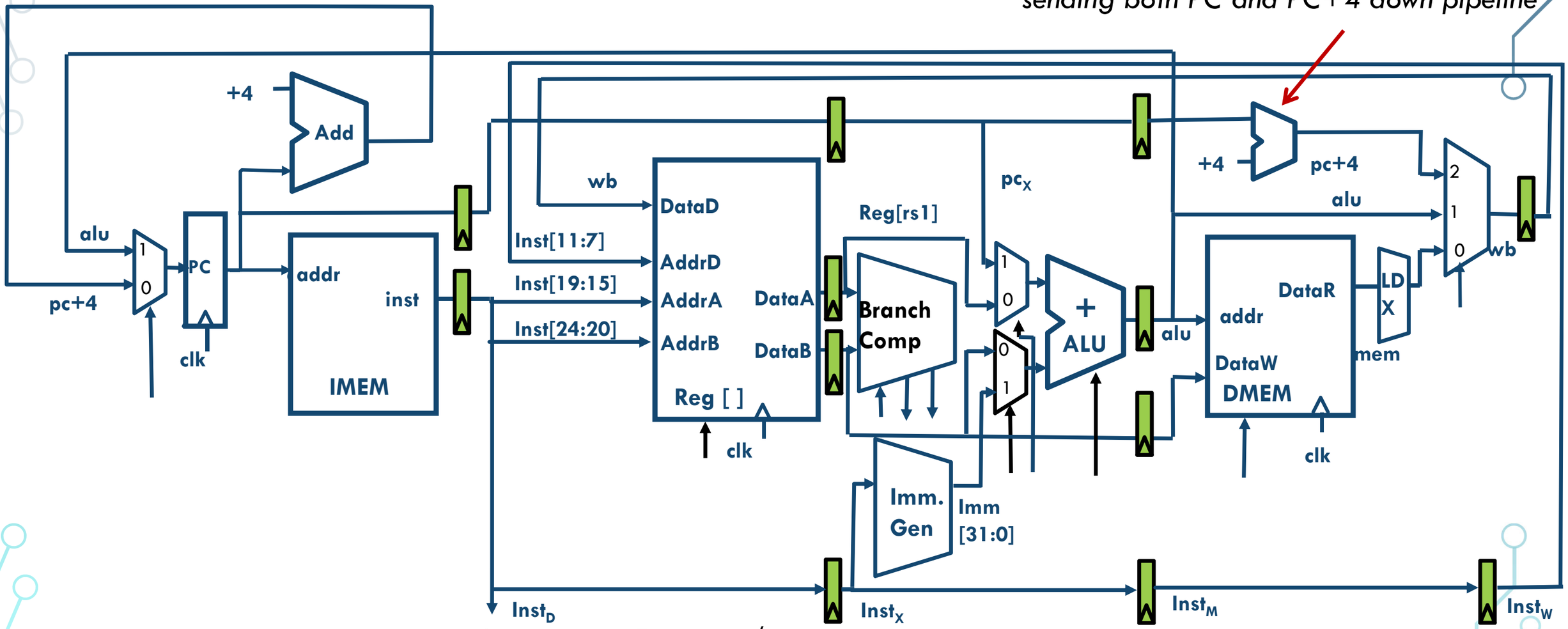
ALU Execute (X)

Memory Access (M)

Write Back (W)

Pipelining RV32I Datapath

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)

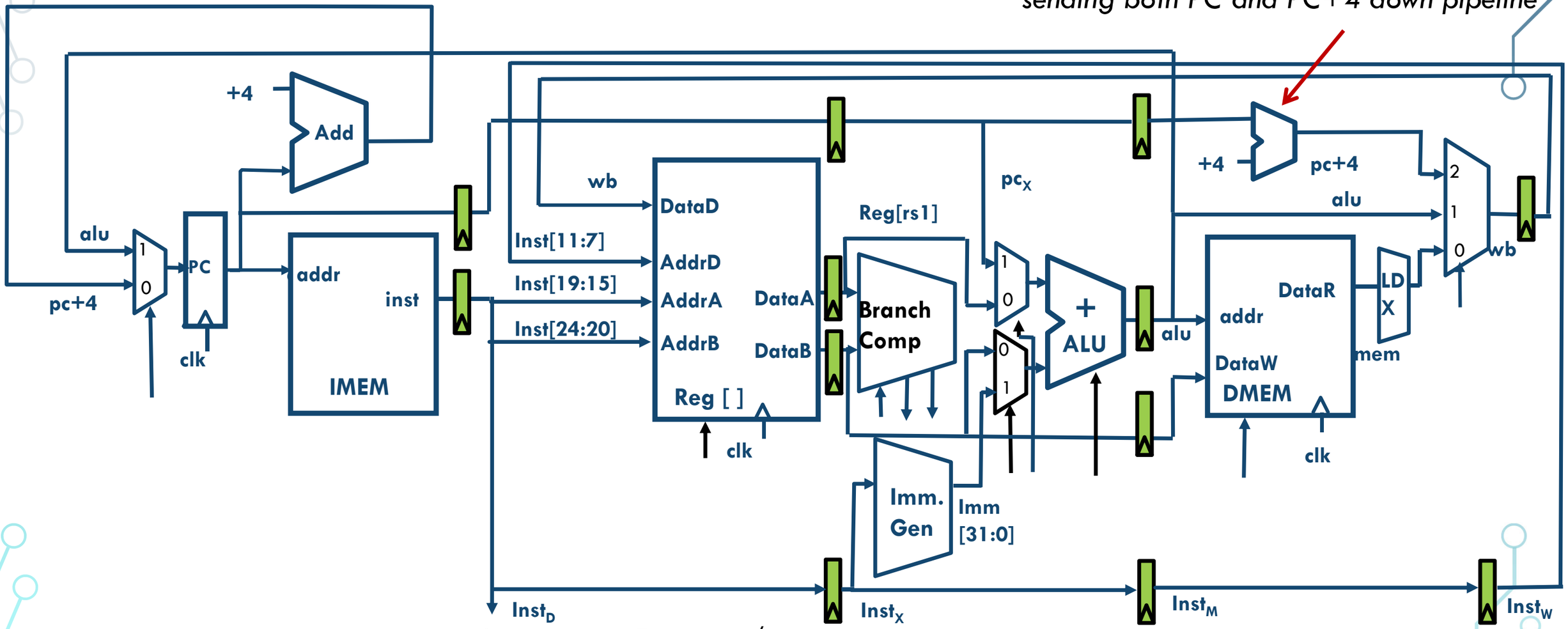
Memory Access (M)

Write Back (W)

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

Pipelining RV32I Datapath

Recalculate PC+4 in M stage to avoid sending both PC and PC+4 down pipeline



Instruction Fetch (F)

Instruction Decode/ Register Read (D)

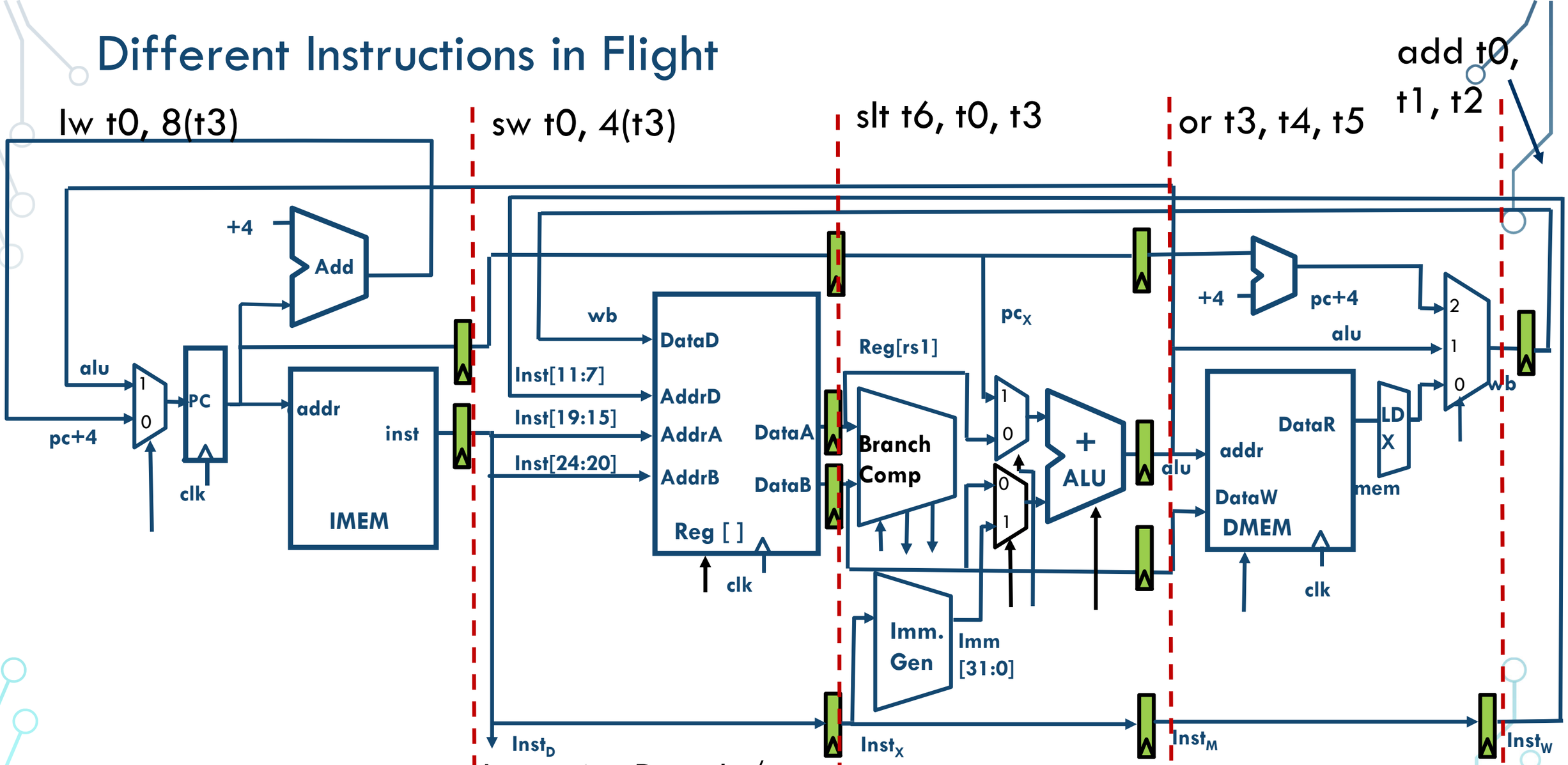
ALU Execute (X)

Memory Access (M)

Write Back (W)

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

Different Instructions in Flight



Instruction Fetch (F)

Instruction Decode/ Register Read (D)

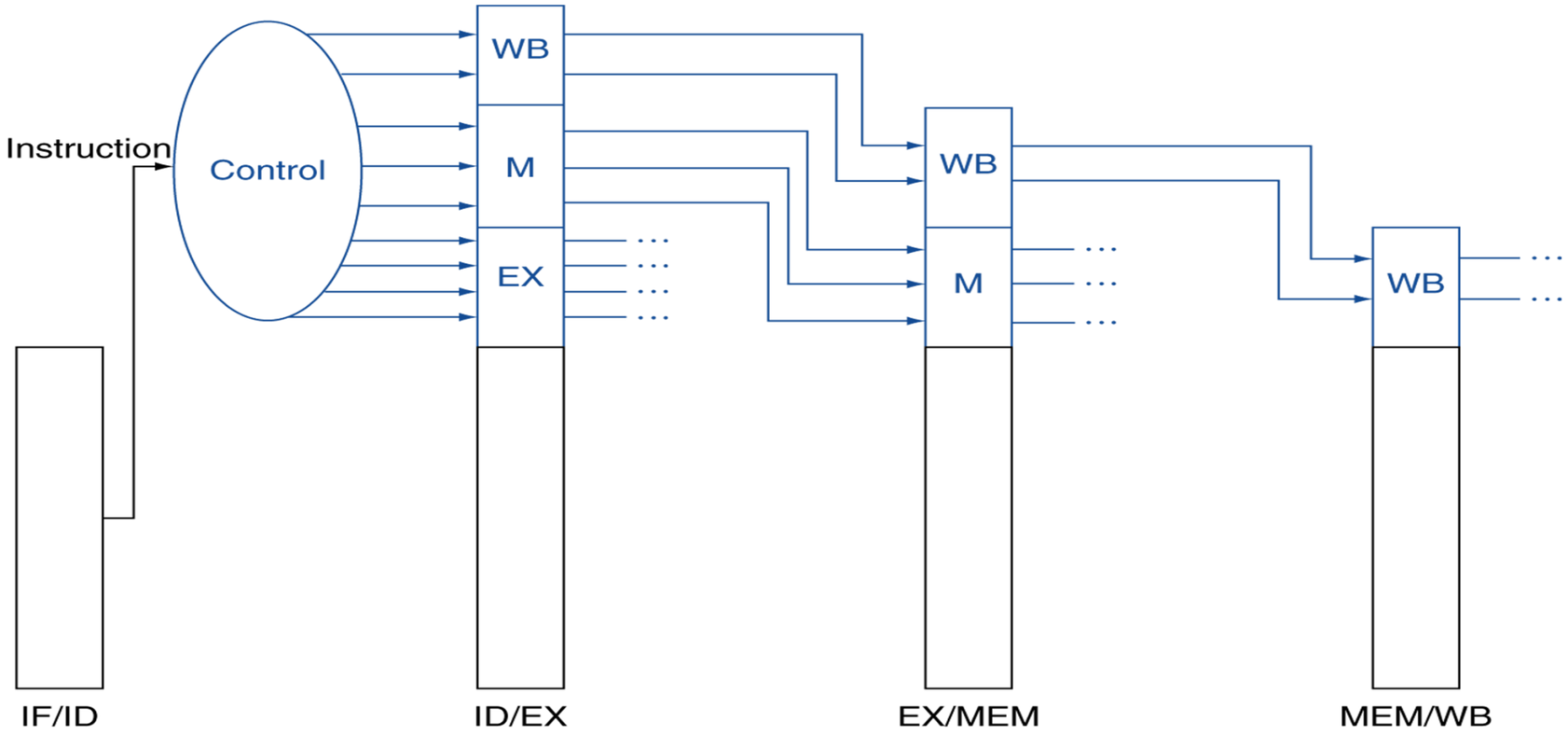
ALU Execute (X)

Memory Access (M)

Write Back (W)

Pipelined Control

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



Administrivia

- Midterm 1: October 2, in class
  - Covers material up to today
  - One double-sided page of hand-written notes
  - Review session this evening
- New lab this week
- New homework posted on Thursday
- Midterm 2: November 6

Pipeline Hazards

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

Regfile Structural Hazards

- 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

Structural Hazard: Memory Access

add t0, t1, t2

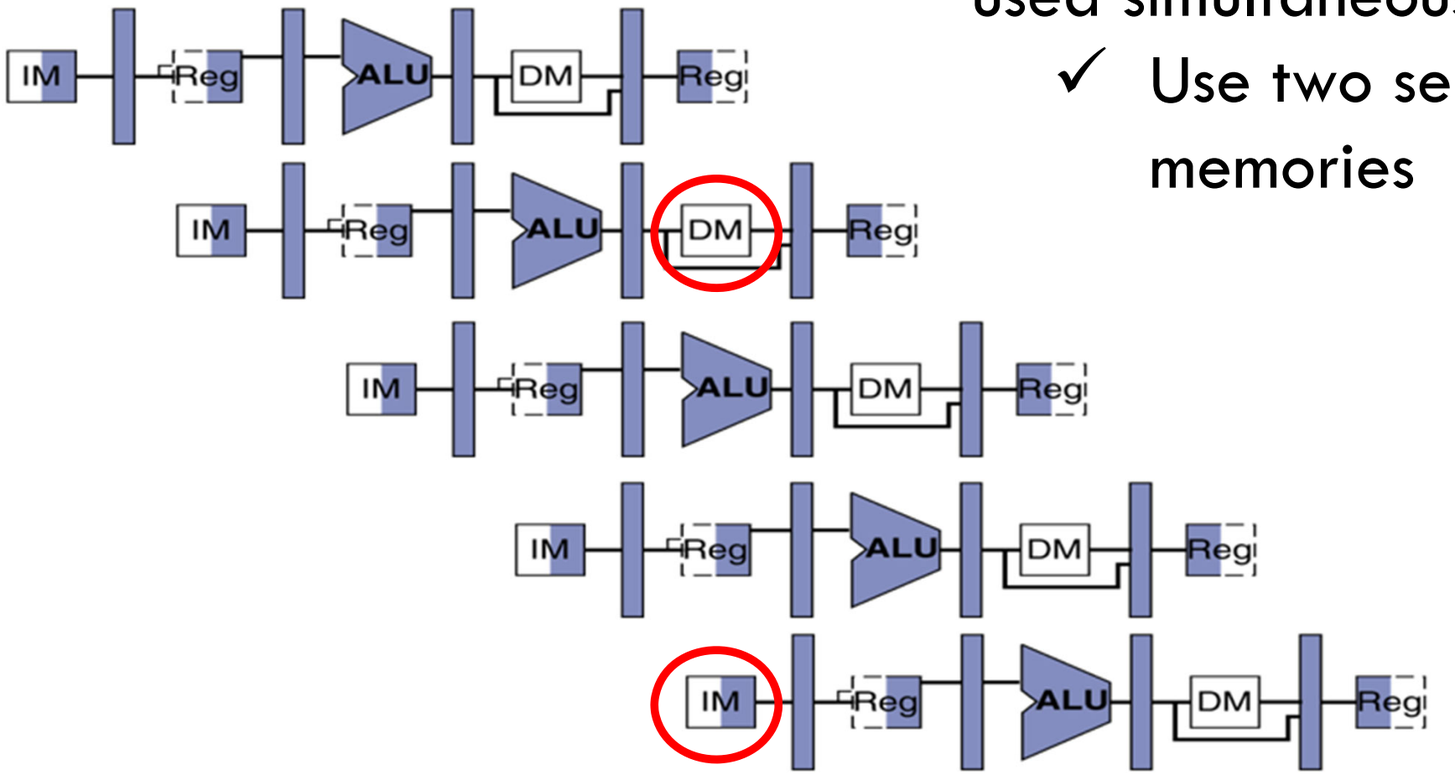
or t3, t4, t5

slt t6, t0, t3

sw t0, 4(t3)

lw t0, 8(t3)

instruction sequence



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

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

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

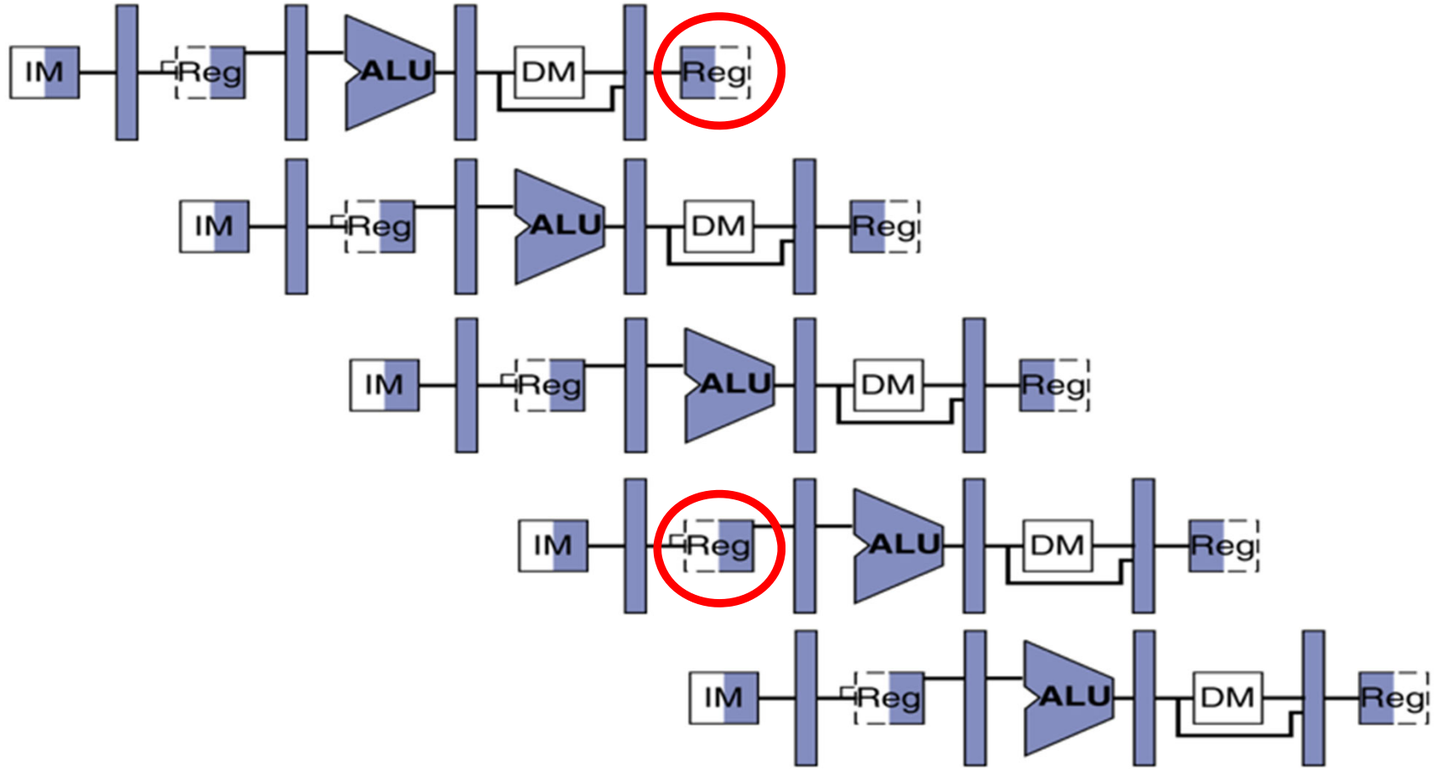
or t3, t4, t5

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

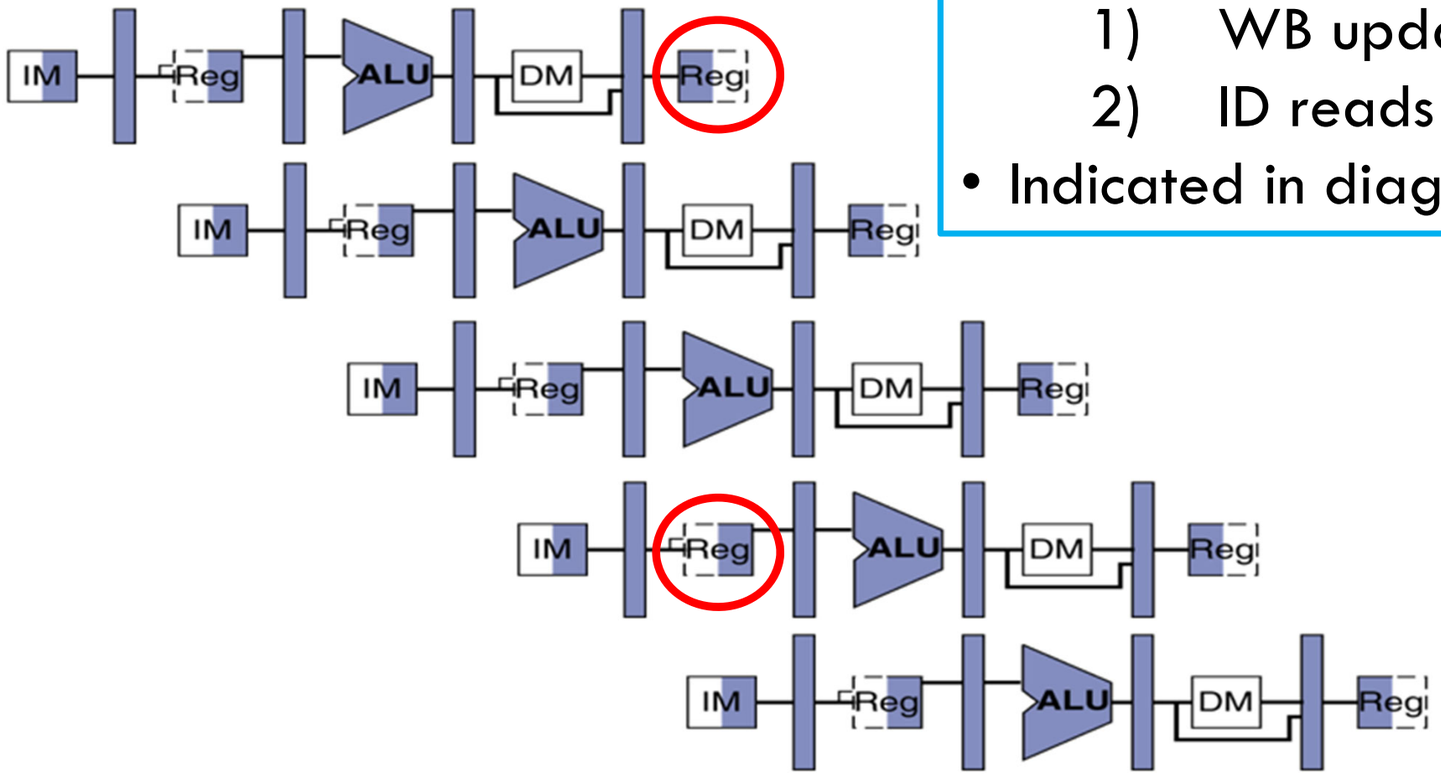
add t0, t1, t2

or t3, t4, t5

slt t6, t0, t3

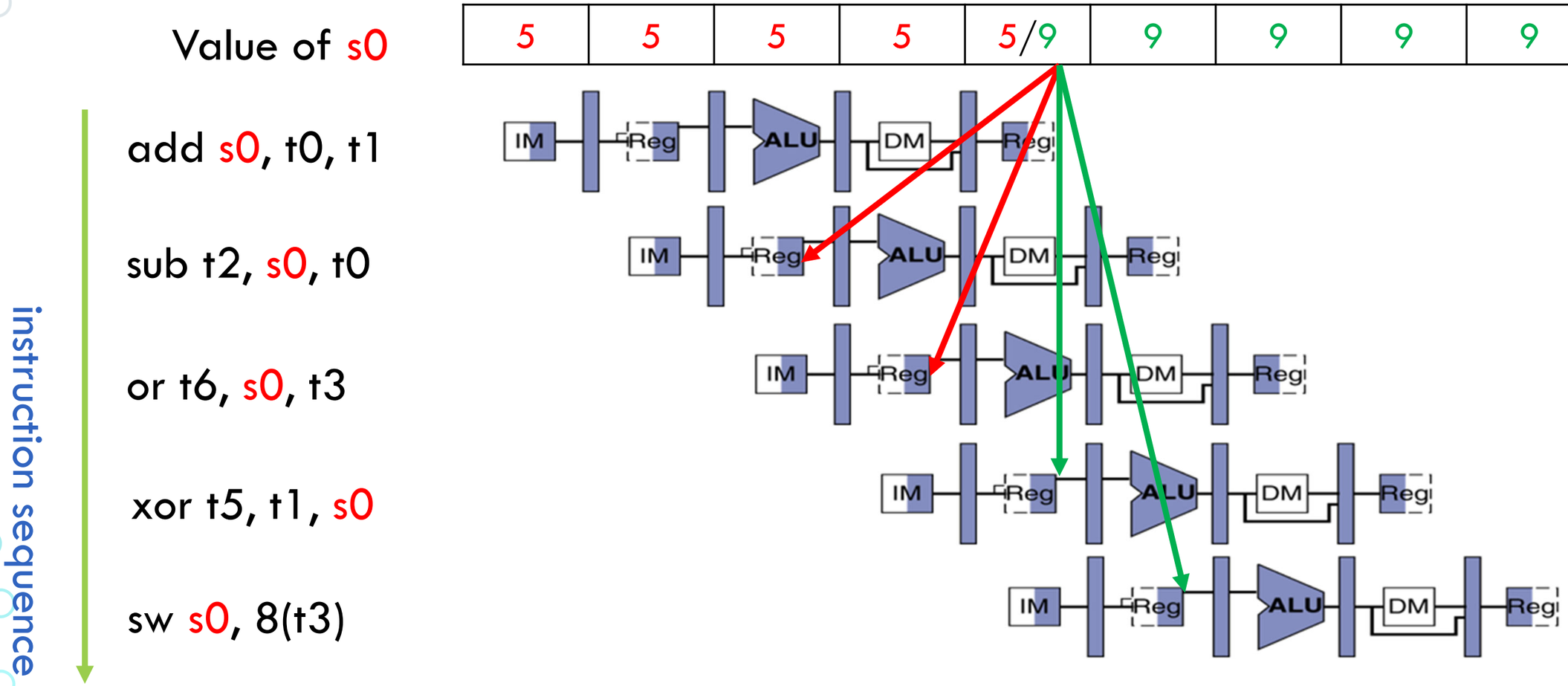
sw t0, 4(t3)

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.

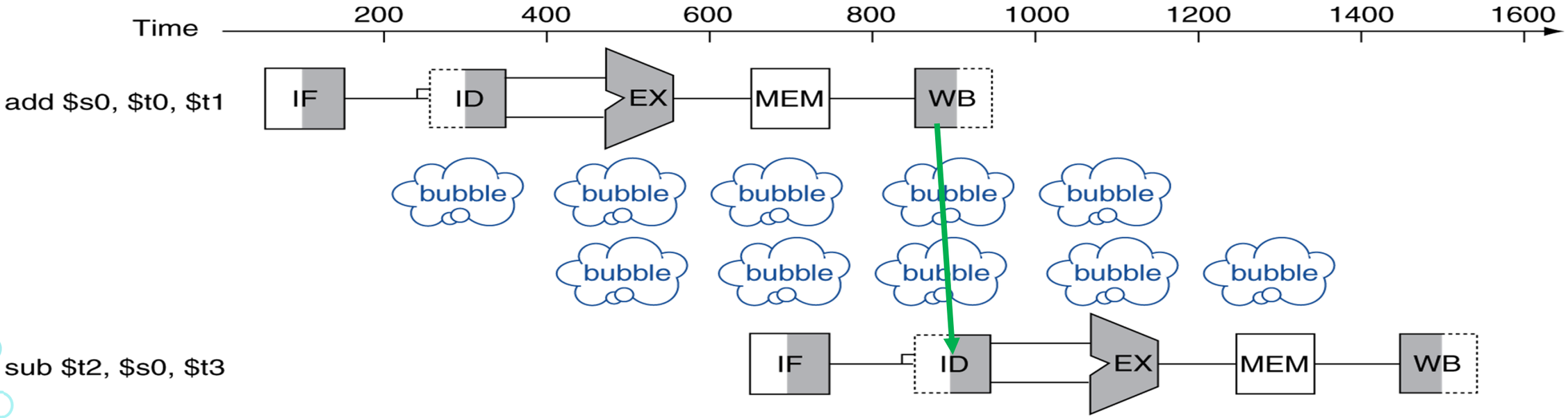
Data Hazard: ALU Result



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

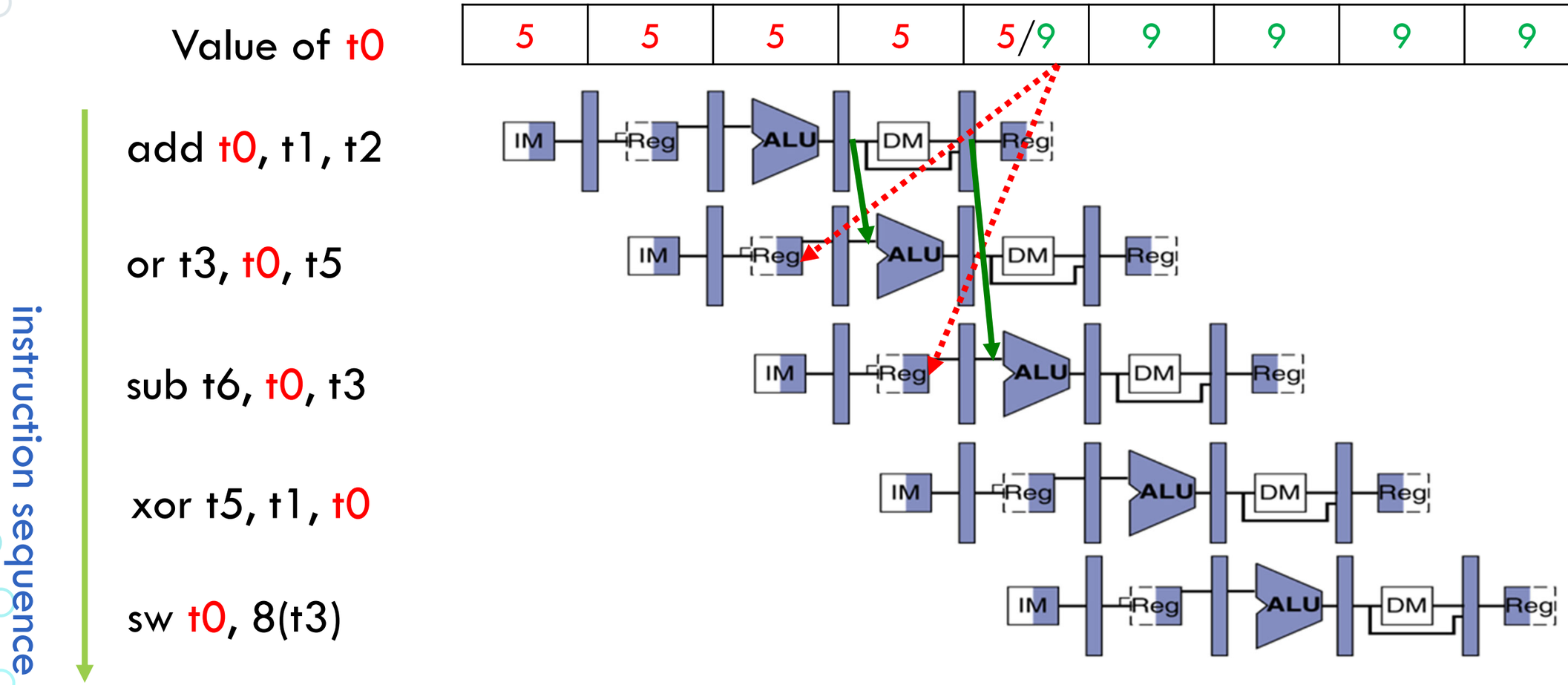


- Bubble:
  - effectively NOP: affected pipeline stages do "nothing"

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
  - Requires knowledge of the pipeline structure

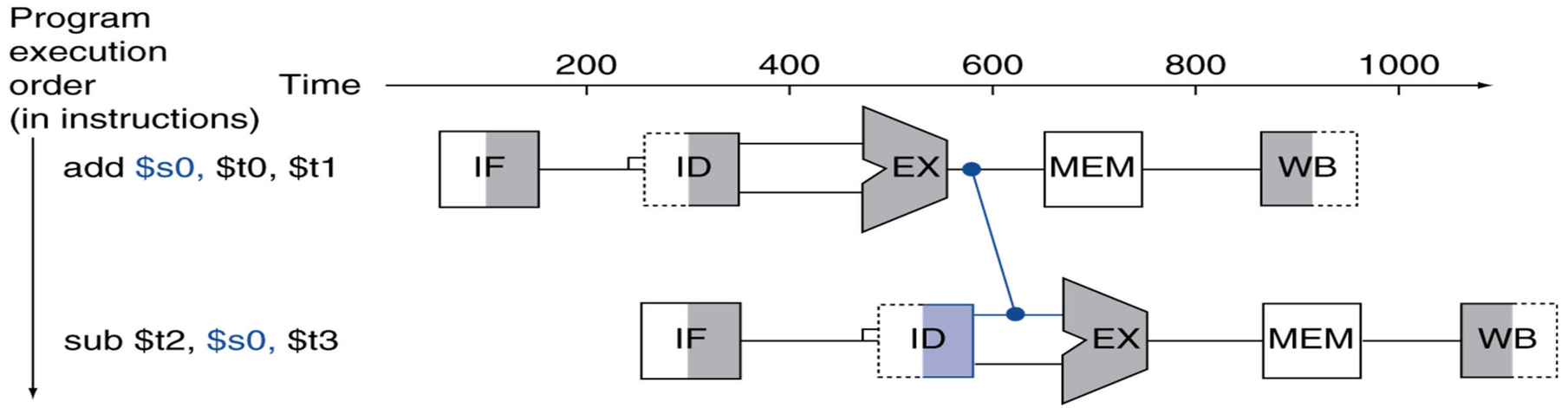
Solution 2: Forwarding



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

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



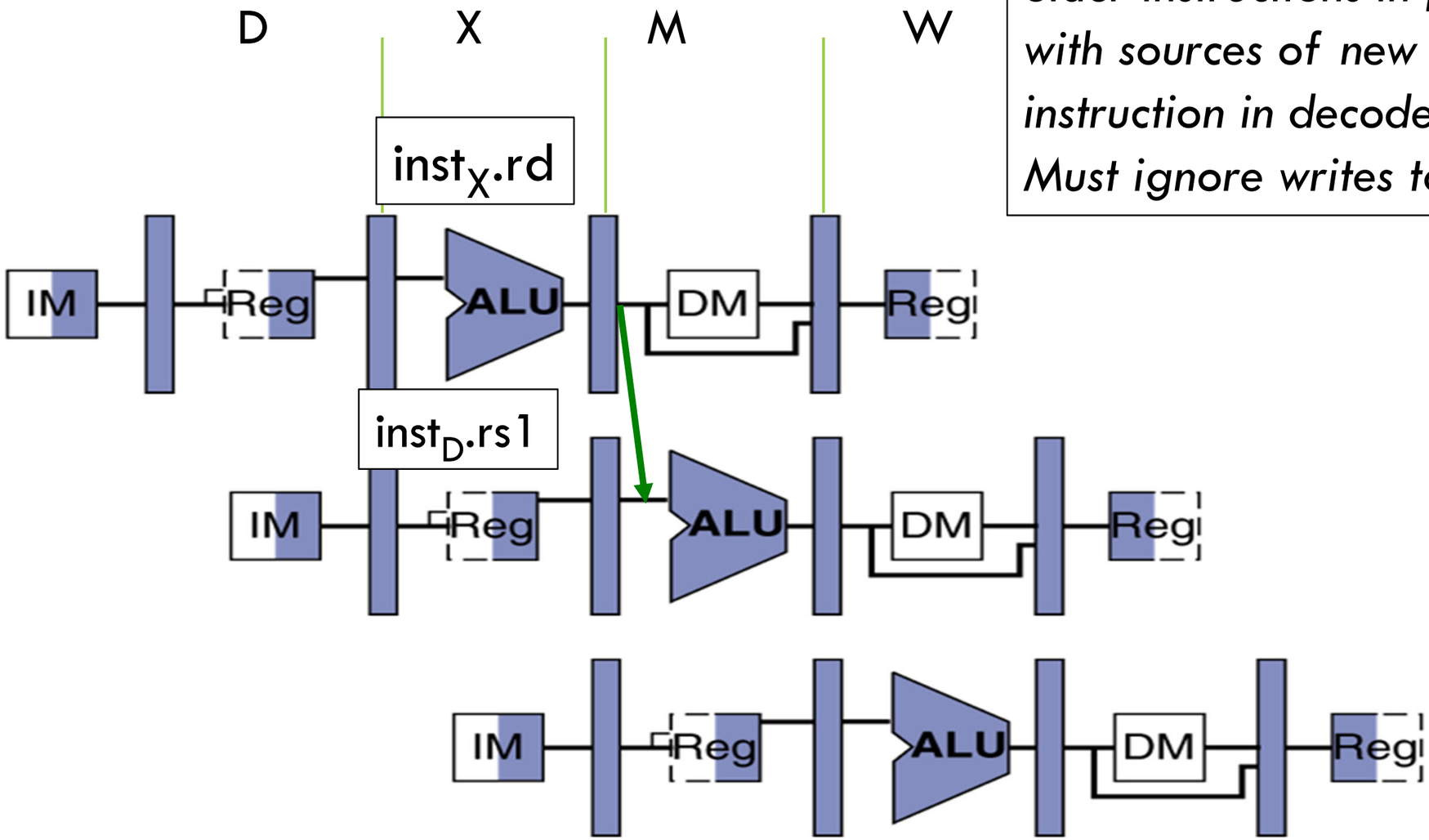
Detect Need for Forwarding (example)

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

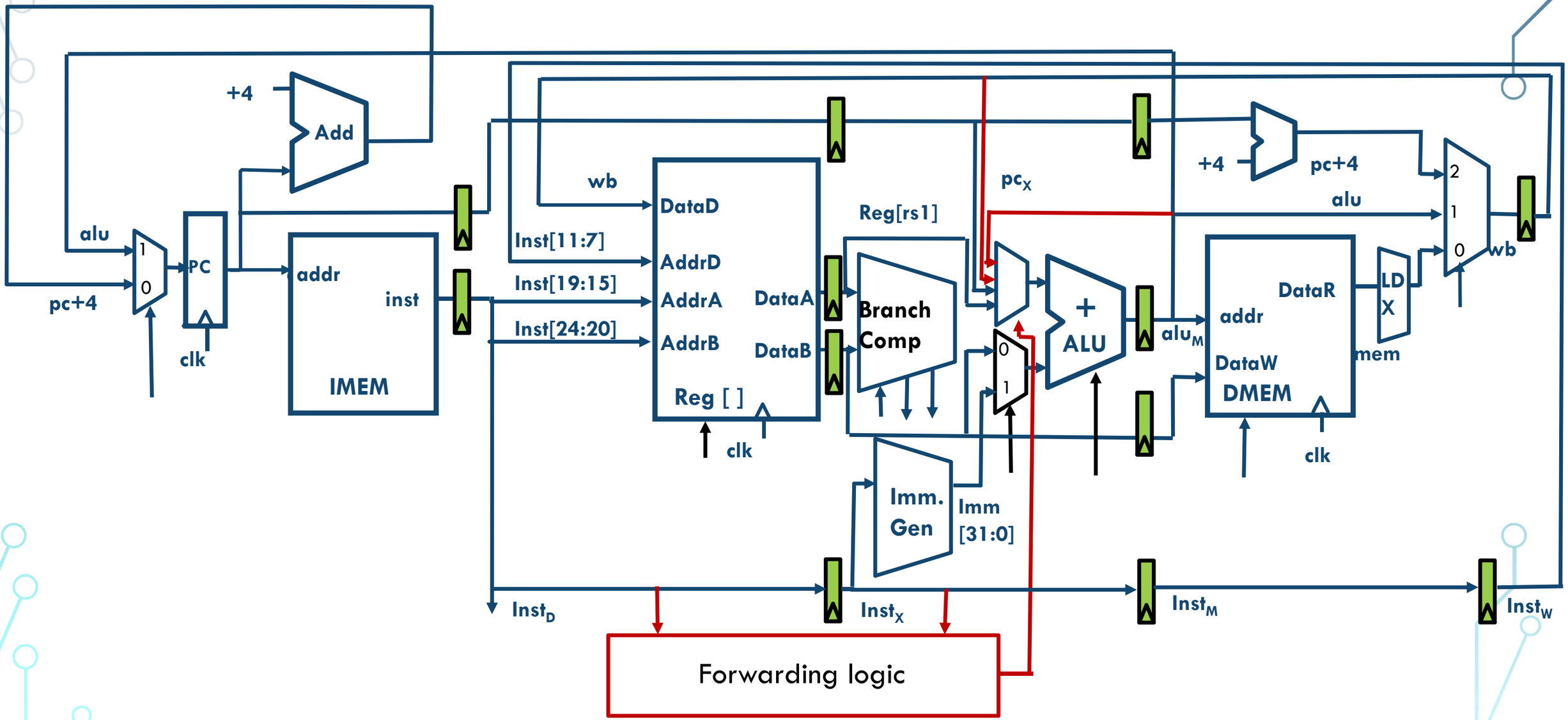
add t0, t1, t2

or t3, t0, t5

sub t6, t0, t3



Forwarding Path



Control Hazards

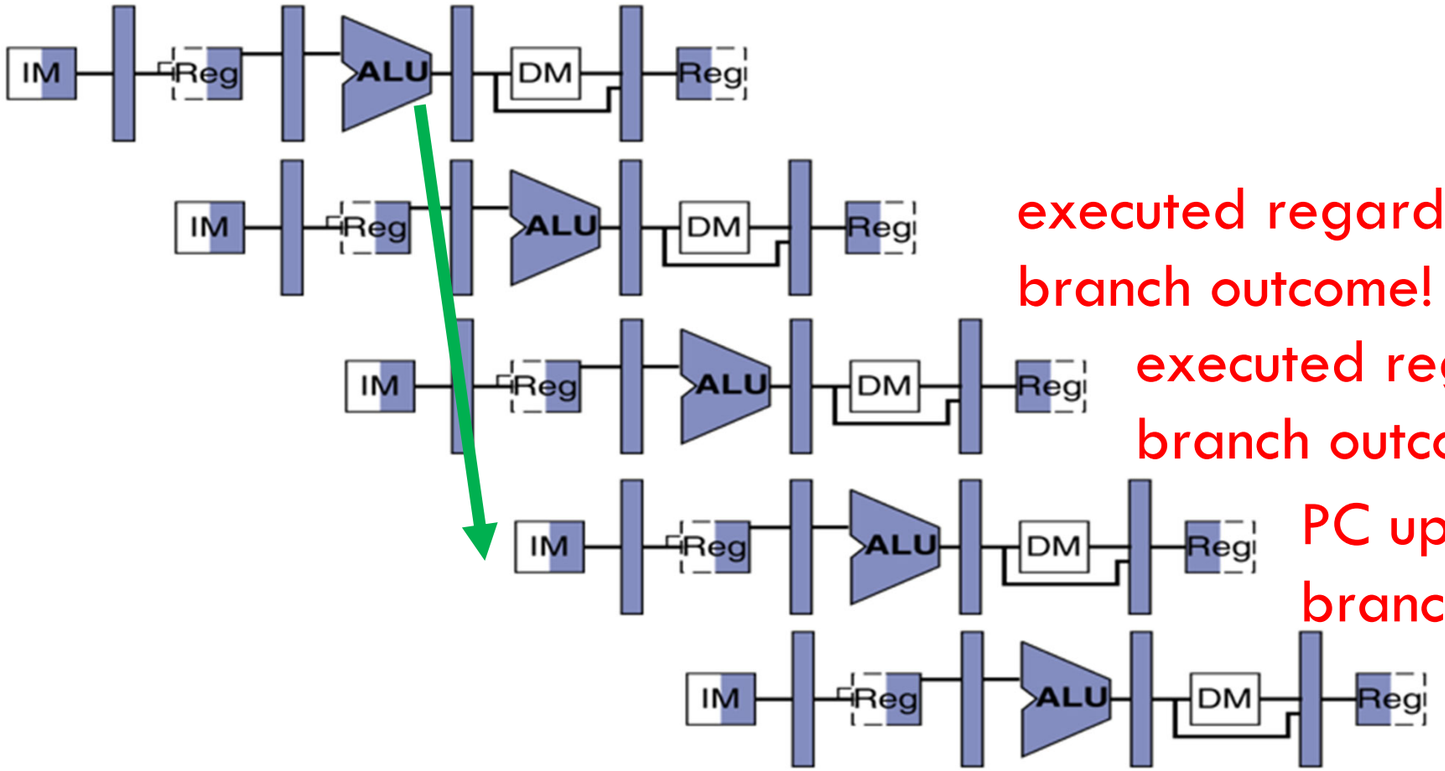
beq t0, t1, label

sub t2, s0, t5

or t6, s0, t3

xor t5, t1, s0

sw s0, 8(t3)



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

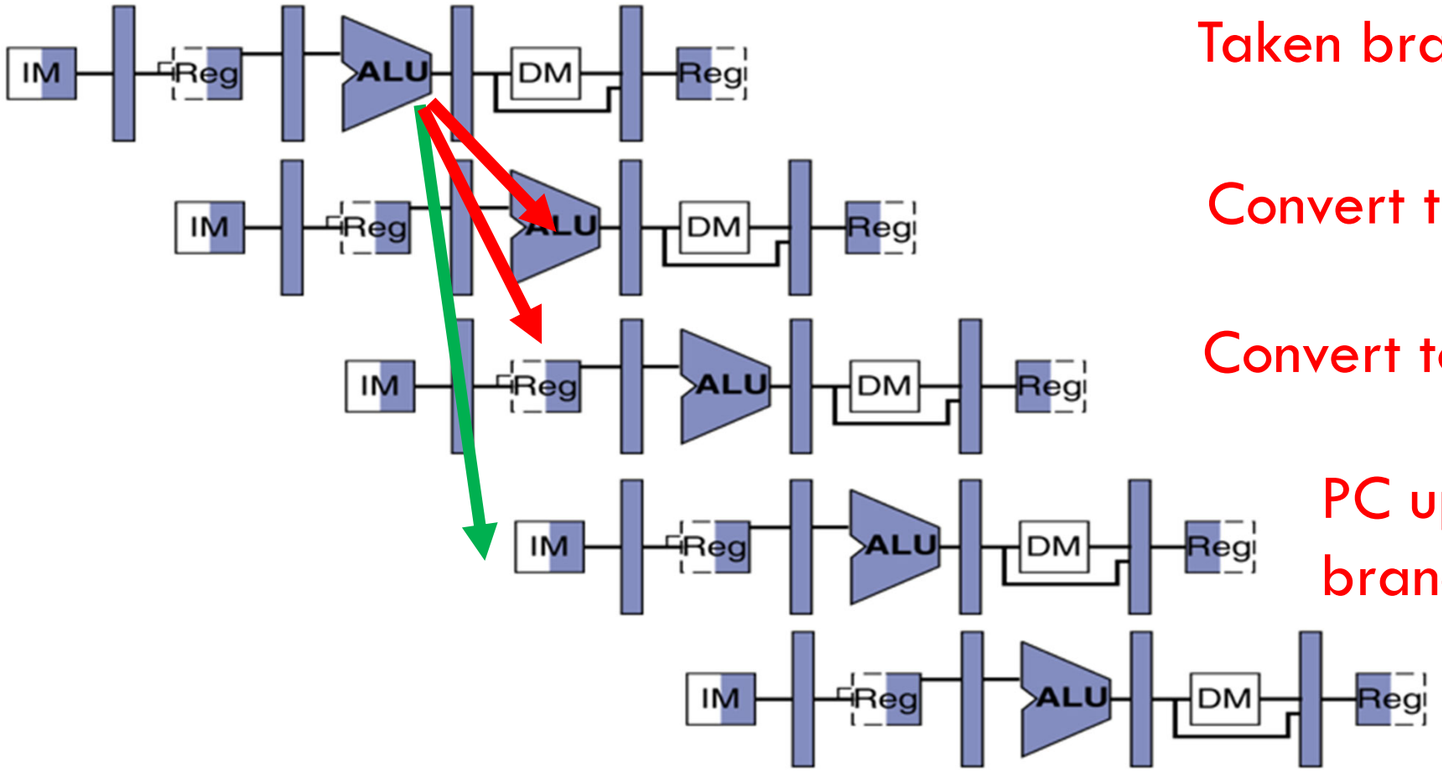
Kill Instructions after Branch if Taken

beq t0, t1, label

sub t2, s0, t5

or t6, s0, t3

label: xxxxxx



Taken branch

Convert to NOP

Convert to NOP

PC updated reflecting branch outcome

Reducing Branch Penalties

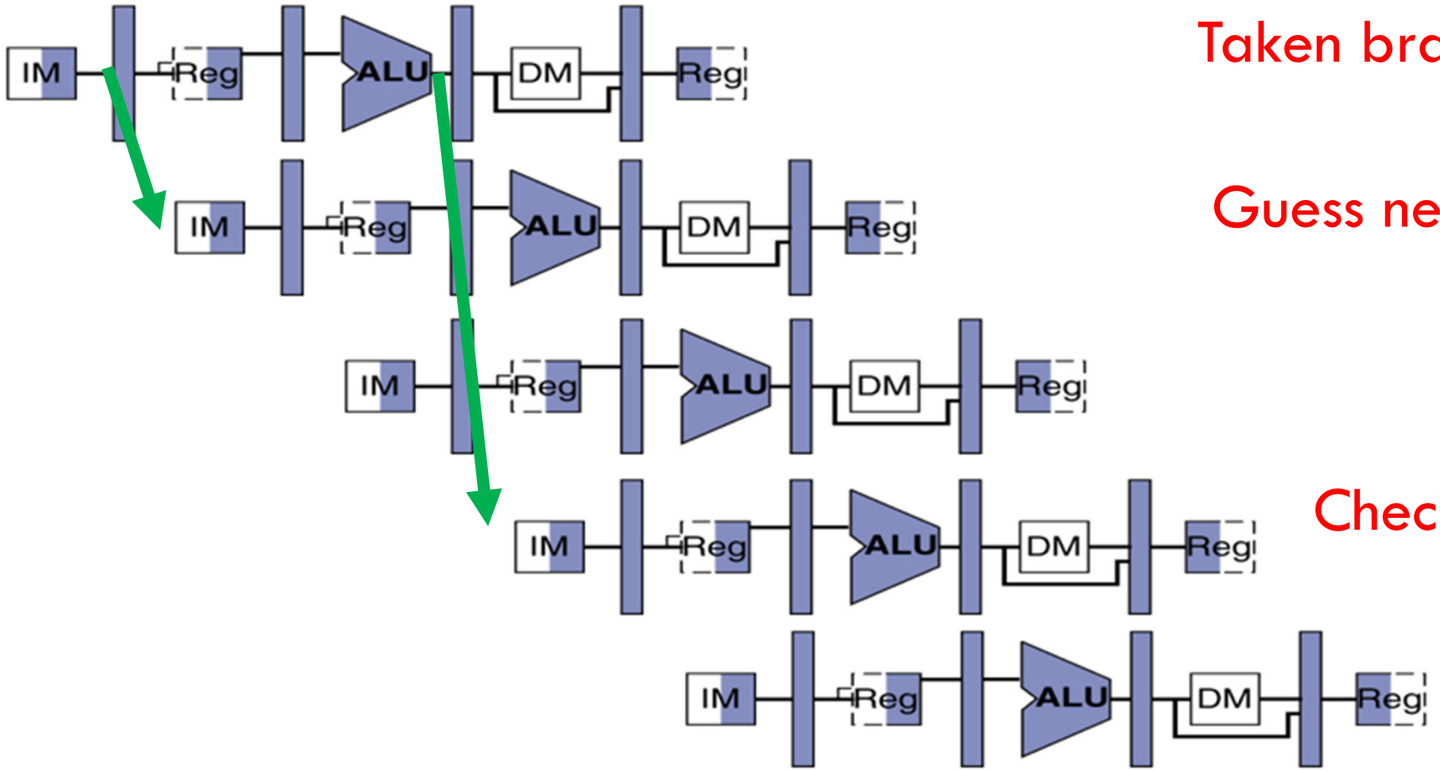
- Every taken branch in simple pipeline costs 2 dead cycles
- To improve performance, use “branch prediction” to guess which way branch will go earlier in pipeline
- Only flush pipeline if branch prediction was incorrect

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
- Control can be implemented as ROM or combinational logic
- Structural, data and control hazards need to be understood and addressed