CSE 520
Computer Architecture II

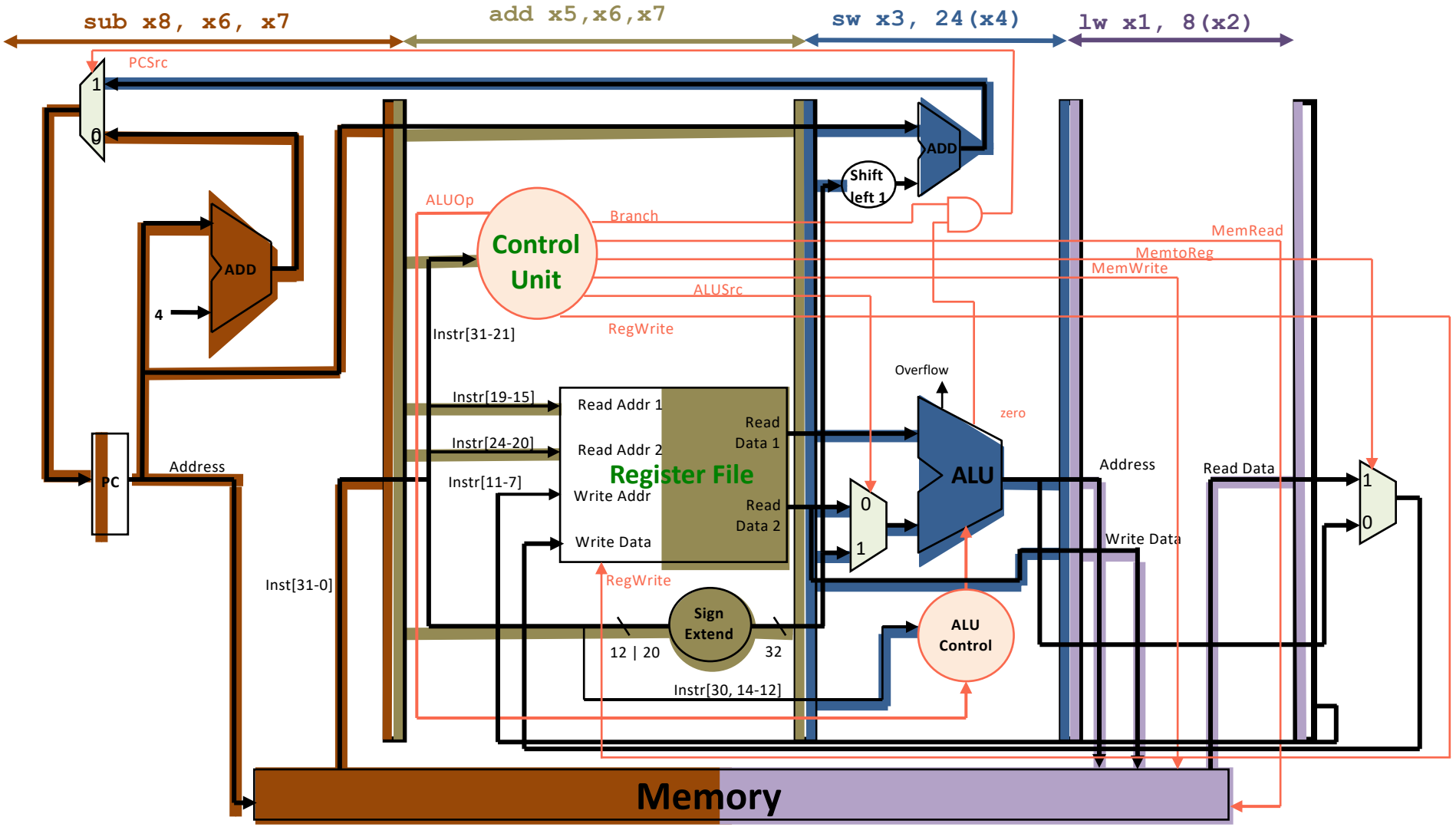
CPU: Hazard Resolution

Prof. Michel A. Kinsy

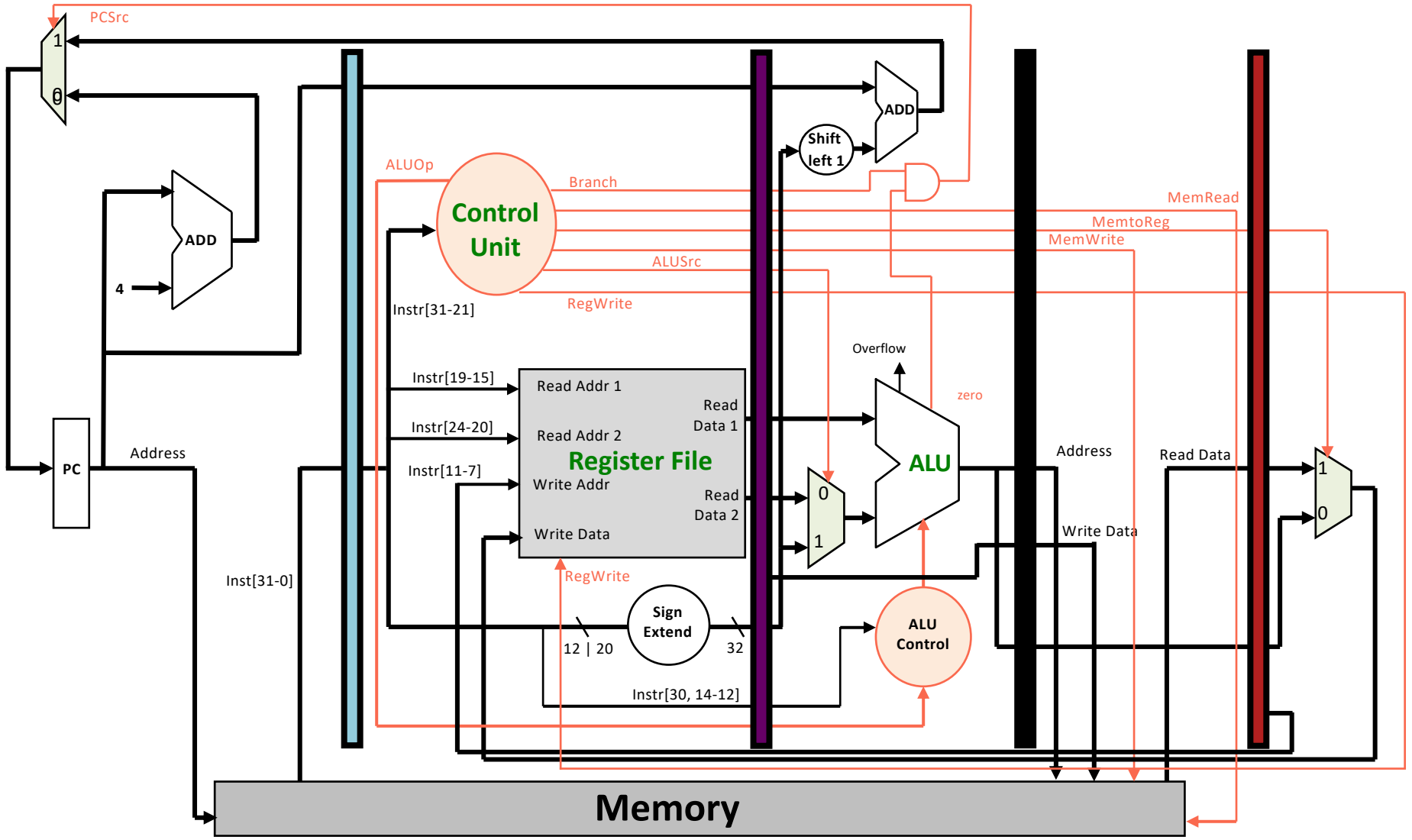
Instruction Interactions

- An instruction in the pipeline may need a resource being used by another instruction in the pipeline
  - Structural hazard
- An instruction may depend on something produced by an earlier instruction
  - Dependence may be for a data calculation
    - Data hazard
  - Dependence may be for calculating the next address
    - Control hazard (branches, interrupts)

Structural Hazard



Multi-Stage RISC-V CPU



Stalled Stages and Pipeline

	time t0	t1	t2	t3	t4	t5	t6	t7 . .
(I_1) r1 ← (r0) + 10	IF_1	ID_1	EX_1	MA_1	WB_1			
(I_2) r4 ← (r1) + 17		IF_2	ID_2	ID_2	ID_2	ID_2	EX_2	MA_2
(I_3)			IF_3	IF_3	IF_3	IF_3	ID_3	EX_3
(I_4)							IF_4	ID_4
(I_5)								IF_5

stalled stages

Resource Usage

	time t0	t1	t2	t3	t4	t5	t6	t7	. . .
IF	I_1	I_2	I_3	I_3	I_3	I_3	I_4	I_5	
ID		I_1	I_2	I_2	I_2	I_2	I_3	I_4	
EX			I_1	nop	nop	nop	I_2	I_3	
MA				I_1	nop	nop	nop	I_2	
WB					I_1	nop	nop	nop	

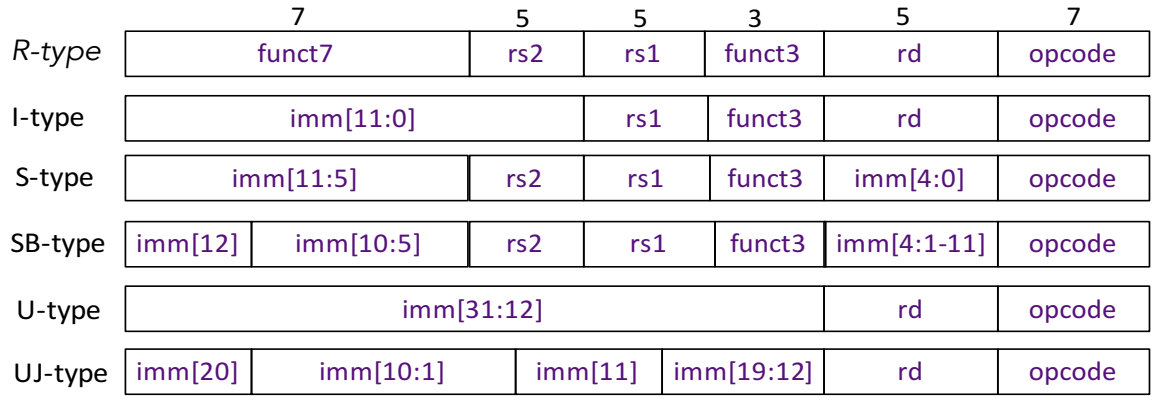
Resolving Data Hazards

- Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  - Interlocks
- Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  - Bypass

Resolving Data Hazards

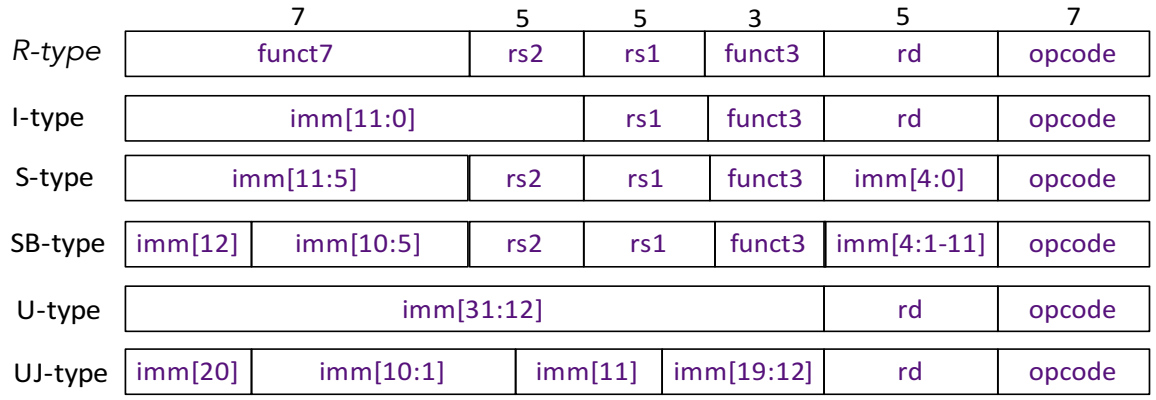
- Strategy 3: Speculate on the dependence
  - Two cases:
    - Guessed correctly
      - Do nothing
    - Guessed incorrectly
      - Kill and restart

Source and Destination Registers



		source(s)	*destination*
ALU	rd ← (rs1) [func3,func7] (rs2)	rs1, rs2	rd
ALUi	rd ← (rs1) [func3] I-imm	rs1	rd
	rd ← (rs1) [funct3, inst[30]] I-imm[4:0]	rs1	rd

Source and Destination Registers



		source(s)	*destination*
ALU	rd ← (rs1) [func3,func7] (rs2)	rs1, rs2	rd
ALUi	rd ← (rs1) [func3] I-imm	rs1	rd
	rd ← (rs1) [funct3, inst[30]] I-imm[4:0]	rs1	rd
LW	rd ← M [(rs1) + imm]	rs1	rd
SW	M [(rs1) + imm] ← (rs2)	rs1, rs2	
LUI	rd ← U-imm		rd
AUIPC	rd ← pc + U-imm		rd

Source and Destination Registers

		source(s)	*destination*
ALU	rd ← (rs1) [func3,func7] (rs2)	rs1, rs2	rd
ALUi	rd ← (rs1) [func3] I-imm	rs1	rd
	rd ← (rs1) [funct3, inst[30]] I-imm[4:0]	rs1	rd
LW	rd ← M [(rs1) + imm]	rs1	rd
SW	M [(rs1) + imm] ← (rs2)	rs1, rs2	
LUI	rd ← U-imm		rd
AUIPC	rd ← pc + U-imm		rd
JAL	rd ← pc + 4		rd
	pc ← pc + J-imm		
JALR	rd ← pc + 4	rs1	rd
	pc ← (rs1 + I-imm) & ~0x01		
BR	pc ←compare(funct3, rs1, rs2) ?	rs1, rs2	
	pc + B-imm : pc + 4		

Types of Data Hazards

- Consider executing a sequence of

  $r_k \leftarrow (r_i)$ op (r_j) type of instructions

Data-dependence

$r_3 \leftarrow (r_1)$ op (r_2) Read-after-Write
$r_5 \leftarrow (r_3)$ op (r_4) (RAW) hazard

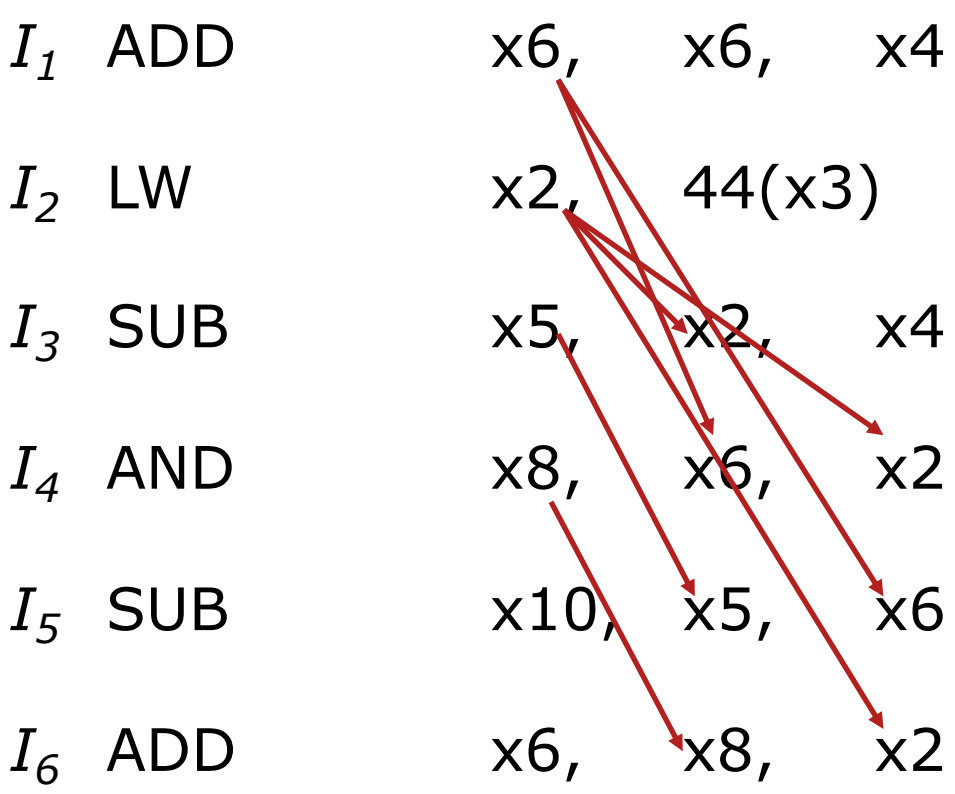
Anti-dependence

$r_3 \leftarrow (r_1)$ op (r_2) Write-after-Read
$r_1 \leftarrow (r_4)$ op (r_5) (WAR) hazard

Output-dependence

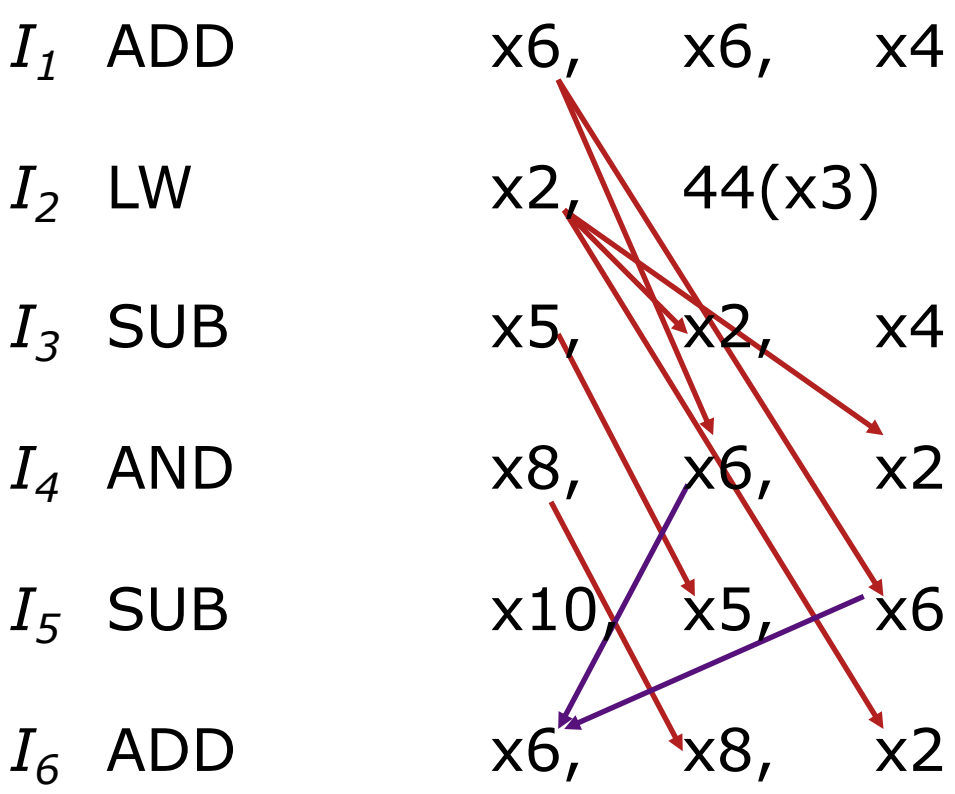
$r_3 \leftarrow (r_1)$ op (r_2) Write-after-Write
$r_3 \leftarrow (r_6)$ op (r_7) (WAW) hazard

Data Hazards: An Example



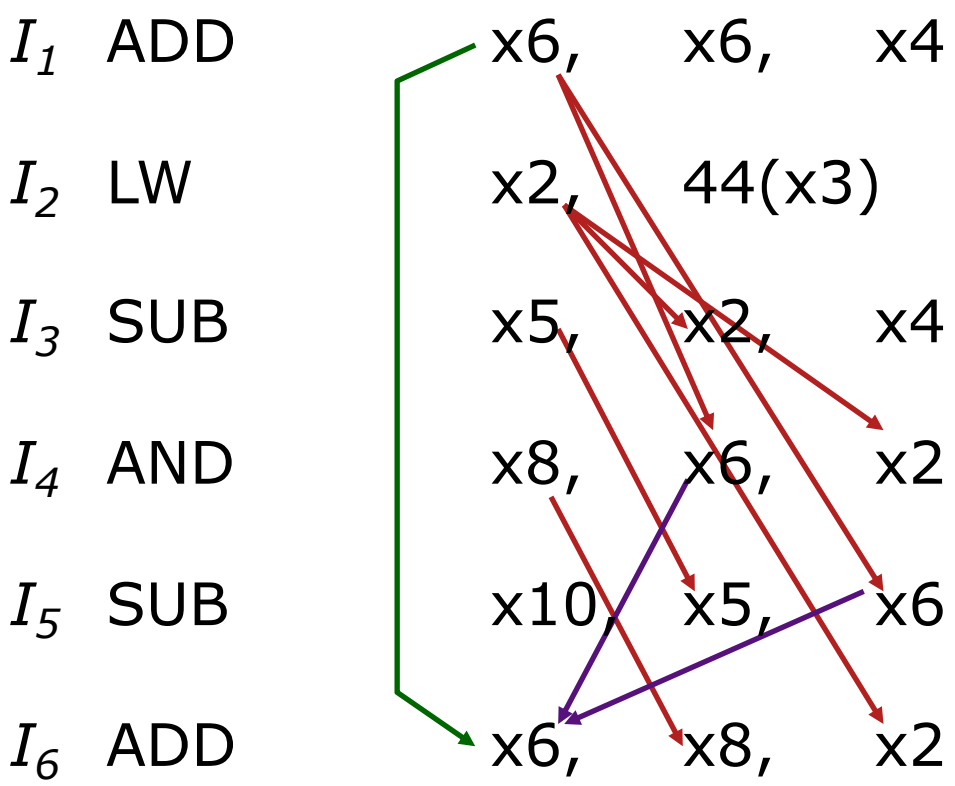
RAW Hazards

Data Hazards: An Example



RAW Hazards
WAR Hazards

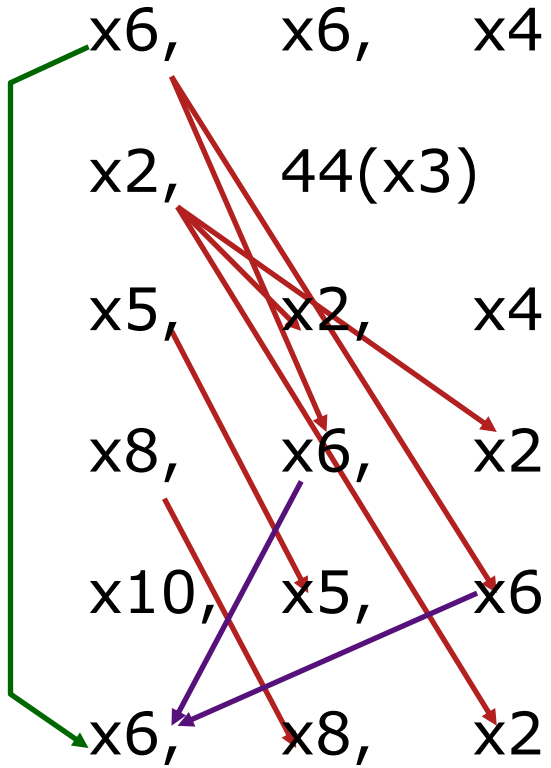
Data Hazards: An Example

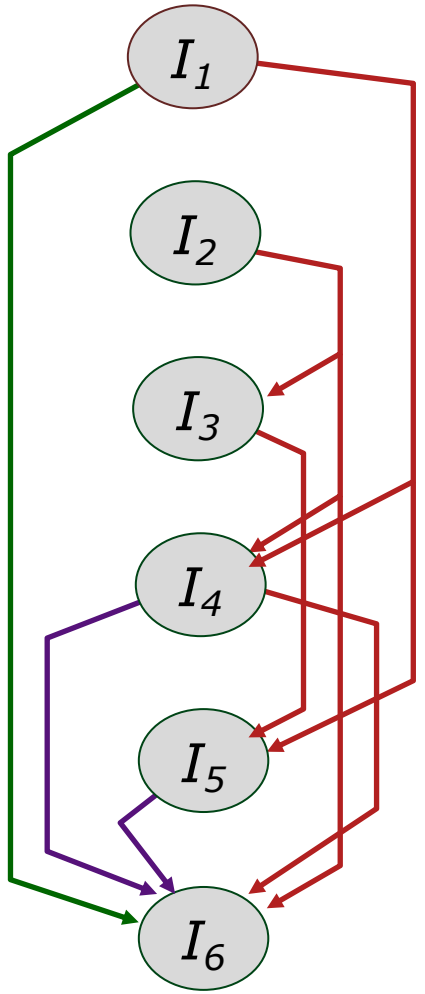


RAW Hazards
WAR Hazards
WAW Hazards

Data Hazards: An Example

I_1	ADD	x6,	x6,	x4
I_2	LW	x2,	44(x3)	
I_3	SUB	x5,	x2,	x4
I_4	AND	x8,	x6,	x2
I_5	SUB	x10,	x5,	x6
I_6	ADD	x6,	x8,	x2



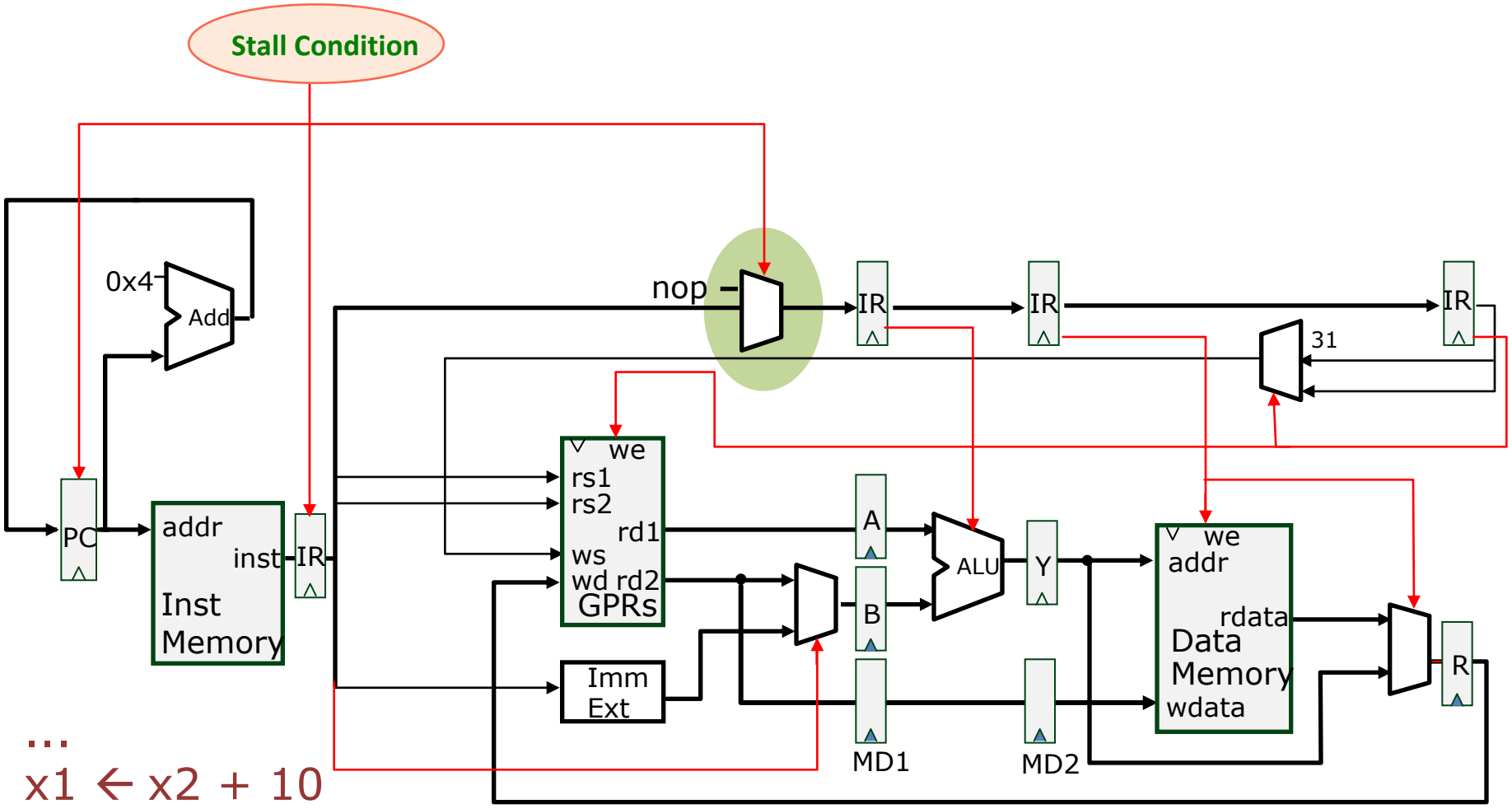


RAW Hazards
WAR Hazards
WAW Hazards

Resolving Data Hazards

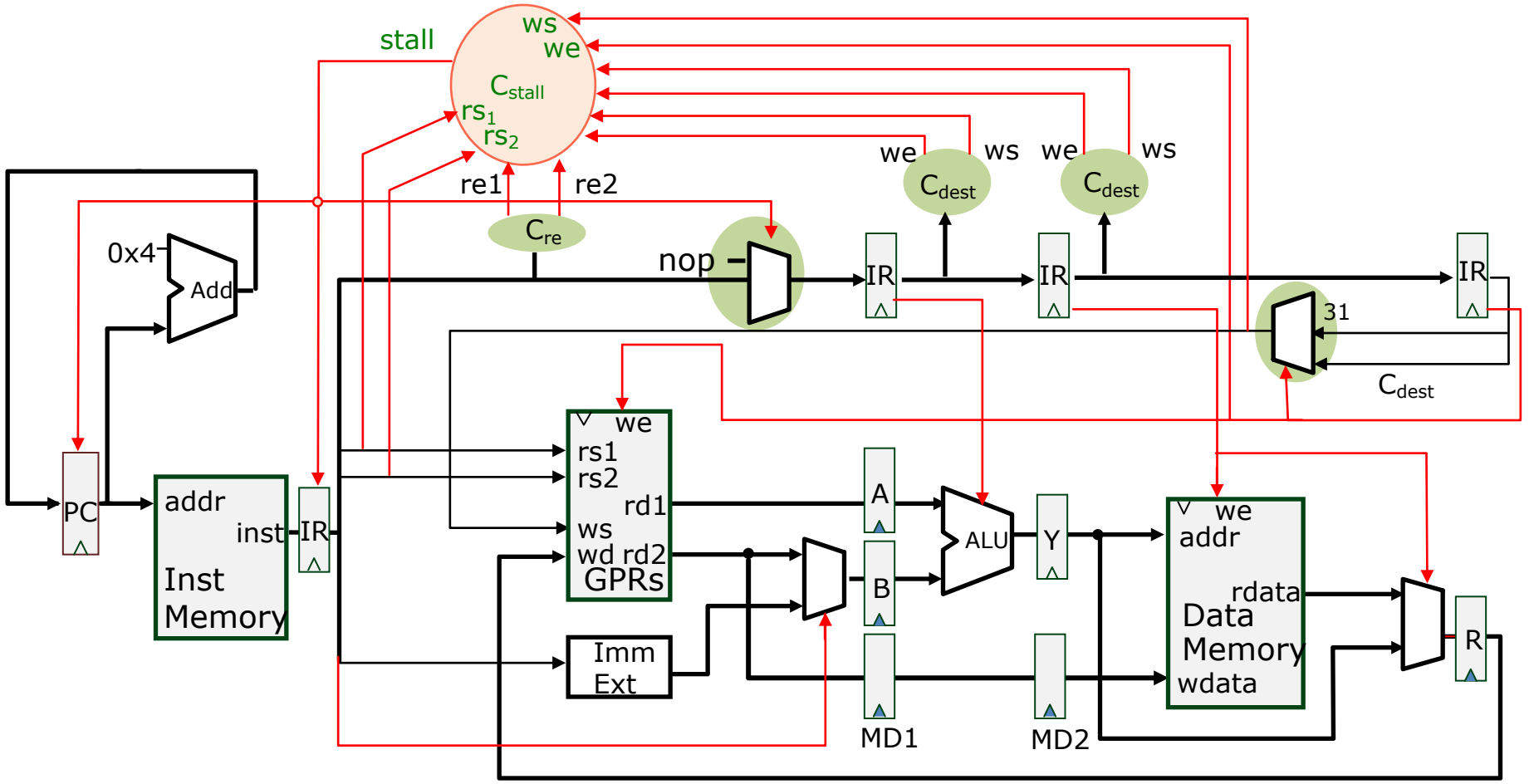
- Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  - Interlocks
- Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  - Bypass

Interlocks to resolve Data Hazards



...
x1 ← x2 + 10
x4 ← x1 + 17
...

Interlock Control Logic



Deriving the Stall Signal

C_{dest}

ws = *Case* opcode
- ALU , ALUi → rd
- LW, LUI → rd
- JAL, JALR, AUIPC → rd

we = *Case* opcode
- ALU, ALUi, LW, LUI → (ws != 0)
- JAL, JALR, AUIPC → on
- … → off

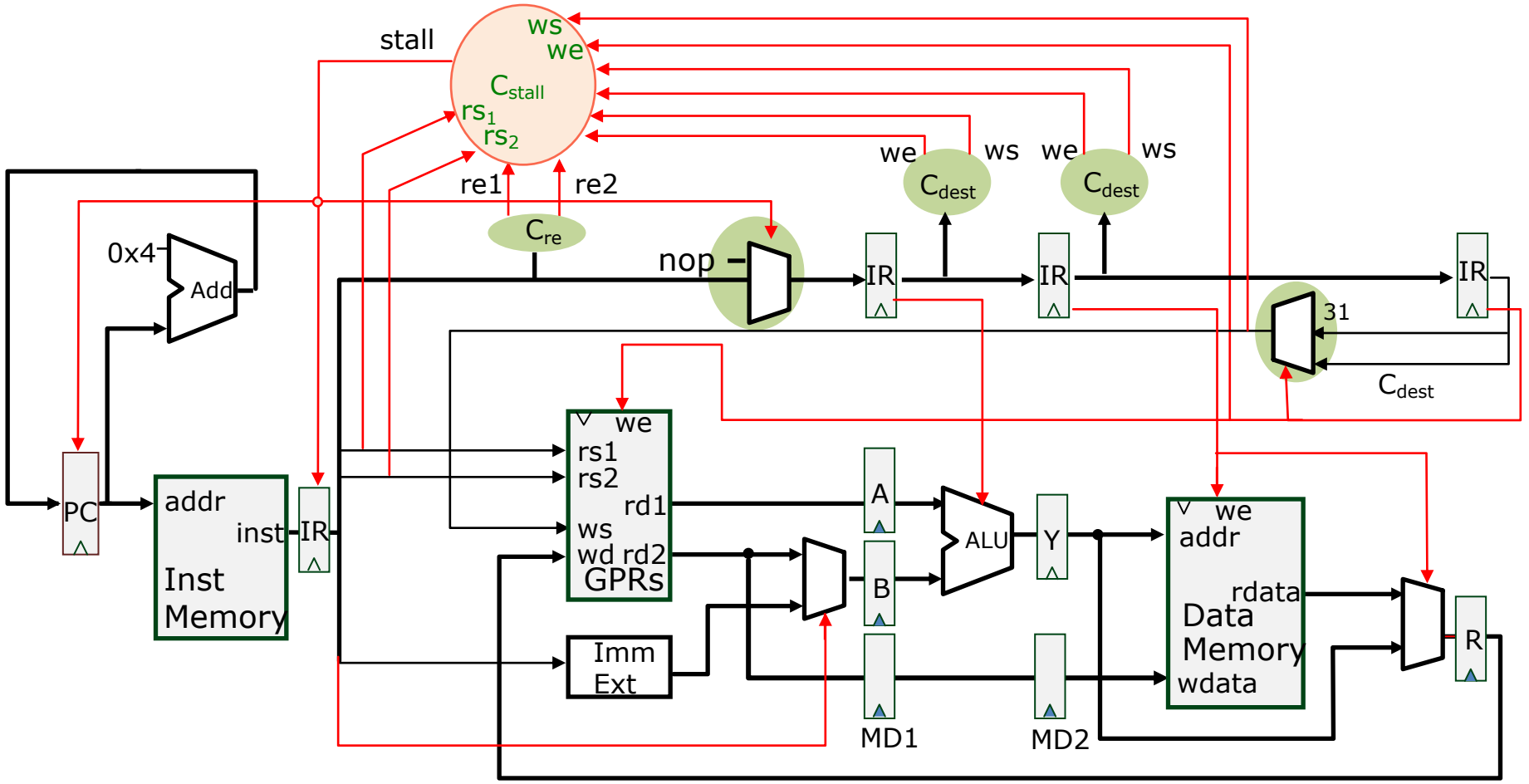
C_{re}

re1 = *Case* opcode
- ALU, ALUi, LW, SW, BR, JALR → on
- LUI, JAL, AUIPC → off

re2 = *Case* opcode
- ALU, SW, BR → on
- … → off

$$C_{stall} = ((rs1_D = ws_E).we_E + (rs1_D = ws_M).we_M + (rs1_D = ws_W).we_W) . re1_D + ((rs2_D = ws_E).we_E + (rs2_D = ws_M).we_M + (rs2_D = ws_W).we_W) . re2_D$$

Interlock Control Logic

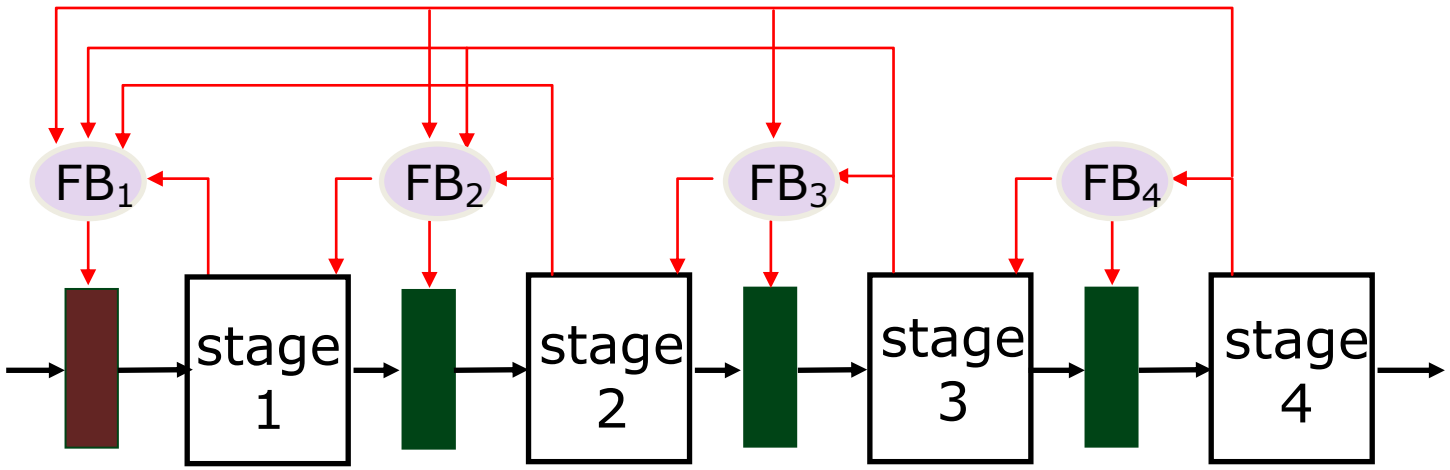


Resolving Data Hazards

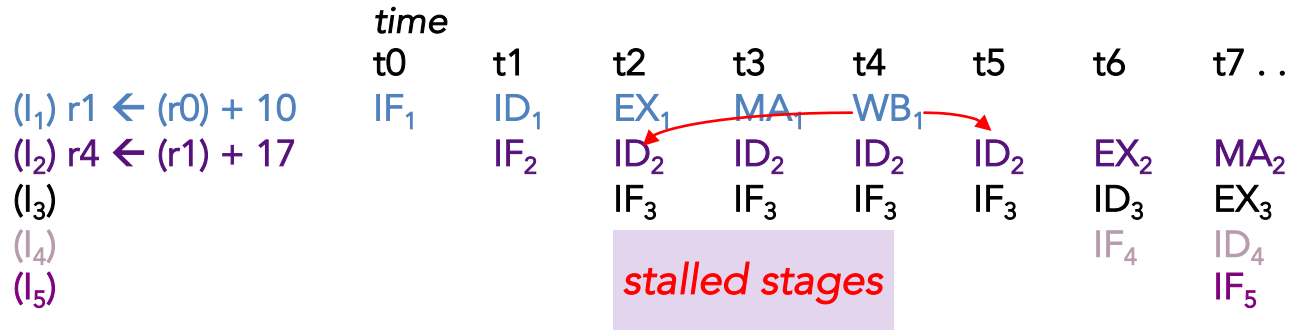
- Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  - Interlocks
- Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  - Bypass

Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can stall (or kill) instructions
- Controlling a pipeline in this manner works provided the instruction at stage i+1 can complete without any interference from instructions in stages 1 to i
  - Otherwise deadlocks may occur

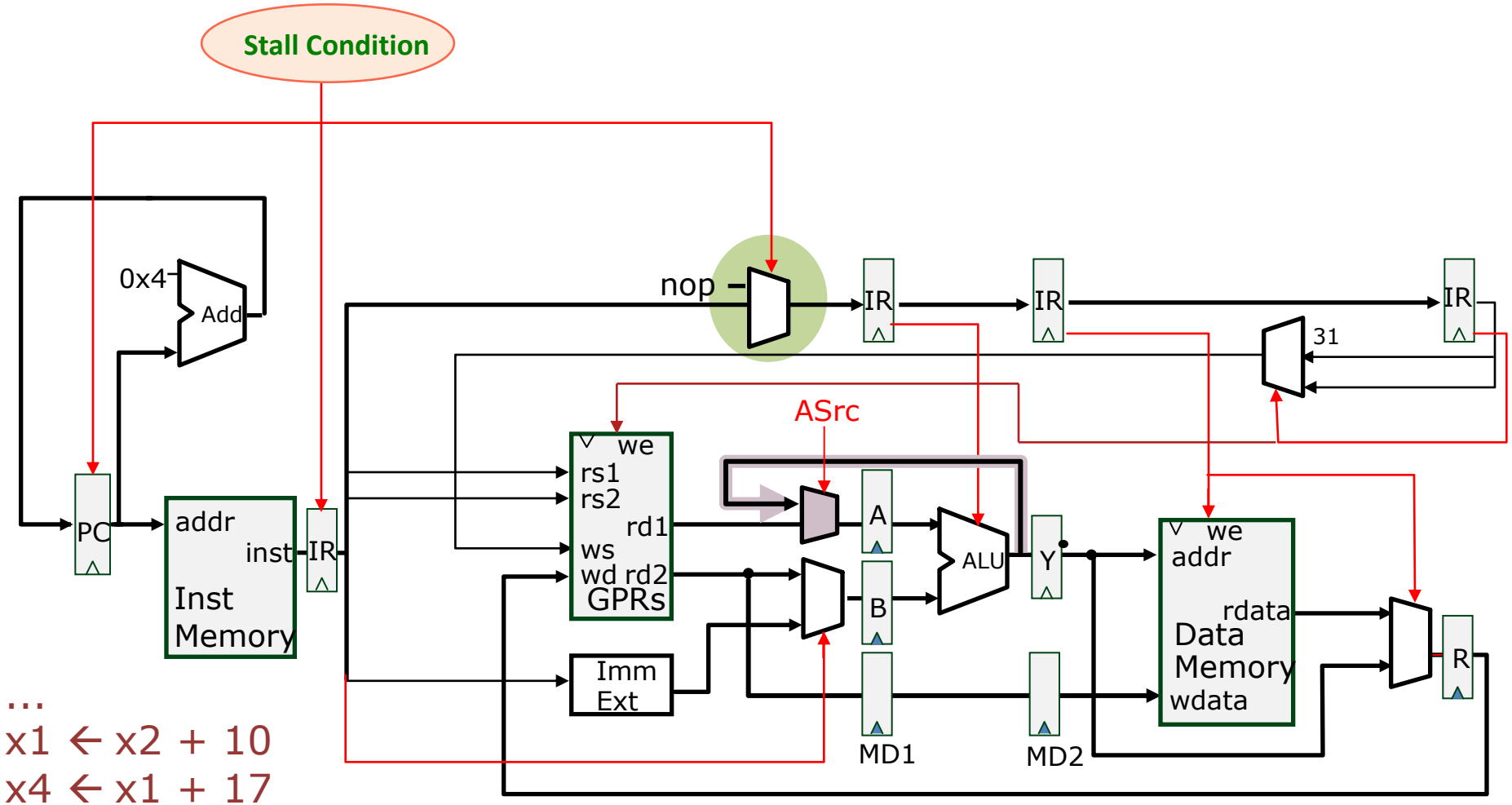


Bypassing



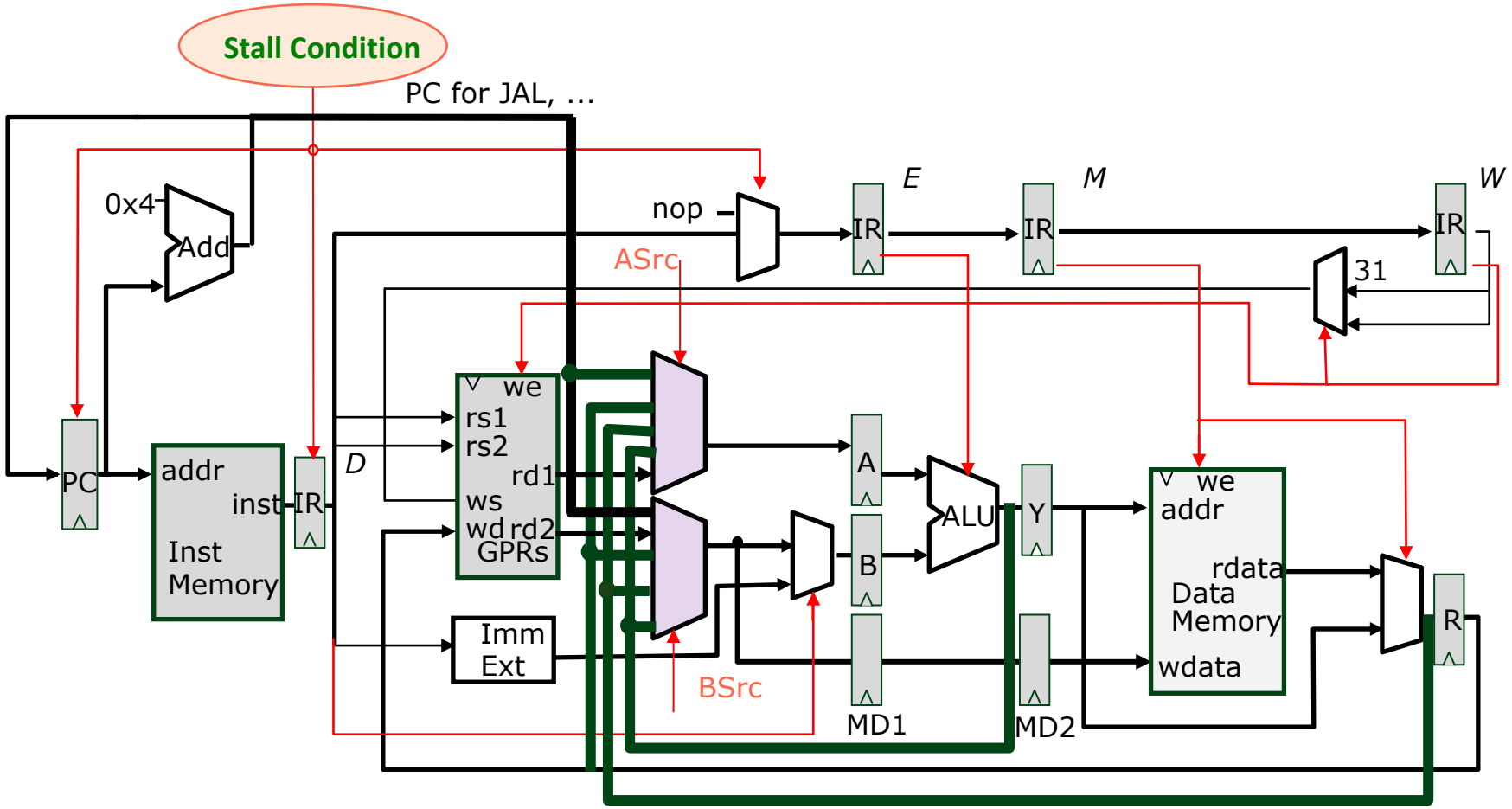
- Each stall or kill introduces a bubble in the pipeline → CPI > 1

Adding a Bypass



...
x1 ← x2 + 10
x4 ← x1 + 17
...

Fully Bypassed Datapath



Why a program may have CPI >1

- Why an Instruction may not be dispatched every cycle (CPI>1)?
  - Full bypassing may be too expensive to implement
    - Typically all frequently used paths are provided
    - Some infrequently used bypass paths may increase cycle time and counteract the benefit of reducing CPI
  - Loads have two cycle latency
    - Instruction after load cannot use load result
    - Some ISA define *load delay slots*, a software-visible pipeline hazard (compiler schedules independent instruction or inserts NOP to avoid hazard)

Next Learning Module

- CPU performance evaluation