

Instruction Pipelining

Ryan Lee

Computer Science and Artificial Intelligence Laboratory
M.I.T.

Announcements

Announcements

- Lab 1 released later today
 - Designing 3 different cache models using Pin
 - Due Oct. 1

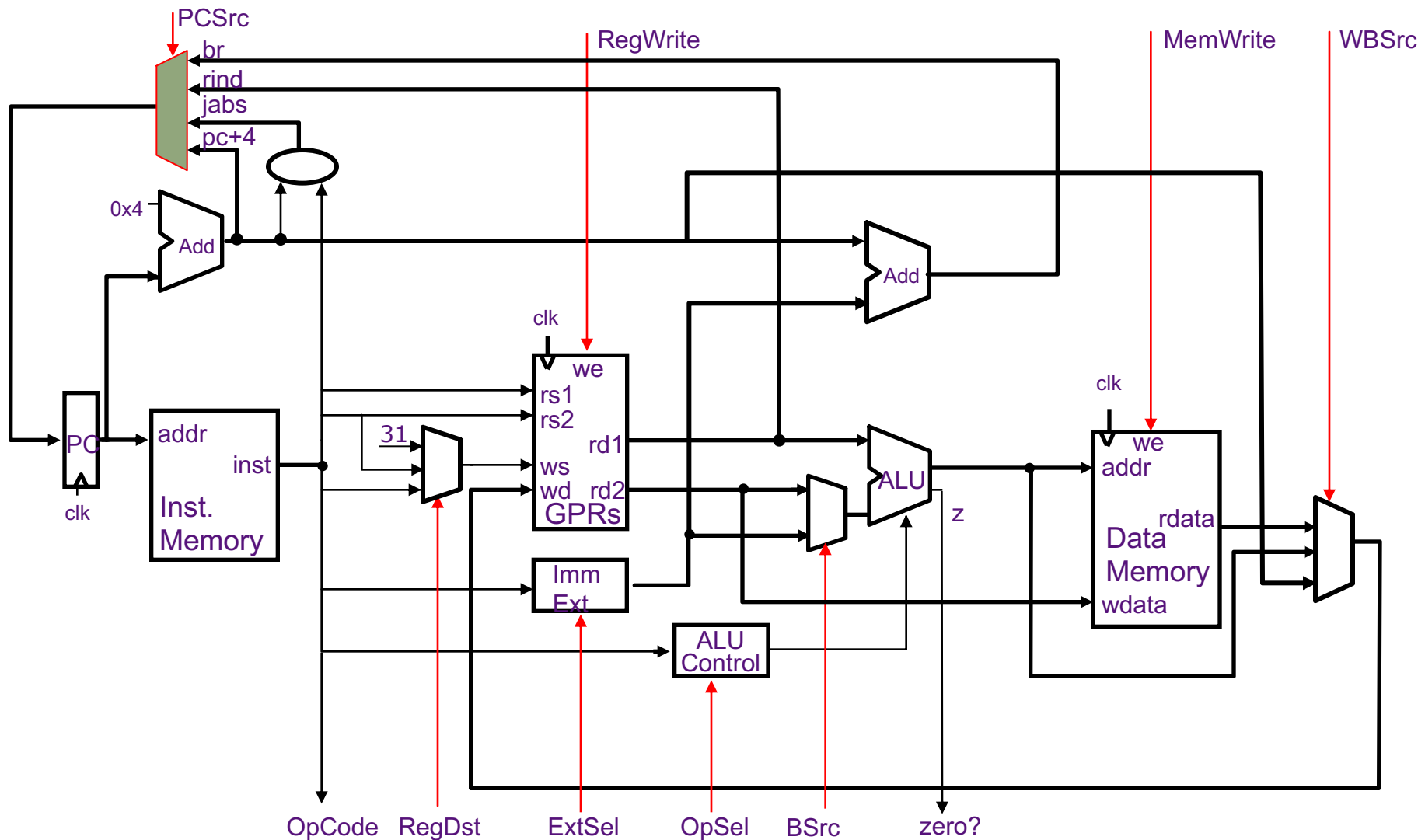
Announcements

- Lab 1 released later today
 - Designing 3 different cache models using Pin
 - Due Oct. 1
- Please view the Pin tutorial video posted on the course website (under Recitation tab)

Announcements

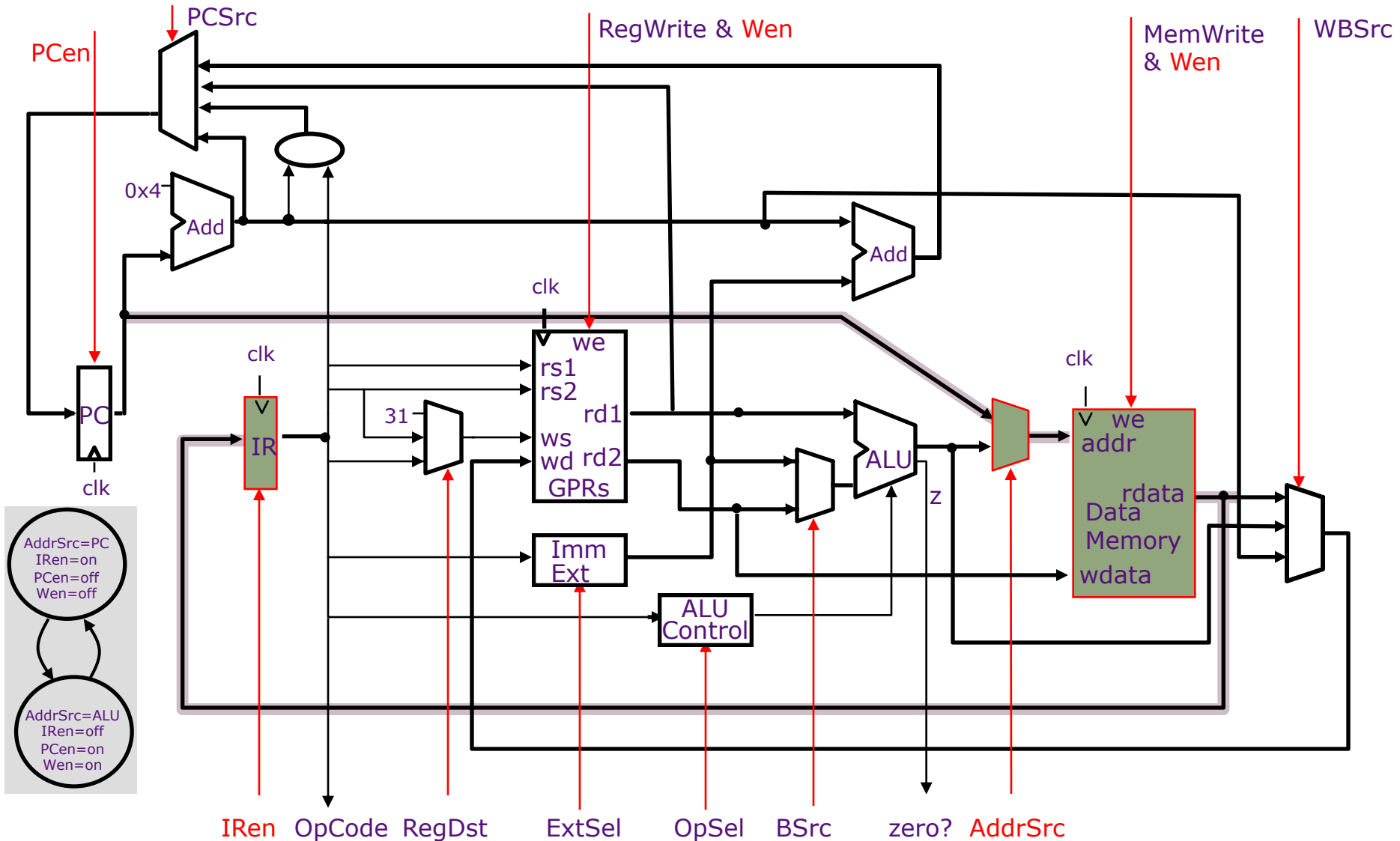
- Lab 1 released later today
 - Designing 3 different cache models using Pin
 - Due Oct. 1
- Please view the Pin tutorial video posted on the course website (under Recitation tab)
- Contact me if you cannot get access to lab machines

Reminder: Harvard-Style Single-Cycle Datapath for MIPS

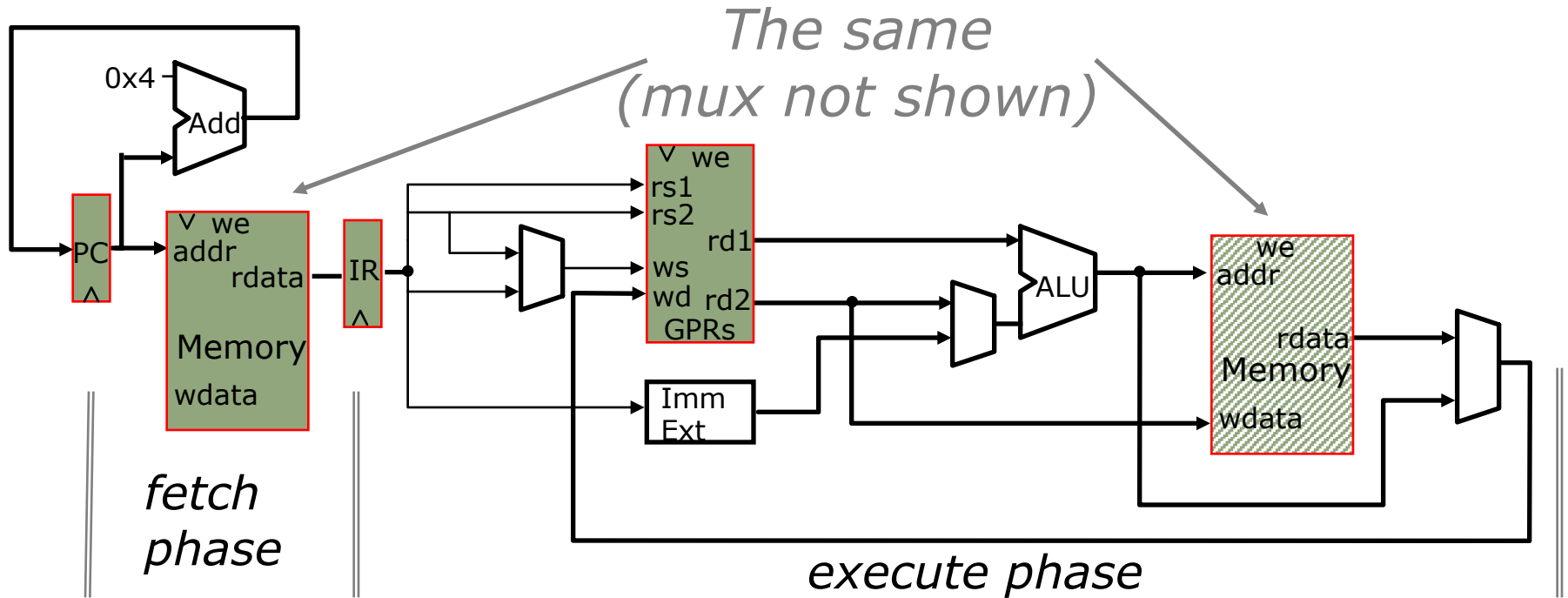


Reminder: Princeton Microarchitecture

Datapath & Control for 2 cycles-per-instruction



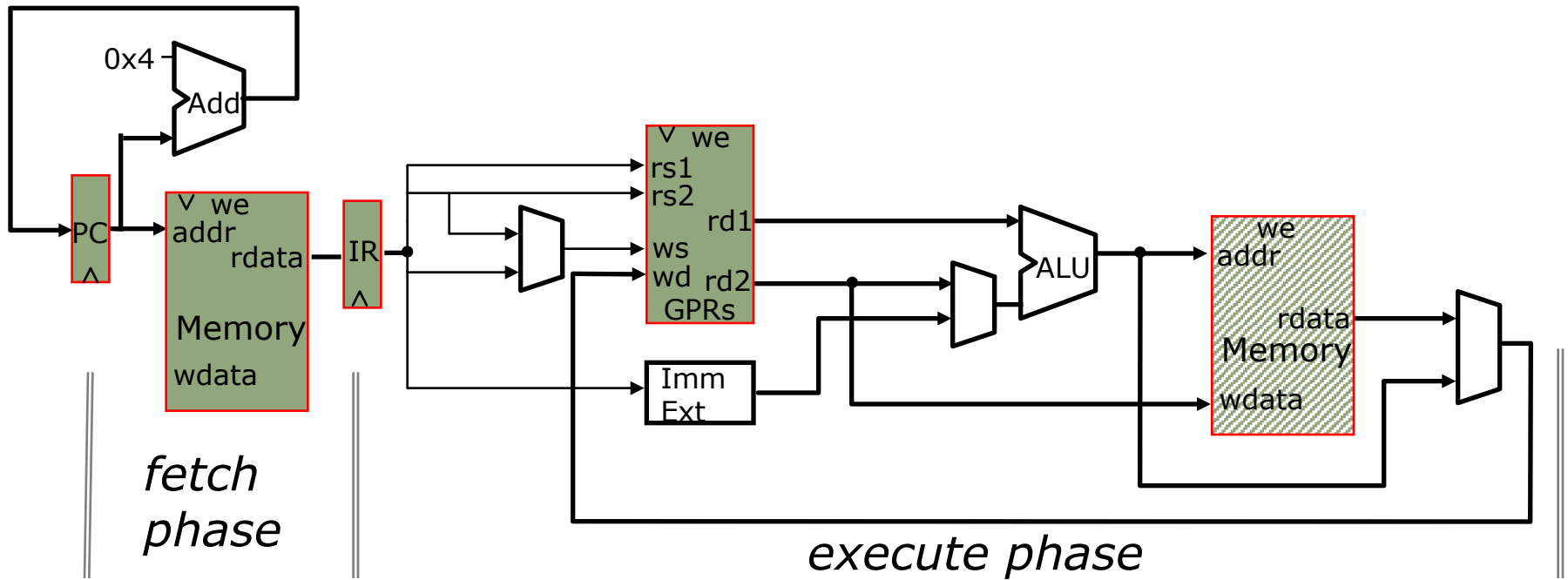
Princeton Microarchitecture (redrawn)



Only one of the phases is active in any cycle
⇒ a lot of datapath not used at any given time

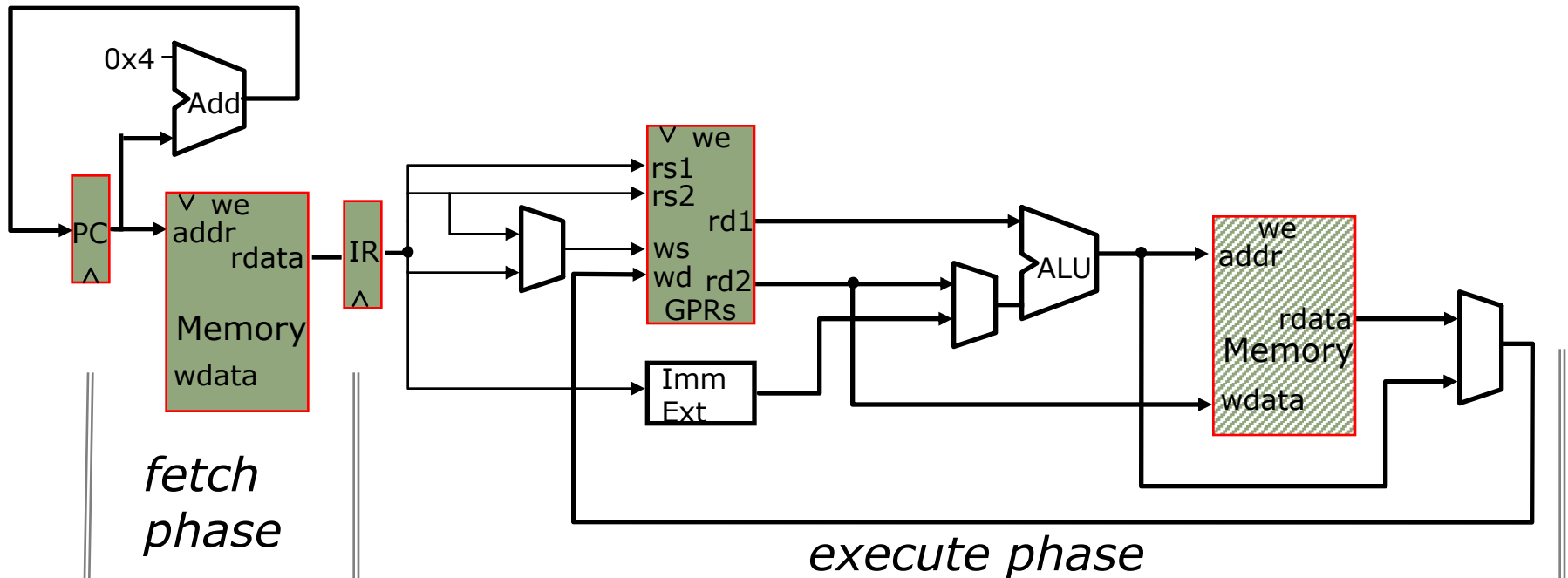
Princeton Microarchitecture

Overlapped execution



Princeton Microarchitecture

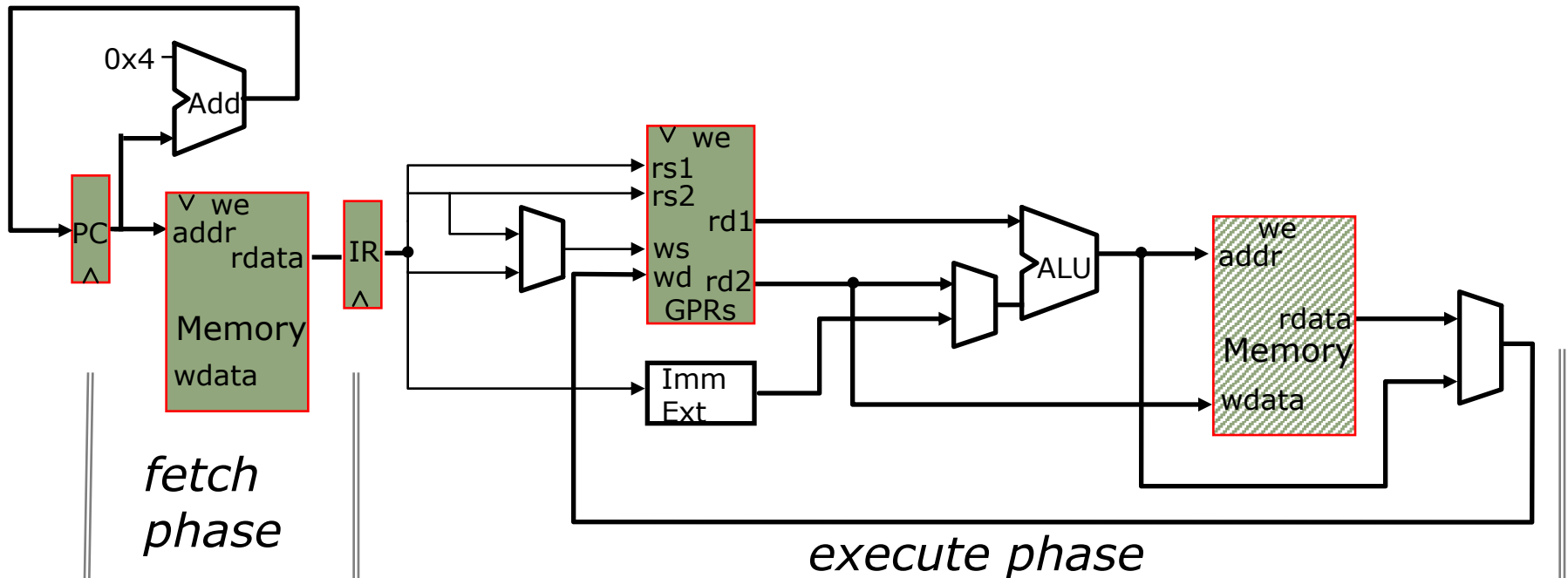
Overlapped execution



Can we overlap instruction fetch and execute?

Princeton Microarchitecture

Overlapped execution

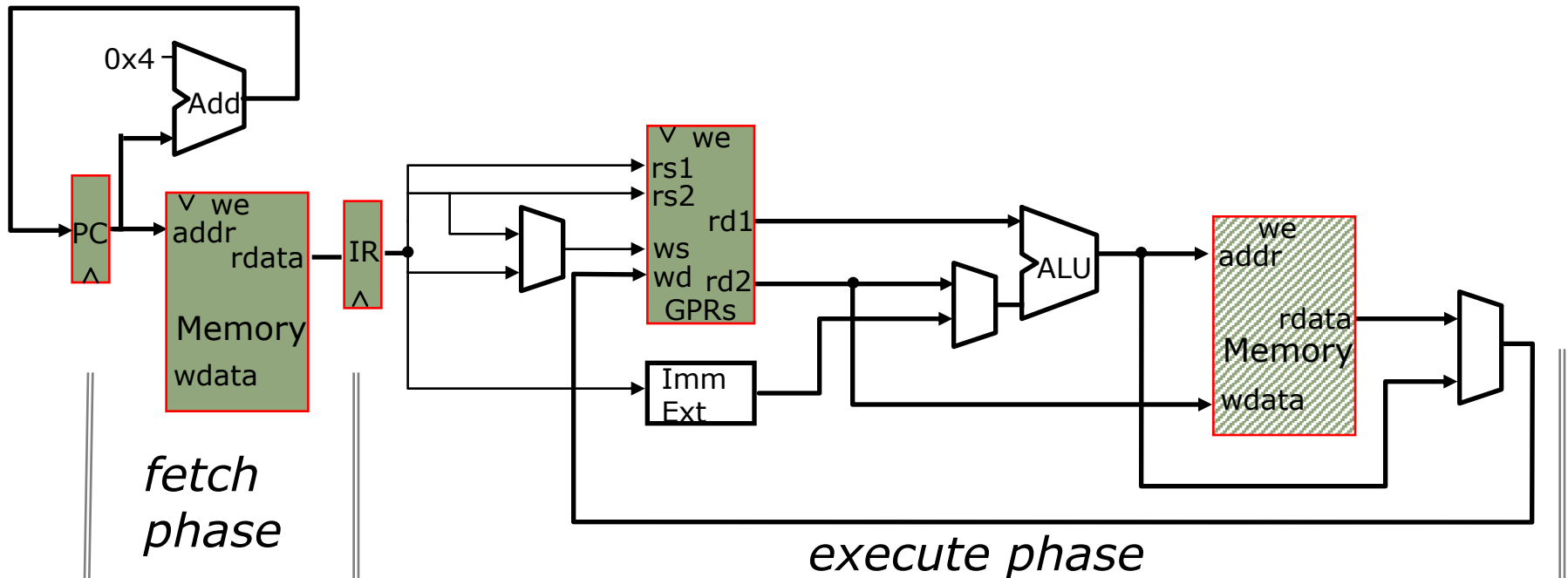


Can we overlap instruction fetch and execute?

Yes, unless IR contains a Load or Store

Princeton Microarchitecture

Overlapped execution



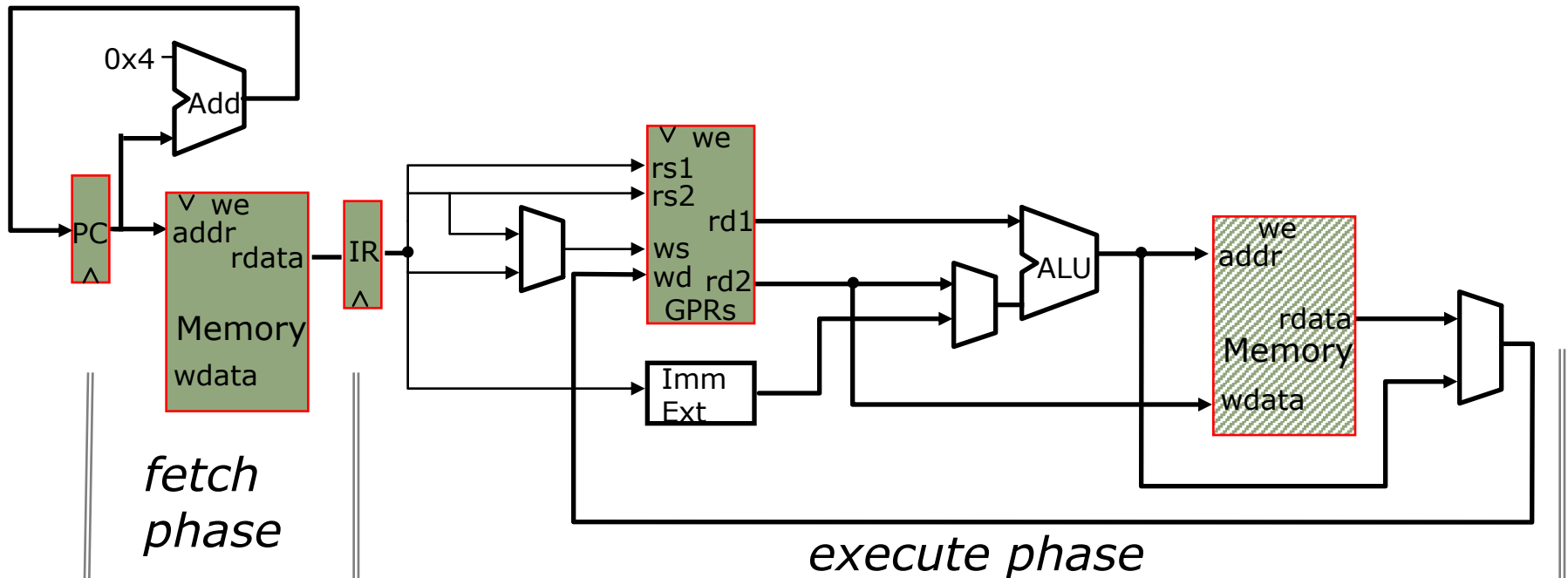
Can we overlap instruction fetch and execute?

Yes, unless IR contains a Load or Store

Which action should be prioritized?

Princeton Microarchitecture

Overlapped execution



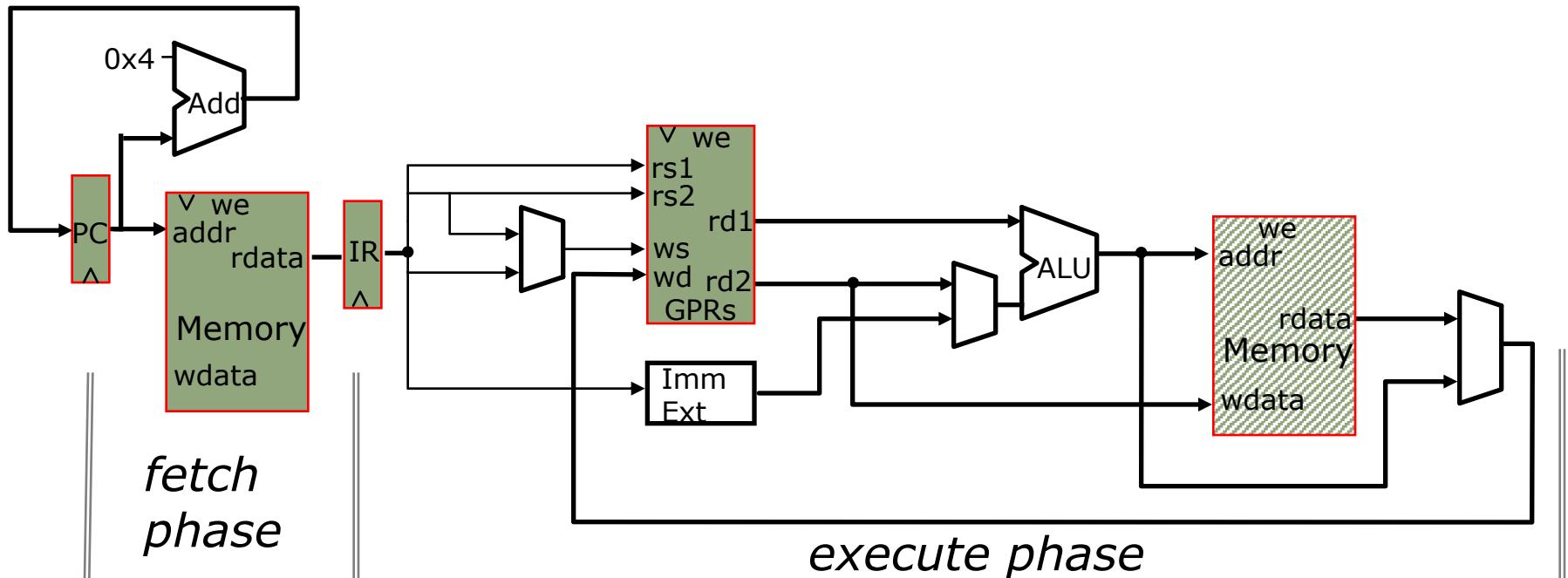
Can we overlap instruction fetch and execute?

Yes, unless IR contains a Load or Store

Which action should be prioritized? Execute

Princeton Microarchitecture

Overlapped execution



Can we overlap instruction fetch and execute?

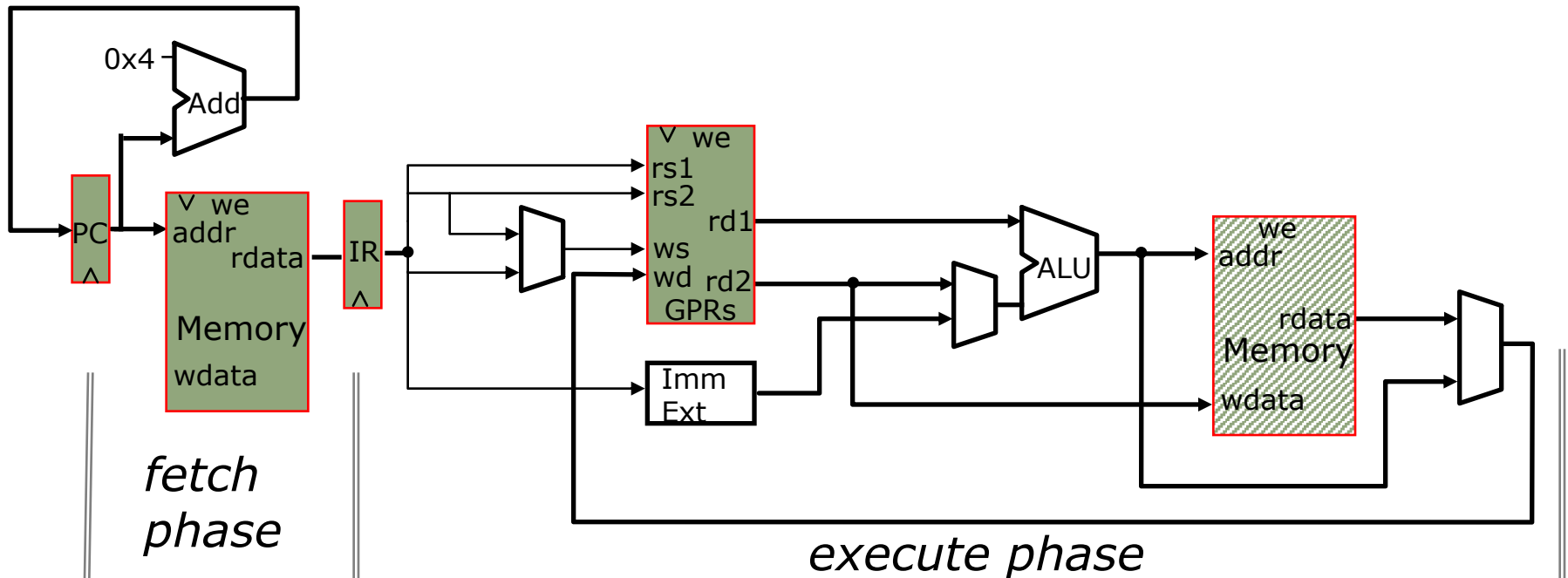
Yes, unless IR contains a Load or Store

Which action should be prioritized? Execute

What do we do with Fetch?

Princeton Microarchitecture

Overlapped execution



Can we overlap instruction fetch and execute?

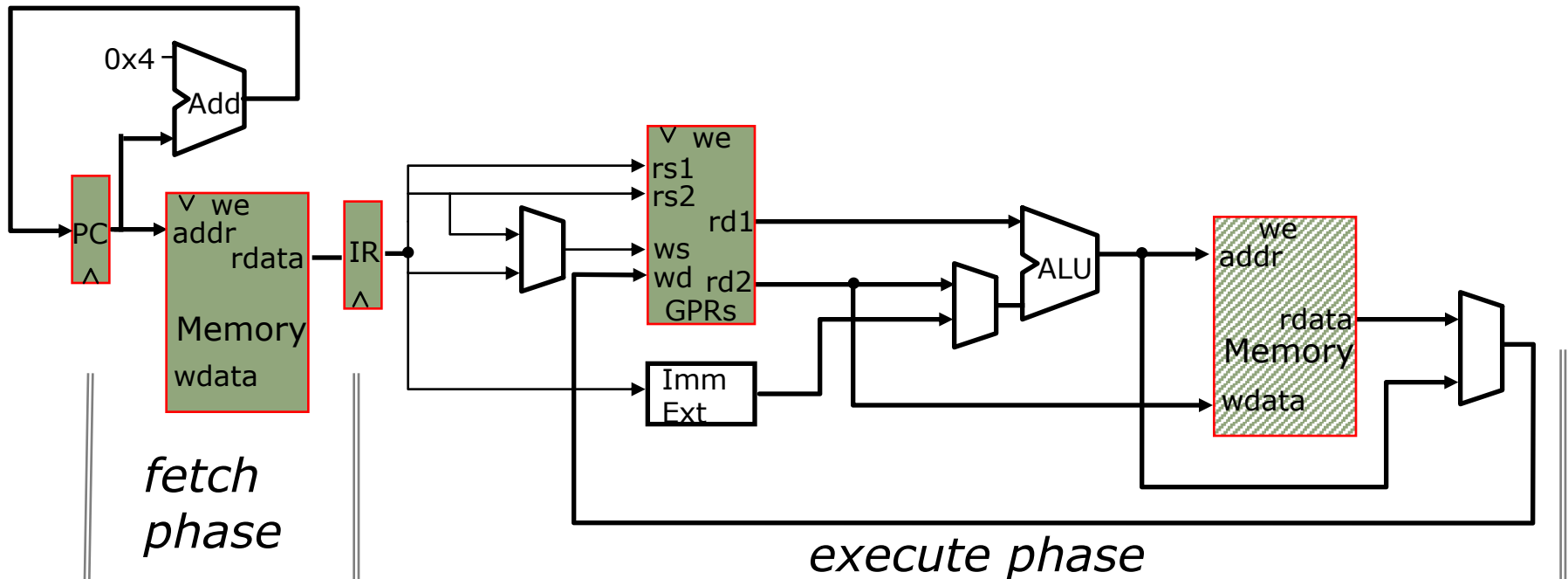
Yes, unless IR contains a Load or Store

Which action should be prioritized? Execute

What do we do with Fetch? Stall it

Princeton Microarchitecture

Overlapped execution



Can we overlap instruction fetch and execute?

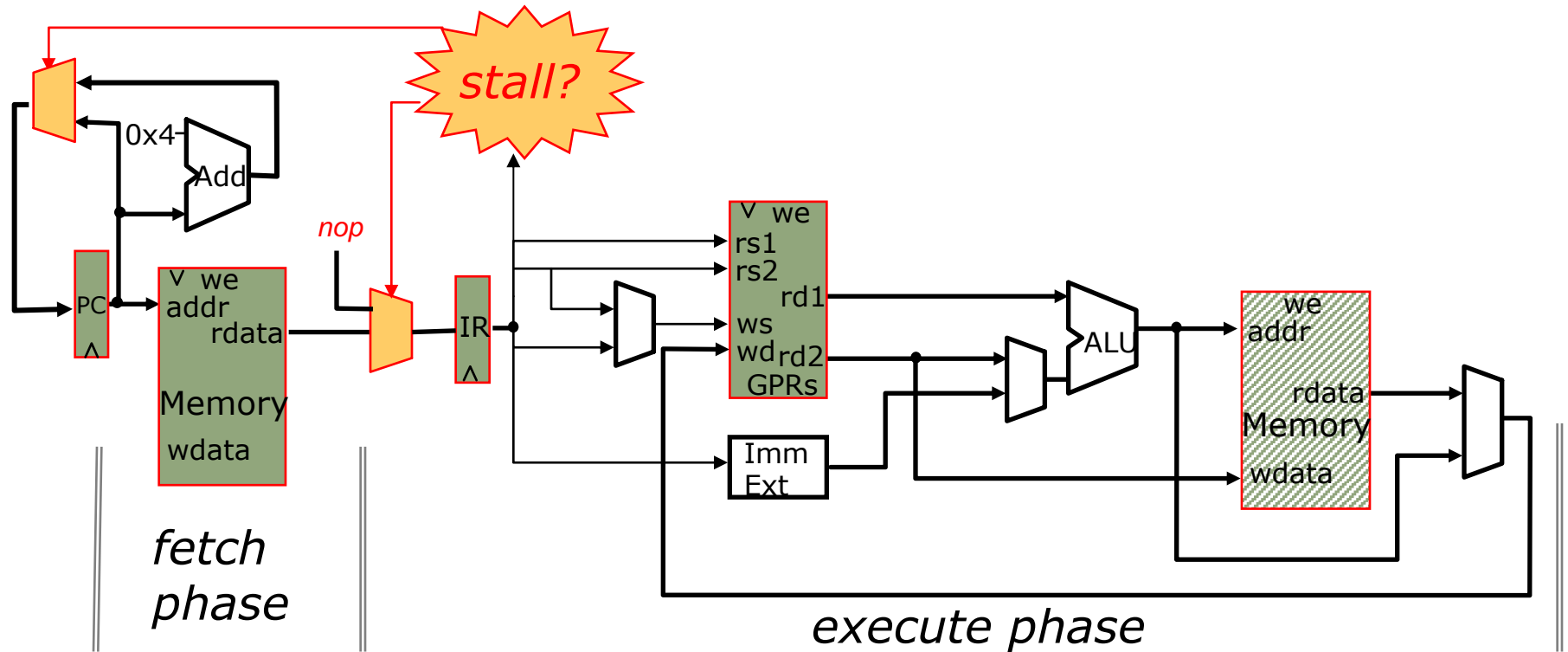
Yes, unless IR contains a Load or Store

Which action should be prioritized? Execute

What do we do with Fetch? Stall it How?

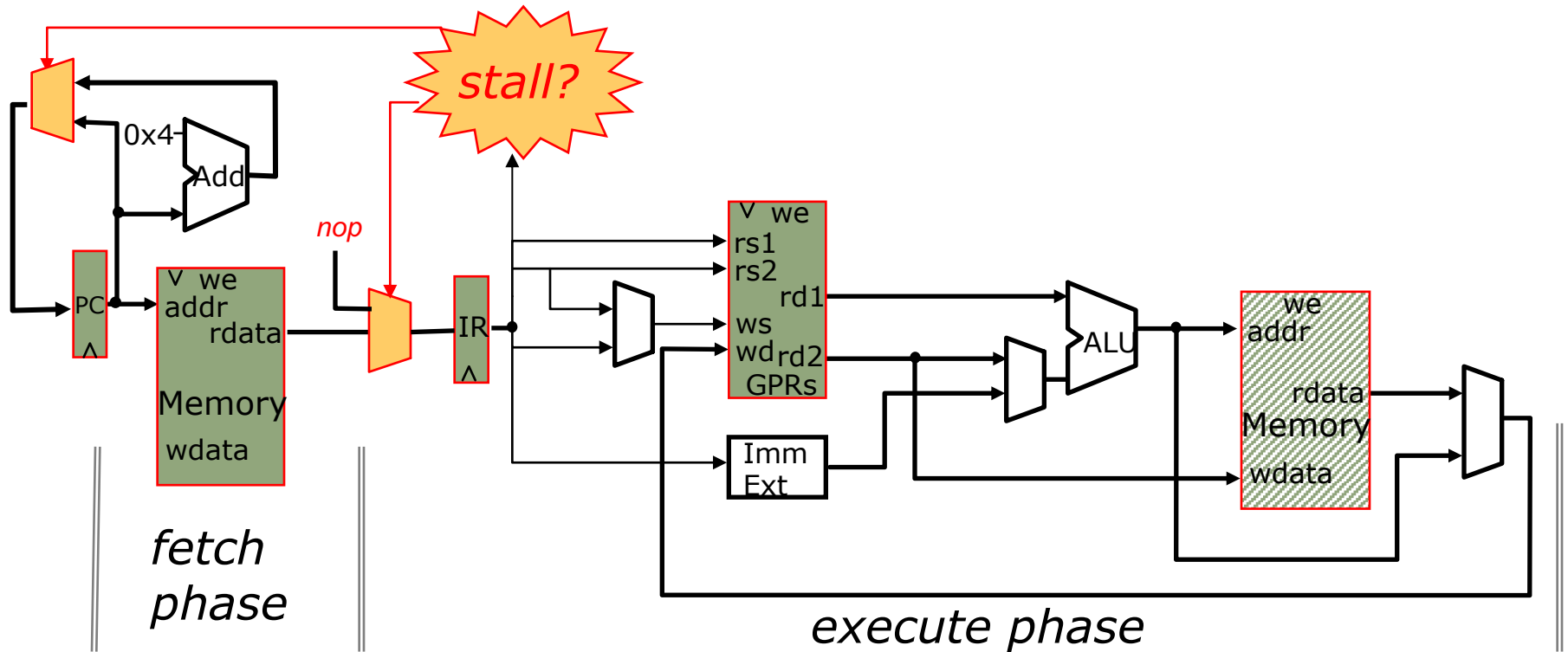
Stalling the instruction fetch

Princeton Microarchitecture



Stalling the instruction fetch

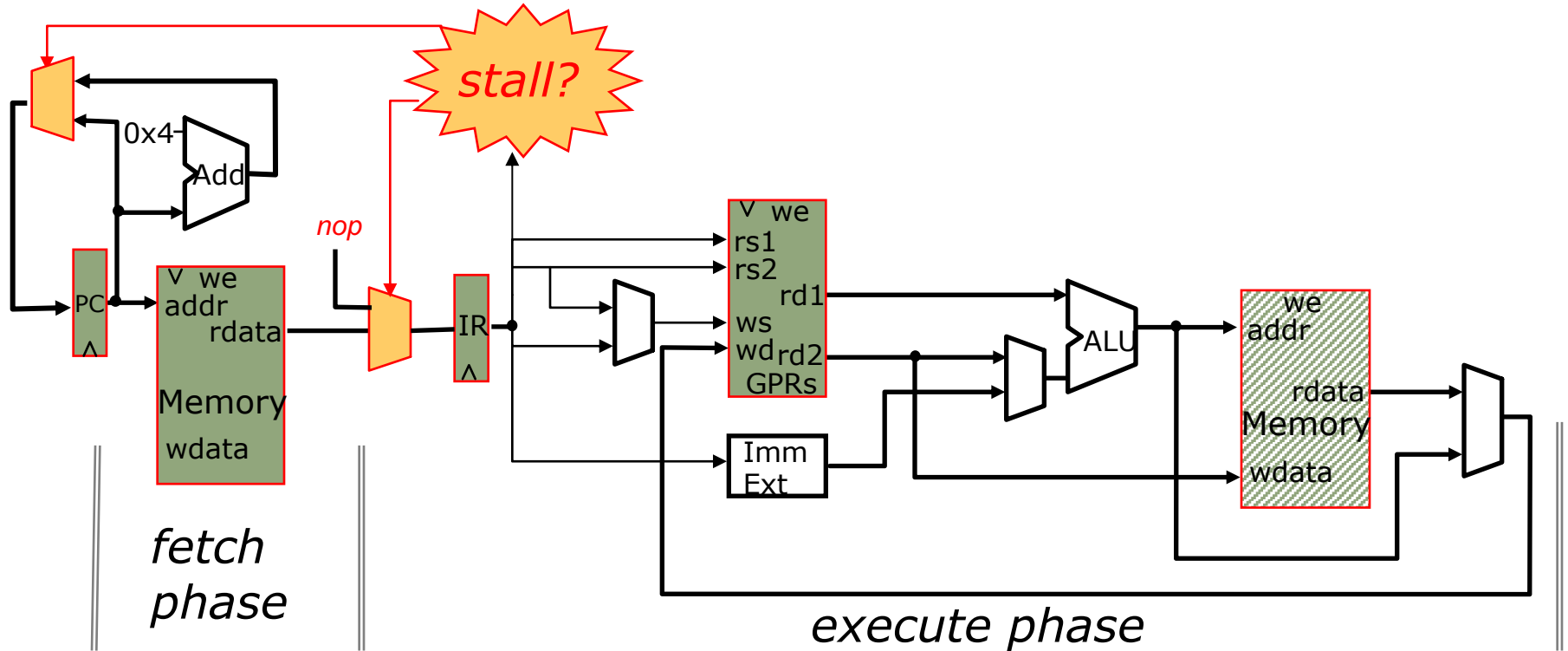
Princeton Microarchitecture



When stall condition is indicated

Stalling the instruction fetch

Princeton Microarchitecture

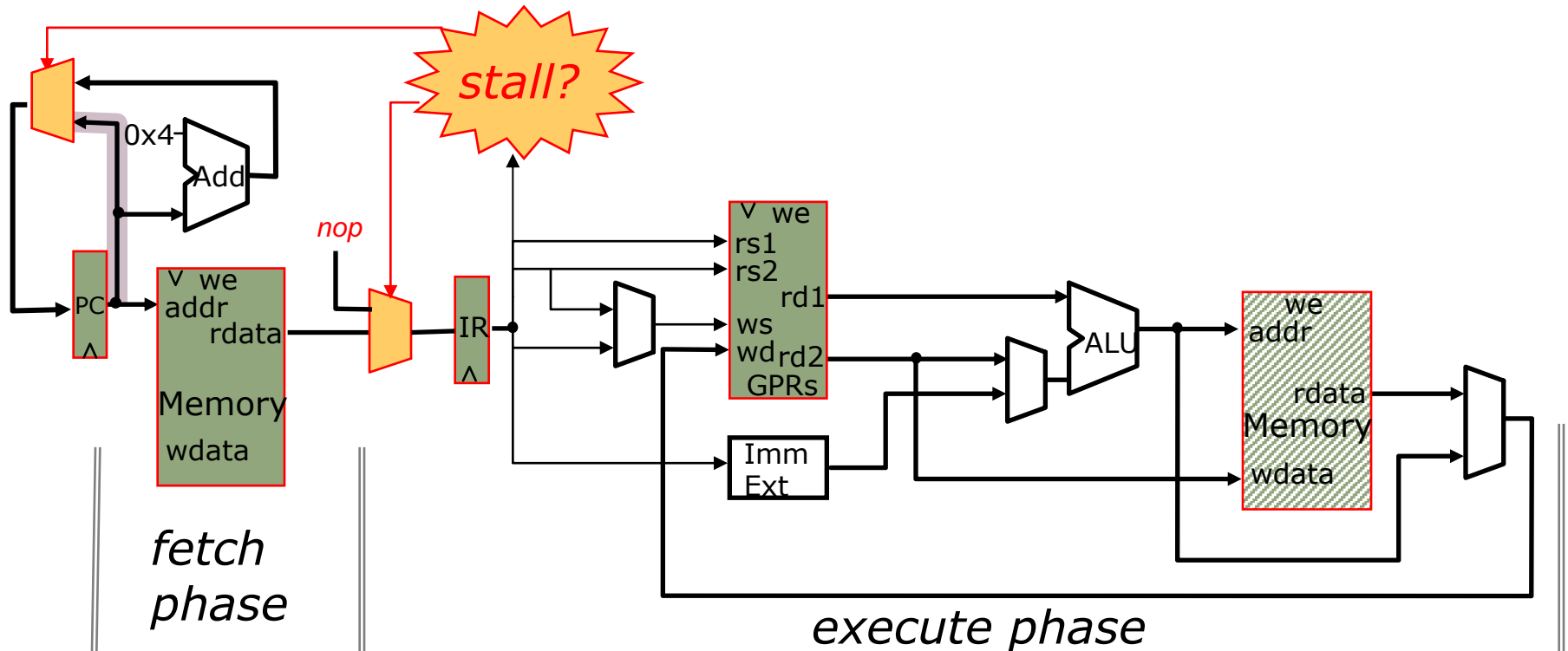


When stall condition is indicated

- *don't fetch a new instruction and don't change the PC*

Stalling the instruction fetch

Princeton Microarchitecture

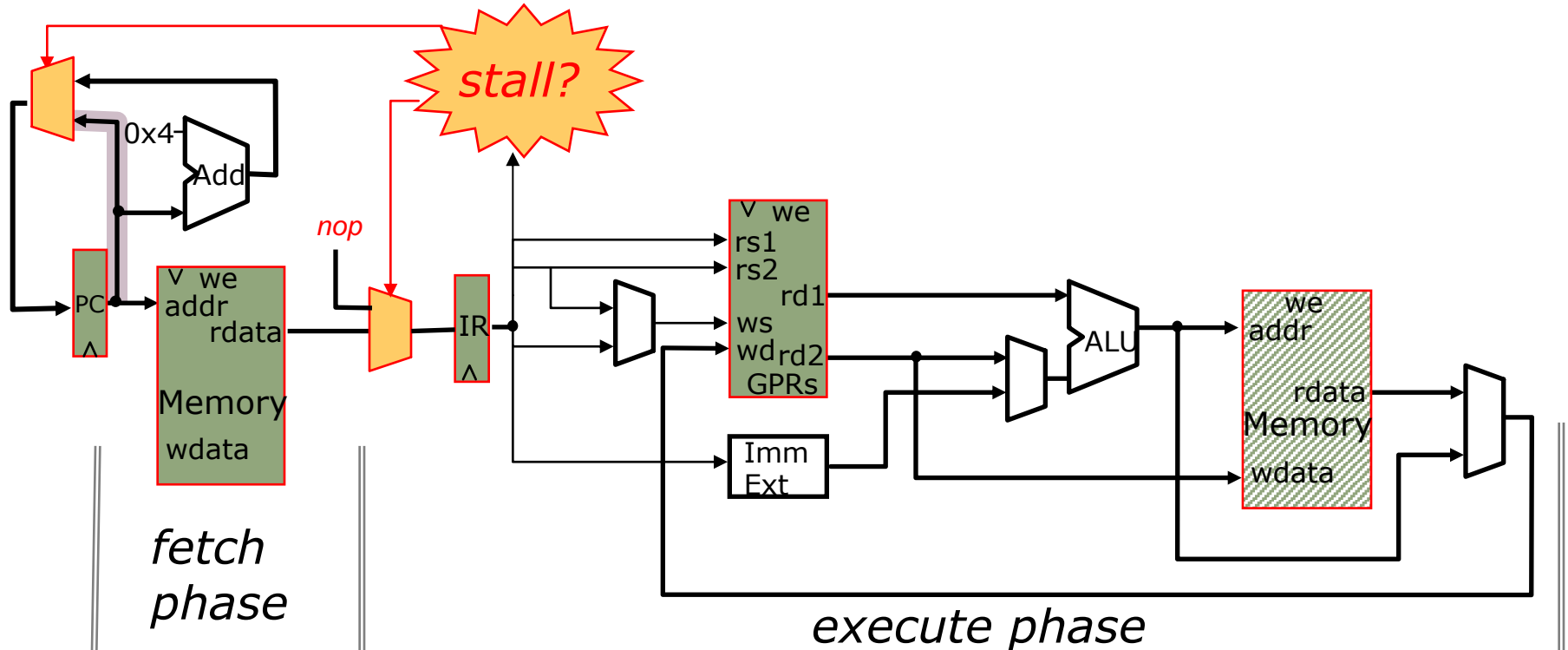


When stall condition is indicated

- *don't fetch a new instruction and don't change the PC*

Stalling the instruction fetch

Princeton Microarchitecture

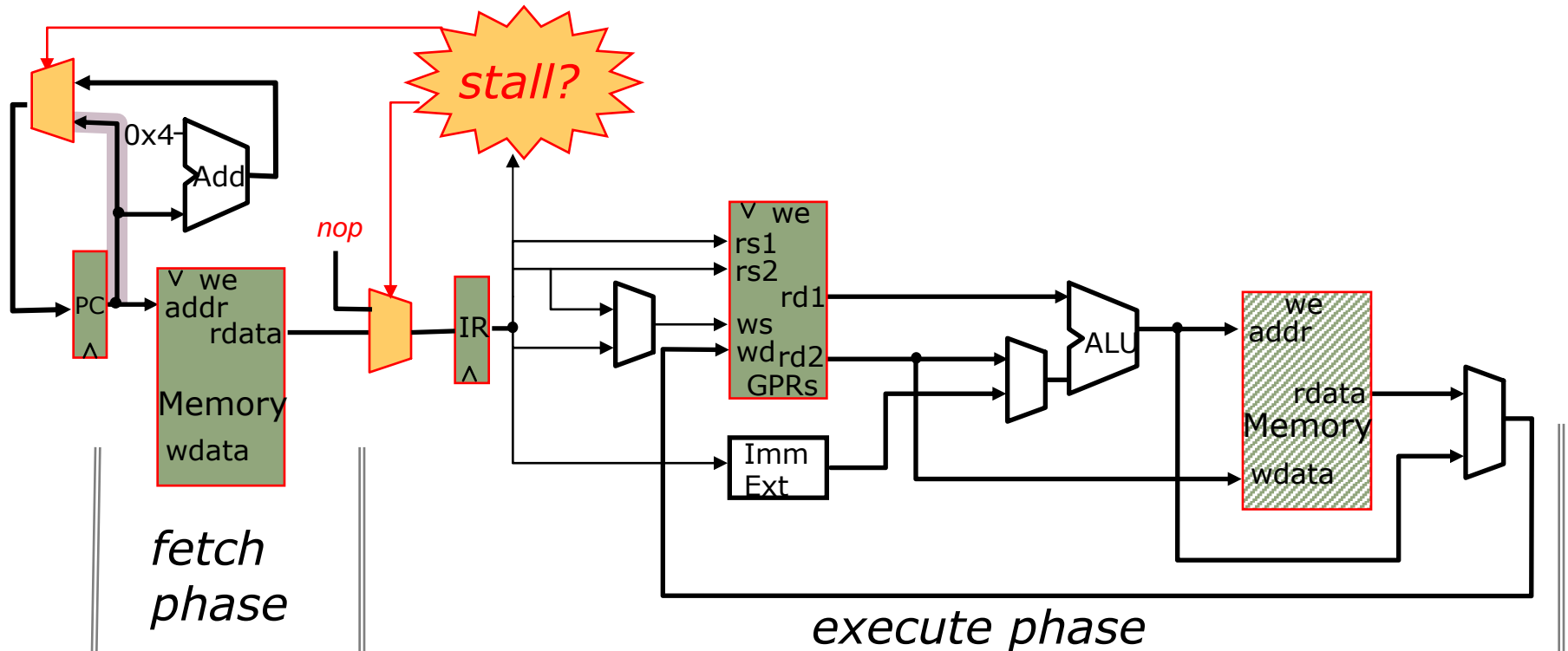


When stall condition is indicated

- *don't fetch a new instruction and don't change the PC*
- *insert a nop in the IR*

Stalling the instruction fetch

Princeton Microarchitecture

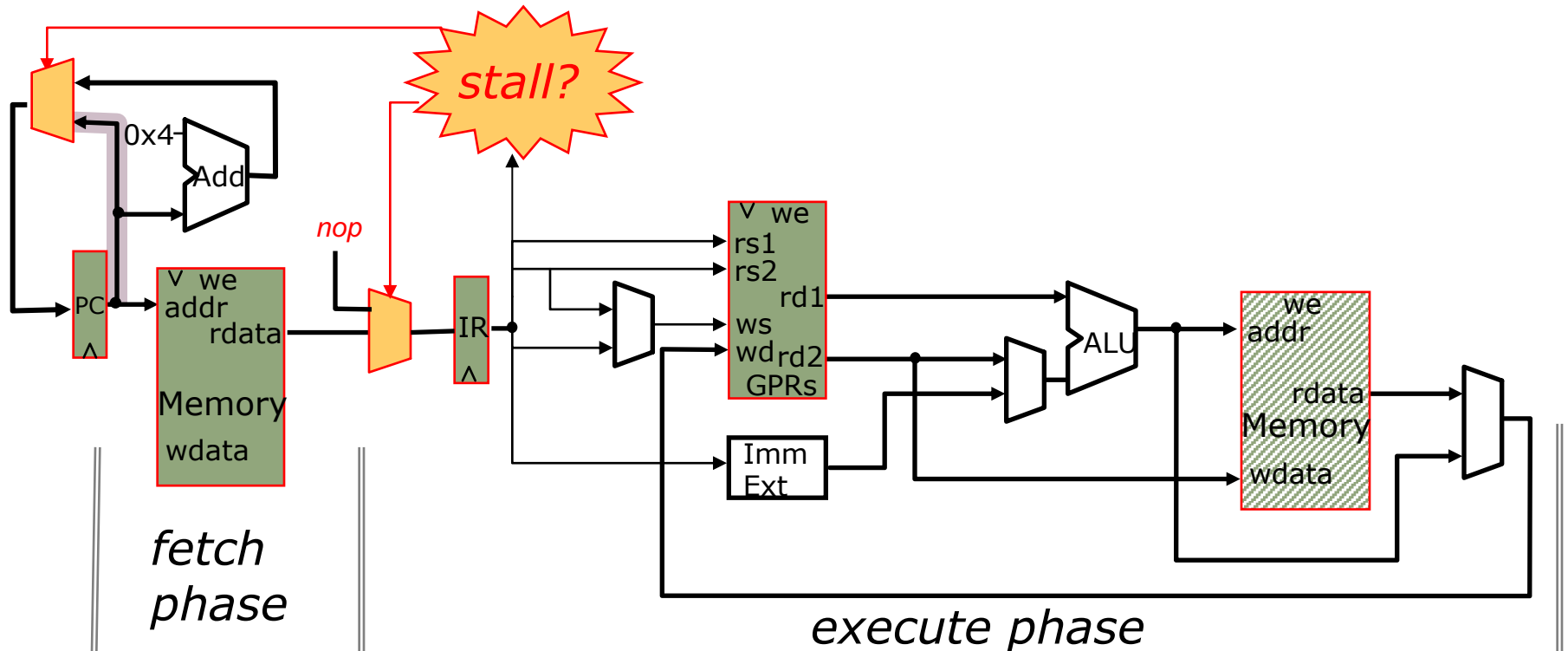


When stall condition is indicated

- *don't fetch a new instruction and don't change the PC*
- *insert a nop in the IR*
- *set the Memory Address mux to ALU (not shown)*

Stalling the instruction fetch

Princeton Microarchitecture



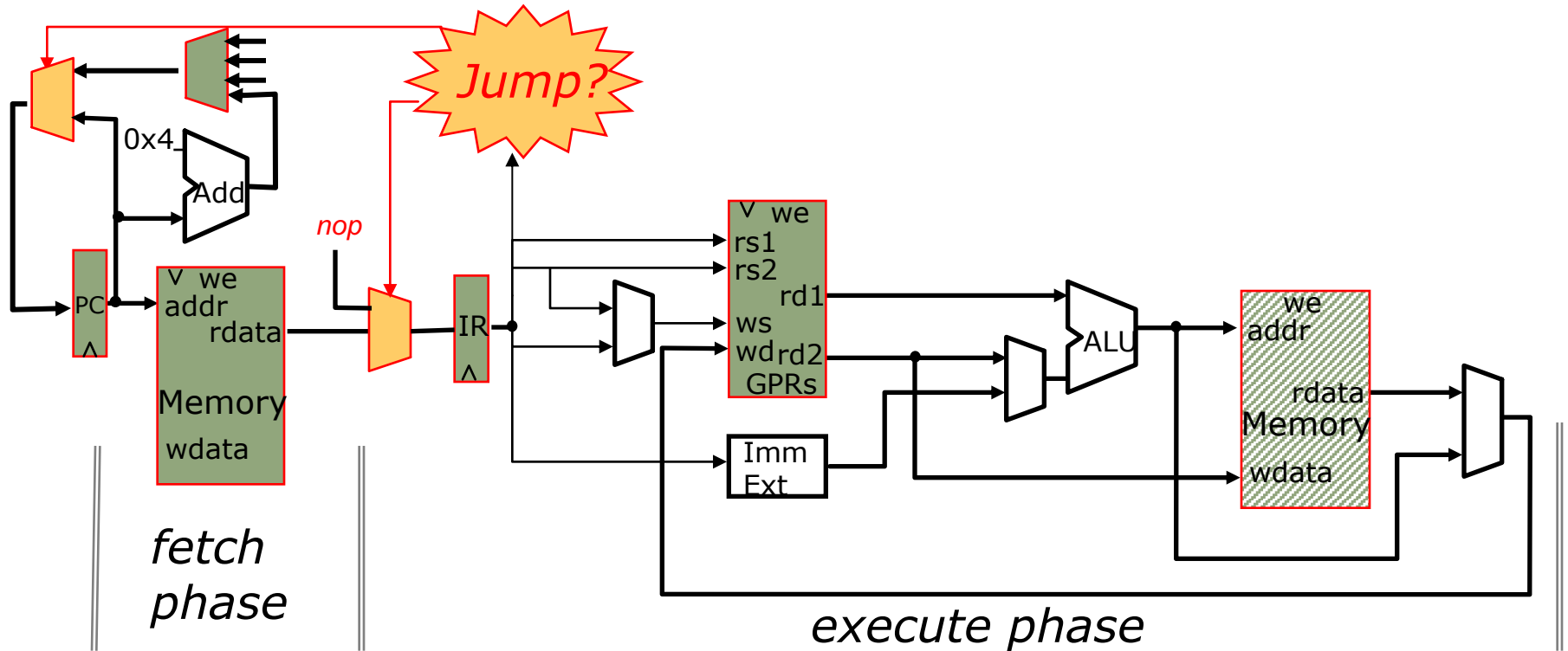
When stall condition is indicated

- *don't fetch a new instruction and don't change the PC*
- *insert a nop in the IR*
- *set the Memory Address mux to ALU (not shown)*

What if IR contains a jump or branch instruction?

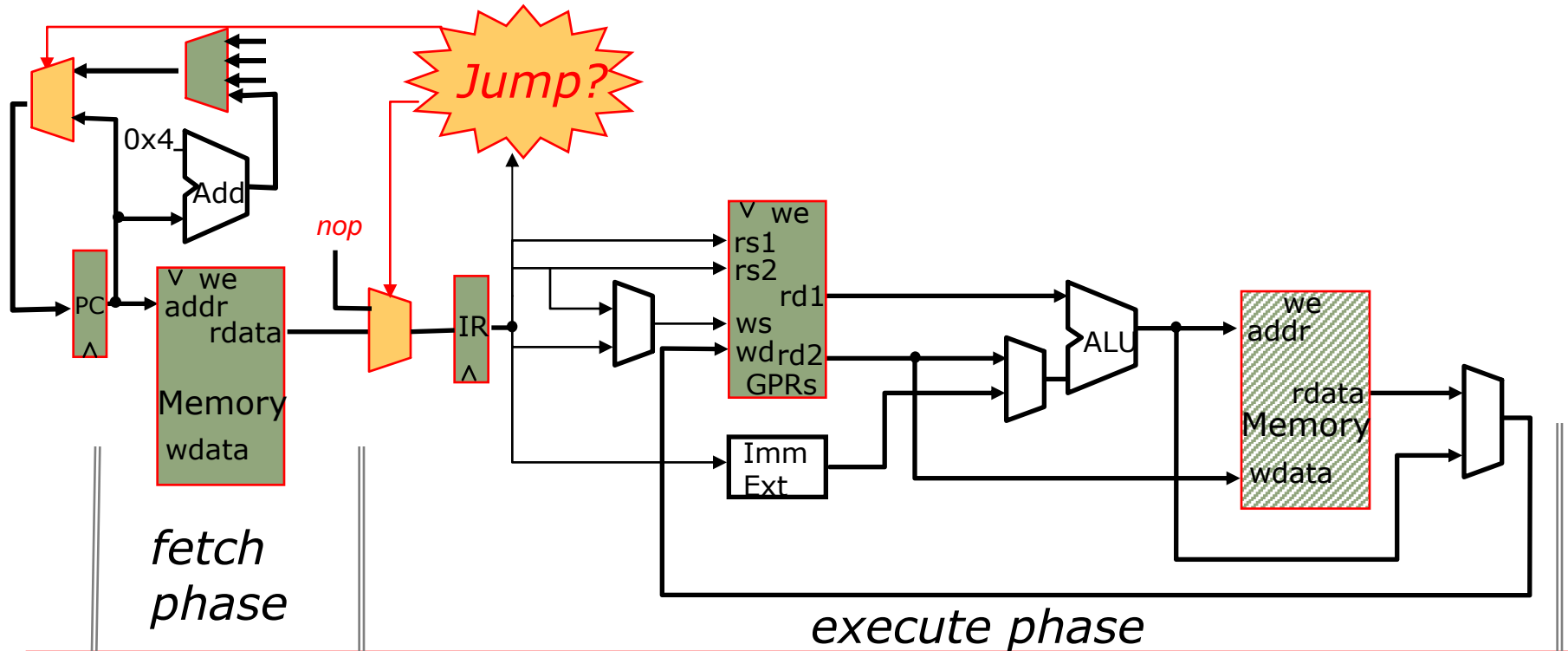
Need to stall on branches

Princeton Microarchitecture



Need to stall on branches

Princeton Microarchitecture

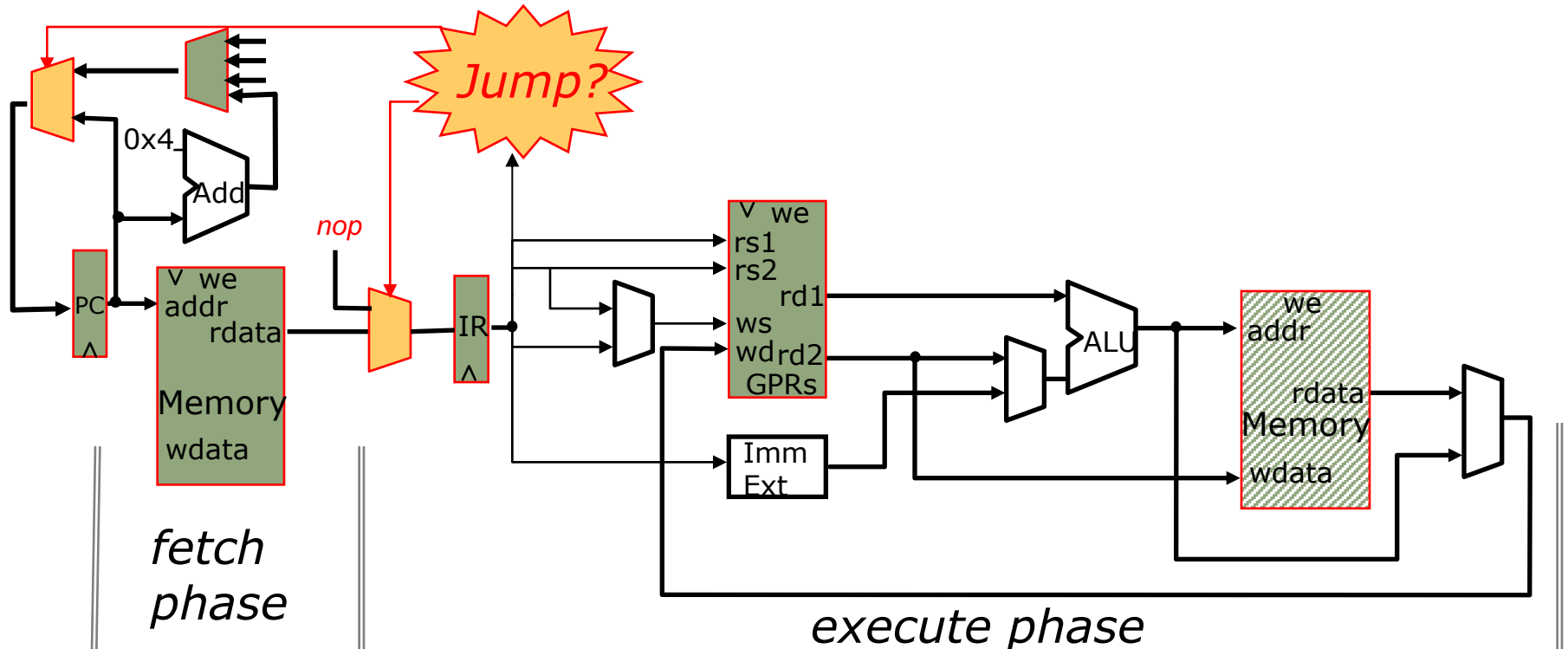


When IR contains a jump or taken branch

- *no "structural conflict" for the memory*

Need to stall on branches

Princeton Microarchitecture

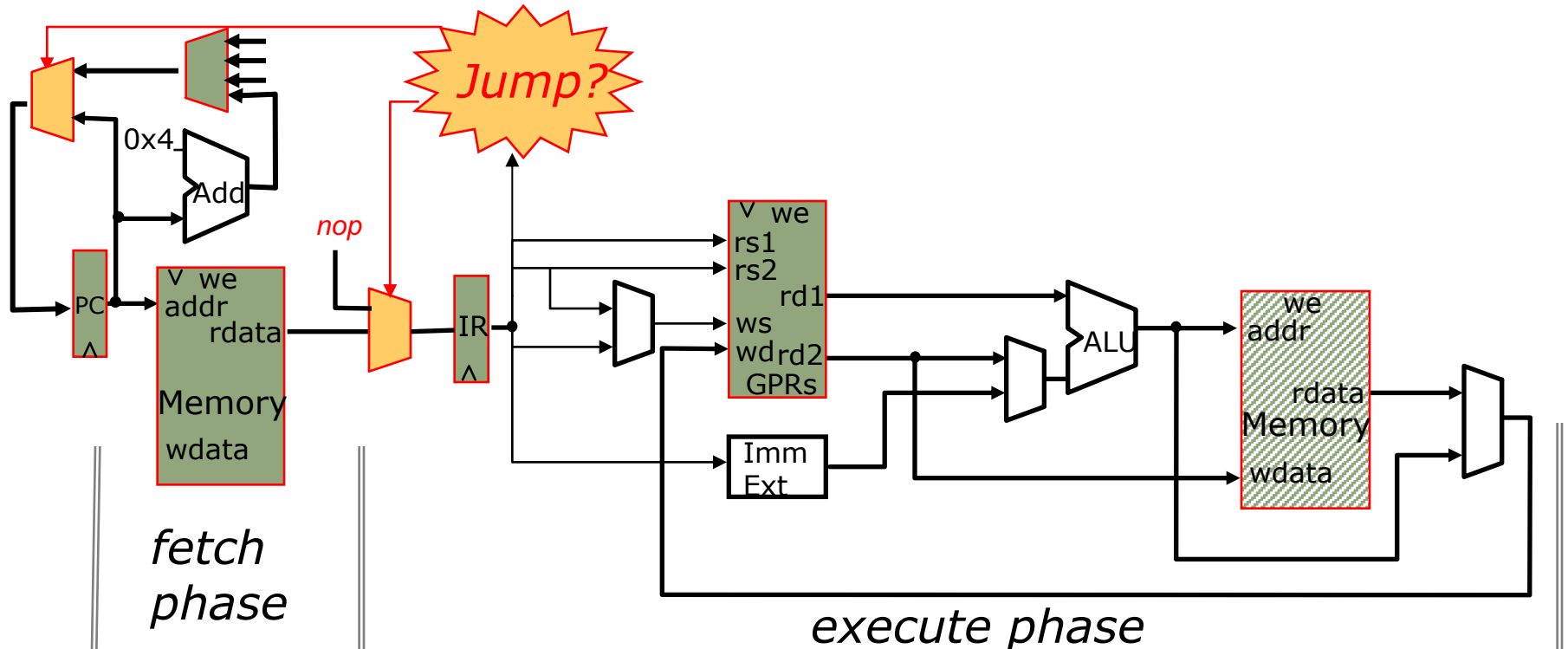


When IR contains a jump or taken branch

- *no "structural conflict" for the memory*
- *but we do not have the correct PC value in the PC*

Need to stall on branches

Princeton Microarchitecture

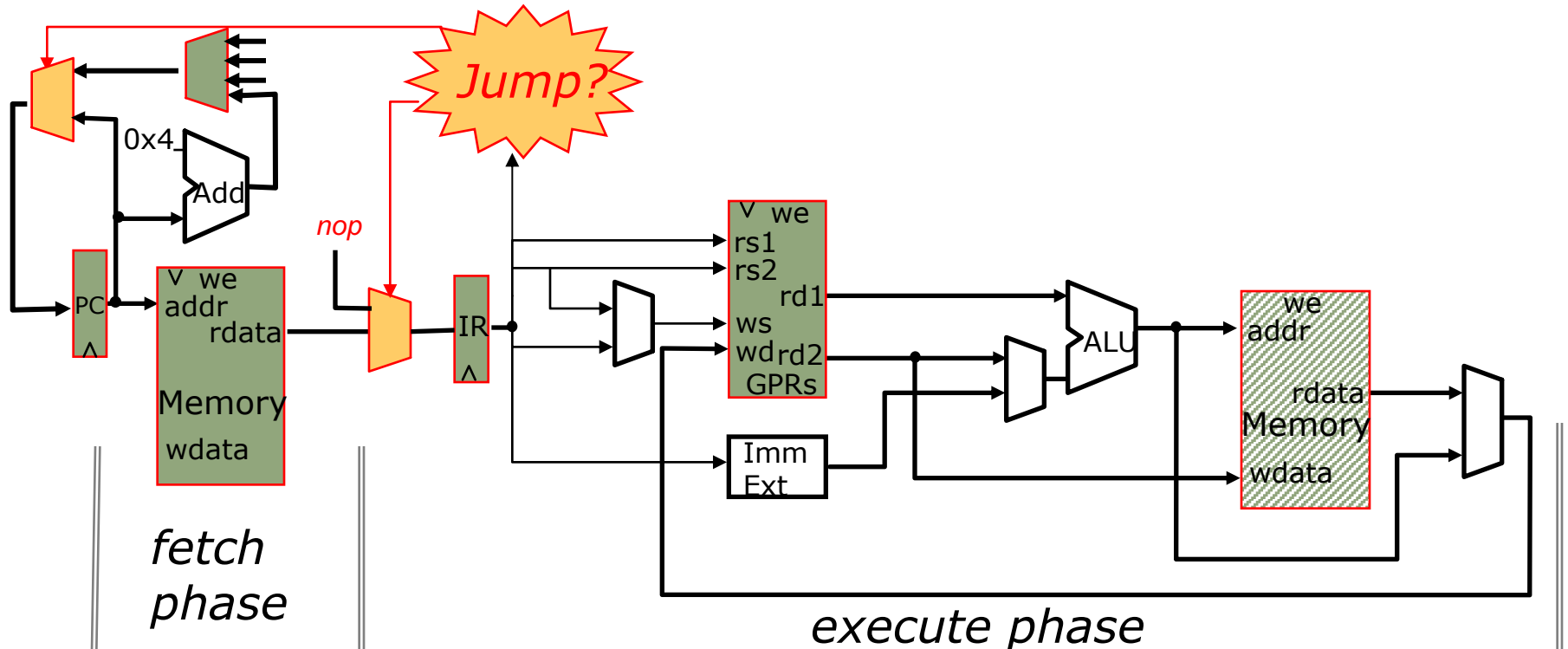


When IR contains a jump or taken branch

- *no "structural conflict" for the memory*
- *but we do not have the correct PC value in the PC*
- *memory cannot be used – Address Mux setting is irrelevant*

Need to stall on branches

Princeton Microarchitecture

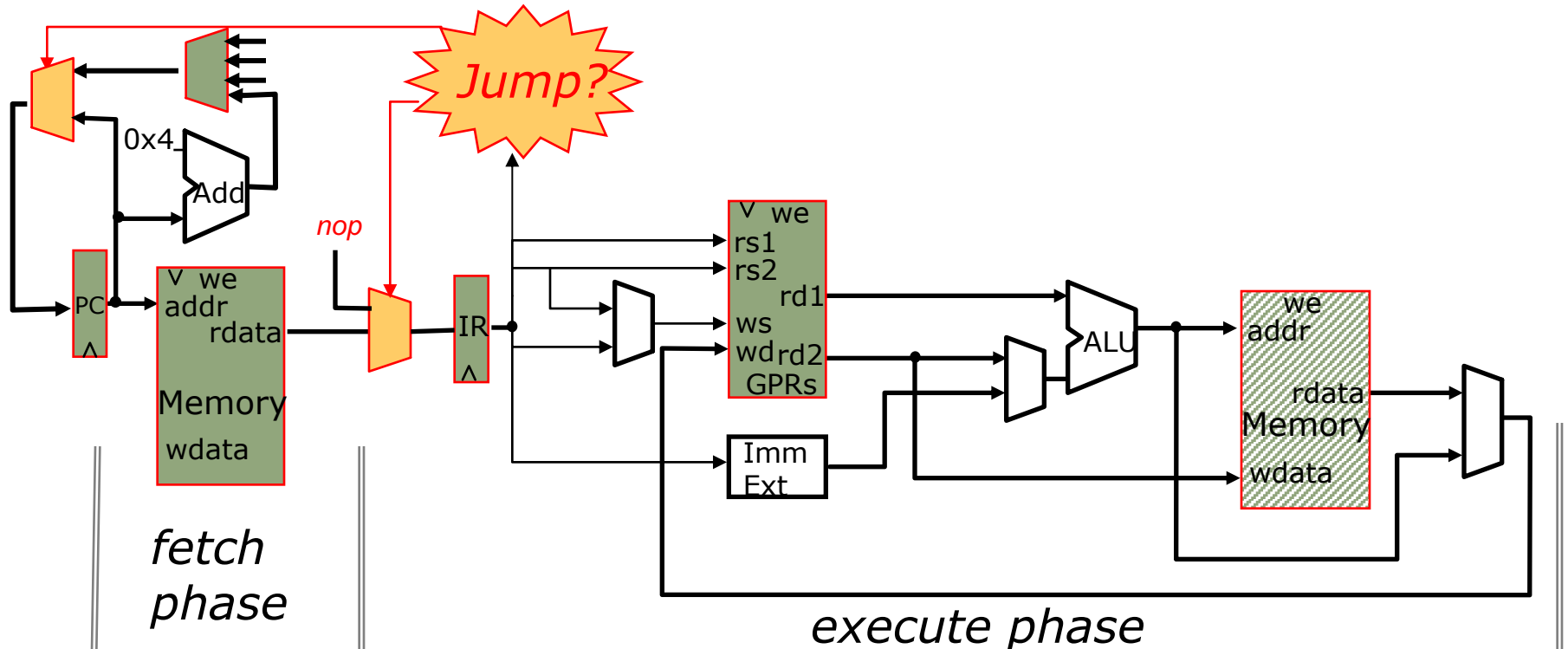


When IR contains a jump or taken branch

- *no "structural conflict" for the memory*
- *but we do not have the correct PC value in the PC*
- *memory cannot be used – Address Mux setting is irrelevant*
- *insert a nop in the IR*

Need to stall on branches

Princeton Microarchitecture

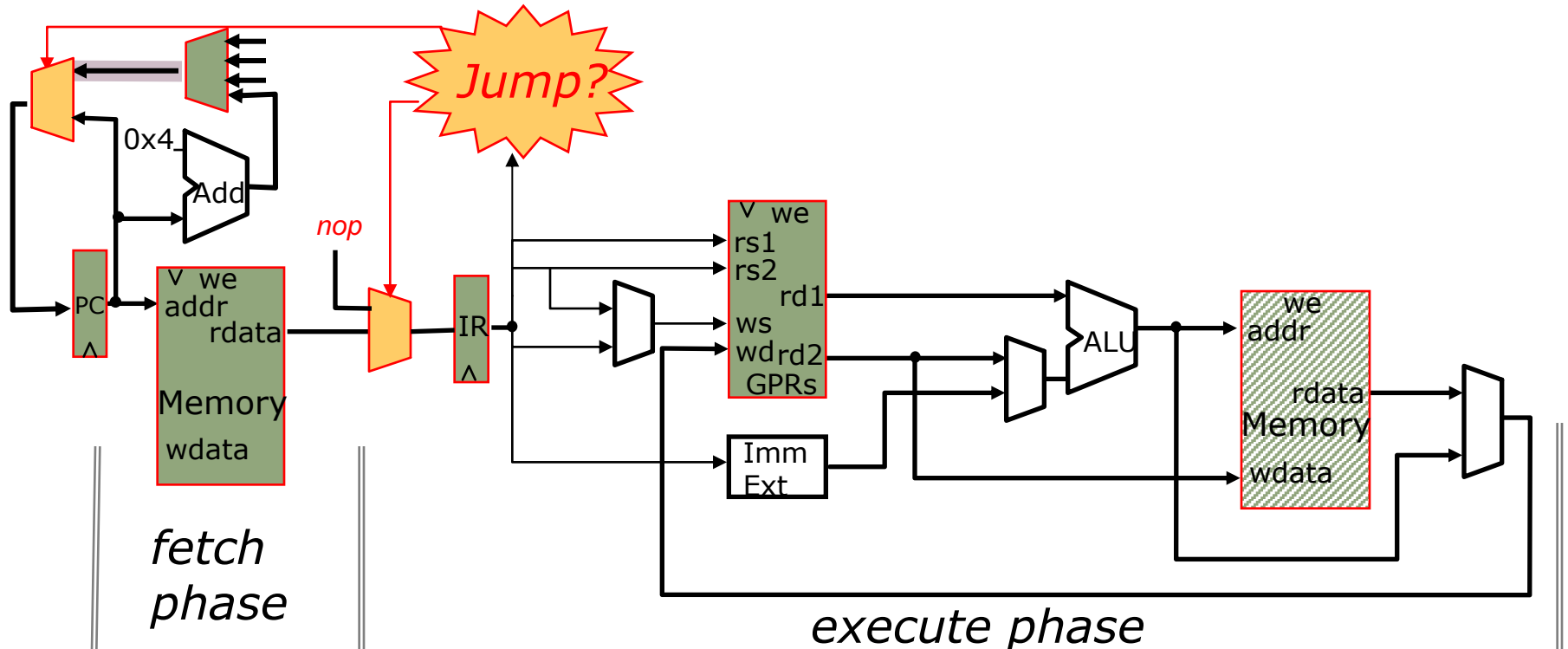


When IR contains a jump or taken branch

- *no "structural conflict" for the memory*
- *but we do not have the correct PC value in the PC*
- *memory cannot be used – Address Mux setting is irrelevant*
- *insert a nop in the IR*
- *insert the nextPC (branch-target) address in the PC*

Need to stall on branches

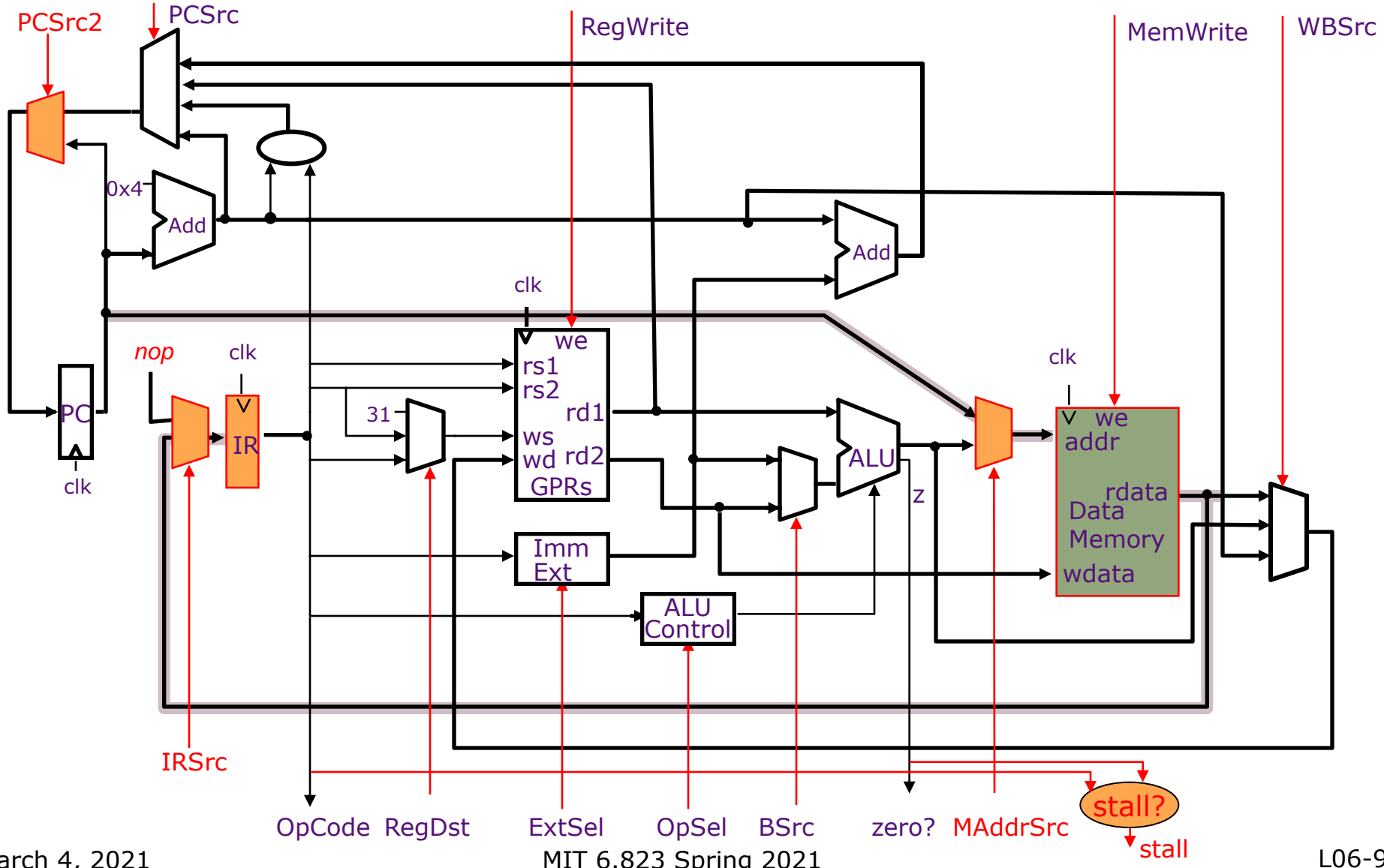
Princeton Microarchitecture



When IR contains a jump or taken branch

- *no "structural conflict" for the memory*
- *but we do not have the correct PC value in the PC*
- *memory cannot be used – Address Mux setting is irrelevant*
- *insert a nop in the IR*
- *insert the nextPC (branch-target) address in the PC*

Pipelined Princeton Microarchitecture



Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU												
ALUi												
ALUiu												
LW												
SW												
BEQZ _{z=1}												
BEQZ _{z=0}												
J												
JAL												
JR												
JALR												
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU		*	Reg	Func	no	yes	ALU	rd	pc+4			
ALUi		sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4			
ALUiu		uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4			
LW		sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4			
SW		sE ₁₆	Imm	+	yes	no	*	*	pc+4			
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4			
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4			
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4			
LW		sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4			
SW		sE ₁₆	Imm	+	yes	no	*	*	pc+4			
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc		
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc		
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc		
LW		sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4			
SW		sE ₁₆	Imm	+	yes	no	*	*	pc+4			
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	
LW		sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4			
SW		sE ₁₆	Imm	+	yes	no	*	*	pc+4			
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW		sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4			
SW		sE ₁₆	Imm	+	yes	no	*	*	pc+4			
BEQZ _{z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4			
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4			
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc		
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc		
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP												

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP	no	*	*	*	no	no	*	*	pc+4			

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP	no	*	*	*	no	no	*	*	pc+4	npc		

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{Z=1}		sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{Z=1}	yes	sE ₁₆	*	0?	no	no	*	*	br			
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{Z=1}	yes	sE ₁₆	*	0?	no	no	*	*	br	npc		
BEQZ _{Z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{z=1}	yes	sE ₁₆	*	0?	no	no	*	*	br	npc	nop	
BEQZ _{z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{z=1}	yes	sE ₁₆	*	0?	no	no	*	*	br	npc	nop	*
BEQZ _{z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J		*	*	*	no	no	*	*	jabs			
JAL		*	*	*	no	yes	PC	R31	jabs			
JR		*	*	*	no	no	*	*	rind			
JALR		*	*	*	no	yes	PC	R31	rind			
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{z=1}	yes	sE ₁₆	*	0?	no	no	*	*	br	npc	nop	*
BEQZ _{z=0}		sE ₁₆	*	0?	no	no	*	*	pc+4			
J	yes	*	*	*	no	no	*	*	jabs	npc	nop	*
JAL	yes	*	*	*	no	yes	PC	R31	jabs	npc	nop	*
JR	yes	*	*	*	no	no	*	*	rind	npc	nop	*
JALR	yes	*	*	*	no	yes	PC	R31	rind	npc	nop	*
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{Z=1}	yes	sE ₁₆	*	0?	no	no	*	*	br	npc	nop	*
BEQZ _{Z=0}	no	sE ₁₆	*	0?	no	no	*	*	pc+4			
J	yes	*	*	*	no	no	*	*	jabs	npc	nop	*
JAL	yes	*	*	*	no	yes	PC	R31	jabs	npc	nop	*
JR	yes	*	*	*	no	no	*	*	rind	npc	nop	*
JALR	yes	*	*	*	no	yes	PC	R31	rind	npc	nop	*
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{Z=1}	yes	sE ₁₆	*	0?	no	no	*	*	br	npc	nop	*
BEQZ _{Z=0}	no	sE ₁₆	*	0?	no	no	*	*	pc+4	npc	mem	pc
J	yes	*	*	*	no	no	*	*	jabs	npc	nop	*
JAL	yes	*	*	*	no	yes	PC	R31	jabs	npc	nop	*
JR	yes	*	*	*	no	no	*	*	rind	npc	nop	*
JALR	yes	*	*	*	no	yes	PC	R31	rind	npc	nop	*
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU
 RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

Pipelined Princeton: Control Table

Opcode	Stall	Ext Sel	B Src	Op Sel	Mem W	Reg W	WB Src	Reg Dst	PC Src1	PC Src2	IR Src	MAddr Src
ALU	no	*	Reg	Func	no	yes	ALU	rd	pc+4	npc	mem	pc
ALUi	no	sE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
ALUiu	no	uE ₁₆	Imm	Op	no	yes	ALU	rt	pc+4	npc	mem	pc
LW	yes	sE ₁₆	Imm	+	no	yes	Mem	rt	pc+4	pc	nop	ALU
SW	yes	sE ₁₆	Imm	+	yes	no	*	*	pc+4	pc	nop	ALU
BEQZ _{Z=1}	yes	sE ₁₆	*	0?	no	no	*	*	br	npc	nop	*
BEQZ _{Z=0}	no	sE ₁₆	*	0?	no	no	*	*	pc+4	npc	mem	pc
J	yes	*	*	*	no	no	*	*	jabs	npc	nop	*
JAL	yes	*	*	*	no	yes	PC	R31	jabs	npc	nop	*
JR	yes	*	*	*	no	no	*	*	rind	npc	nop	*
JALR	yes	*	*	*	no	yes	PC	R31	rind	npc	nop	*
NOP	no	*	*	*	no	no	*	*	pc+4	npc	mem	pc

BSrc = Reg / Imm ; WBSrc = ALU / Mem / PC; IRSrc = nop/mem; MAddrSrc = pc/ALU

RegDst = rt / rd / R31; PCSrc1 = pc+4 / br / rind / jabs; PCSrc2 = pc/nPC

stall & IRSrc columns are identical

Pipelined Princeton Architecture

Clock: $t_{\text{C-Princeton}} > t_{\text{RF}} + t_{\text{ALU}} + t_{\text{M}} + t_{\text{WB}}$

CPI: $(1 - f) + 2f$ cycles per instruction
where f is the fraction of
instructions that cause a stall

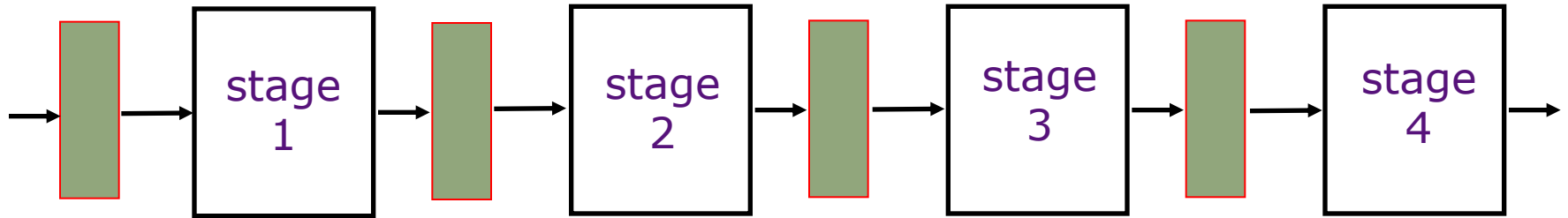
Pipelined Princeton Architecture

Clock: $t_{\text{C-Princeton}} > t_{\text{RF}} + t_{\text{ALU}} + t_{\text{M}} + t_{\text{WB}}$

CPI: $(1 - f) + 2f$ cycles per instruction
where f is the fraction of
instructions that cause a stall

What is a likely value of f ?

An Ideal Pipeline

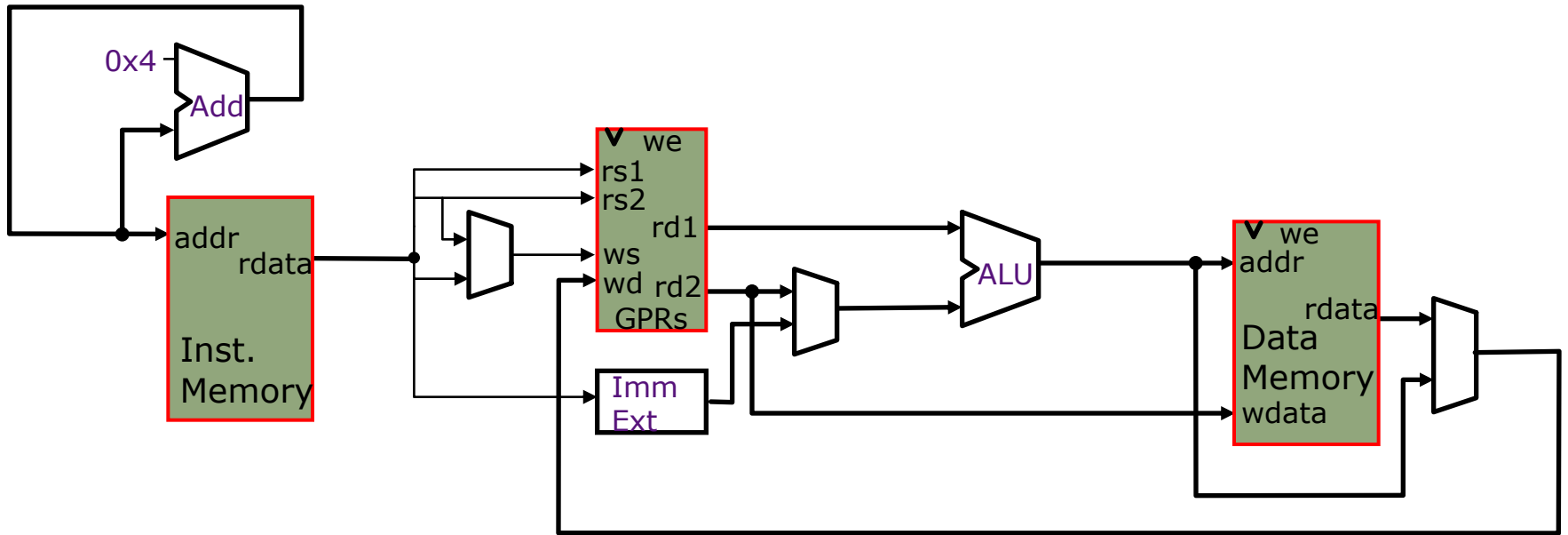


- All objects go through the same stages
- No sharing of resources between any two stages
- Propagation delay through all pipeline stages is equal
- The scheduling of an object entering the pipeline is not affected by the objects in other stages

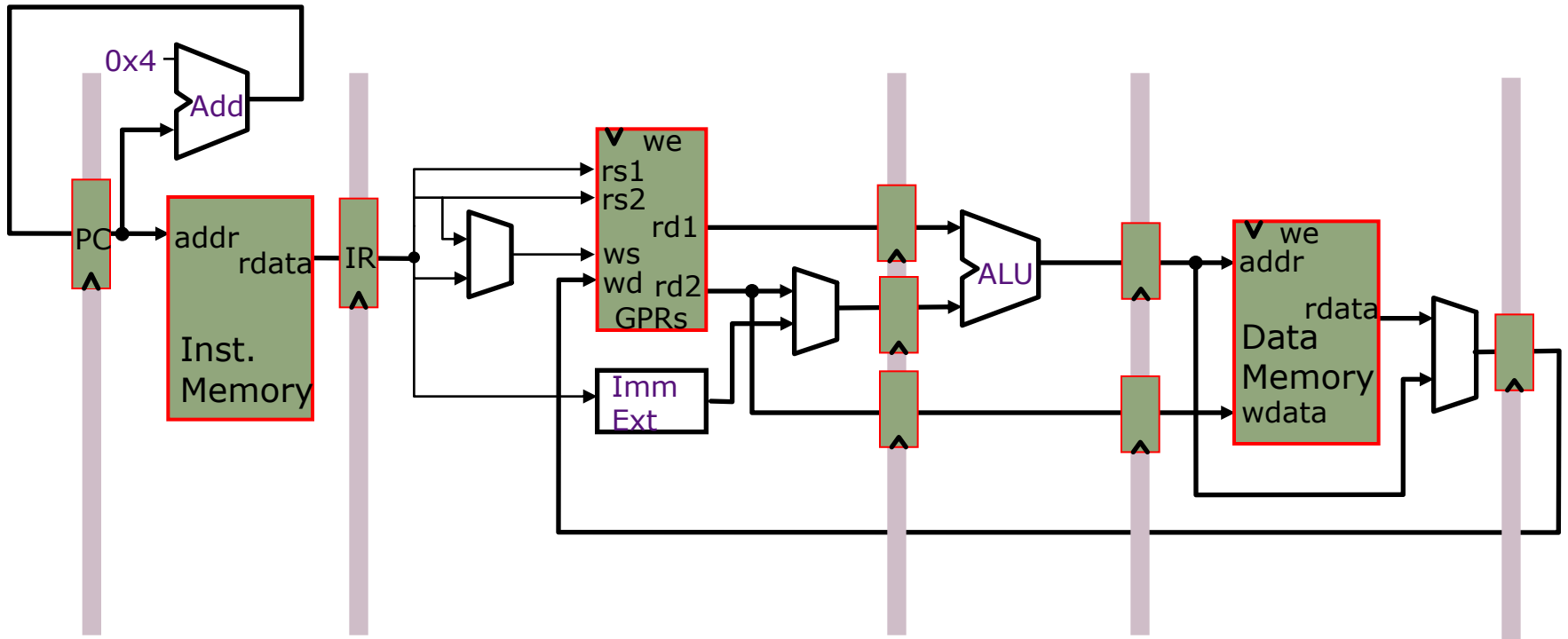
These conditions generally hold for industrial assembly lines.

But what about an instruction pipeline?

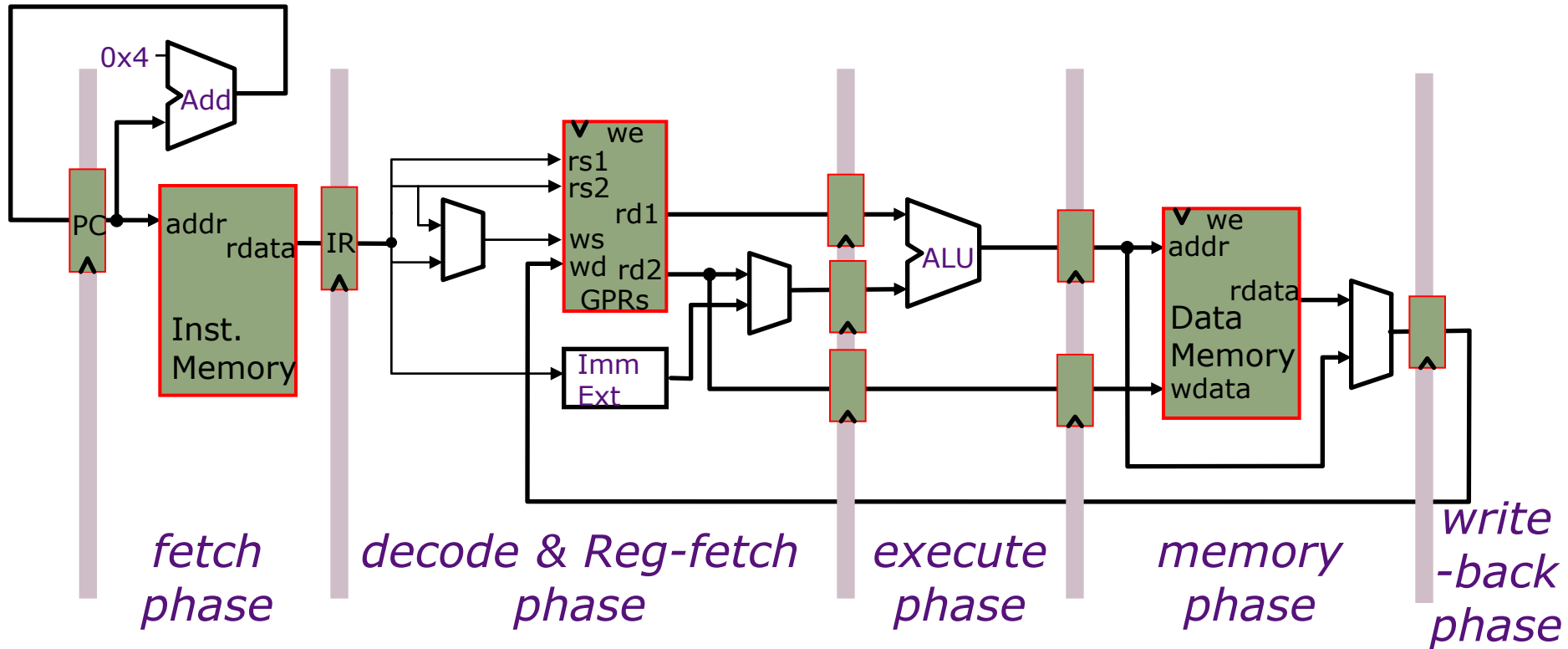
Pipelined Datapath



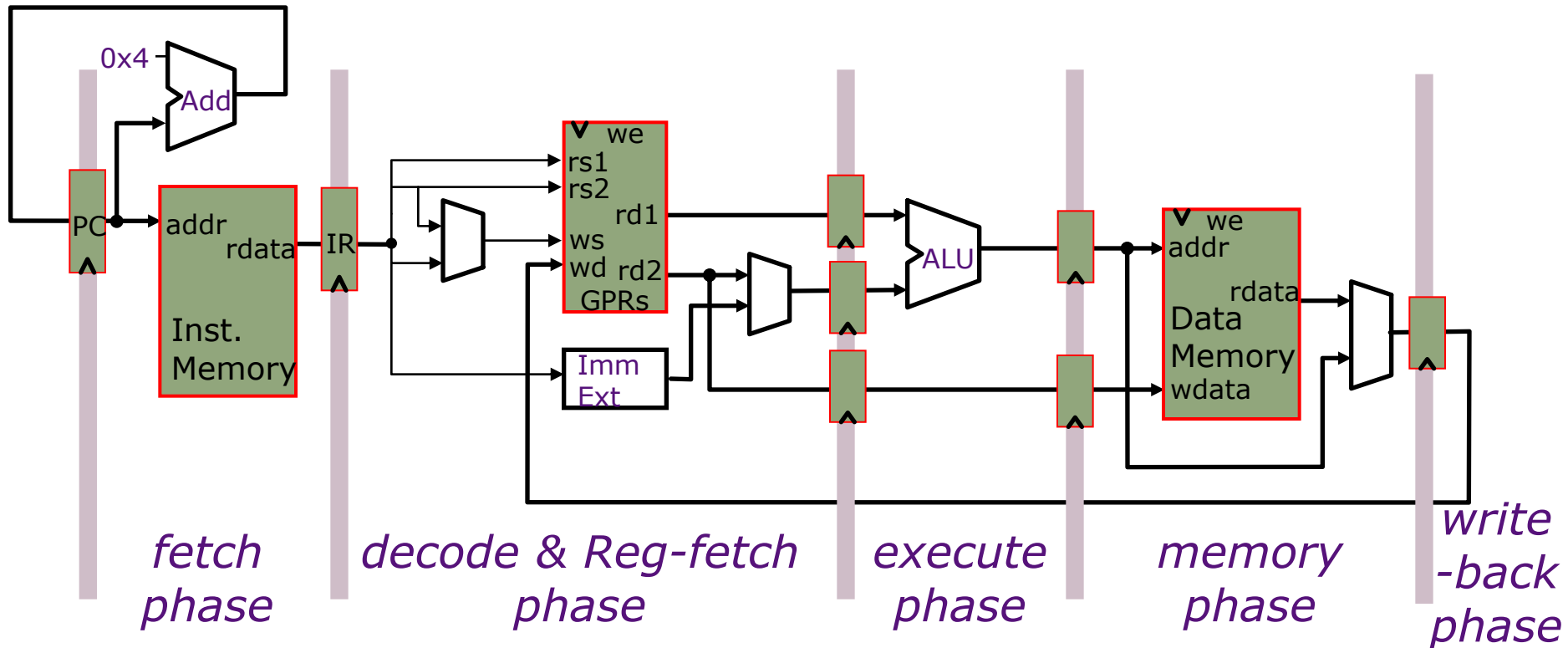
Pipelined Datapath



Pipelined Datapath



Pipelined Datapath



Clock period can be reduced by dividing the execution of an instruction into multiple cycles

$$t_c > \max \{t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW}\} \quad (= t_{DM} \text{ probably})$$

However, CPI will increase unless instructions are pipelined

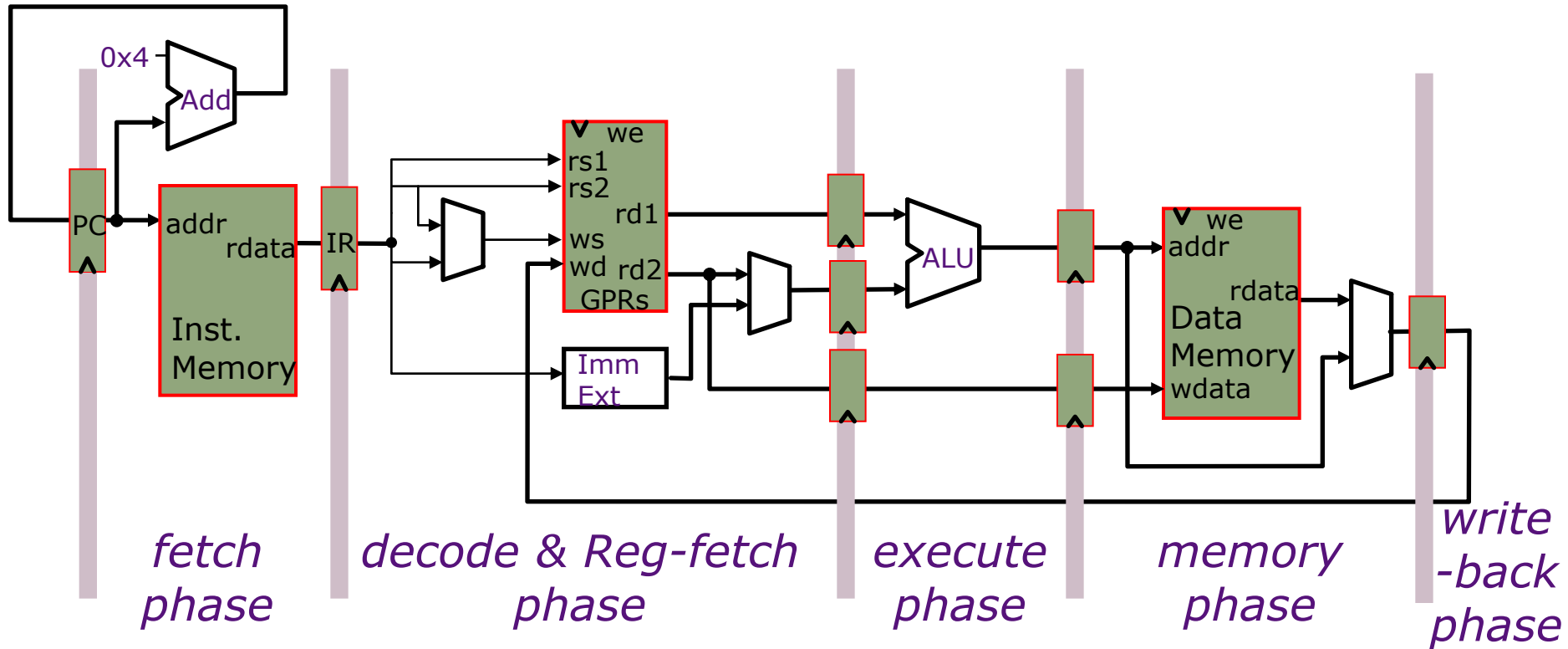
How to divide datapath into stages

Suppose memory is significantly slower than other stages. For example, suppose

$$\begin{aligned}t_{IM} &= 10 \text{ units} \\t_{DM} &= 10 \text{ units} \\t_{ALU} &= 5 \text{ units} \\t_{RF} &= 1 \text{ unit} \\t_{RW} &= 1 \text{ unit}\end{aligned}$$

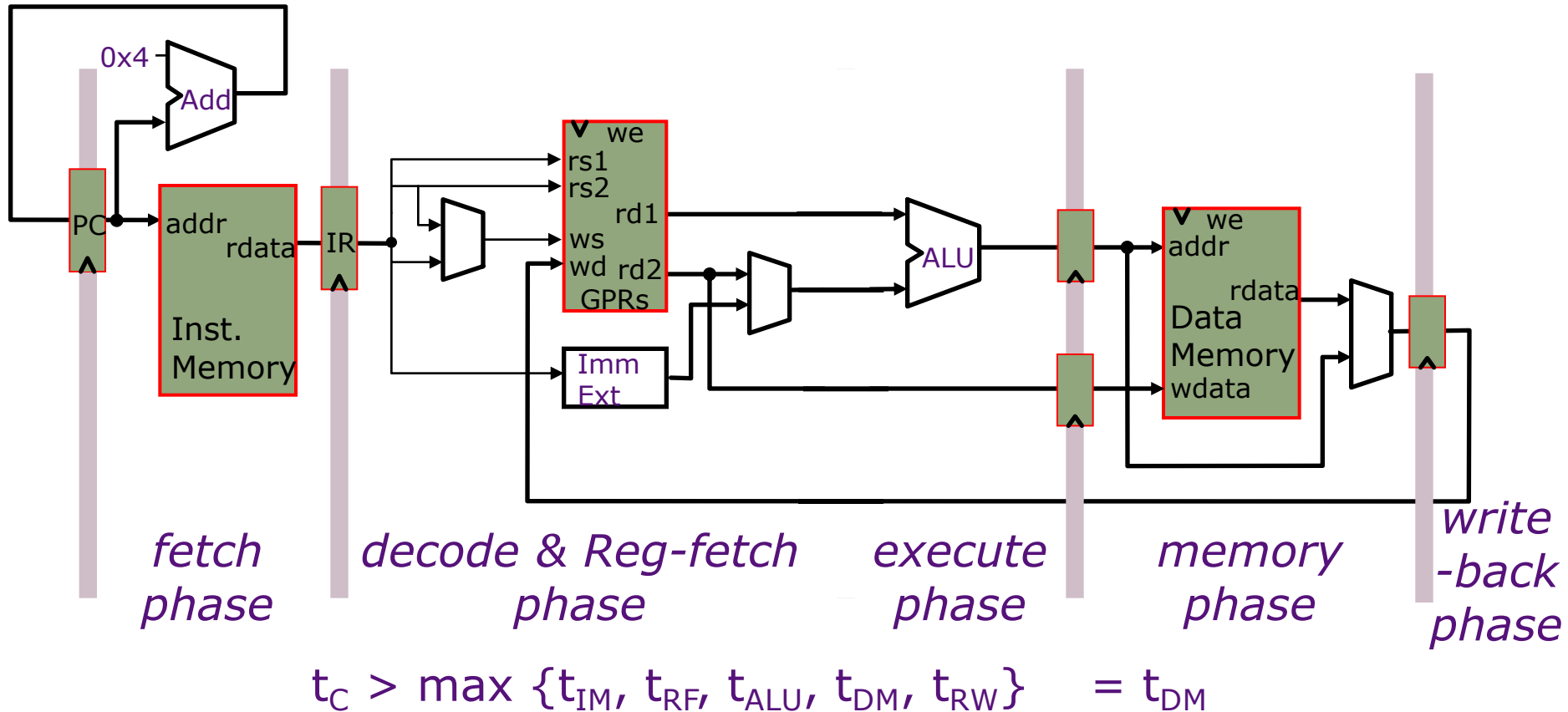
Since the slowest stage determines the clock, it may be possible to combine some stages without any loss of performance

Alternative Pipelining

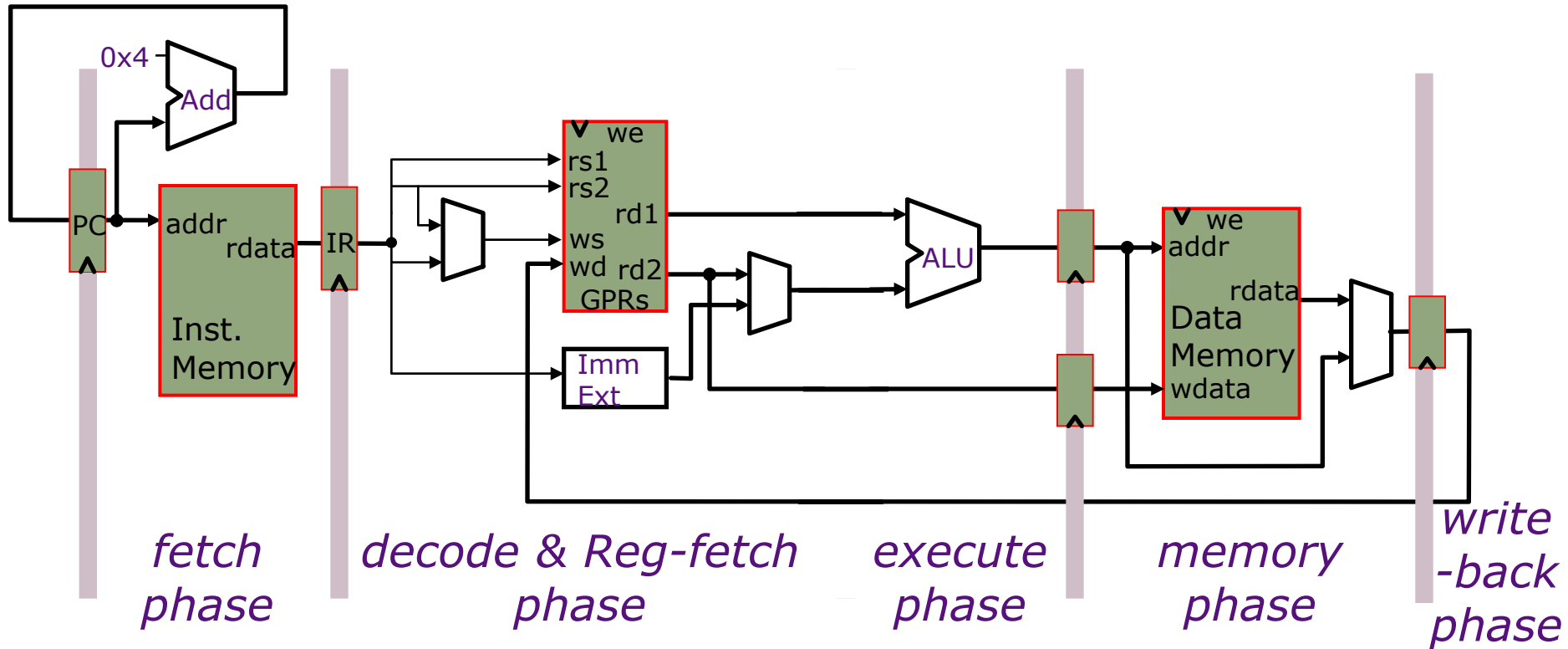


$$t_C > \max \{t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW}\} = t_{DM}$$

Alternative Pipelining

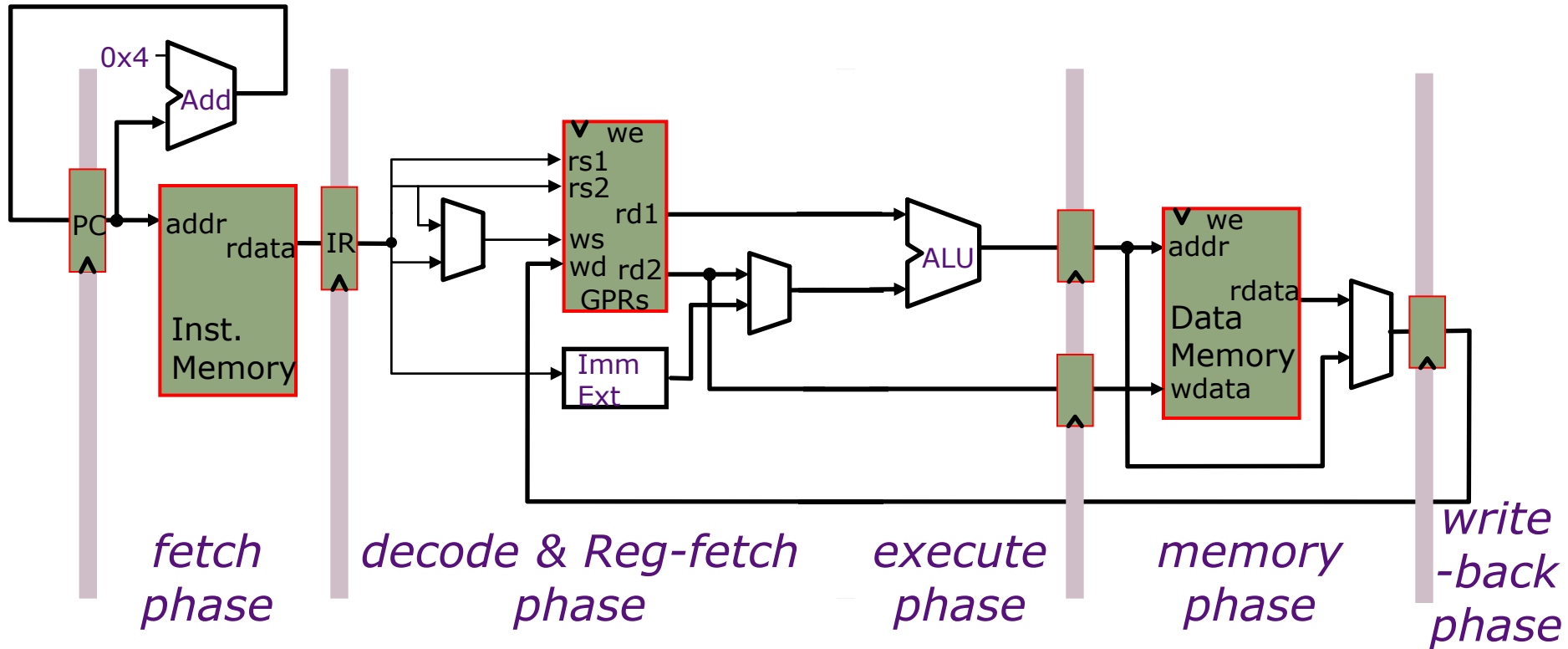


Alternative Pipelining



$$t_C > \max \{t_{IM}, t_{RF} + t_{ALU}, t_{DM}, t_{RW}\} = t_{DM}$$

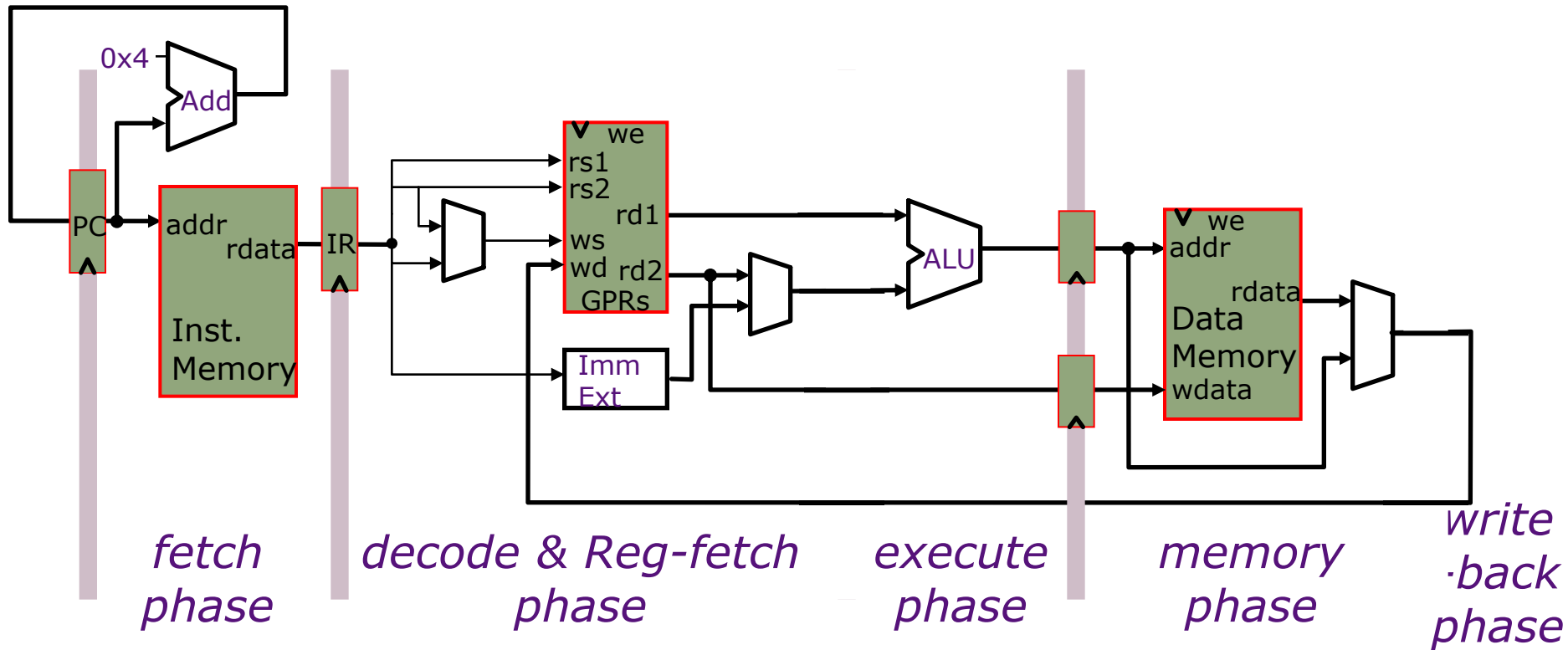
Alternative Pipelining



$$t_C > \max \{t_{IM}, t_{RF} + t_{ALU}, t_{DM}, t_{RW}\} = t_{DM}$$

Write-back stage takes much less time than other stages.
Suppose we combined it with the memory phase

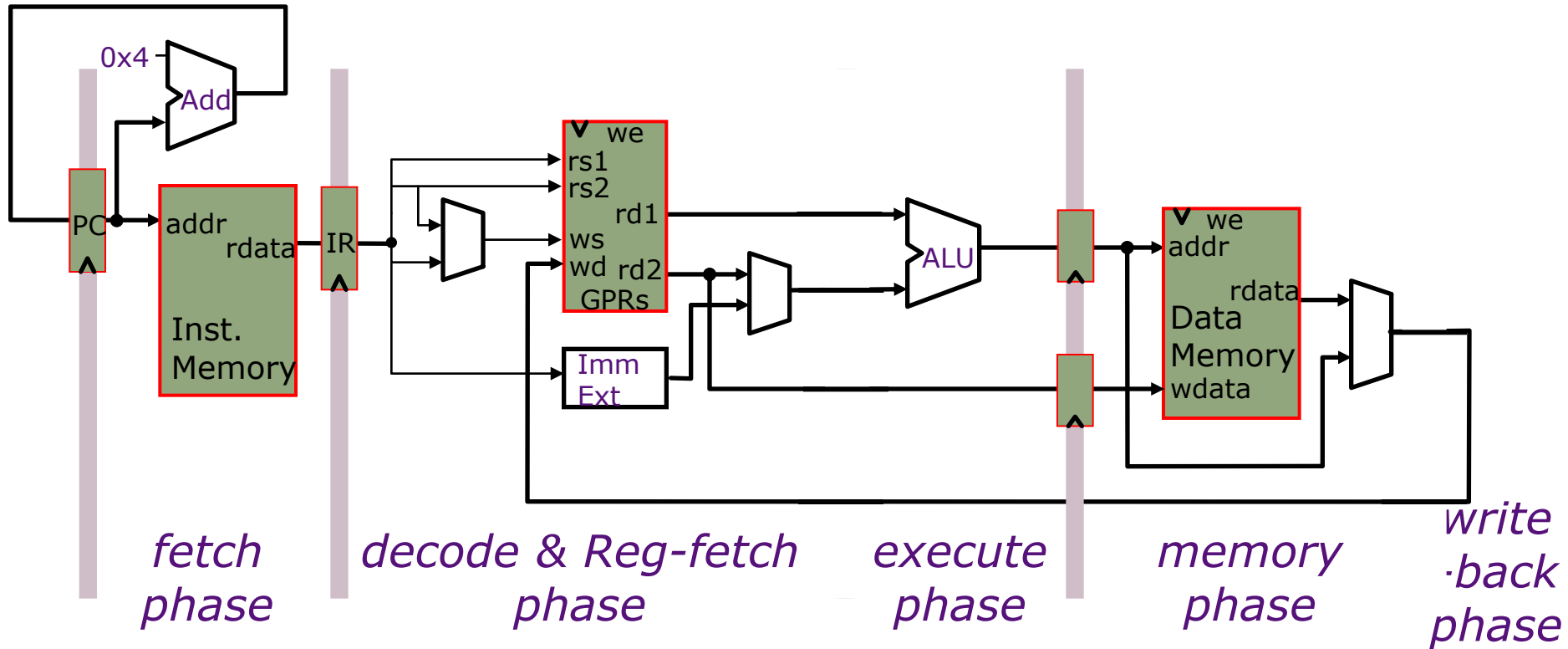
Alternative Pipelining



$$t_C > \max \{t_{IM}, t_{RF} + t_{ALU}, t_{DM}, t_{RW}\} = t_{DM}$$

Write-back stage takes much less time than other stages.
Suppose we combined it with the memory phase

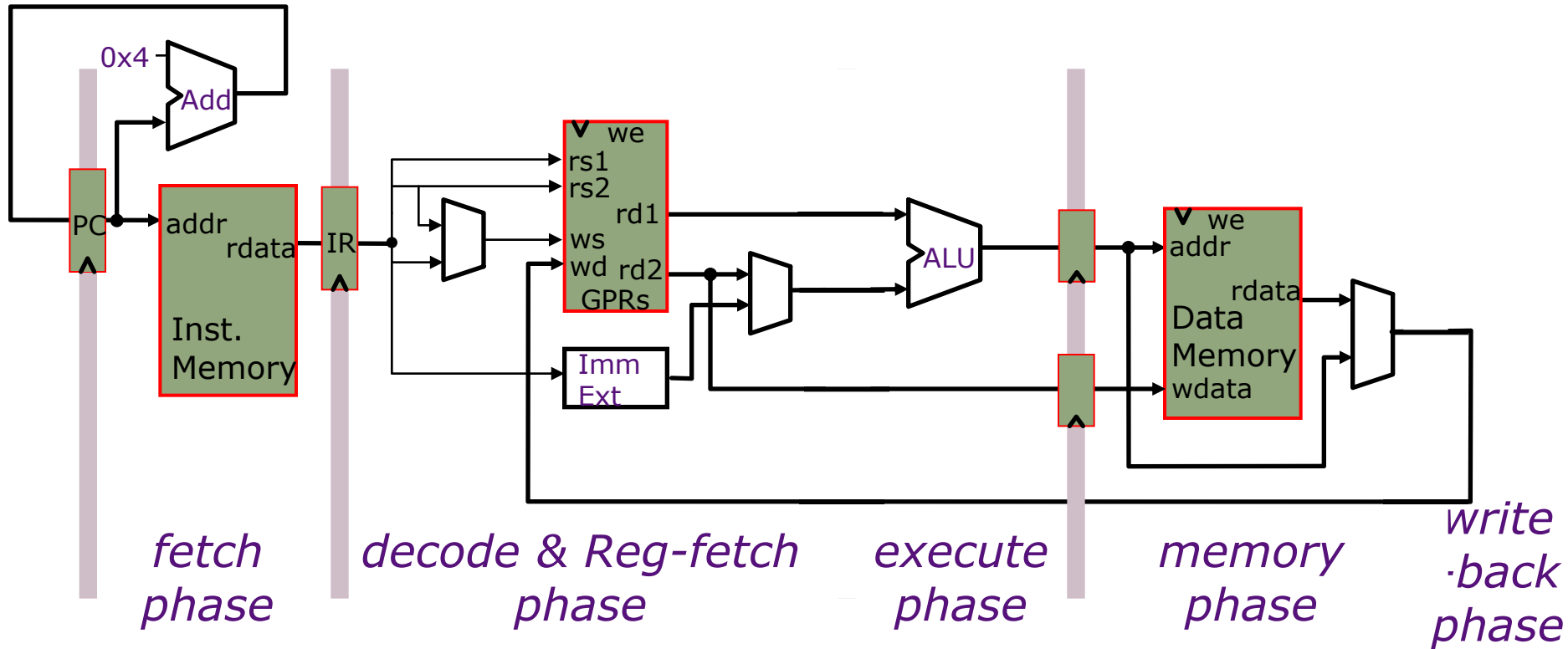
Alternative Pipelining



$$t_C > \max \{t_{IM}, t_{RF}+t_{ALU}, t_{DM}+t_{RW}\} = t_{DM} + t_{RW}$$

Write-back stage takes much less time than other stages.
Suppose we combined it with the memory phase

Alternative Pipelining



$$t_C > \max \{t_{IM}, t_{RF}+t_{ALU}, t_{DM}+t_{RW}\} = t_{DM} + t_{RW}$$

⇒ increase the critical path by 10%

Write-back stage takes much less time than other stages.
Suppose we combined it with the memory phase

Maximum Speedup by Pipelining

Assumptions

Unpipelined

t_c

Pipelined Speedup

t_c

Maximum Speedup by Pipelining

Assumptions

1. $t_{IM} = t_{DM} = 10,$
 $t_{ALU} = 5,$
 $t_{RF} = t_{RW} = 1$
4-stage pipeline

Unpipelined

t_c

Pipelined Speedup

t_c

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined Speedup t_c
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline		27

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined Speedup t_c
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline			

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25		

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	2.5

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	2.5
3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 5-stage pipeline			

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	2.5
3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 5-stage pipeline	25		

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	2.5
3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 5-stage pipeline	25	5	5

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	2.5
3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 5-stage pipeline	25	5	5.0

Maximum Speedup by Pipelining

Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	2.5
3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 5-stage pipeline	25	5	5.0

What seems to be the message here?

Maximum Speedup by Pipelining

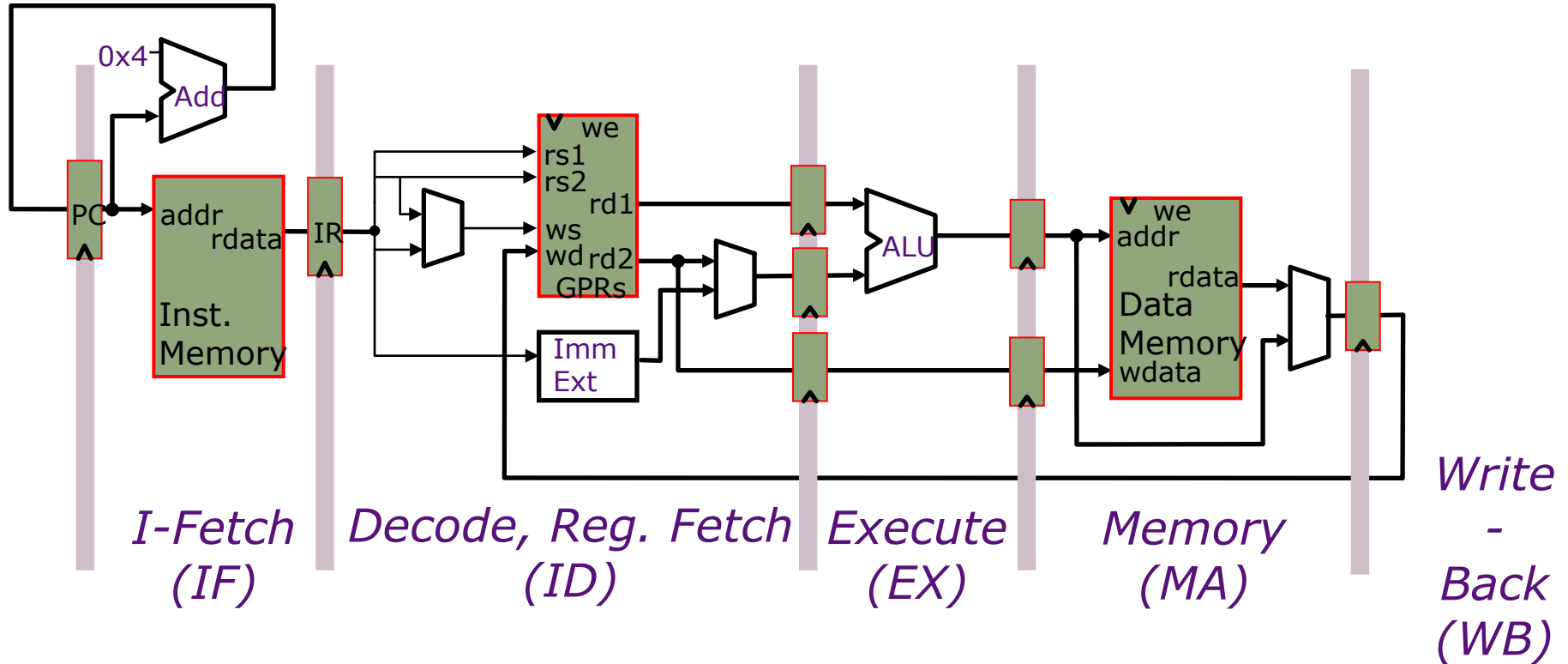
Assumptions	Unpipelined t_c	Pipelined t_c	Speedup
1. $t_{IM} = t_{DM} = 10,$ $t_{ALU} = 5,$ $t_{RF} = t_{RW} = 1$ 4-stage pipeline	27	10	2.7
2. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 4-stage pipeline	25	10	2.5
3. $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ 5-stage pipeline	25	5	5.0

What seems to be the message here?

One can achieve higher speedup with more pipeline stages

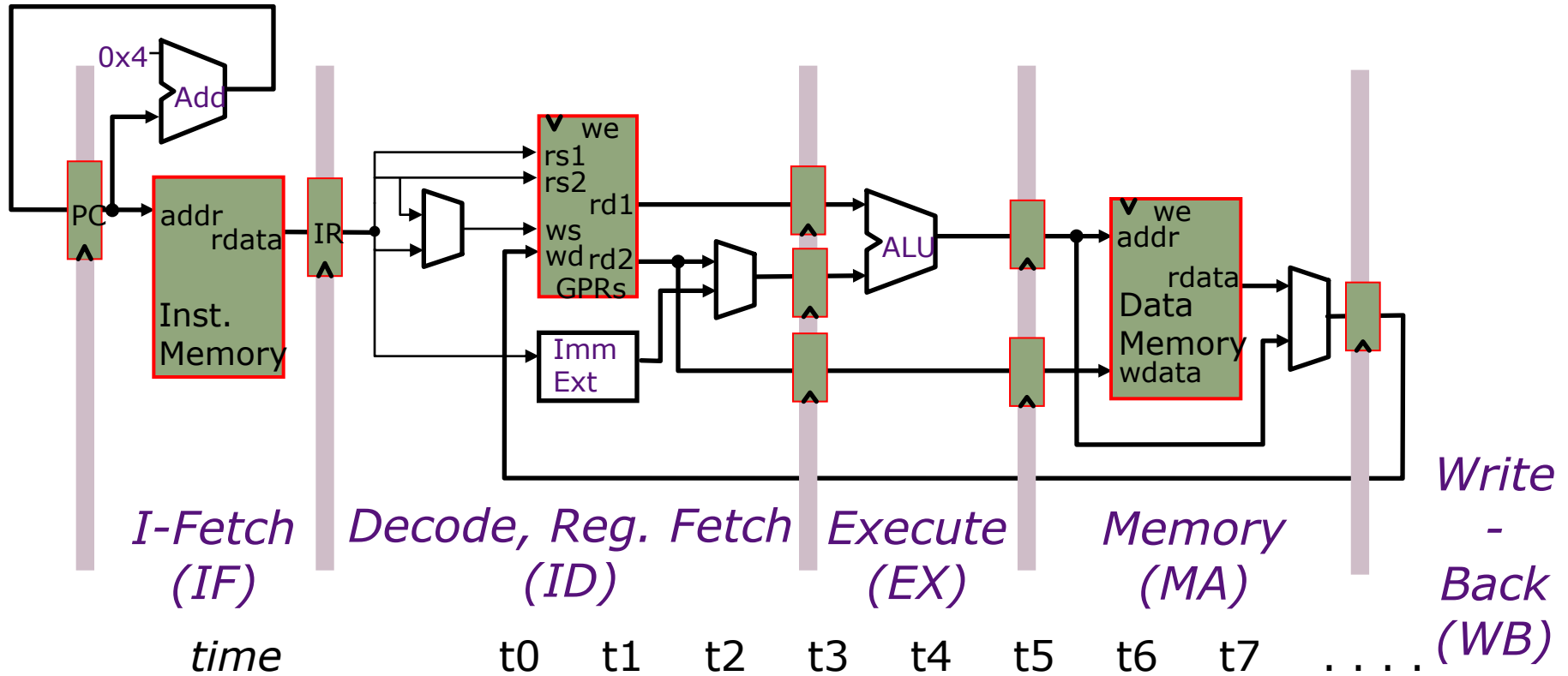
5-Stage Pipelined Execution

Instruction Flow Diagram



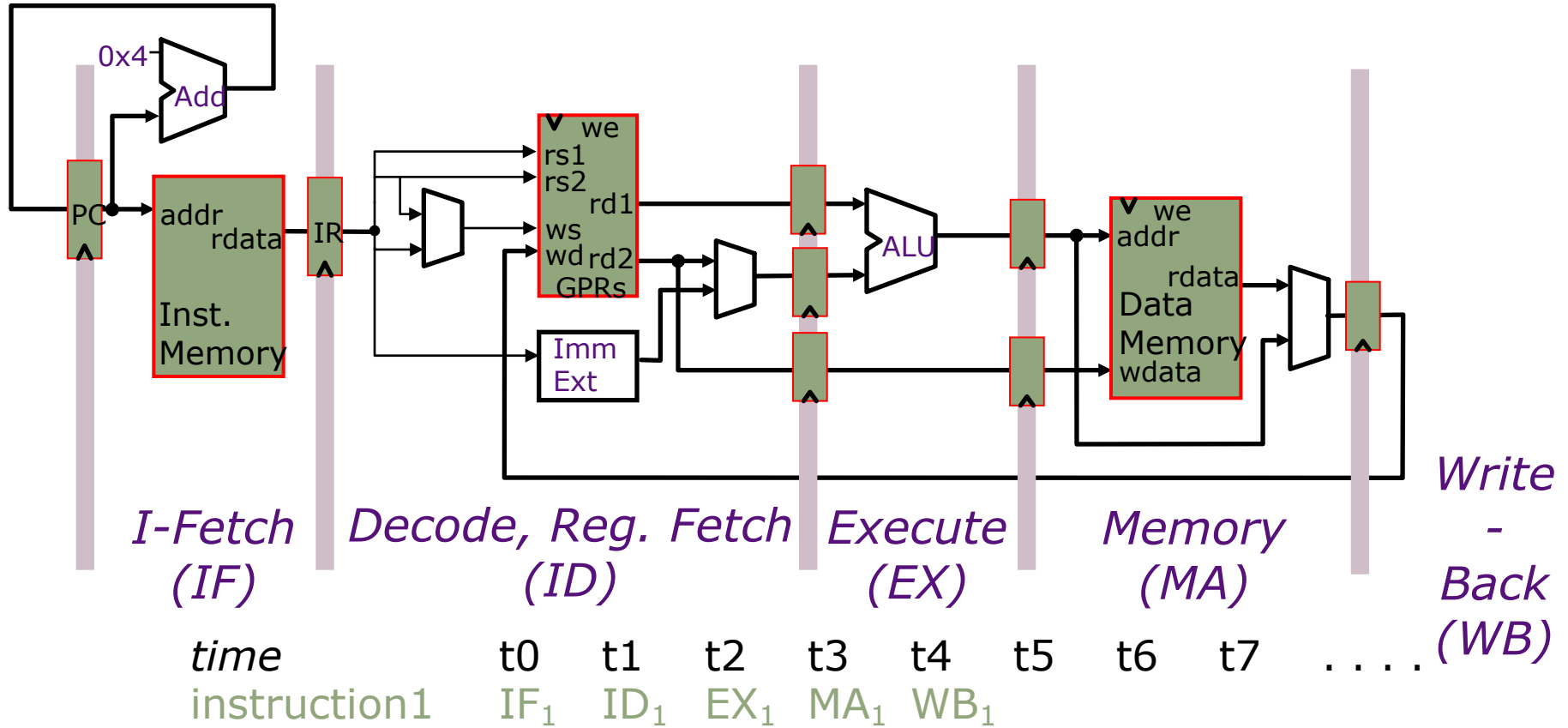
5-Stage Pipelined Execution

Instruction Flow Diagram



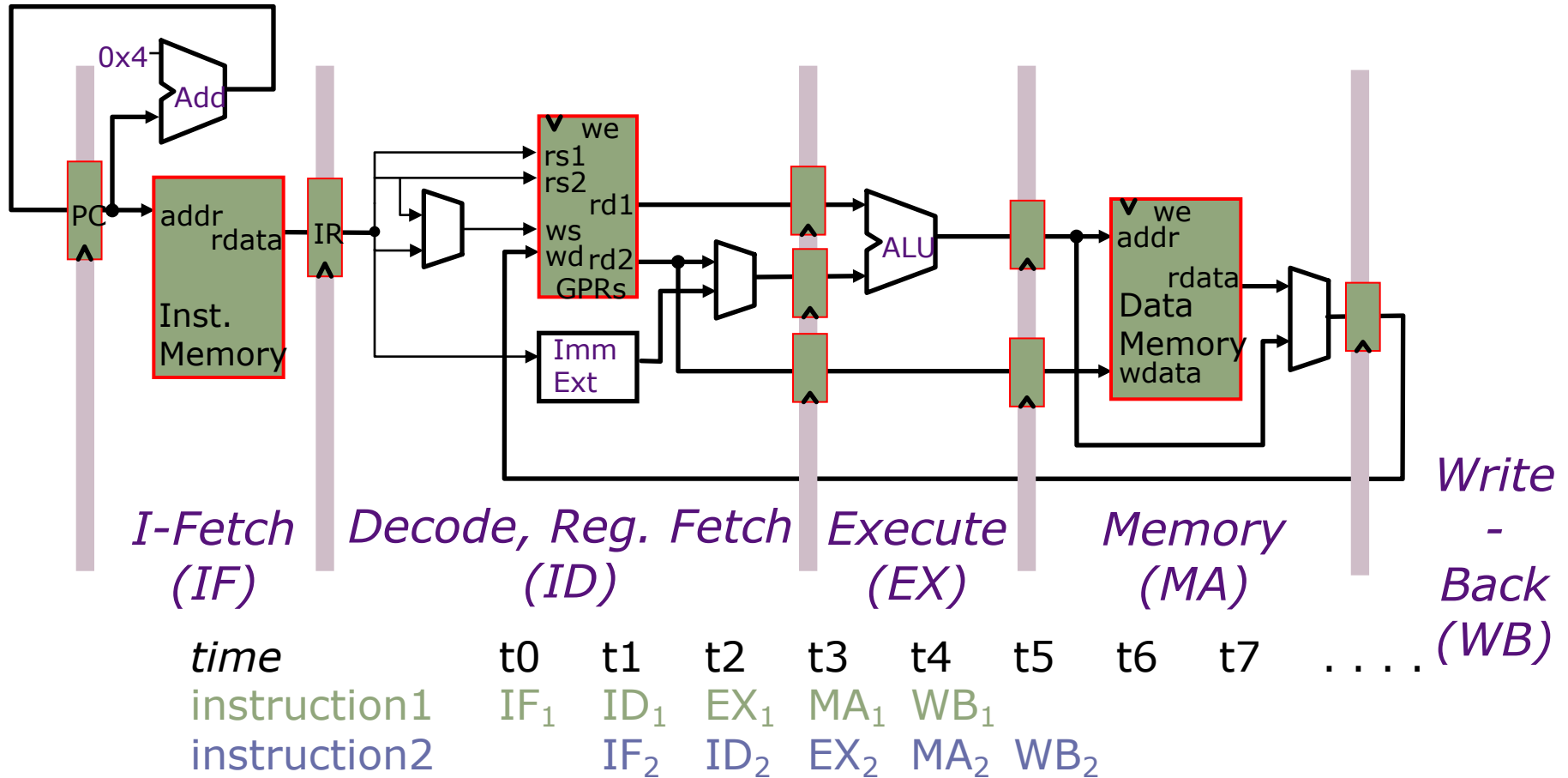
5-Stage Pipelined Execution

Instruction Flow Diagram



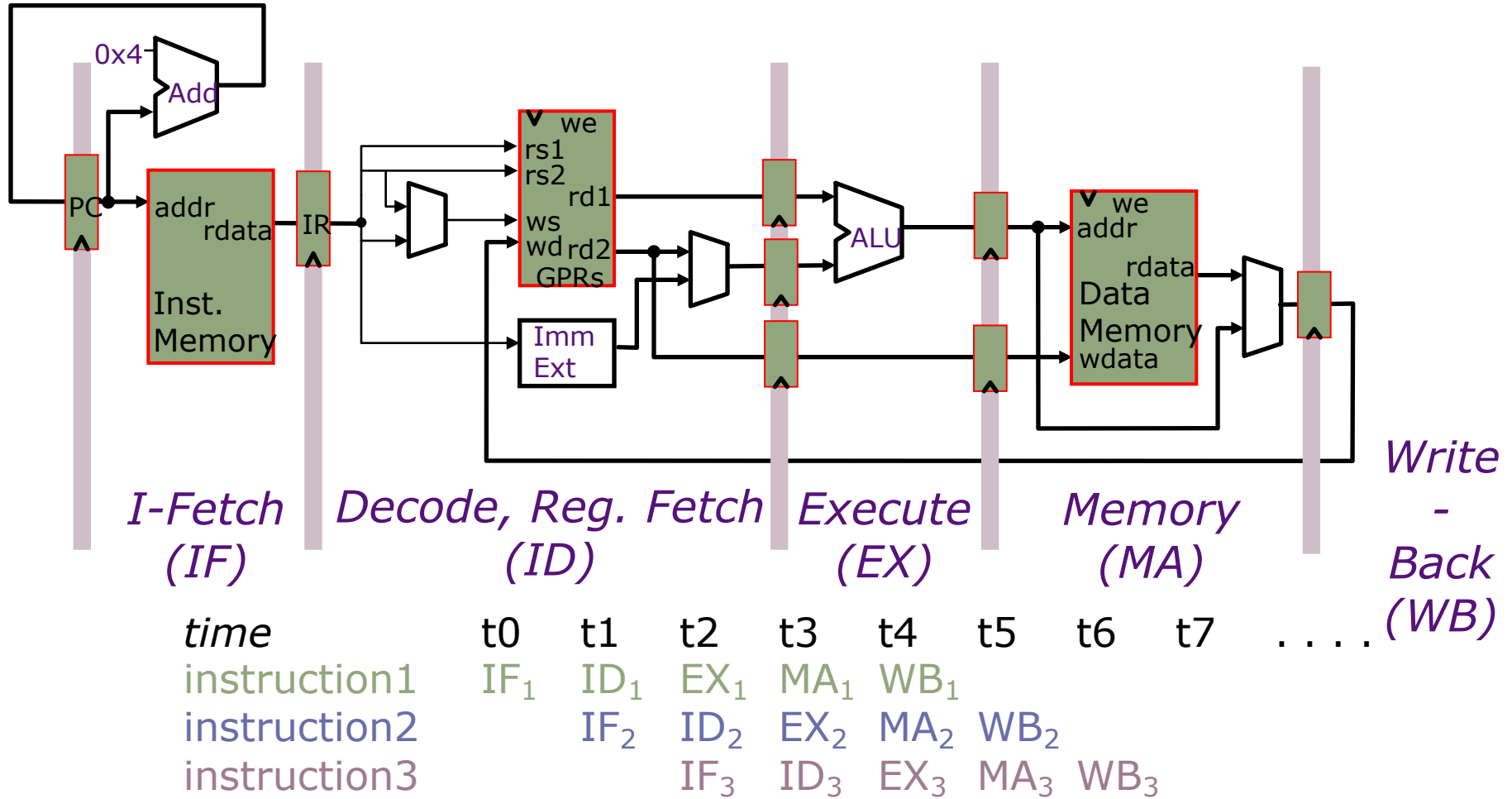
5-Stage Pipelined Execution

Instruction Flow Diagram



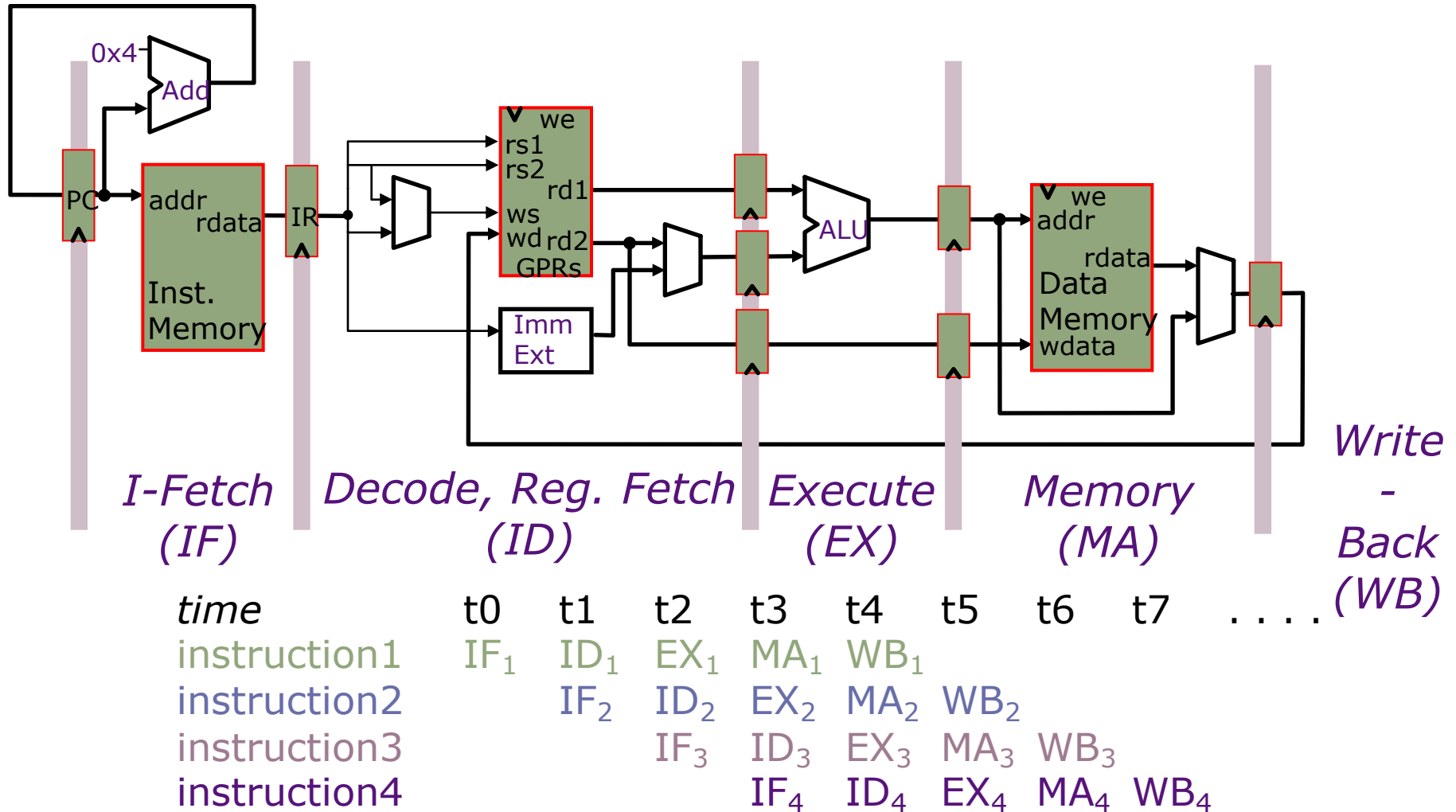
5-Stage Pipelined Execution

Instruction Flow Diagram



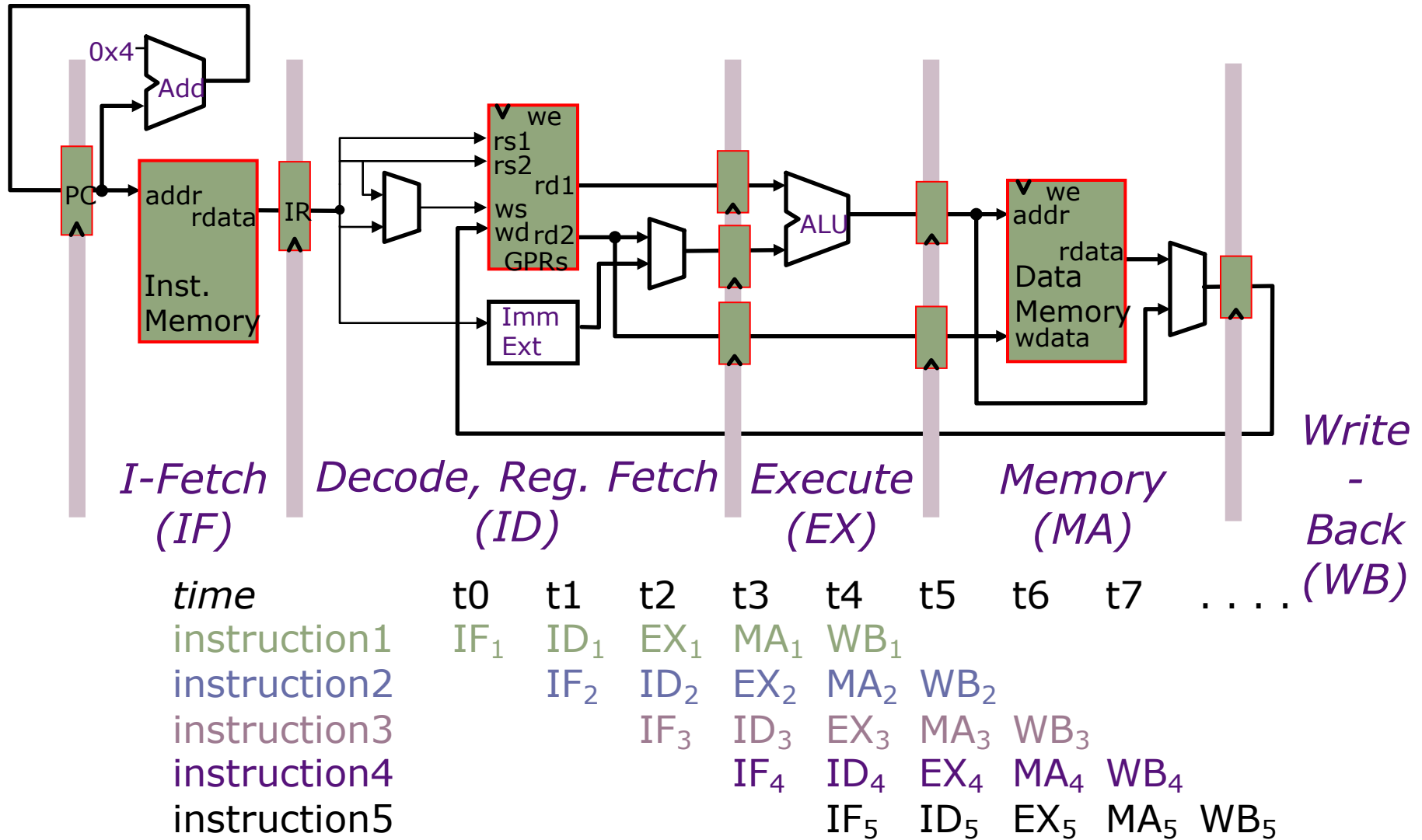
5-Stage Pipelined Execution

Instruction Flow Diagram



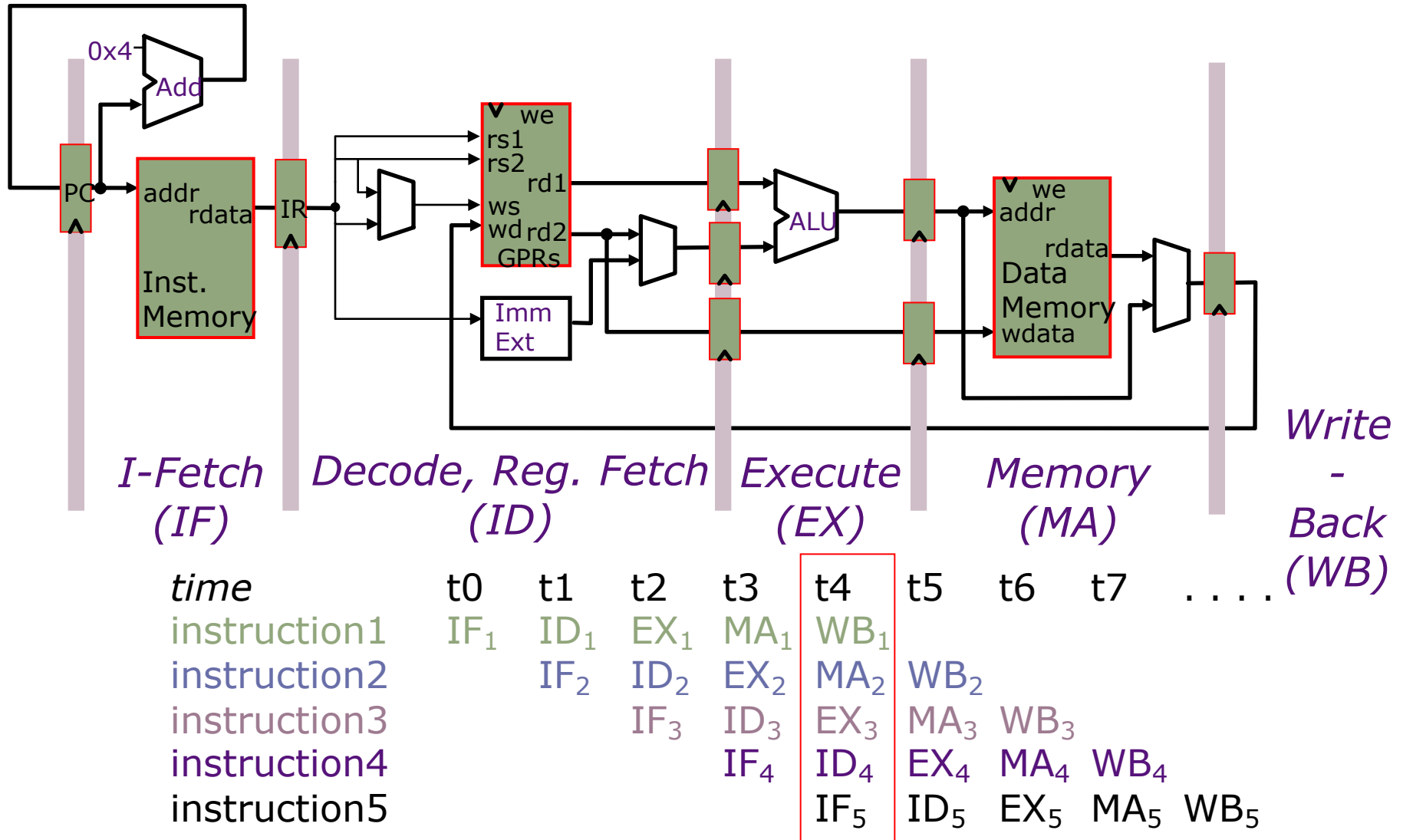
5-Stage Pipelined Execution

Instruction Flow Diagram



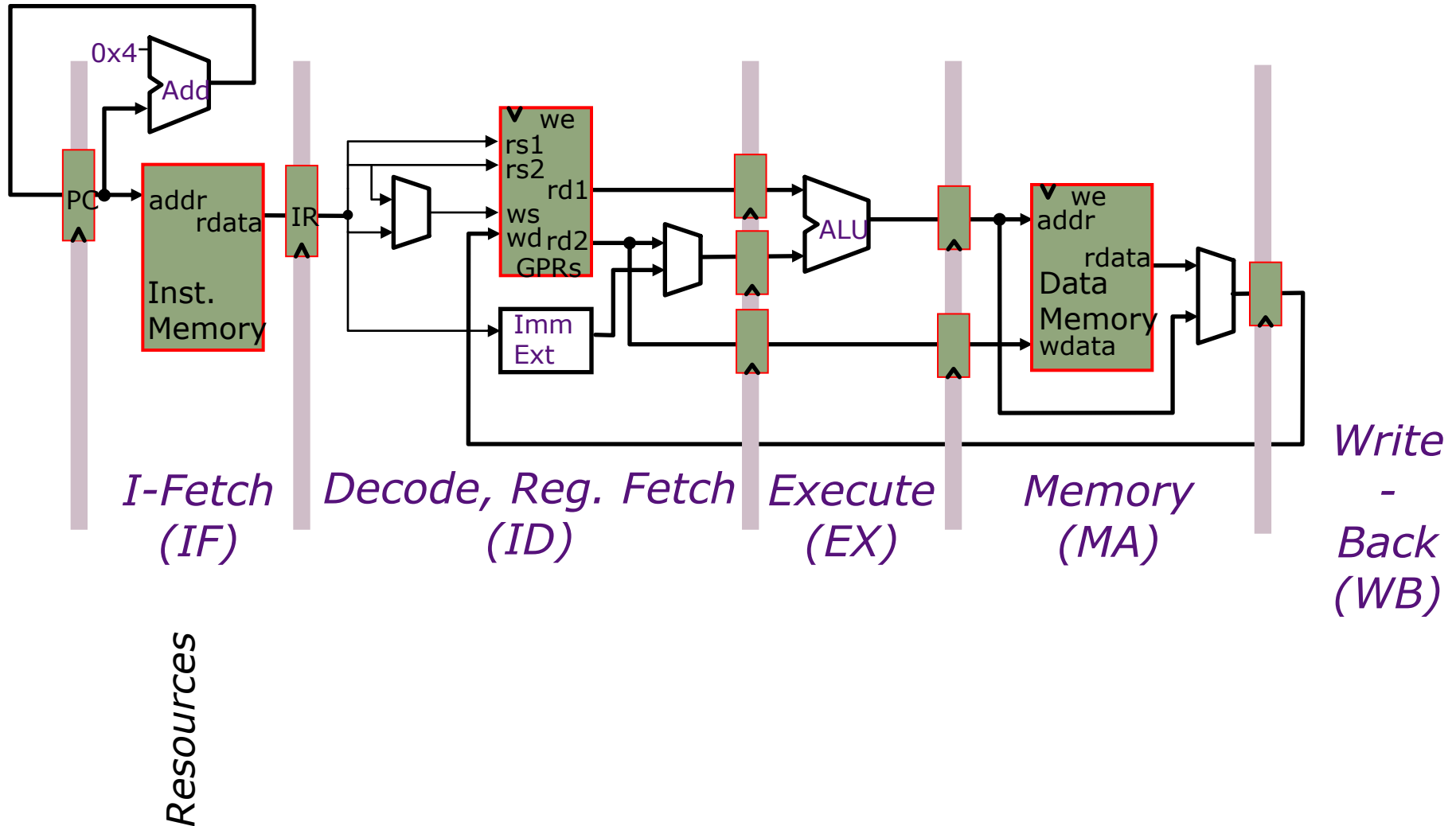
5-Stage Pipelined Execution

Instruction Flow Diagram



5-Stage Pipelined Execution

Resource Usage Diagram

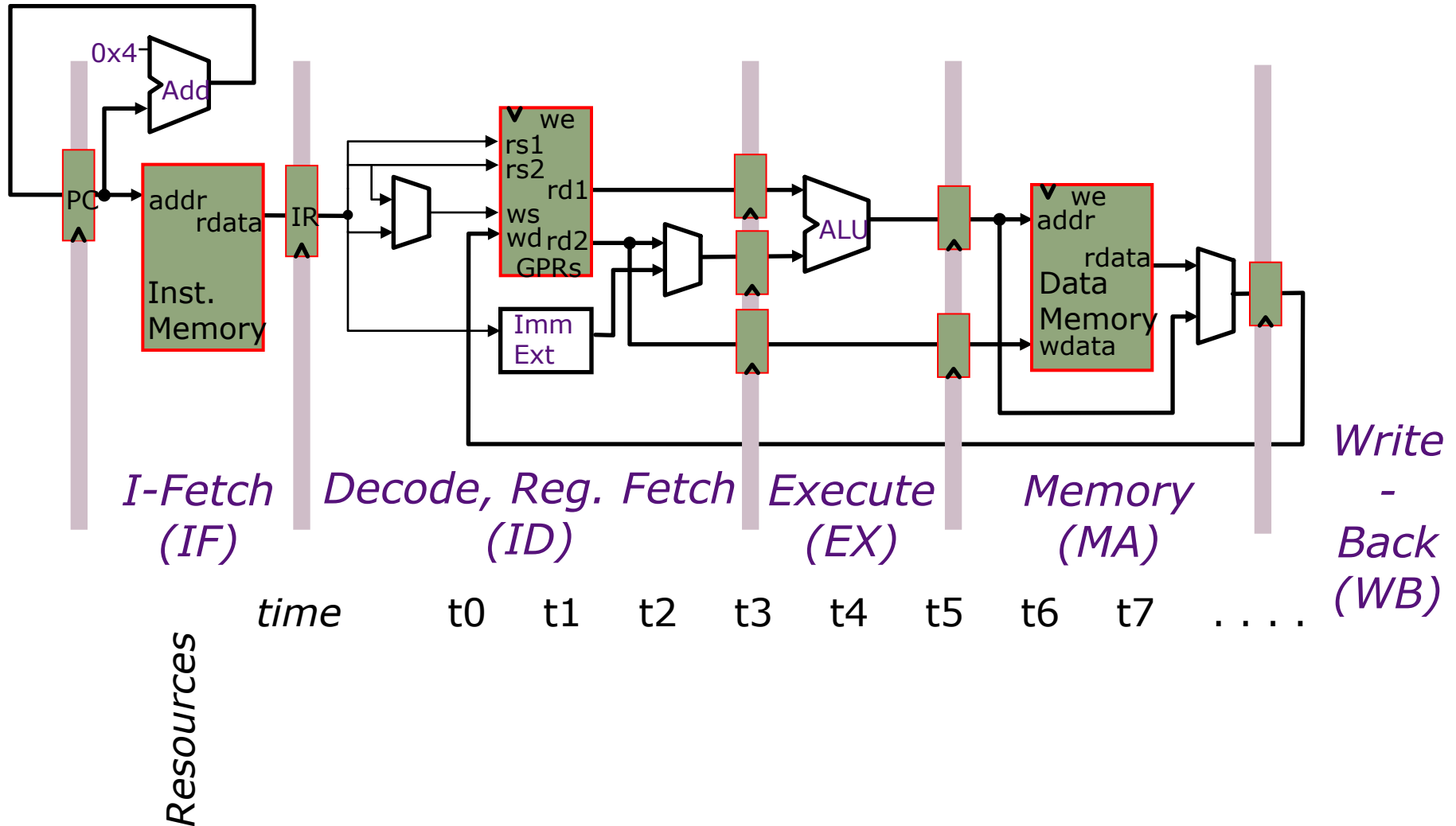


*Write
-
Back
(WB)*

Resources

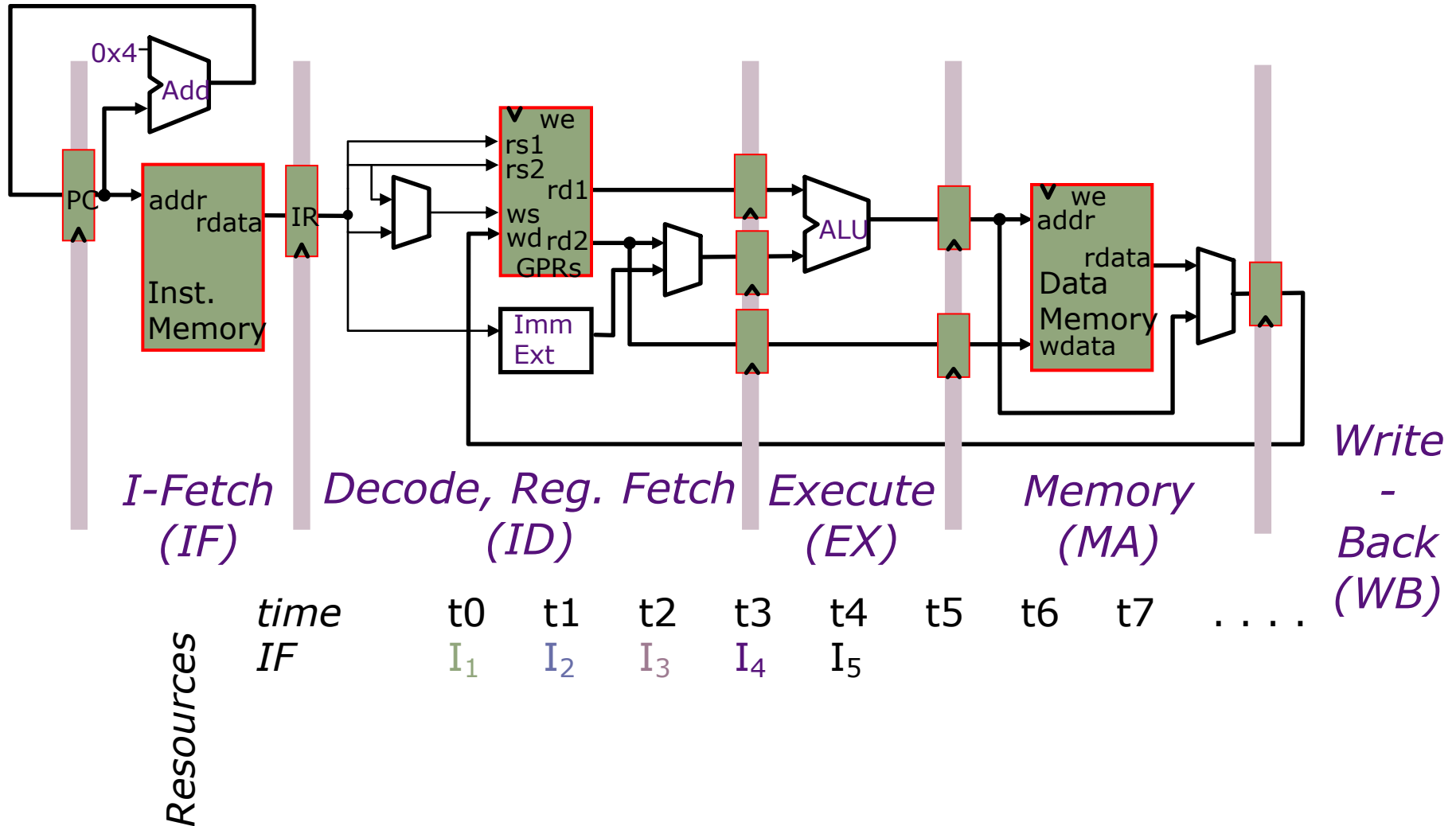
5-Stage Pipelined Execution

Resource Usage Diagram



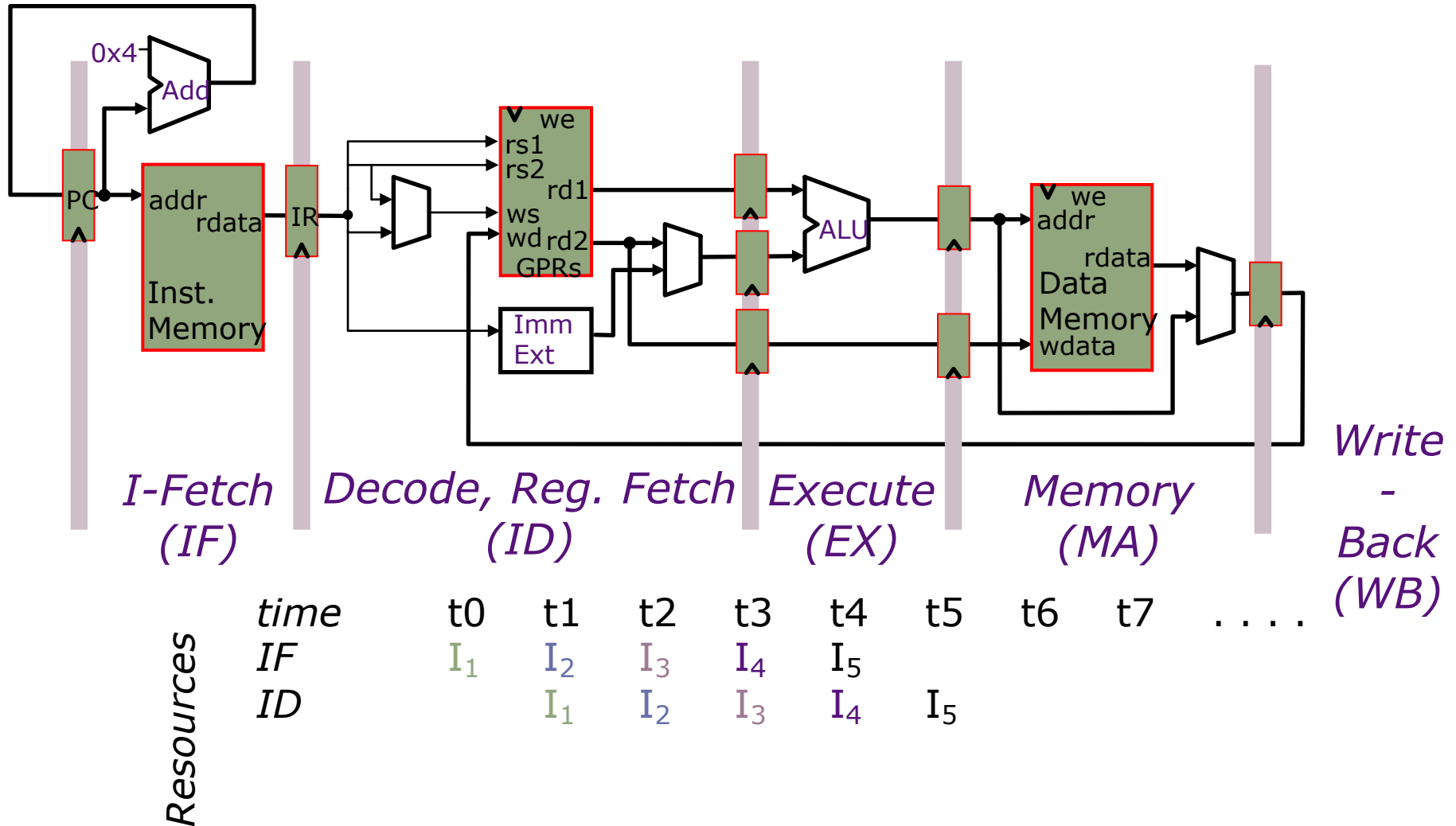
5-Stage Pipelined Execution

Resource Usage Diagram



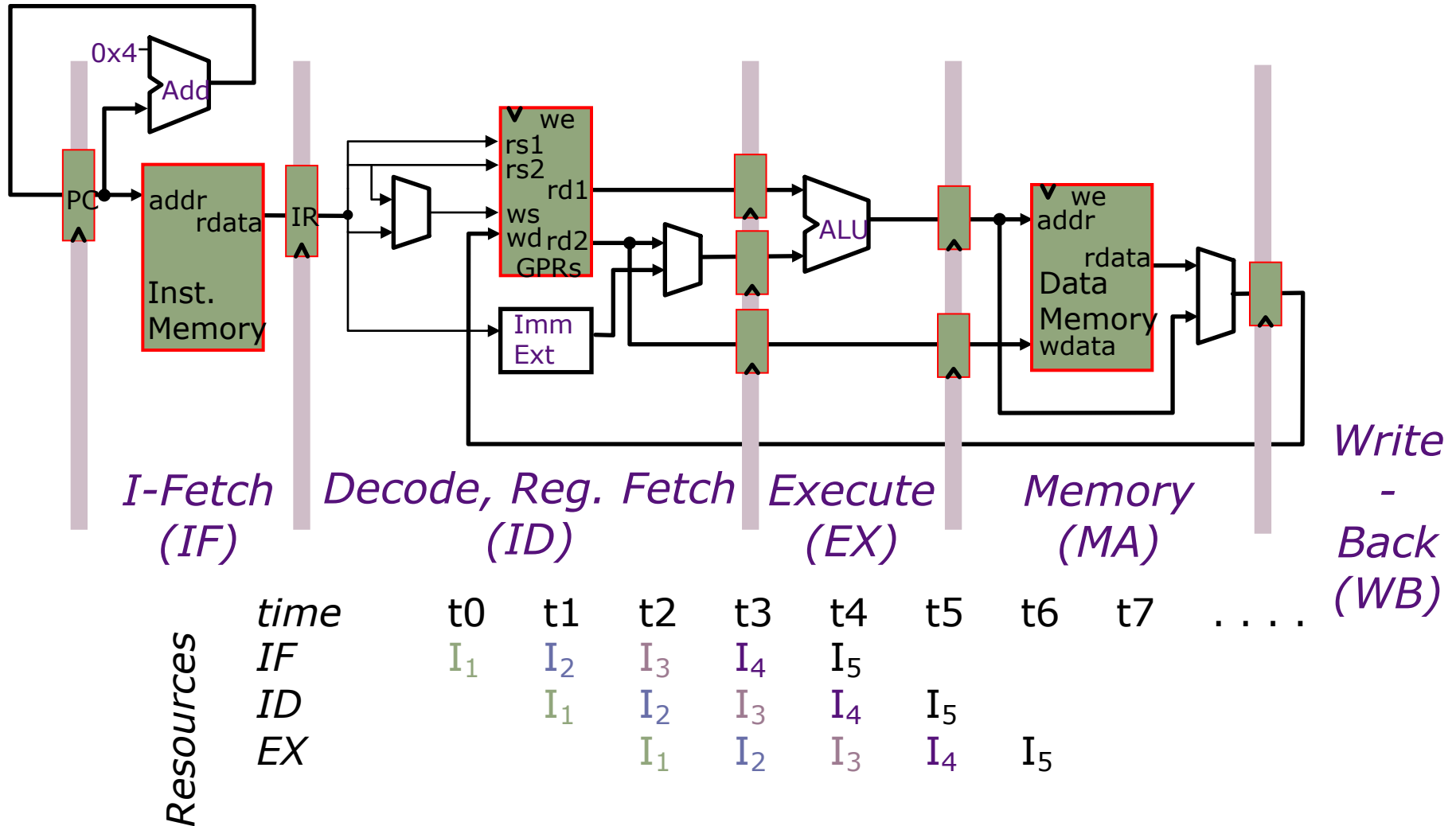
5-Stage Pipelined Execution

Resource Usage Diagram



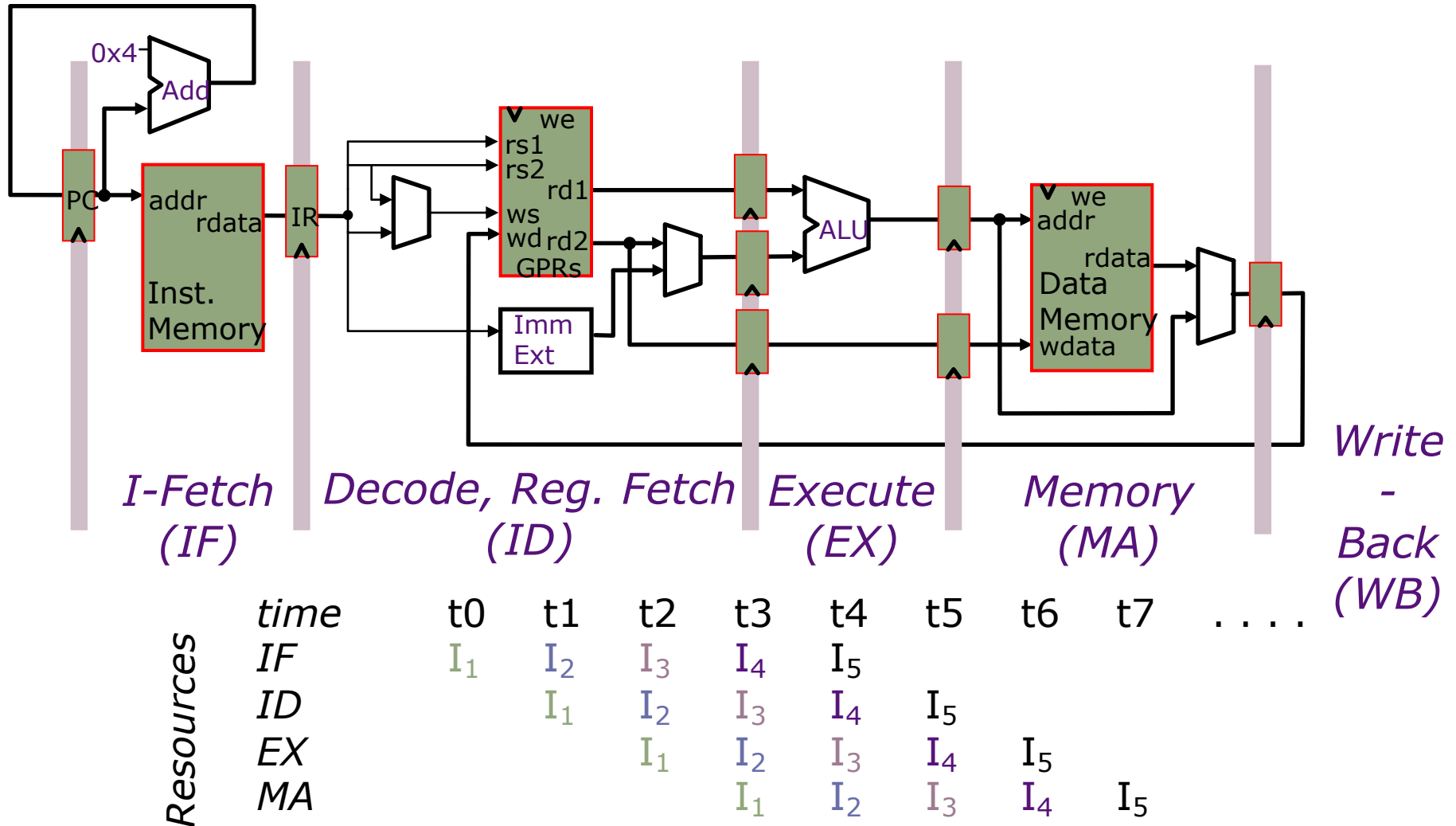
5-Stage Pipelined Execution

Resource Usage Diagram



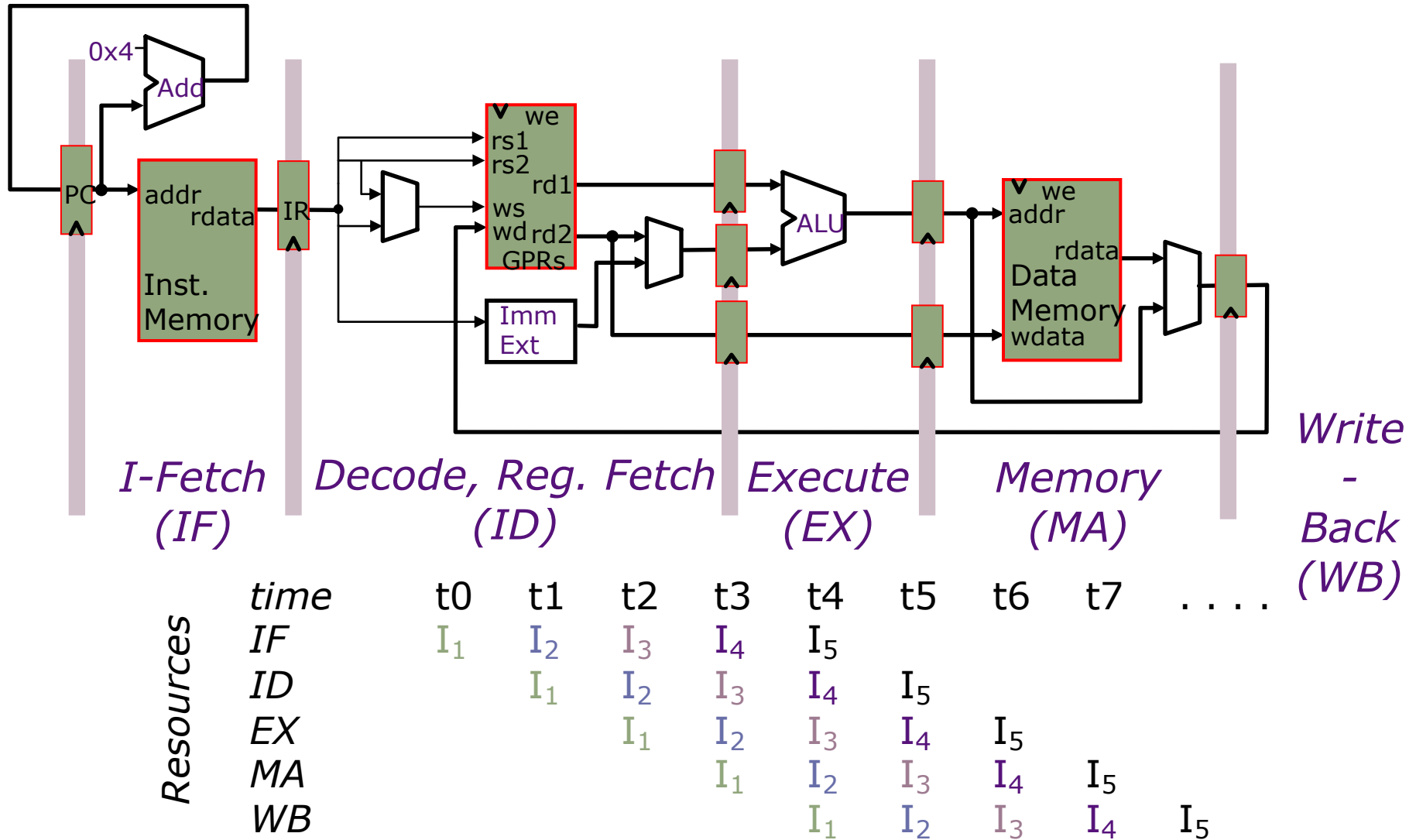
5-Stage Pipelined Execution

Resource Usage Diagram



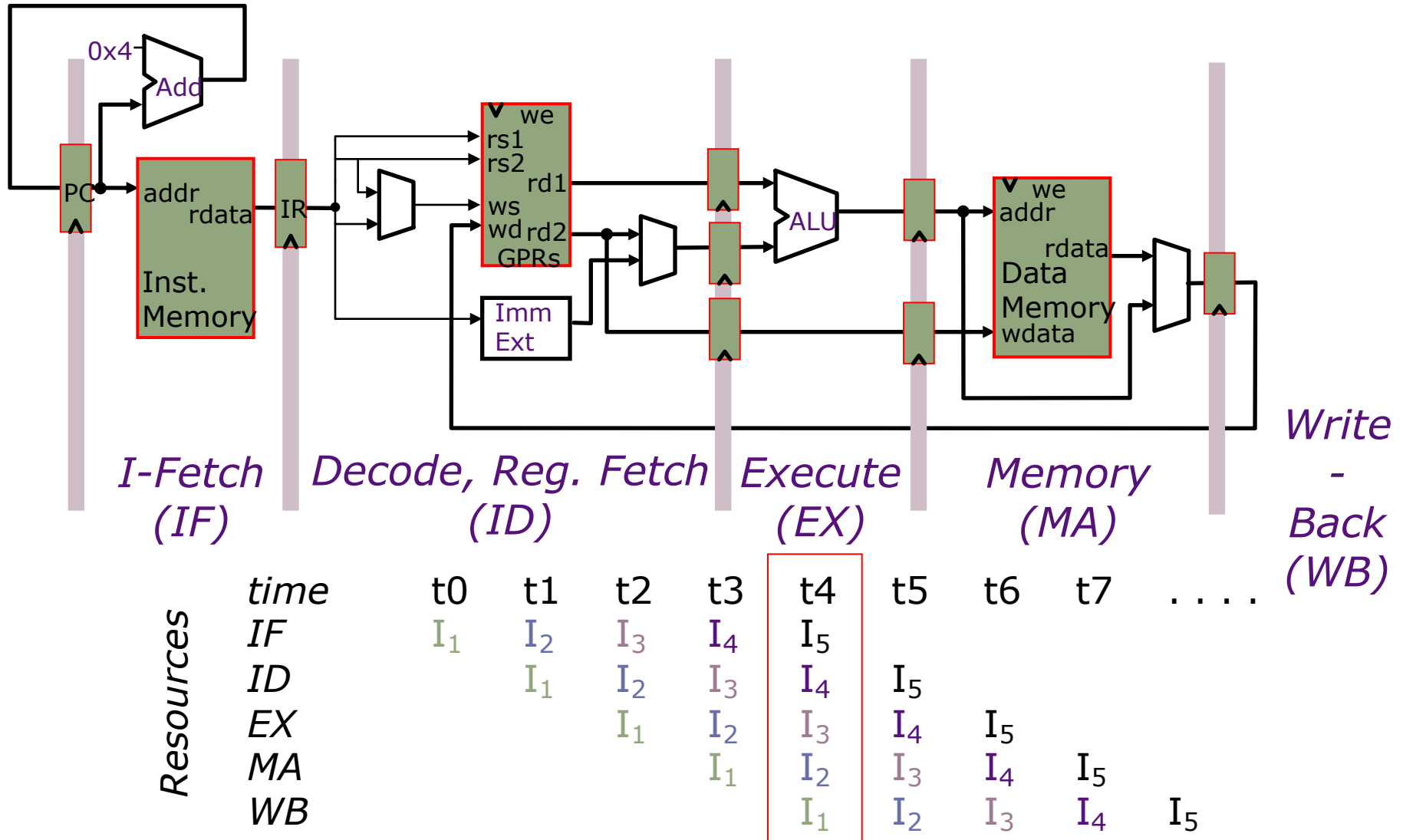
5-Stage Pipelined Execution

Resource Usage Diagram



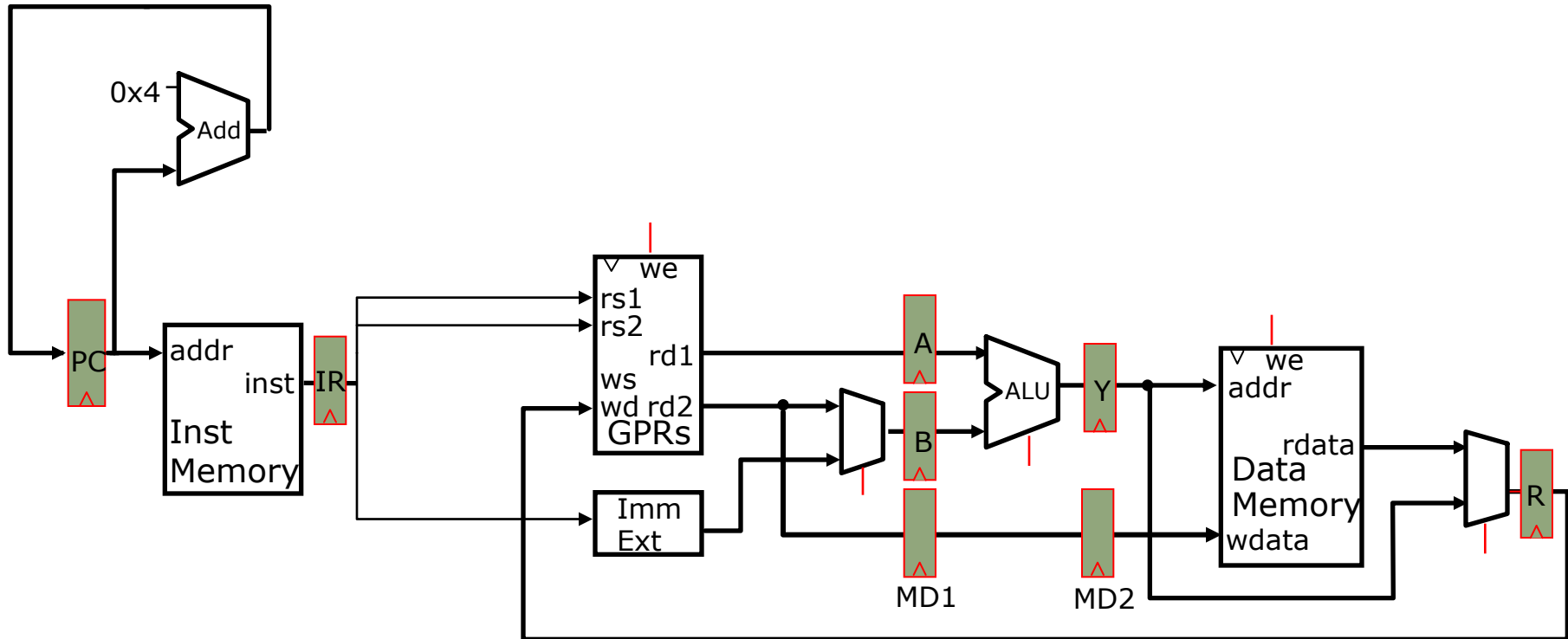
5-Stage Pipelined Execution

Resource Usage Diagram



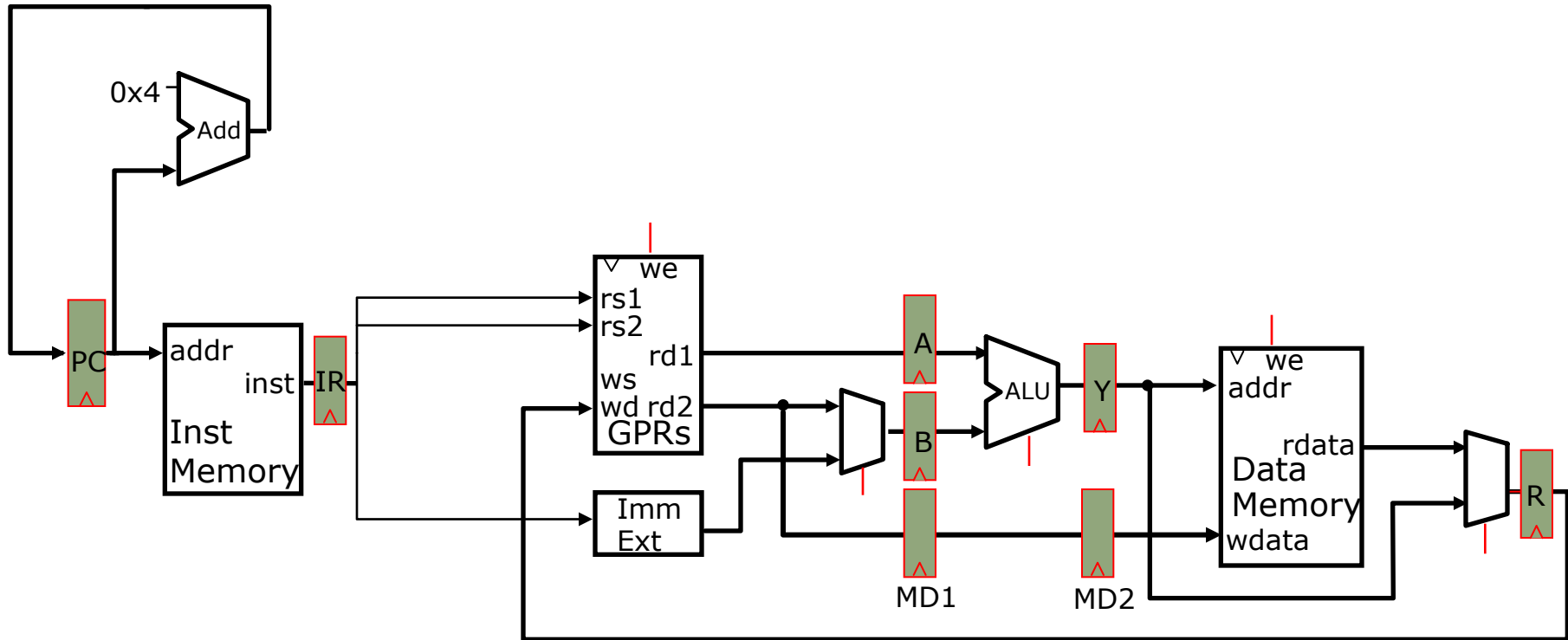
Pipelined Execution

ALU Instructions



Pipelined Execution

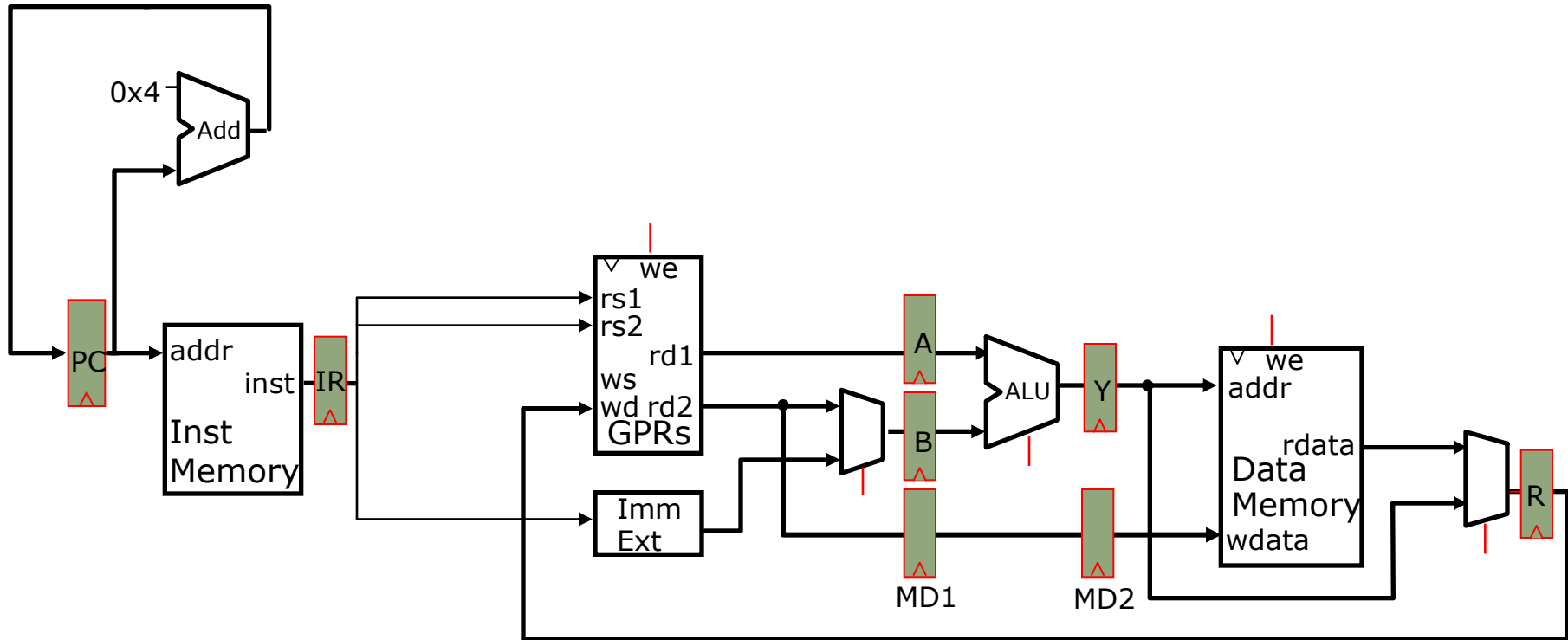
ALU Instructions



Not quite correct!

Pipelined Execution

ALU Instructions

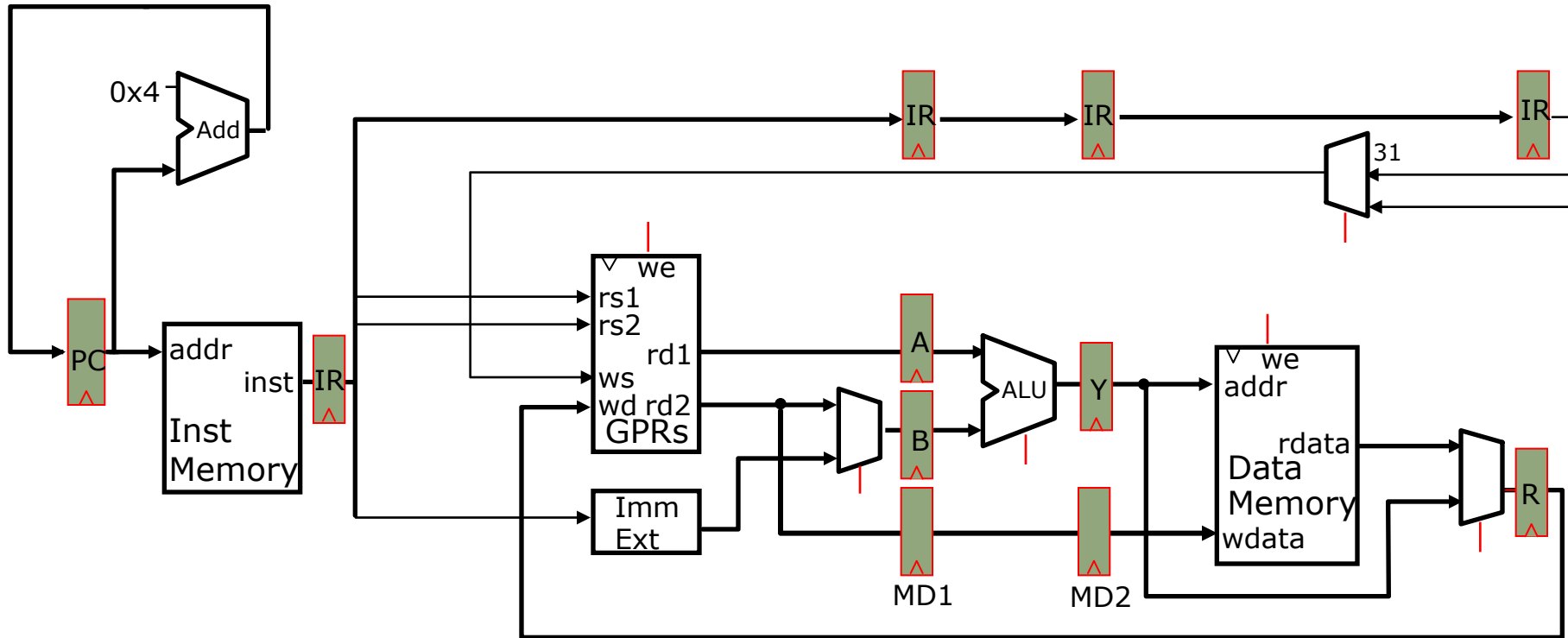


Not quite correct!

We need an Instruction Reg (IR) for each stage

Pipelined Execution

ALU Instructions

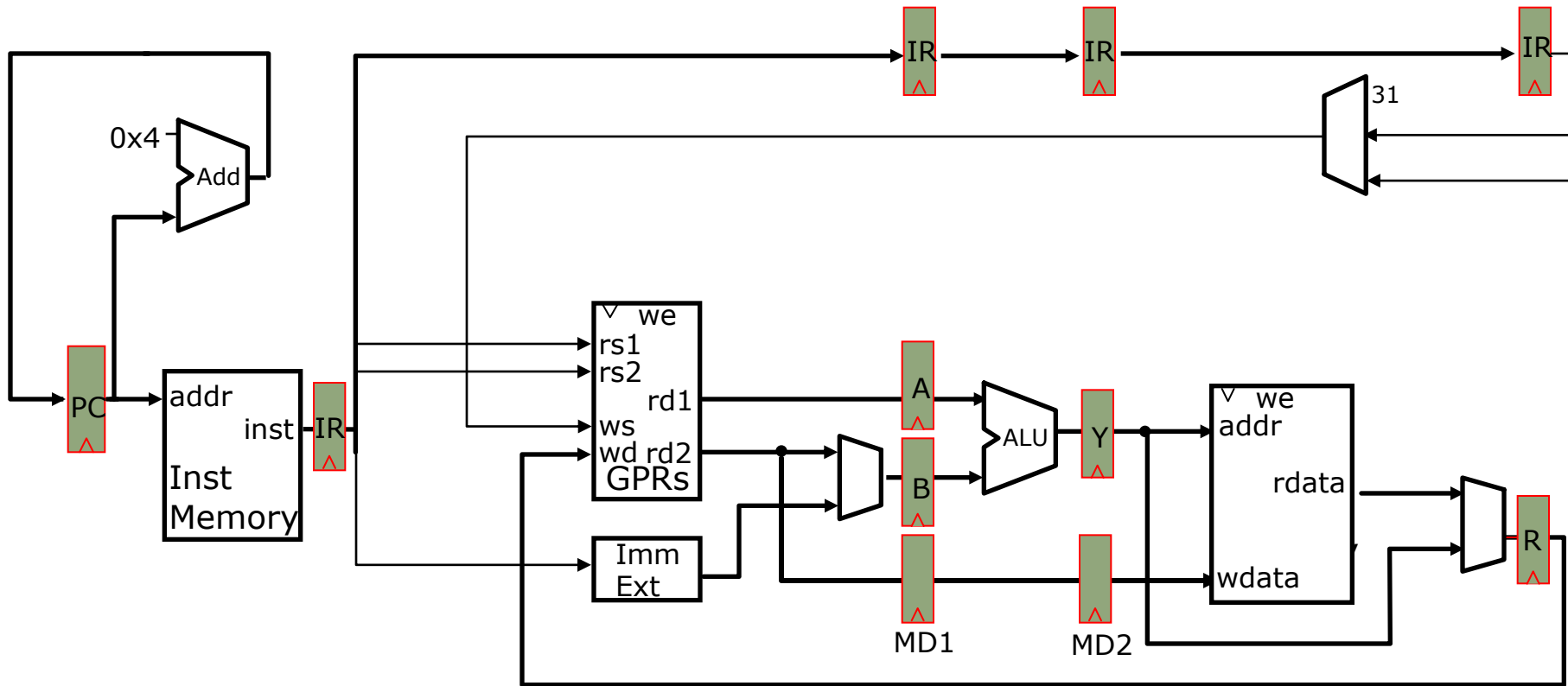


Not quite correct!

We need an Instruction Reg (IR) for each stage

Pipelined MIPS Datapath

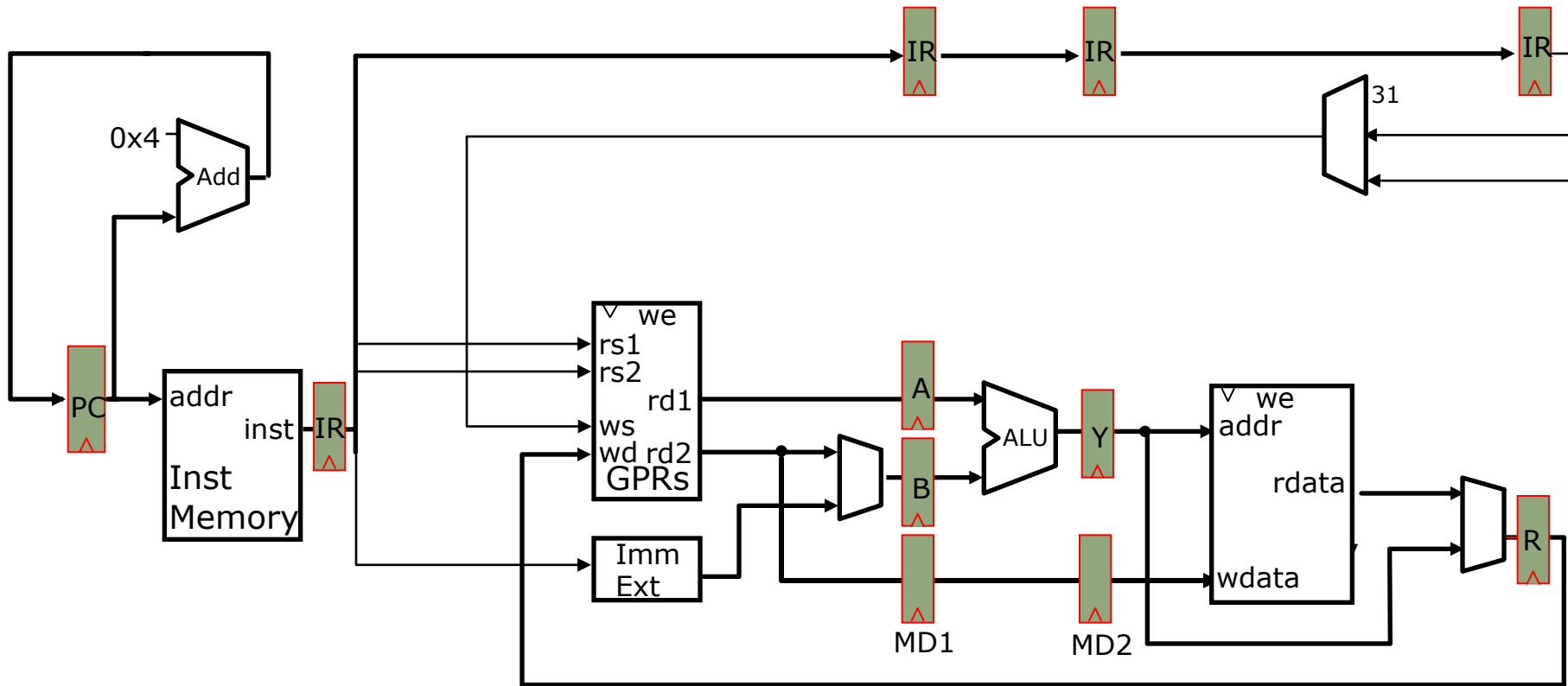
without jumps



What else is needed?

Pipelined MIPS Datapath

without jumps

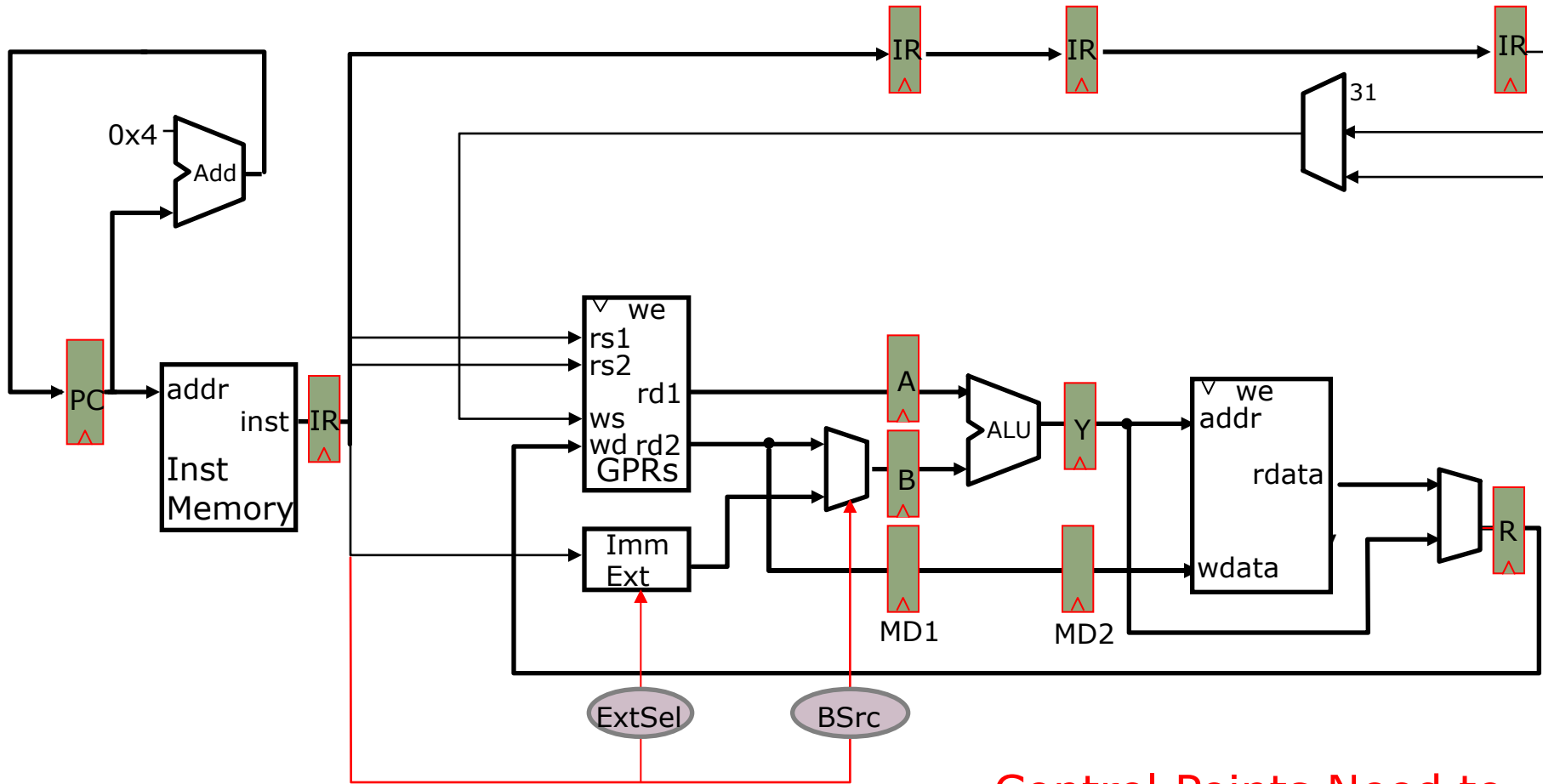


What else is needed?

Control Points Need to Be Connected

Pipelined MIPS Datapath

without jumps

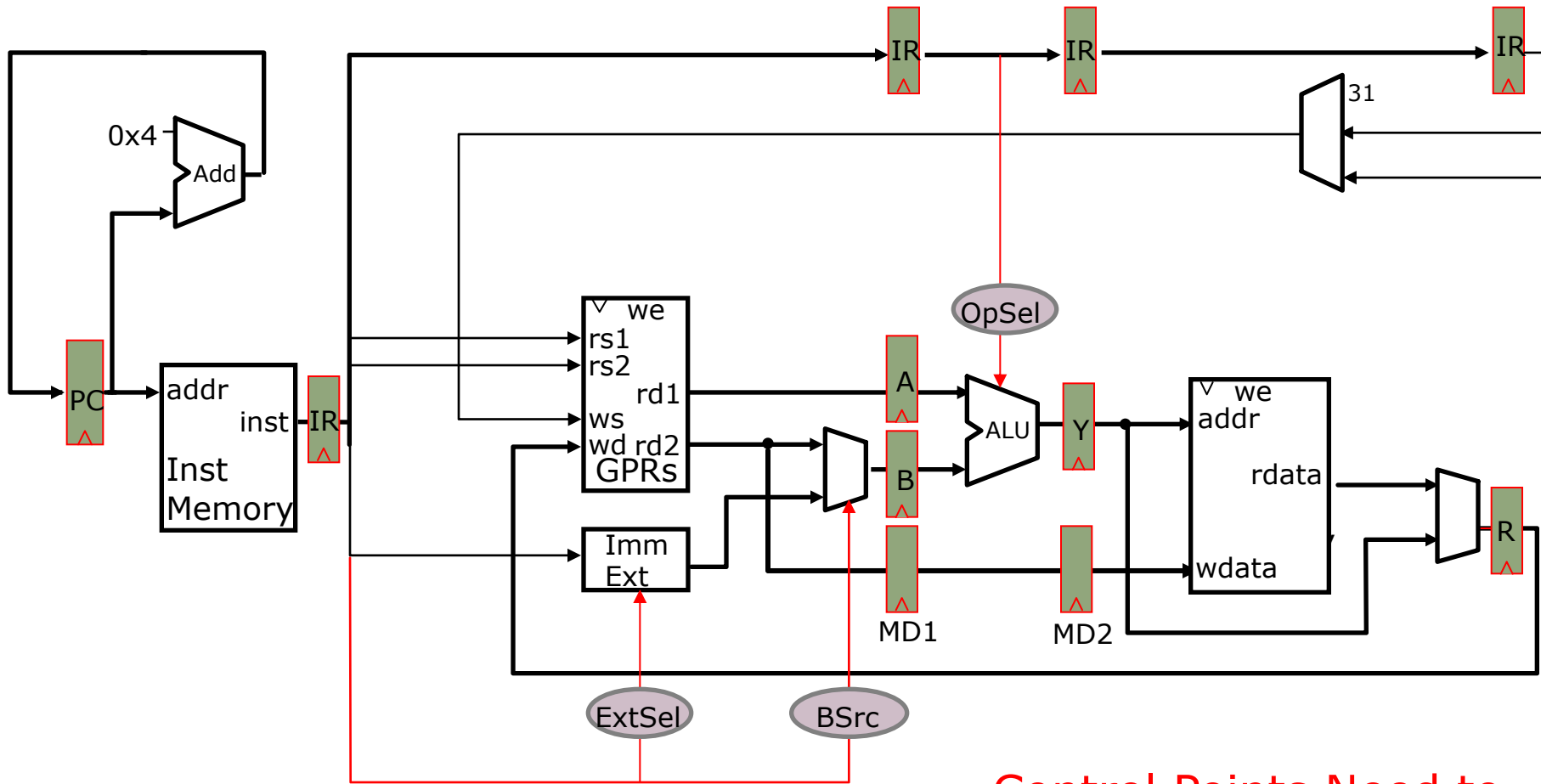


What else is needed?

Control Points Need to Be Connected

Pipelined MIPS Datapath

without jumps

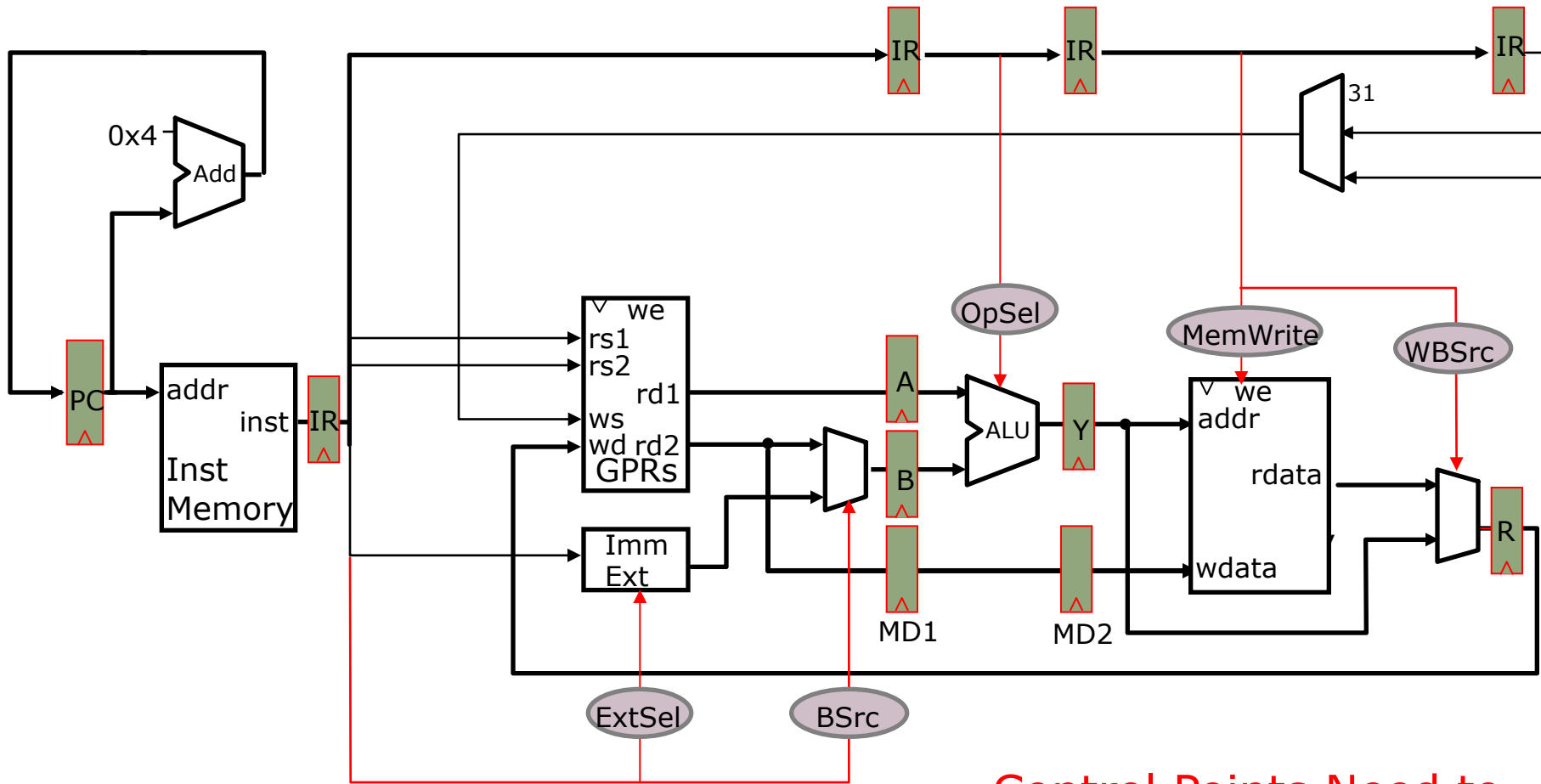


What else is needed?

Control Points Need to Be Connected

Pipelined MIPS Datapath

without jumps

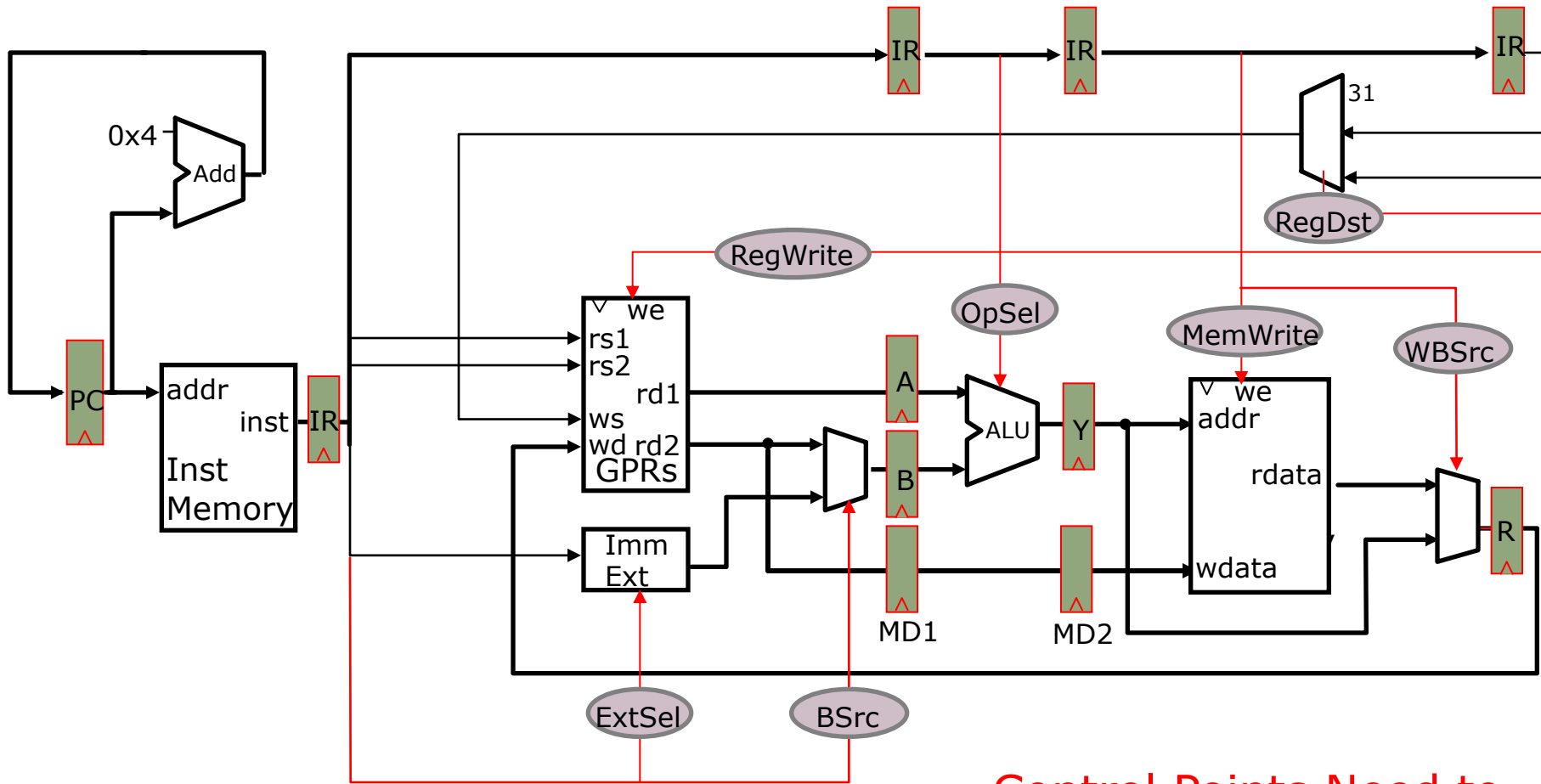


What else is needed?

Control Points Need to Be Connected

Pipelined MIPS Datapath

without jumps

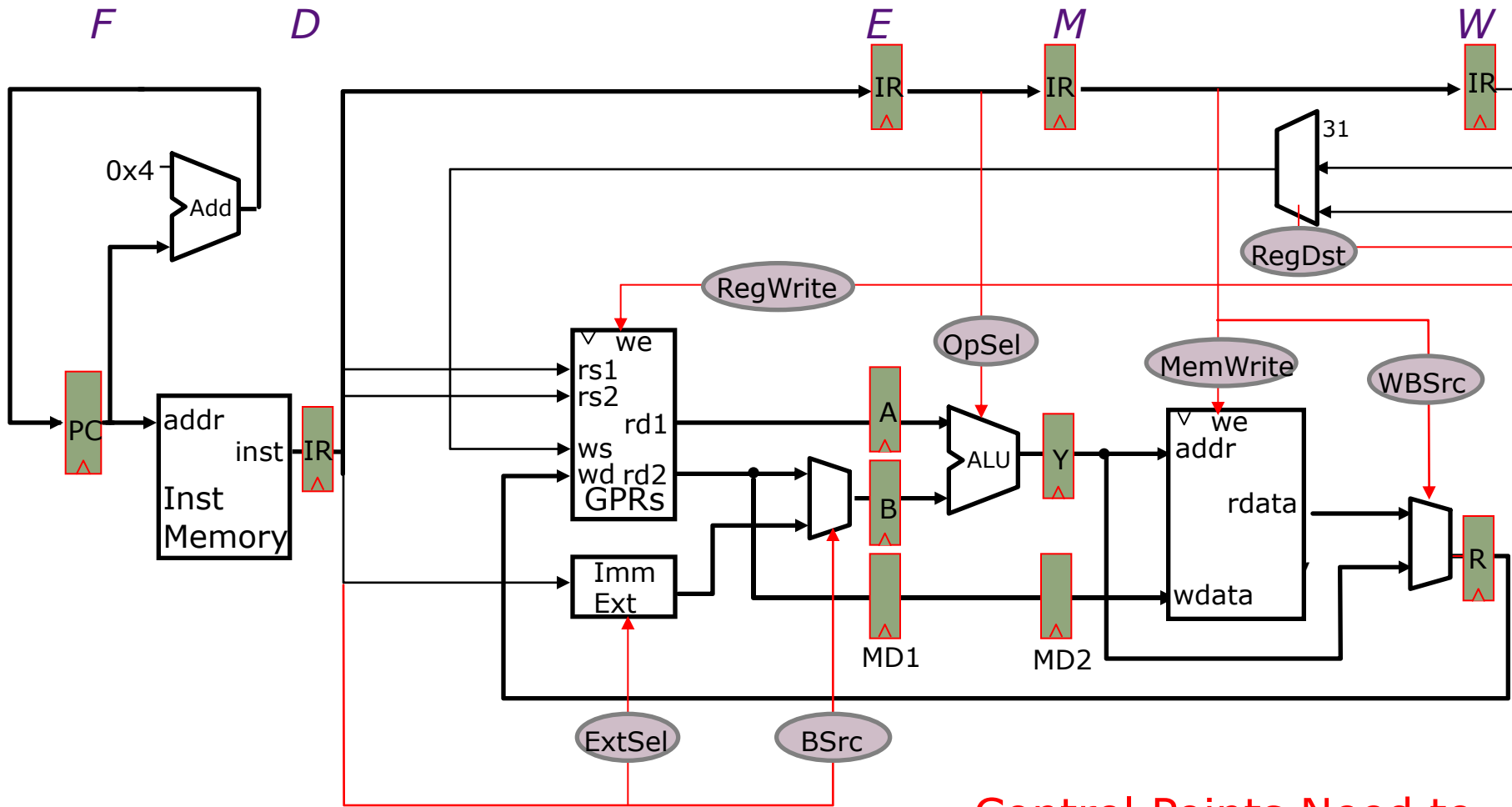


What else is needed?

Control Points Need to Be Connected

Pipelined MIPS Datapath

without jumps



What else is needed?

Control Points Need to Be Connected

How instructions can interact with each other in a pipeline

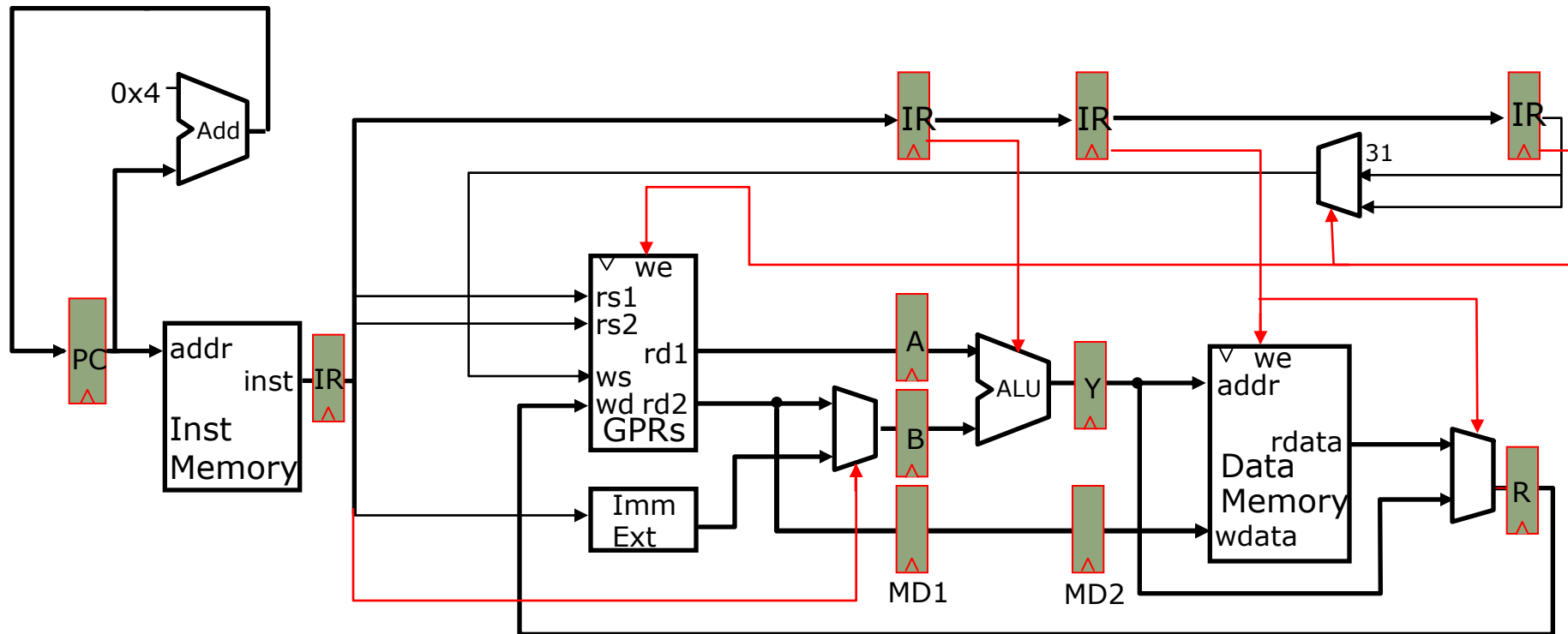
How instructions can interact with each other in a pipeline

- An instruction in the pipeline may need a resource being used by another instruction in the pipeline
→ *structural hazard*

How instructions can interact with each other in a pipeline

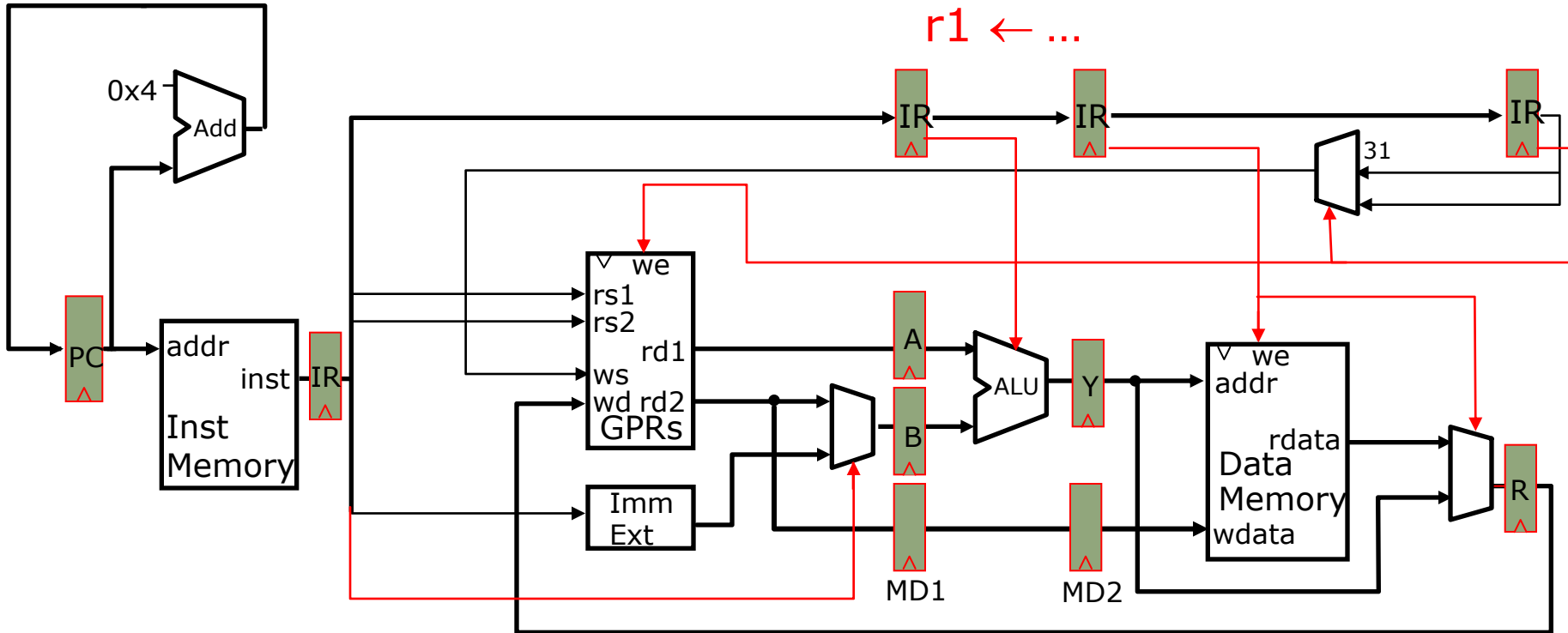
- An instruction in the pipeline may need a resource being used by another instruction in the pipeline
→ *structural hazard*
- An instruction may depend on a value produced by an earlier instruction
 - Dependence may be for a data calculation
→ *data hazard*
 - Dependence may be for calculating the next PC
→ *control hazard (branches, interrupts)*

Data Hazards



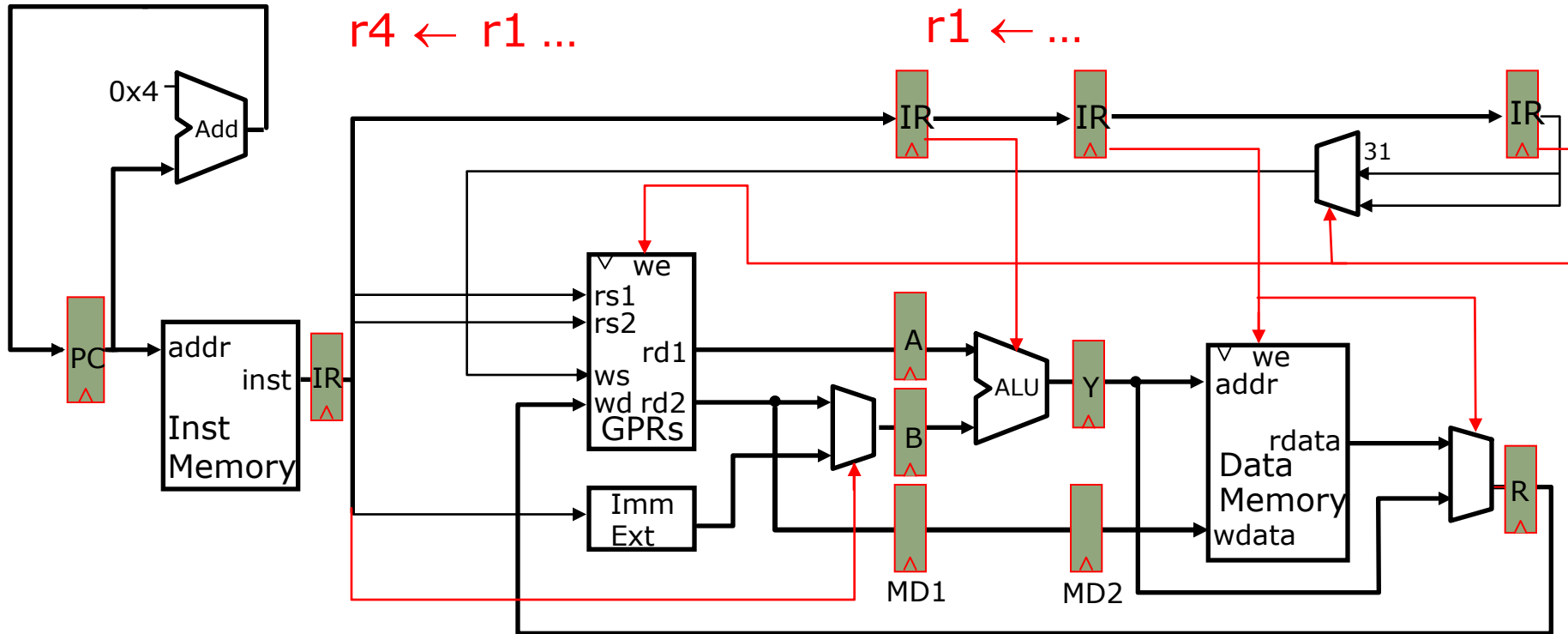
...
 $r1 \leftarrow r0 + 10$
 $r4 \leftarrow r1 + 17$
...

Data Hazards



...
 $r1 \leftarrow r0 + 10$
 $r4 \leftarrow r1 + 17$
 ...

Data Hazards



...
 $r1 \leftarrow r0 + 10$
 $r4 \leftarrow r1 + 17$
 ...

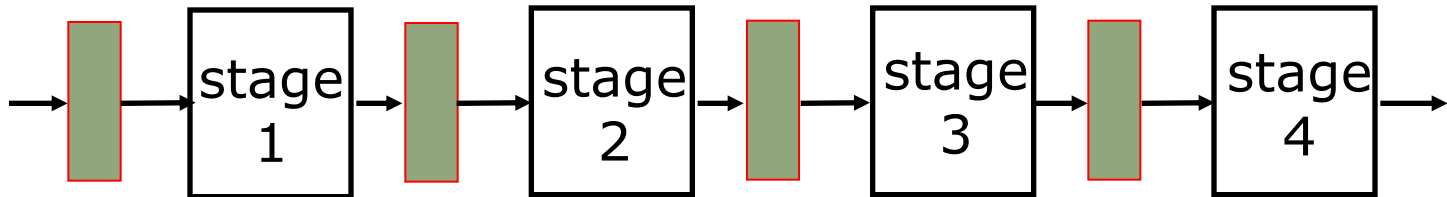
r1 is stale. Oops!

Resolving Data Hazards

Use strategy from Princeton Pipeline:

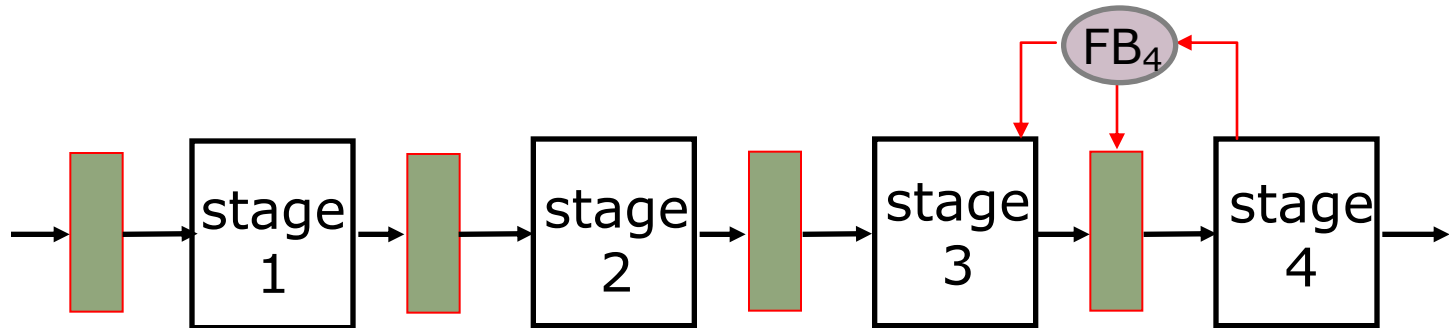
*Wait for the result to be available by freezing earlier pipeline stages → **stall***

Feedback to Resolve Hazards



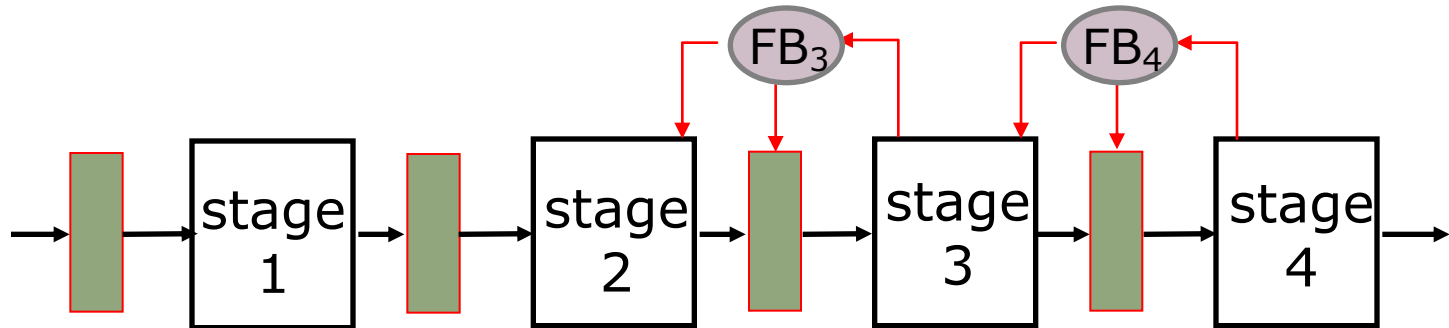
- Later stages provide dependence information to earlier stages which can *stall instructions*

Feedback to Resolve Hazards



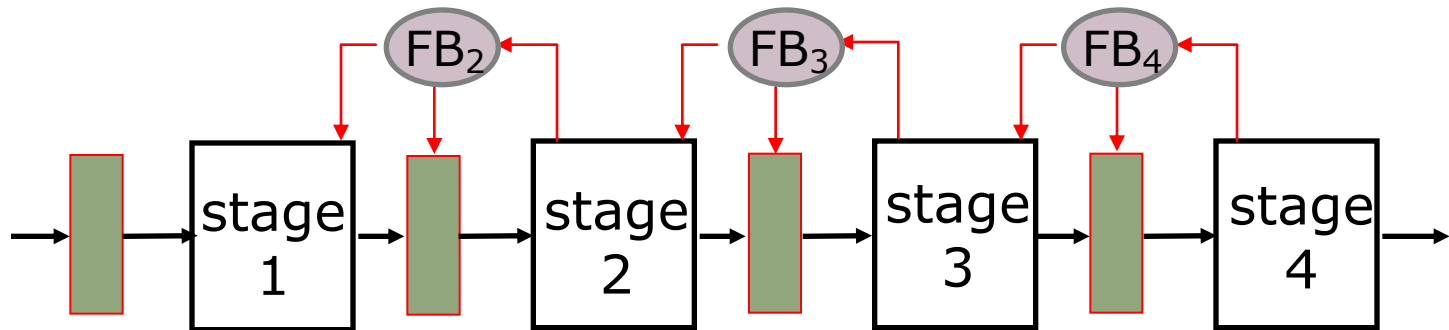
- Later stages provide dependence information to earlier stages which can *stall instructions*

Feedback to Resolve Hazards



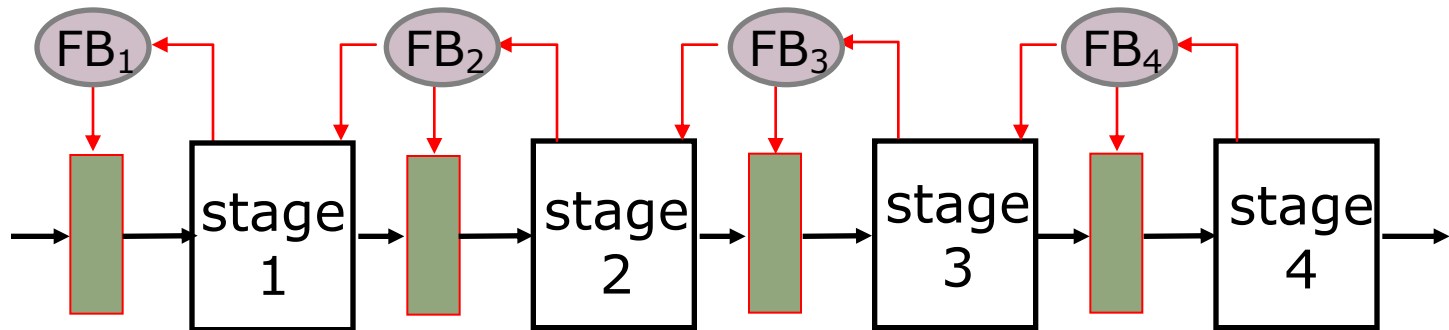
- Later stages provide dependence information to earlier stages which can *stall instructions*

Feedback to Resolve Hazards



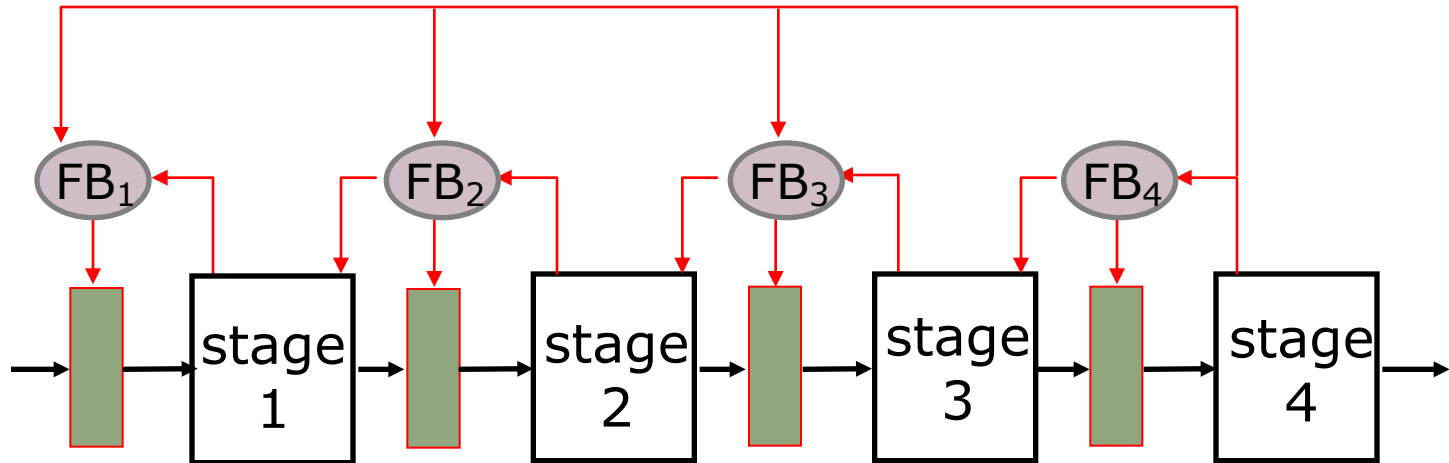
- Later stages provide dependence information to earlier stages which can *stall instructions*

Feedback to Resolve Hazards



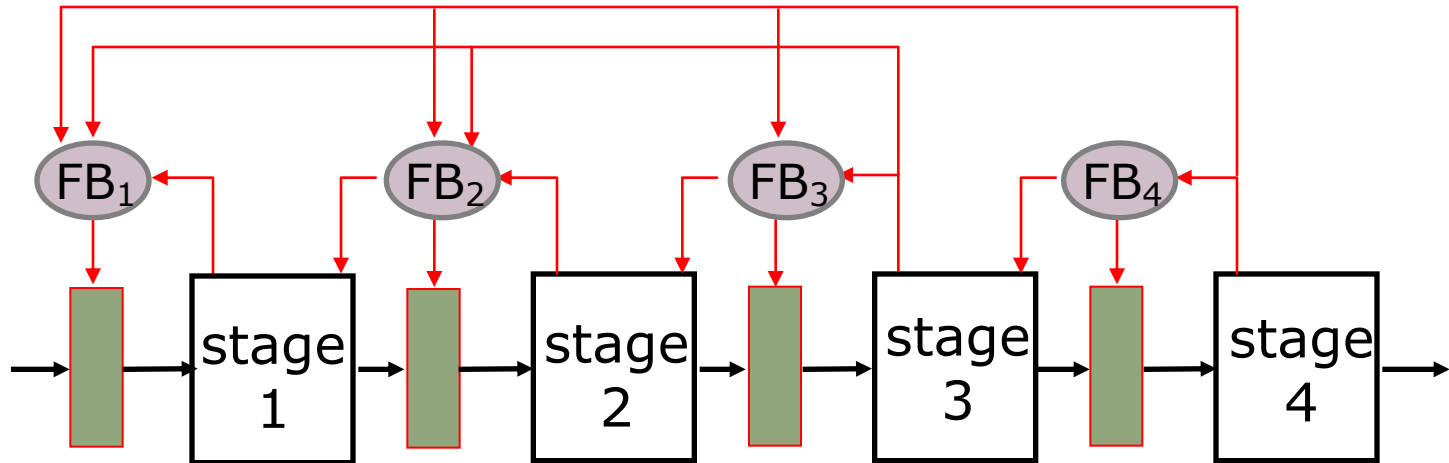
- Later stages provide dependence information to earlier stages which can *stall instructions*

Feedback to Resolve Hazards



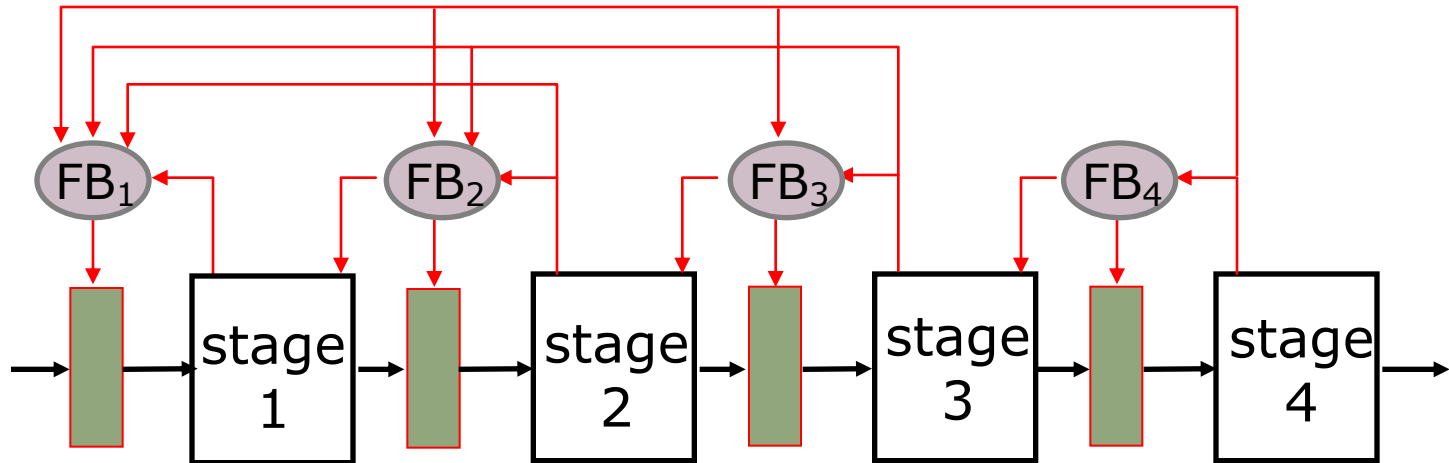
- Later stages provide dependence information to earlier stages which can *stall instructions*

Feedback to Resolve Hazards



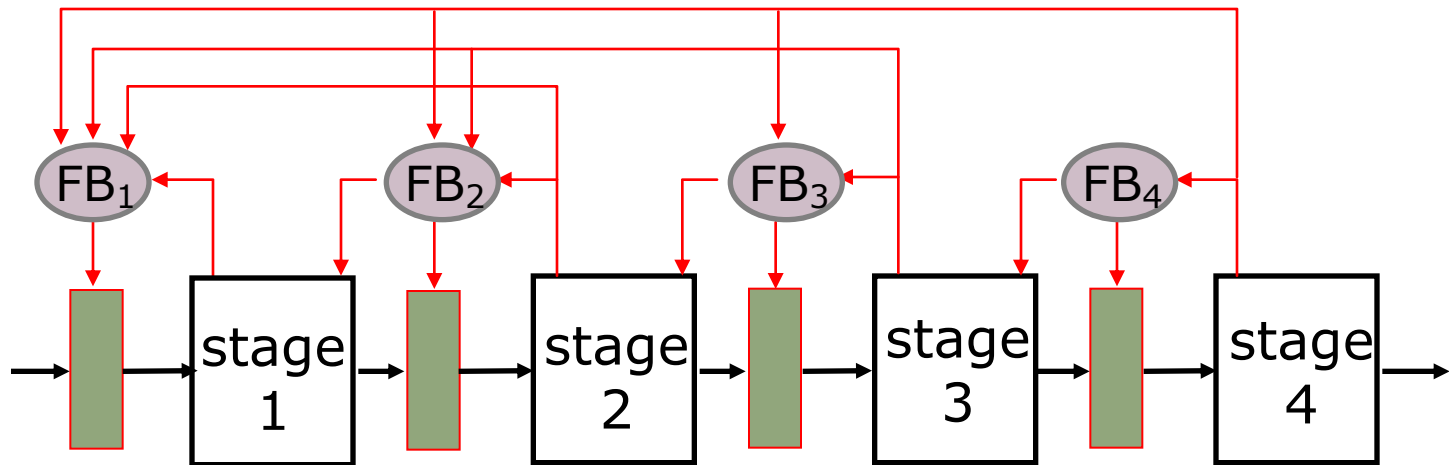
- Later stages provide dependence information to earlier stages which can *stall instructions*

Feedback to Resolve Hazards



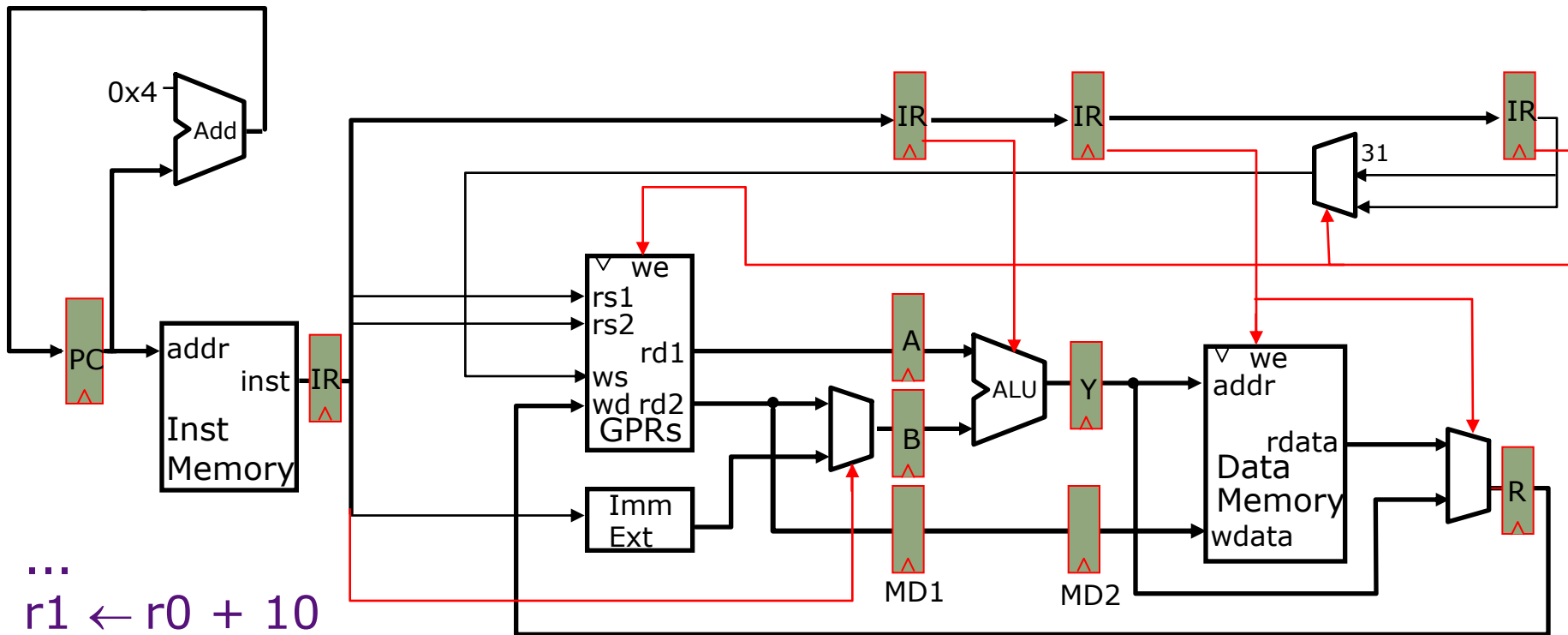
- Later stages provide dependence information to earlier stages which can *stall instructions*

Feedback to Resolve Hazards



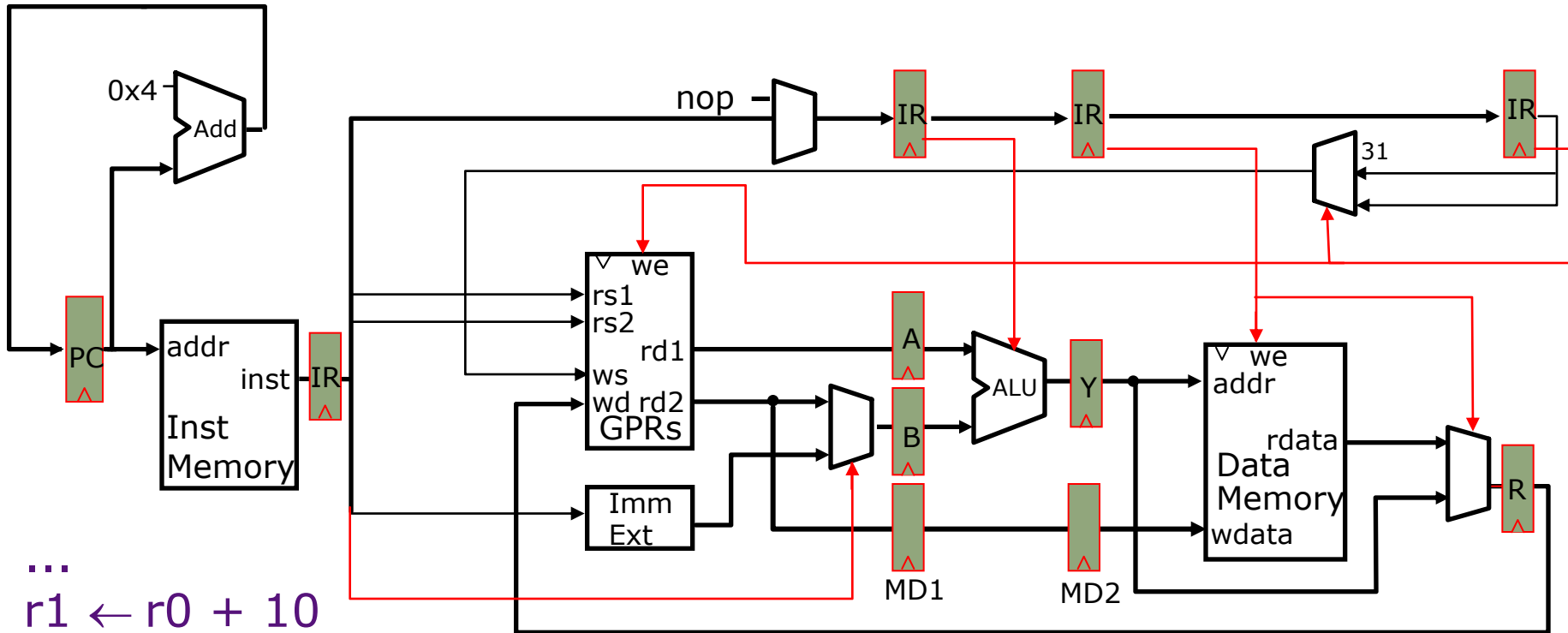
- Later stages provide dependence information to earlier stages which can *stall instructions*
- Controlling a pipeline in this manner works provided *the instruction at stage $i+1$ can complete without any interference from instructions in stages 1 to i* (otherwise deadlocks may occur)

Resolving Data Hazards by Stalling



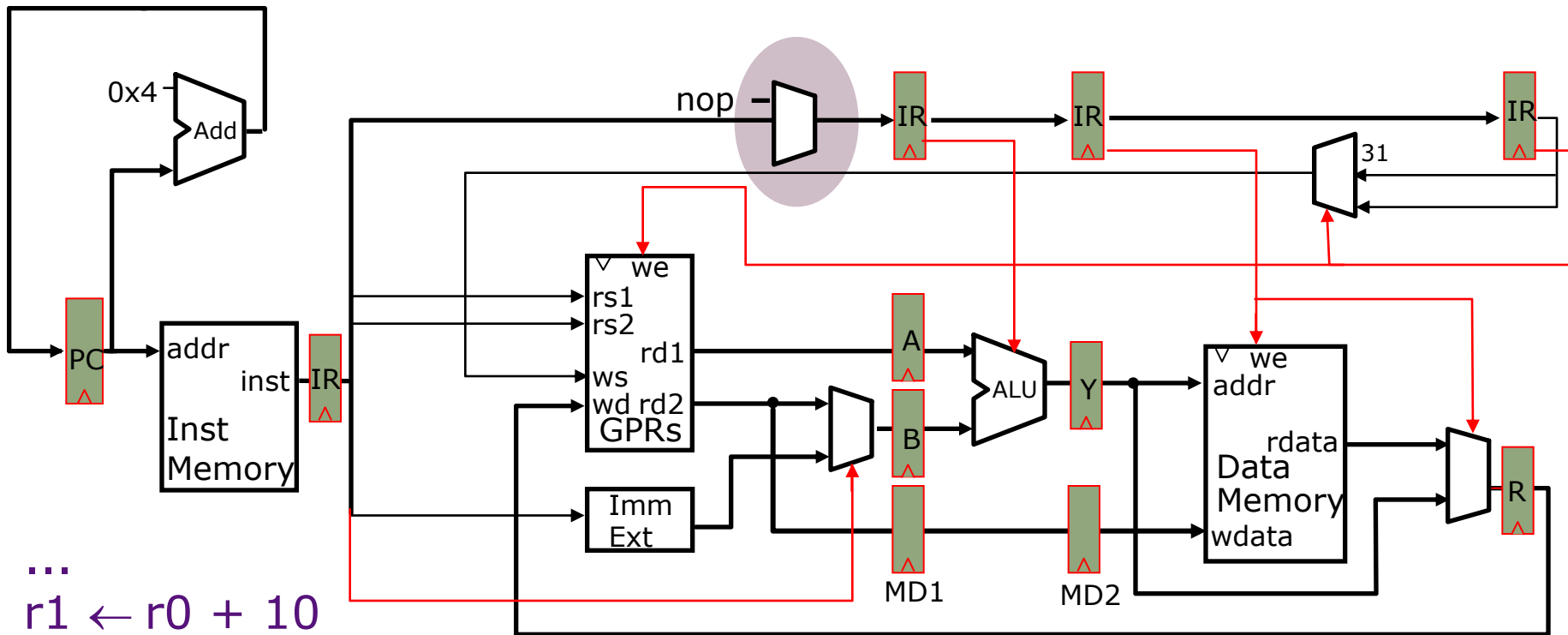
...
 $r1 \leftarrow r0 + 10$
 $r4 \leftarrow r1 + 17$
...

Resolving Data Hazards by Stalling



...
 $r1 \leftarrow r0 + 10$
 $r4 \leftarrow r1 + 17$
...

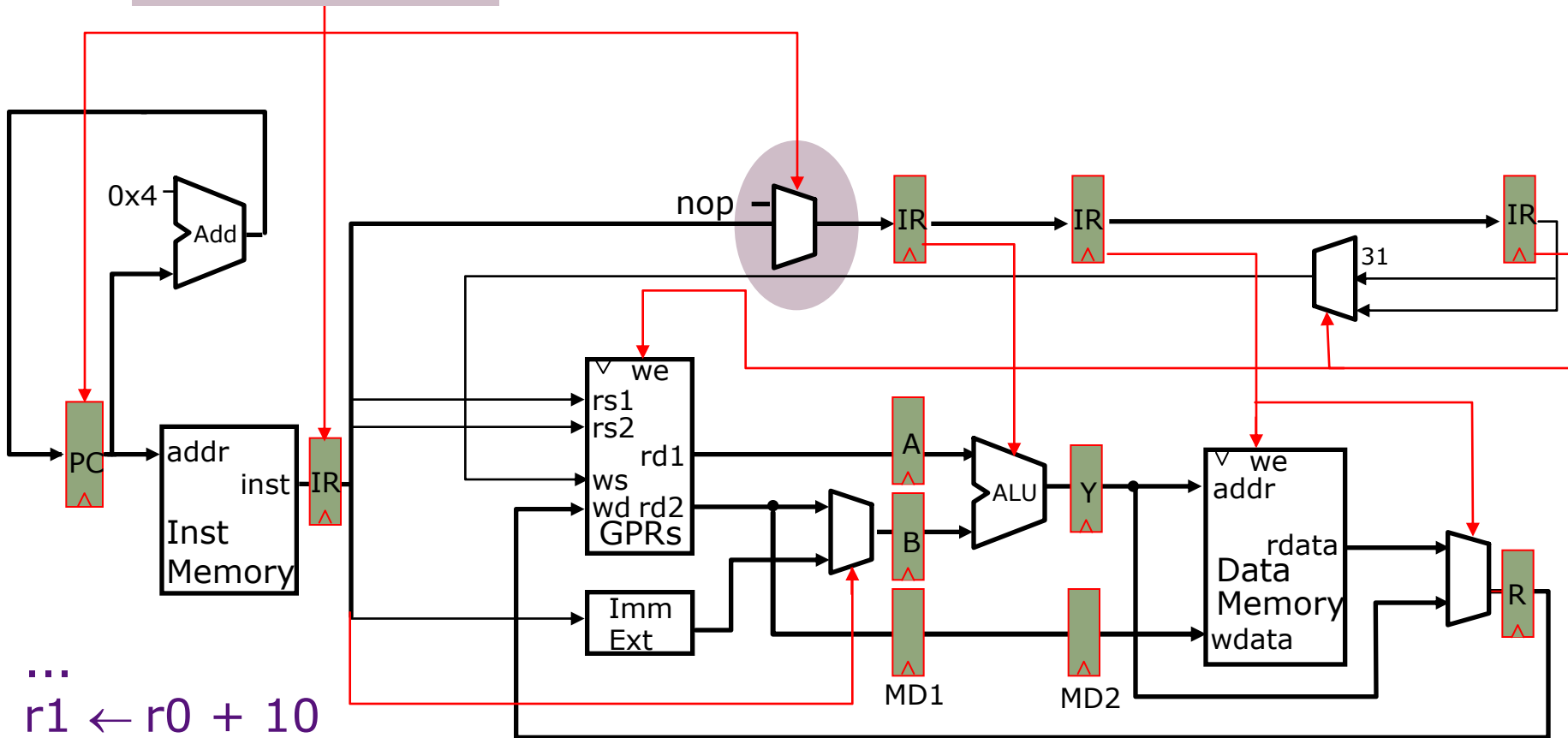
Resolving Data Hazards by Stalling



...
 $r1 \leftarrow r0 + 10$
 $r4 \leftarrow r1 + 17$
...

Resolving Data Hazards by Stalling

Stall Condition



...
 $r1 \leftarrow r0 + 10$
 $r4 \leftarrow r1 + 17$
...

Stalled Stages and Pipeline Bubbles

Stalled Stages and Pipeline Bubbles

time

t0 t1 t2 t3 t4 t5 t6 t7

Stalled Stages and Pipeline Bubbles

time

t0 t1 t2 t3 t4 t5 t6 t7

(I₁) r1 ← (r0) + 10 IF₁ ID₁ EX₁ MA₁ WB₁

Stalled Stages and Pipeline Bubbles

	<i>time</i>								
	t0	t1	t2	t3	t4	t5	t6	t7	...
(I ₁) $r1 \leftarrow (r0) + 10$	IF ₁	ID ₁	EX ₁	MA ₁	WB ₁				
(I ₂) $r4 \leftarrow (r1) + 17$		IF ₂	ID ₂	ID ₂	ID ₂	ID ₂	EX ₂	MA ₂	WB ₂

Stalled Stages and Pipeline Bubbles

	<i>time</i>									
	t0	t1	t2	t3	t4	t5	t6	t7
(I ₁) $r1 \leftarrow (r0) + 10$	IF ₁	ID ₁	EX ₁	MA ₁	WB ₁					
(I ₂) $r4 \leftarrow (r1) + 17$		IF ₂	ID ₂	ID ₂	ID ₂	ID ₂	EX ₂	MA ₂	WB ₂	
(I ₃)			IF ₃	IF ₃	IF ₃	IF ₃	ID ₃	EX ₃	MA ₃	WB ₃

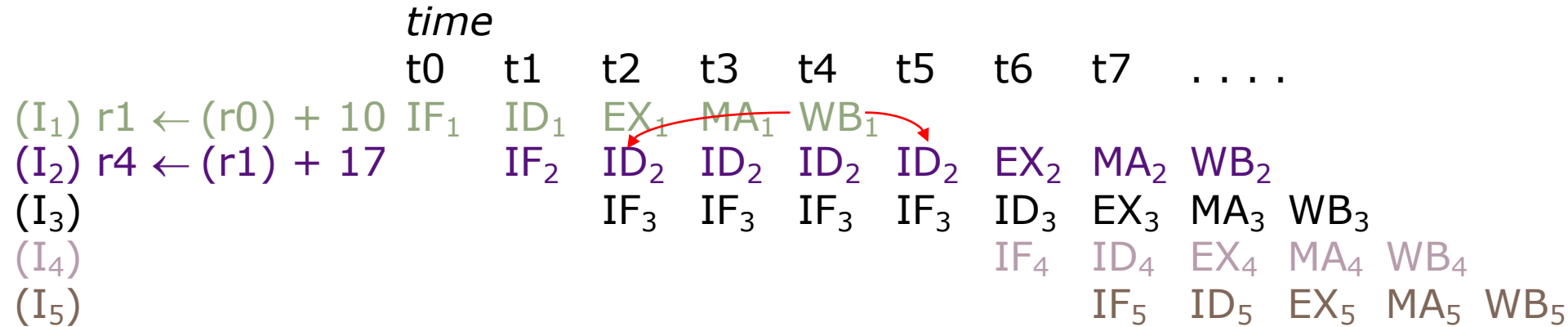
Stalled Stages and Pipeline Bubbles

	<i>time</i>										
	t0	t1	t2	t3	t4	t5	t6	t7	...		
(I ₁) $r1 \leftarrow (r0) + 10$	IF ₁	ID ₁	EX ₁	MA ₁	WB ₁						
(I ₂) $r4 \leftarrow (r1) + 17$		IF ₂	ID ₂	ID ₂	ID ₂	ID ₂	EX ₂	MA ₂	WB ₂		
(I ₃)			IF ₃	IF ₃	IF ₃	IF ₃	ID ₃	EX ₃	MA ₃	WB ₃	
(I ₄)							IF ₄	ID ₄	EX ₄	MA ₄	WB ₄

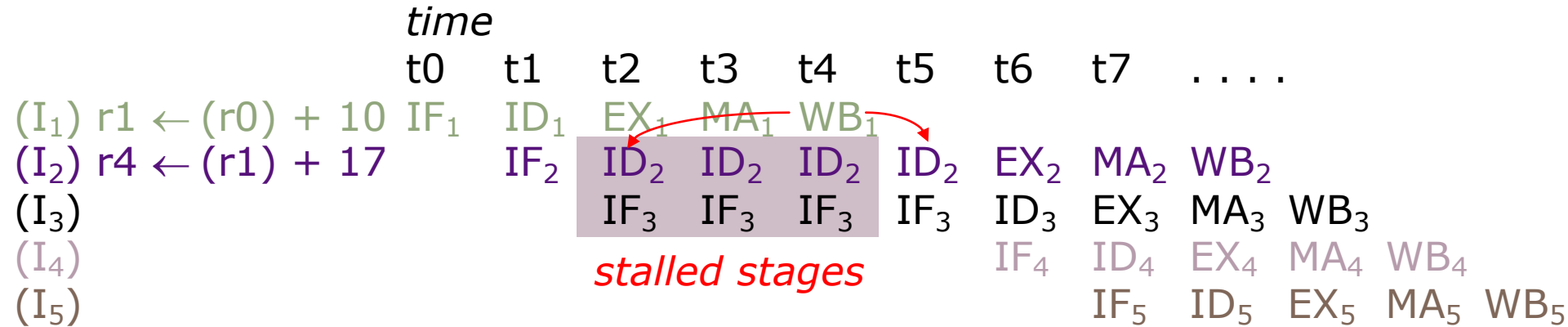
Stalled Stages and Pipeline Bubbles

	<i>time</i>											
	t0	t1	t2	t3	t4	t5	t6	t7			
(I ₁) $r1 \leftarrow (r0) + 10$	IF ₁	ID ₁	EX ₁	MA ₁	WB ₁							
(I ₂) $r4 \leftarrow (r1) + 17$		IF ₂	ID ₂	ID ₂	ID ₂	ID ₂	EX ₂	MA ₂	WB ₂			
(I ₃)			IF ₃	IF ₃	IF ₃	IF ₃	ID ₃	EX ₃	MA ₃	WB ₃		
(I ₄)							IF ₄	ID ₄	EX ₄	MA ₄	WB ₄	
(I ₅)								IF ₅	ID ₅	EX ₅	MA ₅	WB ₅

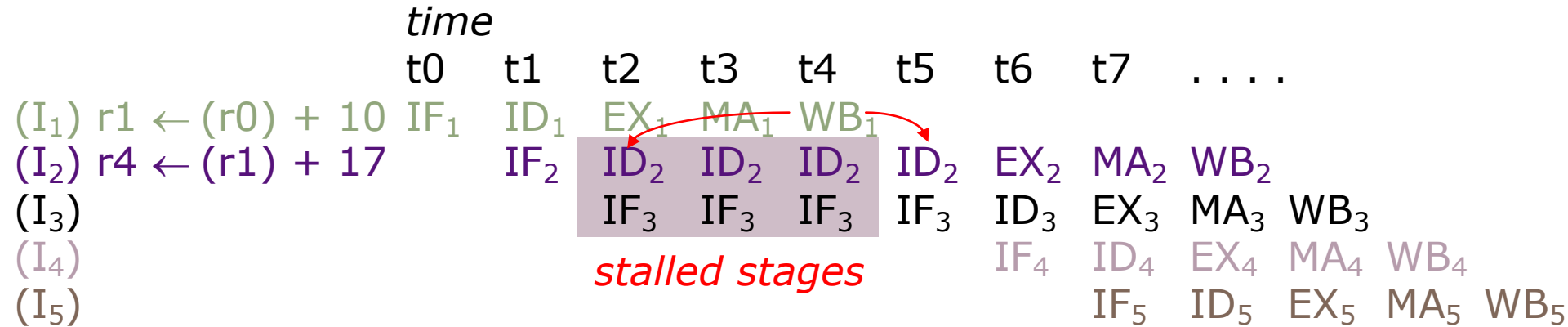
Stalled Stages and Pipeline Bubbles



Stalled Stages and Pipeline Bubbles

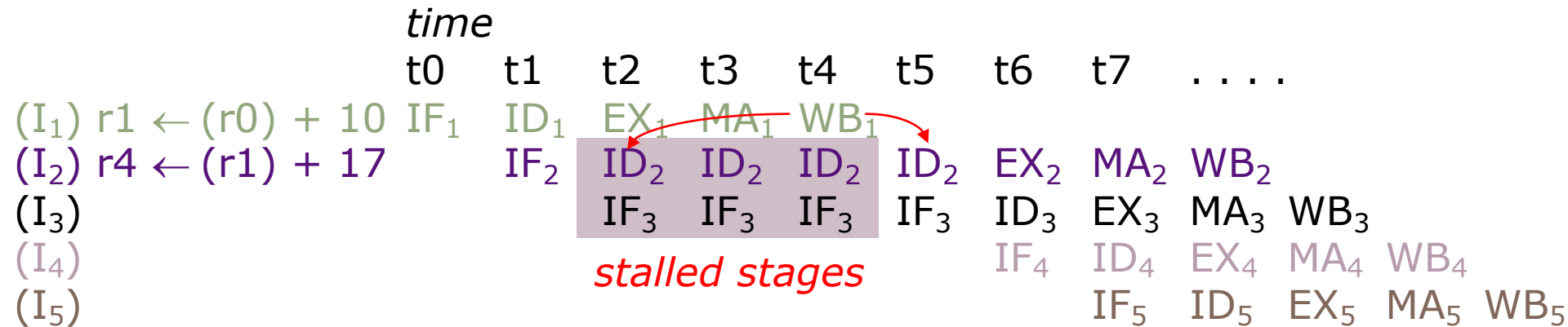


Stalled Stages and Pipeline Bubbles



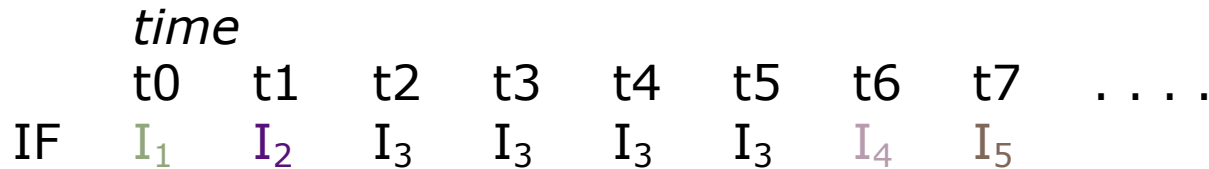
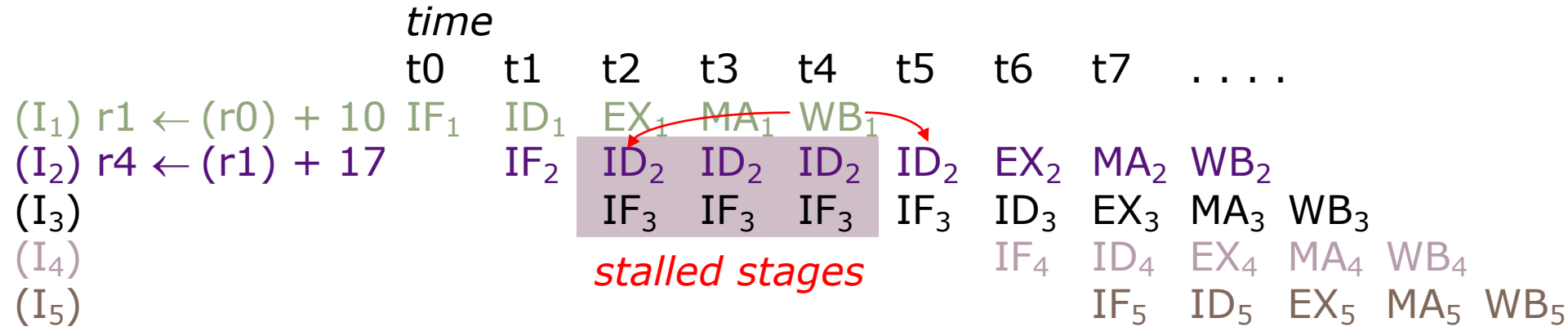
*Resource
Usage*

Stalled Stages and Pipeline Bubbles



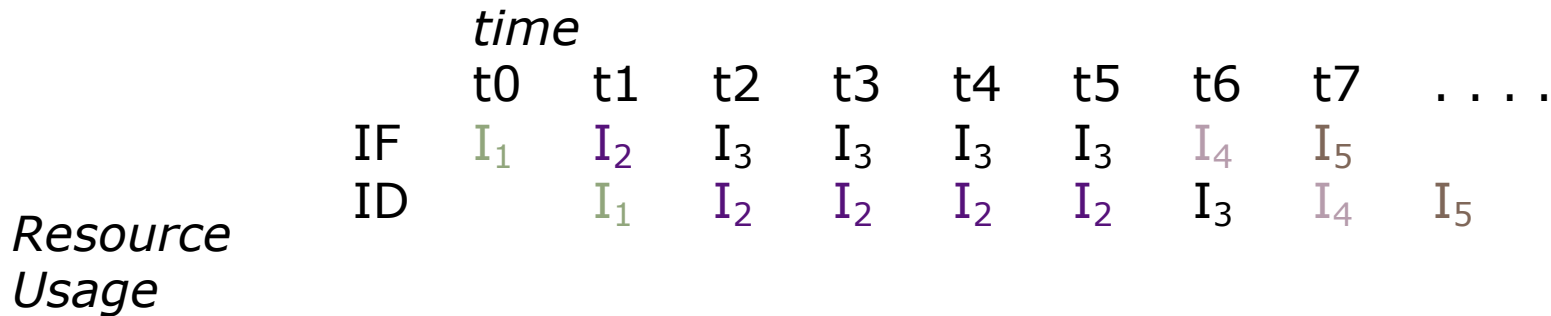
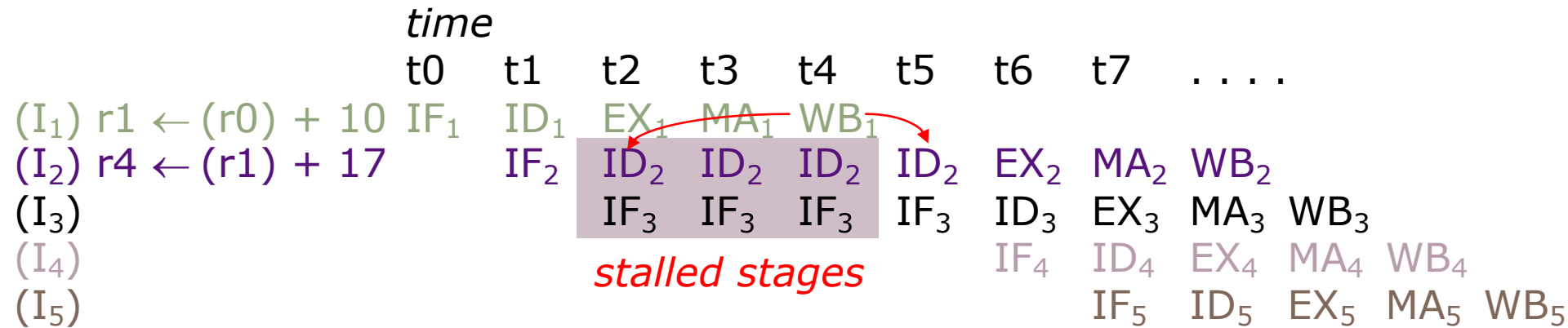
Resource
Usage

Stalled Stages and Pipeline Bubbles

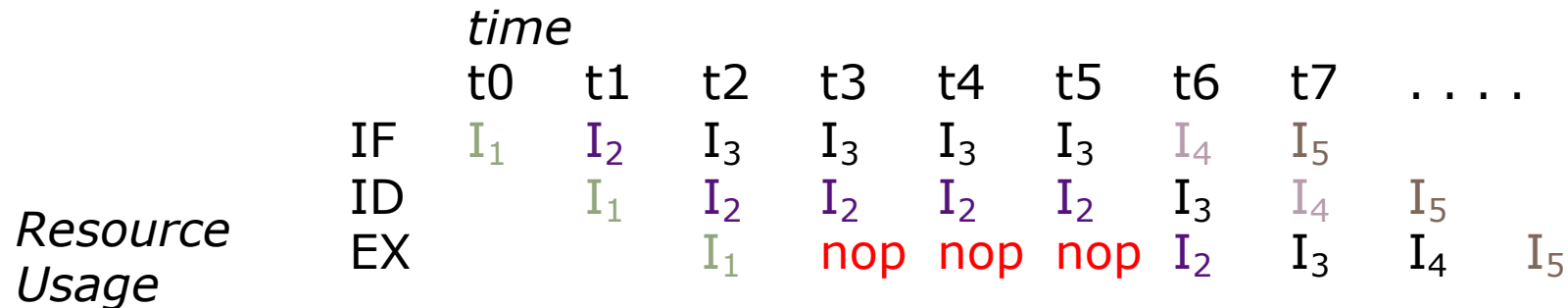
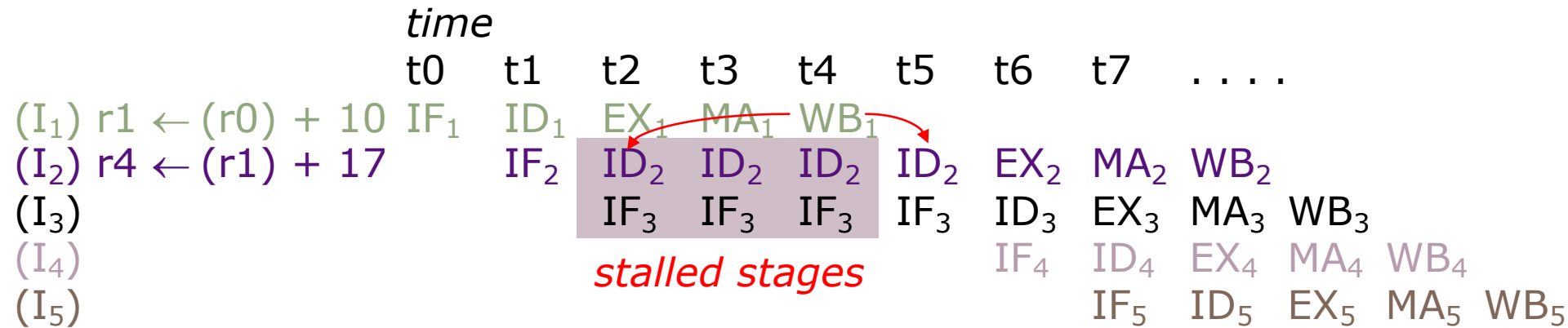


Resource Usage

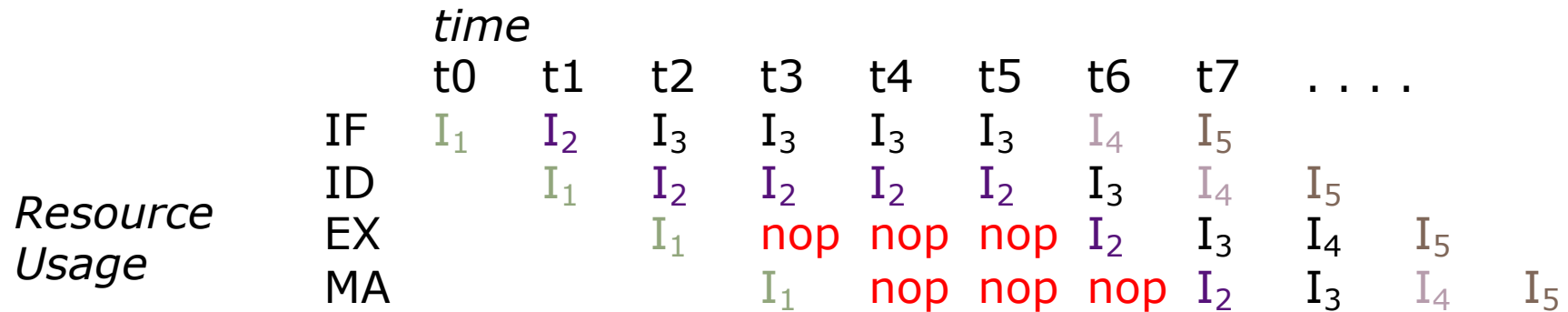
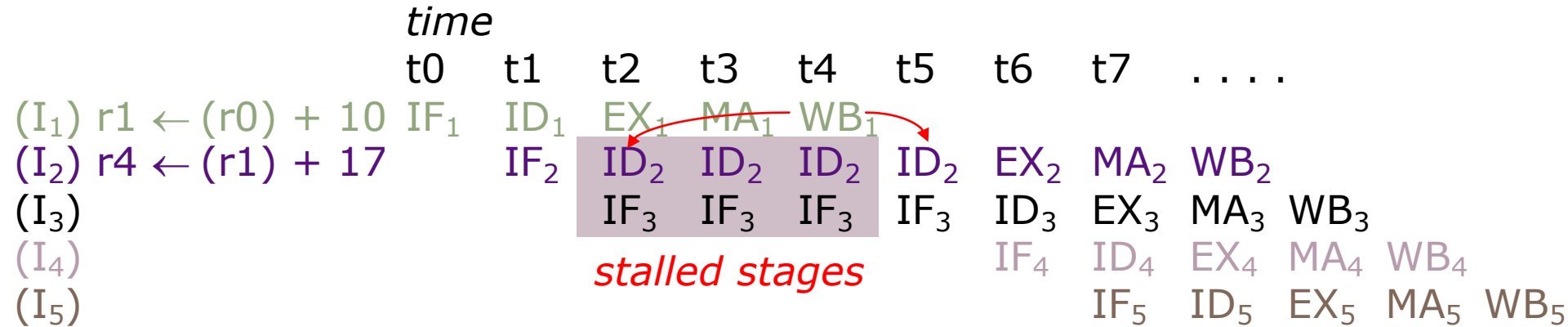
Stalled Stages and Pipeline Bubbles



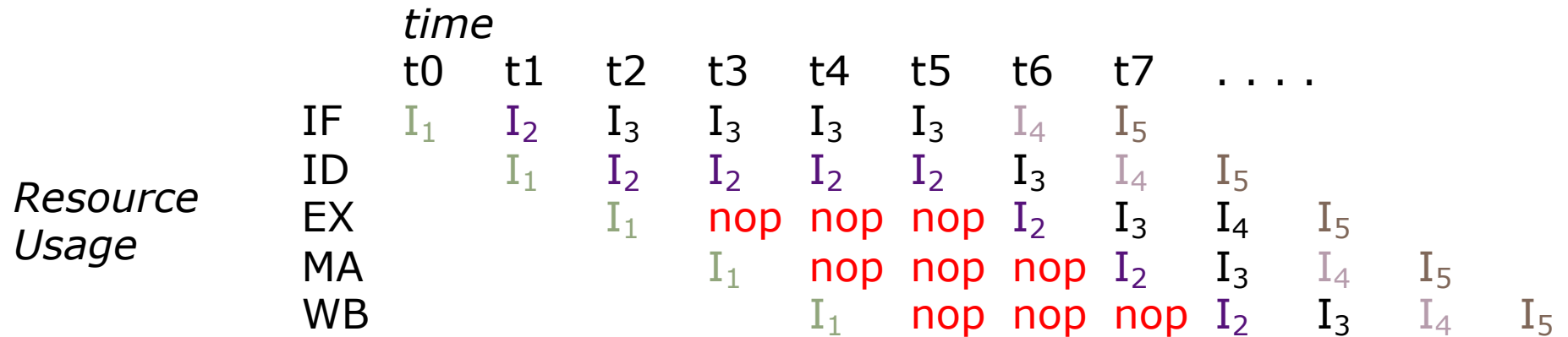
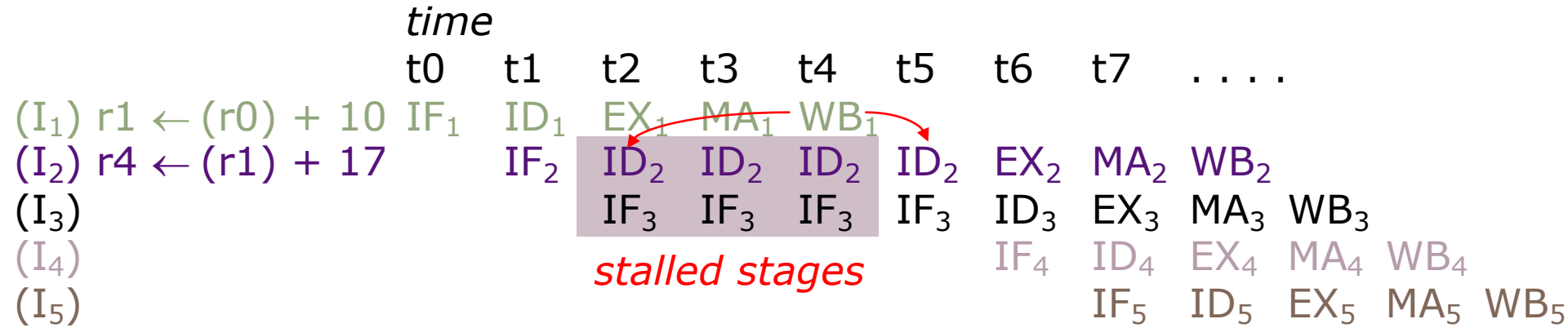
Stalled Stages and Pipeline Bubbles



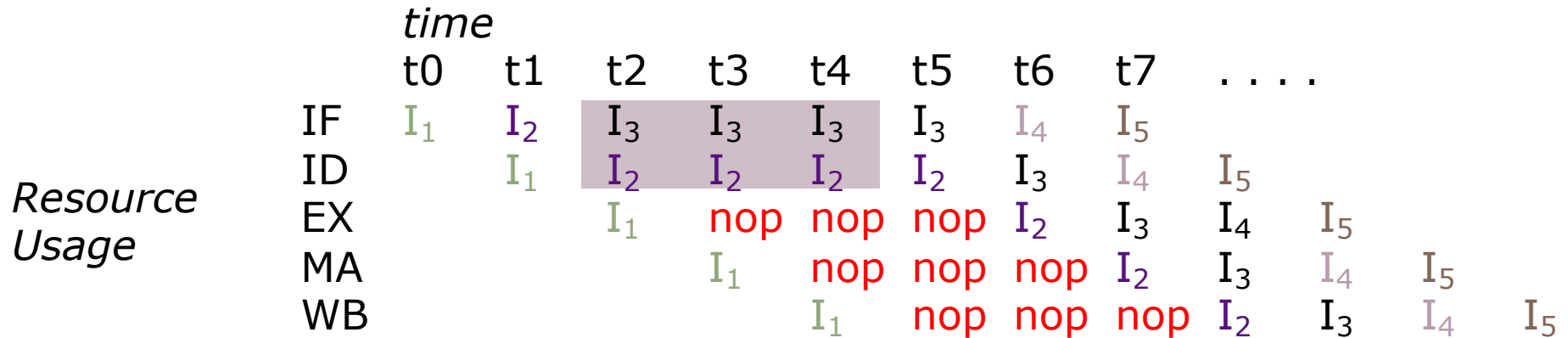
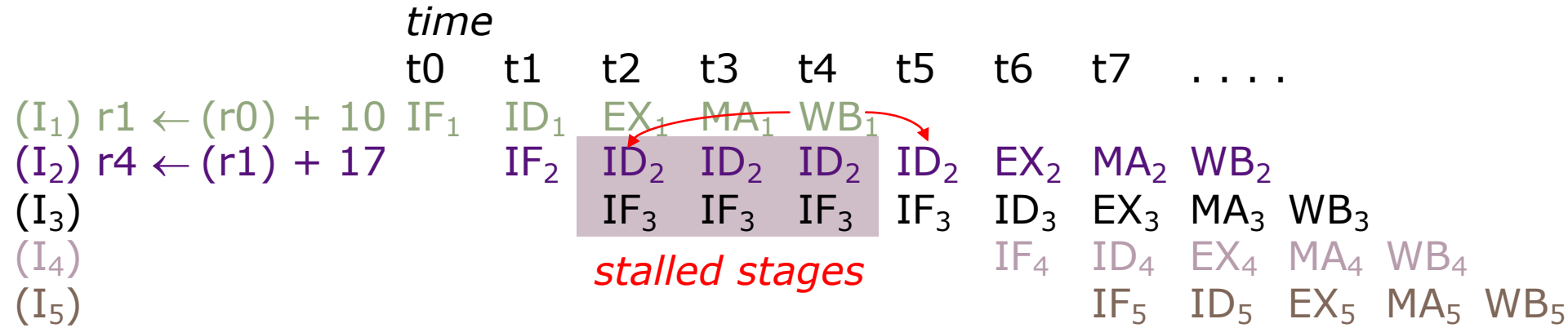
Stalled Stages and Pipeline Bubbles



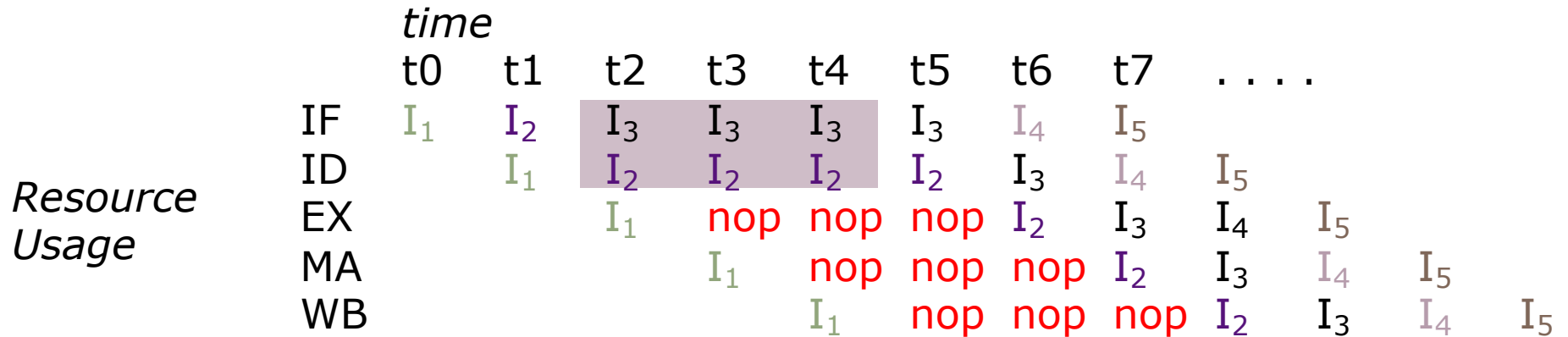
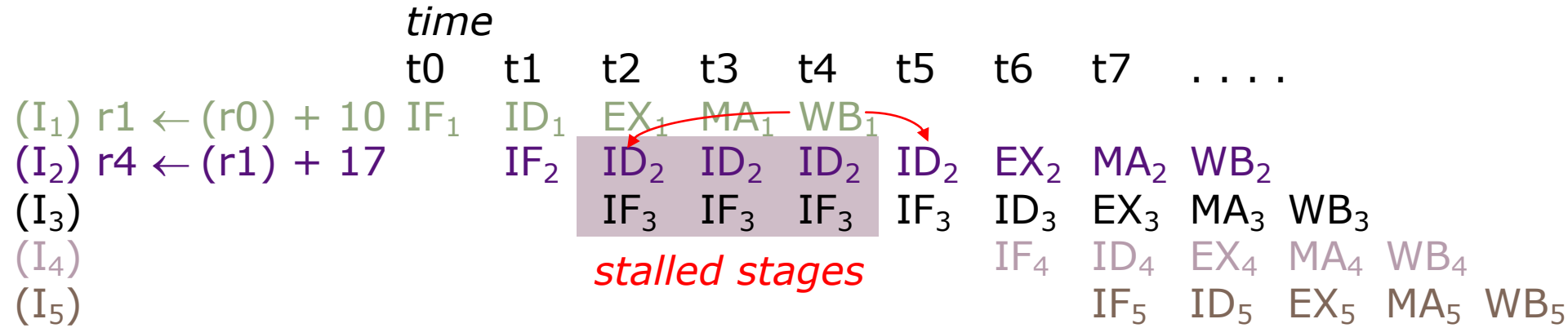
Stalled Stages and Pipeline Bubbles



Stalled Stages and Pipeline Bubbles

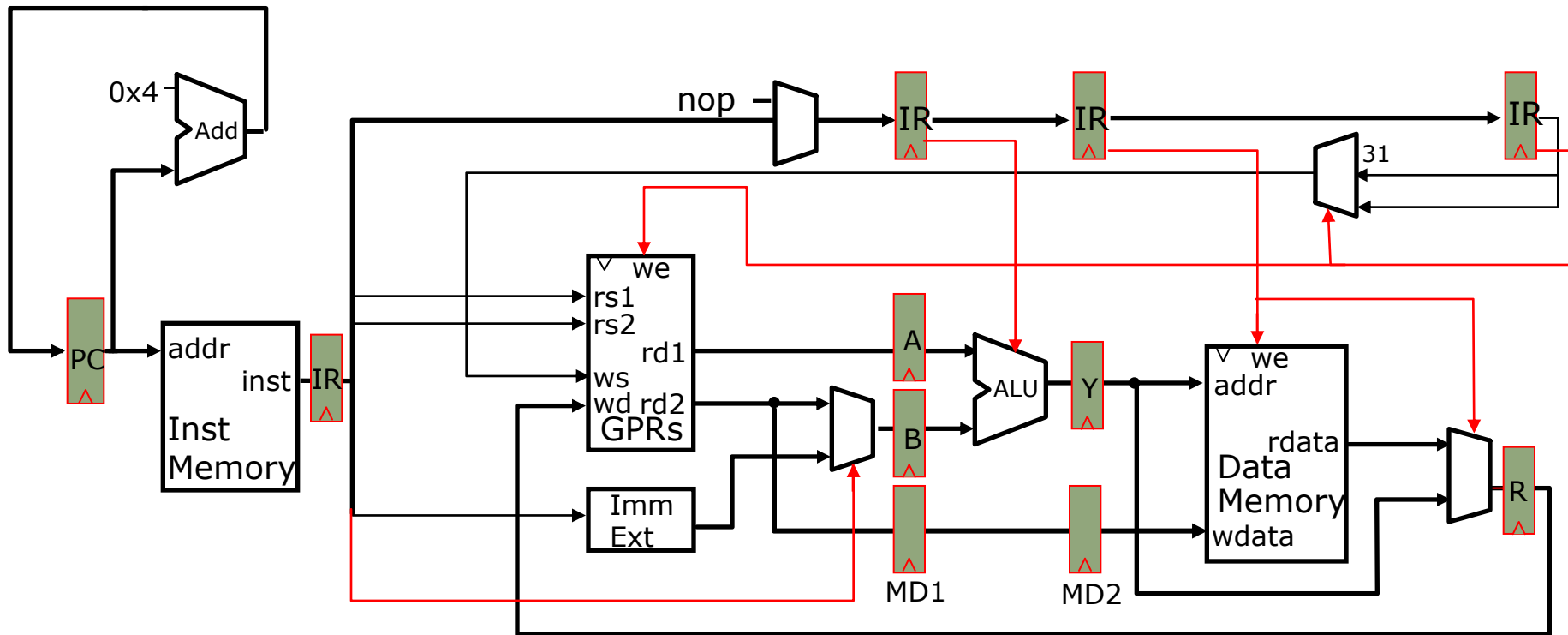


Stalled Stages and Pipeline Bubbles



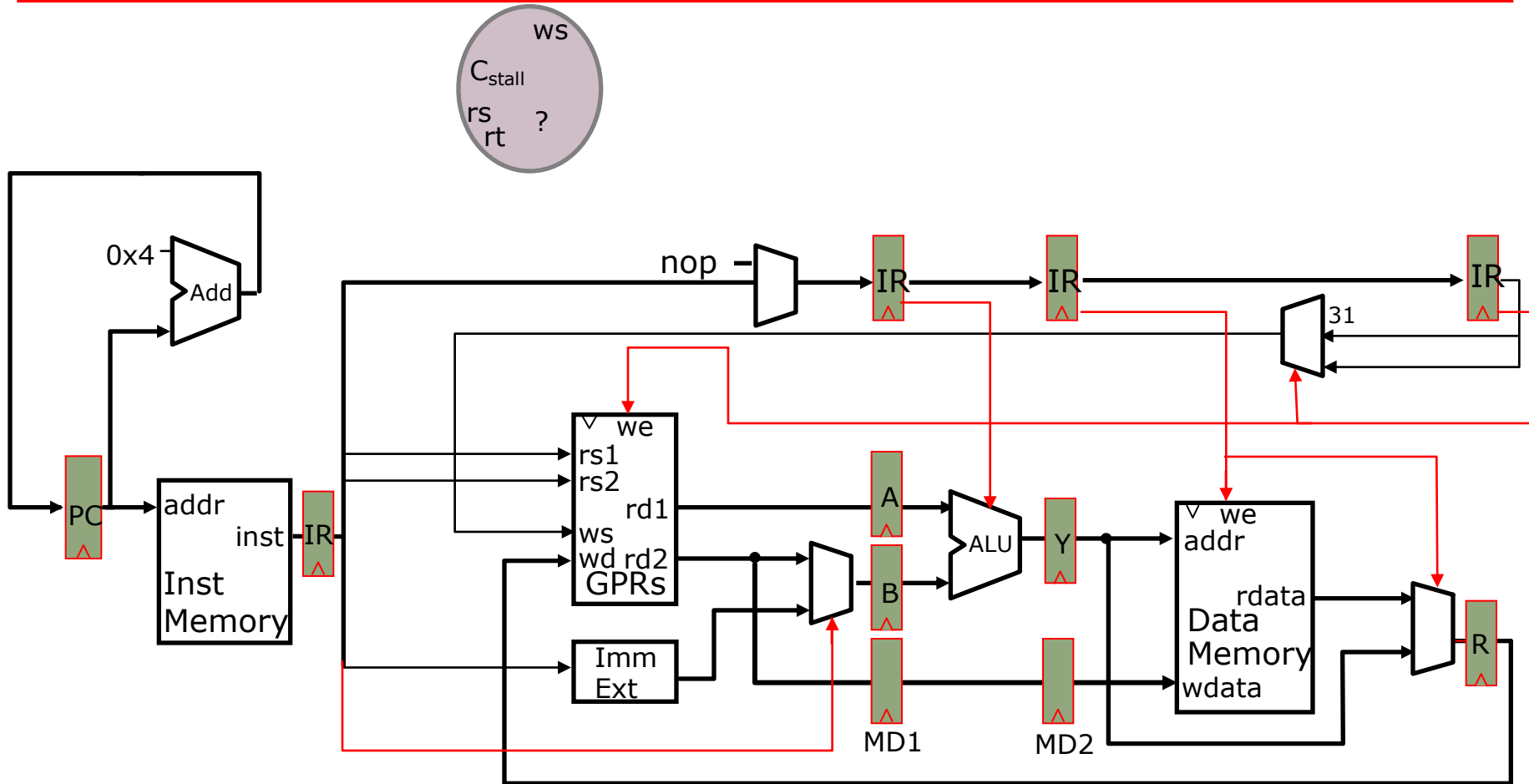
nop ⇒ *pipeline bubble*

Stall Control Logic



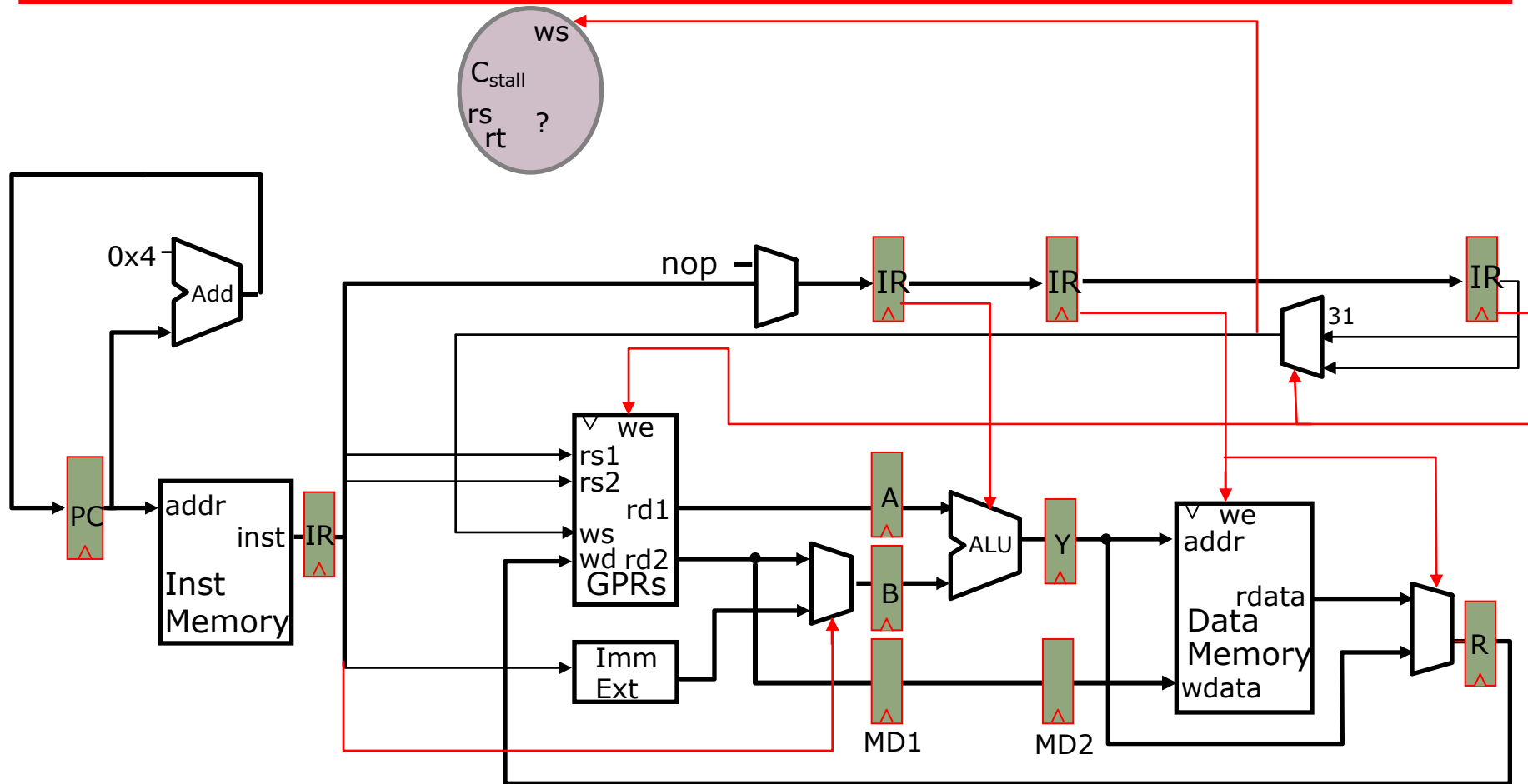
Compare the *source registers* of the instruction in the decode stage with the *destination register* of the *uncommitted instructions*.

Stall Control Logic



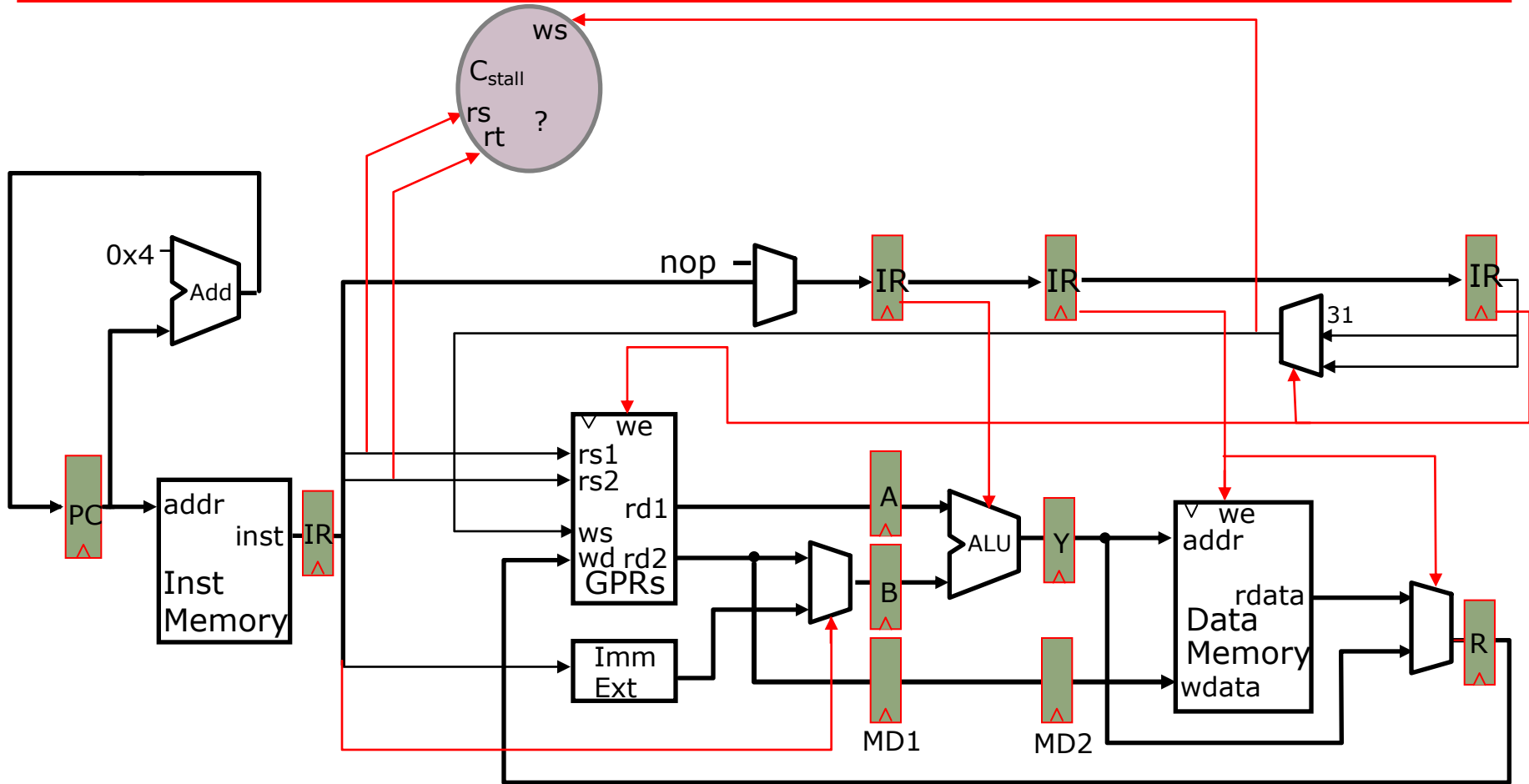
Compare the *source registers* of the instruction in the decode stage with the *destination register* of the *uncommitted instructions*.

Stall Control Logic



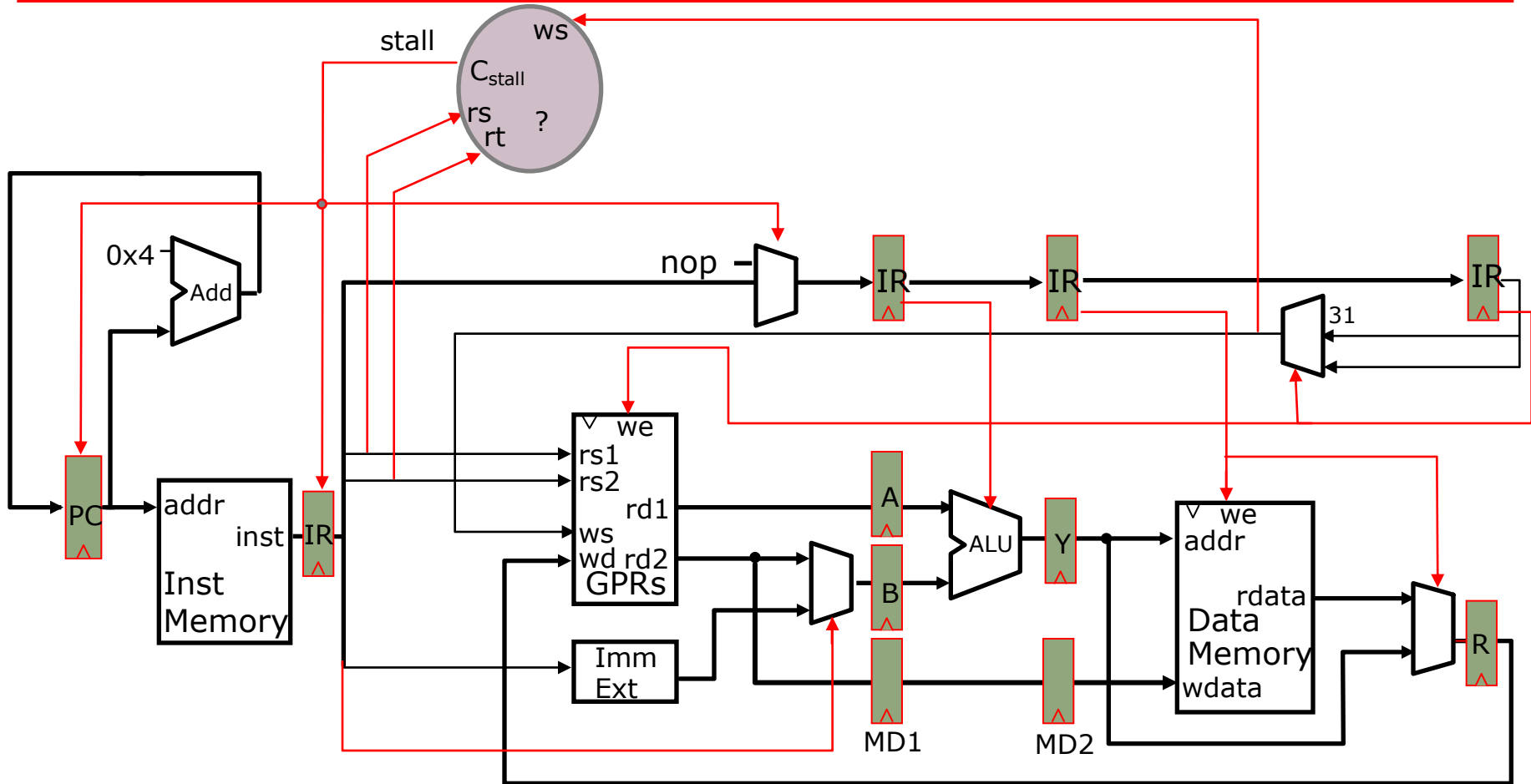
Compare the *source registers* of the instruction in the decode stage with the *destination register* of the *uncommitted instructions*.

Stall Control Logic



Compare the *source registers* of the instruction in the decode stage with the *destination register* of the *uncommitted instructions*.

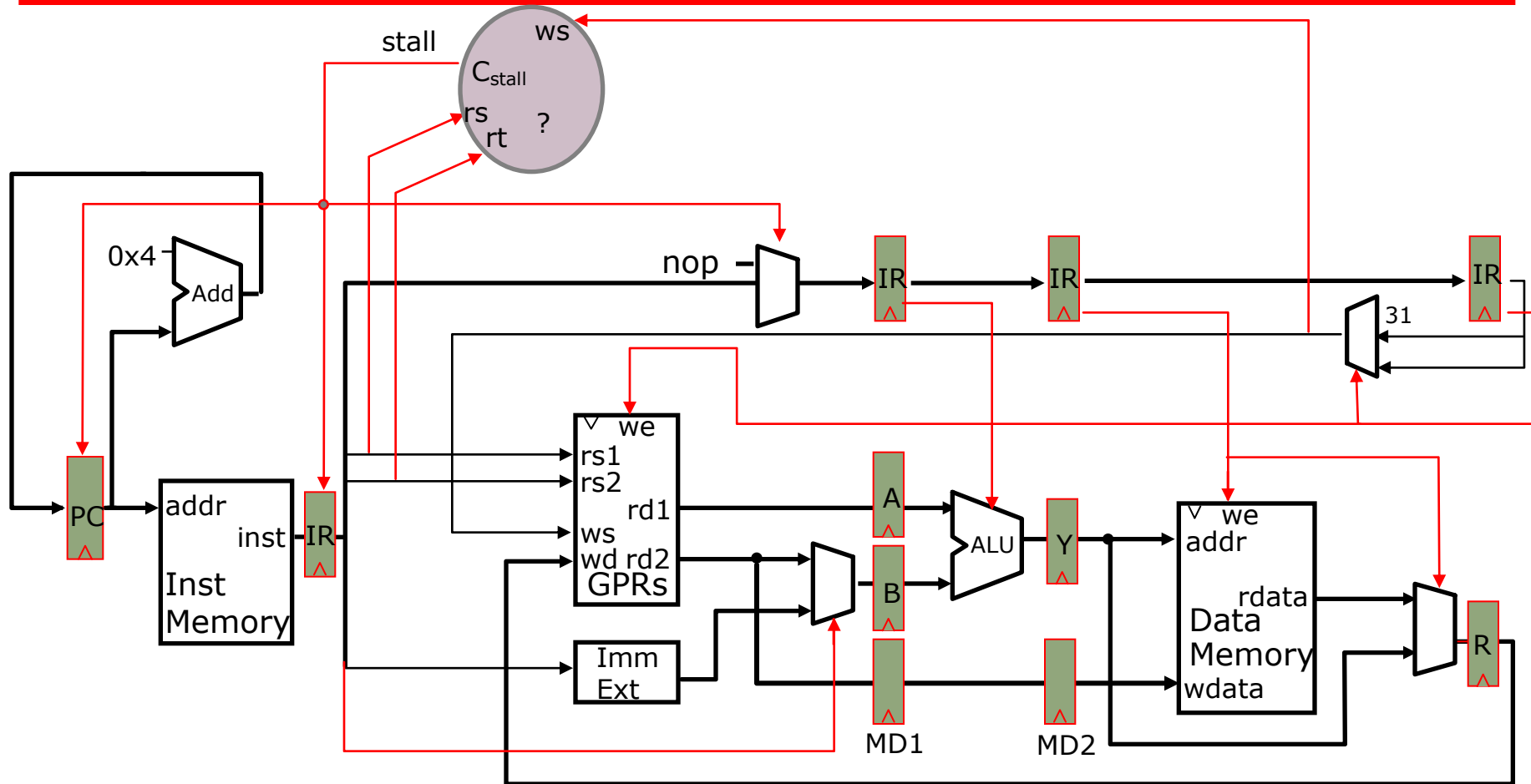
Stall Control Logic



Compare the *source registers* of the instruction in the decode stage with the *destination register* of the *uncommitted instructions*.

Stall Control Logic

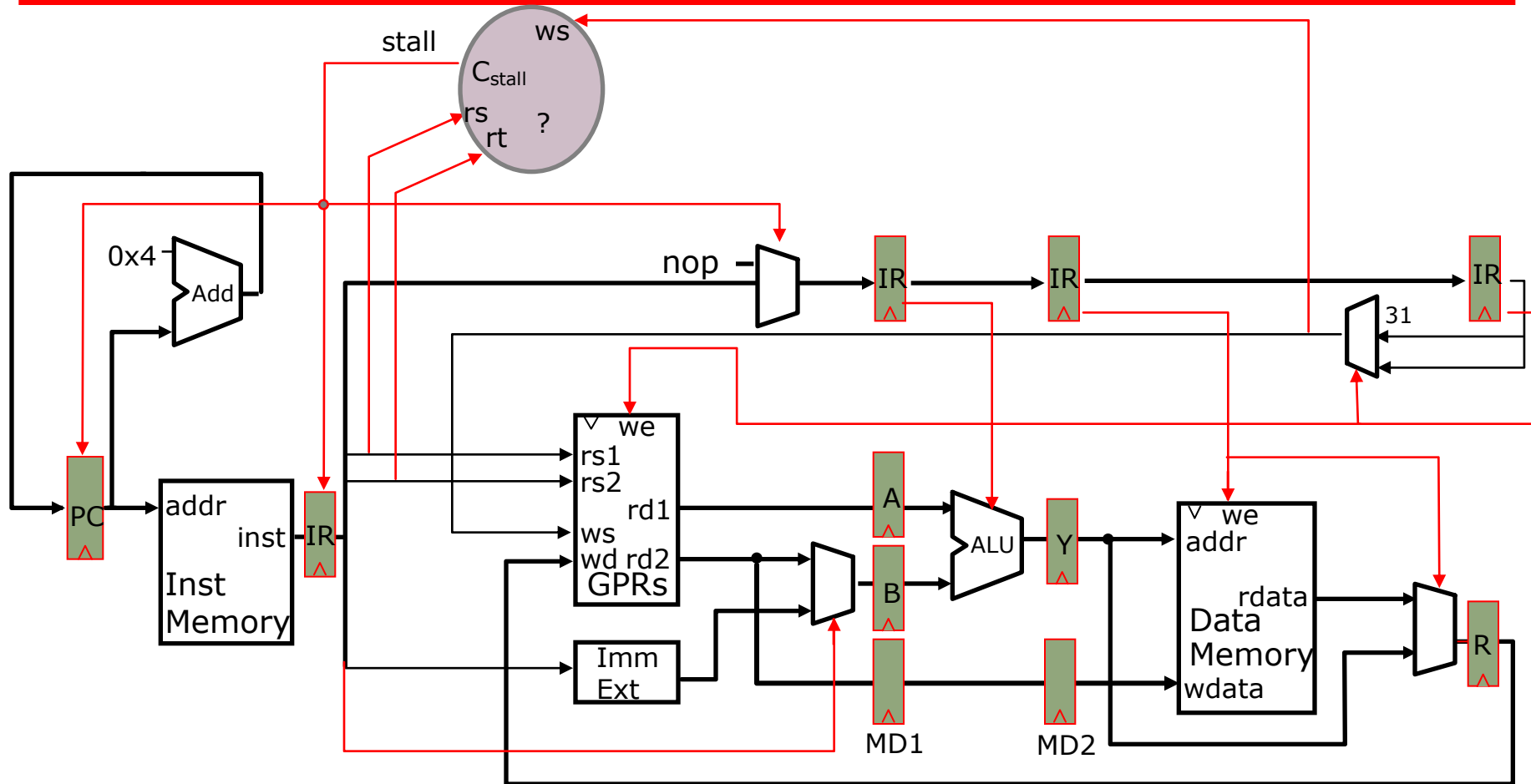
ignoring jumps & branches



Should we always stall if the rs field matches some rd?

Stall Control Logic

ignoring jumps & branches

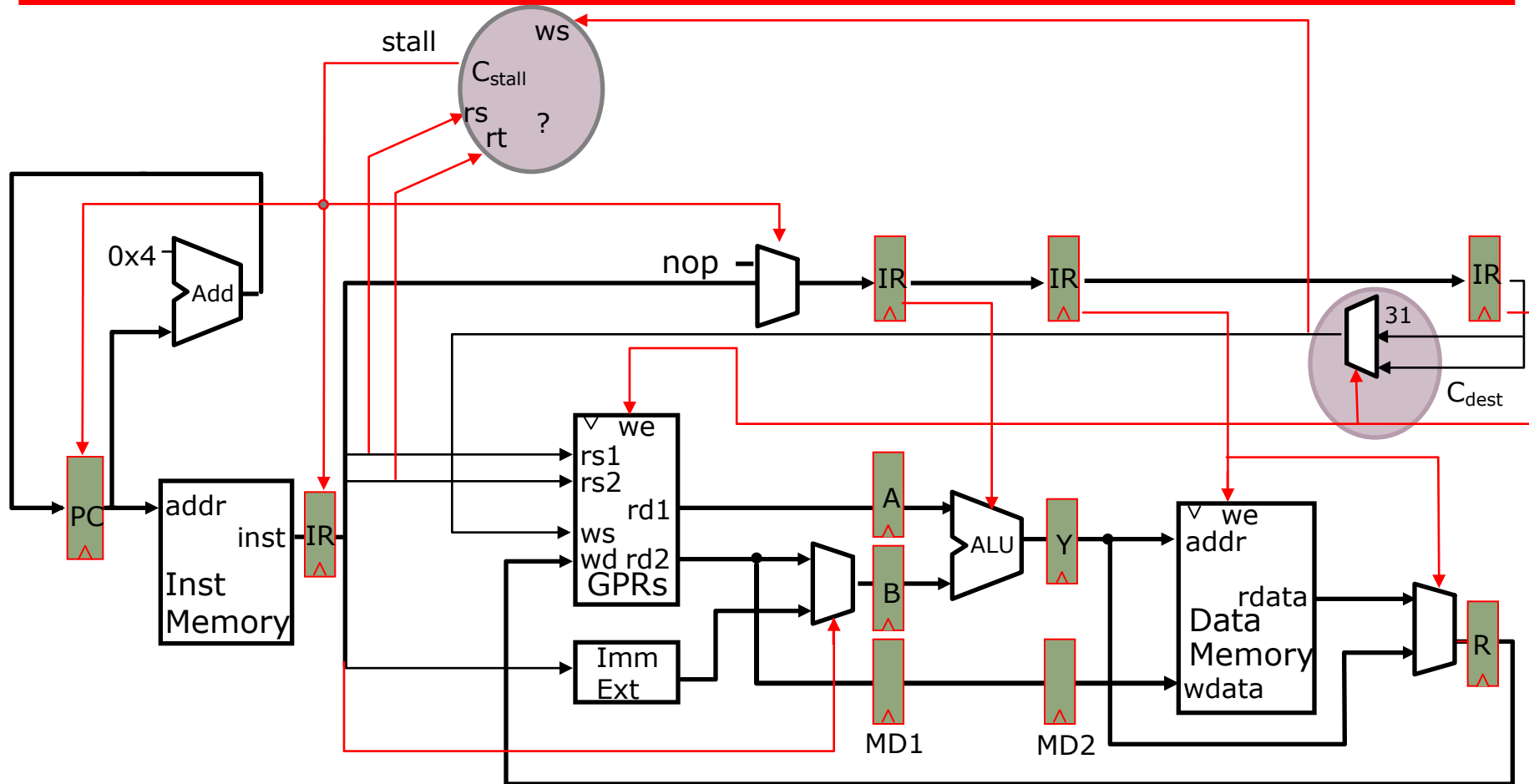


Should we always stall if the rs field matches some rd?

not every instruction writes a register \Rightarrow we
 not every instruction reads a register \Rightarrow re

Stall Control Logic

ignoring jumps & branches

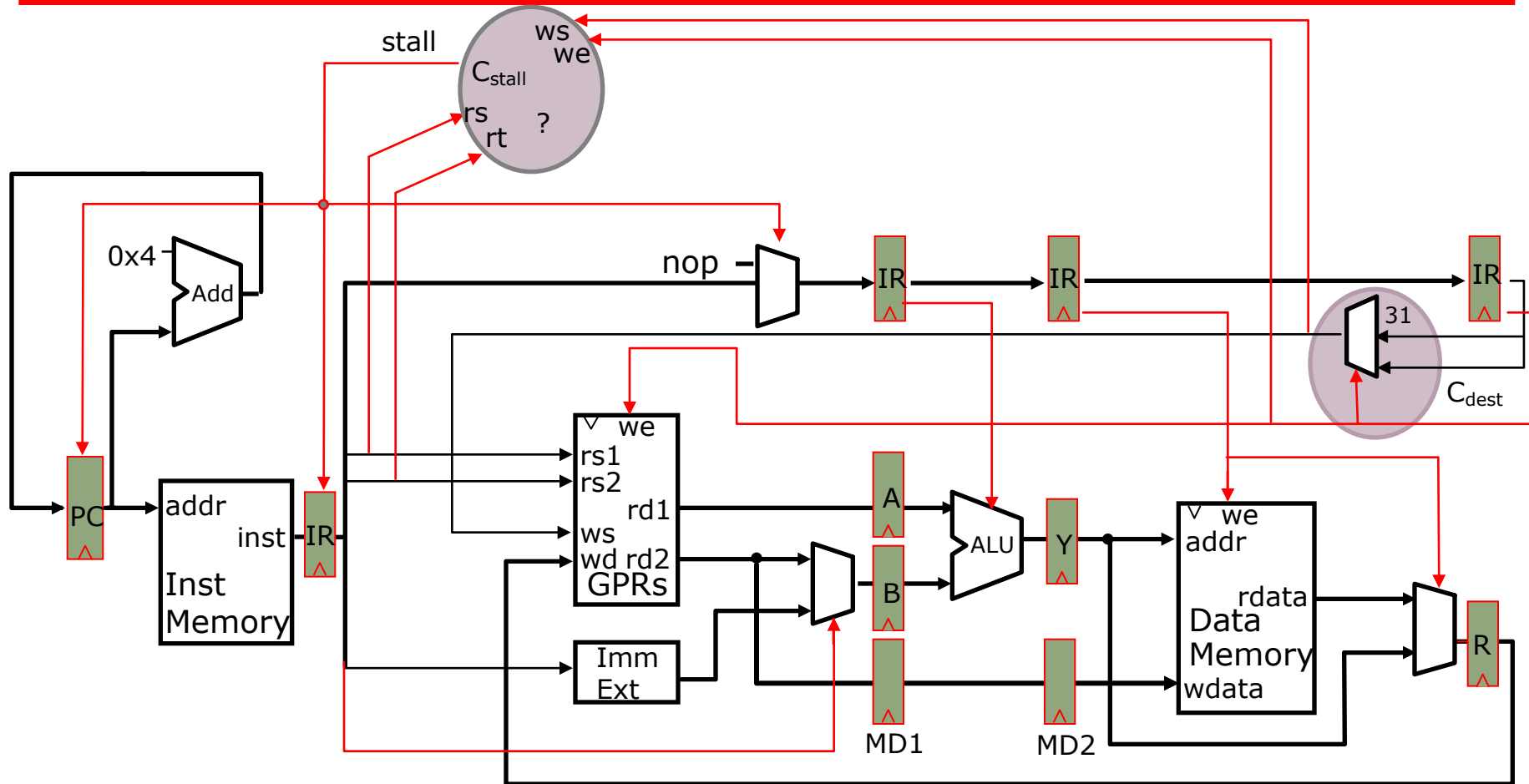


Should we always stall if the rs field matches some rd?

not every instruction writes a register \Rightarrow we
 not every instruction reads a register \Rightarrow re

Stall Control Logic

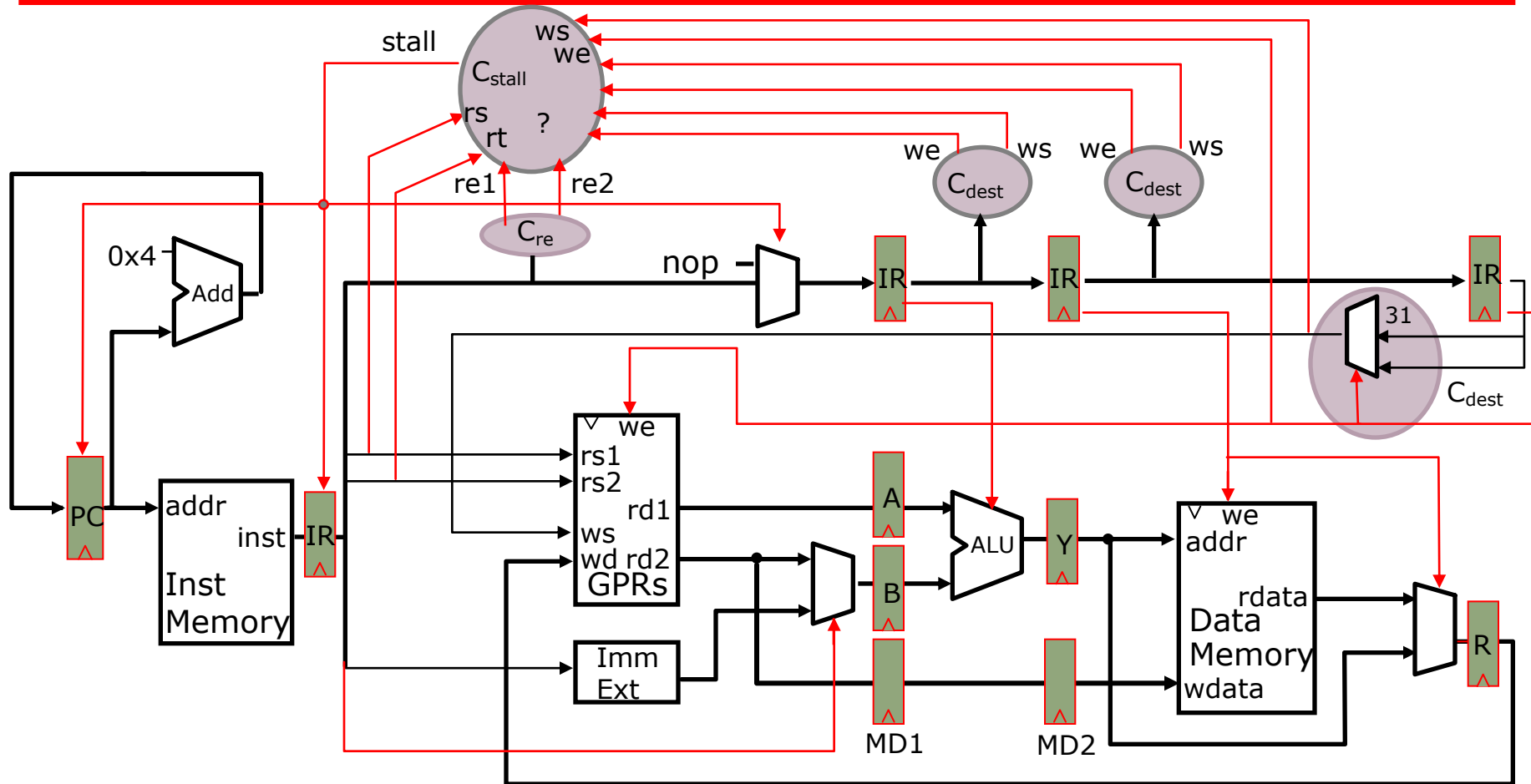
ignoring jumps & branches



Should we always stall if the rs field matches some rd?
 not every instruction writes a register \Rightarrow we
 not every instruction reads a register \Rightarrow re

Stall Control Logic

ignoring jumps & branches



Should we always stall if the *rs* field matches some *rd*?
 not every instruction writes a register \Rightarrow *we*
 not every instruction reads a register \Rightarrow *re*

Thank you!