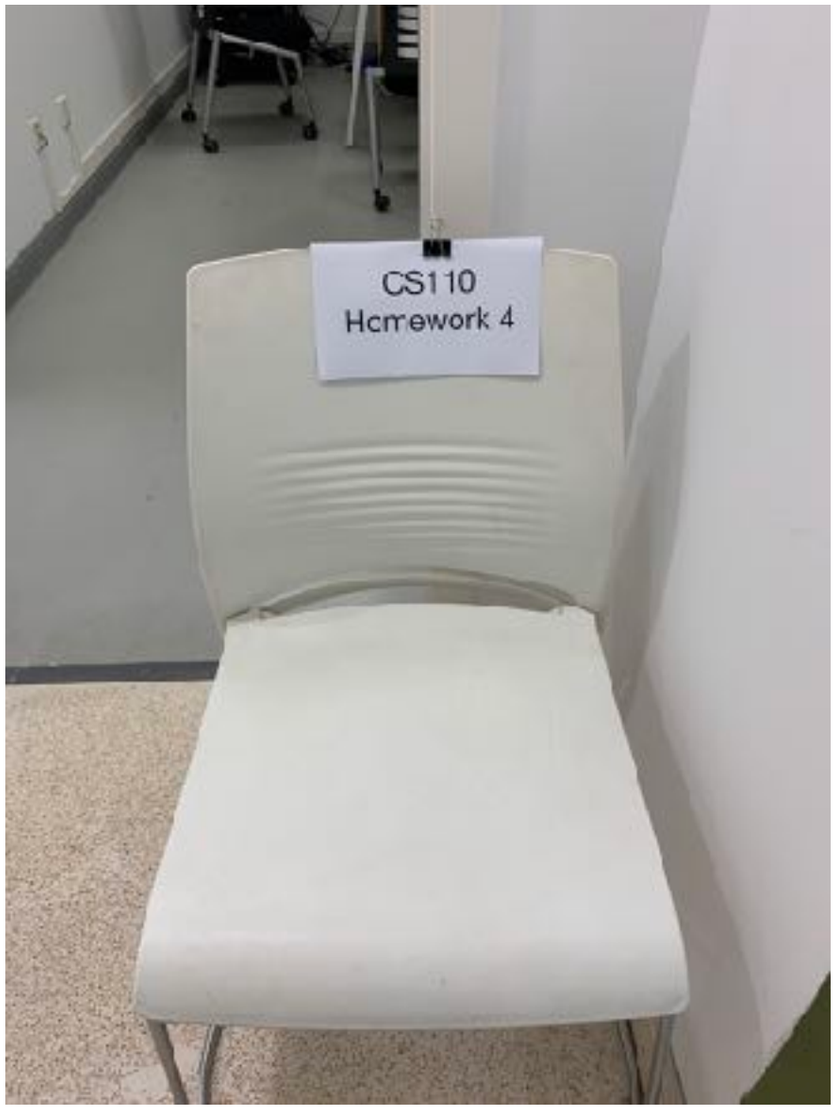
Course Info

- Lab 6 next week, prepare before lab sessions! Keep an eye on piazza.
- Project 1.2 ddl March 31st. Project 2.1 released next week!
- This week discussion on ALU & FSM. Next week discussion on datapath.
- Mid-term I answer & score will be released before next week. If you have questions, feel free to ask on Piazza.

Course Info

- HW4 will be released, keep an eye on piazza. Submit your paper homework to the picture below (SIST 3-322). There will be a box. Put into the box. Remember to add your name. You have only one chance to submit and cannot be withdrawn.



信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture
Datapath & Controller

Instructors:

Siting Liu & Chundong Wang

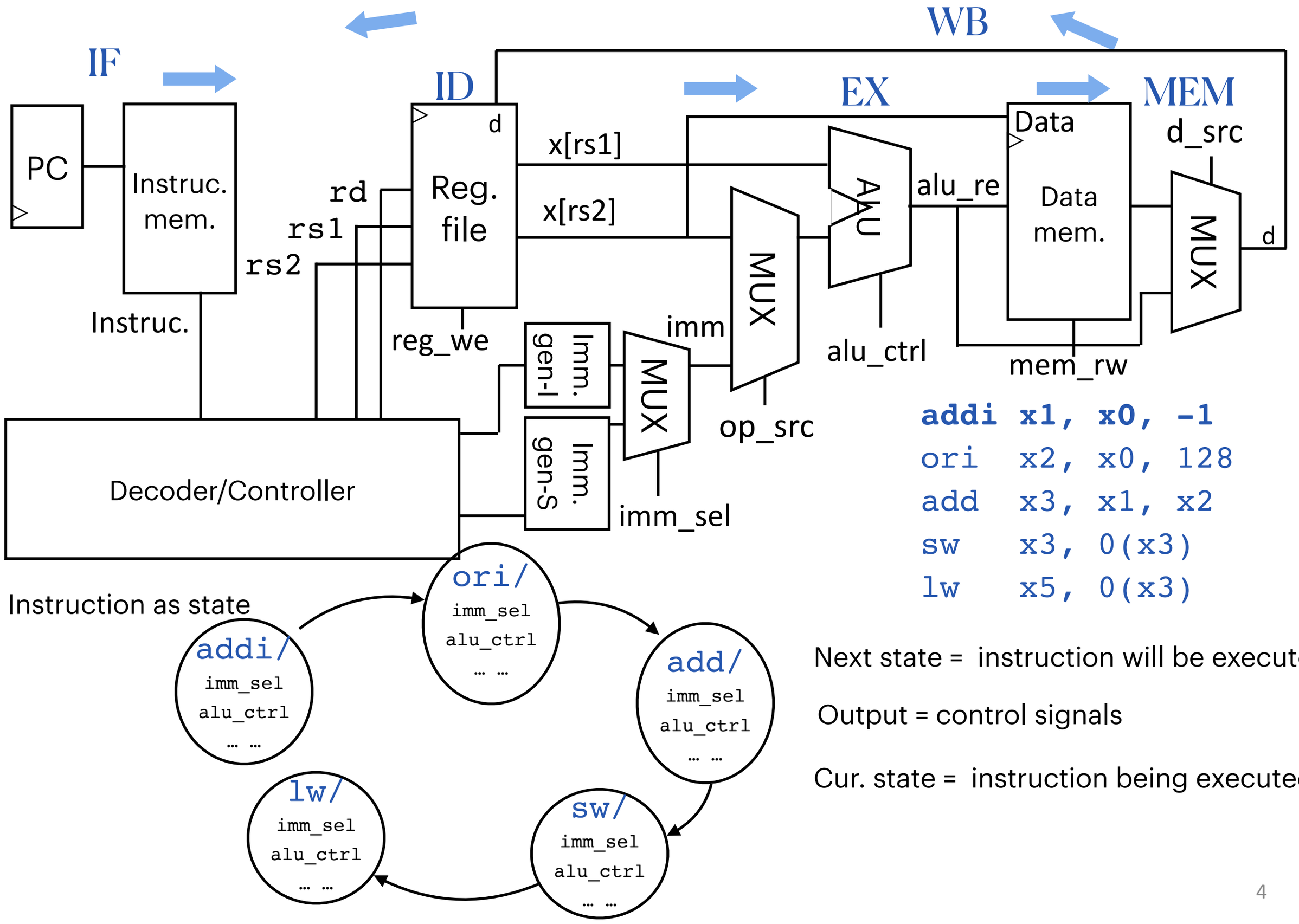
Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2023/index.html

School of Information Science and Technology (SIST)

ShanghaiTech University

2023/3/5

Controller as FSM




```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
```

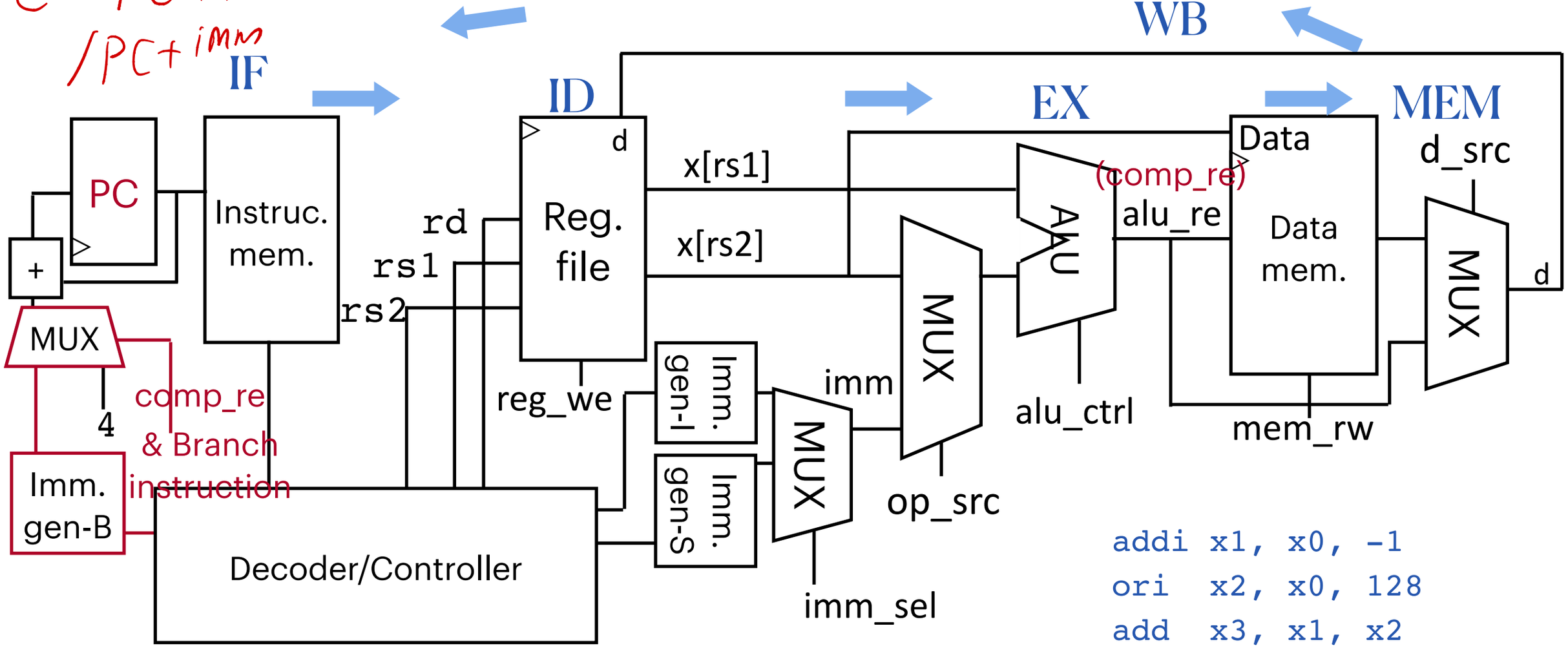
Instruction as state

Next state = instruction will be executed

Output = control signals

Cur. state = instruction being executed

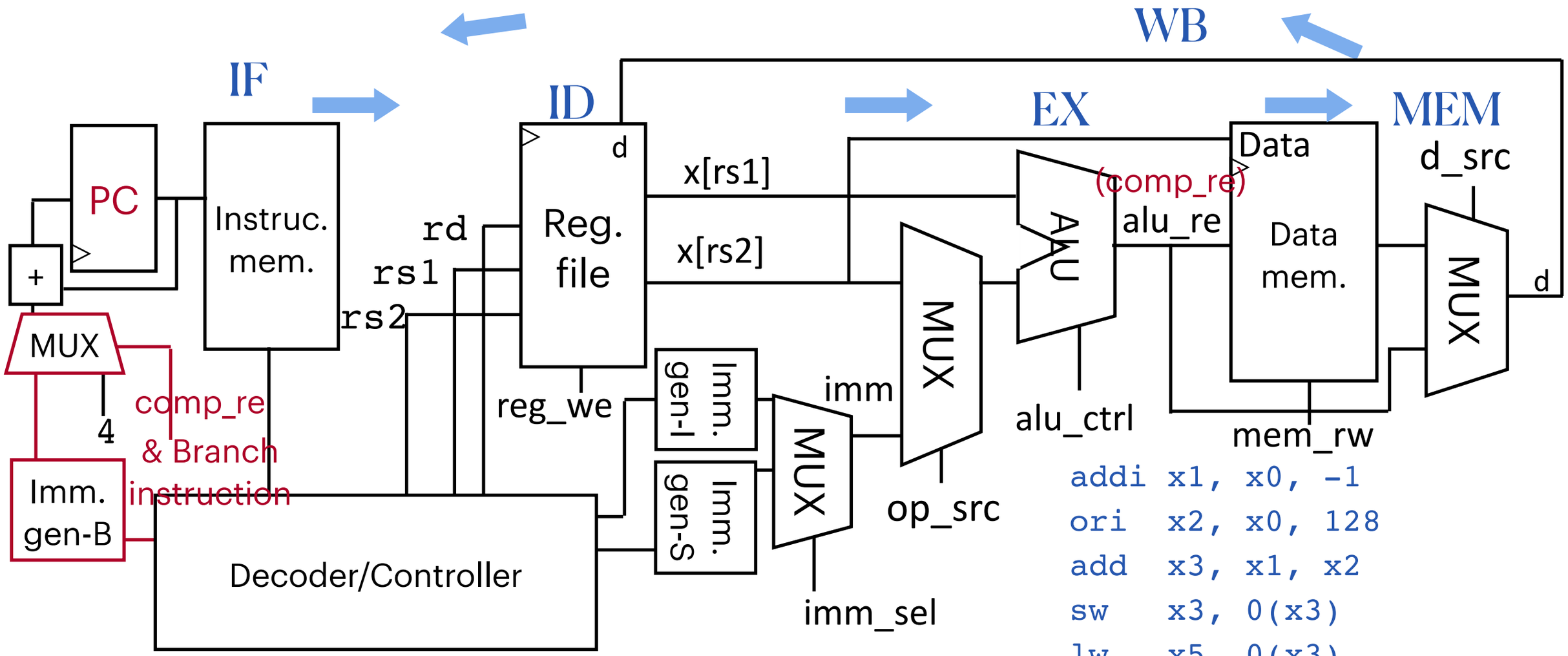
PC = PC+4
/PC+imm

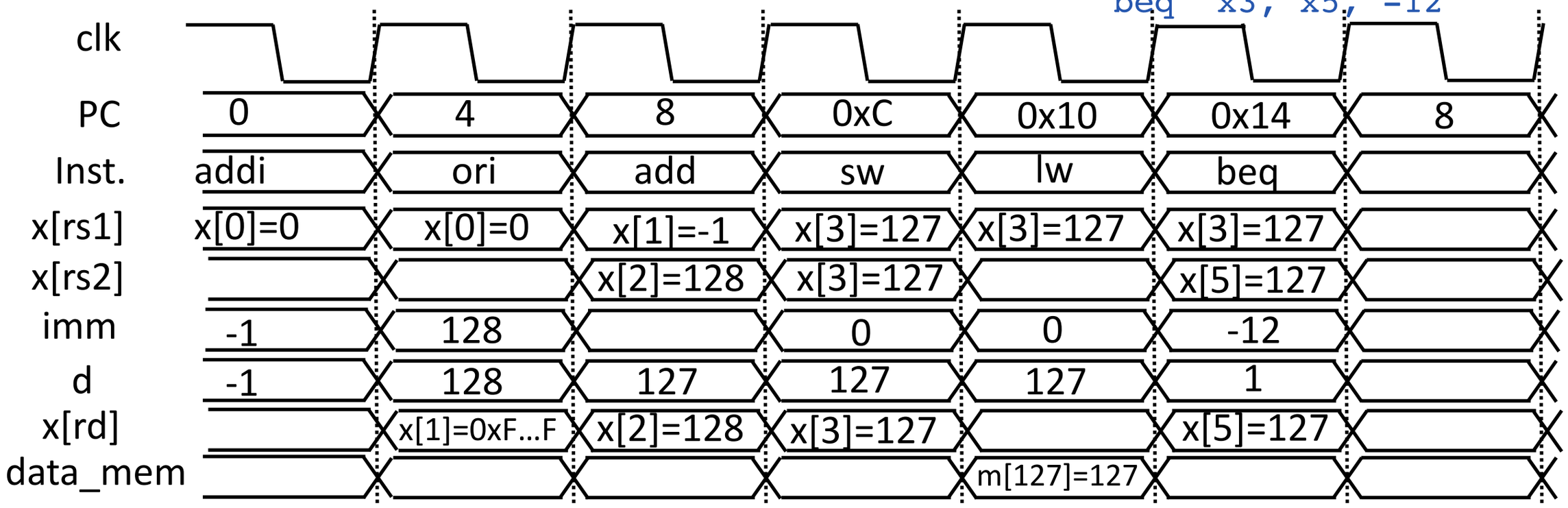



```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
beq  x3, x5, -12
```

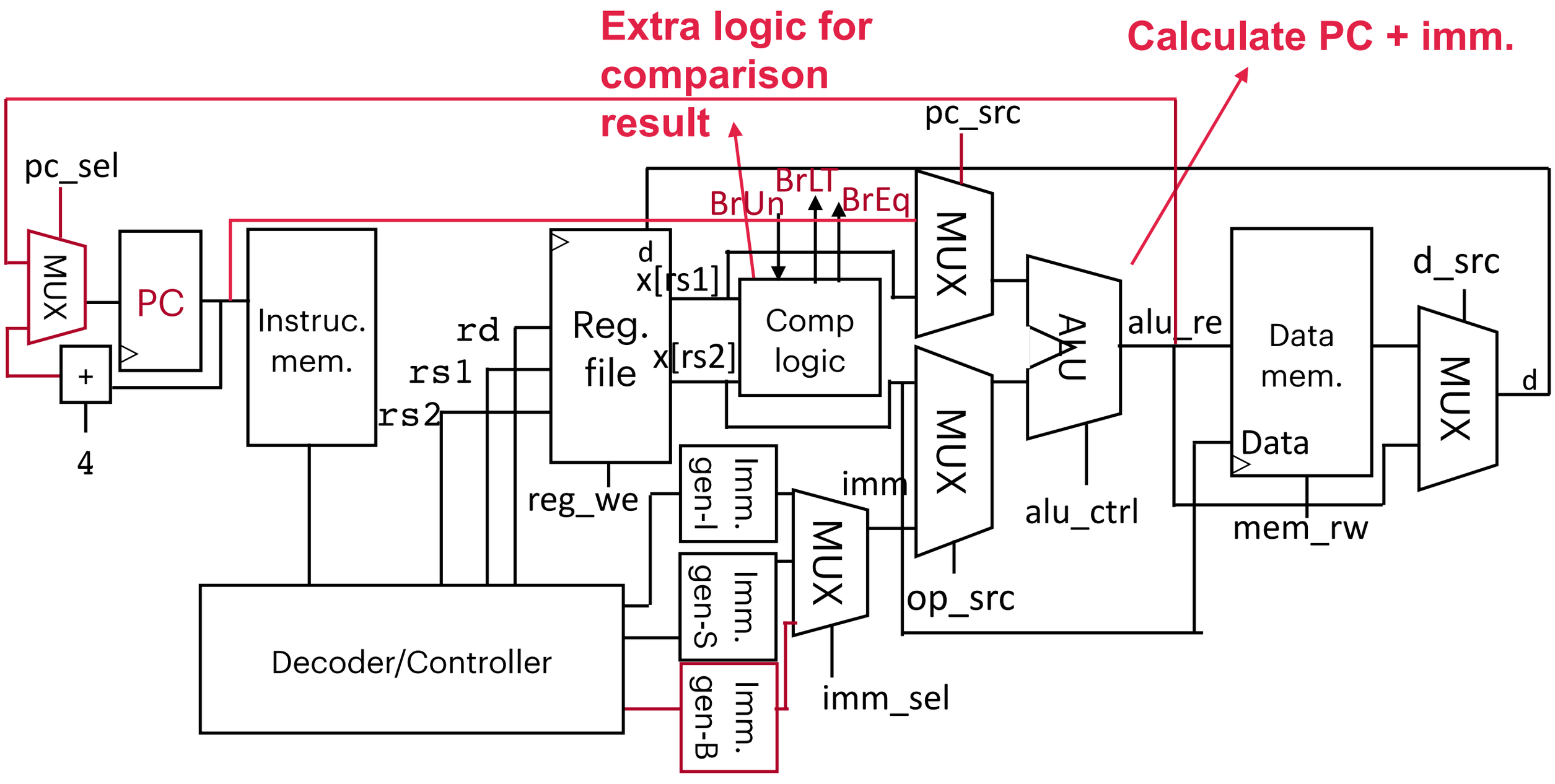
	addi	add	sw	lw	beq
reg_we	1	1	0	1	0
mem_rw	R	R	W	R	R
alu_ctrl	add	add	add	add	comp
imm_sel	I	*	S	I	B*
d_src	alu	alu	*	mem	alu*
op_src	imm	reg	imm	imm	~~reg~~ imm
PC_mux	+4	+4	+4	+4	comp_re & Branch instruction

Output = control signals


IF
ID
EX
MEM
WB
PC
+
MUX
4
Imm.
gen-B
Instruc.
mem.
comp_re
& Branch
instruction
Decoder/Controller
rd
rs1
rs2
d
Reg.
file
reg_we
x[rs1]
x[rs2]
Imm.
gen-I
Imm.
gen-S
MUX
imm
imm_sel
MUX
op_src
ALU
(comp_re)
alu_re
alu_ctrl
Data
Data
mem.
mem_rw
d_src
MUX
d
addi x1, x0, -1
ori x2, x0, 128
add x3, x1, x2
sw x3, 0(x3)
lw x5, 0(x3)
beq x3, x5, -12

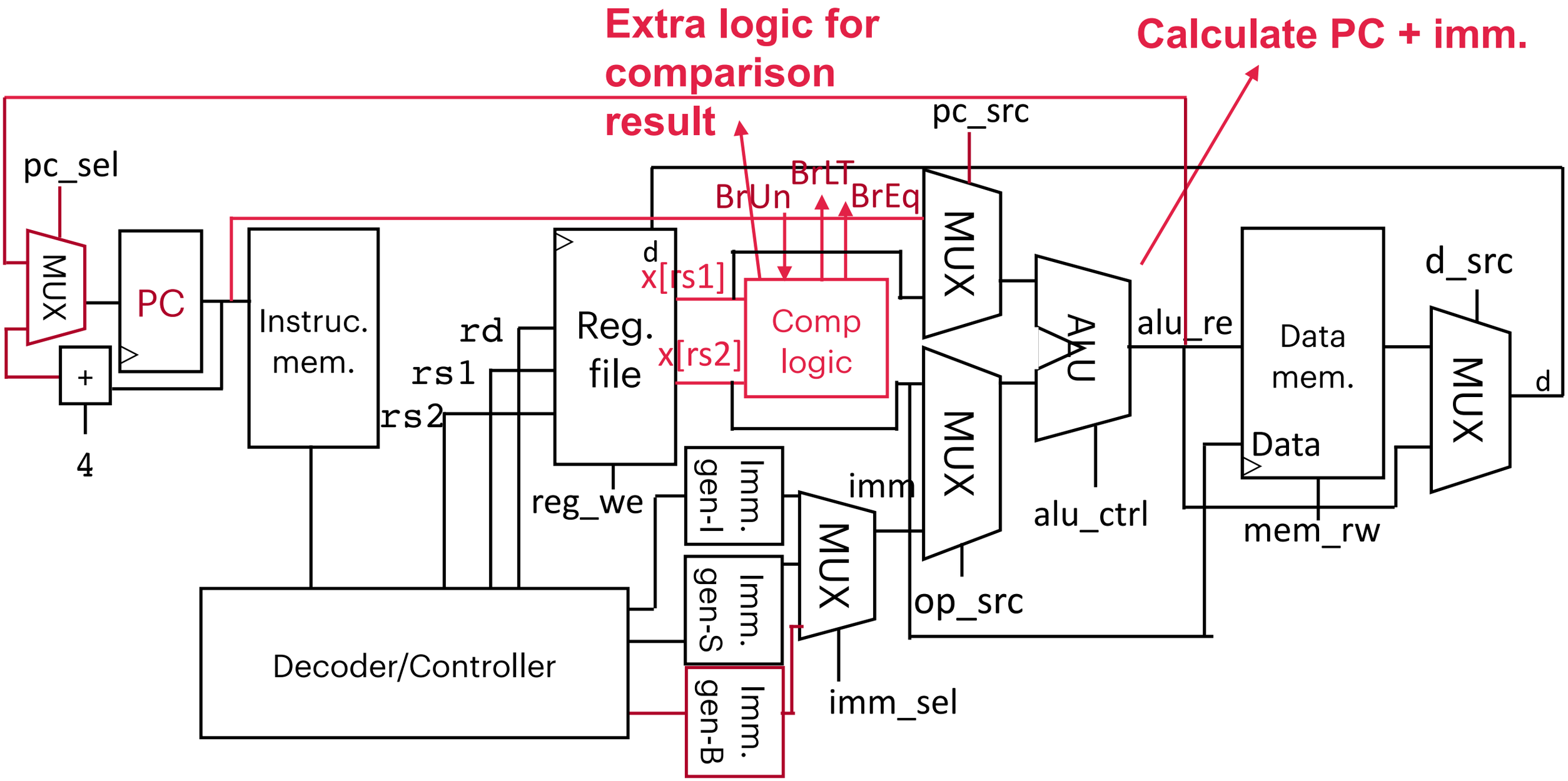

clk
PC
Inst.
x[rs1]
x[rs2]
imm
d
x[rd]
data_mem
0
4
8
0xC
0x10
0x14
8
addi
ori
add
sw
lw
beq
x[0]=0
x[0]=0
x[1]=-1
x[3]=127
x[3]=127
x[3]=127
x[2]=128
x[3]=127
x[5]=127
-1
128
0
0
-12
-1
128
127
127
127
1
x[1]=0xF…F
x[2]=128
x[3]=127
x[5]=127
m[127]=127

Another Implementation



There Are a Thousand Hamlets in a Thousand People's Eyes. — Shakespeare

Compare Logic

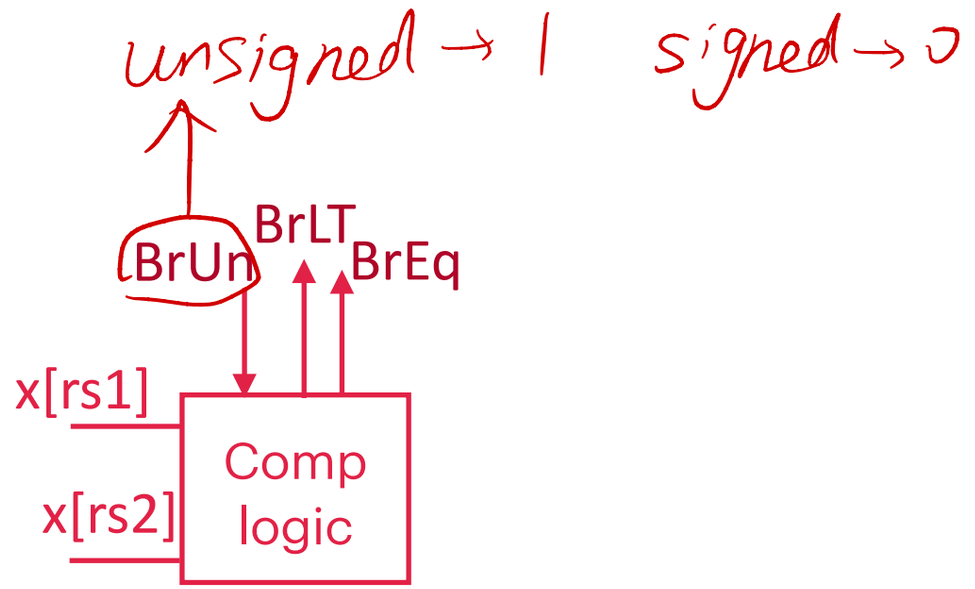


BrUn: control signal (input), whether it is a unsigned or signed comparison;

BrLT: 1 if less than, otherwise 0; output signal to the decoder/controller;

BrEq: 1 if equal, otherwise 0; output signal to the decoder/controller;

Compare Logic



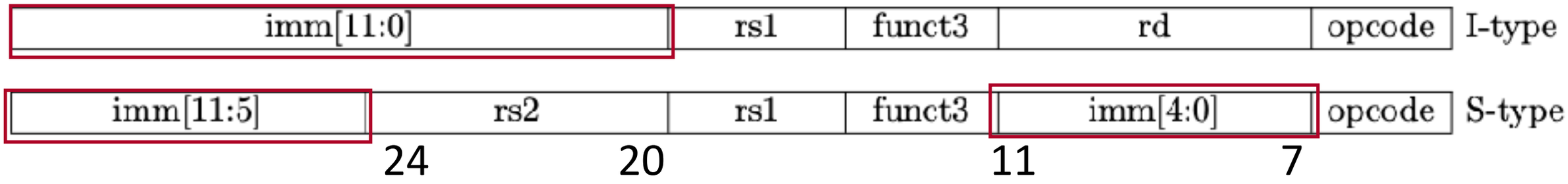
BrUn: control signal (input), whether it is a unsigned or signed comparison;

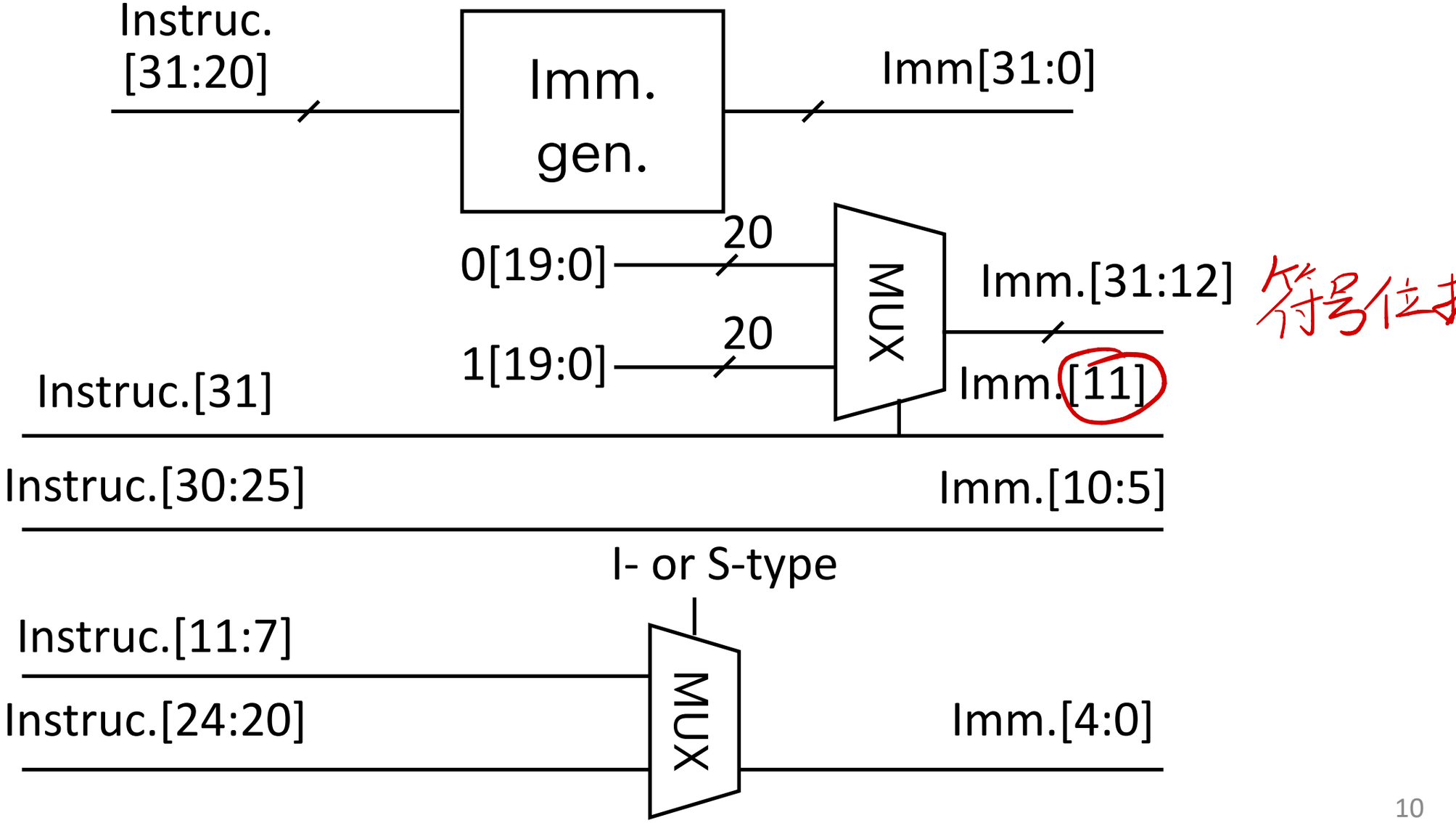
BrLT: 1 if less than, otherwise 0; output signal

BrEq: 1 if equal, otherwise 0; output signal

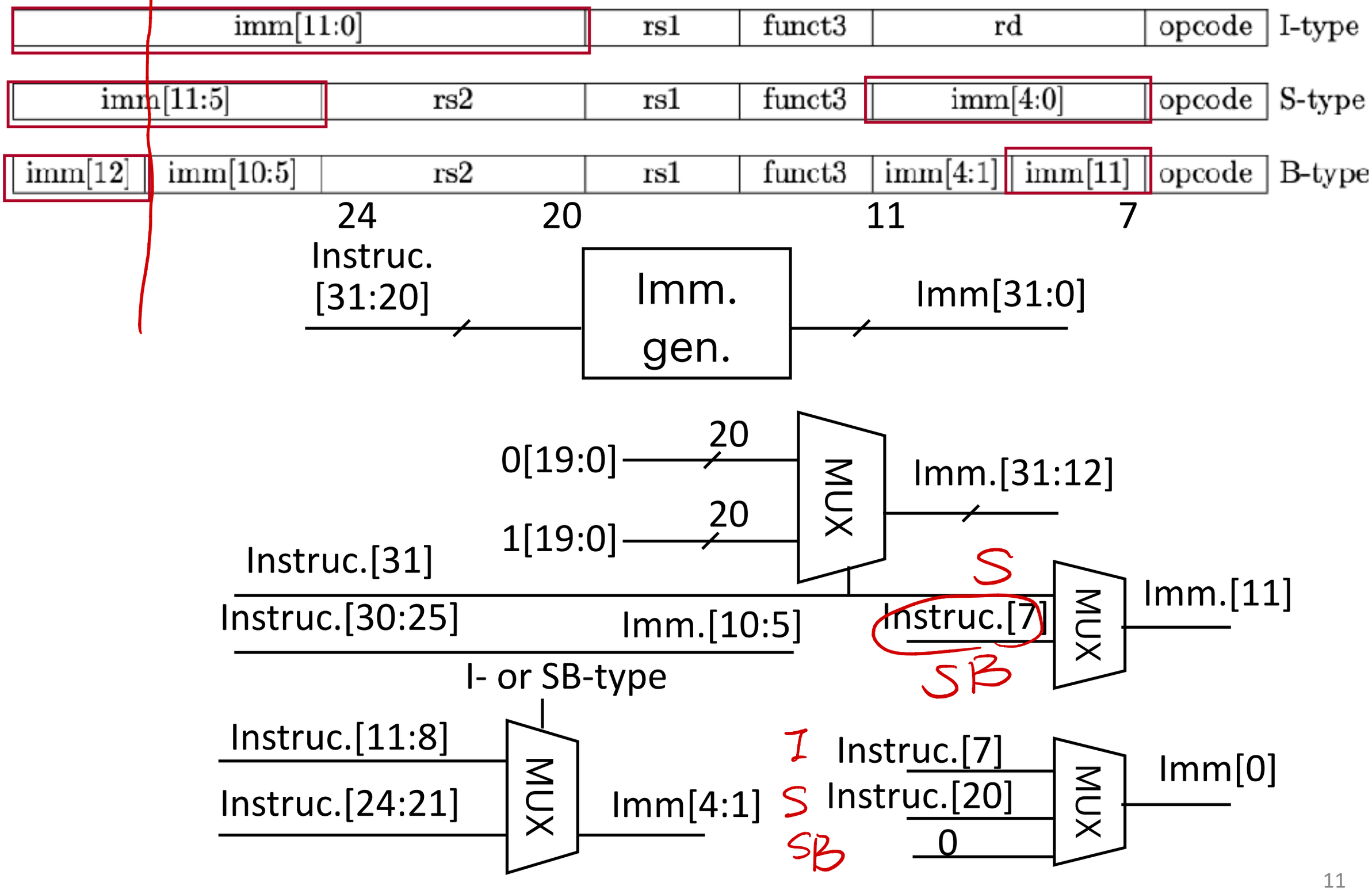
	BrLT	BrEq
BEQ	0	1
BNE	0/1	0
BLT	1	0
BGE	0	0/1

Consideration for IMM Generation

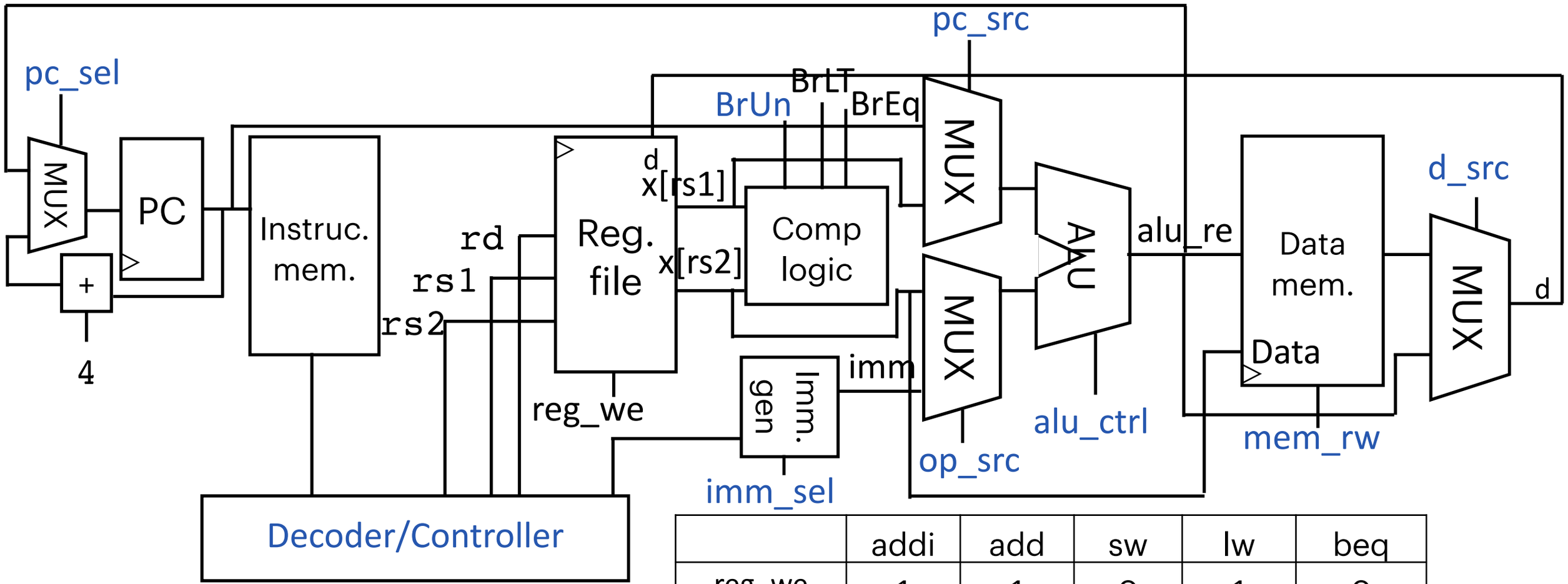




Consideration for IMM Generation



Up to Now

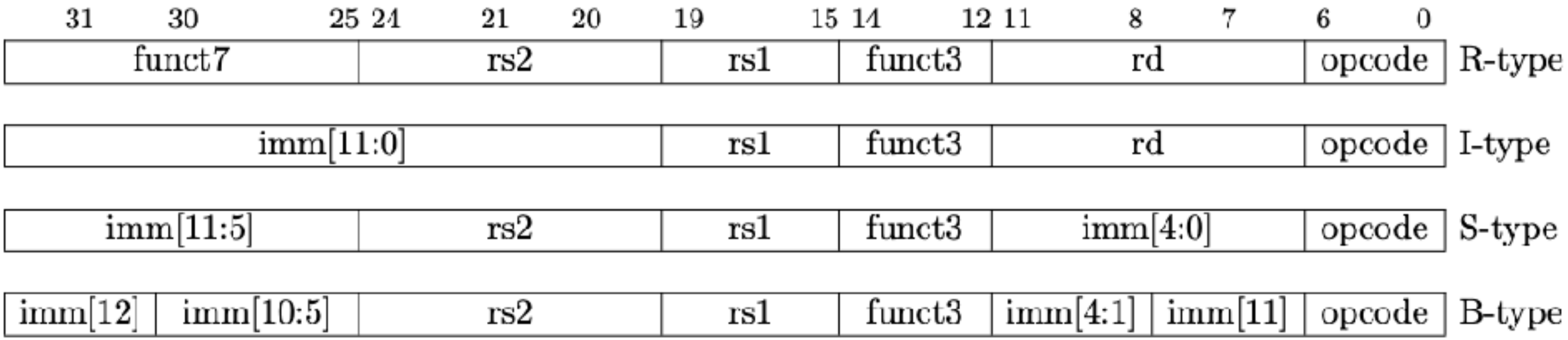


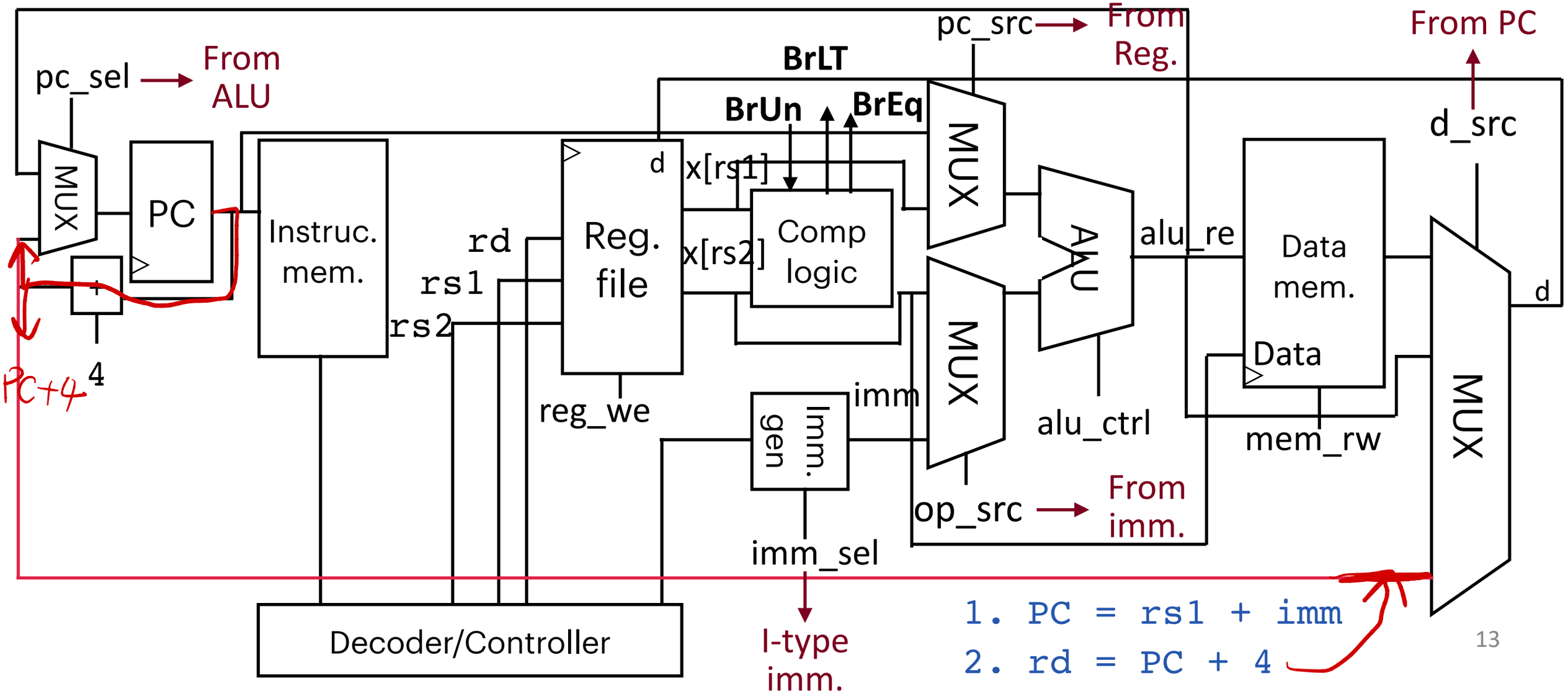
Control signals generated by decoder/controller, according to the instruction

	addi	add	sw	lw	beq
reg_we	1	1	0	1	0
mem_rw	R	R	W	R	R
alu_ctrl	add	add	add	add	add
imm_sel	I	*	S	I	B
d_src	alu	alu	*	mem	*
op_src	imm	reg	imm	imm	imm
pc_sel	+4	+4	+4	+4	Comp.
pc_src	reg	reg	reg	reg	pc
BrUn	*	*	*	*	w/w/o u

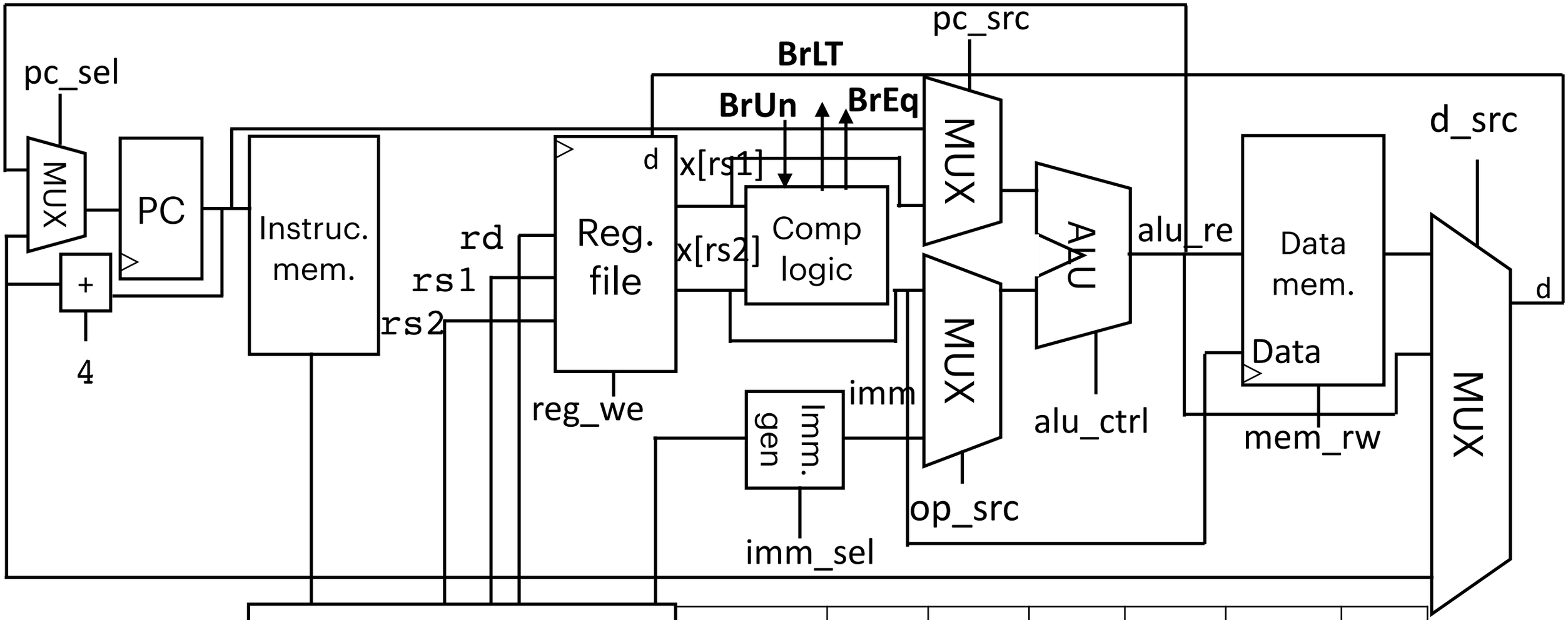
控制讯号 写回 register

Add JALR


31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 rs2 rs1 funct3 rd opcode R-type
imm[11:0] rs1 funct3 rd opcode I-type
imm[11:5] rs2 rs1 funct3 imm[4:0] opcode S-type
imm[12] imm[10:5] rs2 rs1 funct3 imm[4:1] imm[11] opcode B-type


pc_sel
From ALU
pc_src
From Reg.
From PC
BrLT
BrUn
BrEq
MUX
PC
+
4
PC+4
Instruc. mem.
rd
rs1
rs2
Reg. file
d
x[rs1]
x[rs2]
Comp logic
reg_we
Imm. gen
imm
ALU
alu_re
alu_ctrl
Data mem.
Data
mem_rw
d_src
d
op_src
From imm.
imm_sel
I-type imm.
Decoder/Controller
1. PC = rs1 + imm
2. rd = PC + 4

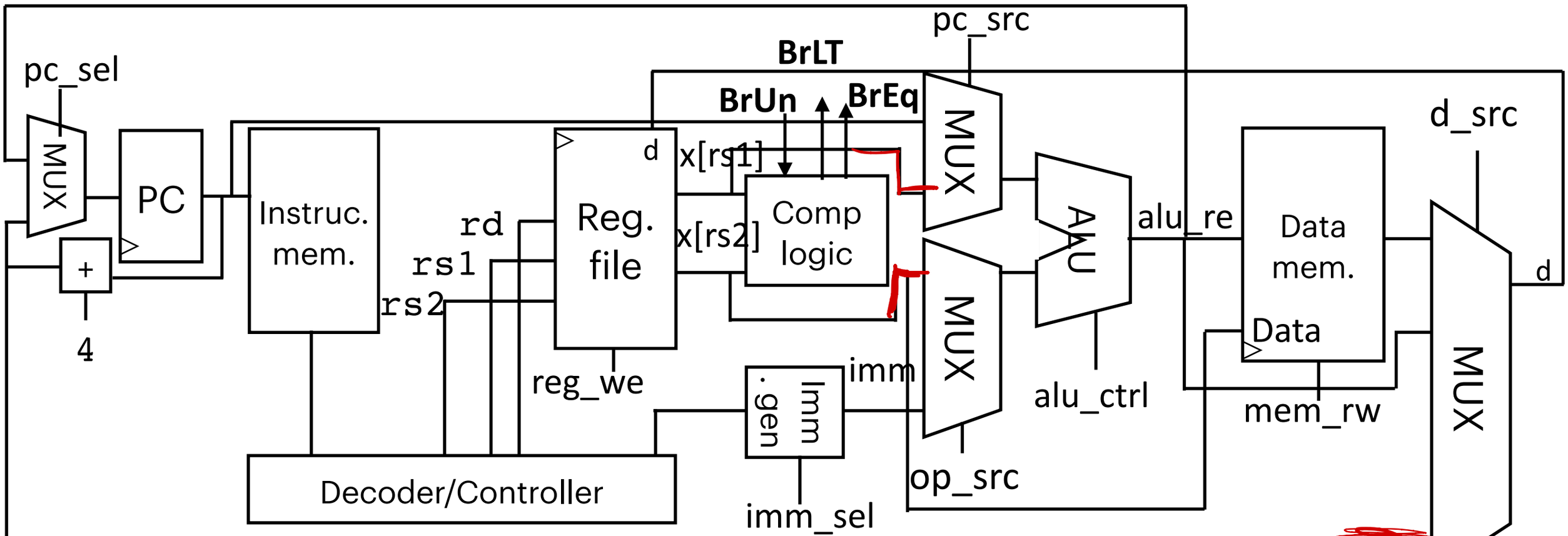
Add JALR



	addi	add	sw	lw	beq	jalr
reg_we	1	1	0	1	0	
mem_rw	R	R	W	R	R	
alu_ctrl	add	add	add	add	add	
imm_sel	I	*	S	I	B	
d_src	alu	alu	*	mem	*	
op_src	imm	reg	imm	imm	imm	
pc_sel	+4	+4	+4	+4	Comp.	
pc_src						

Add JALR

PC = R[rs1] + imm



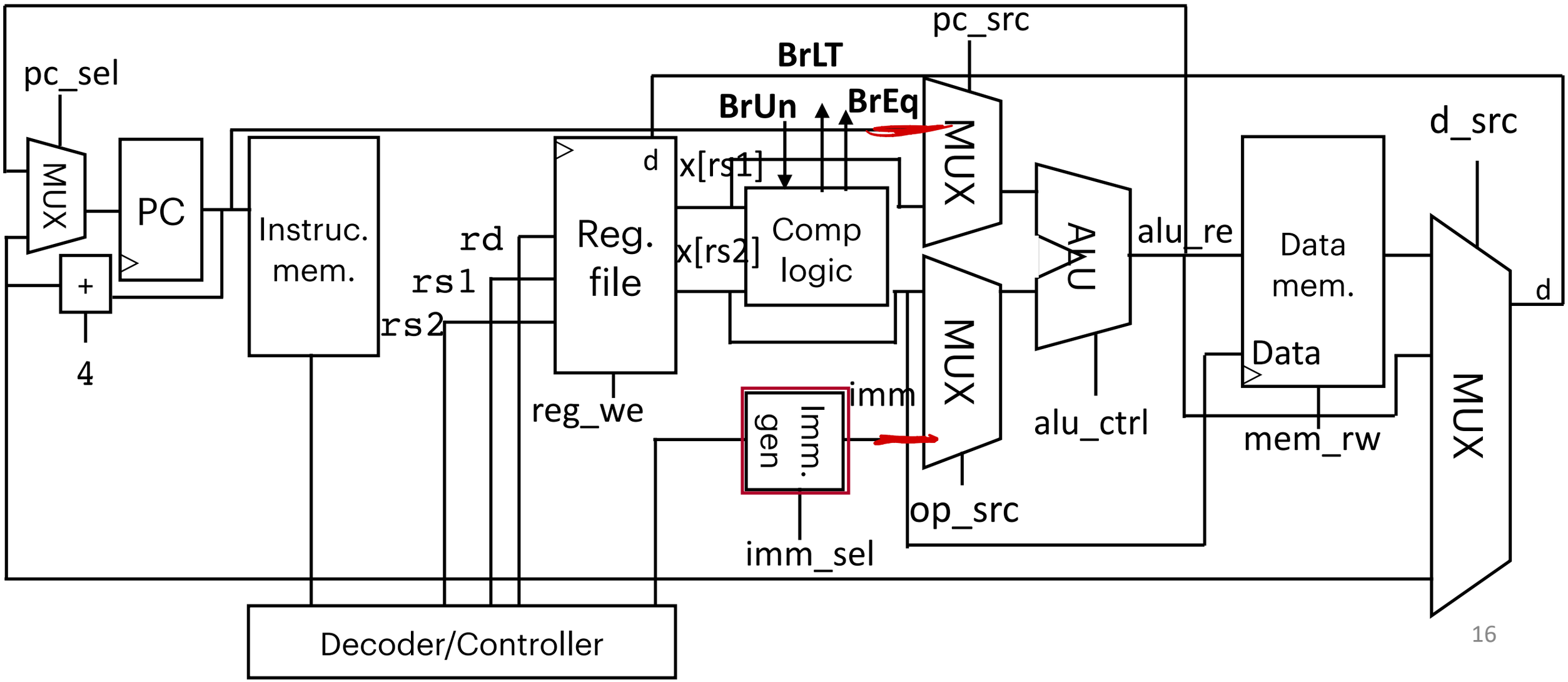
rd = PC + 4

	addi	add	sw	lw	beq	jalr
reg_we	1	1	0	1	0	
mem_rw	R	R	W	R	R	
alu_ctrl	add	add	add	add	add	
imm_sel	I	*	S	I	B	
d_src	alu	alu	*	mem	*	
op_src	imm	reg	imm	imm	imm	
pc_sel	+4	+4	+4	+4	Comp.	
pc_src	reg	reg	reg	reg	pc	
BrUn	*	*	*	*	w/w/o u	

Add JAL

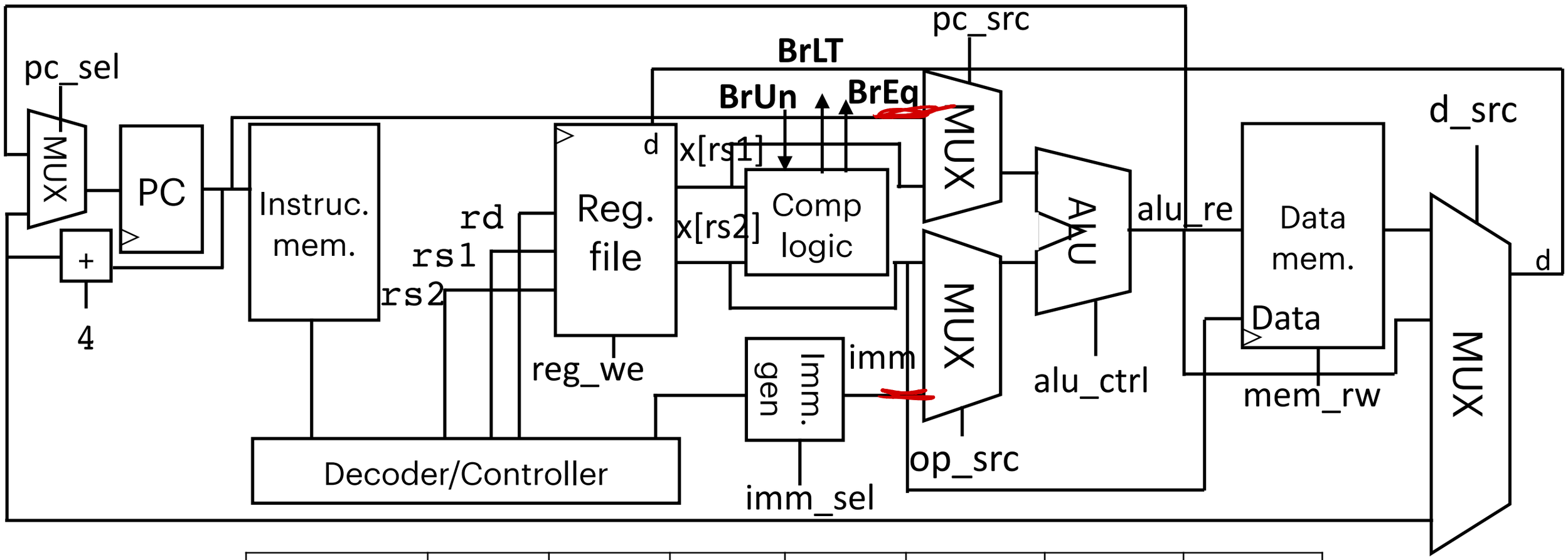
```
1. PC = PC + imm
2. rd = PC + 4
```

imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2		rs1	funct3	imm[4:0]	opcode	S-type
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20]	imm[10:1]	imm[11]	imm[19:12]		rd	opcode	J-type



Add JAL

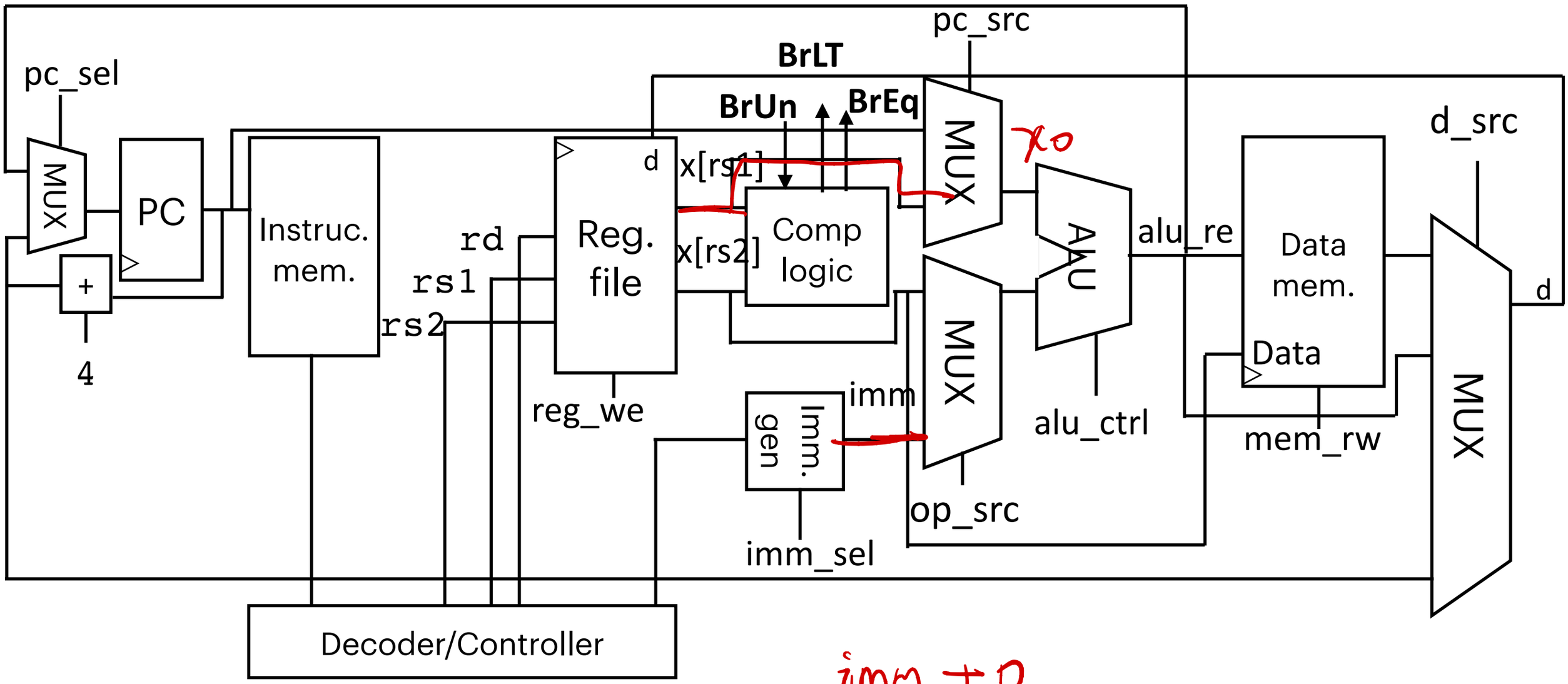
```
1. PC = PC + imm
2. rd = PC + 4
```



	addi	add	sw	lw	beq	jalr	jal
reg_we	1	1	0	1	0	1	
mem_rw	R	R	W	R	R	R	
alu_ctrl	add	add	add	add	add	add	
imm_sel	I	*	S	I	B	I	
d_src	alu	alu	*	mem	*	pc	
op_src	imm	reg	imm	imm	imm	imm	
pc_sel	+4	+4	+4	+4	Comp.	alu_re	
pc_src	reg	reg	reg	reg	pc	reg	
BrUn	*	*	*	*	w/w/o u	*	

Add U-type: `LUI`

imm[31:12]				rd	opcode	U-type
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	J-type

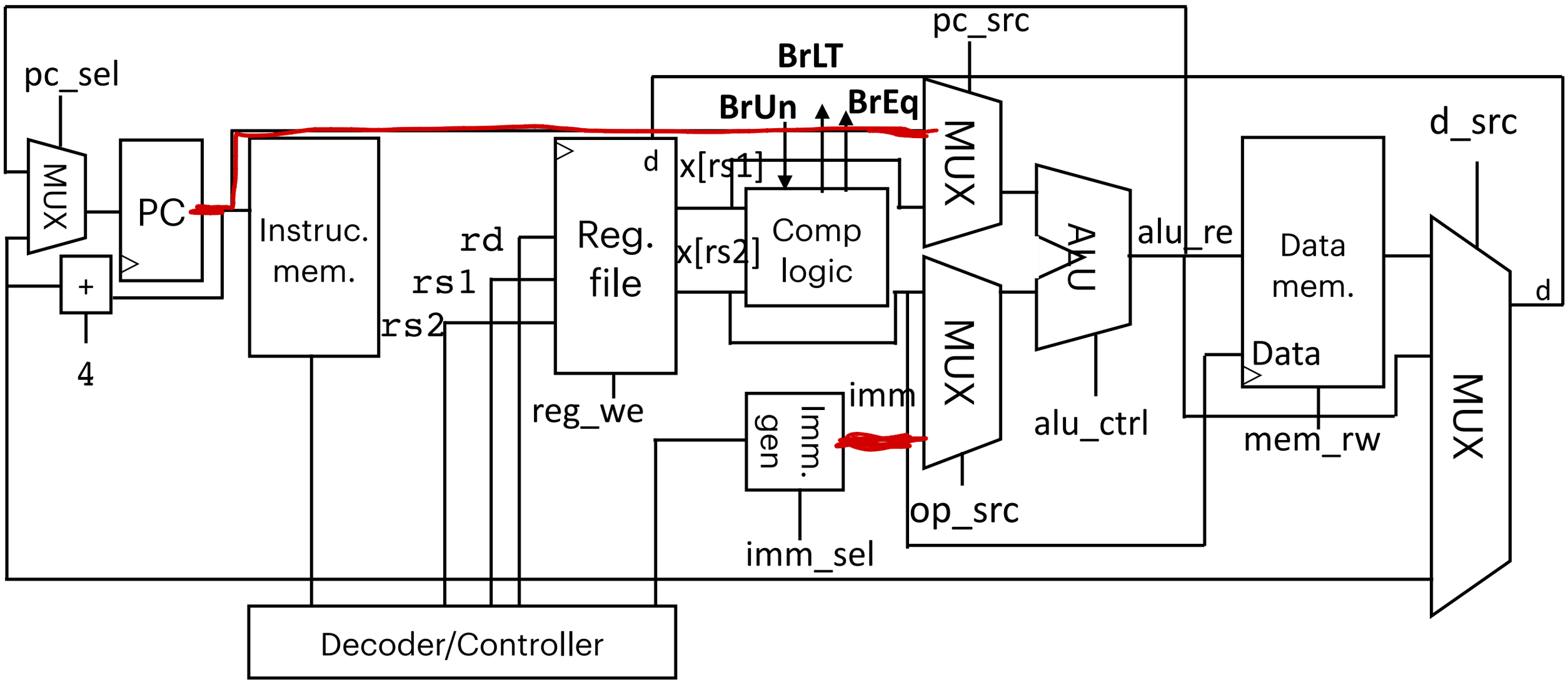


Add U-type: AUIPC

PC=PC+ UI

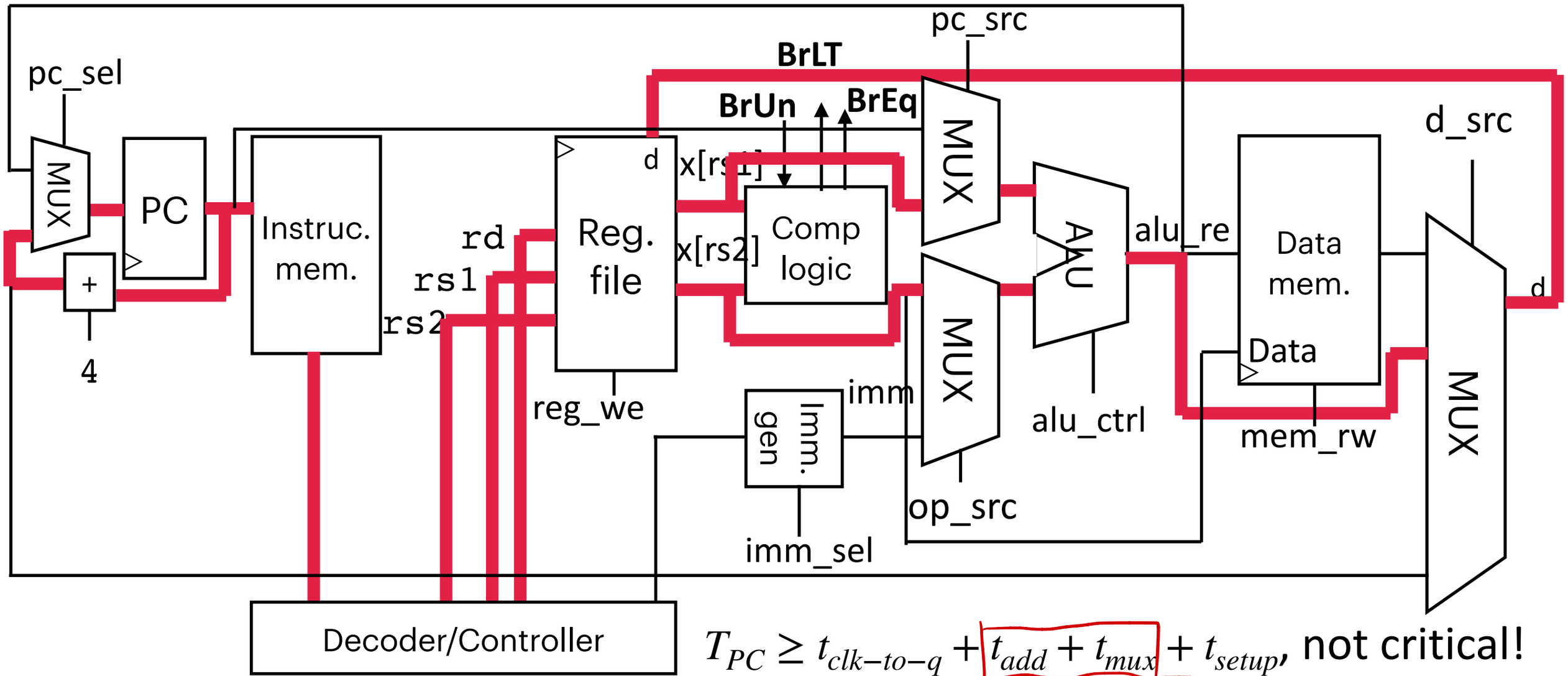
imm[31:12]	rd	opcode	U-type

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	J-type



Timing Analysis

- What is the minimum clock cycle/max frequency of the CPU?

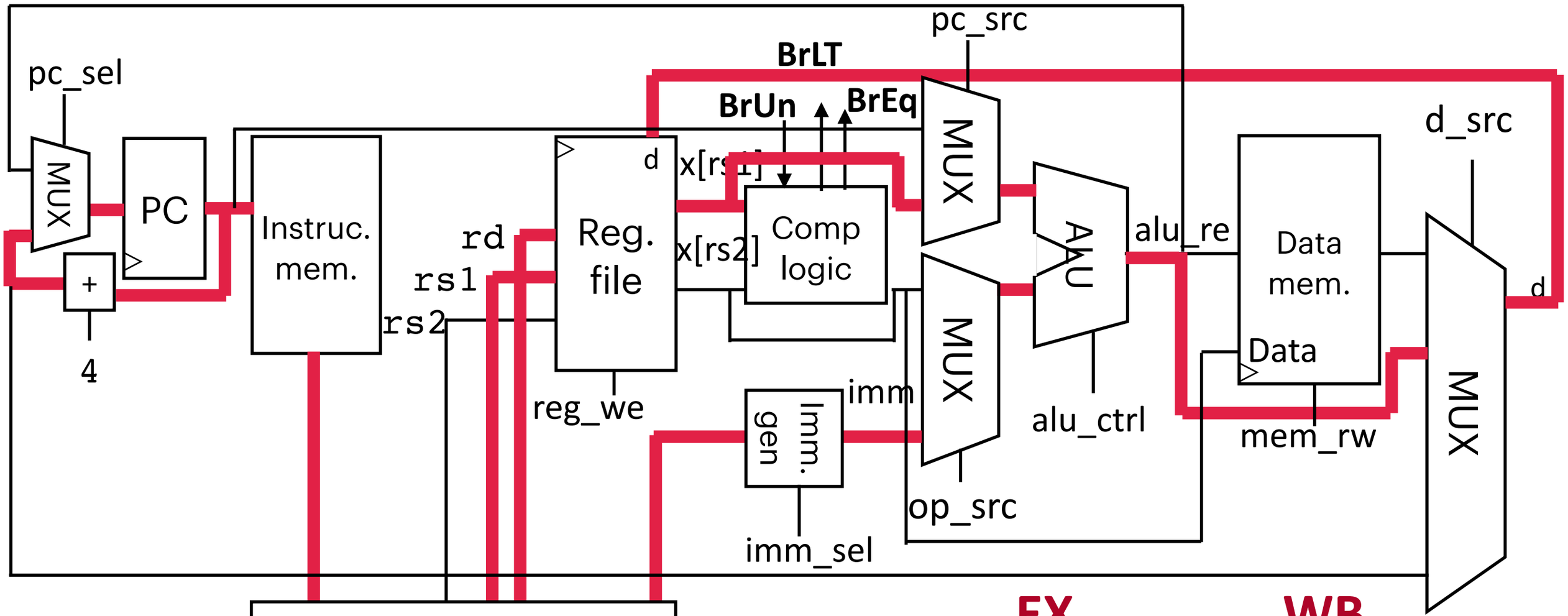


$T_{PC} \geq t_{clk-to-q} + t_{add} + t_{mux} + t_{setup}$, not critical!

R-type: $T \geq t_{clk-to-q} + t_{imem} + t_{dec} + t_{reg} + t_{mux} + t_{alu} + t_{mux} + t_{setup}$

Timing Analysis

- What is the minimum clock cycle/max frequency of the CPU?



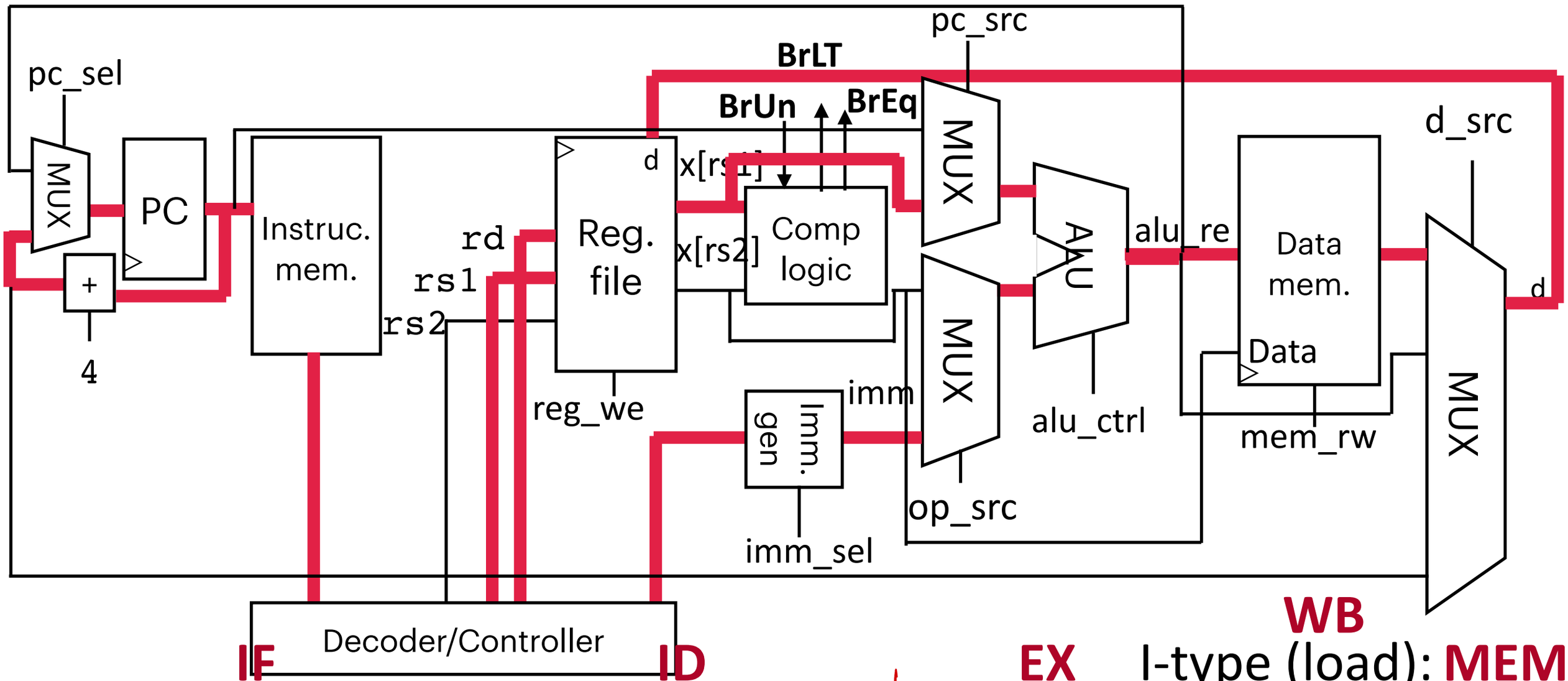
decoder

I-type (arith. & logic):

$$T \geq t_{clk-to-q} + t_{imem} + t_{dec} + \max(t_{reg}, t_{imm}) + t_{mux} + t_{alu} + t_{mux} + t_{setup}$$

Timing Analysis

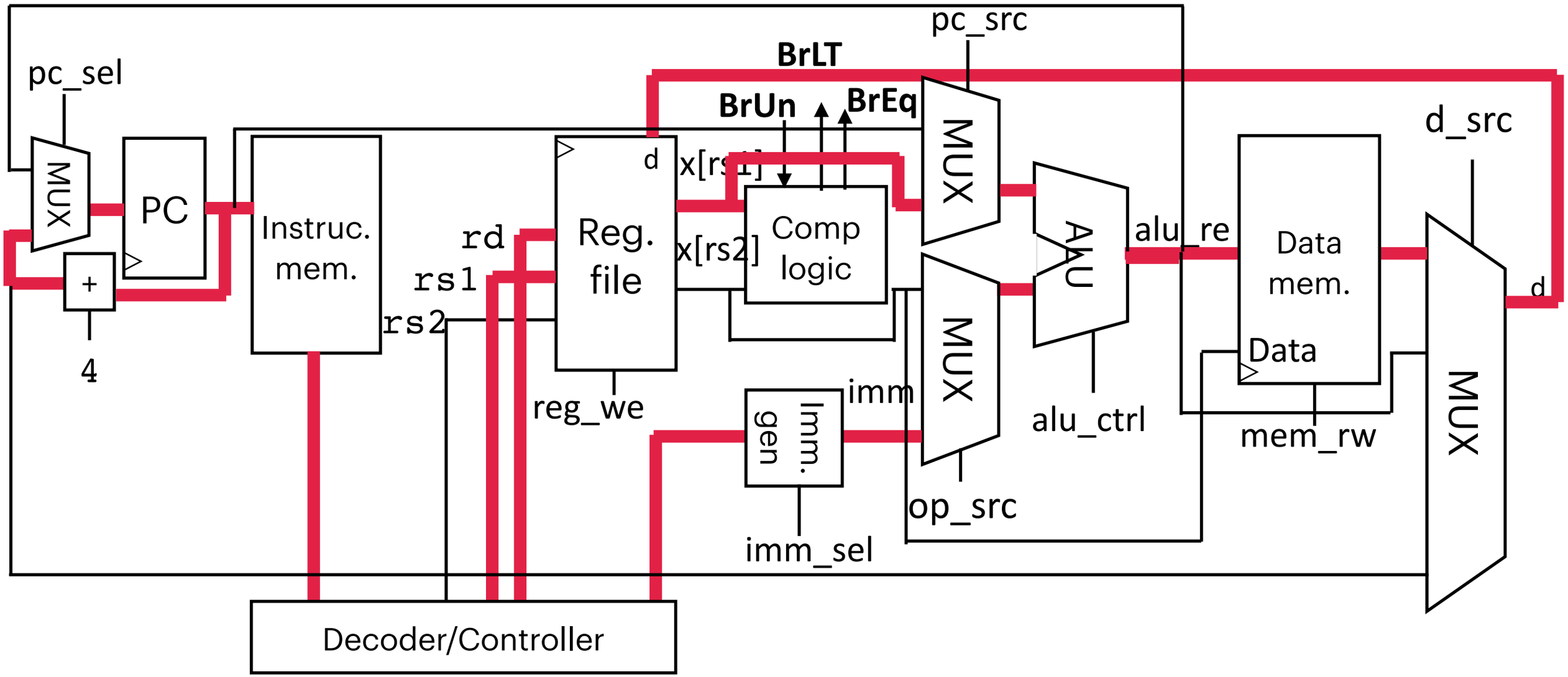
- What is the minimum clock cycle/max frequency of the CPU?



$$T \geq t_{clk-to-q} + t_{imem} + t_{dec} + \max(t_{reg}, t_{imm}) + t_{mux} + t_{alu} + t_{mux} + t_{setup} + t_{dmem}$$

Timing Analysis

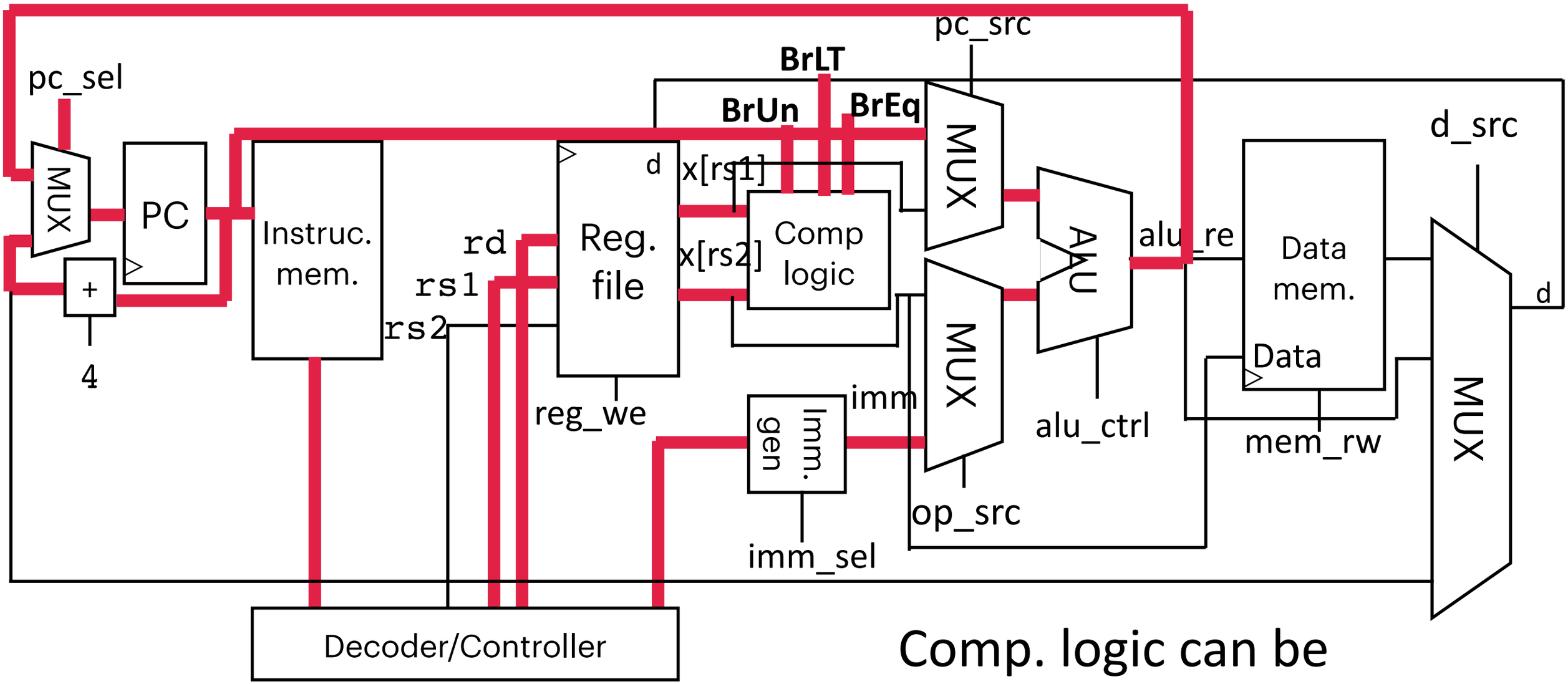
- What is the minimum clock cycle/max frequency of the CPU?



I-type (load): $T \geq t_{IF} + t_{ID} + t_{EXE} + t_{MEM} + t_{WB}$

Timing Analysis

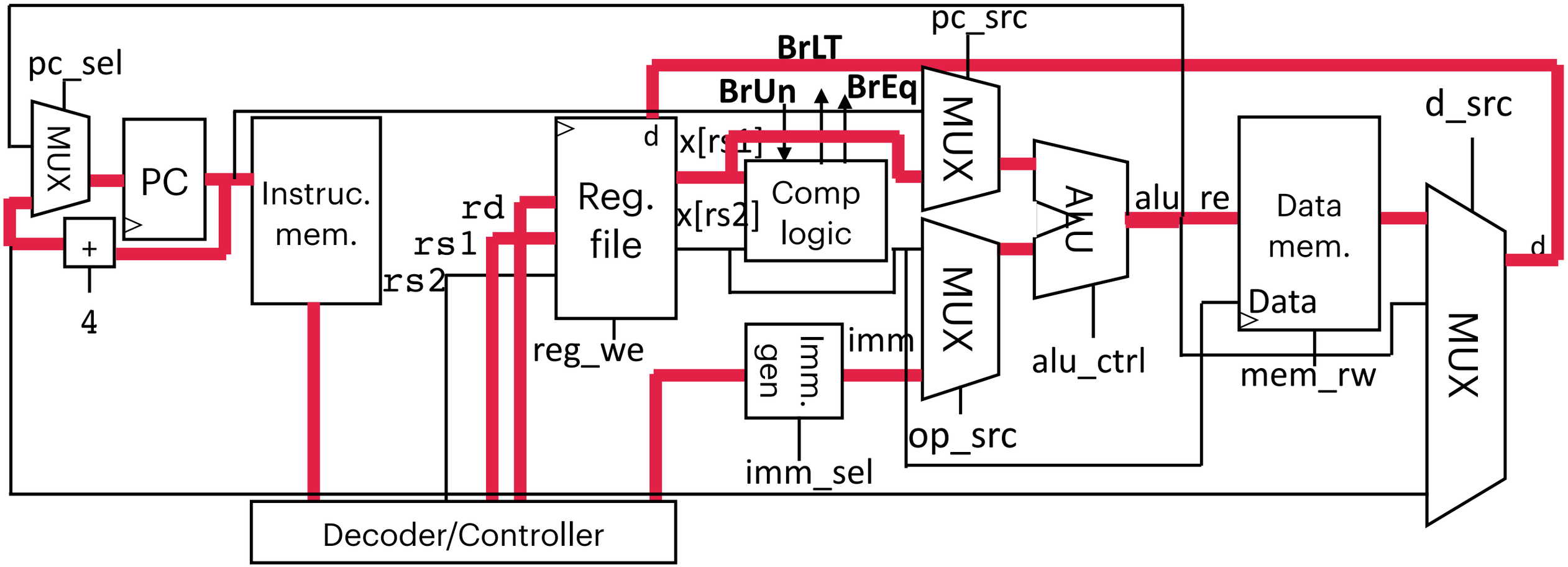
- What is the minimum clock cycle/max frequency of the CPU?

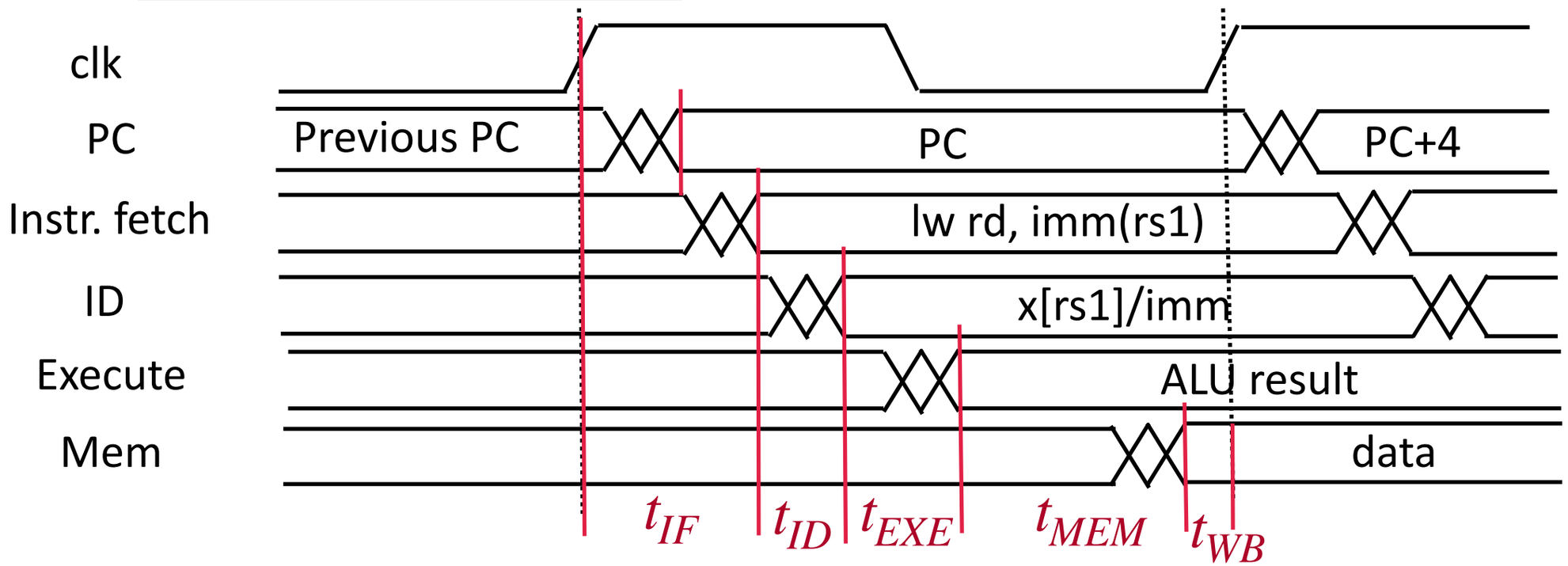


B-type: $T \geq t_{IF} + t_{ID} + t_{EXE}$

Comp. logic can be considered as part of *ID*

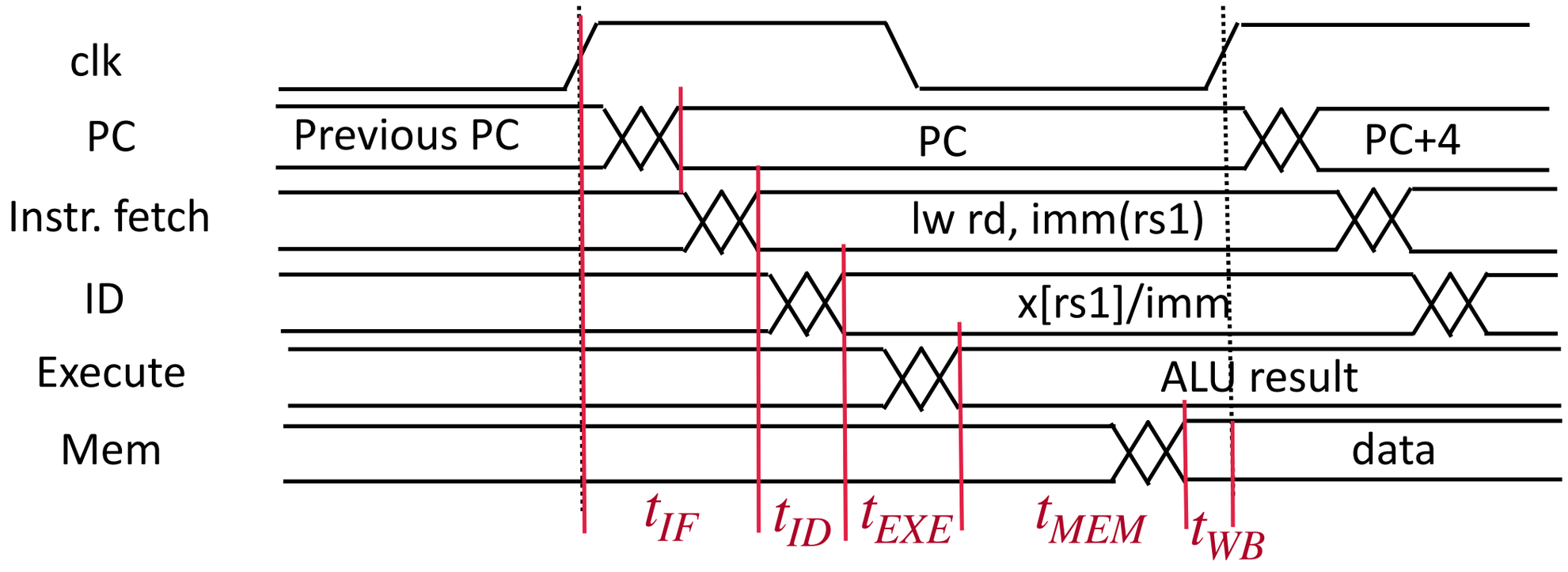
Timing Diagram (Consider delays)



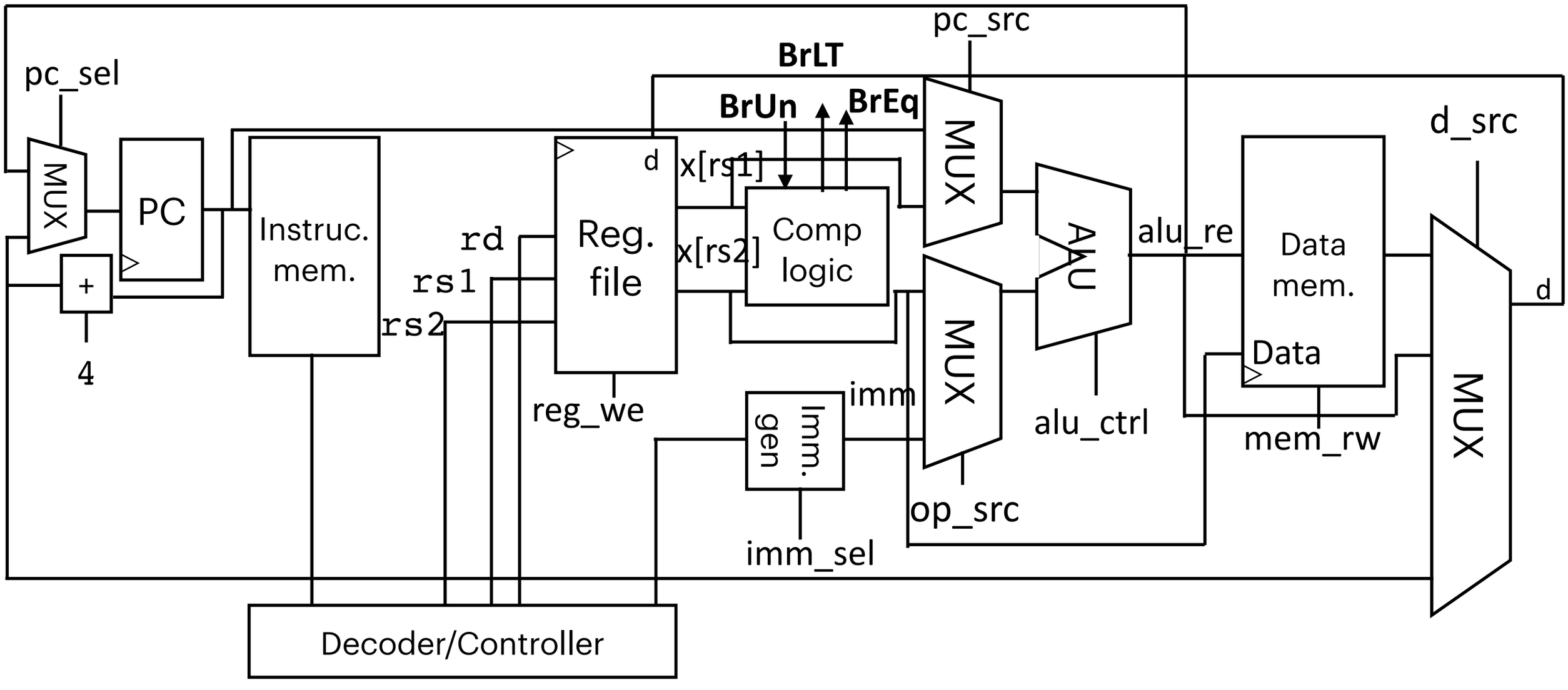


Timing Diagram (Consider delays)

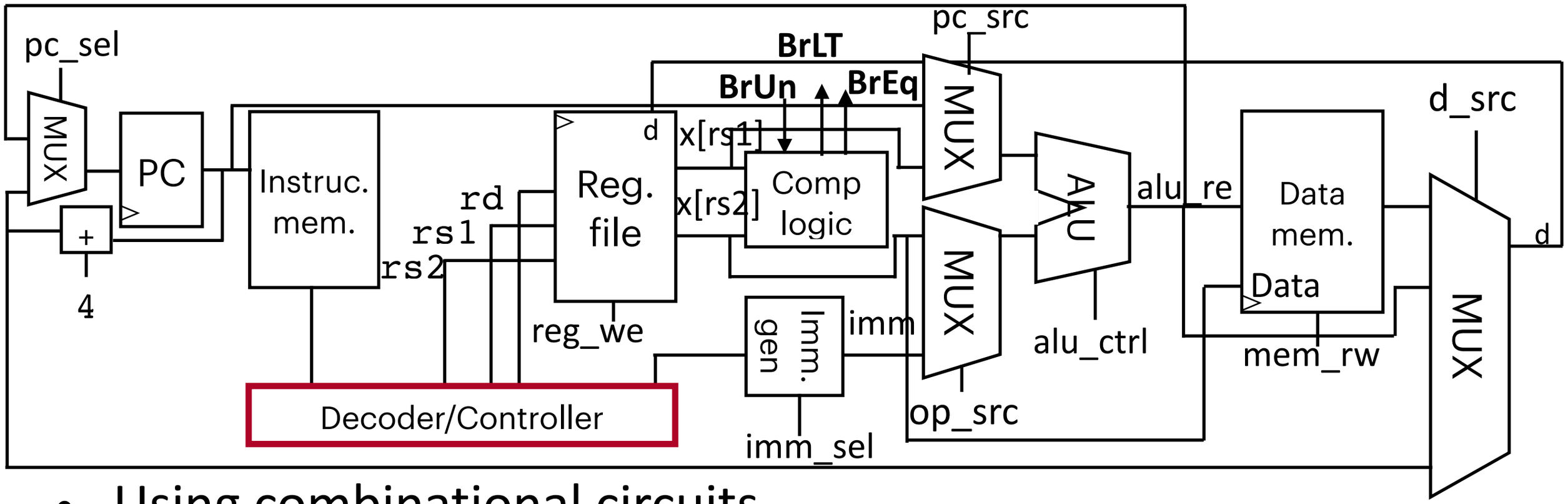
Instru.	IF = 300 ps	ID = 100 ps	EX = 200 ps	MEM = 300 ps	WB = 100 ps	Total
`add`	X	X	X		X	700 ps
`beq`	X	X	X			600 ps
`jal`	X	X	X		X	700 ps
`lw`	X	X	X	X	X	1000 ps
`sw`	X	X	X	X		900 ps



Up to Now


pc_src
BrLT
pc_sel
BrUn
BrEq
MUX
d_src
d
x[rs1]
PC
Instruc.
mem.
rd
Reg.
file
x[rs2]
Comp
logic
MUX
AAU
alu_re
Data
mem.
+
rs1
rs2
d
MUX
Data
4
reg_we
imm
Imm.
gen
alu_ctrl
mem_rw
MUX
op_src
imm_sel
Decoder/Controller

Controller: Two Implementations



- Using combinational circuits

	addi	add	sw	lw	beq	jalr	jal
reg_we	1	1	0	1	0	1	1
mem_rw	R	R	W	R	R	R	R
alu_ctrl	add	add	add	add	add	add	add
imm_sel	I	*	S	I	B	I	J
d_src	alu	alu	*	mem	*	pc	pc
op_src	imm	reg	imm	imm	imm	imm	imm
pc_sel	+4	+4	+4	+4	Comp.	alu_re	alu_re
pc_src	reg	reg	reg	reg	pc	reg	pc
BrUn	*	*	*	*	w/w/o u	*	*

下一页

Combinational Control Logic

i[30] i[14:12] i[6:2]

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
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr			rs1	001	rd	1110011	CSRRW
csr			rs1	010	rd	1110011	CSRRS
csr			rs1	011	rd	1110011	CSRRC
csr			zimm	101	rd	1110011	CSRRWI
csr			zimm	110	rd	1110011	CSRRSI
csr			zimm	111	rd	1110011	CSRRCI

Not in CS110

- Simplest example: BrUn

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

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

- How to decode/control whether BrUn is 1? BrUn = Inst [13] & Branch instruction
- Branch?

1100011

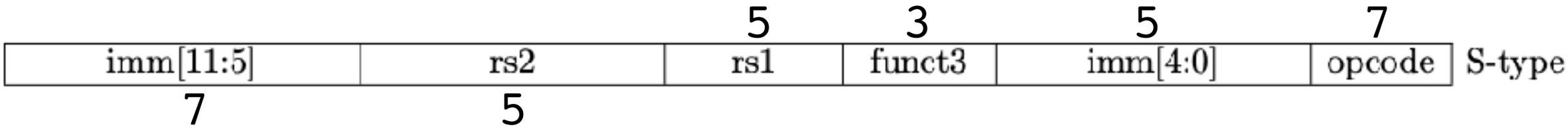
Using Types of Instructions

	addi	add	sw	lw	beq	jalr	jal
reg_we	1	1	0	1	0	1	1
mem_rw	R	R	W	R	R	R	R
alu_ctrl	add	add	add	add	add	add	add
imm_sel	I	*	S	I	B	I	J
d_src	alu	alu	*	mem	*	pc	pc
op_src	imm	reg	imm	imm	imm	imm	imm
pc_sel	+4	+4	+4	+4	Comp.	alu_re	alu_re
pc_src	reg	reg	reg	reg	pc	reg	pc
BrUn	*	*	*	*	w/w/o u	*	*

- Example:
  - mem_rw, 1 when “W”; 0 when “R”
  - Only “W” when S-type store

$$mem\_rw = SW + SH + SB$$

Recall: S-Format Store Instructions



`Assembly: sw/sh/sb rs2, imm(rs1)`

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

$$SB = \overline{i[14]}\ \overline{i[13]}\ \overline{i[12]}\ \overline{i[6]}\ i[5]\ \overline{i[4]}\ \overline{i[3]}\ \overline{i[2]}$$

$$SH = \overline{i[14]}\ \overline{i[13]}\ i[12]\ \overline{i[6]}\ i[5]\ \overline{i[4]}\ \overline{i[3]}\ \overline{i[2]}$$

$$SW = \overline{i[14]}\ i[13]\ \overline{i[12]}\ \overline{i[6]}\ i[5]\ \overline{i[4]}\ \overline{i[3]}\ \overline{i[2]}$$

Combinational Control Logic (Don't-care values)

	addi	add	sw	lw	beq	jalr	jal
reg_we	1	1	0	1	0	1	1
mem_rw	R	R	W	R	R	R	R
alu_ctrl	add	add	add	add	add	add	add
imm_sel	I	*	S	I	B	I	J
d_src	alu	alu	*	mem	*	pc	pc
op_src	imm	reg	imm	imm	imm	imm	imm
pc_sel	+4	+4	+4	+4	Comp.	alu_re	alu_re
pc_src	reg	reg	reg	reg	pc	reg	pc
BrUn	*	*	*	*	w/w/o u	*	*

Combinational Control Logic (Don't-care values)

- imm_sel as an example

0000000	rs2	rs1	000	rd	0110011	ADD
imm		rs1	000	rd	0010011	ADDI
imm		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm		rs1	000	rd	1100111	JALR

- Assume (should be 3-bit, because we still have U-type imm.)

	addi	add	sw	lw	beq	jalr	jal
imm_sel	I	*	S	I	B	I	J
	00	—	01	00	10	00	11

Combinational Control Logic (Don't-care values)

- imm_sel as an example

func3	opcode[6:2]		imm_sel	
000	01100	ADD	*	→ Can be 00, 01, 10, 11
000	00100	ADDI	00	
010	00000	LW	00	
010	01000	SW	01	
000	11000	BEQ	10	
*	11011	JAL	11	
000	11001	JALR	00	

Funct3 (instru[14:12]) instru[6:2]

$$imm\_sel[1] = i[6]\ i[5]\ \overline{i[4]}\ i[3]\ i[2] + i[6]\ i[5]\ \overline{i[4]}\ \overline{i[3]}\ \overline{i[2]}\ \overline{i[14]}\ \overline{i[13]}\ \overline{i[12]}$$

$$+ \overline{i[6]}\ i[5]\ i[4]\ \overline{i[3]}\ \overline{i[2]}\ \overline{i[14]}\ \overline{i[13]}\ \overline{i[12]}$$

ROM-based Implementation

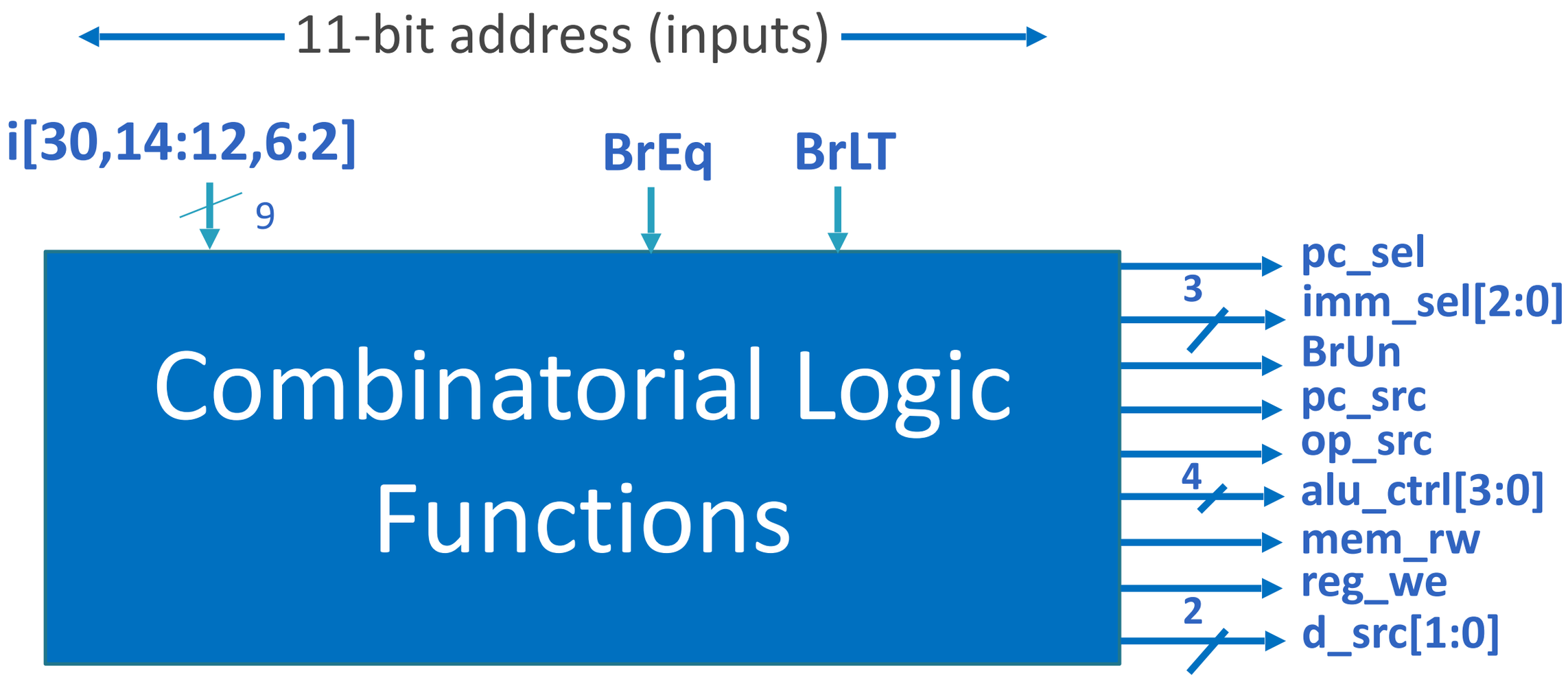
imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	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
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr			rs1	001	rd	1110011	CSRRW
csr			rs1	010	rd	1110011	CSRRS
csr			rs1	011	rd	1110011	CSRRC
csr			zimm	101	rd	1110011	CSRRWI
csr			zimm	110	rd	1110011	CSRRSI
csr			zimm	111	rd	1110011	CSRRCI

Not in CS110

- Read-only memory (ROM)
  - Similar to RAM, given address, the contents are accessed
  - Use 11 bits as address
  - Can be reprogrammed by replacing the stored value (add instructions)

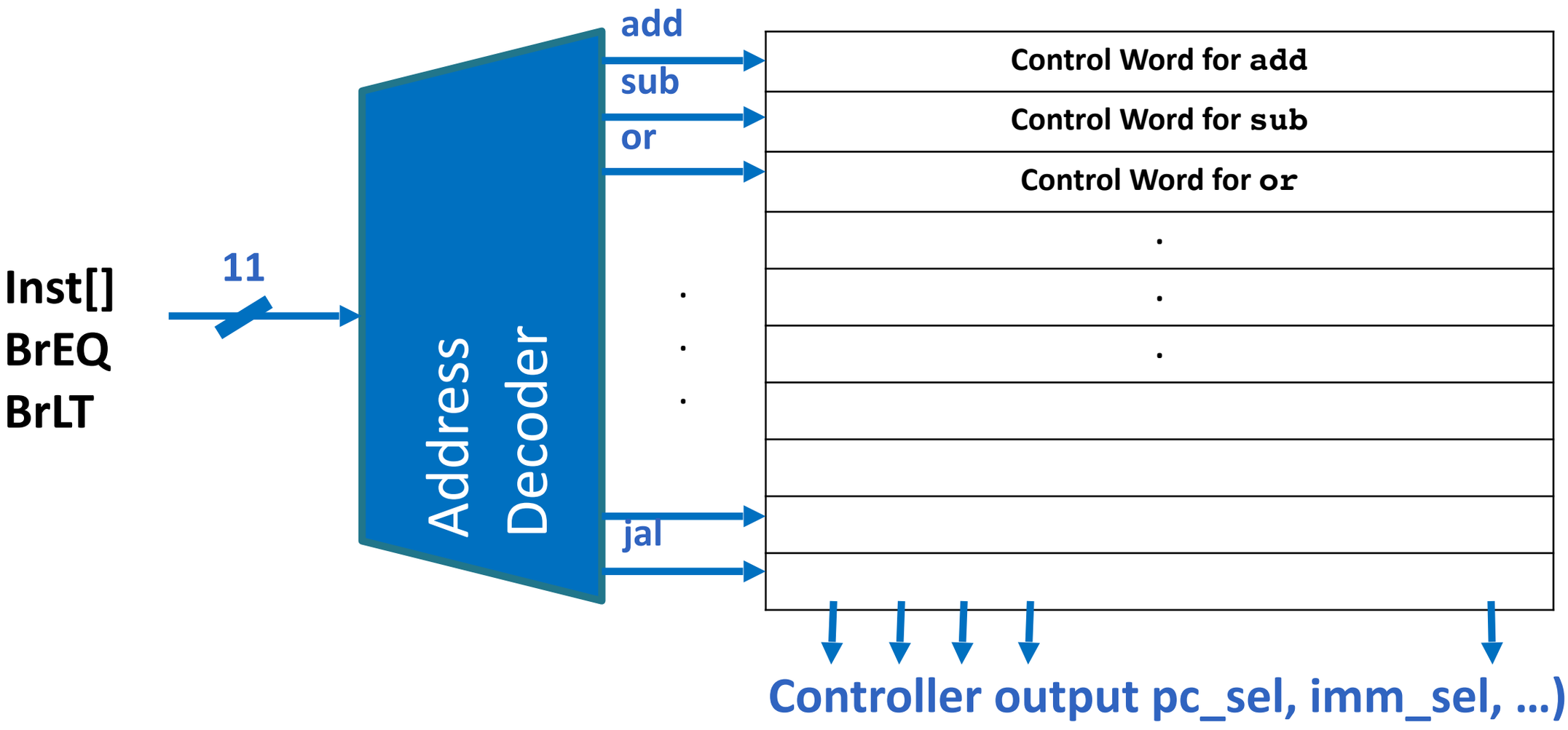
Control Block Design



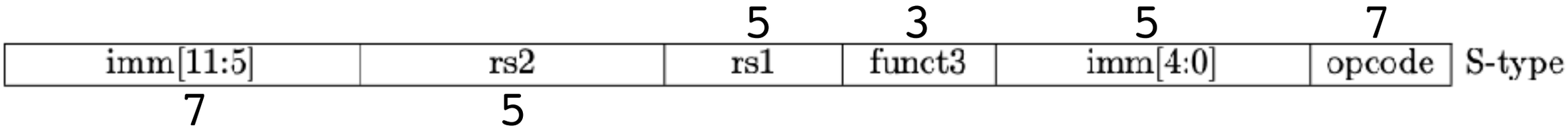
A symbol.

15 data bits (outputs)

ROM Controller Implementation



Extra Consideration for Expansion



`Assembly: sw/sh/sb rs2, imm(rs1)`

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

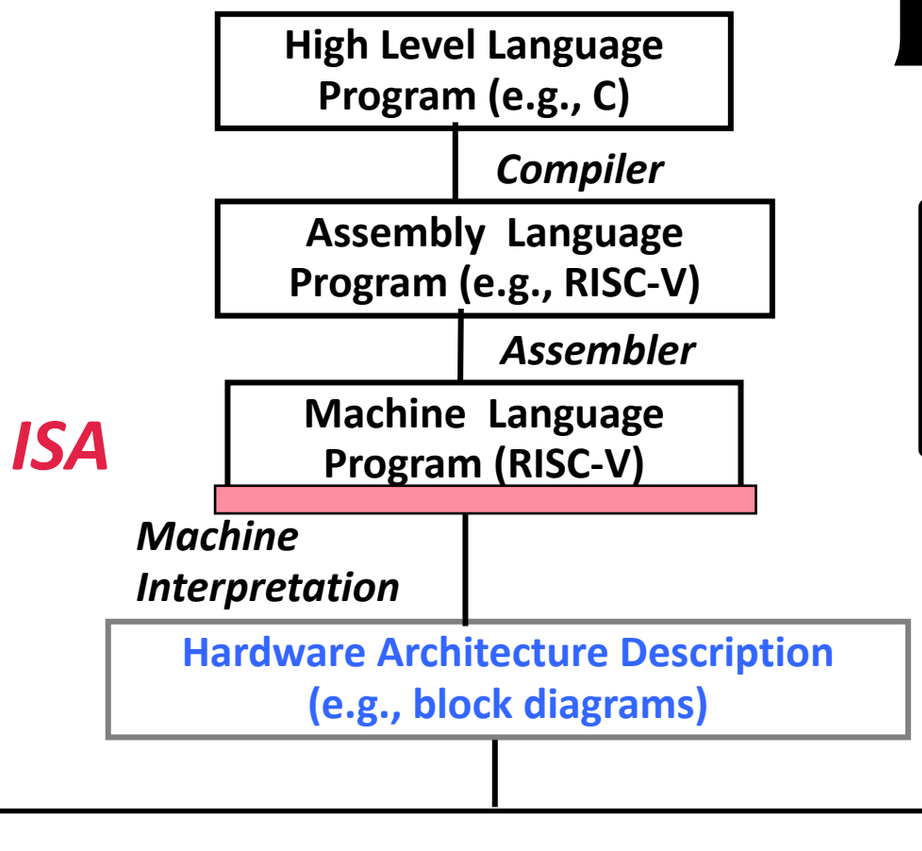
$SB = \overline{i[14]}\ \overline{i[13]}\ \overline{i[12]}\ \overline{i[6]}\ i[5]\ \overline{i[4]}\ \overline{i[3]}\ \overline{i[2]}\ i[1]\ i[0]$

$SH = \overline{i[14]}\ \overline{i[13]}\ i[12]\ \overline{i[6]}\ i[5]\ \overline{i[4]}\ \overline{i[3]}\ \overline{i[2]}\ i[1]\ i[0]$

$SW = \overline{i[14]}\ i[13]\ \overline{i[12]}\ \overline{i[6]}\ i[5]\ \overline{i[4]}\ \overline{i[3]}\ \overline{i[2]}\ i[1]\ i[0]$

`RV32I`

Finally



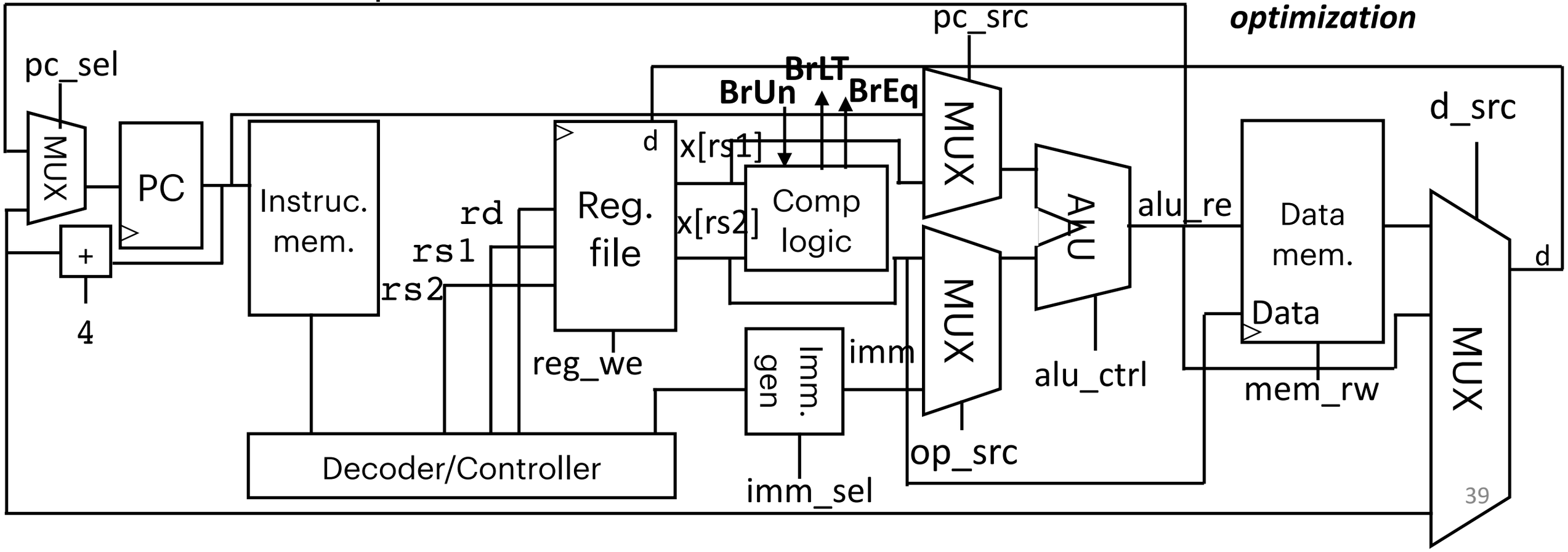
Architecture Implementation

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

Anything can be represented
as a *number*,
i.e., data or instructions

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

Single-core simple CPU w/o any optimization

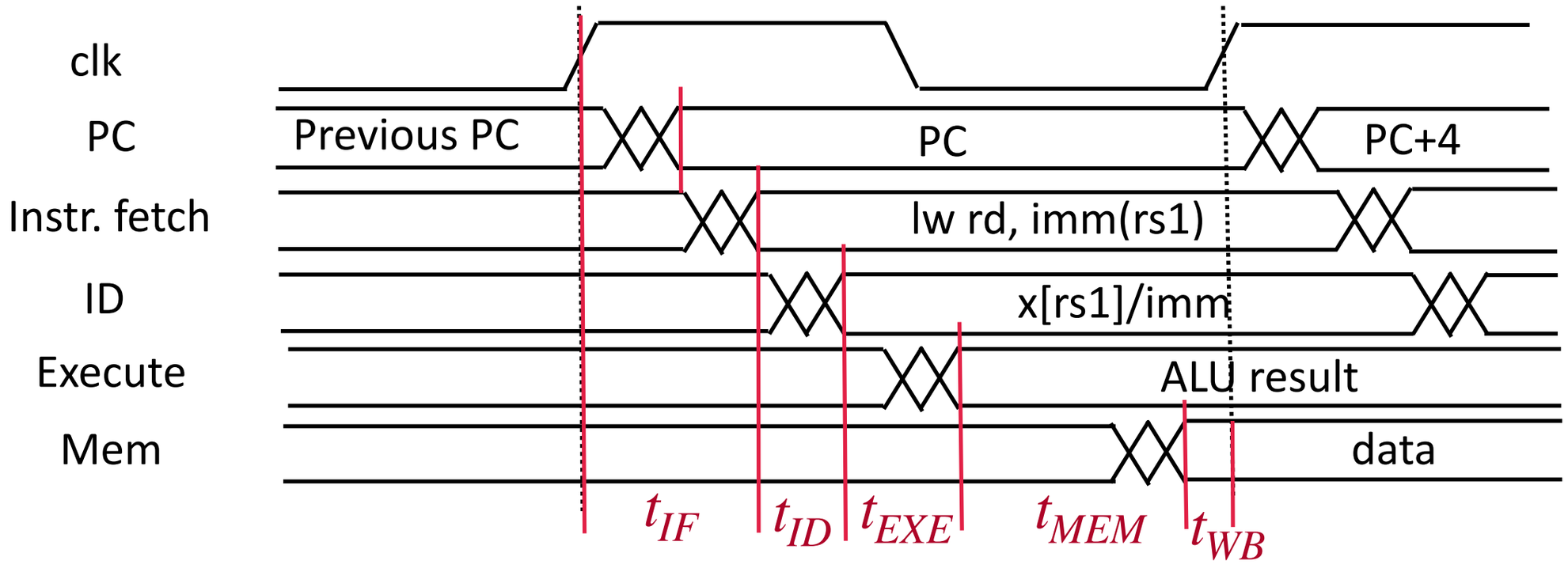


Summary

- We have built a processor!
  - Capable of executing all RISC-V instructions in one cycle each
  - Not all units (hardware) used by all instructions
  - Critical path changes
- 5 Phases of execution
  - IF, ID, EX, MEM, WB
  - Not all instructions are active in all phases
- Controller specifies how to execute instructions
  - Implemented as ROM or logic

Bottleneck

Instru.	IF = 300 ps	ID = 100 ps	EX = 200 ps	MEM = 300 ps	WB = 100 ps	Total
`add`	X	X	X		X	700 ps
`beq`	X	X	X			600 ps
`jal`	X	X	X		X	700 ps
`lw`	X	X	X	X	X	1000 ps
`sw`	X	X	X	X		900 ps



Great Ideas in Computer Architecture

- Abstraction (Layers of Representation / Interpretation)
- Moore's Law
- Principle of Locality/Memory Hierarchy
- Parallelism
- Performance Measurement and Improvement
- Dependability via Redundancy

“Iron Law” of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$

CPI

$$\frac{\text{Instructions}}{\text{Program}}$$

RISC vs. CISC

- ISA-relevant
- Compiler
- What task, complexity
- Etc.

“Iron Law” of Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$

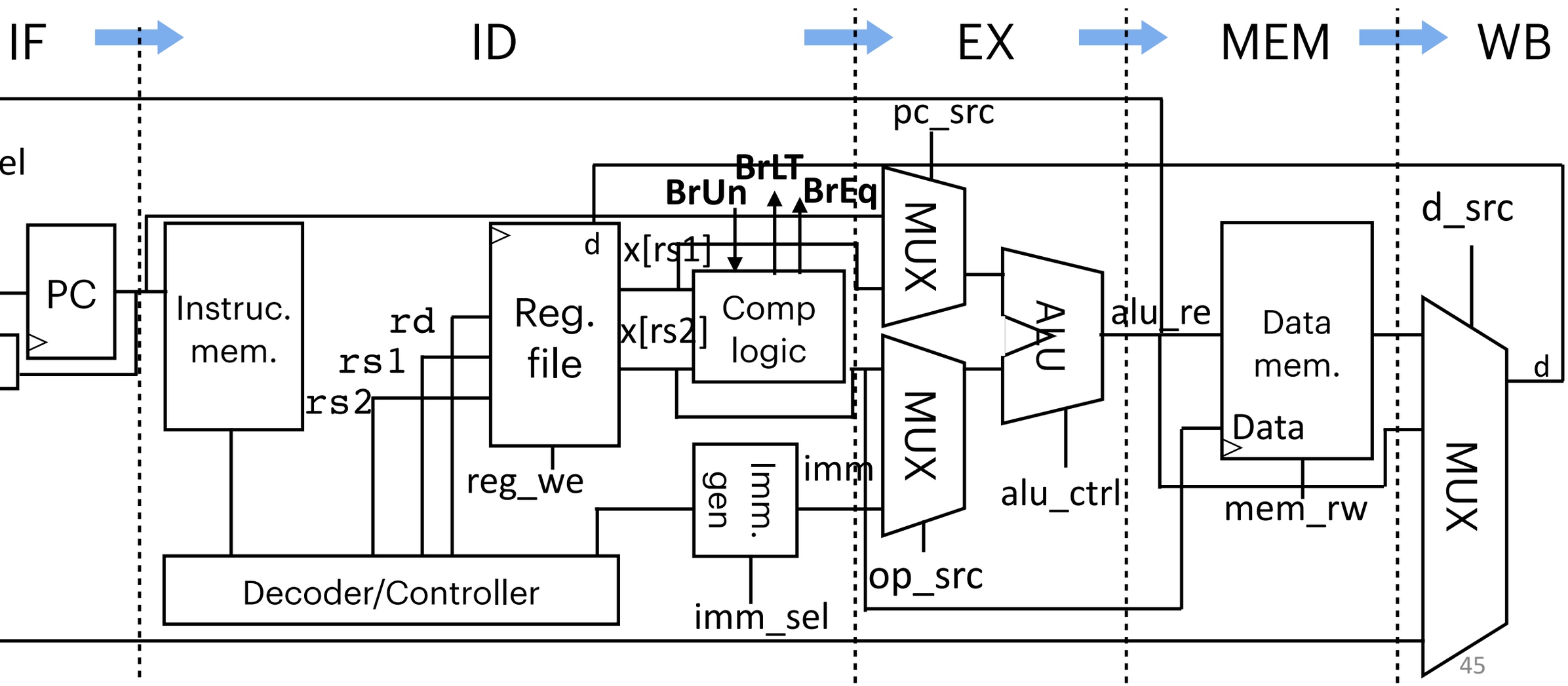
$$\frac{\text{Cycles}}{\text{Instruction}}$$

- Microarchitecture implementation or circuit design
- ISA

Timing Diagram (Consider delays)

Instru.	IF = 300 ps	ID = 100 ps	EX = 200 ps	MEM = 300 ps	WB = 100 ps	Total
`lw`	X	X	X	X	X	1000 ps

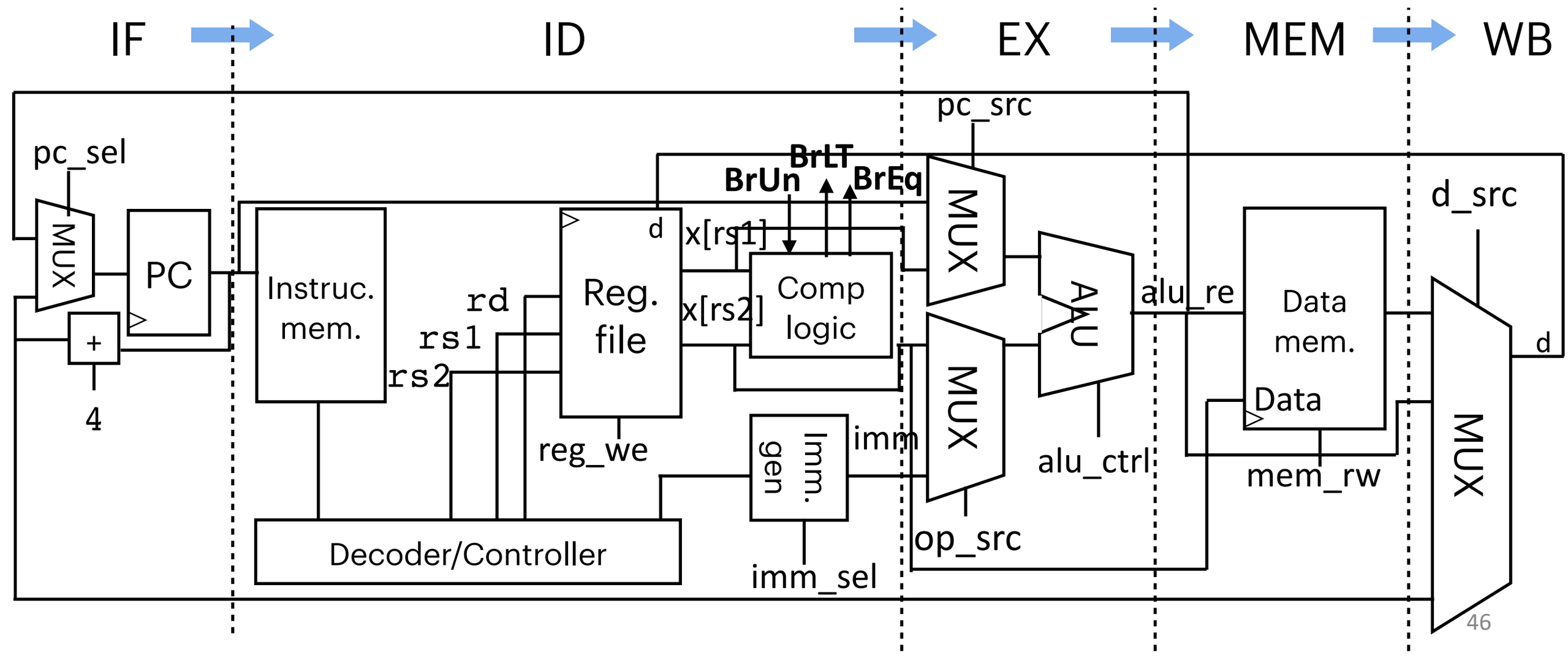
$$\frac{\text{Cycles}}{\text{Instruction}}\ \text{(CPI)} \qquad \frac{\text{Time}}{\text{Cycle}}\ \text{(Critical path; technology; TPD etc.)}$$



Timing Diagram (Consider delays)

Instru.	IF = 300 ps	ID = 100 ps	EX = 200 ps	MEM = 300 ps	WB = 100 ps	Total
`lw`	X	X	X	X	X	1000 ps

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$



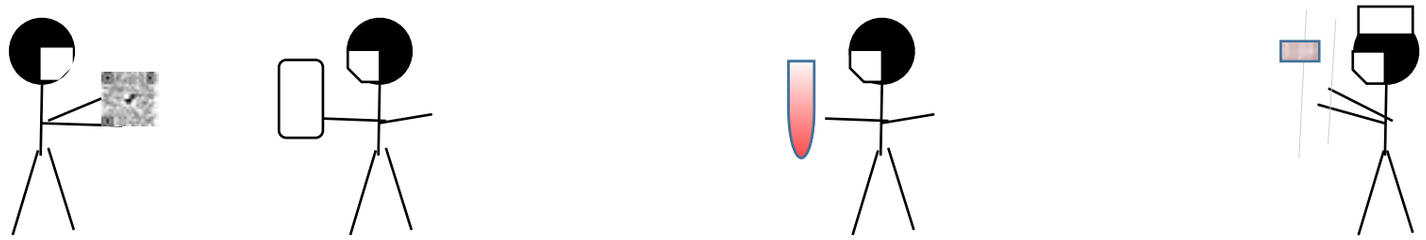
Pipeline: Improve Time/Program

- Instruction: PCR test: 1. scan QR code; 2. get the tube; 3. sample
- 5 seconds for each step:
- Single-cycle (once for all)

$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Time}{Clock\ cycle}.$$

Multi-cycle

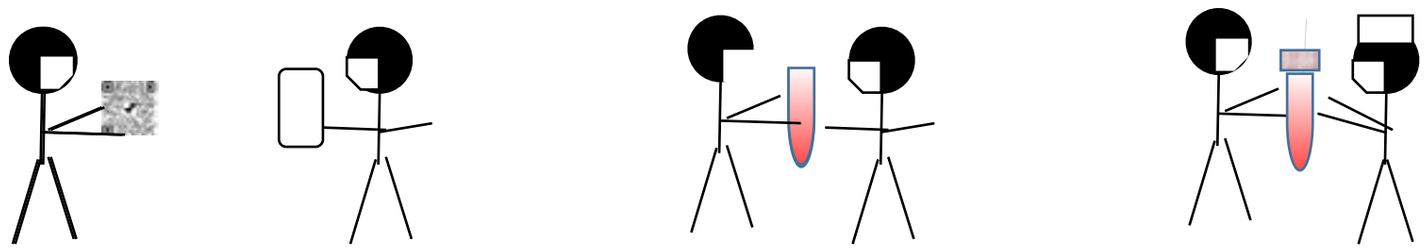
- Instruction: PCR test: 1. scan QR code; 2. get the tube; 3. sample
- 5 seconds for each step
- Multi-cycle or multi-step



$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Time}{Clock\ cycle}.$$

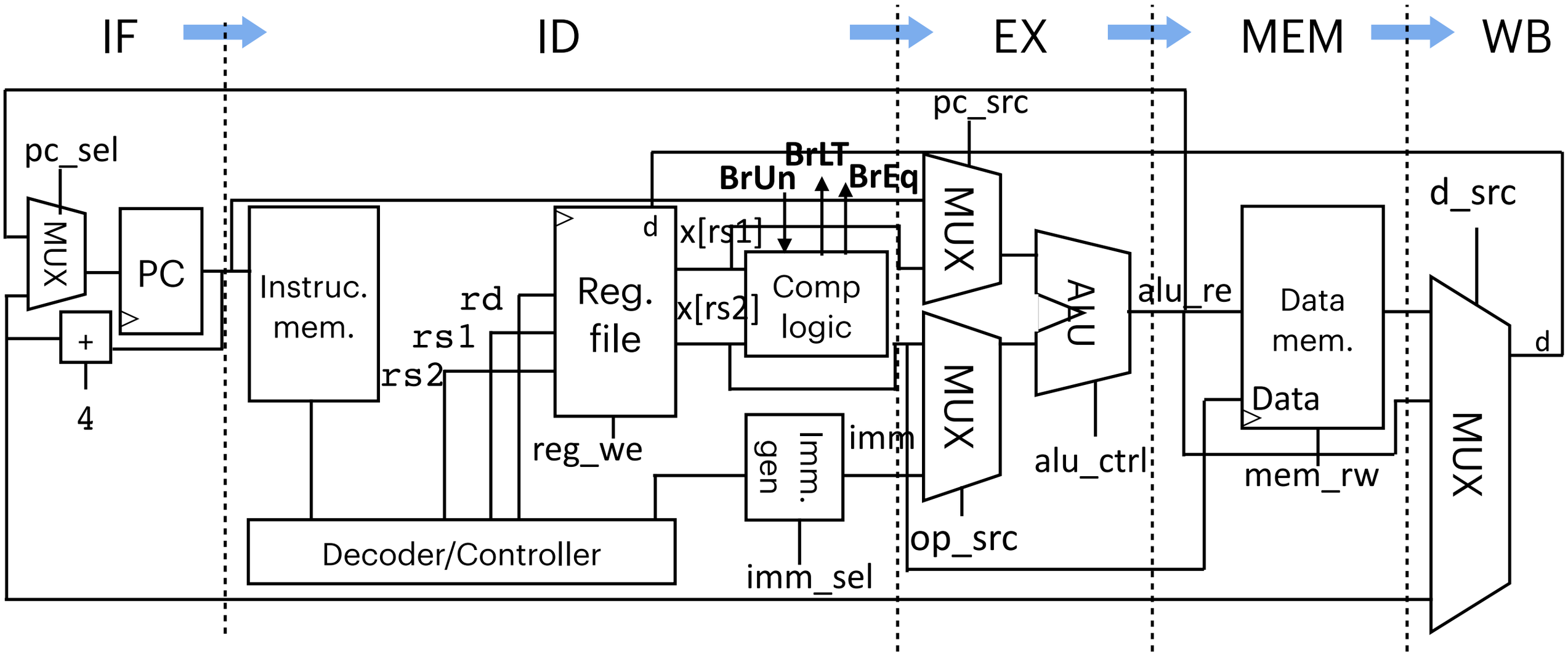
Pipeline

- PCR test in pipeline



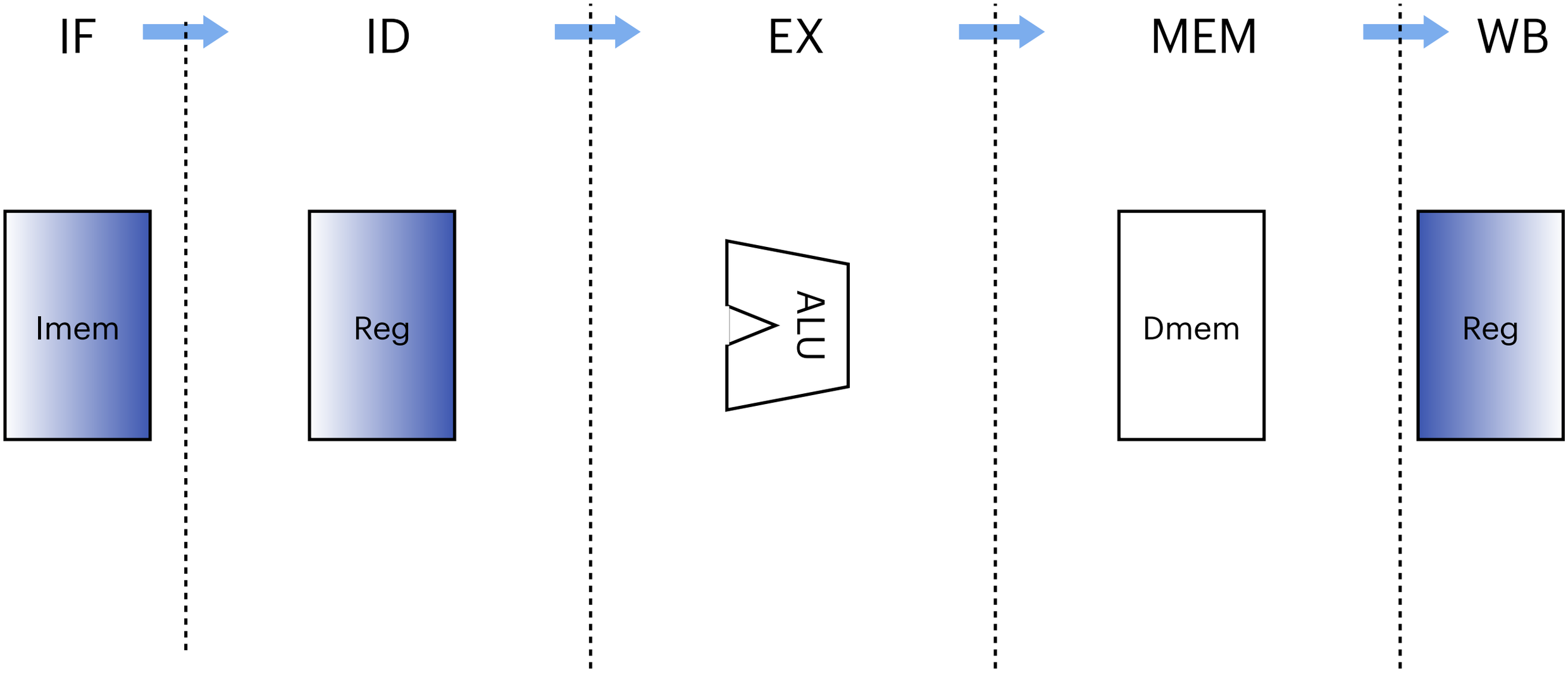
$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Time}{Clock\ cycle}.$$

Analogy in our CPU Design

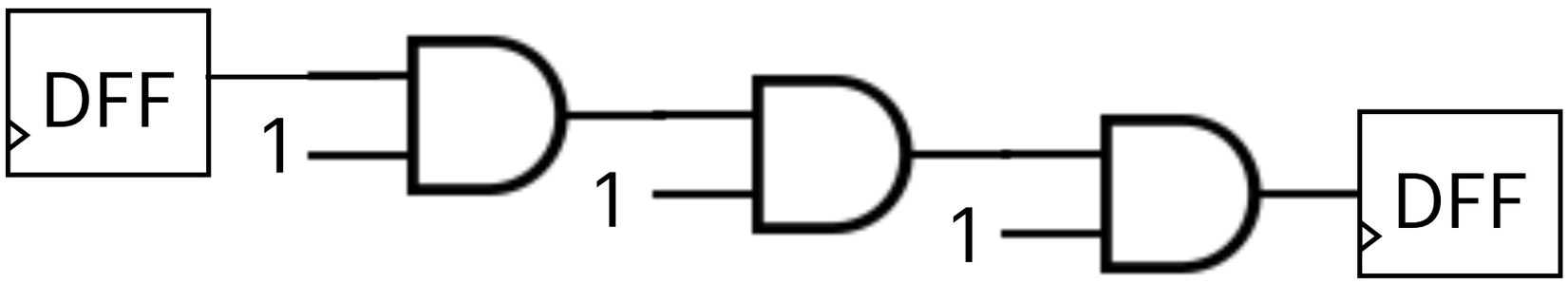


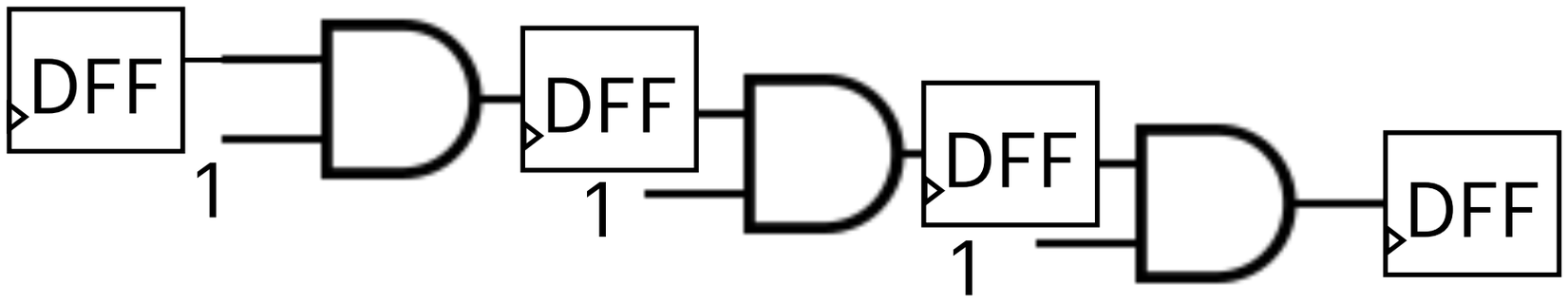
$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Time}{Clock\ cycle}.$$

Simplify the Model

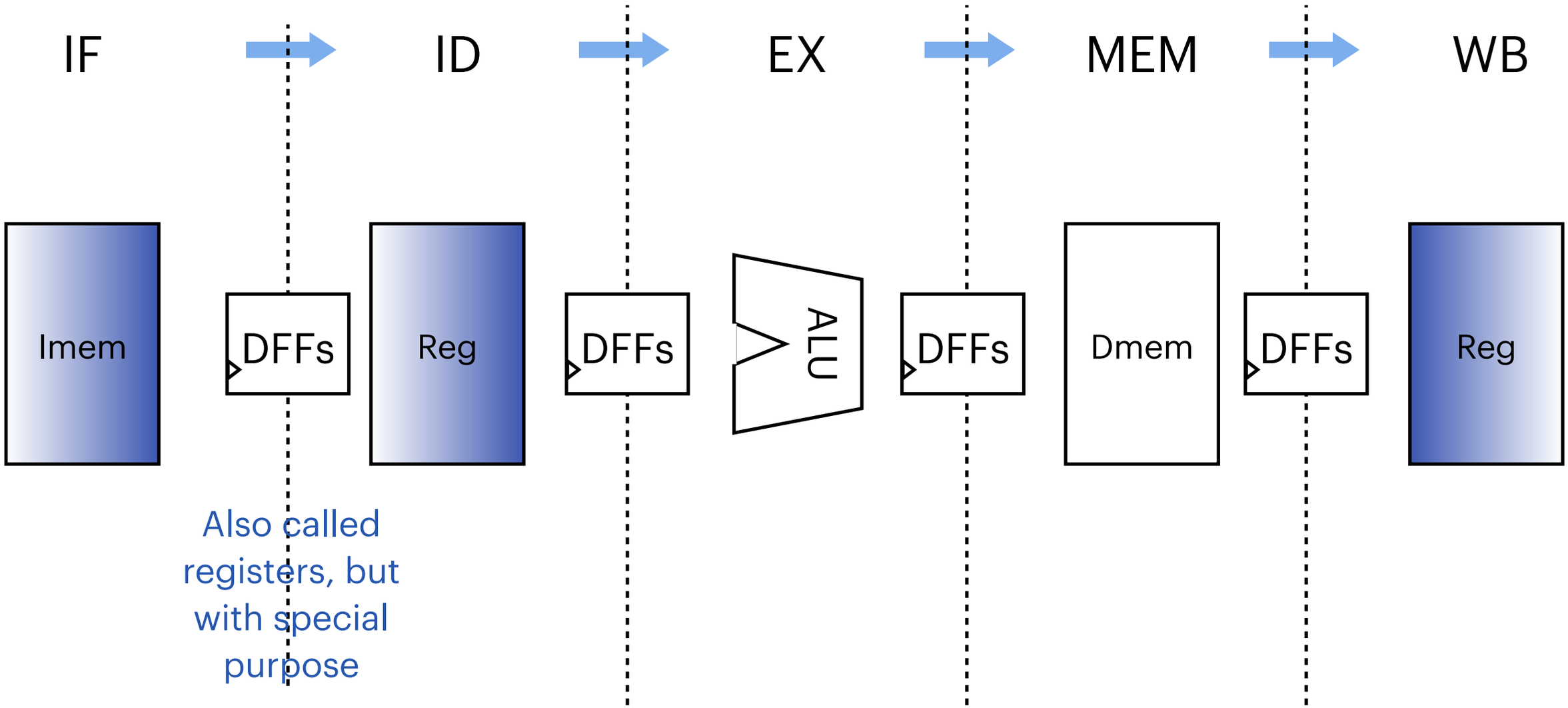


Recall DFFs





Simplify the Model



$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instruction}} \cdot \frac{\text{Time}}{\text{Cycle}}$$