Course Info

- No Lab next week, prepare your mid-term exam!
- Project 1.2 deadline March 31th. Start early!!!
- HW3 ddl March 18th! HW4 will be released next week, keep an eye on piazza.
- Next week discussion on ALU & FSM.
- Update on slip-day policy: automatic slip-day deduction if submission after ddl. No need to email TA/instructor. If you want to test your code after ddl, contact TAs directly.
- Mid-term next Tuesday 8:00 am-10:00 am, we will use teaching center Room 301/404/405. Arrive 7:45 am to check-in and find your seat. Bring your student ID card!

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

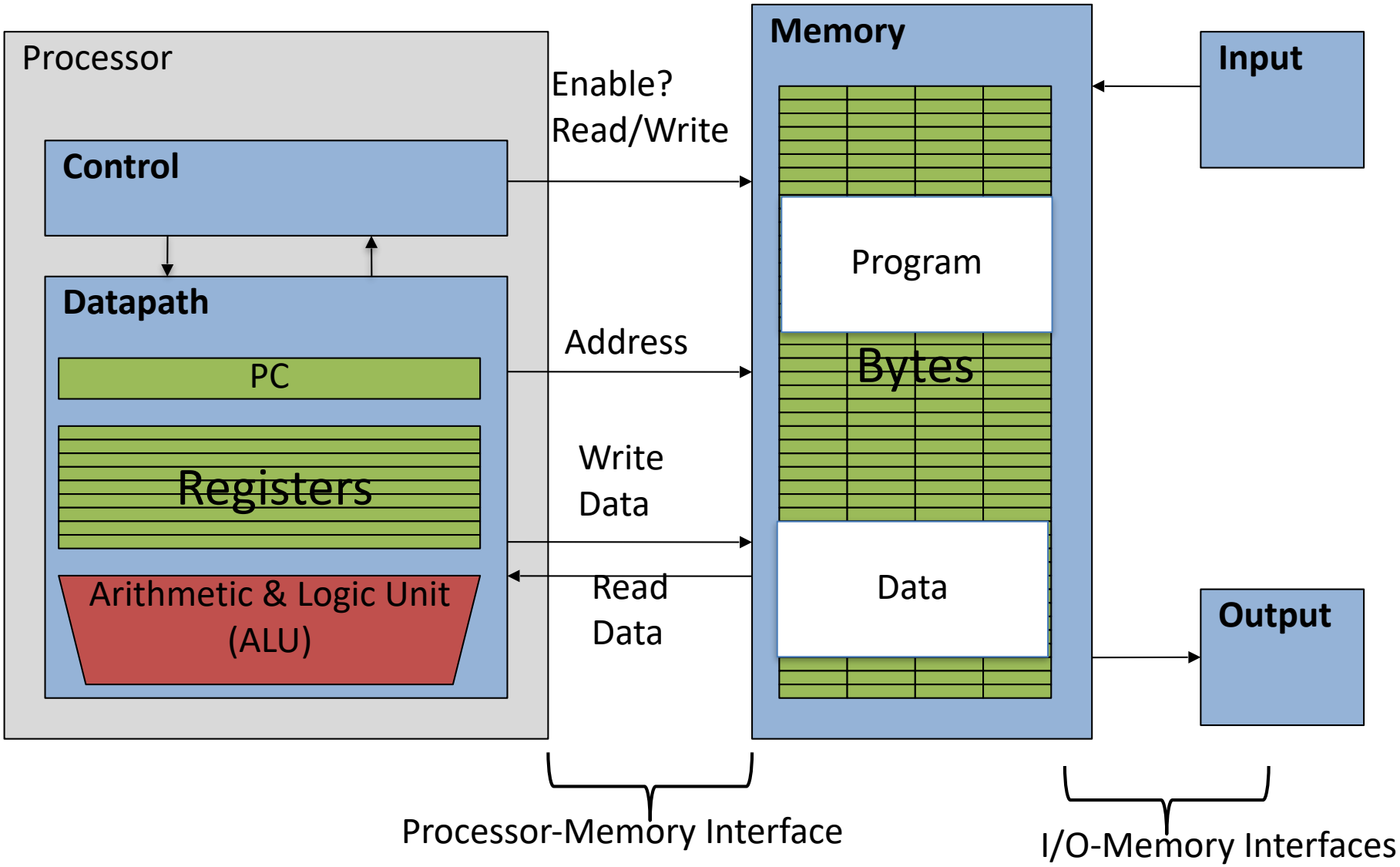
Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2023/index.html

School of Information Science and Technology (SIST)

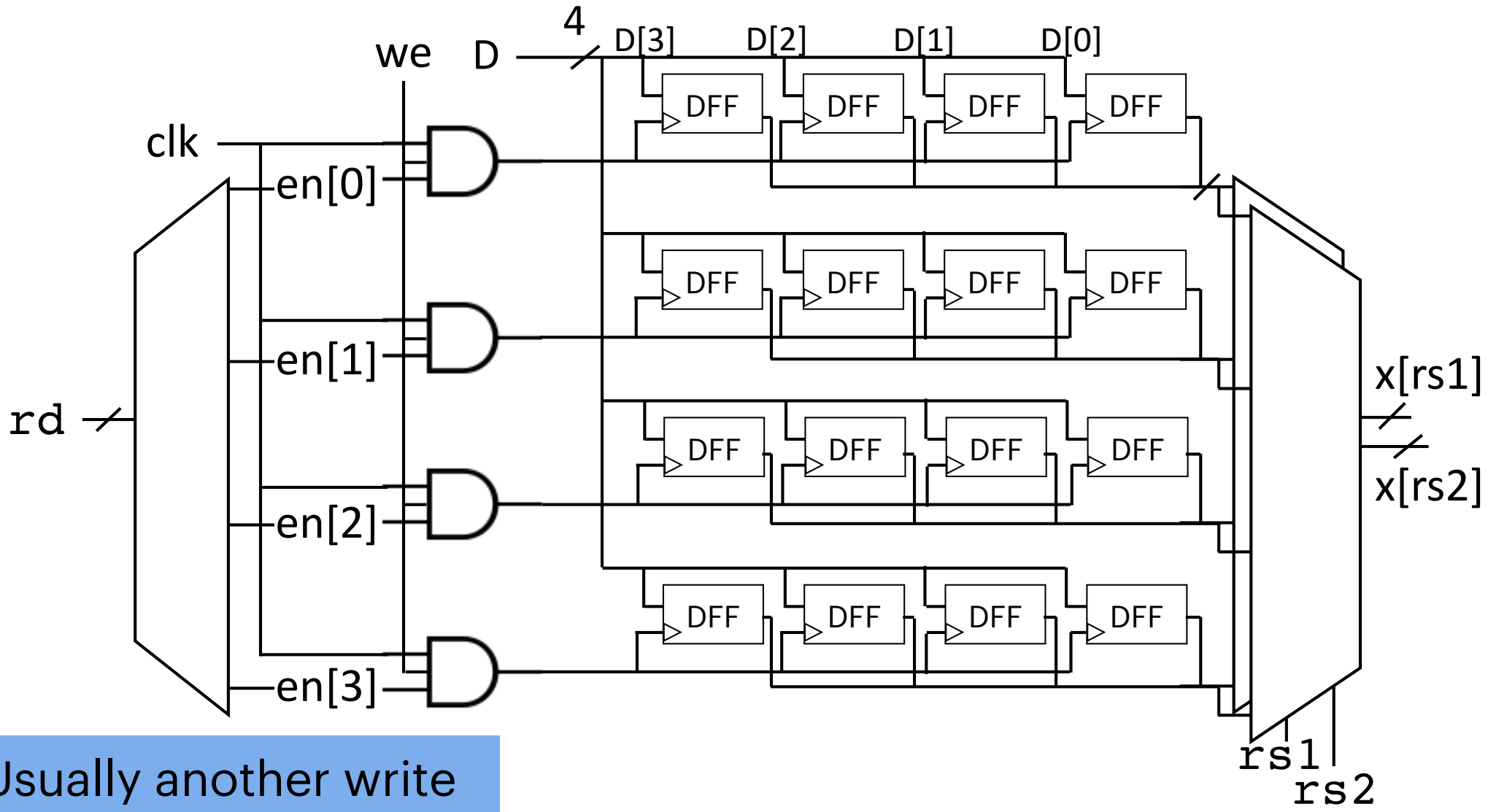
ShanghaiTech University

2023/3/5

Components of a Computer

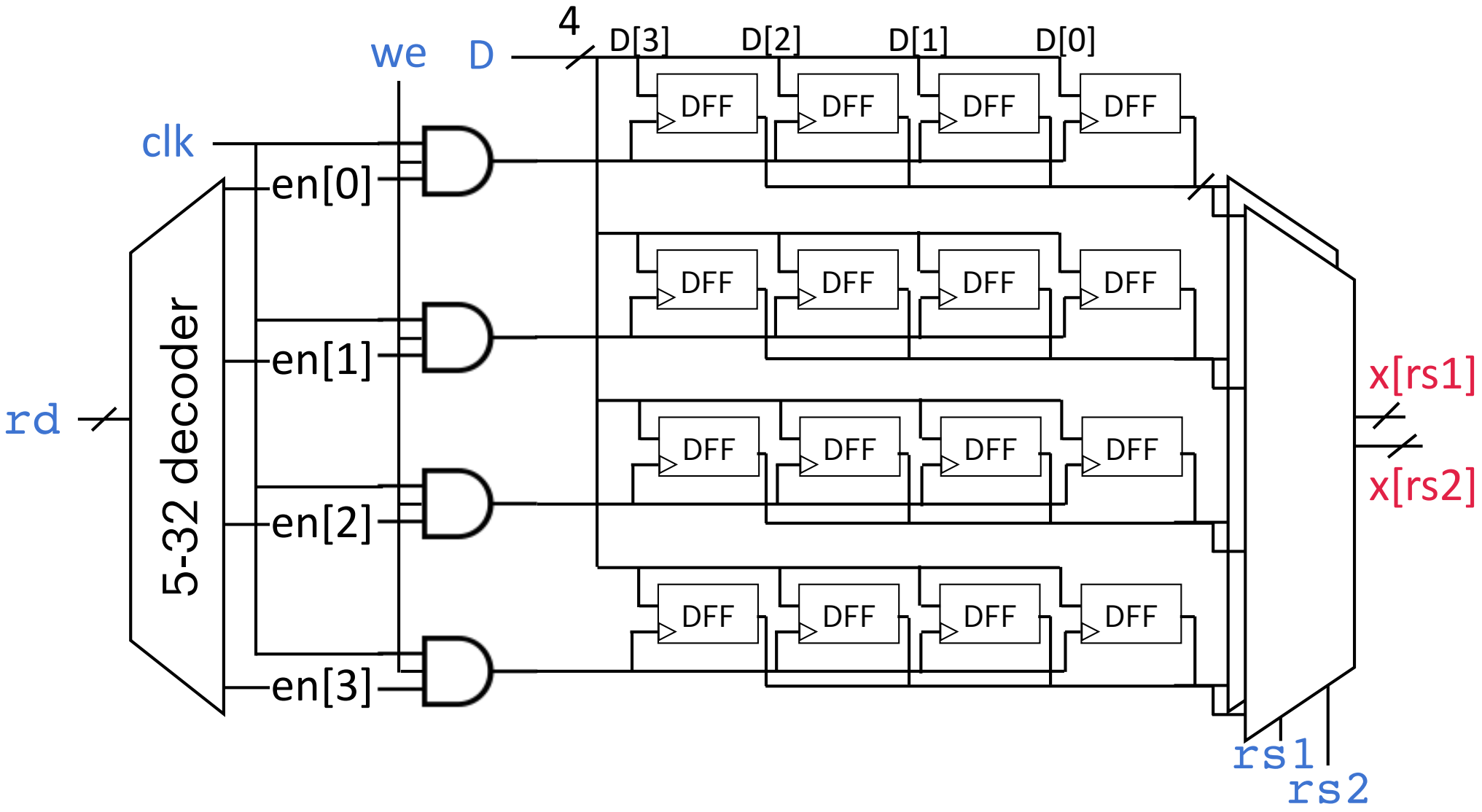


Full Register File

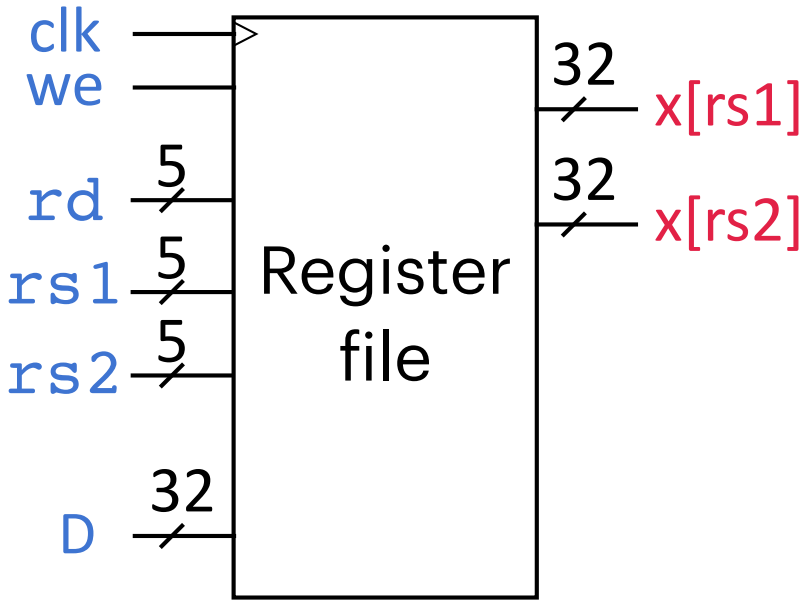


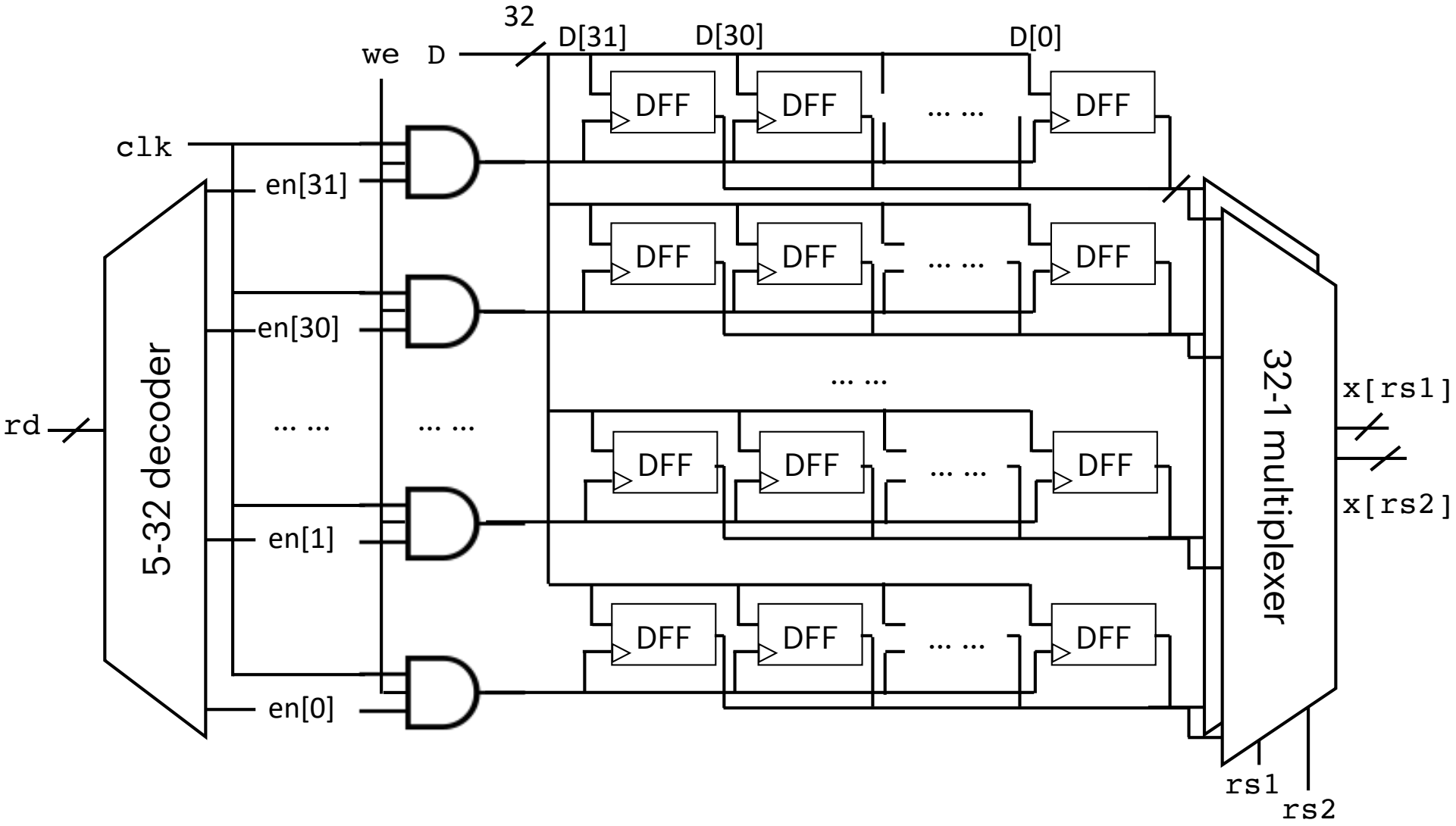
Usually another write enable signal

Full Register File—a Symbol

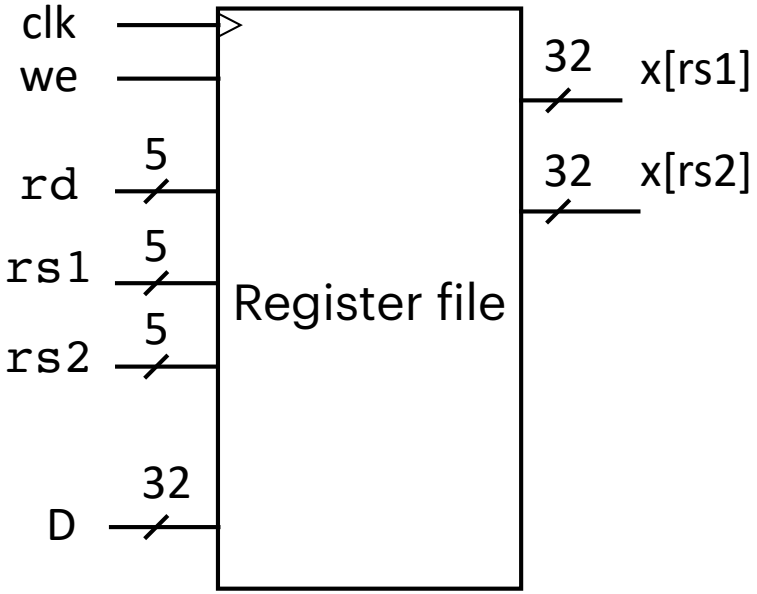


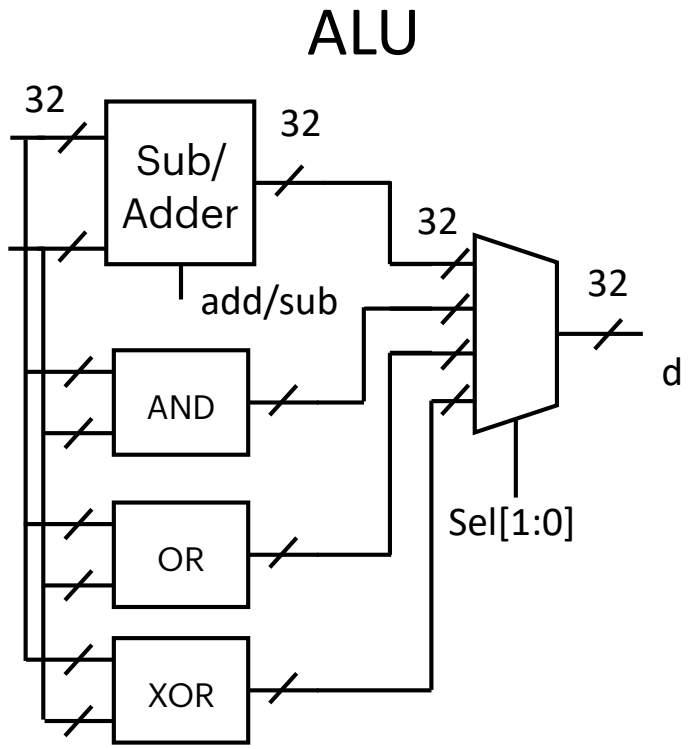
Full Register File—a Symbol



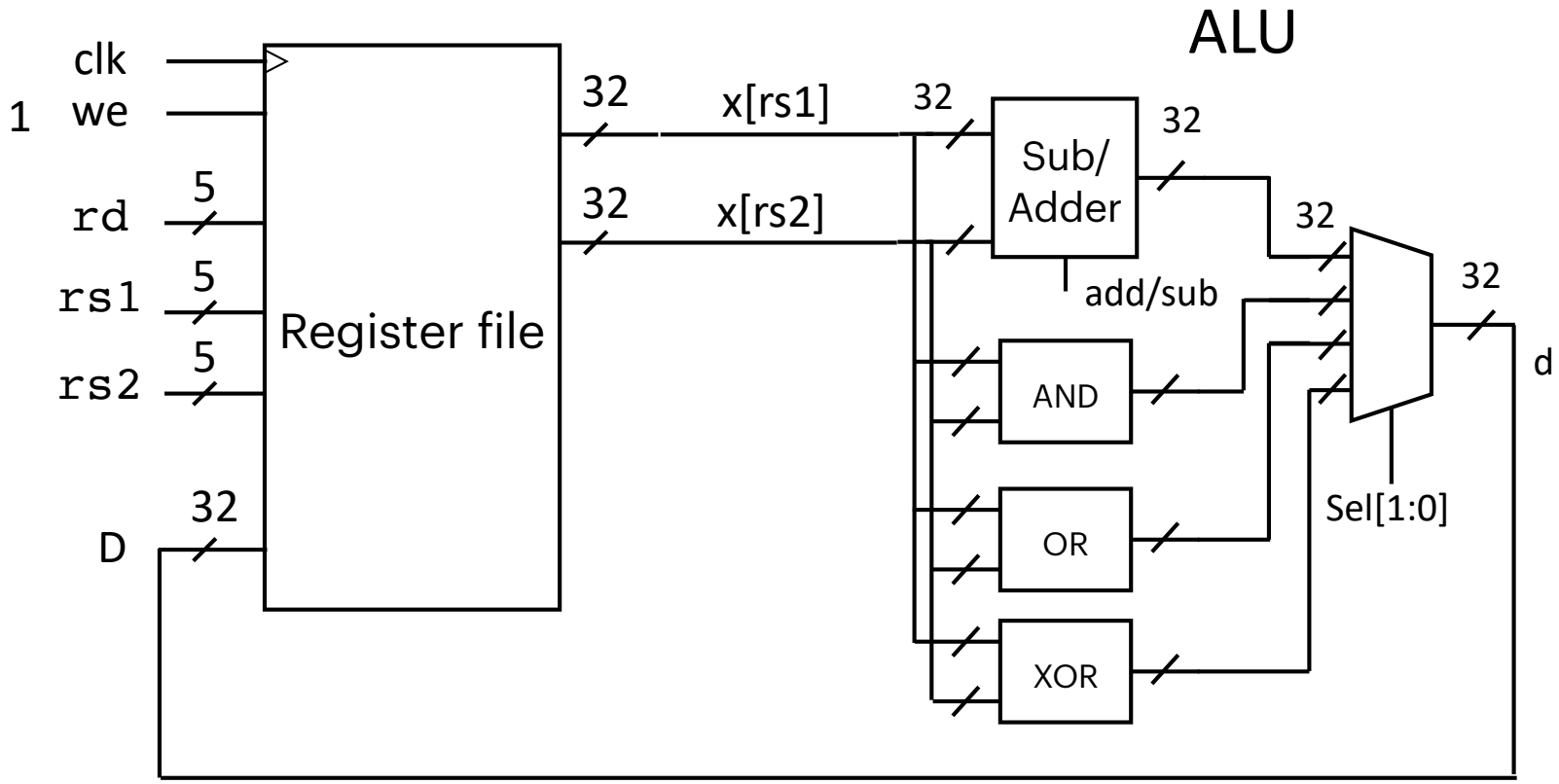

32
we
D
D[31]
D[30]
D[0]
clk
en[31]
en[30]
en[1]
en[0]
rd
5-32 decoder
DFF
... ...
32-1 multiplexer
x[rs1]
x[rs2]
rs1
rs2

Datapath Construction

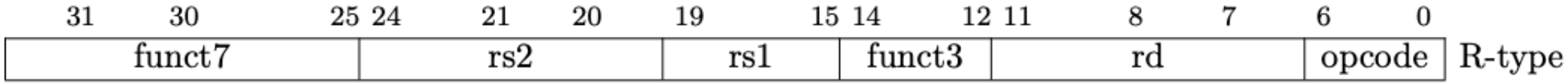


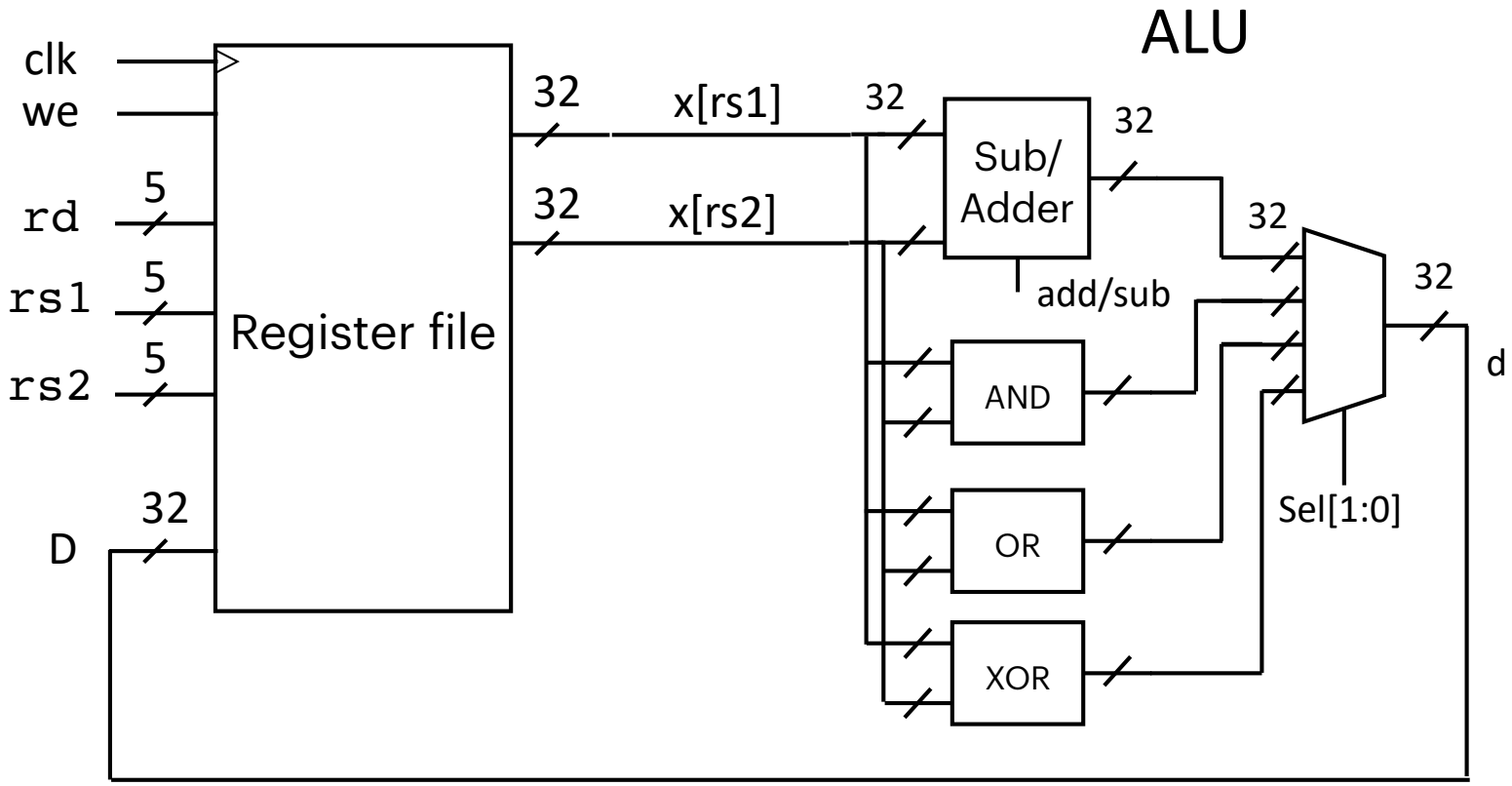


Datapath Construction

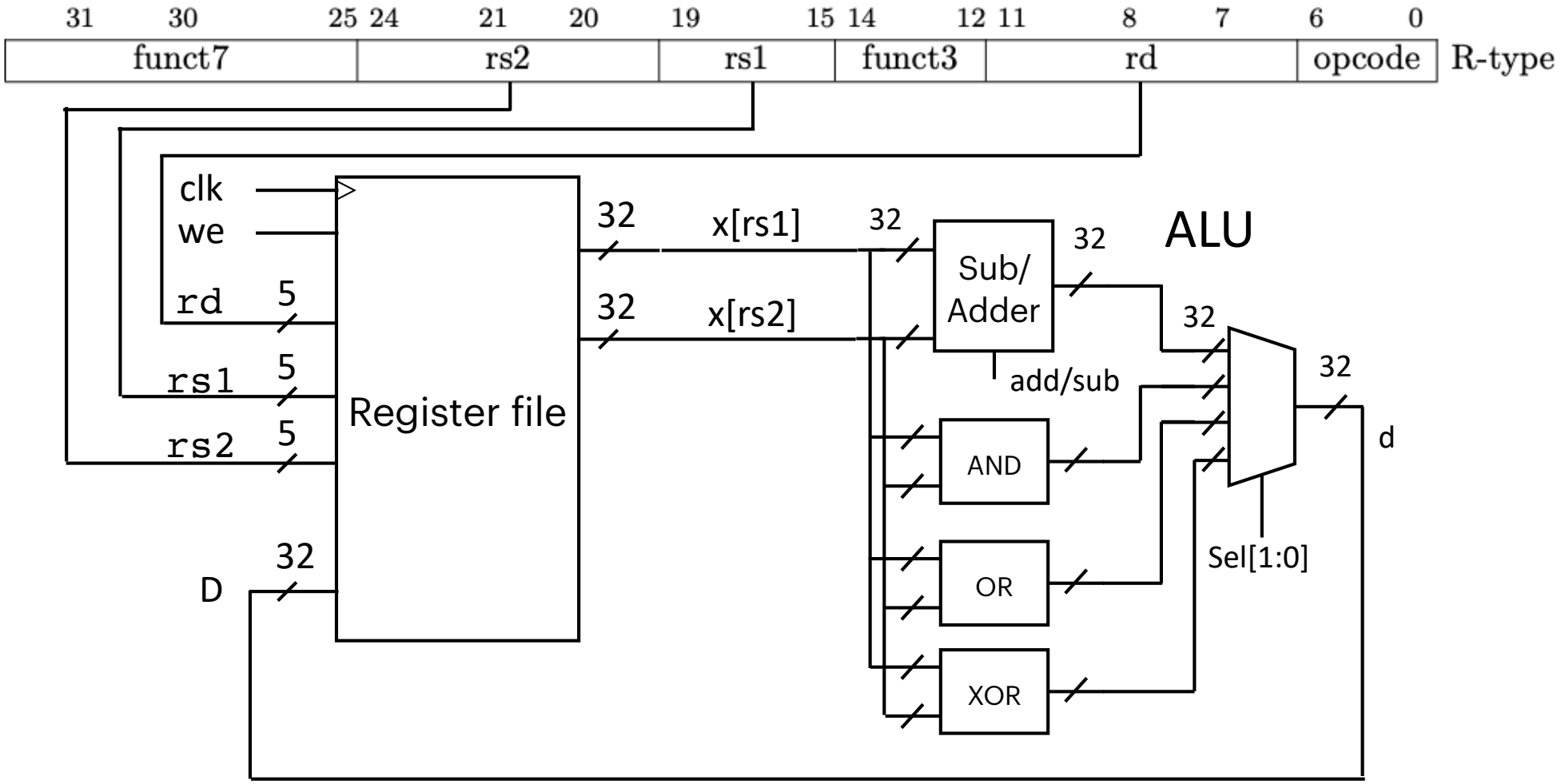


Datapath Construction

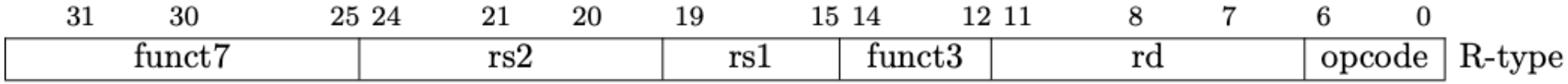


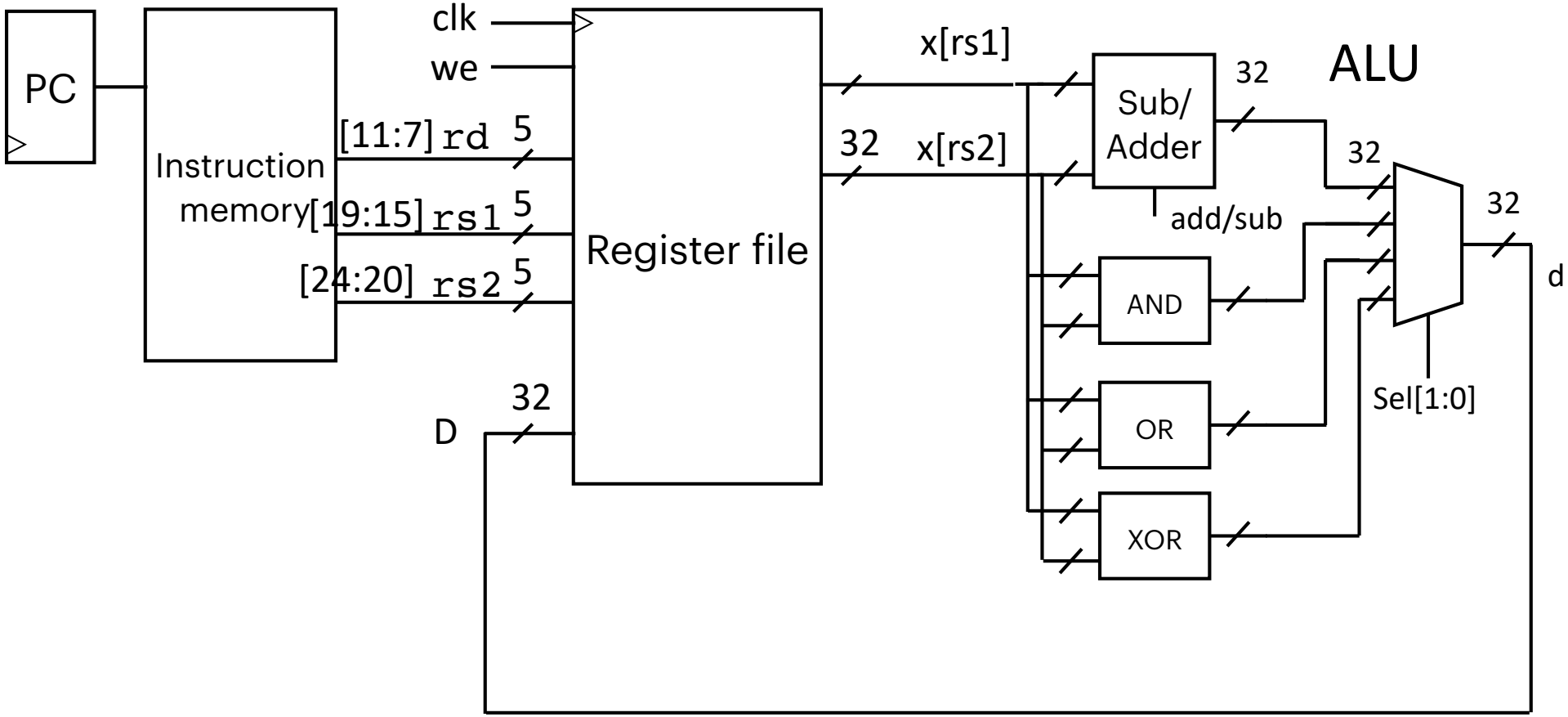


Datapath Construction

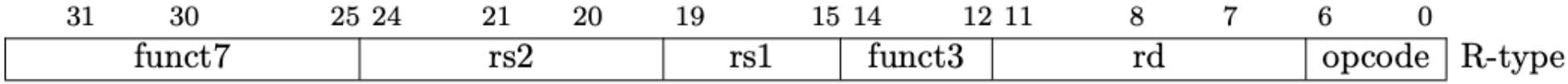


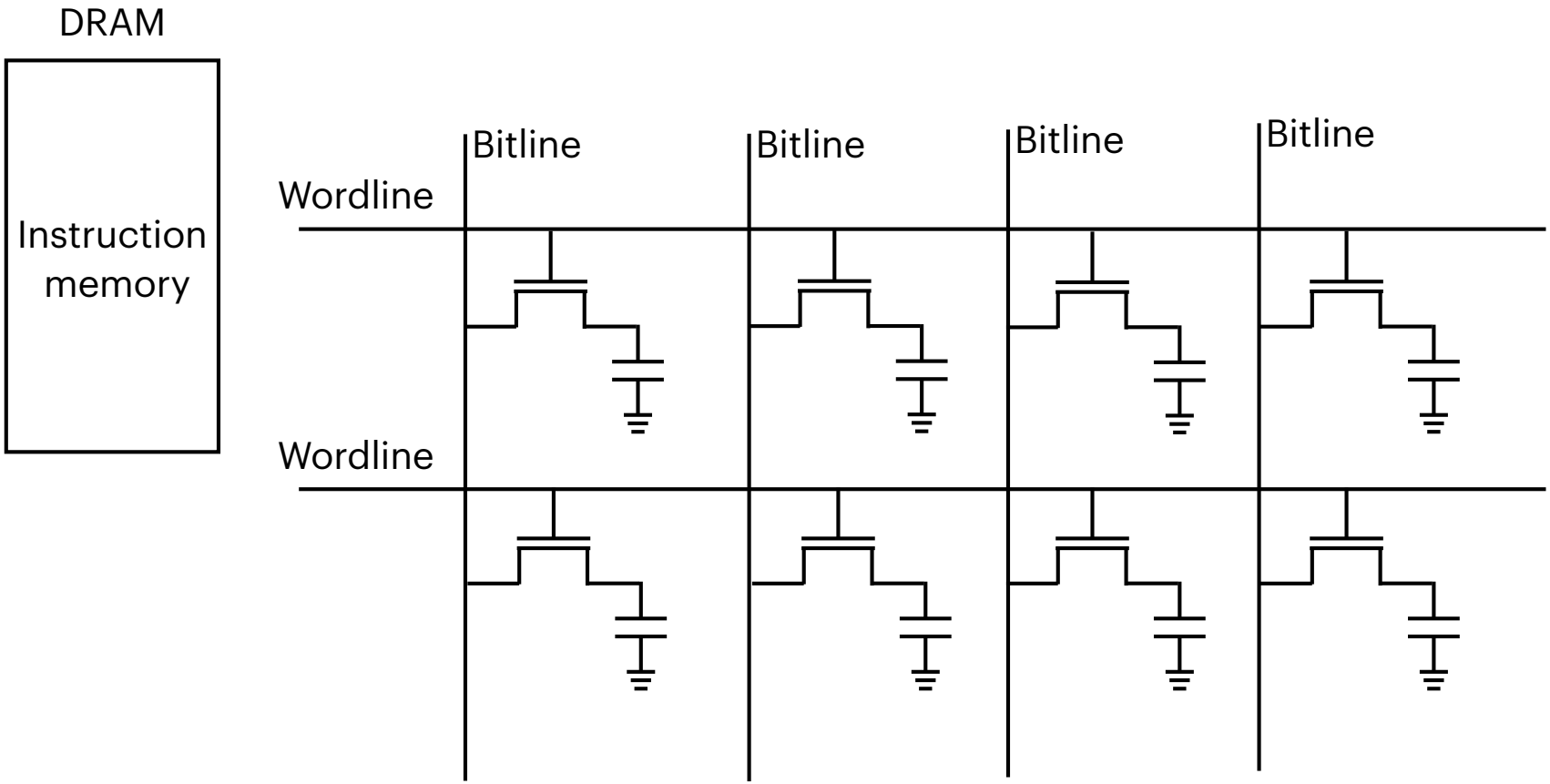
Datapath Construction



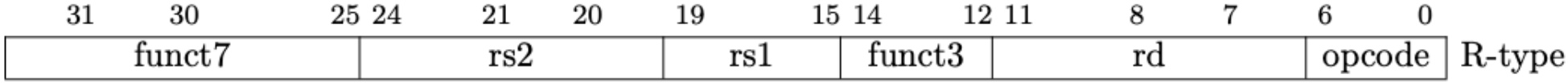


Datapath Construction

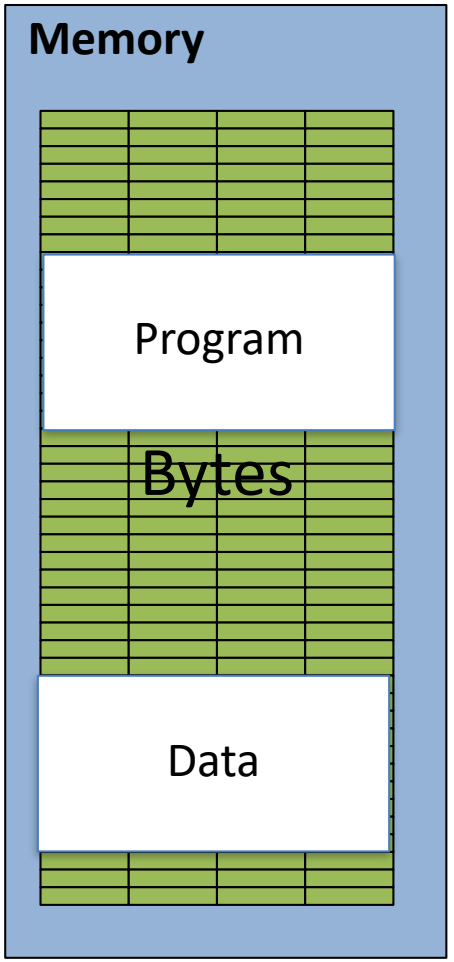




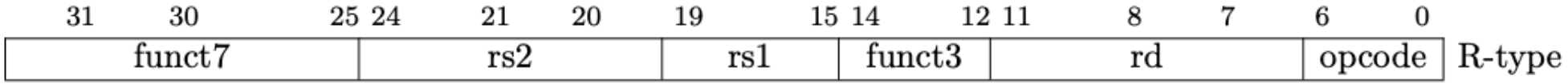
Datapath Construction

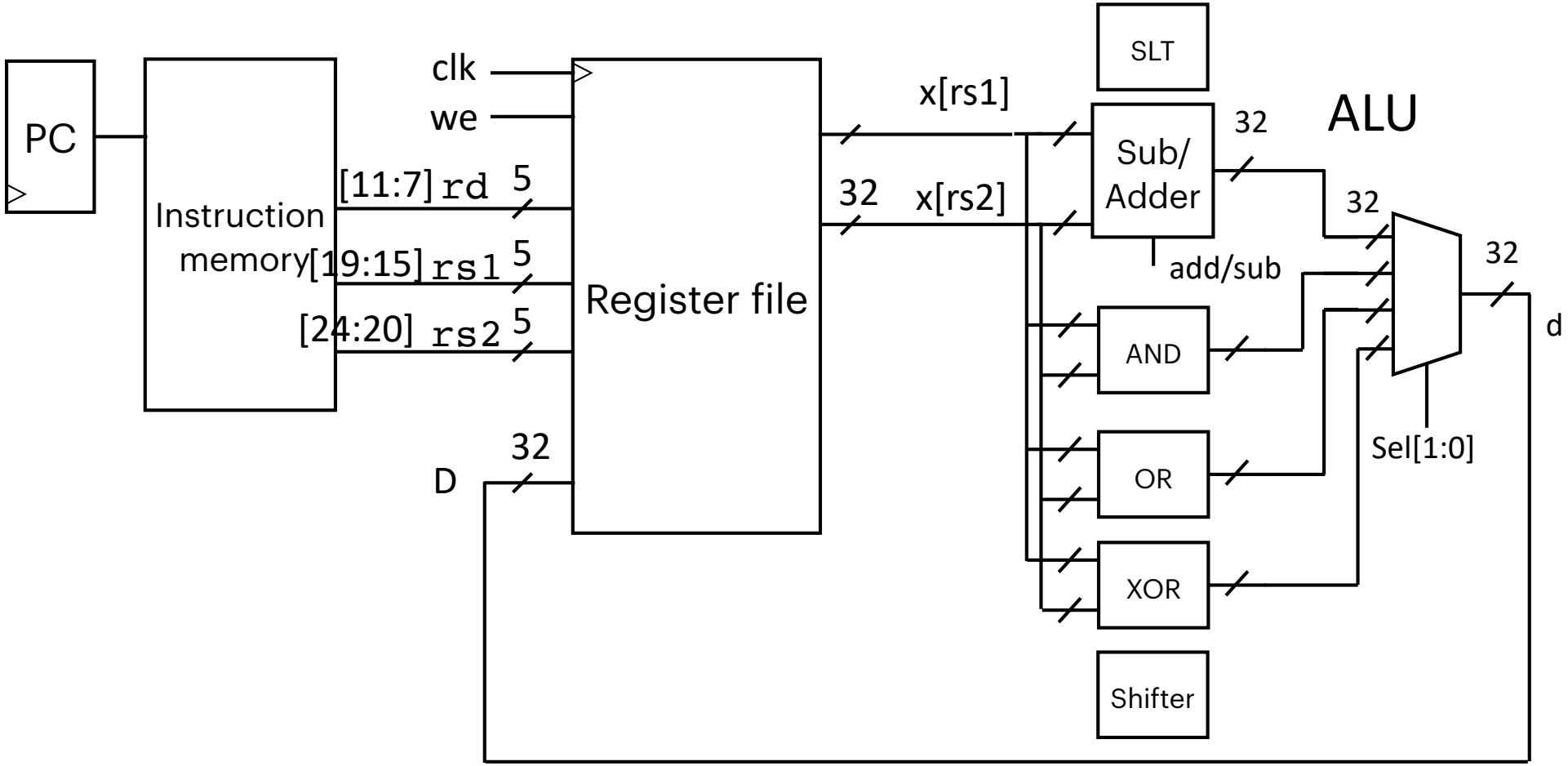


- Other memory considerations
  - Assume separate instruction/data memory
  - Synchronous write & asynchronous read so that all the state elements update at `posedge`
  - Registers behave similarly

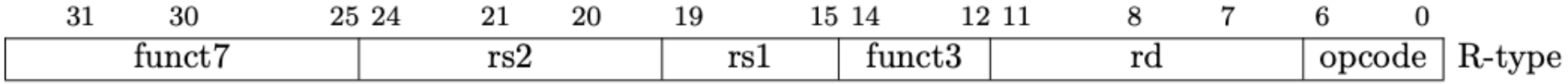


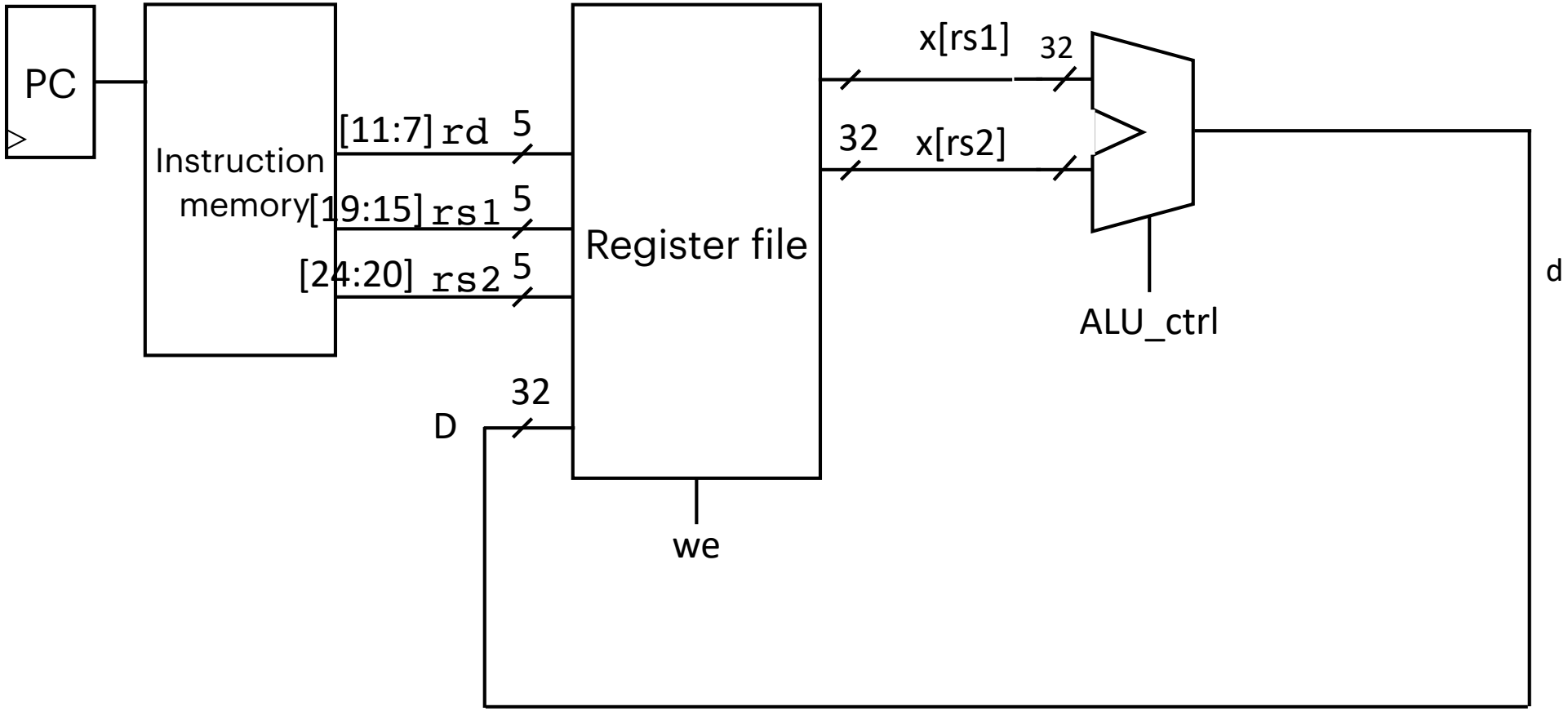
Datapath Construction



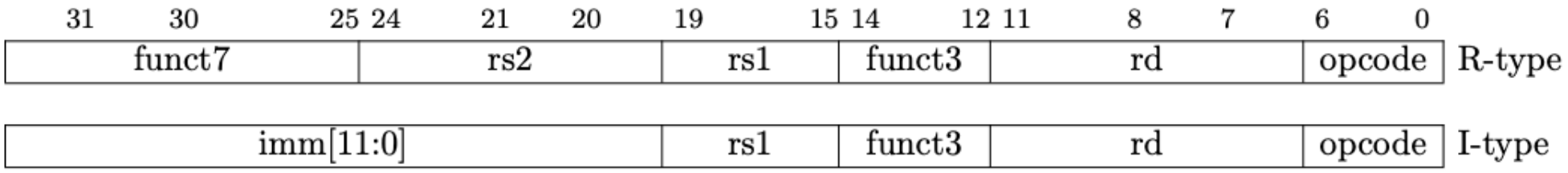


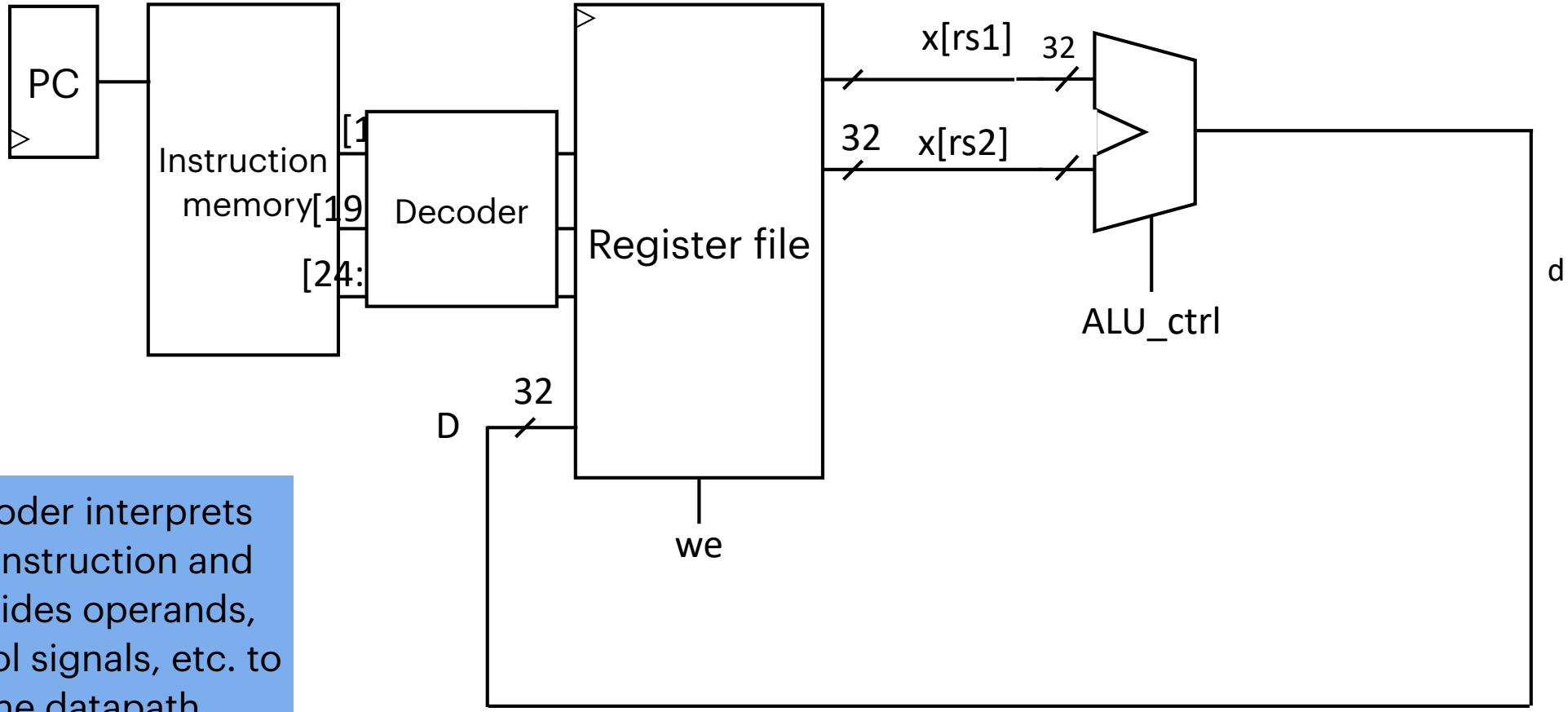
Datapath Construction





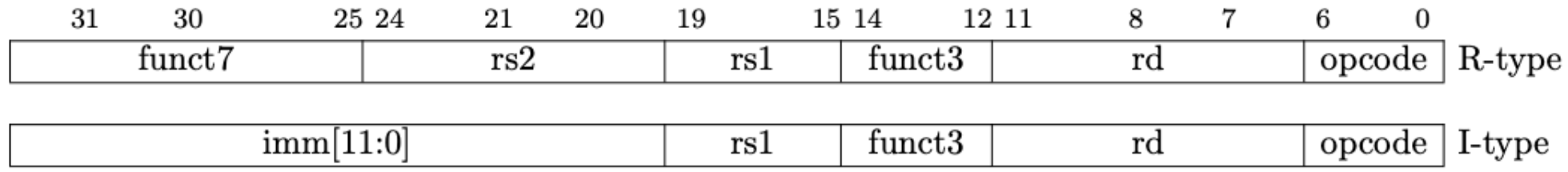
I-type Arithmetic & Logic

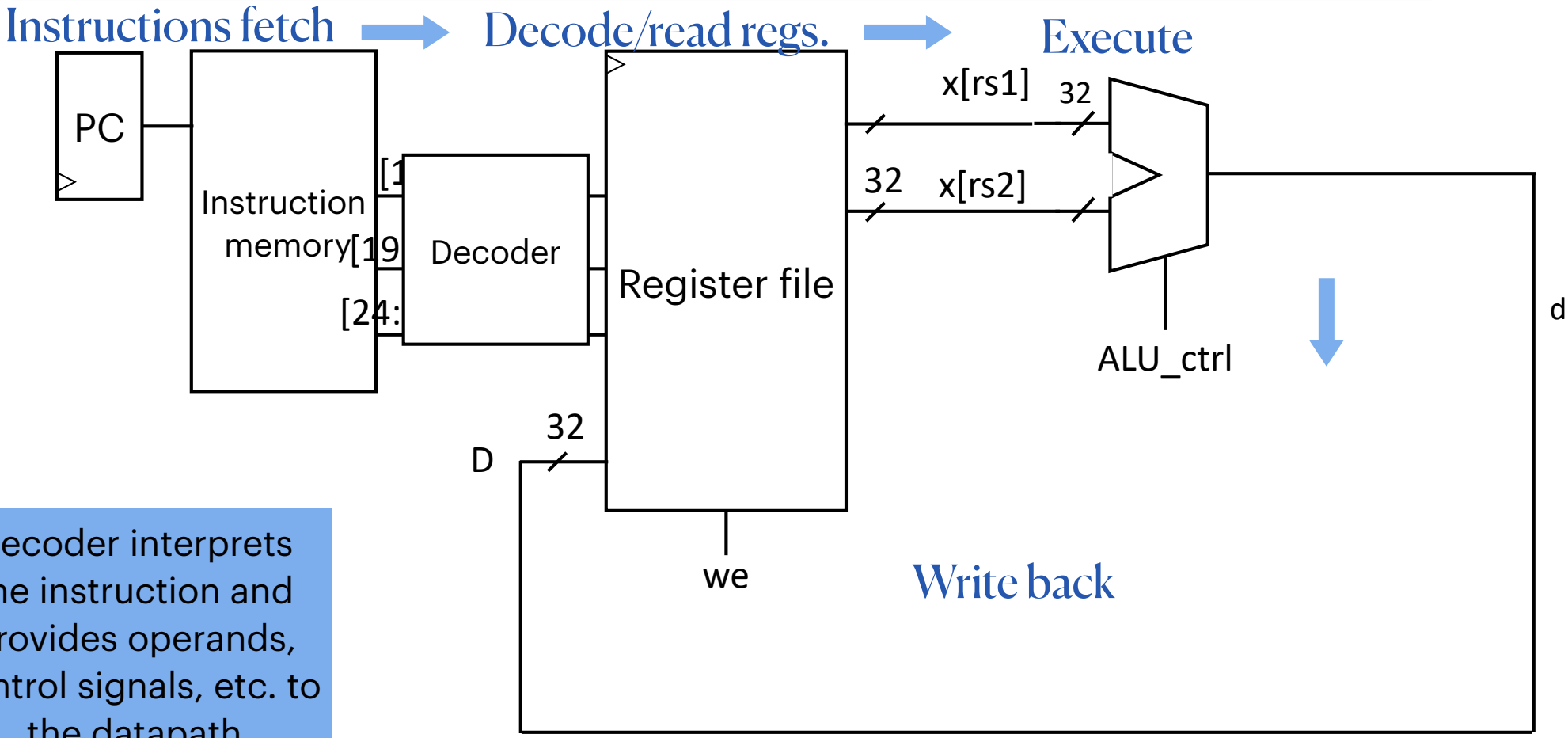




Decoder interprets the instruction and provides operands, control signals, etc. to the datapath

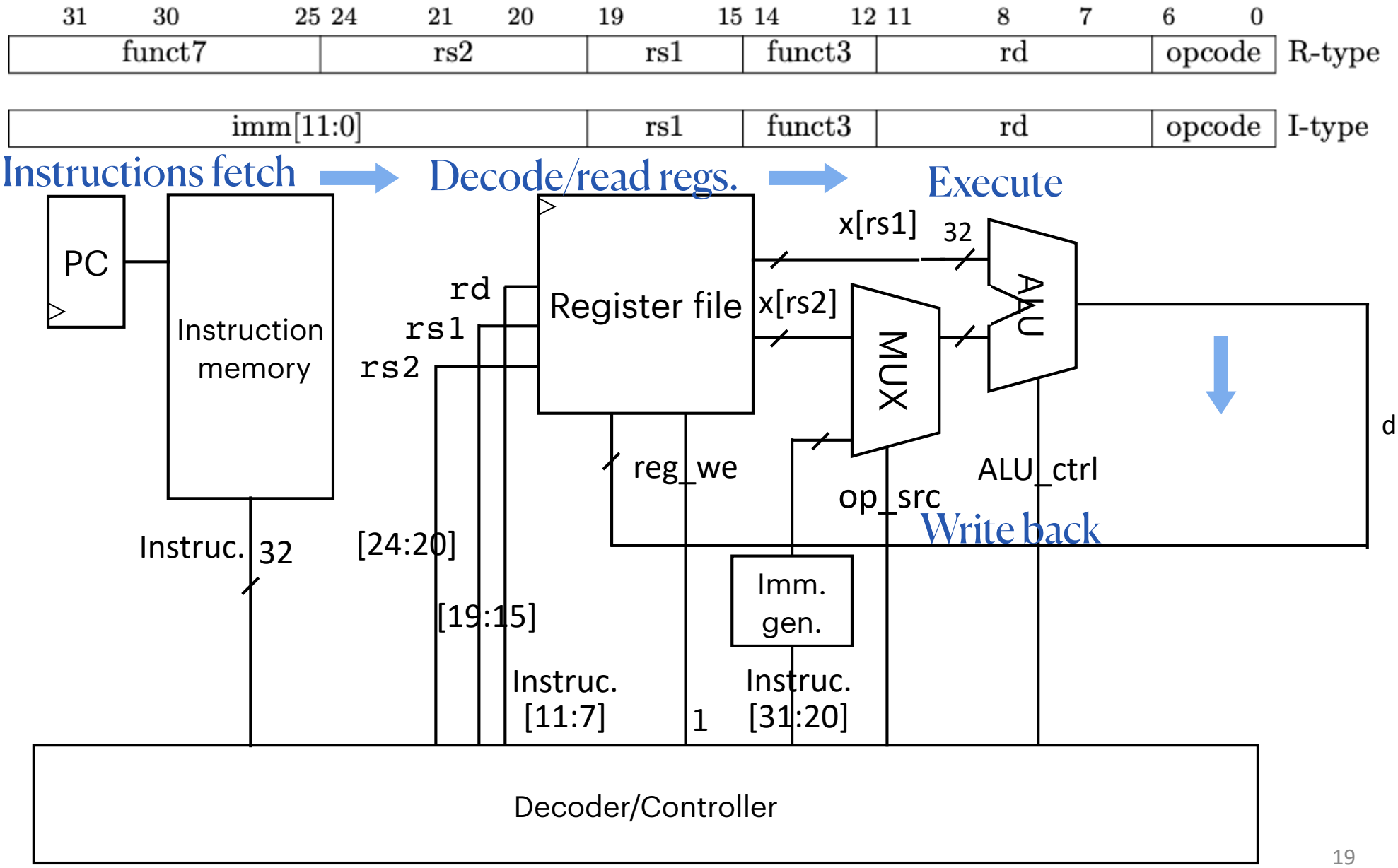
Arithmetic & Logic



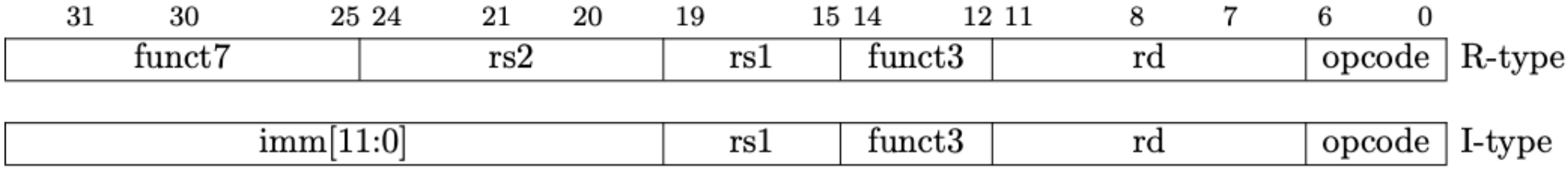


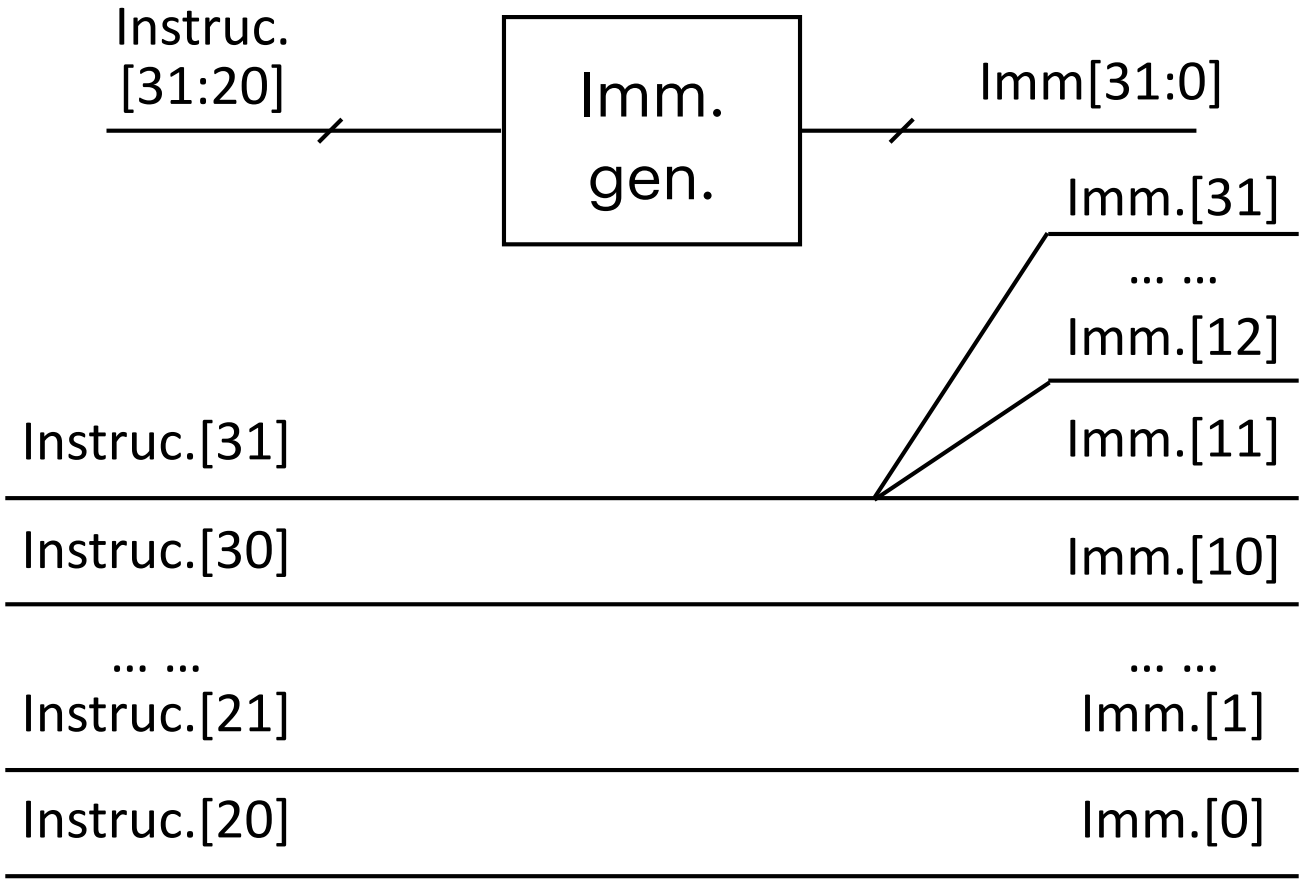
Decoder interprets the instruction and provides operands, control signals, etc. to the datapath

Arithmetic & Logic

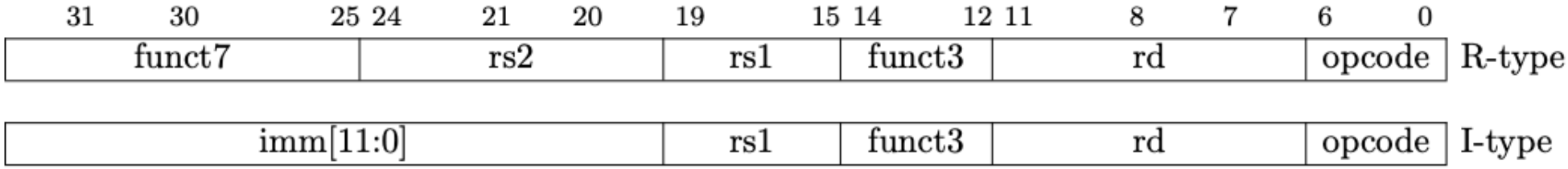


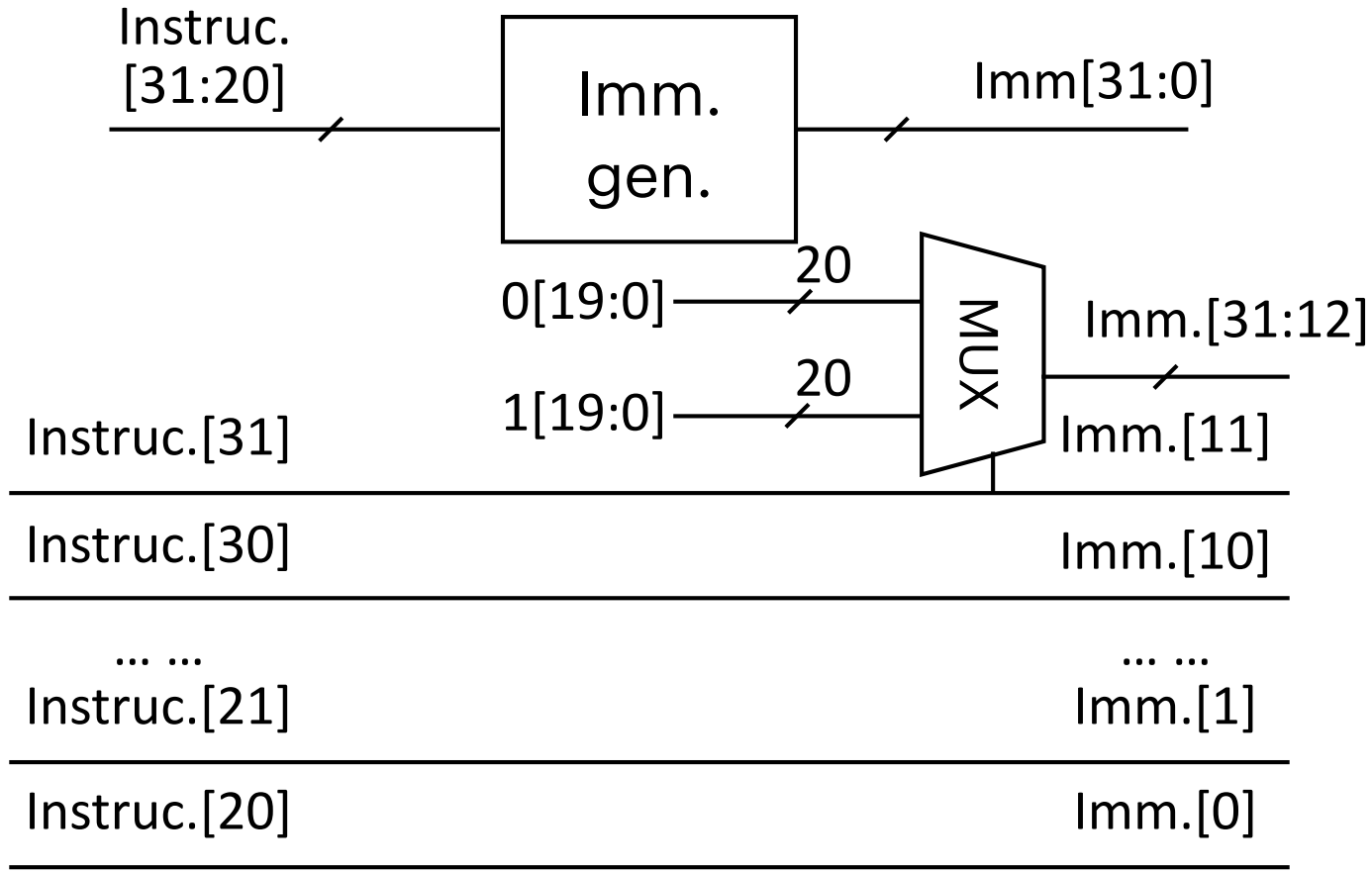
Immediate Generation



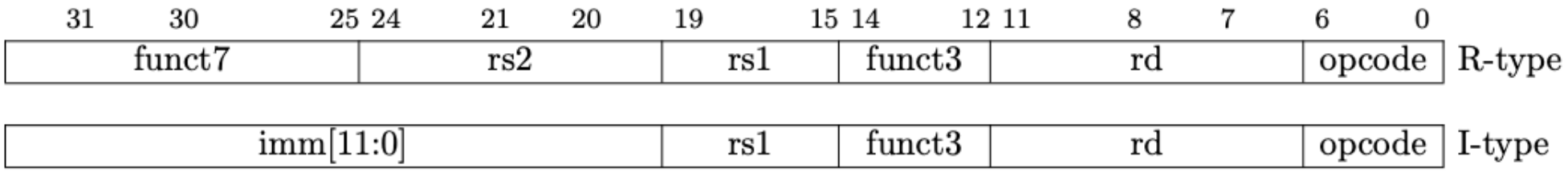


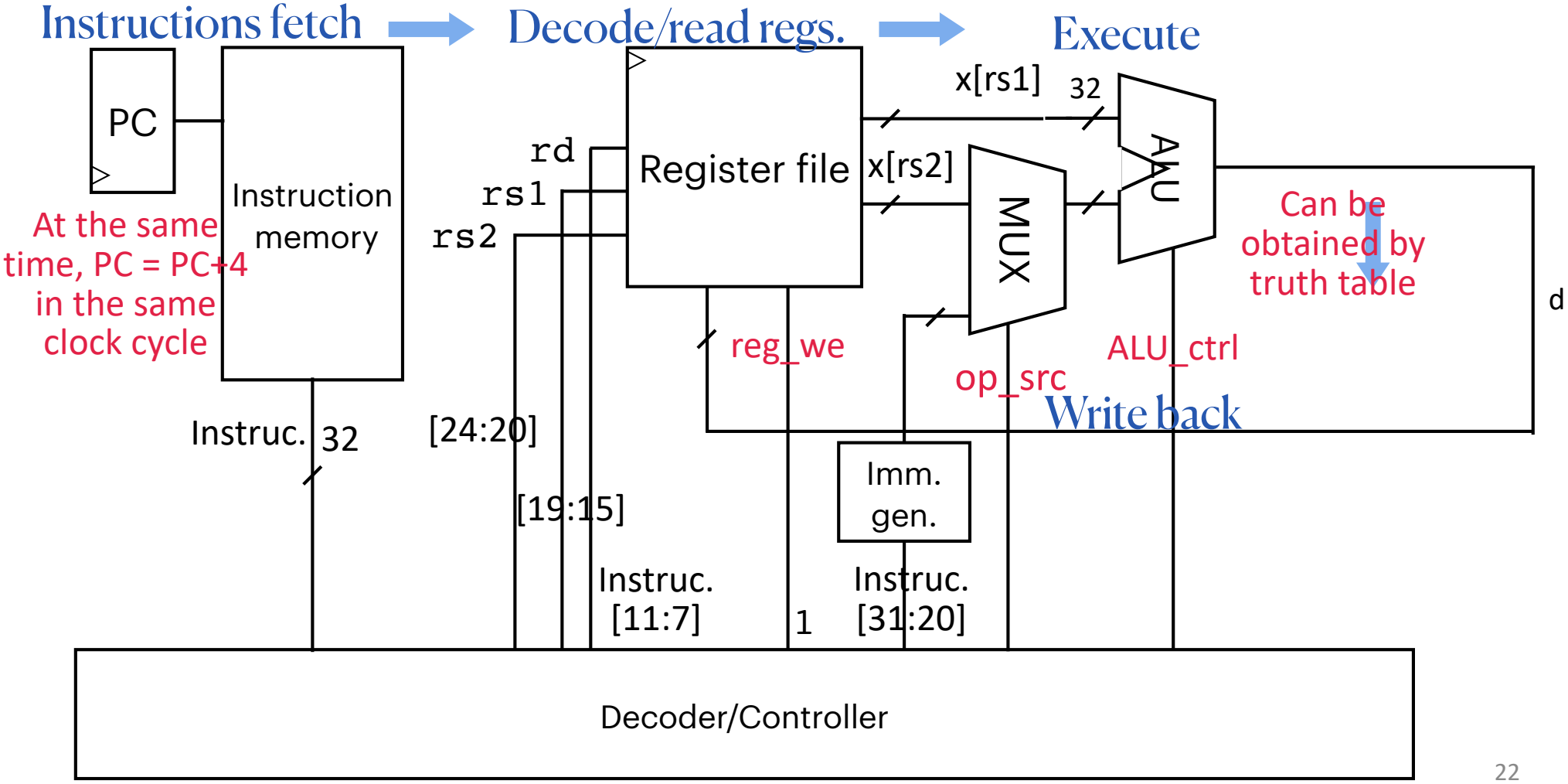
Immediate Generation



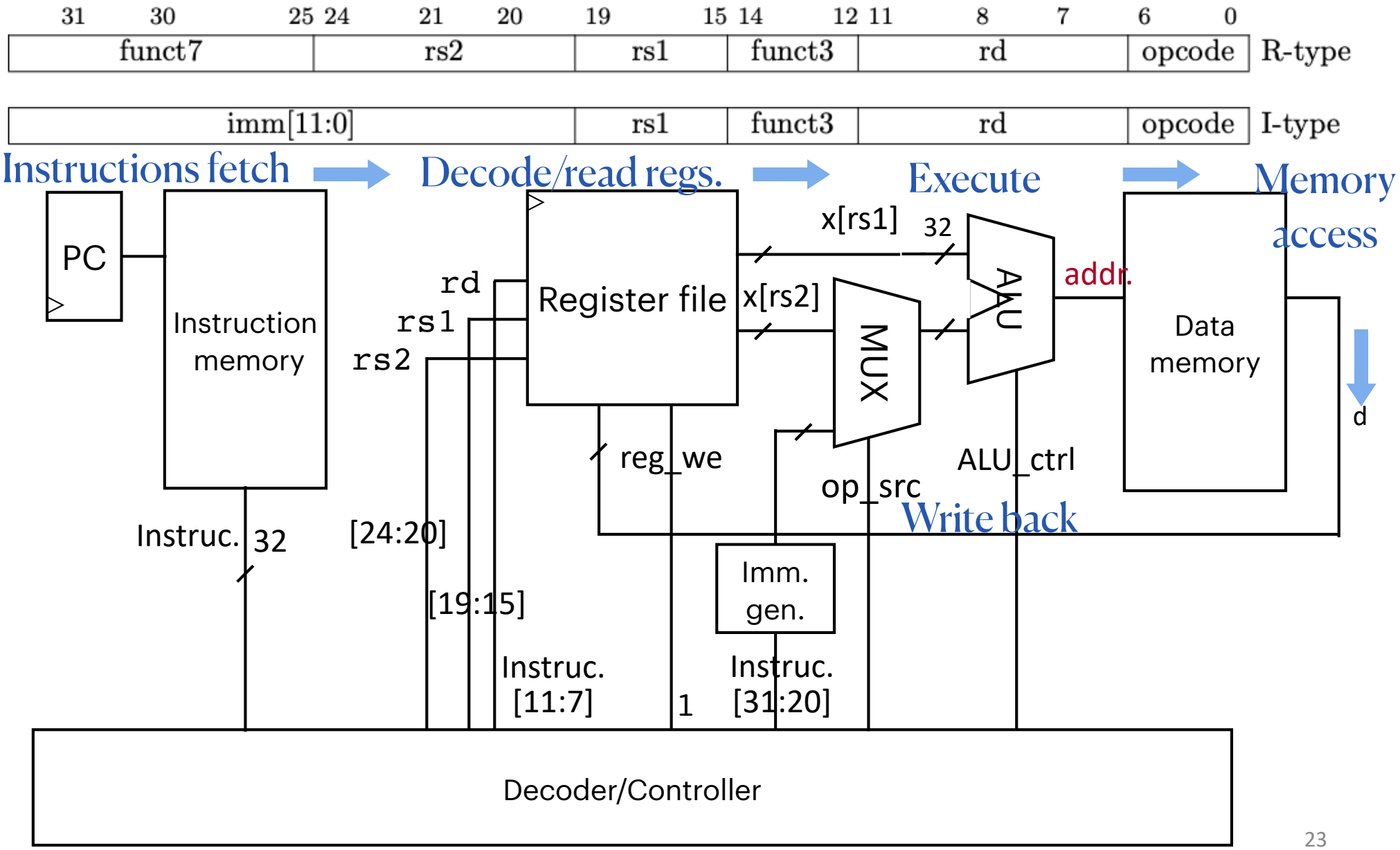


Arithmetic & Logic

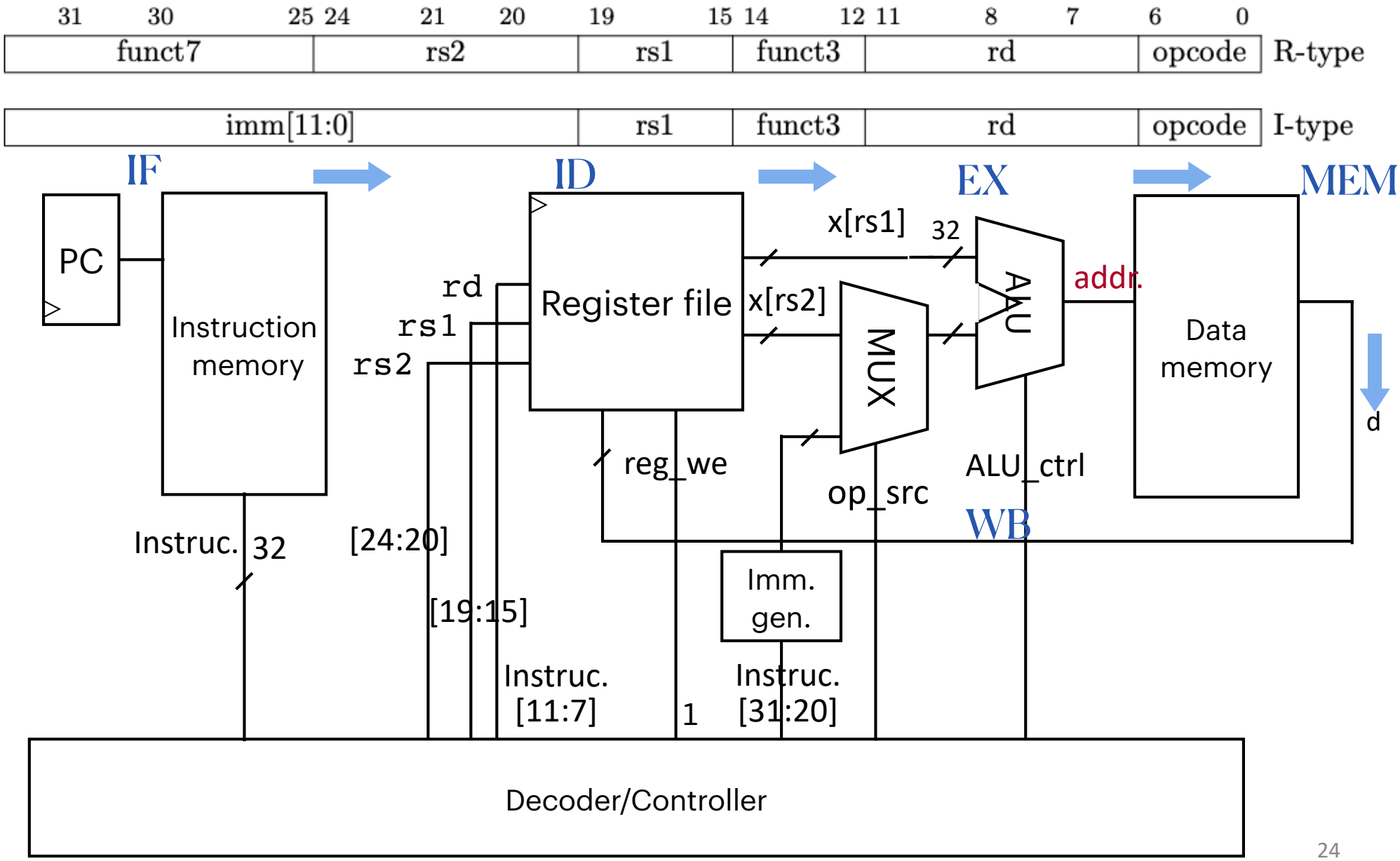




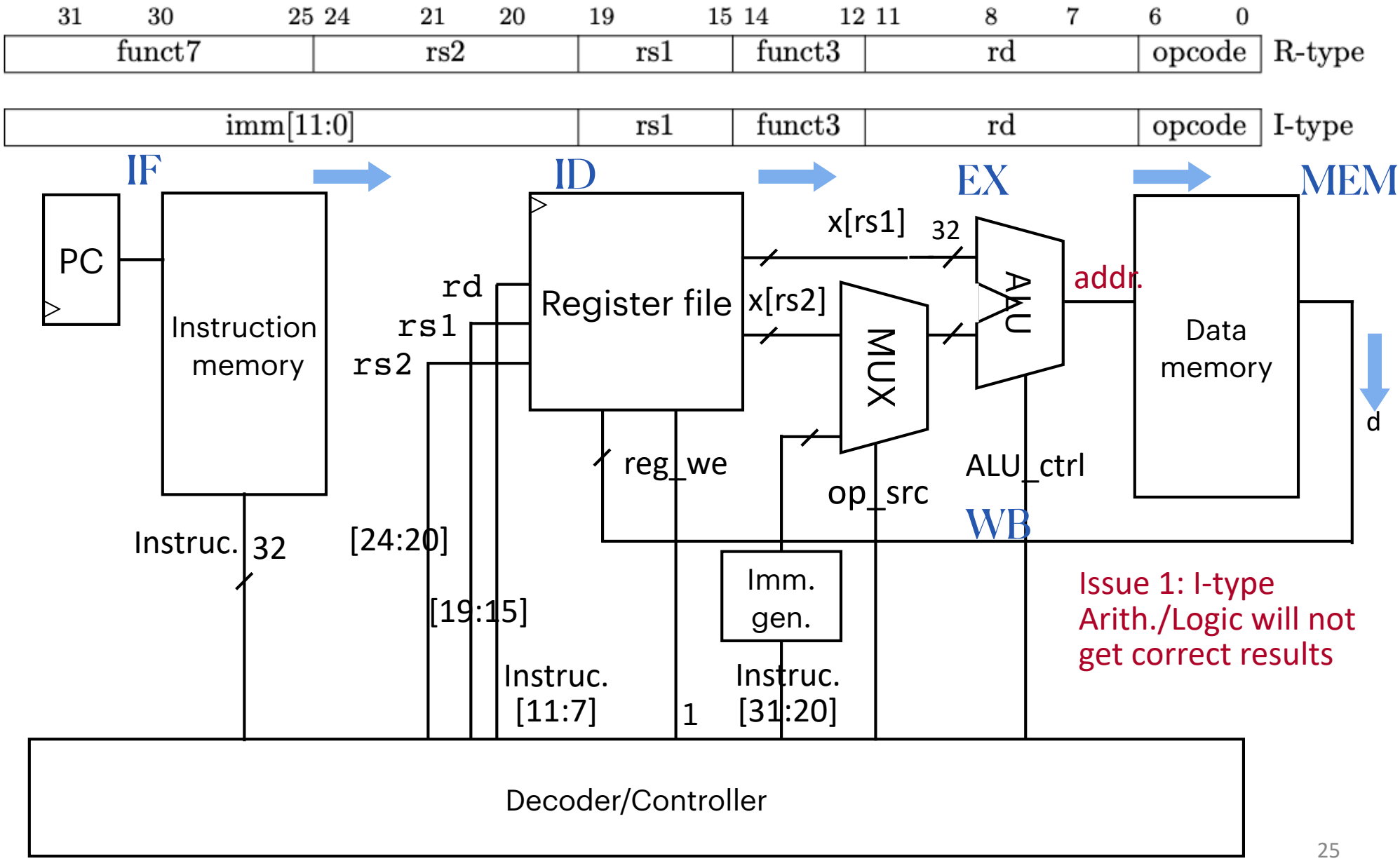
I-type Load


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
Instructions fetch
Decode/read regs.
Execute
Memory access
PC
Instruction memory
Register file
x[rs1]
32
x[rs2]
MUX
ALU
addr.
Data memory
d
rd
rs1
rs2
reg_we
op_src
ALU_ctrl
Write back
Instruc. 32
[24:20]
[19:15]
Instruc. [11:7]
Imm. gen.
1
Instruc. [31:20]
Decoder/Controller

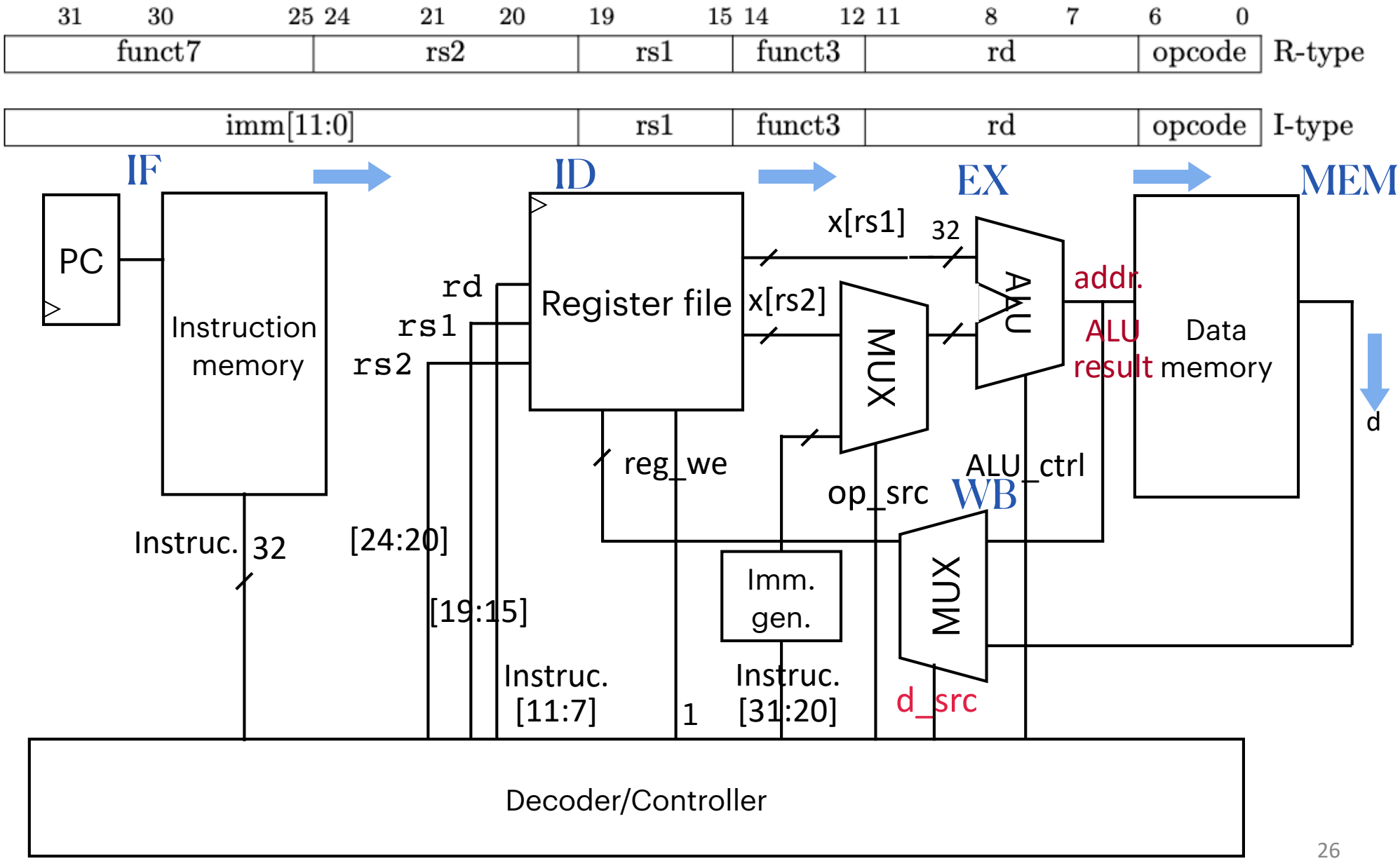
I-type Load


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
IF
ID
EX
MEM
PC
Instruction memory
Register file
x[rs1]
32
x[rs2]
ALU
addr.
Data memory
d
rd
rs1
rs2
MUX
reg_we
ALU_ctrl
op_src
WB
Instruc.
32
[24:20]
[19:15]
Instruc. [11:7]
Imm. gen.
1
Instruc. [31:20]
Decoder/Controller

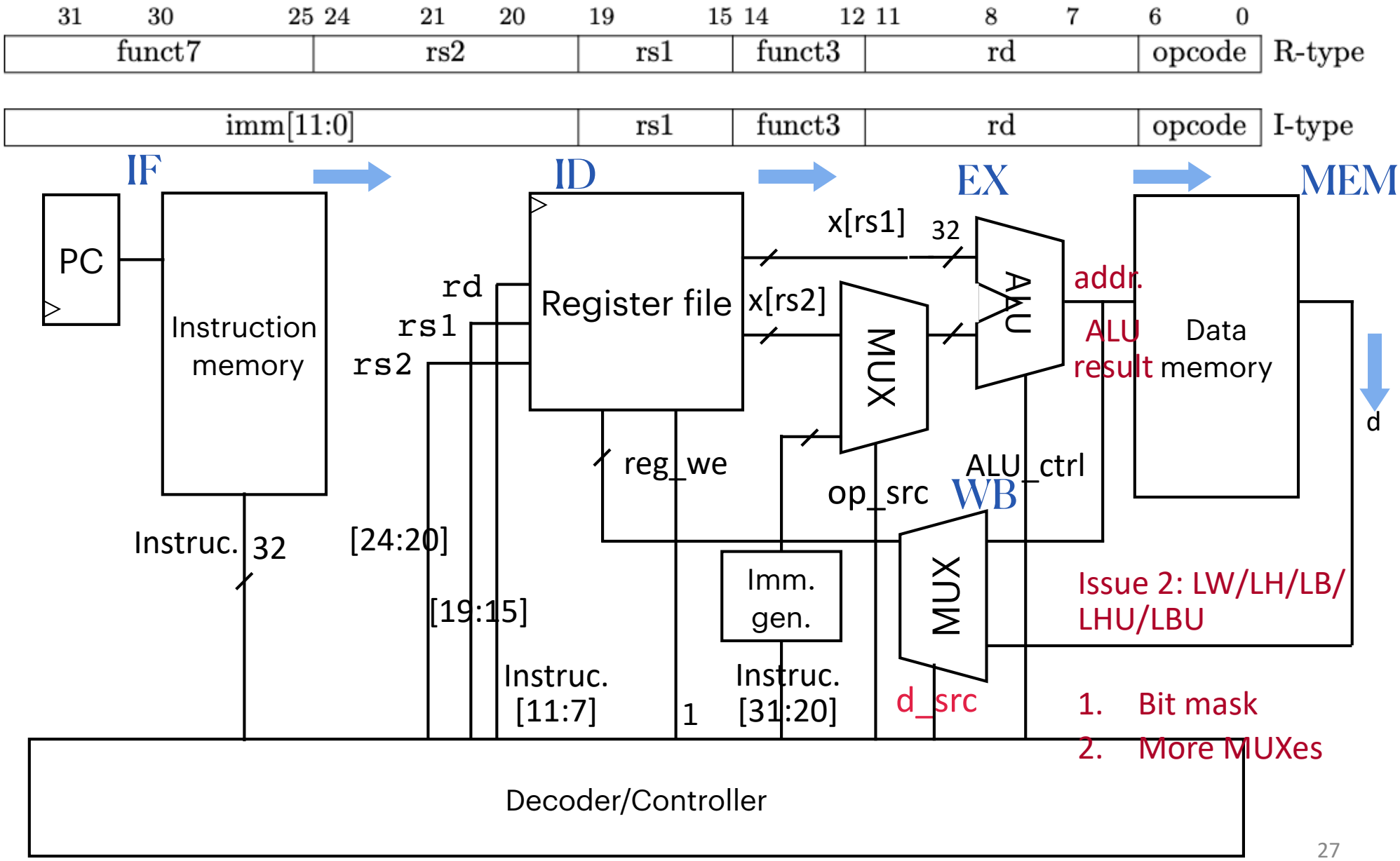
I-type Load


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
IF
ID
EX
MEM
PC
Instruction memory
Instruc. 32
[24:20]
rd
rs1
rs2
[19:15]
Register file
x[rs1]
32
x[rs2]
MUX
ALU
addr.
Data memory
d
reg_we
op_src
ALU_ctrl
WB
Imm. gen.
Instruc. [11:7]
1
Instruc. [31:20]
Decoder/Controller
Issue 1: I-type Arith./Logic will not get correct results

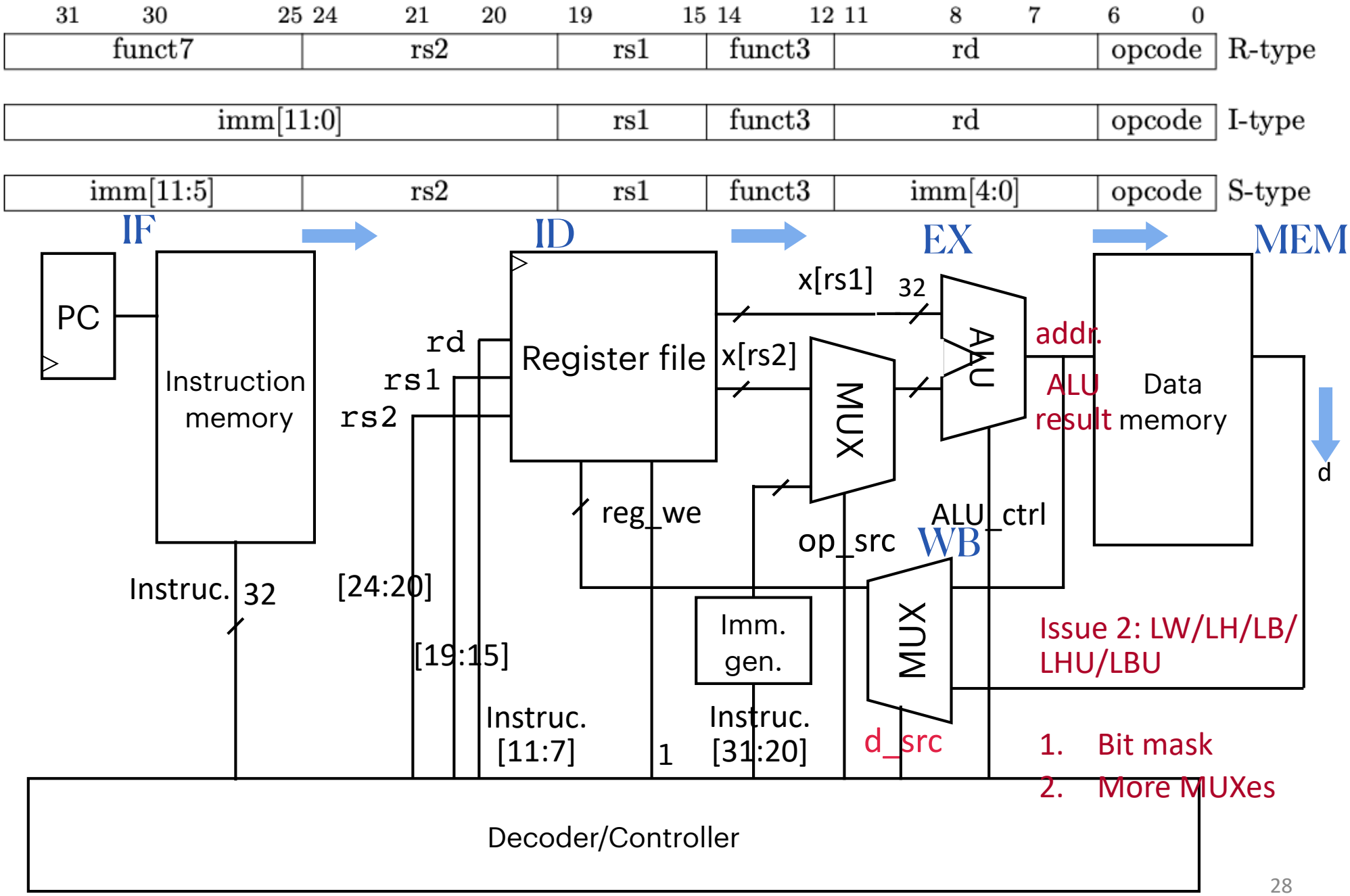
I-type Load


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
IF
ID
EX
MEM
PC
Instruction memory
Instruc.
32
[24:20]
rd
rs1
rs2
[19:15]
Register file
x[rs1]
32
x[rs2]
MUX
ALU
addr.
ALU result
Data memory
d
reg_we
op_src
ALU_ctrl
WB
Imm. gen.
Instruc. [11:7]
1
Instruc. [31:20]
MUX
d_src
Decoder/Controller

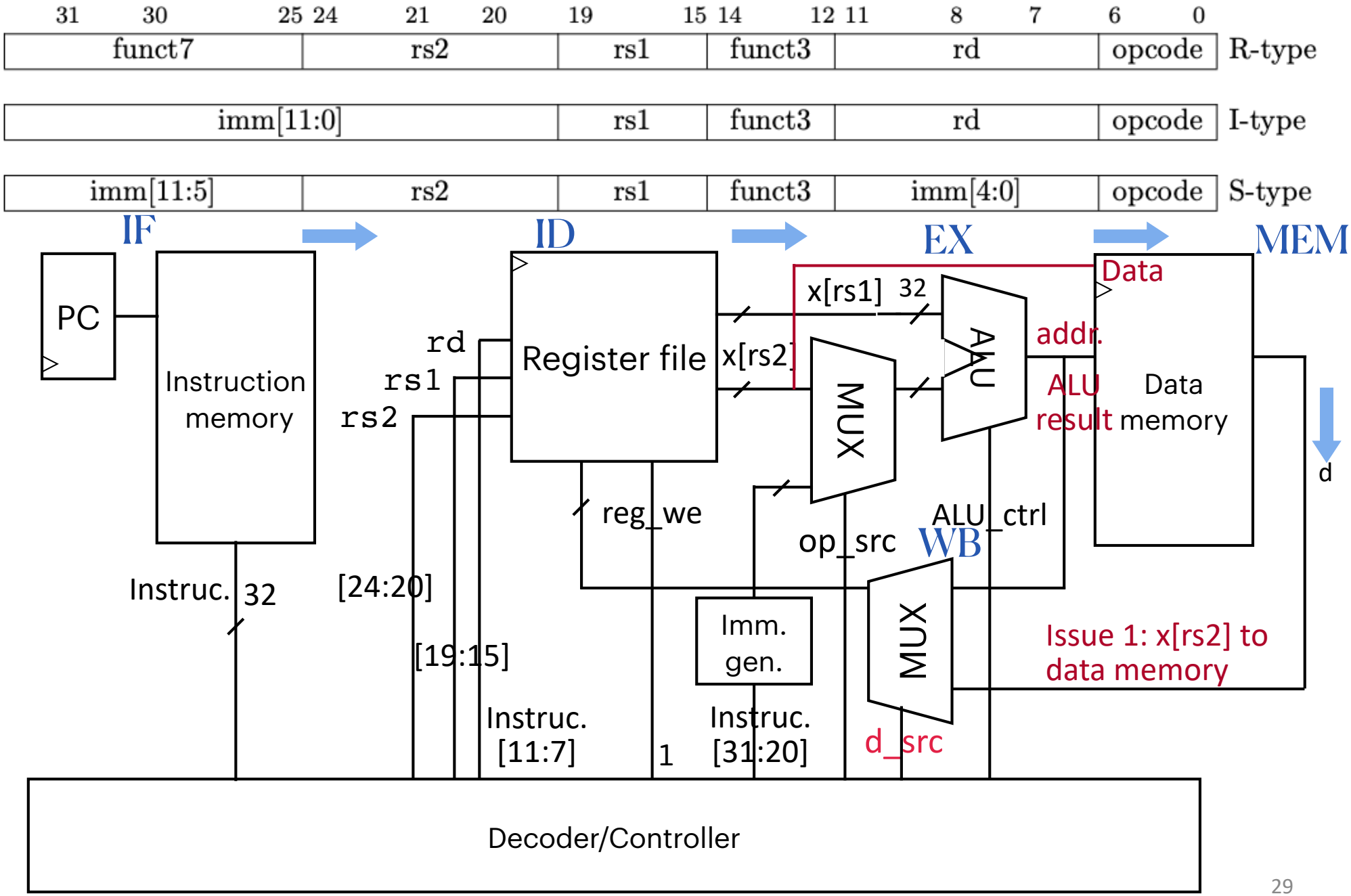
I-type Load


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
IF
ID
EX
MEM
PC
Instruction memory
Instruc.
32
[24:20]
rd
rs1
rs2
[19:15]
Register file
x[rs1]
32
x[rs2]
MUX
ALU
addr.
ALU result
Data memory
d
reg_we
op_src
ALU_ctrl
WB
Imm. gen.
Instruc. [11:7]
1
Instruc. [31:20]
MUX
d_src
Issue 2: LW/LH/LB/ LHU/LBU
1. Bit mask
2. More MUXes
Decoder/Controller

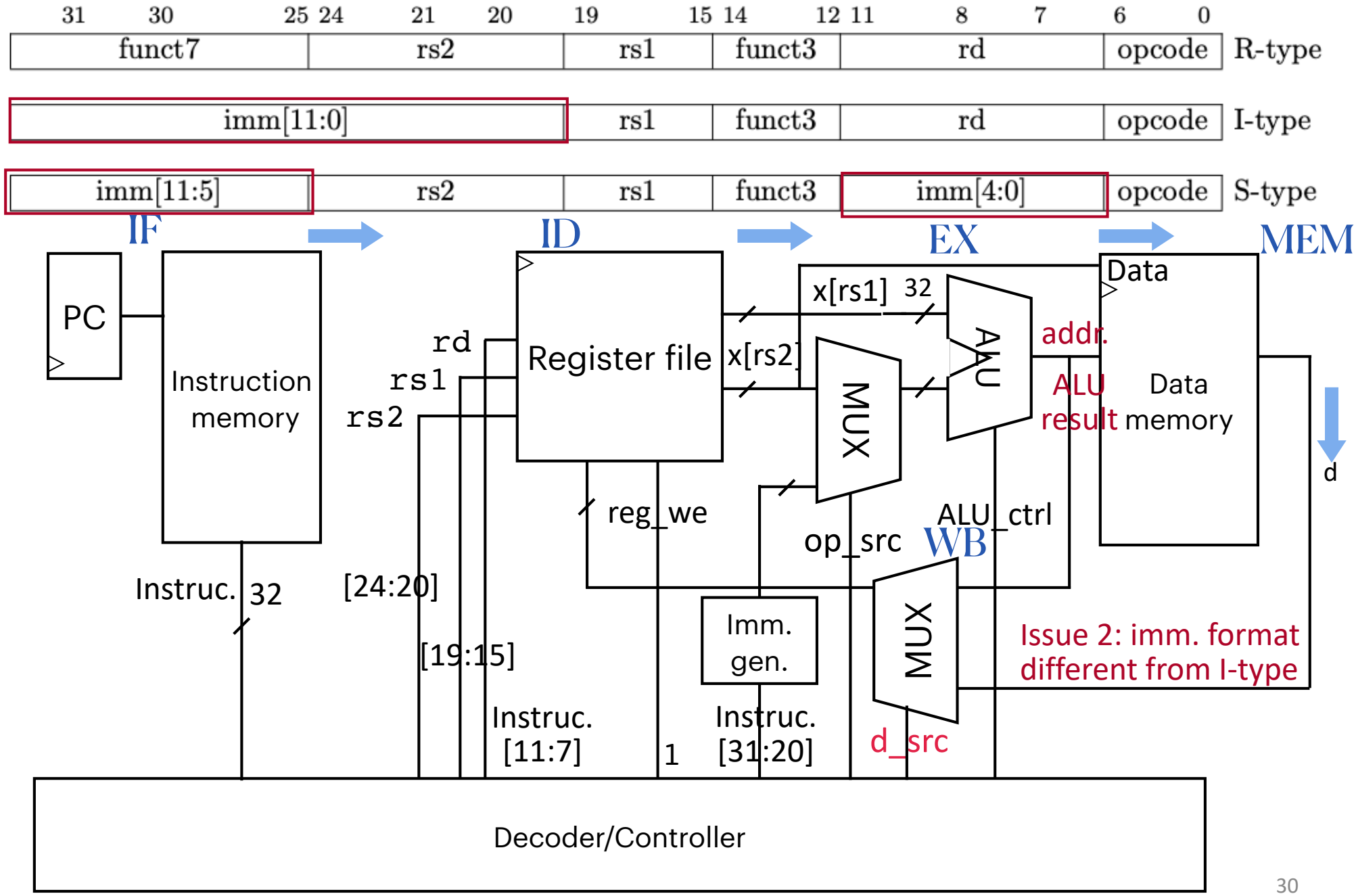
S-type Store


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
IF
ID
EX
MEM
PC
Instruction memory
Instruc. 32
[24:20]
[19:15]
rd
rs1
rs2
Register file
x[rs1]
32
x[rs2]
MUX
ALU
addr.
ALU result
Data memory
d
reg_we
op_src
ALU_ctrl
WB
Imm. gen.
Instruc. [11:7]
1
Instruc. [31:20]
MUX
d_src
Issue 2: LW/LH/LB/ LHU/LBU
1. Bit mask
2. More MUXes
Decoder/Controller

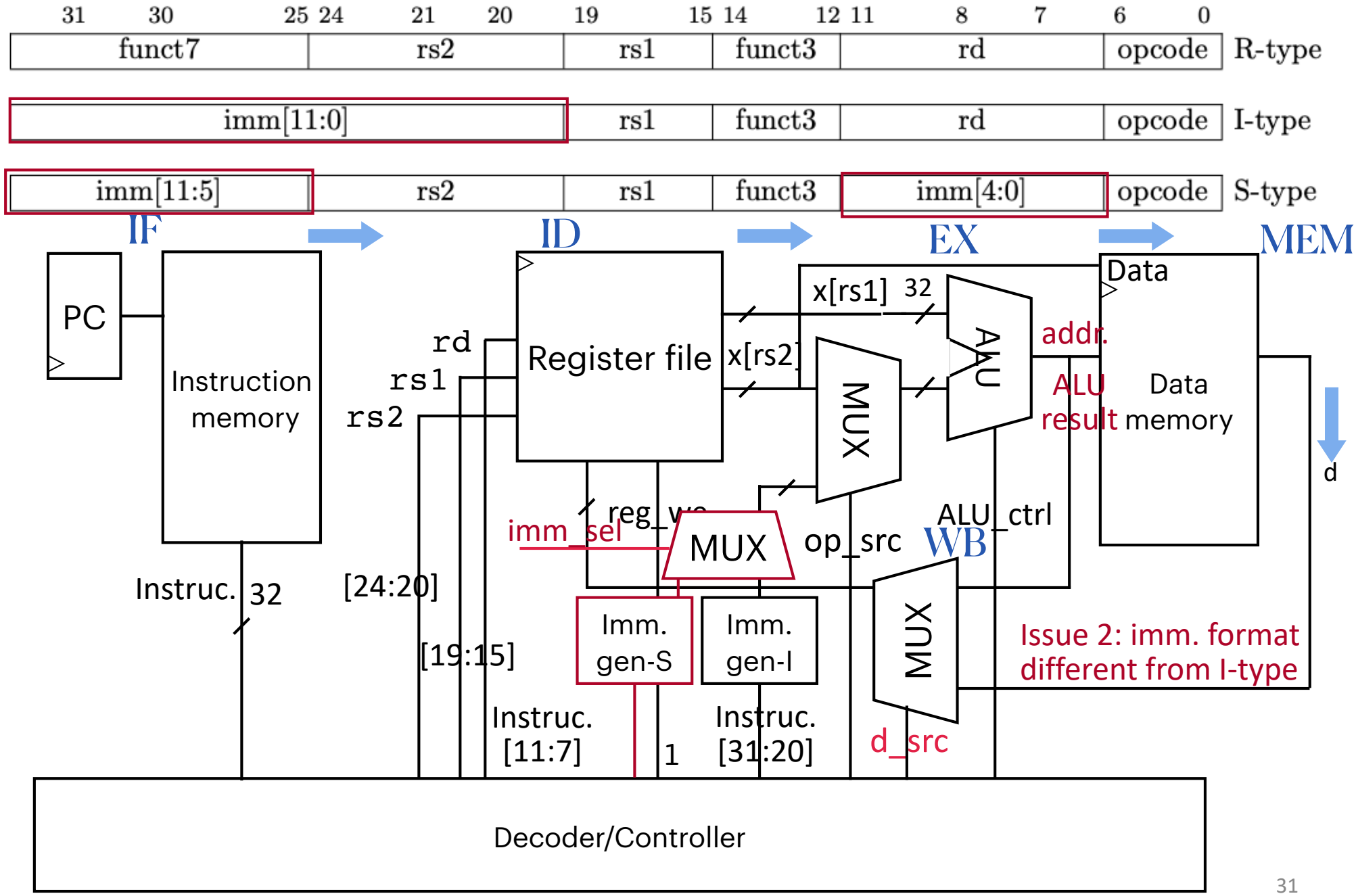
S-type Store


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
IF
ID
EX
MEM
PC
Instruction memory
Instruc.
32
[24:20]
[19:15]
rd
rs1
rs2
Register file
x[rs1]
32
x[rs2]
MUX
ALU
addr.
ALU result
Data
Data memory
d
reg_we
op_src
ALU_ctrl
WB
Imm. gen.
Instruc. [11:7]
1
Instruc. [31:20]
MUX
d_src
Issue 1: x[rs2] to data memory
Decoder/Controller

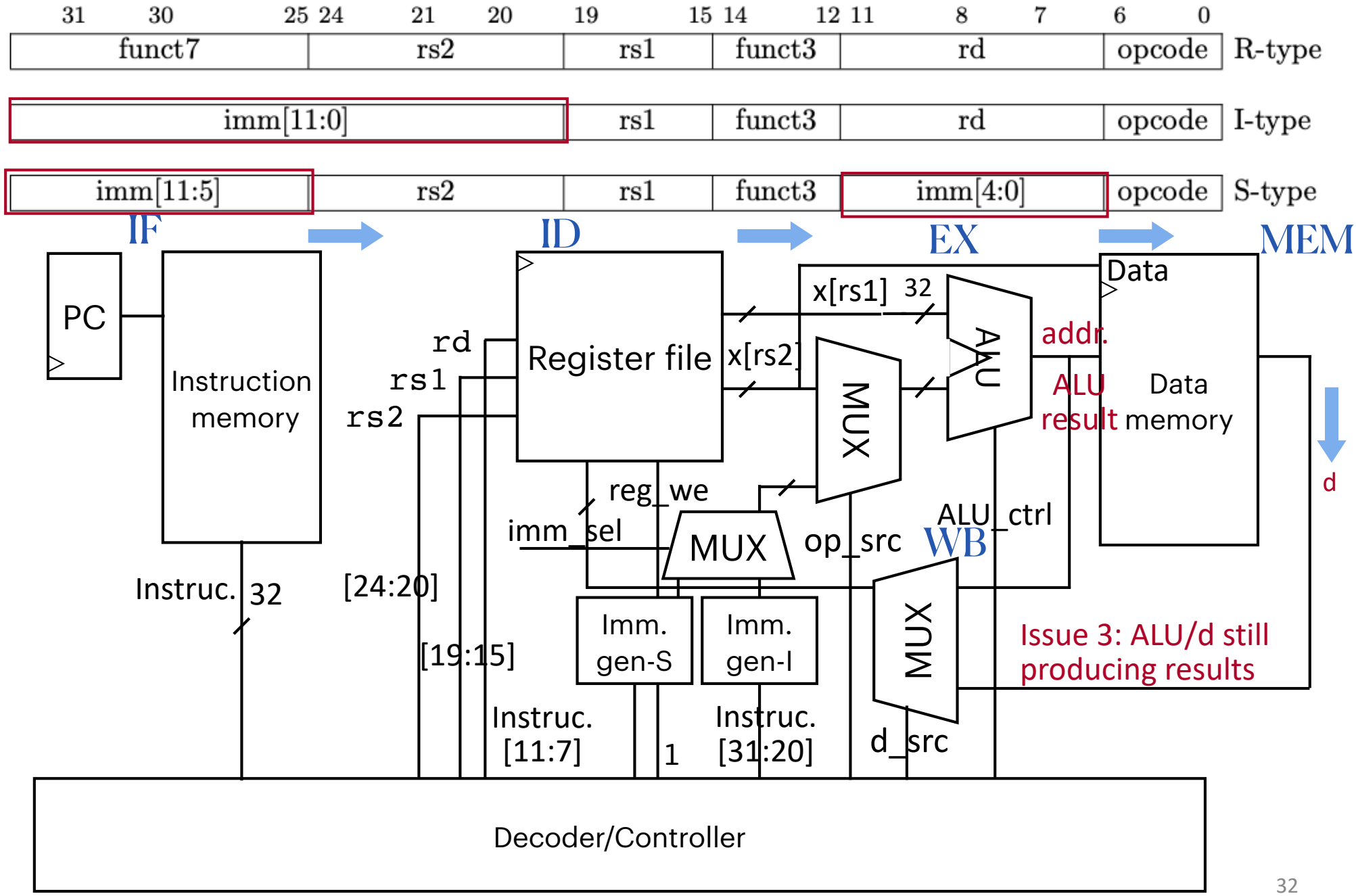
S-type Store


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
IF
ID
EX
MEM
PC
Instruction memory
Instruc. 32
rd
rs1
rs2
[24:20]
[19:15]
Register file
x[rs1] 32
x[rs2]
MUX
ALU
addr.
ALU result
Data memory
Data
d
reg_we
op_src
ALU_ctrl
WB
Imm. gen.
Instruc. [11:7]
1
Instruc. [31:20]
MUX
d_src
Issue 2: imm. format different from I-type
Decoder/Controller

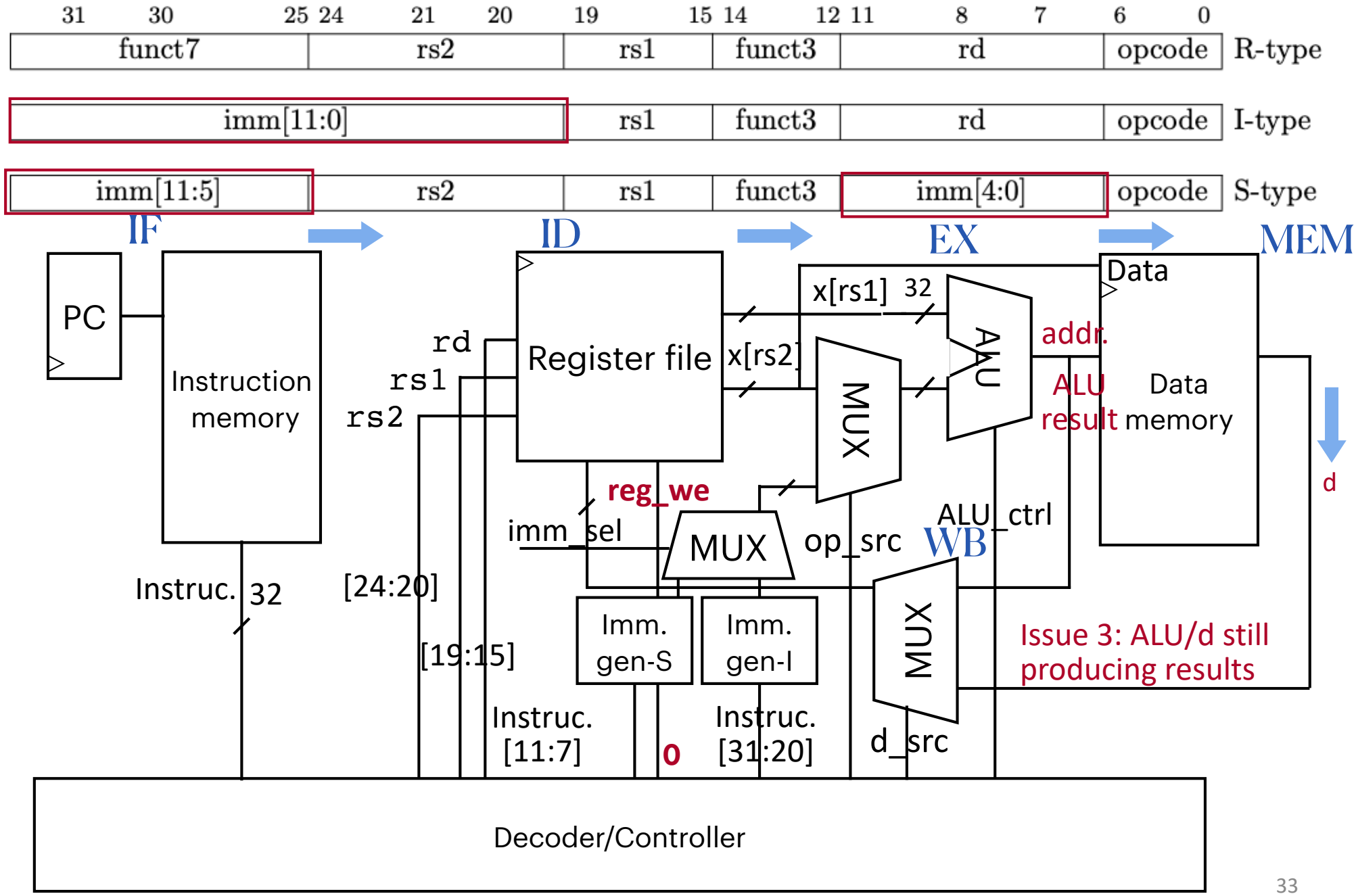
S-type Store


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
IF
ID
EX
MEM
PC
Instruction memory
Instruc. 32
[24:20]
[19:15]
rd
rs1
rs2
Register file
x[rs1] 32
x[rs2]
MUX
ALU
addr.
ALU result
Data
Data memory
d
reg_we
imm_sel
MUX
op_src
ALU_ctrl
WB
Imm. gen-S
Imm. gen-I
Instruc. [11:7]
1
Instruc. [31:20]
MUX
d_src
Issue 2: imm. format different from I-type
Decoder/Controller

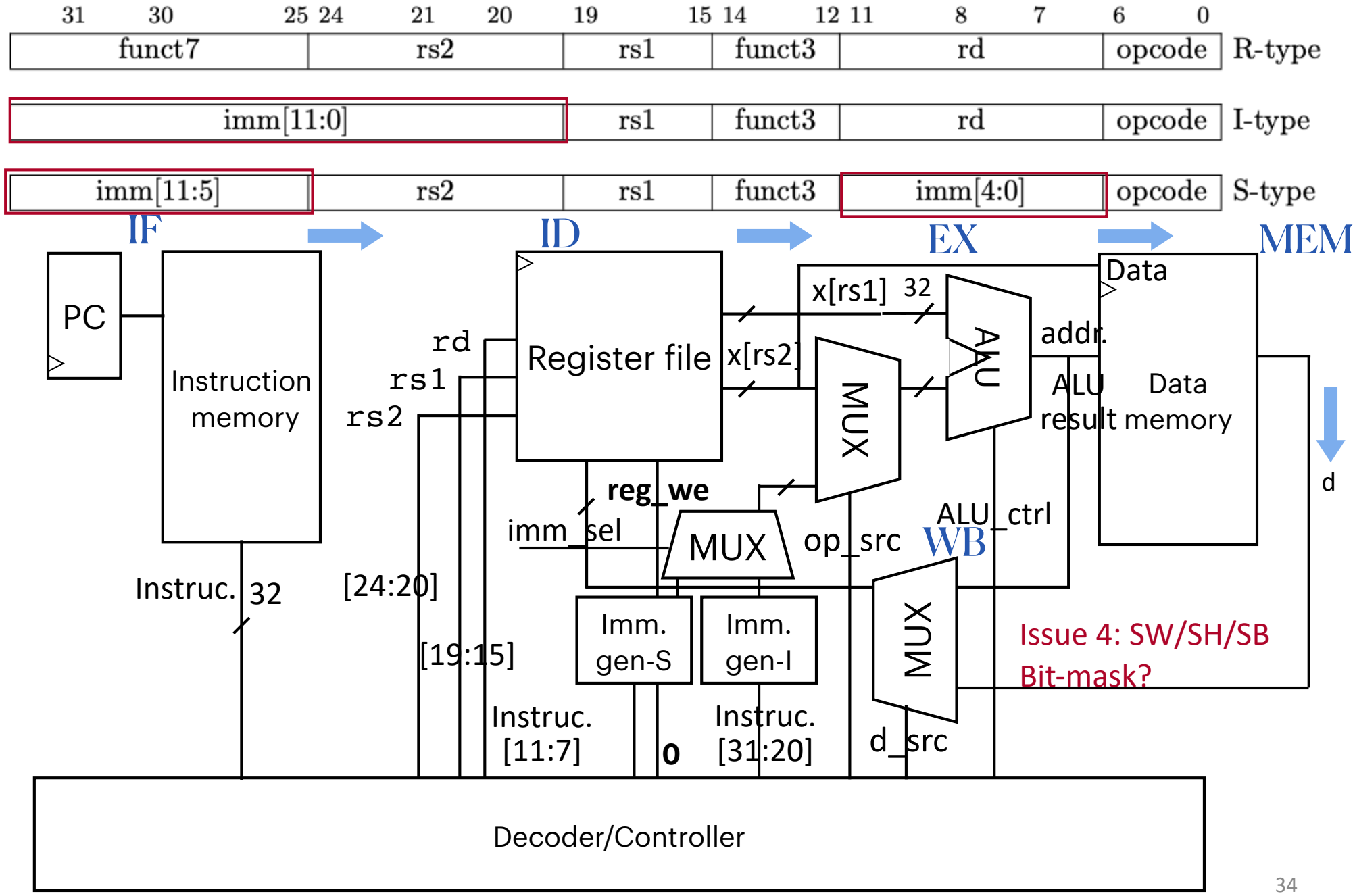
S-type Store


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
IF
ID
EX
MEM
PC
Instruction memory
Instruc. 32
[24:20]
[19:15]
rd
rs1
rs2
Register file
x[rs1] 32
x[rs2]
MUX
ALU
addr.
ALU result
Data
Data memory
d
reg_we
imm_sel
MUX
op_src
ALU_ctrl
WB
Imm. gen-S
Imm. gen-I
Instruc. [11:7]
1
Instruc. [31:20]
MUX
d_src
Issue 3: ALU/d still producing results
Decoder/Controller

S-type Store

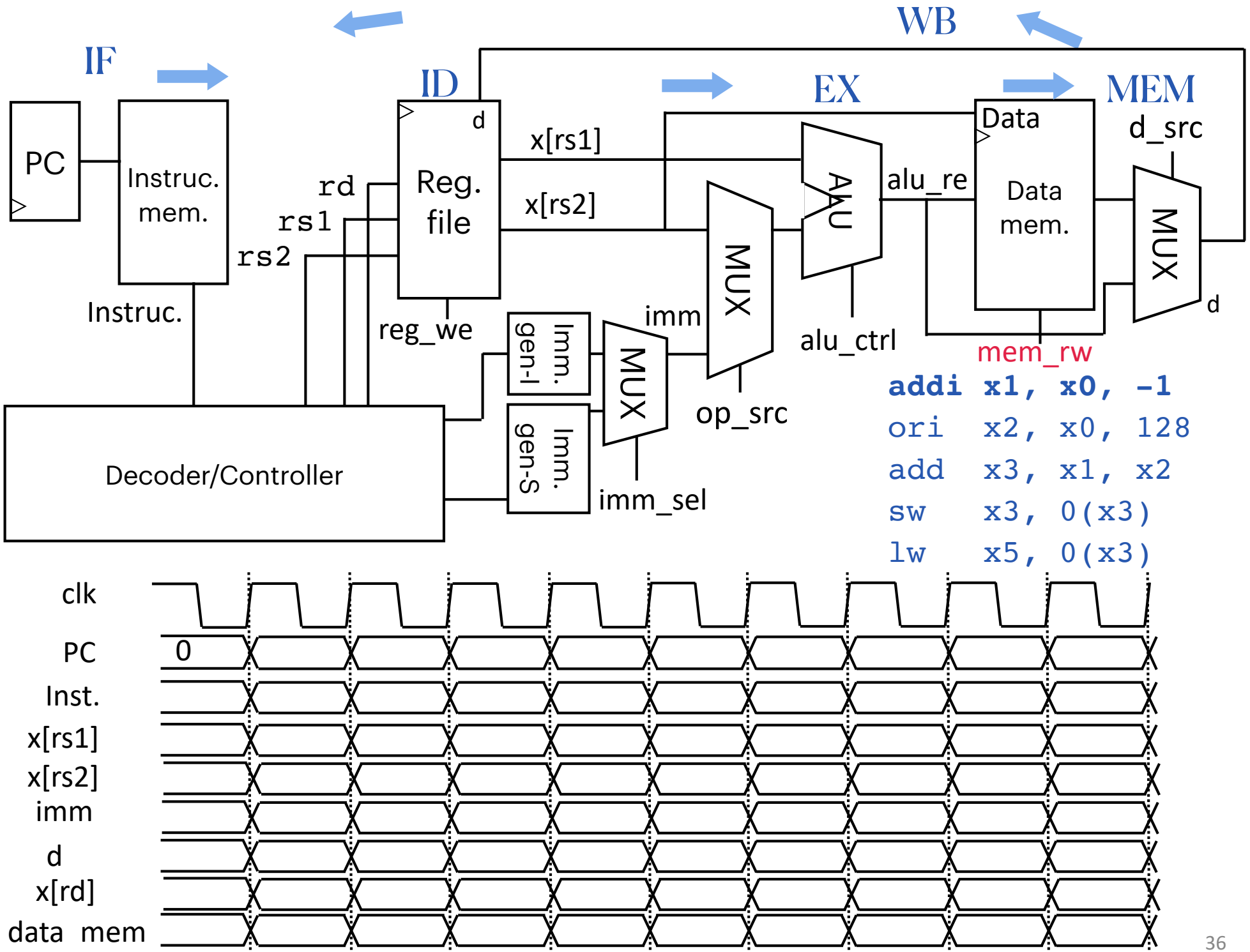

31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 rs2 rs1 funct3 rd opcode R-type
imm[11:0] rs1 funct3 rd opcode I-type
imm[11:5] rs2 rs1 funct3 imm[4:0] opcode S-type
IF
ID
EX
MEM
PC
Instruction memory
Instruc. 32
[24:20]
[19:15]
rd
rs1
rs2
Register file
x[rs1] 32
x[rs2]
MUX
ALU
addr.
ALU result
Data
Data memory
d
reg_we
imm_sel
MUX
op_src
ALU_ctrl
WB
Imm. gen-S
Imm. gen-I
Instruc. [11:7]
0
Instruc. [31:20]
MUX
d_src
Issue 3: ALU/d still producing results
Decoder/Controller

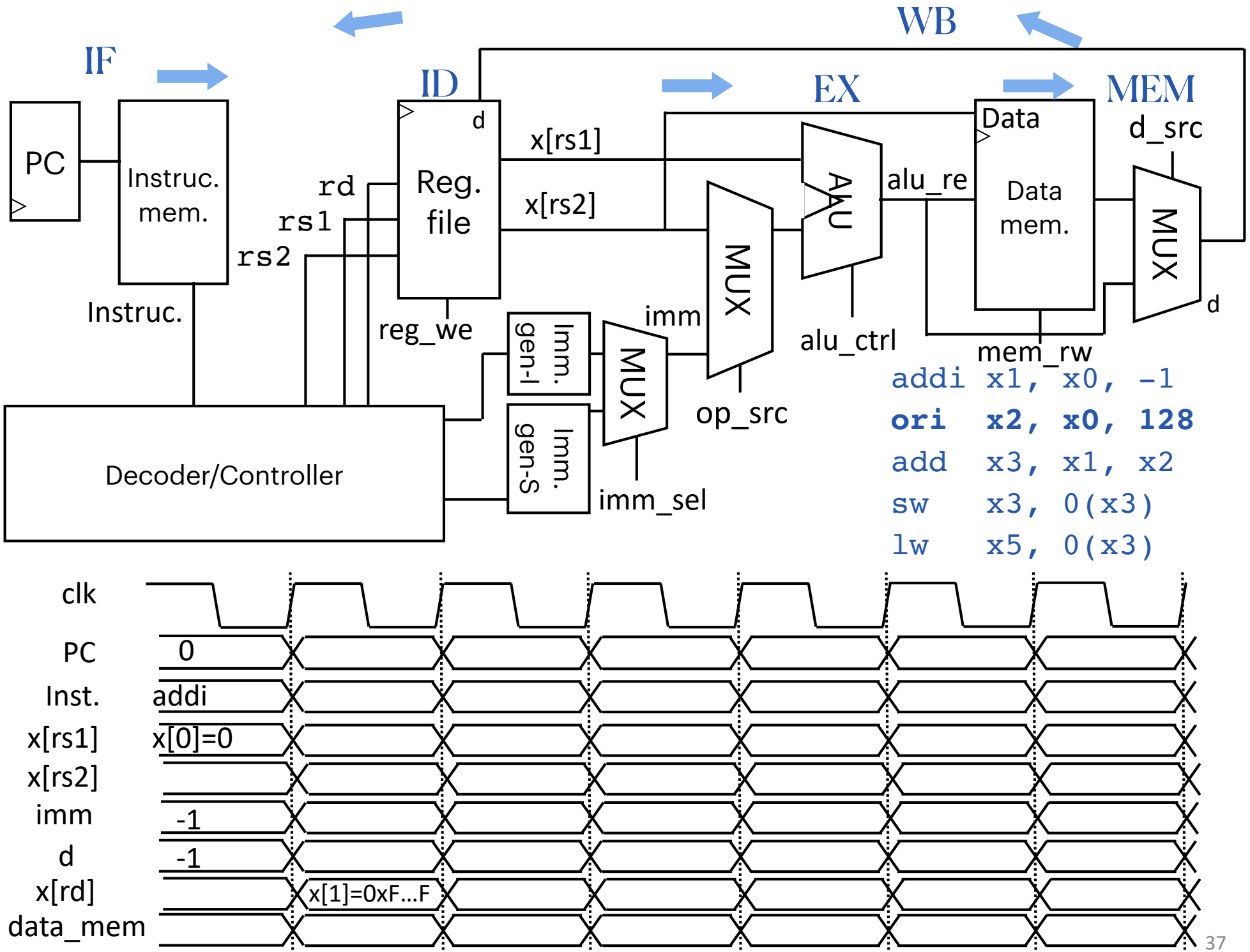
S-type Store

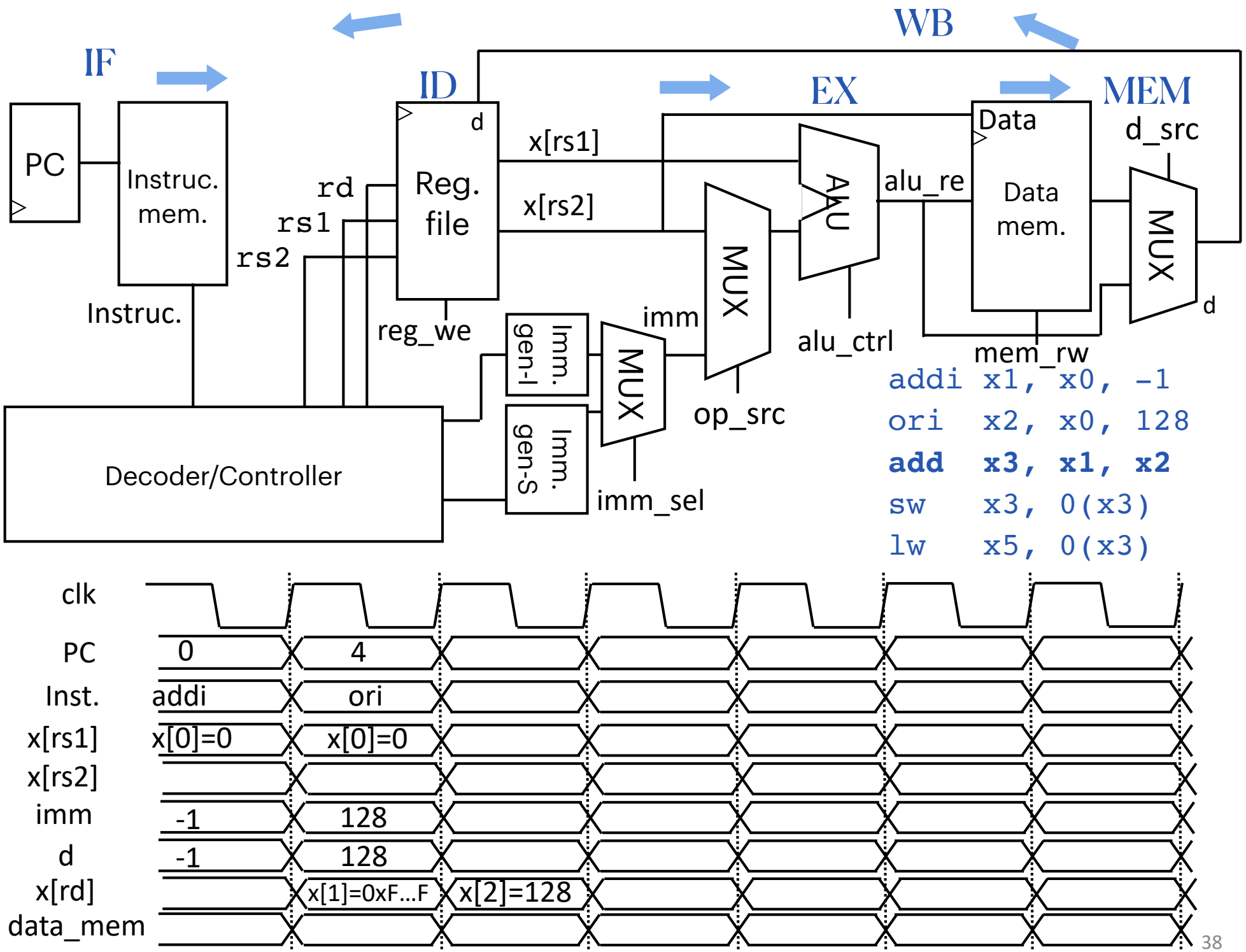

31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 rs2 rs1 funct3 rd opcode R-type
imm[11:0] rs1 funct3 rd opcode I-type
imm[11:5] rs2 rs1 funct3 imm[4:0] opcode S-type
IF
ID
EX
MEM
PC
Instruction memory
Instruc. 32
[24:20]
[19:15]
rd
rs1
rs2
Register file
x[rs1] 32
x[rs2]
MUX
ALU
addr.
ALU result
Data memory
Data
d
reg_we
imm_sel
MUX
op_src
ALU_ctrl
WB
Imm. gen-S
Imm. gen-I
Instruc. [11:7]
0
Instruc. [31:20]
MUX
d_src
Issue 4: SW/SH/SB Bit-mask?
Decoder/Controller

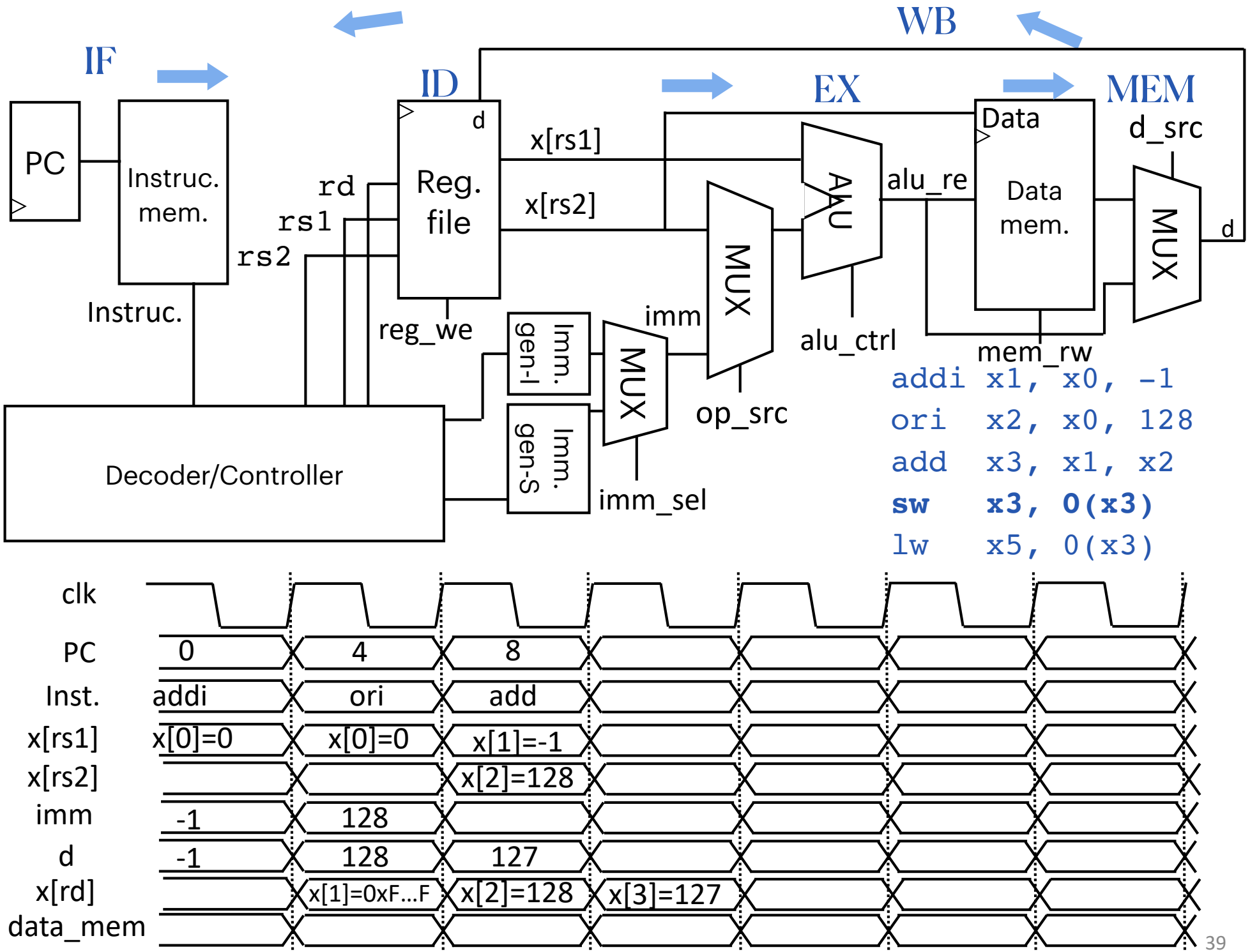
Take a Break!

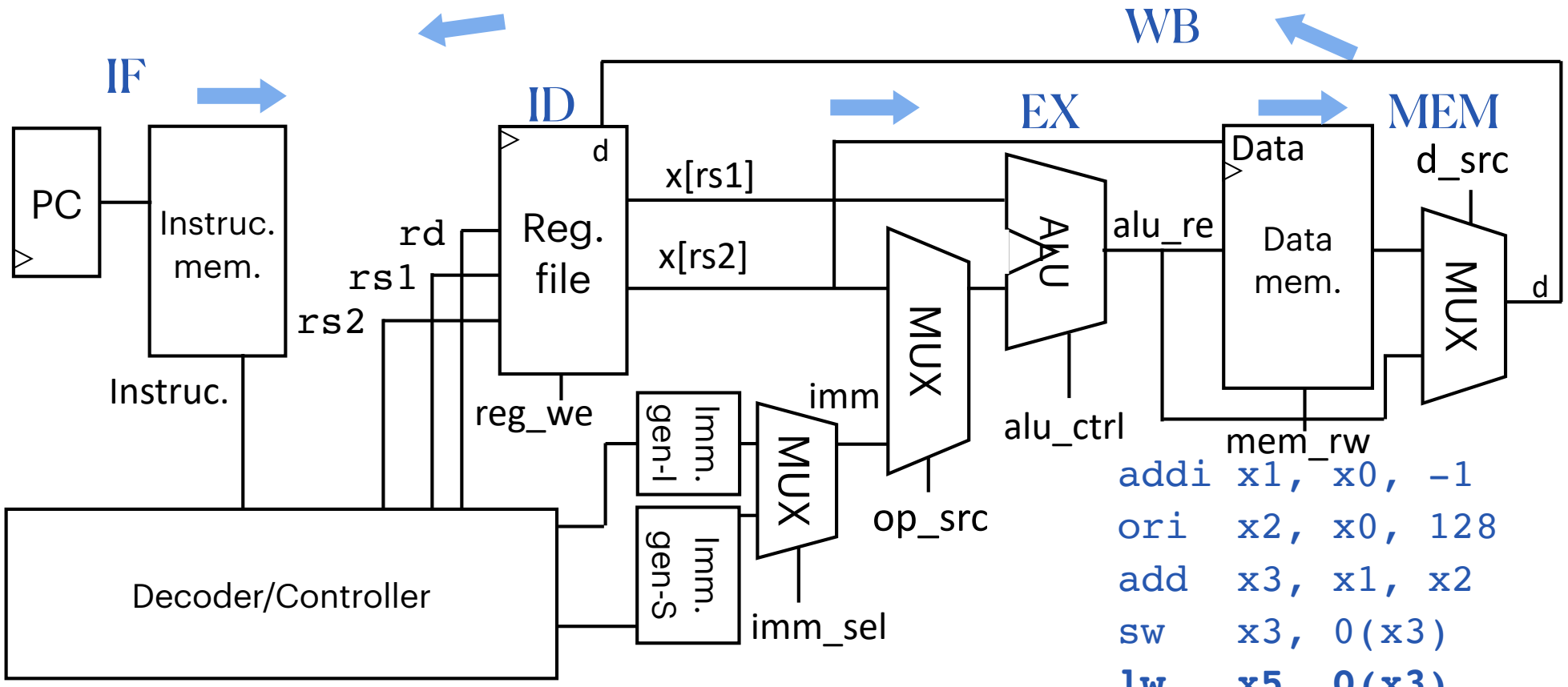
	IF	ID	EX	MEM	WB
R-type					
I-type arith. & logic					
I-type load					
S-type store					


IF
ID
EX
MEM
WB
PC
Instruc. mem.
Instruc.
rd
rs1
rs2
Reg. file
d
reg_we
x[rs1]
x[rs2]
Imm. gen-I
Imm. gen-S
MUX
imm
imm_sel
MUX
op_src
AAU
alu_re
alu_ctrl
Data
Data mem.
mem_rw
d_src
MUX
d
Decoder/Controller
addi x1, x0, -1
ori x2, x0, 128
add x3, x1, x2
sw x3, 0(x3)
lw x5, 0(x3)
clk
PC
0
Inst.
x[rs1]
x[rs2]
imm
d
x[rd]
data mem

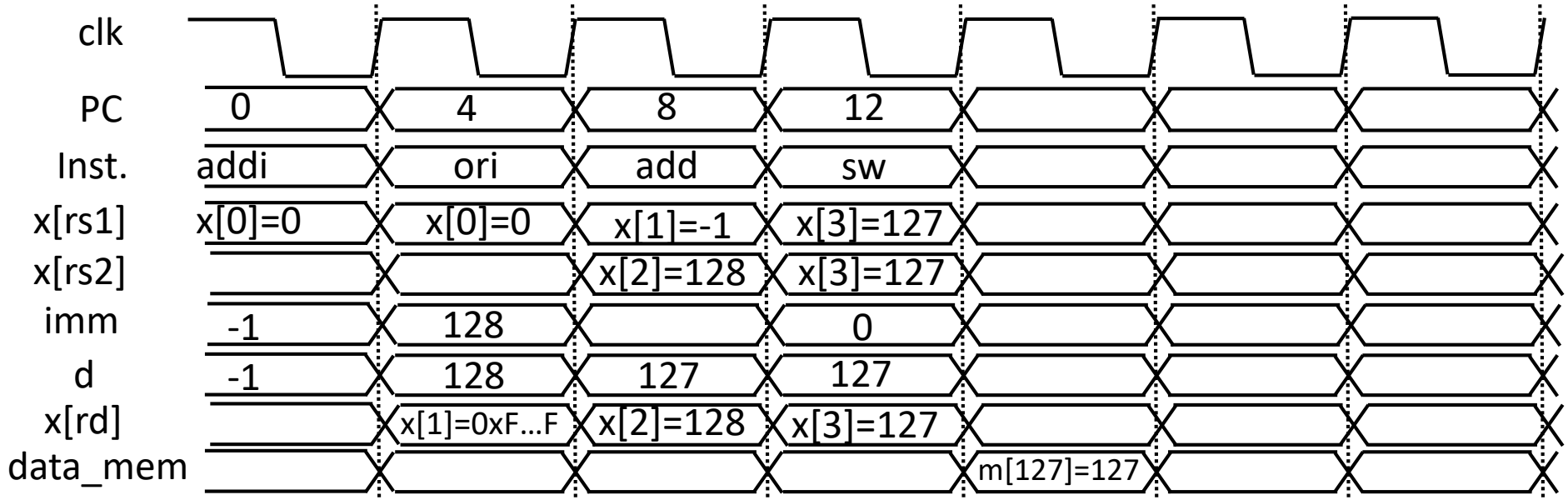

IF
ID
EX
MEM
WB
PC
Instruc. mem.
Instruc.
rd
rs1
rs2
Reg. file
d
reg_we
x[rs1]
x[rs2]
Imm. gen-I
Imm. gen-S
MUX
imm
imm_sel
MUX
op_src
AAU
alu_re
alu_ctrl
Data
Data mem.
mem_rw
d_src
MUX
d
Decoder/Controller
addi x1, x0, -1
ori x2, x0, 128
add x3, x1, x2
sw x3, 0(x3)
lw x5, 0(x3)
clk
PC
0
Inst.
addi
x[rs1]
x[0]=0
x[rs2]
imm
-1
d
-1
x[rd]
x[1]=0xF...F
data_mem

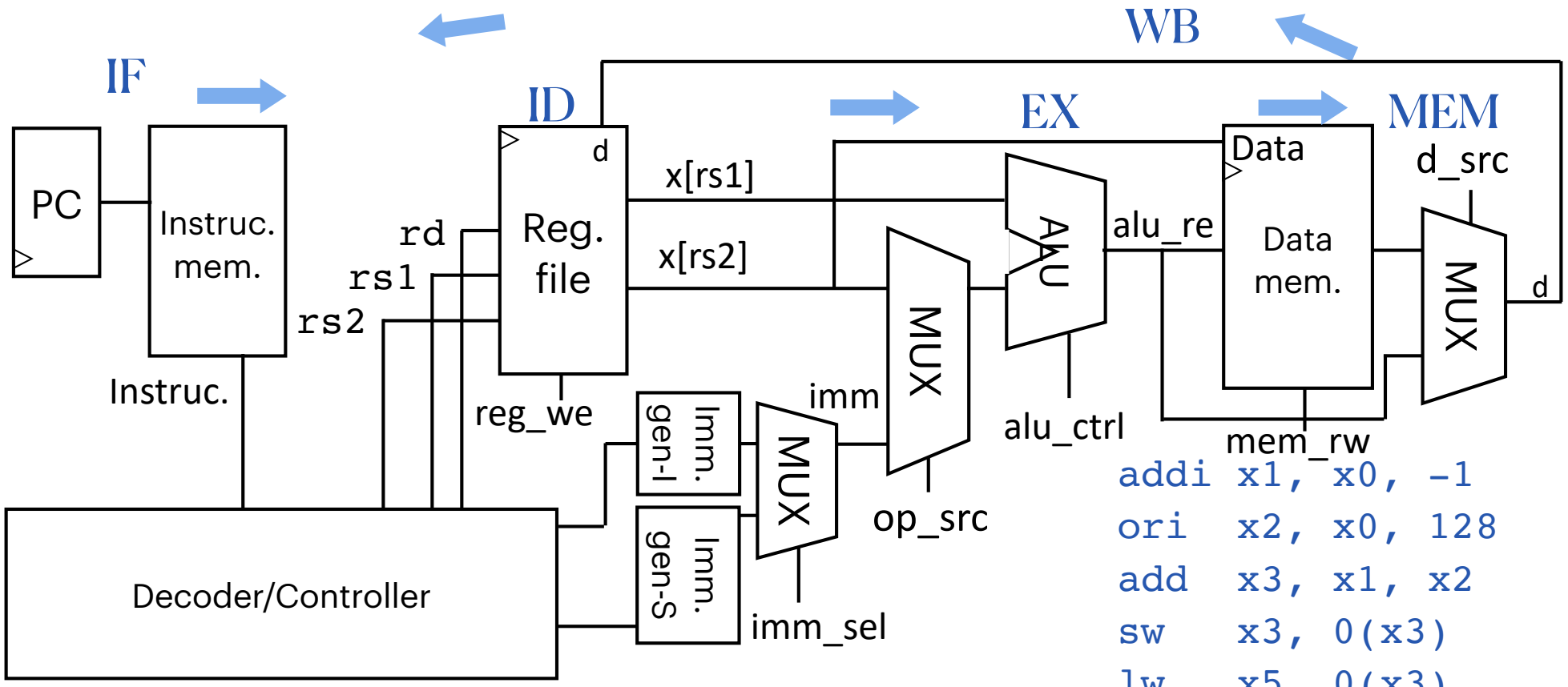

IF
ID
EX
MEM
WB
PC
Instruc. mem.
Instruc.
rd
rs1
rs2
Reg. file
d
reg_we
x[rs1]
x[rs2]
Imm. gen-I
Imm. gen-S
MUX
imm
imm_sel
MUX
op_src
AAU
alu_ctrl
alu_re
Data
Data mem.
mem_rw
d_src
MUX
d
Decoder/Controller
addi x1, x0, -1
ori x2, x0, 128
add x3, x1, x2
sw x3, 0(x3)
lw x5, 0(x3)
clk
PC
Inst.
x[rs1]
x[rs2]
imm
d
x[rd]
data_mem
0
4
addi
ori
x[0]=0
x[0]=0
-1
128
-1
128
x[1]=0xF...F
x[2]=128


IF
ID
EX
MEM
WB
PC
Instruc. mem.
Instruc.
rd
rs1
rs2
Reg. file
d
reg_we
x[rs1]
x[rs2]
Imm. gen-I
Imm. gen-S
MUX
imm
imm_sel
MUX
op_src
AAU
alu_ctrl
alu_re
Data
Data mem.
mem_rw
d_src
MUX
d
Decoder/Controller
addi x1, x0, -1
ori x2, x0, 128
add x3, x1, x2
sw x3, 0(x3)
lw x5, 0(x3)
clk
PC
Inst.
x[rs1]
x[rs2]
imm
d
x[rd]
data_mem
0
4
8
addi
ori
add
x[0]=0
x[0]=0
x[1]=-1
x[2]=128
-1
128
-1
128
127
x[1]=0xF...F
x[2]=128
x[3]=127

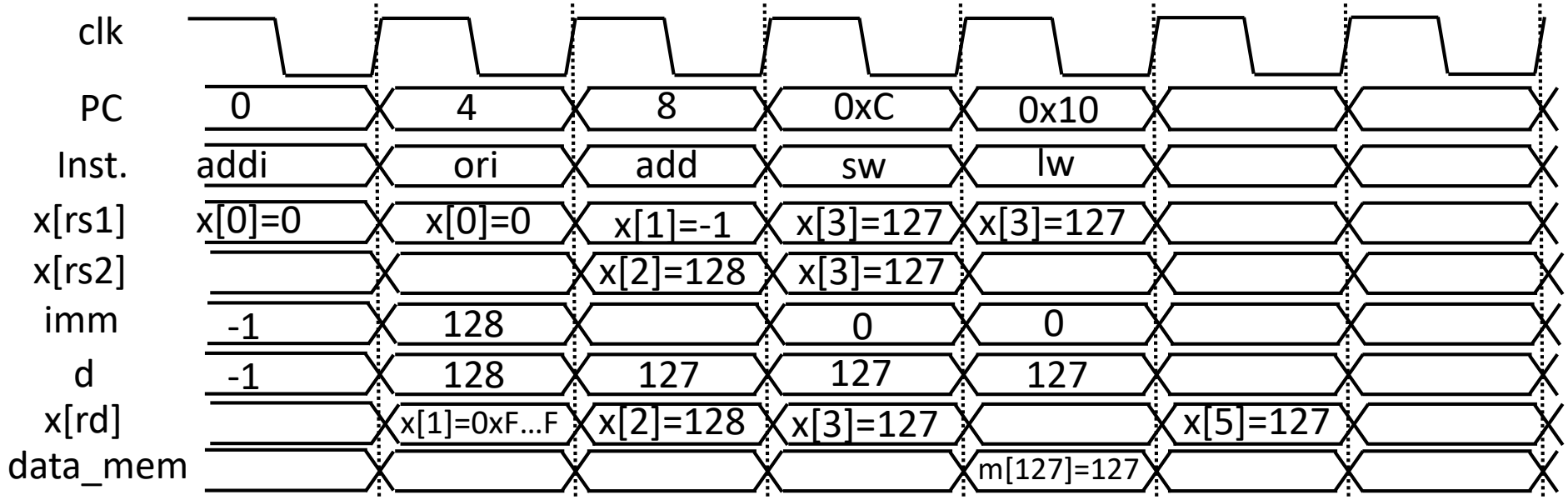



```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
```

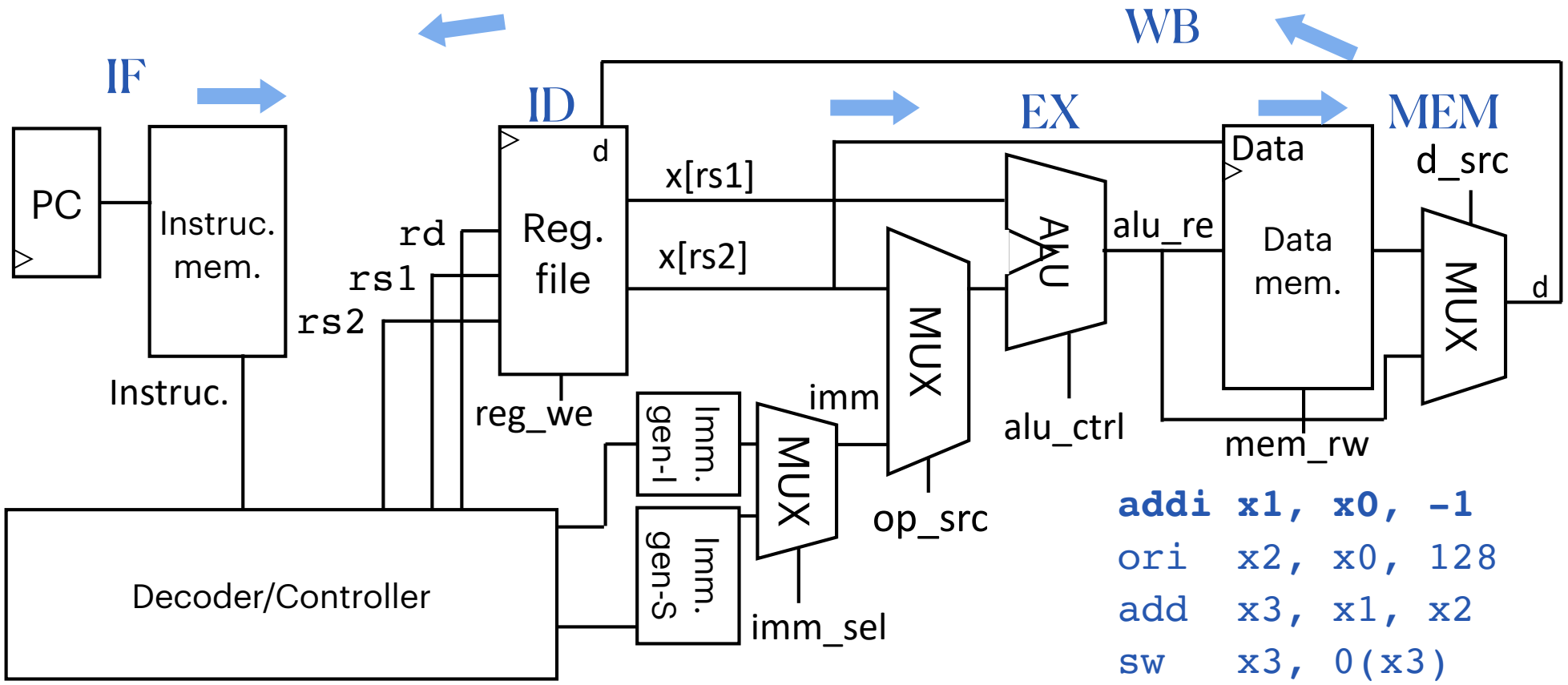





```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
```

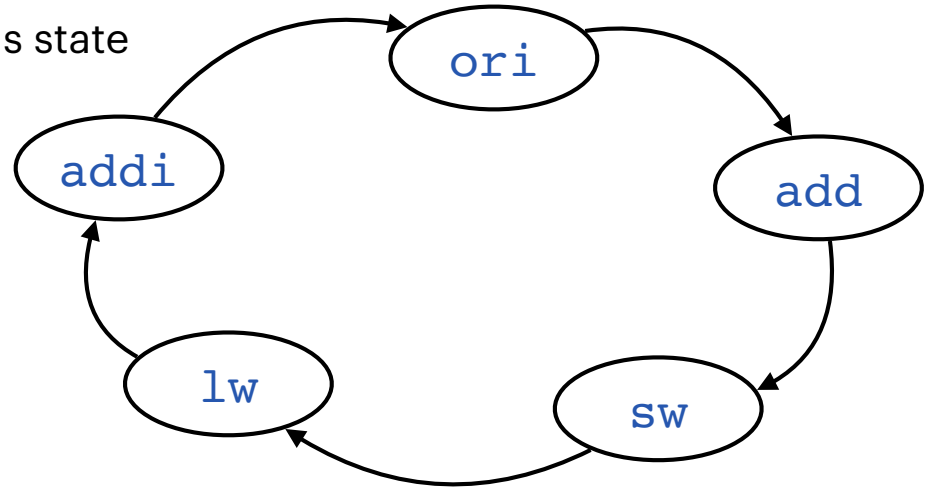


Controller as FSM




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

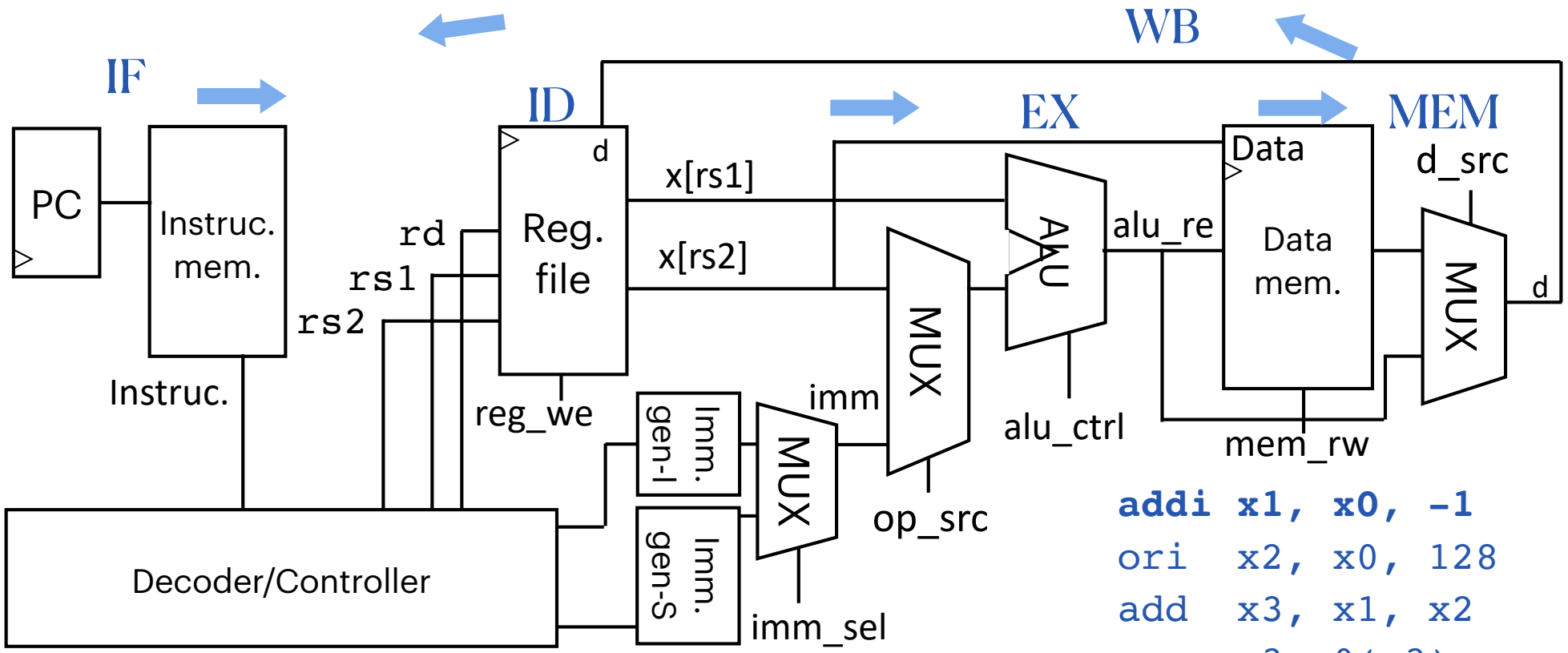
Instruction as state



Next state =

Output = control signals

Controller as FSM



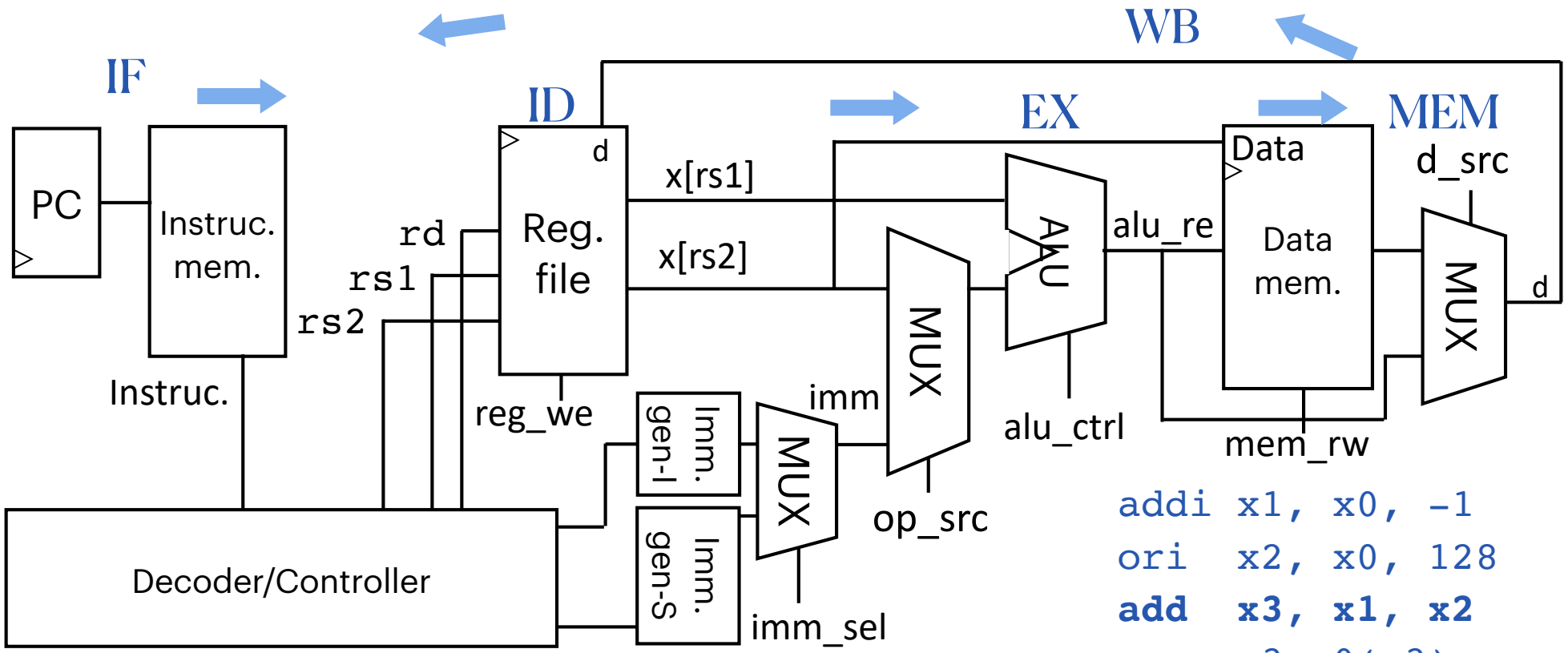

```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
```

	addi				
reg_we					
mem_rw					
alu_ctrl					
imm_sel					
d_src					
op_src					

Next state =

Output = control signals

Controller as FSM



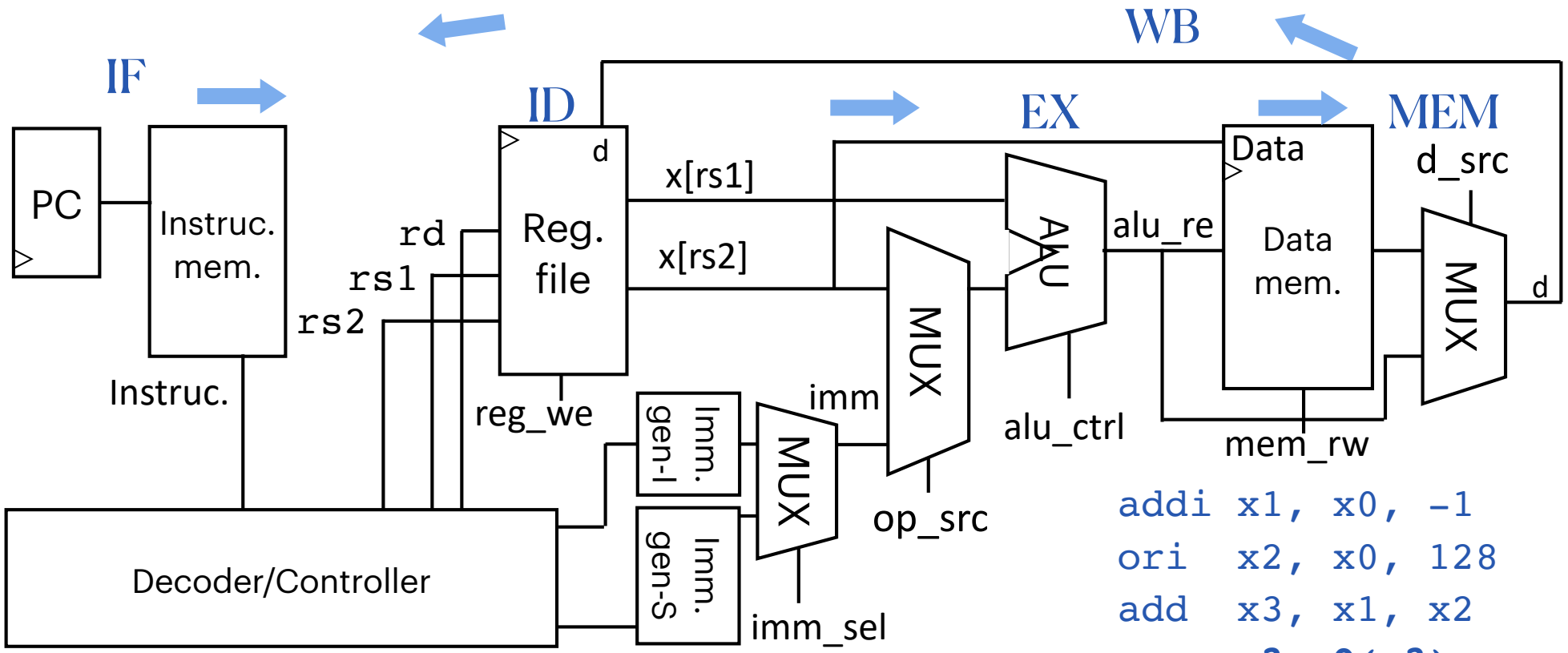

```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
```

	addi	add			
reg_we	1				
mem_rw	R				
alu_ctrl	add				
imm_sel	I				
d_src	alu				
op_src	imm				

Next state =

Output = control signals

Controller as FSM



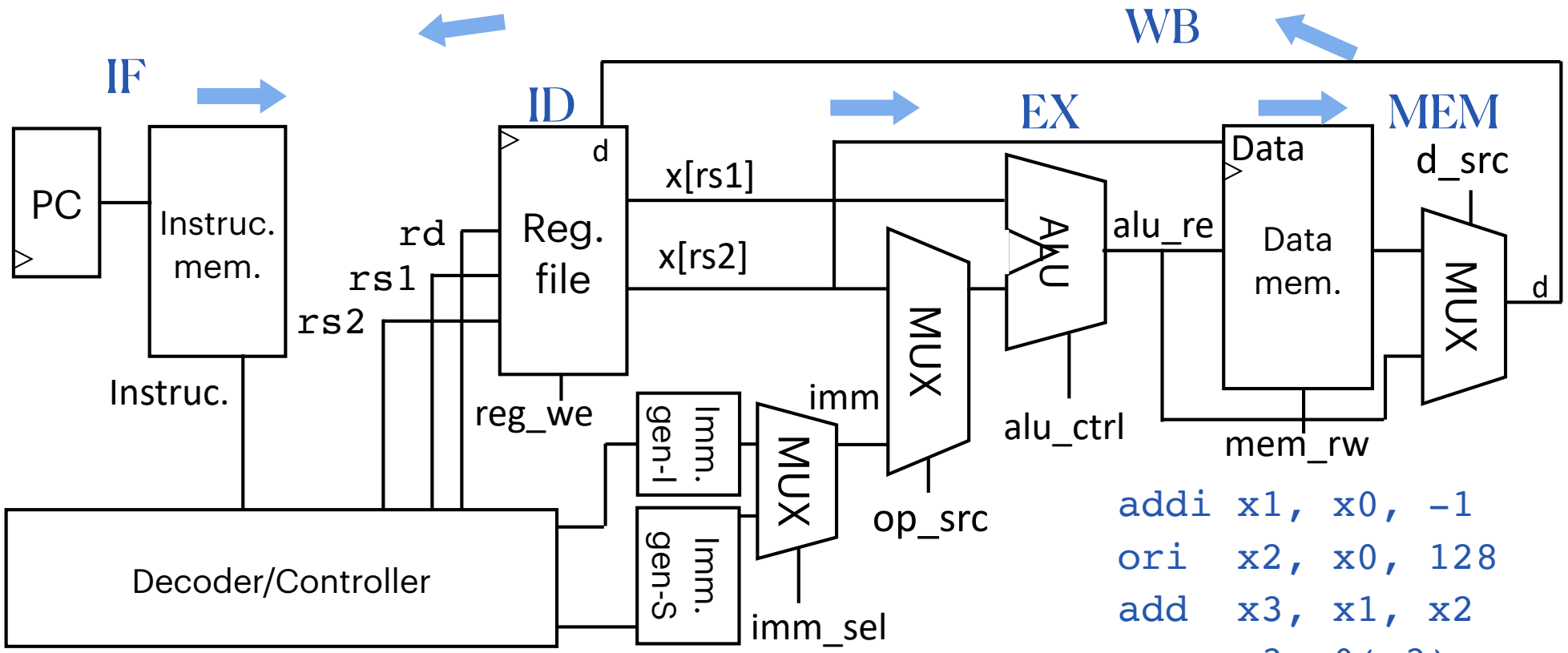

```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
```

	addi	add	sw	
reg_we	1	1		
mem_rw	R	R		
alu_ctrl	add	add		
imm_sel	I	I		
d_src	alu	alu		
op_src	imm	reg		

Next state =

Output = control signals

Controller as FSM



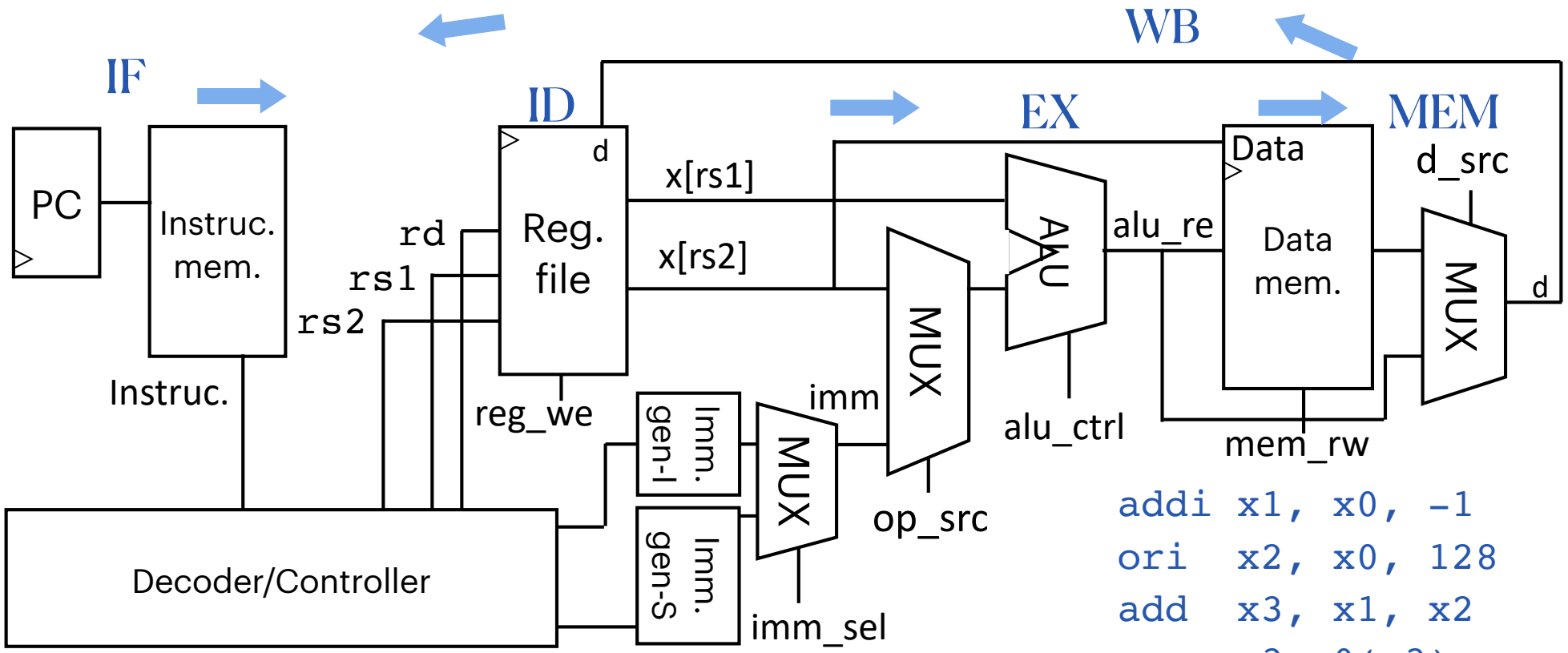

```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
```

	addi	add	sw	lw
reg_we	1	1	0	
mem_rw	R	R	W	
alu_ctrl	add	add	add	
imm_sel	I	I	S	
d_src	alu	alu	*	
op_src	imm	reg	imm	

Next state =

Output = control signals

Controller as FSM




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

	addi	add	sw	lw
reg_we	1	1	0	1
mem_rw	R	R	W	R
alu_ctrl	add	add	add	add
imm_sel	I	I	S	I
d_src	alu	alu	*	mem
op_src	imm	reg	imm	imm

Next state =

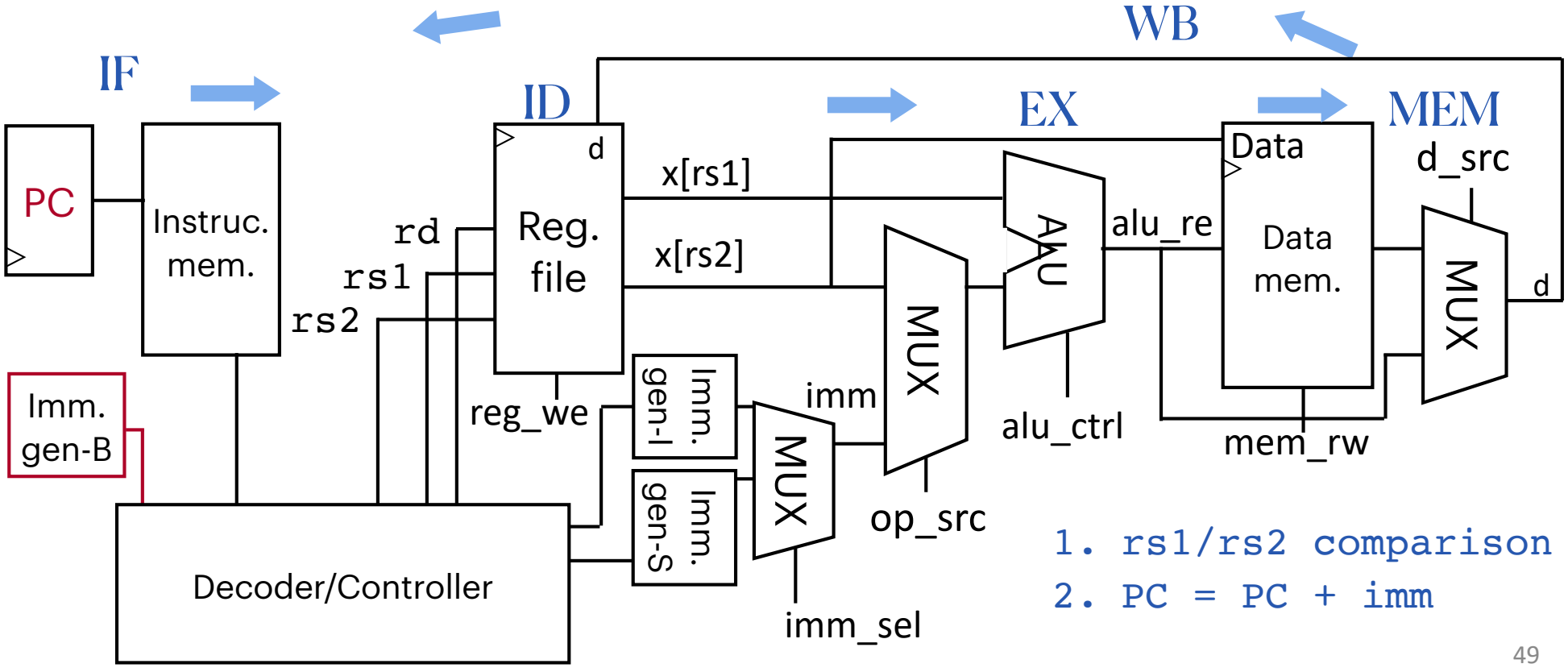
Output = control signals

Continue with Datapath

— Decision Making Instructions

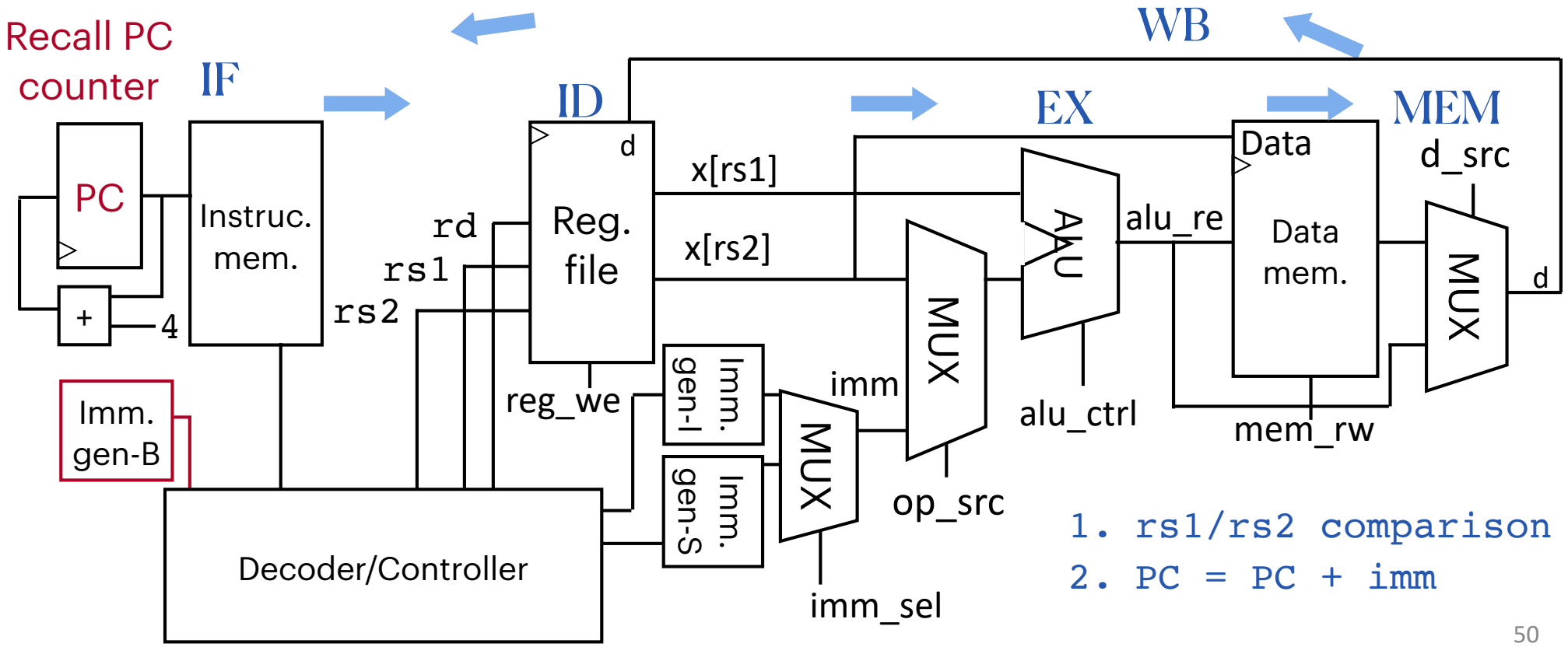
B-type Branch

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



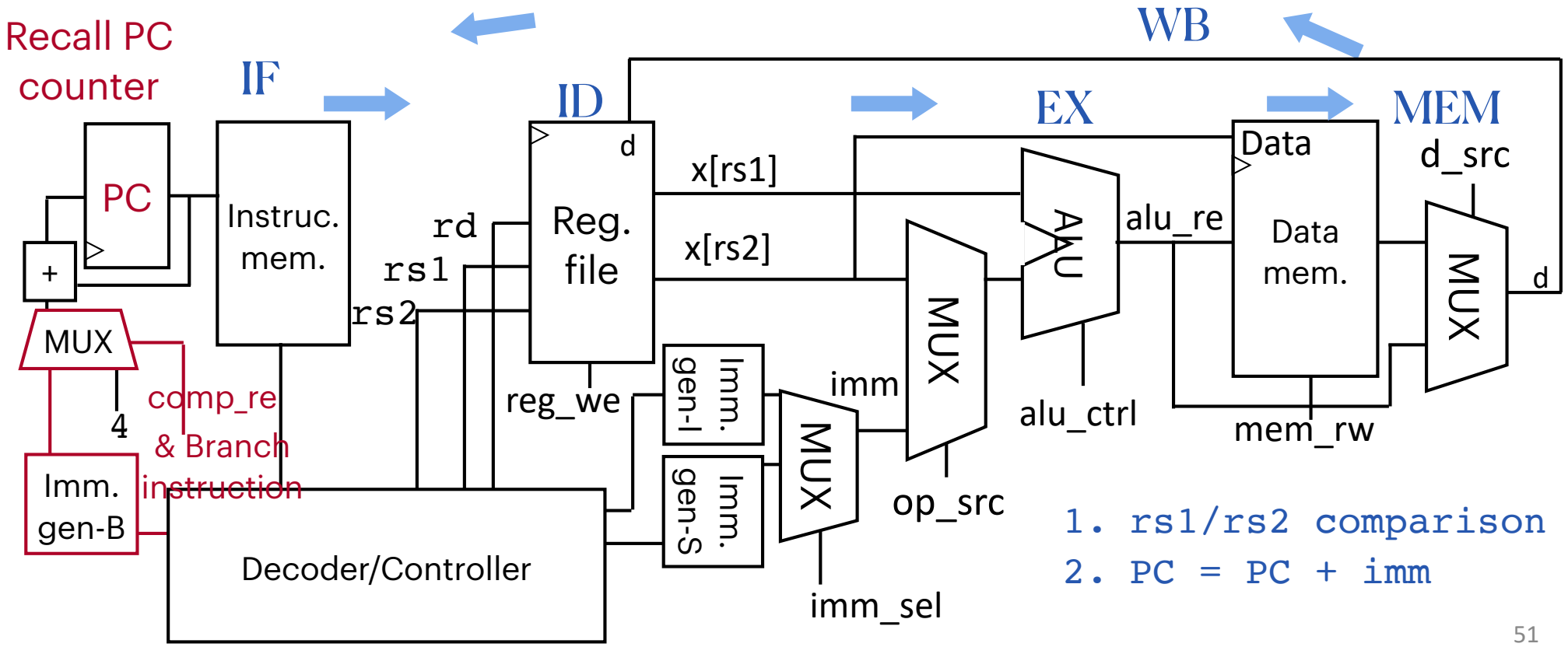
B-type Branch

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

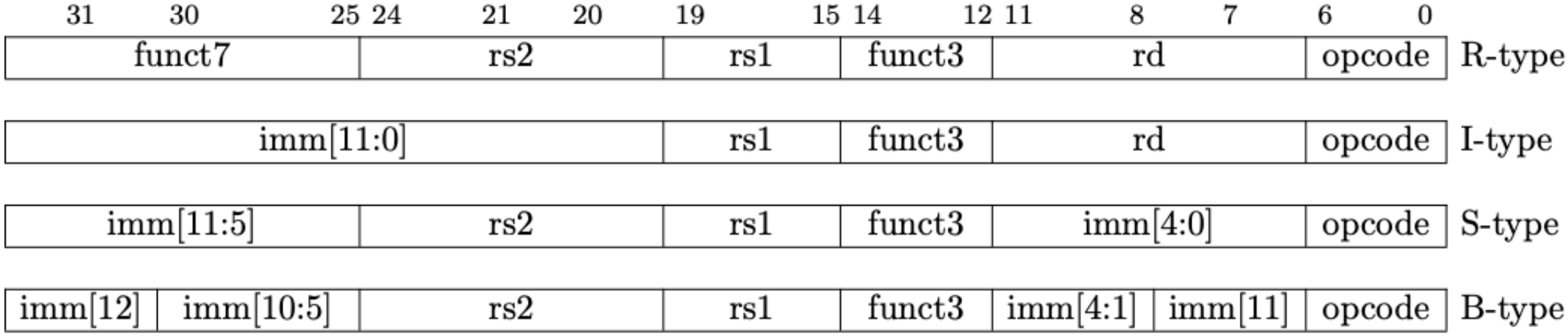


B-type Branch

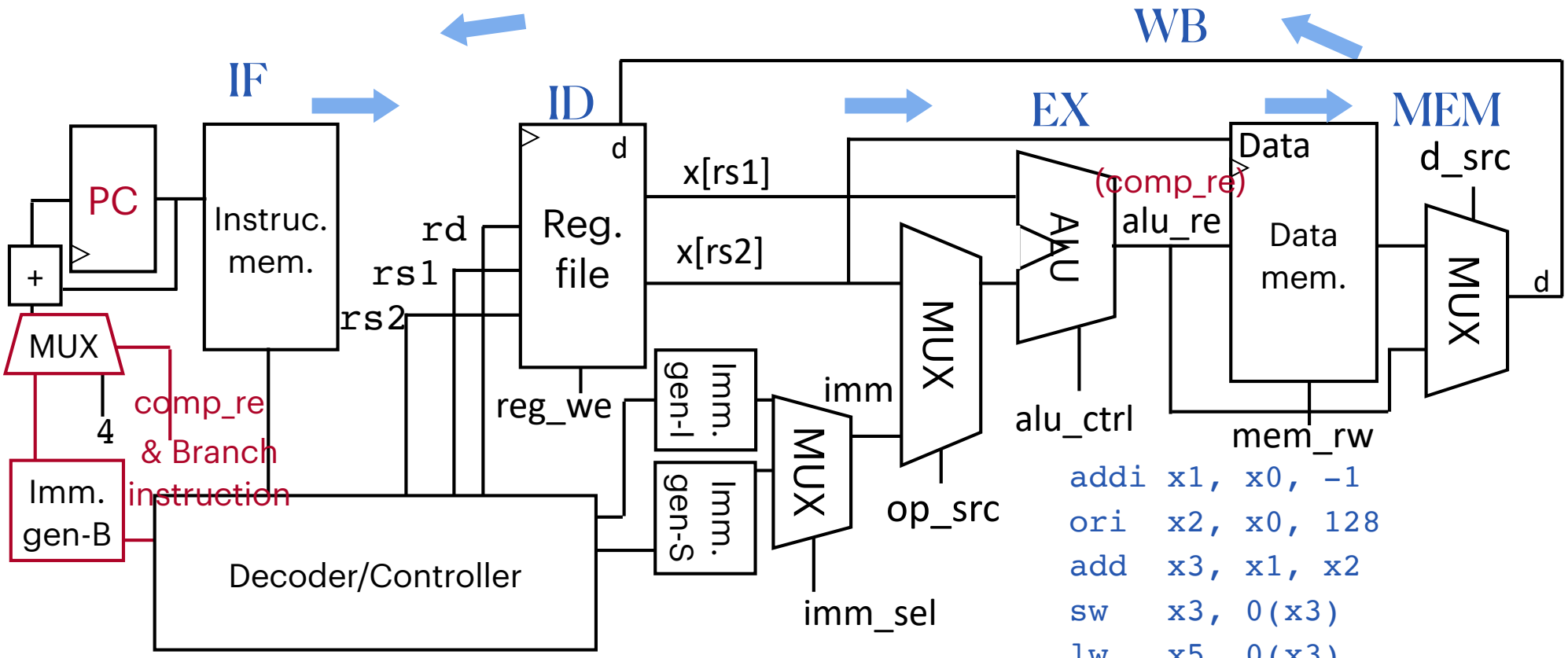
31	30 25	24 21	20 19	15 14 12	11 8	7 6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type



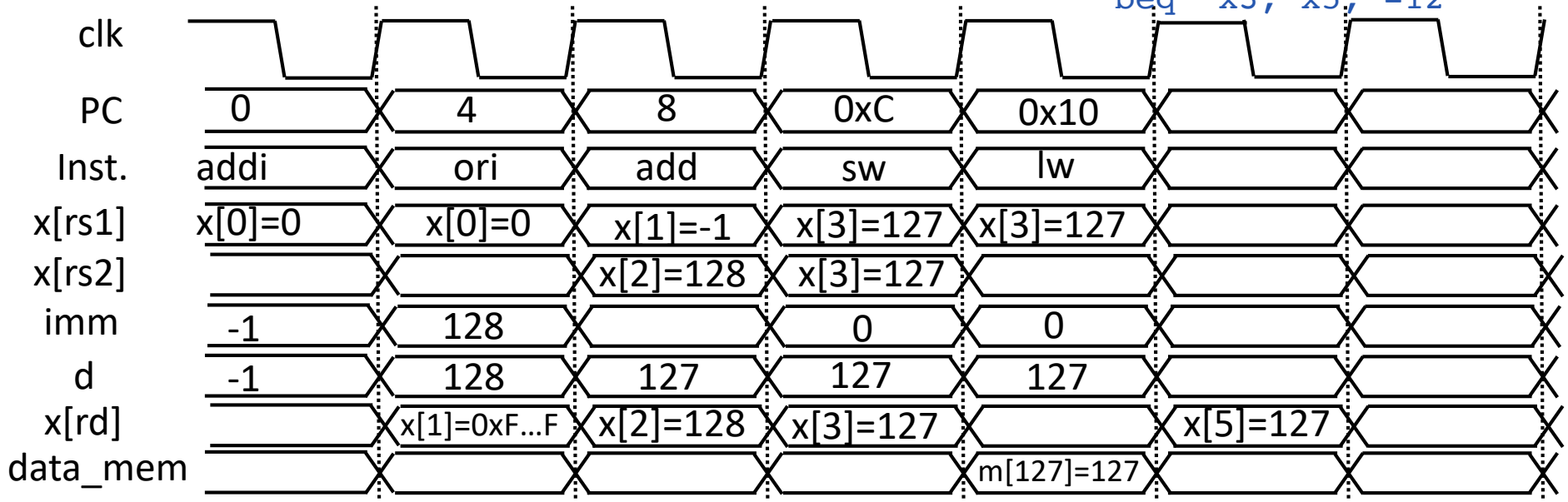
B-type Branch

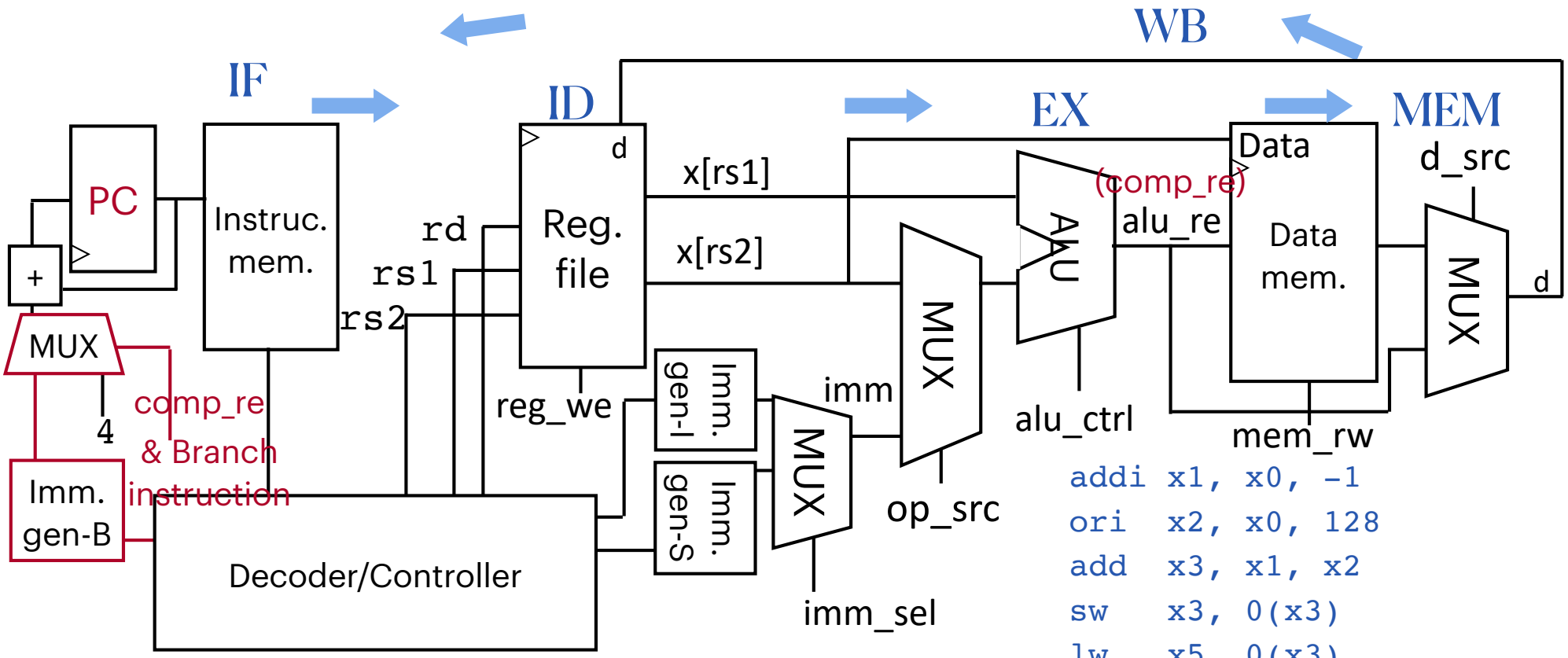


- Implementation of comparison
  - Comparison <=> subtraction, can reuse the hardware of add/sub to generate comparison result

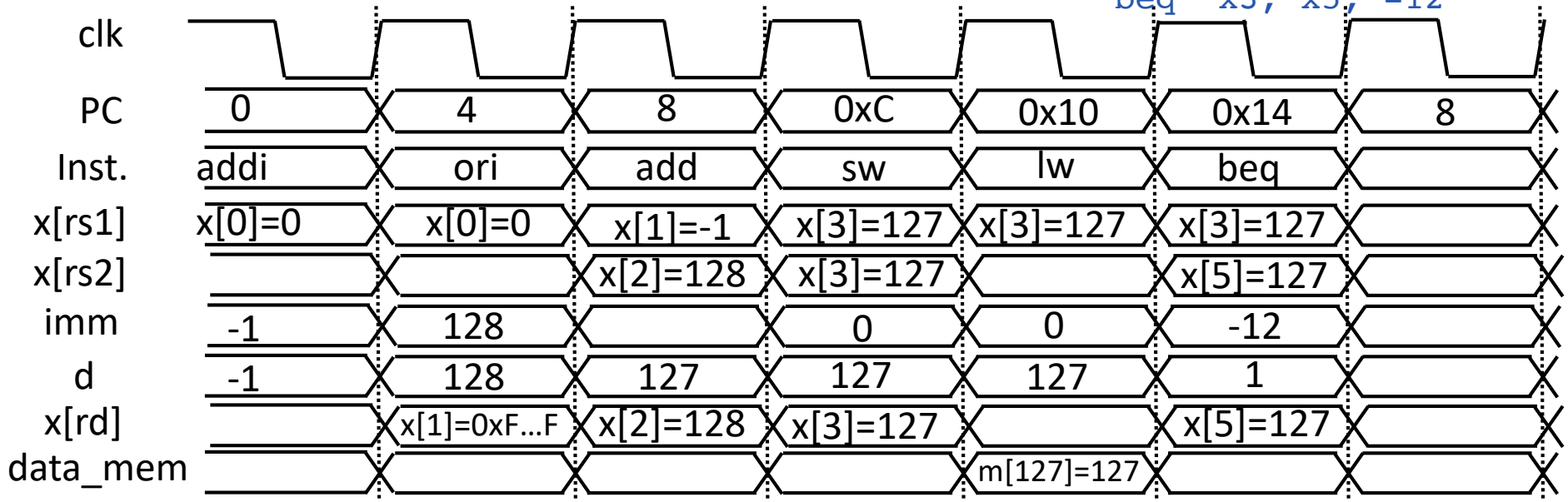


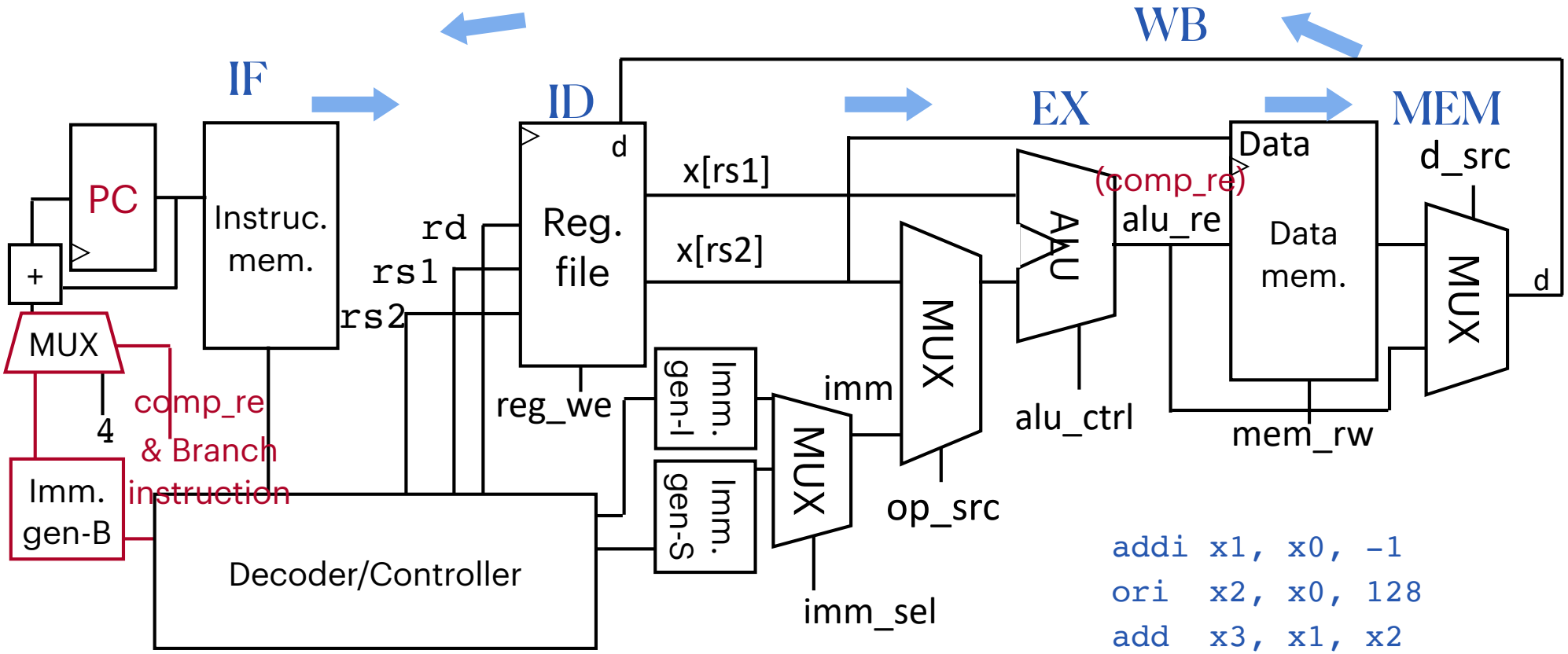

```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
beq  x3, x5, -12
```






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

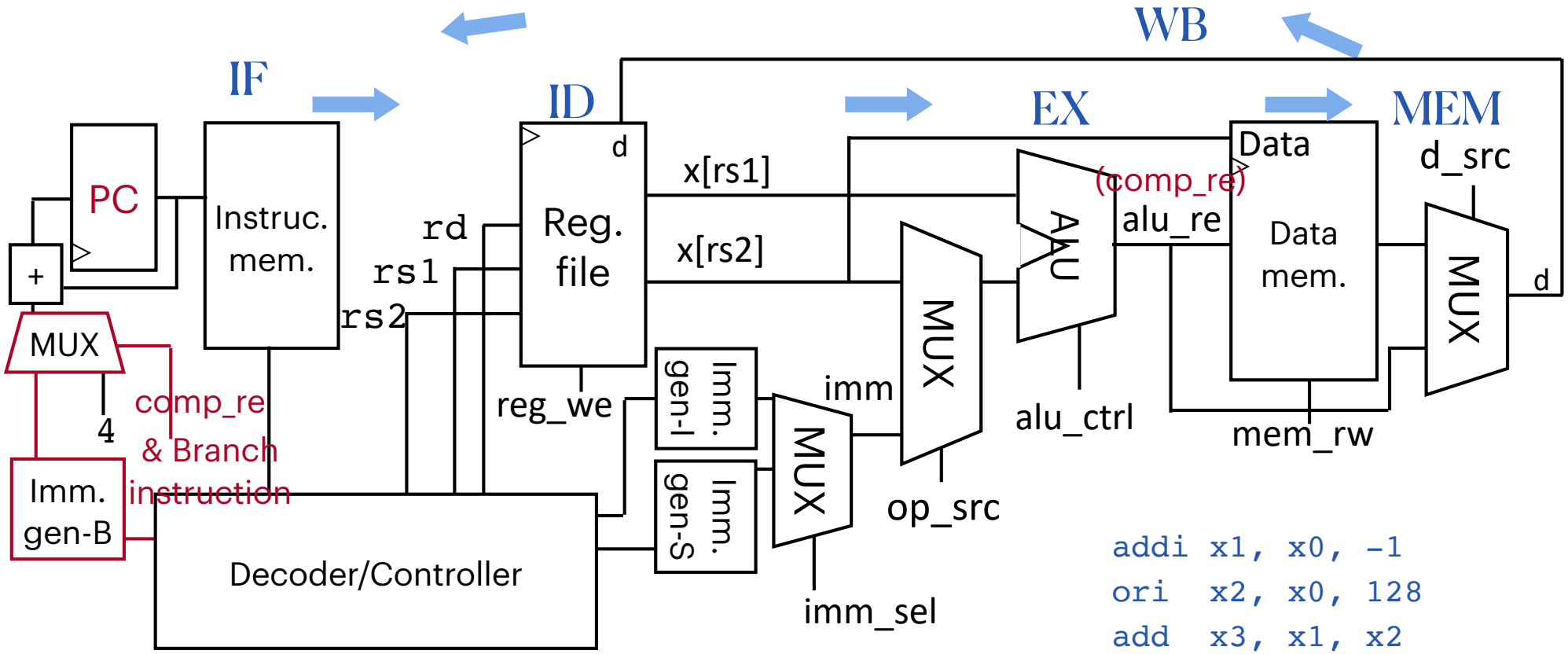





```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
beq  x3, x5, -12
```

	addi	add	sw	lw	beq
reg_we	1	1	0	1	
mem_rw	R	R	W	R	
alu_ctrl	add	add	add	add	
imm_sel	I	I	S	I	
d_src	alu	alu	*	mem	
op_src	imm	reg	imm	imm	
PC_mux					

Output = control signals

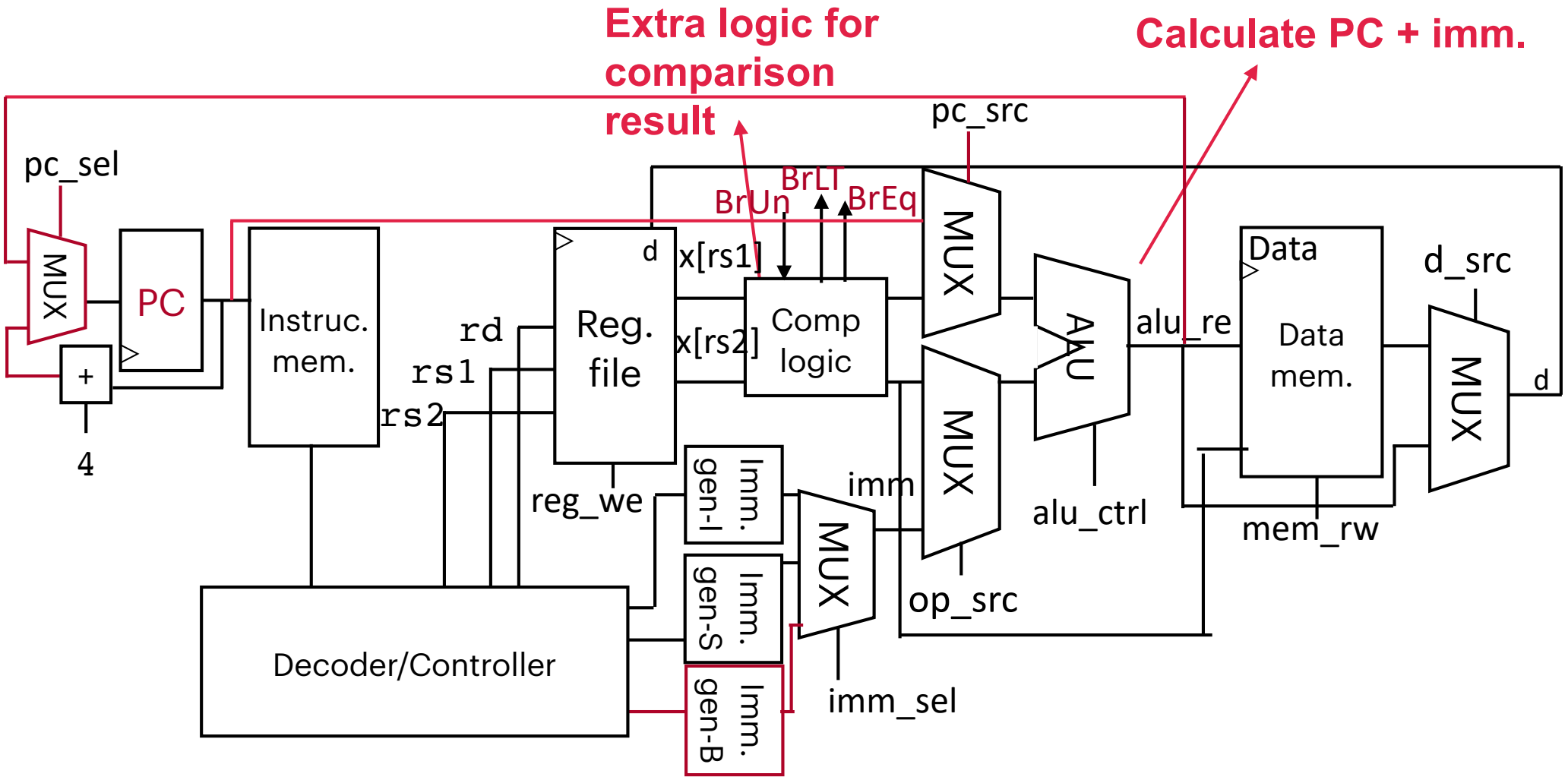



```
addi x1, x0, -1
ori  x2, x0, 128
add  x3, x1, x2
sw   x3, 0(x3)
lw   x5, 0(x3)
beq  x3, x5, -12
```

	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	I	S	I	B
d_src	alu	alu	*	mem	alu
op_src	imm	reg	imm	imm	reg
PC_mux	+4	+4	+4	+4	comp_re & Branch instruction

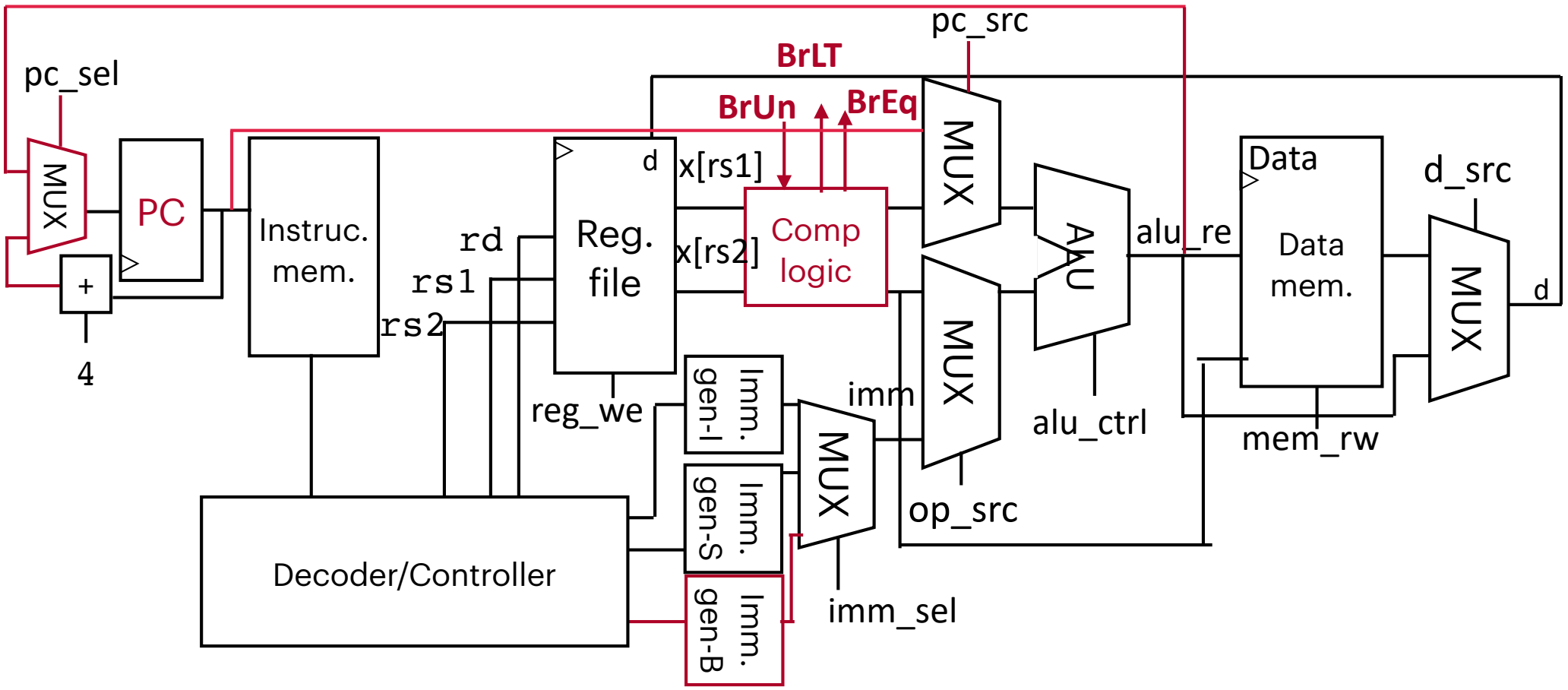
Output = control signals

Another Implementation



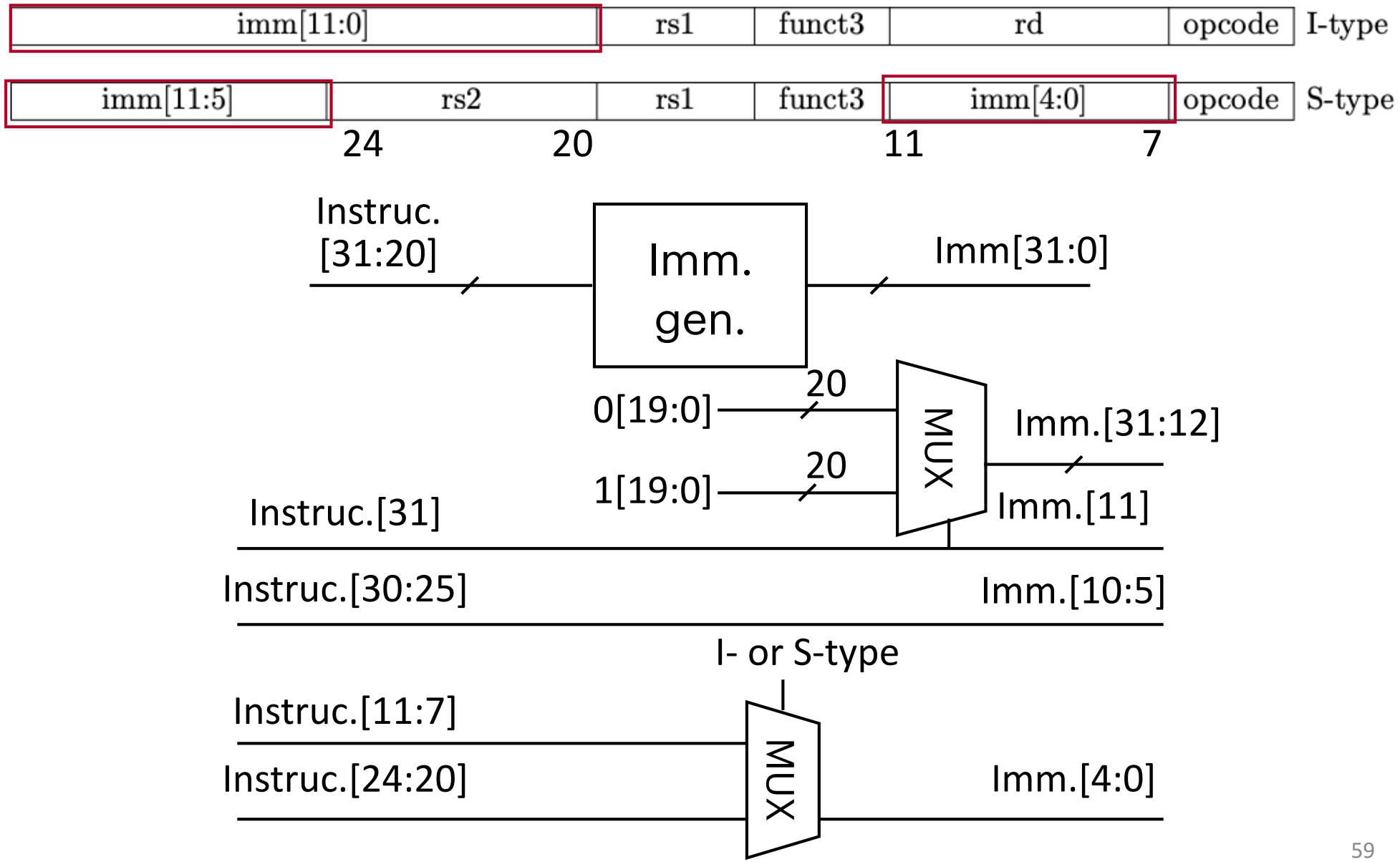
There Are a Thousand Hamlets in a Thousand People's Eyes. — Shakespeare

Compare-Logic



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

Consideration for IMM Generation



Consideration for IMM Generation