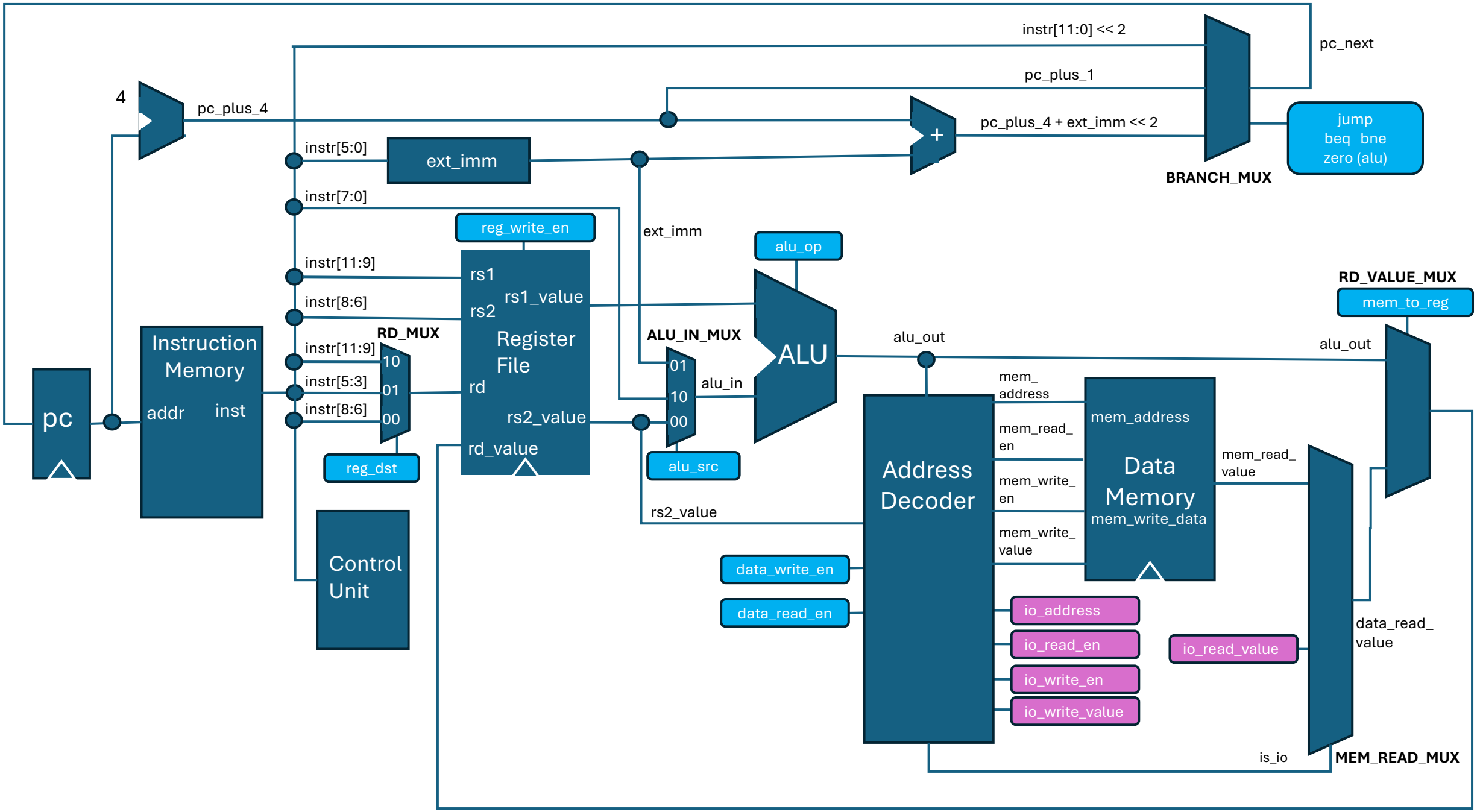
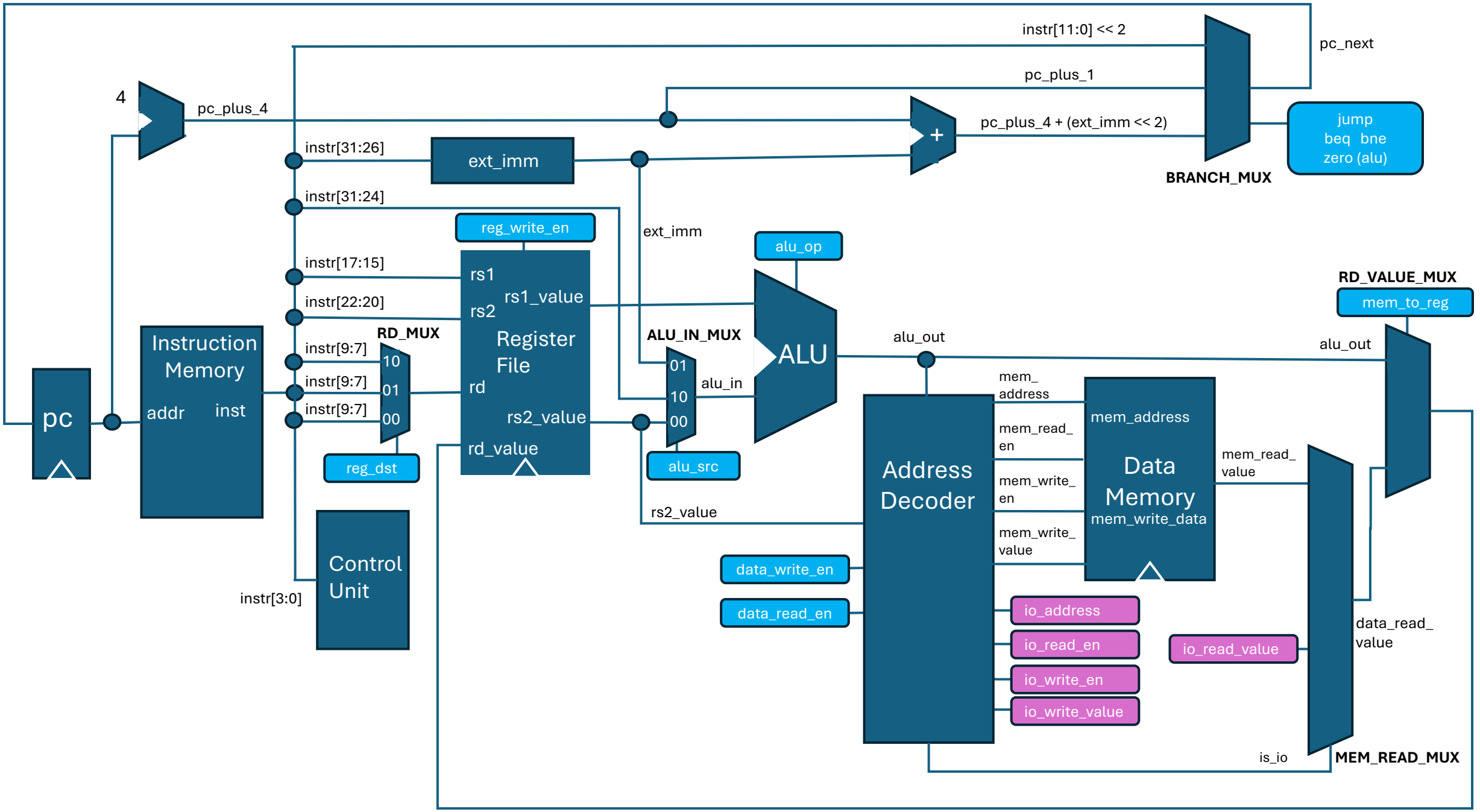
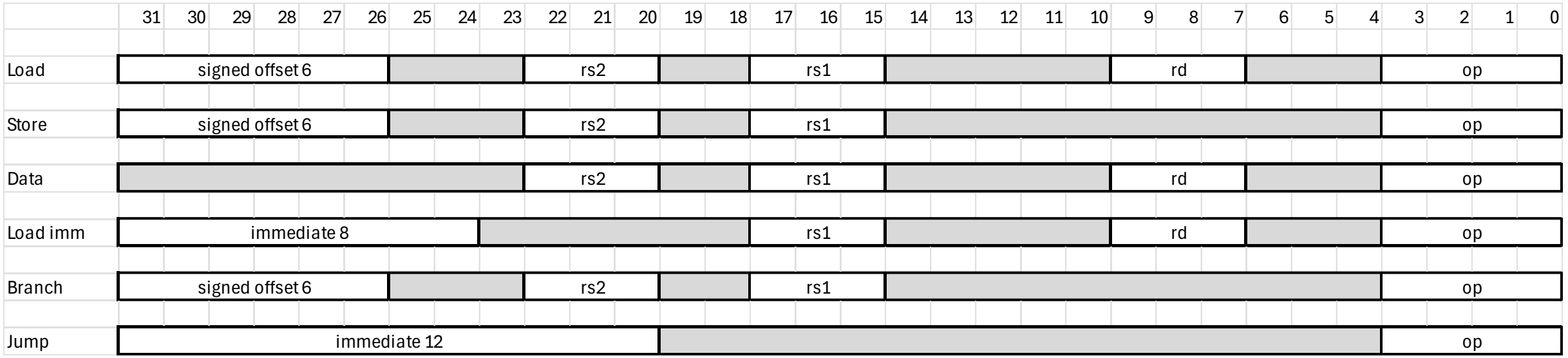
RISC V


instr[11:0] << 2
pc_next
pc_plus_1
4
pc_plus_4
pc_plus_4 + ext_imm << 2
jump
beq bne
zero (alu)
BRANCH_MUX
instr[5:0]
ext_imm
instr[7:0]
reg_write_en
ext_imm
alu_op
instr[11:9]
instr[8:6]
rs1
rs2
rs1_value
Register
File
rd
rs2_value
rd_value
RD_MUX
ALU_IN_MUX
ALU
alu_out
RD_VALUE_MUX
mem_to_reg
alu_out
Instruction
Memory
addr
inst
pc
instr[11:9]
instr[5:3]
instr[8:6]
10
01
00
reg_dst
01
10
00
alu_in
alu_src
rs2_value
Address
Decoder
mem_
address
mem_read_
en
mem_write_
en
mem_write_
value
mem_address
Data
Memory
mem_write_data
mem_read_
value
data_write_en
data_read_en
io_address
io_read_en
io_write_en
io_write_value
io_read_value
data_read_
value
is_io
MEM_READ_MUX
Control
Unit


instr[11:0] << 2
pc_next
pc_plus_1
4
pc_plus_4
+
pc_plus_4 + (ext_imm << 2)
jump
beq bne
zero (alu)
BRANCH_MUX
instr[31:26]
ext_imm
instr[31:24]
reg_write_en
ext_imm
alu_op
instr[17:15]
rs1
rs1_value
instr[22:20]
rs2
Register
File
rd
rs2_value
rd_value
RD_MUX
ALU_IN_MUX
ALU
RD_VALUE_MUX
mem_to_reg
alu_out
alu_out
instr[9:7]
instr[9:7]
instr[9:7]
10
01
00
01
10
00
alu_in
pc
Instruction
Memory
addr
inst
reg_dst
alu_src
rs2_value
mem_
address
mem_read_
en
mem_write_
en
mem_write_
value
Address
Decoder
mem_address
Data
Memory
mem_write_data
mem_read_
value
data_write_en
data_read_en
io_address
io_read_en
io_write_en
io_write_value
io_read_value
data_read_
value
Control
Unit
instr[3:0]
is_io
MEM_READ_MUX

Instruction	Mnemonic	Action
Load word	LD rd, offset6(rs1)	rd := mem [rs1 + offset6]
Store word	ST rs2, offset6(rs1)	mem [rs1 + offset6] := rs2
Add	ADD rd, rs1, rs2	rd := rs1 + rs2
Subtract	SUB rd, rs1, rs2	rd := rs1 - rs2
Invert (1s complement)	INV rd, rs1	rd := !rs1
Logical Shift Left	LSL rd, rs1, rs2	rd := rs1 << rs2
Logical Shift Right	LSR rd, rs1,rs2	rd := rs1 >> rs2
Bitwise AND	AND rd, rs1, rs2	rd := rs1 & rs2
Bitwise OR	OR rd, rs1, rs2	rd := rs1 | rs2
Set on Less Than	SLT rd, rs1, rs2	rd := 1 if rs1 < rs2 rd := 0 if rs1 >= rs2
Branch on Equal	BEQ rs1, rs2, offset6	PC := PC + 4 + offset6 * 4 if rs1 == rs2
Branch on Not Equal	BNE rs1, rs2, offset6	PC := PC + 4 + offset6 * 4 if rs1 != rs2
Jump	JMP offset12	PC := offset12 * 4
Load upper	LUI rd, imm8	rd := {imm8, rd[7:0]}
Load lower	LLI rd, imm8	rd := {rd[15:8], imm8}


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Load
signed offset 6
rs2
rs1
rd
op
Store
signed offset 6
rs2
rs1
op
Data
rs2
rs1
rd
op
Load imm
immediate 8
rs1
rd
op
Branch
signed offset 6
rs2
rs1
op
Jump
immediate 12
op

OP	Instruction
0000	Load word
0001	Store word
0002	Add
0003	Subtract
0004	Invert (1s complement)
0005	Logical Shift Left
0006	Logical Shift Right
0007	Bitwise AND
0008	Bitwise OR
0009	Set on Less Than
0011	Branch on Equal
0012	Branch on Not Equal
0013	Jump
0014	Load upper immediate
0015	Load lower immed

Load word

op	**rs1**	**rd**	**signed offset6**
4	3	3	6

Store word

op	**rs1**	**rs2**	**signed offset6**
4	3	3	6

Data processing

op	**rs1**	**rs2**	**rd**	
4	3	3	3	3

Load imm

op	**Rd**		**imm8**
4	3	1	8

Branch

op	**rs1**	**rs2**	**signed offset6**
4	3	3	6

Jump

op	**imm12**
4	12

Control signals									
Instruction	**Reg Dst**	**ALUSrc**	**Memto Reg**	**Reg Write**	**Mem Read**	**Mem Write**	**Branch**	**ALUOp**	**Jump**
Data-processing	01	00	0	1	0	0	0	see below	0
LW	00	01	1	1	1	0	0	0000	0
SW	00	01	0	0	0	1	0	0000	0
BEQ,BNE	00	00	0	0	0	0	1	0001	0
J	00	00	0	0	0	0	0	0000	1
LUI, LLI	01	10	0	1	0	0	0	see below	0

ALU Op		
Opcode	**ALU Operation**	**ALU Op**
02	ADD	0000
03	SUB	0001
04	INV	0010
05	LSL	0011
06	LSR	0100
07	AND	0101
08	OR	0110
09	SLT	0111
14	LUI	1000
15	LLI	1001

ALU Op			
Opcode	**Instruction**	**ALU Operation**	**ALUop**
00	LD	ADD	0000
01	ST	ADD	0000
02	ADD	ADD	0000
03	SUB	SUB	0001
04	INV	INV	0010
05	LSL	LSL	0011
06	LSR	LSR	0100
07	AND	AND	0101
08	OR	OR	0110
09	SLT	SLT	0111
10			
11	BEQ	SUB	0001
12	BNE	SUB	0001
13	JMP	ADD	0000
14	LUI	LUI	1000
15	LLI	LLI	1001