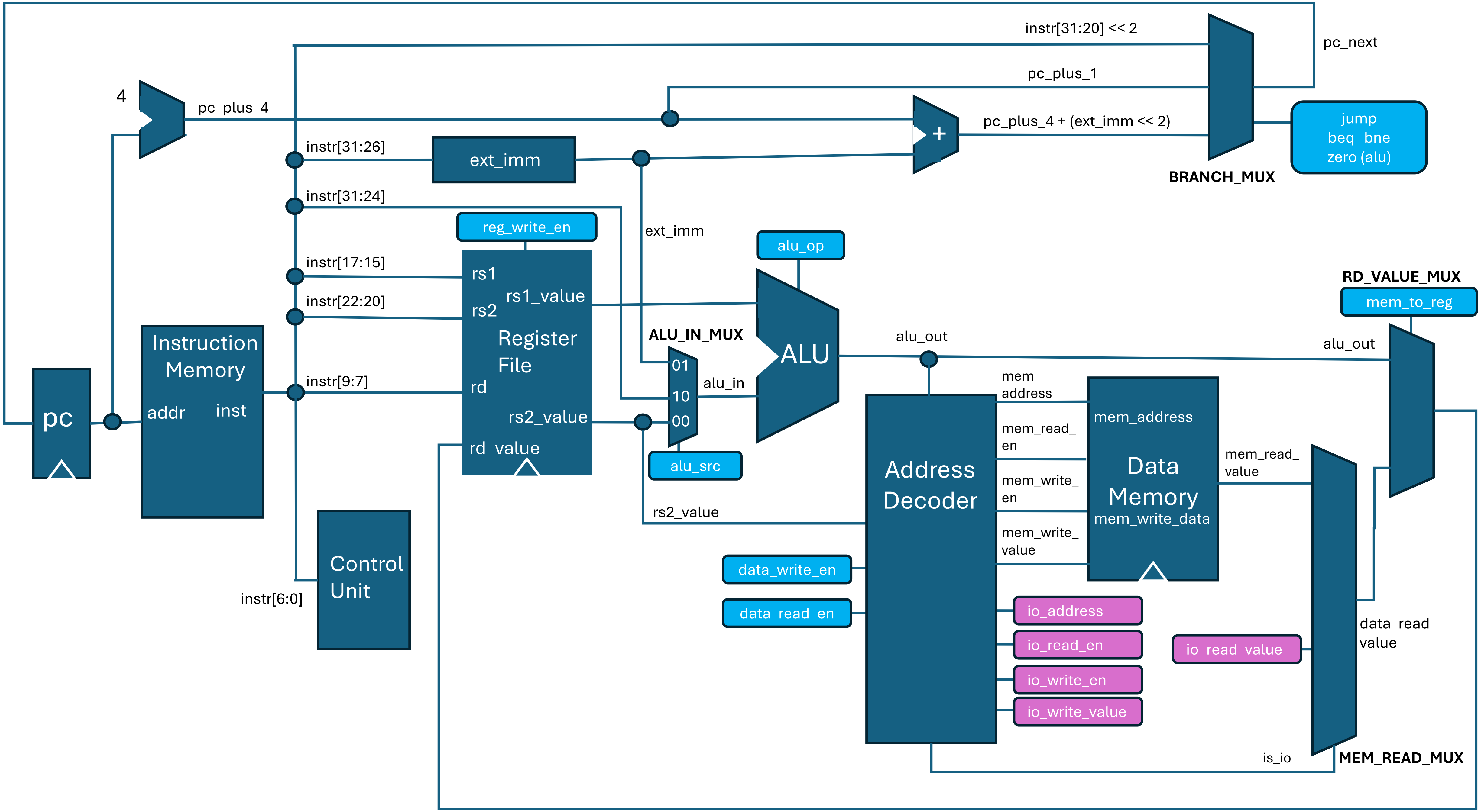
RISC V


pc_next
instr[31:20] << 2
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]
ext_imm
reg_write_en
alu_op
instr[17:15]
rs1
rs1_value
rs2
instr[22:20]
Register
File
rd
rs2_value
rd_value
ALU_IN_MUX
01
10
00
alu_in
alu_src
ALU
alu_out
RD_VALUE_MUX
mem_to_reg
alu_out
Instruction
Memory
addr
inst
instr[9:7]
pc
mem_
address
mem_read_
en
mem_write_
en
mem_write_
value
mem_address
Data
Memory
mem_write_data
mem_read_
value
Address
Decoder
rs2_value
Control
Unit
instr[6:0]
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

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, rs1, imm8	rd := {imm8, rs1[7:0]}
Load lower	LLI rd, rs1, imm8	rd := {rs1[15:8], imm8}

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

Control signals

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

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

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

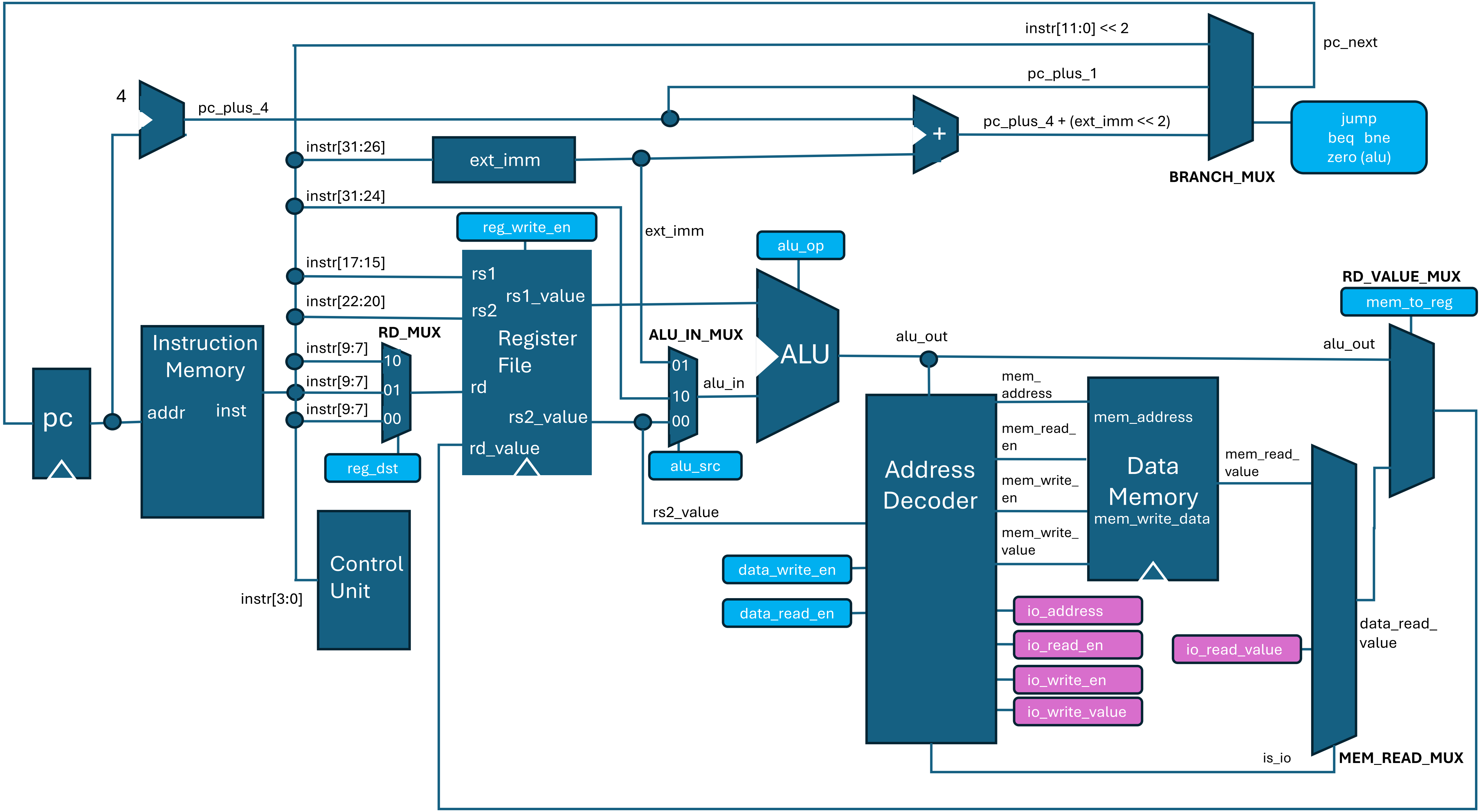
op	Rd		imm8
4	3	1	8

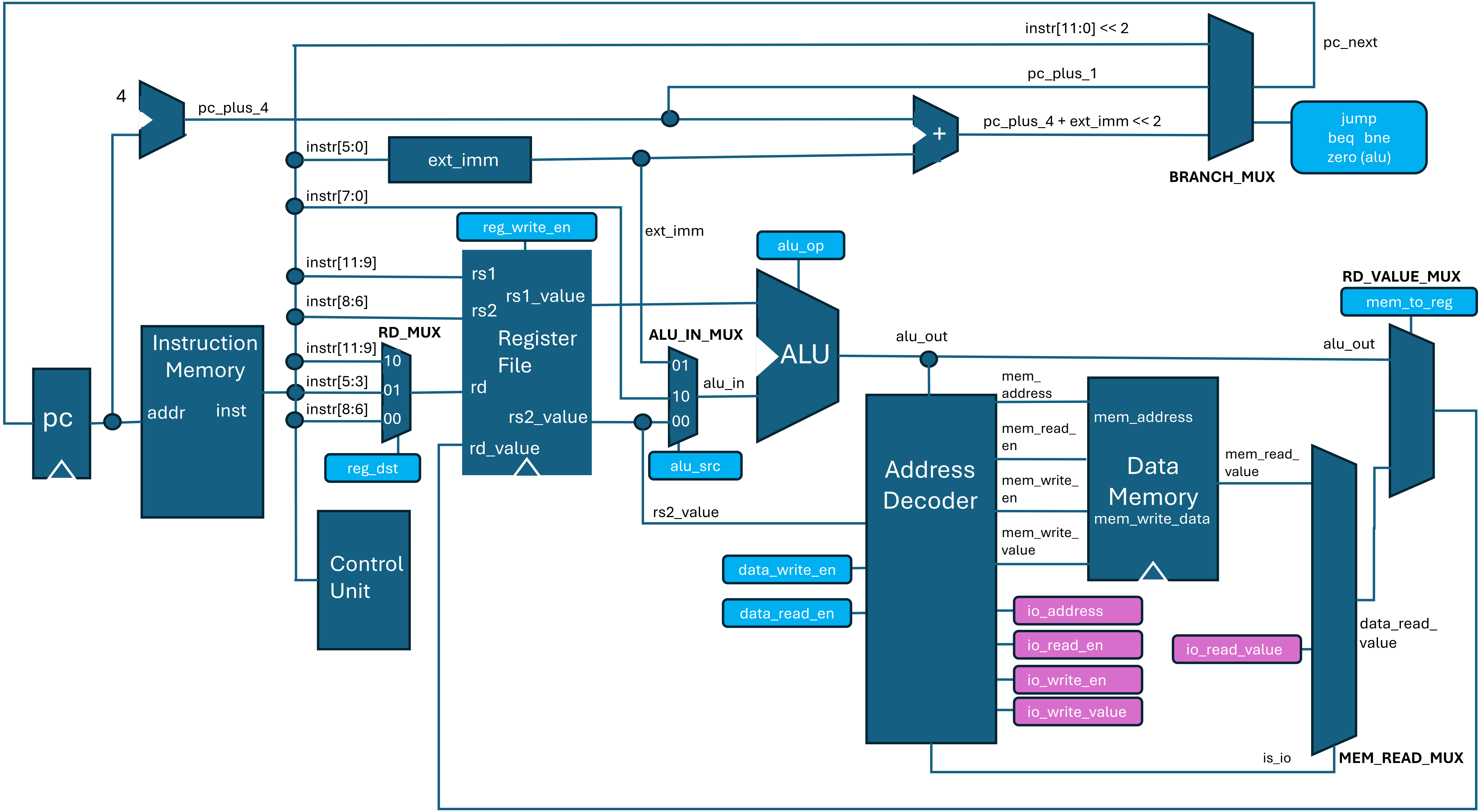
Branch

op	rs1	rs2	signed offset6
4	3	3	6

Jump

op	imm12
4	12


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_VALUE_MUX
mem_to_reg
RD_MUX
ALU_IN_MUX
ALU
alu_out
alu_out
Instruction
Memory
instr[9:7]
10
01
instr[9:7]
01
10
alu_in
rd
mem_
address
instr[9:7]
00
00
pc
addr
inst
rs2_value
mem_address
mem_read_
en
rd_value
reg_dst
alu_src
mem_read_
value
Address
Decoder
Data
Memory
mem_write_
en
mem_write_data
rs2_value
mem_write_
value
Control
Unit
data_write_en
instr[3:0]
data_read_en
io_address
data_read_
value
io_read_en
io_read_value
io_write_en
io_write_value
is_io
MEM_READ_MUX


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]
ext_imm
reg_write_en
alu_op
instr[11:9]
rs1
rs1_value
rs2
instr[8:6]
Register
File
RD_VALUE_MUX
mem_to_reg
RD_MUX
ALU_IN_MUX
Instruction
Memory
instr[11:9]
10
01
ALU
alu_out
alu_out
instr[5:3]
01
rd
10
alu_in
mem_
address
pc
addr
inst
instr[8:6]
00
rs2_value
00
mem_address
mem_read_
en
reg_dst
rd_value
alu_src
mem_read_
value
Address
Decoder
Data
Memory
mem_write_
en
rs2_value
mem_write_data
mem_write_
value
Control
Unit
data_write_en
data_read_en
io_address
data_read_
value
io_read_en
io_read_value
io_write_en
io_write_value
is_io
MEM_READ_MUX