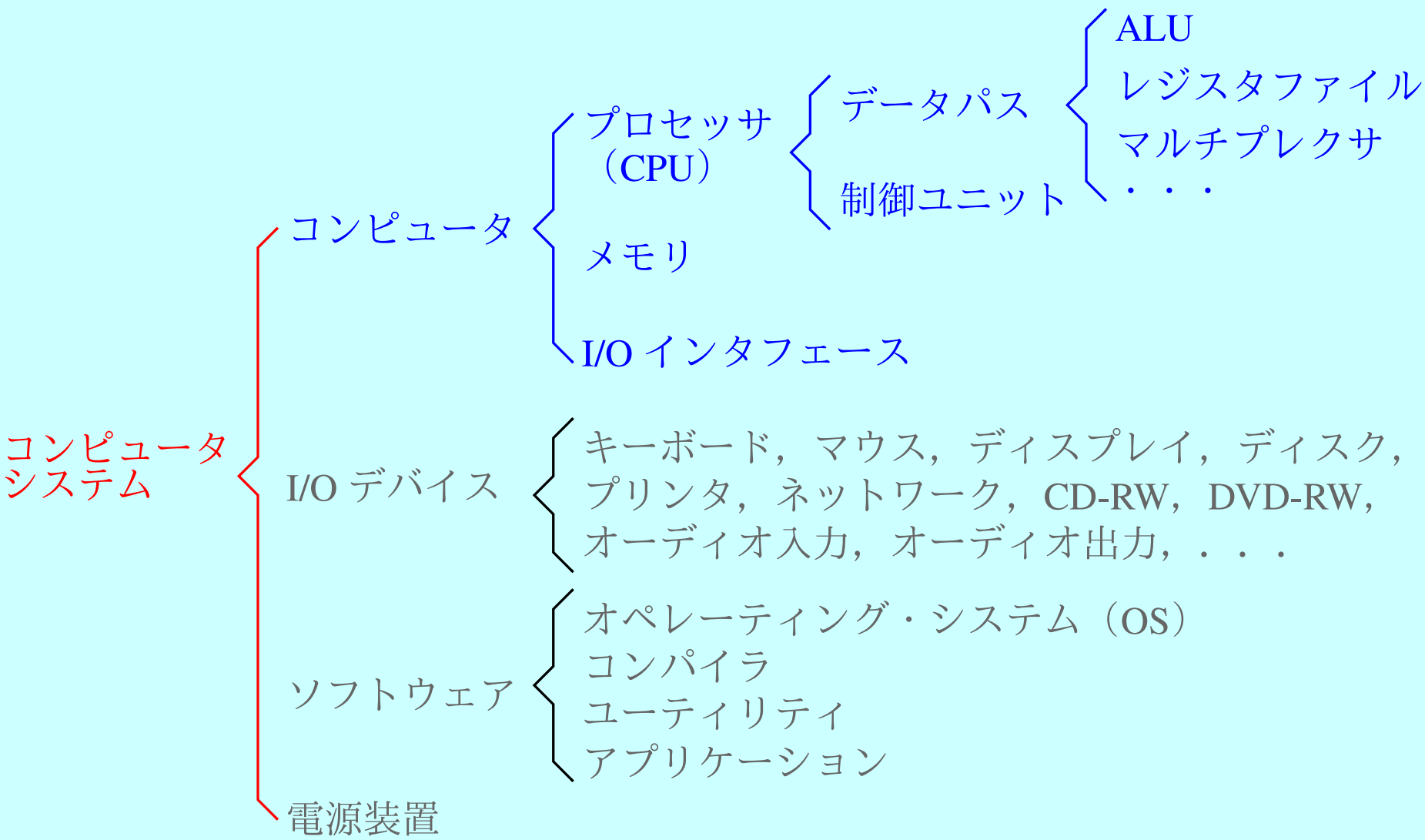
コンピュータ構成と設計（５）

単一サイクル CPU 設計

李　亜民

2022 年 10 月 24 日 (月)

コンピュータとコンピュータシステム



20 RISC-V 命令のまとめ

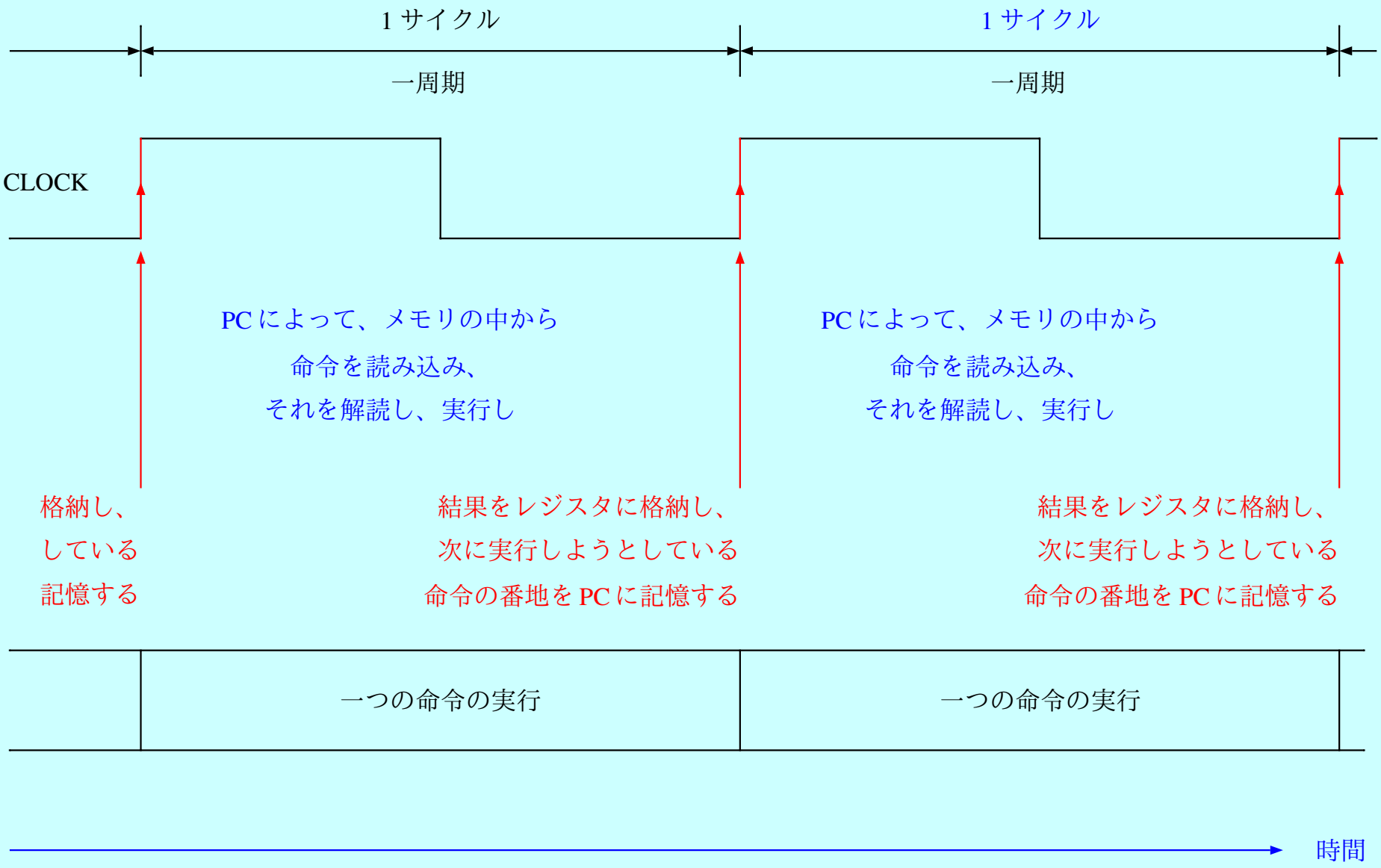
```
 1. add  rd, rs1, rs2    # rd <- rs1 + rs2
 2. sub  rd, rs1, rs2    # rd <- rs1 - rs2
 3. slt  rd, rs1, rs2    # rd <- rs1 < rs2 (signed)
 4. xor  rd, rs1, rs2    # rd <- rs1 ^ rs2
 5. or   rd, rs1, rs2    # rd <- rs1 | rs2
 6. and  rd, rs1, rs2    # rd <- rs1 & rs2
 7. slli rd, rs1, shamt  # rd <- rs1 << shamt
 8. srli rd, rs1, shamt  # rd <- rs1 >> shamt
 9. srai rd, rs1, shamt  # rd <- rs1 >>>shamt
10. jalr rd, rs1, imm    # rd <- pc+4; pc <- rs1+imm
11. addi rd, rs1, imm    # rd <- rs1 + imm
12. xori rd, rs1, imm    # rd <- rs1 ^ imm
13. ori  rd, rs1, imm    # rd <- rs1 | imm
14. andi rd, rs1, imm    # rd <- rs1 & imm
15. lw   rd,  imm(rs1)   # rd <- memory[rs1+imm]
16. sw   rs2, imm(rs1)   # memory[rs1+imm] <- rs2
17. beq  rs1, rs2, label # if (rs1==rs2) pc <- label
18. bne  rs1, rs2, label # if (rs1!=rs2) pc <- label
19. jal  rd,  label      # rd <- pc+4;   pc <- label
20. lui  rd,  imm        # rd <- imm,000000000000
```

RV32I Base Instruction Set Encoding

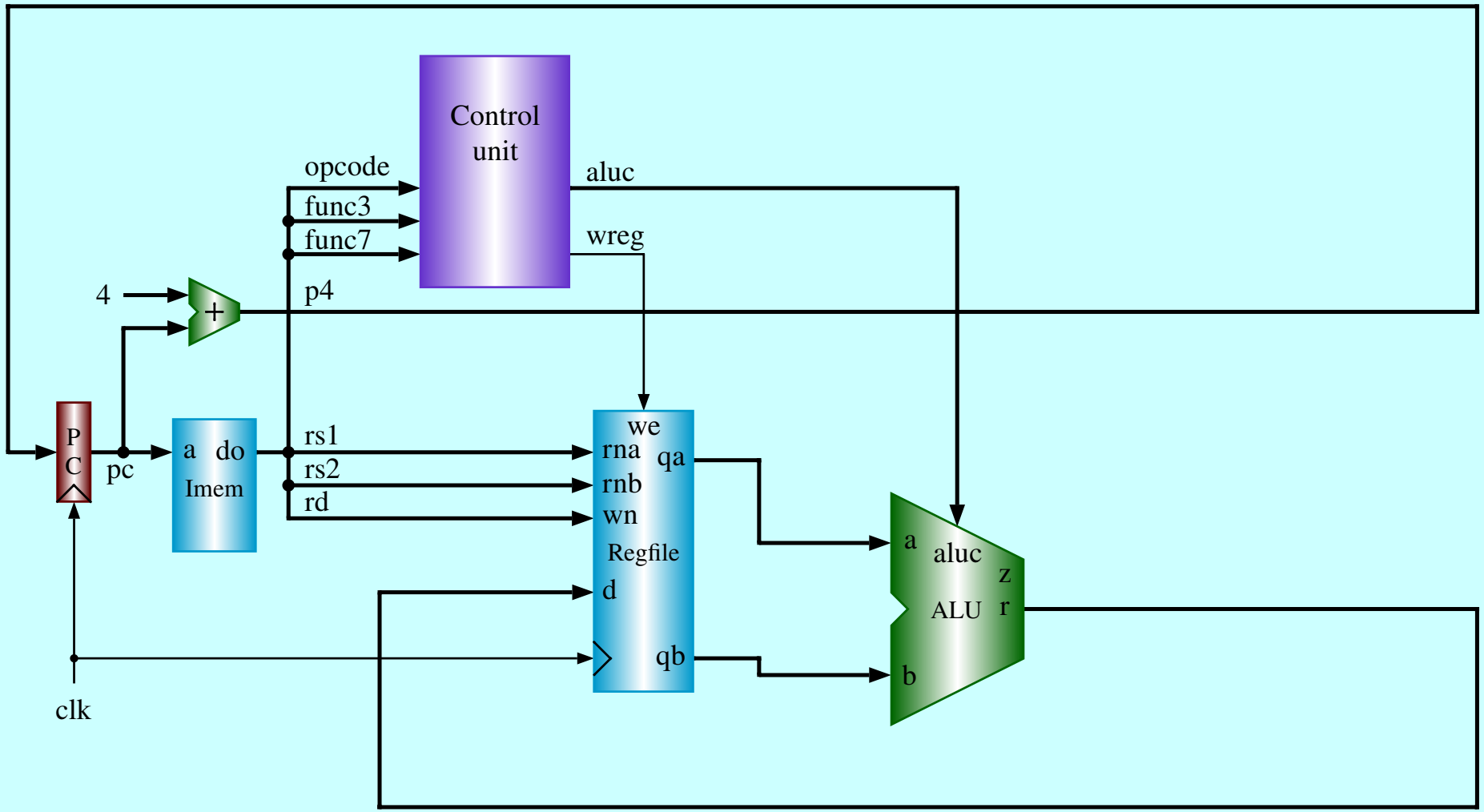
0000000	rs2	rs1	000	rd	0110011
0100000	rs2	rs1	000	rd	0110011
0000000	rs2	rs1	010	rd	0110011
0000000	rs2	rs1	100	rd	0110011
0000000	rs2	rs1	110	rd	0110011
0000000	rs2	rs1	111	rd	0110011
0000000	shamt	rs1	001	rd	0010011
0000000	shamt	rs1	101	rd	0010011
0100000	shamt	rs1	101	rd	0010011
imm[11:0]		rs1	000	rd	1100111
imm[11:0]		rs1	000	rd	0010011
imm[11:0]		rs1	100	rd	0010011
imm[11:0]		rs1	110	rd	0010011
imm[11:0]		rs1	111	rd	0010011
imm[11:0]		rs1	010	rd	0000011
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011
imm[20|10:1|11|19:12]				rd	1101111
imm[31:12]				rd	0110111

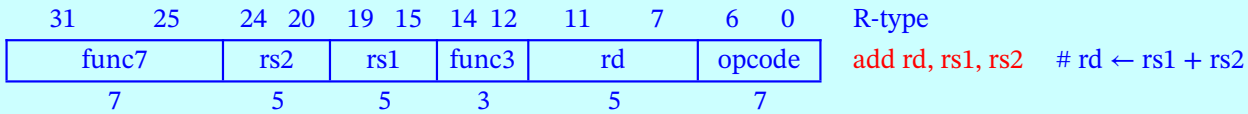
1. add
2. sub
3. slt
4. xor
5. or
6. and
7. slli
8. srli
9. srai
10. jalr
11. addi
12. xori
13. ori
14. andi
15. lw
16. sw
17. beq
18. bne
19. jal
20. lui

単一サイクル CPU


1 サイクル
一周期
1 サイクル
一周期
CLOCK
PC によって、メモリの中から
命令を読み込み、
それを解読し、実行し
PC によって、メモリの中から
命令を読み込み、
それを解読し、実行し
格納し、
している
記憶する
結果をレジスタに格納し、
次に実行しようとしている
命令の番地を PC に記憶する
結果をレジスタに格納し、
次に実行しようとしている
命令の番地を PC に記憶する
一つの命令の実行
一つの命令の実行
時間

RISC-V add, sub, slt, xor, or, and

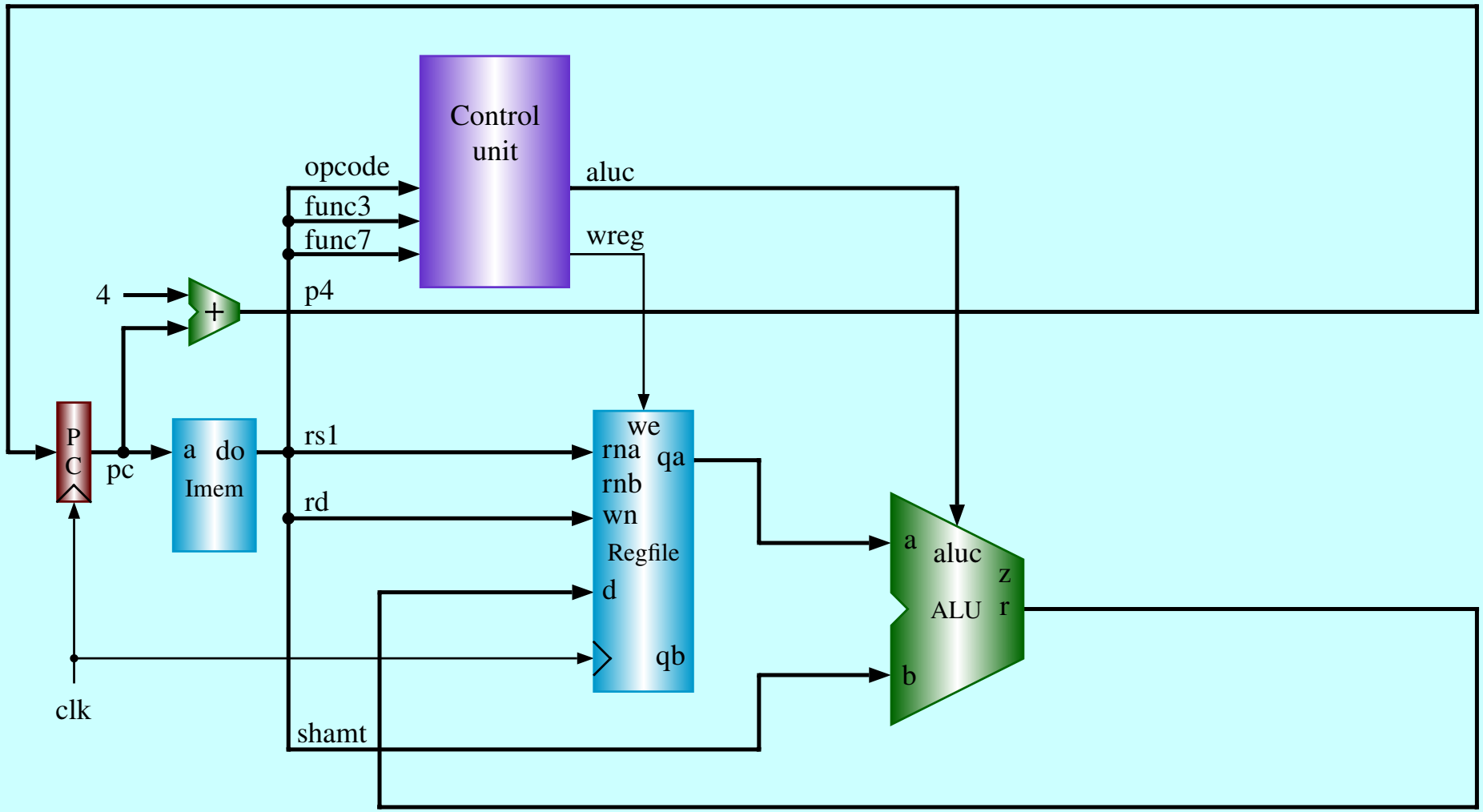




R-type

add rd, rs1, rs2 # rd ← rs1 + rs2

RISC-V slli, srli, srai

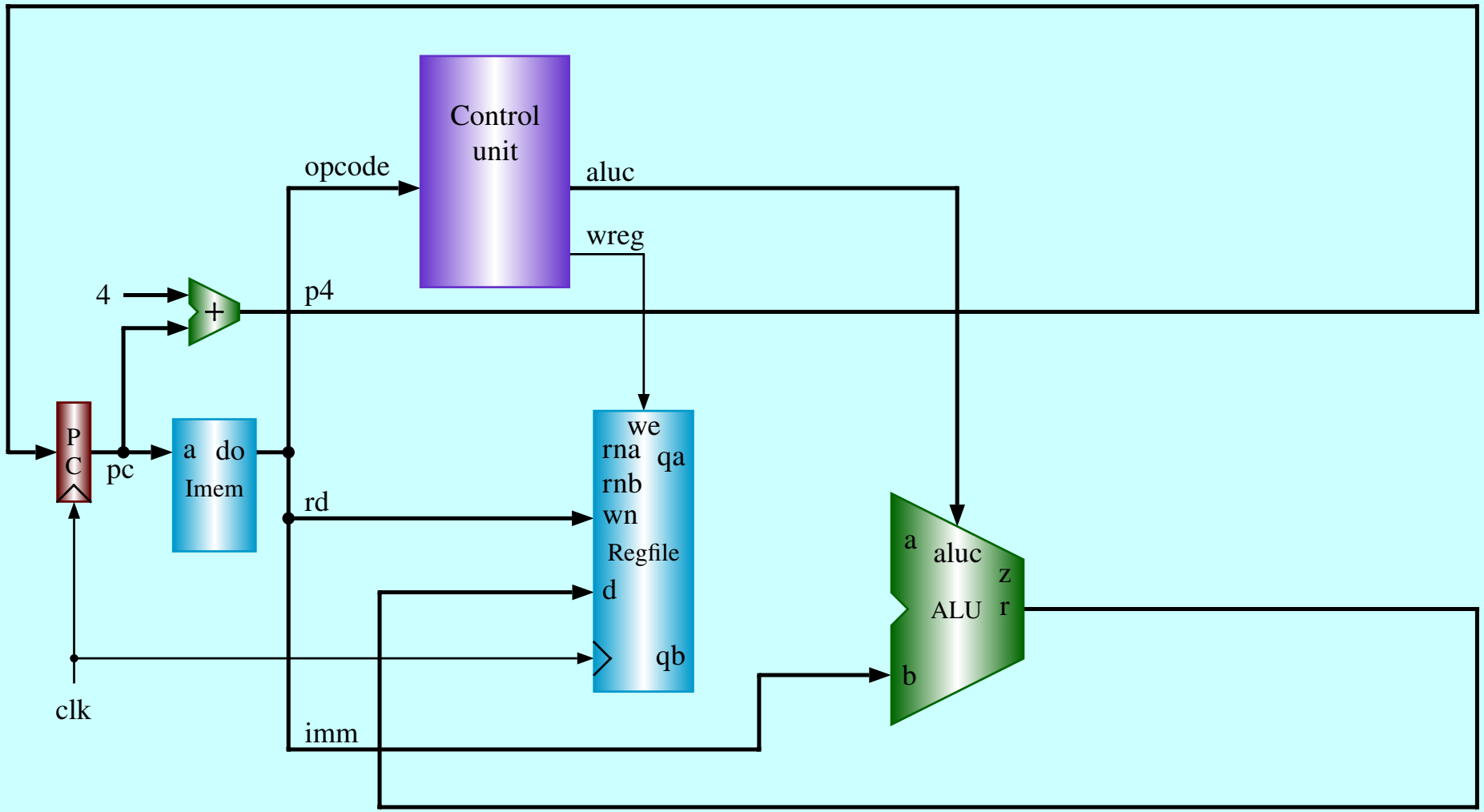


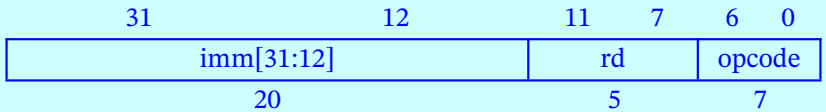
31 25	24 20	19 15	14 12	11 7	6 0
func7	shamt	rs1	func3	rd	opcode
7	5	5	3	5	7

H-type

slli rd, rs1, shamt # rd ← rs1 << shamt

RISC-V lui

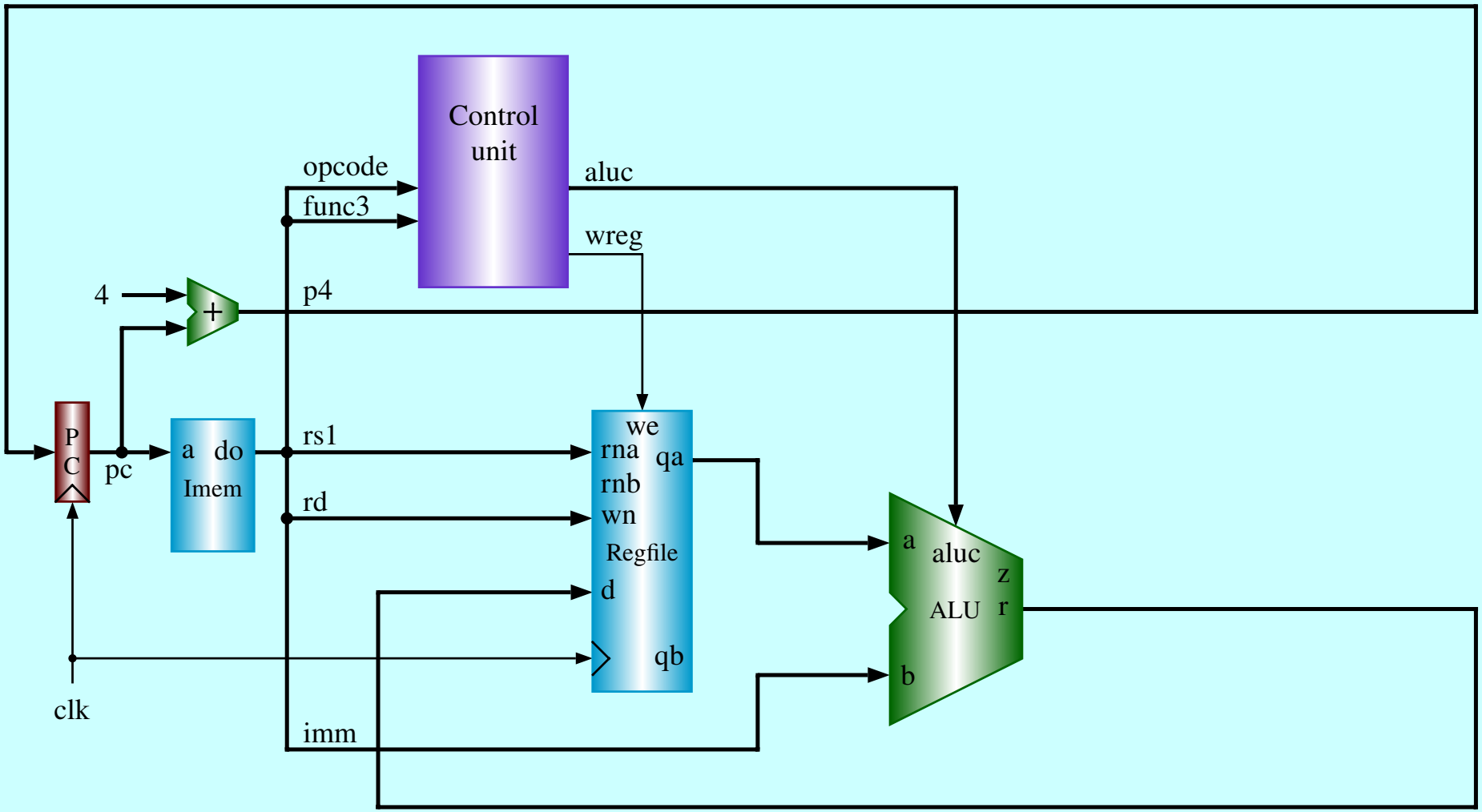




U-type

lui rd, imm # rd ← {imm[31:12],000000000000}

RISC-V addi, xori, ori, andi

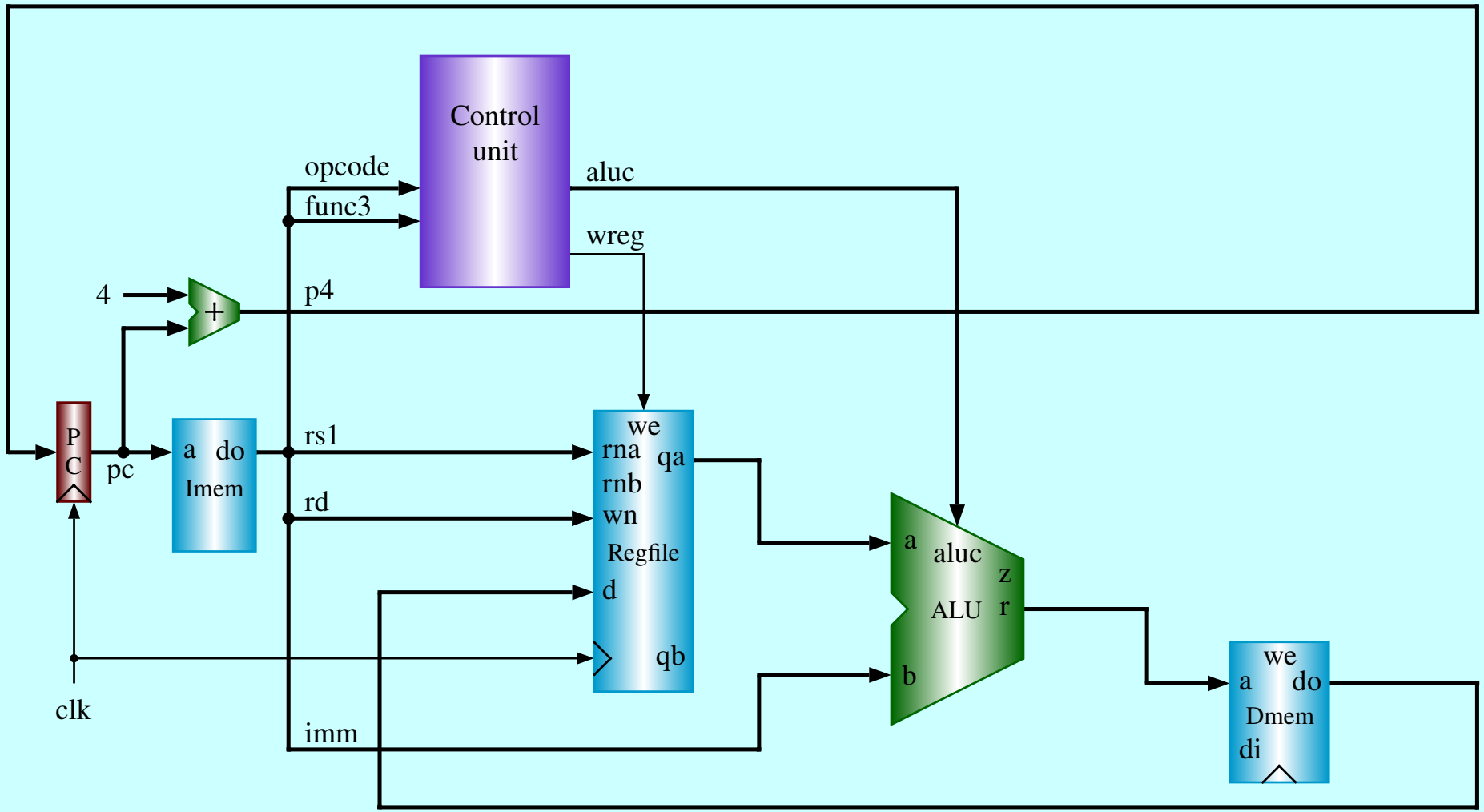


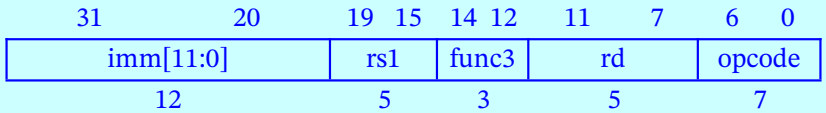
31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
imm[11:0]	rs1	func3	rd	opcode
12	5	3	5	7

I-type

addi rd, rs1, imm # rd ← rs1 + $signed(imm[11:0])

RISC-V lw

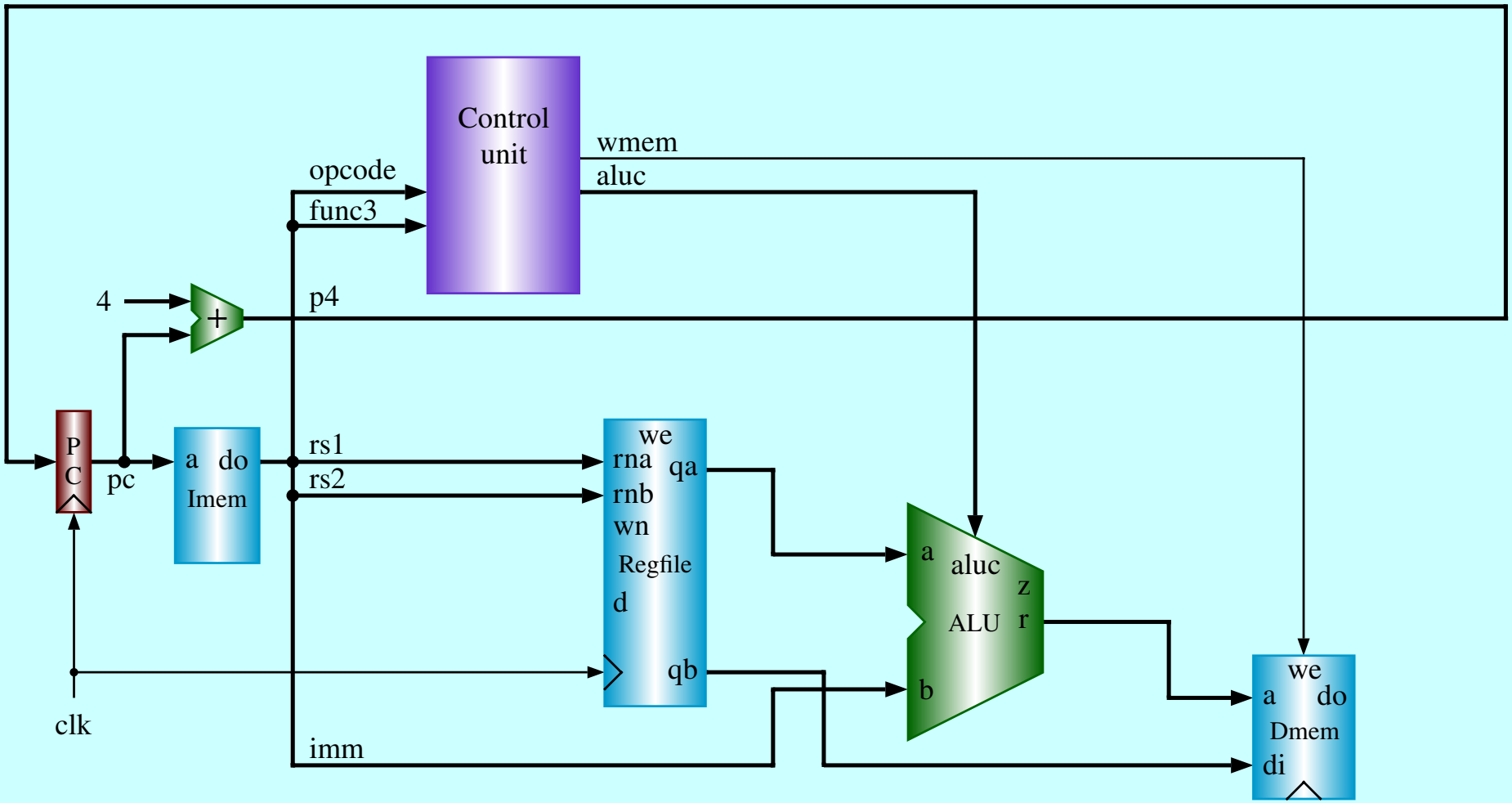




I-type

lw rd, imm(rs1) # rd ← memory[rs1+$signed(imm[11:0])]

RISC-V sw

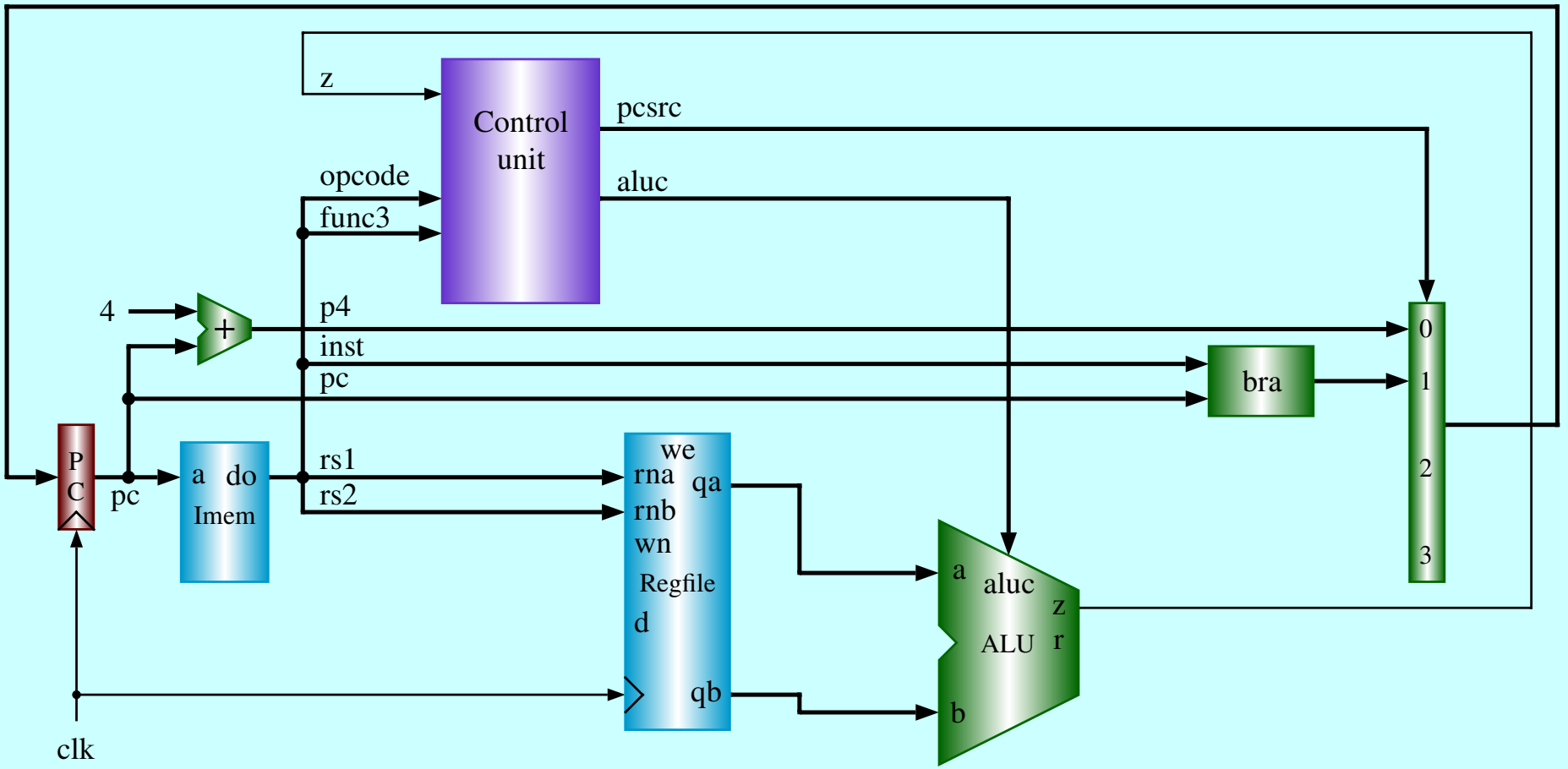


31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode
7	5	5	3	5	7

S-type

sw rs2, imm(rs1) # memory[rs1+$signed(imm[11:0])] ← rs2

RISC-V beq, bne

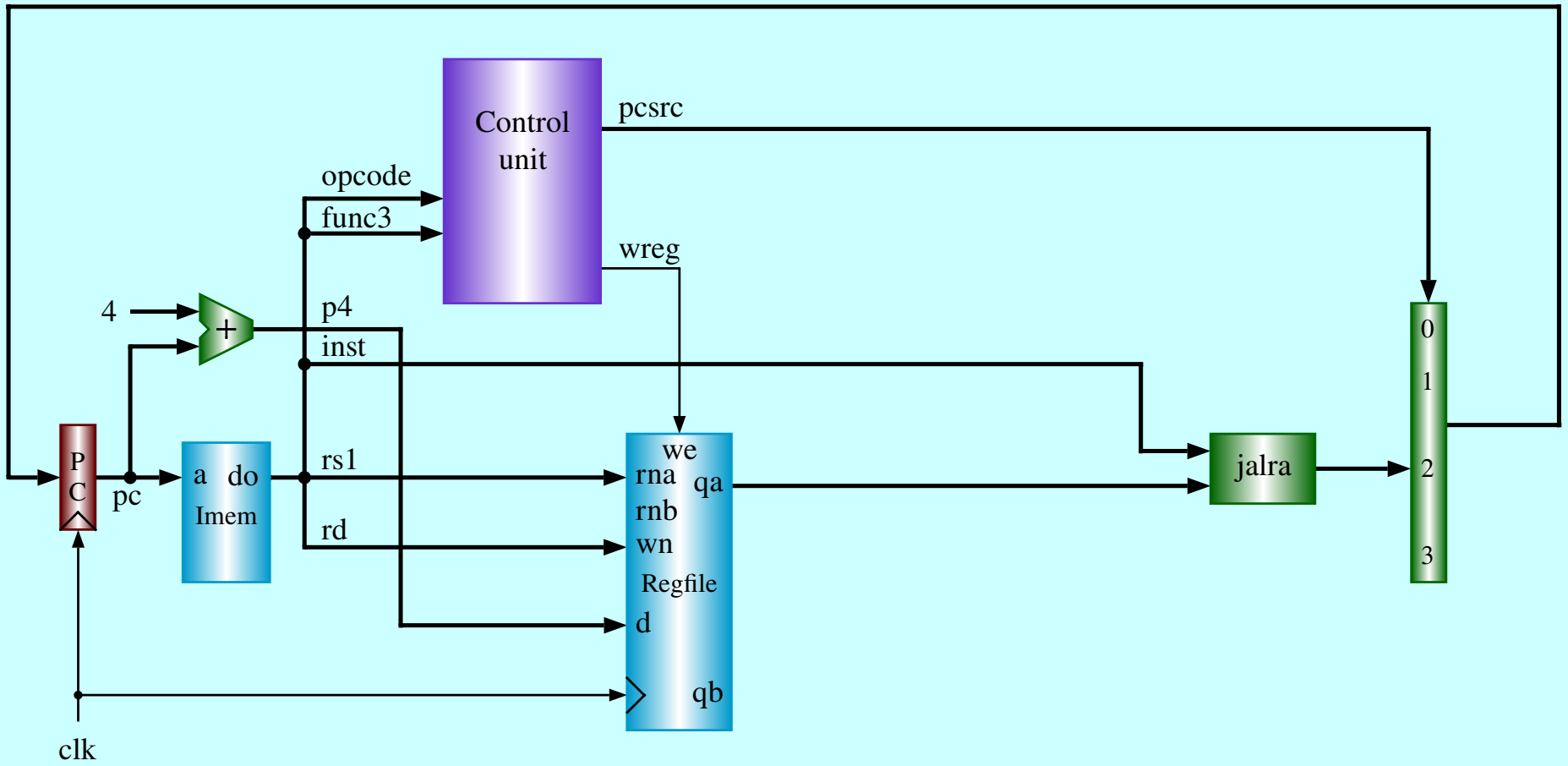


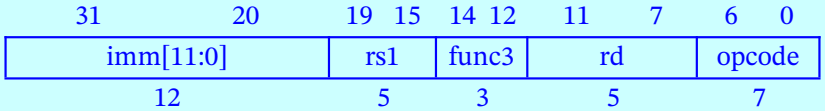
31 25	24 20	19 15	14 12	11 7	6 0
imm[12|10:5]	rs2	rs1	func3	imm[4:1|11]	opcode
7	5	5	3	5	7

B-type

beq rs1, rs2, label　　# if (rs1 = rs2) pc ← pc + $signed({imm[12:1],0})

RISC-V jalr

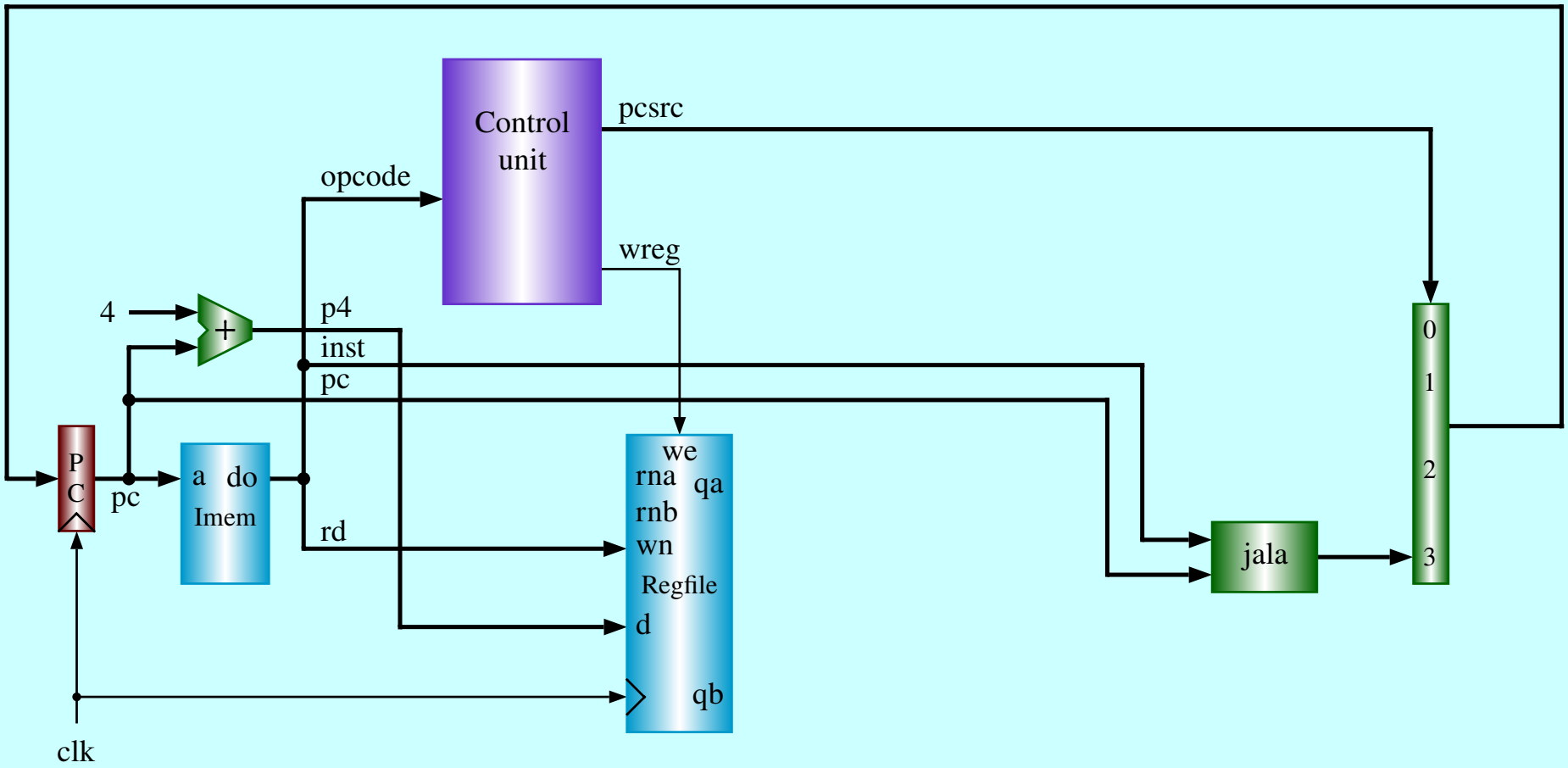


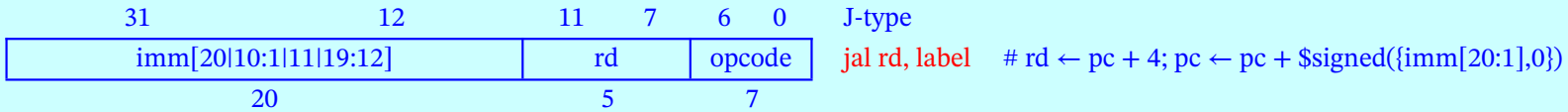


I-type

jalr rd, rs1, imm # rd ← pc + 4; pc ← (rs1 + $signed(imm[11:0]) & 0xfffffffe

RISC-V jal

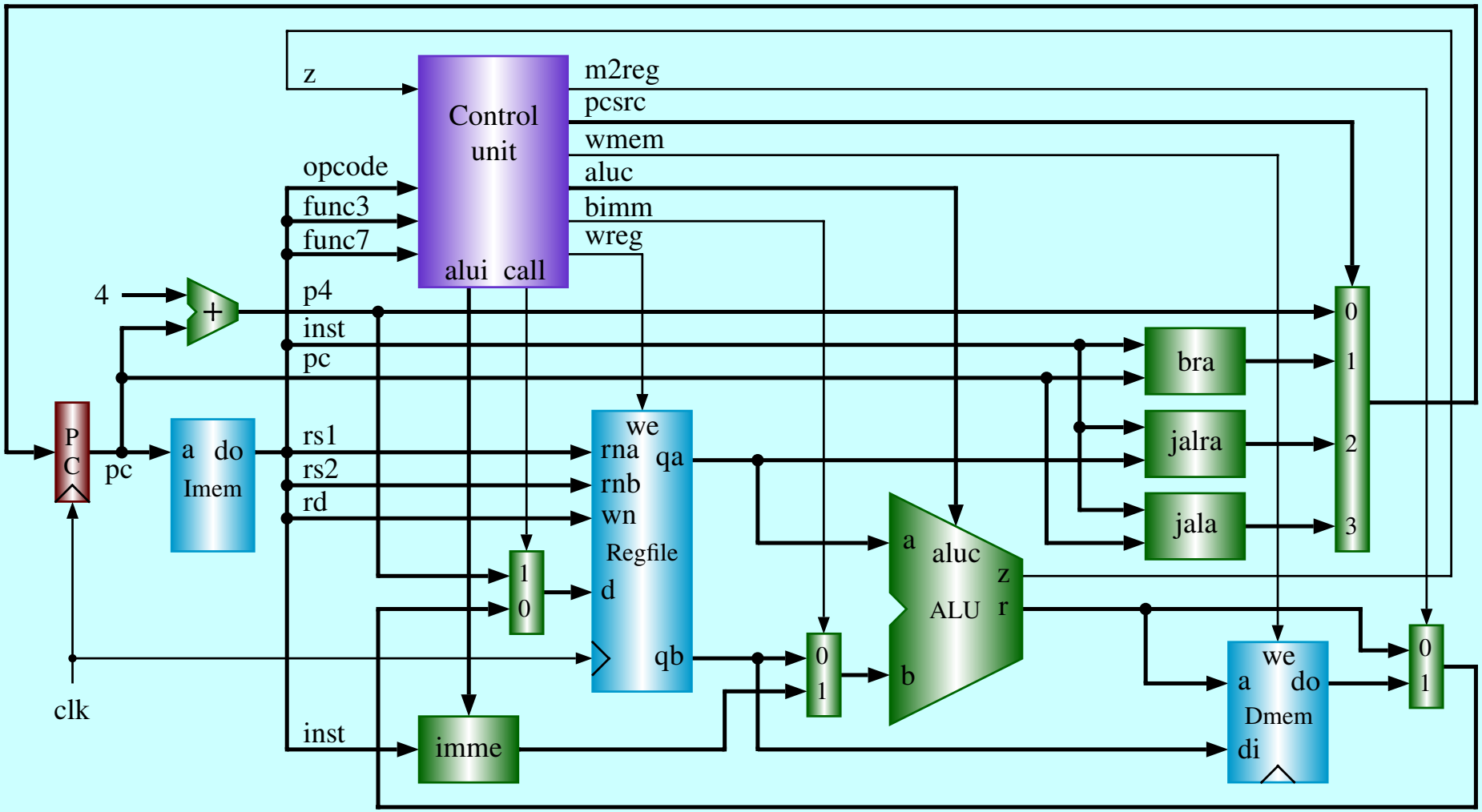




J-type

jal rd, label # rd ← pc + 4; pc ← pc + $signed({imm[20:1],0})

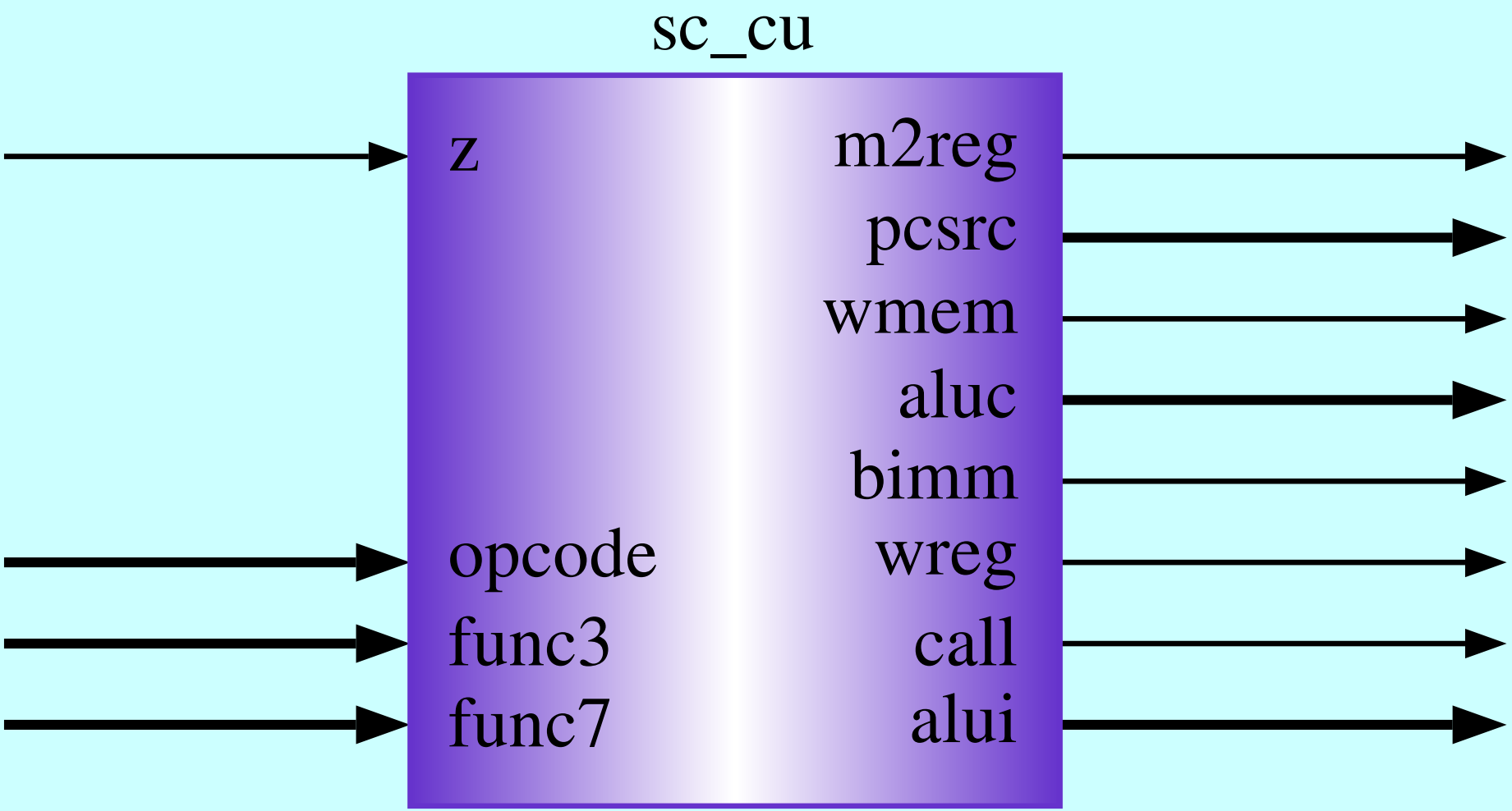
RISC-V コンピュータ



CPU + 命令メモリ Imem + データメモリ Dmem

制御ユニット (Control Unit) の設計

CPU 制御ユニット


sc_cu
z
opcode
func3
func7
m2reg
pcsrc
wmem
aluc
bimm
wreg
call
alui

制御信号

- wreg
  - 1: レジスタ・ファイルに書き込む
  - 0: レジスタ・ファイルに書き込まない
- wmem
  - 1: メモリに書き込む
  - 0: メモリに書き込まない
- m2reg
  - 1: メモリから読み出したデータを選択
  - 0: ALU の出力を選択
- bimm
  - 1: 拡張された即値を選択
  - 0: レジスタ・ファイル内のデータを選択

制御信号

- call
  - 1: PC+4 を選択
  - 0: ALU，または，メモリデータを選択
- pcsrc[1:0]
  - 0 0: PC+4 を選択
  - 0 1: 分岐先アドレスを選択
  - 1 0: レジスタアドレスを選択
  - 1 1: ジャンプ先アドレスを選択
- alui[1:0]
  - 0 0: addi, xori, ori, andi, lw 命令の即値を選択
  - 0 1: slli, srli, srai 命令の shamt を選択
  - 1 0: sw 命令の即値を選択
  - 1 1: lui 命令の即値を選択

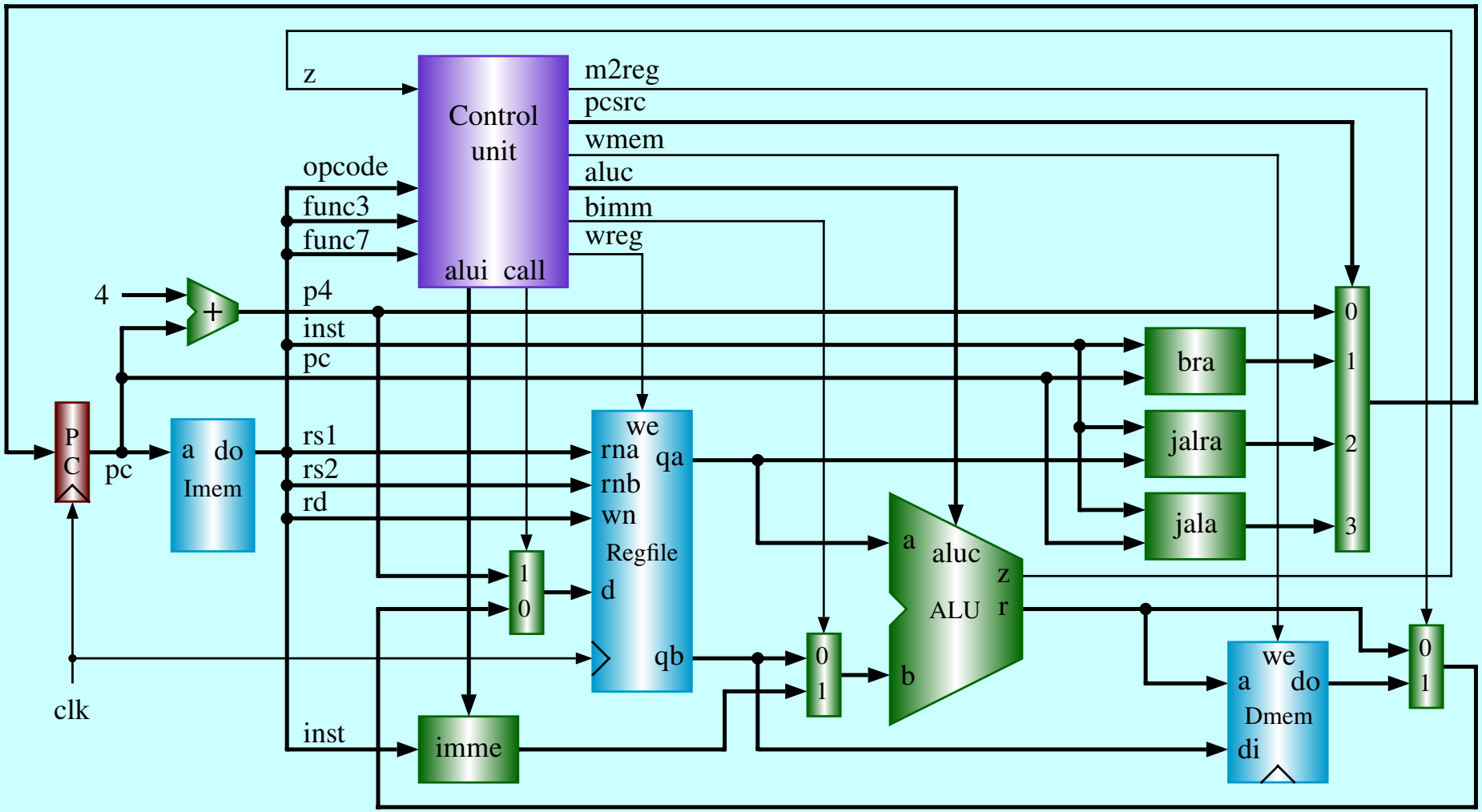
制御信号

- aluc[3:0]
  - x 0 0 0 : ADD　(命令:add, addi, lw, sw)
  - x 0 0 1 : SUB　(命令:sub, beq, bne)
  - x 0 1 0 : SLT　(命令:slt)
  - 1 0 1 1 : XOR　(命令:xor, xori)
  - x 1 0 0 : OR　(命令:or, ori)
  - x 1 0 1 : AND　(命令:and, andi)
  - 0 0 1 1 : SLL　(命令:slli)
  - 0 1 1 1 : SRL　(命令:srli)
  - 1 1 1 1 : SRA　(命令:srai)
  - x 1 1 0 : LUI　(命令:lui)

制御ユニット設計（真理値表）

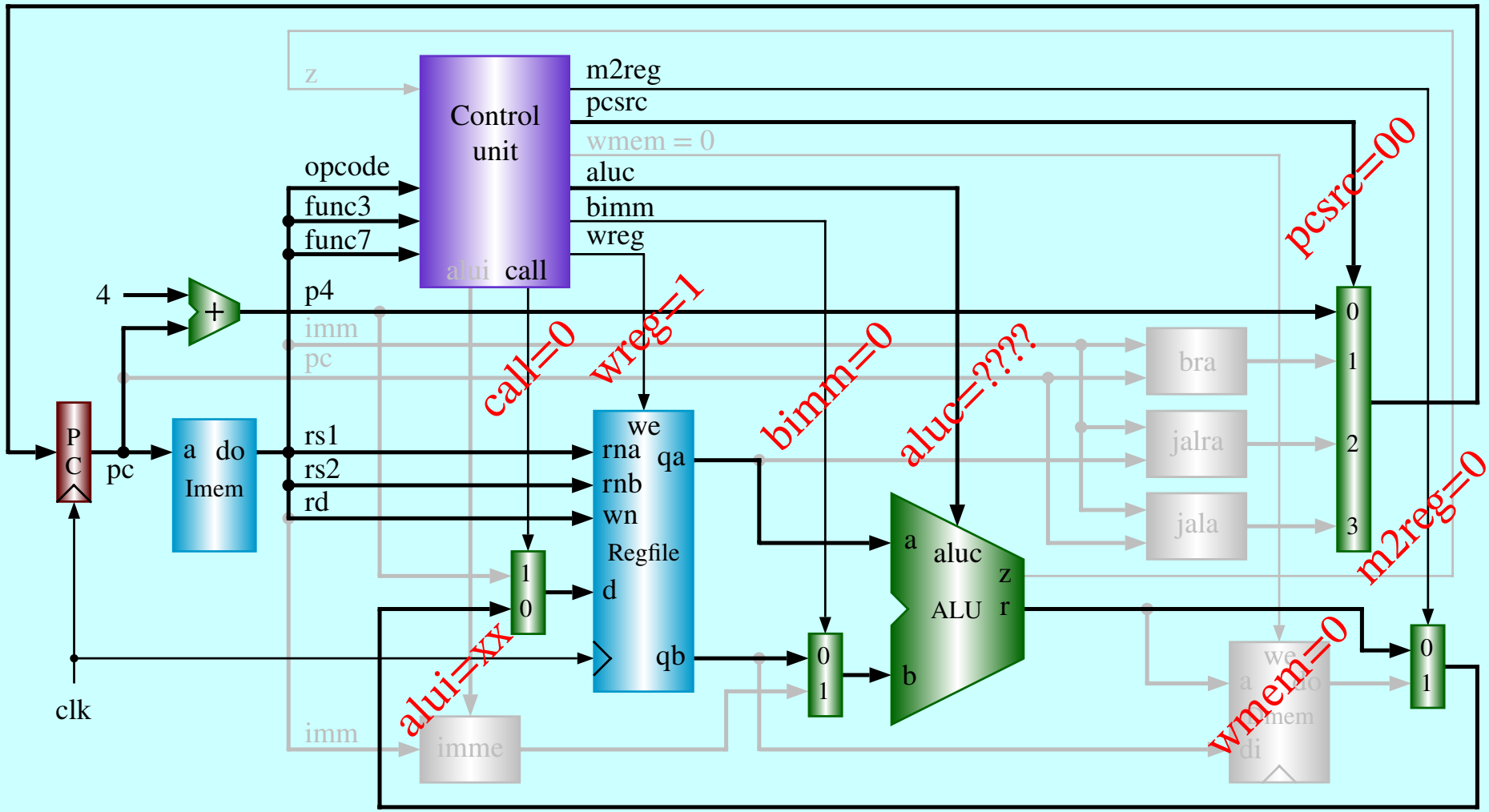
入力			出力							
#	命令	z	pcsrc[1:0]	aluc[3:0]	bimm	alui[1:0]	wreg	call	m2reg	wmem
1.	i_add	x								
2.	i_sub	x								
3.	i_slt	x								
4.	i_xor	x								
5.	i_or	x								
6.	i_and	x								
7.	i_slli	x								
8.	i_srli	x								
9.	i_srai	x								
10.	i_jalr	x								
11.	i_addi	x								
12.	i_xori	x								
13.	i_ori	x								
14.	i_andi	x								
15.	i_lw	x								
16.	i_sw	x								
17.	i_beq	0								
		1								
18.	i_bne	0								
		1								
19.	i_jal	x								
20.	i_lui	x								

RISC-V コンピュータ



CPU + 命令メモリ Imem + データメモリ Dmem

RISC-V add, sub, slt, xor, or, and



31 25	24 20	19 15	14 12	11 7	6 0
func7	rs2	rs1	func3	rd	opcode
7	5	5	3	5	7

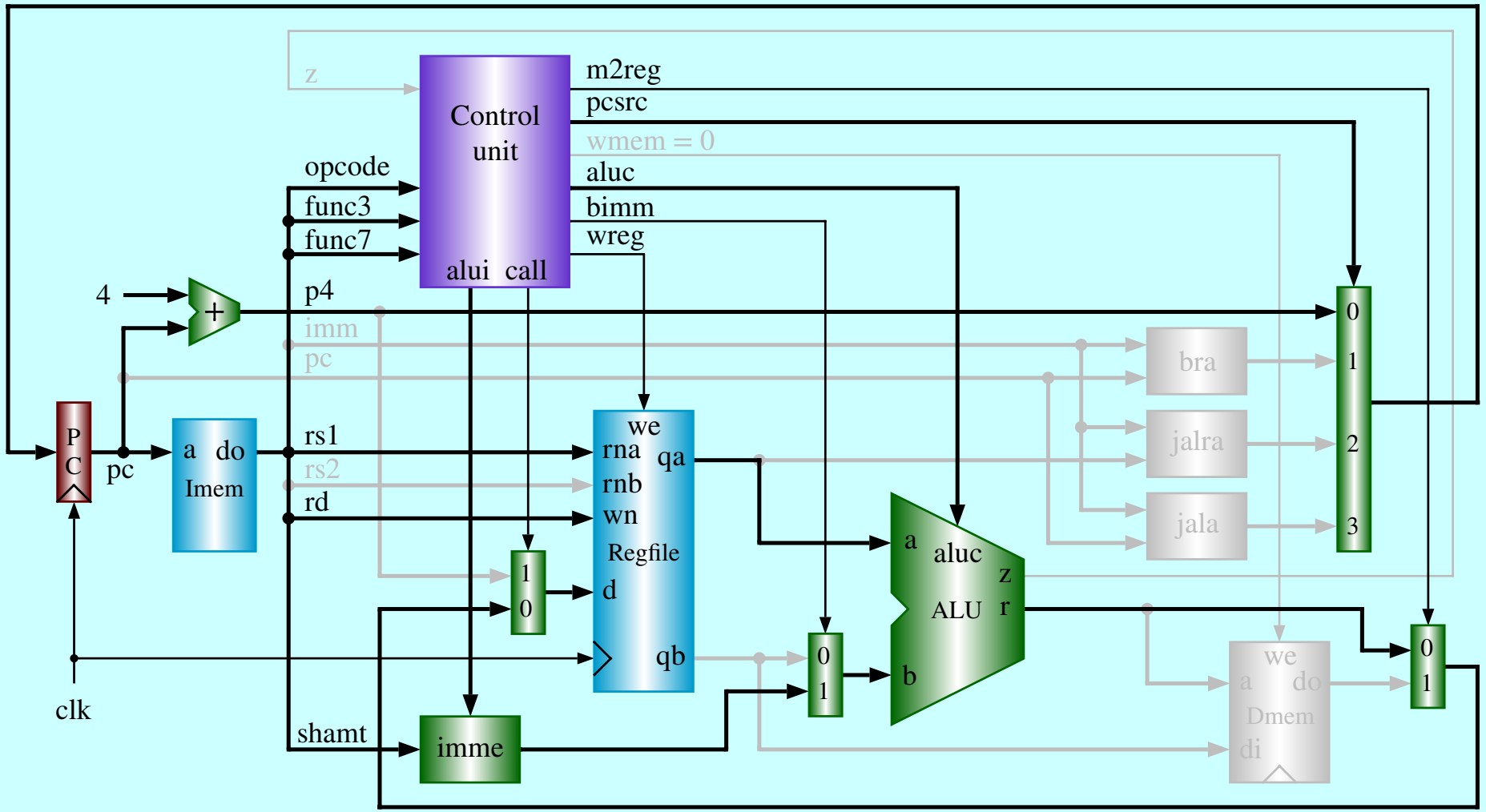
R-type

add rd, rs1, rs2 # rd ← rs1 + rs2

制御ユニット設計（真理値表）

入力			出力							
#	命令	z	pcsrc[1:0]	aluc[3:0]	bimm	alui[1:0]	wreg	call	m2reg	wmem
1.	i_add	x	0 0	x 0 0 0	0	x x	1	0	0	0
2.	i_sub	x	0 0	x 0 0 1	0	x x	1	0	0	0
3.	i_slt	x	0 0	x 0 1 0	0	x x	1	0	0	0
4.	i_xor	x	0 0	1 0 1 1	0	x x	1	0	0	0
5.	i_or	x	0 0	x 1 0 0	0	x x	1	0	0	0
6.	i_and	x	0 0	x 1 0 1	0	x x	1	0	0	0
7.	i_slli	x								
8.	i_srli	x								
9.	i_srai	x								
10.	i_jalr	x								
11.	i_addi	x								
12.	i_xori	x								
13.	i_ori	x								
14.	i_andi	x								
15.	i_lw	x								
16.	i_sw	x								
17.	i_beq	0								
		1								
18.	i_bne	0								
		1								
19.	i_jal	x								
20.	i_lui	x								

RISC-V slli, srli, srai

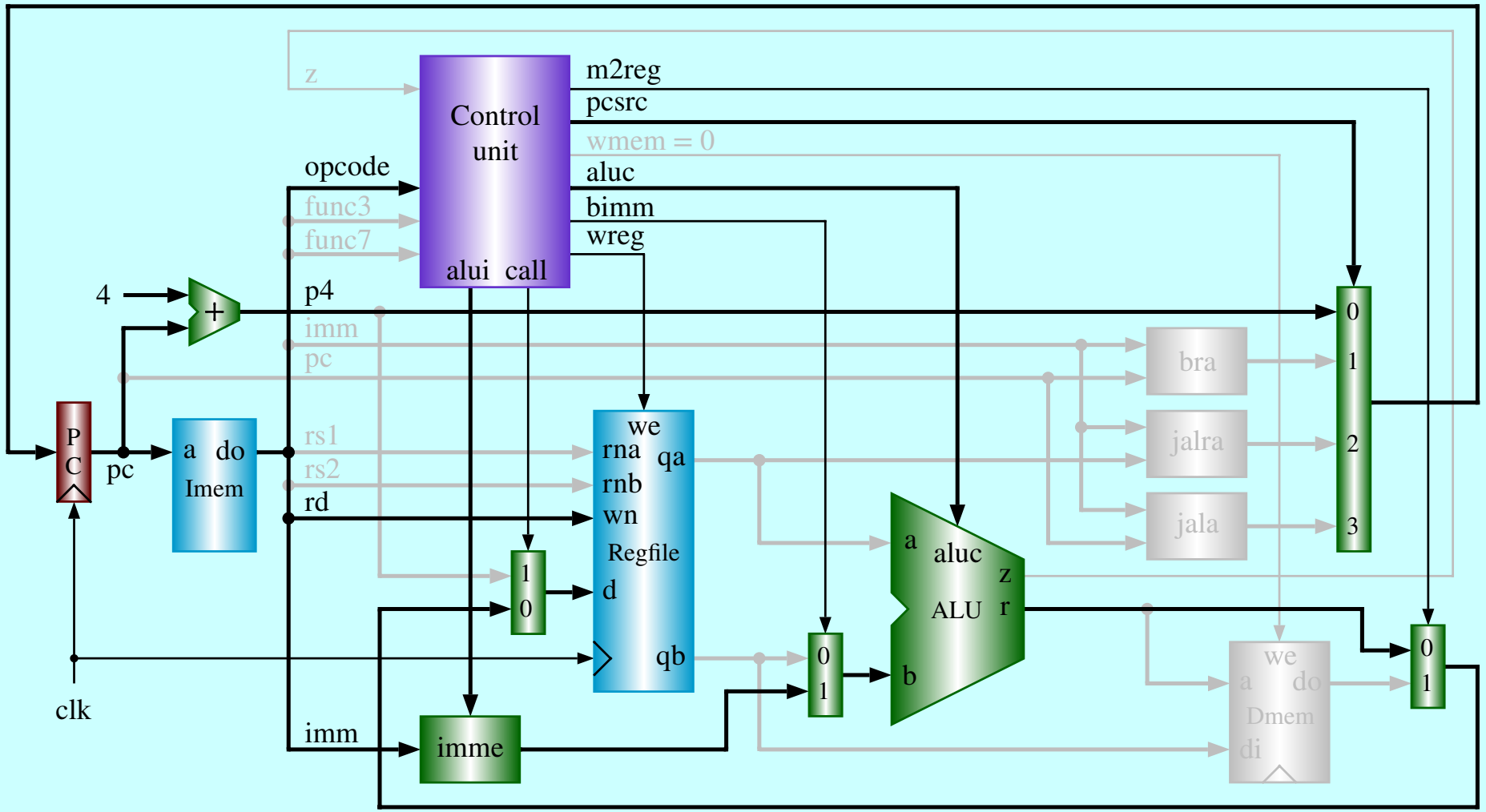


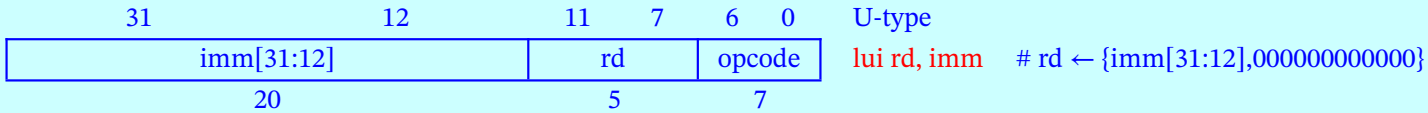
31 25	24 20	19 15	14 12	11 7	6 0
func7	shamt	rs1	func3	rd	opcode
7	5	5	3	5	7

H-type

slli rd, rs1, shamt # rd ← rs1 << shamt

RISC-V lui

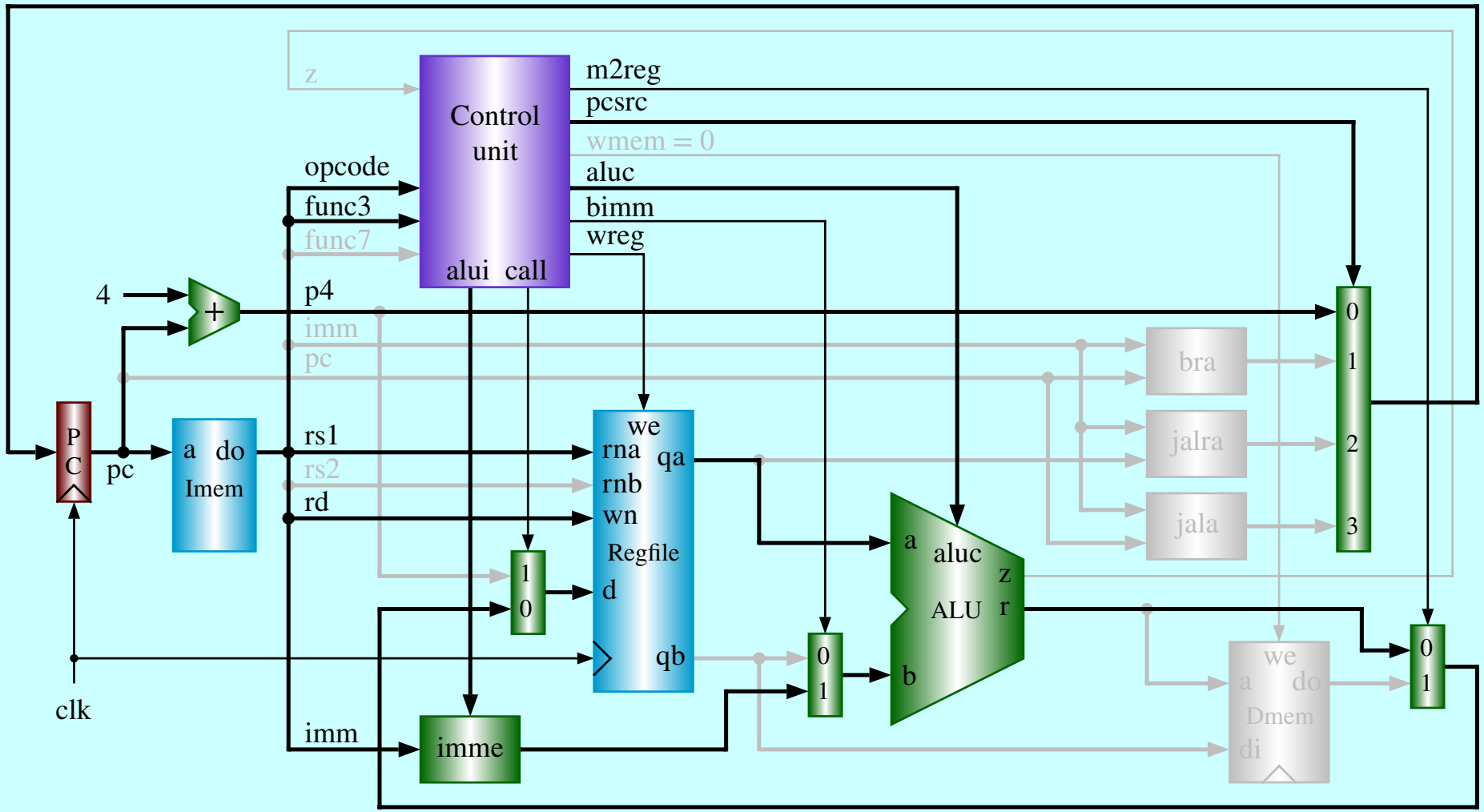




U-type

lui rd, imm # rd ← {imm[31:12],000000000000}

RISC-V addi, xori, ori, andi

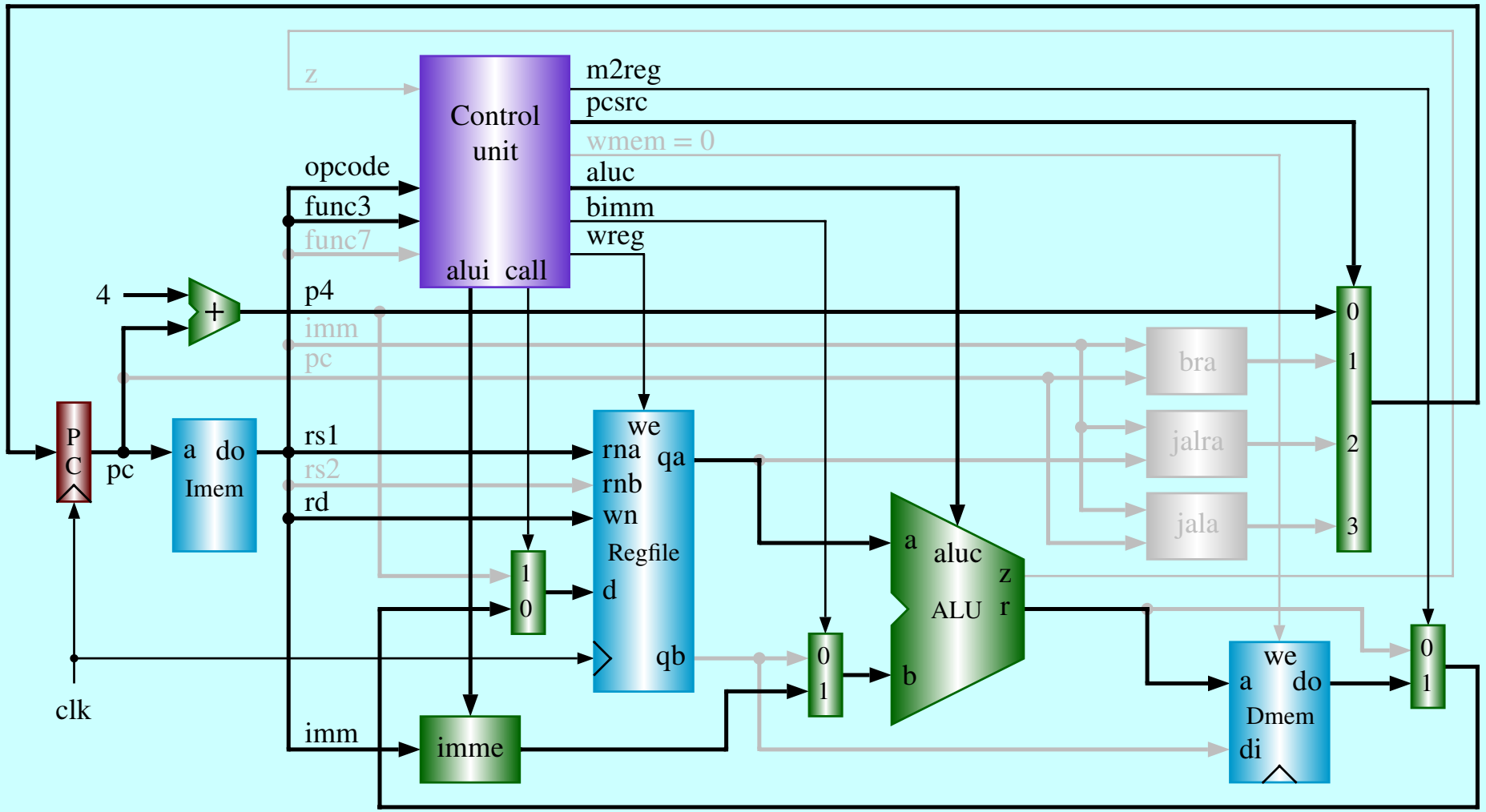


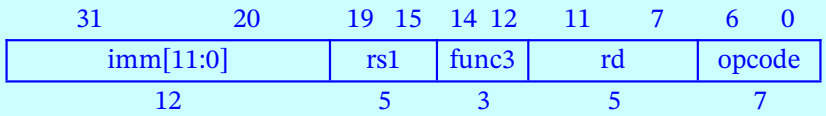
31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	func3	rd	opcode
12	5	3	5	7

I-type

addi rd, rs1, imm # rd ← rs1 + $signed(imm[11:0])

RISC-V lw

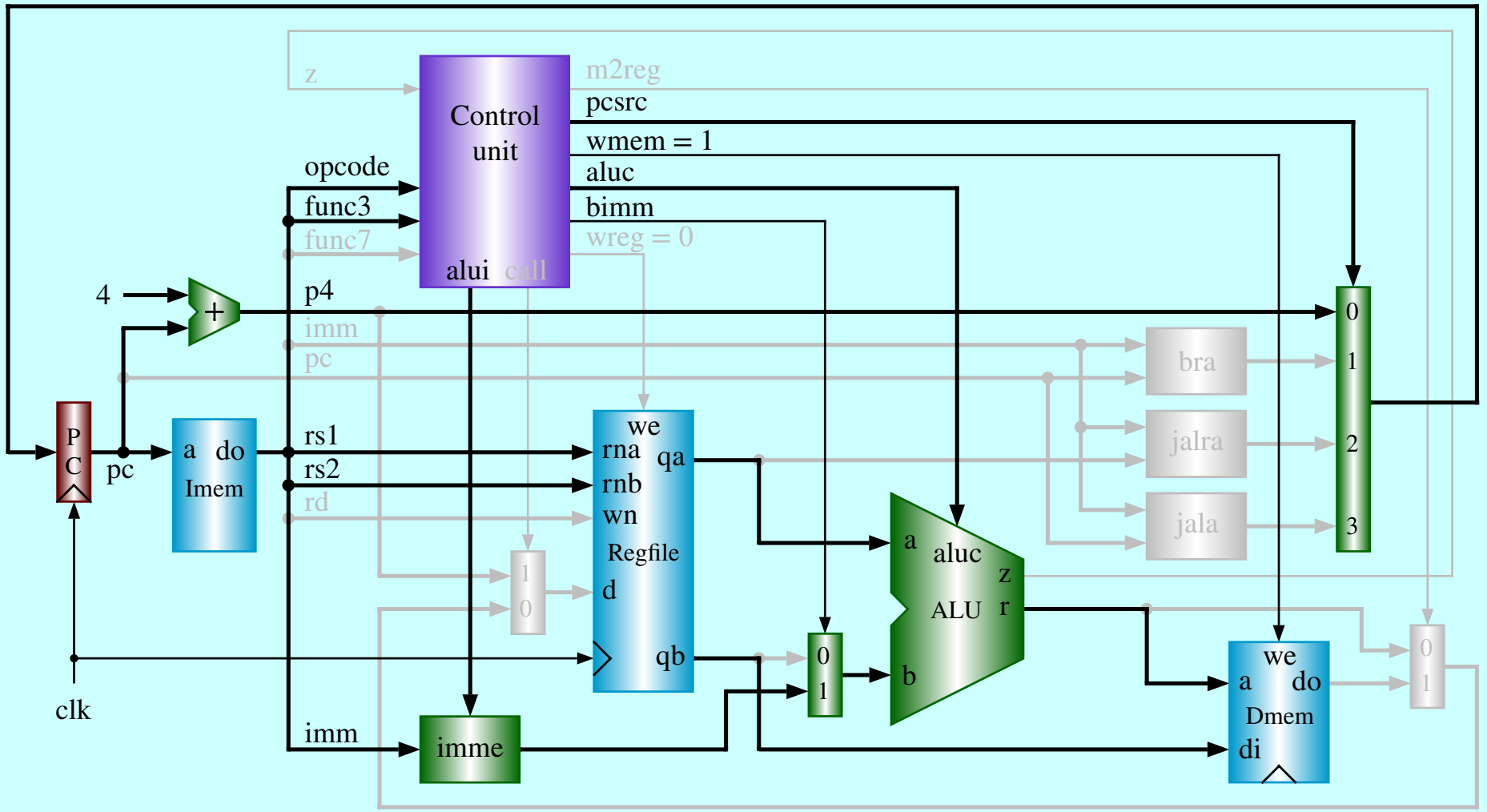




I-type

lw rd, imm(rs1) # rd ← memory[rs1+$signed(imm[11:0])]

RISC-V sw

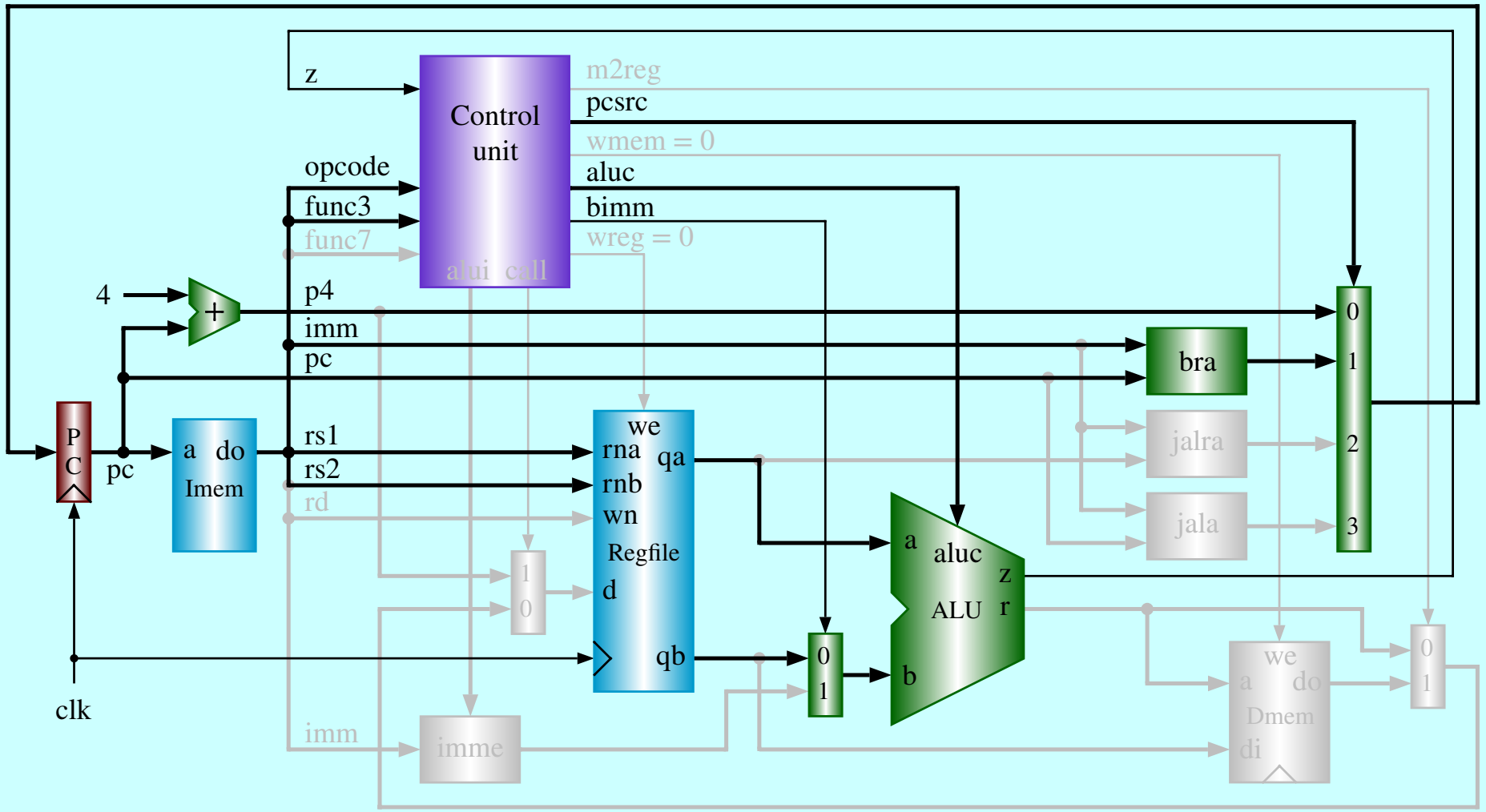


31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	func3	imm[4:0]	opcode
7	5	5	3	5	7

S-type

sw rs2, imm(rs1) # memory[rs1+$signed(imm[11:0])] ← rs2

RISC-V beq, bne

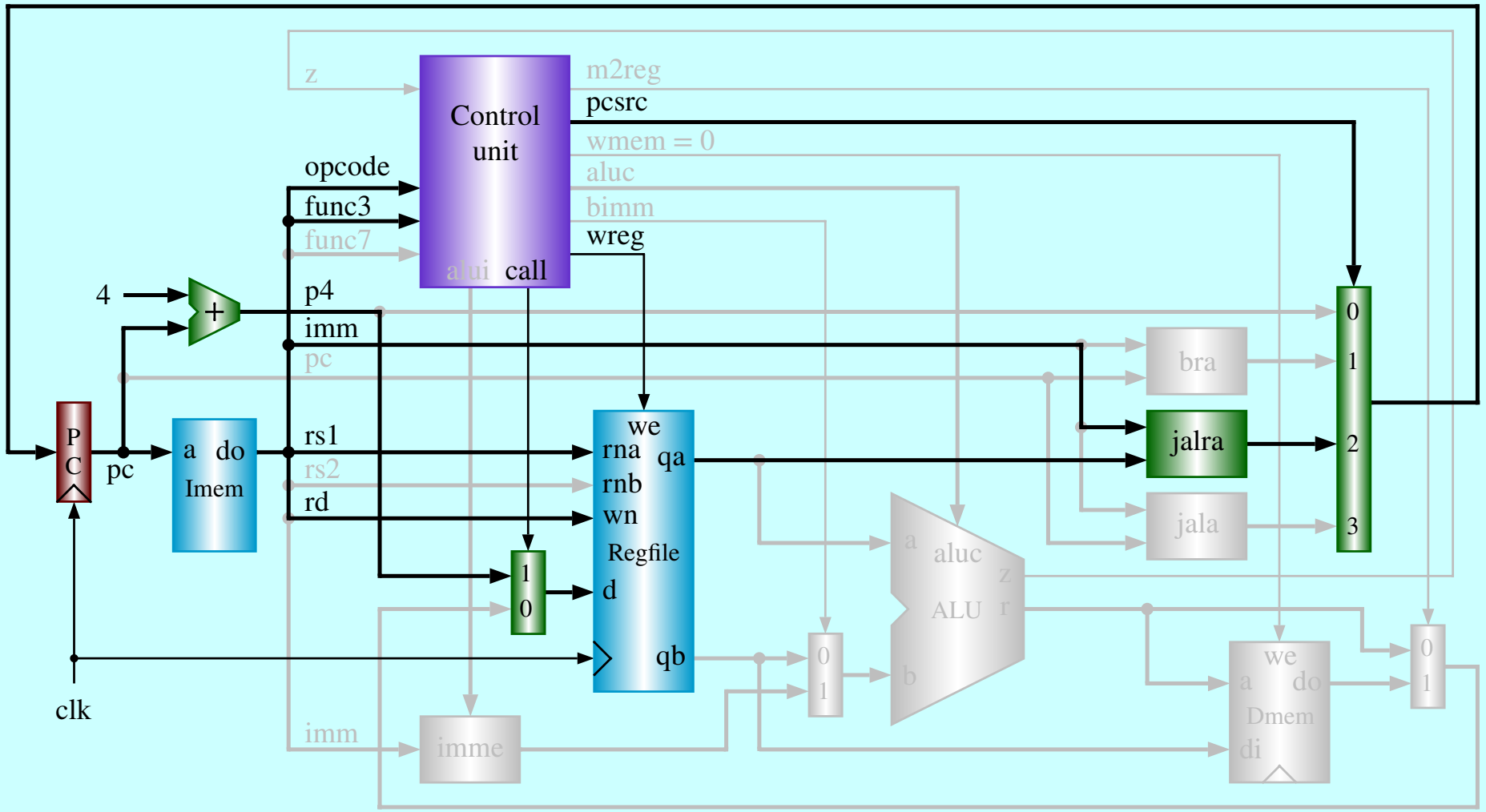


31 25	24 20	19 15	14 12	11 7	6 0
imm[12|10:5]	rs2	rs1	func3	imm[4:1|11]	opcode
7	5	5	3	5	7

B-type

beq rs1, rs2, label # if (rs1 = rs2) pc ← pc + $signed({imm[12:1],0})

RISC-V jalr

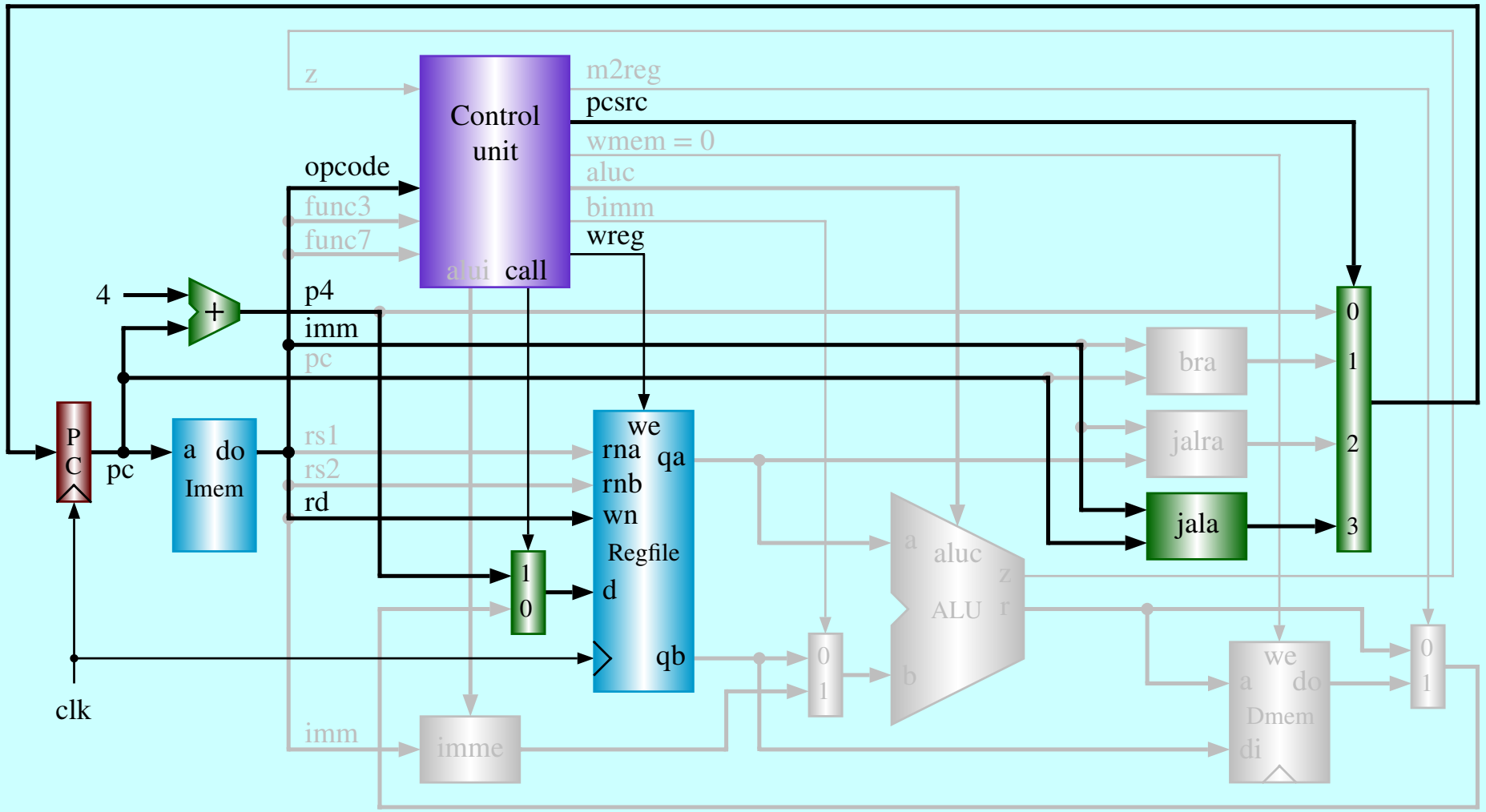


31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	func3	rd	opcode
12	5	3	5	7

I-type

jalr rd, rs1, imm # rd ← pc + 4; pc ← (rs1 + $signed(imm[11:0]) & 0xfffffffe

RISC-V jal



31	12	11	7	6	0	J-type
imm[20|10:1|11|19:12]		rd		opcode		jal rd, label # rd ← pc + 4; pc ← pc + $signed({imm[20:1],0})
20		5		7		

制御ユニット設計（真理値表）

入力			出力							
#	命令	z	pcsrc[1:0]	aluc[3:0]	bimm	alui[1:0]	wreg	call	m2reg	wmem
1.	i_add	x	0 0	x 0 0 0	0	x x	1	0	0	0
2.	i_sub	x	0 0	x 0 0 1	0	x x	1	0	0	0
3.	i_slt	x	0 0	x 0 1 0	0	x x	1	0	0	0
4.	i_xor	x	0 0	1 0 1 1	0	x x	1	0	0	0
5.	i_or	x	0 0	x 1 0 0	0	x x	1	0	0	0
6.	i_and	x	0 0	x 1 0 1	0	x x	1	0	0	0
7.	i_slli	x					1			0
8.	i_srli	x					1			0
9.	i_srai	x					1			0
10.	i_jalr	x					1			0
11.	i_addi	x					1			0
12.	i_xori	x					1			0
13.	i_ori	x					1			0
14.	i_andi	x					1			0
15.	i_lw	x					1			0
16.	i_sw	x					0			1
17.	i_beq	0					0			0
		1								
18.	i_bne	0					0			0
		1								
19.	i_jal	x	1 1	x x x x	x	x x	1	1	x	0
20.	i_lui	x					1			0

制御ユニット設計

```
module sc_cu (opcode,func7,func3,z,aluc,alui,pcsrc,m2reg,bimm,call,wreg,wmem);
    input  [6:0] opcode;
    input  [6:0] func7;
    input  [2:0] func3;
    input        z;

    output [3:0] aluc;
    output [1:0] alui;
    output [1:0] pcsrc;
    output       m2reg;
    output       bimm;
    output       call;
    output       wreg;
    output       wmem;
```

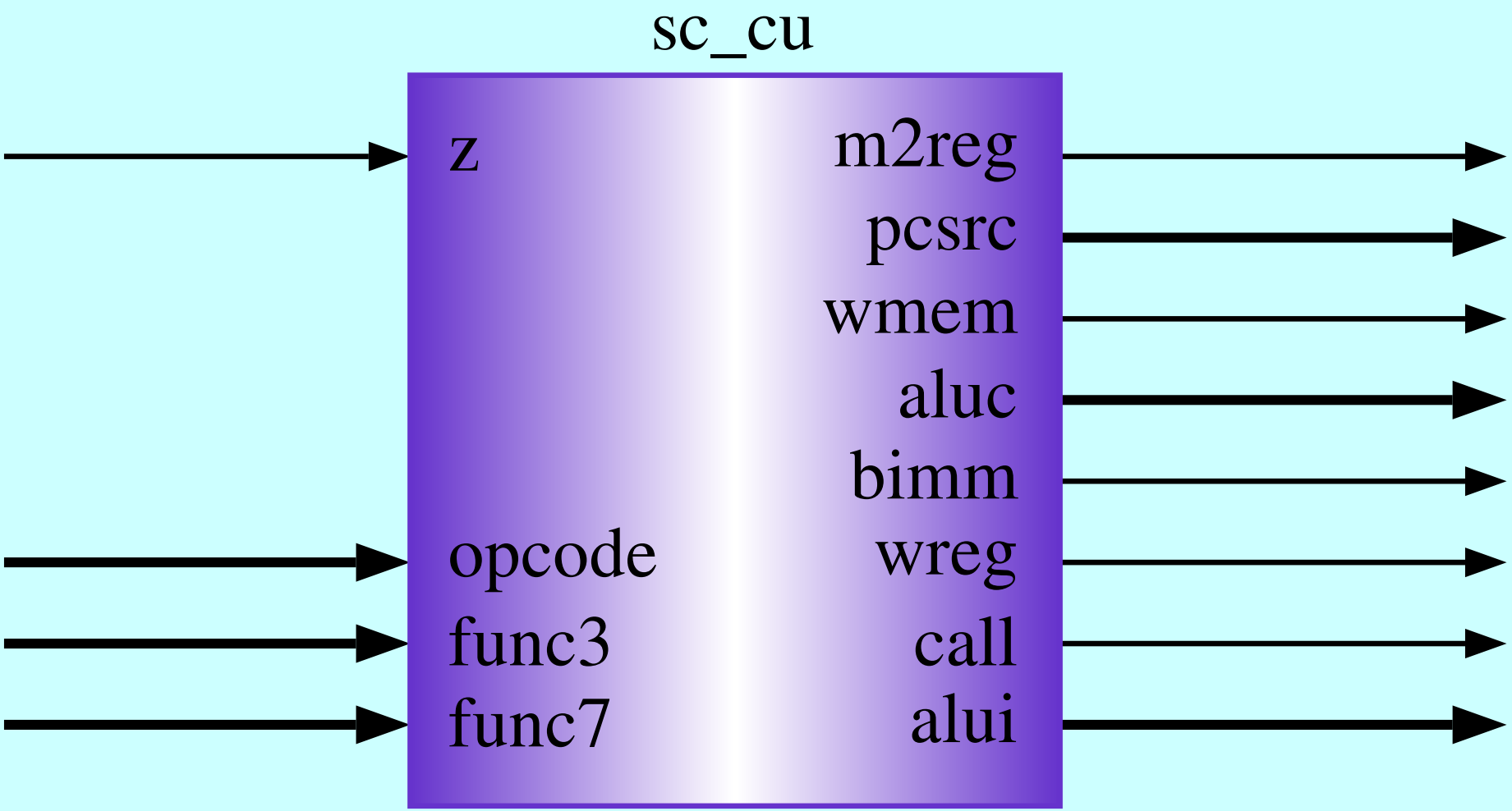
制御ユニット設計

```
// instruction decode
wire i_lui   = (opcode == 7'b0110111);
wire i_jal   = ; //
wire i_jalr  = (opcode == 7'b1100111) & (func3 == 3'b000);
wire i_beq   = ; //
wire i_bne   = ; //
wire i_lw    = ; //
wire i_sw    = ; //
wire i_addi  = ; //
wire i_xori  = ; //
wire i_ori   = ; //
wire i_andi  = ; //
wire i_slli  = (opcode == 7'b0010011) & (func3 == 3'b001) & (func7 == 7'b0000000);
wire i_srli  = ; //
wire i_srai  = ; //
wire i_add   = ; //
wire i_sub   = ; //
wire i_slt   = ; //
wire i_xor   = ; //
wire i_or    = ; //
wire i_and   = ; //
```

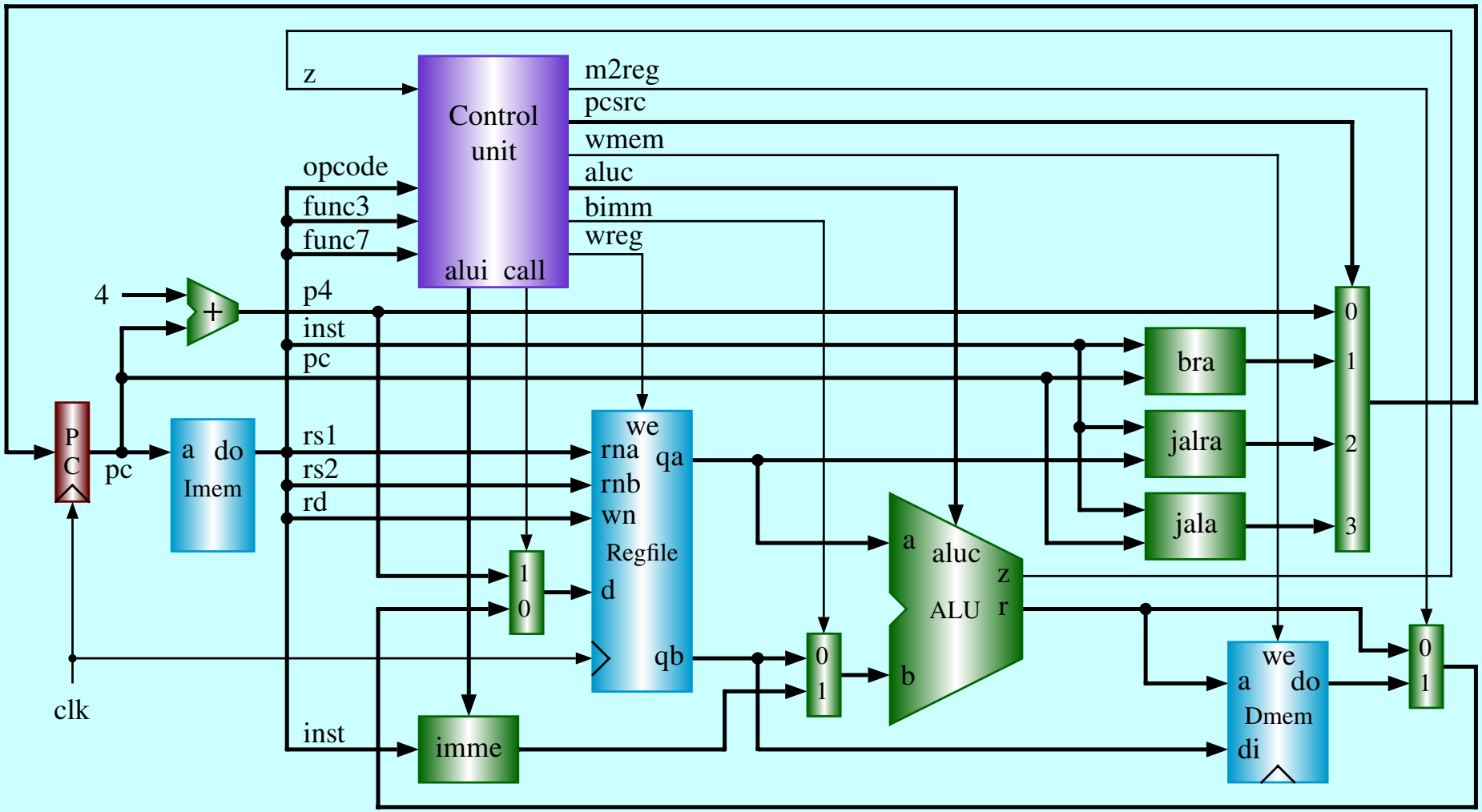
制御ユニット設計

```
    // control signals
    assign aluc[0]  = ; //
    assign aluc[1]  = ; //
    assign aluc[2]  = ; //
    assign aluc[3]  = i_xori | i_xor  | i_srai;
    assign m2reg    = i_lw;
    assign wmem     = i_sw;
    assign wreg     = i_lui  | i_jal  | i_jalr | i_lw   | i_addi | i_xori | i_ori  |
                      i_andi | i_slli | i_srli | i_srai | i_add  | i_sub  | i_slt  |
                      i_xor  | i_or   | i_and;
    assign pcsrc[0] = ; //
    assign pcsrc[1] = ; //
    assign call     = ; //
    assign alui[0]  = ; //
    assign alui[1]  = ; //
    assign bimm     = ; //
endmodule
```

CPU 制御ユニットが完成しました



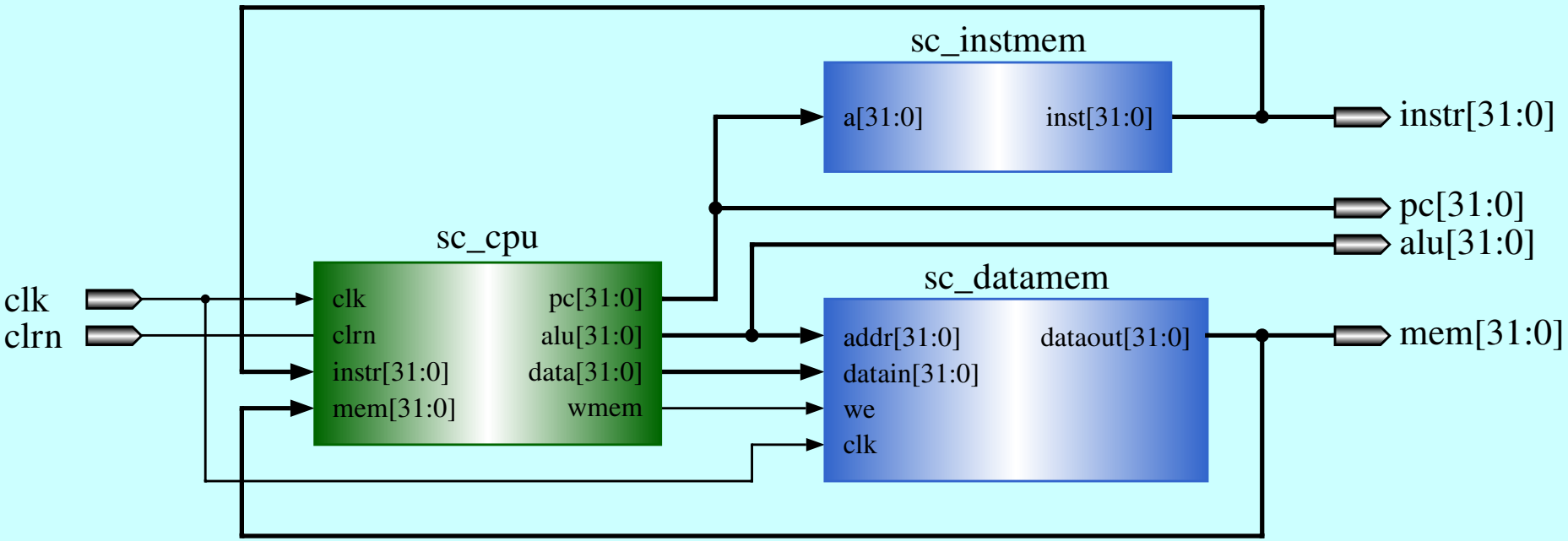
RISC-V コンピュータ



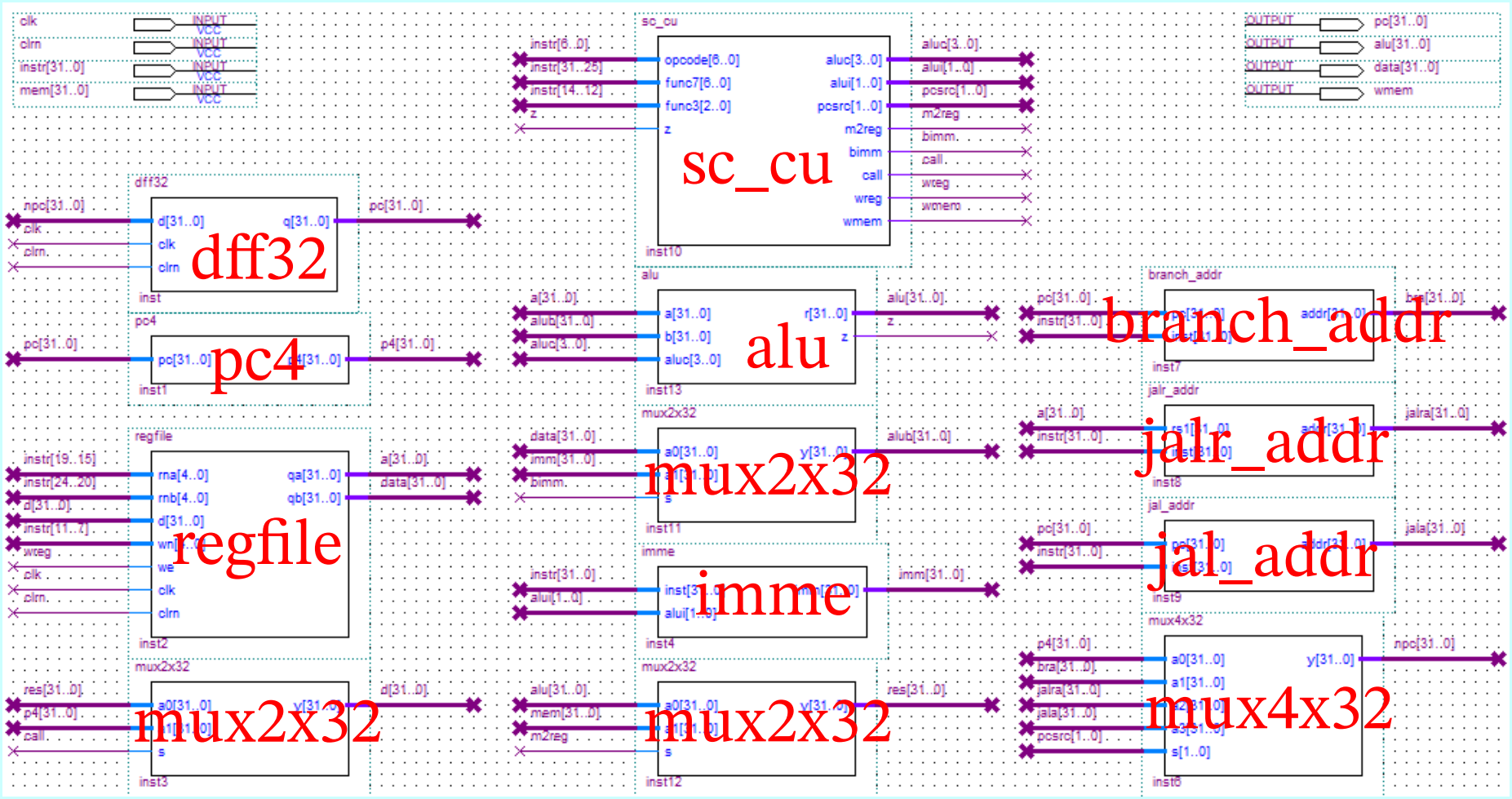
CPU + 命令メモリ Imem + データメモリ Dmem

RISC-V CPU とメモリの回路

単一サイクル RISC-V CPU + 命令メモリ + データメモリ

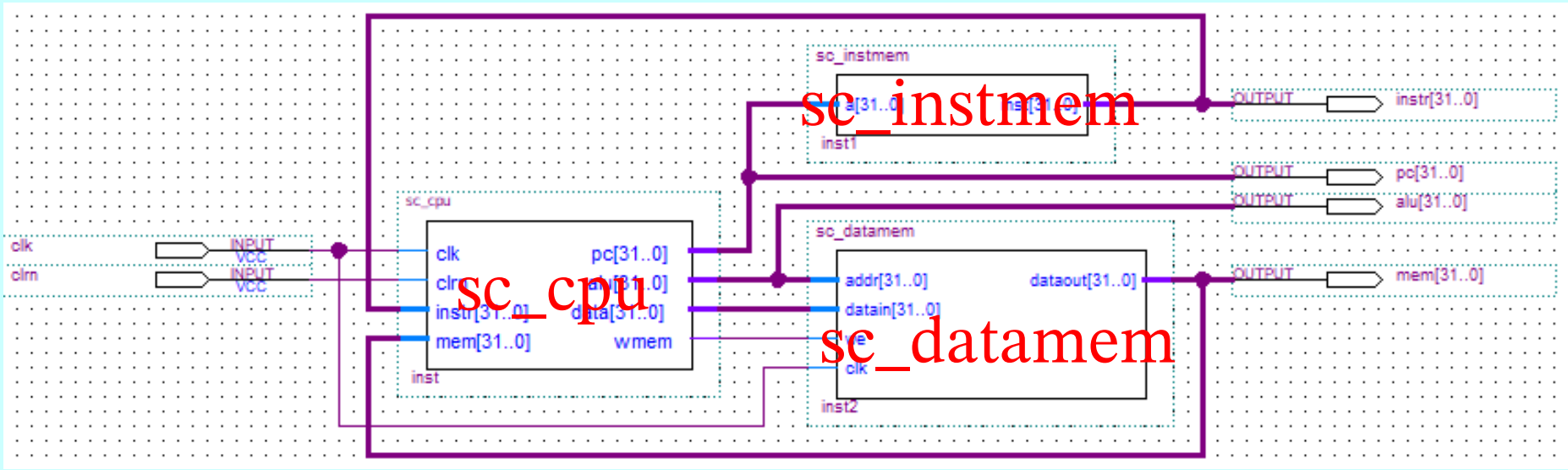


RISC-V CPU



sc_cpu の回路

RISC-V コンピュータ



sc_computer の回路

課題 V (100 点)

1. 制御ユニットの真理値表と sc_cu モジュールの Verilog HDL コードを完成しなさい（シミュレーションしなくてもよい）。
2. 単一サイクル CPU を設計したとする。CPU のクロック周波数は 200MHz である。CPU が以下のプログラムを実行するときの実行時間（ns）を計算しなさい（ループに注意）。

```
.text
main:
    lui  x4, %hi(array)      # address of array[0]
    addi x4, x4, %lo(array) # address of array[0]
    add  x2, x0, x0          # sum = 0
    addi x5, x0, 5           # counter = 5
loop:
    lw   x8, 0(x4)           # load array[i] from memory
    add  x2, x2, x8          # sum = sum + array[i]
    addi x4, x4, 4           # address + 4
    addi x5, x5, -1          # counter--
    bne  x5, x0, loop        # if counter != 0, go to loop
    sw   x2, 0(x4)
.data
array: .word 6, 5, -3, 9, 7, 0
.end
```