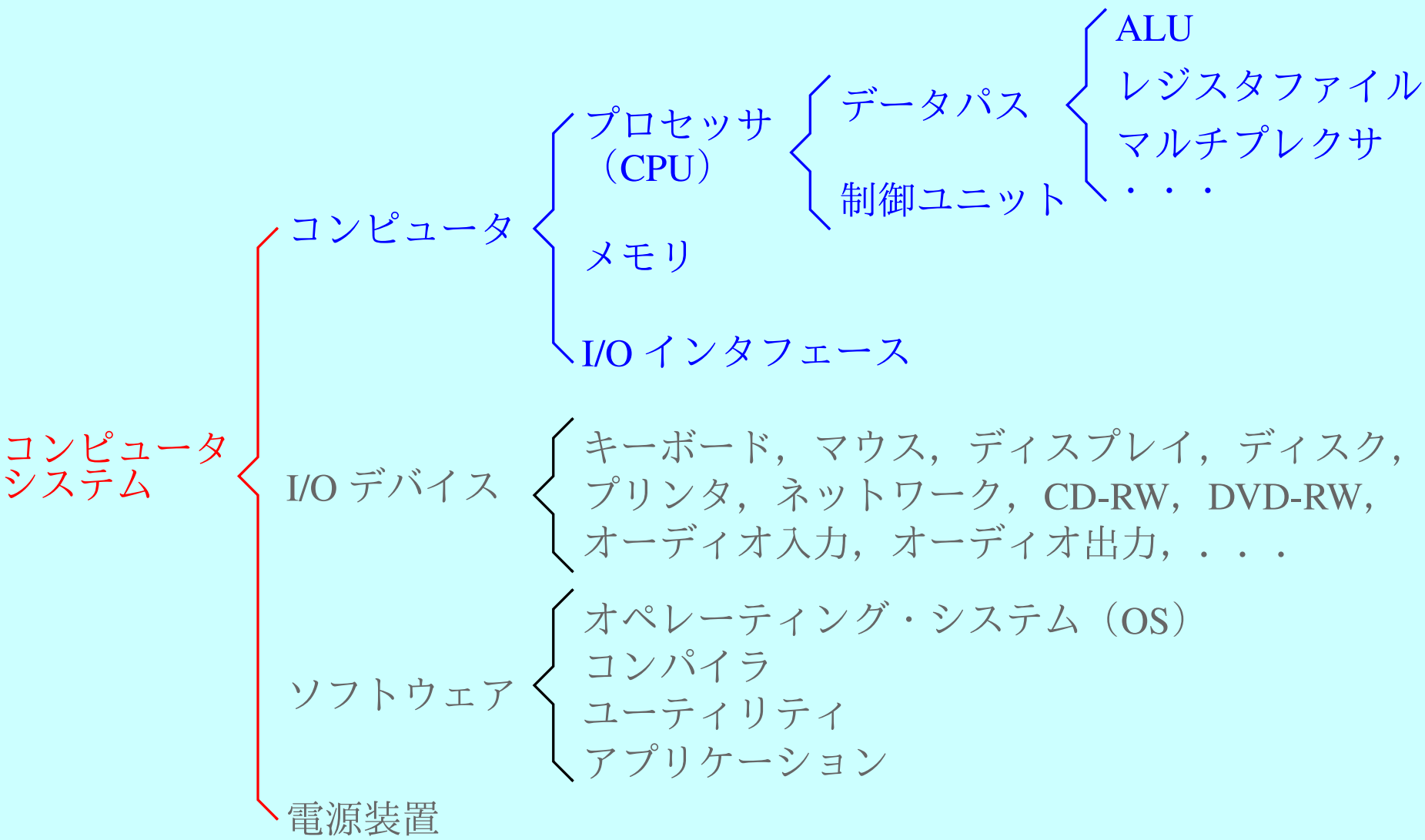
コンピュータ構成と設計（７）

マルチサイクル CPU 設計

李　亜民

2022 年 11 月 7 日 (月)

コンピュータとコンピュータシステム



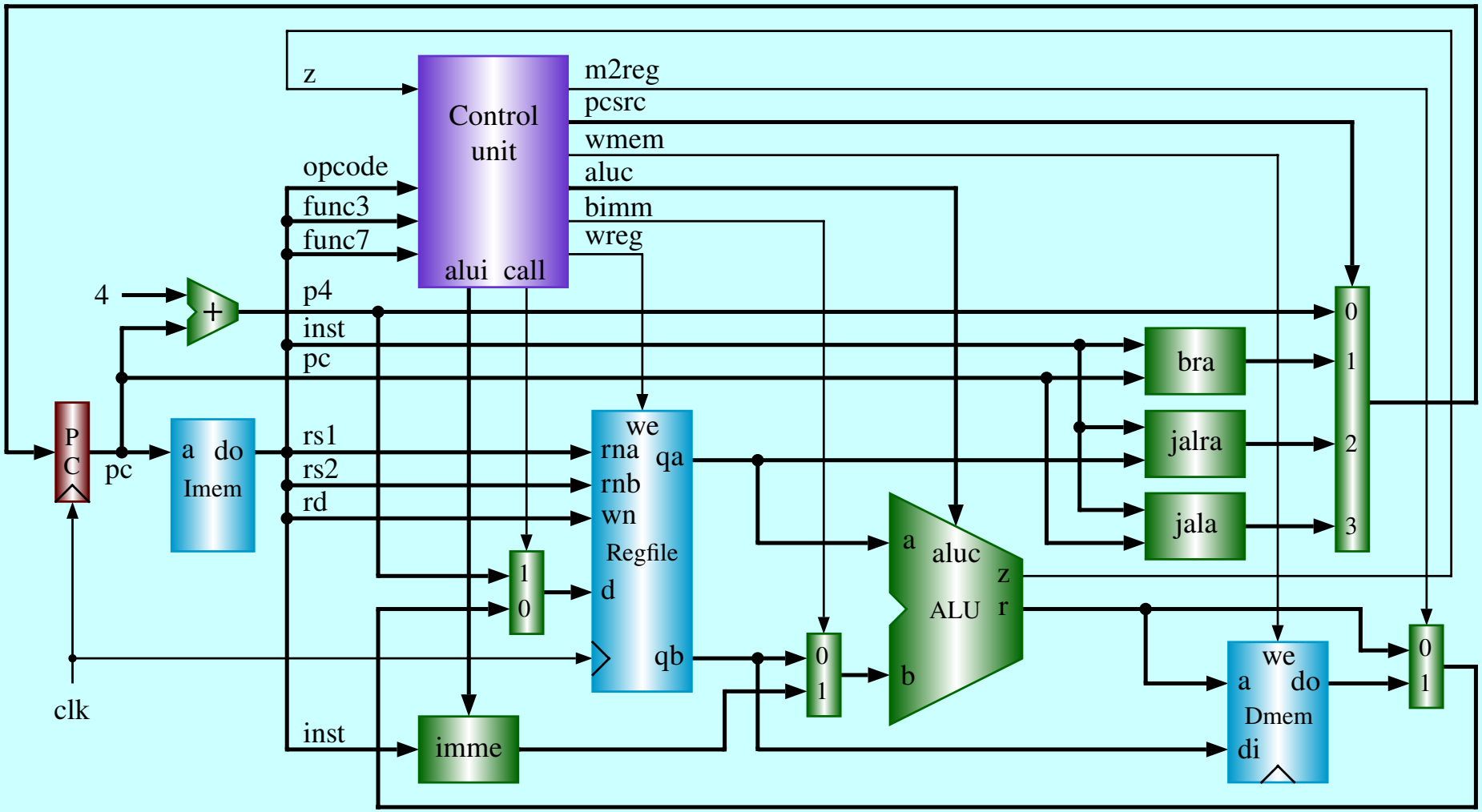
20 RISC-V 命令のまとめ

```
 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

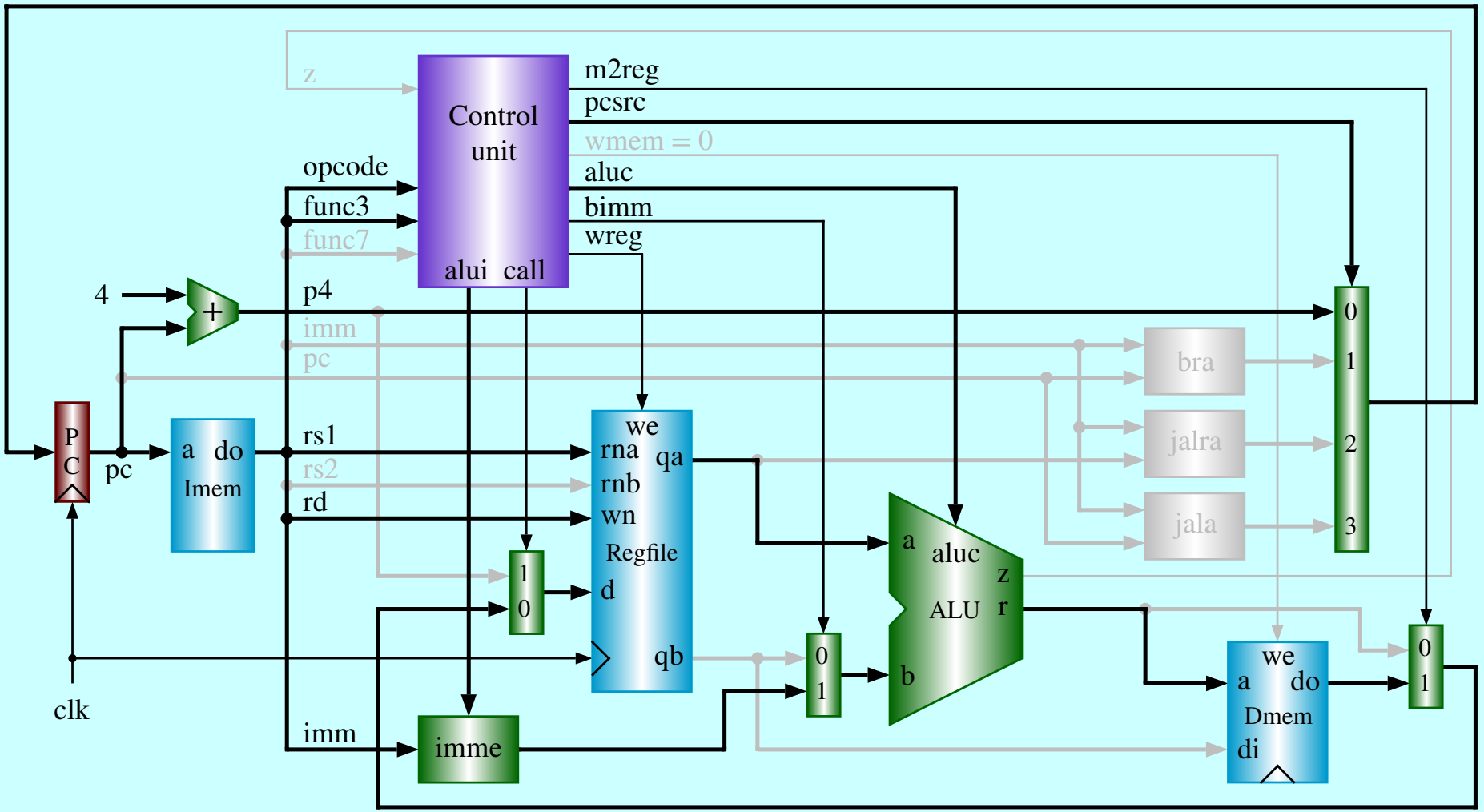
0000000	rs2	rs1	000	rd	0110011	1. add
0100000	rs2	rs1	000	rd	0110011	2. sub
0000000	rs2	rs1	010	rd	0110011	3. slt
0000000	rs2	rs1	100	rd	0110011	4. xor
0000000	rs2	rs1	110	rd	0110011	5. or
0000000	rs2	rs1	111	rd	0110011	6. and
0000000	shamt	rs1	001	rd	0010011	7. slli
0000000	shamt	rs1	101	rd	0010011	8. srli
0100000	shamt	rs1	101	rd	0010011	9. srai
imm[11:0]		rs1	000	rd	1100111	10. jalr
imm[11:0]		rs1	000	rd	0010011	11. addi
imm[11:0]		rs1	100	rd	0010011	12. xori
imm[11:0]		rs1	110	rd	0010011	13. ori
imm[11:0]		rs1	111	rd	0010011	14. andi
imm[11:0]		rs1	010	rd	0000011	15. lw
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	16. sw
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	17. beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	18. bne
imm[20|10:1|11|19:12]				rd	1101111	19. jal
imm[31:12]				rd	0110111	20. lui

RISC-V 単一サイクルコンピュータ



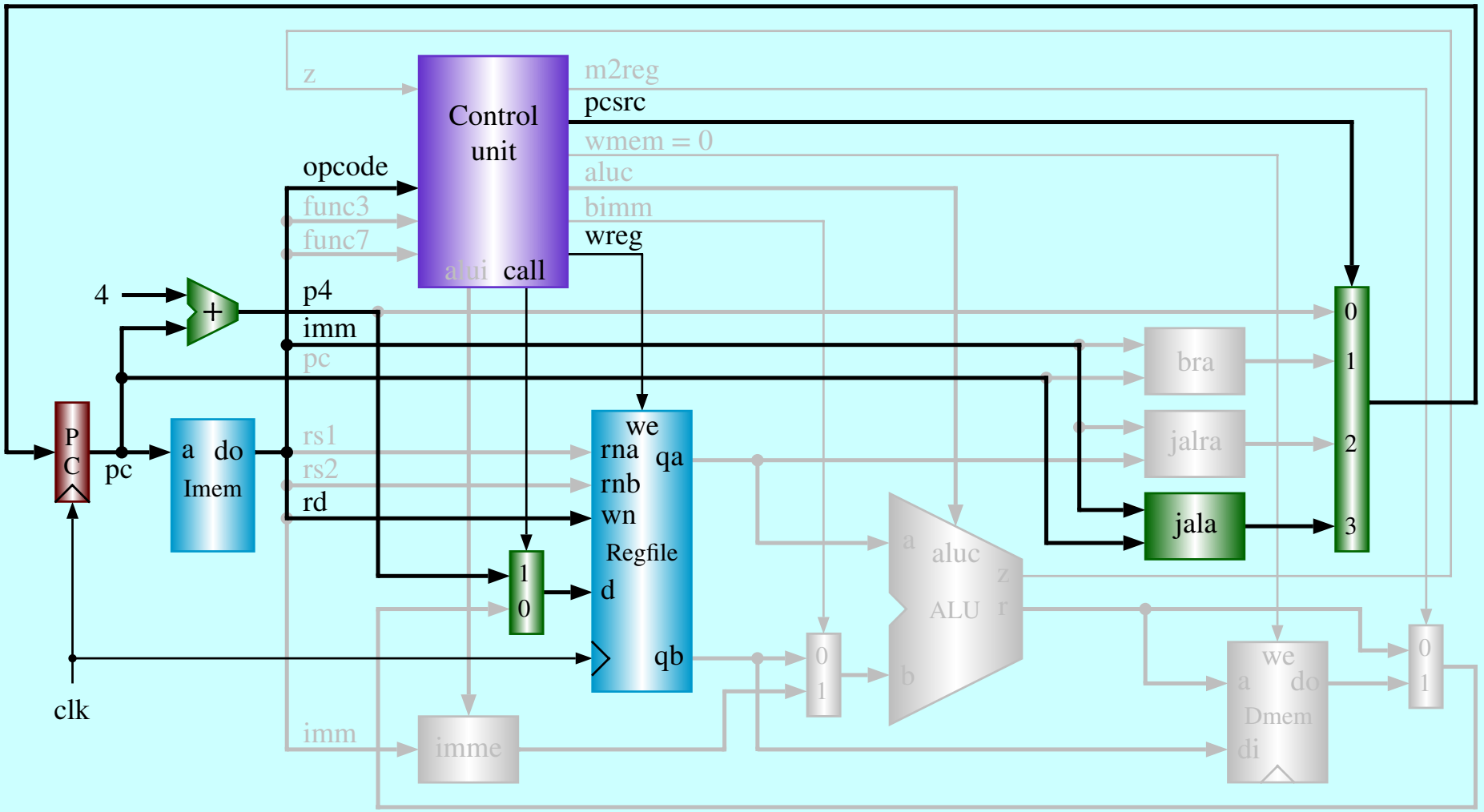
CPU + 命令メモリ Imem + データメモリ Dmem

単一サイクル CPU: lw のための回路



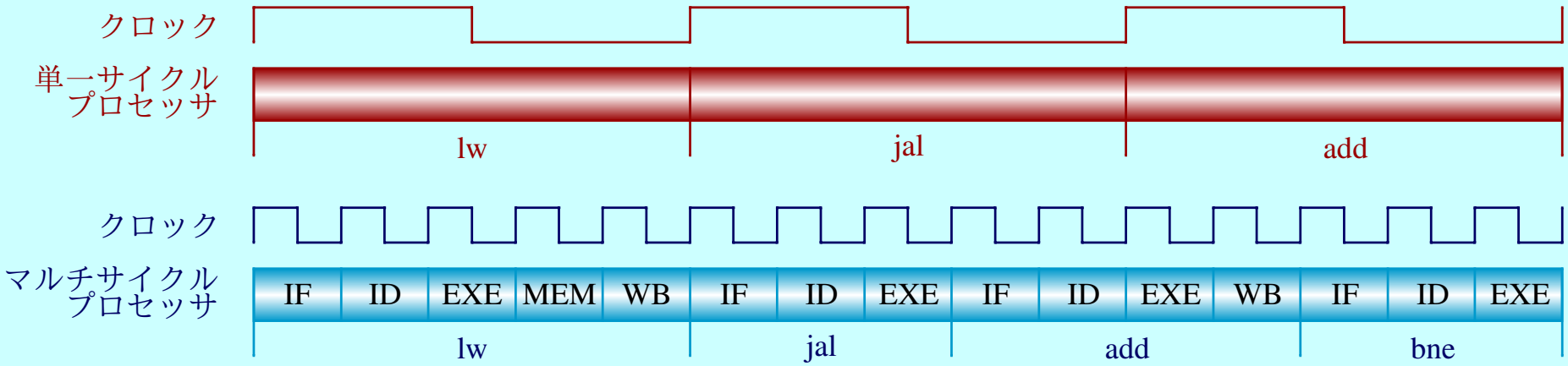
lw rd, imm(rs1)

単一サイクル CPU: jal のための回路



jal addr # lw 命令より早く終わることができる

マルチサイクル RISC-V CPU

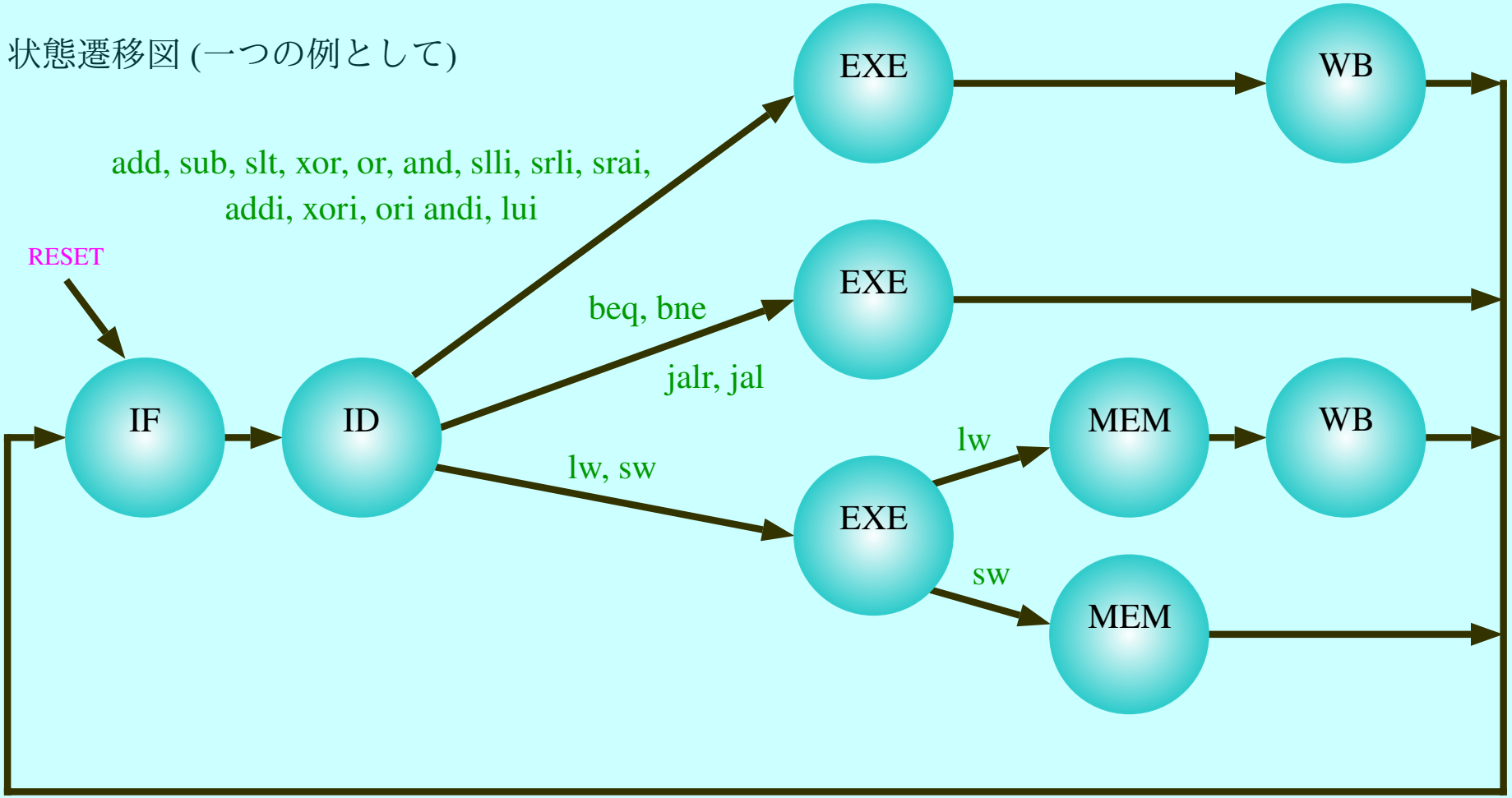


マルチサイクル方式は、

- 1 命令の実行を（同程度の処理量の）複数小作業に分割する
  - 各小作業を 1 クロックサイクルで処理する
  - 1 命令は複数クロックサイクルで実行される
- クロック・サイクル時間を短くする（動作周波数を高くする）

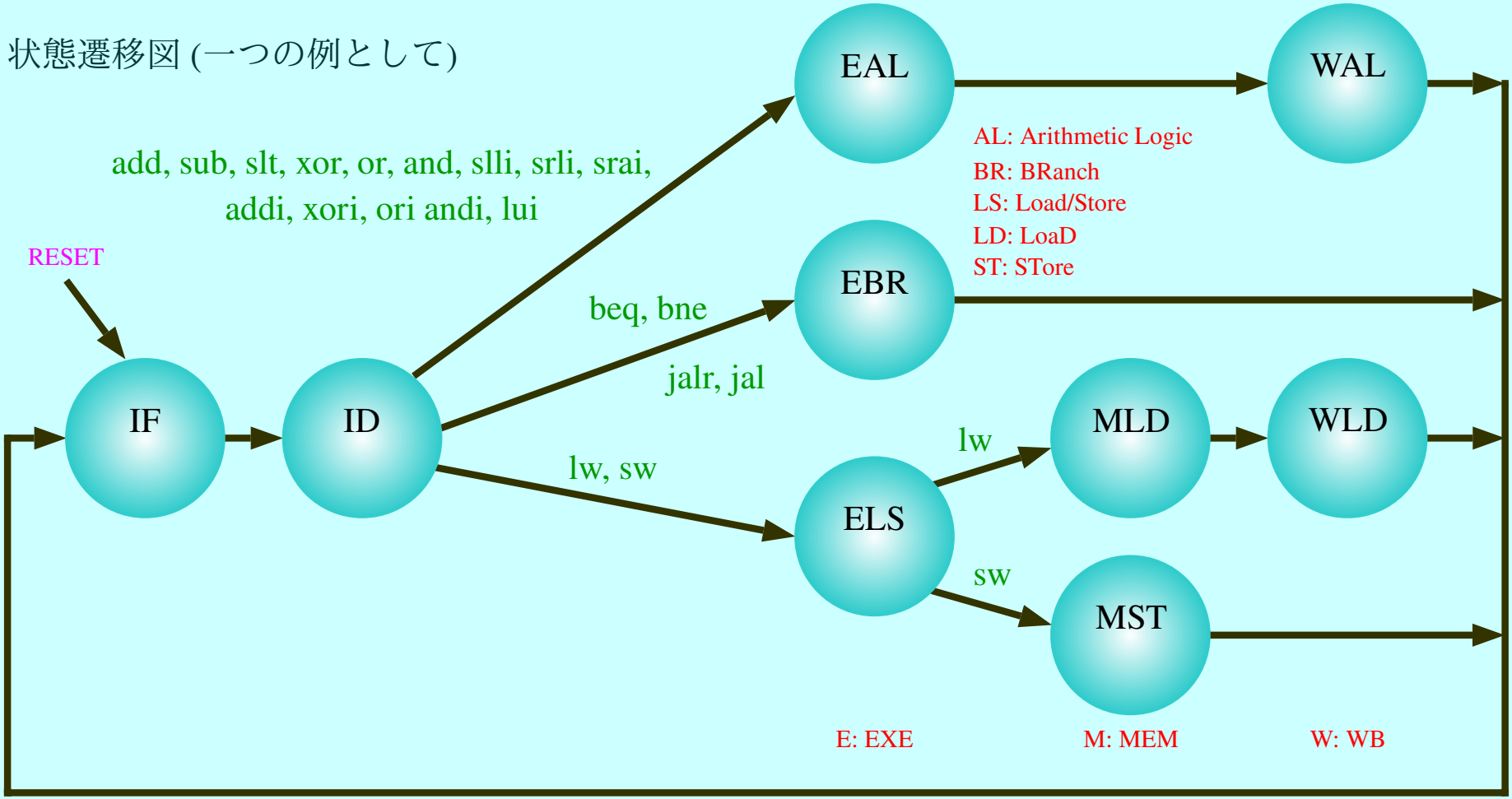
IF – 命令フェッチ (Instruction Fetch)
ID – 命令デコード (Instruction Decode)
EXE – 実行 (EXEcution)
MEM – メモリ・アクセス (MEMory access)
WB – 書き込み (Write Back)

マルチサイクルの実装：有限状態機械



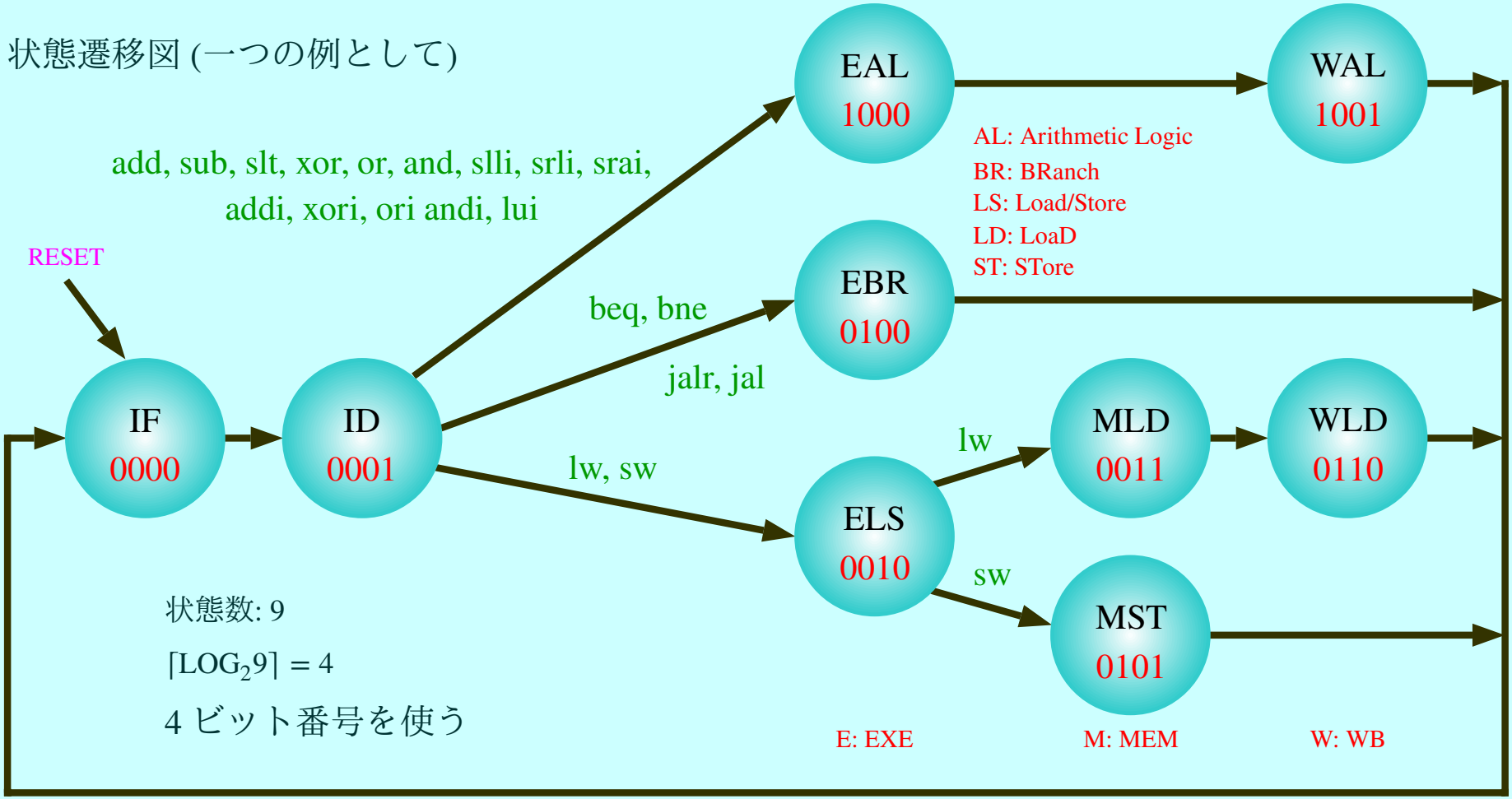
jalr, jal, beq, bne: 3 サイクル、lw: 5 サイクル、それ以外: 4 サイクル

各状態にユニークな名前を付ける



jalr, jal, beq, bne: 3 サイクル、lw: 5 サイクル、それ以外: 4 サイクル

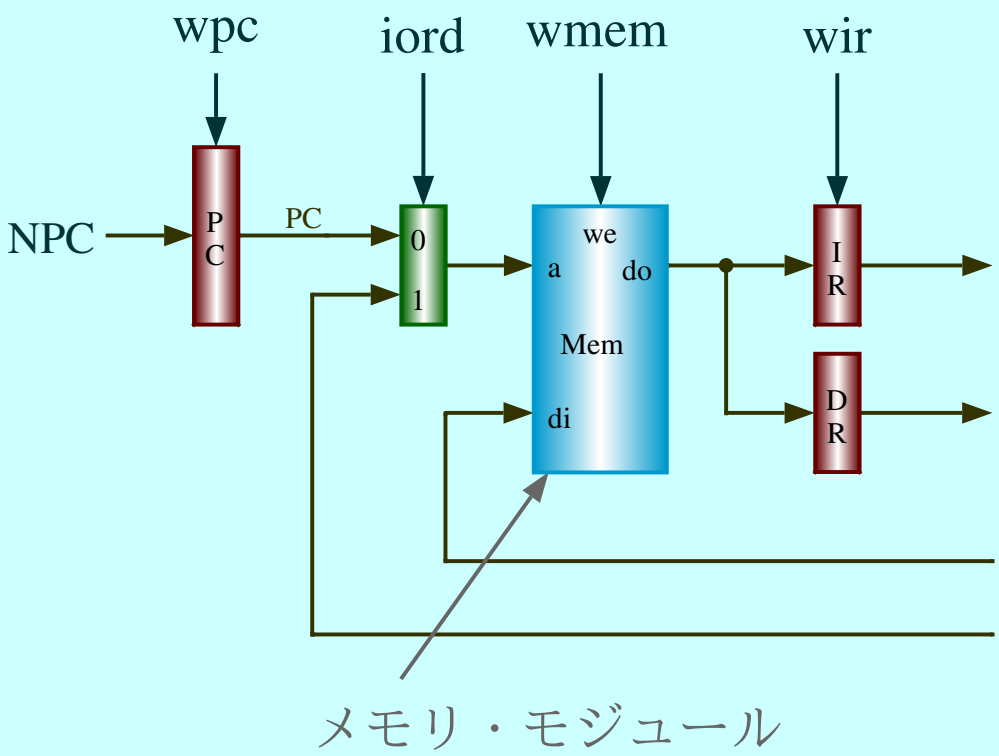
各状態にユニークな番号を付ける



jalr, jal, beq, bne: 3 サイクル、lw: 5 サイクル、それ以外: 4 サイクル

マルチサイクル CPU — メモリひとつ

- 命令用とデータ用に共有されたメモリ・モジュールをひとつ用いる
- レジスタ IR と DR を用いる [1]



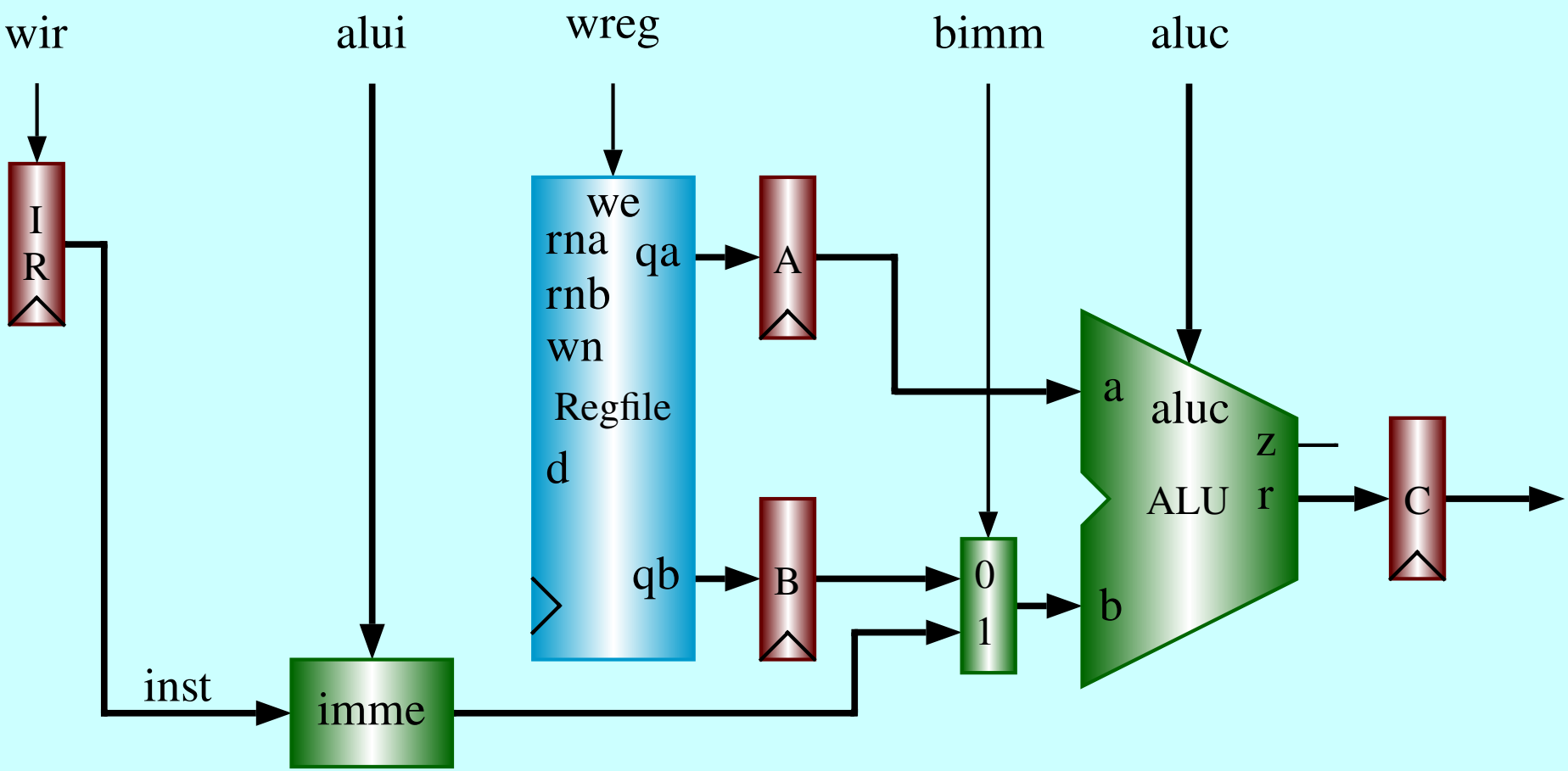
wpc, wmem, wir: Write Enable

iord: Select for memory address

[1] IR は Instruction Register, DR は Data Register の略

マルチサイクル CPU — レジスタ ABC

レジスタ A，B，C を用いる



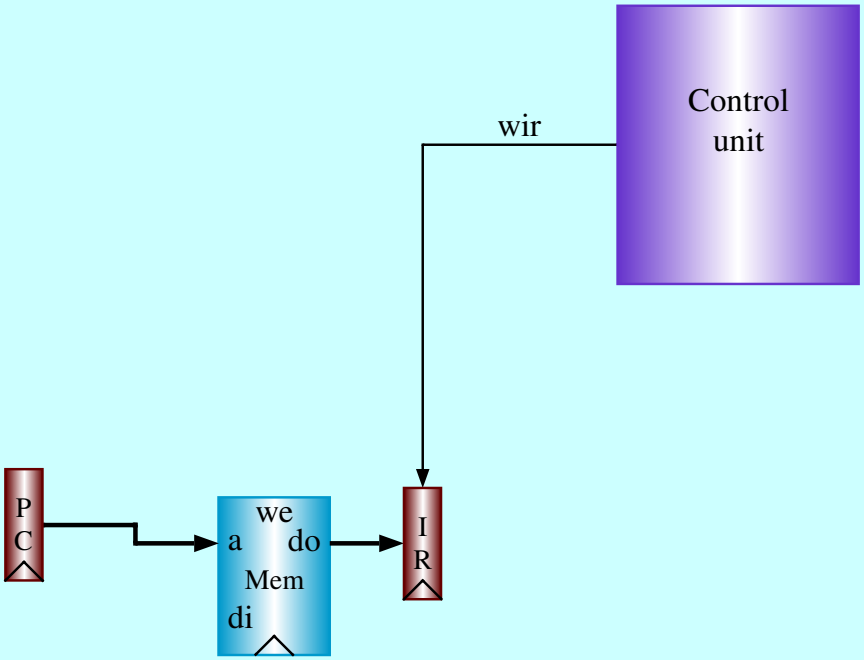
各状態に各命令の操作

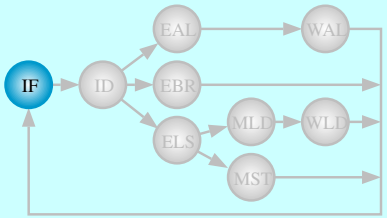
命令	IF	ID	EAL	WAL
add			C ← A + B	Reg[rd] ← C; PC ← PC + 4
sub			C ← A − B	Reg[rd] ← C; PC ← PC + 4
slt			C ← A < B	Reg[rd] ← C; PC ← PC + 4
xor			C ← A ∧ B	Reg[rd] ← C; PC ← PC + 4
or			C ← A | B	Reg[rd] ← C; PC ← PC + 4
and			C ← A & B	Reg[rd] ← C; PC ← PC + 4
slli			C ← B << sa	Reg[rd] ← C; PC ← PC + 4
srli			C ← B >> sa	Reg[rd] ← C; PC ← PC + 4
srai			C ← B >>> sa	Reg[rd] ← C; PC ← PC + 4
addi		A ← Reg[rs1]	C ← A + imm	Reg[rd] ← C; PC ← PC + 4
xori	IR ← mem[PC]	B ← Reg[rs2]	C ← A ∧ imm	Reg[rd] ← C; PC ← PC + 4
ori			C ← A | imm	Reg[rd] ← C; PC ← PC + 4
andi			C ← A & imm	Reg[rd] ← C; PC ← PC + 4
lui			C ← imm << 12	Reg[rd] ← C; PC ← PC + 4

命令	EBR
beq	if (A==B) PC←bra; else PC←PC+4
bne	if (A!=B) PC←bra; else PC←PC+4
jalr	Reg[rd] ← PC + 4; PC ← jalra
jal	Reg[rd] ← PC + 4; PC ← jala

命令	ELS	MLD	WLD
lw		DR ← mem[C]	Reg[rd] ← DR; PC ← PC + 4
命令		MST	
sw	C ← A + imm	mem[C] ← B; PC ← PC + 4	

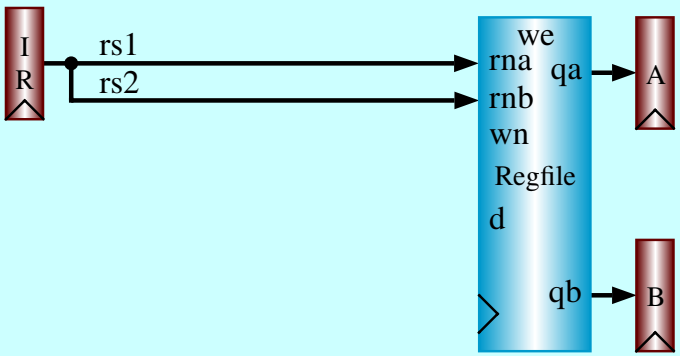
RISC-V add, ori: 1. IF 状態

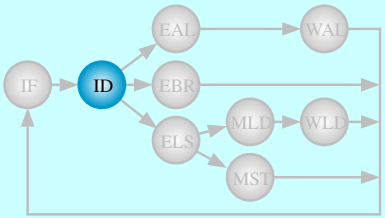




IR ← mem[PC];

RISC-V add, ori: 2. ID 状態

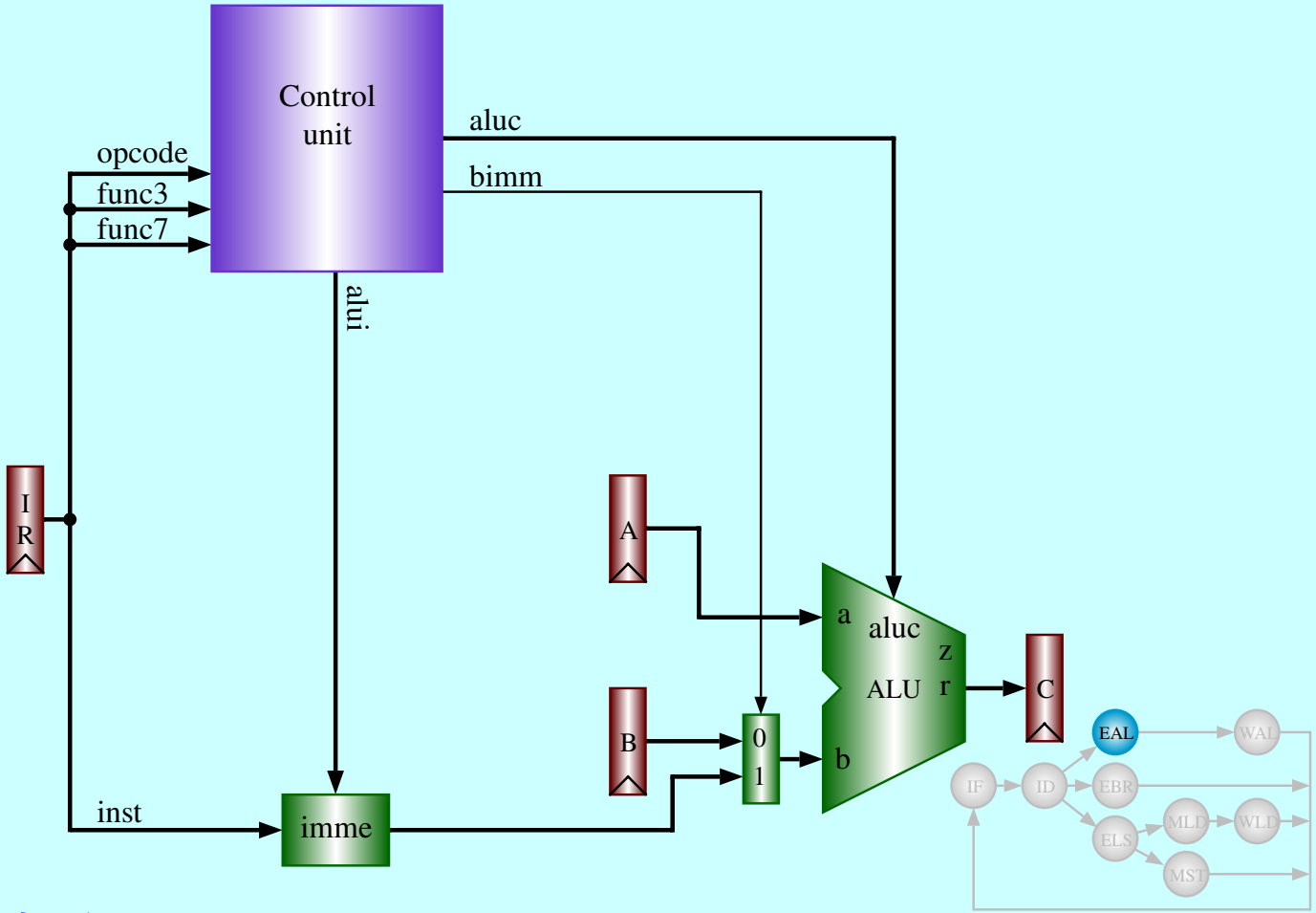




A ← Reg[rs1];

B ← Reg[rs2];

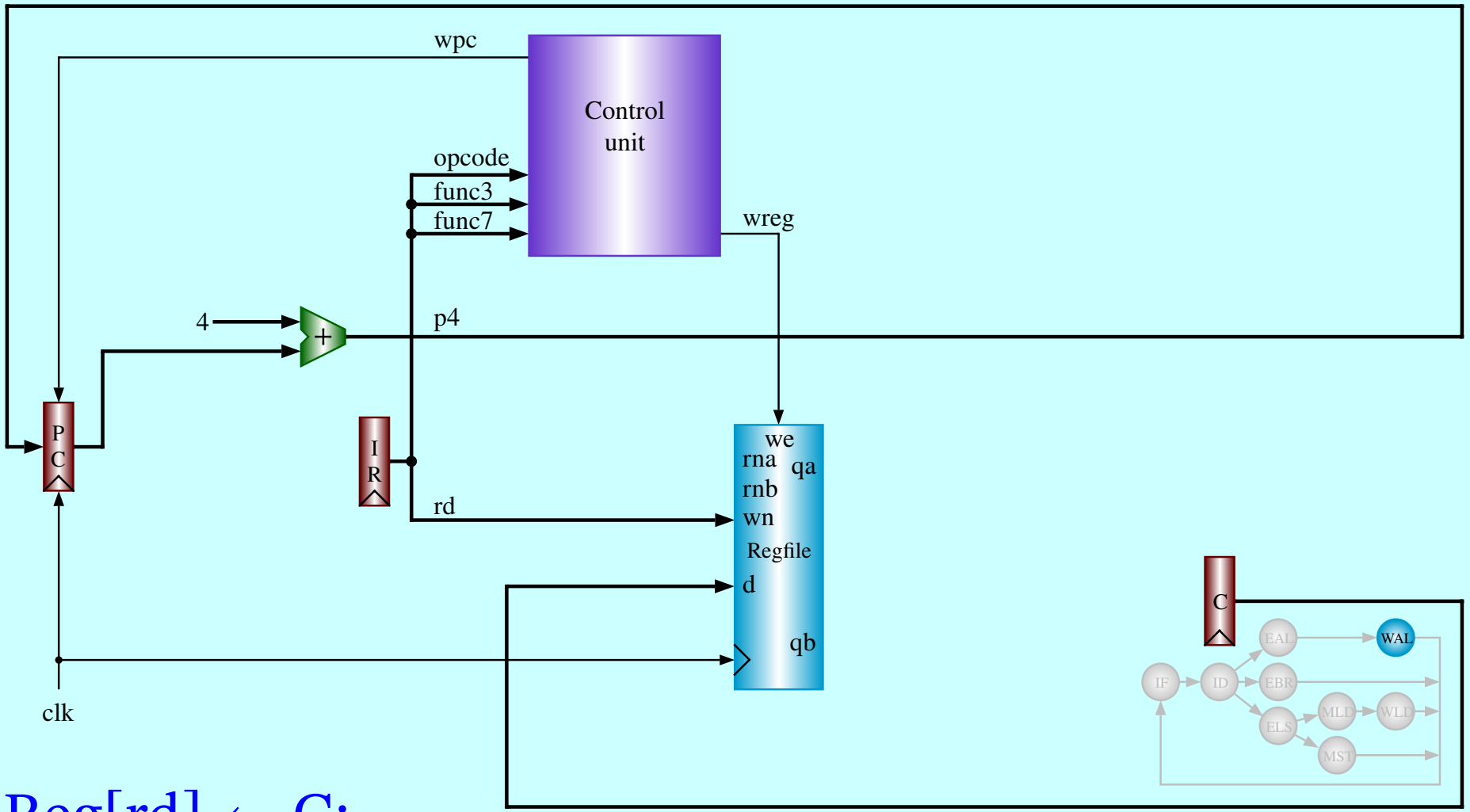
RISC-V add, ori: 3. EAL 状態



C ← A op B; または

C ← A op imm;

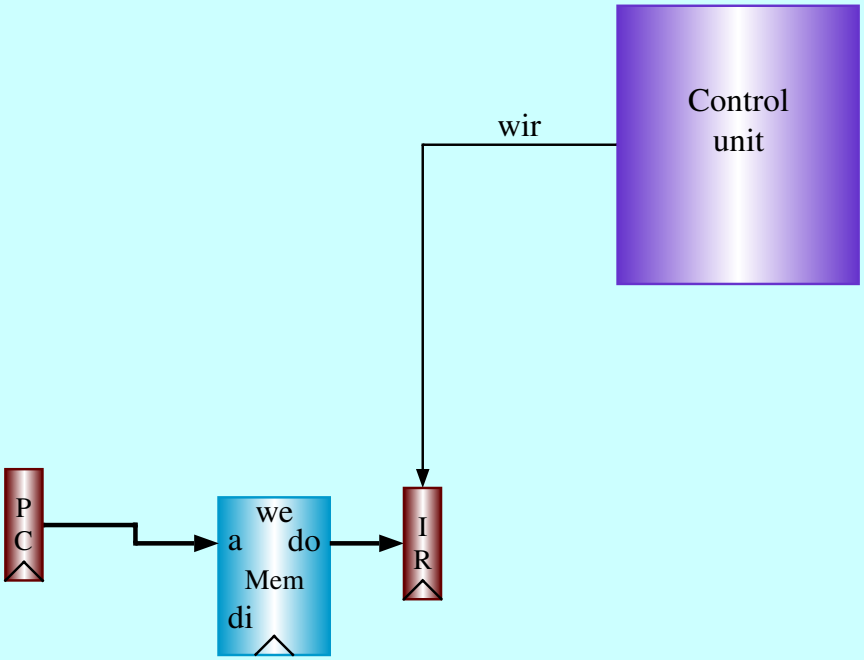
RISC-V add, ori: 4. WAL 状態

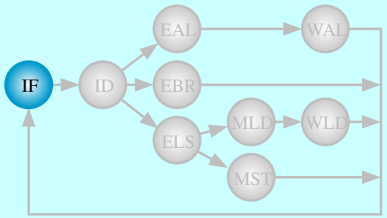


Reg[rd] ← C;

PC ← PC + 4;

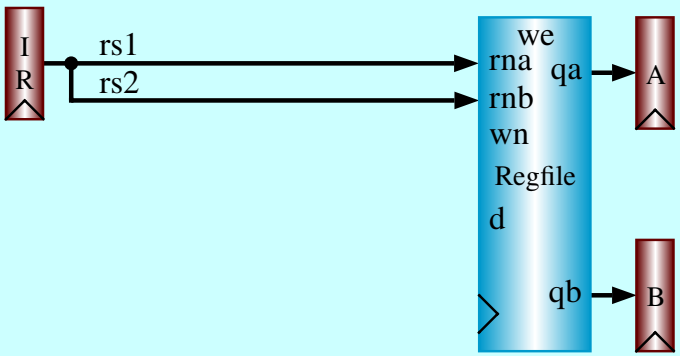
RISC-V beq, jalr, jal: 1. IF 状態

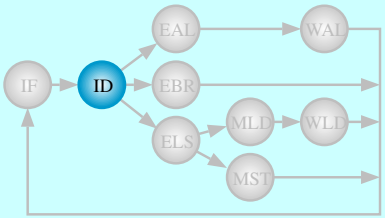




IR ← mem[PC];

RISC-V beq, jalr, jal: 2. ID 状態

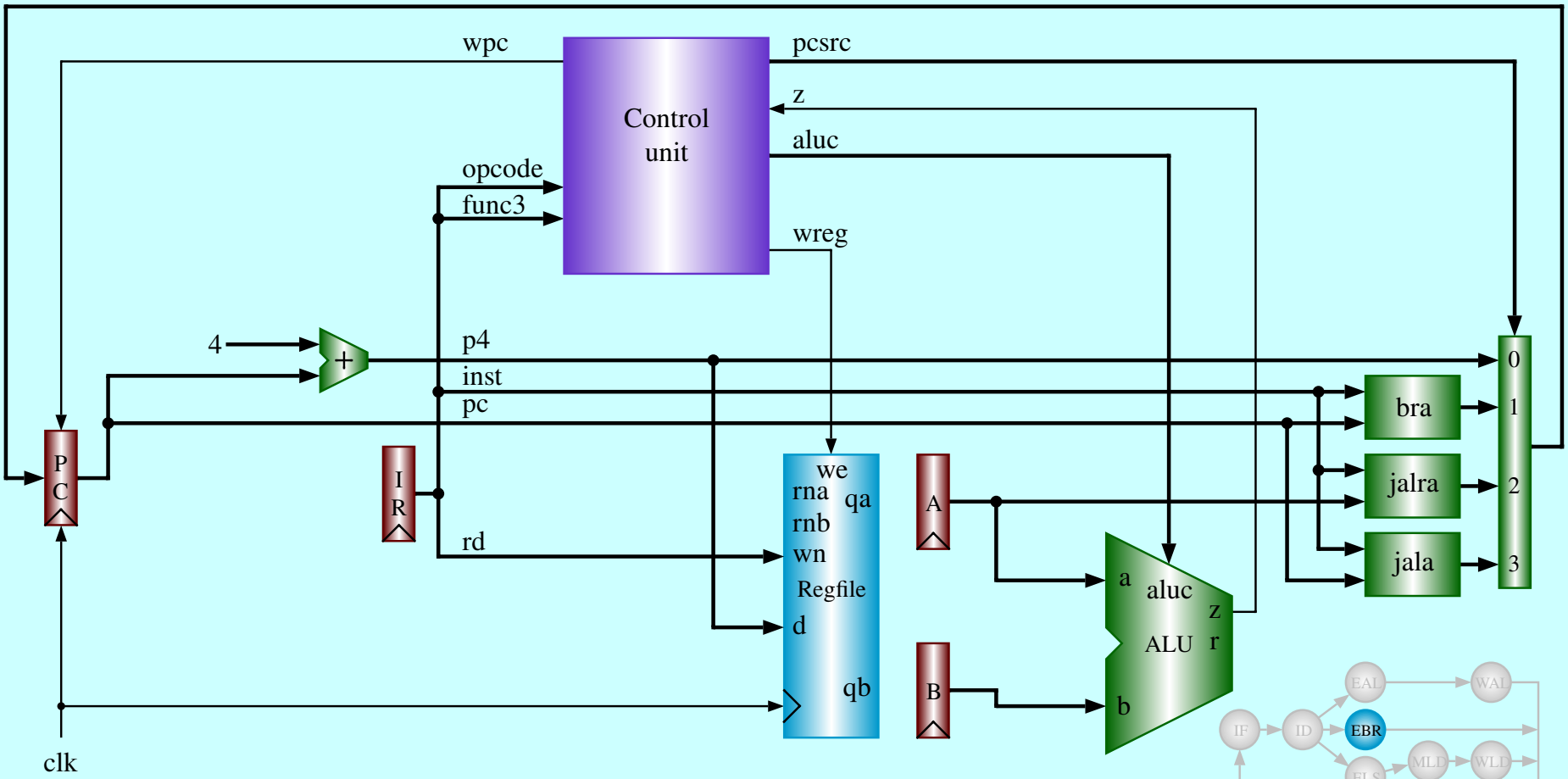




A ← Reg[rs1];

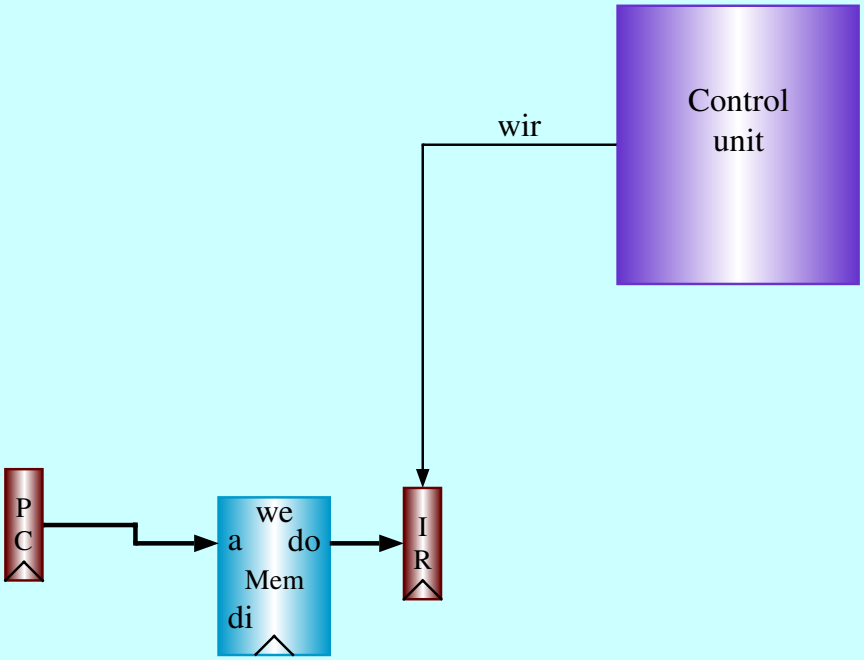
B ← Reg[rs2];

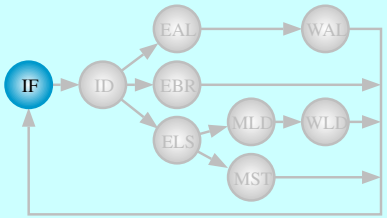
RISC-V beq, jalr, jal: 3. EBR 状態



jal: Reg[rd] ← PC + 4; PC ← jala
jalr: Reg[rd] ← PC + 4; PC ← jalra
beq: if (A==B) PC←bra; else PC←PC+4
bne: if (A!=B) PC←bra; else PC←PC+4

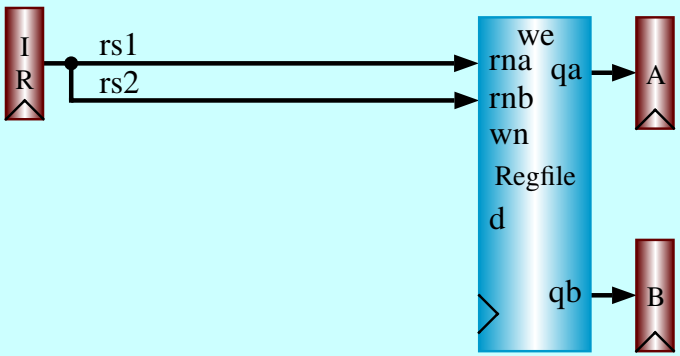
RISC-V lw: 1. IF 状態

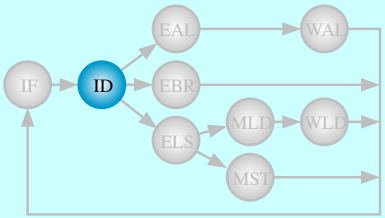




IR ← mem[PC];

RISC-V lw: 2. ID 状態

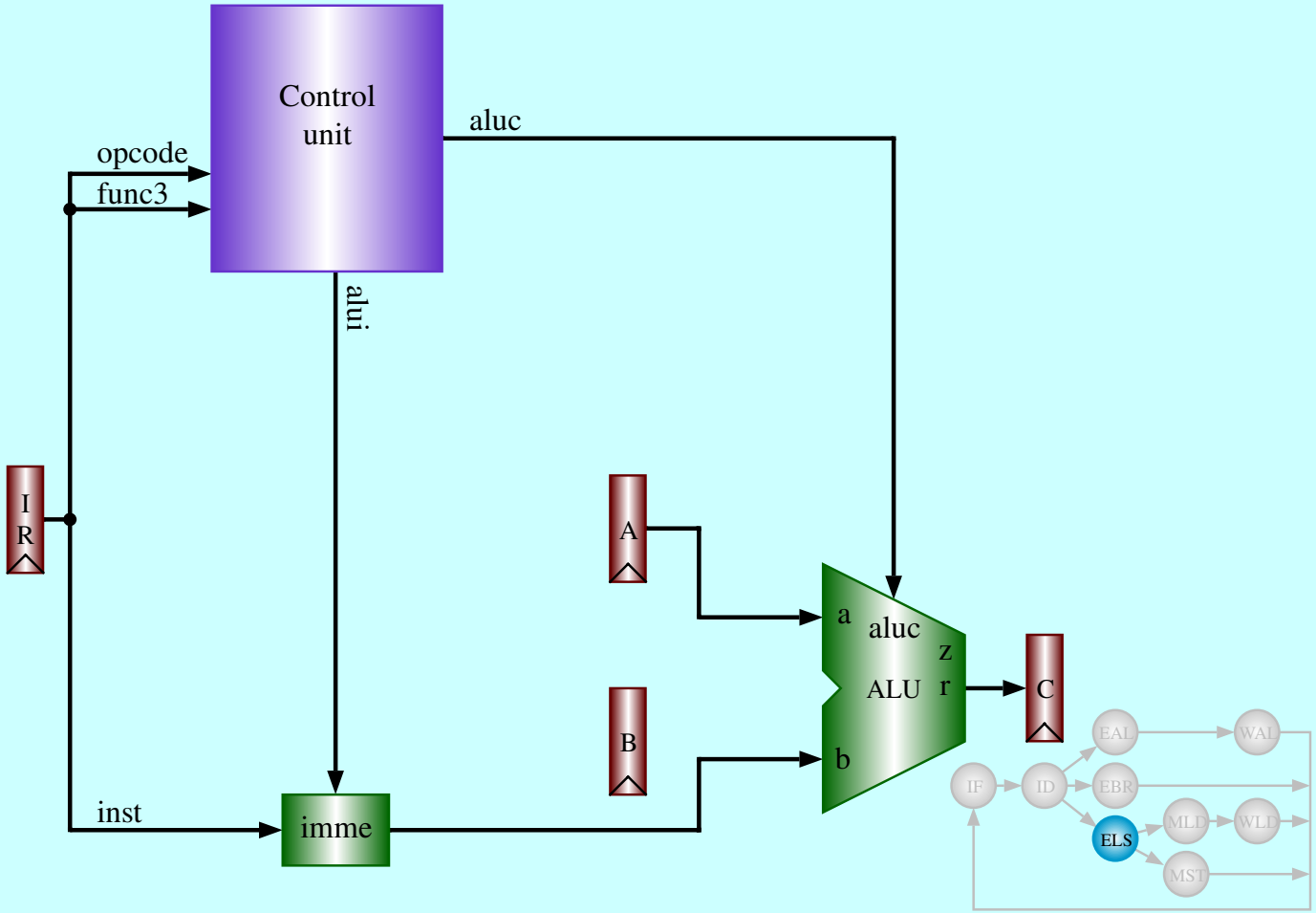




A ← Reg[rs1];

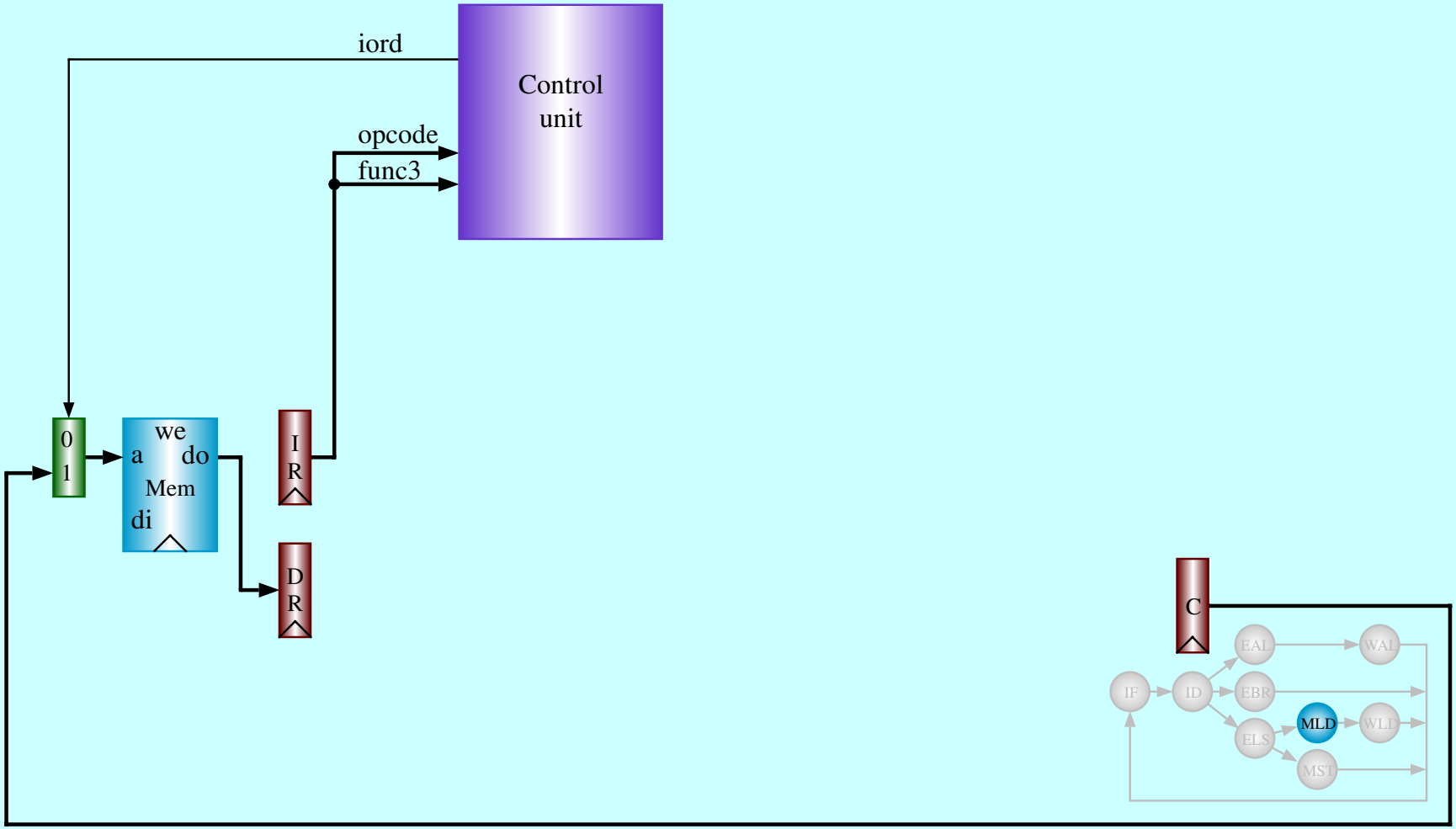
B ← Reg[rs2];

RISC-V lw: 3. ELS 状態



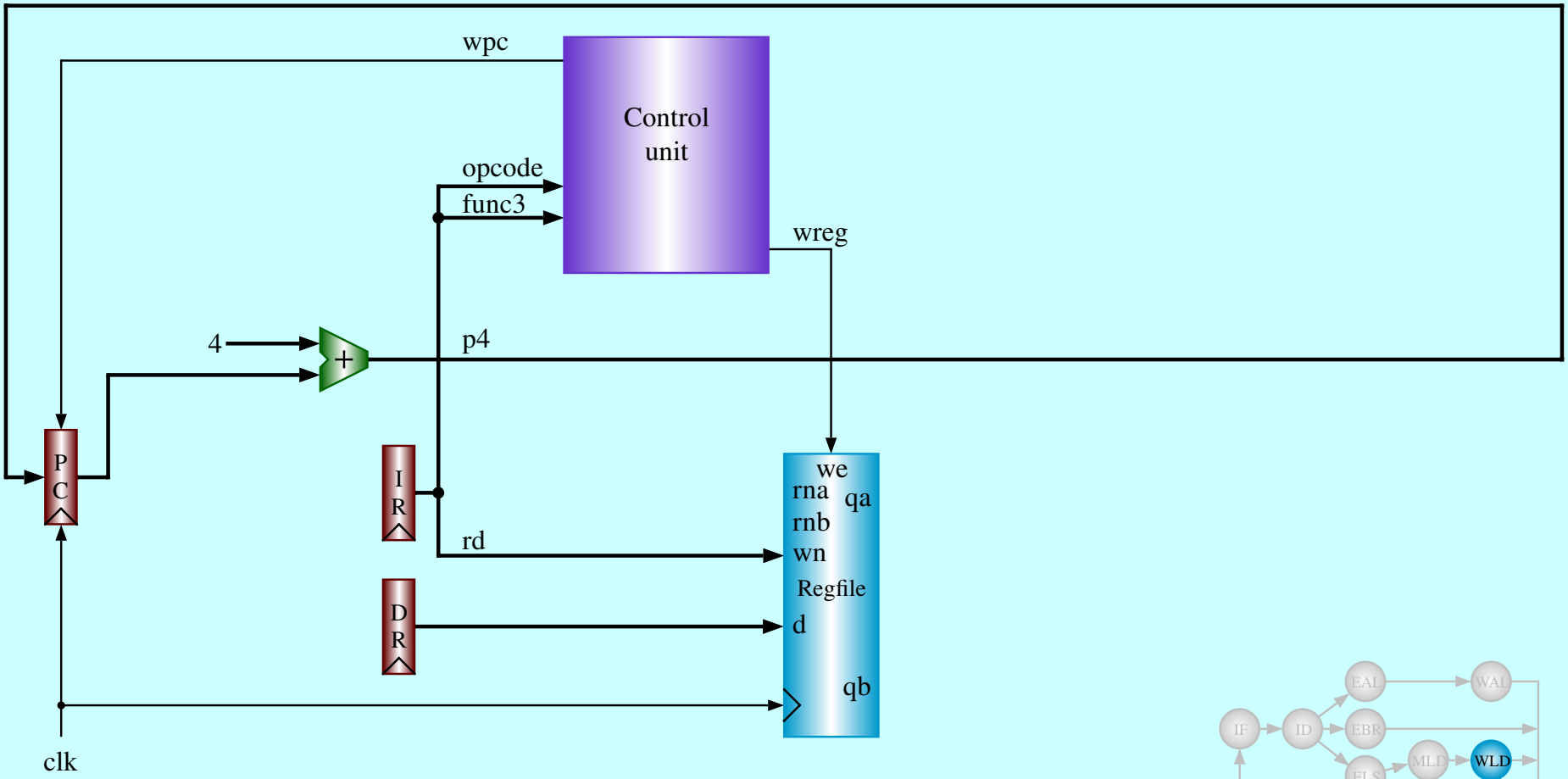
C ← A + imm;

RISC-V lw: 4. MLD 状態



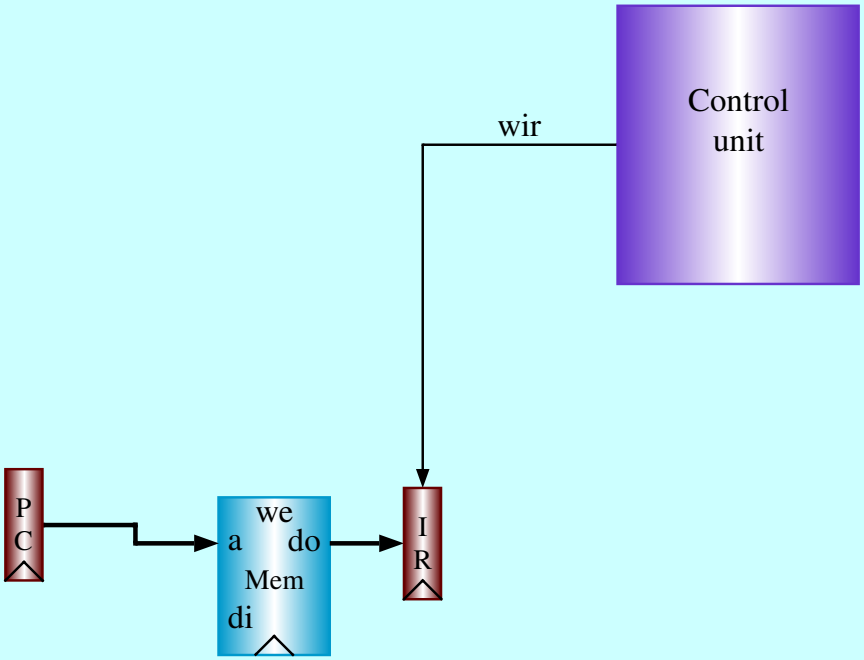
DR ← mem[C];

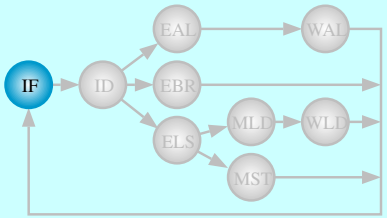
RISC-V lw: 5. WLD 状態



Reg[rd] ← DR;
PC ← PC + 4;

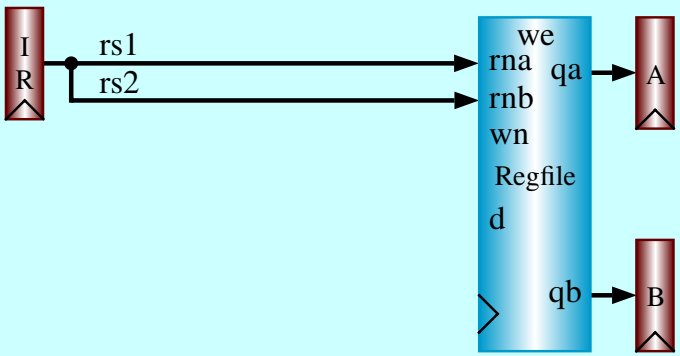
RISC-V sw: 1. IF 状態

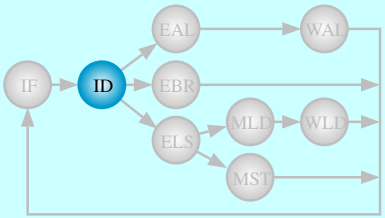




IR ← mem[PC];

RISC-V sw: 2. ID 状態

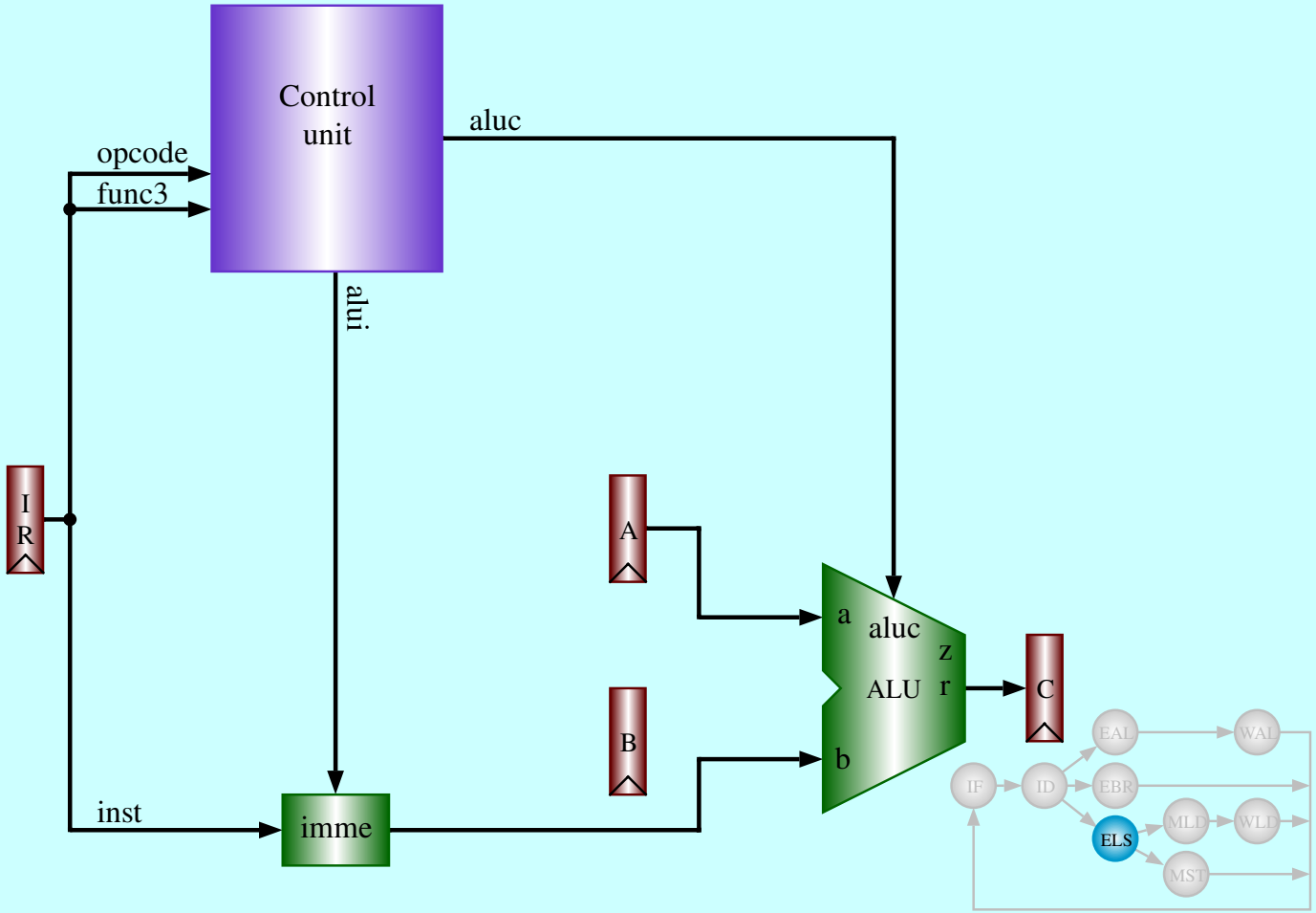




A ← Reg[rs1];

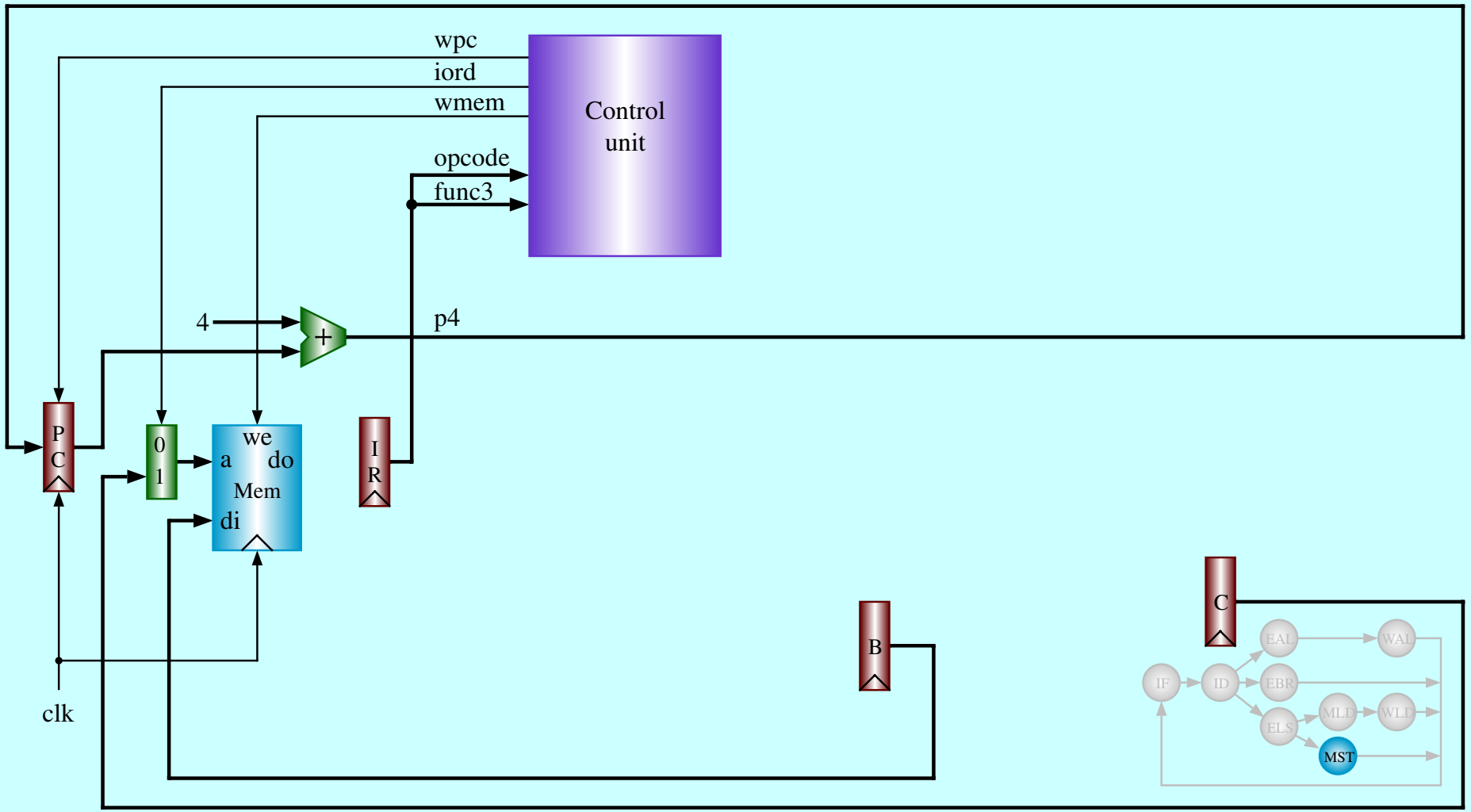
B ← Reg[rs2];

RISC-V sw: 3. ELS 状態



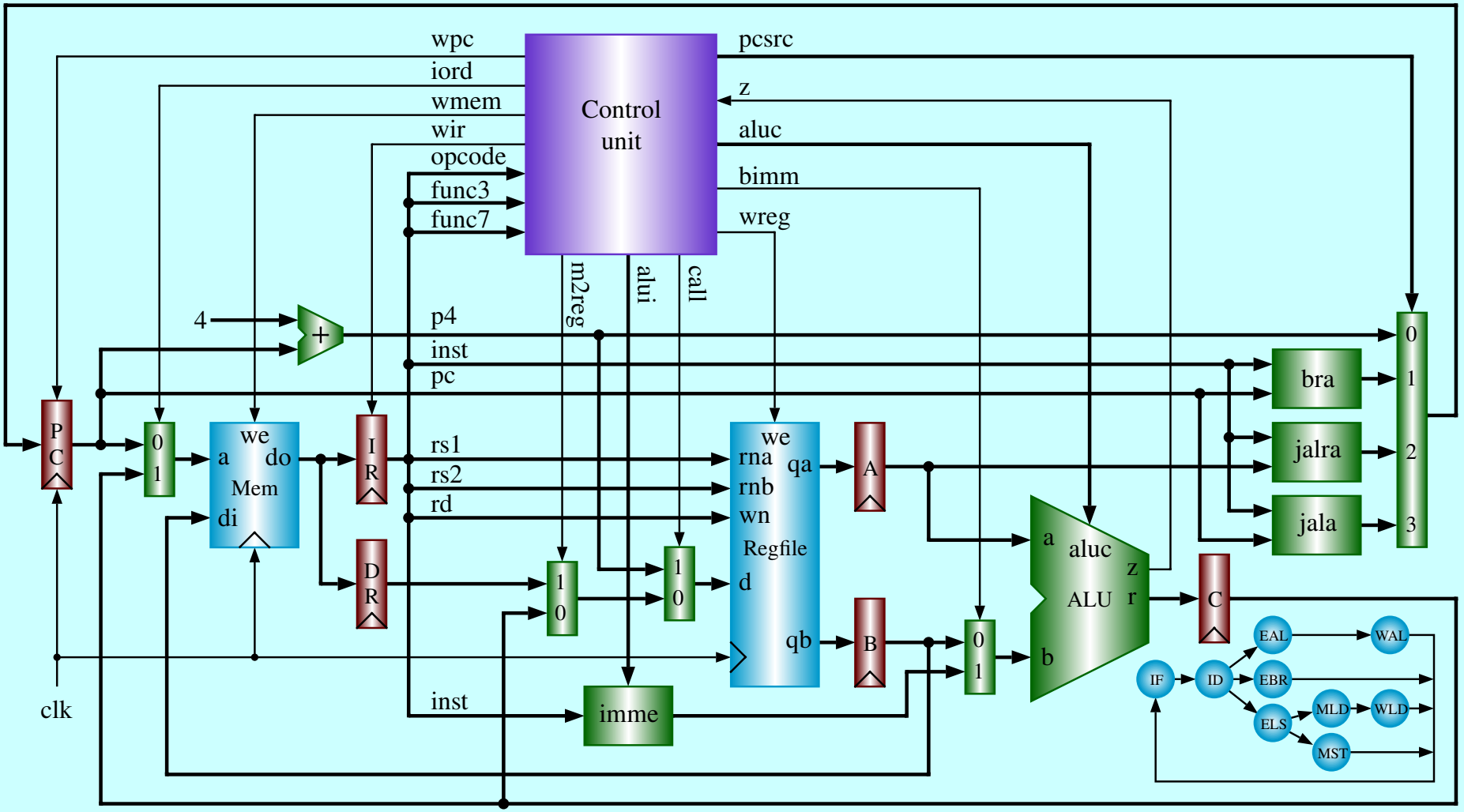
C ← A + imm;

RISC-V sw: 4. MST 状態



mem[C] ← B; PC ← PC + 4;

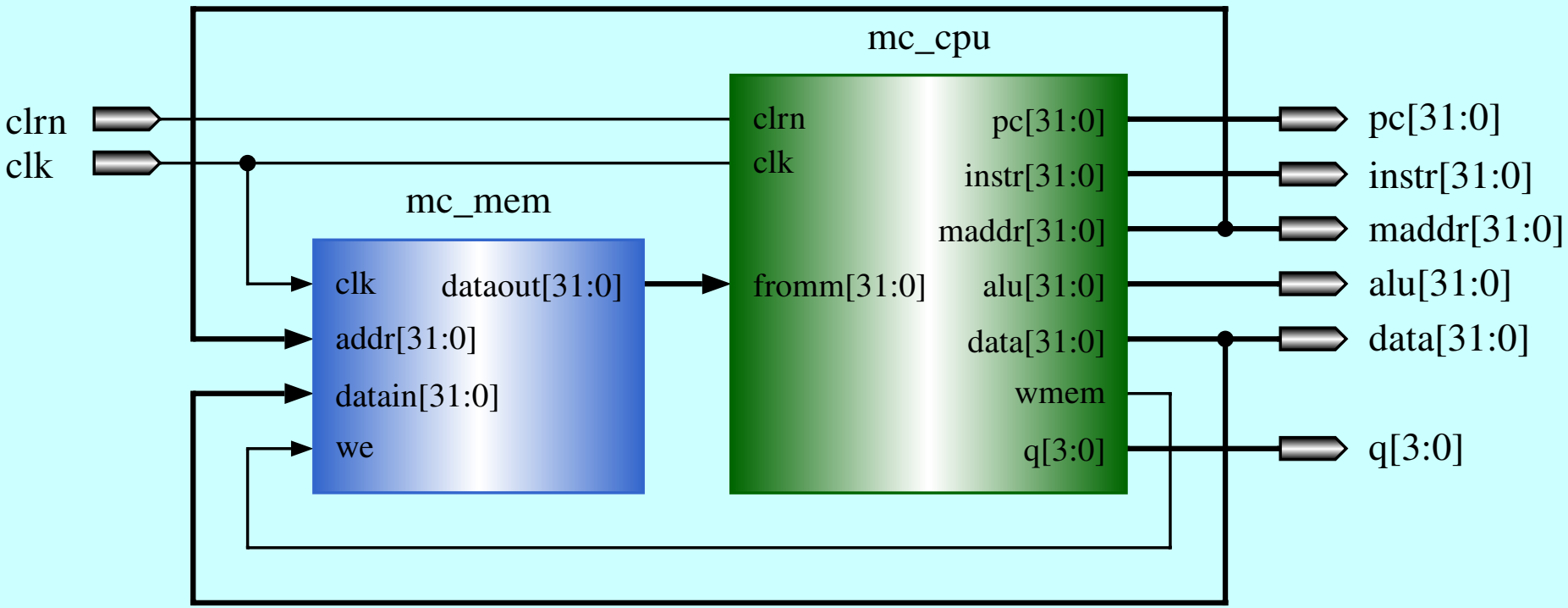
RISC-V マルチサイクルコンピュータ



CPU + メモリ Mem

RISC-V CPU とメモリの回路

マルチサイクル RISC-V CPU + メモリ



課題 VII (100 点)

前回資料を参照し、乗算プログラムを、自分の単一サイクル CPU に実行させなさい。

sc_computer_mul の回路：