

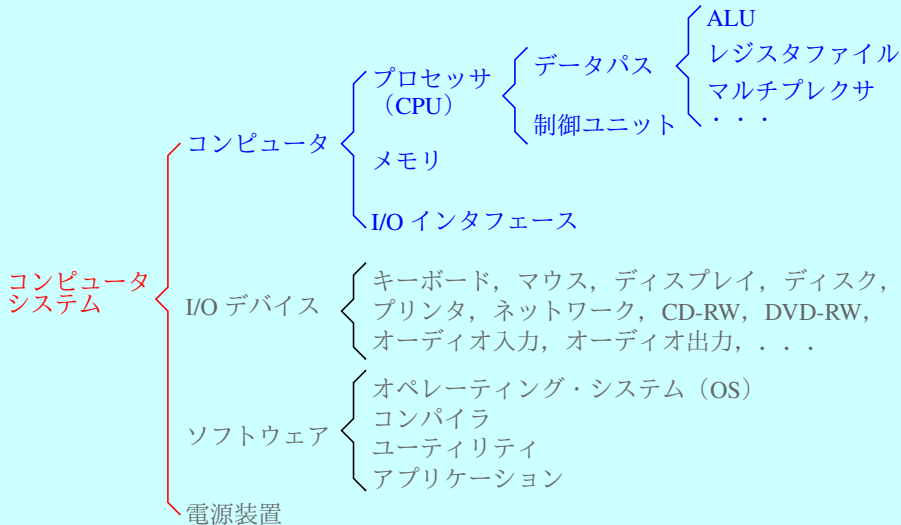
コンピュータ構成と設計（８）

マルチサイクル CPU 設計

李 亜民

2024 年 11 月 21 日 (木)

コンピュータとコンピュータシステム



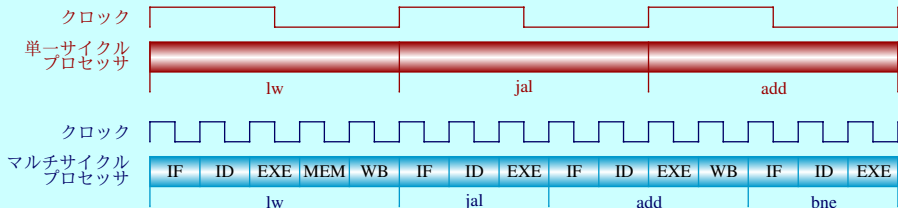
20 RISC-V 命令のまとめ

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

31	25	24	20	19	15	14	12	11	7	6	0	
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

マルチサイクル RISC-V CPU



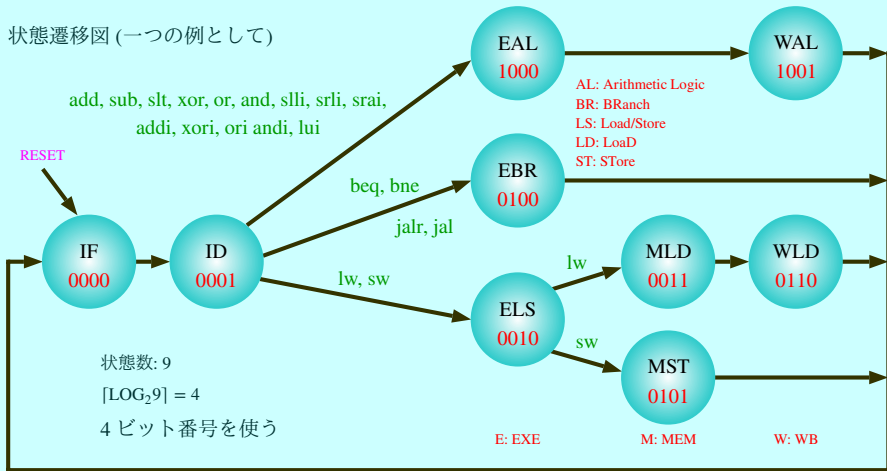
マルチサイクル方式は、

- 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：状態遷移図

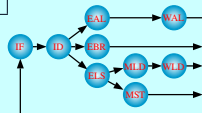
状態遷移図 (一つの例として)



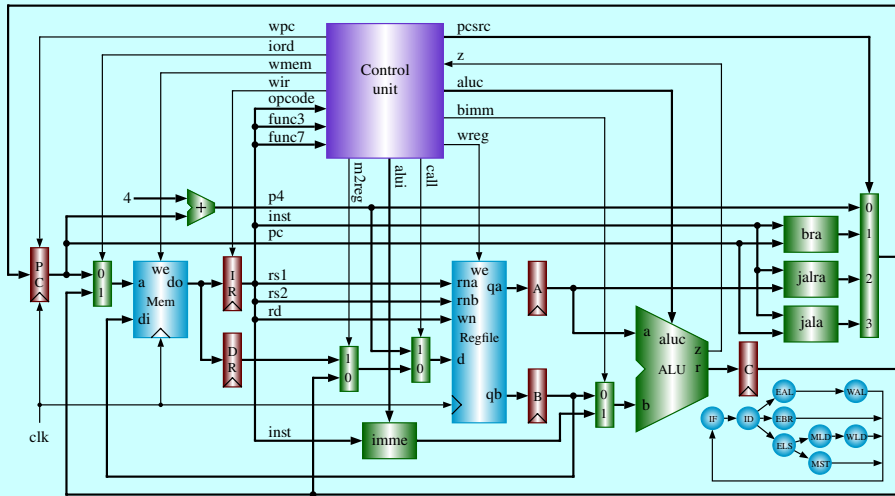
jalr, jal, beq, bne: 3 サイクル、lw: 5 サイクル、それ以外: 4 サイクル

各状態に各命令の操作

命令	IF	ID	EAL	WAL
add			$C \leftarrow A + B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
sub			$C \leftarrow A - B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
slt			$C \leftarrow A < B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
xor			$C \leftarrow A \wedge B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
or			$C \leftarrow A \mid B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
and			$C \leftarrow A \& B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
slli			$C \leftarrow B \ll \text{sa}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
srli			$C \leftarrow B \gg \text{sa}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
srai			$C \leftarrow B \ggg \text{sa}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
addi		$A \leftarrow \text{Reg}[\text{rs1}]$	$C \leftarrow A + \text{imm}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
xori	$\text{IR} \leftarrow \text{mem}[\text{PC}]$	$B \leftarrow \text{Reg}[\text{rs2}]$	$C \leftarrow A \wedge \text{imm}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
ori			$C \leftarrow A \mid \text{imm}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
andi			$C \leftarrow A \& \text{imm}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
lui			$C \leftarrow \text{imm} \ll 12$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
命令			EBR	
beq			if (A==B) $\text{PC} \leftarrow \text{bra}$; else $\text{PC} \leftarrow \text{PC} + 4$	
bne			if (A!=B) $\text{PC} \leftarrow \text{bra}$; else $\text{PC} \leftarrow \text{PC} + 4$	
jalr			$\text{Reg}[\text{rd}] \leftarrow \text{PC} + 4; \text{PC} \leftarrow \text{jalra}$	
jal			$\text{Reg}[\text{rd}] \leftarrow \text{PC} + 4; \text{PC} \leftarrow \text{jala}$	
命令			ELS	MLD
lw				$\text{DR} \leftarrow \text{mem}[\text{C}]$
命令			$C \leftarrow A + \text{imm}$	MST
sw				$\text{mem}[\text{C}] \leftarrow B; \text{PC} \leftarrow \text{PC} + 4$



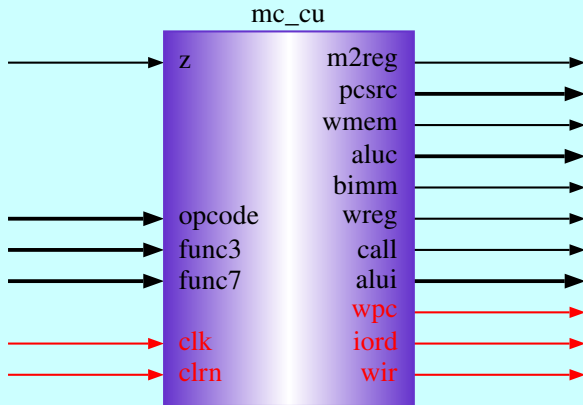
RISC-V マルチサイクルコンピュータ



CPU + メモリ Mem

制御ユニット (Control Unit) の設計

CPU 制御ユニット



赤色の信号は単一サイクルCPUの制御ユニットにはないものである。

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

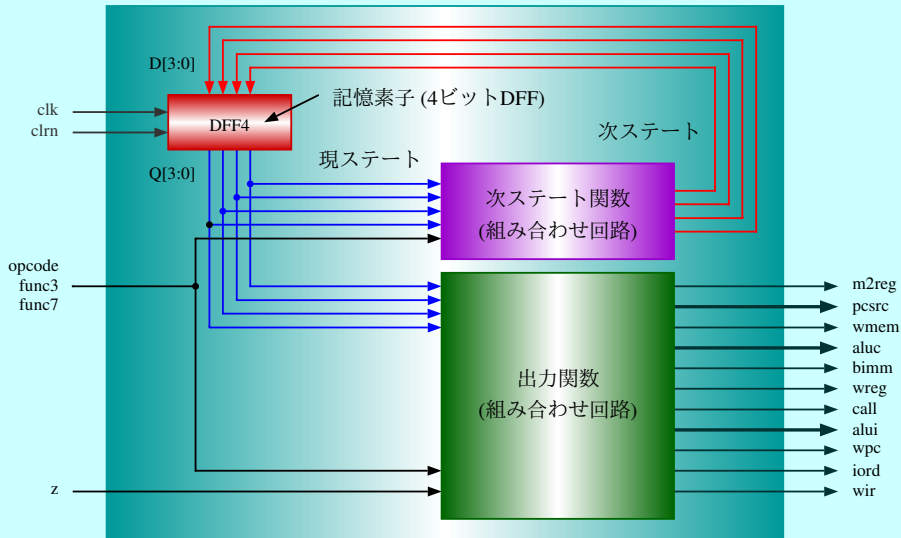
● aluc[3:0]

- ▶ 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)

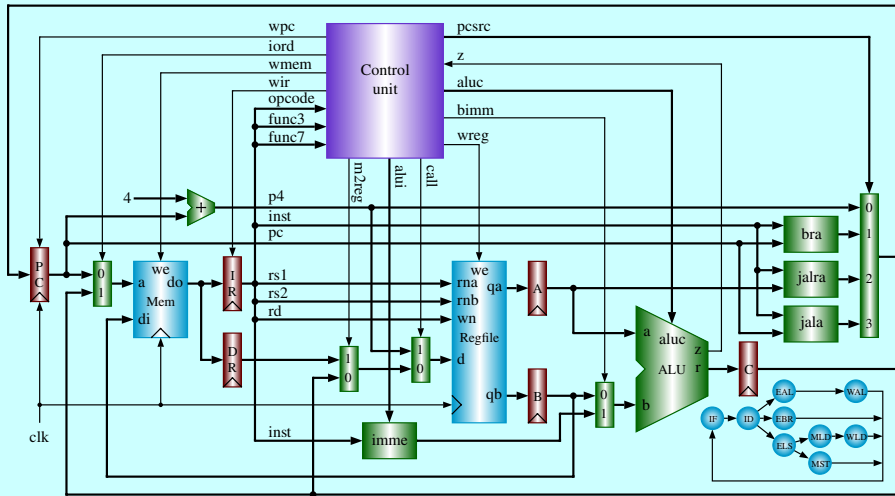
制御信号

- wpc
 - ▶ 1: PC に書き込む
 - ▶ 0: PC に書き込まない
- wir
 - ▶ 1: IR に書き込む
 - ▶ 0: IR に書き込まない
- iord
 - ▶ 1: C の値を選択
 - ▶ 0: PC の値を選択
- clk: Clock
- clrn: Clear (Active low)

制御ユニット設計（順序回路）

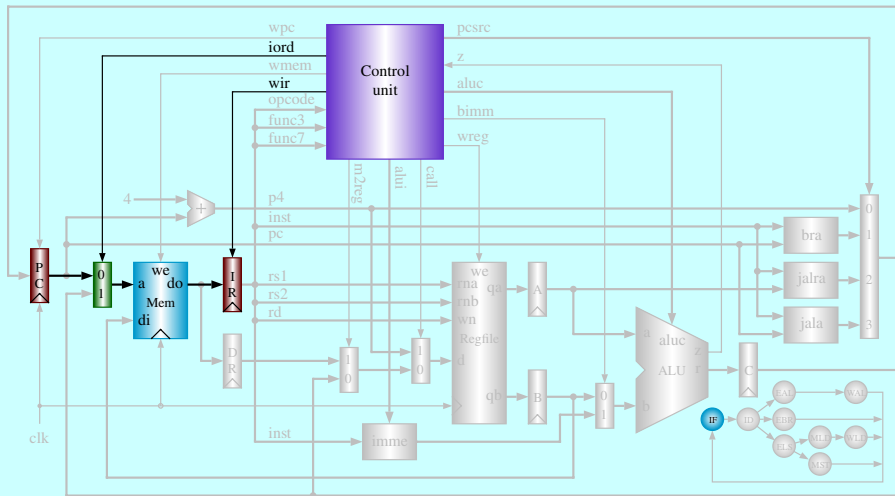


RISC-V マルチサイクルコンピュータ



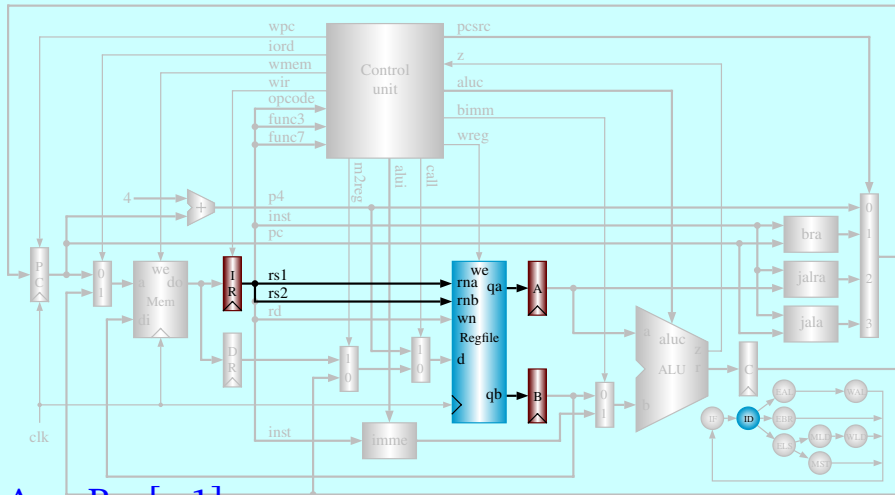
CPU + メモリ Mem

RISC-V add, ori: 1. IF 状態



$IR \leftarrow \text{mem}[PC];$

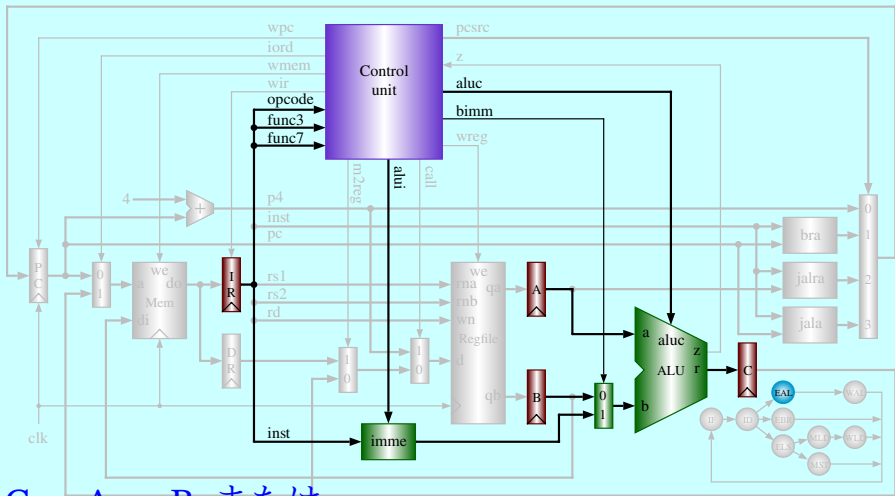
RISC-V add, ori: 2. ID 状態



$A \leftarrow \text{Reg}[\text{rs1}];$

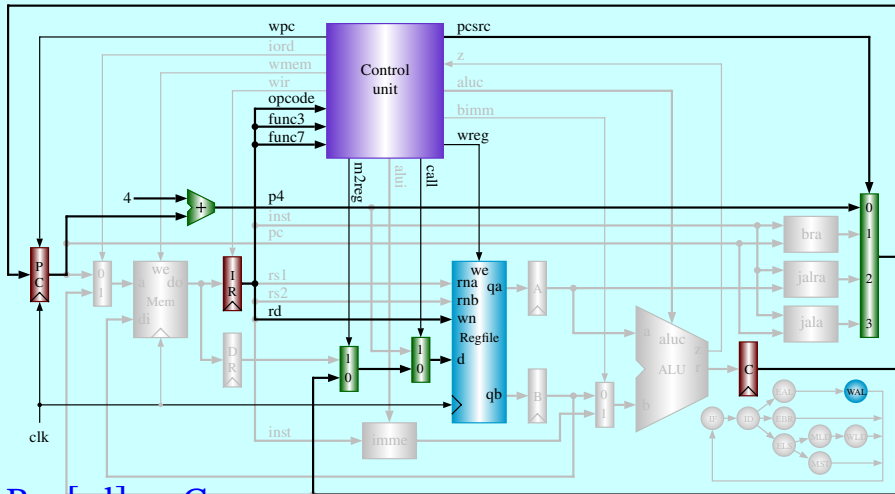
$B \leftarrow \text{Reg}[\text{rs2}];$

RISC-V add, ori: 3. EAL 状態



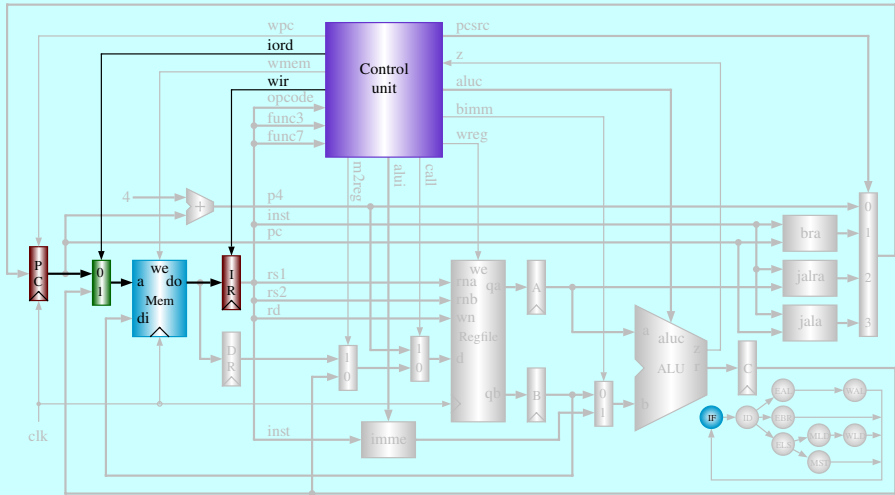
C ← A op B; または
C ← A op imm;

RISC-V add, ori: 4. WAL 状態



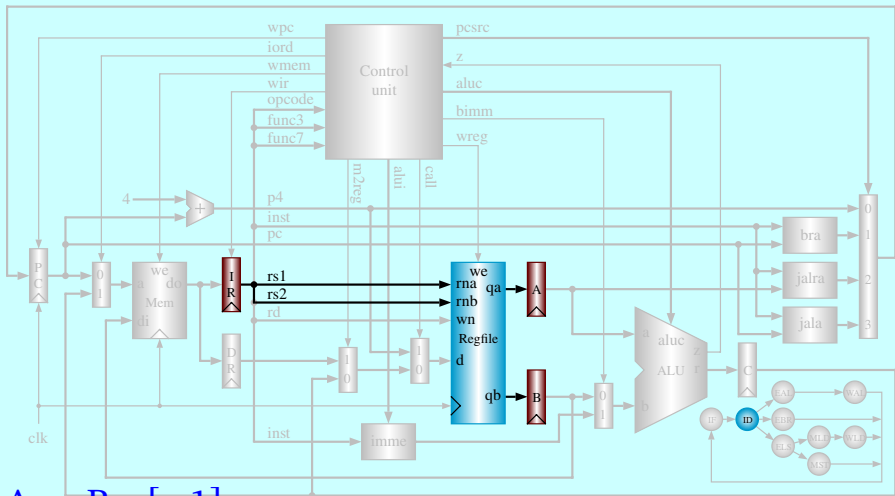
$\text{Reg}[\text{rd}] \leftarrow \text{C};$
 $\text{PC} \leftarrow \text{PC} + 4;$

RISC-V beq, jalr, jal: 1. IF 状態



```
IR ← mem[PC];
```

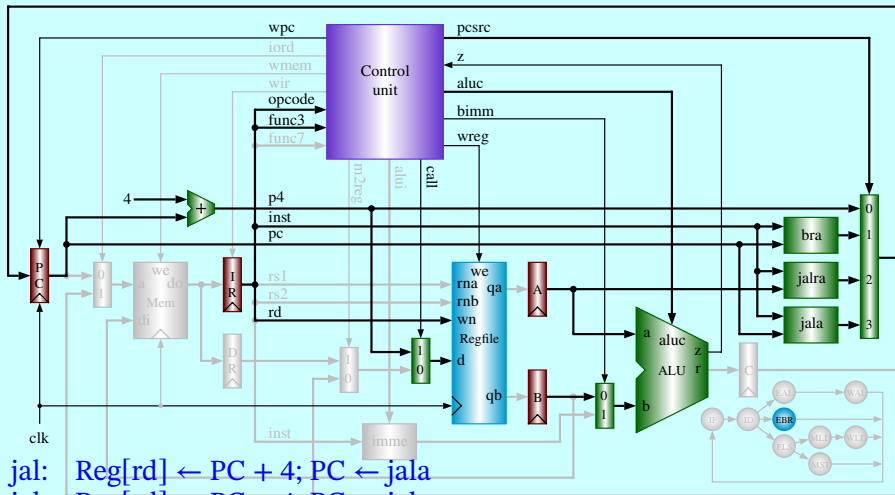
RISC-V beq, jalr, jal: 2. ID 状態



$A \leftarrow \text{Reg}[\text{rs1}];$

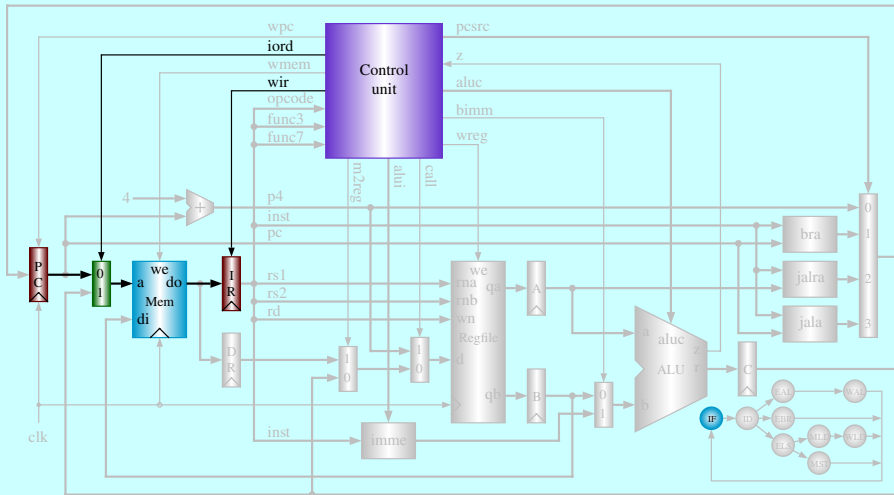
```
B ← Reg[rs2];
```

RISC-V beq, jalr, jal: 3. EBR 状態



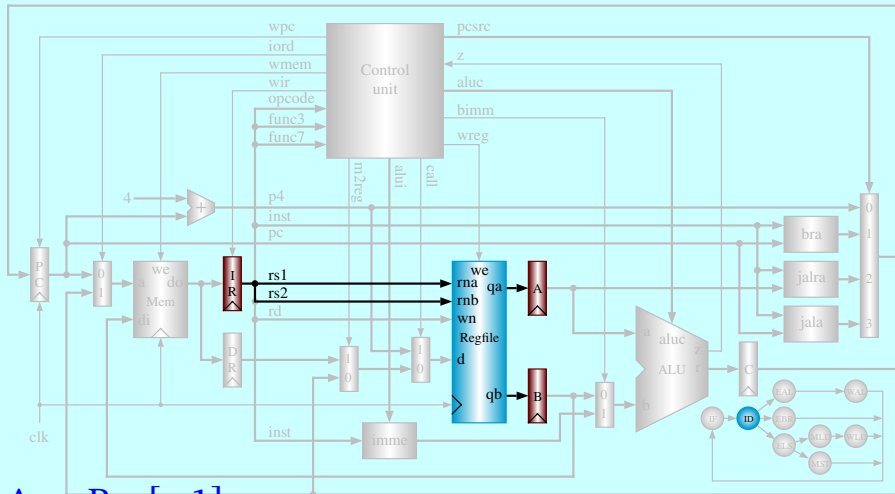
jal: $\text{Reg}[\text{rd}] \leftarrow \text{PC} + 4; \text{PC} \leftarrow \text{jala}$
 jalr: $\text{Reg}[\text{rd}] \leftarrow \text{PC} + 4; \text{PC} \leftarrow \text{jala}$
 beq: if ($A == B$) $\text{PC} \leftarrow \text{bra}$; else $\text{PC} \leftarrow \text{PC} + 4$
 bne: if ($A \neq B$) $\text{PC} \leftarrow \text{bra}$; else $\text{PC} \leftarrow \text{PC} + 4$

RISC-V lw: 1. IF 状態



$IR \leftarrow \text{mem}[PC];$

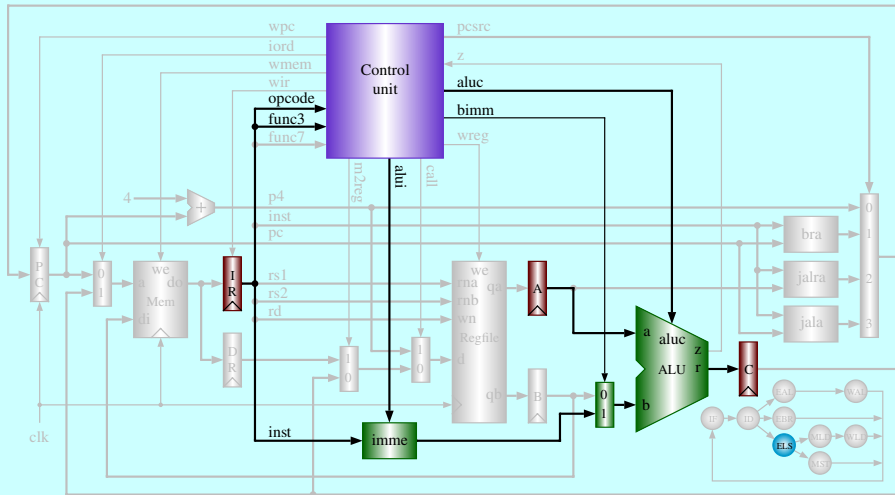
RISC-V lw: 2. ID 状態



$A \leftarrow \text{Reg}[\text{rs1}];$

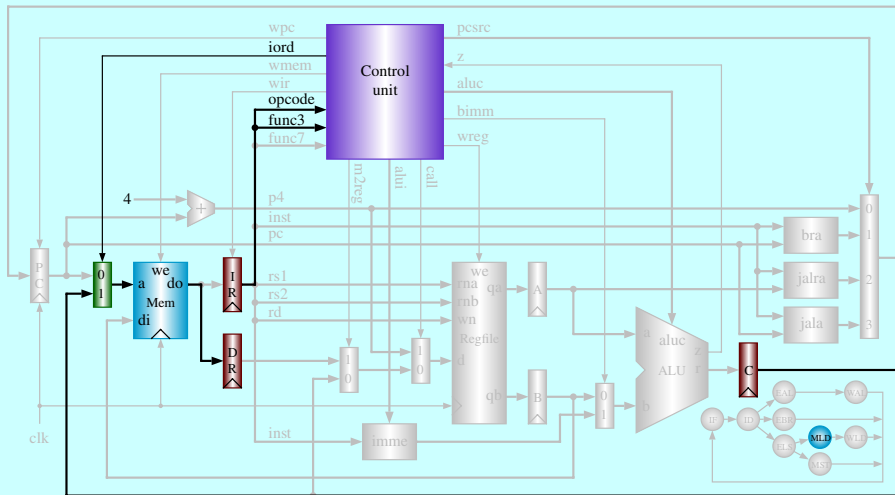
$B \leftarrow \text{Reg}[\text{rs2}];$

RISC-V lw: 3. ELS 状態



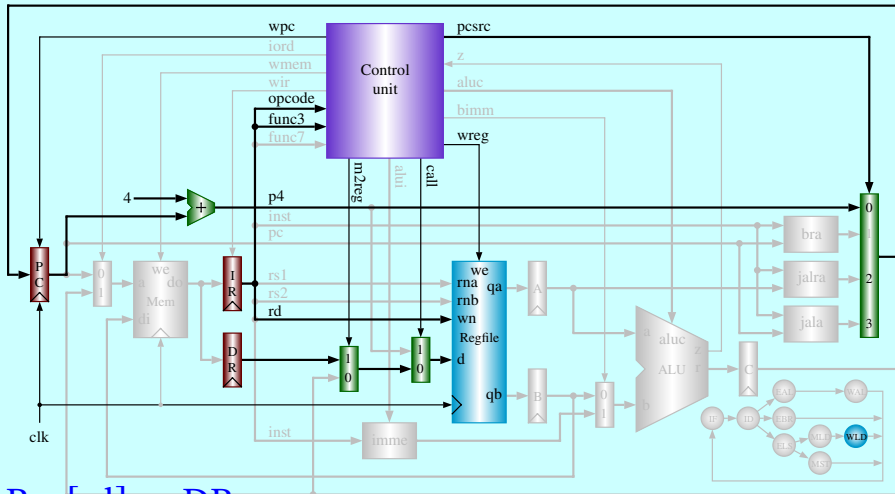
$C \leftarrow A + \text{imm};$

RISC-V lw: 4. MLD 状態



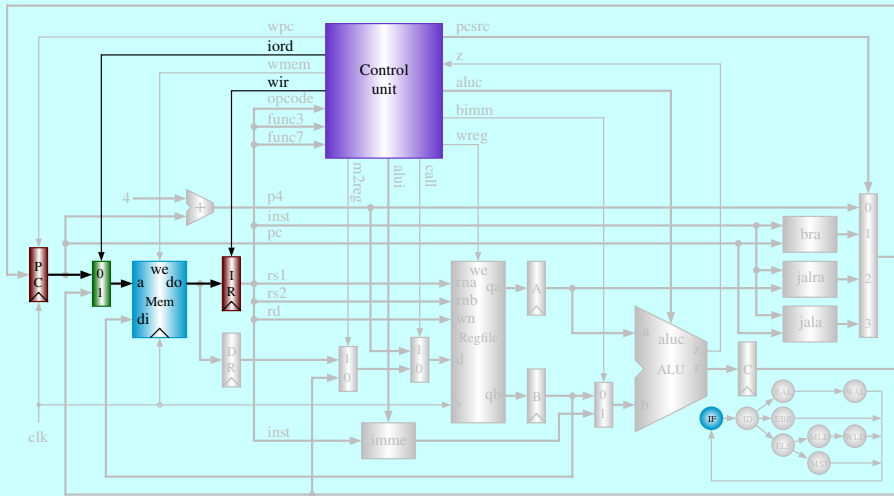
$DR \leftarrow \text{mem}[C];$

RISC-V lw: 5. WLD 状態



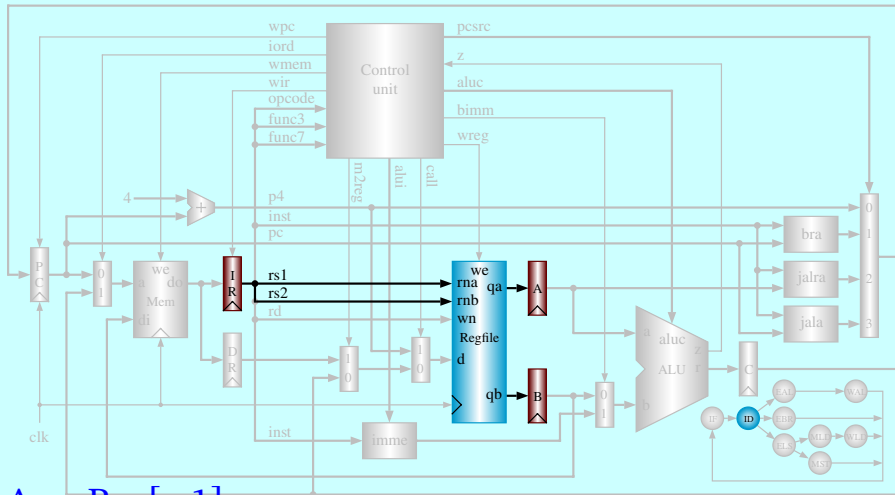
Reg[rd] ← DR;
PC ← PC + 4;

RISC-V sw: 1. IF 状態



$IR \leftarrow \text{mem}[PC];$

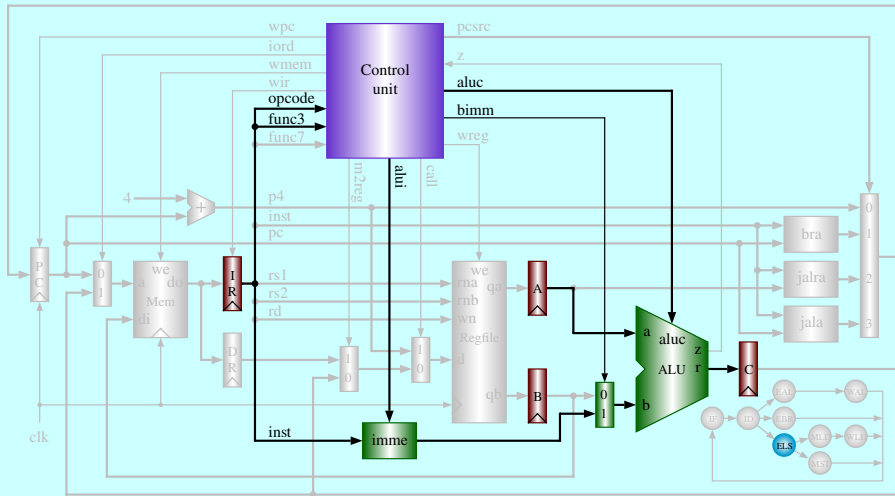
RISC-V sw: 2. ID 状態



$A \leftarrow \text{Reg}[\text{rs1}];$

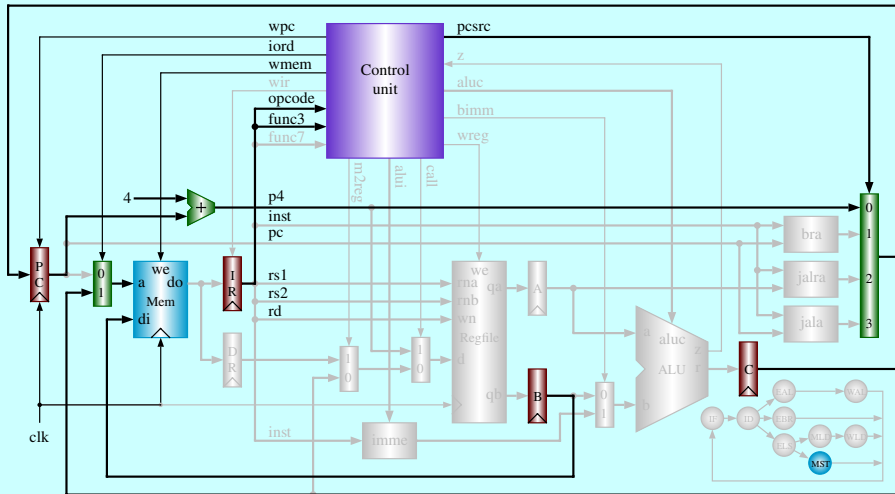
$B \leftarrow \text{Reg}[\text{rs2}];$

RISC-V sw: 3. ELS 状態



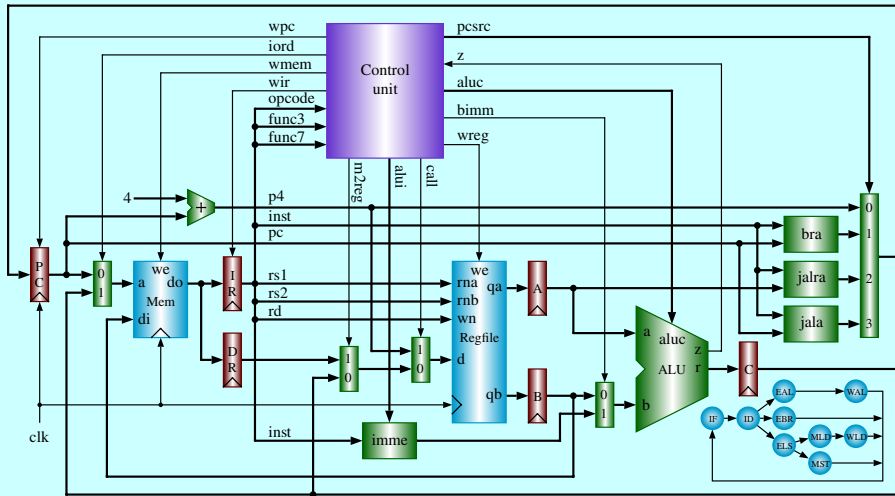
$C \leftarrow A + \text{imm};$

RISC-V sw: 4. MST 状態



$\text{mem}[C] \leftarrow B; \text{PC} \leftarrow \text{PC} + 4;$

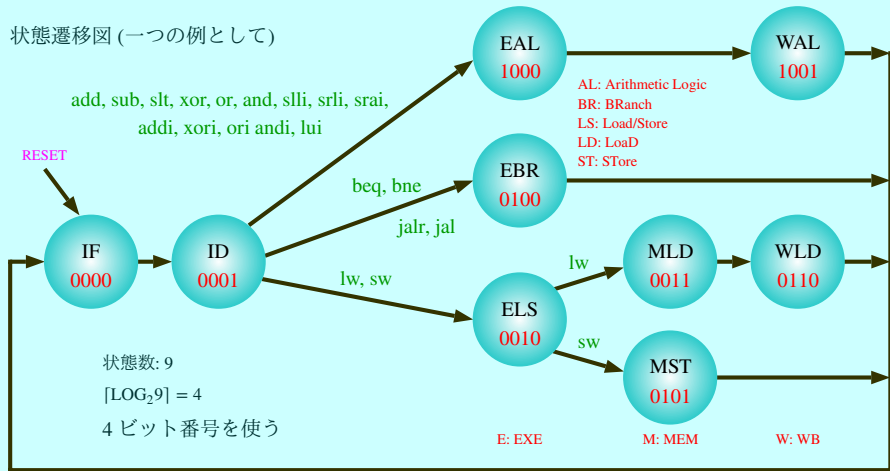
RISC-V マルチサイクルコンピュータ



CPU + メモリ Mem

マルチサイクルの実装：状態遷移図

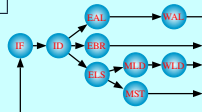
状態遷移図 (一つの例として)



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

各状態に各命令の操作

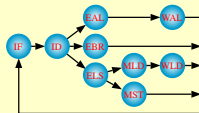
命令	IF	ID	EAL	WAL
add			$C \leftarrow A + B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
sub			$C \leftarrow A - B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
slt			$C \leftarrow A < B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
xor			$C \leftarrow A \wedge B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
or			$C \leftarrow A \mid B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
and			$C \leftarrow A \& B$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
slli			$C \leftarrow B \ll \text{sa}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
srli			$C \leftarrow B \gg \text{sa}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
srai			$C \leftarrow B \ggg \text{sa}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
addi		$A \leftarrow \text{Reg}[\text{rs1}]$	$C \leftarrow A + \text{imm}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
xori	$\text{IR} \leftarrow \text{mem}[\text{PC}]$	$B \leftarrow \text{Reg}[\text{rs2}]$	$C \leftarrow A \wedge \text{imm}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
ori			$C \leftarrow A \mid \text{imm}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
andi			$C \leftarrow A \& \text{imm}$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
lui			$C \leftarrow \text{imm} \ll 12$	$\text{Reg}[\text{rd}] \leftarrow C; \text{PC} \leftarrow \text{PC} + 4$
命令			EBR	
beq			if (A==B) $\text{PC} \leftarrow \text{bra}$; else $\text{PC} \leftarrow \text{PC} + 4$	
bne			if (A!=B) $\text{PC} \leftarrow \text{bra}$; else $\text{PC} \leftarrow \text{PC} + 4$	
jalr			$\text{Reg}[\text{rd}] \leftarrow \text{PC} + 4; \text{PC} \leftarrow \text{jalra}$	
jal			$\text{Reg}[\text{rd}] \leftarrow \text{PC} + 4; \text{PC} \leftarrow \text{jala}$	
命令			ELS	MLD
lw				$\text{DR} \leftarrow \text{mem}[\text{C}]$
命令			$C \leftarrow A + \text{imm}$	MST
sw				$\text{mem}[\text{C}] \leftarrow B; \text{PC} \leftarrow \text{PC} + 4$



制御ユニット設計

```
'timescale 1ns/1ns
module mc_cu (clk,clrn,opcode,func7,func3,z,aluc,alui,pcsrc,m2reg,bimm,call,wreg,wmem,wpc,wir,iord,state);
    input          clk, clrn;                // clock and reset
    input          [6:0] opcode;              // opcode
    input          [6:0] func7;               // func7
    input          [2:0] func3;               // func3
    input          z;                          // for beq,bne
    output reg [3:0] aluc;                    // alu operation control
    output reg [1:0] alui;                    // 00: addi,...; 01: slli,srli,srai; 10: sw; 11: lui
    output reg [1:0] pcsrc;                   // 00: pc+4; 01: beq,bne; 10: jalr; 11: jal
    output reg      m2reg;                    // instruction is an lw
    output reg      bimm;                     // alu input b is an immediate
    output reg      call;                     // instruction is a jalr or jal
    output reg      wreg;                     // write regfile
    output reg      wmem;                     // memory write enable
    output reg      wpc;                      // write pc
    output reg      wir;                      // write ir
    output reg      iord;                     // select memory address
    output reg [3:0] state;                   // state
    reg [3:0] next_state;                     // next state
    parameter [3:0] sif = 4'b0000,           // IF state
                  sid = 4'b0001,           // ID state
                  seal = 4'b1000,          // EXE Arithmetic Logic state
                  sebr = 4'b0100,          // EXE Branch state
                  sels = 4'b0010,          // EXE Load/Store state
                  smld = 4'b0011,          // MEM Load state
                  smst = 4'b0101,          // MEM Store state
                  swal = 4'b1001,          // WB Arithmetic Logic state
                  swld = 4'b0110;          // WB Load state

    // instruction decode
    wire i_lui = (opcode == 7'b0110111);
    wire i_jal = ; // add your code here
    wire i_jalr = (opcode == 7'b1100111) & (func3 == 3'b000);
    wire i_beq = ; // add your code here
    wire i_bne = ; // add your code here
```



制御ユニット設計

```
wire i_lw = ; // add your code here
wire i_sw = ; // add your code here
wire i_addi = ; // add your code here
wire i_xori = ; // add your code here
wire i_ori = ; // add your code here
wire i_andi = ; // add your code here
wire i_slli = (opcode == 7'b0010011) & (func3 == 3'b001) & (func7 == 7'b0000000);
wire i_srli = ; // add your code here
wire i_srai = ; // add your code here
wire i_add = ; // add your code here
wire i_sub = ; // add your code here
wire i_slt = ; // add your code here
wire i_xor = ; // add your code here
wire i_or = ; // add your code here
wire i_and = ; // add your code here
```

```
// control signals
```

```
always @* begin
```

```
    aluc = 4'b000;
```

```
    alui = 2'h0;
```

```
    pcsrc = 2'h0;
```

```
    m2reg = 0;
```

```
    bimm = 0;
```

```
    call = 0;
```

```
    wreg = 0;
```

```
    wmem = 0;
```

```
    wpc = 0;
```

```
    wir = 0;
```

```
    iord = 0;
```

```
    next_state = sif;
```

```
    case (state)
```

```
        sif: begin
```

```
            wir = 1;
```

```
            next_state = sid;
```

```
        end
```

```
// default outputs:
```

```
// alu operation: add
```

```
// select imm[11:0]
```

```
// select pc+4;
```

```
// select reg c
```

```
// select a;
```

```
// neither jalr nor jal
```

```
// do not write regfile
```

```
// do not write memory
```

```
// do not write pc
```

```
// do not write ir
```

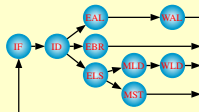
```
// select pc as address
```

```
// IF state ----- IF:
```

```
// write IR
```

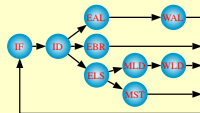
```
// next state: ID
```

```
//
```



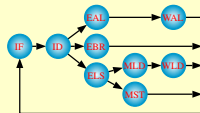
制御ユニット設計

```
sid: begin                                // ID state ----- ID:
  case (1)
    i_beq | i_bne | i_jalr | i_jal: begin // branch or jump instructions
      next_state = sebr;                // next state: EBR
    end
    i_lw | i_sw: begin                  // load/store instructions
      next_state = sels;                // next state: ELS
    end
    default: next_state = seal;         // next state: EAL
  endcase
end
seal: begin                               // EAL state ----- EAL:
  aluc[0] = ; // add your code here
  aluc[1] = ; // add your code here
  aluc[2] = ; // add your code here
  aluc[3] = i_xori | i_xor | i_srai;
  bimm = ; // add your code here
  alui[0] = ; // add your code here
  alui[1] = ; // add your code here
  next_state = swal;                    // next state: WAL
end
sebr: begin                               // EBR state ----- EBR:
  call = i_jalr | i_jal;                // save pc+4
  pcsrc[0] = ; // add your code here
  pcsrc[1] = ; // add your code here
  aluc = 4'bx001;                       // sub for beq, bne; alu does nothing for jalr, jal
  wpc = 1;                               // write PC
  wreg = i_jalr | i_jal;                // save pc+4
  next_state = sif;                     // next state: IF
end
sels: begin                               // ELS state ----- ELS:
  alui[1] = i_sw;                       // alui[1:0] = 10 for sw; 00 for lw
  bimm = 1;                             // select imm
  if (i_lw) next_state = smld;           // next state: MLD
  else next_state = smst;                // next state: MST
```

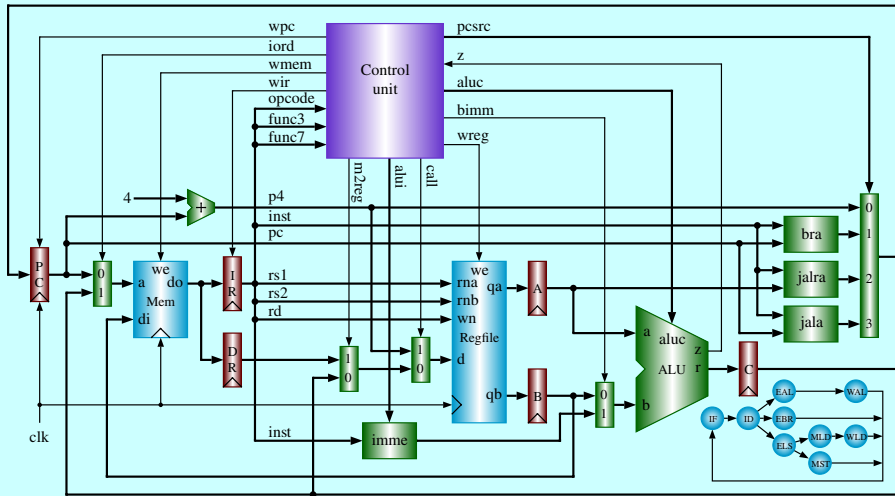


制御ユニット設計

```
end
smld: begin                                // MLD state ----- MLD:
    iord      = 1;                        // memory address = C
    next_state = swld;                    // next state: WLD
end
smst: begin                                // MST state ----- MST:
    iord      = 1;                        // memory address = C
    wmem      = 1;                        // write memory
    wpc       = 1;                        // write PC
    next_state = sif;                    // next state: IF
end
swld: begin                                // WLD state ----- WLD:
    m2reg     = 1;                        // select memory data
    wreg       = 1;                        // write register file
    wpc        = 1;                        // write PC
    next_state = sif;                    // next state: IF
end
swal: begin                                // WAL state ----- WAL:
    wreg       = 1;                        // write register file
    wpc        = 1;                        // write PC
    next_state = sif;                    // next state: IF
end //----- END
default: begin
    next_state = sif;                    // default state: IF
end
endcase
end
always @ (posedge clk or negedge clrn) begin
    if (!clrn) begin
        state <= sif;                    // reset state to IF
    end else begin
        state <= next_state;            // state transition
    end
end
endmodule
```



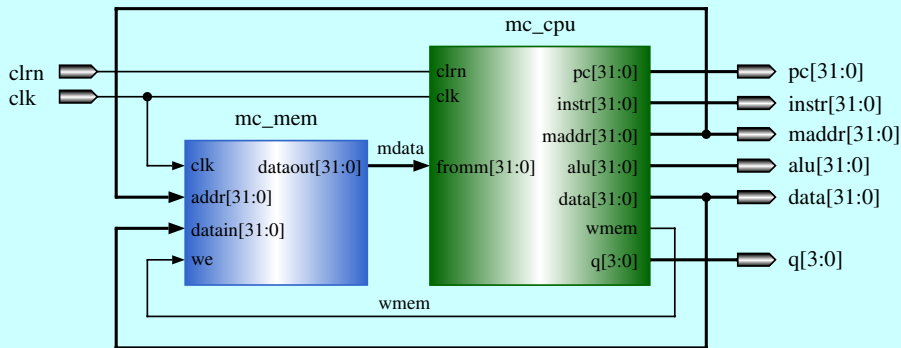
RISC-V マルチサイクルコンピュータ



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



メモリ (テスト・プログラムとデータ)

```
'timescale 1ns/1ns

module mc_mem (addr,datain,we,clk,dataout);

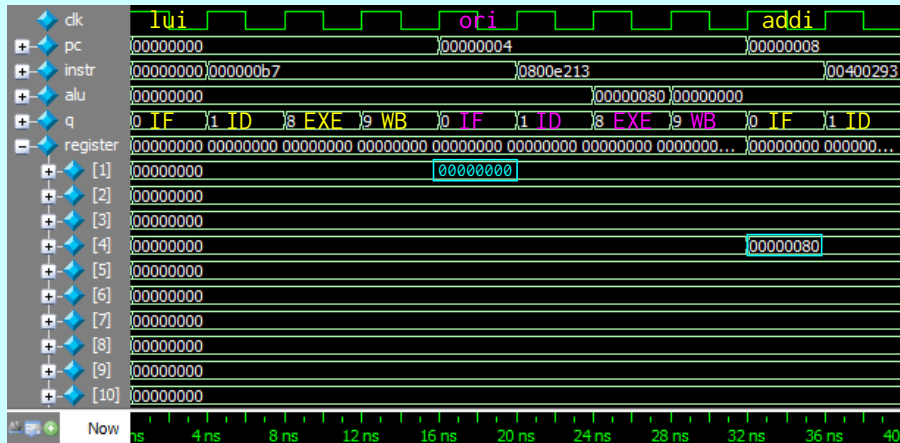
    input      clk, we;
    input [31:0] datain;
    input [31:0] addr;
    output [31:0] dataout;
    reg  [31:0] ram [0:63];
    assign dataout = ram[addr[7:2]];
    always @ (posedge clk)
        if (we) ram[addr[7:2]] = datain;
    initial begin
        ram[6'h00] = 32'b00000000000000000000000010110111; // (00) main:   lui x1, 0           # x1 <- 0
        ram[6'h01] = 32'b00001000000000001110001000010011; // (04)   ori x4, x1, 0x80  # x1 <- 0x80
        ram[6'h02] = 32'b0000000001000000000000001010010011; // (08)   addi x5, x0, 4    # x5 <- 4
        ram[6'h03] = 32'b00000101100000000000000011101111; // (0c) call:  jal x1, sum    # x1 <- 0x10 (return address), call sum
        ram[6'h04] = 32'b0000000001100010001000000100011; // (10)   sw x6, 0(x4)      # memory[x4+0] <- x6
        ram[6'h05] = 32'b000000000000000100010010010000011; // (14)   lw x9, 0(x4)      # x6 <- memory[x4+0]
        ram[6'h06] = 32'b01000000010001001000010000110011; // (18)   sub x8, x9, x4    # x8 <- x9 - x4
        ram[6'h07] = 32'b0000000000000110000000000101001011; // (1c)   addi x5, x0, 3    # x5 <- 3
        ram[6'h08] = 32'b11111111111100101000001010010011; // (20) loop2: addi x5, x5, -1  # x5 <- x5 - 1
        ram[6'h09] = 32'b11111111111100101110010000010011; // (24)   ori x8, x5, -1    # x8 <- x5 | 0xffffffff = 0xffffffff
        ram[6'h0a] = 32'b01010101010101000100010000010011; // (28)   xori x8, x8, 0x555  # x8 <- x8 ^ 0x00000555 = 0xfffffaaa
        ram[6'h0b] = 32'b11111111111100000000010010010011; // (2c)   addi x9, x0, -1    # x9 <- 0xffffffff
        ram[6'h0c] = 32'b1111111111110100111010100010011; // (30)   andi x10,x9, -1    # x10<- x9 & 0xffffffff = 0xffffffff
        ram[6'h0d] = 32'b0000000001001010101000100010011; // (34)   or x4, x10, x9    # x4 <- x10 | x9 = 0xffffffff
```

メモリ (テスト・プログラムとデータ)

```
ram[6'h0e] = 32'b00000000100101010100010000110011; // (38)      xor x8, x10, x9      # x8 <- x10 ^ x9 = 0x00000000
ram[6'h0f] = 32'b00000000010001010111001110110011; // (3c)      and x7, x10, x4      # x7 <- x10 & x4 = 0xffffffff
ram[6'h10] = 32'b0000000000000000101000010001100011; // (40)      beq x5, x0, shift    # if x5 = 0, goto shift
ram[6'h11] = 32'b1111110110110111111111000001101111; // (44)      jal x0, loop2        # jump to loop2
ram[6'h12] = 32'b11111111111100000000001010010011; // (48) shift: addi x5, x0, -1    # x5 <- 0xffffffff
ram[6'h13] = 32'b0000000011110010100101010000010011; // (4c)      slli x8, x5, 15      # x8 <- 0xffffffff << 15 = 0xffff8000
ram[6'h14] = 32'b00000001000001000001010000010011; // (50)      slli x8, x8, 16      # x8 <- 0xffff8000 << 16 = 0x80000000
ram[6'h15] = 32'b01000001000001000101010000010011; // (54)      srli x8, x8, 16      # x8 <- 0x80000000 >>> 16 = 0xffff8000
ram[6'h16] = 32'b00000000111101000101010000010011; // (58)      srli x8, x8, 15      # x8 <- 0xffff8000 >> 15 = 0x0001ffff
ram[6'h17] = 32'b00000000011000100010000110110011; // (5c)      slt x3, x4, x6        # x3 <- 0xffffffff < 0x000002ff = 1
ram[6'h18] = 32'b00000000000000000000000001101111; // (60) finish: jal x0, finish    # dead loop
ram[6'h19] = 32'b0000000000000000000000000110011011; // (64) sum: add x6, x0, x0    # x6 <- 0 (subroutine entry)
ram[6'h1a] = 32'b000000000000000100010010010000011; // (68) loop: lw x9, 0(x4)      # x9 <- memory[x4+0]
ram[6'h1b] = 32'b000000000100001000000001000010011; // (6c)      addi x4, x4, 4        # x4 <- x4 + 4 (address+4)
ram[6'h1c] = 32'b000000000100100110000001100110011; // (70)      add x6, x6, x9        # x6 <- x6 + x9 (sum)
ram[6'h1d] = 32'b11111111111100101000001010010011; // (74)      addi x5, x5, -1       # x5 <- x5 - 1 (counter--)
ram[6'h1e] = 32'b1111111000000010010100011100011; // (78)      bne x5, x0, loop      # if x5 != 0, goto loop
ram[6'h1f] = 32'b00000000000000000100000001100111; // (7c)      ret x1                # return from subroutine
ram[6'h20] = 32'h000000f2; // (80) data[0]
ram[6'h21] = 32'h0000000e; // (84) data[1]
ram[6'h22] = 32'h00000200; // (88) data[2]
ram[6'h23] = 32'hfffffff; // (8c) data[3]
ram[6'h24] = 32'h00000000; // (90) the sum stored by sw instruction

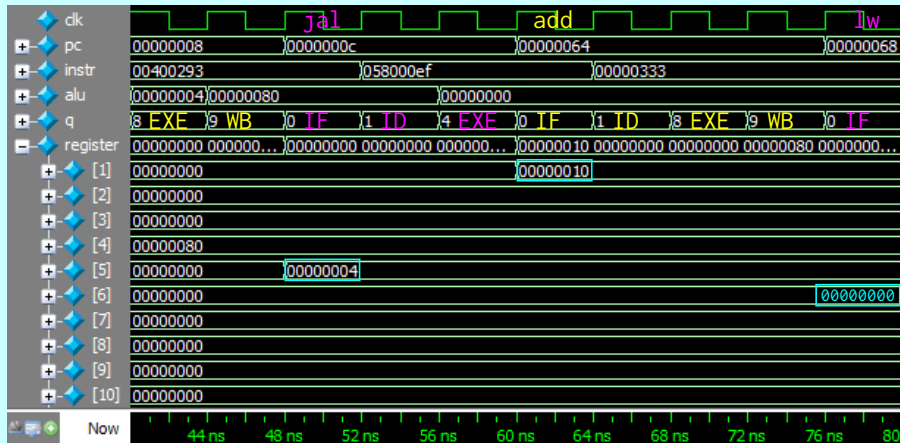
end
endmodule
```

シミュレーション1



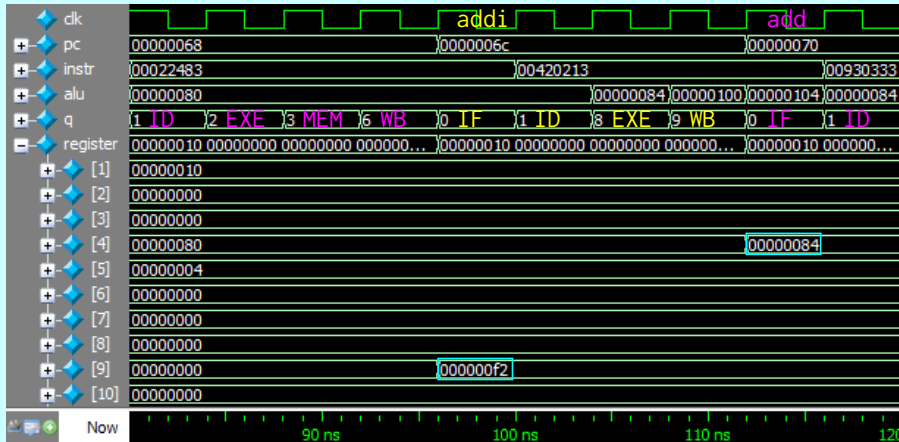
```
(00) main:    lui   x1, 0           # x1 <- 0
(04)          ori   x4, x1, 0x80    # x1 <- 0x80
(08)          addi  x5, x0, 4       # x5 <- 4
```

シミュレーション2



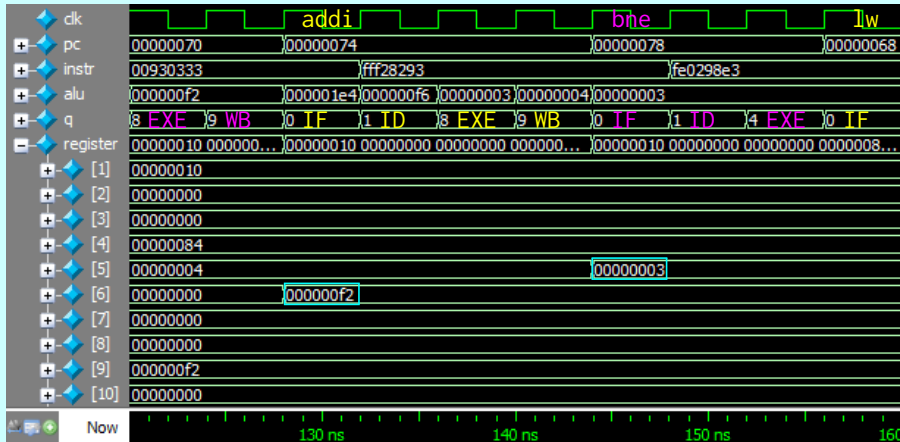
```
(0c) call:    jal    x1, sum        # x1 <- 0x10, jump to sum
(64) sum:     add    x6, x0, x0     # x6 <- 0
(68) loop:    lw     x9, 0(x4)      # x9 <- memory[x4+0] = 0xf2
```

シミュレーション3



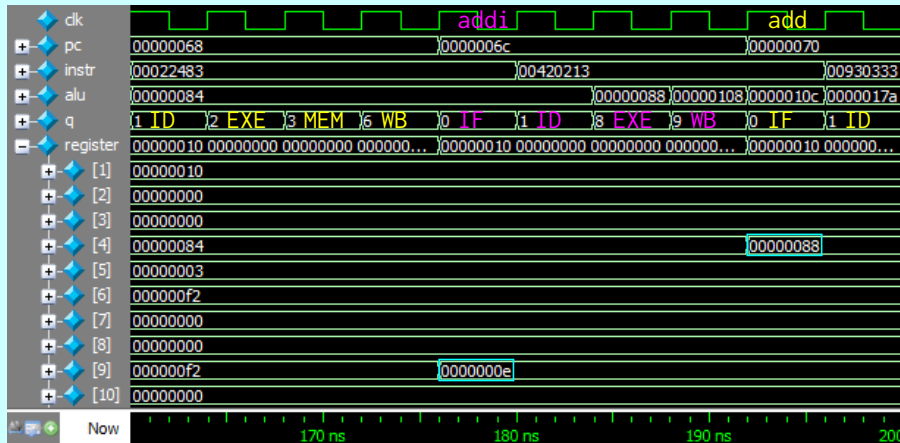
(6c) `addi x4, x4, 4` # $x4 \leftarrow x4 + 4 = 0x84$
(70) `add x6, x6, x9` # $x6 \leftarrow x6 + x9 = 0xf2$

シミュレーション4



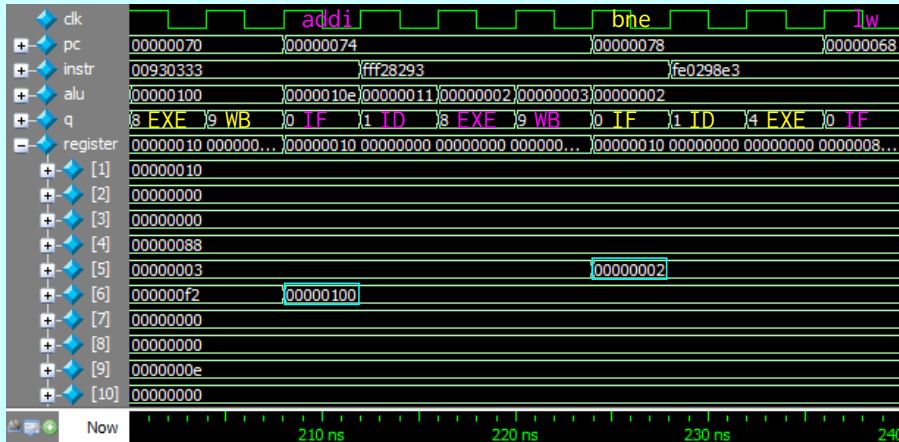
```
(74)      addi x5, x5, -1      # x5 <- x5 - 1 = 3
(78)      bne  x5, x0, loop    # if x5 != 0, goto loop
(68) loop:  lw   x9, 0(x4)     # x9 <- memory[x4+0] = 0xe
```

シミュレーション5



(6c) `addi x4, x4, 4` # $x4 \leftarrow x4 + 4 = 0x88$
(70) `add x6, x6, x9` # $x6 \leftarrow x6 + x9 = 0x100$

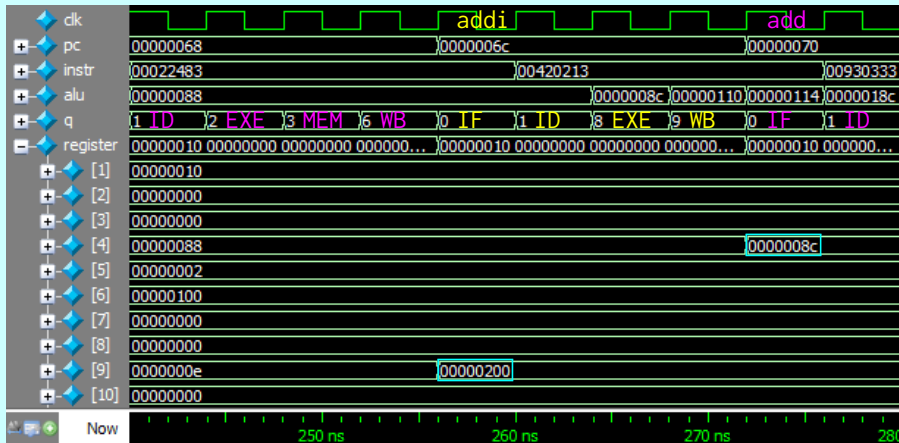
シミュレーション6



```

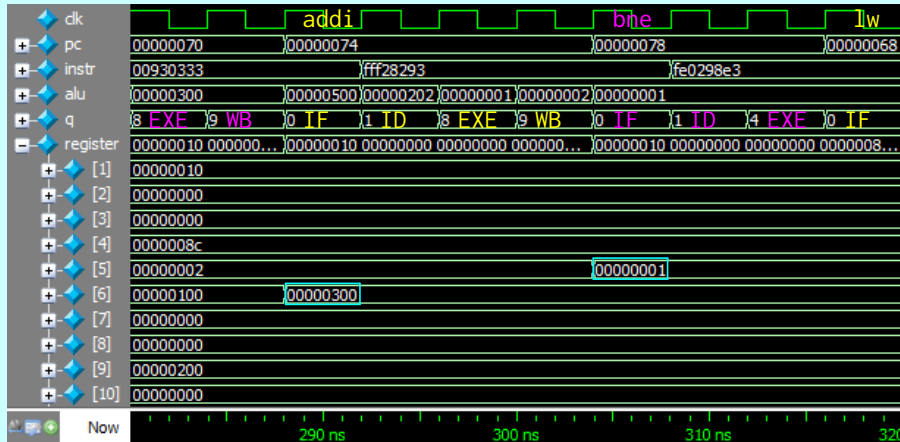
(74)      addi x5, x5, -1      # x5 <- x5 - 1 = 2
(78)      bne  x5, x0, loop    # if x5 != 0, goto loop
(68) loop:  lw   x9, 0(x4)      # x9 <- memory[x4+0] = 0x200
    
```

シミュレーション7



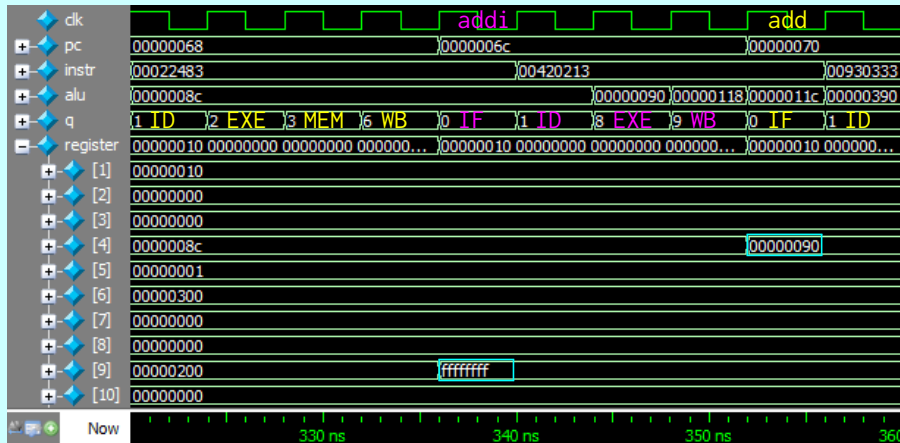
(6c) `addi x4, x4, 4` # $x4 \leftarrow x4 + 4 = 0x8c$
 (70) `add x6, x6, x9` # $x6 \leftarrow x6 + x9 = 0x300$

シミュレーション 8



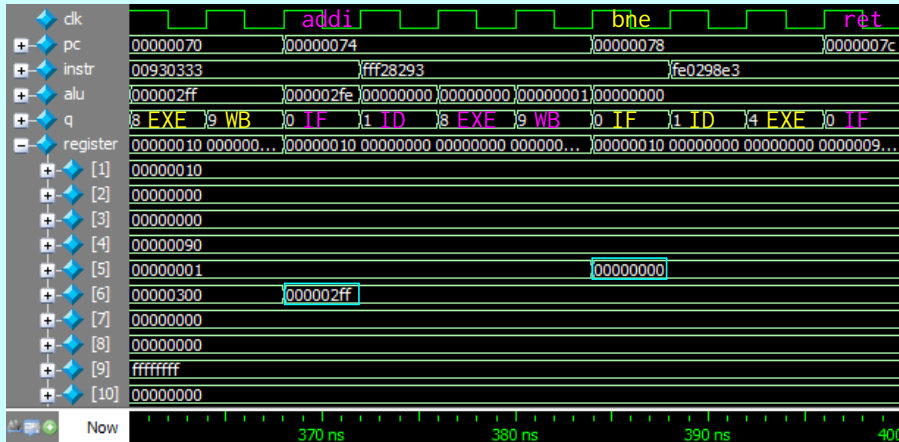
```
(74)      addi x5, x5, -1      # x5 <- x5 - 1 = 1
(78)      bne  x5, x0, loop    # if x5 != 0, goto loop
(68) loop:  lw   x9, 0(x4)     # x9 <- memory[x4+0] = -1
```

シミュレーション9



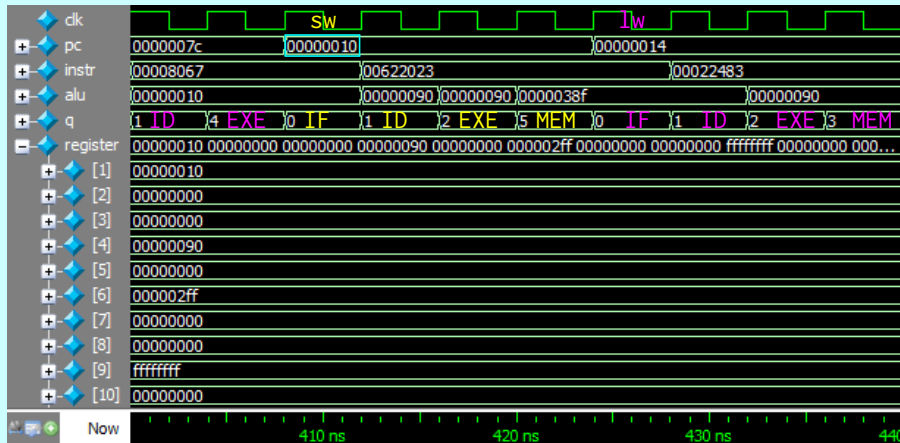
(6c) `addi x4, x4, 4` # $x4 \leftarrow x4 + 4 = 0x90$
 (70) `add x6, x6, x9` # $x6 \leftarrow x6 + x9 = 0x2ff$

シミュレーション10



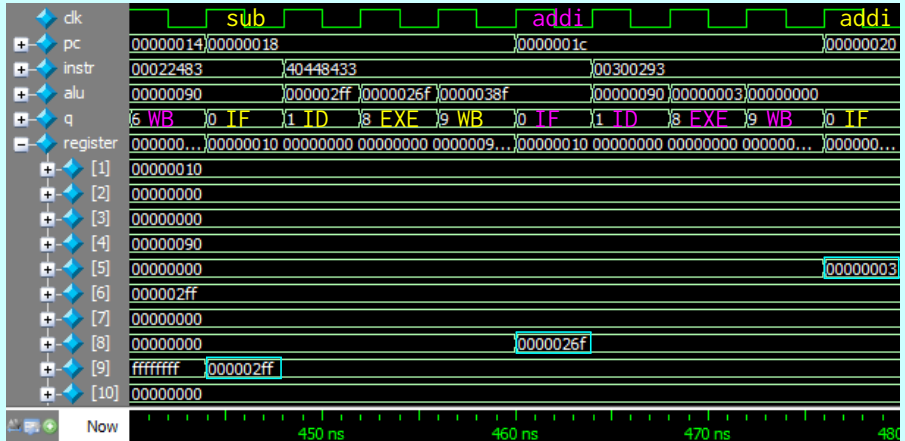
```
(74)      addi x5, x5, -1      # x5 <- x5 - 1 = 0
(78)      bne  x5, x0, loop    # if x5 != 0, goto loop
(7c)      ret   x1             # return from sum
```

シミュレーション11



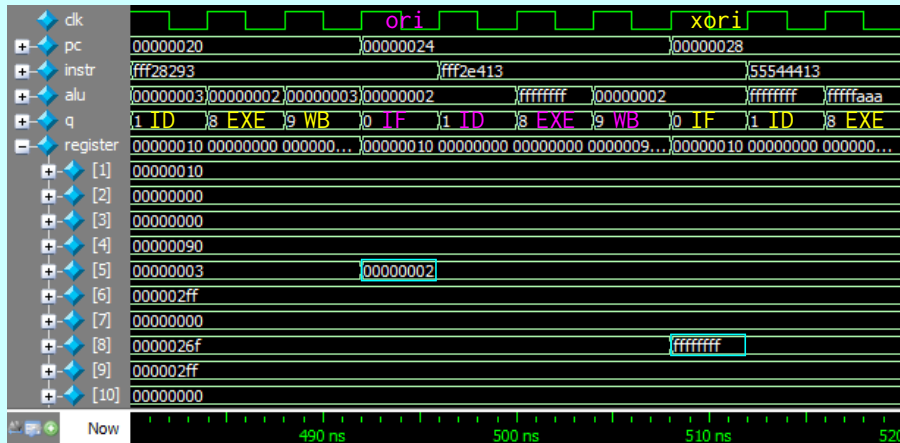
```
(10)      sw    x6, 0(x4)      # memory[x4+0] <- x6 = 0x2ff
(14)      lw    x9, 0(x4)      # x6 <- memory[x4+0] = 0x2ff
```

シミュレーション12



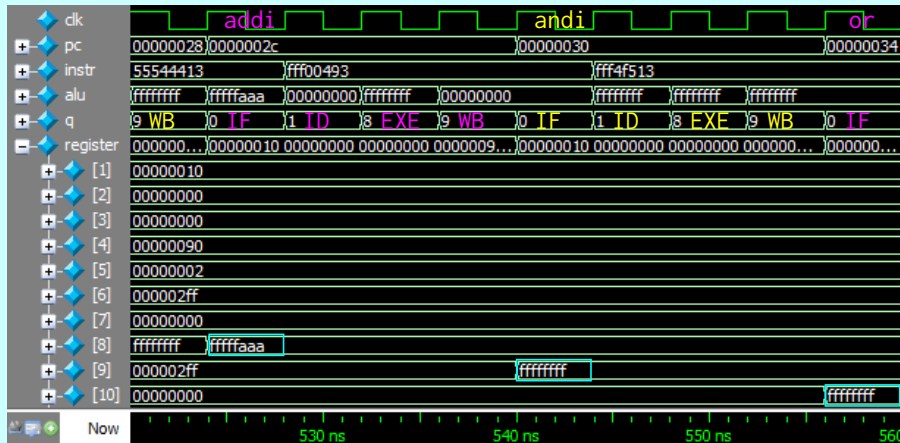
```
(18)      sub  x8, x9, x4      # x8 <- x9 - x4 = 0x26f
(1c)      addi x5, x0,  3      # x5 <- 3
(20) loop2: addi x5, x5, -1     # x5 <- x5 - 1 = 2
```

シミュレーション13



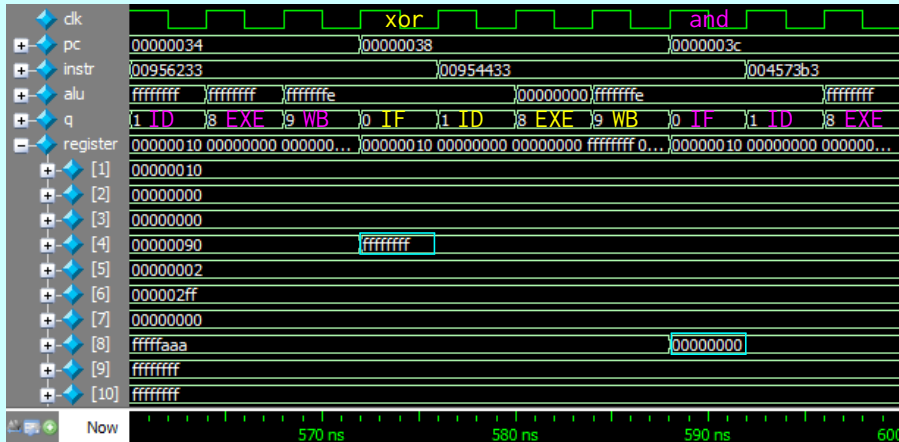
```
(24)      ori  x8, x5, -1      # x8 <- x5 | 0xffffffff = 0xffffffff
(28)      xori x8, x8, 0x555  # x8 <- x8 ^ 0x000000555 = 0xfffffaaa
```

シミュレーション14



```
(2c)    addi x9, x0, -1    # x9 <- 0xffffffff
(30)    andi x10,x9, -1    # x10<- x9 & 0xffffffff = 0xffffffff
(34)    or   x4, x10, x9    # x4 <- x10 | x9 = 0xffffffff
```

シミュレーション15



```
(38)      xor    x8, x10, x9    # x8 <- x10 ^ x9 = 0x00000000
```

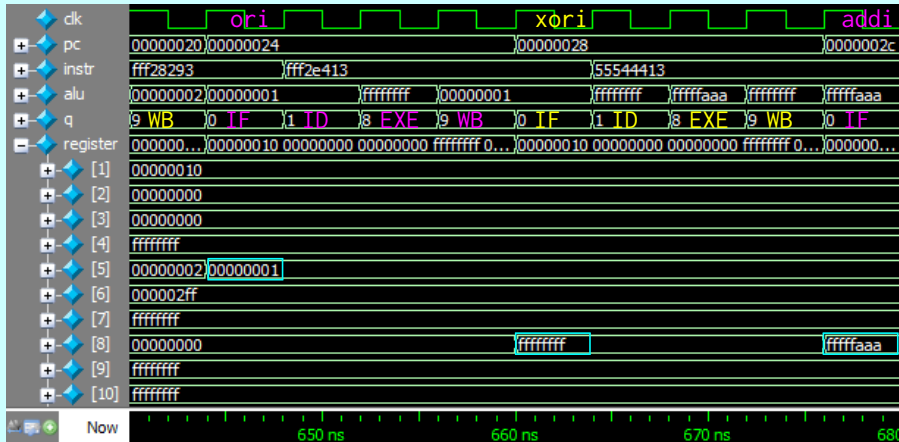
```
(3c)      and x7, x10, x4    # x7 <- x10 & x4 = 0xffffffff
```

シミュレーション16



```
(40)      beq  x5, x0, shift # if x5 = 0, goto shift
(44)      jal  x0, loop2    # jump to loop2
(20) loop2: addi x5, x5, -1  # x5 <- x5 - 1 = 1
```

シミュレーション17



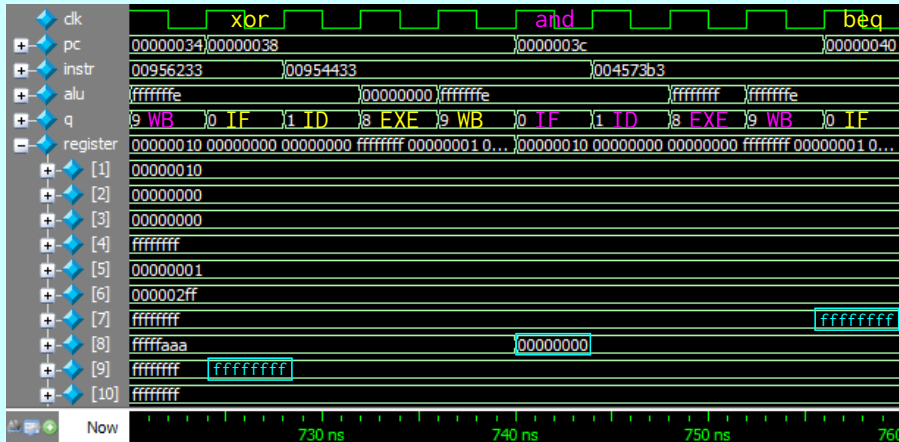
```
(24)      ori  x8, x5, -1      # x8 <- x5 | 0xffffffff = 0xffffffff
(28)      xori x8, x8, 0x555  # x8 <- x8 ^ 0x000000555 = 0xfffffaaa
(2c)      addi x9, x0, -1      # x9 <- 0xffffffff
```

シミュレーション18



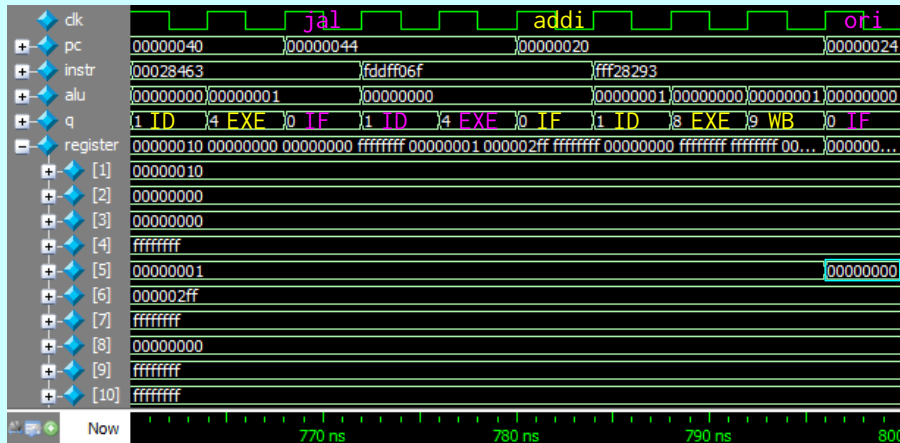
```
(30)    andi x10,x9, -1    # x10<- x9 & 0xffffffff = 0xffffffff
(34)    or   x4, x10, x9   # x4 <- x10 | x9 = 0xffffffff
```

シミュレーション19



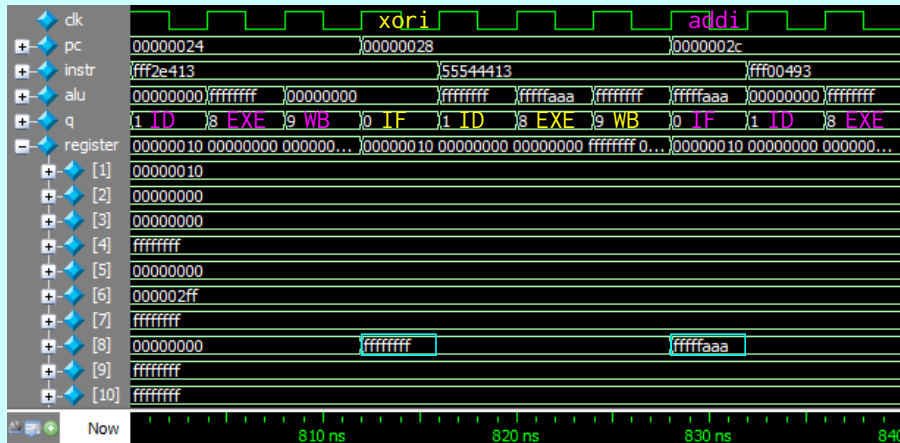
```
(38)      xor   x8, x10, x9    # x8 <- x10 ^ x9 = 0x00000000
(3c)      and   x7, x10, x4    # x7 <- x10 & x4 = 0xffffffff
(40)      beq   x5, x0, shift  # if x5 = 0, goto shift
```

シミュレーション 20



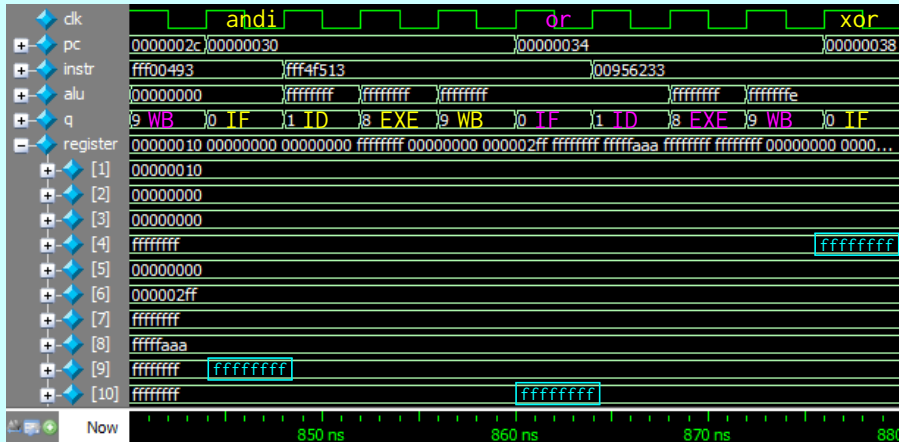
```
(44)      jal  x0, loop2      # jump to loop2
(20) loop2: addi x5, x5, -1    # x5 <- x5 - 1 = 0
(24)      ori  x8, x5, -1     # x8 <- x5 | 0xffffffff = 0xffffffff
```

シミュレーション21



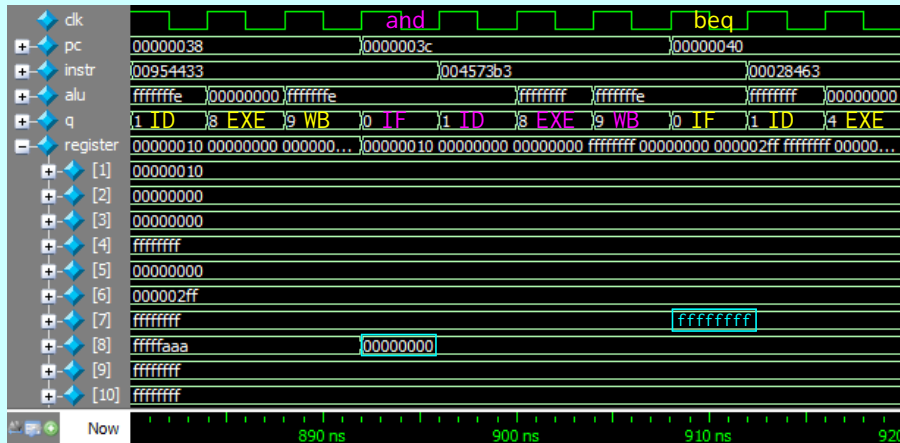
(28) xori x8, x8, 0x555 # x8 <- x8 ^ 0x00000555 = 0xfffffaaa
(2c) addi x9, x0, -1 # x9 <- 0xffffffff

シミュレーション22



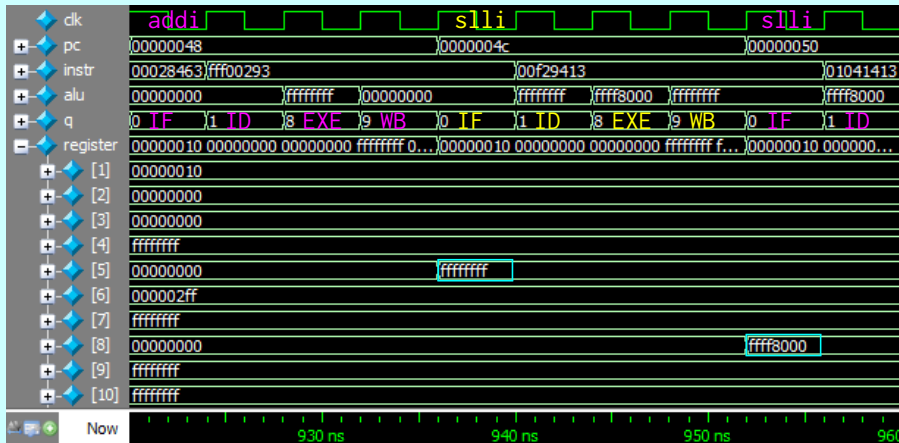
```
(30)      andi x10,x9, -1      # x10<- x9 & 0xffffffff = 0xffffffff
(34)      or   x4, x10, x9     # x4 <- x10 | x9 = 0xffffffff
(38)      xor  x8, x10, x9     # x8 <- x10 ^ x9 = 0x00000000
```

シミュレーション23



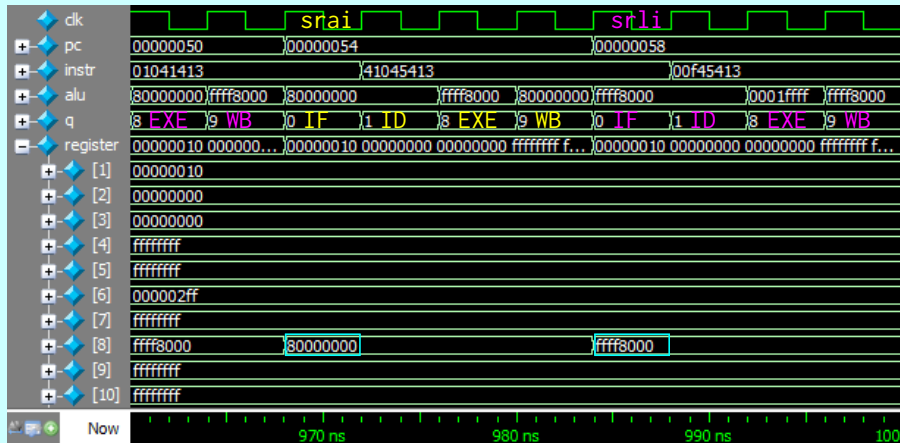
```
(3c)    and  x7, x10, x4    # x7 <- x10 & x4 = 0xffffffff
(40)    beq  x5, x0, shift  # if x5 = 0, goto shift
```

シミュレーション24



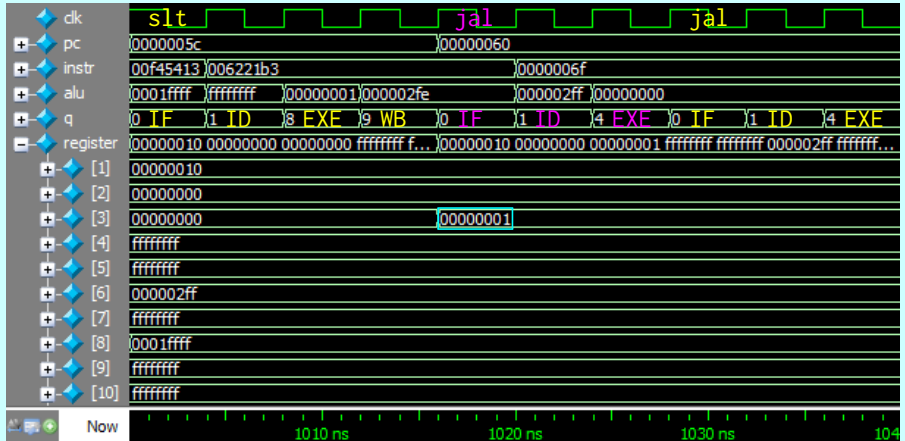
```
(48) shift:  addi x5, x0, -1      # x5 <- 0xffffffff
(4c)          slli x8, x5, 15     # x8 <- 0xffffffff << 15 = 0xffff8000
(50)          slli x8, x8, 16     # x8 <- 0xffff8000 << 16 = 0x80000000
```

シミュレーション 25



```
(54)      srai x8, x8, 16      # x8 <- 0x80000000 >>> 16 = 0xffff8000
(58)      srli x8, x8, 15      # x8 <- 0xffff8000 >> 15 = 0x0001ffff
```

シミュレーション 26



```
(5c)          slt  x3, x4, x6    # x3 <- 0xffffffff < 0x000002ff = 1
(60) finish:  jal  x0, finish    # dead loop
(60) finish:  jal  x0, finish    # dead loop
```

dff32.v

```
'timescale 1ns/1ns
module dff32  (d,clk,clrn,  q);
    input      [31:0] d;
    input      clk, clrn;
    output reg [31:0] q;
    always @(negedge clrn or posedge clk)
        if (!clrn)  q <= 0;
        else       q <= d;
endmodule
```

[dff32.v](#)

dffe32.v

```
'timescale 1ns/1ns
module dffe32 (d,clk,clrn,e,q);
    input      [31:0] d;
    input      clk, clrn, e;
    output reg [31:0] q;
    always @(negedge clrn or posedge clk)
        if (!clrn) q <= 0;
        else if (e) q <= d;
endmodule
```

[dffe32.v](#)

課題 VIII (100 点 + 100 点)

- ① 単一サイクル CPU とマルチサイクル CPU を設計したとする。単一サイクル CPU のクロック周波数は 200MHz である。マルチサイクル CPU のクロック周波数は 1GHz である。マルチサイクル CPU では、「lw」命令は5クロックサイクル、「bne」は3サイクル、残りは4サイクルかかる。2つの CPU が次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。

```
.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
```

- ② オプション (+100 点) : マルチサイクルコンピュータを設計とシミュレーションしなさい。