MIPS32: Single-Cycle CPU

Seungwan Noh

Pusan National University

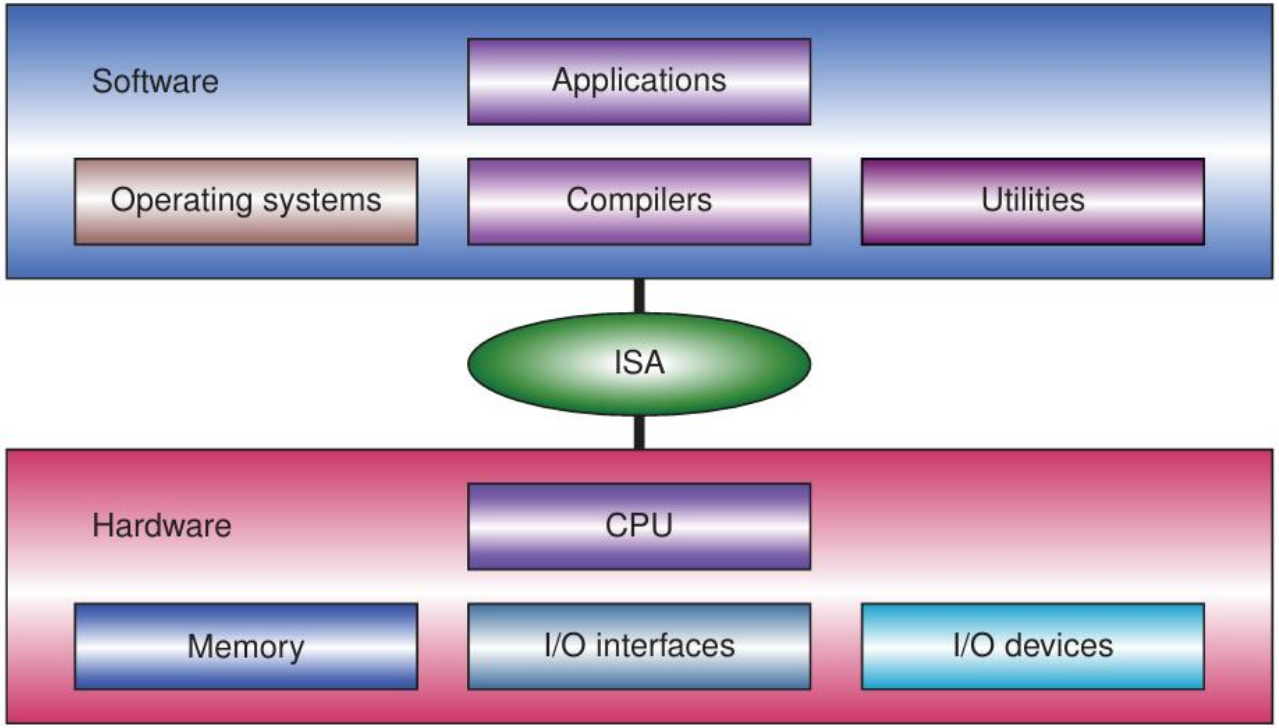
Department of Electronics Engineering

Contents

- Instruction Set Architecture
- Single-cycle CPU Design
- Verilog HDL Implementation
- Simulation
- References

Instruction Set Architecture

- "What a CPU should do."
- CPU designers design a CPU to implement the ISA.
- An interface between SW and HW.



Instruction Set Architecture

- Operand types

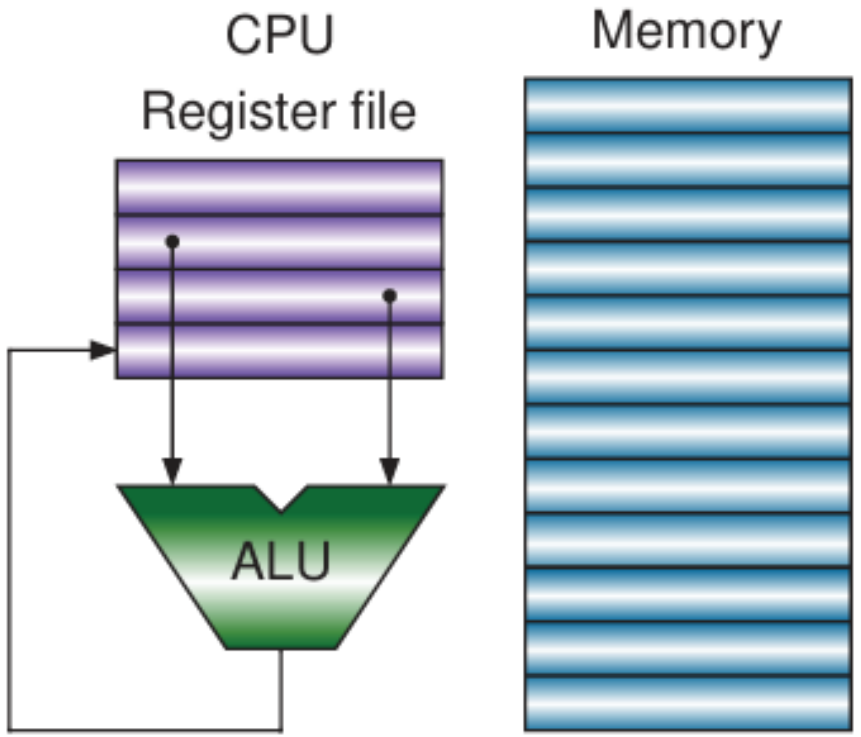
Operand type	Bits	Value range	Corresponding to C
Byte	8	−128 to +127	signed char
Unsigned byte	8	0 to 255	unsigned char
Half word	16	−32,768 to +32,767	short int
Unsigned half word	16	0 to 65,535	unsigned short int
Word	32	−2,147,483,648 to +2,147,483,647	int
Unsigned word	32	0 to 4,294,967,295	unsigned int
Single-precision FP	32		float
Double-precision FP	64		double

Instruction Set Architecture

- Instruction types
  - ✓Arithmetic operation
  - ✓Logic operation
  - ✓Shift operation
  - ✓Memory access
  - ✓I/O access
  - ✓Control transfer
  - ✓Floating-point calculation
  - ✓System control

Instruction Set Architecture

- Register-register architecture
  - ✓All operands are explicitly in a general-purpose register file.
  - ✓Almost all RISC type ISAs use this architecture.



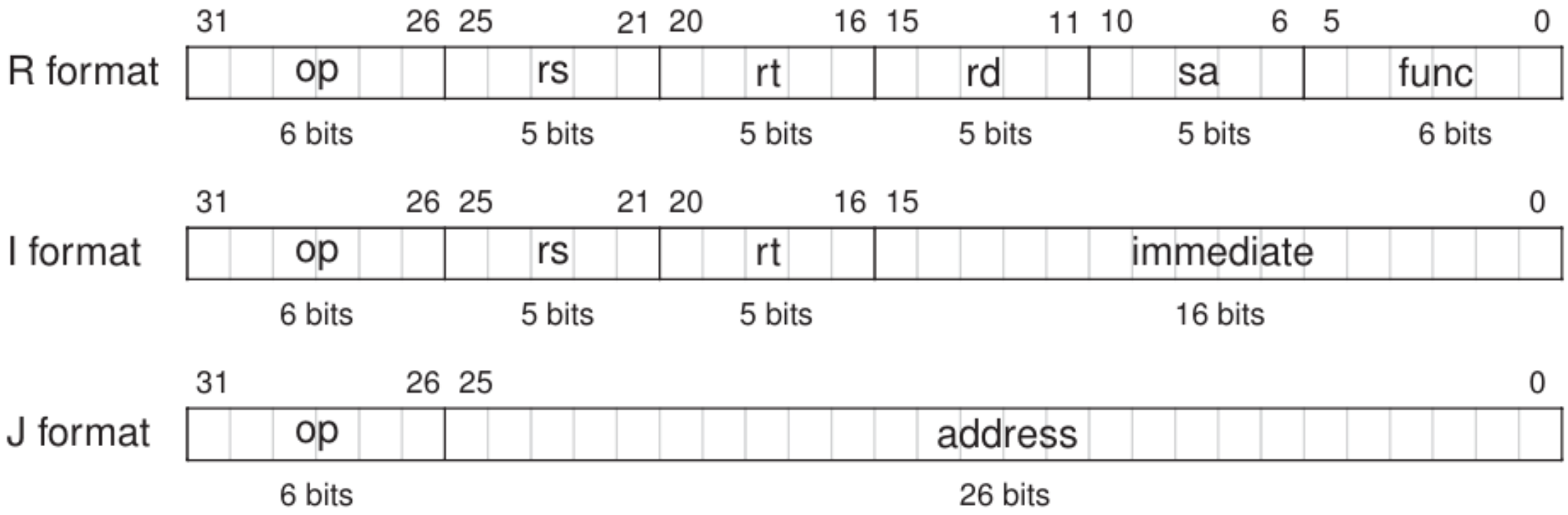
Instruction Set Architecture

- Addressing Modes
  - ✓Register Operand Addressing
    - The operand is in a register file.
    - *add r3, r1, r2*
  - ✓Immediate Addressing
    - The operand is in instruction.
    - *addi r1, r1, -1*
  - ✓Direct Addressing
    - The operand is in the memory whose address is given in the instruction.
    - *add r3, r1, [0x1234]*
  - ✓Register Indirect Addressing
    - The operand is in the memory whose address is given by a register.
    - *add r3, r1, (r2)*
  - ✓Offset Addressing
    - The operand is in the memory whose address is the sum of an offset and a register.
    - *add r3, r1, 0x1234(r2)*

Instruction Set Architecture

- MIPS32 Instruction Format and Registers
- R (register)
  - ✓*op* is 0, the operation is specified by *func*.
  - ✓*Shift* instruction use *sa* to specify the shift amount, *rt* is shifted to *rd*.
  - ✓Other R-format read *rs, rt* and write the result to *rd*.
- I (immediate)
  - ✓Use *immediate* as the second source operand.
  - ✓The 16-bit *immediate* must be sign-extended into 32 bits.
  - ✓The first source operand is *rs*, and the result is written into *rt*.
  - ✓The conditional branch compare *rs* and *rt*, *immediate* as an offset for the branch target address.
  - ✓The load/store instructions load/store data of *rt* to memory, the memory address is the sum of *rs* and *immediate*.
- J (jump)
  - ✓Shifted to the left by 2 bits to form a low 28 bits of the jump target address.

Instruction Set Architecture



Instruction Set Architecture

- MIPS general-purpose registers

Register name	Register number	Use
$zero	0	Constant 0
$at	1	Assembler temporary
$v0 to $v1	2 to 3	Function return value
$a0 to $a3	4 to 7	Function parameters
$t0 to $t7	8 to 15	Temporaries
$s0 to $s7	16 to 23	Saved temporaries
$t8 to $t9	24 to 25	Temporaries
$k0 to $k1	26 to 27	Reserved for OS kernel
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

Instruction Set Architecture

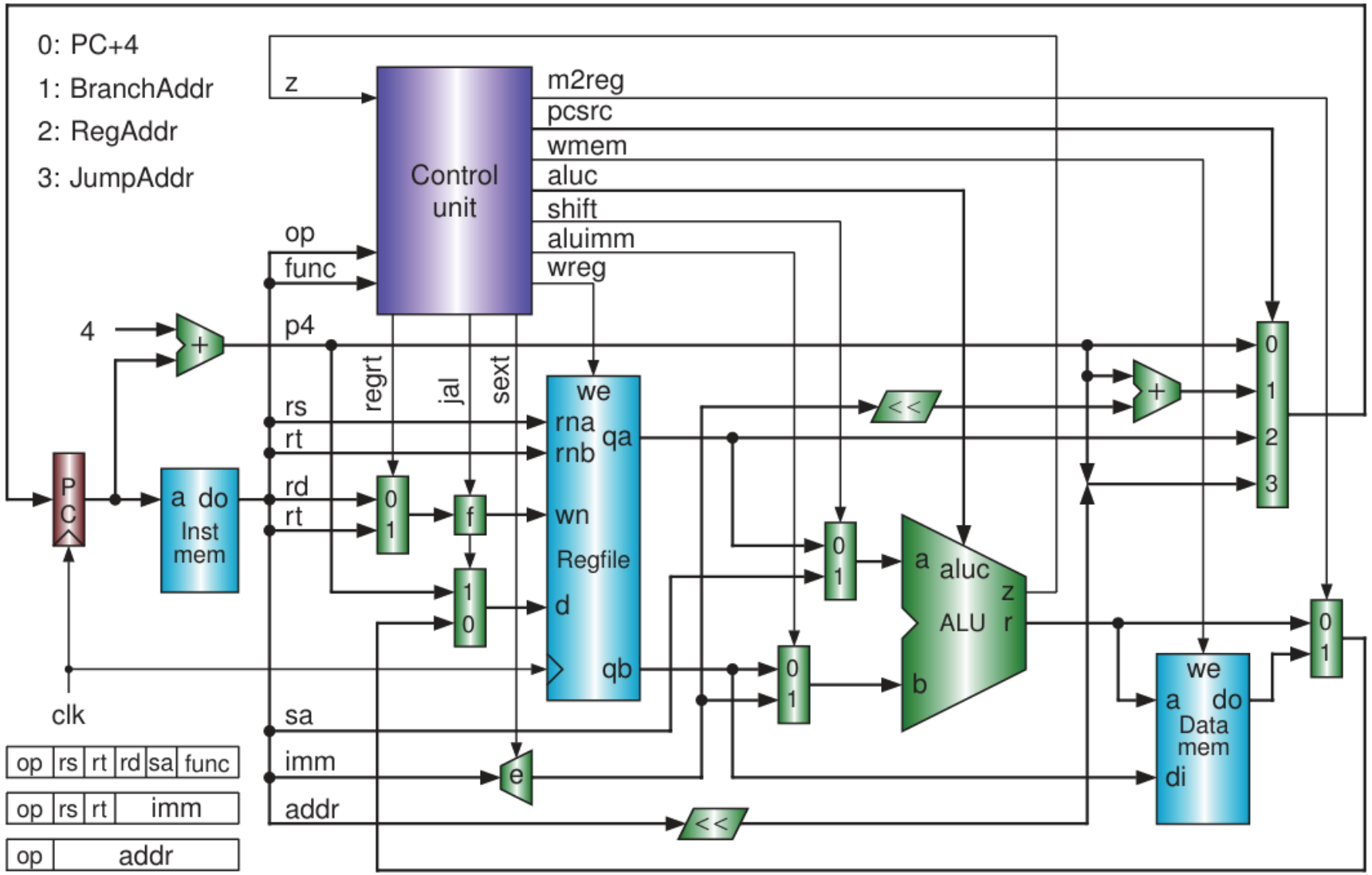
- 20 MIPS integer instructions

Inst.	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	Meaning
add	000000	rs	rt	rd	00000	100000	Register add
sub	000000	rs	rt	rd	00000	100010	Register subtract
and	000000	rs	rt	rd	00000	100100	Register AND
or	000000	rs	rt	rd	00000	100101	Register OR
xor	000000	rs	rt	rd	00000	100110	Register XOR
sll	000000	00000	rt	rd	sa	000000	Shift left
srl	000000	00000	rt	rd	sa	000010	Logical shift right
sra	000000	00000	rt	rd	sa	000011	Arithmetic shift right
jr	000000	rs	00000	00000	00000	001000	Register jump
addi	001000	rs	rt		Immediate		Immediate add
andi	001100	rs	rt		Immediate		Immediate AND
ori	001101	rs	rt		Immediate		Immediate OR
xori	001110	rs	rt		Immediate		Immediate XOR
lw	100011	rs	rt		offset		Load memory word
sw	101011	rs	rt		offset		Store memory word
beq	000100	rs	rt		offset		Branch on equal
bne	000101	rs	rt		offset		Branch on not equal
lui	001111	00000	rt		immediate		Load upper immediate
j	000010			address			Jump
jal	000011			address			Call

Single-Cycle CPU Design

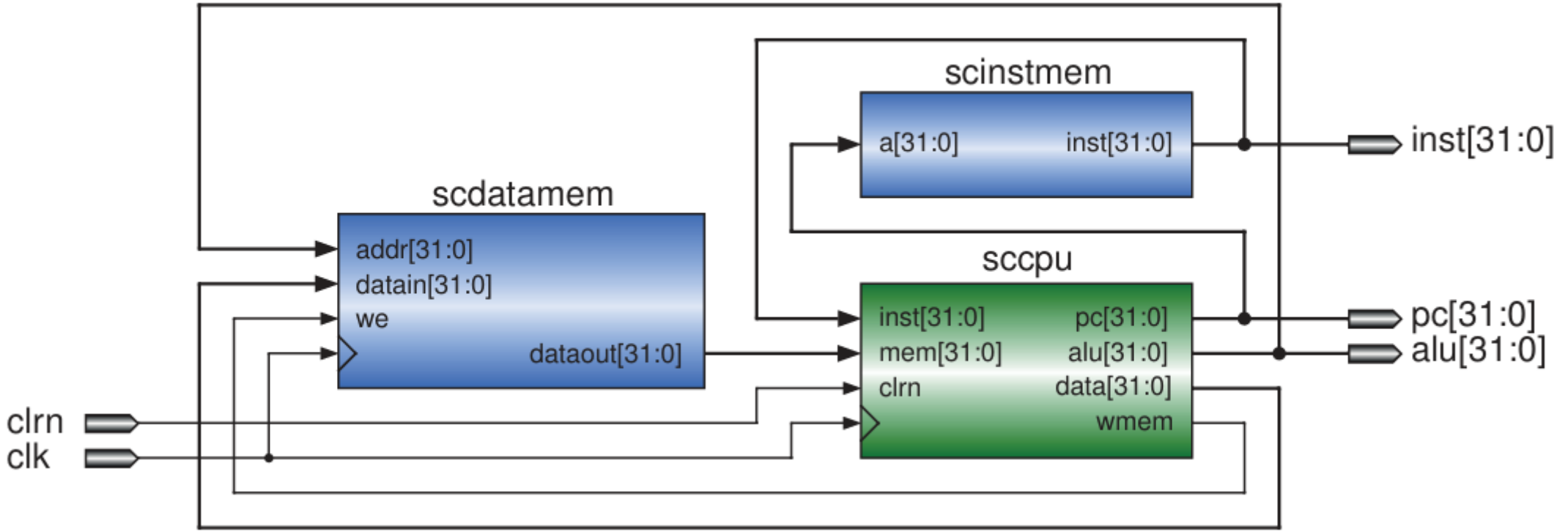
- A single-cycle CPU executes each instruction in on clock cycle.
- The most cost-effective but time-consuming CPU.
- The circuits for instruction fetch
  - ✓PC points a memory location of instructions.
  - ✓The instruction in the location will be executed by the CPU.
  - ✓PC is a byte address.
  - ✓Incremented by 4 because instruction is 32 bits (4 bytes).
- The circuits for instruction execution
  - ✓ALU
  - ✓Register File
  - ✓Control Unit

Single-Cycle CPU



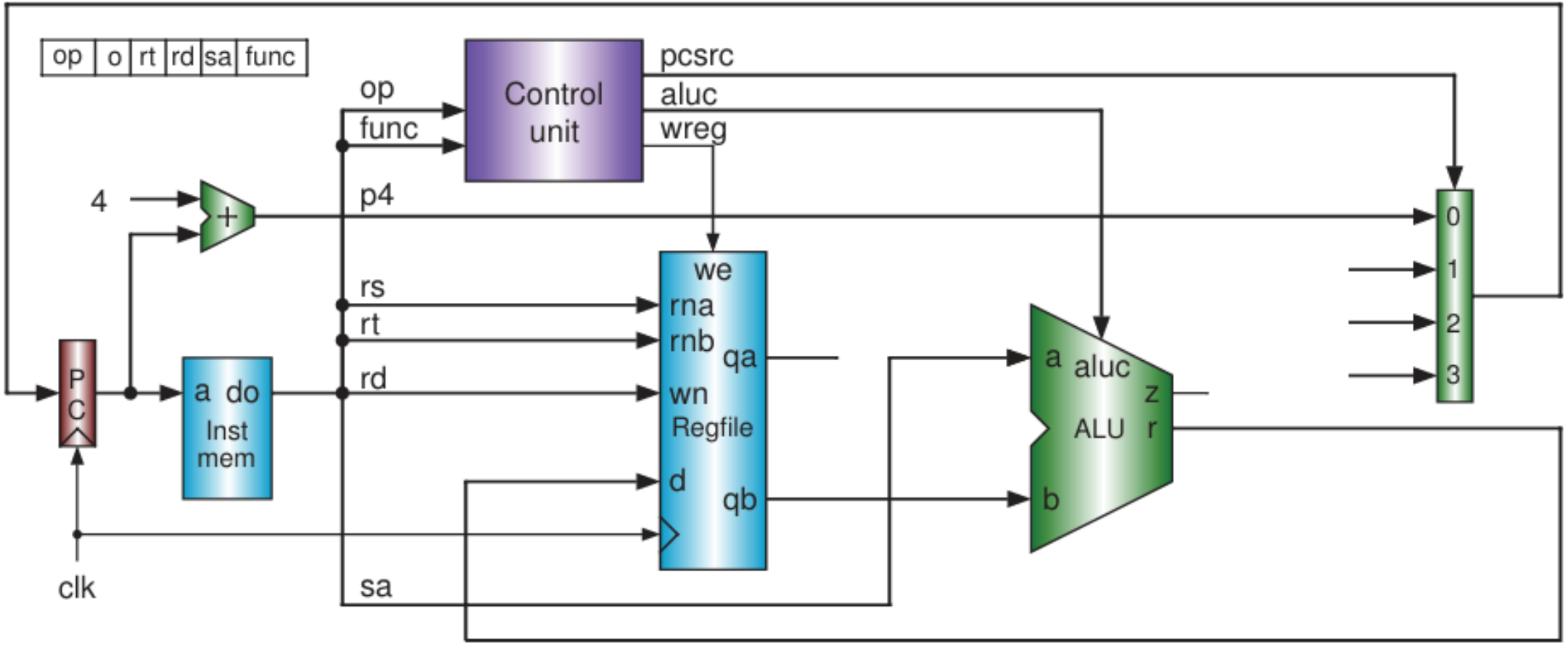
Single-Cycle CPU

- Implement an instruction cache and a data cache inside the CPU so that only one off-chip memory module can be used for storing both the program and data.



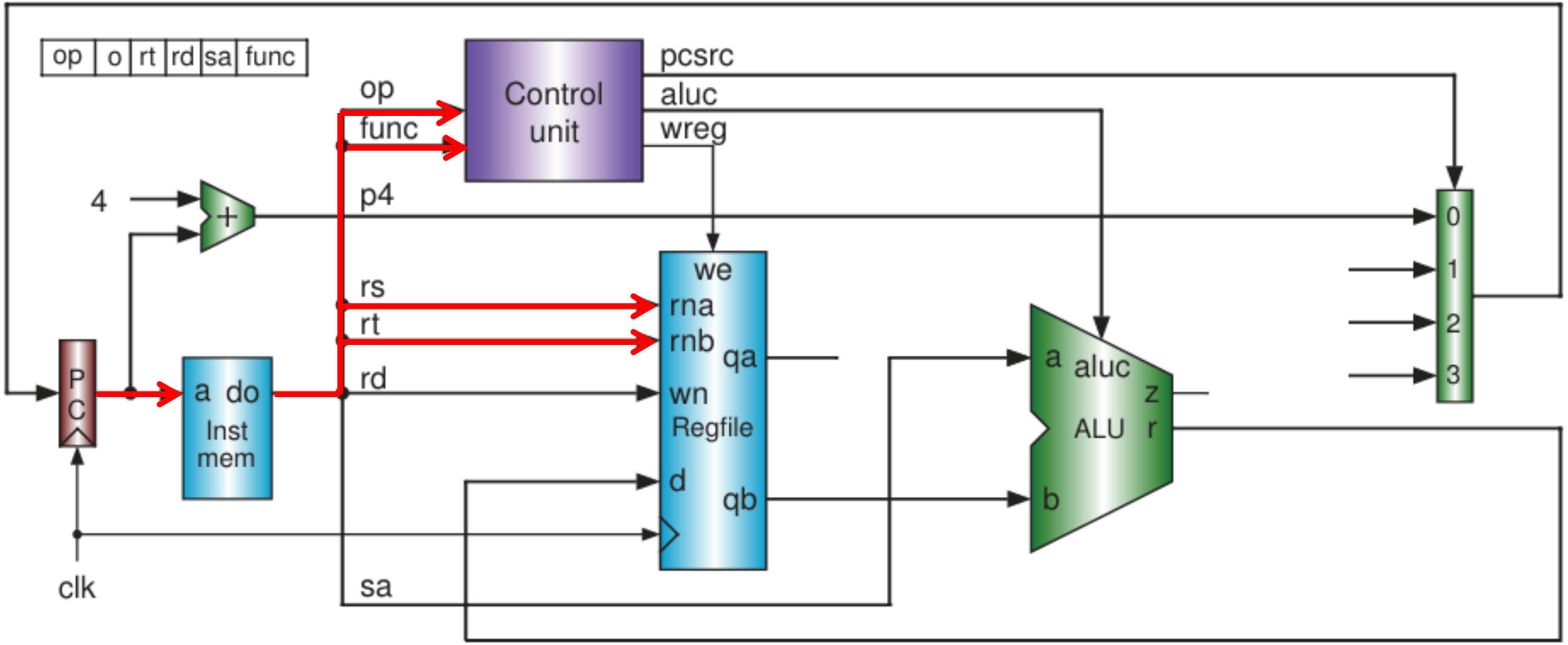
Single-Cycle CPU Design

- The circuits required by R-format instructions.



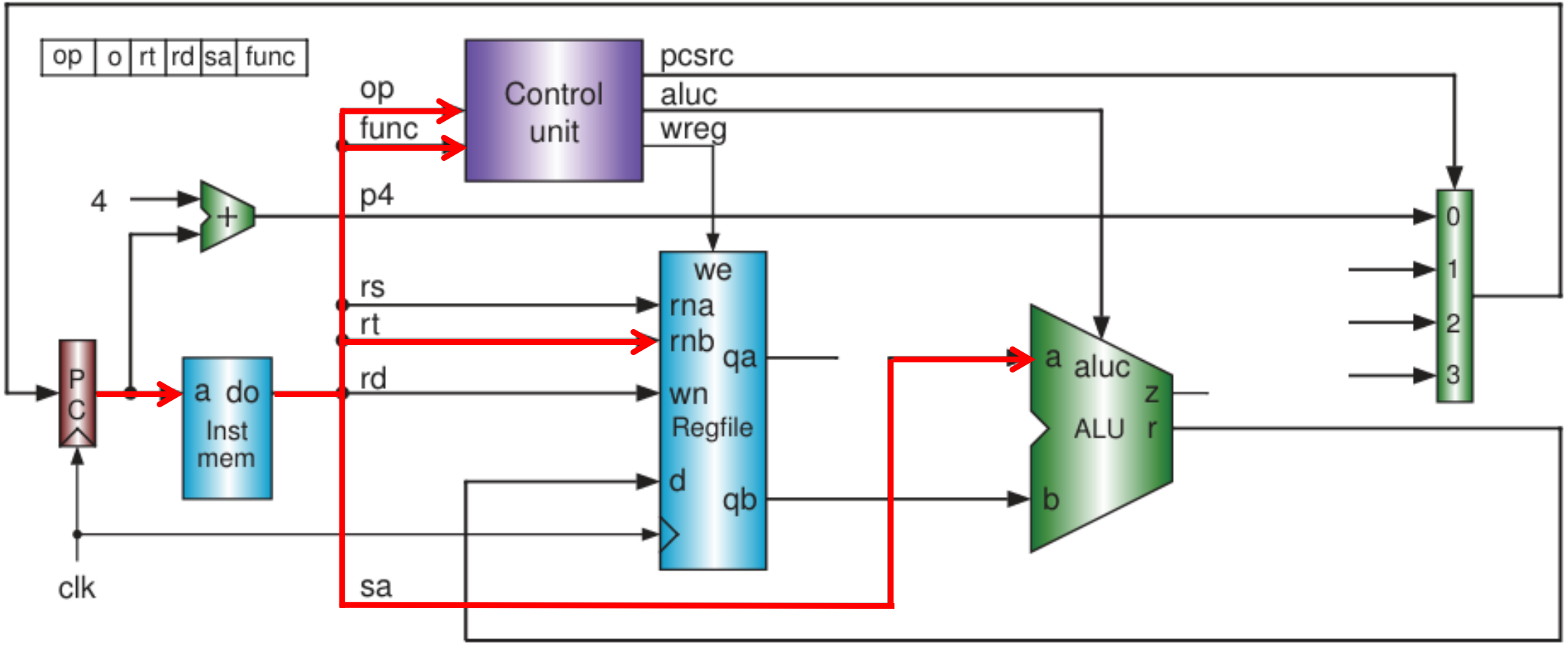
Single-Cycle CPU Design

- *add/sub/and/or/xor* (instruction fetch)



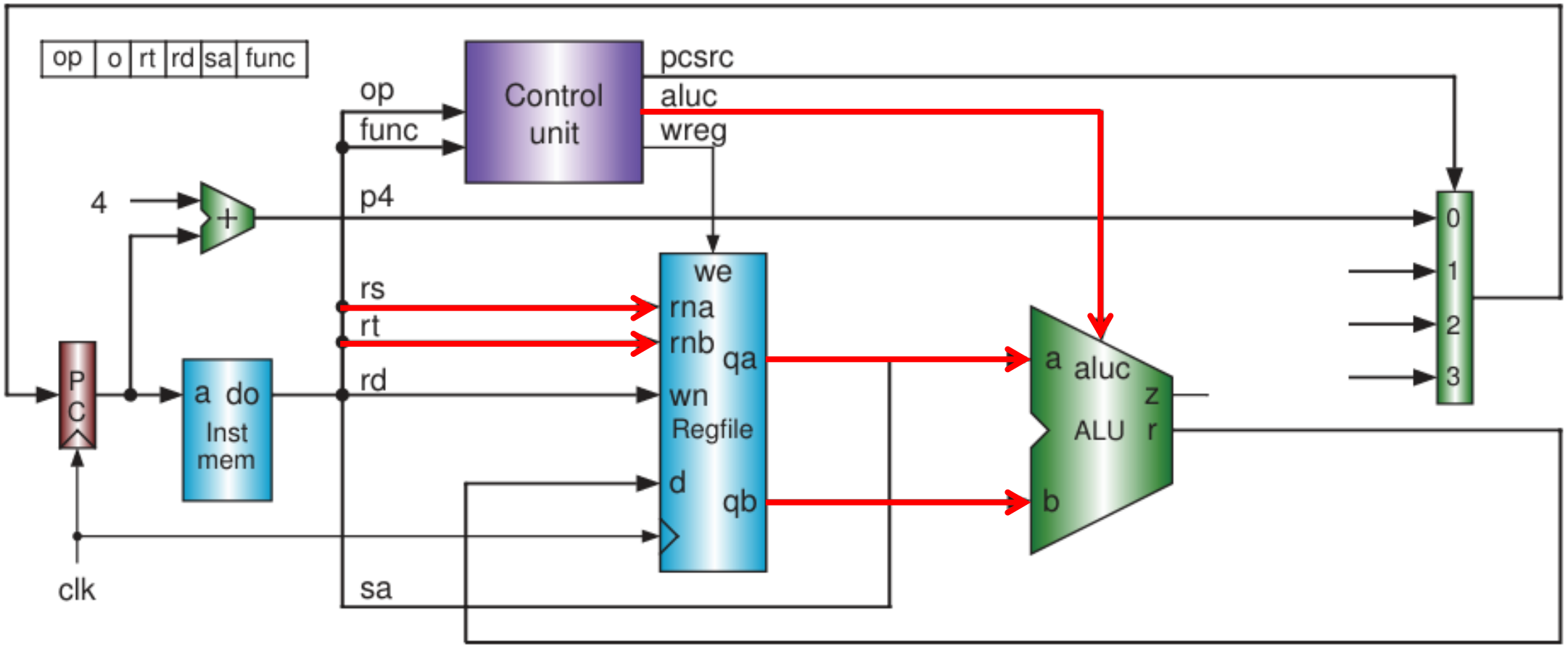
Single-Cycle CPU Design

- *sll/srl/sra* (instruction fetch)



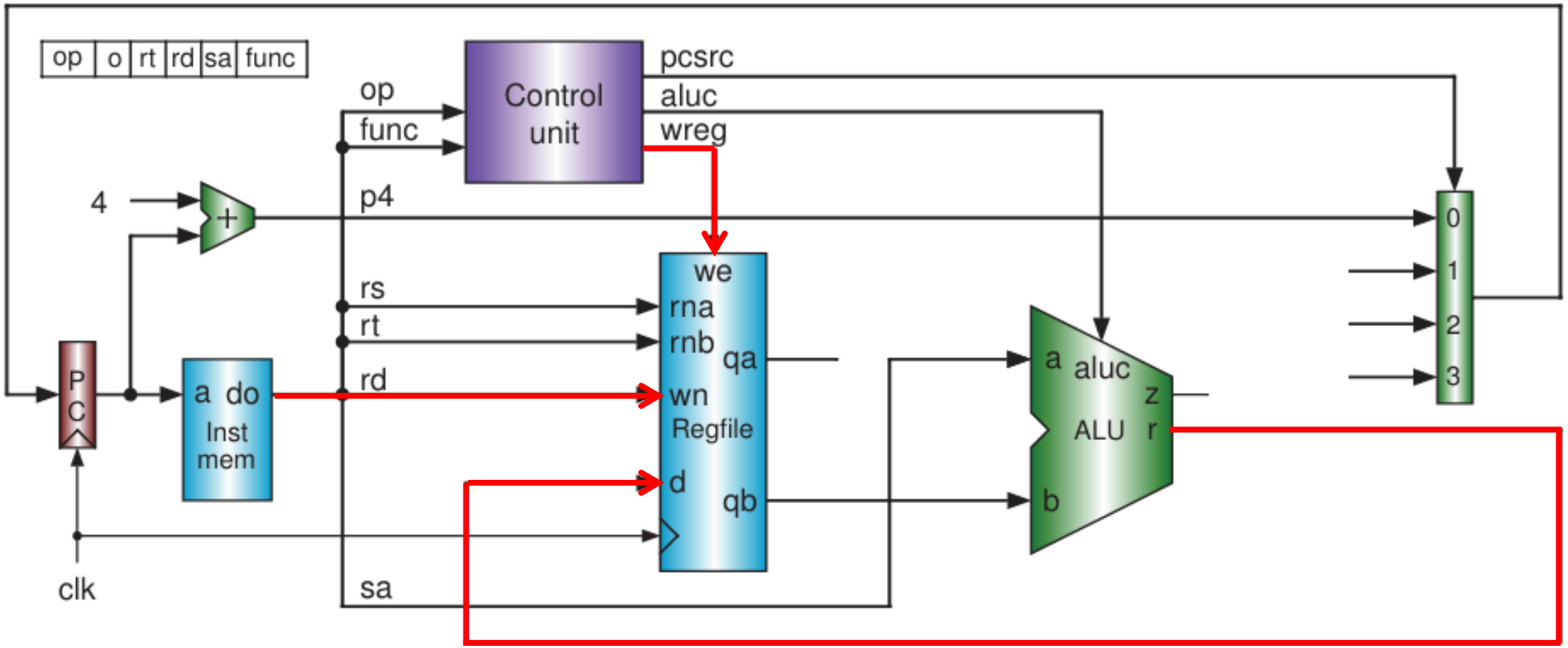
Single-Cycle CPU Design

- *add/sub/and/or/xor* (instruction execute)



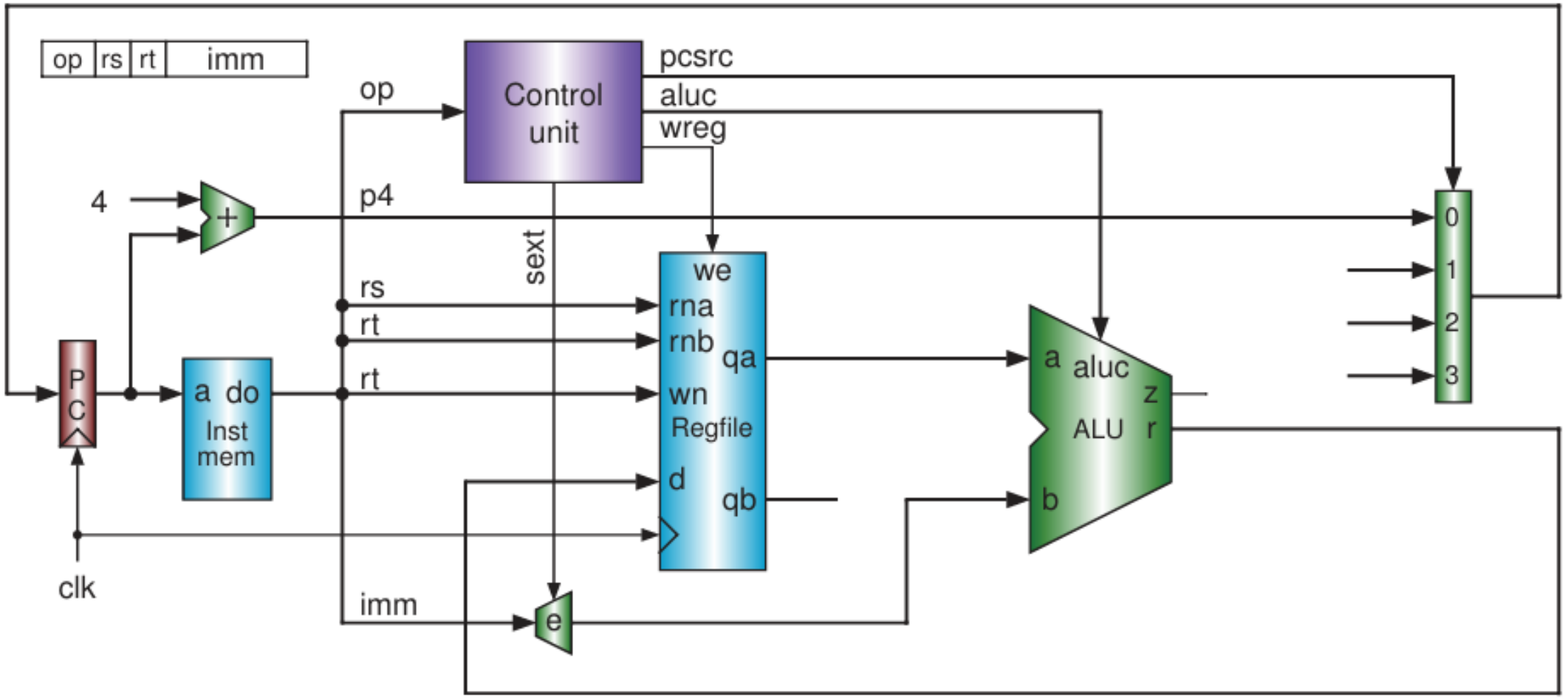
Single-Cycle CPU Design

- *add/sub/and/or/xor* (write back)



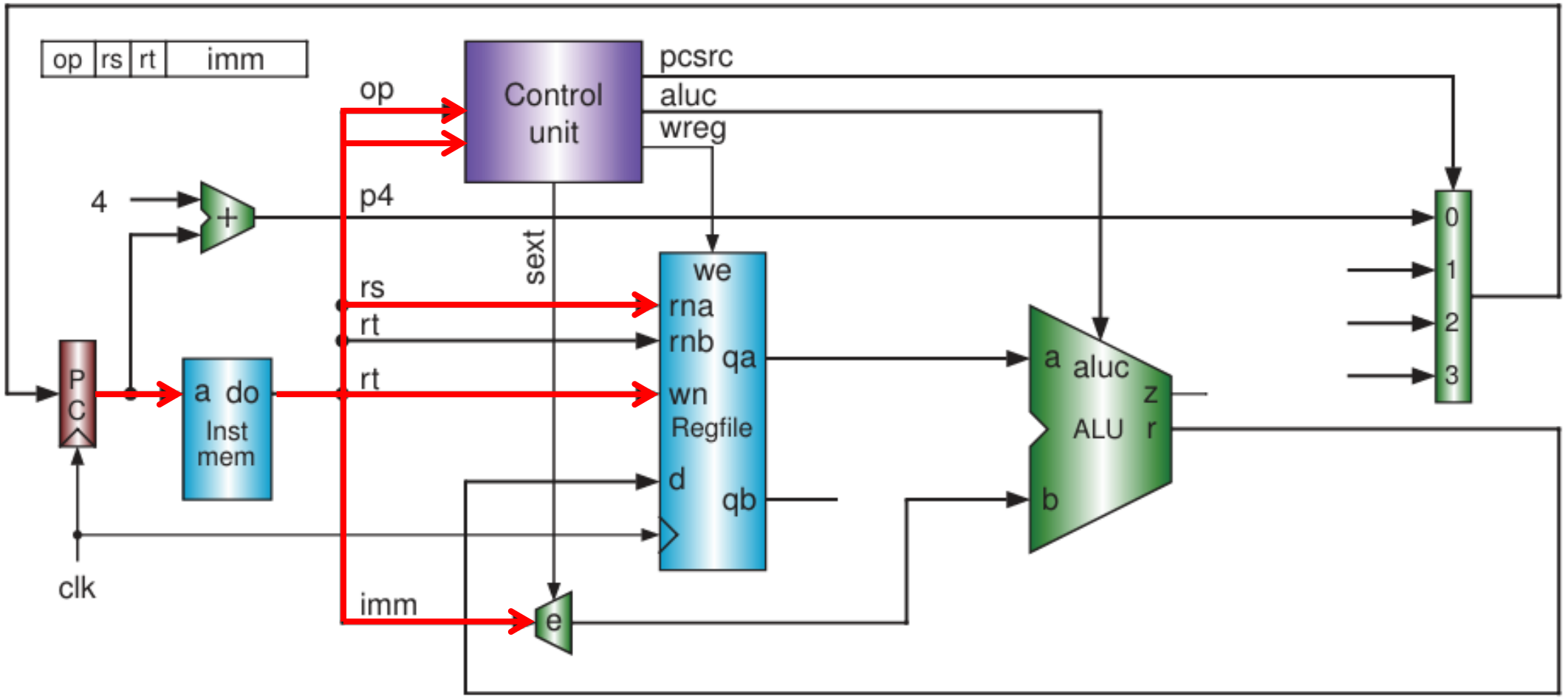
Single-Cycle CPU Design

- The circuits required by I-format instructions.



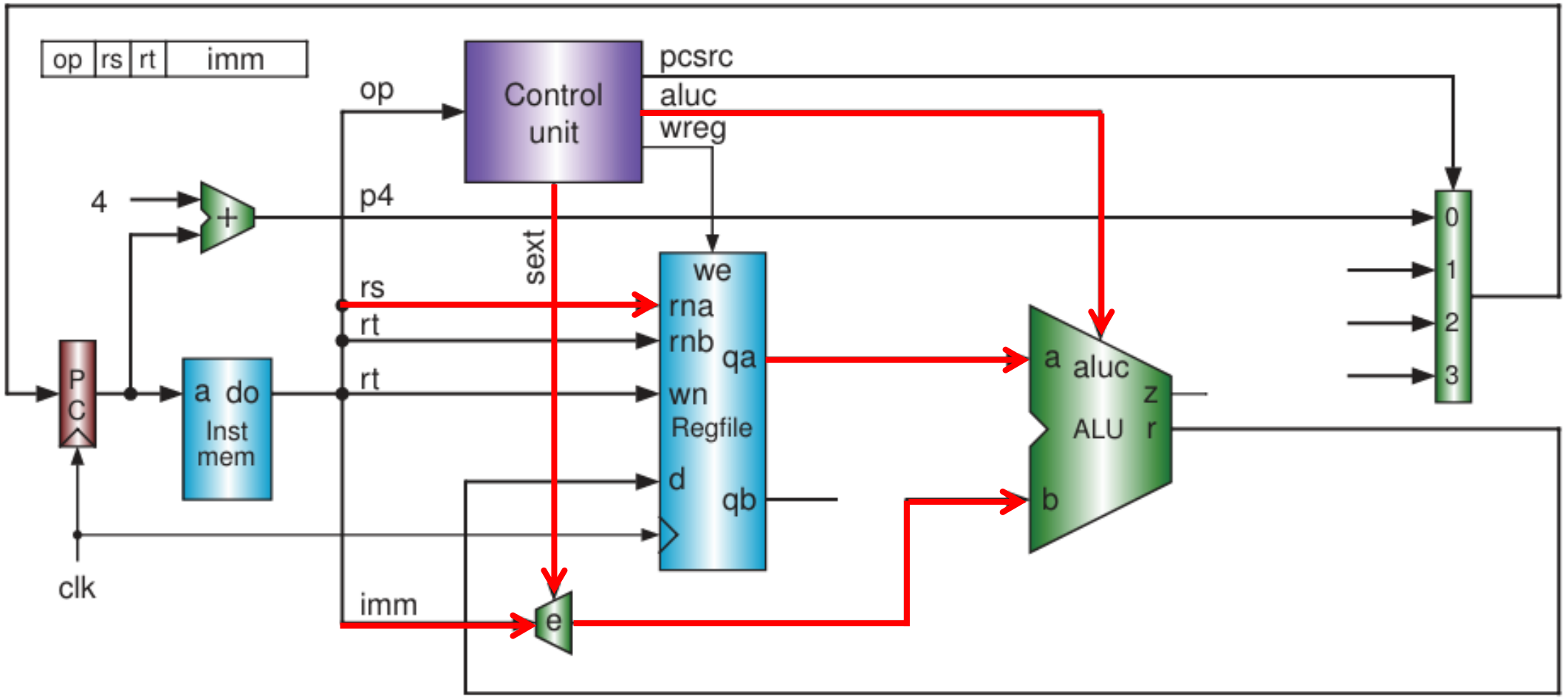
Single-Cycle CPU Design

- *addi/andi/ori/xori/lui* (instruction fetch)



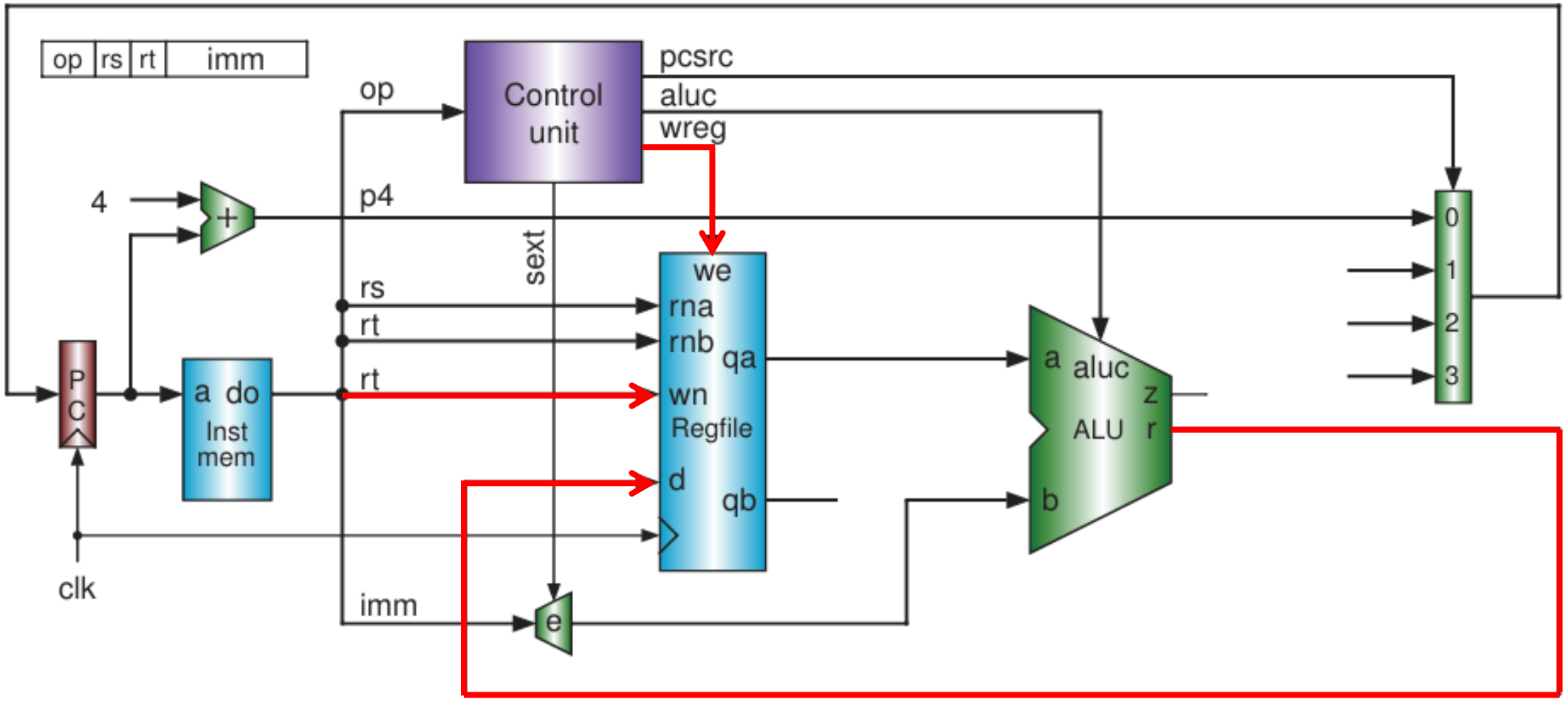
Single-Cycle CPU Design

- *addi/andi/ori/xori/lui* (instruction execute)



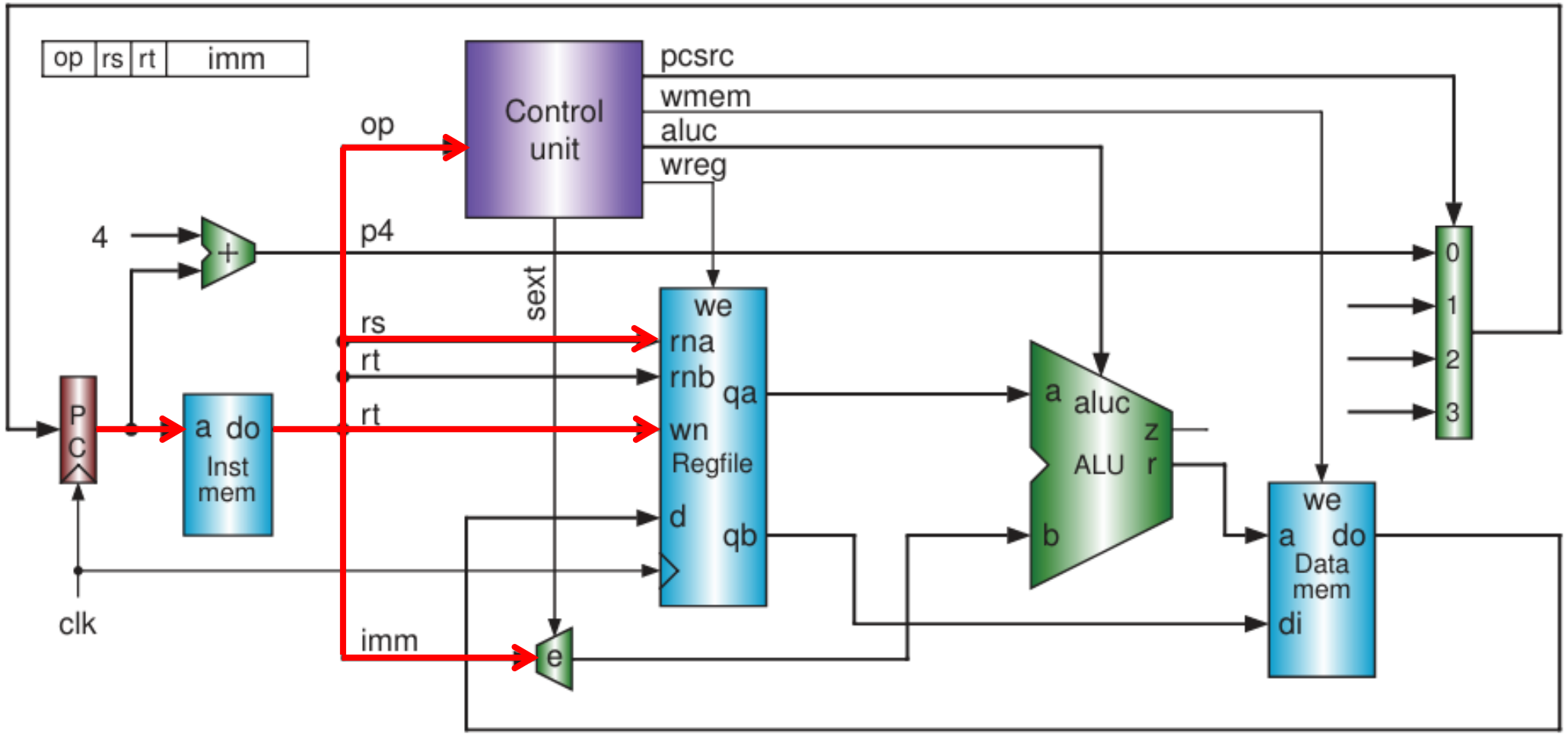
Single-Cycle CPU Design

- *addi/andi/ori/xori/lui* (write back)



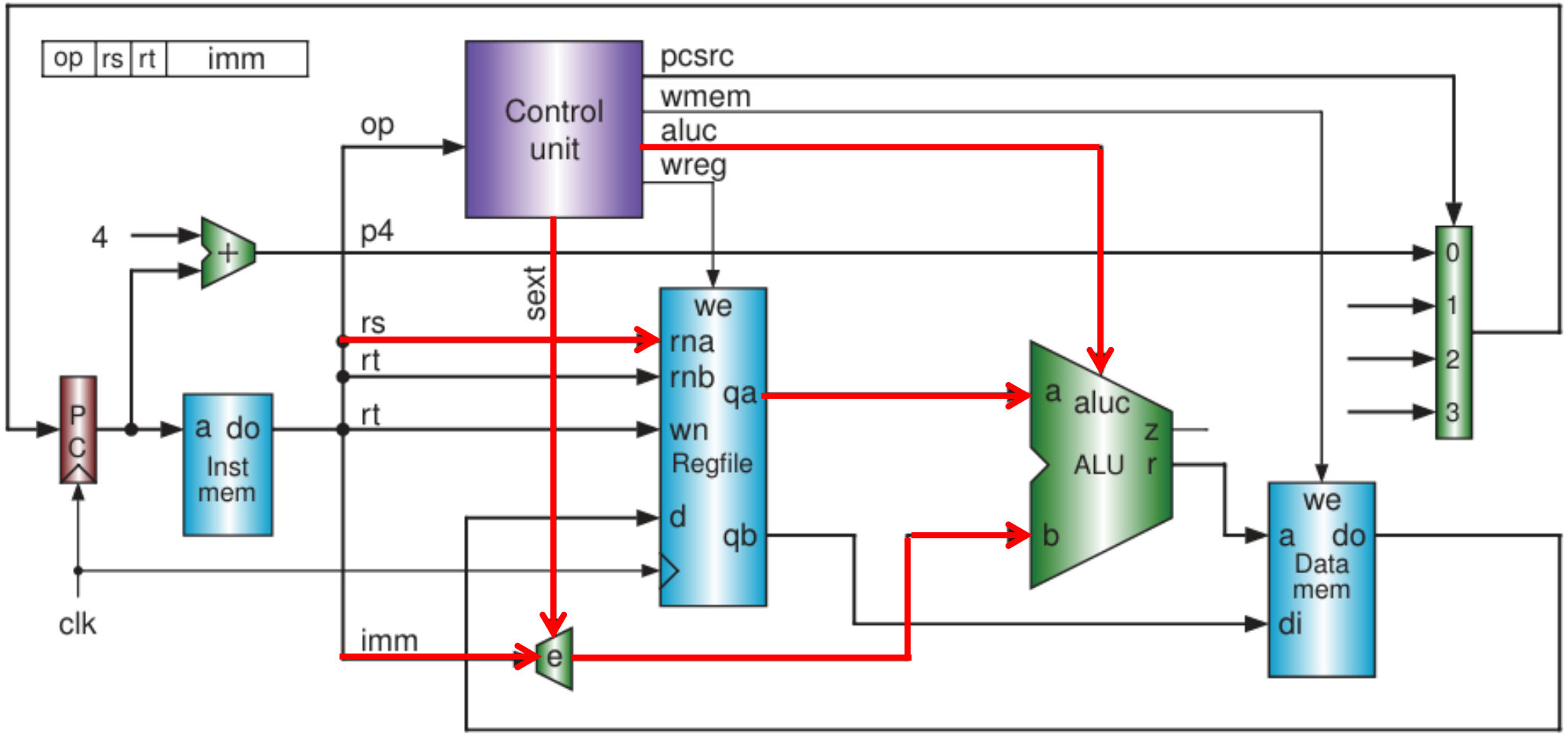
Single-Cycle CPU Design

- The circuit required by load/store instructions.



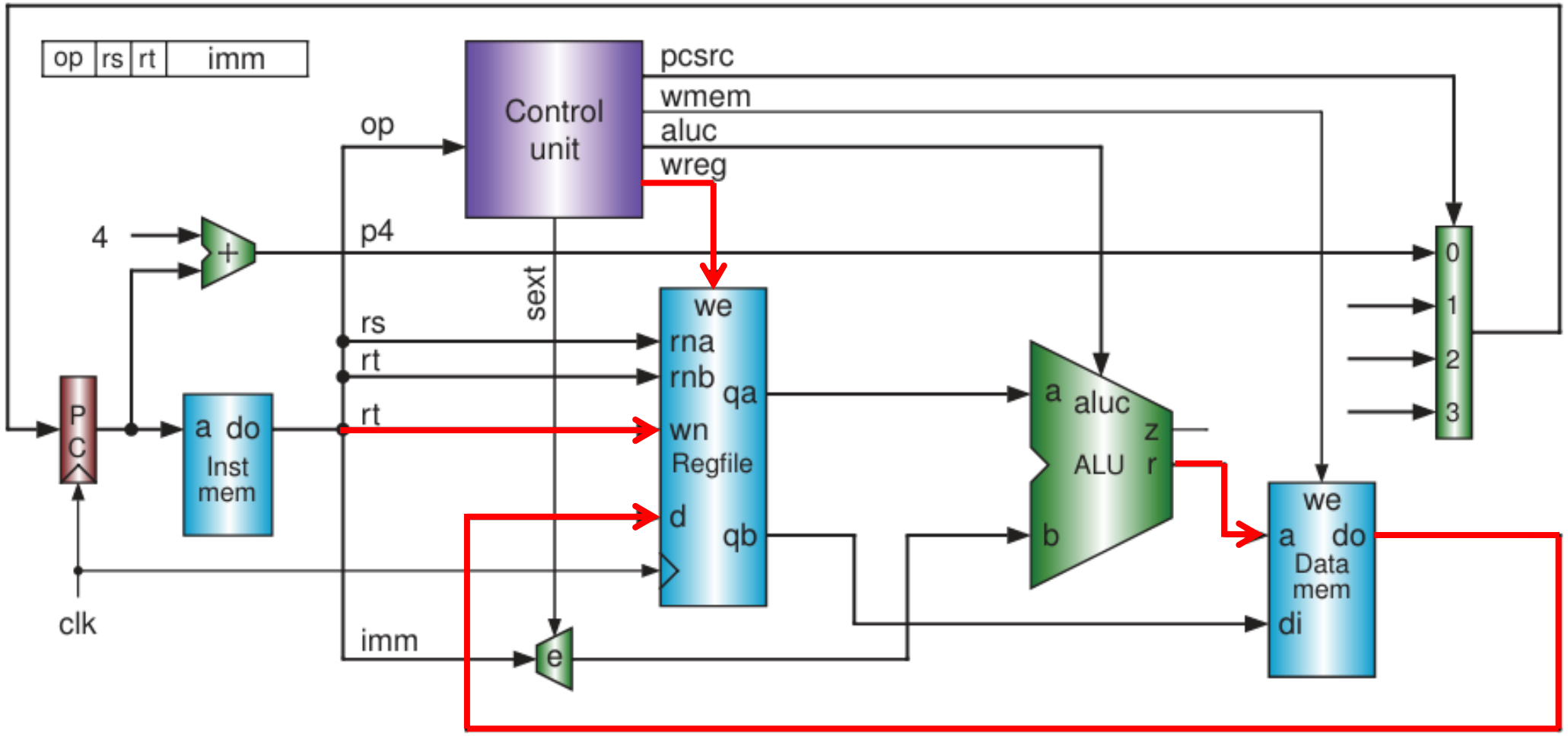
Single-Cycle CPU Design

- The circuit required by load/store instructions.



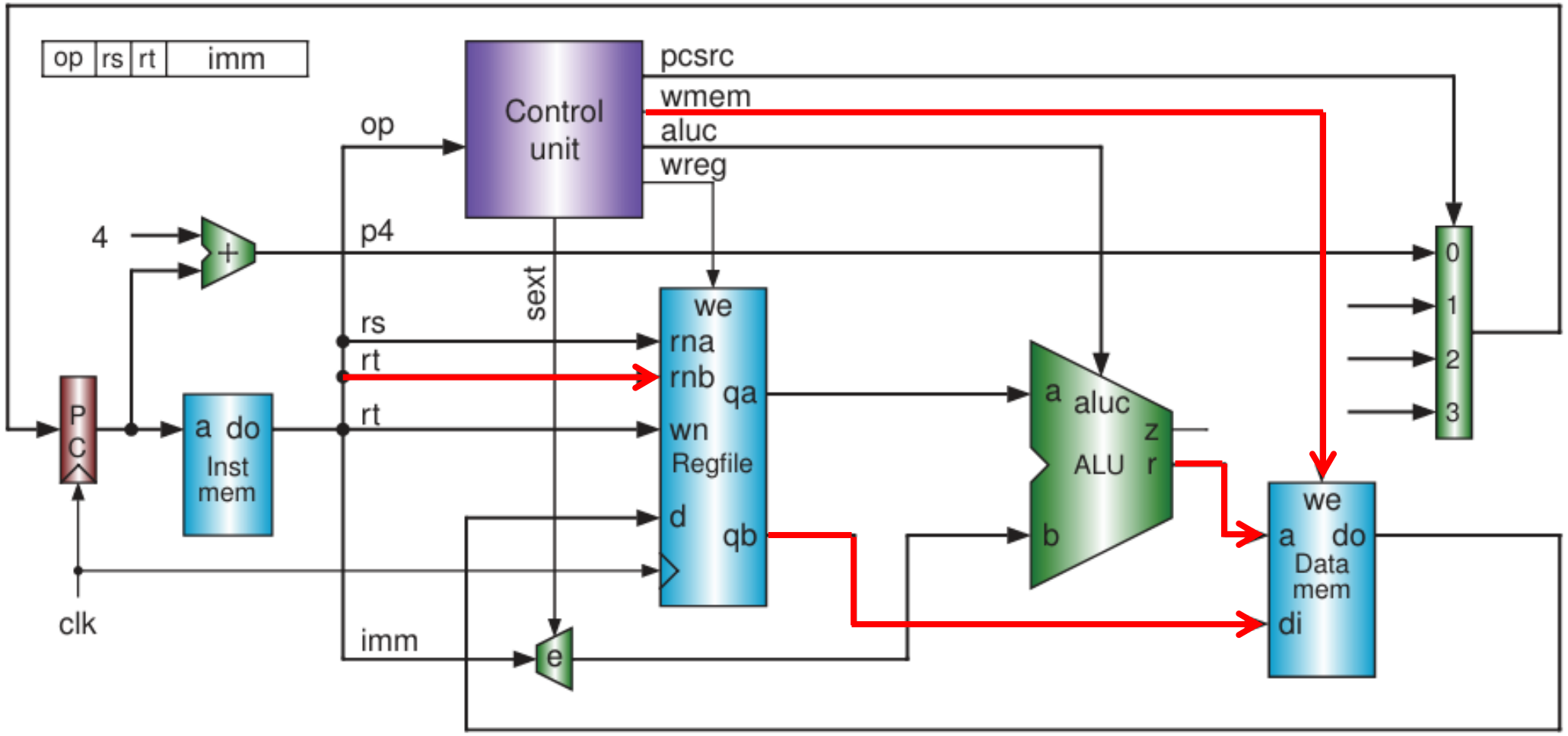
Single-Cycle CPU Design

- *lw*



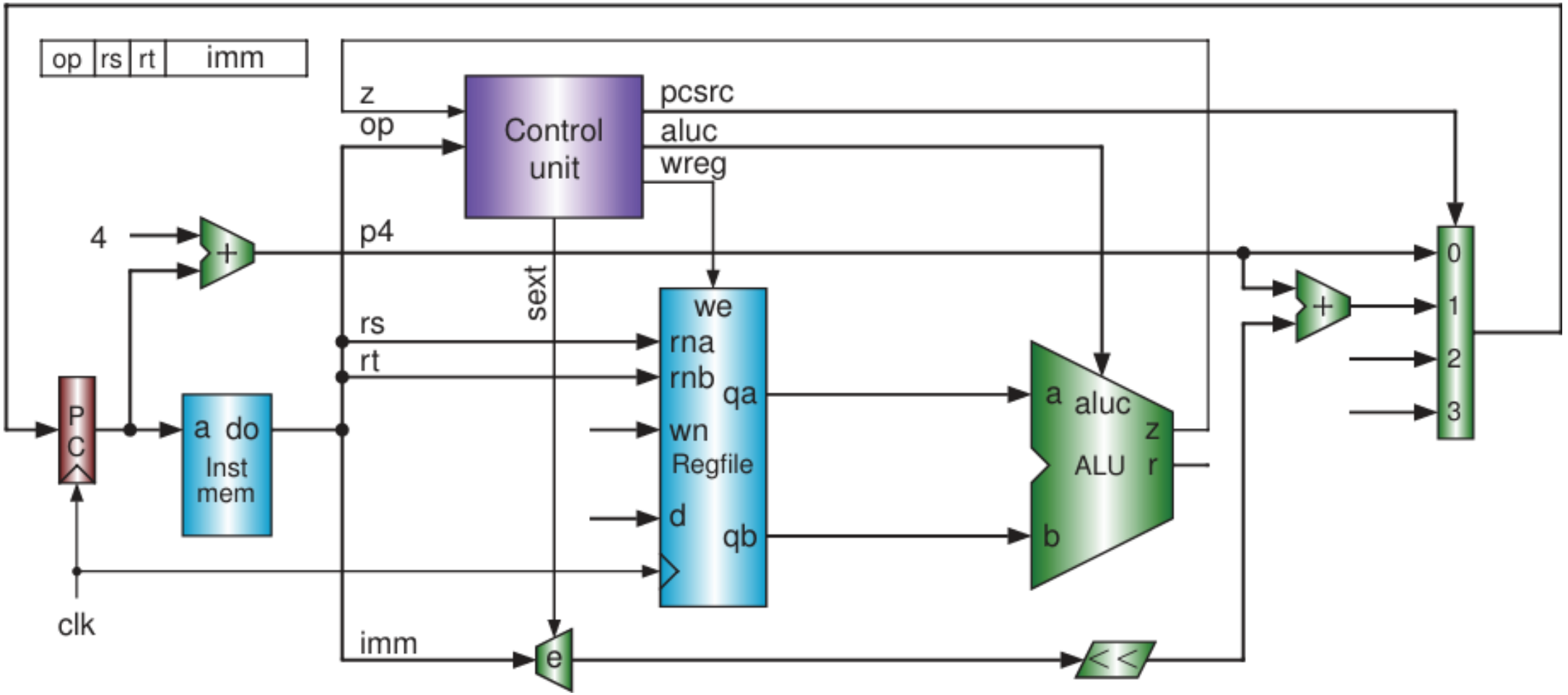
Single-Cycle CPU Design

- *SW*



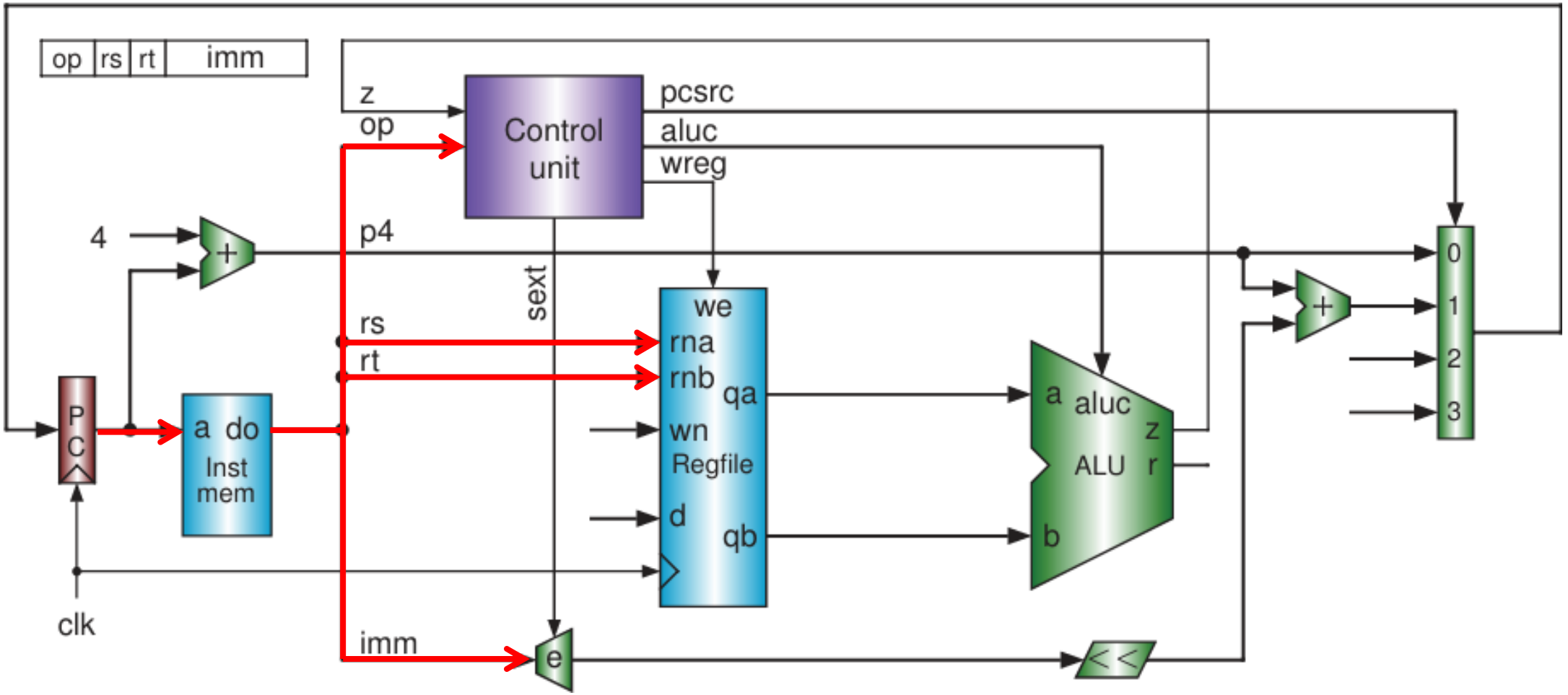
Single-Cycle CPU Design

- The circuits required by conditional branch instructions.



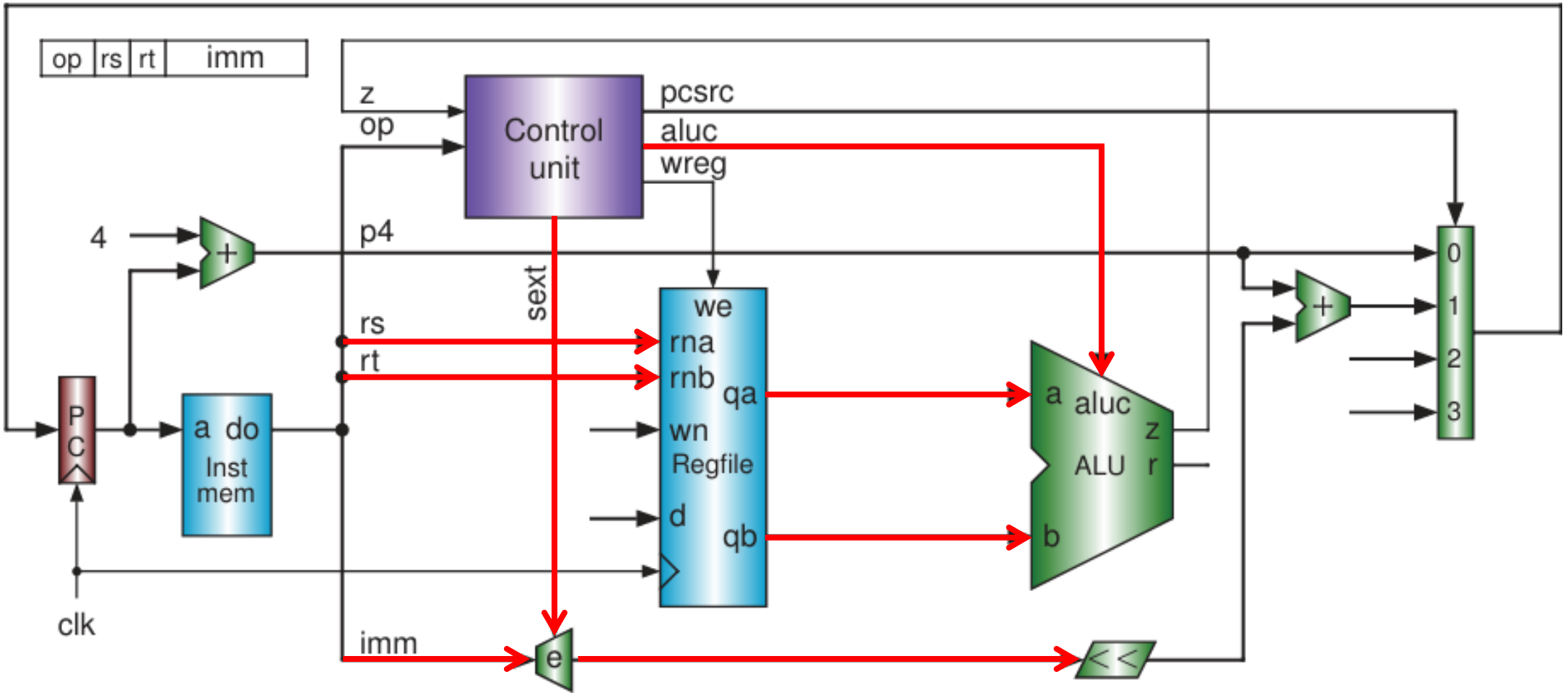
Single-Cycle CPU Design

- *beq/bne* (instruction fetch)



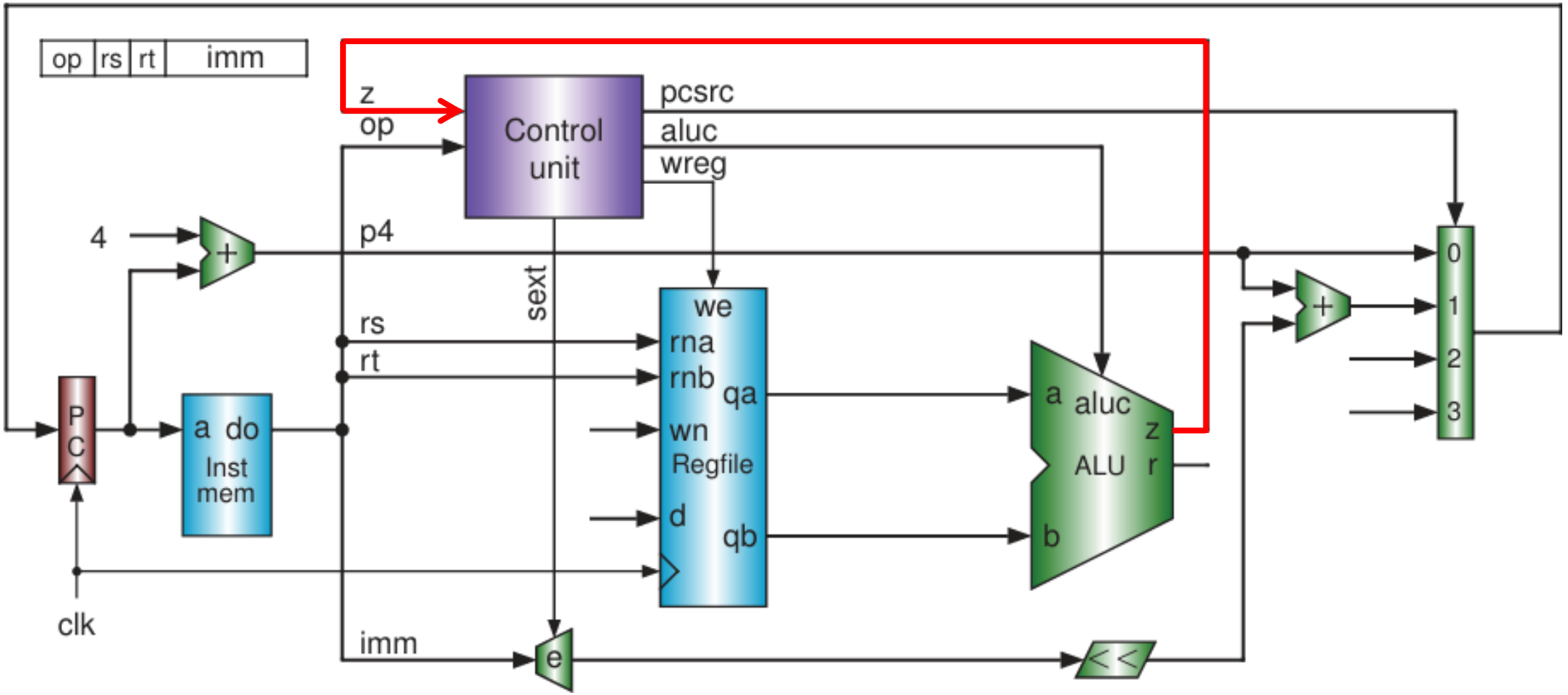
Single-Cycle CPU Design

- *beq/bne* (instruction execute)



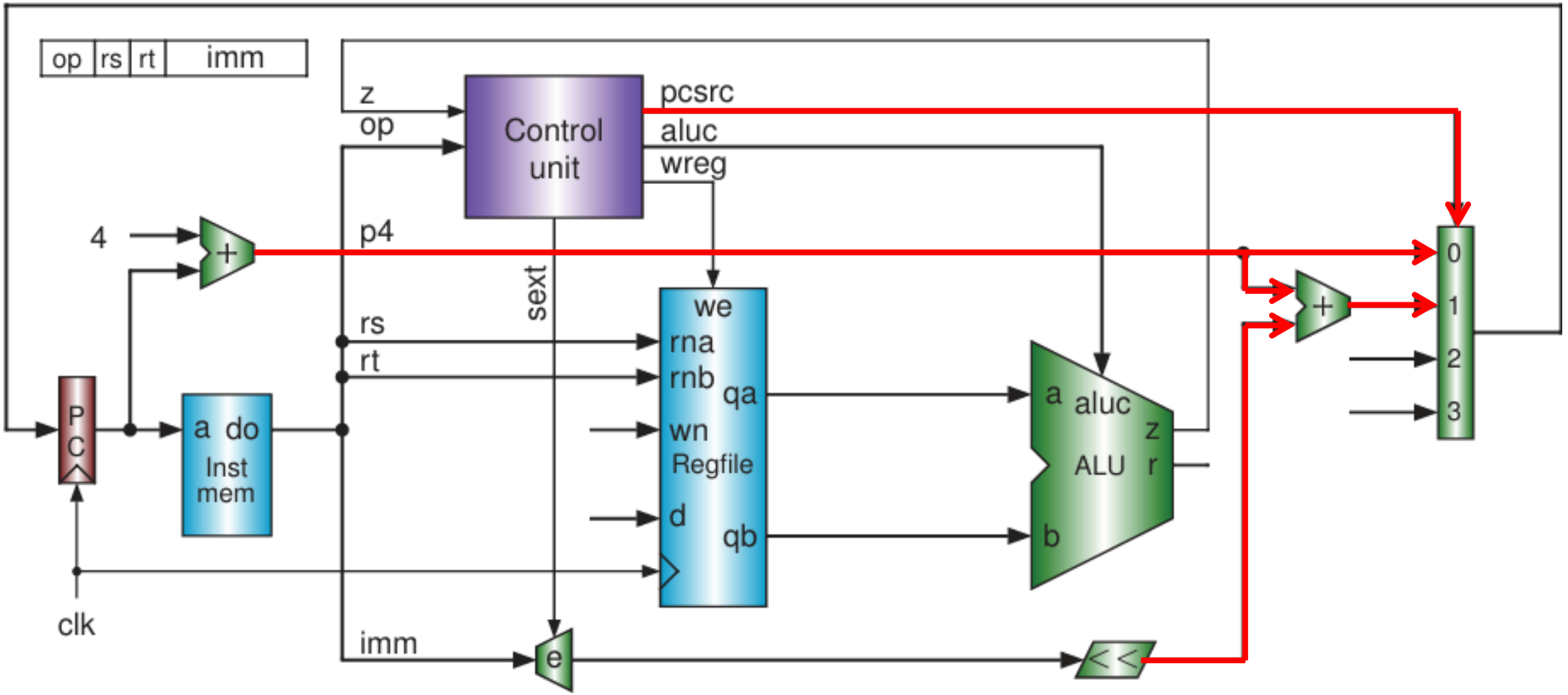
Single-Cycle CPU Design

- *beq/bne* (comparison result z)



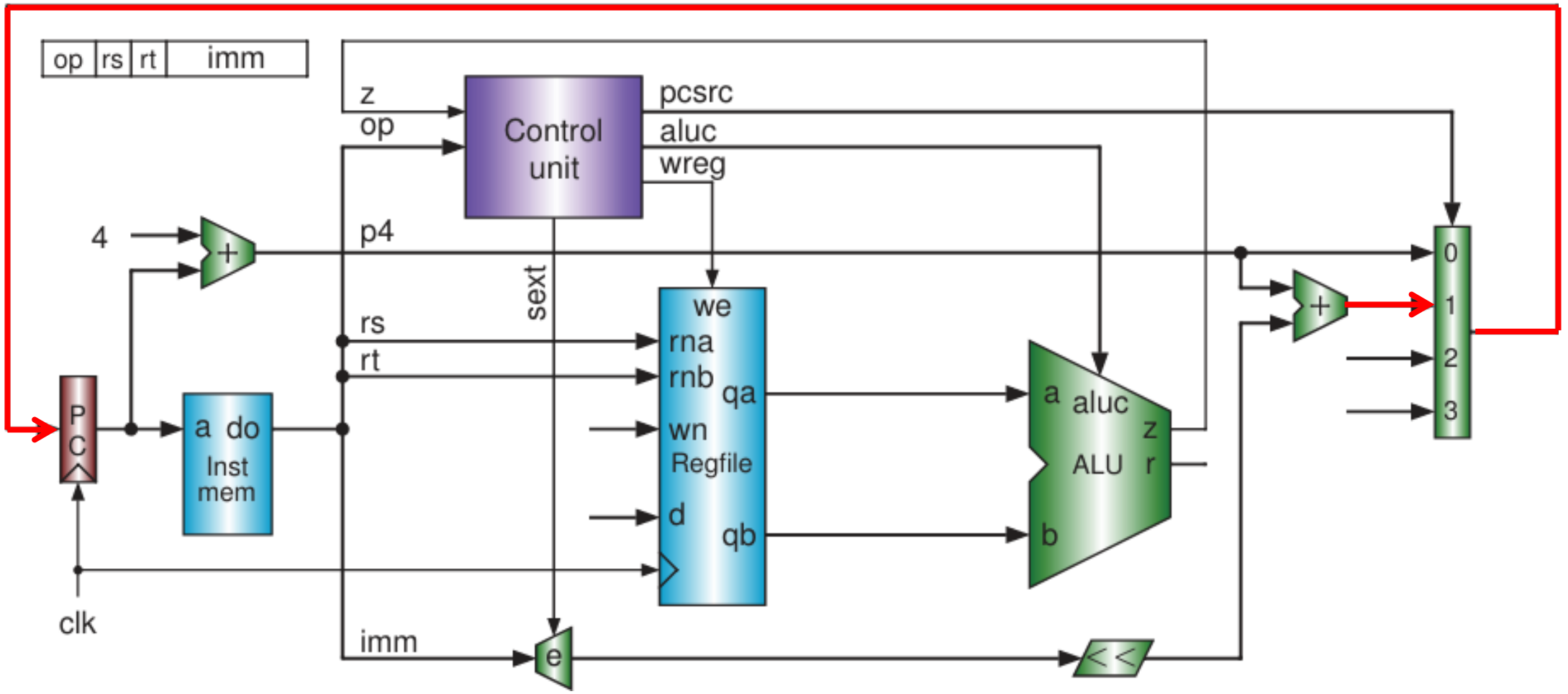
Single-Cycle CPU Design

- *beq/bne* (branch target address)



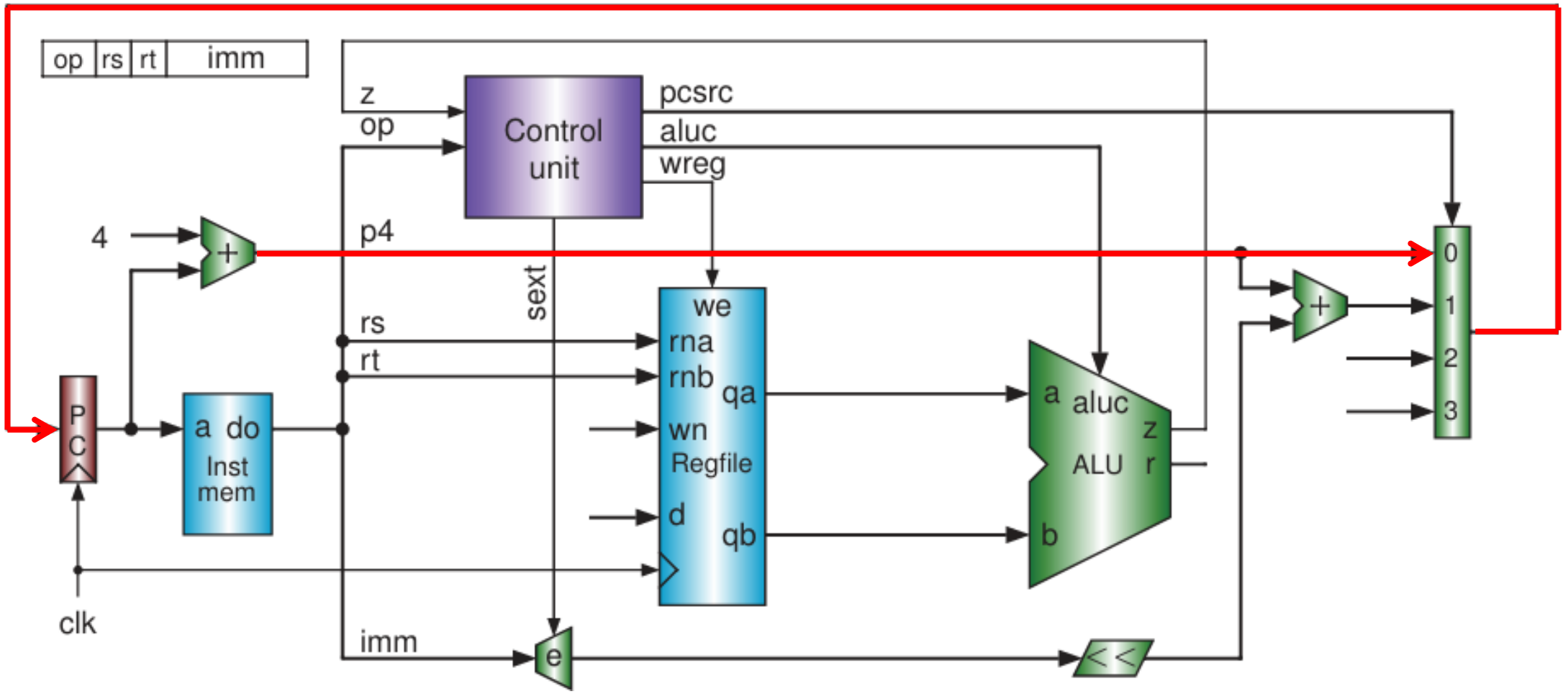
Single-Cycle CPU Design

- *beq/bne* (branch)



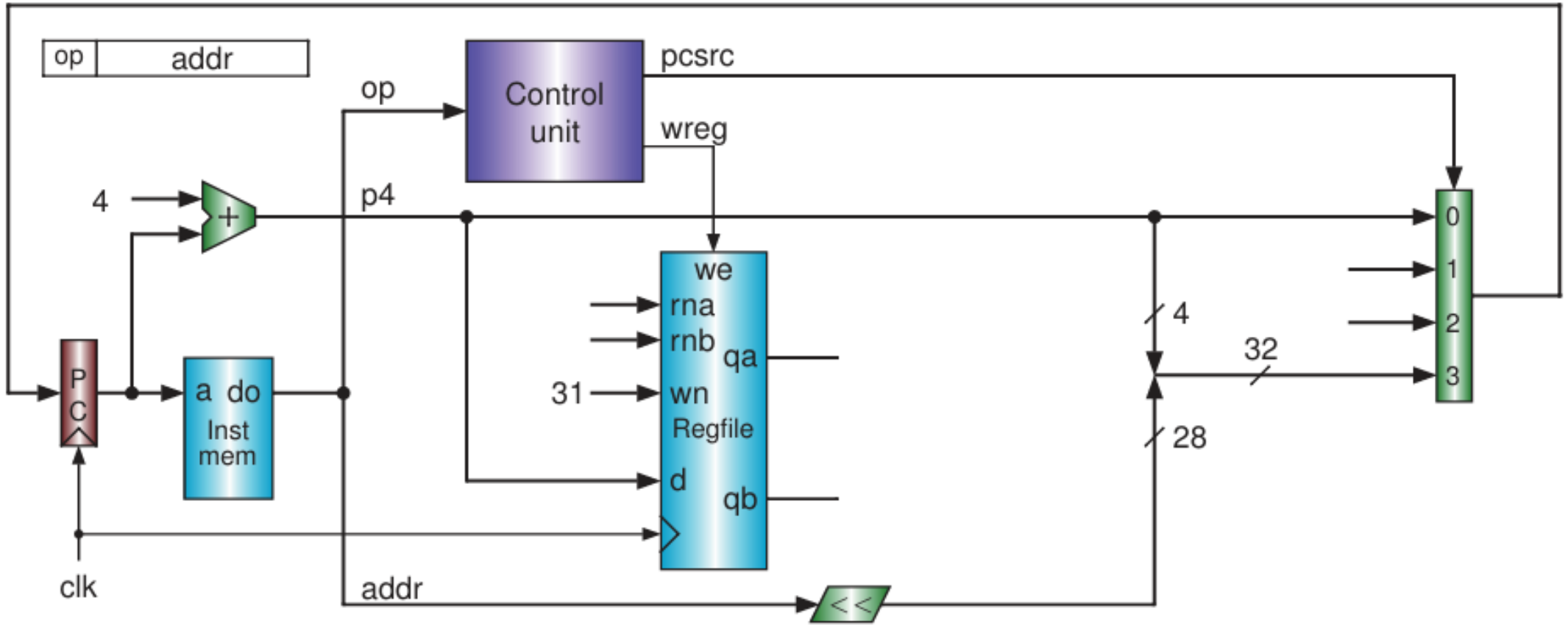
Single-Cycle CPU Design

- *beq/bne* (branch X)



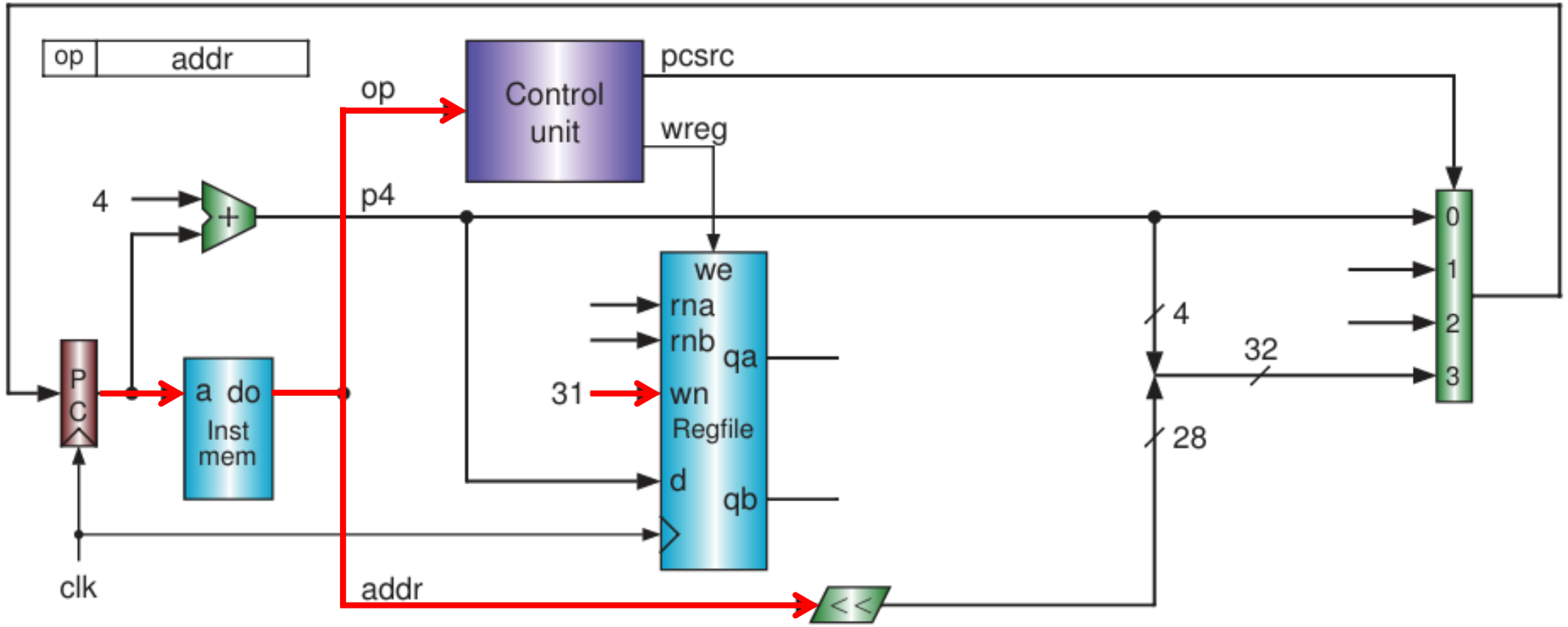
Single-Cycle CPU Design

- The circuits required by J-format instructions.



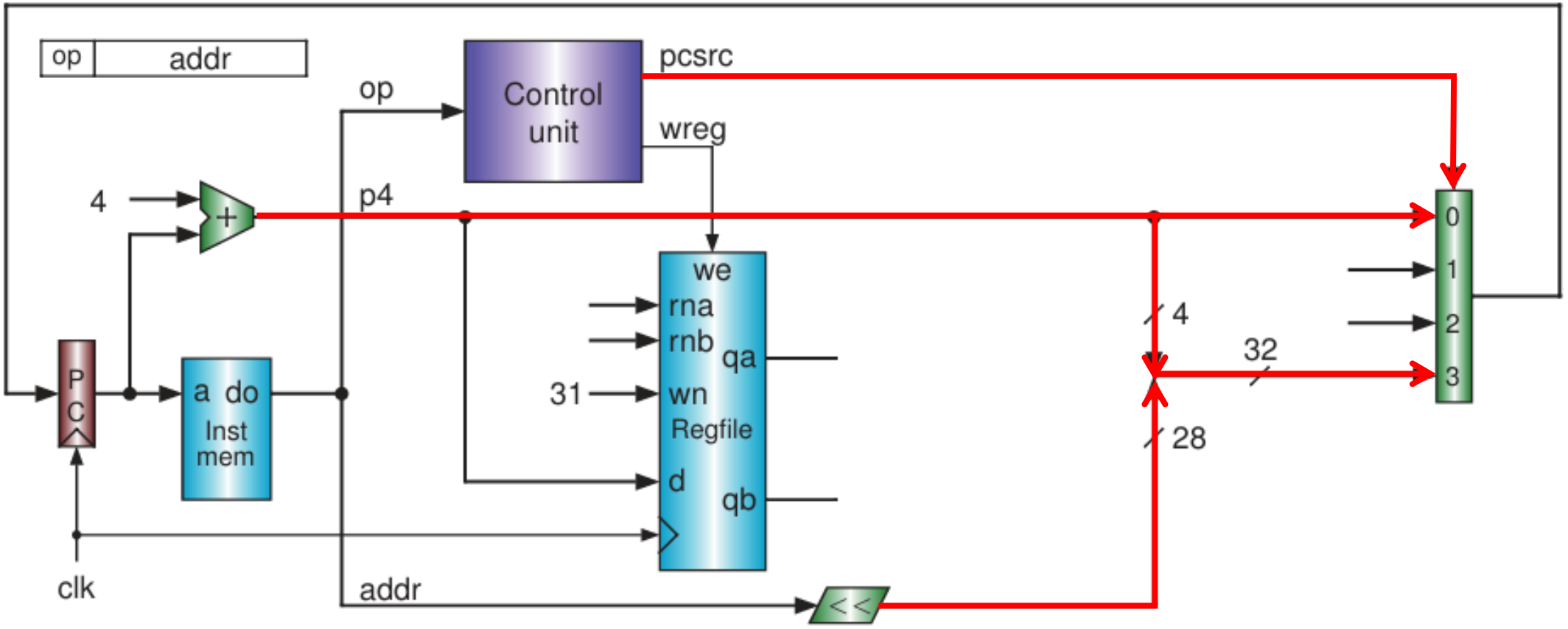
Single-Cycle CPU Design

- The circuits required by J-format *jal* instructions.



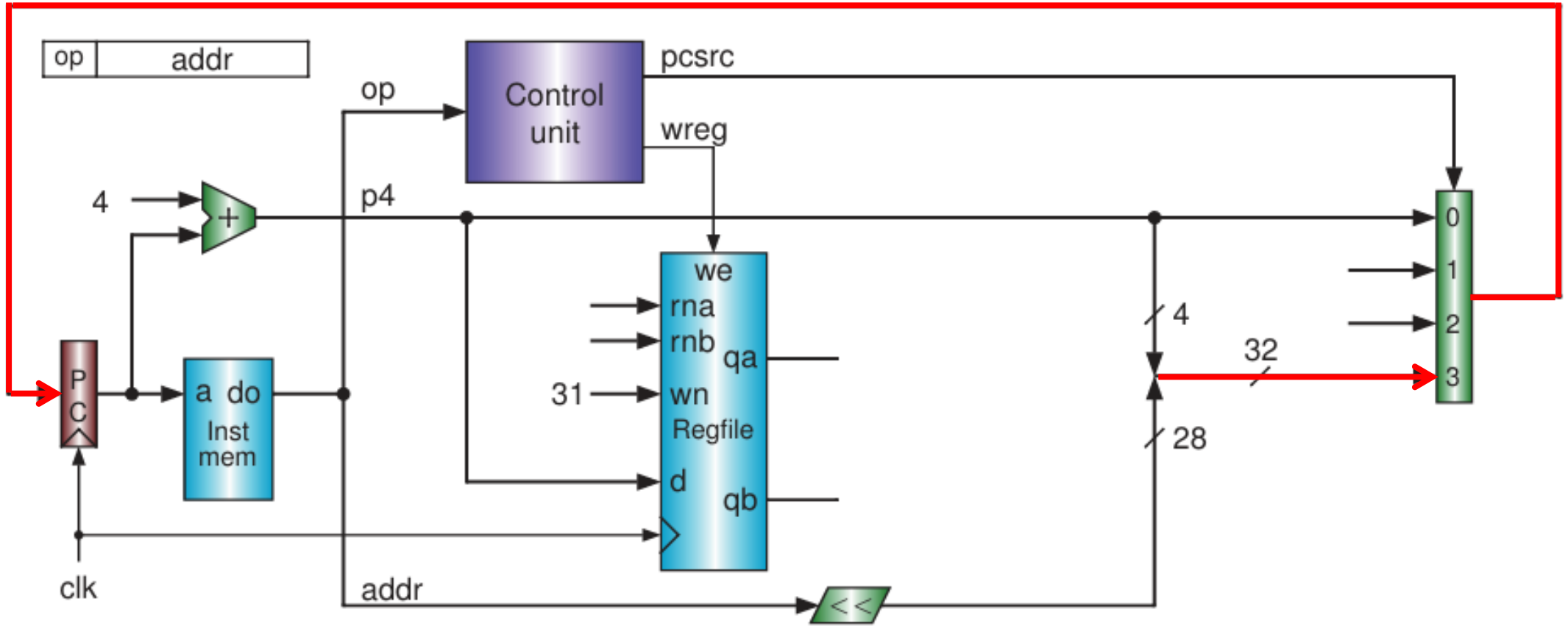
Single-Cycle CPU Design

- The circuits required by J-format *j/jal* instructions.



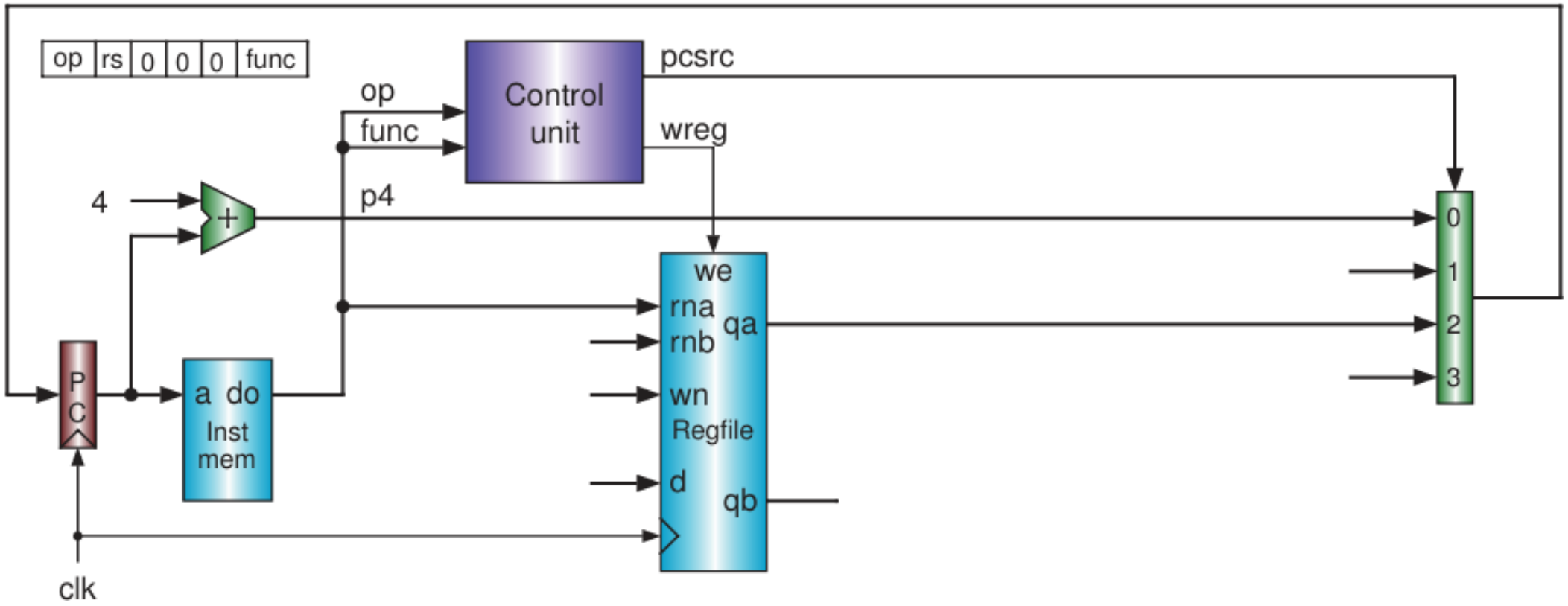
Single-Cycle CPU Design

- The circuits required by J-format *j/jal* instructions.



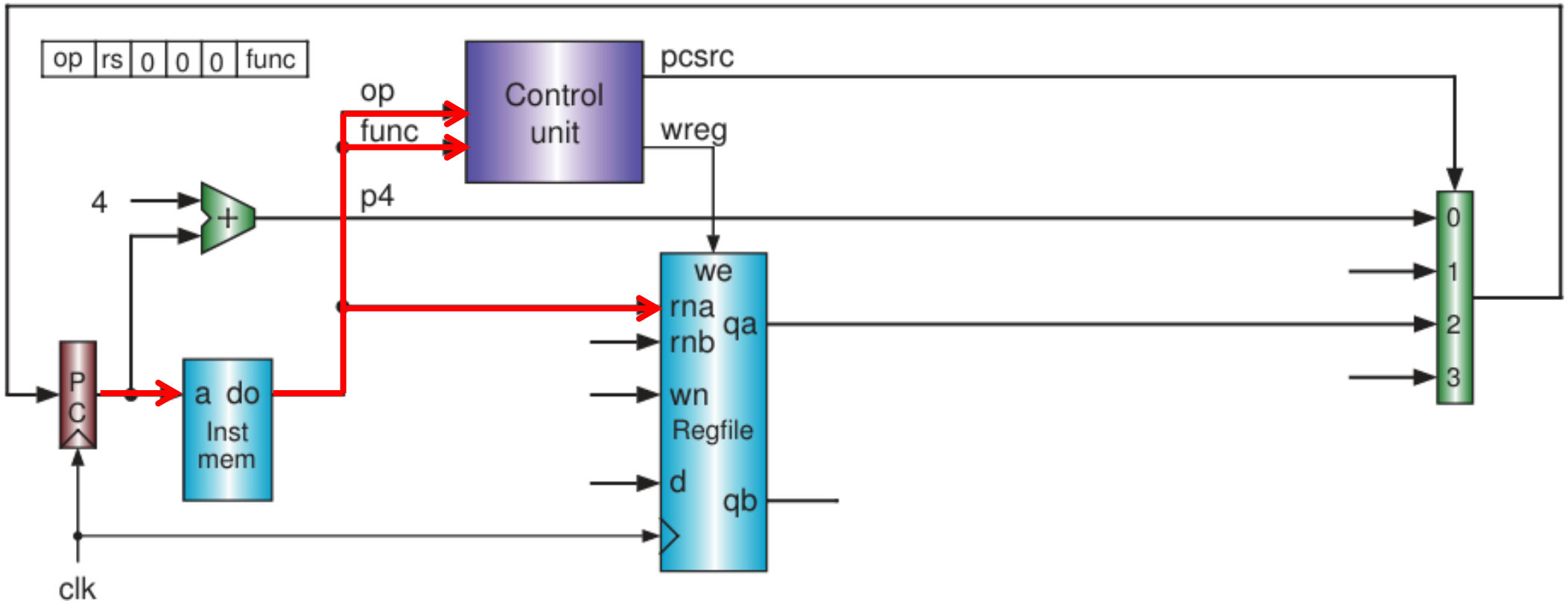
Single-Cycle CPU Design

- The circuits required by R-format *jump-register* instructions.



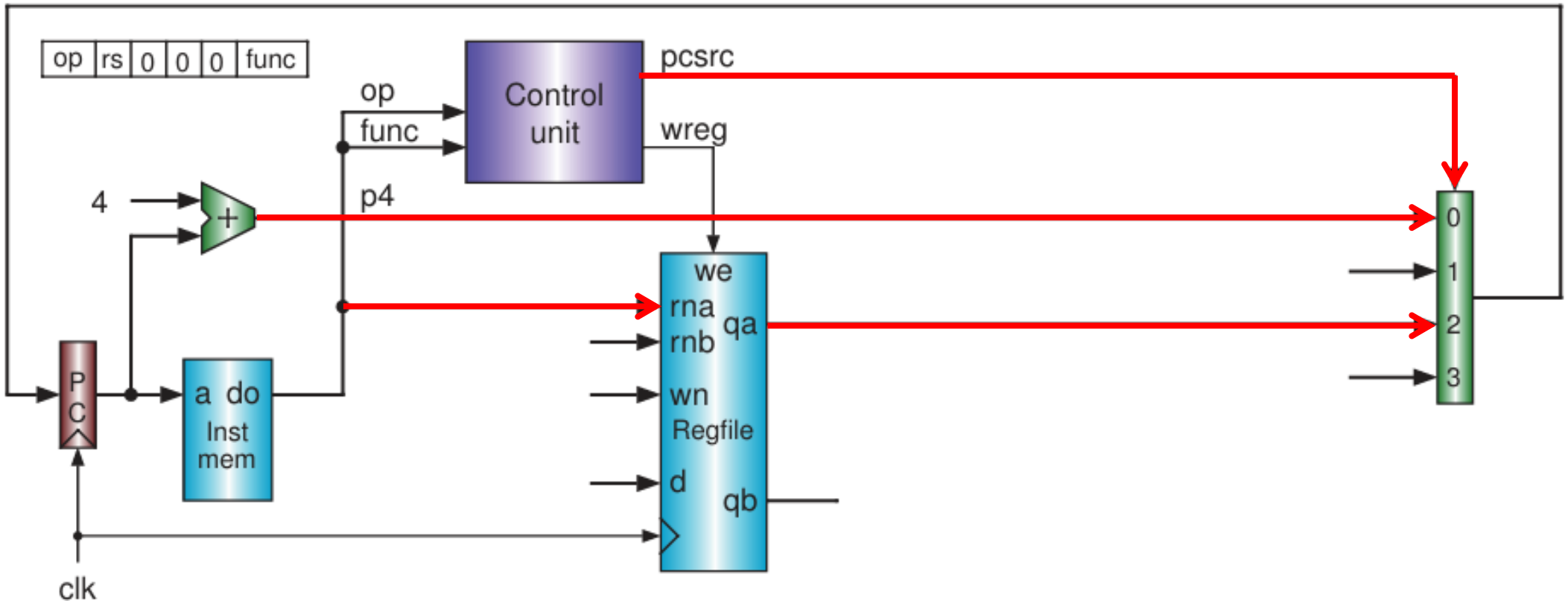
Single-Cycle CPU Design

- The circuits required by R-format *jump-register* instructions.



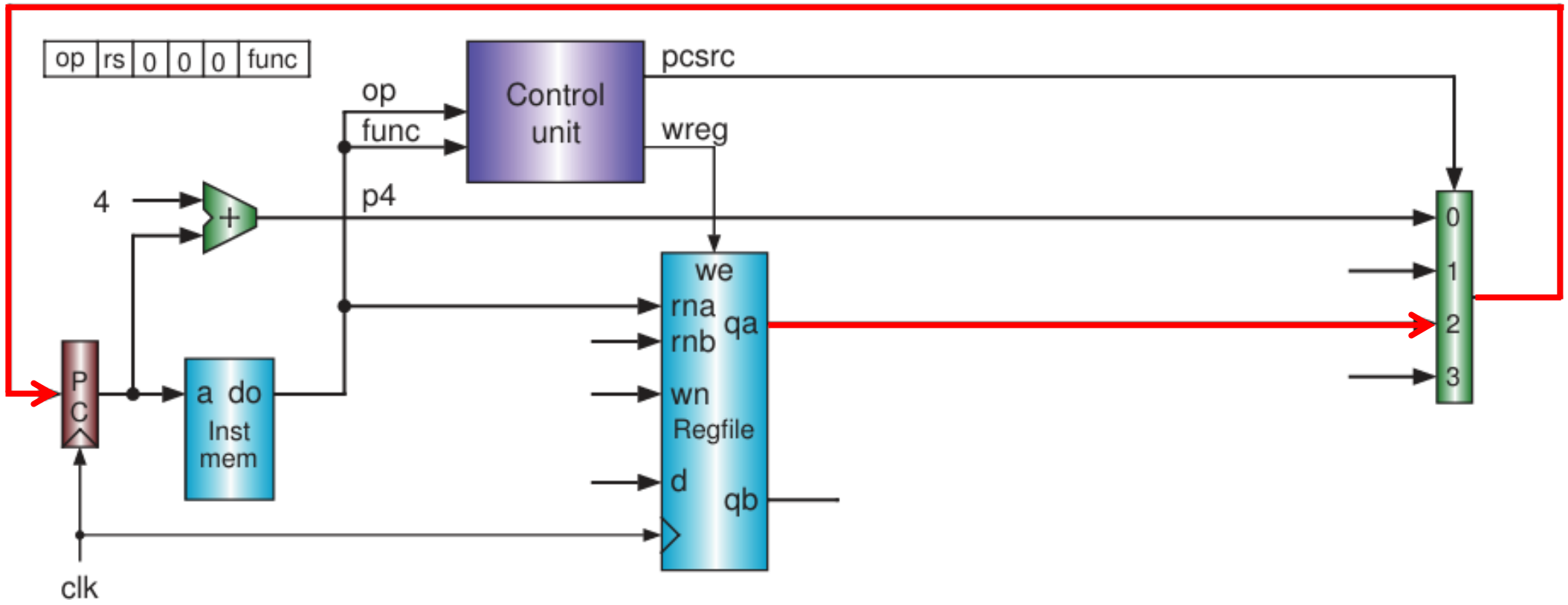
Single-Cycle CPU Design

- The circuits required by R-format *jump-register* instructions.

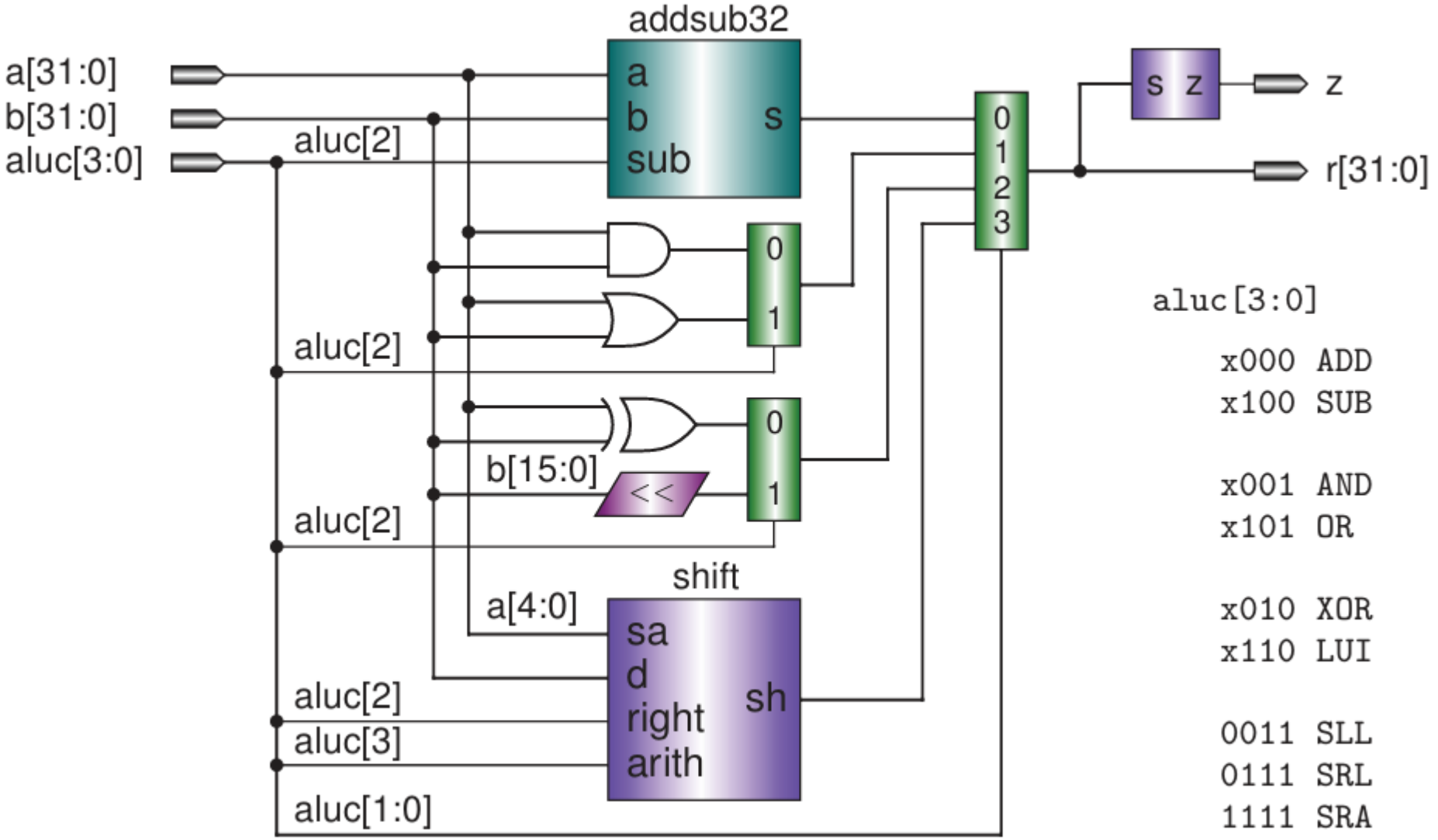


Single-Cycle CPU Design

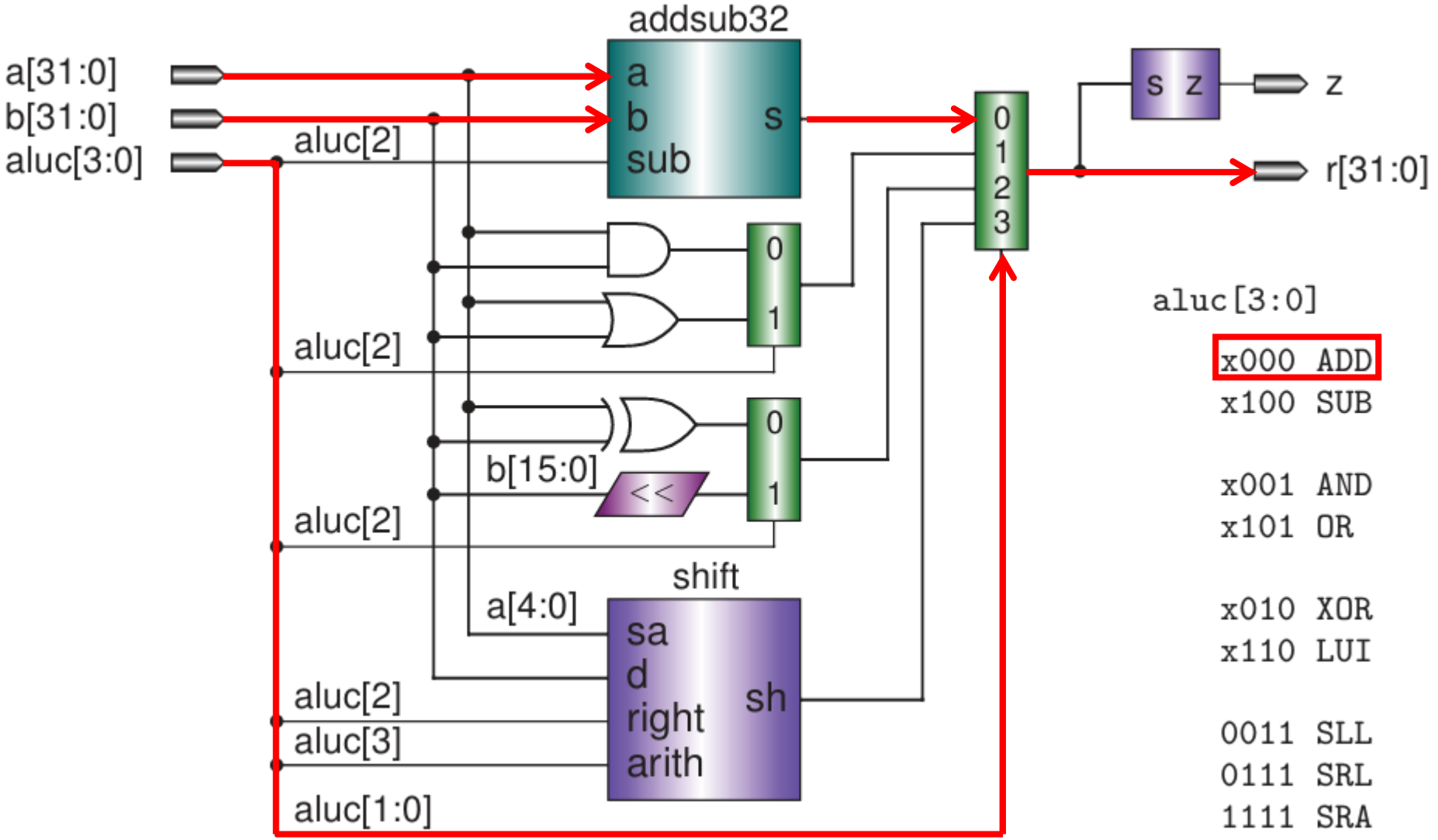
- The circuits required by R-format *jump-register* instructions.



ALU

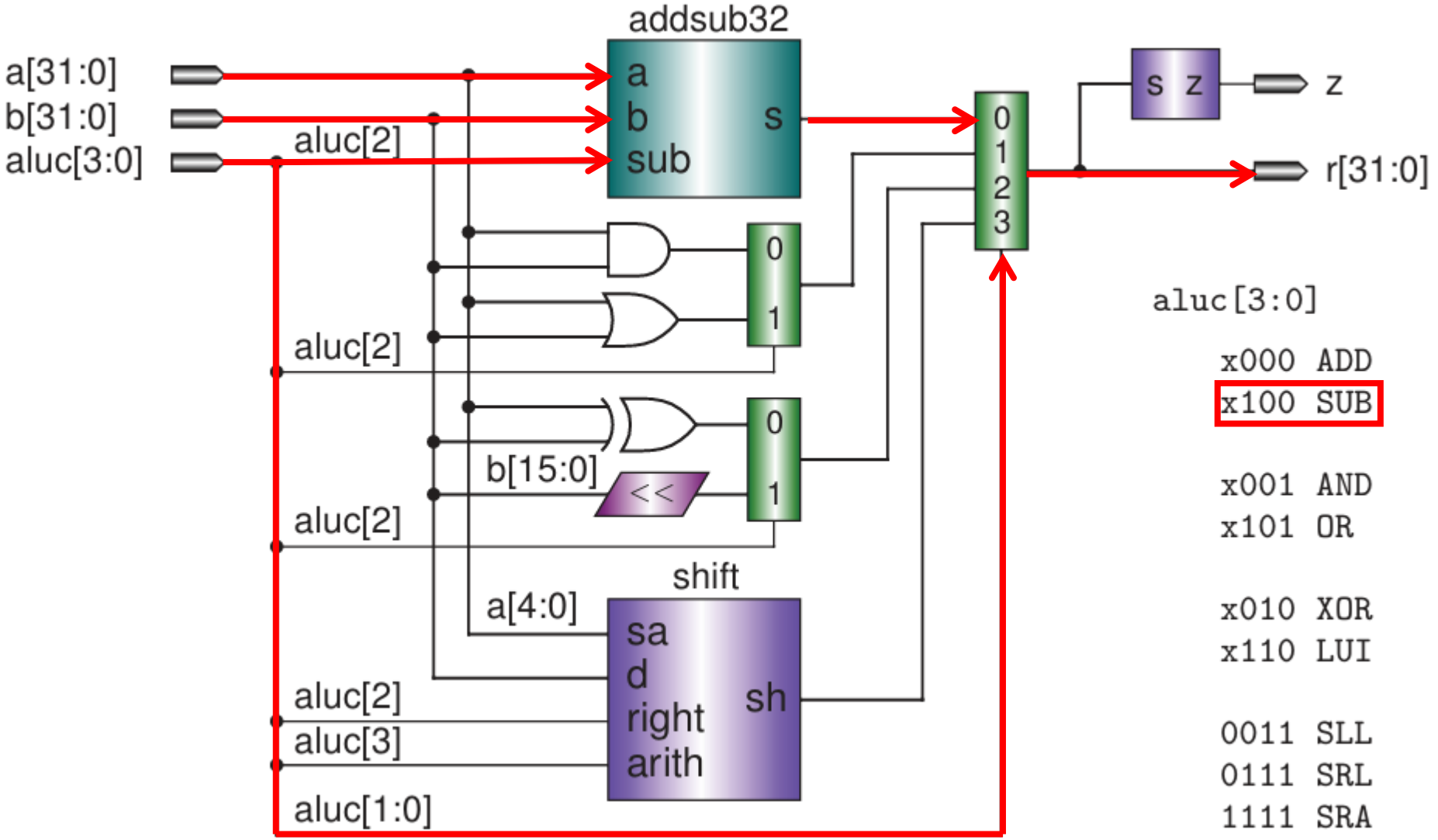

addsub32
a[31:0]
b[31:0]
aluc[3:0]
aluc[2]
a
b
sub
s
0
1
2
3
s z
z
r[31:0]
aluc[2]
0
1
b[15:0]
<<
aluc[2]
shift
a[4:0]
sa
d
right
arith
sh
aluc[2]
aluc[3]
aluc[1:0]
aluc[3:0]
x000 ADD
x100 SUB
x001 AND
x101 OR
x010 XOR
x110 LUI
0011 SLL
0111 SRL
1111 SRA

ALU



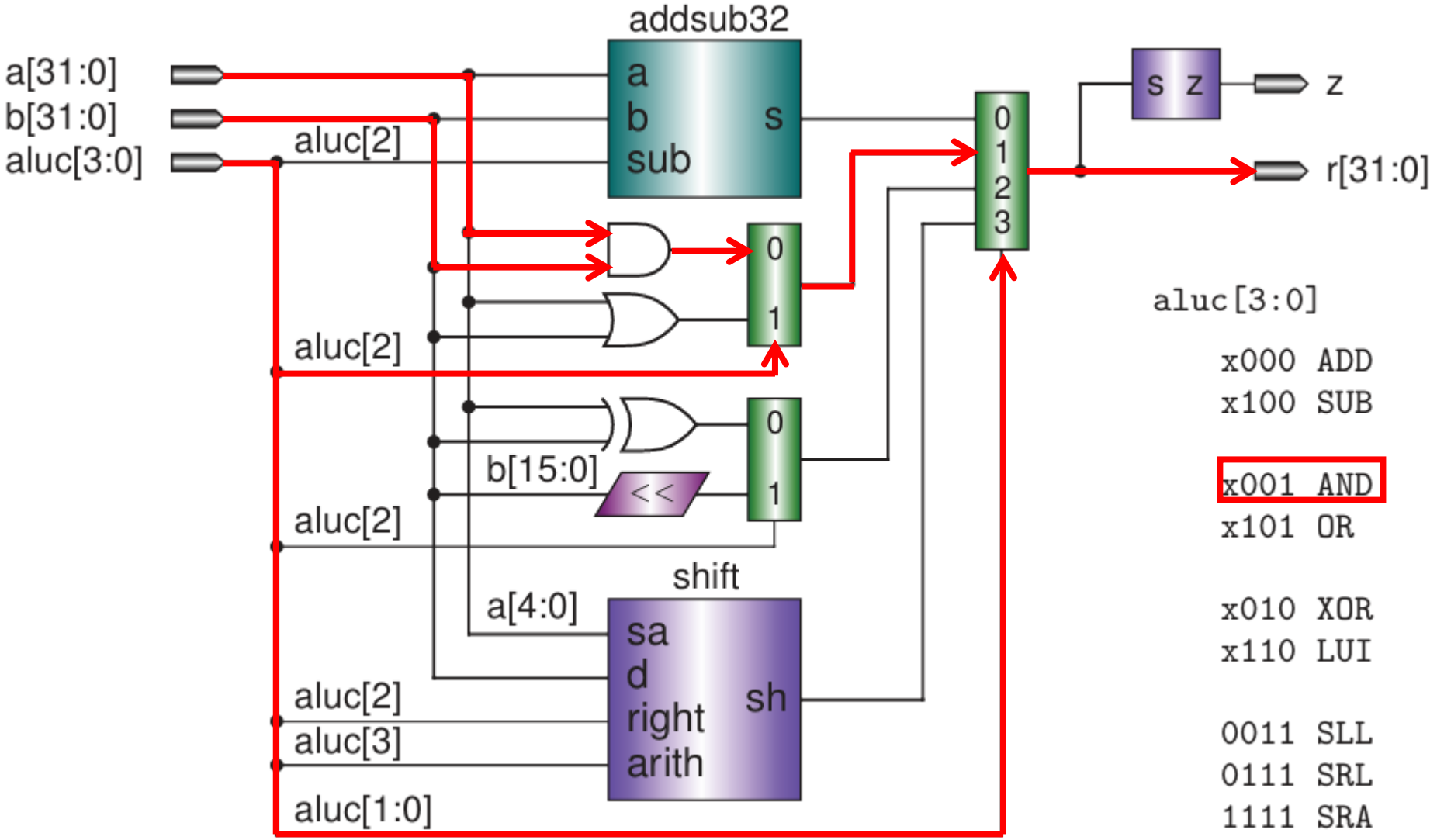

```
aluc[3:0]
    x000 ADD
    x100 SUB

    x001 AND
    x101 OR

    x010 XOR
    x110 LUI

    0011 SLL
    0111 SRL
    1111 SRA
```

ALU

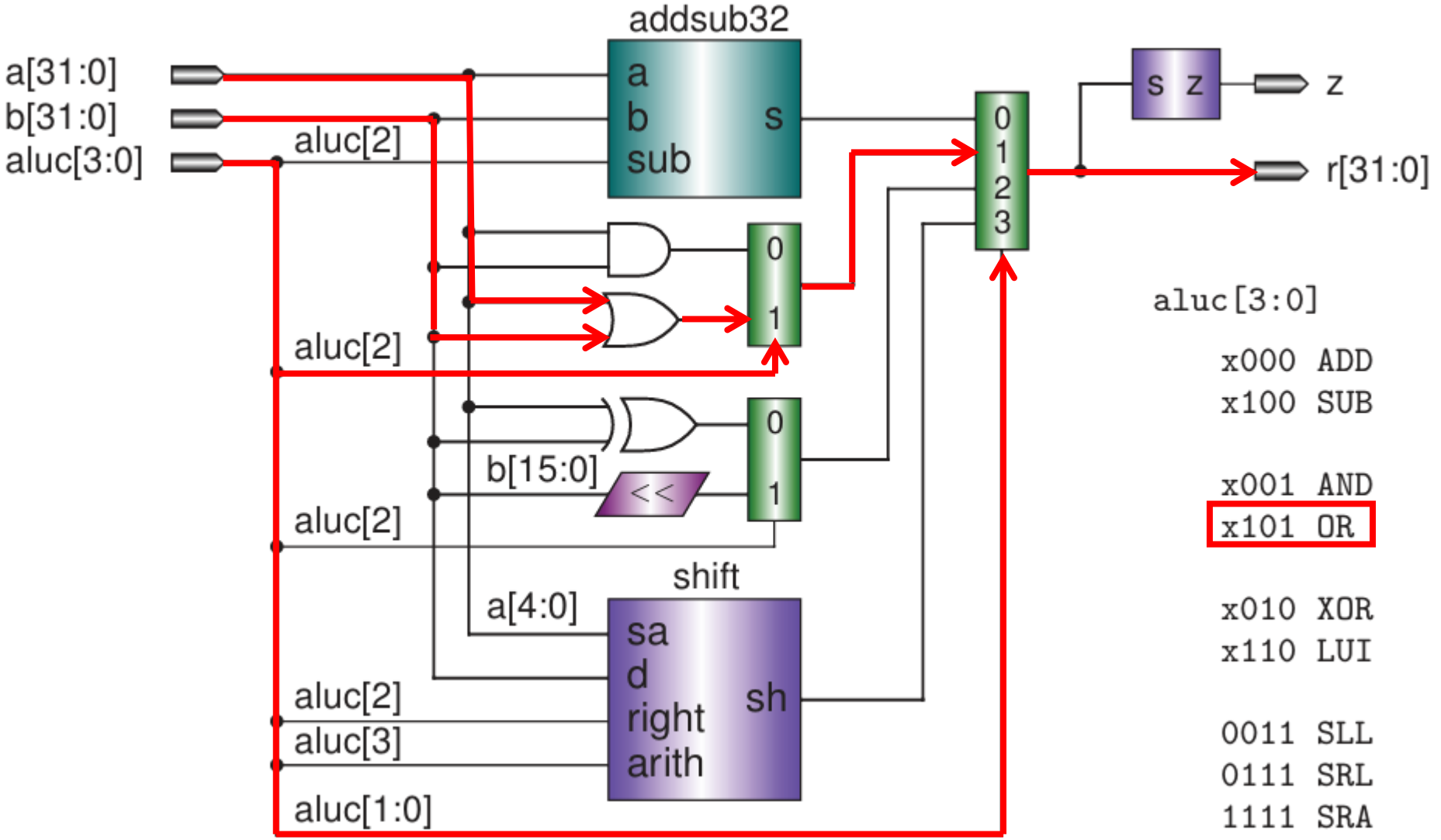



```
aluc[3:0]

    x000 ADD
    x100 SUB

    x001 AND
    x101 OR

    x010 XOR
    x110 LUI

    0011 SLL
    0111 SRL
    1111 SRA
```

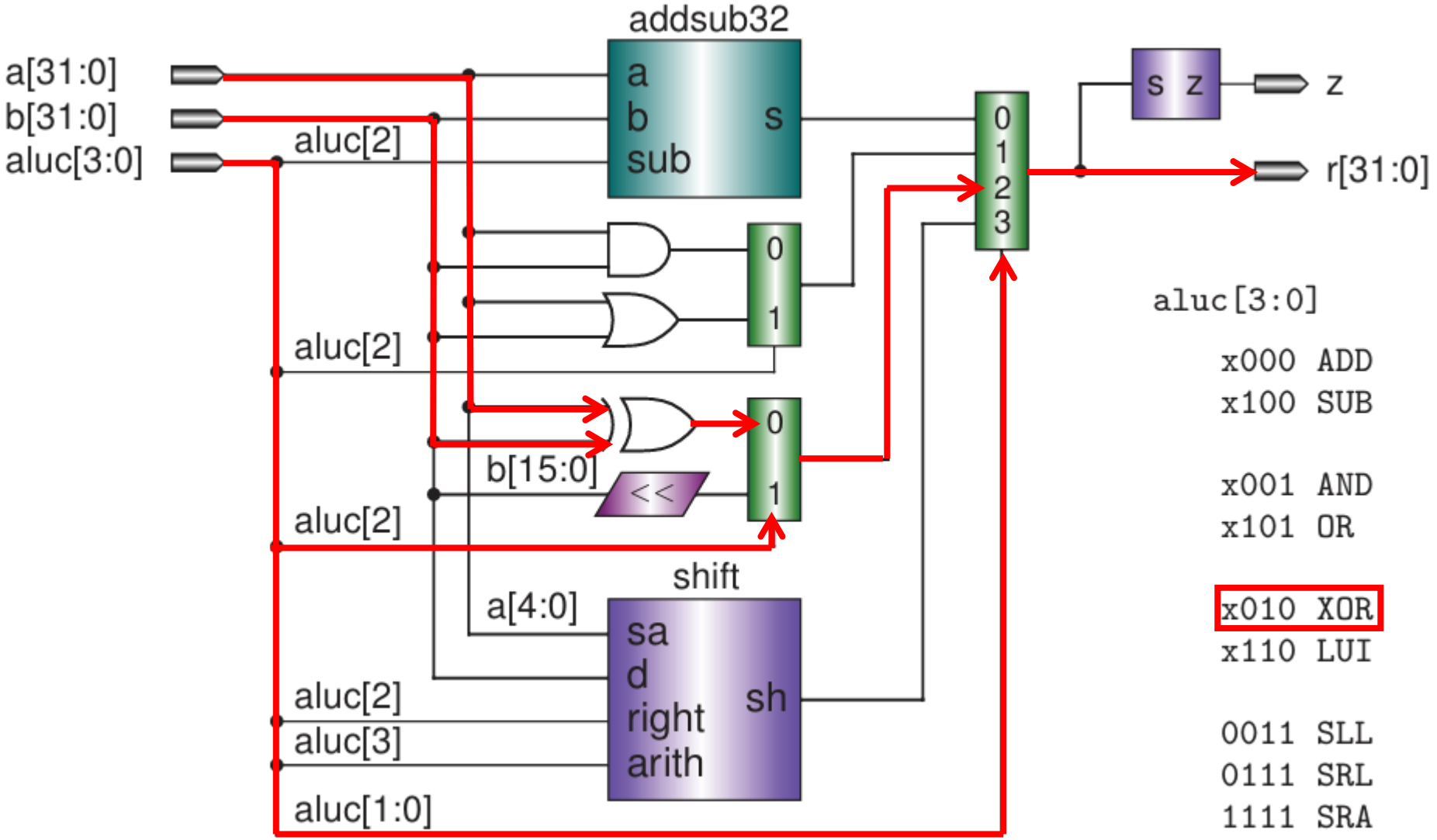
ALU


addsub32
a[31:0]
b[31:0]
aluc[3:0]
aluc[2]
a
b
sub
s
0
1
2
3
s z
z
r[31:0]
aluc[2]
0
1
b[15:0]
<<
aluc[2]
shift
a[4:0]
sa
d
right
arith
sh
aluc[2]
aluc[3]
aluc[1:0]
aluc[3:0]
x000 ADD
x100 SUB
x001 AND
x101 OR
x010 XOR
x110 LUI
0011 SLL
0111 SRL
1111 SRA

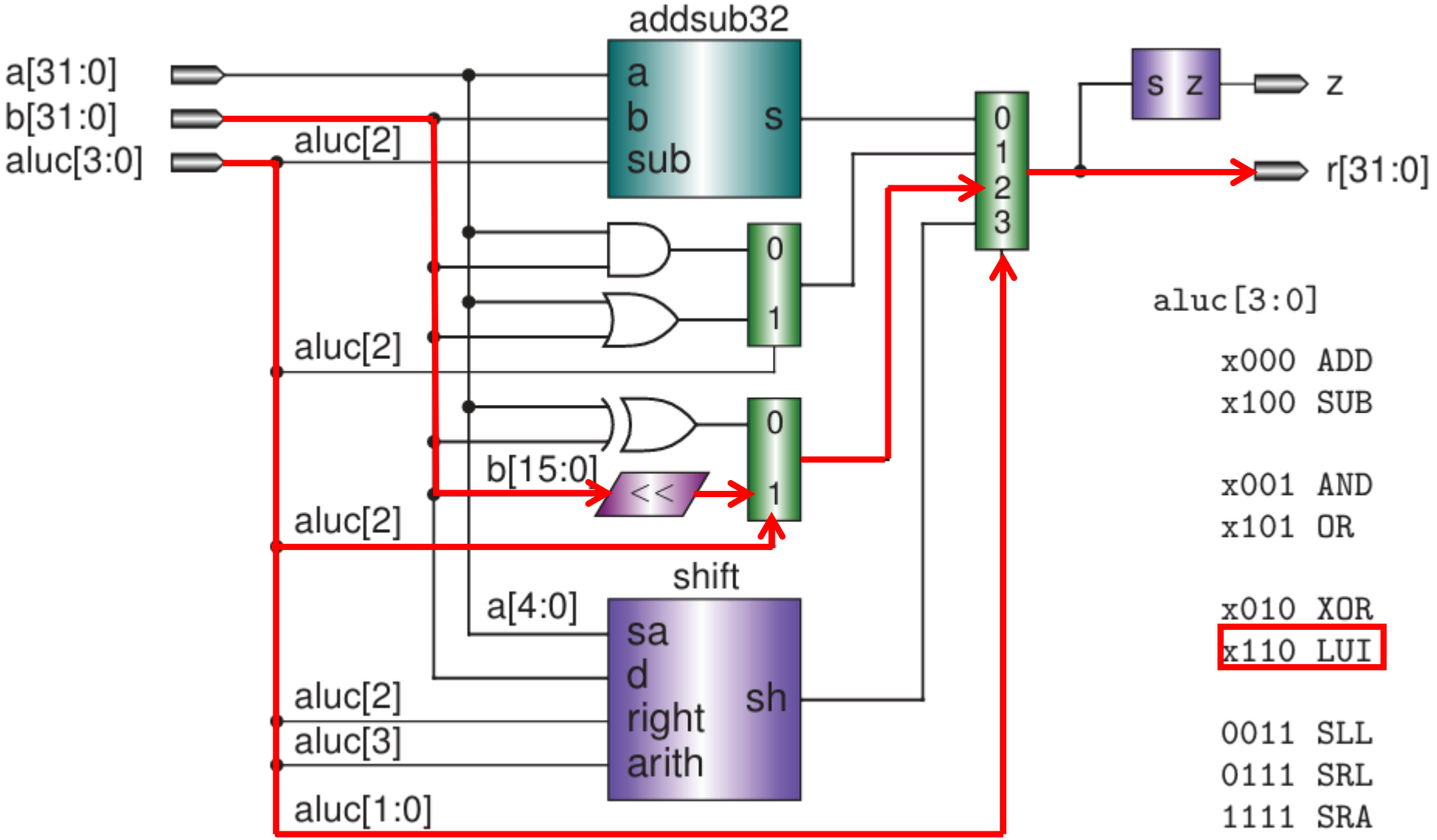
ALU


addsub32
a[31:0]
b[31:0]
aluc[3:0]
aluc[2]
a
b
sub
s
0
1
2
3
s z
z
r[31:0]
aluc[2]
0
1
b[15:0]
<<
aluc[2]
shift
a[4:0]
sa
d
right
arith
sh
aluc[2]
aluc[3]
aluc[1:0]
aluc[3:0]
x000 ADD
x100 SUB
x001 AND
x101 OR
x010 XOR
x110 LUI
0011 SLL
0111 SRL
1111 SRA

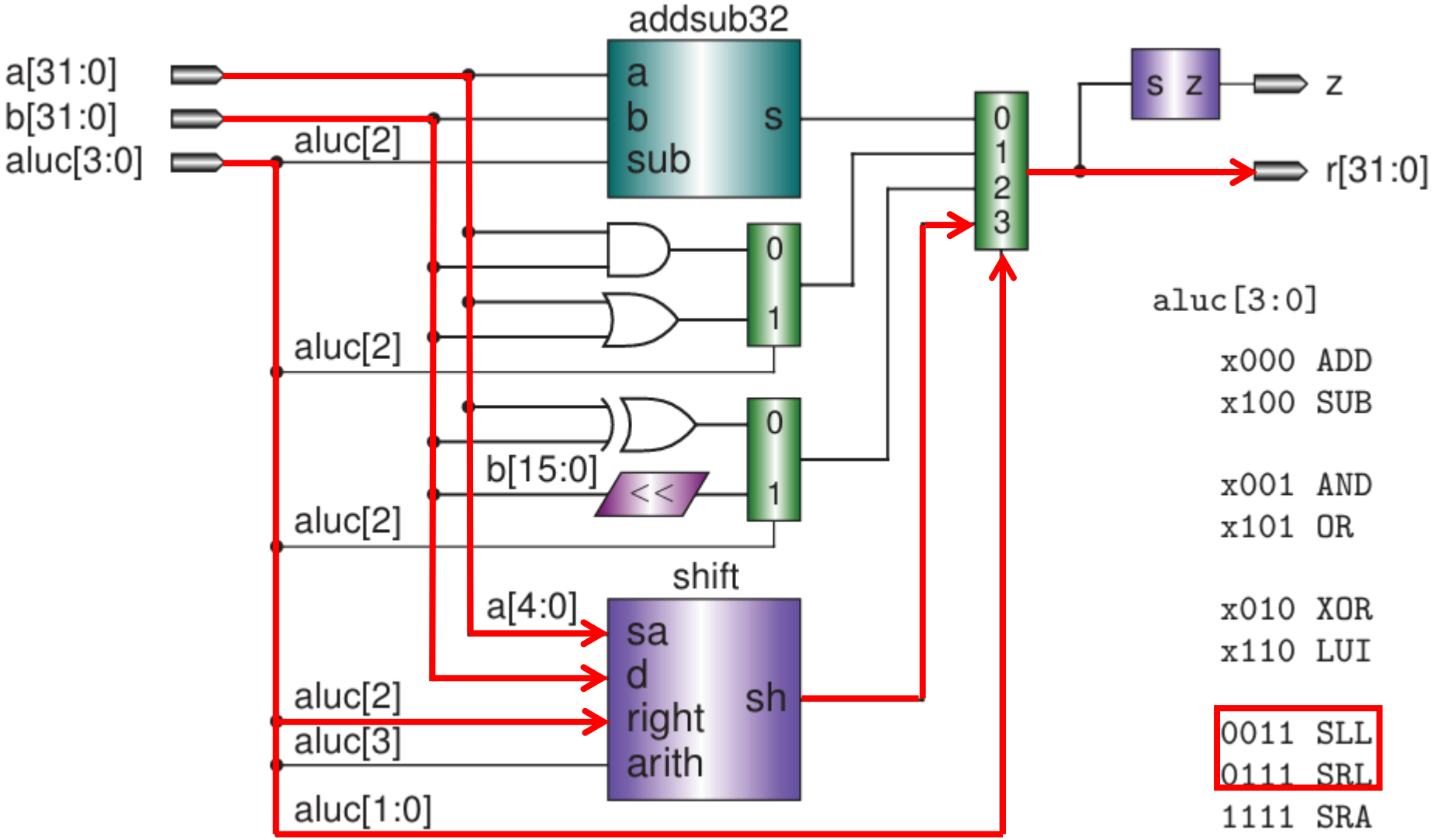
ALU


addsub32
a[31:0]
b[31:0]
aluc[3:0]
aluc[2]
a
b
sub
s
0
1
2
3
s z
z
r[31:0]
aluc[2]
0
1
b[15:0]
<<
aluc[2]
shift
a[4:0]
sa
d
right
arith
sh
aluc[2]
aluc[3]
aluc[1:0]
aluc[3:0]
x000 ADD
x100 SUB
x001 AND
x101 OR
x010 XOR
x110 LUI
0011 SLL
0111 SRL
1111 SRA

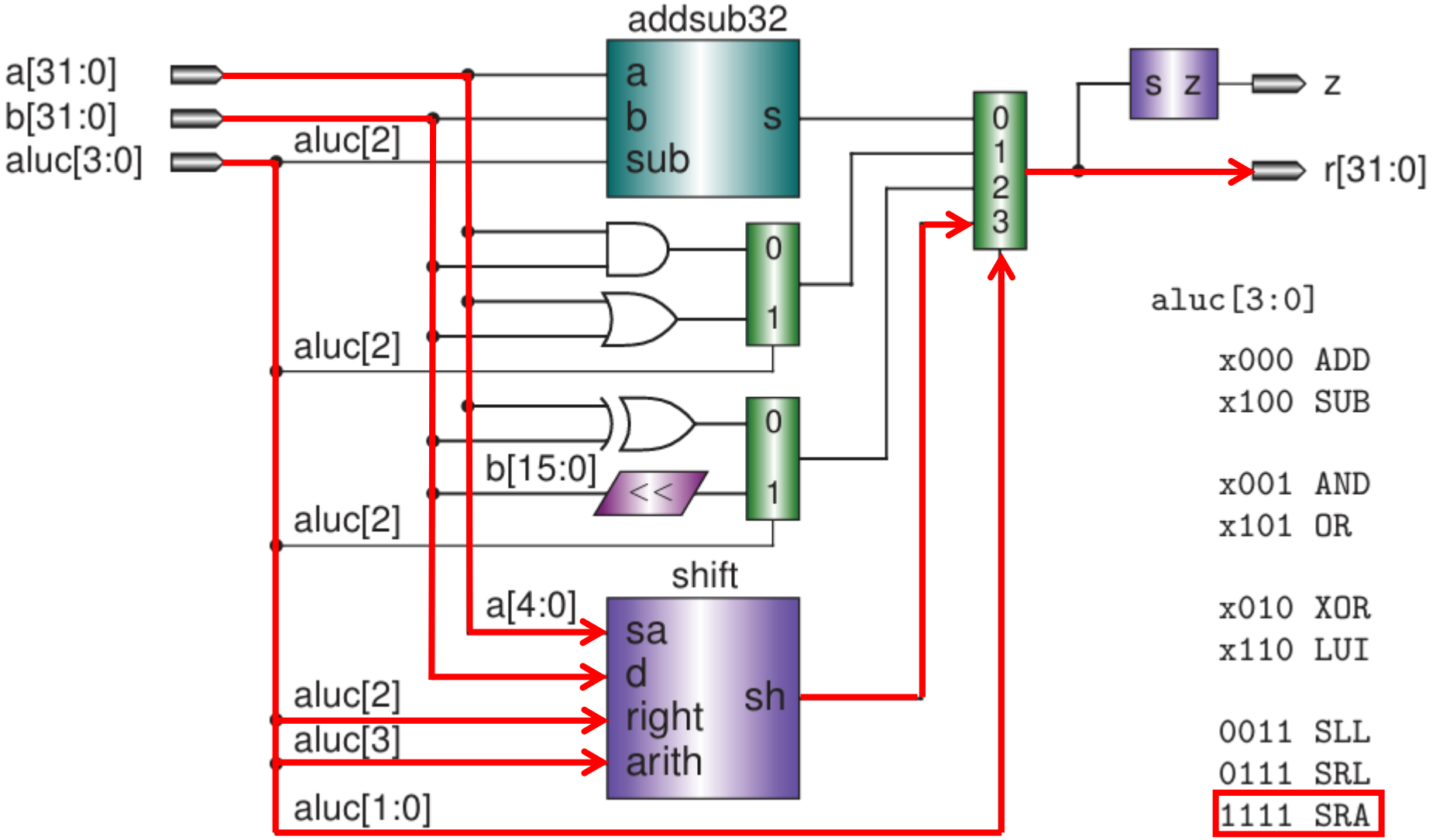
ALU


addsub32
a[31:0]
b[31:0]
aluc[3:0]
aluc[2]
a
b
sub
s
0
1
2
3
s z
z
r[31:0]
aluc[2]
0
1
b[15:0]
<<
aluc[2]
0
1
shift
a[4:0]
sa
d
right
arith
sh
aluc[2]
aluc[3]
aluc[1:0]
aluc[3:0]
x000 ADD
x100 SUB
x001 AND
x101 OR
x010 XOR
x110 LUI
0011 SLL
0111 SRL
1111 SRA

ALU

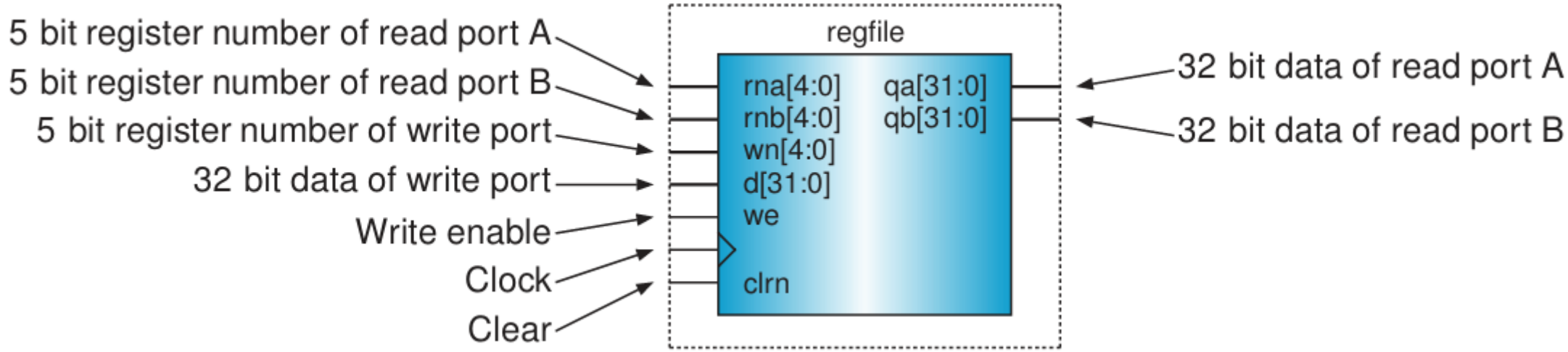

addsub32
a[31:0]
b[31:0]
aluc[3:0]
aluc[2]
a
b
sub
s
0
1
2
3
s z
z
r[31:0]
aluc[2]
0
1
b[15:0]
<<
aluc[2]
shift
a[4:0]
sa
d
right
arith
sh
aluc[2]
aluc[3]
aluc[1:0]
aluc[3:0]
x000 ADD
x100 SUB
x001 AND
x101 OR
x010 XOR
x110 LUI
0011 SLL
0111 SRL
1111 SRA

ALU


addsub32
a[31:0]
b[31:0]
aluc[3:0]
aluc[2]
a
b
sub
s
0
1
2
3
s z
z
r[31:0]
aluc[2]
0
1
b[15:0]
<<
aluc[2]
shift
a[4:0]
sa
d
right
arith
sh
aluc[2]
aluc[3]
aluc[1:0]
aluc[3:0]
x000 ADD
x100 SUB
x001 AND
x101 OR
x010 XOR
x110 LUI
0011 SLL
0111 SRL
1111 SRA

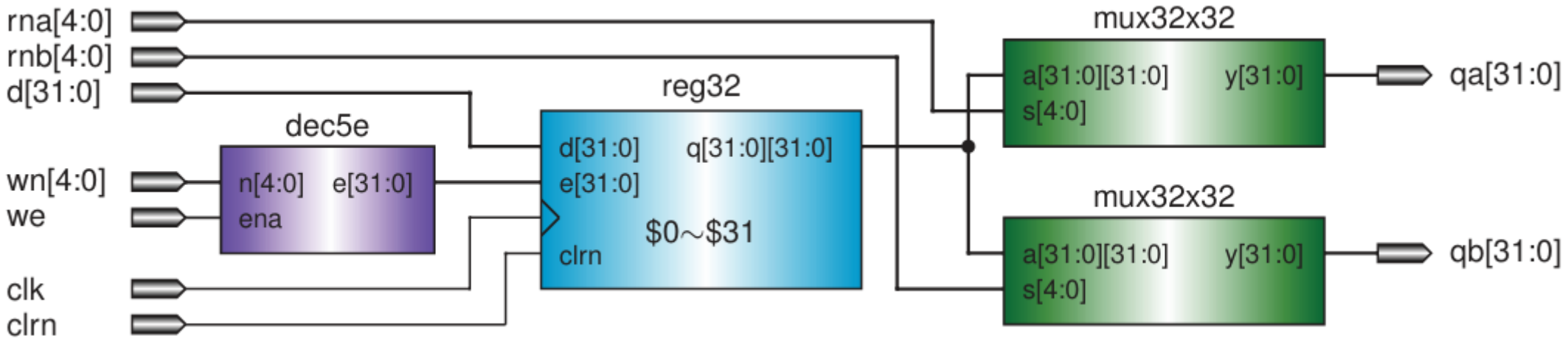
Register File

- The register file contains 32 general-purpose registers.
- Two read ports (port A and B) and write port.
- Each port has a 5-bit address input (register number).
- 32-bit data input and write enable are provided for write port.



Register File

- A register file (RF) has 32 bits, it can be designed with 32 D-FFs.
- Each read port uses a 32-bit 32-to-1 MUX to select one register output according to its 5-bit read register number.



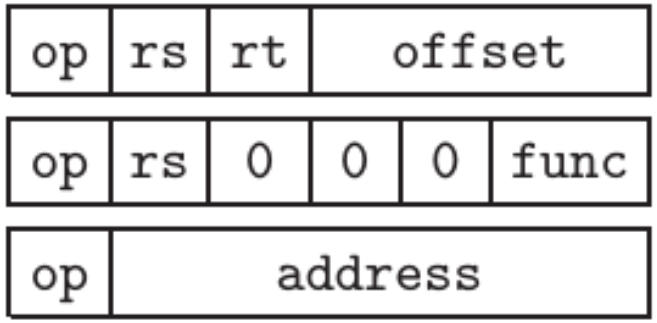
Datapath

- A CPU is consists of a **datapath** and a **control unit**.
- A **datapath** is a collection of functional units.
  - ✓ALU, register file, and MUX
- A **control unit** manages the operations of the **datapath**.

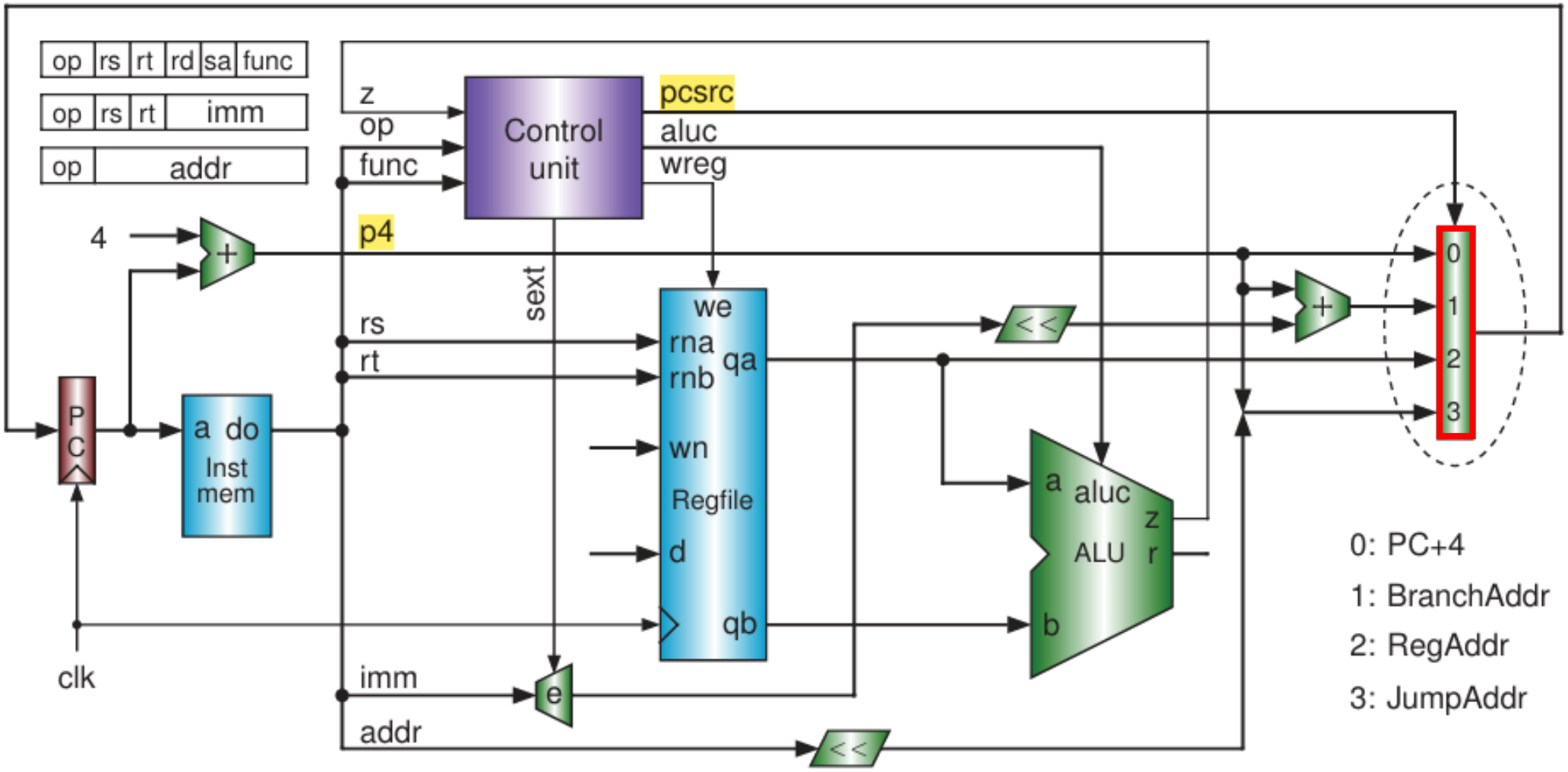
Next PC

- Selection for Next PC (Selection Signal: *pcsrc*)
- The PC will be updated on the rising edge of the clock.
- PC+4 will be written to the PC.
- But *beq/bne, jr, j/jal* may transfer control to a branch or jump target address not PC+4
- 4-to-1 MUX can be used to select an address for the next PC.

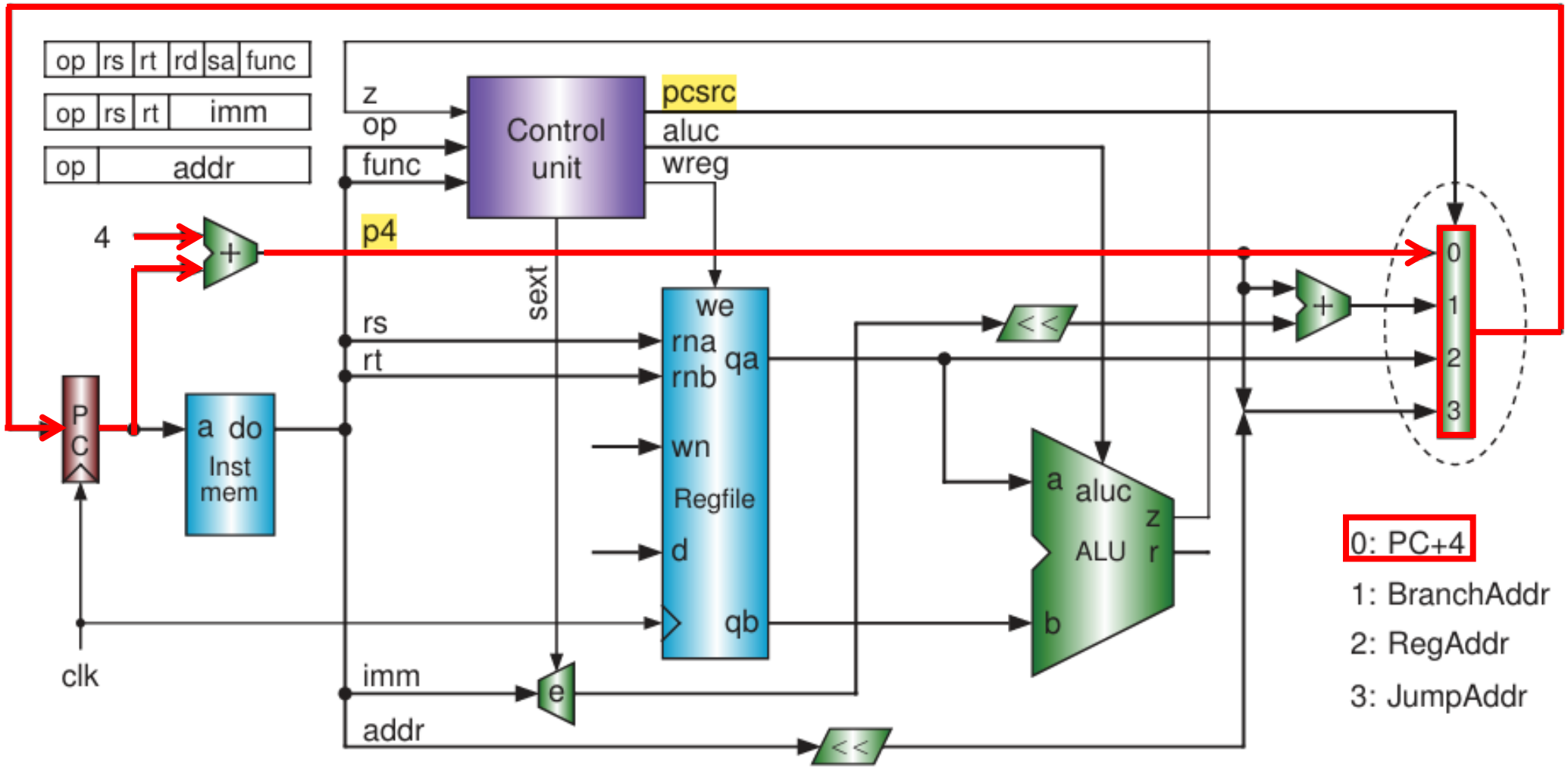
```
beq/bne rs, rt, label  # if (eq/ne) pc <-- label
jr      rs             # pc <-- rs
j/jal   address        # pc <-- address << 2
```



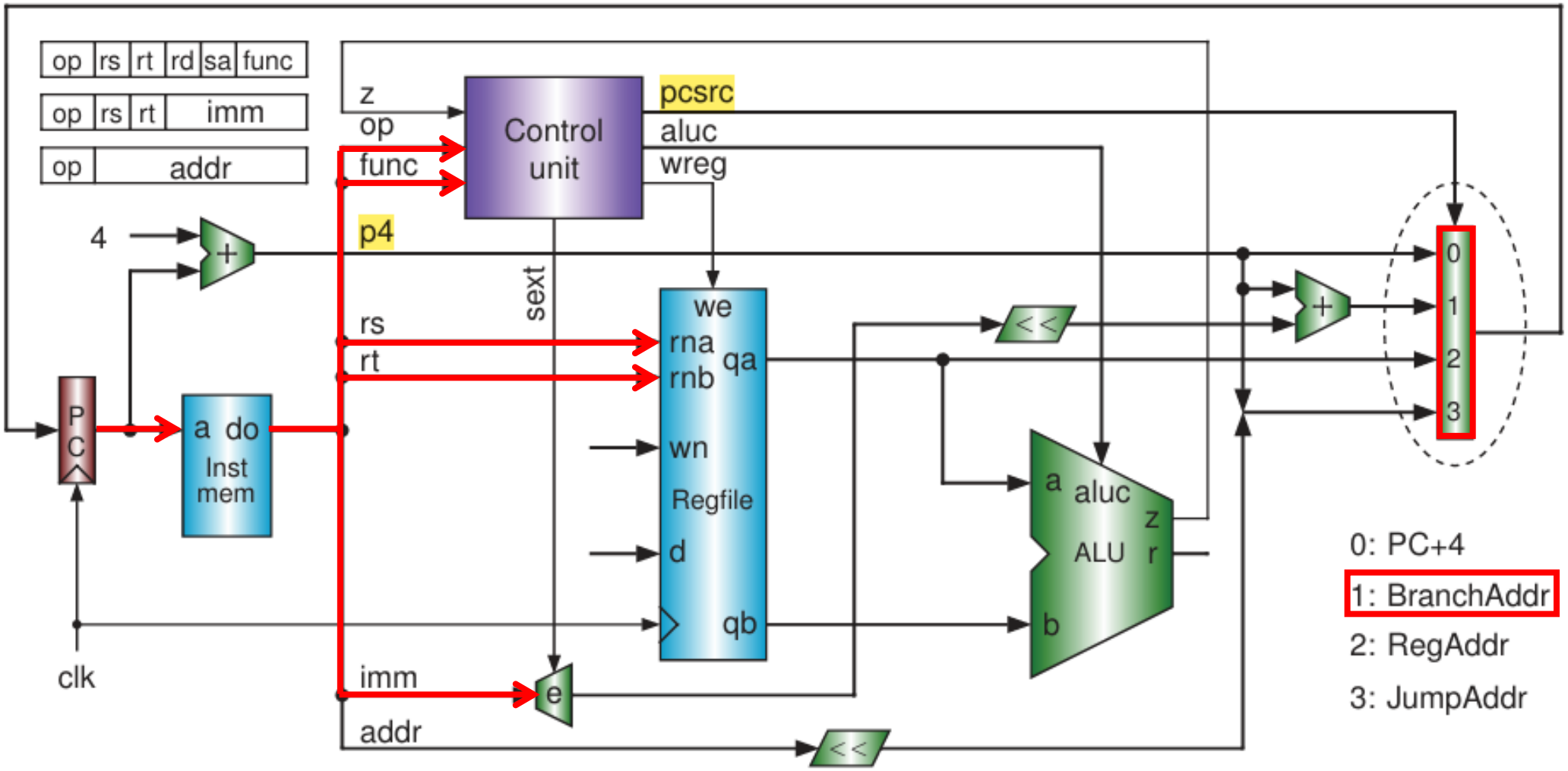
Next PC


op rs rt rd sa func
op rs rt imm
op addr
z
op
func
Control unit
pcsrc
aluc
wreg
4
+
p4
sext
rs
rt
we
rna
rnb
qa
wn
Regfile
d
qb
PC
a do
Inst mem
clk
imm
addr
e
<<
<<
a aluc
ALU
z
r
b
+
0
1
2
3
0: PC+4
1: BranchAddr
2: RegAddr
3: JumpAddr

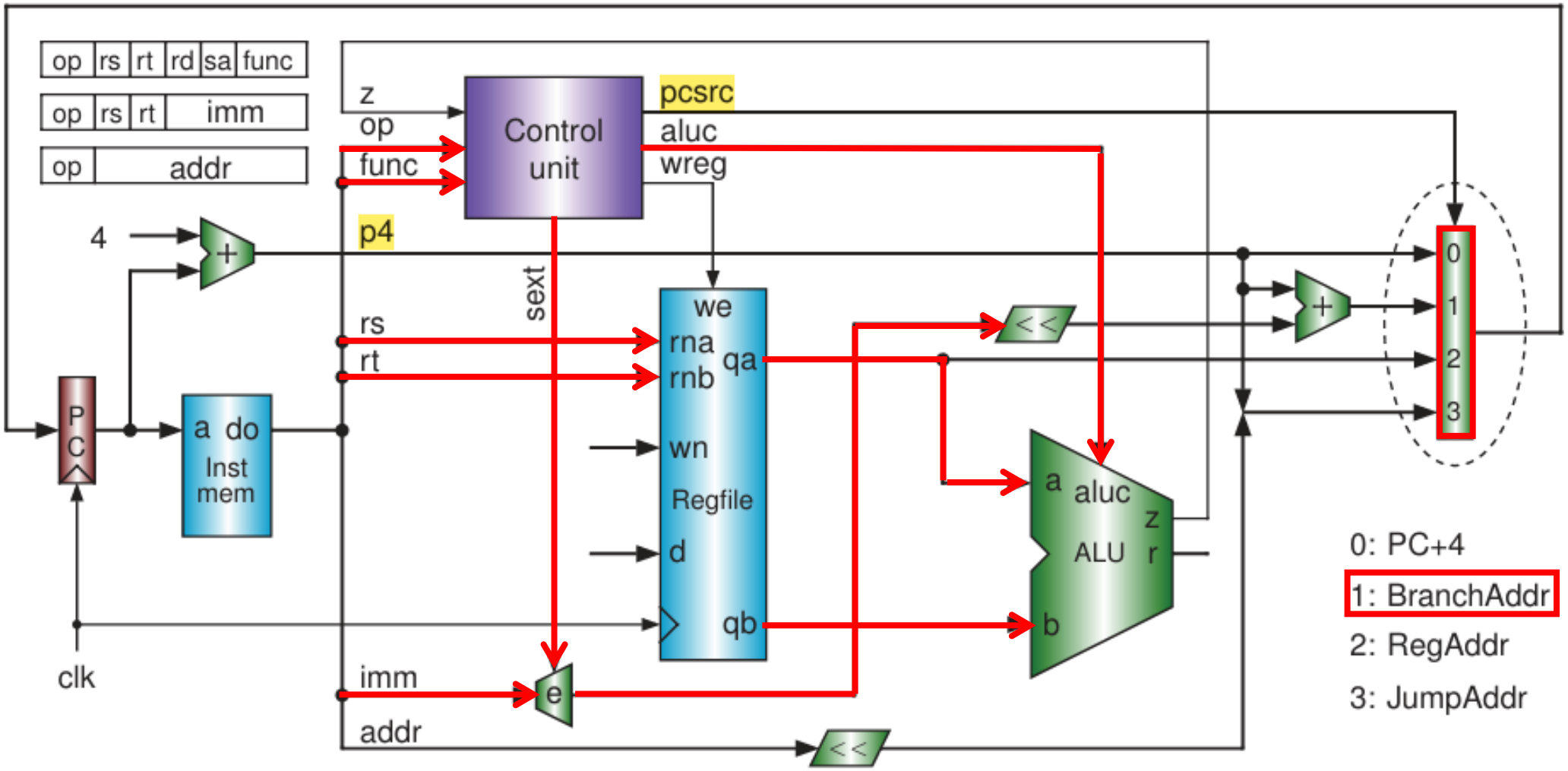
Next PC


op rs rt rd sa func
op rs rt imm
op addr
z
op
func
Control unit
pcsrc
aluc
wreg
4
+
p4
sext
we
rna
rnb
qa
rs
rt
wn
Regfile
d
qb
P C
a do
Inst mem
clk
imm
addr
e
<<
<<
a aluc
ALU
z
r
b
+
0
1
2
3
0: PC+4
1: BranchAddr
2: RegAddr
3: JumpAddr

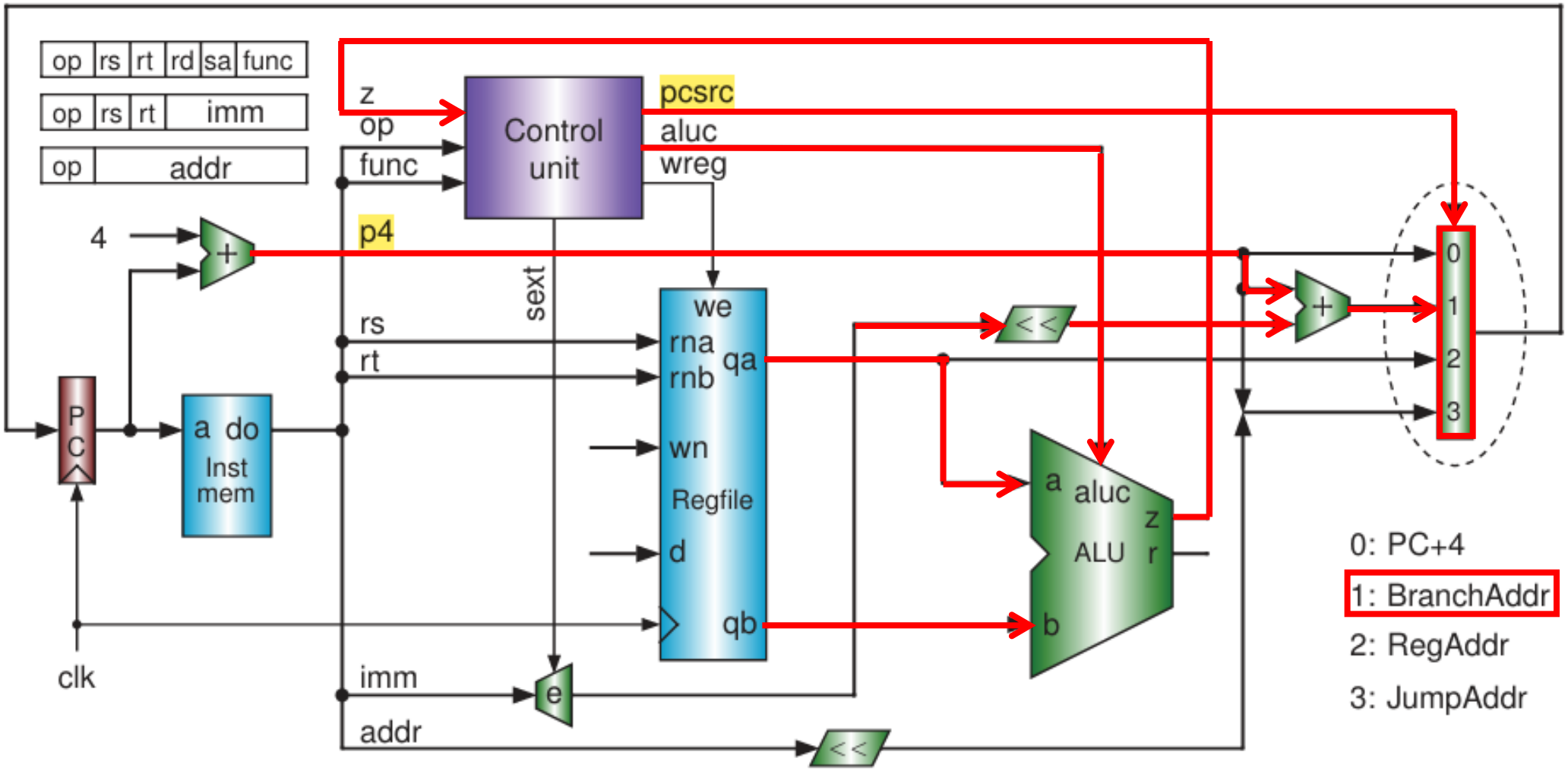
Next PC


op rs rt rd sa func
op rs rt imm
op addr
z
op
func
Control unit
pcsrc
aluc
wreg
4
+
p4
sext
rs
rt
we
rna
rnb
qa
wn
Regfile
d
qb
PC
a do
Inst mem
clk
imm
addr
e
<<
<<
a aluc
ALU
z
r
b
+
0
1
2
3
0: PC+4
1: BranchAddr
2: RegAddr
3: JumpAddr

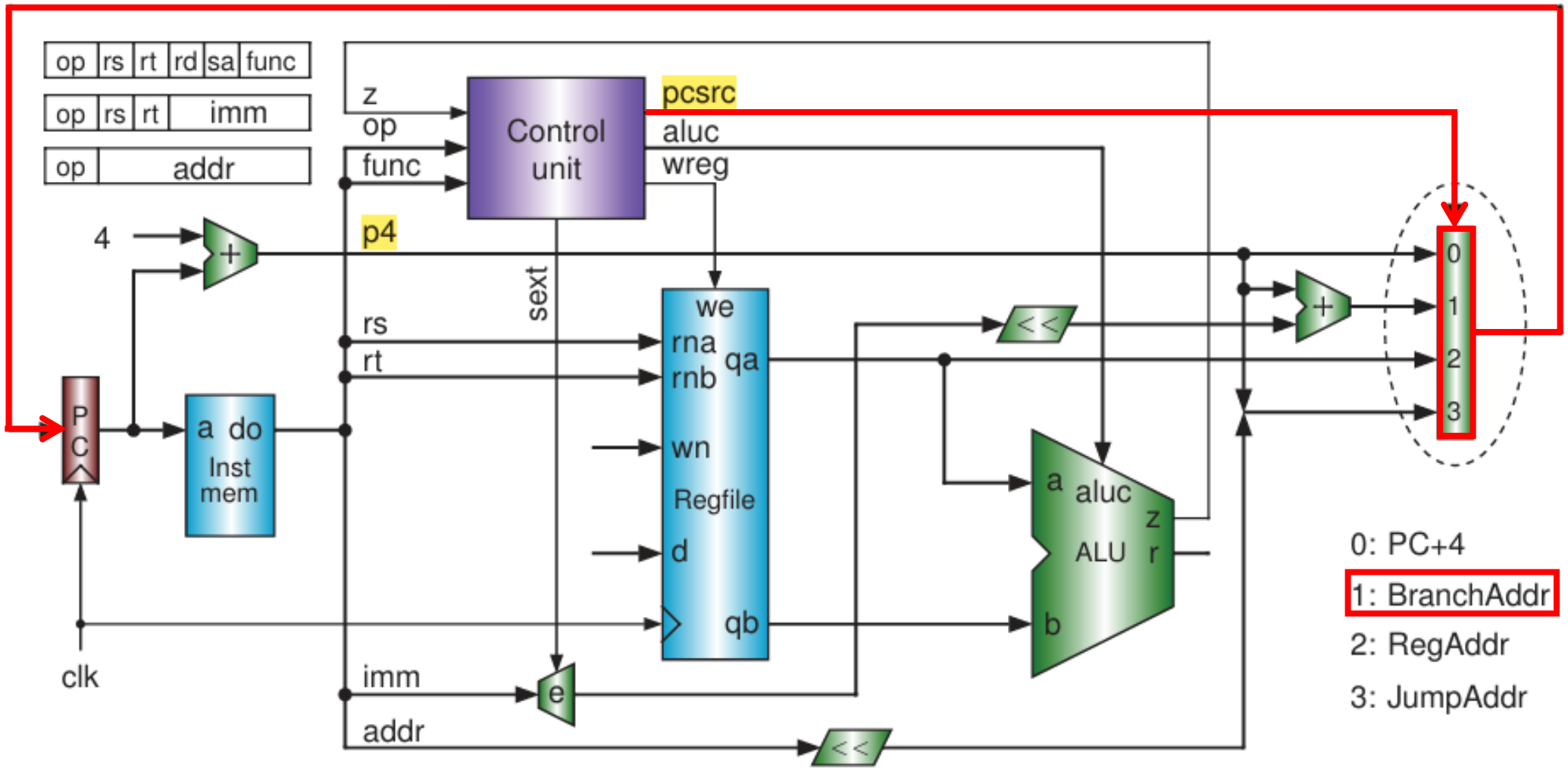
Next PC


op rs rt rd sa func
op rs rt imm
op addr
z
op
func
Control unit
pcsrc
aluc
wreg
4
+
p4
sext
rs
rt
we
rna
rnb
qa
wn
Regfile
d
qb
PC
a do
Inst mem
clk
imm
addr
e
<<
<<
+
a aluc
ALU
z
r
b
0
1
2
3
0: PC+4
1: BranchAddr
2: RegAddr
3: JumpAddr

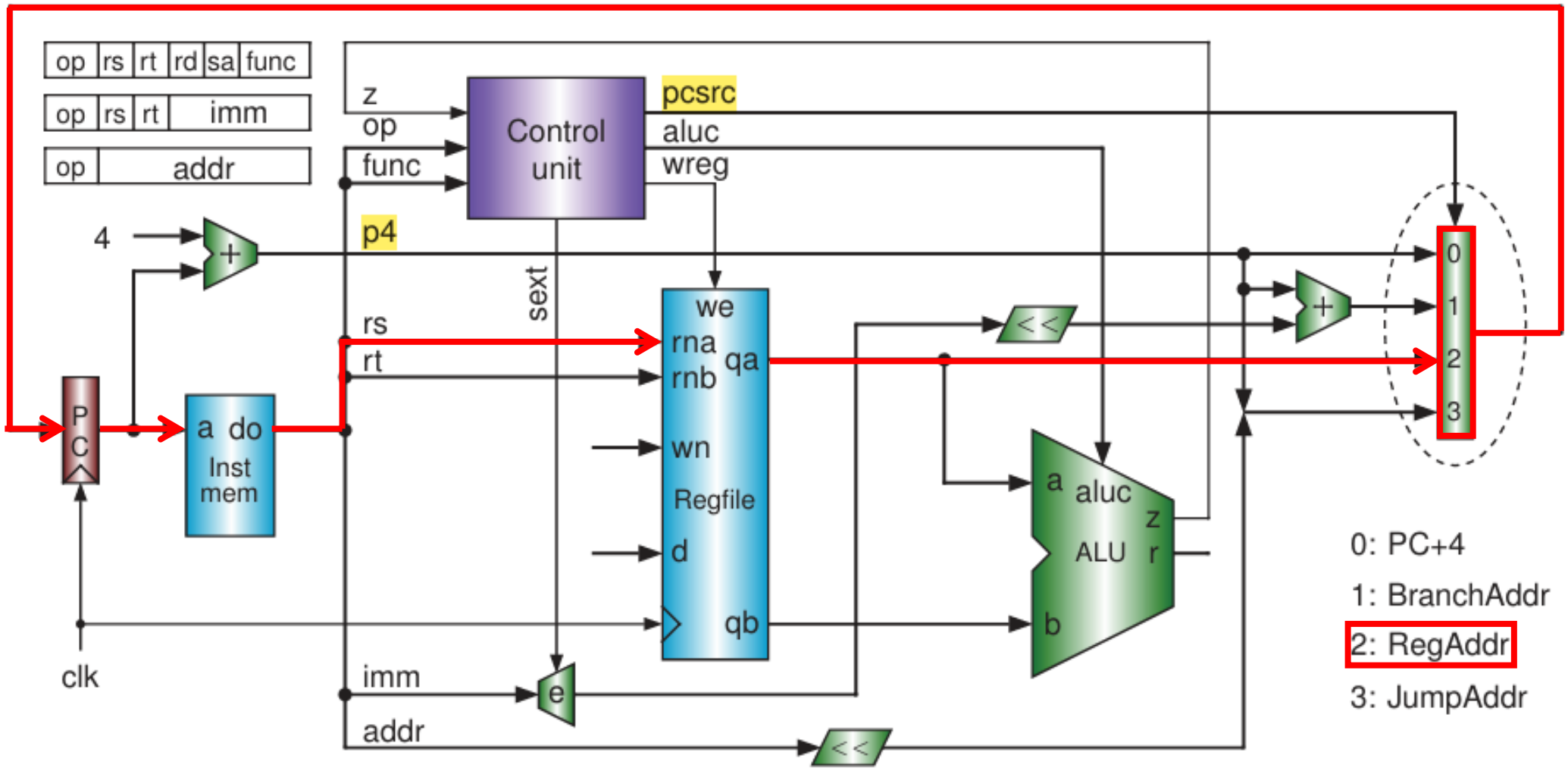
Next PC


op rs rt rd sa func
op rs rt imm
op addr
z
op
func
Control unit
pcsrc
aluc
wreg
4
+
p4
sext
we
rna
rnb
qa
rs
rt
wn
Regfile
d
qb
P C
a do
Inst mem
clk
imm
addr
e
<<
<<
a aluc
ALU
z
r
b
+
0
1
2
3
0: PC+4
1: BranchAddr
2: RegAddr
3: JumpAddr

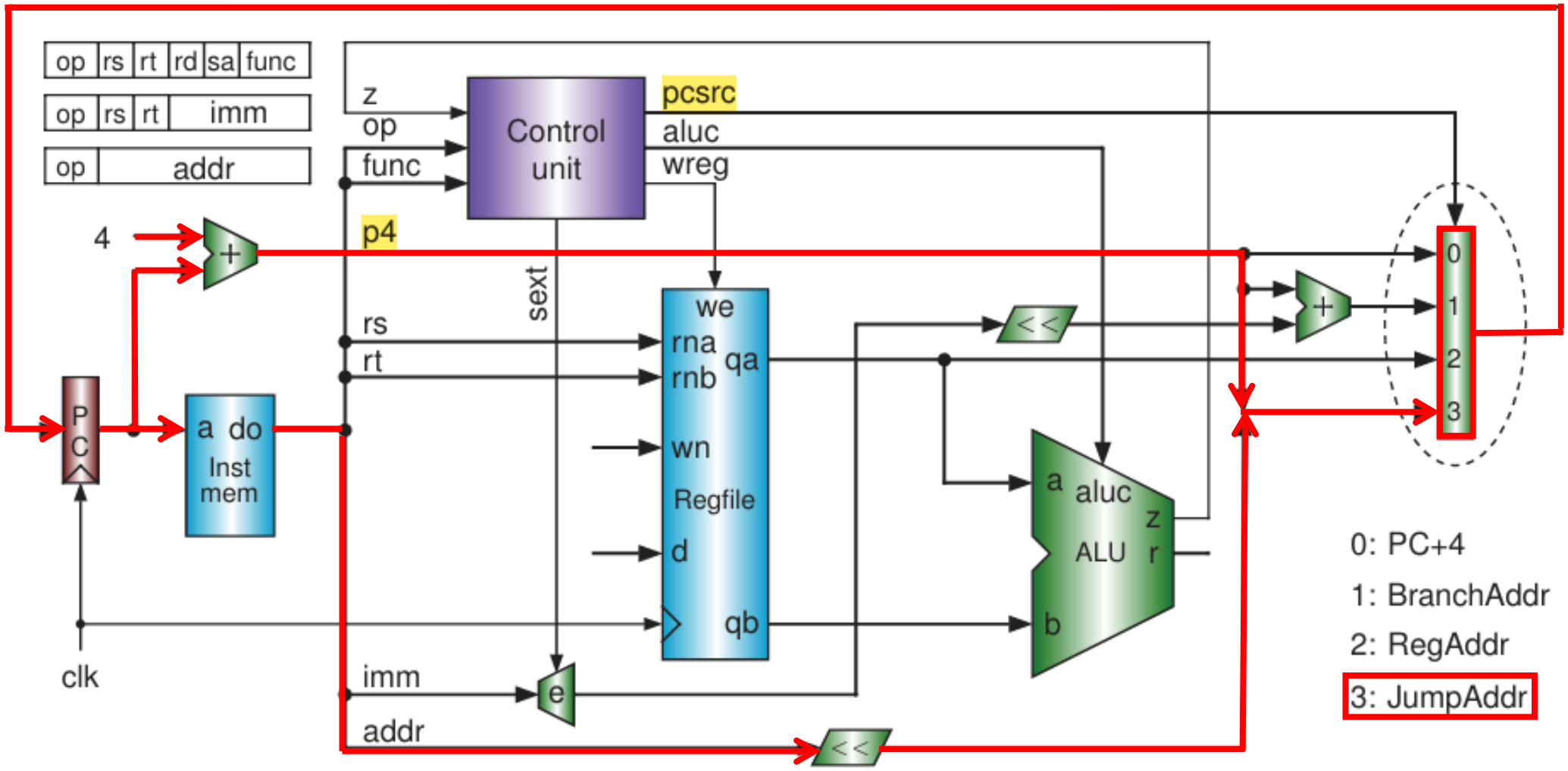
Next PC


op rs rt rd sa func
op rs rt imm
op addr
z
op
func
Control unit
pcsrc
aluc
wreg
4
+
p4
sext
rs
rt
we
rna
rnb
qa
wn
Regfile
d
qb
PC
a do
Inst mem
clk
imm
addr
e
<<
<<
a aluc
ALU
z
r
b
+
0
1
2
3
0: PC+4
1: BranchAddr
2: RegAddr
3: JumpAddr

Next PC

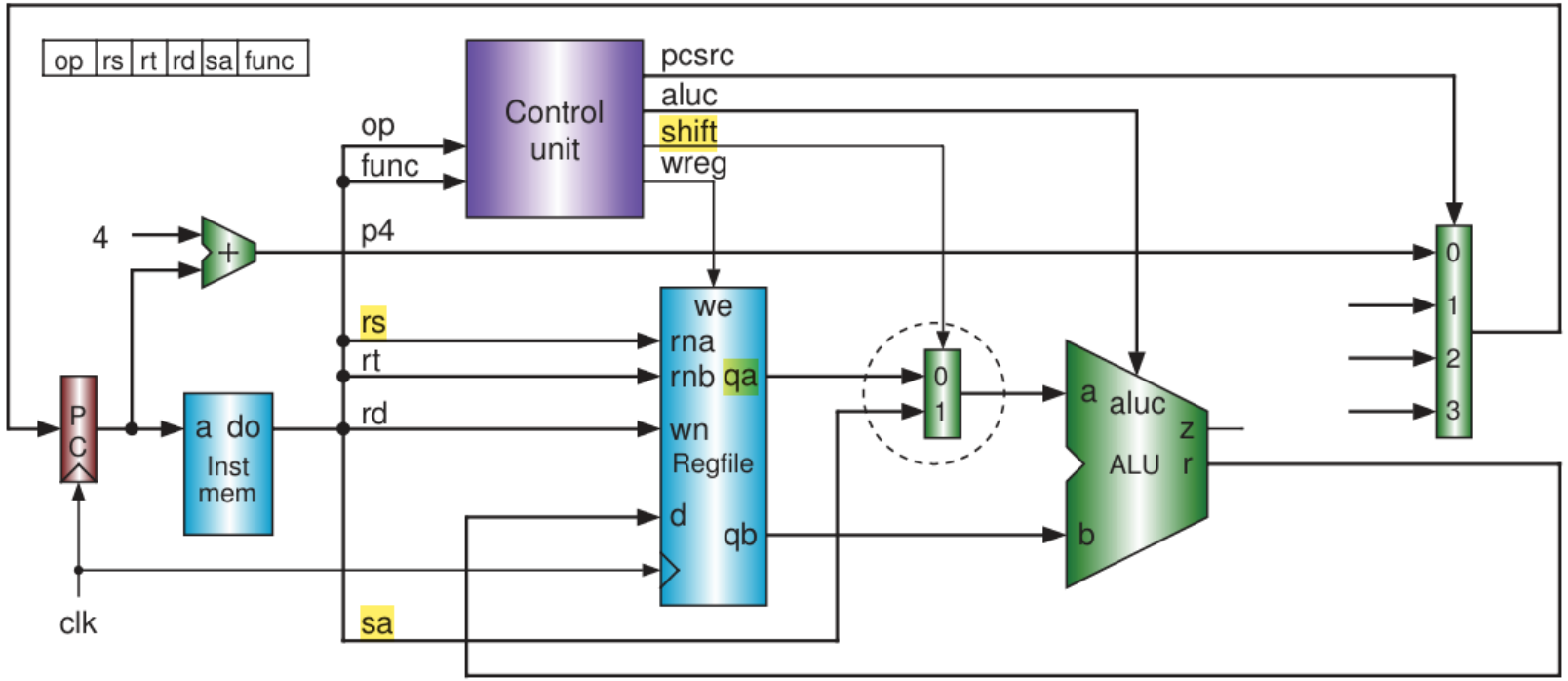

op rs rt rd sa func
op rs rt imm
op addr
z
op
func
Control unit
pcsrc
aluc
wreg
4
+
p4
sext
rs
rt
we
rna
rnb
qa
wn
Regfile
d
qb
PC
a do
Inst mem
clk
imm
addr
e
<<
<<
a aluc
ALU
z
r
b
+
0
1
2
3
0: PC+4
1: BranchAddr
2: RegAddr
3: JumpAddr

Next PC


op rs rt rd sa func
op rs rt imm
op addr
z
op
func
Control unit
pcsrc
aluc
wreg
4
+
p4
sext
rs
rt
we
rna
rnb
qa
wn
Regfile
d
qb
PC
a do
Inst mem
clk
imm
addr
e
<<
<<
a aluc
ALU
z
r
b
+
0
1
2
3
0: PC+4
1: BranchAddr
2: RegAddr
3: JumpAddr

ALU Input A

- Selection for ALU Input A (Selection signal: *shift*)
- Shift instructions use *sa* as the shift amount.
- Other instructions may use the value in the register *rs*.



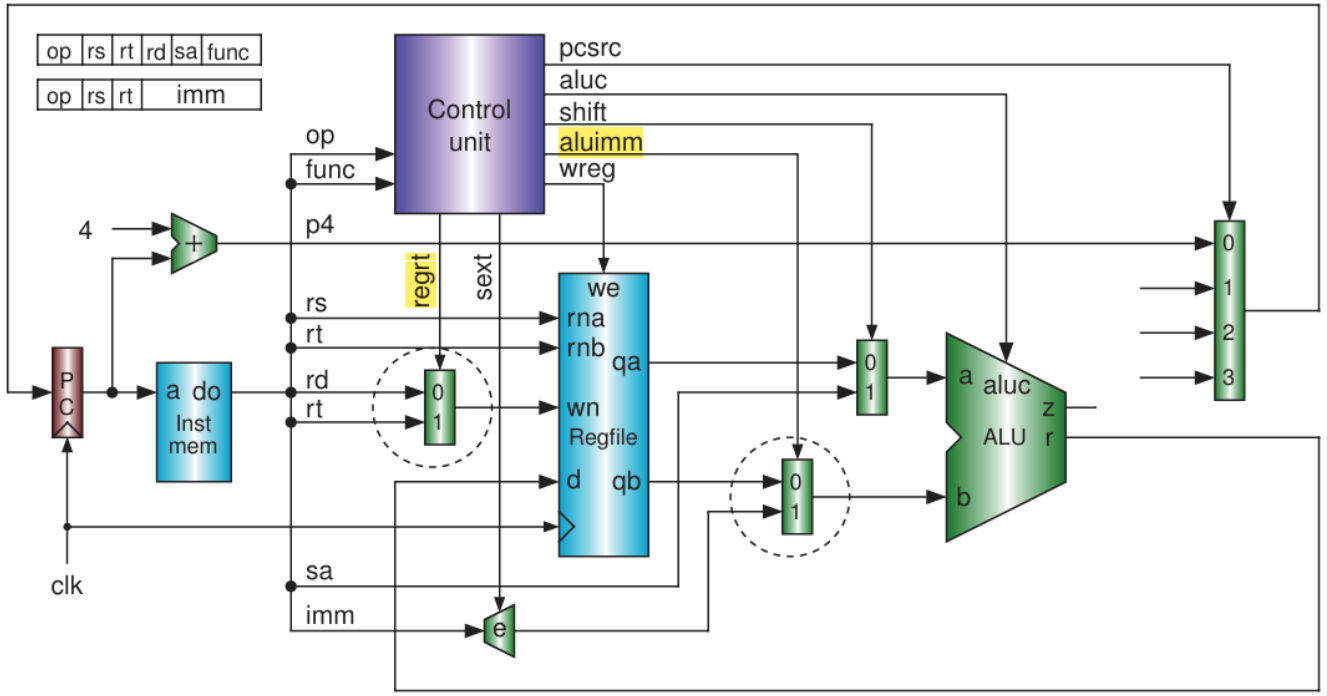

```
add rd, rs, rt        # rd <-- rs + rt
sll rd, rt, sa        # rd <-- rt << sa
```

op	rs	rt	rd	0	func

op	0	rt	rd	sa	func

ALU Input B

- Selection for ALU Input B (Selection Signal: *aluimm* and *regrt*)
- The *immediate* of the I-format instructions will be sent to input b.
- Other instructions may use the value in the register *rt*.




```
add   rd, rs, rt       # rd <-- rs + rt
addi  rt, rs, imm      # rt <-- rs + imm
```

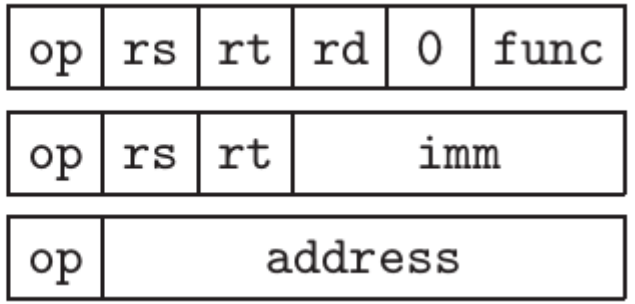
op	rs	rt	rd	0	func

op	rs	rt	imm

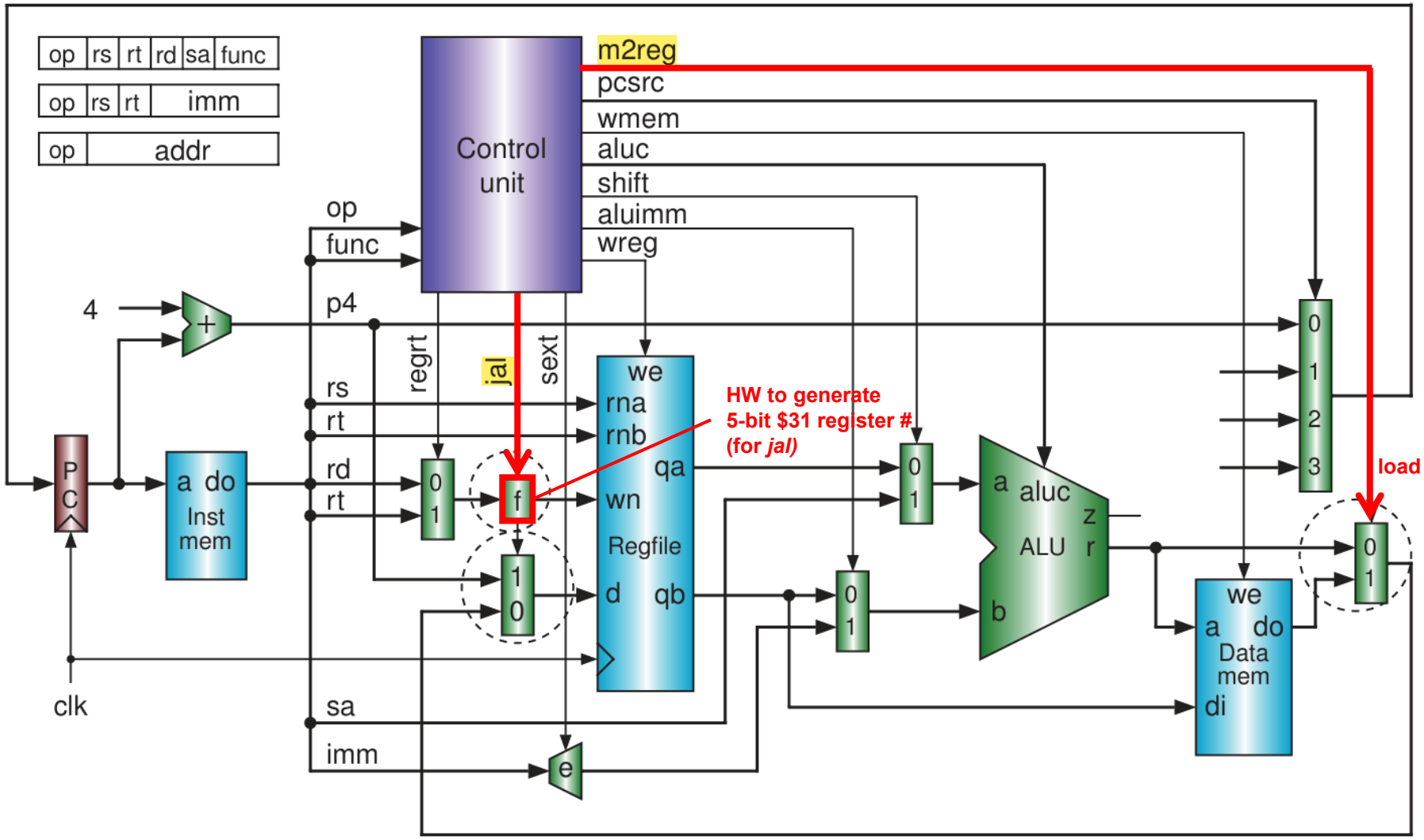
Register File Inputs

- Selection for Register File Inputs (Selection Signals: *m2reg, jal*)
- The data that will be written to the register file may be the output of the ALU, the data in the data memory, or the return address (for *jal*).
- The destination register number may be *rd*, *rt*, or a constant 31.
- The load instruction writes the data read from the memory to the register *rt* of the register file.
- Other instructions may write the output of the ALU to *rd* or *rt*.

```
add rd, rs, rt          # rd <-- rs + rt
lw  rt, imm(rs)         # rt <-- mem[rs+imm]
jal address # $31 <-- pc + 4; pc <-- address << 2
```



Register File Inputs



Control Unit

- Instruction decode

R-format			I- and J-format	
Inst.	op[5:0]	func[5:0]	Inst.	op[5:0]
i_add	000000	100000	i_addi	001000
i_sub	000000	100010	i_andi	001100
i_and	000000	100100	i_ori	001101
i_or	000000	100101	i_xori	001110
i_xor	000000	100110	i_lw	100011
i_sll	000000	000000	i_sw	101011
i_srl	000000	000010	i_beq	000100
i_sra	000000	000011	i_bne	000101
i_jr	000000	001000	i_lui	001111
			i_j	000010
			i_jal	000011

Control Unit

- Control signals

Signal	Meaning	Action
`wreg`	Write register	1: write; 0: do not write
`regrt`	Destination register is `rt`	1: select `rt`; 0: select `rd`
`jal`	Subroutine call	1: is `jal`; 0: is not `jal`
`m2reg`	Save memory data	1: select memory data; 0: select ALU result
`shift`	ALU A uses `sa`	1: select `sa`; 0: select register data
`aluimm`	ALU B uses immediate	1: select immediate; 0: select register data
`sext`	Immediate sign extend	1: sign-extend; 0: zero extend
`aluc[3:0]`	ALU operation control	x000: ADD; x100: SUB; x001: AND x101: OR; x010: XOR; x110: LUI 0011: SLL; 0111: SRL; 1111: SRA
`wmem`	Write memory	1: write memory; 0: do not write
`pcsrc[1:0]`	Next instruction address	00: select PC+4; 01: branch address 10: register data; 11: jump address

Control Unit

- Control signals truth table

Inst.	z	wreg	regrt	jal	m2reg	shift	aluimm	sext	aluc[3:0]	wmem	pcsrc[1:0]
i_add	x	1	0	0	0	0	0	x	x 0 0 0	0	0 0
i_sub	x	1	0	0	0	0	0	x	x 1 0 0	0	0 0
i_and	x	1	0	0	0	0	0	x	x 0 0 1	0	0 0
i_or	x	1	0	0	0	0	0	x	x 1 0 1	0	0 0
i_xor	x	1	0	0	0	0	0	x	x 0 1 0	0	0 0
i_sll	x	1	0	0	0	1	0	x	0 0 1 1	0	0 0
i_srl	x	1	0	0	0	1	0	x	0 1 1 1	0	0 0
i_sra	x	1	0	0	0	1	0	x	1 1 1 1	0	0 0
i_jr	x	0	x	x	x	x	x	x	x x x x	0	1 0
i_addi	x	1	1	0	0	0	1	1	x 0 0 0	0	0 0
i_andi	x	1	1	0	0	0	1	0	x 0 0 1	0	0 0
i_ori	x	1	1	0	0	0	1	0	x 1 0 1	0	0 0
i_xori	x	1	1	0	0	0	1	0	x 0 1 0	0	0 0
i_lw	x	1	1	0	1	0	1	1	x 0 0 0	0	0 0
i_sw	x	0	x	x	x	0	1	1	x 0 0 0	1	0 0
i_beq	0	0	x	x	x	0	0	1	x 0 1 0	0	0 0
i_beq	1	0	x	x	x	0	0	1	x 0 1 0	0	0 1
i_bne	0	0	x	x	x	0	0	1	x 0 1 0	0	0 1
i_bne	1	0	x	x	x	0	0	1	x 0 1 0	0	0 0
i_lui	x	1	1	0	0	x	1	x	x 1 1 0	0	0 0
i_j	x	0	x	x	x	x	x	x	x x x x	0	1 1
i_jal	x	1	x	1	x	x	x	x	x x x x	0	1 1

ALU control signals (*wreg, aluc*)

shift control signals (*wreg, shift, aluc*)

immediate control signals (*wreg, regrt, aluimm, sext, aluc*)

branch control signals (*z, sext, aluc, pcsrc*)

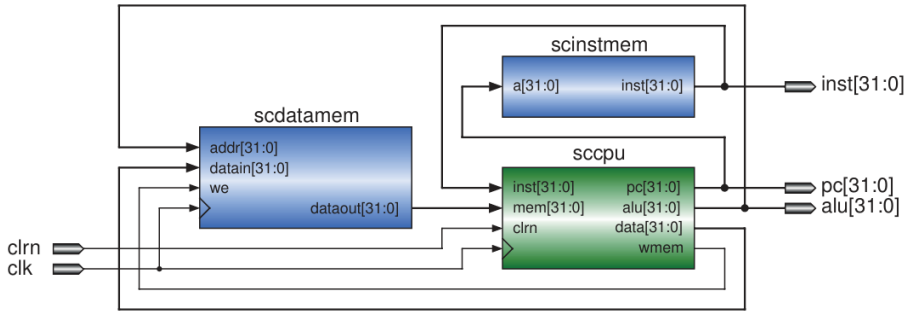
Control Unit

- Control signals truth table

Inst.	z	wreg	regrt	jal	m2reg	shift	aluimm	sext	aluc[3:0]	wmem	pcsrc[1:0]	
i_add	x	1	0	0	0	0	0	x	x 0 0 0	0	0 0	
i_sub	x	1	0	0	0	0	0	x	x 1 0 0	0	0 0	
i_and	x	1	0	0	0	0	0	x	x 0 0 1	0	0 0	
i_or	x	1	0	0	0	0	0	x	x 1 0 1	0	0 0	
i_xor	x	1	0	0	0	0	0	x	x 0 1 0	0	0 0	
i_sll	x	1	0	0	0	1	0	x	0 0 1 1	0	0 0	
i_srl	x	1	0	0	0	1	0	x	0 1 1 1	0	0 0	
i_sra	x	1	0	0	0	1	0	x	1 1 1 1	0	0 0	
i_jr	x	0	x	x	x	x	x	x	x x x x	0	1 0	jump register (*pcsrc*)
i_addi	x	1	1	0	0	0	1	1	x 0 0 0	0	0 0	
i_andi	x	1	1	0	0	0	1	0	x 0 0 1	0	0 0	
i_ori	x	1	1	0	0	0	1	0	x 1 0 1	0	0 0	
i_xori	x	1	1	0	0	0	1	0	x 0 1 0	0	0 0	
i_lw	x	1	1	0	1	0	1	1	x 0 0 0	0	0 0	load (*wreg, regrt, m2reg, aluimm, sext*)
i_sw	x	0	x	x	x	0	1	1	x 0 0 0	1	0 0	store *(aluimm, sext, wmem)*
i_beq	0	0	x	x	x	0	0	1	x 0 1 0	0	0 0	
i_beq	1	0	x	x	x	0	0	1	x 0 1 0	0	0 1	
i_bne	0	0	x	x	x	0	0	1	x 0 1 0	0	0 1	
i_bne	1	0	x	x	x	0	0	1	x 0 1 0	0	0 0	
i_lui	x	1	1	0	0	x	1	x	x 1 1 0	0	0 0	load upper immediate (*wreg, regrt, aluimm, aluc*)
i_j	x	0	x	x	x	x	x	x	x x x x	0	1 1	jump (*pcsrc*)
i_jal	x	1	x	1	x	x	x	x	x x x x	0	1 1	jump and link (*wreg, jal, pcsrc*)

Single-cycle CPU Design in Verilog HDL

- Dataflow style




```v
module sccomp (clk, clrn, inst, pc, aluout, memout);
    input           clk, clrn;
    output  [31:0]  pc;
    output  [31:0]  inst;
    output  [31:0]  aluout;
    output  [31:0]  memout;
    wire    [31:0]  data;
    wire            wmem;

    // cpu
    sccpu cpu (
        .clk(clk),
        .clrn(clrn),
        .inst(inst),
        .mem(memout),
        .pc(pc),
        .wmem(wmem),
        .alu(aluout),
        .data(data)
    );

    // inst memory
    scinstmem imem (
        .a(pc),
        .inst(inst)
    );

    // data memory
    scdatamem dmem (
        .clk(clk),
        .dataout(memout),
        .datain(data),
        .addr(aluout),
        .we(wmem)
    );
endmodule
```

```v
module sccu_dataflow (op, func, z, wmem, wreg, regrt, m2reg, aluc, shift, aluimm, pcsrc, jal, sext);
    input   [5:0]   op, func;               // op, func
    input           z;                      // alu zero tag
    output  [3:0]   aluc;                   // alu operation control
    output  [1:0]   pcsrc;                  // select pc source
    output          wreg;                   // write regfile
    output          regrt;                  // dest reg number is rt
    output          m2reg;                  // instruction is an lw
    output          shift;                  // instruction is a shift
    output          aluimm;                 // instruction is an i32
    output          jal;                    // instruction is an jal
    output          sext;                   // is sign extension
    output          wmem;                   // write data memory

    // decode instructions
    // r format
    wire    rtype = ~|op;
    wire    i_add = rtype &  func[5] & ~func[4] & ~func[3] & ~func[2] & ~func[1] & ~func[0];
    wire    i_sub = rtype &  func[5] & ~func[4] & ~func[3] & ~func[2] &  func[1] & ~func[0];
    wire    i_and = rtype &  func[5] & ~func[4] & ~func[3] &  func[2] & ~func[1] & ~func[0];
    wire    i_or  = rtype &  func[5] & ~func[4] & ~func[3] &  func[2] & ~func[1] &  func[0];
    wire    i_xor = rtype &  func[5] & ~func[4] & ~func[3] &  func[2] &  func[1] & ~func[0];
    wire    i_sll = rtype & ~func[5] & ~func[4] & ~func[3] & ~func[2] & ~func[1] & ~func[0];
    wire    i_srl = rtype & ~func[5] & ~func[4] & ~func[3] & ~func[2] &  func[1] & ~func[0];
    wire    i_sra = rtype & ~func[5] & ~func[4] & ~func[3] & ~func[2] &  func[1] &  func[0];
    wire    i_jr  = rtype & ~func[5] & ~func[4] &  func[3] & ~func[2] & ~func[1] & ~func[0];

    // i format
    wire    i_addi = ~op[5] & ~op[4] &  op[3] & ~op[2] & ~op[1] & ~op[0];
    wire    i_andi = ~op[5] & ~op[4] &  op[3] &  op[2] & ~op[1] & ~op[0];
    wire    i_ori  = ~op[5] & ~op[4] &  op[3] &  op[2] & ~op[1] &  op[0];
    wire    i_xori = ~op[5] & ~op[4] &  op[3] &  op[2] &  op[1] & ~op[0];
    wire    i_lw   =  op[5] & ~op[4] & ~op[3] & ~op[2] &  op[1] &  op[0];
    wire    i_sw   =  op[5] & ~op[4] &  op[3] & ~op[2] &  op[1] &  op[0];
    wire    i_beq  = ~op[5] & ~op[4] & ~op[3] &  op[2] & ~op[1] & ~op[0];
    wire    i_bne  = ~op[5] & ~op[4] & ~op[3] &  op[2] & ~op[1] &  op[0];
    wire    i_lui  = ~op[5] & ~op[4] &  op[3] &  op[2] &  op[1] &  op[0];

    // j format
    wire    i_j    = ~op[5] & ~op[4] & ~op[3] & ~op[2] &  op[1] & ~op[0];
    wire    i_jal  = ~op[5] & ~op[4] & ~op[3] & ~op[2] &  op[1] &  op[0];

    // generate control signals
    assign  regrt   = i_addi | i_andi | i_ori | i_xori | i_lw | i_lui;
    assign  jal     = i_jal;
    assign  m2reg   = i_lw;
    assign  wmem    = i_sw;
    assign  aluc[3] = i_sra;
    assign  aluc[2] = i_sub  | i_or   | i_srl | i_sra  | i_ori  | i_lui;
    assign  aluc[1] = i_xor  | i_sll  | i_srl | i_sra  | i_xori | i_beq  | i_bne | i_lui;
    assign  aluc[0] = i_and  | i_or   | i_sll | i_srl  | i_sra  | i_andi | i_ori;
    assign  shift   = i_sll  | i_srl  | i_sra;
    assign  aluimm  = i_addi | i_andi | i_ori | i_xori | i_lw   | i_lui  | i_sw;
    assign  sext    = i_addi | i_lw   | i_sw  | i_beq  | i_bne;
    assign  pcsrc[1]= i_jr   | i_j    | i_jal;
    assign  pcsrc[0]= i_beq & z | i_bne & ~z | i_j | i_jal;
    assign  wreg    = i_add  | i_sub  | i_and  | i_or   | i_xor  | i_sll  |
                      i_srl  | i_sra  | i_addi | i_andi | i_ori  | i_xori |
                      i_lw   | i_lui  | i_jal;
endmodule
```

Single-cycle CPU Design in Verilog HDL

- Behavioral style

```
1   module sccpu(
2       input               clk,
3       input               clrn,
4       input       [31:0]  mem,
5       input       [31:0]  inst,
6       output  reg [31:0]  pc,
7       output  reg [31:0]  alu,
8       output      [31:0]  data,
9       output  reg         wmem
10  );
11
12      reg             wreg;
13      reg     [4:0]   dest_rn;
14      reg     [31:0]  next_pc;
15      wire    [31:0]  pc4 = pc + 4;
```

```
17      // Instruction Field
18      wire    [5:0]   op   = inst[31:26];
19      wire    [4:0]   rs   = inst[25:21];
20      wire    [4:0]   rt   = inst[20:16];
21      wire    [4:0]   rd   = inst[15:11];
22      wire    [4:0]   sa   = inst[10:06];
23      wire    [5:0]   func = inst[05:00];
24      wire    [15:0]  imm  = inst[15:00];
25      wire    [25:0]  addr = inst[25:00];
26      wire            sign = inst[15];
27      wire    [31:0]  offset = {{14{sign}},imm,2'b00};
28      wire    [31:0]  j_addr = {pc4[31:28],addr,2'b00};
```

Single-cycle CPU Design in Verilog HDL

- Behavioral style

```
30      // Instruction Decode
31      // R-format
32      wire    i_add   = (op == 6'b000000) & (func == 6'b100000);
33      wire    i_sub   = (op == 6'b000000) & (func == 6'b100010);
34      wire    i_and   = (op == 6'b000000) & (func == 6'b100100);
35      wire    i_or    = (op == 6'b000000) & (func == 6'b100101);
36      wire    i_xor   = (op == 6'b000000) & (func == 6'b100110);
37      wire    i_sll   = (op == 6'b000000) & (func == 6'b000000);
38      wire    i_srl   = (op == 6'b000000) & (func == 6'b000010);
39      wire    i_sra   = (op == 6'b000000) & (func == 6'b000011);
40      wire    i_jr    = (op == 6'b000000) & (func == 6'b001000);
41      // I-format
42      wire    i_addi  = (op == 6'b001000);
43      wire    i_andi  = (op == 6'b001100);
44      wire    i_ori   = (op == 6'b001101);
45      wire    i_xori  = (op == 6'b001110);
46      wire    i_lw    = (op == 6'b100011);
47      wire    i_sw    = (op == 6'b101011);
48      wire    i_beq   = (op == 6'b000100);
49      wire    i_bne   = (op == 6'b000101);
50      wire    i_lui   = (op == 6'b001111);
51      // J-format
52      wire    i_j     = (op == 6'b000010);
53      wire    i_jal   = (op == 6'b000011);
```

```
55      // Program Counter
56      always @(posedge clk or negedge clrn) begin
57          if (!clrn) begin
58              pc <= 0;
59          end else begin
60              pc <= next_pc;
61          end
62      end
63
64      // Data written into Register File
65      wire    [31:0]  data_to_regfile = i_lw ? mem : alu;
66
67      // Register File
68      reg     [31:0]  regfile [1:31];
69      wire    [31:0]  a = (rs == 0) ? 0 : regfile[rs];
70      wire    [31:0]  b = (rt == 0) ? 0 : regfile[rt];
71
72      integer i; // to initialize regfile or the error occurs
73      always @(posedge clk or negedge clrn) begin
74          if (!clrn) begin
75              for (i = 1; i < 32; i = i + 1) begin
76                  regfile[i] <= 0;
77              end
78          end else if (wreg && (dest_rn != 0)) begin
79              regfile[dest_rn] <= data_to_regfile;
80          end
81      end
82
83      // Output signals
84      assign  data    = b;
```

Single-cycle CPU Design in Verilog HDL

- Behavioral style

```
87      always @(*) begin
88          alu     = 0;
89          dest_rn = rd;
90          wreg    = 0;
91          wmem    = 0;
92          next_pc = pc4;
93
94          case (1'b1)
95              i_add: begin
96                  alu     = a + b;
97                  wreg    = 1;
98              end
99
100             i_sub: begin
101                 alu     = a - b;
102                 wreg    = 1;
103             end
104
105             i_and: begin
106                 alu     = a & b;
107                 wreg    = 1;
108             end
109
110             i_or: begin
111                 alu     = a | b;
112                 wreg    = 1;
113             end
114
115             i_xor: begin
116                 alu     = a ^ b;
117                 wreg    = 1;
118             end
119
120             i_sll: begin
121                 alu     = b << sa;
122                 wreg    = 1;
123             end
```

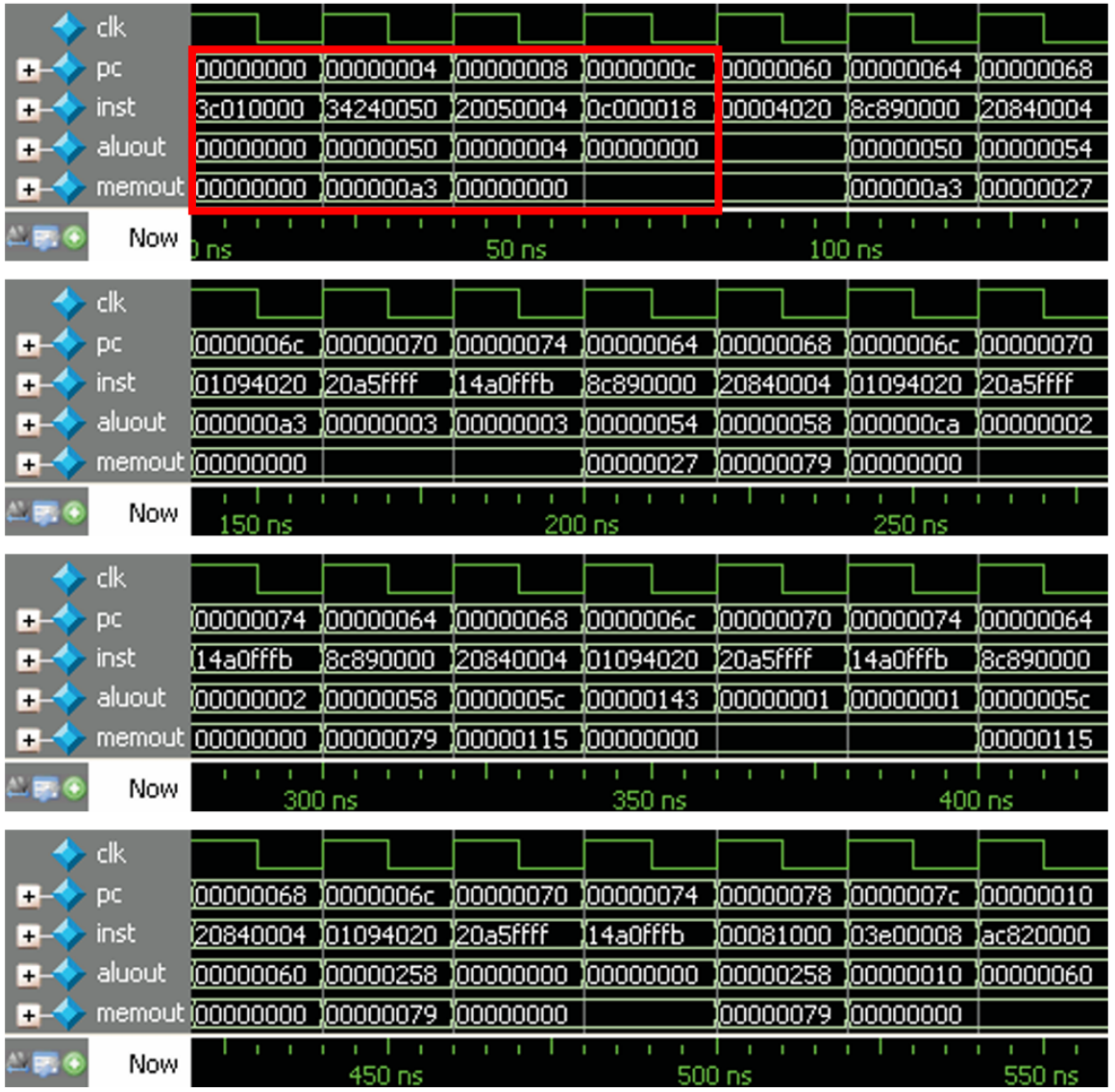
```
125             i_srl: begin
126                 alu     = b >> sa;
127                 wreg    = 1;
128             end
129
130             i_sra: begin
131                 alu     = $signed(b) >>> sa;
132                 wreg    = 1;
133             end
134
135             i_jr: begin
136                 next_pc = a;
137             end
138
139             i_addi: begin
140                 alu     = a + {{16{sign}},imm};
141                 dest_rn = rt;
142                 wreg    = 1;
143             end
144
145             i_andi: begin
146                 alu     = a & {16'h0,imm};
147                 dest_rn = rt;
148                 wreg    = 1;
149             end
150
151             i_ori: begin
152                 alu     = a | {16'h0,imm};
153                 dest_rn = rt;
154                 wreg    = 1;
155             end
156
157             i_xori: begin
158                 alu     = a ^ {16'h0,imm};
159                 dest_rn = rt;
160                 wreg    = 1;
161             end
```

```
163             i_lw: begin
164                 alu     = a + {{16{sign}},imm};
165                 dest_rn = rt;
166                 wreg    = 1;
167             end
168
169             i_sw: begin
170                 alu     = a + {{16{sign}},imm};
171                 dest_rn = rt;
172                 wmem    = 1;
173                 wreg    = 1;
174             end
175
176             i_beq: begin
177                 if (a == b) begin
178                     next_pc = pc4 + offset;
179                 end
180             end
181
182             i_bne: begin
183                 if (a != b) begin
184                     next_pc = pc4 + offset;
185                 end
186             end
187
188             i_lui: begin
189                 alu     = {imm,16'h0};
190                 wreg    = 1;
191             end
192
193             i_j: begin
194                 next_pc = j_addr;
195             end
196
197             i_jal: begin
198                 alu     = pc4;
199                 wreg    = 1;
200                 dest_rn = 5'd31;
201                 next_pc = j_addr;
202             end
```

Simulation

- Test program

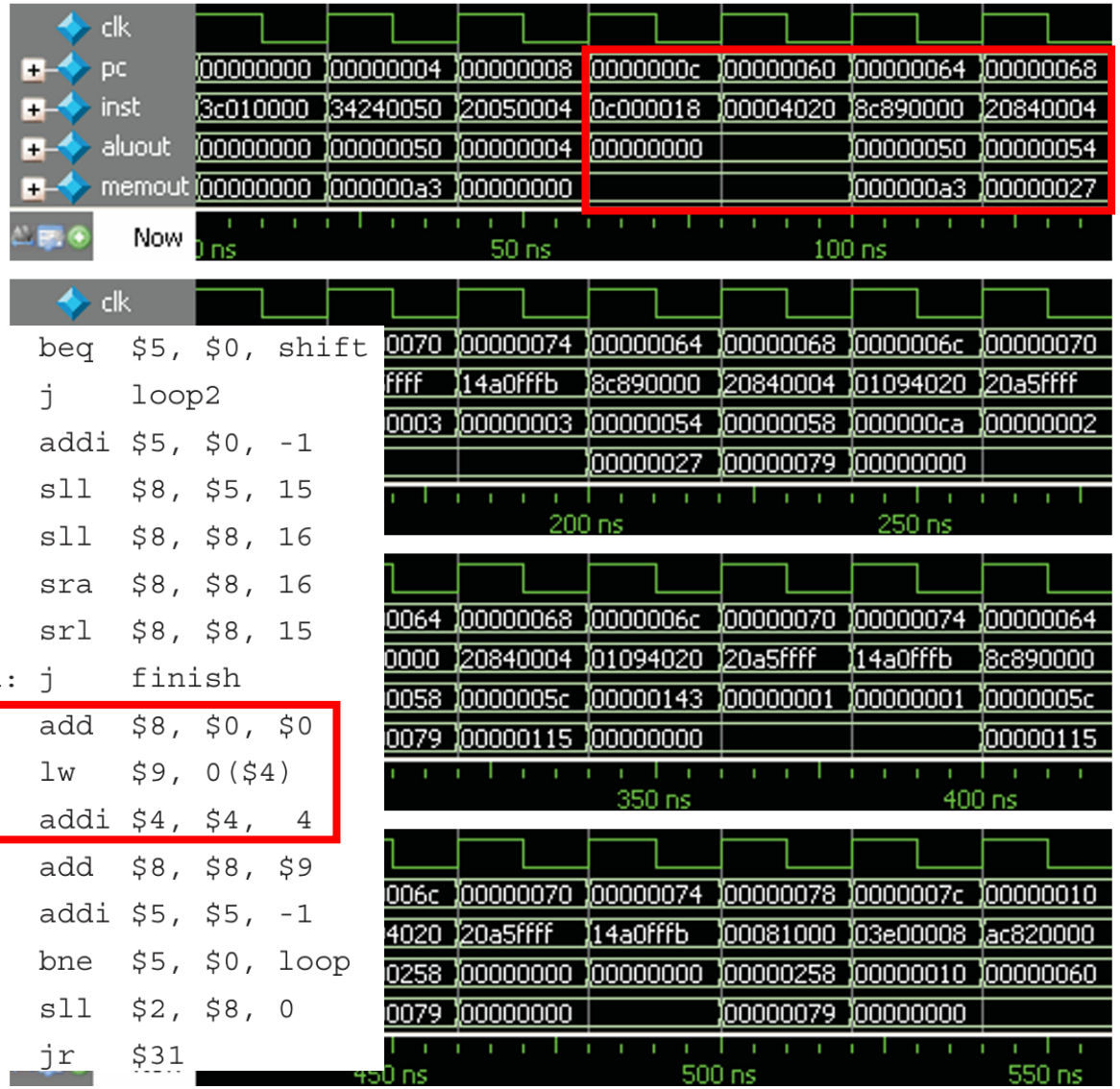
```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
```



Simulation

- Test program

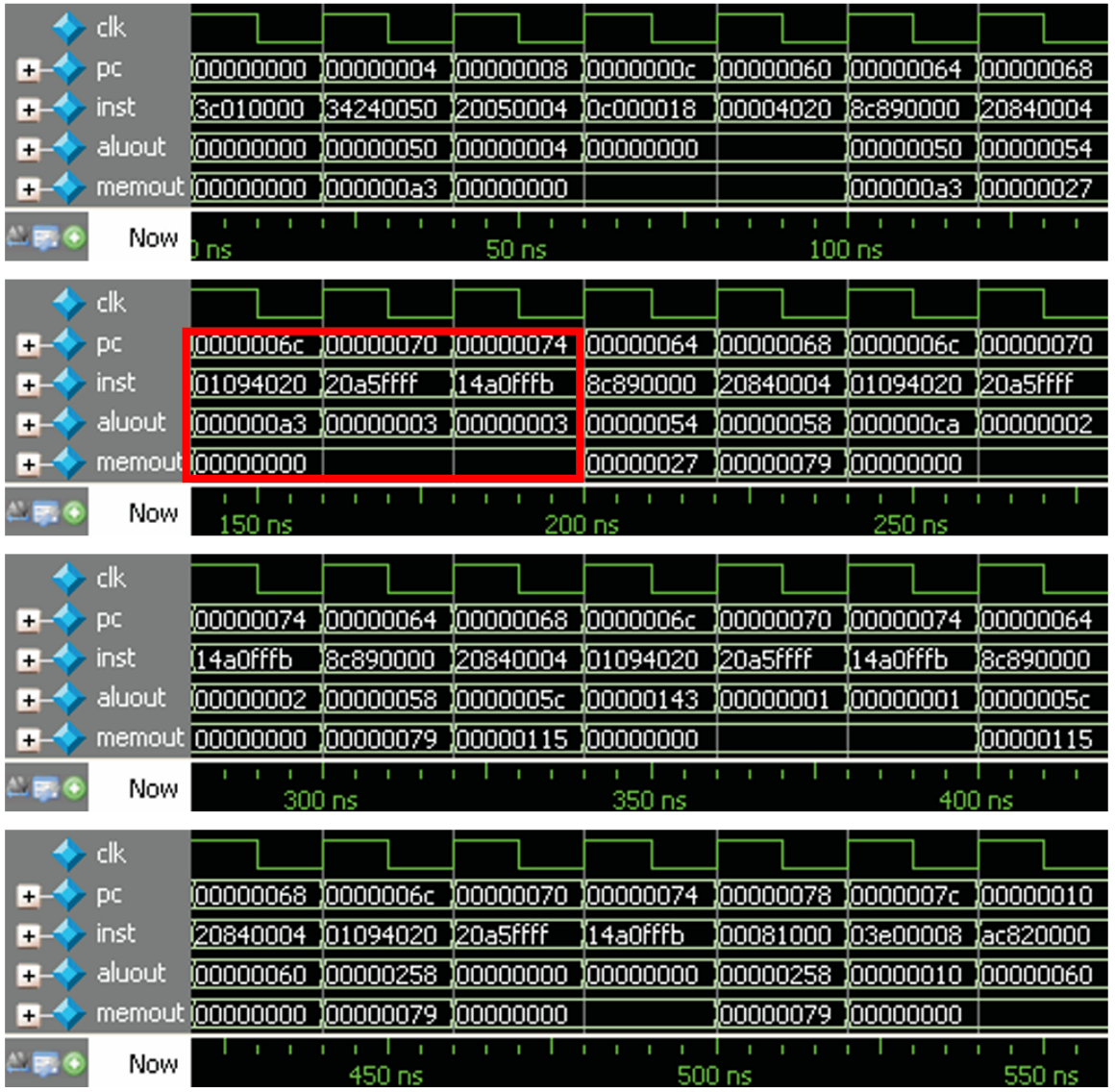
```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xff
(28)          xori $8, $8, 0x55
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

- Test program

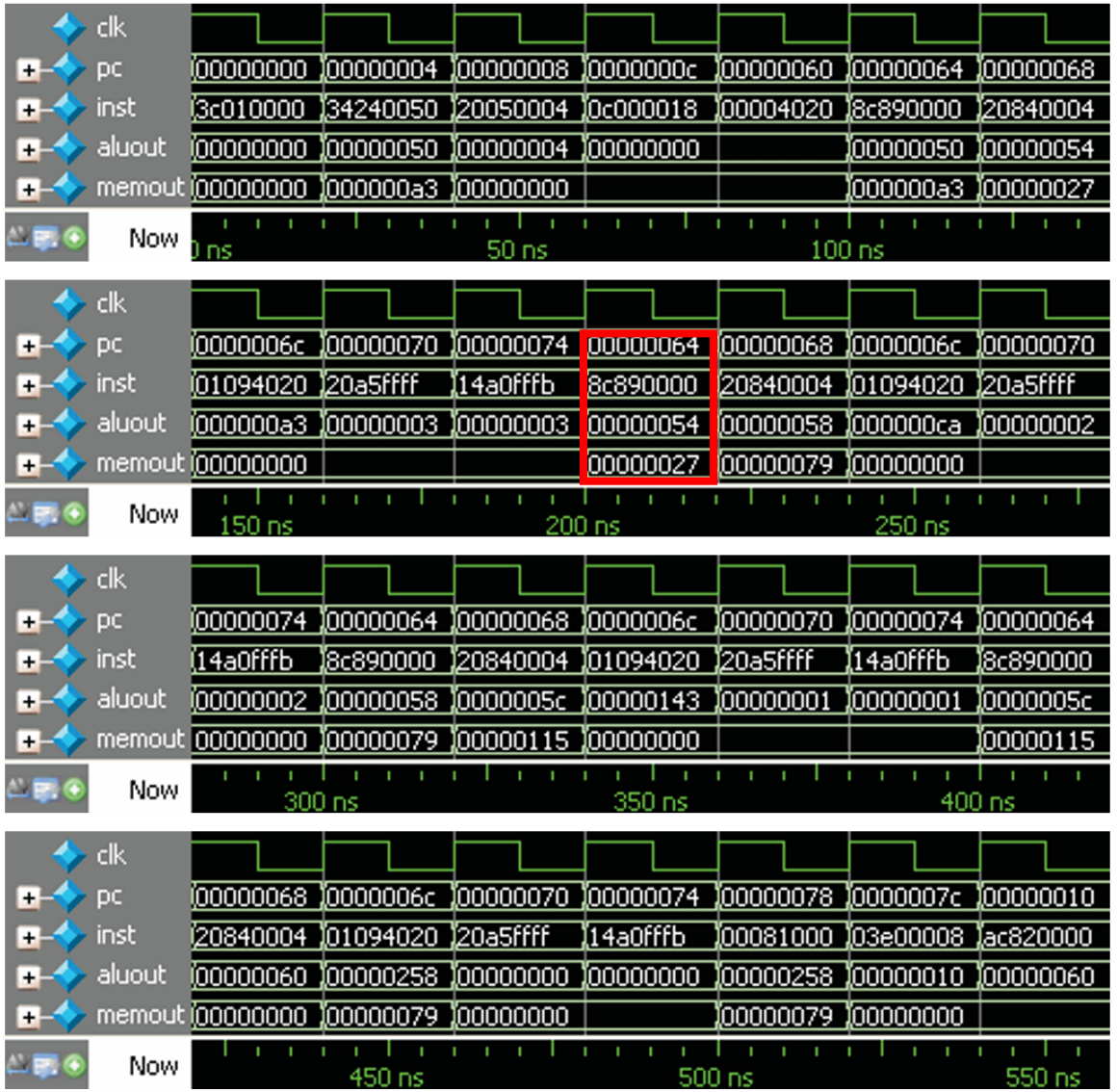
```
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

- Test program

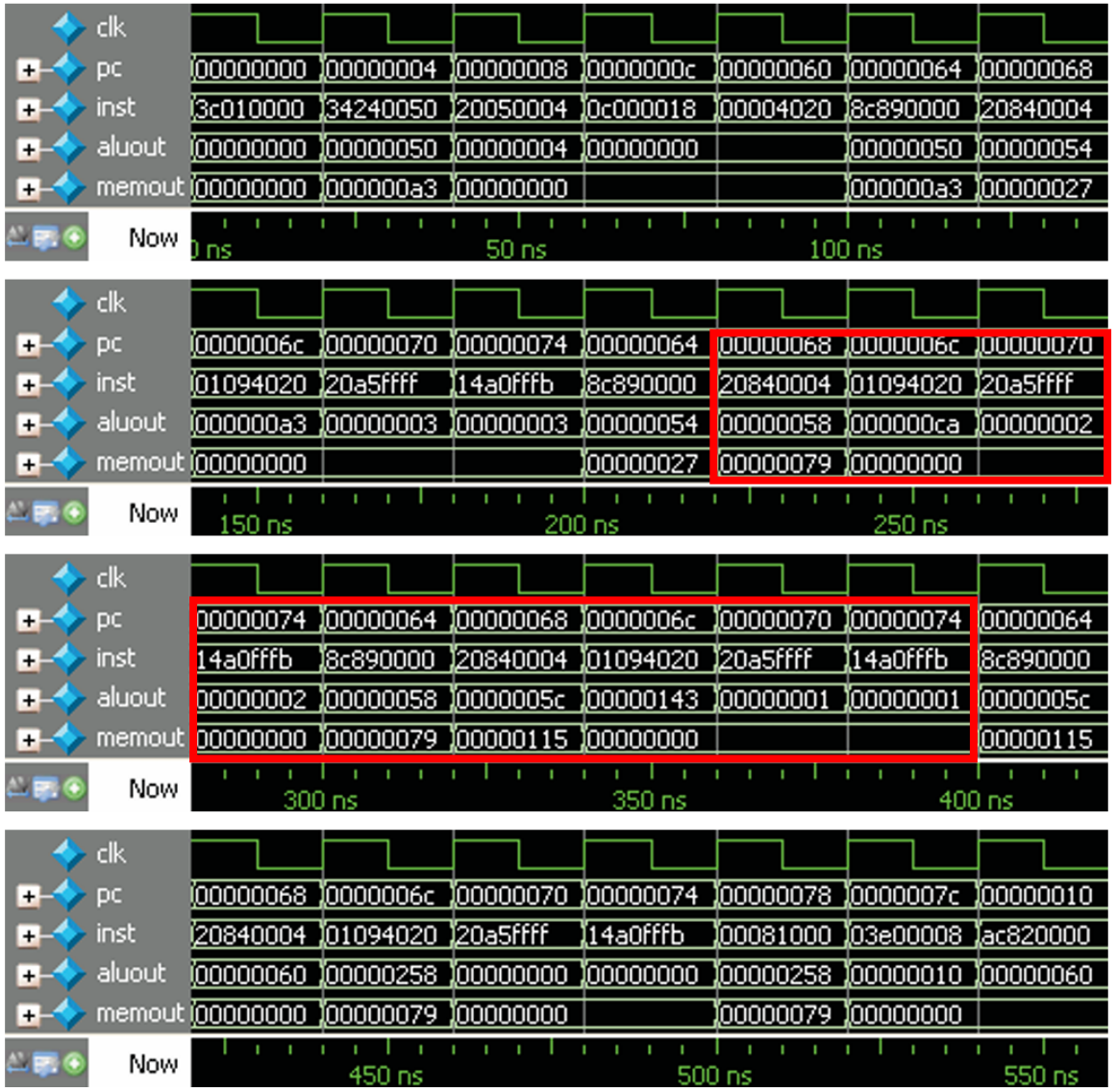
```
(40)          beq   $5, $0, shift
(44)          j     loop2
(48) shift:   addi  $5, $0, -1
(4c)          sll   $8, $5, 15
(50)          sll   $8, $8, 16
(54)          sra   $8, $8, 16
(58)          srl   $8, $8, 15
(5c) finish:  j     finish
(60) sum:     add   $8, $0, $0
(64) loop:    lw    $9, 0($4)
(68)          addi  $4, $4,  4
(6c)          add   $8, $8, $9
(70)          addi  $5, $5, -1
(74)          bne   $5, $0, loop
(78)          sll   $2, $8, 0
(7c)          jr    $31
```



Simulation

- Test program

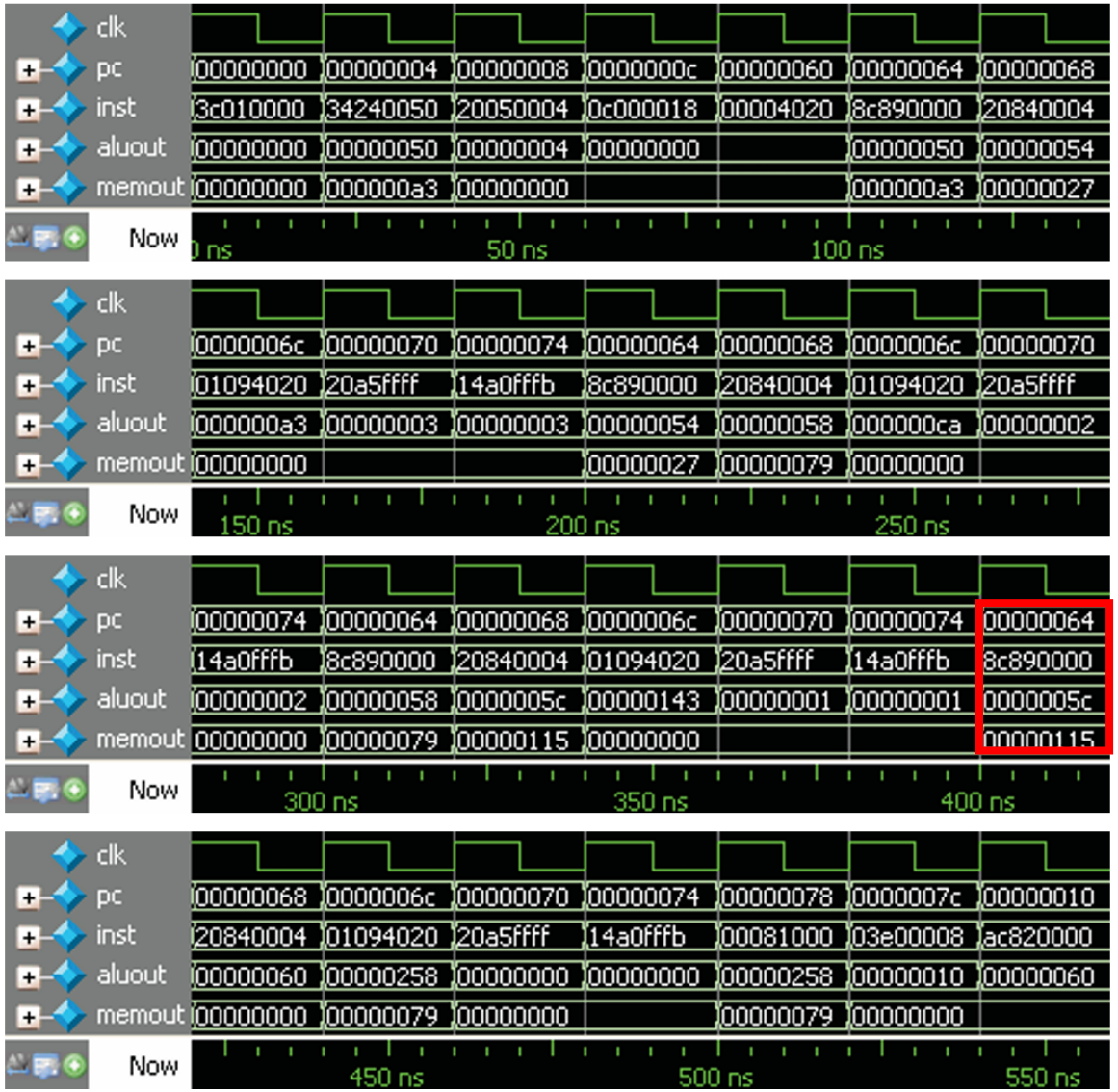
```
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

- Test program

```
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

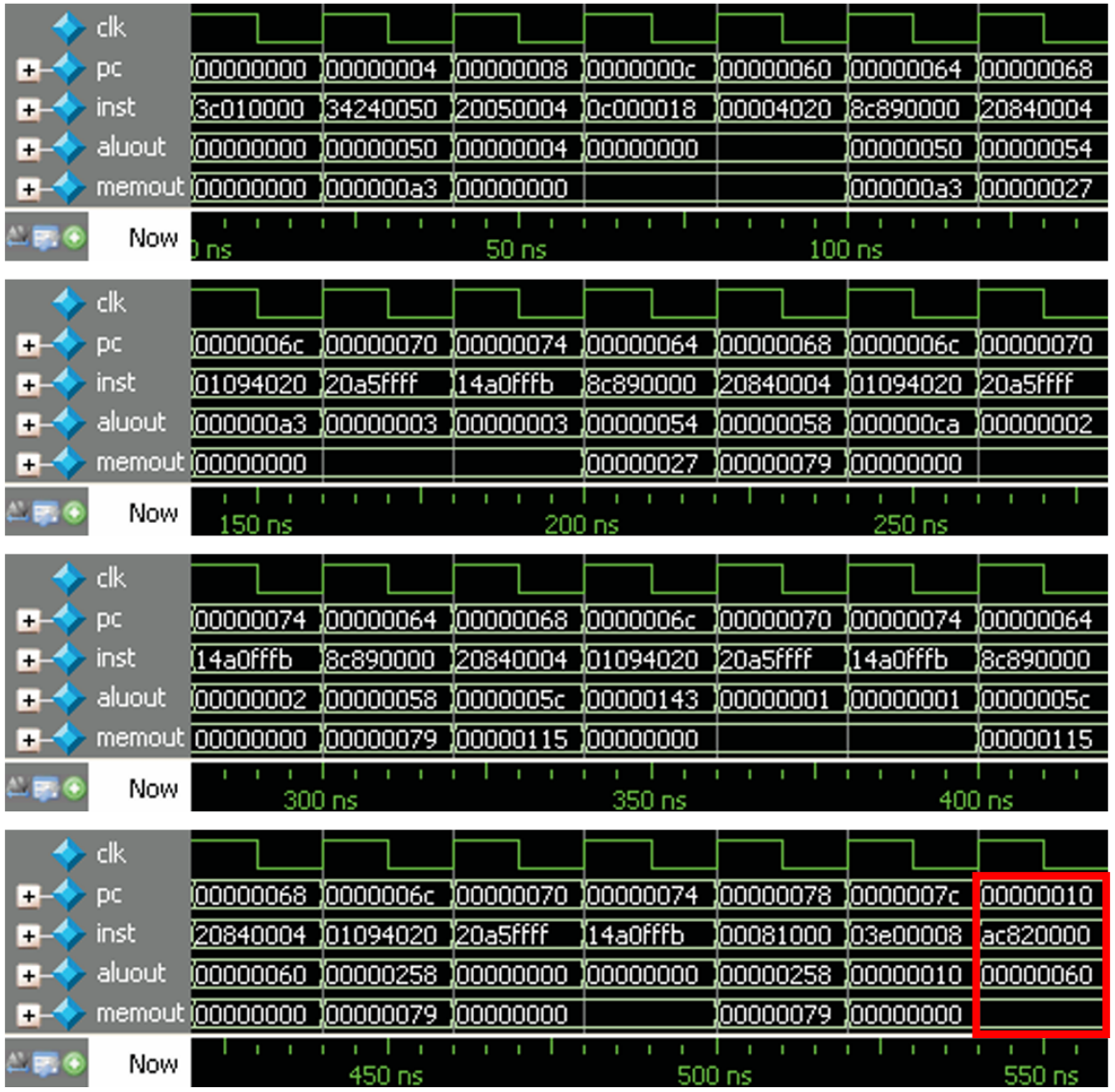
- Test program

```
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```

Simulation

- Test program

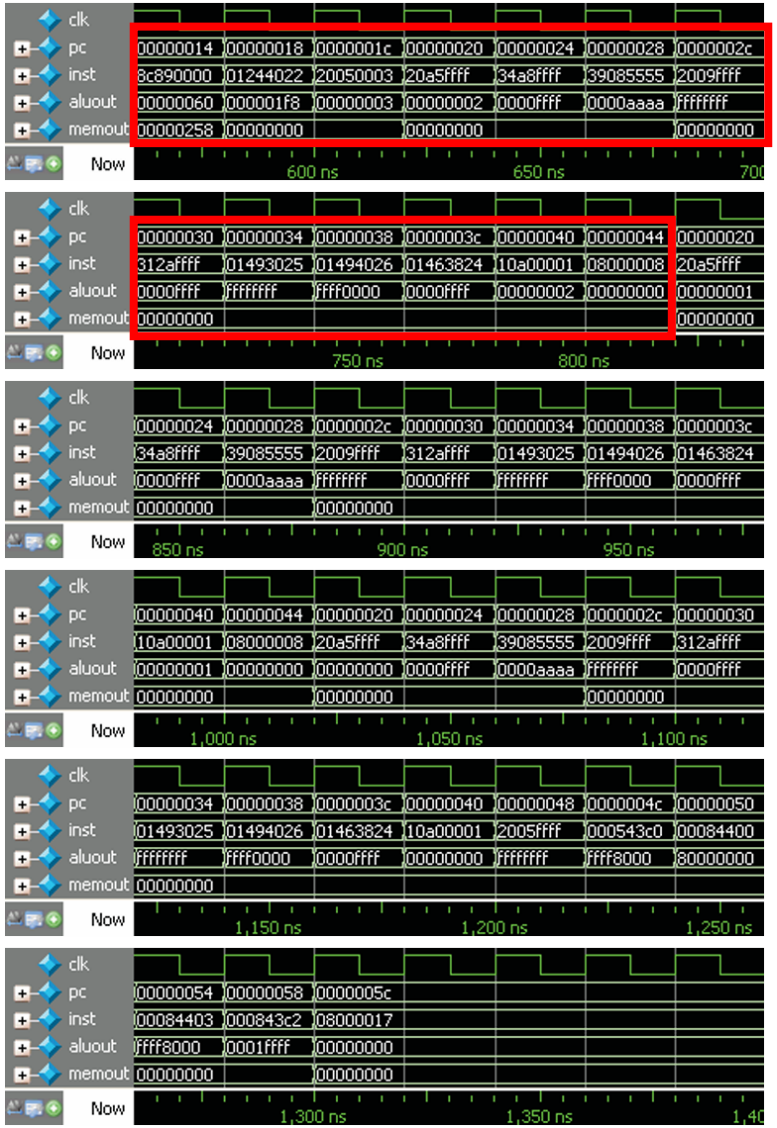
```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
```



Simulation

- Test program

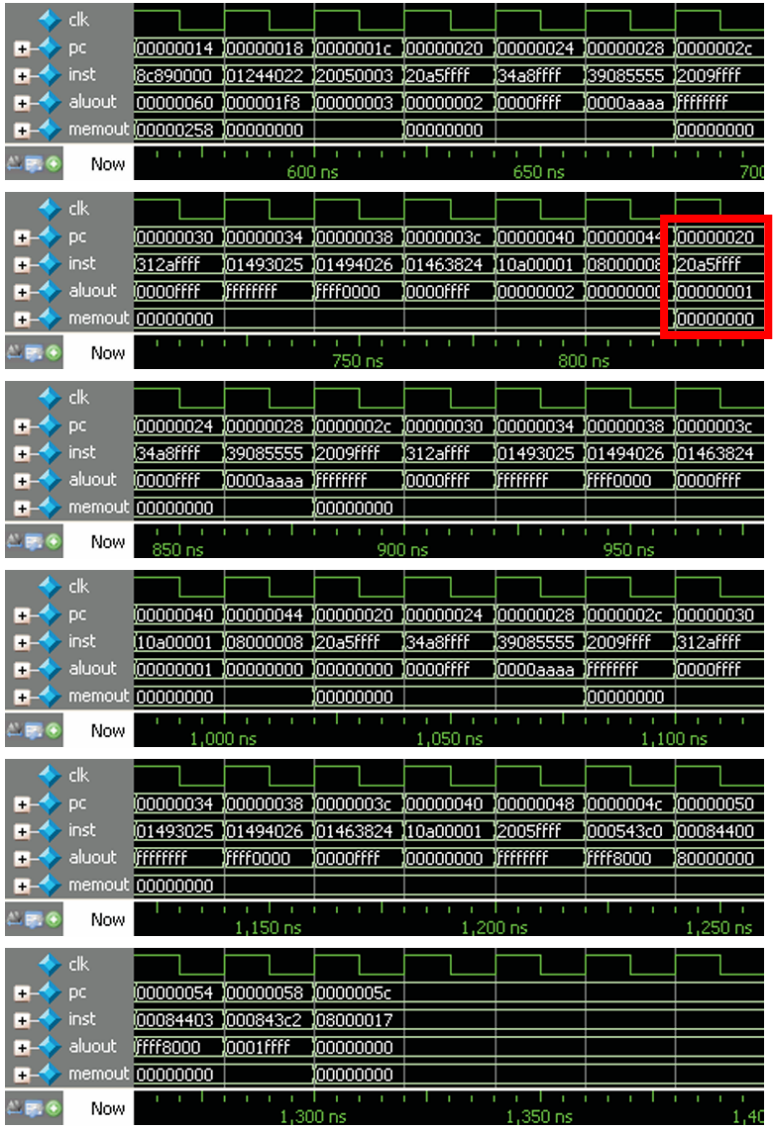
```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

- Test program

```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

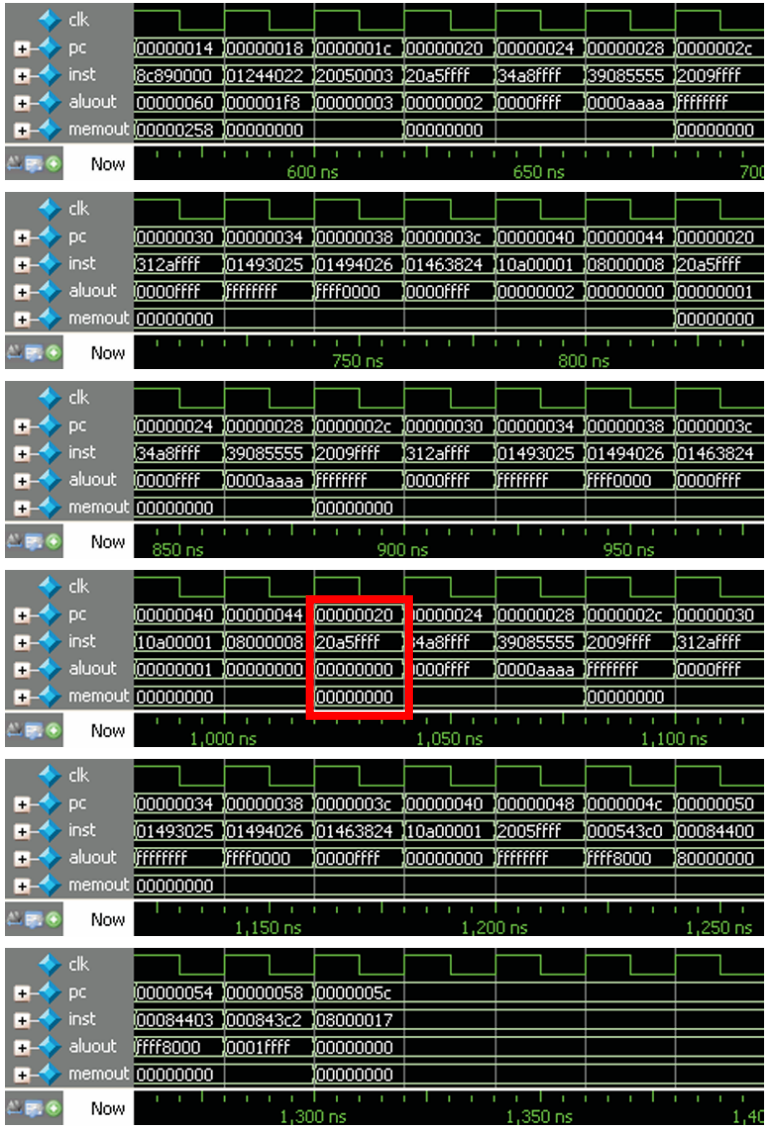
- Test program

```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```

Simulation

- Test program

```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

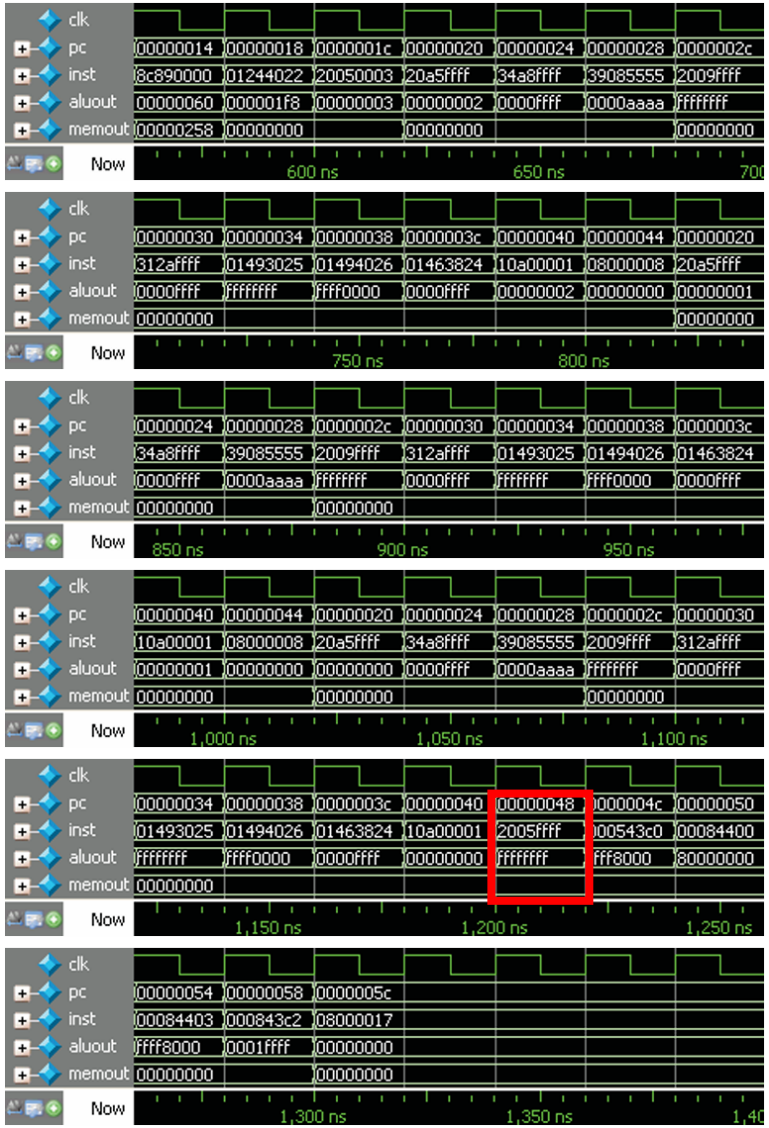
- Test program

```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```

Simulation

- Test program

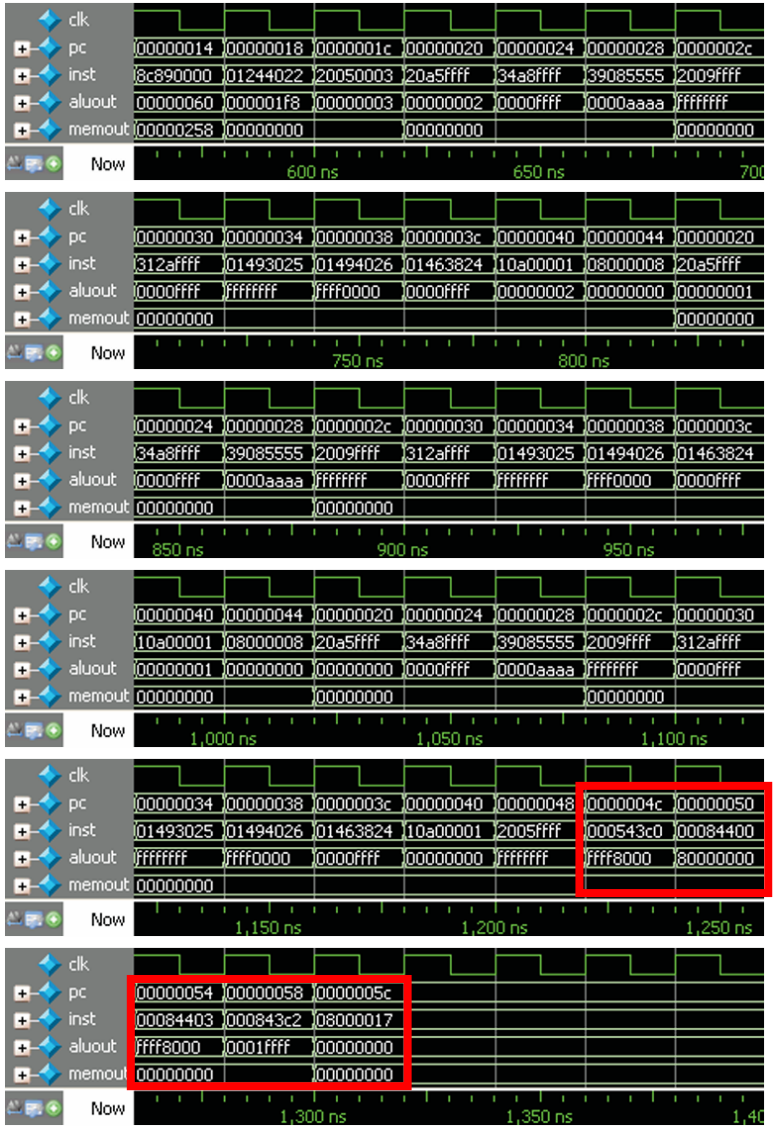
```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

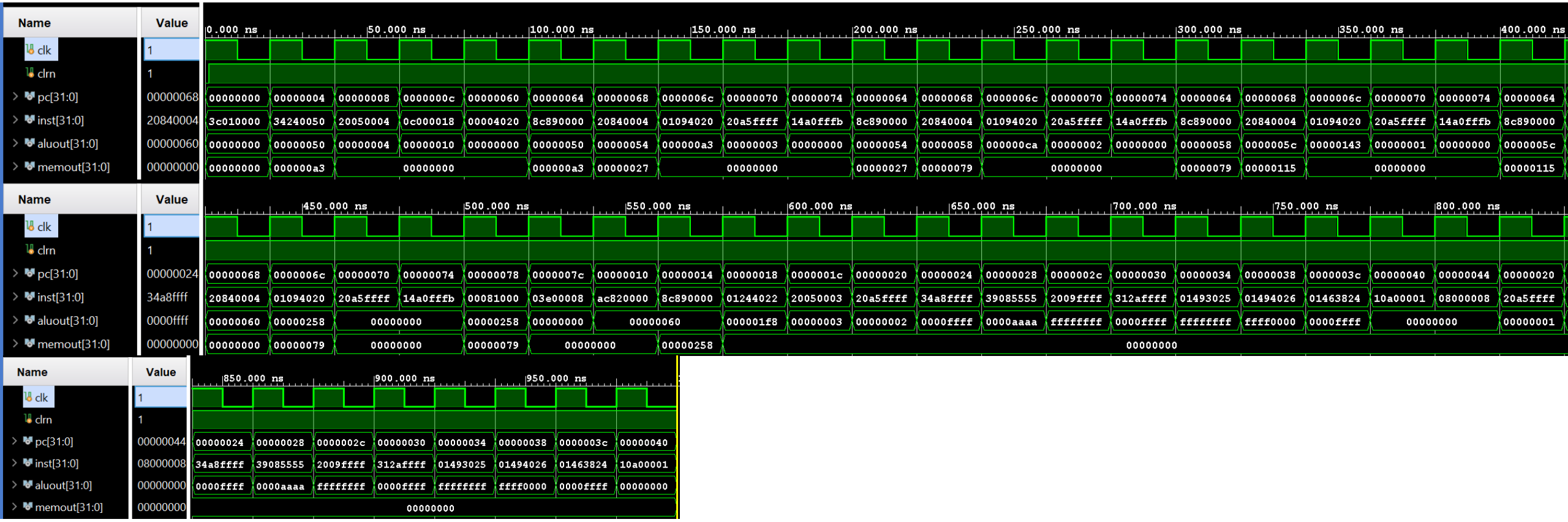
- Test program

```
(pc) label    instruction
(00) main:    lui  $1, 0
(04)          ori  $4, $1, 80
(08)          addi $5, $0,  4
(0c) call:    jal  sum
(10)          sw   $2, 0($4)
(14)          lw   $9, 0($4)
(18)          sub  $8, $9, $4
(1c)          addi $5, $0,  3
(20) loop2:   addi $5, $5, -1
(24)          ori  $8, $5, 0xffff
(28)          xori $8, $8, 0x5555
(2c)          addi $9, $0, -1
(30)          andi $10,$9, 0xffff
(34)          or   $6, $10, $9
(38)          xor  $8, $10, $9
(3c)          and  $7, $10, $6
(40)          beq  $5, $0, shift
(44)          j    loop2
(48) shift:   addi $5, $0, -1
(4c)          sll  $8, $5, 15
(50)          sll  $8, $8, 16
(54)          sra  $8, $8, 16
(58)          srl  $8, $8, 15
(5c) finish:  j    finish
(60) sum:     add  $8, $0, $0
(64) loop:    lw   $9, 0($4)
(68)          addi $4, $4,  4
(6c)          add  $8, $8, $9
(70)          addi $5, $5, -1
(74)          bne  $5, $0, loop
(78)          sll  $2, $8, 0
(7c)          jr   $31
```



Simulation

- Simulation results of the behavioral design.



References

[1] Yamin Li, *Computer Principles and Design in Verilog HDL*, Wiley, 2016.