

SchoolRISCV

<https://github.com/zheInio/schoolRISCV>

Stanislav Zhelnio, 2020

Благодарности

- David Harris & Sarah Harris
- Юрий Панчул
- Александр Романов
- IVA Technologies

Что такое schoolRISCV

- простое процессорное ядро для практического преподавания школьникам основ цифровой схемотехники
- написано на языке Verilog
- реализует подмножество архитектуры RISCV
- вырос из аналогичного проекта schoolMIPS

Микроархитектура

- аппаратная реализация архитектуры в виде схемы
- возможны разные реализации одной архитектуры:
 - одноконтная
 - многотактная
 - конвейерная

Application Software	programs
Operating Systems	device drivers
Architecture	instructions registers
Micro-architecture	datapaths controllers
Logic	adders memories
Digital Circuits	AND gates NOT gates
Analog Circuits	amplifiers filters
Devices	transistors diodes
Physics	electrons

Особенности schoolRISCV

- одноктактная реализация
- нет памяти данных
- словная адресация памяти команд
- 9 инструкций: **add, or, srl, sltu, sub, addi, lui, beq, bne**

...и этого уже достаточно, чтобы посчитать квадратный корень!

Последовательность проектирования

- тракт данных
Data Path
- устройство управления
Control Unit

Спецификация RISC-V

The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document Version 20191213

Editors: Andrew Waterman¹, Krste Asanović^{1,2}
¹SiFive Inc.,

²CS Division, EECS Department, University of California, Berkeley
andrew@sifive.com, krste@berkeley.edu

December 13, 2019

<https://riscv.org/specifications/>

Наборы команд RISC-V: спецификация

Base	Version	Draft Frozen?
RV32I	2.0	Y
RV32E	1.9	N
RV64I	2.0	Y
RV128I	1.7	N
Extension	Version	Frozen?
M	2.0	Y
A	2.0	Y
F	2.0	Y
D	2.0	Y
Q	2.0	Y
L	0.0	N
C	2.0	Y
B	0.0	N
J	0.0	N
T	0.0	N
P	0.1	N
V	0.7	N
N	1.1	N

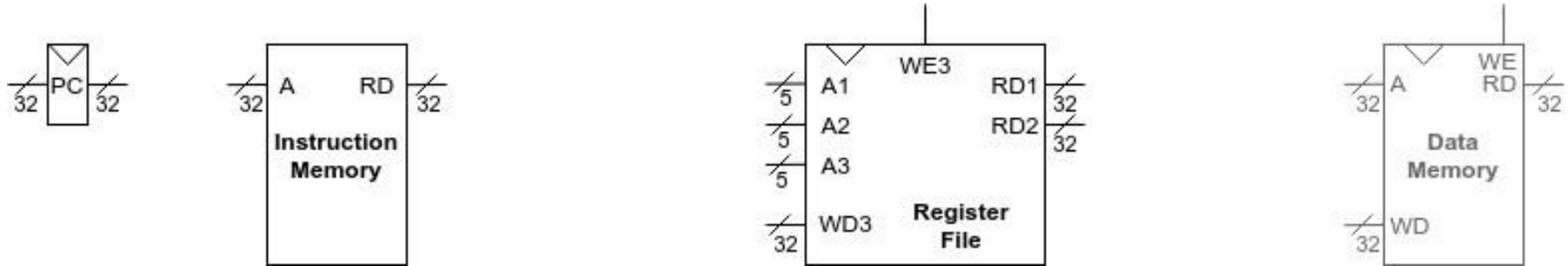
Архитектурное состояние: спецификация

2.1 Programmers' Model for Base Integer ISA

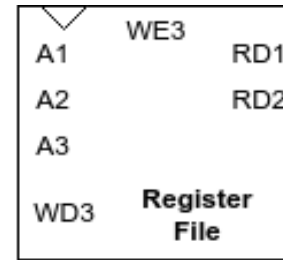
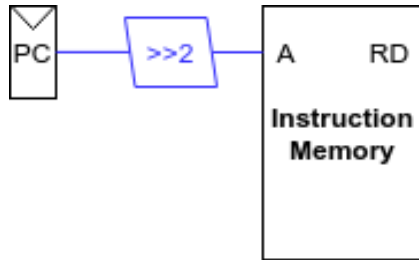
Figure 2.1 shows the unprivileged state for the base integer ISA. For RV32I, the 32 *x* registers are each 32 bits wide, i.e., $XLEN=32$. Register *x0* is hardwired with all bits equal to 0. General purpose registers *x1*–*x31* hold values that various instructions interpret as a collection of Boolean values, or as two's complement signed binary integers or unsigned binary integers.

There is one additional unprivileged register: the program counter *pc* holds the address of the current instruction.

Архитектурное состояние



ADDI: выборка инструкции



ADDI: спецификация

Integer Register-Immediate Instructions

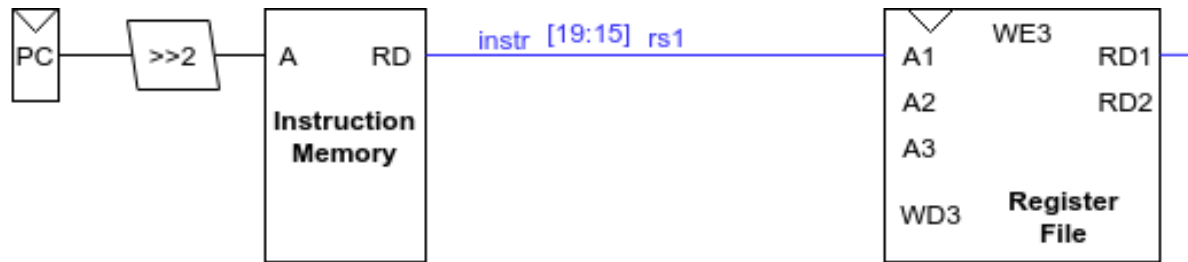
31	20 19	15 14	12 11	7 6	0
imm[11:0]					opcode
12	5	3	5	7	
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM	
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM	

ADDI adds the sign-extended 12-bit immediate to register *rs1*. Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result. `ADDI rd, rs1, 0` is used to implement the `MV rd, rs1` assembler pseudoinstruction.

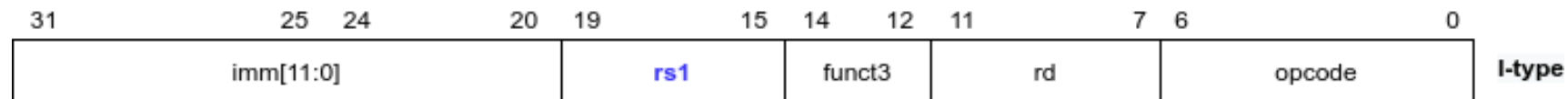
Figure 2.4 shows the immediates produced by each of the base instruction formats, and is labeled to show which instruction bit (`inst[y]`) produces each bit of the immediate value.

31	30	20 19	12	11	10	5	4	1	0	
— inst[31] —					inst[30:25]	inst[24:21]		inst[20]	I-immediate	

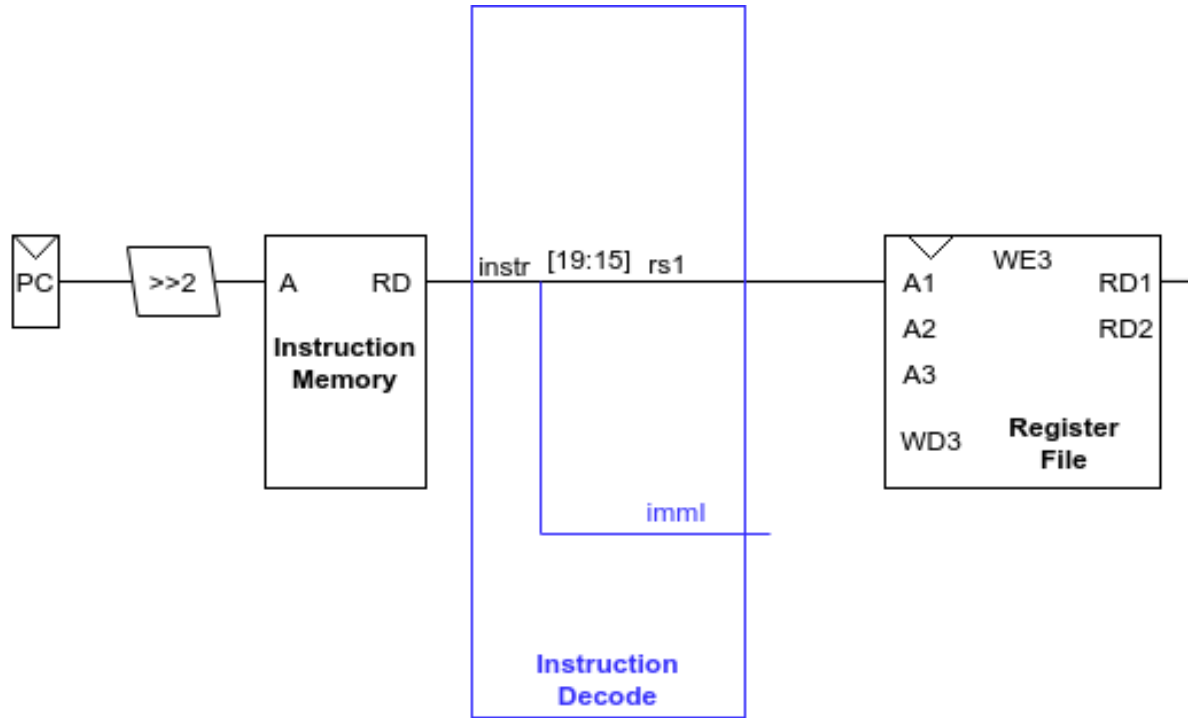
ADDI: считывание операнда из регистрового файла



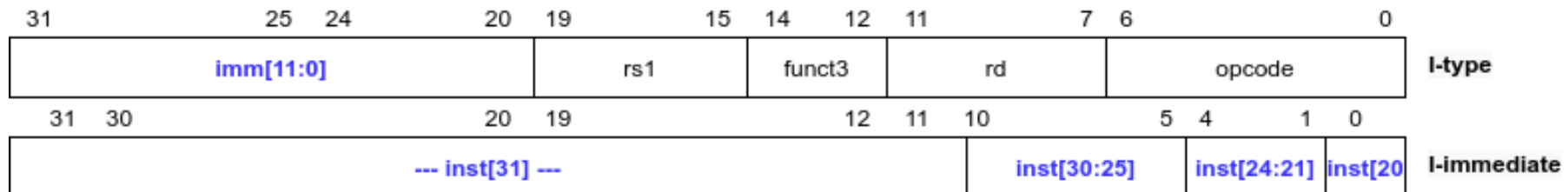
ADDI: I-type, adds the sign-extended 12-bit immediate to register rs1: $rd = rs1 + imm1$



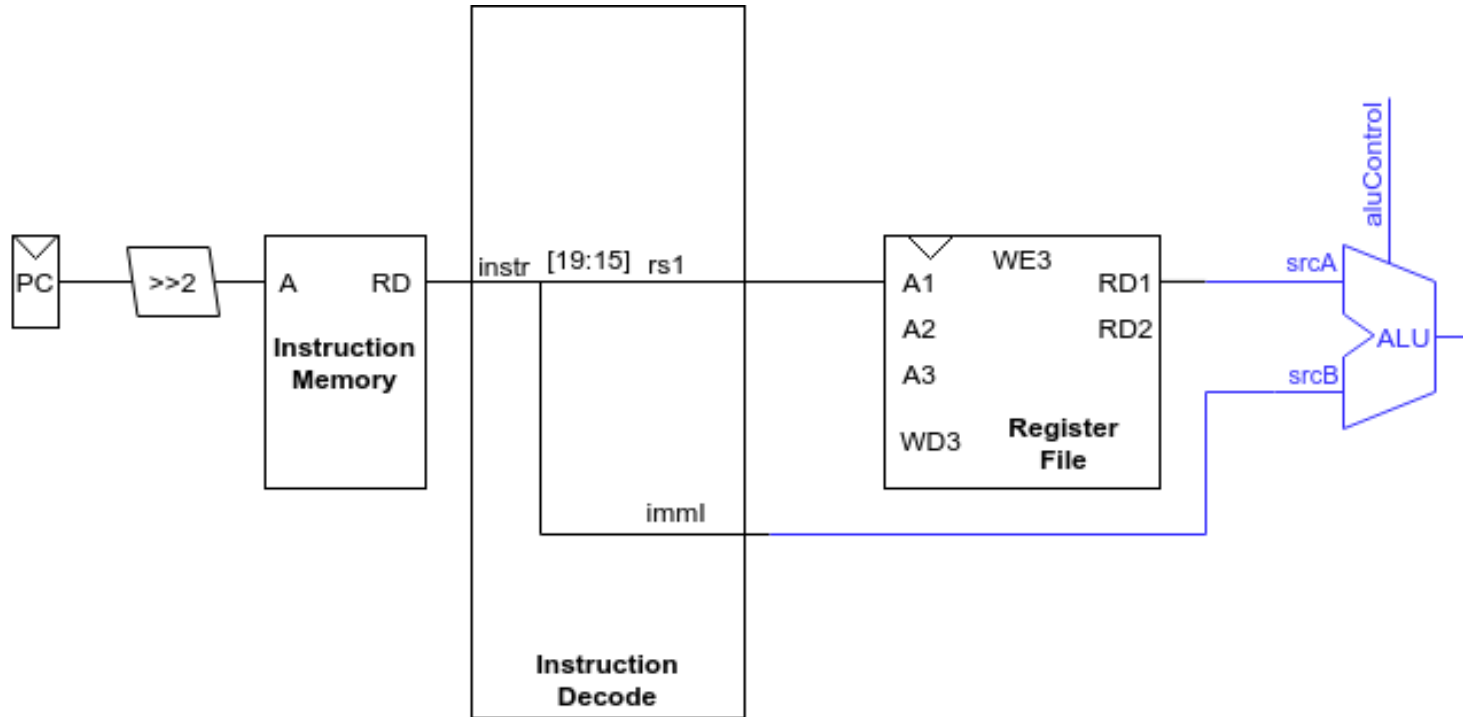
ADDI: декодирование константы из тела инструкции



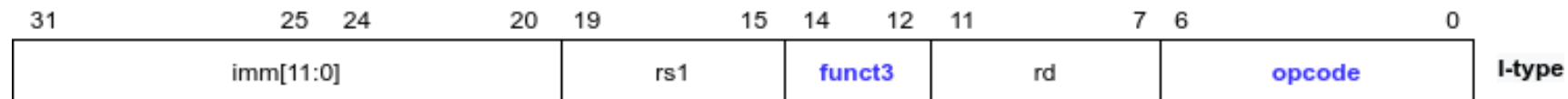
ADDI: I-type, adds the sign-extended 12-bit immediate to register rs1: $rd = rs1 + imml$



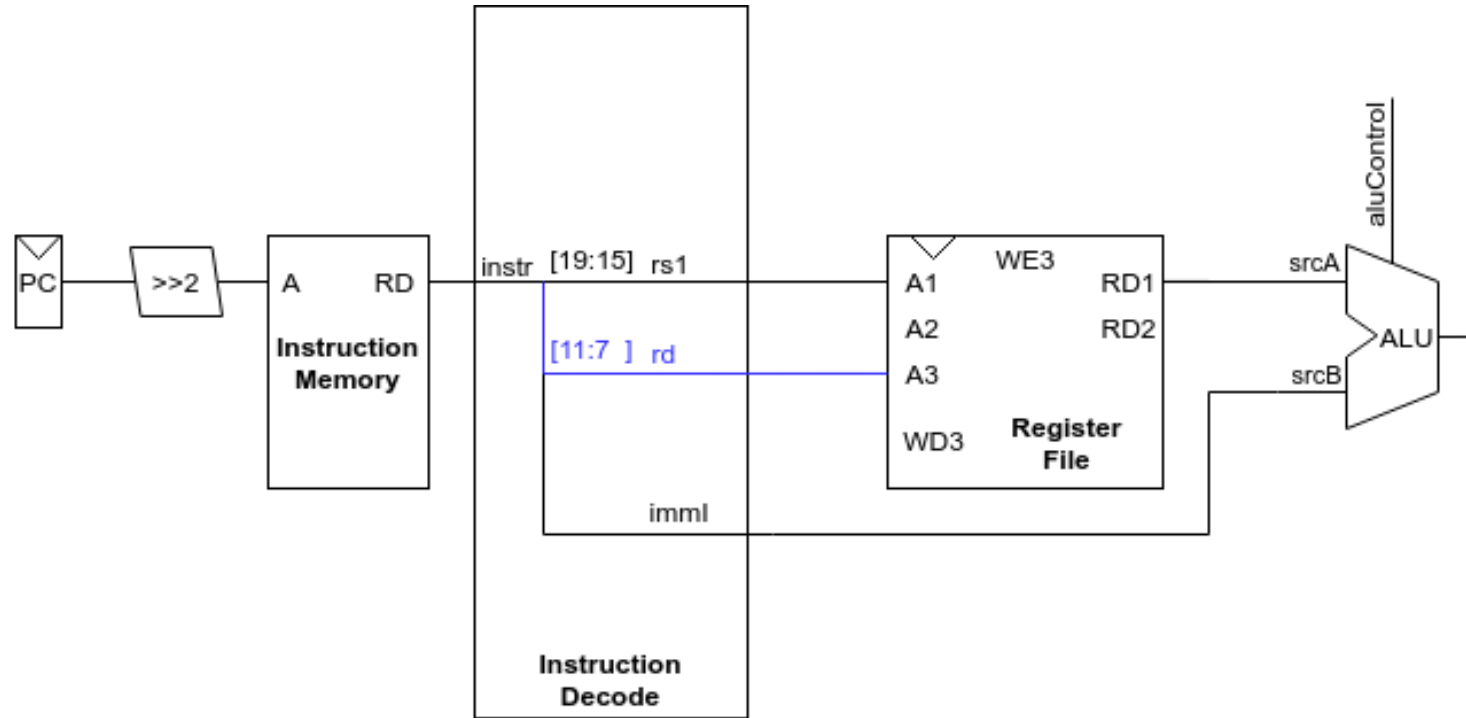
ADDI: вычисление результата арифметической операции



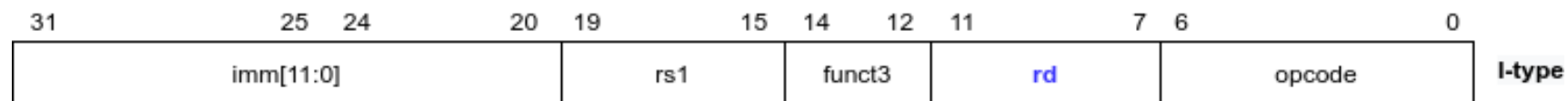
ADDI: I-type, adds the sign-extended 12-bit immediate to register rs1: $rd = rs1 + imm1$



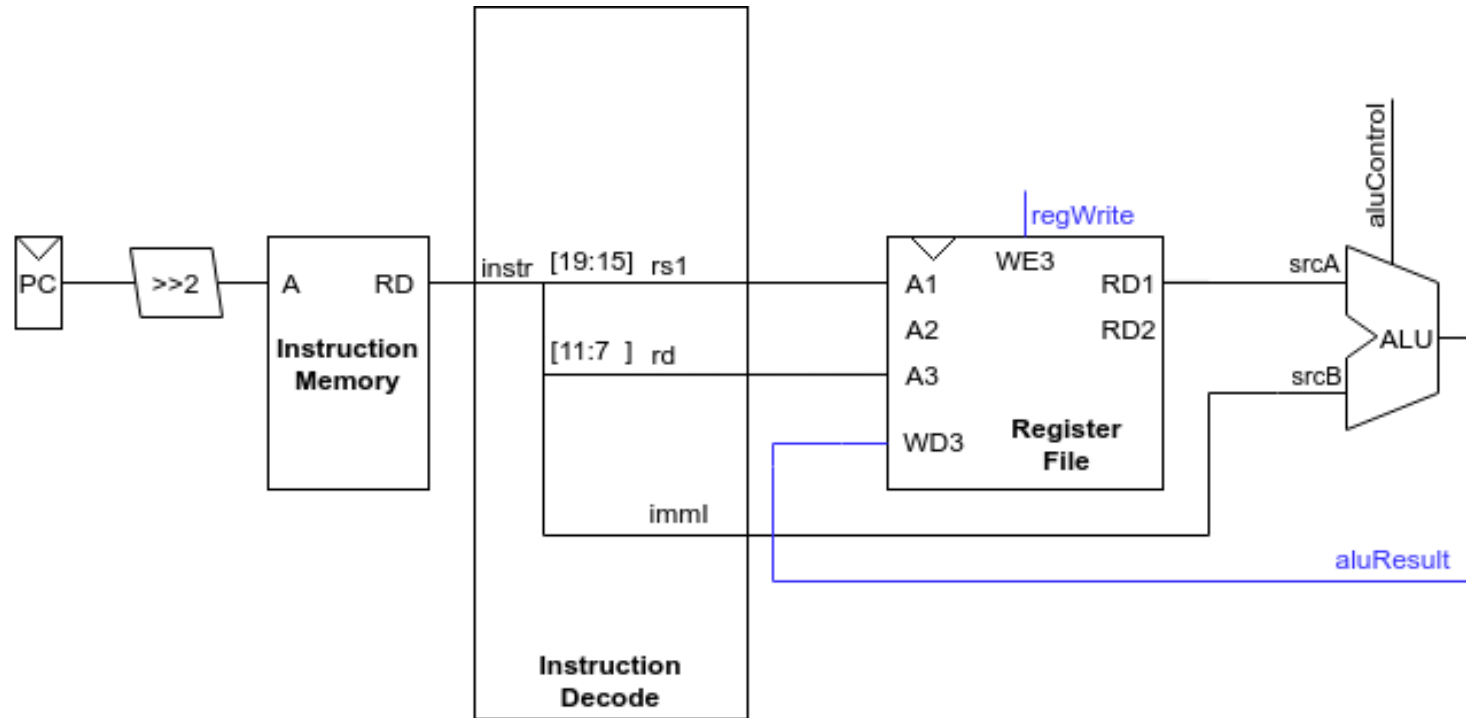
ADDI: декодирование регистра назначения



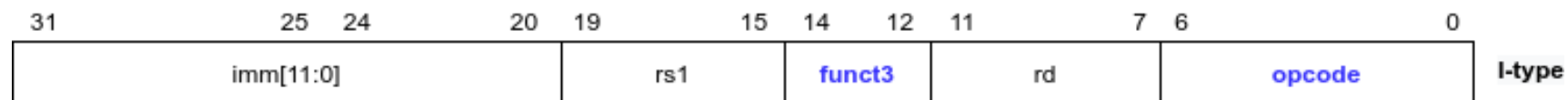
ADDI: I-type, adds the sign-extended 12-bit immediate to register `rs1`: $rd = rs1 + imm1$



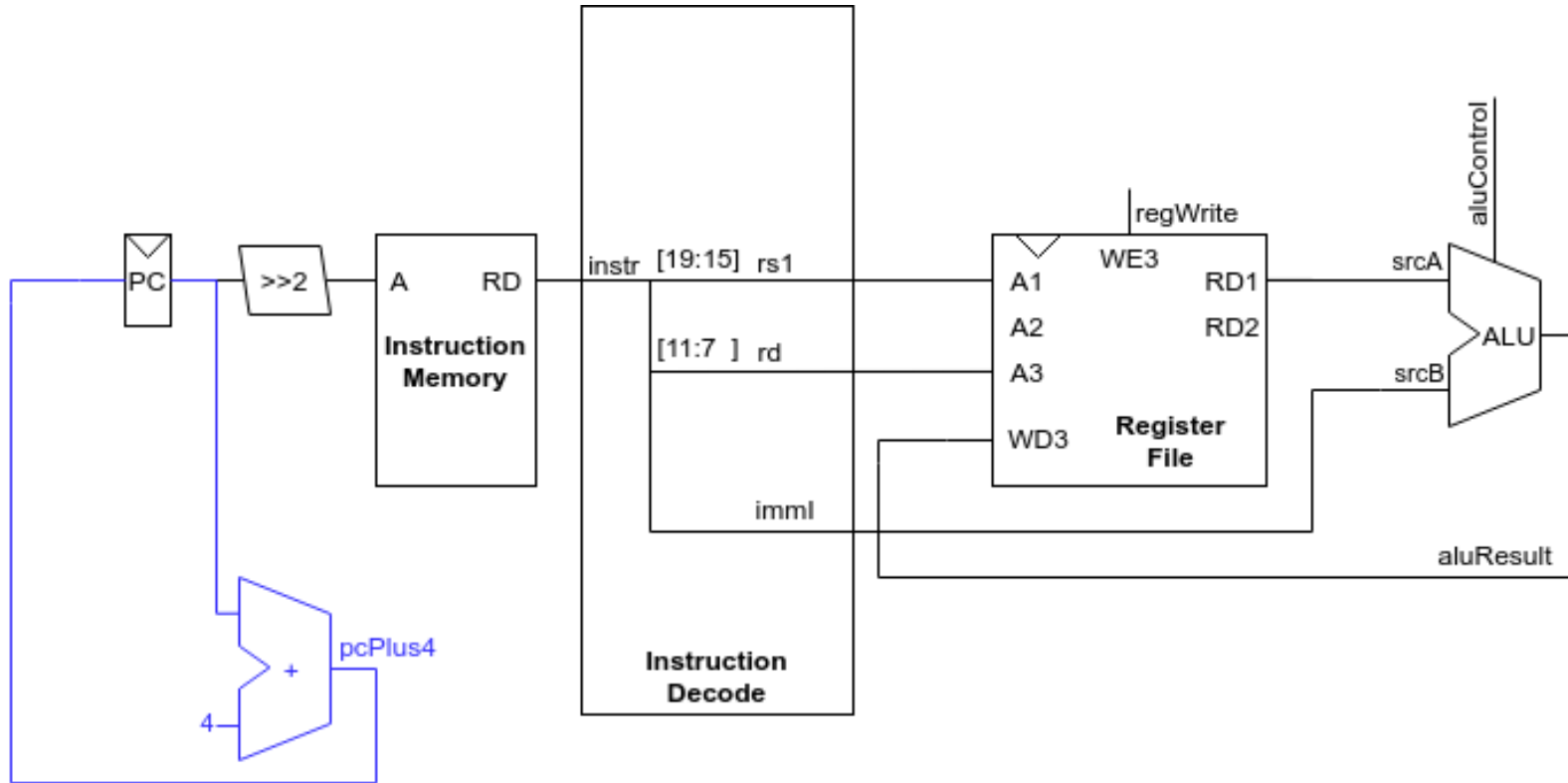
ADDI: запись результата в регистр назначения



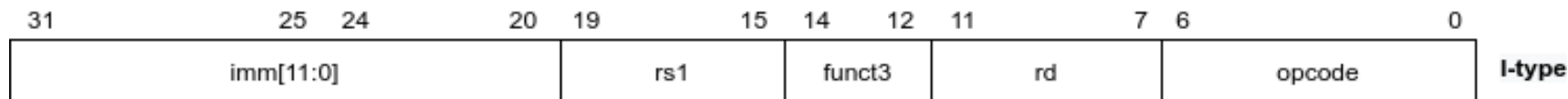
ADDI: I-type, adds the sign-extended 12-bit immediate to register rs1: $rd = rs1 + imm1$



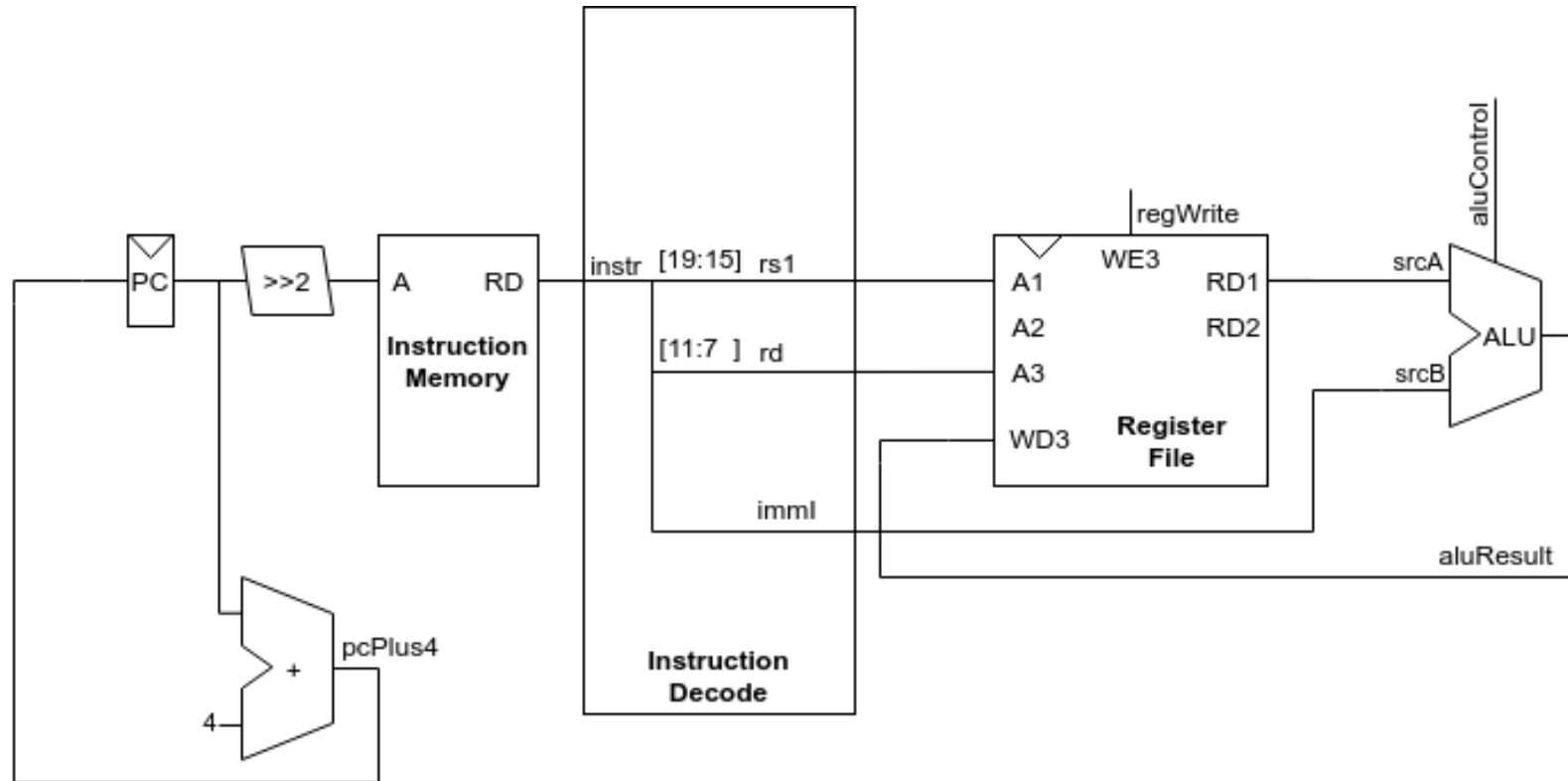
ADDI: вычисление адреса следующей инструкции



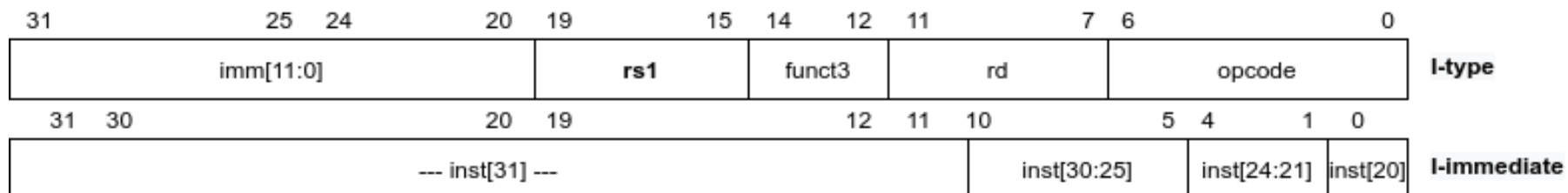
ADDI: I-type, adds the sign-extended 12-bit immediate to register rs1: $rd = rs1 + imm1$



ADDI: итоговая схема



ADDI: I-type, adds the sign-extended 12-bit immediate to register rs1: $rd = rs1 + imm1$



ADD: спецификация

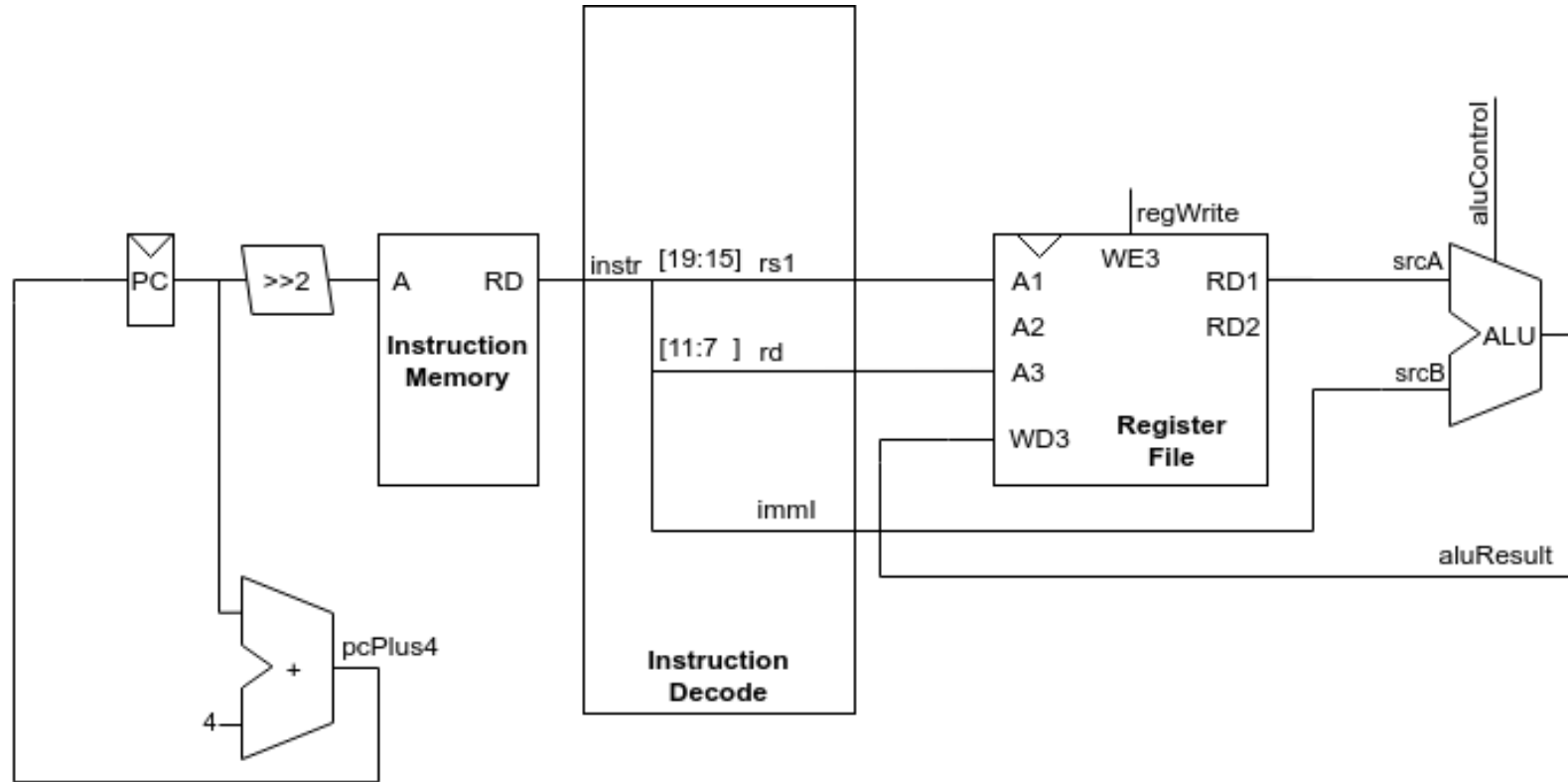
Integer Register-Register Operations

RV32I defines several arithmetic R-type operations. All operations read the *rs1* and *rs2* registers as source operands and write the result into register *rd*. The *funct7* and *funct3* fields select the type of operation.

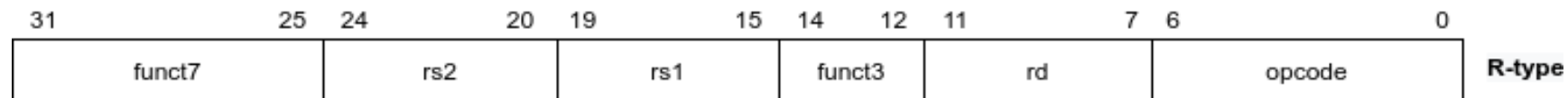
31	25 24	20 19	15 14	12 11	7 6	0
funct7	rs2	rs1	funct3	rd	opcode	
7	5	5	3	5	7	
0000000	src2	src1	ADD/SLT/SLTU	dest	OP	
0000000	src2	src1	AND/OR/XOR	dest	OP	
0000000	src2	src1	SLL/SRL	dest	OP	
0100000	src2	src1	SUB/SRA	dest	OP	

ADD performs the addition of *rs1* and *rs2*. SUB performs the subtraction of *rs2* from *rs1*. Overflows are ignored and the low XLEN bits of results are written to the destination *rd*. SLT and SLTU perform signed and unsigned compares respectively, writing 1 to *rd* if *rs1* < *rs2*, 0 otherwise. Note,

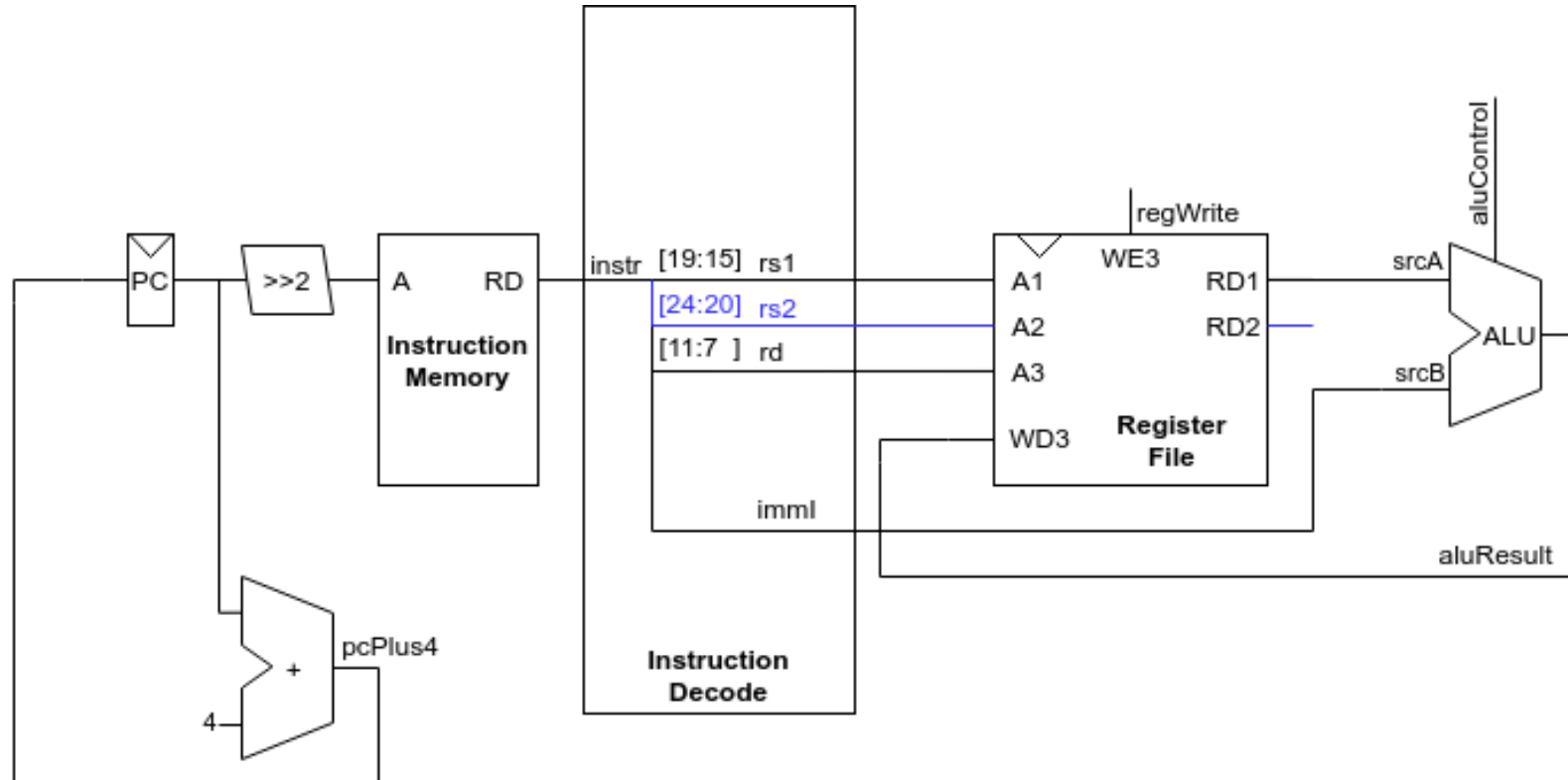
ADD: выборка операнда rs1



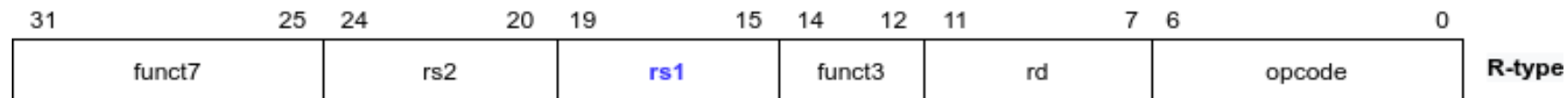
ADD: R-type, performs the addition of rs1 and rs2: $rd = rs1 + rs2$



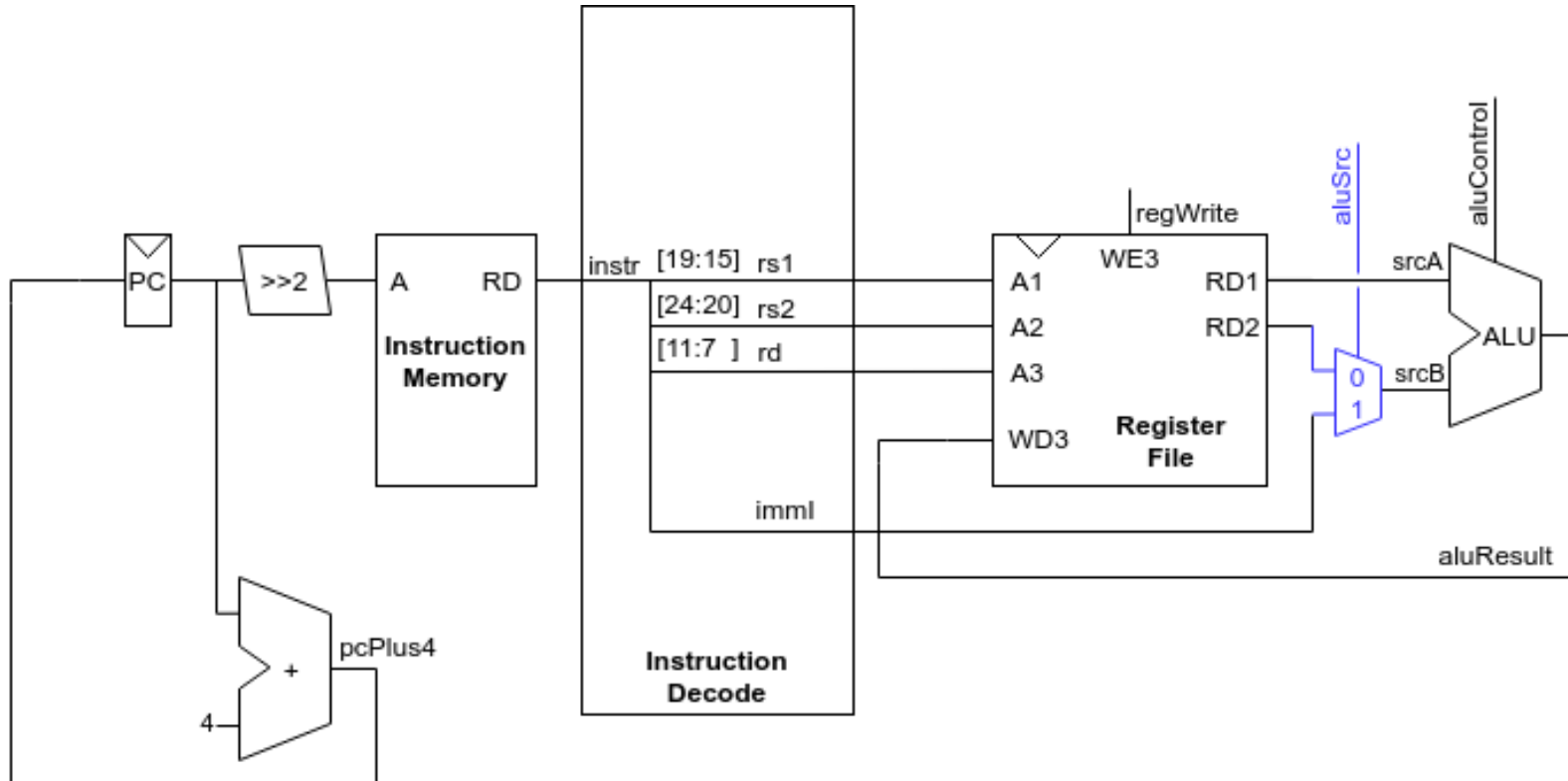
ADD: выборка операнда rs2



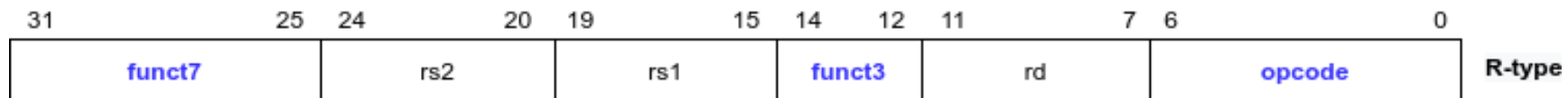
ADD: R-type, performs the addition of rs1 and rs2: $rd = rs1 + rs2$



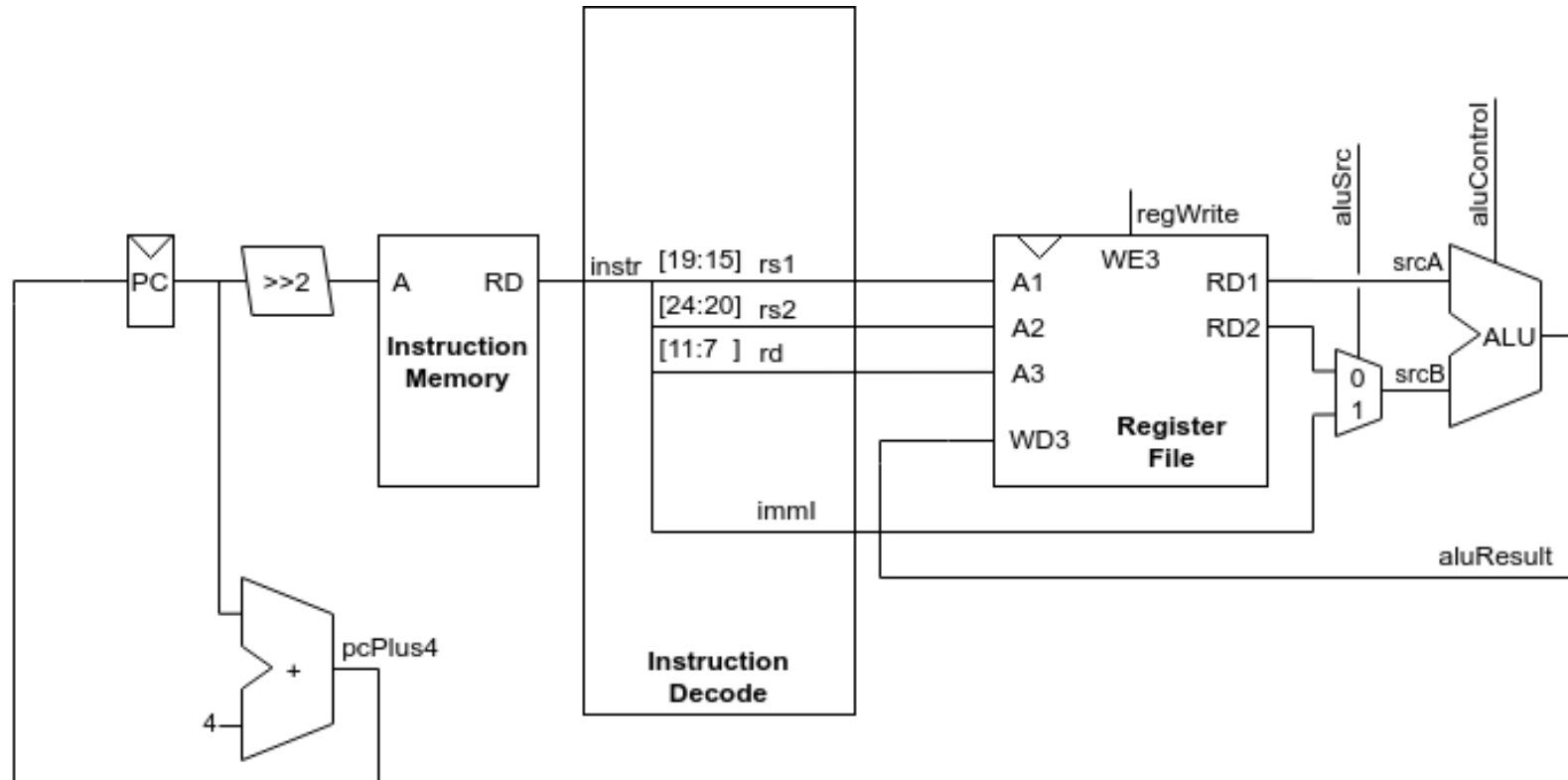
ADD: передача второго операнда в АЛУ



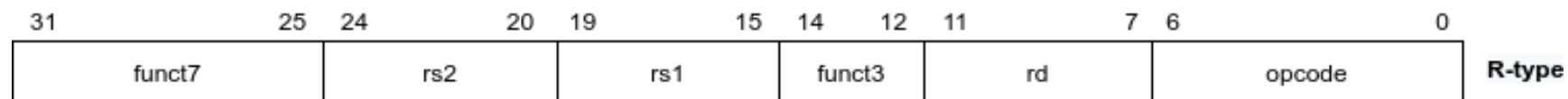
ADD: R-type, performs the addition of rs1 and rs2: $rd = rs1 + rs2$



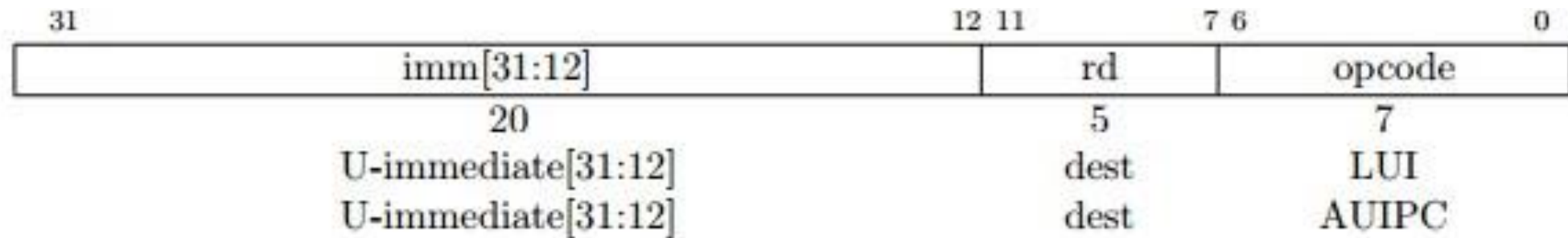
ADD: итоговая схема



ADD: R-type, performs the addition of rs1 and rs2: $rd = rs1 + rs2$



LUI: спецификация

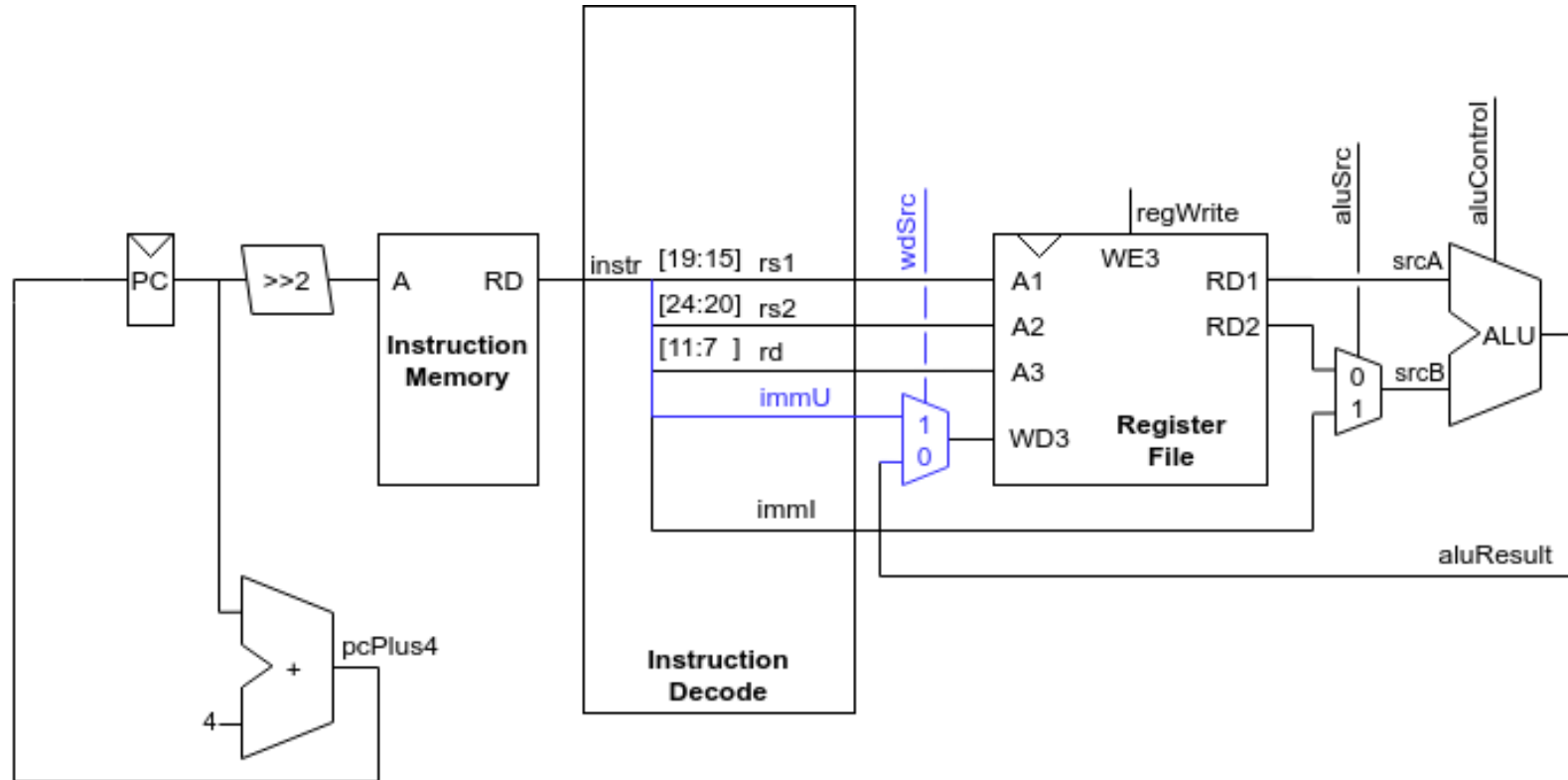


LUI (load upper immediate) is used to build 32-bit constants and uses the U-type format. LUI places the U-immediate value in the top 20 bits of the destination register *rd*, filling in the lowest 12 bits with zeros.

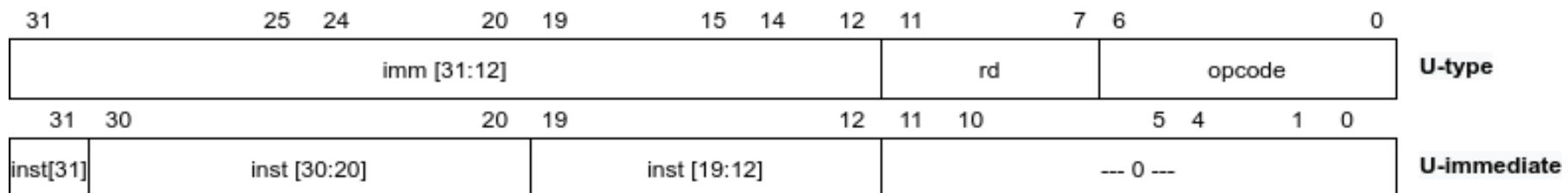
...



LUI: декодирование и передача константы



LUI: U-type, load upper immediate: **rd = immU**



BEQ: спецификация

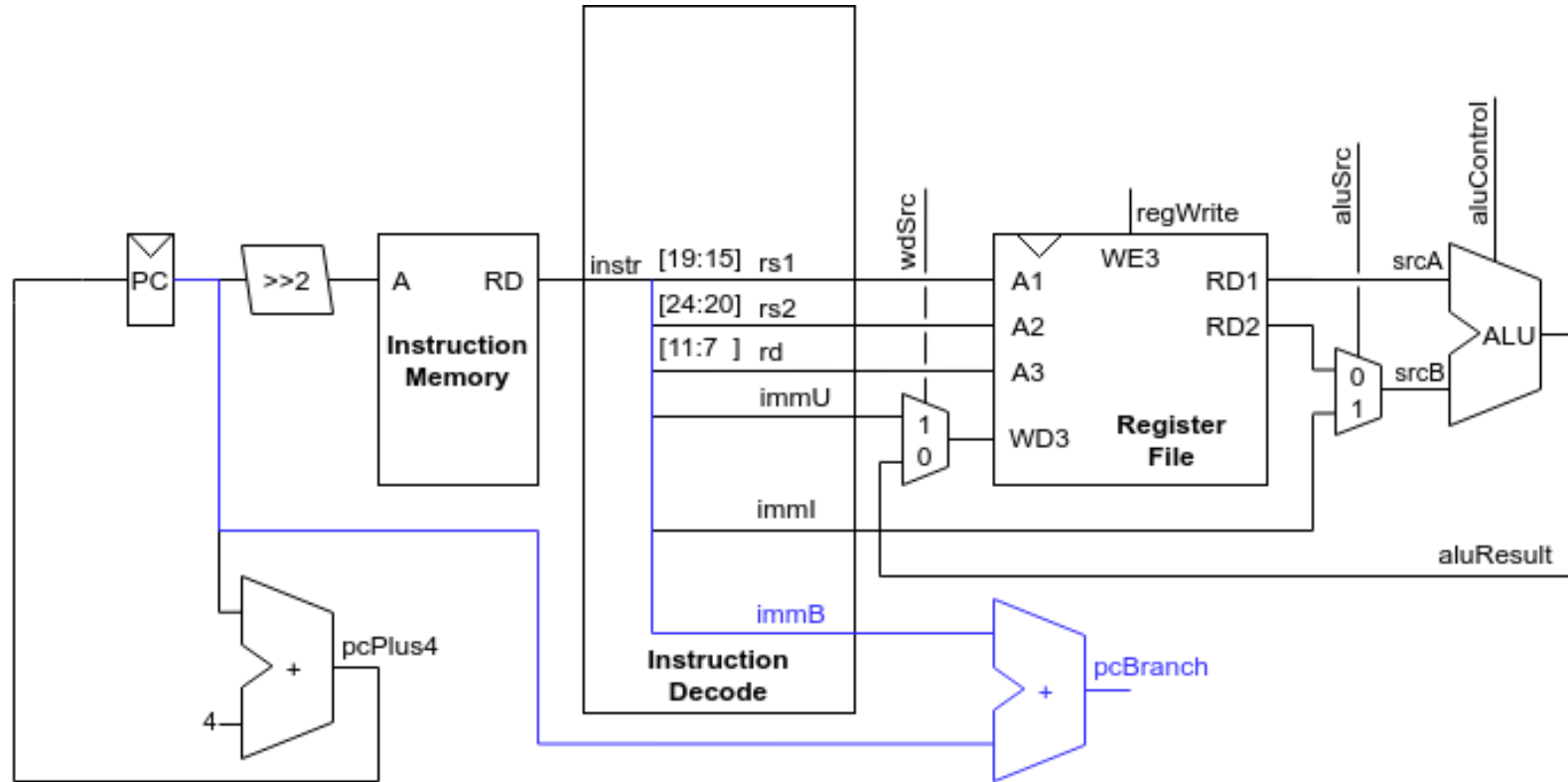
Conditional Branches

All branch instructions use the B-type instruction format. The 12-bit B-immediate encodes signed offsets in multiples of 2 bytes. The offset is sign-extended and added to the address of the branch instruction to give the target address. The conditional branch range is ± 4 KiB.

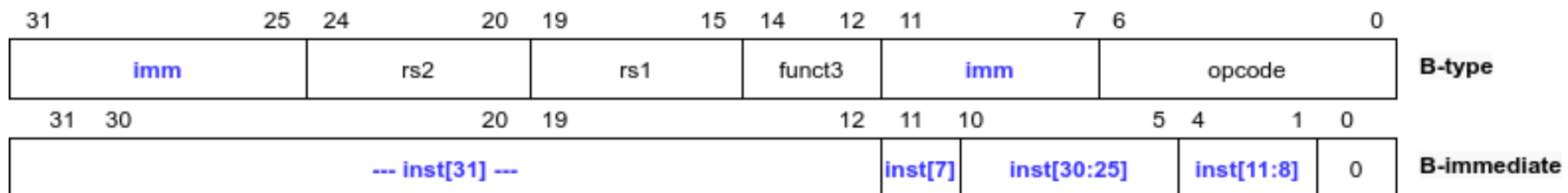
31	30	25 24	20 19	15 14	12 11	8	7	6	0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode		
1	6	5	5	3	4	1	7		
offset[12 10:5]		src2	src1	BEQ/BNE	offset[11 4:1]		BRANCH		
offset[12 10:5]		src2	src1	BLT[U]	offset[11 4:1]		BRANCH		
offset[12 10:5]		src2	src1	BGE[U]	offset[11 4:1]		BRANCH		

Branch instructions compare two registers. BEQ and BNE take the branch if registers *rs1* and *rs2* are equal or unequal respectively. BLT and BLTU take the branch if *rs1* is less than *rs2*, using signed and unsigned comparison respectively. BGE and BGEU take the branch if *rs1* is greater than or equal to *rs2*, using signed and unsigned comparison respectively. Note, BGT, BGTU, BLE, and BLEU can be synthesized by reversing the operands to BLT, BLTU, BGE, and BGEU, respectively.

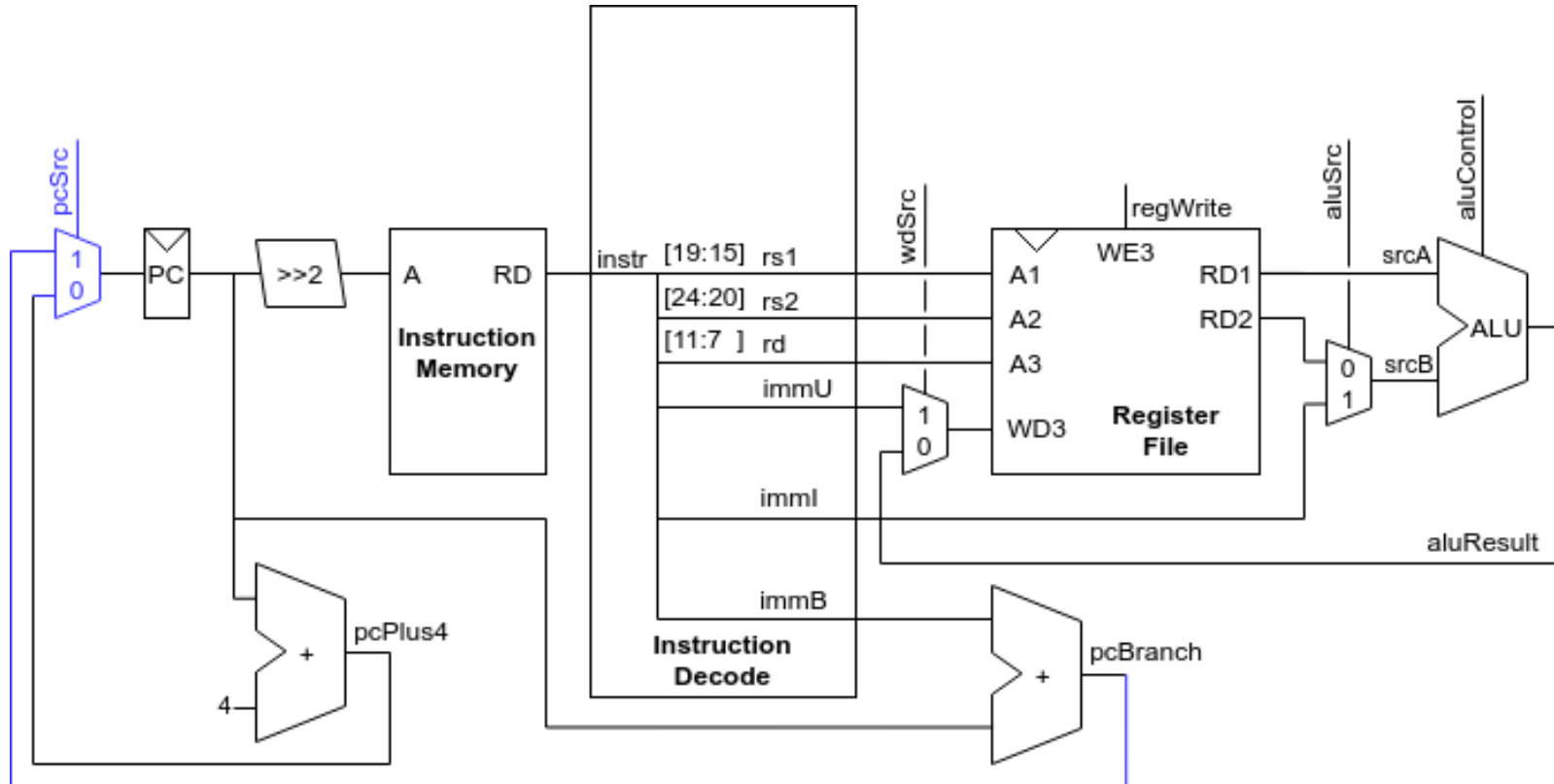
BEQ: вычисление адреса условного перехода



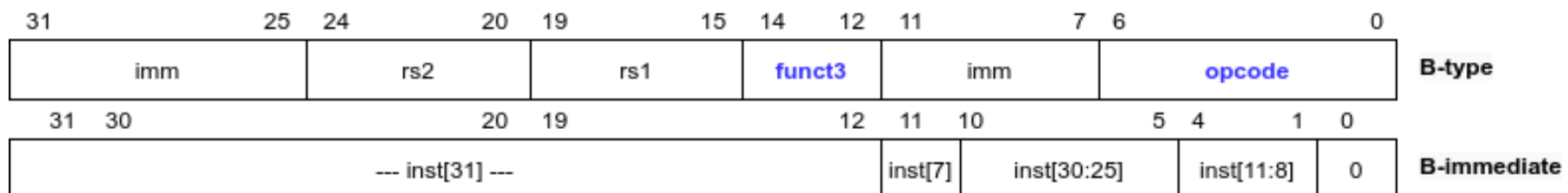
BEQ: B-type, branch on equal: $\text{if } (rs1 == rs2) \text{ PC} = \text{PC} + \text{immB}$



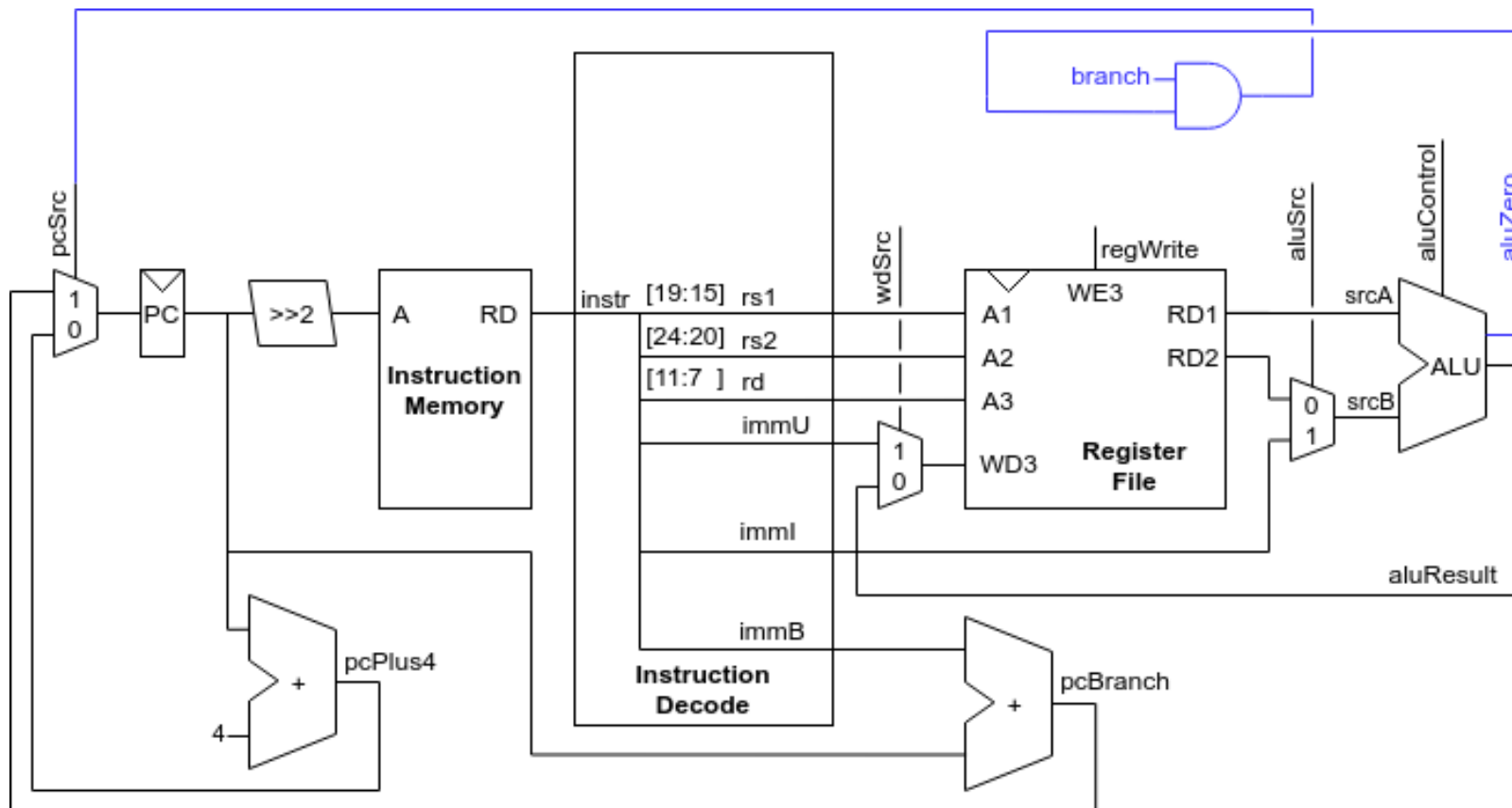
BEQ: выбор адреса



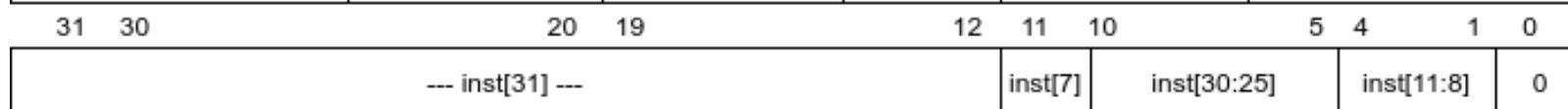
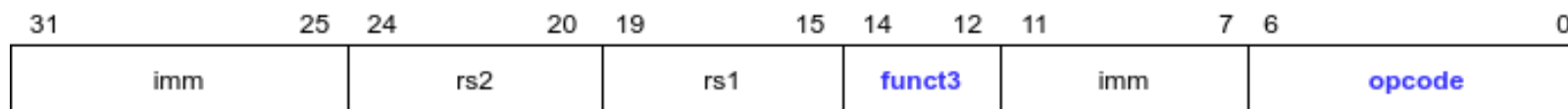
BEQ: B-type, branch on equal: $\text{if } (rs1 == rs2) PC = PC + immB$



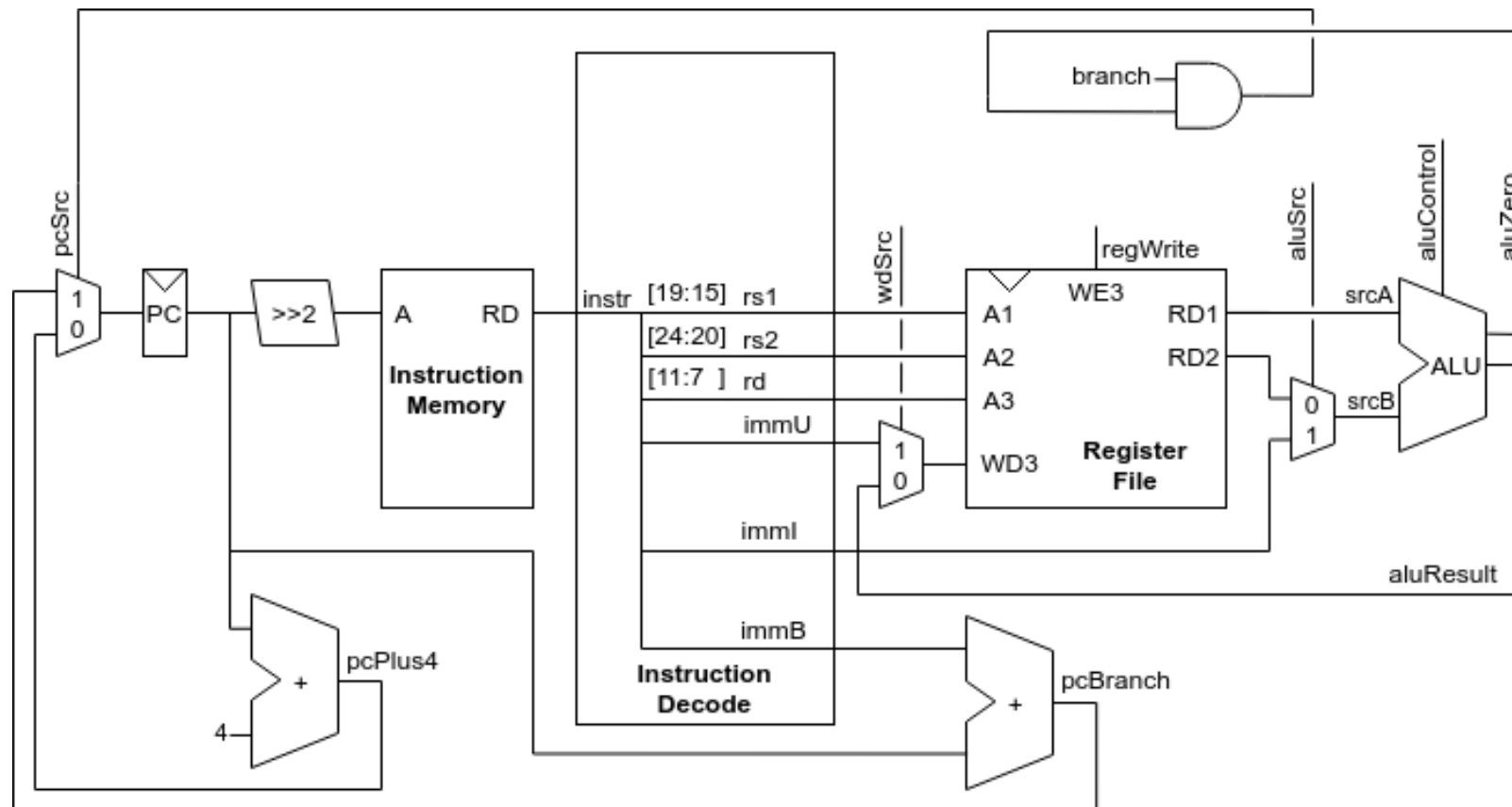
BEQ: определение необходимости перехода



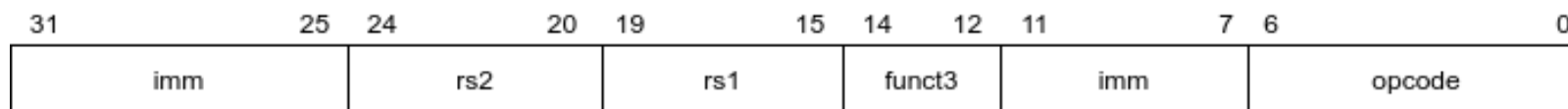
BEQ: B-type, branch on equal: $\text{if } (rs1 == rs2) \text{ PC} = \text{PC} + \text{immB}$



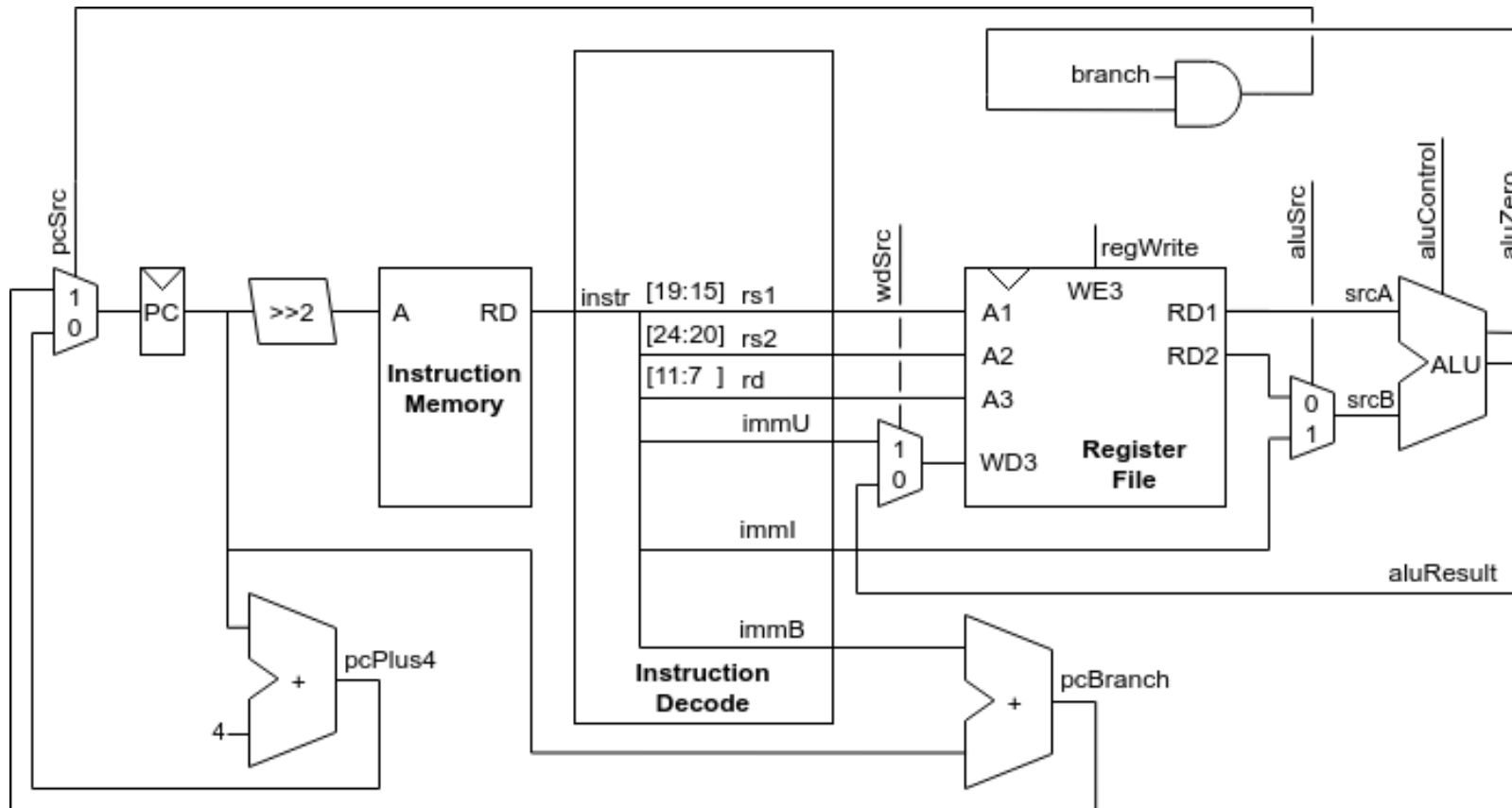
BEQ: итоговая схема



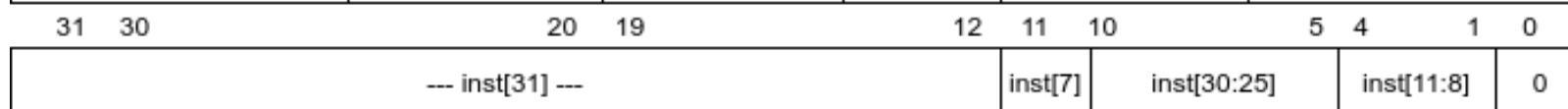
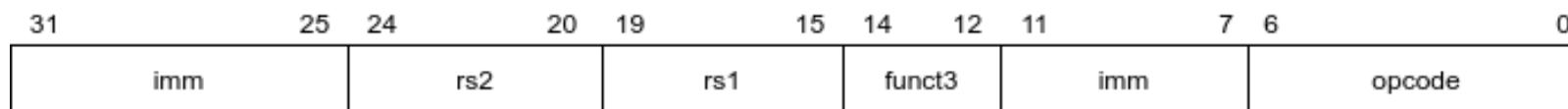
BEQ: B-type, branch on equal: if (rs1 == rs2) PC = PC + immB



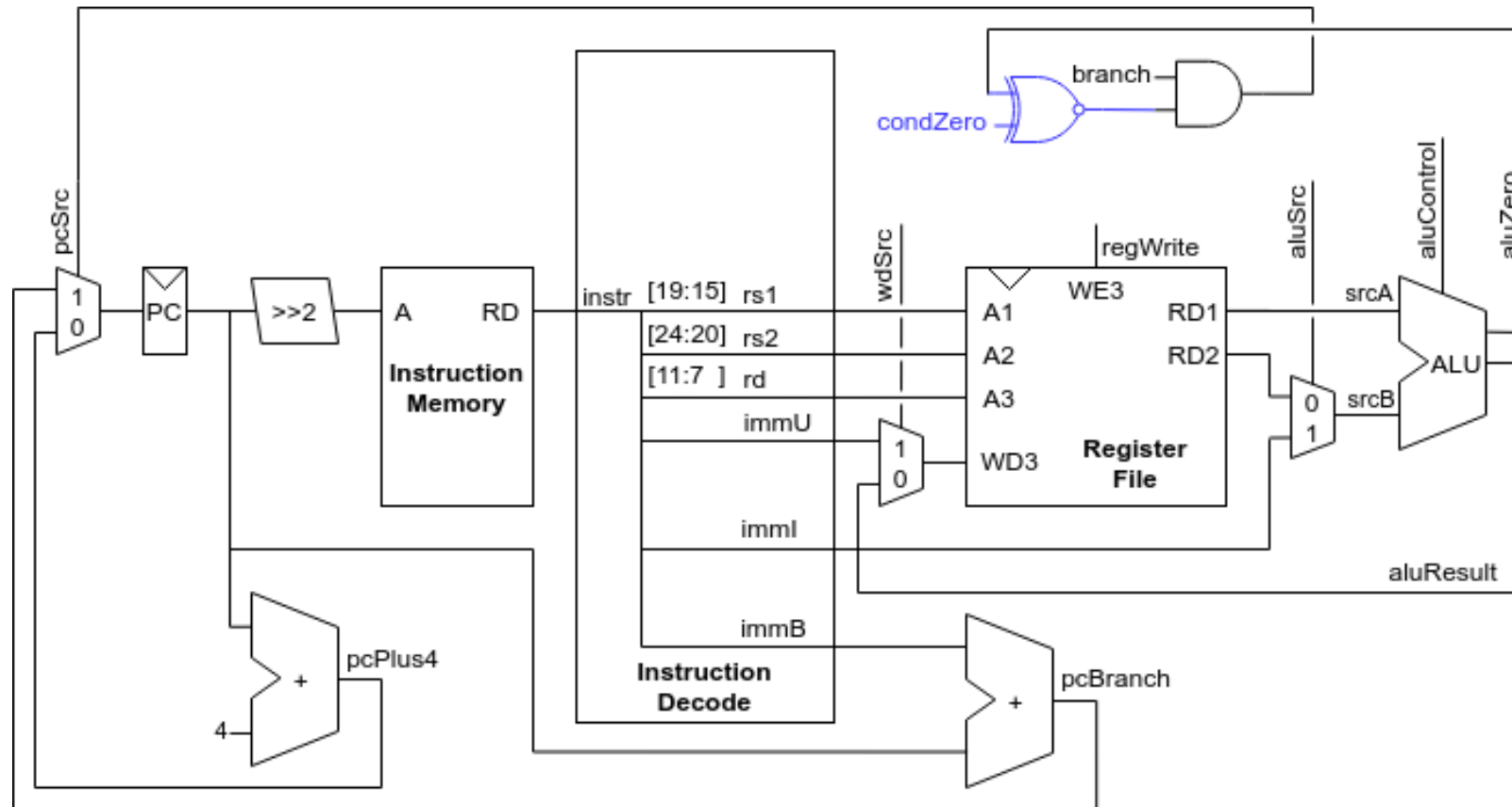
BNE: BEQ наоборот



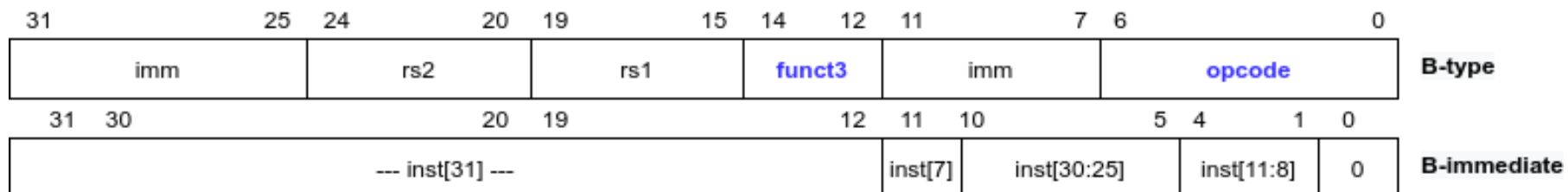
BNE: B-type, branch on non equal: if ($rs1 \neq rs2$) $PC = PC + immB$



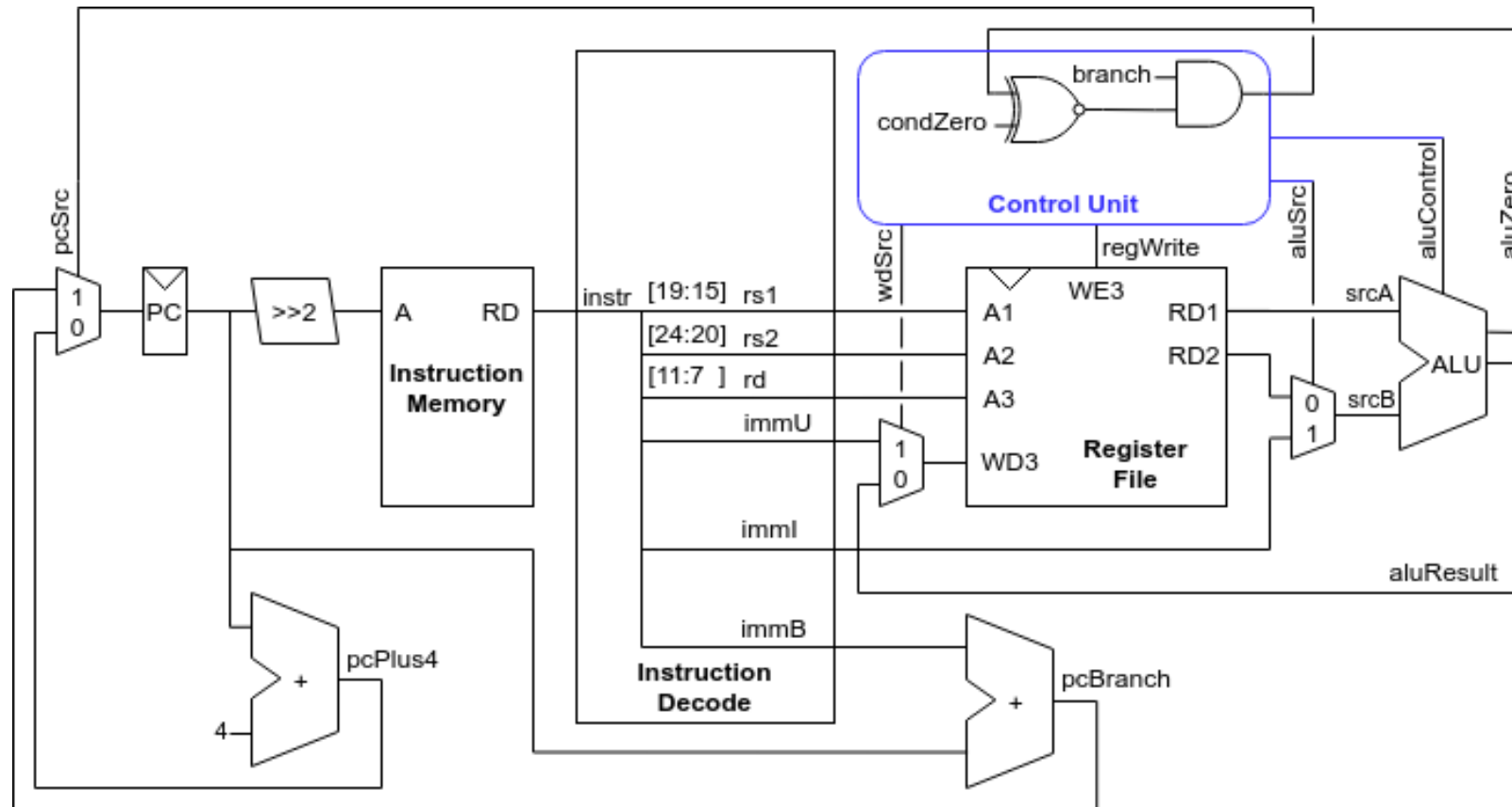
ВНЕ: новая цепь управления



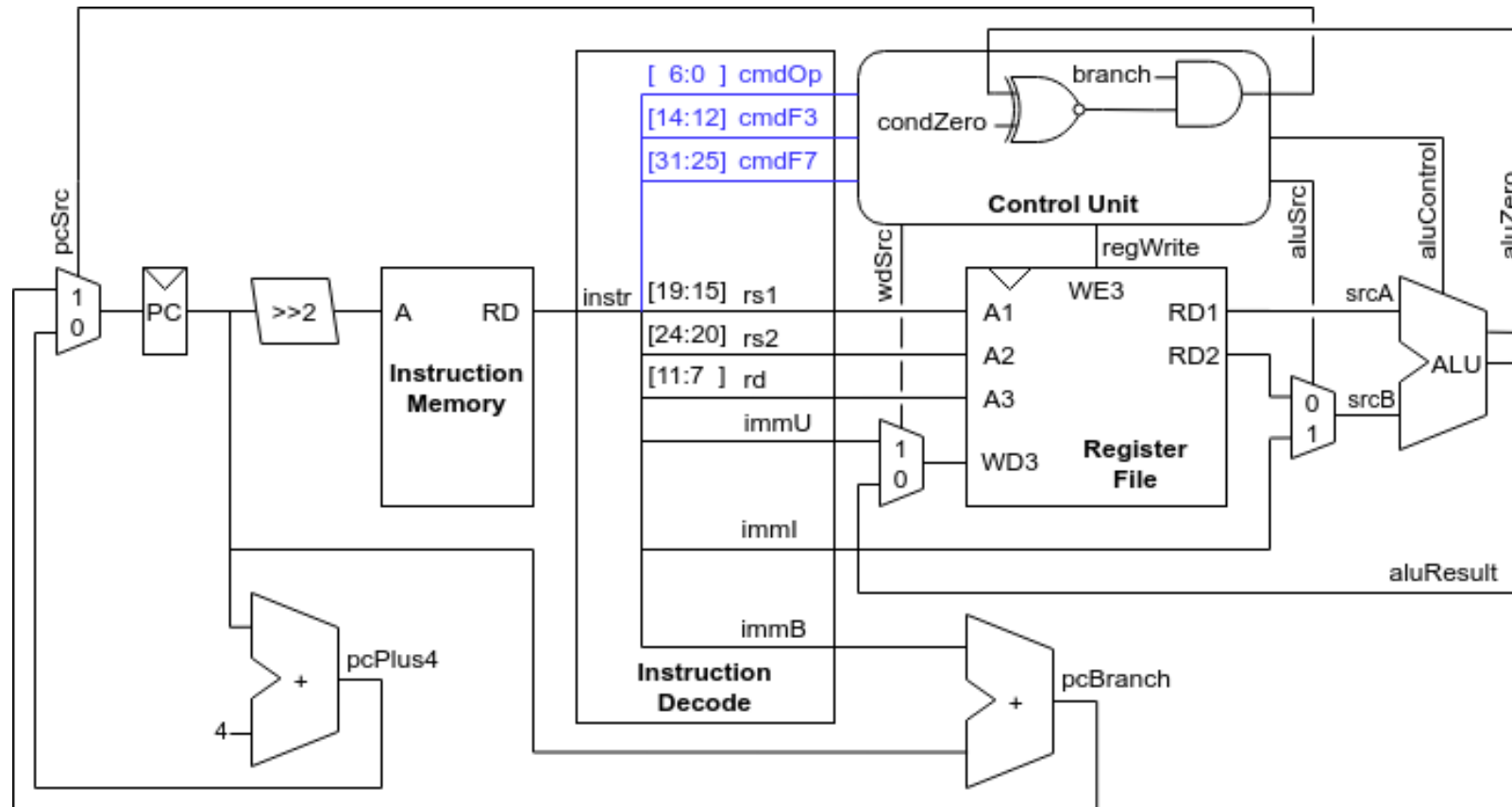
BNE: B-type, branch on non equal: **if (rs1 != rs2) PC = PC + immB**



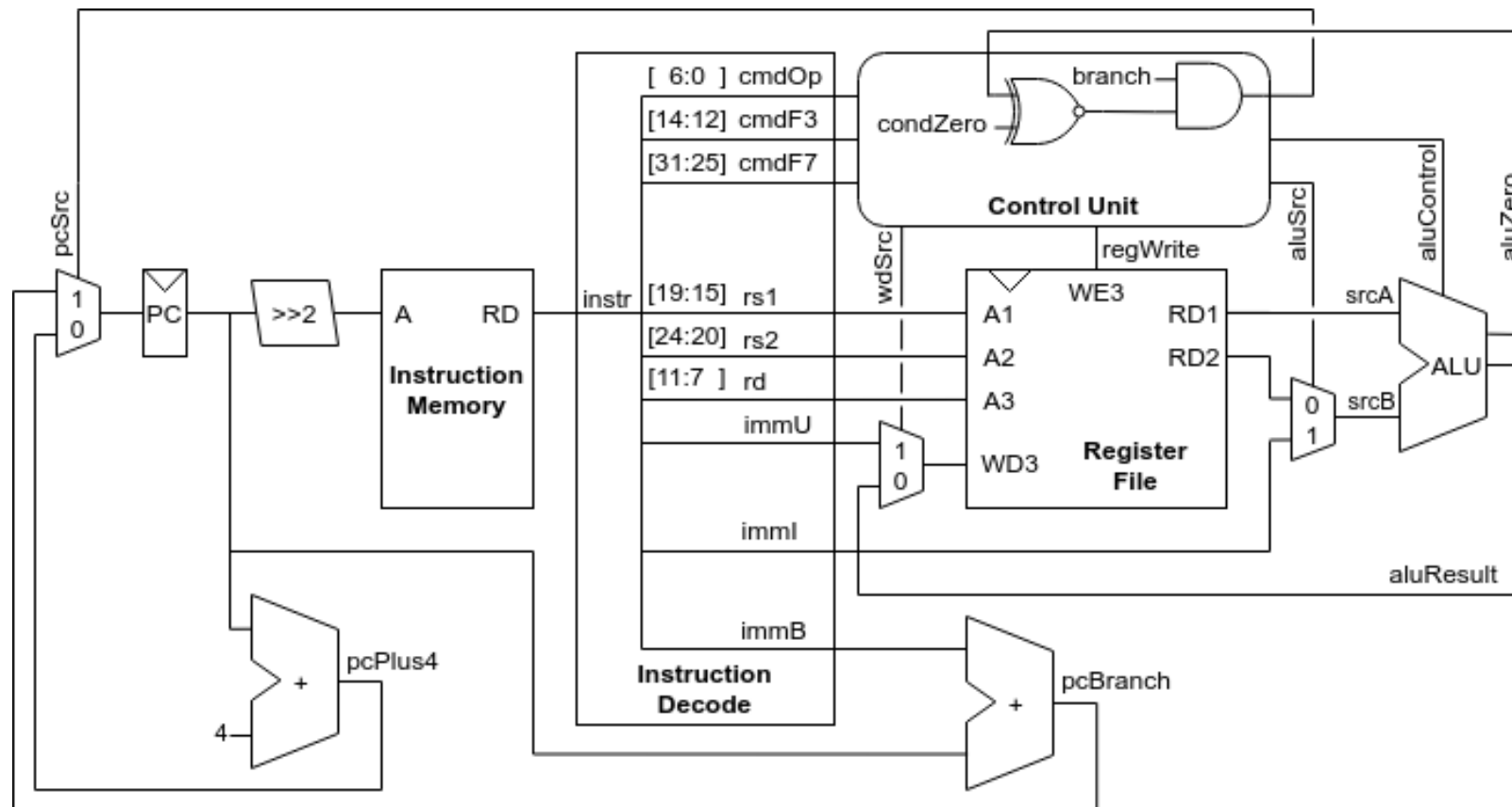
Устройство управления



Декодирование типа инструкции



Итоговая схема процессора



Состав процессора

- Тракт данных
 - Счетчик команд *PC*
 - Память инструкций *Instruction Memory*
 - Декодер инструкций *Instruction Decoder*
 - Регистровый файл *Register File*
 - Арифметико-логическое устройство *ALU*
 - Сумматоры адреса *pcPlus4* и *pcBranch*
 - Мультиплексоры *pcSrc*, *wdSrc* и *aluSrc*
- Устройство управления

Реализация: РС, сумматоры и мультиплексор адреса

```
// sm_register.v
module sm_register (
    input                clk,
    input                rst,
    input                [ 31 : 0 ] d,
    output reg [ 31 : 0 ] q
);
    always @ (posedge clk or negedge rst)
        if(~rst) q <= 32'b0;
        else    q <= d;
endmodule
```

```
// sr_cpu.v
wire [31:0] pc;
wire [31:0] pcBranch = pc + immB;
wire [31:0] pcPlus4   = pc + 4;
wire [31:0] pcNext    = pcSrc ? pcBranch : pcPlus4;
sm_register r_pc(clk ,rst_n, pcNext, pc);
```

Реализация: память инструкций

```
// sm_rom.v
module sm_rom
#(
    parameter SIZE = 64
)
(
    input  [31:0] a,
    output [31:0] rd
);
    reg [31:0] rom [SIZE - 1:0];
    assign rd = rom [a];

    initial begin
        $readmemh ("program.hex", rom);
    end

endmodule
```

```
// sm_top.v
sm_rom reset_rom(imAddr, imData);
```

Реализация: декодер инструкций (начало)

```
// sr_cpu.v
module sr_decode
(
    input      [31:0] instr,
    output     [ 6:0] cmdOp,
    output     [ 4:0] rd,
    output     [ 2:0] cmdF3,
    output     [ 4:0] rs1,
    output     [ 4:0] rs2,
    output     [ 6:0] cmdF7,
    output reg [31:0] immI,
    output reg [31:0] immB,
    output reg [31:0] immU
);

    assign cmdOp = instr[ 6: 0];
    assign rd    = instr[11: 7];
    assign cmdF3 = instr[14:12];
    assign rs1   = instr[19:15];
    assign rs2   = instr[24:20];
    assign cmdF7 = instr[31:25];
```


Реализация: декодер инструкций (продолжение)

```
// I-immediate
always @ (*) begin
    immI[10: 0] = instr[30:20];
    immI[31:11] = { 21 {instr[31]} };
end

// B-immediate
always @ (*) begin
    immB[ 0] = 1'b0;
    immB[ 4: 1] = instr[11:8];
    immB[10: 5] = instr[30:25];
    immB[31:11] = { 21 {instr[31]} };
end

// U-immediate
always @ (*) begin
    immU[11: 0] = 12'b0;
    immU[31:12] = instr[31:12];
end
endmodule
```

Реализация: регистровый файл

```
// sr_cpu.v
module sm_register_file
(
    input          clk,
    input  [ 4:0]  a0,
    input  [ 4:0]  a1,
    input  [ 4:0]  a2,
    input  [ 4:0]  a3,
    output [31:0]  rd0,
    output [31:0]  rd1,
    output [31:0]  rd2,
    input  [31:0]  wd3,
    input          we3
);
    reg [31:0] rf [31:0];

    assign rd0 = (a0 != 0) ? rf [a0] : 32'b0;
    assign rd1 = (a1 != 0) ? rf [a1] : 32'b0;
    assign rd2 = (a2 != 0) ? rf [a2] : 32'b0;

    always @ (posedge clk)
        if(we3) rf [a3] <= wd3;
endmodule
```

Реализация: операции ALU

```
// sr_cpu.vh

`define ALU_ADD      3'b000  // A + B

`define ALU_OR       3'b001  // A | B

`define ALU_SRL      3'b010  // A >> B

`define ALU_SLTU     3'b011  // A < B ? 1 : 0

`define ALU_SUB      3'b100  // A - B
```

Реализация: ALU

```
// sr_cpu.v
module sr_alu
(
    input  [31:0] srcA,
    input  [31:0] srcB,
    input  [ 2:0] oper,
    output          zero,
    output reg [31:0] result
);
    always @ (*) begin
        case (oper)
            default      : result = srcA + srcB;
            `ALU_ADD     : result = srcA + srcB;
            `ALU_OR      : result = srcA | srcB;
            `ALU_SRL     : result = srcA >> srcB [4:0];
            `ALU_SLTU    : result = (srcA < srcB) ? 1 : 0;
            `ALU_SUB     : result = srcA - srcB;
        endcase
    end

    assign zero = (result == 0);
endmodule
```

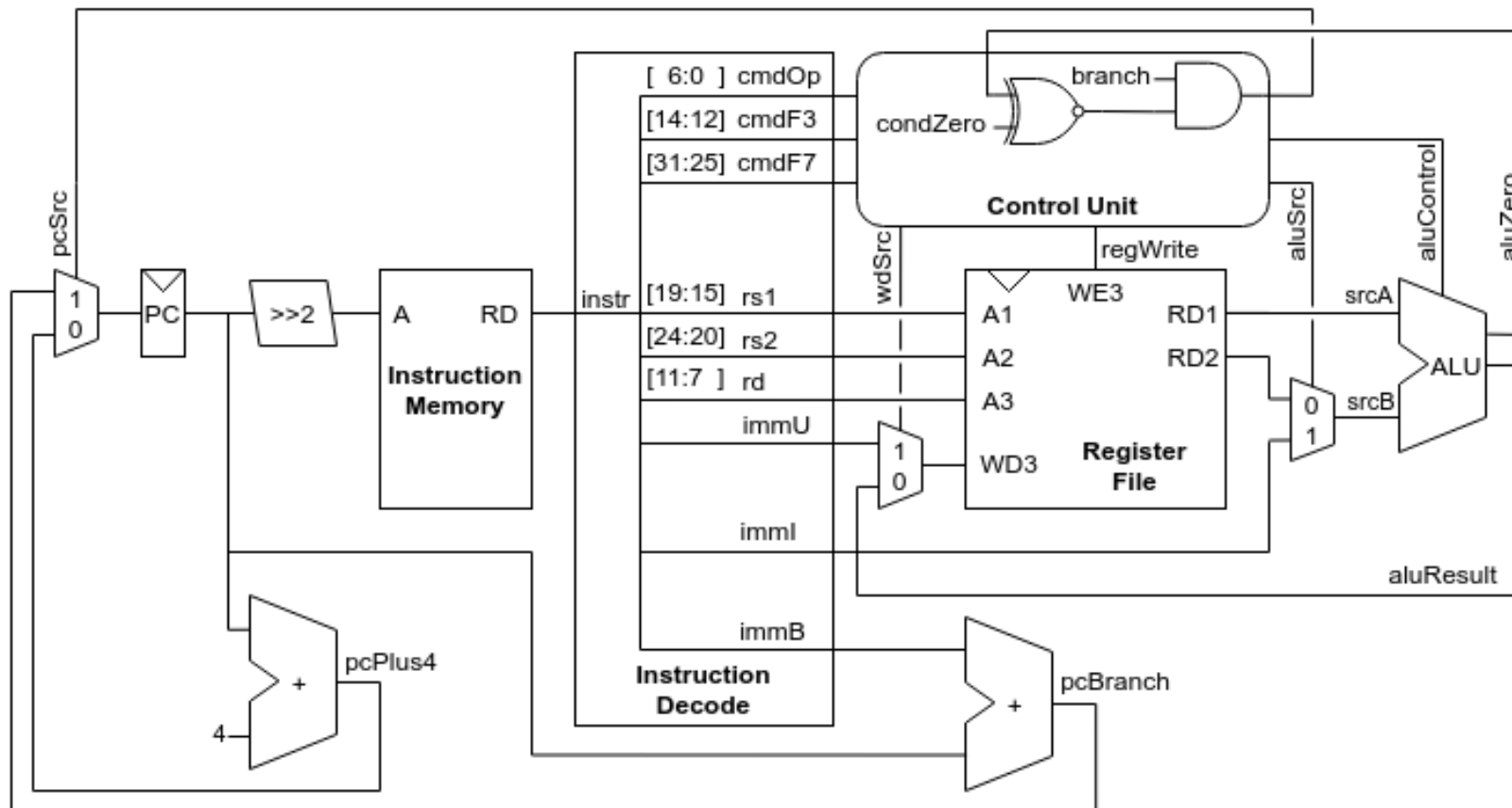
Реализация: мультиплексоры данных

```
// sr_cpu.v  
wire [31:0] srcB = aluSrc ? immI : rd2;
```

```
// sr_cpu.v  
assign wd3 = wdSrc ? immU : aluResult;
```

Сигналы управления 1

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero



Код операции: спецификация

130

Volume I: RISC-V Unprivileged ISA V20191213

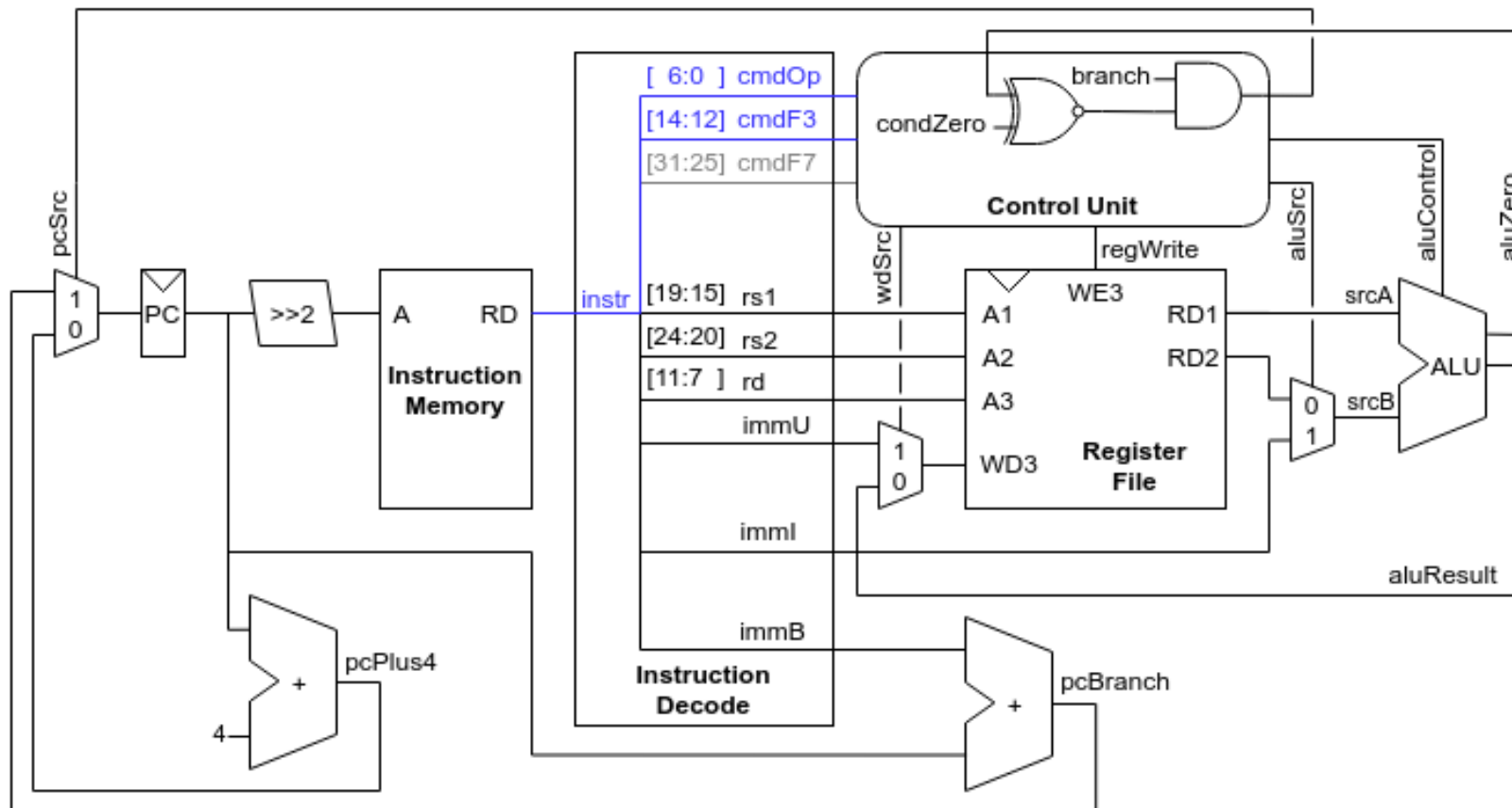
31	27	26	25	24	20	19	15	14	12	11	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type
imm[11:0]					rs1		funct3		rd		opcode		I-type	
imm[11:5]				rs2		rs1		funct3		imm[4:0]		opcode		S-type
imm[12 10:5]				rs2		rs1		funct3		imm[4:1 11]		opcode		B-type
imm[31:12]										rd		opcode		U-type
imm[20 10:1 11 19:12]										rd		opcode		J-type

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20 10:1 11 19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12 10:5]	rs2	rs1	000	imm[4:1 11]	1100011	BEQ
imm[12 10:5]	rs2	rs1	001	imm[4:1 11]	1100011	BNE
imm[12 10:5]	rs2	rs1	100	imm[4:1 11]	1100011	BLT
imm[12 10:5]	rs2	rs1	101	imm[4:1 11]	1100011	BGE
imm[12 10:5]	rs2	rs1	110	imm[4:1 11]	1100011	BLTU
imm[12 10:5]	rs2	rs1	111	imm[4:1 11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI

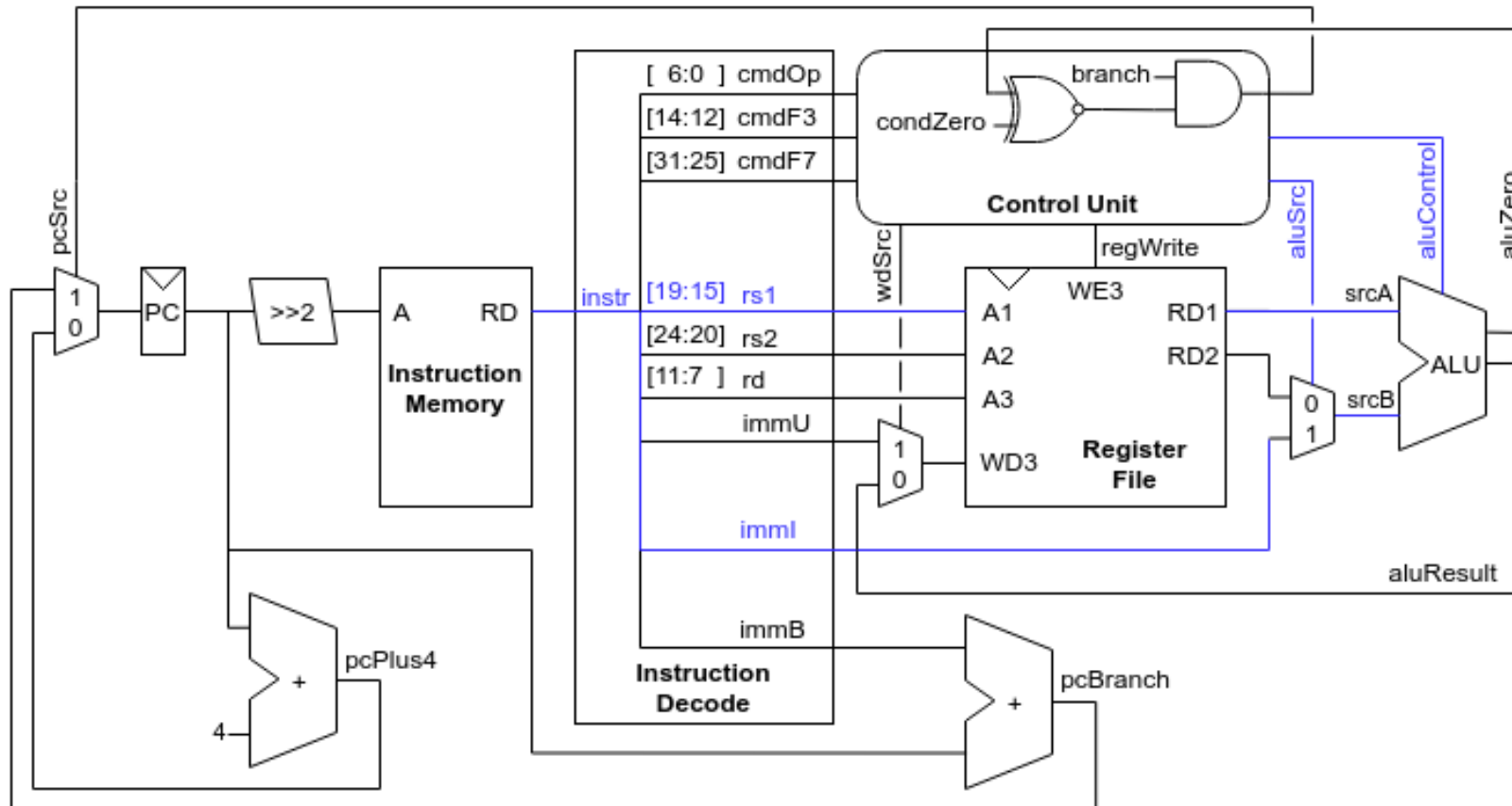
Сигналы управления 2

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????							



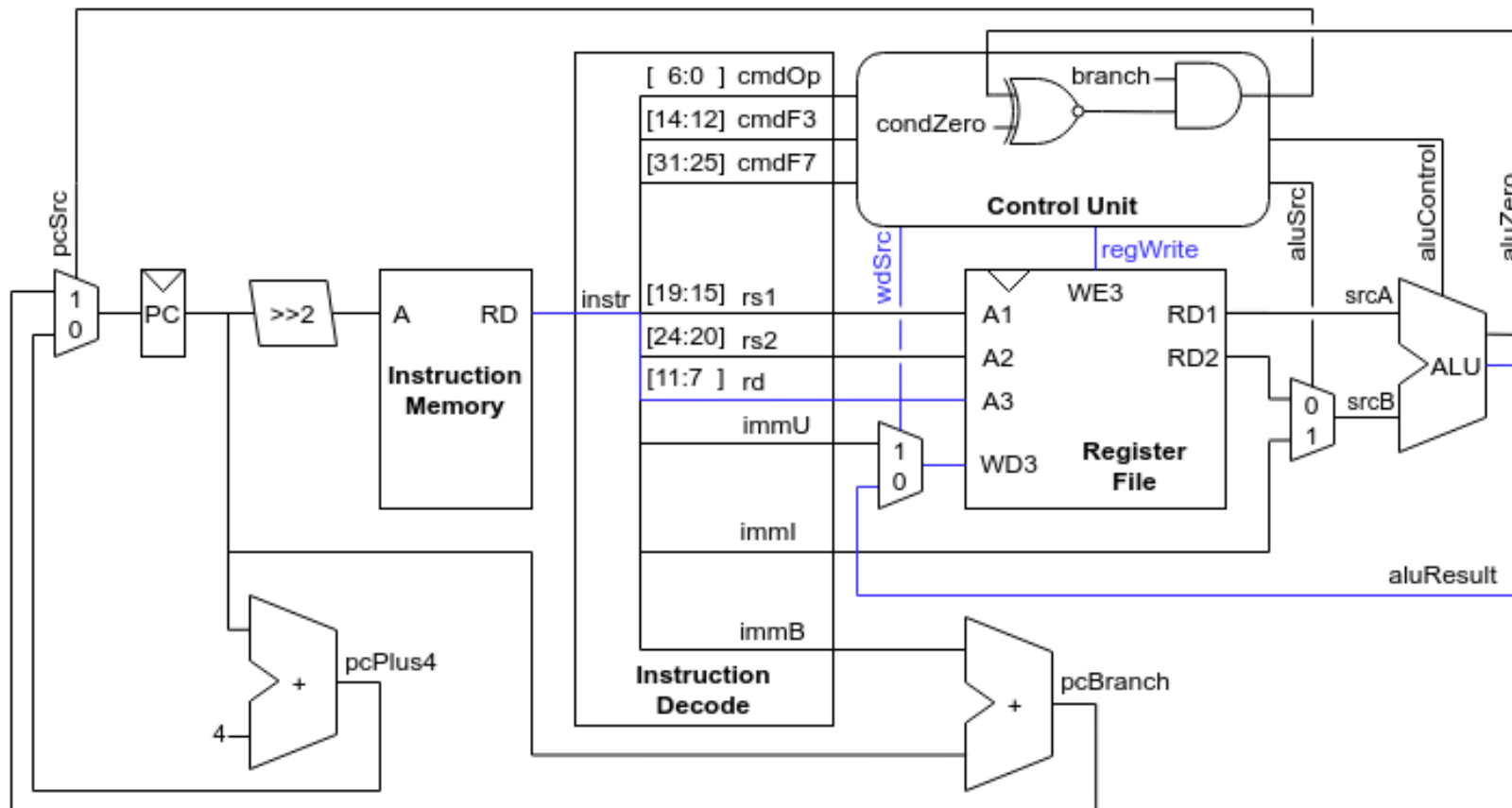
Сигналы управления 3

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000					



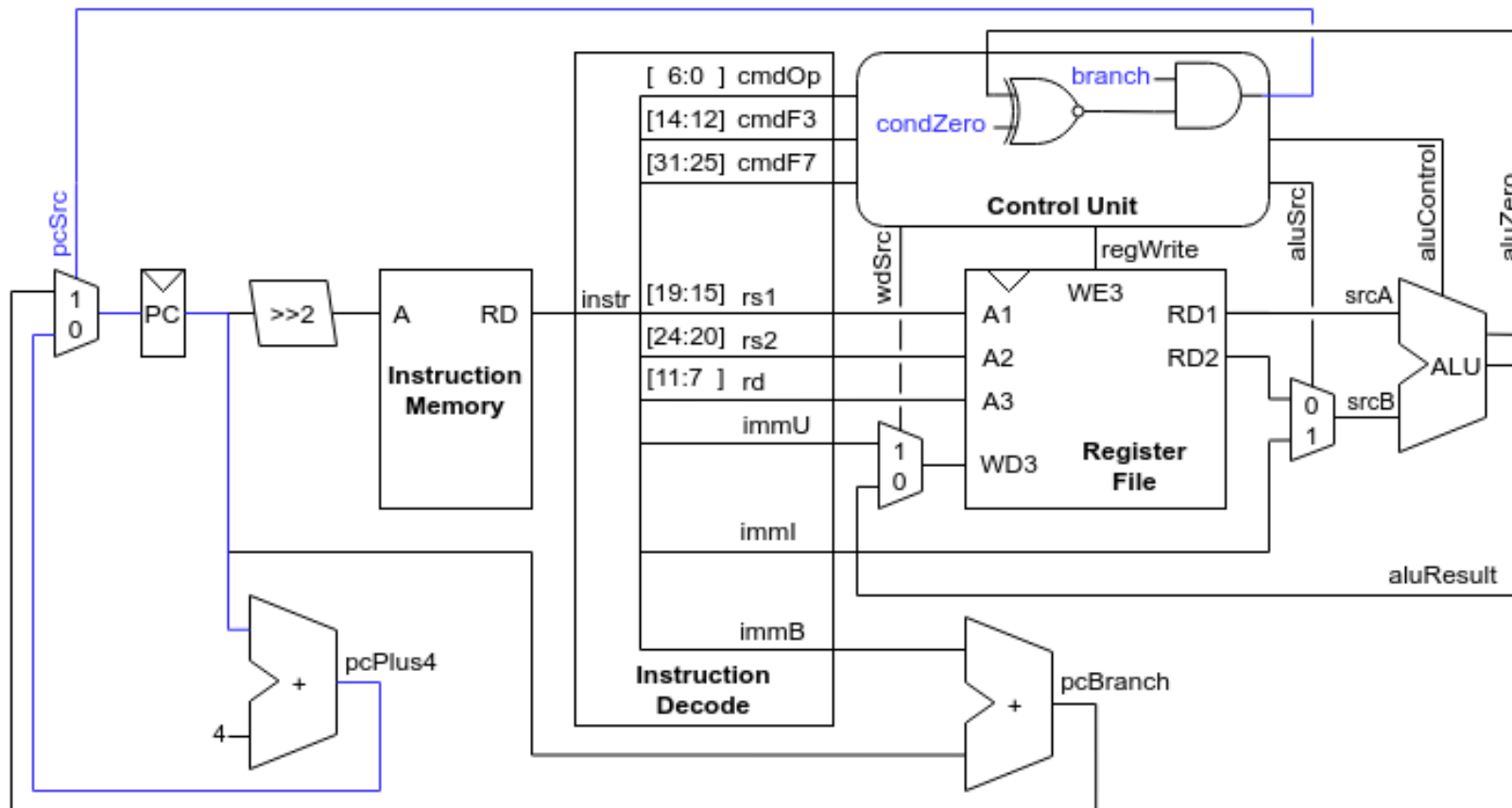
Сигналы управления 4

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1			



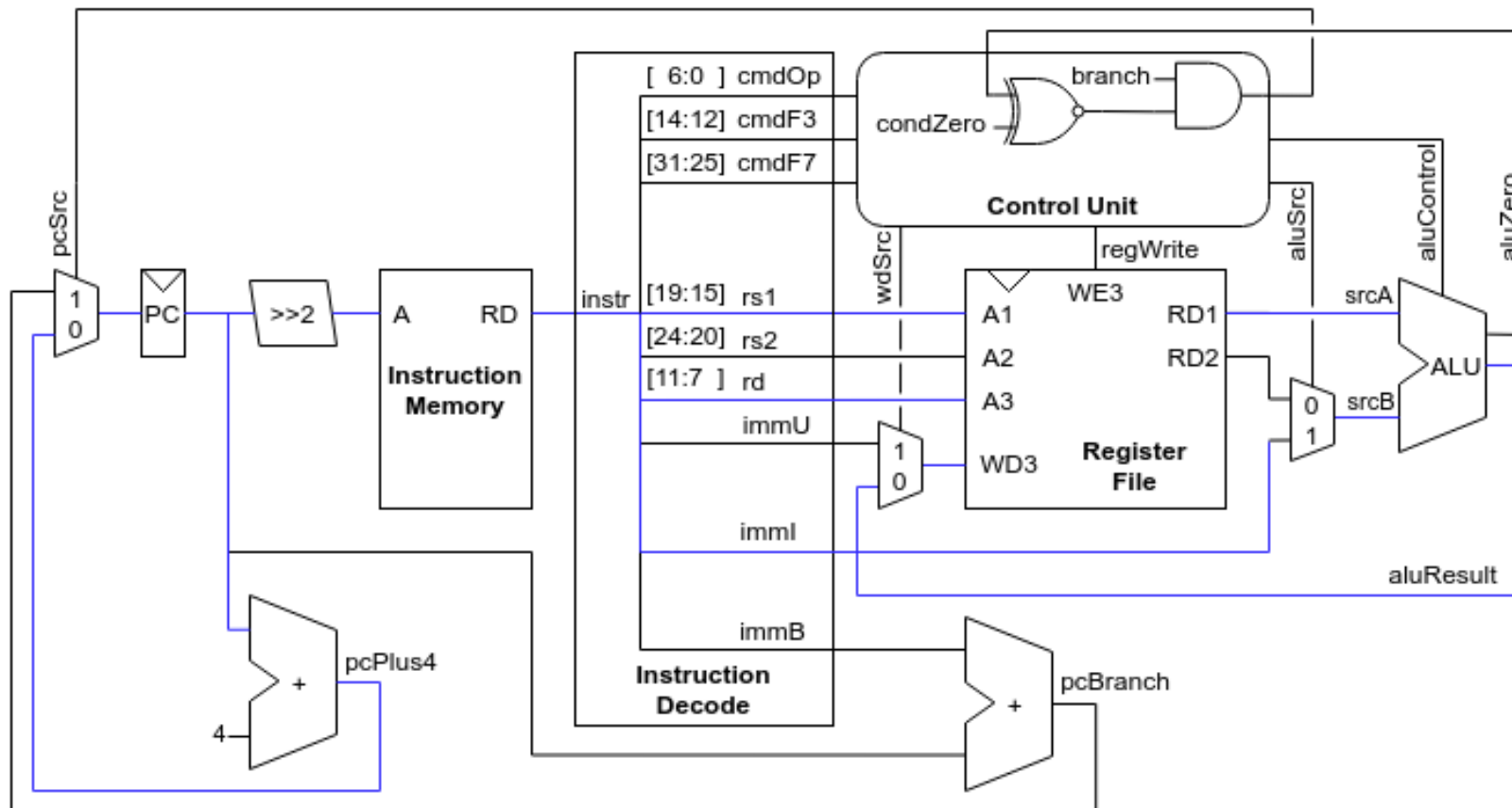
Сигналы управления 5

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0



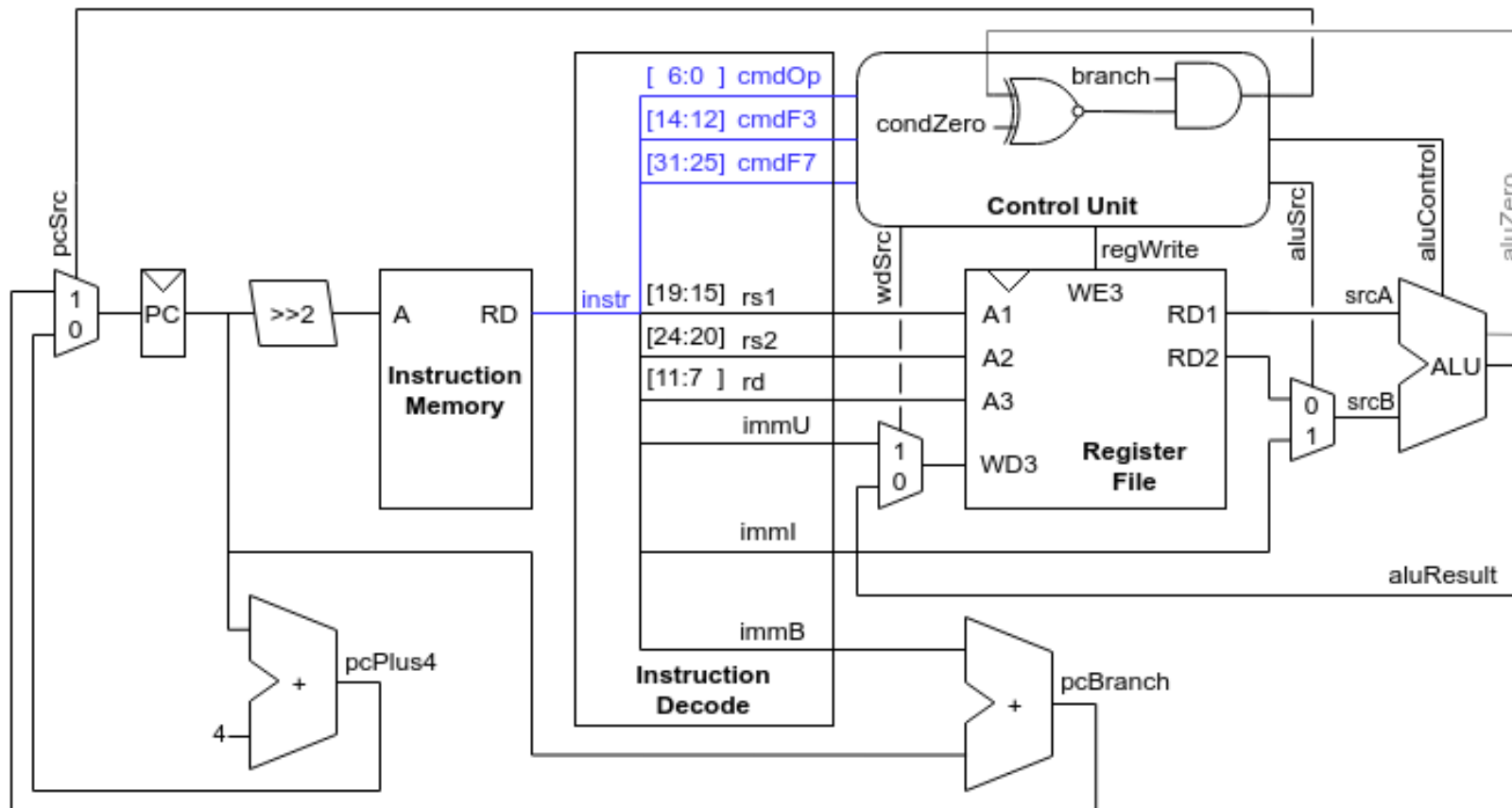
Сигналы управления 6

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0



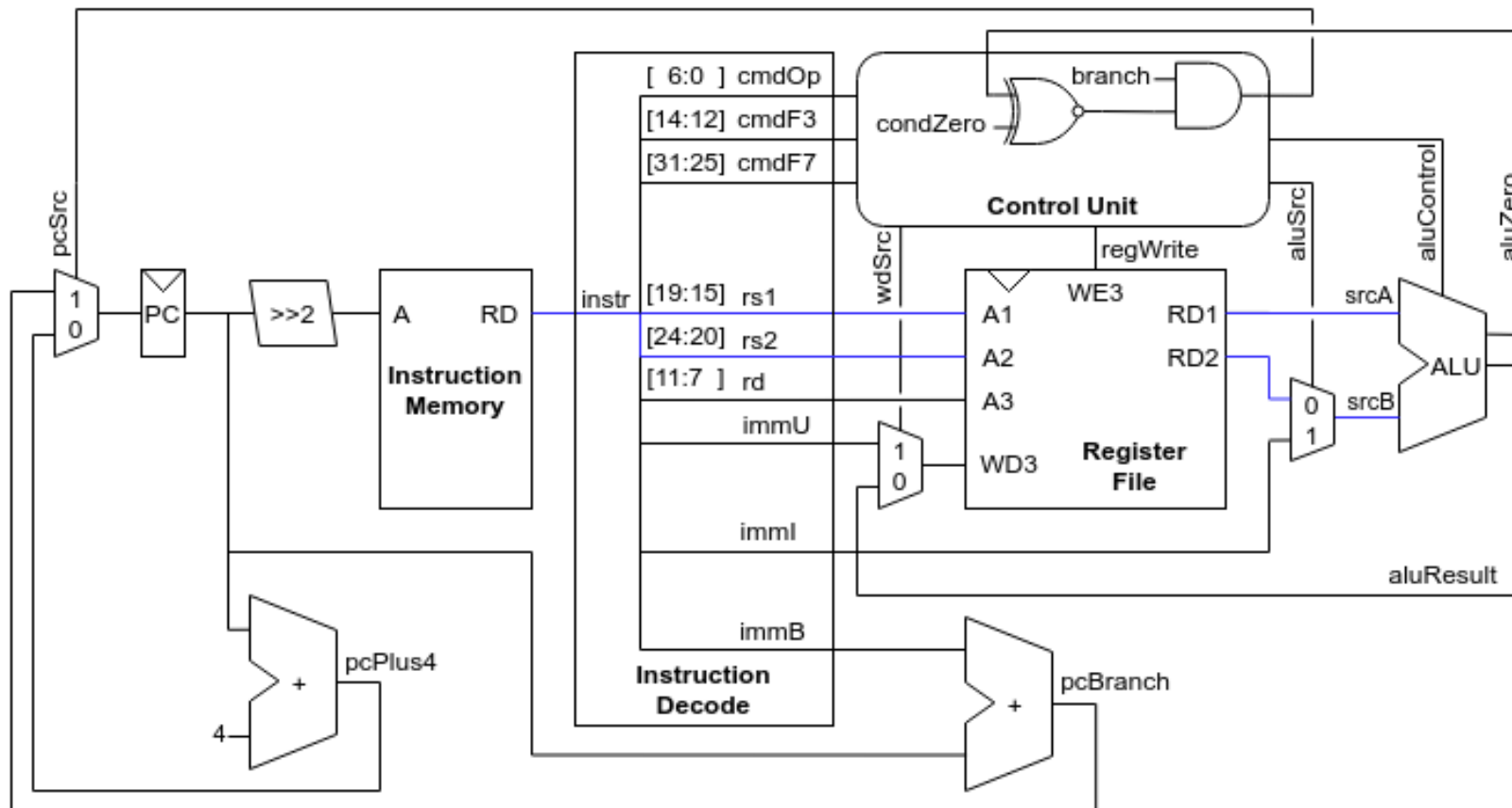
Сигналы управления 7

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000							



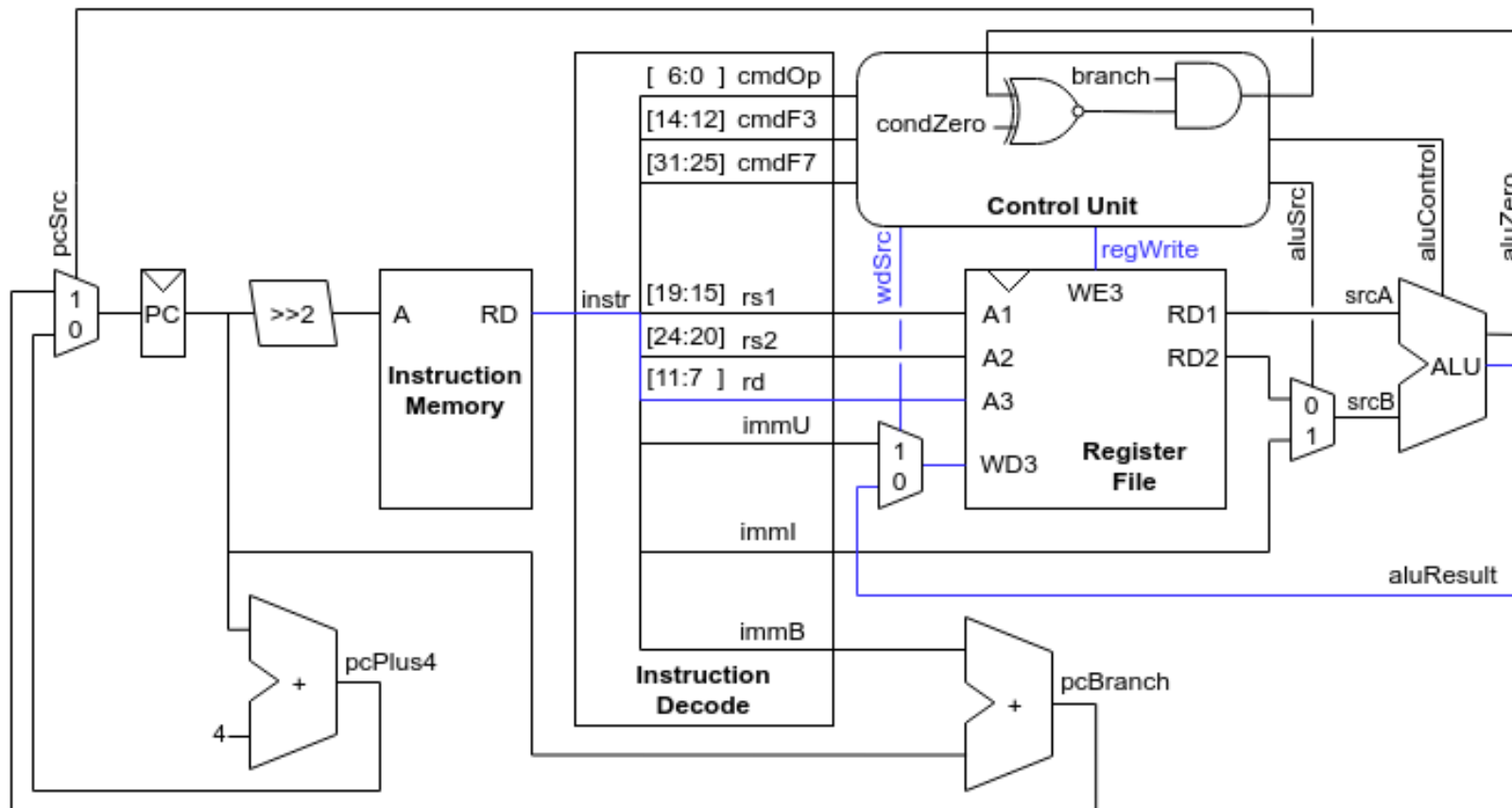
Сигналы управления 8

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000					



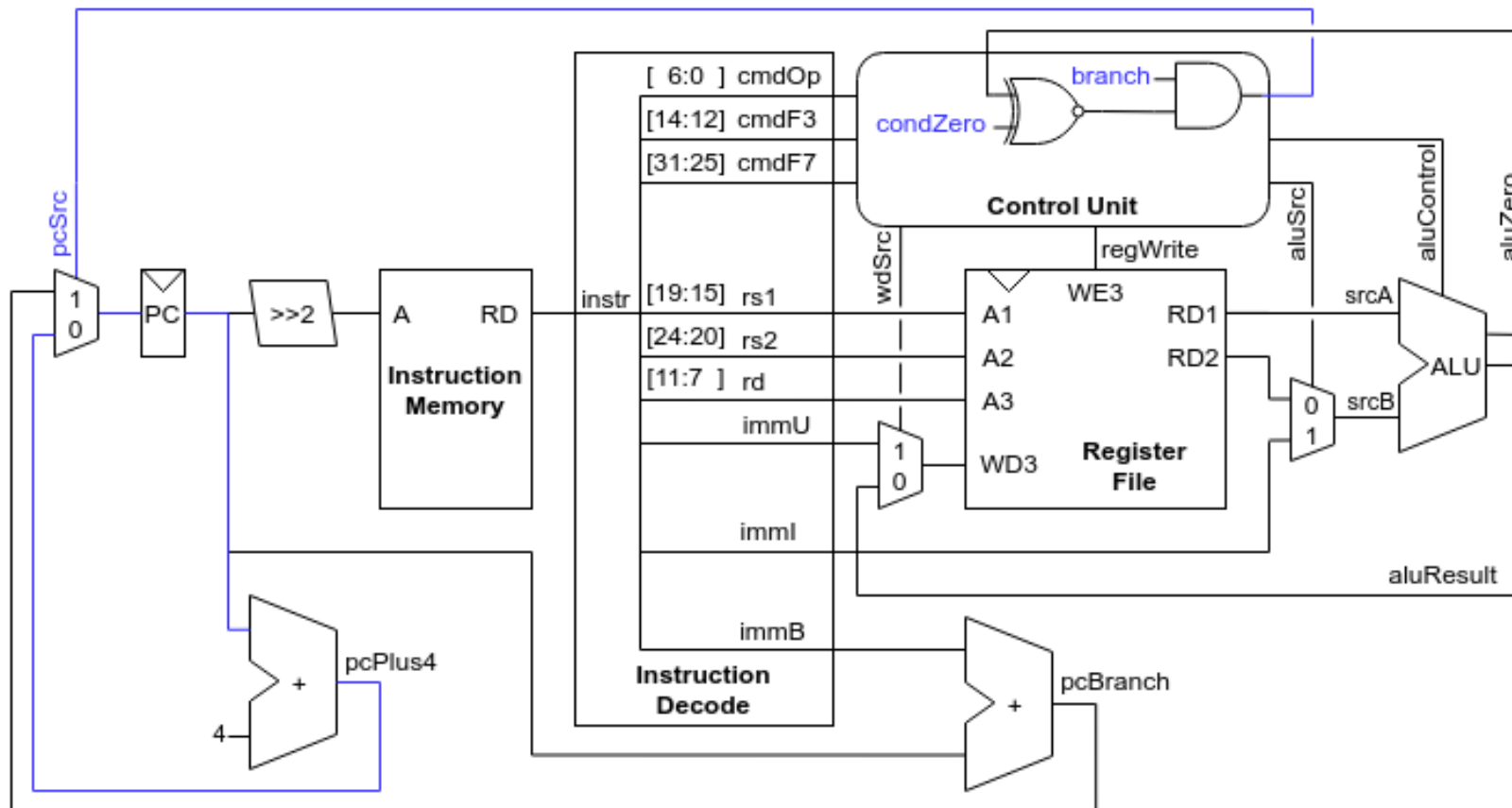
Сигналы управления 9

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1			



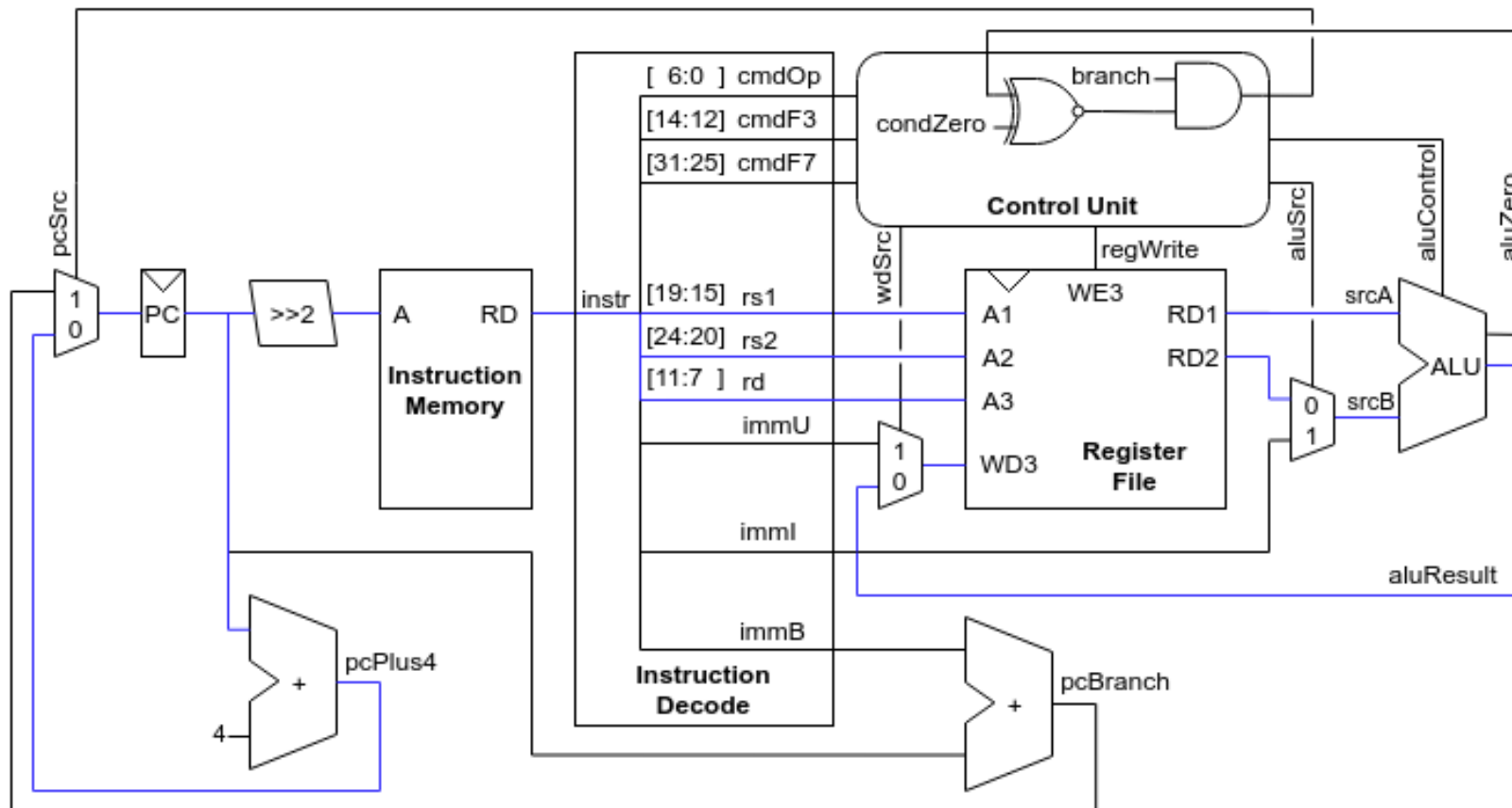
Сигналы управления 10

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0



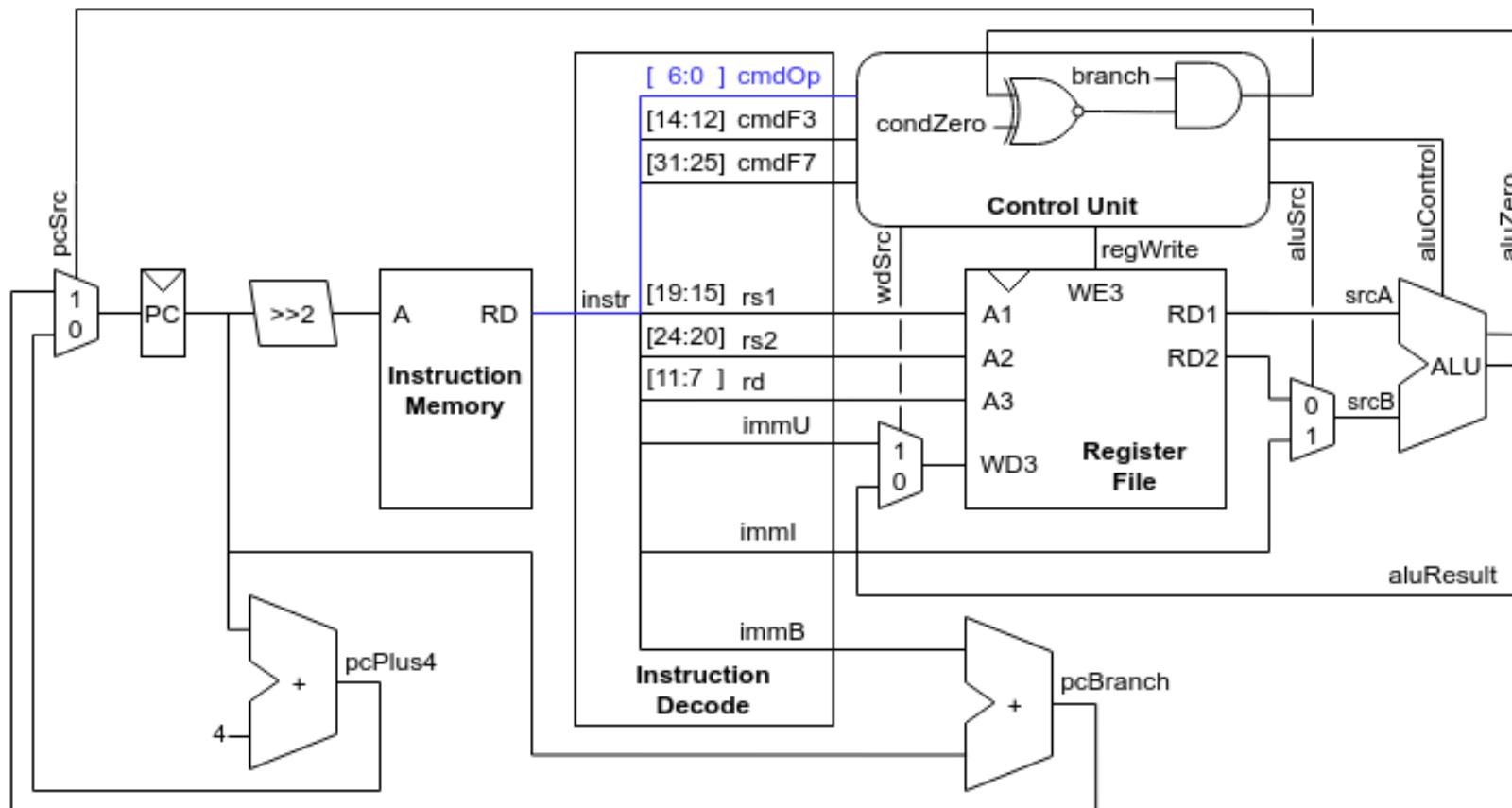
Сигналы управления 11

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0



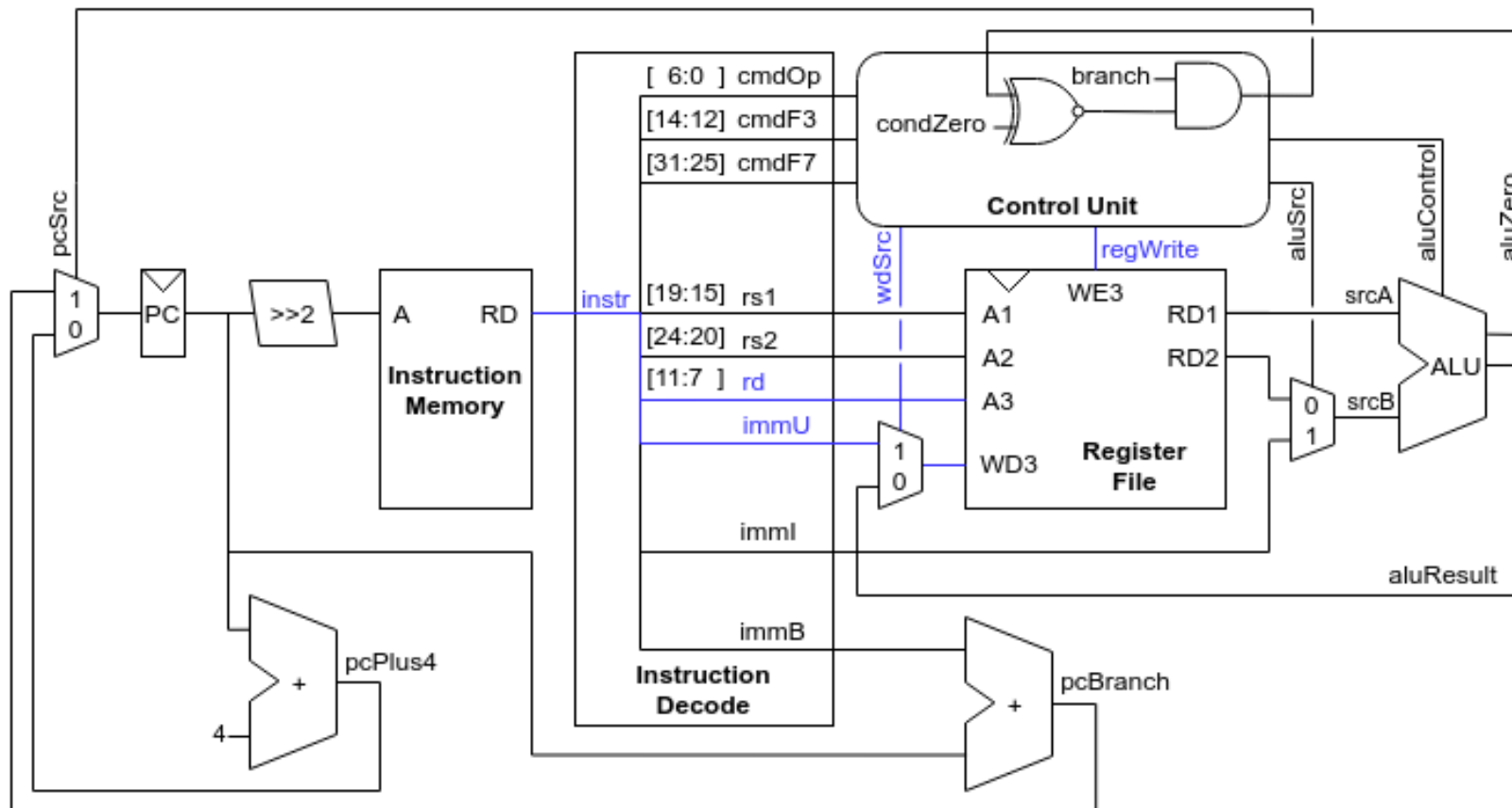
Сигналы управления 12

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	????????							



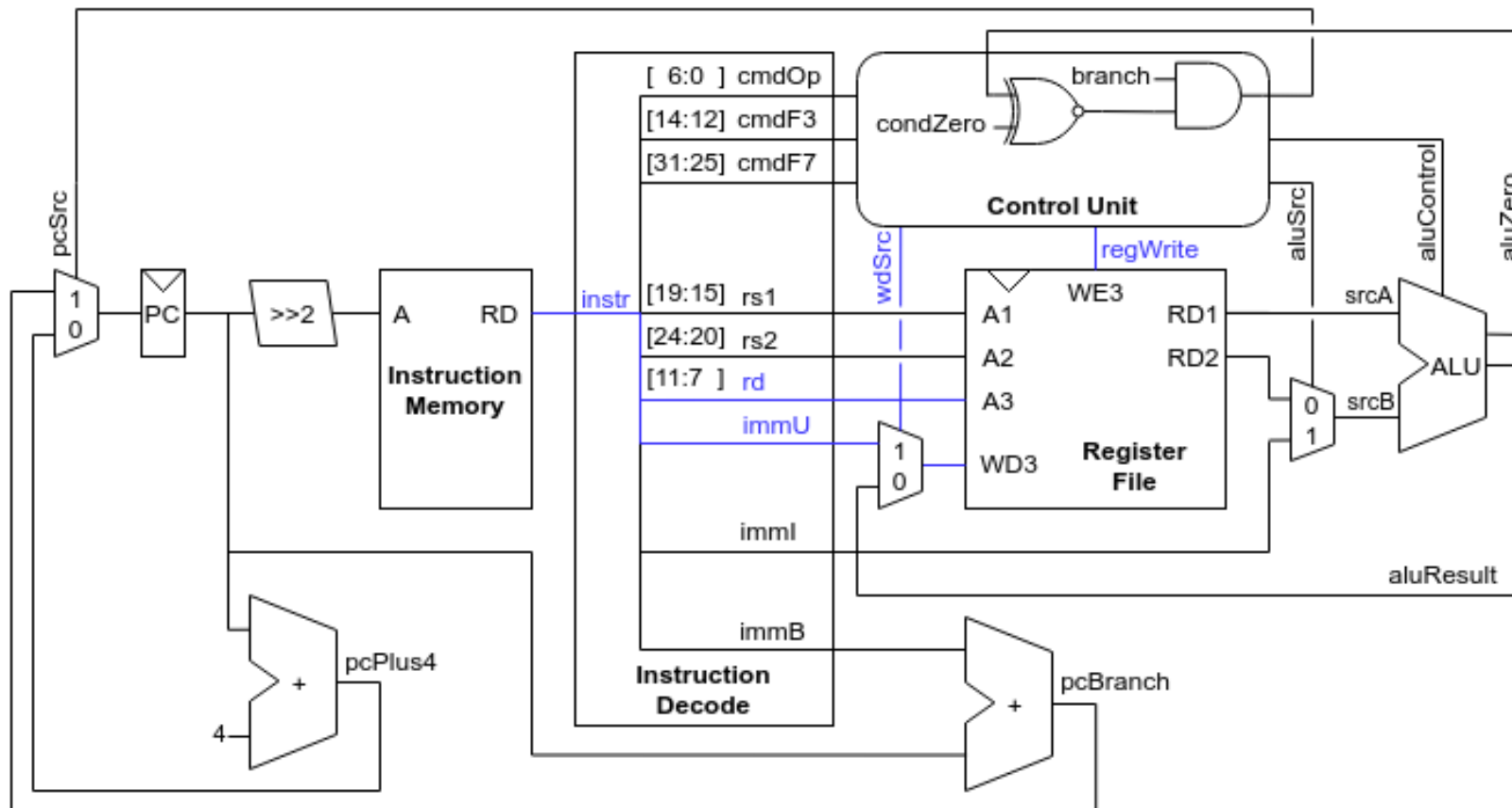
Сигналы управления 13

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	???????			1	1			



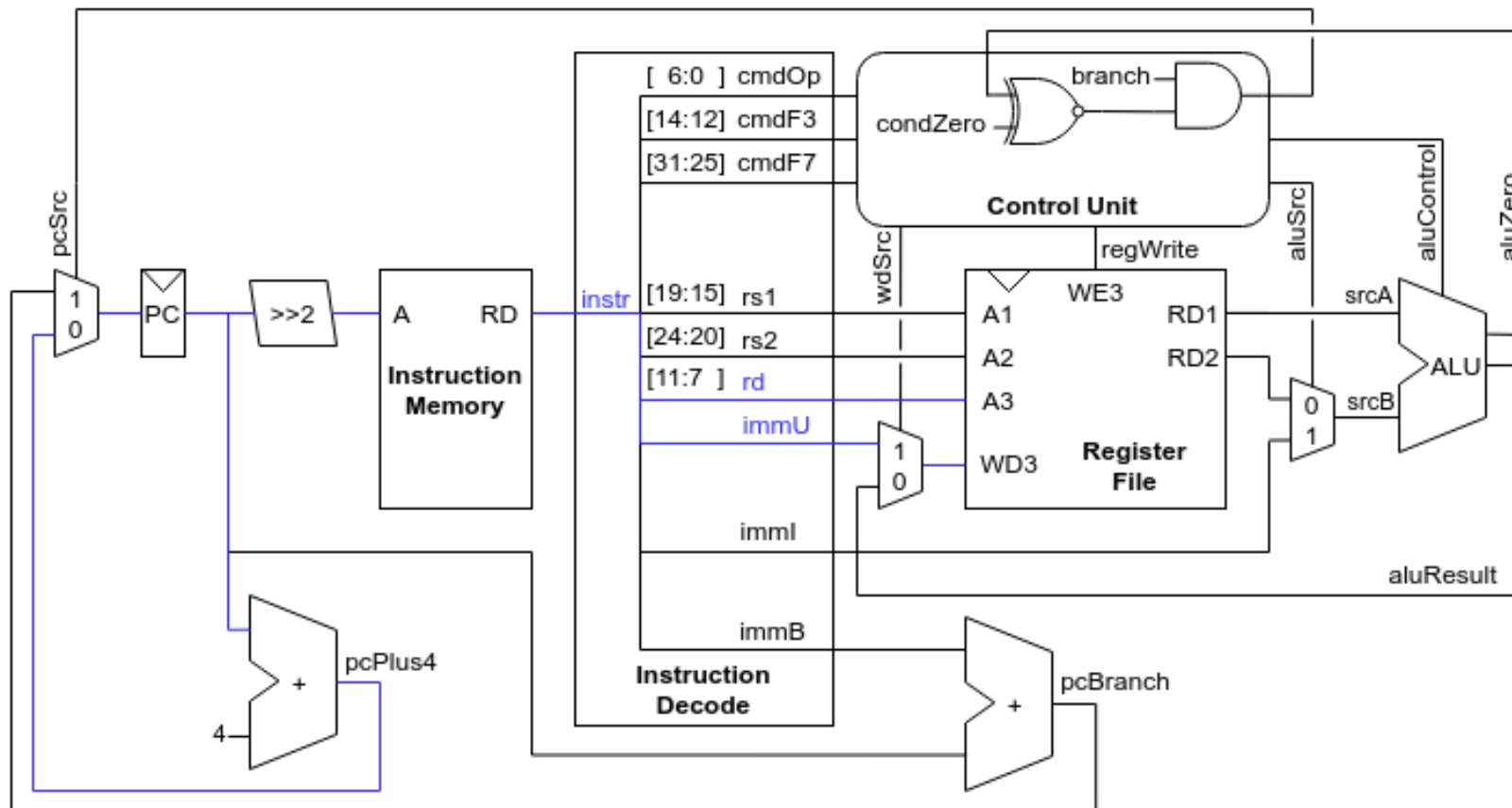
Сигналы управления 14

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	???????	0	000	1	1	0	0	0



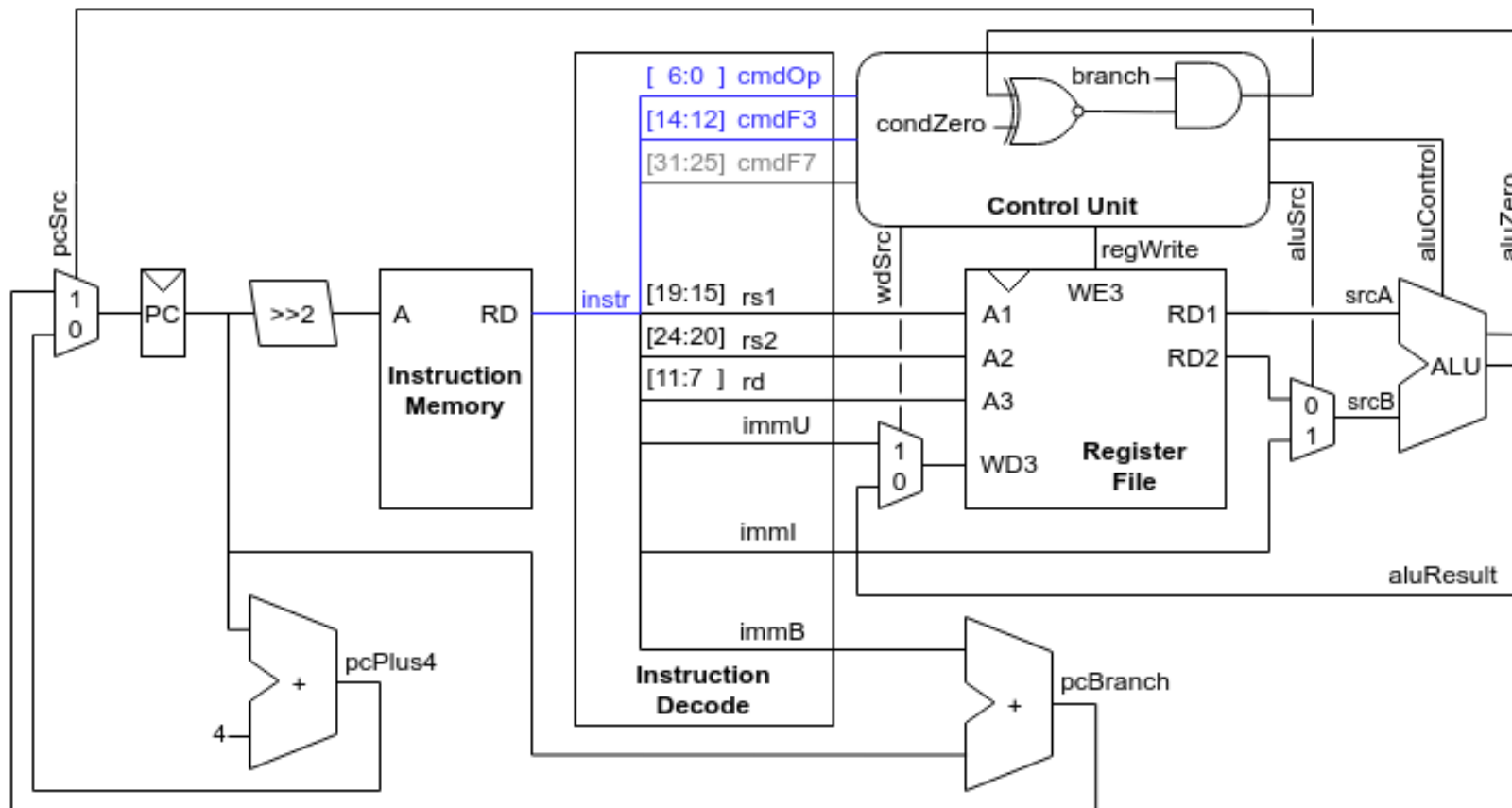
Сигналы управления 15

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000	0	1	0	0	0
ADD	0110011	000	00000000	0	000	0	1	0	0	0
LUI	0110111	???	????????	0	000	1	1	0	0	0



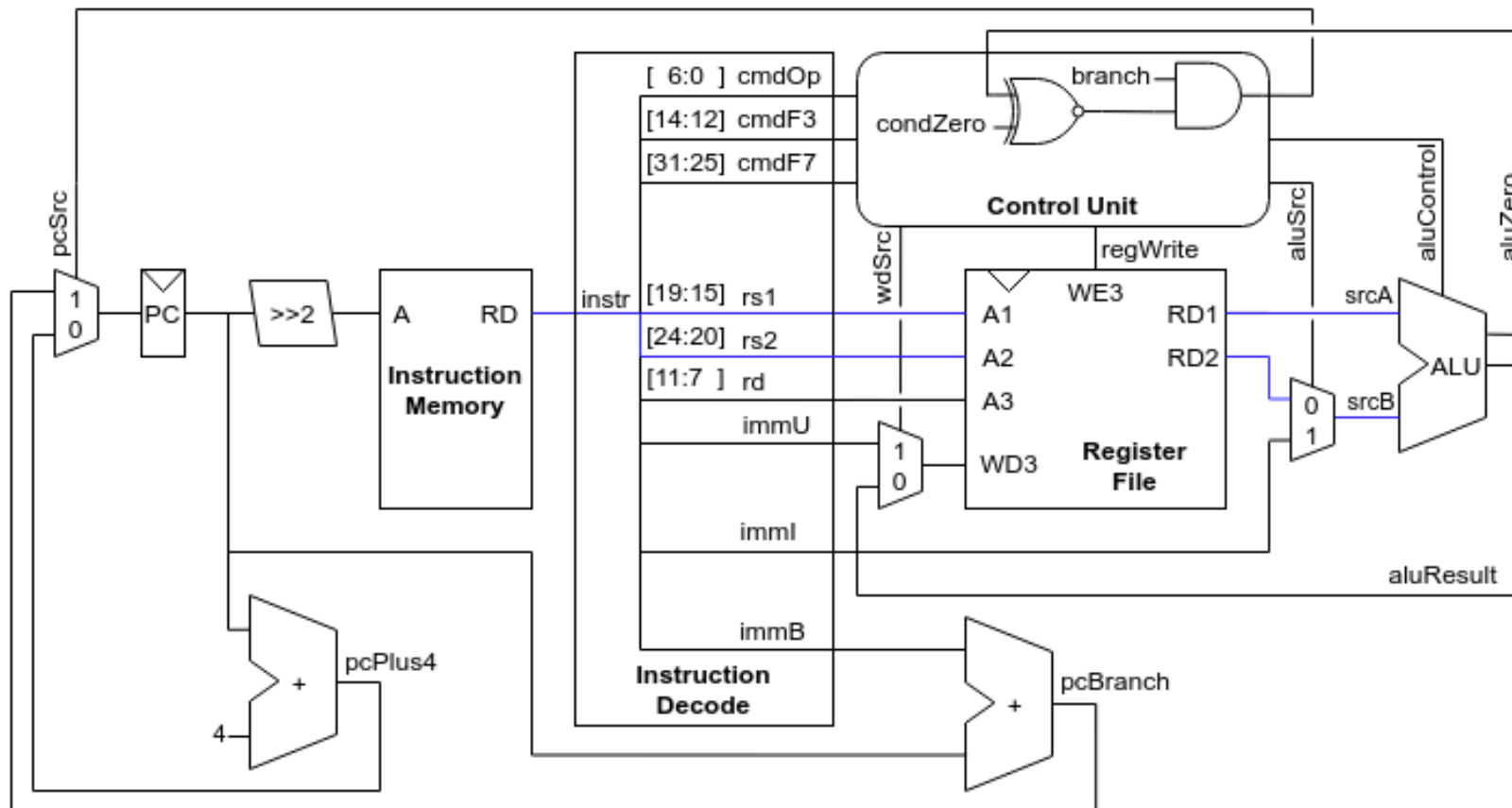
Сигналы управления 16

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	???????	0	000	1	1	0	0	0
BEQ	1100011	000	???????							



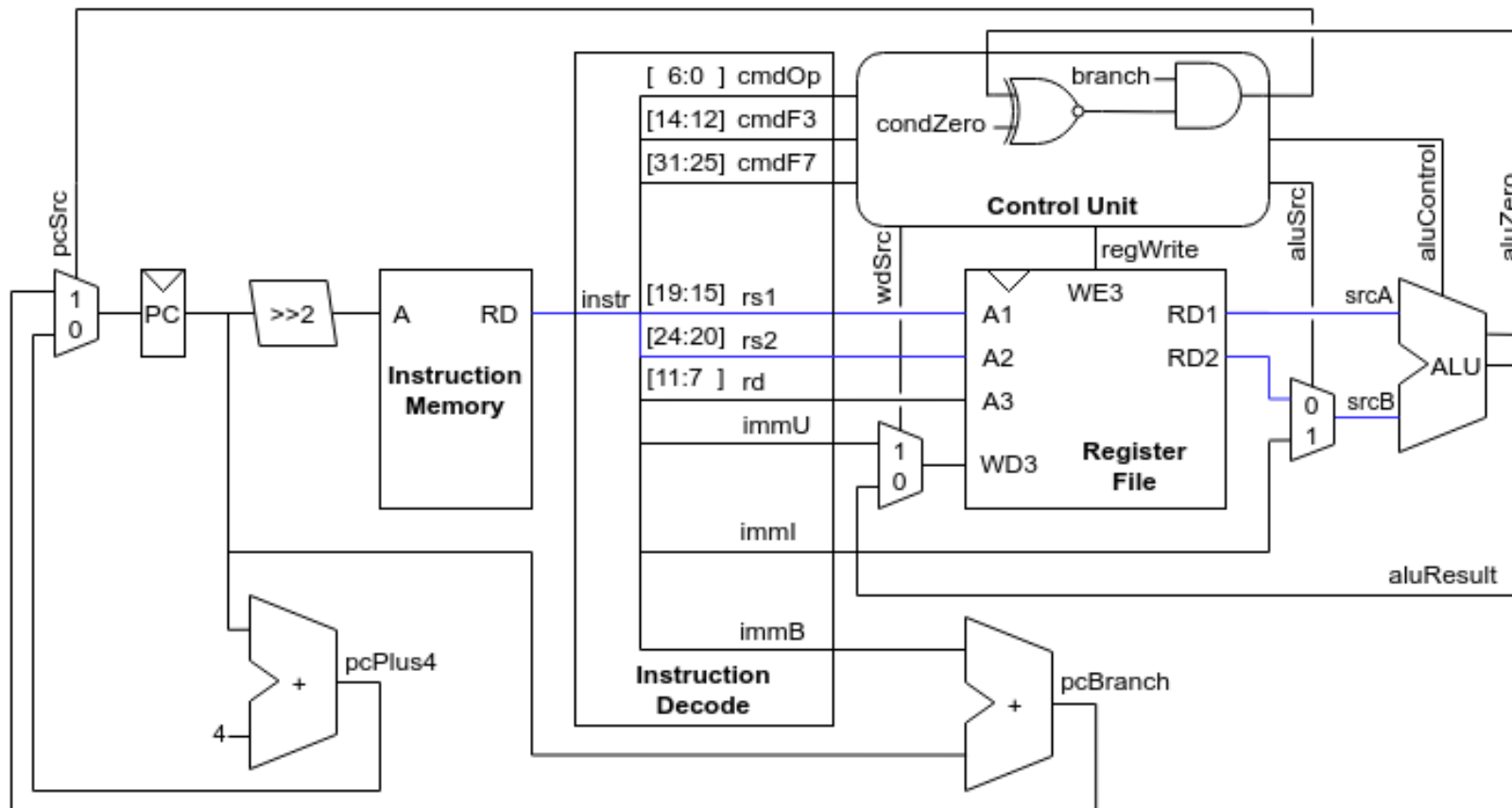
Сигналы управления 17

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000	0	1	0	0	0
ADD	0110011	000	00000000	0	000	0	1	0	0	0
LUI	0110111	???	????????	0	000	1	1	0	0	0
BEQ	1100011	000	????????	0	100					



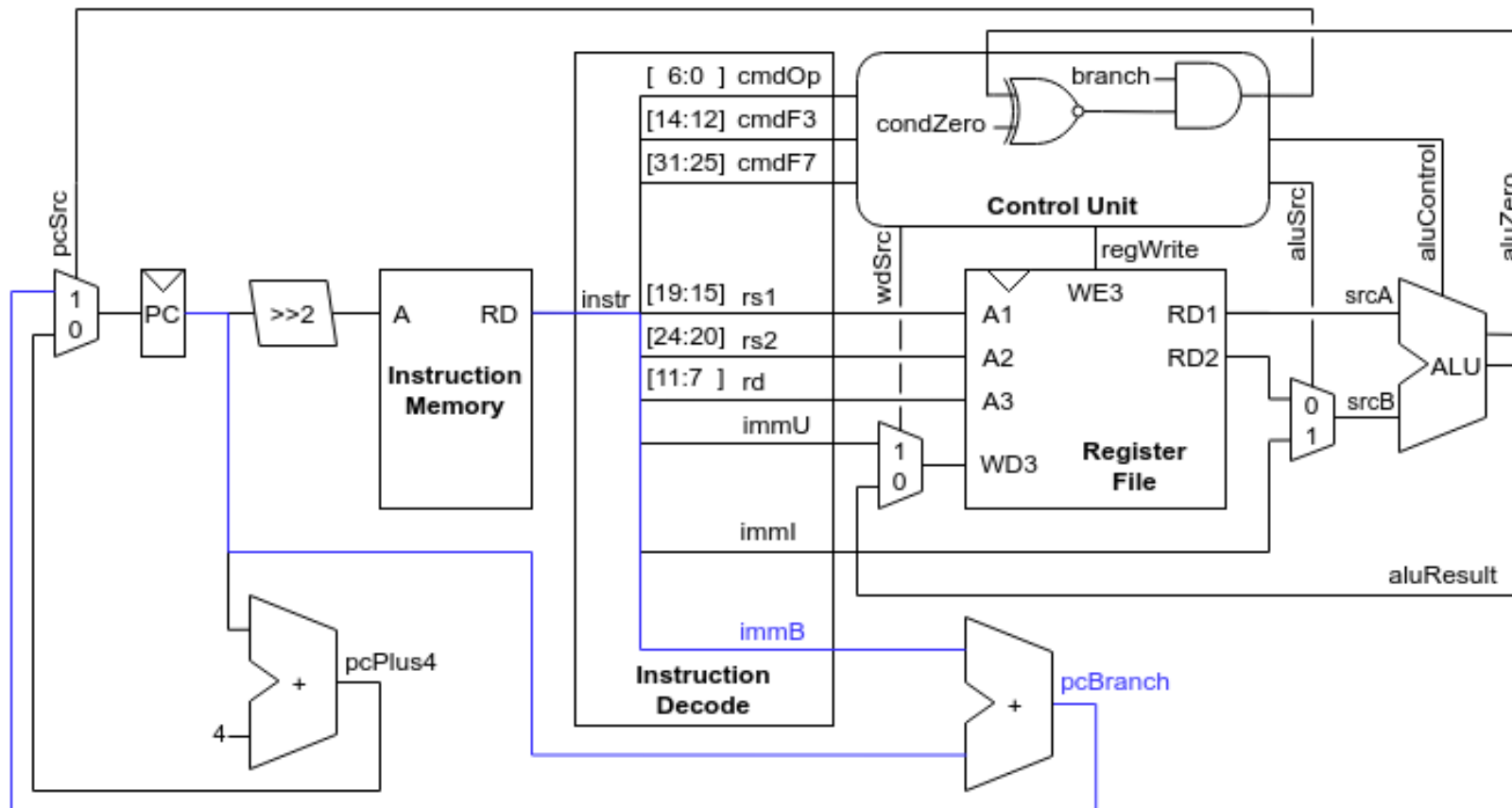
Сигналы управления 18

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000	0	1	0	0	0
ADD	0110011	000	00000000	0	000	0	1	0	0	0
LUI	0110111	???	????????	0	000	1	1	0	0	0
BEQ	1100011	000	????????	0	100	0	0			



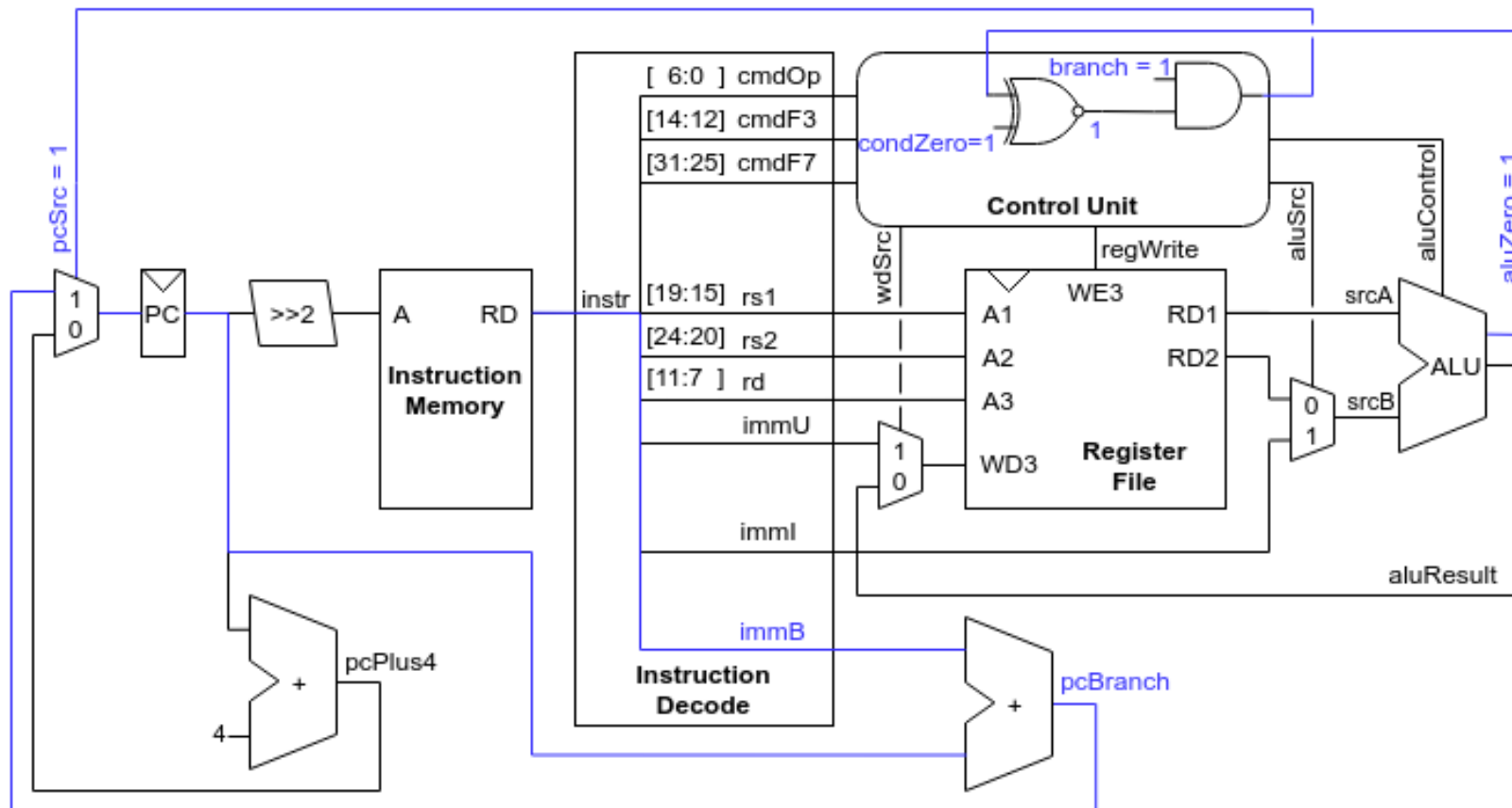
Сигналы управления 19

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000	0	1	0	0	0
ADD	0110011	000	00000000	0	000	0	1	0	0	0
LUI	0110111	???	????????	0	000	1	1	0	0	0
BEQ	1100011	000	????????	0	100	0	0			



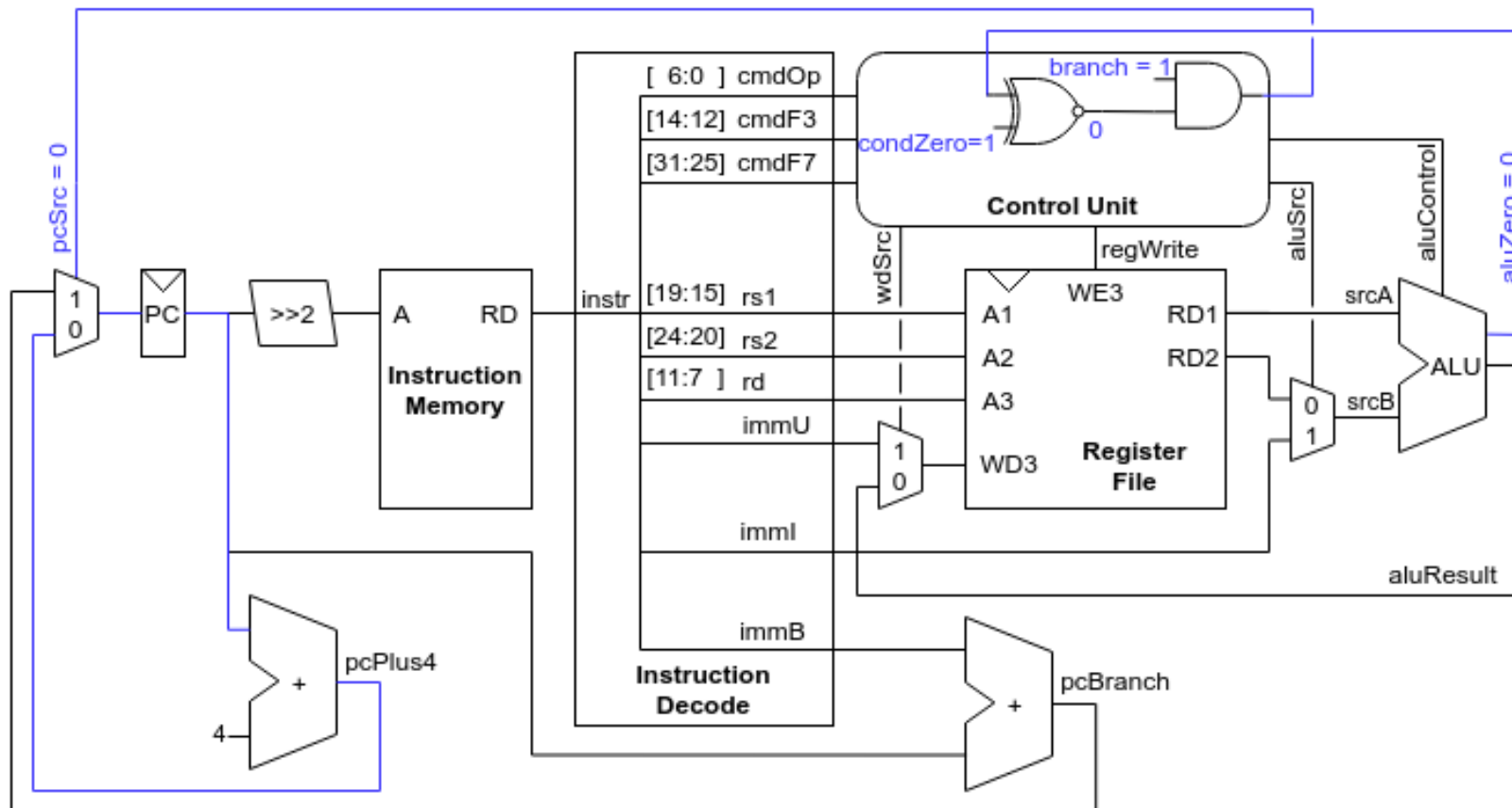
Сигналы управления 20

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000	0	1	0	0	0
ADD	0110011	000	00000000	0	000	0	1	0	0	0
LUI	0110111	???	????????	0	000	1	1	0	0	0
BEQ	1100011	000	????????	0	100	0	0	aluZero	1	1



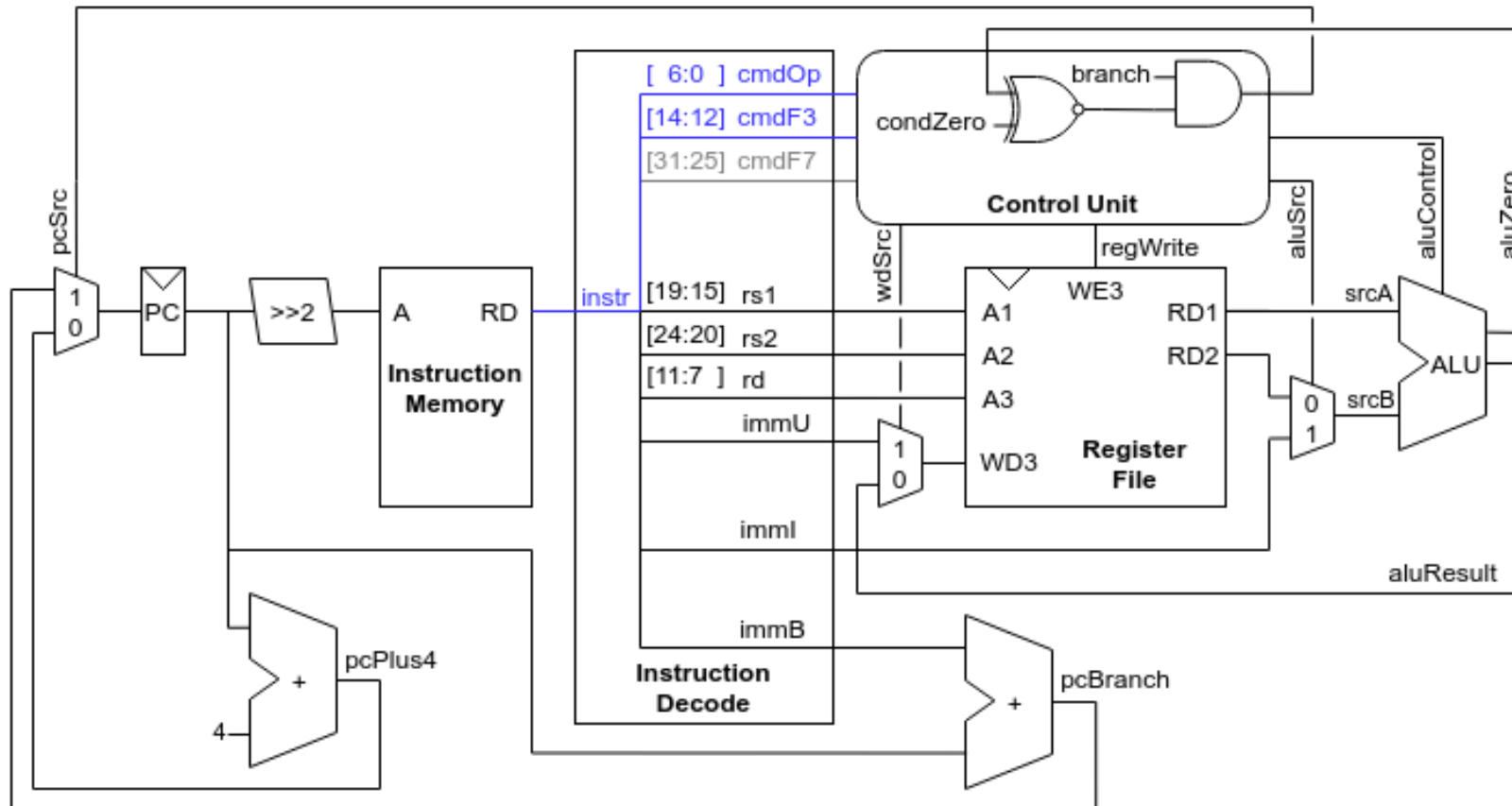
Сигналы управления 21

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	???????	0	000	1	1	0	0	0
BEQ	1100011	000	???????	0	100	0	0	aluZero	1	1



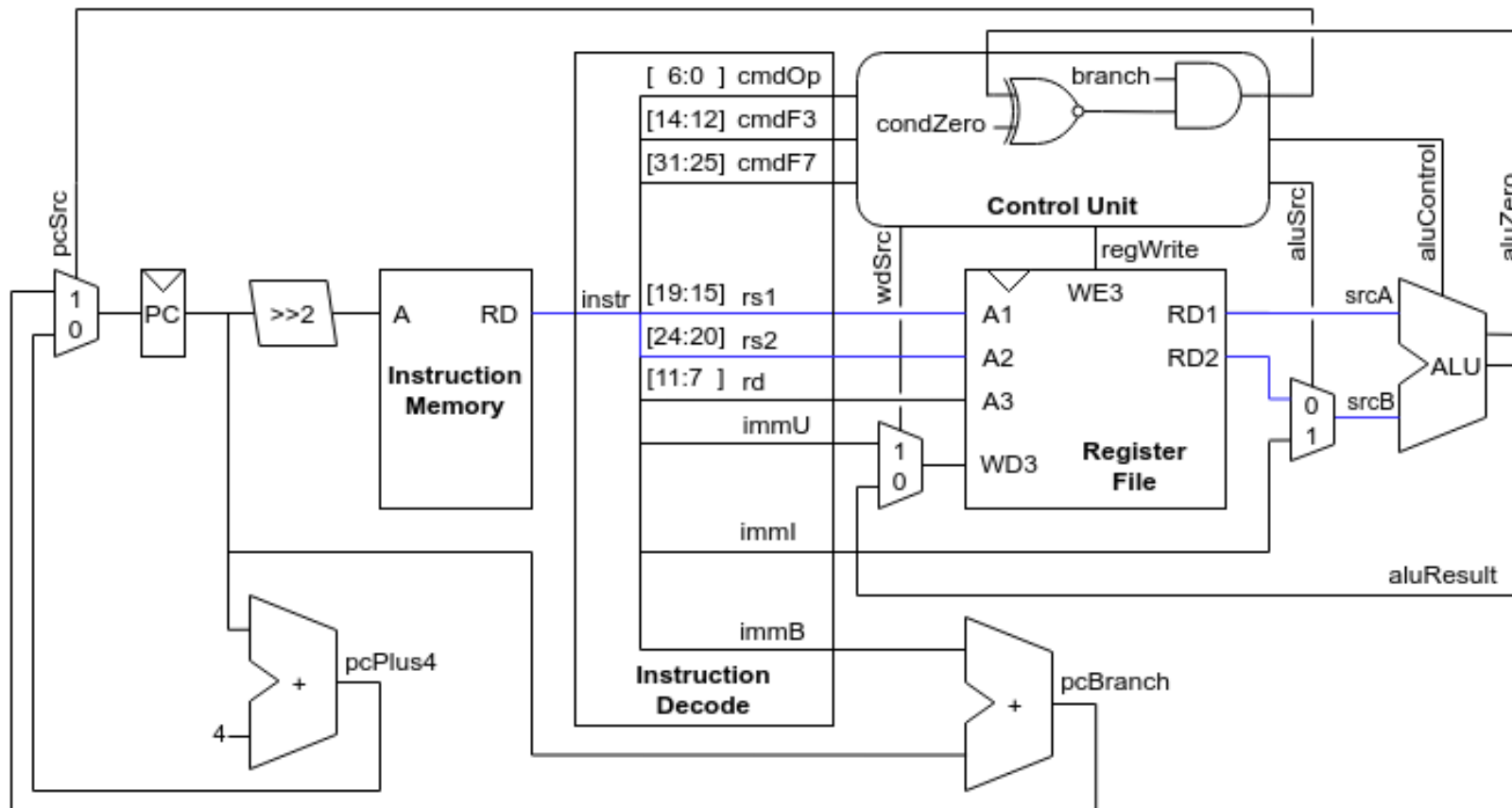
Сигналы управления 22

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	????????	0	000	1	1	0	0	0
BEQ	1100011	000	????????	0	100	0	0	aluZero	1	1
BNE	1100011	001	????????							



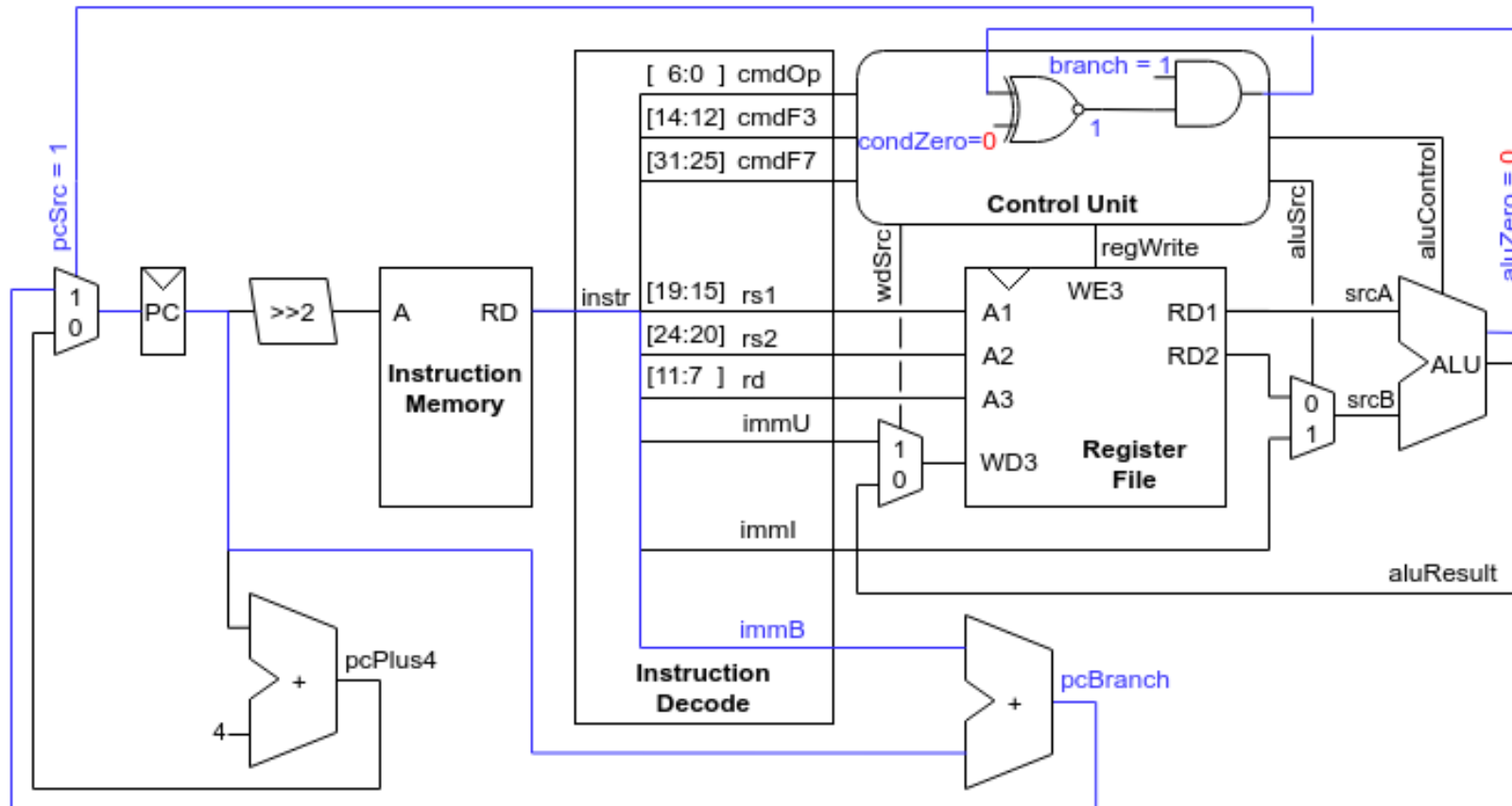
Сигналы управления 23

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	???????	0	000	1	1	0	0	0
BEQ	1100011	000	???????	0	100	0	0	aluZero	1	1
BNE	1100011	001	???????	0	100	0	0			



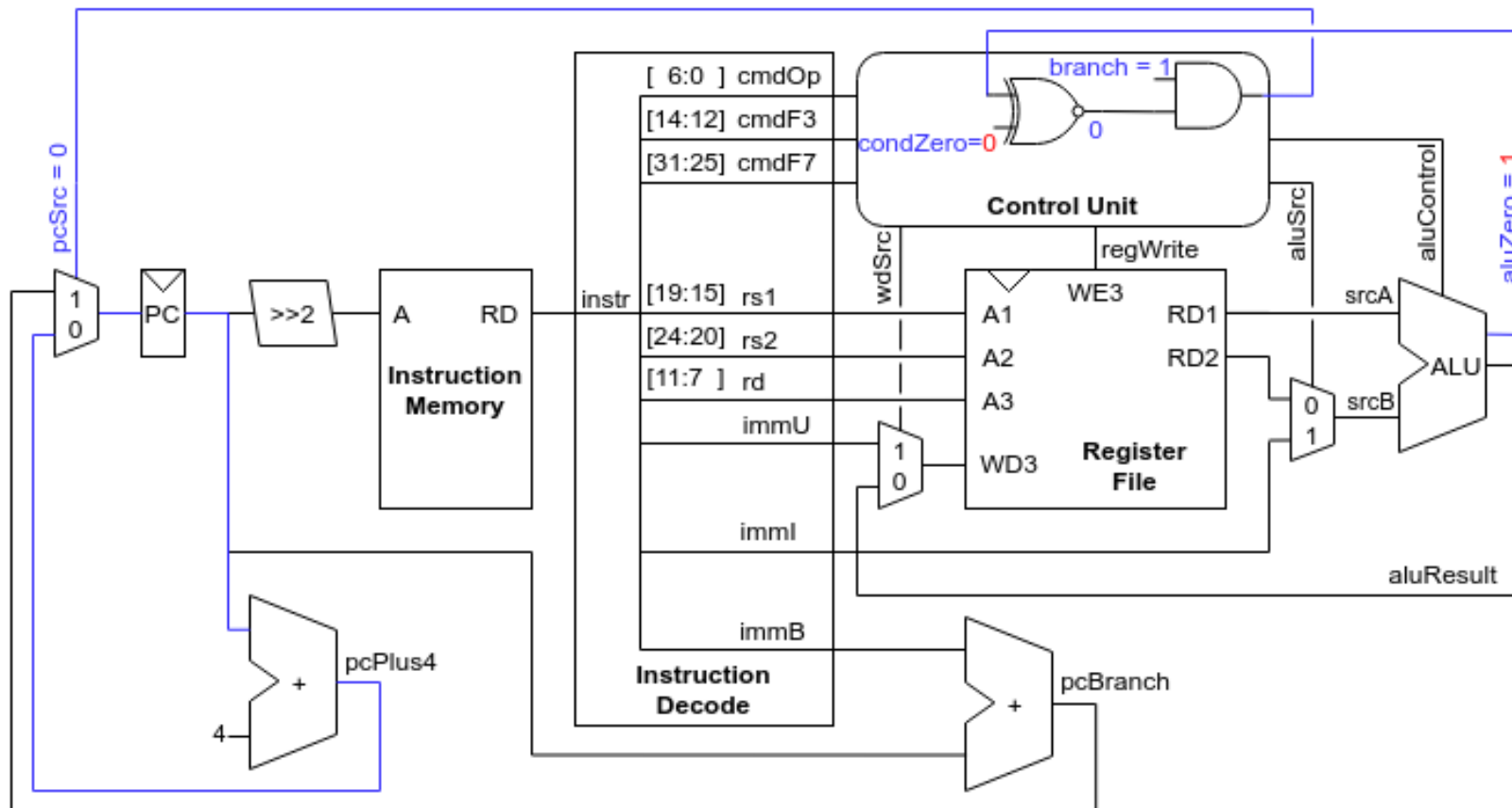
Сигналы управления 24

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	????????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	????????	0	000	1	1	0	0	0
BEQ	1100011	000	????????	0	100	0	0	aluZero	1	1
BNE	1100011	001	????????	0	100	0	0	~aluZero	1	0



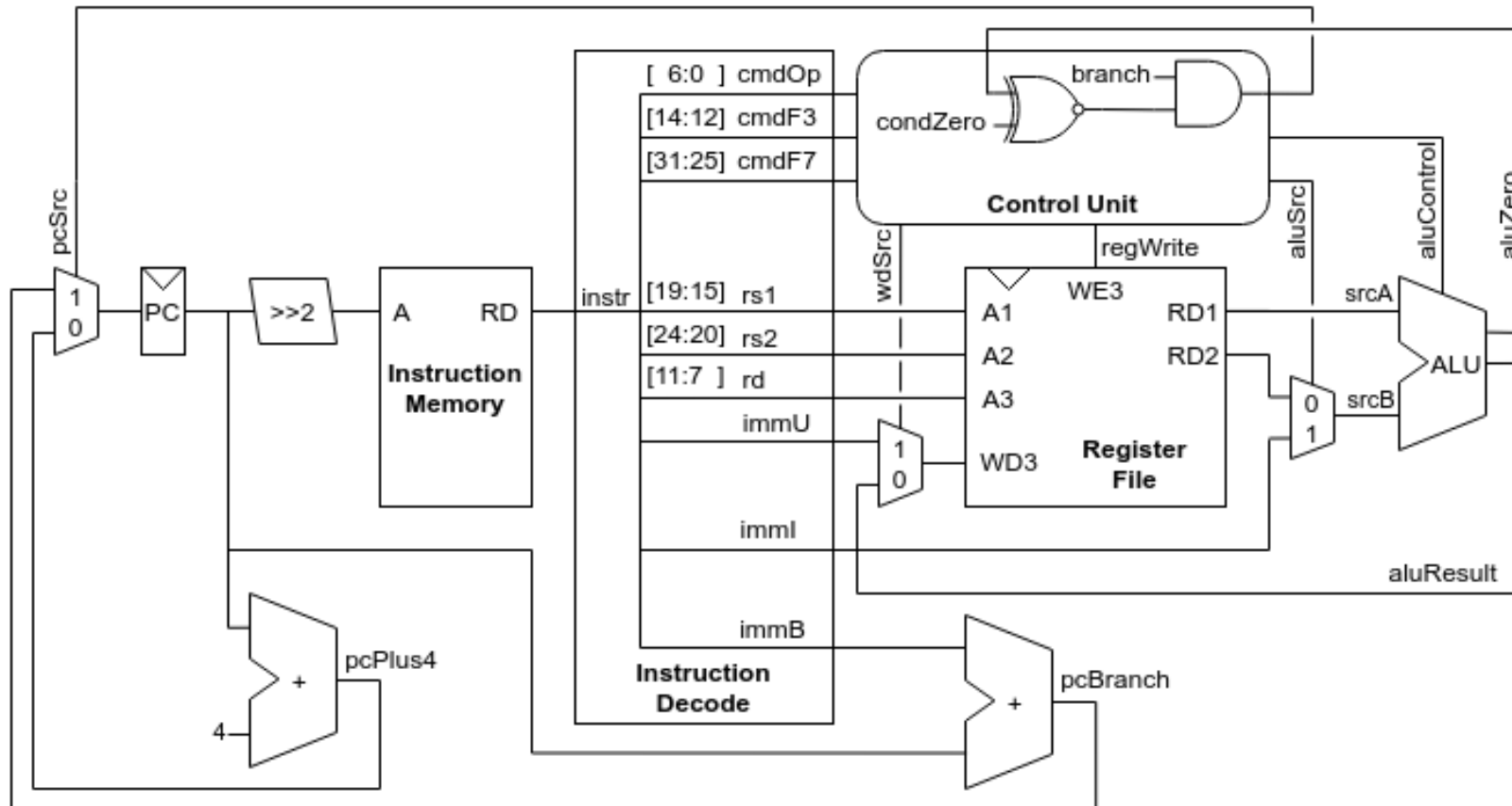
Сигналы управления 25

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	???????	0	000	1	1	0	0	0
BEQ	1100011	000	???????	0	100	0	0	aluZero	1	1
BNE	1100011	001	???????	0	100	0	0	~aluZero	1	0



Сигналы управления 26

Instruction	cmdOp	cmdF3	cmdF7	aluSrc	aluControl	wdSrc	regWrite	pcSrc	branch	condZero
ADDI	0010011	000	???????	1	000	0	1	0	0	0
ADD	0110011	000	0000000	0	000	0	1	0	0	0
LUI	0110111	???	???????	0	000	1	1	0	0	0
BEQ	1100011	000	???????	0	100	0	0	aluZero	1	1
BNE	1100011	001	???????	0	100	0	0	~aluZero	1	0



Реализация: коды инструкций

```
// sr_cpu.vh
// instruction opcode
`define RVOP_ADDI    7'b0010011
`define RVOP_BEQ     7'b1100011
...

// instruction funct3
`define RVF3_ADDI    3'b000
`define RVF3_BEQ     3'b000
`define RVF3_BNE     3'b001
`define RVF3_ADD      3'b000
...
`define RVF3_ANY     3'b???

// instruction funct7
`define RVF7_ADD     7'b0000000
...
`define RVF7_ANY     7'b???????

```

Реализация: устройство управления (начало)

```
// sr_cpu.v
module sr_control
(
    input      [ 6:0] cmdOp,
    input      [ 2:0] cmdF3,
    input      [ 6:0] cmdF7,
    input      aluZero,
    output      pcSrc,
    output reg   regWrite,
    output reg   aluSrc,
    output reg   wdSrc,
    output reg [2:0] aluControl
);
    reg          branch;
    reg          condZero;
    assign pcSrc = branch & (aluZero == condZero);
```

Реализация: устройство управления (продолжение)

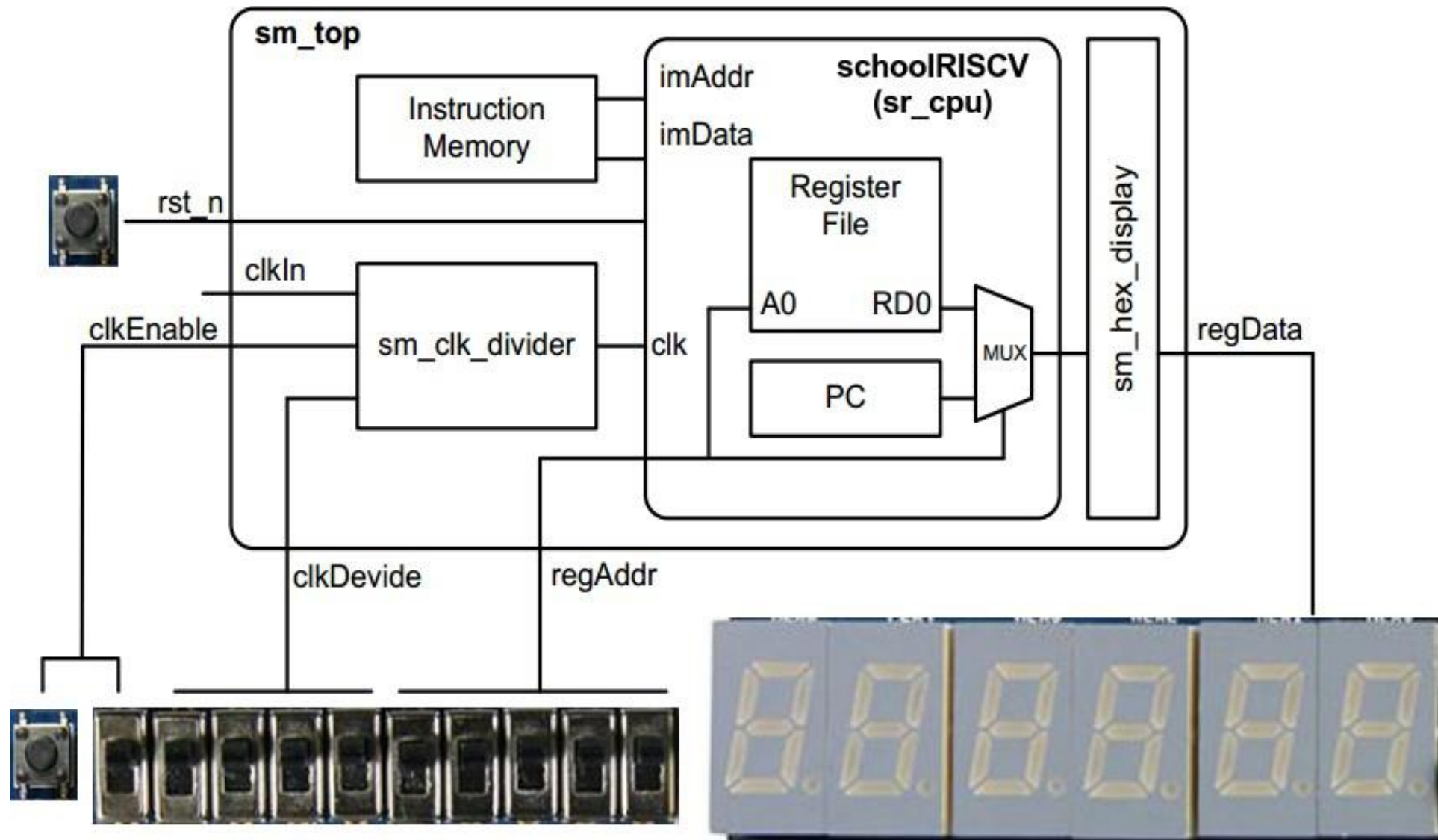
```
// sr_cpu.v
always @ (*) begin
    branch      = 1'b0;
    condZero    = 1'b0;
    regWrite    = 1'b0;
    aluSrc      = 1'b0;
    wdSrc       = 1'b0;
    aluControl   = `ALU_ADD;

    casez( {cmdF7, cmdF3, cmdOp} )
        { `RVF7_ADD,  `RVF3_ADD,  `RVOP_ADD  } : begin regWrite = 1'b1; aluControl = `ALU_ADD; end
        { `RVF7_OR,   `RVF3_OR,   `RVOP_OR   } : begin regWrite = 1'b1; aluControl = `ALU_OR; end
        { `RVF7_SRL,  `RVF3_SRL,  `RVOP_SRL  } : begin regWrite = 1'b1; aluControl = `ALU_SRL; end
        { `RVF7_SLTU, `RVF3_SLTU, `RVOP_SLTU } : begin regWrite = 1'b1; aluControl = `ALU_SLTU; end
        { `RVF7_SUB,  `RVF3_SUB,  `RVOP_SUB  } : begin regWrite = 1'b1; aluControl = `ALU_SUB; end

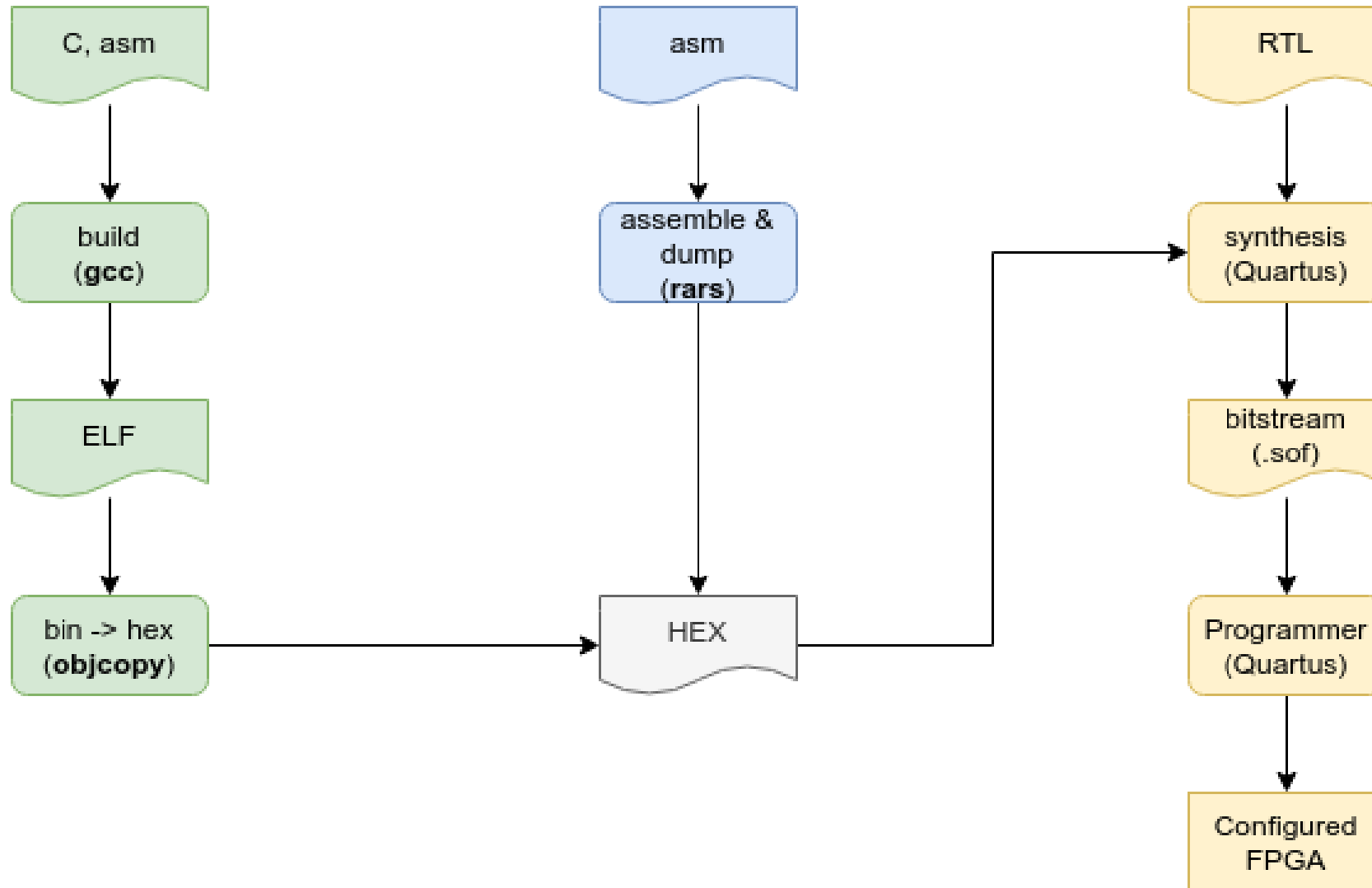
        { `RVF7_ANY,  `RVF3_ADDI, `RVOP_ADDI } : begin regWrite = 1'b1; aluSrc = 1'b1; aluControl = `ALU_ADD; end
        { `RVF7_ANY,  `RVF3_ANY,  `RVOP_LUI  } : begin regWrite = 1'b1; wdSrc  = 1'b1; end

        { `RVF7_ANY,  `RVF3_BEQ,  `RVOP_BEQ  } : begin branch = 1'b1; condZero = 1'b1; aluControl = `ALU_SUB; end
        { `RVF7_ANY,  `RVF3_BNE,  `RVOP_BNE  } : begin branch = 1'b1; aluControl = `ALU_SUB; end
    endcase
end
```

Структура проекта и подключение периферии



Программирование системы



Что дальше?

- Цифровая схемотехника и архитектура компьютера
David Harris & Sarah Harris
ДМК Пресс
- Цифровой синтез: практический курс
Александр Романов & Юрий Панчул
ДМК Пресс
- Syntacore SCR1
<https://github.com/syntacore/scr1>
- Learning RISC-V
<https://riscv.org/community/learn-about-risc-v/>

Ваши вопросы?