Схемотехника
(углубленный курс)

Лекция №5

Таблица истинности основного дешифратора с поддержкой **addi**

Команда	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
Команды типа R	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01
`addi`	001000	1	0	1	0	0	0	00

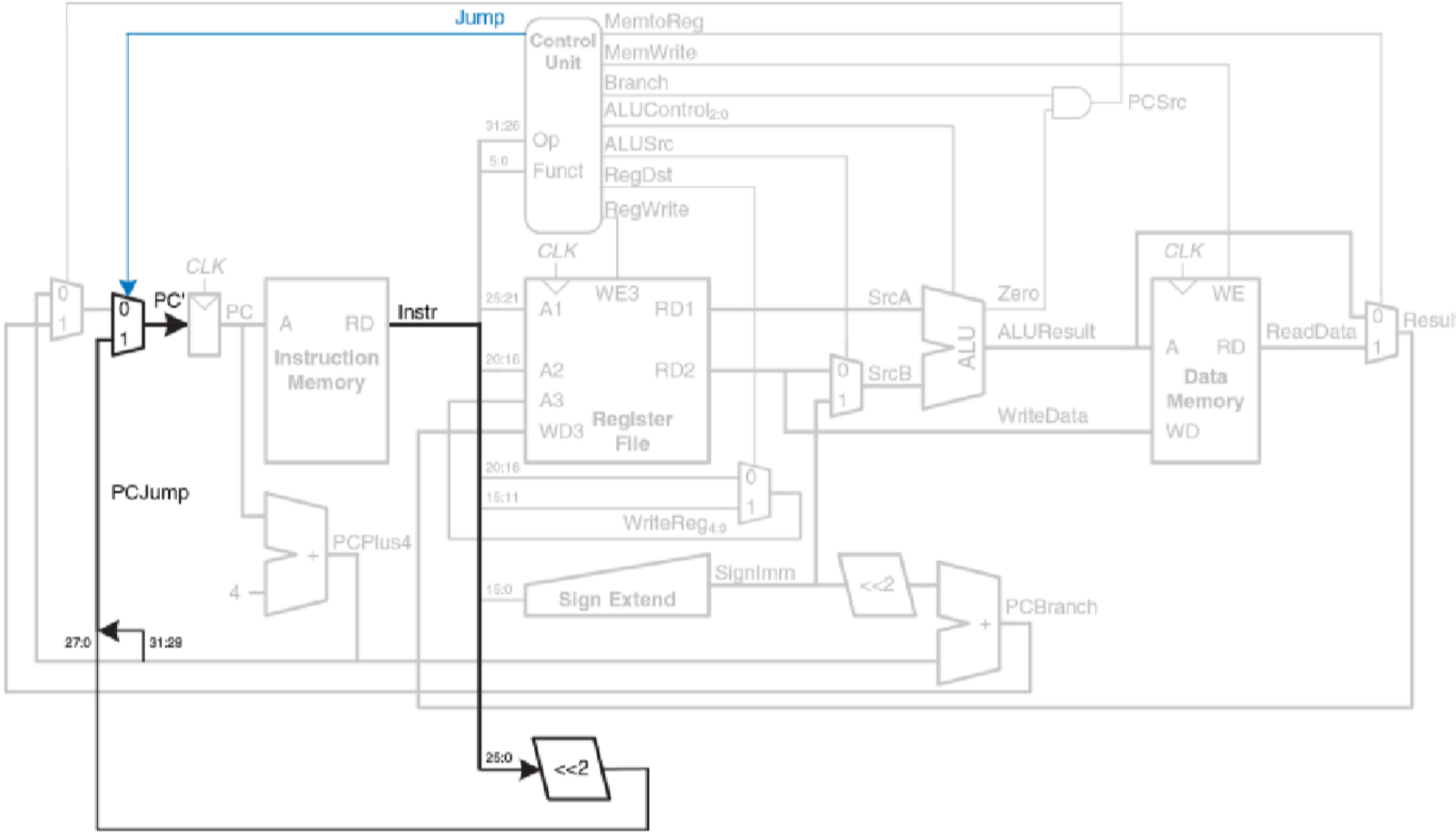
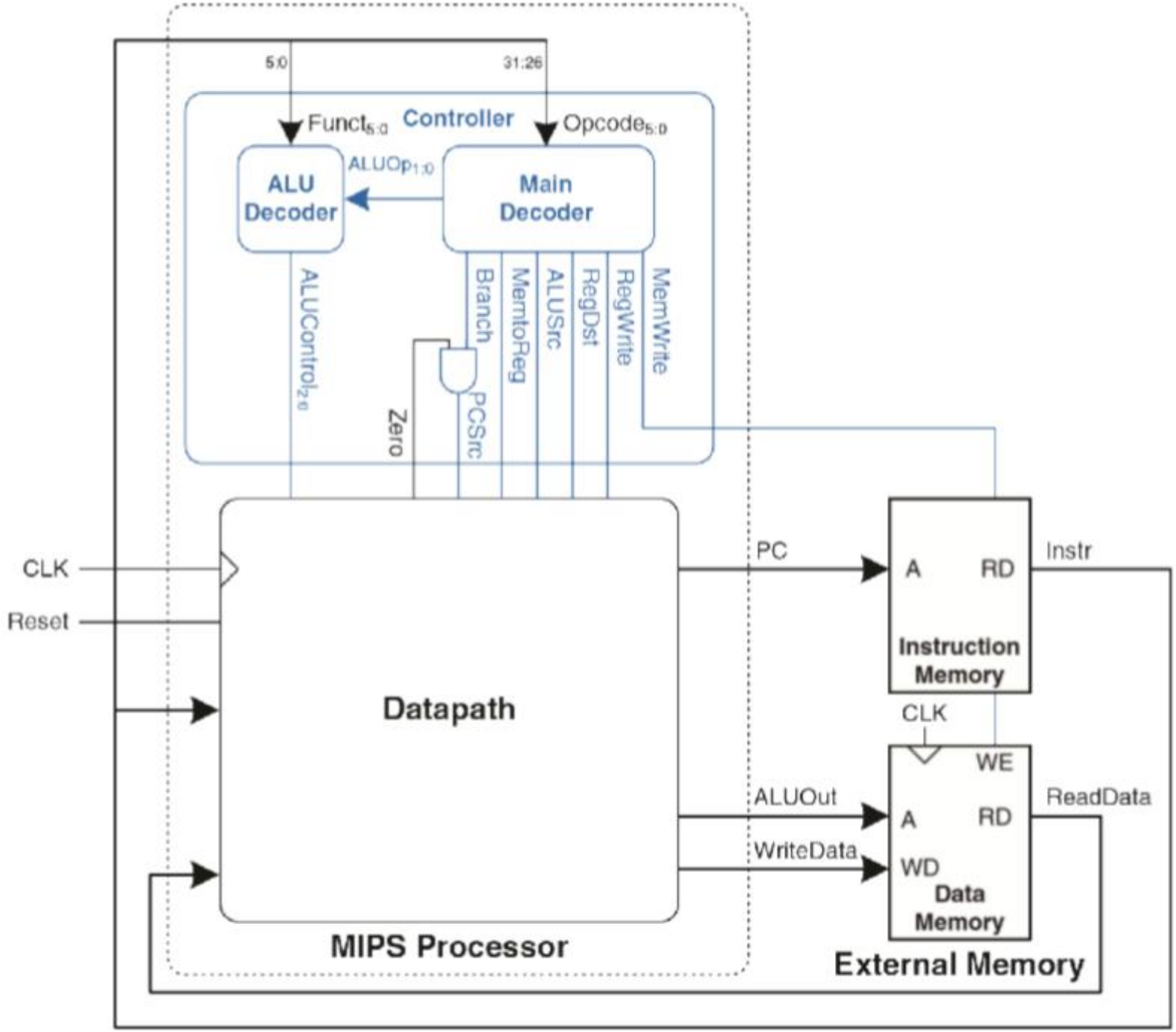

Jump
Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
A
RD
Data Memory
WD
ReadData
Result
PCJump
PCPlus4
4
20:16
15:11
WriteReg4:0
15:0
Sign Extend
SignImm
<<2
PCBranch
27:0
31:28
25:0
<<2

Таблица истинности основного дешифратора с поддержкой j

Команда	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp	Jump
Команды типа R	000000	1	1	0	0	0	0	10	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
addi	001000	1	0	1	0	0	0	00	0
j	000010	0	X	X	X	0	X	XX	1


5:0
31:26
Funct5:0
Controller
Opcode5:0
ALU Decoder
ALUOp1:0
Main Decoder
ALUControl2:0
Branch
MemtoReg
ALUSrc
RegDst
RegWrite
MemWrite
Zero
PCSrc
CLK
Reset
Datapath
MIPS Processor
PC
A
RD
Instr
Instruction Memory
CLK
WE
ALUOut
WriteData
A
RD
ReadData
WD
Data Memory
External Memory

Модуль верхнего уровня

```
module top(
    input logic clk, reset,
    output logic [31:0] writedata, dataadr,
    output logic memwrite
    );

    logic [31:0] pc, instr, readdata;

    mips mips(clk, reset, pc, instr, memwrite, dataadr,
              writedata, readdata);
    imem imem(pc[7:2], instr);
    dmem dmem(clk, memwrite, dataadr, writedata, readdata);

endmodule
```

Память инструкций

```
module imem(
    input logic [5:0] a,
    output logic [31:0] rd
    );

    logic [31:0] RAM[63:0];

    initial
        $readmemh("memfile.dat", RAM);

    assign rd = RAM[a];

endmodule
```

ОЗУ

```
module dmem(
    input logic clk, we,
    input logic [31:0] a, wd,
    output logic [31:0] rd
    );

    logic [31:0] RAM[63:0];

    assign rd = RAM[a[31:2]];

    always_ff @(posedge clk)
        if (we) RAM[a[31:2]] <= wd;

endmodule
```

Модуль процессора

```
module mips(
    input logic clk, reset,
    output logic [31:0] pc,
    input logic [31:0] instr,
    output logic memwrite,
    output logic [31:0] aluout, writedata,
    input logic [31:0] readdata
    );

    logic memtoreg, alusrc, regdst, regwrite, jump, pcsrc, zero;
    logic [2:0] alucontrol;
    controller c(
       instr[31:26], instr[5:0], zero, memtoreg, memwrite, pcsrc,
       alusrc, regdst, regwrite, jump, alucontrol);

    datapath dp(
       clk, reset, memtoreg, pcsrc, alusrc, regdst, regwrite, jump,
       alucontrol, zero, pc, instr, aluout, writedata, readdata);

endmodule
```

Управляющее устройство

```
module controller(
    input logic [5:0] op, funct,
    input logic zero,
    output logic memtoreg, memwrite,
    output logic pcsrc, alusrc,
    output logic regdst, regwrite,
    output logic jump,
    output logic [2:0] alucontrol
    );

    logic [1:0] aluop;
    logic branch;

    maindec md(op, memtoreg, memwrite, branch, alusrc,
               regdst, regwrite, jump, aluop);

    aludec ad(funct, aluop, alucontrol);

    assign pcsrc = branch & zero;

endmodule
```

Главный дешифратор

```
module maindec(
    input logic [5:0] op,
    output logic memtoreg, memwrite,
    output logic branch, alusrc,
    output logic regdst, regwrite,
    output logic jump,
    output logic [1:0] aluop
    );

    logic [8:0] controls;

    assign {regwrite, regdst, alusrc, branch, memwrite,
            memtoreg, jump, aluop} = controls;

    always_comb
       case(op)
          6'b000000: controls <= 9'b110000010; // RTYPE
          6'b100011: controls <= 9'b101001000; // LW
          6'b101011: controls <= 9'b001010000; // SW
          6'b000100: controls <= 9'b000100001; // BEQ
          6'b001000: controls <= 9'b101000000; // ADDI
          6'b000010: controls <= 9'b000000100; // J
          default: controls <= 9'bxxxxxxxxx; // illegal op
       endcase

endmodule
```

Дешифратор АЛУ

```
module aludec(
    input logic [5:0] funct,
    input logic [1:0] aluop,
    output logic [2:0] alucontrol
    );

    always_comb
        case(aluop)
            2'b00: alucontrol <= 3'b010; // add (for lw/sw/addi)
            2'b01: alucontrol <= 3'b110; // sub (for beq)
            default:
                case(funct) // R-type instructions
                    6'b100000: alucontrol <= 3'b010; // add
                    6'b100010: alucontrol <= 3'b110; // sub
                    6'b100100: alucontrol <= 3'b000; // and
                    6'b100101: alucontrol <= 3'b001; // or
                    6'b101010: alucontrol <= 3'b111; // slt
                    default: alucontrol <= 3'bxxx; // ???
                endcase
        endcase
endmodule
```

```
input logic clk, reset,
input logic memtoreg, pcsrc,
input logic alusrc, regdst,
input logic regwrite, jump,
input logic [2:0] alucontrol,
output logic zero,
output logic [31:0] pc,
input  logic [31:0] instr,
output logic [31:0] aluout, writedata,
input  logic [31:0] readdata
);

logic [4:0] writereg;
logic [31:0] pcnext, pcnextbr, pcplus4, pcbranch;
logic [31:0] signimm, signimmsh;
logic [31:0] srca, srcb;
logic [31:0] result;

// next PC logic
flopr #(32) pcreg(clk, reset, pcnext, pc);
adder       pcadd1(pc, 32'b100, pcplus4);
sl2         immsh(signimm, signimmsh);
adder       pcadd2(pcplus4, signimmsh, pcbranch);
mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28], instr[25:0], 2'b00}, jump, pcnext);

// register file logic
regfile     rf(clk, regwrite, instr[25:21], instr[20:16], writereg, result, srca, writedata);
mux2 #(5)   wrmux(instr[20:16], instr[15:11], regdst, writereg);
mux2 #(32)  resmux(aluout, readdata, memtoreg, result);
signext     se(instr[15:0], signimm);

// ALU logic
mux2 #(32)  srcbmux(writedata, signimm, alusrc, srcb);
alu #(32)   alu(srca, srcb, alucontrol, aluout, zero);
```

Файл регистров

```
module regfile(
    input logic clk,
    input logic we3,
    input logic [4:0] ra1, ra2, wa3,
    input logic [31:0] wd3,
    output logic [31:0] rd1, rd2
    );

    logic [31:0] rf[31:0];

    always_ff @(posedge clk)
       if (we3) rf[wa3] <= wd3;

    assign rd1 = (ra1 != 0) ? rf[ra1] : 0;
    assign rd2 = (ra2 != 0) ? rf[ra2] : 0;

endmodule
```

АЛУ

```
module alu #(parameter WIDTH = 4)(
   input logic [WIDTH-1:0] operandA, operandB,
   input logic [2:0] funcsel,
   output logic [WIDTH-1:0] result,
   output logic zero
   );

   logic [WIDTH-1:0] maybeinvertedB;
   assign maybeinvertedB = funcsel[2] ? ~operandB : operandB;

   logic [WIDTH-1:0] sum;
   assign sum = operandA + maybeinvertedB + funcsel[2];

   always_comb
      case(funcsel[1:0])
         2'b00: result <= operandA & maybeinvertedB;
         2'b01: result <= operandA | maybeinvertedB;
         2'b10: result <= sum;
         2'b11: result <= {{WIDTH-1{1'b0}}, sum[WIDTH-1]};
      endcase

   assign zero = (result == {WIDTH{1'b0}}) ? 1'b1 : 1'b0 ;

endmodule
```

```
module adder(
    input logic [31:0] a, b,
    output logic [31:0] y
    );

    assign y = a + b;

endmodule

module sl2(
    input logic [31:0] a,
    output logic [31:0] y
    );

    // shift left by 2
    assign y = {a[29:0], 2'b00};

endmodule

module signext(
    input logic [15:0] a,
    output logic [31:0] y
    );

    assign y = {{16{a[15]}}, a};

endmodule
```

Сумматор, сдвиг влево и знаковое увеличение разрядности

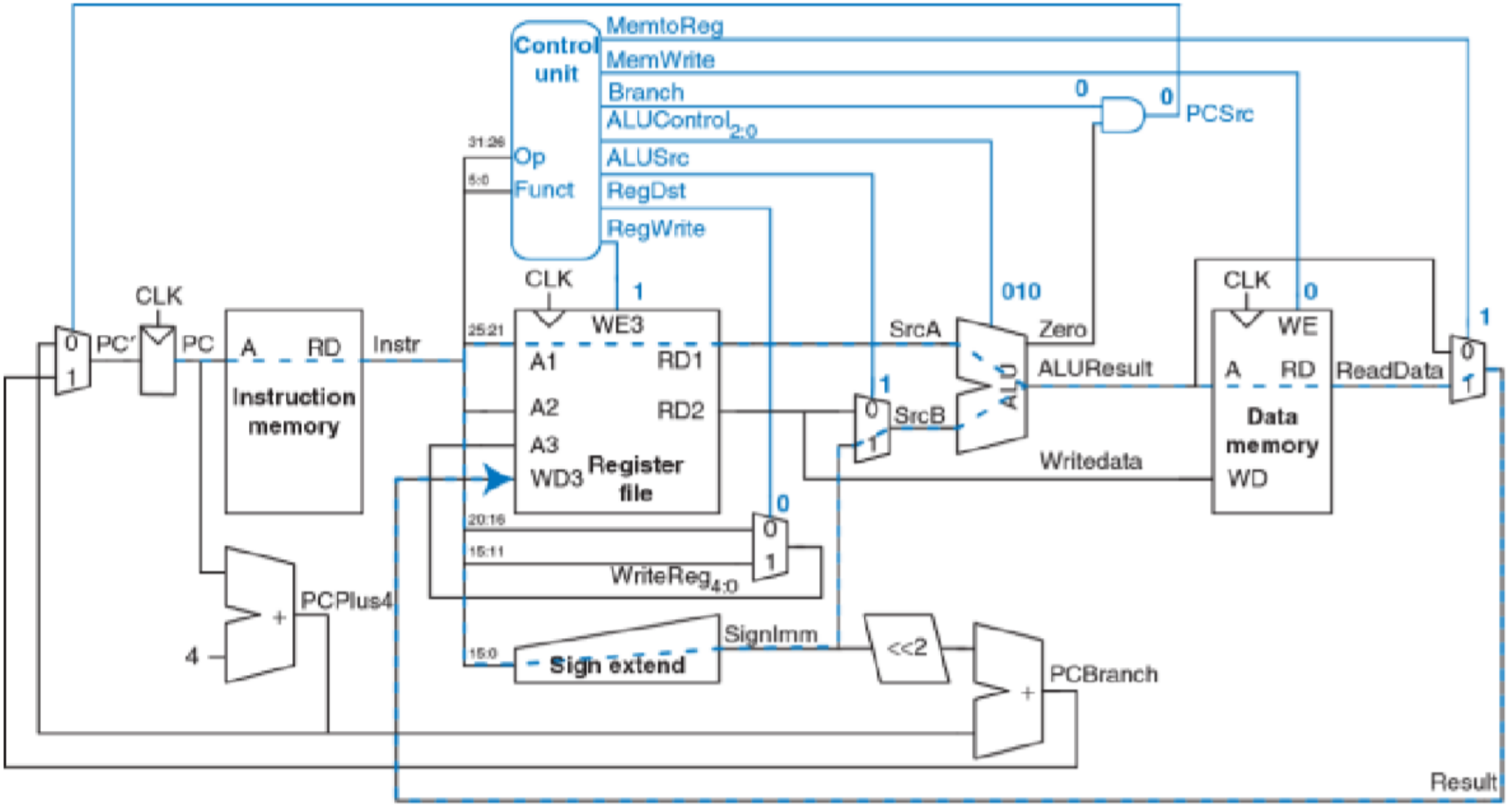
Регистр и мультиплексор

```
module flopr #(parameter WIDTH = 8)(
   input logic clk, reset,
   input logic [WIDTH-1:0] d,
   output logic [WIDTH-1:0] q
   );

   always_ff @(posedge clk, posedge reset)
      if (reset)
         q <= 0;
      else
         q <= d;

endmodule

module mux2 #(parameter WIDTH = 8)(
   input logic [WIDTH-1:0] d0, d1,
   input logic s,
   output logic [WIDTH-1:0] y
   );

   assign y = s ? d1 : d0;

endmodule
```

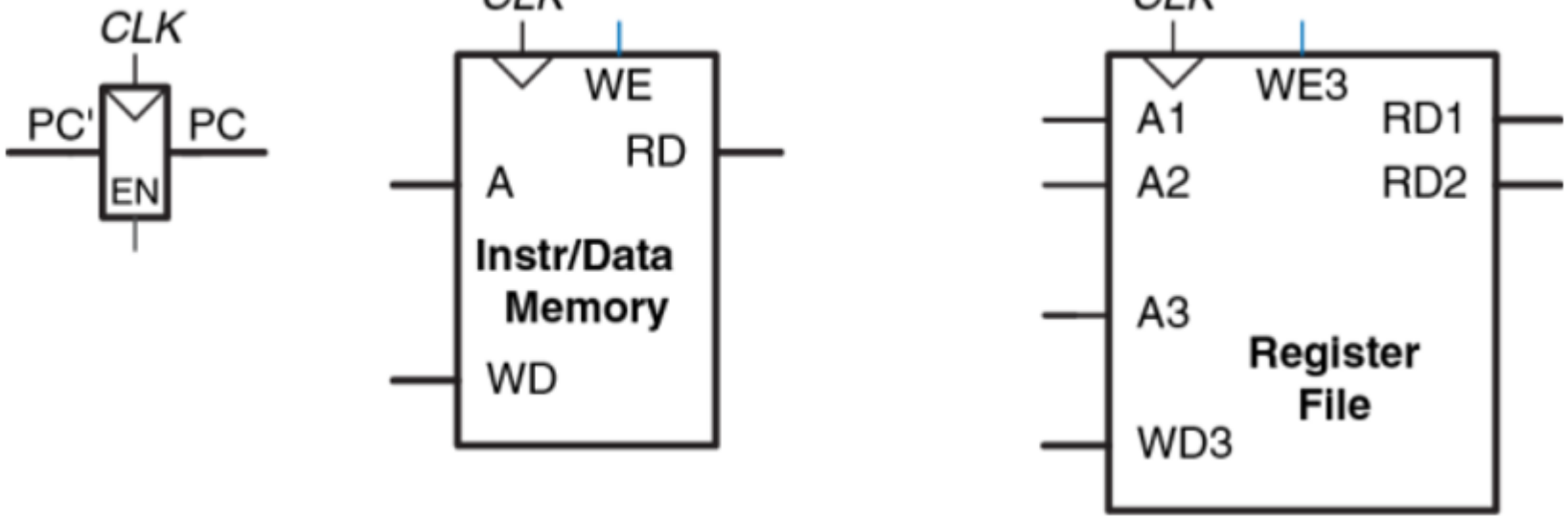
Тестовое окружение

```
module testbench();

   logic clk;
   logic reset;
   logic [31:0] writedata, dataadr;
   logic memwrite;

   // instantiate device to be tested
   top dut (clk, reset, writedata, dataadr, memwrite);

   // initialize test
   initial begin
      reset <= 1; # 22; reset <= 0;
   end

   // generate clock to sequence tests
   always begin
      clk <= 1; # 5; clk <= 0; # 5;
   end

   // check results
   always @(negedge clk) begin
      if (memwrite) begin
         if (dataadr === 84 & writedata === 7) begin
            $display("Simulation succeeded");
            $stop;
         end else if (dataadr !== 80) begin
            $display("Simulation failed");
            $stop;
         end
      end
   end

endmodule
```

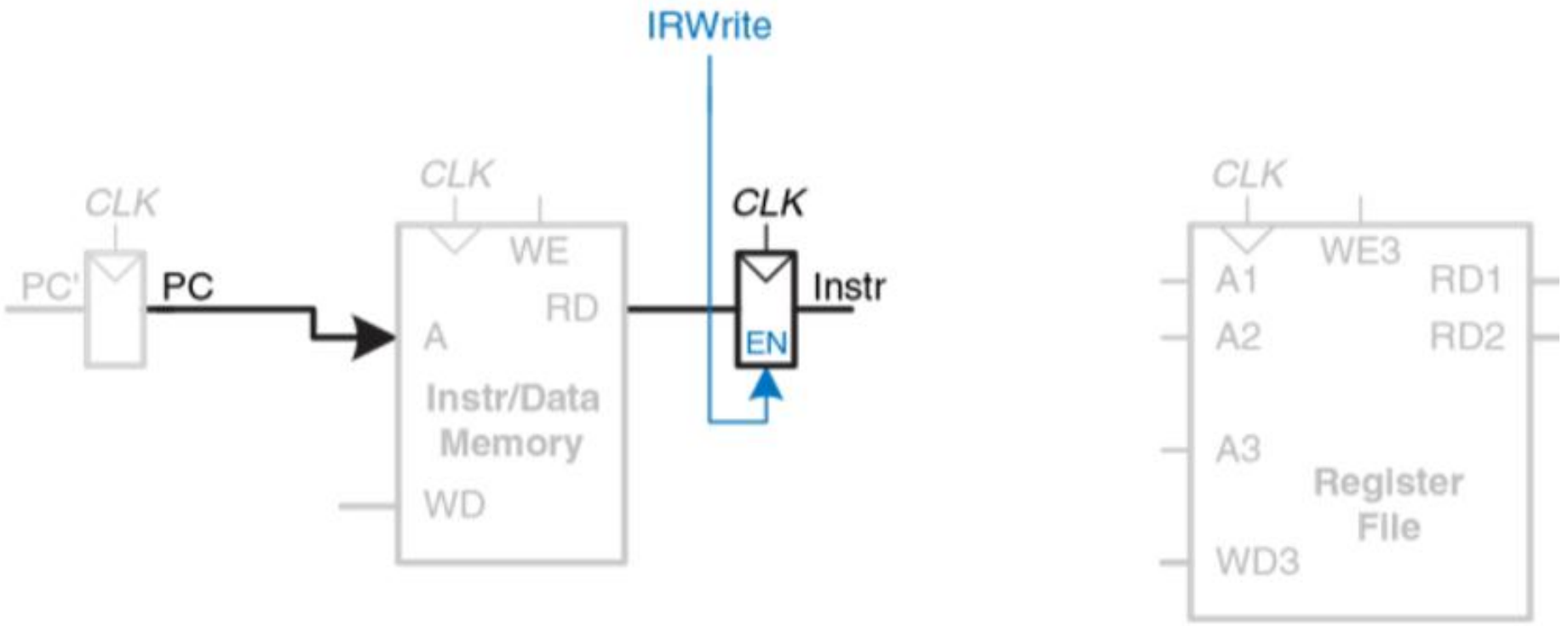
Цепь с наибольшей задержкой для команды **lw**



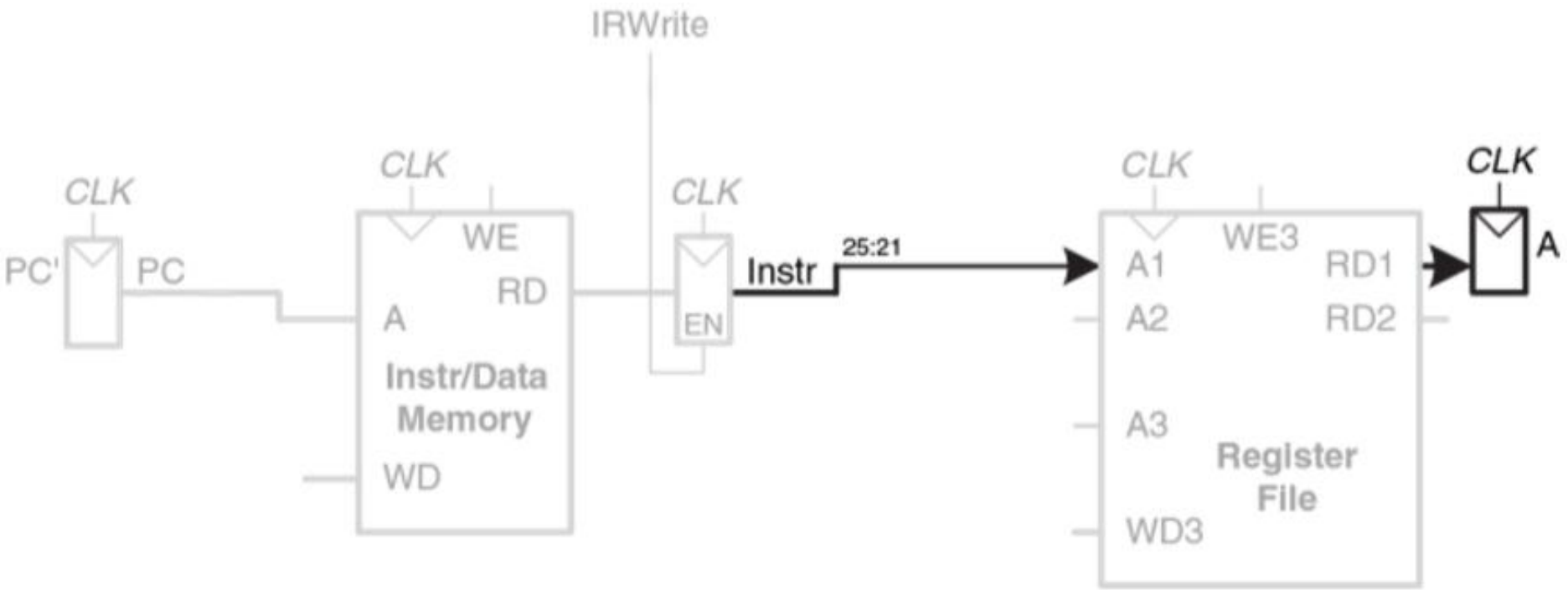
Хранящие архитектурное состояние элементы. Общая память команд и данных.



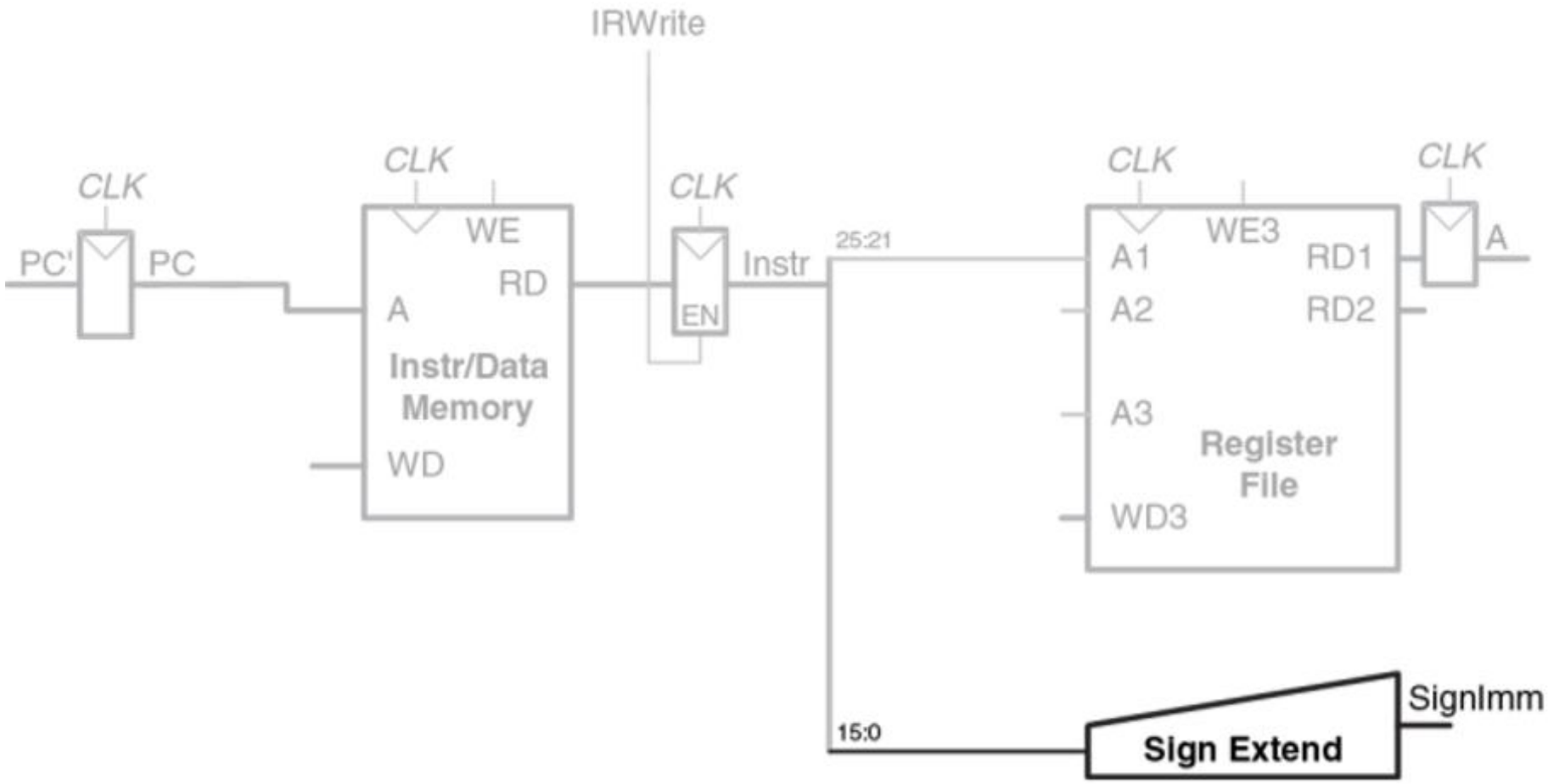
Выборка команды из памяти



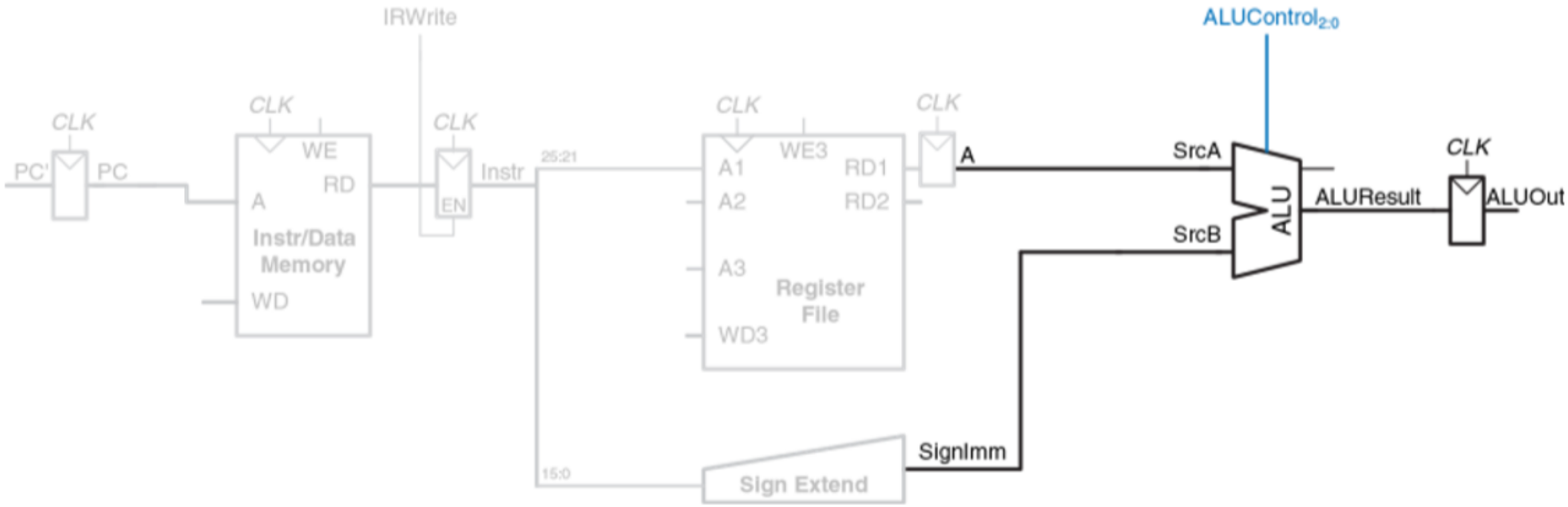
Чтение операнда команды **lw** из регистрового файла



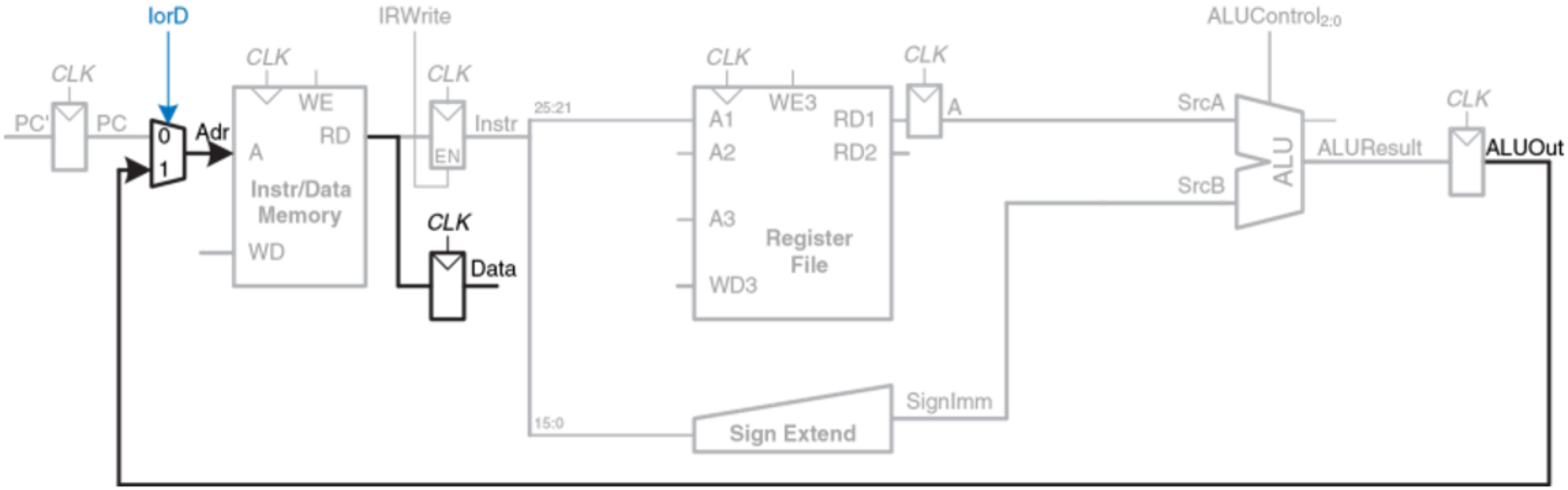
Знаковое расширение непосредственного операнда



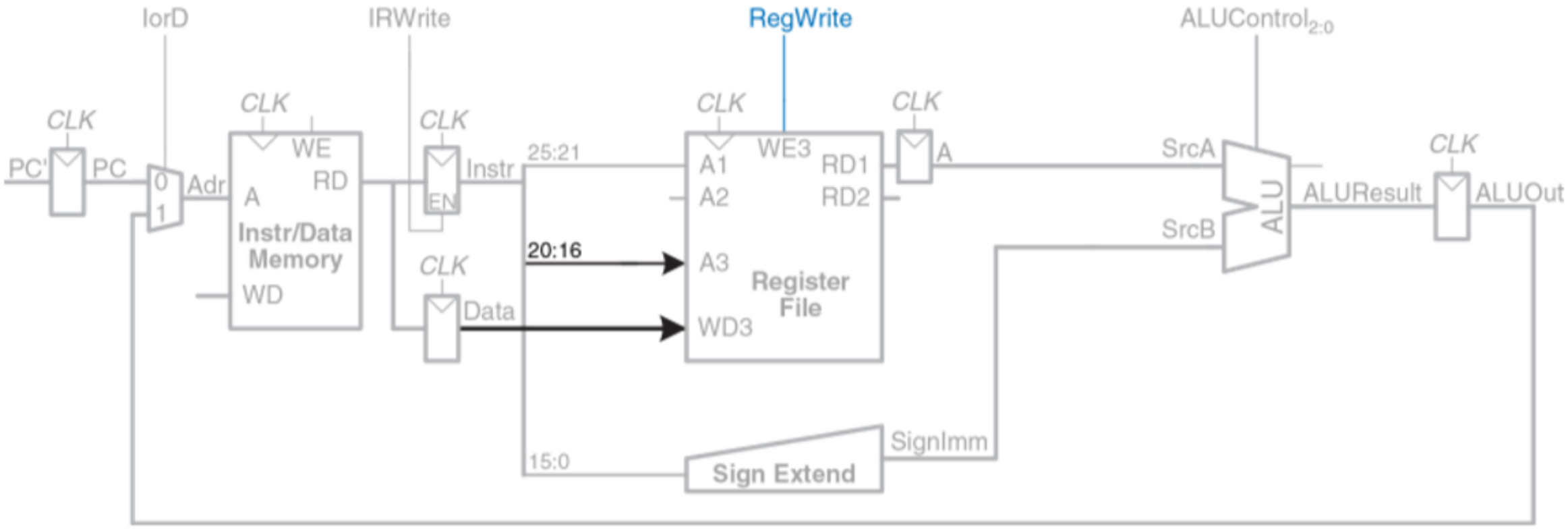
Сложение базового адреса и смещения



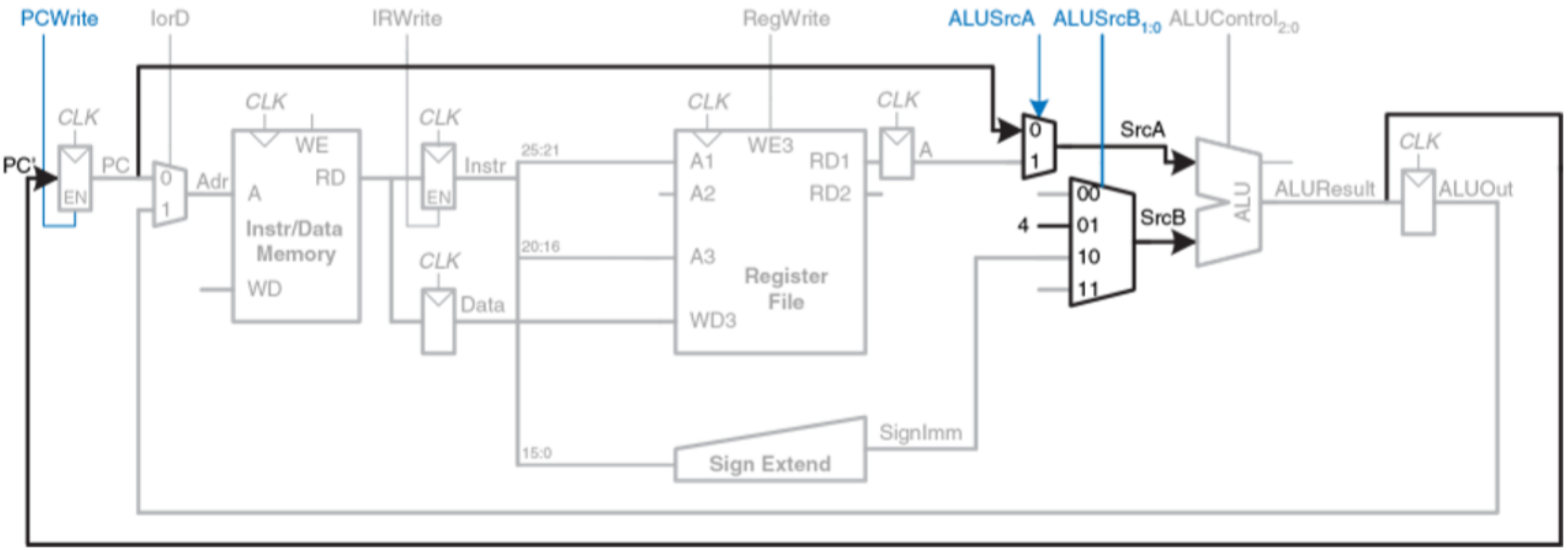
Загрузка данных из памяти



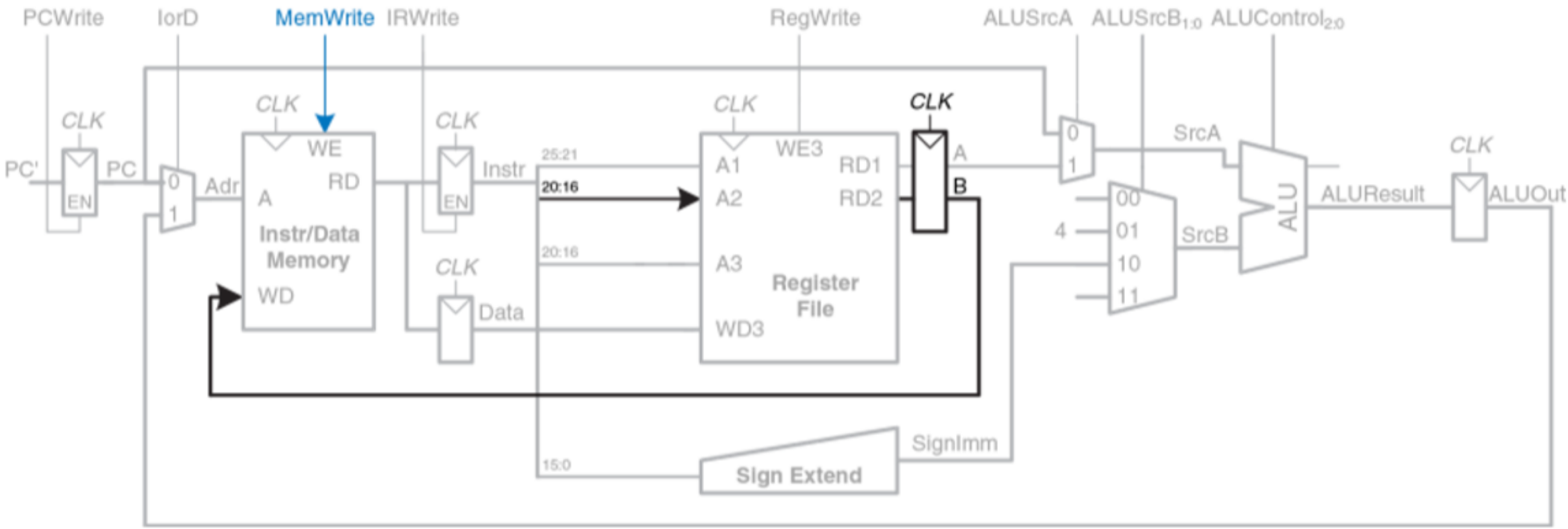
Запись данных в регистровый файл



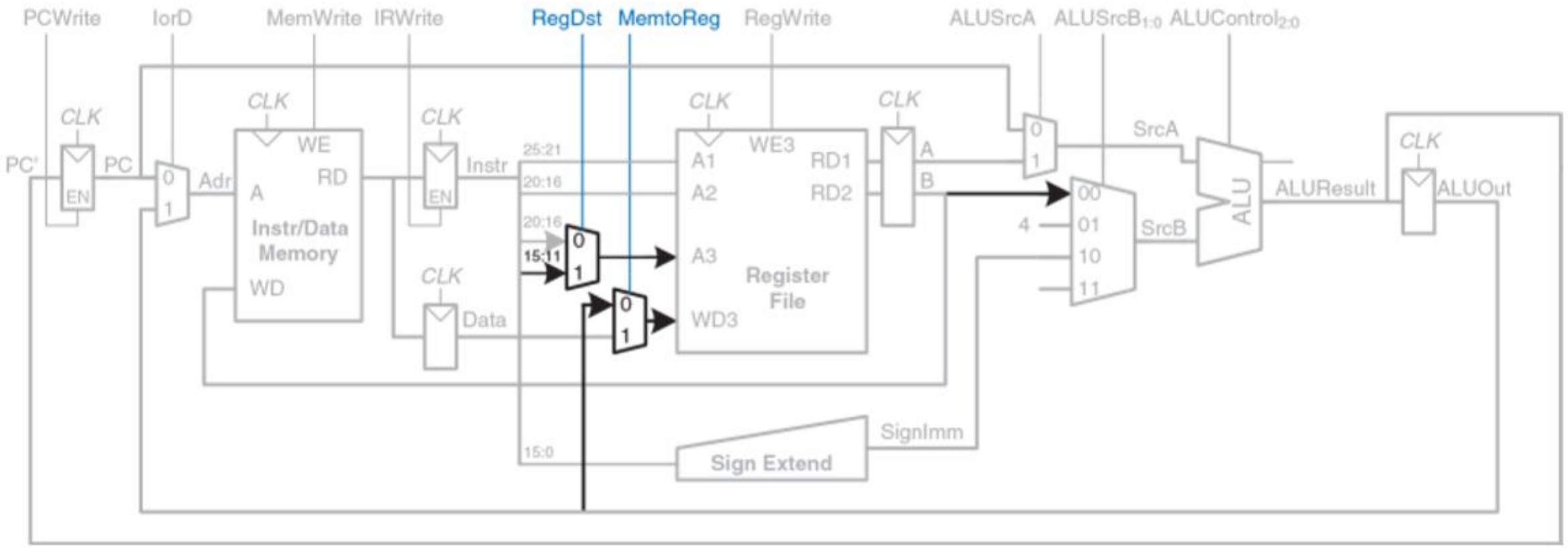
Увеличение счетчика команд на четыре



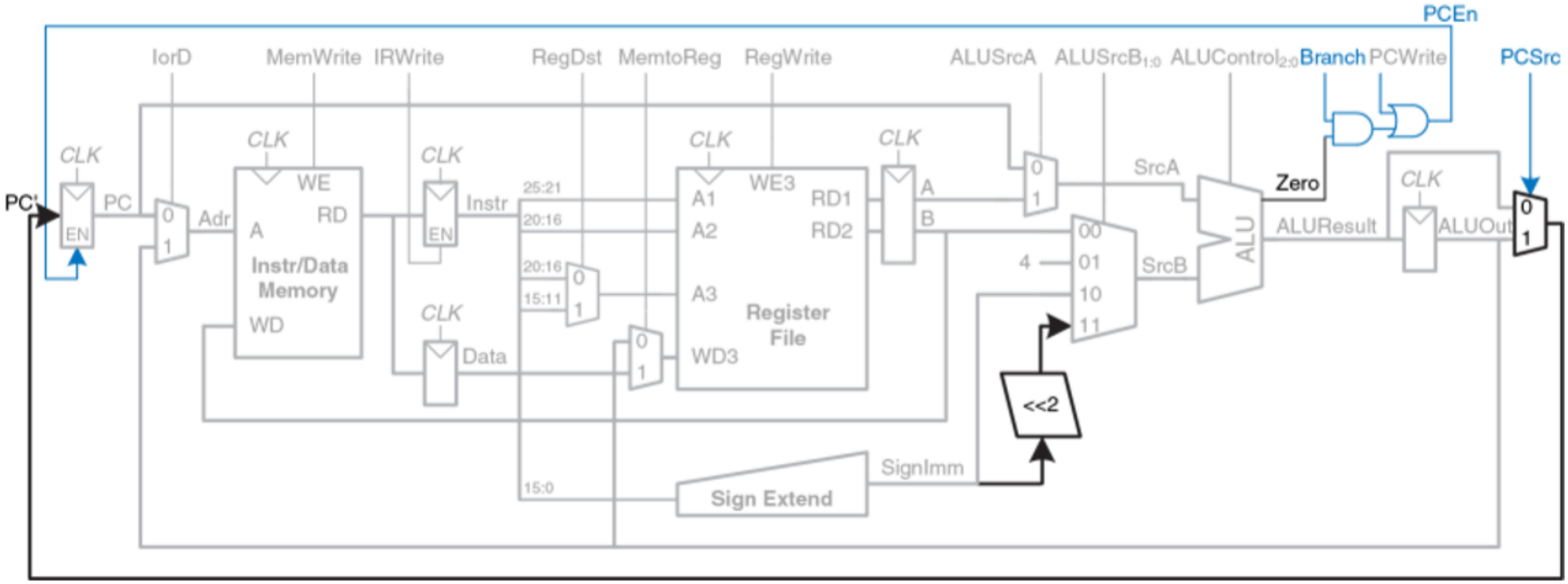
Поддержка команды **SW**

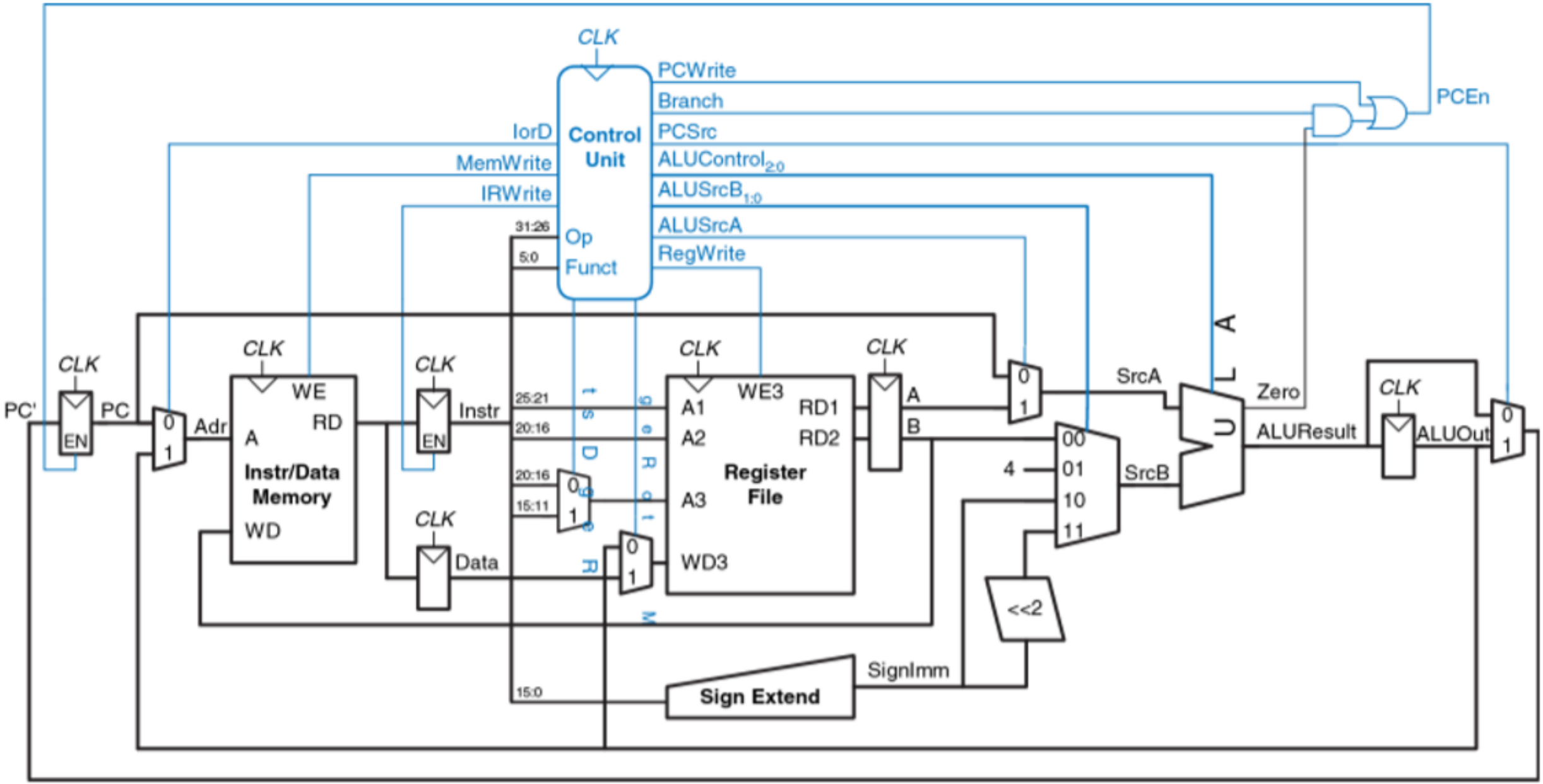


Поддержка команд типа **R**

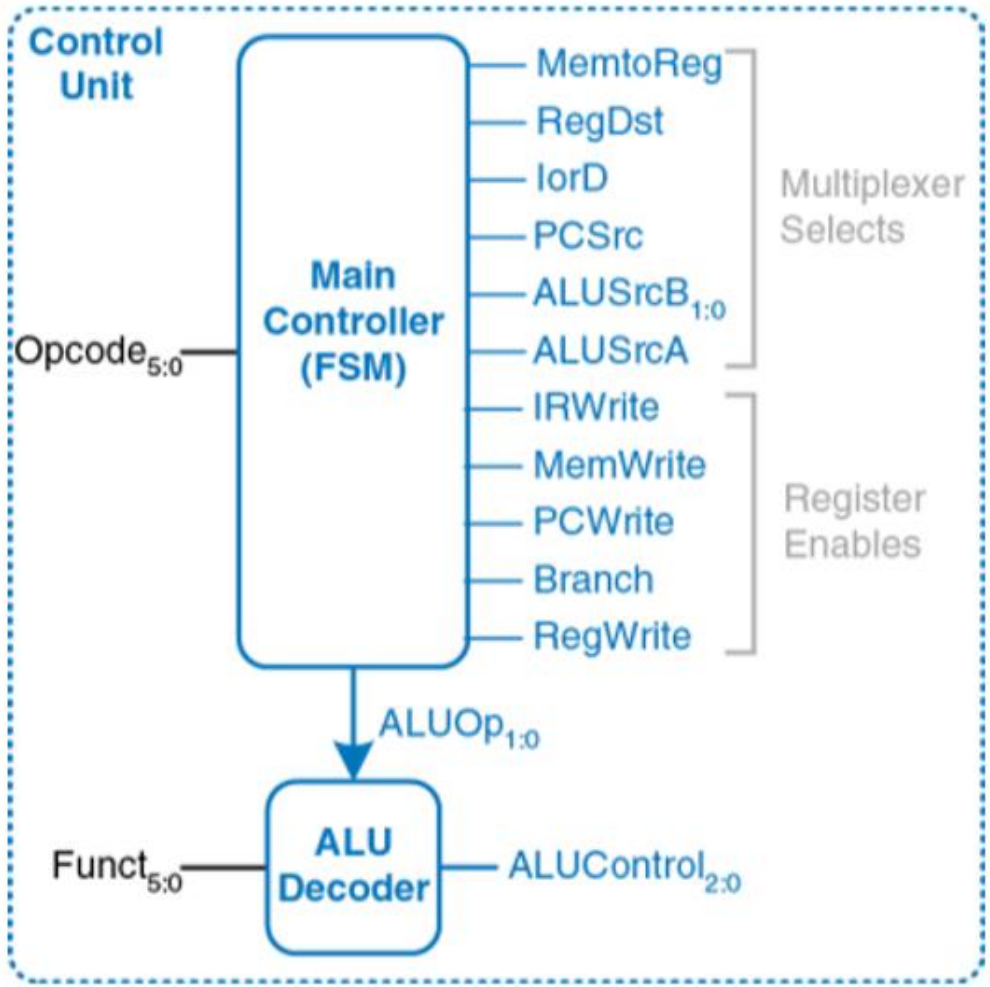


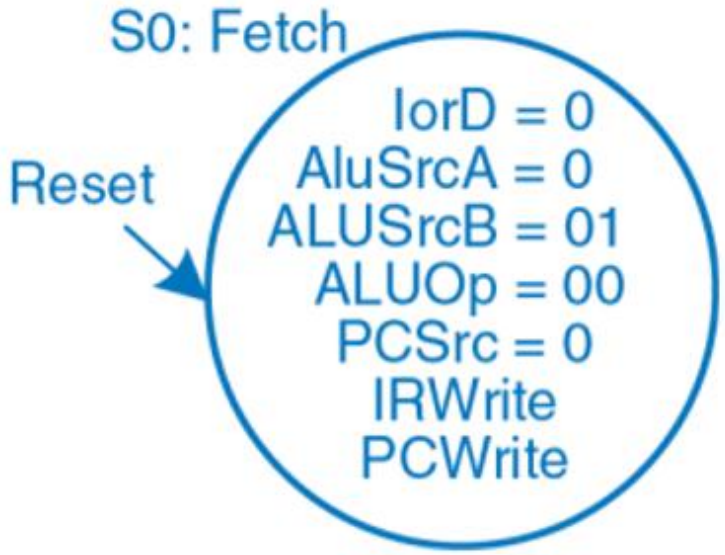
Поддержка команды **beq**




CLK
Control Unit
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
IorD
MemWrite
IRWrite
31:26
5:0
Op
Funct
PCEn
PC'
PC
Adr
Instr/Data Memory
WE
RD
A
WD
Instr
Data
25:21
20:16
15:11
0:15
Instruction Register
Memory Data Register
Register File
A1
A2
A3
WE3
WD3
RD1
RD2
A
B
SrcA
SrcB
ALU
Zero
ALUResult
ALUOut
4
00
01
10
11
<<2
SignImm
Sign Extend
EN

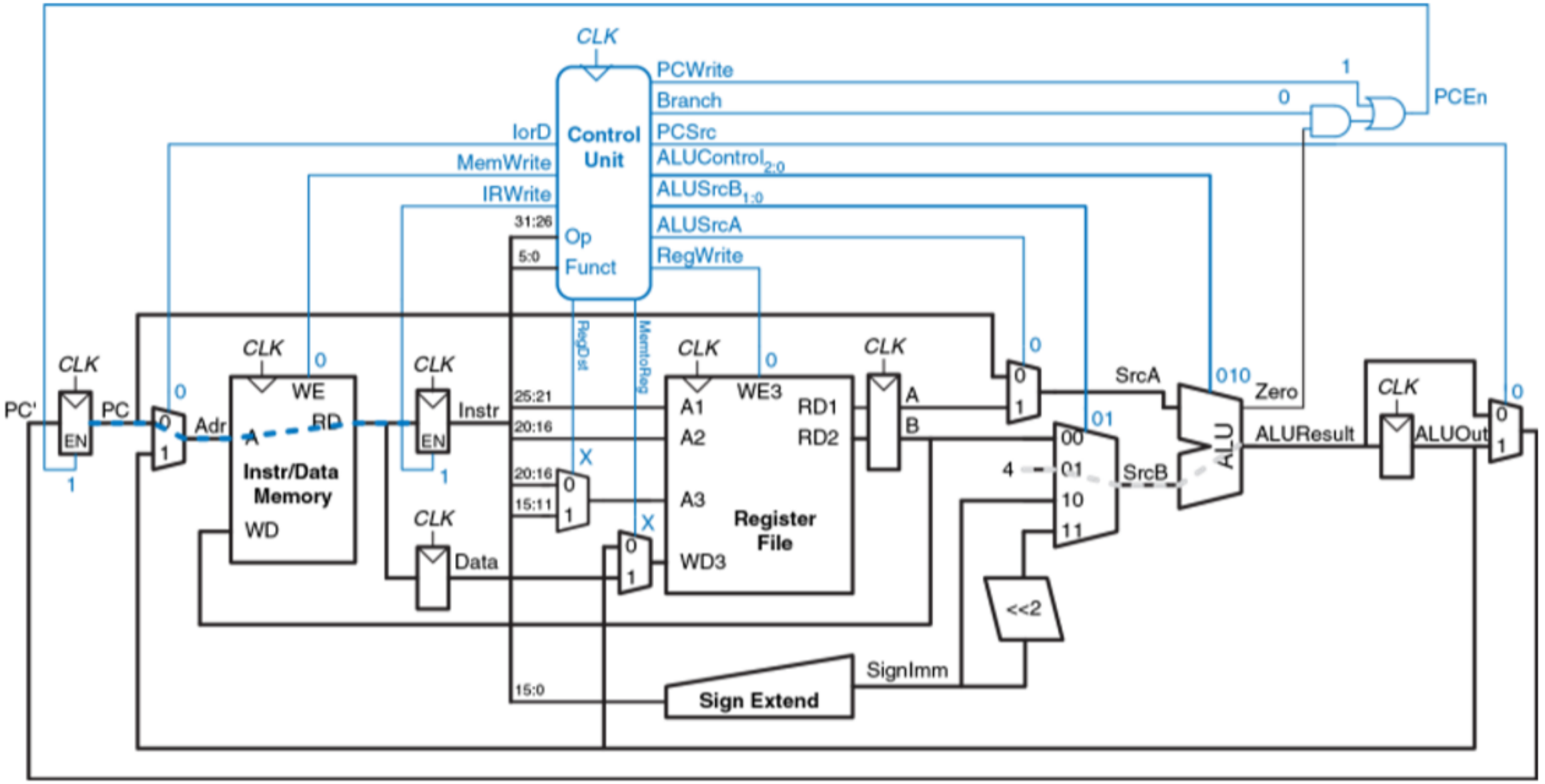
Многотактное устройство управления

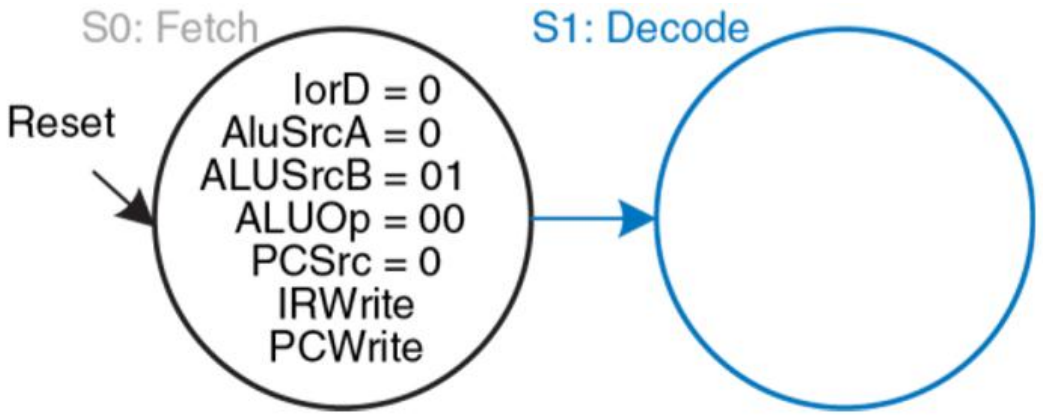




Этап выборки команды

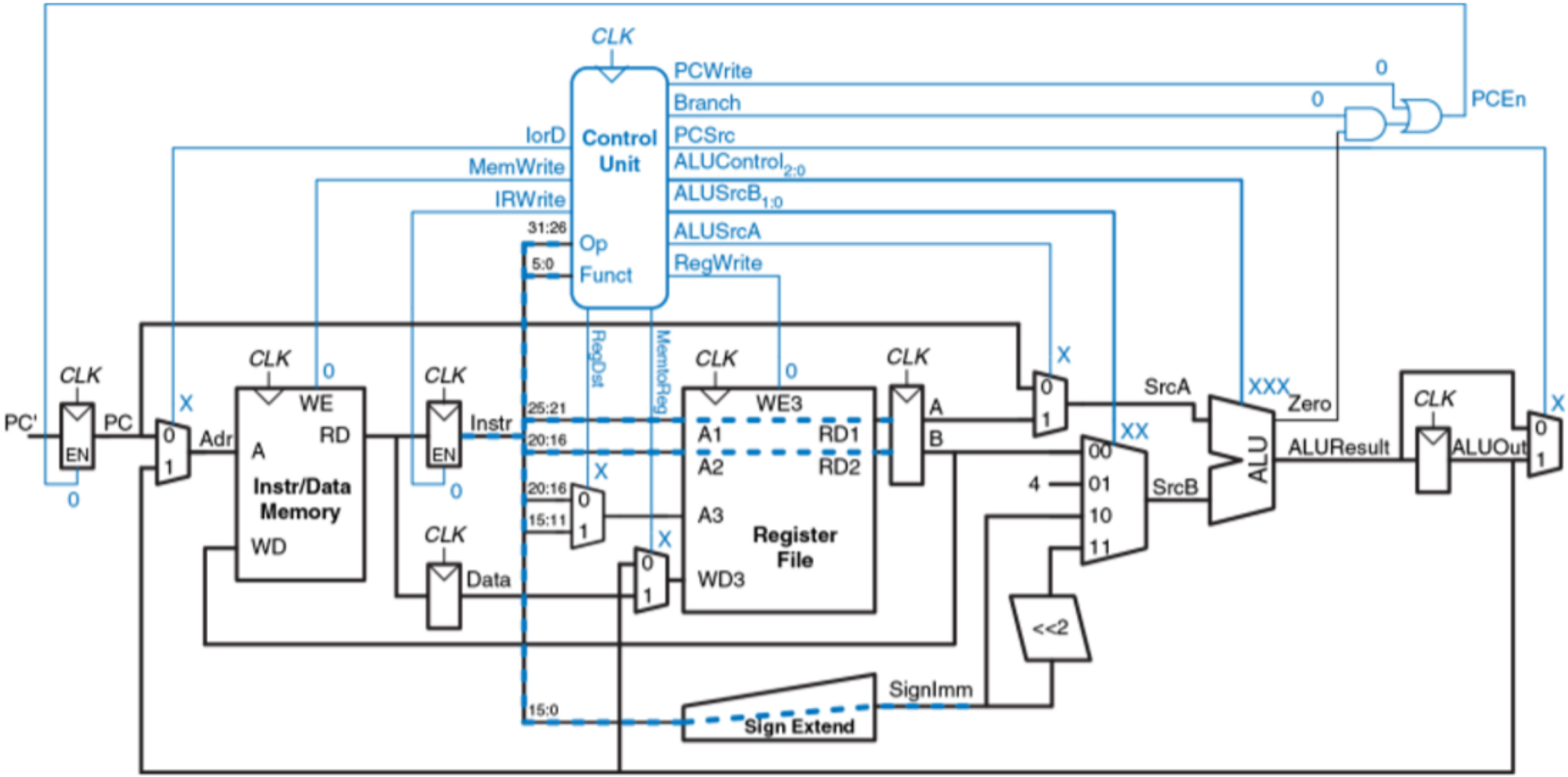
Этап выборки команды

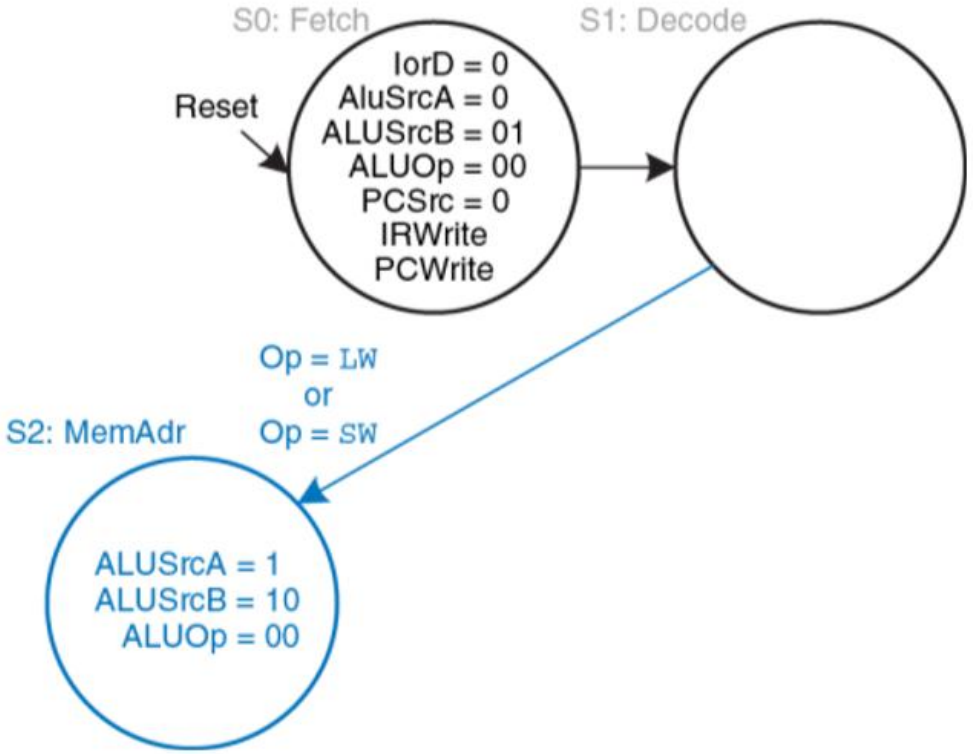




Этап дешифрации команды

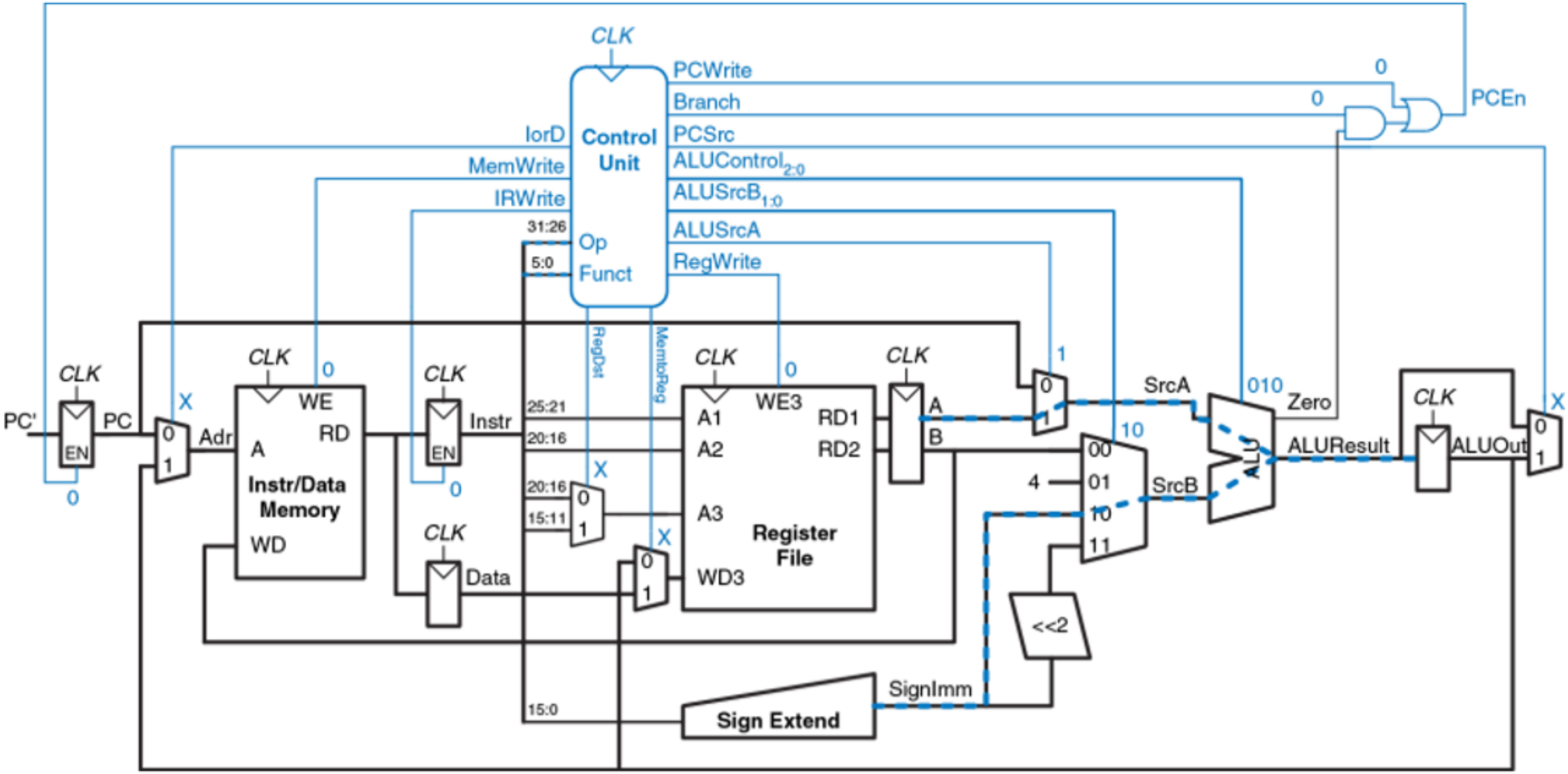
Этап дешифрации команды

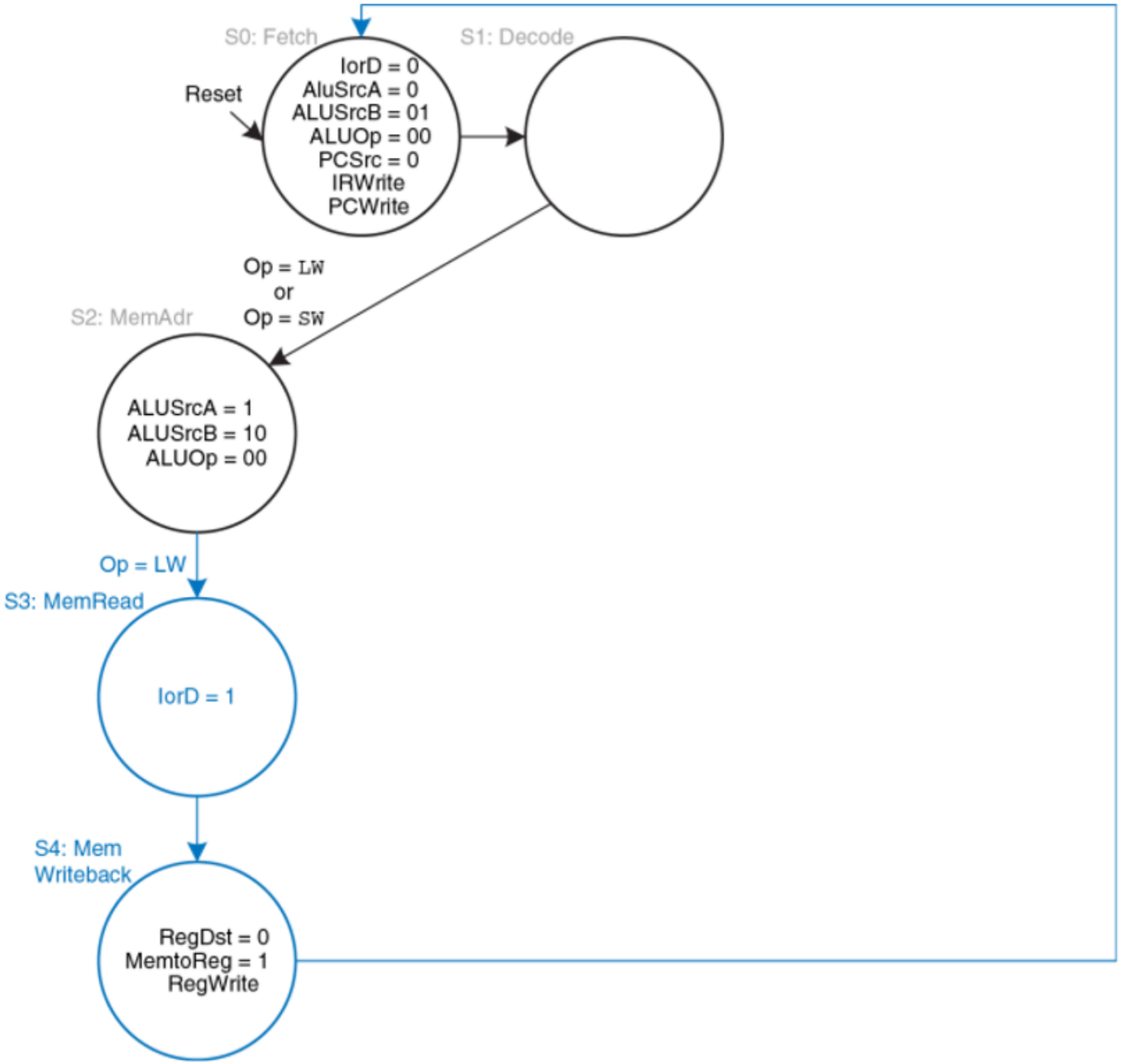




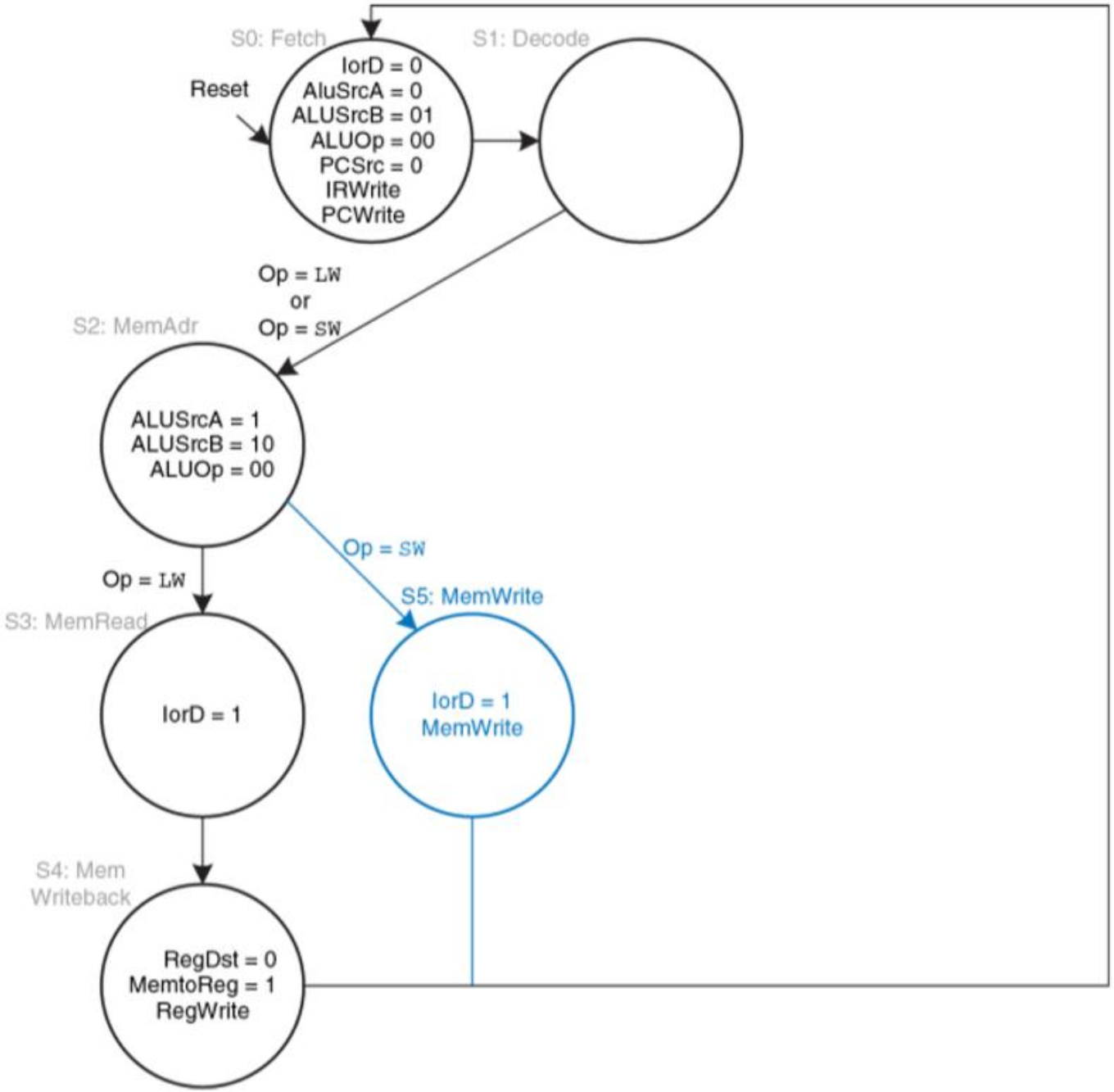
Вычисление адреса в памяти данных

Вычисление адреса в памяти данных

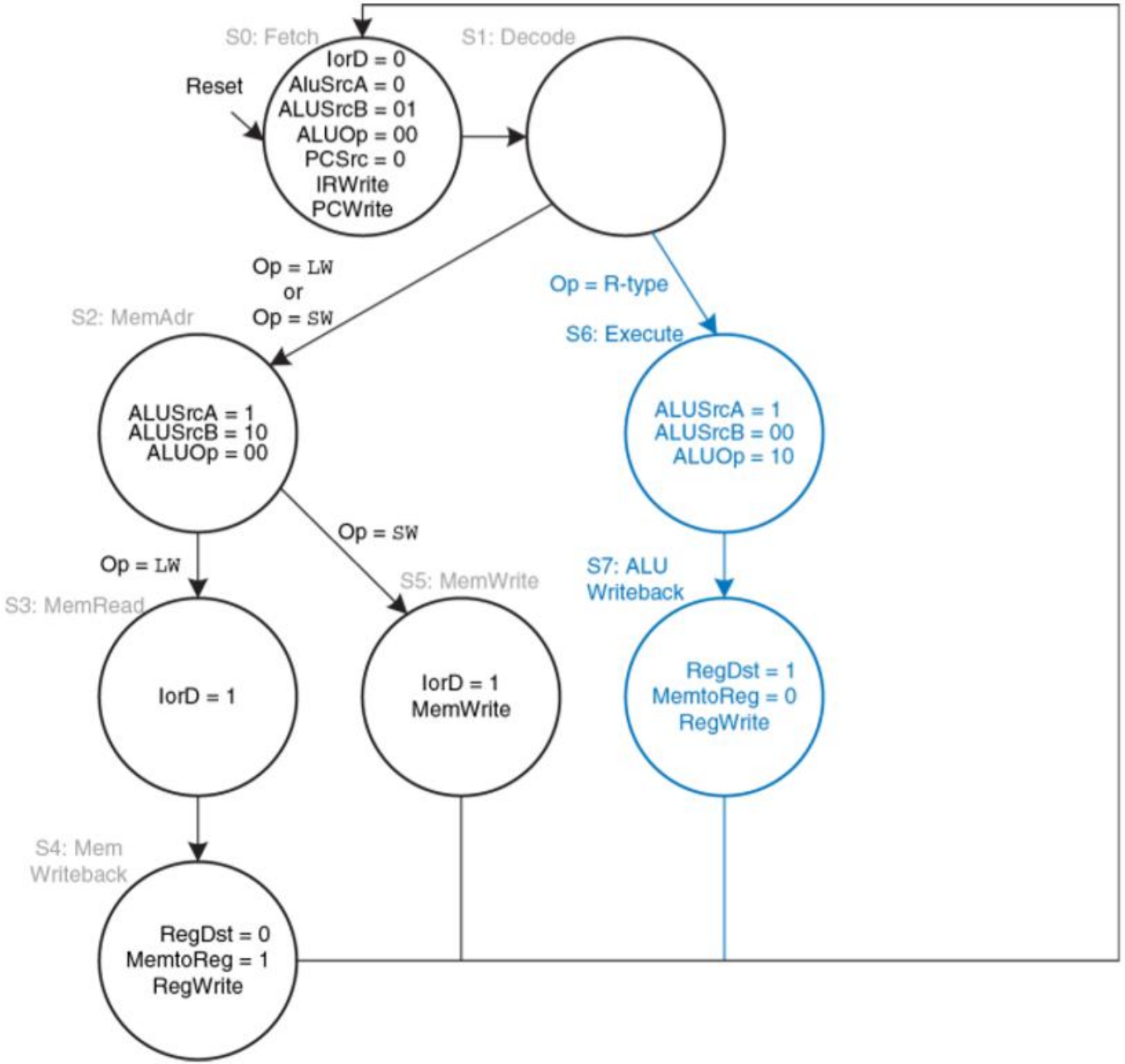




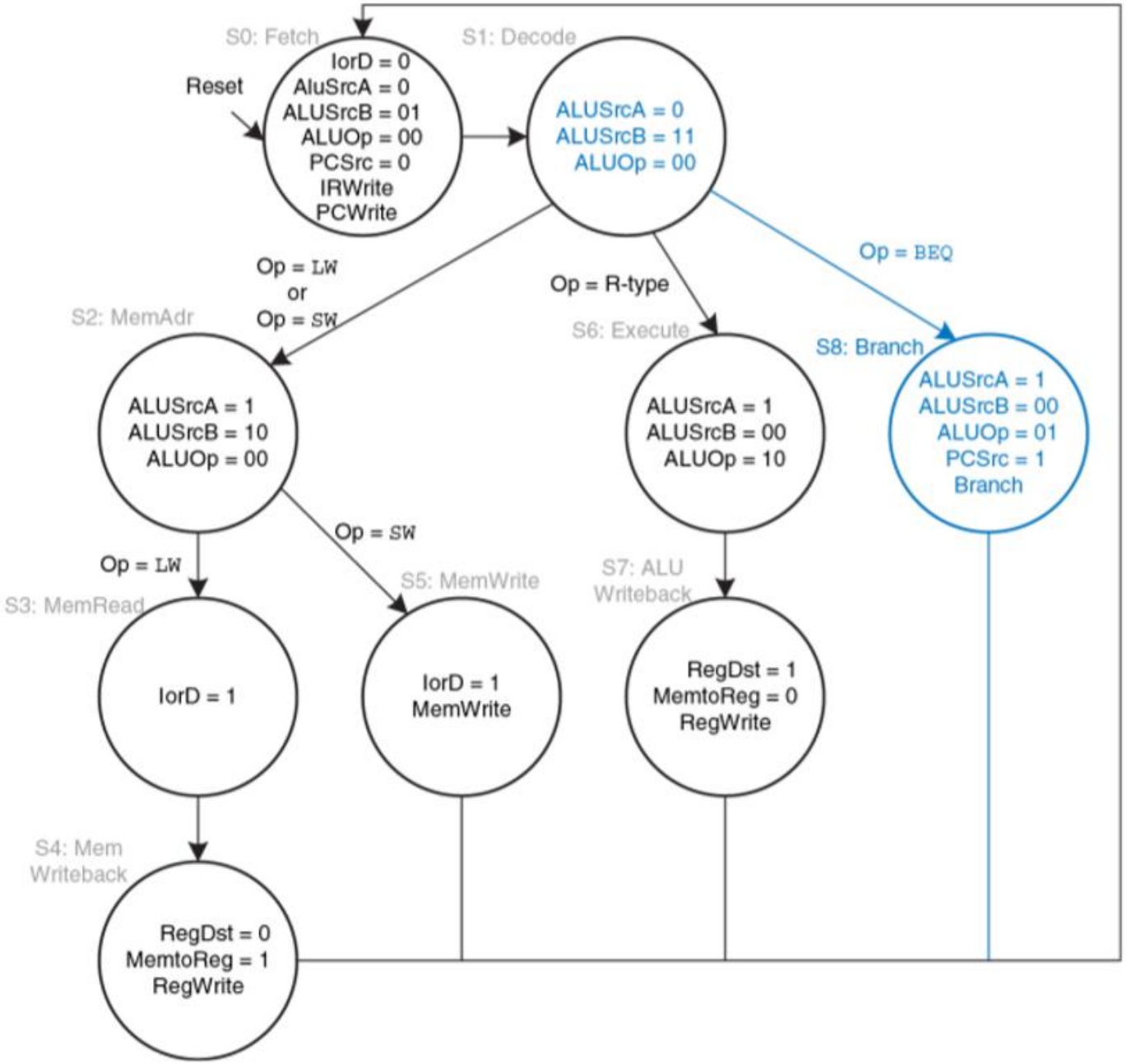
Чтение данных из памяти



Запись в память данных



Вычисление результатов команд типа **R**



Выполнение команды **beq**