Процессорное ядро schoolMIPS

Young Russian Chip Architects

Адрес для скачивания schoolMIPS :
https://github.com/MIPSfpga/schoolMIPS

Данная презентация и другая документация:
https://github.com/MIPSfpga/schoolMIPS/tree/master/doc

Благодарности

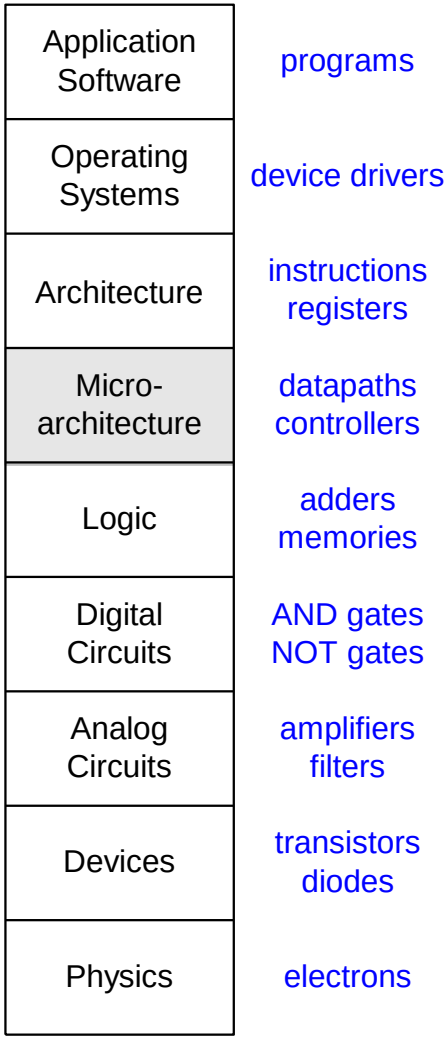
- Сара Л. Харрис, Дэвид М. Харрис – авторы прекрасного учебника «Цифровая схемотехника и архитектура компьютера», однотактный процессор из этой книги послужил основой для schoolMIPS
- Коллектив переводчиков учебника «Цифровая схемотехника и архитектура компьютера» на русский язык
- Участники конференции Young Russian Chip Architects
- Юрий Панчул - старший инженер по разработке и верификации блоков микропроцессорного ядра в команде MIPS I6400, Imagination Technologies, отделение в Санта-Кларе, Калифорния, США. Автор инициативы по преподаванию HDL и ПЛИС школьникам и идеи создания ядра schoolMIPS
- Станислав Жельнио – архитектура и программирование ядра, документация
- Александр Романов - НИУ ВШЭ, МИЭМ – архитектура ядра, тестирование и портирование на различные отладочные платы

Что такое schoolMIPS

- простое процессорное ядро для преподавания школьникам основ цифровой схемотехники, языков описания аппаратуры и использования ПЛИС
- написано на языке Verilog
- реализует подмножество архитектуры MIPS

Введение

- **Микроархитектура:** аппаратная реализация архитектуры в виде схемы
- Процессор:
  - **Тракт данных:** функциональные блоки обработки и передачи данных (арифметико-логическое устройство, регистровый файл, мультиплексоры и т.д.)
  - **Устройство управления:** формирует управляющие сигналы для функциональных блоков



Микроархитектура

- Возможны несколько аппаратных реализаций одной и той же архитектуры:
  - **Однотактная реализация:** каждая инструкция выполняется за один такт
  - **Многотактная реализация:** каждая инструкция разбивается на несколько шагов и выполняется за несколько тактов
  - **Конвейерная реализация:** каждая инструкция разбивается на несколько шагов и несколько инструкций выполняются одновременно

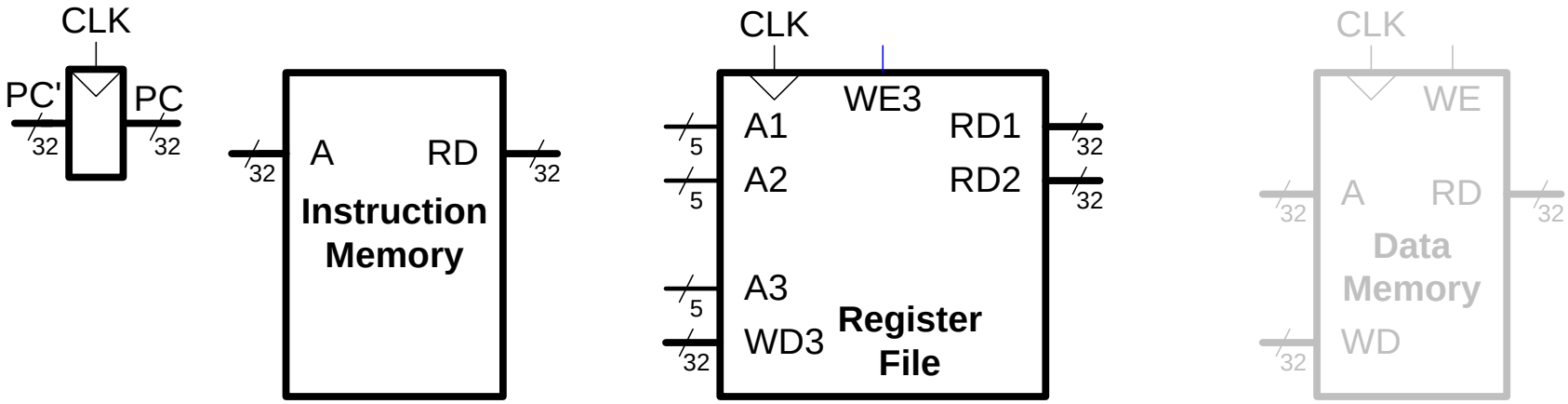
MIPS процессор schoolMIPS

- Однотактная реализация
- Отсутствует память данных
- Словная адресация памяти инструкций
- Инструкции:
  - R-типа (оба аргумента хранятся в регистрах): `addu, or, srl, sltu, subu`
  - I-типа (один из аргументов - константа): `addiu, lui`
  - I-типа (инструкции ветвления): `beq, bne`

Архитектурное состояние

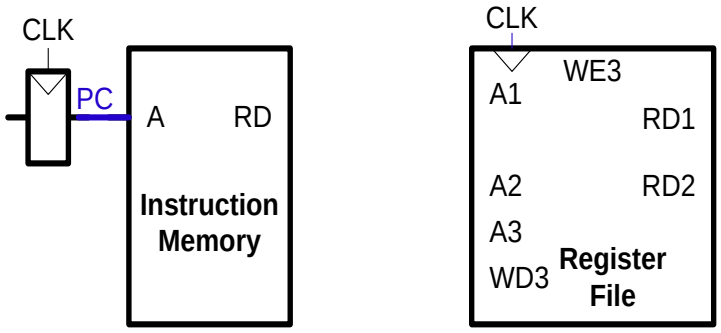
- Определяется:
  - Содержимым счетчика команд (PC)
  - Содержимым 32-х регистров общего назначения
  - Содержимым памяти (команд, данных)

Элементы, хранящие состояние MIPS



Процессор schoolMIPS: инструкция addiu

Шаг 1: Выборка (считывание) инструкции `addiu` из памяти

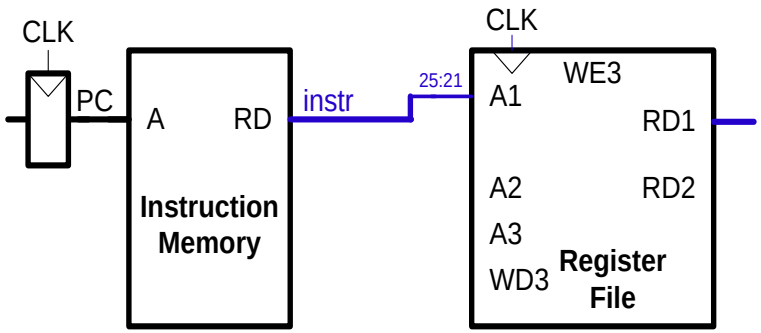


I-type. Integer Add Immediate, rt = rs + Immediate

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **Immediate** 0

Процессор schoolMIPS: инструкция addiu

Шаг 2: считывание операндов-источников из регистрового файла

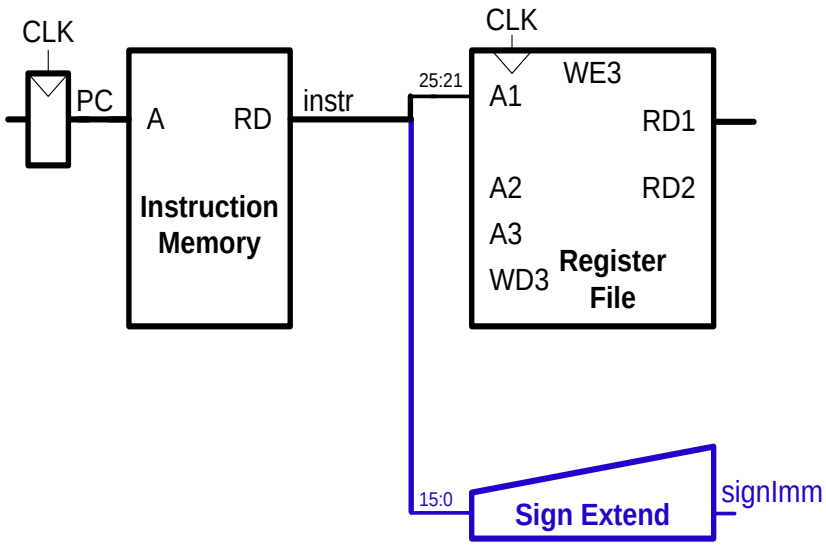


I-type. Integer Add Immediate, rt = rs + Immediate

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **Immediate** 0

Процессор schoolMIPS: инструкция addiu

Шаг 3: расширение 16-битной константы до 32-х разрядов битом знака

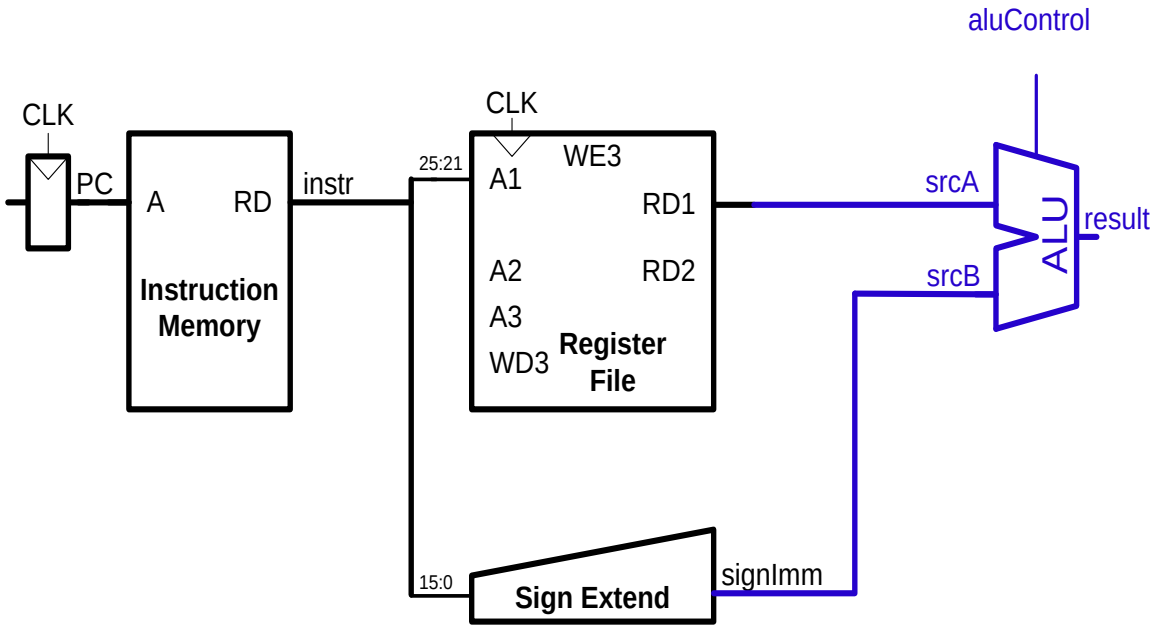


I-type. Integer Add Immediate, rt = rs + Immediate

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **Immediate** 0

Процессор schoolMIPS: инструкция addiu

Шаг 4: вычисление результата арифметической операции

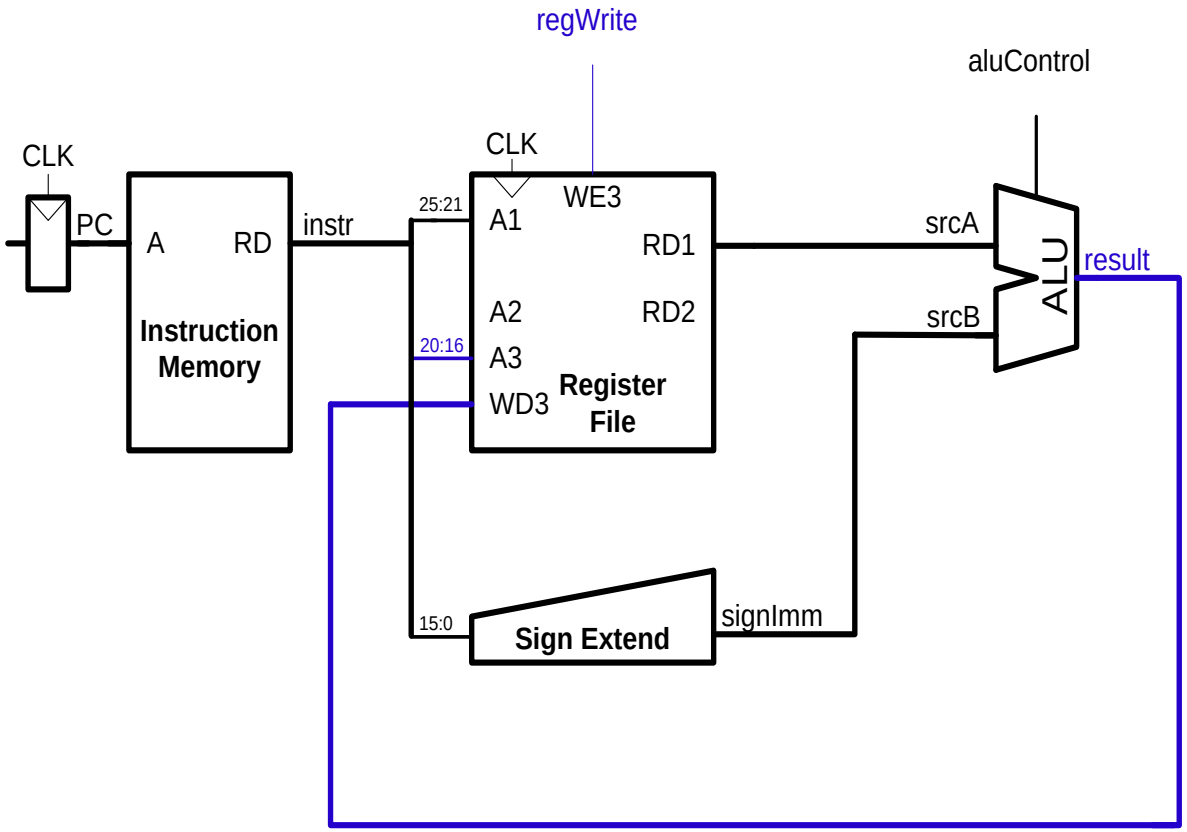


I-type. Integer Add Immediate, rt = rs + Immediate

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **Immediate** 0

Процессор schoolMIPS: инструкция addiu

Шаг 5: запись результата вычислений в регистр

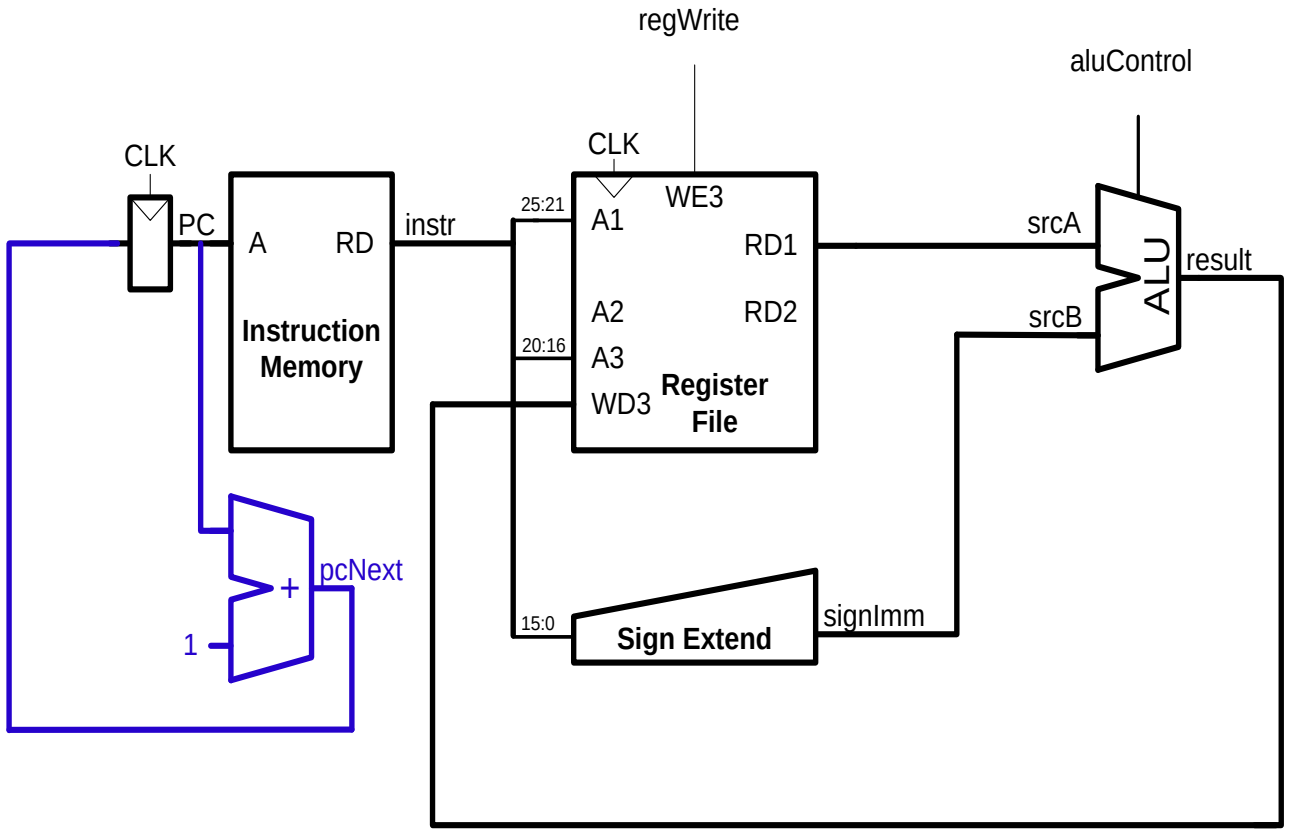


I-type. Integer Add Immediate, rt = rs + Immediate

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **Immediate** 0

Процессор schoolMIPS: инструкция addiu

Шаг 6: вычисление адреса следующей инструкции

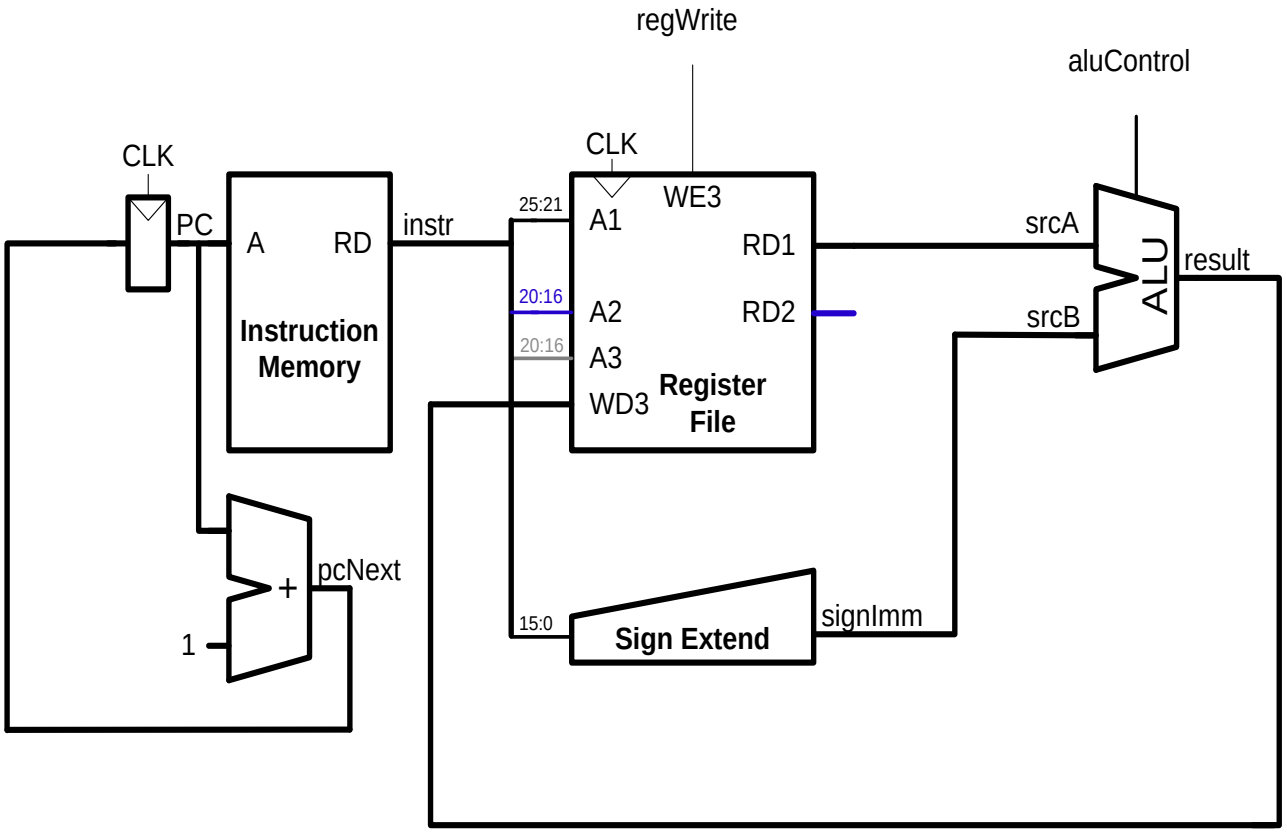


I-type. Integer Add Immediate, rt = rs + Immediate

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **Immediate** 0

Процессор schoolMIPS: инструкция addu

- считывание операнда 2 из регистрового файла

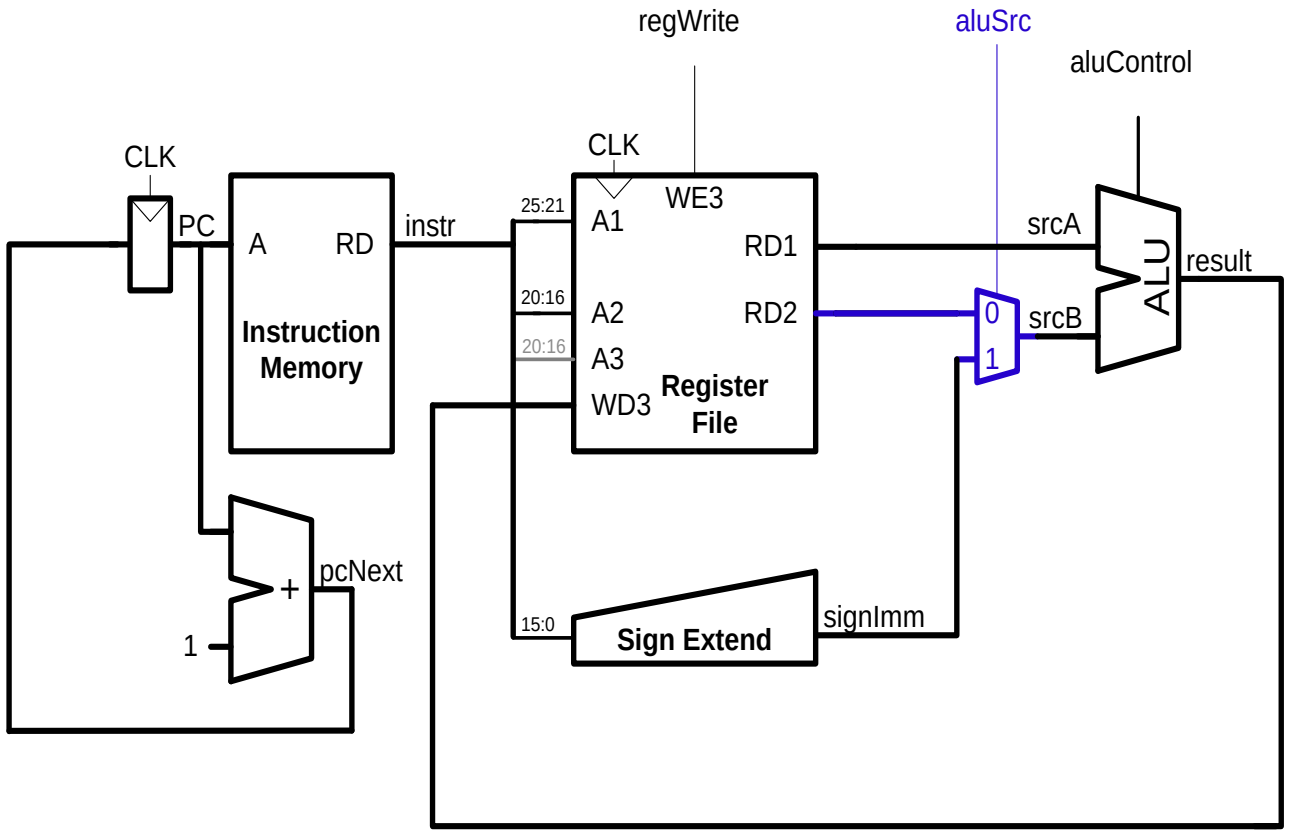


R-type. Integer Add Unsigned, **rd = rs + rt**

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **rd** 11	10 **sa** 6	5 **funct** 0

Процессор schoolMIPS: инструкция addu

- передача данных операнда 2 в арифметико-логическое устройство

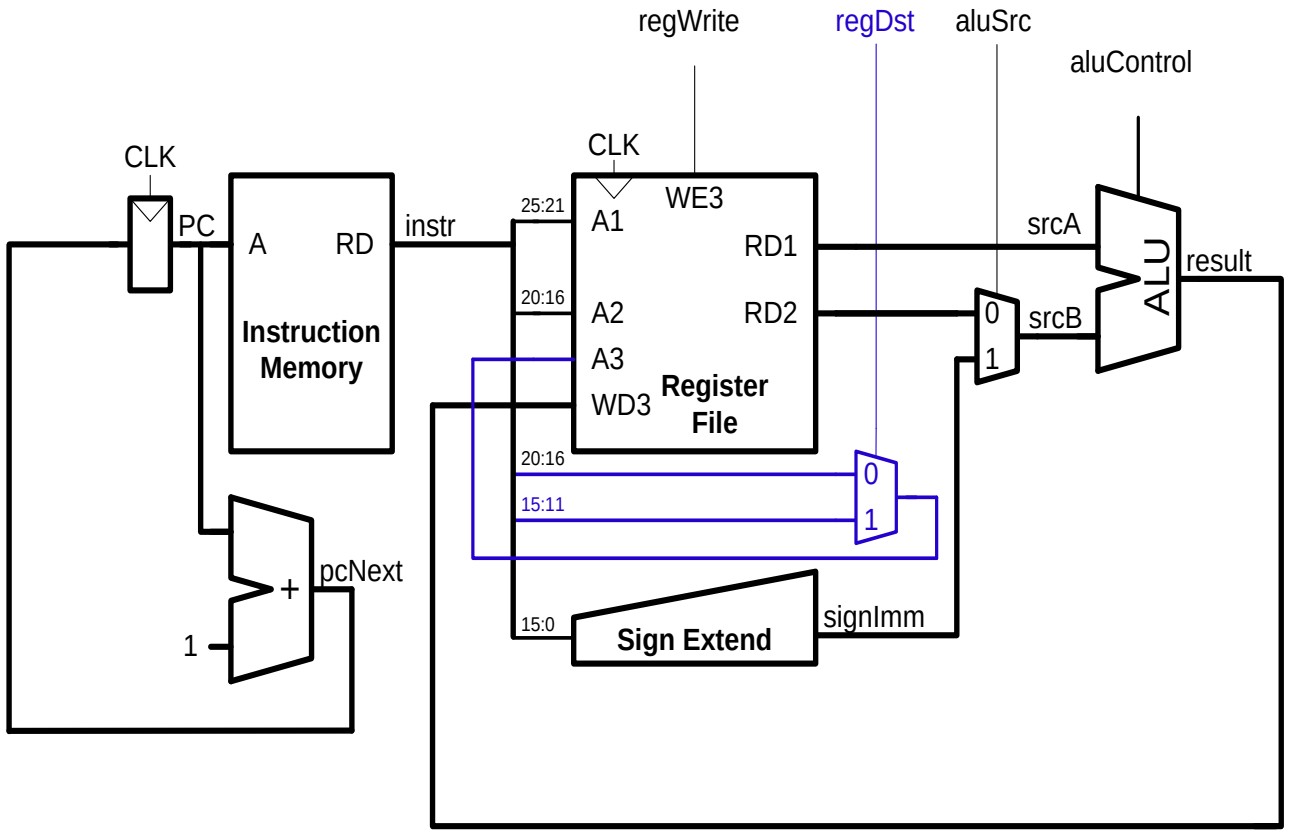


R-type. Integer Add Unsigned, **rd = rs + rt**

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **rd** 11	10 **sa** 6	5 **funct** 0

Процессор schoolMIPS: инструкция addu

- определение регистра для записи результата
- запись результата вычислений

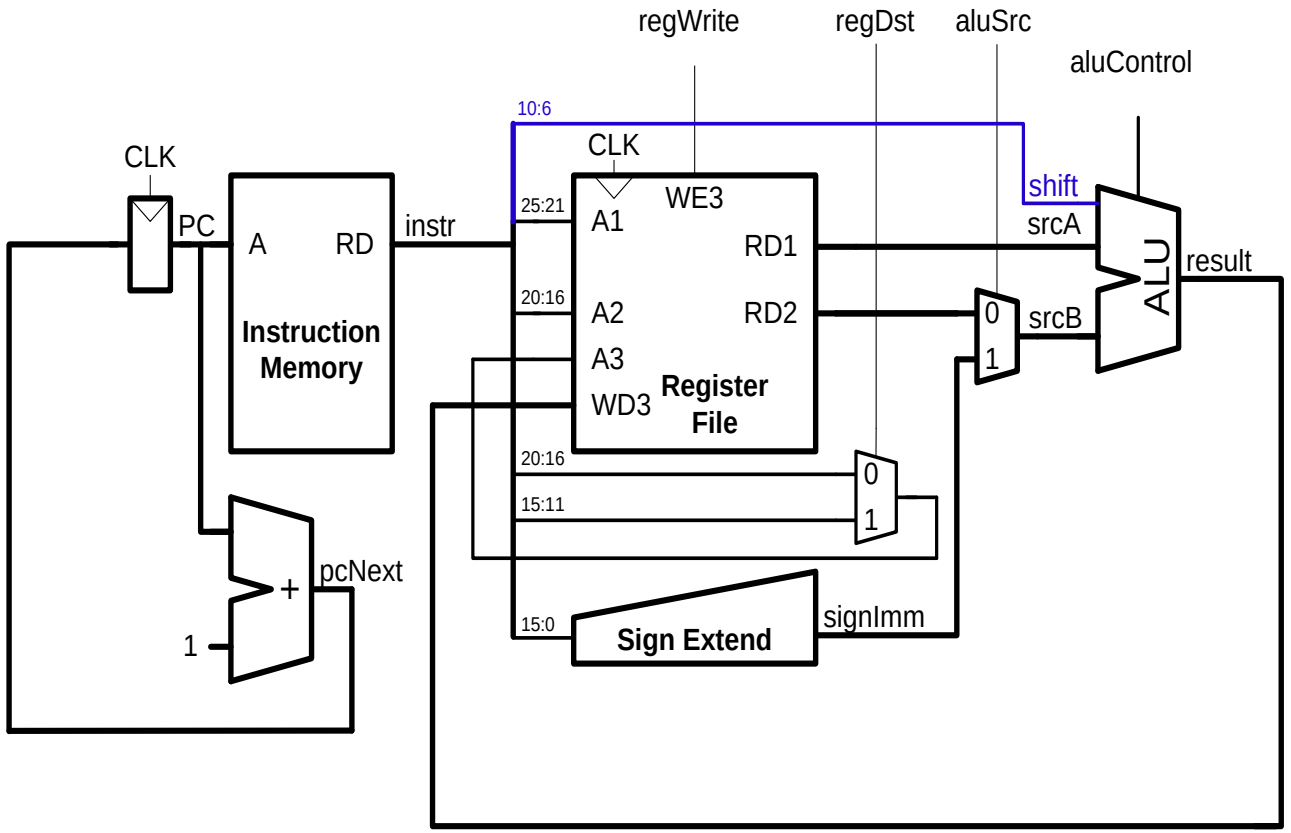


R-type. Integer Add Unsigned, **rd = rs + rt**

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **rd** 11	10 **sa** 6	5 **funct** 0

Процессор schoolMIPS: инструкция srl

- передача данных о размере сдвига в арифметико-логическое устройство

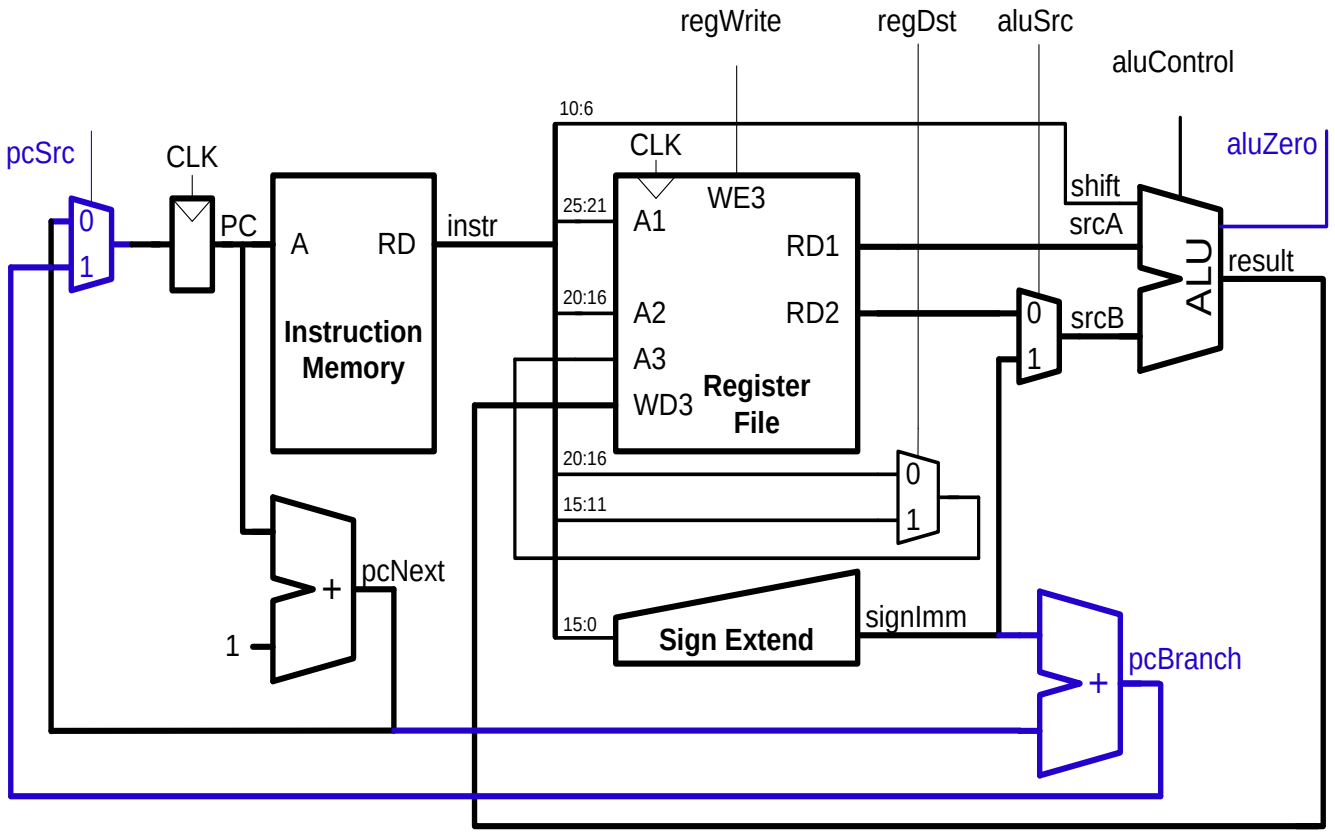


R-type. Shift Right Logical, rd = (uns)rt >> sa

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **rd** 11	10 **sa** 6	5 **funct** 0

Процессор schoolMIPS: инструкция beq

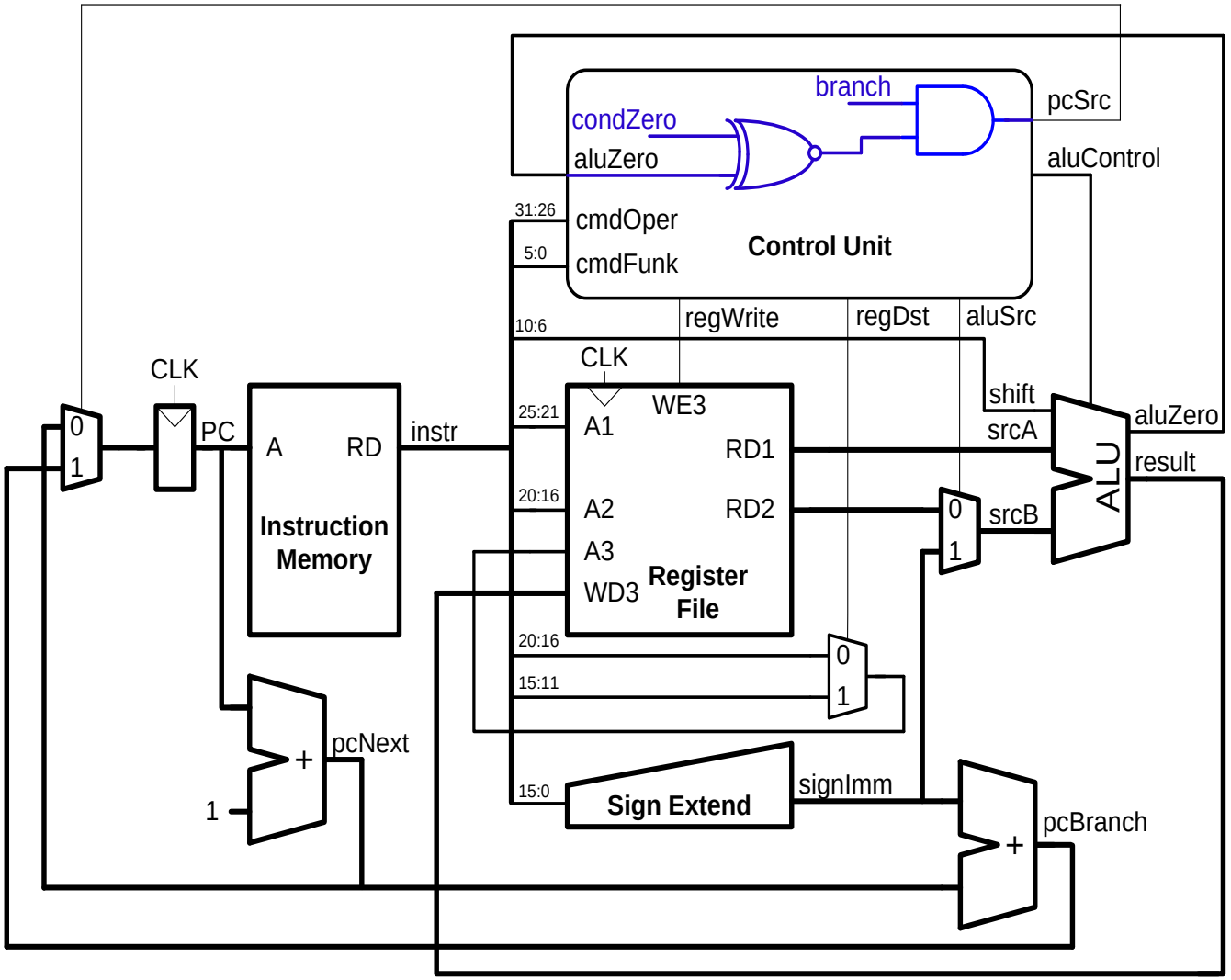
- вычисление адреса следующей инструкции



I-type. Branch On Equal, if (Rs == Rt) PC += (int)offset

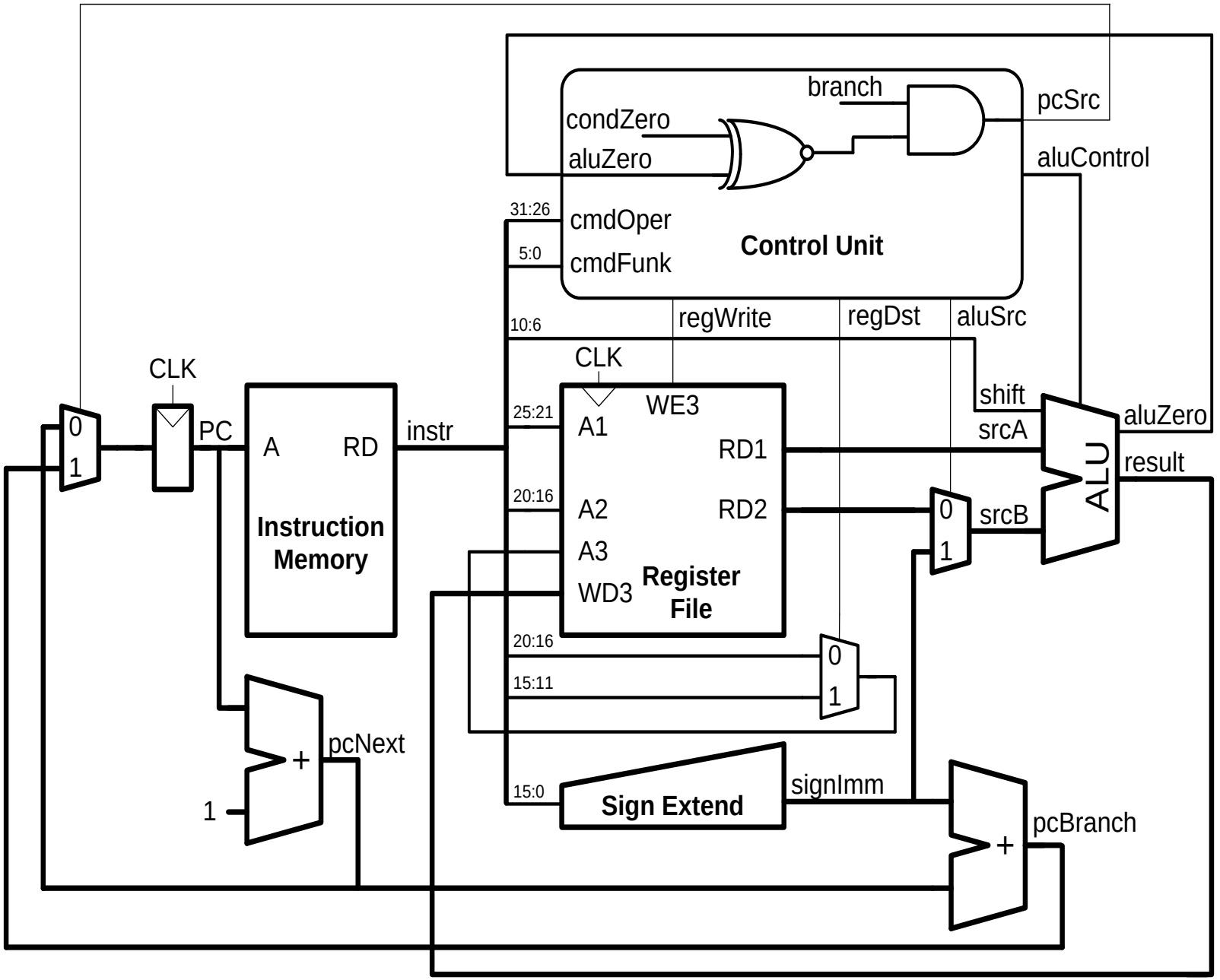
31 **op** 26	25 **rs** 21	20 **rt** 16	15 **Immediate** 0

Процессор schoolMIPS: инструкция beq



- определение необходимости перехода в зависимости от равенства результата нулю

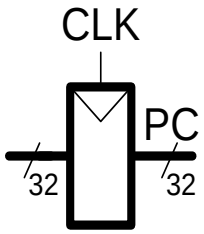
Процессор schoolMIPS. Итоговая схема



Процессор schoolMIPS. Итоговый состав

- Тракт данных
  - Счетчик команд (PC)
  - Память инструкций (Instruction Memory)
  - Регистровый файл (Register File)
  - Арифметико-логическое устройство (ALU)
  - Блок расширения знака (Sign Extend)
  - Сумматоры для вычисления адреса следующей инструкции (pcNext и pcBranch)
  - Мультиплексоры (pcSrc, regDst и aluSrc)
- Устройство управления

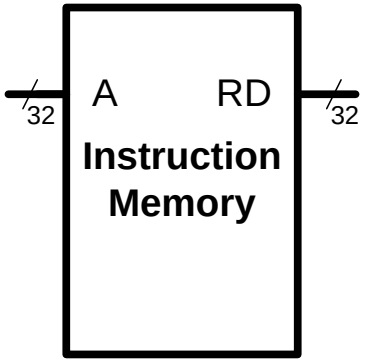
Реализация schoolMIPS. Счетчик команд




```
// sm_cpu.v (line 33)
sm_register r_pc(clk ,rst_n, pc_new, pc);
…
// sm_register.v (line 3-15)
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
```

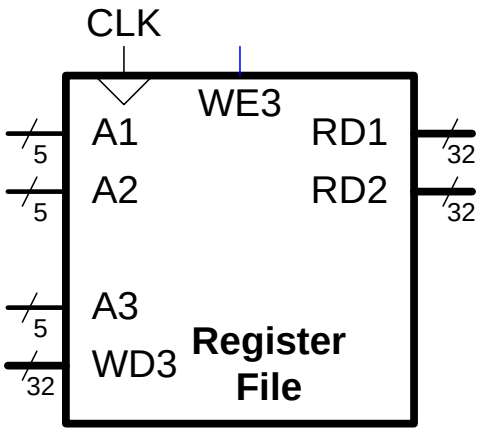
Реализация schoolMIPS. Память инструкций

```
// sm_cpu.v (line 35-37)
sm_rom reset_rom(pc, instr);
…
// sm_rom.v (line 2-17)
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
```

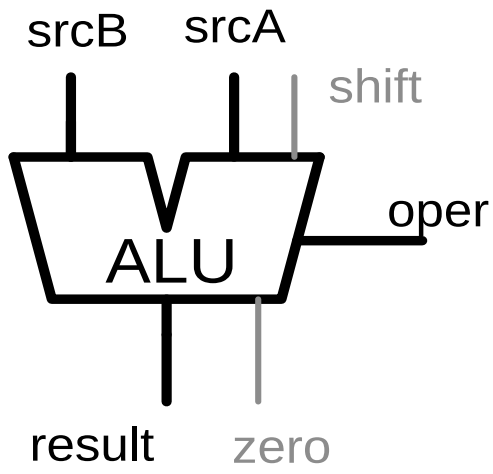


Реализация schoolMIPS. Регистровый файл

```
// sm_cpu.v (line 161-182)
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; //for debug
    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 schoolMIPS. Операции ALU

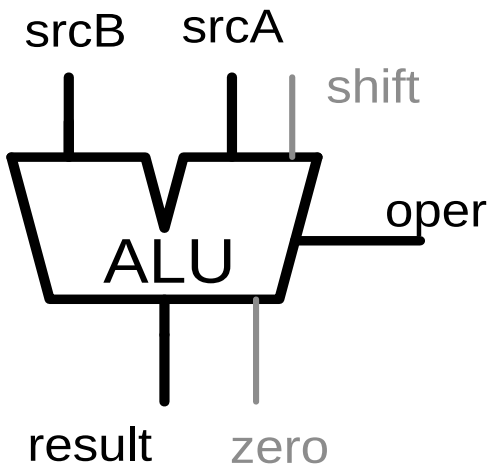


$oper_{2:0}$	Функция	Описание
000	ADD	A + B
001	OR	A | B
010	LUI	B << 16
011	SRL	B >> shift
100	SLTU	(A < B) ? 1 : 0
101	SUBU	A - B
110	Не исп.	
111	Не исп.	

```
// sm_cpu.vh (line 11-17)
`define ALU_ADD  3'b000
`define ALU_OR   3'b001
`define ALU_LUI  3'b010
`define ALU_SRL  3'b011
`define ALU_SLTU 3'b100
`define ALU_SUBU 3'b101
```

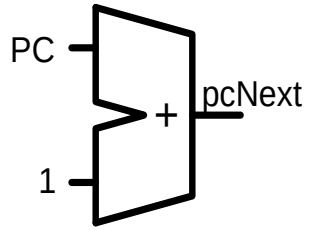
Реализация schoolMIPS. ALU

```
// sm_cpu.v (line 137-159)
module sm_alu (
    input       [31:0] srcA,
    input       [31:0] srcB,
    input       [ 2:0] oper,
    input       [ 4:0] shift,
    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_LUI  : result = (srcB << 16);
            `ALU_SRL  : result = srcB >> shift;
            `ALU_SLTU : result = (srcA < srcB) ? 1 : 0;
            `ALU_SUBU : result = srcA - srcB;
        endcase
    end
    assign zero = (result == 0);
endmodule
```



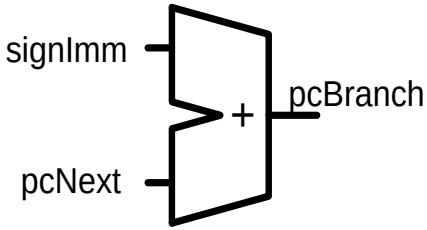
Реализация schoolMIPS. Сумматоры и блок расширения знака

```
//program counter   sm_cpu.v (line 28-31)
wire [31:0] pc;
wire [31:0] pcBranch;
wire [31:0] pcNext = pc + 1;
```



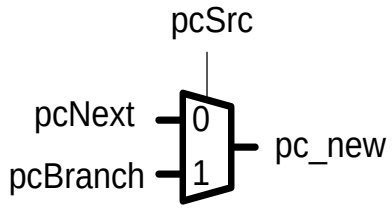

```
//sign extension   sm_cpu.v (line 64)
wire [31:0] signImm
    = { {16 { instr[15] }}, instr[15:0] };
```


```
//branch address calculation    sm_cpu.v (line 65)
assign pcBranch = pcNext + signImm;
```

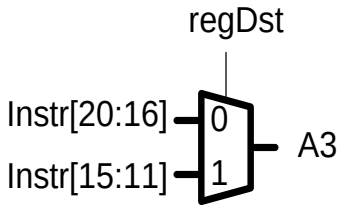


Реализация schoolMIPS. Мультиплексоры

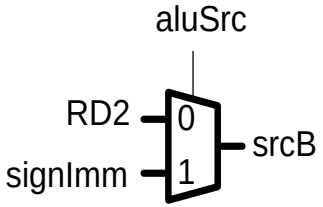
```
// next PC mux: branch or +1 (line 32)
wire [31:0] pc_new = ~pcSrc ? pcNext : pcBranch;
```




```
// register file address A3 (line 44)
wire [ 4:0] a3
     = regDst ? instr[15:11] : instr[20:16];
```




```
// alu source B (line 68)
wire [31:0] srcB = aluSrc ? signImm : rd2;
```



Реализация schoolMIPS. Инструкции I-типа

```
//instruction operation code    sm_cpu.vh (line 21-27)
`define C_ADDIU 6'b001001 // I-type, Integer Add Immediate Unsigned
                          // Rd = Rs + Immed
`define C_BEQ 6'b000100   // I-type, Branch On Equal
                          // if (Rs == Rt) PC += (int)offset
`define C_LUI 6'b001111   // I-type, Load Upper Immediate
                          // Rt = Immed << 16
`define C_BNE 6'b000101   // I-type, Branch on Not Equal
                          // if (Rs != Rt) PC += (int)offset
```

I-type. Integer Add Immediate, rt = rs + Immediate

Реализация schoolMIPS. Инструкции R-типа

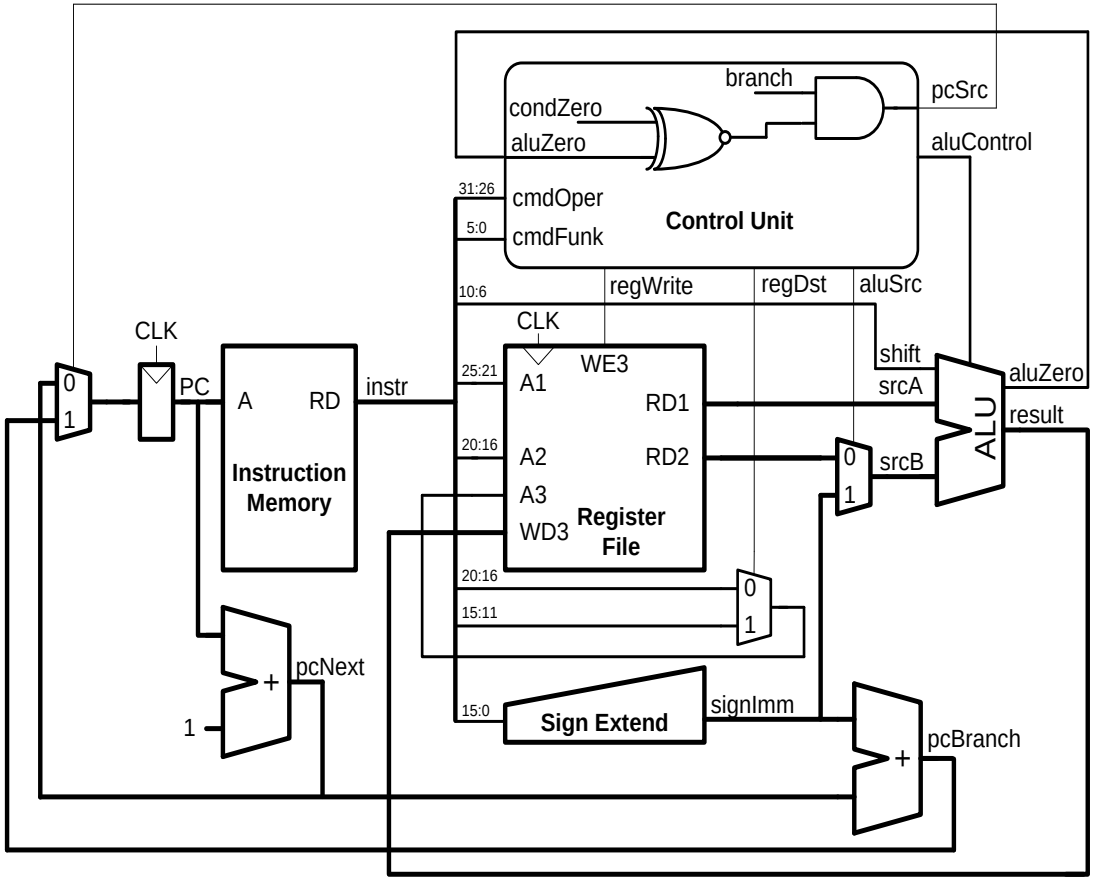
```
//instruction operation code    sm_cpu.vh (line 19-41)
`define C_SPEC 6'b000000 // Special instructions
                         // (depends on function field)

//instruction function field
`define F_ADDU 6'b100001 // R-type, Integer Add Unsigned
                         // Rd = Rs + Rt
`define F_OR   6'b100101 // R-type, Logical OR
                         // Rd = Rs | Rt
`define F_SRL  6'b000010 // R-type, Shift Right Logical
                         // Rd = RsØ >> shift
`define F_SLTU 6'b101011 // R-type, Set on Less Than Unsigned
                         // Rd = (RsØ < RtØ) ? 1 : 0
`define F_SUBU 6'b100011 // R-type, Unsigned Subtract
                         // Rd = Rs – Rt
`define F_ANY  6'b??????
```

R-type. Integer Add Unsigned, **rd = rs + rt**

31 **op** 26	25 **rs** 21	20 **rt** 16	15 **rd** 11	10 **sa** 6	5 **funct** 0

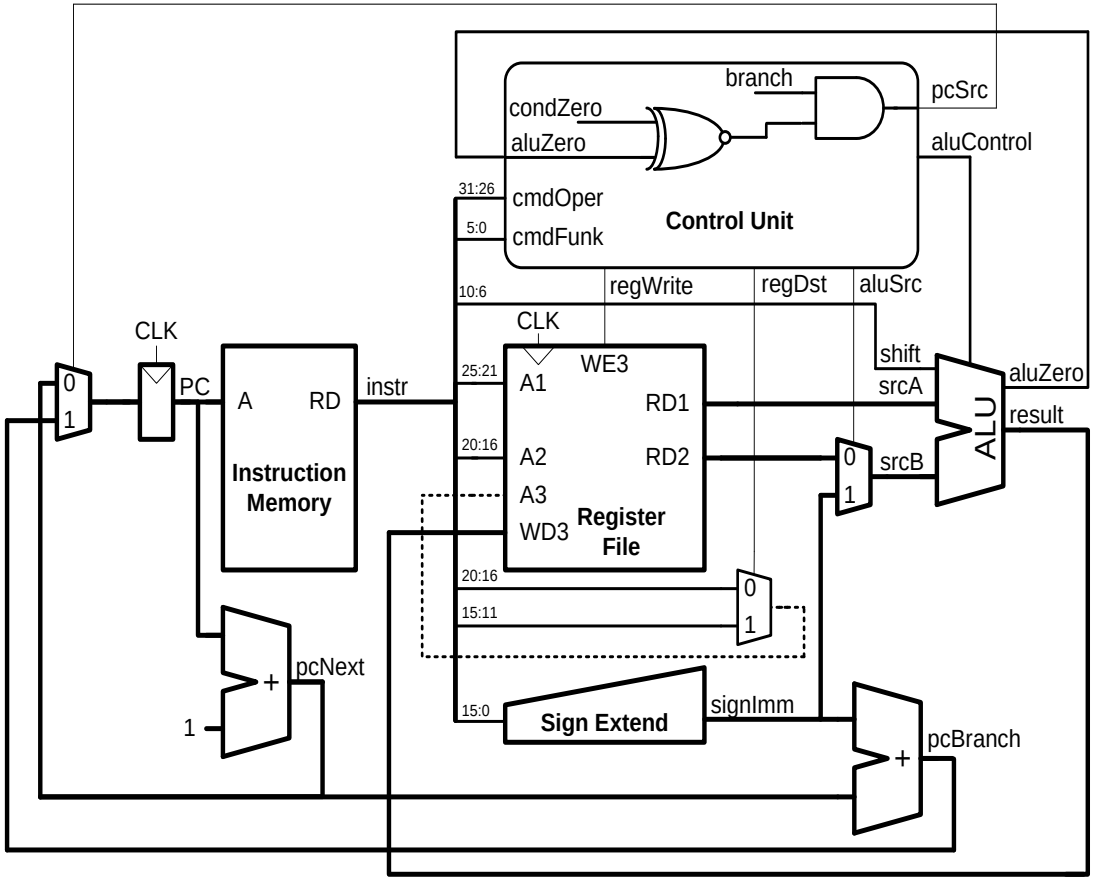
Реализация schoolMIPS: сигналы управления (1)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl



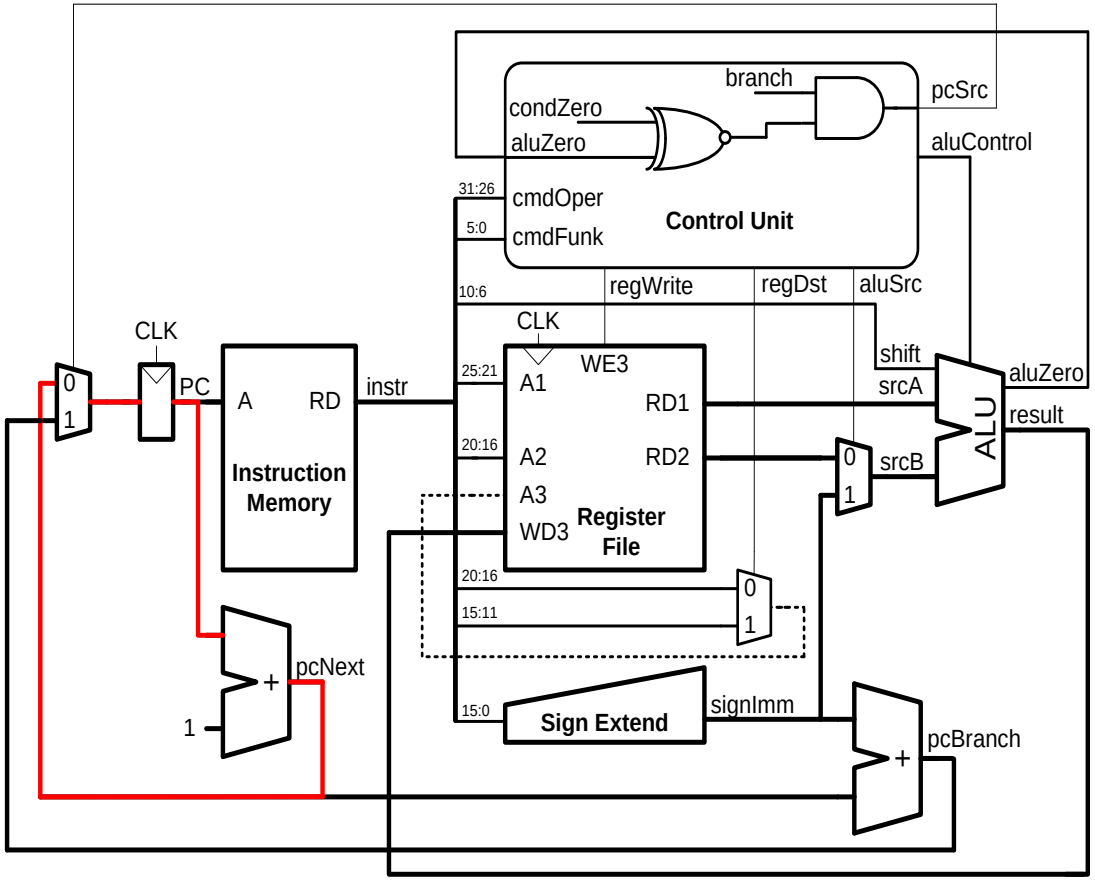
Процессор schoolMIPS: сигналы управления (2)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????						000



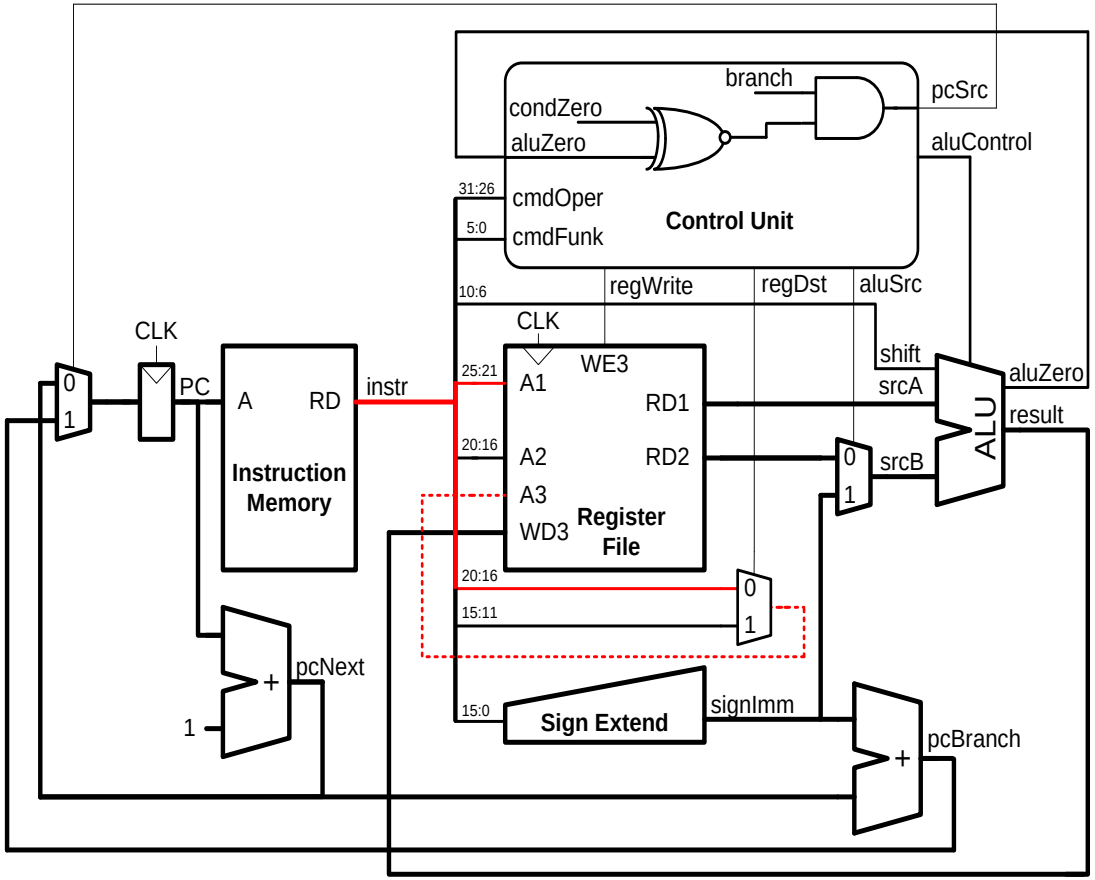
Процессор schoolMIPS: сигналы управления (3)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0				000



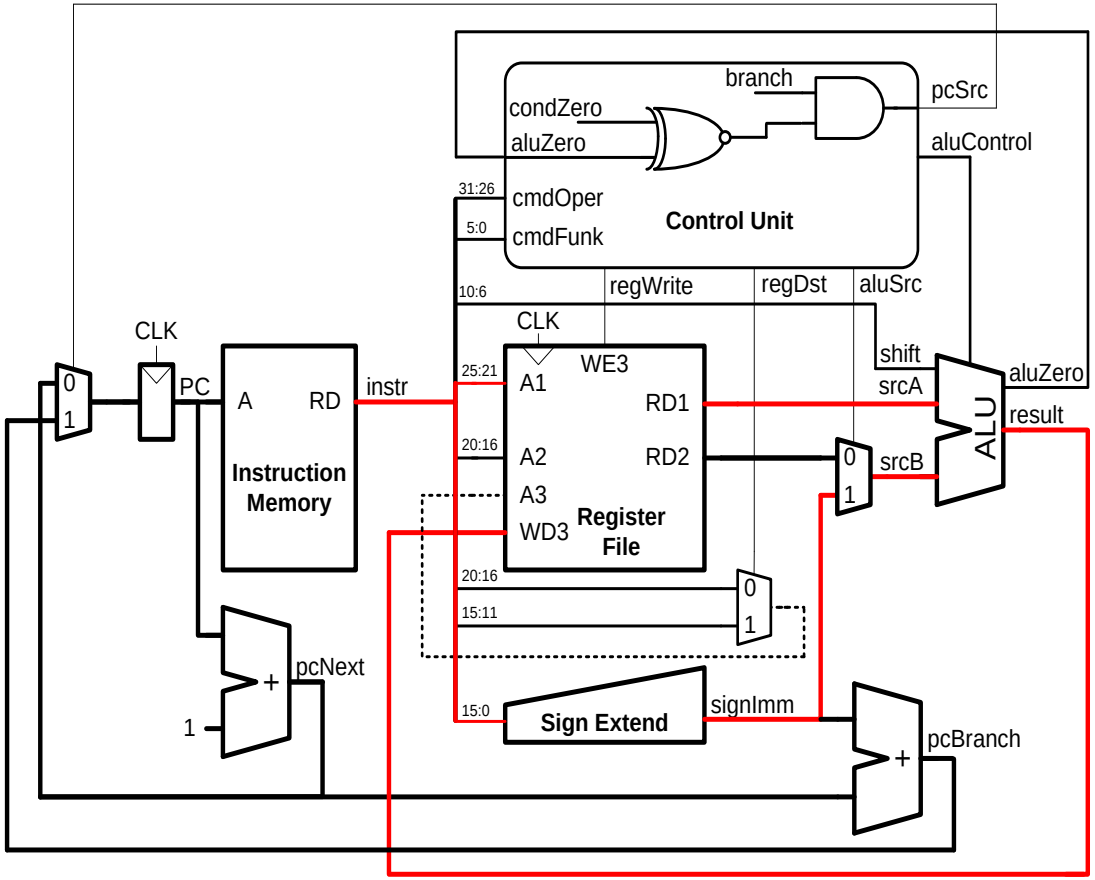
Процессор schoolMIPS: сигналы управления (4)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0			000



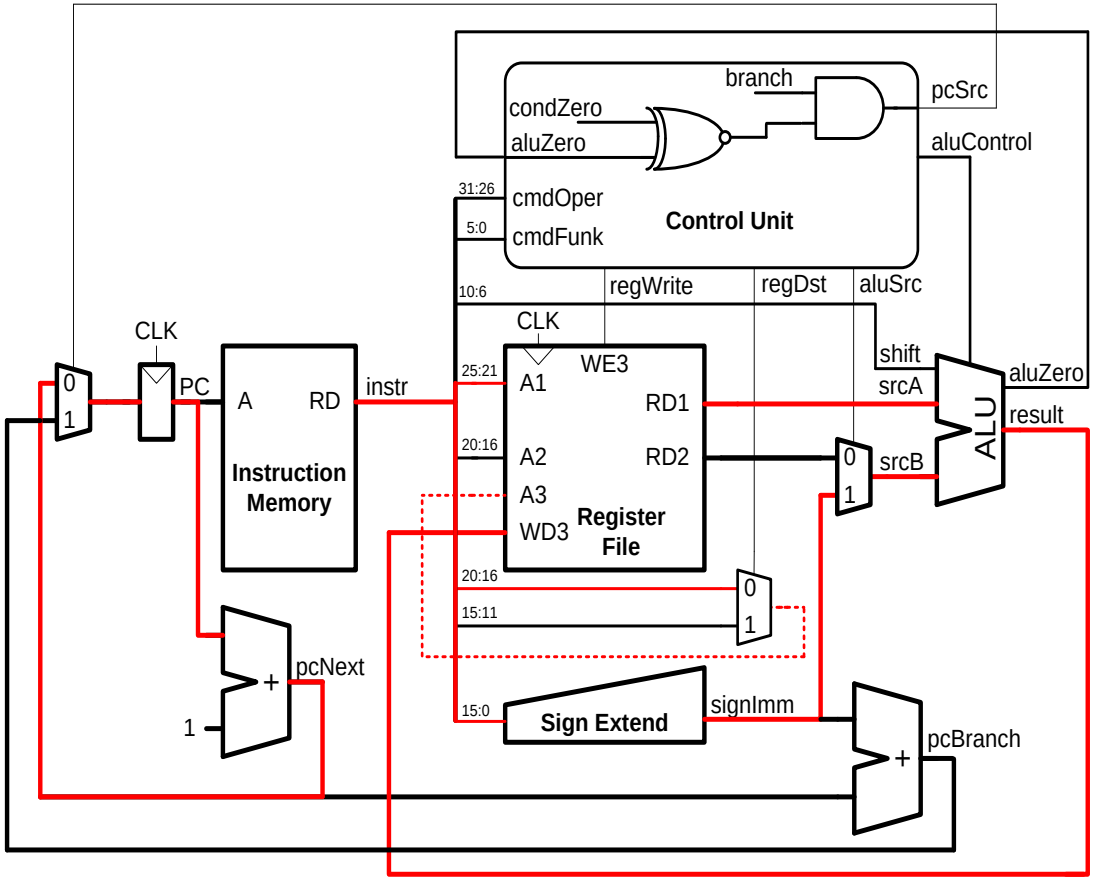
Процессор schoolMIPS: сигналы управления (5)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000



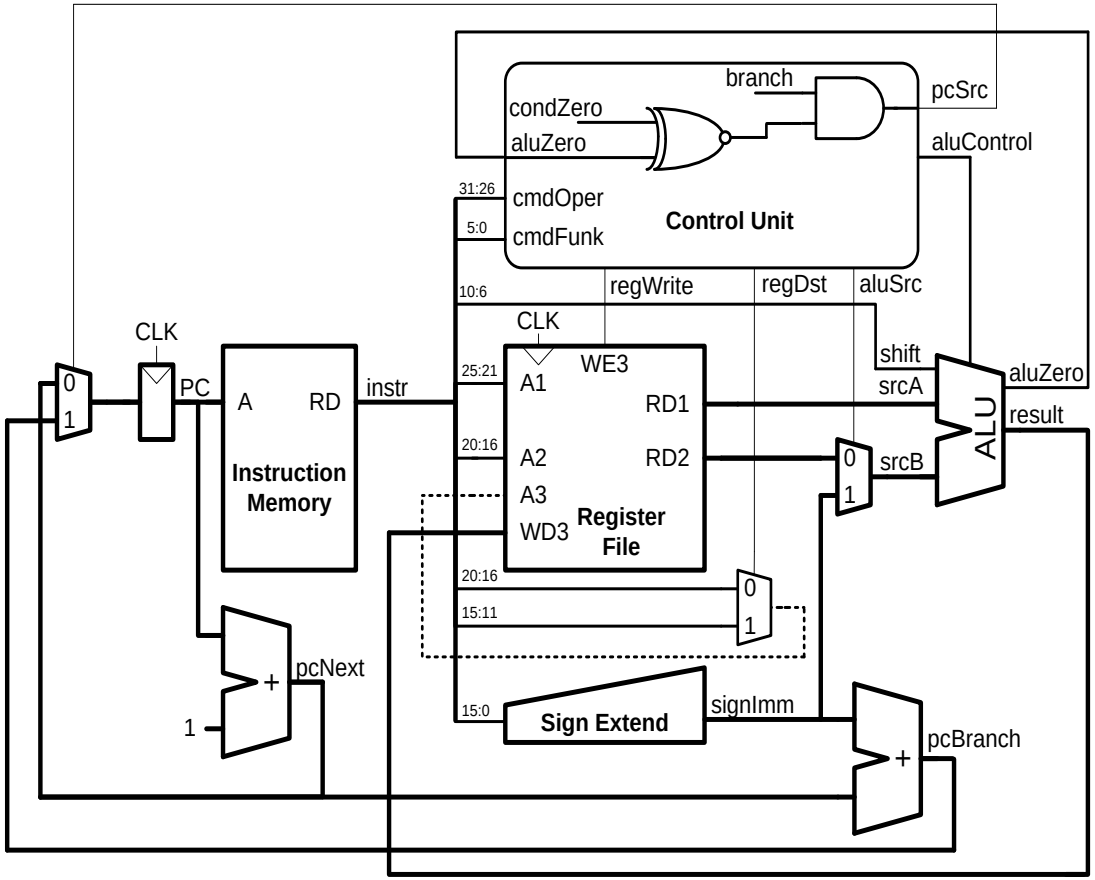
Процессор schoolMIPS: сигналы управления (6)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000



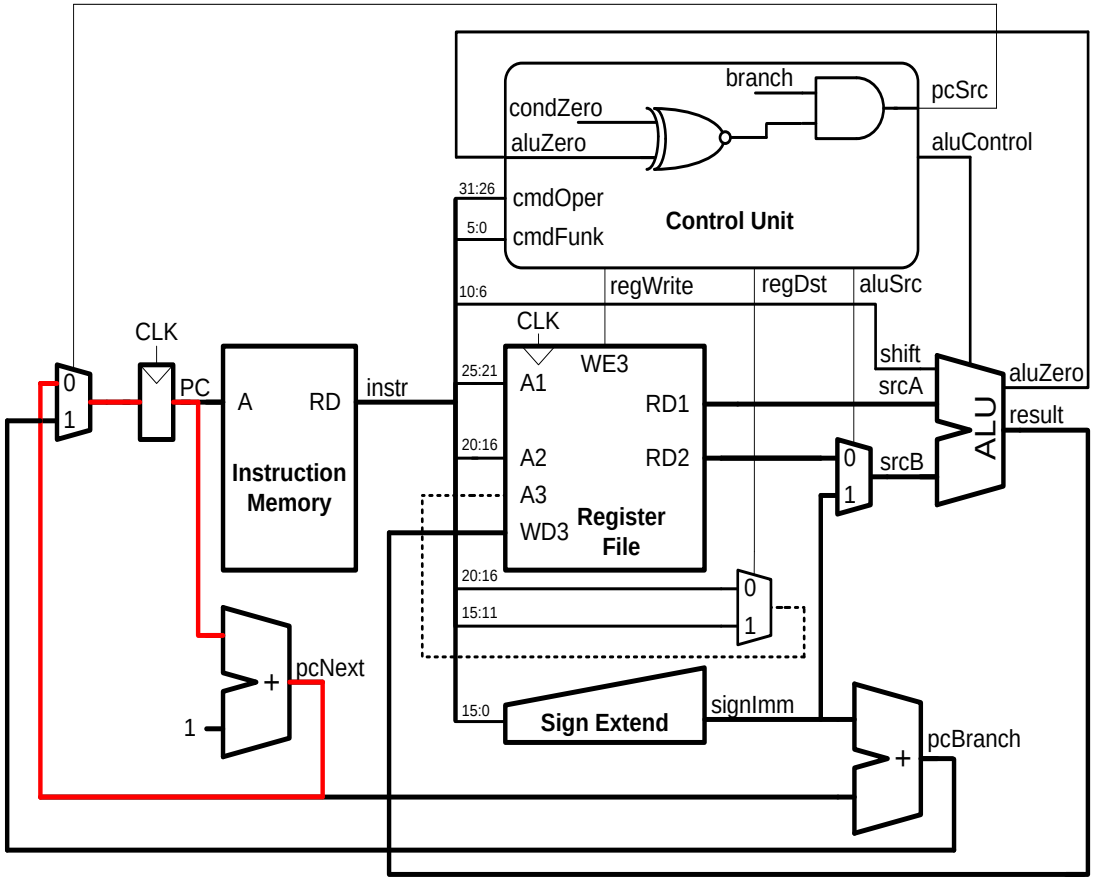
Процессор schoolMIPS: сигналы управления (7)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001						000



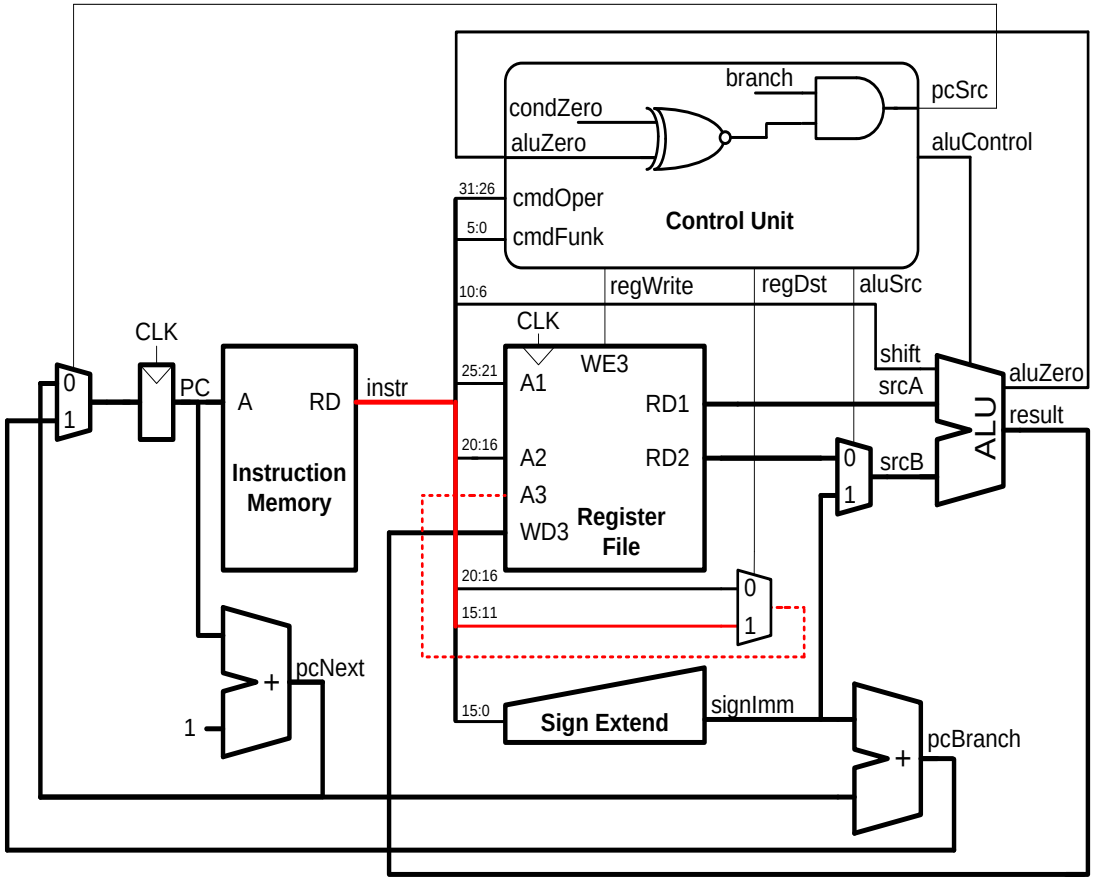
Процессор schoolMIPS: сигналы управления (8)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0				000



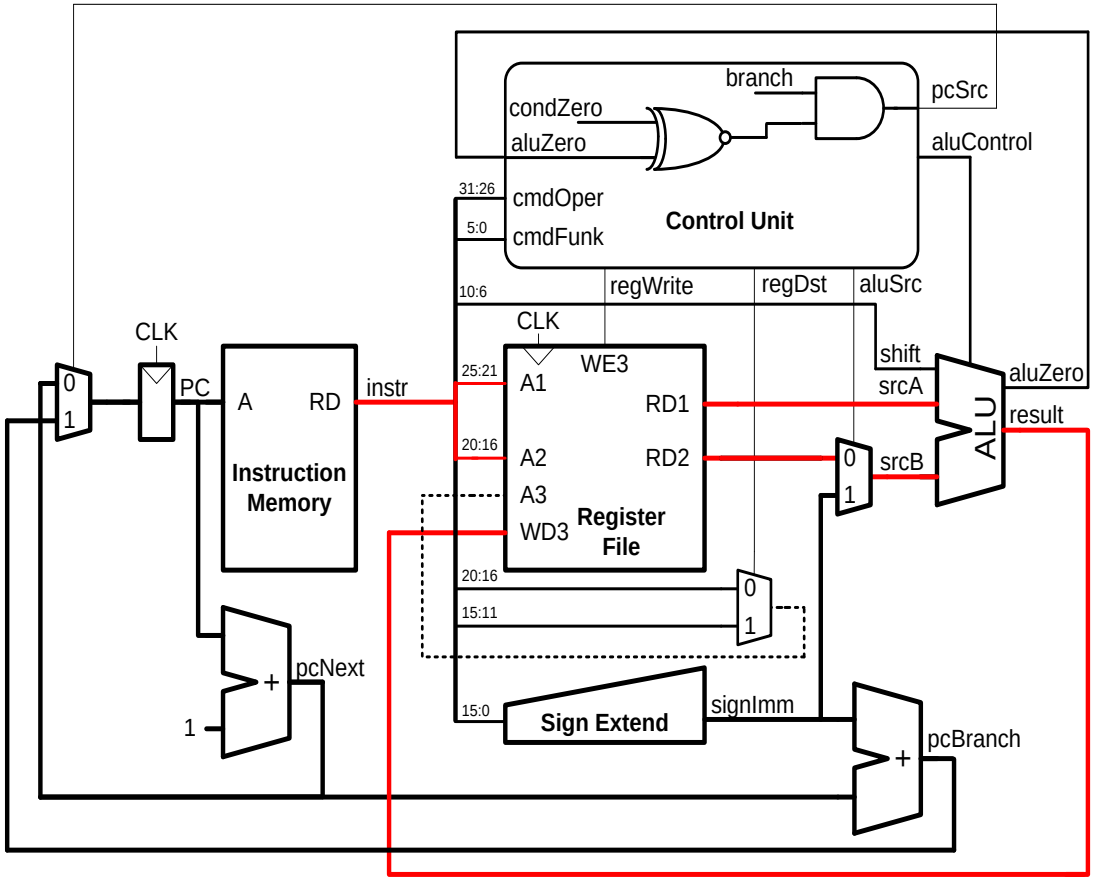
Процессор schoolMIPS: сигналы управления (9)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1			000



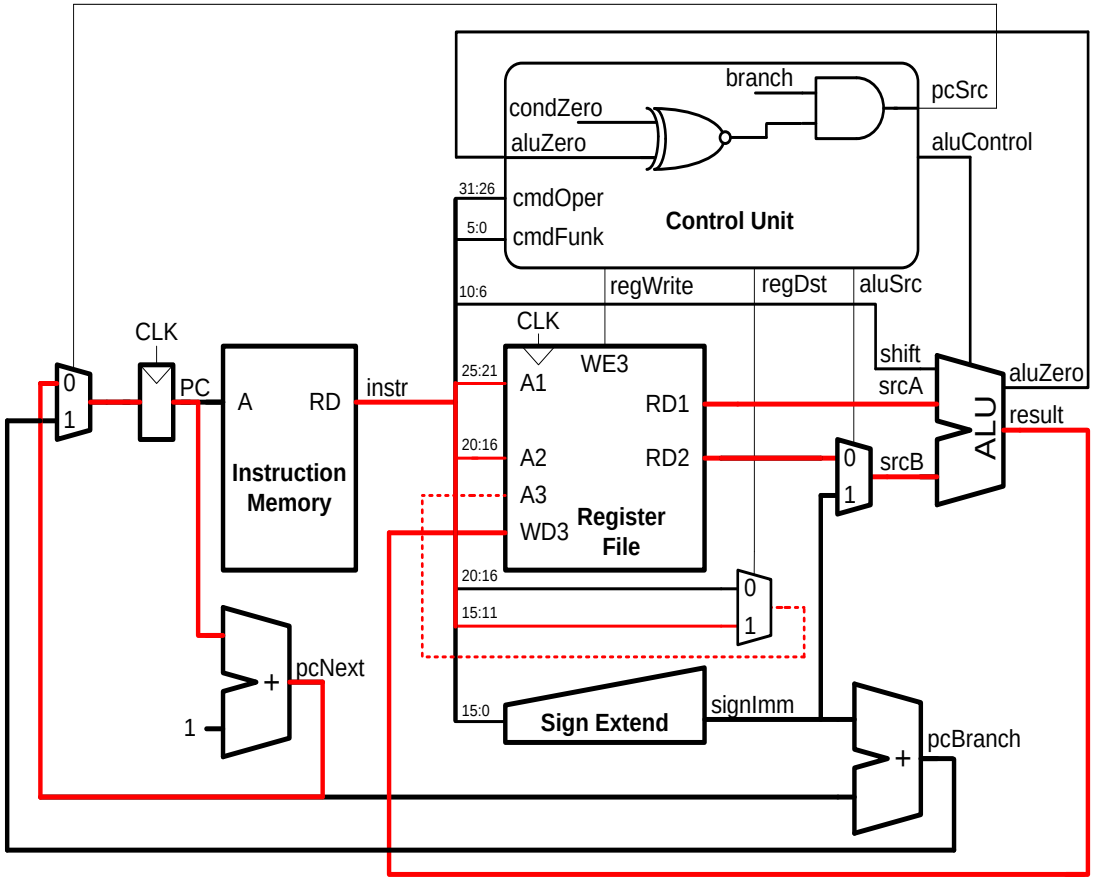
Процессор schoolMIPS: сигналы управления (10)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000



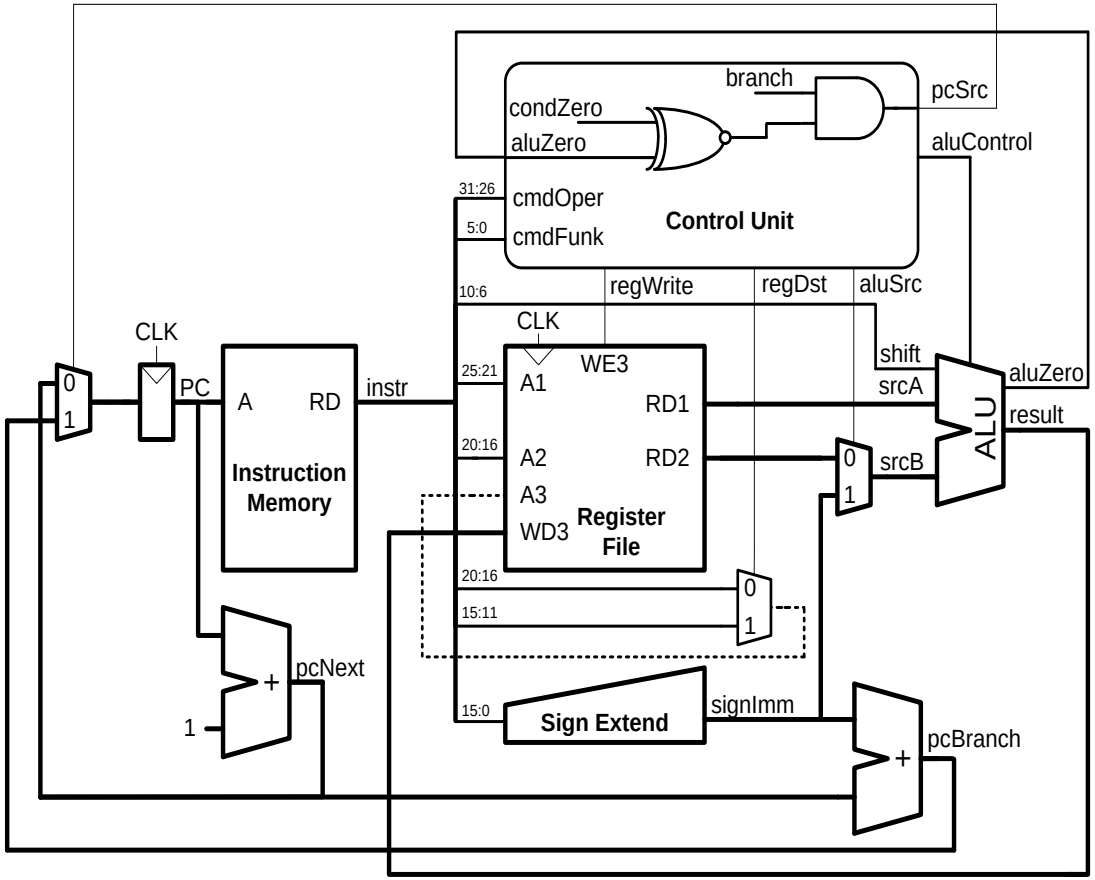
Процессор schoolMIPS: сигналы управления (11)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000



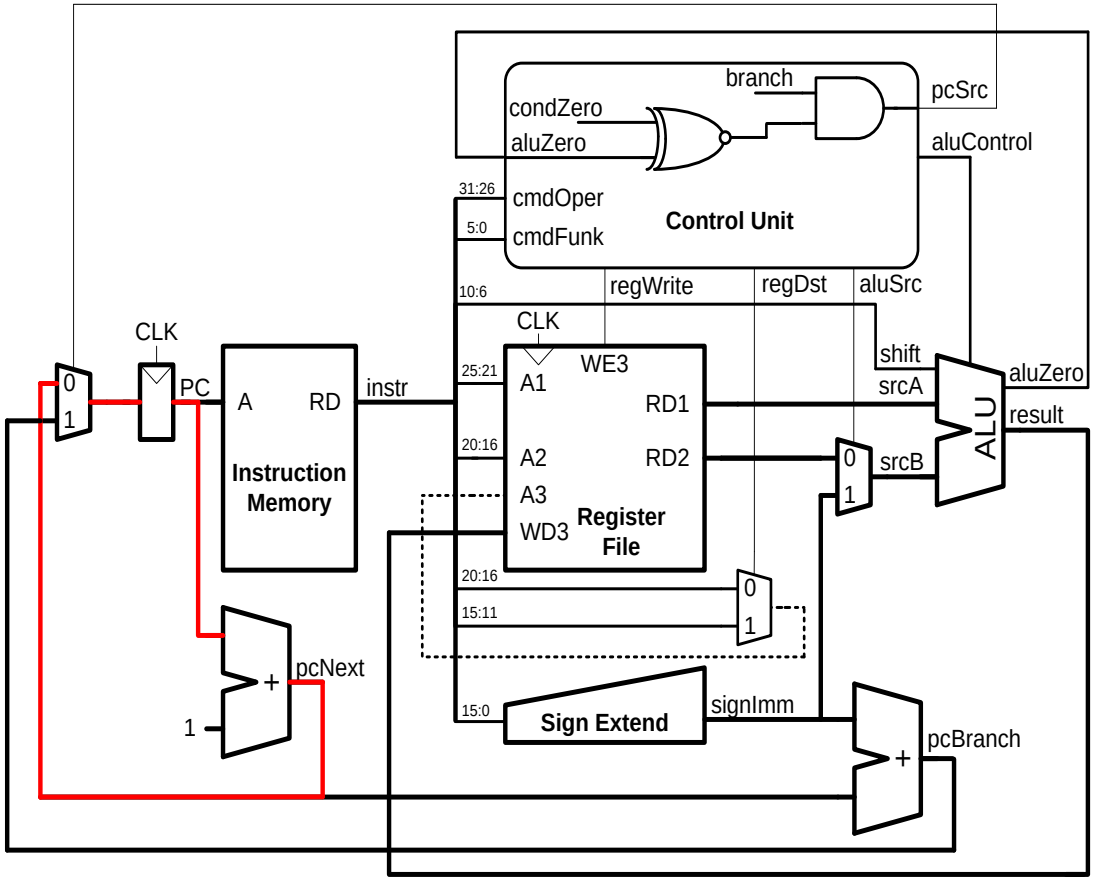
Процессор schoolMIPS: сигналы управления (12)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010						011



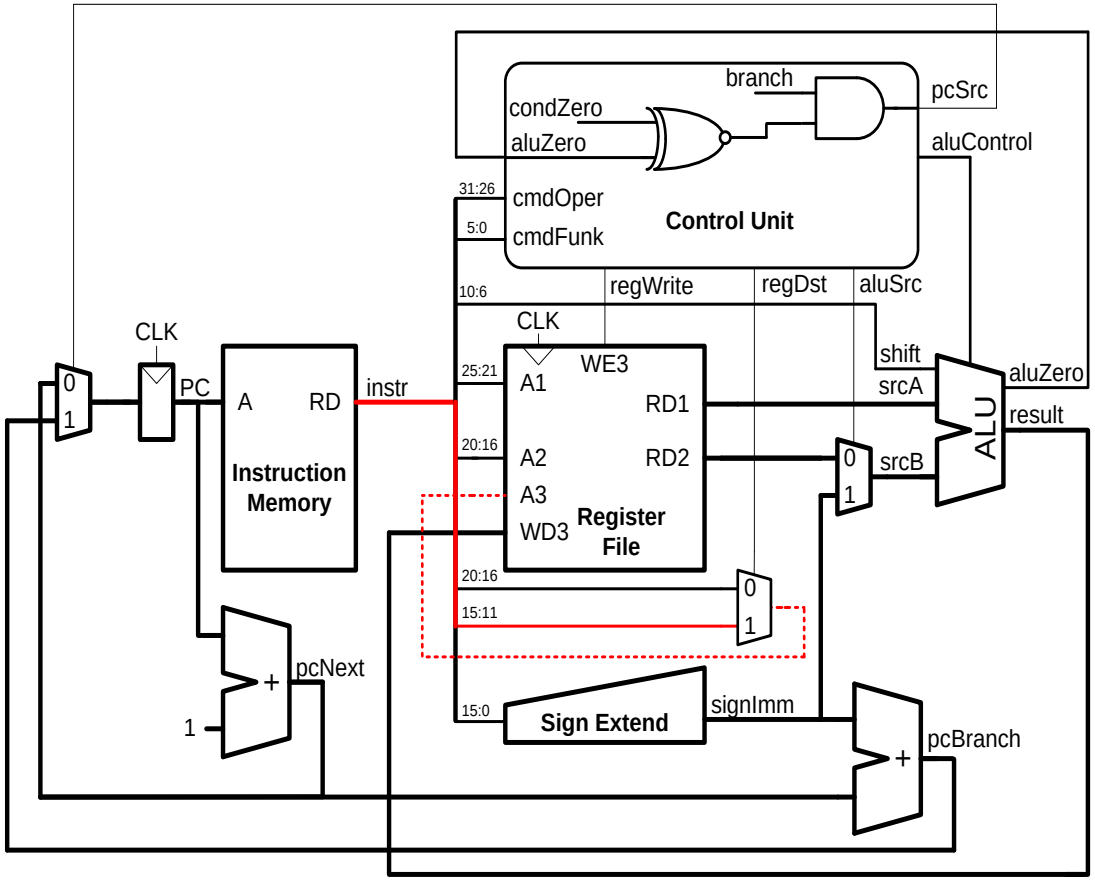
Процессор schoolMIPS: сигналы управления (13)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0				011



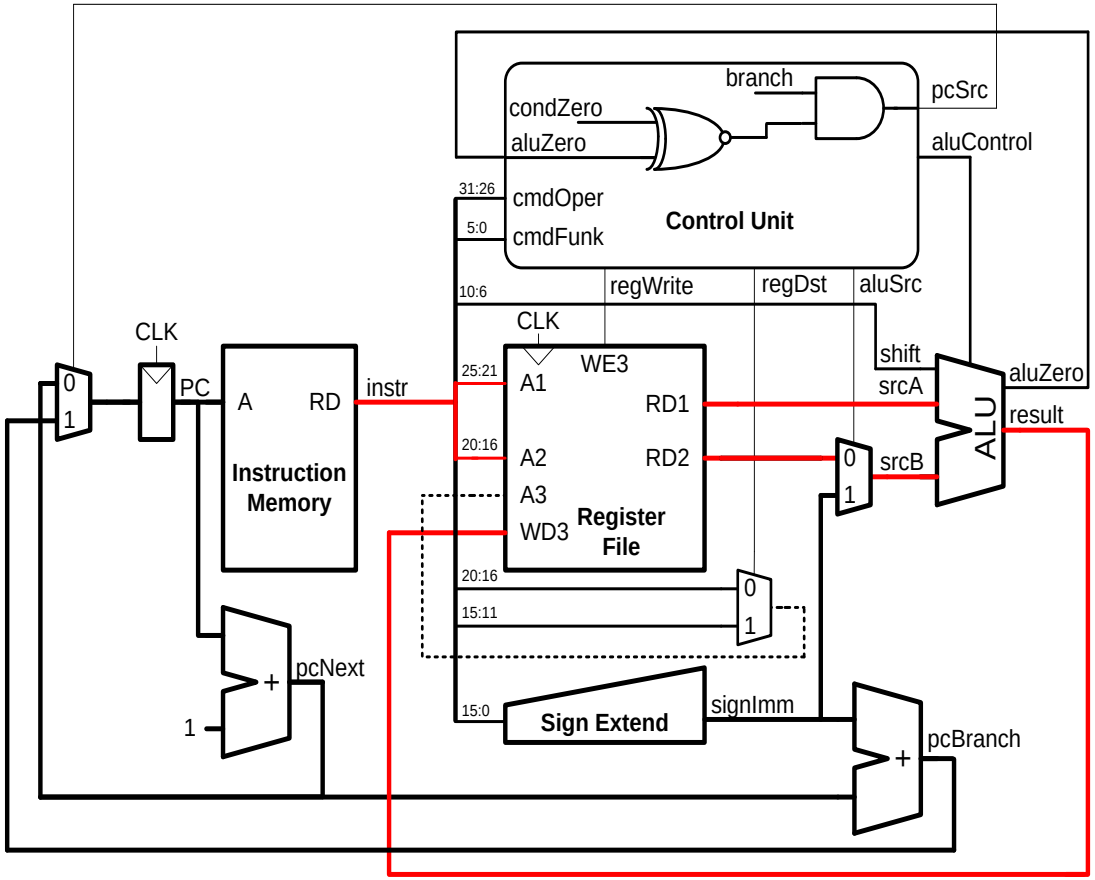
Процессор schoolMIPS: сигналы управления (14)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1			011



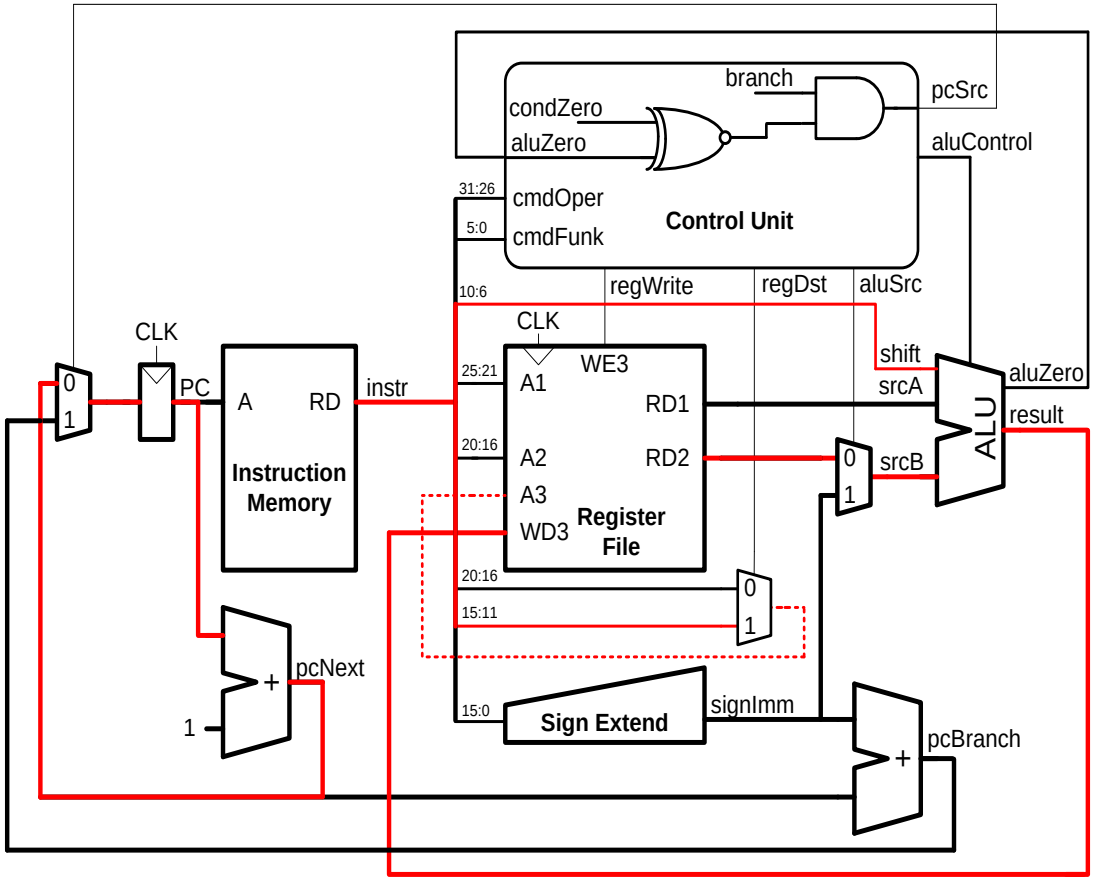
Процессор schoolMIPS: сигналы управления (15)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011



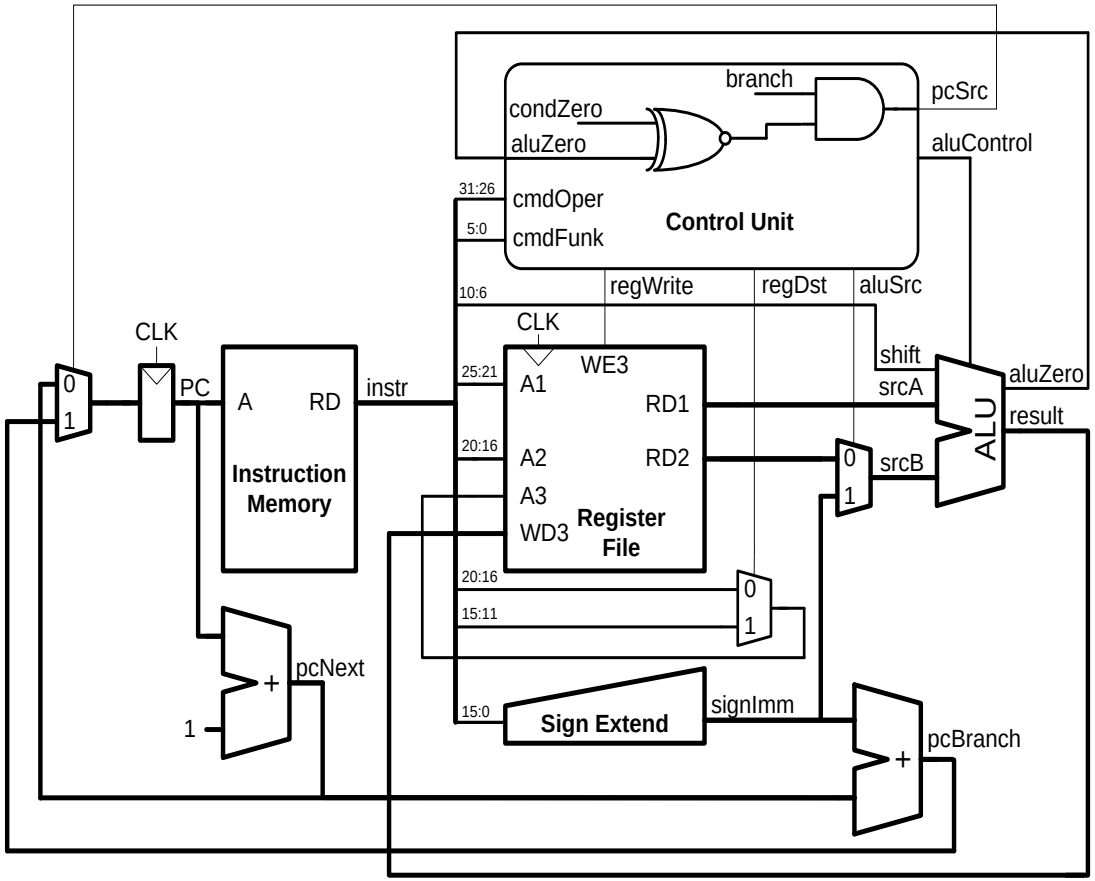
Процессор schoolMIPS: сигналы управления (16)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011



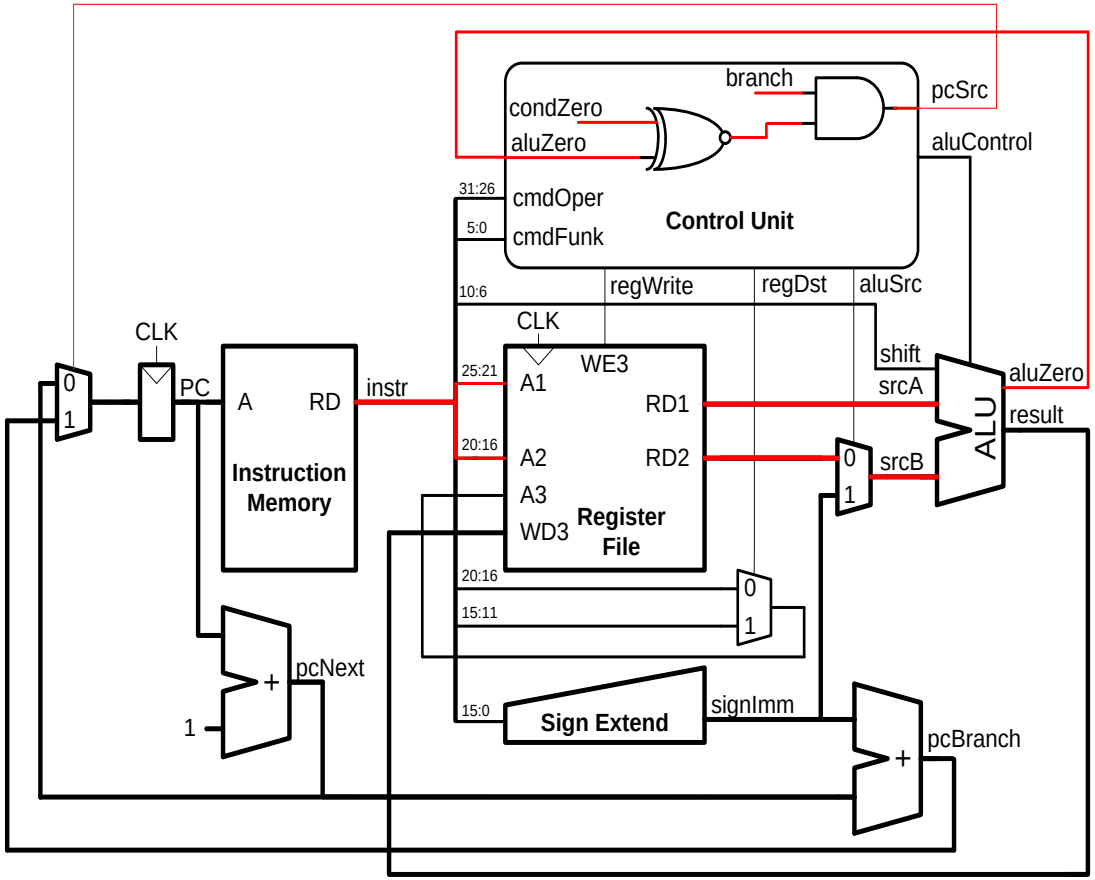
Процессор schoolMIPS: сигналы управления (17)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011
beq	000100	??????						101



Процессор schoolMIPS: сигналы управления (18)

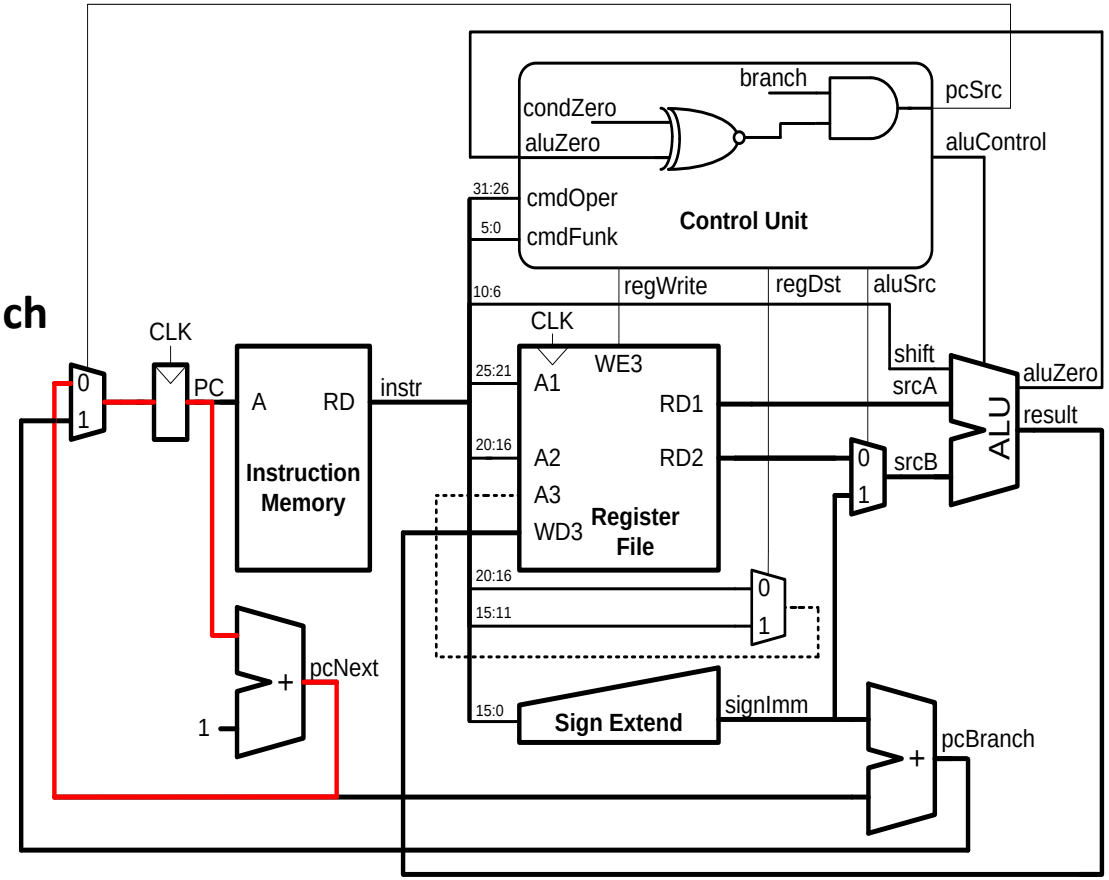
Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011
beq	000100	??????	1	1	0	0	0	101



Процессор schoolMIPS: сигналы управления (19)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011
beq	000100	??????	1	1	0	0	0	101

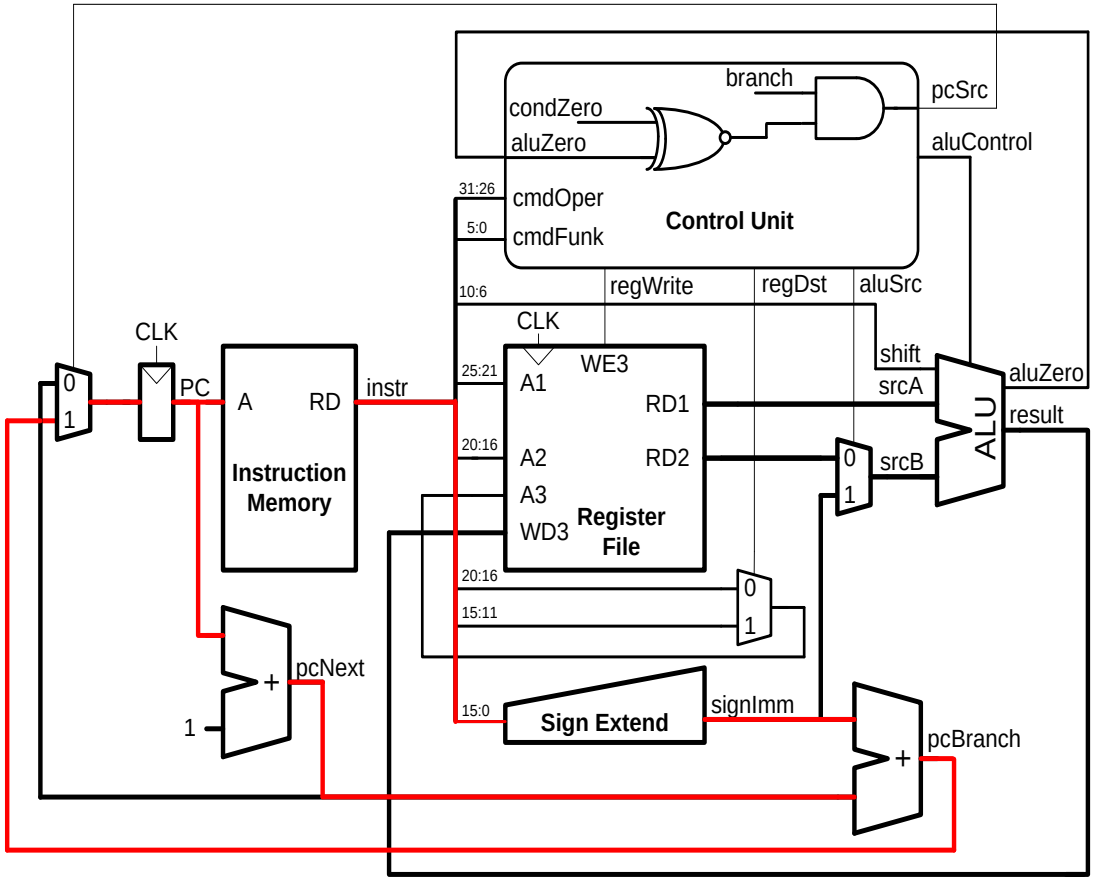
Rs != Rd => no branch



Процессор schoolMIPS: сигналы управления (20)

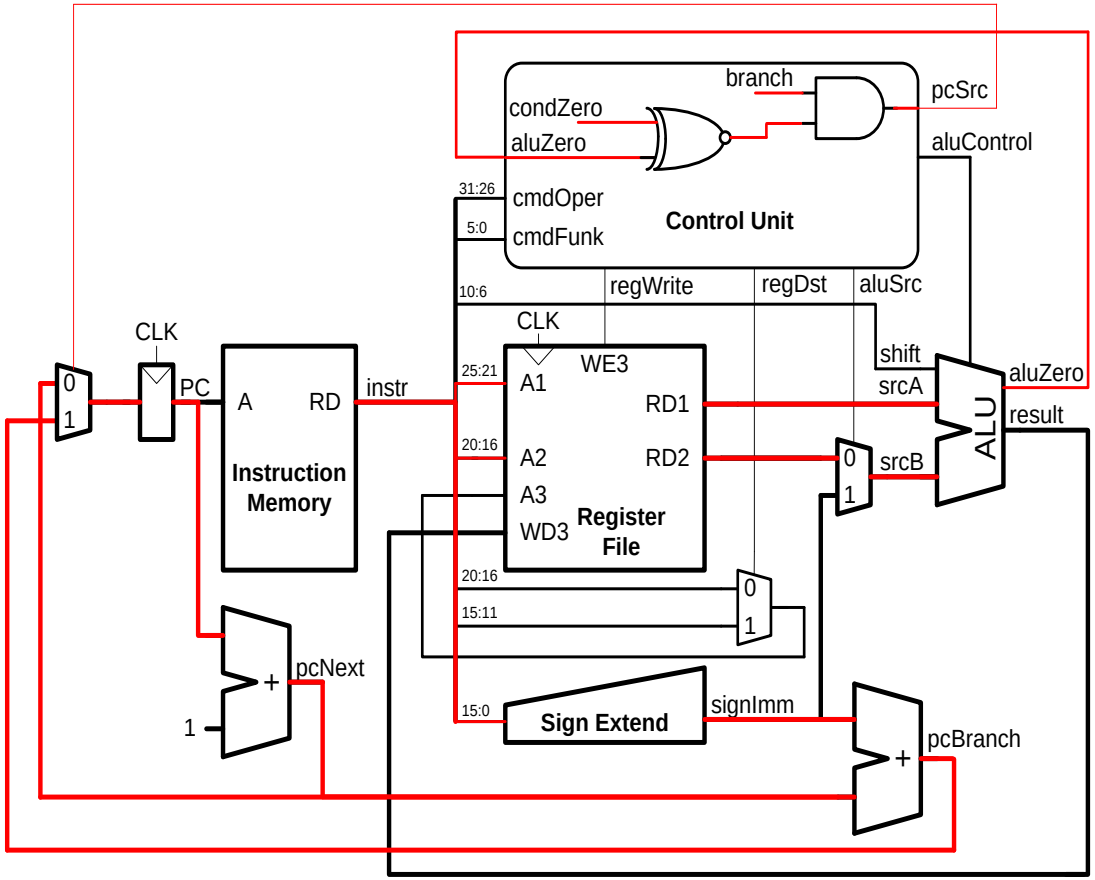
Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011
beq	000100	??????	1	1	0	0	0	101

Rs == Rd => branch



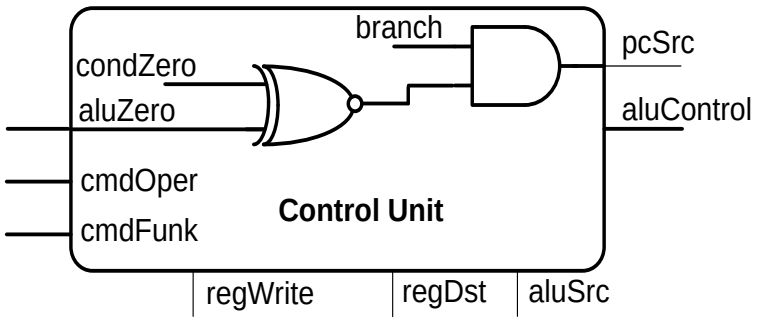
Процессор schoolMIPS: сигналы управления (21)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011
beq	000100	??????	1	1	0	0	0	101



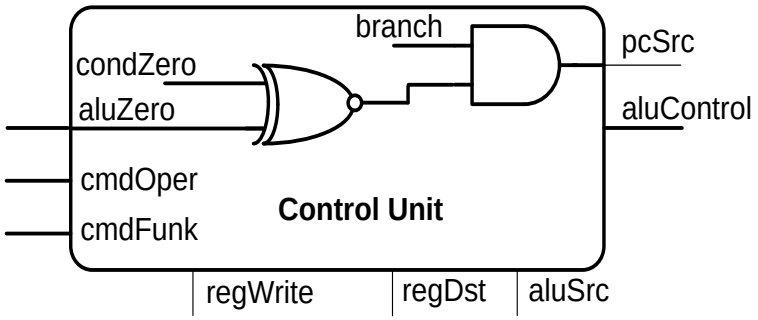
Реализация schoolMIPS. Устройство управления
Интерфейс модуля. Сигналы ветвления

```
// control unit (line 95-134)
module sm_control
(
    input       [5:0] cmdOper,
    input       [5:0] cmdFunk,
    input             aluZero,
    output            pcSrc,
    output reg        regDst,
    output reg        regWrite,
    output reg        aluSrc,
    output reg [2:0]  aluControl
);
    reg branch;
    reg condZero;
    assign pcSrc = branch & (aluZero == condZero);

    always @ (*) begin
    ...
    end
endmodule
```

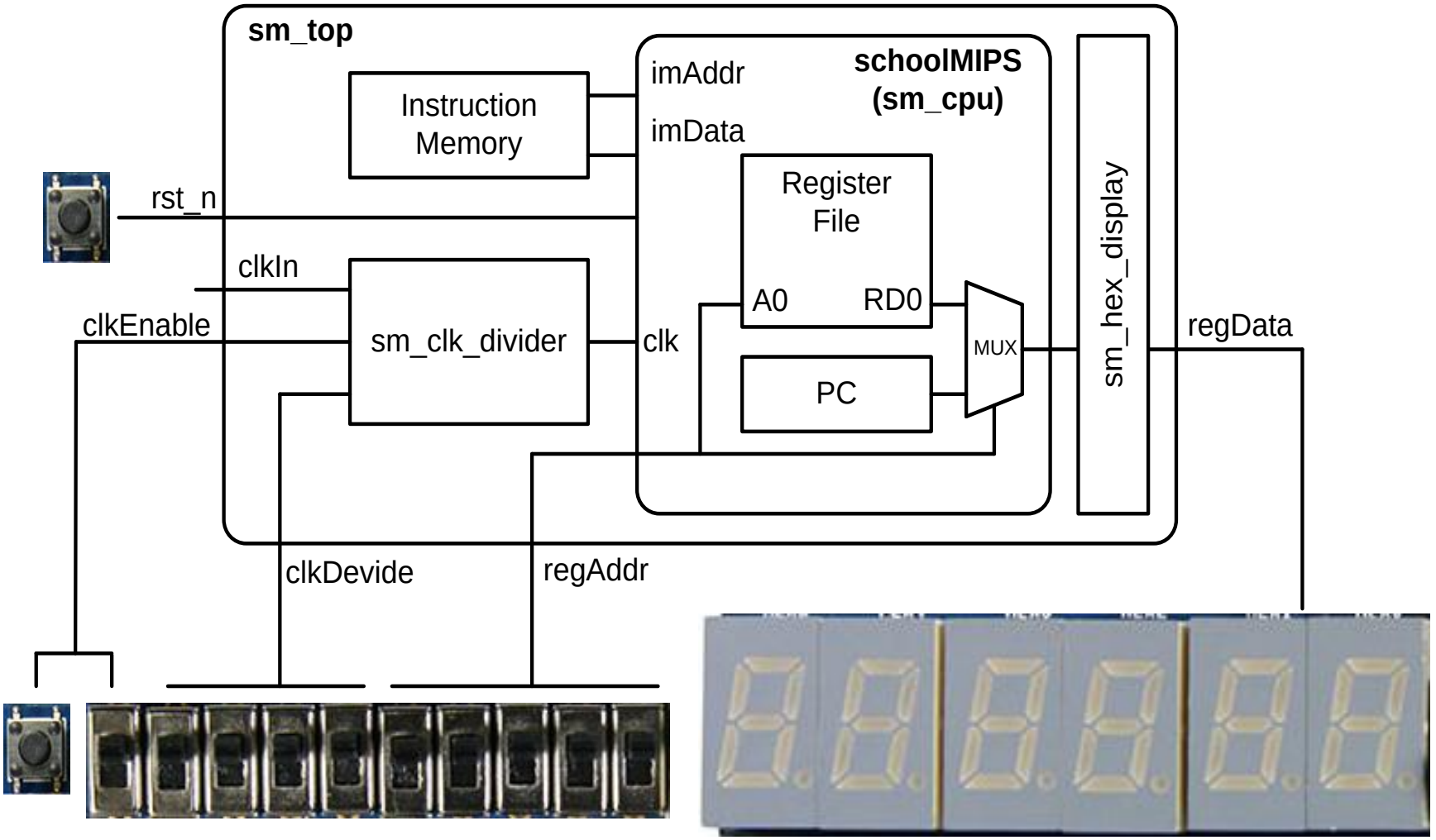


Реализация schoolMIPS. Устройство управления
Управляющие сигналы

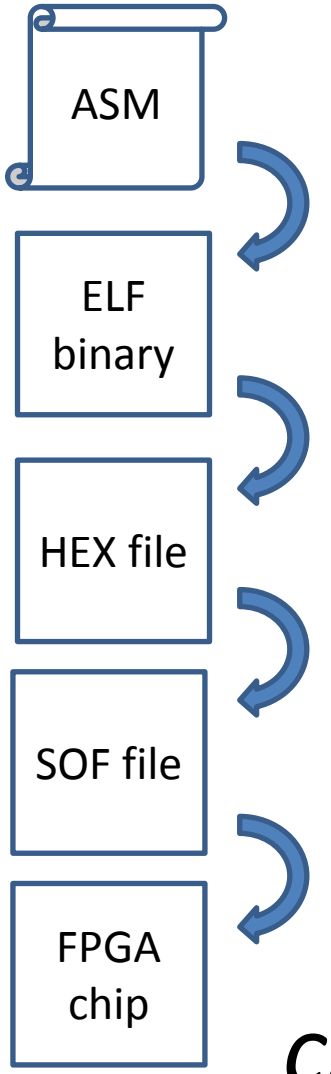
```
// control unit (line 95-134)
module sm_control
...
    always @ (*) begin
        //control signals default values
        branch = 1'b0;
        condZero = 1'b0;
        regDst = 1'b0;
        regWrite = 1'b0;
        aluSrc = 1'b0;
        aluControl = `ALU_ADD;

         casez( {cmdOper,cmdFunk} )
           default                    : ;
            { `C_SPEC, `F_ADDU } : begin
                                      regDst = 1'b1;
                                      regWrite = 1'b1;
                                      aluControl = `ALU_ADD;
                                    end

           ...
         endcase
    end
endmodule
```



Реализация schoolMIPS на отладочной плате



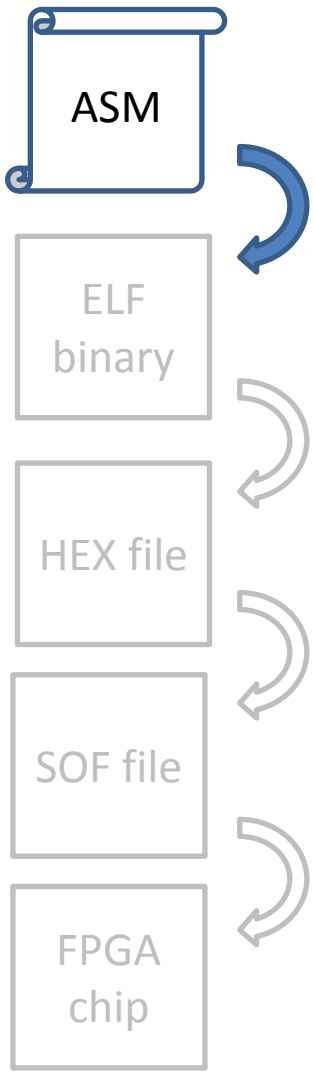
Программирование schoolMIPS



- gcc (MIPS toolchain)
- BIN2HEX converter
- synthesis tool (Quartus Prime)
- programmer (Quartus Prime)

см. Руководство пользователя

Программа на ассемблере MIPS




```
# program/01_fibonacci/main.S (line 3-13)

            .text

start:      move $t0, $0
            li   $t1, 1
            move $v0, $t1

fibonacci:  addu $t0, $t0, $t1
            move $v0, $t0
            addu $t1, $t0, $t1
            move $v0, $t1
            b    fibonacci
```

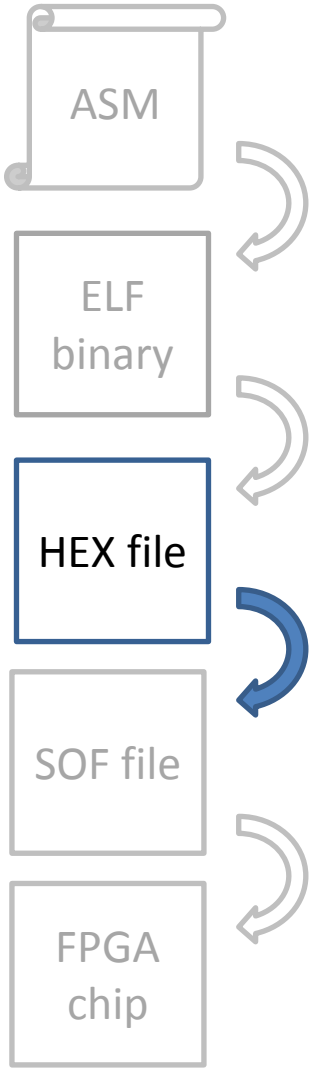
gcc (MIPS toolchain)

Бинарный исполняемый файл


```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00  .ELF............
00000010: 02 00 08 00 01 00 00 00 00 00 00 00 34 00 00 00  ............4...
00000020: 04 01 01 00 00 10 00 50 34 00 20 00 01 00 28 00  .......P4.....(.
00000030: 05 00 02 00 01 00 00 00 00 00 01 00 00 00 00 00  ................
00000040: 00 00 00 00 24 00 00 00 24 00 00 00 05 00 00 00  ....$...$.......
00000050: 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

BIN2HEX converter

HEX-файл



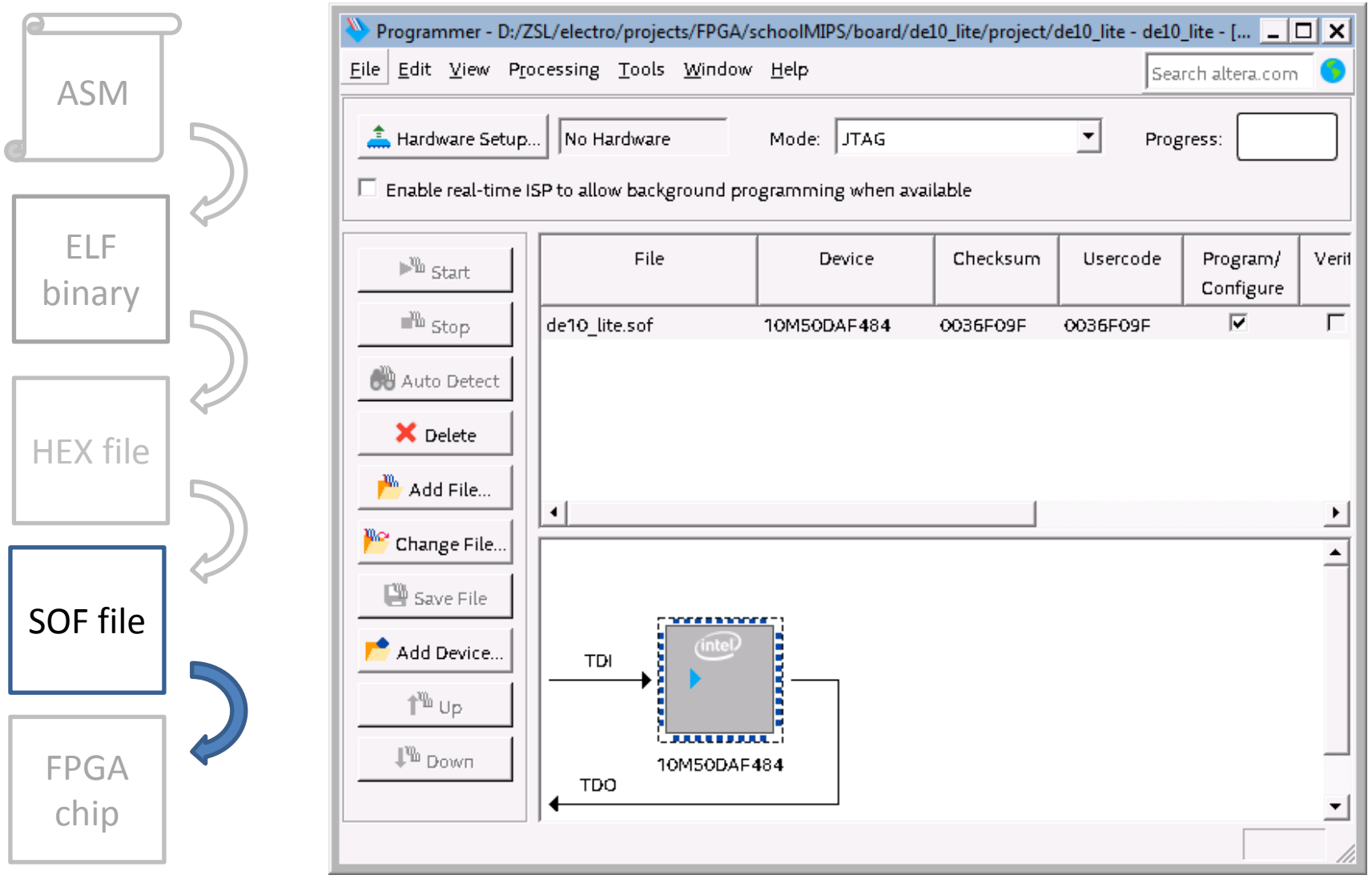
program.hex

```
1   @00000000
2   00004025
3   24090001
4   01201025
5   01094021
6   01001025
7   01094821
8   01201025
9   1000fffb
10  00000000
11
```

Verilog Sources:
*.v, *.vh

synthesis tool (Quartus Prime)

Файл конфигурации ПЛИС



Отладочная плата ПЛИС с загруженной конфигурацией

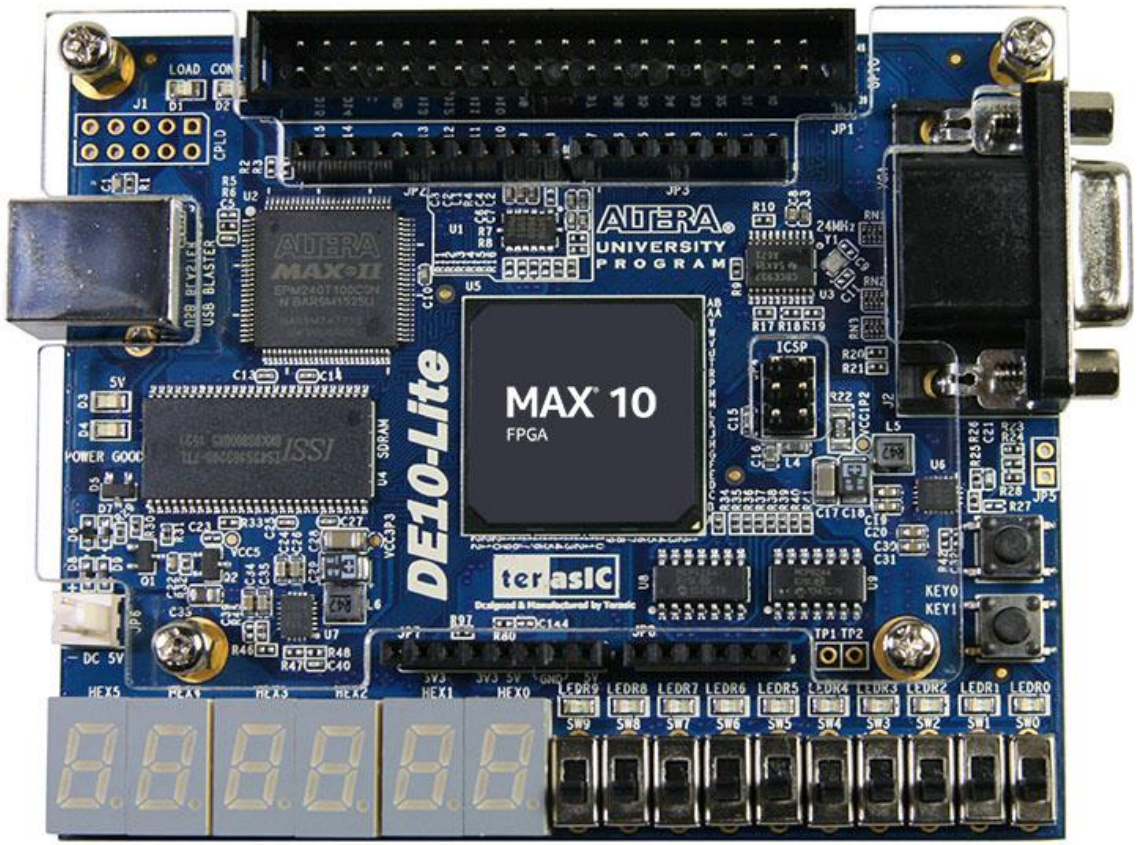
ASM

ELF binary

HEX file

SOF file

FPGA chip



Что дальше?

- учебник «Цифровая схемотехника и архитектура компьютера» авторов Дэвида Харриса и Сары Харрис. Бесплатный русский перевод второго издания этого учебника можно загрузить с сайта компании Imagination Technologies (link)
- процессор MIPSfpga – промышленное процессорное ядро, исходный код которого доступен под академической лицензией в рамках Imagination University Programme (link)

Ваши вопросы?