schoolMIPS CPU Core

Stanislav Zhelnio
2018

Project download link:
https://github.com/MIPSfpga/schoolMIPS

Project documents:
https://github.com/MIPSfpga/schoolMIPS/wiki

Thanks to

- David Harris and Sarah Harris, the authors of the great book “Digital Design and Computer Architecture”. schoolMIPS is based on the CPU that is described in this book
- the team of the “Digital Design and Computer Architecture” book translators
- “Young Russian Chip Architects” Conference Members
- Yuri Panchul, Senior Hardware Design Engineer at Imagination Technologies and MIPS. The author of the schoolMIPS Idea.
- Stanislav Zhelnio, CPU architecture, code and docs
- Alexander Romanov, MIEM HSE, CPU architecture, test and port

What is schoolMIPS?

- the simplest CPU core
- designed for using in the basic course of digital design and computer architecture
- written on pure Verilog
- subset of MIPS instructions implemented

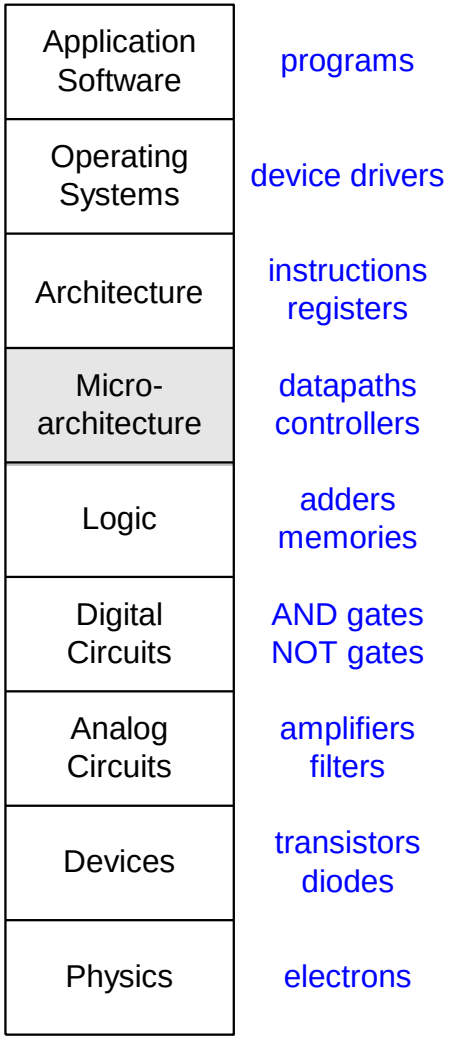
schoolMIPS versions and features

Feature \ GIT branch	00_simple	01_mmio	02_irq	03_pipeline	04_pipeline_irq	05_pipeline_ahb
Single cycle CPU	V	V	V	V	V	V
Example program (assembler)	V	V	V	V	V	V
Build scripts	shell		make			
Data memory (Block RAM)		V	V	V	V	V
Basic memory mapped IO (single cycle access)		V				
System Timer, Interrupts & Exceptions			V		V	V
Pipelined CPU				V	V	V
Pipelined memory mapped IO (AHB-Lite)						V
Scratchpad RAM						V
Example program (C)						V

Introduction

- **Microarchitecture:** how to implement an architecture in hardware

- Processor:
  - **Datapath:** functional blocks (ALU, Register File, etc)
  - **Control:** control signals



Microarchitecture

- Multiple implementations for a single architecture :
  - **Single-cycle:** Each instruction executes in a single cycle
  - **Multicycle:** Each instruction is broken into series of shorter steps
  - **Pipelined:** Each instruction broken up into series of steps & multiple instructions execute at once

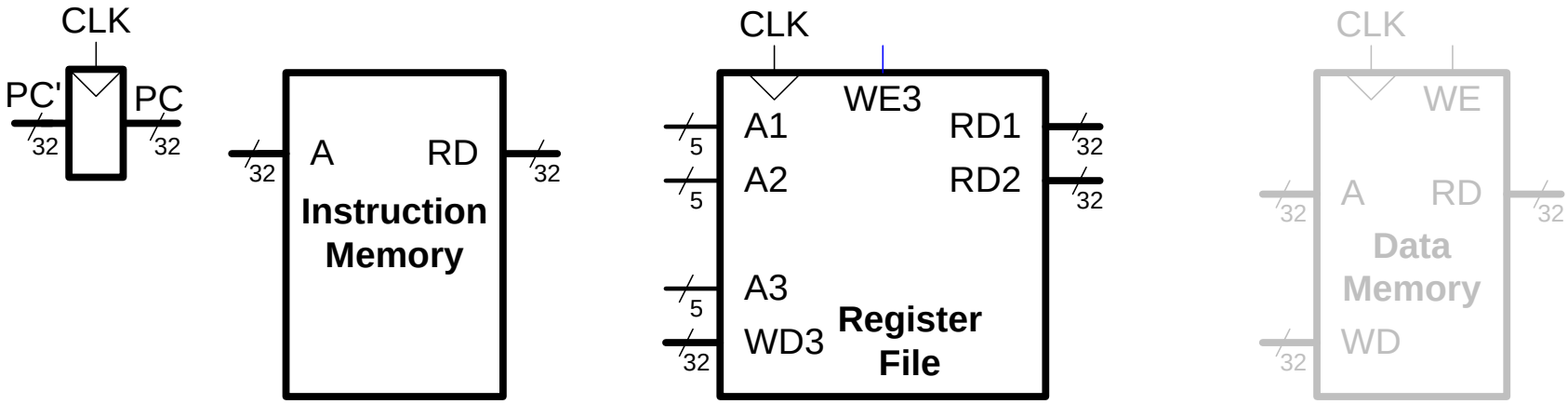
The schoolMIPS CPU (00_simple)

- Single-cycle
- no Data Memory
- The word by word addressing of Instruction Memory
- Subset of MIPS Instructions:
  - R-type instructions (both operands are from RF): `addu, or, srl, sltu, subu`
  - I-type instructions (one of operand is a constant): `addiu, lui`
  - I-type instructions (branch): `beq, bne`

Architectural State

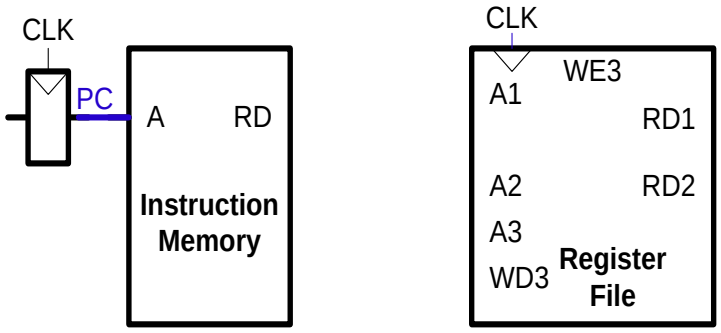
- Determines everything about a processor :
  - Program Counter (PC)
  - Register File (RF)
    with 32 General Purpose Registers (GPR)
  - memory (instructions, data)

MIPS State Elements



schoolMIPS: **addiu** instruction

Step 1: Fetch instruction

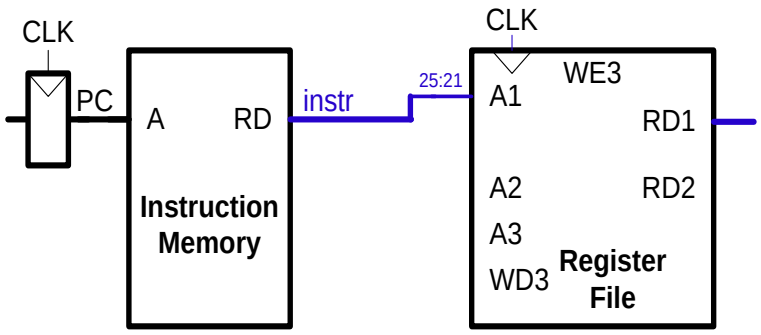


I-type. Integer Add Immediate, rt = rs + Immediate

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

schoolMIPS: **addiu** instruction

Step 2: Read source operands from RF

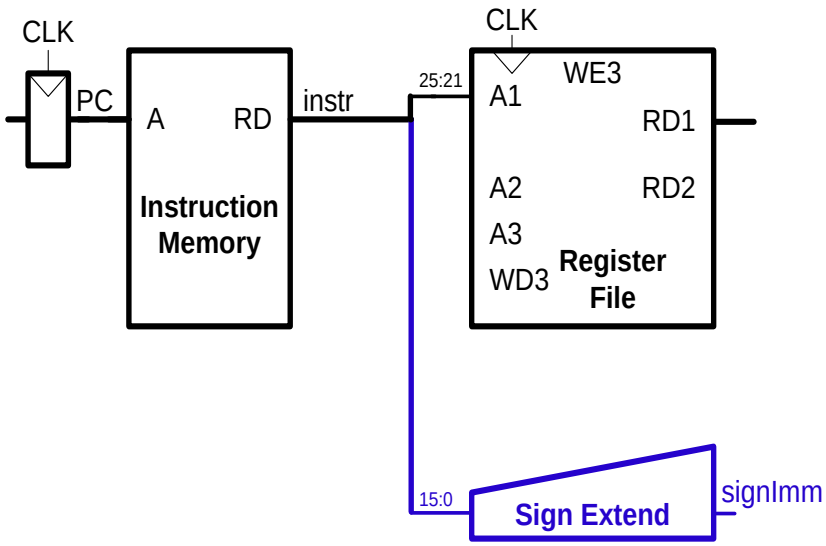


I-type. Integer Add Immediate, rt = rs + Immediate

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

schoolMIPS: **addiu** instruction

Step 3: Sign-extend the immediate operand

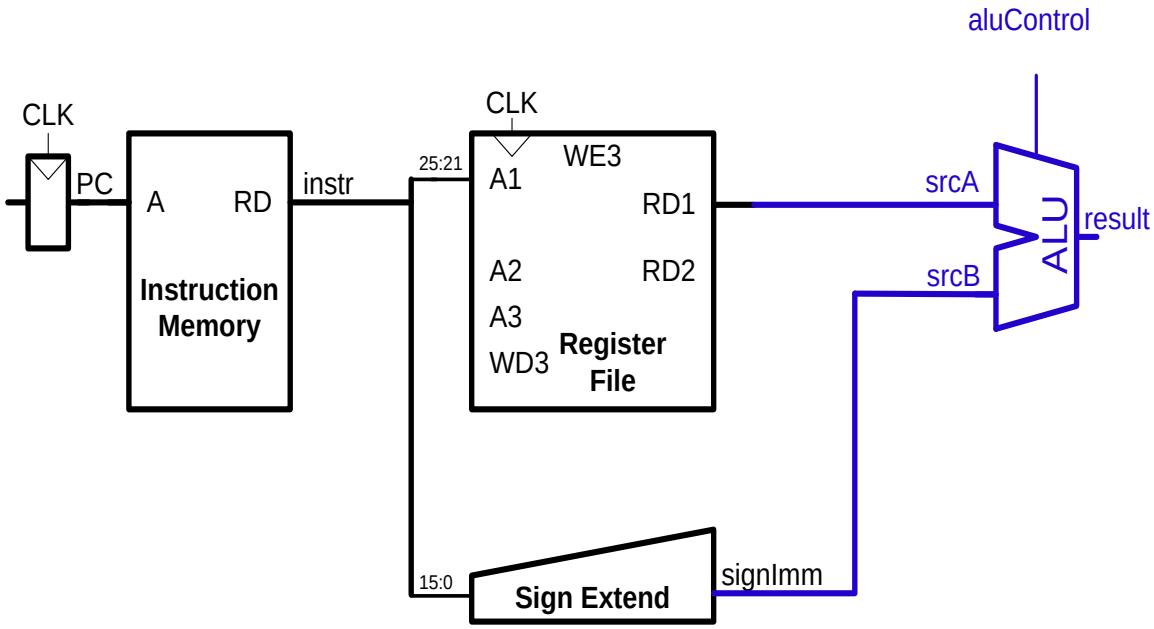


I-type. Integer Add Immediate, rt = rs + Immediate

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

schoolMIPS: **addiu** instruction

Step 4: compute the arithmetic operation result

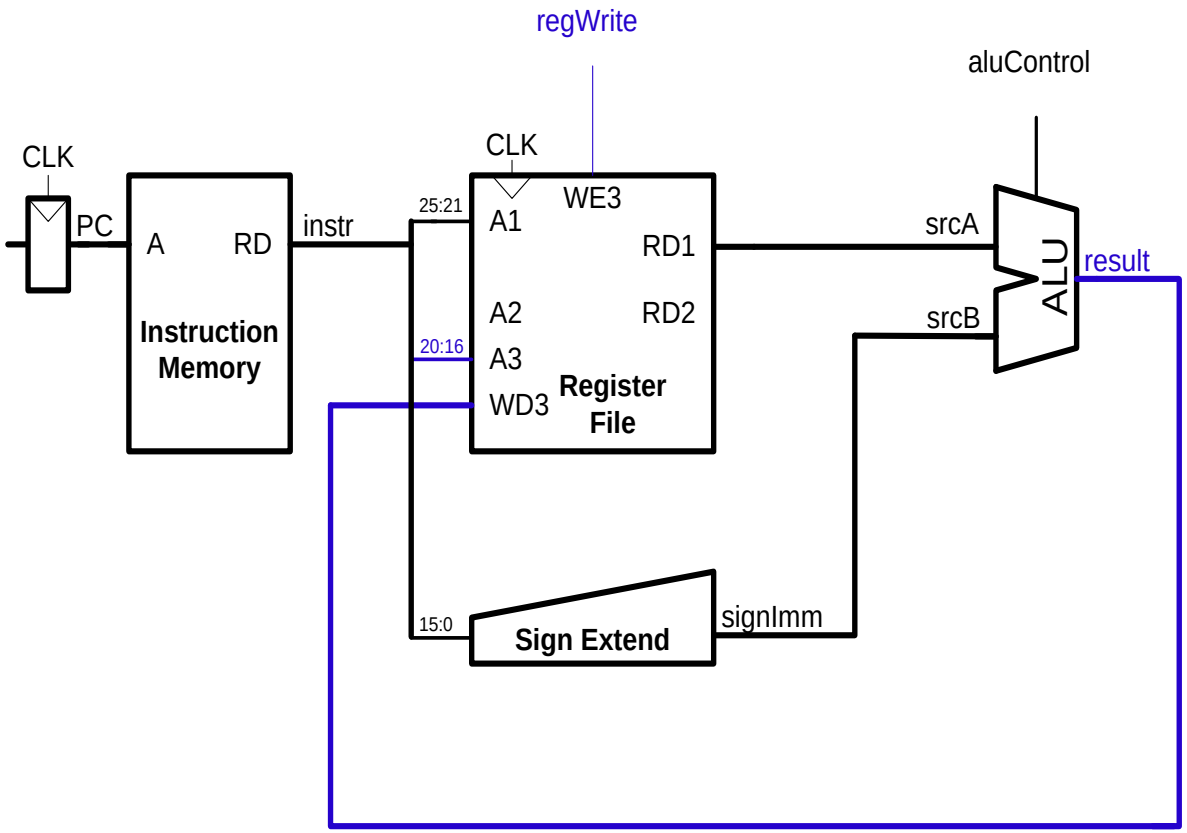


I-type. Integer Add Immediate, rt = rs + Immediate

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

schoolMIPS: **addiu** instruction

Step 5: write the result to the Register File

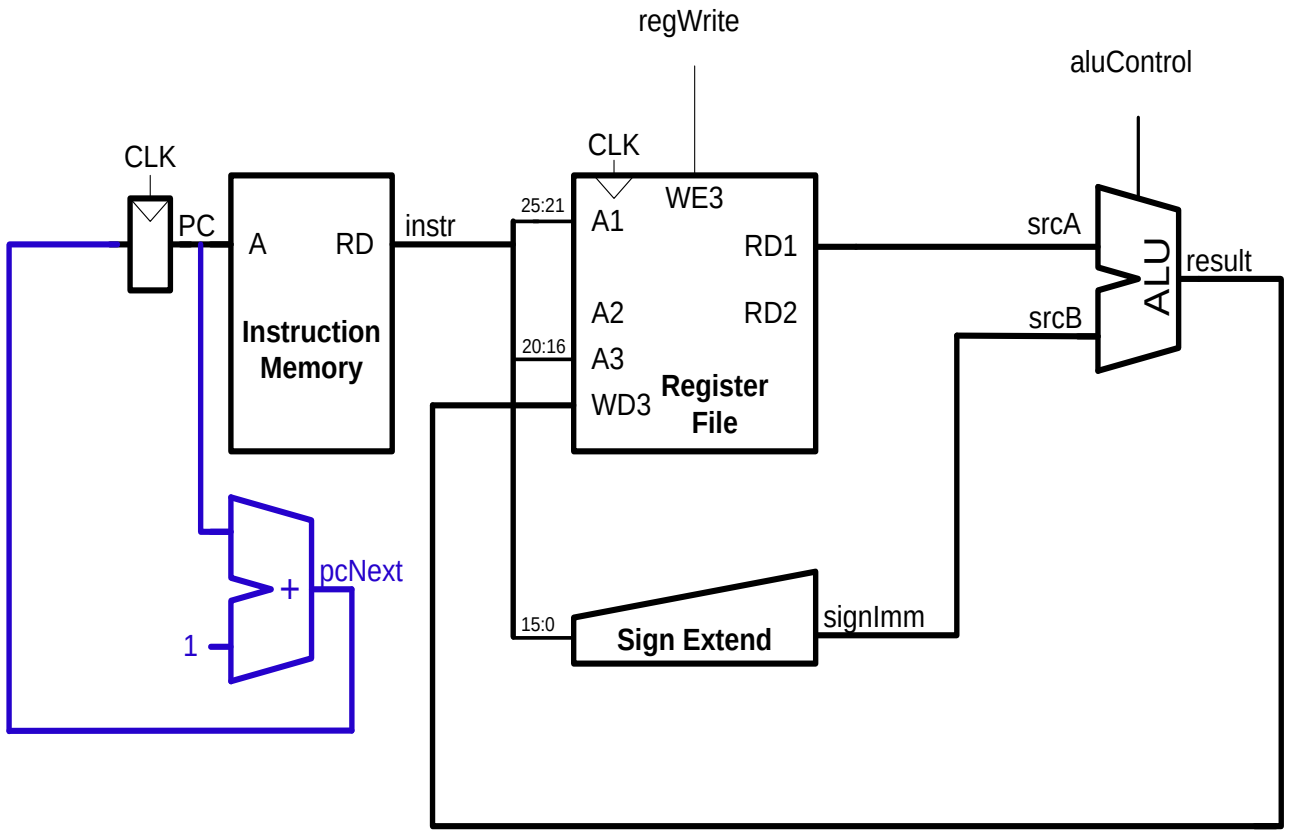


I-type. Integer Add Immediate, rt = rs + Immediate

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

schoolMIPS: **addiu** instruction

Step 6: Determine address of next instruction

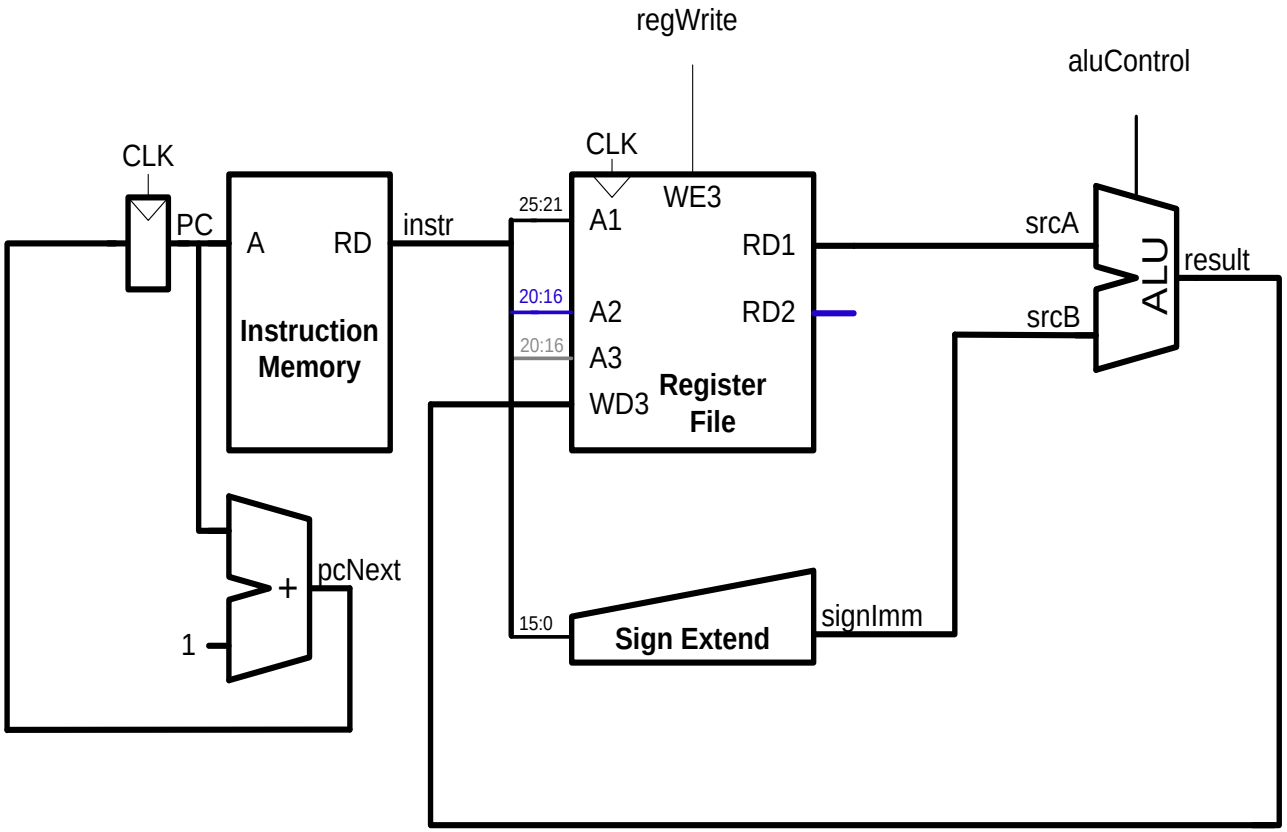


I-type. Integer Add Immediate, rt = rs + Immediate

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

schoolMIPS: **addu** instruction

- Read the second operand from Register File

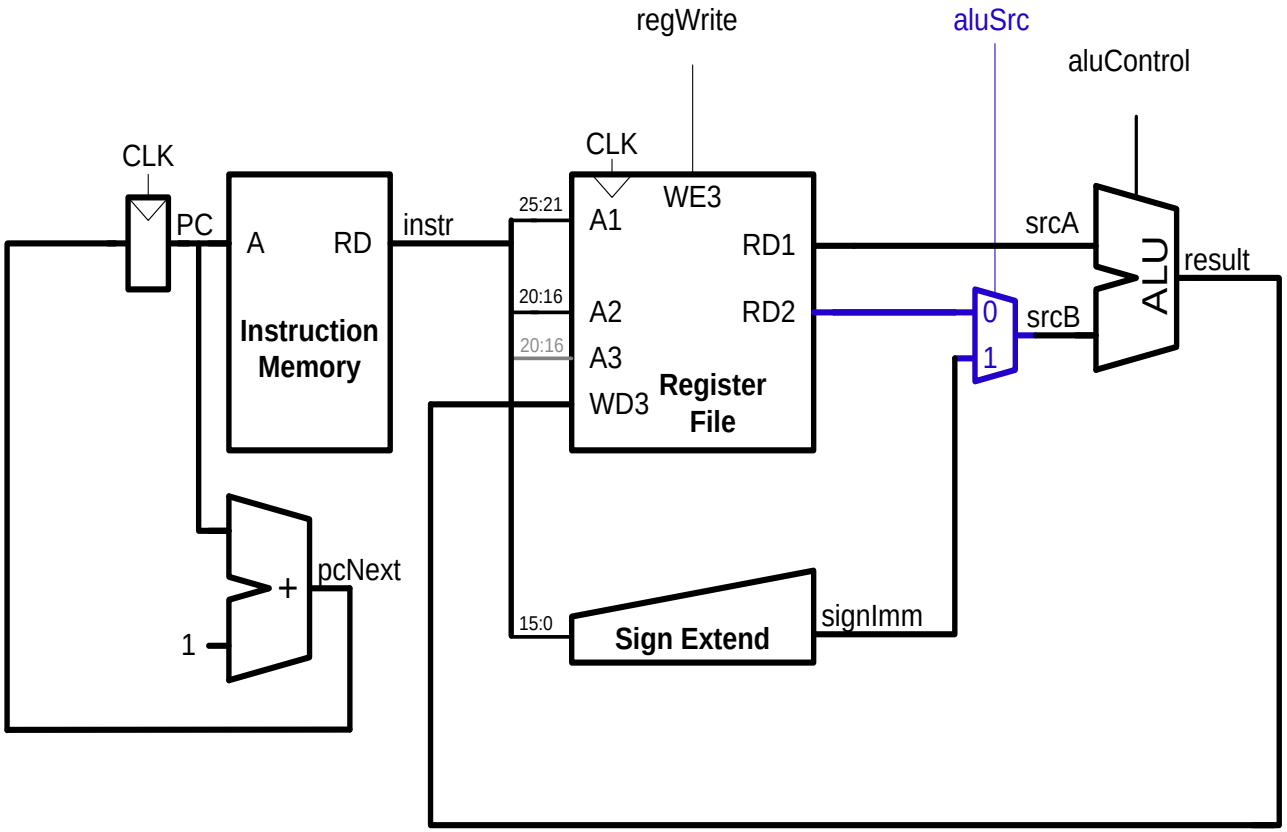


R-type. Integer Add Unsigned, **rd = rs + rt**

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

schoolMIPS: **addu** instruction

- use the register value operand (rt) instead of sign extended value

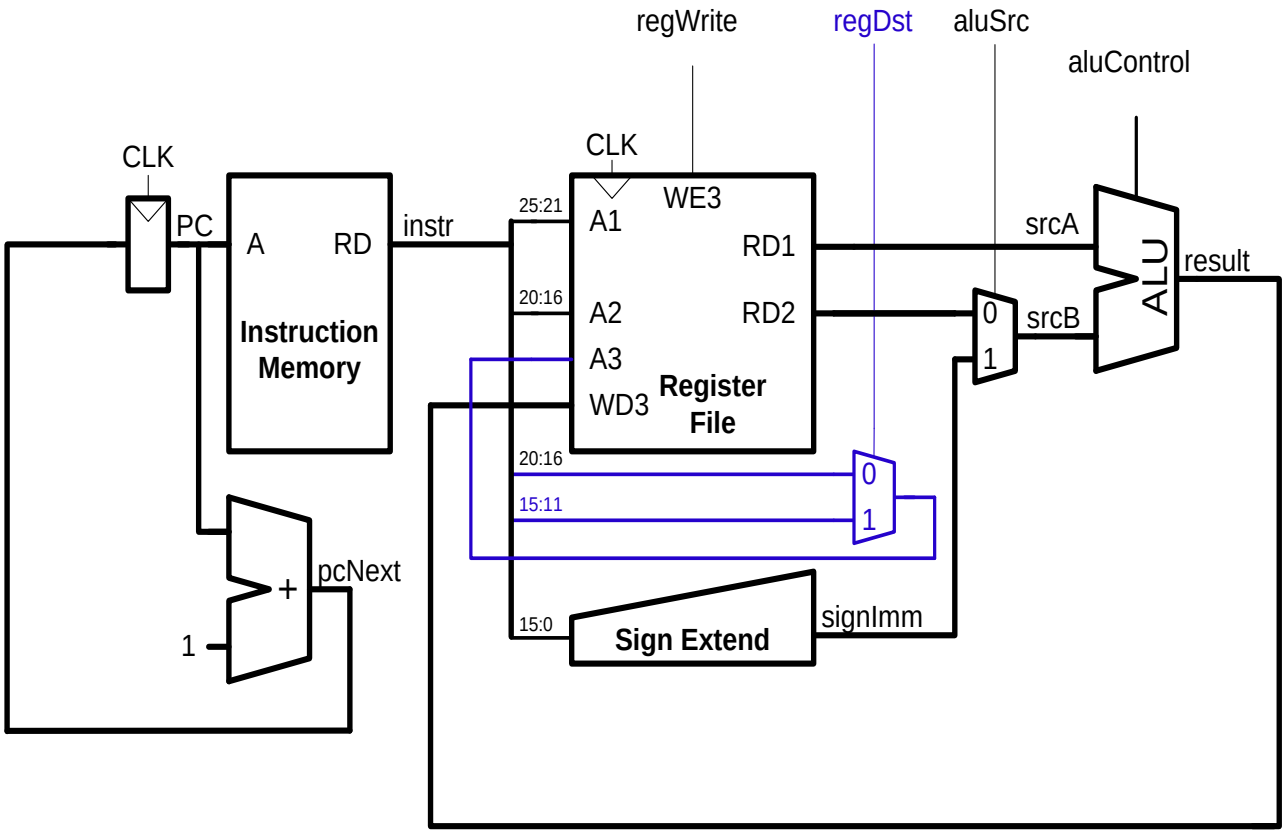


R-type. Integer Add Unsigned, **rd = rs + rt**

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

schoolMIPS: **addu** instruction

- Write the result to `rd` (instead of `rt`)

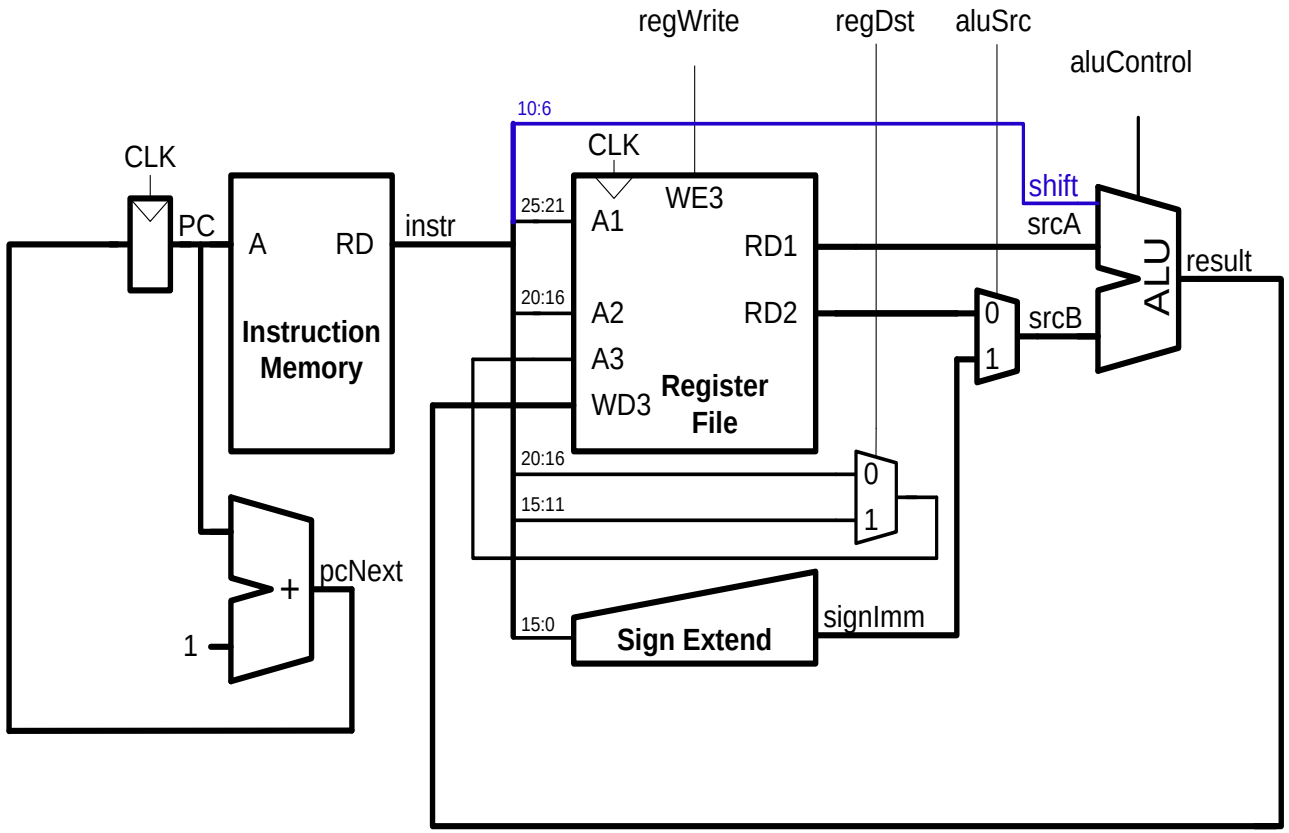


R-type. Integer Add Unsigned, **rd = rs + rt**

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

schoolMIPS: `srl` instruction

- read the shift amount from instruction

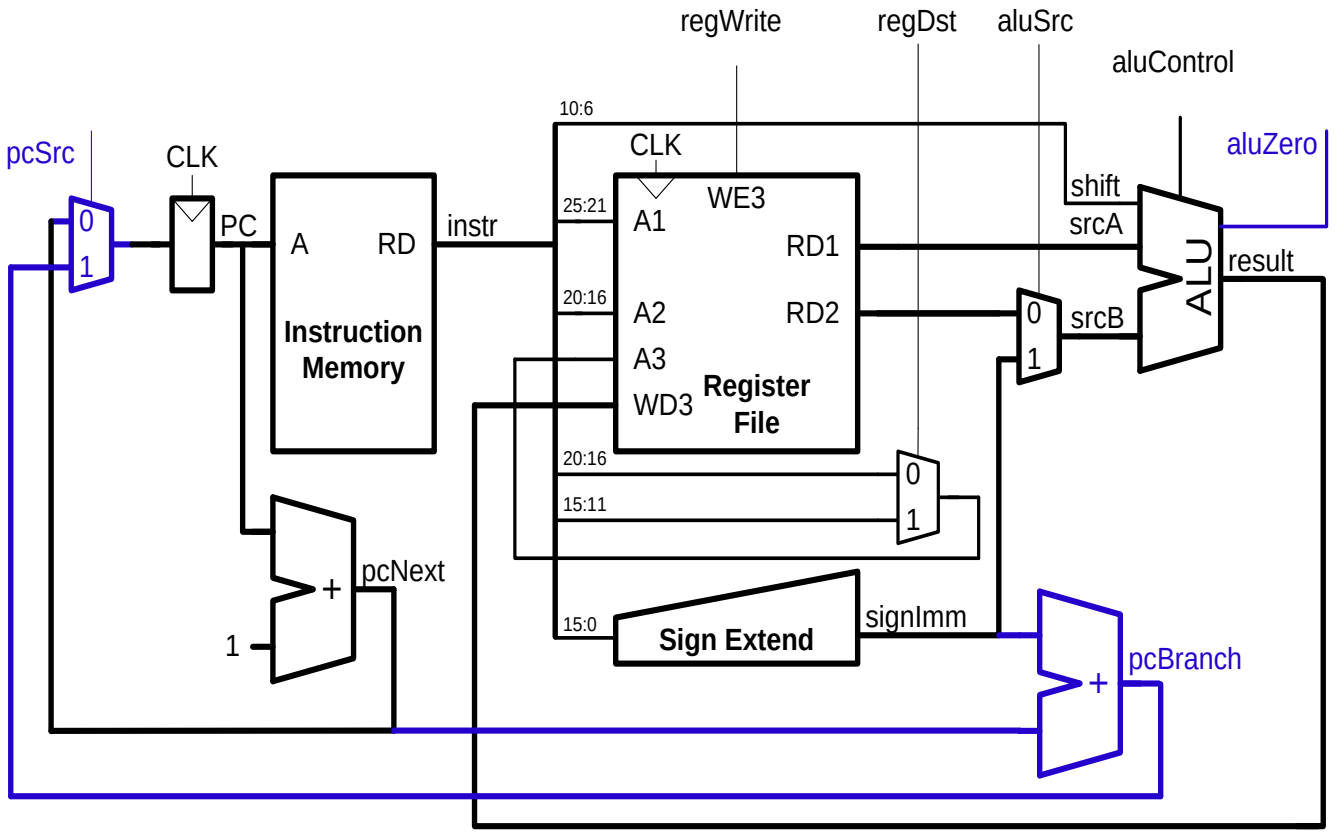


R-type. Shift Right Logical, rd = (uns)rt >> sa

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

schoolMIPS: **beq** instruction

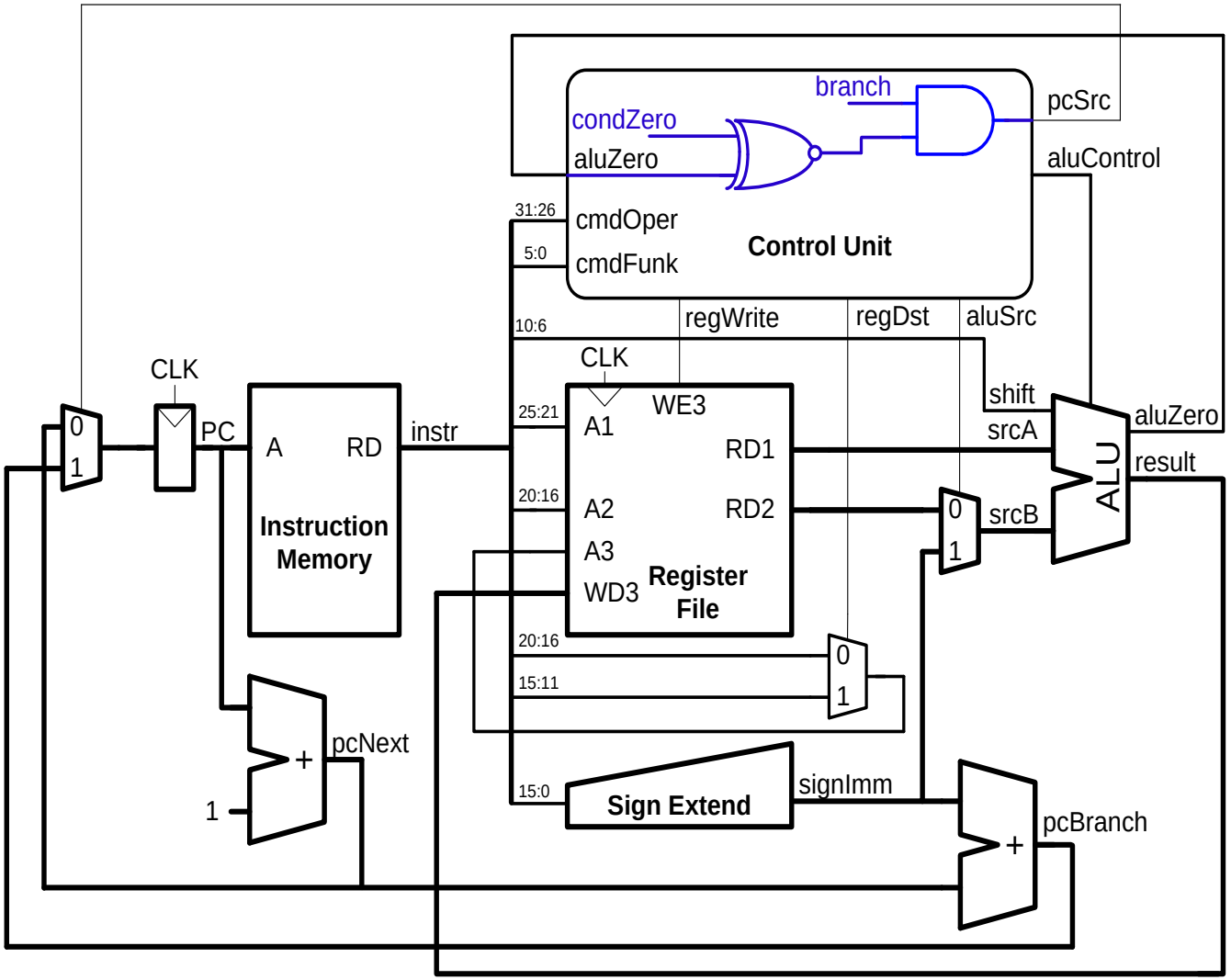
- determine address of next instruction



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

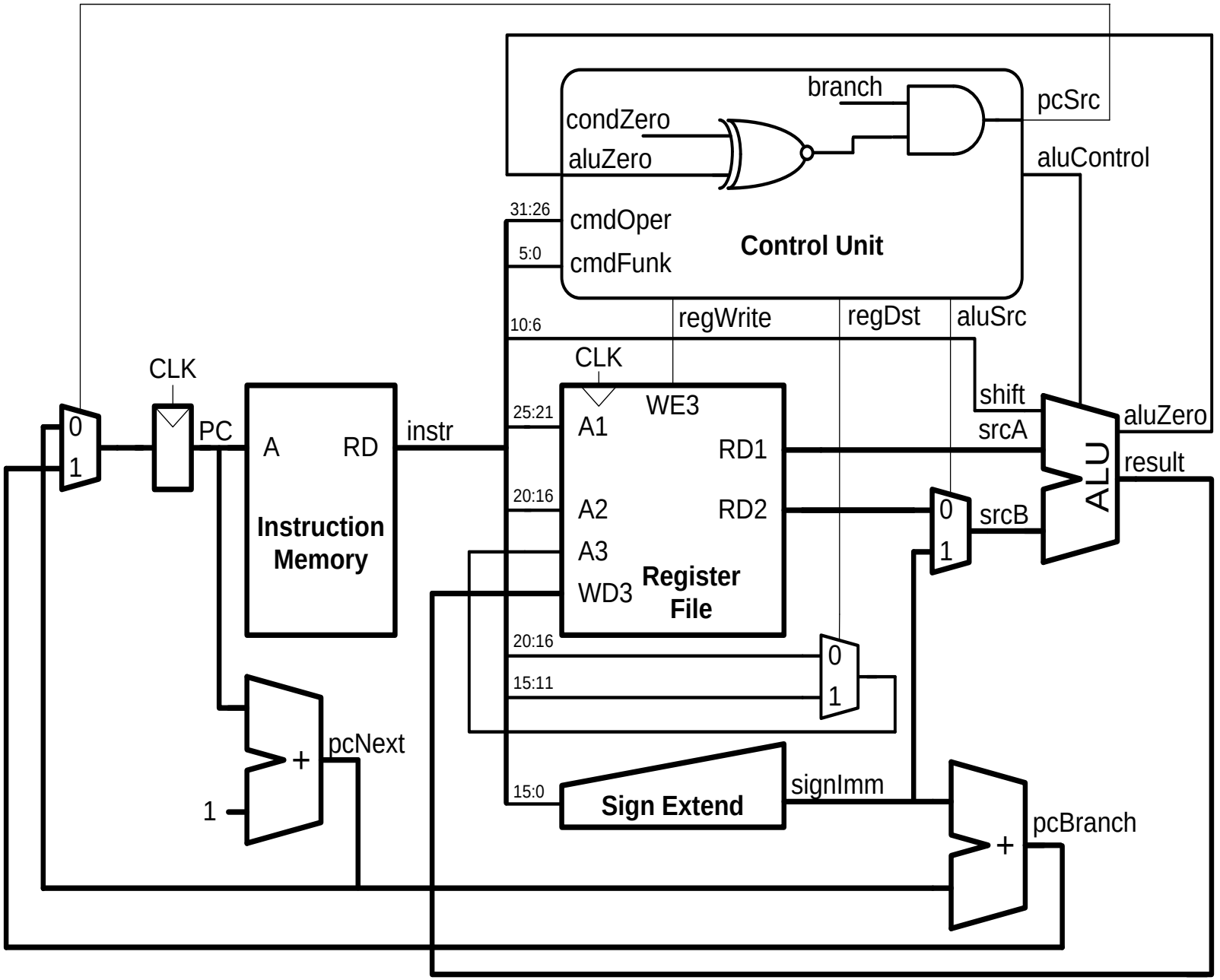
31 **op** 26	25 **rs** 21	20 **rt** 16	15 **Immediate** 0

schoolMIPS: **beq** instruction



- decision to branch or not depending on aluZero

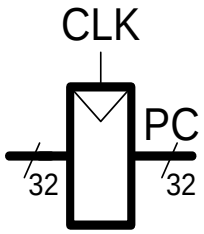
schoolMIPS CPU



schoolMIPS Modules

- Datapath
  - Program Counter (PC)
  - Instruction Memory
  - Register File
  - Arithmetic Logic Unit(ALU)
  - Sign Extending module (Sign Extend)
  - Adders for calculation the next instruction address (pcNext и pcBranch)
  - Multiplexors (pcSrc, regDst и aluSrc)
- Control

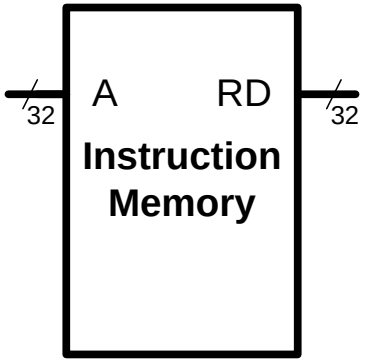
schoolMIPS Program Counter




```
// 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
```

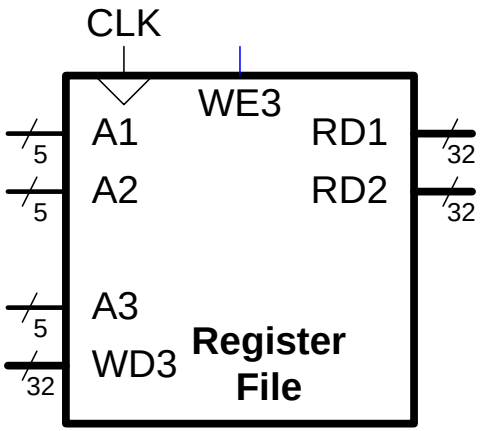
schoolMIPS Instruction Memory

```
// 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
```

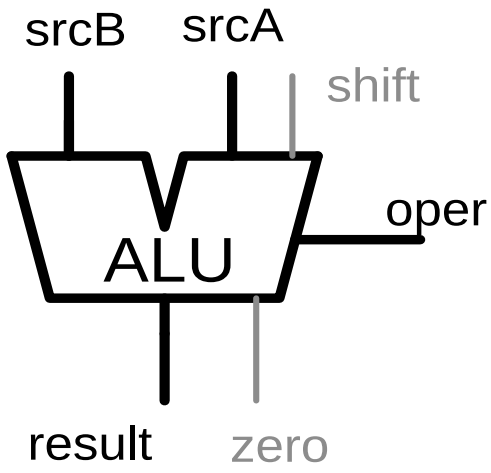


schoolMIPS Register File

```
// 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
```



schoolMIPS ALU

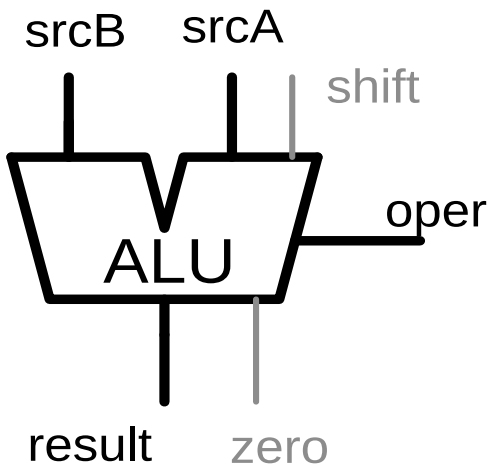


$oper_{2:0}$	Function	Description
000	ADD	A + B
001	OR	A | B
010	LUI	B << 16
011	SRL	B >> shift
100	SLTU	(A < B) ? 1 : 0
101	SUBU	A - B
110	reserved	
111	reserved	

```
// 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
```

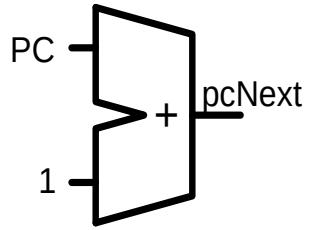
schoolMIPS ALU

```
// 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
```



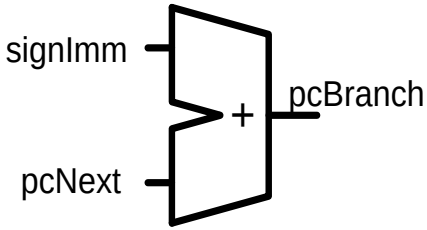
schoolMIPS Adders and Sign Extend

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



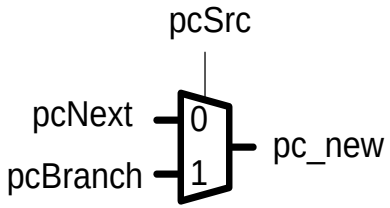

```
//sign extension   sm_cpu.v (line 64)
wire [31:0] signImm
    = { {16 { instr[15] }}, instr[15:0] };
```


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

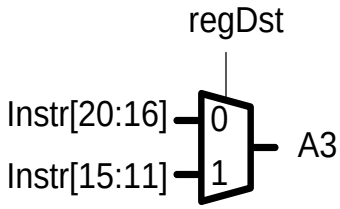


schoolMIPS Multiplexors

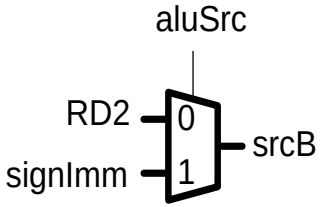
```
// next PC mux: branch or +1 (line 32)
wire [31:0] pc_new = ~pcSrc ? pcNext : pcBranch;
```




```
// 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;
```



schoolMIPS I-type Instructions

```
//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

schoolMIPS R-type Instructions

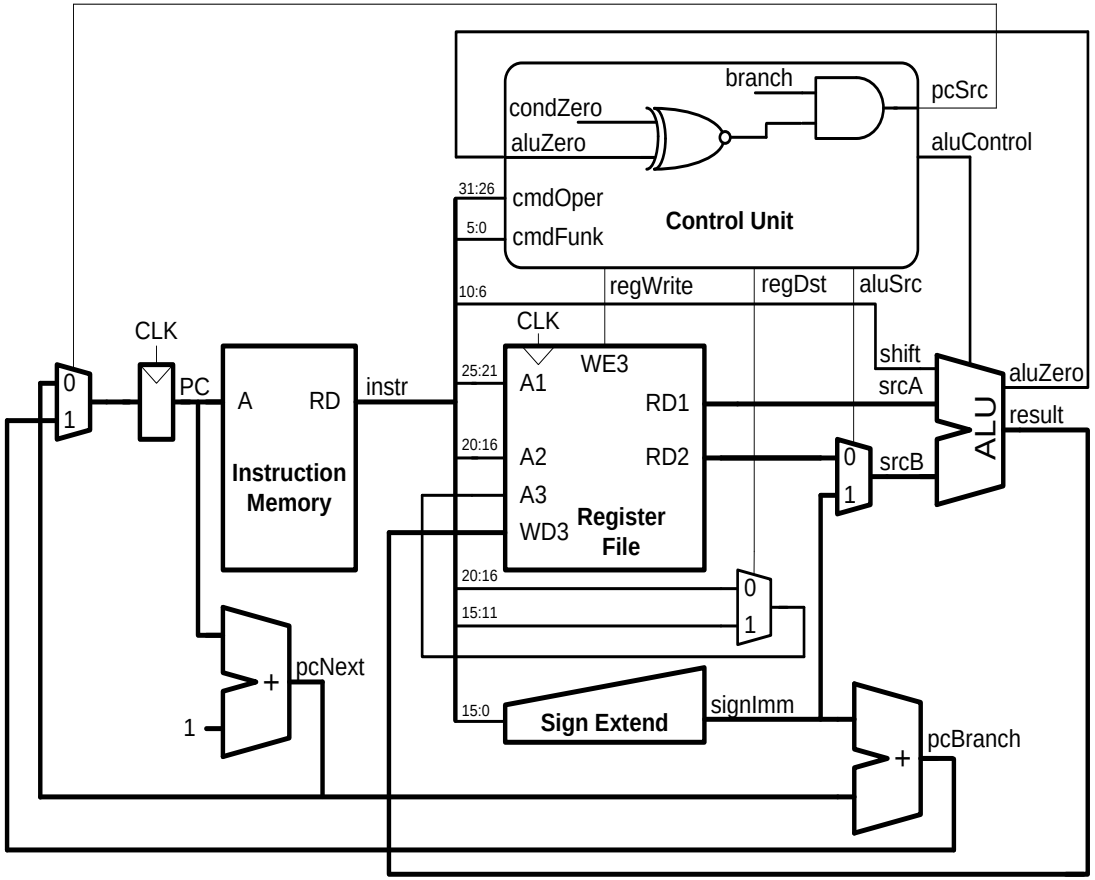
```
//instruction operation code    sm_cpu.vh (line 19-41)
`define C_SPEC 6'b000000 // Special instructions
                         // (depends on function field)

//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

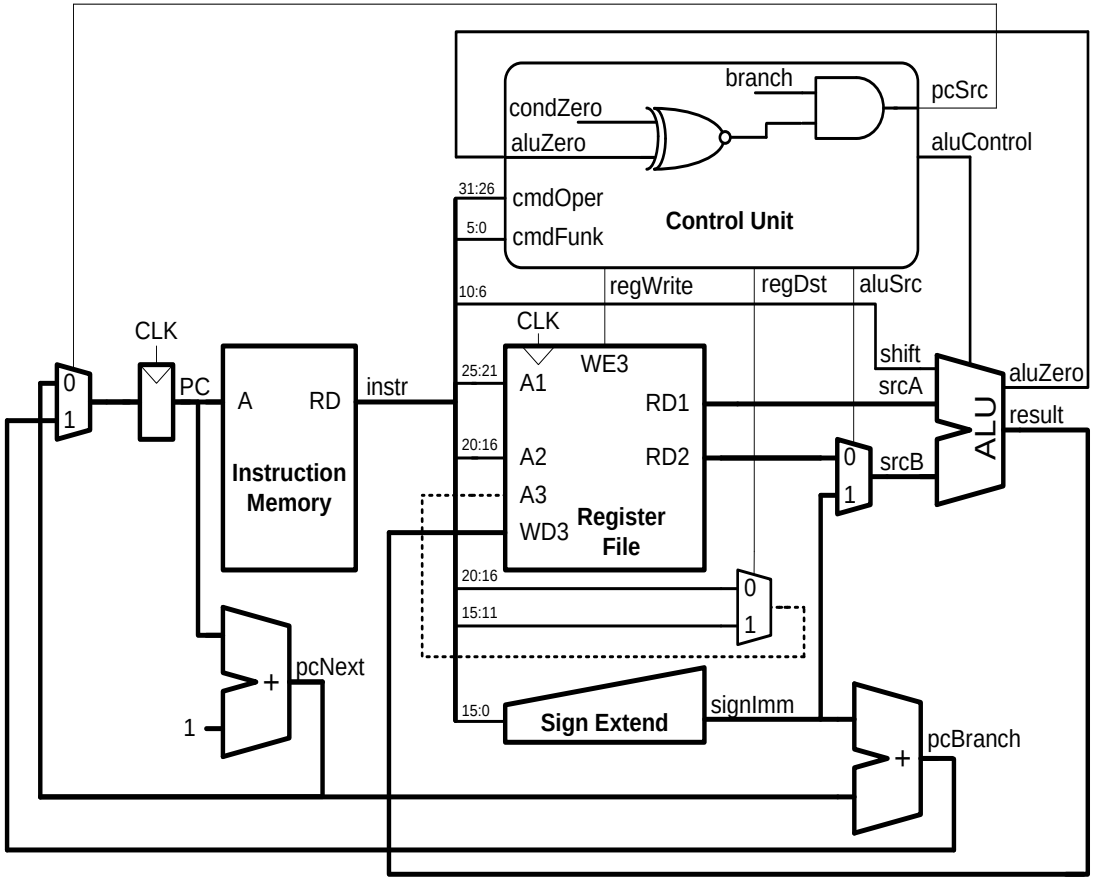
schoolMIPS Control Signals (1)

Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl



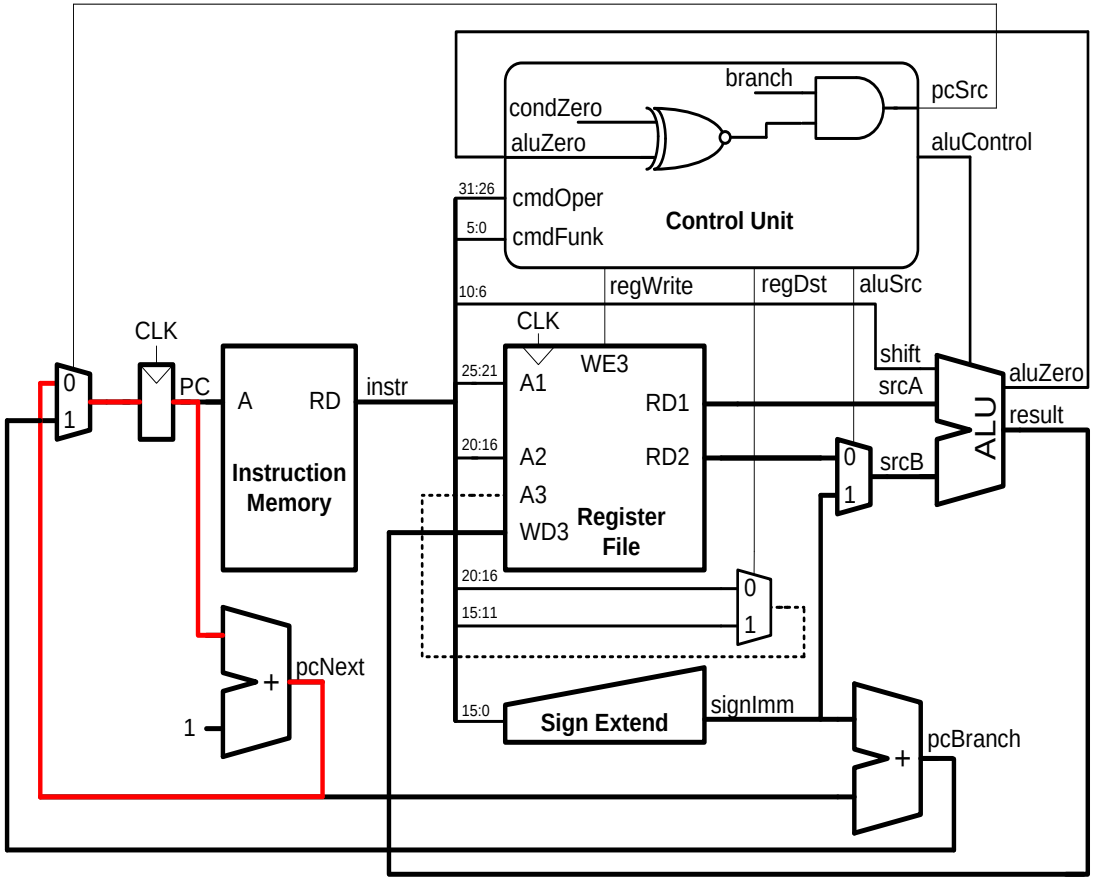
schoolMIPS Control Signals (2)

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



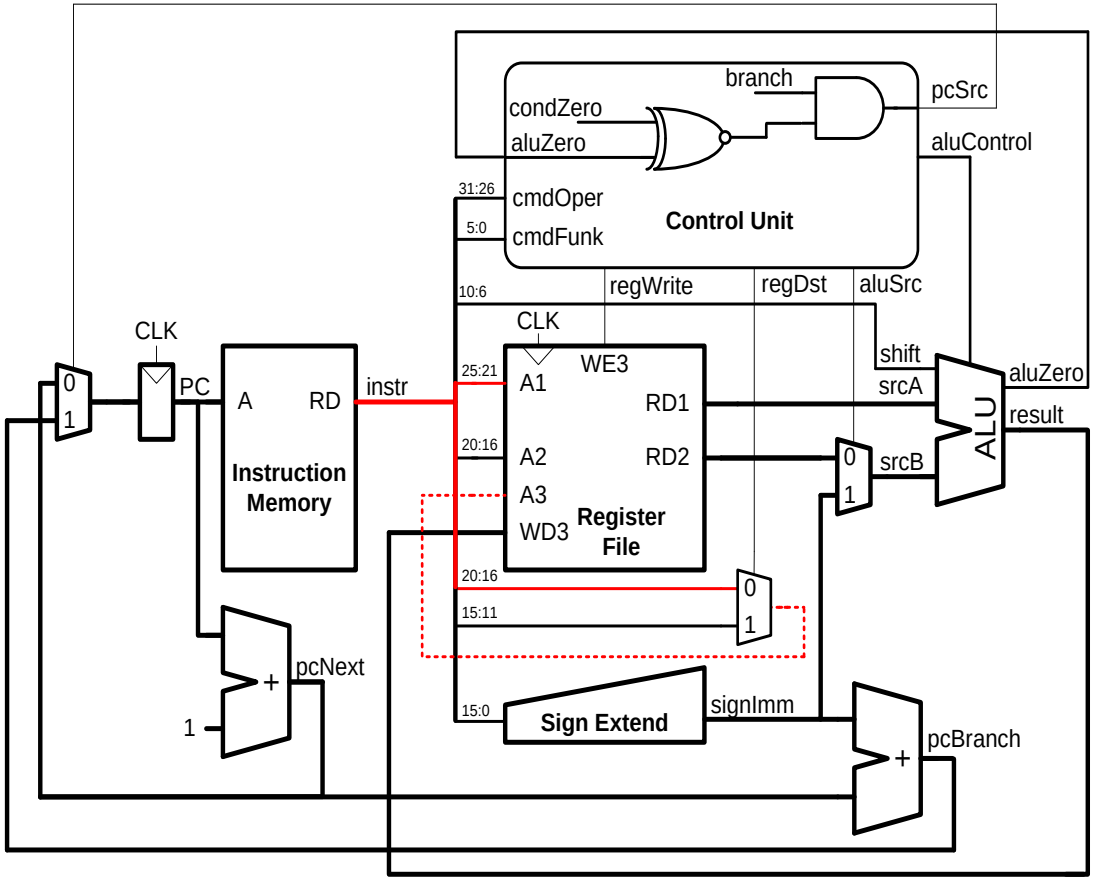
schoolMIPS Control Signals (3)

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



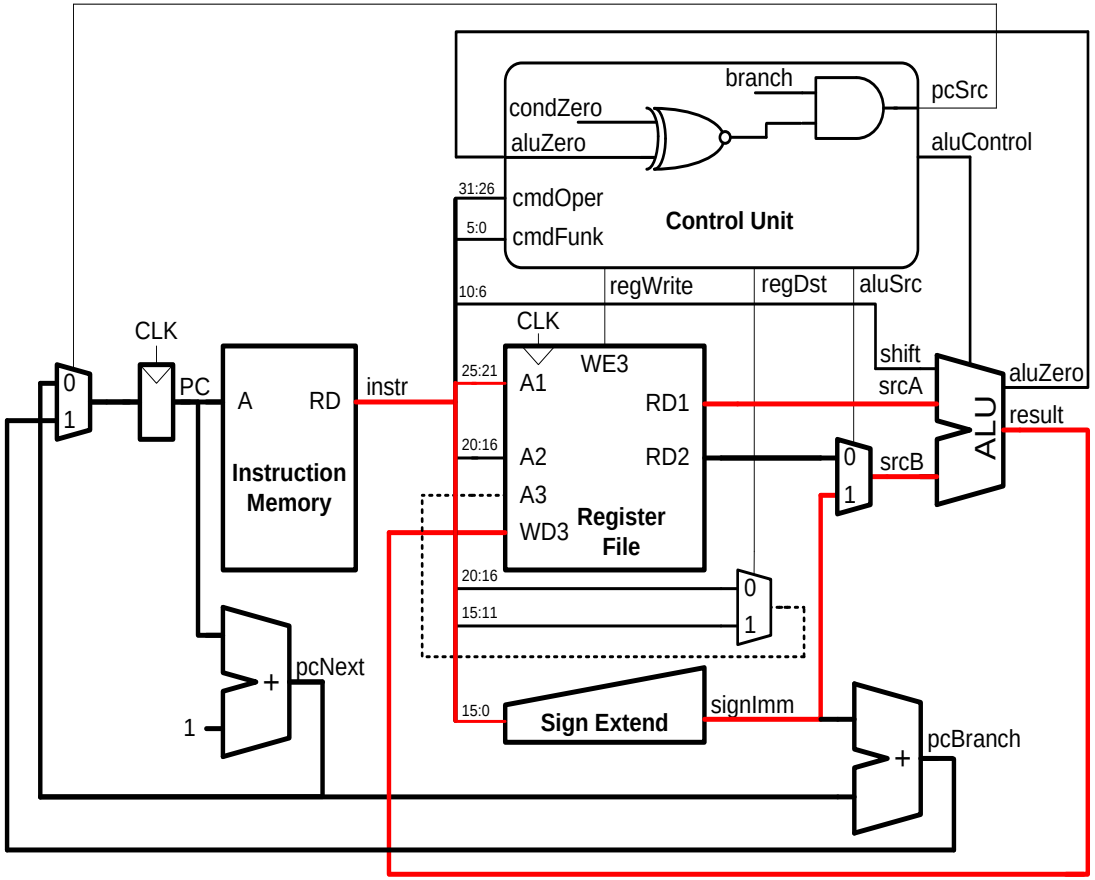
schoolMIPS Control Signals (4)

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



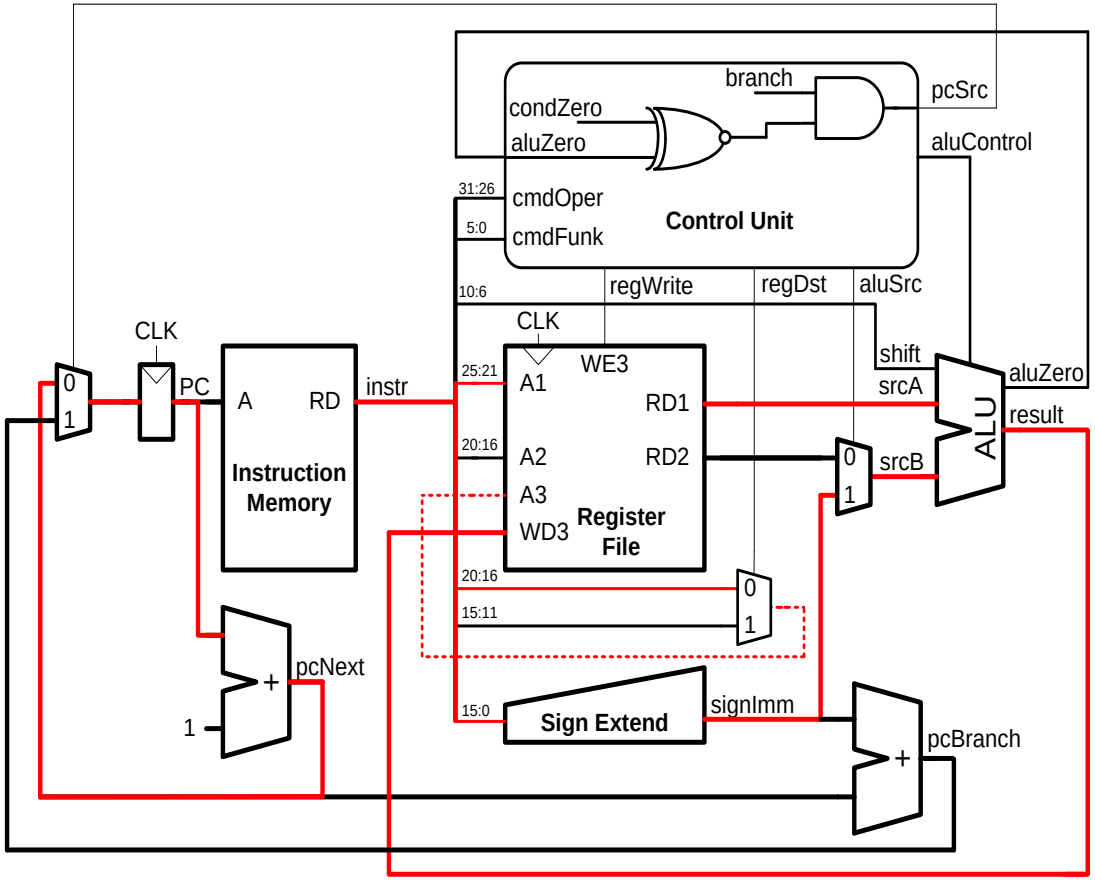
schoolMIPS Control Signals (5)

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



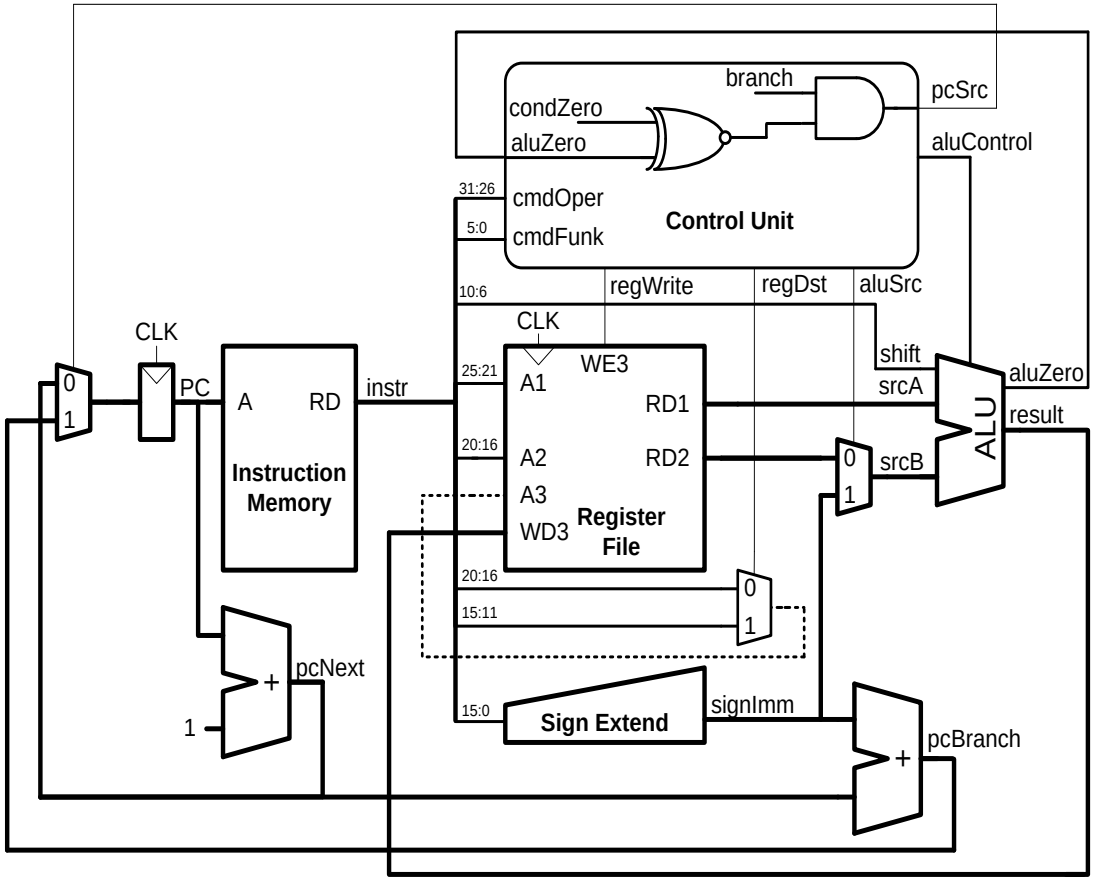
schoolMIPS Control Signals (6)

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



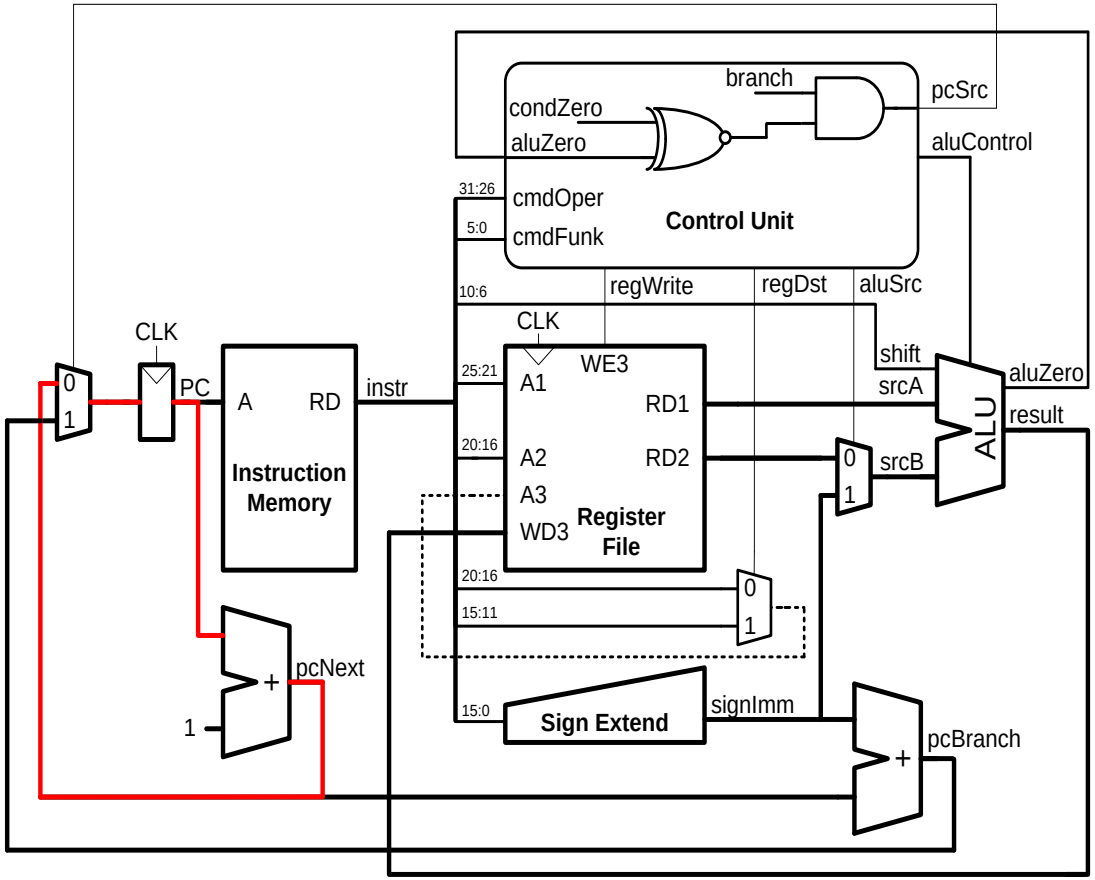
schoolMIPS Control Signals (7)

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



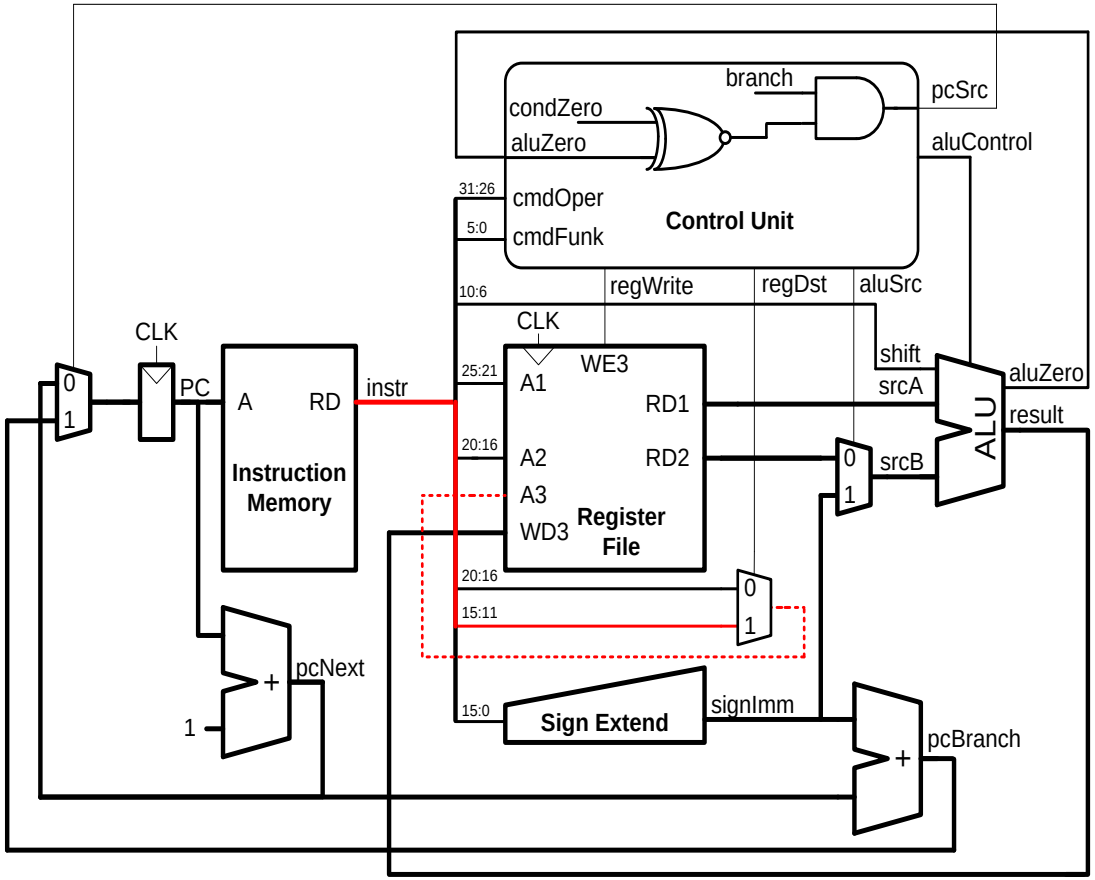
schoolMIPS Control Signals (8)

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



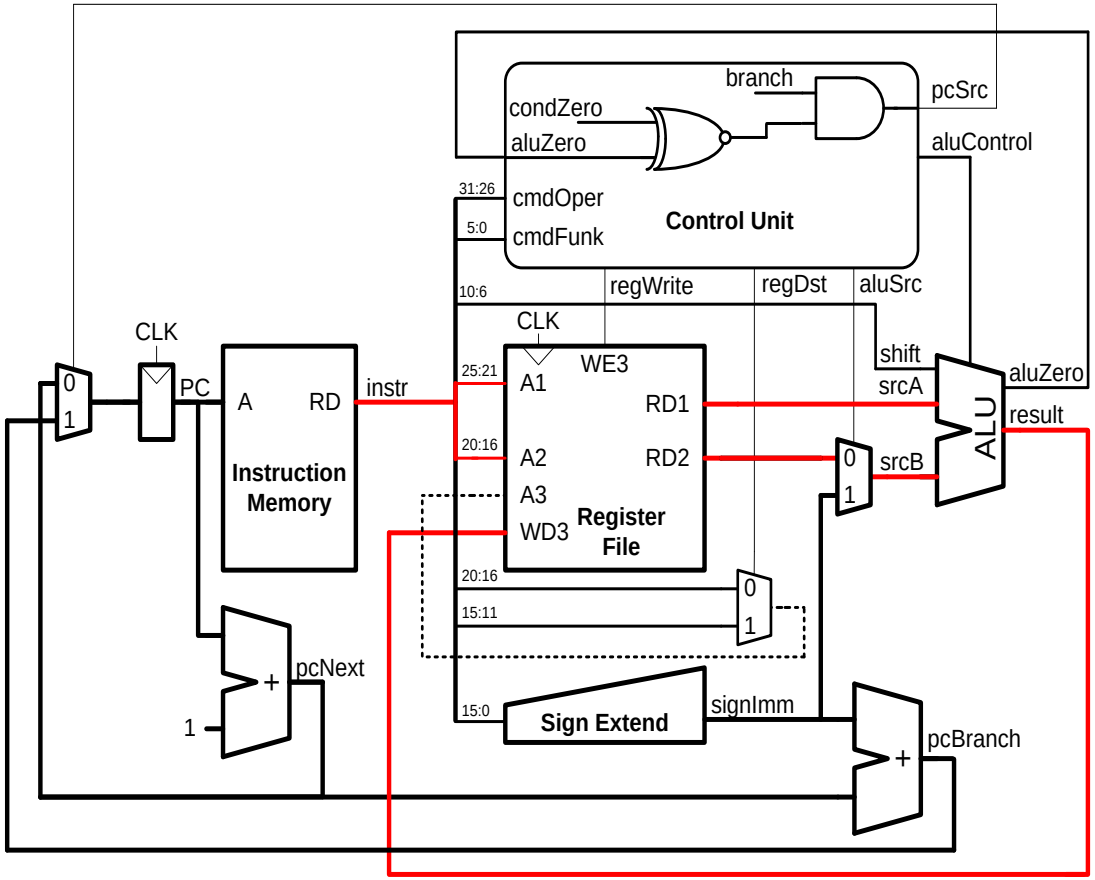
schoolMIPS Control Signals (9)

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



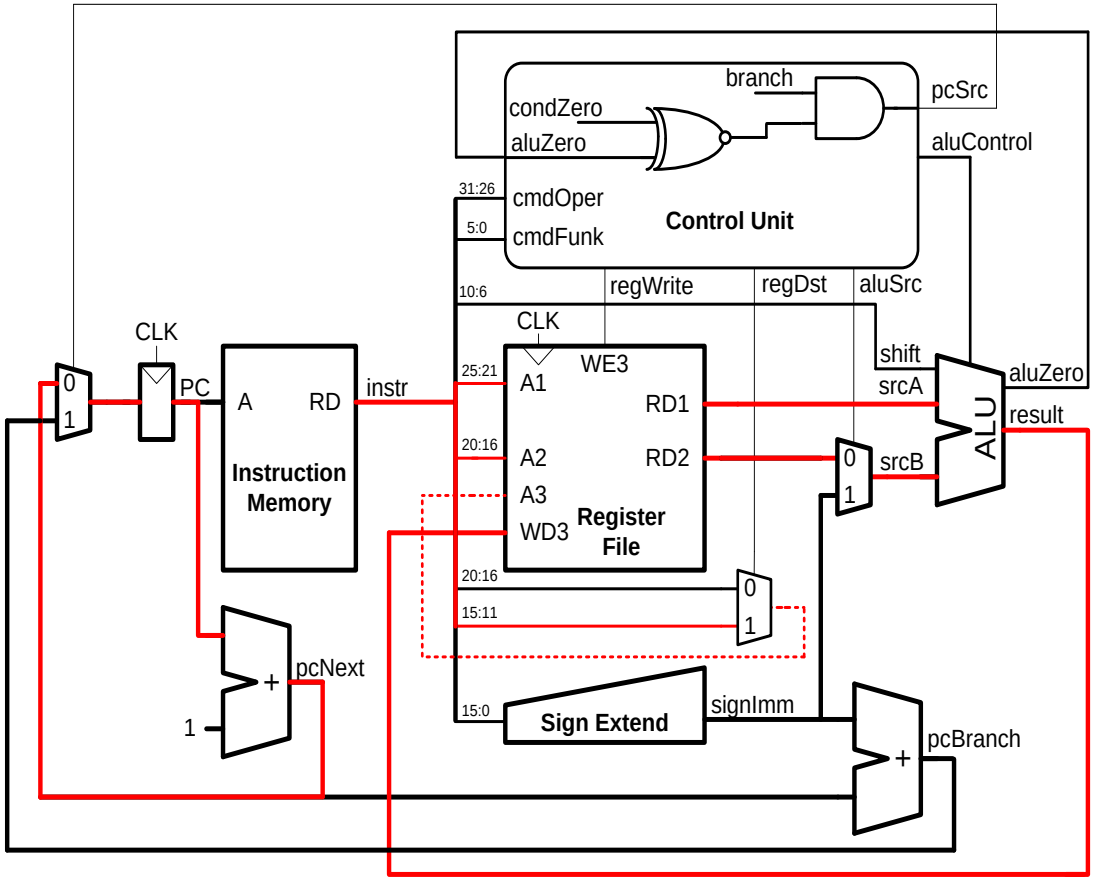
schoolMIPS Control Signals (10)

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



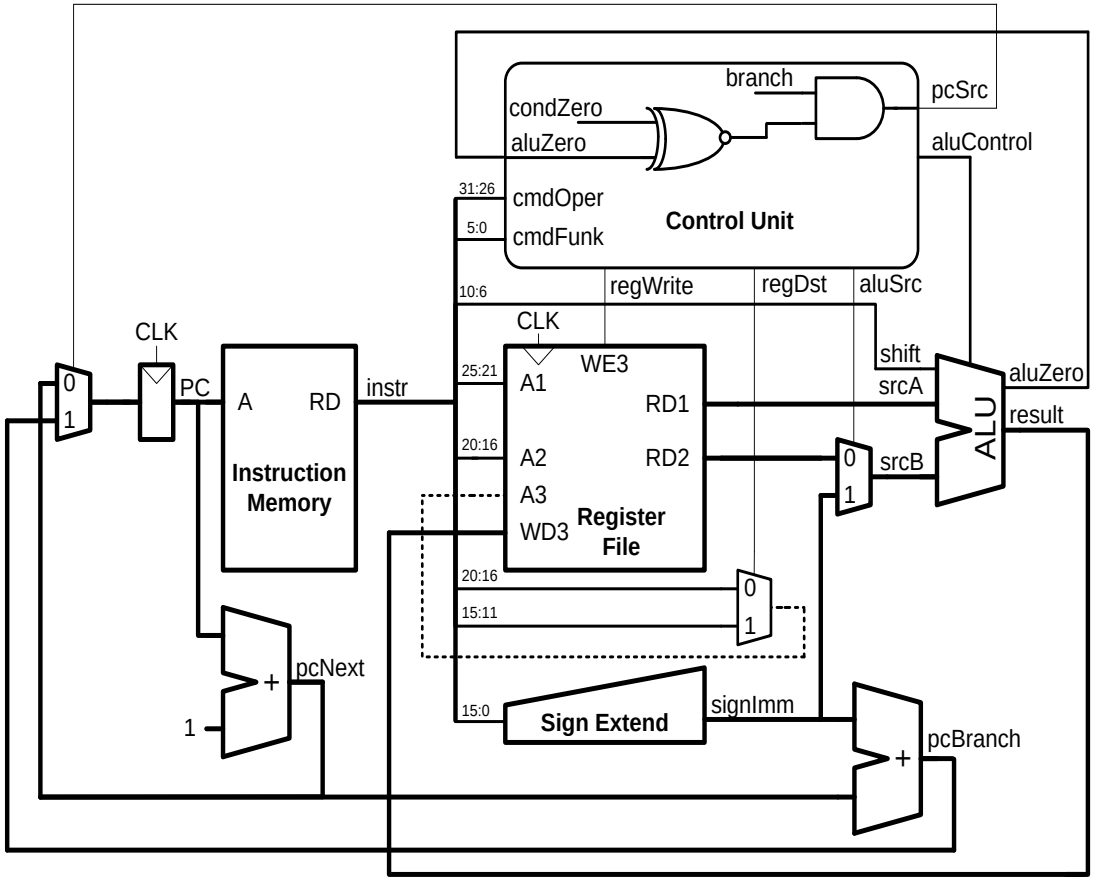
schoolMIPS Control Signals (11)

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



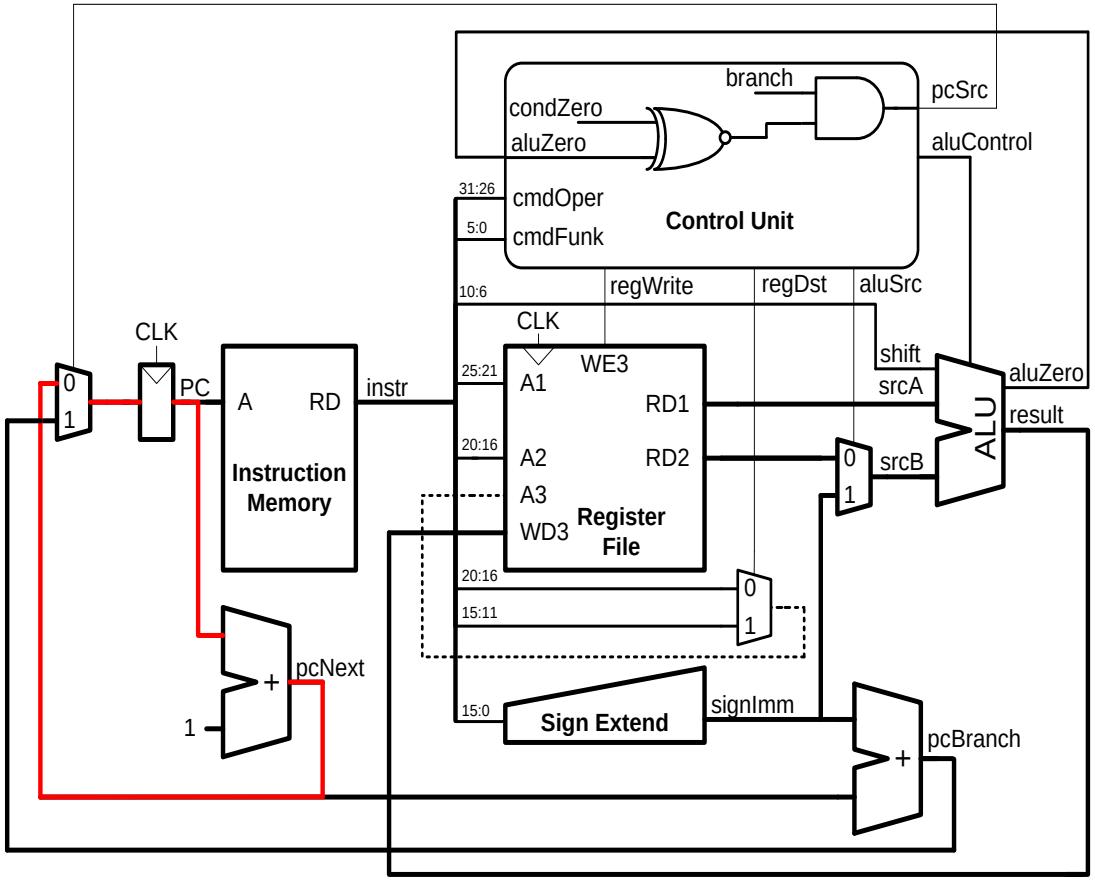
schoolMIPS Control Signals (12)

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



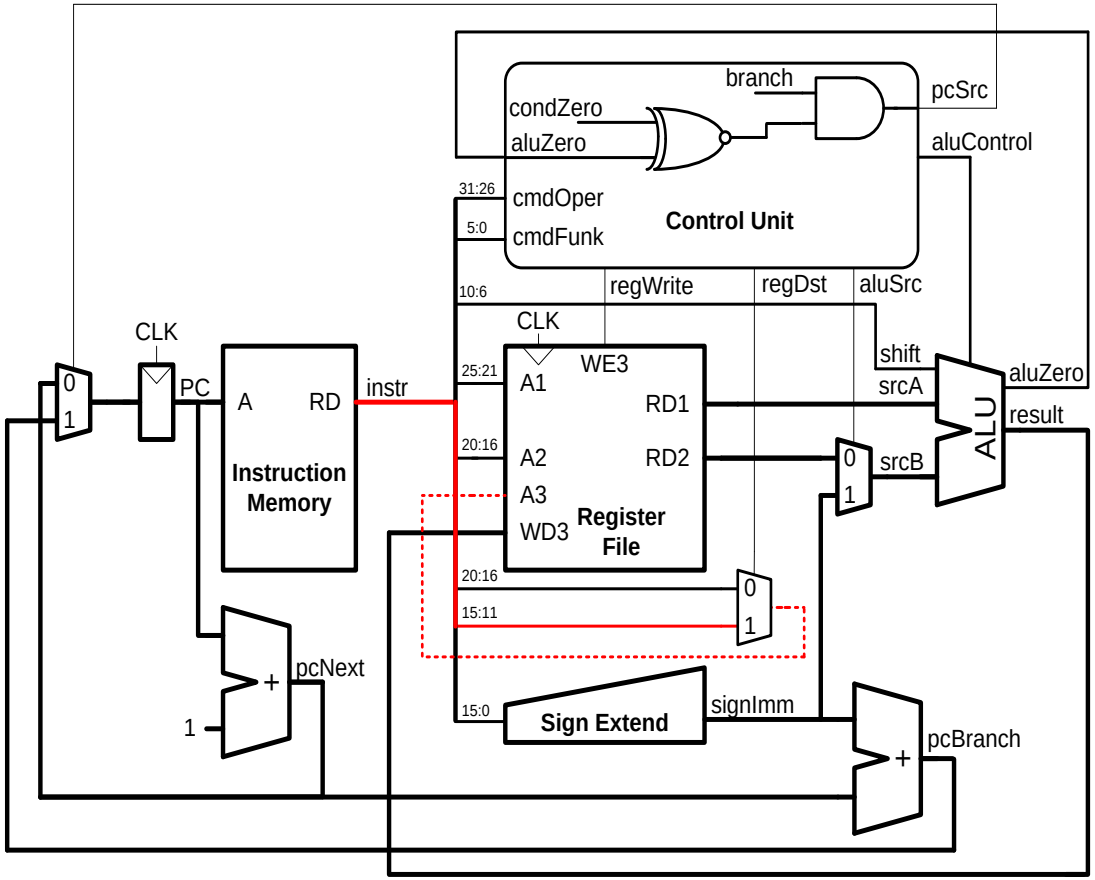
schoolMIPS Control Signals (13)

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



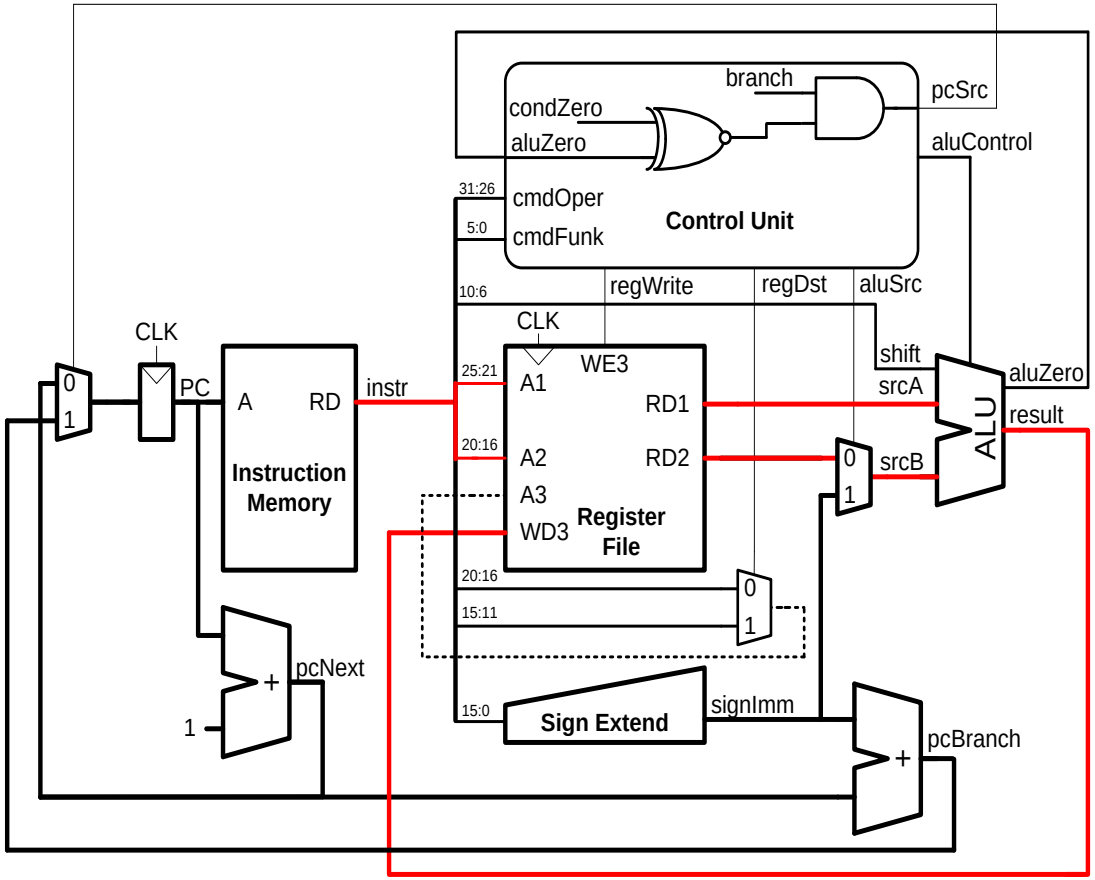
schoolMIPS Control Signals (14)

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



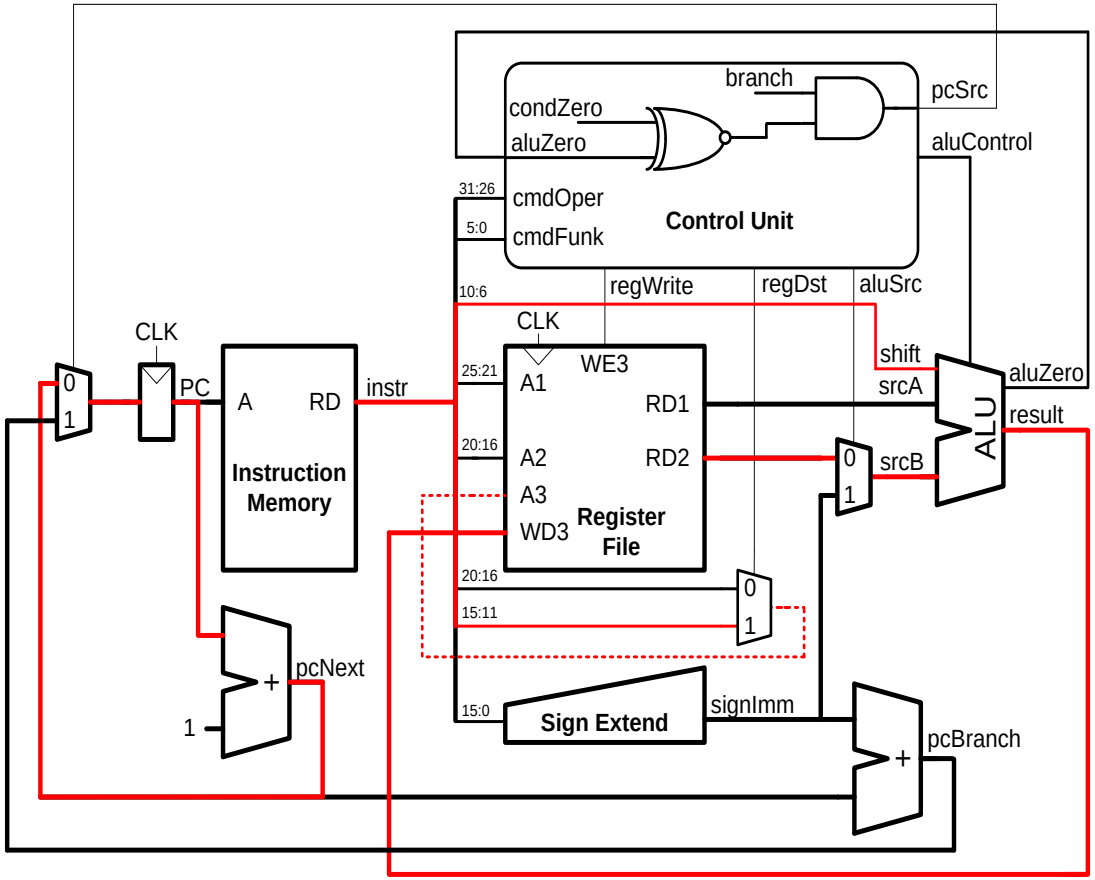
schoolMIPS Control Signals (15)

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



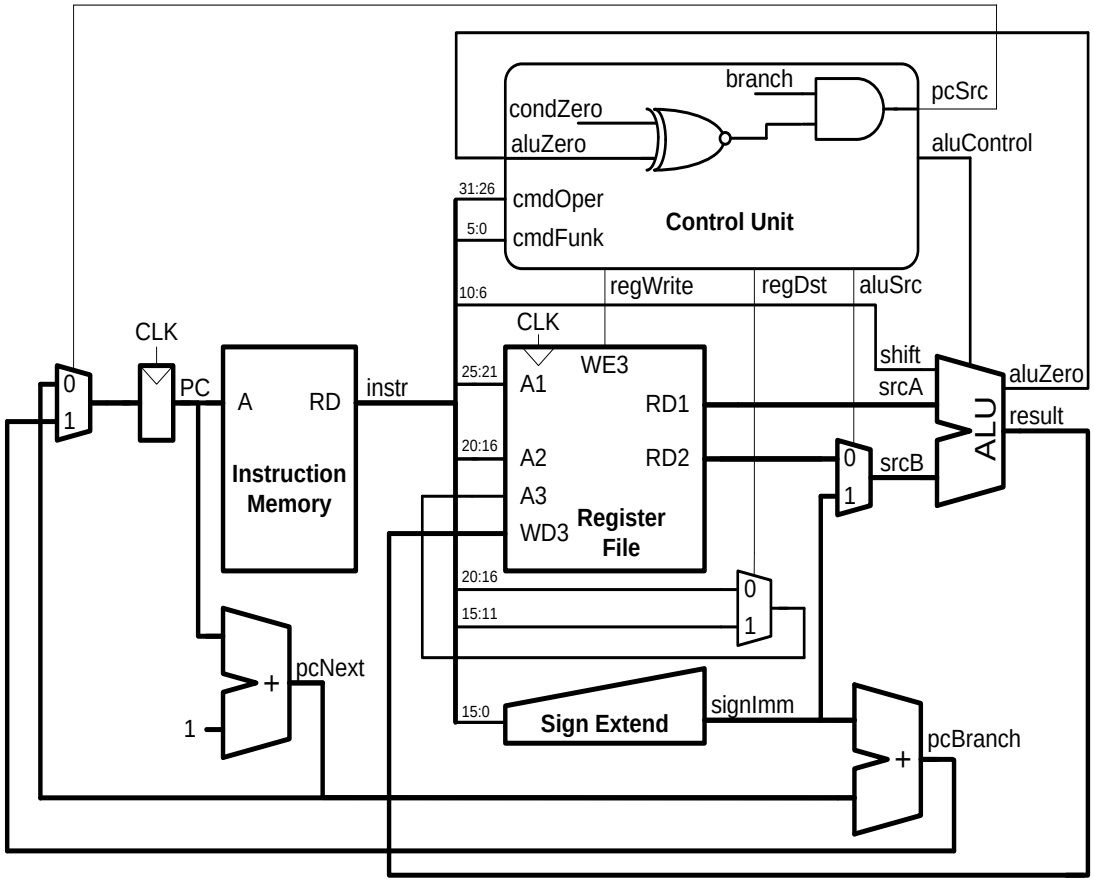
schoolMIPS Control Signals (16)

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



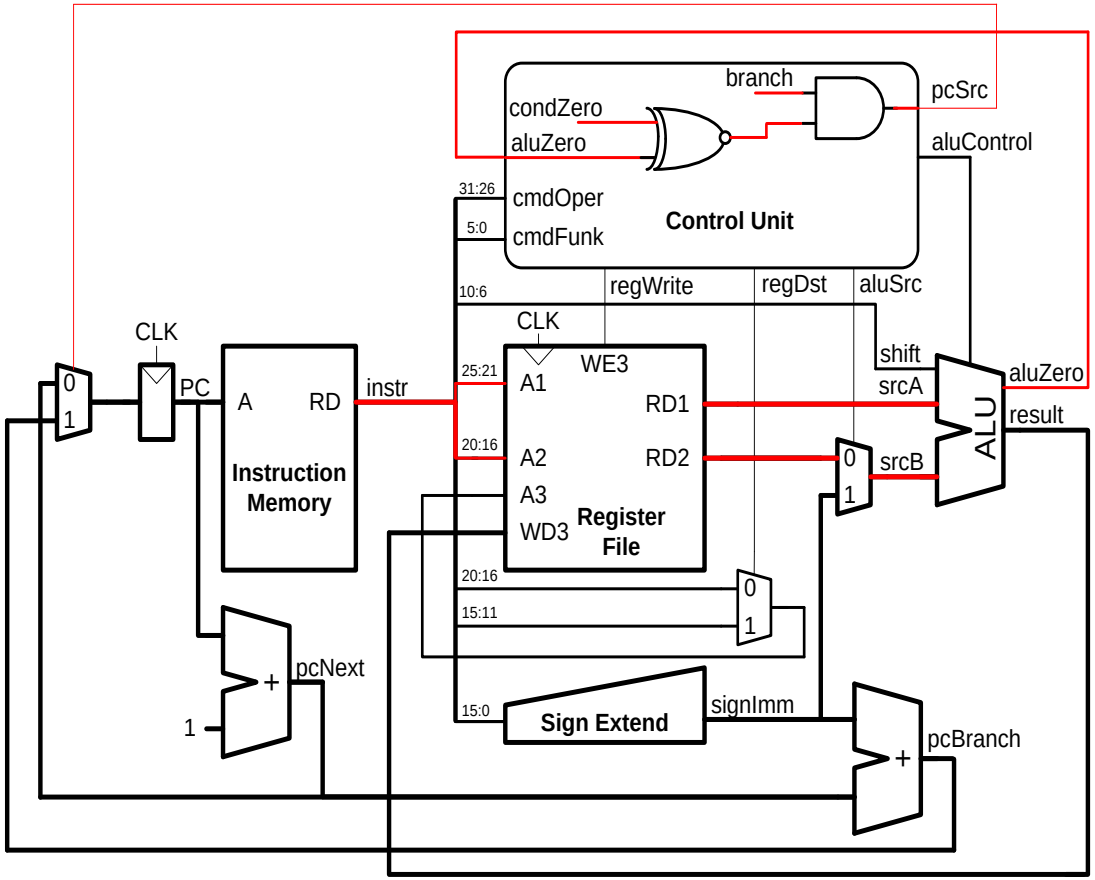
schoolMIPS Control Signals (17)

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



schoolMIPS Control Signals (18)

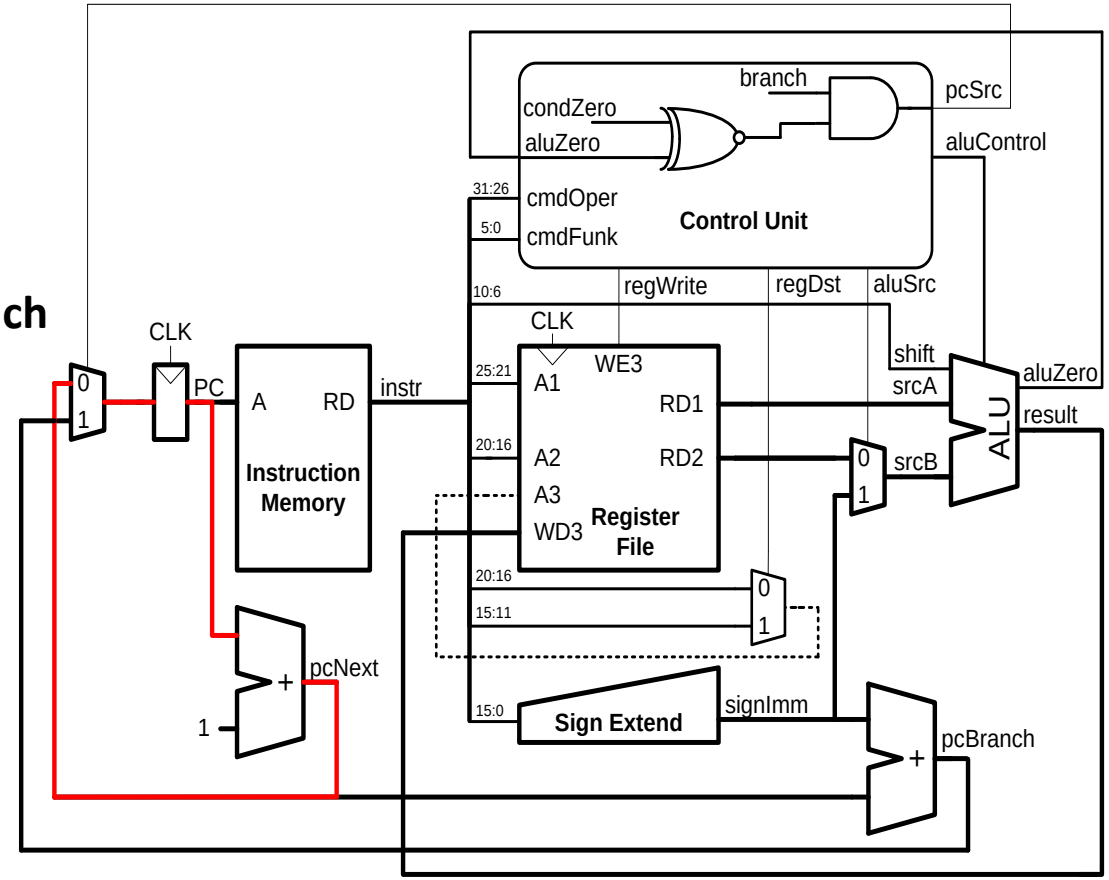
Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011
beq	000100	??????	1	1	0	0	0	101



schoolMIPS Control Signals (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

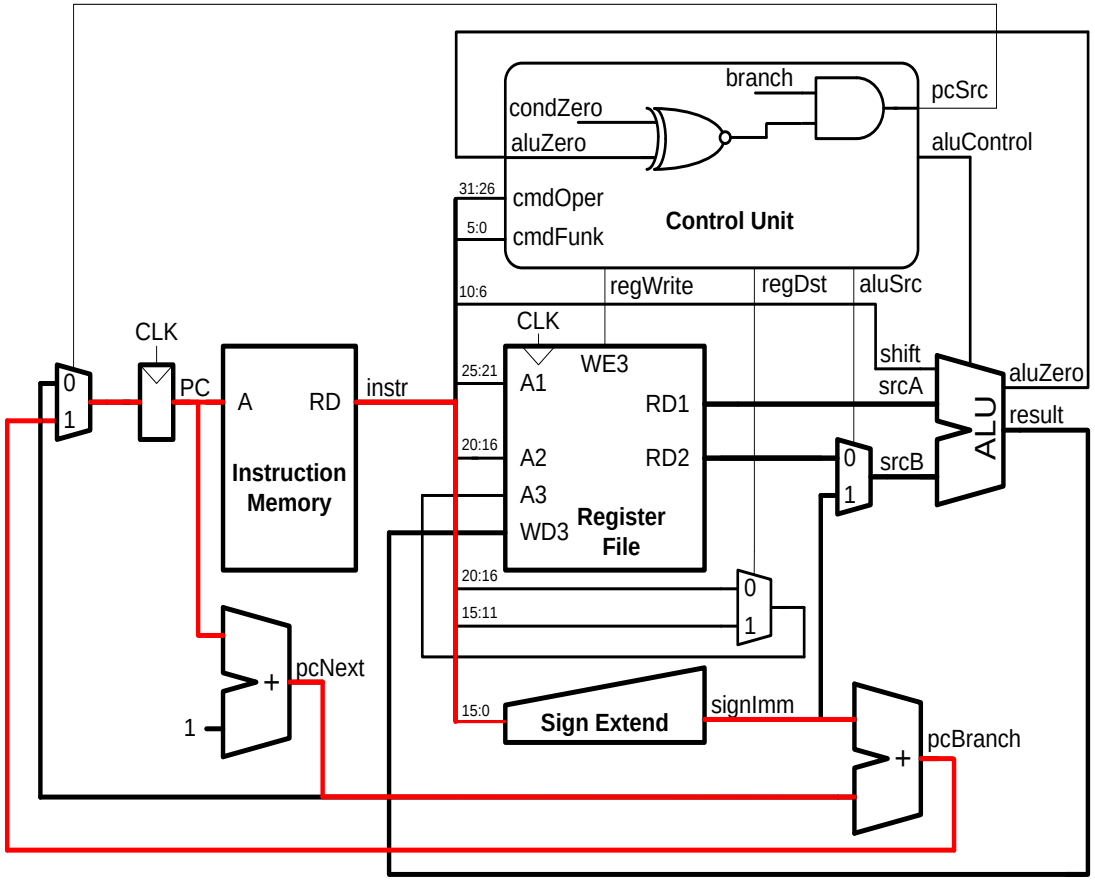
Rs != Rd => no branch



schoolMIPS Control Signals (20)

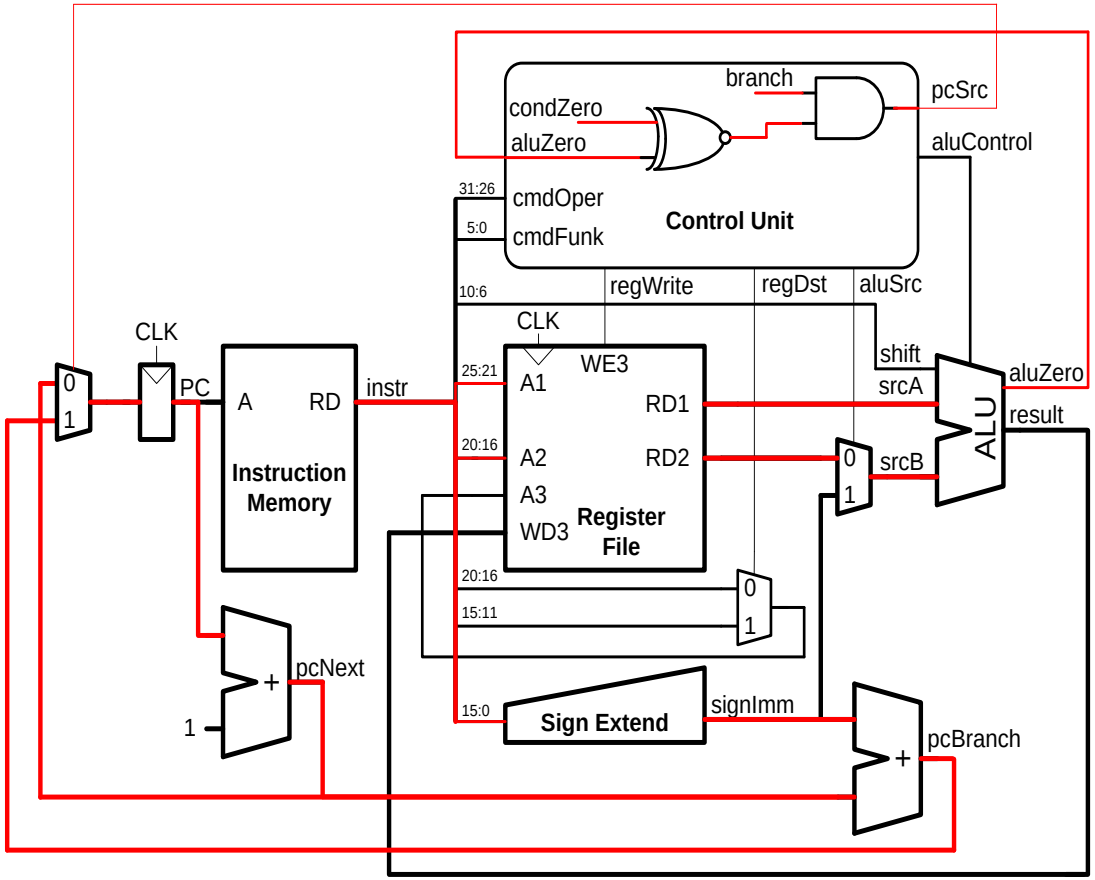
Instr	cmdOper	cmdFunc	branch	condZero	regDst	regWrite	aluSrc	aluControl
addiu	001001	??????	0	0	0	1	1	000
addu	000000	100001	0	0	1	1	0	000
srl	000000	000010	0	0	1	1	0	011
beq	000100	??????	1	1	0	0	0	101

Rs == Rd => branch



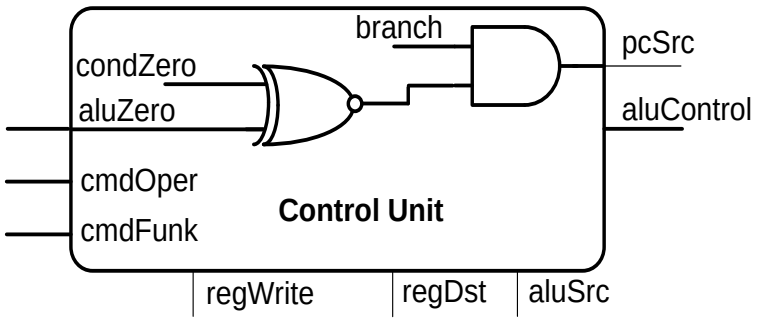
schoolMIPS Control Signals (21)

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



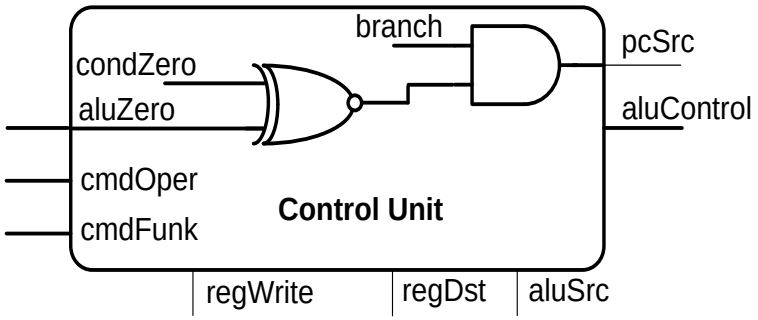
schoolMIPS Control
Branch Signals

```
// 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
```

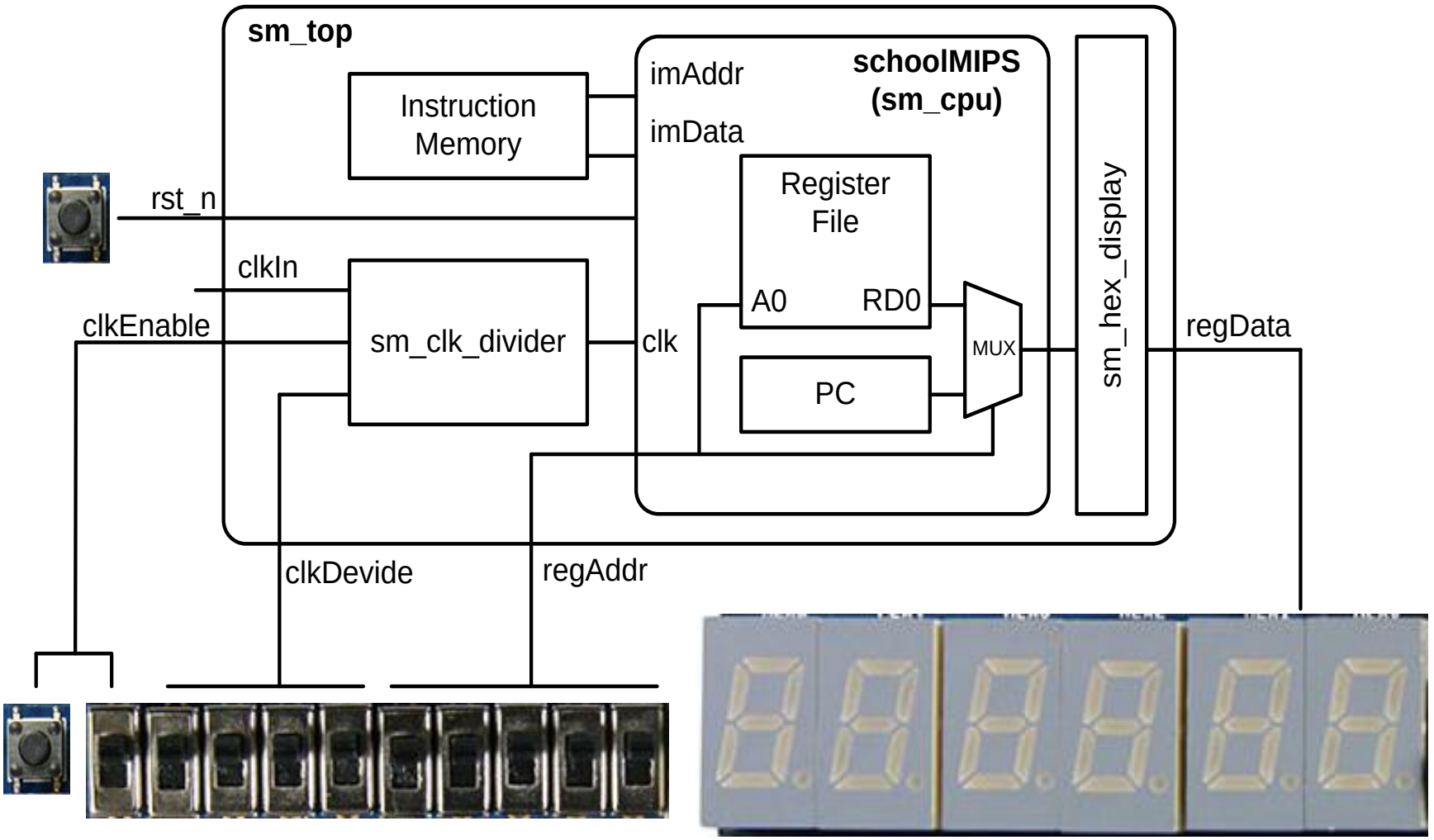


schoolMIPS Control
Control Signals

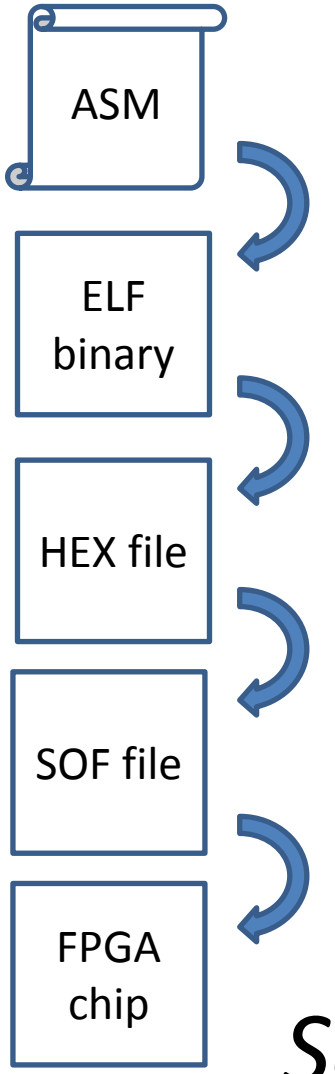
```
// control unit (line 95-134)
module sm_control
...
    always @ (*) begin
        //control signals default values
        branch = 1'b0;
        condZero = 1'b0;
        regDst = 1'b0;
        regWrite = 1'b0;
        aluSrc = 1'b0;
        aluControl = `ALU_ADD;

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

           ...
         endcase
    end
endmodule
```



schoolMIPS and FPGA debug board



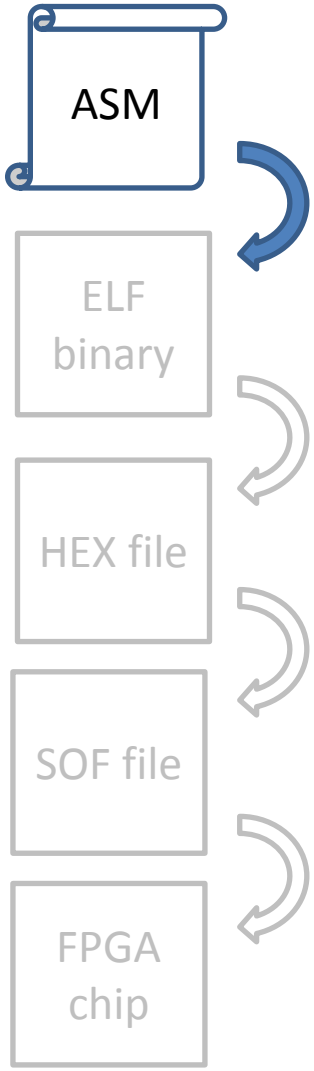
schoolMIPS Programming



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

See details in User Manual

MIPS Assembler Program




```
# 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)

Binary Executable File


```
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  ................
```

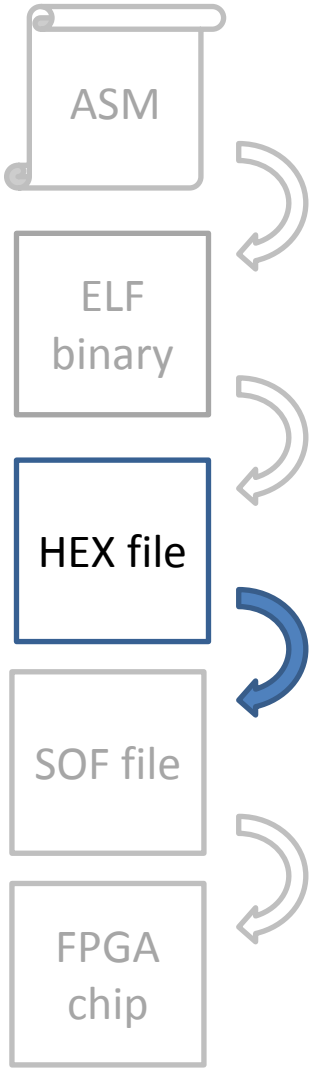
BIN2HEX converter

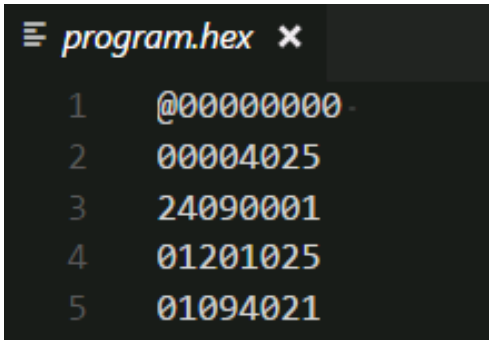
MIPS Assembler Program


```
# 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
```

MARS (MIPS Assembler and Runtime Simulator)

HEX-file



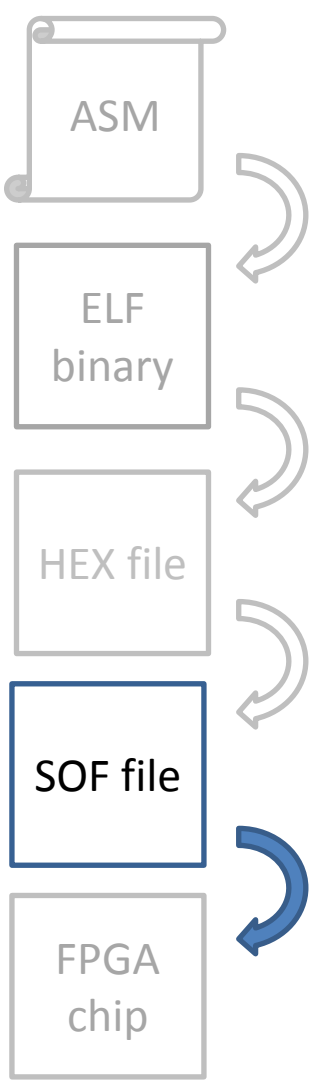


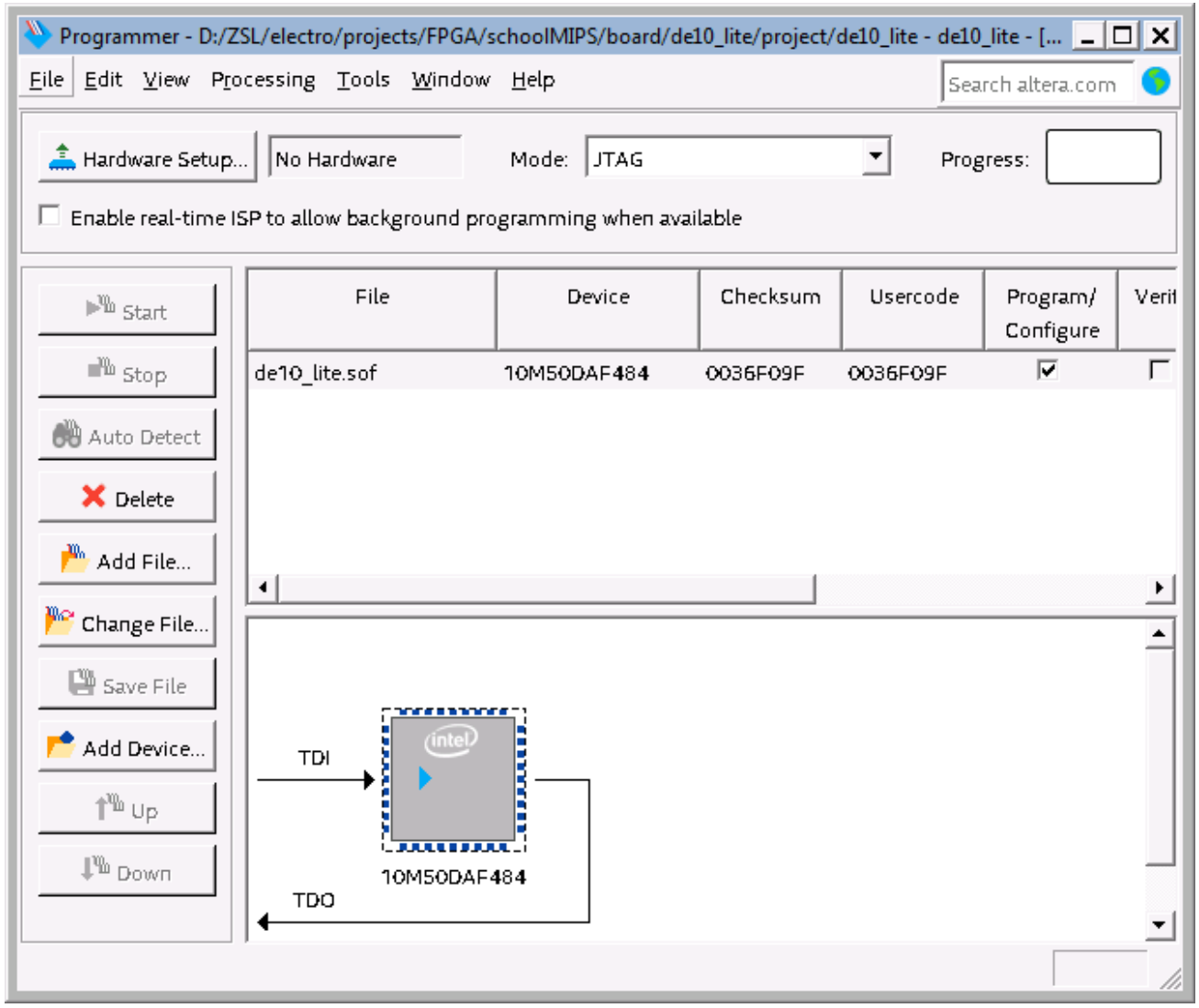
sm_rom.v
$readmemh (…);

***.v, *.vh**
Verilog Sources

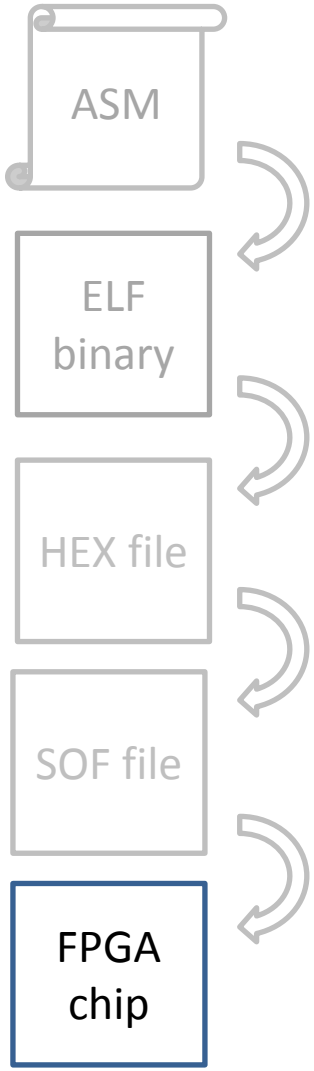
synthesis tool (Quartus Prime)

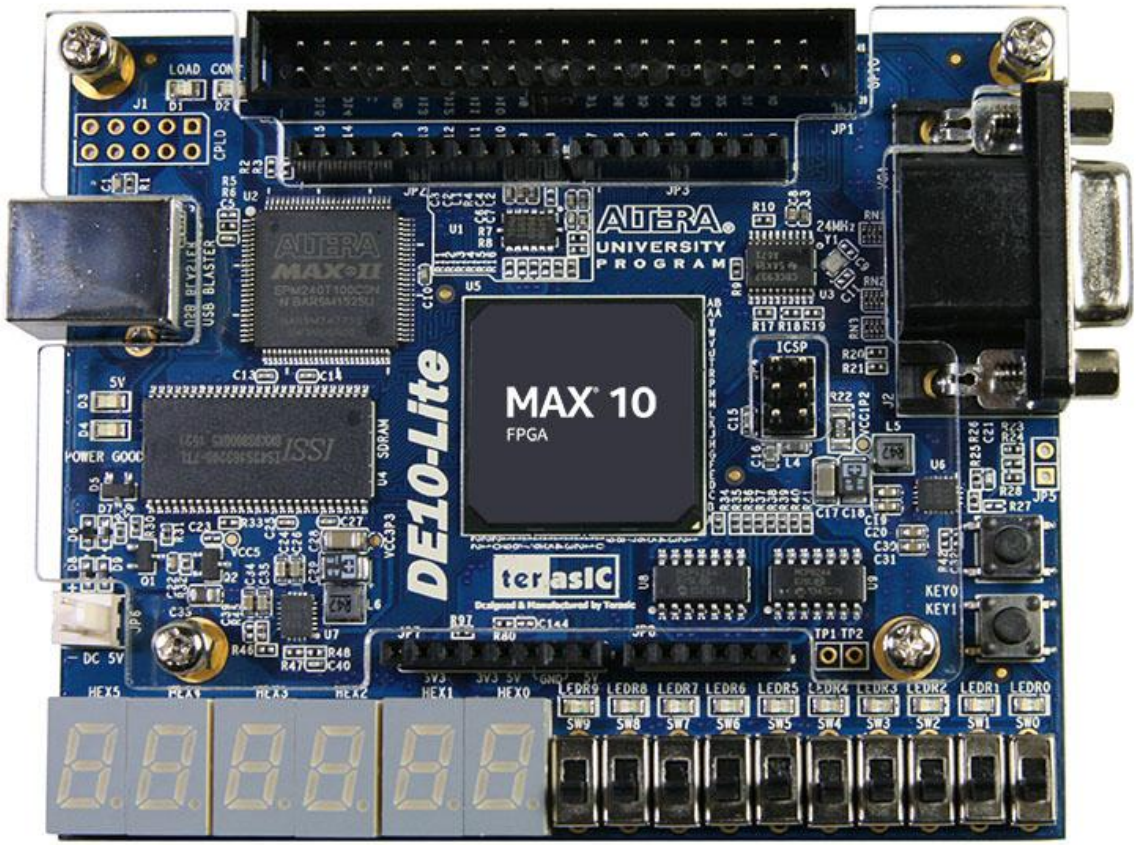
FPGA Configuration File





Configured FPGA chip on the Debug Board





What is Next?

- The great book “Digital Design and Computer Architecture” by David Harris and Sarah Harris

  The free translation (Russian) can be downloaded from MIPS Academic Community (link)

- MIPSfpga – provides the RTL source code of the MIPS microAptiv UP core for implementation on an FPGA (including SoC and Labs). This core is a member of the same microcontroller family found in many embedded devices, including the popular PIC32MZ & PIC32MK microcontrollers from Microchip and the Artik 1 from Samsung (link)

Thank you!
Your questions?