Университет ИТМО

Лабораторная работа №3 «Расширение возможностей учебного процессорного ядра schoolRISCV»

по дисциплине: Функциональная схемотехника
Вариант: LRU

Выполнили:
Кузнецов Даниил
Баянов Равиль
Группа:
P3334
Преподаватель:
Кустарев Павел Валерьевич
Васильев Сергей Михайлович

Санкт-Петербург

2025

Содержание

Задание

В лабораторной работе вам предлагается разобраться во внутреннем устройстве простейшего процессорного ядра архитектуры RISC-V. Результатом изучения микроархитектуры процессорного ядра и системы команд RISC-V станут ваши функциональные и нефункциональные модификации ядра.

Основное задание:

1. Модифицировать процессорное ядро, в соответствии с вашим вариантом;
2. Подготовить тестовое окружение системного уровня и убедиться в корректности вашей реализации путём запуска симуляционных тестов.
3.

Примечание:

При непосредственном описании ваших модификаций в коде проекта запрещено использовать не синтезируемые конструкции и арифметические операции, отличныеот сложения и вычитания (то есть, умножение, деление и возведение в степень реализуйте сами посредством описания любого, понравившегося вам, алгоритма). Однако, в тестовом окружении использовать не синтезируемые конструкции и всевозможные арифметические операции можно (и даже нужно).

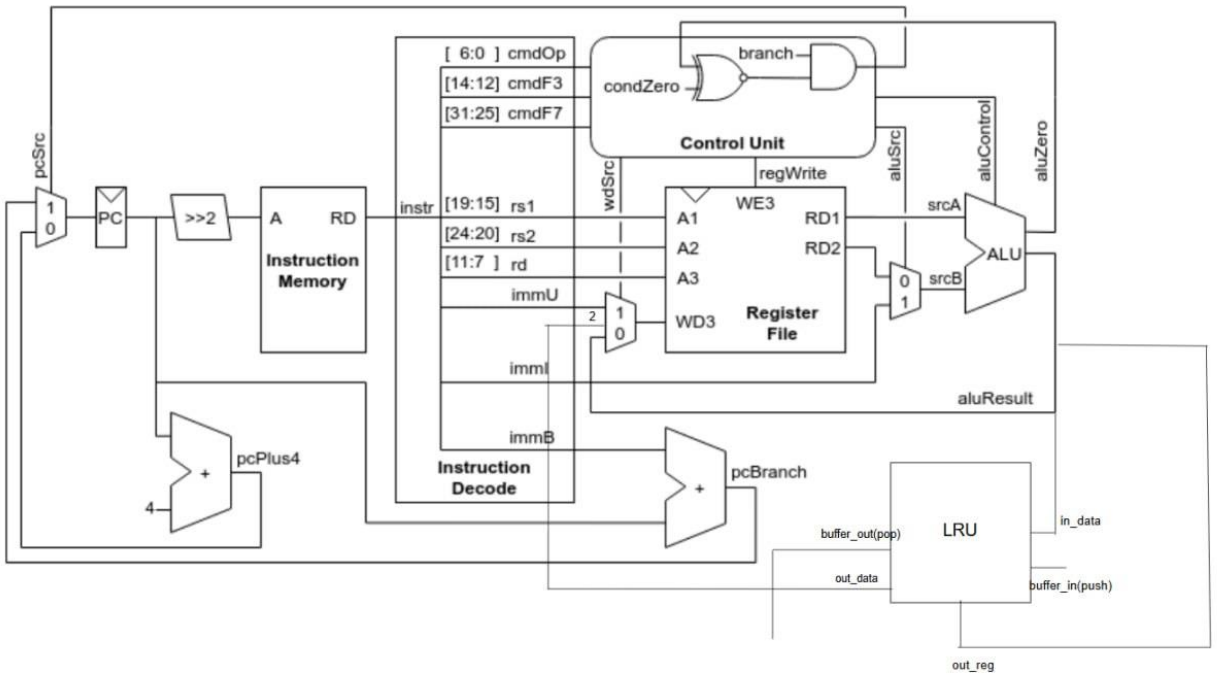
Описание алгоритма работы устройства

Команды, использованные из спецификаций Risc-V для добавления очереди,реализованной в предыдущей лабораторной работе:

push xN – загрузка данных из регистра xN в буффер.

pop xN – загрузка данных из буффера по индексу, хранящемся в xN в индекс хранящийся в xN.

Микроархитектурная диаграмма



Тестовое окружение

```
/*
 * schoolRISCV - small RISC-V CPU
 *
 * originally based on Sarah L. Harris MIPS CPU
 *                   & schoolMIPS project
 *
 * Copyright(c) 2017-2020 Stanislav Zhelnio
 *                        Aleksandr Romanov
 */

`timescale 1 ns / 100 ps

`include "sr_cpu.vh"

`ifndef SIMULATION_CYCLES
    `define SIMULATION_CYCLES 120
`endif

module sm_testbench;

    // simulation options
    parameter Tt     = 20;

    reg         clk;
    reg         rst_n;
    reg  [ 4:0] regAddr;
    wire        cpuClk;

    // ***** DUT start ************************

    sm_top sm_top
    (
        .clkIn     ( clk     ),
        .rst_n     ( rst_n   ),
        .clkDivide ( 4'b0    ),
        .clkEnable ( 1'b1    ),
        .clk       ( cpuClk  ),
        .regAddr   ( 5'b0    ),
        .regData   (         )
    );

    defparam sm_top.sm_clk_divider.bypass = 1;

    // ***** DUT  end  ************************

`ifdef ICARUS
    //iverilog memory dump init workaround
    initial $dumpvars;
    genvar k;
    for (k = 0; k < 32; k = k + 1) begin
        initial $dumpvars(0, sm_top.sm_cpu.rf.rf[k]);
    end
`endif

    // simulation init
    initial begin
        clk = 0;
        forever clk = #(Tt/2) ~clk;
    end

    initial begin
        rst_n   = 0;
        repeat (4)  @(posedge clk);
        rst_n   = 1;
    end

    task disasmInstr;

        reg [ 6:0] cmdOp;
        reg [ 4:0] rd;
```

```
71.         reg [ 2:0] cmdF3;
72.         reg [ 4:0] rs1;
73.         reg [ 4:0] rs2;
74.         reg [ 6:0] cmdF7;
75.         reg [31:0] immI;
76.         reg signed [31:0] immB;
77.         reg [31:0] immU;
78.
79.     begin
80.         cmdOp = sm_top.sm_cpu.cmdOp;
81.         rd    = sm_top.sm_cpu.rd;
82.         cmdF3 = sm_top.sm_cpu.cmdF3;
83.         rs1   = sm_top.sm_cpu.rs1;
84.         rs2   = sm_top.sm_cpu.rs2;
85.         cmdF7 = sm_top.sm_cpu.cmdF7;
86.         immI  = sm_top.sm_cpu.immI;
87.         immB  = sm_top.sm_cpu.immB;
88.         immU  = sm_top.sm_cpu.immU;
89.
90.         $write("   ");
91.         casez( { cmdF7, cmdF3, cmdOp } )
92.             default :                                   $write ("new/unknown");
93.             { `RVF7_ADD,  `RVF3_ADD,  `RVOP_ADD  } : $write ("add   $%1d, $%1d, $%1d", rd, rs1,
rs2);
94.             { `RVF7_OR,   `RVF3_OR,   `RVOP_OR   } : $write ("or    $%1d, $%1d, $%1d", rd, rs1,
rs2);
95.             { `RVF7_SRL,  `RVF3_SRL,  `RVOP_SRL  } : $write ("srl   $%1d, $%1d, $%1d", rd, rs1,
rs2);
96.             { `RVF7_SLTU, `RVF3_SLTU, `RVOP_SLTU } : $write ("sltu  $%1d, $%1d, $%1d", rd, rs1,
rs2);
97.             { `RVF7_SUB,  `RVF3_SUB,  `RVOP_SUB  } : $write ("sub   $%1d, $%1d, $%1d", rd, rs1,
rs2);
98.
99.             { `RVF7_ANY,  `RVF3_ADDI, `RVOP_ADDI } : $write ("addi  $%1d, $%1d, 0x%8h",rd, rs1,
immI);
100.            { `RVF7_ANY,  `RVF3_ANY,  `RVOP_LUI  } : $write ("lui   $%1d, 0x%8h",      rd, immU);
101.
102.            { `RVF7_ANY,  `RVF3_BEQ,  `RVOP_BEQ  } : $write ("beq   $%1d, $%1d, 0x%8h (%1d)", rs1,
rs2, immB, immB);
103.            { `RVF7_ANY,  `RVF3_BNE,  `RVOP_BNE  } : $write ("bne   $%1d, $%1d, 0x%8h (%1d)", rs1,
rs2, immB, immB);
104.
105.            { `RVF7_ANY, `RVF3_PUSH,  `RVOP_PUSH } : $write ("push   $%1d, $%1d, $%1d", rd, rs1,
rs2);
106.            { `RVF7_ANY, `RVF3_POP,  `RVOP_POP }   : $write ("pop   $%1d, $%1d, $%1d", rd, rs1,
rs2);
107.        endcase
108.    end
109.    endtask
110.
111.
112.    //simulation debug output
113.    integer cycle; initial cycle = 0;
114.    integer i;
115.
116.    always @ (posedge clk)
117.    begin
118.        $write ("%5d  pc = %2h instr = %h   a0 = %1d",
119.                  cycle, sm_top.sm_cpu.pc, sm_top.sm_cpu.instr, sm_top.sm_cpu.rf.rf[10]);
120.
121.        disasmInstr();
122.
123.        $write("\n");
124.
125.        $write ("LRU Buffer contents:\n");
126.        for (i = 0; i < 8; i = i + 1) begin
127.            $write ("Cache[%0d] = %d\n", i, sm_top.sm_cpu.buffer.cache_data[i]);
128.        end
129.
130.        $write ("CPU Registers:\n");
131.        for (i = 0; i < 32; i = i + 1) begin
132.            $write ("reg[%0d] = %d ", i, sm_top.sm_cpu.regData[i]);
133.            if ((i + 1) % 8 == 0) begin
134.                $write ("\n");
135.            end
```

```
136.         end
137.
138.         $write("\n");
139.
140.         cycle = cycle + 1;
141.
142.         if (cycle > `SIMULATION_CYCLES)
143.         begin
144.             cycle = 0;
145.             $display ("Timeout");
146.             $stop;
147.         end
148.     end
149.
150. endmodule
151.
```

sr_cpu.vh:

```
 1. /*
 2.  * schoolRISCV - small RISC-V CPU
 3.  *
 4.  * originally based on Sarah L. Harris MIPS CPU
 5.  *                   & schoolMIPS project
 6.  *
 7.  * Copyright(c) 2017-2020 Stanislav Zhelnio
 8.  *                        Aleksandr Romanov
 9.  */
10.
11. //ALU commands
12. `define ALU_ADD     3'b000
13. `define ALU_OR      3'b001
14. `define ALU_SRL     3'b010
15. `define ALU_SLTU    3'b011
16. `define ALU_SUB     3'b100
17.
18. // instruction opcode
19. `define RVOP_ADDI   7'b0010011
20. `define RVOP_BEQ    7'b1100011
21. `define RVOP_LUI    7'b0110111
22. `define RVOP_BNE    7'b1100011
23. `define RVOP_ADD    7'b0110011
24. `define RVOP_OR     7'b0110011
25. `define RVOP_SRL    7'b0110011
26. `define RVOP_SLTU   7'b0110011
27. `define RVOP_SUB    7'b0110011
28.
29. `define RVOP_PUSH   7'b1100110
30. `define RVOP_POP    7'b1100110
31.
32. // instruction funct3
33. `define RVF3_ADDI   3'b000
34. `define RVF3_BEQ    3'b000
35. `define RVF3_BNE    3'b001
36. `define RVF3_ADD    3'b000
37. `define RVF3_OR     3'b110
38. `define RVF3_SRL    3'b101
39. `define RVF3_SLTU   3'b011
40. `define RVF3_SUB    3'b000
41. `define RVF3_ANY    3'b???
42.
43. `define RVF3_PUSH   3'b000
44. `define RVF3_POP    3'b001
45.
46. // instruction funct7
47. `define RVF7_ADD    7'b0000000
48. `define RVF7_OR     7'b0000000
49. `define RVF7_SRL    7'b0000000
50. `define RVF7_SLTU   7'b0000000
51. `define RVF7_SUB    7'b0100000
52. `define RVF7_ANY    7'b???????
53.
```

sr_cpu.v

```
/*
 * schoolRISCV - small RISC-V CPU
 *
 * originally based on Sarah L. Harris MIPS CPU
 *                   & schoolMIPS project
 *
 * Copyright(c) 2017-2020 Stanislav Zhelnio
 *                        Aleksandr Romanov
 */

`include "sr_cpu.vh"

module sr_cpu
(
    input           clk,        // clock
    input           rst_n,      // reset
    input   [ 4:0]  regAddr,    // debug access reg address
    output  [31:0]  regData,    // debug access reg data
    output  [31:0]  imAddr,     // instruction memory address
    input   [31:0]  imData      // instruction memory data
);
    //control wires
    wire        aluZero;
    wire        pcSrc;
    wire        regWrite;
    wire        aluSrc;
    wire  [1:0] wdSrc;
    wire  [2:0] aluControl;

    //instruction decode wires
    wire [ 6:0] cmdOp;
    wire [ 4:0] rd;
    wire [ 2:0] cmdF3;
    wire [ 4:0] rs1;
    wire [ 4:0] rs2;
    wire [ 6:0] cmdF7;
    wire [31:0] immI;
    wire [31:0] immB;
    wire [31:0] immU;

    //program counter
    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);

    //program memory access
    assign imAddr = pc >> 2;
    wire [31:0] instr = imData;

    //instruction decode
    sr_decode id (
        .instr      ( instr        ),
        .cmdOp      ( cmdOp        ),
        .rd         ( rd           ),
        .cmdF3      ( cmdF3        ),
        .rs1        ( rs1          ),
        .rs2        ( rs2          ),
        .cmdF7      ( cmdF7        ),
        .immI       ( immI         ),
        .immB       ( immB         ),
        .immU       ( immU         )
    );

    //register file
    wire [31:0] rd0;
    wire [31:0] rd1;
    wire [31:0] rd2;
    wire [31:0] wd3;

    sm_register_file rf (
```

```
 73.         .clk        ( clk          ),
 74.         .a0         ( regAddr      ),
 75.         .a1         ( rs1          ),
 76.         .a2         ( rs2          ),
 77.         .a3         ( rd           ),
 78.         .rd0        ( rd0          ),
 79.         .rd1        ( rd1          ),
 80.         .rd2        ( rd2          ),
 81.         .wd3        ( wd3          ),
 82.         .we3        ( regWrite     )
 83.     );
 84.
 85.     //debug register access
 86.     assign regData = (regAddr != 0) ? rd0 : pc;
 87.
 88.     //alu
 89.     wire [31:0] srcB = aluSrc ? immI : rd2;
 90.     wire [31:0] aluResult;
 91.
 92.     sr_alu alu (
 93.         .srcA       ( rd1          ),
 94.         .srcB       ( srcB         ),
 95.         .oper       ( aluControl   ),
 96.         .zero       ( aluZero      ),
 97.         .result     ( aluResult    )
 98.     );
 99.
100.     // Buffer Logic
101.     wire [15:0] buffer_out_data;
102.
103.     assign wd3 =
104.         wdSrc == 2'b01 ? immU :
105.         wdSrc == 2'b00 ? aluResult:
106.         buffer_out_data;
107.
108.     wire buffer_in_valid;
109.     wire buffer_out_ready;
110.
111.     //control
112.     sr_control sm_control (
113.         .cmdOp      ( cmdOp        ),
114.         .cmdF3      ( cmdF3        ),
115.         .cmdF7      ( cmdF7        ),
116.         .aluZero    ( aluZero      ),
117.         .pcSrc      ( pcSrc        ),
118.         .regWrite   ( regWrite     ),
119.         .aluSrc     ( aluSrc       ),
120.         .wdSrc      ( wdSrc        ),
121.         .aluControl ( aluControl   ),
122.         .buffer_in_valid ( buffer_in_valid ),
123.         .buffer_out_ready ( buffer_out_ready )
124.     );
125.
126.     // LRU Buffer
127.
128.     wire in_ready;
129.     wire out_valid;
130.
131.     LRU_Buffer #(
132.         .CACHE_SIZE(8),
133.         .DATA_SIZE(16)
134.     ) buffer (
135.         .clk(clk),
136.         .reset(rst_n),
137.         .in_valid(buffer_in_valid),
138.         .in_data(aluResult[15:0]),
139.         .in_ready(in_ready),
140.         .out_data(buffer_out_data),
141.         .out_ready(buffer_out_ready),
142.         .out_valid(out_valid),
143.         .access_index(aluResult[2:0])
144.     );
145.
146. endmodule
147.
```

```
148. module sr_decode
149. (
150.     input      [31:0] instr,
151.     output     [ 6:0] cmdOp,
152.     output     [ 4:0] rd,
153.     output     [ 2:0] cmdF3,
154.     output     [ 4:0] rs1,
155.     output     [ 4:0] rs2,
156.     output     [ 6:0] cmdF7,
157.     output reg [31:0] immI,
158.     output reg [31:0] immB,
159.     output reg [31:0] immU
160. );
161.     assign cmdOp = instr[ 6: 0];
162.     assign rd    = instr[11: 7];
163.     assign cmdF3 = instr[14:12];
164.     assign rs1   = instr[19:15];
165.     assign rs2   = instr[24:20];
166.     assign cmdF7 = instr[31:25];
167.
168.     // I-immediate
169.     always @ (*) begin
170.         immI[10: 0] = instr[30:20];
171.         immI[31:11] = { 21 {instr[31]} };
172.     end
173.
174.     // B-immediate
175.     always @ (*) begin
176.         immB[    0] = 1'b0;
177.         immB[ 4: 1] = instr[11:8];
178.         immB[10: 5] = instr[30:25];
179.         immB[   11] = instr[7];
180.         immB[31:12] = { 20 {instr[31]} };
181.     end
182.
183.     // U-immediate
184.     always @ (*) begin
185.         immU[11: 0] = 12'b0;
186.         immU[31:12] = instr[31:12];
187.     end
188.
189. endmodule
190.
191. module sr_control
192. (
193.     input     [ 6:0] cmdOp,
194.     input     [ 2:0] cmdF3,
195.     input     [ 6:0] cmdF7,
196.     input            aluZero,
197.     output           pcSrc,
198.     output reg       regWrite,
199.     output reg       aluSrc,
200.     output reg [1:0] wdSrc,
201.     output reg [2:0] aluControl,
202.
203.     output reg buffer_in_valid,
204.     output reg buffer_out_ready
205. );
206.     reg          branch;
207.     reg          condZero;
208.     assign pcSrc = branch & (aluZero == condZero);
209.
210.     always @ (*) begin
211.         branch      = 1'b0;
212.         condZero    = 1'b0;
213.         regWrite    = 1'b0;
214.         aluSrc      = 1'b0;
215.         wdSrc       = 2'b00;
216.         aluControl  = `ALU_ADD;
217. //        buffer_in_valid   = 1'b0;
218. //        buffer_out_ready  = 1'b0;
219.
220.         casez( {cmdF7, cmdF3, cmdOp} )
221.             { `RVF7_ADD, `RVF3_ADD,  `RVOP_ADD  }  : begin regWrite = 1'b1; aluControl = `ALU_ADD;
end
```

```
222.             { `RVF7_OR,   `RVF3_OR,   `RVOP_OR   } : begin regWrite = 1'b1; aluControl = `ALU_OR;
end
223.             { `RVF7_SRL,  `RVF3_SRL,  `RVOP_SRL  } : begin regWrite = 1'b1; aluControl = `ALU_SRL;
end
224.             { `RVF7_SLTU, `RVF3_SLTU, `RVOP_SLTU } : begin regWrite = 1'b1; aluControl = `ALU_SLTU;
end
225.             { `RVF7_SUB,  `RVF3_SUB,  `RVOP_SUB  } : begin regWrite = 1'b1; aluControl = `ALU_SUB;
end
226.
227.             { `RVF7_ANY,  `RVF3_ADDI, `RVOP_ADDI } : begin regWrite = 1'b1; aluSrc = 1'b1;
aluControl = `ALU_ADD; end
228.             { `RVF7_ANY,  `RVF3_ANY,  `RVOP_LUI  } : begin regWrite = 1'b1; wdSrc  = 2'b01; end
229.
230.             { `RVF7_ANY,  `RVF3_BEQ,  `RVOP_BEQ  } : begin branch = 1'b1; condZero = 1'b1;
aluControl = `ALU_SUB; end
231.             { `RVF7_ANY,  `RVF3_BNE,  `RVOP_BNE  } : begin branch = 1'b1; aluControl = `ALU_SUB; end
232.
233.             { `RVF7_ANY, `RVF3_PUSH,  `RVOP_PUSH } :
234.             begin
235.                 buffer_in_valid = 1'b1;
236.             end
237.
238.             { `RVF7_ANY, `RVF3_POP,   `RVOP_POP  } :
239.             begin
240.                 buffer_out_ready = 1'b1;
241.                 wdSrc = 2'b10;
242.                 regWrite = 1'b1;
243.             end
244.         endcase
245.     end
246. endmodule
247.
248. module sr_alu
249. (
250.     input  [31:0] srcA,
251.     input  [31:0] srcB,
252.     input  [ 2:0] oper,
253.     output        zero,
254.     output reg [31:0] result
255. );
256.     always @ (*) begin
257.         case (oper)
258.             default   : result = srcA + srcB;
259.             `ALU_ADD  : result = srcA + srcB;
260.             `ALU_OR   : result = srcA | srcB;
261.             `ALU_SRL  : result = srcA >> srcB [4:0];
262.             `ALU_SLTU : result = (srcA < srcB) ? 1 : 0;
263.             `ALU_SUB : result = srcA - srcB;
264.         endcase
265.     end
266.
267.     assign zero   = (result == 0);
268. endmodule
269.
270. module sm_register_file
271. (
272.     input         clk,
273.     input  [ 4:0] a0,
274.     input  [ 4:0] a1,
275.     input  [ 4:0] a2,
276.     input  [ 4:0] a3,
277.     output [31:0] rd0,
278.     output [31:0] rd1,
279.     output [31:0] rd2,
280.     input  [31:0] wd3,
281.     input         we3
282. );
283.     reg [31:0] rf [31:0];
284.
285.     assign rd0 = (a0 != 0) ? rf [a0] : 32'b0;
286.     assign rd1 = (a1 != 0) ? rf [a1] : 32'b0;
287.     assign rd2 = (a2 != 0) ? rf [a2] : 32'b0;
288.
289.     always @ (posedge clk)
290.         if(we3) rf [a3] <= wd3;
```

```
endmodule

```

Результат тестирования

```
    0  pc = 00 instr = 00000513   a0 = 0   addi  $10, $0, 0x00000000
LRU Buffer contents:
Cache[0] =     0
Cache[1] =     0
Cache[2] =     0
Cache[3] =     0
Cache[4] =     0
Cache[5] =     0
Cache[6] =     0
Cache[7] =     0
CPU Registers:
reg[0] = 0 reg[1] = 0 reg[2] = 0 reg[3] = 0 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

    1  pc = 04 instr = 000000523   a0 = 0   addi  $1, $0, 0x00000001
LRU Buffer contents:
Cache[0] =     0
Cache[1] =     0
Cache[2] =     0
Cache[3] =     0
Cache[4] =     0
Cache[5] =     0
Cache[6] =     0
Cache[7] =     0
CPU Registers:
reg[0] = 0 reg[1] = 1 reg[2] = 0 reg[3] = 0 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

    2  pc = 08 instr = 000000521   a0 = 0   addi  $2, $0, 0x00000002
LRU Buffer contents:
Cache[0] =     0
Cache[1] =     0
Cache[2] =     0
Cache[3] =     0
Cache[4] =     0
Cache[5] =     0
Cache[6] =     0
Cache[7] =     0
CPU Registers:
reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 0 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

    3  pc = 0c instr = 00000032   a0 = 0   addi  $3, $0, 0x00000003
LRU Buffer contents:
Cache[0] =     0
Cache[1] =     0
Cache[2] =     0
Cache[3] =     0
Cache[4] =     0
Cache[5] =     0
Cache[6] =     0
Cache[7] =     0
CPU Registers:
reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0
```

```
64.
65.     4  pc = 10 instr = 000000432   a0 = 0   push   $1, $0, $0
66. LRU Buffer contents:
67. Cache[0] =     1
68. Cache[1] =     0
69. Cache[2] =     0
70. Cache[3] =     0
71. Cache[4] =     0
72. Cache[5] =     0
73. Cache[6] =     0
74. Cache[7] =     0
75. CPU Registers:
76. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
77. reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
78. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
79. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0
80.
81.     5  pc = 14 instr = 000000e6   a0 = 0   push   $2, $0, $0
82. LRU Buffer contents:
83. Cache[0] =     1
84. Cache[1] =     2
85. Cache[2] =     0
86. Cache[3] =     0
87. Cache[4] =     0
88. Cache[5] =     0
89. Cache[6] =     0
90. Cache[7] =     0
91. CPU Registers:
92. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
93. reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
94. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
95. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0
96.
97.     6  pc = 18 instr = 000000e6   a0 = 0   push   $3, $0, $0
98. LRU Buffer contents:
99. Cache[0] =     1
100. Cache[1] =     2
101. Cache[2] =     3
102. Cache[3] =     0
103. Cache[4] =     0
104. Cache[5] =     0
105. Cache[6] =     0
106. Cache[7] =     0
107. CPU Registers:
108. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
109. reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
110. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
111. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0
112.
113.     7  pc = 1c instr = 00000513   a0 = 0   addi  $10, $0, 0x00000001
114. LRU Buffer contents:
115. Cache[0] =     1
116. Cache[1] =     2
117. Cache[2] =     3
118. Cache[3] =     0
119. Cache[4] =     0
120. Cache[5] =     0
121. Cache[6] =     0
122. Cache[7] =     0
123. CPU Registers:
124. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
125. reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
126. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
127. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0
128.
129.     8  pc = 20 instr = 00000513   a0 = 0   addi  $11, $0, 0x00000002
130. LRU Buffer contents:
131. Cache[0] =     1
132. Cache[1] =     2
133. Cache[2] =     3
134. Cache[3] =     0
135. Cache[4] =     0
136. Cache[5] =     0
137. Cache[6] =     0
138. Cache[7] =     0
```

```
CPU Registers:
reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

    9  pc = 24 instr = 00000513   a0 = 0   addi  $12, $0, 0x00000003
LRU Buffer contents:
Cache[0] =     1
Cache[1] =     2
Cache[2] =     3
Cache[3] =     0
Cache[4] =     0
Cache[5] =     0
Cache[6] =     0
Cache[7] =     0
CPU Registers:
reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 3 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

    10 pc = 28 instr = 000000e6   a0 = 0   pop   $10, $0, $0
LRU Buffer contents:
Cache[0] =     1
Cache[1] =     2
Cache[2] =     3
Cache[3] =     0
Cache[4] =     0
Cache[5] =     0
Cache[6] =     0
Cache[7] =     0
CPU Registers:
reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 3 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

    11 pc = 2c instr = 000000e6   a0 = 0   pop   $11, $0, $0
LRU Buffer contents:
Cache[0] =     1
Cache[1] =     2
Cache[2] =     3
Cache[3] =     0
Cache[4] =     0
Cache[5] =     0
Cache[6] =     0
Cache[7] =     0
CPU Registers:
reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 3 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

    12 pc = 30 instr = 000000e6   a0 = 0   pop   $12, $0, $0
LRU Buffer contents:
Cache[0] =     1
Cache[1] =     2
Cache[2] =     3
Cache[3] =     0
Cache[4] =     0
Cache[5] =     0
Cache[6] =     0
Cache[7] =     0
CPU Registers:
reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0
reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 3 reg[13] = 0 reg[14] = 0 reg[15] = 0
reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0
reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

```

Программа для тестирования

```
program.hex

0x00000533
0x00000527
0x00000521
0x0000006e
0x00208033
0x00033013
0x00013013
0x00023013
0x00033013

```

Выводы

В ходе лабораторной работы нам удалось познакомиться с упрощённой архитектурой RISC-V на практике, а также самим оценить её преимущества над x86 – в частности преимущество более компактного представления команд и их формата. Также нам удалось получить практический опыт в её совершенствовании – мы полностью разобрались как работает instruction fetch, data path, control unit и смогли реализовать собственные команды. На собственном примере мы убедились в том, как структура такого процессора сложна. Добавление двух новых команд заставило нас проводить поэтапное тестирование кода и тщательное изучение ядра. Но всё же у нас получилось внедрить аккуратно без колоссальных изменений встраиваемый блок буффера и выполнить данную лабораторную работу.