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

[Содержание 2](#_TOC_250007)

[Задание 3](#_TOC_250006)

[Описание алгоритма работы устройства 4](#_TOC_250005)

[Микро-архитектурная диаграмма 5](#_TOC_250004)

[Тестовое окружение 6](#_TOC_250003)

[Результат тестирования 9](#_TOC_250002)

[Программа для тестирования 12](#_TOC_250001)

[Выводы 13](#_TOC_250000)

# 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.

Примечание:

При непосредственном описании ваших модификаций в коде проекта запрещено использовать не синтезируемые конструкции и арифметические операции, отличные от 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

Изображение выглядит как диаграмма, План, Технический чертеж, схематичный

Автоматически созданное описание

# Тестовое окружение

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. `timescale 1 ns / 100 ps

12.

13. `include "sr\_cpu.vh"

14.

15. `ifndef SIMULATION\_CYCLES

16. `define SIMULATION\_CYCLES 120

17. `endif

18.

19. module sm\_testbench;

20.

21. // simulation options

22. parameter Tt = 20;

23.

24. reg clk;

25. reg rst\_n;

26. reg [ 4:0] regAddr;

27. wire cpuClk;

28.

29. // \*\*\*\*\* DUT start \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

30.

31. sm\_top sm\_top

32. (

33. .clkIn ( clk ),

34. .rst\_n ( rst\_n ),

35. .clkDivide ( 4'b0 ),

36. .clkEnable ( 1'b1 ),

37. .clk ( cpuClk ),

38. .regAddr ( 5'b0 ),

39. .regData ( )

40. );

41.

42. defparam sm\_top.sm\_clk\_divider.bypass = 1;

43.

44. // \*\*\*\*\* DUT end \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

45.

46. `ifdef ICARUS

47. //iverilog memory dump init workaround

48. initial $dumpvars;

49. genvar k;

50. for (k = 0; k < 32; k = k + 1) begin

51. initial $dumpvars(0, sm\_top.sm\_cpu.rf.rf[k]);

52. end

53. `endif

54.

55. // simulation init

56. initial begin

57. clk = 0;

58. forever clk = #(Tt/2) ~clk;

59. end

60.

61. initial begin

62. rst\_n = 0;

63. repeat (4) @(posedge clk);

64. rst\_n = 1;

65. end

66.

67. task disasmInstr;

68.

69. reg [ 6:0] cmdOp;

70. 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

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. `include "sr\_cpu.vh"

12.

13. module sr\_cpu

14. (

15. input clk, // clock

16. input rst\_n, // reset

17. input [ 4:0] regAddr, // debug access reg address

18. output [31:0] regData, // debug access reg data

19. output [31:0] imAddr, // instruction memory address

20. input [31:0] imData // instruction memory data

21. );

22. //control wires

23. wire aluZero;

24. wire pcSrc;

25. wire regWrite;

26. wire aluSrc;

27. wire [1:0] wdSrc;

28. wire [2:0] aluControl;

29.

30. //instruction decode wires

31. wire [ 6:0] cmdOp;

32. wire [ 4:0] rd;

33. wire [ 2:0] cmdF3;

34. wire [ 4:0] rs1;

35. wire [ 4:0] rs2;

36. wire [ 6:0] cmdF7;

37. wire [31:0] immI;

38. wire [31:0] immB;

39. wire [31:0] immU;

40.

41. //program counter

42. wire [31:0] pc;

43. wire [31:0] pcBranch = pc + immB;

44. wire [31:0] pcPlus4 = pc + 4;

45. wire [31:0] pcNext = pcSrc ? pcBranch : pcPlus4;

46. sm\_register r\_pc(clk ,rst\_n, pcNext, pc);

47.

48. //program memory access

49. assign imAddr = pc >> 2;

50. wire [31:0] instr = imData;

51.

52. //instruction decode

53. sr\_decode id (

54. .instr ( instr ),

55. .cmdOp ( cmdOp ),

56. .rd ( rd ),

57. .cmdF3 ( cmdF3 ),

58. .rs1 ( rs1 ),

59. .rs2 ( rs2 ),

60. .cmdF7 ( cmdF7 ),

61. .immI ( immI ),

62. .immB ( immB ),

63. .immU ( immU )

64. );

65.

66. //register file

67. wire [31:0] rd0;

68. wire [31:0] rd1;

69. wire [31:0] rd2;

70. wire [31:0] wd3;

71.

72. 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;

291. endmodule

292.

## Результат тестирования

1. 0 pc = 00 instr = 00000513 a0 = 0 addi $10, $0, 0x00000000

2. LRU Buffer contents:

3. Cache[0] = 0

4. Cache[1] = 0

5. Cache[2] = 0

6. Cache[3] = 0

7. Cache[4] = 0

8. Cache[5] = 0

9. Cache[6] = 0

10. Cache[7] = 0

11. CPU Registers:

12. reg[0] = 0 reg[1] = 0 reg[2] = 0 reg[3] = 0 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

13. reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0

14. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

15. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

16.

17. 1 pc = 04 instr = 000000523 a0 = 0 addi $1, $0, 0x00000001

18. LRU Buffer contents:

19. Cache[0] = 0

20. Cache[1] = 0

21. Cache[2] = 0

22. Cache[3] = 0

23. Cache[4] = 0

24. Cache[5] = 0

25. Cache[6] = 0

26. Cache[7] = 0

27. CPU Registers:

28. reg[0] = 0 reg[1] = 1 reg[2] = 0 reg[3] = 0 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

29. reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0

30. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

31. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

32.

33. 2 pc = 08 instr = 000000521 a0 = 0 addi $2, $0, 0x00000002

34. LRU Buffer contents:

35. Cache[0] = 0

36. Cache[1] = 0

37. Cache[2] = 0

38. Cache[3] = 0

39. Cache[4] = 0

40. Cache[5] = 0

41. Cache[6] = 0

42. Cache[7] = 0

43. CPU Registers:

44. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 0 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

45. reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0

46. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

47. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

48.

49. 3 pc = 0c instr = 00000032 a0 = 0 addi $3, $0, 0x00000003

50. LRU Buffer contents:

51. Cache[0] = 0

52. Cache[1] = 0

53. Cache[2] = 0

54. Cache[3] = 0

55. Cache[4] = 0

56. Cache[5] = 0

57. Cache[6] = 0

58. Cache[7] = 0

59. CPU Registers:

60. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

61. reg[8] = 0 reg[9] = 0 reg[10] = 0 reg[11] = 0 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0

62. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

63. 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

139. CPU Registers:

140. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

141. reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 0 reg[13] = 0 reg[14] = 0 reg[15] = 0

142. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

143. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

144.

145. 9 pc = 24 instr = 00000513 a0 = 0 addi $12, $0, 0x00000003

146. LRU Buffer contents:

147. Cache[0] = 1

148. Cache[1] = 2

149. Cache[2] = 3

150. Cache[3] = 0

151. Cache[4] = 0

152. Cache[5] = 0

153. Cache[6] = 0

154. Cache[7] = 0

155. CPU Registers:

156. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

157. reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 3 reg[13] = 0 reg[14] = 0 reg[15] = 0

158. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

159. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

160.

161. 10 pc = 28 instr = 000000e6 a0 = 0 pop $10, $0, $0

162. LRU Buffer contents:

163. Cache[0] = 1

164. Cache[1] = 2

165. Cache[2] = 3

166. Cache[3] = 0

167. Cache[4] = 0

168. Cache[5] = 0

169. Cache[6] = 0

170. Cache[7] = 0

171. CPU Registers:

172. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

173. reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 3 reg[13] = 0 reg[14] = 0 reg[15] = 0

174. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

175. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

176.

177. 11 pc = 2c instr = 000000e6 a0 = 0 pop $11, $0, $0

178. LRU Buffer contents:

179. Cache[0] = 1

180. Cache[1] = 2

181. Cache[2] = 3

182. Cache[3] = 0

183. Cache[4] = 0

184. Cache[5] = 0

185. Cache[6] = 0

186. Cache[7] = 0

187. CPU Registers:

188. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

189. reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 3 reg[13] = 0 reg[14] = 0 reg[15] = 0

190. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

191. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

192.

193. 12 pc = 30 instr = 000000e6 a0 = 0 pop $12, $0, $0

194. LRU Buffer contents:

195. Cache[0] = 1

196. Cache[1] = 2

197. Cache[2] = 3

198. Cache[3] = 0

199. Cache[4] = 0

200. Cache[5] = 0

201. Cache[6] = 0

202. Cache[7] = 0

203. CPU Registers:

204. reg[0] = 0 reg[1] = 1 reg[2] = 2 reg[3] = 3 reg[4] = 0 reg[5] = 0 reg[6] = 0 reg[7] = 0

205. reg[8] = 0 reg[9] = 0 reg[10] = 1 reg[11] = 2 reg[12] = 3 reg[13] = 0 reg[14] = 0 reg[15] = 0

206. reg[16] = 0 reg[17] = 0 reg[18] = 0 reg[19] = 0 reg[20] = 0 reg[21] = 0 reg[22] = 0 reg[23] = 0

207. reg[24] = 0 reg[25] = 0 reg[26] = 0 reg[27] = 0 reg[28] = 0 reg[29] = 0 reg[30] = 0 reg[31] = 0

208.

# Программа для тестирования

1. program.hex

2.

3. 0x00000533

4. 0x00000527

5. 0x00000521

6. 0x0000006e

7. 0x00208033

8. 0x00033013

9. 0x00013013

10. 0x00023013

11. 0x00033013

12.

# 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.