

Министерство образования Республики Беларусь  
Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей  
Кафедра информатики  
Дисциплина: архитектура вычислительных систем

**ОТЧЁТ**  
к лабораторной работе  
на тему  
**Однотактный процессор**

Выполнил: студент группы 053506  
Слуцкий Никита Сергеевич

Проверил: Шиманский Валерий Владимирович

Минск 2022

## Цели лабораторной работы

Лабораторная работа разделена на три части:

В первой части этой лабораторной работы, вы расширите одноктактный процессор RISC-V для поддержки дополнительных инструкций в зависимости от варианта.

Во второй и третьей частях вы будете проектировать многотактный процессор RISC-V на SystemVerilog и протестируете его на простой программе, написанной на машинном языке. Это свяжет воедино всё, что вы узнали из курса о цифровом проектировании, языках описания аппаратуры, языке ассемблера и микроархитектуре, а также даст вам возможность спроектировать и отладить сложную систему. В части 2 вы соберете и протестируете контроллер. В части 3 вы соберете тракт данных и протестируете всю систему.

### Часть 1. Одноктактный процессор RISC-V

В файле `src/riscvtest.txt` находится код процессора, который вам предстоит изменить. На [изобр. 1.1](#) показан полный одноктактный процессор из книги. На [изобр. 1.2](#) показано устройство управления, а на [изобр. 1.3](#) показано АЛУ (Арифметико-логическое устройство). На [таблицах 1.1 и 1.2](#) показаны таблицы истинности главного декодера и декодера АЛУ. На [таблице 1.3](#) показано кодирование ImmSrc. На [изобр. 1.4](#) показана тестовая программа для одноктактного процессора RISC-V из книги.

#### Вариант 5:

Инструкция *lui*.

#### РЕШЕНИЕ:

Код измененного модуля datapath в `riscsingle.sv`:

```

module module datapath(input logic clk, reset,
                      input logic [1:0] ResultSrc,
                      input logic PCSrc, ALUSrc,
                      input logic RegWrite,
                      input logic [1:0] ImmSrc,
                      input logic [2:0] ALUControl,
                      output logic Zero,
                      output logic [31:0] PC,
                      input logic [31:0] Instr,
                      output logic [31:0] ALUResult, WriteData,
                      input logic [31:0] ReadData,
                      input logic UImm);

logic [31:0] PCNext, PCPlus4, PCTarget;
logic [31:0] ImmExt;
logic [31:0] SrcA, SrcB;
logic [31:0] Result;
logic [31:0] tmp;

// next PC logic
flopr #(32) pcreg(clk, reset, PCNext, PC);
adder      pcadd4(PC, 32'd4, PCPlus4);
adder      pcaddbranch(PC, ImmExt, PCTarget);
mux2 #(32) pcmux(PCPlus4, PCTarget, PCSrc, PCNext);
// register file logic
regfile    rf(clk, RegWrite, Instr[19:15], Instr[24:20],
              Instr[11:7], Result, SrcA, WriteData);
extend     ext(Instr[31:7], ImmSrc, ImmExt);

// ALU logic
mux2 #(32) srcbmux(WriteData, ImmExt, ALUSrc, SrcB);
alu        alu(SrcA, SrcB, ALUControl, ALUResult, Zero);
mux3 #(32) resultmux(ALUResult, ReadData, PCPlus4, ResultSrc, tmp);
mux2 #(32) uimmtestmux(tmp, Instr[31:12] << 12, UImm, Result);

endmodule

```

Схема одноктактного процессора:

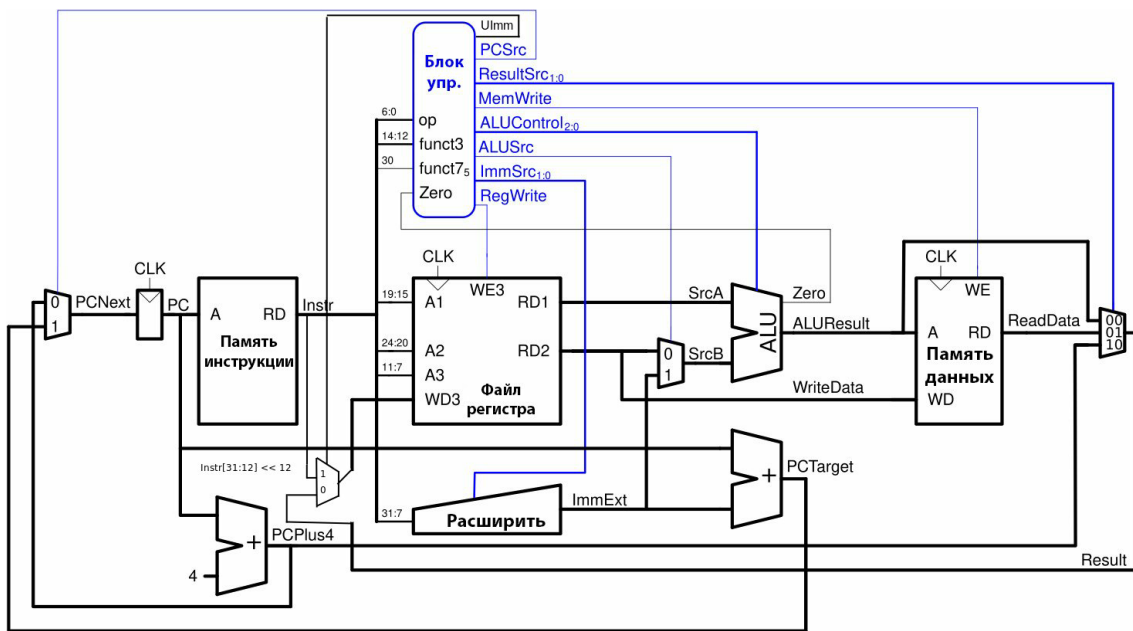
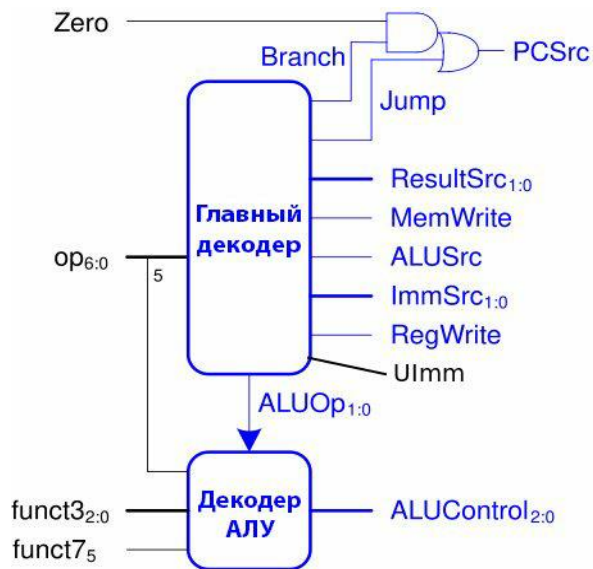


Схема измененного контроллера:



Измененный файл для тестов riscvtest.s:

#	RISC-V Assembly	Description	Address	Machine
main:	addi x2, x0, 5	# x2 = 5	0	00500113
	addi x3, x0, 12	# x3 = 12	4	00C00193
	addi x7, x3, -9	# x7 = (12 - 9) = 3	8	FF718393
	or x4, x7, x2	# x4 = (3 OR 5) = 7	C	0023E233
	and x5, x3, x4	# x5 = (12 AND 7) = 4	10	0041F2B3
	add x5, x5, x4	# x5 = (4 + 7) = 11	14	004282B3
	beq x5, x7, end	# shouldn't be taken	18	02728863
	slt x4, x3, x4	# x4 = (12 < 7) = 0	1C	0041A233
	beq x4, x0, around	# should be taken	20	00020463
	addi x5, x0, 0	# shouldn't happen	24	00000293
around:	slt x4, x7, x2	# x4 = (3 < 5) = 1	28	0023A233
	add x7, x4, x5	# x7 = (1 + 11) = 12	2C	005203B3
	sub x7, x7, x2	# x7 = (12 - 5) = 7	30	402383B3
	sw x7, 84(x3)	# [96] = 7	34	0471AA23
	lw x2, 96(x0)	# x2 = [96] = 7	38	06002103
	add x9, x2, x5	# x9 = (7 + 11) = 18	3C	005104B3
	jal x3, end	# jump to end, x3 = 0x44	40	008001EF
	addi x2, x0, 1	# shouldn't happen	44	00100113
end:	add x2, x2, x9	# x2 = (7 + 18) = 25	48	00910133
	# new logic begin			
	addi x2, x2, -1024	# x2 = x2 - 1024	4C	C0010113
	addi x2, x2, -1024	# x2 = x2 - 1024	50	C0010113
	addi x2, x2, -1024	# x2 = x2 - 1024	54	C0010113
	addi x2, x2, -1024	# x2 = x2 - 1024	58	C0010113
	lui x4, 1	# x4 = 4096	5C	00001237
	add x2, x2, x4	# x2 = x2 - 4096 + 4096 = 25	60	00410133
	# new logic end			
	sw x2, 0x20(x3)	# mem[100] = 25	64	0221A023
done:	beq x2, x2, done	# infinite loop	68	00210063

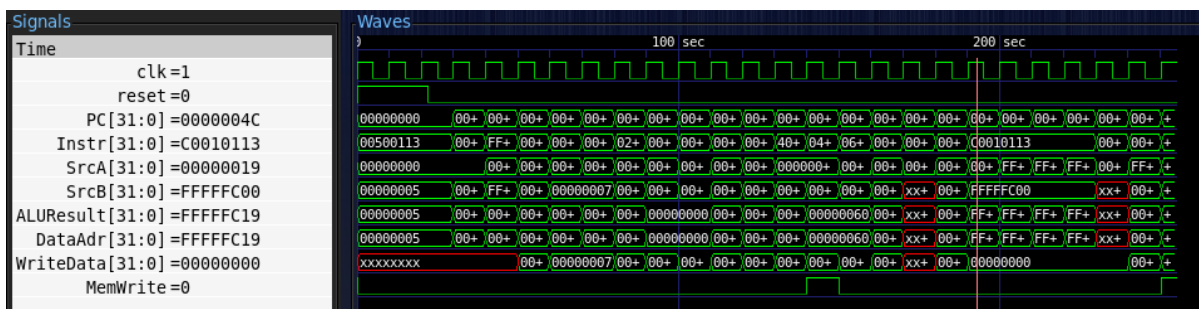
Вывод программы:

```

dkefir03@debian ~/labs/csa-univer/lab04/src <main>
$ vvp rsingle.vvp
WARNING: riscvsingle.sv:348: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: riscvsingle.sv:348: Excess hex digits (2 of '0x00500113') while reading 32-bit words.
WARNING: riscvsingle.sv:348: $readmemh(riscvtest.txt): Not enough words in the file for the requested range [0:63].
VCD info: dumpfile test.vcd opened for output.
Simulation succeeded
** VVP Stop(0) **
** Flushing output streams. **
** Current simulation time is 195 ticks.
> finish
** Continue **
dkefir03@debian ~/labs/csa-univer/lab04/src <main>
$ ~

```

Полученные осциллограммы симуляции:



## Часть 2. Многотактный контроллер RISC-V

Перед началом разработки контроллера, взгляните на следующие диаграммы. Все изображения и таблицы находятся в конце данной страницы.

- На [изобр. 2.1](#) показана диаграмма блока управления многотактным контроллером
- На [изобр. 2.2](#) показана диаграмма многотактного управления главного конечного автомата
- [Таблица 2.1](#) и [пример языка ОА 1](#) (описания аппаратуры) определяют логику декодера АЛУ.
- [Таблица 2.2](#) и [пример языка ОА 2](#) определяют логику декодера инструкций.

Напишите иерархическое описание многотактного контроллера на SystemVerilog. Когда вам не важны выводы, вы можете установить их в 0, чтобы они имели детерминированное значение для простоты тестирования.

### РЕШЕНИЕ:

Полученный модуль `controller` в файле `controller_testbench.sv`:

```

typedef enum logic[6:0] {r_type_op=7'b0110011,
                          i_type_alu_op=7'b0010011,
                          lw_op=7'b0000011,
                          sw_op=7'b0100011,
                          beq_op=7'b1100011,
                          jal_op=7'b1101111} opcode_type;

module controller(input logic clk,
                  input logic reset,
                  input opcode_type op,
                  input logic [2:0] funct3,
                  input logic funct7b5,
                  input logic Zero,

                  output logic [1:0] ImmSrc,
                  output logic [1:0] ALUSrcA, ALUSrcB,
                  output logic [1:0] ResultSrc,
                  output logic AdrSrc,
                  output logic [2:0] ALUControl,
                  output logic IRWrite, PCWrite,
                  output logic RegWrite, MemWrite);

    typedef enum logic [5:0] {
        SFetch,
        SDecode,
        SMemAdr,
        SMemRead,
        SMemWB,
        SMemWrite,
        SExecuteR,
        SExecuteI,
        SALUWB,
        SBEQ,
        SJAL
    } ControllerState;

    ControllerState state, nextstate;

    logic [1:0] ALUOp;

    aludec      ad(op[5], funct3, funct7b5, ALUOp, ALUControl);
    instrdec    id(op, ImmSrc);

```

```

always_ff @(posedge clk, posedge reset)
    if (reset) state <= SFetch;
    else state <= nextstate;

always_comb
    case (state)
        SFetch: begin
            AdrSrc = 0;
            ALUSrcA = 2'b00;
            ALUSrcB = 2'b10;
            ALUOp = 2'b00;
            ResultSrc = 2'b10;
            PCWrite = 1;
            IRWrite = 1;
            RegWrite = 0;
            MemWrite = 0;
            nextstate = SDecode;
        end
        SDecode: begin
            ALUSrcA = 2'b01;
            ALUSrcB = 2'b01;
            ALUOp = 2'b00;
            PCWrite = 0;
            IRWrite = 0;
            RegWrite = 0;
            MemWrite = 0;
            if (op === lw_op | op === sw_op) begin
                nextstate = SMemAdr;
            end
            else if (op === r_type_op) begin
                nextstate = SExecuteR;
            end
            else if (op === i_type_alu_op) begin
                nextstate = SExecuteI;
            end
            else if (op === jal_op) begin
                nextstate = SJAL;
            end
            else if (op === beq_op) begin
                nextstate = SBEQ;
            end
        end
    end
end

```



```

SMemAdr: begin
    ALUSrcA = 2'b10;
    ALUSrcB = 2'b01;
    ALUOp = 2'b00;
    PCWrite = 0;
    IRWrite = 0;
    RegWrite = 0;
    MemWrite = 0;
    if (op === lw_op) begin
        nextstate = SMemRead;
    end
    else begin
        nextstate = SMemWrite;
    end
end
SMemRead: begin
    ResultSrc = 2'b00;
    AdrSrc = 1;
    PCWrite = 0;
    IRWrite = 0;
    RegWrite = 0;
    MemWrite = 0;
    nextstate = SMemWB;
end
SMemWB: begin
    ResultSrc = 2'b01;
    PCWrite = 0;
    IRWrite = 0;
    RegWrite = 1;
    MemWrite = 0;
    nextstate = SFetch;
end
SMemWrite: begin
    ResultSrc = 2'b00;
    AdrSrc = 1;
    MemWrite = 1;
    nextstate = SFetch;
end

```

```

SExecuterR: begin
    ALUSrcA = 2'b10;
    ALUSrcB = 2'b00;
    ALUOp = 2'b10;
    PCWrite = 0;
    IRWrite = 0;
    RegWrite = 0;
    MemWrite = 0;
    nextstate = SALUWB;
end
SExecuteI: begin
    ALUSrcA = 2'b10;
    ALUSrcB = 2'b01;
    ALUOp = 2'b10;
    PCWrite = 0;
    IRWrite = 0;
    RegWrite = 0;
    MemWrite = 0;
    nextstate = SALUWB;
end
SJAL: begin
    ALUSrcA = 2'b01;
    ALUSrcB = 2'b10;
    ALUOp = 2'b00;
    ResultSrc = 2'b00;
    PCWrite = 1;
    IRWrite = 0;
    RegWrite = 0;
    MemWrite = 0;
    nextstate = SALUWB;
end
SBEQ: begin
    ALUSrcA = 2'b10;
    ALUSrcB = 2'b00;
    ALUOp = 2'b01;
    ResultSrc = 2'b00;
    PCWrite = Zero;
    IRWrite = 0;
    RegWrite = 0;
    MemWrite = 0;
    nextstate = SFetch;
end

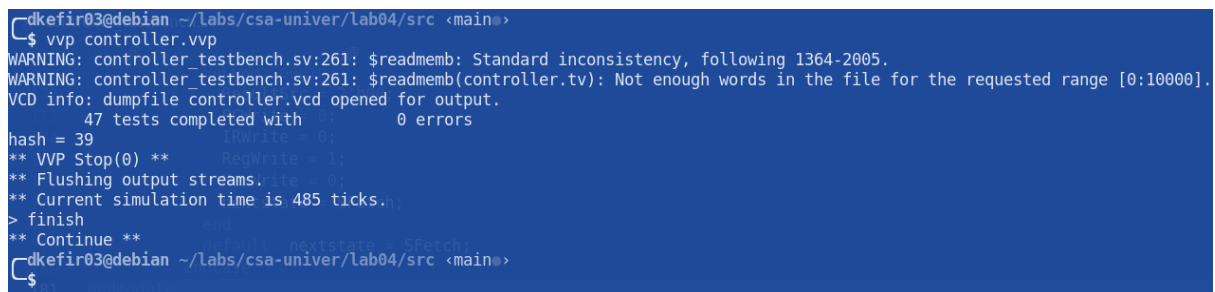
```

```

        SALUWB: begin
            ResultSrc = 2'b00;
            PCWrite = 0;
            IRWrite = 0;
            RegWrite = 1;
            MemWrite = 0;
            nextstate = SFetch;
        end
        default: nextstate = SFetch;
    endcase
endmodule

```

Вывод программы:



```

dkefir03@debian ~/labs/csa-univer/lab04/src <main>
$ vvp controller.vvp
WARNING: controller_testbench.sv:261: $readmemb: Standard inconsistency, following 1364-2005.
WARNING: controller_testbench.sv:261: $readmemb(controller.tv): Not enough words in the file for the requested range [0:10000].
VCD info: dumpfile controller.vcd opened for output.
47 tests completed with 0 errors
hash = 39
** VVP Stop(0) **
** Flushing output streams...
** Current simulation time is 485 ticks.
> finish
** Continue **
dkefir03@debian ~/labs/csa-univer/lab04/src <main>
$

```

### Часть 3. Многотактный процессор RISC-V

На [изобр. 3.1](#) показан полный многотактный процессор.

На [изобр. 3.2](#) (в конце лабораторной работы) показана иерархия высокого уровня одноктактного процессора, включая связи между контроллером, трактом данных, памятью инструкций и памятью данных. Ваш многотактный процессор имеет только одну унифицированную память и немного другие сигналы управления, поэтому вам нужно будет изменить эти соединения. Нарисуйте схему, подобную [изобр. 2](#), на которой показаны контроллер, тракт данных и модули памяти. Нарисуйте блок для модуля `riscv`, который должен охватывать контроллер и тракт данных. Пометьте сигналы, проходящие между блоками.

Напишите иерархическое описание процессора на **SystemVerilog**. Объявления некоторых модулей можете найти в `riscvmulti.sv`. Сигналы памяти топ-модуля выведены в целях тестирования. Используйте ваш контроллер из лабораторной работы 10 и любые общие строительные блоки **Verilog**, которые вам нужны (например, `mux`'ы, `flor`'ы, сумматоры, АЛУ, регистровый файл, немедленный расширитель и т.д.) из одноктактного процессора.

## РЕШЕНИЕ:

Примеры модулей из файла `riscmulti.sv`:

top:

```
]module top(input logic      clk, reset,
            output logic [31:0] WriteData, DataAdr,
            output logic      MemWrite);
-
    logic [31:0] ReadData;
    // instantiate processor and memories
] riscmulti rvmulti(clk, reset, MemWrite, DataAdr,
-
                    WriteData, ReadData);
    mem mem(clk, MemWrite, DataAdr, WriteData, ReadData);
endmodule
```

riscmulti:

```
]module riscmulti(input logic      clk, reset,
                  output logic      MemWrite,
                  output logic [31:0] Adr, WriteData,
                  input  logic [31:0] ReadData);
-
    logic      ALUSrc, Jump, Zero, AdrSrc, IRWrite, PCWrite, RegWrite;
    logic [1:0] ResultSrc, ImmSrc, ALUSrcA, ALUSrcB;
    logic [2:0] ALUControl;

    logic [31:0] ALUResult, Instr;

] controller c (clk, reset, Instr[6:0], Instr[14:12], Instr[30], Zero,
               ImmSrc, ALUSrcA, ALUSrcB, ResultSrc, AdrSrc, ALUControl,
               IRWrite, PCWrite, RegWrite, MemWrite);
-
] datapath d (clk, reset, ResultSrc, PCWrite, ALUSrc, RegWrite, IRWrite,
             AdrSrc, ImmSrc, ALUSrcA, ALUSrcB, ALUControl, Zero, Adr, Instr,
             ReadData, ALUResult, WriteData);
-
endmodule
```

datapath:

```

module datapath(input logic clk, reset,
               input logic [1:0] ResultSrc,
               input logic PCWrite, ALUSrc,
               input logic RegWrite, IRWrite, AdrSrc,
               input logic [1:0] ImmSrc, ALUSrcA, ALUSrcB,
               input logic [2:0] ALUControl,

               output logic Zero,
               output logic [31:0] PC, CurrentInstr,
               input logic [31:0] Instr,
               output logic [31:0] ALUResult, WriteData);

    logic [31:0] tmpPC, OldPC; //CurrentInstr; //PCPlus4, PCTarget;
    logic [31:0] ImmExt, Data;
    logic [31:0] SrcA, SrcB, tmpSrcA, A;
    logic [31:0] Result, ALUOut, PCORData, tmpWriteData;
    // logic [31:0] tmp;

    dfflipflop2 #(32) ir(clk, reset, IRWrite, Instr, tmpPC, CurrentInstr, OldPC);
    dfflipflop #(32) data(clk, reset, 1'b1, Instr, Data);

    mux2 #(32) adrsrcmux(tmpPC, Result, AdrSrc, PC);
} regfile      rf(clk, RegWrite, CurrentInstr[19:15], CurrentInstr[24:20],
    extend      CurrentInstr[11:7], Result, tmpSrcA, tmpWriteData);
    ext(CurrentInstr[31:7], ImmSrc, ImmExt);

    dfflipflop2 #(32) regf(clk, reset, 1'b1, tmpSrcA, tmpWriteData, A, WriteData);

    mux3 #(32) srcamux(tmpPC, OldPC, A, ALUSrcA, SrcA);
    mux3 #(32) srcbmux(tmpWriteData, ImmExt, 32'd4, ALUSrcB, SrcB);
    alu      alu(SrcA, SrcB, ALUControl, ALUResult, Zero);

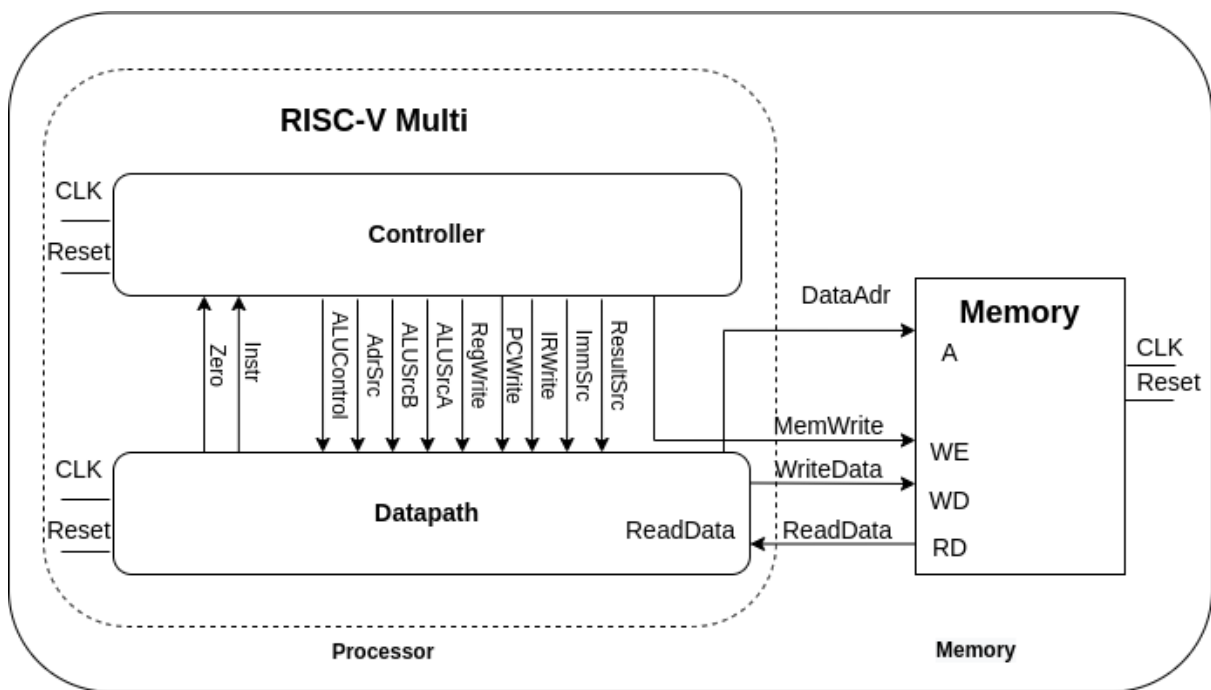
    dfflipflop #(32) aluresf(clk, reset, 1'b1, ALUResult, ALUOut);

    mux3 #(32) aluresmux(ALUOut, Data, ALUResult, ResultSrc, Result);

    dfflipflop #(32) pc(clk, reset, PCWrite, Result, tmpPC);
endmodule

```

Полученная диаграмма иерархии:



Тестовые данные:

#	RISC-V Assembly	Description	Address	Machine Code
main:	addi x2, x0, 5	# x2 = 5	0	00500113
	addi x3, x0, 12	# x3 = 12	4	00C00193
	addi x7, x3, -9	# x7 = (12 - 9) = 3	8	FF718393
	or x4, x7, x2	# x4 = (3 OR 5) = 7	C	0023E233
	and x5, x3, x4	# x5 = (12 AND 7) = 4	10	0041F2B3
	add x5, x5, x4	# x5 = (4 + 7) = 11	14	004282B3
	beq x5, x7, end	# shouldn't be taken	18	02728863
	slt x4, x3, x4	# x4 = (12 < 7) = 0	1C	0041A233
	beq x4, x0, around	# should be taken	20	00020463
	addi x5, x0, 0	# shouldn't happen	24	00000293
around:	slt x4, x7, x2	# x4 = (3 < 5) = 1	28	0023A233
	add x7, x4, x5	# x7 = (1 + 11) = 12	2C	005203B3
	sub x7, x7, x2	# x7 = (12 - 5) = 7	30	402383B3
	addi x7, x7, 8			
	addi x7, x7, -8			
	sw x7, 84(x3)	# [96] = 7	34	0471AA23
	lw x2, 96(x0)	# x2 = [96] = 7	38	06002103
	add x9, x2, x5	# x9 = (7 + 11) = 18	3C	005104B3
	jal x3, end	# jump to end, x3 = 0x44	40	008001EF
	addi x2, x0, 1	# shouldn't happen	44	00100113
end:	add x2, x2, x9	# x2 = (7 + 18) = 25	48	00910133
	sw x2, 0x20(x3)	# mem[100] = 25	4C	0221A023
done:	beq x2, x2, done	# infinite loop	50	00210063

## Результат работы программы:

```
cdkefir03@debian ~/labs/csa-univer/lab04/src <main>
$ vvp rmulti.vvp
WARNING: riscvmulti.sv:36: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: riscvmulti.sv:36: Excess hex digits (2 of '0x00500113') while reading 32-bit words.
WARNING: riscvmulti.sv:36: $readmemh(riscvtest.txt): Not enough words in the file for the requested range [0:63].
VCD info: dumpfile rmulti.vcd opened for output.
Simulation succeeded
hash = 00000000
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 725 ticks.
> finish
** Continue **
cdkefir03@debian ~/labs/csa-univer/lab04/src <main>
$
```

## Полученные осциллограммы симуляции:



## Выводы

В ходе лабораторной работы №5 мною была добавлена поддержка инструкции RISC-V *lui* для базового одноктактного процессора RISC-V на языке SystemVerilog. Был разработан контроллер для многотактного процессора, работающий по принципу конечного автомата. Был написан и протестирован многотактный процессор RISC-V. Все сделанные задания были протестированы с помощью тестов и тестовых векторов, а также построены соответствующие осциллограммы симуляций. Цели работы можно считать достигнутыми.