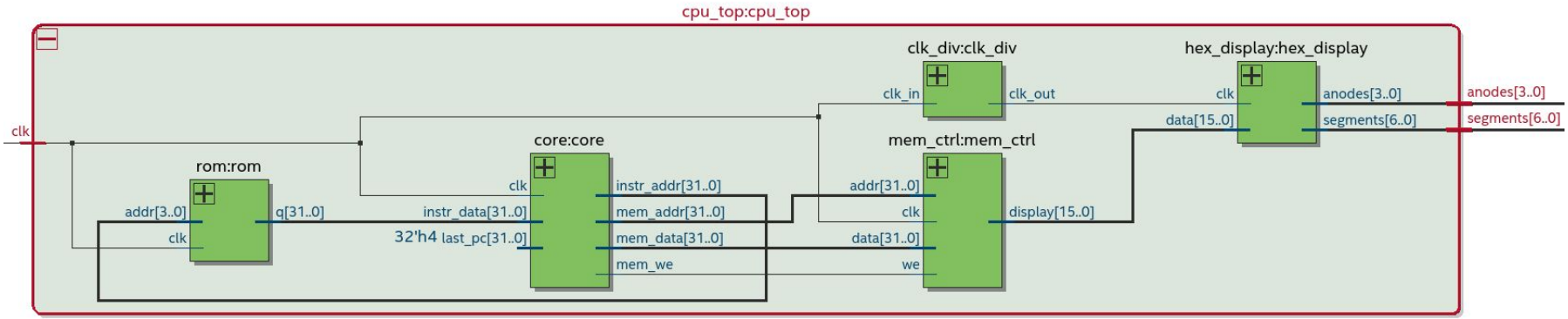
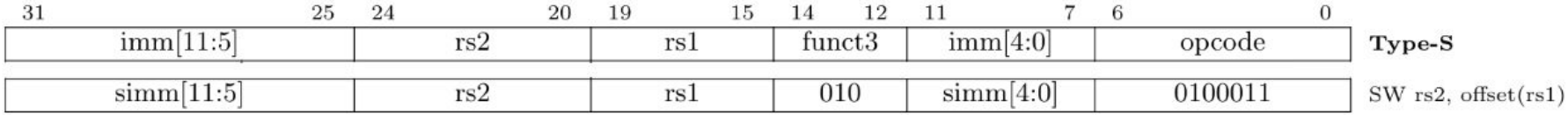
Ввод-вывод. MMIO

PMIO и MMIO

- Ввод-вывод через порты (Port-mapped I/O)
  - I/O осуществляется специальными инструкциями
  - Пример: `inb`, `outb`, `inw`, `outw`, и т.д. в x86
- Ввод-вывод через память (Memory-mapped I/O)
  - I/O осуществляется инструкциями работы с памятью



Инструкция SW



SW сохраняет значение 32-битного слова из регистра rs2 в память по адресу rs1 + offset, где offset — знаковое 12-битное число (расширенное до 32 бит). Адрес должен быть выровнен на границу слова.

- Инструкция определяется полями `funct3` и `opcode`
- Поле `imm` содержит непосредственное значение операнда
- Поля `rs1` и `rs2` содержат номера регистров-источников

Машинный код	Код на языке ассемблера RISC-V	Псевдокод
`01800593`	`li x11, 0x42`	`x11 <- 0x18`
`01000393`	`li x7, 0x10`	`x7 <- 0x10`
`00b3a823`	`sw x11, 0x10(x7)`	`[x7+0x10] <- x11`

Контроллер памяти и ввода-вывода

```
module mem_ctrl(
    input clk,
    input [31:0]addr,
    input [31:0]data,
    input we,

    output reg [15:0]display = 16'b0
);

always @(posedge clk) begin
    if (we) begin
        $display("[%h] <- %h", addr, data);
        if (addr == 32'h20)
            display <= data[15:0];
    end
end

endmodule
```

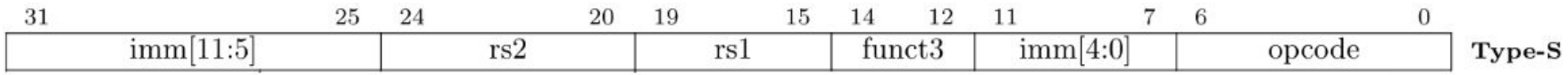
Однотактное устройство управления v2

```
module control(
    input [31:0]instr,

    output reg [11:0]imm12,
    output reg rf_we,
    output reg alu_op,
    output reg mem_we
);

wire [6:0]opcode = instr[6:0];
wire [2:0]funct3 = instr[14:12];

always @(*) begin
    rf_we = 1'b0;
    alu_op = 1'b0;
    imm12 = 12'b0;
    mem_we = 1'b0;

    ............

endmodule
```

```
casez ({funct3, opcode})
    10'b000_0010011: begin // ADDI
        rf_we = 1'b1;
        alu_op = 1'b1;
        imm12 = instr[31:20];
    end
    default: ;
endcase
```

Из первой версии

```
casez ({funct3, opcode})
    10'b000_0010011: begin // ADDI
        rf_we = 1'b1;
        alu_op = 1'b1;
        imm12 = instr[31:20];
    end
    10'b010_0100011: begin // SW
        alu_op = 1'b1;
        imm12 = {instr[31:25], instr[11:7]};
        mem_we = 1'b1;
    end
    default: ;
endcase
```



Однотактное ядро v2

```
module core(
    input [31:0]instr_data,
    input [31:0]last_pc,

    output [31:0]instr_addr
    output [31:0]instr_addr,
    output [31:0]mem_addr,
    output [31:0]mem_data,
    output mem_we
);

.................

wire [4:0]rd = instr[11:7];
wire [4:0]rs1 = instr[19:15];
wire [4:0]rs2 = instr[24:20];

wire [31:0]rf_rdata0;
wire [4:0]rf_raddr0 = rs1;

wire [31:0]rf_rdata1;
wire [4:0]rf_raddr1 = rs2;

wire [31:0]rf_wdata = alu_result;
wire [4:0]rf_waddr = rd;
```

```
wire rf_we;

assign mem_addr = alu_result;
assign mem_data = rf_rdata1;

.................

reg_file rf(
    .clk(clk),
    .raddr0(rf_raddr0), .rdata0(rf_rdata0),
    .raddr1(rf_raddr1), .rdata1(rf_rdata1),
    .waddr(rf_waddr), .wdata(rf_wdata), .we(rf_we)
);

.................

control control(
    .instr(instr),
    .imm12(imm12),
    .rf_we(rf_we),
    .alu_op(alu_op),
    .mem_we(mem_we)
);

endmodule
```

Тестирование

```
.text
.globl _start
.globl _finish

_start:
    li x11, 0x42
    li  x7, 0x10
    sw x11, 0x10(x7)

_finish:
     nop
```

```
$ whisper -l --isa=i sw.elf
#1 0 00010054 04200593 r 0b         00000042  addi     x11, x0, 0x42
#2 0 00010058 01000393 r 07         00000010  addi     x7,  x0, 0x10
#3 0 0001005c 00b3a823 m 00000020   00000042  sw       x11, 0x10(x7)
```

GitHub

github.com/viktor-prutyanov/drec-fpga-intro