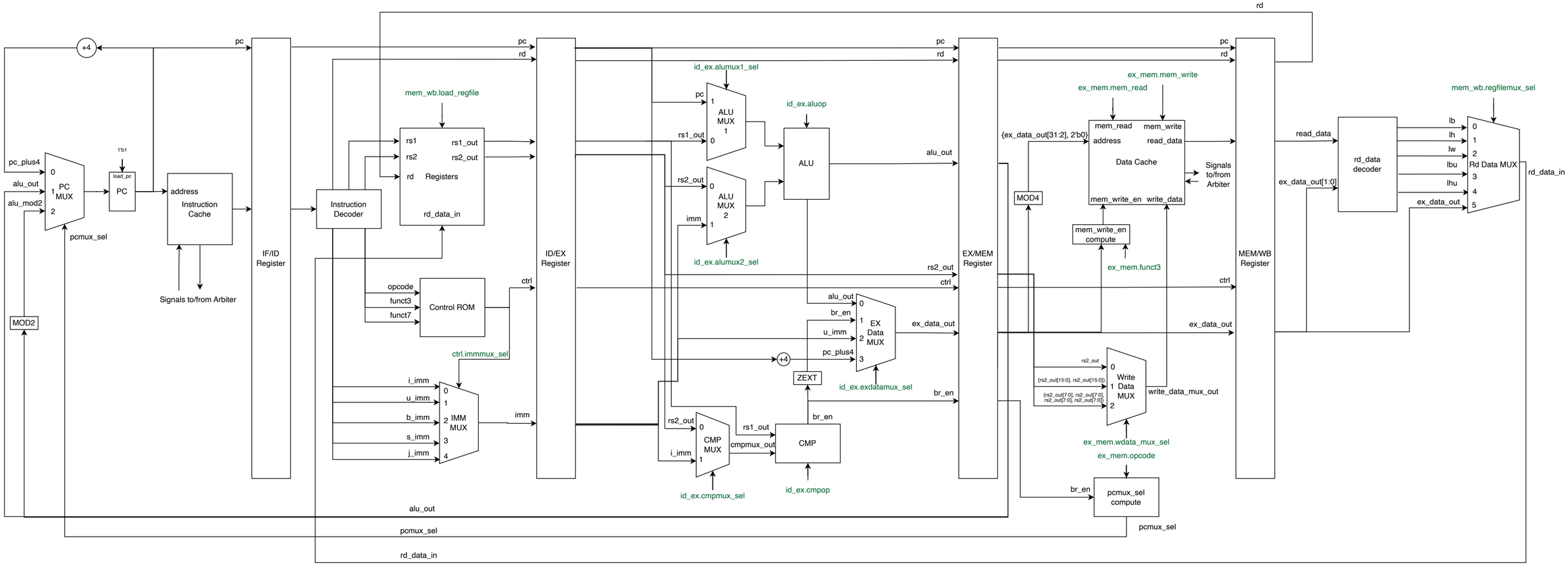
ECE 411 MP4
CP0 Design Document

Neha Agarwal
Yu Li
Mitchell Bifeld
TA: Yian Wang

March 10th, 2023


+4
pc
pc_plus4
alu_out
alu_mod2
0
1
2
PC MUX
pcmux_sel
1'b1
load_pc
PC
address
Instruction Cache
Signals to/from Arbiter
MOD2
IF/ID Register
pc
rd
Instruction Decoder
mem_wb.load_regfile
rs1
rs2
rd
Registers
rs1_out
rs2_out
rd_data_in
opcode
funct3
funct7
Control ROM
ctrl
ctrl.immmux_sel
i_imm
u_imm
b_imm
s_imm
j_imm
IMM MUX
imm
ID/EX Register
id_ex.alumux1_sel
ALU MUX 1
id_ex.aluop
ALU
alu_out
id_ex.alumux2_sel
ALU MUX 2
rs2_out
br_en
u_imm
pc_plus4
EX Data MUX
ex_data_out
id_ex.exdatamux_sel
ZEXT
CMP MUX
cmpmux_out
rs1_out
CMP
id_ex.cmpop
id_ex.cmpmux_sel
EX/MEM Register
MOD4
{ex_data_out[31:2], 2'b0}
ex_mem.mem_read
ex_mem.mem_write
mem_read
mem_write
address
read_data
Data Cache
mem_write_en
write_data
Signals to/from Arbiter
mem_write_en compute
ex_mem.funct3
Write Data MUX
write_data_mux_out
{rs2_out[15:0], rs2_out[15:0]}
{rs2_out[7:0], rs2_out[7:0], rs2_out[7:0], rs2_out[7:0]}
ex_mem.wdata_mux_sel
ex_mem.opcode
pcmux_sel compute
MEM/WB Register
read_data
ex_data_out[1:0]
rd_data decoder
mem_wb.regfilemux_sel
lb
lh
lw
lbu
lhu
Rd Data MUX
rd_data_in
ex_data_out
alu_out
pcmux_sel
rd_data_in

Controller Datapath Addendum:

Following is pseudo code for the non-standard blocks in the datapath which were not expanded in the datapath for clarity.

pcmux_sel_compute:

```
if ((op_br & br_en) | op_jal)
      pcmux_sel = pcmux::alu_out;
else if (op_jalr)
      pcmux_sel = pcmux::aluout_mod2;
else
      pcmux_sel = pcmux::pc_plus4;
```

rd_data_encoder:

```
lw = read_data
lh = sext32(addr[1] ? read_data[31:16] : read_data[15:0]);
lhu = zext32(addr[1] ? read_data[31:16] : read_data[15:0]);
lb = sext32(addr[0] ? lh[15:8] : lh[7:0]);
lbu = zext32(addr[0] ? lh[15:8] : lh[7:0]);
```

mem_write_en_compute:

```
case(funct3)
      3'b000: mem_write_en = (4'b0001 << ex_data_out[1:0]);
      3'b001: mem_write_en = (4'b0011 << ex_data_out[1:0]);
      3'b010: mem_write_en = 4'b1111;
      defaults: mem_write_en = 4'b0000;
endcase
```

Controller Description:

There is no state machine for the pipeline controller as this is a single cycle design. However, the control unit does output various signals to control the pipeline. As the implementation of the controller is simply setting the control signals based on the opcode, funct3, and funct7 of the current instruction, the entire controller implementation is provided.

```
import rv32i_types::*;

module control_rom
(
    input rv32i_opcode opcode,
    input [2:0] funct3,
    input [6:0] funct7,
    output rv32i_control_word ctrl
);

always_comb
begin
    /* Default assignments */
    ctrl.opcode = opcode;
    crtl.funct3 = funct3;
    ctrl.immmux_sel = immmux::u_imm;
    ctrl.load_regfile = 1'b0;
    ctrl.aluop = alu_ops'(funct3);
    ctrl.cmpop = cmp_ops'(funct3);
    ctrl.pcmux_sel = pcmux::pc_plus4;
    ctrl.alumux1_sel = alumux1::rs1_out;
    ctrl.alumux2_sel = alumux2::rs2_out;
    ctrl.cmpmux_sel = cmpmux::rs2_out;
    ctrl.exdatamux_sel = exdatamux::alu_out;
    ctrl.regfilemux_sel = regfilemux::mem_data;
    ctrl.writedatamux_sel = writedatamux::word;
    ctrl.mem_read = 1'b0;
    ctrl.mem_write = 1'b0;

    /* Assign control signals based on opcode */
    case(opcode)
        op_auipc: begin
            ctrl.aluop = alu_add;
            ctrl.alumux1_sel = alumux1::pc_out;
            ctrl.alumux2_sel = alumux2::imm;//u_imm
            ctrl.load_regfile = 1'b1;
            ctrl.immmux_sel = immmux::u_imm;
            ctrl.regfilemux_sel = regfilemux::ex_data_out; // Main computation result of EX stage
            ctrl.exdatamux_sel = exdatamux::alu_out;
        end
        op_lui: begin
            ctrl.load_regfile = 1'b1;
            ctrl.regfilemux_sel = regfilemux::ex_data_out;
            ctrl.immmux_sel = immmux::u_imm;
            ctrl.exdatamux_sel = exdatamux::u_imm;
        end
        op_br: begin
            ctrl.cmpop = funct3;
            ctrl.alumux1_sel = alumux1::pc_out;
            ctrl.alumux2_sel = alumux2::imm;//b_imm
            ctrl.alu_ops = alu_add;
            ctrl.immmux_sel = immmux::b_imm;
            ctrl.cmpmux_sel = cmpmux::rs2_out;
        end
        op_load: begin
            ctrl.alu_ops = alu_add;
            ctrl.mem_read = 1'b1;
            ctrl.alumux1_sel = alumux1::rs1_out;
            ctrl.alumux2_sel = alumux2::imm;
            ctrl.exdatamux_sel = exdatamux::alu_out;
            ctrl.load_regfile = 1'b1;
            ctrl.immmux_sel = immmux::i_imm;
            unique case(funct3)
                lb:  ctrl.regfilemux_sel = regfilemux::lb;
                lh:  ctrl.regfilemux_sel = regfilemux::lh;
                lbu: ctrl.regfilemux_sel = regfilemux::lbu;
                lhu: ctrl.regfilemux_sel = regfilemux::lhu;
                lw:  ctrl.regfilemux_sel = regfilemux::lw;
                default: ctrl.regfilemux_sel = regfilemux::ex_data_out;
            endcase
        end
        op_store: begin
            ctrl.mem_write = 1'b1;
            ctrl.alumux1_sel = alumux1::rs1_out;
            ctrl.alumux2_sel = alumux2::imm;
```

```
            ctrl.aluop = alu_add;
            ctrl.exdatamux_sel = exdatamux::alu_out;
            ctrl.immmux_sel = immmux::s_imm;
            case(funct3)
                sb: ctrl.writedatamux_sel = writedatamux::byte;
                sh: ctrl.writedatamux_sel = writedatamux::half;
                sw: ctrl.writedatamux_sel = writedatamux::word;
                default: ctrl.writedatamux_sel = writedatamux::byte;
            endcase
        end
        op_imm: begin
            ctrl.load_regfile = 1'b1;
            ctrl.regfilemux_sel = regfilemux::ex_data_out;
            ctrl.immmux_sel = immmux::i_imm;
            ctrl.alumux1_sel = alumux1::rs1_out;
            ctrl.alumux2_sel = alumux2::imm;
            if(funct3 == slt) begin
                ctrl.cmpmux_sel = cmpmux::i_imm;
                ctrl.cmpop = lt;
                ctrl.exdatamux_sel = exdatamux::br_en;
            end
            else if (funct3 == sltu) begin
                ctrl.cmpmux_sel = cmpmux::i_imm;
                ctrl.cmpop = ltu;
                ctrl.exdatamux_sel = exdatamux::br_en;
            end
            else if (funct3 == sr) begin
                ctrl.exdatamux_sel = exdatamux::alu_out;
                if(funct7[5]) ctrl.aluop = alu_sra;
                else          ctrl.aluop = alu_srl;
            end
            else begin
                ctrl.exdatamux_sel = exdatamux::alu_out;
                ctrl.aluop = alu_ops'(funct3);
            end
        end
        op_reg: begin
            ctrl.load_regfile = 1'b1;
            ctrl.regfilemux_sel = regfilemux::ex_data_out;
            ctrl.alumux1_sel  = alumux1::rs1_out;
            ctrl.alumux2_sel  = alumux2::rs2_out;
            if(funct3 == slt)begin
                ctrl.cmpmux_sel = cmpmux::rs2_out;
                ctrl.cmpop = lt;
                ctrl.exdatamux_sel = exdatamux::br_en;
            end
            else if (funct3 == sltu)begin
                ctrl.cmpmux_sel = cmpmux::rs2_out;
                ctrl.cmpop = ltu;
                ctrl.exdatamux_sel = exdatamux::br_en;
            end
            else if (funct3 == sr)begin
                ctrl.exdatamux_sel = exdatamux::alu_out;
                if(funct7[5]) ctrl.aluop = alu_sra;
                else          ctrl.aluop = alu_srl;
            end
            else if (funct3 == add) begin
                ctrl.exdatamux_sel = exdatamux::alu_out;
                if(funct7[5]) ctrl.aluop = alu_sub;
                else          ctrl.aluop = alu_add;
            end
            else begin
                ctrl.exdatamux_sel = exdatamux::alu_out;
                ctrl.aluop = alu_ops'(funct3);
            end
        end
        op_jal: begin
            ctrl.load_regfile = 1'b1;
            ctrl.regfilemux_sel = regfilemux::ex_data_out;
            ctrl.exdatamux_sel = exdatamux::pc_plus4;
            ctrl.alumux1_sel = alumux1::pc_out;
            ctrl.alumux2_sel = alumux2::imm;
            ctrl.aluop = alu_add;
            ctrl.immmux_sel = immmux::j_imm;
        end
```

```
        op_jalr: begin
            ctrl.load_regfile = 1'b1;
            ctrl.regfilemux_sel = regfilemux::ex_data_out;
            ctrl.exdatamux_sel = exdatamux::pc_plus4;
            ctrl.alumux1_sel = alumux1::rs1_out;
            ctrl.alumux2_sel = alumux2::imm;
            ctrl.aluop = alu_add;
            ctrl.immmux_sel = immmux::i_imm;
        end

        default: begin
            ctrl = 0;   /* Unknown opcode, set control word to zero */
        end
    endcase
end
endmodule : control_rom
```