lab6

1. fetch

```
`ifndef __FETCH_SV
`define __FETCH_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`include "pipeline/fetch/pcselect.sv"
`include "pipeline/decode/decoder.sv"
`else

`endif

module fetch
    import common::*;
    import pipes::*;(
    input u1 clk, reset,
    input u64 pcplus4,
    output u64 pc_selected,
    output fetch_data_t dataF,
    input u32 raw_instr,
    input u64 pc,
    input ibus_resp_t iresp,
    input dbus_resp_t dresp,
    output dbus_req_t dreq,
    output ibus_req_t ireq,
    output logic stallf,
    input logic stalld, stalle, stallm,
    input u1 branch,
    input u64 jump
);

    assign pc_selected = branch ? jump : pcplus4;

    // assign stallf = (~iresp.data_ok);
```

```
    // assign stallf = ireq.valid && ~iresp.data_ok;

    // always_ff @(posedge clk)
    //     if(reset)
    //         dataF <= '0;
    //     else begin
            assign dataF.pc = pc;
            assign dataF.instr = raw_instr;
            assign dataF.valid = ~stalld & ~stallm & iresp.data_ok;
        // end

endmodule



`endif
```

fetch模块主要进行pc的生成，并发送访存请求。

```
`ifndef __PCSELECT_SV
`define __PCSELECT_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`else

`endif

module pcselect
    import common::*;
    import pipes::*;(
    input u1 clk, reset,
    input u64 pcplus4,
    output u64 pc_selected

);

    assign pc_selected = pcplus4;


endmodule

`endif
```

2. decode

```
`ifndef __DECODE_SV
`define __DECODE_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`include "pipeline/decode/decoder.sv"
`include "pipeline/decode/immediate.sv"
`else

`endif

module decode
    import common::*;
    import pipes::*;(
    input u1 clk, reset,
    input fetch_data_t dataF,
    output decode_data_t dataD,
    output creg_addr_t ra1, ra2,
    input word_t rd1, rd2,
    input ibus_req_t ireq,
    input ibus_resp_t iresp,
    output logic stallf, stalld, stalle, stallm, stall_raw,
    input u5 dstD, dstE, dstM, dstW,
    output dbus_req_t dreq,
    input dbus_resp_t dresp,
    input u1 branch,
    output csr_addr_t csraddr,
    input word_t csrdata,
    input csr_addr_t csrD, csrE, csrM, csrW
);

    control_t ctl;

    // assign stall_raw = bubble;

    decoder decoder (
        .raw_instr(dataF.instr),
        .ctl(ctl)
    );
```

```
    logic iscsr = (ctl.op == CSR || ctl.op == CSRI);

    assign ra1 = dataF.instr[19:15];
    assign ra2 = dataF.instr[24:20];

    assign csraddr = dataF.instr[31:20];

    word_t temp1, temp2;
    logic bubble, bubble1, bubble2;

    assign bubble1 = ra1 != 0 && (ra1 == dstD || ra1 == dstE || ra1 == dstM);
    assign bubble2 = ra2 != 0 && (ra2 == dstD || ra2 == dstE || ra2 == dstM);
    // assign bubble3 = (iscsr && (csraddr == csrD || csraddr == csrE || csraddr == csrM));

    immediate immediate(
        .scra(rd1),
        .scrb(rd2),
        .ctl(ctl),
        .instr(dataF.instr),
        .temp1,
        .temp2,
        .bubble,
        .bubble1,
        .bubble2,
        .bubble3((csraddr == csrD || csraddr == csrE || csraddr == csrM)),
        .csrdata
    );

    assign stalld = bubble;

//         reg [63:0] prev_addr;
// always @(posedge clk) begin
//     if (addr == 64'h80028768) begin
//         $display("New PC: 0x%h", dataE.pc);
//     end
//     prev_addr <= addr;
// end

    // always_ff @(posedge clk)
    //     if(reset)
    //         stall_raw <= 0;
    //     else begin
```

```
    //         if(ctl.op == LD || ctl.op == SD)
    //             stall_raw <= 1;
    //         if(dresp.data_ok)
    //             stall_raw <= 0;

    // assign csrs = dataF.instr[31:20];

    // always_ff @(posedge clk)
    //     if(reset)
    //         dataD <= '0;
    //     else begin
            assign dataD.pc = dataF.pc;
            assign dataD.instr = dataF.instr;
            assign dataD.valid = ~stalld & ~stallm & dataF.valid;
            assign dataD.ctl = ctl;
            assign dataD.dst = dataF.instr[11:7];
            assign dataD.srca = temp1;
            assign dataD.srcb = temp2;
            assign dataD.rega = ra1;
            assign dataD.regb = ra2;
            assign dataD.store_data = rd2;
            assign dataD.csr = temp2;
            assign dataD.csrdst = dataF.instr[31:20];
        // end

endmodule


`endif
```

在decode阶段生成控制信号，并从指令中获取需要写入的寄存器和操作数。同时进行数据冒险的判断，如果当前读的寄存器与上一条指令的写的寄存器相同，则需要暂停fetch阶段，等待数据写回。若并将指令送至execute阶段。此lab添加了csr寄存器的数据冒险判断。

```
`ifndef __DECODER_SV
`define __DECODER_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`else

`endif

module decoder
    import common::*;
    import pipes::*;(
    input u32 raw_instr,
    output control_t ctl

);

    u7 f7 = raw_instr[6:0];
    u3 f3 = raw_instr[14:12];
    u7 f7_ = raw_instr[31:25];

    always_comb begin
        ctl = '0;
        unique case (f7)
            F7_ADDI: begin
                ctl.op = ALUI;
                ctl.regwrite = 1'b1;
                unique case (f3)
                    F3_ADDI: begin
                        ctl.alufunc = ADD;
                    end

                    F3_XORI: begin
                        ctl.alufunc = XOR;
                    end

                    F3_ORI: begin
                        ctl.alufunc = OR;
                    end

                    F3_ANDI: begin
                        ctl.alufunc = AND;
```

```
            end

            F3_SLTI: begin
                ctl.alufunc = SLT;
            end

            F3_SLTIU: begin
                ctl.alufunc = SLTU;
            end

            F3_SLLI: begin
                ctl.alufunc = SLL;
            end

            F3_SRLI: begin
                ctl.alufunc = raw_instr[30] ? SRA : SRL;
            end

            default: begin
                ctl.alufunc = NOTALU;
                ctl.regwrite = 1'b0;
            end
        endcase
    end

    F7_ADD: begin
        ctl.op = ALU;
        ctl.regwrite = 1'b1;
        unique case (f3)
            F3_ADD: begin
                unique case (f7_)
                    F7_ADD_: begin
                        ctl.alufunc = ADD;
                    end

                    F7_SUB_: begin
                        ctl.alufunc = SUB;
                    end

                    default: begin
                        ctl.alufunc = NOTALU;
                    end
                endcase
```

```
end

F3_AND: begin
    ctl.alufunc = AND;
end

F3_OR: begin
    ctl.alufunc = OR;
end

F3_XOR: begin
    ctl.alufunc = XOR;
end

F3_SLT: begin
    ctl.alufunc = SLT;
end

F3_SLTU: begin
    ctl.alufunc = SLTU;
end

F3_SLL: begin
    ctl.alufunc = SLL;
end

F3_SRL: begin
    unique case (f7_)
        F7_SRL_: begin
            ctl.alufunc = SRL;
        end

        F7_SRA_: begin
            ctl.alufunc = SRA;
        end

        default: begin
            ctl.alufunc = NOTALU;
        end
    endcase
end

default: begin
```

```
            ctl.alufunc = NOTALU;
        end
    endcase
end

F7_ADDIW: begin
    ctl.op = ALUIW;
    ctl.regwrite = 1'b1;
    unique case (f3)
        F3_ADDIW: begin
            ctl.alufunc = ADD;
        end

        F3_SLLIW: begin
            ctl.alufunc = SLLW;
        end

        F3_SRLIW: begin
            ctl.alufunc = raw_instr[30] ? SRAW : SRLW;
        end

        default: begin
            ctl.alufunc = NOTALU;
        end
    endcase
end

F7_ADDW: begin
    ctl.op = ALUW;
    ctl.regwrite = 1'b1;
    unique case (f3)
        F3_ADDW:
        unique case (f7_)
            F7_ADDW_: begin
                ctl.alufunc = ADD;
            end

            F7_SUBW_: begin
                ctl.alufunc = SUB;
            end

            default: begin
                ctl.alufunc = NOTALU;
```

```
            end
        endcase

        F3_SLLW: begin
            ctl.alufunc = SLLW;
        end

        F3_SRLW: begin
            unique case (f7_)
                F7_SRLW_: begin
                    ctl.alufunc = SRLW;
                end

                F7_SRAW_: begin
                    ctl.alufunc = SRAW;
                end

                default: begin
                    ctl.alufunc = NOTALU;
                end
            endcase
        end

        default: begin
            ctl.alufunc = NOTALU;
        end
    endcase
end

F7_LD: begin
    ctl.op = LD;
    ctl.regwrite = 1'b1;
    ctl.memtoreg = 1'b1;
    ctl.alufunc = ADD;
end

F7_SD: begin
    ctl.op = SD;
    ctl.regwrite = 1'b0;
    ctl.memwrite = 1'b1;
    ctl.alufunc = ADD;
end
```

```
F7_LUI: begin
    ctl.op = LUI;
    ctl.regwrite = 1'b1;
    ctl.alufunc = CPYB;
end

F7_BEQ: begin
    ctl.regwrite = 1'b0;
    ctl.op = f3[0] ? BNZ : BZ;
    unique case (f3[2:1])
        2'b00: begin
            ctl.alufunc = EQL;
        end

        2'b10: begin
            ctl.alufunc = SLT;
        end

        2'b11: begin
            ctl.alufunc = SLTU;
        end

        default: begin
            ctl.alufunc = NOTALU;
        end
    endcase
end

F7_AUIPC: begin
    ctl.op = AUIPC;
    ctl.regwrite = 1'b1;
    ctl.alufunc = AUI;
end

F7_JAL: begin
    ctl.op = JAL;
    ctl.regwrite = 1'b1;
    ctl.alufunc = ADD;
end

F7_JALR: begin
    ctl.op = JALR;
    ctl.regwrite = 1'b1;
```

```
        ctl.alufunc = ADD;
    end

    F7_CSRRC: begin
        ctl.regwrite = 1'b1;
        unique case(f3)
            F3_CSRRC: begin
                ctl.op = CSR;
                ctl.alufunc = ALU_CSRC;
            end

            F3_CSRRCI: begin
                ctl.op = CSRI;
                ctl.alufunc = ALU_CSRCI;
            end

            F3_CSRRS: begin
                ctl.op = CSR;
                ctl.alufunc = ALU_CSRS;
            end

            F3_CSRRSI: begin
                ctl.op = CSRI;
                ctl.alufunc = ALU_CSRSI;
            end

            F3_CSRRW: begin
                ctl.op = CSR;
                ctl.alufunc = ALU_CSRW;
            end

            F3_CSRRWI: begin
                ctl.op = CSRI;
                ctl.alufunc = ALU_CSRWI;
            end

            default: begin
                ctl.op = UNKNOWN;
                ctl.alufunc = NOTALU;
            end
        endcase
    end
```

```
            default: begin
                ctl.op = UNKNOWN;
                ctl.alufunc = NOTALU;
                ctl.regwrite = 1'b0;
            end
        endcase

    end


endmodule


`endif
```

获取指令的opcode和func。

```
`ifndef __IMMEDIATE_SV
`define __IMMEDIATE_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`endif

module immediate
        import common::*;
        import pipes::*;(
    input  word_t    scrb, scra,
    input  control_t ctl,
    input  u32    instr,
    output word_t temp1, temp2,
    output logic bubble,
    input logic bubble1, bubble2, bubble3,
    input word_t csrdata
);
    always_comb begin
        temp1 = scra;
        temp2 = scrb;
        unique case (ctl.op)
            ALUW: begin
                bubble = bubble1 | bubble2;
            end

            ALU: begin
                bubble = bubble1 | bubble2;
            end

            ALUI, ALUIW, LD: begin
                temp2 = {{52{instr[31]}}, instr[31:20]};
                bubble = bubble1;
            end

            LUI: begin
                temp2 = {{32{instr[31]}}, instr[31:12], 12'b0};
                bubble = bubble1;
            end

            SD: begin
                temp2 = {{52{instr[31]}}, instr[31:25], instr[11:7]};
```

```
                bubble = bubble1 | bubble2;
            end

            AUIPC: begin
                temp2 = {{32{instr[31]}}, instr[31:12], 12'b0};
                bubble = 0;
            end

            JAL: begin
                bubble = 0;
            end

            JALR: begin
                bubble = bubble1;
            end

            CSR: begin
                temp2 = csrdata;
                // temp2 = scra;
                bubble = bubble1 | bubble3;
            end

            CSRI: begin
                temp1 = csrdata;
                temp2 = {59'b0, instr[19:15]};
                bubble = bubble3;
            end

            default: begin
                bubble = bubble1 | bubble2;
            end
        endcase
    end
endmodule

`endif
```

获取立即数。并辅助进行数据冒险的判断。

3. execute

```
`ifndef __EXCUTE_SV
`define __EXCUTE_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`include "pipeline/execute/alu.sv"
`endif

module execute
    import common::*;
        import pipes::*;(
    input  u1 clk, reset,
    input  decode_data_t dataD,
    output execute_data_t dataE,
    input logic stallf, stalld, stalle, stallm,
    output u1 branch,
    output u64 jump
);
    word_t result, csrresult;

    alu alu_inst(
        .clk(clk),
        .reset(reset),
        .srca(dataD.srca),
        .srcb(dataD.srcb),
        .alufunc(dataD.ctl.alufunc),
        .result(result),
        .csrres(csrresult),
        .choose(dataD.ctl.op == ALUW || dataD.ctl.op == ALUIW),
        .pc(dataD.pc)
    );

    assign branch = ((dataD.ctl.op == BZ && result == 1) || (dataD.ctl.op == BNZ && result == 0

    assign jump =   dataD.ctl.op == JAL                 ? dataD.pc + {{44{dataD.instr[31]}}, da
                  : dataD.ctl.op == JALR                ? (dataD.srca + {{52{dataD.instr[31]}}
                  : (dataD.ctl.op == BZ && result == 1)  ? dataD.pc + {{52{dataD.instr[31]}}, da
                  : (dataD.ctl.op == BNZ && result == 0) ? dataD.pc + {{52{dataD.instr[31]}}, da
                  :                                        dataD.pc + 4;
```

```
    // always_ff @(posedge clk)
    //     if(reset)
    //         dataE <= '0;
    //     else begin
            assign dataE.result = (dataD.ctl.op == JAL || dataD.ctl.op == JALR) ? dataD.pc + 4
            assign dataE.ctl = dataD.ctl;
            assign dataE.dst = dataD.dst;
            assign dataE.pc = dataD.pc;
            assign dataE.instr = dataD.instr;
            assign dataE.valid = ~stallm & dataD.valid;
            assign dataE.store_data = dataD.store_data;
            assign dataE.csrres = csrresult;
            // assign dataE.csr = dataD.csr;
            // assign dataE.csrdst = dataD.csrdst;
        // end

endmodule

`endif
```

根据decode阶段的得到的opcode和func，对操作数进行运算。并判断当前指令是否进行跳转操作。

4. memory

```
`ifndef _MEMORY_SV
`define _MEMORY_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`endif

module memory
    import common::*;
    import pipes::*;(
    input  logic clk, reset,
    input  execute_data_t dataE,
    output memory_data_t dataM,
    output dbus_req_t dreq,
    input dbus_resp_t dresp,
    output logic stallf, stalld, stalle, stallm, stall
);

    msize_t size;
    strobe_t strobe;
    logic load, store;
    addr_t addr = dataE.result;

    logic req_active;
    logic new_request;

    assign load = dataE.ctl.op == LD;
    assign store = dataE.ctl.op == SD;

    assign new_request = (load | store) & dataE.valid;

    // 更新请求活跃状态（时序逻辑）
    always_ff @(posedge clk) begin
        if (reset) begin
            req_active <= 1'b0;
        end else begin
            if (dresp.data_ok) begin
                // 响应完成时清除活跃状态
                req_active <= 1'b0;
```

```
        end else if (new_request && !req_active) begin
            // 新请求且当前无活跃请求时激活
            req_active <= 1'b1;
        end
    end
end

u64 in = dataE.store_data;
u64 off = {58'b0, addr[2:0], 3'b0};

// always_ff @(posedge clk) begin
//     if(reset) dreq <= '0;
//     else if(dresp.data_ok && dresp.addr_ok) dreq <= '0;
//     else if((load | store) & ~dreq.valid) begin
//         dreq.valid <= '1;
//         dreq.addr <= dataE.result;
//         dreq.size <= size;
//         if(store) begin
//             dreq.data <= in << off;
//             case(size)
//                 MSIZE1: dreq.strobe <= 8'h01 << off;
//                 MSIZE2: dreq.strobe <= 8'h03 << off;
//                 MSIZE4: dreq.strobe <= 8'h0f << off;
//                 MSIZE8: dreq.strobe <= 8'hff << off;
//                 default: dreq.strobe <= '0;
//             endcase
//         end
//     end
// end

// 输出 valid 信号：活跃状态或新请求（组合逻辑）
assign dreq.valid = req_active || new_request;

assign dreq.addr = addr;
assign dreq.size = size;
assign dreq.strobe = store ? strobe << addr[2:0] : 0;

assign dreq.data = in << off;

// always_comb begin
//     case(size)
//         MSIZE1: begin
//             case(addr[2:0])
```

```
//                 3'b000: dreq.data = {56'b0, in[7:0]};
//                 3'b001: dreq.data = {48'b0, in[7:0], 8'b0};
//                 3'b010: dreq.data = {40'b0, in[7:0], 16'b0};
//                 3'b011: dreq.data = {32'b0, in[7:0], 24'b0};
//                 3'b100: dreq.data = {24'b0, in[7:0], 32'b0};
//                 3'b101: dreq.data = {16'b0, in[7:0], 40'b0};
//                 3'b110: dreq.data = {8'b0, in[7:0], 48'b0};
//                 3'b111: dreq.data = {in[7:0], 56'b0};
//             endcase
//         end

//         MSIZE2: begin
//             case(addr[2:0])
//                 3'b000: dreq.data = {48'b0, in[15:0]};
//                 3'b010: dreq.data = {32'b0, in[15:0], 16'b0};
//                 3'b100: dreq.data = {16'b0, in[15:0], 32'b0};
//                 3'b110: dreq.data = {in[15:0], 48'b0};
//                 default: dreq.data = {48'b0, in[15:0]};
//             endcase
//         end

//         MSIZE4: begin
//             case(addr[2:0])
//                 3'b000: dreq.data = {32'b0, in[31:0]};
//                 3'b100: dreq.data = {in[31:0], 32'b0};
//                 default: dreq.data = {32'b0, in[31:0]};
//             endcase
//         end

//         MSIZE8: begin
//             dreq.data = in;
//         end

//         default: dreq.data = in;
//     endcase
// end

// always_comb begin
//     dreq.data = dataE.store_data;
//     for (int i = 0;i < 8;i++) begin
//         dreq.data = dreq.data << addr[2:0];
//     end
// end
```

```
always_comb case (dataE.instr[13:12])
    2'b00: begin size = MSIZE1; strobe = 8'b00000001; end //sb
    2'b01: begin size = MSIZE2; strobe = 8'b00000011; end //sh
    2'b10: begin size = MSIZE4; strobe = 8'b00001111; end //sw
    2'b11: begin size = MSIZE8; strobe = 8'b11111111; end //sd
endcase

// always_comb case (dataE.instr[13:12])
//     2'b00: begin
//         size = MSIZE1;
//         case(addr[2:0])
//             3'b000: strobe = 8'b00000001;
//             3'b001: strobe = 8'b00000010;
//             3'b010: strobe = 8'b00000100;
//             3'b011: strobe = 8'b00001000;
//             3'b100: strobe = 8'b00010000;
//             3'b101: strobe = 8'b00100000;
//             3'b110: strobe = 8'b01000000;
//             3'b111: strobe = 8'b10000000;
//         endcase
//     end //sb

//     2'b01: begin
//         size = MSIZE2;
//         case(addr[2:0])
//             3'b000: strobe = 8'b00000011;
//             3'b010: strobe = 8'b00001100;
//             3'b100: strobe = 8'b00110000;
//             3'b110: strobe = 8'b11000000;
//             default: strobe = 8'b00000000;
//         endcase
//     end //sh

//     2'b10: begin
//         size = MSIZE4;
//         case(addr[2:0])
//             3'b000: strobe = 8'b00001111;
//             3'b100: strobe = 8'b11110000;
//             default: strobe = 8'b00000000;
//         endcase
//     end //sw
```

```
    //     2'b11: begin size = MSIZE8; strobe = 8'b11111111; end //sd
    // endcase

    u64 out, data;
    assign data = dresp.data >> off;

    always_comb case (dataE.instr[14:12])
        3'b000: out = {{56{data[7]}}, data[7:0]}; // lb
        3'b001: out = {{48{data[15]}}, data[15:0]}; // lh
        3'b010: out = {{32{data[31]}}, data[31:0]}; // lw
        3'b011: out = data; // ld
        3'b100: out = {{56'b0}, data[7:0]}; // lbu
        3'b101: out = {{48'b0}, data[15:0]}; // lhu
        3'b110: out = {{32'b0}, data[31:0]}; // lwu
        3'b111: out = 0; // not used
    endcase

//     reg [63:0] prev_addr;
// always @(posedge clk) begin
//     if (addr == 64'h80028768) begin
//         $display("New PC: 0x%h", dataE.pc);
//     end
//     prev_addr <= addr;
// end

    // word_t csrresult;
    // assign csrrresult = dataE.csrres;

    // always_ff @(posedge clk)
    //     if(reset)
    //         dataM <= '0;
    //     else begin
            assign dataM.result = (load | store) ? out : dataE.result;
            assign dataM.ctl = dataE.ctl;
            assign dataM.dst = dataE.dst;
            assign dataM.pc = dataE.pc;
            assign dataM.instr = dataE.instr;
            assign dataM.valid = ~stallm & dataE.valid;
            assign dataM.memaddr = (load | store) ? dataE.result : dataE.csrres;
            // assign dataM.csrres = dataE.csrres;
            // assign dataM.csrdst = dataE.csrdst;
            // assign dataM.csr = dataE.csr;
        // end
```

```
endmodule

`endif
```

若指令为load或store，则将数据从ibus或dbus中读出或写入。

5. writeback

```
`ifndef _WRITEBACK_SV
`define _WRITEBACK_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`endif

module writeback
    import common::*;
    import pipes::*;(
    input  logic clk, reset,
    input memory_data_t dataM,
    output writeback_data_t dataW,
    input logic wvalid,
    output logic [63:0] regs[31:0],
    output logic [63:0] regs_nxt[31:0]
);

    creg_addr_t wa;
    word_t wd;

    assign wa = dataM.dst;
    assign wd = dataM.result;

    always_ff @(posedge clk) begin
                regs_nxt <= regs;
                regs_nxt[0] <= '0;
        end

        for (genvar i = 1; i < 32; i++)
                always_comb
                        regs[i] = (i == wa && wvalid) ? wd : regs_nxt[i];

    // always_ff @(posedge clk)
    //     if(reset)
    //         dataW <= '0;
    //     else begin
         assign dataW.result = dataM.result;
         assign dataW.ctl = dataM.ctl;
```

```
            assign dataW.dst = dataM.dst;
            assign dataW.pc = dataM.pc;
            assign dataW.instr = dataM.instr;
            assign dataW.valid = dataM.valid;
            assign dataW.memaddr = dataM.memaddr;
            // assign dataW.csrres = dataM.csrres;
            // assign dataW.csr = dataM.csr;
            // assign dataW.csrdst = dataM.csrdst;
        // end

endmodule

`endif
```

将运算结果写入指定寄存器。

```
`ifndef __REGFILE_SV
`define __REGFILE_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`endif

module regfile
        import common::*;
    import pipes::*;(
        input logic clk, reset,
        input creg_addr_t ra1, ra2,
        output word_t rd1, rd2
);
        logic [63:0] regs[31:0], regs_nxt[31:0];

        assign rd1 = (ra1 != 0) ? regs[ra1] : 0;
        assign rd2 = (ra2 != 0) ? regs[ra2] : 0;

endmodule

`endif
```

```
``ifndef __CSRFILE_SV
`define __CSRFILE_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`include "include/csr_pkg.sv"
`else

`endif

module csrfile
        import common::*;
        import pipes::*;
    import csr_pkg::*;(
        input logic clk, reset,
    input u12 csr_ra,
    output word_t csr_out,
    input logic csr_wen,
    input u12 csr_wa,
    input word_t csrresult,
        input writeback_data_t dataW,
        output logic flushall,
        output u64 csrpc,
        input logic trint, swint, exint
);

csr_regs_t csrs, csrs_nxt;
word_t csr_wd;

logic error = dataW.error != NOERROR;
logic is_ecall = dataW.ctl.op == ECALL;
logic interrupt = (mode != 2'd3 || csrs.mstatus.mie) && ((trint && csrs.mie[7]) || (swint && csı

u2 mode, mode_nxt;

always_comb begin
        unique case(csr_ra)
                CSR_MIE: begin csr_out = csrs_nxt.mie; end
                CSR_MIP: begin csr_out = csrs_nxt.mip; end
                CSR_MTVEC: begin csr_out = csrs_nxt.mtvec; end
                CSR_MSTATUS: begin csr_out = csrs_nxt.mstatus; end
                CSR_MSCRATCH: begin csr_out = csrs_nxt.mscratch; end
```

```
                CSR_MEPC: begin csr_out = csrs_nxt.mepc; end
                CSR_SATP: begin csr_out = csrs_nxt.satp; end
                CSR_MCAUSE: begin csr_out = csrs_nxt.mcause; end
                CSR_MCYCLE: begin csr_out = csrs_nxt.mcycle; end
                CSR_MTVAL: begin csr_out = csrs_nxt.mtval; end
                CSR_SSTATUS: begin csr_out = csrs_nxt.mstatus & SSTATUS_MASK; end
                default: csr_out = '0;
        endcase
end

// assign flushall = 0;
// assign flushall = (dataW.ctl.op == MRET);
assign flushall = dataW.ctl.op == MRET;
// assign csrpc = '0;

// assign mode_nxt = 2'd3;

always_comb begin
        csrpc = '0;
        // flushall = 0;
        mode_nxt = mode;
        csrs_nxt = csrs;
        csrs_nxt.mcycle = csrs.mcycle + 1;
        // if(dataW.valid && dataW.ctl.op == ECALL) begin
        //      csrs_nxt.mepc = dataW.pc;
        //      csrs_nxt.mcause[63:0] = 64'b0;
        //      if(mode == 2'b0) csrs_nxt.mcause[62:0] = 63'd8;
        //      else if(mode == 2'd3) csrs_nxt.mcause[62:0] = 63'd11;
        //      csrs_nxt.mstatus.mpie = csrs.mstatus.mie;
        //      csrs_nxt.mstatus.mie = '0;
        //      csrs_nxt.mstatus.mpp = mode;
        //      mode_nxt = 2'd3;
        // end
        if((dataW.valid && (is_ecall))) begin
                csrs_nxt.mepc = dataW.pc;
                csrs_nxt.mcause[63:0] = 64'b0;
                unique case(dataW.error)
                        INSTR_MISALIGN: csrs_nxt.mcause[62:0] = 63'd0;
                        EDECODE: csrs_nxt.mcause[62:0] = 63'd2;
                        LOAD_MISALIGN: csrs_nxt.mcause[62:0] = 63'd4;
                        STORE_MISALIGN: csrs_nxt.mcause[62:0] = 63'd6;
                        default: begin
                                if(mode == 2'b0) csrs_nxt.mcause[62:0] = 63'd8;
```

```
                        else if(mode == 2'd3) csrs_nxt.mcause[62:0] = 63'd11;
                end
            endcase
            csrs_nxt.mstatus.mpie = csrs.mstatus.mie;
            csrs_nxt.mstatus.mie = '0;
            csrs_nxt.mstatus.mpp = mode;
            mode_nxt = 2'd3;
    end
    // if (dataW.valid && interrupt) begin
    //      mode_nxt = 2'd3;
//      // csrpc = csrs.mtvec;
//      csrs_nxt.mepc = dataW.pc;
//      csrs_nxt.mcause[62:0] = 63'b0;
//      csrs_nxt.mcause[63] = 1'b1;
//      if (trint) csrs_nxt.mcause[62:0] = 63'd7;
//      else if (swint) csrs_nxt.mcause[62:0] = 63'd3;
//      else if (exint) csrs_nxt.mcause[62:0] = 63'd11;
//      csrs_nxt.mstatus.mpie = csrs.mstatus.mie;
//      csrs_nxt.mstatus.mie = '0;
//      csrs_nxt.mstatus.mpp = mode;
    // end
    else if (csr_wen) begin
            // flushall = 1;
            // csrpc = dataW.pc + 4;
            csr_wd = csrresult;
            // unique case(dataW.ctl.alufunc)
            //      ALU_CSRW: csr_wd = csrresult;
            //      ALU_CSRS: csr_wd = csrresult;
            //      ALU_CSRC: csr_wd = csrresult;
            //      default: ;
            // endcase
            unique case(csr_wa)
                    CSR_MIE: csrs_nxt.mie = csr_wd;
                    CSR_MIP: csrs_nxt.mip = csr_wd & MIP_MASK;
                    CSR_MTVEC: csrs_nxt.mtvec = csr_wd & MTVEC_MASK;
                    CSR_MSTATUS: csrs_nxt.mstatus = csr_wd & MSTATUS_MASK;
                    CSR_MSCRATCH: csrs_nxt.mscratch = csr_wd;
                    CSR_MEPC: csrs_nxt.mepc = csr_wd;
                    CSR_SATP: csrs_nxt.satp = csr_wd;
                    CSR_MCAUSE: csrs_nxt.mcause = csr_wd;
                    CSR_MCYCLE: csrs_nxt.mcycle = csr_wd;
                    CSR_MTVAL: csrs_nxt.mtval = csr_wd;
                    CSR_SSTATUS: csrs_nxt.mstatus = csr_wd & SSTATUS_MASK;
```

```
                        default: ;
                endcase
        end
        else if (dataW.valid && flushall) begin
                // flushall = 1;
                // csrpc = csrs_nxt.mepc;
                csrs_nxt.mstatus.mie = csrs.mstatus.mpie;
                csrs_nxt.mstatus.mpie = 1'b1;
                mode_nxt = csrs.mstatus.mpp;
                csrs_nxt.mstatus.mpp = 2'b0;
                csrs_nxt.mstatus.xs = '0;
        end
        else begin
                // flushall = 0;
        end
end

always_ff @(posedge clk or posedge reset) begin
        if (reset) begin
                csrs <= '0;
                mode <= 2'd3;
                csrs.mcause[1] <= 1'b1;
        csrs.mepc[31]  <= 1'b1;
        end else begin
                csrs <= csrs_nxt;
                mode <= mode_nxt;
        end
end


endmodule

`endif
```

7. pipes.sv

```
`ifndef __PIPES_SV
`define __PIPES_SV

`ifdef VERILATOR
`include "include/common.sv"
`endif

package pipes;
    import common::*;


    // 定义指令解码规则
    parameter F7_ADDI = 7'b0010011;
    parameter F3_ADDI = 3'b000;

    parameter F7_XORI = 7'b0010011;
    parameter F3_XORI = 3'b100;

    parameter F7_ORI = 7'b0010011;
    parameter F3_ORI = 3'b110;

    parameter F7_ANDI = 7'b0010011;
    parameter F3_ANDI = 3'b111;

    parameter F7_ADD = 7'b0110011;
    parameter F3_ADD = 3'b000;
    parameter F7_ADD_ = 7'b0000000;

    parameter F7_SUB = 7'b0110011;
    parameter F3_SUB = 3'b000;
    parameter F7_SUB_ = 7'b0100000;

    parameter F7_AND = 7'b0110011;
    parameter F3_AND = 3'b111;

    parameter F7_OR = 7'b0110011;
    parameter F3_OR = 3'b110;

    parameter F7_XOR = 7'b0110011;
    parameter F3_XOR = 3'b100;
```

```
parameter F7_ADDIW = 7'b0011011;
parameter F3_ADDIW = 3'b000;

parameter F7_ADDW = 7'b0111011;
parameter F3_ADDW = 3'b000;
parameter F7_ADDW_ = 7'b0000000;

parameter F7_SUBW = 7'b0111011;
parameter F3_SUBW = 3'b000;
parameter F7_SUBW_ = 7'b0100000;

parameter F7_LD = 7'b0000011;
parameter F3_LD = 3'b011;

parameter F7_SD = 7'b0100011;
parameter F3_SD = 3'b011;

parameter F7_LB = 7'b0000011;
parameter F3_LB = 3'b000;

parameter F7_LH = 7'b0000011;
parameter F3_LH = 3'b001;

parameter F7_LW = 7'b0000011;
parameter F3_LW = 3'b010;

parameter F7_LBU = 7'b0000011;
parameter F3_LBU = 3'b100;

parameter F7_LHU = 7'b0000011;
parameter F3_LHU = 3'b101;

parameter F7_LWU = 7'b0000011;
parameter F3_LWU = 3'b110;

parameter F7_SB = 7'b0100011;
parameter F3_SB = 3'b000;

parameter F7_SH = 7'b0100011;
parameter F3_SH = 3'b001;

parameter F7_SW = 7'b0100011;
```

```
parameter F3_SW = 3'b010;

parameter F7_LUI = 7'b0110111;

parameter F7_BEQ = 7'b1100011;
parameter F3_BEQ = 3'b000;

parameter F7_BNE = 7'b1100011;
parameter F3_BNE = 3'b001;

parameter F7_BLT = 7'b1100011;
parameter F3_BLT = 3'b100;

parameter F7_BGE = 7'b1100011;
parameter F3_BGE = 3'b101;

parameter F7_BLTU = 7'b1100011;
parameter F3_BLTU = 3'b110;

parameter F7_BGEU = 7'b1100011;
parameter F3_BGEU = 3'b111;

parameter F7_SLTI = 7'b0010011;
parameter F3_SLTI = 3'b010;

parameter F7_SLTIU = 7'b0010011;
parameter F3_SLTIU = 3'b011;

parameter F7_SLLI = 7'b0010011;
parameter F3_SLLI = 3'b001;

parameter F7_SRLI = 7'b0010011;
parameter F3_SRLI = 3'b101;

parameter F7_SRAI = 7'b0010011;
parameter F3_SRAI = 3'b101;

parameter F7_SLL = 7'b0110011;
parameter F3_SLL = 3'b001;

parameter F7_SLT = 7'b0110011;
parameter F3_SLT = 3'b010;
```

```
parameter F7_SLTU = 7'b0110011;
parameter F3_SLTU = 3'b011;

parameter F7_SRL = 7'b0110011;
parameter F3_SRL = 3'b101;
parameter F7_SRL_ = 7'b0000000;

parameter F7_SRA = 7'b0110011;
parameter F3_SRA = 3'b101;
parameter F7_SRA_ = 7'b0100000;

parameter F7_SLLIW = 7'b0011011;
parameter F3_SLLIW = 3'b001;

parameter F7_SRLIW = 7'b0011011;
parameter F3_SRLIW = 3'b101;

parameter F7_SRAIW = 7'b0011011;
parameter F3_SRAIW = 3'b101;

parameter F7_SLLW = 7'b0111011;
parameter F3_SLLW = 3'b001;

parameter F7_SRLW = 7'b0111011;
parameter F3_SRLW = 3'b101;
parameter F7_SRLW_ = 7'b0000000;

parameter F7_SRAW = 7'b0111011;
parameter F3_SRAW = 3'b101;
parameter F7_SRAW_ = 7'b0100000;

parameter F7_AUIPC = 7'b0010111;

parameter F7_JAL = 7'b1101111;

parameter F7_JALR = 7'b1100111;

parameter F7_CSRRW = 7'b1110011;
parameter F3_CSRRW = 3'b001;

parameter F7_CSRRS = 7'b1110011;
parameter F3_CSRRS = 3'b010;
```

```
parameter F7_CSRRC = 7'b1110011;
parameter F3_CSRRC = 3'b011;

parameter F7_CSRRWI = 7'b1110011;
parameter F3_CSRRWI = 3'b101;

parameter F7_CSRRCI = 7'b1110011;
parameter F3_CSRRCI = 3'b111;

parameter F7_CSRRSI = 7'b1110011;
parameter F3_CSRRSI = 3'b110;

// typedef enum logic {
//     IDLE,
//     TEMP
// } state_t;

typedef struct packed {
    logic valid;
    u64 pc;
    u32 instr;
} fetch_data_t;

typedef struct packed {
    decode_op_t op;
    alufunc_t alufunc;
    u1 regwrite, memtoreg, memwrite;
} control_t;

typedef struct packed {
    logic valid;
    u64 pc;
    u32 instr;
    u5 rega, regb;
    word_t srca, srcb;
    // word_t rd1, rd2;
    logic [51:0] rd1;
    csr_addr_t csrdst;
    word_t csr;
    word_t store_data;
    control_t ctl;
    creg_addr_t dst;
    // csr_t csr;
```

```
    // csr_addr_t csrdst;
    // u12 csrdst;
    // word_t csr;
} decode_data_t;

typedef struct packed {
    logic valid;
    u64 pc;
    u32 instr;
    word_t result;
    // csr_addr_t csrdst;
    // word_t csr;
    word_t store_data;
    control_t ctl;
    creg_addr_t dst;
    word_t csrres;
    // csr_addr_t csrdst;
    // u12 csrdst;
    // word_t csr;
} execute_data_t;

typedef struct packed {
    logic valid;
    u64 pc;
    u32 instr;
    word_t result;
    word_t memaddr;
    control_t ctl;
    creg_addr_t dst;
    word_t csrres;
    // csr_addr_t csrdst;
    // u12 csrdst;
    // word_t csr;
} memory_data_t;

typedef struct packed {
    logic valid;
    u64 pc;
    u32 instr;
    word_t result;
    word_t memaddr;
    control_t ctl;
    creg_addr_t dst;
```

```
        word_t csrres;
        // csr_addr_t csrdst;
        // u12 csrdst;
        // word_t csr;
    } writeback_data_t;

endpackage

`endif
```

定义所需常量和结构体。

8. core.sv

```
`ifndef __CORE_SV
`define __CORE_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "pipeline/regfile/regfile.sv"
`include "pipeline/regfile/csrfile.sv"
`include "pipeline/fetch/fetch.sv"
`include "pipeline/fetch/pcselect.sv"
`include "pipeline/decode/decode.sv"
`include "pipeline/pipeline_reg/pipeline_reg.sv"
`include "pipeline/execute/execute.sv"
`include "pipeline/memory/memory.sv"
`include "pipeline/writeback/writeback.sv"

`else

`endif

module core
        import common::*;
        import pipes::*;(
        input logic clk, reset,
        output ibus_req_t  ireq,
        input  ibus_resp_t iresp,
        output dbus_req_t  dreq,
        input  dbus_resp_t dresp,
        input logic trint, swint, exint
);
        /* TODO: Add your pipeline here. */

        u1 stallpc, stallf, stalld, stalle, stallm, stall_raw, stall, need_nop;


        // assign stallf = (dataE.ctl.op == LD);
        // assign stalld = stall_raw | stallpc | stall;
        // assign stalle = stallpc | stall;
        // assign stallm = stallpc | stall;
        // assign stalld = (dataD.rega != 0 && dataD.rega == dstE) || (dataD.regb != 0 && dataD.
        // assign stallf = stallpc;
```

```
    assign stallm = dreq.valid && ~dresp.data_ok;
    // assign stall = dreq.valid && ~dresp.data_ok;

    u64 pc, pc_nxt, pc_prev;
    u1 branch;
    u64 jump;
    u1 branch_enable;
    u64 branch_target;

    assign stallpc = ireq.valid && ~iresp.data_ok;

// state_t state;

// logic [4:0] temp_counter;

    always_ff @( posedge clk ) begin
            if(reset) begin
                    pc <= 64'h8000_0000;
            end
            else if(stallpc | stallm | stalld) begin
                    pc <= pc;
            end
            // else if(branch_enable) begin
            //      pc <= branch_target;
            // end
            else begin
                    pc <= pc_nxt;
            end
    end

    always_ff @(posedge clk) begin
            if(reset) begin
                    jump <= '0;
            end else if(branch | stallm) begin
                    jump <= jump;
            end else if((!stallpc && !stalld && !stallm) || branch == '0) begin
                    jump <= branch_target;
            end
    end

    always_ff @(posedge clk) begin
            if(reset) begin
                    branch <= '0;
```

```
            end else if(stallm) begin
                    branch <= branch;
            end
            else if((!stallpc && !stalld && !stallm) || branch == '0) begin
                    branch <= branch_enable;
            end
    end

    // logic ireq_active;

    // // 更新请求活跃状态（时序逻辑）
// always_ff @(posedge clk) begin
//     if (reset) begin
//         ireq_active <= 1'b1;
//     end else begin
//         if(iresp.data_ok) begin
    //                     if(stalld | stallm) begin
    //                             ireq_active <= 1'b0;
    //                     end
    //             end
    //             if(!stalld & !stallm) begin
    //                     ireq_active <= 1'b1;
    //             end
//     end
// end

    assign ireq.valid = 1'b1;
    assign ireq.addr = pc;

    u32 raw_instr;

    assign raw_instr = iresp.data;

    fetch_data_t dataF, dataF_nxt;
    decode_data_t dataD, dataD_nxt;
    execute_data_t dataE, dataE_nxt;
    memory_data_t dataM, dataM_nxt;
    writeback_data_t dataW, dataW_nxt;

    creg_addr_t ra1, ra2;
    // csr_addr_t csraddr;
    word_t rd1, rd2;
    // word_t csrdata;
```

```
csr_addr_t csraddr;
word_t csrdata;

// assign csrdata = 64'b0;
// assign csraddr = 12'b0;

u1 flushF, stop;

assign flushF = ireq.valid & ~iresp.data_ok;

u5 dstD, dstE, dstM, dstW;
// u1 branchD, branchE, branchM, branchW;

assign dstD = (dataD.ctl.regwrite && dataD.valid) ? dataD.dst : 0;
assign dstE = (dataE.ctl.regwrite && dataE.valid) ? dataE.dst : 0;
assign dstM = (dataM.ctl.regwrite && dataM.valid) ? dataM.dst : 0;
assign dstW = (dataW.ctl.regwrite && dataW.valid) ? dataW.dst : 0;

// csr_addr_t csrD, csrE, csrM, csrW;

// assign csrD = ((dataD.ctl.op == CSR || dataD.ctl.op == CSRI) && dataD.valid) ? dataD.
// assign csrE = ((dataE.ctl.op == CSR || dataE.ctl.op == CSRI) && dataE.valid) ? dataE.
// assign csrM = ((dataM.ctl.op == CSR || dataM.ctl.op == CSRI) && dataM.valid) ? dataM.
// assign csrW = ((dataW.ctl.op == CSR || dataW.ctl.op == CSRI) && dataW.valid) ? dataW.

// assign branchD = (dataD.ctl.op == JAL) || (dataD.ctl.op == JALR) || (dataD.ctl.op ==
// assign branchE = (dataE.ctl.op == JAL) || (dataE.ctl.op == JALR) || (dataE.ctl.op ==
// assign branchM = (dataM.ctl.op == JAL) || (dataM.ctl.op == JALR) || (dataM.ctl.op ==
// assign branchW = (dataW.ctl.op == JAL) || (dataW.ctl.op == JALR) || (dataW.ctl.op ==

// u12 csrs;

pipeline_reg pipeline_reg (
        .clk, .reset,
        .flushF,
        .stallf, .stalld, .stalle, .stallm, .stall, .stallpc, .stall_raw,
        .ireq, .iresp, .dreq, .dresp,
        .dataF_nxt, .dataF,
        .dataD_nxt, .dataD,
        .dataE_nxt, .dataE,
        .dataM_nxt, .dataM,
        .dataW_nxt, .dataW,
```

```
        .need_nop,
        .branch,
        .branch_enable
);

regfile regfile(
        .clk, .reset,
        .ra1,
        .ra2,
        .rd1,
        .rd2
);

fetch fetch (
        .clk, .reset,
        .pcplus4(pc + 4),
        .pc_selected(pc_nxt),
        .dataF(dataF_nxt),
        .raw_instr(raw_instr),
        .pc(pc),
        .iresp,
        .dresp,
        .ireq,
        .dreq,
        .stallf, .stalld, .stalle, .stallm,
        .branch,
        .jump
);

decode decode (
        .clk, .reset,
        .dataF,
        .dataD(dataD_nxt),
        .ra1, .ra2, .rd1, .rd2,
        .ireq,
        .iresp,
        .stallf, .stalld, .stalle, .stallm, .stall_raw,
        .dstD, .dstE, .dstM, .dstW,
        .dreq, .dresp,
        .branch,
        .csraddr,
        .csrdata,
        .csrD(((dataD.ctl.op == CSR || dataD.ctl.op == CSRI) && dataD.valid) ? dataD.ins
```

```
        .csrE(((dataE.ctl.op == CSR || dataE.ctl.op == CSRI) && dataE.valid) ? dataE.ins
        .csrM(((dataM.ctl.op == CSR || dataM.ctl.op == CSRI) && dataM.valid) ? dataM.ins
        .csrW(((dataW.ctl.op == CSR || dataW.ctl.op == CSRI) && dataW.valid) ? dataW.ins
);

execute execute(
        .clk, .reset,
        .dataD,
        .dataE(dataE_nxt),
        .stallf, .stalld, .stalle, .stallm,
        .branch(branch_enable),
        .jump(branch_target)
);

memory memory(
        .clk, .reset,
        .dataE,
        .dataM(dataM_nxt),
        .dreq,
        .dresp,
        .stallf, .stalld, .stalle, .stallm, .stall
);

writeback writeback(
        .clk, .reset,
        .dataM,
        .dataW(dataW_nxt),
        .wvalid(dataM.ctl.regwrite),
        .regs(regfile.regs),
        .regs_nxt(regfile.regs_nxt)
);

csrfile csrfile(
        .clk, .reset,
        .csr_ra(csraddr),
        .csr_out(csrdata),
        .csr_wen(dataW.ctl.op == CSR || dataW.ctl.op == CSRI),
        .csr_wa(dataW.instr[31:20]),
        .csrresult(dataW.memaddr),
        .dataW
);

// u1 commit_valid;
```

```
        // u64 pc_prev;
        // logic [63:0] pc_prev;
        // logic commit_ok;

        always_ff @(posedge clk) begin
                if(reset) begin
                        pc_prev <= '0;
                end
                else if (dataW.valid) begin
                        pc_prev <= dataW.pc;
                end
        end

        always_ff @(posedge clk) begin
                if(reset) begin
                        pc_prev <= '0;
                end
                else if (dataW.instr != '0) begin
                        pc_prev <= dataW.pc;
                end
        end

`ifdef VERILATOR
        DifftestInstrCommit DifftestInstrCommit(
                .clock              (clk),
                .coreid             (0),
                .index              (0),
                .valid              (dataW.valid & dataW.pc != pc_prev),
                .pc                 (dataW.pc),
                .instr              (dataW.instr),
                .skip               ((dataW.ctl.op == LD || dataW.ctl.op == SD) && dataW.memaddı
                .isRVC              (0),
                .scFailed           (0),
                .wen                (dataW.ctl.regwrite),
                .wdest              ({3'b0, dataW.dst}),
                .wdata              (dataW.result)
        );

        DifftestArchIntRegState DifftestArchIntRegState (
                .clock              (clk),
                .coreid             (0),
                .gpr_0              (regfile.regs_nxt[0]),
                .gpr_1              (regfile.regs_nxt[1]),
```

```
            .gpr_2              (regfile.regs_nxt[2]),
            .gpr_3              (regfile.regs_nxt[3]),
            .gpr_4              (regfile.regs_nxt[4]),
            .gpr_5              (regfile.regs_nxt[5]),
            .gpr_6              (regfile.regs_nxt[6]),
            .gpr_7              (regfile.regs_nxt[7]),
            .gpr_8              (regfile.regs_nxt[8]),
            .gpr_9              (regfile.regs_nxt[9]),
            .gpr_10             (regfile.regs_nxt[10]),
            .gpr_11             (regfile.regs_nxt[11]),
            .gpr_12             (regfile.regs_nxt[12]),
            .gpr_13             (regfile.regs_nxt[13]),
            .gpr_14             (regfile.regs_nxt[14]),
            .gpr_15             (regfile.regs_nxt[15]),
            .gpr_16             (regfile.regs_nxt[16]),
            .gpr_17             (regfile.regs_nxt[17]),
            .gpr_18             (regfile.regs_nxt[18]),
            .gpr_19             (regfile.regs_nxt[19]),
            .gpr_20             (regfile.regs_nxt[20]),
            .gpr_21             (regfile.regs_nxt[21]),
            .gpr_22             (regfile.regs_nxt[22]),
            .gpr_23             (regfile.regs_nxt[23]),
            .gpr_24             (regfile.regs_nxt[24]),
            .gpr_25             (regfile.regs_nxt[25]),
            .gpr_26             (regfile.regs_nxt[26]),
            .gpr_27             (regfile.regs_nxt[27]),
            .gpr_28             (regfile.regs_nxt[28]),
            .gpr_29             (regfile.regs_nxt[29]),
            .gpr_30             (regfile.regs_nxt[30]),
            .gpr_31             (regfile.regs_nxt[31])
    );

DifftestTrapEvent DifftestTrapEvent(
            .clock              (clk),
            .coreid             (0),
            .valid              (0),
            .code               (0),
            .pc                 (0),
            .cycleCnt           (0),
            .instrCnt           (0)
    );

    DifftestCSRState DifftestCSRState(
```

```
                .clock              (clk),
                .coreid             (0),
                .priviledgeMode     (3),
                .mstatus            (csrfile.csrs_nxt.mstatus),
                .sstatus            (csrfile.csrs_nxt.mstatus & 64'h800000030001e000),
                .mepc               (csrfile.csrs_nxt.mepc),
                .sepc               (0),
                .mtval              (csrfile.csrs_nxt.mtval),
                .stval              (0),
                .mtvec              (csrfile.csrs_nxt.mtvec),
                .stvec              (0),
                .mcause             (csrfile.csrs_nxt.mcause),
                .scause             (0),
                .satp               (csrfile.csrs_nxt.satp),
                .mip                (csrfile.csrs_nxt.mip),
                .mie                (csrfile.csrs_nxt.mie),
                .mscratch           (csrfile.csrs_nxt.mscratch),
                .sscratch           (0),
                .mideleg            (0),
                .medeleg            (0)
        );
`endif
endmodule
`endif
```

跳转信号在阻塞结束时发出。

9. pipeline_reg.sv

```
`ifndef __PIPELINE_REG_SV
`define __PIPELINE_REG_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`include "pipeline/decode/decoder.sv"
`else

`endif

module pipeline_reg
    import common::*;
    import pipes::*; (
        input logic clk, reset,
        input logic flushF, stallf, stalld, stalle, stallm, stall, stallpc, stall_raw,
        input ibus_req_t ireq,
        input ibus_resp_t iresp,
        input dbus_req_t dreq,
        input dbus_resp_t dresp,
        input fetch_data_t dataF_nxt,
        output fetch_data_t dataF,

        input decode_data_t dataD_nxt,
        output decode_data_t dataD,

        input execute_data_t dataE_nxt,
        output execute_data_t dataE,

        input memory_data_t dataM_nxt,
        output memory_data_t dataM,

        input writeback_data_t dataW_nxt,
        output writeback_data_t dataW,

        output logic need_nop,
        input u1 branch,
        input u1 branch_enable
    );
```

```
// always_ff @(posedge clk) begin
// if(reset) begin
//     state <= IDLE;
//     dataF <= '0;
//     dataD <= '0;
//     dataE <= '0;
//     dataM <= '0;
//     dataW <= '0;
//     temp_counter <= 0;
// end
// else begin
//     case(state)
//         IDLE: begin
//             dataF <= dataF_nxt;
//             dataD <= dataD_nxt;
//             dataE <= dataE_nxt;
//             dataM <= dataM_nxt;
//             dataW <= dataW_nxt;
//             temp_counter <= 0;
//             state <= TEMP;
//         end
//         TEMP: begin
//             if(temp_counter < 15) begin
//                 temp_counter <= temp_counter + 1;
//             end
//             else begin
//                 state <= IDLE;
//             end
//         end
//     endcase
// end
// end

// always_ff @(posedge clk) begin
//     if (reset) begin
//         dataF <= '0;
//     end
//     else if(stalld | stallm) begin
//         dataF <= dataF;
//     end
//     else if (~iresp.data_ok) begin
//         dataF.valid <= 0;
//     end
```

```
//     else begin
//         dataF <= dataF_nxt;
//     end
// end

// always_ff @(posedge clk) begin
//     if (reset) begin
//         dataD <= '0;
//     end
//     else if(stallm) begin
//         dataD <= dataD;
//     end
//     else if(stalld) begin
//         dataD.valid <= 0;
//     end
//     else begin
//         dataD <= dataD_nxt;
//     end
// end

// always_ff @(posedge clk) begin
//     if (reset) begin
//         dataE <= '0;
//     end
//     else if(stallm) begin
//         dataE <= dataE;
//     end
//     else begin
//         dataE <= dataE_nxt;
//     end
// end

// always_ff @(posedge clk) begin
//     if (reset) begin
//         dataM <= '0;
//     end
//     else if (stallm) begin
//         dataM.valid <= 0;
//     end
//     else begin
//         dataM <= dataM_nxt;
//     end
// end
```

```
// always_ff @(posedge clk) begin
//     if (reset) begin
//         dataW <= '0;
//     end
//     else begin
//         dataW <= dataW_nxt;
//     end
// end

always_ff @(posedge clk) begin
    if (reset) begin
        dataF <= '0;
    end
    else if(stalld | stallm) begin
        dataF <= dataF;
    end
    else if(branch) begin
        dataF <= '0;
    end
    else begin
        dataF <= dataF_nxt;
    end
end

always_ff @(posedge clk) begin
    if (reset) begin
        dataD <= '0;
    end
    else if(stalld) begin
        dataD <= '0;
    end
    else if(stallm) begin
        dataD <= dataD;
    end
    else if(branch) begin
        dataD <= '0;
    end
    else begin
        dataD <= dataD_nxt;
    end
end
```

```
always_ff @(posedge clk) begin
    if (reset) begin
        dataE <= '0;
    end
    else if(stallm) begin
        dataE <= dataE;
    end
    else if(branch) begin
        dataE <= '0;
    end
    else begin
        dataE <= dataE_nxt;
    end
end

always_ff @(posedge clk) begin
    if (reset) begin
        dataM <= '0;
    end
    else if(stallm) begin
        dataM <= dataM;
    end
    else begin
        dataM <= dataM_nxt;
    end
end

always_ff @(posedge clk) begin
    if (reset) begin
        dataW <= '0;
    end
    else begin
        dataW <= dataW_nxt;
    end
end

// state_t state;

// always_ff @(posedge clk) begin
//     if(reset) begin
//         state <= IDLE;
//     end
//     else begin
```

```
//         case (state)
//             IDLE:
//                 if(stall_raw) begin
//                     state <= INSERT_NOP1;
//                 end
//             INSERT_NOP1: state <= IDLE;
//             INSERT_NOP2: state <= IDLE;
//             INSERT_NOP3: state <= IDLE;
//         endcase
//     end
// end

// assign need_nop = (state != IDLE);

// always_ff @(posedge clk) begin
//     if (reset) begin
//         dataF <= '0;
//         dataD <= '0;
//         dataE <= '0;
//         dataM <= '0;
//         dataW <= '0;
//     end
//     // else if (stall_raw) begin
//     // // 插入气泡逻辑
//     //     dataF <= dataF;          // 冻结 IF 阶段（PC 已由外部逻辑暂停）
//     //     dataD <= dataD;             // 清零 ID 阶段（插入 NOP）
//     //     dataE <= '0;
//     //     dataM <= dataM_nxt;
//     //     dataW <= dataW_nxt;
//     // end
//     // else if (need_nop) begin
//     // // 插入气泡逻辑
//     //     dataF <= dataF;          // 冻结 IF 阶段（PC 已由外部逻辑暂停）
//     //     dataD <= dataD;             // 清零 ID 阶段（插入 NOP）
//     //     dataE <= '0;
//     //     dataM <= '0;
//     //     dataW <= dataW_nxt;
//     // end
//     // else if (need_nop) begin
//     //     dataF <= dataF;
//     //     dataD <= '0;
//     //     dataE <= dataE_nxt;
//     //     dataM <= dataM_nxt;
```

```
    //     //     dataW <= dataW_nxt;
    //     // end
    //     else if (stall) begin
    //         dataF <= dataF;
    //         dataD <= dataD;
    //         dataE <= dataE;
    //         dataM <= dataM;
    //         dataW <= dataW;
    //     end
    //     else begin
    //         dataF <= dataF_nxt;
    //         dataD <= dataD_nxt;
    //         dataE <= dataE_nxt;
    //         dataM <= dataM_nxt;
    //         dataW <= dataW_nxt;
    //     end
    // end

    // always_ff @(posedge clk) begin
    //     if (reset) begin
    //         dataF <= '0;
    //         dataD <= '0;
    //         dataE <= '0;
    //         dataM <= '0;
    //         dataW <= '0;
    //     end else if (!stall) begin
    //         dataF <= dataF_nxt;
    //         dataD <= dataD_nxt;
    //         dataE <= dataE_nxt;
    //         dataM <= dataM_nxt;
    //         dataW <= dataW_nxt;
    //     end else begin
    //         dataF <= dataF;
    //         dataD <= dataD;
    //         dataE <= dataE;
    //         dataM <= dataM;
    //         dataW <= dataW;
    //     end
    // end


endmodule
```

```
  `endif
```

流水线寄存器传递逻辑。

10. mmu.sv

```
`ifndef __MMU_SV
`define __MMU_SV

`ifdef VERILATOR
`include "include/common.sv"
`include "include/pipes.sv"
`else

`endif

module mmu
    import common::*;
    import pipes::*;
    import csr_pkg::*;(
        input u1 clk, reset,
        input satp_t satp,
        input u2 mode,

        input u1 mmu_req_valid,
        output u1 mmu_ok,

        input cbus_req_t vreq,
        input cbus_resp_t presp,
        output cbus_req_t preq
    );

    u64 va;
    assign va = vreq.addr;

    u64 mem_rdata;
    assign mem_rdata = presp.data;

    mmu_state_t state, next_state;

    u9 vpn[2:0];
    u44 root_ppn;
    u64 pte;
    u2 level;
    u12 page_offset;
```

```
assign vpn[0] = va[20:12];
assign vpn[1] = va[29:21];
assign vpn[2] = va[38:30];
assign page_offset = va[11:0];

/* FSM reg */
always_ff @(posedge clk or posedge reset) begin
    if (reset) state <= IDLE;
    else state <= next_state;
end

/* State Transfer Logic */
always_comb begin
    next_state = state;
    unique case (state)
        IDLE: begin
            if (mmu_req_valid) begin
                next_state = READ_L1;
                if (mode == 2'd3) next_state = READ_DATA;
            end
        end
        READ_L1: begin next_state = WAIT_L1; end
        WAIT_L1: begin
            if (presp.ready) begin
                next_state = (mem_rdata[1] || mem_rdata[2]) ? READ_DATA : READ_L2;
            end
        end
        READ_L2: begin next_state = WAIT_L2; end
        WAIT_L2: begin
            if (presp.ready) begin
                next_state = (mem_rdata[1] || mem_rdata[2]) ? READ_DATA : READ_L3;
            end
        end
        READ_L3: begin next_state = WAIT_L3; end
        WAIT_L3: begin
            if (presp.ready) begin
                next_state = READ_DATA;
            end
        end
        READ_DATA: next_state = presp.ready ? OUTPUT : WAIT_DATA;
        WAIT_DATA: if (presp.ready) next_state = OUTPUT;
        OUTPUT: if (!mmu_req_valid) next_state = IDLE;
    endcase
```

```
end

/* root addr */
always_ff @(posedge clk or posedge reset) begin
    if (state == IDLE && mmu_req_valid) begin
        root_ppn <= satp.ppn;
    end
end

/* mem req */
u64 table_base, pte_addr;

always_comb begin
    table_base = (state == READ_L1 || state == WAIT_L1) ? {8'b0, root_ppn, 12'b0} :
                 (state == READ_L2 || state == WAIT_L2) ? {8'b0, pte[53:10], 12'b0} :
                 (state == READ_L3 || state == WAIT_L3) ? {8'b0, pte[53:10], 12'b0} : 64'b0

    level = (state == READ_L1 || state == WAIT_L1) ? 2 :
            (state == READ_L2 || state == WAIT_L2) ? 1 :
            (state == READ_L3 || state == WAIT_L3) ? 0 : 0;

    pte_addr = table_base + {52'b0, vpn[level], 3'b000};
end

assign preq.valid = (
    state == READ_L1 || state == WAIT_L1 ||
    state == READ_L2 || state == WAIT_L2 ||
    state == READ_L3 || state == WAIT_L3 ||
    state == READ_DATA || state == WAIT_DATA
);
// assign preq.addr = pte_addr;
assign preq.addr =  (mode == 2'd3) ? vreq.addr :
                    (state == READ_DATA || state == WAIT_DATA) ? {8'b0, pte[53:10], page_of
assign preq.size = vreq.size;
assign preq.data = vreq.data;
assign preq.strobe = (state == READ_DATA || state == WAIT_DATA) ? vreq.strobe : 8'b0;
assign preq.is_write = vreq.is_write;

always_ff @(posedge clk) begin
    if (presp.ready && (
        state == WAIT_L1 || state == WAIT_L2 || state == WAIT_L3
    )) pte <= mem_rdata;
end
```

```
    // assign mmu_ok = (state == OUTPUT) ? 1'b1 : 1'b0;
    assign mmu_ok = state == WAIT_DATA && presp.ready;

endmodule


`endif
```

拦截cbus信号，实现虚拟地址向物理地址的转换。