MIPS32: Pipelined CPU

Seungwan Noh

Pusan National University

Department of Electronics Engineering

Contents

Pipelined CPU

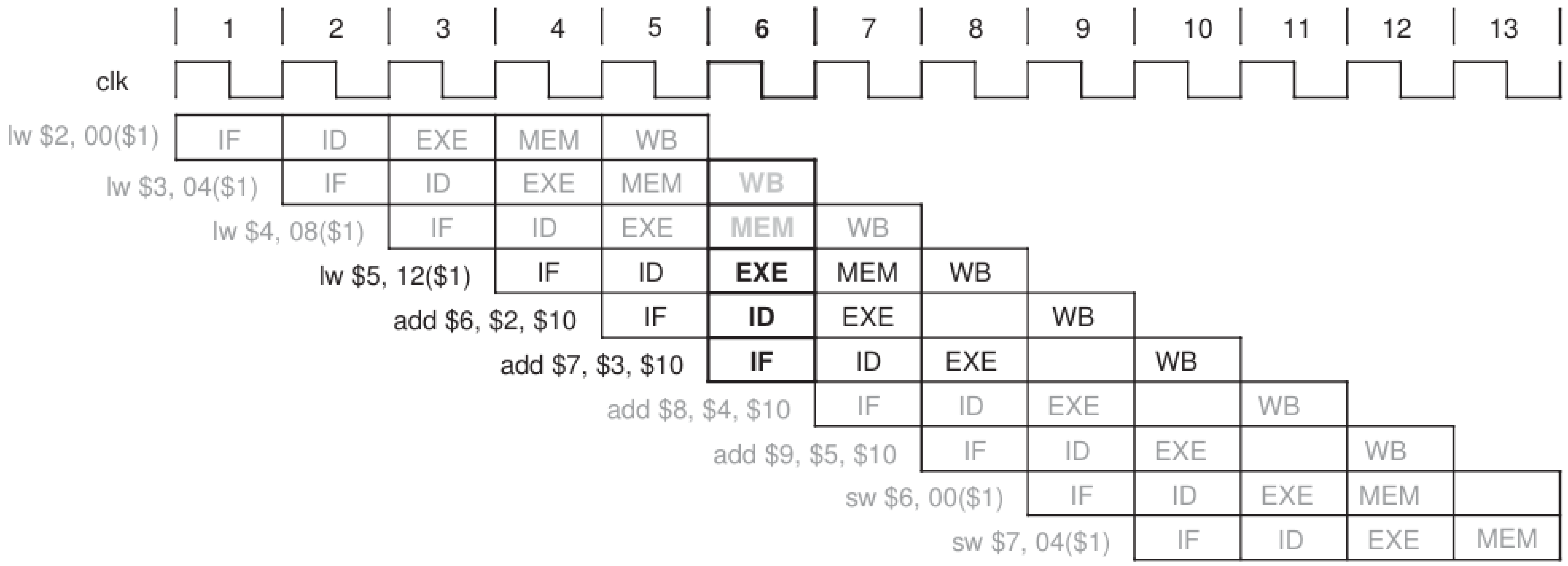
- The single-cycle CPU can execute another instruction only after the execution of its prior instruction is completed.
- The pipelined CPU can overlap multiple instructions.
- Ideally the pipelined CPU can produce a result in every clock.
- Because of the overlapping multiple instructions, the pipelined CPU may encounter three types of hazards.
  - ✓Structural hazards
  - ✓Control hazards
  - ✓Data hazards

Pipelined CPU

- Divide the execution of an instruction into five stages.
  - ✓Instruction Fetch (IF)
  - ✓Instruction Decode (ID)
  - ✓Execution (EXE)
  - ✓Memory Access (MEM)
  - ✓Write Back (WB)
- *SCCPU: n x m* cycles
- PLCPU: $n + m - 1$ cycles

Pipelined CPU

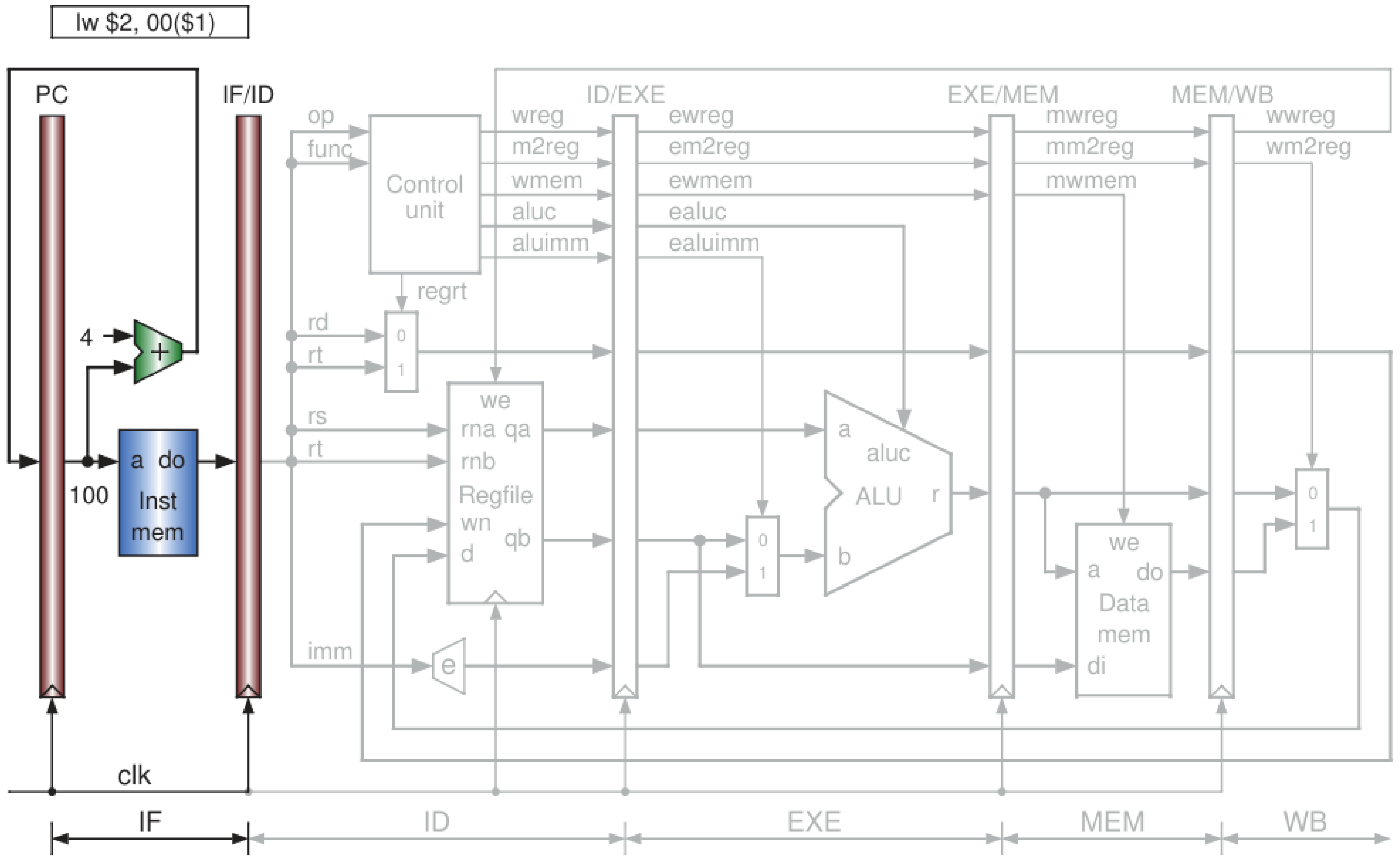
- Multiple operations in each clock cycle in PLCPU.
- Must save the temporary results in each pipeline stage.



	1	2	3	4	5	6	7	8	9	10	11	12	13
lw $2, 00($1)	IF	ID	EXE	MEM	WB								
lw $3, 04($1)		IF	ID	EXE	MEM	WB							
lw $4, 08($1)			IF	ID	EXE	MEM	WB						
lw $5, 12($1)				IF	ID	**EXE**	MEM	WB					
add $6, $2, $10					IF	**ID**	EXE		WB				
add $7, $3, $10						**IF**	ID	EXE		WB			
add $8, $4, $10							IF	ID	EXE		WB		
add $9, $5, $10								IF	ID	EXE		WB	
sw $6, 00($1)									IF	ID	EXE	MEM	
sw $7, 04($1)										IF	ID	EXE	MEM

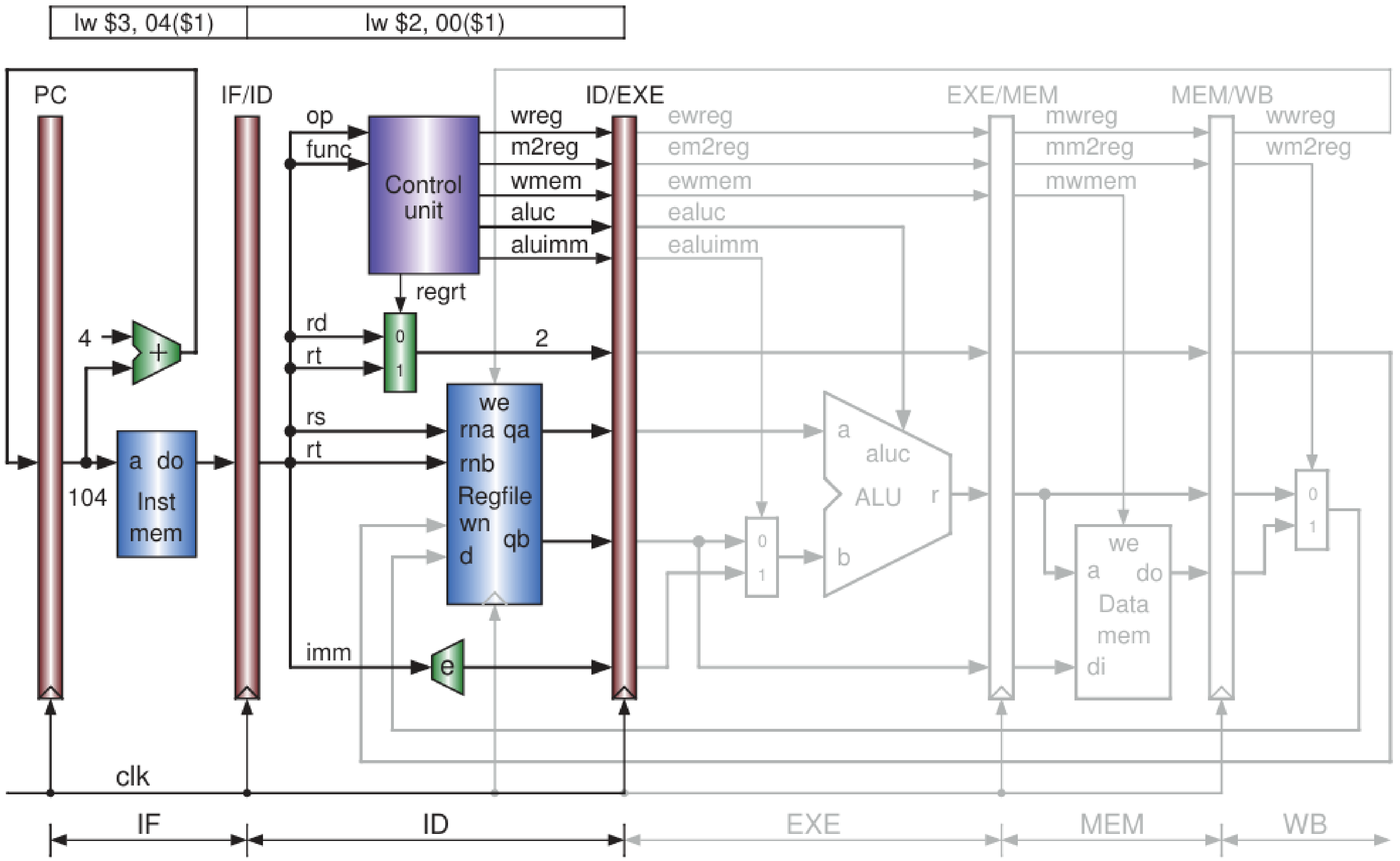
Pipelined CPU

- Circuits of the IF stage



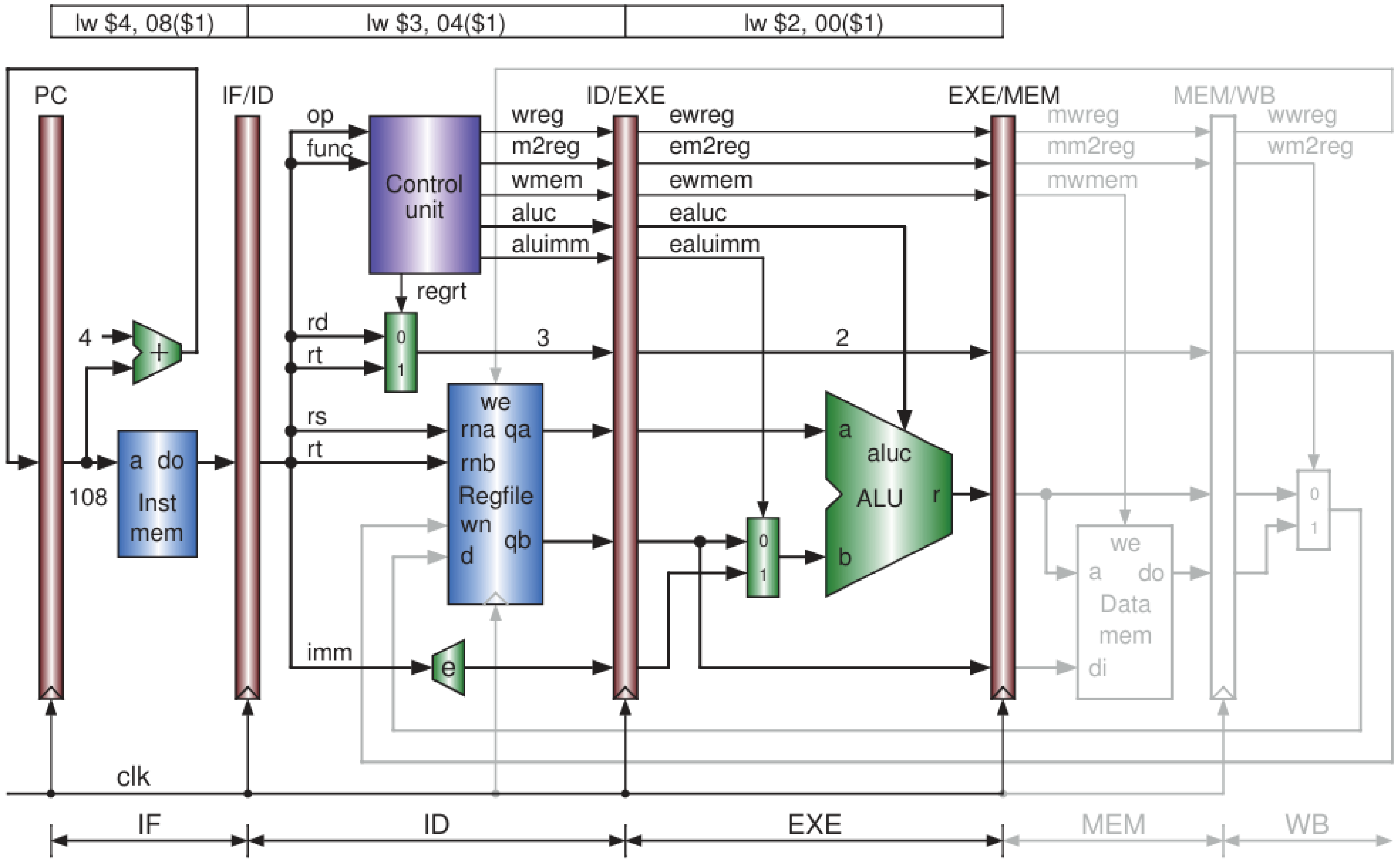
Pipelined CPU

- Circuits of the ID stage



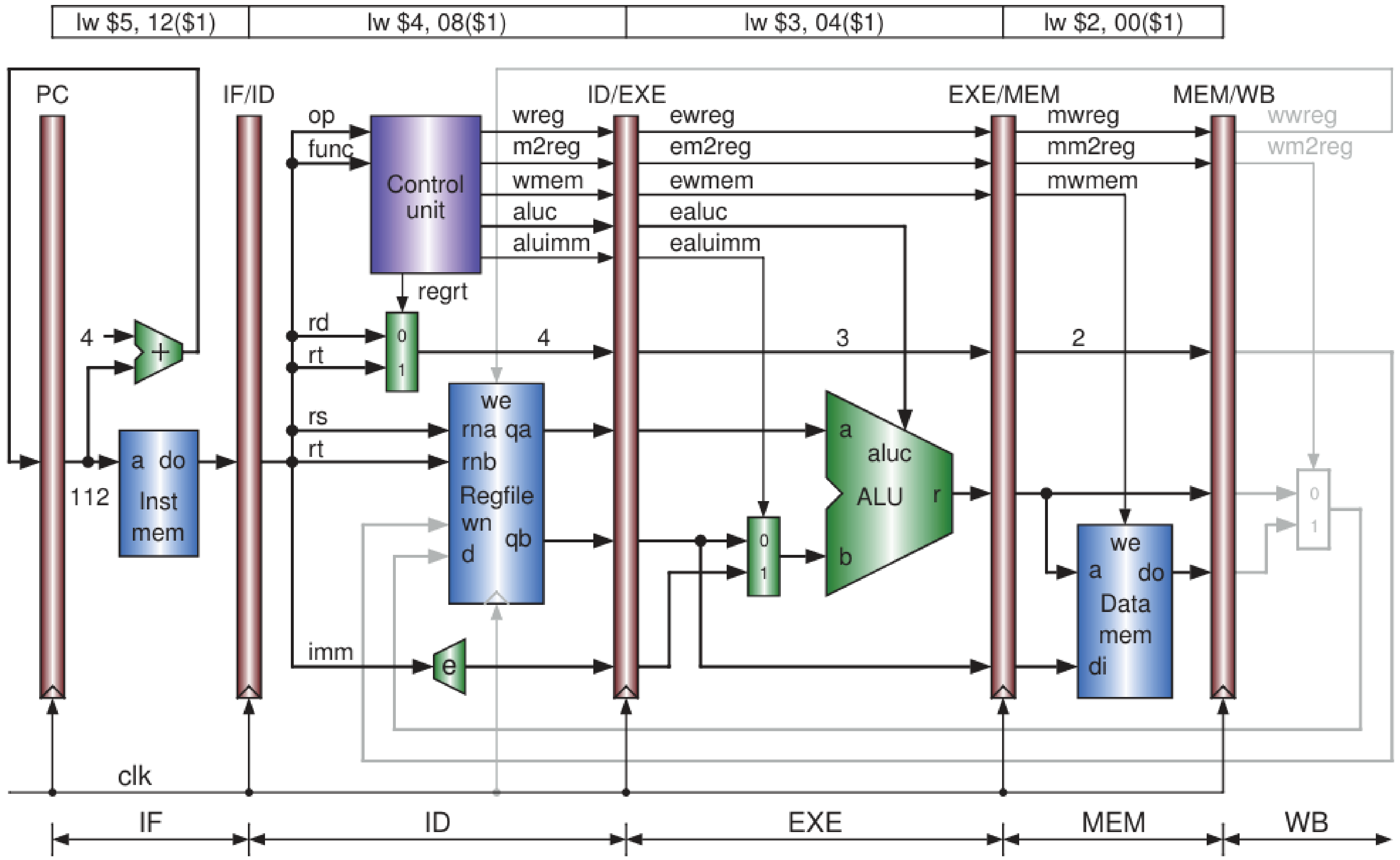
Pipelined CPU

- Circuits of the EXE stage



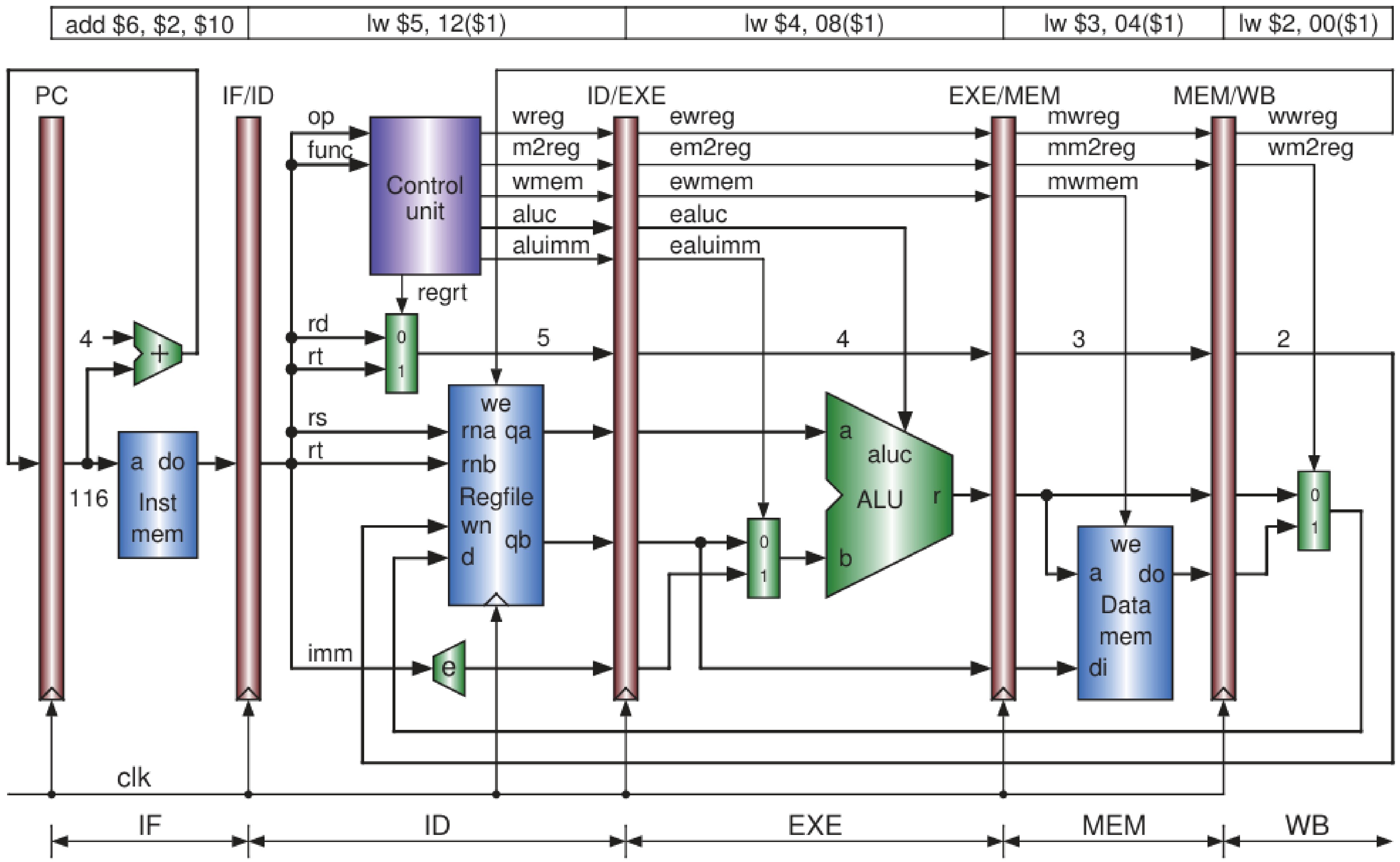
Pipelined CPU

- Circuits of the MEM stage



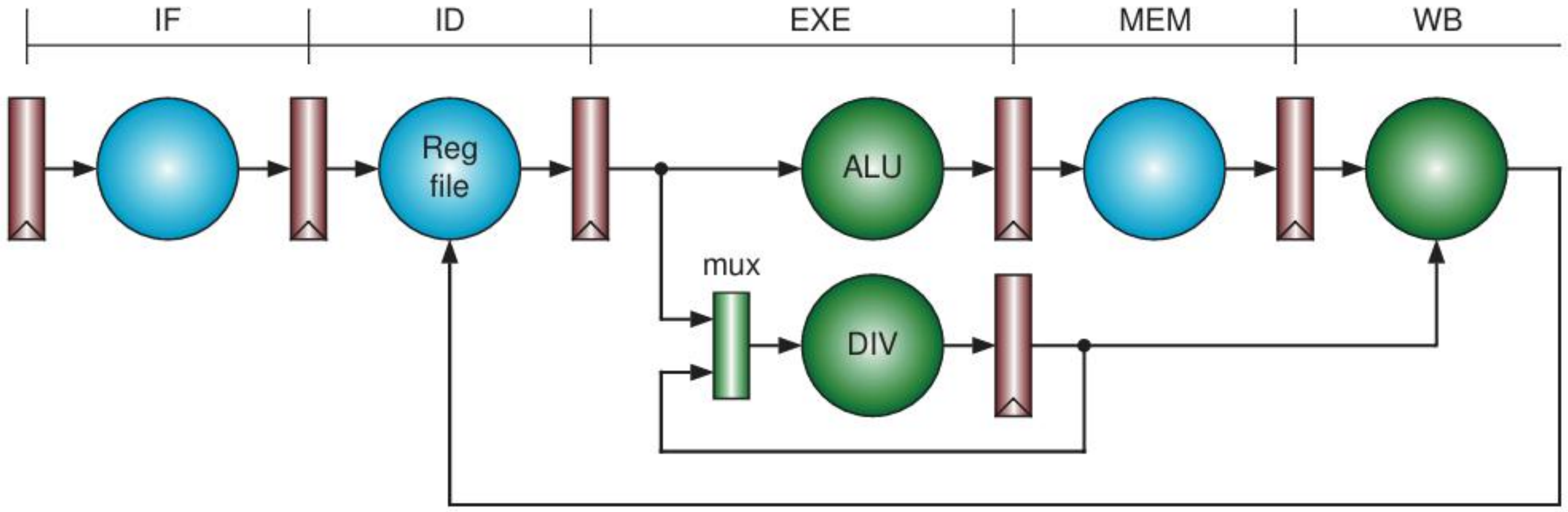
Pipelined CPU

- Circuits of the WB stage



Pipeline Hazards and Solutions

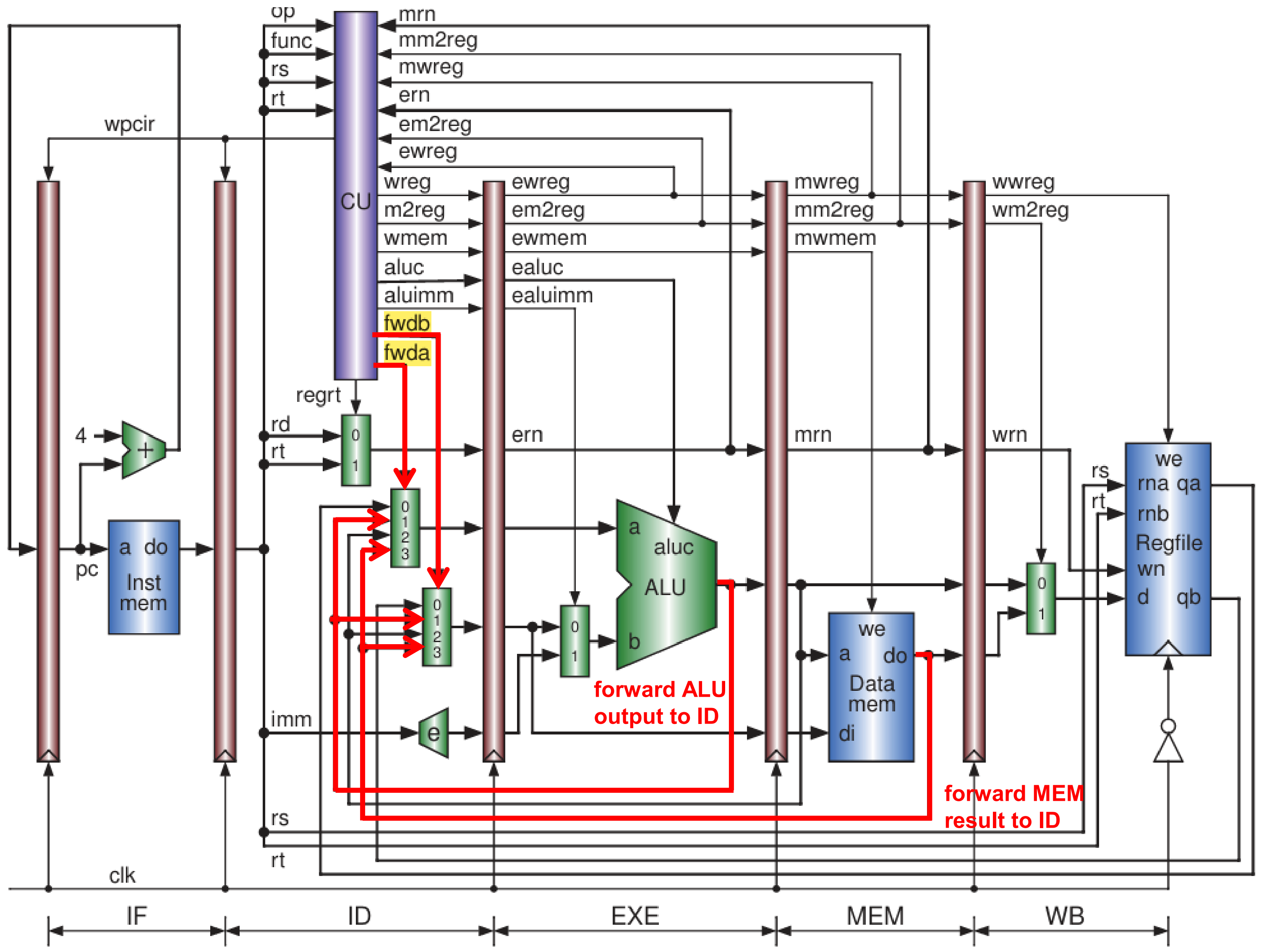
- **Structural Hazards and Solutions**
- Two or more instructions attempt to use a HW at the same time.



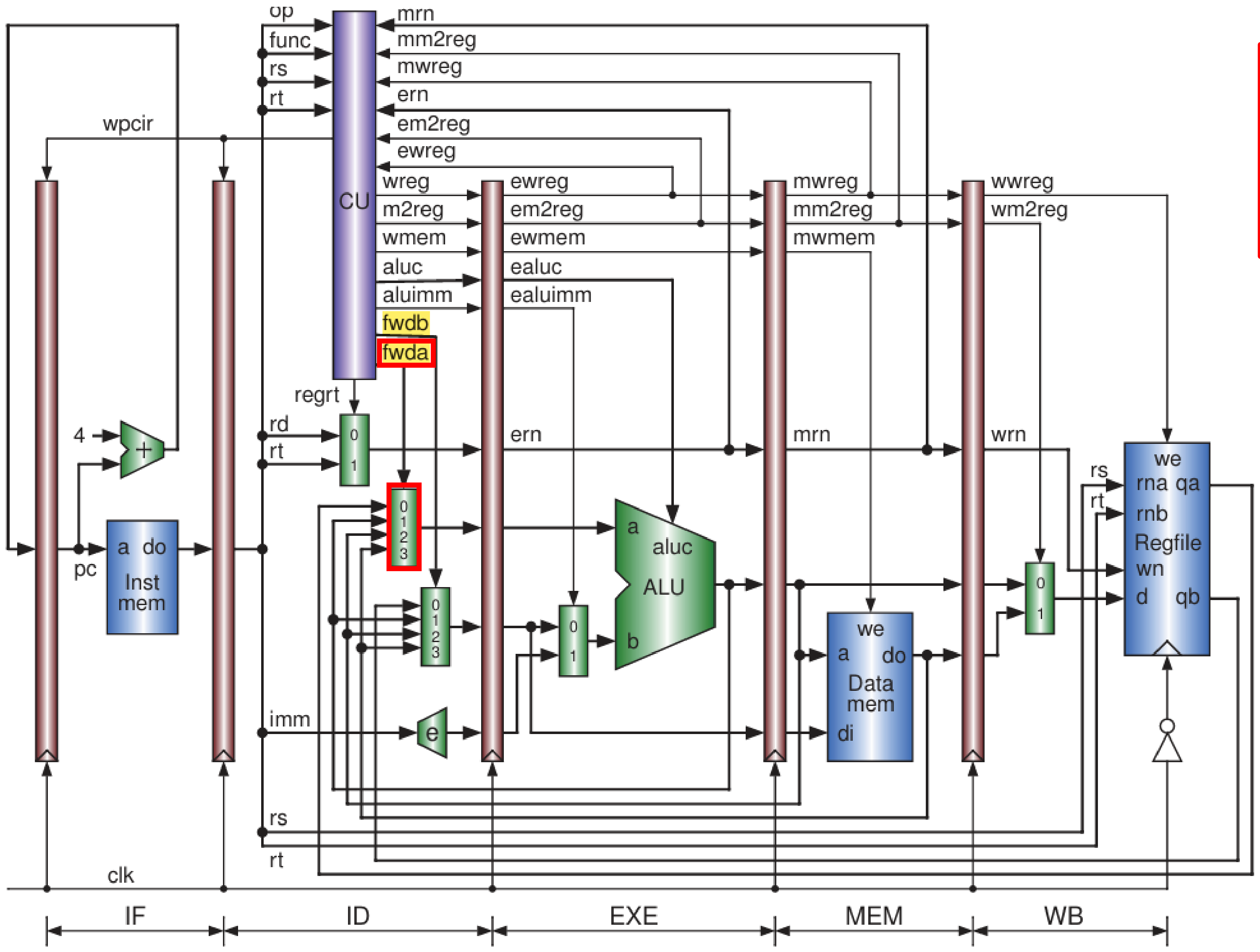
Pipeline Hazards and Solutions

- **Data Hazards and Internal Forwarding**
- Data dependency
- The instruction *i* uses the execution result of the instruction *i-1.*
- Stall
- Forwarding (bypass)

Data Hazards and Forwarding

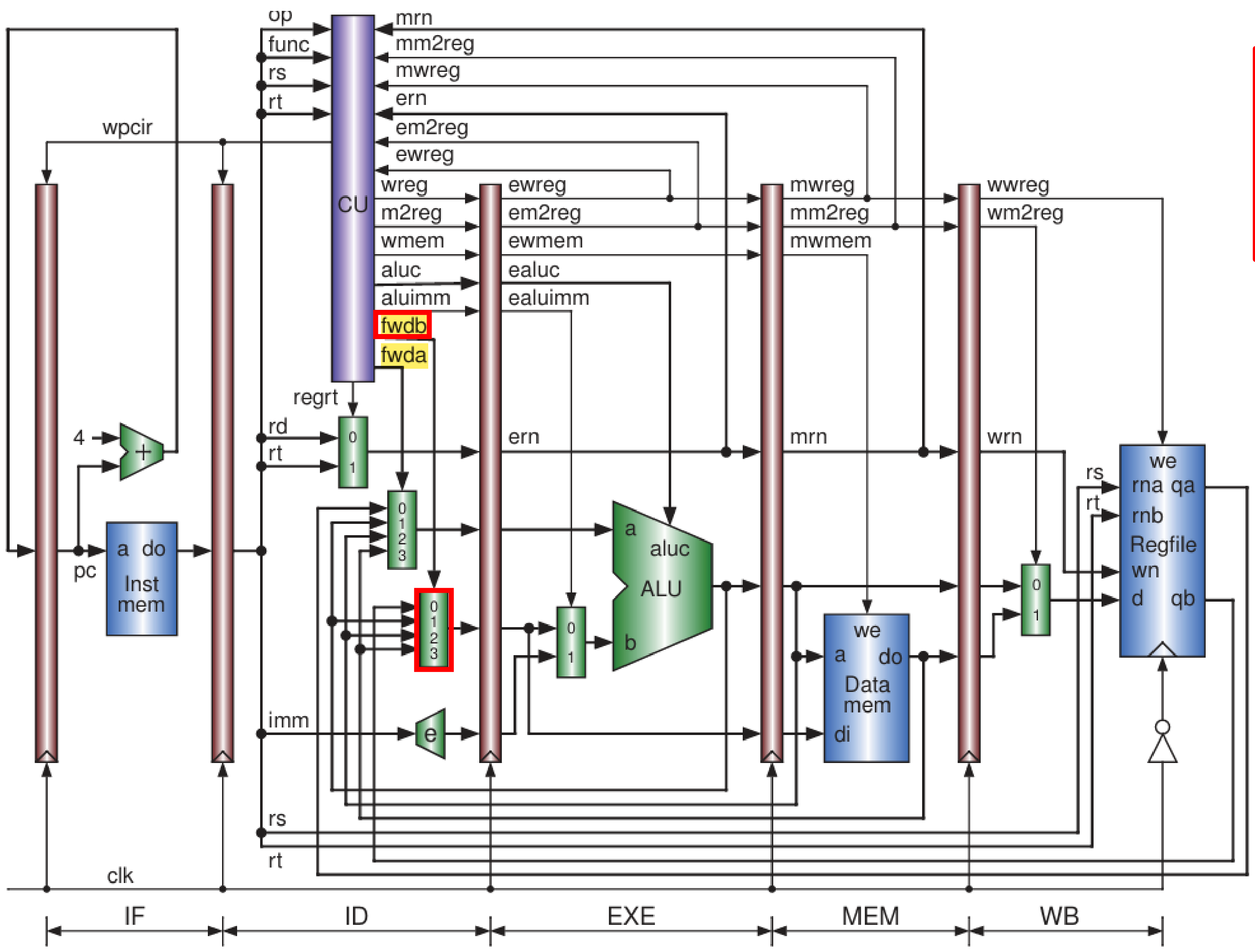


Data Hazards and Forwarding



```
fwda = 2'b00;  // default
fwda = 2'b01;  // exe_alu
fwda = 2'b10;  // mem_alu
fwda = 2'b11;  // mem_lw
```

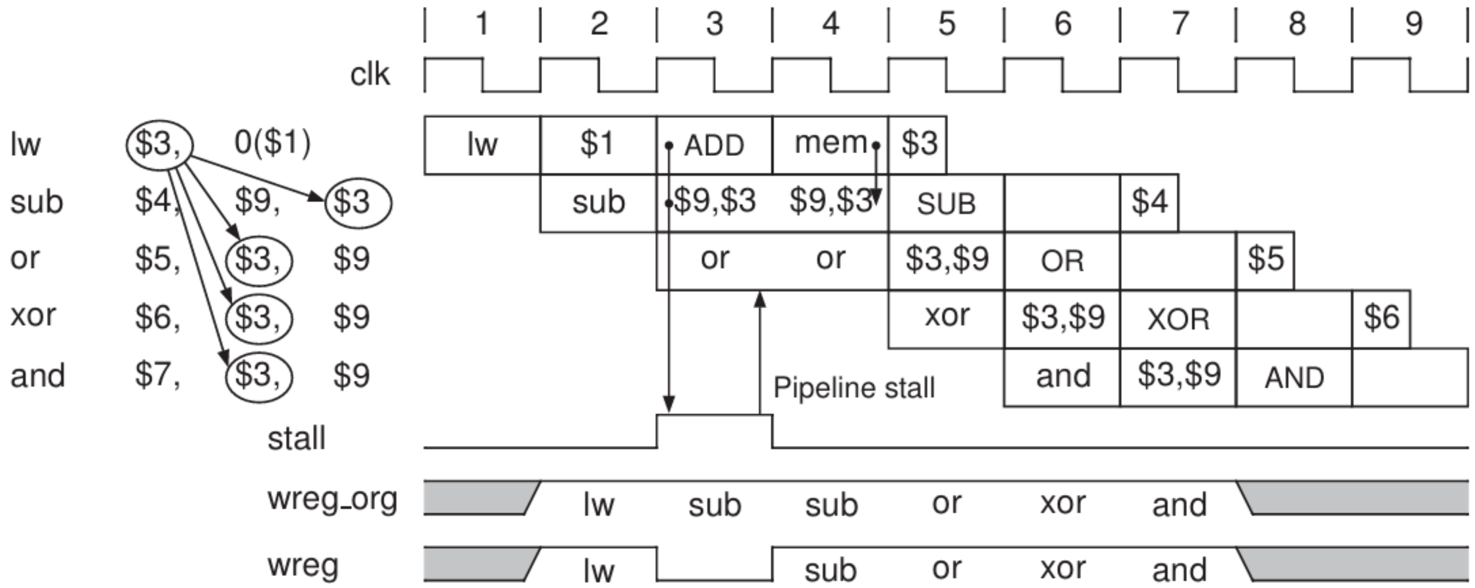
Data Hazards and Forwarding



```
fwdb = 2'b00;  // default
fwdb = 2'b01;  // exe_alu
fwdb = 2'b10;  // mem_alu
fwdb = 2'b11;  // mem_lw
```

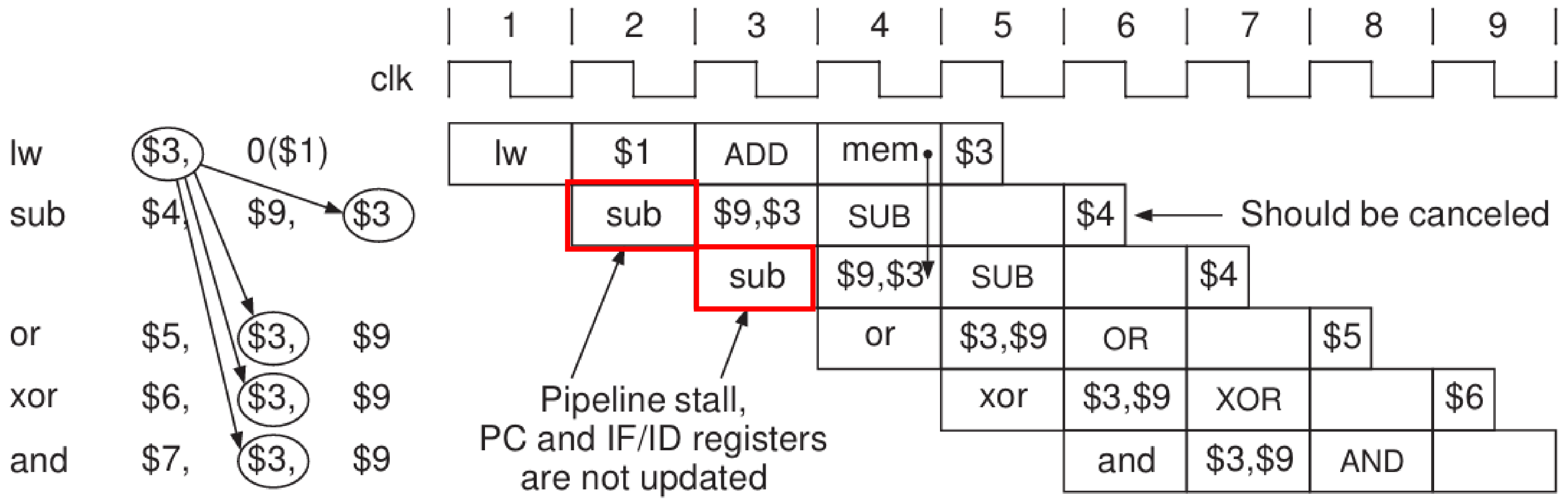
Data Hazards and Stall

- For the *lw* instruction, the pipeline must be **stalled** for one clock cycle for waiting for the memory data.
- The pipeline stall prohibits the updates of the PC and the IF/ID pipeline register.
- The instruction is already in IF/ID register and it will be executed twice.



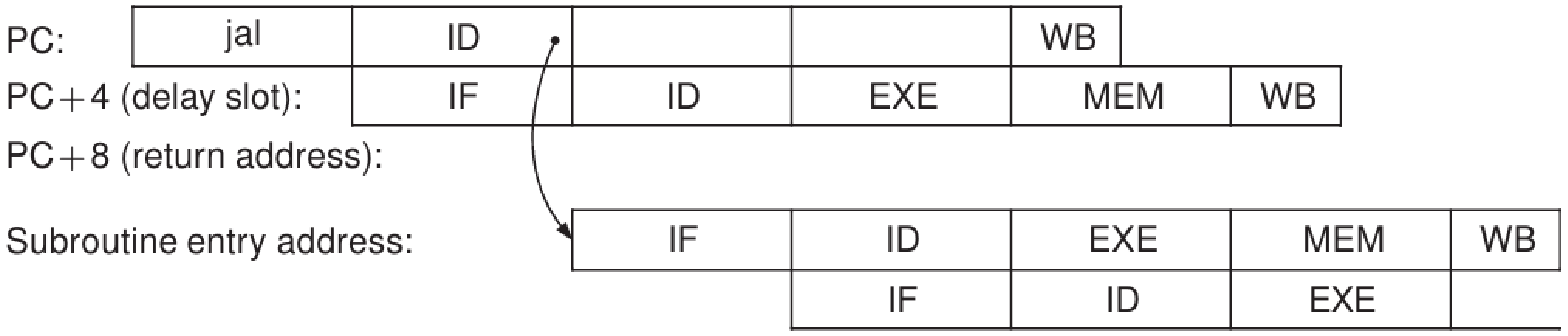
Data Hazards and Stall

- To prevent an instruction from being executed twice, we must cancel the first instruction.
- Prevent it from updating the states of the CPU and memory.
  - ✓Disable the *wreg* and *wmem*

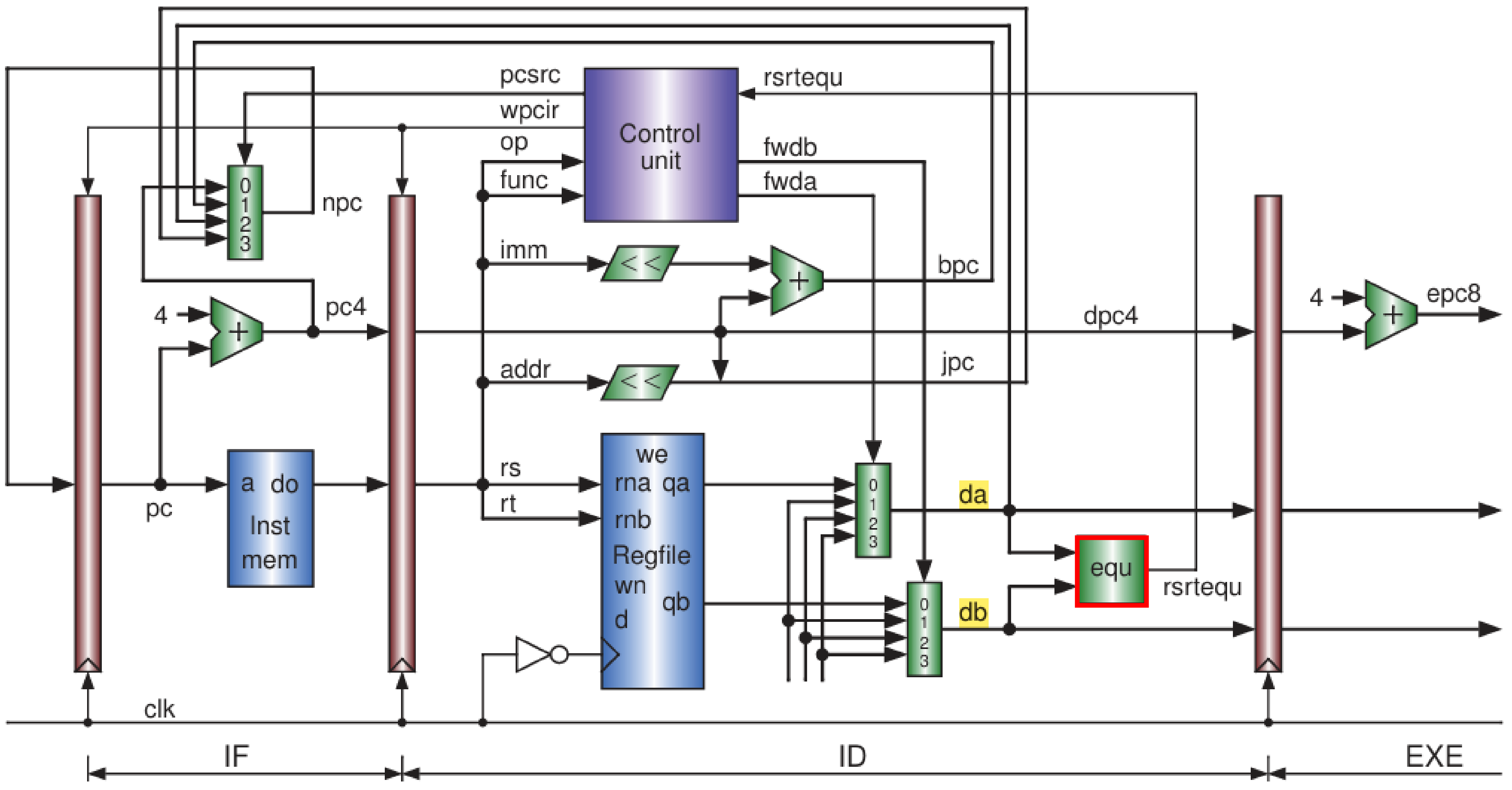


Control Hazards and Delayed Branch

- The **control hazards** occurs when a pipeline CPU executes a branch or jump instruction.
- **Delayed branch** is one method to deal with the control hazards.
- MIPS ISA adopts a <u>one-delay-slot</u> mechanism.
  - ✓The instruction located in **delay slot** is always executed no matter what.
- The return address of *jal* is PC+8 (delayed branch).



Control Hazards and Delayed Branch

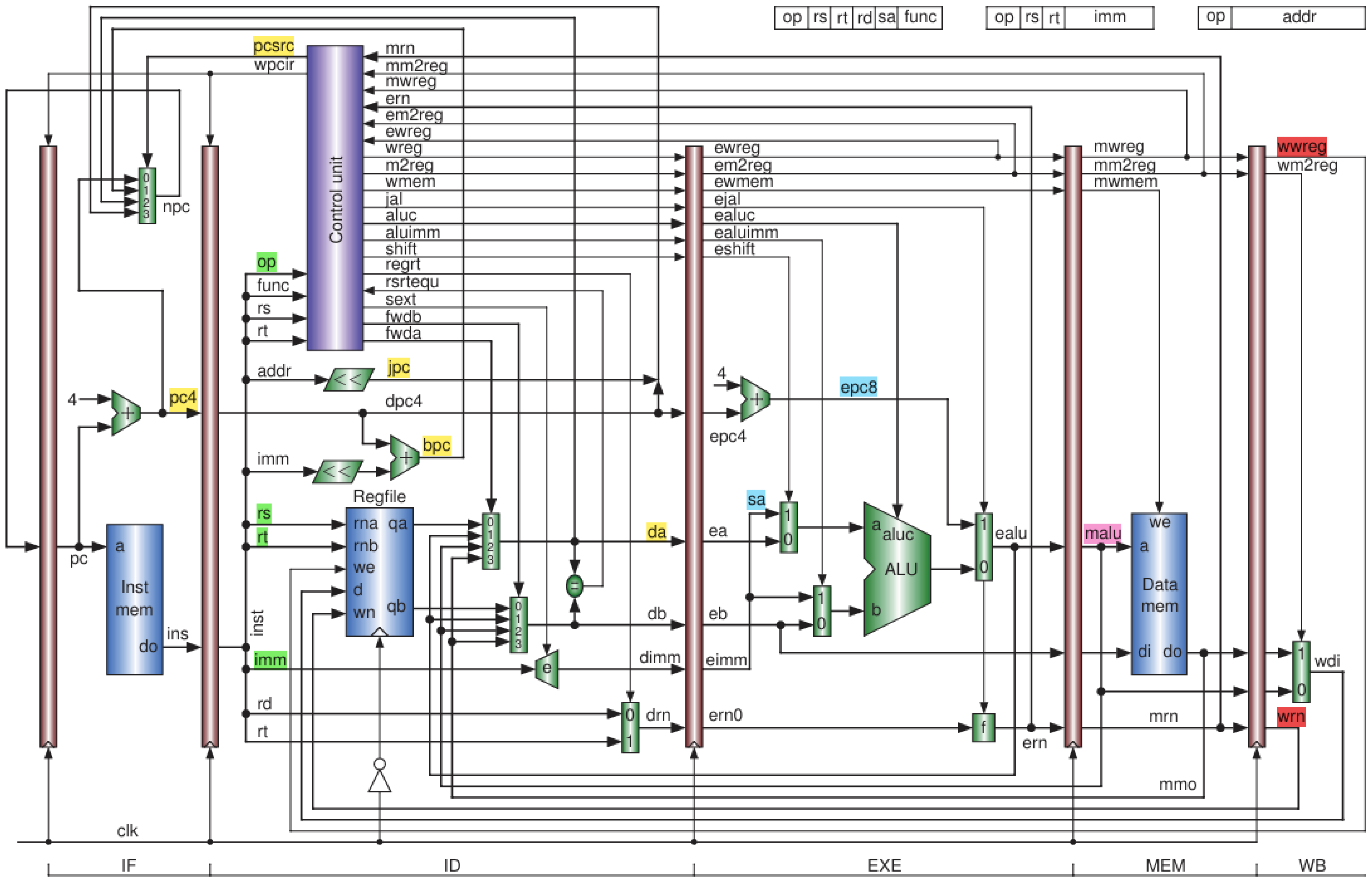


Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

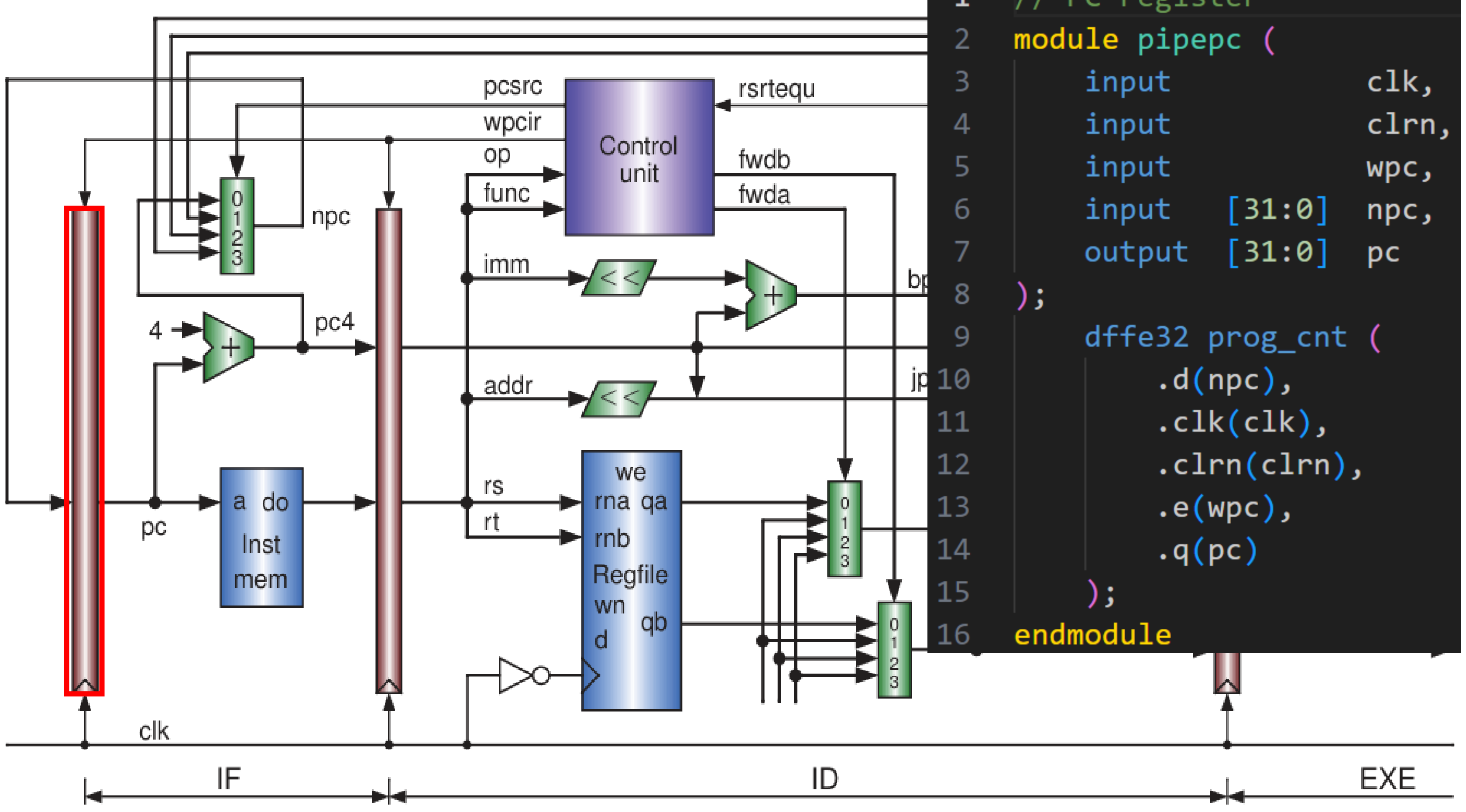
Verilog HDL Implementation

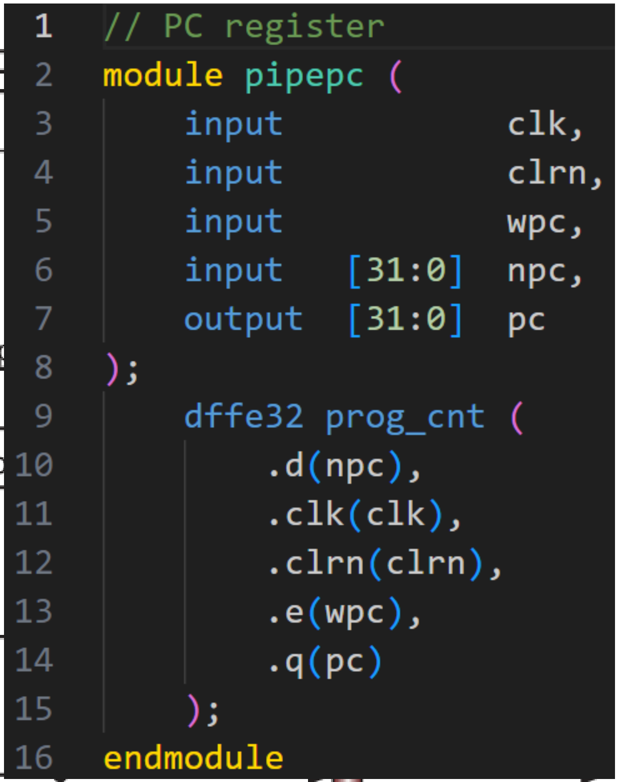
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

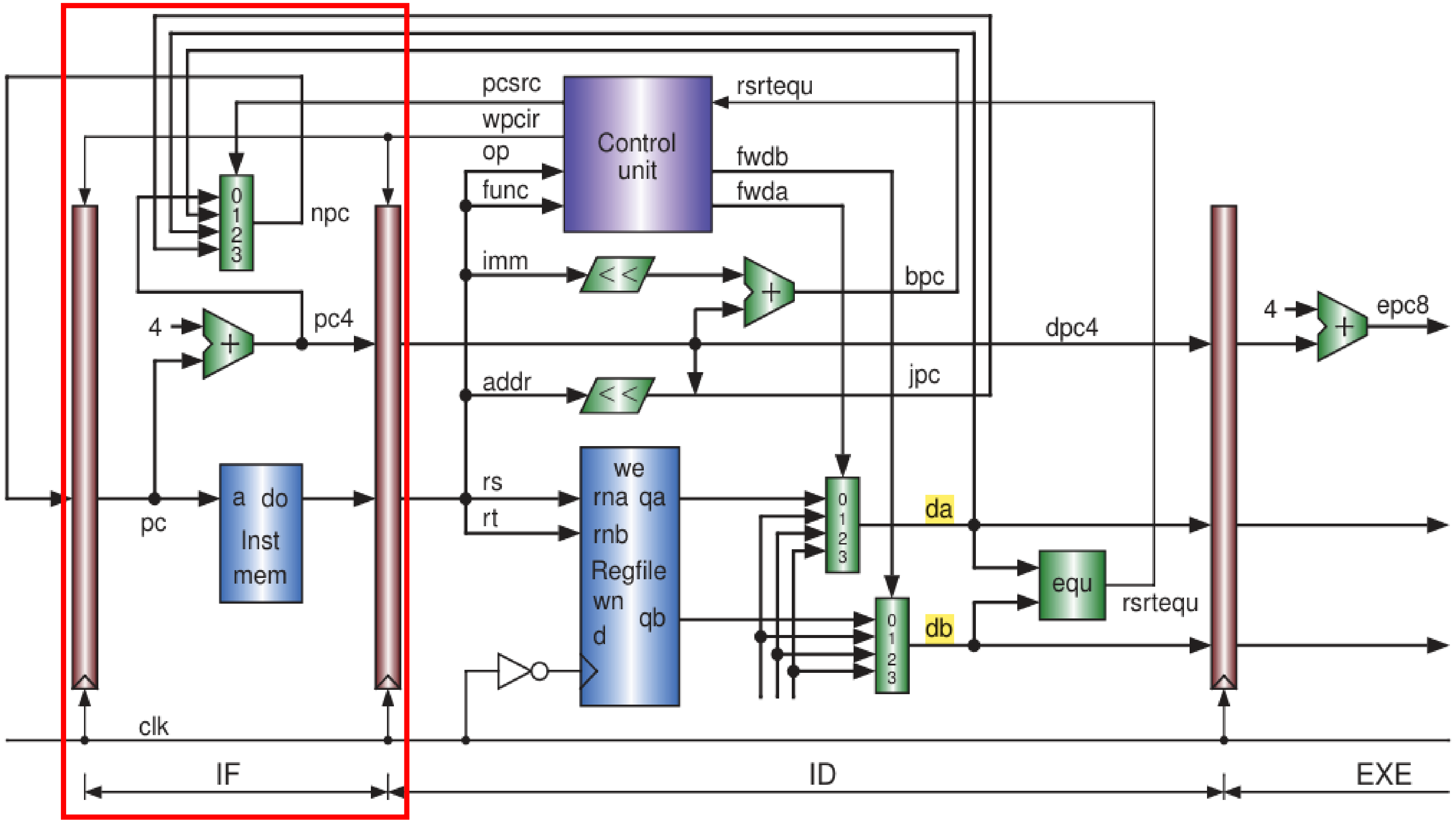



```verilog
// PC register
module pipepc (
    input           clk,
    input           clrn,
    input           wpc,
    input   [31:0]  npc,
    output  [31:0]  pc
);
    dffe32 prog_cnt (
        .d(npc),
        .clk(clk),
        .clrn(clrn),
        .e(wpc),
        .q(pc)
    );
endmodule
```

PC is considered as the first pipeline register

Verilog HDL Implementation

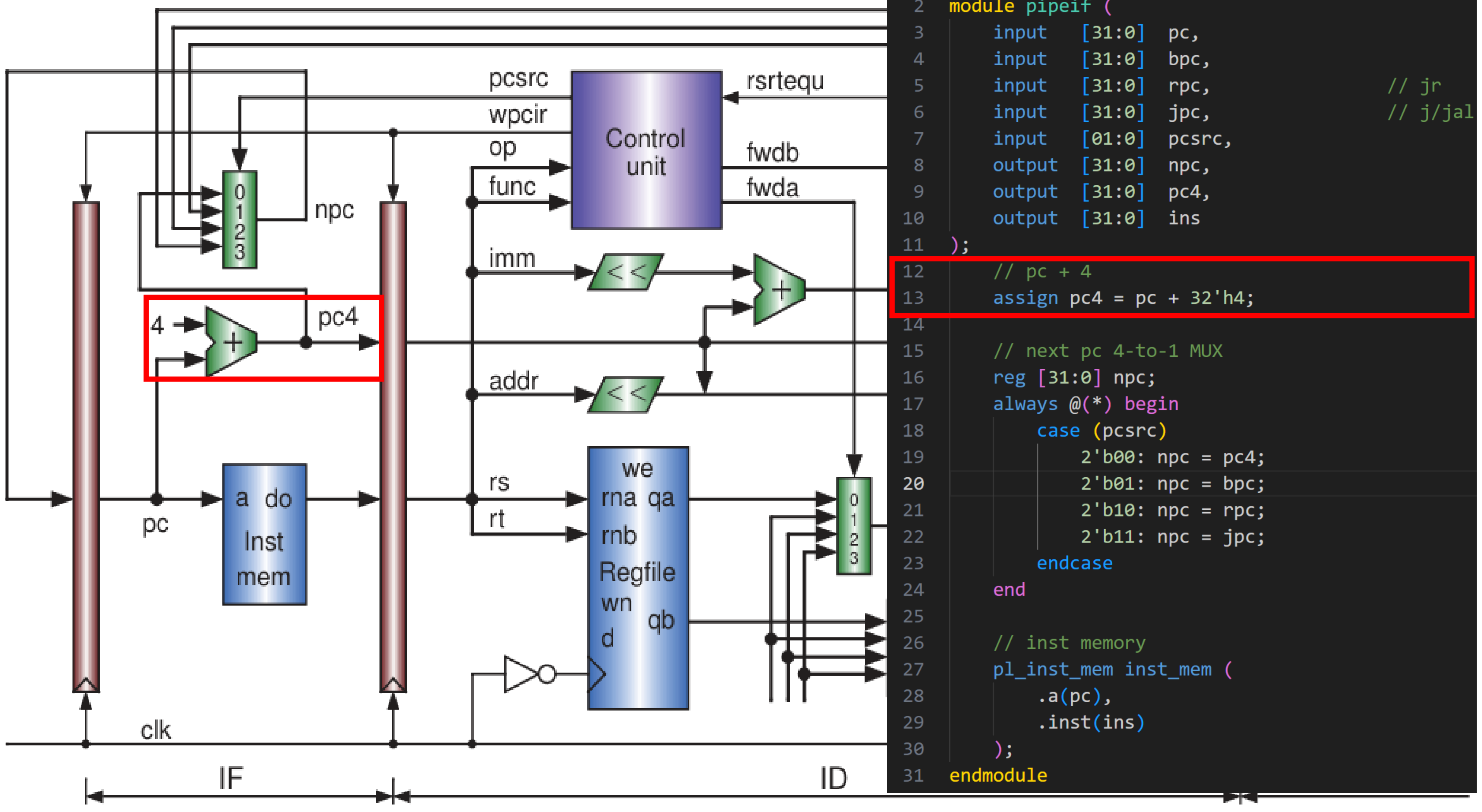
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

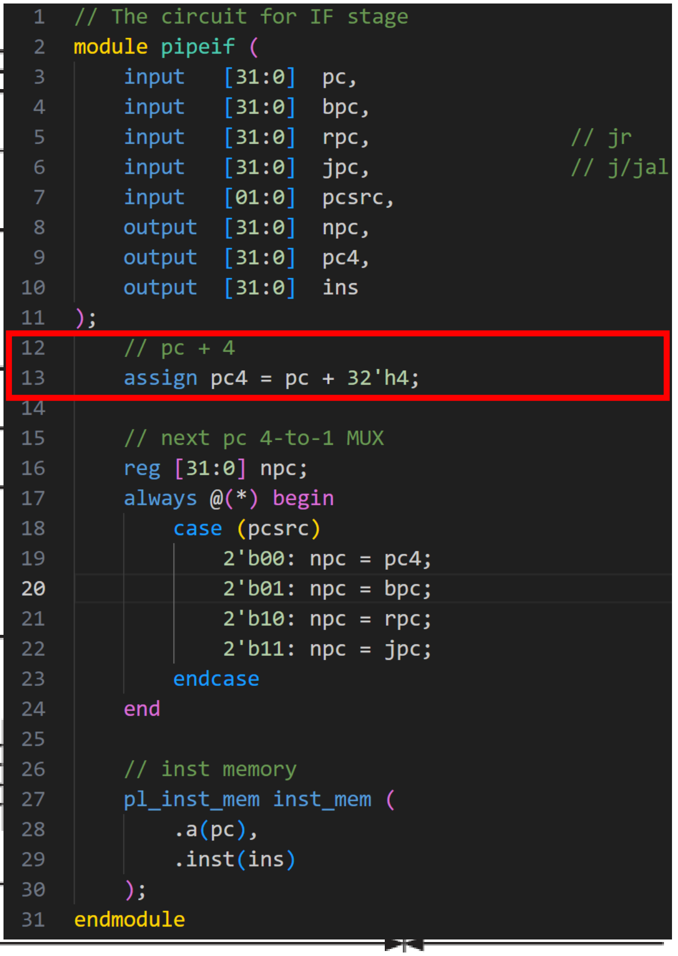


PC select MUX & PC+4 & Instruction Memory

Verilog HDL Implementation

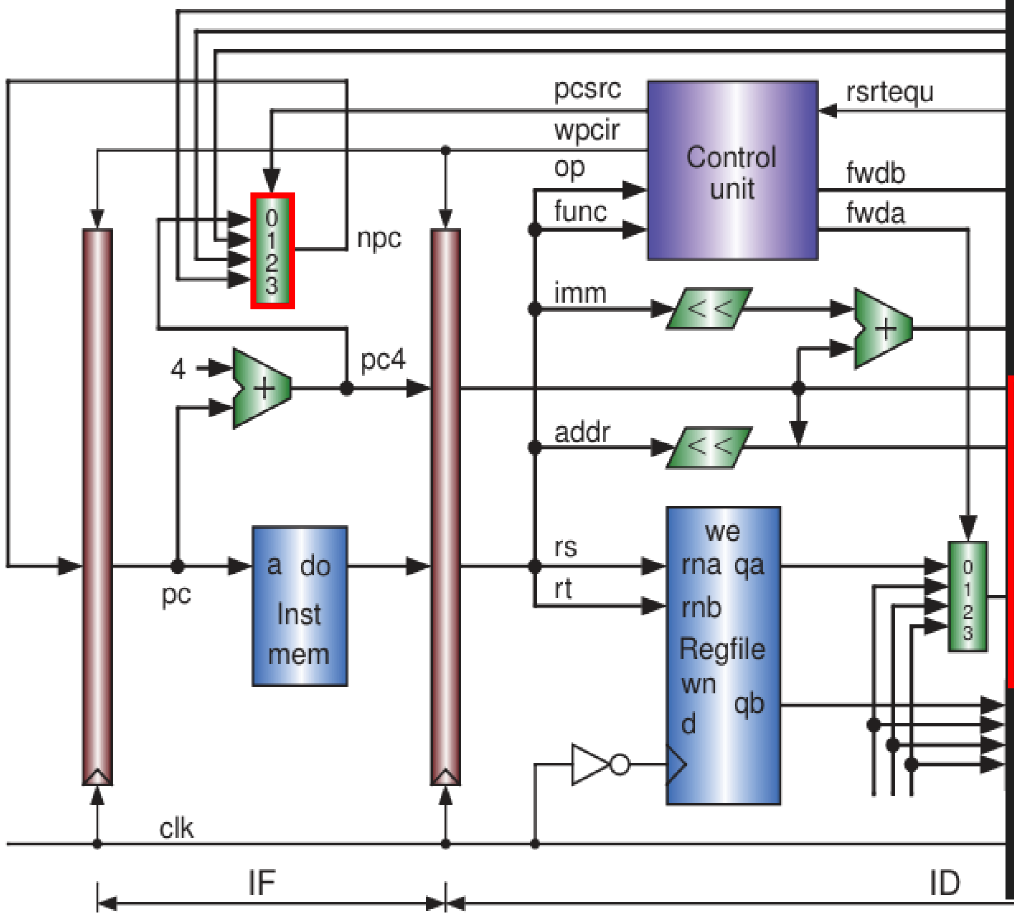
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

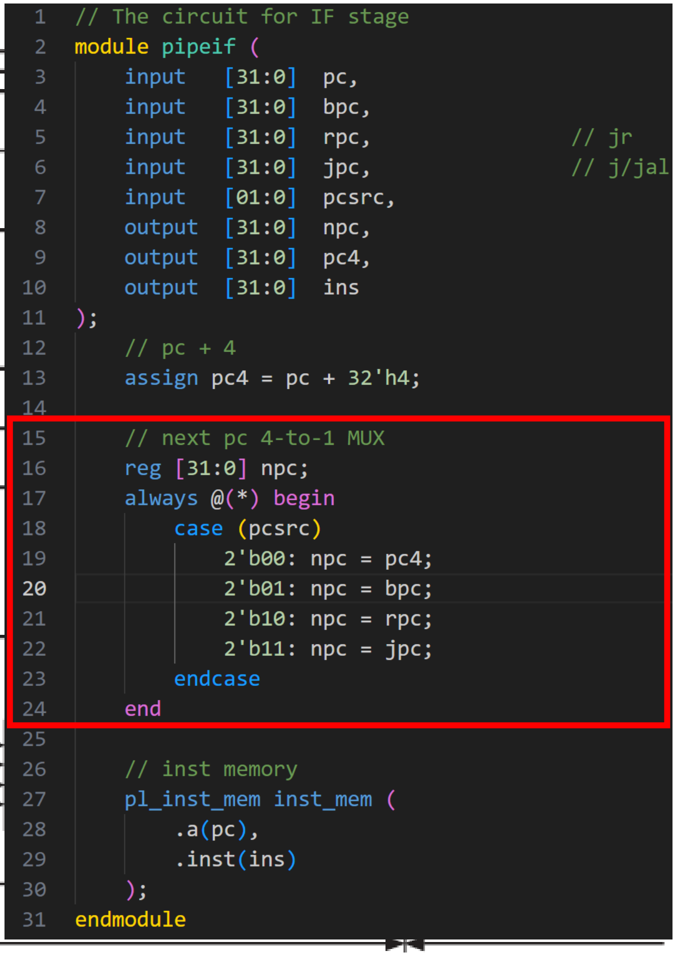



```
// The circuit for IF stage
module pipeif (
    input   [31:0]  pc,
    input   [31:0]  bpc,
    input   [31:0]  rpc,                // jr
    input   [31:0]  jpc,                // j/jal
    input   [01:0]  pcsrc,
    output  [31:0]  npc,
    output  [31:0]  pc4,
    output  [31:0]  ins
);
    // pc + 4
    assign pc4 = pc + 32'h4;

    // next pc 4-to-1 MUX
    reg [31:0] npc;
    always @(*) begin
        case (pcsrc)
            2'b00: npc = pc4;
            2'b01: npc = bpc;
            2'b10: npc = rpc;
            2'b11: npc = jpc;
        endcase
    end

    // inst memory
    pl_inst_mem inst_mem (
        .a(pc),
        .inst(ins)
    );
endmodule
```

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



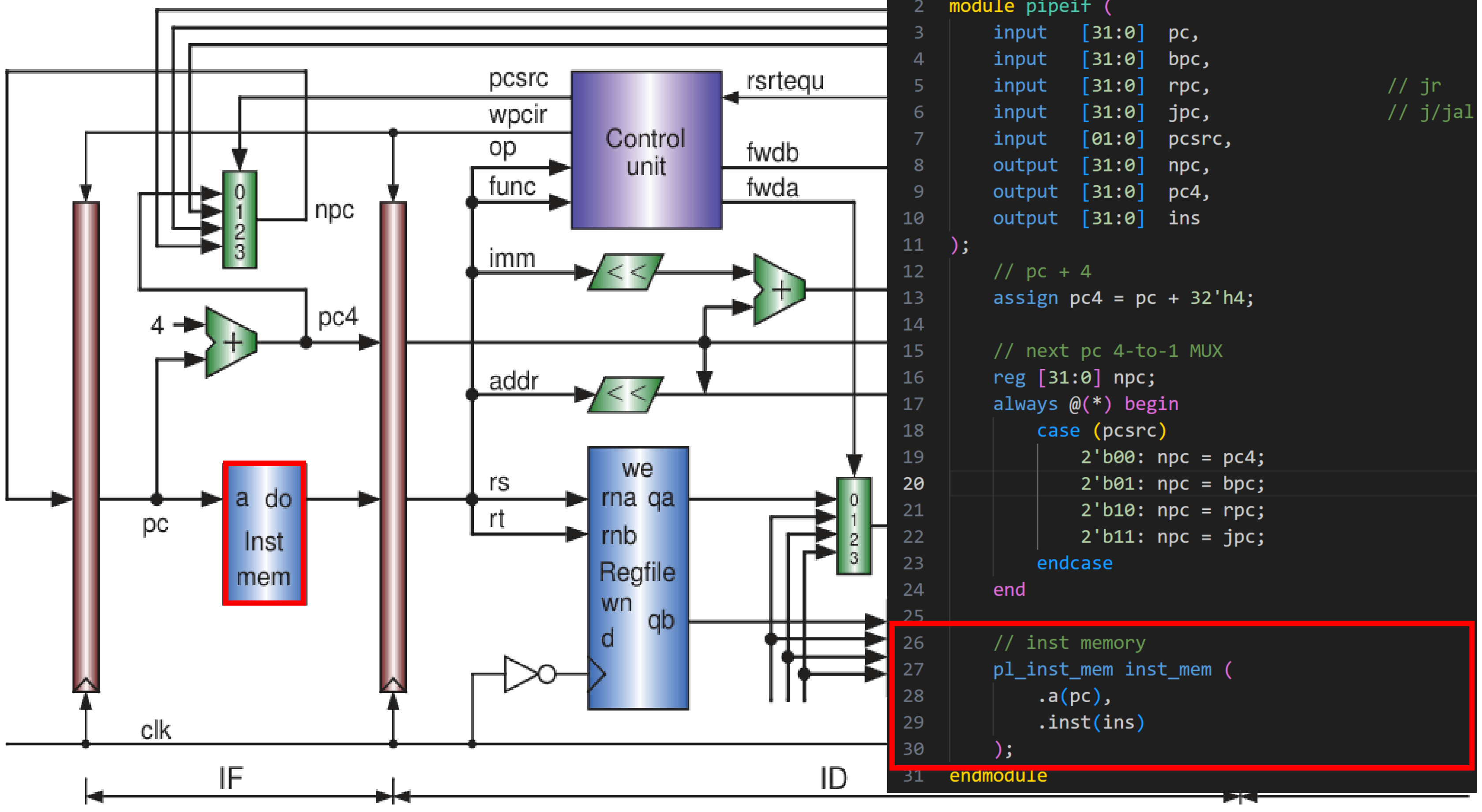


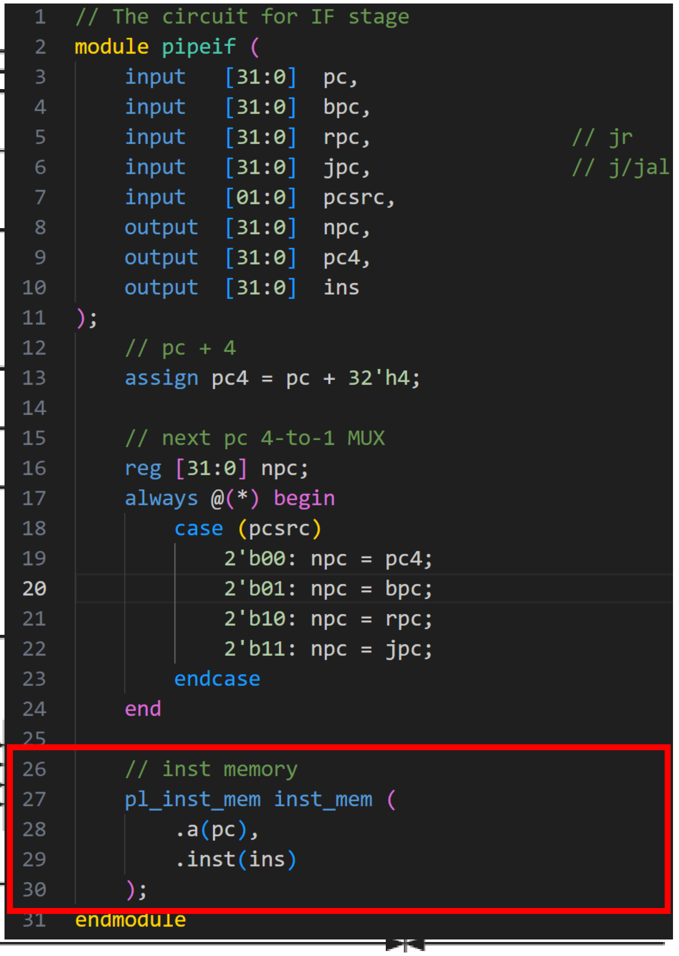
```verilog
// The circuit for IF stage
module pipeif (
    input   [31:0]  pc,
    input   [31:0]  bpc,
    input   [31:0]  rpc,                // jr
    input   [31:0]  jpc,                // j/jal
    input   [01:0]  pcsrc,
    output  [31:0]  npc,
    output  [31:0]  pc4,
    output  [31:0]  ins
);
    // pc + 4
    assign pc4 = pc + 32'h4;

    // next pc 4-to-1 MUX
    reg [31:0] npc;
    always @(*) begin
        case (pcsrc)
            2'b00: npc = pc4;
            2'b01: npc = bpc;
            2'b10: npc = rpc;
            2'b11: npc = jpc;
        endcase
    end

    // inst memory
    pl_inst_mem inst_mem (
        .a(pc),
        .inst(ins)
    );
endmodule
```

PC+4, Branch PC, Jump reg PC, Jump PC

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

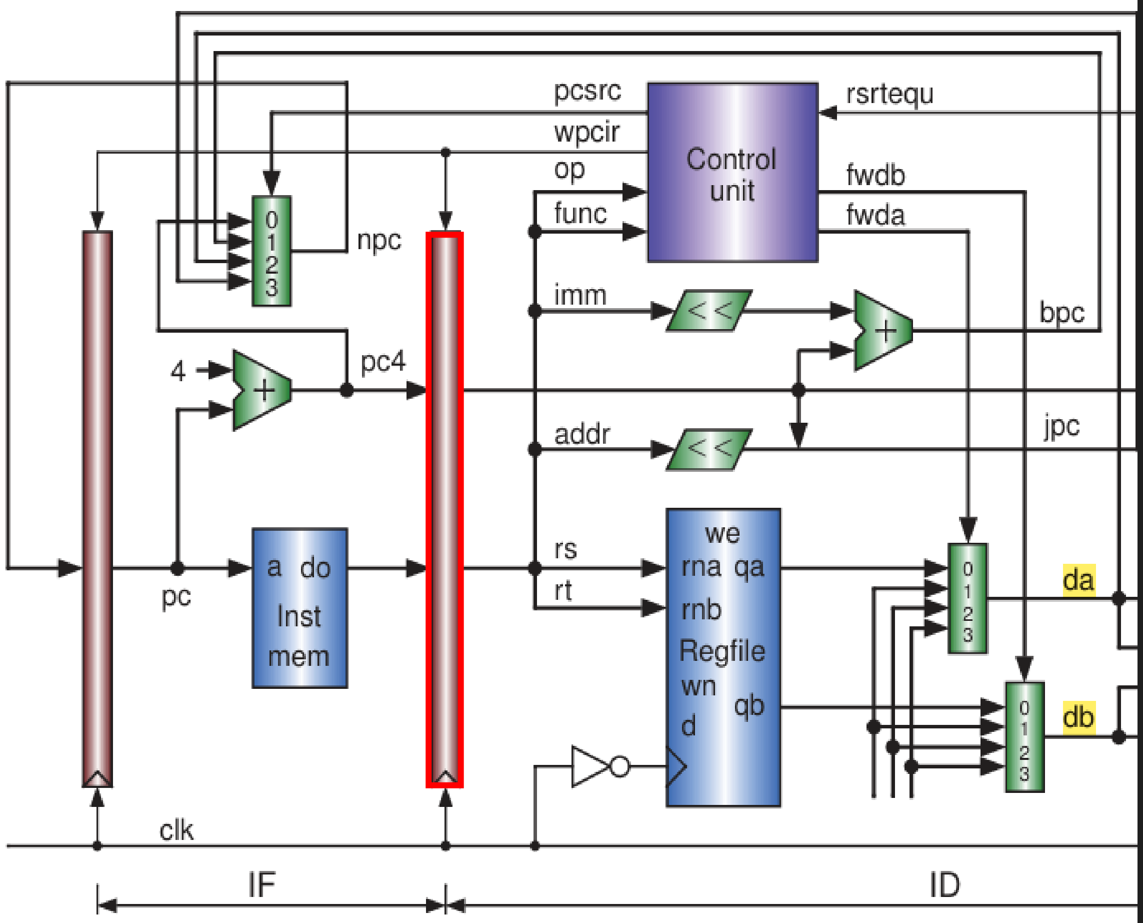


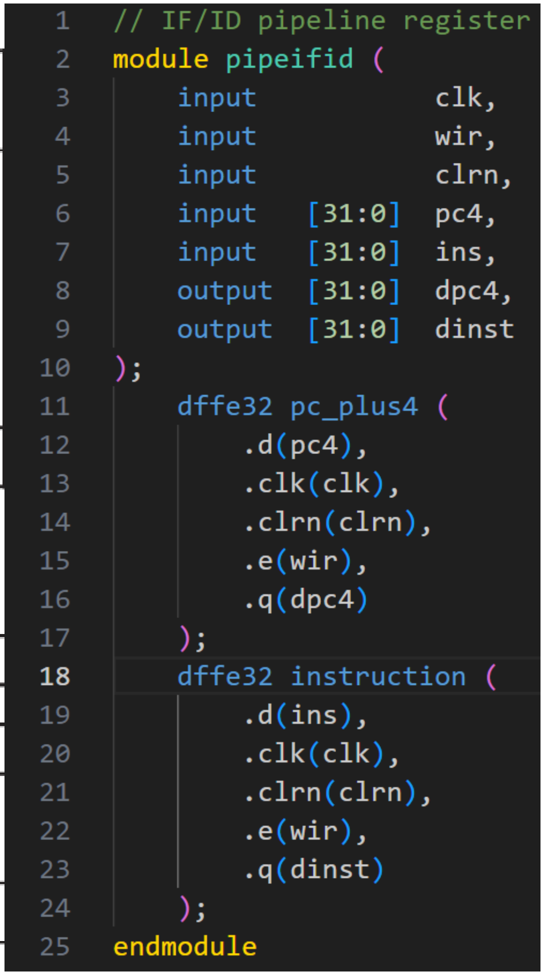


```verilog
// The circuit for IF stage
module pipeif (
    input   [31:0]  pc,
    input   [31:0]  bpc,
    input   [31:0]  rpc,                // jr
    input   [31:0]  jpc,                // j/jal
    input   [01:0]  pcsrc,
    output  [31:0]  npc,
    output  [31:0]  pc4,
    output  [31:0]  ins
);
    // pc + 4
    assign pc4 = pc + 32'h4;

    // next pc 4-to-1 MUX
    reg [31:0] npc;
    always @(*) begin
        case (pcsrc)
            2'b00: npc = pc4;
            2'b01: npc = bpc;
            2'b10: npc = rpc;
            2'b11: npc = jpc;
        endcase
    end

    // inst memory
    pl_inst_mem inst_mem (
        .a(pc),
        .inst(ins)
    );
endmodule
```

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



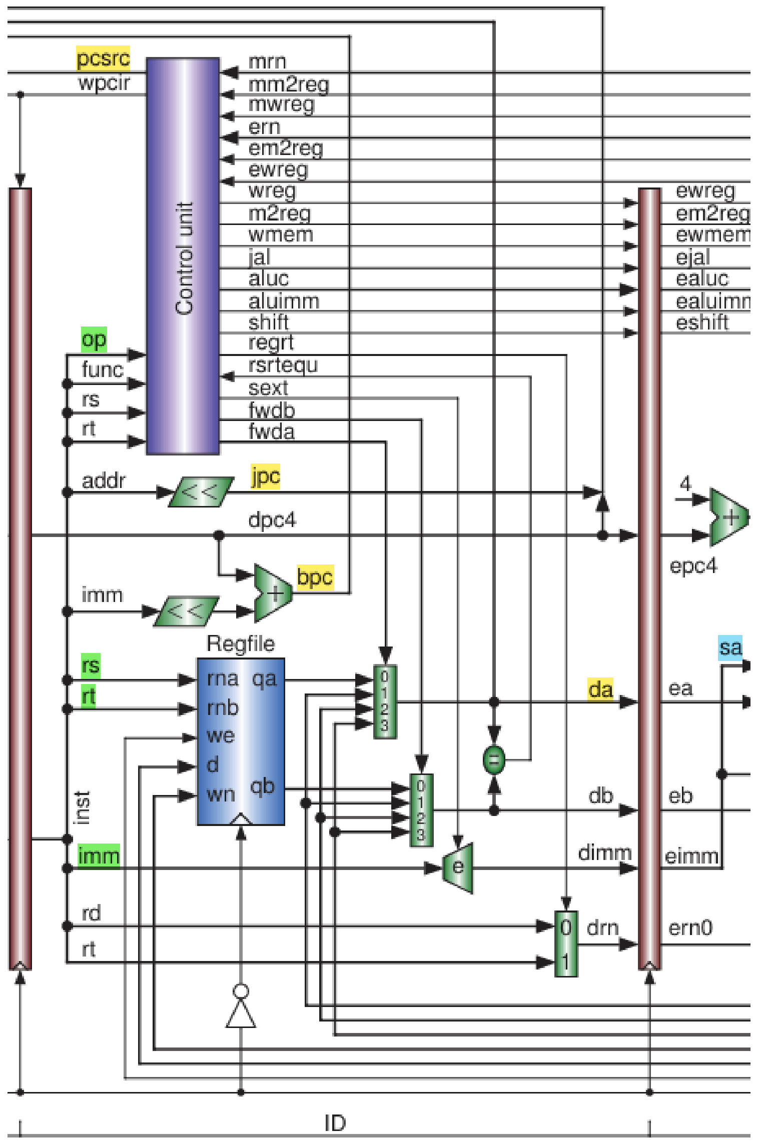


```verilog
// IF/ID pipeline register
module pipeifid (
    input           clk,
    input           wir,
    input           clrn,
    input   [31:0]  pc4,
    input   [31:0]  ins,
    output  [31:0]  dpc4,
    output  [31:0]  dinst
);
    dffe32 pc_plus4 (
        .d(pc4),
        .clk(clk),
        .clrn(clrn),
        .e(wir),
        .q(dpc4)
    );
    dffe32 instruction (
        .d(ins),
        .clk(clk),
        .clrn(clrn),
        .e(wir),
        .q(dinst)
    );
endmodule
```

The pipeline register between IF and ID

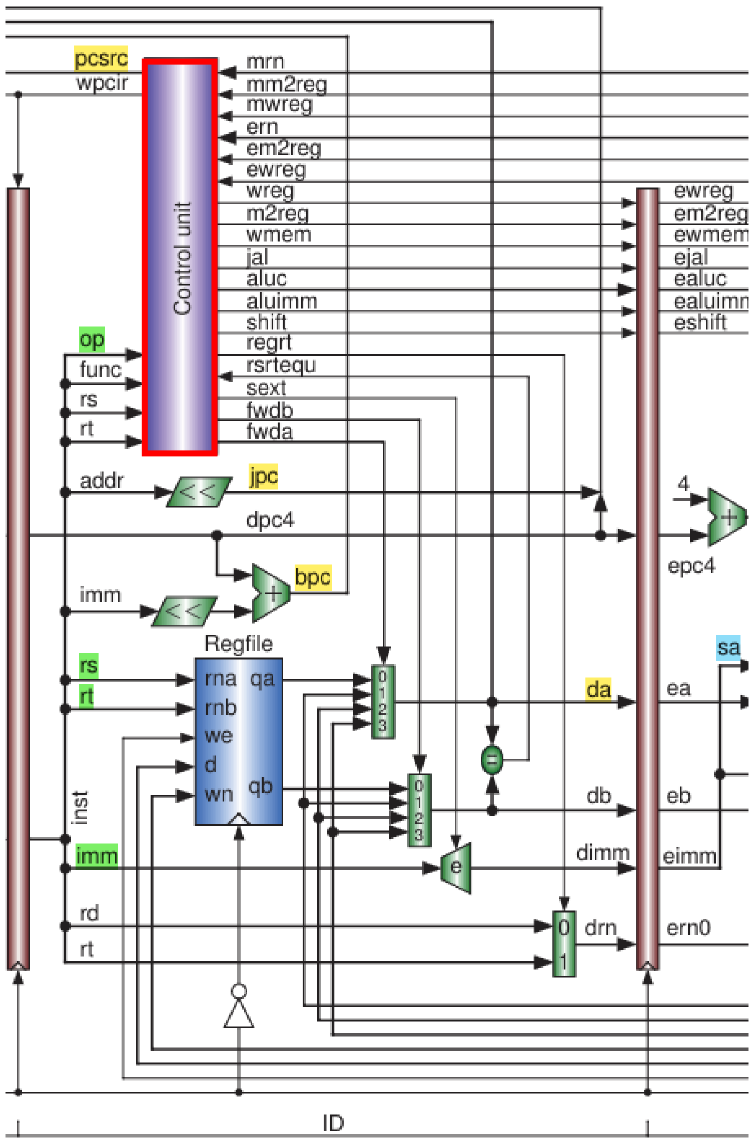
Verilog HDL Implementation

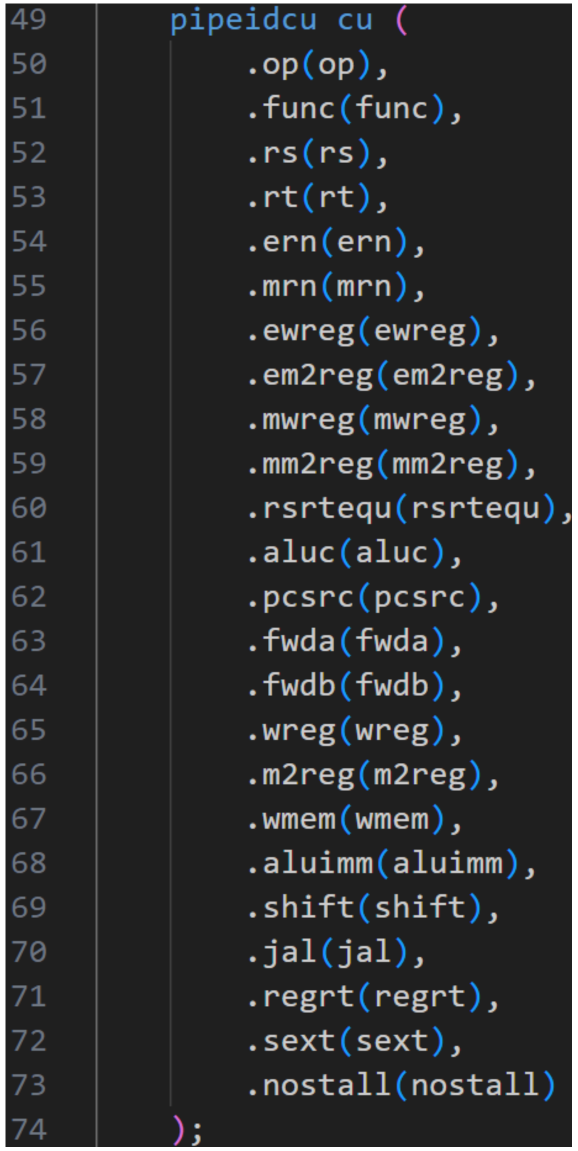
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
  - ✓pipeidcu.v
  - ✓regfile.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



Verilog HDL Implementation

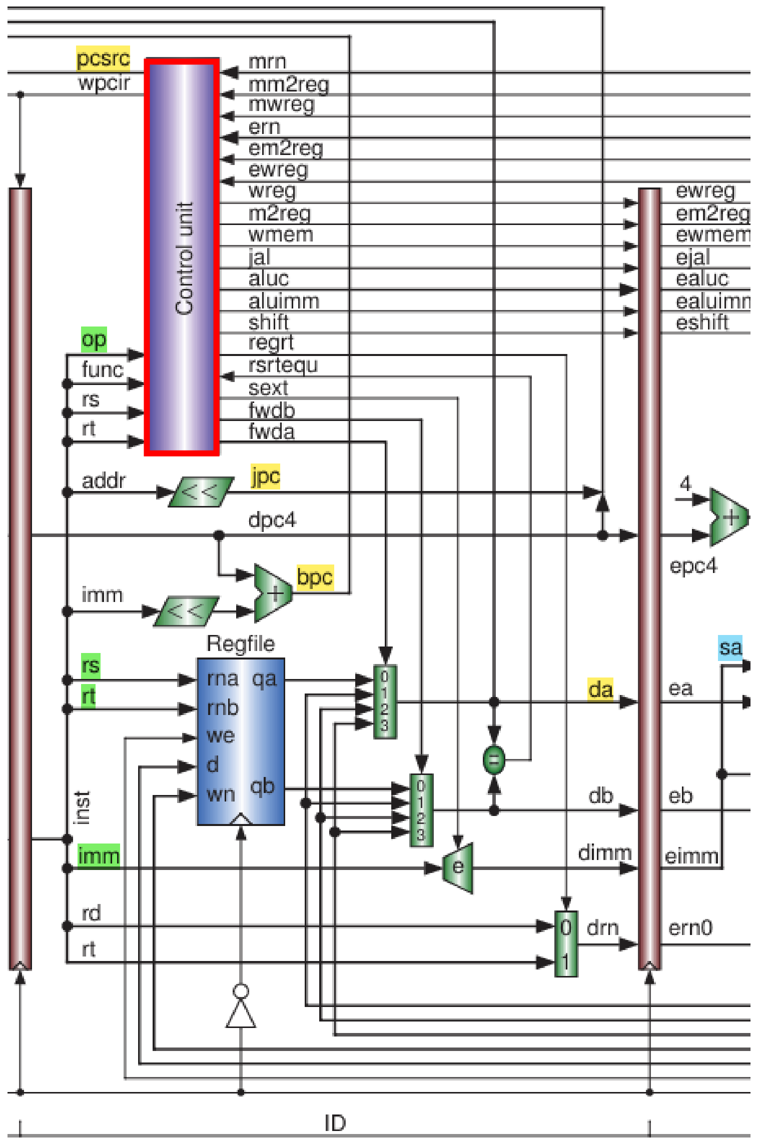
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
    - ✓pipeidcu.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

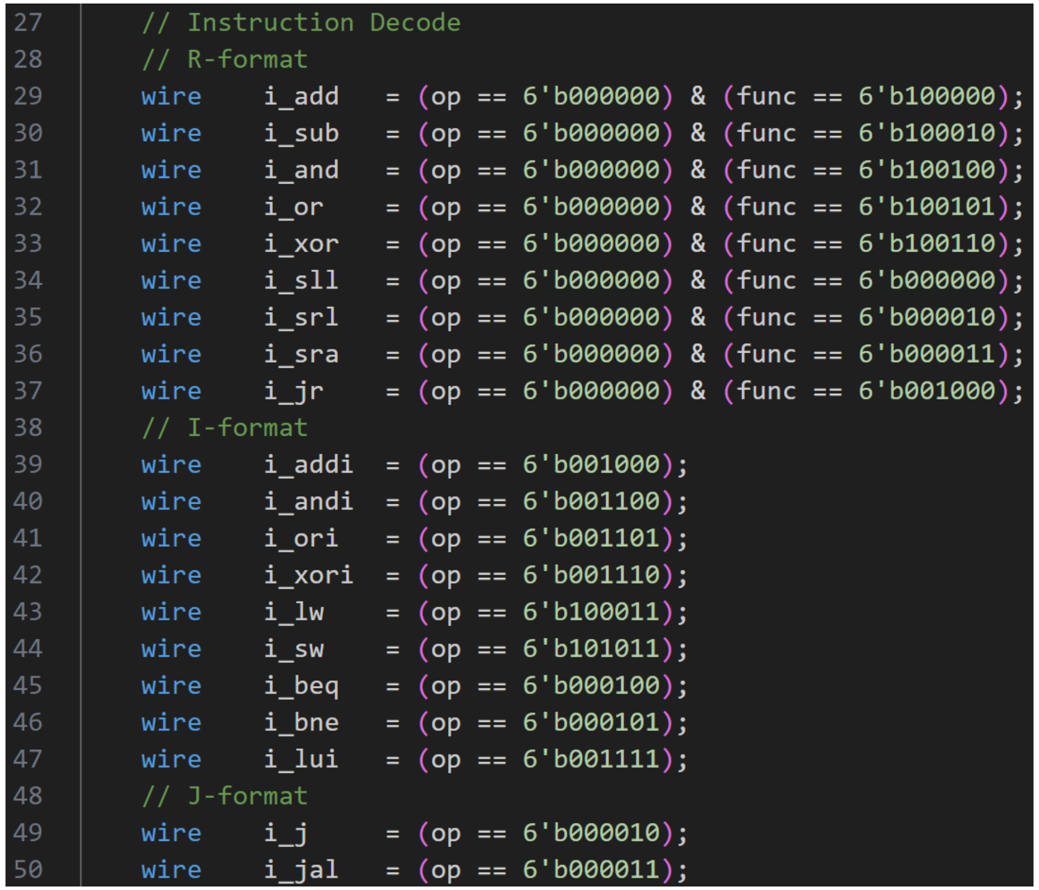



```
        pipeidcu cu (
            .op(op),
            .func(func),
            .rs(rs),
            .rt(rt),
            .ern(ern),
            .mrn(mrn),
            .ewreg(ewreg),
            .em2reg(em2reg),
            .mwreg(mwreg),
            .mm2reg(mm2reg),
            .rsrtequ(rsrtequ),
            .aluc(aluc),
            .pcsrc(pcsrc),
            .fwda(fwda),
            .fwdb(fwdb),
            .wreg(wreg),
            .m2reg(m2reg),
            .wmem(wmem),
            .aluimm(aluimm),
            .shift(shift),
            .jal(jal),
            .regrt(regrt),
            .sext(sext),
            .nostall(nostall)
        );
```

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
  - ✓pipeidcu.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

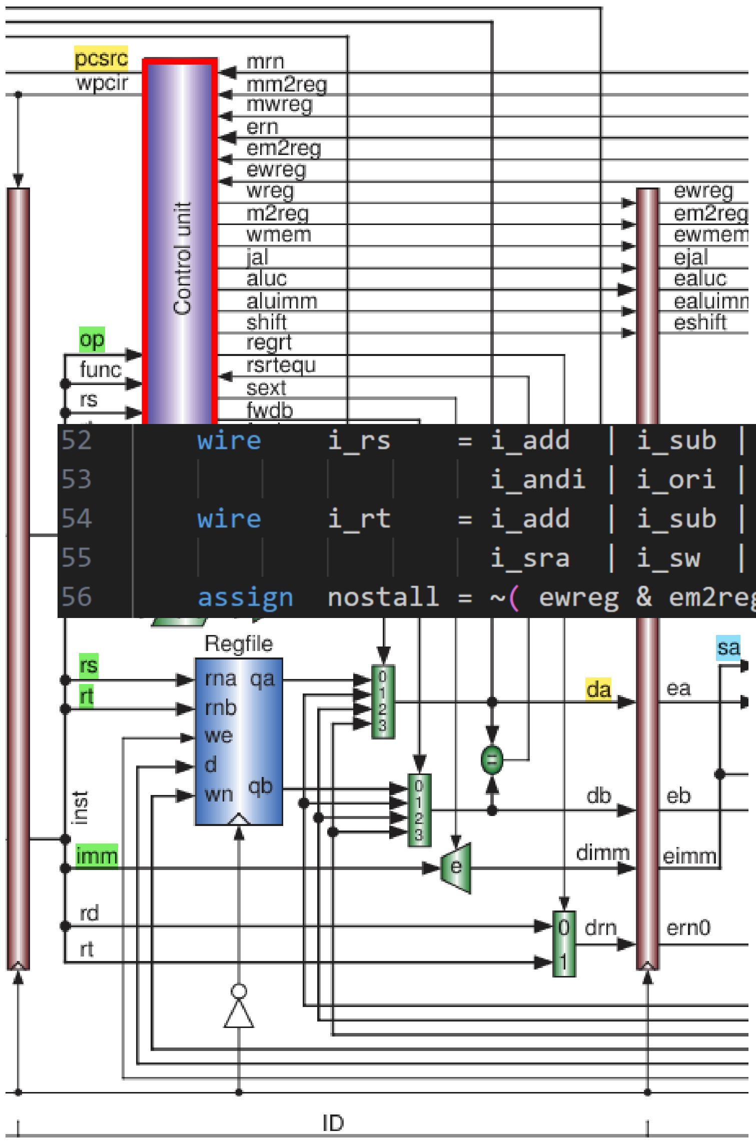


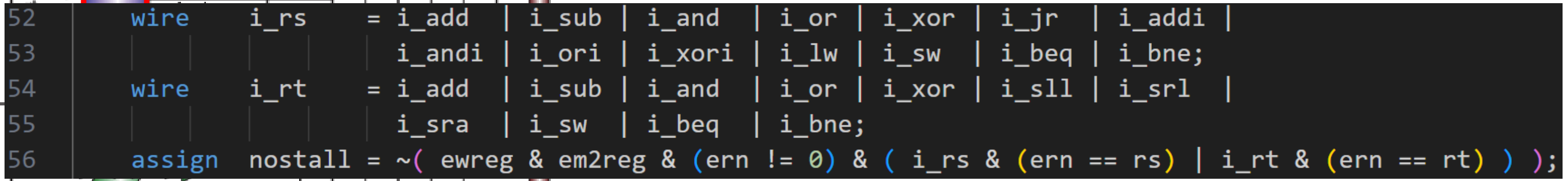

```verilog
    // Instruction Decode
    // R-format
    wire    i_add   = (op == 6'b000000) & (func == 6'b100000);
    wire    i_sub   = (op == 6'b000000) & (func == 6'b100010);
    wire    i_and   = (op == 6'b000000) & (func == 6'b100100);
    wire    i_or    = (op == 6'b000000) & (func == 6'b100101);
    wire    i_xor   = (op == 6'b000000) & (func == 6'b100110);
    wire    i_sll   = (op == 6'b000000) & (func == 6'b000000);
    wire    i_srl   = (op == 6'b000000) & (func == 6'b000010);
    wire    i_sra   = (op == 6'b000000) & (func == 6'b000011);
    wire    i_jr    = (op == 6'b000000) & (func == 6'b001000);
    // I-format
    wire    i_addi  = (op == 6'b001000);
    wire    i_andi  = (op == 6'b001100);
    wire    i_ori   = (op == 6'b001101);
    wire    i_xori  = (op == 6'b001110);
    wire    i_lw    = (op == 6'b100011);
    wire    i_sw    = (op == 6'b101011);
    wire    i_beq   = (op == 6'b000100);
    wire    i_bne   = (op == 6'b000101);
    wire    i_lui   = (op == 6'b001111);
    // J-format
    wire    i_j     = (op == 6'b000010);
    wire    i_jal   = (op == 6'b000011);
```

Instruction decode

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
  - ✓pipeidcu.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

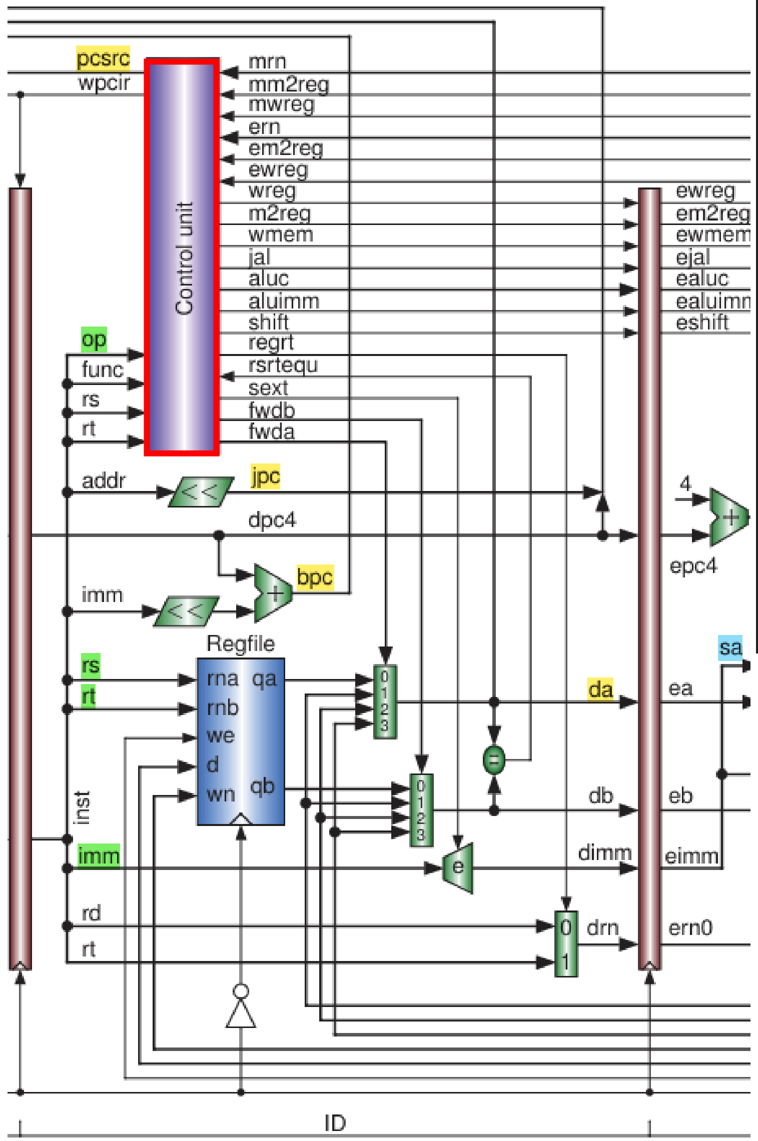


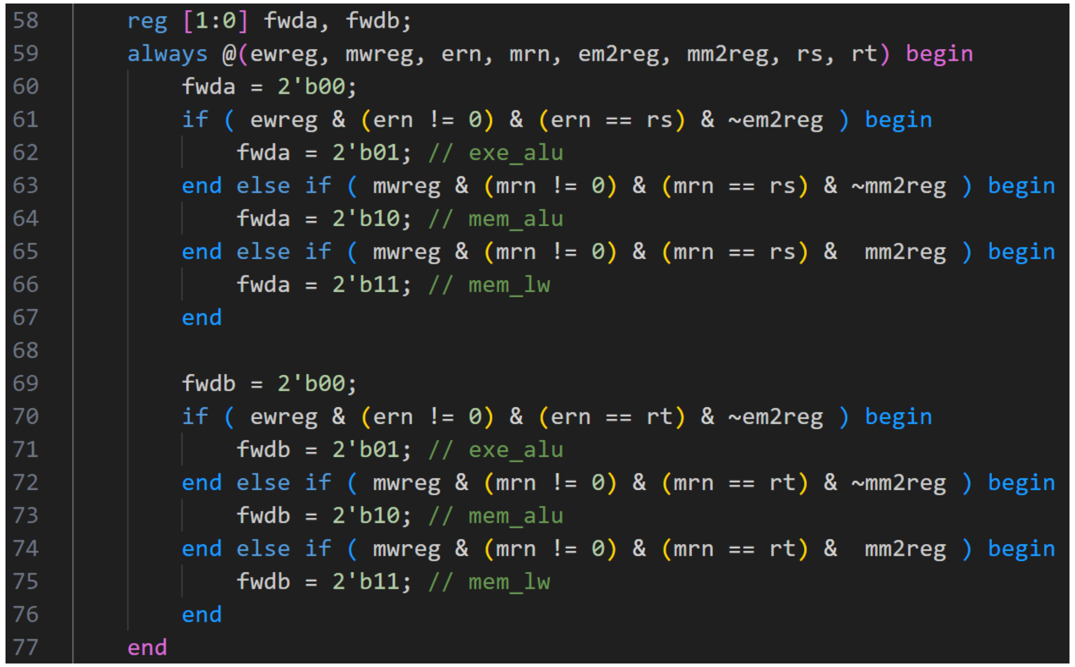

```
wire    i_rs    = i_add  | i_sub | i_and  | i_or | i_xor | i_jr  | i_addi |
                  i_andi | i_ori | i_xori | i_lw | i_sw  | i_beq | i_bne;
wire    i_rt    = i_add  | i_sub | i_and  | i_or | i_xor | i_sll | i_srl  |
                  i_sra  | i_sw  | i_beq  | i_bne;
assign  nostall = ~( ewreg & em2reg & (ern != 0) & ( i_rs & (ern == rs) | i_rt & (ern == rt) ) );
```

Stall when instruction in ID stage need the same register used in EXE stage in pipeline when RF write signals in EXE stage is set.

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
    - ✓pipeidcu.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



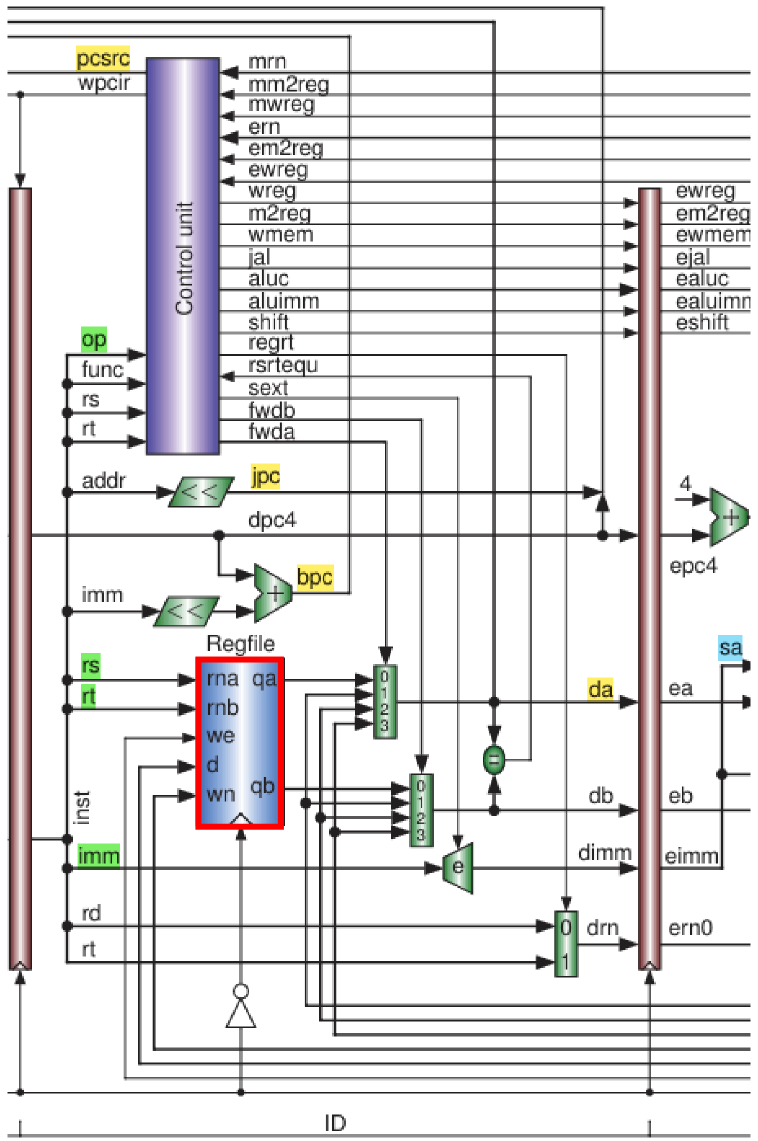


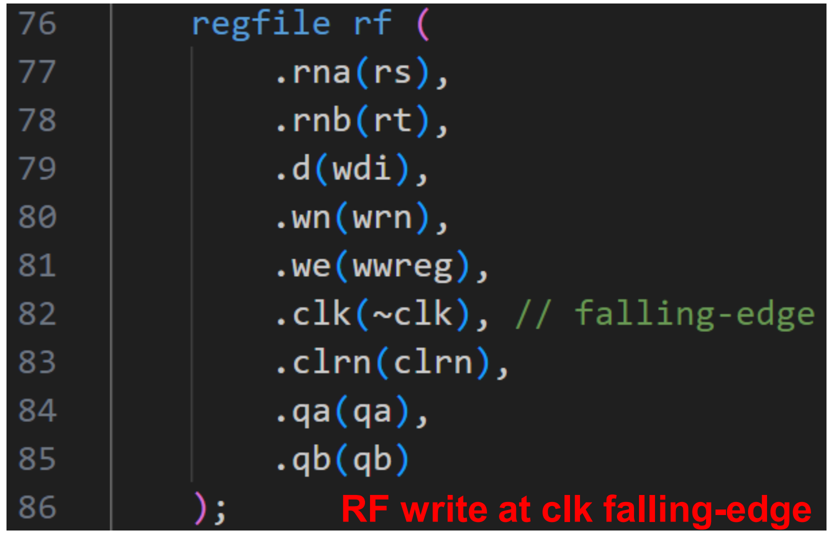
```verilog
    reg [1:0] fwda, fwdb;
    always @(ewreg, mwreg, ern, mrn, em2reg, mm2reg, rs, rt) begin
        fwda = 2'b00;
        if ( ewreg & (ern != 0) & (ern == rs) & ~em2reg ) begin
            fwda = 2'b01; // exe_alu
        end else if ( mwreg & (mrn != 0) & (mrn == rs) & ~mm2reg ) begin
            fwda = 2'b10; // mem_alu
        end else if ( mwreg & (mrn != 0) & (mrn == rs) &  mm2reg ) begin
            fwda = 2'b11; // mem_lw
        end

        fwdb = 2'b00;
        if ( ewreg & (ern != 0) & (ern == rt) & ~em2reg ) begin
            fwdb = 2'b01; // exe_alu
        end else if ( mwreg & (mrn != 0) & (mrn == rt) & ~mm2reg ) begin
            fwdb = 2'b10; // mem_alu
        end else if ( mwreg & (mrn != 0) & (mrn == rt) &  mm2reg ) begin
            fwdb = 2'b11; // mem_lw
        end
    end
```

Internal forwarding control signal is based on register number and write signals in EXE and MEM stage.

Verilog HDL Implementation

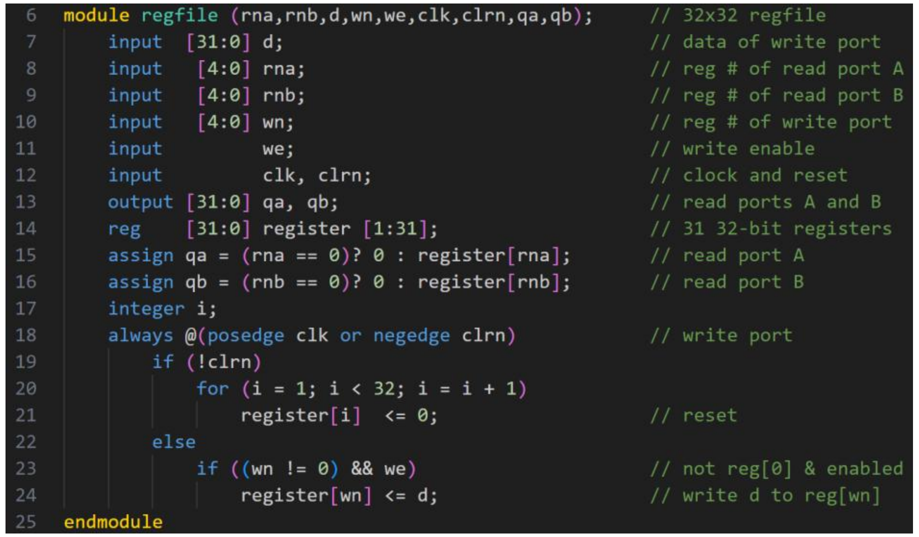
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
  - ✓regfile.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v





```verilog
    regfile rf (
        .rna(rs),
        .rnb(rt),
        .d(wdi),
        .wn(wrn),
        .we(wwreg),
        .clk(~clk), // falling-edge
        .clrn(clrn),
        .qa(qa),
        .qb(qb)
    );
```

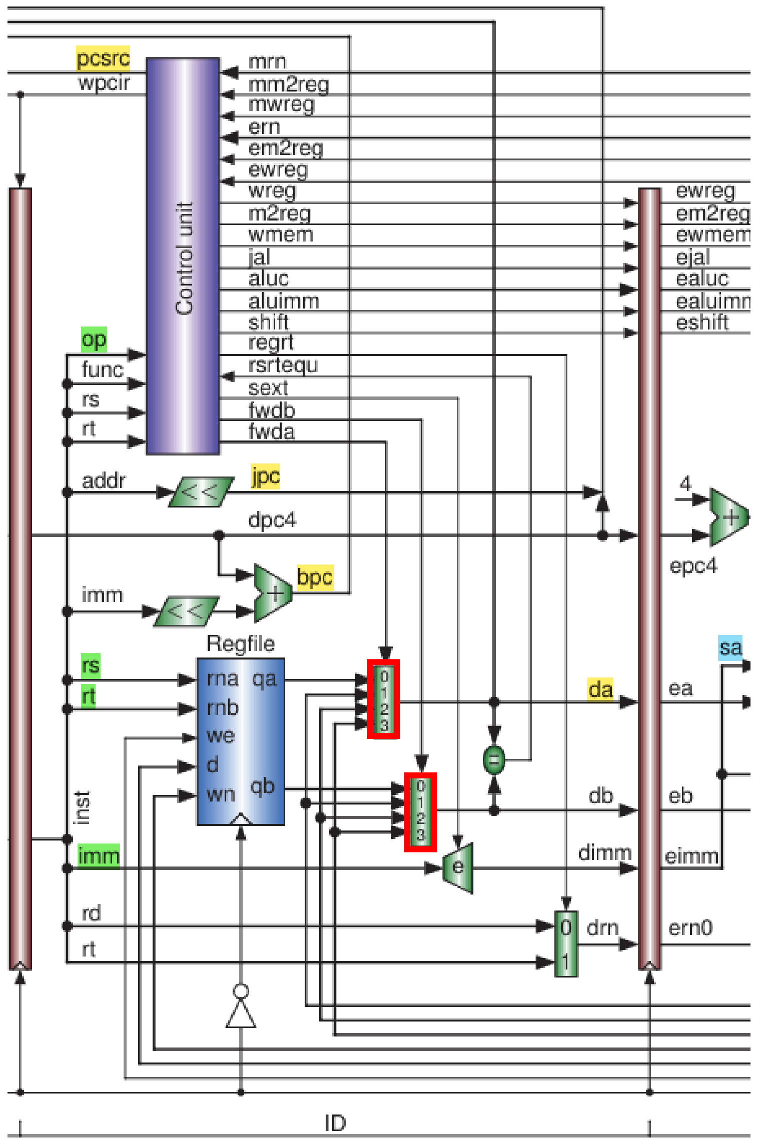
RF write at clk falling-edge

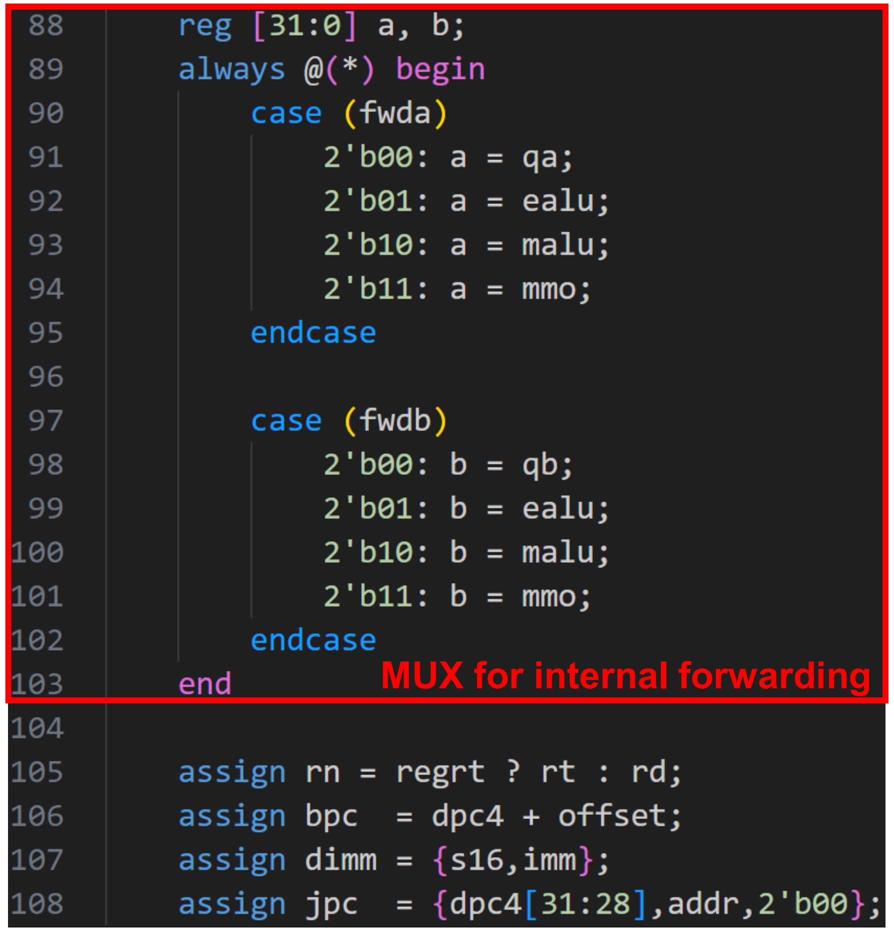


```verilog
module regfile (rna,rnb,d,wn,we,clk,clrn,qa,qb);           // 32x32 regfile
    input  [31:0] d;                                        // data of write port
    input   [4:0] rna;                                      // reg # of read port A
    input   [4:0] rnb;                                      // reg # of read port B
    input   [4:0] wn;                                       // reg # of write port
    input         we;                                       // write enable
    input         clk, clrn;                                // clock and reset
    output [31:0] qa, qb;                                   // read ports A and B
    reg    [31:0] register [1:31];                          // 31 32-bit registers
    assign qa = (rna == 0)? 0 : register[rna];              // read port A
    assign qb = (rnb == 0)? 0 : register[rnb];              // read port B
    integer i;
    always @(posedge clk or negedge clrn)                   // write port
        if (!clrn)
            for (i = 1; i < 32; i = i + 1)
                register[i]  <= 0;                          // reset
        else
            if ((wn != 0) && we)                            // not reg[0] & enabled
                register[wn] <= d;                          // write d to reg[wn]
endmodule
```

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipeexe.v
- pipeexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



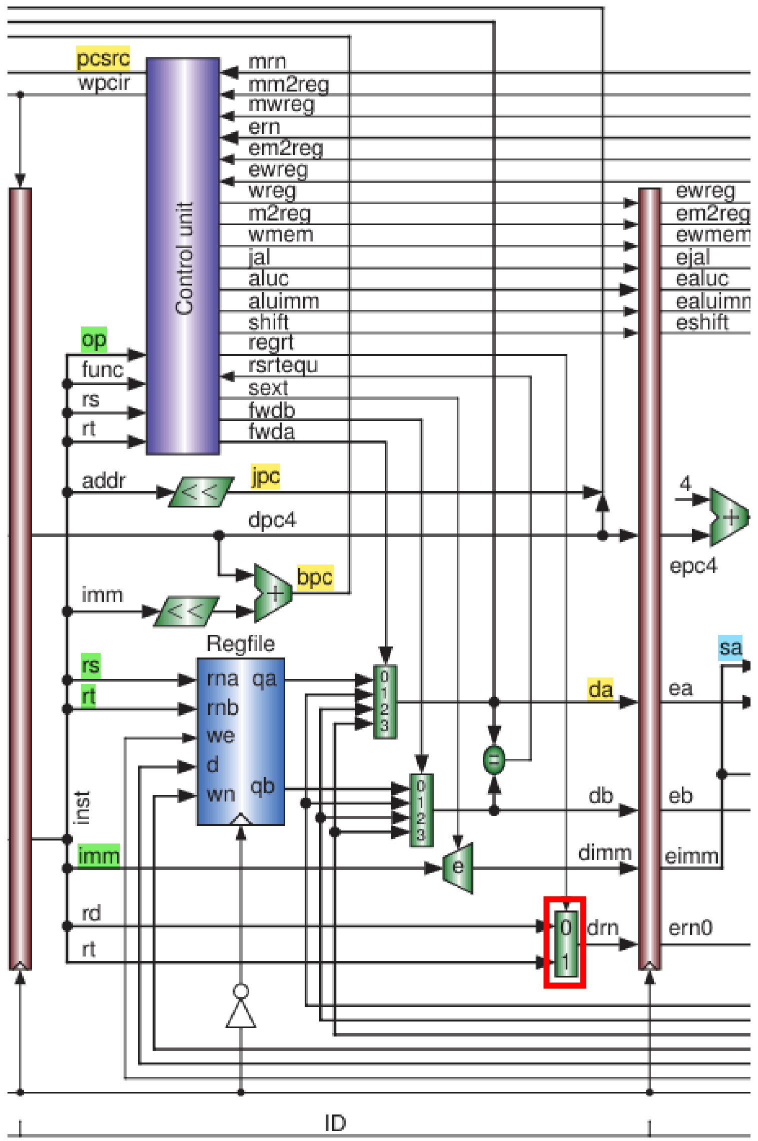


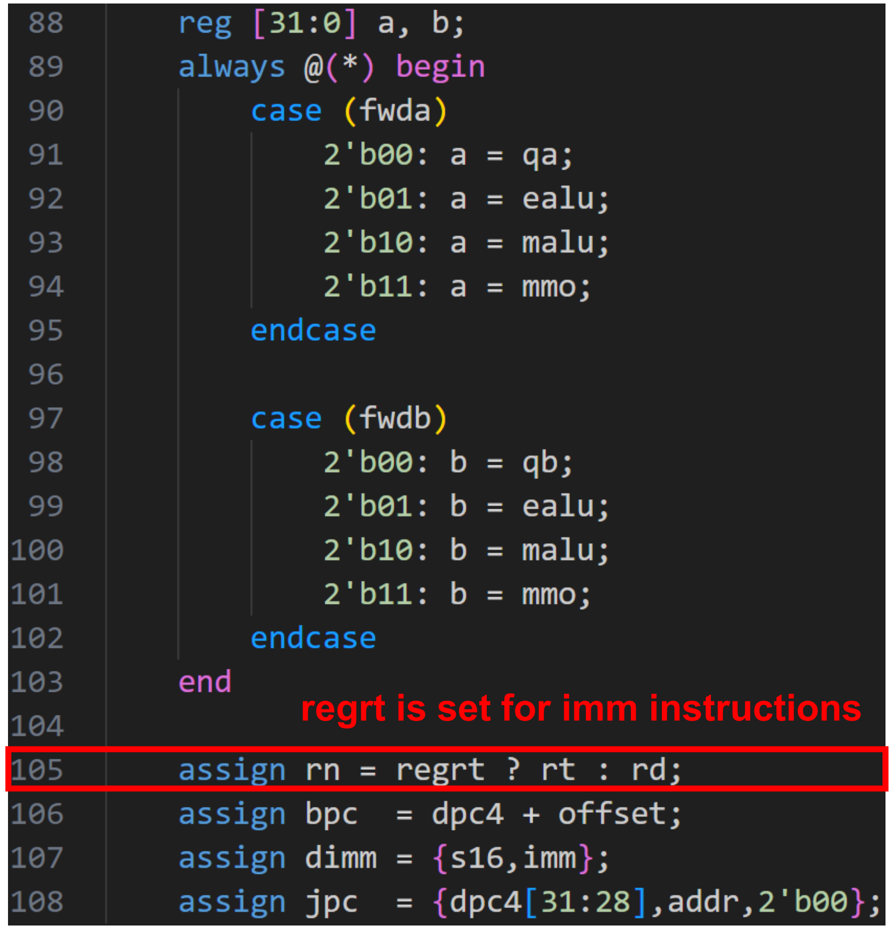
```verilog
    reg [31:0] a, b;
    always @(*) begin
        case (fwda)
            2'b00: a = qa;
            2'b01: a = ealu;
            2'b10: a = malu;
            2'b11: a = mmo;
        endcase

        case (fwdb)
            2'b00: b = qb;
            2'b01: b = ealu;
            2'b10: b = malu;
            2'b11: b = mmo;
        endcase
    end

    assign rn = regrt ? rt : rd;
    assign bpc  = dpc4 + offset;
    assign dimm = {s16,imm};
    assign jpc  = {dpc4[31:28],addr,2'b00};
```

MUX for internal forwarding

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



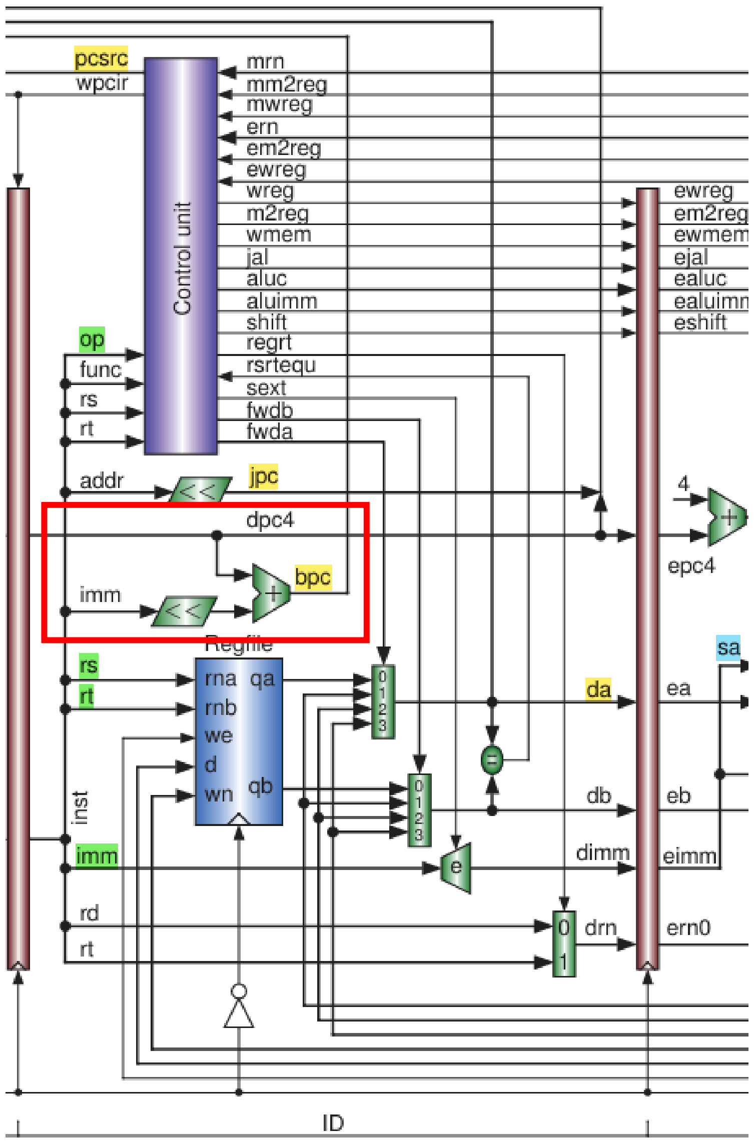


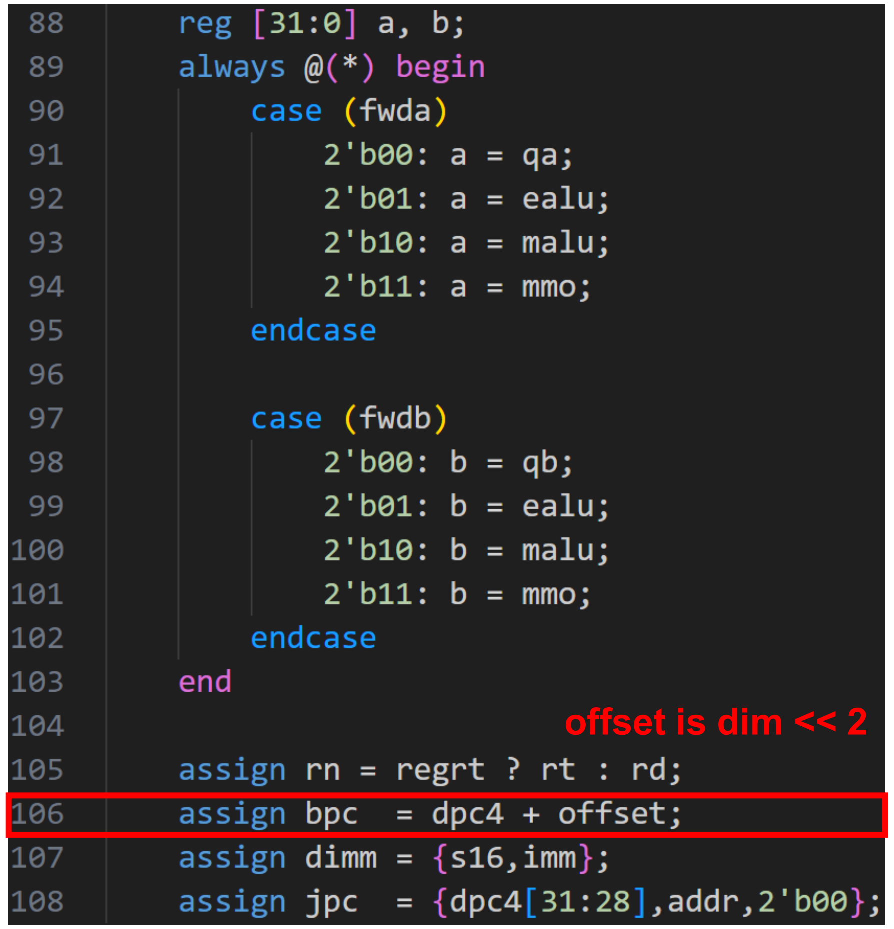
```
    reg [31:0] a, b;
    always @(*) begin
        case (fwda)
            2'b00: a = qa;
            2'b01: a = ealu;
            2'b10: a = malu;
            2'b11: a = mmo;
        endcase

        case (fwdb)
            2'b00: b = qb;
            2'b01: b = ealu;
            2'b10: b = malu;
            2'b11: b = mmo;
        endcase
    end
    assign rn = regrt ? rt : rd;
    assign bpc  = dpc4 + offset;
    assign dimm = {s16,imm};
    assign jpc  = {dpc4[31:28],addr,2'b00};
```

regrt is set for imm instructions

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipeexe.v
- pipeexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

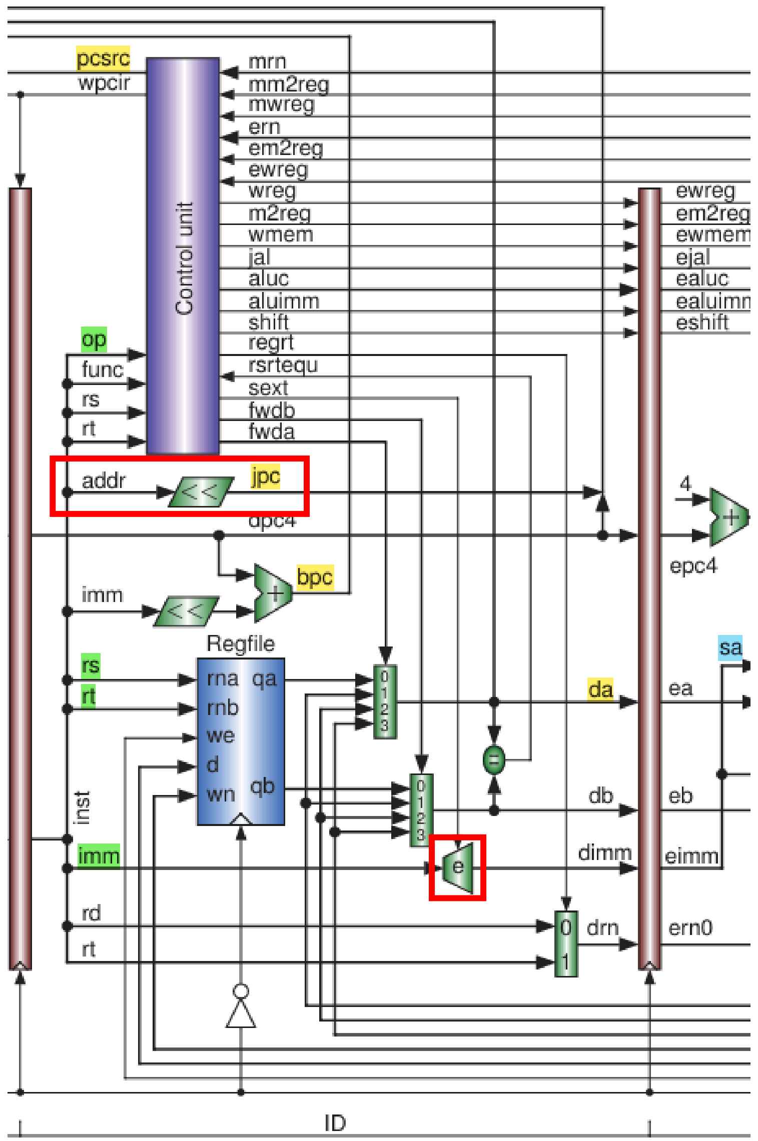


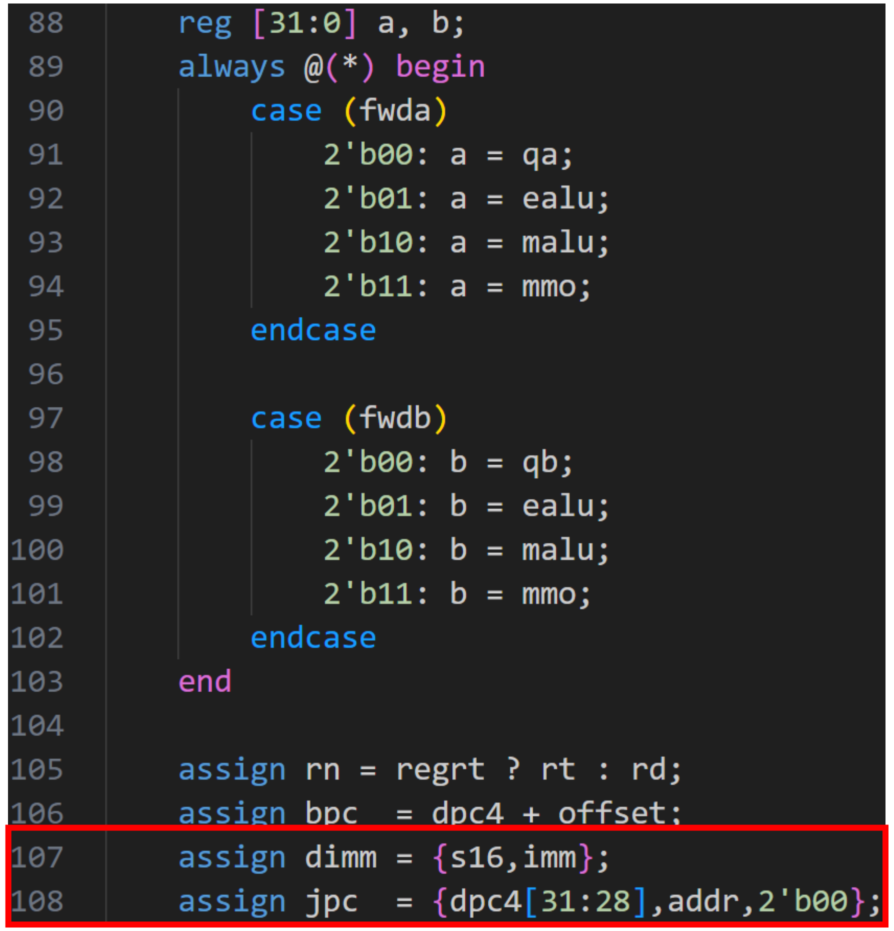


```
    reg [31:0] a, b;
    always @(*) begin
        case (fwda)
            2'b00: a = qa;
            2'b01: a = ealu;
            2'b10: a = malu;
            2'b11: a = mmo;
        endcase

        case (fwdb)
            2'b00: b = qb;
            2'b01: b = ealu;
            2'b10: b = malu;
            2'b11: b = mmo;
        endcase
    end
                                offset is dim << 2
    assign rn = regrt ? rt : rd;
    assign bpc  = dpc4 + offset;
    assign dimm = {s16,imm};
    assign jpc  = {dpc4[31:28],addr,2'b00};
```

Verilog HDL Implementation

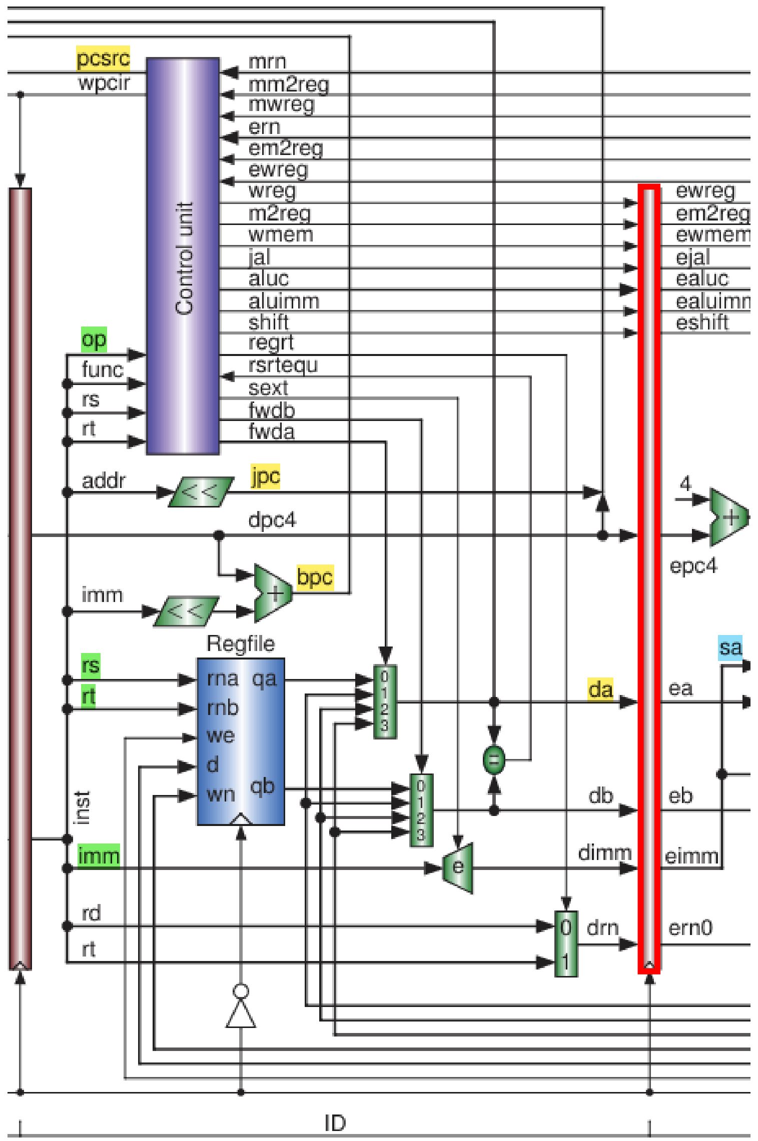
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

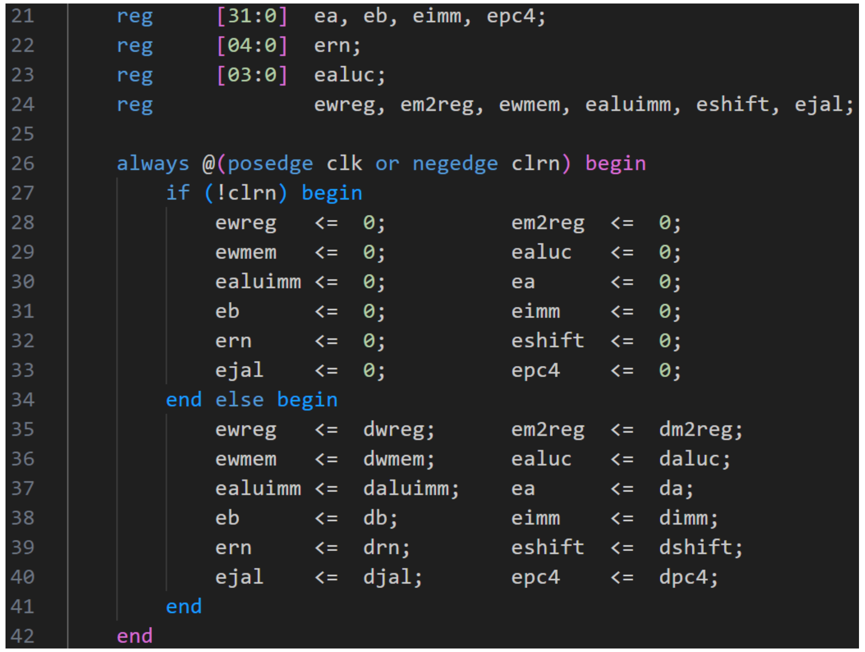



```
    reg [31:0] a, b;
    always @(*) begin
        case (fwda)
            2'b00: a = qa;
            2'b01: a = ealu;
            2'b10: a = malu;
            2'b11: a = mmo;
        endcase

        case (fwdb)
            2'b00: b = qb;
            2'b01: b = ealu;
            2'b10: b = malu;
            2'b11: b = mmo;
        endcase
    end

    assign rn = regrt ? rt : rd;
    assign bpc  = dpc4 + offset;
    assign dimm = {s16,imm};
    assign jpc  = {dpc4[31:28],addr,2'b00};
```

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

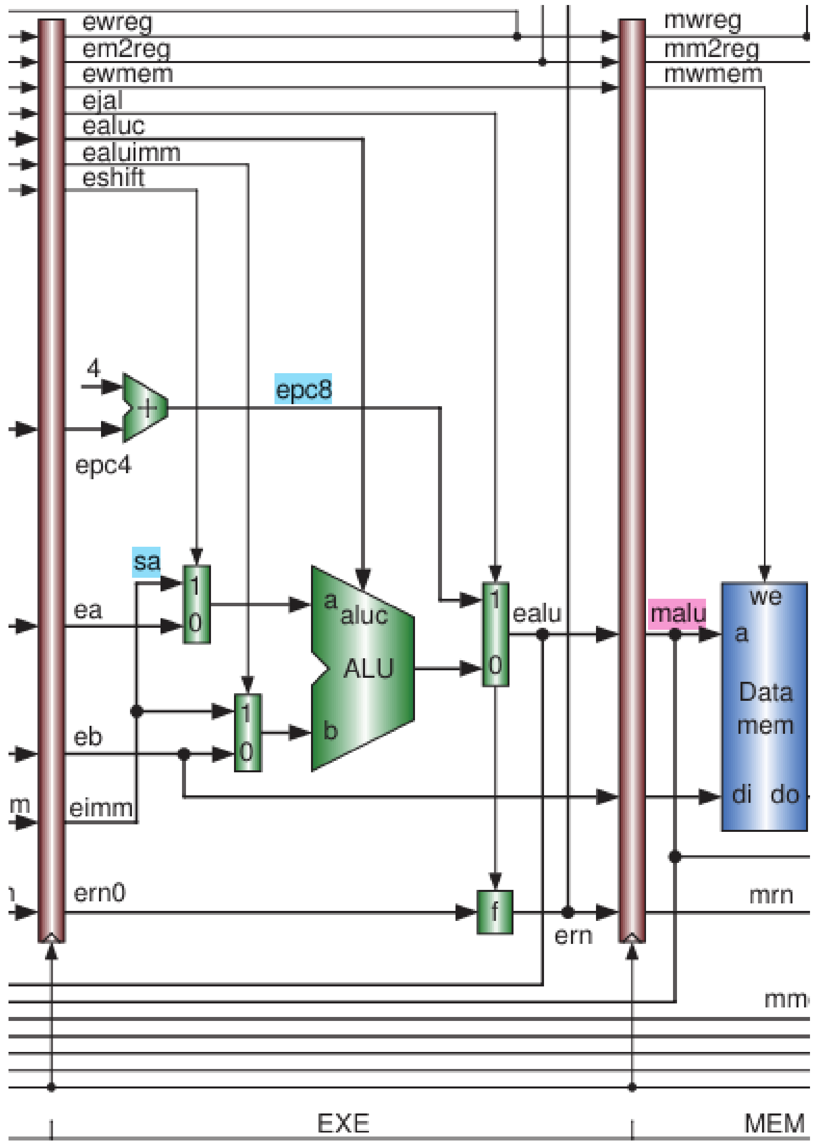



```
    reg     [31:0]  ea, eb, eimm, epc4;
    reg     [04:0]  ern;
    reg     [03:0]  ealuc;
    reg             ewreg, em2reg, ewmem, ealuimm, eshift, ejal;

    always @(posedge clk or negedge clrn) begin
        if (!clrn) begin
            ewreg   <= 0;           em2reg  <= 0;
            ewmem   <= 0;           ealuc   <= 0;
            ealuimm <= 0;           ea      <= 0;
            eb      <= 0;           eimm    <= 0;
            ern     <= 0;           eshift  <= 0;
            ejal    <= 0;           epc4    <= 0;
        end else begin
            ewreg   <= dwreg;       em2reg  <= dm2reg;
            ewmem   <= dwmem;       ealuc   <= daluc;
            ealuimm <= daluimm;     ea      <= da;
            eb      <= db;          eimm    <= dimm;
            ern     <= drn;         eshift  <= dshift;
            ejal    <= djal;        epc4    <= dpc4;
        end
    end
```

pipeline register for ID/EXE stage

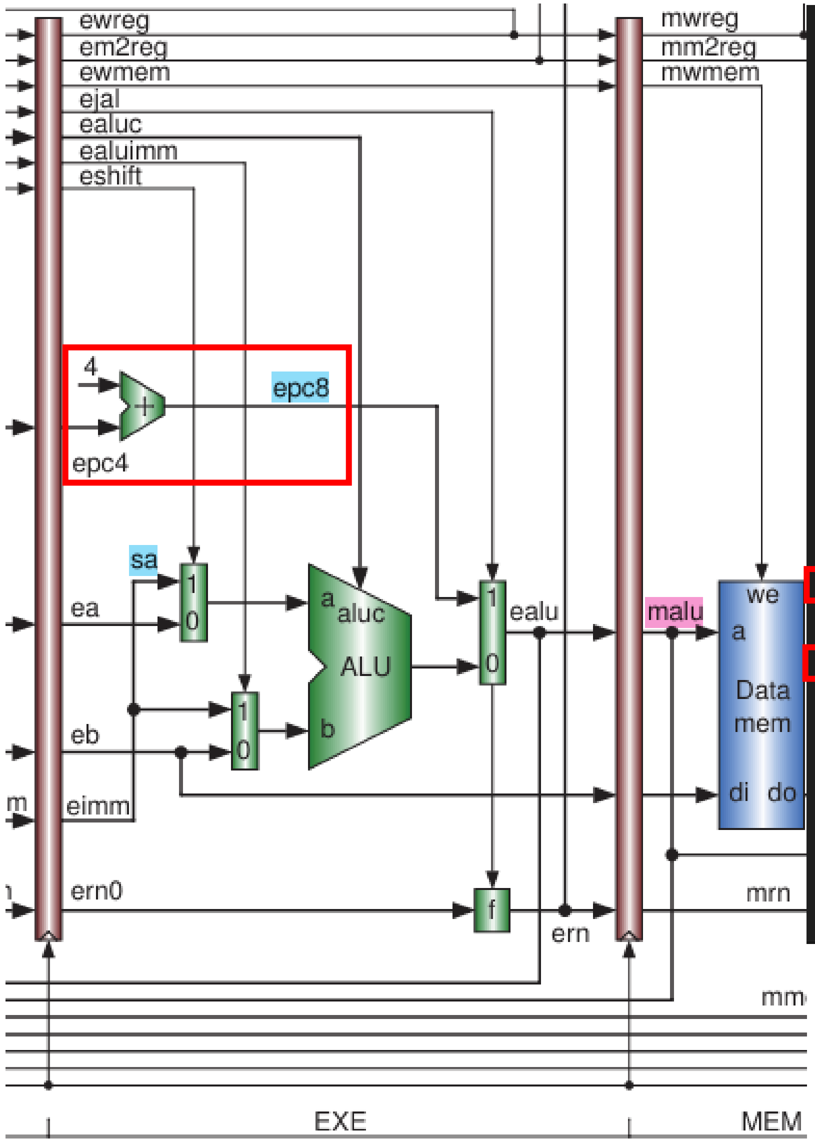
Verilog HDL Implementation

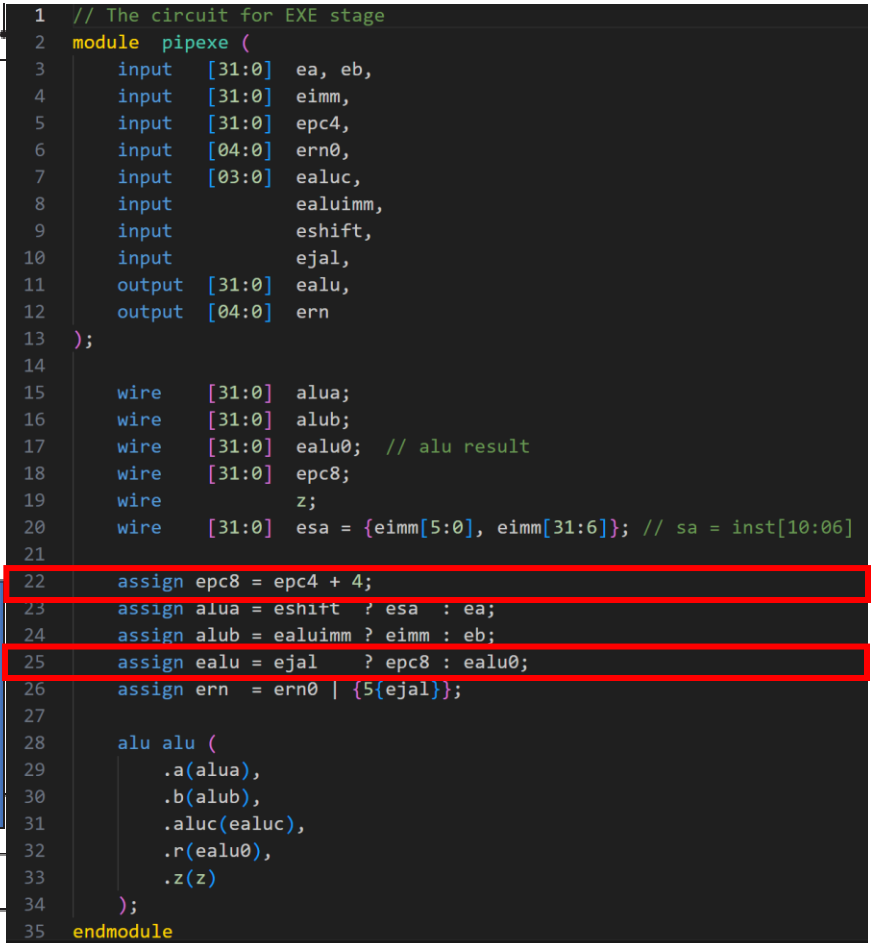
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



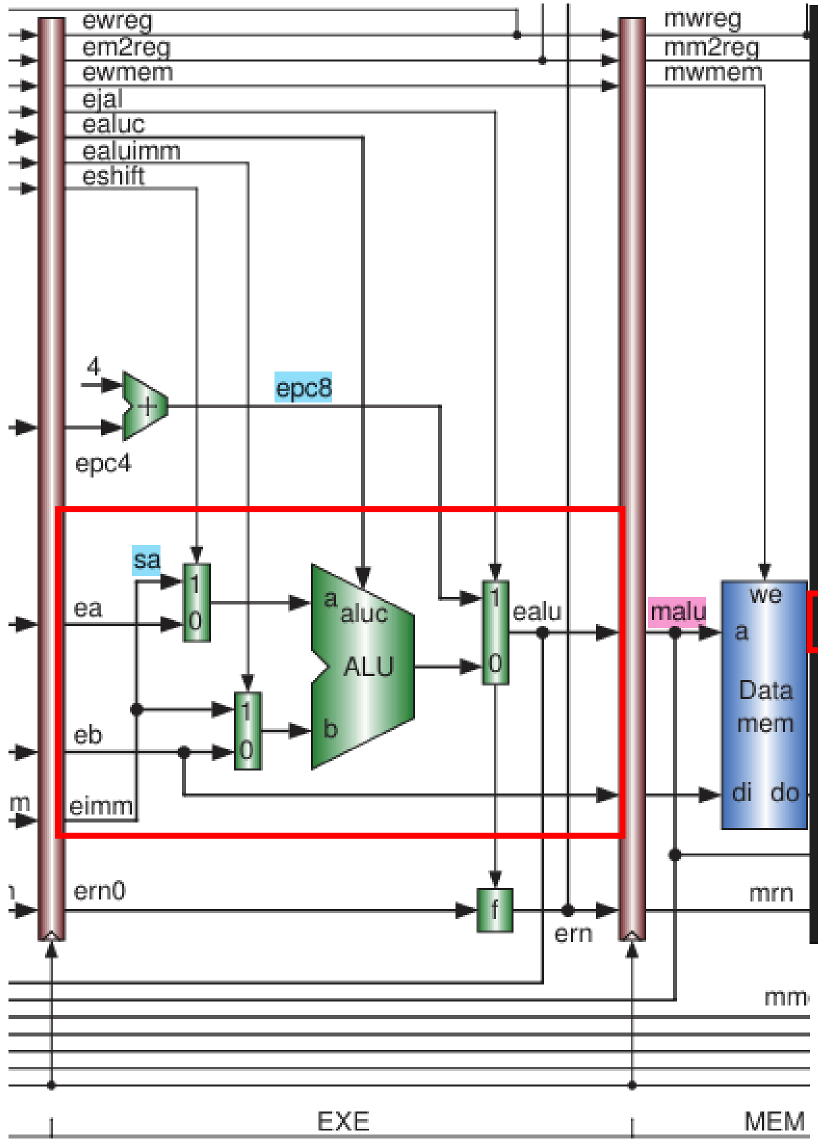


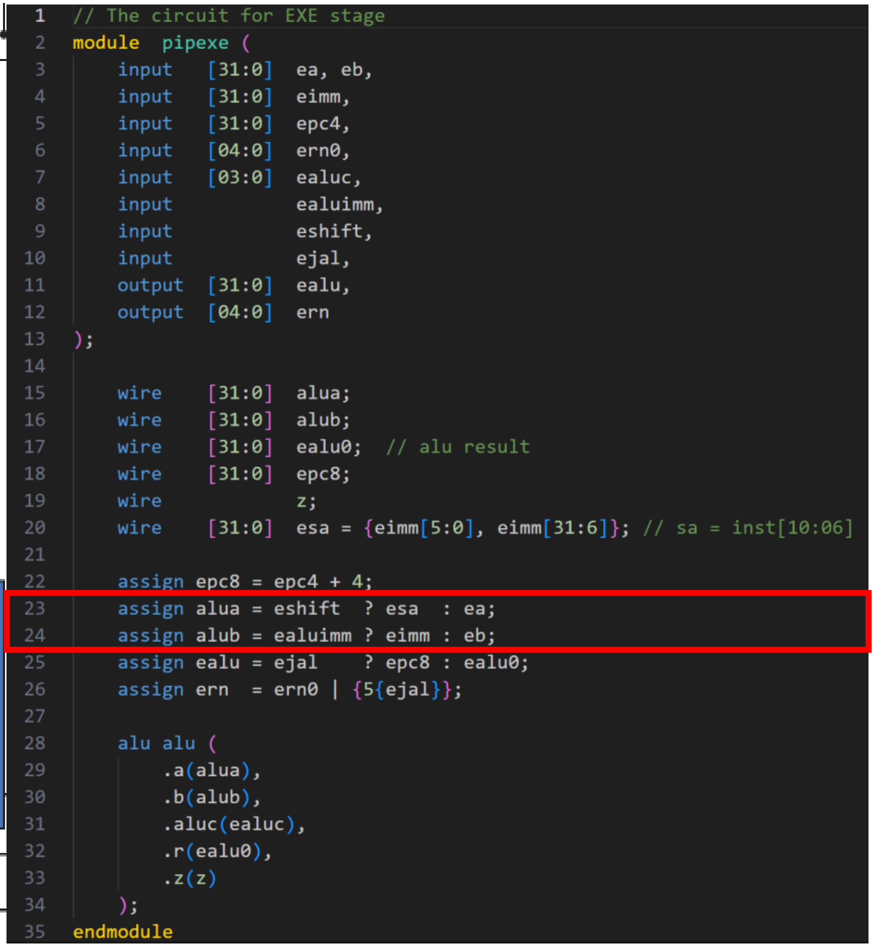
```verilog
// The circuit for EXE stage
module  pipexe (
    input   [31:0]  ea, eb,
    input   [31:0]  eimm,
    input   [31:0]  epc4,
    input   [04:0]  ern0,
    input   [03:0]  ealuc,
    input           ealuimm,
    input           eshift,
    input           ejal,
    output  [31:0]  ealu,
    output  [04:0]  ern
);

    wire    [31:0]  alua;
    wire    [31:0]  alub;
    wire    [31:0]  ealu0;  // alu result
    wire    [31:0]  epc8;
    wire            z;
    wire    [31:0]  esa = {eimm[5:0], eimm[31:6]}; // sa = inst[10:06]

    assign epc8 = epc4 + 4;
    assign alua = eshift  ? esa  : ea;
    assign alub = ealuimm ? eimm : eb;
    assign ealu = ejal    ? epc8 : ealu0;
    assign ern  = ern0 | {5{ejal}};

    alu alu (
        .a(alua),
        .b(alub),
        .aluc(ealuc),
        .r(ealu0),
        .z(z)
    );
endmodule
```

pc+8 for return address of jal

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

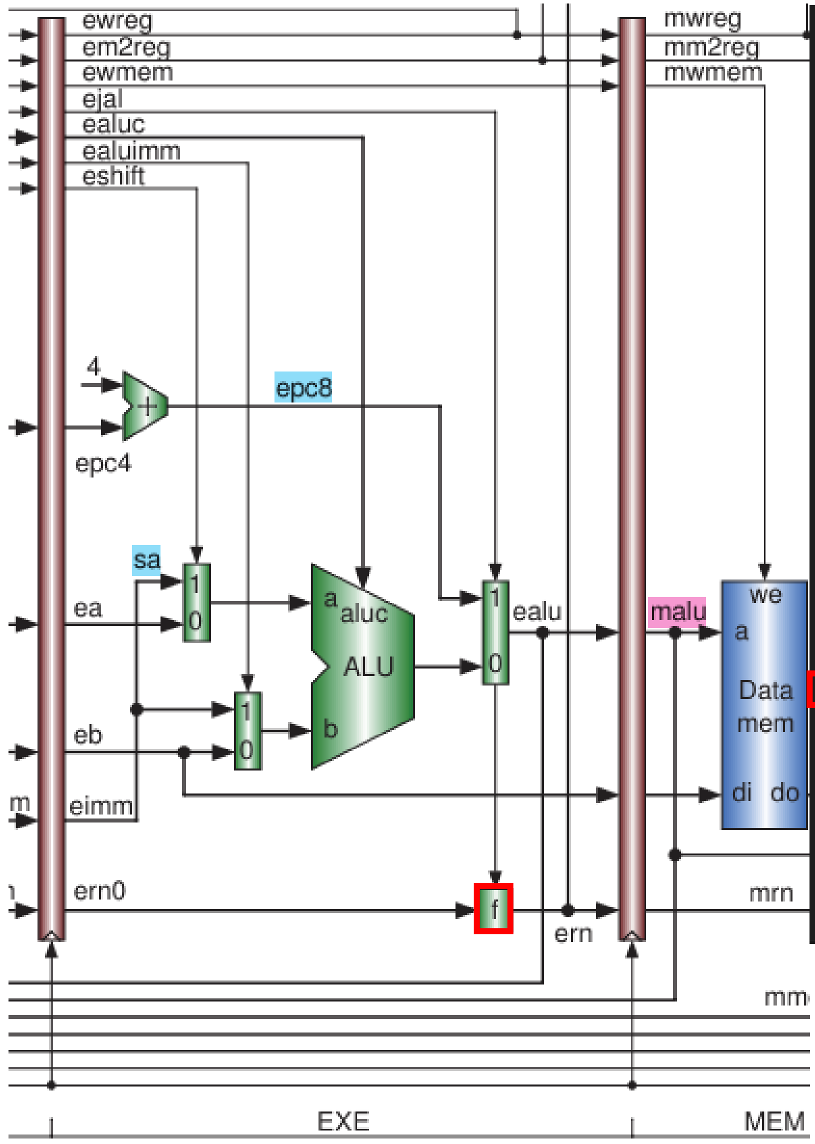


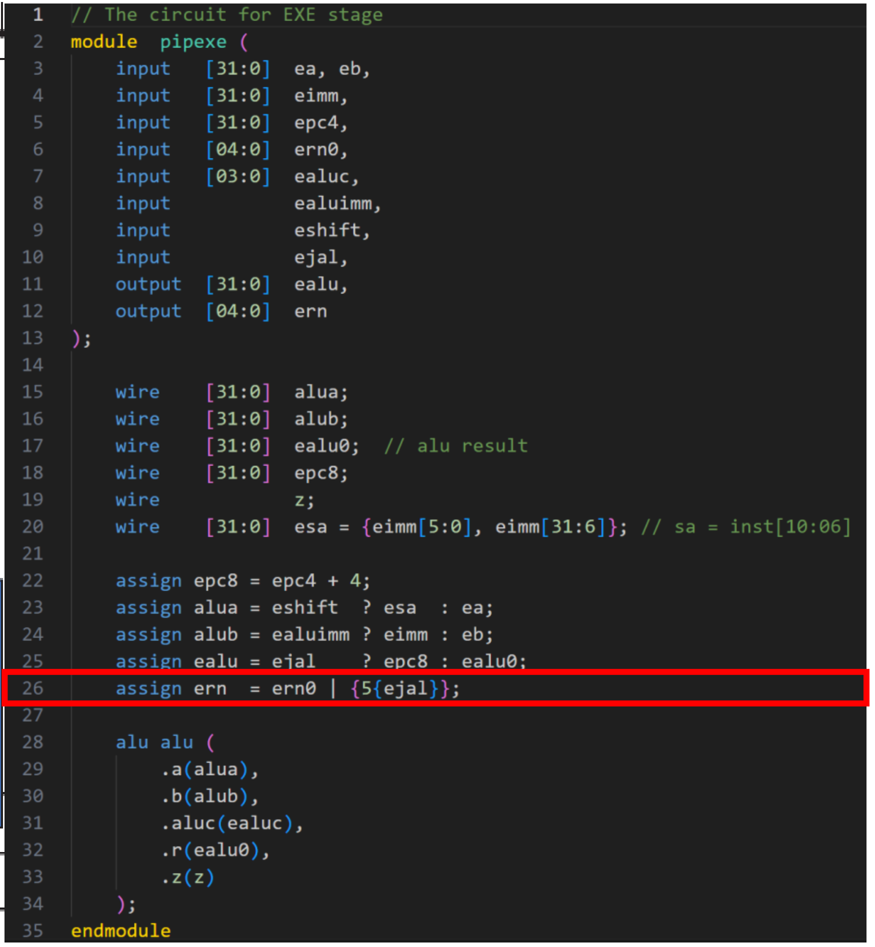

```verilog
// The circuit for EXE stage
module  pipexe (
    input   [31:0]  ea, eb,
    input   [31:0]  eimm,
    input   [31:0]  epc4,
    input   [04:0]  ern0,
    input   [03:0]  ealuc,
    input           ealuimm,
    input           eshift,
    input           ejal,
    output  [31:0]  ealu,
    output  [04:0]  ern
);

    wire    [31:0]  alua;
    wire    [31:0]  alub;
    wire    [31:0]  ealu0;  // alu result
    wire    [31:0]  epc8;
    wire            z;
    wire    [31:0]  esa = {eimm[5:0], eimm[31:6]}; // sa = inst[10:06]

    assign epc8 = epc4 + 4;
    assign alua = eshift   ? esa  : ea;
    assign alub = ealuimm ? eimm : eb;
    assign ealu = ejal     ? epc8 : ealu0;
    assign ern  = ern0 | {5{ejal}};

    alu alu (
        .a(alua),
        .b(alub),
        .aluc(ealuc),
        .r(ealu0),
        .z(z)
    );
endmodule
```

ALU input a is a shift amount if shift instruction is fetched.
ALU input b is a immediate value if imm instruction is fetched.

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

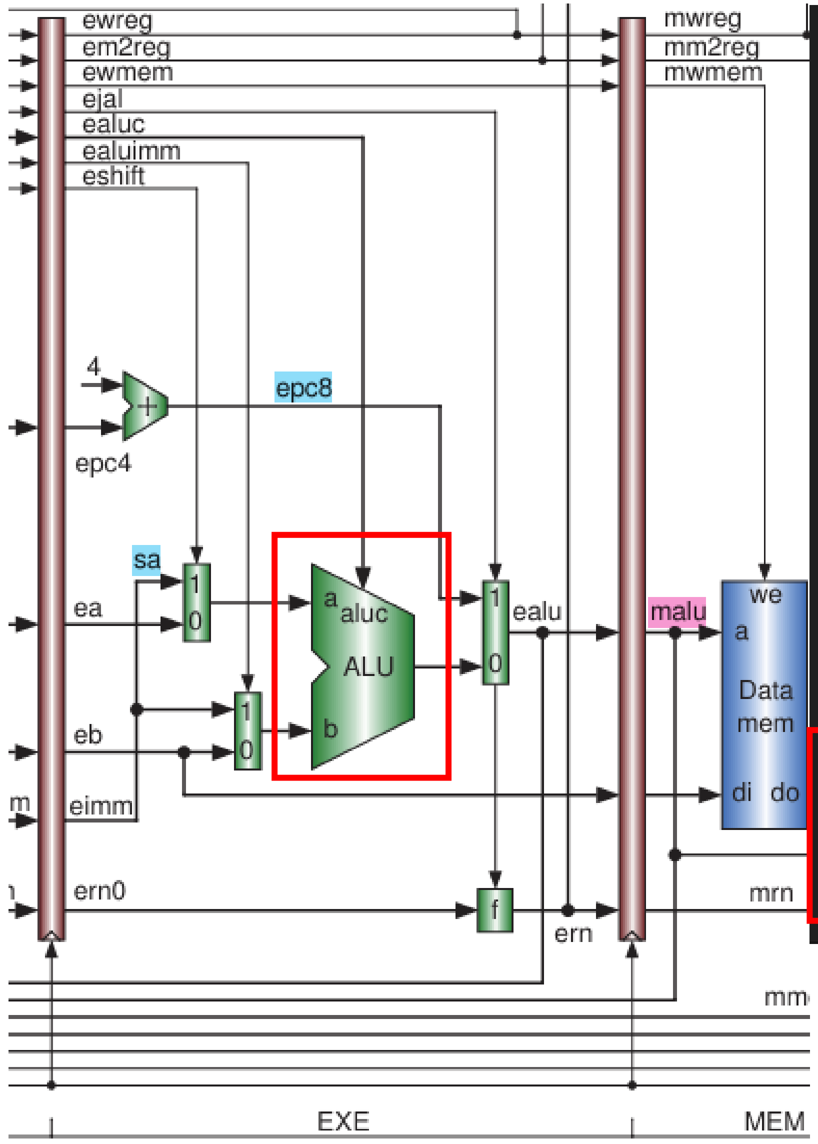


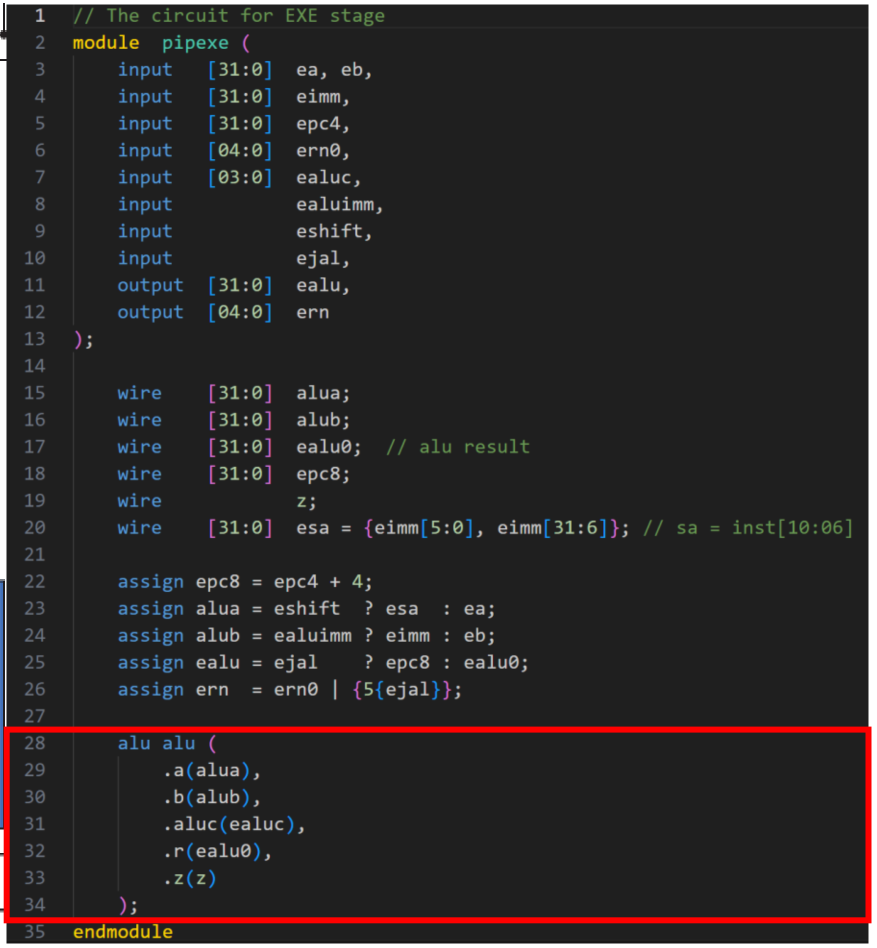

```verilog
// The circuit for EXE stage
module  pipexe (
    input   [31:0]  ea, eb,
    input   [31:0]  eimm,
    input   [31:0]  epc4,
    input   [04:0]  ern0,
    input   [03:0]  ealuc,
    input           ealuimm,
    input           eshift,
    input           ejal,
    output  [31:0]  ealu,
    output  [04:0]  ern
);

    wire    [31:0]  alua;
    wire    [31:0]  alub;
    wire    [31:0]  ealu0;  // alu result
    wire    [31:0]  epc8;
    wire            z;
    wire    [31:0]  esa = {eimm[5:0], eimm[31:6]}; // sa = inst[10:06]

    assign epc8 = epc4 + 4;
    assign alua = eshift  ? esa  : ea;
    assign alub = ealuimm ? eimm : eb;
    assign ealu = ejal    ? epc8 : ealu0;
    assign ern  = ern0 | {5{ejal}};

    alu alu (
        .a(alua),
        .b(alub),
        .aluc(ealuc),
        .r(ealu0),
        .z(z)
    );
endmodule
```

$31 register is selected for return address of jal instruction. (5’b11111 = 32)

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
  - ✓alu.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

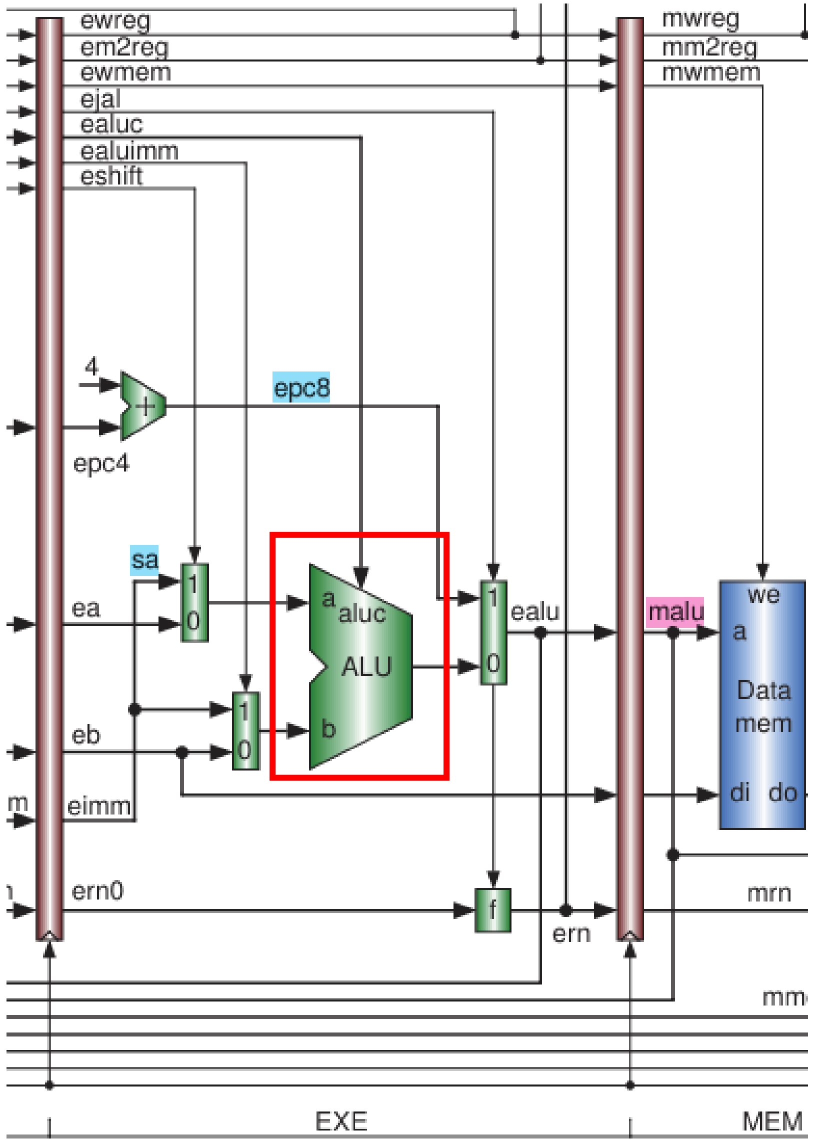


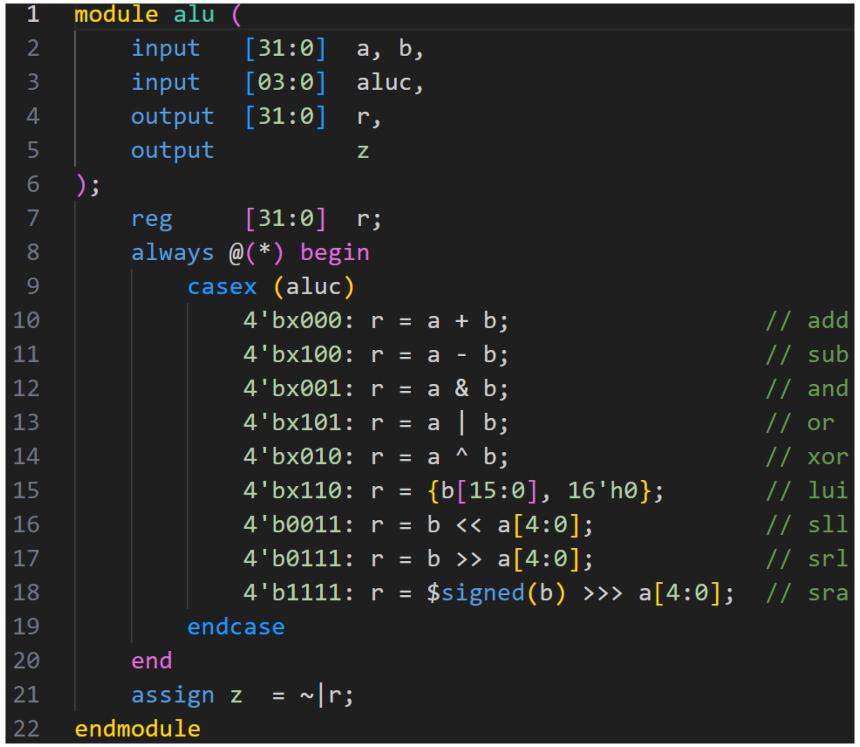


```
// The circuit for EXE stage
module  pipexe (
    input   [31:0]  ea, eb,
    input   [31:0]  eimm,
    input   [31:0]  epc4,
    input   [04:0]  ern0,
    input   [03:0]  ealuc,
    input           ealuimm,
    input           eshift,
    input           ejal,
    output  [31:0]  ealu,
    output  [04:0]  ern
);

    wire    [31:0]  alua;
    wire    [31:0]  alub;
    wire    [31:0]  ealu0;  // alu result
    wire    [31:0]  epc8;
    wire            z;
    wire    [31:0]  esa = {eimm[5:0], eimm[31:6]}; // sa = inst[10:06]

    assign epc8 = epc4 + 4;
    assign alua = eshift  ? esa  : ea;
    assign alub = ealuimm ? eimm : eb;
    assign ealu = ejal    ? epc8 : ealu0;
    assign ern  = ern0 | {5{ejal}};

    alu alu (
        .a(alua),
        .b(alub),
        .aluc(ealuc),
        .r(ealu0),
        .z(z)
    );
endmodule
```

Verilog HDL Implementation

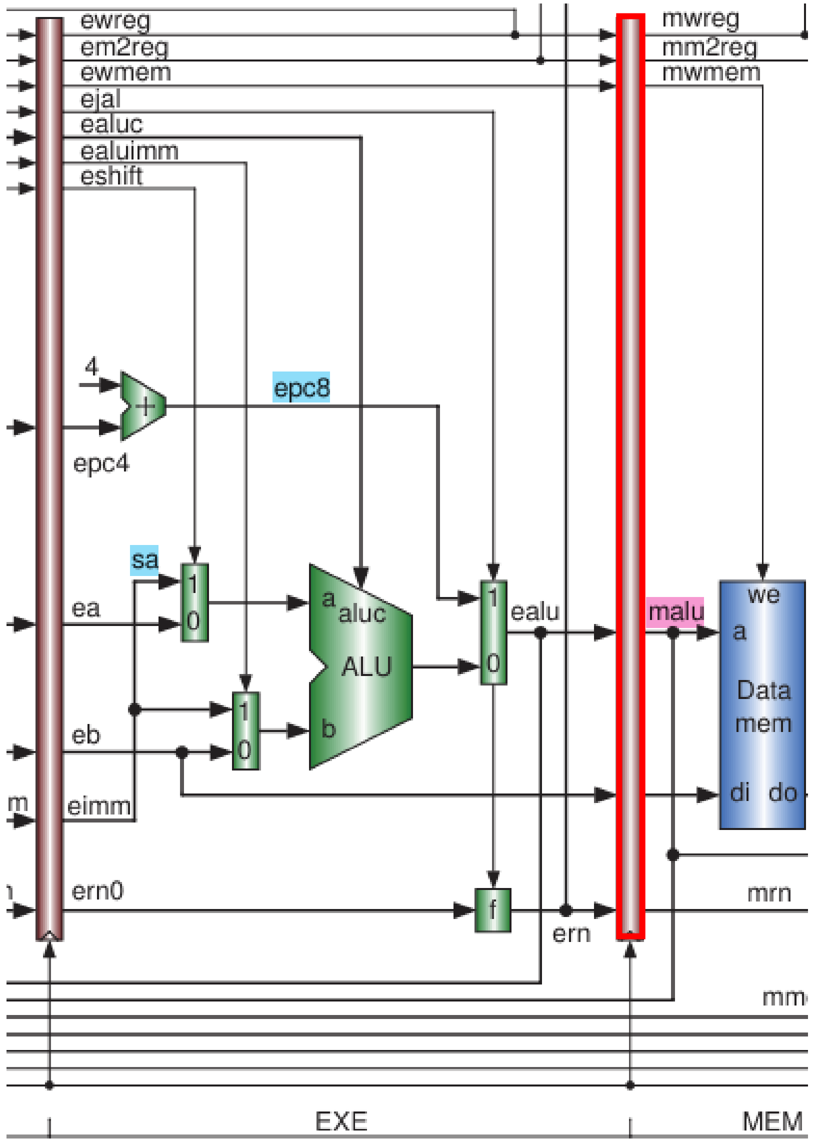
- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
  - ✓alu.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

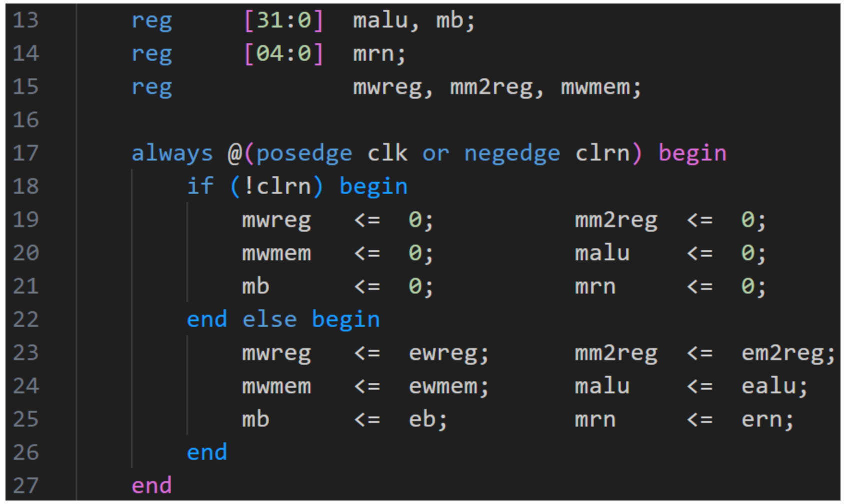



```
module alu (
    input   [31:0]  a, b,
    input   [03:0]  aluc,
    output  [31:0]  r,
    output          z
);
    reg     [31:0]  r;
    always @(*) begin
        casex (aluc)
            4'bx000: r = a + b;                      // add
            4'bx100: r = a - b;                      // sub
            4'bx001: r = a & b;                      // and
            4'bx101: r = a | b;                      // or
            4'bx010: r = a ^ b;                      // xor
            4'bx110: r = {b[15:0], 16'h0};           // lui
            4'b0011: r = b << a[4:0];                // sll
            4'b0111: r = b >> a[4:0];                // srl
            4'b1111: r = $signed(b) >>> a[4:0];      // sra
        endcase
    end
    assign z  = ~|r;
endmodule
```

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

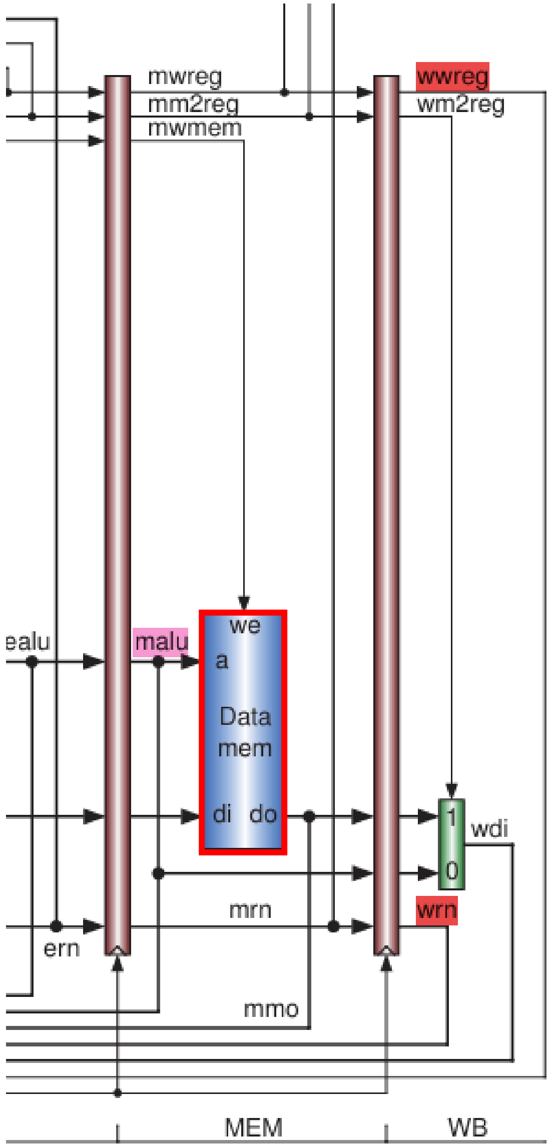


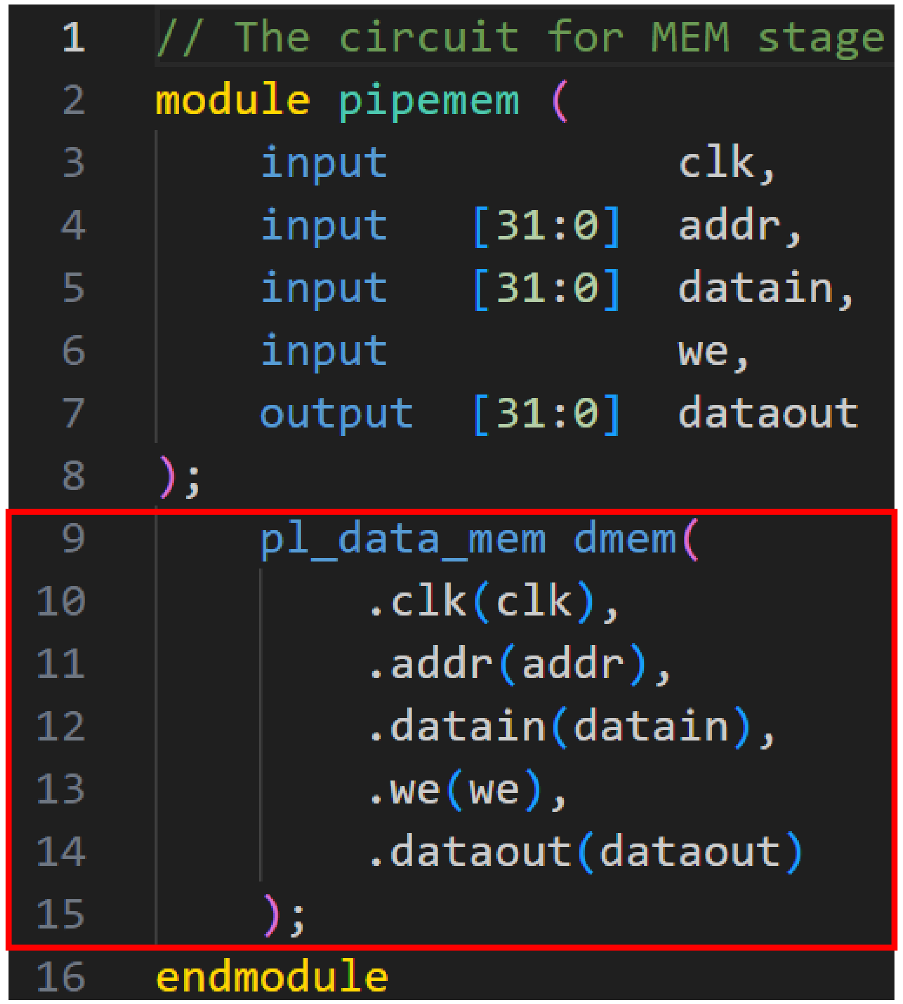

```
    reg     [31:0]  malu, mb;
    reg     [04:0]  mrn;
    reg             mwreg, mm2reg, mwmem;

    always @(posedge clk or negedge clrn) begin
        if (!clrn) begin
            mwreg   <=  0;          mm2reg  <=  0;
            mwmem   <=  0;          malu    <=  0;
            mb      <=  0;          mrn     <=  0;
        end else begin
            mwreg   <=  ewreg;      mm2reg  <=  em2reg;
            mwmem   <=  ewmem;      malu    <=  ealu;
            mb      <=  eb;         mrn     <=  ern;
        end
    end
```

pipeline register for EXE/MEM stage.

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v

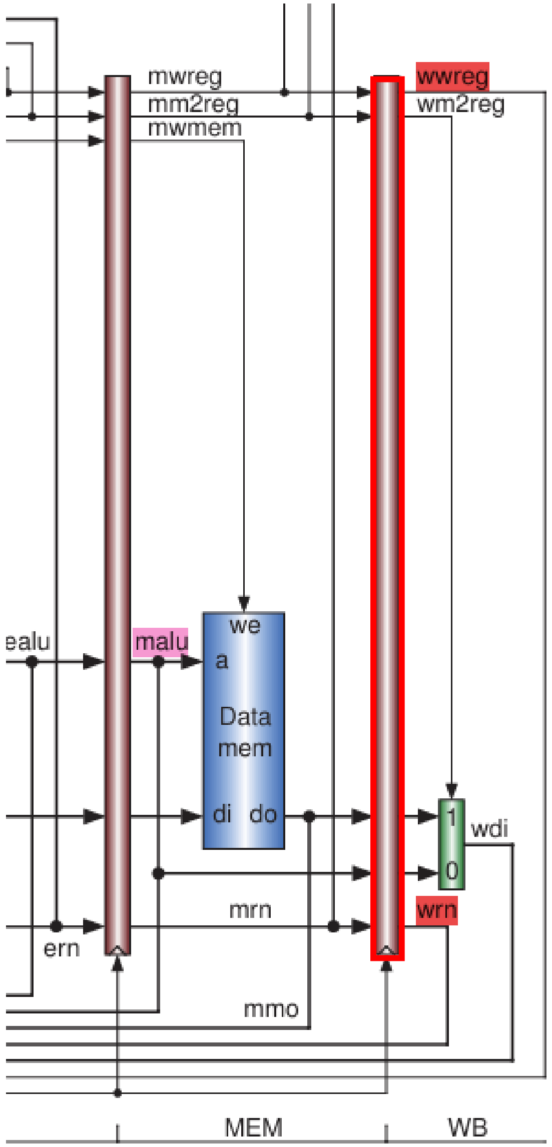


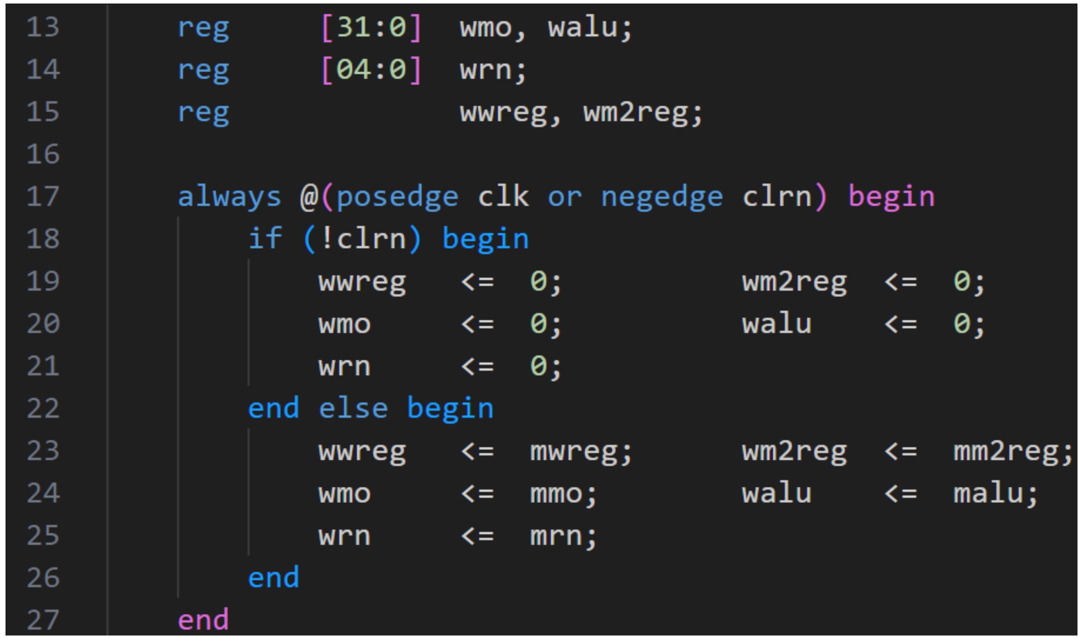


```
// The circuit for MEM stage
module pipemem (
    input           clk,
    input   [31:0]  addr,
    input   [31:0]  datain,
    input           we,
    output  [31:0]  dataout
);
    pl_data_mem dmem(
        .clk(clk),
        .addr(addr),
        .datain(datain),
        .we(we),
        .dataout(dataout)
    );
endmodule
```

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipeexe.v
- pipeexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



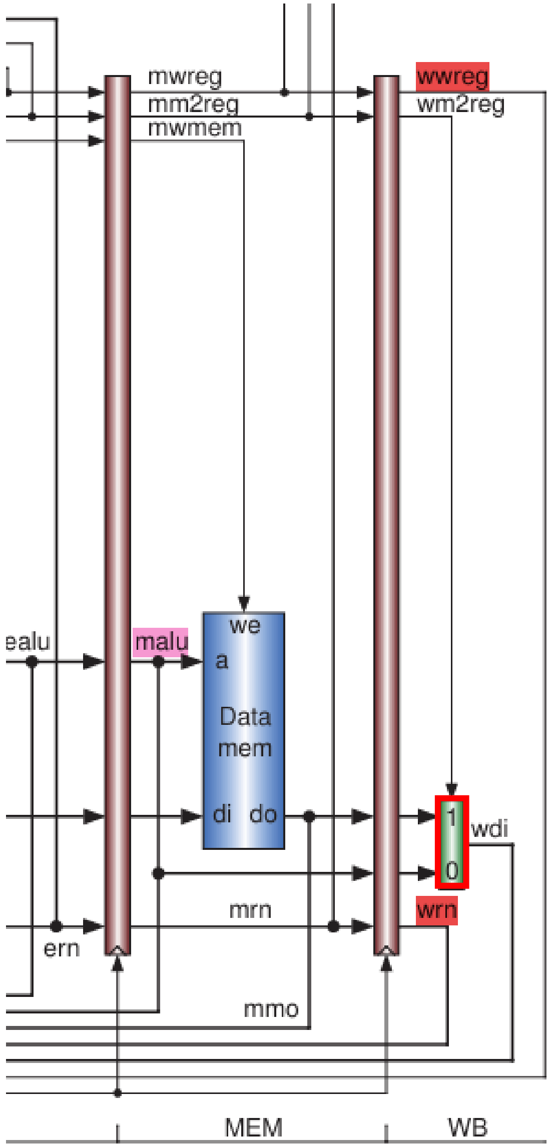


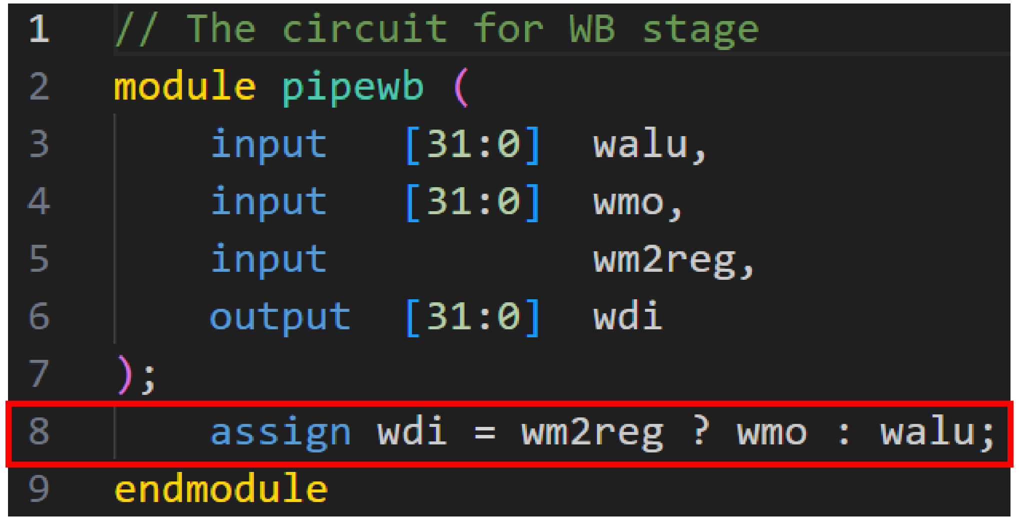
```
    reg     [31:0]  wmo, walu;
    reg     [04:0]  wrn;
    reg             wwreg, wm2reg;

    always @(posedge clk or negedge clrn) begin
        if (!clrn) begin
            wwreg   <=  0;          wm2reg  <=  0;
            wmo     <=  0;          walu    <=  0;
            wrn     <=  0;
        end else begin
            wwreg   <=  mwreg;      wm2reg  <=  mm2reg;
            wmo     <=  mmo;        walu    <=  malu;
            wrn     <=  mrn;
        end
    end
```

pipeline register for MEM/WB stage.

Verilog HDL Implementation

- pipecpu.v
- pipepc.v
- pipeif.v
- pipeifid.v
- pipeid.v
- pipeidexe.v
- pipexe.v
- pipexemem.v
- pipemem.v
- pipememwb.v
- pipewb.v



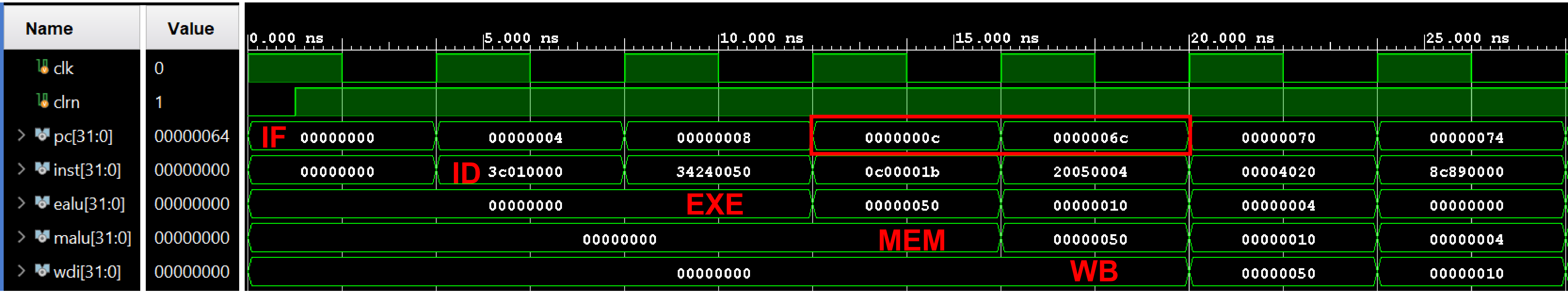


```
// The circuit for WB stage
module pipewb (
    input    [31:0]  walu,
    input    [31:0]  wmo,
    input            wm2reg,
    output   [31:0]  wdi
);
    assign wdi = wm2reg ? wmo : walu;
endmodule
```

MUX for wdi (Data memory read or ALU result).

Simulation

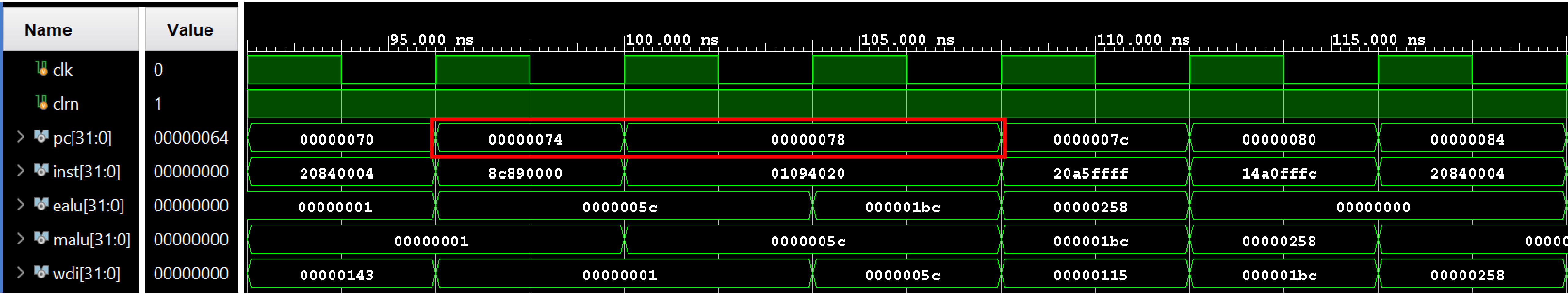
- Call subroutine



Name	Value							
clk	0							
clrn	1							
pc[31:0]	00000064	00000000	00000004	00000008	0000000c	0000006c	00000070	00000074
inst[31:0]	00000000	00000000	3c010000	34240050	0c00001b	20050004	00004020	8c890000
ealu[31:0]	00000000	00000000			00000050	00000010	00000004	00000000
malu[31:0]	00000000	00000000				00000050	00000010	00000004
wdi[31:0]	00000000	00000000					00000050	00000010

Simulation

- Pipeline stall

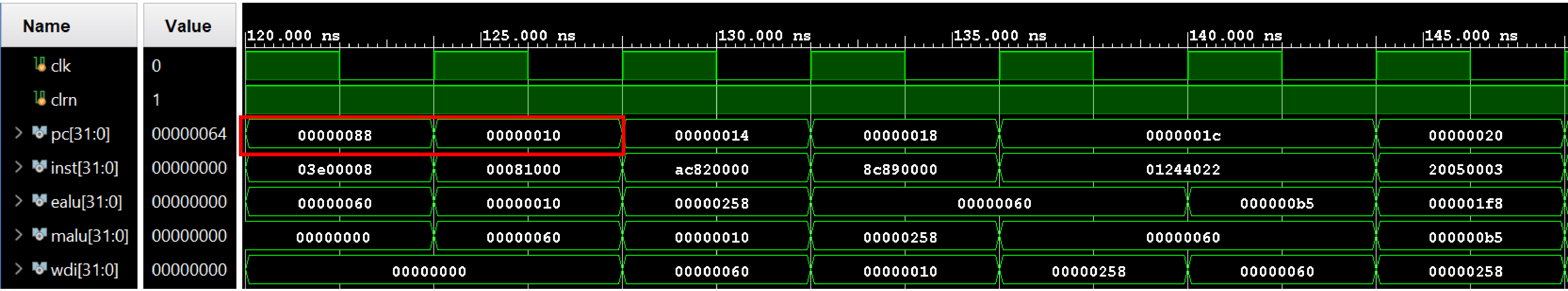


Name	Value
clk	0
clrn	1
pc[31:0]	00000064
inst[31:0]	00000000
ealu[31:0]	00000000
malu[31:0]	00000000
wdi[31:0]	00000000

95.000 ns
100.000 ns
105.000 ns
110.000 ns
115.000 ns
00000070 00000074 00000078 0000007c 00000080 00000084
20840004 8c890000 01094020 20a5ffff 14a0fffc 20840004
00000001 0000005c 000001bc 00000258 00000000
00000001 0000005c 000001bc 00000258 00000
00000143 00000001 0000005c 00000115 000001bc 00000258

Simulation

- Return from subroutine



```
Name        Value     |120.000 ns            |125.000 ns            |130.000 ns            |135.000 ns            |140.000 ns            |145.000 ns
clk         0
clrn        1
pc[31:0]    00000064   00000088   00000010   00000014   00000018        0000001c             00000020
inst[31:0]  00000000   03e00008   00081000   ac820000   8c890000        01244022             20050003
ealu[31:0]  00000000   00000060   00000010   00000258        00000060         000000b5       000001f8
malu[31:0]  00000000   00000000   00000060   00000010   00000258        00000060             000000b5
wdi[31:0]   00000000        00000000         00000060   00000010   00000258   00000060       00000258
```

References

- [1] Yamin Li, *Computer Principles and Design in Verilog HDL*, Wiley, 2016.