付録B

CPU実装の Verilog ファイル

computer.v

```
module computer(clk, rstd);
  input clk, rstd;

  wire [31:0] pc, ins, reg1, reg2, result, nextpc;
  wire [4:0] wra;
  wire [3:0] wren;

  fetch fetch_body(pc[7:0], ins);
  execute execute_body(clk, ins, pc, reg1, reg2, wra, result, nextpc);
  writeback writeback_body(clk, rstd, nextpc, pc);
  reg_file rf_body(clk, rstd, result, ins[25:21], ins[20:16], wra,
(~|wra), reg1, reg2);

  initial
    $monitor($time, "rstd=%d, clk=%d, pc=%d, nextpc=%d, ins=%h, wra=%h,
reg1=%h, reg2=%h", rstd, clk, pc, nextpc, ins, wra, reg1, reg2);
endmodule

module test_computer;
  reg clk, rstd;

  initial
    begin
      rstd = 1;
      #10 rstd = 0;
      #10 rstd = 1;
    end

  initial
    begin
      clk = 0;
      forever #50 clk = !clk;
    end

  computer computer_body(clk, rstd);
endmodule // test_computer
```

data_mem.v

```
module data_mem(address, clk, write_data, wren, read_data);
  input [7:0] address;
```

```
  input clk, wren;
  input [7:0] write_data;
  output [7:0] read_data;

  reg [7:0] d_mem [0:255];

  always @ (posedge clk)
    if (wren == 0) d_mem[address] <= write_data;
  assign read_data = d_mem[address];
endmodule

module test_mem;
  reg [7:0] address;
  reg clk, wren;
  reg [31:0] ra, wa, write_data;
  wire [31:0] read_data;

  initial
    begin
      clk = 0;
      forever #50 clk = !clk;
    end

  initial
    begin
      #40 address = 0; write_data = 8'h21; wren = 0;
      #100 address = 1; write_data = 8'h43; wren = 0;
      #100 address = 2; write_data = 8'h65; wren = 1;
      #100 address = 2; write_data = 8'h87; wren = 0;
      #100 address = 3; write_data = 8'ha9; wren = 0;
      #100 address = 0; wren = 1;
      #100 address = 1; wren = 1;
      #100 address = 2; wren = 1;
      #100 address = 3; wren = 1;
    end

  initial
    $monitor($stime, " address=%d, clk=%d, write_data=%h, wren=%d,
read_data=%h", address, clk, write_data, wren, read_data);

  data_mem data_mem_body(address, clk, write_data, wren, read_data);
endmodule
```

execute.v

```
module execute(clk, ins, pc, reg1, reg2, wra, result, nextpc);
  input clk;
  input [31:0] ins, pc, reg1, reg2;
  output [4:0] wra;
  output[31:0] result, nextpc;
```

```
  wire [5:0] op;
  wire [4:0] shift, operation;
  wire [25:0] addr;
  wire signed [31:0] dpl_imm, operand2, alu_result, nonbranch, branch,
mem_address, dm_r_data;
  wire [3:0] wren;

  function [4:0] opr_gen;
    input [5:0] op;
    input [4:0] operation;

    case (op)
      6'd0: opr_gen = operation;
      6'd1: opr_gen = 5'd0;
      6'd4: opr_gen = 5'd8;
      6'd5: opr_gen = 5'd9;
      6'd6: opr_gen = 5'd10;
      6'd16, 6'd18, 6'd20, 6'd24, 6'd26, 6'd28: opr_gen = 5'd0;
      default: opr_gen = 5'h1f;
    endcase
  endfunction

  function [31:0] alu;
    input [4:0] opr, shift;
    input signed [31:0] operand1, operand2;

    case (opr)
      5'd0: alu = operand1 + operand2;
      5'd2: alu = operand1 - operand2;
      5'd8: alu = operand1 & operand2;
      5'd9: alu = operand1 | operand2;
      5'd10: alu = operand1 ^ operand2;
      5'd11: alu = ~(operand1 | operand2);
      5'd16: alu = operand1 << shift;
      5'd17: alu = operand1 >> shift;
      5'd18: alu = operand1 >>> shift;
      default: alu = 32'hffffffff;
    endcase
  endfunction

  function [31:0] calc;
    input [5:0] op;
    input [31:0] alu_result, dpl_imm, dm_r_data, pc;

    case (op)
      6'd0, 6'd1, 6'd4, 6'd5, 6'd6: calc = alu_result;
      6'd3: calc = dpl_imm << 16;
      6'd16: calc = dm_r_data;
      6'd18: calc = {{16{dm_r_data[15]}}, dm_r_data[15:0]};
      6'd20: calc = {{24{dm_r_data[7]}}, dm_r_data[7:0]};
      6'd41: calc = pc + 32'd1;
      default: calc = 32'hffffffff;
    endcase
```

```
  endfunction

  function [31:0] npc;
    input [5:0] op;
    input signed [31:0] reg1, reg2;
    input [31:0] branch, nonbranch, addr;

    case (op)
      6'd32: npc = (reg1 == reg2) ? branch : nonbranch;
      6'd33: npc = (reg1 != reg2) ? branch : nonbranch;
      6'd34: npc = (reg1 < reg2) ? branch : nonbranch;
      6'd35: npc = (reg1 <= reg2) ? branch : nonbranch;
      6'd40, 6'd41: npc = addr;
      6'd42: npc = reg1;
      default: npc = nonbranch;
    endcase
  endfunction

  function [4:0] wreg;
    input [5:0] op;
    input [4:0] rt, rd;

    case (op)
      6'd0: wreg = rd;
      6'd1, 6'd3, 6'd4, 6'd5, 6'd6, 6'd16, 6'd18, 6'd20: wreg = rt;
      6'd41: wreg = 5'd31;
      default: wreg = 5'd0;
    endcase
  endfunction

  function [3:0] wrengen;
    input [5:0] op;

    case (op)
      6'd24: wrengen = 4'b0000;
      6'd26: wrengen = 4'b1100;
      6'd28: wrengen = 4'b1110;
      default: wrengen = 4'b1111;
    endcase
  endfunction

  assign op = ins[31:26];
  assign shift = ins[10:6];
  assign operation = ins[4:0];
  assign dpl_imm = {{16{ins[15]}}, ins[15:0]};
  assign operand2 = (op == 6'd0) ? reg2 : dpl_imm;
  assign alu_result = alu(opr_gen(op, operation), shift, reg1, operand2);

  assign mem_address = alu_result >>> 2;
  assign wren = wrengen(op);

  data_mem data_mem_body0(mem_address[7:0], clk, reg2[7:0], wren[0],
dm_r_data[7:0]);
  data_mem data_mem_body1(mem_address[7:0], clk, reg2[15:8], wren[1],
```

```
dm_r_data[15:8]);
  data_mem data_mem_body2(mem_address[7:0], clk, reg2[23:16], wren[2],
dm_r_data[23:16]);
  data_mem data_mem_body3(mem_address[7:0], clk, reg2[31:24], wren[3],
dm_r_data[31:24]);

  assign wra = wreg(op, ins[20:16], ins[15:11]);
  assign result = calc(op, alu_result, dpl_imm, dm_r_data, pc);

  assign addr = ins[25:0];
  assign nonbranch = pc + 32'd1;
  assign branch = nonbranch + dpl_imm;
  assign nextpc = npc(op, reg1, reg2, branch, nonbranch, addr);
endmodule

module test_opr_gen;
  reg [5:0] op;
  reg [4:0] operation;
  reg [4:0] opr;

  function [4:0] opr_gen;
    input [5:0] op;
    input [4:0] operation;

    case (op)
      6'd0: opr_gen = operation;
      6'd1: opr_gen = 5'd0;
      6'd4: opr_gen = 5'd8;
      6'd5: opr_gen = 5'd9;
      6'd6: opr_gen = 5'd10;
      default: opr_gen = 5'h1f;
    endcase
  endfunction

  initial
    begin
      op = 6'd0; operation = 5'd0; opr = opr_gen(op, operation);
      #100 op = 6'd0; operation = 5'd8; opr = opr_gen(op, operation);
      #100 op = 6'd0; operation = 5'd11; opr = opr_gen(op, operation);
      #100 op = 6'd1; operation = 5'd0; opr = opr_gen(op, operation);
      #100 op = 6'd4; operation = 5'd3; opr = opr_gen(op, operation);
      #100 op = 6'd5; operation = 5'd9; opr = opr_gen(op, operation);
      #100 op = 6'd6; operation = 5'd11; opr = opr_gen(op, operation);
      #100 op = 6'd2; operation = 5'd0; opr = opr_gen(op, operation);
      #100 op = 6'd10; operation = 5'd11; opr = opr_gen(op, operation);
    end

  initial
    $monitor($stime, " op=%d, operation=%d, opr=%d", op, operation, opr);
endmodule

module test_alu;
  reg [4:0] opr, shift;
  reg [31:0] operand1, operand2, result;
```

```
  function [31:0] alu;
    input [4:0] opr, shift;
    input signed [31:0] operand1, operand2;

    case (opr)
      5'd0: alu = operand1 + operand2;
      5'd2: alu = operand1 - operand2;
      5'd8: alu = operand1 & operand2;
      5'd9: alu = operand1 | operand2;
      5'd10: alu = operand1 ^ operand2;
      5'd11: alu = ~(operand1 | operand2);
      5'd16: alu = operand1 << shift;
      5'd17: alu = operand1 >> shift;
      5'd18: alu = operand1 >>> shift;
      default: alu = 32'hffffffff;
    endcase
  endfunction

  initial
    begin
      opr = 0;
      shift = 0;
      operand1 = 32'h00000000;
      operand2 = 32'h00000000;
      result = alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'h00000000; operand2 = 32'h00000001; result =
alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'h0fffffff; operand2 = 32'h00000001; result =
alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'hffffffff; operand2 = 32'hffffffff; result =
alu(opr, shift, operand1, operand2);
      #100 opr = 1; operand1 = 32'h00000000; operand2 = 32'h00000000;
result = alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'hffffffff; operand2 = 32'hfffffffe; result =
alu(opr, shift, operand1, operand2);
      #100 opr = 8; operand1 = 32'h00000000; operand2 = 32'hffffffff;
result = alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'h55555555; operand2 = 32'haaaaaaaa; result =
alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'hffffffff; operand2 = 32'hffffffff; result =
alu(opr, shift, operand1, operand2);
      #100 opr = 0; operand1 = 32'h00000000; operand2 = 32'hffffffff;
result = alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'h55555555; operand2 = 32'haaaaaaaa; result =
alu(opr, shift, operand1, operand2);
      #100 opr = 10; operand1 = 32'h00000000; operand2 = 32'hffffffff;
result = alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'h55555555; operand2 = 32'h55555555; result =
alu(opr, shift, operand1, operand2);
      #100 opr = 11; operand1 = 32'h00000000; operand2 = 32'hffffffff;
result = alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'h55555555; operand2 = 32'h55555555; result =
alu(opr, shift, operand1, operand2);
```

```
      #100 opr = 16; operand1 = 32'h12345678; shift = 2'h1; result =
alu(opr, shift, operand1, operand2);
      #100 opr = 17; operand1 = 32'h12345678; shift = 2'h1; result =
alu(opr, shift, operand1, operand2);
      #100 opr = 18; operand1 = 32'h12345678; shift = 2'h1; result =
alu(opr, shift, operand1, operand2);
      #100 operand1 = 32'h92345678; shift = 2'h1; result = alu(opr, shift,
operand1, operand2);
      #100 opr = 2; result = alu(opr, shift, operand1, operand2);
    end

  initial
    $monitor($stime, " op=%h, shift=%h, op1=%h, op2=%h, result=%h", opr,
shift, operand1, operand2, result);
endmodule

module test_npc;
  reg [5:0] op;
  reg signed [31:0] reg1, reg2;
  reg [31:0] branch, nonbranch, addr, nextpc;

  function [31:0] npc;
    input [5:0] op;
    input signed [31:0] reg1, reg2;
    input [31:0] branch, nonbranch, addr;

    case (op)
      6'd32: npc = (reg1 == reg2) ? branch : nonbranch;
      6'd33: npc = (reg1 != reg2) ? branch : nonbranch;
      6'd34: npc = (reg1 < reg2) ? branch : nonbranch;
      6'd35: npc = (reg1 <= reg2) ? branch : nonbranch;
      6'd40, 6'd41: npc = addr;
      6'd42: npc = reg1;
      default: npc = nonbranch;
    endcase
  endfunction

  initial
    begin
      op = 0;
      reg1 = 32'h00000010;
      reg2 = 32'h00000010;
      branch = 32'hffffffff;
      nonbranch = 32'h00000001;
      addr = 32'h00000002;
      nextpc = npc(op, reg1, reg2, branch, nonbranch, addr);
      #100 op = 1; nextpc = npc(op, reg1, reg2, branch, nonbranch, addr);
      #100 op = 32; reg2 = 32'h00000010; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 32; reg2 = 32'h00000001; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 32; reg2 = 32'h00000011; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 33; reg2 = 32'h00000010; nextpc = npc(op, reg1, reg2,
```

```
branch, nonbranch, addr);
      #100 op = 33; reg2 = 32'h00000001; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 33; reg2 = 32'h00000011; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 34; reg2 = 32'h00000010; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 34; reg2 = 32'h00000001; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 34; reg2 = 32'h00000011; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 35; reg2 = 32'h00000010; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 35; reg2 = 32'h00000001; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 35; reg2 = 32'h00000011; nextpc = npc(op, reg1, reg2,
branch, nonbranch, addr);
      #100 op = 40; nextpc = npc(op, reg1, reg2, branch, nonbranch, addr);
      #100 op = 41; nextpc = npc(op, reg1, reg2, branch, nonbranch, addr);
      #100 op = 42; nextpc = npc(op, reg1, reg2, branch, nonbranch, addr);
    end

  initial
    $monitor($stime, " op=%d, reg1=%h, reg2=%h, nextpc=%h", op, reg1,
reg2, nextpc);
endmodule
```

fetch.v

```
module fetch(pc, ins);
  input [7:0] pc;
  output [31:0] ins;

  reg [31:0] ins_mem [0:255];

  initial $readmemb("test.bnr", ins_mem);
  assign ins = ins_mem[pc];
endmodule

module test_fetch;
  reg clk, rstd;
  reg [31:0] pc;
  wire [31:0] ins;

  initial
    begin
      clk = 0;
      forever #50 clk = !clk;
    end
```

```
  initial
    begin
      rstd = 1;
      #10 rstd = 0;
      #20 rstd = 1;
    end

  always @ (negedge rstd or posedge clk)
    begin
      if (rstd == 0) pc <= 0;
      else if (clk == 1) pc <= pc+1;
    end

  initial
    $monitor($stime, " rstd=%b, clk=%b, pc=%d, ins=%b", rstd, clk, pc,
ins);

  fetch fetch_body(pc, ins);
endmodule
```

reg_file.v

```
module reg_file(clk, rstd, wr, ra1, ra2, wa, wren, rr1, rr2);
  input clk, rstd, wren;
  input [31:0] wr;
  input [4:0] ra1, ra2, wa;
  output [31:0] rr1, rr2;

  reg [31:0] rf [0:31];

  assign rr1 = rf[ra1];
  assign rr2 = rf[ra2];
  always @ (negedge rstd or posedge clk)
    begin
      if (rstd == 0) rf[0] <= 32'h00000000;
      else if (wren == 0) rf[wa] <= wr;
    end
endmodule

module test_register_file;
  reg clk, rstd, wren;
  reg [4:0] ra1, ra2, wa;
  wire [31:0] rr1, rr2;
  reg [31:0] wr;

  initial
    begin
      clk = 0;
      forever #50 clk = !clk;
    end
```

```
  initial
    begin
      rstd = 1;
      #30 rstd = 0;
      #40 rstd = 1;
      #10 wren = 0; ra1 = 1; ra2 = 2; wa = 3; wr = 32'haaaaaaaa;
      #100 ra1 = 3; ra2 = 3; wa = 4; wr = 32'h55555555;
      #100 ra1 = 4; ra2 = 5; wa = 5; wr = 32'h12345678;
      #100 ra1 = 5; ra2 = 4; wa = 6; wr = 32'h87654321;
      #100 ra1 = 6; ra2 = 0; wa = 1; wr = 32'h11111111;
      #100 ra1 = 1; ra2 = 6; wa = 2; wr = 32'h22222222;
      #100 ra1 = 1; ra2 = 2; wa = 7; wr = 32'h77777777;
      #100 wren = 1; ra1 = 1; ra2 = 2; wa = 8; wr = 32'haaaaaaaa;
      #100 ra1 = 3; ra2 = 4; wa = 9; wr = 32'h11111111;
      #100 ra1 = 5; ra2 = 6; wa = 10; wr = 32'hbbbbbbbb;
      #100 ra1 = 7; ra2 = 8; wa = 11; wr = 32'hcccccccc;
      #100 ra1 = 9; ra2 = 10; wa = 11; wr = 32'hdddddddd;
    end

  reg_file rf_body(clk, rstd, wr, ra1, ra2, wa, wren, rr1, rr2);

  initial
    $monitor($stime, " clk=%d, rstd=%d, ra1=%h, ra2=%h, wa=%h, rr1=%h,
rr2=%h, wr=%h, wren=%h", clk, rstd, ra1, ra2, wa, rr1, rr2, wr, wren);
endmodule
```