VE370 Project Report

Project 2 - Fall 2020

Gengchen Yang (518370910088)
Jiaying Xu (518021910738)
Qinhang Wu (518370910041)
Yuru Liu (518021910652)

Table of Contents

VE370 Project 2 Report

Introduction

Processors are commonly used in people's daily lives. Almost all electronic devices have processors in them. Meantime, pipeline processor is a wide-used and efficient implementation of processor. It is improved based on the single cycle processor. By adding registers to separate the execution of an instruction into several stages, this enables multiple instructions being executed at a time. The stages are IF(instruction fetch), ID(instruction decode), EX(execution), MEM(memory access), WB(write back), resulting in significant reduction in total execution time. The following figure is a top-level block diagram of Pipelined implementation of MIPS architecture.

Figure 1. Pipelined implementation of MIPS architecture

Pipeline Processor Implementation - Modules

The Verilog codes along with comments will be shown below each subtitle.

PC

1. First initialize the output: `out = -4`, where we use a negative value to indicate it will never be accessed
2. When there's a clock rising edge and `PCWrite==1`, output is renewed. Note we always assume branch **NOT TAKEN**, thus when `if_flush==0`, `out` is assigned with the following instruction; when `if_flush==1` (branch or jump instruction needs to be taken care of), `out` is assigned with target address

```
module program_counter(
  input clk,
  input [31:0] bj_next, // the input address; result from next stage
  input [31:0] normal_next, // normal next pc (pc+4)
  input c_if_flush, // if not asserted, pc=pc+4
  input c_PCWrite, // if not asserted, the PC won't move on
  output reg [31:0] out // the output address
);

  initial begin
    out = -4; // NEVER REACHED ADDRESS
  end


  always @(posedge clk) begin
    if (c_PCWrite == 1) begin
      if (c_if_flush == 0) begin
        out = normal_next;
      end else begin
        out = bj_next;
      end
    end
  end

endmodule
```

Instruction Memory

1. Inputs is 32-bit `addr`. Outputs are 6-bit control input `ctr` and 6-bit function code `funcode`.
2. Initialize `reg mem` using instructions given in `InstructionMem_for_P2_Demo.txt`
3. Retrieve outputs by accessing `memory` with mem index: `addr` shifted right by 2 bits

```
module instru_memory(
  input [31:0] addr,
  output reg [5:0] ctr, // [31-26]
  output reg [5:0] funcode, // [5-0]
);

  parameter SIZE_IM = 128; // size of this memory, by default 128*32
  reg [31:0] mem [SIZE_IM-1:0]; // instruction memory

  integer n;
  initial begin
    for(n=0;n<SIZE_IM;n=n+1) begin
      mem[n] = 32'b11111100000000000000000000000000;
    end
    $readmemb("C:\\Users\\William Wu\\Documents\\Mainframe Files\\UMJI-SJTU\\1 Academy\\20 Fall\\VE370\\Project\\p2\\single_cycle\\testcases\\testcase.txt",mem);
    instru = 32'b11111100000000000000000000000000;
  end

  always @(*) begin
    if (addr == -4) begin // init
      instru = 32'b11111100000000000000000000000000;
    end else begin
      instru = mem[addr >> 2];
    end
  end
  always @(*)begin
    ctr = instru[31:26];
    funcode = instru[5:0];
  end

endmodule
```

next_PC

1. Inputs are 32-bit `old` and `instru`, control signals `Jump`, `Branch`, `Bne`, and comparator output `zero`; and output are 32-bit `next` an`c_if_flush
2. Utilize a 3-to-1 MUX to select `next` as the next input of PC:

- If `Branch==1, Bne==0, zero_alter==1` (beq) or `Branch==1,Bne==1,zero_alter==1` (bne), a `branch` instruction is performed, that is: `next=pc+4+sign_ext<<2`
- Else if 'Jump==1'(j), a `jump` instruction is performed, that is: `next={pc[31:28],instru[25:0]<<2}`
- Else, `next=pc+4`

3. When a `branch` or `jump` instruction is successfully taken, assign `c_if_flush=1`

```
module next_pc(
  input [31:0] old, // IF/ID.pc
  input [31:0] instru, // IF/ID.instruction
    // [15-0] used for sign-extention
    // [25-0] used for shift-left-2
  input Jump, // ID.control
  input Branch,
  input Bne,
  input zero, // it now depends on the RG. comparator.
  output reg [31:0] next,
  output reg c_if_flush // IF.Flush
);

  // no connection with Hazard-detection

  reg [31:0] sign_ext;
  reg [31:0] old_alter; // pc+4
  reg [31:0] jump; // jump addr.
  reg zero_alter;

  initial begin
    next = 32'b0;
  end

  always @(old) begin
    old_alter = old + 4;
  end

  always @(zero,Bne) begin
    zero_alter = zero;
    if (Bne == 1) begin
      zero_alter = ! zero_alter;
    end
  end

  always @(instru) begin
    // jump-shift-left
    jump = {4'b0,instru[25:0],2'b0};

    // sign-extension
    if (instru[15] == 1'b0) begin
      sign_ext = {16'b0,instru[15:0]};
    end else begin
      sign_ext = {{16{1'b1}},instru[15:0]};
    end
    sign_ext = {sign_ext[29:0],2'b0}; // shift left
  end

  always @(instru or old_alter or jump) begin
    jump = {old_alter[31:28],jump[27:0]};
  end

  always @(*) begin
    // assign next program counter value
    if (Jump == 1) begin
      next = jump;
      c_if_flush = 1;
    end else begin
      if (Branch == 1 & zero_alter == 1) begin
        next = old_alter + sign_ext;
        c_if_flush = 1;
      end else begin
        next = old_alter;
        c_if_flush = 0;
      end
    end
  end

endmodule
```

Register File

1. Inputs are clock signal `clk`, 32-bit instruction `instru`, 32-bit write data `WriteData`,5-bit `writeReg` and control signal `RegDst`; outputs are 32-bit `ReadData1` and `ReadData2`, and comparator result `reg_zero`
2. If `RegWrite==1`, and `WriteReg` is identical to read address, directly assign read data with write data
3. If there's clock rising edge and `RegWrite==1`, write `WriteData` into `RegData[WriteReg]`

```
module register(
  input clk,
  input [31:0] instru, // the raw 32-bit instruction
  input RegWrite, // from WB stage!
  input RegDst,
  input [31:0] WriteData, // from WB stage
  input [4:0] WriteReg, // from WB stage
  output reg [31:0] ReadData1,
  output reg [31:0] ReadData2,
  output reg reg_zero // comparator result
);

  reg [31:0] RegData [31:0]; // register data

  // initialize the register data
  integer i;
  initial begin
    for(i=0;i<32;i=i+1) begin
      RegData[i] = 32'b0;
    end
  end

  always @(*) begin
```

```
    if(WriteReg==instru[25:21] && RegWrite==1) begin
      ReadData1 = WriteData;
    end else begin
      ReadData1 = RegData[instru[25:21]];
    end

    if(WriteReg==instru[20:16] && RegWrite==1) begin
      ReadData2 = WriteData;
    end else begin
      ReadData2 = RegData[instru[20:16]];
    end
  end

  always @(posedge clk) begin // RegWrite, RegDst, WriteData, instru)
    if (RegWrite == 1'b1) begin
      $display("Reg_WriteData: 0x%H | WriteReg: %d",WriteData,
WriteReg);
      RegData[WriteReg] = WriteData;
    end
  end

  always @(*) begin
    if (ReadData1 == ReadData2) begin
      reg_zero = 1;
    end else begin
      reg_zero = 0;
    end
  end

endmodule
```

ALU

1. The arithmetic logic unit is designed for `add`, `sub`, `and`, `or`, `slt`, `nor` operations.
2. For forwarding path related to `data1`:

- When selection signal `c_data1_src=2'b00` (no data hazard), `data1_fin=data1`
- When selection signal `c_data1_src=2'b01` (1 & 3data hazard from MEM/WB register), `data1_fin=mem_wb_fwd`
- When selection signal `c_data1_src=2'b10` (1 & 2 data hazard from EX/MEM register), `data1_fin=ex_mem_fwd`

3. For forwarding path related to `read2`, 2 MUXes are on this path We also need to consider the future `WriteData` input of Data memory(namely, output of forwarding path MUX), in our case: `data2_fwd`:

- When `ALUSrc==0`, no need to consider immediate number:
  - When selection signal `c_data2_src=2'b00` (no data hazard), `data2_fin=data2` `data2_fwd=data2_fwd_old`
  - When selection signal `c_data2_src=2'b01` (1 & 3data hazard from MEM/WB register), `data2_fin=mem_wb_fwd` `data2_fwd=data2_fin`

    - When selection signal `c_data2_src=2’b00` (1 & 2 data hazard from EX/MEM register), `data2_fin=ex_mem_fwd` `data2_fwd=data2_fin`
- When `ALUSrc==1`, `data2_fin` is sign extended 32-bit `instru[15:0]`

4. If `ALUresult==0`, assign `zero=1`

```
module alu(
  input [31:0] data1,
  input [31:0] read2, // candidate for data2
  input [31:0] instru, // candidate for data2; used for sign-extension
  input ALUSrc,
  input [3:0] ALUcontrol,

  input [31:0] ex_mem_fwd, // forwarded data from EX/MEM
  input [31:0] mem_wb_fwd, // forwarded data from MEM/WB
  input [1:0] c_data1_src,
  input [1:0] c_data2_src,

  output reg [31:0] data2_fwd, // connect to DM
  input [31:0] data2_fwd_old,
  output reg zero,
  output reg [31:0] ALUresult
);

  reg [31:0] data1_fin;
  reg [31:0] data2_fin;


  always @(*) begin
    case (c_data1_src)
      2'b00: // from current stage
        data1_fin = data1;
      2'b10: // from EX/MEM
        data1_fin = ex_mem_fwd;
      2'b01: // from from MEM/WB
        data1_fin = mem_wb_fwd;
      default:
        ;
    endcase
  end

  always @(*) begin
    if (ALUSrc == 0) begin
      case (c_data2_src)
        2'b00: begin// from current stage
          data2_fin = read2;
          data2_fwd = data2_fwd_old;
        end
        2'b10: begin// from EX/MEM
          data2_fin = ex_mem_fwd;
          data2_fwd = data2_fin;
        end
        2'b01: begin// from from MEM/WB
          data2_fin = mem_wb_fwd;
          data2_fwd = data2_fin;
        end
        default:
          ;
      endcase

    end else begin
      // SignExt[Instru[15:0]]
      if (instru[15] == 1'b0) begin
        data2_fin = {16'b0,instru[15:0]};
      end else begin
        data2_fin = {{16{1'b1}},instru[15:0]};
      end
    end
  end

  always @(*) begin
    case (ALUcontrol)
      4'b0000: // AND
        ALUresult = data1_fin & data2_fin;
      4'b0001: // OR
        ALUresult = data1_fin | data2_fin;
      4'b0010: // ADD
        ALUresult = data1_fin + data2_fin;
      4'b0110: // SUB
        ALUresult = data1_fin - data2_fin;
      4'b0111: // SLT
        ALUresult = (data1_fin < data2_fin) ? 1 : 0;
      4'b1100: // NOR
        ALUresult = data1_fin |~ data2_fin;
      default:
        ;
    endcase
    if (ALUresult == 0) begin
      zero = 1;
    end else begin
      zero = 0;
    end
  end

endmodule
```

ALU Control

1. Inputs are 2-bit control signal `ALUOp`, 6-bit `instru`, and output is 4-bit `ALUcontrol`
2. `ALUOp` is utilized to distinguish instructions that have different operations in `ALU` component Specification: `andi` has `ALUOp=2’b11`, thus correponding `ALUcontrol=4’b0000` (AND)
3. `funct` is utilized to further distinguish different R-format instructions

```
module alu_control(
  input [1:0] ALUOp,
  input [5:0] instru,
  output reg [3:0] ALUcontrol
);

  always @(*) begin
    case (ALUOp)
      2'b00:
        ALUcontrol = 4'b0010;
      2'b01:
        ALUcontrol = 4'b0110;
      2'b10: begin
        case (instru)
          6'b100000: // add
            ALUcontrol = 4'b0010;
          6'b100010: // sub
            ALUcontrol = 4'b0110;
          6'b100100: // and
            ALUcontrol = 4'b0000;
          6'b100101: // or
            ALUcontrol = 4'b0001;
          6'b101010: // slt
            ALUcontrol = 4'b0111;
          default:
            ;
        endcase
      end
      2'b11:
        ALUcontrol = 4'b0000;
      default:
        ;
    endcase
  end

endmodule
```

Control

1. Input are 32-bit instruction `instru`, and control signal `c_clearControl`, and output are control signals `RegDst`, `Jump`, `Branch`, `Bne`, `MemRead`, `MemWrite`, `MemtoReg`, `ALUSrc`, `RegWrite` and 2-bit `ALUOp`. First Initialize all the control signals as `0`
2. If `c_clearControl==0`, control signals are assigned according to `instru`; otherwise, control signals remain `0`
3. For instructions of different types, different control signals are assigned
4. Specifications:

- For `addi`, in `ALU` component, it conducts `$register + 32-bit immediate number`, thus `ALUOp=2'b00`, the same as `sw` and `lw`
- For `andi`, it needs special care in `ALU` component, thus assign `ALUOp=2'b11` so that to differentiate it from other instructions

- For `j`, `ALUOp` is not needed in its execution, thus we assign `ALUOp=2'b01`
- if the instruction is `beq`, `Branch=1,Bne=0`; if the instruction is `bne`, `Branch=1,Bne=1`

```
module control(
  input [31:0] instru,
  input c_clearControl,
  output reg RegDst,
  output reg Jump,
  output reg Branch,
  output reg Bne, // 1 indicates bne
  output reg MemRead,
  output reg MemtoReg,
  output reg [1:0] ALUOp,
  output reg MemWrite,
  output reg ALUSrc,
  output reg RegWrite

  initial begin
    RegDst = 0;
    Jump = 0;
    Branch = 0;
    MemRead = 0;
    MemtoReg = 0;
    ALUOp = 2'b00;
    MemWrite = 0;
    ALUSrc = 0;
    RegWrite = 0;
  end

  always @(*) begin
    if (c_clearControl == 0) begin
      case (instru[31:26])
        6'b000000: begin// ARITHMETIC
          RegDst = 1;
          ALUSrc = 0;
          MemtoReg = 0;
          RegWrite = 1;
          MemRead = 0;
          MemWrite = 0;
          Branch = 0;
          Bne = 0;
          ALUOp = 2'b10;
          Jump = 0;
        end
        6'b001000: begin// addi
          RegDst = 0;
          ALUSrc = 1;
          MemtoReg = 0;
          RegWrite = 1;
          MemRead = 0;
          MemWrite = 0;
          Branch = 0;
          Bne = 0;
          ALUOp = 2'b00;
          Jump = 0;
        end
        6'b001100: begin// andi
          RegDst = 0;
          ALUSrc = 1;
          MemtoReg = 0;
          RegWrite = 1;
          MemRead = 0;
          MemWrite = 0;
          Branch = 0;
          Bne = 0;
          ALUOp = 2'b11;
          Jump = 0;
        end
        6'b100011: begin // lw
          RegDst = 0;
          ALUSrc = 1;
          MemtoReg = 1;
          RegWrite = 1;
          MemRead = 1;
          MemWrite = 0;
          Branch = 0;
          Bne = 0;
          ALUOp = 2'b00;
          Jump = 0;
        end
        6'b101011: begin // sw
          RegDst = 0; // x
          ALUSrc = 1;
          MemtoReg = 0; // x
          RegWrite = 0;
          MemRead = 0;
          MemWrite = 1;
          Branch = 0;
          Bne = 0;
          ALUOp = 2'b00;
          Jump = 0;
        end
        6'b000100: begin // beq
          RegDst = 0; // x
          ALUSrc = 0;
          MemtoReg = 0; // x
          RegWrite = 0;
          MemRead = 0;
          MemWrite = 0;
          Branch = 1;
          Bne = 0;
          ALUOp = 2'b01;
          Jump = 0;
        end
```

```verilog
          6'b000101: begin // bne
            RegDst = 0; // x
            ALUSrc = 0;
            MemtoReg = 0; // x
            RegWrite = 0;
            MemRead = 0;
            MemWrite = 0;
            Branch = 1;
            Bne = 1;
            ALUOp = 2'b01;
            Jump = 0;
          end
          6'b000010: begin // j
            RegDst = 0; // x
            ALUSrc = 0;
            MemtoReg = 0; // x
            RegWrite = 0;
            MemRead = 0;
            MemWrite = 0;
            Branch = 0;
            Bne = 0;
            ALUOp = 2'b01;
            Jump = 1;
          end
          default: begin
            RegDst = 0; // x
            ALUSrc = 0;
            MemtoReg = 0; // x
            RegWrite = 0;
            MemRead = 0;
            MemWrite = 0;
            Branch = 0;
            Bne = 0;
            ALUOp = 2'b00;
            Jump = 0;
          end
        endcase
      end else begin
        RegDst = 0;
        Jump = 0;
        Branch = 0;
        MemRead = 0;
        MemtoReg = 0;
        ALUOp = 2'b00;
        MemWrite = 0;
        ALUSrc = 0;
        RegWrite = 0;
      end
    end

endmodule
```

Data Memory

1. Inputs are 32-bit reading address `addr` and `ALUresult`, 32-bit write data `wData`, control signals `MemWrite`, `MemRead` and `MemtoReg`, and clock signal `clk`; output is 32-bit `rData`
2. When there's a clock rising edge and `memWrite==1’b1`, write `mem[addr]` with `rData`
3. If `MemRead==1` (lw), assign `rData=mem[addr]`; otherwise, assign `rData=ALUresult` note: `addr` has already been shifted right by 2 bits

```
module data_memory(
  input clk,
  input [31:0] addr,
  input [31:0] wData,
  input [31:0] ALUresult,
  input MemWrite,
  input MemRead,
  input MemtoReg,
  output reg [31:0] rData
);

  parameter SIZE_DM = 128; // size of this memory, by default 128*32
  reg [31:0] mem [SIZE_DM-1:0]; // instruction memory

  // initially set default data to 0
  integer i;
  initial begin
    for(i=0; i<SIZE_DM-1; i=i+1) begin
      mem[i] = 32'b0;
    end
  end

  always @(addr or MemRead or MemtoReg or ALUresult) begin
    if (MemRead == 1) begin
      if (MemtoReg == 1) begin
        rData = mem[addr];
      end else begin
        rData = ALUresult; // x ?
      end
    end else begin
      rData = ALUresult;
    end
  end

  always @(posedge clk) begin // MemWrite, wData, addr
    if (MemWrite == 1) begin
      mem[addr] = wData;
    end
  end

endmodule
```

Pipeline Registers

There are four Pipeline Registers. They divide the whole pipeline processor into 5 different stages. They temporarily store the output from different modules and receives output from hazard detection unit and forwarding unit to prevent hazard. Take register IF/ID as an example, since they all have a similar structure.

1. Inputs are clock signal `clk`, write signal `c_IFIDWrite`, flush signal to prevent hazard `c_if_flush`, 6-bit signal for `ctr_in`, 6-bit for storing function code `funcode_in`, 32-bit to store the whole instruction `instru_in`, 32-bit to store the next program counter's address `nextpc_in`.
2. Outputs are 6-bit function code `funcode`, 6-bit instruction code `instru`, 32-bit of next program counter `nextpc` and the result of PC+4 `normal_nextpc`.

For `IF/ID` register:

1. If `c_IFIDWrite==0`, content in `IF/ID` register is reserved
2. If `c_IFIDWrite==1`:

- If `c_if_flush==1`, flush all the data in `IF/ID` register
- If `c_if_flush==0`, transfer all the data to `ID` stage

```
module pr_if_id(
  input clk,
  input c_IFIDWrite,
  input c_if_flush,
  input [5:0] ctr_in,
  input [5:0] funcode_in,
  input [31:0] instru_in,
  input [31:0] nextpc_in, // next pc
  output reg [5:0] ctr, // [31-26]
  output reg [5:0] funcode, // [5-0]
  output reg [31:0] instru, // [31-0]
  output reg [31:0] nextpc, // to next_pc.v
  output reg [31:0] normal_nextpc // pc+4, passing to pc
);

  initial begin
    ctr = 6'b111111;
    funcode = 6'b000000;
    instru = 32'b11111100000000000000000000000000;
    nextpc = 32'b0;
  end

  always @(posedge clk) begin
    if (c_IFIDWrite == 1) begin
      if (c_if_flush == 0) begin
        ctr = ctr_in;
        funcode = funcode_in;
        instru = instru_in;
        nextpc = nextpc_in;
      end else begin
        ctr = 6'b111111;
        funcode = 6'b000000;
        instru = 32'b11111100000000000000000000000000;
        nextpc = 32'b0;
      end
    end
  end

  always @(nextpc_in) begin
    normal_nextpc = nextpc_in + 4;
  end

endmodule
```

For `ID/EX` register: Because `EX.flush` is performed in `Control`, thus in this register, only data transfer is performed

```
module pr_id_ex(
  input clk,
  // this can be modified to one 11-bit control signal, but I'm running out of time...
  input RegDst_in, // EX
  input Jump_in, // MEM
  input Branch_in, // MEM
  input Bne_in, // MEM
  input MemRead_in, // MEM
  input MemtoReg_in, // WB
  input [1:0] ALUOp_in, // EX
  input MemWrite_in, // MEM
  input ALUSrc_in, // EX
  input RegWrite_in, // WB
  output reg RegDst,
  output reg Jump,
  output reg Branch,
  output reg Bne, // 1 indicates bne
  output reg MemRead,
  output reg MemtoReg,
  output reg [1:0] ALUOp,
  output reg MemWrite,
  output reg ALUSrc,
  output reg RegWrite,

  input [31:0] nextPc_in, // from pc
  output reg [31:0] nextPc,
  input [31:0] ReadData1_in, // from reg
  input [31:0] ReadData2_in,
  input [5:0] funcode_in,
  output reg [31:0] ReadData1,
  output reg [31:0] ReadData2,
  output reg [5:0] funcode,
  input [31:0] instru_in, // raw instruction from IF/ID-reg
  output reg [31:0] instru
);

```

```
  initial begin
    RegDst = 0;
    Jump = 0;
    Branch = 0;
    Bne = 0;
    MemRead = 0;
    MemtoReg = 0;
    ALUOp = 2'b00;
    MemWrite = 0;
    ALUSrc = 0;
    RegWrite = 0;
  end

  always @(posedge clk) begin
    RegDst = RegDst_in;
    Jump = Jump_in;
    Branch = Branch_in;
    Bne = Bne_in;
    MemRead = MemRead_in;
    MemtoReg = MemtoReg_in;
    ALUOp = ALUOp_in;
    MemWrite = MemWrite_in;
    ALUSrc = ALUSrc_in;
    RegWrite = RegWrite_in;

    nextPc = nextPc_in;
    ReadData1 = ReadData1_in;
    ReadData2 = ReadData2_in;
    funcode = funcode_in;
    instru = instru_in;
  end

endmodule
```

For `EX/MEM` registeer:

```
module pr_ex_mem(
  input clk,
  input Jump_in, // MEM
  input Branch_in, // MEM
  input Bne_in, // MEM
  input MemRead_in, // MEM
  input MemtoReg_in, // WB
  input MemWrite_in, // MEM
  input RegWrite_in, // WB
  input RegDst_in, // EX, only used here
  output reg Jump,
  output reg Branch,
  output reg Bne,
  output reg MemRead,
  output reg MemtoReg,
  output reg MemWrite,
  output reg RegWrite,

  input zero_in,
  input [31:0] ALUresult_in,
  input [31:0] instru_in, // receive instru and transf to WriteReg (w.b. to Reg)
  input [31:0] regData2_in,
  output reg zero,
  output reg [31:0] ALUresult,
  output reg [4:0] WriteReg,
  output reg [31:0] instru,
  output reg [31:0] regData2
);

  // TODO: add support for EX.Flush

  initial begin
    Jump = 0;
    Branch = 0;
    Bne = 0;
    MemRead = 0;
    MemtoReg = 0;
    MemWrite = 0;
    RegWrite = 0;
  end

  always @(posedge clk) begin
    Jump = Jump_in;
    Branch = Branch_in;
    Bne = Bne_in;
    MemRead = MemRead_in;
    MemtoReg = MemtoReg_in;
    MemWrite = MemWrite_in;
    RegWrite = RegWrite_in;
    instru = instru_in;

    zero = zero_in;
    ALUresult = ALUresult_in;
    regData2 = regData2_in;
    if (RegDst_in == 1'b0) begin
      WriteReg = instru_in[20:16];
    end else begin
      WriteReg = instru_in[15:11];
    end
  end

endmodule
```

For `MEM/WB` register:

```
module pr_mem_wb(
  input clk,
```

```verilog
  input MemtoReg_in, // WB
  input RegWrite_in, // WB
  output reg MemtoReg,
  output reg RegWrite,

  input [31:0] wData_in,
  input [4:0] writeReg_in,
  input [31:0] instru_in,
  output reg [31:0] wData,
  output reg [4:0] writeReg,
  output reg [31:0] instru
);

  initial begin
    MemtoReg = 0;
    RegWrite = 0;
  end

  always @(posedge clk) begin
    MemtoReg = MemtoReg_in;
    RegWrite = RegWrite_in;

    wData = wData_in;
    writeReg = writeReg_in;
    instru = instru_in;
  end

endmodule
```

Forwarding Unit

1. Inputs are data and control signals in `IF/ID`, `ID/EX`, `EX/MEM` and `MEM/WB` register, and outputs are select signals `FullFwdA/FullFwdB` and `BranFwdA/BranFwdB`
2. `FullFwdA/FullFwdB` are select signals for full forwarding paths, while `BranFwdA/BranFwdB` are select signals for branch forwarding paths
3. For full forwarding paths, when there's a 1&2 hazard (`R-format` instruction), `FullFwdA/FullFwdB=2’b10`; when there's a 1&3 hazard(`R-format` or `lw` instruction) , `FullFwdA/FullFwdB=2’b01`
4. For branch forwarding paths, when there's a 1&3 hazard (`R-format` instruction), `BranFwdA/BranFwdB=1’b1`

```verilog
// forwarding unit, in stage EX

module forward(
  input [31:0] ex_instru, // ID/EX.instru
  input [31:0] ex_mem_instru, // EX/MEM.WriteReg
  input [31:0] mem_wb_instru, // MEM/WB.WriteReg
  input c_ex_mem_RegWrite, // EX/MEM.RegWrite
  input c_mem_wb_RegWrite, // MEM/WB.RegWrite
  output reg [1:0] c_data1_src, // ALU.data1.src (A)
  output reg [1:0] c_data2_src // ALU.data2.src (B)
```

```
);
  // Note: ex_instru[25:21] == ID/EX.Rs
  //       ex_instru[20:16] == ID/EX.Rt
  reg [4:0] ex_mem_wReg,mem_wb_wReg; // exactly Rd

  // if I-type, use Rt; if R-type, use Rd
  always @(*) begin
    if(ex_mem_instru[31:26] == 0) begin
      ex_mem_wReg = ex_mem_instru[15:11];
    end else begin
      ex_mem_wReg = ex_mem_instru[20:16]; // I
    end
    if(mem_wb_instru[31:26] == 0) begin
      mem_wb_wReg = mem_wb_instru[15:11];
    end else begin
      mem_wb_wReg = mem_wb_instru[20:16]; // I
    end
  end

  always @(*) begin
    if(c_ex_mem_RegWrite==1 & (ex_mem_wReg != 0) & (ex_mem_wReg == ex_instru[25:21])) begin
      c_data1_src = 2'b10; // from EX/MEM
    end else if (c_mem_wb_RegWrite==1 & (mem_wb_wReg != 0) & (mem_wb_wReg == ex_instru[25:21]) &
    !(c_ex_mem_RegWrite==1 & (ex_mem_wReg != 0) & (ex_mem_wReg == ex_instru[25:21]))) begin
      c_data1_src = 2'b01; // from from MEM/WB
    end else begin
      c_data1_src = 2'b00; // from current stage
    end
  end

  always @(*) begin
    if(c_ex_mem_RegWrite==1 & (ex_mem_wReg != 0) & (ex_mem_wReg == ex_instru[20:16])) begin
      c_data2_src = 2'b10; // from EX/MEM
    end else if (c_mem_wb_RegWrite==1 & (mem_wb_wReg != 0) & (mem_wb_wReg == ex_instru[20:16]) &
    !(c_ex_mem_RegWrite==1 & (ex_mem_wReg != 0) & (ex_mem_wReg == ex_instru[20:16]))) begin
      c_data2_src = 2'b01; // from from MEM/WB
    end else begin
      c_data2_src = 2'b00; // from current stage
    end
  end

endmodule

module Forwarding_bonus(
    input [4:0] IF_ID_Rs,IF_ID_Rt,ID_EX_Rs,ID_EX_Rt,EX_MEM_Rd,MEM_WB_Rd,
```

```verilog
    // note: MEM_WB_Rd is destination of MEM/WB register, for lw & addi: rt, add: rd
    input EX_MEM_RegWrite,MEM_WB_RegWrite,ID_beq,ID_bne,
    output reg [1:0] FullFwdA,FullFwdB,
    output reg BranFwdA,BranFwdB
);

initial begin
    FullFwdA=2'b00;
    FullFwdB=2'b00;
    BranFwdA=1'b0;
    BranFwdB=1'b0;
end

always @(*) begin // FullFwdA
if ((ID_EX_Rs == EX_MEM_Rd) && EX_MEM_RegWrite && (EX_MEM_Rd!=5'b0)) // R-format
    FullFwdA=2'b10;

else if ((ID_EX_Rs == MEM_WB_Rd) && MEM_WB_RegWrite && (MEM_WB_Rd!=5'b0))  // lw & R-format
    FullFwdA=2'b01;

else
    FullFwdA=2'b00;
end

always @(*) begin // FullFwdB
if ((ID_EX_Rt == EX_MEM_Rd) && EX_MEM_RegWrite && (EX_MEM_Rd!=5'b0))
    FullFwdB=2'b10;

else if ((ID_EX_Rt == MEM_WB_Rd) && MEM_WB_RegWrite && (MEM_WB_Rd!=5'b0))
    FullFwdB=2'b01;

else
    FullFwdB=2'b00;

end

always @(*) begin //BranFwdA
    if((ID_beq || ID_bne) && (IF_ID_Rs == EX_MEM_Rd) && (EX_MEM_Rd!=5'b0) && EX_MEM_RegWrite)
        BranFwdA=1'b1;

    else
        BranFwdA=1'b0;

end


always @(*) begin //BranFwdB
```

```
    if((ID_beq || ID_bne) && (IF_ID_Rt == EX_MEM_Rd) && (EX_MEM_Rd!=5'b0) && EX_MEM_RegWrite)
        BranFwdB=1'b1;

    else
        BranFwdB=1'b0;

end

endmodule
```

Hazard Detection Unit

1. Inputs are data and control signals in `IF/ID`, `ID/EX`, `EX/MEM` register, and outputs are control signals working on `IF/ID`, `ID/EX` register and `PC`
2. Assign `PCHold=1` when there's hazard detected. Hazards are classified as:

- data hazard: load use hazard
- control hazard: 1&2, 1&3 `lw` and `branch` hazard
- control hazard: 1&2 `R-format` and `branch` hazard
- control hazard: 1&2 `andi` and `branch` hazard

3. When there's hazard detected, assign control signals in hazard detection unit: `PCWrite`, `IF_ID_Write` and `clearControl`

```
module hazard_det(
  input id_ex_memRead, // ID/EX.MemRead
  input [31:0] if_id_instru, // IF/ID.instru
  input [31:0] id_ex_instru, // ID/EX.instru
  output reg c_PCWrite, // -> PC
  output reg c_IFIDWrite, // -> IF/ID pipelined reg
  output reg c_clearControl // -> control (ID/EX.Flush)
  // FIXME: bool convention
);


  initial begin
    c_PCWrite = 1;
    c_IFIDWrite = 1;
    c_clearControl = 0;
  end

  always @(*) begin
    if (id_ex_memRead==1 && ((id_ex_instru[20:16] == if_id_instru[25:21]) || (id_ex_instru[20:16] == if_id_instru[20:16]))) begin
      c_PCWrite = 0; // if PCWrite==0, don't write in new instruction, IM decode the current instruction again
      c_IFIDWrite = 0; // if IF_ID_Write==0, IF/ID register keeps the current instruction
      c_clearControl = 1; // if ID_EX_Flush=1, all control signals in ID/EX are 0
```

```verilog
    end else begin
      c_PCWrite = 1;
      c_IFIDWrite = 1;
      c_clearControl = 0;
    end
  end

endmodule


module Hazard_bonus( // TODO: [bonus] consider control hazard: branch
  // created by lyr
    input [4:0] IF_ID_Rs,IF_ID_Rt,ID_EX_Rt,EX_MEM_Rt,ID_EX_Rd,
    input ID_EX_MemRead,EX_MEM_MemRead,ID_beq,ID_bne,ID_EX,RegWrite,ID_jump,ID_equal,ID_EX_RegWrite,
    output PCWrite,IF_ID_Write,ID_EX_Flush,IF_Flush
);

wire PCHold; // if PCHold==1, hold PC and IF/ID

assign PCHold = ( (ID_EX_MemRead) && (ID_EX_Rt == IF_ID_Rs || ID_EX_Rt == IF_ID_Rt) ) // lw hazard
                || ( (ID_beq || ID_bne) && (ID_EX_MemRead) && (ID_EX_Rt == IF_ID_Rs || ID_EX_Rt == IF_ID_Rt) ) // lw followed by branch
                || ( (ID_beq || ID_bne) && (EX_MEM_MemRead) && (EX_MEM_Rt == IF_ID_Rs || EX_MEM_Rt == IF_ID_Rt) ) // lw followed by nop and then branch
                || ( (ID_beq || ID_bne) && (ID_EX_RegWrite) && (ID_EX_Rd != 5'b0) && (ID_EX_Rd == IF_ID_Rs || ID_EX_Rd == IF_ID_Rt) ) // R-format followed by branch
                || ( (ID_beq || ID_bne) && (ID_EX_RegWrite) && (ID_EX_Rd == 5'b0) && (ID_EX_Rt == IF_ID_Rs || ID_EX_Rt == IF_ID_Rt) ); // addi followed by branch

// note we leave out the case that R-format followed by a nop then a branch, because that is solved by forwarding path
assign PCWrite=~PCHold; // if PCWrite==0, don't write in new instruction, IM decode the current instruction again

assign IF_ID_Write=~PCHold; // if IF_ID_Write==0, IF/ID register keeps the current instruction

assign ID_EX_Flush=PCHold; // if ID_EX_Flush=1, all control signals in ID/EX are 0 (implemented in ID/EX register later)

assign IF_Flush = (PCHold==0) && ( (ID_jump) || (ID_beq && ID_equal) || (ID_bne && ID_equal));

endmodule

```

Pipeline top module

This is the module that links all modules above together to form a complete datapath.

1. The input for pipeline top module is clock signal.
2. There is no output for main module, since the instructions all run within the processor.

```
timescale 1ns / 1ps
`include "program_counter.v"
`include "next_pc.v"
`include "instru_memory.v"
`include "pr_if_id.v"
`include "register.v"
`include "control.v"
`include "hazard_det.v"
`include "pr_id_ex.v"
`include "alu.v"
`include "alu_control.v"
`include "forward.v"
`include "pr_ex_mem.v"
`include "data_memory.v"
`include "pr_mem_wb.v"

module main(
  input clk // clock signal for PC, RD, PRs
);

  wire [31:0] pc_in;
  wire [31:0] normal_next_pc;

  wire [5:0]  ctr_a,
              ctr_b,
              funcode_a,
              funcode_b,
              funcode_d;
  wire [31:0] instru_a,
              instru_b,
              nextpc_a,
              nextpc_b;

  wire c_RegDst_1_a,c_Jump_1_a,c_Branch_1_a,c_Bne_1_a,
       c_MemRead_1_a,c_MemtoReg_1_a,c_MemWrite_1_a,
       c_ALUSrc_1_a,c_RegWrite_1_a;
  wire [1:0] c_ALUOp_1_a;
  wire c_RegDst_1_b,c_Jump_1_b,c_Branch_1_b,c_Bne_1_b,
       c_MemRead_1_b,c_MemtoReg_1_b,c_MemWrite_1_b,
       c_ALUSrc_1_b,c_RegWrite_1_b;
  wire [1:0] c_ALUOp_1_b;
  wire [31:0] nextpc_d;
  wire [31:0] r_read1_a,
              r_read1_b,
              r_read2_a,
              r_read2_b,r_read2_d;
  wire [31:0] instru_d,instru_f,instru_h;

  wire c_if_flush;

  wire c_PCWrite_w;
  wire c_IFIDWrite_w;
  wire c_clearControl_w;

  wire [3:0] ALUcontrol_out;

  wire [1:0] c_data1_src_w; // forward
  wire [1:0] c_data2_src_w;

  wire [31:0] rData2_ex_fwd;

  wire c_Jump_2_b,c_Branch_2_b,c_Bne_2_b,c_MemRead_2_b,
       c_MemtoReg_2_b,c_MemWrite_2_b,c_RegWrite_2_b;
  wire zero_a,zero_b,zero_reg;
  wire [31:0] ALUresult_a, ALUresult_b;
  wire [4:0] WriteReg_b;

  wire c_MemtoReg_3_b, c_RegWrite_3_b;
  wire [4:0] WriteReg_d;

  wire [31:0] memWriteData_a,
              memWriteData_b;

  program_counter asset_pc(
    .clk (clk),
    .bj_next (pc_in),
    .normal_next (normal_next_pc),
    .c_if_flush (c_if_flush),
    .c_PCWrite (c_PCWrite_w),
    .out (nextpc_a)
  );

  instru_memory asset_im(
    .addr (nextpc_a),
    .ctr (ctr_a),
    .funcode (funcode_a),
    .instru (instru_a)
  );

  pr_if_id asset_ifid(
    .clk (clk),
    .c_IFIDWrite (c_IFIDWrite_w),
    .c_if_flush (c_if_flush),
    .ctr_in (ctr_a),
    .funcode_in (funcode_a),
    .instru_in (instru_a),
    .nextpc_in (nextpc_a),
```

```verilog
    .ctr (ctr_b),
    .funcode (funcode_b),
    .instru (instru_b),
    .nextpc (nextpc_b), // pc instead of pc+4
    .normal_nextpc (normal_next_pc)
  );

  next_pc asset_nextPc(
    .old (nextpc_b),
    .instru (instru_b),
    .Jump (c_Jump_1_a),
    .Branch (c_Branch_1_a),
    .Bne (c_Bne_1_a),
    .zero (zero_reg),
    .next (pc_in),
    .c_if_flush (c_if_flush)
  );

  hazard_det asset_hDet(
    .id_ex_memRead (c_MemRead_1_b),
    .if_id_instru (instru_b),
    .id_ex_instru (instru_d), // TODO
    .c_PCWrite (c_PCWrite_w),
    .c_IFIDWrite (c_IFIDWrite_w),
    .c_clearControl (c_clearControl_w)
  );

  register asset_reg(
    .clk (clk),
    .instru (instru_b),
    .RegWrite (c_RegWrite_3_b), // from WB stage
    .RegDst (c_RegDst_1_a),
    .WriteData (memWriteData_b),
    .WriteReg (WriteReg_d),
    .ReadData1 (r_read1_a),
    .ReadData2 (r_read2_a),
    .reg_zero (zero_reg)
  );

  control asset_control(
    .instru (instru_b),
    .c_clearControl (c_clearControl_w),
    .RegDst (c_RegDst_1_a),
    .Jump (c_Jump_1_a),
    .Branch (c_Branch_1_a),
    .Bne (c_Bne_1_a),
    .MemRead (c_MemRead_1_a),
    .MemtoReg (c_MemtoReg_1_a),
    .ALUOp (c_ALUOp_1_a),
    .MemWrite (c_MemWrite_1_a),
    .ALUSrc (c_ALUSrc_1_a),
    .RegWrite (c_RegWrite_1_a)
```

```
);

pr_id_ex asset_idex(
  .clk (clk),
  .RegDst_in (c_RegDst_1_a),
  .Jump_in (c_Jump_1_a),
  .Branch_in (c_Branch_1_a),
  .Bne_in (c_Bne_1_a),
  .MemRead_in (c_MemRead_1_a),
  .MemtoReg_in (c_MemtoReg_1_a),
  .ALUOp_in (c_ALUOp_1_a),
  .MemWrite_in (c_MemWrite_1_a),
  .ALUSrc_in (c_ALUSrc_1_a),
  .RegWrite_in (c_RegWrite_1_a),
  .RegDst (c_RegDst_1_b),
  .Jump (c_Jump_1_b),
  .Branch (c_Branch_1_b),
  .Bne (c_Bne_1_b),
  .MemRead (c_MemRead_1_b),
  .MemtoReg (c_MemtoReg_1_b),
  .ALUOp (c_ALUOp_1_b),
  .MemWrite (c_MemWrite_1_b),
  .ALUSrc (c_ALUSrc_1_b),
  .RegWrite (c_RegWrite_1_b),

  .nextPc_in (nextpc_b),
  .nextPc (nextpc_d),
  .ReadData1_in (r_read1_a),
  .ReadData2_in (r_read2_a),
  .funcode_in (funcode_b),
  .ReadData1 (r_read1_b),
  .ReadData2 (r_read2_b),
  .instru_in (instru_b),
  .instru (instru_d),
  .funcode (funcode_d)
);

alu_control asset_aluControl(
  .ALUOp (c_ALUOp_1_b),
  .instru (funcode_d),
  .ALUcontrol (ALUcontrol_out)
);

forward asset_forward(
  .ex_instru (instru_d),
  .ex_mem_instru (instru_f), // should be exactly Rd, not wReg
  .mem_wb_instru (instru_h), // same error, fixed
  .c_ex_mem_RegWrite (c_RegWrite_2_b),
  .c_mem_wb_RegWrite (c_RegWrite_3_b),
  .c_data1_src (c_data1_src_w),
  .c_data2_src (c_data2_src_w)
);
```

```

  alu asset_alu(
    .data1 (r_read1_b),
    .read2 (r_read2_b),
    .instru (instru_d),
    .ALUSrc (c_ALUSrc_1_b),
    .ALUcontrol (ALUcontrol_out),
    .ex_mem_fwd (ALUresult_b),
    .mem_wb_fwd (memWriteData_b),
    .c_data1_src (c_data1_src_w),
    .c_data2_src (c_data2_src_w),
    .data2_fwd (rData2_ex_fwd),
    .data2_fwd_old (r_read2_d),
    .zero (zero_a),
    .ALUresult (ALUresult_a)
  );

  pr_ex_mem asset_exmem(
    .clk (clk),
    .Jump_in (c_Jump_1_b),
    .Branch_in (c_Branch_1_b),
    .Bne_in (c_Bne_1_b),
    .MemRead_in (c_MemRead_1_b),
    .MemtoReg_in (c_MemtoReg_1_b),
    .MemWrite_in (c_MemWrite_1_b),
    .RegWrite_in (c_RegWrite_1_b),
    .RegDst_in (c_RegDst_1_b),
    .Jump (c_Jump_2_b),
    .Branch (c_Branch_2_b),
    .Bne (c_Bne_2_b),
    .MemRead (c_MemRead_2_b),
    .MemtoReg (c_MemtoReg_2_b),
    .MemWrite (c_MemWrite_2_b),
    .RegWrite (c_RegWrite_2_b),

    .zero_in (zero_a),
    .ALUresult_in (ALUresult_a),
    .instru_in (instru_d),
    .regData2_in (r_read2_b),
    .zero (zero_b), // no longer necessary
    .ALUresult (ALUresult_b),
    .WriteReg (WriteReg_b),
    .instru (instru_f),
    .regData2 (r_read2_d)
  );

  data_memory asset_dm(
    .clk (clk),
    .addr (ALUresult_b),
    .wData (rData2_ex_fwd), // r_read2_d, "reg.read2 | forward"
    .ALUresult (ALUresult_b),
    .MemWrite (c_MemWrite_2_b),
    .MemRead (c_MemRead_2_b),
    .MemtoReg (c_MemtoReg_2_b),
    .rData (memWriteData_a)
  );

  pr_mem_wb asset_memwb(
    .clk (clk),
    .MemtoReg_in (c_MemtoReg_2_b),
    .RegWrite_in (c_RegWrite_2_b),
    .MemtoReg (c_MemtoReg_3_b),
    .RegWrite (c_RegWrite_3_b),

    .wData_in (memWriteData_a), // data to Reg (W.B.)
    .writeReg_in (writeReg_b),
    .instru_in (instru_f),
    .wData (memWriteData_b),
    .writeReg (writeReg_d),
    .instru (instru_h)
  );

endmodule

```

Driver

This is the module that contains clock divider, ring counter and ssd transformation to help implementation on the FPGA board.

1. Clock divider
   The internal clock of the FPGA board is 500MHz, which is far too large. Hence we will make the clock around 100Hz, which is enough that human eye can not recognize. Thus we need a clock divider that reduce initial frequency by a coefficient of $1/100000$
2. Ring counter
   Because the cathode is commonly used, we need to enable anode in a ring manner to display the 4-bit output, so that human eyes can observe a 4-bit hexdecimal number.
3. Input selection
   We are going to select the signal we wish to display from PC and 32 registers, hence this block is actually a 32*1 MUX. And this requires adding extra wire to read the register content.
4. SSD transformation
   This block transforms binary value into 4 groups of 7 bit output whose value is equal to the original value when displayed.

```
 `timescale 1ns / 1ps
 `include "main.v"

module driver(
  input clk,
  input reset,
  input [7:0] switch,
  output [3:0] A,
  output [6:0] ssd
);

  reg [15:0] data;
  reg clock=0;
  //reg tmp=0;
  wire[15:0] tmp;

  main uut(
    .clk (clock)
  );
  io asset_io(clk, data, A, ssd);

  initial begin
    // clock = 0;
    // tmp = 0;
    uut.asset_pc.out = -4;
  end

  always @(*) begin
    case (switch[7:5])
      3'b000: // normal mode, display reg value
        data = uut.asset_reg.RegData[switch[4:0]][15:0];
      3'b001: // display PC
        data = uut.asset_pc.out[15:0];
      3'b010: // display RegID
        data = switch[4:0];
      3'b011:
        data = tmp;
      default: // undefined
        data = 16'b0101010110101100;
    endcase
  end

  assign tmp[15:12] = uut.asset_pc.normal_next;
  assign tmp[12:8] = uut.asset_ifid.clk;
  assign tmp[7:0] = uut.asset_im.instru;

  always @(posedge reset) begin
    clock = ~clock;
  end

endmodule

module io(
  input clk,
  input [15:0] data,
  output [3:0] A, // anode
  output reg [6:0] ssd // cathod
);

```

```
  wire d500;
  wire [6:0] o1,o2,o3,o4;

  divider500 di500(clk,d500);
  ring_cnt_4 rt(d500,A);

  tssd outssd1(data[3:0],o1); // left-most
  tssd outssd2(data[7:4],o2);
  tssd outssd3(data[11:8],o3);
  tssd outssd4(data[15:12],o4); // right-most

  always @(posedge clk) begin
    case (A)
      4'b1110: ssd = o1;
      4'b1101: ssd = o2;
      4'b1011: ssd = o3;
      4'b0111: ssd = o4;
    endcase
  end
endmodule


module divider500(clock, clk_500);
  parameter MAXN = 200000;
  input clock;
  output clk_500;
  reg [17:0] cnt = 18'b0;
  reg clk_500 = 0;

  always @(posedge clock) begin
    if (cnt == MAXN-1) begin
      clk_500 <= 1;
      cnt <= 18'b0;
    end else begin
      cnt <= cnt + 1;
      clk_500 <= 0;
    end
  end
endmodule


module ring_cnt_4(clk_500, A);
  input clk_500;
  output [3:0] A;
  reg [3:0] A = 4'b1110;

  always @(posedge clk_500) begin
    A[1] <= A[0];
    A[2] <= A[1];
    A[3] <= A[2];
    A[0] <= A[3];
  end
```

```
endmodule

module tssd(number, ssd); // to ssd
  input [3:0] number;
  output [6:0] ssd;
  reg [6:0] ssd;

  always @(*) begin
    case (number)
      0: ssd <= 7'b0000001;
      1: ssd <= 7'b1001111;
      2: ssd <= 7'b0010010;
      3: ssd <= 7'b0000110;
      4: ssd <= 7'b1001100;
      5: ssd <= 7'b0100100;
      6: ssd <= 7'b0100000;
      7: ssd <= 7'b0001111;
      8: ssd <= 7'b0000000;
      9: ssd <= 7'b0000100;
      4'b1010: ssd <= 7'b0001000; // A
      4'b1011: ssd <= 7'b1100000;
      4'b1100: ssd <= 7'b0110001;
      4'b1101: ssd <= 7'b1000010;
      4'b1110: ssd <= 7'b0110000;
      4'b1111: ssd <= 7'b0111000;
    endcase
  end
endmodule
```

Testbench

This program is written to output the registers during different clock cycles. Because it is necessary to **output the registers after the operation is completed**, we deliberately set the clock cycle to be **exactly 1 behind** the clock cycle.

```
module testbench;
  integer currTime;
  reg clk;

  main uut(
    .clk (clk)
  );

  initial begin
    #0
    clk = 0;
    currTime = -10;
    uut.asset_pc.out = -4;

$display("==========================================================");

```

```
    #988 $display("=========================================================");
    #989 $stop;
  end

  always @(posedge clk) begin
    // indicating a posedge clk triggered
    $display("---------------------------------------------------------");
    #1; // wait for writing back
    $display("Time: %d, CLK = %d, PC = 0x%H",currTime, clk, uut.asset_pc.out);
    $display("[$s0] = 0x%H, [$s1] = 0x%H, [$s2] = 0x%H",uut.asset_reg.RegData[16],uut.asset_reg.RegData[17],uut.asset_reg.RegData[18]);
    $display("[$s3] = 0x%H, [$s4] = 0x%H, [$s5] = 0x%H",uut.asset_reg.RegData[19],uut.asset_reg.RegData[20],uut.asset_reg.RegData[21]);
    $display("[$s6] = 0x%H, [$s7] = 0x%H, [$t0] = 0x%H",uut.asset_reg.RegData[22],uut.asset_reg.RegData[23],uut.asset_reg.RegData[8]);
    $display("[$t1] = 0x%H, [$t2] = 0x%H, [$t3] = 0x%H",uut.asset_reg.RegData[9],uut.asset_reg.RegData[10],uut.asset_reg.RegData[11]);
    $display("[$t4] = 0x%H, [$t5] = 0x%H, [$t6] = 0x%H",uut.asset_reg.RegData[12],uut.asset_reg.RegData[13],uut.asset_reg.RegData[14]);
    $display("[$t7] = 0x%H, [$t8] = 0x%H, [$t9] = 0x%H",uut.asset_reg.RegData[15],uut.asset_reg.RegData[24],uut.asset_reg.RegData[25]);
  end

  always #10 begin
    clk = ~clk;
    currTime = currTime + 10;
  end

endmodule
```

FPGA implementation

```
`include "main.v"
module driver(
  input clk,
  input reset,
  input [7:0] switch,
  output [3:0] A,
  output [6:0] ssd
);

  reg [15:0] data;
```

```verilog
  reg clock;
  reg [15:0] tmp2;
  wire [4:0] regDst;
  wire [31:0] regOut;
  wire [31:0] pcOut;

  main uut(
    .clk (clock),
    .syn_reg_dst (regDst),
    .syn_reg_out (regOut),
    .syn_pc (pcOut)
  );

  io asset_io(clk, data, A, ssd);

  initial begin
    clock = 0;
    tmp2 = 16'b0;
  end

  assign regDst = switch[4:0];

  always @(switch) begin
    case (switch[7:5])
      3'b000: // normal mode, display reg value
        data = regOut[15:0];
      3'b001: // display PC
        data = pcOut[15:0];
      3'b010: // display RegID
        data = switch[4:0];
      3'b011: // debug
        data = tmp2;
      default: // undefined
        data = 16'b0101010110101100;
    endcase
  end


  always @(posedge clock) begin
    if (tmp2 < 500) tmp2 <= tmp2 + 1;
  end

  always @(posedge reset) begin
    clock = ~clock;
  end

endmodule

module io(
  input clk,
  input [15:0] data,
  output [3:0] A, // anode
  output reg [6:0] ssd // cathod
);

  wire d500;
  wire [6:0] o1,o2,o3,o4;

  divider500 di500(clk,d500);
  ring_cnt_4 rt(d500,A);

  tssd outssd1(data[3:0],o1); // left-most
  tssd outssd2(data[7:4],o2);
  tssd outssd3(data[11:8],o3);
  tssd outssd4(data[15:12],o4); // right-most


  always @(posedge clk) begin
    case (A)
      4'b1110: ssd = o1;
      4'b1101: ssd = o2;
      4'b1011: ssd = o3;
      4'b0111: ssd = o4;
    endcase
  end
endmodule


module divider500(clock, clk_500);
  parameter MAXN = 200000;
  input clock;
  output clk_500;
  reg [17:0] cnt = 18'b0;
  reg clk_500 = 0;

  always @(posedge clock) begin
    if (cnt == MAXN-1) begin
      clk_500 <= 1;
      cnt <= 18'b0;
    end else begin
      cnt <= cnt + 1;
      clk_500 <= 0;
    end
  end
endmodule


module ring_cnt_4(clk_500, A);
  input clk_500;
  output [3:0] A;
  reg [3:0] A = 4'b1110;

  always @(posedge clk_500) begin
    A[1] <= A[0];
    A[2] <= A[1];
    A[3] <= A[2];
    A[0] <= A[3];
  end
endmodule

module tssd(number, ssd); // to ssd
  input [3:0] number;
  output [6:0] ssd;
  reg [6:0] ssd;

  always @(*) begin
    case (number)
      0: ssd <= 7'b0000001;
      1: ssd <= 7'b1001111;
      2: ssd <= 7'b0010010;
      3: ssd <= 7'b0000110;
      4: ssd <= 7'b1001100;
      5: ssd <= 7'b0100100;
      6: ssd <= 7'b0100000;
      7: ssd <= 7'b0001111;
      8: ssd <= 7'b0000000;
      9: ssd <= 7'b0000100;
      4'b1010: ssd <= 7'b0001000; // A
      4'b1011: ssd <= 7'b1100000;
      4'b1100: ssd <= 7'b0110001;
      4'b1101: ssd <= 7'b1000010;
      4'b1110: ssd <= 7'b0110000;
      4'b1111: ssd <= 7'b0111000;
    endcase
  end
endmodule
```

Simulation Results

Instructions

We use two sets of instructions to test our pipeline processor. The First Set:

```
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001001 00000000 00110111 //addi $t1, $zero, 0x37
00000001 00001001 10000000 00100100 //and $s0, $t0, $t1
00000001 00001001 10000000 00100101 //or $s0, $t0, $t1
10101100 00010000 00000000 00000100 //sw $s0, 4($zero)
10101100 00001000 00000000 00001000 //sw $t0, 8($zero)
00000001 00001001 10001000 00100000 //add $s1, $t0, $t1
00000001 00001001 10010000 00100010 //sub $s2, $t0, $t1
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
```

```
00010010 00110010 00000000 00010010 //beq $s1, $s2, error0
10001100 00010001 00000000 00000100 //lw $s1, 4($zero)
00110010 00110010 00000000 01001000 //andi $s2, $s1, 0x48
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00010010 00110010 00000000 00001111 //beq $s1, $s2, error1
10001100 00010011 00000000 00001000 //lw $s3, 8($zero)
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00010010 00010011 00000000 00001101 //beq $s0, $s3, error2
00000010 01010001 10100000 00101010 //slt $s4, $s2, $s1 (Last)
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
00010010 10000000 00000000 00001111 //beq $s4, $0, EXIT
00000010 00100000 10010000 00100000 //add $s2, $s1, $0
00001000 00000000 00000000 00010111 //j Last
00100000 00001000 00000000 00000000 //addi $t0, $0, 0(error0)
00100000 00001001 00000000 00000000 //addi $t1, $0, 0
00001000 00000000 00000000 00111111 //j EXIT
00100000 00001000 00000000 00000001 //addi $t0, $0, 1(error1)
00100000 00001001 00000000 00000001 //addi $t1, $0, 1
00001000 00000000 00000000 00111111 //j EXIT
00100000 00001000 00000000 00000010 //addi $t0, $0, 2(error2)
00100000 00001001 00000000 00000010 //addi $t1, $0, 2
00001000 00000000 00000000 00111111 //j EXIT
00100000 00001000 00000000 00000011 //addi $t0, $0, 3(error3)
00100000 00001001 00000000 00000011 //addi $t1, $0, 3
00001000 00000000 00000000 00111111 //j EXIT
```

Simulation

The Simulation Results of the first set is shown below:

Simulation results of instructions(1)

```
-----------------------------------------------------------
Time:           0, CLK = 1, PC = 0x00000000
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
-----------------------------------------------------------
Time:          20, CLK = 1, PC = 0x00000004
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
```

```
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          40, CLK = 1, PC = 0x00000008
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          60, CLK = 1, PC = 0x0000000c
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          80, CLK = 1, PC = 0x00000010
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:         100, CLK = 1, PC = 0x00000014
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000037 | WriteReg:  9
Time:         120, CLK = 1, PC = 0x00000018
[$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg: 16
Time:         140, CLK = 1, PC = 0x0000001c
[$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
```

```
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
--------------------------------------------------------
Reg_WriteData: 0x00000037 | WriteReg: 16
Time:          160, CLK = 1, PC = 0x00000020
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
--------------------------------------------------------
Time:          180, CLK = 1, PC = 0x00000024
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
--------------------------------------------------------
Time:          200, CLK = 1, PC = 0x00000028
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
--------------------------------------------------------
Reg_WriteData: 0x00000057 | WriteReg: 17
Time:          220, CLK = 1, PC = 0x0000002c
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
--------------------------------------------------------
Reg_WriteData: 0xffffffe9 | WriteReg: 18
Time:          240, CLK = 1, PC = 0x00000030
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
--------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          260, CLK = 1, PC = 0x00000034
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
```

```
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
-----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          280, CLK = 1, PC = 0x00000038
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
-----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          300, CLK = 1, PC = 0x00000038
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
-----------------------------------------------------------
Time:          320, CLK = 1, PC = 0x0000003c
[$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
-----------------------------------------------------------
Reg_WriteData: 0x00000037 | WriteReg: 17
Time:          340, CLK = 1, PC = 0x00000040
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
-----------------------------------------------------------
Time:          360, CLK = 1, PC = 0x00000044
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffe9
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
-----------------------------------------------------------
Reg_WriteData: 0x00000000 | WriteReg: 18
Time:          380, CLK = 1, PC = 0x00000048
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
```

```
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          400, CLK = 1, PC = 0x0000004c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          420, CLK = 1, PC = 0x00000050
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          440, CLK = 1, PC = 0x00000054
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          460, CLK = 1, PC = 0x00000058
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg: 19
Time:          480, CLK = 1, PC = 0x0000005c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          500, CLK = 1, PC = 0x00000060
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
```

```
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
---------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          520, CLK = 1, PC = 0x00000064
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
---------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          540, CLK = 1, PC = 0x00000068
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
---------------------------------------------------------
Time:          560, CLK = 1, PC = 0x0000006c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
---------------------------------------------------------
Reg_WriteData: 0x00000001 | WriteReg: 20
Time:          580, CLK = 1, PC = 0x00000070
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
---------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          600, CLK = 1, PC = 0x00000074
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
---------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          620, CLK = 1, PC = 0x00000078
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
```

```
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:          640, CLK = 1, PC = 0x0000005c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          660, CLK = 1, PC = 0x00000060
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000037 | WriteReg: 18
Time:          680, CLK = 1, PC = 0x00000064
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          700, CLK = 1, PC = 0x00000068
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          720, CLK = 1, PC = 0x0000006c
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000000 | WriteReg: 20
Time:          740, CLK = 1, PC = 0x00000070
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
```

```
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:         760, CLK = 1, PC = 0x000000ac
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:         780, CLK = 1, PC = 0x000000b0
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Reg_WriteData: 0x00000020 | WriteReg:  8
Time:         800, CLK = 1, PC = 0x000000b4
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:         820, CLK = 1, PC = 0x000000b8
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:         840, CLK = 1, PC = 0x000000bc
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:         860, CLK = 1, PC = 0x000000c0
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
```

```
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          880, CLK = 1, PC = 0x000000c4
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          900, CLK = 1, PC = 0x000000c8
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          920, CLK = 1, PC = 0x000000cc
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          940, CLK = 1, PC = 0x000000d0
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
----------------------------------------------------------
Time:          960, CLK = 1, PC = 0x000000d4
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
==========================================================
----------------------------------------------------------
Time:          980, CLK = 1, PC = 0x000000d8
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

The simulation results of the seconds set of instructions is shown below:

RTL schematic

Conclusion

We can find that the instructions runs quite well on our pipeline processor. All the output signals are the same as the logic suggests. The longer the pipeline implementation runs , the CPI of the processor is closer to 1, which is huge improvement compared with single-stage-processor under the same clock frequency.

Appendix

RTL schematics