

VE370 Project Report

Project 2 - Fall 2020

Gengchen Yang (518370910088)

Jiaying Xu (518021910738)

Qinhang Wu (518370910041)

Yuru Liu (518021910652)

Table of Contents

- Pipelined Processor Design
- FPGA Implementation

VE370 Project 2 Report

VE370 Project 2 Report

Introduction

Pipeline Processor Implementation - Modules

PC

Instruction Memory

next_PC

Register File

ALU

ALU Control

Control

Data Memory

Pipeline Registers

Forwarding Unit

Hazard Detection Unit

Pipeline top module

Driver

Testbench

FPGA implementation

Simulation Results

Instructions

Simulation

RTL schematic

Conclusion

Appendix

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

```
1  module program_counter(  
2      input clk,  
3      input [31:0] bj_next, // the input address; result from next stage  
4      input [31:0] normal_next, // normal next pc (pc+4)  
5      input c_if_flush, // if not asserted, pc=pc+4  
6      input c_PCwrite, // if not asserted, the PC won't move on  
7      output reg [31:0] out // the output address  
8  );  
9  
10     initial begin  
11         out = -4; // NEVER REACHED ADDRESS  
12     end  
13  
14  
15     always @(posedge clk) begin  
16         if (c_PCwrite == 1) begin  
17             if (c_if_flush == 0) begin  
18                 out = normal_next;  
19             end else begin  
20                 out = bj_next;  
21             end  
22         end  
23     end  
24  
25 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

```
1 module instru_memory(  
2     input [31:0] addr,  
3     output reg [5:0] ctr, // [31-26]  
4     output reg [5:0] funcode, // [5-0]  
5 );  
6  
7     parameter SIZE_IM = 128; // size of this memory, by default 128*32  
8     reg [31:0] mem [SIZE_IM-1:0]; // instruction memory  
9  
10    integer n;  
11    initial begin  
12        for(n=0;n<SIZE_IM;n=n+1) begin  
13            mem[n] = 32'b11111100000000000000000000000000;  
14        end  
15        $readmemb("C:\\Users\\William Wu\\Documents\\Mainframe Files\\UMJI-  
SJTU\\1 Academy\\20  
Fall\\VE370\\Project\\p2\\single_cycle\\testcases\\testcase.txt",mem);  
16        instru = 32'b11111100000000000000000000000000;  
17    end  
18  
19    always @(*) begin  
20        if (addr == -4) begin // init  
21            instru = 32'b11111100000000000000000000000000;  
22        end else begin  
23            instru = mem[addr >> 2];  
24        end  
25    end  
26    always @(*)begin  
27        ctr = instru[31:26];  
28        funcode = instru[5:0];  
29    end  
30  
31 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` and `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`

```
1  module next_pc(
2      input [31:0] old, // IF/ID.pc
3      input [31:0] instru, // IF/ID.instruction
4          // [15:0] used for sign-extention
5          // [25:0] used for shift-left-2
6      input Jump, // ID.control
7      input Branch,
8      input Bne,
9      input zero, // it now depends on the RG. comparator.
10     output reg [31:0] next,
11     output reg c_if_flush // IF.Flush
12 );
13
14     // no connection with Hazard-detection
15
16     reg [31:0] sign_ext;
17     reg [31:0] old_alter; // pc+4
18     reg [31:0] jump; // jump addr.
19     reg zero_alter;
20
21     initial begin
22         next = 32'b0;
23     end
24
25     always @(old) begin
26         old_alter = old + 4;
27     end
28
29     always @(zero,Bne) begin
30         zero_alter = zero;
31         if (Bne == 1) begin
32             zero_alter = ! zero_alter;
33         end
34     end
35
36     always @(instru) begin
37         // jump-shift-left
38         jump = {4'b0,instru[25:0],2'b0};
39
40         // sign-extension
41         if (instru[15] == 1'b0) begin
42             sign_ext = {16'b0,instru[15:0]};
43         end else begin
44             sign_ext = {{16{1'b1}},instru[15:0]};
45         end
46         sign_ext = {sign_ext[29:0],2'b0}; // shift left
47     end
48
49     always @(instru or old_alter or jump) begin
50         jump = {old_alter[31:28],jump[27:0]};
```

```

51     end
52
53     always @(*) begin
54         // assign next program counter value
55         if (Jump == 1) begin
56             next = jump;
57             c_if_flush = 1;
58         end else begin
59             if (Branch == 1 & zero_alter == 1) begin
60                 next = old_alter + sign_ext;
61                 c_if_flush = 1;
62             end else begin
63                 next = old_alter;
64                 c_if_flush = 0;
65             end
66         end
67     end
68
69 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]`

```

1  module register(
2      input clk,
3      input [31:0] instru, // the raw 32-bit instruction
4      input RegWrite, // from WB stage!
5      input RegDst,
6      input [31:0] WriteData, // from WB stage
7      input [4:0] writeReg, // from WB stage
8      output reg [31:0] ReadData1,
9      output reg [31:0] ReadData2,
10     output reg reg_zero // comparator result
11 );
12
13     reg [31:0] RegData [31:0]; // register data
14
15     // initialize the register data
16     integer i;
17     initial begin
18         for(i=0;i<32;i=i+1) begin
19             RegData[i] = 32'b0;
20         end
21     end
22
23     always @(*) begin

```

```

24     if(WriteReg==instru[25:21] && RegWrite==1) begin
25         ReadData1 = WriteData;
26     end else begin
27         ReadData1 = RegData[instru[25:21]];
28     end
29
30     if(WriteReg==instru[20:16] && RegWrite==1) begin
31         ReadData2 = WriteData;
32     end else begin
33         ReadData2 = RegData[instru[20:16]];
34     end
35 end
36
37 always @(posedge clk) begin // RegWrite, RegDst, WriteData, instru)
38     if (RegWrite == 1'b1) begin
39         $display("Reg_WriteData: 0x%H | WriteReg: %d",WriteData,
40         WriteReg);
41         RegData[WriteReg] = WriteData;
42     end
43 end
44
45 always @(*) begin
46     if (ReadData1 == ReadData2) begin
47         reg_zero = 1;
48     end else begin
49         reg_zero = 0;
50     end
51 end
52 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`

```

1  module alu(
2      input [31:0] data1,
3      input [31:0] read2, // candidate for data2
4      input [31:0] instru, // candidate for data2; used for sign-extension
5      input ALUSrc,
6      input [3:0] ALUcontrol,
7
8      input [31:0] ex_mem_fwd, // forwarded data from EX/MEM
9      input [31:0] mem_wb_fwd, // forwarded data from MEM/WB
10     input [1:0] c_data1_src,
11     input [1:0] c_data2_src,
12
13     output reg [31:0] data2_fwd, // connect to DM
14     input [31:0] data2_fwd_old,
15     output reg zero,
16     output reg [31:0] ALUresult
17 );
18
19     reg [31:0] data1_fin;
20     reg [31:0] data2_fin;
21
22
23     always @(*) begin
24         case (c_data1_src)
25             2'b00: // from current stage
26                 data1_fin = data1;
27             2'b10: // from EX/MEM
28                 data1_fin = ex_mem_fwd;
29             2'b01: // from from MEM/WB
30                 data1_fin = mem_wb_fwd;
31             default:
32                 ;
33         endcase
34     end
35
36     always @(*) begin
37         if (ALUSrc == 0) begin
38             case (c_data2_src)
39                 2'b00: begin// from current stage
40                     data2_fin = read2;
41                     data2_fwd = data2_fwd_old;
42                 end
43                 2'b10: begin// from EX/MEM
44                     data2_fin = ex_mem_fwd;
45                     data2_fwd = data2_fin;
46                 end

```

```

47     2'b01: begin// from from MEM/WB
48         data2_fin = mem_wb_fwd;
49         data2_fwd = data2_fin;
50     end
51     default:
52         ;
53     endcase
54
55 end else begin
56     // SignExt[Instru[15:0]]
57     if (instru[15] == 1'b0) begin
58         data2_fin = {16'b0,instru[15:0]};
59     end else begin
60         data2_fin = {{16{1'b1}},instru[15:0]};
61     end
62 end
63 end
64
65 always @(*) begin
66     case (ALUcontrol)
67         4'b0000: // AND
68             ALUresult = data1_fin & data2_fin;
69         4'b0001: // OR
70             ALUresult = data1_fin | data2_fin;
71         4'b0010: // ADD
72             ALUresult = data1_fin + data2_fin;
73         4'b0110: // SUB
74             ALUresult = data1_fin - data2_fin;
75         4'b0111: // SLT
76             ALUresult = (data1_fin < data2_fin) ? 1 : 0;
77         4'b1100: // NOR
78             ALUresult = data1_fin |~ data2_fin;
79     default:
80         ;
81     endcase
82     if (ALUresult == 0) begin
83         zero = 1;
84     end else begin
85         zero = 0;
86     end
87 end
88
89 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 corresponding `ALUcontrol=4'b0000` (AND)
3. `funct` is utilized to further distinguish different R-format instructions


```

1  module alu_control(
2      input [1:0] ALUOp,
3      input [5:0] instru,
4      output reg [3:0] ALUcontrol
5  );
6
7      always @(*) begin
8          case (ALUOp)
9              2'b00:
10                 ALUcontrol = 4'b0010;
11              2'b01:
12                 ALUcontrol = 4'b0110;
13              2'b10: begin
14                  case (instru)
15                      6'b100000: // add
16                         ALUcontrol = 4'b0010;
17                      6'b100010: // sub
18                         ALUcontrol = 4'b0110;
19                      6'b100100: // and
20                         ALUcontrol = 4'b0000;
21                      6'b100101: // or
22                         ALUcontrol = 4'b0001;
23                      6'b101010: // slt
24                         ALUcontrol = 4'b0111;
25                      default:
26                          ;
27                  endcase
28              end
29              2'b11:
30                 ALUcontrol = 4'b0000;
31              default:
32                  ;
33          endcase
34      end
35
36  endmodule

```

Control

- 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
- If `c_clearControl==0`, control signals are assigned according to `instru`; otherwise, control signals remain 0
- For instructions of different types, different control signals are assigned
- 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`

```

1  module control(
2      input [31:0] instru,
3      input c_clearControl,
4      output reg RegDst,
5      output reg Jump,
6      output reg Branch,
7      output reg Bne, // 1 indicates bne
8      output reg MemRead,
9      output reg MemtoReg,
10     output reg [1:0] ALUOp,
11     output reg MemWrite,
12     output reg ALUSrc,
13     output reg RegWrite
14
15     initial begin
16         RegDst = 0;
17         Jump = 0;
18         Branch = 0;
19         MemRead = 0;
20         MemtoReg = 0;
21         ALUOp = 2'b00;
22         MemWrite = 0;
23         ALUSrc = 0;
24         RegWrite = 0;
25     end
26
27     always @(*) begin
28         if (c_clearControl == 0) begin
29             case (instru[31:26])
30                 6'b000000: begin// ARITHMETIC
31                     RegDst = 1;
32                     ALUSrc = 0;
33                     MemtoReg = 0;
34                     RegWrite = 1;
35                     MemRead = 0;
36                     MemWrite = 0;
37                     Branch = 0;
38                     Bne = 0;
39                     ALUOp = 2'b10;
40                     Jump = 0;
41                 end
42                 6'b001000: begin// addi
43                     RegDst = 0;
44                     ALUSrc = 1;
45                     MemtoReg = 0;
46                     RegWrite = 1;
47                     MemRead = 0;
48                     MemWrite = 0;
49                     Branch = 0;

```

```

50         Bne = 0;
51         ALUOp = 2'b00;
52         Jump = 0;
53     end
54     6'b001100: begin // andi
55         RegDst = 0;
56         ALUSrc = 1;
57         MemtoReg = 0;
58         RegWrite = 1;
59         MemRead = 0;
60         MemWrite = 0;
61         Branch = 0;
62         Bne = 0;
63         ALUOp = 2'b11;
64         Jump = 0;
65     end
66     6'b100011: begin // lw
67         RegDst = 0;
68         ALUSrc = 1;
69         MemtoReg = 1;
70         RegWrite = 1;
71         MemRead = 1;
72         MemWrite = 0;
73         Branch = 0;
74         Bne = 0;
75         ALUOp = 2'b00;
76         Jump = 0;
77     end
78     6'b101011: begin // sw
79         RegDst = 0; // x
80         ALUSrc = 1;
81         MemtoReg = 0; // x
82         RegWrite = 0;
83         MemRead = 0;
84         MemWrite = 1;
85         Branch = 0;
86         Bne = 0;
87         ALUOp = 2'b00;
88         Jump = 0;
89     end
90     6'b000100: begin // beq
91         RegDst = 0; // x
92         ALUSrc = 0;
93         MemtoReg = 0; // x
94         RegWrite = 0;
95         MemRead = 0;
96         MemWrite = 0;
97         Branch = 1;
98         Bne = 0;
99         ALUOp = 2'b01;
100        Jump = 0;
101    end

```

```

102         6'b000101: begin // bne
103             RegDst = 0; // x
104             ALUSrc = 0;
105             MemtoReg = 0; // x
106             RegWrite = 0;
107             MemRead = 0;
108             MemWrite = 0;
109             Branch = 1;
110             Bne = 1;
111             ALUOp = 2'b01;
112             Jump = 0;
113         end
114         6'b000010: begin // j
115             RegDst = 0; // x
116             ALUSrc = 0;
117             MemtoReg = 0; // x
118             RegWrite = 0;
119             MemRead = 0;
120             MemWrite = 0;
121             Branch = 0;
122             Bne = 0;
123             ALUOp = 2'b01;
124             Jump = 1;
125         end
126         default: begin
127             RegDst = 0; // x
128             ALUSrc = 0;
129             MemtoReg = 0; // x
130             RegWrite = 0;
131             MemRead = 0;
132             MemWrite = 0;
133             Branch = 0;
134             Bne = 0;
135             ALUOp = 2'b00;
136             Jump = 0;
137         end
138     endcase
139 end else begin
140     RegDst = 0;
141     Jump = 0;
142     Branch = 0;
143     MemRead = 0;
144     MemtoReg = 0;
145     ALUOp = 2'b00;
146     MemWrite = 0;
147     ALUSrc = 0;
148     RegWrite = 0;
149 end
150 end
151
152 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

```
1  module data_memory(  
2      input clk,  
3      input [31:0] addr,  
4      input [31:0] wData,  
5      input [31:0] ALUresult,  
6      input MemWrite,  
7      input MemRead,  
8      input MemtoReg,  
9      output reg [31:0] rData  
10 );  
11  
12 parameter SIZE_DM = 128; // size of this memory, by default 128*32  
13 reg [31:0] mem [SIZE_DM-1:0]; // instruction memory  
14  
15 // initially set default data to 0  
16 integer i;  
17 initial begin  
18     for(i=0; i<SIZE_DM-1; i=i+1) begin  
19         mem[i] = 32'b0;  
20     end  
21 end  
22  
23 always @(addr or MemRead or MemtoReg or ALUresult) begin  
24     if (MemRead == 1) begin  
25         if (MemtoReg == 1) begin  
26             rData = mem[addr];  
27         end else begin  
28             rData = ALUresult; // x ?  
29         end  
30     end else begin  
31         rData = ALUresult;  
32     end  
33 end  
34  
35 always @(posedge clk) begin // MemWrite, wData, addr  
36     if (MemWrite == 1) begin  
37         mem[addr] = wData;  
38     end  
39 end  
40  
41 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

```
1  module pr_if_id(  
2      input clk,  
3      input c_IFIDWrite,  
4      input c_if_flush,  
5      input [5:0] ctr_in,  
6      input [5:0] funcode_in,  
7      input [31:0] instru_in,  
8      input [31:0] nextpc_in, // next pc  
9      output reg [5:0] ctr, // [31-26]  
10     output reg [5:0] funcode, // [5-0]  
11     output reg [31:0] instru, // [31-0]  
12     output reg [31:0] nextpc, // to next_pc.v  
13     output reg [31:0] normal_nextpc // pc+4, passing to pc  
14 );  
15  
16 initial begin  
17     ctr = 6'b111111;  
18     funcode = 6'b000000;  
19     instru = 32'b111111000000000000000000000000;  
20     nextpc = 32'b0;  
21 end  
22  
23 always @(posedge clk) begin  
24     if (c_IFIDWrite == 1) begin  
25         if (c_if_flush == 0) begin  
26             ctr = ctr_in;  
27             funcode = funcode_in;  
28             instru = instru_in;  
29             nextpc = nextpc_in;  
30         end else begin  
31             ctr = 6'b111111;  
32             funcode = 6'b000000;
```

```

33     instru = 32'b11111000000000000000000000000000;
34     nextpc = 32'b0;
35     end
36 end
37 end
38
39 always @(nextpc_in) begin
40     normal_nextpc = nextpc_in + 4;
41 end
42
43 endmodule

```

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

```

1  module pr_id_ex(
2      input clk,
3      // this can be modified to one 11-bit control signal, but I'm running
      out of time...
4      input RegDst_in, // EX
5      input Jump_in, // MEM
6      input Branch_in, // MEM
7      input Bne_in, // MEM
8      input MemRead_in, // MEM
9      input MemtoReg_in, // WB
10     input [1:0] ALUOp_in, // EX
11     input MemWrite_in, // MEM
12     input ALUSrc_in, // EX
13     input RegWrite_in, // WB
14     output reg RegDst,
15     output reg Jump,
16     output reg Branch,
17     output reg Bne, // 1 indicates bne
18     output reg MemRead,
19     output reg MemtoReg,
20     output reg [1:0] ALUOp,
21     output reg MemWrite,
22     output reg ALUSrc,
23     output reg RegWrite,
24
25     input [31:0] nextPc_in, // from pc
26     output reg [31:0] nextPc,
27     input [31:0] ReadData1_in, // from reg
28     input [31:0] ReadData2_in,
29     input [5:0] funcode_in,
30     output reg [31:0] ReadData1,
31     output reg [31:0] ReadData2,
32     output reg [5:0] funcode,
33     input [31:0] instru_in, // raw instruction from IF/ID-reg
34     output reg [31:0] instru
35 );
36

```

```

37  initial begin
38      RegDst = 0;
39      Jump = 0;
40      Branch = 0;
41      Bne = 0;
42      MemRead = 0;
43      MemtoReg = 0;
44      ALUOp = 2'b00;
45      MemWrite = 0;
46      ALUSrc = 0;
47      RegWrite = 0;
48  end
49
50  always @(posedge clk) begin
51      RegDst = RegDst_in;
52      Jump = Jump_in;
53      Branch = Branch_in;
54      Bne = Bne_in;
55      MemRead = MemRead_in;
56      MemtoReg = MemtoReg_in;
57      ALUOp = ALUOp_in;
58      MemWrite = MemWrite_in;
59      ALUSrc = ALUSrc_in;
60      RegWrite = RegWrite_in;
61
62      nextPc = nextPc_in;
63      ReadData1 = ReadData1_in;
64      ReadData2 = ReadData2_in;
65      funcode = funcode_in;
66      instru = instru_in;
67  end
68
69  endmodule

```

For EX/MEM registeer:

```

1  module pr_ex_mem(
2      input clk,
3      input Jump_in, // MEM
4      input Branch_in, // MEM
5      input Bne_in, // MEM
6      input MemRead_in, // MEM
7      input MemtoReg_in, // WB
8      input MemWrite_in, // MEM
9      input RegWrite_in, // WB
10     input RegDst_in, // EX, only used here
11     output reg Jump,
12     output reg Branch,
13     output reg Bne,
14     output reg MemRead,
15     output reg MemtoReg,
16     output reg MemWrite,

```



```

17     output reg RegWrite,
18
19     input zero_in,
20     input [31:0] ALUresult_in,
21     input [31:0] instru_in, // receive instru and transf to WriteReg (w.b.
    to Reg)
22     input [31:0] regData2_in,
23     output reg zero,
24     output reg [31:0] ALUresult,
25     output reg [4:0] WriteReg,
26     output reg [31:0] instru,
27     output reg [31:0] regData2
28 );
29
30 // TODO: add support for EX.Flush
31
32 initial begin
33     Jump = 0;
34     Branch = 0;
35     Bne = 0;
36     MemRead = 0;
37     MemtoReg = 0;
38     MemWrite = 0;
39     RegWrite = 0;
40 end
41
42 always @(posedge clk) begin
43     Jump = Jump_in;
44     Branch = Branch_in;
45     Bne = Bne_in;
46     MemRead = MemRead_in;
47     MemtoReg = MemtoReg_in;
48     MemWrite = MemWrite_in;
49     RegWrite = RegWrite_in;
50     instru = instru_in;
51
52     zero = zero_in;
53     ALUresult = ALUresult_in;
54     regData2 = regData2_in;
55     if (RegDst_in == 1'b0) begin
56         WriteReg = instru_in[20:16];
57     end else begin
58         WriteReg = instru_in[15:11];
59     end
60 end
61
62 endmodule

```

For MEM/WB register:

```

1 module pr_mem_wb(
2     input clk,

```

```

3   input MemtoReg_in, // WB
4   input RegWrite_in, // WB
5   output reg MemtoReg,
6   output reg RegWrite,
7
8   input [31:0] wData_in,
9   input [4:0] writeReg_in,
10  input [31:0] instru_in,
11  output reg [31:0] wData,
12  output reg [4:0] writeReg,
13  output reg [31:0] instru
14 );
15
16 initial begin
17     MemtoReg = 0;
18     RegWrite = 0;
19 end
20
21 always @(posedge clk) begin
22     MemtoReg = MemtoReg_in;
23     RegWrite = RegWrite_in;
24
25     wData = wData_in;
26     writeReg = writeReg_in;
27     instru = instru_in;
28 end
29
30 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

```

1 // forwarding unit, in stage EX
2
3 module forward(
4     input [31:0] ex_instru, // ID/EX.instru
5     input [31:0] ex_mem_instru, // EX/MEM.WriteReg
6     input [31:0] mem_wb_instru, // MEM/WB.WriteReg
7     input c_ex_mem_RegWrite, // EX/MEM.RegWrite
8     input c_mem_wb_RegWrite, // MEM/WB.RegWrite
9     output reg [1:0] c_data1_src, // ALU.data1.src (A)
10    output reg [1:0] c_data2_src // ALU.data2.src (B)

```

```

11 );
12 // Note: ex_instru[25:21] == ID/EX.Rs
13 //       ex_instru[20:16] == ID/EX.Rt
14 reg [4:0] ex_mem_wReg, mem_wb_wReg; // exactly Rd
15
16 // if I-type, use Rt; if R-type, use Rd
17 always @(*) begin
18     if(ex_mem_instru[31:26] == 0) begin
19         ex_mem_wReg = ex_mem_instru[15:11];
20     end else begin
21         ex_mem_wReg = ex_mem_instru[20:16]; // I
22     end
23     if(mem_wb_instru[31:26] == 0) begin
24         mem_wb_wReg = mem_wb_instru[15:11];
25     end else begin
26         mem_wb_wReg = mem_wb_instru[20:16]; // I
27     end
28 end
29
30 always @(*) begin
31     if(c_ex_mem_RegWrite==1 & (ex_mem_wReg != 0) & (ex_mem_wReg ==
ex_instru[25:21])) begin
32         c_data1_src = 2'b10; // from EX/MEM
33     end else if (c_mem_wb_RegWrite==1 & (mem_wb_wReg != 0) &
(mem_wb_wReg == ex_instru[25:21]) &
34         !(c_ex_mem_RegWrite==1 & (ex_mem_wReg != 0) & (ex_mem_wReg ==
ex_instru[25:21]))) begin
35         c_data1_src = 2'b01; // from from MEM/WB
36     end else begin
37         c_data1_src = 2'b00; // from current stage
38     end
39 end
40
41 always @(*) begin
42     if(c_ex_mem_RegWrite==1 & (ex_mem_wReg != 0) & (ex_mem_wReg ==
ex_instru[20:16])) begin
43         c_data2_src = 2'b10; // from EX/MEM
44     end else if (c_mem_wb_RegWrite==1 & (mem_wb_wReg != 0) &
(mem_wb_wReg == ex_instru[20:16]) &
45         !(c_ex_mem_RegWrite==1 & (ex_mem_wReg != 0) & (ex_mem_wReg ==
ex_instru[20:16]))) begin
46         c_data2_src = 2'b01; // from from MEM/WB
47     end else begin
48         c_data2_src = 2'b00; // from current stage
49     end
50 end
51
52 endmodule
53
54 module Forwarding_bonus(
55     input [4:0]
    IF_ID_RS, IF_ID_Rt, ID_EX_RS, ID_EX_Rt, EX_MEM_Rd, MEM_WB_Rd,

```

```

56     // note: MEM_WB_Rd is destination of MEM/WB register, for lw &
    addi: rt, add: rd
57     input  EX_MEM_RegWrite, MEM_WB_RegWrite, ID_beq, ID_bne,
58     output reg [1:0] FullFwdA, FullFwdB,
59     output reg BranFwdA, BranFwdB
60 );
61
62 initial begin
63     FullFwdA=2'b00;
64     FullFwdB=2'b00;
65     BranFwdA=1'b0;
66     BranFwdB=1'b0;
67 end
68
69 always @(*) begin // FullFwdA
70     if ((ID_EX_Rs == EX_MEM_Rd) && EX_MEM_RegWrite && (EX_MEM_Rd!=5'b0))
        // R-format
71         FullFwdA=2'b10;
72
73     else if ((ID_EX_Rs == MEM_WB_Rd) && MEM_WB_RegWrite &&
        (MEM_WB_Rd!=5'b0)) // lw & R-format
74         FullFwdA=2'b01;
75
76     else
77         FullFwdA=2'b00;
78 end
79
80 always @(*) begin // FullFwdB
81     if ((ID_EX_Rt == EX_MEM_Rd) && EX_MEM_RegWrite && (EX_MEM_Rd!=5'b0))
82         FullFwdB=2'b10;
83
84     else if ((ID_EX_Rt == MEM_WB_Rd) && MEM_WB_RegWrite &&
        (MEM_WB_Rd!=5'b0))
85         FullFwdB=2'b01;
86
87     else
88         FullFwdB=2'b00;
89
90 end
91
92 always @(*) begin //BranFwdA
93     if((ID_beq || ID_bne) && (IF_ID_Rs == EX_MEM_Rd) &&
        (EX_MEM_Rd!=5'b0) && EX_MEM_RegWrite)
94         BranFwdA=1'b1;
95
96     else
97         BranFwdA=1'b0;
98
99 end
100
101
102 always @(*) begin //BranFwdB

```

```

103     if((ID_beq || ID_bne) && (IF_ID_Rt == EX_MEM_Rd) &&
        (EX_MEM_Rd!=5'b0) && EX_MEM_RegWrite)
104         BranFwdB=1'b1;
105
106     else
107         BranFwdB=1'b0;
108
109 end
110
111 endmodule

```

Hazard Detection Unit

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

```

1  module hazard_det(
2      input id_ex_memRead, // ID/EX.MemRead
3      input [31:0] if_id_instru, // IF/ID.instru
4      input [31:0] id_ex_instru, // ID/EX.instru
5      output reg c_PCWrite, // -> PC
6      output reg c_IFIDWrite, // -> IF/ID pipelined reg
7      output reg c_clearControl // -> control (ID/EX.Flush)
8      // FIXME: bool convention
9  );
10
11
12  initial begin
13      c_PCWrite = 1;
14      c_IFIDWrite = 1;
15      c_clearControl = 0;
16  end
17
18  always @(*) begin
19      if (id_ex_memRead==1 && ((id_ex_instru[20:16] ==
20  if_id_instru[25:21]) || (id_ex_instru[20:16] == if_id_instru[20:16])))
21  begin
22      c_PCWrite = 0; // if PCWrite==0, don't write in new instruction,
23  IM decode the current instruction again
24      c_IFIDWrite = 0; // if IF_ID_Write==0, IF/ID register keeps the
25  current instruction
26      c_clearControl = 1; // if ID_EX_Flush=1, all control signals in
27  ID/EX are 0

```

```

23     end else begin
24         c_PCWrite = 1;
25         c_IFIDWrite = 1;
26         c_clearControl = 0;
27     end
28 end
29
30 endmodule
31
32
33 module Hazard_bonus( // TODO: [bonus] consider control hazard: branch
34     // created by lyr
35     input [4:0] IF_ID_RS, IF_ID_Rt, ID_EX_Rt, EX_MEM_Rt, ID_EX_Rd,
36     input
37     ID_EX_MemRead, EX_MEM_MemRead, ID_beq, ID_bne, ID_EX_RegWrite, ID_jump, ID_equal, ID_EX_RegWrite,
38     output PCWrite, IF_ID_Write, ID_EX_Flush, IF_Flush
39 );
40 wire PCHold; // if PCHold==1, hold PC and IF/ID
41
42 assign PCHold = ( (ID_EX_MemRead) && (ID_EX_Rt == IF_ID_RS || ID_EX_Rt
43 == IF_ID_Rt) ) // lw hazard
44 || ( (ID_beq || ID_bne) && (ID_EX_MemRead) && (ID_EX_Rt
45 == IF_ID_RS || ID_EX_Rt == IF_ID_Rt) ) // lw followed by branch
46 || ( (ID_beq || ID_bne) && (EX_MEM_MemRead) &&
47 (EX_MEM_Rt == IF_ID_RS || EX_MEM_Rt == IF_ID_Rt) ) // lw followed by nop
48 and then branch
49 || ( (ID_beq || ID_bne) && (ID_EX_RegWrite) && (ID_EX_Rd
50 != 5'b0) && (ID_EX_Rd == IF_ID_RS || ID_EX_Rd == IF_ID_Rt) ) // R-format
51 followed by branch
52 || ( (ID_beq || ID_bne) && (ID_EX_RegWrite) && (ID_EX_Rd
53 == 5'b0) && (ID_EX_Rt == IF_ID_RS || ID_EX_Rt == IF_ID_Rt) ); // addi
54 followed by branch
55
56 // note we leave out the case that R-format followed by a nop then a
57 branch, because that is solved by forwarding path
58 assign PCWrite=~PCHold; // if PCWrite==0, don't write in new
59 instruction, IM decode the current instruction again
60
61 assign IF_ID_Write=~PCHold; // if IF_ID_Write==0, IF/ID register keeps
62 the current instruction
63
64 assign ID_EX_Flush=PCHold; // if ID_EX_Flush=1, all control signals in
65 ID/EX are 0 (implemented in ID/EX register later)
66
67 assign IF_Flush = (PCHold==0) && ( (ID_jump) || (ID_beq && ID_equal) ||
68 (ID_bne && ID_equal));
69
70 endmodule
71
72

```

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.

```
1  timescale 1ns / 1ps
2  `include "program_counter.v"
3  `include "next_pc.v"
4  `include "instru_memory.v"
5  `include "pr_if_id.v"
6  `include "register.v"
7  `include "control.v"
8  `include "hazard_det.v"
9  `include "pr_id_ex.v"
10 `include "alu.v"
11 `include "alu_control.v"
12 `include "forward.v"
13 `include "pr_ex_mem.v"
14 `include "data_memory.v"
15 `include "pr_mem_wb.v"
16
17 module main(
18     input clk // clock signal for PC, RD, PRs
19 );
20
21     wire [31:0] pc_in;
22     wire [31:0] normal_next_pc;
23
24     wire [5:0]   ctr_a,
25                 ctr_b,
26                 funcode_a,
27                 funcode_b,
28                 funcode_d;
29     wire [31:0] instru_a,
30                 instru_b,
31                 nextpc_a,
32                 nextpc_b;
33
34     wire c_RegDst_1_a, c_Jump_1_a, c_Branch_1_a, c_Bne_1_a,
35         c_MemRead_1_a, c_MemtoReg_1_a, c_MemWrite_1_a,
36         c_ALUSrc_1_a, c_RegWrite_1_a;
37     wire [1:0] c_ALUOp_1_a;
38     wire c_RegDst_1_b, c_Jump_1_b, c_Branch_1_b, c_Bne_1_b,
39         c_MemRead_1_b, c_MemtoReg_1_b, c_MemWrite_1_b,
40         c_ALUSrc_1_b, c_RegWrite_1_b;
41     wire [1:0] c_ALUOp_1_b;
42     wire [31:0] nextpc_d;
43     wire [31:0] r_read1_a,
44                 r_read1_b,
45                 r_read2_a,
```

```

46         r_read2_b,r_read2_d;
47     wire [31:0] instru_d,instru_f,instru_h;
48
49     wire c_if_flush;
50
51     wire c_PCWrite_w;
52     wire c_IFIDWrite_w;
53     wire c_clearControl_w;
54
55     wire [3:0] ALUcontrol_out;
56
57     wire [1:0] c_data1_src_w; // forward
58     wire [1:0] c_data2_src_w;
59
60     wire [31:0] rData2_ex_fwd;
61
62     wire c_Jump_2_b,c_Branch_2_b,c_Bne_2_b,c_MemRead_2_b,
63         c_MemtoReg_2_b,c_MemWrite_2_b,c_RegWrite_2_b;
64     wire zero_a,zero_b,zero_reg;
65     wire [31:0] ALUresult_a, ALUresult_b;
66     wire [4:0] writeReg_b;
67
68     wire c_MemtoReg_3_b, c_RegWrite_3_b;
69     wire [4:0] writeReg_d;
70
71     wire [31:0] memWriteData_a,
72         memWriteData_b;
73
74     program_counter asset_pc(
75         .clk (clk),
76         .bj_next (pc_in),
77         .normal_next (normal_next_pc),
78         .c_if_flush (c_if_flush),
79         .c_PCWrite (c_PCWrite_w),
80         .out (nextpc_a)
81     );
82
83     instru_memory asset_im(
84         .addr (nextpc_a),
85         .ctr (ctr_a),
86         .funcode (funcode_a),
87         .instru (instru_a)
88     );
89
90     pr_if_id asset_ifid(
91         .clk (clk),
92         .c_IFIDWrite (c_IFIDWrite_w),
93         .c_if_flush (c_if_flush),
94         .ctr_in (ctr_a),
95         .funcode_in (funcode_a),
96         .instru_in (instru_a),
97         .nextpc_in (nextpc_a),

```



```

98     .ctr (ctr_b),
99     .funcode (funcode_b),
100    .instru (instru_b),
101    .nextpc (nextpc_b), // pc instead of pc+4
102    .normal_nextpc (normal_next_pc)
103 );
104
105 next_pc asset_nextPc(
106     .old (nextpc_b),
107     .instru (instru_b),
108     .Jump (c_Jump_1_a),
109     .Branch (c_Branch_1_a),
110     .Bne (c_Bne_1_a),
111     .zero (zero_reg),
112     .next (pc_in),
113     .c_if_flush (c_if_flush)
114 );
115
116 hazard_det asset_hDet(
117     .id_ex_memRead (c_MemRead_1_b),
118     .if_id_instru (instru_b),
119     .id_ex_instru (instru_d), // TODO
120     .c_PCWrite (c_PCWrite_w),
121     .c_IFIDWrite (c_IFIDWrite_w),
122     .c_clearControl (c_clearControl_w)
123 );
124
125 register asset_reg(
126     .clk (clk),
127     .instru (instru_b),
128     .RegWrite (c_RegWrite_3_b), // from WB stage
129     .RegDst (c_RegDst_1_a),
130     .writeData (memWriteData_b),
131     .writeReg (writeReg_d),
132     .ReadData1 (r_read1_a),
133     .ReadData2 (r_read2_a),
134     .reg_zero (zero_reg)
135 );
136
137 control asset_control(
138     .instru (instru_b),
139     .c_clearControl (c_clearControl_w),
140     .RegDst (c_RegDst_1_a),
141     .Jump (c_Jump_1_a),
142     .Branch (c_Branch_1_a),
143     .Bne (c_Bne_1_a),
144     .MemRead (c_MemRead_1_a),
145     .MemtoReg (c_MemtoReg_1_a),
146     .ALUOp (c_ALUOp_1_a),
147     .MemWrite (c_MemWrite_1_a),
148     .ALUSrc (c_ALUSrc_1_a),
149     .RegWrite (c_RegWrite_1_a)

```

```

150 );
151
152 pr_id_ex asset_idxex(
153     .clk (clk),
154     .RegDst_in (c_RegDst_1_a),
155     .Jump_in (c_Jump_1_a),
156     .Branch_in (c_Branch_1_a),
157     .Bne_in (c_Bne_1_a),
158     .MemRead_in (c_MemRead_1_a),
159     .MemtoReg_in (c_MemtoReg_1_a),
160     .ALUOp_in (c_ALUOp_1_a),
161     .MemWrite_in (c_MemWrite_1_a),
162     .ALUSrc_in (c_ALUSrc_1_a),
163     .RegWrite_in (c_RegWrite_1_a),
164     .RegDst (c_RegDst_1_b),
165     .Jump (c_Jump_1_b),
166     .Branch (c_Branch_1_b),
167     .Bne (c_Bne_1_b),
168     .MemRead (c_MemRead_1_b),
169     .MemtoReg (c_MemtoReg_1_b),
170     .ALUOp (c_ALUOp_1_b),
171     .MemWrite (c_MemWrite_1_b),
172     .ALUSrc (c_ALUSrc_1_b),
173     .RegWrite (c_RegWrite_1_b),
174
175     .nextPc_in (nextpc_b),
176     .nextPc (nextpc_d),
177     .ReadData1_in (r_read1_a),
178     .ReadData2_in (r_read2_a),
179     .funcode_in (funcode_b),
180     .ReadData1 (r_read1_b),
181     .ReadData2 (r_read2_b),
182     .instru_in (instru_b),
183     .instru (instru_d),
184     .funcode (funcode_d)
185 );
186
187 alu_control asset_aluControl(
188     .ALUOp (c_ALUOp_1_b),
189     .instru (funcode_d),
190     .ALUcontrol (ALUcontrol_out)
191 );
192
193 forward asset_forward(
194     .ex_instru (instru_d),
195     .ex_mem_instru (instru_f), // should be exactly Rd, not wReg
196     .mem_wb_instru (instru_h), // same error, fixed
197     .c_ex_mem_RegWrite (c_RegWrite_2_b),
198     .c_mem_wb_RegWrite (c_RegWrite_3_b),
199     .c_data1_src (c_data1_src_w),
200     .c_data2_src (c_data2_src_w)
201 );

```

```

202
203     alu asset_alu(
204         .data1 (r_read1_b),
205         .read2 (r_read2_b),
206         .instru (instru_d),
207         .ALUSrc (c_ALUSrc_1_b),
208         .ALUcontrol (ALUcontrol_out),
209         .ex_mem_fwd (ALUresult_b),
210         .mem_wb_fwd (memWriteData_b),
211         .c_data1_src (c_data1_src_w),
212         .c_data2_src (c_data2_src_w),
213         .data2_fwd (rData2_ex_fwd),
214         .data2_fwd_old (r_read2_d),
215         .zero (zero_a),
216         .ALUresult (ALUresult_a)
217     );
218
219     pr_ex_mem asset_exmem(
220         .clk (clk),
221         .Jump_in (c_Jump_1_b),
222         .Branch_in (c_Branch_1_b),
223         .Bne_in (c_Bne_1_b),
224         .MemRead_in (c_MemRead_1_b),
225         .MemtoReg_in (c_MemtoReg_1_b),
226         .MemWrite_in (c_MemWrite_1_b),
227         .RegWrite_in (c_RegWrite_1_b),
228         .RegDst_in (c_RegDst_1_b),
229         .Jump (c_Jump_2_b),
230         .Branch (c_Branch_2_b),
231         .Bne (c_Bne_2_b),
232         .MemRead (c_MemRead_2_b),
233         .MemtoReg (c_MemtoReg_2_b),
234         .MemWrite (c_MemWrite_2_b),
235         .RegWrite (c_RegWrite_2_b),
236
237         .zero_in (zero_a),
238         .ALUresult_in (ALUresult_a),
239         .instru_in (instru_d),
240         .regData2_in (r_read2_b),
241         .zero (zero_b), // no longer necessary
242         .ALUresult (ALUresult_b),
243         .writeReg (writeReg_b),
244         .instru (instru_f),
245         .regData2 (r_read2_d)
246     );
247
248     data_memory asset_dm(
249         .clk (clk),
250         .addr (ALUresult_b),
251         .wData (rData2_ex_fwd), // r_read2_d, "reg.read2 | forward"
252         .ALUresult (ALUresult_b),
253         .MemWrite (c_MemWrite_2_b),

```

```

254     .MemRead (c_MemRead_2_b),
255     .MemtoReg (c_MemtoReg_2_b),
256     .rData (memWriteData_a)
257 );
258
259 pr_mem_wb asset_memwb(
260     .clk (clk),
261     .MemtoReg_in (c_MemtoReg_2_b),
262     .RegWrite_in (c_RegWrite_2_b),
263     .MemtoReg (c_MemtoReg_3_b),
264     .RegWrite (c_RegWrite_3_b),
265
266     .wData_in (memWriteData_a), // data to Reg (W.B.)
267     .writeReg_in (writeReg_b),
268     .instru_in (instru_f),
269     .wData (memWriteData_b),
270     .writeReg (writeReg_d),
271     .instru (instru_h)
272 );
273
274 endmodule
275

```

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 hexadecimal 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.

```

1  `timescale 1ns / 1ps
2  `include "main.v"
3
4  module driver(
5      input clk,
6      input reset,
7      input [7:0] switch,

```

```

8     output [3:0] A,
9     output [6:0] ssd
10 );
11
12     reg [15:0] data;
13     reg clock=0;
14     //reg tmp=0;
15     wire[15:0] tmp;
16
17     main uut(
18         .clk (clock)
19     );
20     io asset_io(clk, data, A, ssd);
21
22     initial begin
23         // clock = 0;
24         // tmp = 0;
25         uut.asset_pc.out = -4;
26     end
27
28     always @(*) begin
29         case (switch[7:5])
30             3'b000: // normal mode, display reg value
31                 data = uut.asset_reg.RegData[switch[4:0]][15:0];
32             3'b001: // display PC
33                 data = uut.asset_pc.out[15:0];
34             3'b010: // display RegID
35                 data = switch[4:0];
36             3'b011:
37                 data = tmp;
38             default: // undefined
39                 data = 16'b0101010110101100;
40         endcase
41     end
42
43     assign tmp[15:12] = uut.asset_pc.normal_next;
44     assign tmp[12:8] = uut.asset_ifid.clk;
45     assign tmp[7:0] = uut.asset_im.instru;
46
47     always @(posedge reset) begin
48         clock = ~clock;
49     end
50
51 endmodule
52
53 module io(
54     input clk,
55     input [15:0] data,
56     output [3:0] A, // anode
57     output reg [6:0] ssd // cathod
58 );
59

```

```

60    wire d500;
61    wire [6:0] o1,o2,o3,o4;
62
63    divider500 di500(clk,d500);
64    ring_cnt_4 rt(d500,A);
65
66    tssd outssd1(data[3:0],o1); // left-most
67    tssd outssd2(data[7:4],o2);
68    tssd outssd3(data[11:8],o3);
69    tssd outssd4(data[15:12],o4); // right-most
70
71    always @(posedge clk) begin
72        case (A)
73            4'b1110: ssd = o1;
74            4'b1101: ssd = o2;
75            4'b1011: ssd = o3;
76            4'b0111: ssd = o4;
77        endcase
78    end
79 endmodule
80
81
82 module divider500(clock, clk_500);
83     parameter MAXN = 200000;
84     input clock;
85     output clk_500;
86     reg [17:0] cnt = 18'b0;
87     reg clk_500 = 0;
88
89     always @(posedge clock) begin
90         if (cnt == MAXN-1) begin
91             clk_500 <= 1;
92             cnt <= 18'b0;
93         end else begin
94             cnt <= cnt + 1;
95             clk_500 <= 0;
96         end
97     end
98 endmodule
99
100
101 module ring_cnt_4(clk_500, A);
102     input clk_500;
103     output [3:0] A;
104     reg [3:0] A = 4'b1110;
105
106     always @(posedge clk_500) begin
107         A[1] <= A[0];
108         A[2] <= A[1];
109         A[3] <= A[2];
110         A[0] <= A[3];
111     end

```

```

112 endmodule
113
114 module tssd(number, ssd); // to ssd
115     input [3:0] number;
116     output [6:0] ssd;
117     reg [6:0] ssd;
118
119     always @(*) begin
120         case (number)
121             0: ssd <= 7'b0000001;
122             1: ssd <= 7'b1001111;
123             2: ssd <= 7'b0010010;
124             3: ssd <= 7'b0000110;
125             4: ssd <= 7'b1001100;
126             5: ssd <= 7'b0100100;
127             6: ssd <= 7'b0100000;
128             7: ssd <= 7'b0001111;
129             8: ssd <= 7'b0000000;
130             9: ssd <= 7'b0000100;
131             4'b1010: ssd <= 7'b0001000; // A
132             4'b1011: ssd <= 7'b1100000;
133             4'b1100: ssd <= 7'b0110001;
134             4'b1101: ssd <= 7'b1000010;
135             4'b1110: ssd <= 7'b0110000;
136             4'b1111: ssd <= 7'b0111000;
137         endcase
138     end
139 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.

```

1  module testbench;
2      integer currTime;
3      reg clk;
4
5      main uut(
6          .clk (clk)
7      );
8
9      initial begin
10         #0
11         clk = 0;
12         currTime = -10;
13         uut.asset_pc.out = -4;
14
15         $display("=====");

```

```

16     #988
    $display("=====");
17     #989 $stop;
18     end
19
20     always @(posedge clk) begin
21         // indicating a posedge clk triggered
22         $display("-----
");
23         #1; // wait for writing back
24         $display("Time: %d, CLK = %d, PC = 0x%H",currTime, clk,
    uut.asset_pc.out);
25         $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]);
26         $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]);
27         $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]);
28         $display("[t1] = 0x%H, [t2] = 0x%H, [t3] =
    0x%H",uut.asset_reg.RegData[9],uut.asset_reg.RegData[10],uut.asset_reg.R
    egData[11]);
29         $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]);
30         $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]);
31     end
32
33     always #10 begin
34         clk = ~clk;
35         currTime = currTime + 10;
36     end
37
38 endmodule

```

FPGA implementation

```

1  `include "main.v"
2  module driver(
3      input clk,
4      input reset,
5      input [7:0] switch,
6      output [3:0] A,
7      output [6:0] ssd
8  );
9
10     reg [15:0] data;

```



```

11  reg clock;
12  reg [15:0] tmp2;
13  wire [4:0] regDst;
14  wire [31:0] regOut;
15  wire [31:0] pcOut;
16
17  main uut(
18      .clk (clock),
19      .syn_reg_dst (regDst),
20      .syn_reg_out (regOut),
21      .syn_pc (pcOut)
22  );
23
24  io asset_io(clk, data, A, ssd);
25
26  initial begin
27      clock = 0;
28      tmp2 = 16'b0;
29  end
30
31  assign regDst = switch[4:0];
32
33  always @(switch) begin
34      case (switch[7:5])
35          3'b000: // normal mode, display reg value
36              data = regOut[15:0];
37          3'b001: // display PC
38              data = pcOut[15:0];
39          3'b010: // display RegID
40              data = switch[4:0];
41          3'b011: // debug
42              data = tmp2;
43          default: // undefined
44              data = 16'b0101010110101100;
45      endcase
46  end
47
48
49  always @(posedge clock) begin
50      if (tmp2 < 500) tmp2 <= tmp2 + 1;
51  end
52
53  always @(posedge reset) begin
54      clock = ~clock;
55  end
56
57  endmodule
58
59  module io(
60      input clk,
61      input [15:0] data,
62      output [3:0] A, // anode

```

```

63     output reg [6:0] ssd // cathod
64 );
65
66     wire d500;
67     wire [6:0] o1,o2,o3,o4;
68
69     divider500 di500(clk,d500);
70     ring_cnt_4 rt(d500,A);
71
72     tssd outssd1(data[3:0],o1); // left-most
73     tssd outssd2(data[7:4],o2);
74     tssd outssd3(data[11:8],o3);
75     tssd outssd4(data[15:12],o4); // right-most
76
77
78     always @(posedge clk) begin
79         case (A)
80             4'b1110: ssd = o1;
81             4'b1101: ssd = o2;
82             4'b1011: ssd = o3;
83             4'b0111: ssd = o4;
84         endcase
85     end
86 endmodule
87
88
89 module divider500(clock, clk_500);
90     parameter MAXN = 200000;
91     input clock;
92     output clk_500;
93     reg [17:0] cnt = 18'b0;
94     reg clk_500 = 0;
95
96     always @(posedge clock) begin
97         if (cnt == MAXN-1) begin
98             clk_500 <= 1;
99             cnt <= 18'b0;
100         end else begin
101             cnt <= cnt + 1;
102             clk_500 <= 0;
103         end
104     end
105 endmodule
106
107
108 module ring_cnt_4(clk_500, A);
109     input clk_500;
110     output [3:0] A;
111     reg [3:0] A = 4'b1110;
112
113     always @(posedge clk_500) begin
114         A[1] <= A[0];

```

```

115     A[2] <= A[1];
116     A[3] <= A[2];
117     A[0] <= A[3];
118     end
119 endmodule

120
121 module tssd(number, ssd); // to ssd
122     input [3:0] number;
123     output [6:0] ssd;
124     reg [6:0] ssd;
125
126     always @(*) begin
127         case (number)
128             0: ssd <= 7'b0000001;
129             1: ssd <= 7'b1001111;
130             2: ssd <= 7'b0010010;
131             3: ssd <= 7'b0000110;
132             4: ssd <= 7'b1001100;
133             5: ssd <= 7'b0100100;
134             6: ssd <= 7'b0100000;
135             7: ssd <= 7'b0001111;
136             8: ssd <= 7'b0000000;
137             9: ssd <= 7'b0000100;
138             4'b1010: ssd <= 7'b0001000; // A
139             4'b1011: ssd <= 7'b1100000;
140             4'b1100: ssd <= 7'b0110001;
141             4'b1101: ssd <= 7'b1000010;
142             4'b1110: ssd <= 7'b0110000;
143             4'b1111: ssd <= 7'b0111000;
144         endcase
145     end
146 endmodule

```

Simulation Results

Instructions

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

```

1  00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
2  00100000 00001001 00000000 00110111 //addi $t1, $zero, 0x37
3  00000001 00001001 10000000 00100100 //and $s0, $t0, $t1
4  00000001 00001001 10000000 00100101 //or $s0, $t0, $t1
5  10101100 00010000 00000000 00000100 //sw $s0, 4($zero)
6  10101100 00001000 00000000 00001000 //sw $t0, 8($zero)
7  00000001 00001001 10001000 00100000 //add $s1, $t0, $t1
8  00000001 00001001 10010000 00100010 //sub $s2, $t0, $t1
9  00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
10 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
11 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20

```

```

12 00010010 00110010 00000000 00010010 //beq $s1, $s2, error0
13 10001100 00010001 00000000 00000100 //lw $s1, 4($zero)
14 00110010 00110010 00000000 01001000 //andi $s2, $s1, 0x48
15 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
16 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
17 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
18 00010010 00110010 00000000 00001111 //beq $s1, $s2, error1
19 10001100 00010011 00000000 00001000 //lw $s3, 8($zero)
20 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
21 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
22 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
23 00010010 00010011 00000000 00001101 //beq $s0, $s3, error2
24 00000010 01010001 10100000 00101010 //slt $s4, $s2, $s1 (Last)
25 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
26 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
27 00100000 00001000 00000000 00100000 //addi $t0, $zero, 0x20
28 00010010 10000000 00000000 00001111 //beq $s4, $0, EXIT
29 00000010 00100000 10010000 00100000 //add $s2, $s1, $0
30 00001000 00000000 00000000 00010111 //j Last
31 00100000 00001000 00000000 00000000 //addi $t0, $0, 0(error0)
32 00100000 00001001 00000000 00000000 //addi $t1, $0, 0
33 00001000 00000000 00000000 00111111 //j EXIT
34 00100000 00001000 00000000 00000001 //addi $t0, $0, 1(error1)
35 00100000 00001001 00000000 00000001 //addi $t1, $0, 1
36 00001000 00000000 00000000 00111111 //j EXIT
37 00100000 00001000 00000000 00000010 //addi $t0, $0, 2(error2)
38 00100000 00001001 00000000 00000010 //addi $t1, $0, 2
39 00001000 00000000 00000000 00111111 //j EXIT
40 00100000 00001000 00000000 00000011 //addi $t0, $0, 3(error3)
41 00100000 00001001 00000000 00000011 //addi $t1, $0, 3
42 00001000 00000000 00000000 00111111 //j EXIT

```

Simulation

The Simulation Results of the first set is shown below:

Simulation results of instructions(1)

```

1 -----
2 Time:           0, CLK = 1, PC = 0x00000000
3 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
4 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
5 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
6 [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
7 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
8 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
9 -----
10 Time:          20, CLK = 1, PC = 0x00000004
11 [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
12 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
13 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000

```

```

14  [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
15  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
16  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
17  -----
18  Time:          40, CLK = 1, PC = 0x00000008
19  [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
20  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
21  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
22  [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
23  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
24  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
25  -----
26  Time:          60, CLK = 1, PC = 0x0000000c
27  [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
28  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
29  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
30  [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
31  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
32  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
33  -----
34  Time:          80, CLK = 1, PC = 0x00000010
35  [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
36  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
37  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000000
38  [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
39  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
40  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
41  -----
42  Reg_WriteData: 0x00000020 | WriteReg: 8
43  Time:          100, CLK = 1, PC = 0x00000014
44  [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
45  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
46  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
47  [$t1] = 0x00000000, [$t2] = 0x00000000, [$t3] = 0x00000000
48  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
49  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
50  -----
51  Reg_WriteData: 0x00000037 | WriteReg: 9
52  Time:          120, CLK = 1, PC = 0x00000018
53  [$s0] = 0x00000000, [$s1] = 0x00000000, [$s2] = 0x00000000
54  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
55  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
56  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
57  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
58  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
59  -----
60  Reg_WriteData: 0x00000020 | WriteReg: 16
61  Time:          140, CLK = 1, PC = 0x0000001c
62  [$s0] = 0x00000020, [$s1] = 0x00000000, [$s2] = 0x00000000
63  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
64  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
65  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000

```

```

66  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
67  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
68  -----
69  Reg_WriteData: 0x00000037 | WriteReg: 16
70  Time:           160, CLK = 1, PC = 0x00000020
71  [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
72  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
73  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
74  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
75  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
76  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
77  -----
78  Time:           180, CLK = 1, PC = 0x00000024
79  [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
80  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
81  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
82  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
83  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
84  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
85  -----
86  Time:           200, CLK = 1, PC = 0x00000028
87  [$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
88  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
89  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
90  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
91  [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
92  [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
93  -----
94  Reg_WriteData: 0x00000057 | WriteReg: 17
95  Time:           220, CLK = 1, PC = 0x0000002c
96  [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0x00000000
97  [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
98  [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
99  [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
100 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
101 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
102 -----
103 Reg_WriteData: 0xffffffff9 | WriteReg: 18
104 Time:           240, CLK = 1, PC = 0x00000030
105 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff9
106 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
107 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
108 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
109 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
110 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
111 -----
112 Reg_WriteData: 0x00000020 | WriteReg: 8
113 Time:           260, CLK = 1, PC = 0x00000034
114 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffff9
115 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
116 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
117 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000

```

```

118 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
119 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
120 -----
121 Reg_WriteData: 0x00000020 | WriteReg: 8
122 Time:           280, CLK = 1, PC = 0x00000038
123 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
124 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
125 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
126 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
127 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
128 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
129 -----
130 Reg_WriteData: 0x00000020 | WriteReg: 8
131 Time:           300, CLK = 1, PC = 0x00000038
132 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
133 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
134 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
135 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
136 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
137 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
138 -----
139 Time:           320, CLK = 1, PC = 0x0000003c
140 [$s0] = 0x00000037, [$s1] = 0x00000057, [$s2] = 0xffffffffe9
141 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
142 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
143 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
144 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
145 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
146 -----
147 Reg_WriteData: 0x00000037 | WriteReg: 17
148 Time:           340, CLK = 1, PC = 0x00000040
149 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffffe9
150 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
151 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
152 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
153 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
154 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
155 -----
156 Time:           360, CLK = 1, PC = 0x00000044
157 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0xffffffffe9
158 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
159 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
160 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
161 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
162 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
163 -----
164 Reg_WriteData: 0x00000000 | WriteReg: 18
165 Time:           380, CLK = 1, PC = 0x00000048
166 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
167 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
168 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
169 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000

```

```

170 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
171 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
172 -----
173 Reg_WriteData: 0x00000020 | WriteReg: 8
174 Time:           400, CLK = 1, PC = 0x0000004c
175 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
176 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
177 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
178 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
179 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
180 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
181 -----
182 Reg_WriteData: 0x00000020 | WriteReg: 8
183 Time:           420, CLK = 1, PC = 0x00000050
184 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
185 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
186 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
187 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
188 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
189 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
190 -----
191 Reg_WriteData: 0x00000020 | WriteReg: 8
192 Time:           440, CLK = 1, PC = 0x00000054
193 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
194 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
195 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
196 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
197 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
198 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
199 -----
200 Time:           460, CLK = 1, PC = 0x00000058
201 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
202 [$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
203 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
204 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
205 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
206 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
207 -----
208 Reg_WriteData: 0x00000020 | WriteReg: 19
209 Time:           480, CLK = 1, PC = 0x0000005c
210 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
211 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
212 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
213 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
214 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
215 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
216 -----
217 Reg_WriteData: 0x00000020 | WriteReg: 8
218 Time:           500, CLK = 1, PC = 0x00000060
219 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
220 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
221 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020

```



```

222 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
223 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
224 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
225 -----
226 Reg_WriteData: 0x00000020 | WriteReg: 8
227 Time: 520, CLK = 1, PC = 0x00000064
228 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
229 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
230 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
231 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
232 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
233 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
234 -----
235 Reg_WriteData: 0x00000020 | WriteReg: 8
236 Time: 540, CLK = 1, PC = 0x00000068
237 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
238 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
239 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
240 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
241 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
242 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
243 -----
244 Time: 560, CLK = 1, PC = 0x0000006c
245 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
246 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
247 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
248 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
249 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
250 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
251 -----
252 Reg_WriteData: 0x00000001 | WriteReg: 20
253 Time: 580, CLK = 1, PC = 0x00000070
254 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
255 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
256 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
257 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
258 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
259 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
260 -----
261 Reg_WriteData: 0x00000020 | WriteReg: 8
262 Time: 600, CLK = 1, PC = 0x00000074
263 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
264 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
265 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
266 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
267 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
268 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
269 -----
270 Reg_WriteData: 0x00000020 | WriteReg: 8
271 Time: 620, CLK = 1, PC = 0x00000078
272 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
273 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000

```

```

274 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
275 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
276 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
277 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
278 -----
279 Reg_WriteData: 0x00000020 | WriteReg: 8
280 Time:           640, CLK = 1, PC = 0x0000005c
281 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
282 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
283 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
284 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
285 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
286 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
287 -----
288 Time:           660, CLK = 1, PC = 0x00000060
289 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000000
290 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
291 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
292 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
293 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
294 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
295 -----
296 Reg_WriteData: 0x00000037 | WriteReg: 18
297 Time:           680, CLK = 1, PC = 0x00000064
298 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
299 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
300 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
301 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
302 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
303 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
304 -----
305 Time:           700, CLK = 1, PC = 0x00000068
306 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
307 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
308 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
309 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
310 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
311 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
312 -----
313 Time:           720, CLK = 1, PC = 0x0000006c
314 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
315 [$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
316 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
317 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
318 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
319 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
320 -----
321 Reg_WriteData: 0x00000000 | WriteReg: 20
322 Time:           740, CLK = 1, PC = 0x00000070
323 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
324 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
325 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020

```

```

326 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
327 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
328 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
329 -----
330 Reg_WriteData: 0x00000020 | WriteReg: 8
331 Time: 760, CLK = 1, PC = 0x000000ac
332 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
333 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
334 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
335 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
336 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
337 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
338 -----
339 Reg_WriteData: 0x00000020 | WriteReg: 8
340 Time: 780, CLK = 1, PC = 0x000000b0
341 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
342 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
343 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
344 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
345 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
346 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
347 -----
348 Reg_WriteData: 0x00000020 | WriteReg: 8
349 Time: 800, CLK = 1, PC = 0x000000b4
350 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
351 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
352 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
353 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
354 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
355 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
356 -----
357 Time: 820, CLK = 1, PC = 0x000000b8
358 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
359 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
360 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
361 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
362 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
363 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
364 -----
365 Time: 840, CLK = 1, PC = 0x000000bc
366 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
367 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
368 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
369 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
370 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
371 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
372 -----
373 Time: 860, CLK = 1, PC = 0x000000c0
374 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
375 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
376 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
377 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000

```

```

378 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
379 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
380 -----
381 Time:           880, CLK = 1, PC = 0x000000c4
382 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
383 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
384 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
385 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
386 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
387 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
388 -----
389 Time:           900, CLK = 1, PC = 0x000000c8
390 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
391 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
392 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
393 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
394 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
395 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
396 -----
397 Time:           920, CLK = 1, PC = 0x000000cc
398 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
399 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
400 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
401 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
402 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
403 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
404 -----
405 Time:           940, CLK = 1, PC = 0x000000d0
406 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
407 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
408 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
409 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
410 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
411 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
412 -----
413 Time:           960, CLK = 1, PC = 0x000000d4
414 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
415 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
416 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
417 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
418 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
419 [$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
420 =====
421 -----
422 Time:           980, CLK = 1, PC = 0x000000d8
423 [$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
424 [$s3] = 0x00000020, [$s4] = 0x00000000, [$s5] = 0x00000000
425 [$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
426 [$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
427 [$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
428 [$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