

# COD\_\_LAB5 多周期 MIPS-CPU

黄业琦 PB17000144

May 7, 2019

## Contents

<b>1</b>	<b>实验目的</b>	<b>2</b>
<b>2</b>	<b>实验环境</b>	<b>3</b>
<b>3</b>	<b>逻辑设计</b>	<b>3</b>
<b>4</b>	<b>仿真截图</b>	<b>9</b>
<b>5</b>	<b>性能评测截图</b>	<b>10</b>
<b>6</b>	<b>实验代码</b>	<b>10</b>
6.1	烧写板版本代码 . . . . .	10
6.2	模拟版本代码 . . . . .	31
<b>7</b>	<b>总结</b>	<b>41</b>
<b>8</b>	<b>意见和建议</b>	<b>41</b>

# 1 实验目的

设计实现多周期 MIPS-CPU，可执行如下指令：

- add, sub, and, or, xor, nor, slt
- addi, andi, ori, xori, slti
- lw, sw
- beq, bne, j

数据通路和控制单元 (状态图) 参见后页，其中寄存器堆中 R0 内容恒定为 0，存储器容量为 256x32 位。

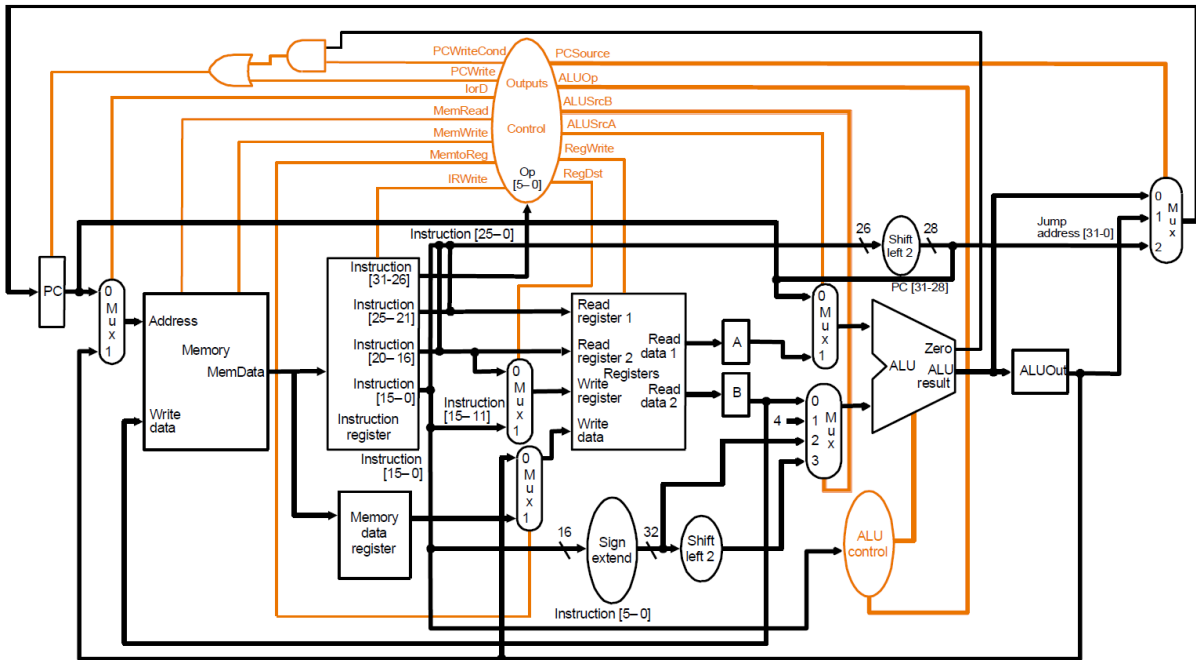


Figure 1: Route1

DDU: Debug and Display Unit, 调试和显示单元

下载测试时，用于控制 CPU 运行方式和显示运行结果

数据通路中寄存器堆和存储器均需要增加 1 个读端口，供 DDU 读取并显示其中内容

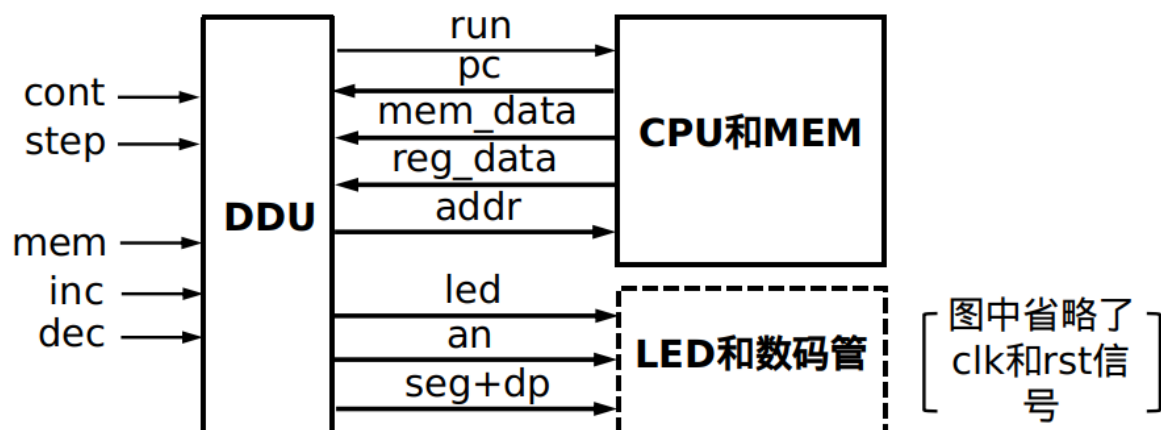


Figure 2: DDU

控制 CPU 运行方式:

- **cont = 1**: **run = 1**, 控制 CPU 连续执行指令
- **cont = 0**: 每按动 **step** 一次, **run** 输出维持一个时钟周期的脉冲, 控制 CPU 执行一条指令

查看 CPU 运行状态:

- **mem**: 1, 查看 MEM; 0, 查看 RF
- **inc/dec**: 增加或减小待查看 RF/MEM 的地址 **addr**
- **reg\_data/mem\_data**: 从 RF/MEM 读取的数据
- 8 位数码管显示 RF/MEM 的一个 32 位数据
- 16 位 LED 指示 RF/MEM 的地址和 PC 的值

## 2 实验环境

Linux 下编程调试和仿真, 使用 IVerilog, GtkWave 系列工具。

Windows 下用于生成比特流文件, 使用 Vivado 2018.2, Verilog HDL

所有下载均在 Nexsy4-DDR 实验板完成。

优秀的代码风格和规范的代码格式也很重要, 本次实验借助 Vscode 的 verilog-format 插件进行整理代码的工作。

## 3 逻辑设计

代码逻辑结构参照我们的 Figure1:Route1, 但是我们需要做一点修改以方便编程实现。我们为了简化 **beq** 和 **bne**, 我们需要额外增加一个加法器, 专门实现 **imm** 和 **pc** 的相加。

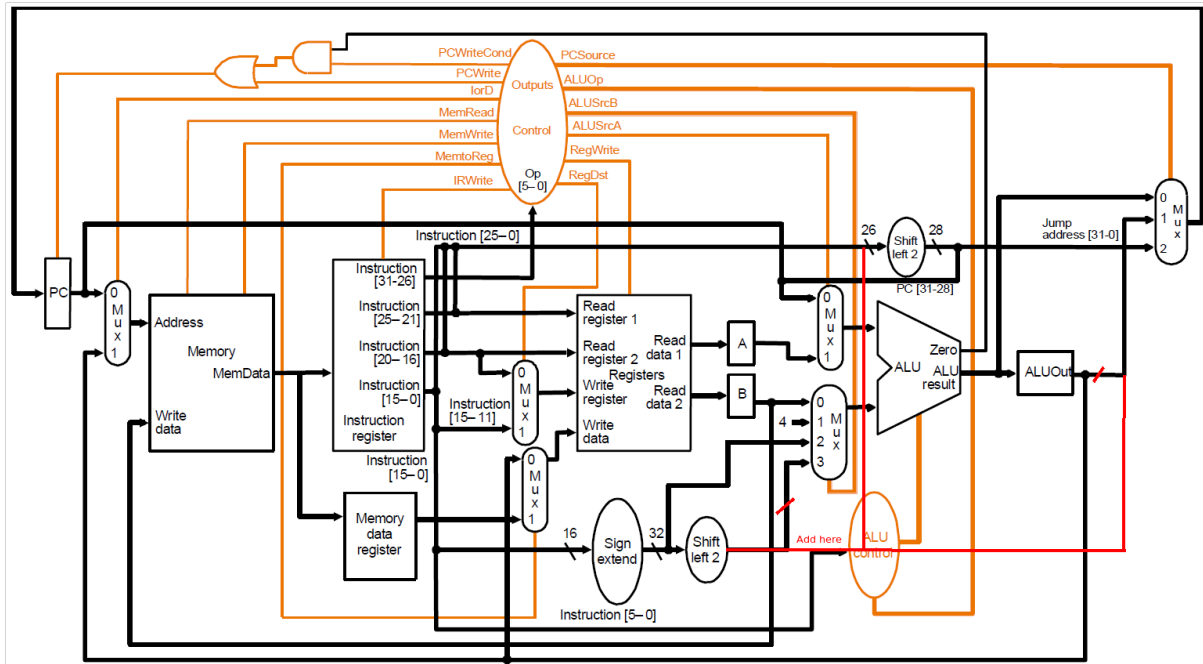


Figure 3: New Route

关于逻辑结构，我们使用有限状态机进行描述。我们讲义给定的参考状态图是 Figure: fsm，但是我们图中并没有我们关于 I-type 的算数运算的状态，为了弥补这一问题，我们对于一个状态进行了调整：fsm\_new。

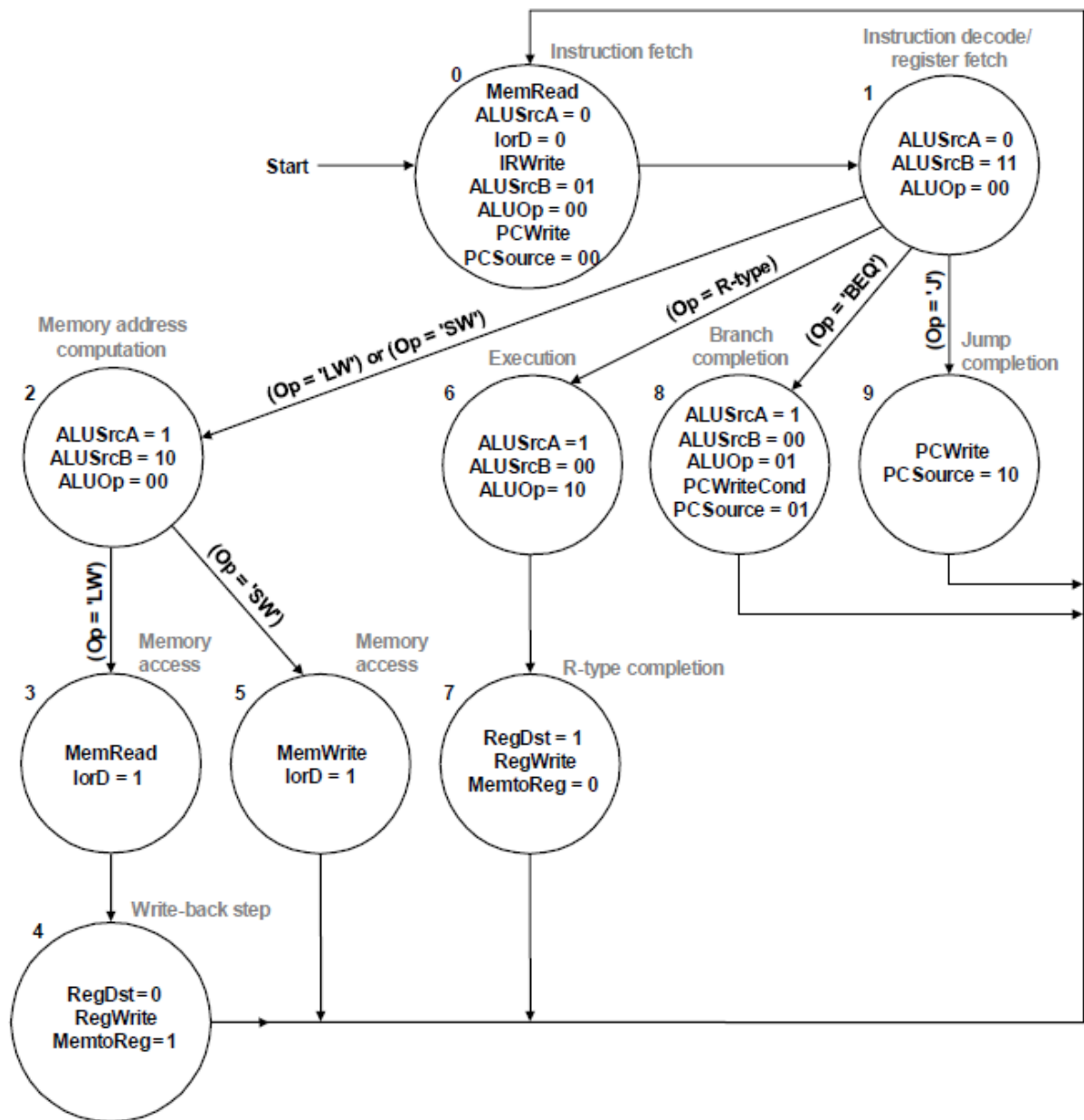


Figure 4: FSM

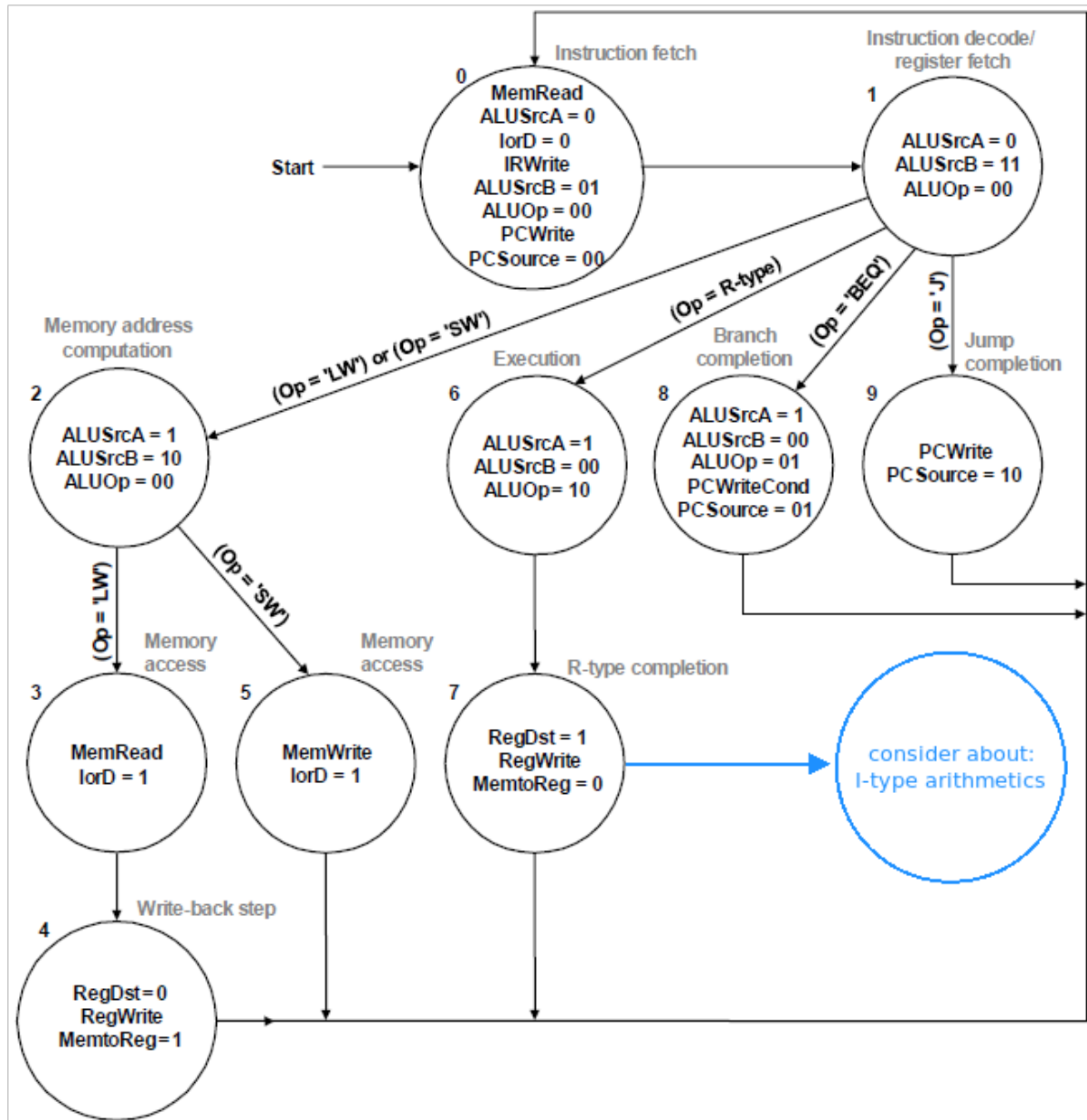


Figure 5: FSM\_new

这样我们就可以处理关于立即数计算的指令。

同时为了适应简化后的设计线路图，我们还做了以下改动：

1. 我们将 aluop 设作虚设，不做使用。
2. alusrcb 只有一位就足以区分，关于 pc 的处理单独设立的新的加法器。
3. alusrcA 的值进行了小调整。
4. 关于 bne 和 beq 的设计。再 alu 中新添加了 Compare equal 的功能，这一设计个人认为十分巧妙在 pc\_reg 中有具体介绍这么做的原因。

状态机的详细设计再 define 文件中有相应的描述。

状态设计如下，注释已经详细介绍了各个状态的具体功能。

Listing 1: State design

```

‘define IF_STATE      4'b0000 // * if state
‘define ID_STATE      4'b0001 // * id state
‘define EXE1_STATE    4'b0010 // * exe for lw or sw
‘define EXE2_STATE    4'b0011 // * exe for add/sub...(R&I type together)
‘define EXE3_STATE    4'b0100 // * exe for branch (beq & bne)
‘define EXE4_STATE    4'b0101 // * exe for j inst
‘define MEM1_STATE    4'b0110 // * mem for lw - memread
‘define MEM2_STATE    4'b0111 // * mem for sw - memwrite
‘define WB1_STATE     4'b1000 // * write back for memory inst
‘define WB2_STATE     4'b1001 // * write back for add/sub...

```

关于代码，采用模块化的设计，各模块及其功能如下（顺序为字典序）：

- IP 核 clk\_wiz: 避免时钟周期过短。
- IP 核 dist\_mem: 实现内存。
- alu: 实现运算器，纯组合逻辑。
- alu\_ctrl: 将指令的 opcode 和 funt 部分转化为可以 alu 的具体功能。
- bcd27: 数码管显示用，纯逻辑。
- control: 控制器，根据 6 位 op 实现各个控制信号的输出，同时控制有限状态机。
- cpu: cpu，组合各个 cpu 组件。
- ddu: 顶层，控制 cpu 运行以及 led、数码管的显示。
- define: 定义常量。
- extend: 16 位转换为 32 位。
- ins\_split: ir 控制读写，并且拆解指令。
- mdr: 将 mem 中读取的数暂存，用于 lw 指令。
- mem: 内存控制，用于包装 IP 核。
- mux2: 二选一选择器。
- opj\_extend: 针对 j 指令的扩展命令。
- pc\_reg: 控制 PC。
- regfile: 寄存器组。
- segout: 数码管显示。

用于测试的汇编代码为：

```
1 # 本文档存储器以字节编址
2 j _start
3
4 .data
5 .word 0,8,1,6,0xffffffff,1,3,5,0
6 #编译成机器码时便器器会在前面多加个0，所以后面lw指令地址会多加4
7
8 _start:
9 addi $t0,$0,3      #t0=3
10 addi $t1,$0,5      #t1=5
11 addi $t2,$0,1      #t2=1
12
13 add $s0,$t1,$t0
14 #s0=t1+t0=8 测试add指令 正确继续执行
15 lw $s1,12($0)
16 bne $s1,$s0,_fail
17 #不正确跳到fail
18
19 and $s0,$t1,$t0
20 #s0=t1&t0=1 测试and指令 正确继续执行
21 lw $s1,16($0)
22 bne $s1,$s0,_fail
23
24 xor $s0,$t1,$t0
25 #s0=t1^t0=6 测试xor指令 正确继续执行
26 lw $s1,20($0)
27 bne $s1,$s0,_fail
28
29 nor $s0,$t1,$t0
30 #s0=t1 nor t0=0xffffffff
31 lw $s1,24($0)
32 bne $s1,$s0,_fail
33
34 slt $s0,$t0,$t1 #s0=1
35 lw $s1,28($0)
36 bne $s1,$s0,_fail
37
38 andi $s0,$t0,7 #s0=3
39 lw $s1,32($0)
40 bne $s1,$s0,_fail
41
42 ori $s0,$t1,4 #s0=5
43 lw $s1,36($0)
44 bne $s1,$s0,_fail
45
46 sw $t1,40($0)
47 lw $s1,40($0)
48 beq $t1,$s1,_sucess
49
50 _fail:
51 sw $0,8($0)
52 #失败通过看存储器地址0x08里值，若为0则测试不通过，最初地址0x08里值为0
53 j _fail
54
55 _sucess:
56 sw $t2,8($0)
```



```

57 #全部测试通过，存储器地址0x08里值为1
58 j      _success
59
60 #判断测试通过的条件是最后存储器地址0x08里值为1，说明全部通过测试

```

也就是说，加入程序正确，那么程序将在 success 处进入死循环。我们利用 ip 核的 dist\_mem 直接将汇编好的文件转为 coe 文件，直接导入。节约工作量。

## 4 仿真截图

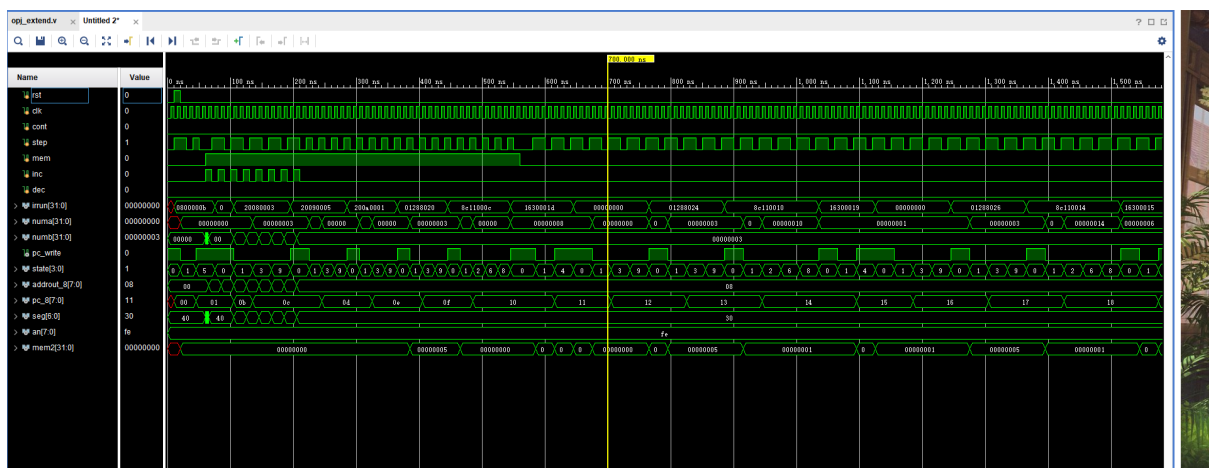


Figure 6: simulation1

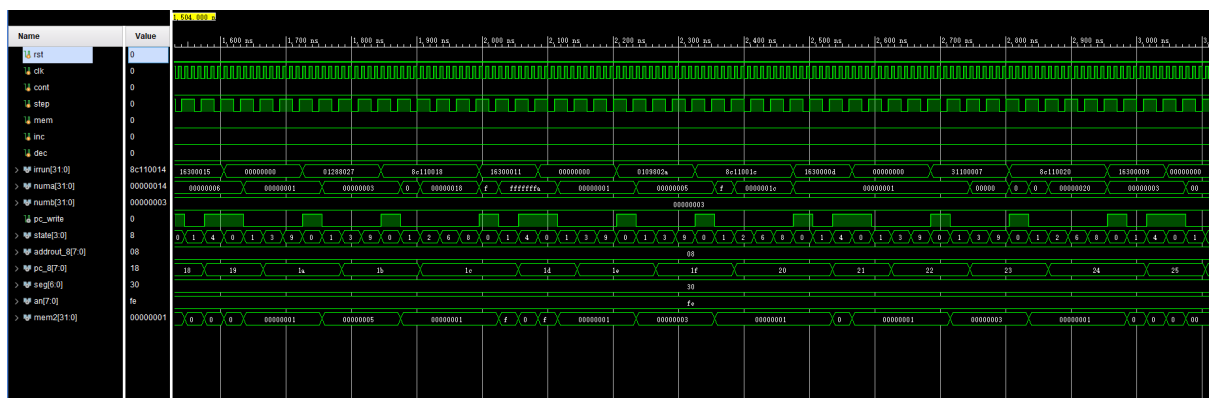


Figure 7: simulation2

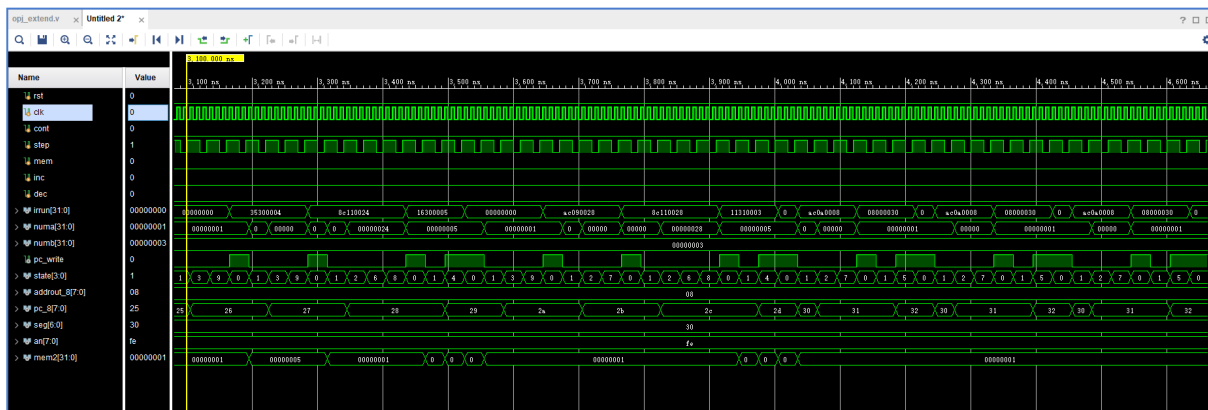


Figure 8: simulation3

拼接起来看图，最终确实进入了那个关于 success 的死循环之中。

## 5 性能评测截图

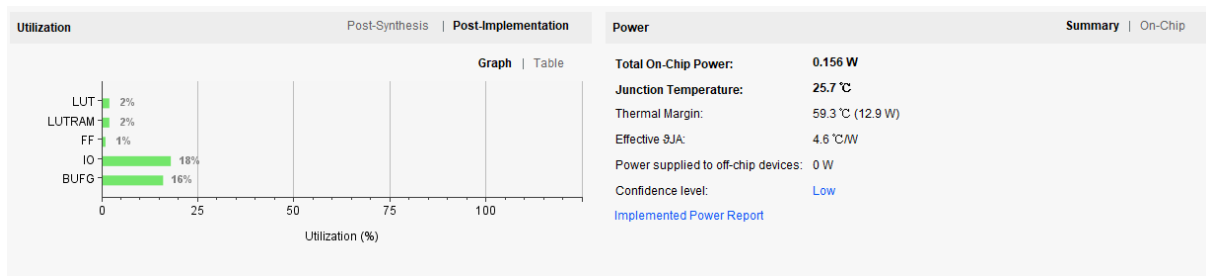


Figure 9: performance1

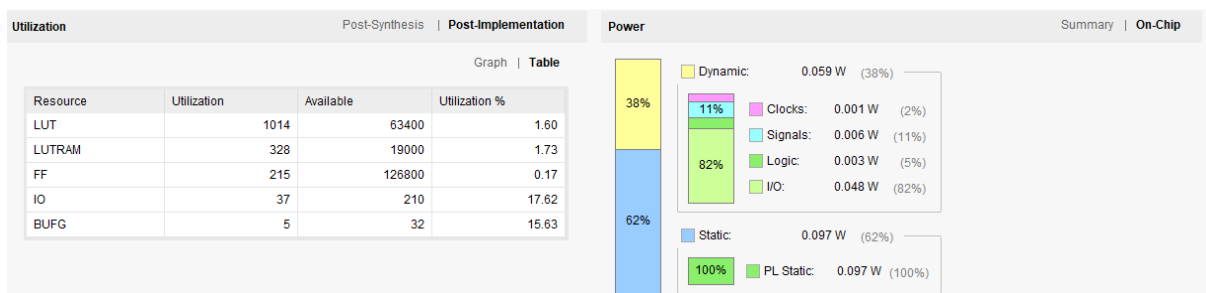


Figure 10: performance2

## 6 实验代码

### 6.1 烧写板版本代码

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module alu (input wire [`EXE_BUS] aluop,
5             input wire [`REG_BUS] numa,
6             input wire [`REG_BUS] numb,
7             output reg [`REG_BUS] num_out,
8             output reg iszero);
9
10 always @* begin
11     case (aluop)
12         `EXE_ADD :num_out = numa + numb;
13         `EXE_SUB :num_out = numa - numb;
14         `EXE_CMP :num_out = numa == numb ? 1 :0;
15         `EXE_SLT :num_out = numa < numb ? 1 :0;
16         `EXE_NOR :num_out = ~ numa;
17         `EXE_OR  :num_out = numa | numb;
18         `EXE_AND :num_out = numa & numb;
19         `EXE_XOR :num_out = numa ^ numb;
20         `EXE_NOP :num_out = numa;
21     endcase
22 end
23
24 always @* begin
25     iszero = (num_out == 0 ? 1 :0);
26 end
27 endmodule

```

Listing 3: alu.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module alu_ctrl (input wire[`OP_BUS] opcode,
5                  input wire [`FUNT_BUS] funt,
6                  output reg [`EXE_BUS] alu_op);
7
8  always @* begin
9      case (opcode)
10         `OP_R_TYPE :begin
11             case (funt) // * R - type translate
12                 `FUNC_ADD :alu_op = `EXE_ADD;
13                 `FUNC_SUB :alu_op = `EXE_SUB;
14                 `FUNC_AND :alu_op = `EXE_AND;
15                 `FUNC_OR  :alu_op = `EXE_OR;
16                 `FUNC_XOR :alu_op = `EXE_XOR;
17                 `FUNC_NOR :alu_op = `EXE_NOR;
18                 `FUNC_SLT :alu_op = `EXE_SLT;
19             endcase
20         end
21         `OP_ADDI :alu_op = `EXE_ADD;

```

```

22     `OP_ANDI :alu_op = `EXE_AND;
23     `OP_ORI  :alu_op = `EXE_OR;
24     `OP_XORI :alu_op = `EXE_XOR;
25     `OP_LW   :alu_op = `EXE_ADD;
26     `OP_SW   :alu_op = `EXE_ADD;
27     `OP_BEQ  :alu_op = `EXE_CMP;
28     `OP_BNE  :alu_op = `EXE_CMP;
29     `OP_SLTI :alu_op = `EXE_SLT;
30     `OP_J    :alu_op = `EXE_NOP;
31 endcase
32 end
33
34 endmodule

```

Listing 4: alu\_ctrl.v

```

1  `timescale 1ns / 1ps
2
3  module BCD27(input [3:0]m,
4              output [6:0] out);
5  reg[6:0]seg;
6  assign out = seg;
7  always@(m)
8      case(m)
9          //-----1234567
10         4'b0000:seg = 7'b1000000; // 0
11         4'b0001:seg = 7'b1111001; // 1
12         4'b0010:seg = 7'b0100100; // 2
13         4'b0011:seg = 7'b0110000; // 3
14         4'b0100:seg = 7'b0011001; // 4
15         4'b0101:seg = 7'b0010010; // 5
16         4'b0110:seg = 7'b0000010; // 6
17         4'b0111:seg = 7'b1111000; // 7
18         4'b1000:seg = 7'b0000000; // 8
19         4'b1001:seg = 7'b0010000; // 9
20         4'b1010:seg = 7'b0001000; // A
21         4'b1011:seg = 7'b0000011; // b
22         4'b1100:seg = 7'b1000110; // c
23         4'b1101:seg = 7'b0100001; // d
24         4'b1110:seg = 7'b0000110; // E
25         4'b1111:seg = 7'b0001110; // F
26         default:seg = 7'b1111111;
27     endcase
28 end
29 endmodule

```

Listing 5: bcd27.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  /* Controller

```

```

5  /** input with op
6  /** output control information
7
8  module control (input wire clk,
9      input wire rst,
10     input wire [`OP_BUS] op, // * input the opcode part
11     output reg pc_write_cond, // * control pc
12     output reg pc_write,      // * control pc
13     output reg ins_or_data,   // * control mem
14     output reg mem_read,      // * control mem
15     output reg mem_write,     // * control mem
16     output reg ir_write,      // * control ir
17     output reg [1:0] pc_source, // * control pc source
18     output reg [1:0] alu_op,  // * control alu
19     output reg alu_srca,      // * control numa
20     output reg alu_srcb,      // * control numb
21     output reg reg_write,     // * control regfile writable
22     output reg regdst,        // * control where to write data
23     output reg mem_to_reg); // * control the data source
24
25  reg [`STATE_BUS] state,next_state;
26
27  initial begin
28      state      = `IF_STATE;
29      pc_write_cond = 0;
30      pc_write     = 0;
31      ins_or_data  = 0;
32      mem_read     = 0;
33      mem_write    = 0;
34      ir_write     = 0;
35      pc_source    = 0;
36      alu_op       = 0;
37      alu_srca     = 0;
38      alu_srcb     = 0;
39      reg_write    = 0;
40      regdst       = 0;
41  end
42
43  always @(posedge clk or posedge rst) begin
44      if (rst) begin
45          state <= `IF_STATE;
46      end else begin
47          state <= next_state;
48      end
49  end
50
51  always @* begin
52      case (state)
53          `IF_STATE :next_state = `ID_STATE;
54          `ID_STATE :begin
55              case (op)

```

```

56         `OP_LW : next_state = `EXE1_STATE;
57         `OP_SW : next_state = `EXE1_STATE;
58         `OP_BEQ : next_state = `EXE3_STATE;
59         `OP_BNE : next_state = `EXE3_STATE;
60         `OP_J  : next_state = `EXE4_STATE;
61         default : next_state = `EXE2_STATE;
62     endcase
63 end
64 `EXE1_STATE : begin
65     if (op == `OP_LW)
66         next_state = `MEM1_STATE;
67     else
68         next_state = `MEM2_STATE;
69     end
70     `EXE2_STATE : next_state = `WB2_STATE;
71     `EXE3_STATE : next_state = `IF_STATE;
72     `EXE4_STATE : next_state = `IF_STATE;
73     `MEM1_STATE : next_state = `WB1_STATE;
74     `MEM2_STATE : next_state = `IF_STATE;
75     `WB1_STATE : next_state = `IF_STATE;
76     `WB2_STATE : next_state = `IF_STATE;
77 endcase
78 end
79
80 always @* begin
81     // * if state == > initial state
82     if (state == `IF_STATE) begin
83         pc_write_cond = 0;
84         pc_write      = 1;      // * pc writable
85         ins_or_data = 0;
86         mem_read      = 1;      // * read mem for instructions
87         mem_write     = 0;
88         ir_write       = 1;      // * ir writable
89         pc_source      = 2'b00;
90         alu_op         = 0;
91         alu_srca       = 0;
92         alu_srcb       = 0;
93         reg_write      = 0;
94         regdst         = 0;
95     end
96
97     // * id state
98     if (state == `ID_STATE) begin
99         pc_write = 0;
100         ir_write = 0;
101         mem_read = 0;
102     end
103
104     // * exe1 state
105     if (state == `EXE1_STATE) begin
106         alu_srca = 0;

```

```

107     alu_srcb = 1;
108     alu_op  = 2'b00;
109 end
110
111 // * exe2 state
112 if (state == `EXE2_STATE) begin
113     alu_srca = 0;
114     alu_srcb = (op == `OP_R_TYPE ? 0 :1);
115     alu_op  = 2'b10;
116 end
117
118 // * exe3 state
119 if (state == `EXE3_STATE) begin
120     alu_srca    = 1;
121     alu_srcb    = 0;
122     alu_op      = 2'b01;
123     pc_write    = 1;
124     pc_write_cond = (op == `OP_BEQ ? 1 :0);
125     pc_source   = 2'b01;
126 end
127
128 // * exe4 state
129 if (state == `EXE4_STATE) begin
130     pc_write = 1;
131     pc_source = 2'b10;
132 end
133
134 // * mem1 state
135 if (state == `MEM1_STATE) begin
136     mem_read  = 1;
137     ins_or_data = 1;
138 end
139
140 // * mem2 state
141 if (state == `MEM2_STATE) begin
142     mem_write = 1;
143     ins_or_data = 1;
144 end
145
146 // * wb1 state
147 if (state == `WB1_STATE) begin
148     regdst    = 0;
149     reg_write = 1;
150     mem_to_reg = 1;
151 end
152
153 // * wb2 state
154 if (state == `WB2_STATE) begin
155     regdst    = (op == `OP_R_TYPE ? 1 :0);
156     reg_write = 1;
157     mem_to_reg = 0;

```

```

158         end
159     end
160 endmodule

```

Listing 6: control.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module cpu (input wire clk,
5              input wire rst,
6              output [`REG_BUS] pcout,
7              input  [`REG_BUS] inputaddr,
8              output  [`REG_BUS] memout_show,
9              output  [`REG_BUS] regout_show);
10
11  // * ir split
12  wire [`NUM1_BUS] num1;
13  wire [`NUM2_BUS] num2;
14  wire [`NUM3_BUS] num3;
15  wire [`NUM4_BUS] num4;
16  wire [`NUM5_BUS] num5;
17  wire [`NUM6_BUS] num6;
18
19  wire [`REG_BUS] jext;
20  wire [`REG_BUS] aluout;
21  wire [`REG_BUS] numa;
22  wire [`REG_BUS] numb;
23  wire [`REG_BUS] alu_numa;
24  wire [`REG_BUS] alu_numb;
25  wire [`REG_BUS] memaddr;
26  wire [`REG_BUS] memmdr;
27  wire [`REG_BUS] regmem;
28  wire [`REG_BUS] immext;
29  wire [`REG_BUS] memdata;
30
31  wire [`EXE_BUS] aluexe;
32  wire [`OP_BUS] opcode;
33
34  // * control wire
35  wire ins_or_data;
36  wire pc_write_cond;
37  wire pc_write;
38  wire alu_srca;
39  wire alu_srcb;
40  wire ir_write;
41  wire mem_write;
42  wire mem_read;
43  wire is_zero;
44  wire mem_to_reg;
45  wire reg_write;
46  wire regdst;
47  wire [1:0] pc_source;

```



```

47  wire [1:0] alu_op;
48
49  opj_extend jexter (
50  .ins (num5),
51  .pc (pcout),
52  .extd (jext));
53
54  pc_reg pc (
55  .clk (clk),
56  .rst (rst),
57  .pccond (pc_write_cond),
58  .pcsource (pc_source),
59  .aluinfo(is_zero),
60  .pc_write(pc_write),
61  .immpc (num3),
62  .address(jext),
63  .pcout (pcout));
64
65  // * A little change here
66  // * address must divide by 4 here --> real addr in mem
67
68  // * get the address
69  mux2 mux_ins_or_data (
70  .select (ins_or_data),
71  .route1 (aluout >> 2),
72  .route2 (pcout >> 2),
73  .outdata (memaddr));
74
75  // * memory
76  mem memory (
77  .clk(clk),
78  .memwrite (mem_write),
79  .indata (numb),
80  .addr (memaddr),
81  .memdata (memdata),
82  .dpra (inputaddr),
83  .dpo (memout_show));
84
85  // * ir
86  ins_split ir (
87  .ir_write (ir_write),
88  .ins (memdata),
89  .opcode (opcode),
90  .num1 (num1),
91  .num2 (num2),
92  .num3 (num3),
93  .num4 (num4),
94  .num5 (num5),
95  .num6(num6),
96  .irrun (irrun));
97

```

```

98 // * mdr
99 mdr memory_data_reg (
100 .memread (mem_read),
101 .memdata (memdata),
102 .memout (memmdr));
103
104 // * control
105 control ctrl (
106 .clk (clk),
107 .rst (rst),
108 .op (opcode),
109 .pc_write_cond (pc_write_cond),
110 .pc_write (pc_write),
111 .ins_or_data (ins_or_data),
112 .mem_read(mem_read),
113 .mem_write (mem_write),
114 .ir_write(ir_write),
115 .pc_source(pc_source),
116 .alu_op (alu_op),
117 .alu_srca (alu_srca),
118 .alu_srcb (alu_srcb),
119 .reg_write (reg_write),
120 .regdst (regdst),
121 .mem_to_reg (mem_to_reg));
122
123 // * reg write
124 mux2 reg_write_mem (
125 .select (mem_to_reg),
126 .route1(memmdr),
127 .route2(aluout),
128 .outdata (regmem));
129
130 // * reg
131 regfile regs (
132 .clk (clk),
133 .rs (num1),
134 .rt (num2),
135 .rd (num4),
136 .numa (numa),
137 .numb (numb),
138 .reg_write (reg_write),
139 .regdst (regdst),
140 .indata (regmem),
141 .regaddr (inputaddr),
142 .reg_out(regout_show));
143
144 // * mux for numa
145 mux2 muxnuma (
146 .select (alu_srca),
147 .route1 (pcout),
148 .route2 (numa) ,

```

```

149     .outdata (alu_numa));
150
151     // * extend immediate num
152     extend extder (
153     .ins (num3),
154     .extd (immext));
155
156     // * mux for numb
157     mux2 muxnumb (
158     .select (alu_srcb),
159     .route1 (immext),
160     .route2(numb),
161     .outdata (alu_numb));
162
163     // * translate alu info
164     alu_ctrl aluexe_maker (
165     .opcode (opcode),
166     .funt (num6),
167     .alu_op (aluexe));
168
169     // * alu
170     alu alucalc (
171     .aluop (aluexe),
172     .numa (alu_numa),
173     .numb(alu_numb),
174     .num_out (aluout),
175     .iszero (is_zero));
176 endmodule

```

Listing 7: cpu.v

```

1  `timescale 1ns / 1ps
2  `include "define.v"
3
4  module ddu(input cont,
5             input rst,
6             input clk_board100,
7             input step,
8             input mem,
9             input inc,
10             input dec,
11             output reg [7:0] addrout_8,
12             output wire [7:0] pc_8,
13             output wire [6:0] seg,
14             output wire [7:0] an);
15
16     wire clk_boardtmp;
17
18     // * translate 100Mhz
19     clk_wiz_0 (.clk_in1(clk_board100),.clk_out1(clk_boardtmp));
20
21     wire [`REG_BUS] memshow;

```

```

22  reg [`REG_BUS] addrout;
23  wire [`REG_BUS] pc;
24  reg clk_board = 0;
25  reg [14:0] cnt;
26
27  // * slow clock
28  always @(posedge clk_boardtmp) begin
29      if (cnt >= 15'd100) begin
30          cnt = 0;
31          clk_board = ~clk_board;
32      end
33      else
34          cnt = cnt + 1;
35  end
36
37  reg [`REG_BUS] addr = 0;
38  wire [`REG_BUS] mem1;
39  wire [`REG_BUS] mem2;
40  reg clkt;
41  reg delay;
42  reg incdelay;
43  reg decdelay;
44  wire clk;
45
46  always @(posedge clk_board or negedge rst)
47  begin
48      if (rst) begin
49          addr <= 0;
50          incdelay = 1;
51          decdelay = 1;
52          delay = 0;
53      end
54      else
55      begin
56          // * control addr
57          if (!incdelay && inc) addr = addr + 1;
58          if (!decdelay && dec) addr = addr - 1;
59
60          incdelay = inc;
61          decdelay = dec;
62
63          addrout = addr;
64
65          // * generate only one pulse
66          if (!delay && step)
67              clkt = 1;
68          else
69              clkt = 0;
70
71          delay = step;
72  end

```

```

73     end
74
75     assign memshow = mem ? mem1 : mem2;
76     assign clk      = cont ? clk_board : clkt;
77
78     // * cpu
79     cpu CPU_sim (.clk (clk), .rst (rst), .pcout (pc), .inputaddr(addr) , .
        memout_show (mem1), .regout_show(mem2));
80
81     always @* addrout_8 = {addr [7:0]};
82     assign pc_8         = {pc[9:2]};
83
84     // * segout
85     segout segmental (.clk (clk_board), .indata (memshow), .an(an) ,.seg(seg));
86 endmodule

```

Listing 8: ddu.v

```

1  /* r_type
2  /* begin with 000000
3  `define OP_R_TYPE 6'b000000
4  `define FUNC_ADDU 6'b100001
5  `define FUNC_ADD 6'b100000 /* do
6  `define FUNC_SUB 6'b100010 /* do
7  `define FUNC_SUBU 6'b100011
8  `define FUNC_AND 6'b100100 /* do
9  `define FUNC_OR 6'b100101 /* do
10 `define FUNC_XOR 6'b100110 /* do
11 `define FUNC_NOR 6'b100111 /* do
12 `define FUNC_SLT 6'b101010 /* do
13 `define FUNC_SLTU 6'b101011
14 `define FUNC_SLL 6'b000000 /* do
15 `define FUNC_SRL 6'b000010 /* do
16 `define FUNC_SRA 6'b000011
17 `define FUNC_SLLV 6'b000100
18 `define FUNC_SRLV 6'b000110
19 `define FUNC_SRAV 6'b000111
20 `define FUNC_JR 6'b001000
21
22 /* i_type
23 `define OP_ADDI 6'b001000 /* do
24 `define OP_ANDI 6'b001100 /* do
25 `define OP_ORI 6'b001101 /* do
26 `define OP_XORI 6'b001110 /* do
27 `define OP_LUI 6'b001111
28 `define OP_LW 6'b100011 /* do
29 `define OP_SW 6'b101011 /* do
30 `define OP_BEQ 6'b000100 /* do
31 `define OP_BNE 6'b000101 /* do
32 `define OP_SLTI 6'b001010 /* do
33 `define OP_SLTIU 6'b001101
34

```

```

35  /* j_type
36  `define OP_J      6'b000010 /* do
37  `define OP_JAL   6'b000011
38
39  /* exe will be used
40  `define EXE_ADD  4'b0000
41  `define EXE_SUB  4'b0001
42  `define EXE_CMP  4'b0010
43  `define EXE_SLT  4'b0011
44  `define EXE_NOR  4'b0100
45  `define EXE_OR   4'b0101
46  `define EXE_AND  4'b0110
47  `define EXE_XOR  4'b0111
48  `define EXE_NOP  4'b1111
49
50  /* bus_information
51  `define PCCTRL_BUS 1:0
52  `define STATE_BUS  3:0
53  `define EXE_BUS    3:0
54  `define RS_BUS     4:0
55  `define RT_BUS     4:0
56  `define RD_BUS     4:0
57  `define SHAMT_BUS  4:0
58  `define STALL_BUS  5:0
59  `define OP_BUS     5:0
60  `define FUNT_BUS   5:0
61  `define ADDR_BUS   7:0
62  `define DATA_BUS  31:0
63  `define REG_BUS    31:0
64
65  `define OPCODE_BUS 31:26
66  `define NUM1_BUS   25:21
67  `define NUM2_BUS   20:16
68  `define NUM3_BUS   15:0
69  `define NUM4_BUS   15:11
70  `define NUM5_BUS   25:0
71  `define NUM6_BUS   5:0
72  `define FUNTCODE_BUS 5:0
73
74  `define REG_VEC   0:31
75  `define MEM_VEC   0:255
76
77  /* control tags
78  `define INS_TAG   0
79  `define REG_TAG   1
80
81  /* control state machine
82  `define IF_STATE  4'b0000 // * if state
83  `define ID_STATE  4'b0001 // * id state
84  `define EXE1_STATE 4'b0010 // * exe for lw or sw
85  `define EXE2_STATE 4'b0011 // * exe for add/sub...(R&I type together)

```

```

86 `define EXE3_STATE 4'b0100 // * exe for branch (beq & bne)
87 `define EXE4_STATE 4'b0101 // * exe for j inst
88 `define MEM1_STATE 4'b0110 // * mem for lw - memread
89 `define MEM2_STATE 4'b0111 // * mem for sw - memwrite
90 `define WB1_STATE 4'b1000 // * write back for memory inst
91 `define WB2_STATE 4'b1001 // * write back for add/sub...

```

Listing 9: define.v

```

1 `include "define.v"
2 `timescale 1ns / 1ps
3
4 module extend(input wire [`NUM3_BUS] ins,
5               output wire [`REG_BUS] extd);
6 assign
7 extd = 32'b0 | ins[`NUM3_BUS];
8
9 endmodule

```

Listing 10: extend.v

```

1 `include "define.v"
2 `timescale 1ns / 1ps
3
4 /* Instruction splitter
5 /* Opcode 31:26
6 /* num1 25:21
7 /* num2 20:16
8 /* num3 15: 0
9
10 module ins_split (input wire ir_write,
11                  input wire [`REG_BUS] ins,
12                  output reg [`OP_BUS] opcode,
13                  output reg [`RS_BUS] num1,
14                  output reg [`RT_BUS] num2,
15                  output reg [`NUM3_BUS] num3,
16                  output reg [`RD_BUS] num4,
17                  output reg [`NUM5_BUS] num5,
18                  output reg [`NUM6_BUS] num6,
19                  output wire [`REG_BUS] irrun);
20
21 reg[`REG_BUS] ir;
22 always @* begin
23     if (ir_write) ir = ins;
24     opcode        = {ir[`OPCODE_BUS]};
25     num1          = {ir[`NUM1_BUS]};
26     num2          = {ir[`NUM2_BUS]};
27     num3          = {ir[`NUM3_BUS]};
28     num4          = {ir[`NUM4_BUS]};
29     num5          = {ir[`NUM5_BUS]};
30     num6          = {ir[`NUM6_BUS]};
31 end

```

```

32
33     assign irrun = ir;
34 endmodule

```

Listing 11: ins\_split.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module mdr (input wire memread,
5              input wire [`REG_BUS] memdata,
6              output reg  [`REG_BUS] memout);
7
8      reg [`REG_BUS] mem;
9
10     initial begin
11         mem = 0;
12     end
13
14
15     always @* begin
16         if (memread)
17             mem = memdata;
18
19         memout = mem;
20     end
21
22 endmodule

```

Listing 12: mdr.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module mem (input wire clk,
5              input wire memwrite,
6              input wire [`REG_BUS] indata,
7              input wire [`REG_BUS] addr,
8              output wire [`REG_BUS] memdata,
9              input wire [`REG_BUS] dpra,
10             output wire [`REG_BUS] dpo);
11
12     dist_mem_gen_0 memorys (.a(addr), .d(indata), .clk(clk), .we(memwrite), .
13                             spo(memdata), .dpra(dpra), .dpo(dpo));
14 endmodule

```

Listing 13: mem.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3

```



```

4 module mux2(input wire select,
5             input wire [`REG_BUS] route1,
6             input wire [`REG_BUS] route2,
7             output reg  [`REG_BUS] outdata);
8
9     always @*
10    if (select)
11        outdata = route1;
12    else
13        outdata = route2;
14
15 endmodule

```

Listing 14: mux2.v

```

1 `include "define.v"
2 `timescale 1ns / 1ps
3
4 module opj_extend (input wire [`NUM5_BUS] ins,
5                   input wire [`REG_BUS] pc,
6                   output wire [`REG_BUS] extd);
7
8 assign extd = (ins << 2) | ((pc >> 28) << 28);
9
10 endmodule

```

Listing 15: opj\_extend.v

```

1 `include "define.v"
2 `timescale 1ns / 1ps
3
4 /* 00. normal
5 /*    PC = PC + 4
6 /* 01. special case 1 - bne & beq
7 /*    PC = immpc + 4
8 /* 10. special case 2 - j
9 /*    PC = jump address
10
11 module pc_reg (input wire clk,
12               input wire rst,           // * clk & reset
13               input wire pccond,
14               input wire aluinfo,
15               input wire pc_write,
16               input wire [1:0] pcsource,
17               input wire [`NUM3_BUS] immpc,
18               input wire [`REG_BUS] address,
19               output reg  [`REG_BUS] pcout);
20
21 reg [`REG_BUS] pc;
22
23 initial begin
24     pc = 0;

```

```

25     end
26
27     always @(posedge clk or posedge rst) begin
28         pcout = pc;
29         if (rst) begin
30             pc <= 0;
31         end
32         else
33         begin
34             if (pc_write) begin
35                 case (pcsource)
36                     2'b00: pc <= pc + 4;
37                     2'b01:
38                     begin
39                         // * 2 cases: pccond = 1 (beq) & iszero (from cmp = 1) = 0
40                         // *      or pccond = 0 (bne) & iszero (from cmp = 0) = 1
41                         if (pccond ^ aluinfo)
42                             pc <= pc;
43                         else
44                             pc <= pc + (immpc << 2);
45                     end
46                     2'b10: pc <= address;
47                     default :pc <= pc;
48                 endcase
49             end
50         end
51
52
53
54     end
55 endmodule

```

Listing 16: pc\_reg.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module regfile(input wire clk,
5                 input wire [`RS_BUS] rs,
6                 input wire [`RT_BUS] rt,
7                 input wire [`RD_BUS] rd,
8                 input wire reg_write,
9                 input wire regdst,
10                 input wire [`REG_BUS] indata,
11                 output reg [`REG_BUS] numa,
12                 output reg [`REG_BUS] numb,
13                 input wire [`REG_BUS] regaddr,
14                 output reg [`REG_BUS] reg_out);
15
16     reg [`REG_BUS] register [0:31];
17     integer i;
18

```

```

19  initial begin
20      for (i = 0; i < 32 ; i = i + 1) begin
21          register[i] = 0;
22      end
23  end
24
25  always @(posedge clk) begin
26      if (reg_write) begin
27          if (regdst) begin
28              register[rd] = indata;
29          end else begin
30              register[rt] = indata;
31          end
32      end
33
34      numa = register [rs];
35      numb = register [rt];
36  end
37  always@* reg_out = register [(regaddr % 31)];
38 endmodule

```

Listing 17: regfile.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module segout (input wire clk,
5                 input wire [`REG_BUS] indata,
6                 output reg [7:0] an,
7                 output reg [6:0] seg);
8
9      reg [3:0] Q [7:0];
10     reg [3:0] num;
11     wire [6:0] seg0[7:0];
12     reg [31:0] count;
13     reg clk1;
14
15     BCD27 B0(Q[0],seg0[0]);
16     BCD27 B1(Q[1],seg0[1]);
17     BCD27 B2(Q[2],seg0[2]);
18     BCD27 B3(Q[3],seg0[3]);
19     BCD27 B4(Q[4],seg0[4]);
20     BCD27 B5(Q[5],seg0[5]);
21     BCD27 B6(Q[6],seg0[6]);
22     BCD27 B7(Q[7],seg0[7]);
23
24     initial
25     begin
26         an    <= 8'b11111111;
27         clk1  <= 0;
28         num   <= 0;
29         count <= 0;

```

```

30     end
31
32     always @*
33     begin
34         Q[7] <= {indata[31:28]}; //(indata & 32'
35             b1111_0000_0000_0000_0000_0000_0000) >> 28;
36         Q[6] <= {indata[27:24]}; // (indata & 32'
37             b0000_1111_0000_0000_0000_0000_0000) >> 24;
38         Q[5] <= {indata[23:20]}; // (indata & 32'
39             b0000_0000_1111_0000_0000_0000_0000) >> 20;
40         Q[4] <= {indata[19:16]}; // (indata & 32'
41             b0000_0000_0000_1111_0000_0000_0000) >> 16;
42         Q[3] <= {indata[15:12]}; // (indata & 32'
43             b0000_0000_0000_0000_1111_0000_0000) >> 12;
44         Q[2] <= {indata[11: 8]}; // (indata & 32'
45             b0000_0000_0000_0000_0000_1111_0000_0000) >> 8;
46         Q[1] <= {indata[7: 4]}; // (indata & 32'
47             b0000_0000_0000_0000_0000_0000_1111_0000) >> 4;
48         Q[0] <= {indata[3: 0]}; // (indata & 32'
49             b0000_0000_0000_0000_0000_0000_0000_1111);
50     end
51
52     always@(posedge clk)
53         if (count >= 31'd19999)
54             begin
55                 clk1 <= ~clk1;
56                 count = 31'b0;
57                 num <= (num+1)%8;
58             end
59         else
60             count = count+1;
61
62     always@*
63     case(num)
64         0:begin
65             an <= 8'b11111110;
66             seg <= seg0[0];
67         end
68         1:begin
69             an <= 8'b11111101;
70             seg <= seg0[1];
71         end
72         2:begin
73             an <= 8'b11111011;
74             seg <= seg0[2];
75         end
76         3:begin
77             an <= 8'b11110111;
78             seg <= seg0[3];
79         end
80         4:begin

```

```

73         an <= 8'b11101111;
74         seg <= seg0[4];
75     end
76     5:begin
77         an <= 8'b11011111;
78         seg <= seg0[5];
79     end
80     6:begin
81         an <= 8'b10111111;
82         seg <= seg0[6];
83     end
84     7:begin
85         an <= 8'b01111111;
86         seg <= seg0[7];
87     end
88 endcase
89 endmodule

```

Listing 18: segout.v

```

1 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets inc_IBUF];
2 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets dec_IBUF];
3 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets rst_IBUF];
4
5 ## Clock signal
6 set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports {
    clk_board100 }]; #IO_L12P_T1_MRCC_35 Sch=clk
7 create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {
    clk_board100}];
8
9
10 ##Switches
11 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { cont
    }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
12 set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { inc }];
    #IO_L24P_T3_35 Sch=sw[12]
13 set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { dec
    }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
14 set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { step
    }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
15 set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { mem
    }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
16
17
18 ## LEDs
19 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports {
    add rout_8[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
20 set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports {
    add rout_8[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
21 set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports {
    add rout_8[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
22 set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports {

```

```

    addROUT_8[3] }); #IO_L8P_T1_D11_14 Sch=led[3]
23 set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports {
    addROUT_8[4] }); #IO_L7P_T1_D09_14 Sch=led[4]
24 set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports {
    addROUT_8[5] }); #IO_L18N_T2_A11_D27_14 Sch=led[5]
25 set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports {
    addROUT_8[6] }); #IO_L17P_T2_A14_D30_14 Sch=led[6]
26 set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports {
    addROUT_8[7] }); #IO_L18P_T2_A12_D28_14 Sch=led[7]
27 set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { pc_8
    [0] }); #IO_L16N_T2_A15_D31_14 Sch=led[8]
28 set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { pc_8
    [1] }); #IO_L14N_T2_SRCC_14 Sch=led[9]
29 set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { pc_8
    [2] }); #IO_L22P_T3_A05_D21_14 Sch=led[10]
30 set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { pc_8
    [3] }); #IO_L15N_T2_DQS_DOUT_CS0_B_14 Sch=led[11]
31 set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { pc_8
    [4] }); #IO_L16P_T2_CSI_B_14 Sch=led[12]
32 set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { pc_8
    [5] }); #IO_L22N_T3_A04_D20_14 Sch=led[13]
33 set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { pc_8
    [6] }); #IO_L20N_T3_A07_D23_14 Sch=led[14]
34 set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { pc_8
    [7] }); #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]
35
36
37 ##7 segment display
38 set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 } [get_ports { seg[0]
    }]; #IO_L24N_T3_A00_D16_14 Sch=ca
39 set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 } [get_ports { seg[1]
    }]; #IO_25_14 Sch=cb
40 set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 } [get_ports { seg[2]
    }]; #IO_25_15 Sch=cc
41 set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 } [get_ports { seg[3]
    }]; #IO_L17P_T2_A26_15 Sch=cd
42 set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 } [get_ports { seg[4]
    }]; #IO_L13P_T2_MRCC_14 Sch=ce
43 set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 } [get_ports { seg[5]
    }]; #IO_L19P_T3_A10_D26_14 Sch=cf
44 set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 } [get_ports { seg[6]
    }]; #IO_L4P_T0_D04_14 Sch=cg
45
46 set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 } [get_ports { an[0]
    }]; #IO_L23P_T3_F0E_B_15 Sch=an[0]
47 set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 } [get_ports { an[1]
    }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
48 set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 } [get_ports { an[2]
    }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
49 set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 } [get_ports { an[3]
    }]; #IO_L19P_T3_A22_15 Sch=an[3]

```

```

50 set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { an[4]
    }]; #IO_L8N_T1_D12_14 Sch=an[4]
51 set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 } [get_ports { an[5]
    }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
52 set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 } [get_ports { an[6]
    }]; #IO_L23P_T3_35 Sch=an[6]
53 set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports { an[7]
    }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
54
55
56 ##Buttons
57 set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports { rst
    }]; #IO_L9P_T1_DQS_14 Sch=btnc

```

Listing 19: constraints.xdc

## 6.2 模拟版本代码

模拟版本代码中 cpu 和 ddu 以及一些模块有所不同，添加了更多的 wire output 取显示更多的调试信息。

下面仅贴出不同部分的代码，其余相同部分不再重复复制。

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module cpu (input wire clk,
5              input wire rst,
6              output [`REG_BUS] pcout,
7              input  [`REG_BUS] inputaddr,
8              output [`REG_BUS] memout_show,
9              output [`REG_BUS] regout_show,
10             output wire pc_write,
11             output wire [`STATE_BUS] state,
12             output wire [`REG_BUS] numatest,
13             output wire [`REG_BUS] numbtest,
14             output wire [`REG_BUS] irrunc);
15
16     wire [`NUM1_BUS] num1;
17     wire [`NUM2_BUS] num2;
18     wire [`NUM3_BUS] num3;
19     wire [`NUM4_BUS] num4;
20     wire [`NUM5_BUS] num5;
21     wire [`NUM6_BUS] num6;
22
23     wire [`REG_BUS] jext;
24     wire [`REG_BUS] aluout;
25     wire [`REG_BUS] numa;
26     wire [`REG_BUS] numb;
27     wire [`REG_BUS] alu_numa;
28     wire [`REG_BUS] alu_numb;

```

```

29  wire [`REG_BUS] memaddr;
30  wire [`REG_BUS] memmdr;
31  wire [`REG_BUS] regmem;
32  wire [`REG_BUS] immext;
33  wire [`REG_BUS] memdata;
34
35  wire [`EXE_BUS] aluexe;
36  wire [`OP_BUS] opcode;
37
38  wire ins_or_data;
39  wire pc_write_cond;
40  wire pc_write;
41  wire alu_srca;
42  wire alu_srcb;
43  wire ir_write;
44  wire mem_write;
45  wire mem_read;
46  wire is_zero;
47  wire mem_to_reg;
48  wire reg_write;
49  wire regdst;
50  wire [1:0] pc_source;
51  wire [1:0] alu_op;
52  wire [`REG_BUS] irrund;
53  wire [`STATE_BUS] state;
54
55  opj_extend jexter (.ins (num5), .pc (pcout), .extd (jext));
56  pc_reg pc (.clk (clk), .rst (rst), .pccond (pc_write_cond), .pcsource (
    pc_source), .aluinfo(is_zero), .pc_write(pc_write), .immpc (num3), .
    address(jext), .pcout (pcout));
57  mux2 mux_ins_or_data (.select (ins_or_data), .route1 (aluout >> 2), .route2
    (pcout >> 2), .outdata (memaddr));
58  mem memory (.clk(clk), .memwrite (mem_write), .indata (numb), .addr (
    memaddr), .memdata (memdata), .dpra (inputaddr), .dpo (memout_show));
59  ins_split ir (.ir_write (ir_write), .ins (memdata), .opcode (opcode), .num1
    (num1), .num2 (num2), .num3 (num3), .num4 (num4), .num5 (num5), .num6
    (num6), .irrund (irrund));
60  mdr memory_data_reg (.memread (mem_read), .memdata (memdata), .memout (
    memmdr));
61  control ctrl (.clk (clk), .rst (rst), .op (opcode), .pc_write_cond (
    pc_write_cond), .pc_write (pc_write), .ins_or_data (ins_or_data), .
    mem_read(mem_read), .mem_write (mem_write), .ir_write(ir_write), .
    pc_source(pc_source), .alu_op (alu_op), .alu_srca (alu_srca), .alu_srcb
    (alu_srcb), .reg_write (reg_write), .regdst (regdst), .mem_to_reg (
    mem_to_reg), .state_out (state));
62  mux2 reg_write_mem (.select (mem_to_reg), .route1(memmdr), .route2(aluout),
    .outdata (regmem));
63  regfile regs (.clk (clk), .rs (num1), .rt (num2), .rd (num4), .numa (numa),
    .numb (numb), .reg_write (reg_write), .regdst (regdst), .indata (
    regmem), .regaddr (inputaddr), .reg_out(regout_show));// , .reg_clone(
    regshow));

```



```

64     mux2 muxnuma (.select (alu_srca), .route1 (pcout), .route2 (numa) , .
        outdata (alu_numa));
65     extend extder (.ins (num3), .extd (immext));
66     mux2 muxnumb (.select (alu_srcb), .route1 (immext), .route2(numb), .outdata
        (alu_numb));
67     alu_ctrl aluexe_maker (.opcode (opcode), .funt (num6), .alu_op (aluexe));
68     alu alucalc (.aluop (aluexe), .numa (alu_numa), .numb(alu_numb), .num_out (
        aluout) , .iszero (is_zero));
69
70     assign numatest = alu_numa;
71     assign numbtest = alu_numb;
72 endmodule

```

Listing 20: cpu.v

```

1  `timescale 1ns / 1ps
2  `include "define.v"
3
4  module ddu(input cont,
5             input rst,
6             input clk_board,
7             input step,
8             input mem,
9             input inc,
10            input dec,
11            output reg [7:0] addrout_8,
12            output wire [7:0] pc_8,
13            output pc_write,
14            output wire [`STATE_BUS] state,
15            output wire [`REG_BUS] irrund,
16            output wire [6:0] seg,
17            output wire [7:0] an,
18            output wire [1023:0] reg_clone,
19            output [`REG_BUS] numa,
20            output [`REG_BUS] numb,
21            output wire [`REG_BUS] memshow);
22
23     wire [`REG_BUS] memshow;
24     reg [`REG_BUS] addrout;
25     wire [`REG_BUS] pc;
26
27     reg [`REG_BUS] addr = 0;
28     wire [`REG_BUS] mem1;
29     wire [`REG_BUS] mem2;
30     reg clk;
31     reg delay;
32     reg incdelay;
33     reg decdelay;
34
35     always @(posedge clk_board or negedge rst)
36     begin
37         if (rst) begin

```

```

38     addr <= 0;
39     incdelay = 1;
40     decdelay = 1;
41     delay    = 0;
42     end else begin
43
44         if (!incdelay && inc) addr = addr + 1;
45         if (!decdelay && dec) addr = addr - 1;
46
47         incdelay = inc;
48         decdelay = dec;
49
50         addrout      = addr;
51         if (!delay && step) clk = 1;
52         else clk      = 0;
53
54         delay = step;
55
56     end
57 end
58 assign memshow = mem ? mem1 : mem2;
59
60 cpu CPU_sim (.clk (clk), .rst (rst), .pcout (pc), .inputaddr(addr) , .
    memout_show (mem1), .regout_show(mem2) , .regshow(reg_clone)
61 , .state(state) , .numatest(numa) , .numbtest(numb), .pc_write(pc_write) , .
    irrund (irrund));
62
63 always @* addrout_8 = {addr [7:0]};
64
65 assign pc_8 = {pc[9:2]};
66
67 segout segmental (.clk (clk_board), .indata (memshow), .an(an) , .seg(seg));
68 endmodule

```

Listing 21: ddu.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  module regfile(input wire clk,
5                 input wire [`RS_BUS] rs,
6                 input wire [`RT_BUS] rt,
7                 input wire [`RD_BUS] rd,
8                 input wire reg_write,
9                 input wire regdst,
10                 input wire [`REG_BUS] indata,
11                 output reg [`REG_BUS] numa,
12                 output reg [`REG_BUS] numb,
13                 input wire [`REG_BUS] regaddr,
14                 output reg [`REG_BUS] reg_out,
15                 output reg [1023:0] reg_clone);
16

```

```

17  reg [`REG_BUS] register [0:31];
18  integer i;
19
20  initial begin
21      for (i = 0; i < 32 ; i = i + 1) begin
22          register[i] = 0;
23      end
24  end
25
26  always @* begin
27      reg_clone = 0;
28      for (i = 31; i >= 0 ; i = i - 1) begin
29          reg_clone = (reg_clone << 32) | register[i];
30      end
31  end
32
33  always @(posedge clk) begin
34      if (reg_write) begin
35          if (regdst) begin
36              register[rd] = indata;
37          end else begin
38              register[rt] = indata;
39          end
40      end
41
42      numa = register [rs];
43      numb = register [rt];
44  end
45  always@* reg_out = register [(regaddr % 31)];
46  endmodule

```

Listing 22: regfile.v

```

1  `include "define.v"
2  `timescale 1ns / 1ps
3
4  /* Controller
5  /* input with op
6  /* output control information
7
8  module control (input wire clk,
9                  input wire rst,
10                 input wire [`OP_BUS] op,           // * input the opcode part
11                 output reg pc_write_cond,          // * control pc
12                 output reg pc_write,               // * control pc
13                 output reg ins_or_data,            // * control mem
14                 output reg mem_read,               // * control mem
15                 output reg mem_write,             // * control mem
16                 output reg ir_write,               // * control ir
17                 output reg [1:0] pc_source,        // * control pc source
18                 output reg [1:0] alu_op,           // * control alu
19                 output reg alu_srca,               // * control numa

```

```

20         output reg alu_srcb,           // * control numb
21         output reg reg_write,         // * control regfile writable
22         output reg regdst,
23         output reg mem_to_reg,
24         output reg [`STATE_BUS] state_out);
25
26 reg [`STATE_BUS] state,next_state;
27
28 initial begin
29     state      = `IF_STATE;
30     pc_write_cond = 0;
31     pc_write    = 0;
32     ins_or_data = 0;
33     mem_read    = 0;
34     mem_write   = 0;
35     ir_write    = 0;
36     pc_source   = 0;
37     alu_op      = 0;
38     alu_srca    = 0;
39     alu_srcb    = 0;
40     reg_write   = 0;
41     regdst      = 0;
42 end
43
44 always @*
45 state_out = state;
46
47 always @(posedge clk or posedge rst) begin
48     if (rst) begin
49         state <= `IF_STATE;
50     end else begin
51         state <= next_state;
52     end
53 end
54
55 always @* begin
56     case (state)
57         `IF_STATE :next_state = `ID_STATE;
58         `ID_STATE :begin
59             case (op)
60                 `OP_LW : next_state = `EXE1_STATE;
61                 `OP_SW : next_state = `EXE1_STATE;
62                 `OP_BEQ :next_state = `EXE3_STATE;
63                 `OP_BNE :next_state = `EXE3_STATE;
64                 `OP_J  : next_state = `EXE4_STATE;
65                 default :next_state = `EXE2_STATE;
66             endcase
67         end
68         `EXE1_STATE :begin
69             if (op == `OP_LW)
70                 next_state = `MEM1_STATE;

```

```

71         else
72             next_state = `MEM2_STATE;
73         end
74         `EXE2_STATE :next_state = `WB2_STATE;
75         `EXE3_STATE :next_state = `IF_STATE;
76         `EXE4_STATE :next_state = `IF_STATE;
77         `MEM1_STATE :next_state = `WB1_STATE;
78         `MEM2_STATE :next_state = `IF_STATE;
79         `WB1_STATE :next_state = `IF_STATE;
80         `WB2_STATE :next_state = `IF_STATE;
81     endcase
82 end
83
84 always @* begin
85     // * if state == > initial state
86     if (state == `IF_STATE) begin
87         pc_write_cond = 0;
88         pc_write      = 1; // * pc writable
89         ins_or_data = 0;
90         mem_read      = 1; // * read mem for instructions
91         mem_write     = 0;
92         ir_write      = 1; // * ir writable
93         pc_source     = 2'b00;
94         alu_op        = 0;
95         alu_srca      = 0;
96         alu_srcb      = 0;
97         reg_write     = 0;
98         regdst        = 0;
99     end
100
101     // * id state
102     if (state == `ID_STATE) begin
103         pc_write = 0;
104         ir_write = 0;
105         mem_read = 0;
106     end
107
108     // * exe1 state
109     if (state == `EXE1_STATE) begin
110         alu_srca = 0;
111         alu_srcb = 1;
112         alu_op   = 2'b00;
113     end
114
115     // * exe2 state
116     if (state == `EXE2_STATE) begin
117         alu_srca = 0;
118         alu_srcb = (op == `OP_R_TYPE ? 0 : 1);
119         alu_op   = 2'b10;
120     end
121

```

```

122 // * exe3 state
123 if (state == `EXE3_STATE) begin
124     alu_srca    = 1;
125     alu_srcb    = 0;
126     alu_op      = 2'b01;
127     pc_write    = 1;
128     pc_write_cond = (op == `OP_BEQ ? 1 :0);
129     pc_source    = 2'b01;
130 end
131
132 // * exe4 state
133 if (state == `EXE4_STATE) begin
134     pc_write = 1;
135     pc_source = 2'b10;
136 end
137
138 // * mem1 state
139 if (state == `MEM1_STATE) begin
140     mem_read  = 1;
141     ins_or_data = 1;
142 end
143
144 // * mem2 state
145 if (state == `MEM2_STATE) begin
146     mem_write = 1;
147     ins_or_data = 1;
148 end
149
150 // * wb1 state
151 if (state == `WB1_STATE) begin
152     regdst    = 0;
153     reg_write = 1;
154     mem_to_reg = 1;
155 end
156
157 // * wb2 state
158 if (state == `WB2_STATE) begin
159     regdst    = (op == `OP_R_TYPE ? 1 :0);
160     reg_write = 1;
161     mem_to_reg = 0;
162 end
163 end
164 endmodule

```

Listing 23: control.v

```

1 `timescale 1ns / 1ps
2 `include "define.v"
3
4 module cpu_sim1();
5
6     reg    rst    ;

```

```

7  reg    clk    ;
8  reg [`REG_BUS] dpra;
9  wire  [`REG_BUS] dpo;
10 wire  [`REG_BUS] pcout;
11 wire  [`REG_BUS] regdata;
12
13 reg cont;
14 reg step;
15 reg mem;
16 reg inc;
17 reg dec;
18
19 wire [`REG_BUS] pc;
20 wire clkout;
21 wire [`REG_BUS] addrout;
22 wire [`REG_BUS] memdata;
23 wire delay1;
24 wire delay2;
25 wire [`NUM5_BUS] num5;
26 wire [`REG_BUS] irrun;
27 wire [`REG_BUS] numa;
28 wire [`REG_BUS] numb;
29 initial
30 begin
31     rst <= 0;
32     cont <= 0;
33     inc <= 0;
34     dec <= 0;
35     mem <= 0;
36     step <= 0;
37     #10 rst = 1;
38     #10 rst = 0;
39     #20 step = 1;
40     #10 step = 0;
41     #10 mem = 1;
42     inc      = 1;
43     #10 inc = 0;
44     #10 inc = 1;
45     #10 inc = 0;
46     #10 inc = 1;
47     #10 inc = 0;
48     #10 inc = 1;
49     #10 inc = 0;
50     #10 inc = 1;
51     #10 inc = 0;
52     #10 inc = 1;
53     #10 inc = 0;
54     #10 inc = 1;
55     #10 inc = 0;
56     #10 inc = 1;
57     #10 inc = 0;

```

```

58     #10 step = 1;
59     #10 step = 0;
60     #10 step = 1;
61     #10 step = 0;
62     #10 step = 1;
63     #10 step = 0;
64     #10 step = 1;
65     #10 step = 0;
66     #10 step = 1;
67     #10 step = 0;
68     #10 step = 1;
69     #10 step = 0;
70     #10 step = 1;
71     #10 step = 0;
72     #10 step = 1;
73     #10 step = 0;
74     #10 step = 1;
75     #10 step = 0;
76     #10 step = 1;
77     #10 step = 0;
78     #10 step = 1;
79     #10 step = 0;
80     #10 step = 1;
81     #10 step = 0;
82     #10 step = 1;
83     #10 step = 0;
84     #10 step = 1;
85     #10 step = 0;
86     #10 step = 1;
87     #10 step = 0;
88     #10 step = 1;
89     #10 step = 0;
90     #10 step = 1;
91     #10 step = 0;
92     #10 mem = 0;
93     #10000;
94 end
95
96 initial clk <= 0;
97 always #5 clk <= ~clk;
98 always begin #10 step = 1; #20 step = 0; end
99
100 wire pc_write;
101 wire [`STATE_BUS] state;
102
103 wire [7:0] addrout_8;
104 wire [7:0] pc_8;
105
106 wire [6:0] seg;
107 wire [7:0] an;
108 wire [1023:0] reg_clone;

```



```

109     wire reg_write;
110     wire [31:0] memmdr;
111     wire [31:0] mem2;
112
113     ddu DUT (cont,rst,clk,step,mem,inc,dec,addrout_8,pc_8,pc_write,state,irrun,
        seg,an,mem2,numa,numb,regclone);
114 endmodule

```

Listing 24: cpu\_sim1.v

## 7 总结

本次实验相当于一次比较综合的复习，复习了我们学习 COD 至多周期的各种知识。本次实验对于我来说最大的困难就是如何将程序正确的烧写到板子上。事实上，我只用了 3 天就完成了正确的模拟和仿真，但是到最终完成花费了整整 3 周的时间。具体原因在于对于时序逻辑和组合逻辑的写法规范性。

举个例子，再我 extend 的模块中，本来应该用 wire 就可以完成的任务，就不必多加一个 reg 进行中介保存，虽然理论上有着一样的效果，但是实际上，这样的代码过多会倒置无意义的锁存现象出现，从而出现各种诡异的 bug 出现。此外另一个困扰我整整一周的实验 bug 是板载时钟周期过短，无法再一个时钟周期内完成应该执行完成的任务，出现了神奇的 bug。

最终，实验也再截止日期前完成，总体来说还算顺利。

## 8 意见和建议

本次实验个人感觉，难度并不是很大，但是再我们烧写板子的时候，比较考研问题分析解决能力和动手能力。个人建议，之后实验助教要求检查的只有我们的仿真模拟，至于烧录可以考虑作为附加分处理。

此外，我决定我们之后可以增加一次关于流水实现的实验，也是只要求模拟，不要求仿真。考虑流水的复杂性和代码量，个人认为，设计流水 CPU 再多周期之后，并且要求同学们合作完成。这样可以达到更好的效果。

最后表示，老师还是最好不要删除单周期的实验，第一在于……通知太晚，我都写完了才说取消实验。第二在于，单周期作为复习也是很有必要的。而且，如果单周期的实验吐过顺利完成，之后的多周期会少很多问题，例如办在诗中的处理和锁存器等等。