UM–SJTU JOINT INSTITUTE

INTRODUCTION TO COMPUTER ORGANIZATION
(VE370)

PROJECT 2 INDIVIDUAL REPORT

Zhan Yuhong ID: 518021910565

Date: November 12, 2020

1 Introduction

In Project 2, we are required to model a single cycle processor in Verilog HDL. I use the MIPS assembly program provided by TAs to verify that the processor can execute those instructions continuously and correctly.

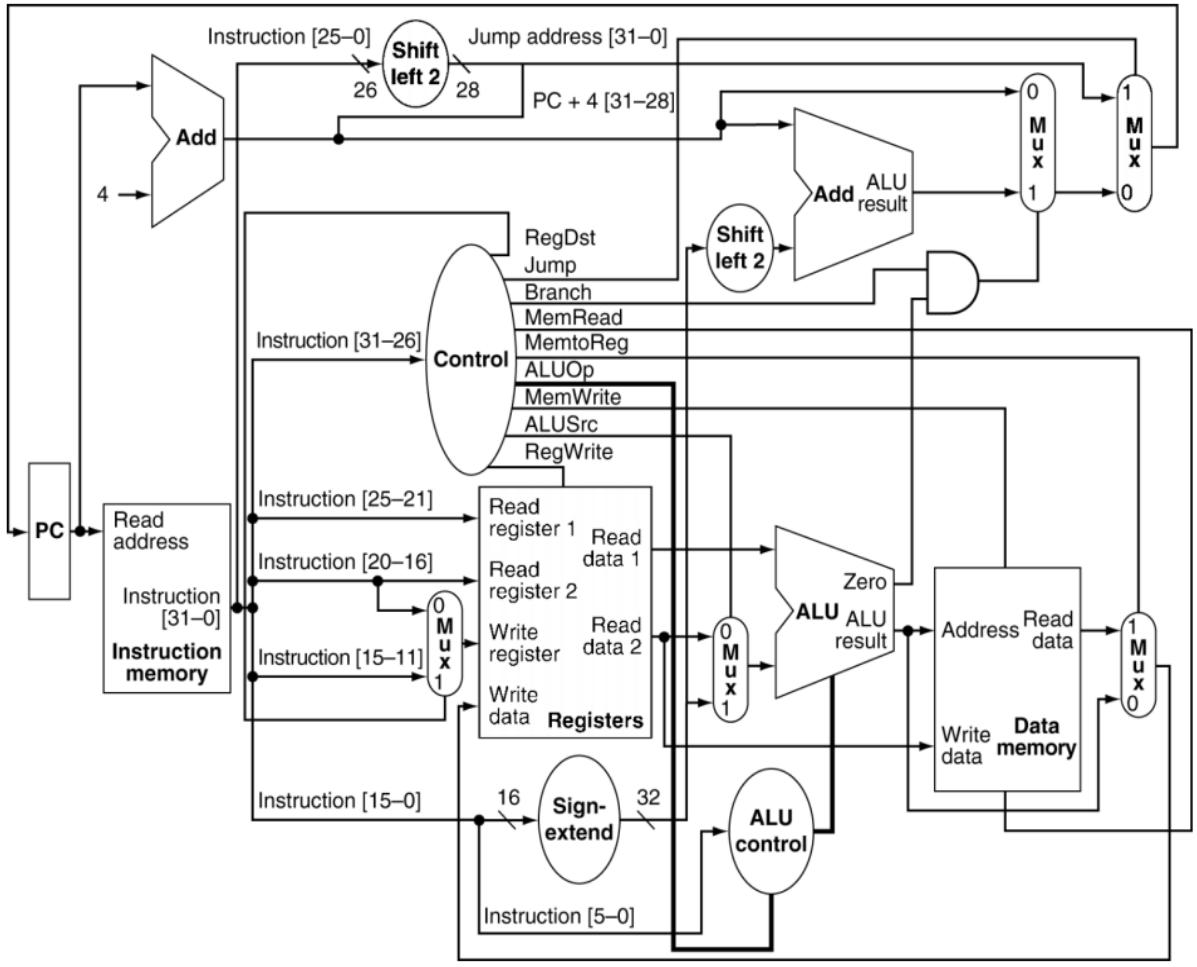

Figure 1: Single cycle implementation of MIPS architecture[1]

The graph above is the single cycle implementation of MIPS architecture, but there are some extra MIPS instructions added such as ”andi” ”bne”.

2 Screen shots of simulation results for each type of instruction

2.1 addi $t0, $zero, 0x20

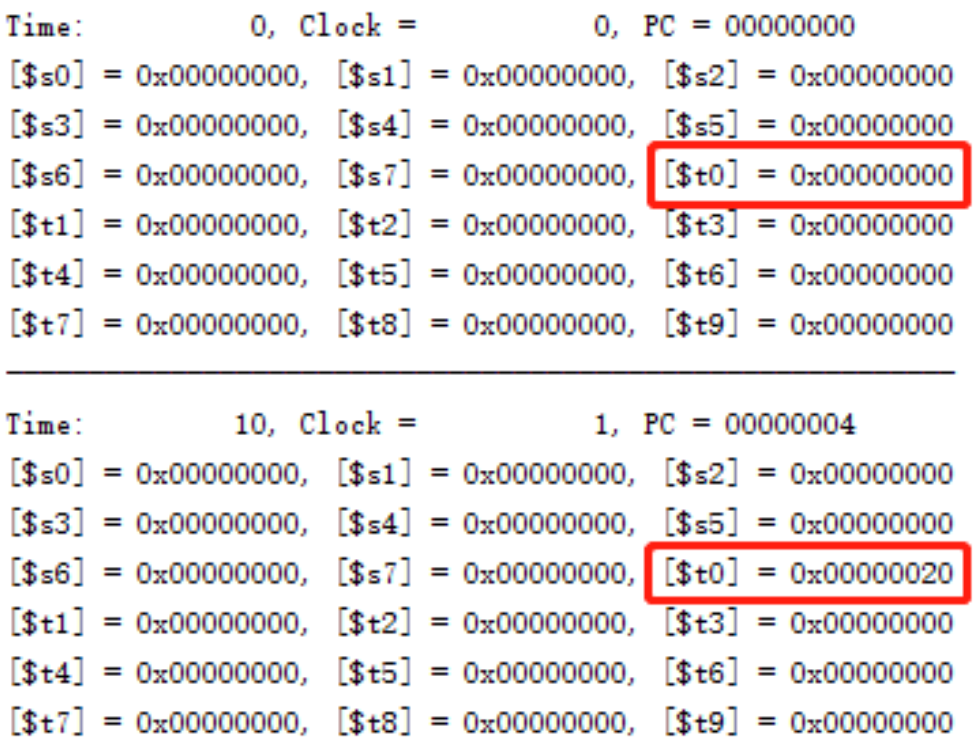

Figure 2: addi

addi $t0, $zero, 0x20

2.2 and $s0, $t0, $t1

```
Time:          20,  Clock =           2,  PC = 00000008
[$s0] = 0x00000000,  [$s1] = 0x00000000,  [$s2] = 0x00000000
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
--------------------------------------------------------------
Time:          30,  Clock =           3,  PC = 0000000c
[$s0] = 0x00000020,  [$s1] = 0x00000000,  [$s2] = 0x00000000
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
```

Figure 3: and

and $s0, $t0, $t1

2.3 or $s0, $t0, $t1

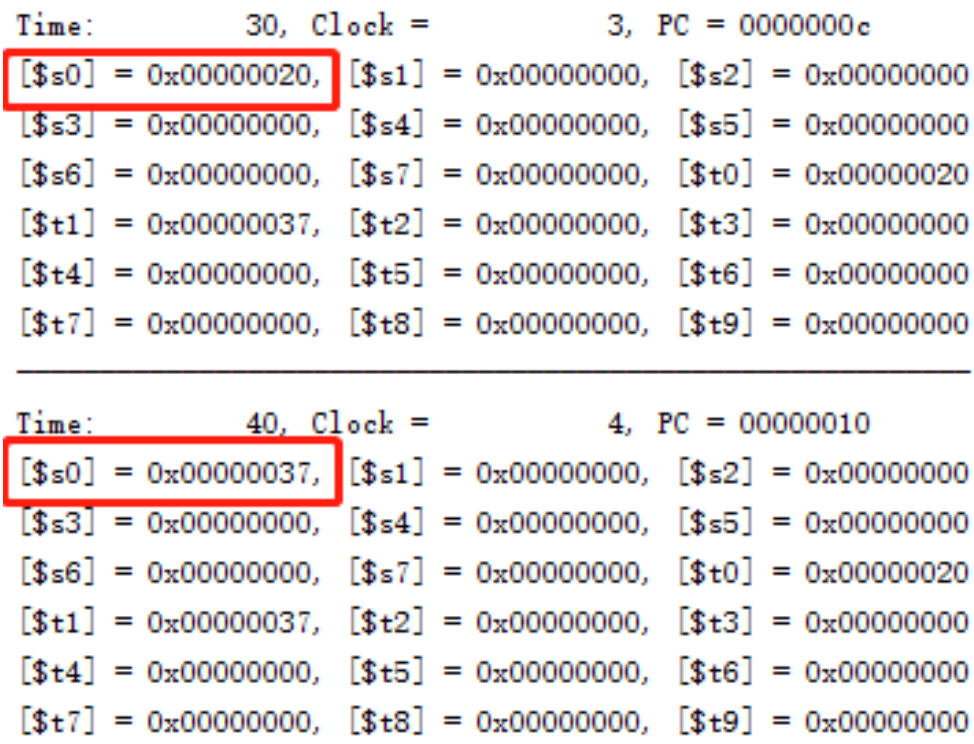

Figure 4: or

or $s0, $t0, $t1

2.4 sw $s0, 4($zero)

```
Time:          40, Clock =           4, PC = 00000010
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
------------------------------------------------------------
Time:          50, Clock =           5, PC = 00000014
[$s0] = 0x00000037, [$s1] = 0x00000000, [$s2] = 0x00000000
[$s3] = 0x00000000, [$s4] = 0x00000000, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

Figure 5: sw

sw $s0, 4($zero)

Since I don't display the content of data memory, the result of sw instruction should be showed in the following lw part.

2.5 lw $s1, 4($zero)

```
Time:          90,  Clock =           9,  PC = 00000024
[$s0] = 0x00000037,  [$s1] = 0x00000057,  [$s2] = 0xfffffff9
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
--------------------------------------------------------------
Time:         100,  Clock =          10,  PC = 00000028
[$s0] = 0x00000037,  [$s1] = 0x00000037,  [$s2] = 0xfffffff9
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
```

Figure 6: lw

lw $s1, 4($zero)

2.6 add $s1, $t0, $t1

```
Time:          60,  Clock =           6,  PC = 00000018
[$s0] = 0x00000037,  [$s1] = 0x00000000,  [$s2] = 0x00000000
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
-------------------------------------------------------------
Time:          70,  Clock =           7,  PC = 0000001c
[$s0] = 0x00000037,  [$s1] = 0x00000057,  [$s2] = 0x00000000
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
```

Figure 7: add

add $s1, $t0, $t1

2.7 sub $s2, $t0, $t1

```
Time:          70,  Clock =           7,  PC = 0000001c
[$s0] = 0x00000037,  [$s1] = 0x00000057,  [$s2] = 0x00000000
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
-------------------------------------------------------------
Time:          80,  Clock =           8,  PC = 00000020
[$s0] = 0x00000037,  [$s1] = 0x00000057,  [$s2] = 0xfffffff9
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
```

Figure 8: sub

sub $s2, $t0, $t1

2.8 beq $s1, $s2, error0 (not branch)

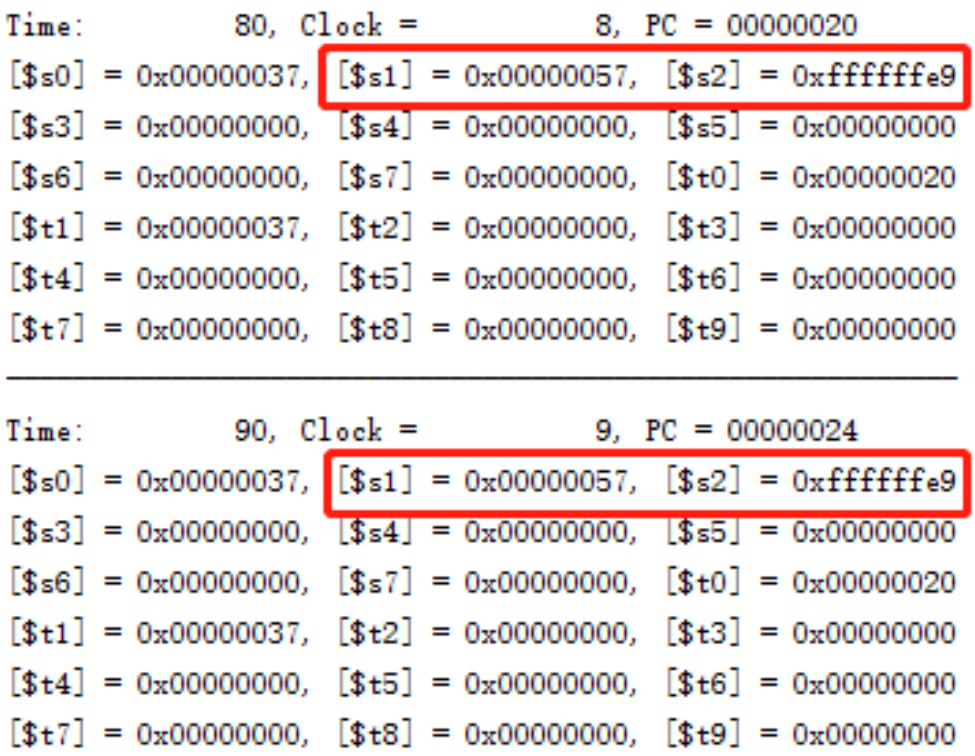

Figure 9: beq (not branch)

beq $s1, $s2, error0

2.9 andi $s2, $s1, 0x48

```
Time:         100,  Clock =           10,  PC = 00000028
[$s0] = 0x00000037,  [$s1] = 0x00000037,  [$s2] = 0xfffffff9
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
--------------------------------------------------------------
Time:         110,  Clock =           11,  PC = 0000002c
[$s0] = 0x00000037,  [$s1] = 0x00000037,  [$s2] = 0x00000000
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
```

Figure 10: andi

andi $s2, $s1, 0x48

2.10 slt $s4, $s2, $s1 (Last)

```
Time:         140,  Clock =           14,  PC = 00000038
[$s0] = 0x00000037,  [$s1] = 0x00000037,  [$s2] = 0x00000000
[$s3] = 0x00000020,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
--------------------------------------------------------------
Time:         150,  Clock =           15,  PC = 0000003c
[$s0] = 0x00000037,  [$s1] = 0x00000037,  [$s2] = 0x00000000
[$s3] = 0x00000020,  [$s4] = 0x00000001,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
```

Figure 11: slt

slt $s4, $s2, $s1 (Last)

2.11 j Last

```
Time:        170, Clock =          17, PC = 00000044
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
--------------------------------------------------------------------------------
Time:        180, Clock =          18, PC = 00000038
[$s0] = 0x00000037, [$s1] = 0x00000037, [$s2] = 0x00000037
[$s3] = 0x00000020, [$s4] = 0x00000001, [$s5] = 0x00000000
[$s6] = 0x00000000, [$s7] = 0x00000000, [$t0] = 0x00000020
[$t1] = 0x00000037, [$t2] = 0x00000000, [$t3] = 0x00000000
[$t4] = 0x00000000, [$t5] = 0x00000000, [$t6] = 0x00000000
[$t7] = 0x00000000, [$t8] = 0x00000000, [$t9] = 0x00000000
```

Figure 12: j

j Last

2.12 beq $s4, $0, EXIT (branch)

```
Time:         190,  Clock =           19,  PC = 0000003c
[$s0] = 0x00000037,  [$s1] = 0x00000037,  [$s2] = 0x00000037
[$s3] = 0x00000020,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
--------------------------------------------------------------
Time:         200,  Clock =           20,  PC = 0000007c
[$s0] = 0x00000037,  [$s1] = 0x00000037,  [$s2] = 0x00000037
[$s3] = 0x00000020,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
```

Figure 13: beq (branch)

beq $s4, $0, EXIT

2.13 bne $s0,t0, EXIT (not branch)

```
Time:          30,  Clock =           3,  PC = 0000000c
[$s0] = 0x00000020,  [$s1] = 0x00000000,  [$s2] = 0x00000000
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
--------------------------------------------------------------
Time:          40,  Clock =           4,  PC = 00000010
[$s0] = 0x00000020,  [$s1] = 0x00000000,  [$s2] = 0x00000000
[$s3] = 0x00000000,  [$s4] = 0x00000000,  [$s5] = 0x00000000
[$s6] = 0x00000000,  [$s7] = 0x00000000,  [$t0] = 0x00000020
[$t1] = 0x00000037,  [$t2] = 0x00000000,  [$t3] = 0x00000000
[$t4] = 0x00000000,  [$t5] = 0x00000000,  [$t6] = 0x00000000
[$t7] = 0x00000000,  [$t8] = 0x00000000,  [$t9] = 0x00000000
```

Figure 14: bne (not branch)

bne $s0, $t0, EXIT

2.14 bne $s0,t0, EXIT (branch)

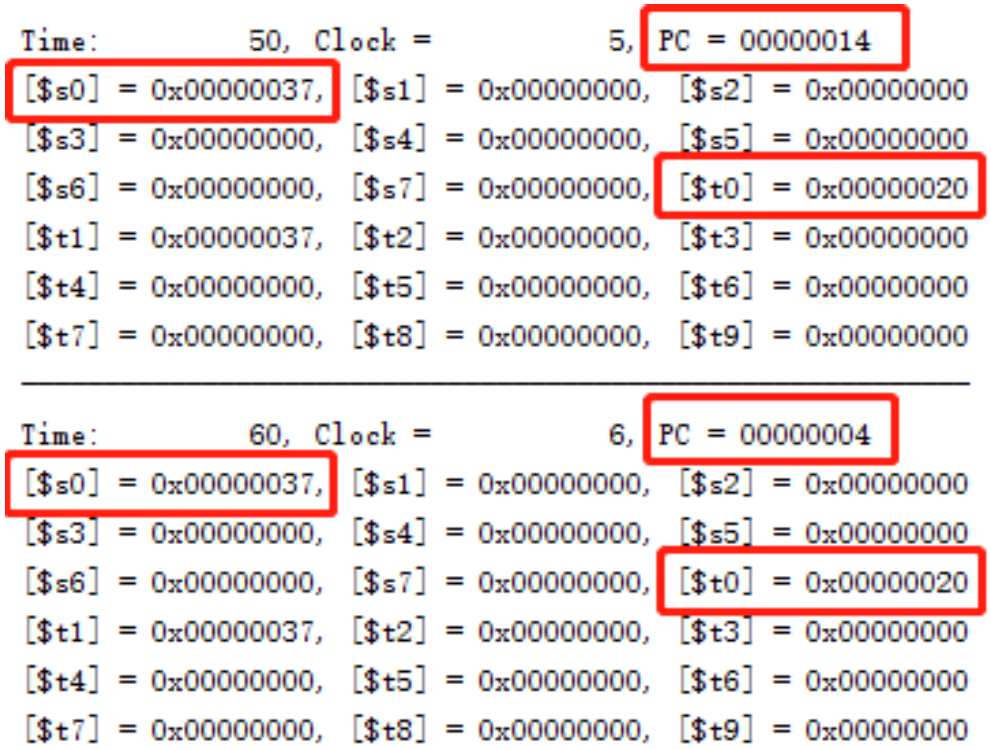

Figure 15: bne (branch)

bne $s0, $t0, EXIT

3 Testcase

3.1 single_bonus.txt (addi to beq)

```
00100000000010000000000000100000 //addi $t0, $zero, 0x20
00100000000010010000000000110111 //addi $t1, $zero, 0x37
00000001000010011000000000100100 //and $s0, $t0, $t1
00000001000010011000000000100101 //or $s0, $t0, $t1
10101100000100000000000000000100 //sw $s0, 4($zero)
10101100000010000000000000001000 //sw $t0, 8($zero)
00000001000010011000100000100000 //add $s1, $t0, $t1
00000001000010011001000000100010 //sub $s2, $t0, $t1
00010010001100100000000000001001 //beq $s1, $s2, error0
10001100000100010000000000000100 //lw $s1, 4($zero)
```

```
00110010001100100000000001001000 //andi $s2, $s1, 0x48
00010010001100100000000000001001 //beq $s1, $s2, error1
10001100000100110000000000001000 //lw $s3, 8($zero)
00010010000100110000000000001010 //beq $s0, $s3, error2
00000010010100011010000000101010 //slt $s4, $s2, $s1 (Last)
00010010100000000000000000001111 //beq $s4, $0, EXIT
00000010001000001001000000100000 //add $s2, $s1, $0
00001000000000000000000000001110 //j Last
00100000000010000000000000000000 //addi $t0, $0, 0(error0)
00100000000010010000000000000000 //addi $t1, $0, 0
00001000000000000000000000011111 //j EXIT
00100000000010000000000000000001 //addi $t0, $0, 1(error1)
00100000000010010000000000000001 //addi $t1, $0, 1
00001000000000000000000000011111 //j EXIT
00100000000010000000000000000010 //addi $t0, $0, 2(error2)
00100000000010010000000000000010 //addi $t1, $0, 2
00001000000000000000000000011111 //j EXIT
00100000000010000000000000000011 //addi $t0, $0, 3(error3)
00100000000010010000000000000011 //addi $t1, $0, 3
00001000000000000000000000011111 //j EXIT
```

3.2 single_bne.txt (bne)

```
00100000000010000000000000100000 //addi $t0, $zero, 0x20
00100000000010010000000000110111 //addi $t1, $zero, 0x37 (EXIT)
00000001000010011000000000100100 //and $s0, $t0, $t1
00010110000010001111111111111101 //bne $s0, $t0, EXIT
00000001000010011000000000100101 //or $s0, $t0, $t1
00010110000010001111111111111011 //bne $s0, $t0, EXIT
```

4 Peer evaluation

Name	Level of contribution	Description of contribution
Yuhong Zhan	5	FPGA implementation, Debug Pipelined blocks, Proofread team report
Chenfzhang Lin	5	Debug Pipelined blocks, Write team report
Ruge Xu	5	FPGA implementation, Write team report, Debug Pipelined blocks
Yipeng Lin	5	Design Pipelined blocks, Debug Pipelined blocks

Table 1: Peer evaluation

5 Reference

[1]Zheng Gang, Ve370 Introduction to Computer Organization Project 2. 2020.11.12

[2]IEEE Computer Society, Design Automation Standards Committee, IEEE Standard Verilog Hardware Description Language, The Institute of Electrical and Electronics Engineers, Inc. 3 Park Avenue, New York, NY 10016-5997, USA. 28 September 2001.

6 Appendix

6.1 Verilog source files

```
1  `timescale 1ns / 1ps
2
3  module single_cycle(clk);
4      input clk;
5      wire RegDst, beq, bne, jump, MemRead, MemtoReg, MemWrite, ALUSrc,
6      RegWrite, zero, Branch;
7      wire [1:0] ALUop;
8      wire [3:0] ALUcontrol;
9      wire [4:0] Write_reg;
10     wire [31:0] PC_in, PC_out, instruction, PC_next, jump_addr, branch_addr,
11     Read_data1, Read_data2, Write_data, sign_extend, ALU_in, ALU_result, Read_data,
12     Add1, Add2, branch_out;
13
14     assign jump_addr = {Add1[31:28], instruction[25:0], 2'b00};
15
16     Add add1(PC_out, 4, Add1);
17     Add add2(Add1, sign_extend * 4, Add2);
18
19     Mux_2to1 #(32) branch(((beq && zero) | (bne && ~zero)), Add1, Add2, branch_out);
20     Mux_2to1 #(32) Jump(jump, branch_out, jump_addr, PC_in);
21     Mux_2to1 #(5) write_reg(RegDst, instruction[20:16], instruction[15:11], Write_reg);
22     Mux_2to1 #(32) ALUin(ALUSrc, Read_data2, sign_extend, ALU_in);
23     Mux_2to1 #(32) write_data(MemtoReg, ALU_result, Read_data, Write_data);
24
25     PC pc(clk, PC_in, PC_out);
26     Ins_memory IM(PC_out, instruction);
27     Control control(instruction[31:26], RegDst, beq, bne, jump, MemRead,
28     MemtoReg, MemWrite, ALUop, ALUSrc, RegWrite);
29     Registers Reg_file(clk, RegWrite, instruction[25:21], instruction[20:16],
30     Write_reg, Write_data, Read_data1, Read_data2);
31     Sign_extend sign(instruction[15:0], sign_extend);
32     ALU_control ALUctrl(ALUop, instruction[5:0], ALUcontrol);
33     ALU alu(Read_data1, ALU_in, ALUcontrol, zero, ALU_result);
34     Data_memory datamem(clk, MemRead, MemWrite, ALU_result, Read_data2, Read_data);
35 endmodule
36
37 module Add(a, b, sum);
38     input [31:0] a, b;
39     output reg [31:0] sum;
40     always @ (*) begin
41         sum = a + b;
42     end
```

```
43 endmodule
44
45 module Mux_2to1(sel, data1, data2, result);
46     parameter N = 32;
47     input sel;
48     input [N - 1:0] data1, data2;
49     output reg [N - 1:0] result;
50     always @ (*) begin
51         result = (sel == 0) ? data1:data2;
52     end
53 endmodule
54
55 module PC(clk, PC_in, PC_out);
56     input clk;
57     input [31:0] PC_in;
58     output reg [31:0] PC_out;
59     initial begin
60         PC_out = 0;
61     end
62     always @ (posedge clk) begin
63         PC_out <= PC_in;
64     end
65 endmodule
66
67 module Ins_memory(Read_addr, Instruction);
68     input [31:0] Read_addr;
69     output reg [31:0] Instruction;
70     reg [31:0] ins_memory [63:0];
71     integer i;
72     initial begin
73         for (i = 0; i < 64; i = i + 1) ins_memory[i] = 0;
74         $readmemb("E:/zhan/VE370/Project/Project2/single_bonus.txt", ins_memory);
75     end
76     always @ (Read_addr) begin
77         Instruction = ins_memory[Read_addr/4];
78     end
79     //assign Instruction = ins_memory[Read_addr/4];
80 endmodule
81
82 module Control(op, RegDst, beq, bne, jump, MemRead, MemtoReg, MemWrite, ALUop,
83 ALUSrc, RegWrite);
84     input [5:0] op;
85     output reg RegDst, beq, bne, jump, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite;
86     output reg [1:0] ALUop;
87     initial begin
88         RegDst = 0;
```

```
89          beq = 0;
90          bne = 0;
91          jump = 0;
92          MemRead = 0;
93          MemtoReg = 0;
94          MemWrite = 0;
95          ALUSrc = 0;
96          RegWrite = 0;
97          ALUop = 0;
98      end
99      always @ (op) begin
100         case (op)
101             // R
102             6'b000000: begin
103                 RegDst <= 1;
104                 beq <= 0;
105                 bne <= 0;
106                 jump <= 0;
107                 MemRead <= 0;
108                 MemtoReg <= 0;
109                 MemWrite <= 0;
110                 ALUSrc <= 0;
111                 RegWrite <= 1;
112                 ALUop <= 2'b10;
113             end
114             // j
115             6'b000010: begin
116                 RegDst <= 0;
117                 beq <= 0;
118                 bne <= 0;
119                 jump <= 1;
120                 MemRead <= 0;
121                 MemtoReg <= 0;
122                 MemWrite <= 0;
123                 ALUSrc <= 0;
124                 RegWrite <= 0;
125                 ALUop <= 2'b10;
126             end
127             // beq
128             6'b000100: begin
129                 RegDst <= 0;
130                 beq <= 1;
131                 bne <= 0;
132                 jump <= 0;
133                 MemRead <= 0;
134                 MemtoReg <= 0;
```

```
135                 MemWrite <= 0;
136                 ALUSrc <= 0;
137                 RegWrite <= 0;
138                 ALUop <= 2'b01;
139             end
140             // bne
141             6'b000101: begin
142                 RegDst <= 0;
143                 beq <= 0;
144                 bne <= 1;
145                 jump <= 0;
146                 MemRead <= 0;
147                 MemtoReg <= 0;
148                 MemWrite <= 0;
149                 ALUSrc <= 0;
150                 RegWrite <= 0;
151                 ALUop <= 2'b01;
152             end
153             // addi
154             6'b001000: begin
155                 RegDst <= 0;
156                 beq <= 0;
157                 bne <= 0;
158                 jump <= 0;
159                 MemRead <= 0;
160                 MemtoReg <= 0;
161                 MemWrite <= 0;
162                 ALUSrc <= 1;
163                 RegWrite <= 1;
164                 ALUop <= 2'b00;
165             end
166             // andi
167             6'b001100: begin
168                 RegDst <= 0;
169                 beq <= 0;
170                 bne <= 0;
171                 jump <= 0;
172                 MemRead <= 0;
173                 MemtoReg <= 0;
174                 MemWrite <= 0;
175                 ALUSrc <= 1;
176                 RegWrite <= 1;
177                 ALUop <= 2'b11;
178             end
179             // lw
180             6'b100011: begin
```

```
                    RegDst <= 0;
                    beq <= 0;
                    bne <= 0;
                    jump <= 0;
                    MemRead <= 1;
                    MemtoReg <= 1;
                    MemWrite <= 0;
                    ALUSrc <= 1;
                    RegWrite <= 1;
                    ALUop <= 2'b00;
                end
                // sw
                6'b101011: begin
                    RegDst <= 0;
                    beq <= 0;
                    bne <= 0;
                    jump <= 0;
                    MemRead <= 0;
                    MemtoReg <= 0;
                    MemWrite <= 1;
                    ALUSrc <= 1;
                    RegWrite <= 0;
                    ALUop <= 2'b00;
                end
            endcase
        end
endmodule

module Registers(clk, RegWrite, Read_reg1, Read_reg2, Write_reg, Write_data,
Read_data1, Read_data2);
    input clk, RegWrite;
    input [4:0] Read_reg1, Read_reg2, Write_reg;
    input [31:0] Write_data;
    output [31:0] Read_data1, Read_data2;
    reg [31:0] memory [31:0];
    integer i;
    initial begin
        for (i = 0; i < 32; i = i + 1)
            memory[i] = 0;
    end
    assign Read_data1 = memory[Read_reg1];
    assign Read_data2 = memory[Read_reg2];
    always @ (posedge clk) begin
        if (RegWrite == 1)
            memory[Write_reg] <= Write_data;
        end
```

```
endmodule

module Sign_extend(in, out);
    input [15:0] in;
    output [31:0] out;
    assign out = {{16{in[15]}},in};
endmodule

module ALU_control(ALUop, funct, out_control);
    input [1:0] ALUop;
    input [5:0] funct;
    output reg [3:0] out_control;
    always @ (*) begin
        case(ALUop)
            // lw, sw, addi
            2'b00: out_control = 4'b0010; // add
            // beq and bne
            2'b01: out_control = 4'b0110; // subtract
            // R-type
            2'b10: begin
                case (funct)
                    6'b100000: out_control = 4'b0010; // add
                    6'b100010: out_control = 4'b0110; // subtract
                    6'b100100: out_control = 4'b0000; // and
                    6'b100101: out_control = 4'b0001; // or
                    6'b101010: out_control = 4'b0111; // stl
                endcase
            end
            // andi
            2'b11: out_control = 4'b0000; // and
        endcase
    end
endmodule

module ALU(a, b, control, zero, result);
    input [31:0] a, b;
    input [3:0] control;
    output zero;
    output reg [31:0] result;
    initial begin
        result = 0;
    end
    always @ * begin
        case (control)
            4'b0000: result = a & b;
            4'b0001: result = a | b;
```

```
273             4'b0010: result = a + b;
274             4'b0110: result = a - b;
275             4'b0111: begin
276                 if (a < b)
277                     result = 1;
278                 else
279                     result = 0;
280             end
281             4'b1100: result = ~(a | b);
282         endcase
283     end
284     assign zero = (result) ? 0 : 1;
285 endmodule
286
287 module Data_memory(clk, MemRead, MemWrite, addr, Write_data, Read_data);
288     input clk, MemRead, MemWrite;
289     input [31:0] addr, Write_data;
290     output [31:0] Read_data;
291     reg [31:0] memory [63:0];
292     integer i;
293     initial begin
294         for (i = 0; i < 64; i = i + 1)
295             memory[i] = 0;
296     end
297     always @ (addr) begin
298         if (MemWrite) memory[addr / 4] = Write_data;
299     end
300     assign Read_data = (MemRead) ? memory[addr / 4] : 0;
301 endmodule
```