CO002
Lab 11

LI XINGYOU

1809853U-I011-0037

Waveform

Run 1600 ns in the simulation.

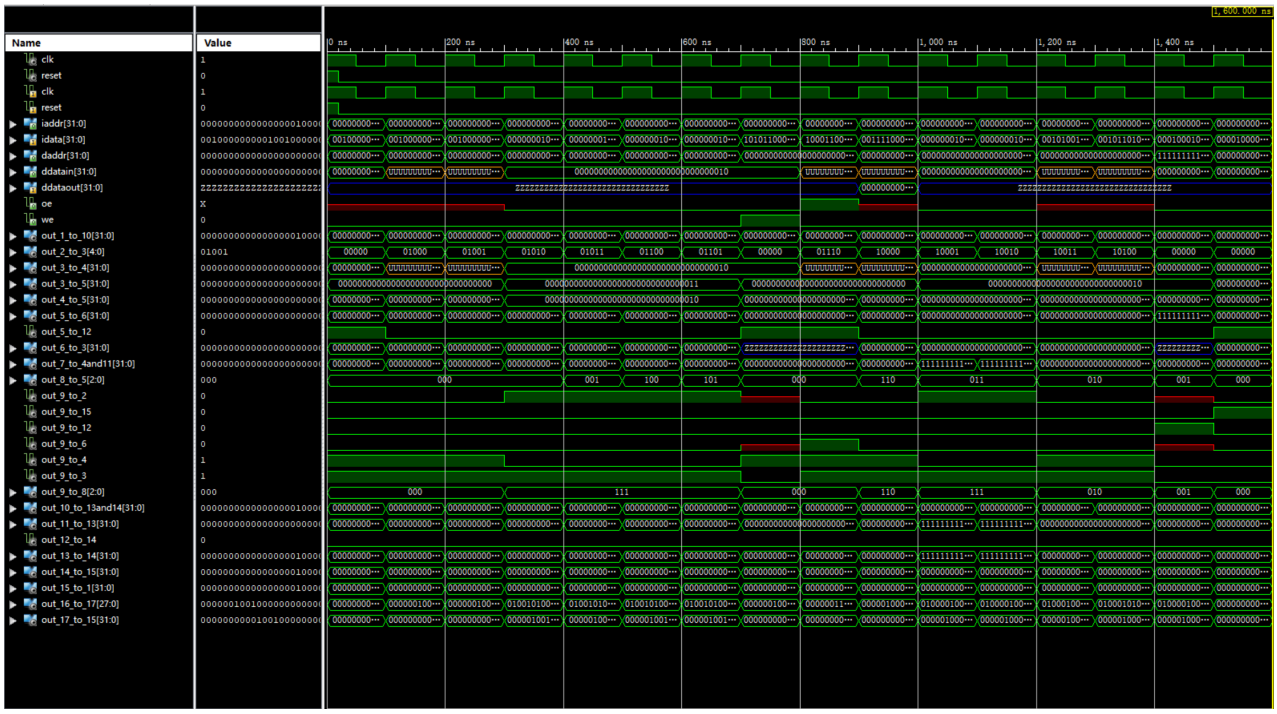


Figure 1: Full view

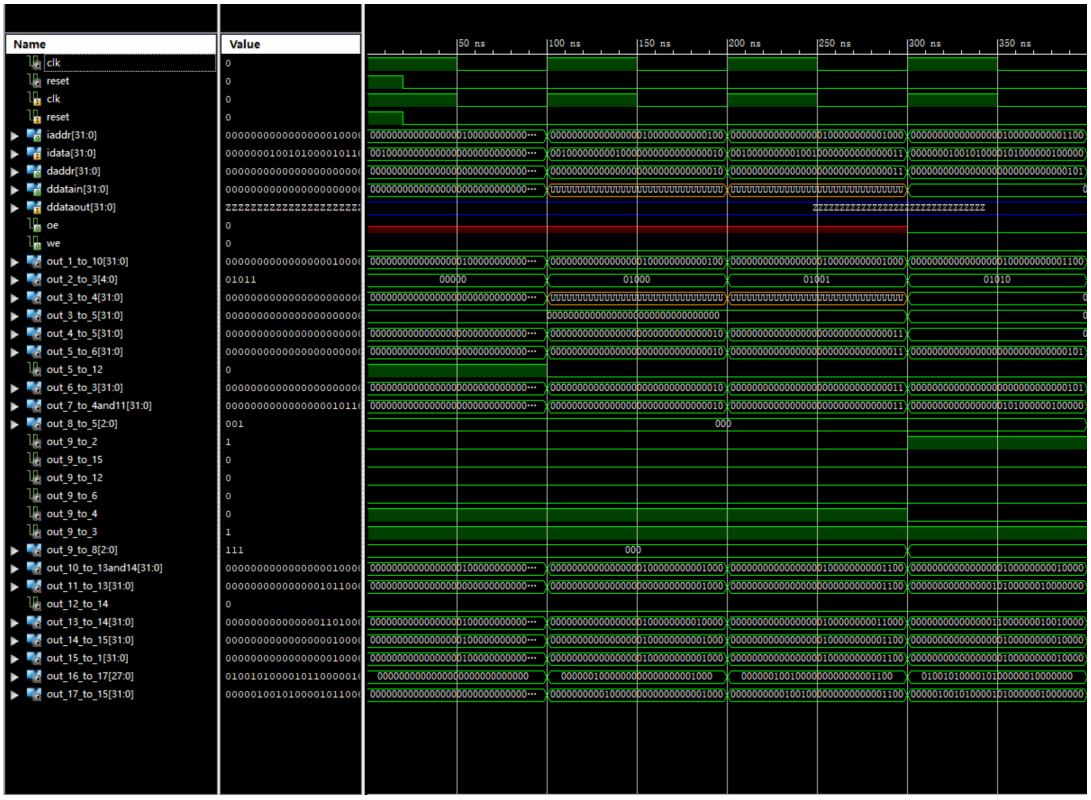


Figure 2: part1

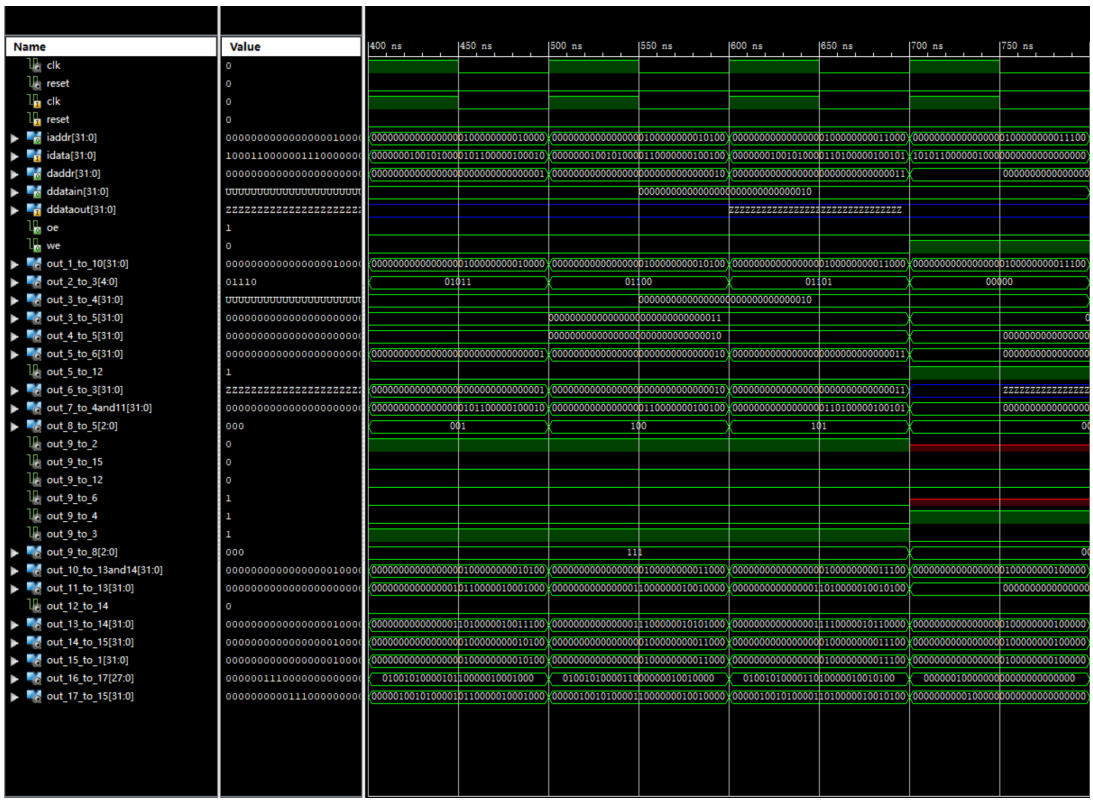


Figure 3: part2

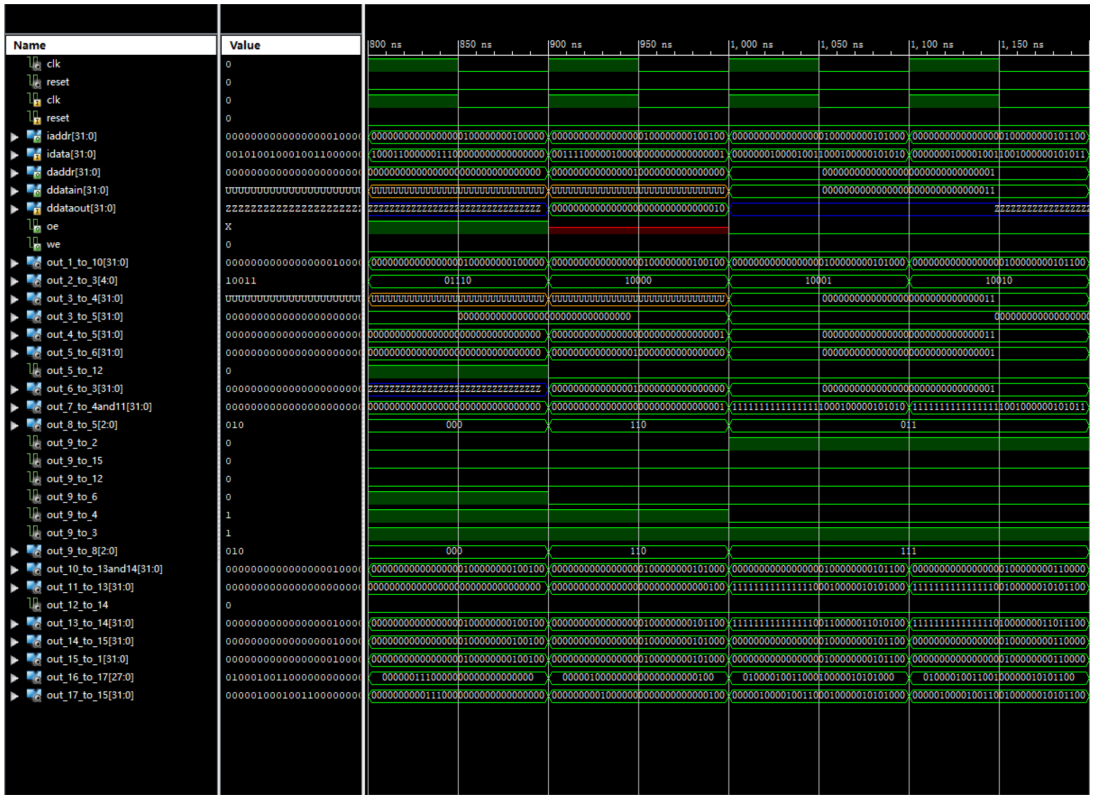


Figure 4: part3

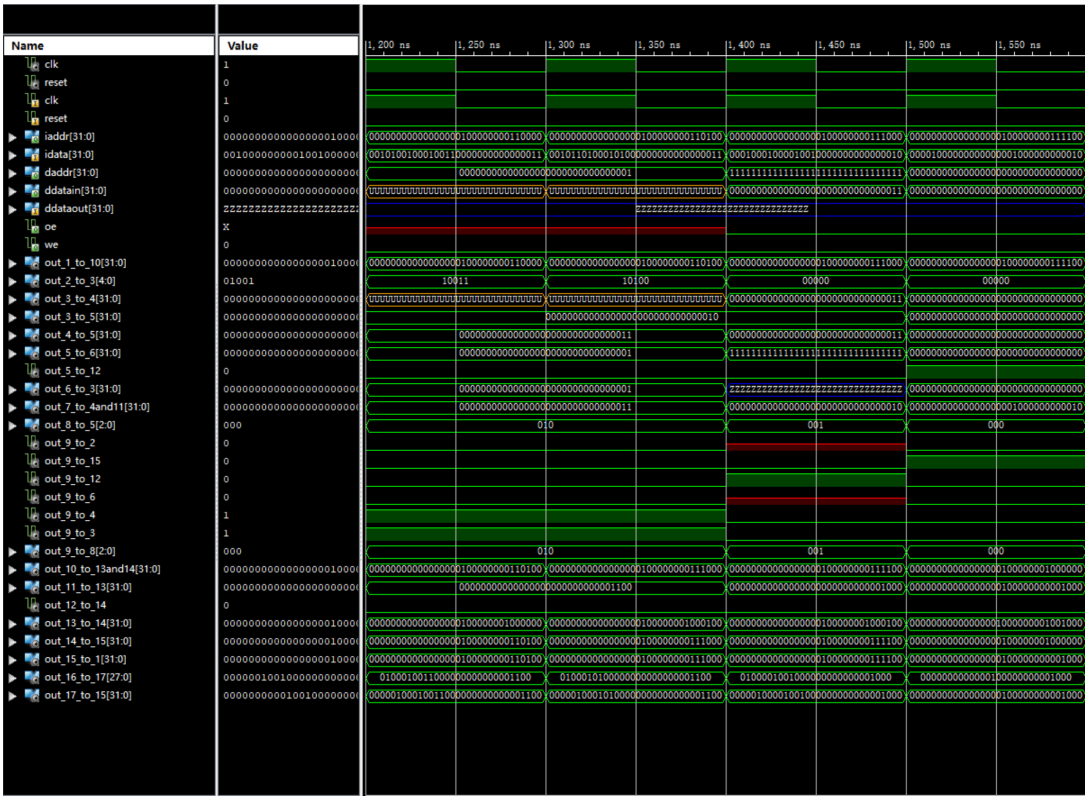


Figure 5: part4

Discussion

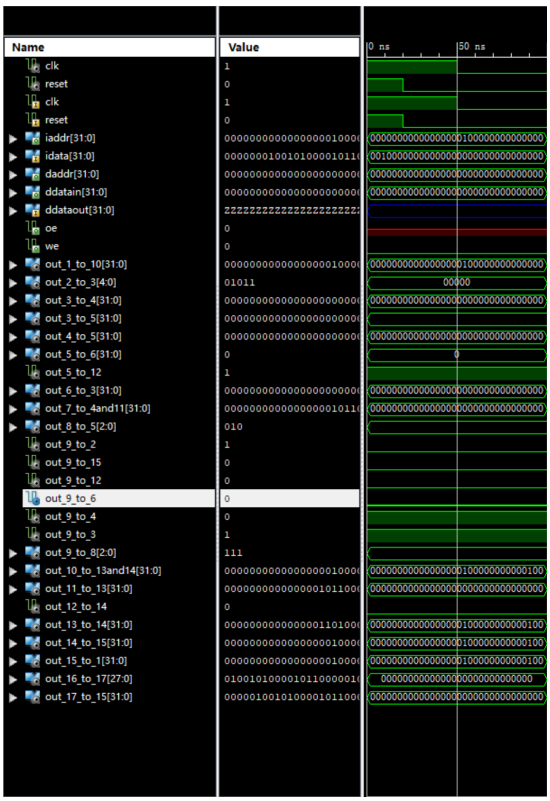
In the waveform figure, *out_X_to_Y* means the signal of component X transmit to component Y.

The test *programm.asc* is

```
20000000    # 00004000 addi $zero $zero 0
20080002    # 00004004 addi $t0 $zero 2
20090003    # 00004008 addi $t1 $zero 3
01285020    # 0000400c add $t2 $t1 $t0
01285822    # 00004010 sub $t3 $t1 $t0
01286024    # 00004014 and $t4 $t1 $t0
01286825    # 00004018 or $t5 $t1 $t0
ac080000    # 0000401c sw $t0 0($zero)
8c0e0000    # 00004020 lw $t6 0($zero)
3c100001    # 00004024 lui $s0 1
0109882a    # 00004028 slt $s1 $t0 $t1
0109902b    # 0000402c sltu $s2 $t0 $t1
29130003    # 00004030 slti $s3 $t0 3
2d140003    # 00004034 sltiu $s4 $t0 3
11090002    # 0000400c beq $t0 $t1 quit
08001002    # 00004014 j loop
```

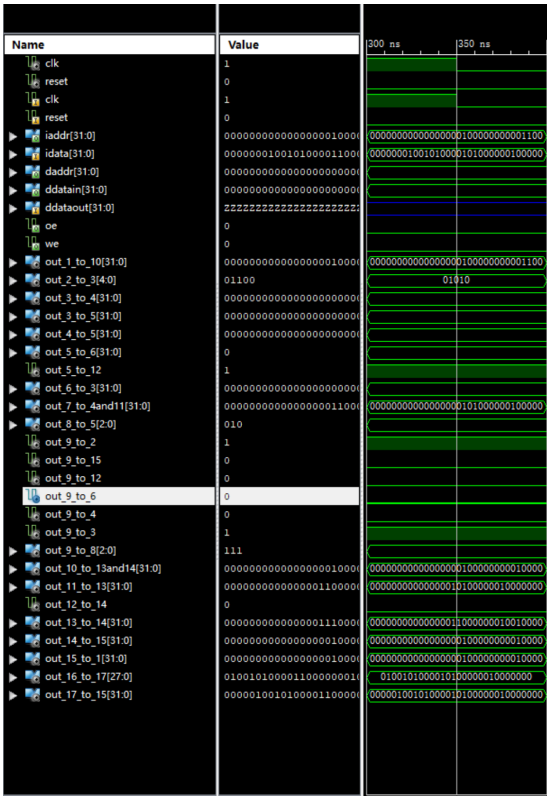
Verification 1

The first Hexadecimal command is 20000000, convert it into 32-bit binary is 00100000000000000 00000000000000, and the first 6 bits of the binary command is 001000 means *addi*, the last 16 bits are all zeros which is the immediate number, the 21 25 bit are all zeors which is the register source, the 16 20 bits are all zeros which is the register target, the result is 0. All the theorical results are match to the information in the figure.



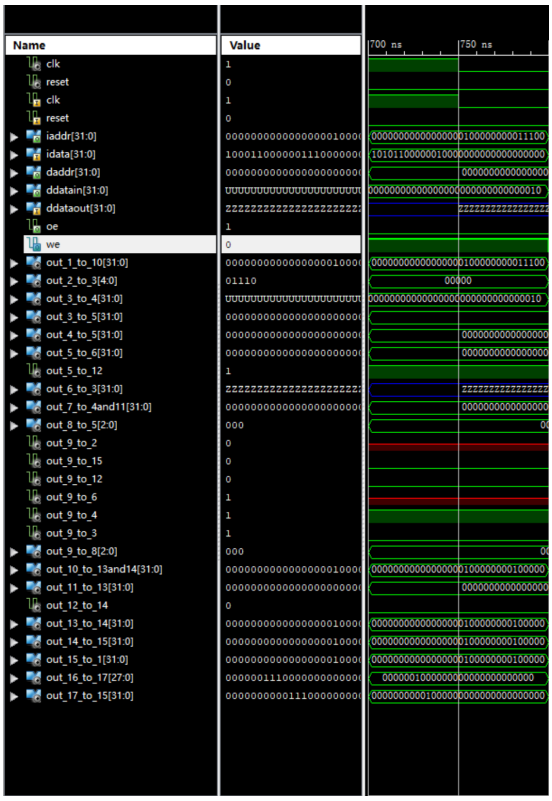
Verification 2

The forth Hexadecimal command is 01285020, convert it into 32-bit binary is 00000001001010000101000000100000, and the first 6 bits of the binary command is 000000 means $R-type$, the last 6 bits are 100000 which is function code which means *add*, the two registers are 010010 and 100001 which store the value 2 and 3, so the result is 5. All the theorical results are match to the information in the figure.



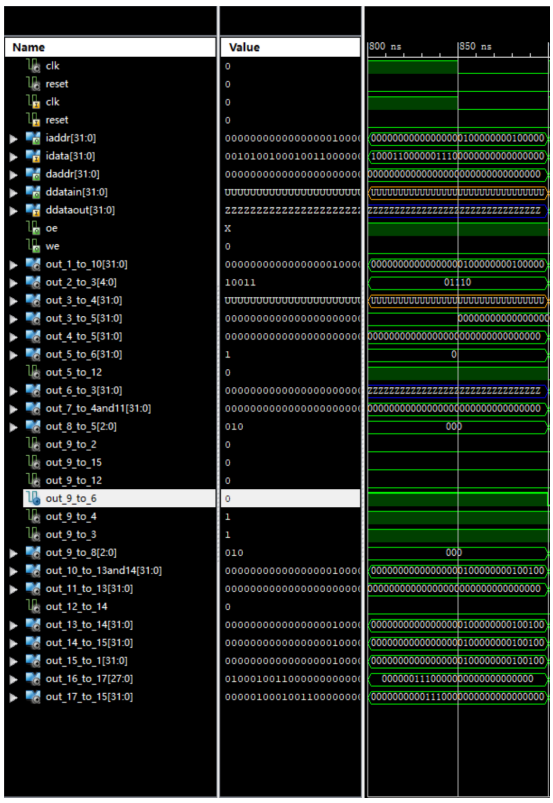
Verification 3

The eighth Hexadecimal command is 01285020, convert it into 32-bit binary is 10101100000010000000000000000000, and the first 6 bits of the binary command is 101011 means *sw*, the last 16 bits are all zeors which is the immediate number, and the register target is zero, so the result is zero. And only in this moment, the signal *we* could be 1 in this test programm. All the theorical results are match to the information in the figure.



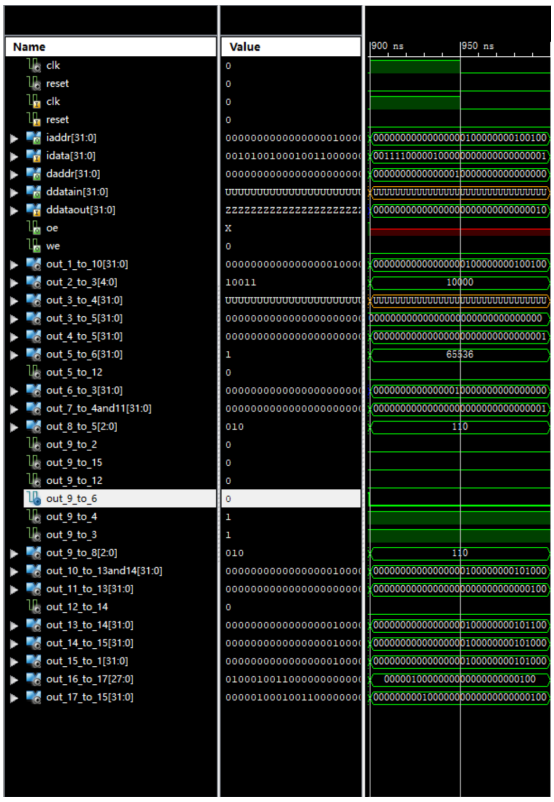
Verification 4

The ninth Hexadecimal command is 8c0e0000, convert it into 32-bit binary is 10001100000011100000000000000000, and the first 6 bits of the binary command is 100011 means *lw*, the last 16 bits are all zeors which is the immediate number, and the register source is zero, so the result is zero. And only in this moment, the signal *oe* could be 1 in this test programm. All the theorical results are match to the information in the figure.



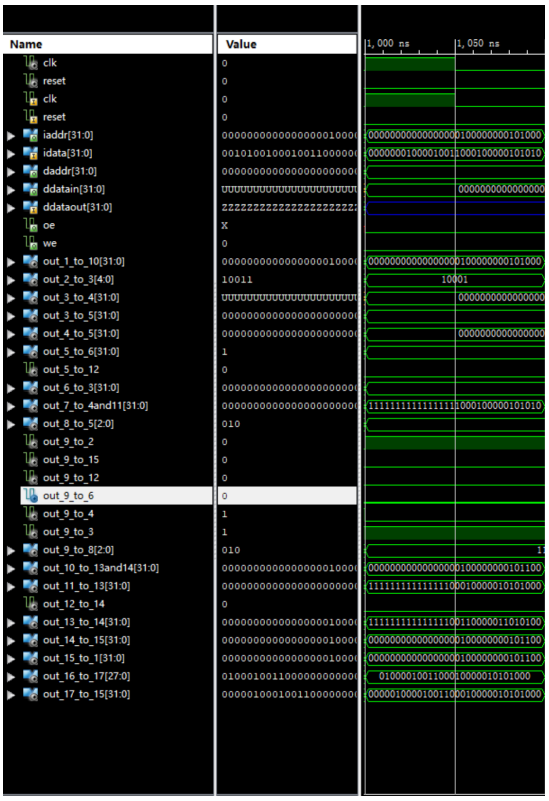
Verification 5

The tenth Hexadecimal command is 3c100001, convert it into 32-bit binary is 00111100000100000000 00000000001, and the first 6 bits of the binary command is 001111 means *lui*, the last 16 bits is 1, so it means shift this number left 16 bit,and the result is 00000000000000010000000000000000. And only in this moment, *ddataout* could be assigned value in this test programm. All the theorical results are match to the information in the figure.



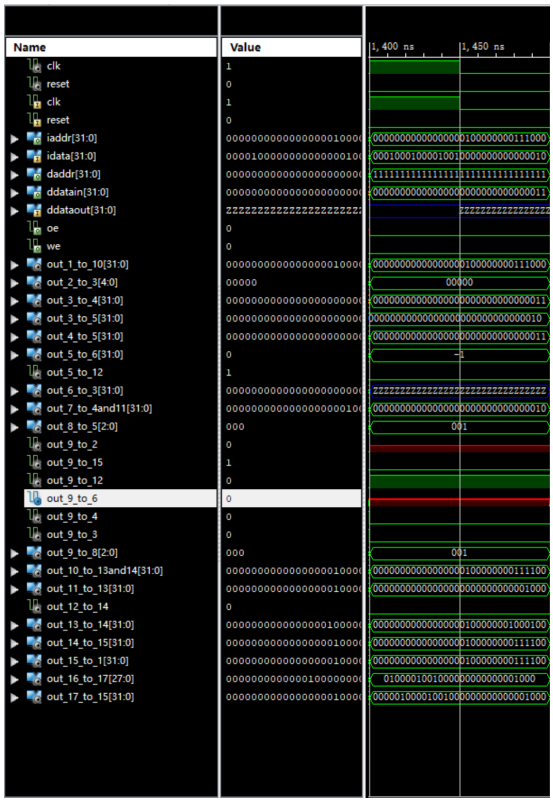
Verification 6

The eleventh Hexadecimal command is 0109882a, convert it into 32-bit binary is 00000001000010011000100000101010, and the first 6 bits of the binary command is 000000 means $R-type$, the last 6 bits are 101010 which which is function code which means *slt*, here the two registers are 010000 and 100110 which store value 2 and 3, so the result is 1. All the theorical results are match to the information in the figure.



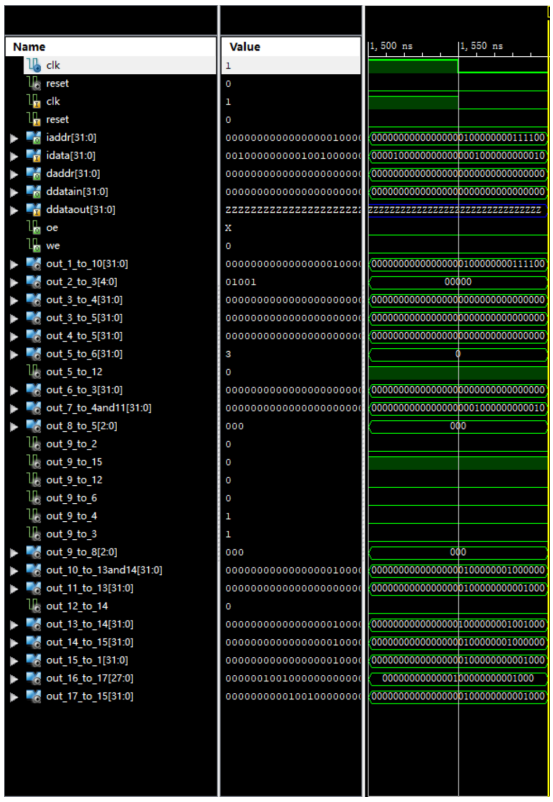
Verification 7

The fiftennth Hexadecimal command is 0109882a, convert it into 32-bit binary is 00010001000010010000000000010, and the first 6 bits of the binary command is 000100 means *beq*, the two registers are 010000 and 100100 which store value 2 and 3, so the result is -1. All the theorical results are match to the information in the figure.



Verification 8

The fiftennth Hexadecimal command is 0109882a, convert it into 32-bit binary is 00001000000000000001000000000010, and the first 6 bits of the binary command is 000010 means *jump*, the last 16 bits are the immediate number which is 0001000000000010. All the theorical results are match to the information in the figure.



Code

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6
7
8
9
10 entity ALU is
11     Port ( ALUcontrol : in  STD_LOGIC_VECTOR (2 downto 0);
12            A : in  STD_LOGIC_VECTOR (31 downto 0);
13            B : in  STD_LOGIC_VECTOR (31 downto 0);
14            zero : out  STD_LOGIC;
15            ALUresult : out  STD_LOGIC_VECTOR (31 downto 0));
16 end ALU;
17
18 architecture Behavioral of ALU is
19
20 signal result : STD_LOGIC_VECTOR(31 downto 0);
21
22 begin
23     result <= (A + B) when (ALUcontrol = "000") else
24               (A - B) when (ALUcontrol = "001") else
25               "00000000000000000000000000000001" when ((ALUcontrol = "010") and(A ...
                     < B)) else
26               "00000000000000000000000000000000" when ((ALUcontrol = "010") and(A >=...
                      B)) else
27               "00000000000000000000000000000001" when ((ALUcontrol = "011") ...
                     and(unsigned(A) < unsigned(B))) else
28               "00000000000000000000000000000000" when ((ALUcontrol = "011") ...
                     and(unsigned(A) >= unsigned(B))) else
29               (A and B) when (ALUcontrol = "100") else
30               (A or B) when (ALUcontrol = "101") else
31               (B(15 downto 0) & "0000000000000000") when (ALUcontrol = "110") else
32               "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU";
33     ALUresult <= result;
34     with result select
35         zero <= '1' when "00000000000000000000000000000000",
36                 '0' when others;
37 end Behavioral;
```

ALU.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7  entity ALU_mux is
8      Port ( ALUSrc : in  STD_LOGIC;
9             Reg : in  STD_LOGIC_VECTOR (31 downto 0);
10            imm : in  STD_LOGIC_VECTOR (31 downto 0);
11            ALU_mux_out : out  STD_LOGIC_VECTOR (31 downto 0));
12 end ALU_mux;
13
```

```
14 architecture Behavioral of ALU_mux is
15
16 begin
17     ALU_mux_out <= Reg when (ALUSrc = '0') else
18                    imm;
19
20 end Behavioral;
```

ALU_mux.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7  entity ALUcontrol is
8      Port ( funct : in  STD_LOGIC_VECTOR (5 downto 0);
9             ALUOp : in  STD_LOGIC_VECTOR (2 downto 0);
10            ALUcontrol : out  STD_LOGIC_VECTOR (2 downto 0));
11 end ALUcontrol;
12
13 architecture Behavioral of ALUcontrol is
14
15 signal Rtype : STD_LOGIC;
16
17 begin
18     ALUcontrol <= "000" when (ALUOp = "000") else --lw sw j -add
19                   "001" when (ALUOp = "001") else --beq -sub
20                   "010" when (ALUOp = "010") else --slti
21                   "010" when (ALUOp = "010") else --sltiu
22                   "110" when (ALUOp = "110") else --s16
23                   "000" when (ALUOp = "111")and (funct = "100000") else ...
                         --R type--add
24                   "001" when (ALUOp = "111")and (funct = "100010") else ...
                         --R type--sub
25                   "011" when (ALUOp = "111")and (funct = "101010") else ...
                         --R type--slt
26                   "100" when (ALUOp = "111")and (funct = "100100") else ...
                         --R type--and
27                   "101" when (ALUOp = "111")and (funct = "100101") else ...
                         --R type--or
28                   "011" when (ALUOp = "111")and (funct = "101011") else ...
                         --R type--sltu
29                   "UUU";
30 end Behavioral;
```

ALUcontrol.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7  entity Branch_mux is
8      Port ( Branch_result : in  STD_LOGIC;
9             PC_4 : in  STD_LOGIC_VECTOR (31 downto 0);
10            BranchAdder : in  STD_LOGIC_VECTOR (31 downto 0);
11            result : out  STD_LOGIC_VECTOR (31 downto 0));
```

```
12 end Branch_mux;
13
14 architecture Behavioral of Branch_mux is
15
16 begin
17     result <= PC_4 when (Branch_result = '0') else
18                     BranchAdder ;
19 end Behavioral;
```

Branch_mux.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  use IEEE.NUMERIC_STD.ALL;
7
8
9  entity BranchAddressAdder is
10     Port ( PC_4 : in  STD_LOGIC_VECTOR (31 downto 0);
11            ShiftLeft2 : in  STD_LOGIC_VECTOR (31 downto 0);
12            result : out  STD_LOGIC_VECTOR (31 downto 0));
13 end BranchAddressAdder;
14
15 architecture Behavioral of BranchAddressAdder is
16
17 begin
18     result <= PC_4 + ShiftLeft2;
19 end Behavioral;
```

BranchAddressAdder.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6  entity CheckBranch is
7      Port ( Branch : in  STD_LOGIC;
8             zero : in  STD_LOGIC;
9             branch_out : out  STD_LOGIC);
10 end CheckBranch;
11
12 architecture Behavioral of CheckBranch is
13
14 begin
15     branch_out <= (Branch and zero);
16 end Behavioral;
```

CheckBranch.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5  entity ControlSignalGeneration is
6      Port ( op : in  STD_LOGIC_VECTOR (5 downto 0);
7             RegDst : out  STD_LOGIC;
```

```
8               ALUSrc : out  STD_LOGIC;
9               MemtoReg : out  STD_LOGIC;
10              RegWrite : out  STD_LOGIC;
11              MemRead : out  STD_LOGIC;
12              MemWrite : out  STD_LOGIC;
13              Branch : out  STD_LOGIC;
14              Jump : out  STD_LOGIC;
15              ALUOp : out  STD_LOGIC_VECTOR (2 downto 0));
16  end ControlSignalGeneration;
17
18  architecture Behavioral of ControlSignalGeneration is
19  begin
20      with op select
21          RegDst <= '1' when "000000", -- R type
22                    '0' when "100011", -- lw
23                    'X' when "101011", -- sw
24                    'X' when "000100", -- beq
25                    '0' when "000010", -- j
26                    '0' when "001111", -- lui
27                    '0' when "001010", -- slti
28                    '0' when "001000", -- addi
29                    '0' when "001011", -- sltiu
30                    'U' when others;
31      with op select
32          ALUSrc <= '0' when "000000",
33                    '1' when "100011",
34                    '1' when "101011",
35                    '0' when "000100",
36                    '0' when "000010",
37                    '1' when "001111",
38                    '1' when "001010",
39                    '1' when "001000",
40                    '1' when "001011",
41                    'U' when others;
42      with op select
43          MemtoReg <= '0' when "000000",
44                      '1' when "100011",
45                      'X' when "101011",
46                      'X' when "000100",
47                      '0' when "000010",
48                      '0' when "001111",
49                      '0' when "001010",
50                      '0' when "001000",
51                      '0' when "001011",
52                      'U' when others;
53      with op select
54          RegWrite <= '1' when "000000",
55                      '1' when "100011",
56                      '0' when "101011",
57                      '0' when "000100",
58                      '0' when "000010",
59                      '1' when "001111",
60                      '1' when "001010",
61                      '1' when "001000",
62                      '1' when "001011",
63                      'U' when others;
64      with op select
65          MemRead <= '0' when "000000",
66                     '1' when "100011",
67                     '0' when "101011",
68                     '0' when "000100",
69                     '0' when "000010",
```

```
                        'X' when "001111",
                        'X' when "001010",
                        'X' when "001000",
                        'X' when "001011",
                        'U' when others;
    with op select
        MemWrite <= '0' when "000000",
                        '0' when "100011",
                        '1' when "101011",
                        '0' when "000100",
                        '0' when "000010",
                        '0' when "001111",
                        '0' when "001010",
                        '0' when "001000",
                        '0' when "001011",
                        'U' when others;
    with op select
        Branch <= '0' when "000000",
                        '0' when "100011",
                        '0' when "101011",
                        '1' when "000100",
                        '0' when "000010",
                        '0' when "001111",
                        '0' when "001010",
                        '0' when "001000",
                        '0' when "001011",
                        'U' when others;
    with op select
        Jump <= '0' when "000000",
                        '0' when "100011",
                        '0' when "101011",
                        '0' when "000100",
                        '1' when "000010",
                        '0' when "001111",
                        '0' when "001010",
                        '0' when "001000",
                        '0' when "001011",
                        'U' when others;
    with op select
        ALUOp <= "111" when "000000",
                        "000" when "100011", -- add
                        "000" when "101011", -- add
                        "001" when "000100", -- sub
                        "000" when "000010", -- add
                        "110" when "001111", -- shift
                        "010" when "001010", -- slti
                        "000" when "001000", -- addi
                        "010" when "001011", -- sltiu
                        "UUU" when others;

end Behavioral;
```

ControlSignalGeneration.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity Jump_mux is
```

```
8       Port ( Jump : in  STD_LOGIC;
9              J_Address : in  STD_LOGIC_VECTOR (31 downto 0);
10             Branch_mux_result : in  STD_LOGIC_VECTOR (31 downto 0);
11             newPC : out  STD_LOGIC_VECTOR (31 downto 0));
12 end Jump_mux;
13
14 architecture Behavioral of Jump_mux is
15
16 begin
17     newPC <= Branch_mux_result when (Jump = '0') else
18                 J_Address ;
19 end Behavioral;
```

Jump_mux.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7  entity JumpAddrCon is
8      Port ( Addr_extend : in  STD_LOGIC_VECTOR (27 downto 0);
9             PC_4 : in  STD_LOGIC_VECTOR (3 downto 0);
10            JumpAddr : out  STD_LOGIC_VECTOR (31 downto 0));
11 end JumpAddrCon;
12
13 architecture Behavioral of JumpAddrCon is
14
15 begin
16     JumpAddr <= (PC_4 & Addr_extend);
17 end Behavioral;
```

JumpAddrCon.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6  entity JumpAddressShift is
7      Port ( addr : in  STD_LOGIC_VECTOR (25 downto 0);
8             AddrShift : out  STD_LOGIC_VECTOR (27 downto 0));
9  end JumpAddressShift;
10
11 architecture Behavioral of JumpAddressShift is
12
13 begin
14     AddrShift <= (addr & "00");
15 end Behavioral;
```

JumpAddressShift.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  use IEEE.NUMERIC_STD.ALL;
```

```
7
8  entity PCadd4 is
9      Port ( PC : in  STD_LOGIC_VECTOR (31 downto 0);
10            PC_4 : out  STD_LOGIC_VECTOR (31 downto 0));
11 end PCadd4;
12
13 architecture Behavioral of PCadd4 is
14
15 begin
16     PC_4 <= PC + 4;
17 end Behavioral;
```

PCadd4.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6  entity PC is
7      Port ( clk : in  STD_LOGIC;
8             reset : in  STD_LOGIC;
9             in_pc : in  STD_LOGIC_VECTOR (31 downto 0);
10            out_pc : out  STD_LOGIC_VECTOR (31 downto 0));
11 end PC;
12
13 architecture Behavioral of PC is
14
15
16
17 begin
18     process(clk,reset,in_pc)
19     begin
20         if (reset = '1') then
21             out_pc <= "00000000000000000100000000000000";
22         elsif (clk'event and clk = '1') then
23             out_pc <= in_pc;
24         end if;
25     end process;
26
27
28
29 end Behavioral;
```

PC.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6  entity RegDst_mux is
7      Port ( RegDst : in  STD_LOGIC;
8             RT : in  STD_LOGIC_VECTOR (4 downto 0);
9             RD : in  STD_LOGIC_VECTOR (4 downto 0);
10            WriteReg : out  STD_LOGIC_VECTOR (4 downto 0));
11 end RegDst_mux;
12
13 architecture Behavioral of RegDst_mux is
14
```

```
15 begin
16     WriteReg <= RT when (RegDst = '0') else
17                 RD ;
18
19
20 end Behavioral;
```

RegDst_mux.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7
8
9  entity registers is
10     Port ( clk : in  STD_LOGIC;
11            reset : in  STD_LOGIC;
12            instruction : in  STD_LOGIC_VECTOR (31 downto 0);
13            wa : in  STD_LOGIC_VECTOR (4 downto 0);
14            rd1 : out  STD_LOGIC_VECTOR (31 downto 0); --
15            rd2 : out  STD_LOGIC_VECTOR (31 downto 0); --
16            wd : in  STD_LOGIC_VECTOR (31 downto 0); --
17            we : in  STD_LOGIC;
18               regview_t1 : out STD_LOGIC_VECTOR (31 downto 0));
19 end registers;
20
21
22 architecture Behavioral of registers is
23 TYPE register_file is array (0 to 31) of STD_LOGIC_VECTOR (31 DOWNTO 0);
24 SIGNAL reg_file : register_file := ...
       ("00000000000000000000000000000000","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
25 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
26 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
27 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
28 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
29 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
30 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
31 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
32 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
33 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
34 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
35 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
36 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
37 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
38 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU",
39 "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU","UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU"
40             );
41 SIGNAL ra1 :  STD_LOGIC_VECTOR (4 downto 0);
42 SIGNAL ra2 :  STD_LOGIC_VECTOR (4 downto 0);
43 begin
44 ra1 <= instruction(25 downto 21);
45 ra2 <= instruction(20 downto 16);
46 rd1 <= reg_file(conv_integer(ra1));
47 rd2 <= reg_file(conv_integer(ra2));
48
49 PROCESS (clk, reset, we, wa, wd)
50 begin
51 if(reset='1') then
```

```
52
53  elsif (clk 'EVENT AND clk='1') then
54      regview_t1 <= reg_file(20);
55      if (we = '1') and (wa /= 0) then
56        reg_file(conv_integer(wa)) <= wd;
57      end if;
58  end if;
59
60  end PROCESS;
61
62  end Behavioral;
```

registers.vhd

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.STD_LOGIC_ARITH.ALL;
5   use IEEE.STD_LOGIC_UNSIGNED.ALL;
6   use IEEE.NUMERIC_STD.ALL;
7
8
9
10  entity scpu is
11      Port ( clk : in  STD_LOGIC;
12             reset : in  STD_LOGIC;
13                iaddr : out std_logic_vector(31 downto 0);
14                idata : in std_logic_vector(31 downto 0);
15                daddr : out std_logic_vector(31 downto 0);
16                ddatain : out std_logic_vector(31 downto 0);
17                ddataout : in std_logic_vector(31 downto 0);
18                oe : out std_logic;
19                we : out std_logic);
20  end scpu;
21
22  architecture Behavioral of scpu is
23
24  component PC
25      Port ( clk : in  STD_LOGIC;
26             reset : in  STD_LOGIC;
27             in_pc : in  STD_LOGIC_VECTOR (31 downto 0);
28             out_pc : out  STD_LOGIC_VECTOR (31 downto 0));
29  end component;
30
31  component RegDst_mux
32      Port ( RegDst : in  STD_LOGIC;
33             RT : in  STD_LOGIC_VECTOR (4 downto 0);
34             RD : in  STD_LOGIC_VECTOR (4 downto 0);
35             WriteReg : out  STD_LOGIC_VECTOR (4 downto 0));
36  end component;
37
38  component registers
39      port ( clk: in std_logic;
40             reset: in std_logic;
41             instruction : in  STD_LOGIC_VECTOR (31 downto 0);
42             wa: in std_logic_vector(4 downto 0);
43             rd1: out std_logic_vector(31 downto 0);
44             rd2: out std_logic_vector(31 downto 0);
45             wd: in std_logic_vector(31 downto 0);
46             we: in std_logic );
47  end component;
```

```
48
49  component ALU_mux
50      Port ( ALUSrc : in  STD_LOGIC;
51             Reg : in  STD_LOGIC_VECTOR (31 downto 0);
52             imm : in  STD_LOGIC_VECTOR (31 downto 0);
53             ALU_mux_out : out  STD_LOGIC_VECTOR (31 downto 0));
54  end component;
55
56  component ALU
57      Port ( ALUcontrol : in  STD_LOGIC_VECTOR (2 downto 0);
58             A : in  STD_LOGIC_VECTOR (31 downto 0);
59             B : in  STD_LOGIC_VECTOR (31 downto 0);
60             zero : out  STD_LOGIC;
61             ALUresult : out  STD_LOGIC_VECTOR (31 downto 0));
62  end component;
63
64  component WriteData_mux
65      Port ( MemtoReg : in  STD_LOGIC;
66             ddataout : in  STD_LOGIC_VECTOR (31 downto 0);
67             ALUresult : in  STD_LOGIC_VECTOR (31 downto 0);
68             WriteData : out  STD_LOGIC_VECTOR (31 downto 0));
69  end component;
70
71  component  SignExtend
72      Port ( imm : in  STD_LOGIC_VECTOR (15 downto 0);
73             imm32 : out  STD_LOGIC_VECTOR (31 downto 0));
74  end component;
75
76  component ALUcontrol
77      Port ( funct : in  STD_LOGIC_VECTOR (5 downto 0);
78             ALUOp : in  STD_LOGIC_VECTOR (2 downto 0);
79             ALUcontrol : out  STD_LOGIC_VECTOR (2 downto 0));
80  end component;
81
82  component ControlSignalGeneration
83      Port ( op : in  STD_LOGIC_VECTOR (5 downto 0);
84             RegDst : out  STD_LOGIC;
85             ALUSrc : out  STD_LOGIC;
86             MemtoReg : out  STD_LOGIC;
87             RegWrite : out  STD_LOGIC;
88             MemRead : out  STD_LOGIC;
89             MemWrite : out  STD_LOGIC;
90             Branch : out  STD_LOGIC;
91             Jump : out  STD_LOGIC;
92             ALUOp : out  STD_LOGIC_VECTOR (2 downto 0));
93  end component;
94
95  component PCadd4
96      Port ( PC : in  STD_LOGIC_VECTOR (31 downto 0);
97             PC_4 : out  STD_LOGIC_VECTOR (31 downto 0));
98  end component;
99
100 component ShiftImmLeft
101     Port ( Extend_Imm : in  STD_LOGIC_VECTOR (31 downto 0);
102            ShiftLeft2 : out  STD_LOGIC_VECTOR (31 downto 0));
103 end component;
104
105 component CheckBranch
106     Port ( Branch : in  STD_LOGIC;
107            zero : in  STD_LOGIC;
108            branch_out : out  STD_LOGIC);
109 end component;
```

```
110
111 component BranchAddressAdder
112     Port ( PC_4 : in  STD_LOGIC_VECTOR (31 downto 0);
113            ShiftLeft2 : in  STD_LOGIC_VECTOR (31 downto 0);
114            result : out  STD_LOGIC_VECTOR (31 downto 0));
115 end component;
116
117 component Branch_mux
118     Port ( Branch_result : in  STD_LOGIC;
119            PC_4 : in  STD_LOGIC_VECTOR (31 downto 0);
120            BranchAdder : in  STD_LOGIC_VECTOR (31 downto 0);
121            result : out  STD_LOGIC_VECTOR (31 downto 0));
122 end component;
123
124 component Jump_mux
125     Port ( Jump : in  STD_LOGIC;
126            J_Address : in  STD_LOGIC_VECTOR (31 downto 0);
127            Branch_mux_result : in  STD_LOGIC_VECTOR (31 downto 0);
128            newPC : out  STD_LOGIC_VECTOR (31 downto 0));
129 end component;
130
131 component JumpAddressShift
132     Port ( addr : in  STD_LOGIC_VECTOR (25 downto 0);
133            AddrShift : out  STD_LOGIC_VECTOR (27 downto 0));
134 end component;
135
136 component JumpAddrCon
137     Port ( Addr_extend : in  STD_LOGIC_VECTOR (27 downto 0);
138            PC_4 : in  STD_LOGIC_VECTOR (3 downto 0);
139            JumpAddr : out  STD_LOGIC_VECTOR (31 downto 0));
140 end component;
141
142 signal out_1_to_10 : STD_LOGIC_VECTOR (31 downto 0);
143 signal out_2_to_3 : STD_LOGIC_VECTOR (4 downto 0);
144 signal out_3_to_4 : STD_LOGIC_VECTOR (31 downto 0);
145 signal out_3_to_5 : STD_LOGIC_VECTOR (31 downto 0);
146 signal out_4_to_5 : STD_LOGIC_VECTOR (31 downto 0);
147 signal out_5_to_6 : STD_LOGIC_VECTOR (31 downto 0);
148 signal out_5_to_12 : STD_LOGIC;
149 signal out_6_to_3 : STD_LOGIC_VECTOR (31 downto 0);
150 signal out_7_to_4and11 : STD_LOGIC_VECTOR (31 downto 0);
151 signal out_8_to_5 : STD_LOGIC_VECTOR (2 downto 0);
152 signal out_9_to_2 : STD_LOGIC;
153 signal out_9_to_15 : STD_LOGIC;
154 signal out_9_to_12 : STD_LOGIC;
155 signal out_9_to_6 : STD_LOGIC;
156 signal out_9_to_4 : STD_LOGIC;
157 signal out_9_to_3 : STD_LOGIC;
158 signal out_9_to_8 : STD_LOGIC_VECTOR (2 downto 0);
159 signal out_10_to_13and14 : STD_LOGIC_VECTOR (31 downto 0);
160 signal out_11_to_13 : STD_LOGIC_VECTOR (31 downto 0);
161 signal out_12_to_14 : STD_LOGIC;
162 signal out_13_to_14 : STD_LOGIC_VECTOR (31 downto 0);
163 signal out_14_to_15 : STD_LOGIC_VECTOR (31 downto 0);
164 signal out_15_to_1 : STD_LOGIC_VECTOR (31 downto 0);
165 signal out_16_to_17 : STD_LOGIC_VECTOR (27 downto 0);
166 signal out_17_to_15 : STD_LOGIC_VECTOR (31 downto 0);
167
168
169 begin
170
171 PC1 : PC port map(
```

```
172             clk => clk,
173             reset => reset,
174             in_pc => out_15_to_1,
175             out_pc => out_1_to_10
176 );
177
178
179 RegDst_mux2 : RegDst_mux port map(
180             RegDst => out_9_to_2,
181             RT => idata(20 downto 16),
182             RD => idata(15 downto 11),
183             WriteReg => out_2_to_3
184 );
185
186 registers3 : registers port map(
187             clk => clk,
188             reset => reset,
189             wa => out_2_to_3,
190             rd1 => out_3_to_5,
191             rd2 => out_3_to_4,
192             wd => out_6_to_3,
193             we => out_9_to_3,
194             instruction => idata
195 );
196
197 ALU_mux4 : ALU_mux port map(
198             ALUSrc => out_9_to_4,
199             Reg => out_3_to_4,
200             imm => out_7_to_4and11,
201             ALU_mux_out => out_4_to_5
202 );
203
204 ALU5 : ALU port map(
205             ALUcontrol => out_8_to_5,
206             A => out_3_to_5,
207             B => out_4_to_5,
208             zero => out_5_to_12,
209             ALUresult => out_5_to_6
210 );
211
212 WriteData_mux6 : WriteData_mux port map(
213             MemtoReg => out_9_to_6,
214             ddataout => ddataout,
215             ALUresult => out_5_to_6,
216             WriteData => out_6_to_3
217 );
218
219 SignExtend7 : SignExtend port map(
220             imm => idata(15 downto 0),
221             imm32 => out_7_to_4and11
222 );
223
224 ALUcontrol8 : ALUcontrol port map(
225             funct => idata(5 downto 0),
226             ALUOp => out_9_to_8,
227             ALUcontrol => out_8_to_5
228 );
229
230 ControlSignalGeneration9 : ControlSignalGeneration port map(
231             op => idata(31 downto 26),
232             RegDst => out_9_to_2,
233             ALUSrc => out_9_to_4,
```

```
234                 MemtoReg => out_9_to_6,
235                 RegWrite => out_9_to_3,
236                 MemRead => oe,
237                 MemWrite => we,
238                 Branch => out_9_to_12,
239                 Jump => out_9_to_15,
240                 ALUOp => out_9_to_8
241 );
242
243 PCadd4_10 : PCadd4 port map(
244                 PC => out_1_to_10,
245                 PC_4 => out_10_to_13and14
246 );
247
248 ShiftImmLeft11 : ShiftImmLeft port map(
249                 Extend_Imm => out_7_to_4and11,
250                 ShiftLeft2 => out_11_to_13
251 );
252
253 CheckBranch12 : CheckBranch port map(
254                 Branch => out_9_to_12,
255                 zero => out_5_to_12,
256                 branch_out => out_12_to_14
257 );
258
259 BranchAddressAdder13 : BranchAddressAdder port map(
260                 PC_4 => out_10_to_13and14,
261                 ShiftLeft2 => out_11_to_13,
262                 result => out_13_to_14
263 );
264
265 Branch_mux14 : Branch_mux port map(
266                 Branch_result => out_12_to_14,
267                 PC_4 => out_10_to_13and14,
268                 BranchAdder => out_13_to_14,
269                 result => out_14_to_15
270 );
271
272 Jump_mux15 : Jump_mux port map(
273                 Jump => out_9_to_15,
274                 J_Address => out_17_to_15,
275                 Branch_mux_result => out_14_to_15,
276                 newPC => out_15_to_1
277 );
278
279 JumpAddressShift16 : JumpAddressShift port map(
280                 addr => idata(25 downto 0),
281                 AddrShift => out_16_to_17
282 );
283
284 JumpAddrCon17 : JumpAddrCon port map(
285                 Addr_extend => out_16_to_17,
286                 PC_4 => out_10_to_13and14(31 downto 28),
287                 JumpAddr => out_17_to_15
288 );
289
290 iaddr <= out_1_to_10;
291 ddatain <= out_3_to_4;
292 daddr <= out_5_to_6;
293
294
295
```

```
296 end Behavioral;
```

scpu.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  use IEEE.NUMERIC_STD.ALL;
7
8
9  entity scpu_top is
10     Port ( clk : in  STD_LOGIC;
11            reset : in  STD_LOGIC);
12 end scpu_top;
13
14 architecture Behavioral of scpu_top is
15
16 component scpu
17     Port ( clk : in  STD_LOGIC;
18            reset : in  STD_LOGIC;
19               iaddr : out std_logic_vector(31 downto 0);
20               idata : in std_logic_vector(31 downto 0);
21               daddr : out std_logic_vector(31 downto 0);
22               ddatain : out std_logic_vector(31 downto 0);
23               ddataout : in std_logic_vector(31 downto 0);
24               oe : out std_logic;
25               we : out std_logic);
26 end component;
27
28 component imem
29 generic (
30     datafile: string := "program.asc" );
31     Port ( clk : in  STD_LOGIC;
32            reset : in  STD_LOGIC;
33               iaddr : in std_logic_vector(31 downto 0);
34               idata : out std_logic_vector(31 downto 0));
35 end component;
36
37 component dmem
38     Port ( clk : in  STD_LOGIC;
39            reset : in  STD_LOGIC;
40               daddr : in std_logic_vector(31 downto 0);
41               ddatain : in std_logic_vector(31 downto 0);
42               ddataout : out std_logic_vector(31 downto 0);
43               oe : in std_logic;
44               we : in std_logic);
45 end component;
46
47 signal i_iaddr_s : STD_LOGIC_VECTOR (31 downto 0);
48 signal i_idata_s : STD_LOGIC_VECTOR (31 downto 0);
49 signal m_daddr_s : STD_LOGIC_VECTOR (31 downto 0);
50 signal m_ddatain_s : STD_LOGIC_VECTOR (31 downto 0);
51 signal m_ddataout_s : STD_LOGIC_VECTOR (31 downto 0);
52 signal m_oe_s : STD_LOGIC;
53 signal m_we_s : STD_LOGIC;
54
55 begin
56
57
```

```
58 imem1 : imem generic map(
59               "program.asc")
60 port map(
61               clk => clk,
62               reset => reset,
63               iaddr => i_iaddr_s,
64               idata => i_idata_s
65 );
66
67
68 scpu0 : scpu port map(
69               clk => clk,
70               reset => reset,
71               iaddr => i_iaddr_s,
72               idata => i_idata_s,
73               daddr => m_daddr_s,
74               ddatain => m_ddatain_s,
75               ddataout => m_ddataout_s,
76               oe => m_oe_s,
77               we => m_we_s
78 );
79
80 dmem2 : dmem port map(
81               clk => clk,
82               reset => reset,
83               daddr => m_daddr_s,
84               ddatain => m_ddatain_s,
85               ddataout => m_ddataout_s,
86               oe => m_oe_s,
87               we => m_we_s
88 );
89
90
91
92 end Behavioral;
```

scpu_top.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use ieee.numeric_std.all;
5
6  entity ShiftImmLeft is
7      Port ( Extend_Imm : in  STD_LOGIC_VECTOR (31 downto 0);
8             ShiftLeft2 : out  STD_LOGIC_VECTOR (31 downto 0));
9  end ShiftImmLeft;
10
11 architecture Behavioral of ShiftImmLeft is
12
13 begin
14     ShiftLeft2 <= std_logic_vector(unsigned(Extend_Imm) sll 2);
15 end Behavioral;
```

ShiftImmLeft.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
```

```
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6  use IEEE.NUMERIC_STD.ALL;
7
8
9  entity SignExtend is
10     Port ( imm : in  STD_LOGIC_VECTOR (15 downto 0);
11            imm32 : out  STD_LOGIC_VECTOR (31 downto 0));
12 end SignExtend;
13
14 architecture Behavioral of SignExtend is
15
16 signal first_bit: STD_LOGIC;
17
18 begin
19     first_bit <= imm(15);
20     process(first_bit, imm)
21     begin
22         if (first_bit = '0') then
23             imm32 <= "0000000000000000" & imm;
24         else
25             imm32 <= "1111111111111111" & imm;
26         end if;
27     end process;
28
29
30 end Behavioral;
```

SignExtend.vhd

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6  entity WriteData_mux is
7      Port ( MemtoReg : in  STD_LOGIC;
8             ddataout : in  STD_LOGIC_VECTOR (31 downto 0);
9             ALUresult : in  STD_LOGIC_VECTOR (31 downto 0);
10            WriteData : out  STD_LOGIC_VECTOR (31 downto 0));
11 end WriteData_mux;
12
13 architecture Behavioral of WriteData_mux is
14
15 begin
16     WriteData <= ALUresult when (MemtoReg = '0') else
17                  ddataout;
18 end Behavioral;
```

WriteData_mux.vhd