

## Pc\_Unit

```
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity Pc is
33     Port ( input : in  STD_LOGIC_VECTOR (31 downto 0);
34           output : out STD_LOGIC_VECTOR (31 downto 0);
35           CLK : in  STD_LOGIC);
36 end Pc;
37
38 architecture Behavioral of Pc is
39     signal temp: STD_LOGIC_VECTOR(31 downto 0) := X"00000000";
40
41 begin
42     process (input ,CLK)
43     begin
44
45         if rising_edge(CLK) then
46             output <= temp;
47         end if;
48         if falling_edge(CLK) then
49             temp <= input;
50         end if;
51
52     end process;
53 end Behavioral;
54
55
```

## Pc\_Adder

```
29 --library UNISIM;
30 --library UNISIM;
31 --use UNISIM.VComponents.all;
32
33 entity Pc_Adder is
34     Port ( i1 : in  STD_LOGIC_VECTOR (31 downto 0);
35           i2 : in  STD_LOGIC_VECTOR (31 downto 0);
36           output : out STD_LOGIC_VECTOR (31 downto 0));
37 end Pc_Adder;
38
39 architecture Behavioral of Pc_Adder is
40
41 begin
42
43     output <= i1 + i2;
44
45 end Behavioral;
46
47
```

## Branch\_Adder

```
32
33 entity Adder is
34     Port ( ip1 : in  STD_LOGIC_VECTOR (31 downto 0);
35           ip2 : in  STD_LOGIC_VECTOR (31 downto 0);
36           op  : out STD_LOGIC_VECTOR (31 downto 0));
37 end Adder;
38
39 architecture Behavioral of Adder is
40
41 begin
42
43     op <= ip1 + ip2;
44
45 end Behavioral;
46
```

## Instruction\_Memory

```
31
32 entity Instruction_Memory is
33     Port ( pc : in  STD_LOGIC_VECTOR (31 downto 0);
34           instruction : out STD_LOGIC_VECTOR (31 downto 0);
35           CLK: in  STD_LOGIC
36           );
37 end Instruction_Memory;
38
39 architecture Behavioral of Instruction_Memory is
40
41 type arr is array(0 to 23) of STD_LOGIC_VECTOR (7 downto 0);
42
43 signal memory: arr:= (
44
45     "00000000", "10000101", "00010000", "00100000", -- add $v0, $a0, $a1
46     "10101100", "00000010", "00000000", "00001000", -- sw $v0, 8($Zero)
47     "10001100", "00000110", "00000000", "00001000", -- lw $a2, 8($Zero)
48     "00010000", "11000010", "00000000", "00000001", -- beq $v0, $a2, 1
49     "00000000", "01000110", "10001000", "00101010", -- slt $s1, $v0, $a2
50     "00000000", "10100100", "10001000", "00100010" -- sub $s1, $a1, $a0
51 );
52
53 begin
54
55
56     instruction(31 downto 24) <= memory(to_integer(unsigned(pc)));
57     instruction(23 downto 16) <= memory(to_integer(unsigned(pc)+1));
58     instruction(15 downto 8) <= memory(to_integer(unsigned(pc)+2));
59     instruction(7 downto 0) <= memory(to_integer(unsigned(pc)+3));
60
61
62 end Behavioral;
63
```

## Control\_Unit

```
31 entity control is
32   Port ( Operation : in  STD_LOGIC_VECTOR (5 downto 0);
33         RegDst : out  STD_LOGIC;
34         AluSrc : out  STD_LOGIC;
35         Memoryreg : out STD_LOGIC;
36         RegWrite : out STD_LOGIC;
37         MemRead : out  STD_LOGIC;
38         MemWrite : out  STD_LOGIC;
39         Branch : out  STD_LOGIC;
40         AluOp : out  STD_LOGIC_VECTOR (1 downto 0));
41 end control;
42
43 architecture Behavioral of control is
44 begin
45   process(Operation)
46   begin
47     if Operation = "000000" then
48       RegDst <= '1';
49       AluSrc <= '0';
50       Memoryreg <= '0';
51       RegWrite <= '1';
52       MemRead <= '0';
53       MemWrite <= '0';
54       Branch <= '0';
55       AluOp <= "10";
56     elsif Operation = "100011" then
57       RegDst <= '0';
58       AluSrc <= '1';
59       Memoryreg <= '1';
60       RegWrite <= '1';
61       MemRead <= '1';
62       MemWrite <= '0';
63       Branch <= '0';
64       AluOp <= "00";
65     elsif Operation = "101011" then
66       RegDst <= '1';
67       AluSrc <= '1';
68       Memoryreg <= '1';
69       RegWrite <= '0';
70       MemRead <= '0';
71       MemWrite <= '1';
72       Branch <= '0';
73       AluOp <= "00";
74     elsif Operation <= "000100" then
75       RegDst <= '1';
76       AluSrc <= '0';
77       Memoryreg <= '1';
78       RegWrite <= '0';
79       MemRead <= '0';
80       MemWrite <= '0';
81       Branch <= '1';
82       AluOp <= "01";
83     end if;
84   end process;
```

## Mux\_RegDst

```
1 entity MemReg_Mux is
2   Port ( i1 : in  STD_LOGIC_VECTOR (4 downto 0);
3         i2 : in  STD_LOGIC_VECTOR (4 downto 0);
4         sel_line : in  STD_LOGIC;
5         output : out STD_LOGIC_VECTOR (4 downto 0));
6 end MemReg_Mux;
7
8 architecture Behavioral of MemReg_Mux is
9 begin
10   process (i1,i2,sel_line)
11   begin
12     if sel_line = '0' then
13       output <= i1;
14     else
15       output <= i2;
16     end if;
17   end process;
18 end Behavioral;
```

## Mux\_Memory

```
31
32 entity Memory_Mux is
33     Port ( i1 : in  STD_LOGIC_VECTOR (31 downto 0);
34           i2 : in  STD_LOGIC_VECTOR (31 downto 0);
35           sel_line : in  STD_LOGIC;
36           output : out  STD_LOGIC_VECTOR (31 downto 0));
37 end Memory_Mux;
38
39 architecture Behavioral of Memory_Mux is
40
41 begin
42     process (i1,i2,sel_line)
43     begin
44         if sel_line <= '0' then
45             output <= i1;
46         else
47             output <= i2;
48         end if;
49     end process;
50 end Behavioral;
51
```

## Mux\_Alu

```
1
2 entity Alu_Mux is
3     Port ( i1 : in  STD_LOGIC_VECTOR (31 downto 0);
4           i2 : in  STD_LOGIC_VECTOR (31 downto 0);
5           sel_line : in  STD_LOGIC;
6           output : out  STD_LOGIC_VECTOR (31 downto 0));
7 end Alu_Mux;
8
9 architecture Behavioral of Alu_Mux is
10
11 begin
12     process (i1,i2,sel_line)
13     begin
14         if sel_line = '0' then
15             output <= i1;
16         else
17             output <= i2;
18         end if;
19     end process;
20 end Behavioral;
21
```

## Mux\_Pc

```
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity Pc_Mux is
33     Port ( i1 : in  STD_LOGIC_VECTOR (31 downto 0);
34           i2 : in  STD_LOGIC_VECTOR (31 downto 0);
35           sel_line : in  STD_LOGIC;
36           output : out  STD_LOGIC_VECTOR (31 downto 0));
37 end Pc_Mux;
38
39 architecture Behavioral of Pc_Mux is
40
41 begin
42     process (i1 , i2 ,sel_line)
43     begin
44         if(sel_line = '0') then
45             output <= i1;
46         elsif(sel_line = '1') then
47             output <= i2;
48         end if;
49     end process;
50 end Behavioral;
51
52
```

## Register\_File

```
31
32 entity RegFile is
33     Port ( read_Reg1 : in  STD_LOGIC_VECTOR (4 downto 0);
34           read_Reg2 : in  STD_LOGIC_VECTOR (4 downto 0);
35           write_Reg  : in  STD_LOGIC_VECTOR (4 downto 0);
36           write_data : in  STD_LOGIC_VECTOR (31 downto 0);
37           read_data1 : out STD_LOGIC_VECTOR (31 downto 0);
38           read_data2 : out STD_LOGIC_VECTOR (31 downto 0);
39           CLK: in  STD_LOGIC;
40           RegWrite   : in  STD_LOGIC);
41 end RegFile;
42
43 architecture Behavioral of RegFile is
44
45     type RegFileType is array (0 to 31) of STD_LOGIC_VECTOR (31 downto 0);
46
47     signal array_reg : RegFileType := ( X"00000000", X"00000000", X"00000000", X"00000000",
48                                         X"00000005", X"00000007", X"00000000", X"00000000",
49                                         X"00000000", X"00000000", X"00000000", X"00000000",
50                                         X"00000000", X"00000000", X"00000000", X"00000000",
51                                         X"00000000", X"00000000", X"00000000", X"00000000",
52                                         X"00000000", X"00000000", X"00000000", X"00000000",
53                                         X"00000000", X"00000000", X"00000000", X"00000000",
54                                         X"00000000", X"00000000", X"00000000", X"00000000");
55
56
57 begin
58     read_data1 <= array_reg(to_integer(unsigned(read_Reg1)));
59     read_data2 <= array_reg(to_integer(unsigned(read_Reg2)));
60
61     process (RegWrite,CLK)
62     begin
63         if (RegWrite = '1' and rising_edge(CLK)) then
64             array_reg(to_integer(unsigned(write_Reg))) <= write_data;
65
66         end if;
67     end process;
68 end Behavioral;
69
```

## Sign\_Extend

```
entity Sign_Extend is
    Port ( input : in  STD_LOGIC_VECTOR (15 downto 0);
          output : out STD_LOGIC_VECTOR (31 downto 0));
end Sign_Extend;

architecture Behavioral of Sign_Extend is

begin

    process (input)
    begin

        output(15 downto 0) <= input(15 downto 0);

        if input(15) = '0' then
            output(31 downto 16) <= X"0000";

        elsif input(15) = '1' then
            output(31 downto 16) <= X"FFFF";

        end if;
    end process;

end Behavioral;
```

## Shift\_Left\_2

```
entity shift_left_32_32 is
    Port ( input : in  STD_LOGIC_VECTOR (31 downto 0);
          output : out STD_LOGIC_VECTOR (31 downto 0));
end shift_left_32_32;

architecture Behavioral of shift_left_32_32 is

begin

    output(31 downto 2) <= input(29 downto 0);
    output(1 downto 0) <= "00";

end Behavioral;
```

## Alu\_Control

```
1 entity Alu_Control is
2     Port ( AluOp : in  STD_LOGIC_VECTOR (1 downto 0);
3           Instruction : in  STD_LOGIC_VECTOR (5 downto 0);
4           operation : out STD_LOGIC_VECTOR (3 downto 0));
5 end Alu_Control;
6
7 architecture Behavioral of Alu_Control is
8
9 begin
10
11 process(AluOp, Instruction)
12 begin
13     if(AluOp = "00" )then
14         operation <= "0010";
15     elsif AluOp = "01" then
16         operation <= "0110";
17     elsif AluOp = "10" then
18         if Instruction(3 downto 0) = "0000" then
19             operation <= "0010";
20         elsif Instruction(3 downto 0) = "0100" then
21             operation <= "0000";
22         elsif Instruction(3 downto 0) = "0101" then
23             operation <= "0001";
24
25         -
26         elsif AluOp(1) = '1' then
27             if Instruction(3 downto 0) = "0010" then
28                 operation <= "0110";
29             elsif Instruction(3 downto 0) = "1010" then
30                 operation <= "0111";
31             end if;
32         end if;
33     elsif AluOp(1) = '1' then
34         if Instruction(3 downto 0) = "0010" then
35             operation <= "0110";
36         elsif Instruction(3 downto 0) = "1010" then
37             operation <= "0111";
38         end if;
39     end if;
40 end process;
```

## Alu\_Unit

```
33
34 entity ALU is
35     Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
36           B : in  STD_LOGIC_VECTOR (31 downto 0);
37           AluOperation : in  STD_LOGIC_VECTOR (3 downto 0);
38           Zero : out  STD_LOGIC;
39           res : out  STD_LOGIC_VECTOR (31 downto 0));
40 end ALU;
41
42 architecture Behavioral of ALU is
43
44 begin
45 process (A,B,AluOperation)
46 begin
47     if AluOperation = "0000" then
48         res <= A AND B;
49     elsif AluOperation = "0001" then
50         res <= A OR B;
51     elsif AluOperation = "0111" then
52         if A < B then
53             res <= X"00000001";
54         else res <= X"00000000";
55         end if;
56     end if;
57     elsif AluOperation = "0010" then
58         res <= A + B;
59     elsif AluOperation = "0110" then
60         res <= A - B;
61     elsif AluOperation = "1100" then
62         res <= A NOR B;
63     end if;
64 if A = B then
65     Zero <= '1';
66 else
67     Zero <= '0';
68 end if;
69 end process;
70 end Behavioral;
```

## Data\_Memory

```
31
32 entity Data_Memory is
33     Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
34           write_Data : in  STD_LOGIC_VECTOR (31 downto 0);
35           read_Data : out STD_LOGIC_VECTOR (31 downto 0);
36           mem_read : in  STD_LOGIC;
37           mem_write : in  STD_LOGIC;
38           CLK: in  STD_LOGIC);
39 end Data_Memory;
40
41 architecture Behavioral of Data_Memory is
42
43 type Data_Memory_Type is array (0 to 35) of STD_LOGIC_VECTOR (7 downto 0);
44
45 signal array_Memory : Data_Memory_Type := (
46     X"AB", X"CD", X"EF", X"00",
47     X"75", X"74", X"65", X"72",
48     X"20", X"41", X"72", X"63",
49     X"68", X"69", X"74", X"65",
50     X"12", X"34", X"56", X"78",
51     X"7F", X"7F", X"7D", X"7D",
52     X"00", X"00", X"00", X"00",
53     X"78", X"78", X"6A", X"6A",
54     X"00", X"00", X"00", X"01");
55 begin
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
```

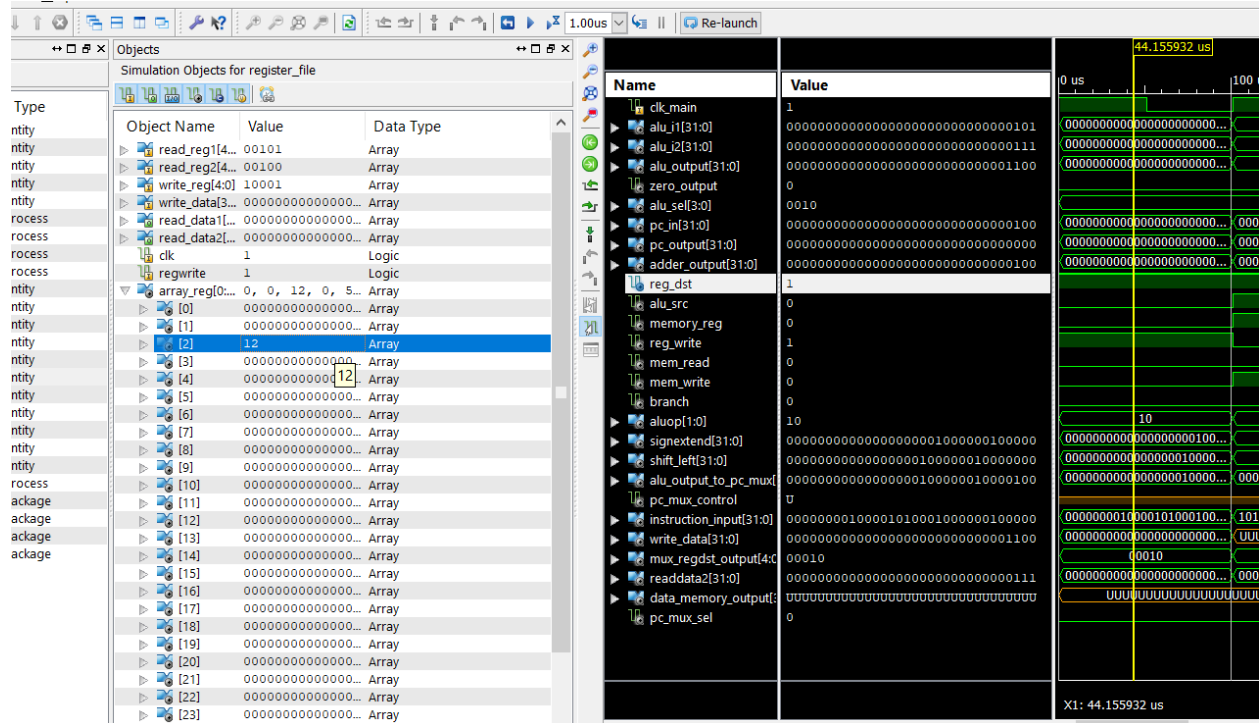
```

34
55
56
57 process(Address, mem_read, mem_write,CLK)
58 begin
59     if(mem_read = '1' and mem_write = '0' ) then
60         read_Data(31 downto 24) <= array_Memory(to_integer(unsigned(Address)));
61         read_Data(23 downto 16) <= array_Memory(to_integer(unsigned(Address)+1));
62         read_Data(15 downto 8) <= array_Memory(to_integer(unsigned(Address)+2));
63         read_Data(7 downto 0) <= array_Memory(to_integer(unsigned(Address)+3));
64
65
66         elsif(mem_read = '0' and mem_write = '1' and rising_edge(CLK) ) then
67             array_Memory(to_integer(unsigned(Address))) <= write_Data(31 downto 24);
68             array_Memory(to_integer(unsigned(Address)+1)) <= write_Data(23 downto 16);
69             array_Memory(to_integer(unsigned(Address)+2)) <= write_Data(15 downto 8);
70             array_Memory(to_integer(unsigned(Address)+3)) <= write_Data(7 downto 0);
71         end if;
72
73 end process;
74 end Behavioral;
75
```

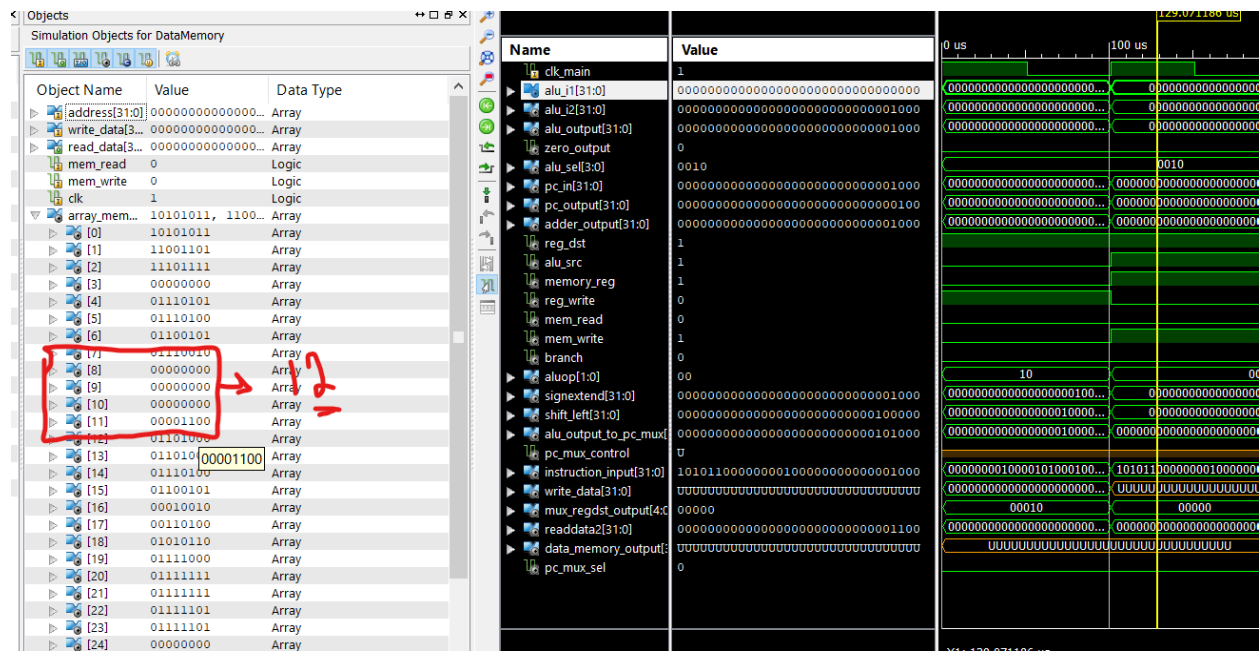


## ADD \$v0, \$a0, \$a1

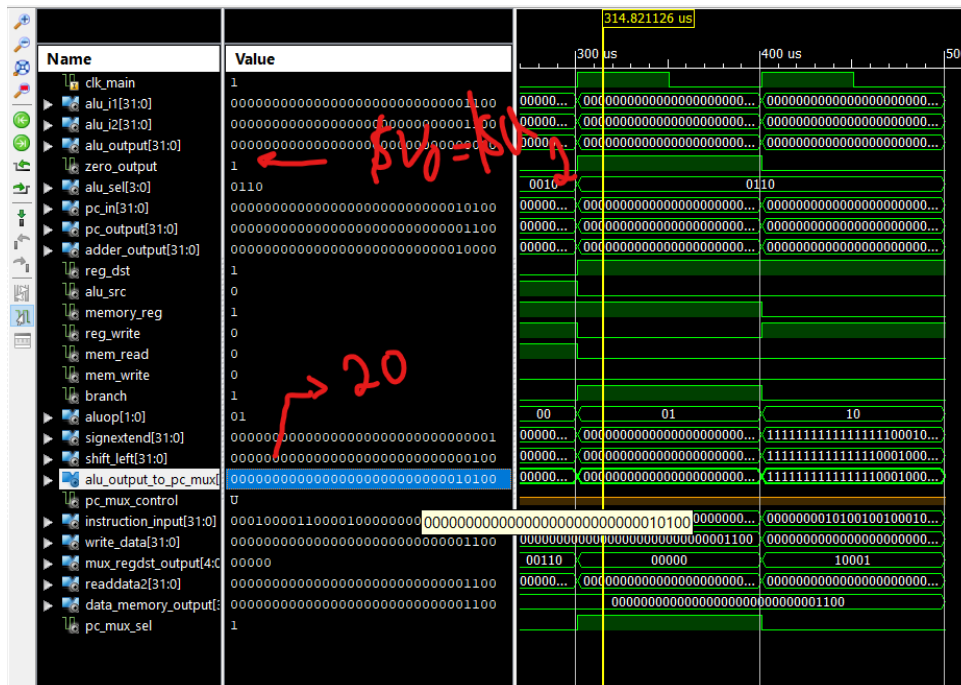
**\$v0 = 12**



**SW \$V0 , 8(\$ZERO)**



**\$a2 = 12**



Sub \$s1, \$a1, \$a0

$\$a1 - \$a0 = 7 - 5 = 2$

$\$s1 = 2$

Simulation Objects for register_file				
Object Name	Value	Data Type	Name	Value
clk	1	Logic	clk_main	1
regwrite	1	Logic	alu_i1[31:0]	00000000000000000000000000000111
array_reg[0]...	0, 0, 12, 0, 5...	Array	alu_i2[31:0]	00000000000000000000000000000101
[0]	0000000000000000...	Array	alu_output[31:0]	00000000000000000000000000000010
[1]	0000000000000000...	Array	zero_output	0
[2]	12	Array	alu_sel[3:0]	0110
[3]	0000000000000000...	Array	pc_in[31:0]	000000000000000000000000000011000
[4]	0000000000000000...	Array	pc_output[31:0]	000000000000000000000000000010100
[5]	0000000000000000...	Array	adder_output[31:0]	000000000000000000000000000011000
[6]	12	Array	reg_dst	1
[7]	0000000000000000...	Array	alu_src	0
[8]	0000000000000000...	Array	memory_reg	0
[9]	0000000000000000...	Array	reg_write	1
[10]	0000000000000000...	Array	mem_read	0
[11]	0000000000000000...	Array	mem_write	0
[12]	0000000000000000...	Array	branch	0
[13]	0000000000000000...	Array	aluop[1:0]	10
[14]	0000000000000000...	Array	signextend[31:0]	11111111111111111000100000100010
[15]	0000000000000000...	Array	shift_left[31:0]	1111111111111111100010000010001000
[16]	0000000000000000...	Array	alu_output_to_pc_mux[	1111111111111111100010000010100000
[17]	2	Array	pc_mux_control	0
[18]	0000000000000000...	Array	instruction_input[31:0]	00000000101001001000100000100010
[19]	0000000000000000...	Array	write_data[31:0]	00000000000000000000000000000010
[20]	0000000000000000...	Array	mux_regdst_output[4:0]	10001
[21]	0000000000000000...	Array	readdata2[31:0]	00000000000000000000000000000101
[22]	0000000000000000...	Array	data_memory_output[31:0]	00000000000000000000000000001100
[23]	0000000000000000...	Array	pc_mux_sel	0
[24]	0000000000000000...	Array		
[25]	0000000000000000...	Array		
[26]	0000000000000000...	Array		