

FINAL Laboratory Project

Ahmad Adil

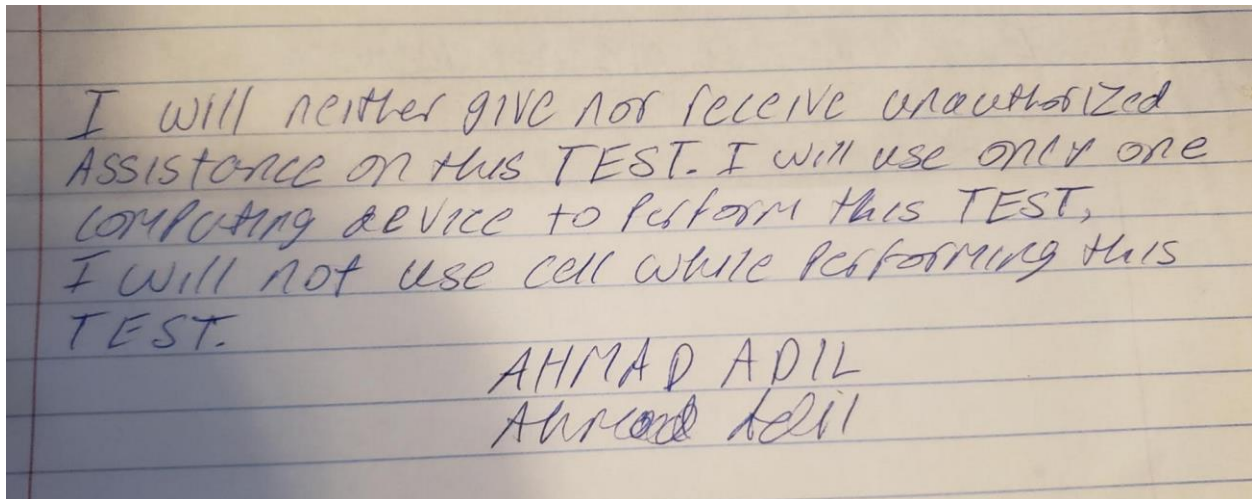
7/22/21

Professor Gertner

Computer Organization 342/343

Table of Contents

Objective:	2
Registers:	3
[ADD, ADDU, SUB, SUBU, AND, NOR, OR]	4
[ADDI, ADDIU, ANDI, ORI]	9
Load Word (LW)	15
Store Word (SW)	18
Conclusion	22



I will neither give nor receive unauthorized assistance on this TEST. I will use only one computing device to perform this TEST. I will not use cell while performing this TEST.

AHMAD ADIL
Ahmad Adil

Objective:

The main objective of this project was that we needed to build an ALU. An arithmetic logic unit (ALU) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers. The inputs within a ALU are what are being operated on, which are called operands. The ALU can also have flags attached to it that tell it overflow, zero and negative. We designed this ALU in three separate

parts, one for the R_TYPE INSTRUCTIONS. Then next for I arithmetic logic TYPE INSTRUCTIONS and finally for MEMORY ACCESS INSTRUCTIONS, for both load word and store word.

Registers:

16-bit Registers:

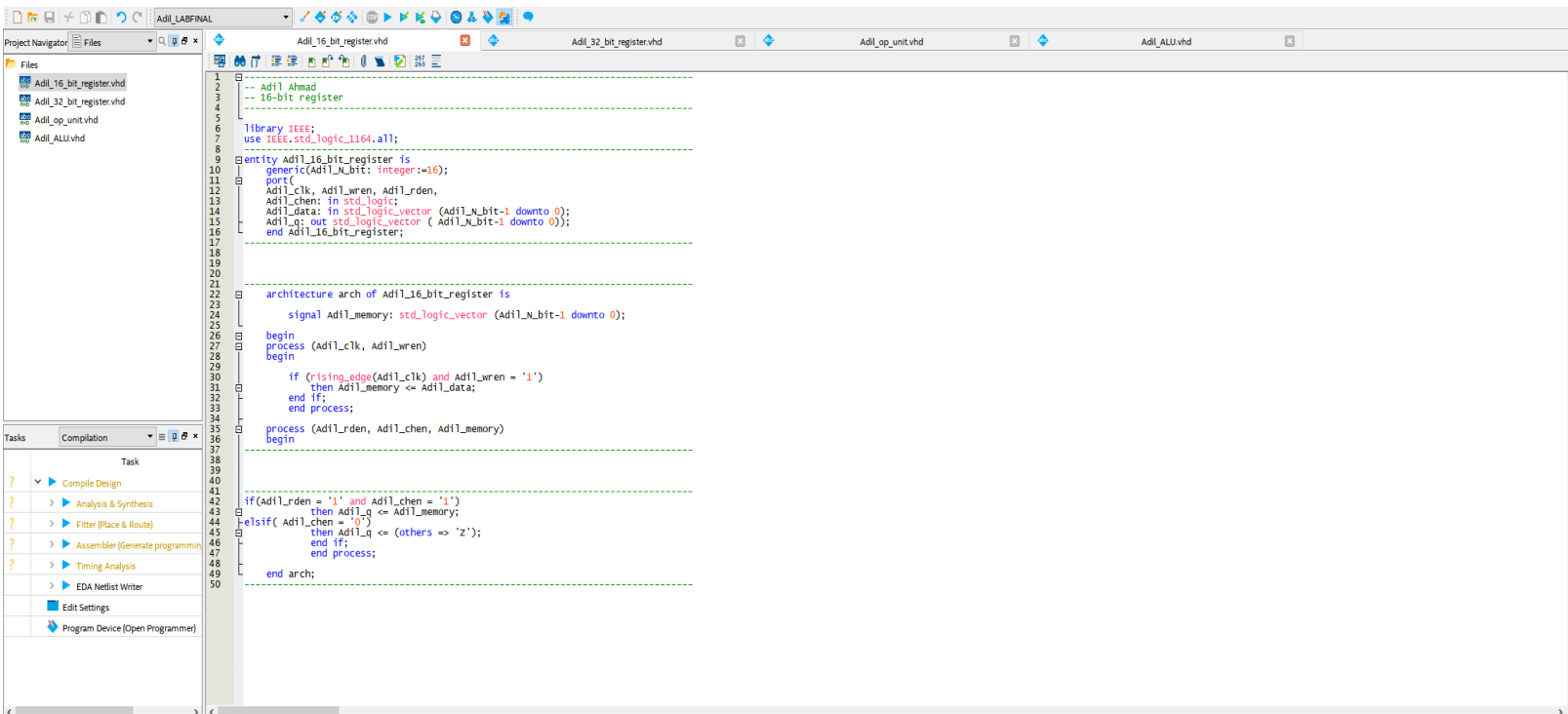


Figure 1: 16-bit Registers

```

1  -- Adil Ahmad
2  -- 32-bit register
3
4
5
6
7
8
9  library IEEE;
10 use IEEE.std_logic_1164.all;
11
12 entity Adil_32_bit_register is
13   generic(Adil_N_bit: integer:=32);
14   port(
15     Adil_clk, Adil_wren, Adil_rden,
16     Adil_chen: in std_logic;
17     Adil_data: in std_logic_vector (Adil_N_bit-1 downto 0);
18     Adil_q: out std_logic_vector (Adil_N_bit-1 downto 0));
19 end Adil_32_bit_register;
20
21 architecture arch of Adil_32_bit_register is
22   signal Adil_memory: std_logic_vector (Adil_N_bit-1 downto 0);
23
24   begin
25     process (Adil_clk, Adil_wren)
26     begin
27       if (rising_edge(Adil_clk) and Adil_wren = '1')
28       then Adil_memory <= Adil_data;
29       end if;
30     end process;
31
32     process (Adil_rden, Adil_chen, Adil_memory)
33     begin
34       if(Adil_rden = '1' and Adil_chen = '1')
35       then Adil_q <= Adil_memory;
36       elsif( Adil_chen = '0')
37       then Adil_q <= (others => 'Z');
38       end if;
39     end process;
40   end arch;
41
42
43
44
45
46

```

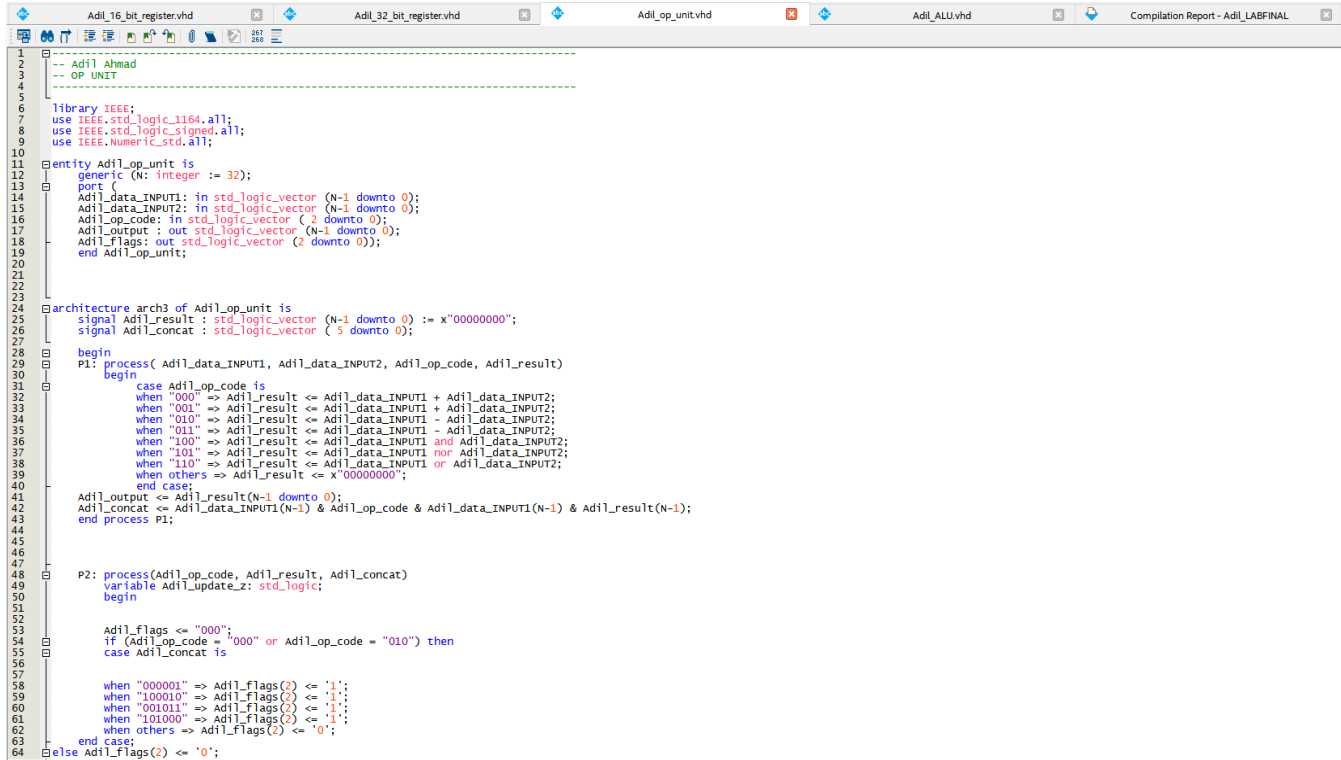
Figure 2 32-bit register

This figure above shows the initial 16-bit register that will be used in the project. This is used to build the immediate register.

This figure above 32-bit register component to build RS, RT, RD, MAR and MDR registers. Flags are a vector of 3 bits. V, N, and Z which are zero flag if the bit 0, negative flag for if bit 1, and overflow flag for if bit 2.

[ADD, ADDU, SUB, SUBU, AND, NOR, OR]

OP VHDL code:



```

1  -- Adil Ahmad
2  -- OP UNIT
3
4
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8  use IEEE.std_logic_signed.all;
9  use IEEE.Numeric_STD.all;
10
11 entity Adil_OP is
12     generic (N: integer := 32);
13     port (
14         Adil_data_INPUT1: in std_logic_vector (N-1 downto 0);
15         Adil_data_INPUT2: in std_logic_vector (N-1 downto 0);
16         Adil_op_code: in std_logic_vector (2 downto 0);
17         Adil_output : out std_logic_vector (N-1 downto 0);
18         Adil_flags: out std_logic_vector (2 downto 0);
19     end Adil_OP;
20
21
22
23 architecture arch3 of Adil_OP is
24     signal Adil_result : std_logic_vector (N-1 downto 0) := x"00000000";
25     signal Adil_concat : std_logic_vector (5 downto 0);
26
27     begin
28
29     P1: process( Adil_data_INPUT1, Adil_data_INPUT2, Adil_op_code, Adil_result)
30     begin
31         case Adil_op_code is
32             when "000" => Adil_result <= Adil_data_INPUT1 + Adil_data_INPUT2;
33             when "001" => Adil_result <= Adil_data_INPUT1 + Adil_data_INPUT2;
34             when "010" => Adil_result <= Adil_data_INPUT1 - Adil_data_INPUT2;
35             when "011" => Adil_result <= Adil_data_INPUT1 - Adil_data_INPUT2;
36             when "100" => Adil_result <= Adil_data_INPUT1 and Adil_data_INPUT2;
37             when "101" => Adil_result <= Adil_data_INPUT1 nor Adil_data_INPUT2;
38             when "110" => Adil_result <= Adil_data_INPUT1 or Adil_data_INPUT2;
39             when others => Adil_result <= x"00000000";
40         end case;
41         Adil_output <= Adil_result(N-1 downto 0);
42         Adil_concat <= Adil_data_INPUT1(N-1) & Adil_op_code & Adil_data_INPUT1(N-1) & Adil_result(N-1);
43     end process P1;
44
45
46
47
48     P2: process(Adil_op_code, Adil_result, Adil_concat)
49     variable Adil_update_z: std_logic;
50     begin
51
52         Adil_flags <= "000";
53         if (Adil_op_code = "000" or Adil_op_code = "010") then
54             case Adil_concat is
55                 when "000001" => Adil_flags(2) <= '1';
56                 when "100010" => Adil_flags(2) <= '1';
57                 when "001011" => Adil_flags(2) <= '1';
58                 when "101000" => Adil_flags(2) <= '1';
59                 when others => Adil_flags(2) <= '0';
60             end case;
61         else
62             Adil_flags(2) <= '0';
63         end if;
64     end process P2;
65

```

Figure 3: VHDL code for OP

```

26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84

-- Adil Ahmad
-- ALU UNIT

library IEEE;
use IEEE.std_logic_1164.all;

entity Adil_ALU is
    generic(N: integer := 32);
    port(
        Adil_clk: in std_logic;
        Adil_RS: in std_logic_vector(N-1 downto 0);
        Adil_RT: in std_logic_vector(N-1 downto 0);
        Adil_op_code: in std_logic_vector(2 downto 0);
        Adil_rd: out std_logic_vector(N-1 downto 0);
        Adil_flags: out std_logic_vector(2 downto 0));
end Adil_ALU;

architecture arch4 of Adil_ALU is
    component Adil_32_bit_register is
        generic (Adil_N_bit: integer := 32);
        port(
            Adil_clk: in std_logic;
            Adil_wren: in std_logic;
            Adil_rden: in std_logic;
            Adil_chen: in std_logic;
            Adil_data: in std_logic_vector(N-1 downto 0);
            Adil_q: out std_logic_vector(N-1 downto 0));
    end component;

    component Adil_op_unit is
        generic (N: integer := 32);
        port(
            Adil_data_in1: in std_logic_vector(N-1 downto 0);
            Adil_data_in2: in std_logic_vector(N-1 downto 0);
            Adil_op_code: in std_logic_vector(2 downto 0);
            Adil_output: out std_logic_vector(N-1 downto 0);
            Adil_flags: out std_logic_vector(2 downto 0));
    end component;

    signal Adil_RS_OUT, Adil_RT_OUT, Adil_RD_IN: std_logic_vector(N-1 downto 0);

    -- N
    if(Ac)
    begin
        D1: Adil_32_bit_register port map(Adil_clk, '1', '1', '1', Adil_RS, Adil_RS_OUT);
        D2: Adil_32_bit_register port map(Adil_clk, '1', '1', '1', Adil_RT, Adil_RT_OUT);
        D3: Adil_op_unit port map(Adil_RS_OUT, Adil_RT_OUT, Adil_op_code, Adil_RD_IN, Adil_flags);
        D4: Adil_32_bit_register port map(Adil_clk, '1', '1', '1', Adil_RD_IN, Adil_RD);
    end arch4;
end arch4;

```

Figure 4: OP unit continued

Figure 5: ALU code

In this part of the program, we use equation $R[rd] = R[rs]$ operation $R[rt]$ for the arithmetic and logical operations of registers.

Waveforms:

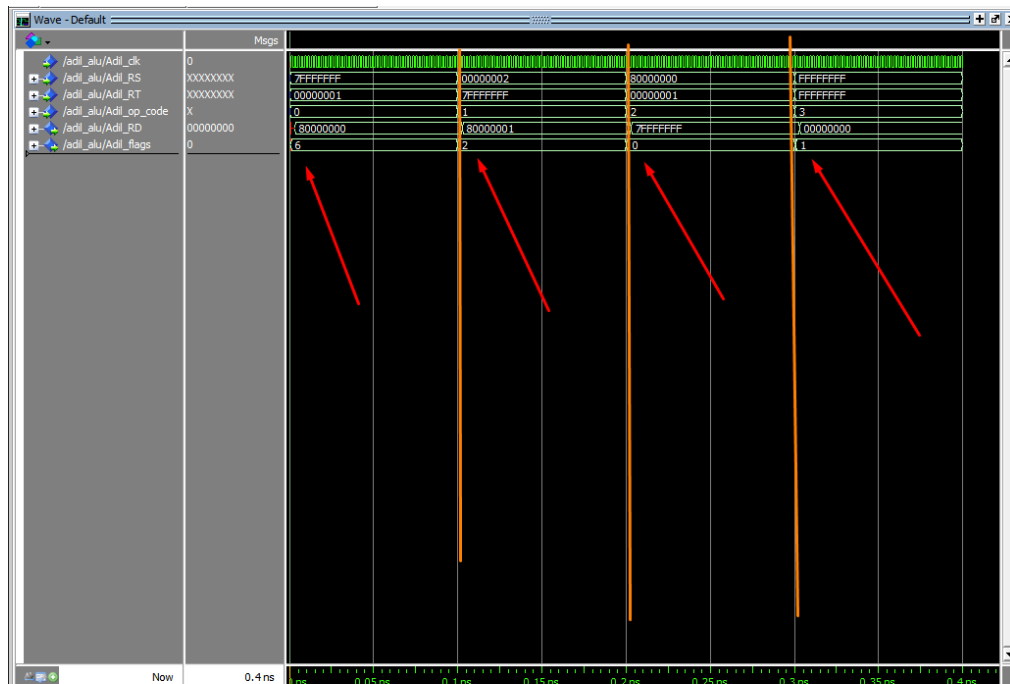


Figure 6: waveforms add/sub

This here is the first waveform, which is the addition and subtraction shown for the first equation. For this waveform, the operation codes that were set were, 0000 for ADD, 0001 for ADDU, 0010 for SUB, and 0011 for SUBU. For the first one, $0x7FFFFFFF + 1 = 0x80000000$. This will set the flags for the overflow and negative flags. Then for the second one, $2 + 0x7FFFFFFF = 0x80000001$ using addu, will set the negative. However, the overflow flag will be ignored. Then negative - 1 = $0x7FFFFFFF$ sets the V flag since adding 2 negative must be negative. Finally, doing subtraction on two same numbers = 0, which sets the zero flag correctly.

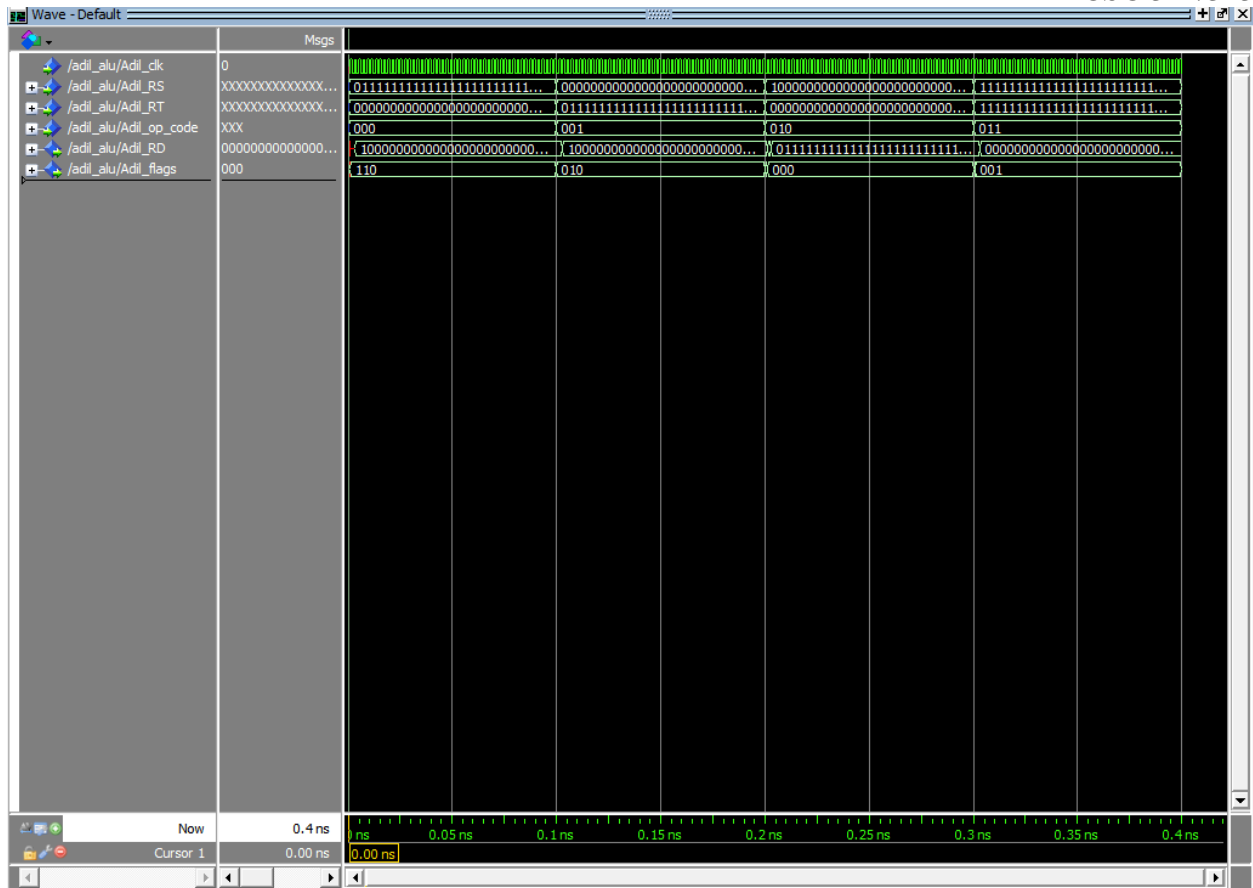


Figure 7: AND

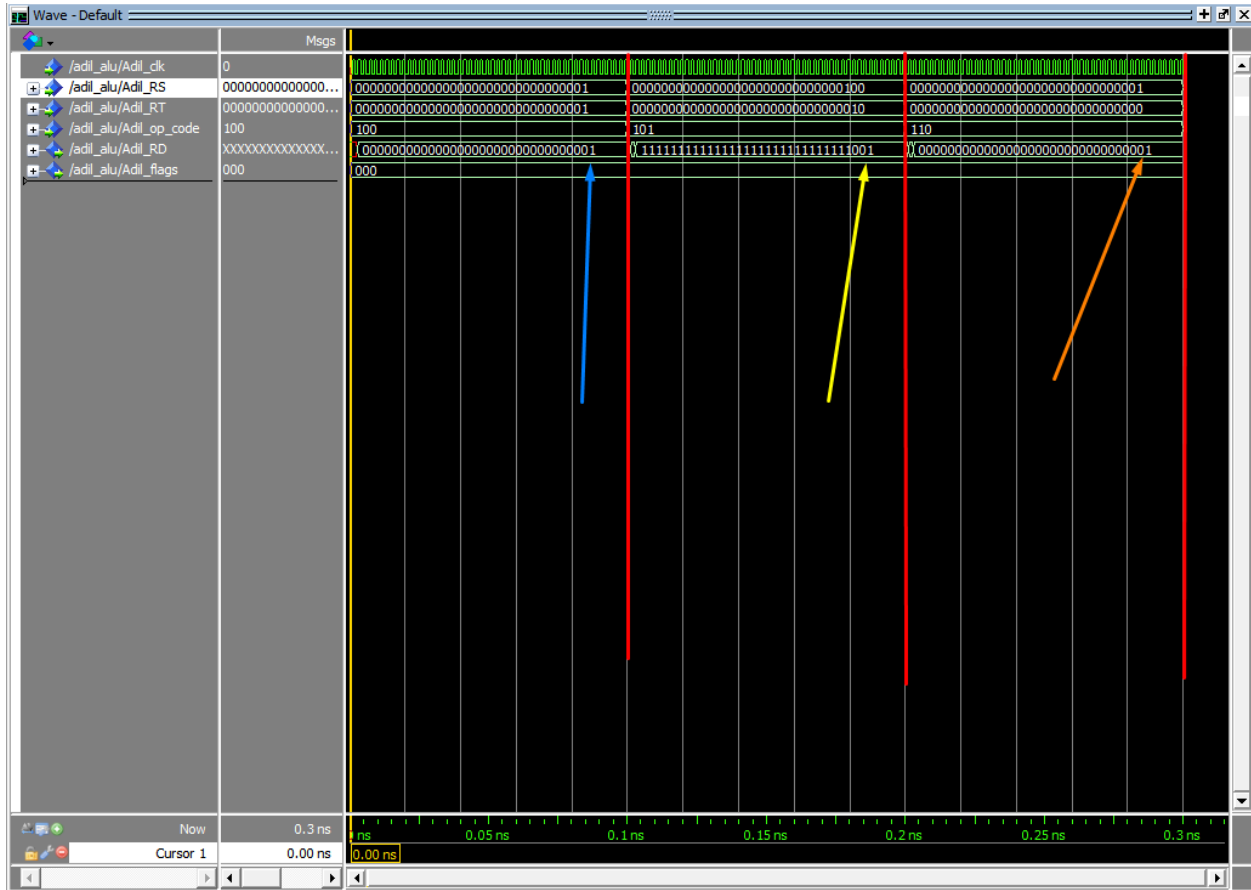


Figure 8: AND/NOR/OR

Within this waveform simulation, the op codes 0100 for AND, 0101 for NOR, and 0110 for OR. RD has value where bits are set to 1 only when both RS and RT have '1' in their bits when using AND instruction. For the NOR we can see that bits are 1 in RD only on, 0 or 0, bit locations. For OR, 1110 OR 0001 = 1111 This can show us that the answer is when if either RS or RT has 1. For Bitwise operations, flags are not affected.

[ADDI, ADDIU, ANDI, ORI]

```

1  -----
2  -- Adil Ahmad
3  -- operation code updated
4  -----
5
6
7
8  library IEEE;
9  use IEEE.std_logic_1164.all;
10 use IEEE.std_logic_signed.all;
11 use IEEE.Numeric_std.all;
12
13 entity Adil_op_unit_updated is
14   generic (Adil_N_bit: integer := 32);
15   port
16     Adil_INPUT1 : in std_logic_vector (Adil_N_bit-1 downto 0);
17     Adil_INPUT2 : in std_logic_vector (Adil_N_bit-1 downto 0);
18     Adil_imm : in std_logic_vector (13 downto 0);
19     Adil_extend : in std_logic_vector (Adil_N_bit-1 downto 0);
20     Adil_code : in std_logic_vector (3 downto 0);
21     Adil_imm_inst_out : out std_logic_vector (Adil_N_bit-1 downto 0);
22     Adil_output : out std_logic_vector (Adil_N_bit-1 downto 0);
23     Adil_flags : out std_logic_vector (2 downto 0);
24   );
25 end Adil_op_unit_updated;
26
27 architecture arch3 of Adil_op_unit_updated is
28   signal Adil_result : std_logic_vector (Adil_N_bit-1 downto 0) := x"00000000";
29   signal Adil_concat : std_logic_vector (6 downto 0);
30
31   begin
32   P1: process (Adil_INPUT1, Adil_INPUT2, Adil_code, Adil_result, Adil_imm, Adil_extend)
33     begin
34       case Adil_code is
35         when "0000" => Adil_result <= Adil_INPUT1 + Adil_INPUT2;
36         when "0001" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; Adil_extend <= Adil_extend; --addi
37         when "0010" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; Adil_extend <= Adil_extend; --addui
38         when "0011" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --addu
39         when "0100" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --sub
40         when "0101" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --subu
41         when "0110" => Adil_result <= Adil_INPUT1 and Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --and
42         when "0111" => Adil_result <= Adil_INPUT1 and Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --andi
43         when "1000" => Adil_result <= Adil_INPUT1 nor Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --nor
44         when "1001" => Adil_result <= Adil_INPUT1 or Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --ori
45         when "1010" => Adil_result <= Adil_INPUT1 or Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --ori
46         when others => Adil_result <= x"00000000";
47       end case;
48
49       if (Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
50       then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
51       else
52         Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
53       end if;
54       Adil_concat <= Adil_INPUT1(Adil_N_bit-1) & Adil_code & (Adil_INPUT2(Adil_N_bit-1) or Adil_extend(Adil_N_bit-1)) & Adil_result(Adil_N_bit-1);
55
56       --Adil_output <= Adil_result(N-1 downto 0);
57       --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT2(N-1) & Adil_result(N-1);
58     end process P1;
59
60   P2: process (Adil_code, Adil_result, Adil_concat)
61     variable Adil_update_z: std_logic;
62     begin
63       Adil_flags <= "000";
64
65       if (Adil_code = "0000" or Adil_code = "0001" or Adil_code = "0100") then
66         case Adil_concat is
67           when "0000001" => Adil_flags(2) <= '1';
68           when "1000010" => Adil_flags(2) <= '1';
69           when "0010011" => Adil_flags(2) <= '1';
70           when "1010000" => Adil_flags(2) <= '1';
71           when "0000101" => Adil_flags(2) <= '1';
72           when "1000110" => Adil_flags(2) <= '1';
73           when others => Adil_flags(2) <= '0';
74         end case;
75       else Adil_flags(2) <= '0';
76       end if;
77
78       -- N flag
79       if (Adil_code = "0000" or Adil_code = "0001" or
80         Adil_code = "0010" or Adil_code = "0011" or
81         Adil_code = "0100" or Adil_code = "0101")
82       then
83         Adil_flags(1) <= Adil_result(Adil_N_bit-1);
84       end if;
85
86       -- Z flag
87       Adil_update_z := '0';
88       for i in 0 to (Adil_N_bit-1) loop
89         Adil_update_z := Adil_update_z or Adil_result(i);
90       end loop;
91       Adil_flags(0) <= not Adil_update_z;
92     end process P2;
93

```

Figure 9 OP code

```

35 when "0000" => Adil_result <= Adil_INPUT1 + Adil_INPUT2;
36 when "0001" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; Adil_extend <= Adil_extend; --addi
37 when "0010" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; Adil_extend <= Adil_extend; --addui
38 when "0011" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --addu
39 when "0100" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --sub
40 when "0101" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --subu
41 when "0110" => Adil_result <= Adil_INPUT1 and Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --and
42 when "0111" => Adil_result <= Adil_INPUT1 and Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --andi
43 when "1000" => Adil_result <= Adil_INPUT1 nor Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --nor
44 when "1001" => Adil_result <= Adil_INPUT1 or Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --ori
45 when "1010" => Adil_result <= Adil_INPUT1 or Adil_INPUT2; Adil_INPUT2 <= Adil_INPUT2; --ori
46 when others => Adil_result <= x"00000000";
47 end case;
48
49 if (Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
50 then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
51 else
52   Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
53 end if;
54 Adil_concat <= Adil_INPUT1(Adil_N_bit-1) & Adil_code & (Adil_INPUT2(Adil_N_bit-1) or Adil_extend(Adil_N_bit-1)) & Adil_result(Adil_N_bit-1);
55
56 --Adil_output <= Adil_result(N-1 downto 0);
57 --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT2(N-1) & Adil_result(N-1);
58 end process P1;
59
60 P2: process (Adil_code, Adil_result, Adil_concat)
61   variable Adil_update_z: std_logic;
62   begin
63     Adil_flags <= "000";
64     if (Adil_code = "0000" or Adil_code = "0001" or Adil_code = "0100") then
65       case Adil_concat is
66         when "0000001" => Adil_flags(2) <= '1';
67         when "1000010" => Adil_flags(2) <= '1';
68         when "0010011" => Adil_flags(2) <= '1';
69         when "1010000" => Adil_flags(2) <= '1';
70         when "0000101" => Adil_flags(2) <= '1';
71         when "1000110" => Adil_flags(2) <= '1';
72         when others => Adil_flags(2) <= '0';
73       end case;
74     else Adil_flags(2) <= '0';
75     end if;
76
77     -- N flag
78     if (Adil_code = "0000" or Adil_code = "0001" or
79       Adil_code = "0010" or Adil_code = "0011" or
80       Adil_code = "0100" or Adil_code = "0101")
81     then
82       Adil_flags(1) <= Adil_result(Adil_N_bit-1);
83     end if;
84
85     -- Z flag
86     Adil_update_z := '0';
87     for i in 0 to (Adil_N_bit-1) loop
88       Adil_update_z := Adil_update_z or Adil_result(i);
89     end loop;
90     Adil_flags(0) <= not Adil_update_z;
91   end process P2;
92

```

Figure 10: OP code updated

```

1  -- Adil Ahmad
2  -- Bit Extender and zero extender
3  -----
4
5
6
7  library IEEE;
8  use IEEE.std_logic_1164.all;
9  use IEEE.STD_LOGIC_signed.all;
10 use IEEE.NUMERIC_STD.all;
11
12
13 entity Adil_extunit is
14 port (
15   Ahmad_INPUT: in std_logic_vector (15 downto 0);
16   Ahmad_sel: in std_logic;
17   Ahmad_out: out std_logic_vector (31 downto 0));
18 end Adil_extunit;
19
20
21 architecture Ahmad_behavior of Adil_extunit is
22 signal result: std_logic_vector (31 downto 0) := x"00000000";
23 signal extend: std_logic_vector (15 downto 0) := x"0000";
24 begin
25   process (Ahmad_INPUT, Ahmad_sel, extend, result)
26   begin
27     case Ahmad_sel is
28
29
30     when '0' => result <= x"0000" & Ahmad_INPUT;
31     when '1' => extend <= (15 downto 0 => Ahmad_INPUT(15));
32     result <= extend & Ahmad_INPUT;
33     when others => result <= x"00000000";
34
35   end case;
36   Ahmad_out <= result;
37 end process;
38 end Ahmad_behavior;
39

```

Figure 11: Extended

```

1  -- Adil Ahmad
2  -- ALU code updated
3  -----
4
5  library IEEE;
6  use IEEE.std_logic_1164.all;
7  use IEEE.NUMERIC_STD.all;
8
9  entity Adil_ALU_updated is
10 generic (Adil_N_bit: integer := 32);
11 port(
12   Adil_clk : in std_logic;
13   Adil_RS : in std_logic_vector(Adil_N_bit-1 downto 0);
14   Adil_RT : in std_logic_vector(Adil_N_bit-1 downto 0);
15   Adil_IMM : in std_logic_vector(15 downto 0);
16   Adil_SZ : in std_logic;
17   Adil_code : in std_logic_vector(3 downto 0);
18   Adil_Rd : out std_logic_vector(Adil_N_bit-1 downto 0);
19   Adil_RTout: out std_logic_vector(Adil_N_bit-1 downto 0);
20   Adil_flags: out std_logic_vector(2 downto 0)
21 );
22 end Adil_ALU_updated;
23
24 architecture Adil_structure of Adil_ALU_updated is
25
26
27   component Adil_32_bit_Register is
28     generic (Adil_N_bit: integer := 32);
29     port(
30       Adil_clk : in std_logic;
31       Adil_wren : in std_logic;
32       Adil_rden : in std_logic;
33       Adil_chen : in std_logic;
34       Adil_data : in std_logic_vector(Adil_N_bit-1 downto 0);
35       Adil_q : out std_logic_vector(Adil_N_bit-1 downto 0)
36     );
37   end component;
38
39   component Adil_16_bit_Register is
40     generic (Adil_N_bit: integer := 16);
41     port(
42       Adil_clk : in std_logic;
43       Adil_wren : in std_logic;
44       Adil_rden : in std_logic;
45       Adil_chen : in std_logic;
46       Adil_data : in std_logic_vector(Adil_N_bit-1 downto 0);
47       Adil_q : out std_logic_vector(Adil_N_bit-1 downto 0)
48     );
49   end component;
50
51   component Adil_op_unit_updated is
52     generic (Adil_N_bit: integer := 32);
53     port(
54       Adil_INPUT1 : in std_logic_vector (Adil_N_bit-1 downto 0);
55       Adil_INPUT2 : in std_logic_vector (Adil_N_bit-1 downto 0);
56       Adil_IMM : in std_logic_vector (15 downto 0);
57       Adil_extend : in std_logic_vector (Adil_N_bit-1 downto 0);
58       Adil_code : in std_logic_vector (3 downto 0);
59       AdilImmInst_out : out std_logic_vector (Adil_N_bit-1 downto 0);
60       Adil_output : out std_logic_vector (Adil_N_bit-1 downto 0);
61       Adil_flags : out std_logic_vector (2 downto 0);
62     );
63   end component;
64

```

Figure 12 ALU

```

26
27
28 component Adil_32_bit_Register is
29 generic (Adil_N_bit: integer := 32);
30 port(
31     Adil_clk      : in std_logic;
32     Adil_wren     : in std_logic;
33     Adil_rden     : in std_logic;
34     Adil_chen     : in std_logic;
35     Adil_data     : in std_logic_vector(Adil_N_bit-1 downto 0);
36     Adil_q        : out std_logic_vector(Adil_N_bit-1 downto 0)
37 );
38 end component;
39
40
41 component Adil_16_bit_Register is
42 generic (Adil_N_bit: integer := 16);
43 port(
44     Adil_clk      : in std_logic;
45     Adil_wren     : in std_logic;
46     Adil_rden     : in std_logic;
47     Adil_chen     : in std_logic;
48     Adil_data     : in std_logic_vector(Adil_N_bit-1 downto 0);
49     Adil_q        : out std_logic_vector(Adil_N_bit-1 downto 0)
50 );
51 end component;
52
53 component Adil_op_unit_updated is
54 generic (Adil_N_bit: integer := 32);
55 port(
56     Adil_INPUT1   : in std_logic_vector (Adil_N_bit-1 downto 0);
57     Adil_INPUT2   : in std_logic_vector (Adil_N_bit-1 downto 0);
58     Adil_imm      : in std_logic_vector (15 downto 0);
59     Adil_extend   : in std_logic_vector (Adil_N_bit-1 downto 0);
60     Adil_code     : in std_logic_vector (3 downto 0);
61     Adil_inst_out : out std_logic_vector (Adil_N_bit-1 downto 0);
62     Adil_output   : out std_logic_vector (Adil_N_bit-1 downto 0);
63     Adil_Flags    : out std_logic_vector (2 downto 0)
64 );
65 end component;
66
67
68 component Adil_extunit is
69 port (
70     Ahmad_INPUT: in std_logic_vector (15 downto 0);
71     Ahmad_sel:  in std_logic;
72     Ahmad_out:  out std_logic_vector (31 downto 0);
73 );
74 end component;
75
76
77 signal RS_OUT, RT_OUT, EXT_OUT, RD_IN, RTreg: std_logic_vector(Adil_N_bit-1 downto 0);
78 signal IMM_OUT: std_logic_vector(15 downto 0);
79
80
81 begin
82     Imm      : Adil_16_bit_Register      port map(Adil_clk, '1', '1', '1', Adil_IMM, IMM_OUT);
83     sz_ext   : Adil_extunit              port map(Adil_IMM, Adil_SZ, EXT_OUT);
84     RS       : Adil_32_bit_Register      port map(Adil_clk, '1', '1', '1', Adil_RS, RS_OUT);
85     RT       : Adil_32_bit_Register      port map(Adil_clk, '1', '1', '1', Adil_RT, RT_OUT);
86     Op       : Adil_op_unit_updated      port map(RS_OUT, RT_OUT, IMM_OUT, EXT_OUT, Adil_code, RTreg, RD_IN, Adil_Flags);
87     Rd       : Adil_32_bit_Register      port map(Adil_clk, '1', '1', '1', RD_IN, Adil_RD);
88
89 end Adil_structure;

```

Figure 14ALU

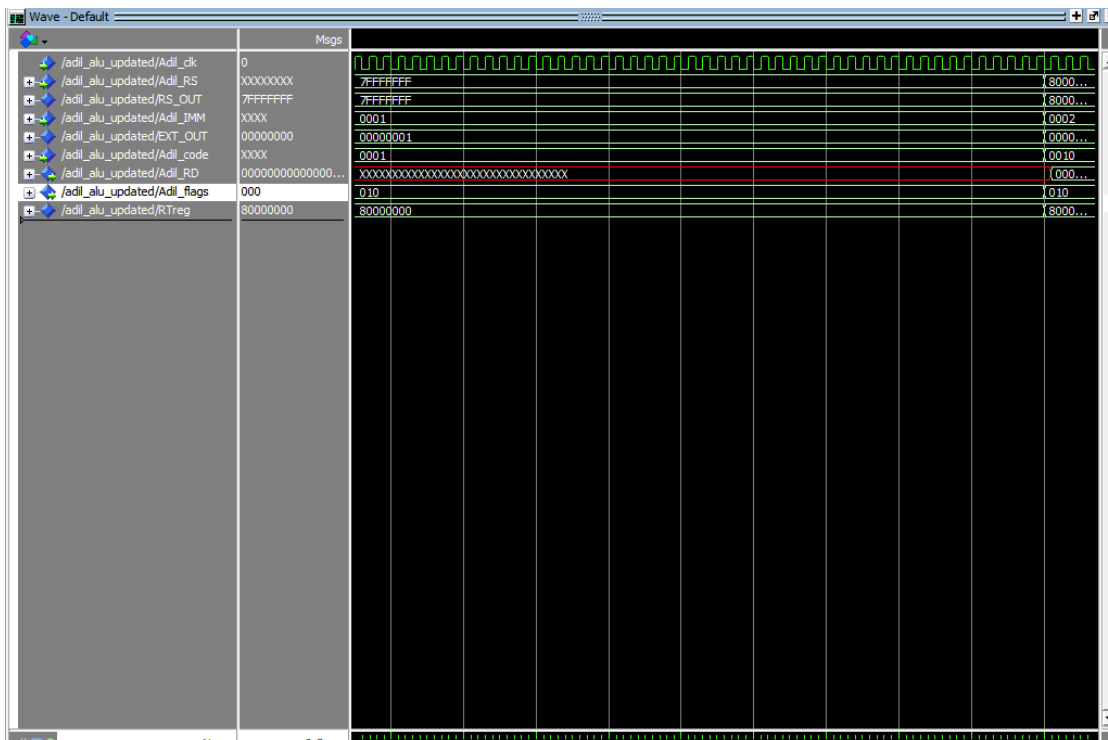


Figure 13Waveform

Here within the first waveform figure, we see that the results follow the flag conditions. $0x7FFFFFFF + 1 = 0x80000000$. This is a both overflow and negative flag. This is also a 16-bit immediate as well. This gets extended to 32-bits. RD is not shown which shows the formula works. The formula for this is, $R[rt] = R[rs]$ operation [SignExtendImmediate].

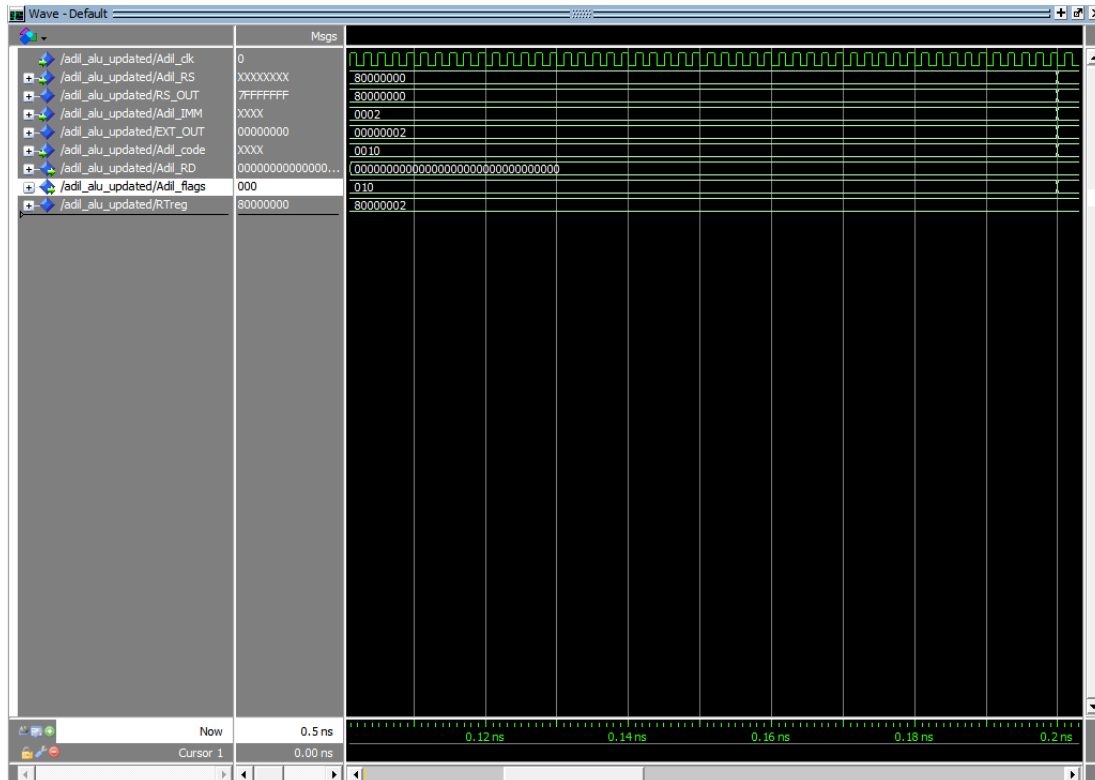


Figure 15 addiu

This waveform, ADDIU instruction does not affect the overflow flag. The results show the output and flags. $0x7FFFFFFF + [\text{signextended}]1 = 0x80000000$ which is negative.

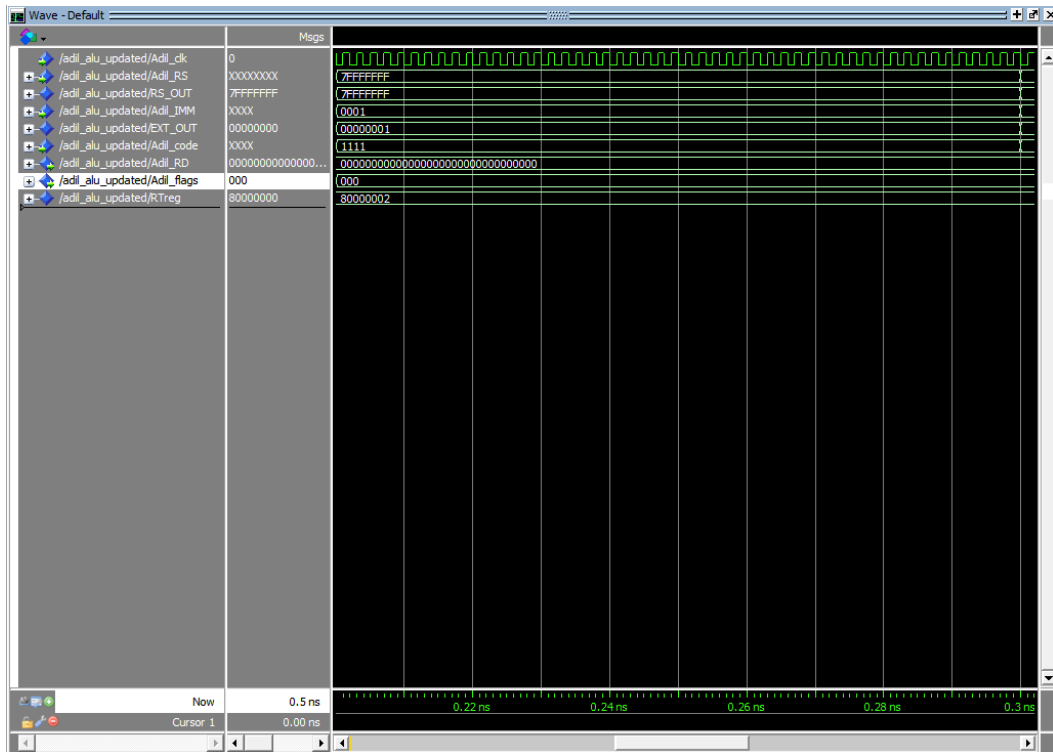


Figure 16 ANDI

This shown above show that andi instruction is working properly. Also, the flags are ignored since logical operations does not set flags.

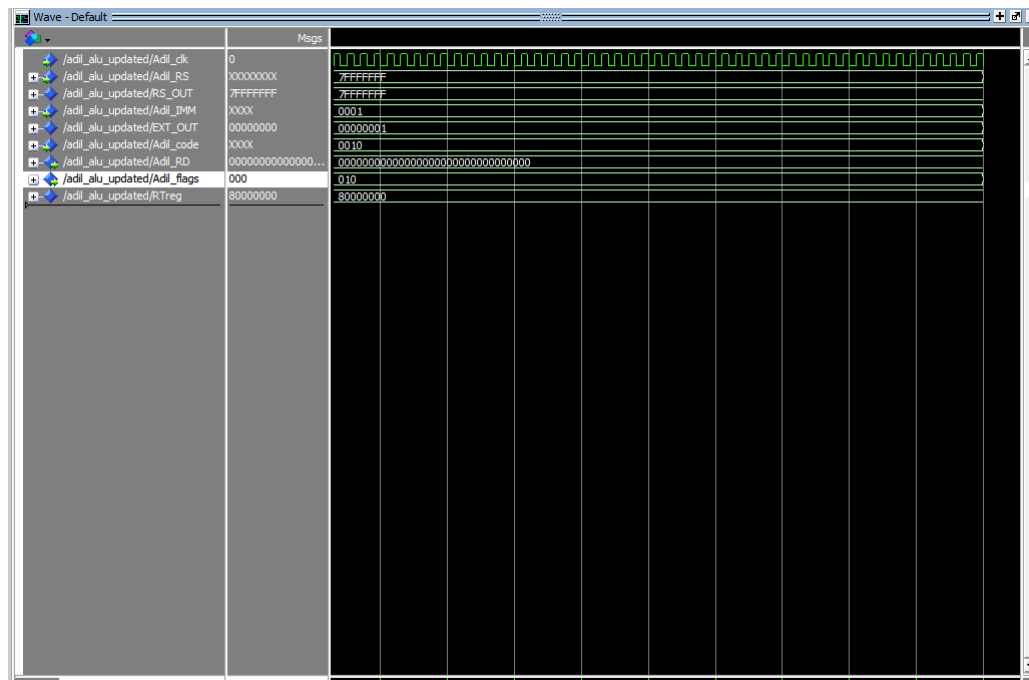
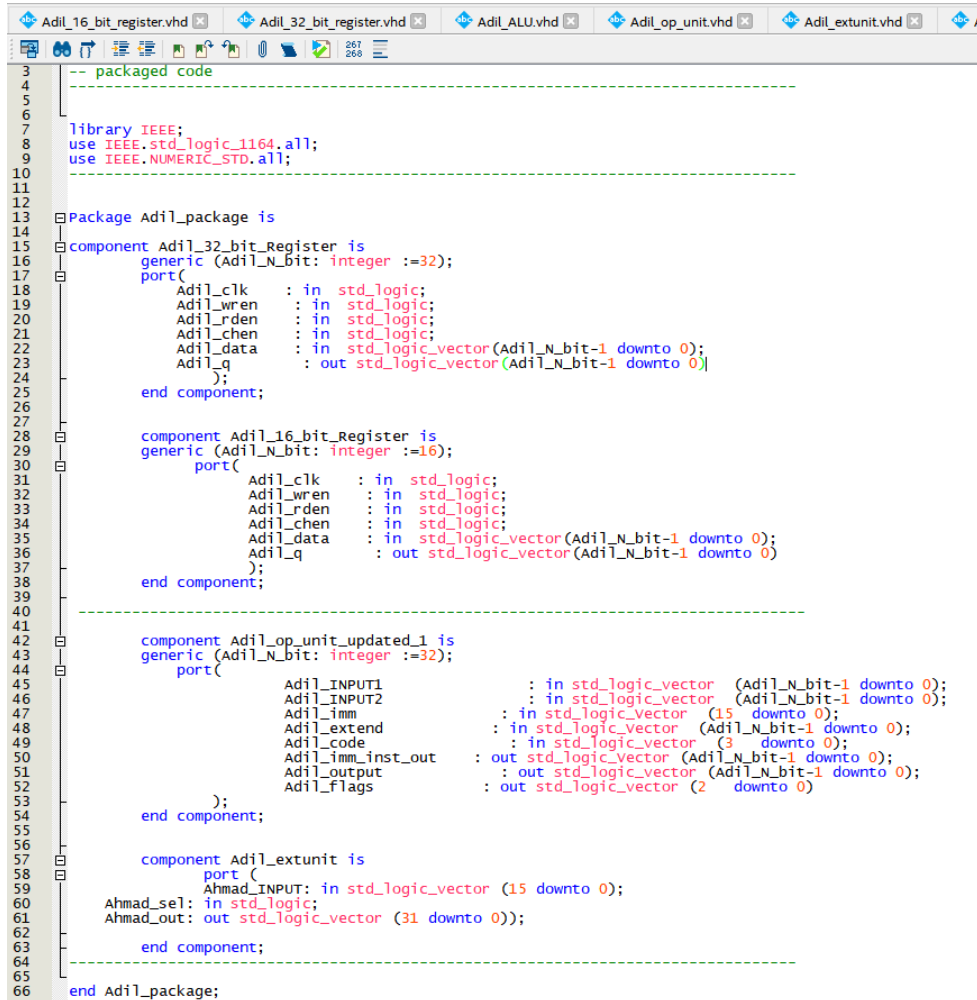


Figure 17: ORI

The figure above, displays the proper results and does not disturb the flags just like the and instruction above. 0x7FFFFFFF and signextend results to 0x7FFFFFFF.

Load Word (LW)



```
3  -- packaged code
4  -----
5
6
7  library IEEE;
8  use IEEE.std_logic_1164.all;
9  use IEEE.NUMERIC_STD.all;
10 -----
11
12
13 package Adil_package is
14
15   component Adil_32_bit_Register is
16     generic (Adil_N_bit: integer := 32);
17     port(
18       Adil_clk      : in std_logic;
19       Adil_wren     : in std_logic;
20       Adil_rden     : in std_logic;
21       Adil_chen     : in std_logic;
22       Adil_data     : in std_logic_vector(Adil_N_bit-1 downto 0);
23       Adil_q        : out std_logic_vector(Adil_N_bit-1 downto 0);
24     );
25   end component;
26
27
28   component Adil_16_bit_Register is
29     generic (Adil_N_bit: integer := 16);
30     port(
31       Adil_clk      : in std_logic;
32       Adil_wren     : in std_logic;
33       Adil_rden     : in std_logic;
34       Adil_chen     : in std_logic;
35       Adil_data     : in std_logic_vector(Adil_N_bit-1 downto 0);
36       Adil_q        : out std_logic_vector(Adil_N_bit-1 downto 0);
37     );
38   end component;
39
40 -----
41
42   component Adil_op_unit_updated_1 is
43     generic (Adil_N_bit: integer := 32);
44     port(
45       Adil_INPUT1   : in std_logic_vector (Adil_N_bit-1 downto 0);
46       Adil_INPUT2   : in std_logic_vector (Adil_N_bit-1 downto 0);
47       Adil_imm       : in std_logic_vector (15 downto 0);
48       Adil_extend    : in std_logic_vector (Adil_N_bit-1 downto 0);
49       Adil_code      : in std_logic_vector (3 downto 0);
50       Adil_inst_out  : out std_logic_vector (Adil_N_bit-1 downto 0);
51       Adil_output    : out std_logic_vector (Adil_N_bit-1 downto 0);
52       Adil_flags     : out std_logic_vector (2 downto 0);
53     );
54   end component;
55
56
57   component Adil_extunit is
58     port (
59       Ahmad_INPUT: in std_logic_vector (15 downto 0);
60       Ahmad_sel: in std_logic;
61       Ahmad_out: out std_logic_vector (31 downto 0));
62   end component;
63 -----
64
65 end Adil_package;
```

Figure 18: Packaged

```

1  -- Adil Ahmad
2  -- Operation code updated
3
4  library IEEE;
5  use IEEE.std_logic_1164.all;
6  use IEEE.std_logic_signed.all;
7  use IEEE.Numeric_std.all;
8
9
10 entity Adil_op_unit_updated_1 is
11   generic (Adil_N_bit: integer := 32);
12   port (
13     Adil_INPUT1 : in std_logic_vector (Adil_N_bit-1 downto 0);
14     Adil_INPUT2 : in std_logic_vector (Adil_N_bit-1 downto 0);
15     Adil_imm     : in std_logic_vector (15 downto 0);
16     Adil_extend  : in std_logic_vector (Adil_N_bit-1 downto 0);
17     Adil_code    : in std_logic_vector (3 downto 0);
18     Adil_wren_MAR : out std_logic;
19     Adil_wren_MDR : out std_logic;
20     Adil_imm_inst_out : out std_logic_vector (Adil_N_bit-1 downto 0);
21     Adil_output    : out std_logic_vector (Adil_N_bit-1 downto 0);
22     Adil_flags     : out std_logic_vector (2 downto 0);
23   );
24 end entity Adil_op_unit_updated_1;
25
26 architecture arch3 of Adil_op_unit_updated_1 is
27   signal Adil_result : std_logic_vector (Adil_N_bit-1 downto 0) := x"00000000";
28   signal Adil_concat : std_logic_vector (6 downto 0);
29   signal wrenmar, wrenmdr : std_logic;
30
31 begin
32   P1: process( Adil_INPUT1, Adil_INPUT2, Adil_code, Adil_result, Adil_imm, Adil_extend)
33   begin
34     case Adil_code is
35       when "0000" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; --addi
36       when "0001" => Adil_result <= Adil_INPUT1 + Adil_extend; --addi
37       when "0010" => Adil_result <= Adil_INPUT1 + Adil_extend; --addi
38       when "0011" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; --addu
39       when "0100" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; --sub
40       when "0101" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; --subu
41       when "0110" => Adil_result <= Adil_INPUT1 and Adil_INPUT2; --and
42       when "0111" => Adil_result <= Adil_INPUT1 and Adil_extend; --andi
43       when "1000" => Adil_result <= Adil_INPUT1 nor Adil_INPUT2; --nor
44       when "1001" => Adil_result <= Adil_INPUT1 or Adil_INPUT2; --or
45       when "1010" => Adil_result <= Adil_INPUT1 or Adil_extend; --ori
46       wrenmar <= '1';
47       wrenmdr <= '1';
48       when others => Adil_result <= x"00000000";
49     end case;
50
51     if(Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
52     then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
53     else
54       Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
55     end if;
56     Adil_concat <= Adil_INPUT1(Adil_N_bit-1) & Adil_code & (Adil_INPUT2(Adil_N_bit-1) or Adil_extend(Adil_N_bit-1)) & Adil_result(Adil_N_bit-1);
57
58     Adil_wren_MAR <= wrenmar;
59     Adil_wren_MDR <= wrenmdr;
60
61     --Adil_output <= Adil_result(N-1 downto 0);
62     --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT2(N-1) & Adil_result(N-1);
63   end process P1;
64

```

Figure 19: OP code updated

```

40   when "0101" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; --subu
41   when "0110" => Adil_result <= Adil_INPUT1 and Adil_INPUT2; --and
42   when "0111" => Adil_result <= Adil_INPUT1 and Adil_extend; --andi
43   when "1000" => Adil_result <= Adil_INPUT1 nor Adil_INPUT2; --nor
44   when "1001" => Adil_result <= Adil_INPUT1 or Adil_INPUT2; --or
45   when "1010" => Adil_result <= Adil_INPUT1 or Adil_extend; --ori
46   wrenmar <= '1';
47   wrenmdr <= '1';
48   when others => Adil_result <= x"00000000";
49 end case;
50
51 if(Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
52 then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
53 else
54   Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
55 end if;
56 Adil_concat <= Adil_INPUT1(Adil_N_bit-1) & Adil_code & (Adil_INPUT2(Adil_N_bit-1) or Adil_extend(Adil_N_bit-1)) & Adil_result(Adil_N_bit-1);
57
58 Adil_wren_MAR <= wrenmar;
59 Adil_wren_MDR <= wrenmdr;
60
61 --Adil_output <= Adil_result(N-1 downto 0);
62 --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT2(N-1) & Adil_result(N-1);
63 end process P1;
64
65
66 P2: process(Adil_code, Adil_result, Adil_concat)
67   variable Adil_update_z : std_logic;
68   begin
69     Adil_flags <= "000";
70     if (Adil_code = "0000" or Adil_code = "0001" or Adil_code = "0100") then
71       case Adil_concat is
72         when "00000001" => Adil_flags(2) <= '1';
73         when "10000010" => Adil_flags(2) <= '1';
74         when "01000011" => Adil_flags(2) <= '1';
75         when "10100000" => Adil_flags(2) <= '1';
76         when "0000101" => Adil_flags(2) <= '1';
77         when "1000110" => Adil_flags(2) <= '1';
78         when others => Adil_flags(2) <= '0';
79       end case;
80     else Adil_flags(2) <= '0';
81     end if;
82
83     -- N flag
84     if(Adil_code = "0000" or Adil_code = "0001" or
85       Adil_code = "0010" or Adil_code = "0011" or
86       Adil_code = "0100" or Adil_code = "0101")
87     then
88       Adil_flags(1) <= Adil_result(Adil_N_bit-1);
89     end if;
90
91     -- Z flag
92     Adil_update_z := '0';
93     for i in 0 to (Adil_N_bit-1) loop
94       Adil_update_z := Adil_update_z or Adil_result(i);
95     end loop;
96     Adil_flags(0) <= not Adil_update_z;
97

```

Figure 20: OP Code updated


```

1  -- Adil Ahmad
2  -- ALU code updated
3  -----
4
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8  use work.Adil_package.all;
9
10 entity Adil_ALU_updated_1 is
11   generic (N: integer := 32);
12   port(
13     Adil_clk      : in  std_logic;
14     Adil_RS       : in  std_logic_vector(N-1 downto 0);
15     Adil_RT       : in  std_logic_vector(N-1 downto 0);
16     Adil_MDR      : in  std_logic_vector(N-1 downto 0); -- mdr input
17     Adil_IMM      : in  std_logic_vector(15 downto 0);
18     Adil_SZ       : in  std_logic;
19     Adil_code     : in  std_logic_vector(3 downto 0);
20     Adil_RD       : out std_logic_vector(N-1 downto 0);
21     Adil_RTout    : out std_logic_vector(N-1 downto 0);
22     Adil_Flags    : out std_logic_vector(2 downto 0));
23 end Adil_ALU_updated_1;
24
25 architecture Adil_structure of Adil_ALU_updated_1 is
26   signal RS_OUT, RT_OUT, EXT_OUT, RD_IN, RTreg: std_logic_vector(N-1 downto 0);
27   signal IMM_OUT: std_logic_vector(15 downto 0);
28   signal MAR_OUT, MDR_OUT: std_logic_vector(N-1 downto 0);
29   signal wrenmar, wrenmdr: std_logic;
30
31 begin
32   Imm      : Adil_16_bit_register      port map(Adil_clk, '1','1','1', Adil_IMM, IMM_OUT);
33   sz_ext   : Adil_extunit              port map(Adil_IMM, Adil_SZ, EXT_OUT);
34   RS       : Adil_32_bit_register      port map(Adil_clk, '1','1','1', Adil_RS, RS_OUT);
35   RT       : Adil_32_bit_register      port map(Adil_clk, '1','1','1', Adil_RT, RT_OUT);
36   Op       : Adil_op_unit_updated_1 port map(RS_OUT, RT_OUT, IMM_OUT, EXT_OUT, Adil_code, RTreg, RD_IN, Adil_Flags);
37   Rd       : Adil_32_bit_register      port map(Adil_clk, '1','1','1', RD_IN, Adil_RD);
38   MAR      : Adil_32_bit_register      port map(Adil_clk, wrenmar, wrenmar, '1', Adil_MDR, MDR_OUT);
39   MDR      : Adil_32_bit_register      port map(Adil_clk, '1','1','1', MDR_OUT, RT_OUT);
40
41 end Adil_structure;

```

Figure 21 ALU updated

The components are already included in the figures above along with the registers.

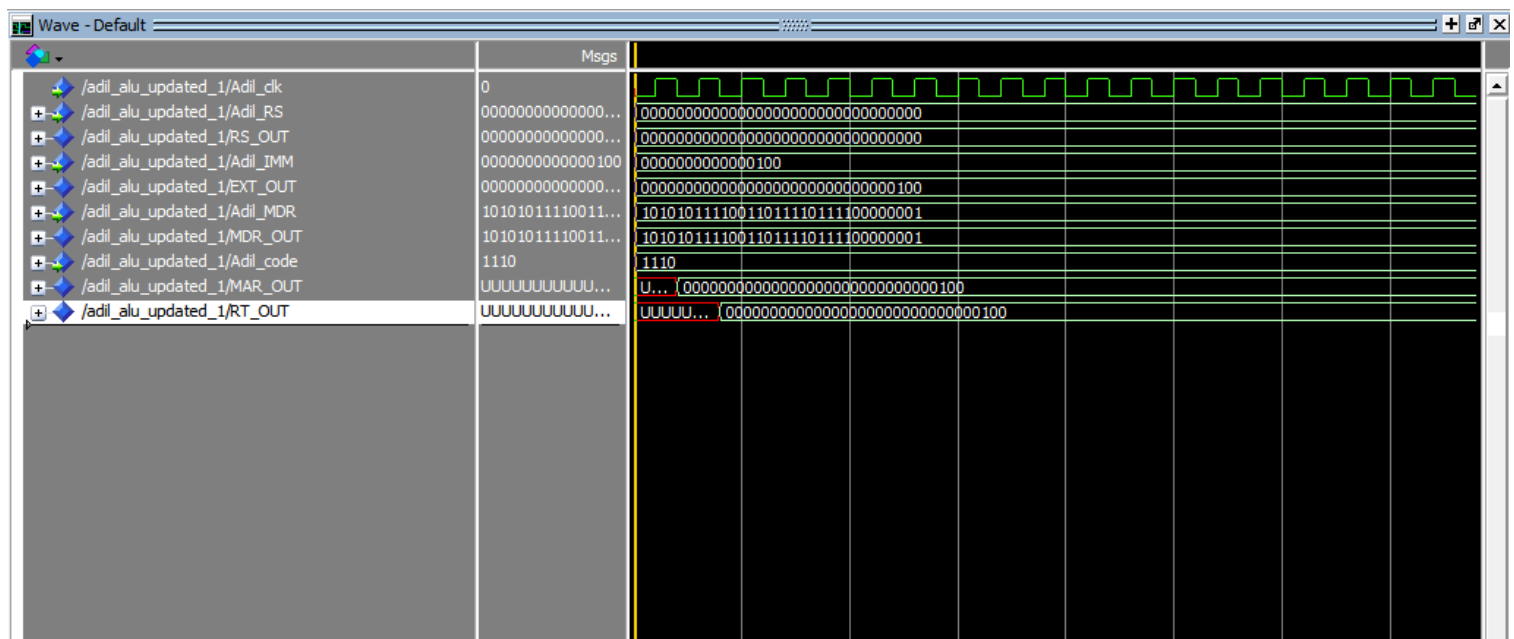


Figure 22: LW

For LW, this initialized MDR to 0xABCDEF01 so we can update register R[rt]. For this part, MAR = R[rs] +

signextend = 0 + 4 = 0x00000004. Afterwards R[rt] becomes the value of MDR, which is 0xABCDEF01. So

then data is loaded to address 0x00000000 with offset of 4.

Store Word (SW)

```

1  -- Adil Ahmad
2  -- Operation code updated
3  -----
4  library IEEE;
5  use IEEE.std_logic_1164.all;
6  use IEEE.std_logic_signed.all;
7  use IEEE.Numeric_std.all;
8
9
10 entity Adil_op_unit_updated_2 is
11   generic (Adil_N_bit: integer := 32);
12   port (
13     Adil_INPUT1 : in std_logic_vector (Adil_N_bit-1 downto 0);
14     Adil_INPUT2 : in std_logic_vector (Adil_N_bit-1 downto 0);
15     Adil_imm     : in std_logic_vector (15 downto 0);
16     Adil_extend  : in std_logic_vector (Adil_N_bit-1 downto 0);
17     Adil_code    : in std_logic_vector (3 downto 0);
18     Adil_wren_MAR : out std_logic;
19     Adil_wren_MDR : out std_logic;
20     Adil_imm_inst_out : out std_logic_vector (Adil_N_bit-1 downto 0);
21     Adil_output    : out std_logic_vector (Adil_N_bit-1 downto 0);
22     Adil_flags     : out std_logic_vector (2 downto 0);
23   );
24 end Adil_op_unit_updated_2;
25
26 architecture arch3 of Adil_op_unit_updated_2 is
27   signal Adil_result : std_logic_vector (Adil_N_bit-1 downto 0) := x"00000000";
28   signal Adil_concat : std_logic_vector ( 6 downto 0);
29   signal wrenmar, wrenmdr : std_logic;
30
31   begin
32     P1: process ( Adil_INPUT1, Adil_INPUT2, Adil_code, Adil_result, Adil_imm, Adil_extend)
33     begin
34       case Adil_code is
35         when "0000" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; --addi
36         when "0001" => Adil_result <= Adil_INPUT1 + Adil_extend; --addi
37         when "0010" => Adil_result <= Adil_INPUT1 + Adil_extend; --addi
38         when "0011" => Adil_result <= Adil_INPUT1 + Adil_INPUT2; --addu
39         when "0100" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; --sub
40         when "0101" => Adil_result <= Adil_INPUT1 - Adil_INPUT2; --subu
41         when "0110" => Adil_result <= Adil_INPUT1 and Adil_INPUT2; --and
42         when "0111" => Adil_result <= Adil_INPUT1 and Adil_extend; --andi
43         when "1000" => Adil_result <= Adil_INPUT1 nor Adil_INPUT2; --nor
44         when "1001" => Adil_result <= Adil_INPUT1 or Adil_INPUT2; --or
45         when "1010" => Adil_result <= Adil_INPUT1 or Adil_extend; --ori
46         wrenmar <= '1';
47         wrenmdr <= '1';
48
49         when "1111" => Adil_result <= Adil_extend;
50         wrenmar <= '1';
51         wrenmdr <= '1';
52         when others => Adil_result <= x"00000000";
53       end case;
54
55       if (Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
56       then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
57       else
58         Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
59       end if;
60       Adil_concat <= Adil_INPUT1(Adil_N_bit-1) & Adil_code & (Adil_INPUT2(Adil_N_bit-1) or Adil_extend(Adil_N_bit-1)) & Adil_result(Adil_N_bit-1);
61
62       Adil_wren_MAR <= wrenmar;
63       Adil_wren_MDR <= wrenmdr;
64     end process P1;
65

```

Figure 23: SW

```

44 when "1001" => Adil_result <= Adil_INPUT1 or Adil_INPUT2; --or
45 when "1010" => Adil_result <= Adil_INPUT1 or Adil_extend; --or1
46 wrenmar <= '1';
47 wrenmdr <= '1';
48
49 when "1111" => Adil_result <= Adil_extend;
50 wrenmar <= '1';
51 wrenmdr <= '1';
52 when others => Adil_result <= x"00000000";
53 end case;
54
55 if (Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
56 then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
57 else
58 Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
59 end if;
60 Adil_concat <= Adil_INPUT1(Adil_N_bit-1) & Adil_code & (Adil_INPUT2(Adil_N_bit-1) or Adil_extend(Adil_N_bit-1)) & Adil_result(Adil_N_bit-1);
61
62 Adil_wren_MAR <= wrenmar;
63 Adil_wren_MDR <= wrenmdr;
64
65 --Adil_output <= Adil_result(N-1 downto 0);
66 --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT2(N-1) & Adil_result(N-1);
67 end process P1;
68
69 P2: process(Adil_code, Adil_result, Adil_concat)
70 variable Adil_update_z: std_logic;
71 begin
72
73 Adil_flags <= "000";
74 if (Adil_code = "0000" or Adil_code = "0001" or Adil_code = "0100") then
75 case Adil_concat is
76
77 when "0000001" => Adil_flags(2) <= '1';
78 when "1000010" => Adil_flags(2) <= '1';
79 when "0010011" => Adil_flags(2) <= '1';
80 when "1010000" => Adil_flags(2) <= '1';
81 when "0000101" => Adil_flags(2) <= '1';
82 when "1000110" => Adil_flags(2) <= '1';
83 when others => Adil_flags(2) <= '0';
84 end case;
85 else Adil_flags(2) <= '0';
86
87 end if;
88
89 -- N flag
90 if (Adil_code = "0000" or Adil_code = "0001" or
91 Adil_code = "0010" or Adil_code = "0011" or
92 Adil_code = "0100" or Adil_code = "0101")
93 then
94 Adil_flags(1) <= Adil_result(Adil_N_bit-1);
95
96 -- Z flag
97 Adil_update_z := '0';
98 for i in 0 to (Adil_N_bit-1) loop
99 Adil_update_z := Adil_update_z or Adil_result(i);
100 end loop;
101 Adil_flags(0) <= not Adil_update_z;
102 else Adil_flags <= "000";
103 end if;
104 end process P2;
105
106 end arch3;
107

```

Figure 24: SW OP

```

1
2 -- Adil Ahmad
3 -- ALU code updated
4
5
6 library IEEE;
7 use IEEE.std_logic_1164.all;
8 use work.Adil_package.all;
9
10 entity Adil_ALU_updated_2 is
11 generic (N: integer := 32);
12 port(
13     Adil_clk : in std_logic;
14     Adil_RS : in std_logic_vector(N-1 downto 0);
15     Adil_RT : in std_logic_vector(N-1 downto 0);
16     Adil_MDR : in std_logic_vector(N-1 downto 0); -- mdr input
17     Adil_IMM : in std_logic_vector(15 downto 0);
18     Adil_SZ : in std_logic;
19     Adil_code : in std_logic_vector(3 downto 0);
20     Adil_RD : out std_logic_vector(N-1 downto 0);
21     Adil_RTout : out std_logic_vector(N-1 downto 0);
22     Adil_flags : out std_logic_vector(2 downto 0);
23 end Adil_ALU_updated_2;
24
25 architecture Adil_structure of Adil_ALU_updated_2 is
26
27     signal RS_OUT, EXT_OUT, RD_IN, RTreg: std_logic_vector(N-1 downto 0);
28
29     signal IMM_OUT: std_logic_vector(15 downto 0);
30     signal MAR_OUT, MDR_OUT: std_logic_vector(N-1 downto 0);
31     signal wrenmar, wrenmdr: std_logic;
32
33 begin
34
35     Imm : Adil_16_bit_register
36     sz_ext : Adil_extunit
37     RS : Adil_32_bit_register
38     RT : Adil_32_bit_register
39     Op : Adil_op_unit_updated_1
40     RD : Adil_32_bit_register
41     MAR : Adil_32_bit_register
42
43     port map(Adil_clk, '1', '1', '1', Adil_IMM, IMM_OUT);
44     port map(Adil_IMM, Adil_SZ, EXT_OUT);
45     port map(Adil_clk, '1', '1', '1', Adil_RS, RS_OUT);
46     port map(Adil_clk, '1', '1', '1', Adil_RT, RT_OUT);
47     port map(RS_OUT, RT_OUT, IMM_OUT, EXT_OUT, Adil_code, RTreg, RD_IN, Adil_flags);
48     port map(Adil_clk, '1', '1', '1', RD_IN, Adil_RD);
49     port map(Adil_clk, wrenmar, wrenmdr, '1', RD_IN, MAR_OUT);
50

```

Figure 25: ALU updated

This here is the final ALU updated with all the operation codes and packages updated

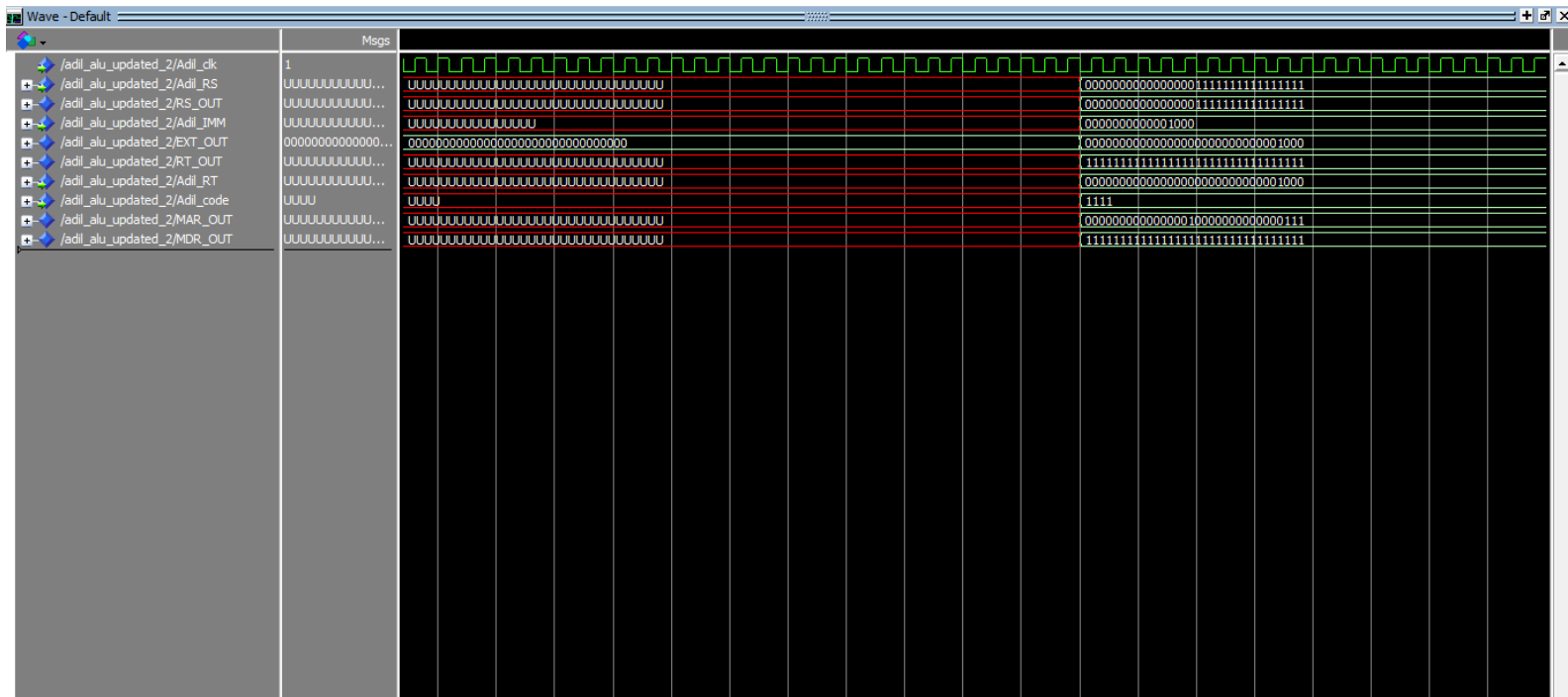


Figure 27: Final ALU

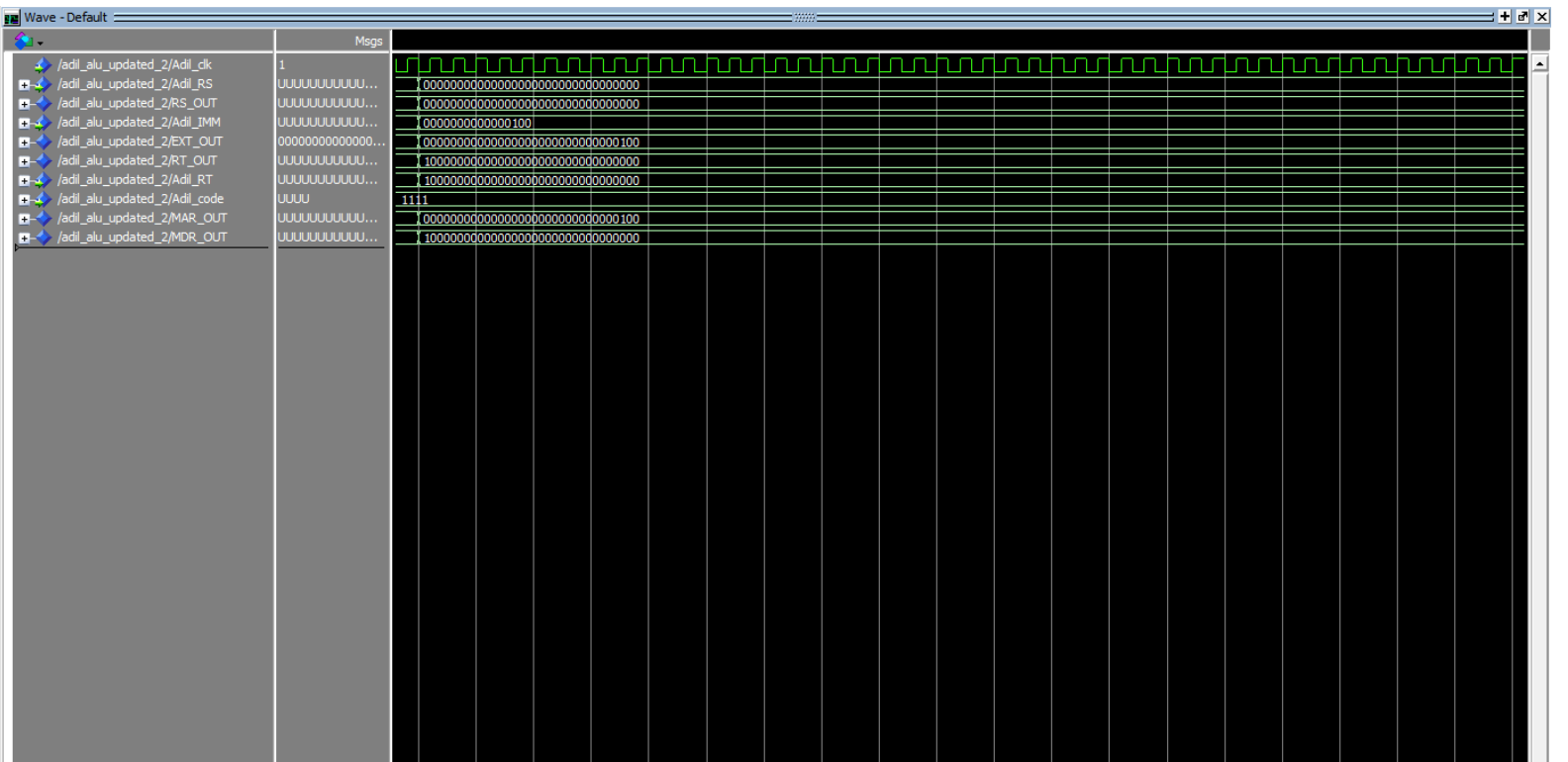


Figure 26: final ALU

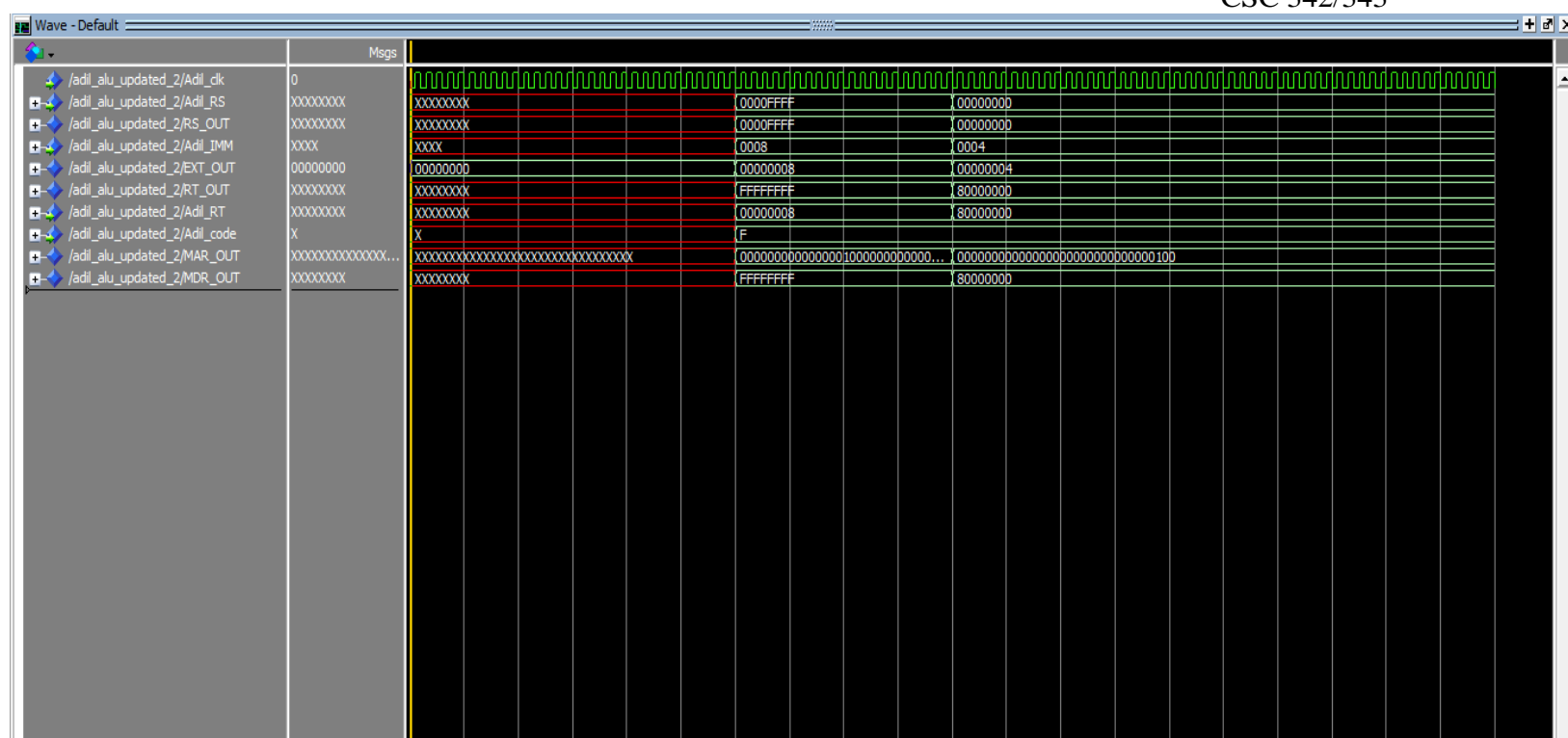


Figure 28: final ALU

This waveform shows that MDR is initialized in the beginning. When the instruction is used, $MAR = R[rt]$ + signed immediate, and MDR is updated to the value of $R[rt]$. So, $MAR = 00010007$ in binary with MDR value of $0xFFFFFFFF$, and for the other wave, $MAR = 0x00000004$ with MDR of $0x80000000$.

Conclusion

In conclusion, I learned a lot within this lab that helped me understand the structure and design of ALUs. This lab taught me important and complex syntax for VHDL. We used this to design the internal structure of the ALU and all its data paths. For the first data path, we used the instructions ADD, ADDU, SUB, SUBU, AND, NOR, OR. These instructions were built on R_TYPE INSTRUCTIONS with 3 bits. For the next design we used the instructions, ADDI, ADDIU, ANDI, ORI. These use the I arithmetic logic TYPE INSTRUCTIONS. Finally, we use for the memory address instructions, Store word which uses I type instructions. Also, for the load word, uses I type instructions also.