FINAL Laboratory Project

Ahmad Adil

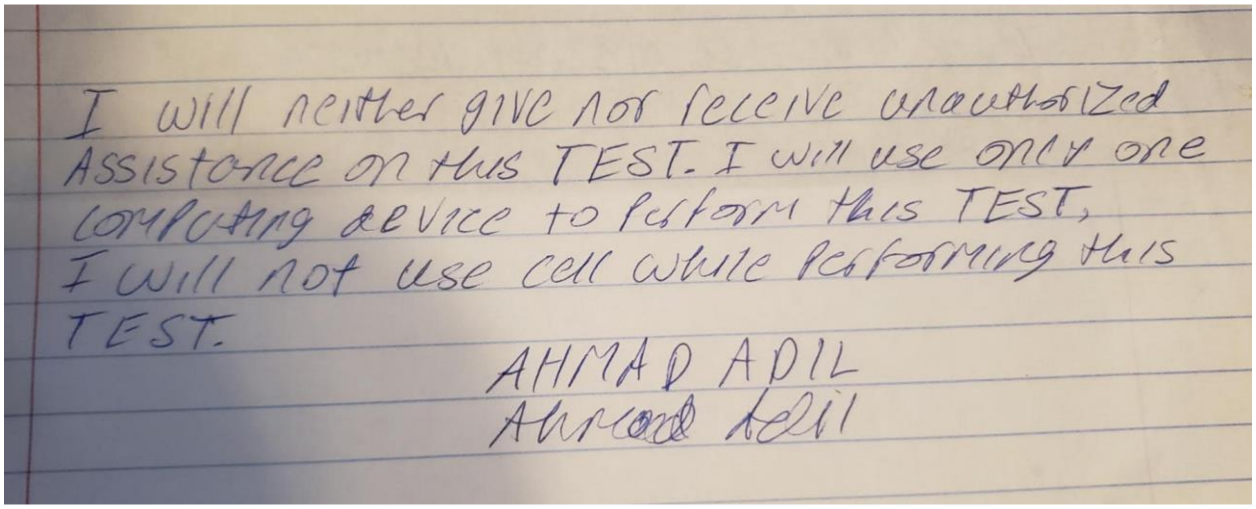
7/22/21

Professor Gertner

Computer Organization 342/343

Adil, Ahmad
7/22/21
Professor Gertner
CSC 342/343

Table of Contents



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:

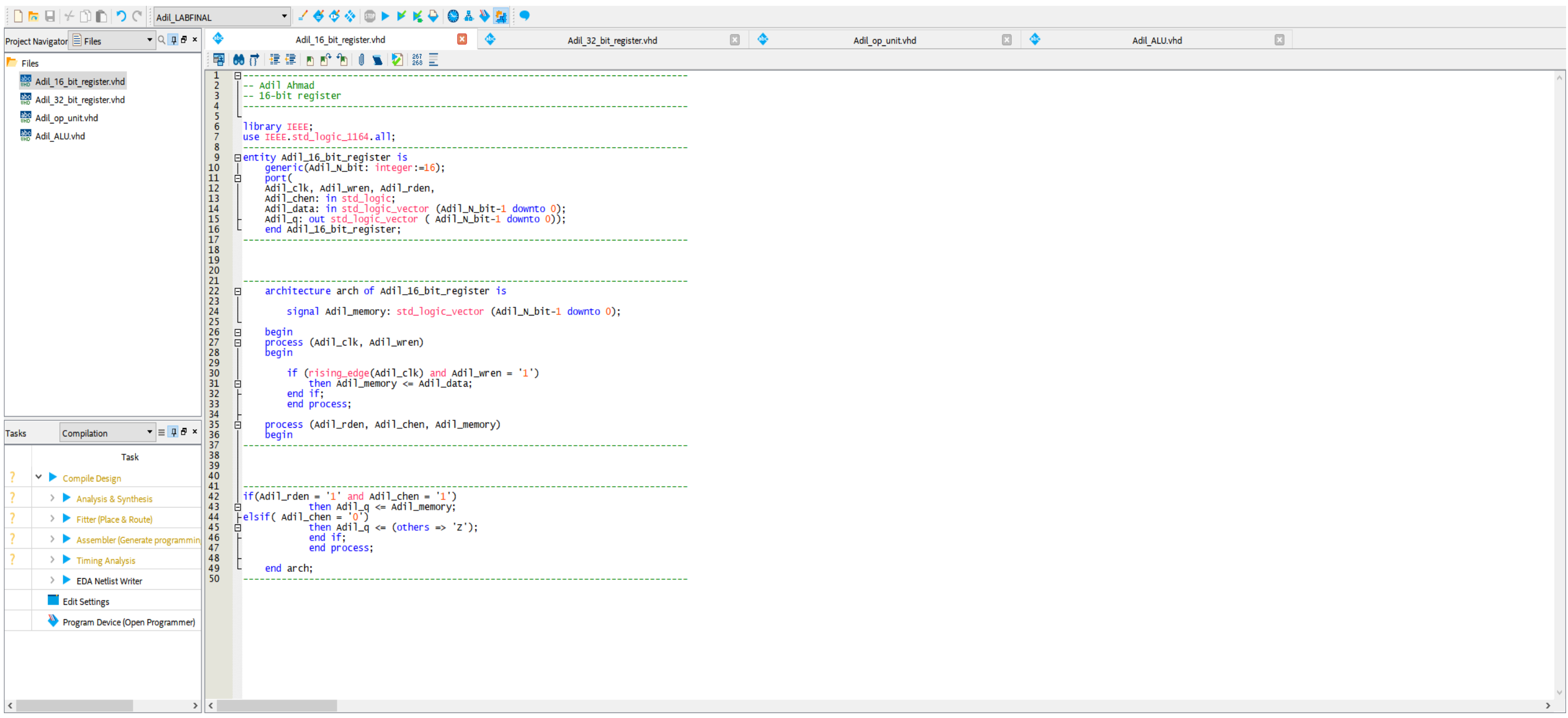
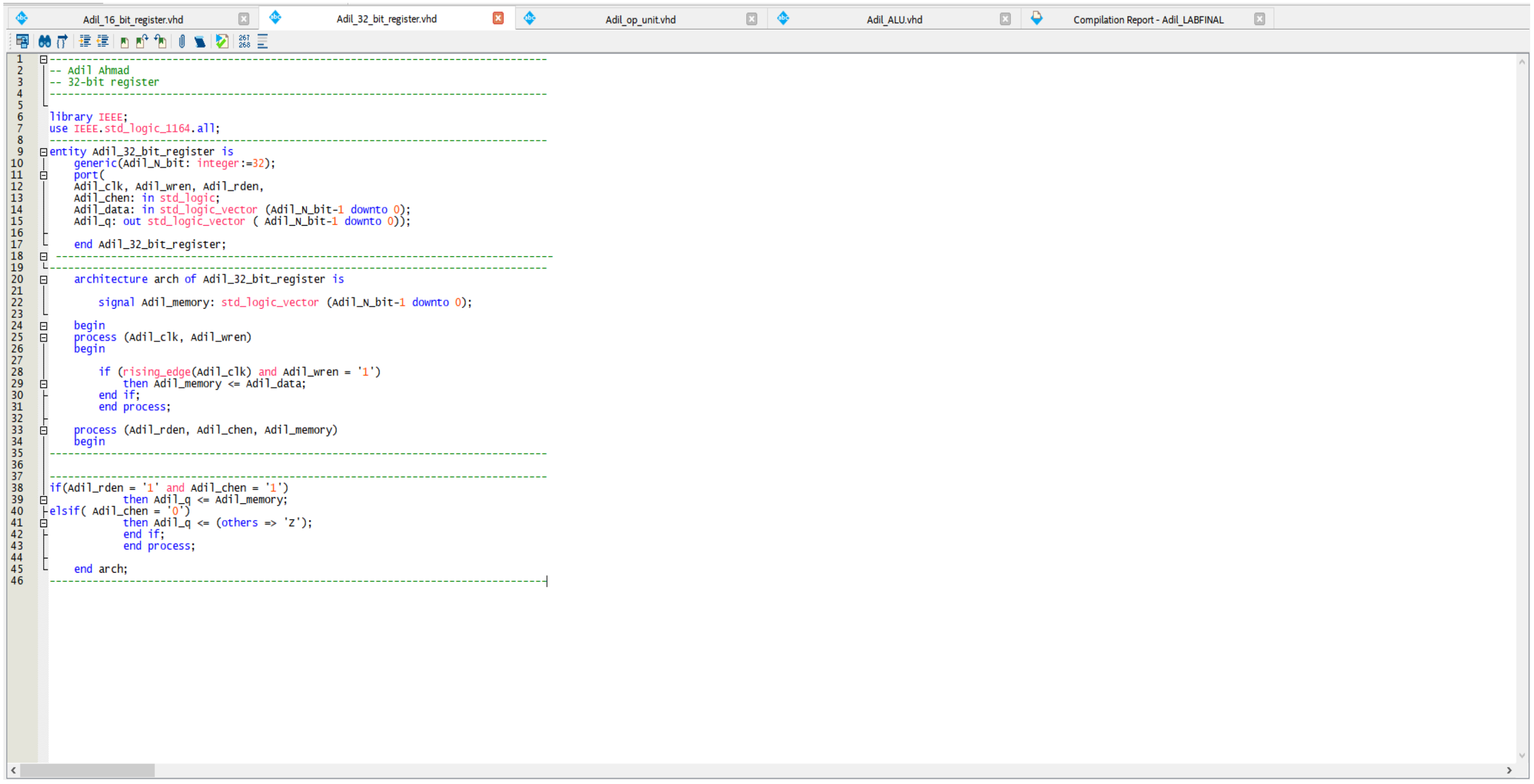

```
----------------------------------------------------------------------
-- Adil Ahmad
-- 16-bit register
----------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
----------------------------------------------------------------------
entity Adil_16_bit_register is
    generic(Adil_N_bit: integer:=16);
    port(
    Adil_clk, Adil_wren, Adil_rden,
    Adil_chen: in std_logic;
    Adil_data: in std_logic_vector (Adil_N_bit-1 downto 0);
    Adil_q: out std_logic_vector ( Adil_N_bit-1 downto 0));
    end Adil_16_bit_register;
----------------------------------------------------------------------




----------------------------------------------------------------------
    architecture arch of Adil_16_bit_register is

        signal Adil_memory: std_logic_vector (Adil_N_bit-1 downto 0);

    begin
    process (Adil_clk, Adil_wren)
    begin

        if (rising_edge(Adil_clk) and Adil_wren = '1')
            then Adil_memory <= Adil_data;
        end if;
        end process;

    process (Adil_rden, Adil_chen, Adil_memory)
    begin
----------------------------------------------------------------------




----------------------------------------------------------------------
if(Adil_rden = '1' and Adil_chen = '1')
            then Adil_q <= Adil_memory;
elsif( Adil_chen = '0')
            then Adil_q <= (others => 'Z');
            end if;
            end process;

    end arch;
----------------------------------------------------------------------
```

Figure 1: 16-bit Registers



```
------------------------------------------------------------------------------
-- Adil Ahmad
-- 32-bit register
------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
------------------------------------------------------------------------------
entity Adil_32_bit_register is
    generic(Adil_N_bit: integer:=32);
    port(
    Adil_clk, Adil_wren, Adil_rden,
    Adil_chen: in std_logic;
    Adil_data: in std_logic_vector (Adil_N_bit-1 downto 0);
    Adil_q: out std_logic_vector ( Adil_N_bit-1 downto 0));

    end Adil_32_bit_register;
 ------------------------------------------------------------------------------
------------------------------------------------------------------------------
    architecture arch of Adil_32_bit_register is

        signal Adil_memory: std_logic_vector (Adil_N_bit-1 downto 0);

    begin
    process (Adil_clk, Adil_wren)
    begin

        if (rising_edge(Adil_clk) and Adil_wren = '1')
            then Adil_memory <= Adil_data;
        end if;
        end process;

    process (Adil_rden, Adil_chen, Adil_memory)
    begin
------------------------------------------------------------------------------


------------------------------------------------------------------------------
if(Adil_rden = '1' and Adil_chen = '1')
            then Adil_q <= Adil_memory;
elsif( Adil_chen = '0')
            then Adil_q <= (others => 'Z');
            end if;
            end process;

    end arch;
------------------------------------------------------------------------------
```

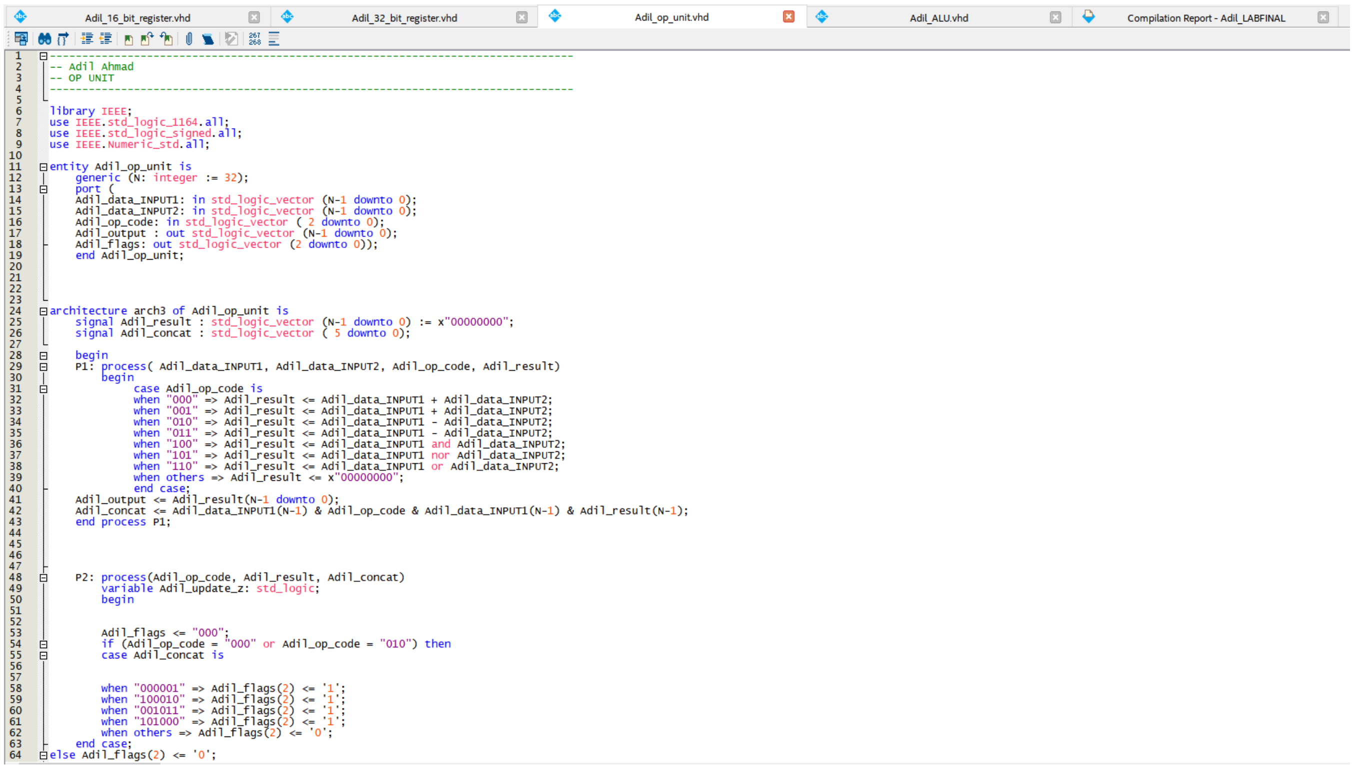
Figure 2 32-bit register

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:



```vhdl
--------------------------------------------------------------------------------
-- Adil Ahmad
-- OP UNIT
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.Numeric_std.all;

entity Adil_op_unit is
    generic (N: integer := 32);
    port (
    Adil_data_INPUT1: in std_logic_vector (N-1 downto 0);
    Adil_data_INPUT2: 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 Adil_op_unit;




architecture arch3 of Adil_op_unit is
    signal Adil_result : std_logic_vector (N-1 downto 0) := x"00000000";
    signal Adil_concat : std_logic_vector ( 5 downto 0);

    begin
    P1: process( Adil_data_INPUT1, Adil_data_INPUT2, Adil_op_code, Adil_result)
        begin
            case Adil_op_code is
            when "000" => Adil_result <= Adil_data_INPUT1 + Adil_data_INPUT2;
            when "001" => Adil_result <= Adil_data_INPUT1 + Adil_data_INPUT2;
            when "010" => Adil_result <= Adil_data_INPUT1 - Adil_data_INPUT2;
            when "011" => Adil_result <= Adil_data_INPUT1 - Adil_data_INPUT2;
            when "100" => Adil_result <= Adil_data_INPUT1 and Adil_data_INPUT2;
            when "101" => Adil_result <= Adil_data_INPUT1 nor Adil_data_INPUT2;
            when "110" => Adil_result <= Adil_data_INPUT1 or Adil_data_INPUT2;
            when others => Adil_result <= x"00000000";
            end case;
    Adil_output <= Adil_result(N-1 downto 0);
    Adil_concat <= Adil_data_INPUT1(N-1) & Adil_op_code & Adil_data_INPUT1(N-1) & Adil_result(N-1);
    end process P1;




    P2: process(Adil_op_code, Adil_result, Adil_concat)
        variable Adil_update_z: std_logic;
        begin


        Adil_flags <= "000";
        if (Adil_op_code = "000" or Adil_op_code = "010") then
        case Adil_concat is


        when "000001" => Adil_flags(2) <= '1';
        when "100010" => Adil_flags(2) <= '1';
        when "001011" => Adil_flags(2) <= '1';
        when "101000" => Adil_flags(2) <= '1';
        when others => Adil_flags(2) <= '0';
    end case;
else Adil_flags(2) <= '0';
```

Figure 3: VHDL code for OP

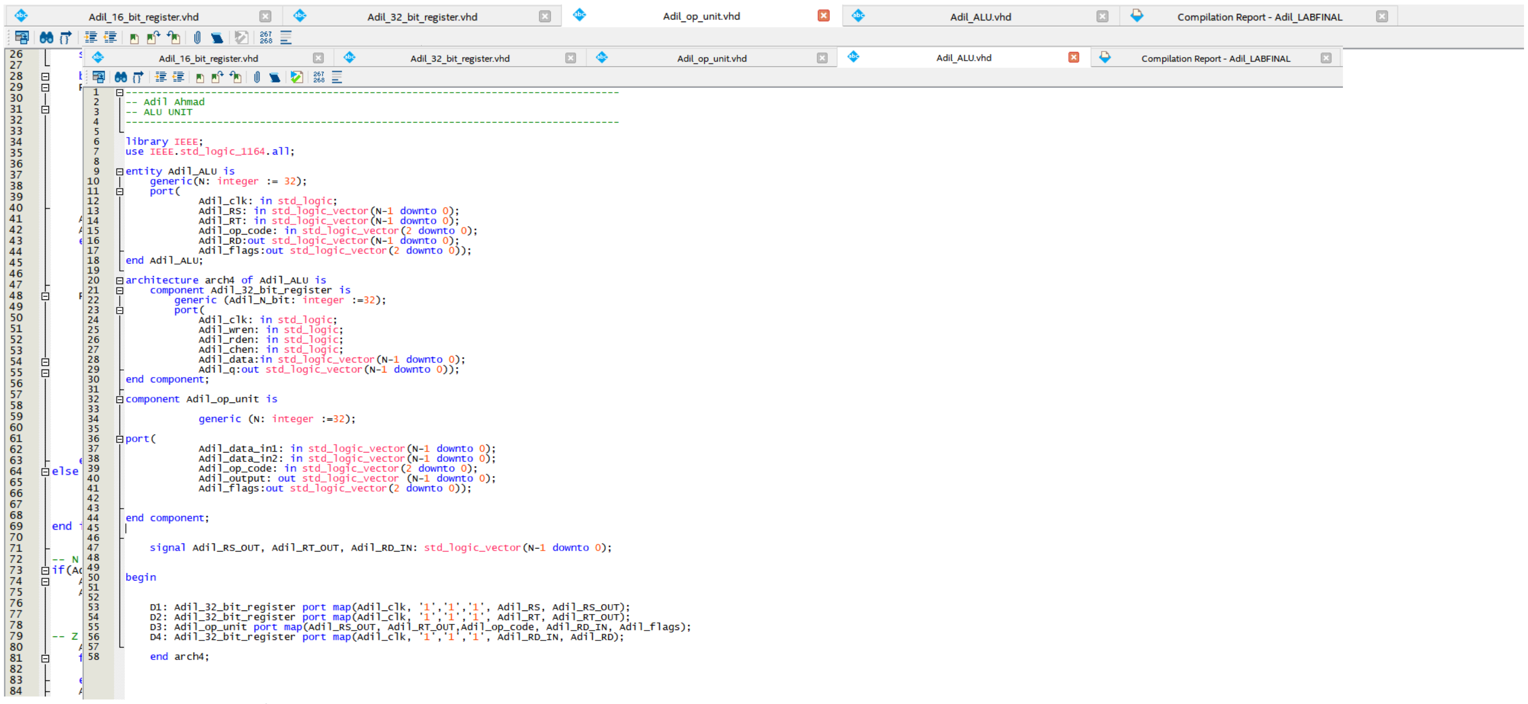


Figure 4: OP unit continued

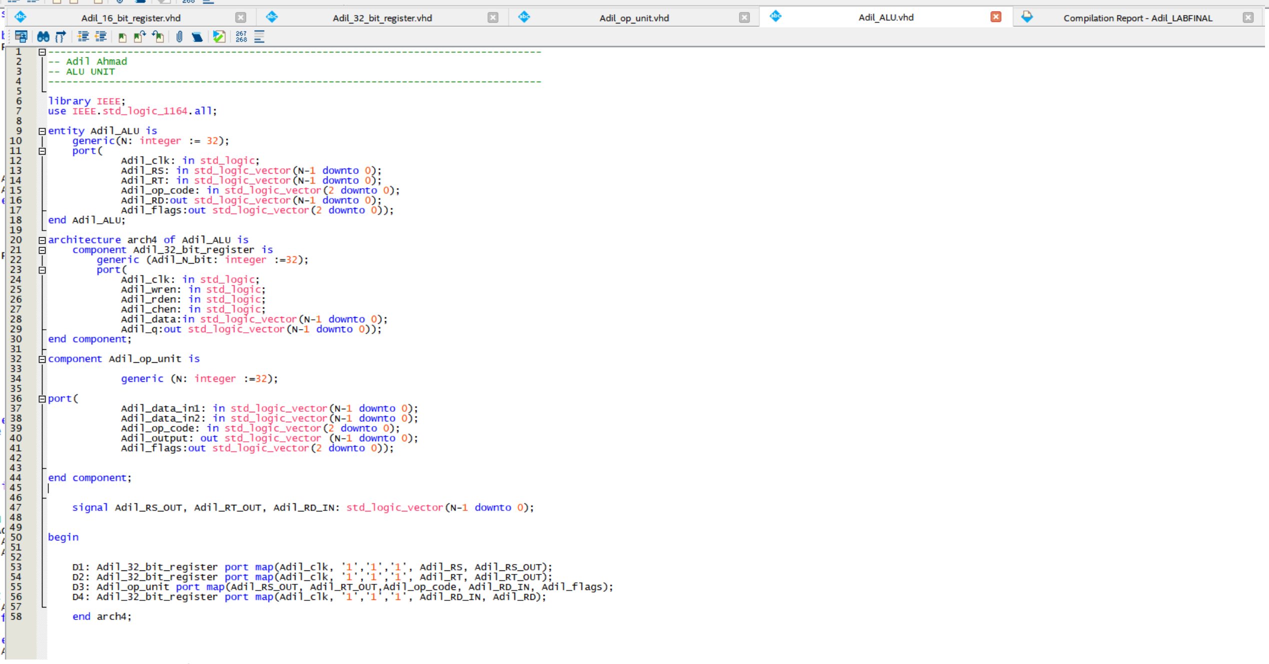


Figure 5: ALU code

```vhdl
-----------------------------------------------------------------------------
-- 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);


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;
```

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:

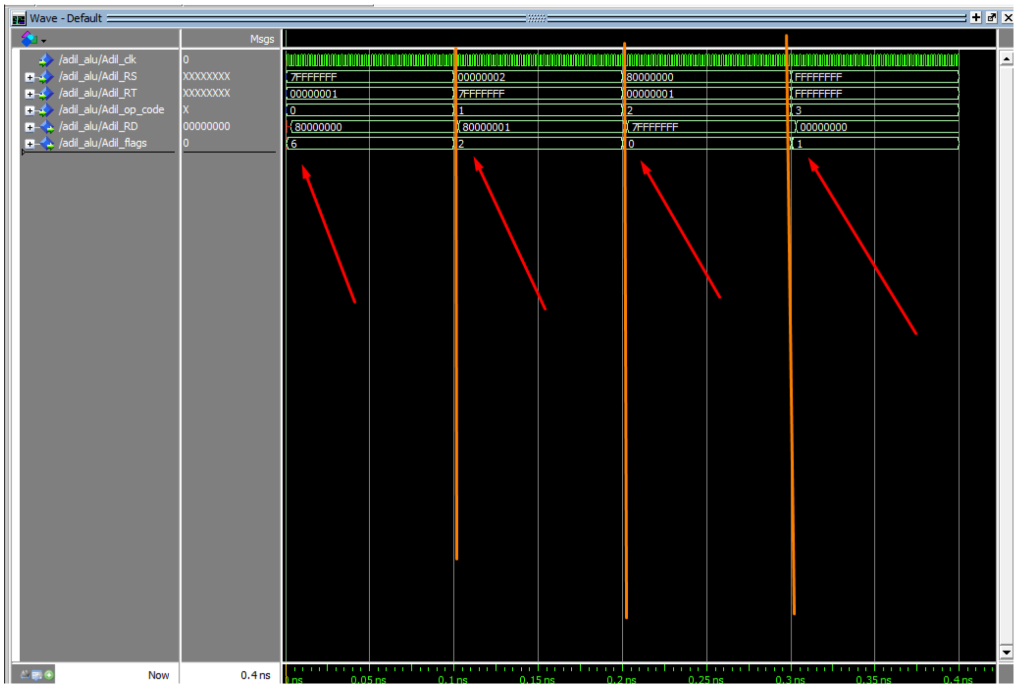
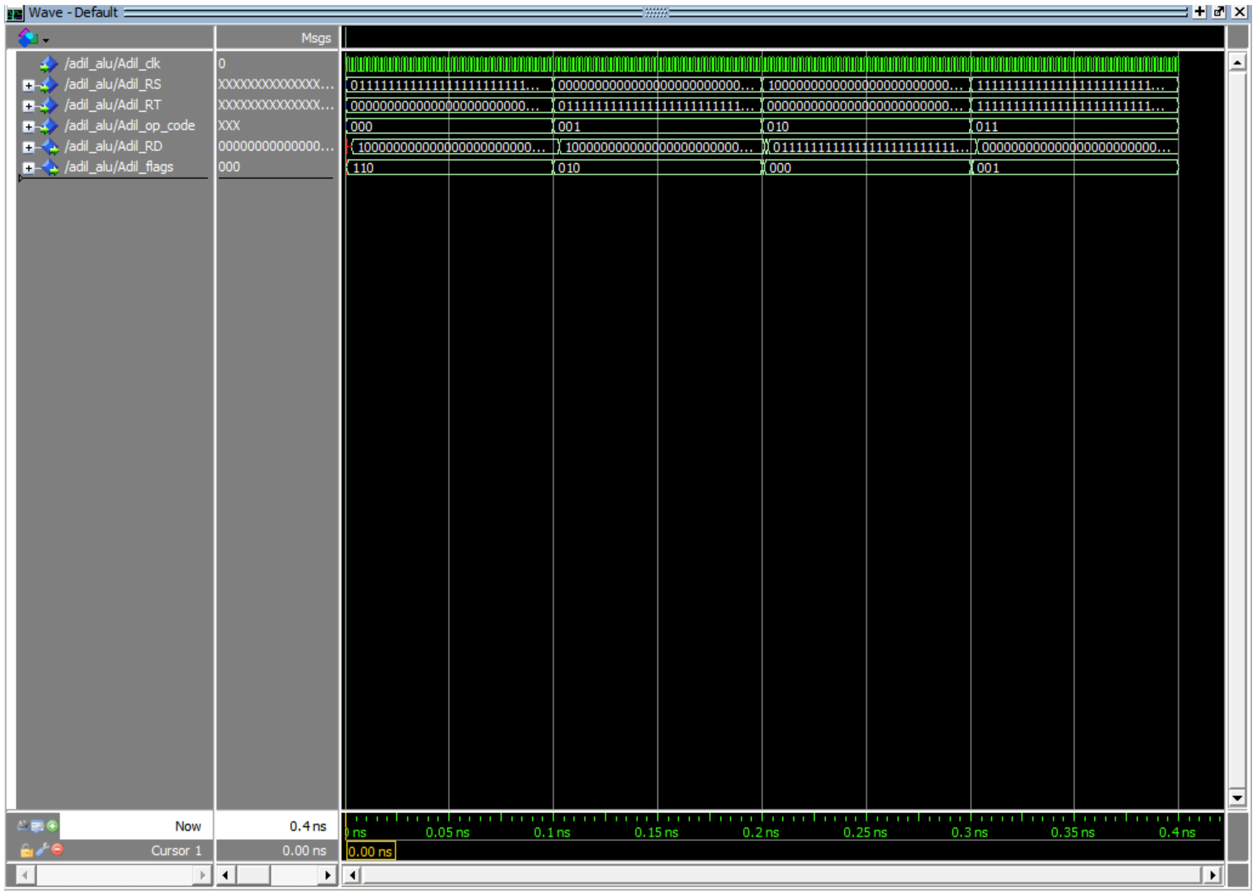


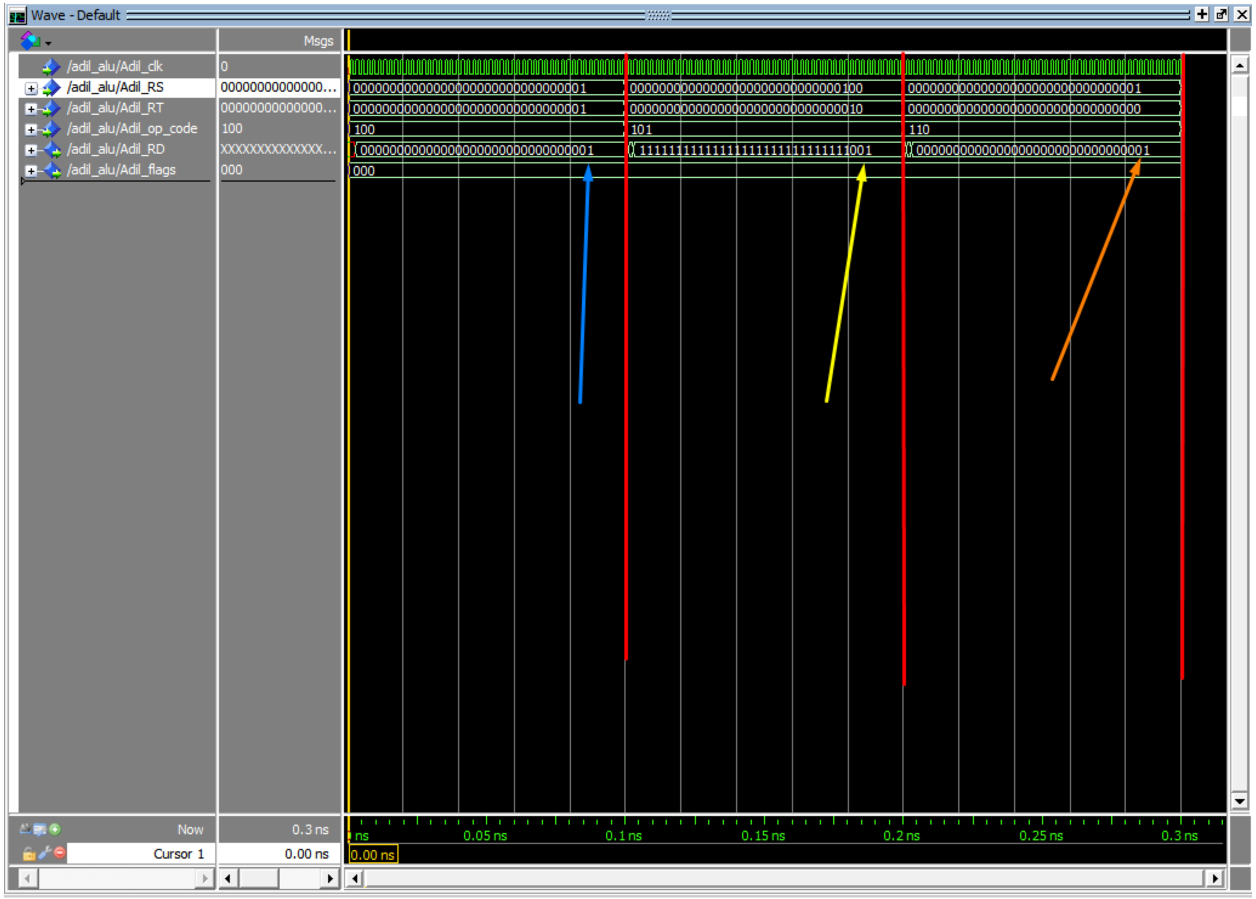
Figure 6:waveforms add/sub

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.



```
/adil_alu/Adil_clk      0
/adil_alu/Adil_RS       XXXXXXXXXXXXXX...  01111111111111111111111111...  00000000000000000000000000...  10000000000000000000000000...  1111111111111111111111111...
/adil_alu/Adil_RT       XXXXXXXXXXXXXX...  0000000000000000000000000...   0111111111111111111111111111... 00000000000000000000000000...  1111111111111111111111111...
/adil_alu/Adil_op_code  XXX                000                            001                             010                             011
/adil_alu/Adil_RD       0000000000000...   1000000000000000000000000...   1000000000000000000000000...    0111111111111111111111111...    0000000000000000000000000...
/adil_alu/Adil_flags    000                110                            010                             000                             001
```

Figure 7: AND



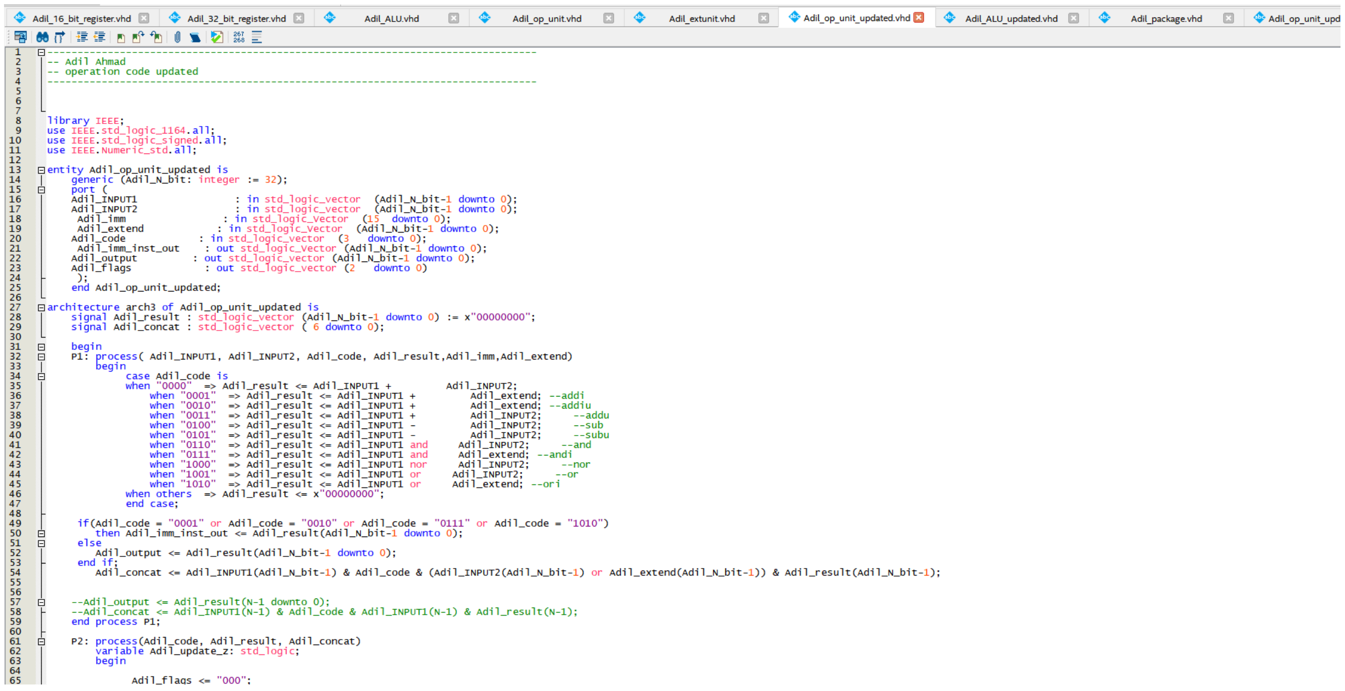
```
Wave - Default
                        Msgs
/adil_alu/Adil_clk      0
/adil_alu/Adil_RS       00000000000000...   00000000000000000000000000000001   00000000000000000000000000000100   00000000000000000000000000000001
/adil_alu/Adil_RT       00000000000000...   00000000000000000000000000000001   00000000000000000000000000000010   00000000000000000000000000000000
/adil_alu/Adil_op_code  100                 100                                101                                110
/adil_alu/Adil_RD       XXXXXXXXXXXXXX...   00000000000000000000000000000001   11111111111111111111111111111001   00000000000000000000000000000001
/adil_alu/Adil_flags    000                 000

Now       0.3 ns    ns   0.05 ns   0.1 ns   0.15 ns   0.2 ns   0.25 ns   0.3 ns
Cursor 1  0.00 ns   0.00 ns
```

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.

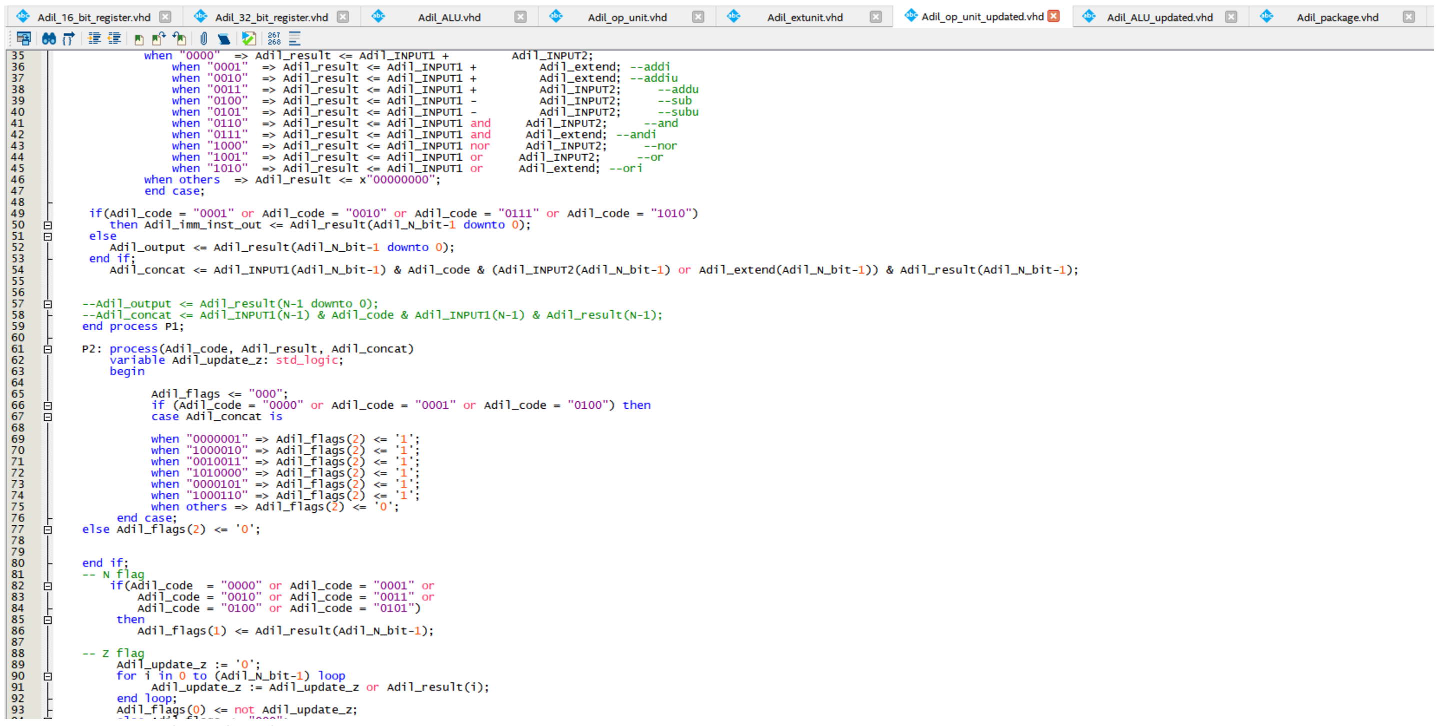
[ADDI, ADDIU, ANDI, ORI]

Adil, Ahmad
7/22/21
Professor Gertner
CSC 342/343



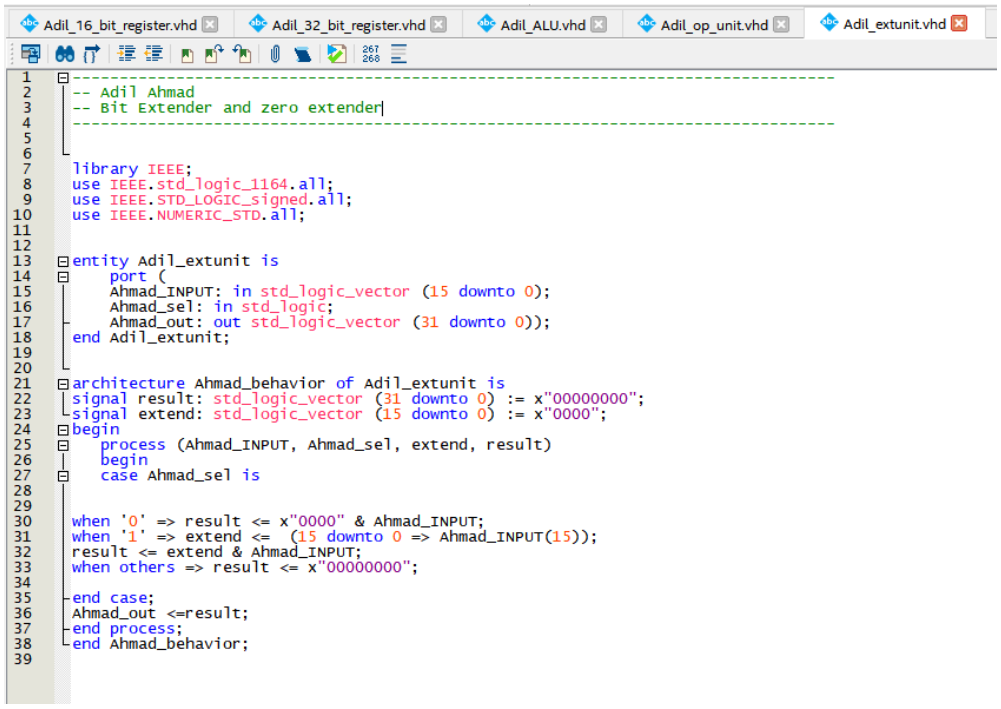
```vhdl
-------------------------------------------------------------------------------
-- Adil Ahmad
-- operation code updated
-------------------------------------------------------------------------------



library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.Numeric_std.all;

entity Adil_op_unit_updated is
    generic (Adil_N_bit: integer := 32);
    port (
    Adil_INPUT1                : in std_logic_vector  (Adil_N_bit-1 downto 0);
    Adil_INPUT2                : in std_logic_vector  (Adil_N_bit-1 downto 0);
     Adil_imm                : in std_logic_Vector  (15  downto 0);
     Adil_extend            : in std_logic_Vector  (Adil_N_bit-1 downto 0);
    Adil_code            : in std_logic_vector  (3   downto 0);
     Adil_imm_inst_out     : out std_logic_Vector (Adil_N_bit-1 downto 0);
    Adil_output          : out std_logic_vector (Adil_N_bit-1 downto 0);
    Adil_flags             : out std_logic_vector (2   downto 0)
     );
    end Adil_op_unit_updated;

architecture arch3 of Adil_op_unit_updated is
    signal Adil_result : std_logic_vector (Adil_N_bit-1 downto 0) := x"00000000";
    signal Adil_concat : std_logic_vector ( 6 downto 0);

    begin
    P1: process( Adil_INPUT1, Adil_INPUT2, Adil_code, Adil_result,Adil_imm,Adil_extend)
        begin
            case Adil_code is
            when "0000"  => Adil_result <= Adil_INPUT1 +         Adil_INPUT2;
                when "0001"  => Adil_result <= Adil_INPUT1 +         Adil_extend; --addi
                when "0010"  => Adil_result <= Adil_INPUT1 +         Adil_extend; --addiu
                when "0011"  => Adil_result <= Adil_INPUT1 +         Adil_INPUT2;     --addu
                when "0100"  => Adil_result <= Adil_INPUT1 -         Adil_INPUT2;     --sub
                when "0101"  => Adil_result <= Adil_INPUT1 -         Adil_INPUT2;     --subu
                when "0110"  => Adil_result <= Adil_INPUT1 and     Adil_INPUT2;     --and
                when "0111"  => Adil_result <= Adil_INPUT1 and     Adil_extend; --andi
                when "1000"  => Adil_result <= Adil_INPUT1 nor     Adil_INPUT2;     --nor
                when "1001"  => Adil_result <= Adil_INPUT1 or     Adil_INPUT2;     --or
                when "1010"  => Adil_result <= Adil_INPUT1 or     Adil_extend; --ori
            when others  => Adil_result <= x"00000000";
            end case;

     if(Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
        then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
     else
        Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
     end if;
        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);


    --Adil_output <= Adil_result(N-1 downto 0);
    --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT1(N-1) & Adil_result(N-1);
    end process P1;

    P2: process(Adil_code, Adil_result, Adil_concat)
        variable Adil_update_z: std_logic;
        begin

              Adil_flags <= "000";
```

Figure 9 OP code



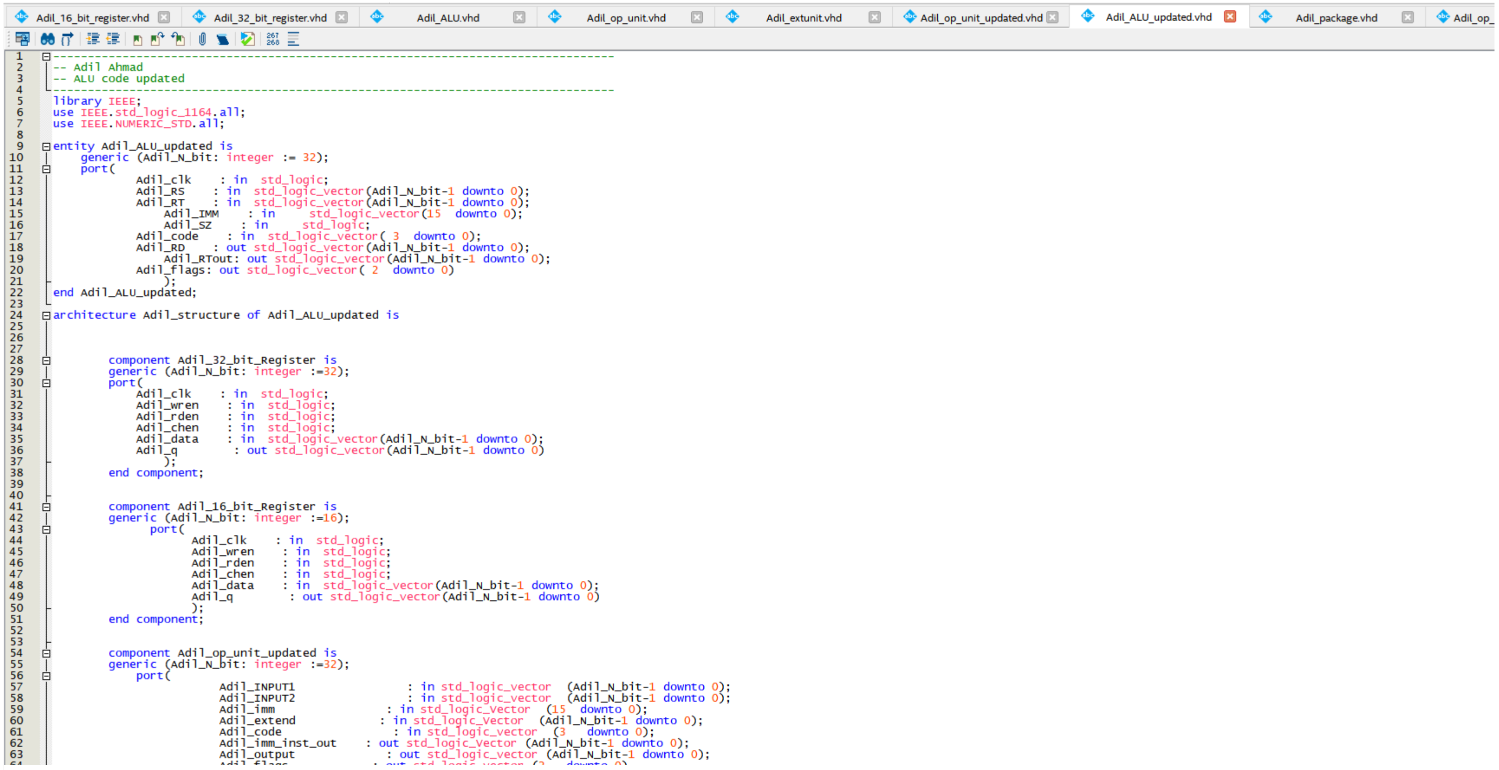
```vhdl
            when "0000"  => Adil_result <= Adil_INPUT1 +         Adil_INPUT2;
                when "0001"  => Adil_result <= Adil_INPUT1 +         Adil_extend; --addi
                when "0010"  => Adil_result <= Adil_INPUT1 +         Adil_extend; --addiu
                when "0011"  => Adil_result <= Adil_INPUT1 +         Adil_INPUT2;     --addu
                when "0100"  => Adil_result <= Adil_INPUT1 -         Adil_INPUT2;     --sub
                when "0101"  => Adil_result <= Adil_INPUT1 -         Adil_INPUT2;     --subu
                when "0110"  => Adil_result <= Adil_INPUT1 and     Adil_INPUT2;     --and
                when "0111"  => Adil_result <= Adil_INPUT1 and     Adil_extend; --andi
                when "1000"  => Adil_result <= Adil_INPUT1 nor     Adil_INPUT2;     --nor
                when "1001"  => Adil_result <= Adil_INPUT1 or     Adil_INPUT2;     --or
                when "1010"  => Adil_result <= Adil_INPUT1 or     Adil_extend; --ori
            when others  => Adil_result <= x"00000000";
            end case;

     if(Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
        then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
     else
        Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
     end if;
        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);


    --Adil_output <= Adil_result(N-1 downto 0);
    --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT1(N-1) & Adil_result(N-1);
    end process P1;

    P2: process(Adil_code, Adil_result, Adil_concat)
        variable Adil_update_z: std_logic;
        begin

              Adil_flags <= "000";
              if (Adil_code = "0000" or Adil_code = "0001" or Adil_code = "0100") then
              case Adil_concat is

              when "0000001" => Adil_flags(2) <= '1';
              when "1000010" => Adil_flags(2) <= '1';
              when "0010011" => Adil_flags(2) <= '1';
              when "1010000" => Adil_flags(2) <= '1';
              when "0000101" => Adil_flags(2) <= '1';
              when "1000110" => Adil_flags(2) <= '1';
              when others => Adil_flags(2) <= '0';
         end case;
    else Adil_flags(2) <= '0';


    end if;
    -- N flag
        if(Adil_code  = "0000" or Adil_code = "0001" or
           Adil_code = "0010" or Adil_code = "0011" or
           Adil_code = "0100" or Adil_code = "0101")
         then
           Adil_flags(1) <= Adil_result(Adil_N_bit-1);

    -- Z flag
         Adil_update_z := '0';
         for i in 0 to (Adil_N_bit-1) loop
              Adil_update_z := Adil_update_z or Adil_result(i);
         end loop;
         Adil_flags(0) <= not Adil_update_z;
```

Figure 10: OP code updated



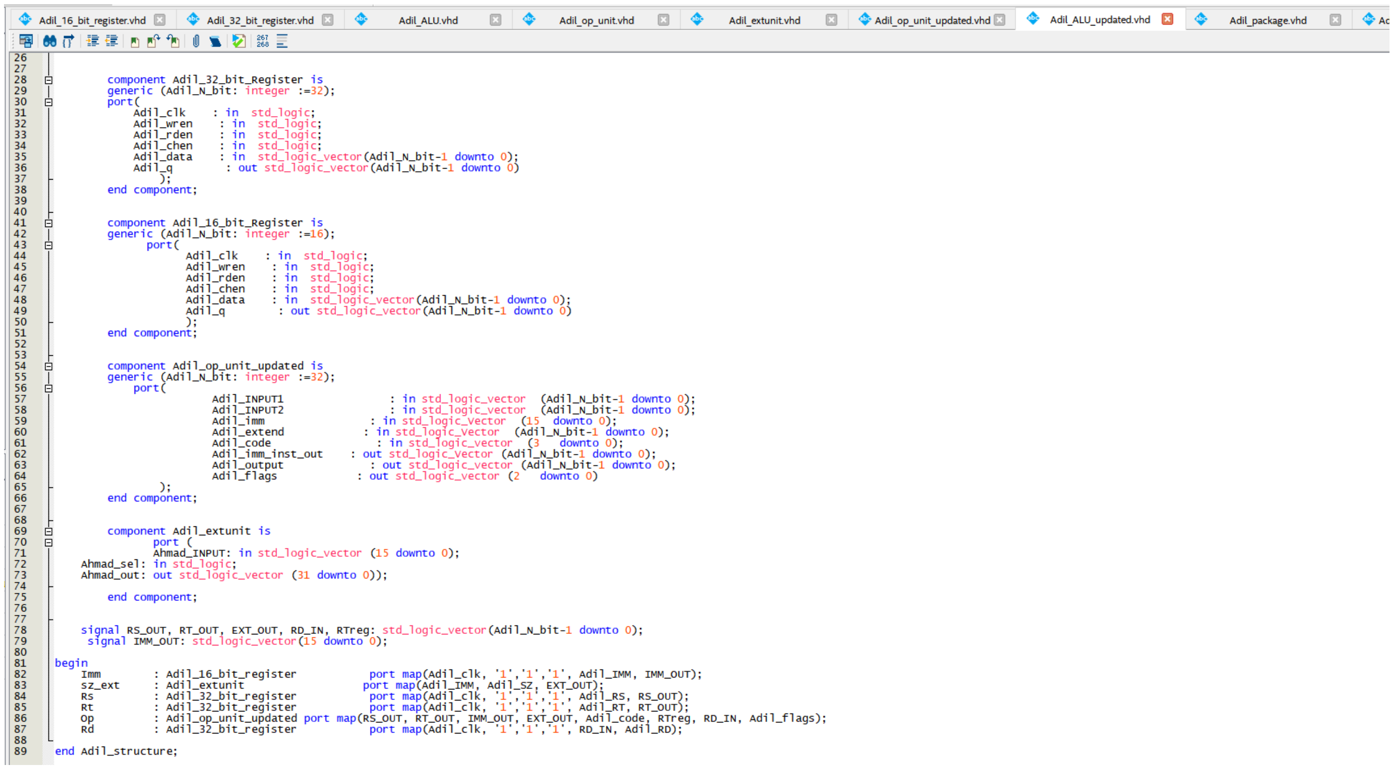
```vhdl
----------------------------------------------------------------------------------
-- Adil Ahmad
-- Bit Extender and zero extender
----------------------------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_signed.all;
use IEEE.NUMERIC_STD.all;


entity Adil_extunit is
    port (
    Ahmad_INPUT: in std_logic_vector (15 downto 0);
    Ahmad_sel: in std_logic;
    Ahmad_out: out std_logic_vector (31 downto 0));
end Adil_extunit;


architecture Ahmad_behavior of Adil_extunit is
signal result: std_logic_vector (31 downto 0) := x"00000000";
signal extend: std_logic_vector (15 downto 0) := x"0000";
begin
   process (Ahmad_INPUT, Ahmad_sel, extend, result)
   begin
   case Ahmad_sel is


when '0' => result <= x"0000" & Ahmad_INPUT;
when '1' => extend <=  (15 downto 0 => Ahmad_INPUT(15));
result <= extend & Ahmad_INPUT;
when others => result <= x"00000000";

end case;
Ahmad_out <=result;
end process;
end Ahmad_behavior;
```

Figure 11: Extended



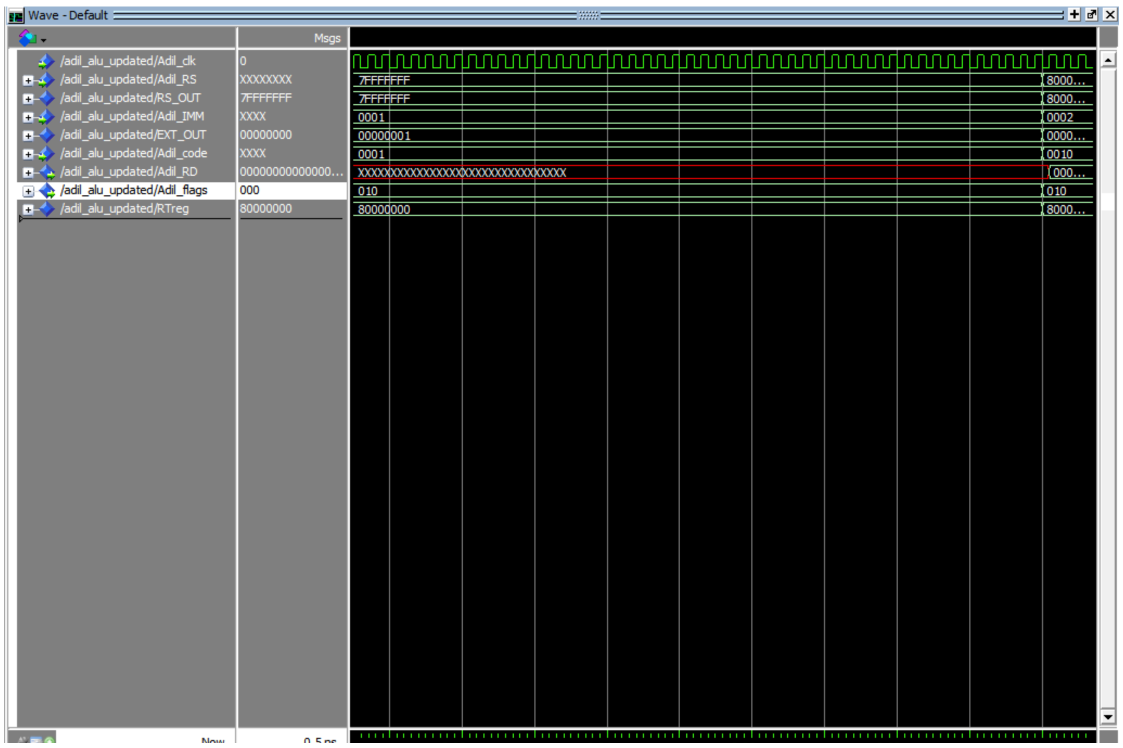
```vhdl
----------------------------------------------------------------------------------
-- Adil Ahmad
-- ALU code updated
----------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;

entity Adil_ALU_updated is
    generic (Adil_N_bit: integer := 32);
    port(
            Adil_clk    : in  std_logic;
            Adil_RS    : in  std_logic_vector(Adil_N_bit-1 downto 0);
            Adil_RT    : in  std_logic_vector(Adil_N_bit-1 downto 0);
                Adil_IMM     : in     std_logic_vector(15  downto 0);
                Adil_SZ     : in     std_logic;
            Adil_code    : in  std_logic_vector( 3  downto 0);
            Adil_RD    : out std_logic_vector(Adil_N_bit-1 downto 0);
                Adil_RTout: out std_logic_vector(Adil_N_bit-1 downto 0);
            Adil_flags: out std_logic_vector( 2  downto 0)
                );
end Adil_ALU_updated;

architecture Adil_structure of Adil_ALU_updated 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(Adil_N_bit-1 downto 0);
            Adil_q        : out std_logic_vector(Adil_N_bit-1 downto 0)
                );
        end component;


        component Adil_16_bit_Register is
        generic (Adil_N_bit: integer :=16);
              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(Adil_N_bit-1 downto 0);
                    Adil_q        : out std_logic_vector(Adil_N_bit-1 downto 0)
                    );
        end component;


        component Adil_op_unit_updated is
        generic (Adil_N_bit: integer :=32);
            port(
                        Adil_INPUT1                 : in std_logic_vector  (Adil_N_bit-1 downto 0);
                        Adil_INPUT2                 : in std_logic_vector  (Adil_N_bit-1 downto 0);
                        Adil_imm                 : in std_logic_vector  (15  downto 0);
                        Adil_extend             : in std_logic_vector  (Adil_N_bit-1 downto 0);
                        Adil_code                 : in std_logic_vector  (3   downto 0);
                        Adil_imm_inst_out    : out std_logic_vector (Adil_N_bit-1 downto 0);
                        Adil_output             : out std_logic_vector (Adil_N_bit-1 downto 0);
```

Figure 12 ALU



```vhdl
	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(Adil_N_bit-1 downto 0);
	    Adil_q         : out std_logic_vector(Adil_N_bit-1 downto 0)
	        );
	end component;


	component Adil_16_bit_Register is
	generic (Adil_N_bit: integer :=16);
	      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(Adil_N_bit-1 downto 0);
	            Adil_q         : out std_logic_vector(Adil_N_bit-1 downto 0)
	            );
	end component;


	component Adil_op_unit_updated is
	generic (Adil_N_bit: integer :=32);
	    port(
	                Adil_INPUT1                 : in std_logic_vector  (Adil_N_bit-1 downto 0);
	                Adil_INPUT2                 : in std_logic_vector  (Adil_N_bit-1 downto 0);
	                Adil_imm                : in std_logic_vector  (15  downto 0);
	                Adil_extend            : in std_logic_vector  (Adil_N_bit-1 downto 0);
	                Adil_code                : in std_logic_vector  (3   downto 0);
	                Adil_imm_inst_out    : out std_logic_vector (Adil_N_bit-1 downto 0);
	                Adil_output               : out std_logic_vector (Adil_N_bit-1 downto 0);
	                Adil_flags            : out std_logic_vector (2   downto 0)
	        );
	end component;


	component Adil_extunit is
	       port (
	       Ahmad_INPUT: in std_logic_vector (15 downto 0);
    Ahmad_sel: in std_logic;
    Ahmad_out: out std_logic_vector (31 downto 0));

	end component;


    signal RS_OUT, RT_OUT, EXT_OUT, RD_IN, RTreg: std_logic_vector(Adil_N_bit-1 downto 0);
     signal IMM_OUT: std_logic_vector(15 downto 0);

begin
    Imm        : Adil_16_bit_register          port map(Adil_clk, '1','1','1', Adil_IMM, IMM_OUT);
    sz_ext     : Adil_extunit                 port map(Adil_IMM, Adil_SZ, EXT_OUT);
    Rs         : Adil_32_bit_register          port map(Adil_clk, '1','1','1', Adil_RS, RS_OUT);
    Rt         : Adil_32_bit_register          port map(Adil_clk, '1','1','1', Adil_RT, RT_OUT);
    Op         : Adil_op_unit_updated port map(RS_OUT, RT_OUT, IMM_OUT, EXT_OUT, Adil_code, RTreg, RD_IN, Adil_flags);
    Rd         : Adil_32_bit_register          port map(Adil_clk, '1','1','1', RD_IN, Adil_RD);

end Adil_structure;
```

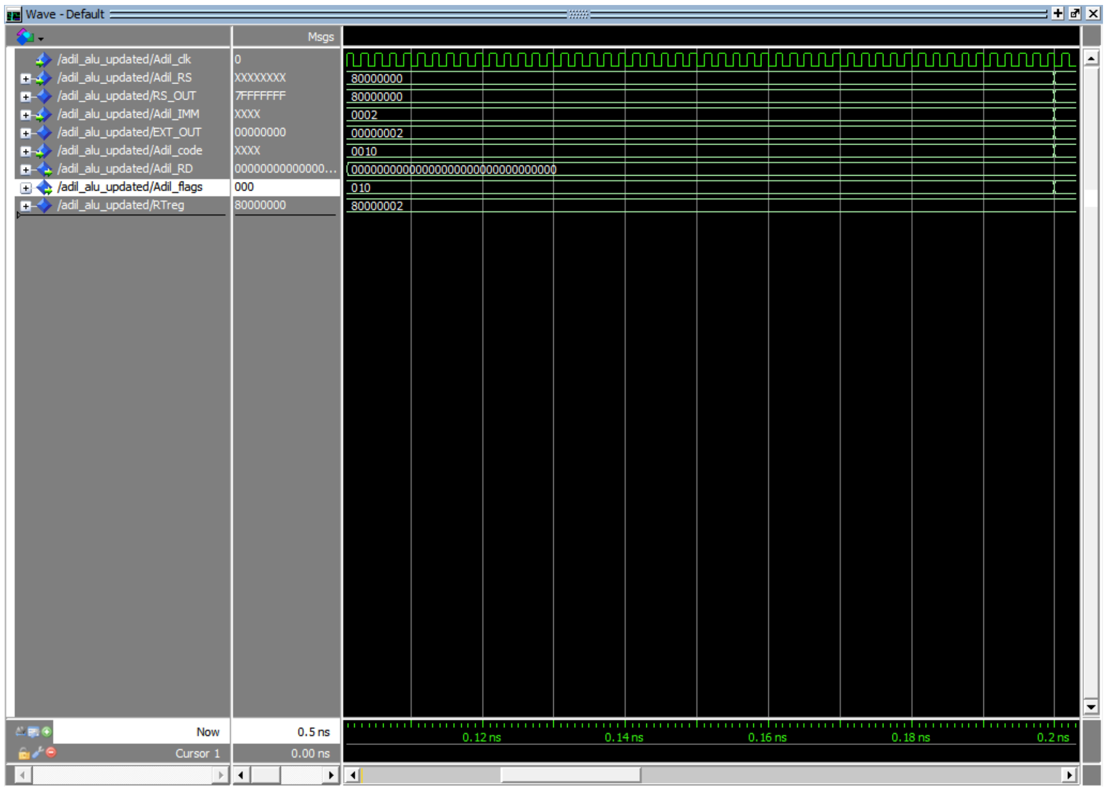
Figure 14ALU



```
/adil_alu_updated/Adil_clk      0
/adil_alu_updated/Adil_RS       XXXXXXXX          7FFFFFFF                          8000...
/adil_alu_updated/RS_OUT        7FFFFFFF          7FFFFFFF                          8000...
/adil_alu_updated/Adil_IMM      XXXX              0001                              0002
/adil_alu_updated/EXT_OUT       00000000          00000001                          0000...
/adil_alu_updated/Adil_code     XXXX              0001                              0010
/adil_alu_updated/Adil_RD       00000000000000... XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  000...
/adil_alu_updated/Adil_flags    000               010                               010
/adil_alu_updated/RTreg         80000000          80000000                          8000...
```

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].

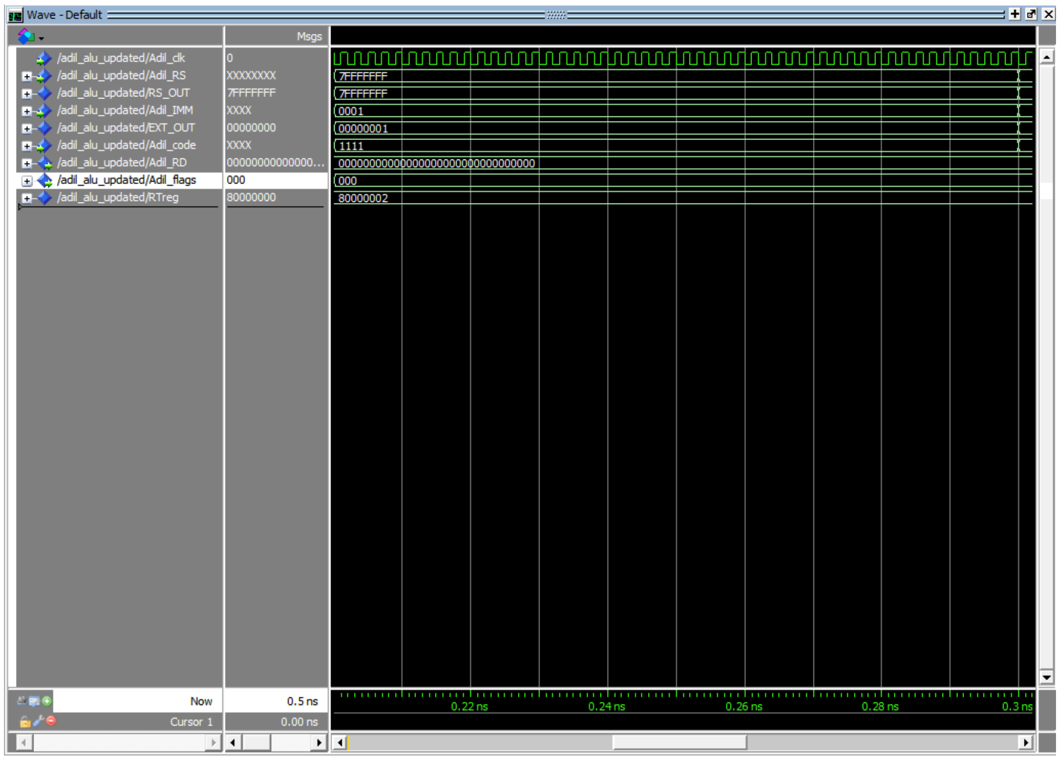


```
/adil_alu_updated/Adil_clk        0
/adil_alu_updated/Adil_RS         XXXXXXXX             80000000
/adil_alu_updated/RS_OUT          7FFFFFFF             80000000
/adil_alu_updated/Adil_IMM        XXXX                 0002
/adil_alu_updated/EXT_OUT         00000000             00000002
/adil_alu_updated/Adil_code       XXXX                 0010
/adil_alu_updated/Adil_RD         00000000000000...    00000000000000000000000000000000
/adil_alu_updated/Adil_flags      000                  010
/adil_alu_updated/RTreg           80000000             80000002
```

Figure 15 addiu

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

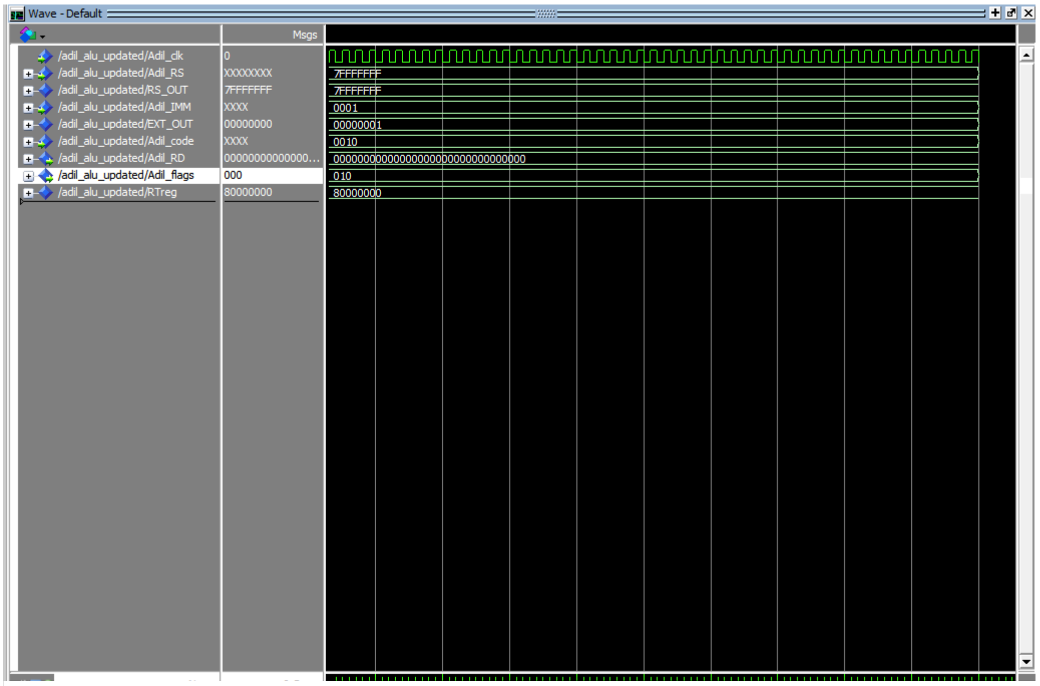
Adil, Ahmad
7/22/21
Professor Gertner
CSC 342/343



```
Wave - Default
                                   Msgs
/adil_alu_updated/Adil_clk         0
/adil_alu_updated/Adil_RS          XXXXXXXX           7FFFFFFF
/adil_alu_updated/RS_OUT           7FFFFFFF           7FFFFFFF
/adil_alu_updated/Adil_IMM         XXXX               0001
/adil_alu_updated/EXT_OUT          00000000           00000001
/adil_alu_updated/Adil_code        XXXX               1111
/adil_alu_updated/Adil_RD          00000000000000...  00000000000000000000000000000000
/adil_alu_updated/Adil_flags       000                000
/adil_alu_updated/RTreg            80000000           80000002

Now          0.5 ns        0.22 ns   0.24 ns   0.26 ns   0.28 ns   0.3 ns
Cursor 1     0.00 ns
```

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.

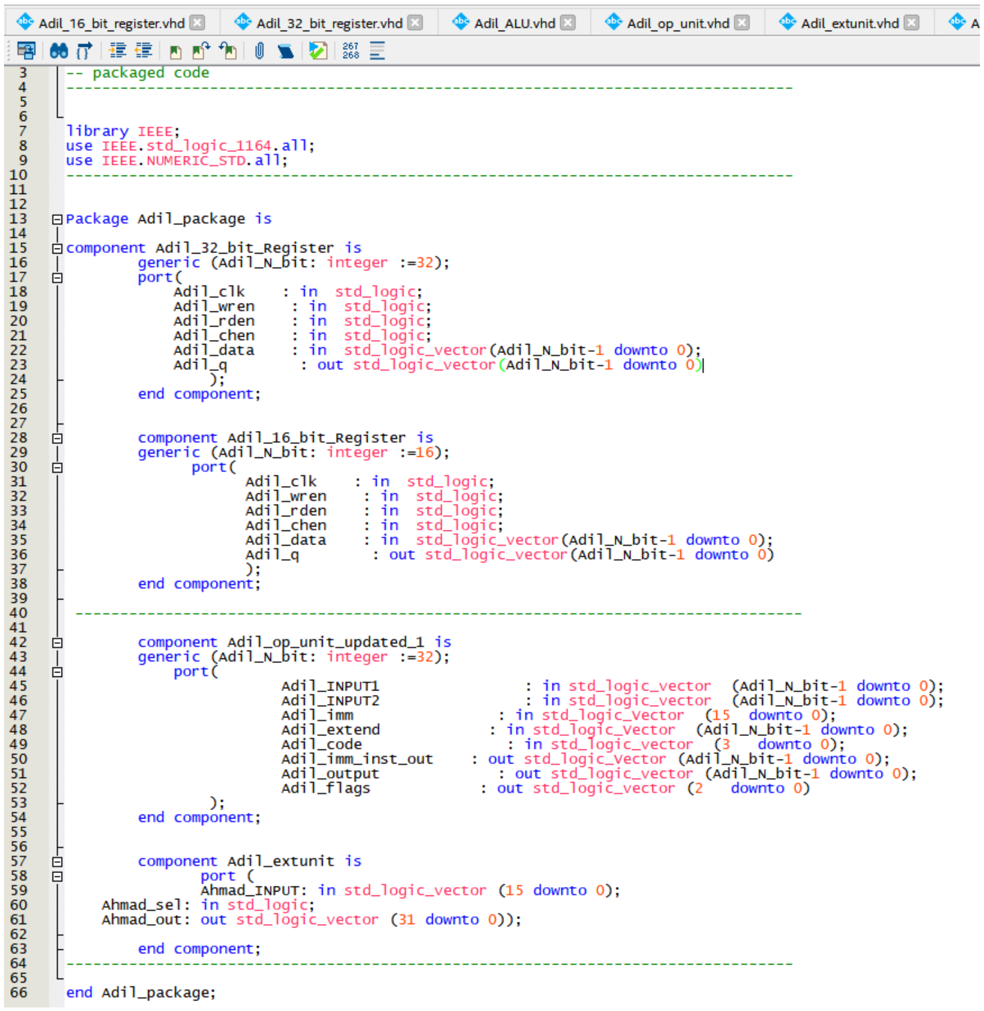


```
Wave - Default
                                   Msgs
/adil_alu_updated/Adil_clk         0
/adil_alu_updated/Adil_RS          XXXXXXXX           7FFFFFFF
/adil_alu_updated/RS_OUT           7FFFFFFF           7FFFFFFF
/adil_alu_updated/Adil_IMM         XXXX               0001
/adil_alu_updated/EXT_OUT          00000000           00000001
/adil_alu_updated/Adil_code        XXXX               0010
/adil_alu_updated/Adil_RD          00000000000000...  00000000000000000000000000000000
/adil_alu_updated/Adil_flags       000                010
/adil_alu_updated/RTreg            80000000           80000000
```

Figure 17: ORI

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

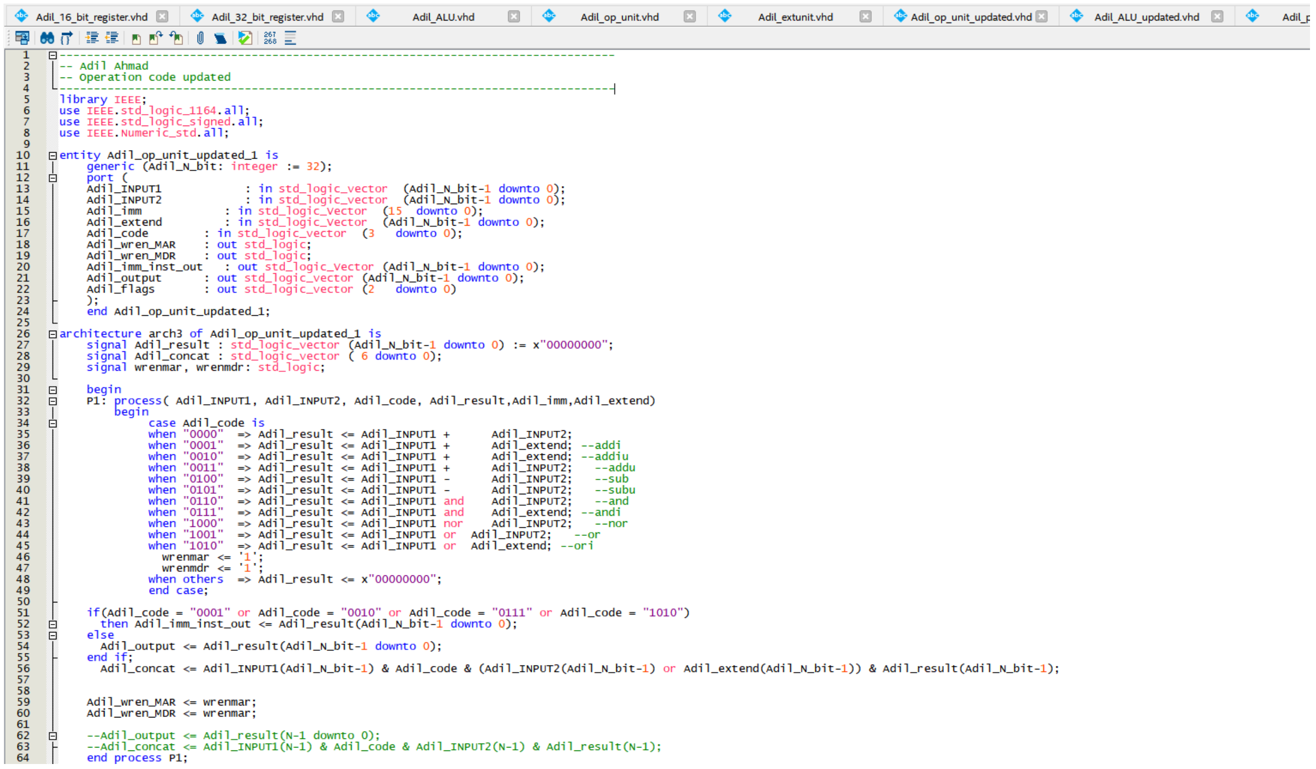
Load Word (LW)



```
-- packaged code
--------------------------------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;
--------------------------------------------------------------------------------


Package Adil_package 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(Adil_N_bit-1 downto 0);
            Adil_q        : out std_logic_vector(Adil_N_bit-1 downto 0)
                );
        end component;


        component Adil_16_bit_Register is
        generic (Adil_N_bit: integer :=16);
              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(Adil_N_bit-1 downto 0);
                    Adil_q        : out std_logic_vector(Adil_N_bit-1 downto 0)
                    );
        end component;

 --------------------------------------------------------------------------------

        component Adil_op_unit_updated_1 is
        generic (Adil_N_bit: integer :=32);
            port(
                        Adil_INPUT1                 : in std_logic_vector  (Adil_N_bit-1 downto 0);
                        Adil_INPUT2                 : in std_logic_vector  (Adil_N_bit-1 downto 0);
                        Adil_imm                 : in std_logic_vector  (15  downto 0);
                        Adil_extend             : in std_logic_vector  (Adil_N_bit-1 downto 0);
                        Adil_code                  : in std_logic_vector  (3   downto 0);
                        Adil_imm_inst_out     : out std_logic_vector (Adil_N_bit-1 downto 0);
                        Adil_output              : out std_logic_vector (Adil_N_bit-1 downto 0);
                        Adil_flags             : out std_logic_vector (2   downto 0)
                );
        end component;


        component Adil_extunit is
                port (
                Ahmad_INPUT: in std_logic_vector (15 downto 0);
    Ahmad_sel: in std_logic;
    Ahmad_out: out std_logic_vector (31 downto 0));

        end component;
--------------------------------------------------------------------------------

end Adil_package;
```

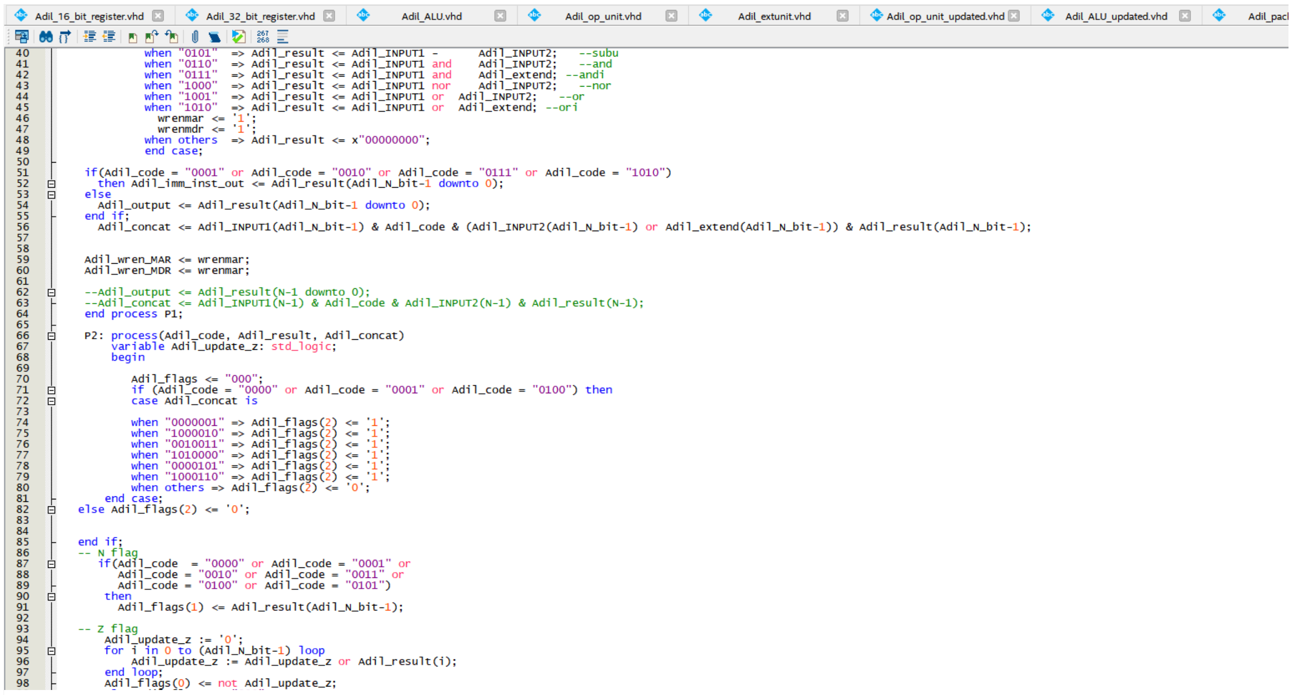
Figure 18: Packaged

Adil, Ahmad
7/22/21
Professor Gertner
CSC 342/343



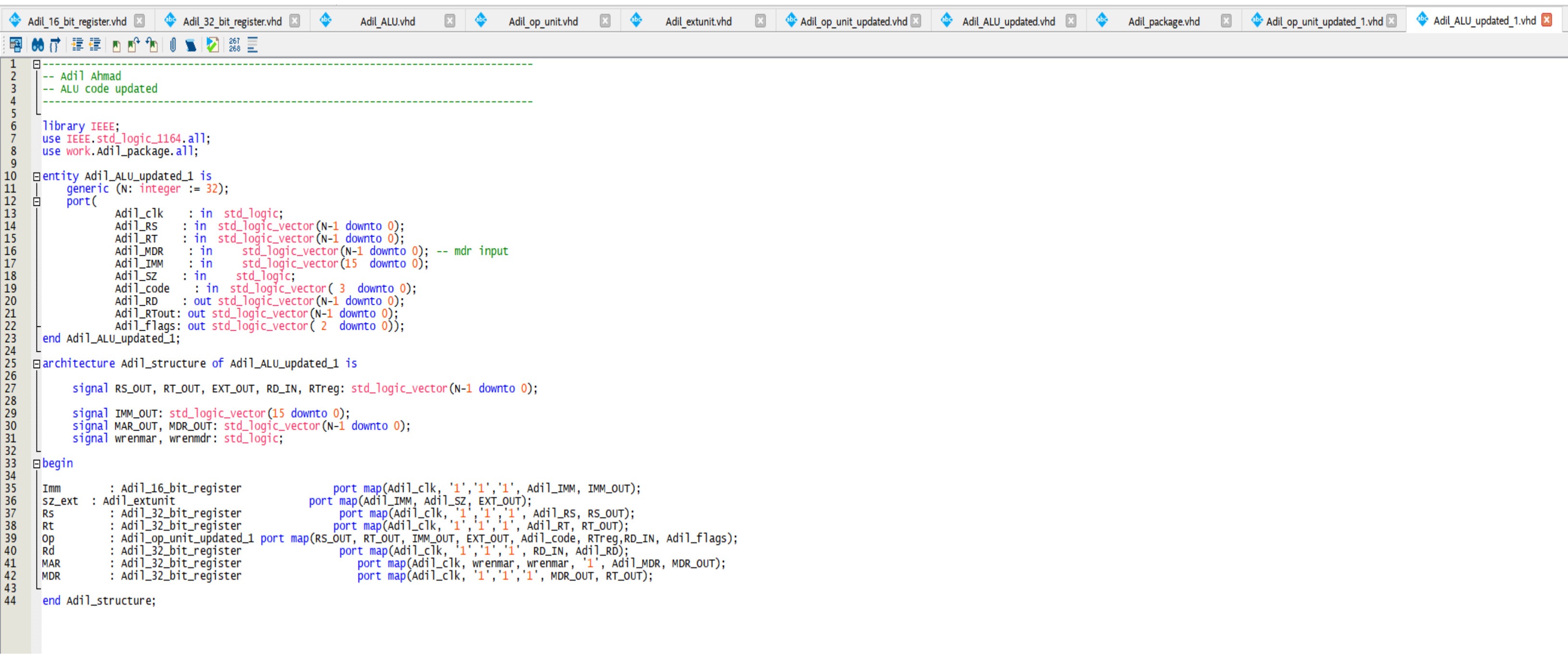
```vhdl
--------------------------------------------------------------------------------
-- Adil Ahmad
-- Operation code updated
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.Numeric_std.all;

entity Adil_op_unit_updated_1 is
    generic (Adil_N_bit: integer := 32);
    port (
    Adil_INPUT1             : in std_logic_vector  (Adil_N_bit-1 downto 0);
    Adil_INPUT2             : in std_logic_vector  (Adil_N_bit-1 downto 0);
    Adil_imm             : in std_logic_Vector  (15  downto 0);
    Adil_extend          : in std_logic_Vector  (Adil_N_bit-1 downto 0);
    Adil_code        : in std_logic_vector  (3   downto 0);
    Adil_wren_MAR    : out std_logic;
    Adil_wren_MDR    : out std_logic;
    Adil_imm_inst_out   : out std_logic_Vector (Adil_N_bit-1 downto 0);
    Adil_output      : out std_logic_vector (Adil_N_bit-1 downto 0);
    Adil_flags       : out std_logic_vector (2   downto 0)
    );
    end Adil_op_unit_updated_1;

architecture arch3 of Adil_op_unit_updated_1 is
    signal Adil_result : std_logic_vector (Adil_N_bit-1 downto 0) := x"00000000";
    signal Adil_concat : std_logic_vector ( 6 downto 0);
    signal wrenmar, wrenmdr: std_logic;

    begin
    P1: process( Adil_INPUT1, Adil_INPUT2, Adil_code, Adil_result,Adil_imm,Adil_extend)
        begin
            case Adil_code is
            when "0000"  => Adil_result <= Adil_INPUT1 +      Adil_INPUT2;
            when "0001"  => Adil_result <= Adil_INPUT1 +      Adil_extend; --addi
            when "0010"  => Adil_result <= Adil_INPUT1 +      Adil_extend; --addiu
            when "0011"  => Adil_result <= Adil_INPUT1 +      Adil_INPUT2;   --addu
            when "0100"  => Adil_result <= Adil_INPUT1 -      Adil_INPUT2;   --sub
            when "0101"  => Adil_result <= Adil_INPUT1 -      Adil_INPUT2;   --subu
            when "0110"  => Adil_result <= Adil_INPUT1 and    Adil_INPUT2;   --and
            when "0111"  => Adil_result <= Adil_INPUT1 and    Adil_extend; --andi
            when "1000"  => Adil_result <= Adil_INPUT1 nor    Adil_INPUT2;   --nor
            when "1001"  => Adil_result <= Adil_INPUT1 or  Adil_INPUT2;   --or
            when "1010"  => Adil_result <= Adil_INPUT1 or  Adil_extend; --ori
              wrenmar <= '1';
              wrenmdr <= '1';
            when others  => Adil_result <= x"00000000";
            end case;

    if(Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
      then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
    else
      Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
    end if;
      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);


    Adil_wren_MAR <= wrenmar;
    Adil_wren_MDR <= wrenmar;

    --Adil_output <= Adil_result(N-1 downto 0);
    --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT2(N-1) & Adil_result(N-1);
    end process P1;
```

Figure 19: OP code updated



```vhdl
            when "0101"  => Adil_result <= Adil_INPUT1 -      Adil_INPUT2;   --subu
            when "0110"  => Adil_result <= Adil_INPUT1 and    Adil_INPUT2;   --and
            when "0111"  => Adil_result <= Adil_INPUT1 and    Adil_extend; --andi
            when "1000"  => Adil_result <= Adil_INPUT1 nor    Adil_INPUT2;   --nor
            when "1001"  => Adil_result <= Adil_INPUT1 or  Adil_INPUT2;   --or
            when "1010"  => Adil_result <= Adil_INPUT1 or  Adil_extend; --ori
              wrenmar <= '1';
              wrenmdr <= '1';
            when others  => Adil_result <= x"00000000";
            end case;

    if(Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
      then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
    else
      Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
    end if;
      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);


    Adil_wren_MAR <= wrenmar;
    Adil_wren_MDR <= wrenmar;

    --Adil_output <= Adil_result(N-1 downto 0);
    --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT2(N-1) & Adil_result(N-1);
    end process P1;

    P2: process(Adil_code, Adil_result, Adil_concat)
        variable Adil_update_z: std_logic;
        begin

           Adil_flags <= "000";
           if (Adil_code = "0000" or Adil_code = "0001" or Adil_code = "0100") then
           case Adil_concat is

           when "0000001" => Adil_flags(2) <= '1';
           when "1000010" => Adil_flags(2) <= '1';
           when "0010011" => Adil_flags(2) <= '1';
           when "1010000" => Adil_flags(2) <= '1';
           when "0000101" => Adil_flags(2) <= '1';
           when "1000110" => Adil_flags(2) <= '1';
           when others => Adil_flags(2) <= '0';
       end case;
   else Adil_flags(2) <= '0';


   end if;
   -- N flag
      if(Adil_code  = "0000" or Adil_code = "0001" or
         Adil_code = "0010" or Adil_code = "0011" or
         Adil_code = "0100" or Adil_code = "0101")
       then
         Adil_flags(1) <= Adil_result(Adil_N_bit-1);

   -- Z flag
       Adil_update_z := '0';
       for i in 0 to (Adil_N_bit-1) loop
           Adil_update_z := Adil_update_z or Adil_result(i);
       end loop;
       Adil_flags(0) <= not Adil_update_z;
```

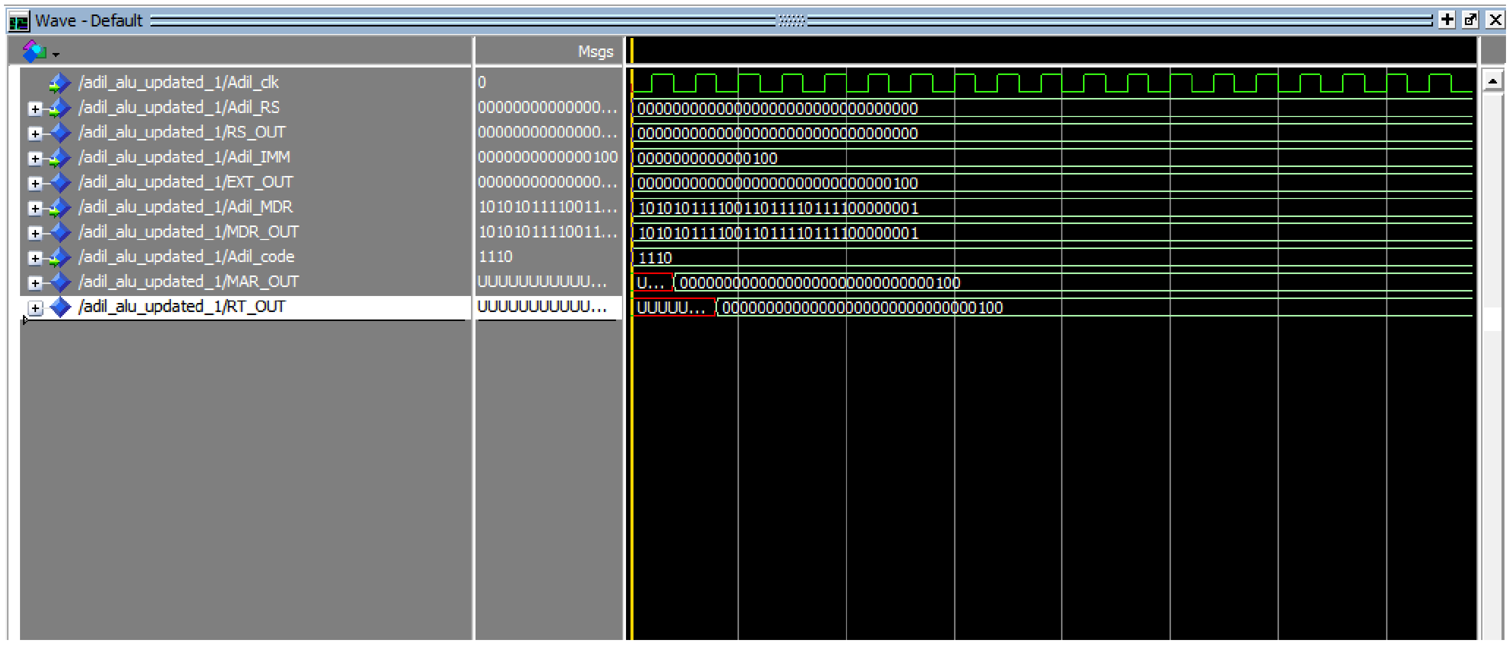
Figure 20: OP Code updated



```
------------------------------------------------------------------------
-- Adil Ahmad
-- ALU code updated
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.Adil_package.all;

entity Adil_ALU_updated_1 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_MDR    : in     std_logic_vector(N-1 downto 0); -- mdr input
            Adil_IMM    : in     std_logic_vector(15  downto 0);
            Adil_SZ    : in     std_logic;
            Adil_code    : in  std_logic_vector( 3  downto 0);
            Adil_RD    : out std_logic_vector(N-1 downto 0);
            Adil_RTout: out std_logic_vector(N-1 downto 0);
            Adil_flags: out std_logic_vector( 2  downto 0));
end Adil_ALU_updated_1;

architecture Adil_structure of Adil_ALU_updated_1 is

     signal RS_OUT, RT_OUT, EXT_OUT, RD_IN, RTreg: std_logic_vector(N-1 downto 0);

     signal IMM_OUT: std_logic_vector(15 downto 0);
     signal MAR_OUT, MDR_OUT: std_logic_vector(N-1 downto 0);
     signal wrenmar, wrenmdr: std_logic;

begin

Imm        : Adil_16_bit_register              port map(Adil_clk, '1','1','1', Adil_IMM, IMM_OUT);
sz_ext  : Adil_extunit                    port map(Adil_IMM, Adil_SZ, EXT_OUT);
Rs         : Adil_32_bit_register               port map(Adil_clk, '1','1','1', Adil_RS, RS_OUT);
Rt         : Adil_32_bit_register              port map(Adil_clk, '1','1','1', Adil_RT, RT_OUT);
Op         : Adil_op_unit_updated_1 port map(RS_OUT, RT_OUT, IMM_OUT, EXT_OUT, Adil_code, RTreg,RD_IN, Adil_flags);
Rd         : Adil_32_bit_register               port map(Adil_clk, '1','1','1', RD_IN, Adil_RD);
MAR        : Adil_32_bit_register                  port map(Adil_clk, wrenmar, wrenmar, '1', Adil_MDR, MDR_OUT);
MDR        : Adil_32_bit_register                  port map(Adil_clk, '1','1','1', MDR_OUT, RT_OUT);

end Adil_structure;
```

Figure 21 ALU updated

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

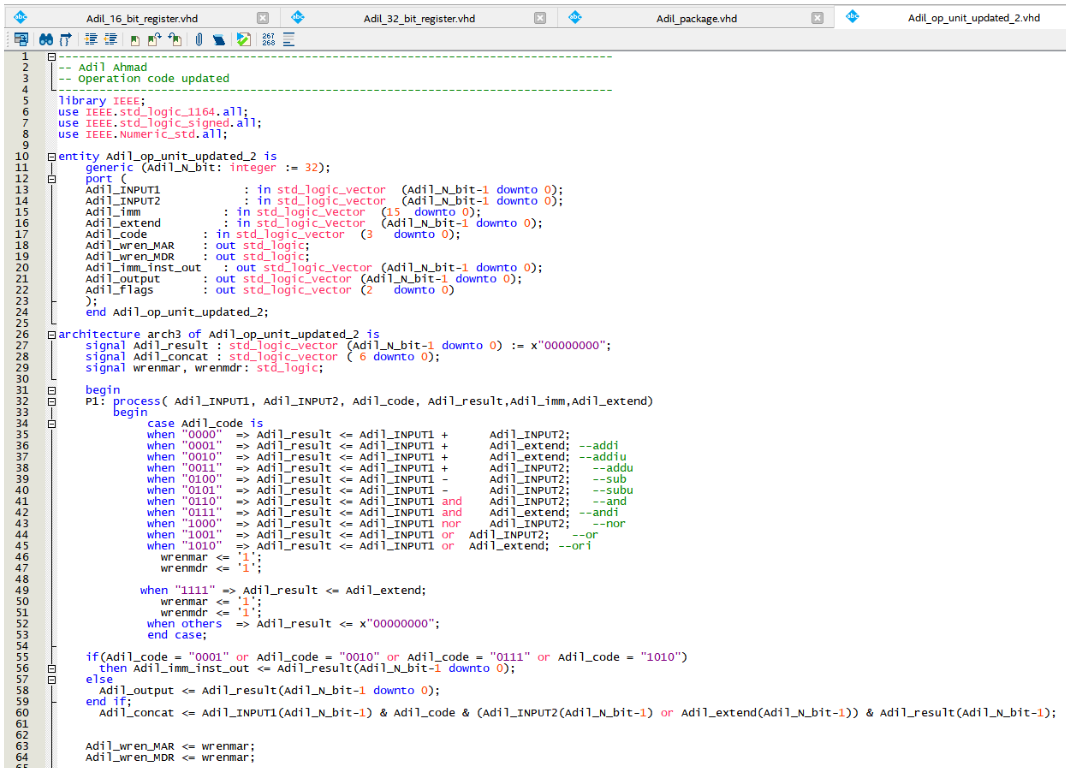


```
Wave - Default
                                   Msgs
/adil_alu_updated_1/Adil_clk       0
/adil_alu_updated_1/Adil_RS        00000000000000...   00000000000000000000000000000000
/adil_alu_updated_1/RS_OUT         00000000000000...   00000000000000000000000000000000
/adil_alu_updated_1/Adil_IMM       0000000000000100    0000000000000100
/adil_alu_updated_1/EXT_OUT        00000000000000...   00000000000000000000000000000100
/adil_alu_updated_1/Adil_MDR       10101011110011...   10101011110011011110111100000001
/adil_alu_updated_1/MDR_OUT        10101011110011...   10101011110011011110111100000001
/adil_alu_updated_1/Adil_code      1110                1110
/adil_alu_updated_1/MAR_OUT        UUUUUUUUUUU...      U... 00000000000000000000000000000100
/adil_alu_updated_1/RT_OUT         UUUUUUUUUUU...      UUUUU... 00000000000000000000000000000100
```

Figure 22: LW

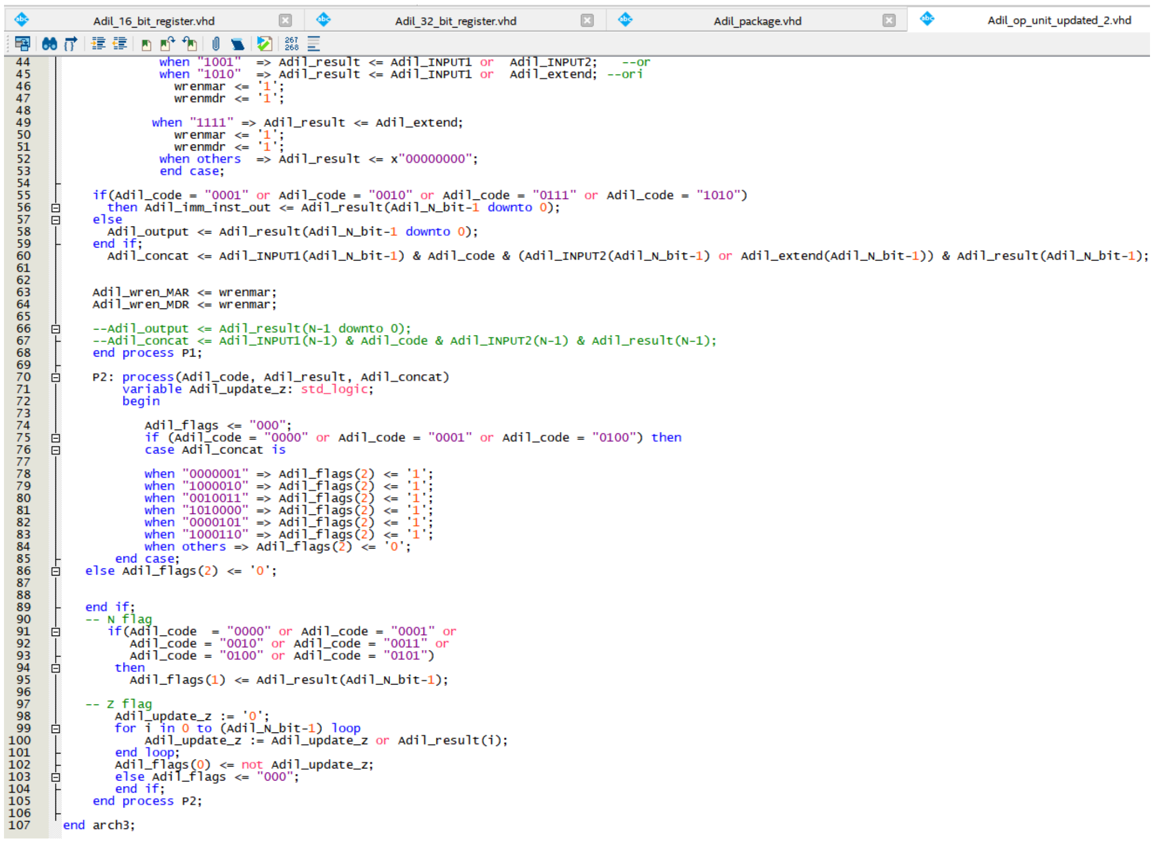
For LW, this initialized MDR to 0xABCDEF01 so we can update register R[rt]. For this part, MAR = R[rs] + signextend = 0 + 4 = 0x00000004. Aftwards R[rt] becomes the value of MDR, which is 0xABCDEF01. So then data is loadead to address 0x00000000 with offset of 4.

Store Word (SW)


Adil_16_bit_register.vhd | Adil_32_bit_register.vhd | Adil_package.vhd | Adil_op_unit_updated_2.vhd

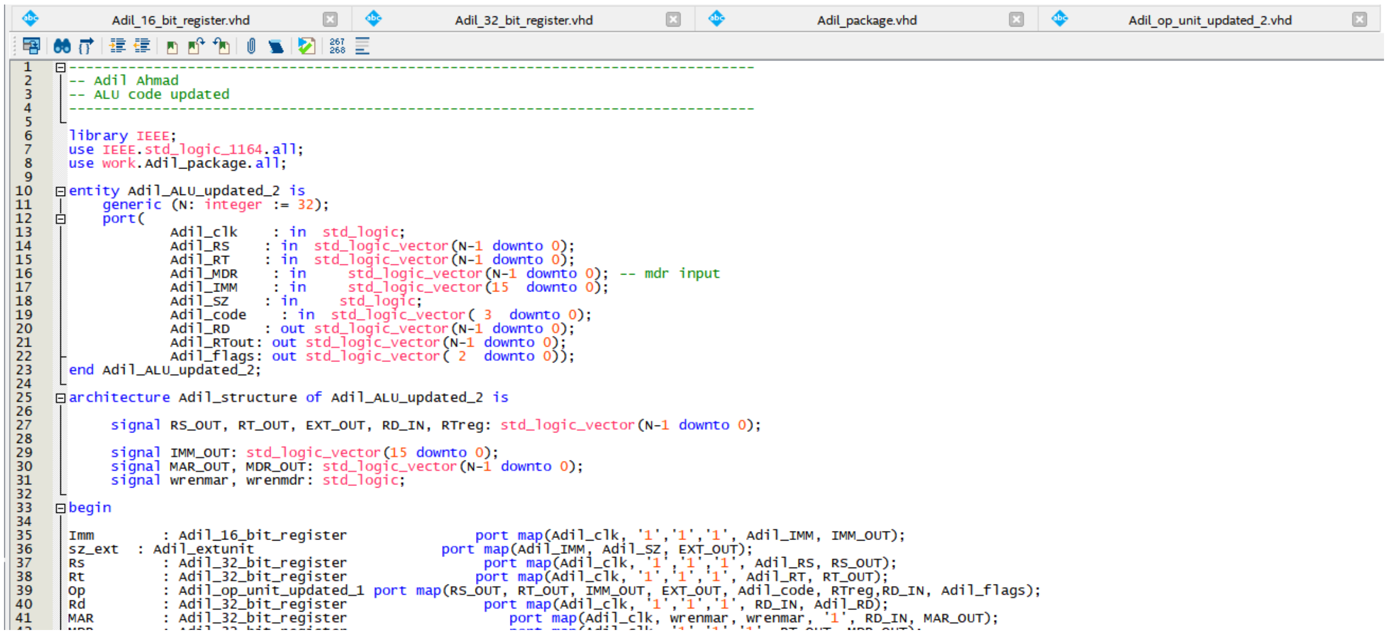
```vhdl
--------------------------------------------------------------------------------
-- Adil Ahmad
-- Operation code updated
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.Numeric_std.all;

entity Adil_op_unit_updated_2 is
    generic (Adil_N_bit: integer := 32);
    port (
    Adil_INPUT1             : in std_logic_vector   (Adil_N_bit-1 downto 0);
    Adil_INPUT2             : in std_logic_vector   (Adil_N_bit-1 downto 0);
    Adil_imm             : in std_logic_Vector   (15  downto 0);
    Adil_extend          : in std_logic_Vector   (Adil_N_bit-1 downto 0);
    Adil_code         : in std_logic_vector   (3   downto 0);
    Adil_wren_MAR     : out std_logic;
    Adil_wren_MDR     : out std_logic;
    Adil_imm_inst_out    : out std_logic_Vector (Adil_N_bit-1 downto 0);
    Adil_output       : out std_logic_vector (Adil_N_bit-1 downto 0);
    Adil_flags        : out std_logic_vector (2   downto 0)
    );
    end Adil_op_unit_updated_2;

architecture arch3 of Adil_op_unit_updated_2 is
    signal Adil_result : std_logic_vector (Adil_N_bit-1 downto 0) := x"00000000";
    signal Adil_concat : std_logic_vector ( 6 downto 0);
    signal wrenmar, wrenmdr: std_logic;

    begin
    P1: process( Adil_INPUT1, Adil_INPUT2, Adil_code, Adil_result,Adil_imm,Adil_extend)
        begin
            case Adil_code is
            when "0000"  => Adil_result <= Adil_INPUT1 +      Adil_INPUT2;
            when "0001"  => Adil_result <= Adil_INPUT1 +      Adil_extend; --addi
            when "0010"  => Adil_result <= Adil_INPUT1 +      Adil_extend; --addiu
            when "0011"  => Adil_result <= Adil_INPUT1 +      Adil_INPUT2;   --addu
            when "0100"  => Adil_result <= Adil_INPUT1 -      Adil_INPUT2;   --sub
            when "0101"  => Adil_result <= Adil_INPUT1 -      Adil_INPUT2;   --subu
            when "0110"  => Adil_result <= Adil_INPUT1 and    Adil_INPUT2;   --and
            when "0111"  => Adil_result <= Adil_INPUT1 and    Adil_extend; --andi
            when "1000"  => Adil_result <= Adil_INPUT1 nor    Adil_INPUT2;   --nor
            when "1001"  => Adil_result <= Adil_INPUT1 or  Adil_INPUT2;   --or
            when "1010"  => Adil_result <= Adil_INPUT1 or  Adil_extend; --ori
              wrenmar <= '1';
              wrenmdr <= '1';

           when "1111" => Adil_result <= Adil_extend;
              wrenmar <= '1';
              wrenmdr <= '1';
            when others  => Adil_result <= x"00000000";
            end case;

    if(Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
      then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
    else
      Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
    end if;
      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);


    Adil_wren_MAR <= wrenmar;
    Adil_wren_MDR <= wrenmar;
```

Figure 23: SW



```
                when "1001"  => Adil_result <= Adil_INPUT1 or  Adil_INPUT2;   --or
                when "1010"  => Adil_result <= Adil_INPUT1 or  Adil_extend; --ori
                  wrenmar <= '1';
                  wrenmdr <= '1';

               when "1111" => Adil_result <= Adil_extend;
                  wrenmar <= '1';
                  wrenmdr <= '1';
                when others  => Adil_result <= x"00000000";
                end case;

        if(Adil_code = "0001" or Adil_code = "0010" or Adil_code = "0111" or Adil_code = "1010")
          then Adil_imm_inst_out <= Adil_result(Adil_N_bit-1 downto 0);
        else
          Adil_output <= Adil_result(Adil_N_bit-1 downto 0);
        end if;
          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);


        Adil_wren_MAR <= wrenmar;
        Adil_wren_MDR <= wrenmar;

        --Adil_output <= Adil_result(N-1 downto 0);
        --Adil_concat <= Adil_INPUT1(N-1) & Adil_code & Adil_INPUT2(N-1) & Adil_result(N-1);
        end process P1;

        P2: process(Adil_code, Adil_result, Adil_concat)
            variable Adil_update_z: std_logic;
            begin

               Adil_flags <= "000";
               if (Adil_code = "0000" or Adil_code = "0001" or Adil_code = "0100") then
               case Adil_concat is

               when "0000001" => Adil_flags(2) <= '1';
               when "1000010" => Adil_flags(2) <= '1';
               when "0010011" => Adil_flags(2) <= '1';
               when "1010000" => Adil_flags(2) <= '1';
               when "0000101" => Adil_flags(2) <= '1';
               when "1000110" => Adil_flags(2) <= '1';
               when others => Adil_flags(2) <= '0';
            end case;
       else Adil_flags(2) <= '0';


       end if;
       -- N flag
          if(Adil_code  = "0000" or Adil_code = "0001" or
             Adil_code = "0010" or Adil_code = "0011" or
             Adil_code = "0100" or Adil_code = "0101")
           then
             Adil_flags(1) <= Adil_result(Adil_N_bit-1);

       -- Z flag
           Adil_update_z := '0';
           for i in 0 to (Adil_N_bit-1) loop
               Adil_update_z := Adil_update_z or Adil_result(i);
           end loop;
           Adil_flags(0) <= not Adil_update_z;
           else Adil_flags <= "000";
           end if;
        end process P2;

end arch3;
```

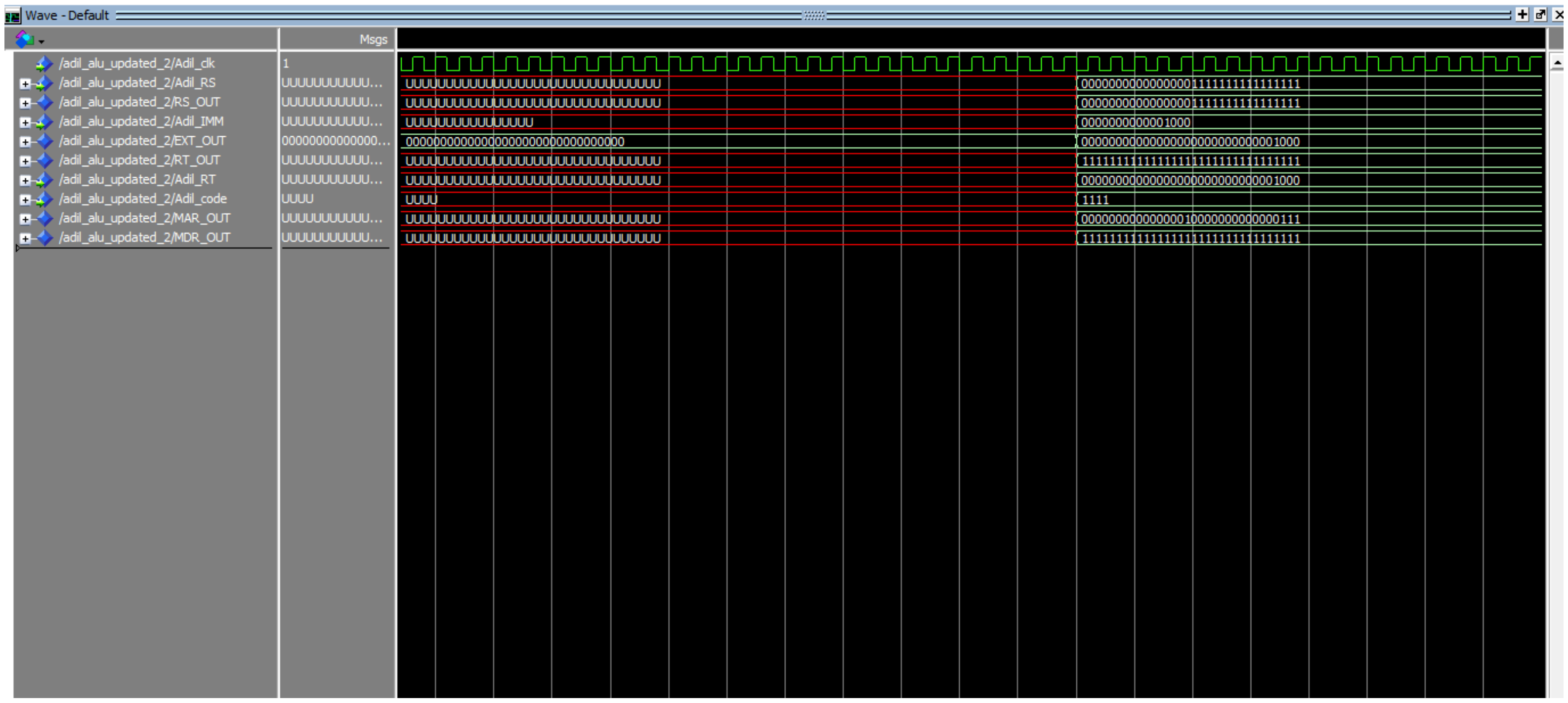
Figure 24: SW OP



```
------------------------------------------------------------------------
-- Adil Ahmad
-- ALU code updated
------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use work.Adil_package.all;

entity Adil_ALU_updated_2 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_MDR    : in     std_logic_vector(N-1 downto 0); -- mdr input
             Adil_IMM    : in     std_logic_vector(15  downto 0);
             Adil_SZ    : in     std_logic;
             Adil_code    : in  std_logic_vector( 3  downto 0);
             Adil_RD    : out std_logic_vector(N-1 downto 0);
             Adil_RTout: out std_logic_vector(N-1 downto 0);
             Adil_flags: out std_logic_vector( 2  downto 0));
end Adil_ALU_updated_2;

architecture Adil_structure of Adil_ALU_updated_2 is

      signal RS_OUT, RT_OUT, EXT_OUT, RD_IN, RTreg: std_logic_vector(N-1 downto 0);

      signal IMM_OUT: std_logic_vector(15 downto 0);
      signal MAR_OUT, MDR_OUT: std_logic_vector(N-1 downto 0);
      signal wrenmar, wrenmdr: std_logic;

begin

Imm        : Adil_16_bit_register                port map(Adil_clk, '1','1','1', Adil_IMM, IMM_OUT);
sz_ext   : Adil_extunit                    port map(Adil_IMM, Adil_SZ, EXT_OUT);
Rs         : Adil_32_bit_register                 port map(Adil_clk, '1','1','1', Adil_RS, RS_OUT);
Rt         : Adil_32_bit_register                port map(Adil_clk, '1','1','1', Adil_RT, RT_OUT);
Op         : Adil_op_unit_updated_1 port map(RS_OUT, RT_OUT, IMM_OUT, EXT_OUT, Adil_code, RTreg,RD_IN, Adil_flags);
Rd         : Adil_32_bit_register                 port map(Adil_clk, '1','1','1', RD_IN, Adil_RD);
MAR        : Adil_32_bit_register                    port map(Adil_clk, wrenmar, wrenmar, '1', RD_IN, MAR_OUT);
```

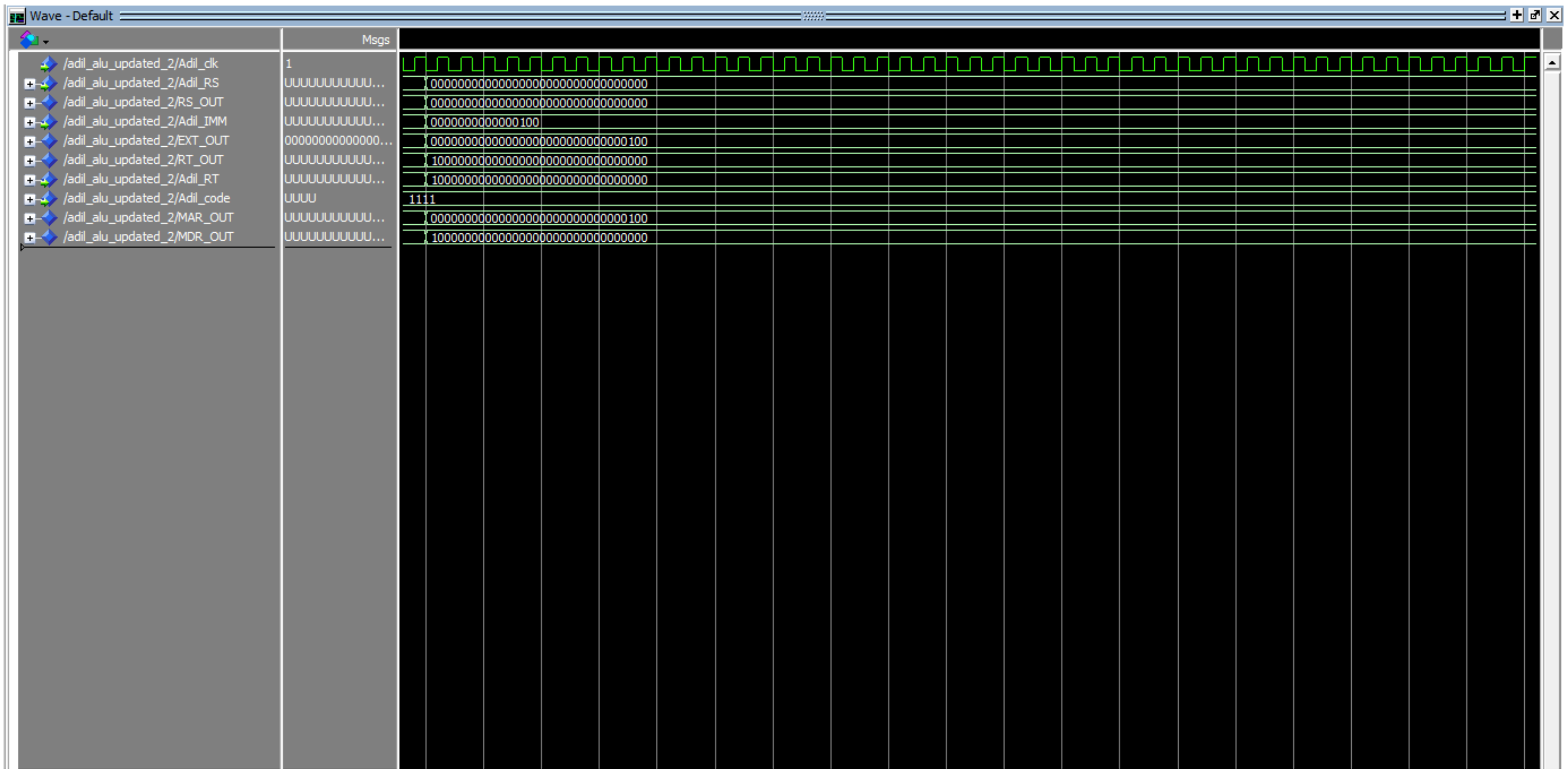
Figure 25: ALU updated

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



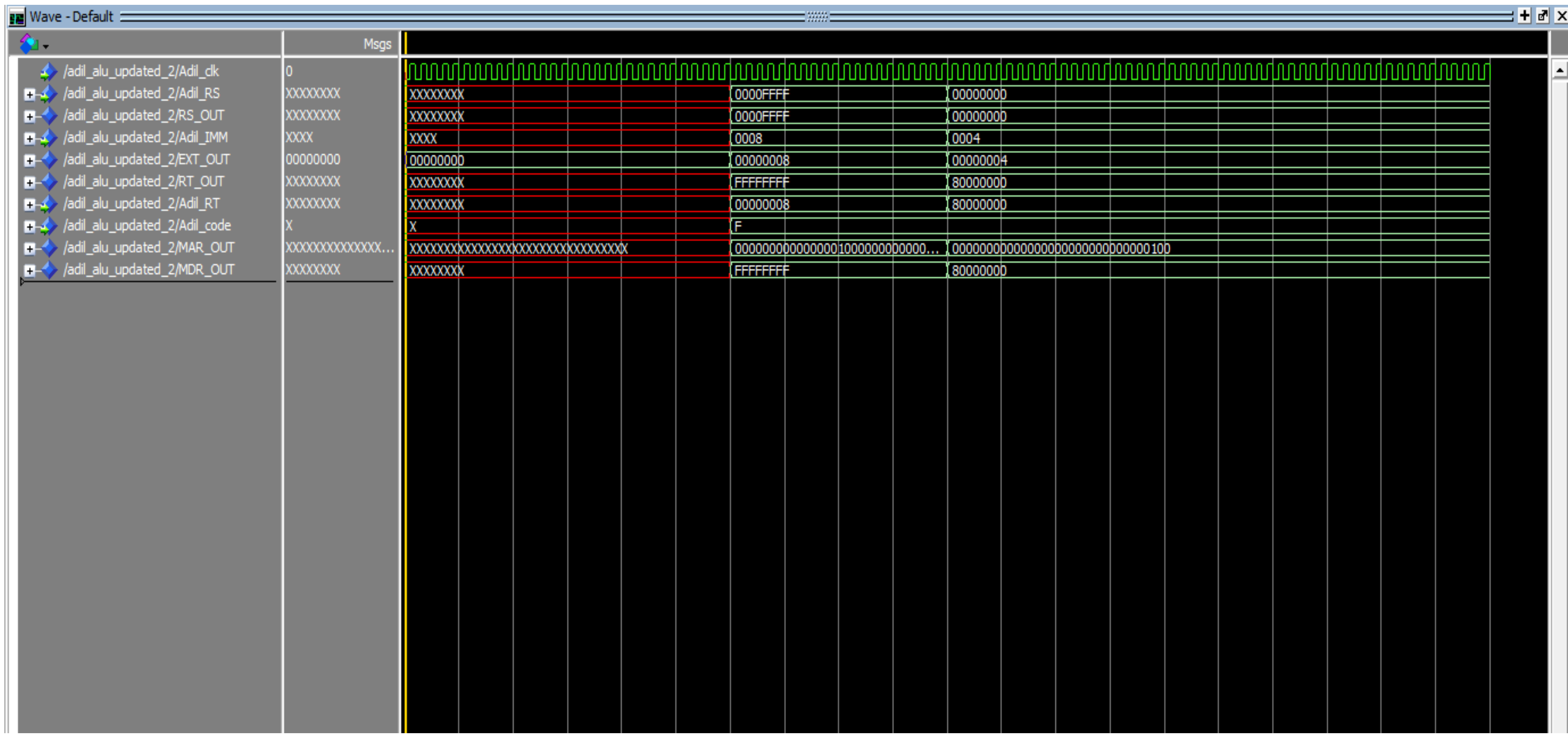
```
Wave - Default
                                  Msgs
/adil_alu_updated_2/Adil_clk      1
/adil_alu_updated_2/Adil_RS       UUUUUUUUUUU...   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   00000000000000001111111111111111
/adil_alu_updated_2/RS_OUT        UUUUUUUUUUU...   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   00000000000000001111111111111111
/adil_alu_updated_2/Adil_IMM      UUUUUUUUUUU...   UUUUUUUUUUUUUUUU                   0000000000001000
/adil_alu_updated_2/EXT_OUT       00000000000000...00000000000000000000000000000000   00000000000000000000000000001000
/adil_alu_updated_2/RT_OUT        UUUUUUUUUUU...   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   11111111111111111111111111111111
/adil_alu_updated_2/Adil_RT       UUUUUUUUUUU...   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   00000000000000000000000000001000
/adil_alu_updated_2/Adil_code     UUUU             UUUU                               1111
/adil_alu_updated_2/MAR_OUT       UUUUUUUUUUU...   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   00000000000000010000000000000111
/adil_alu_updated_2/MDR_OUT       UUUUUUUUUUU...   UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   11111111111111111111111111111111
```

Figure 27: Final ALU



```
Wave - Default
                                  Msgs
/adil_alu_updated_2/Adil_clk      1
/adil_alu_updated_2/Adil_RS       UUUUUUUUUUU...   00000000000000000000000000000000
/adil_alu_updated_2/RS_OUT        UUUUUUUUUUU...   00000000000000000000000000000000
/adil_alu_updated_2/Adil_IMM      UUUUUUUUUUU...   0000000000000100
/adil_alu_updated_2/EXT_OUT       00000000000000...00000000000000000000000000000100
/adil_alu_updated_2/RT_OUT        UUUUUUUUUUU...   10000000000000000000000000000000
/adil_alu_updated_2/Adil_RT       UUUUUUUUUUU...   10000000000000000000000000000000
/adil_alu_updated_2/Adil_code     UUUU             1111
/adil_alu_updated_2/MAR_OUT       UUUUUUUUUUU...   00000000000000000000000000000100
/adil_alu_updated_2/MDR_OUT       UUUUUUUUUUU...   10000000000000000000000000000000
```

Figure 26: final ALU



```
/adil_alu_updated_2/Adil_clk    0
/adil_alu_updated_2/Adil_RS     XXXXXXXX          XXXXXXXX                          0000FFFF                          00000000
/adil_alu_updated_2/RS_OUT      XXXXXXXX          XXXXXXXX                          0000FFFF                          00000000
/adil_alu_updated_2/Adil_IMM    XXXX              XXXX                              0008                              0004
/adil_alu_updated_2/EXT_OUT     00000000          00000000                          00000008                          00000004
/adil_alu_updated_2/RT_OUT      XXXXXXXX          XXXXXXXX                          FFFFFFFF                          80000000
/adil_alu_updated_2/Adil_RT     XXXXXXXX          XXXXXXXX                          00000008                          80000000
/adil_alu_updated_2/Adil_code   X                 X                                 F
/adil_alu_updated_2/MAR_OUT     XXXXXXXXXXXXXX... XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  000000000000000100000000000... 00000000000000000000000000000100
/adil_alu_updated_2/MDR_OUT     XXXXXXXX          XXXXXXXX                          FFFFFFFF                          80000000
```

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.