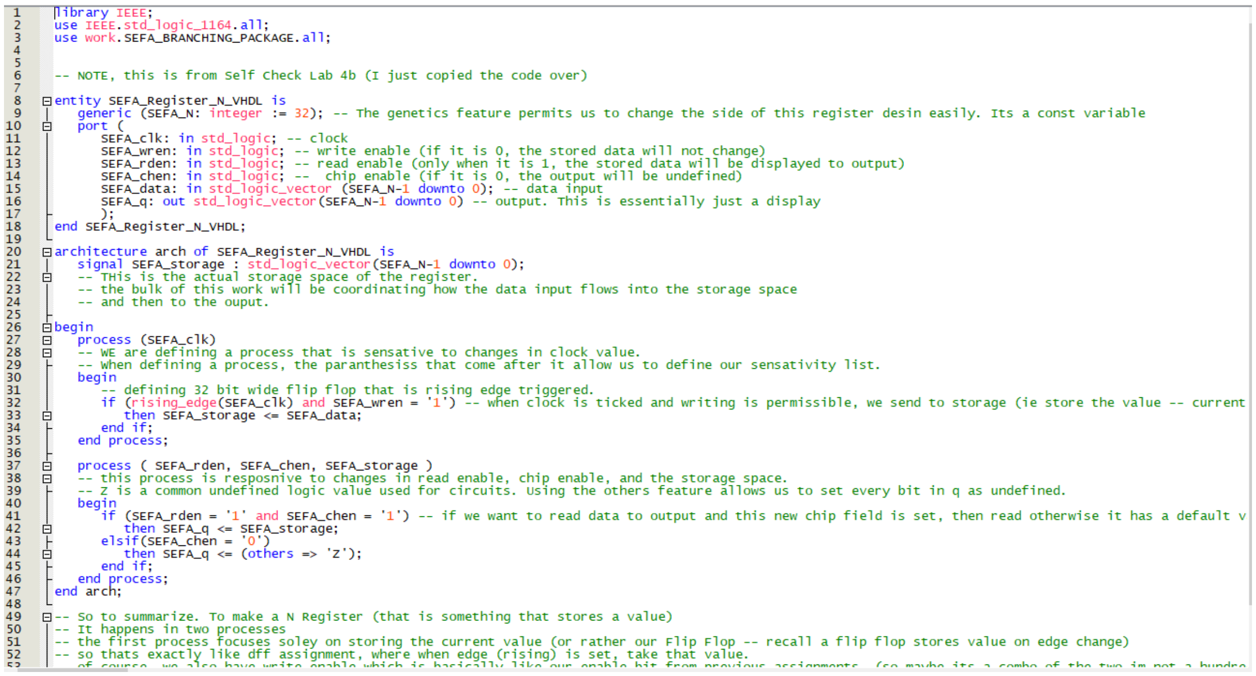
Branching Lab
April 18 2021

CSC 342/343 Spring 2021
Bujar Sefa

Objective

- To learn the importance of comparator lab
- To learn the importance of Register File and Registers
- To Design and implement in VHDL MIPS instruction BEQ, BNE, J
- To learn about I Type format instructions.

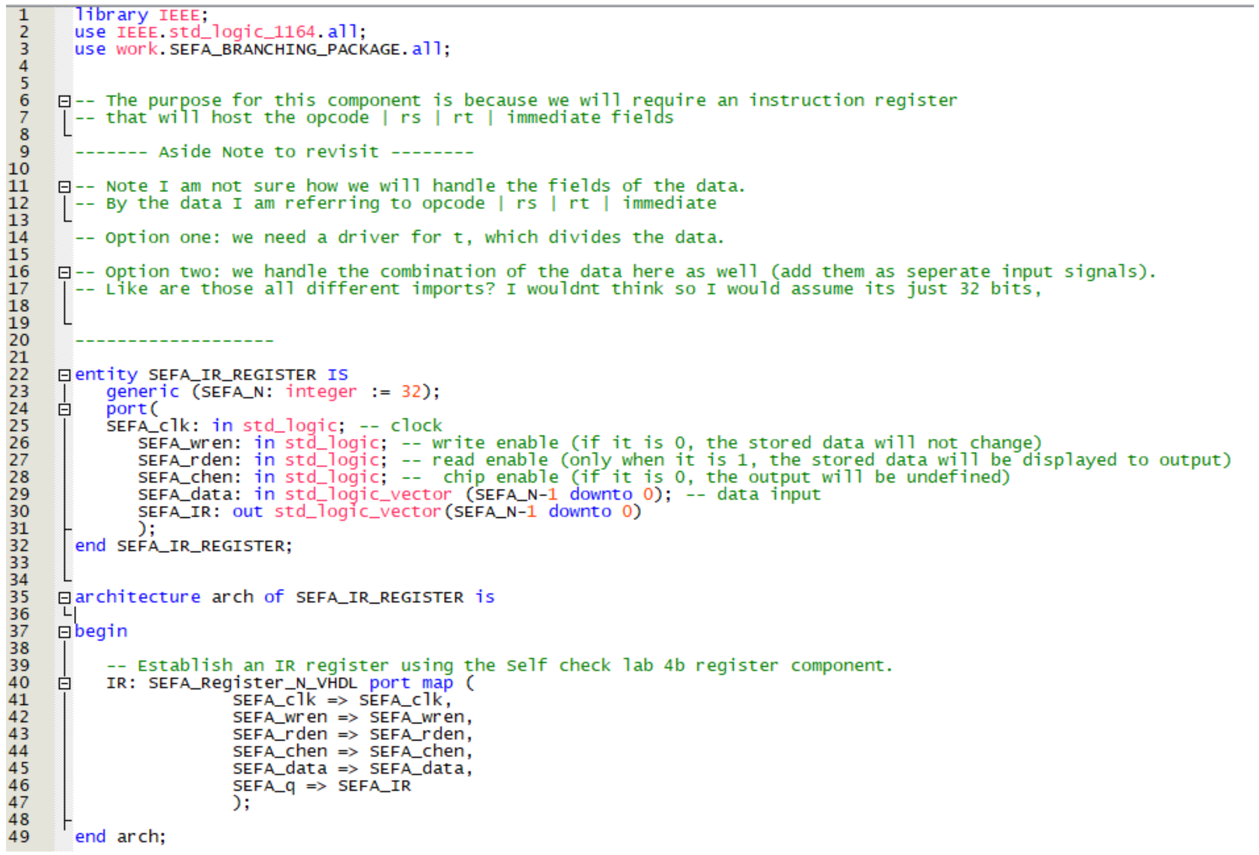
SEFA_REGISTER_N



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


-- NOTE, this is from Self Check Lab 4b (I just copied the code over)

entity SEFA_Register_N_VHDL is
   generic (SEFA_N: integer := 32); -- The genetics feature permits us to change the side of this register desin easily. Its a const variable
   port (
      SEFA_clk: in std_logic; -- clock
      SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
      SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
      SEFA_chen: in std_logic; --  chip enable (if it is 0, the output will be undefined)
      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
      SEFA_q: out std_logic_vector(SEFA_N-1 downto 0) -- output. This is essentially just a display
      );
end SEFA_Register_N_VHDL;

architecture arch of SEFA_Register_N_VHDL is
   signal SEFA_storage : std_logic_vector(SEFA_N-1 downto 0);
   -- THis is the actual storage space of the register.
   -- the bulk of this work will be coordinating how the data input flows into the storage space
   -- and then to the ouput.

begin
   process (SEFA_clk)
   -- WE are defining a process that is sensative to changes in clock value.
   -- When defining a process, the paranthesiss that come after it allow us to define our sensativity list.
   begin
      -- defining 32 bit wide flip flop that is rising edge triggered.
      if (rising_edge(SEFA_clk) and SEFA_wren = '1') -- when clock is ticked and writing is permissible, we send to storage (ie store the value -- current
         then SEFA_storage <= SEFA_data;
      end if;
   end process;

   process ( SEFA_rden, SEFA_chen, SEFA_storage )
   -- this process is resposnive to changes in read enable, chip enable, and the storage space.
   -- Z is a common undefined logic value used for circuits. Using the others feature allows us to set every bit in q as undefined.
   begin
      if (SEFA_rden = '1' and SEFA_chen = '1') -- if we want to read data to output and this new chip field is set, then read otherwise it has a default v
         then SEFA_q <= SEFA_storage;
      elsif(SEFA_chen = '0')
         then SEFA_q <= (others => 'Z');
      end if;
   end process;
end arch;

-- So to summarize. To make a N Register (that is something that stores a value)
-- It happens in two processes
-- the first process focuses soley on storing the current value (or rather our Flip Flop -- recall a flip flop stores value on edge change)
-- so thats exactly like dff assignment, where when edge (rising) is set, take that value.
-- of course, we also have write enable which is basically like our enable bit from previous assignments. (so maybe its a combo of the two im not a hundre
```

The purpose of this file is to create an n bit register using clock and event. We can read and write to this register, depending on the selected signals. This component is used to create the RS, RT, IMM, and PC registers.

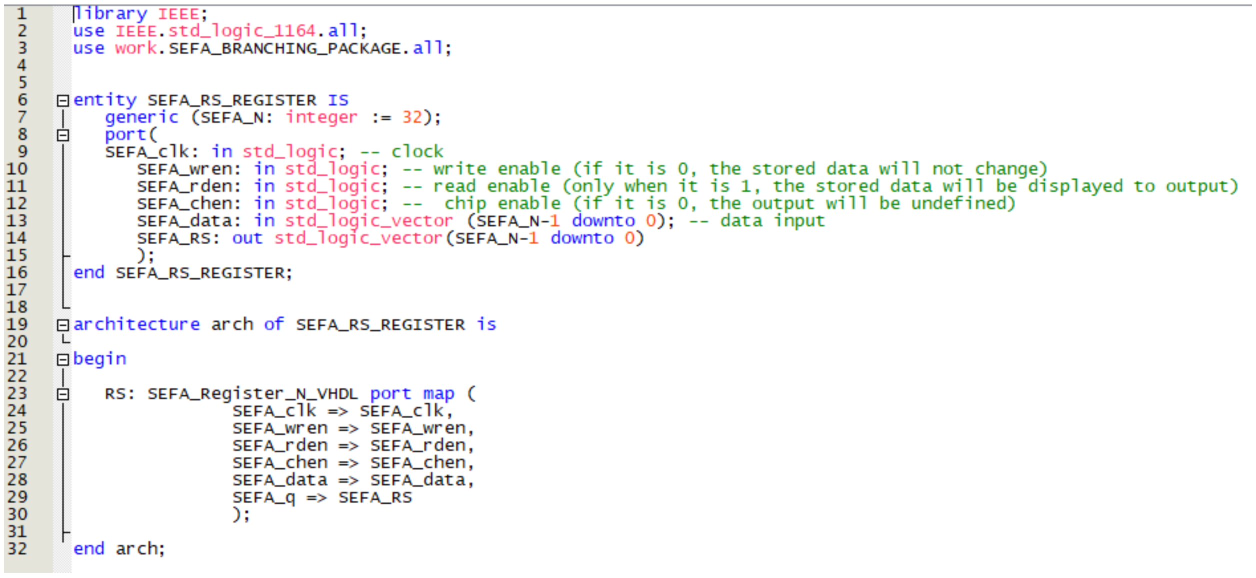
SEFA_IR_REGISTER



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


-- The purpose for this component is because we will require an instruction register
-- that will host the opcode | rs | rt | immediate fields

------- Aside Note to revisit --------

-- Note I am not sure how we will handle the fields of the data.
-- By the data I am referring to opcode | rs | rt | immediate

-- Option one: we need a driver for t, which divides the data.

-- Option two: we handle the combination of the data here as well (add them as seperate input signals).
-- Like are those all different imports? I wouldnt think so I would assume its just 32 bits,


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

entity SEFA_IR_REGISTER IS
    generic (SEFA_N: integer := 32);
    port(
    SEFA_clk: in std_logic; -- clock
        SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
        SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
        SEFA_chen: in std_logic; --  chip enable (if it is 0, the output will be undefined)
        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
        SEFA_IR: out std_logic_vector(SEFA_N-1 downto 0)
        );
end SEFA_IR_REGISTER;


architecture arch of SEFA_IR_REGISTER is

begin

    -- Establish an IR register using the Self check lab 4b register component.
    IR: SEFA_Register_N_VHDL port map (
                    SEFA_clk => SEFA_clk,
                    SEFA_wren => SEFA_wren,
                    SEFA_rden => SEFA_rden,
                    SEFA_chen => SEFA_chen,
                    SEFA_data => SEFA_data,
                    SEFA_q => SEFA_IR
                    );

end arch;
```

This is the register that will hold the instruction, that is with opcode for BEQ, BNE, J and its associated components, such as RS, RT, and or IMM. This component utilizes SEFA_REGISTER_N.

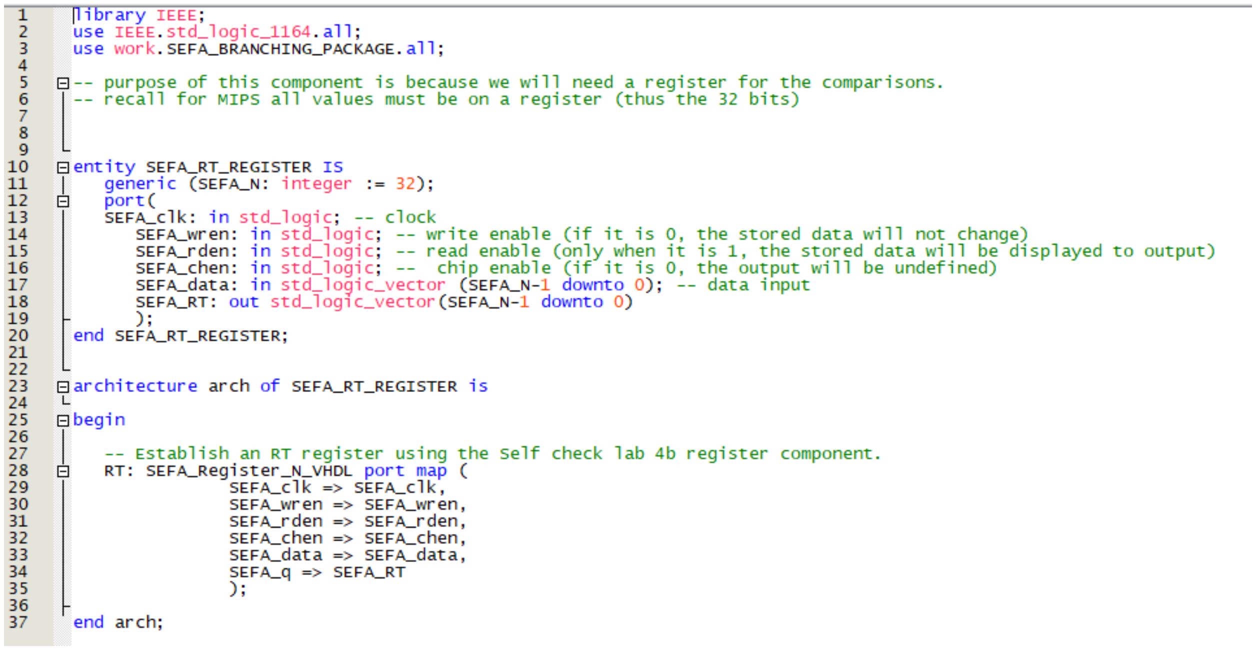
SEFA_RS_REGISTER



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


entity SEFA_RS_REGISTER IS
    generic (SEFA_N: integer := 32);
    port(
    SEFA_clk: in std_logic; -- clock
        SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
        SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
        SEFA_chen: in std_logic; --  chip enable (if it is 0, the output will be undefined)
        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
        SEFA_RS: out std_logic_vector(SEFA_N-1 downto 0)
        );
end SEFA_RS_REGISTER;


architecture arch of SEFA_RS_REGISTER is

begin

    RS: SEFA_Register_N_VHDL port map (
                SEFA_clk => SEFA_clk,
                SEFA_wren => SEFA_wren,
                SEFA_rden => SEFA_rden,
                SEFA_chen => SEFA_chen,
                SEFA_data => SEFA_data,
                SEFA_q => SEFA_RS
                );

end arch;
```

This is the register that holds the RS value of I Type format. This is for BEQ and BNE MIPS instructions. This component utilizes SEFA_REGISTER_N.

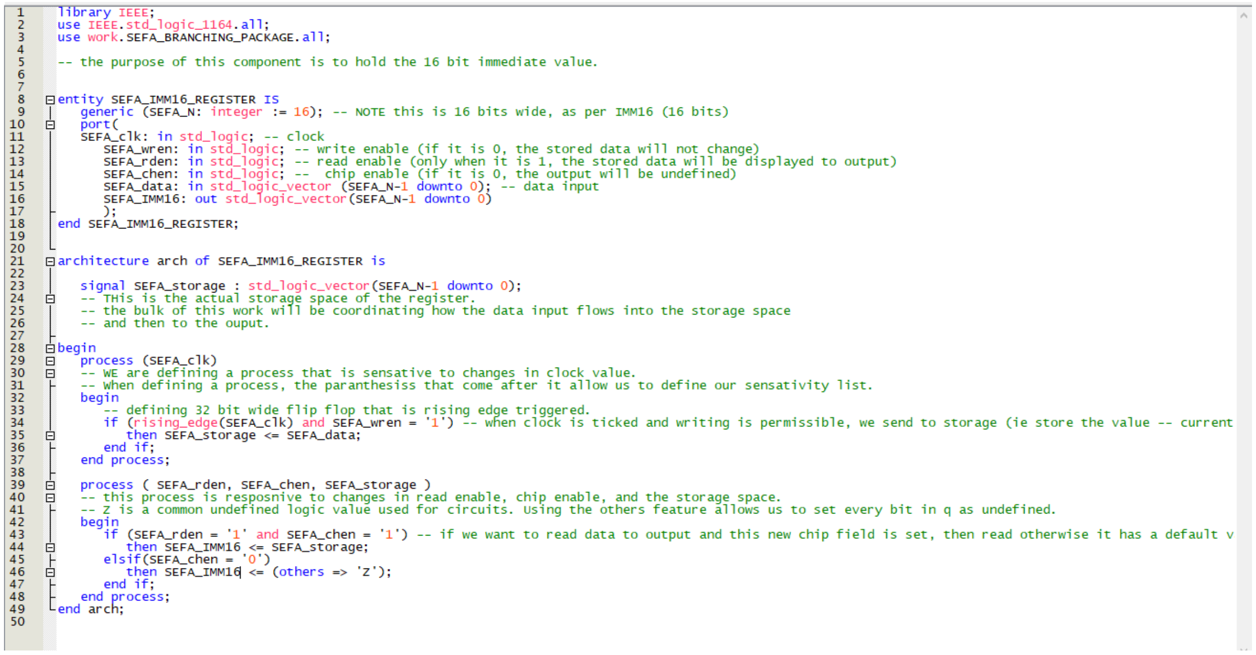
SEFA_RT_REGISTER



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;

-- purpose of this component is because we will need a register for the comparisons.
-- recall for MIPS all values must be on a register (thus the 32 bits)



entity SEFA_RT_REGISTER IS
    generic (SEFA_N: integer := 32);
    port(
    SEFA_clk: in std_logic; -- clock
        SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
        SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
        SEFA_chen: in std_logic; --  chip enable (if it is 0, the output will be undefined)
        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
        SEFA_RT: out std_logic_vector(SEFA_N-1 downto 0)
        );
end SEFA_RT_REGISTER;


architecture arch of SEFA_RT_REGISTER is

begin

    -- Establish an RT register using the Self check lab 4b register component.
    RT: SEFA_Register_N_VHDL port map (
                SEFA_clk => SEFA_clk,
                SEFA_wren => SEFA_wren,
                SEFA_rden => SEFA_rden,
                SEFA_chen => SEFA_chen,
                SEFA_data => SEFA_data,
                SEFA_q => SEFA_RT
                );

end arch;
```

This is the register that holds the RT value of I Type format. This is for BEQ and BNE MIPS instructions. This component utilizes SEFA_REGISTER_N.

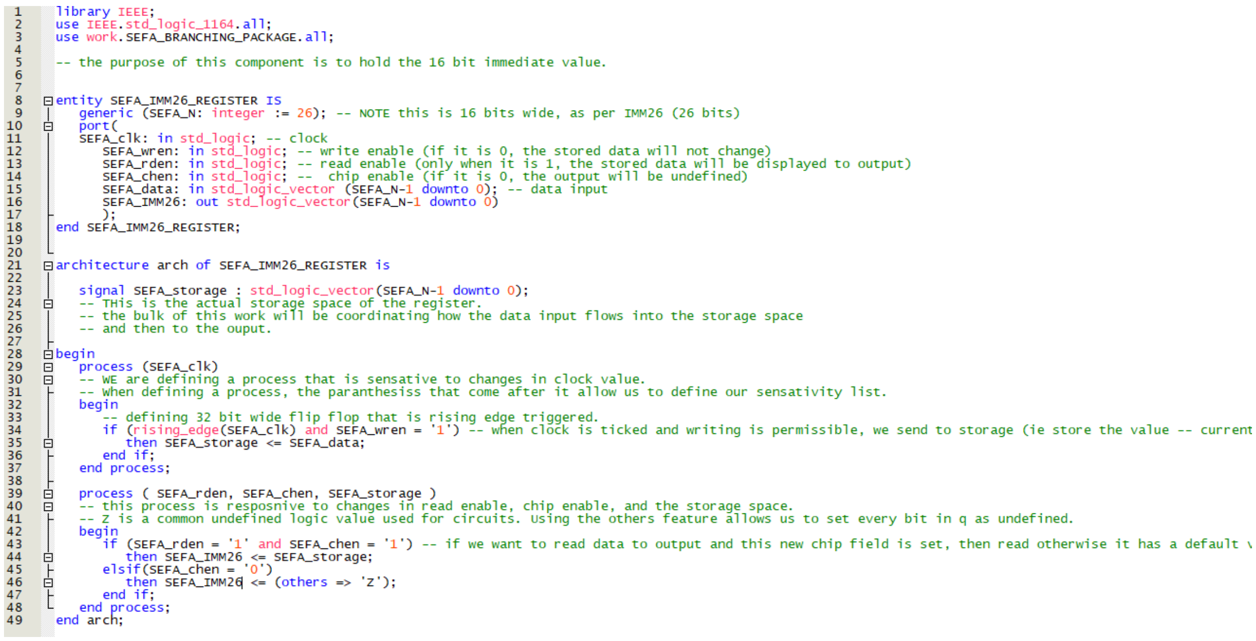
SEFA_IMM16_REGISTER



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;

-- the purpose of this component is to hold the 16 bit immediate value.


entity SEFA_IMM16_REGISTER IS
   generic (SEFA_N: integer := 16); -- NOTE this is 16 bits wide, as per IMM16 (16 bits)
   port(
   SEFA_clk: in std_logic; -- clock
      SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
      SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
      SEFA_chen: in std_logic; --  chip enable (if it is 0, the output will be undefined)
      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
      SEFA_IMM16: out std_logic_vector(SEFA_N-1 downto 0)
      );
end SEFA_IMM16_REGISTER;


architecture arch of SEFA_IMM16_REGISTER is

   signal SEFA_storage : std_logic_vector(SEFA_N-1 downto 0);
   -- THis is the actual storage space of the register.
   -- the bulk of this work will be coordinating how the data input flows into the storage space
   -- and then to the ouput.

begin
   process (SEFA_clk)
   -- WE are defining a process that is sensative to changes in clock value.
   -- when defining a process, the paranthesiss that come after it allow us to define our sensativity list.
   begin
      -- defining 32 bit wide flip flop that is rising edge triggered.
      if (rising_edge(SEFA_clk) and SEFA_wren = '1') -- when clock is ticked and writing is permissible, we send to storage (ie store the value -- current
         then SEFA_storage <= SEFA_data;
      end if;
   end process;

   process ( SEFA_rden, SEFA_chen, SEFA_storage )
   -- this process is resposnive to changes in read enable, chip enable, and the storage space.
   -- Z is a common undefined logic value used for circuits. Using the others feature allows us to set every bit in q as undefined.
   begin
      if (SEFA_rden = '1' and SEFA_chen = '1') -- if we want to read data to output and this new chip field is set, then read otherwise it has a default v
         then SEFA_IMM16 <= SEFA_storage;
      elsif(SEFA_chen = '0')
         then SEFA_IMM16 <= (others => 'Z');
      end if;
   end process;
end arch;
```

This is the register that holds the 16 bit immediate value of I Type format. This is for BEQ and BNE MIPS instructions. This component utilizes SEFA_REGISTER_N.

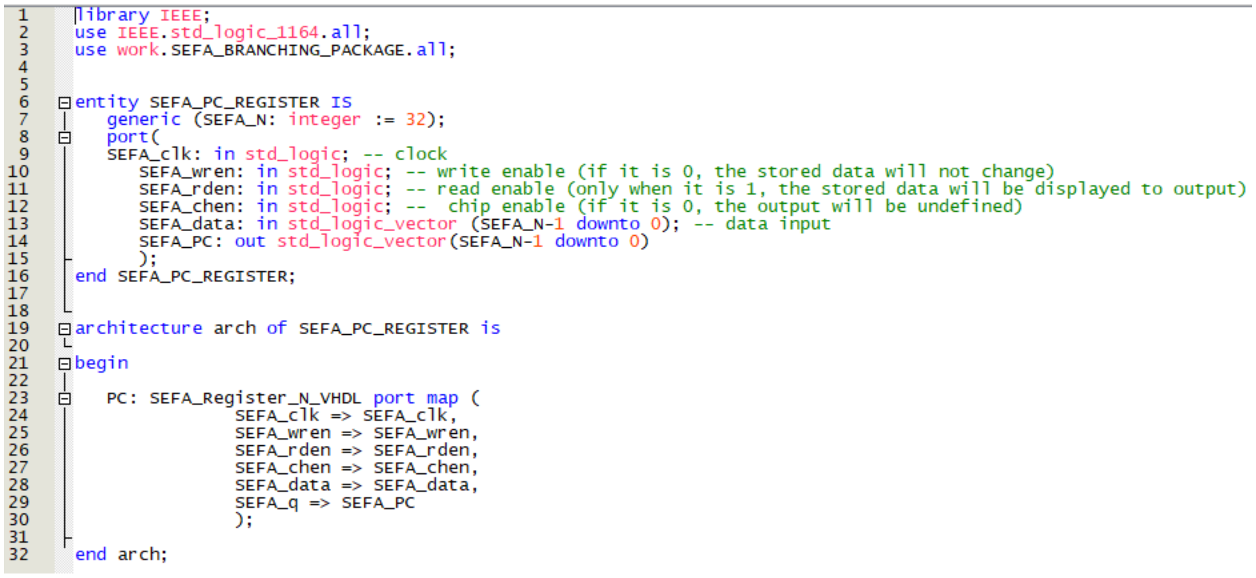
SEFA_IMM26_REGISTER



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;

-- the purpose of this component is to hold the 16 bit immediate value.


entity SEFA_IMM26_REGISTER IS
   generic (SEFA_N: integer := 26); -- NOTE this is 16 bits wide, as per IMM26 (26 bits)
   port(
   SEFA_clk: in std_logic; -- clock
      SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
      SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
      SEFA_chen: in std_logic; --  chip enable (if it is 0, the output will be undefined)
      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
      SEFA_IMM26: out std_logic_vector(SEFA_N-1 downto 0)
      );
end SEFA_IMM26_REGISTER;


architecture arch of SEFA_IMM26_REGISTER is

   signal SEFA_storage : std_logic_vector(SEFA_N-1 downto 0);
   -- THis is the actual storage space of the register.
   -- the bulk of this work will be coordinating how the data input flows into the storage space
   -- and then to the ouput.

begin
   process (SEFA_clk)
   -- WE are defining a process that is sensative to changes in clock value.
   -- when defining a process, the paranthesiss that come after it allow us to define our sensativity list.
   begin
      -- defining 32 bit wide flip flop that is rising edge triggered.
      if (rising_edge(SEFA_clk) and SEFA_wren = '1') -- when clock is ticked and writing is permissible, we send to storage (ie store the value -- current
         then SEFA_storage <= SEFA_data;
      end if;
   end process;

   process ( SEFA_rden, SEFA_chen, SEFA_storage )
   -- this process is resposnive to changes in read enable, chip enable, and the storage space.
   -- Z is a common undefined logic value used for circuits. Using the others feature allows us to set every bit in q as undefined.
   begin
      if (SEFA_rden = '1' and SEFA_chen = '1') -- if we want to read data to output and this new chip field is set, then read otherwise it has a default
         then SEFA_IMM26 <= SEFA_storage;
      elsif(SEFA_chen = '0')
         then SEFA_IMM26 <= (others => 'Z');
      end if;
   end process;
end arch;
```

This is the register that holds the 16 bit immediate value. This is for J (Jump) MIPS instructions. This component utilizes SEFA_REGISTER_N.

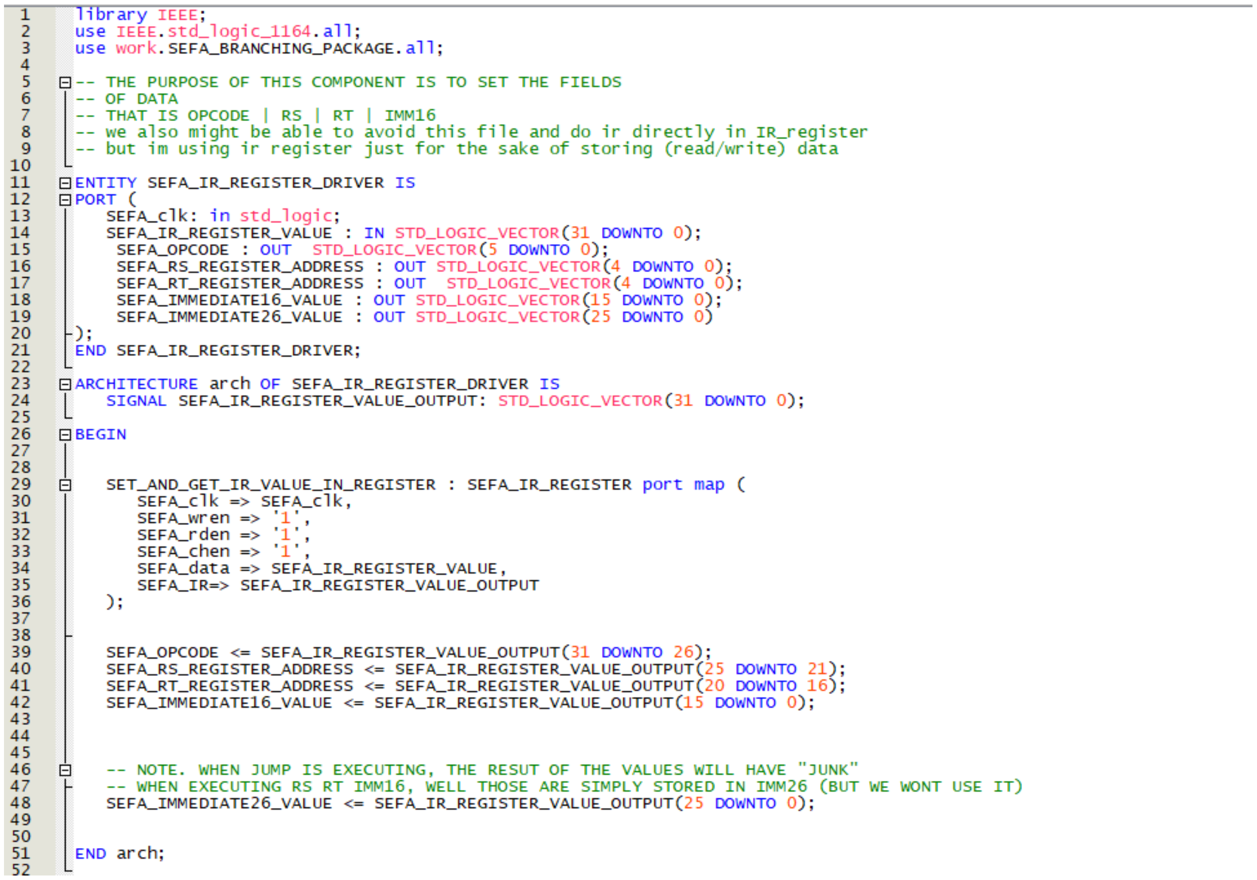
SEFA_PC_REGISTER



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


entity SEFA_PC_REGISTER IS
   generic (SEFA_N: integer := 32);
   port(
   SEFA_clk: in std_logic; -- clock
      SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
      SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
      SEFA_chen: in std_logic; --  chip enable (if it is 0, the output will be undefined)
      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
      SEFA_PC: out std_logic_vector(SEFA_N-1 downto 0)
      );
end SEFA_PC_REGISTER;


architecture arch of SEFA_PC_REGISTER is

begin

   PC: SEFA_Register_N_VHDL port map (
               SEFA_clk => SEFA_clk,
               SEFA_wren => SEFA_wren,
               SEFA_rden => SEFA_rden,
               SEFA_chen => SEFA_chen,
               SEFA_data => SEFA_data,
               SEFA_q => SEFA_PC
               );

end arch;
```

This is the register that holds the address of the next instruction to execute, We pre-initialize this value. For this lab, we are focused on obtaining the PC after the execution of BNE, BEQ, J and this we must use this value to verify the new (which would be updated in the future). This component utilizes SEFA_REGISTER_N.

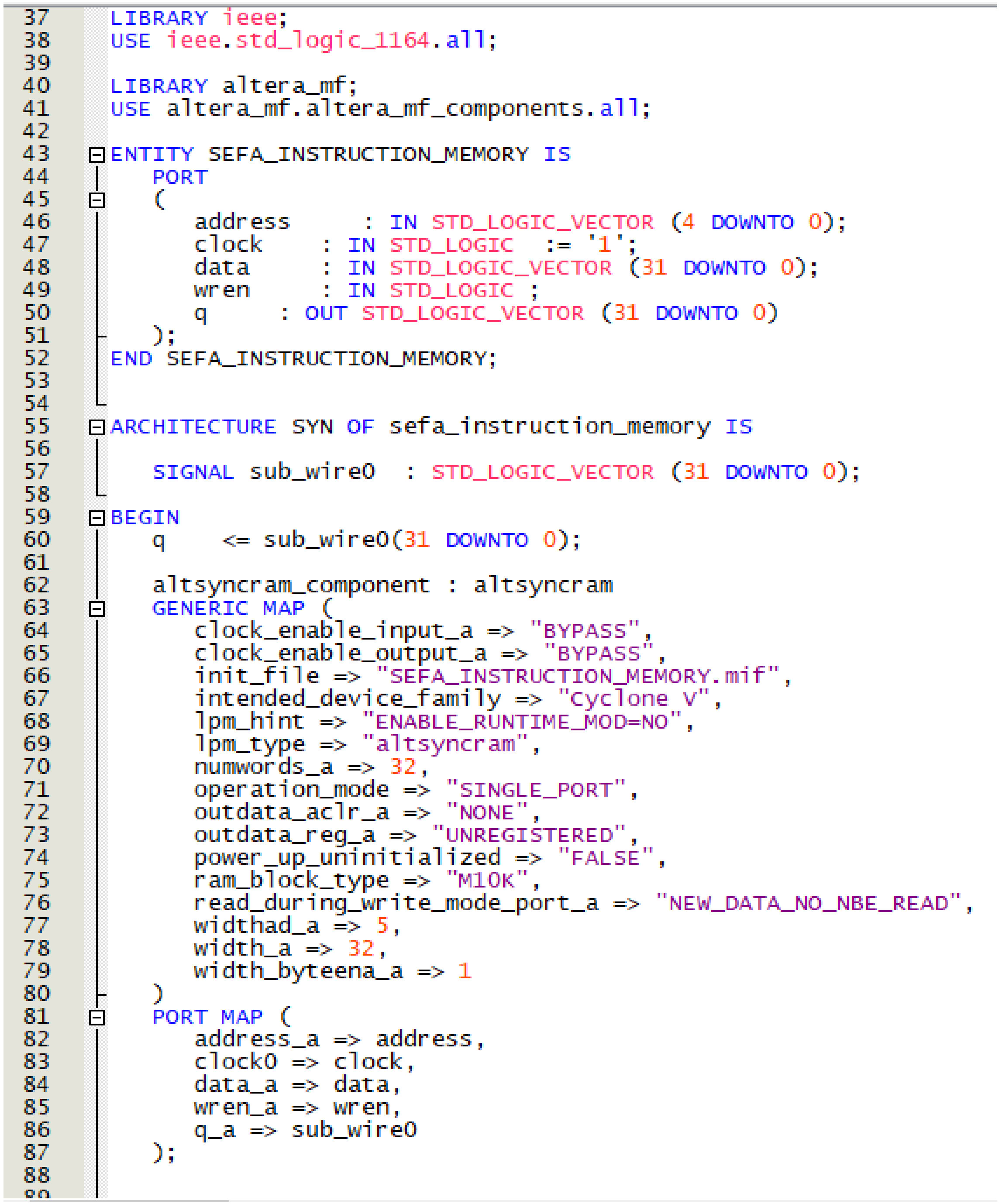
SEFA_IR_REGISTER_DRIVER



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;

-- THE PURPOSE OF THIS COMPONENT IS TO SET THE FIELDS
-- OF DATA
-- THAT IS OPCODE | RS | RT | IMM16
-- we also might be able to avoid this file and do ir directly in IR_register
-- but im using ir register just for the sake of storing (read/write) data

ENTITY SEFA_IR_REGISTER_DRIVER IS
PORT (
    SEFA_clk: in std_logic;
    SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
     SEFA_OPCODE : OUT  STD_LOGIC_VECTOR(5 DOWNTO 0);
     SEFA_RS_REGISTER_ADDRESS : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
     SEFA_RT_REGISTER_ADDRESS : OUT  STD_LOGIC_VECTOR(4 DOWNTO 0);
     SEFA_IMMEDIATE16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
     SEFA_IMMEDIATE26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNTO 0)
);
END SEFA_IR_REGISTER_DRIVER;

ARCHITECTURE arch OF SEFA_IR_REGISTER_DRIVER IS
    SIGNAL SEFA_IR_REGISTER_VALUE_OUTPUT: STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN


    SET_AND_GET_IR_VALUE_IN_REGISTER : SEFA_IR_REGISTER port map (
       SEFA_clk => SEFA_clk,
       SEFA_wren => '1',
       SEFA_rden => '1',
       SEFA_chen => '1',
       SEFA_data => SEFA_IR_REGISTER_VALUE,
       SEFA_IR=> SEFA_IR_REGISTER_VALUE_OUTPUT
    );


    SEFA_OPCODE <= SEFA_IR_REGISTER_VALUE_OUTPUT(31 DOWNTO 26);
    SEFA_RS_REGISTER_ADDRESS <= SEFA_IR_REGISTER_VALUE_OUTPUT(25 DOWNTO 21);
    SEFA_RT_REGISTER_ADDRESS <= SEFA_IR_REGISTER_VALUE_OUTPUT(20 DOWNTO 16);
    SEFA_IMMEDIATE16_VALUE <= SEFA_IR_REGISTER_VALUE_OUTPUT(15 DOWNTO 0);



    -- NOTE. WHEN JUMP IS EXECUTING, THE RESUT OF THE VALUES WILL HAVE "JUNK"
    -- WHEN EXECUTING RS RT IMM16, WELL THOSE ARE SIMPLY STORED IN IMM26 (BUT WE WONT USE IT)
    SEFA_IMMEDIATE26_VALUE <= SEFA_IR_REGISTER_VALUE_OUTPUT(25 DOWNTO 0);


END arch;

```

This file is known as the IR driverIt takes in the Instruction Register value and pulls out the different components such as OPCODE, RS address, RT address, and IMM value(s).

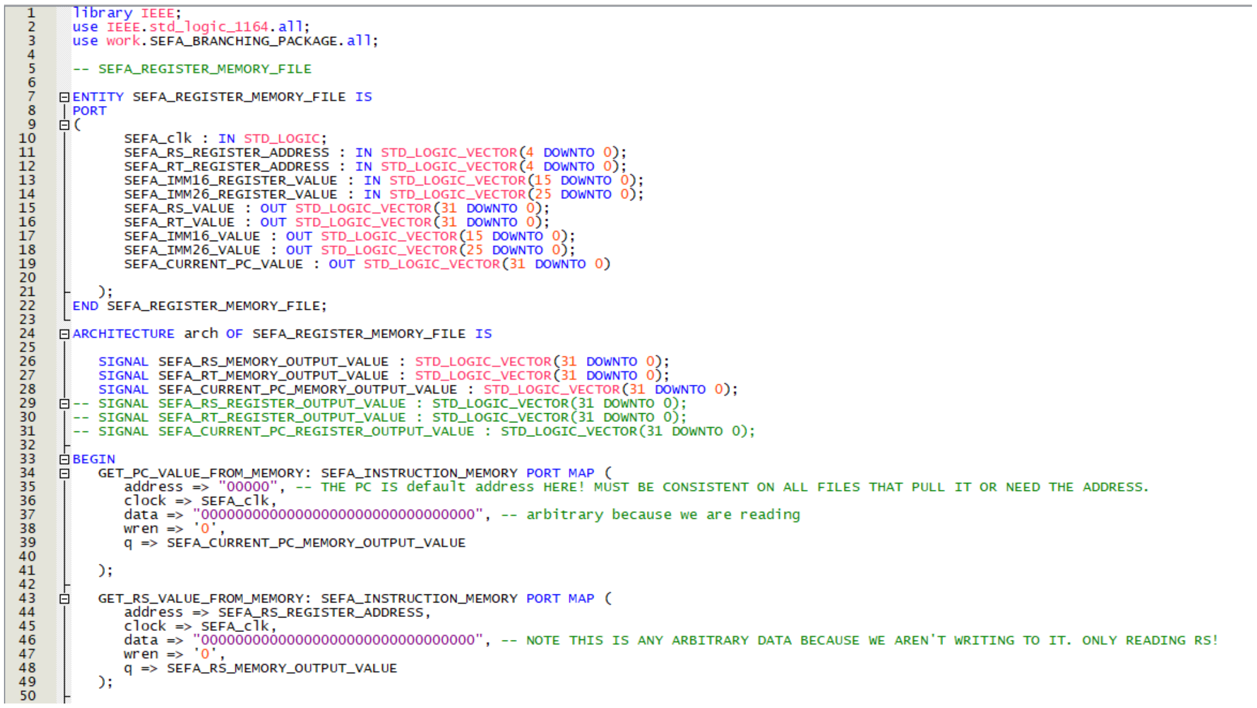
SEFA_INSTRUCTION_MEMORY



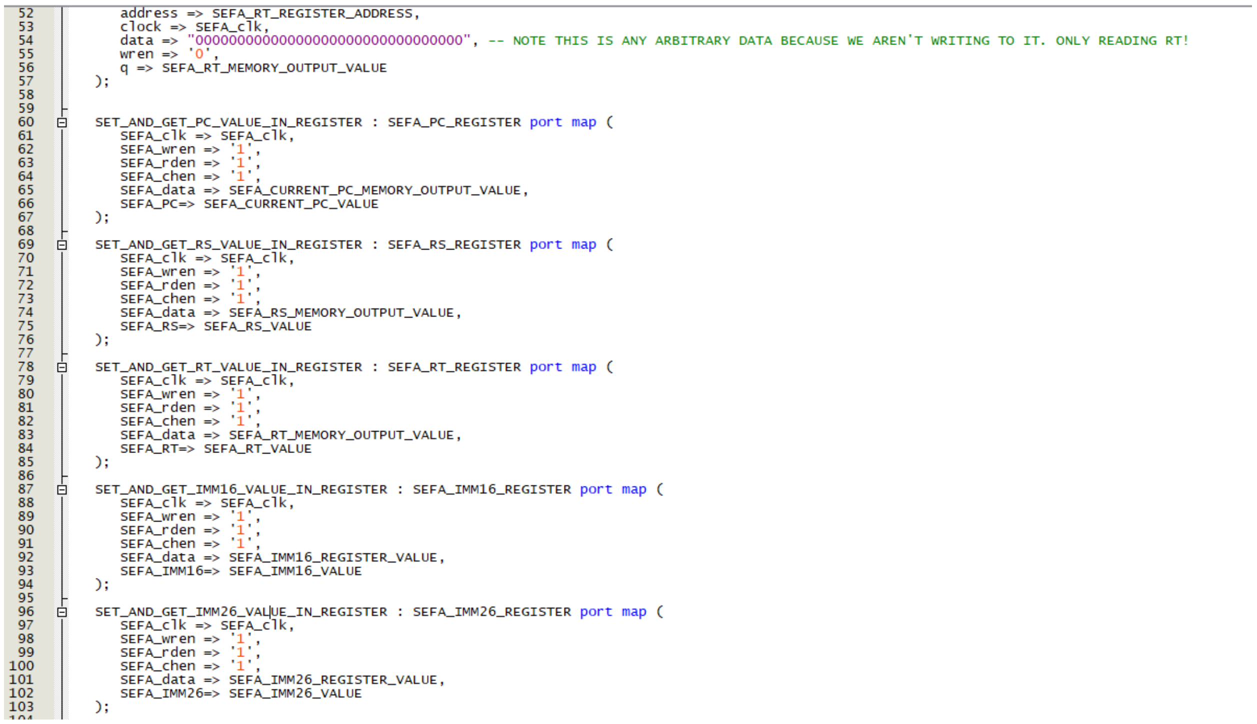
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY SEFA_INSTRUCTION_MEMORY IS
    PORT
    (
        address     : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        clock     : IN STD_LOGIC   := '1';
        data      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        wren      : IN STD_LOGIC ;
        q      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END SEFA_INSTRUCTION_MEMORY;


ARCHITECTURE SYN OF sefa_instruction_memory IS

    SIGNAL sub_wire0  : STD_LOGIC_VECTOR (31 DOWNTO 0);

BEGIN
    q    <= sub_wire0(31 DOWNTO 0);

    altsyncram_component : altsyncram
    GENERIC MAP (
        clock_enable_input_a => "BYPASS",
        clock_enable_output_a => "BYPASS",
        init_file => "SEFA_INSTRUCTION_MEMORY.mif",
        intended_device_family => "Cyclone V",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 32,
        operation_mode => "SINGLE_PORT",
        outdata_aclr_a => "NONE",
        outdata_reg_a => "UNREGISTERED",
        power_up_uninitialized => "FALSE",
        ram_block_type => "M10K",
        read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
        widthad_a => 5,
        width_a => 32,
        width_byteena_a => 1
    )
    PORT MAP (
        address_a => address,
        clock0 => clock,
        data_a => data,
        wren_a => wren,
        q_a => sub_wire0
    );

```

This is the RAM file which we use to access the values of RS, RT, and PC from. It was made for the convenience of updating and pulling values. Our registers, from the register file, will use this to get and load the value into registers. This component was built using the Midterm Lab.

SEFA_REGISTER_MEMORY_FILE



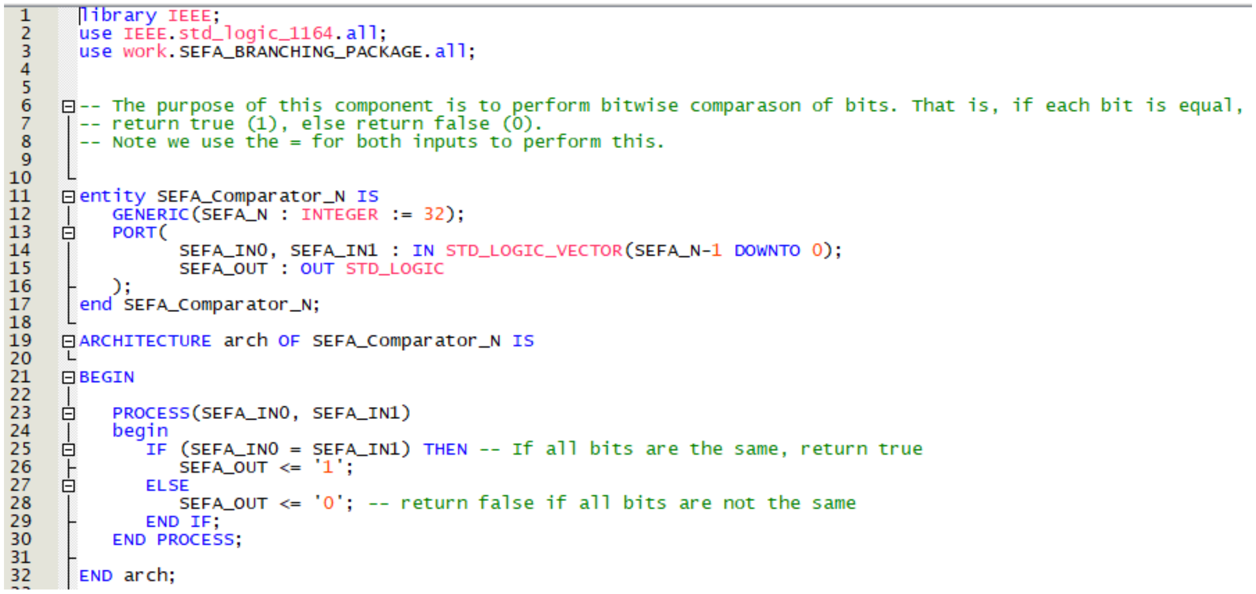
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;

-- SEFA_REGISTER_MEMORY_FILE

ENTITY SEFA_REGISTER_MEMORY_FILE IS
PORT
(
      SEFA_clk : IN STD_LOGIC;
      SEFA_RS_REGISTER_ADDRESS : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
      SEFA_RT_REGISTER_ADDRESS : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
      SEFA_IMM16_REGISTER_VALUE : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      SEFA_IMM26_REGISTER_VALUE : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
      SEFA_RS_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
      SEFA_RT_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
      SEFA_IMM16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
      SEFA_IMM26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNTO 0);
      SEFA_CURRENT_PC_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)

   );
END SEFA_REGISTER_MEMORY_FILE;

ARCHITECTURE arch OF SEFA_REGISTER_MEMORY_FILE IS

   SIGNAL SEFA_RS_MEMORY_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
   SIGNAL SEFA_RT_MEMORY_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
   SIGNAL SEFA_CURRENT_PC_MEMORY_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
-- SIGNAL SEFA_RS_REGISTER_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
-- SIGNAL SEFA_RT_REGISTER_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
-- SIGNAL SEFA_CURRENT_PC_REGISTER_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN
   GET_PC_VALUE_FROM_MEMORY: SEFA_INSTRUCTION_MEMORY PORT MAP (
      address => "00000", -- THE PC IS default address HERE! MUST BE CONSISTENT ON ALL FILES THAT PULL IT OR NEED THE ADDRESS.
      clock => SEFA_clk,
      data => "00000000000000000000000000000000", -- arbitrary because we are reading
      wren => '0',
      q => SEFA_CURRENT_PC_MEMORY_OUTPUT_VALUE

   );

   GET_RS_VALUE_FROM_MEMORY: SEFA_INSTRUCTION_MEMORY PORT MAP (
      address => SEFA_RS_REGISTER_ADDRESS,
      clock => SEFA_clk,
      data => "00000000000000000000000000000000", -- NOTE THIS IS ANY ARBITRARY DATA BECAUSE WE AREN'T WRITING TO IT. ONLY READING RS!
      wren => '0',
      q => SEFA_RS_MEMORY_OUTPUT_VALUE
   );

```



```vhdl
      address => SEFA_RT_REGISTER_ADDRESS,
      clock => SEFA_clk,
      data => "00000000000000000000000000000000", -- NOTE THIS IS ANY ARBITRARY DATA BECAUSE WE AREN'T WRITING TO IT. ONLY READING RT!
      wren => '0',
      q => SEFA_RT_MEMORY_OUTPUT_VALUE
   );


   SET_AND_GET_PC_VALUE_IN_REGISTER : SEFA_PC_REGISTER port map (
      SEFA_clk => SEFA_clk,
      SEFA_wren => '1',
      SEFA_rden => '1',
      SEFA_chen => '1',
      SEFA_data => SEFA_CURRENT_PC_MEMORY_OUTPUT_VALUE,
      SEFA_PC=> SEFA_CURRENT_PC_VALUE
   );

   SET_AND_GET_RS_VALUE_IN_REGISTER : SEFA_RS_REGISTER port map (
      SEFA_clk => SEFA_clk,
      SEFA_wren => '1',
      SEFA_rden => '1',
      SEFA_chen => '1',
      SEFA_data => SEFA_RS_MEMORY_OUTPUT_VALUE,
      SEFA_RS=> SEFA_RS_VALUE
   );

   SET_AND_GET_RT_VALUE_IN_REGISTER : SEFA_RT_REGISTER port map (
      SEFA_clk => SEFA_clk,
      SEFA_wren => '1',
      SEFA_rden => '1',
      SEFA_chen => '1',
      SEFA_data => SEFA_RT_MEMORY_OUTPUT_VALUE,
      SEFA_RT=> SEFA_RT_VALUE
   );

   SET_AND_GET_IMM16_VALUE_IN_REGISTER : SEFA_IMM16_REGISTER port map (
      SEFA_clk => SEFA_clk,
      SEFA_wren => '1',
      SEFA_rden => '1',
      SEFA_chen => '1',
      SEFA_data => SEFA_IMM16_REGISTER_VALUE,
      SEFA_IMM16=> SEFA_IMM16_VALUE
   );

   SET_AND_GET_IMM26_VALUE_IN_REGISTER : SEFA_IMM26_REGISTER port map (
      SEFA_clk => SEFA_clk,
      SEFA_wren => '1',
      SEFA_rden => '1',
      SEFA_chen => '1',
      SEFA_data => SEFA_IMM26_REGISTER_VALUE,
      SEFA_IMM26=> SEFA_IMM26_VALUE
   );
```

This file is what is known as the registers file component. It takes RS address, RT address, and it gets the values from RAM (utilizing the Midterm Lab). The values from RAM are then loaded onto the respective registers, which we then use for the rest of the lab. RAM is already defaulted to addresses for RS, RT, and PC which we will see in the testbench.

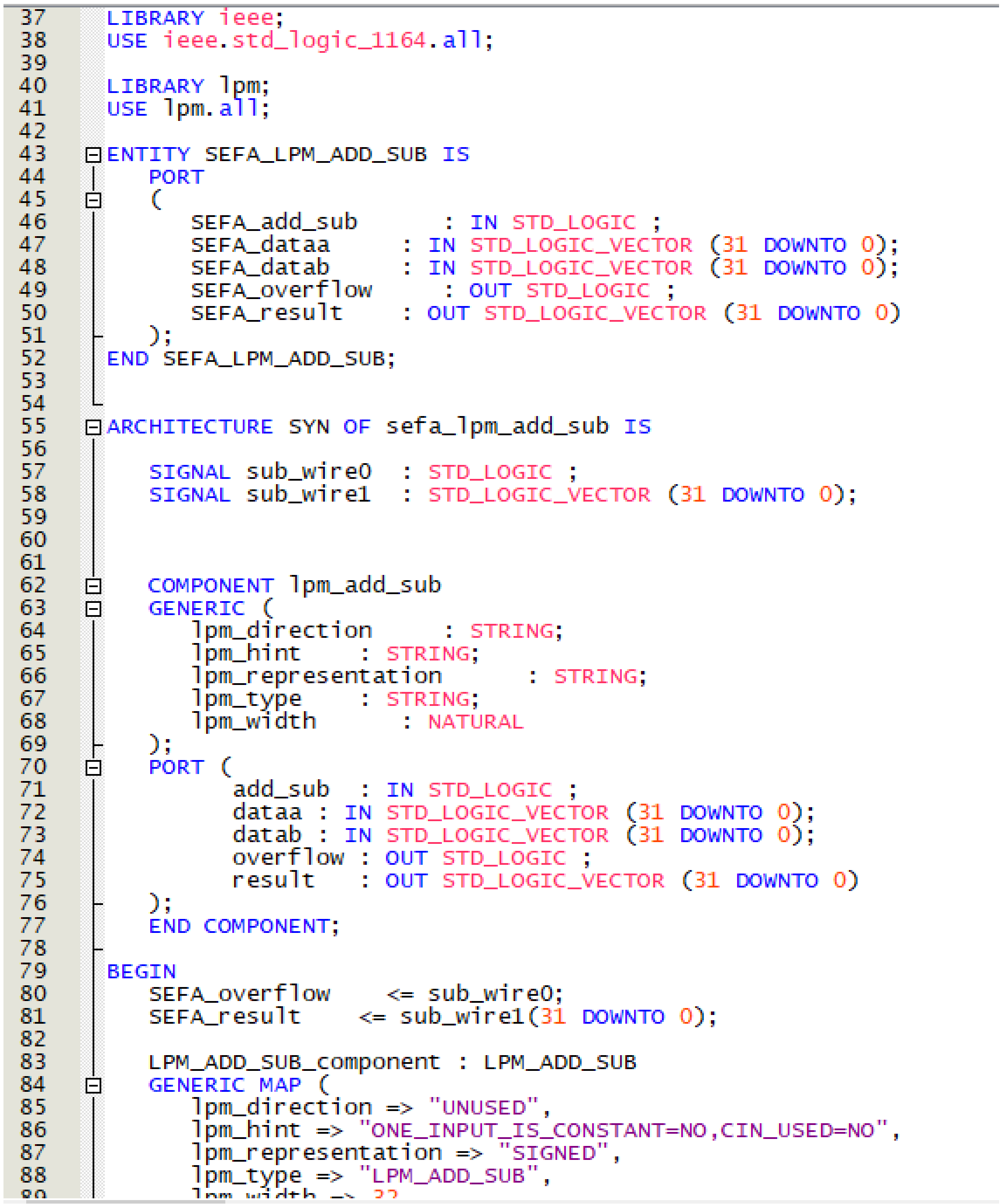
SEFA_Comparator_N



```
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


-- The purpose of this component is to perform bitwise comparason of bits. That is, if each bit is equal,
-- return true (1), else return false (0).
-- Note we use the = for both inputs to perform this.


entity SEFA_Comparator_N IS
    GENERIC(SEFA_N : INTEGER := 32);
    PORT(
            SEFA_IN0, SEFA_IN1 : IN STD_LOGIC_VECTOR(SEFA_N-1 DOWNTO 0);
            SEFA_OUT : OUT STD_LOGIC
    );
end SEFA_Comparator_N;

ARCHITECTURE arch OF SEFA_Comparator_N IS

BEGIN

    PROCESS(SEFA_IN0, SEFA_IN1)
    begin
        IF (SEFA_IN0 = SEFA_IN1) THEN -- If all bits are the same, return true
            SEFA_OUT <= '1';
        ELSE
            SEFA_OUT <= '0'; -- return false if all bits are not the same
        END IF;
    END PROCESS;

END arch;
```

This is a 32 bit comparator for the RS and RT values used in BEQ and BNE. It will evaluate if all the bits in RS = RT.

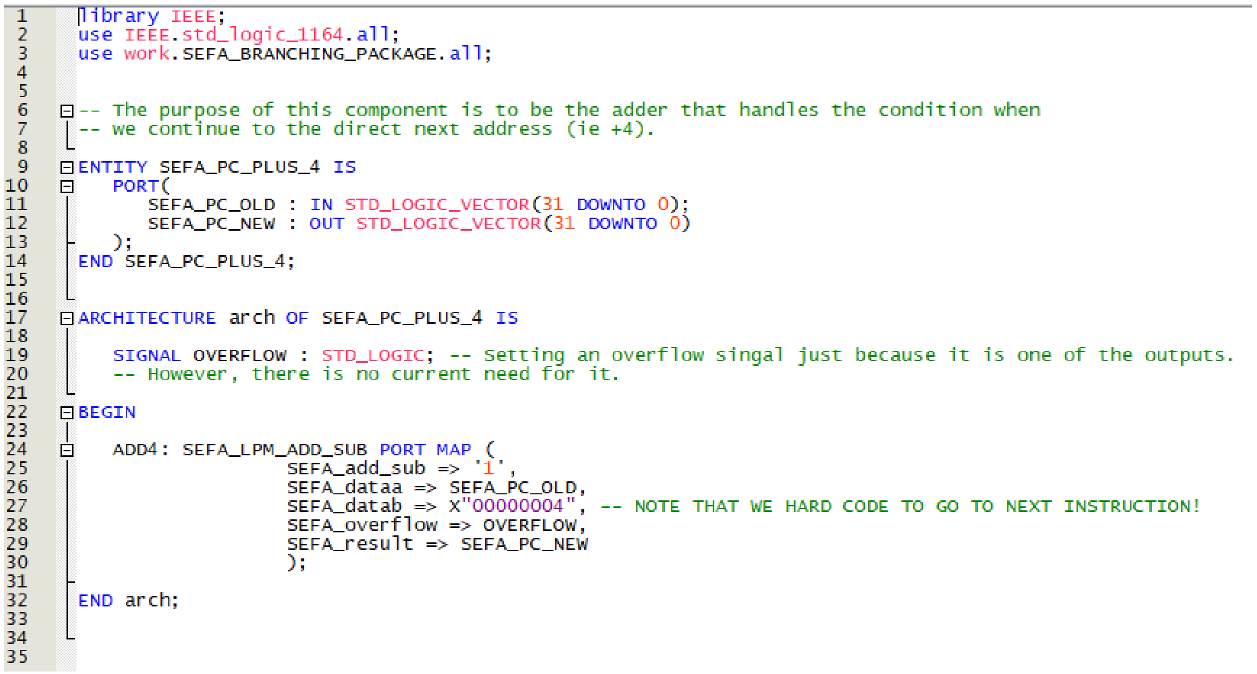
SEFA_LPM_ADD_SUB



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.all;

ENTITY SEFA_LPM_ADD_SUB IS
    PORT
    (
        SEFA_add_sub        : IN STD_LOGIC ;
        SEFA_dataa      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        SEFA_datab      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        SEFA_overflow       : OUT STD_LOGIC ;
        SEFA_result     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END SEFA_LPM_ADD_SUB;


ARCHITECTURE SYN OF sefa_lpm_add_sub IS

    SIGNAL sub_wire0  : STD_LOGIC ;
    SIGNAL sub_wire1  : STD_LOGIC_VECTOR (31 DOWNTO 0);



    COMPONENT lpm_add_sub
    GENERIC (
        lpm_direction      : STRING;
        lpm_hint     : STRING;
        lpm_representation      : STRING;
        lpm_type     : STRING;
        lpm_width       : NATURAL
    );
    PORT (
            add_sub   : IN STD_LOGIC ;
            dataa : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            datab : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            overflow : OUT STD_LOGIC ;
            result    : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
    END COMPONENT;

BEGIN
    SEFA_overflow    <= sub_wire0;
    SEFA_result    <= sub_wire1(31 DOWNTO 0);

    LPM_ADD_SUB_component : LPM_ADD_SUB
    GENERIC MAP (
        lpm_direction => "UNUSED",
        lpm_hint => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
        lpm_representation => "SIGNED",
        lpm_type => "LPM_ADD_SUB",
        lpm_width => 32
```

This is the LPM ADD SUB that we have used in previous labs. It is used to add PC+4 or PC+4+IMM

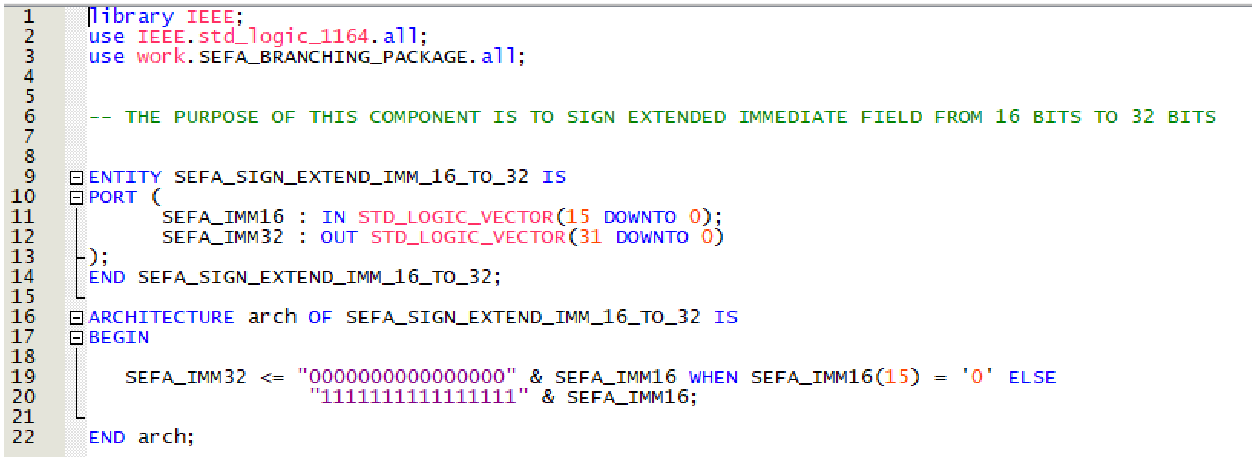
SEFA_PC_PLUS_4



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


-- The purpose of this component is to be the adder that handles the condition when
-- we continue to the direct next address (ie +4).

ENTITY SEFA_PC_PLUS_4 IS
   PORT(
      SEFA_PC_OLD : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
   );
END SEFA_PC_PLUS_4;


ARCHITECTURE arch OF SEFA_PC_PLUS_4 IS

   SIGNAL OVERFLOW : STD_LOGIC; -- Setting an overflow singal just because it is one of the outputs.
   -- However, there is no current need for it.

BEGIN

   ADD4: SEFA_LPM_ADD_SUB PORT MAP (
                  SEFA_add_sub => '1',
                  SEFA_dataa => SEFA_PC_OLD,
                  SEFA_datab => X"00000004", -- NOTE THAT WE HARD CODE TO GO TO NEXT INSTRUCTION!
                  SEFA_overflow => OVERFLOW,
                  SEFA_result => SEFA_PC_NEW
                  );

END arch;
```

This component adds four to the current PC value. It uses the LPM adder/subtractor.

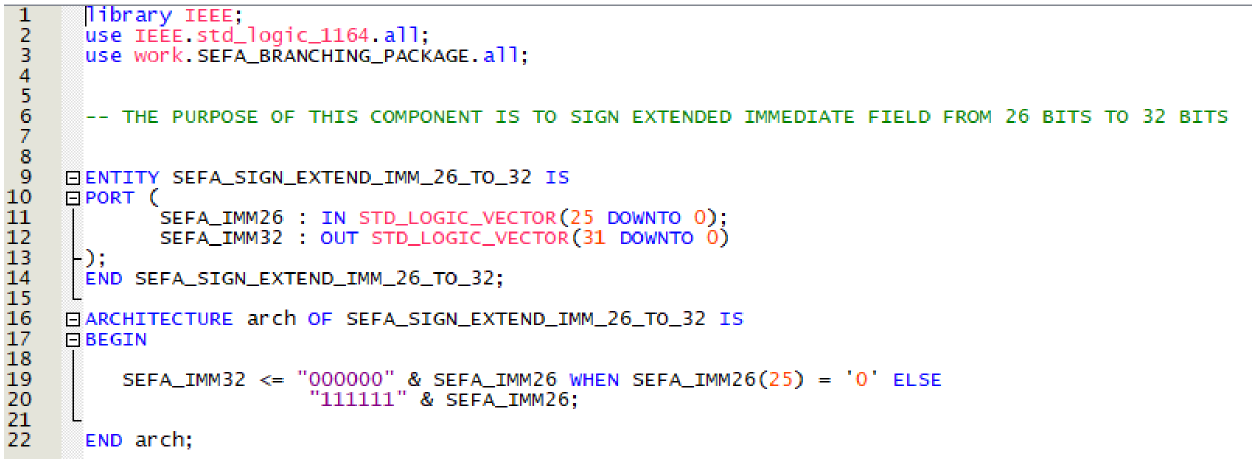
SEFA_SIGN_EXTEND_IMM_16_TO_32



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


-- THE PURPOSE OF THIS COMPONENT IS TO SIGN EXTENDED IMMEDIATE FIELD FROM 16 BITS TO 32 BITS


ENTITY SEFA_SIGN_EXTEND_IMM_16_TO_32 IS
PORT (
      SEFA_IMM16 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      SEFA_IMM32 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END SEFA_SIGN_EXTEND_IMM_16_TO_32;

ARCHITECTURE arch OF SEFA_SIGN_EXTEND_IMM_16_TO_32 IS
BEGIN

   SEFA_IMM32 <= "0000000000000000" & SEFA_IMM16 WHEN SEFA_IMM16(15) = '0' ELSE
                  "1111111111111111" & SEFA_IMM16;

END arch;
```

This component sign extends the 16 bit immediate to 32 bit immediate depending on the MSB of the 16 bit. If MSB = 1, concat 16 1's the front. If MSB = 0, concat 16 0's to the front.

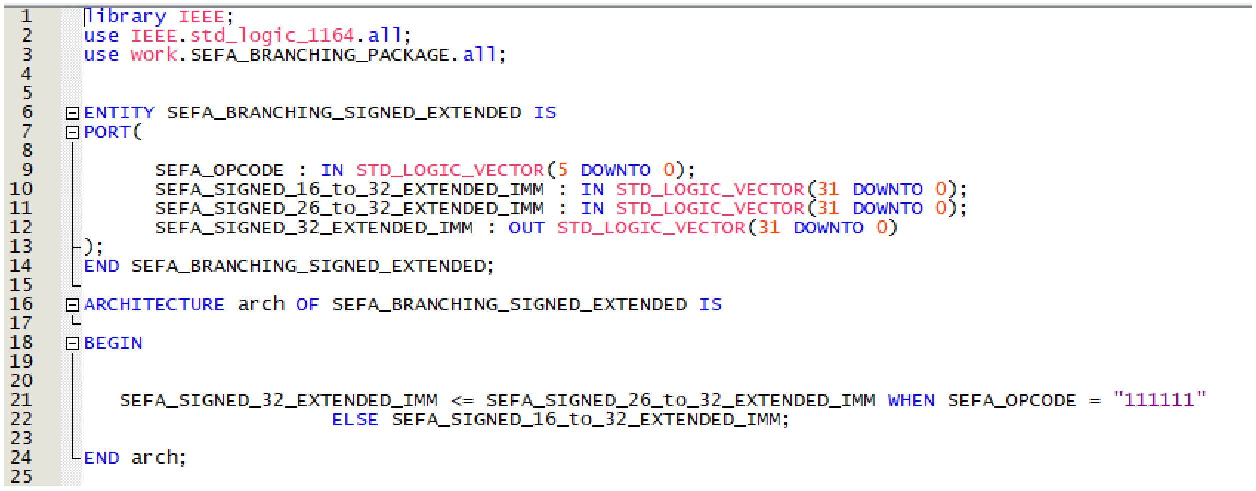
SEFA_SIGN_EXTEND_IMM_26_TO_32



```
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


-- THE PURPOSE OF THIS COMPONENT IS TO SIGN EXTENDED IMMEDIATE FIELD FROM 26 BITS TO 32 BITS


ENTITY SEFA_SIGN_EXTEND_IMM_26_TO_32 IS
PORT (
          SEFA_IMM26 : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
          SEFA_IMM32 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END SEFA_SIGN_EXTEND_IMM_26_TO_32;

ARCHITECTURE arch OF SEFA_SIGN_EXTEND_IMM_26_TO_32 IS
BEGIN

     SEFA_IMM32 <= "000000" & SEFA_IMM26 WHEN SEFA_IMM26(25) = '0' ELSE
                    "111111" & SEFA_IMM26;

END arch;
```

This component sign extends the 26 bit immediate to 32 bit immediate depending on the MSB of the 26 bit. If MSB = 1, concat 26 1's the front. If MSB = 0, concat 26 0's to the front.

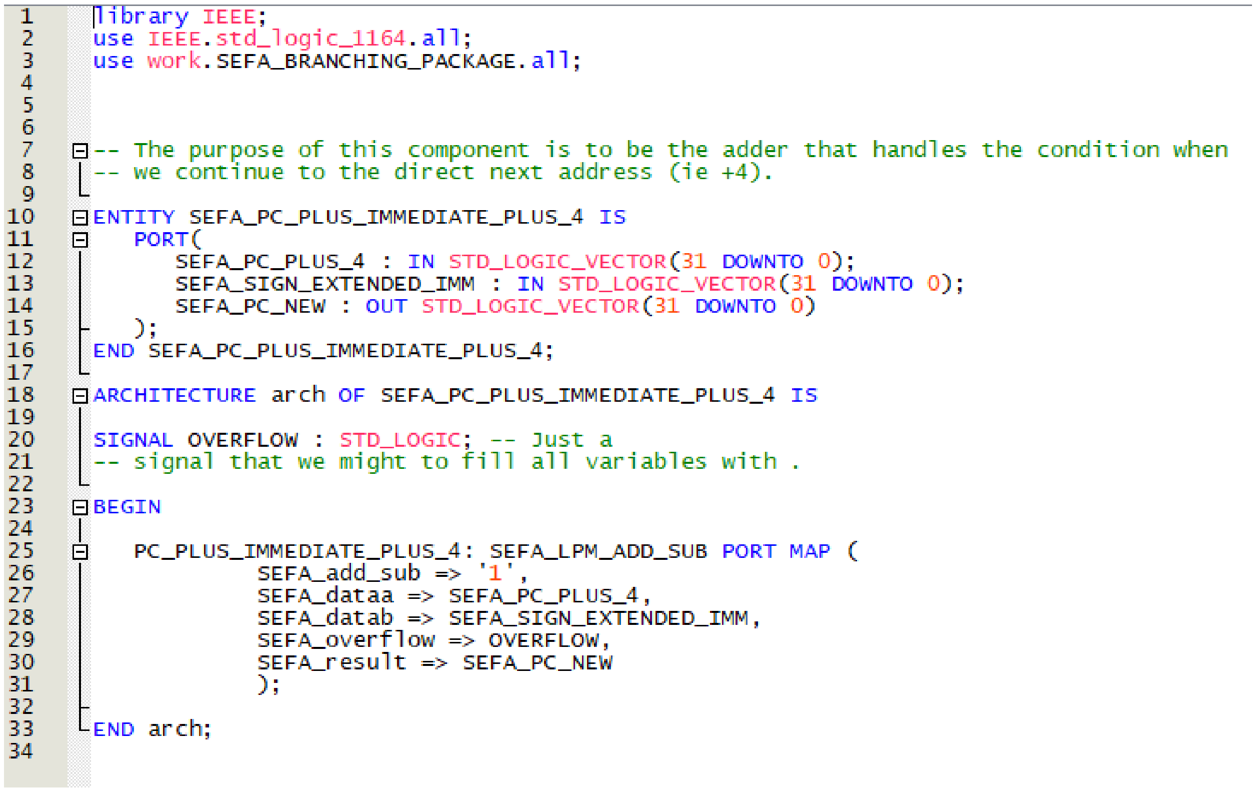
SEFA_BRANCHING_SIGN_EXTENDED



```
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


ENTITY SEFA_BRANCHING_SIGNED_EXTENDED IS
PORT(

          SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
          SEFA_SIGNED_16_to_32_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
          SEFA_SIGNED_26_to_32_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
          SEFA_SIGNED_32_EXTENDED_IMM : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END SEFA_BRANCHING_SIGNED_EXTENDED;

ARCHITECTURE arch OF SEFA_BRANCHING_SIGNED_EXTENDED IS

BEGIN


     SEFA_SIGNED_32_EXTENDED_IMM <= SEFA_SIGNED_26_to_32_EXTENDED_IMM WHEN SEFA_OPCODE = "111111"
                                 ELSE SEFA_SIGNED_16_to_32_EXTENDED_IMM;

END arch;

```

This component is used to choose which Sign extended value to compute, that is 16 bit for BNE or BEQ or 26 bit for J MIPS instruction. This is done using a MUX and the opcode.

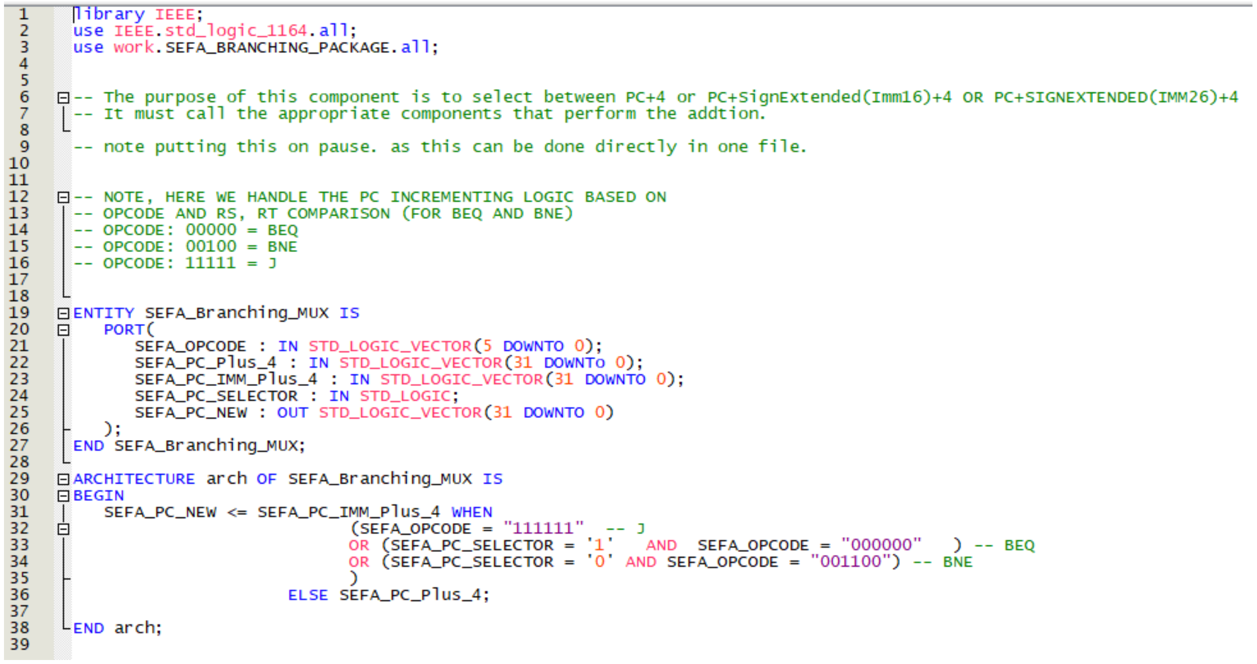
SEFA_PC_PLUS_IMMEDIATE_PLUS_4



```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;



-- The purpose of this component is to be the adder that handles the condition when
-- we continue to the direct next address (ie +4).

ENTITY SEFA_PC_PLUS_IMMEDIATE_PLUS_4 IS
    PORT(
        SEFA_PC_PLUS_4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_SIGN_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END SEFA_PC_PLUS_IMMEDIATE_PLUS_4;

ARCHITECTURE arch OF SEFA_PC_PLUS_IMMEDIATE_PLUS_4 IS

SIGNAL OVERFLOW : STD_LOGIC; -- Just a
-- signal that we might to fill all variables with .

BEGIN

    PC_PLUS_IMMEDIATE_PLUS_4: SEFA_LPM_ADD_SUB PORT MAP (
                SEFA_add_sub => '1',
                SEFA_dataa => SEFA_PC_PLUS_4,
                SEFA_datab => SEFA_SIGN_EXTENDED_IMM,
                SEFA_overflow => OVERFLOW,
                SEFA_result => SEFA_PC_NEW
                );

END arch;
```

This component adds the immediate value, which was previously chosen to PC_PLUS_4. Note the PC_PLUS_4 was already computed via its own component.

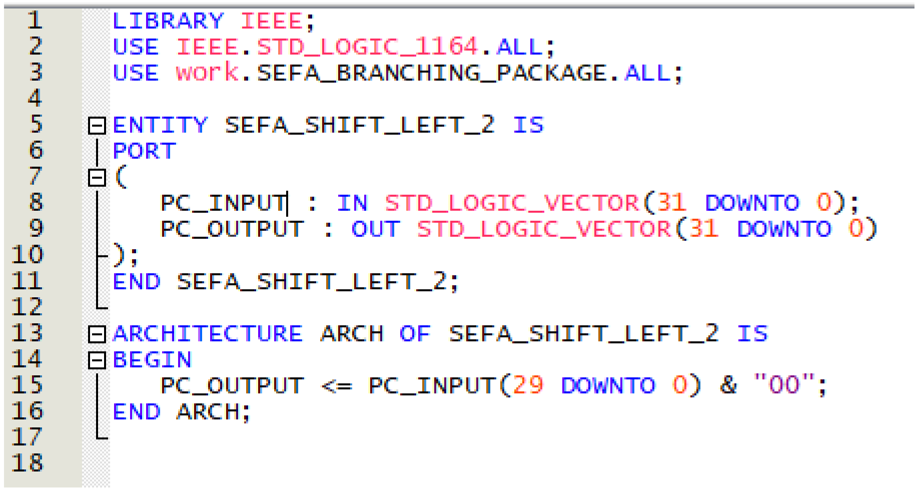
SEFA_Branching_MUX



```
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


-- The purpose of this component is to select between PC+4 or PC+SignExtended(Imm16)+4 OR PC+SIGNEXTENDED(IMM26)+4
-- It must call the appropriate components that perform the addtion.

-- note putting this on pause. as this can be done directly in one file.


-- NOTE, HERE WE HANDLE THE PC INCREMENTING LOGIC BASED ON
-- OPCODE AND RS, RT COMPARISON (FOR BEQ AND BNE)
-- OPCODE: 00000 = BEQ
-- OPCODE: 00100 = BNE
-- OPCODE: 11111 = J


ENTITY SEFA_Branching_MUX IS
    PORT(
        SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        SEFA_PC_Plus_4 : IN STD_LOGIC_VECTOR(31 DOWNTo 0);
        SEFA_PC_IMM_Plus_4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_PC_SELECTOR : IN STD_LOGIC;
        SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END SEFA_Branching_MUX;

ARCHITECTURE arch OF SEFA_Branching_MUX IS
BEGIN
    SEFA_PC_NEW <= SEFA_PC_IMM_Plus_4 WHEN
                                (SEFA_OPCODE = "111111"  -- J
                                OR (SEFA_PC_SELECTOR = '1'   AND  SEFA_OPCODE = "000000"   ) -- BEQ
                                OR (SEFA_PC_SELECTOR = '0' AND SEFA_OPCODE = "001100") -- BNE
                                )
                        ELSE SEFA_PC_Plus_4;

END arch;

```

This component selects which PC value to take, via a behavioral multiplexer. It uses the PC cond value (for BNE and BEQ) as well as the OPCODE value to decide if we compute PC+4 or PC+IMM+4.

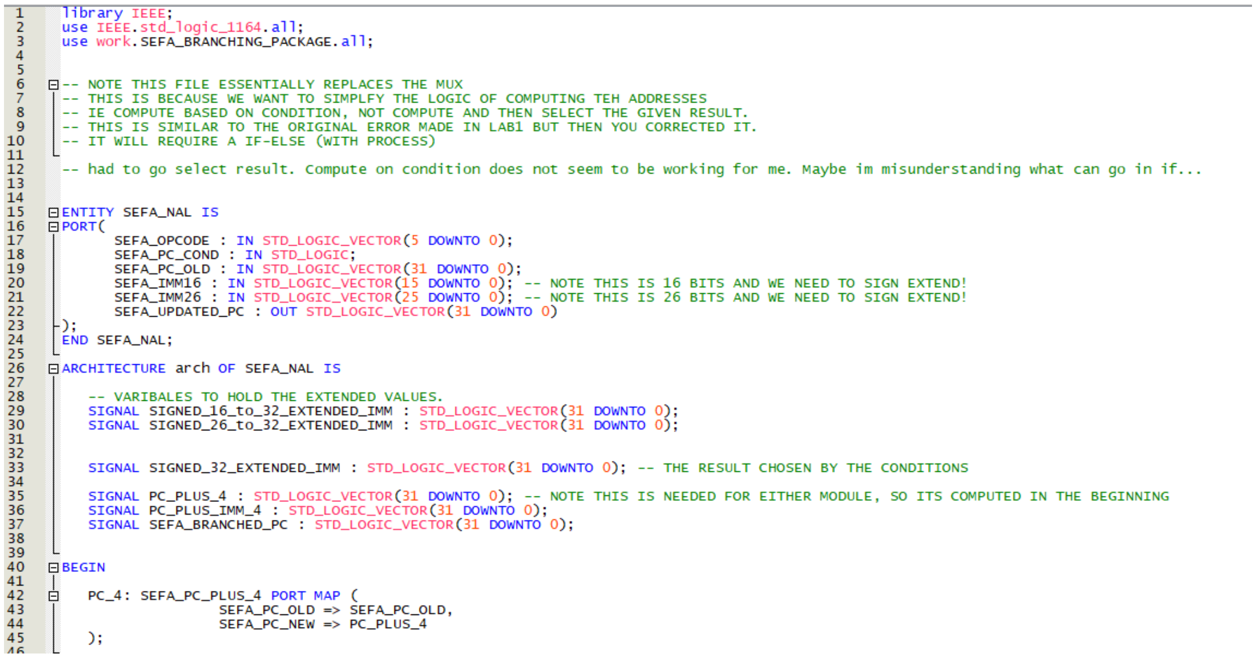
SEFA_SHIFT_LEFT_2



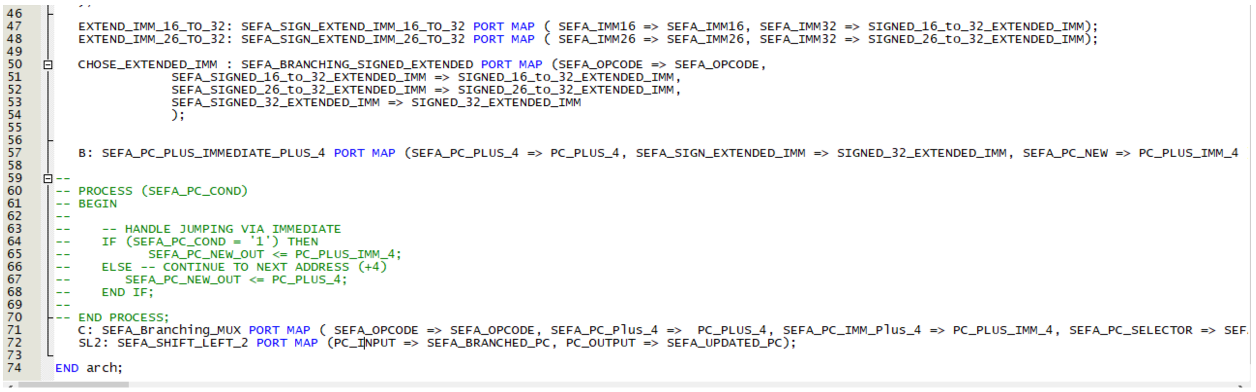
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE work.SEFA_BRANCHING_PACKAGE.ALL;

ENTITY SEFA_SHIFT_LEFT_2 IS
PORT
(
    PC_INPUT : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    PC_OUTPUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END SEFA_SHIFT_LEFT_2;

ARCHITECTURE ARCH OF SEFA_SHIFT_LEFT_2 IS
BEGIN
    PC_OUTPUT <= PC_INPUT(29 DOWNTO 0) & "00";
END ARCH;

```

This component shifts the new PC value by 2 bits, as described in the diagram. This is done by adding two 00 bits to the end of the 29-0 bits of the computed PC value.

SEFA_NAL



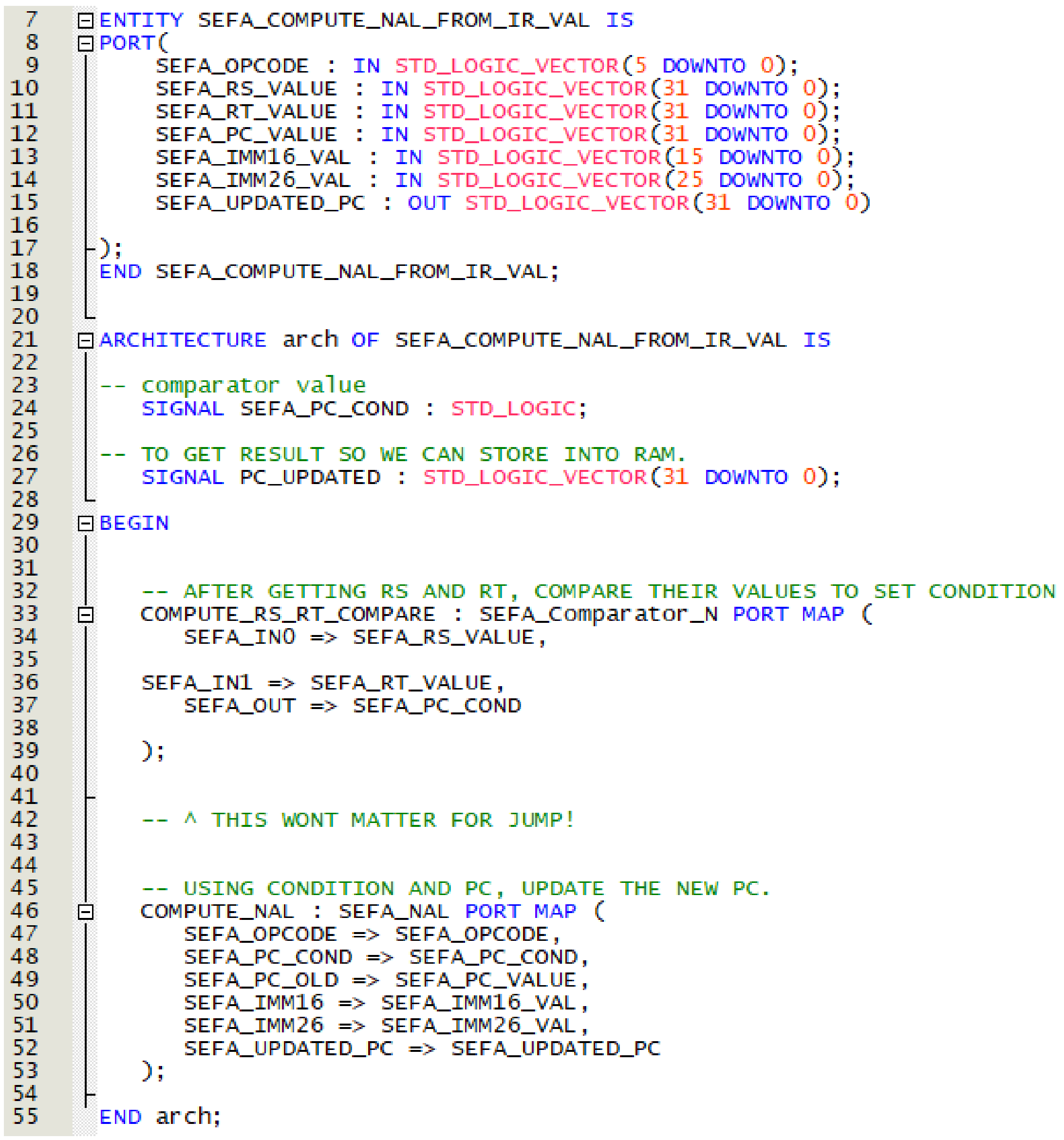
```
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;


-- NOTE THIS FILE ESSENTIALLY REPLACES THE MUX
-- THIS IS BECAUSE WE WANT TO SIMPLFY THE LOGIC OF COMPUTING TEH ADDRESSES
-- IE COMPUTE BASED ON CONDITION, NOT COMPUTE AND THEN SELECT THE GIVEN RESULT.
-- THIS IS SIMILAR TO THE ORIGINAL ERROR MADE IN LAB1 BUT THEN YOU CORRECTED IT.
-- IT WILL REQUIRE A IF-ELSE (WITH PROCESS)

-- had to go select result. Compute on condition does not seem to be working for me. Maybe im misunderstanding what can go in if...


ENTITY SEFA_NAL IS
PORT(
      SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
      SEFA_PC_COND : IN STD_LOGIC;
      SEFA_PC_OLD : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      SEFA_IMM16 : IN STD_LOGIC_VECTOR(15 DOWNTO 0); -- NOTE THIS IS 16 BITS AND WE NEED TO SIGN EXTEND!
      SEFA_IMM26 : IN STD_LOGIC_VECTOR(25 DOWNTO 0); -- NOTE THIS IS 26 BITS AND WE NEED TO SIGN EXTEND!
      SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END SEFA_NAL;

ARCHITECTURE arch OF SEFA_NAL IS

   -- VARIBALES TO HOLD THE EXTENDED VALUES.
   SIGNAL SIGNED_16_to_32_EXTENDED_IMM : STD_LOGIC_VECTOR(31 DOWNTO 0);
   SIGNAL SIGNED_26_to_32_EXTENDED_IMM : STD_LOGIC_VECTOR(31 DOWNTO 0);


   SIGNAL SIGNED_32_EXTENDED_IMM : STD_LOGIC_VECTOR(31 DOWNTO 0); -- THE RESULT CHOSEN BY THE CONDITIONS

   SIGNAL PC_PLUS_4 : STD_LOGIC_VECTOR(31 DOWNTO 0); -- NOTE THIS IS NEEDED FOR EITHER MODULE, SO ITS COMPUTED IN THE BEGINNING
   SIGNAL PC_PLUS_IMM_4 : STD_LOGIC_VECTOR(31 DOWNTO 0);
   SIGNAL SEFA_BRANCHED_PC : STD_LOGIC_VECTOR(31 DOWNTO 0);


BEGIN

   PC_4: SEFA_PC_PLUS_4 PORT MAP (
                  SEFA_PC_OLD => SEFA_PC_OLD,
                  SEFA_PC_NEW => PC_PLUS_4
   );
```



```
   EXTEND_IMM_16_TO_32: SEFA_SIGN_EXTEND_IMM_16_TO_32 PORT MAP ( SEFA_IMM16 => SEFA_IMM16, SEFA_IMM32 => SIGNED_16_to_32_EXTENDED_IMM);
   EXTEND_IMM_26_TO_32: SEFA_SIGN_EXTEND_IMM_26_TO_32 PORT MAP ( SEFA_IMM26 => SEFA_IMM26, SEFA_IMM32 => SIGNED_26_to_32_EXTENDED_IMM);

   CHOSE_EXTENDED_IMM : SEFA_BRANCHING_SIGNED_EXTENDED PORT MAP (SEFA_OPCODE => SEFA_OPCODE,
               SEFA_SIGNED_16_to_32_EXTENDED_IMM => SIGNED_16_to_32_EXTENDED_IMM,
               SEFA_SIGNED_26_to_32_EXTENDED_IMM => SIGNED_26_to_32_EXTENDED_IMM,
               SEFA_SIGNED_32_EXTENDED_IMM => SIGNED_32_EXTENDED_IMM
               );


   B: SEFA_PC_PLUS_IMMEDIATE_PLUS_4 PORT MAP (SEFA_PC_PLUS_4 => PC_PLUS_4, SEFA_SIGN_EXTENDED_IMM => SIGNED_32_EXTENDED_IMM, SEFA_PC_NEW => PC_PLUS_IMM_4

--
-- PROCESS (SEFA_PC_COND)
-- BEGIN
--
--    -- HANDLE JUMPING VIA IMMEDIATE
--    IF (SEFA_PC_COND = '1') THEN
--          SEFA_PC_NEW_OUT <= PC_PLUS_IMM_4;
--    ELSE -- CONTINUE TO NEXT ADDRESS (+4)
--       SEFA_PC_NEW_OUT <= PC_PLUS_4;
--    END IF;
--
-- END PROCESS;
   C: SEFA_Branching_MUX PORT MAP ( SEFA_OPCODE => SEFA_OPCODE, SEFA_PC_Plus_4 =>  PC_PLUS_4, SEFA_PC_IMM_Plus_4 => PC_PLUS_IMM_4, SEFA_PC_SELECTOR => SEF
   SL2: SEFA_SHIFT_LEFT_2 PORT MAP (PC_INPUT => SEFA_BRANCHED_PC, PC_OUTPUT => SEFA_UPDATED_PC);

END arch;
```

This is the NAL component which extends the immediate values, computes the respective PC additions, calles the 2-1 MUX to handle which PC value is accepted, and shifts left two the PC value.

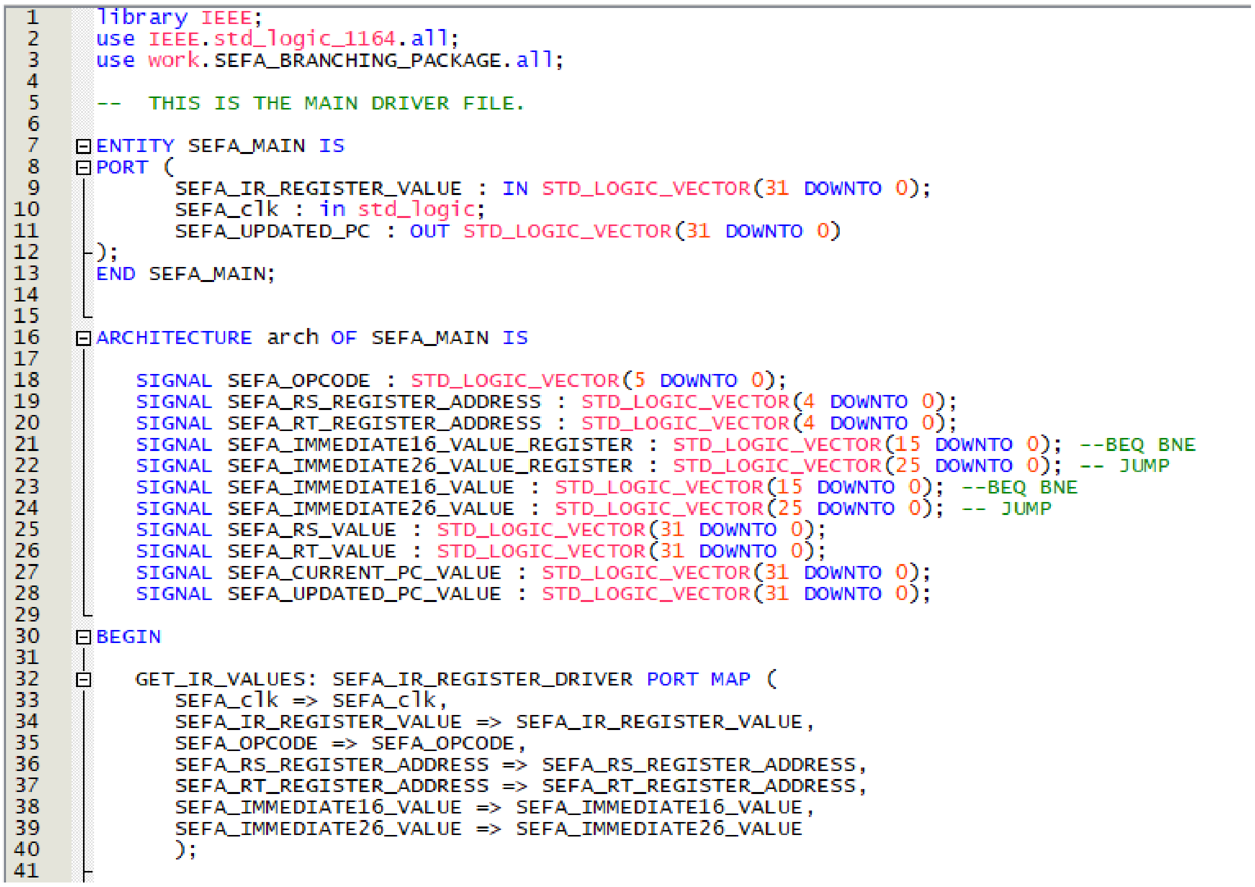
SEFA_COMPUTE_NAL_FROM_IR_VALUE



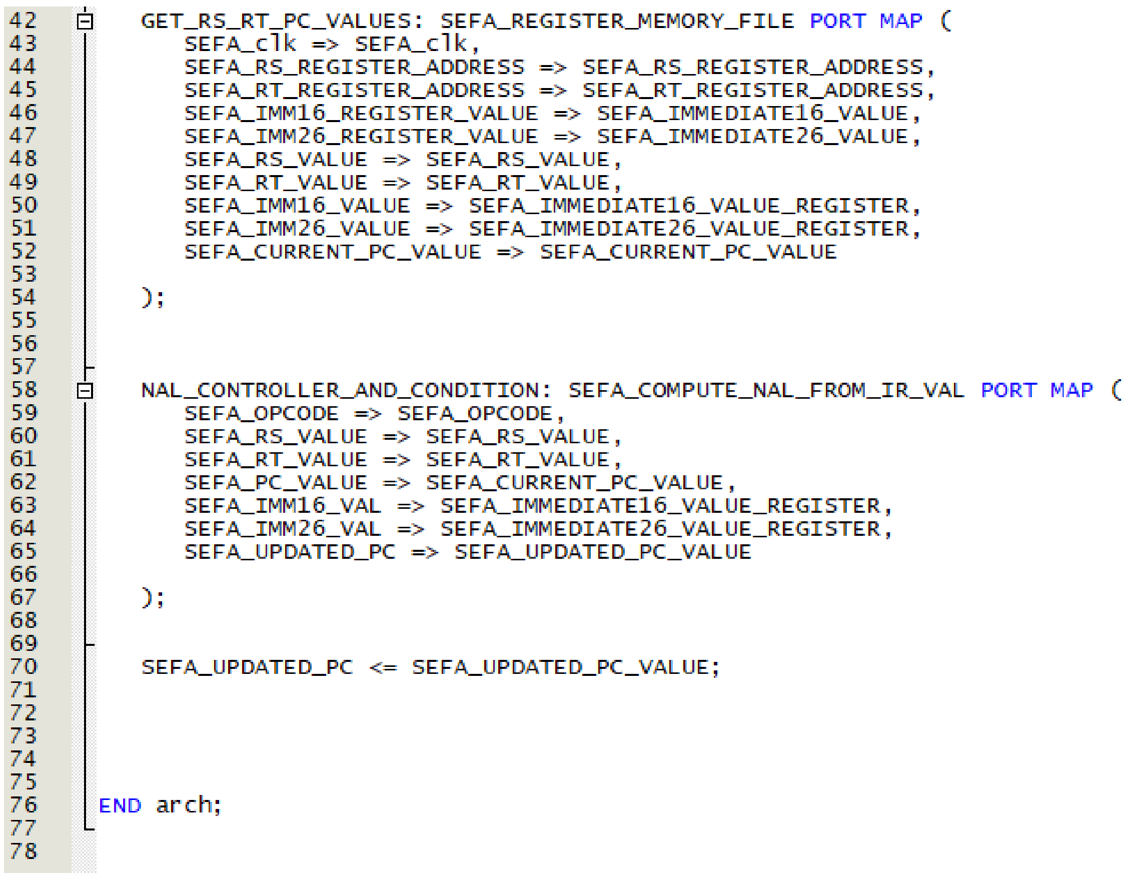
```vhdl
ENTITY SEFA_COMPUTE_NAL_FROM_IR_VAL IS
PORT(
    SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
    SEFA_RS_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    SEFA_RT_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    SEFA_PC_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    SEFA_IMM16_VAL : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    SEFA_IMM26_VAL : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
    SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)

);
END SEFA_COMPUTE_NAL_FROM_IR_VAL;


ARCHITECTURE arch OF SEFA_COMPUTE_NAL_FROM_IR_VAL IS

-- comparator value
   SIGNAL SEFA_PC_COND : STD_LOGIC;

-- TO GET RESULT SO WE CAN STORE INTO RAM.
   SIGNAL PC_UPDATED : STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN


   -- AFTER GETTING RS AND RT, COMPARE THEIR VALUES TO SET CONDITION
   COMPUTE_RS_RT_COMPARE : SEFA_Comparator_N PORT MAP (
      SEFA_IN0 => SEFA_RS_VALUE,

   SEFA_IN1 => SEFA_RT_VALUE,
      SEFA_OUT => SEFA_PC_COND

   );


   -- ^ THIS WONT MATTER FOR JUMP!


   -- USING CONDITION AND PC, UPDATE THE NEW PC.
   COMPUTE_NAL : SEFA_NAL PORT MAP (
      SEFA_OPCODE => SEFA_OPCODE,
      SEFA_PC_COND => SEFA_PC_COND,
      SEFA_PC_OLD => SEFA_PC_VALUE,
      SEFA_IMM16 => SEFA_IMM16_VAL,
      SEFA_IMM26 => SEFA_IMM26_VAL,
      SEFA_UPDATED_PC => SEFA_UPDATED_PC
   );

END arch;
```

This file is a controller for NAL. It calls the comparator to get the PC condition (used for BEQ and BNE). It then passes in the PC condition value into SEFA_NAL which handles the NAL logic.

SEFA_MAIN



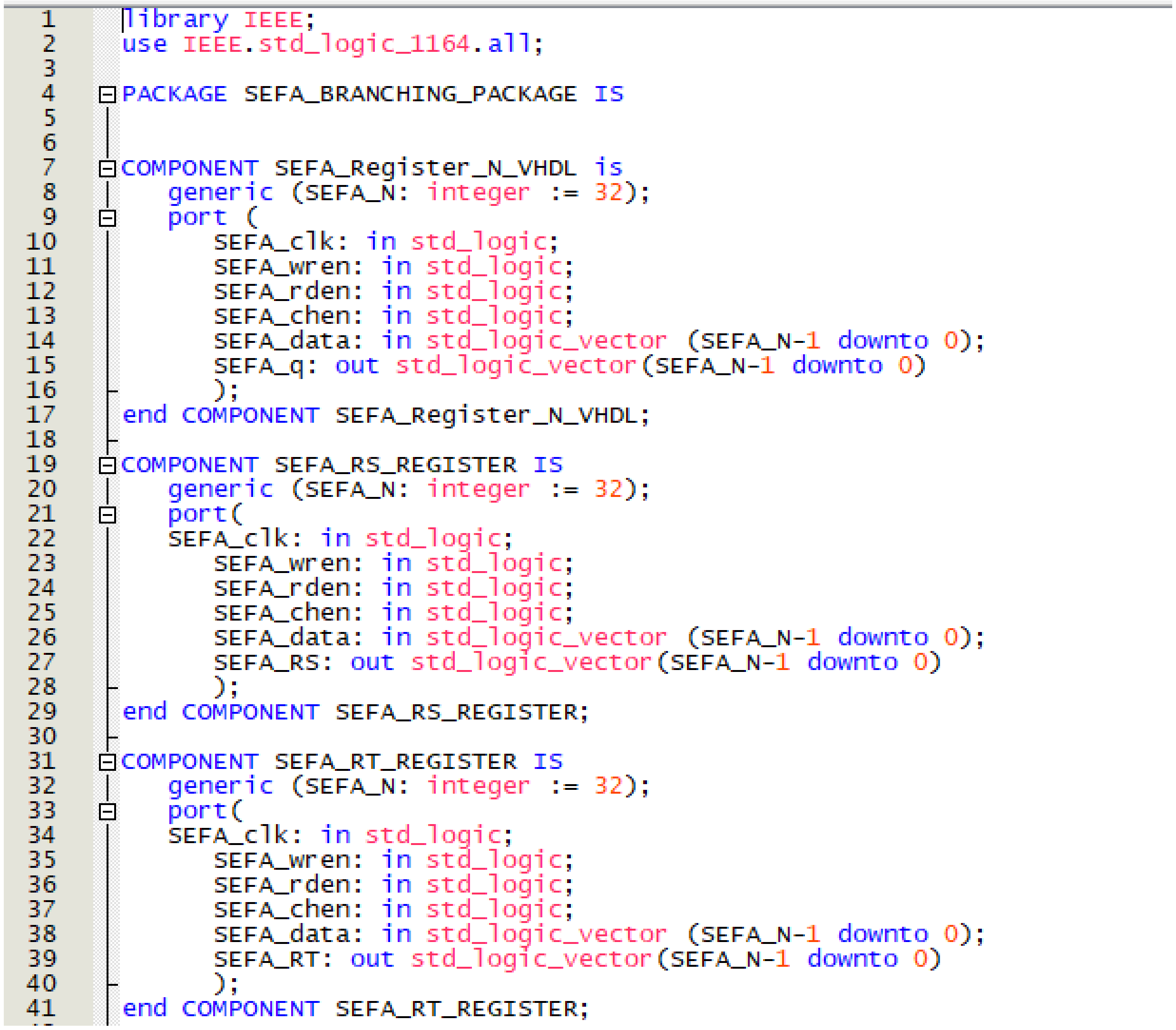
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.SEFA_BRANCHING_PACKAGE.all;

--  THIS IS THE MAIN DRIVER FILE.

ENTITY SEFA_MAIN IS
PORT (
      SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      SEFA_clk : in std_logic;
      SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END SEFA_MAIN;


ARCHITECTURE arch OF SEFA_MAIN IS

   SIGNAL SEFA_OPCODE : STD_LOGIC_VECTOR(5 DOWNTO 0);
   SIGNAL SEFA_RS_REGISTER_ADDRESS : STD_LOGIC_VECTOR(4 DOWNTO 0);
   SIGNAL SEFA_RT_REGISTER_ADDRESS : STD_LOGIC_VECTOR(4 DOWNTO 0);
   SIGNAL SEFA_IMMEDIATE16_VALUE_REGISTER : STD_LOGIC_VECTOR(15 DOWNTO 0); --BEQ BNE
   SIGNAL SEFA_IMMEDIATE26_VALUE_REGISTER : STD_LOGIC_VECTOR(25 DOWNTO 0); -- JUMP
   SIGNAL SEFA_IMMEDIATE16_VALUE : STD_LOGIC_VECTOR(15 DOWNTO 0); --BEQ BNE
   SIGNAL SEFA_IMMEDIATE26_VALUE : STD_LOGIC_VECTOR(25 DOWNTO 0); -- JUMP
   SIGNAL SEFA_RS_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
   SIGNAL SEFA_RT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
   SIGNAL SEFA_CURRENT_PC_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
   SIGNAL SEFA_UPDATED_PC_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);

BEGIN

   GET_IR_VALUES: SEFA_IR_REGISTER_DRIVER PORT MAP (
      SEFA_clk => SEFA_clk,
      SEFA_IR_REGISTER_VALUE => SEFA_IR_REGISTER_VALUE,
      SEFA_OPCODE => SEFA_OPCODE,
      SEFA_RS_REGISTER_ADDRESS => SEFA_RS_REGISTER_ADDRESS,
      SEFA_RT_REGISTER_ADDRESS => SEFA_RT_REGISTER_ADDRESS,
      SEFA_IMMEDIATE16_VALUE => SEFA_IMMEDIATE16_VALUE,
      SEFA_IMMEDIATE26_VALUE => SEFA_IMMEDIATE26_VALUE
      );

```



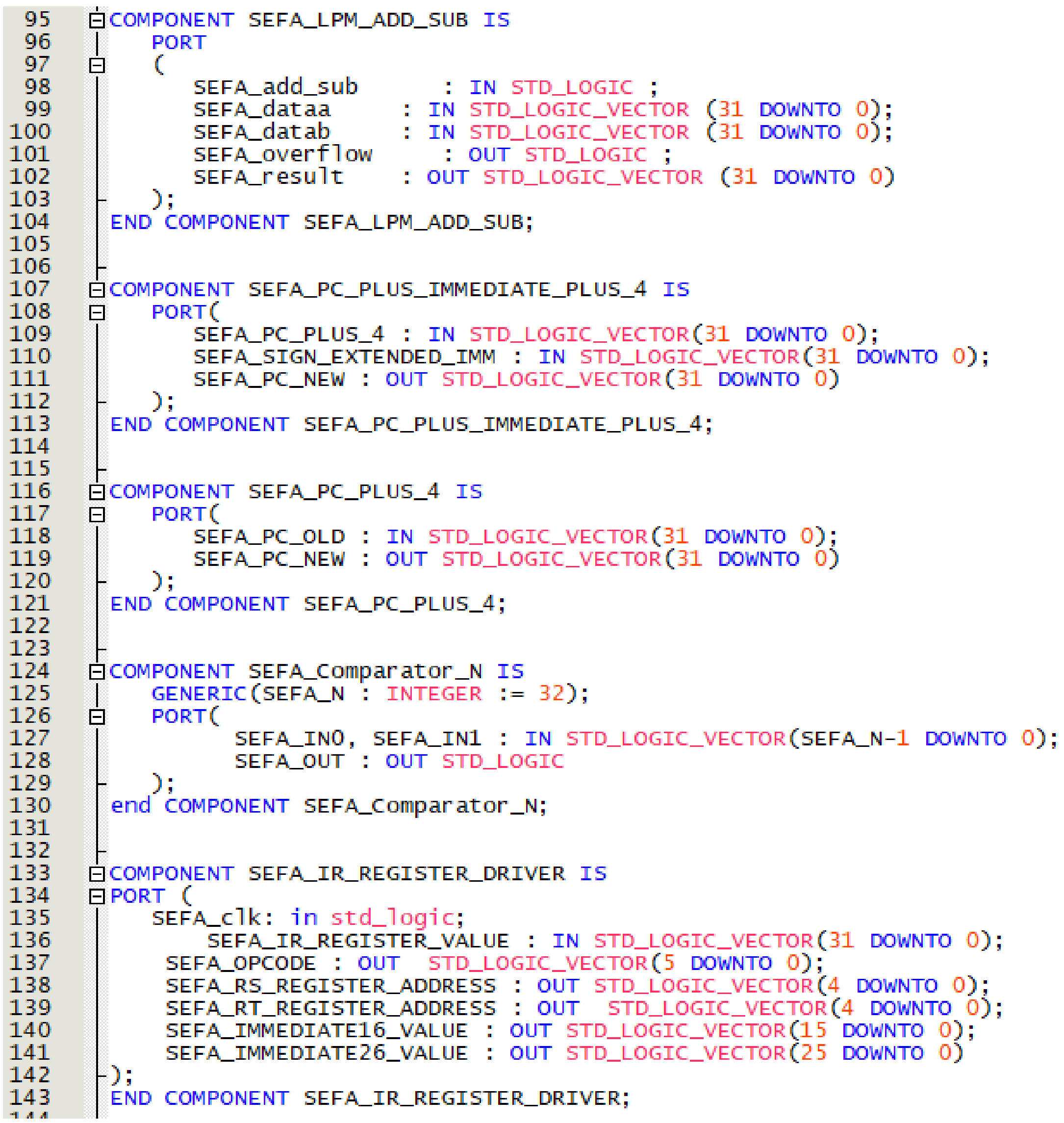
```
	GET_RS_RT_PC_VALUES: SEFA_REGISTER_MEMORY_FILE PORT MAP (
	   SEFA_clk => SEFA_clk,
	   SEFA_RS_REGISTER_ADDRESS => SEFA_RS_REGISTER_ADDRESS,
	   SEFA_RT_REGISTER_ADDRESS => SEFA_RT_REGISTER_ADDRESS,
	   SEFA_IMM16_REGISTER_VALUE => SEFA_IMMEDIATE16_VALUE,
	   SEFA_IMM26_REGISTER_VALUE => SEFA_IMMEDIATE26_VALUE,
	   SEFA_RS_VALUE => SEFA_RS_VALUE,
	   SEFA_RT_VALUE => SEFA_RT_VALUE,
	   SEFA_IMM16_VALUE => SEFA_IMMEDIATE16_VALUE_REGISTER,
	   SEFA_IMM26_VALUE => SEFA_IMMEDIATE26_VALUE_REGISTER,
	   SEFA_CURRENT_PC_VALUE => SEFA_CURRENT_PC_VALUE

	);



	NAL_CONTROLLER_AND_CONDITION: SEFA_COMPUTE_NAL_FROM_IR_VAL PORT MAP (
	   SEFA_OPCODE => SEFA_OPCODE,
	   SEFA_RS_VALUE => SEFA_RS_VALUE,
	   SEFA_RT_VALUE => SEFA_RT_VALUE,
	   SEFA_PC_VALUE => SEFA_CURRENT_PC_VALUE,
	   SEFA_IMM16_VAL => SEFA_IMMEDIATE16_VALUE_REGISTER,
	   SEFA_IMM26_VAL => SEFA_IMMEDIATE26_VALUE_REGISTER,
	   SEFA_UPDATED_PC => SEFA_UPDATED_PC_VALUE

	);


	SEFA_UPDATED_PC <= SEFA_UPDATED_PC_VALUE;





END arch;
```

This component is the main controller for the project. It takes in the IR value, and the clock, and calls the respective components to compute the new PC value.
It calls the SEFA_IR_REGISTER_VALUE component to split apart the IR into its respective fields; It calls the SEFA_REGISTER_MEMORY_FILE to handle getting RS, RT, IMM, and PC from RAM (if applicable) and into Registers; It calls SEFA_COMPUTE_NAL_FROM_IR_VALUE which hands all of the NAL logic as well as comparator logic.

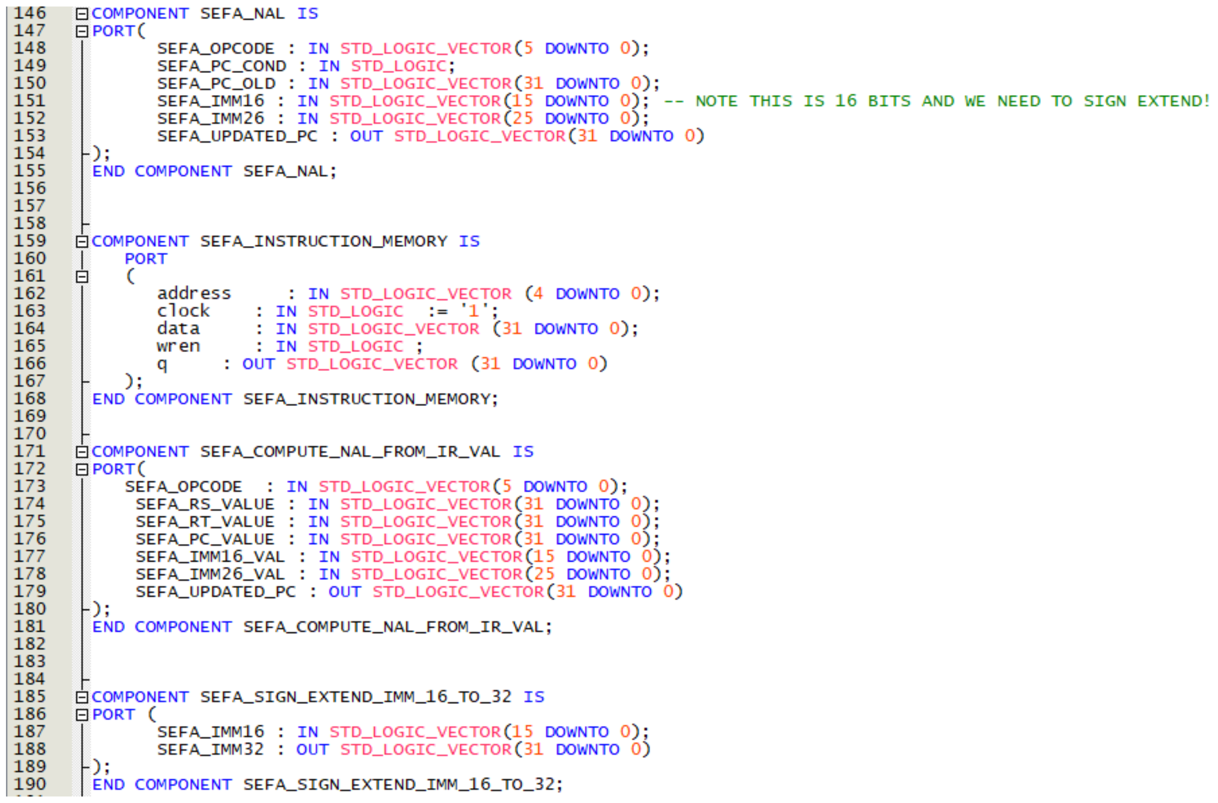
SEFA_BRANCHING_PACKAGE



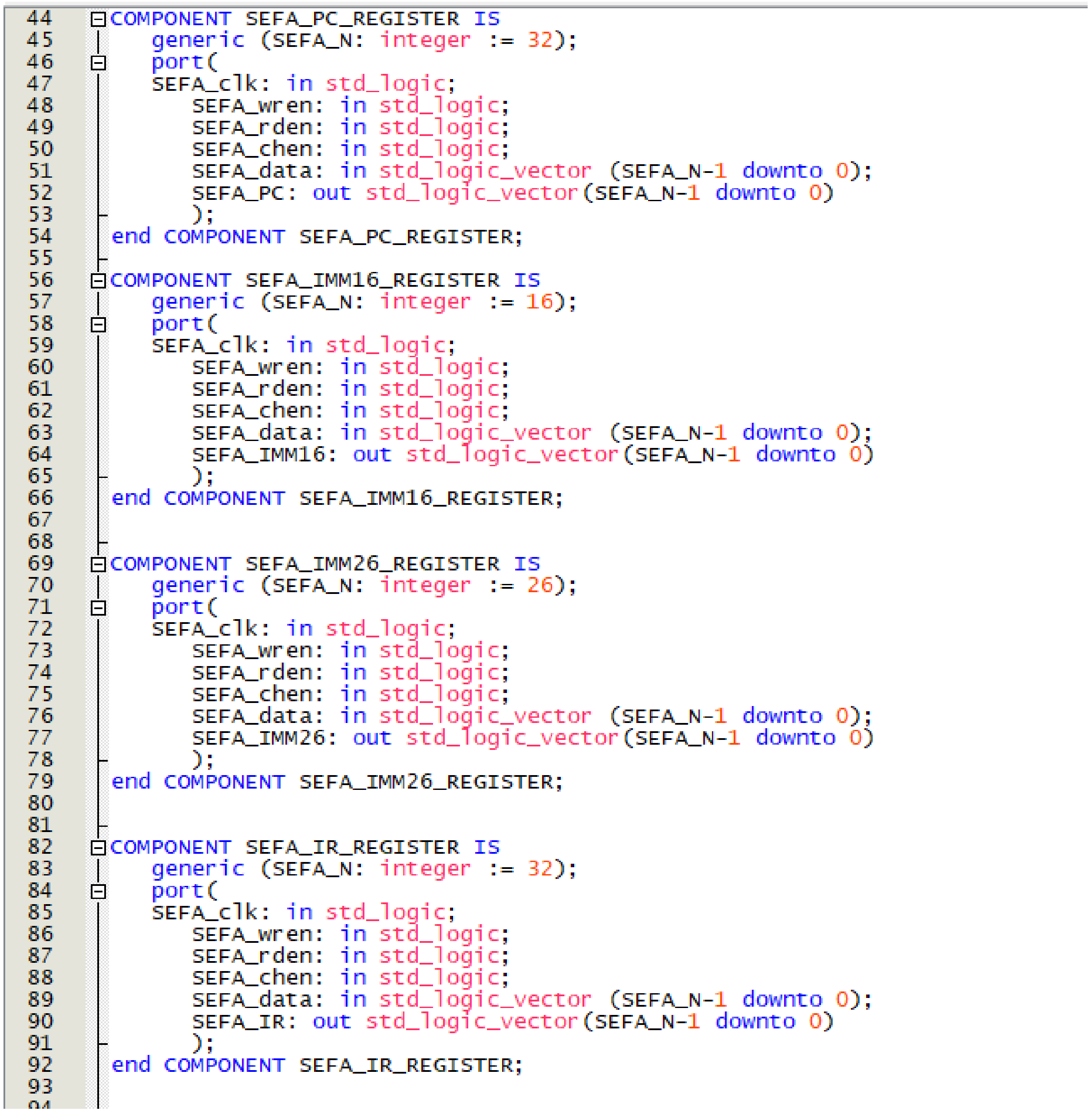
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

PACKAGE SEFA_BRANCHING_PACKAGE IS


COMPONENT SEFA_Register_N_VHDL is
    generic (SEFA_N: integer := 32);
    port (
        SEFA_clk: in std_logic;
        SEFA_wren: in std_logic;
        SEFA_rden: in std_logic;
        SEFA_chen: in std_logic;
        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
        SEFA_q: out std_logic_vector(SEFA_N-1 downto 0)
        );
end COMPONENT SEFA_Register_N_VHDL;

COMPONENT SEFA_RS_REGISTER IS
    generic (SEFA_N: integer := 32);
    port(
    SEFA_clk: in std_logic;
        SEFA_wren: in std_logic;
        SEFA_rden: in std_logic;
        SEFA_chen: in std_logic;
        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
        SEFA_RS: out std_logic_vector(SEFA_N-1 downto 0)
        );
end COMPONENT SEFA_RS_REGISTER;

COMPONENT SEFA_RT_REGISTER IS
    generic (SEFA_N: integer := 32);
    port(
    SEFA_clk: in std_logic;
        SEFA_wren: in std_logic;
        SEFA_rden: in std_logic;
        SEFA_chen: in std_logic;
        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
        SEFA_RT: out std_logic_vector(SEFA_N-1 downto 0)
        );
end COMPONENT SEFA_RT_REGISTER;
```



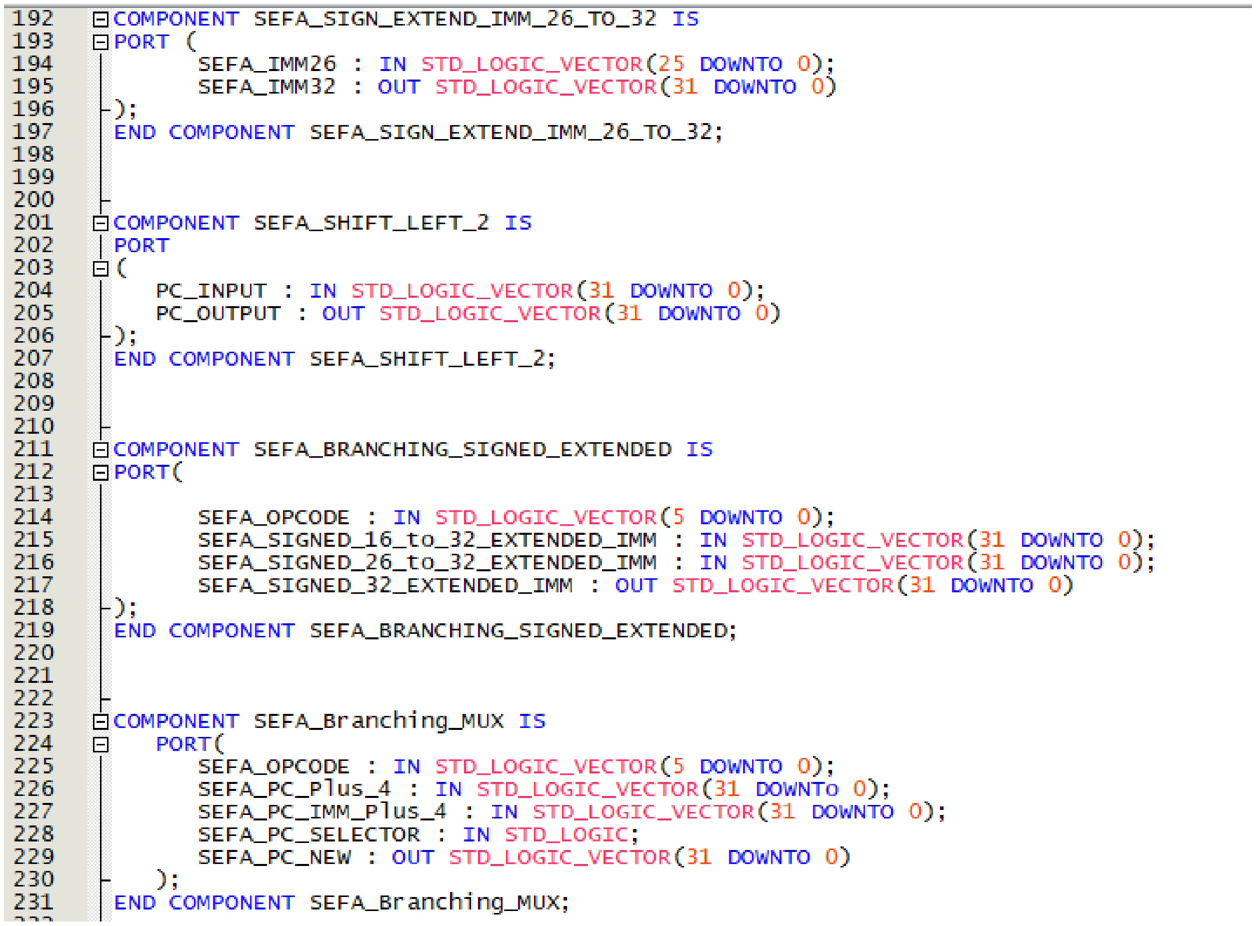
```
COMPONENT SEFA_LPM_ADD_SUB IS
    PORT
    (
        SEFA_add_sub        : IN STD_LOGIC ;
        SEFA_dataa      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        SEFA_datab      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        SEFA_overflow       : OUT STD_LOGIC ;
        SEFA_result     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END COMPONENT SEFA_LPM_ADD_SUB;


COMPONENT SEFA_PC_PLUS_IMMEDIATE_PLUS_4 IS
    PORT(
        SEFA_PC_PLUS_4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_SIGN_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END COMPONENT SEFA_PC_PLUS_IMMEDIATE_PLUS_4;


COMPONENT SEFA_PC_PLUS_4 IS
    PORT(
        SEFA_PC_OLD : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END COMPONENT SEFA_PC_PLUS_4;


COMPONENT SEFA_Comparator_N IS
    GENERIC(SEFA_N : INTEGER := 32);
    PORT(
            SEFA_IN0, SEFA_IN1 : IN STD_LOGIC_VECTOR(SEFA_N-1 DOWNTO 0);
            SEFA_OUT : OUT STD_LOGIC
    );
end COMPONENT SEFA_Comparator_N;


COMPONENT SEFA_IR_REGISTER_DRIVER IS
PORT (
    SEFA_clk: in std_logic;
        SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
     SEFA_OPCODE : OUT  STD_LOGIC_VECTOR(5 DOWNTO 0);
     SEFA_RS_REGISTER_ADDRESS : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
     SEFA_RT_REGISTER_ADDRESS : OUT  STD_LOGIC_VECTOR(4 DOWNTO 0);
     SEFA_IMMEDIATE16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
     SEFA_IMMEDIATE26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNTO 0)
);
END COMPONENT SEFA_IR_REGISTER_DRIVER;
```



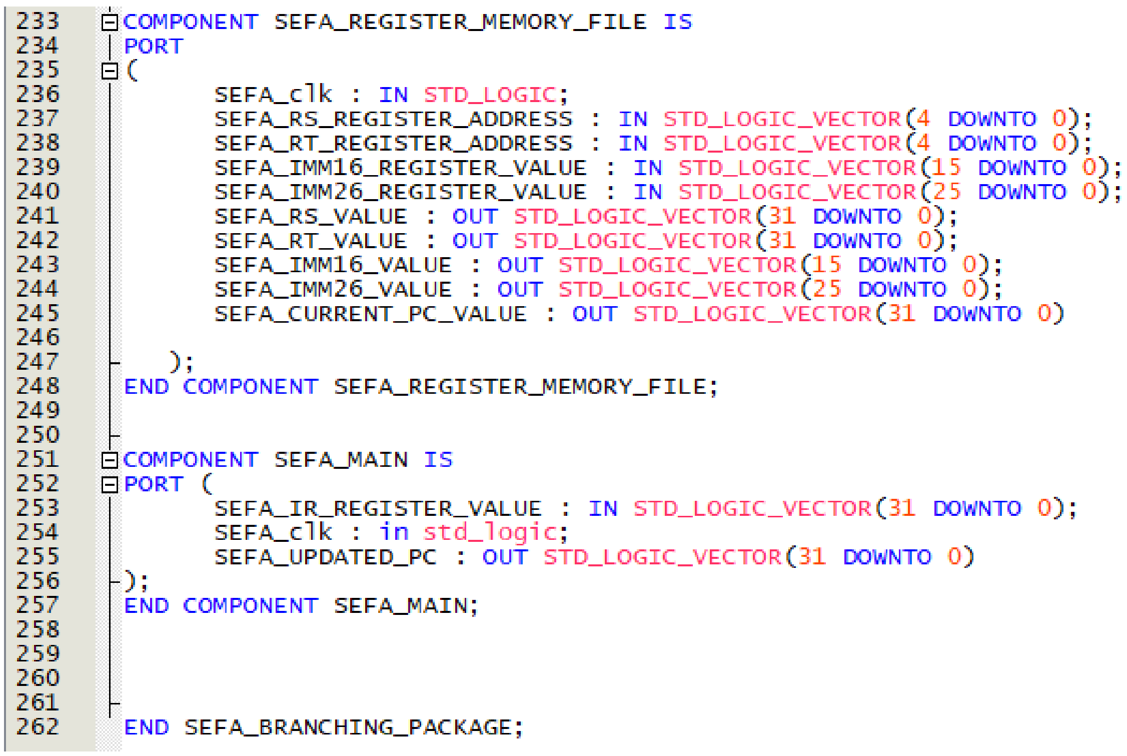
```
COMPONENT SEFA_NAL IS
PORT(
        SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        SEFA_PC_COND : IN STD_LOGIC;
        SEFA_PC_OLD : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_IMM16 : IN STD_LOGIC_VECTOR(15 DOWNTO 0); -- NOTE THIS IS 16 BITS AND WE NEED TO SIGN EXTEND!
        SEFA_IMM26 : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
        SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END COMPONENT SEFA_NAL;



COMPONENT SEFA_INSTRUCTION_MEMORY IS
    PORT
    (
        address     : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        clock     : IN STD_LOGIC  := '1';
        data      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        wren      : IN STD_LOGIC ;
        q      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END COMPONENT SEFA_INSTRUCTION_MEMORY;


COMPONENT SEFA_COMPUTE_NAL_FROM_IR_VAL IS
PORT(
    SEFA_OPCODE  : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
     SEFA_RS_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
     SEFA_RT_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
     SEFA_PC_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
     SEFA_IMM16_VAL : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
     SEFA_IMM26_VAL : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
     SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END COMPONENT SEFA_COMPUTE_NAL_FROM_IR_VAL;



COMPONENT SEFA_SIGN_EXTEND_IMM_16_TO_32 IS
PORT (
        SEFA_IMM16 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        SEFA_IMM32 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END COMPONENT SEFA_SIGN_EXTEND_IMM_16_TO_32;
```



```vhdl
COMPONENT SEFA_PC_REGISTER IS
   generic (SEFA_N: integer := 32);
   port(
   SEFA_clk: in std_logic;
      SEFA_wren: in std_logic;
      SEFA_rden: in std_logic;
      SEFA_chen: in std_logic;
      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
      SEFA_PC: out std_logic_vector(SEFA_N-1 downto 0)
      );
end COMPONENT SEFA_PC_REGISTER;

COMPONENT SEFA_IMM16_REGISTER IS
   generic (SEFA_N: integer := 16);
   port(
   SEFA_clk: in std_logic;
      SEFA_wren: in std_logic;
      SEFA_rden: in std_logic;
      SEFA_chen: in std_logic;
      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
      SEFA_IMM16: out std_logic_vector(SEFA_N-1 downto 0)
      );
end COMPONENT SEFA_IMM16_REGISTER;


COMPONENT SEFA_IMM26_REGISTER IS
   generic (SEFA_N: integer := 26);
   port(
   SEFA_clk: in std_logic;
      SEFA_wren: in std_logic;
      SEFA_rden: in std_logic;
      SEFA_chen: in std_logic;
      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
      SEFA_IMM26: out std_logic_vector(SEFA_N-1 downto 0)
      );
end COMPONENT SEFA_IMM26_REGISTER;


COMPONENT SEFA_IR_REGISTER IS
   generic (SEFA_N: integer := 32);
   port(
   SEFA_clk: in std_logic;
      SEFA_wren: in std_logic;
      SEFA_rden: in std_logic;
      SEFA_chen: in std_logic;
      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
      SEFA_IR: out std_logic_vector(SEFA_N-1 downto 0)
      );
end COMPONENT SEFA_IR_REGISTER;

```



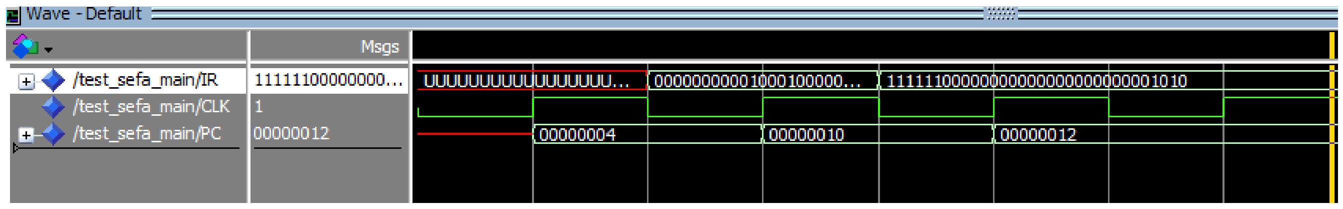
```
COMPONENT SEFA_SIGN_EXTEND_IMM_26_TO_32 IS
PORT (
        SEFA_IMM26 : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
        SEFA_IMM32 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END COMPONENT SEFA_SIGN_EXTEND_IMM_26_TO_32;



COMPONENT SEFA_SHIFT_LEFT_2 IS
PORT
(
    PC_INPUT : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    PC_OUTPUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END COMPONENT SEFA_SHIFT_LEFT_2;



COMPONENT SEFA_BRANCHING_SIGNED_EXTENDED IS
PORT(

        SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        SEFA_SIGNED_16_to_32_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_SIGNED_26_to_32_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_SIGNED_32_EXTENDED_IMM : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END COMPONENT SEFA_BRANCHING_SIGNED_EXTENDED;



COMPONENT SEFA_Branching_MUX IS
    PORT(
        SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        SEFA_PC_Plus_4 : IN STD_LOGIC_VECTOR(31 DOWNTo 0);
        SEFA_PC_IMM_Plus_4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_PC_SELECTOR : IN STD_LOGIC;
        SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END COMPONENT SEFA_Branching_MUX;
```



```
COMPONENT SEFA_REGISTER_MEMORY_FILE IS
PORT
(
        SEFA_clk : IN STD_LOGIC;
        SEFA_RS_REGISTER_ADDRESS : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        SEFA_RT_REGISTER_ADDRESS : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
        SEFA_IMM16_REGISTER_VALUE : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        SEFA_IMM26_REGISTER_VALUE : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
        SEFA_RS_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_RT_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_IMM16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        SEFA_IMM26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNTO 0);
        SEFA_CURRENT_PC_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)

    );
END COMPONENT SEFA_REGISTER_MEMORY_FILE;


COMPONENT SEFA_MAIN IS
PORT (
        SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        SEFA_clk : in std_logic;
        SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END COMPONENT SEFA_MAIN;




END SEFA_BRANCHING_PACKAGE;
```

This is the project package file with all of the components.

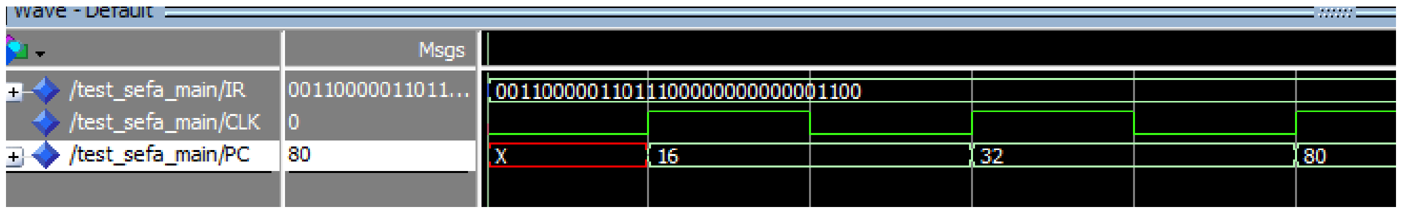
SEFA_TEST_MAIN



```
/test_sefa_main/IR   11111100000000...  UUUUUUUUUUUUUUUUU... | 00000000001000100000... | 11111100000000000000000000001010
/test_sefa_main/CLK  1
/test_sefa_main/PC   00000012           | 00000004 | 00000010 | 00000012
```

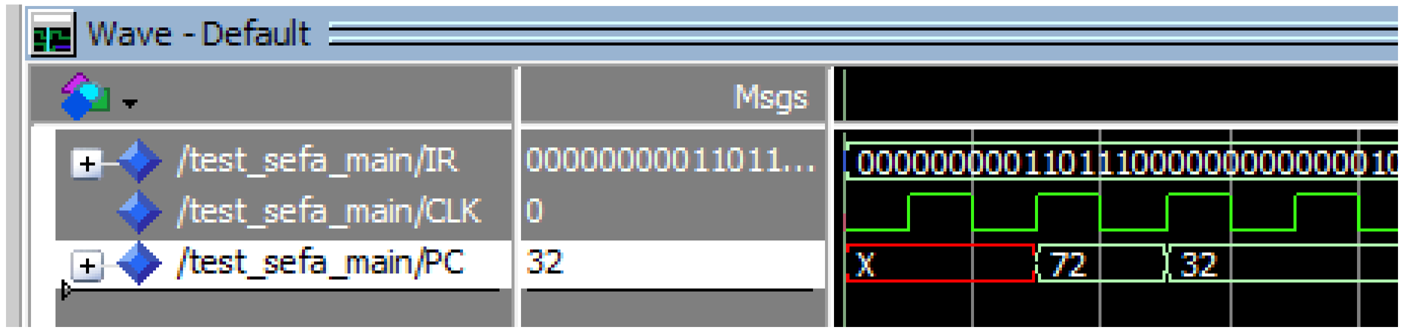
BEQ : IMM

J



```
/test_sefa_main/IR   00110000011011...  00110000011011100000000000001100
/test_sefa_main/CLK  0
/test_sefa_main/PC   80                 X | 16 | 32 | 80
```

BNE: IMM



```
/test_sefa_main/IR   00000000011011...  0000000001101110000000000000010
/test_sefa_main/CLK  0
/test_sefa_main/PC   32                 X | 72 | 32
```

BEQ: PC + 4

This is the test bench file for the project. Here we can verify that the PC is indeed updated respectively. That is for BEQ, when the value of RS and RT is equal, PC+IMM+4 happens, else PC+4. For BNE, when RS does not equal RT PC+IMM+4 happens, else PC+4. For J, it is just PC+IMM+4.