

Branching Lab
April 18 2021

CSC 342/343 Spring 2021
Bujar Sefa

Objective	2
SEFA_REGISTER_N	2
SEFA_IR_REGISTER	3
SEFA_RS_REGISTER	4
SEFA_RT_REGISTER	4
SEFA_IMM16_REGISTER	5
SEFA_IMM26_REGISTER	5
SEFA_PC_REGISTER	6
SEFA_IR_REGISTER_DRIVER	7
SEFA_INSTRUCTION_MEMORY	7
SEFA_REGISTER_MEMORY_FILE	9
SEFA_Comparator_N	10
SEFA_PC_PLUS_4	10
SEFA_SIGN_EXTEND_IMM_16_TO_32	11
SEFA_SIGN_EXTEND_IMM_26_TO_32	11
SEFA_BRANCHING_SIGN_EXTENDED	12
SEFA_PC_PLUS_IMMEDIATE_PLUS_4	12
SEFA_Branching_MUX	13
SEFA_SHIFT_LEFT_2	13
SEFA_NAL	14
SEFA_COMPUTE_NAL_FROM_IR_VALUE	14
SEFA_MAIN	16
SEFA_BRANCHING_PACKAGE	19

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

```

1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4
5  |
6  |-- NOTE, this is from Self Check Lab 4b (I just copied the code over)
7  |
8  |entity SEFA_Register_N_VHDL is
9  |  generic (SEFA_N: Integer := 32); -- The genetics feature permits us to change the side of this register desin easily. Its a const variable
10 |  port (
11 |    SEFA_clk: in std_logic; -- clock
12 |    SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
13 |    SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
14 |    SEFA_chen: in std_logic; -- chip enable (if it is 0, the output will be undefined)
15 |    SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
16 |    SEFA_q: out std_logic_vector (SEFA_N-1 downto 0) -- output. This is essentially just a display
17 |  );
18 |end SEFA_Register_N_VHDL;
19 |
20 |architecture arch of SEFA_Register_N_VHDL is
21 |  signal SEFA_storage : std_logic_vector (SEFA_N-1 downto 0);
22 |  -- This is the actual storage space of the register.
23 |  -- the bulk of this work will be coordinating how the data input flows into the storage space
24 |  -- and then to the output.
25 |begin
26 |  process (SEFA_clk)
27 |  -- we are defining a process that is sensitive to changes in clock value.
28 |  -- when defining a process, the paranthesis that come after it allow us to define our sensativity list.
29 |  begin
30 |    -- defining 32 bit wide flip flop that is rising edge triggered.
31 |    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
32 |    then SEFA_storage <= SEFA_data;
33 |    end if;
34 |  end process;
35 |
36 |  process (SEFA_rden, SEFA_chen, SEFA_storage)
37 |  -- this process is responsive to changes in read enable, chip enable, and the storage space.
38 |  -- Z is a common undefined logic value used for circuits. using the others feature allows us to set every bit in q as undefined.
39 |  begin
40 |    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
41 |    then SEFA_q <= SEFA_storage;
42 |    elsif (SEFA_chen = '0')
43 |    then SEFA_q <= (others => 'z');
44 |    end if;
45 |  end process;
46 |end arch;
47 |
48 |
49 |-- So to summarize. To make a N Register (that is something that stores a value)
50 |-- It happens in two processes
51 |-- the first process focuses soley on storing the current value (or rather our Flip Flop -- recall a flip flop stores value on edge change)
52 |-- so thats exactly like dff assignment, where when edge (rising) is set, take that value.
53 |-- of course we also have write enable which is basically like our enable bit from previous assignments. (so makeing a combo of the two is not a burden

```

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

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  -- The purpose for this component is because we will require an instruction register
7  -- that will host the opcode | rs | rt | immediate fields
8  [
9  ----- Aside Note to revisit -----
10
11  -- Note I am not sure how we will handle the fields of the data.
12  -- By the data I am referring to opcode | rs | rt | immediate
13  [
14  -- Option one: we need a driver for t, which divides the data.
15
16  -- Option two: we handle the combination of the data here as well (add them as separate input signals).
17  -- Like are those all different imports? I wouldn't think so I would assume its just 32 bits,
18  [
19  -----
20
21
22  entity SEFA_IR_REGISTER IS
23  generic (SEFA_N: integer := 32);
24  port(
25  SEFA_clk: in std_logic; -- clock
26  SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
27  SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
28  SEFA_chen: in std_logic; -- chip enable (if it is 0, the output will be undefined)
29  SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
30  SEFA_IR: out std_logic_vector (SEFA_N-1 downto 0);
31  );
32  end SEFA_IR_REGISTER;
33
34
35  architecture arch of SEFA_IR_REGISTER is
36  [
37  begin
38
39  -- Establish an IR register using the Self check lab 4b register component.
40  IR: SEFA_Register_N_VHDL port map (
41  SEFA_clk => SEFA_clk,
42  SEFA_wren => SEFA_wren,
43  SEFA_rden => SEFA_rden,
44  SEFA_chen => SEFA_chen,
45  SEFA_data => SEFA_data,
46  SEFA_q => SEFA_IR
47  );
48
49  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

```
1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  entity SEFA_RS_REGISTER IS
7  |generic (SEFA_N: integer := 32);
8  |port(
9  |    SEFA_clk: in std_logic; -- clock
10 |    SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
11 |    SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
12 |    SEFA_chen: in std_logic; -- chip enable (if it is 0, the output will be undefined)
13 |    SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
14 |    SEFA_RS: out std_logic_vector (SEFA_N-1 downto 0)
15 |);
16 |end SEFA_RS_REGISTER;
17
18
19 |architecture arch of SEFA_RS_REGISTER is
20 |
21 |begin
22 |
23 |    RS: SEFA_Register_N_VHDL port map (
24 |        SEFA_clk => SEFA_clk,
25 |        SEFA_wren => SEFA_wren,
26 |        SEFA_rden => SEFA_rden,
27 |        SEFA_chen => SEFA_chen,
28 |        SEFA_data => SEFA_data,
29 |        SEFA_q => SEFA_RS
30 |    );
31 |
32 |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

```
1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4
5  |-- purpose of this component is because we will need a register for the comparisons.
6  |-- recall for MIPS all values must be on a register (thus the 32 bits)
7
8
9
10 |entity SEFA_RT_REGISTER IS
11 |generic (SEFA_N: integer := 32);
12 |port(
13 |    SEFA_clk: in std_logic; -- clock
14 |    SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
15 |    SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
16 |    SEFA_chen: in std_logic; -- chip enable (if it is 0, the output will be undefined)
17 |    SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
18 |    SEFA_RT: out std_logic_vector (SEFA_N-1 downto 0)
19 |);
20 |end SEFA_RT_REGISTER;
21
22
23 |architecture arch of SEFA_RT_REGISTER is
24 |
25 |begin
26 |
27 |    -- Establish an RT register using the self check lab 4b register component.
28 |    RT: SEFA_Register_N_VHDL port map (
29 |        SEFA_clk => SEFA_clk,
30 |        SEFA_wren => SEFA_wren,
31 |        SEFA_rden => SEFA_rden,
32 |        SEFA_chen => SEFA_chen,
33 |        SEFA_data => SEFA_data,
34 |        SEFA_q => SEFA_RT
35 |    );
36 |
37 |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

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5  -- the purpose of this component is to hold the 16 bit immediate value.
6
7
8  entity SEFA_IMM16_REGISTER is
9  generic (SEFA_N: integer := 16); -- NOTE this is 16 bits wide, as per IMM16 (16 bits)
10 port(
11     SEFA_clk: in std_logic; -- clock
12     SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
13     SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
14     SEFA_chen: in std_logic; -- chip enable (if it is 0, the output will be undefined)
15     SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
16     SEFA_IMM16: out std_logic_vector (SEFA_N-1 downto 0)
17 );
18 end SEFA_IMM16_REGISTER;
19
20
21 architecture arch of SEFA_IMM16_REGISTER is
22
23     signal SEFA_storage : std_logic_vector (SEFA_N-1 downto 0);
24     -- This is the actual storage space of the register.
25     -- the bulk of this work will be coordinating how the data input flows into the storage space
26     -- and then to the output.
27
28 begin
29     process (SEFA_clk)
30     -- We are defining a process that is sensitive to changes in clock value.
31     -- when defining a process, the parenthesis that come after it allow us to define our sensitivity list.
32     begin
33         -- defining 32 bit wide flip flop that is rising edge triggered.
34         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
35         then SEFA_storage <= SEFA_data;
36         end if;
37     end process;
38
39     process (SEFA_rden, SEFA_chen, SEFA_storage)
40     -- this process is responsive to changes in read enable, chip enable, and the storage space.
41     -- Z is a common undefined logic value used for circuits. Using the others feature allows us to set every bit in q as undefined.
42     begin
43         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
44         then SEFA_IMM16 <= SEFA_storage;
45         elsif (SEFA_chen = '0')
46         then SEFA_IMM16 <= (others => 'Z');
47         end if;
48     end process;
49 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

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5  -- the purpose of this component is to hold the 16 bit immediate value.
6
7
8  entity SEFA_IMM26_REGISTER is
9  generic (SEFA_N: integer := 26); -- NOTE this is 16 bits wide, as per IMM26 (26 bits)
10 port(
11     SEFA_clk: in std_logic; -- clock
12     SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
13     SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
14     SEFA_chen: in std_logic; -- chip enable (if it is 0, the output will be undefined)
15     SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
16     SEFA_IMM26: out std_logic_vector (SEFA_N-1 downto 0)
17 );
18 end SEFA_IMM26_REGISTER;
19
20
21 architecture arch of SEFA_IMM26_REGISTER is
22
23     signal SEFA_storage : std_logic_vector (SEFA_N-1 downto 0);
24     -- This is the actual storage space of the register.
25     -- the bulk of this work will be coordinating how the data input flows into the storage space
26     -- and then to the output.
27
28 begin
29     process (SEFA_clk)
30     -- We are defining a process that is sensitive to changes in clock value.
31     -- when defining a process, the parenthesis that come after it allow us to define our sensitivity list.
32     begin
33         -- defining 32 bit wide flip flop that is rising edge triggered.
34         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
35         then SEFA_storage <= SEFA_data;
36         end if;
37     end process;
38
39     process (SEFA_rden, SEFA_chen, SEFA_storage)
40     -- this process is responsive to changes in read enable, chip enable, and the storage space.
41     -- Z is a common undefined logic value used for circuits. Using the others feature allows us to set every bit in q as undefined.
42     begin
43         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
44         then SEFA_IMM26 <= SEFA_storage;
45         elsif (SEFA_chen = '0')
46         then SEFA_IMM26 <= (others => 'Z');
47         end if;
48     end process;
49 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

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  entity SEFA_PC_REGISTER IS
7      generic (SEFA_N: integer := 32);
8      port(
9          SEFA_clk: in std_logic; -- clock
10         SEFA_wren: in std_logic; -- write enable (if it is 0, the stored data will not change)
11         SEFA_rden: in std_logic; -- read enable (only when it is 1, the stored data will be displayed to output)
12         SEFA_chen: in std_logic; -- chip enable (if it is 0, the output will be undefined)
13         SEFA_data: in std_logic_vector (SEFA_N-1 downto 0); -- data input
14         SEFA_PC: out std_logic_vector (SEFA_N-1 downto 0)
15     );
16 end SEFA_PC_REGISTER;
17
18
19 architecture arch of SEFA_PC_REGISTER is
20
21 begin
22
23     PC: SEFA_Register_N_VHDL port map (
24         SEFA_clk => SEFA_clk,
25         SEFA_wren => SEFA_wren,
26         SEFA_rden => SEFA_rden,
27         SEFA_chen => SEFA_chen,
28         SEFA_data => SEFA_data,
29         SEFA_q => SEFA_PC
30     );
31
32 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

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5  -- THE PURPOSE OF THIS COMPONENT IS TO SET THE FIELDS
6  -- OF DATA
7  -- THAT IS OPCODE | RS | RT | IMM16
8  -- we also might be able to avoid this file and do ir directly in IR_register
9  -- but im using ir register just for the sake of storing (read/write) data
10
11  ENTITY SEFA_IR_REGISTER_DRIVER IS
12  PORT (
13      SEFA_clk: in std_logic;
14      SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
15      SEFA_OPCODE : OUT STD_LOGIC_VECTOR(5 DOWNTO 0);
16      SEFA_RS_REGISTER_ADDRESS : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
17      SEFA_RT_REGISTER_ADDRESS : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
18      SEFA_IMMEDIATE16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
19      SEFA_IMMEDIATE26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNTO 0)
20  );
21  END SEFA_IR_REGISTER_DRIVER;
22
23  ARCHITECTURE arch OF SEFA_IR_REGISTER_DRIVER IS
24      SIGNAL SEFA_IR_REGISTER_VALUE_OUTPUT: STD_LOGIC_VECTOR(31 DOWNTO 0);
25
26  BEGIN
27
28
29      SET_AND_GET_IR_VALUE_IN_REGISTER : SEFA_IR_REGISTER port map (
30          SEFA_clk => SEFA_clk,
31          SEFA_wren => '1',
32          SEFA_rden => '1',
33          SEFA_chen => '1',
34          SEFA_data => SEFA_IR_REGISTER_VALUE,
35          SEFA_IR=> SEFA_IR_REGISTER_VALUE_OUTPUT
36      );
37
38
39      SEFA_OPCODE <= SEFA_IR_REGISTER_VALUE_OUTPUT(31 DOWNTO 26);
40      SEFA_RS_REGISTER_ADDRESS <= SEFA_IR_REGISTER_VALUE_OUTPUT(25 DOWNTO 21);
41      SEFA_RT_REGISTER_ADDRESS <= SEFA_IR_REGISTER_VALUE_OUTPUT(20 DOWNTO 16);
42      SEFA_IMMEDIATE16_VALUE <= SEFA_IR_REGISTER_VALUE_OUTPUT(15 DOWNTO 0);
43
44
45
46      -- NOTE. WHEN JUMP IS EXECUTING, THE RESUT OF THE VALUES WILL HAVE "JUNK"
47      -- WHEN EXECUTING RS RT IMM16, WELL THOSE ARE SIMPLY STORED IN IMM26 (BUT WE WONT USE IT)
48      SEFA_IMMEDIATE26_VALUE <= SEFA_IR_REGISTER_VALUE_OUTPUT(25 DOWNTO 0);
49
50
51  END arch;
```

This file is known as the IR driver. It 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

```

37  LIBRARY ieee;
38  USE ieee.std_logic_1164.all;
39
40  LIBRARY altera_mf;
41  USE altera_mf.altera_mf_components.all;
42
43  ENTITY SEFA_INSTRUCTION_MEMORY IS
44  PORT
45  (
46      address      : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
47      clock        : IN STD_LOGIC := '1';
48      data         : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
49      wren         : IN STD_LOGIC;
50      q            : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
51  );
52  END SEFA_INSTRUCTION_MEMORY;
53
54
55  ARCHITECTURE SYN OF sefa_instruction_memory IS
56
57      SIGNAL sub_wire0 : STD_LOGIC_VECTOR (31 DOWNTO 0);
58
59  BEGIN
60      q      <= sub_wire0(31 DOWNTO 0);
61
62      altsyncram_component : altsyncram
63  GENERIC MAP (
64      clock_enable_input_a => "BYPASS",
65      clock_enable_output_a => "BYPASS",
66      init_file => "SEFA_INSTRUCTION_MEMORY.mif",
67      intended_device_family => "Cyclone V",
68      lpm_hint => "ENABLE_RUNTIME_MOD=NO",
69      lpm_type => "altsyncram",
70      numwords_a => 32,
71      operation_mode => "SINGLE_PORT",
72      outdata_aclr_a => "NONE",
73      outdata_reg_a => "UNREGISTERED",
74      power_up_uninitialized => "FALSE",
75      ram_block_type => "M10K",
76      read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
77      widthad_a => 5,
78      width_a => 32,
79      width_byteena_a => 1
80  )
81  PORT MAP (
82      address_a => address,
83      clock0 => clock,
84      data_a => data,
85      wren_a => wren,
86      q_a => sub_wire0
87  );
88
89

```

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

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5  -- SEFA_REGISTER_MEMORY_FILE
6
7  ENTITY SEFA_REGISTER_MEMORY_FILE IS
8  PORT
9  (
10     SEFA_clk : IN STD_LOGIC;
11     SEFA_RS_REGISTER_ADDRESS : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
12     SEFA_RT_REGISTER_ADDRESS : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
13     SEFA_IMM16_REGISTER_VALUE : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
14     SEFA_IMM26_REGISTER_VALUE : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
15     SEFA_RS_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
16     SEFA_RT_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
17     SEFA_IMM16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
18     SEFA_IMM26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNTO 0);
19     SEFA_CURRENT_PC_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
20 );
21 END SEFA_REGISTER_MEMORY_FILE;
22
23 ARCHITECTURE arch OF SEFA_REGISTER_MEMORY_FILE IS
24
25     SIGNAL SEFA_RS_MEMORY_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
26     SIGNAL SEFA_RT_MEMORY_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
27     SIGNAL SEFA_CURRENT_PC_MEMORY_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
28
29     -- SIGNAL SEFA_RS_REGISTER_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
30     -- SIGNAL SEFA_RT_REGISTER_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
31     -- SIGNAL SEFA_CURRENT_PC_REGISTER_OUTPUT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
32
33 BEGIN
34     GET_PC_VALUE_FROM_MEMORY: SEFA_INSTRUCTION_MEMORY PORT MAP (
35         address => "00000", -- THE PC IS default address HERE! MUST BE CONSISTENT ON ALL FILES THAT PULL IT OR NEED THE ADDRESS.
36         clock => SEFA_clk,
37         data => "00000000000000000000000000000000", -- arbitrary because we are reading
38         wren => '0',
39         q => SEFA_CURRENT_PC_MEMORY_OUTPUT_VALUE
40     );
41
42     GET_RS_VALUE_FROM_MEMORY: SEFA_INSTRUCTION_MEMORY PORT MAP (
43         address => SEFA_RS_REGISTER_ADDRESS,
44         clock => SEFA_clk,
45         data => "00000000000000000000000000000000", -- NOTE THIS IS ANY ARBITRARY DATA BECAUSE WE AREN'T WRITING TO IT. ONLY READING RS!
46         wren => '0',
47         q => SEFA_RS_MEMORY_OUTPUT_VALUE
48     );
49
50     GET_RT_VALUE_FROM_MEMORY: SEFA_INSTRUCTION_MEMORY PORT MAP (
51         address => SEFA_RT_REGISTER_ADDRESS,
52         clock => SEFA_clk,
53         data => "00000000000000000000000000000000", -- NOTE THIS IS ANY ARBITRARY DATA BECAUSE WE AREN'T WRITING TO IT. ONLY READING RT!
54         wren => '0',
55         q => SEFA_RT_MEMORY_OUTPUT_VALUE
56     );
57
58     SET_AND_GET_PC_VALUE_IN_REGISTER : SEFA_PC_REGISTER port map (
59         SEFA_clk => SEFA_clk,
60         SEFA_wren => '1',
61         SEFA_rden => '1',
62         SEFA_chen => '1',
63         SEFA_data => SEFA_CURRENT_PC_MEMORY_OUTPUT_VALUE,
64         SEFA_PC => SEFA_CURRENT_PC_VALUE
65     );
66
67     SET_AND_GET_RS_VALUE_IN_REGISTER : SEFA_RS_REGISTER port map (
68         SEFA_clk => SEFA_clk,
69         SEFA_wren => '1',
70         SEFA_rden => '1',
71         SEFA_chen => '1',
72         SEFA_data => SEFA_RS_MEMORY_OUTPUT_VALUE,
73         SEFA_RS => SEFA_RS_VALUE
74     );
75
76     SET_AND_GET_RT_VALUE_IN_REGISTER : SEFA_RT_REGISTER port map (
77         SEFA_clk => SEFA_clk,
78         SEFA_wren => '1',
79         SEFA_rden => '1',
80         SEFA_chen => '1',
81         SEFA_data => SEFA_RT_MEMORY_OUTPUT_VALUE,
82         SEFA_RT => SEFA_RT_VALUE
83     );
84
85     SET_AND_GET_IMM16_VALUE_IN_REGISTER : SEFA_IMM16_REGISTER port map (
86         SEFA_clk => SEFA_clk,
87         SEFA_wren => '1',
88         SEFA_rden => '1',
89         SEFA_chen => '1',
90         SEFA_data => SEFA_IMM16_REGISTER_VALUE,
91         SEFA_IMM16 => SEFA_IMM16_VALUE
92     );
93
94     SET_AND_GET_IMM26_VALUE_IN_REGISTER : SEFA_IMM26_REGISTER port map (
95         SEFA_clk => SEFA_clk,
96         SEFA_wren => '1',
97         SEFA_rden => '1',
98         SEFA_chen => '1',
99         SEFA_data => SEFA_IMM26_REGISTER_VALUE,
100        SEFA_IMM26 => SEFA_IMM26_VALUE
101    );
102
103
```

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

```
1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4  |
5  |
6  |-- The purpose of this component is to perform bitwise comparason of bits. That is, if each bit is equal,
7  |-- return true (1), else return false (0).
8  |-- Note we use the = for both inputs to perform this.
9  |
10 |
11 |entity SEFA_Comparator_N IS
12 |    GENERIC(SEFA_N : INTEGER := 32);
13 |    PORT(
14 |        SEFA_IN0, SEFA_IN1 : IN STD_LOGIC_VECTOR(SEFA_N-1 DOWNT0 0);
15 |        SEFA_OUT : OUT STD_LOGIC
16 |    );
17 |end SEFA_Comparator_N;
18 |
19 |ARCHITECTURE arch OF SEFA_Comparator_N IS
20 |
21 |BEGIN
22 |
23 |    PROCESS(SEFA_IN0, SEFA_IN1)
24 |    begin
25 |        IF (SEFA_IN0 = SEFA_IN1) THEN -- If all bits are the same, return true
26 |            SEFA_OUT <= '1';
27 |        ELSE
28 |            SEFA_OUT <= '0'; -- return false if all bits are not the same
29 |        END IF;
30 |    END PROCESS;
31 |
32 |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

```

37  LIBRARY ieee;
38  USE ieee.std_logic_1164.all;
39
40  LIBRARY lpm;
41  USE lpm.all;
42
43  ENTITY SEFA_LPM_ADD_SUB IS
44  PORT
45  (
46      SEFA_add_sub      : IN STD_LOGIC ;
47      SEFA_dataaa       : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
48      SEFA_datab        : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
49      SEFA_overflow     : OUT STD_LOGIC ;
50      SEFA_result       : OUT STD_LOGIC_VECTOR (31 DOWNT0 0)
51  );
52  END SEFA_LPM_ADD_SUB;
53
54
55  ARCHITECTURE SYN OF sefa_lpm_add_sub IS
56
57      SIGNAL sub_wire0  : STD_LOGIC ;
58      SIGNAL sub_wire1  : STD_LOGIC_VECTOR (31 DOWNT0 0);
59
60
61
62      COMPONENT lpm_add_sub
63      GENERIC (
64          lpm_direction : STRING;
65          lpm_hint       : STRING;
66          lpm_representation : STRING;
67          lpm_type       : STRING;
68          lpm_width      : NATURAL
69      );
70      PORT (
71          add_sub : IN STD_LOGIC ;
72          dataaa  : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
73          datab   : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
74          overflow : OUT STD_LOGIC ;
75          result  : OUT STD_LOGIC_VECTOR (31 DOWNT0 0)
76      );
77      END COMPONENT;
78
79  BEGIN
80      SEFA_overflow <= sub_wire0;
81      SEFA_result  <= sub_wire1(31 DOWNT0 0);
82
83      LPM_ADD_SUB_component : LPM_ADD_SUB
84      GENERIC MAP (
85          lpm_direction => "UNUSED",
86          lpm_hint      => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
87          lpm_representation => "SIGNED",
88          lpm_type       => "LPM_ADD_SUB",
89          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

```
1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  |-- The purpose of this component is to be the adder that handles the condition when
7  |-- we continue to the direct next address (ie +4).
8
9  |ENTITY SEFA_PC_PLUS_4 IS
10 |  PORT(
11 |    SEFA_PC_OLD : IN STD_LOGIC_VECTOR(31 DOWNT0 0);
12 |    SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)
13 |  );
14 |  END SEFA_PC_PLUS_4;
15
16 |
17 |ARCHITECTURE arch OF SEFA_PC_PLUS_4 IS
18 |
19 |    SIGNAL OVERFLOW : STD_LOGIC; -- Setting an overflow singal just because it is one of the outputs.
20 |    -- However, there is no current need for it.
21 |
22 |  BEGIN
23 |
24 |    ADD4: SEFA_LPM_ADD_SUB PORT MAP (
25 |      SEFA_add_sub => '1',
26 |      SEFA_dataaa => SEFA_PC_OLD,
27 |      SEFA_datab => X"00000004", -- NOTE THAT WE HARD CODE TO GO TO NEXT INSTRUCTION!
28 |      SEFA_overflow => OVERFLOW,
29 |      SEFA_result => SEFA_PC_NEW
30 |    );
31 |
32 |  END arch;
33
34
35
```

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

SEFA_SIGN_EXTEND_IMM_16_TO_32

```
1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  |-- THE PURPOSE OF THIS COMPONENT IS TO SIGN EXTENDED IMMEDIATE FIELD FROM 16 BITS TO 32 BITS
7
8
9  |ENTITY SEFA_SIGN_EXTEND_IMM_16_TO_32 IS
10 |  PORT (
11 |    SEFA_IMM16 : IN STD_LOGIC_VECTOR(15 DOWNT0 0);
12 |    SEFA_IMM32 : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)
13 |  );
14 |  END SEFA_SIGN_EXTEND_IMM_16_TO_32;
15
16 |ARCHITECTURE arch OF SEFA_SIGN_EXTEND_IMM_16_TO_32 IS
17 |  BEGIN
18 |
19 |    SEFA_IMM32 <= "0000000000000000" & SEFA_IMM16 WHEN SEFA_IMM16(15) = '0' ELSE
20 |      "1111111111111111" & SEFA_IMM16;
21 |
22 |  END arch;
23
```

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

```
1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  |-- THE PURPOSE OF THIS COMPONENT IS TO SIGN EXTENDED IMMEDIATE FIELD FROM 26 BITS TO 32 BITS
7
8
9  |ENTITY SEFA_SIGN_EXTEND_IMM_26_TO_32 IS
10 |PORT (
11 |    SEFA_IMM26 : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
12 |    SEFA_IMM32 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
13 |);
14 |END SEFA_SIGN_EXTEND_IMM_26_TO_32;
15
16 |ARCHITECTURE arch OF SEFA_SIGN_EXTEND_IMM_26_TO_32 IS
17 |BEGIN
18 |
19 |    SEFA_IMM32 <= "000000" & SEFA_IMM26 WHEN SEFA_IMM26(25) = '0' ELSE
20 |                "111111" & SEFA_IMM26;
21 |
22 |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

```
1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  |ENTITY SEFA_BRANCHING_SIGNED_EXTENDED IS
7  |PORT(
8  |
9  |    SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
10 |    SEFA_SIGNED_16_to_32_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
11 |    SEFA_SIGNED_26_to_32_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
12 |    SEFA_SIGNED_32_EXTENDED_IMM : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
13 |);
14 |END SEFA_BRANCHING_SIGNED_EXTENDED;
15
16 |ARCHITECTURE arch OF SEFA_BRANCHING_SIGNED_EXTENDED IS
17 |
18 |BEGIN
19 |
20 |
21 |    SEFA_SIGNED_32_EXTENDED_IMM <= SEFA_SIGNED_26_to_32_EXTENDED_IMM WHEN SEFA_OPCODE = "111111"
22 |                                ELSE SEFA_SIGNED_16_to_32_EXTENDED_IMM;
23 |
24 |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

```
1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3  |use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6
7  |-- The purpose of this component is to be the adder that handles the condition when
8  |-- we continue to the direct next address (ie +4).
9  |
10 |ENTITY SEFA_PC_PLUS_IMMEDIATE_PLUS_4 IS
11 |    PORT(
12 |        SEFA_PC_PLUS_4 : IN STD_LOGIC_VECTOR(31 DOWNT0 0);
13 |        SEFA_SIGN_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNT0 0);
14 |        SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)
15 |    );
16 |END SEFA_PC_PLUS_IMMEDIATE_PLUS_4;
17
18 |ARCHITECTURE arch OF SEFA_PC_PLUS_IMMEDIATE_PLUS_4 IS
19 |    SIGNAL OVERFLOW : STD_LOGIC; -- Just a
20 |    -- signal that we might to fill all variables with .
21 |
22 |BEGIN
23 |
24 |    PC_PLUS_IMMEDIATE_PLUS_4: SEFA_LPM_ADD_SUB PORT MAP (
25 |        SEFA_add_sub => '1',
26 |        SEFA_dataaa => SEFA_PC_PLUS_4,
27 |        SEFA_datab => SEFA_SIGN_EXTENDED_IMM,
28 |        SEFA_overflow => OVERFLOW,
29 |        SEFA_result => SEFA_PC_NEW
30 |    );
31 |
32 |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

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  -- The purpose of this component is to select between PC+4 or PC+SignExtended(Imm16)+4 OR PC+SIGNEXTENDED(IMM26)+4
7  -- It must call the appropriate components that perform the addition.
8
9  -- note putting this on pause. as this can be done directly in one file.
10
11
12  -- NOTE, HERE WE HANDLE THE PC INCREMENTING LOGIC BASED ON
13  -- OPCODE AND RS, RT COMPARISON (FOR BEQ AND BNE)
14  -- OPCODE: 00000 = BEQ
15  -- OPCODE: 00100 = BNE
16  -- OPCODE: 11111 = J
17
18  ENTITY SEFA_Branching_MUX IS
19  PORT(
20    SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
21    SEFA_PC_Plus_4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
22    SEFA_PC_IMM_Plus_4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
23    SEFA_PC_SELECTOR : IN STD_LOGIC;
24    SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
25  );
26  END SEFA_Branching_MUX;
27
28  ARCHITECTURE arch OF SEFA_Branching_MUX IS
29  BEGIN
30    SEFA_PC_NEW <= SEFA_PC_IMM_Plus_4 WHEN
31      (SEFA_OPCODE = "11111" -- J
32      OR (SEFA_PC_SELECTOR = '1' AND SEFA_OPCODE = "00000") -- BEQ
33      OR (SEFA_PC_SELECTOR = '0' AND SEFA_OPCODE = "00100") -- BNE
34      )
35    ELSE SEFA_PC_Plus_4;
36  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

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE work.SEFA_BRANCHING_PACKAGE.ALL;
4
5  ENTITY SEFA_SHIFT_LEFT_2 IS
6  PORT
7  (
8    PC_INPUT : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
9    PC_OUTPUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
10 );
11 END SEFA_SHIFT_LEFT_2;
12
13 ARCHITECTURE ARCH OF SEFA_SHIFT_LEFT_2 IS
14 BEGIN
15   PC_OUTPUT <= PC_INPUT(29 DOWNTO 0) & "00";
16 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

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5
6  -- NOTE THIS FILE ESSENTIALLY REPLACES THE MUX
7  -- THIS IS BECAUSE WE WANT TO SIMPLIFY THE LOGIC OF COMPUTING THE ADDRESSES
8  -- IE COMPUTE BASED ON CONDITION, NOT COMPUTE AND THEN SELECT THE GIVEN RESULT.
9  -- THIS IS SIMILAR TO THE ORIGINAL ERROR MADE IN LAB1 BUT THEN YOU CORRECTED IT.
10 -- IT WILL REQUIRE A IF-ELSE (WITH PROCESS)
11
12 -- had to go select result. Compute on condition does not seem to be working for me. Maybe im misunderstanding what can go in if...
13
14
15 ENTITY SEFA_NAL IS
16 PORT(
17     SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
18     SEFA_PC_COND : IN STD_LOGIC;
19     SEFA_PC_OLD : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
20     SEFA_IMM16 : IN STD_LOGIC_VECTOR(15 DOWNTO 0); -- NOTE THIS IS 16 BITS AND WE NEED TO SIGN EXTEND!
21     SEFA_IMM26 : IN STD_LOGIC_VECTOR(25 DOWNTO 0); -- NOTE THIS IS 26 BITS AND WE NEED TO SIGN EXTEND!
22     SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
23 );
24 END SEFA_NAL;
25
26 ARCHITECTURE arch OF SEFA_NAL IS
27
28     -- VARIABLES TO HOLD THE EXTENDED VALUES.
29     SIGNAL SIGNED_16_to_32_EXTENDED_IMM : STD_LOGIC_VECTOR(31 DOWNTO 0);
30     SIGNAL SIGNED_26_to_32_EXTENDED_IMM : STD_LOGIC_VECTOR(31 DOWNTO 0);
31
32     SIGNAL SIGNED_32_EXTENDED_IMM : STD_LOGIC_VECTOR(31 DOWNTO 0); -- THE RESULT CHOSEN BY THE CONDITIONS
33
34     SIGNAL PC_PLUS_4 : STD_LOGIC_VECTOR(31 DOWNTO 0); -- NOTE THIS IS NEEDED FOR EITHER MODULE, SO ITS COMPUTED IN THE BEGINNING
35     SIGNAL PC_PLUS_IMM_4 : STD_LOGIC_VECTOR(31 DOWNTO 0);
36     SIGNAL SEFA_BRANCHED_PC : STD_LOGIC_VECTOR(31 DOWNTO 0);
37
38
39 BEGIN
40
41     PC_4: SEFA_PC_PLUS_4 PORT MAP (
42         SEFA_PC_OLD => SEFA_PC_OLD,
43         SEFA_PC_NEW => PC_PLUS_4
44     );
45
46     --
47     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);
48     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);
49
50     CHOSE_EXTENDED_IMM : SEFA_BRANCHING_SIGNED_EXTENDED PORT MAP (SEFA_OPCODE => SEFA_OPCODE,
51         SEFA_SIGNED_16_to_32_EXTENDED_IMM => SIGNED_16_to_32_EXTENDED_IMM,
52         SEFA_SIGNED_26_to_32_EXTENDED_IMM => SIGNED_26_to_32_EXTENDED_IMM,
53         SEFA_SIGNED_32_EXTENDED_IMM => SIGNED_32_EXTENDED_IMM
54     );
55
56     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
57
58
59 -- PROCESS (SEFA_PC_COND)
60 -- BEGIN
61 --
62 --     -- HANDLE JUMPING VIA IMMEDIATE
63 --     IF (SEFA_PC_COND = '1') THEN
64 --         SEFA_PC_NEW_OUT <= PC_PLUS_IMM_4;
65 --     ELSE -- CONTINUE TO NEXT ADDRESS (+4)
66 --         SEFA_PC_NEW_OUT <= PC_PLUS_4;
67 --     END IF;
68 --
69 -- END PROCESS;
70
71 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
72 SL2: SEFA_SHIFT_LEFT_2 PORT MAP (PC_INPUT => SEFA_BRANCHED_PC, PC_OUTPUT => SEFA_UPDATED_PC);
73
74 END arch;
```

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

SEFA_COMPUTE_NAL_FROM_IR_VALUE


```

7  ENTITY SEFA_COMPUTE_NAL_FROM_IR_VAL IS
8  PORT(
9      SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
10     SEFA_RS_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
11     SEFA_RT_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
12     SEFA_PC_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
13     SEFA_IMM16_VAL : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
14     SEFA_IMM26_VAL : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
15     SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
16 );
17 END SEFA_COMPUTE_NAL_FROM_IR_VAL;
18
19
20
21 ARCHITECTURE arch OF SEFA_COMPUTE_NAL_FROM_IR_VAL IS
22     -- comparator value
23     SIGNAL SEFA_PC_COND : STD_LOGIC;
24
25     -- TO GET RESULT SO WE CAN STORE INTO RAM.
26     SIGNAL PC_UPDATED : STD_LOGIC_VECTOR(31 DOWNTO 0);
27
28 BEGIN
29
30
31     -- AFTER GETTING RS AND RT, COMPARE THEIR VALUES TO SET CONDITION
32     COMPUTE_RS_RT_COMPARE : SEFA_Comparator_N PORT MAP (
33         SEFA_IN0 => SEFA_RS_VALUE,
34
35         SEFA_IN1 => SEFA_RT_VALUE,
36         SEFA_OUT => SEFA_PC_COND
37     );
38
39
40
41     -- ^ THIS WONT MATTER FOR JUMP!
42
43
44     -- USING CONDITION AND PC, UPDATE THE NEW PC.
45     COMPUTE_NAL : SEFA_NAL PORT MAP (
46         SEFA_OPCODE => SEFA_OPCODE,
47         SEFA_PC_COND => SEFA_PC_COND,
48         SEFA_PC_OLD => SEFA_PC_VALUE,
49         SEFA_IMM16 => SEFA_IMM16_VAL,
50         SEFA_IMM26 => SEFA_IMM26_VAL,
51         SEFA_UPDATED_PC => SEFA_UPDATED_PC
52     );
53
54 END arch;
55

```

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

```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use work.SEFA_BRANCHING_PACKAGE.all;
4
5  -- THIS IS THE MAIN DRIVER FILE.
6
7  ENTITY SEFA_MAIN IS
8  PORT (
9      SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
10     SEFA_clk : in std_logic;
11     SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
12 );
13 END SEFA_MAIN;
14
15
16 ARCHITECTURE arch OF SEFA_MAIN IS
17
18     SIGNAL SEFA_OPCODE : STD_LOGIC_VECTOR(5 DOWNTO 0);
19     SIGNAL SEFA_RS_REGISTER_ADDRESS : STD_LOGIC_VECTOR(4 DOWNTO 0);
20     SIGNAL SEFA_RT_REGISTER_ADDRESS : STD_LOGIC_VECTOR(4 DOWNTO 0);
21     SIGNAL SEFA_IMMEDIATE16_VALUE_REGISTER : STD_LOGIC_VECTOR(15 DOWNTO 0); --BEQ BNE
22     SIGNAL SEFA_IMMEDIATE26_VALUE_REGISTER : STD_LOGIC_VECTOR(25 DOWNTO 0); -- JUMP
23     SIGNAL SEFA_IMMEDIATE16_VALUE : STD_LOGIC_VECTOR(15 DOWNTO 0); --BEQ BNE
24     SIGNAL SEFA_IMMEDIATE26_VALUE : STD_LOGIC_VECTOR(25 DOWNTO 0); -- JUMP
25     SIGNAL SEFA_RS_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
26     SIGNAL SEFA_RT_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
27     SIGNAL SEFA_CURRENT_PC_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
28     SIGNAL SEFA_UPDATED_PC_VALUE : STD_LOGIC_VECTOR(31 DOWNTO 0);
29
30 BEGIN
31
32     GET_IR_VALUES: SEFA_IR_REGISTER_DRIVER PORT MAP (
33         SEFA_clk => SEFA_clk,
34         SEFA_IR_REGISTER_VALUE => SEFA_IR_REGISTER_VALUE,
35         SEFA_OPCODE => SEFA_OPCODE,
36         SEFA_RS_REGISTER_ADDRESS => SEFA_RS_REGISTER_ADDRESS,
37         SEFA_RT_REGISTER_ADDRESS => SEFA_RT_REGISTER_ADDRESS,
38         SEFA_IMMEDIATE16_VALUE => SEFA_IMMEDIATE16_VALUE,
39         SEFA_IMMEDIATE26_VALUE => SEFA_IMMEDIATE26_VALUE
40     );
41

```

```

42 GET_RS_RT_PC_VALUES: SEFA_REGISTER_MEMORY_FILE PORT MAP (
43     SEFA_clk => SEFA_clk,
44     SEFA_RS_REGISTER_ADDRESS => SEFA_RS_REGISTER_ADDRESS,
45     SEFA_RT_REGISTER_ADDRESS => SEFA_RT_REGISTER_ADDRESS,
46     SEFA_IMM16_REGISTER_VALUE => SEFA_IMMEDIATE16_VALUE,
47     SEFA_IMM26_REGISTER_VALUE => SEFA_IMMEDIATE26_VALUE,
48     SEFA_RS_VALUE => SEFA_RS_VALUE,
49     SEFA_RT_VALUE => SEFA_RT_VALUE,
50     SEFA_IMM16_VALUE => SEFA_IMMEDIATE16_VALUE_REGISTER,
51     SEFA_IMM26_VALUE => SEFA_IMMEDIATE26_VALUE_REGISTER,
52     SEFA_CURRENT_PC_VALUE => SEFA_CURRENT_PC_VALUE
53 );
54
55
56
57
58 NAL_CONTROLLER_AND_CONDITION: SEFA_COMPUTE_NAL_FROM_IR_VAL PORT MAP (
59     SEFA_OPCODE => SEFA_OPCODE,
60     SEFA_RS_VALUE => SEFA_RS_VALUE,
61     SEFA_RT_VALUE => SEFA_RT_VALUE,
62     SEFA_PC_VALUE => SEFA_CURRENT_PC_VALUE,
63     SEFA_IMM16_VAL => SEFA_IMMEDIATE16_VALUE_REGISTER,
64     SEFA_IMM26_VAL => SEFA_IMMEDIATE26_VALUE_REGISTER,
65     SEFA_UPDATED_PC => SEFA_UPDATED_PC_VALUE
66 );
67
68
69
70 SEFA_UPDATED_PC <= SEFA_UPDATED_PC_VALUE;
71
72
73
74
75
76 END arch;
77
78

```

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

```

1  |library IEEE;
2  |use IEEE.std_logic_1164.all;
3
4  |PACKAGE SEFA_BRANCHING_PACKAGE IS
5  |
6  |
7  |COMPONENT SEFA_Register_N_VHDL is
8  |    generic (SEFA_N: integer := 32);
9  |    port (
10 |        SEFA_clk: in std_logic;
11 |        SEFA_wren: in std_logic;
12 |        SEFA_rden: in std_logic;
13 |        SEFA_chen: in std_logic;
14 |        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
15 |        SEFA_q: out std_logic_vector (SEFA_N-1 downto 0)
16 |    );
17 |end COMPONENT SEFA_Register_N_VHDL;
18 |
19 |COMPONENT SEFA_RS_REGISTER IS
20 |    generic (SEFA_N: integer := 32);
21 |    port(
22 |        SEFA_clk: in std_logic;
23 |        SEFA_wren: in std_logic;
24 |        SEFA_rden: in std_logic;
25 |        SEFA_chen: in std_logic;
26 |        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
27 |        SEFA_RS: out std_logic_vector (SEFA_N-1 downto 0)
28 |    );
29 |end COMPONENT SEFA_RS_REGISTER;
30 |
31 |COMPONENT SEFA_RT_REGISTER IS
32 |    generic (SEFA_N: integer := 32);
33 |    port(
34 |        SEFA_clk: in std_logic;
35 |        SEFA_wren: in std_logic;
36 |        SEFA_rden: in std_logic;
37 |        SEFA_chen: in std_logic;
38 |        SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
39 |        SEFA_RT: out std_logic_vector (SEFA_N-1 downto 0)
40 |    );
41 |end COMPONENT SEFA_RT_REGISTER;

```

```

95  ⊞ COMPONENT SEFA_LPM_ADD_SUB IS
96      PORT
97      (
98          SEFA_add_sub      : IN STD_LOGIC ;
99          SEFA_dataaa       : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
100         SEFA_datab        : IN STD_LOGIC_VECTOR (31 DOWNT0 0);
101         SEFA_overflow      : OUT STD_LOGIC ;
102         SEFA_result        : OUT STD_LOGIC_VECTOR (31 DOWNT0 0)
103     );
104  END COMPONENT SEFA_LPM_ADD_SUB;
105
106  ⊞ COMPONENT SEFA_PC_PLUS_IMMEDIATE_PLUS_4 IS
107      PORT(
108          SEFA_PC_PLUS_4 : IN STD_LOGIC_VECTOR(31 DOWNT0 0);
109          SEFA_SIGN_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNT0 0);
110          SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)
111      );
112  END COMPONENT SEFA_PC_PLUS_IMMEDIATE_PLUS_4;
113
114  ⊞ COMPONENT SEFA_PC_PLUS_4 IS
115      PORT(
116          SEFA_PC_OLD : IN STD_LOGIC_VECTOR(31 DOWNT0 0);
117          SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNT0 0)
118      );
119  END COMPONENT SEFA_PC_PLUS_4;
120
121  ⊞ COMPONENT SEFA_Comparator_N IS
122      GENERIC(SEFA_N : INTEGER := 32);
123      PORT(
124          SEFA_IN0, SEFA_IN1 : IN STD_LOGIC_VECTOR(SEFA_N-1 DOWNT0 0);
125          SEFA_OUT : OUT STD_LOGIC
126      );
127  end COMPONENT SEFA_Comparator_N;
128
129  ⊞ COMPONENT SEFA_IR_REGISTER_DRIVER IS
130      PORT (
131          SEFA_clk: in std_logic;
132          SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNT0 0);
133          SEFA_OPCODE : OUT STD_LOGIC_VECTOR(5 DOWNT0 0);
134          SEFA_RS_REGISTER_ADDRESS : OUT STD_LOGIC_VECTOR(4 DOWNT0 0);
135          SEFA_RT_REGISTER_ADDRESS : OUT STD_LOGIC_VECTOR(4 DOWNT0 0);
136          SEFA_IMMEDIATE16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNT0 0);
137          SEFA_IMMEDIATE26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNT0 0)
138      );
139  END COMPONENT SEFA_IR_REGISTER_DRIVER;
140
141  ⊞ COMPONENT SEFA_IR_REGISTER_DRIVER IS
142      PORT (
143          SEFA_clk: in std_logic;
144          SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNT0 0);
145          SEFA_OPCODE : OUT STD_LOGIC_VECTOR(5 DOWNT0 0);
146          SEFA_RS_REGISTER_ADDRESS : OUT STD_LOGIC_VECTOR(4 DOWNT0 0);
147          SEFA_RT_REGISTER_ADDRESS : OUT STD_LOGIC_VECTOR(4 DOWNT0 0);
148          SEFA_IMMEDIATE16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNT0 0);
149          SEFA_IMMEDIATE26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNT0 0)
150      );
151  END COMPONENT SEFA_IR_REGISTER_DRIVER;

```

```

146 COMPONENT SEFA_NAL IS
147 PORT(
148     SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
149     SEFA_PC_COND : IN STD_LOGIC;
150     SEFA_PC_OLD  : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
151     SEFA_IMM16   : IN STD_LOGIC_VECTOR(15 DOWNTO 0); -- NOTE THIS IS 16 BITS AND WE NEED TO SIGN EXTEND!
152     SEFA_IMM26   : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
153     SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
154 );
155 END COMPONENT SEFA_NAL;
156
157
158
159 COMPONENT SEFA_INSTRUCTION_MEMORY IS
160 PORT
161 (
162     address      : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
163     clock        : IN STD_LOGIC := '1';
164     data         : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
165     wren         : IN STD_LOGIC ;
166     q            : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
167 );
168 END COMPONENT SEFA_INSTRUCTION_MEMORY;
169
170
171 COMPONENT SEFA_COMPUTE_NAL_FROM_IR_VAL IS
172 PORT(
173     SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
174     SEFA_RS_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
175     SEFA_RT_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
176     SEFA_PC_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
177     SEFA_IMM16_VAL : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
178     SEFA_IMM26_VAL : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
179     SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
180 );
181 END COMPONENT SEFA_COMPUTE_NAL_FROM_IR_VAL;
182
183
184
185 COMPONENT SEFA_SIGN_EXTEND_IMM_16_TO_32 IS
186 PORT (
187     SEFA_IMM16 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
188     SEFA_IMM32 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
189 );
190 END COMPONENT SEFA_SIGN_EXTEND_IMM_16_TO_32;

```



```

44  □ COMPONENT SEFA_PC_REGISTER IS
45      generic (SEFA_N: integer := 32);
46  □ port(
47      SEFA_clk: in std_logic;
48      SEFA_wren: in std_logic;
49      SEFA_rden: in std_logic;
50      SEFA_chen: in std_logic;
51      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
52      SEFA_PC: out std_logic_vector (SEFA_N-1 downto 0)
53  );
54  end COMPONENT SEFA_PC_REGISTER;
55
56  □ COMPONENT SEFA_IMM16_REGISTER IS
57      generic (SEFA_N: integer := 16);
58  □ port(
59      SEFA_clk: in std_logic;
60      SEFA_wren: in std_logic;
61      SEFA_rden: in std_logic;
62      SEFA_chen: in std_logic;
63      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
64      SEFA_IMM16: out std_logic_vector (SEFA_N-1 downto 0)
65  );
66  end COMPONENT SEFA_IMM16_REGISTER;
67
68
69  □ COMPONENT SEFA_IMM26_REGISTER IS
70      generic (SEFA_N: integer := 26);
71  □ port(
72      SEFA_clk: in std_logic;
73      SEFA_wren: in std_logic;
74      SEFA_rden: in std_logic;
75      SEFA_chen: in std_logic;
76      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
77      SEFA_IMM26: out std_logic_vector (SEFA_N-1 downto 0)
78  );
79  end COMPONENT SEFA_IMM26_REGISTER;
80
81
82  □ COMPONENT SEFA_IR_REGISTER IS
83      generic (SEFA_N: integer := 32);
84  □ port(
85      SEFA_clk: in std_logic;
86      SEFA_wren: in std_logic;
87      SEFA_rden: in std_logic;
88      SEFA_chen: in std_logic;
89      SEFA_data: in std_logic_vector (SEFA_N-1 downto 0);
90      SEFA_IR: out std_logic_vector (SEFA_N-1 downto 0)
91  );
92  end COMPONENT SEFA_IR_REGISTER;
93
94

```

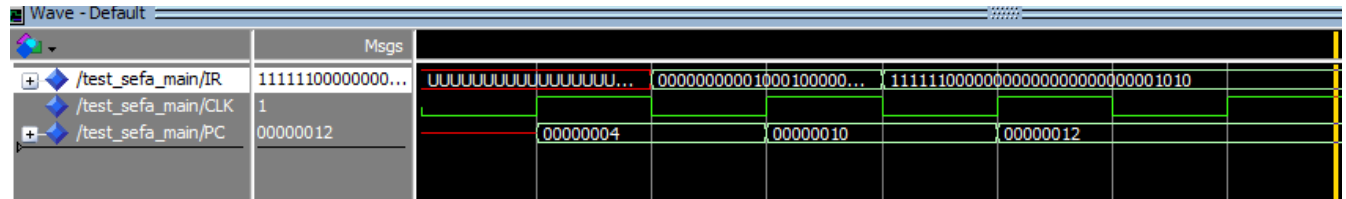
```

192  ⊞ COMPONENT SEFA_SIGN_EXTEND_IMM_26_TO_32 IS
193  ⊞ PORT (
194      SEFA_IMM26 : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
195      SEFA_IMM32 : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
196  );
197  END COMPONENT SEFA_SIGN_EXTEND_IMM_26_TO_32;
198
199
200
201  ⊞ COMPONENT SEFA_SHIFT_LEFT_2 IS
202  ⊞ PORT
203  ⊞ (
204      PC_INPUT : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
205      PC_OUTPUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
206  );
207  END COMPONENT SEFA_SHIFT_LEFT_2;
208
209
210
211  ⊞ COMPONENT SEFA_BRANCHING_SIGNED_EXTENDED IS
212  ⊞ PORT(
213
214      SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
215      SEFA_SIGNED_16_to_32_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
216      SEFA_SIGNED_26_to_32_EXTENDED_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
217      SEFA_SIGNED_32_EXTENDED_IMM : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
218  );
219  END COMPONENT SEFA_BRANCHING_SIGNED_EXTENDED;
220
221
222
223  ⊞ COMPONENT SEFA_Branching_MUX IS
224  ⊞ PORT(
225      SEFA_OPCODE : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
226      SEFA_PC_Plus_4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
227      SEFA_PC_IMM_Plus_4 : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
228      SEFA_PC_SELECTOR : IN STD_LOGIC;
229      SEFA_PC_NEW : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
230  );
231  END COMPONENT SEFA_Branching_MUX;
232
233
234  ⊞ COMPONENT SEFA_REGISTER_MEMORY_FILE IS
235  ⊞ PORT
236  ⊞ (
237      SEFA_clk : IN STD_LOGIC;
238      SEFA_RS_REGISTER_ADDRESS : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
239      SEFA_RT_REGISTER_ADDRESS : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
240      SEFA_IMM16_REGISTER_VALUE : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
241      SEFA_IMM26_REGISTER_VALUE : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
242      SEFA_RS_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
243      SEFA_RT_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
244      SEFA_IMM16_VALUE : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
245      SEFA_IMM26_VALUE : OUT STD_LOGIC_VECTOR(25 DOWNTO 0);
246      SEFA_CURRENT_PC_VALUE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
247  );
248  END COMPONENT SEFA_REGISTER_MEMORY_FILE;
249
250
251  ⊞ COMPONENT SEFA_MAIN IS
252  ⊞ PORT (
253      SEFA_IR_REGISTER_VALUE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
254      SEFA_clk : in std_logic;
255      SEFA_UPDATED_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
256  );
257  END COMPONENT SEFA_MAIN;
258
259
260
261
262  END SEFA_BRANCHING_PACKAGE;

```

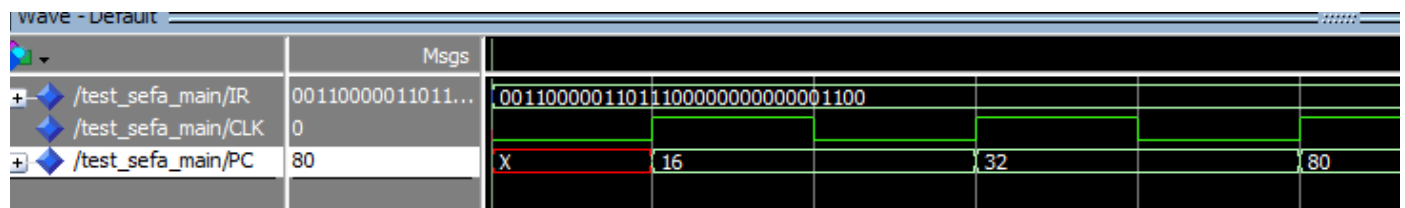
This is the project package file with all of the components.

SEFA_TEST_MAIN

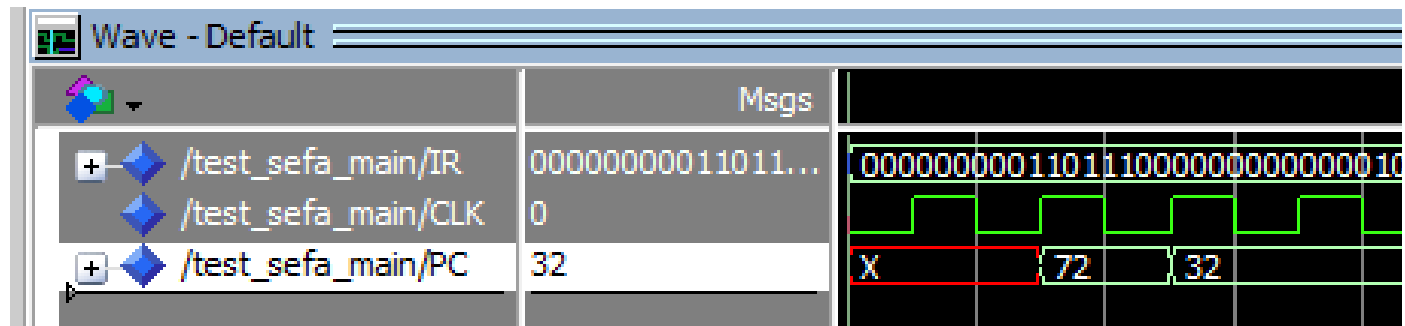


BEQ : IMM

J



BNE: IMM



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.