```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_signed.all;
use work.output_array_types.all;


entity CPU_final is
port (
 clk,reset,enable: in std_logic;
 pc_out, alu_out: out std_logic_vector(31 downto 0)
);
end CPU_final;

architecture Behavioral of CPU_final is
signal pc_current,pc_next,pc_next2: std_logic_vector(31 downto
0) := (others => '0');
signal instruction: std_logic_vector(31 downto 0) := (others =>
'0');

signal read_data_1 : std_logic_vector(31 downto 0) := (others =>
'0');
signal read_data_2 : std_logic_vector(31 downto 0) := (others =>
'0');


signal sel: STD_LOGIC := '0';
signal sel1: STD_LOGIC := '0';
signal data_out: STD_LOGIC_VECTOR(4 downto 0) := (others => '0');

signal control_signals_out: std_logic_vector(7 downto 0) :=
(others => '0');

signal s32 : std_logic_vector(31 downto 0) := (others => '0');

signal JMP: STD_LOGIC;
signal JR: STD_LOGIC;
signal ACU: std_logic_vector(2 downto 0):= (others => '0');
signal OCU: std_logic_vector(2 downto 0):= (others => '0');

signal en1: std_logic := '1';
signal left_shifter_output: std_logic_vector(31 downto 0);
signal adder_output1: std_logic_vector(31 downto 0):= (others =>
'0');
signal zero_out:std_logic:='0';
signal mux_out_32: std_logic_vector(31 downto 0);
signal mux_sel:std_logic:='0';

signal data_out1: std_logic_vector(31 downto 0);
signal dataout2: std_logic_vector(2 downto 0);

signal aluout: std_logic_vector(31 downto 0);
```

```
signal read_data: STD_LOGIC_VECTOR(31 downto 0);

signal write_back_out: std_logic_vector(31 downto 0);

signal mem_unit_in_fd: mem_array;
signal mem_unit_out_fd: mem_array;

signal mem_unit_in_de: mem_array;
signal mem_unit_out_de: mem_array;

signal mem_unit_in_em: mem_array;
signal mem_unit_out_em: mem_array;

signal mem_unit_in_mw: mem_array;
signal mem_unit_out_mw: mem_array;

signal mem_unit_in_wf: mem_array;
signal mem_unit_out_wf: mem_array;

begin

process(clk,reset)
begin
 if(reset='1') then
  pc_current <= x"00000000";
      pc_out<=pc_current;
 elsif(rising_edge(clk)) then
  pc_current <= pc_next2;
pc_out<=pc_current;

 end if;
end process;

--------FETCH PHASE STARTS HERE-------------------------
pc_next <= pc_current + x"00000001";


--Instruction memory
Instruction_Memory: entity work.Instruction_Memory
        port map
       (
        pc=> pc_current,
        instruction => instruction
        );
--end of Instruction memory

--------END OF FETCH PHASE--------------------------------



mem_unit_in_fd(0) <= instruction;
mem_unit_in_fd(1) <= pc_next;
```

```
pipeline_register1: entity work.pipeline_register
     port map(
          clk => clk,
          mem_unit => mem_unit_in_fd,
          mem_unit_out => mem_unit_out_fd
     );


--------DECODE PHASE STARTS HERE--------------------------



--control unit


control_unit: entity work.control_unit
port map(
 opcode=> mem_unit_out_fd(0)(31 downto 26),
 control_signals_out => control_signals_out
 );

sel <= control_signals_out(0);

--end of control unit

--mux_2by1_5_bit
mux_2by1_5_bit: entity work.mux_2by1_5_bit
port map(
          data_in_lsb => mem_unit_out_fd(0)(20 downto 16),
          data_in_msb => mem_unit_out_fd(0)(15 downto 11),
          sel => sel,
          data_out => data_out
     );
--end of mux_2by1_5_bit


--register file

rf32: entity work.rf32
port map(
 rst_n => reset,
 write_en => mem_unit_out_wf(2)(7),
 write_addr => mem_unit_out_wf(1)(4 downto 0),
 write_data => mem_unit_out_wf(0),
 read_addr_1 => mem_unit_out_fd(0)(25 downto 21),
 read_data_1 => read_data_1,
 read_addr_2 => mem_unit_out_fd(0)(20 downto 16),
 read_data_2 => read_data_2
);
--end of register file
```

```
--sign extender

sign16x32: entity work.sign16x32
port map(
    s16 => mem_unit_out_fd(0)(15 downto 0),
    s32 => s32
);
--end of sign extender

--OP_Control

OP_Control:entity work.op_control
port map(
      opcode => mem_unit_out_fd(0)(31 downto 26),
      OCU => OCU,
      en => enable
);

-- end of OP_Control

--ALU_Control

ALU_Control:entity work.ALU_Control
port map(
      funct => mem_unit_out_fd(0)(5 downto 0),
      ACU => ACU,
      en => enable,
      JMP => JMP
);
-- end of ALU_Control

sel1<= (not mem_unit_out_fd(0)(31)) and (not mem_unit_out_fd(0)
(30))and (not mem_unit_out_fd(0)(29))and (not mem_unit_out_fd(0)
(28))and (not mem_unit_out_fd(0)(27))and (mem_unit_out_fd(0)
(26));

--mux_2by1_3_bit
mux_2by1_3_bit: entity work.mux_2by1_3_bit
port map(
           data_in_lsb => OCU,
           data_in_msb => ACU,
           sel => sel1,
           data_out => dataout2
      );
--end of mux_2by1_3_bit

--Left shifter
Left_shifter: entity work.Left_shifter
port map(
      A => s32,
      en => enable,
```

```
      shifted => left_shifter_output
);
--end left_Shifter

mux_2by1_32_bit1: entity work.mux_2by1_32_bit
port map(
      data_in_lsb =>read_data_2,
      data_in_msb => s32,
      sel => control_signals_out(6),
      data_out => data_out1
);


--adder
adder_output1<= left_shifter_output+mem_unit_out_fd(1);

--end of adder

zero_detect: entity work.zero_detect
port map(
      A=>read_data_1,
      B=>data_out1,
      outzero=>zero_out
);
--end of zero detector

mux_sel<=zero_out and control_signals_out(1);

mux_2by1_32_bit2: entity work.mux_2by1_32_bit
port map(
      data_in_lsb =>pc_next,
      data_in_msb => adder_output1,
      sel => mux_sel,
      data_out => mux_out_32
);

JR <= JMP and (not mem_unit_out_fd(0)(31)) and (not
mem_unit_out_fd(0)(30))and (not mem_unit_out_fd(0)(29))and (not
mem_unit_out_fd(0)(28))and (not mem_unit_out_fd(0)(27))and
(mem_unit_out_fd(0)(26));
mux_2by1_32_bit3: entity work.mux_2by1_32_bit
port map(
      data_in_lsb =>mux_out_32,
      data_in_msb => read_data_1,
      sel => JR,
      data_out => pc_next2
);

----pc updated
```

```
----END OF DECODE
PHASE-----------------------------------------------------

mem_unit_in_de(0)<= read_data_1;
mem_unit_in_de(1)<= data_out1;
mem_unit_in_de(2)<= read_data_2;
mem_unit_in_de(3)<= (28 downto 0=>'0')&dataout2;
mem_unit_in_de(4)<=(23 downto 0=>'0')&control_signals_out;
mem_unit_in_de(5)<=(26 downto 0=>'0')&data_out;

pipeline_register2: entity work.pipeline_register
      port map(
            clk => clk,
            mem_unit => mem_unit_in_de,
            mem_unit_out => mem_unit_out_de
      );



-------EXECUTE PHASE STARTS HERE-------------------------

alu_clk: entity work.alu_clk
port map(
      InRegA =>mem_unit_out_de(0),
      InRegB => mem_unit_out_de(1),
      InOp => mem_unit_out_de(3)(2 downto 0),
      OutReg => aluout
);
alu_out<=aluout;
------END OF EXECUTE PHASE----------------------------------

mem_unit_in_em(0)<=aluout;
mem_unit_in_em(1)<=mem_unit_out_de(2);
mem_unit_in_em(2)<=mem_unit_out_de(4);
mem_unit_in_em(3)<=mem_unit_out_de(5);

pipeline_register3: entity work.pipeline_register
      port map(
            clk => clk,
            mem_unit => mem_unit_in_em,
            mem_unit_out => mem_unit_out_em
      );

-------MEM ACCESS PHASE STARTS HERE----------------------------

data_memory: entity work.data_memory
port map(
      mem_rst => reset,
      mem_write => mem_unit_out_em(2)(5),
      mem_read => mem_unit_out_em(2)(2),
      mem_enable => enable,
      address => mem_unit_out_em(0),
```

```
      write_data => mem_unit_out_em(1),
      read_data => read_data
);


------END OF MEMORY ACCESS PHASE--------------------------------

mem_unit_in_mw(0)<=read_data;
mem_unit_in_mw(1)<=mem_unit_out_em(0);
mem_unit_in_mw(2)<=mem_unit_out_em(2);
mem_unit_in_mw(3)<=mem_unit_out_em(3);

pipeline_register4: entity work.pipeline_register
      port map(
            clk => clk,
            mem_unit => mem_unit_in_mw,
            mem_unit_out => mem_unit_out_mw
      );

-------WRITE BACK PHASE STARTS
HERE-----------------------------

mux_2by1_32_bit4: entity work.mux_2by1_32_bit
port map(
      data_in_lsb => mem_unit_out_mw(1),
      data_in_msb => mem_unit_out_mw(0),
      sel => mem_unit_out_mw(2)(3),
      data_out => write_back_out
);

------END OF WRITE BACK PHASE---------------------------------

mem_unit_in_wf(0)<=write_back_out;
mem_unit_in_wf(1)<=mem_unit_out_mw(3);
mem_unit_in_wf(2)<=mem_unit_out_mw(2);
pipeline_register5: entity work.pipeline_register
      port map(
            clk => clk,
            mem_unit => mem_unit_in_wf,
            mem_unit_out => mem_unit_out_wf
      );

end behavioral;
```