

1. Lab Objective:

The objective of this lab is to integrate the previously designed data path and control units into a complete CPU system, incorporating a reset circuit to ensure proper initialization and operation. Students will implement the reset circuit, synchronizing its operation with the CPU clock cycles, and then assemble the final CPU by interconnecting all components. The completion of this lab project enables the implementation of features described in the CPU specification document and requires thorough testing and demonstration of the functioning CPU system.

2. Experiment Details:

<pre> 36 LIBRARY ieee; 37 USE ieee.std_logic_1164.all; 38 39 LIBRARY altera_mf; 40 USE altera_mf.all; 41 42 ENTITY system_memory IS 43 PORT 44 (45 address : IN STD_LOGIC_VECTOR (5 DOWNTO 0); 46 clock : IN STD_LOGIC := '1'; 47 data : IN STD_LOGIC_VECTOR (31 DOWNTO 0); 48 wren : IN STD_LOGIC; 49 q : OUT STD_LOGIC_VECTOR (31 DOWNTO 0) 50); 51 END system_memory; 52 53 54 ARCHITECTURE SYN OF system_memory IS 55 56 SIGNAL sub_wire0 : STD_LOGIC_VECTOR (31 DOWNTO 0); 57 58 59 60 COMPONENT altsyncram 61 GENERIC (62 clock_enable_input_a : STRING; 63 clock_enable_output_a : STRING; 64 init_file : STRING; 65 intended_device_family : STRING; 66 lpm_hint : STRING; 67 lpm_type : STRING; 68 numwords_a : NATURAL; 69 operation_mode : STRING; 70 71 outdata_aclr_a : STRING; 72 outdata_reg_a : STRING; 73 power_up_uninitialized : STRING; 74 widthad_a : NATURAL; 75 width_a : NATURAL; 76 width_bytenea_a : NATURAL 77); 78 PORT (79 address_a : IN STD_LOGIC_VECTOR (5 DOWNTO 0); 80 clock0 : IN STD_LOGIC ; 81 data_a : IN STD_LOGIC_VECTOR (31 DOWNTO 0); 82 wren_a : IN STD_LOGIC ; 83 q_a : OUT STD_LOGIC_VECTOR (31 DOWNTO 0) 84); 85 86 END COMPONENT; 87 88 BEGIN 89 q <= sub_wire0(31 DOWNTO 0); 90 91 altsyncram_component : altsyncram 92 GENERIC MAP 93 clock_enable_input_a => "BYPASS", 94 clock_enable_output_a => "BYPASS", 95 init_file => "system_memory.mif", 96 intended_device_family => "Cyclone II", 97 lpm_hint => "ENABLE_RUNTIME_MOD=NO", 98 lpm_type => "altsyncram", 99 numwords_a => 64, 100 operation_mode => "SINGLE_PORT", 101 outdata_aclr_a => "NONE", 102 outdata_reg_a => "CLOCK0", 103 power_up_uninitialized => "FALSE", 104 widthad_a => 6, 105 width_a => 32, 106 width_bytenea_a => 1 107) 108 PORT MAP (109 address_a => address, 110 clock0 => clock, 111 data_a => data, 112 wren_a => wren, 113 q_a => sub_wire0 114); 115 116 END SYN; </pre>	<pre> 1 library ieee; 2 use ieee.std_logic_1164.all; 3 use ieee.std_logic_arith.all; 4 use ieee.std_logic_unsigned.all; 5 6 entity pc is 7 port(8 clk : in std_logic; 9 id : in std_logic; 10 ld : in std_logic; 11 inc : in std_logic; 12 d : in std_logic_vector(31 downto 0); 13 q : out std_logic_vector(31 downto 0) 14); 15 end pc; 16 17 architecture Behavior of pc is 18 component add 19 port(20 A : in std_logic_vector(31 downto 0); 21 B : out std_logic_vector(31 downto 0) 22); 23 end component; 24 25 component mux2to1 26 port(s : in std_logic; 27 w0, w1 : in std_logic_vector(31 downto 0); 28 f : out std_logic_vector(31 downto 0) 29); 30 end component; 31 32 component register32 33 port(34 d : in std_logic_vector(31 downto 0); 35 id : in std_logic; 36 clk : in std_logic; 37 q : out std_logic_vector(31 downto 0) 38); 39 end component; 40 41 signal add_out : std_logic_vector(31 downto 0); 42 signal mux_out : std_logic_vector(31 downto 0); 43 signal q_out : std_logic_vector(31 downto 0); 44 begin 45 add : add port map(q_out, add_out); 46 mux : mux2to1 port map(inc, d, add_out, mux_out); 47 reg0: register32 port map (mux_out, id, clk, q_out); 48 q <= q_out; 49 end Behavior; </pre>
Figure 1: system_memory VHDL code	Figure 2: register32 VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity mux4to1 is
5 port(
6     s : in std_logic_vector(1 downto 0);
7     X1,X2,X3,X4 : in std_logic_vector(31 downto 0);
8     f : out std_logic_vector(31 downto 0));
9
10 end mux4to1;
11
12 architecture Behavior of mux4to1 is
13 begin
14     with s select
15         f<= X1 when "00",
16             X2 when "01",
17             X3 when "10",
18             X4 when "11";
19 end Behavior;

```

Figure 4: mux4to1 VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity mux2to1 is
5 port(
6     s : in std_logic;
7     w0, w1 : in std_logic_vector(31 downto 0);
8     f : out std_logic_vector(31 downto 0));
9
10 end mux2to1;
11
12 architecture Behavior of mux2to1 is
13 begin
14     with s select
15         f<= w0 when '0',
16             w1 when others;
17 end behavior;

```

Figure 5: mux2to1 VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity fulladd is
5 port(
6     Cin, x, y : in std_logic;
7     s, Cout : out std_logic);
8
9 end fulladd;
10
11 architecture Behavior of fulladd is
12 begin
13     s <= x xor y xor Cin;
14     Cout <= (x and y) or (Cin and x) or (Cin and y);
15 end Behavior;

```

Figure 6: fulladd VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity data_mem is
6 port(
7     clk : in std_logic;
8     addr : in unsigned(7 downto 0);
9     data_in : in std_logic_vector(31 downto 0);
10    wen : in std_logic;
11    en : in std_logic;
12    data_out : out std_logic_vector(31 downto 0));
13
14 end data_mem;
15
16 architecture Behavior of data_mem is
17 type RAM is array (0 to 255) of std_logic_vector(31 downto 0);
18 signal DATAMEM : RAM;
19
20 begin
21 process(clk, en, wen)
22 begin
23 if(clk'event and clk='1') then
24     if(en = '0') then
25         data_out <= (others => '0');
26     else
27         if(wen = '1') then
28             data_out <= DATAMEM(to_integer(addr));
29         end if;
30         if(wen='1') then
31             DATAMEM(to_integer(addr)) <= data_in;
32         end if;
33     end if;
34 end if;
35 end process;
36 end Behavior;

```

Figure 7: data_mem VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 use ieee.std_logic_UNSIGNED.all;
5
6 entity ALU is
7 port(
8     a : in std_logic_vector(31 downto 0);
9     b : in std_logic_vector(31 downto 0);
10    op : in std_logic_vector(2 downto 0);
11    result : out std_logic_vector(31 downto 0);
12    zero : out std_logic;
13    Cout : out std_logic);
14
15 end ALU;
16
17 architecture Behavior of ALU is
18 component adder32
19 port(
20     Cin : in std_logic;
21     X,Y : in std_logic_vector(31 downto 0);
22     S : out std_logic_vector(31 downto 0);
23     Cout : out std_logic;
24 );
25 end component;
26
27 begin
28     signal result_s : std_logic_vector(31 downto 0):=(others=>'0');
29     signal result_a : std_logic_vector(31 downto 0):=(others=>'0');
30     signal result_b : std_logic_vector(31 downto 0):=(others=>'0');
31     signal cout_zi : std_logic:=';
32     signal cout_add : std_logic:=';
33     signal cout_sub : std_logic:=';
34     signal zero_si : std_logic:=';
35
36     begin
37         add0 : adder32 port map (op(2), a, b, result_add, cout_add);
38         sub : adder32 port map (op(2), a, not b, result_sub, cout_sub);
39
40     process (a, b, op)
41     begin
42         case (op) is

```

Figure 8: ALU VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity adder32 is
5 port(
6     Cin : in std_logic;
7     X,Y : in std_logic_vector(31 downto 0);
8     S : out std_logic_vector(31 downto 0);
9     Cout : out std_logic);
10
11 end adder32;
12
13 architecture Behavior of adder32 is
14 component adder21
15 port(
16     Cin : in std_logic;
17     X,Y : in std_logic_vector(15 downto 0);
18     S : out std_logic_vector(15 downto 0);
19     Cout : out std_logic);
20
21 end component;
22
23 begin
24     signal Cin : std_logic;
25     signal X,Y : std_logic_vector(31 downto 0);
26     signal S : std_logic_vector(31 downto 0);
27     signal Cout : std_logic;
28
29     add0 : adder32 port map (op(2), a, b, result_add, cout_add);
30     sub : adder32 port map (op(2), a, not b, result_sub, cout_sub);
31
32     process (a, b, op)
33     begin
34         case (op) is

```

Figure 9: adder32 VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity adder16 is
5   port(
6     Cin : in std_logic;
7     X,Y : in std_logic_vector(15 downto 0);
8     S,Cout : out std_logic_vector(15 downto 0);
9   );
10 end adder16;
11
12 architecture Behavior of adder16 is
13 begin
14   component adder4 is
15   port(
16     Cin : in std_logic;
17     X,Y : in std_logic_vector(3 downto 0);
18     S,Cout : out std_logic_vector(3 downto 0);
19   );
20   end component;
21
22   signal C : std_logic_vector(1 to 12);
23
24 begin
25   stage0: adder4 port map (Cin, X(0), Y(0), S(0), Cout(0));
26   stage1: adder4 port map (C(0), X(1), Y(1), S(1), Cout(1));
27   stage2: adder4 port map (C(1), X(2), Y(2), S(2), Cout(2));
28   stage3: adder4 port map (C(2), X(3), Y(3), S(3), Cout);
29 end Behavior;

```

Figure 10: adder16 VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity adder4 is
5   port(
6     Cin : in std_logic;
7     X,Y : in std_logic_vector(3 downto 0);
8     S,Cout : out std_logic_vector(3 downto 0);
9   );
10 end adder4;
11
12 architecture Behavior of adder4 is
13 begin
14   component fulladd is
15   port(
16     Cin, x, y : in std_logic;
17     S,Cout : out std_logic;
18   );
19   end component;
20
21   signal C : std_logic_vector (1 to 3);
22
23 begin
24   stage0: fulladd port map (Cin, X(0), Y(0), S(0), Cout(0));
25   stage1: fulladd port map (C(0), X(1), Y(1), S(1), Cout(1));
26   stage2: fulladd port map (C(1), X(2), Y(2), S(2), Cout(2));
27   stage3: fulladd port map (C(2), X(3), Y(3), S(3), Cout);
28 end Behavior;

```

Figure 11: adder4 VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity data_path is
7   port(
8     --Clock Signals
9     Clk, mClk : in std_logic;
10    --Memory Signals
11    WEN, EN : in std_logic;
12
13    --Register Control Signals
14    Clr_A, Ld_A : in std_logic;
15    Clr_B, Ld_B : in std_logic;
16    Clr_C, Ld_C : in std_logic;
17    Clr_D, Ld_D : in std_logic;
18    Clr_FC, Ld_FC : in std_logic;
19    Clr_IR, Ld_IR : in std_logic;
20
21    --Register Outputs
22    Out_A : out std_logic_vector(31 downto 0);
23    Out_B : out std_logic_vector(31 downto 0);
24    Out_C : out std_logic;
25    Out_2 : out std_logic;
26    Out_FC : out std_logic_vector(31 downto 0);
27    Out_IR : out std_logic_vector(31 downto 0);
28
29    --Special PC Inputs
30    Inc_FC : in std_logic;
31
32    --Address and Data Bus Signals
33    ADDR_OUT : out std_logic_vector(31 downto 0);
34    DATA_IN : in std_logic_vector(31 downto 0);
35    DATA_BUS, MEM_OUT, MEM_IN : out std_logic_vector(31 downto 0);
36    MEM_ADDR : out unsigned(7 downto 0);
37
38    --MUX Controls
39    DATA_MUX : in std_logic_vector(1 downto 0);
40    MUX_MN : in std_logic;
41    MUX_AN, B_MUX : in std_logic;
42
43    A_MUX, B_MUX : in std_logic;
44    IM_MUX : in std_logic;
45    IM_MUXN : in std_logic_vector(1 downto 0);
46
47    --ALU Operations
48    ALU_Op : in std_logic_vector(2 downto 0);
49  );
50
51 end entity;
52
53 architecture Behavior of data_path is
54 begin
55   component data_mem is
56   port(
57     clk : in std_logic;
58     addr : in unsigned(7 downto 0);
59     data_in : in std_logic_vector(31 downto 0);
60     wen : in std_logic;
61     en : in std_logic;
62     data_out : out std_logic_vector(31 downto 0)
63   );
64   end component;
65
66   component register32 is
67   port (
68     d : in std_logic_vector(31 downto 0) ;
69     id : in std_logic ;
70     clk : in std_logic ;
71     q : out std_logic_vector(31 downto 0) ;
72   );
73   end component;
74
75   component pc is
76   port (
77     clk : in std_logic ;
78     id : in std_logic ;
79     inc : in std_logic ;
80     d : in std_logic_vector(31 downto 0) ;
81     q : out std_logic_vector(31 downto 0) );
82   end component;
83
84   component LZE is
85   port(
86     LZE_in : in std_logic_vector(31 downto 0);
87     LZE_out : out std_logic_vector(31 downto 0)
88   );
89   end component;
90
91   component UZE is
92   port(
93     UZE_in : in std_logic_vector(31 downto 0);
94     UZE_out : out std_logic_vector(31 downto 0)
95   );
96   end component;
97
98   component RED is
99   port(
100    RED_in : in std_logic_vector(31 downto 0);
101    RED_out : out unsigned(7 downto 0)
102  );
103   end component;
104
105   component mux2to1 is
106   port (
107     s : in std_logic ;
108     w0, w1 : std_logic_vector(31 downto 0) ;
109     f : out std_logic_vector(31 downto 0) );
110   end component;
111
112   component mux4to1 is
113   port(
114     s : in std_logic_vector(1 downto 0) ;
115     w1, w2, w3, w4 : in std_logic_vector(31 downto 0) ;
116     f : out std_logic_vector(31 downto 0)
117   );
118   end component;
119
120   component alu is

```

```

120  component alu is
121    port(
122      a, b : in std_logic_vector(31 downto 0);
123      op : in std_logic_vector(2 downto 0);
124      result : out std_logic_vector(31 downto 0);
125      zero, cout : out std_logic );
126    end component;
127
128  signal IR_OUT : std_logic_vector(31 downto 0);
129  signal data_bus_s : std_logic_vector(31 downto 0);
130  signal LZE_out_A : std_logic_vector(31 downto 0);
131  signal LZE_out_A_mux : std_logic_vector(31 downto 0);
132  signal LZE_out_B_mux : std_logic_vector(31 downto 0);
133  signal LZE_out_B : std_logic_vector(31 downto 0);
134  signal reg_A_out : std_logic_vector(31 downto 0);
135  signal reg_B_out : std_logic_vector(31 downto 0);
136  signal reg_A_out : std_logic_vector(31 downto 0);
137  signal reg_B_out : std_logic_vector(31 downto 0);
138  signal LD_A_out : std_logic_vector(31 downto 0);
139  signal data_mem_out : std_logic_vector(31 downto 0);
140  signal UZI_IN_MNXL_out : std_logic_vector(31 downto 0);
141  signal IM_MNXL_out : std_logic_vector(31 downto 0);
142  signal LD_B_out : std_logic_vector(31 downto 0);
143  signal IM_HNZL_out : std_logic_vector(31 downto 0);
144  signal ALD_out : std_logic_vector(31 downto 0);
145  signal zero_flag : std_logic;
146  signal LD_out : std_logic;
147  signal temp : std_logic_vector(30 downto 0) := (others => '0');
148  signal out_pc_sig : std_logic_vector(31 downto 0);
149
150 begin
151   IR: register32 port map(
152     data_bus_s,
153     ld_IR,
154     ClrIR,
155     Clk,
156     IR_OUT
157   );
158
159   LZE_PC: LZE port map(
160     IR_OUT,
161     LZE_out_PC
162   );
163
164   PC0: pc port map(
165     CLRPC,
166     Clk,
167     ld_PC,
168     INC_PC,
169     LZE_out_PC,
170     out_pc_sig
171   );
172
173   LZE_A_Mux: LZE port map(
174     IR_OUT,
175     LZE_out_A_mux
176   );
177
178   A_Mux0: mux2tol port map(
179     A_MUX,
180     data_bus_s,
181     LZE_out_A_mux,
182     A_mux_out
183   );
184
185   Reg_A: register32 port map(
186     A_mux_out,
187     ld_A,
188     Clr_A,
189     Clk,
190     reg_A_out
191   );
192
193   LZE_B_Mux: LZE port map(
194     IR_OUT,
195     LZE_out_B_mux
196   );
197
198   B_Mux0: mux2tol port map(

```

```

198      B_Mux0: mux2tol port map(
199        B_MUX,
200        data_bus_s,
201        LZE_out_B_mux,
202        B_mux_out
203      );
204
205      Reg_B: register32 port map(
206        B_mux_out,
207        Ld_B,
208        Clz_B,
209        Clk,
210        reg_B_out
211      );
212
213      Reg_Mux: mux2tol port map(
214        REG_MUX,
215        Reg_A_out,
216        Reg_B_out,
217        Reg_Mux_out
218      );
219
220      RED_Data_Mem: RED port map(
221        IR_OUT,
222        RED_out_data_mem
223      );
224
225      Data_Mem0: data_mem port map(
226        mClk,
227        RED_out_data_mem,
228        Reg_Mux_out,
229        WEN,
230        EN,
231        data_mem_out
232      );
233
234      UZE_IM_MUX1: UZE port map(
235        IR_OUT,
236        UZE_IM_MUX1_out
237      );
238
239      IM_MUX1a: mux2tol port map(
240        IM_MUX1,
241        req_A_out,
242        UZE_IM_MUX1_out,
243        IM_MUX1_out
244      );
245
246      LZE_IM_MUX2: LZE port map(
247        IR_OUT,
248        LZE_IM_MUX2_out
249      );
250
251      IM_MUX2a: mux4tol port map(
252        IM_MUX2,
253        req_B_out,
254        LZE_IM_MUX2_out, (temp & '1'), (others =>'0'),
255        IM_MUX2_out
256      );
257
258      ALU0: ALU port map(
259        IM_MUX1_out,
260        IM_MUX2_out,
261        ALU_Op,
262        ALU_out,
263        zero_flag,
264        carry_flag
265      );
266
267      DATA_MUX0: mux4tol port map(
268        DATA_MUX,
269        DATA_IN,
270        data_mem_out,
271        ALU_out,
272        (others => '0'),
273        data_bus_s
274      );
275
276      DATA_BUS <= data_bus_s;
277      OUT_A <= reg_A_out;
278      OUT_B <= reg_B_out;
279      OUT_IR <= IR_OUT;
280      ADDR_OUT <= out_pc_sig;
281      OUT_PC <= out_pc_sig;
282
283      MEM_ADDR <= RED_out_data_mem;
284      MEM_IN <= Reg_Mux_out;
285      MEM_OUT <= data_mem_out;
286
287    end Behavior;

```

Figure 12: data_path VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 ENTITY cpu_test_sim IS
5 PORT(
6   memClk : in std_logic;
7   memDataIn : in std_logic_vector(31 downto 0);
8   memDataOut : out std_logic_vector(31 downto 0);
9   -- Debug data.
10  outA, outB : out std_logic_vector(31 downto 0);
11  outC, outD : out std_logic;
12  outIR : out std_logic_vector(31 downto 0);
13  outPC : out std_logic_vector(5 downto 0);
14  -- Processor-Instruction Memory Interface.
15  address : out std_logic_vector(5 downto 0);
16  wR : in std_logic;
17  memDataOut : out std_logic_vector(31 downto 0);
18  memDataIn : in std_logic_vector(31 downto 0);
19  -- Processor State
20  T_Info : out std_logic_vector(2 downto 0);
21  -- data Memory Interface
22  wen_mem, en_mem : out std_logic;
23  end cpu_test_sim;
24
25 ARCHITECTURE behavior OF cpu_test_sim IS
26 COMPONENT system_memory
27 PORT(
28   address : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
29   clock : IN STD_LOGIC;
30   data : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
31   wren : IN STD_LOGIC;
32   q : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
33 );
34 END COMPONENT;
35
36 COMPONENT cpu
37 PORT(
38   clk : in std_logic;
39   mem_clk : in std_logic;
40   rst : in std_logic;
41   dataIn : in std_logic_vector(31 downto 0);
42   dataOut : out std_logic_vector(31 downto 0);
43   addrOut : out std_logic_vector(31 downto 0);
44   wen : out std_logic;
45   dOutA, dOutB : out std_logic_vector(31 downto 0);
46   dOutC, dOutZ : out std_logic;
47   dOutIR : out std_logic_vector(31 downto 0);
48   dOutPC : out std_logic_vector(31 downto 0);
49   outT : out std_logic_vector(2 downto 0);
50   wen_mem, en_mem : out std_logic;
51 );
52 END COMPONENT;
53
54 signal cpu_to_mem : std_logic_vector(31 downto 0);
55 signal mem_to_cpu : std_logic_vector(31 downto 0);
56 signal add_from_cpu : std_logic_vector(31 downto 0);
57 signal wen_from_cpu : std_logic;
58
59 BEGIN
60   -- Component instantiations.
61   main_MEMORY : system_memory
62   PORT MAP(
63     address => add_from_cpu(5 downto 0),
64     clock => memClk,
65     data => cpu_to_mem,
66     wren => wen_from_cpu,
67     q => mem_to_cpu
68   );
69   main_processor : cpu
70   PORT MAP(
71     clk => cpuClk,
72     mem_clk => memClk,
73     rst => rst,
74     dataIn => mem_to_cpu,
75     dataOut => cpu_to_mem,
76     addrOut => add_from_cpu,
77     wen => wen_from_cpu,
78     dOutA => outA,
79     dOutB => outB,
80     dOutC => outC,
81     dOutZ => outZ,
82     dOutIR => outIR,
83     dOutPC => outPC,
84     outT => T_Info,
85     wen_mem => wen_mem,
86     en_mem => en_mem
87   );
88   addrOut <= add_from_cpu(5 downto 0);
89   wen <-> wen_from_cpu;
90   memDataOut <= mem_to_cpu;
91   memDataIn <= cpu_to_mem;
92 END behavior;

```

Figure 13: *cpu_test_sim*
VHDL code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use ieee.std_logic_unsigned.all;
5
6 ENTITY add IS
7 PORT(A : in std_logic_vector(31 downto 0);
8       B : out std_logic_vector(31 downto 0));
9      );
10
11
12 ARCHITECTURE Behavior OF add IS
13 BEGIN
14   B <= A + 1;
15 END Behavior;

```

Figure 14: *add* VHDL code

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 USE ieee.std_logic_arith.ALL;
4 USE ieee.std_logic_unsigned.ALL;
5
6 ENTITY reset_circuit IS
7 PORT(
8   Reset : IN STD_LOGIC;
9   Clk : IN STD_LOGIC;
10  Enable_PD : OUT STD_LOGIC := '1';
11  Clr_PC : OUT STD_LOGIC;
12  );
13 END reset_circuit;
14
15 ARCHITECTURE Behavior OF reset_circuit IS
16 TYPE clkNum IS (clk0, clk1, clk2, clk3);
17 SIGNAL present_clk: clkNum;
18 BEGIN
19  process (Clk) begin
20    if rising_edge(Clk) then
21      if Reset = '1' then
22        Clr_PC <= '1';
23        Enable_PD <= '0';
24        present_clk <= clk0;
25      elsif present_clk <= clk0 then
26        present_clk <= clk1;
27      elsif present_clk <= clk1 then
28        present_clk <= clk2;
29      elsif present_clk <= clk2 then
30        present_clk <= clk3;
31      elsif present_clk <= clk3 then
32        Clr_PC <= '0';
33        Enable_PD <= '1';
34      end if;
35    end if;
36  end process;
37 END Behavior;

```

Figure 15: *reset_circuit* VHDL
code

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 ENTITY Control_New IS
5 PORT(
6   clk : IN STD_LOGIC;
7   enable : IN STD_LOGIC;
8   statusC, statusD : IN STD_LOGIC;
9   RST : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
10  ADR_Mem, REG_Max : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
11  DATA_Max : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
12  ADD_pc : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
13  inc_pc : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
14  id_pc : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
15  id_ir : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
16  id_pc_sig : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
17  id_A : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
18  id_B : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
19  id_C : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
20  id_D : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
21  wen : OUT STD_LOGIC;
22  );
23 END Control_New;
24
25 ARCHITECTURE description OF Control_New IS
26 BEGIN
27 PROCESS begin
28   begin
29     if present_state = state_1 then
30       if present_state = state_1 then
31         if present_state = state_1 then
32           if present_state = state_1 then
33             if present_state = state_1 then
34               if present_state = state_1 then
35                 if present_state = state_1 then
36                   if present_state = state_1 then
37                     if present_state = state_1 then
38                       if present_state = state_1 then
39                         if present_state = state_1 then
40                           if present_state = state_1 then
41                             if present_state = state_1 then
42                               if present_state = state_1 then
43                                 if present_state = state_1 then
44                                   if present_state = state_1 then
45                                     if present_state = state_1 then
46                                       if present_state = state_1 then
47                                         if present_state = state_1 then
48                                           if present_state = state_1 then
49                                             if present_state = state_1 then
50                                               if present_state = state_1 then
51                                                 if present_state = state_1 then
52                                                   if present_state = state_1 then
53                                                     if present_state = state_1 then
54                                                       if present_state = state_1 then
55                                                         if present_state = state_1 then
56               end if;
57             end if;
58           end if;
59         end if;
60       end if;
61     end if;
62   end if;
63   end if;
64   end if;
65   end if;
66   end if;
67   end if;
68   end if;
69   end if;
70   end if;
71   end if;
72   end if;
73   end if;
74   end if;
75   end if;
76   end if;
77   end if;
78   end if;
79   end if;
80   end if;
81   end if;
82   end if;
83   end if;
84   end if;
85   end if;
86   end if;
87   end if;
88   end if;
89   end if;
90   end if;
91   end if;
92   end if;
93   end if;
94   end if;
95   end if;
96   end if;
97   end if;
98   end if;
99   end if;
100  end if;
101  end if;
102  end if;
103  end if;
104  end if;
105  end if;
106  end if;
107  end if;
108  end if;
109  end if;
110  end if;
111  end if;
112  end if;
113  end if;
114  end if;
115  end if;
116  end if;
117  end if;
118  end if;
119  end if;
120  end if;
121  end if;
122  end if;
123  end if;
124  end if;
125  end if;
126  end if;
127  end if;
128  end if;
129  end if;
130  end if;
131  end if;
132  end if;
133  end if;
134  end if;
135  end if;
136  end if;
137  end if;
138  end if;
139  end if;
140  end if;
141  end if;
142  end if;
143  end if;
144  end if;
145  end if;
146  end if;
147  end if;
148  end if;
149  end if;
150  end if;
151  end if;
152  end if;
153  end if;
154  end if;
155  end if;
156  end if;
157  end if;
158  end if;
159  end if;
160  end if;
161  end if;
162  end if;
163  end if;
164  end if;

```

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 USE ieee.std_logic_arith.all;
4 USE ieee.std_logic_unsigned.all;
5
6 ENTITY cpu1 IS
7 PORT(
8   clk : IN STD_LOGIC;
9   mem_clk : IN STD_LOGIC;
10  rex : IN STD_LOGIC;
11  wen : IN STD_LOGIC;
12  dataIn : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
13  dataOut : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
14  addROut : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
15  dOutA : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
16  dOutB : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
17  dOutC : OUT STD_LOGIC;
18  dOutZ : OUT STD_LOGIC;
19  dOutIR : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
20  dOutPC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
21  wEn : OUT STD_LOGIC;
22  outt : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
23  wen_mem : OUT STD_LOGIC;
24  en_mem : OUT STD_LOGIC;
25  );
26 END cpu1;
27
28 ARCHITECTURE description OF cpu1 IS
29 COMPONENT Data_Path IS
30 PORT(
31   Clk, mclk : IN STD_LOGIC; -- clock Signal
32   --Memory Signals
33   WEN, EN : IN STD_LOGIC;
34   Clr_A, Ld_A : IN STD_LOGIC;
35   Clr_B, Ld_B : IN STD_LOGIC;
36   Clr_C, Ld_C : IN STD_LOGIC;
37   Clr_Z, Ld_Z : IN STD_LOGIC;
38   ClrPC, Ld_PC : IN STD_LOGIC;
39   ClrIR, Ld_IR : IN STD_LOGIC;
40   Out_A : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
41   Out_B : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
42   Out_C : OUT STD_LOGIC;
43   Out_Z : OUT STD_LOGIC;
44   Out PC : OUT STD LOGIC VECTOR(31 DOWNTO 0);
45   );
46
47 OUT_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
48 OUT_IR : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
49 -- Special Inputs to PC:
50 PC : IN STD_LOGIC;
51 -- Address and Data Bus signals for debugging.
52 ADDR_OUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
53 DATA_IN : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
54 DATA_BUS, MEM_OUT, MEM_IN : OUT STD LOGIC VECTOR(31 DOWNTO 0);
55 MEM_ADDR : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
56
57 -- Various ROM controls.
58 DATA_ROM : ROM :> STD_LOGIC_VECTOR(31 DOWNTO 0);
59 REG_MUX : IN STD_LOGIC;
60 A_MUX, B_MUX : IN STD_LOGIC;
61 IM_MUX1 : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
62 IM_MUX2 : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
63 -- ALU Operations.
64 ALU_Op : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
65
66 END COMPONENT;
67
68 COMPONENT Control_New IS
69 PORT(
70   clk : IN STD_LOGIC;
71   enable : IN STD_LOGIC;
72   statusC, statusD : IN STD_LOGIC;
73   INST : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
74   A_MUX, B_MUX : OUT STD_LOGIC;
75   IM_MUX1, REG_MUX : OUT STD_LOGIC;
76   IM_MUX2, DATA_Max : OUT STD LOGIC_VECTOR(1 DOWNTO 0);
77   inc_pc : OUT STD_LOGIC;
78   id_pc : OUT STD_LOGIC;
79   id_A : OUT STD_LOGIC;
80   id_B : OUT STD LOGIC;
81   id_C : OUT STD LOGIC;
82   id_D : OUT STD LOGIC;
83   id_ir : OUT STD LOGIC;
84   id_pc_sig : OUT STD LOGIC;
85   id_A : OUT STD LOGIC;
86   id_B : OUT STD LOGIC;
87   id_C : OUT STD LOGIC;
88   id_D : OUT STD LOGIC;
89   wen : OUT STD LOGIC;
90   );
91 END COMPONENT;
92
93 COMPONENT reset_circuit IS
94 PORT(
95   Reset : IN STD_LOGIC;
96   CLK : IN STD LOGIC;
97   Enable_P0 : OUT STD LOGIC;
98   CLK_P0 : OUT STD LOGIC;
99   );
100 END COMPONENT;
101
102 begin
103   data_path : Data_Path
104   begin
105     Clk <- clk;
106     mclk <- mem_clk;
107     rex <- rex;
108     wen <- wen;
109     dataIn <- dataIn;
110     dataOut <- dataOut;
111     addROut <- addROut;
112     dOutA <- dOutA;
113     dOutB <- dOutB;
114     dOutC <- dOutC;
115     dOutZ <- dOutZ;
116     dOutIR <- dOutIR;
117     dOutPC <- dOutPC;
118     wEn <- wEn;
119     outt <- outt;
120     wen_mem <- wen_mem;
121     en_mem <- en_mem;
122   end;
123
124   control_new : Control_New
125   begin
126     clk <- clk;
127     mem_clk <- mem_clk;
128     rex <- rex;
129     wen <- wen;
130     dataIn <- dataIn;
131     dataOut <- dataOut;
132     addROut <- addROut;
133     dOutA <- dOutA;
134     dOutB <- dOutB;
135     dOutC <- dOutC;
136     dOutZ <- dOutZ;
137     dOutIR <- dOutIR;
138     dOutPC <- dOutPC;
139     wEn <- wEn;
140     outt <- outt;
141     wen_mem <- wen_mem;
142     en_mem <- en_mem;
143   end;
144
145   reset : reset_circuit
146   begin
147     Reset <- rst;
148     CLK <- clk;
149     Enable_P0 <- empd;
150     CLK_P0 <- pdlr;
151   end;
152
153   end;
154
155   end;
156
157   end;
158
159   end;
160
161   end;
162
163   end;
164
165   end;
166
167   end;
168
169   end;
170
171   end;
172
173   end;
174
175   end;
176
177   end;
178
179   end;
180
181   end;
182
183   end;
184
185   end;
186
187   end;
188
189   end;
190
191   end;
192
193   end;
194
195   end;
196
197   end;
198
199   end;
200
201   end;
202
203   end;
204
205   end;
206
207   end;
208
209   end;
210
211   end;
212
213   end;
214
215   end;
216
217   end;
218
219   end;
220
221   end;
222
223   end;
224
225   end;
226
227   end;
228
229   end;
230
231   end;
232
233   end;
234
235   end;
236
237   end;
238
239   end;
240
241   end;
242
243   end;
244
245   end;
246
247   end;
248
249   end;
250
251   end;
252
253   end;
254
255   end;
256
257   end;
258
259   end;
260
261   end;
262
263   end;
264
265   end;
266
267   end;
268
269   end;
270
271   end;
272
273   end;
274
275   end;
276
277   end;
278
279   end;
280
281   end;
282
283   end;
284
285   end;
286
287   end;
288
289   end;
290
291   end;
292
293   end;
294
295   end;
296
297   end;
298
299   end;
300
301   end;
302
303   end;
304
305   end;
306
307   end;
308
309   end;
310
311   end;
312
313   end;
314
315   end;
316
317   end;
318
319   end;
320
321   end;
322
323   end;
324
325   end;
326
327   end;
328
329   end;
330
331   end;
332
333   end;
334
335   end;
336
337   end;
338
339   end;
340
341   end;
342
343   end;
344
345   end;
346
347   end;
348
349   end;
350
351   end;
352
353   end;
354
355   end;
356
357   end;
358
359   end;
360
361   end;
362
363   end;
364
365   end;
366
367   end;
368
369   end;
370
371   end;
372
373   end;
374
375   end;
376
377   end;
378
379   end;
380
381   end;
382
383   end;
384
385   end;
386
387   end;
388
389   end;
390
391   end;
392
393   end;
394
395   end;
396
397   end;
398
399   end;
400
401   end;
402
403   end;
404
405   end;
406
407   end;
408
409   end;
410
411   end;
412
413   end;
414
415   end;
416
417   end;
418
419   end;
420
421   end;
422
423   end;
424
425   end;
426
427   end;
428
429   end;
430
431   end;
432
433   end;
434
435   end;
436
437   end;
438
439   end;
440
441   end;
442
443   end;
444
445   end;
446
447   end;
448
449   end;
450
451   end;
452
453   end;
454
455   end;
456
457   end;
458
459   end;
460
461   end;
462
463   end;
464
465   end;
466
467   end;
468
469   end;
470
471   end;
472
473   end;
474
475   end;
476
477   end;
478
479   end;
480
481   end;
482
483   end;
484
485   end;
486
487   end;
488
489   end;
490
491   end;
492
493   end;
494
495   end;
496
497   end;
498
499   end;
500
501   end;
502
503   end;
504
505   end;
506
507   end;
508
509   end;
510
511   end;
512
513   end;
514
515   end;
516
517   end;
518
519   end;
520
521   end;
522
523   end;
524
525   end;
526
527   end;
528
529   end;
530
531   end;
532
533   end;
534
535   end;
536
537   end;
538
539   end;
540
541   end;
542
543   end;
544
545   end;
546
547   end;
548
549   end;
550
551   end;
552
553   end;
554
555   end;
556
557   end;
558
559   end;
560
561   end;
562
563   end;
564
565   end;
566
567   end;
568
569   end;
570
571   end;
572
573   end;
574
575   end;
576
577   end;
578
579   end;
580
581   end;
582
583   end;
584
585   end;
586
587   end;
588
589   end;
590
591   end;
592
593   end;
594
595   end;
596
597   end;
598
599   end;
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
734
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
13
```

```

164         clz_A <= '0';
165         clz_B <= '0';
166         clz_C <= '0';
167         clz_D <= '0';
168         clz_E <= '0';
169         clz_F <= '0';
170         clz_G <= '0';
171         clz_H <= '0';
172         clz_I <= '0';
173
174         elsif Instruction_sig = "1000" then --BNE
175             clz_IR <= '0';
176             clz_H <= '0';
177             idz_PC <= '1';
178             idz_A <= '0';
179             idz_B <= '0';
180             idz_C <= '0';
181             idz_D <= '0';
182             idz_E <= '0';
183             idz_F <= '0';
184             idz_G <= '0';
185             idz_H <= '0';
186
187         elsif Instruction_sig = "1001" then --LDA
188             clz_IR <= '0';
189             idz_H <= '0';
190             idz_PC <= '0';
191             incz_PC <= '0';
192             clz_A <= '0';
193             clz_B <= '0';
194             idz_B <= '0';
195             clz_B <= '0';
196             clz_C <= '0';
197             idz_C <= '0';
198             clz_D <= '0';
199             idz_D <= '0';
200             A_Mem <= '0';
201             A_Mem_Next <= '01';
202             en <= '1';
203             wen <= '0';
204
205         elsif Instruction_sig = "1010" then --LCB
206             clz_IR <= '0';
207             idz_H <= '0';
208             idz_PC <= '0';
209             idz_FC <= '0';
210             clz_A <= '0';
211             idz_A <= '0';
212             idz_B <= '0';
213             idz_C <= '0';
214             idz_D <= '0';
215             idz_E <= '0';
216             idz_F <= '0';
217             B_Mem <= '0';
218
219             B_Mem <= '0';
220             DATA_Mem <= "01";
221             en <= '1';
222             wen <= '0';
223
224         elsif Instruction_sig = "0010" then --STA
225             clz_IR <= '0';
226             idz_H <= '0';
227             idz_PC <= '0';
228             incz_PC <= '0';
229             clz_A <= '0';
230             clz_B <= '0';
231             clz_C <= '0';
232             clz_D <= '0';
233             idz_E <= '0';
234             idz_F <= '0';
235             idz_G <= '0';
236             idz_H <= '0';
237             DATA_Mem <= "00";
238             en <= '1';
239             wen <= '1';
240
241         elsif Instruction_sig = "0011" then --STB
242             clz_IR <= '0';
243             idz_H <= '0';
244             idz_PC <= '0';
245             incz_PC <= '0';
246             clz_A <= '0';
247             clz_B <= '0';
248             idz_B <= '0';
249             clz_B <= '0';
250             clz_C <= '0';
251             idz_C <= '0';
252             idz_D <= '0';
253             idz_E <= '0';
254             idz_F <= '0';
255             DATA_Mem <= "00";
256             en <= '1';
257             wen <= '1';
258
259         elsif Instruction_sig = "0000" then --LDALI
260             clz_IR <= '0';
261             idz_H <= '0';
262             idz_PC <= '0';
263             incz_PC <= '0';
264             clz_A <= '0';
265             clz_B <= '0';
266             idz_B <= '0';
267             clz_B <= '0';
268             clz_C <= '0';
269             idz_C <= '0';
270             idz_D <= '0';
271             idz_E <= '0';
272             A_Mem <= '1';
273             A_Mem <= '1';
274
275         elsif Instruction_sig = "0000" then --LDISI
276             clz_IR <= '0';
277             idz_H <= '0';
278             idz_PC <= '0';
279             incz_PC <= '0';
280             clz_A <= '0';
281             idz_A <= '1';
282             idz_B <= '0';
283             clz_B <= '0';
284             idz_C <= '0';
285             idz_D <= '0';
286             idz_E <= '0';
287             B_Mem <= '1';
288
289         elsif Instruction_sig = "0100" then --LUI
290             clz_IR <= '0';
291             idz_H <= '0';
292             idz_PC <= '0';
293             incz_PC <= '0';
294             clz_A <= '0';
295             idz_A <= '0';
296             idz_B <= '0';
297             clz_B <= '1';
298             idz_C <= '0';
299             idz_D <= '0';
300             idz_E <= '0';
301             idz_F <= '0';
302             ADD_op <= "001";
303             A_Mem <= '0';
304             DATA_Mem <= "10";
305             IM_MemX <= '1';
306
307         elsif Instruction_sig2 = "01111001" then --ANDI
308             clz_H <= '0';
309             idz_H <= '0';
310             idz_PC <= '0';
311             idz_FC <= '0';
312             clz_A <= '0';
313             idz_A <= '0';
314             idz_B <= '0';
315             clz_B <= '0';
316             idz_C <= '0';
317             idz_D <= '0';
318             clz_D <= '0';
319             idz_E <= '1';
320             A_Mem <= "000";
321             A_Mem <= '0';
322             DATA_Mem <= "10";
323             IM_MemX <= '0';
324             IM_MemZ <= '0';
325
326         elsif Instruction_sig2 = "01111110" then --DECA
327
328

```

```

126
127         elsif Instruction_sig2 = "01111110" then --DECA
128             id_IR <= '0';
129             id_PC <= '0';
130             id_A <= '0';
131             id_B <= '0';
132             id_C <= '1';
133             id_D <= '0';
134             id_E <= '0';
135             id_F <= '0';
136             id_G <= '0';
137             id_H <= '0';
138             id_I <= '1';
139             id_Z <= '1';
140             MUX_M <= "0000";
141             A_MUX <= '0';
142             DATA_MUX <= "100";
143             IM_M0X1 <= "10";
144             IM_M0X2 <= "10";
145
146         elsif Instruction_sig2 = "01110000" then --ADD
147             id_IR <= '0';
148             id_PC <= '0';
149             id_A <= '0';
150             id_B <= '0';
151             id_C <= '0';
152             id_D <= '1';
153             id_E <= '0';
154             id_F <= '0';
155             id_G <= '1';
156             id_H <= '0';
157             id_I <= '0';
158             ALU_op <= "0101";
159             A_MUX <= '0';
160             DATA_MUX <= "10";
161             IM_M0X1 <= '0';
162             IM_M0X2 <= "000";
163
164         elsif Instruction_sig2 = "01110010" then --SUB
165             id_IR <= '0';
166             id_PC <= '0';
167             id_A <= '0';
168             id_B <= '0';
169             id_C <= '0';
170             id_D <= '1';
171             id_E <= '0';
172             id_F <= '0';
173             id_G <= '0';
174             id_H <= '0';
175             id_I <= '0';
176             id_Z <= '1';
177             ALU_op <= "0101";
178             A_MUX <= '0';
179             DATA_MUX <= "000";
180             IM_M0X1 <= '0';
181             IM_M0X2 <= "00";
182
183         elsif Instruction_sig2 = "01110011" then --INCA
184             id_IR <= '0';
185             id_IR <= '0';
186             id_PC <= '0';
187             id_A <= '0';
188             id_A <= '1';
189             id_B <= '0';
190             id_C <= '0';
191             id_D <= '0';
192             id_E <= '0';
193             id_F <= '0';
194             id_G <= '0';
195             id_H <= '0';
196             id_I <= '0';
197             ALU_op <= "010";
198             A_MUX <= '0';
199             DATA_MUX <= "10";
200             IM_M0X1 <= "10";
201             IM_M0X2 <= "00";
202
203         elsif Instruction_sig2 = "01111011" then --AND
204             id_IR <= '0';
205             id_IR <= '0';
206             id_PC <= '0';
207             id_A <= '0';
208             id_A <= '1';
209             id_B <= '0';
210             id_C <= '0';
211             id_D <= '0';
212             id_E <= '0';
213             id_F <= '0';
214             id_G <= '0';
215             id_H <= '0';
216             id_I <= '0';
217             ALU_op <= "0000";
218             A_MUX <= '0';
219             DATA_MUX <= "000";
220             IM_M0X1 <= '0';
221             IM_M0X2 <= "00";
222
223         elsif Instruction_sig2 = "01110001" then --ADDI
224             id_IR <= '0';
225             id_IR <= '0';
226             id_PC <= '0';
227             id_A <= '0';
228             id_A <= '0';
229             id_B <= '0';
230             id_C <= '0';
231             id_D <= '0';
232             id_E <= '0';
233             id_F <= '0';
234             id_G <= '0';
235             id_H <= '0';
236             id_I <= '0';
237             id_Z <= '0';
238             ALU_op <= "010";

```

```

434 ALU_op <= "010";
435 A_Max <= "1";
436 DATA_Max <= "10";
437 IM_HDXL <= "0";
438 IM_HDXR <= "01";
439
440 elsif Instruction_sig2 = "01111101" then --ORI
441   clz_IR <= "0";
442   id_IR <= "0";
443   id_FC <= "01";
444   inc_FC <= "0";
445   clz_A <= "0";
446   id_A <= "0";
447   id_B <= "0";
448   clz_B <= "0";
449   id_C <= "0";
450   id_FC <= "1";
451   id_Z <= "0";
452   id_E <= "1";
453   ALU_op <= "001";
454   A_Max <= "01";
455   DATA_Max <= "10";
456   IM_HDXL <= "0";
457   IM_HDXR <= "01";
458
459 elsif Instruction_sig2 = "01111010" then --ROL
460   clz_IR <= "0";
461   id_IR <= "0";
462   id_FC <= "01";
463   inc_FC <= "0";
464   clz_A <= "0";
465   id_A <= "1";
466   id_B <= "0";
467   clz_B <= "0";
468   id_C <= "0";
469   id_D <= "1";
470   clz_E <= "0";
471   id_E <= "0";
472   ALU_op <= "100";
473   A_Max <= "01";
474   DATA_Max <= "10";
475   IM_HDXL <= "0";
476
477 elsif Instruction_sig2 = "01111111" then --RRR
478   clz_IR <= "0";
479   id_IR <= "0";
480   id_FC <= "01";
481   inc_FC <= "0";
482   clz_A <= "0";
483   id_A <= "1";
484   id_B <= "0";
485   clz_B <= "0";
486   id_C <= "0";
487   id_D <= "1";
488   clz_E <= "0";
489   id_E <= "0";
490
491 elsif Instruction_sig2 = "011110101" then --CLR_A
492   clz_IR <= "0";
493   id_ID <= "0";
494   id_FC <= "0";
495   inc_FC <= "0";
496   A_Max <= "01";
497   DATA_Max <= "10";
498   IM_HDXL <= "0";
499
500 elsif Instruction_sig2 = "01111010" then --CLR_B
501   clz_IR <= "0";
502   id_ID <= "0";
503   id_FC <= "0";
504   inc_FC <= "0";
505   clz_A <= "0";
506   id_A <= "0";
507   id_B <= "0";
508   clz_B <= "0";
509   id_C <= "0";
510   id_D <= "0";
511   id_E <= "0";
512
513 elsif Instruction_sig2 = "01111011" then --CLR_C
514   clz_IR <= "0";
515   id_ID <= "0";
516   id_FC <= "0";
517   inc_FC <= "0";
518   clz_A <= "0";
519   id_A <= "0";
520   id_B <= "0";
521   clz_B <= "0";
522   id_C <= "0";
523   id_D <= "0";
524   id_E <= "0";
525
526 elsif Instruction_sig2 = "01111000" then --CLR_D
527   clz_IR <= "0";
528   id_ID <= "0";
529   id_FC <= "0";
530   inc_FC <= "0";
531   clz_A <= "0";
532   id_A <= "0";
533   id_B <= "0";
534   clz_B <= "0";
535   id_C <= "0";
536   id_D <= "0";
537
538 elsif Instruction_sig2 = "01111010" then --TEST
539   if (status = "1") then
540     clz_IR <= "0"; --INCREMENT PC COUNTER
541     id_FC <= "1";
542     inc_FC <= "1";
543     id_ID <= "0";
544     id_A <= "0";
545     id_B <= "0";
546     clz_B <= "0";
547     id_C <= "0";
548     id_D <= "0";
549     id_E <= "0";
550
551   elsif Instruction_sig2 = "01111100" then --TESTC
552   if (status = "1") then
553     clz_IR <= "0"; --INCREMENT PC COUNTER
554     id_FC <= "1";
555     inc_FC <= "1";
556     id_ID <= "0";
557     id_A <= "0";
558     id_B <= "0";
559     clz_B <= "0";
560     id_C <= "0";
561     id_D <= "0";
562     id_E <= "0";
563
564   end if;
565
566 elsif Instruction_sig2 = "01111100" then --TESTC
567   if (status = "1") then
568     clz_IR <= "0"; --INCREMENT PC COUNTER
569     id_FC <= "1";
570     inc_FC <= "1";
571     id_ID <= "0";
572     id_A <= "0";
573     id_B <= "0";
574     clz_B <= "0";
575     id_C <= "0";
576     id_D <= "0";
577     id_E <= "0";
578
579   end if;
580
581 end if;
582
583 end if; --For State 2 Ops
584
585 end if; --For Enable
586
587 END process;
588
589 ----- SYSTEM MACHINE -----
590
591 PROCESS (clz, enable)
592 begin
593   if enable = "1" then
594     if clz = "0" then
595       present_state = state_0 when present_state <= state_3;
596       elsif present_state = state_1 then
597         present_state = state_2;
598       elsif present_state = state_2 then
599         present_state = state_3;
600       end if;
601     end if;
602   else
603     present_state = state_0;
604   end if;
605 end process;

```

```

556      else present_state <= state_0;
557    end if;
558  END process;
559
560  WITH present_state select
561    T <= "001" when state_0,
562      "010" when state_1,
563      "100" when state_2,
564      "001" when others;
565  END description;

```

Figure 16: Control VHDL code

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity RED is
6  port (
7    RED_in : in std_logic_vector(31 downto 0);
8    RED_out : out unsigned(7 downto 0)
9  );
10 end entity;
11
12 architecture Behavior of RED is
13 begin
14   L: begin
15     RED_out <= unsigned (RED_in(7 downto 0));
16   end Behavior;

```

Figure 19: RED VHDL code

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity LZE is
6  port (
7    LZE_in : in std_logic_vector(31 downto 0);
8    LZE_out : out std_logic_vector(31 downto 0)
9  );
10 end entity;
11
12 architecture Behavior of LZE is
13 signal zeros: std_logic_vector(15 downto 0) := (others => '0');
14 begin
15   L: begin
16     LZE_out <= zeros & LZE_in(15 downto 0);
17   end Behavior;

```

Figure 20: LZE VHDL code

Figure 1 shows VHDL code that defines a memory module (system_memory). The memory has a single port and a width of 32 bits and a depth of 64 words. It reads and writes data based on the input address, clock, data, and write enable signals. The memory is initialized from a memory initialization file (system_memory.mif) and is configured for a Cyclone II device.

Figure 2 shows VHDL code for a 32-bit register (register32), featuring input data (d), load control (ld), clear control (clr), clock signal (clk), and output data (Q). It operates on rising clock edges (clk'event and clk = '1'), loading input data (d) when ld is asserted and clearing the register to '0' when clr is asserted. This code represents a standard synchronous register design for data storage and updating in digital systems.

Figure 3 shows VHDL code for a simple program counter (PC) that can increment its value based on control signals. It employs components such as an adder, a 2-to-1 multiplexer, and a 32-bit register. The PC can be cleared (clr signal), loaded with a new value (ld signal), or incremented (inc signal), with its current value output through the q port. The internal operations involve addition, multiplexing, and storage in a register, facilitating basic program control and sequencing in digital systems.

Figure 4 shows VHDL code for a 4-to-1 multiplexer (mux4to1) with 32-bit input vectors (X1, X2, X3, X4) and a 2-bit control vector (s). The output (f) is selected based on the control vector's binary value, with X1 selected for "00", X2 for "01", X3 for "10", and X4 for "11".

Figure 5 shows VHDL code for a 32-bit 2-to-1 multiplexer (mux2to1) selecting between input vectors w0 and w1 based on control signal called, s.

Figure 6 shows VHDL code for a full adder (fulladd) computing sum (s) and carry-out (Cout) from three inputs (Cin, x, y) using XOR and AND logic.

Figure 7 shows VHDL code for a 256-word, 32-bit vector data memory (data_mem) with clock, address, data input, write enable, read enable, and data output ports. Operations are controlled

by rising clock edges, with data stored in DATAMEM array, performing read/write based on control signals and address.

Figure 8 shows VHDL code for a 32-bit Arithmetic Logic Unit (ALU) handling inputs (a and b), a 3-bit operation code (op), and producing outputs (result, zero, and cout). The ALU executes operations such as AND, OR, addition, subtraction, shifts, and bit manipulation using an adder component (adder32) and zero flag logic.

Figure 9 shows VHDL code for a 32-bit adder (adder32) composed of two 16-bit adders (adder16). The inputs (X and Y) are split into 16-bit segments for two-stage addition. The carry-out from the first stage feeds into the second, yielding a 32-bit sum (S) and carry-out (Cout), demonstrating a common modular design technique for building larger adders.

Figure 10 shows VHDL code for a 16-bit adder (adder16), employing four 4-bit adder instances (adder4). Inputs X and Y are split into four 4-bit segments, processed in four stages with carry-out (C) feeding into the next stage. The output comprises a 16-bit sum (S) and final carry-out (Cout), illustrating a common modular design for larger adders.

Figure 11 shows VHDL code for a 4-bit adder (adder4) utilizing four full adder instances (fulladd). Inputs X and Y are partitioned into four 1-bit segments, processed in four stages with carry-out (C) feeding into the subsequent stage. The output consists of a 4-bit sum (S) and final carry-out (Cout), showcasing a typical modular approach in digital circuit design for building larger adders through smaller ones' cascading.

Figure 12 shows VHDL code for data path (data_path), which models a data path in a digital system. It includes various components such as registers, multiplexers, an arithmetic logic unit (ALU), and a memory unit. The architecture Behavior instantiates these components and connects them to implement the desired functionality, such as loading data into registers, performing arithmetic operations, and accessing memory. The code also handles control signals for various operations and data flow within the system.

Figure 13 shows VHDL code that describes a testbench for simulating a CPU design. It instantiates two components: "system_memory" representing memory and "cpu1" representing the CPU itself. The CPU communicates with memory through signals "cpu_to_mem" and "mem_to_cpu" for data transfer, and it controls memory writes and reads through signals "add_from_cpu" and "wen_from_cpu". The testbench facilitates the interaction between the CPU and memory during simulation, allowing for testing and debugging of the CPU's functionality.

Figure 14 is a VHDL code that defines a simple adder (add) with a 32-bit input vector (A) and a 32-bit output vector (B). The adder performs the operation $B \leftarrow A + 1$, incrementing the value of the input vector A by 1. This code represents a basic arithmetic operation and is useful in various digital systems for performing simple addition operations.

Figure 15 is a VHDL code that defines a synchronous reset circuit (`reset_circuit`). It takes two inputs, `Reset` and `Clk`, and provides two outputs, `Enable_PD` and `Clr_PC`. When the clock signal (`Clk`) rises, it checks if the reset signal (`Reset`) is asserted. If it is, it sets `Clr_PC` to '1' and `Enable_PD` to '0'. Otherwise, it sequentially updates the `present_clk` signal through different states (`clk0` to `clk3`) and adjusts `Clr_PC` and `Enable_PD` accordingly to manage the reset behavior.

Figure 16 shows VHDL code for Control Unit (Control), which represents a control unit in a digital system. It takes various inputs such as clock signals, instruction signals, and status flags, and produces control signals for various components in the system. The architecture description implements a state machine to decode instructions and generate appropriate control signals based on the current instruction and system status. It handles instruction decoding and generates control signals for operations such as load/store, arithmetic operations, and conditional branching. Additionally, it incorporates a state machine to manage the state transitions of the control unit.

Figure 17 shows VHDL code that defines the architecture of a CPU design (`cpu1`). It comprises three components: "Data_Path", "Control Unit", and "reset_circuit". The "Data_Path" component handles data flow and control signals within the CPU, while "Control Unit" manages the control signals based on the current instruction and CPU status. The "reset_circuit" component handles the reset functionality. The signals between these components are interconnected using port maps, defining how data and control signals flow within the CPU architecture.

Figure 18 shows VHDL code for an upper zero extender (UZE), expanding a 32-bit input vector (`UZE_in`) by padding its lower 16 bits with leading zeros, resulting in a 32-bit output vector (`UZE_out`). This extends the input vector's width and is frequently applied in digital systems for data consistency.

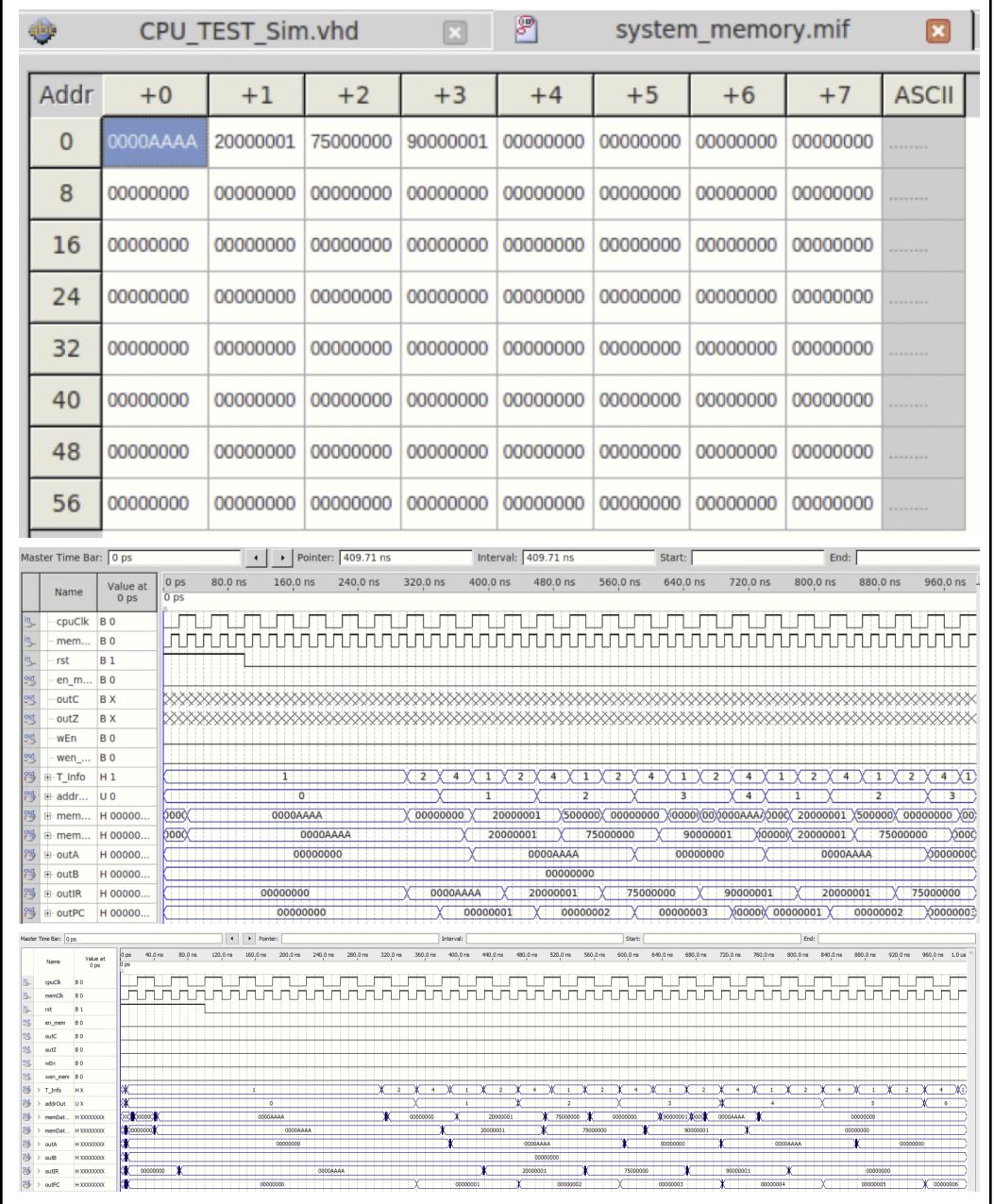
Figure 19 shows VHDL code for a vector reducer (RED), which converts a 32-bit input vector (`RED_in`) into an unsigned 8-bit vector (`RED_out`) by extracting and converting its lower 8 bits. This process reduces the vector size while preserving its numerical value within the lower bit range, a common operation in data manipulation.

Figure 20 shows VHDL code for a zero extender (LZE), augmenting a 32-bit input vector (`LZE_in`) by adding 16 leading zeros, thereby expanding the vector by 16 bits. Such operations are essential in digital systems to maintain uniform data width for further processing.

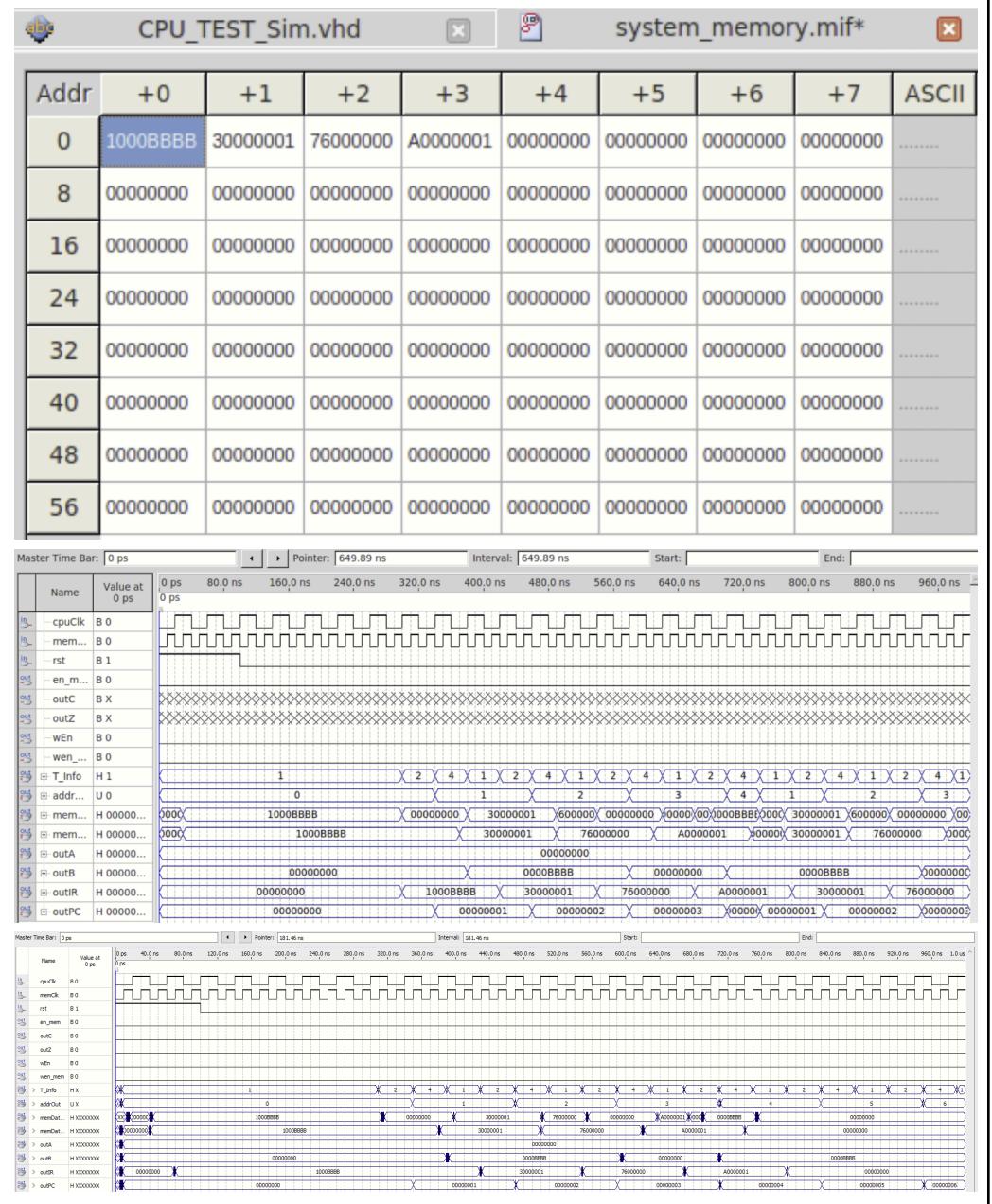
3. Results:

Operations	Figures of MIP, Functional, and Timing Waveforms
------------	--

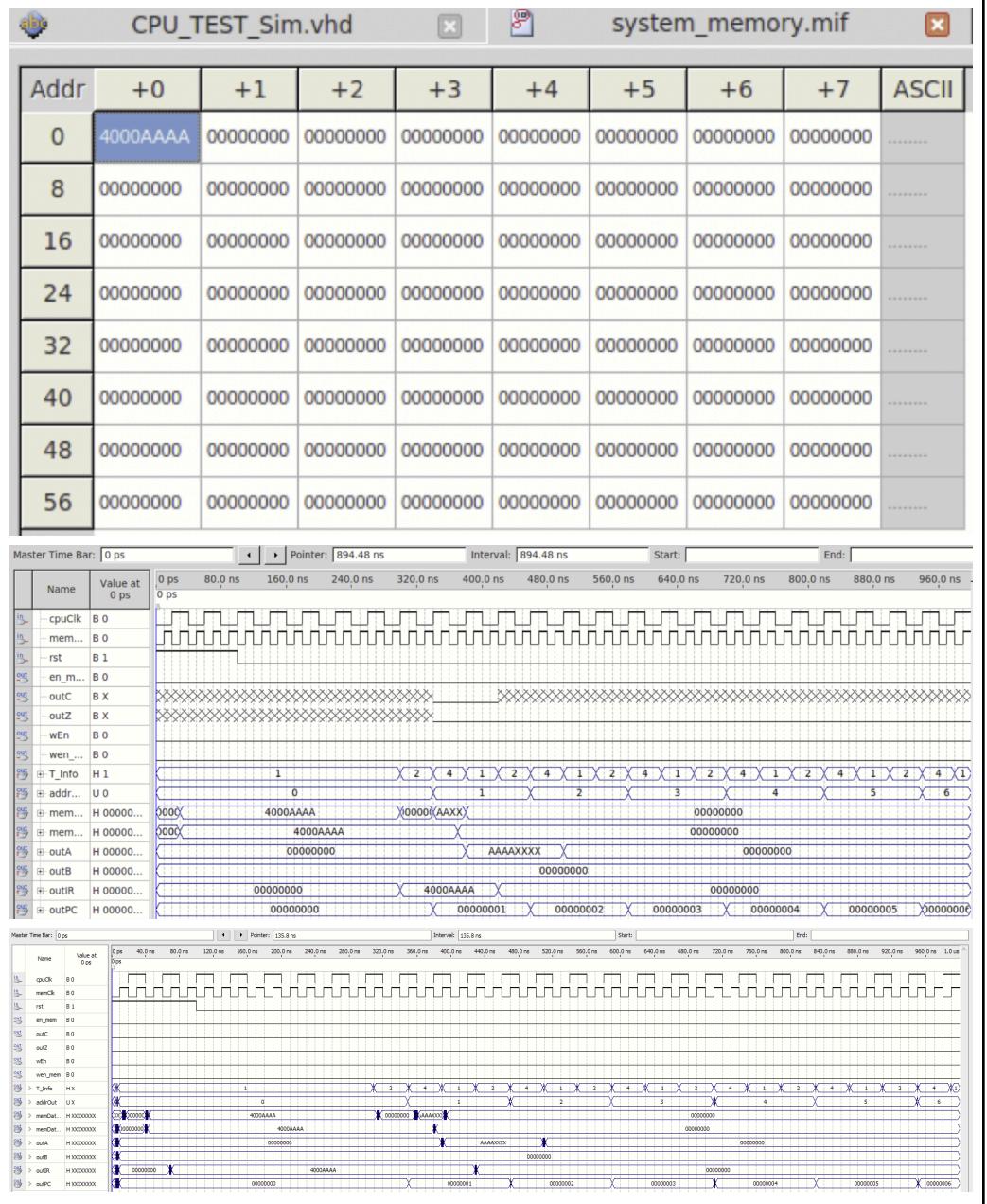
LDAI, STA, CLRA, LDA



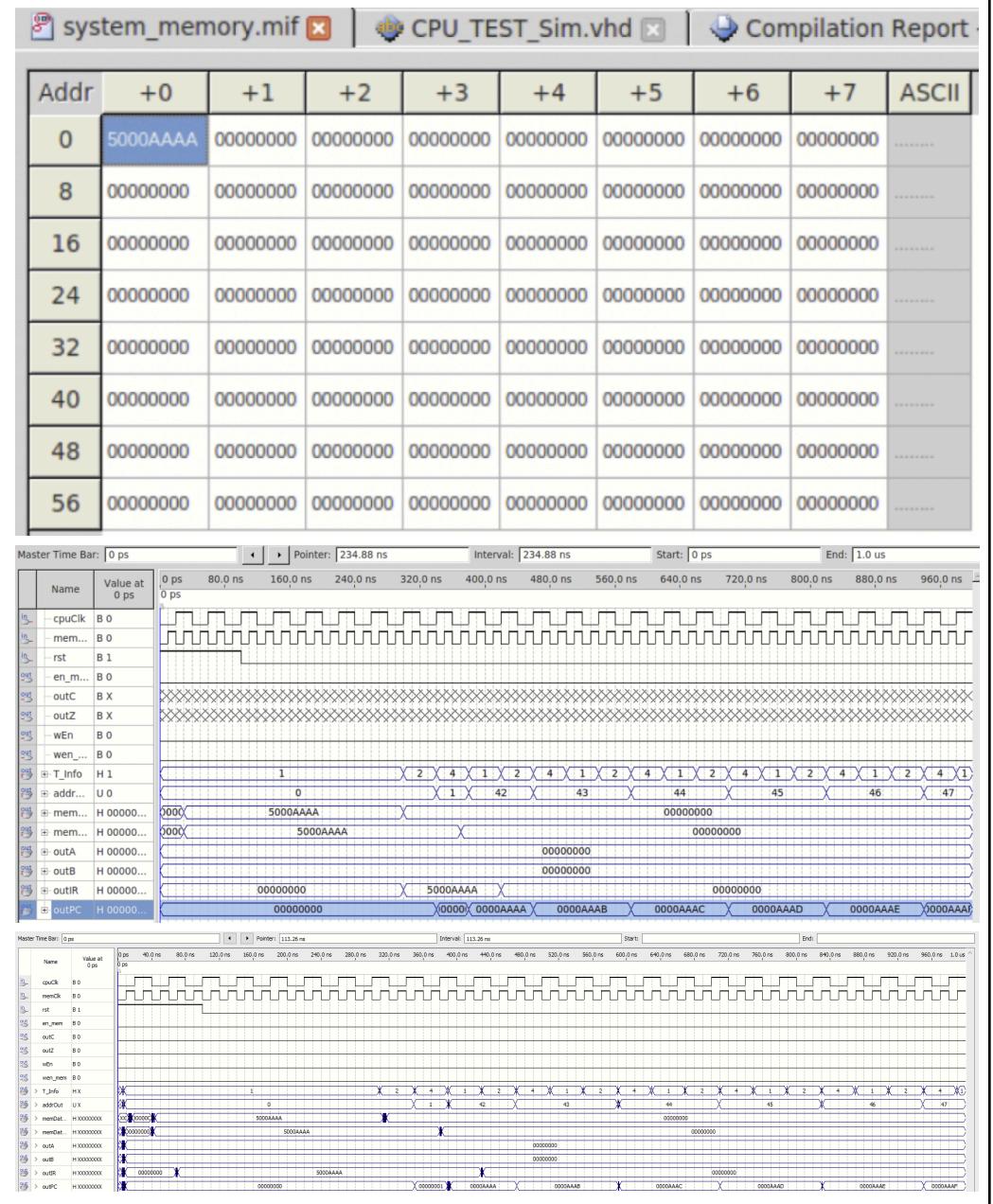
LDBI, STB, CLRB, LDB



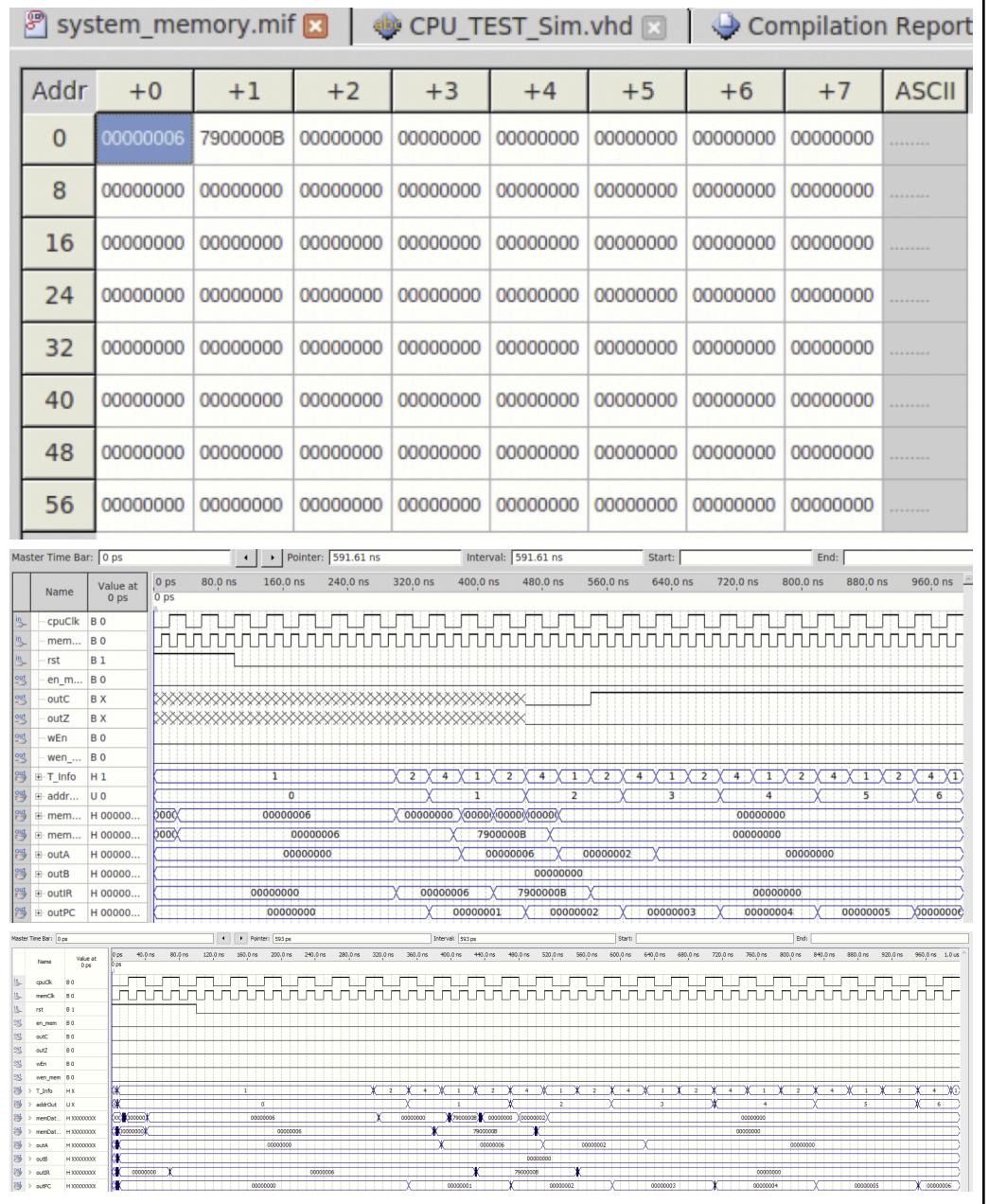
LUI



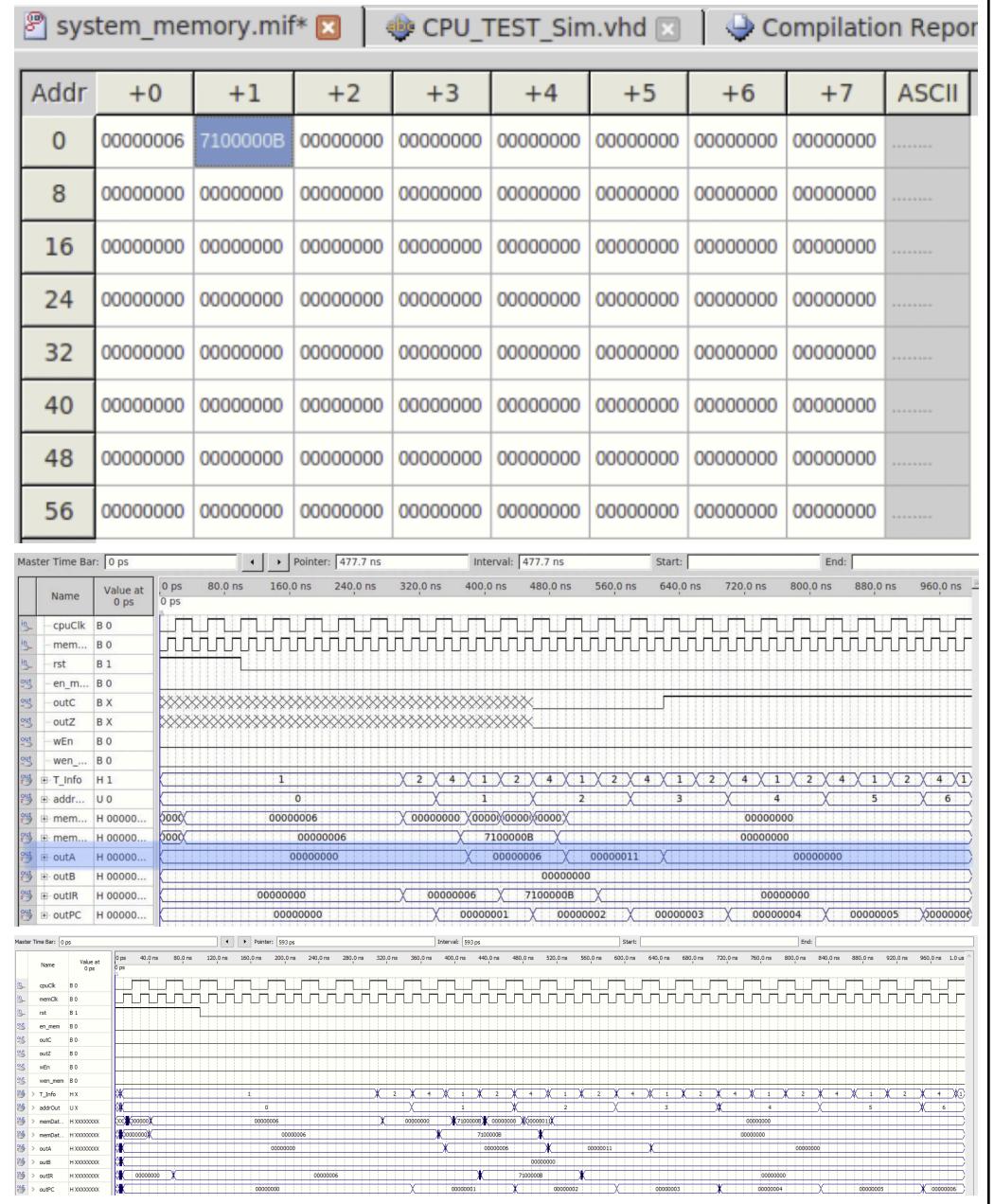
JMP



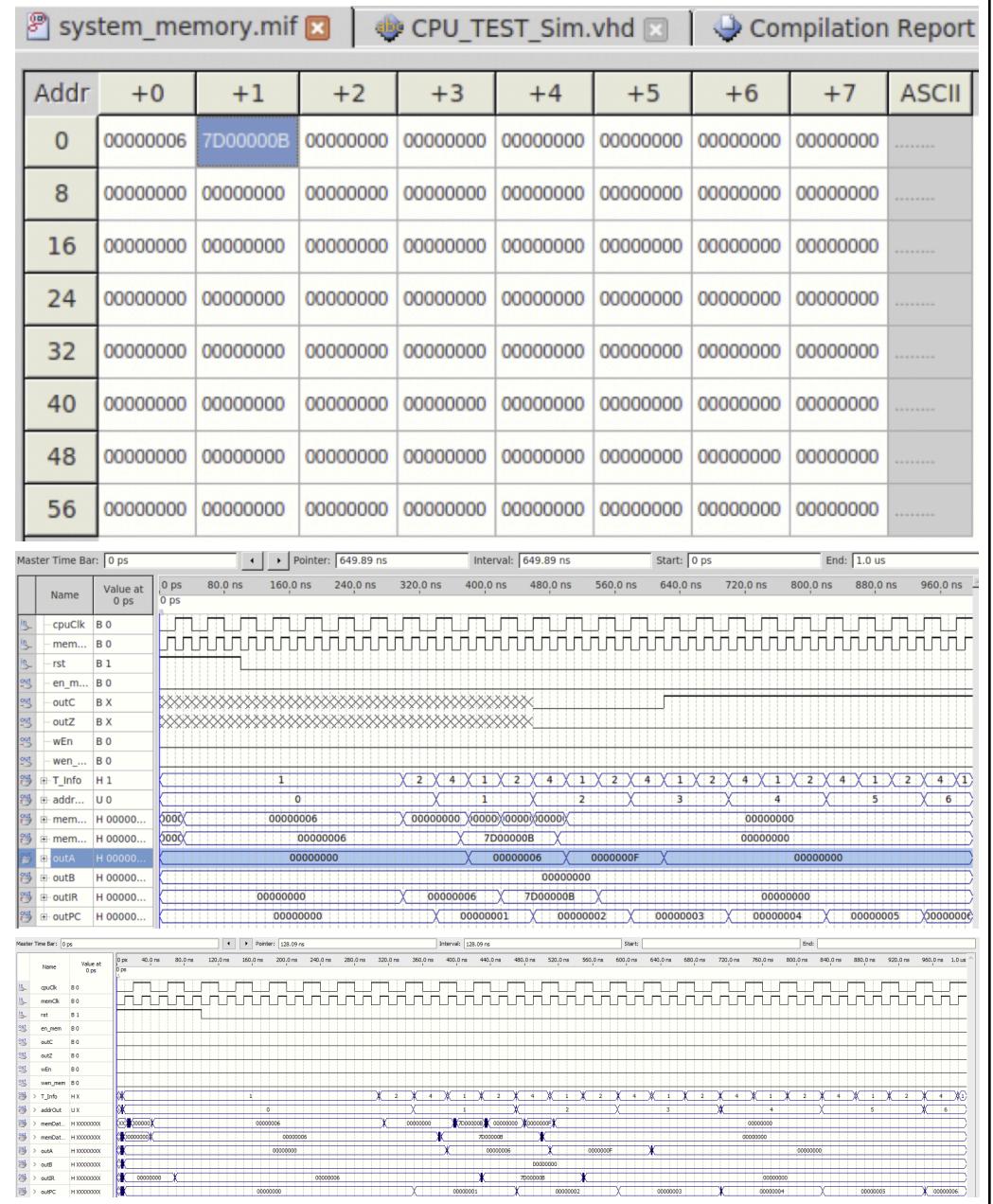
ANDI



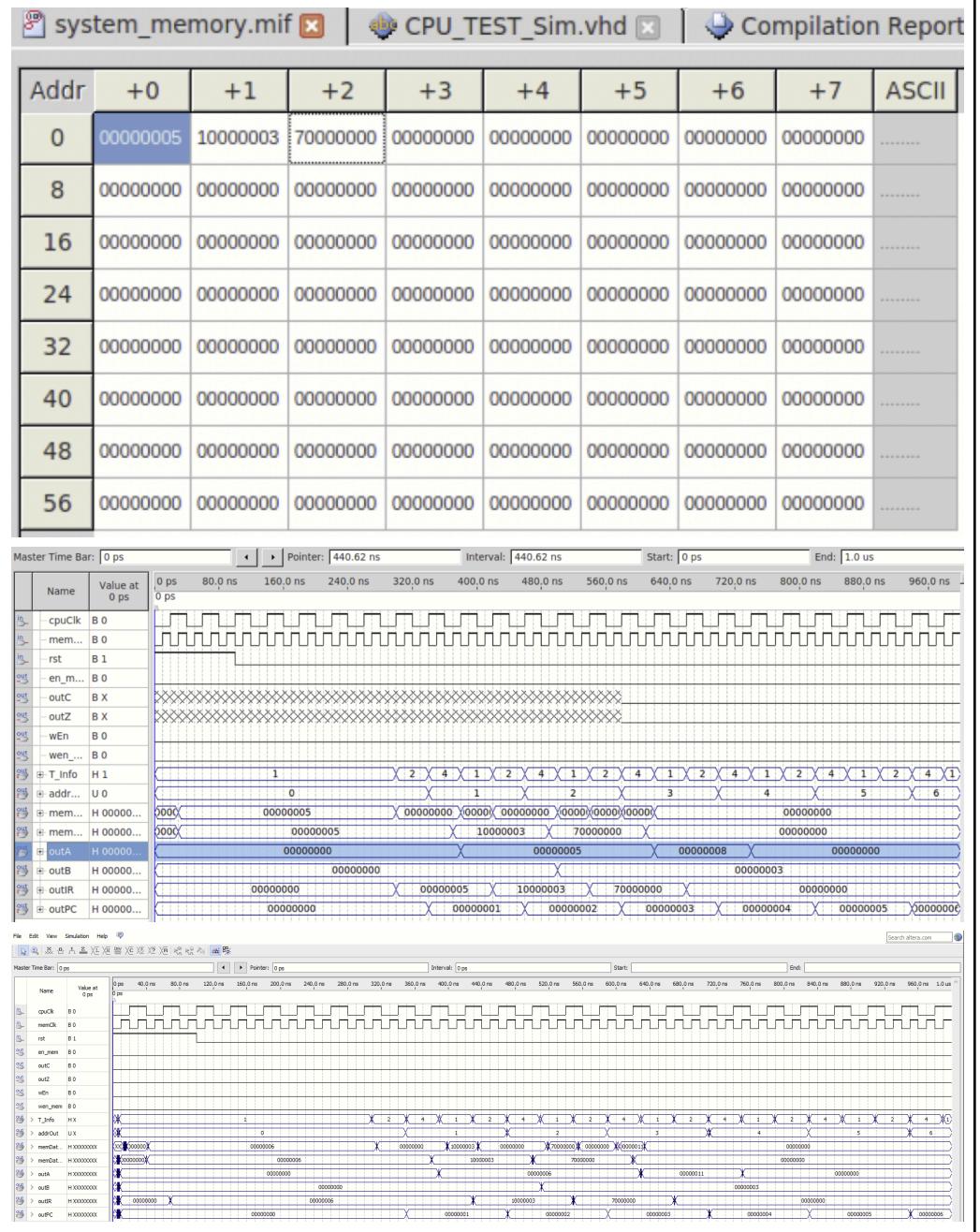
ADDI



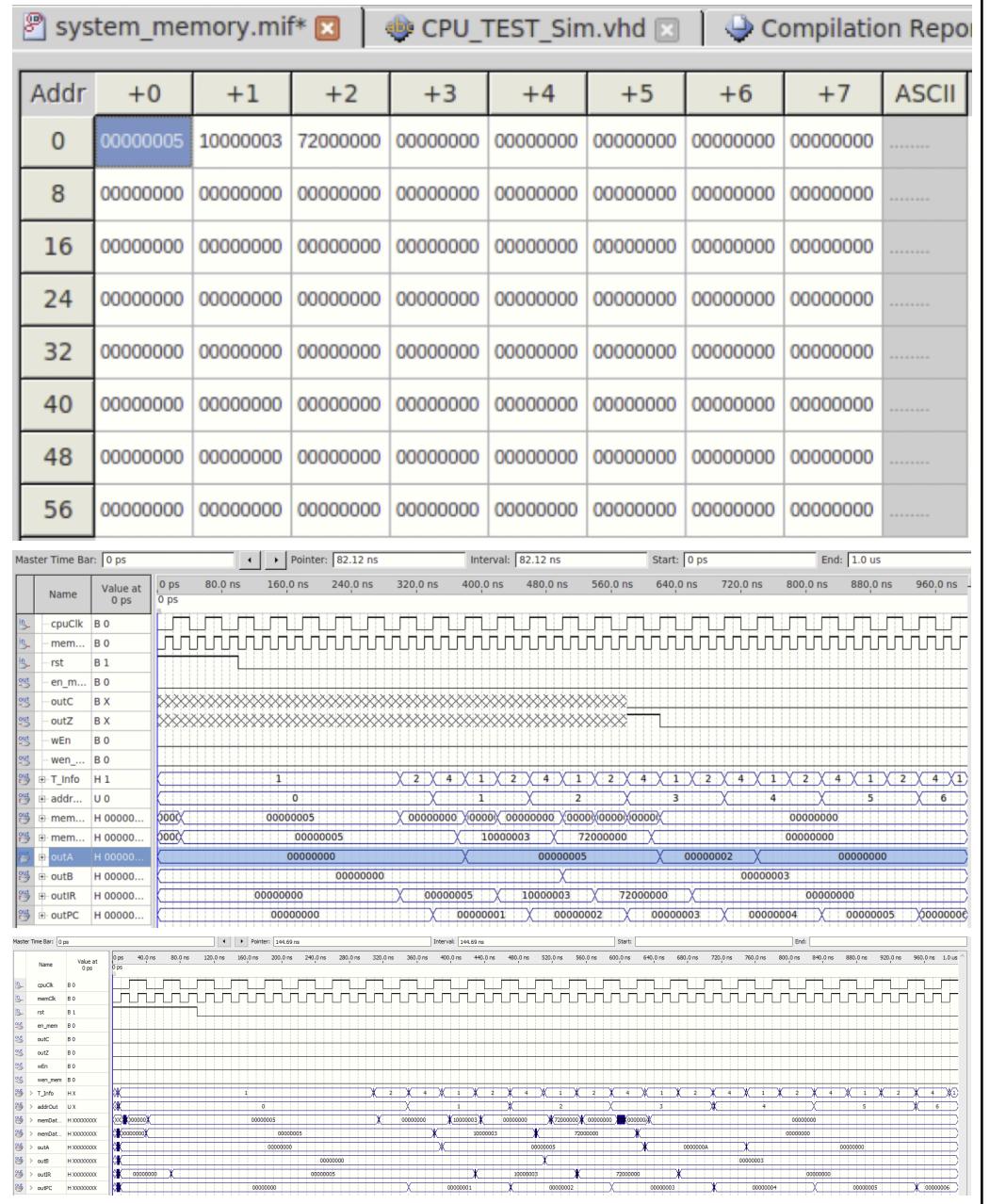
ORI



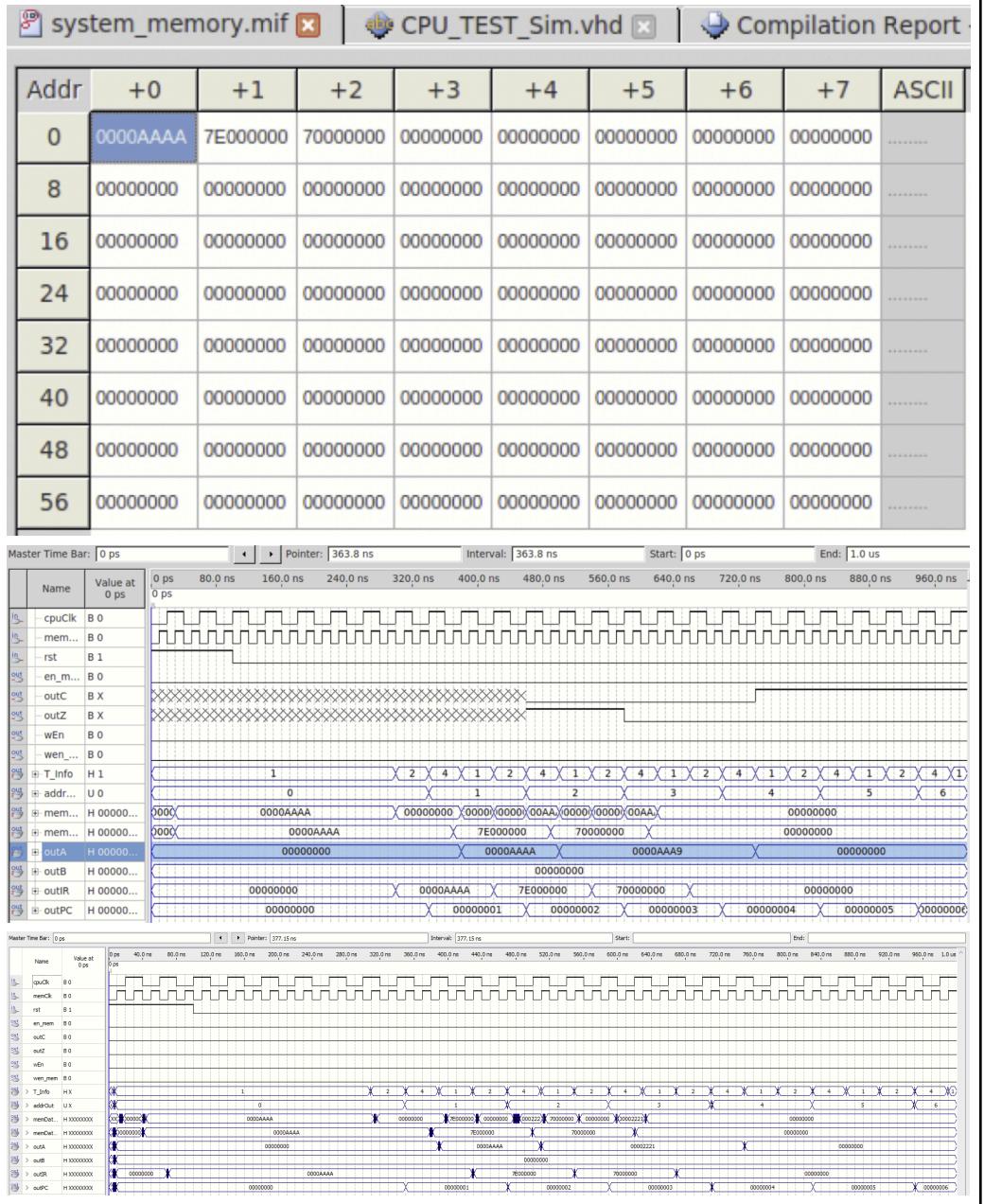
ADD



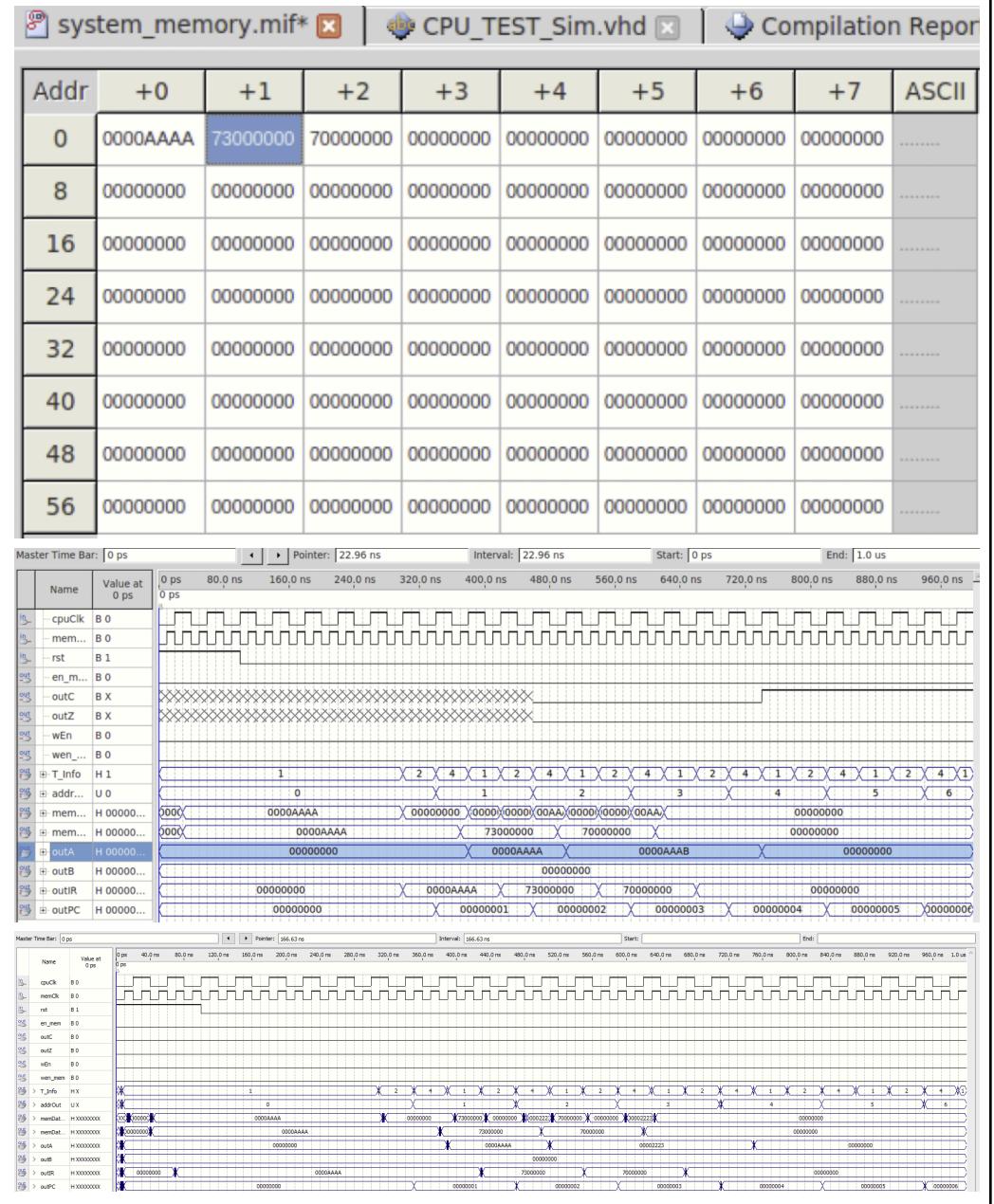
SUB



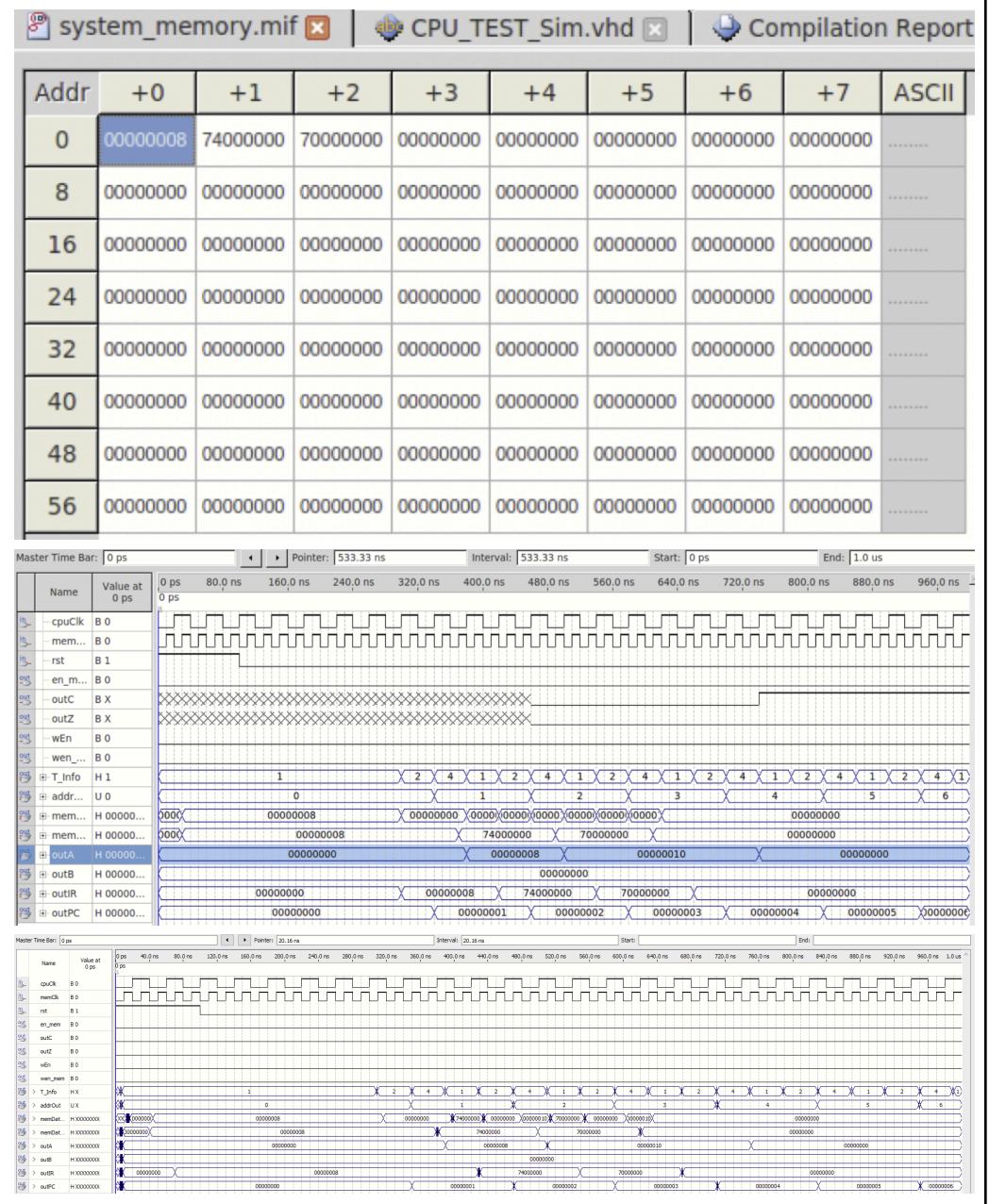
DECA



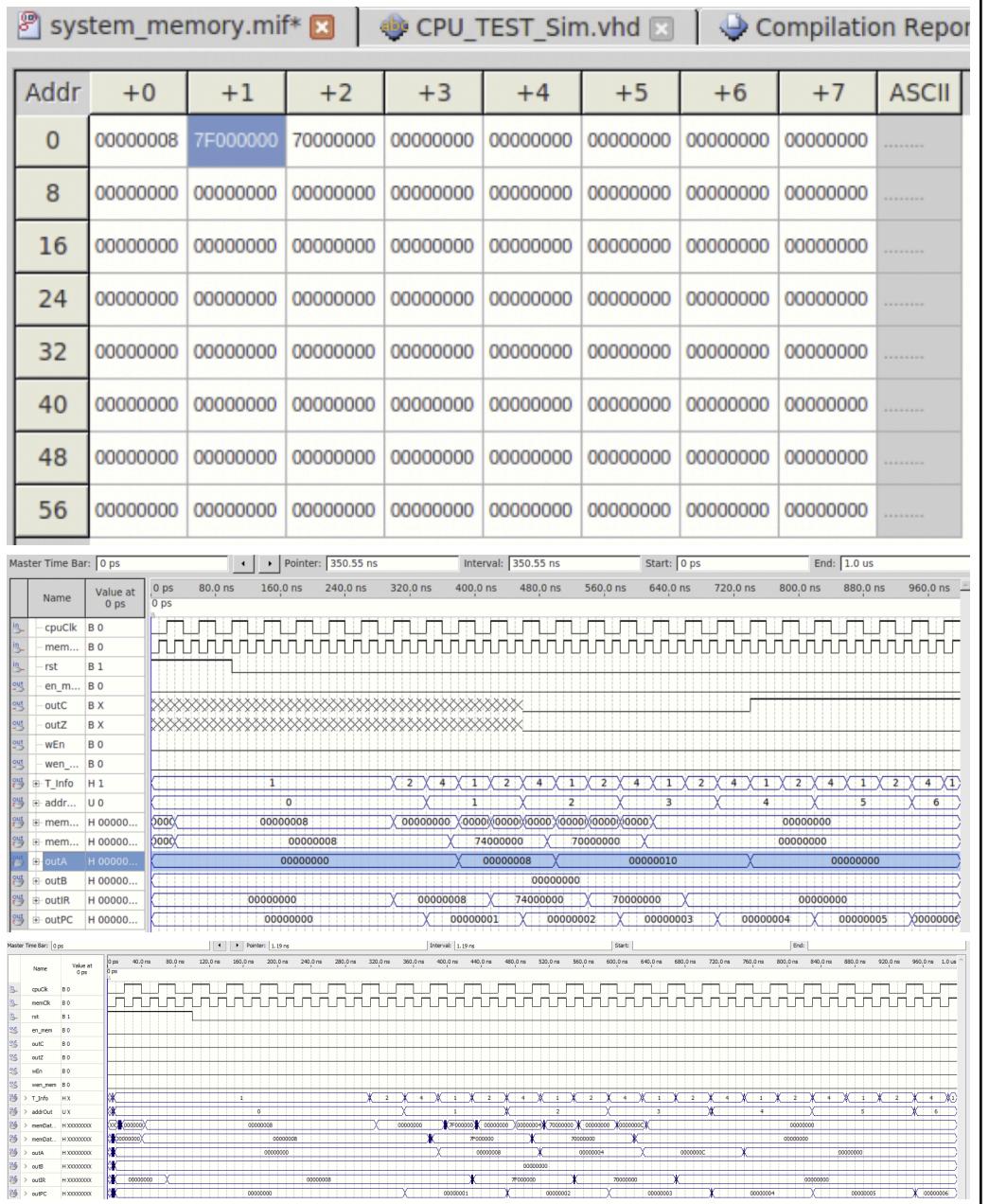
INCA



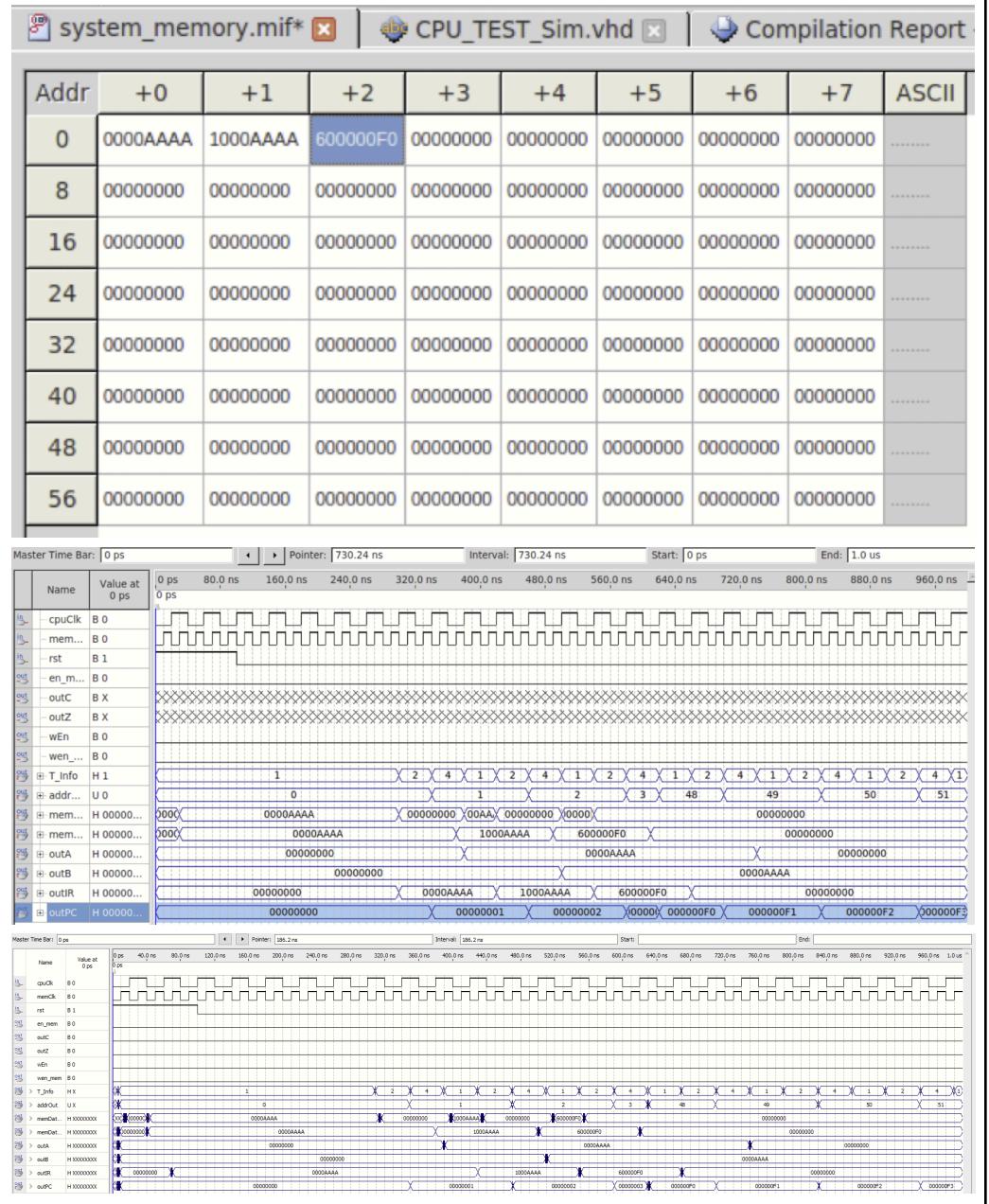
ROL



ROR



BEQ



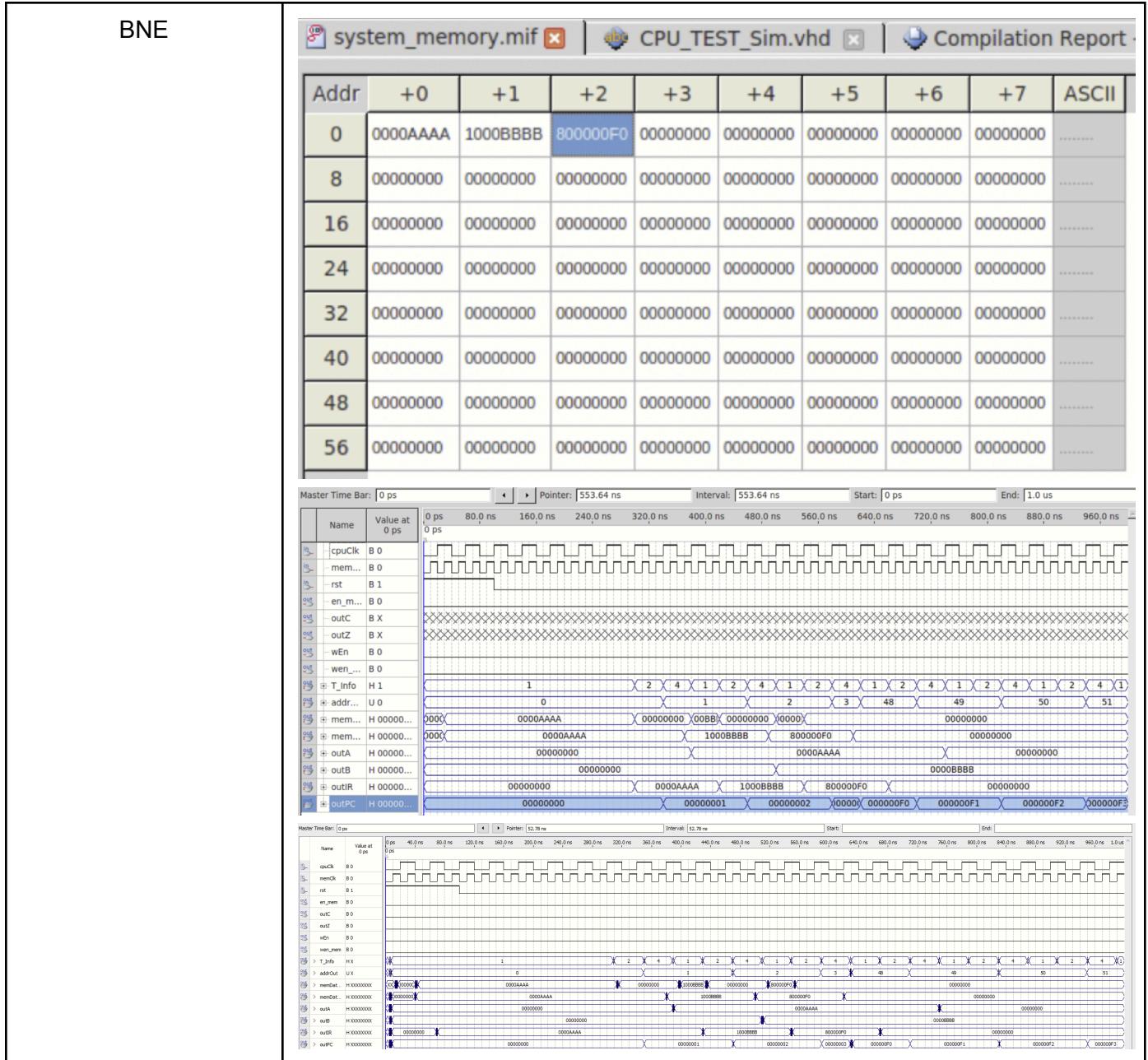


Table 1, Operations and their MIP, Functional Waveforms and Timing Waveforms

Table 1, Functional Waveforms are similar to Table 1, Timing Waveforms, the difference is a small delay, since the occurrence of this glitch can be attributed to the propagation delays within the FPGA device, as variations in input signals may not reach the logic element responsible for generating output signals simultaneously. Therefore, the timing diagram is giving the correct output as Table 1, Functional Waveforms.

4. Discussion:

In conclusion, the completion of Lab 6 marks the successful integration of the CPU reset circuitry and the assembly of the entire CPU system, including the data path and control unit.

Through thorough VHDL implementation and simulation, the reset circuit ensures proper initialization and synchronization of the CPU, laying the foundation for seamless operation. With the incorporation of all subsystems and adherence to specifications, the final CPU demonstrates functionality in accordance with the outlined objectives and requirements. Therefore, looking at the waveforms and the mif we see that the results were correct, resulting in a successful lab.

5. References:

1. Toronto Metropolitan University. (2024). COE 608 Lab 6 – The Complete CPU (Overall Project). Available at: <http://www.ee.ryerson.ca/~courses/coe608>.