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:

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

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY system_memory IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (5 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		data		: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
		wren		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
	);
END system_memory;

ARCHITECTURE SYN OF system_memory IS

	SIGNAL sub_wire0	: STD_LOGIC_VECTOR (31 DOWNTO 0);



	COMPONENT altsyncram
	GENERIC (
		clock_enable_input_a	: STRING;
		clock_enable_output_a	: STRING;
		init_file		: STRING;
		intended_device_family	: STRING;
		lpm_hint		: STRING;
		lpm_type		: STRING;
		numwords_a		: NATURAL;
		operation_mode		: STRING;
		operation_mode		: STRING;
		outdata_aclr_a		: STRING;
		outdata_reg_a		: STRING;
		power_up_uninitialized	: STRING;
		widthad_a		: NATURAL;
		width_a		: NATURAL;
		width_byteena_a		: NATURAL
	);
	PORT (
			address_a	: IN STD_LOGIC_VECTOR (5 DOWNTO 0),
			clock0	: IN STD_LOGIC ;
			data_a	: IN STD_LOGIC_VECTOR (31 DOWNTO 0);
			wren_a	: IN STD_LOGIC ;
			q_a	: OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
	);
	END COMPONENT;

BEGIN
	q	<= sub_wire0(31 DOWNTO 0);

	altsyncram_component : altsyncram
	GENERIC MAP (
		clock_enable_input_a => "BYPASS",
		clock_enable_output_a => "BYPASS",
		init_file => "system_memory.mif",
		intended_device_family => "Cyclone II",
		lpm_hint => "ENABLE_RUNTIME_MOD=NO",
		lpm_type => "altsyncram",
		numwords_a => 64,
		operation_mode => "SINGLE_PORT",
		outdata_aclr_a => "NONE",
		outdata_reg_a => "CLOCK0",
		power_up_uninitialized => "FALSE",
		widthad_a => 6,
		widthad_a => 6,
		width_a => 32,
		width_byteena_a => 1
	)
	PORT MAP (
		address_a => address,
		clock0 => clock,
		data_a => data,
		wren_a => wren,
		q_a => sub_wire0
	);


END SYN;
```

Figure 1: system_memory VHDL code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY register32 IS
PORT(
    d     :IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    ld    :IN STD_LOGIC;
    clr   :IN STD_LOGIC;
    clk   :IN STD_LOGIC;
    Q     :OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
);
END register32;

ARCHITECTURE Behaviour OF register32 IS
BEGIN
    PROCESS(ld, clr, clk)
    BEGIN
        IF (clr = '1')THEN
            Q <= (OTHERS => '0');
        ELSIF ((clk'EVENT and clk = '1') and (ld = '1')) THEN
            Q <= d;
        END IF;
    END PROCESS;
END Behaviour;
```

Figure 2: register32 VHDL code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pc is
port(
clr : in   std_logic;
clk : in   std_logic;
ld  : in   std_logic;
inc : in   std_logic;
d   : in   std_logic_vector(31 downto 0);
q   : out   std_logic_vector(31 downto 0)
);
end pc;

architecture Behavior of pc is
component add
port(
A   : in  std_logic_vector(31 downto 0);
B   : out std_logic_vector(31 downto 0)
);
end component;

component mux2to1
port (s   :  in std_logic;
w0, w1 :  in std_logic_vector(31 downto 0);
f      :  out std_logic_vector(31 downto 0)
);
end component;

component register32
port (
d   : in std_logic_vector(31 downto 0);
ld   : in std_logic;
clr   : in std_logic;
clk   : in std_logic;
Q   : out std_logic_vector(31 downto 0)
);
end component;

signal add_out : std_logic_vector(31 downto 0);
signal mux_out : std_logic_vector(31 downto 0);
signal q_out   : std_logic_vector(31 downto 0);
begin
add0: add port map(q_out, add_out);
mux0: mux2to1 port map(inc, d, add_out, mux_out);
reg0: register32 port map (mux_out, ld, clr, clk, q_out);
q <= q_out;
end Behavior;
```

Figure 3: pc VHDL code

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

entity mux4to1 is
port(
        s           : in  std_logic_vector(1 downto 0);
        X1,X2,X3,X4 :  in std_logic_vector(31 downto 0);
        f           : out std_logic_vector(31 downto 0));

end mux4to1;

architecture Behavior of mux4to1 is
begin
    with s select
        f<= X1 when "00",
            X2 when "01",
            X3 when "10",
            X4 when "11";
end Behavior;
```

Figure 4: mux4to1 VHDL code

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

entity mux2to1 is
port(
        s         : in  std_logic;
        w0, w1    :  in std_logic_vector(31 downto 0);
        f         : out std_logic_vector(31 downto 0));

end mux2to1;

architecture Behavior of mux2to1 is
begin
    with s select
        f<= w0 when '0',
            w1 when others;
end behavior;
```

Figure 5: mux2to1 VHDL code

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


entity fulladd is
port(
    Cin, x, y     : in std_logic;
    s, Cout       : out std_logic
    );
end fulladd;

architecture Behavior of fulladd is
begin
    s <= x xor y xor Cin;
    Cout <= (x and y) or (Cin and x) or (Cin and y);
end Behavior;
```

Figure 6: fulladd VHDL code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity data_mem is
port(
    clk   : in std_logic;
    addr  : in unsigned(7 downto 0);
    data_in  : in std_logic_vector(31 downto 0);
    wen   : in std_logic;
    en    : in std_logic;
    data_out : out std_logic_vector(31 downto 0)
);

end data_mem;

architecture Behavior of data_mem is
    type RAM is array (0 to 255) of std_logic_vector(31 downto 0);
    signal DATAMEM : RAM;
begin
    process(clk, en, wen)
    begin
        if(clk'event and clk='0') then
            if (en = '0') then
                data_out <= (others => '0');
            else
                if (wen = '0') then
                    data_out <= DATAMEM(to_integer(addr));
                end if;
                if (wen='1') then
                    DATAMEM(to_integer(addr)) <= data_in;
                    data_out <= (others => '0');
                end if;
            end if;
        end if;
    end process;
end Behavior;
```

Figure 7: data_mem VHDL code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity ALU is
    port(
        a  :    in  std_logic_vector(31 downto 0);
        b  :    in  std_logic_vector(31 downto 0);
        op :    in  std_logic_vector(2 downto 0);
        result  :   out  std_logic_vector(31 downto 0);
        zero :    out std_logic;
        Cout :    out std_logic
    );
end ALU;

architecture Behavior of ALU is
    component adder32
    port(
    Cin   :    in   std_logic;
    X,Y   :    in   std_logic_vector(31 downto 0);
    S     :    out  std_logic_vector(31 downto 0);
    Cout  :    out  std_logic
    );
    end component;

    signal result_s: std_logic_vector(31 downto 0):= (others=>'0');
    signal result_add: std_logic_vector(31 downto 0):= (others=>'0');
    signal result_sub: std_logic_vector(31 downto 0):= (others=>'0');
    signal cout_s: std_logic:='0';
    signal cout_add: std_logic:='0';
    signal cout_sub: std_logic:='0';
    signal zero_s: std_logic:='0';

begin
        add0 : adder32 port map (op(2), a, b, result_add, cout_add);
        sub0 : adder32 port map (op(2), a, not b, result_sub, cout_sub);

    process (a, b, op)
    begin
        case (op) is
            when "000" =>
                result_s<= a and b;
                cout_s <= '0';
            when "001" =>
                result_s<= a or b;
                cout_s <= '0';
            when "010" =>
                result_s<= result_add;
                cout_s <= cout_add;
            when "011" =>
                result_s<= b;
                cout_s <= '0';
            when "110" =>
                result_s<= result_sub;
                cout_s <= cout_sub;
            when "100" =>
                result_s<= a(30 downto 0) & '0';
                cout_s <= a(31);
            when "101" =>
                result_s<= '0' & a(31 downto 1);
                cout_s <= '0';
            when others =>
                result_s <= a;
                cout_s <= '0';
            end case;

            case (result_s) is
                when (others => '0') =>
                    zero_s <= '1';
                when others =>
                    zero_s <= '0';
            end case;
        end process;

        result <= result_s;
        cout <= cout_s;
        zero <= zero_s;
    end Behavior;
```

Figure 8: ALU VHDL code

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

entity adder32 is
port(
    Cin  :   in  std_logic;
    X,Y  :   in  std_logic_vector(31 downto 0);
    S    :   out std_logic_vector(31 downto 0);
    Cout :   out std_logic
    );
end adder32;

architecture Behavior of adder32 is
    component adder16
    port(
    Cin   :    in   std_logic;
    X,Y   :    in   std_logic_vector(15 downto 0);
    S     :    out  std_logic_vector(15 downto 0);
    Cout  :    out  std_logic
    );
    end component;

    signal C : std_logic;
begin
    stage0: adder16 port map (Cin, X(15 downto 0), Y(15 downto 0),  S(15 downto 0), C);
    stage1: adder16 port map (C,   X(31 downto 16),Y(31 downto 16), S(31 downto 16),Cout);
end Behavior;
```

Figure 9: adder32 VHDL code

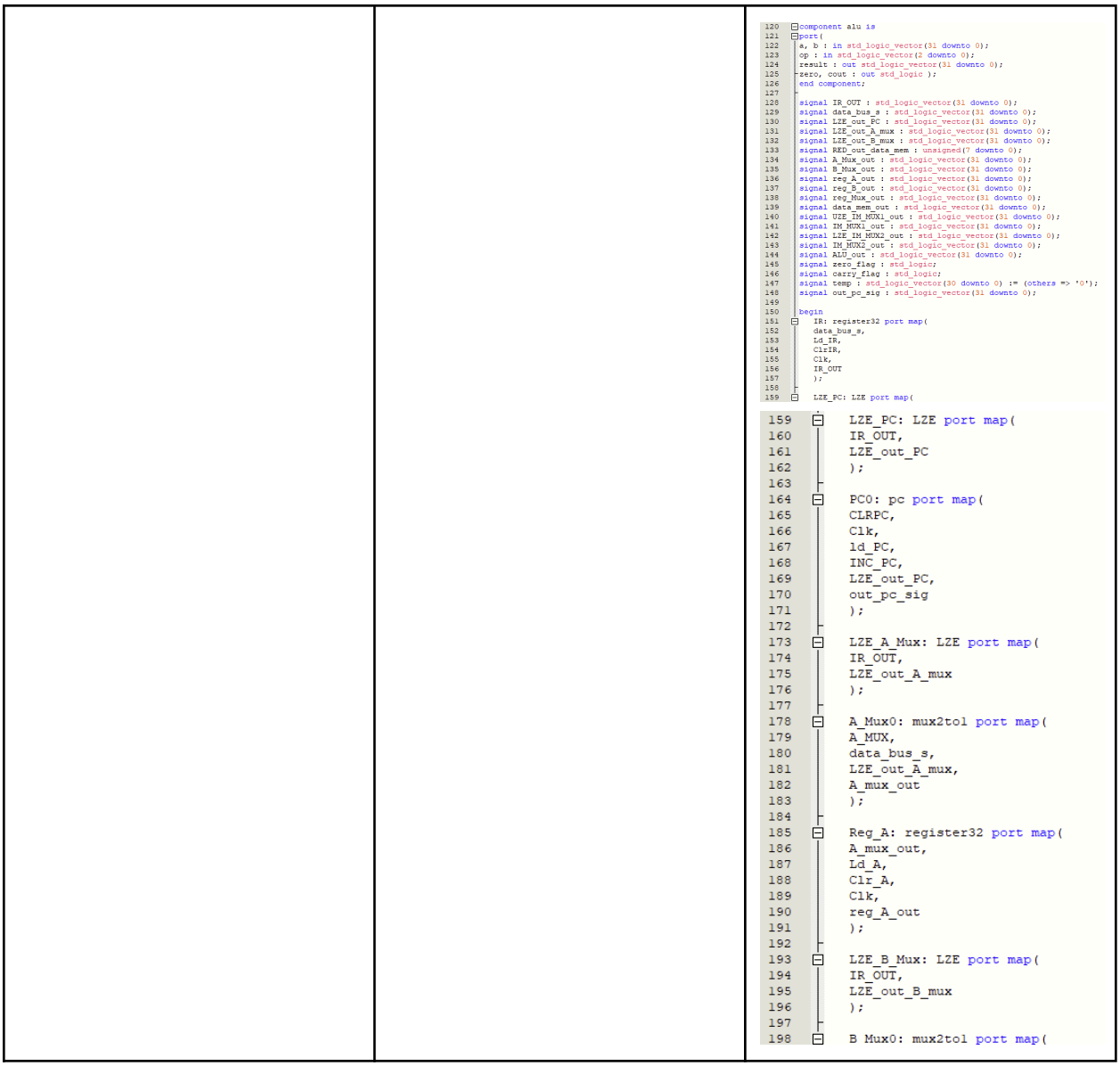
```vhdl
library ieee;
use ieee.std_logic_1164.all;


entity adder16 is
port(
    Cin    :    in    std_logic;
    X,Y    :    in    std_logic_vector(15 downto 0);
    S      :    out   std_logic_vector(15 downto 0);
    Cout   :    out   std_logic
    );
end adder16;

architecture Behavior of adder16 is
    component adder4
    port(
    Cin    :    in       std_logic;
    X,Y    :    in    std_logic_vector(3 downto 0);
    S      :    out   std_logic_vector(3 downto 0);
    Cout   :    out      std_logic
    );
    end component;

    signal C    :    std_logic_vector(1 to 3);
begin
    stage0:    adder4 port map (Cin, X(3 downto 0), Y(3 downto 0), S(3 downto 0), C(1));
    stage1:    adder4 port map (C(1), X(7 downto 4), Y(7 downto 4), S(7 downto 4), C(2));
    stage2:    adder4 port map (C(2), X(11 downto 8), Y(11 downto 8), S(11 downto 8), C(3));
    stage3:    adder4 port map (C(3), X(15 downto 12), Y(15 downto 12), S(15 downto 12), Cout);
    end Behavior;
```

Figure 10: adder16 VHDL code

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


entity adder4 is
port(
    Cin    :    in    std_logic;
    X,Y    :    in    std_logic_vector(3 downto 0);
    S      :    out   std_logic_vector(3 downto 0);
    Cout   :    out   std_logic
    );
end adder4;

architecture Behavior of adder4 is
    component fulladd
    port(
    Cin, x, y    :    in        std_logic;
    s, Cout      :    out    std_logic
    );
    end component;

    signal C        :    std_logic_vector (1 to 3);
begin
    stage0:    fulladd port map (Cin, X(0), Y(0), S(0), C(1));
    stage1:    fulladd port map (C(1), X(1), Y(1), S(1), C(2));
    stage2:    fulladd port map (C(2), X(2), Y(2), S(2), C(3));
    stage3:    fulladd port map (C(3), X(3), Y(3), S(3), Cout);
end Behavior;
```

Figure 11: adder4 VHDL code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity data_path is
port(
--Clock Signals
Clk, mClk : in std_logic;

--Memory Signals
WEN, EN : in std_logic;

--Register Control Signals
Clr_A, Ld_A : in std_logic;
Clr_B, Ld_B : in std_logic;
Clr_C, Ld_C : in std_logic;
Clr_Z, Ld_Z : in std_logic;
ClrPC, Ld_PC : in std_logic;
ClrIR, Ld_IR : in std_logic;

--Register Outputs
Out_A : out std_logic_vector(31 downto 0);
Out_B : out std_logic_vector(31 downto 0);
Out_C : out std_logic;
Out_Z : out std_logic;
Out_PC : out std_logic_vector(31 downto 0);
Out_IR : out std_logic_vector(31 downto 0);

--Special PC Inputs
Inc_PC : in std_logic;

--Address and Data Bus Signals
ADDR_OUT : out std_logic_vector(31 downto 0);
DATA_IN : in std_logic_vector(31 downto 0);
DATA_BUS, MEM_OUT, MEM_IN : out std_logic_vector(31 downto 0);
MEM_ADDR : out unsigned(7 downto 0);

--MUX Controls
DATA_MUX : in std_logic_vector(1 downto 0);
REG_MUX : in std_logic;
A_MUX, B_MUX : in std_logic;
A_MUX, B_MUX : in std_logic;
IM_MUX1 : in std_logic;
IM_MUX2 : in std_logic_vector(1 downto 0);

--ALU Operations
ALU_Op : in std_logic_vector(2 downto 0)
);
end entity;


architecture Behavior of data_path is

component data_mem is
port(
clk : in std_logic;
addr : in unsigned(7 downto 0);
data_in : in std_logic_vector(31 downto 0);
wen : in std_logic;
en : in std_logic;
data_out : out std_logic_vector(31 downto 0)
);
end component;

component register32 is
port (
d : in std_logic_vector(31 downto 0) ;
ld : in std_logic ;
clr : in std_logic ;
clk : in std_logic ;
Q : out std_logic_vector(31 downto 0)) ;
end component;

component pc is
port (
clr : in std_logic ;
clk : in std_logic ;
ld : in std_logic ;
inc : in std_logic ;
d : in std_logic_vector(31 downto 0) ;
q : out std logic vector(31 downto 0) ) ;
q : out std_logic_vector(31 downto 0) ) ;
end component;

component LZE is
port(
LZE_in : in std_logic_vector(31 downto 0);
LZE_out : out std_logic_vector(31 downto 0)
);
end component;

component UZE is
port(
UZE_in : in std_logic_vector(31 downto 0);
UZE_out : out std_logic_vector(31 downto 0)
);
end component;

component RED is
port(
RED_in : in std_logic_vector(31 downto 0);
RED_out : out unsigned(7 downto 0)
);
end component;

component mux2to1 is
port (
s : in std_logic ;
w0, w1 : std_logic_vector(31 downto 0) ;
f : out std_logic_vector(31 downto 0) ) ;
end component;

component mux4to1 is
port(
s : in std_logic_vector(1 downto 0);
X1, X2, X3, X4 : in std_logic_vector(31 downto 0);
f: out std_logic_vector(31 downto 0)
);
end component;

component alu is
```



```
component alu is
port(
a, b : in std_logic_vector(31 downto 0);
op : in std_logic_vector(2 downto 0);
result : out std_logic_vector(31 downto 0);
zero, cout : out std_logic );
end component;

signal IR_OUT : std_logic_vector(31 downto 0);
signal data_bus_s : std_logic_vector(31 downto 0);
signal LZE_out_PC : std_logic_vector(31 downto 0);
signal LZE_out_A_mux : std_logic_vector(31 downto 0);
signal LZE_out_B_mux : std_logic_vector(31 downto 0);
signal RED_out_data_mem : unsigned(7 downto 0);
signal A_Mux_out : std_logic_vector(31 downto 0);
signal B_Mux_out : std_logic_vector(31 downto 0);
signal reg_A_out : std_logic_vector(31 downto 0);
signal reg_B_out : std_logic_vector(31 downto 0);
signal reg_Mux_out : std_logic_vector(31 downto 0);
signal data_mem_out : std_logic_vector(31 downto 0);
signal LZE_IM_MUX1_out : std_logic_vector(31 downto 0);
signal IM_MUX1_out : std_logic_vector(31 downto 0);
signal LZE_IM_MUX2_out : std_logic_vector(31 downto 0);
signal IM_MUX2_out : std_logic_vector(31 downto 0);
signal ALU_out : std_logic_vector(31 downto 0);
signal zero_flag : std_logic;
signal carry_flag : std_logic;
signal temp : std_logic_vector(30 downto 0) := (others => '0');
signal out_pc_sig : std_logic_vector(31 downto 0);

begin
   IR: register32 port map(
   data_bus_s,
   Ld_IR,
   ClrIR,
   Clk,
   IR_OUT
   );

   LZE_PC: LZE port map(
```

```
   LZE_PC: LZE port map(
   IR_OUT,
   LZE_out_PC
   );

   PC0: pc port map(
   CLRPC,
   Clk,
   ld_PC,
   INC_PC,
   LZE_out_PC,
   out_pc_sig
   );

   LZE_A_Mux: LZE port map(
   IR_OUT,
   LZE_out_A_mux
   );

   A_Mux0: mux2to1 port map(
   A_MUX,
   data_bus_s,
   LZE_out_A_mux,
   A_mux_out
   );

   Reg_A: register32 port map(
   A_mux_out,
   Ld_A,
   Clr_A,
   Clk,
   reg_A_out
   );

   LZE_B_Mux: LZE port map(
   IR_OUT,
   LZE_out_B_mux
   );

   B_Mux0: mux2to1 port map(
```

```
198     B_Mux0: mux2to1 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     Clr_B,
209     Clk,
210     reg_B_out
211     );
212
213     Reg_Mux0: mux2to1 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     );
```

```
237   );
238
239   IM_MUX1a: mux2to1 port map(
240   IM_MUX1,
241   reg_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: mux4to1 port map(
252   IM_MUX2,
253   reg_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: mux4to1 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;
```

```
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

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

ENTITY cpu_test_sim IS
   PORT(
      cpuClk : in std_logic;
      memClk : in std_logic;
      rst : in std_logic;
      -- Debug data.
      outA, outB : out std_logic_vector(31 downto 0);
      outC, outZ : out std_logic;
      outIR : out std_logic_vector(31 downto 0);
      outPC : out std_logic_vector(31 downto 0);
      -- Processor-Inst Memory Interface.
      addrOut : out std_logic_vector(5 downto 0);
      wEn : out std_logic;
      memDataOut : out std_logic_vector(31 downto 0);
      memDataIn : out std_logic_vector(31 downto 0);
      -- Processor State
      T_Info : out std_logic_vector(2 downto 0);
      --data Memory Interface
      wen_mem, en_mem : out std_logic);
END cpu_test_sim;

ARCHITECTURE behavior OF cpu_test_sim IS
   COMPONENT system_memory
      PORT(
         address  : IN  STD_LOGIC_VECTOR (5 DOWNTO 0);
         clock    : IN  STD_LOGIC ;
         data     : IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
         wren     : IN  STD_LOGIC ;
         q        : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
   );
   END COMPONENT;

   COMPONENT cpu1
      PORT(
         clk      : in std_logic;
         mem_clk  : in std_logic;
         rst      : in std_logic;
         dataIn   : in std_logic_vector(31 downto 0);
         dataOut  : out std_logic_vector(31 downto 0);
         addrOut  : out std_logic_vector(31 downto 0);
         wEn : out std_logic;
         dOutA, dOutB : out std logic vector(31 downto 0);
         dOutA, dOutB : out std_logic_vector(31 downto 0);
         dOutC, dOutZ : out std_logic;
         dOutIR : out std_logic_vector(31 downto 0);
         dOutPC : out std_logic_vector(31 downto 0);
         outT : out std_logic_vector(2 downto 0);
         wen_mem, en_mem : out std_logic);
   END COMPONENT;

   signal cpu_to_mem: std_logic_vector(31 downto 0);
   signal mem_to_cpu: std_logic_vector(31 downto 0);
   signal add_from_cpu: std_logic_vector(31 downto 0);
   signal wen_from_cpu: std_logic;

BEGIN
 -- Component instantiations.
  main_memory : system_memory
     PORT MAP(
         address => add_from_cpu(5 downto 0),
         clock => memClk,
         data => cpu_to_mem,
         wren => wen_from_cpu,
         q => mem_to_cpu
      );
  main_processor : cpu1
     PORT MAP(
      clk => cpuClk,
      mem_clk => memClk,
      rst => rst,
      dataIn => mem_to_cpu,
      dataOut => cpu_to_mem,
      addrOut => add_from_cpu,
      wEn => wen_from_cpu,
      dOutA => outA,
      dOutB => outB,
      dOutC => outC,
      dOutZ => outZ,
      dOutIR => outIR,
      dOutPC => outPC,
      outT => T_Info,
      wen_mem => wen_mem,
      en_mem => en_mem
      );

  addrOut <= add_from_cpu(5 downto 0);
  wEn <= wen_from_cpu;
     wEn <= wen_from_cpu;
     memDataOut <= mem_to_cpu;
     memDataIn <= cpu_to_mem;
  END behavior;
```

Figure 13: cpu_test_sim VHDL code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity add is
port (A  :  in  std_logic_vector(31 downto 0);
      B  :  out std_logic_vector(31 downto 0);
      );
end add;

architecture Behavior of add is
begin
   B <= A + 1;
end Behavior;
```

Figure 14: add VHDL code

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY reset_circuit IS
   PORT (
         Reset       : IN  STD_LOGIC;
         Clk         : IN  STD_LOGIC;
         Enable_PD   : OUT STD_LOGIC := '1';
         Clr_PC      : OUT STD_LOGIC;
   );
END reset_circuit;

ARCHITECTURE Behavior OF reset_circuit IS
   TYPE clkNum IS (clk0, clk1, clk2, clk3);
   SIGNAL present_clk: clkNum;
BEGIN
   process (Clk) begin
      if rising_edge(Clk) then
         if Reset = '1' then
            Clr_PC <= '1';
            Enable_PD <= '0';
            present_clk <= clk0;
         elsif present_clk <= clk0 then
            present_clk <= clk1;
         elsif present_clk <= clk1 then
            present_clk <= clk2;
         elsif present_clk <= clk2 then
            present_clk <= clk3;
         elsif present_clk <= clk3 then
            Clr_PC <= '0';
            Enable_PD <= '1';
         end if;
      end if;
   end process;
END Behavior;
```

Figure 15: reset_circuit VHDL code

```
library ieee;
use ieee.std_logic_1164.ALL;

ENTITY Control_New IS
    PORT(
        clk, mclk           : IN  STD_LOGIC;
        enable              : IN  STD_LOGIC;
        statusC, statusZ    : IN  STD_LOGIC;
        INST                : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
        A_MUX, B_MUX        : OUT STD_LOGIC;
        IM_MUX1, REG_Mux    : OUT STD_LOGIC;
        IM_MUX2, DATA_Mux   : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        ALU_op              : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        inc_PC, ld_PC       : OUT STD_LOGIC;
        clr_IR              : OUT STD_LOGIC;
        ld_IR               : OUT STD_LOGIC;
        clr_A, clr_B, clr_C, clr_Z  : OUT STD_LOGIC;
        ld_A, ld_B, ld_C, ld_Z      : OUT STD_LOGIC;
        T                   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        wen, en             : OUT STD_LOGIC
    );
END Control_New;

ARCHITECTURE description OF Control_New IS
    TYPE STATETYPE IS (state_0, state_1, state_2);
    SIGNAL present_state: STATETYPE;
    SIGNAL Instruction_sig: STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL Instruction_sig2: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    Instruction_sig <= INST(31 DOWNTO 29);
    Instruction_sig2 <= INST(31 DOWNTO 24);

    ------- OPERATION DECODER --------
    PROCESS (present_state, INST, statusC, statusZ, enable, Instruction_sig, Instruction_sig2)
    BEGIN
        if enable = '1' then
            if present_state = state_0 then
                DATA_Mux <= "00"; --Fetch address of next instruction
                clr_IR <= '0';
                ld_IR <= '1';
                ld_PC <= '0';
                inc_PC <= '0';
                clr_A <= '0';
                ld_A <= '0';
                ld_B <= '0';
                clr_B <= '0';
                clr_C <= '0';
                ld_C <= '0';
                clr_Z <= '0';
                ld_Z <= '0';
                en <= '0';
                wen <= '1';

            elsif present_state = state_1 then
                clr_IR <= '0'; --INCREMENT PC COUNTER
                ld_IR <= '0';
                ld_IR <= '0';
                ld_PC <= '1';
                inc_PC <= '1';
                clr_A <= '0';
                ld_A <= '0';
                ld_B <= '0';
                clr_B <= '0';
                clr_C <= '0';
                ld_C <= '0';
                clr_Z <= '0';
                ld_Z <= '0';
                en <= '0';
                wen <= '0';

                if Instruction_sig = "0010" then --STA
                    clr_IR <= '0';
                    ld_IR <= '0';
                    ld_PC <= '1';
                    inc_PC <= '1';
                    clr_A <= '0';
                    ld_A <= '0';
                    ld_B <= '0';
                    clr_B <= '0';
                    clr_C <= '0';
                    ld_C <= '0';
                    clr_Z <= '0';
                    ld_Z <= '0';
                    REG_Mux <= '0';
                    DATA_Mux <= "00";
                    en <= '1';
                    wen <= '1';

                elsif Instruction_sig = "0011" then --STB
                    clr_IR <= '0';
                    --ld_Z <= '0';
                    ld_IR <= '0';
                    ld_PC <= '1';
                    inc_PC <= '1';
                    clr_A <= '0';
                    ld_A <= '0';
                    ld_B <= '0';
                    clr_B <= '0';
                    clr_C <= '0';
                    ld_C <= '0';
                    clr_Z <= '0';
                    ld_Z <= '0';
                    REG_Mux <= '1';
                    DATA_Mux <= "00";
                    en <= '1';
                    wen <= '1';

                elsif Instruction_sig = "1001" then --LDA
                    clr_IR <= '0';
                    ld_IR <= '0';
                    ld_PC <= '1';
                    ld_PC <= '1';
                    inc_PC <= '1';
                    clr_A <= '0';
                    ld_A <= '1';
                    ld_B <= '0';
                    clr_B <= '0';
                    clr_C <= '0';
                    ld_C <= '0';
                    clr_Z <= '0';
                    ld_Z <= '0';
                    A_Mux <= '0';
                    DATA_Mux <= "01";
                    en <= '1';  --EN
                    wen <= '0'; --WEN

                elsif Instruction_sig = "1010" then --LDB
                    clr_IR <= '0';
                    ld_IR <= '0';
                    ld_PC <= '1';
                    inc_PC <= '1';
                    clr_A <= '0';
                    ld_A <= '0';
                    ld_B <= '1';
                    clr_B <= '0';
                    clr_C <= '0';
                    ld_C <= '0';
                    clr_Z <= '0';
                    ld_Z <= '0';
                    B_Mux <= '0';
                    DATA_Mux <= "01";
                    en <= '1';  --EN
                    wen <= '0'; --WEN
                end if;  --END IF FOR LOAD STORE IN STAGE 1

            elsif present_state = state_2 then
                if Instruction_sig = "0101" then --JUMP
                    clr_IR <= '0';
                    ld_IR <= '0';
                    ld_PC <= '1';
                    inc_PC <= '0';
                    clr_A <= '0';
                    ld_A <= '0';
                    ld_B <= '0';
                    clr_B <= '0';
                    clr_C <= '0';
                    ld_C <= '0';
                    clr_Z <= '0';
                    ld_Z <= '0';

                elsif Instruction_sig = "0110" then --BEQ
                    clr_IR <= '0';
                    ld_IR <= '0';
                    ld_PC <= '1';
                    inc_PC <= '0';
                    clr_A <= '0';
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY cpu1 IS
    PORT (
        clk : IN STD_LOGIC;
        mem_clk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        dataIn : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        dataOut : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        addrOut : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        dOutA : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        dOutB : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        dOutC : OUT STD_LOGIC;
        dOutZ : OUT STD_LOGIC;
        dOutIR : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        dOutPC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        wEn : OUT STD_LOGIC;
        outT : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        wen_mem : OUT STD_LOGIC;
        en_mem : OUT STD_LOGIC
    );
END cpu1;

ARCHITECTURE description OF cpu1 IS
    COMPONENT Data_Path IS
        PORT (
            Clk, mclk : IN STD_LOGIC; -- clock Signal
            --Memory Signals
            WEN, EN : IN STD_LOGIC;

            Clr_A, Ld_A : IN STD_LOGIC;
            Clr_B, Ld_B : IN STD_LOGIC;
            Clr_C, Ld_C : IN STD_LOGIC;
            Clr_Z, Ld_Z : IN STD_LOGIC;
            ClrPC, Ld_PC : IN STD_LOGIC;
            ClrIR, Ld_IR : IN STD_LOGIC;

            Out_A : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
            Out_B : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
            Out_C : OUT STD_LOGIC;
            Out_Z : OUT STD_LOGIC;
            Out PC : OUT STD LOGIC VECTOR(31 DOWNTO 0);
```

```
        Out_PC : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        Out_IR : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        -- Special inputs to PC.
        Inc_PC : IN STD_LOGIC;
        -- Address and Data Bus signals for debugging.
        ADDR_OUT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        DATA_IN : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        DATA_BUS, MEM_OUT, MEM_IN : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        MEM_ADDR : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        -- Various MUX controls.
        DATA_Mux : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        REG_Mux : IN STD_LOGIC;
        A_MUX, B_MUX : IN STD_LOGIC;
        IM_MUX1 : IN STD_LOGIC;
        IM_MUX2 : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        -- ALU Operations.
        ALU_Op : IN STD_LOGIC_VECTOR(2 DOWNTO 0)
    );
END COMPONENT;

COMPONENT Control_NEW IS
    PORT (
        clk, mclk : IN STD_LOGIC;
        enable : IN STD_LOGIC;
        statusC, statusZ : IN STD_LOGIC;
        INST : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        A_Mux, B_Mux : OUT STD_LOGIC;
        IM_MUX1, REG_Mux : OUT STD_LOGIC;
        IM_MUX2, DATA_Mux : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        ALU_op : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        inc_PC, ld_PC : OUT STD_LOGIC;
        clr_IR : OUT STD_LOGIC;
        ld_IR : OUT STD_LOGIC;
        clr_A, clr_B, clr_C, clr_Z : OUT STD_LOGIC;
        ld_A, ld_B, ld_C, ld_Z : OUT STD_LOGIC;
        T : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        wen, en : OUT STD_LOGIC
    );
END COMPONENT;

COMPONENT reset_circuit IS
    PORT (
        Reset : IN STD_LOGIC;
```

```
        Reset : IN STD_LOGIC;
        Clk : IN STD_LOGIC;
        Enable_PO : OUT STD_LOGIC;
        Clr_PC : OUT STD_LOGIC
    );
END COMPONENT;

SIGNAL dp_mux1, dp_clrA, dp_ldA, dp_clrB, dp_ldB, dp_clrC, dp_ldC, dp_clrZ,
dp_ldZ, memWEN, memEN, dp_mux4, dp_mux5 : STD_LOGIC;
SIGNAL mux_data, reg, enpd, irlc, irld, pinc, pclr, pcld, out0, out1, out7, outf : STD_LOGIC;
SIGNAL outIR : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL alu : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL dp_mux2, dp_muxData : STD_LOGIC_VECTOR(1 DOWNTO 0);

BEGIN
    dat : data_path
    PORT MAP(
        Clk => clk,
        mClk => mem_clk,
        WEN => memWEN,
        EN => memEN,
        Clr_A => dp_clrA,
        Ld_A => dp_ldA, Clr_B => dp_clrB, Ld_B => dp_ldB, Clr_C => dp_clrC, Ld_C
        dp_ldC, Clr_Z => dp_clrZ, Ld_Z => dp_ldZ,
        ClrPC => pclr, Ld_PC => pcld, ClrIR => irlc, Ld_IR => irld,
        Out_A => dOutA, Out_B => dOutB, Out_C => out0, Out_Z => out1, Out_PC =>
        dOutPC, Out_IR => outIR, Inc_PC => pinc,
        ADDR_OUT => addrOut, DATA_IN => dataIn, DATA_BUS => dataOut, DATA_Mux
        dp_muxData, REG_Mux => reg, A_MUX => dp_muxA, B_MUX => dp_muxB, IM_MUX1 =>
        dp_mux1, IM_MUX2 => dp_mux2,
        ALU_Op => alu
    );

    control_unit : Control_NEW
    PORT MAP(
        clk => clk, mclk => mem_clk, enable => enpd, statusC => out0, statusZ =>
        out1, INST => outIR,

        A_Mux => dp_muxA, B_Mux => dp_muxB, IM_MUX1 => dp_mux1, REG_Mux => reg,
        IM_MUX2 => dp_mux2, DATA_Mux => dp_muxData,
        ALU_op => alu, inc_PC => pinc, ld_PC => pcld, clr_IR => irlc, ld_IR =>
        irld,
        clr_A => dp_clrA, clr_B => dp_clrB, clr_C => dp_clrC, clr_Z => dp_clrZ,
```

```
        clr_A => dp_clrA, clr_B => dp_clrB, clr_C => dp_clrC, clr_Z => dp_clrZ,
        ld_A => dp_ldA, ld_B => dp_ldB, ld_C => dp_ldC, ld_Z => dp_ldZ,
        T => outT, wen => memWEN, en => memEN
    );

    reset : reset_circuit
    PORT MAP(
        Reset => rst,
        Clk => clk,
        Enable_PO => enpd,
        Clr_PC => pclr
    );
    dOutC <= out0;
    dOutZ <= out1;
    dOutIR <= outIR;
    wen_mem <= out7;
    en_mem <= outf;

END description;
```

Figure 17: cpu1 VHDL code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity UZE is
    port(
        UZE_in   :  in std_logic_vector(31 downto 0);
        UZE_out  :  out std_logic_vector(31 downto 0)
    );
end entity;

architecture Behavior of UZE is
    signal zeros: std_logic_vector(15 downto 0) := (others => '0');
begin
    UZE_out <= UZE_in(15 downto 0) & zeros;
end Behavior;
```

Figure 18: UZE VHDL code

```
            clr_A <= '0';
            ld_A <= '0';
            ld_B <= '0';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';

        elsif Instruction_sig = "1000" then --BNE
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '1';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '0';
            ld_B <= '0';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';

        elsif Instruction_sig = "1001" then --LDA
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '0';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '1';
            ld_B <= '0';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';
            A_Mux <= '0';
            DATA_Mux <= "01";
            en <= '1';
            wen <= '0';

        elsif Instruction_sig = "1010" then --LDB
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '0';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '0';
            ld_B <= '1';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';
            B_Mux <= '0';

            B_Mux <= '0';
            DATA_Mux <= "01";
            en <= '1';
            wen <= '0';

        elsif Instruction_sig = "0010" then --STA
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '0';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '0';
            ld_B <= '0';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';
            REG_Mux <= '0';
            DATA_Mux <= "00";
            en <= '1';
            wen <= '1';

        elsif Instruction_sig = "0011" then --STB
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '0';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '0';
            ld_B <= '0';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';
            REG_Mux <= '1';
            DATA_Mux <= "00";
            en <= '1';
            wen <= '1';

        elsif Instruction_sig = "0000" then --LDAI
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '0';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '1';
            ld_B <= '0';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';
            A_Mux <= '1';

            A_Mux <= '1';

        elsif Instruction_sig = "0000" then --LDBI
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '0';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '0';
            ld_B <= '1';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';
            B_Mux <= '1';

        elsif Instruction_sig = "0100" then --LUI
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '0';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '1';
            ld_B <= '0';
            clr_B <= '1';
            clr_C <= '0';
            ld_C <= '0';
            clr_Z <= '0';
            ld_Z <= '0';
            ALU_op <= "001";
            A_Mux <= '0';
            DATA_Mux <= "10";
            IM_MUX1 <= '1';

        elsif Instruction_sig2 = "01111001" then --ANDI
            clr_IR <= '0';
            ld_IR <= '0';
            ld_PC <= '0';
            inc_PC <= '0';
            clr_A <= '0';
            ld_A <= '1';
            ld_B <= '0';
            clr_B <= '0';
            clr_C <= '0';
            ld_C <= '1';
            clr_Z <= '0';
            ld_Z <= '1';
            ALU_op <= "000";
            A_Mux <= '0';
            DATA_Mux <= "10";
            IM_MUX1 <= '0';
            IM_MUX2 <= "01";

        elsif Instruction_sig2 = "01111110" then --DECA
```

```
        elsif Instruction_sig2 = "01111110" then --DECA
           clr_IR <= '0';
           ld_IR <= '0';
           ld_PC <= '0';
           inc_PC <= '0';
           clr_A <= '0';
           ld_A <= '1';
           ld_B <= '0';
           clr_B <= '0';
           clr_C <= '0';
           ld_C <= '1';
           clr_Z <= '0';
           ld_Z <= '1';
           ALU_op <= "110";
           A_Mux <= '0';
           DATA_Mux <= "10";
           IM_MUX1 <= '0';
           IM_MUX2 <= "10";

        elsif Instruction_sig2 = "01110000" then --ADD
           clr_IR <= '0';
           ld_IR <= '0';
           ld_PC <= '0';
           inc_PC <= '0';
           clr_A <= '0';
           ld_A <= '1';
           ld_B <= '0';
           clr_B <= '0';
           clr_C <= '0';
           ld_C <= '1';
           clr_Z <= '0';
           ld_Z <= '1';
           ALU_op <= "010";
           A_Mux <= '0';
           DATA_Mux <= "10";
           IM_MUX1 <= '0';
           IM_MUX2 <= "00";

        elsif Instruction_sig2 = "01110010" then --SUB
           clr_IR <= '0';
           ld_IR <= '0';
           ld_PC <= '0';
           inc_PC <= '0';
           clr_A <= '0';
           ld_A <= '1';
           ld_B <= '0';
           clr_B <= '0';
           clr_C <= '0';
           ld_C <= '1';
           clr_Z <= '0';
           ld_Z <= '1';
           ALU_op <= "110";
           A_Mux <= '0';
           DATA_Mux <= "00";
           IM_MUX1 <= '0';
           IM_MUX2 <= "00";

         elsif Instruction_sig2 = "01110011" then --INCA
           clr_IR <= '0';
           ld_IR <= '0';
           ld_PC <= '0';
           inc_PC <= '0';
           clr_A <= '0';
           ld_A <= '1';
           ld_B <= '0';
           clr_B <= '0';
           clr_C <= '0';
           ld_C <= '1';
           clr_Z <= '0';
           ld_Z <= '1';
           ALU_op <= "010";
           A_Mux <= '0';
           DATA_Mux <= "10";
           IM_MUX1 <= '0';
           IM_MUX2 <= "10";

        elsif Instruction_sig2 = "01111011" then --AND
           clr_IR <= '0';
           ld_IR <= '0';
           ld_PC <= '0';
           inc_PC <= '0';
           clr_A <= '0';
           ld_A <= '1';
           ld_B <= '0';
           clr_B <= '0';
           clr_C <= '0';
           ld_C <= '1';
           clr_Z <= '0';
           ld_Z <= '1';
           ALU_op <= "000";
           A_Mux <= '0';
           DATA_Mux <= "10";
           IM_MUX1 <= '0';
           IM_MUX2 <= "00";

        elsif Instruction_sig2 = "01110001" then --ADDI
           clr_IR <= '0';
           ld_IR <= '0';
           ld_PC <= '0';
           inc_PC <= '0';
           clr_A <= '0';
           ld_A <= '1';
           ld_B <= '0';
           clr_B <= '0';
           clr_C <= '0';
           ld_C <= '1';
           clr_Z <= '0';
           ld_Z <= '1';
           ALU_op <= "010";
```

```
                ALU_op <= "010";
                A_Mux <= '0';
                DATA_Mux <= "10";
                IM_MUX1 <= '0';
                IM_MUX2 <= "01";

            elsif Instruction_sig2 = "01111101" then --ORI
                clr_IR <= '0';
                ld_IR <= '0';
                ld_PC <= '0';
                inc_PC <= '0';
                clr_A <= '0';
                ld_A <= '1';
                ld_B <= '0';
                clr_B <= '0';
                clr_C <= '0';
                ld_C <= '1';
                clr_Z <= '0';
                ld_Z <= '1';
                ALU_op <= "001";
                A_Mux <= '0';
                DATA_Mux <= "10";
                IM_MUX1 <= '0';
                IM_MUX2 <= "01";

            elsif Instruction_sig2 = "01110100" then --ROL
                clr_IR <= '0';
                ld_IR <= '0';
                ld_PC <= '0';
                inc_PC <= '0';
                clr_A <= '0';
                ld_A <= '1';
                ld_B <= '0';
                clr_B <= '0';
                clr_C <= '0';
                ld_C <= '1';
                clr_Z <= '0';
                ld_Z <= '1';
                ALU_op <= "100";
                A_Mux <= '0';
                DATA_Mux <= "10";
                IM_MUX1 <= '0';

            elsif Instruction_sig2 = "01111111" then --ROR
                clr_IR <= '0';
                ld_IR <= '0';
                ld_PC <= '0';
                inc_PC <= '0';
                clr_A <= '0';
                ld_A <= '1';
                ld_B <= '0';
                clr_B <= '0';
                clr_C <= '0';
                ld_C <= '1';
                clr_Z <= '0';
                ld_Z <= '0';
                ALU_op <= "101";
                A_Mux <= '0';
                DATA_Mux <= "10";
                IM_MUX1 <= '0';

            elsif Instruction_sig2 = "01110101" then --CLR_A
                clr_IR <= '0';
                ld_IR <= '0';
                ld_PC <= '0';
                inc_PC <= '0';
                clr_A <= '1';
                ld_A <= '0';
                ld_B <= '0';
                clr_B <= '0';
                clr_C <= '0';
                ld_C <= '0';
                clr_Z <= '0';
                ld_Z <= '0';

            elsif Instruction_sig2 = "01110110" then --CLR_B
                clr_IR <= '0';
                ld_IR <= '0';
                ld_PC <= '0';
                inc_PC <= '0';
                clr_A <= '0';
                ld_A <= '0';
                ld_B <= '0';
                clr_B <= '1';
                clr_C <= '0';
                ld_C <= '0';
                clr_Z <= '0';
                ld_Z <= '0';

            elsif Instruction_sig2 = "01110111" then --CLR_C
                clr_IR <= '0';
                ld_IR <= '0';
                ld_PC <= '0';
                inc_PC <= '0';
                clr_A <= '0';
                ld_A <= '0';
                ld_B <= '0';
                clr_B <= '0';
                clr_C <= '1';
                ld_C <= '0';
                clr_Z <= '0';
                ld_Z <= '0';

            elsif Instruction_sig2 = "01111000" then --CLR_Z
                clr_IR <= '0';
                ld_IR <= '0';
                ld_PC <= '0';
                inc_PC <= '0';
                clr_A <= '0';
                clr_A <= '0';
                ld_A <= '0';
                ld_B <= '0';
                clr_B <= '0';
                clr_C <= '0';
                ld_C <= '0';
                clr_Z <= '1';
                ld_Z <= '0';

            elsif Instruction_sig2 = "01111010" then --TSTZ
                if (statusZ = '1') then
                    clr_IR <= '0'; --INCREMENT PC COUNTER
                    ld_IR <= '0';
                    ld_PC <= '1';
                    inc_PC <= '1';
                    clr_A <= '0';
                    ld_A <= '0';
                    ld_B <= '0';
                    clr_B <= '0';
                    clr_C <= '0';
                    ld_C <= '0';
                    clr_Z <= '0';
                    ld_Z <= '0';
                end if;

            elsif Instruction_sig2 = "01111100" then --TSTC
                if (statusC = '1') then
                    clr_IR <= '0'; --INCREMENT PC COUNTER
                    ld_IR <= '0';
                    ld_PC <= '1';
                    inc_PC <= '1';
                    clr_A <= '0';
                    ld_A <= '0';
                    ld_B <= '0';
                    clr_B <= '0';
                    clr_C <= '0';
                    ld_C <= '0';
                    clr_Z <= '0';
                    ld_Z <= '0';
                end if;
            end if;  --For state 2 Ops
        end if;
    end if;  --For Enable
END process;
-------- STATE MACHINE --------
PROCESS (clk, enable)
begin
    if enable = '1' then
        if rising_edge (clk) then
            if present_state = state_0 then present_state <= state_1;
            elsif present_state = state_1 then present_state <= state_2;
            else present_state <= state_0;
            end if;
        end if;
    else present_state <= state_0;
```

```
596         else present_state <= state_0;
597       end if;
598     END process;
599
600     WITH present_state select
601       T  <= "001" when state_0,
602             "010" when state_1,
603             "100" when state_2,
604             "001" when others;
605   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      RED_out <= unsigned (RED_in(7 downto 0));
15  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      LZE_out <= zeros & LZE_in(15 downto 0);
16  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 <= 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.

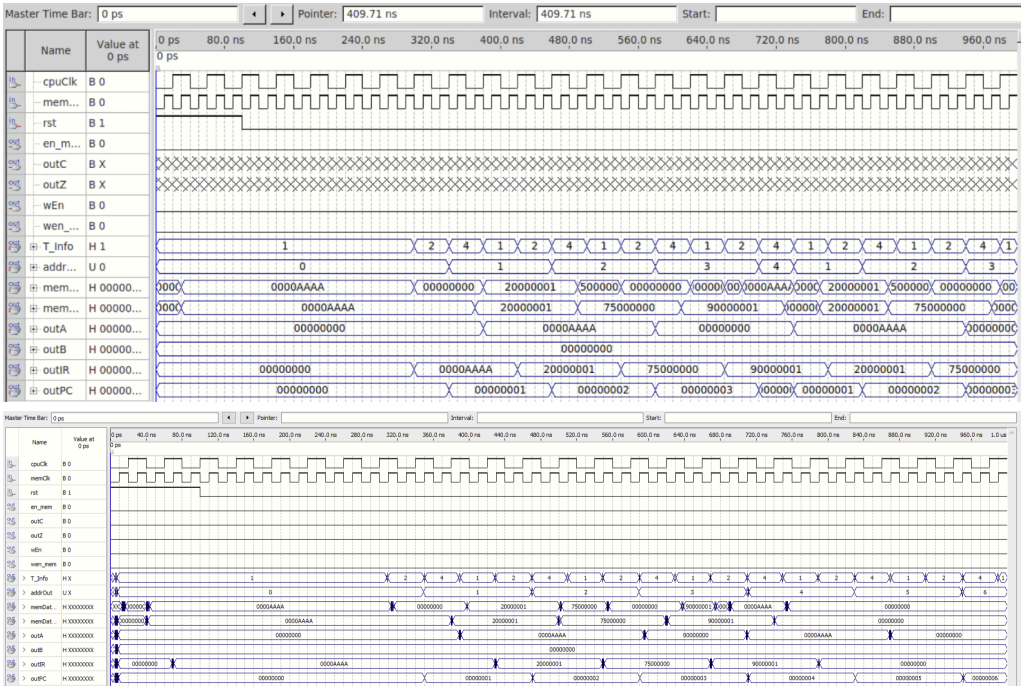
3. Results:

Operations	Figures of MIP, Functional, and Timing Waveforms

LDAI, STA, CLRA, LDA

CPU_TEST_Sim.vhd | system_memory.mif

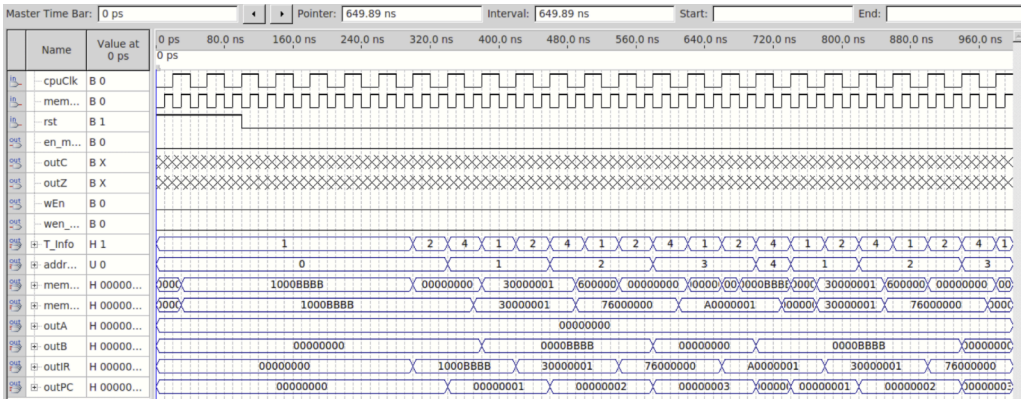
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0000AAAA	20000001	75000000	90000001	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

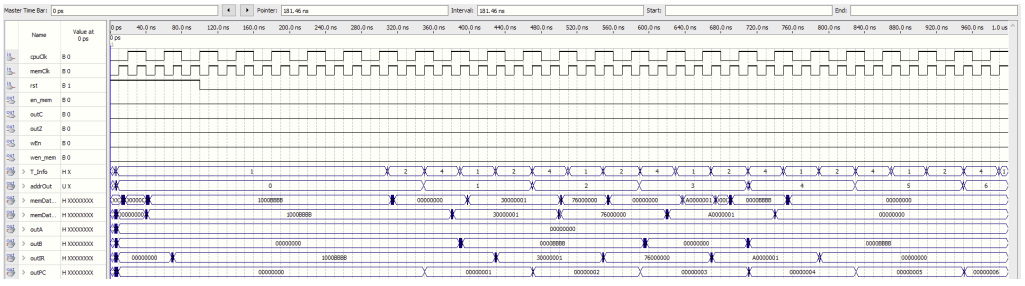


LDBI, STB, CLRB, LDB

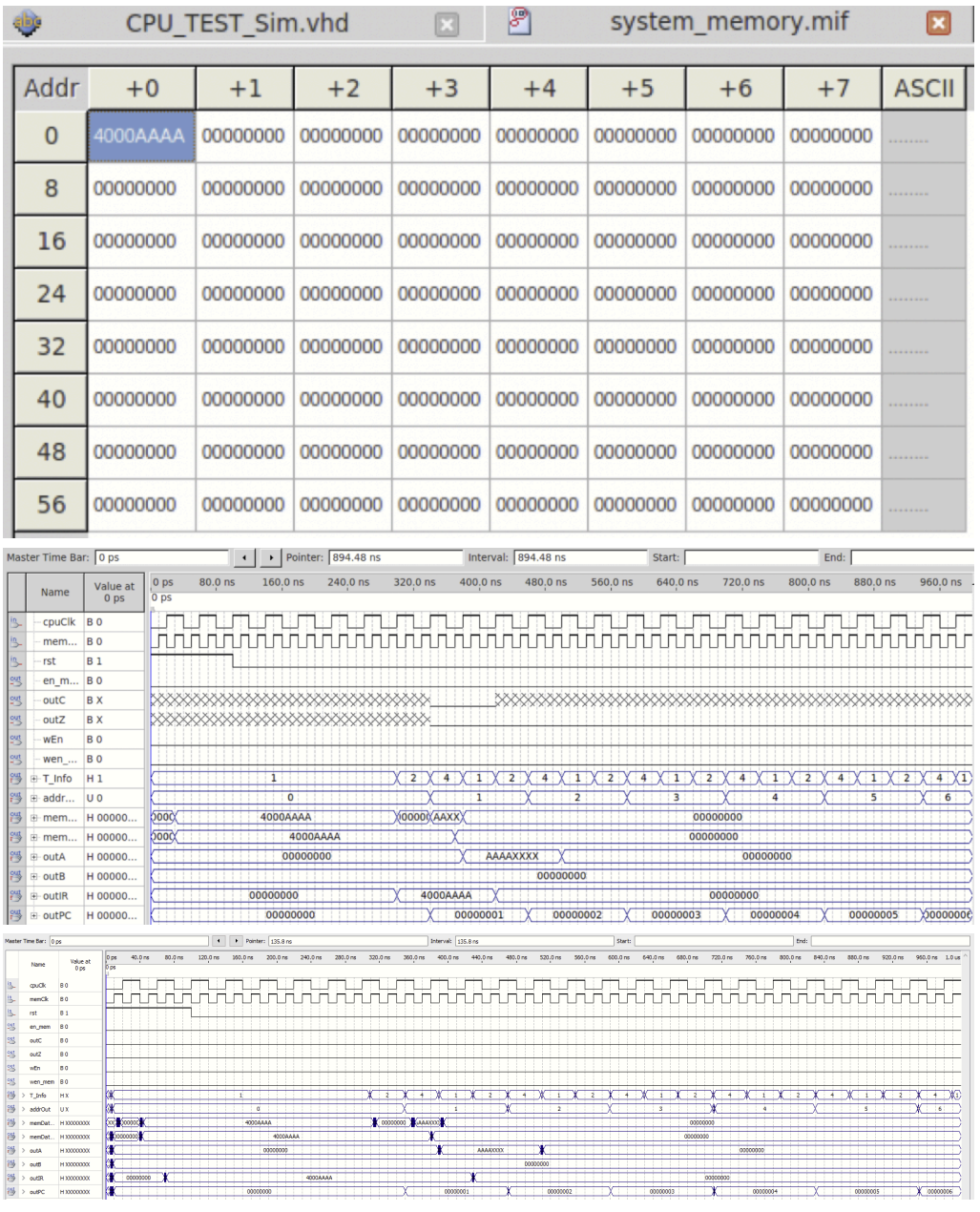
CPU_TEST_Sim.vhd | system_memory.mif*

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	1000BBBB	30000001	76000000	A0000001	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....





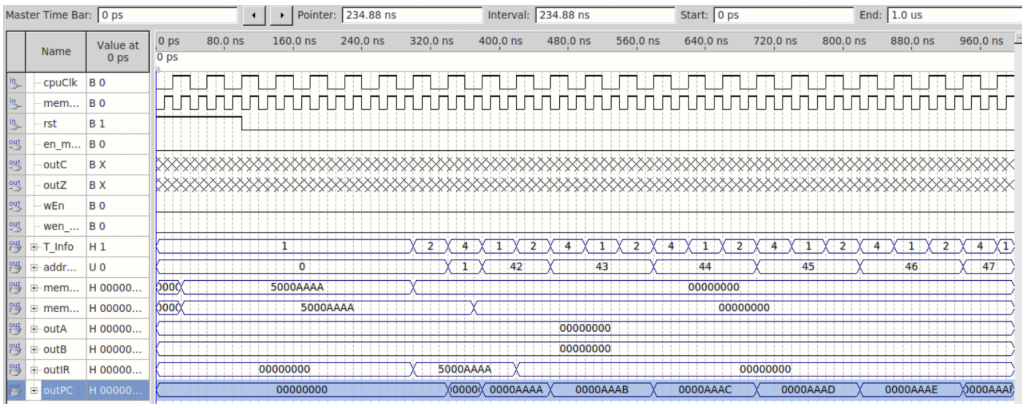
LUI

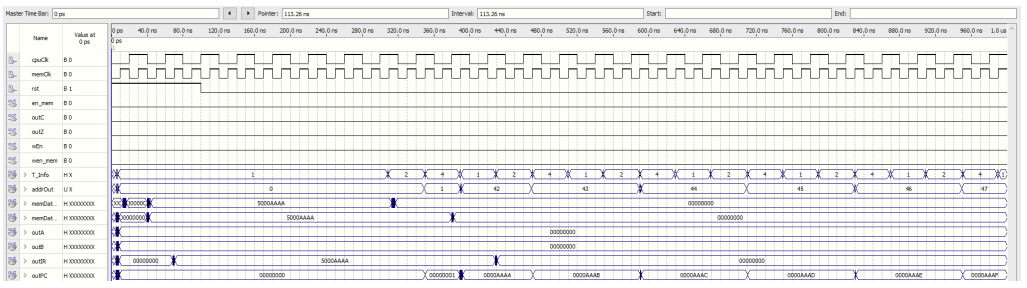

CPU_TEST_Sim.vhd
system_memory.mif
Addr +0 +1 +2 +3 +4 +5 +6 +7 ASCII
0 4000AAAA 00000000 00000000 00000000 00000000 00000000 00000000 00000000
8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
16 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
24 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
32 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
40 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
48 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
56 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Master Time Bar: 0 ps
Pointer: 894.48 ns
Interval: 894.48 ns
Start:
End:
Name
Value at 0 ps
cpuClk
mem...
rst
en_m...
outC
outZ
wEn
wen_...
T_Info
addr...
mem...
mem...
outA
outB
outIR
outPC

JMP

system_memory.mif | CPU_TEST_Sim.vhd | Compilation Report

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	5000AAAA	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

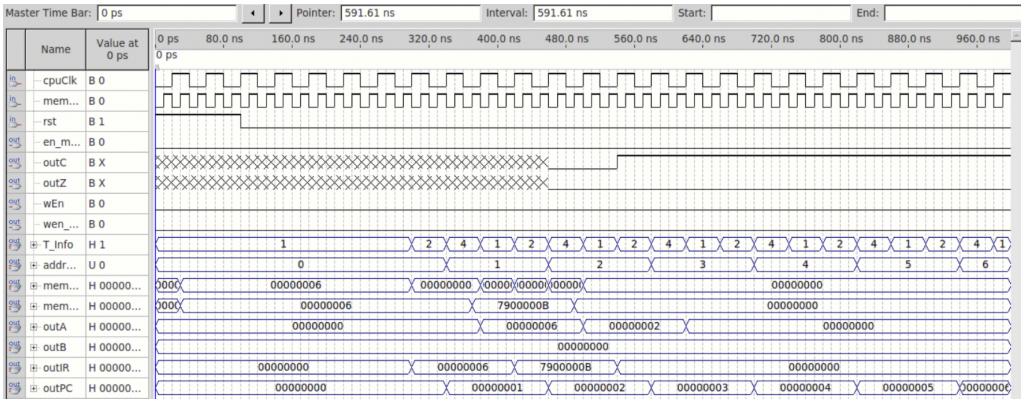


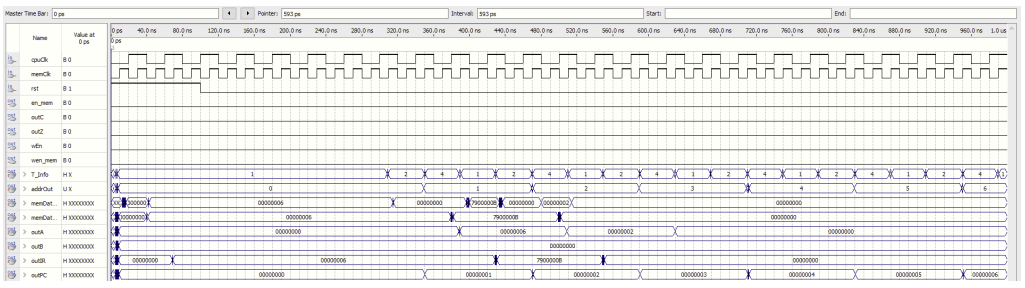


ANDI

system_memory.mif | CPU_TEST_Sim.vhd | Compilation Report

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000006	7900000B	00000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

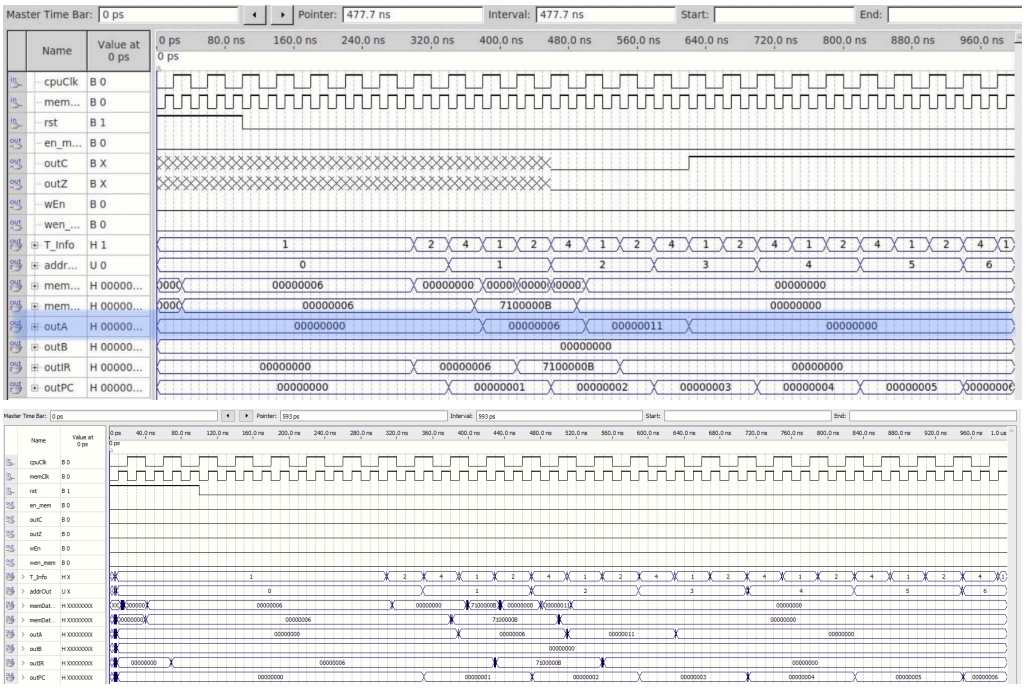




ADDI

system_memory.mif* | CPU_TEST_Sim.vhd | Compilation Repor

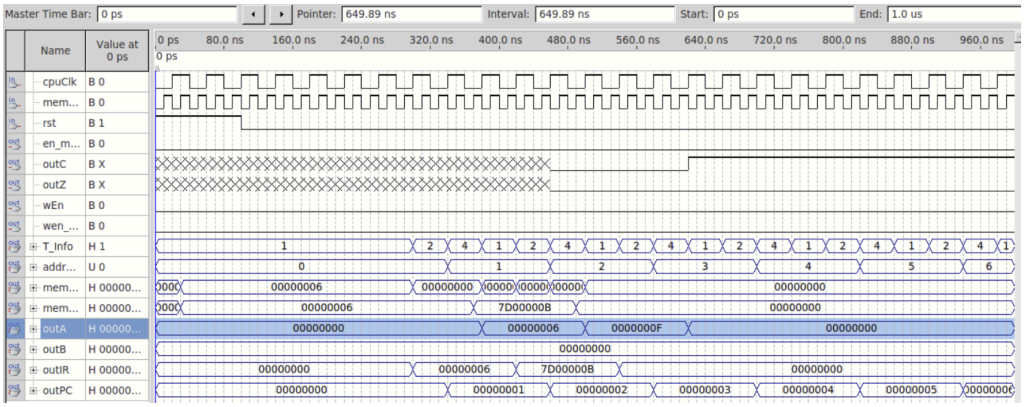
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000006	7100000B	00000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

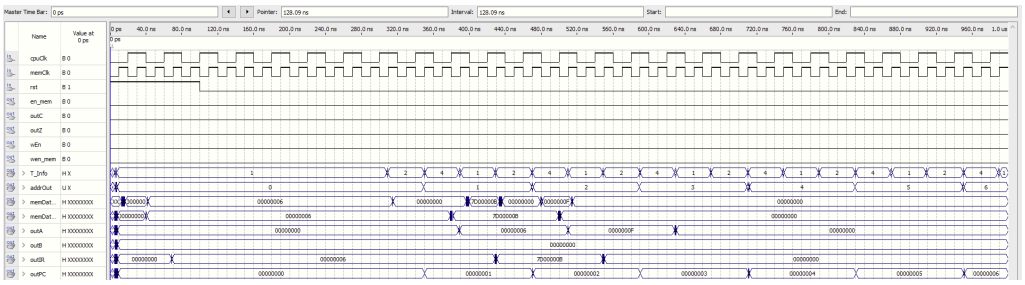


ORI

system_memory.mif | CPU_TEST_Sim.vhd | Compilation Report

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000006	7D00000B	00000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

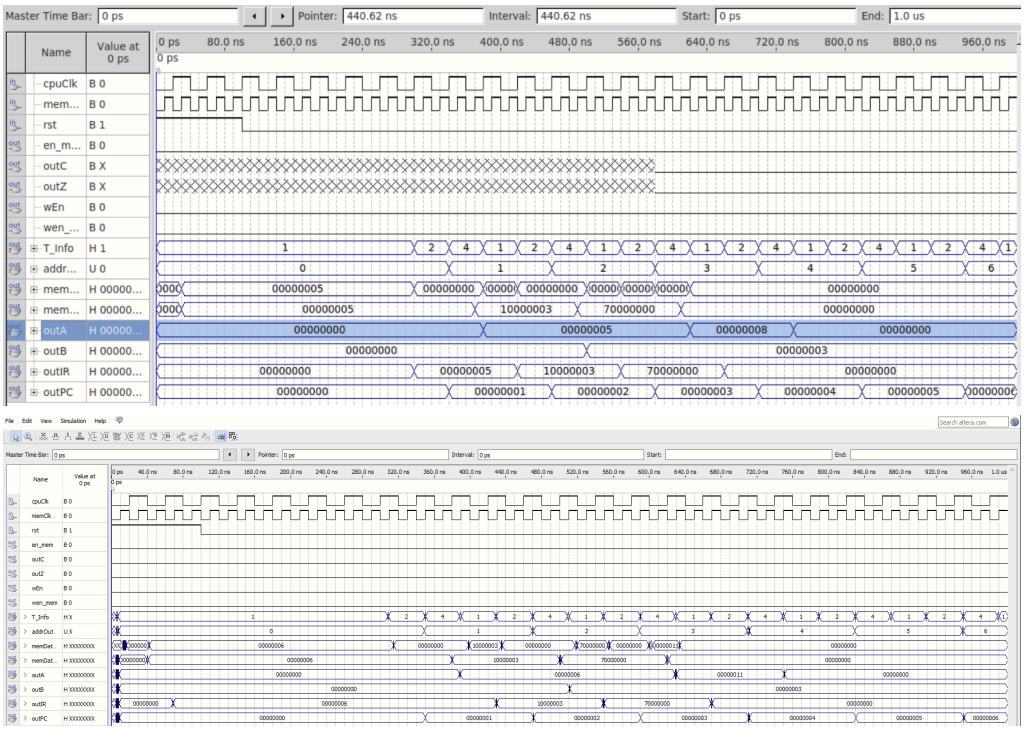




ADD

system_memory.mif | CPU_TEST_Sim.vhd | Compilation Report

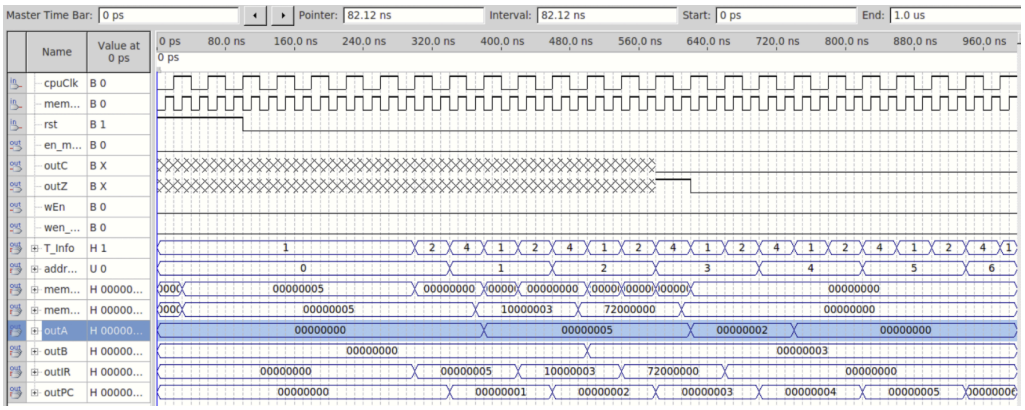
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000005	10000003	70000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

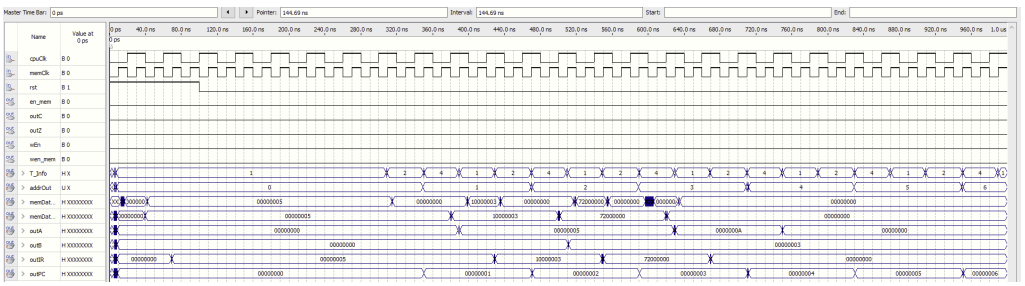


SUB

system_memory.mif* | CPU_TEST_Sim.vhd | Compilation Repo

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000005	10000003	72000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....

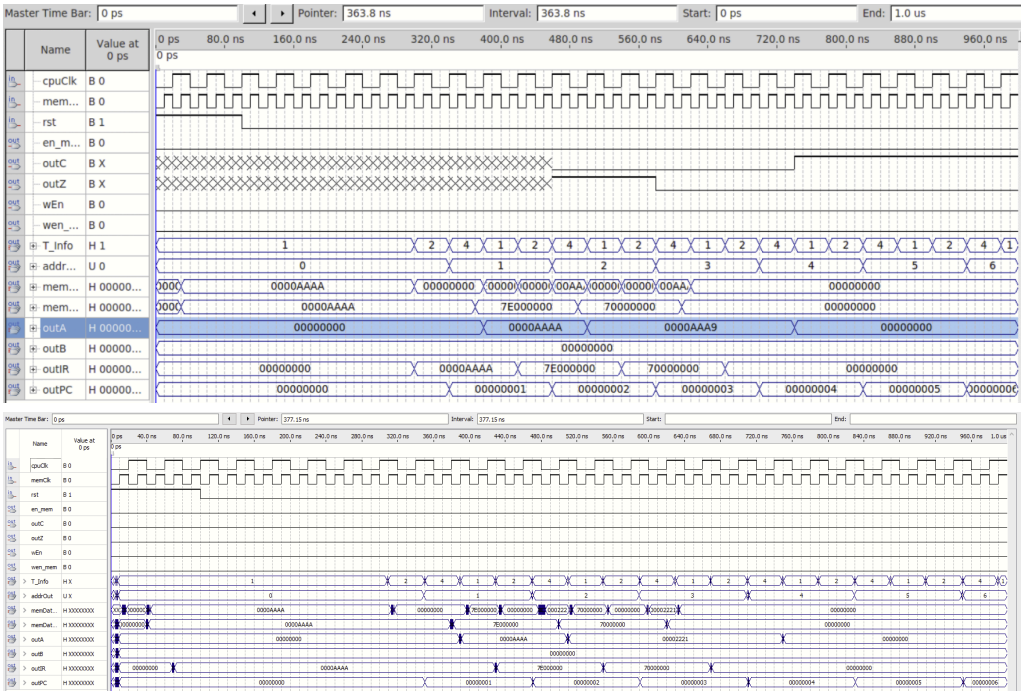




DECA

system_memory.mif | CPU_TEST_Sim.vhd | Compilation Report

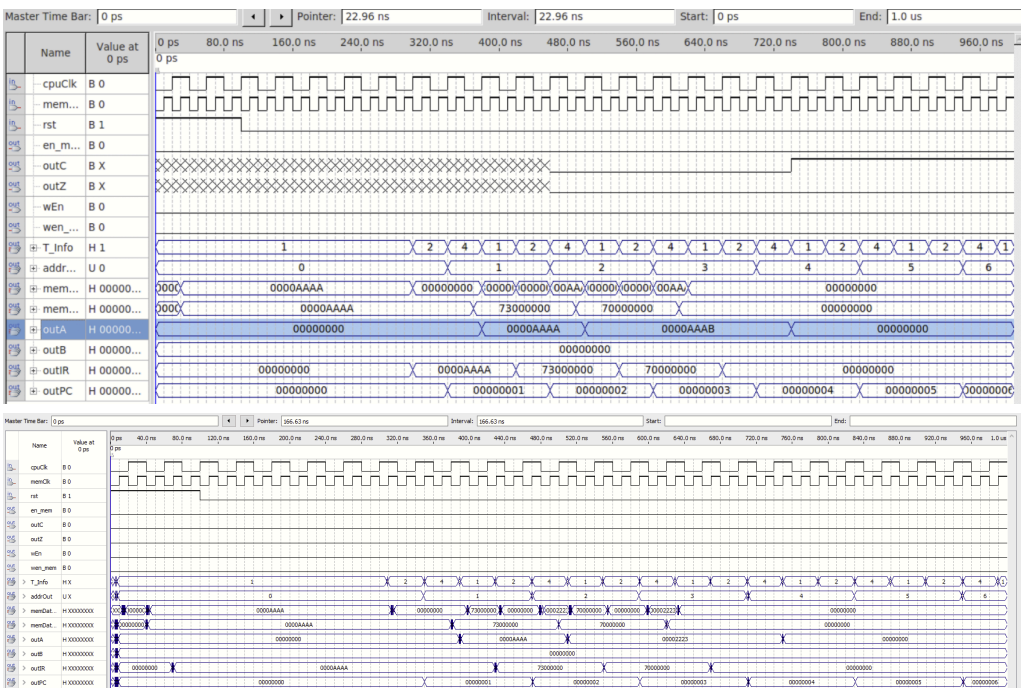
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0000AAAA	7E000000	70000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....



INCA

system_memory.mif* | CPU_TEST_Sim.vhd | Compilation Repor

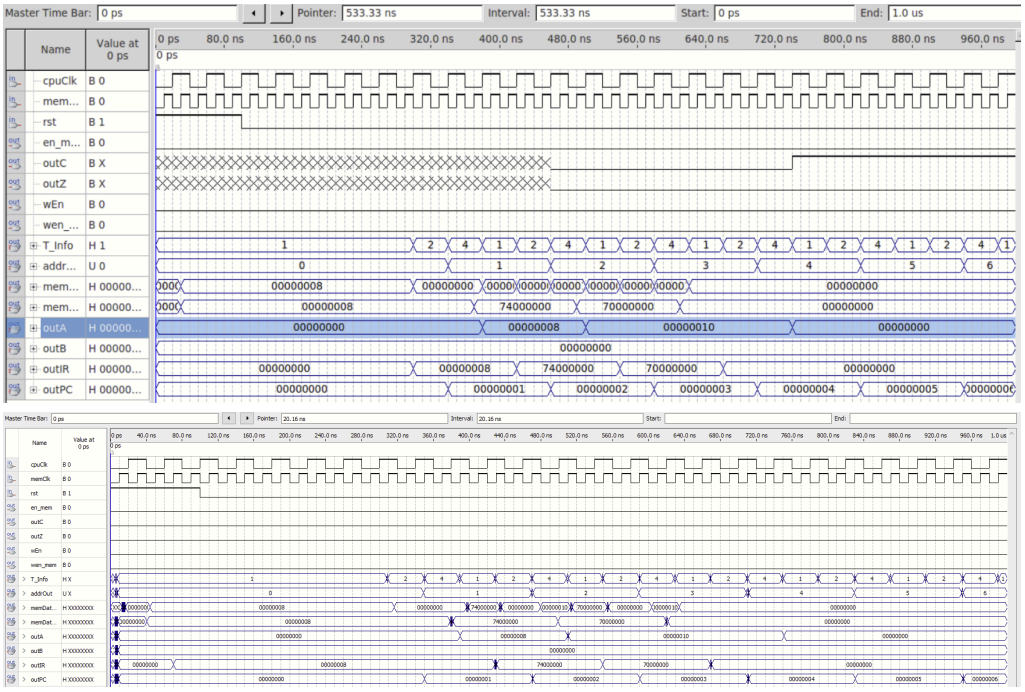
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0000AAAA	73000000	70000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....



ROL

system_memory.mif | CPU_TEST_Sim.vhd | Compilation Report

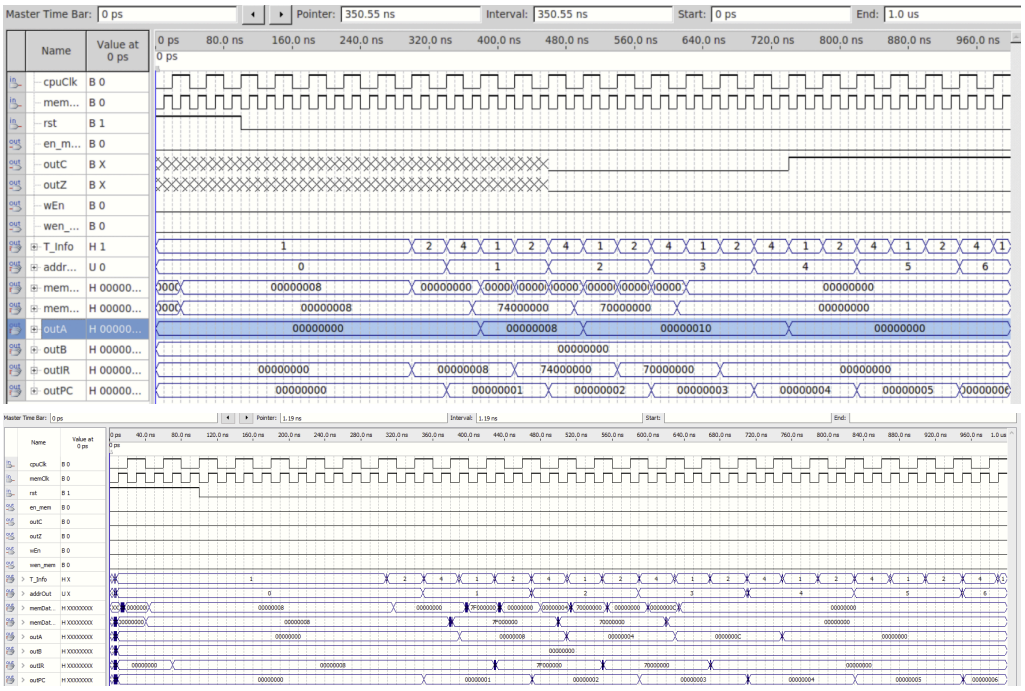
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000008	74000000	70000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....



ROR

system_memory.mif* | CPU_TEST_Sim.vhd | Compilation Repor

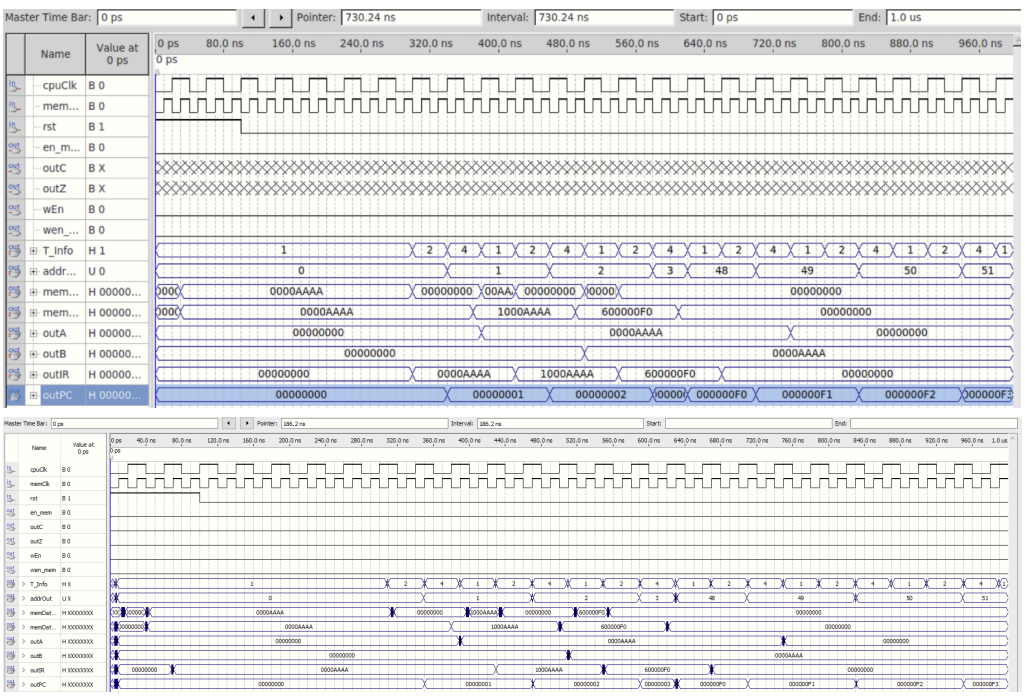
Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000008	7F000000	70000000	00000000	00000000	00000000	00000000	00000000	.....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	.....



BEQ

system_memory.mif* | CPU_TEST_Sim.vhd | Compilation Report

Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0000AAAA	1000AAAA	600000F0	00000000	00000000	00000000	00000000	00000000	....
8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
16	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
24	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
32	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....
56	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	....



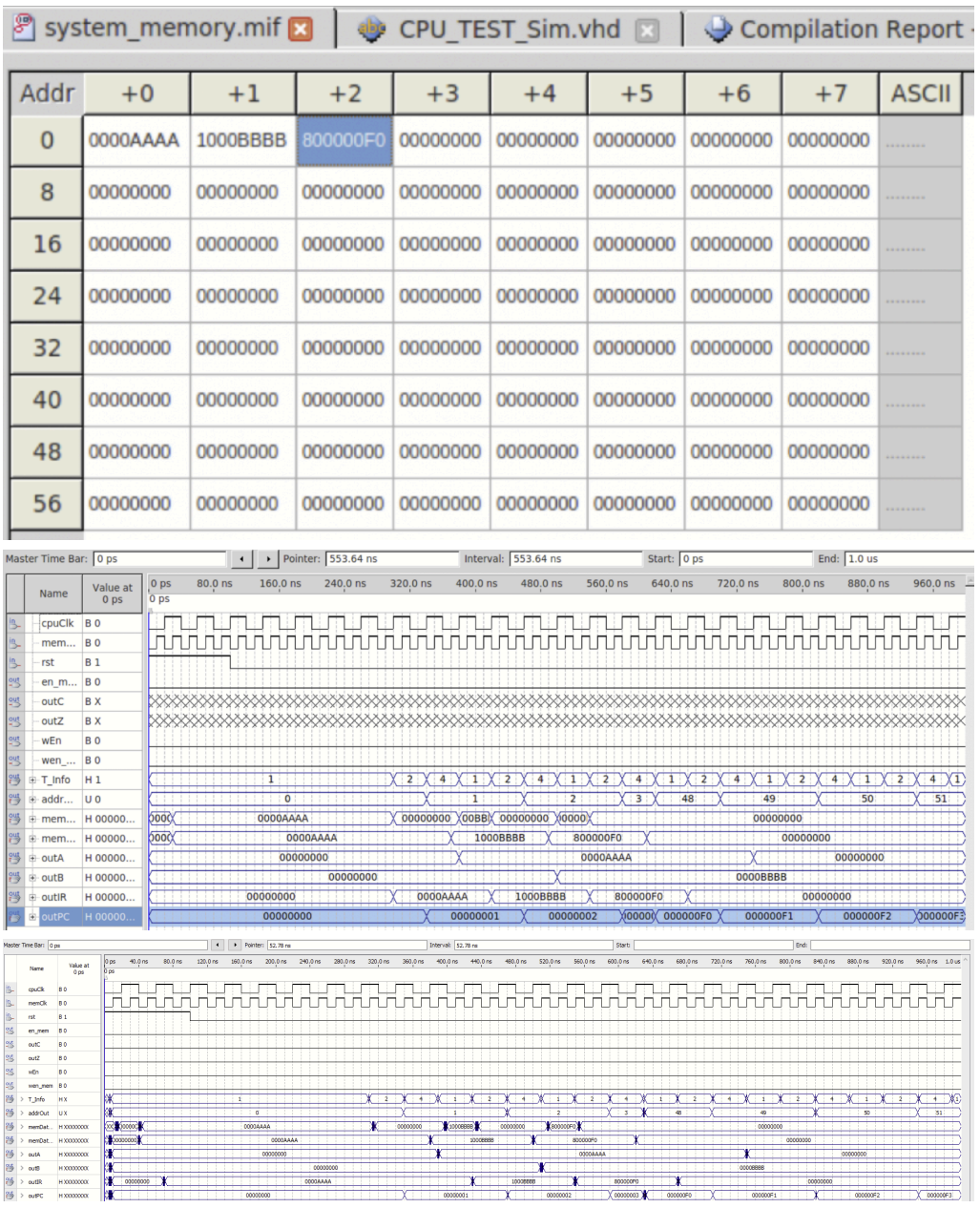
BNE	

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.