华南理工大学

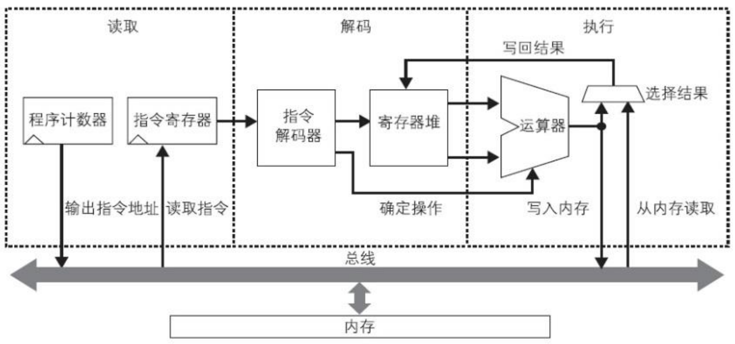
South China University of Technology

Quartus

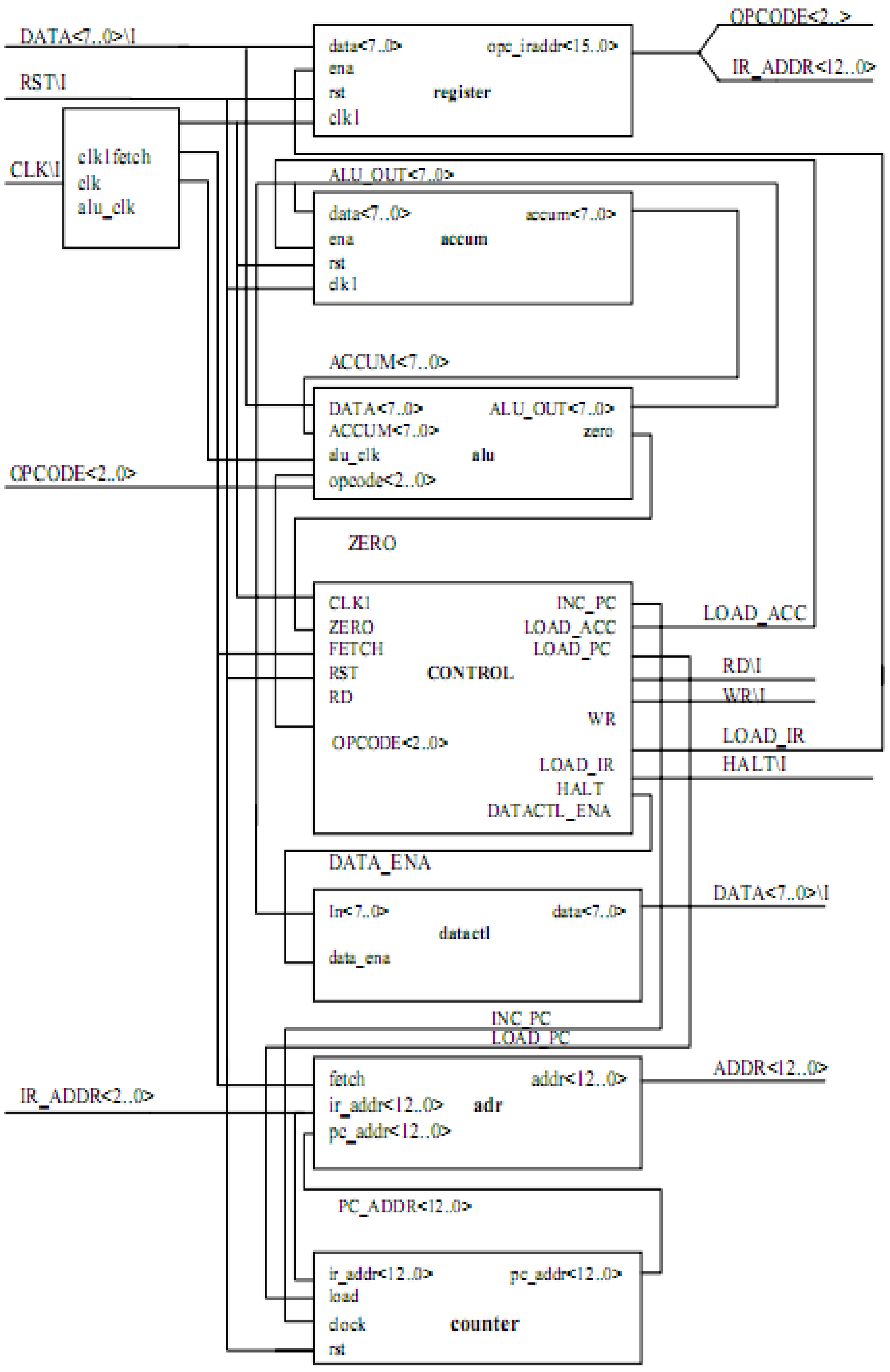
➢

➢

➢

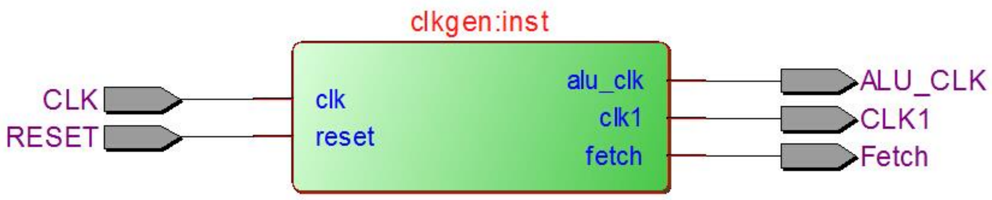

读取
解码
执行
写回结果
程序计数器
指令寄存器
指令解码器
寄存器堆
运算器
选择结果
输出指令地址
读取指令
确定操作
写入内存
从内存读取
总线
内存

35 te



DATA<7..0>\I
RST\I
CLK\I
clk1fetch
clk
alu_clk
data<7..0>
ena
rst
clk1
register
opc_iraddr<15..0>
OPCODE<2..>
IR_ADDR<12..0>
ALU_OUT<7..0>
data<7..0>
ena
rst
clk1
accum
accum<7..0>
ACCUM<7..0>
DATA<7..0>
ACCUM<7..0>
alu_clk
opcode<2..0>
alu
ALU_OUT<7..0>
zero
OPCODE<2..0>
ZERO
CLK1
ZERO
FETCH
RST
RD
OPCODE<2..0>
CONTROL
INC_PC
LOAD_ACC
LOAD_PC
WR
LOAD_IR
HALT
DATACTL_ENA
LOAD_ACC
RD\I
WR\I
LOAD_IR
HALT\I
DATA_ENA
In<7..0>
data_ena
datactl
data<7..0>
DATA<7..0>\I
INC_PC
LOAD_PC
fetch
ir_addr<12..0>
pc_addr<12..0>
adr
addr<12..0>
ADDR<12..0>
IR_ADDR<2..0>
PC_ADDR<12..0>
ir_addr<12..0>
load
clock
rst
counter
pc_addr<12..0>

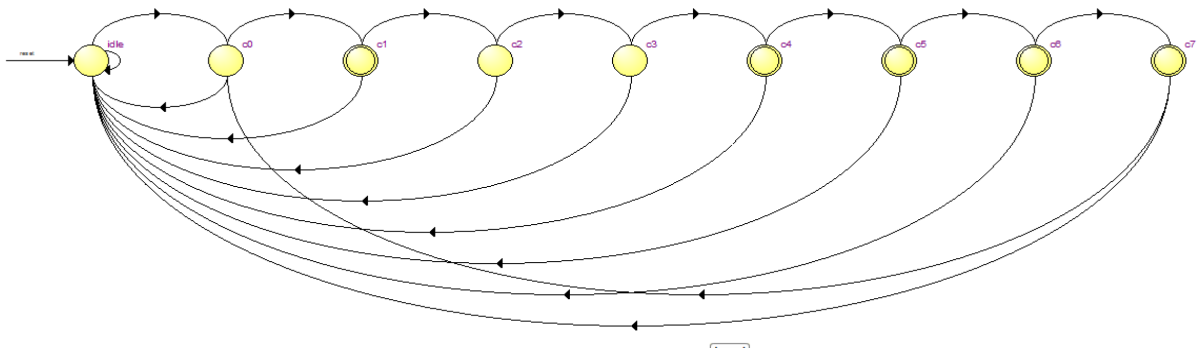
➢



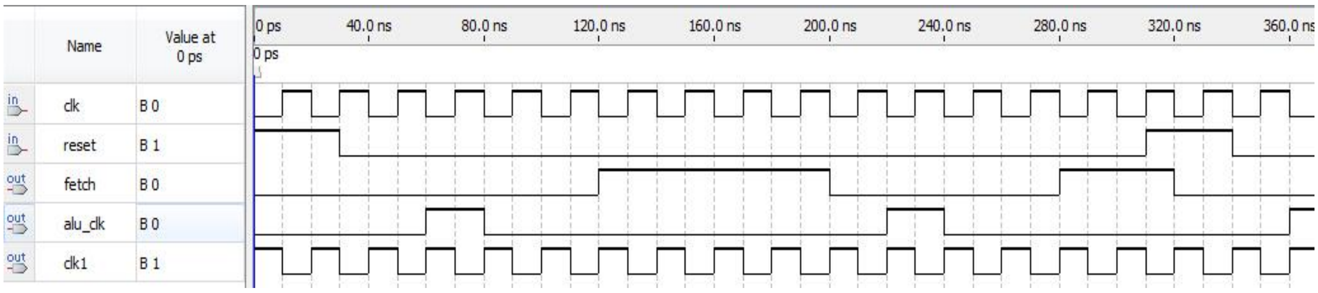
➢

```
library ieee ;
use ieee.std_logic_1164.all;
entity clkgen is
port(clk,reset :in std_logic;
        clk1,fetch,alu_clk: out std_logic);
end clkgen;
architecture behav of clkgen is
type my_type is (idle,c0,c1,c2,c3,c4,c5,c6,c7);
signal pr_state,next_state : my_type;
begin
  seq:process(clk)
  begin
  clk1<=not clk;
  if(falling_edge(clk)) then
  if (reset='1') then
  pr_state<=idle;
  else
  pr_state<=next_state;
  end if;
```

```
    end if;
    end process seq;
    ns:process(pr_state)
    begin
    case pr_state is
    when c0=>next_state<=c1;
    when c1=>next_state<=c2;
    when c2=>next_state<=c3;
    when c3=>next_state<=c4;
    when c4=>next_state<=c5;
    when c5=>next_state<=c6;
    when c6=>next_state<=c7;
    when c7=>next_state<=c0;
    when idle=>next_state<=c0;
    end case;
    end process ns;
    op:process(pr_state)
    begin
   case pr_state is
    when c0=>fetch<='0';alu_clk<='0';
    when c1=>fetch<='0';alu_clk<='1';
    when c2=>fetch<='0';alu_clk<='0';
    when c3=>fetch<='0';alu_clk<='0';
    when c4=>fetch<='1';alu_clk<='0';
    when c5=>fetch<='1';alu_clk<='0';
    when c6=>fetch<='1';alu_clk<='0';
    when c7=>fetch<='1';alu_clk<='0';
    when idle=>fetch<='0';alu_clk<='0';
    end case;
    end process op;
    end behav;
```



➢



➢

clk1 clk1

8 8 clk1

CPU load_ir load_ir ena

2 16 5 11 CPU

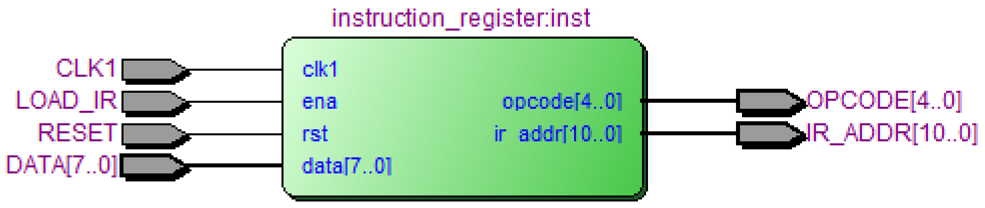
11 2K 8

8 8 8 8 state state

8 8 state 1 state 1

8 8

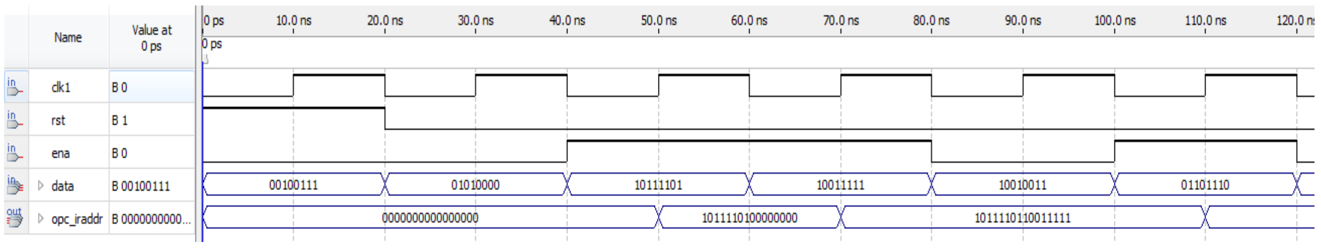
➢



➢

```
library ieee;
use ieee.std_logic_1164.all;
entity instruction_register is
port(data: in std_logic_vector(7 downto 0);
          ena : in std_logic;
           clk1 :in std_logic;
           rst : in std_logic;
           opcode : out std_logic_vector(4 downto 0);
           ir_addr : out std_logic_vector(10 downto 0));
           end instruction_register;
architecture behav of instruction_register is
signal state : std_logic;
begin
process(clk1)
begin
if(rising_edge(clk1)) then
if(rst='1')then
opcode<="00000";
ir_addr<="00000000000";
state<='0';
elsif(ena='1')then
case state is
when'0'=>
opcode<=data(7 downto 3);
ir_addr(10 downto 8)<= data(2 downto 0);
state<='1';
when'1'=>
ir_addr(7 downto 0)<=data;
state<='0';
when others=>
opcode<="XXXXX";
ir_addr<="XXXXXXXXXXX";
state<='X';
end case;
else state<='0';
end if;
end if;
end process;
end behav;
```

➢

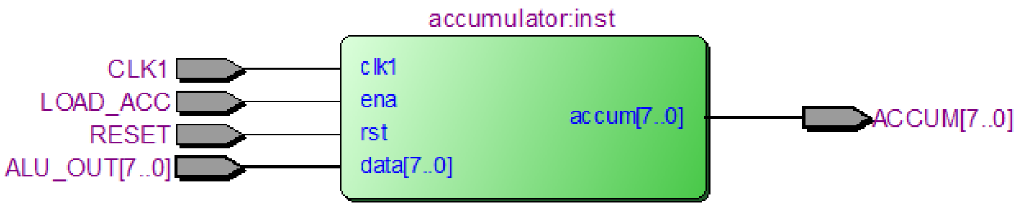


➢

8 8

ena CPU load_acc clk1

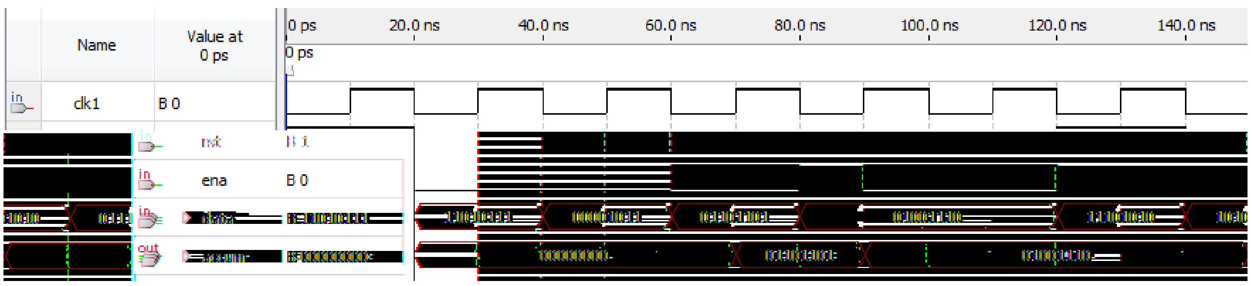
➢



➢

```
library ieee;
use ieee.std_logic_1164.all;
entity accum is
port(data : in std_logic_vector(7 downto 0);
         ena : in std_logic;
          clk1: in std_logic;
          rst : in std_logic;
          accum : out std_logic_vector(7 downto 0));
end accum;
architecture behav of accum is
begin
process(clk1)
begin
if(rising_edge(clk1)) then
if(rst='1') then
accum<="00000000";
elsif(ena='1') then
accum<=data;
end if;
end if;
end process;
end behav;
```

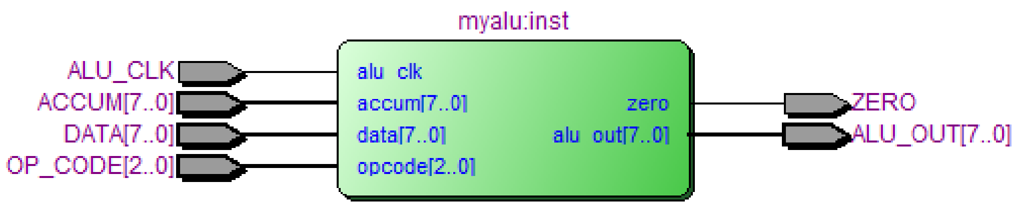
➢



➢

ena CPU load_acc clk1

➢

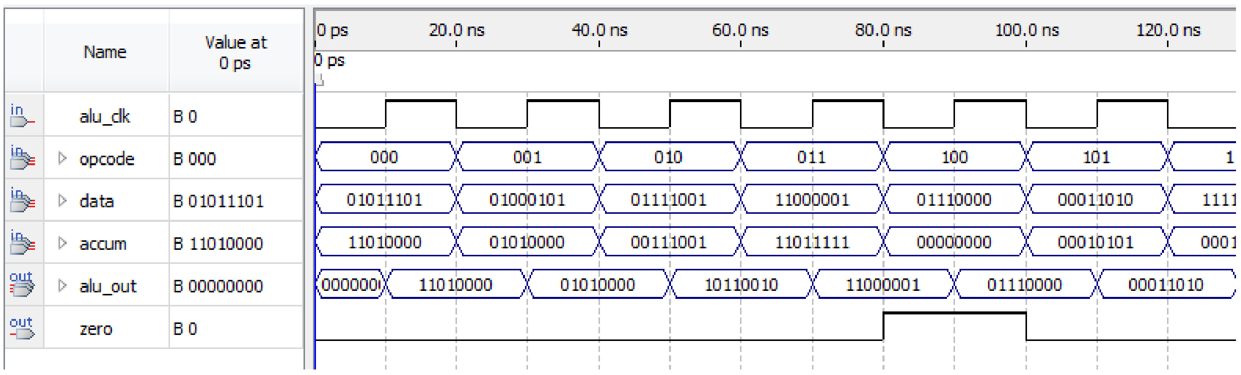


➢

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity myalu is
port(alu_clk : in std_logic;
         accum,data : in std_logic_vector(7 downto 0);
          opcode : in std_logic_vector(4 downto 0);
          data_specical : in std_logic_vector(10 downto 0);
          zero : out std_logic;
          alu_out: out std_logic_vector(7 downto 0));
end myalu;
architecture behav of myalu is
begin
zero<='1' when accum="00000000" else
          '0' ;
process(alu_clk)
begin
if (rising_edge(alu_clk)) then
case opcode is
when "00000"=>alu_out<=accum;--HLT
when "00010"=>alu_out<=accum;--SKZ
when "00100"=>alu_out<=data+accum;--ADD
when "00110"=>alu_out<=data and accum;--ANDD
when "01000"=>alu_out<=data xor accum;--XORR
when "01010"=>alu_out<=data;--LDA
when "01100"=>alu_out<=accum;--STO
when "01110"=>alu_out<=accum;--JMP
-----------------------------------------------------------
--when "00001"=>alu_out<=accum;--HLT#
--when "00011"=>alu_out<=accum;--SKZ#
when "00101"=>alu_out<=data_specical(7 downto 0)+accum;--ADD#
when "00111"=>alu_out<=data_specical(7 downto 0) and accum;--ANDD#
when "01001"=>alu_out<=data_specical(7 downto 0) xor accum;--XORR#
when "01011"=>alu_out<=data_specical(7 downto 0);--LDA#
--when "01101"=>alu_out<=accum;--STO#
--when "01111"=>alu_out<=accum;--JMP#
------------------------------------------------------------
when "10000"=>alu_out<=accum(6 downto 0) & '0';--SLL
when "10010"=>alu_out<='0' & accum(7 downto 1);--SRL
when "10100"=>alu_out<=accum(6 downto 0) & accum(7);--ROL
when "10110"=>alu_out<=accum(0) & accum(7 downto 1);--ROR
when
```

```
"11000"=>alu_out<=conv_std_logic_vector(conv_integer(accum)*conv_integer(data)
,8);--*
when
"11010"=>alu_out<=conv_std_logic_vector(conv_integer(accum)/conv_integer(data)
,8);--/
when "11100"=>alu_out<=accum-data;--SUB
when "11110"=>alu_out<=not accum;--NOT
-------------------------------------------------------------
--when "10001"=>alu_out<=accum(6 downto 0) & '0';--SLL#
--when "10011"=>alu_out<='0' & accum(7 downto 1);--SRL#
--when "10101"=>alu_out<=accum(6 downto 0) & accum(7);--ROL#
--when "10111"=>alu_out<=accum(7) & accum(6 downto 0);--ROR#
when
"11001"=>alu_out<=conv_std_logic_vector(conv_integer(accum)*conv_integer(data
_specical(7 downto 0)),8);--*#
when
"11011"=>alu_out<=conv_std_logic_vector(conv_integer(accum)/conv_integer(data_
specical(7 downto 0)),8);--/#
when "11101"=>alu_out<=accum-data_specical(7 downto 0);--SUB#
--when "11111"=>alu_out<=not accum;--NOT#
when others => alu_out<="XXXXXXXX";
end case;
end if;
end process;
end behav;
```

➢

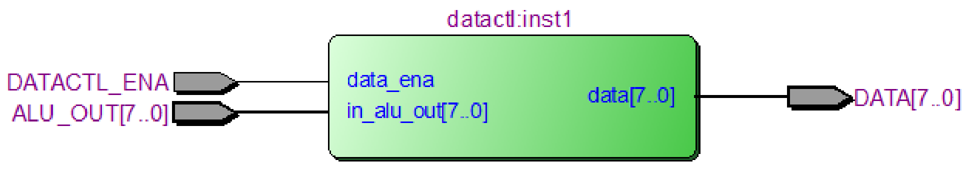


Name	Value at 0 ps
alu_clk	B 0
opcode	B 000
data	B 01011101
accum	B 11010000
alu_out	B 00000000
zero	B 0

0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
opcode: 000, 001, 010, 011, 100, 101, 1
data: 01011101, 01000101, 01111001, 11000001, 01110000, 00011010, 1111
accum: 11010000, 01010000, 00111001, 11011111, 00000000, 00010101, 0001
alu_out: 000000, 11010000, 01010000, 10110010, 11000001, 01110000, 00011010

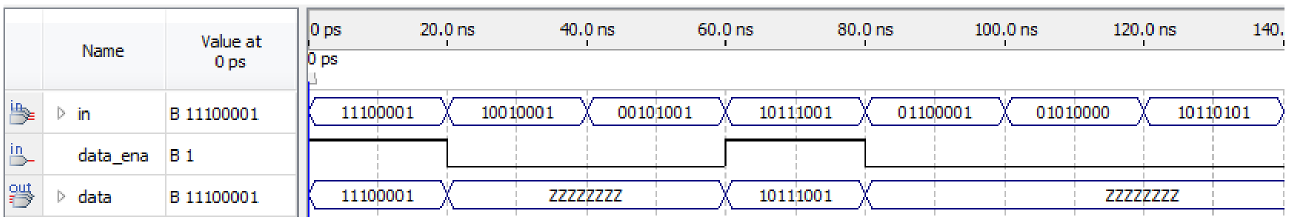
➢

➢



➢

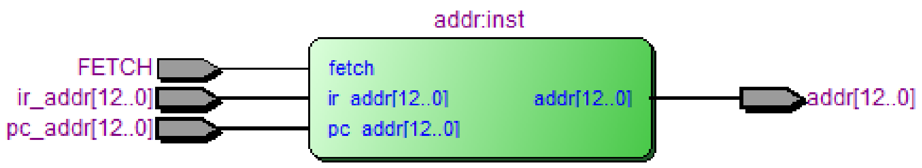
```
library ieee;
use ieee.std_logic_1164.all;
```



Name	Value at 0 ps	0 ps	20.0 ns	40.0 ns	60.0 ns	80.0 ns	100.0 ns	120.0 ns
in	B 11100001	11100001	10010001	00101001	10111001	01100001	01010000	10110101
data_ena	B 1	1	0	0	1	0	0	0
data	B 11100001	11100001	ZZZZZZZZ	ZZZZZZZZ	10111001	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ

➢

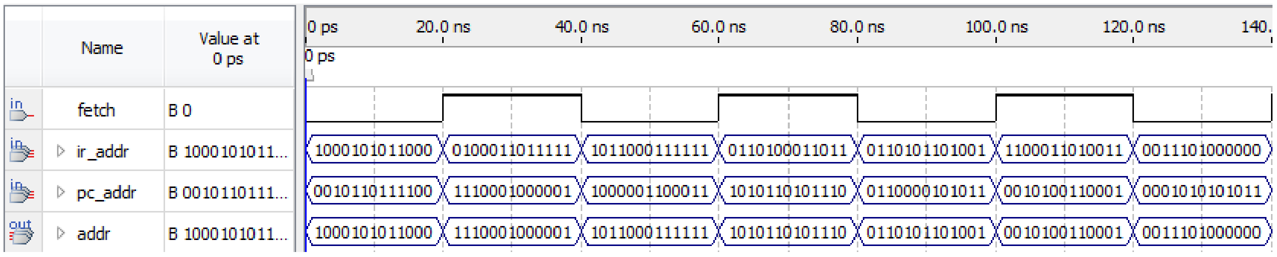
➢



➢

```
library ieee;
use ieee.std_logic_1164.all;
entity myaddr is
port (fetch : in std_logic;
        ir_addr,pc_addr : in std_logic_vector( 10 downto 0);
         addr : out std_logic_vector(10 downto 0));
end myaddr;
architecture behav of myaddr is
begin
addr<=ir_addr when fetch='0' else
          pc_addr when fetch='1' else
            "XXXXXXXXXXX";
end behav;
```

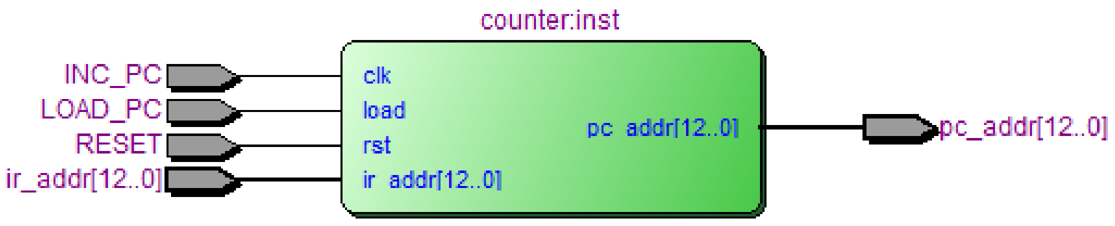
➢



Name	Value at 0 ps							
fetch	B 0							
ir_addr	B 1000101011...	1000101011000	0100011011111	1011000111111	0110100011011	0110101101001	1100011010011	0011101000000
pc_addr	B 0010110111...	0010110111100	1110001000001	1000001100011	1010110101110	0110000101011	0010100110001	0001010101011
addr	B 1000101011...	1000101011000	1110001000001	1011000111111	1010110101110	0110101101001	0010100110001	0011101000000

➢

➢

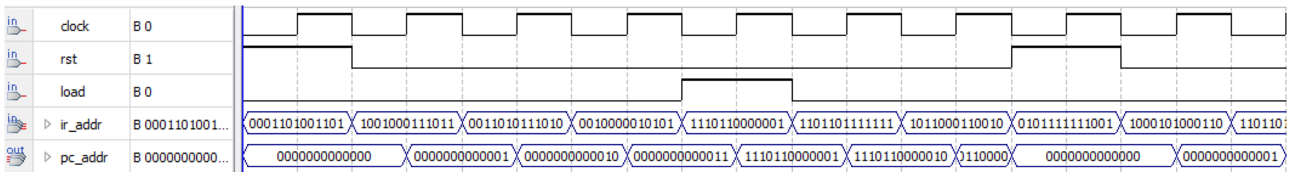


➢

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
port(load,clk,rst : in std_logic;
        ir_addr : in std_logic_vector(10 downto 0);
         pc_addr : buffer std_logic_vector(10 downto 0));
```

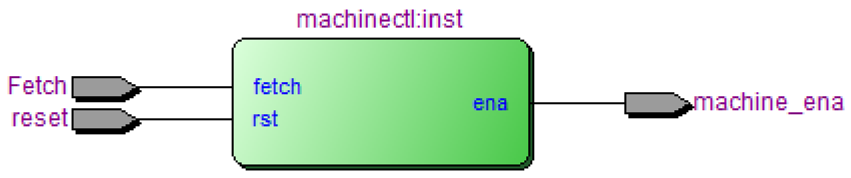
```
end counter;
architecture behav of counter is
begin
process(clk,rst)
begin
if(rst='1') then
pc_addr<="00000000000";
elsif(rising_edge(clk)) then
if(load='1') then
pc_addr<=ir_addr;
else
pc_addr<=pc_addr+1;
end if;
end if;
end process;
end behav;
```

➢



➢

➢

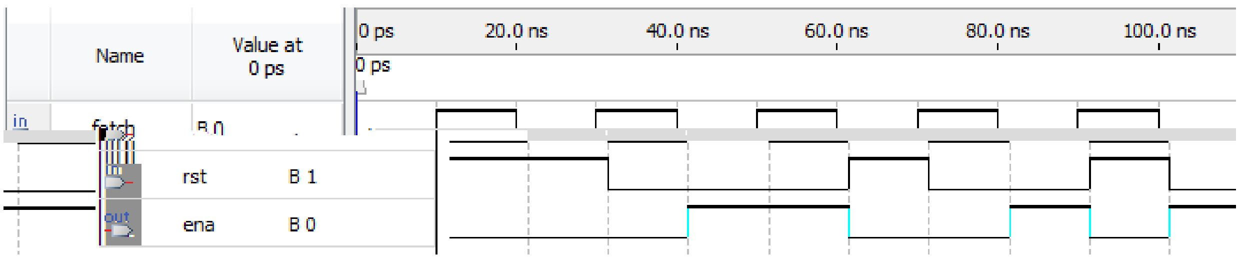


➢

```
library ieee;
use ieee.std_logic_1164.all;
entity machinectl is
port(ena : out std_logic;
         fetch : in std_logic;
          rst :in std_logic);
end machinectl;
```

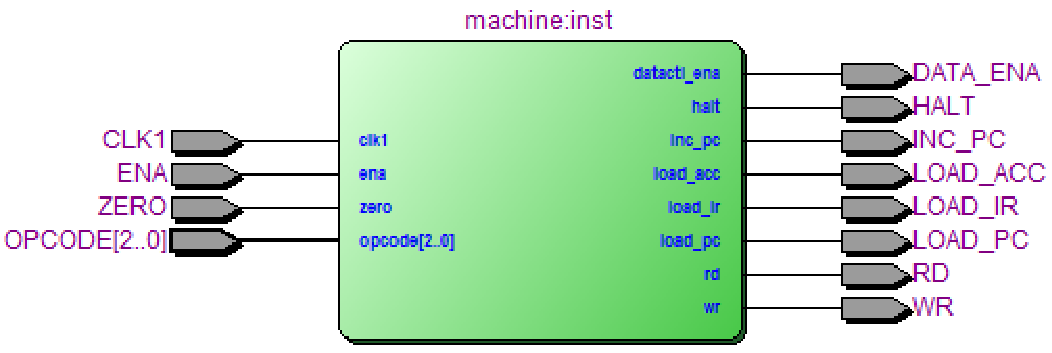
```
architecture behav of machinectl is
begin
process(fetch,rst)
begin
if(rst='1') then
ena<='0';
elsif(rising_edge(fetch)) then
ena<='1';
end if;
end process;
end behav;
```

➢



➢

➢



➢

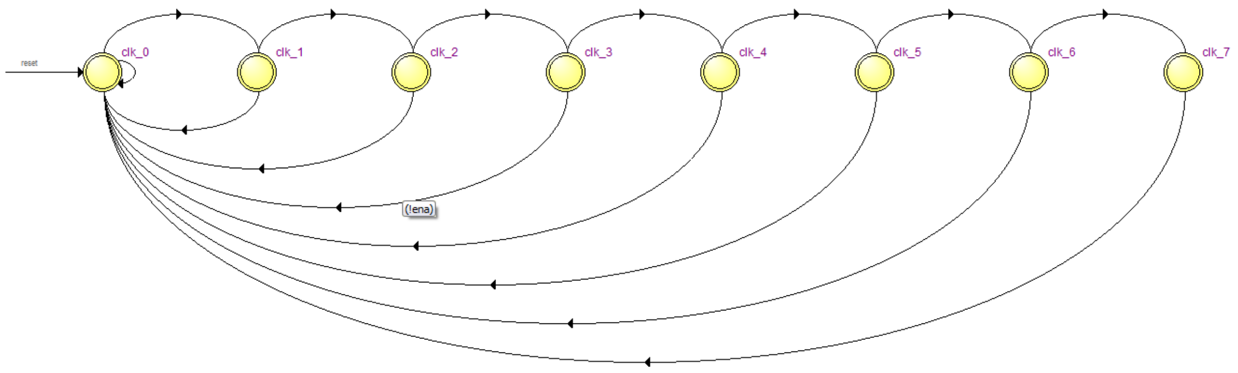
```
LIBRARY IEEE;
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
entity machine is
      port(clk1: in std_logic;                        --the clock of cpu
            zero: in std_logic;                             --data of acc is zero
            ena: in std_logic;                        --enable port
            opcode: in std_logic_vector(4 downto 0);--operation code
            inc_pc:out std_logic;                     --increase pc point
            load_acc: out std_logic;                  --acc output enable
            load_pc: out std_logic;                         --pc point load
            rd:out std_logic;                         --read from ROM
            wr:out std_logic;                         --write to RAM
            load_ir:out std_logic;                    --load target address
            datactl_ena:out std_logic;                      --data out enable
            halt:out std_logic);                      --halt code
end machine;
architecture behav of machine is
begin

main:process(clk1,zero,ena,opcode)
      type state_type is (clk_0,clk_1,clk_2,clk_3,clk_4,clk_5,clk_6,clk_7);
      --define eight state represent for eight clocks
      variable state:state_type;
      --define eight codes using constant standard logic
      constant HLT:std_logic_vector:= "00000";
      constant SKZ:std_logic_vector:= "00010";
      constant ADD:std_logic_vector:= "00100";
      constant ANDD:std_logic_vector:="00110";
      constant XORR:std_logic_vector:="01000";
      constant LDA:std_logic_vector:= "01010";
      constant STO:std_logic_vector:= "01100";
      constant JMP:std_logic_vector:= "01110";
      ------------------------------------------------
      --constant HLT_1:std_logic_vector:= "00001";
      --constant SKZ_1:std_logic_vector:= "00011";
      constant ADD_1:std_logic_vector:= "00101";
      constant ANDD_1:std_logic_vector:="00111";
      constant XORR_1:std_logic_vector:="01001";
      constant LDA_1:std_logic_vector:= "01011";
      --constant STO_1:std_logic_vector:= "01101";
      --constant JMP_1:std_logic_vector:= "01111";
      ------------------------------------------------
```

```
constant SLLL:std_logic_vector:= "10000";
constant SRLL:std_logic_vector:= "10010";
constant ROLL:std_logic_vector:= "10100";
constant RORR:std_logic_vector:="10110";
constant MUL:std_logic_vector:="11000";
constant DIV:std_logic_vector:= "11010";
constant SUB:std_logic_vector:= "11100";
constant NOTT:std_logic_vector:= "11110";
-----------------------------------------------------
--constant SLLL_1:std_logic_vector:= "10001";
--constant SRLL_1:std_logic_vector:= "10011";
--constant ROLL_1:std_logic_vector:= "10101";
--constant RORR_1:std_logic_vector:="10111";
constant MUL_1:std_logic_vector:="11001";
constant DIV_1:std_logic_vector:= "11011";
constant SUB_1:std_logic_vector:= "11101";
--constant NOTT_1:std_logic_vector:= "11111";
-----------------------------------------------------------
begin
if(clk1'event and clk1='0')then                 --the negative edge of clock
      if(ena='0')then                                 --state loop enable
            state:=clk_0;
            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';--initialize;
      else
            case state is
            --load the high 8bits instruction
            when clk_0 =>                             --the zreoth clock
                  inc_pc<='0';load_acc<='0';load_pc<='0';rd<='1';--read from ROM
                  wr<='0';load_ir<='1';datactl_ena<='0';halt<='0';
                        state:=clk_1;
            --pc increase then load low 8bits instruction
            when clk_1 =>                             --the first clock
                        inc_pc<='1';load_acc<='0';load_pc<='0';rd<='1';
                        --pc increase ,read from ROM
                        wr<='0';load_ir<='1';datactl_ena<='0';halt<='0';
                        --load the target pc point
                        state:=clk_2;
            --idle
            when clk_2 =>                             --the second clock
                  inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                  wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        state:=clk_3;
            --the instruction of the halt code
            when clk_3 =>                       --the third clock
                        if(opcode=HLT)then
                            inc_pc<='1';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='1';
                        else
                            inc_pc<='1';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        end if;
                        state:=clk_4;
            --other code instruction
            when clk_4 =>                       --the forth clock
                        if(opcode=JMP)then
                            inc_pc<='0';load_acc<='0';load_pc<='1';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        elsif(opcode=ADD or opcode=ANDD or opcode=XORR or
opcode=LDA
                           or opcode=ADD_1 or opcode=ANDD_1 or
opcode=XORR_1 or opcode=LDA_1
                            or opcode=MUL or opcode=DIV or opcode=SUB or
opcode=NOTT
                            or opcode=MUL_1 or opcode=DIV_1 or opcode=SUB_1
                            or opcode=SLLL or opcode=SRLL or opcode=ROLL or
opcode=RORR    )then
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='1';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        elsif(opcode=STO)then
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='1';halt<='0';
                        else
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        end if;
                        state:=clk_5;
            when clk_5 =>                       --the fifth clock
                        if(opcode=ADD or opcode=ANDD or opcode=XORR or
opcode=LDA
                           or opcode=ADD_1 or opcode=ANDD_1 or
opcode=XORR_1 or opcode=LDA_1
                            or opcode=MUL or opcode=DIV or opcode=SUB or
opcode=NOTT
                            or opcode=MUL_1 or opcode=DIV_1 or opcode=SUB_1
                            or opcode=SLLL or opcode=SRLL or opcode=ROLL or
opcode=RORR   )then
                            inc_pc<='0';load_acc<='1';load_pc<='0';rd<='1';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        elsif(opcode=SKZ and zero='1')then
                            inc_pc<='1';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        elsif(opcode=JMP)then
                            inc_pc<='1';load_acc<='0';load_pc<='1';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        elsif(opcode=STO)then
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='1';load_ir<='0';datactl_ena<='1';halt<='0';
                        else
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        end if;
                        state:=clk_6;
            when clk_6 =>                           --the sixth clock
                        if(opcode=STO)then
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='1';halt<='0';
                            --output the data
                        elsif(opcode=ADD or opcode=ANDD or opcode=XORR or
opcode=LDA
                           or opcode=ADD_1 or opcode=ANDD_1 or
opcode=XORR_1 or opcode=LDA_1
                            or opcode=MUL or opcode=DIV or opcode=SUB or
opcode=NOTT
                            or opcode=MUL_1 or opcode=DIV_1 or opcode=SUB_1
                            or opcode=SLLL or opcode=SRLL or opcode=ROLL or
opcode=RORR
                        )then
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='1';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        else
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        end if;
                        state:=clk_7;
            when clk_7 =>                           --the seventh clock
                        if(opcode=SKZ and zero='1')then
                            inc_pc<='1';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        else
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        end if;
                        state:=clk_0;
            when others =>                      --for other state(s)
                            inc_pc<='0';load_acc<='0';load_pc<='0';rd<='0';
                            wr<='0';load_ir<='0';datactl_ena<='0';halt<='0';
                        state:=clk_0;
            end case;
        end if;
    else null;
    end if;
end process main;

end behav;
```

➢



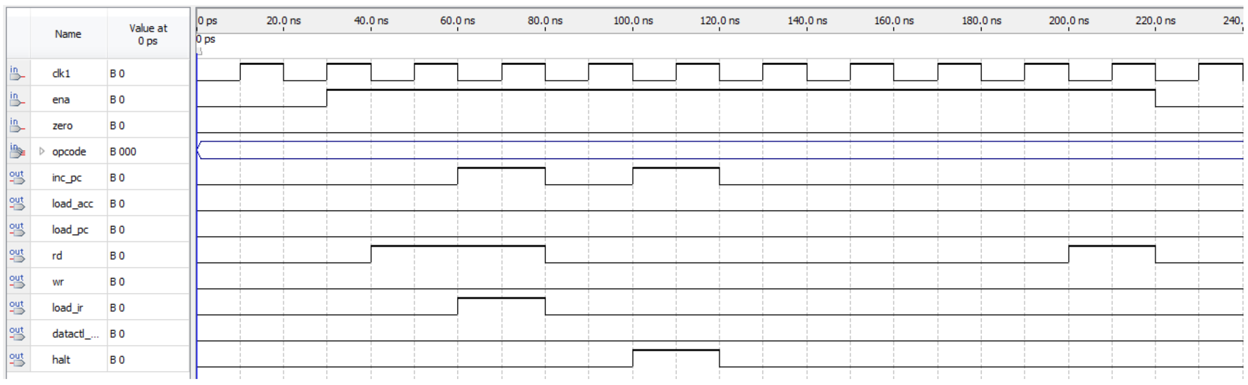
➢

1 HLT cpu ena=1 rd=1

inc_pc=1 load_ir=1

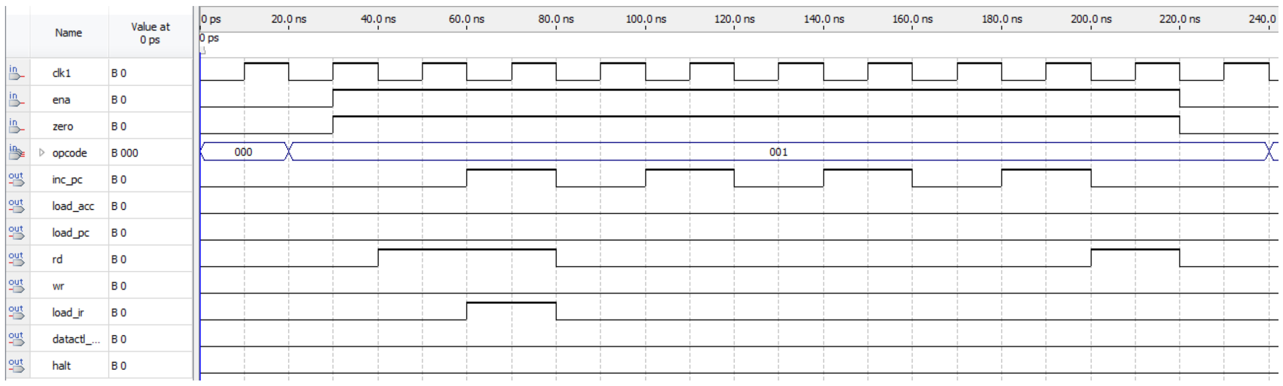
inc_pc=0 load_ir=0 rd=0 inc_pc=1

halt=1

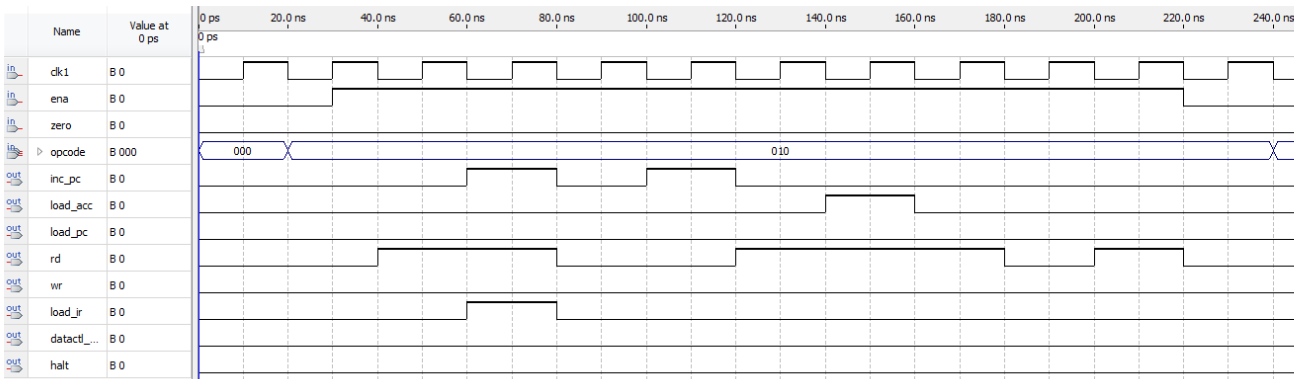


Name
Value at 0 ps
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
140.0 ns
160.0 ns
180.0 ns
200.0 ns
220.0 ns
240.
clk1 B 0
ena B 0
zero B 0
opcode B 000
inc_pc B 0
load_acc B 0
load_pc B 0
rd B 0
wr B 0
load_ir B 0
datactl_... B 0
halt B 0

2



Name
Value at 0 ps
0 ps
20.0 ns
40.0 ns
60.0 ns
80.0 ns
100.0 ns
120.0 ns
140.0 ns
160.0 ns
180.0 ns
200.0 ns
220.0 ns
240.0
clk1 B 0
ena B 0
zero B 0
opcode B 000
000
001
inc_pc B 0
load_acc B 0
load_pc B 0
rd B 0
wr B 0
load_ir B 0
datactl_... B 0
halt B 0



4 STO ena=1 rd=1

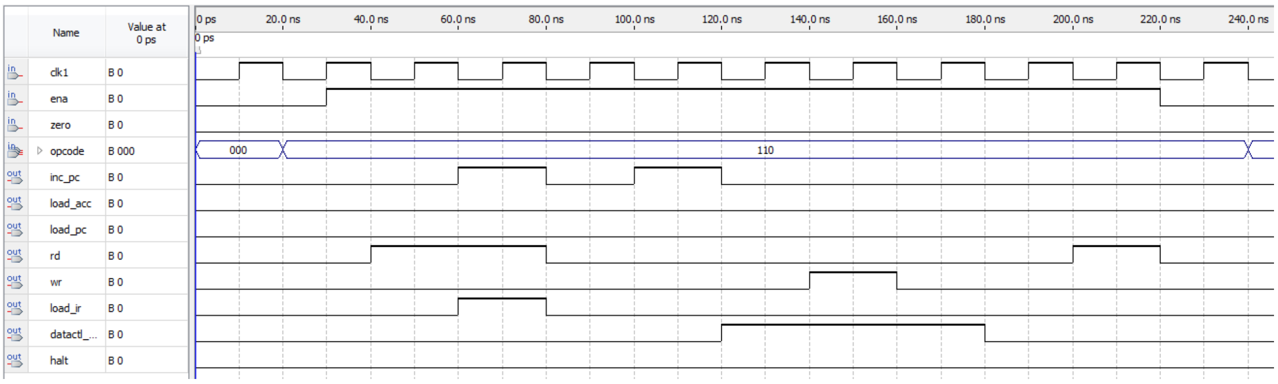
inc_pc=1 load_ir=1

inc_pc=0 load_ir=0 rd=0 inc_pc=1

datactl_ena=1

datactl_ena=1 wr=1 RAM

datactl_ena=1



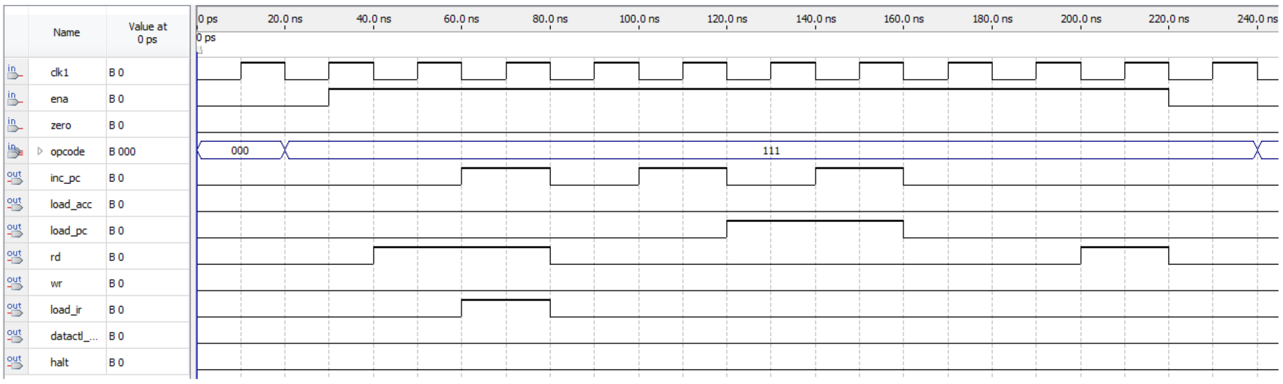
5 JMP ena=1 rd=1

inc_pc=1 load_ir=1

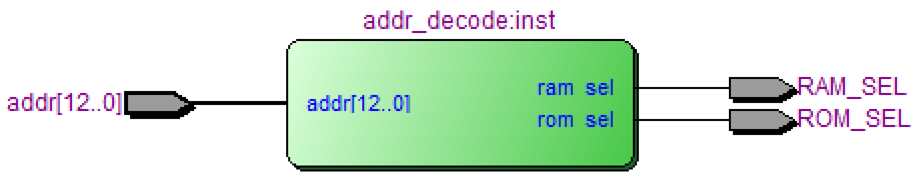
inc_pc=0 load_ir=0 rd=0 inc_pc=1

halt=1 inc_pc=0 halt=0 load_pc=1,

inc_pc=1 load_pc=1,



➢

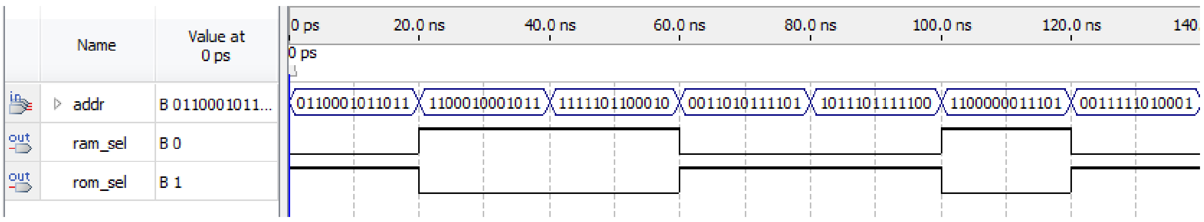


➢

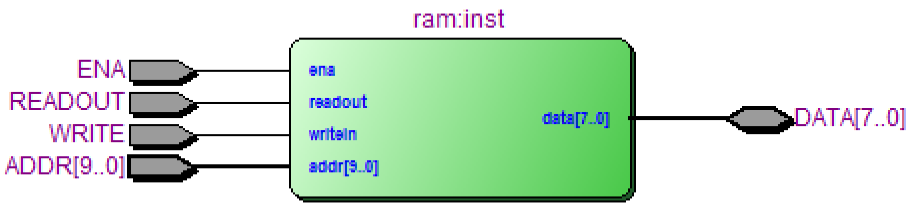
```
library ieee;
library ieee;
use ieee.std_logic_1164.all;
entity addr_decodeer is
port(addrin : in std_logic_vector(10 downto 0);
        addrout : out std_logic_vector(9 downto 0);
       rom_sel,ram_sel : out std_logic);
end addr_decodeer;
architecture behav of addr_decodeer is
begin
addrout<=addrin(9 downto 0);
process(addrin)
begin
case addrin(10) is
```

```
when '0'=>
rom_sel<='1';ram_sel<='0';
when '1'=>
rom_sel<='0';ram_sel<='1';
when others =>
rom_sel<='0';ram_sel<='0';
end case;
end process;
end behav;
```

➢



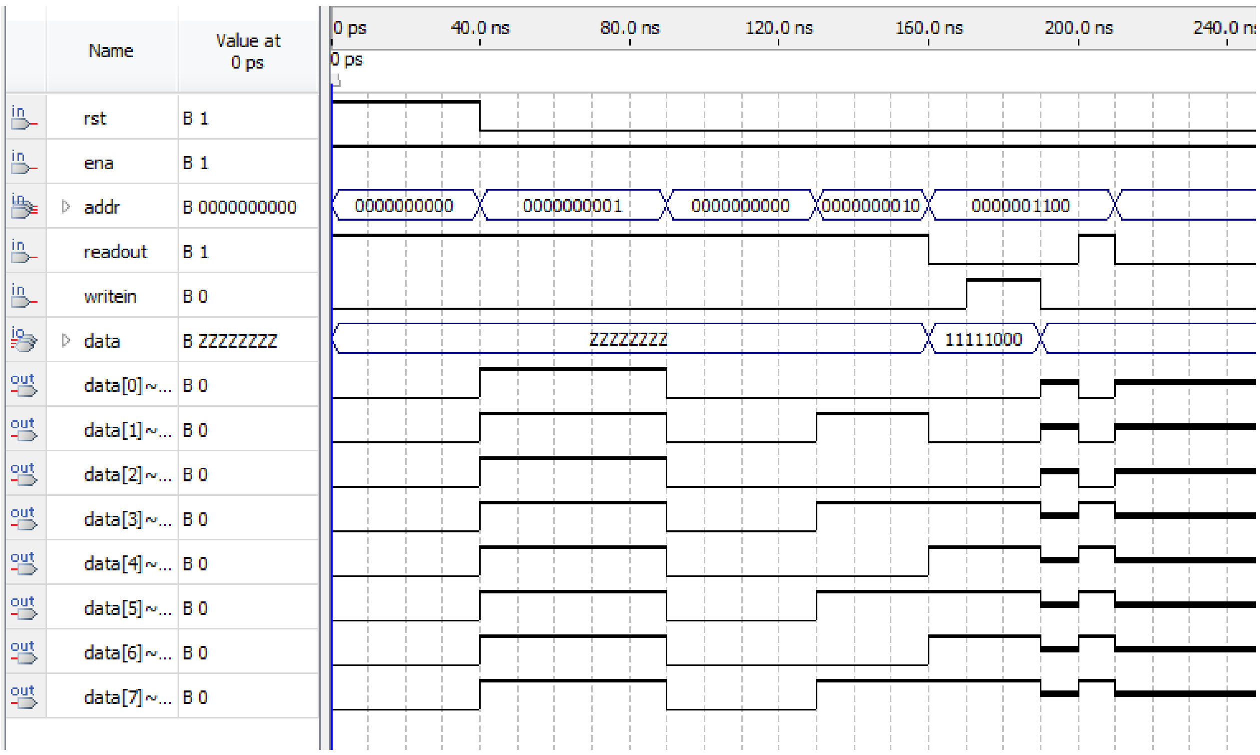
➢

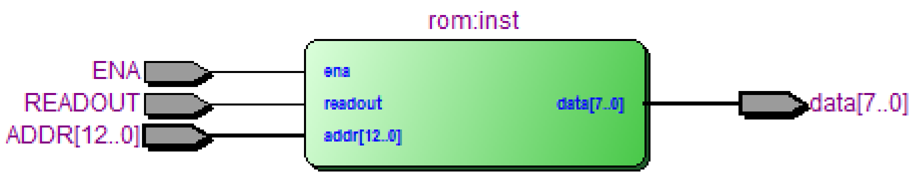


➢

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram is
port (rst : in std_logic;
            ena : in std_logic;
            writein,readout : in std_logic;
              data : inout std_logic_vector(7 downto 0);
              addr : in std_logic_vector(9 downto 0));
end ram;
architecture behav of ram is
type ram_array is array(0 to 255) of std_logic_vector (7 downto 0);
signal    mem:ram_array;
begin
process(readout,data,addr,ena)
begin
if(readout='1' and ena='1') then
data<=mem(conv_integer(addr));
else
data<="ZZZZZZZZ";
end if;
end process;
process(writein,rst)
begin
if(rst='1') then
mem(0)<="10111011";--800H DATA_1;
mem(1)<="11111111";--801H DATA_2;
mem(2)<="10101010";-----
elsif(rising_edge(writein)) then
--elsif(writein='1') then
mem(conv_integer(addr))<=data;
end if;
end process;
end behav;
```

➢





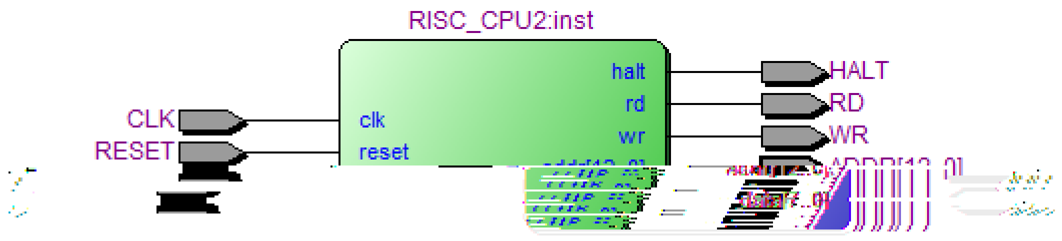
➢

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rom is
port (ena : in std_logic;
            readout : in std_logic;
              data : out std_logic_vector(7 downto 0);
              addr : in std_logic_vector(12 downto 0));
end rom;
architecture behav of rom is
```

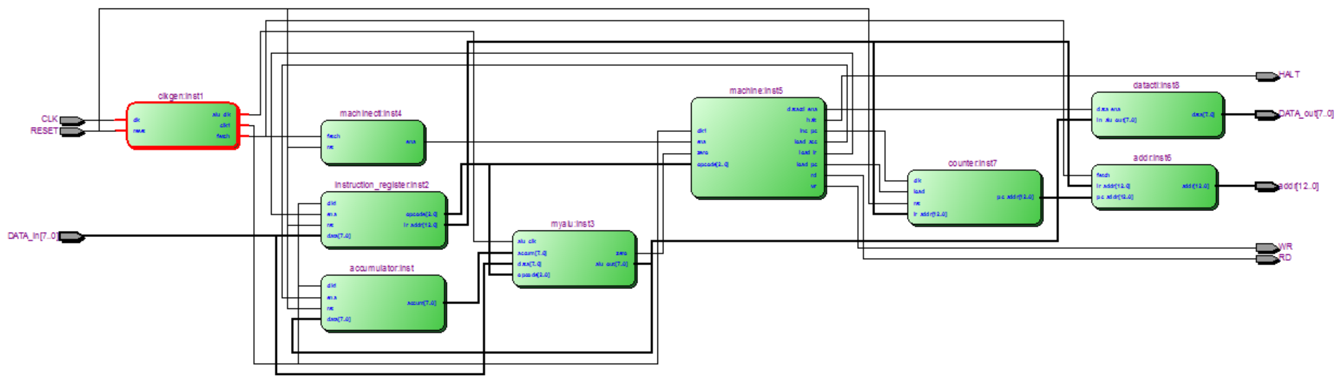
```
type rom_array is array(0 to 1023) of std_logic_vector (7 downto 0);
signal    mem:rom_array;
begin
process(addr,ena,readout)
begin
if(readout='1'and ena='1') then
data<=mem(conv_integer(addr));
else
data<="ZZZZZZZZ";
end if;
end process;
end behav;
```

RISC_CPU

➢



➢



➢

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity RISC_CPU is
port(clk,reset :in std_logic;
         data : inout std_logic_vector(7 downto 0);
          rd,wr,halt : out std_logic;
          addr: buffer std_logic_vector(10 downto 0));
end RISC_CPU;
architecture instru of RISC_CPU is
component clkgen is
port(clk,reset :in std_logic;
         clk1,fetch,alu_clk: out std_logic);
end component;
component instruction_register is
port(data: in std_logic_vector(7 downto 0);
         ena : in std_logic;
          clk1 :in std_logic;
          rst : in std_logic;
          opcode : out std_logic_vector(4 downto 0);
          ir_addr : out std_logic_vector(10 downto 0));
end component;
component accumulator is
port(data : in std_logic_vector(7 downto 0);
         ena : in std_logic;
          clk1: in std_logic;
          rst : in std_logic;
          accum : out std_logic_vector(7 downto 0));
end component;
component myalu is
```

```
port(alu_clk : in std_logic;
          accum,data : in std_logic_vector(7 downto 0);
           opcode : in std_logic_vector(4 downto 0);
           data_specical : in std_logic_vector(10 downto 0);
           zero : out std_logic;
           alu_out: out std_logic_vector(7 downto 0));
end component;
component machinectl is
port(ena : out std_logic;
          fetch : in std_logic;
           rst :in std_logic);
end component;
component machine is
port(clk1: in std_logic;                          --the clock of cpu
              zero: in std_logic;                             --data of acc is zero
              ena: in std_logic;                        --enable port
              opcode: in std_logic_vector(4 downto 0);--operation code
              inc_pc:out std_logic;                     --increase pc point
              load_acc: out std_logic;                  --acc output enable
              load_pc: out std_logic;                          --pc point load
              rd:out std_logic;                         --read from ROM
              wr:out std_logic;                         --write to RAM
              load_ir:out std_logic;                    --load target address
              datactl_ena:out std_logic;                       --data out enable
              halt:out std_logic);                      --halt code
end component;
component myaddr is
port (fetch : in std_logic;
          ir_addr,pc_addr : in std_logic_vector( 10 downto 0);
           addr : out std_logic_vector(10 downto 0));
end component;
component counter is
port(load,clk,rst : in std_logic;
          ir_addr : in std_logic_vector(10 downto 0);
           pc_addr : buffer std_logic_vector(10 downto 0));
end component;
component datactl is
port(in_alu_out : in std_logic_vector(7 downto 0);
          data_ena : in std_logic;
           data : out std_logic_vector(7 downto 0));
end component;
-------------------------------------------------------------
signal carry_clk1,carry_fetch,carry_alu_clock : std_logic;
signal carry_zero,carry_machine_ena : std_logic;
signal carry_inc_pc,carry_load_acc,carry_load_pc : std_logic;
signal carry_load_ir,carry_datactl_ena : std_logic;
signal carry_opcode : std_logic_vector(4 downto 0);
signal carry_ir_addr : std_logic_vector(10 downto 0);
signal carry_alu_out : std_logic_vector(7 downto 0);
signal carry_accum : std_logic_vector( 7 downto 0);
signal carry_pc_addr : std_logic_vector( 10 downto 0);
signal carry_specicaldata : std_logic_vector( 10 downto 0);
begin
U0:clkgen   port map(clk,reset,carry_clk1,carry_fetch,carry_alu_clock);
U1:instruction_register                                                    port
    map(data,carry_load_ir,carry_clk1,reset,carry_opcode,carry_ir_addr);
U2:accumulator                        port                                 map
    (carry_alu_out,carry_load_acc,carry_clk1,reset,carry_accum);
U3:myalu                                                                   port
    map(carry_alu_clock,carry_accum,data,carry_opcode,addr,carry_zero,carry_alu_
    out);
U4:machinectl port map(carry_machine_ena,carry_fetch,reset);
U5:machine                                                                 port
    map(carry_clk1,carry_zero,carry_machine_ena,carry_opcode,carry_inc_pc,

    carry_load_acc,carry_load_pc,rd,wr,carry_load_ir,carry_datactl_ena,halt);
U6:myaddr port map(carry_fetch,carry_ir_addr,carry_pc_addr,addr);
U7:counter                                                                 port
    map(carry_load_pc,carry_inc_pc,reset,carry_ir_addr,carry_pc_addr);
U8:datactl port map(carry_alu_out,carry_datactl_ena,data);
end instru;
```

➢

CPU RISC_CPU

1

2

3

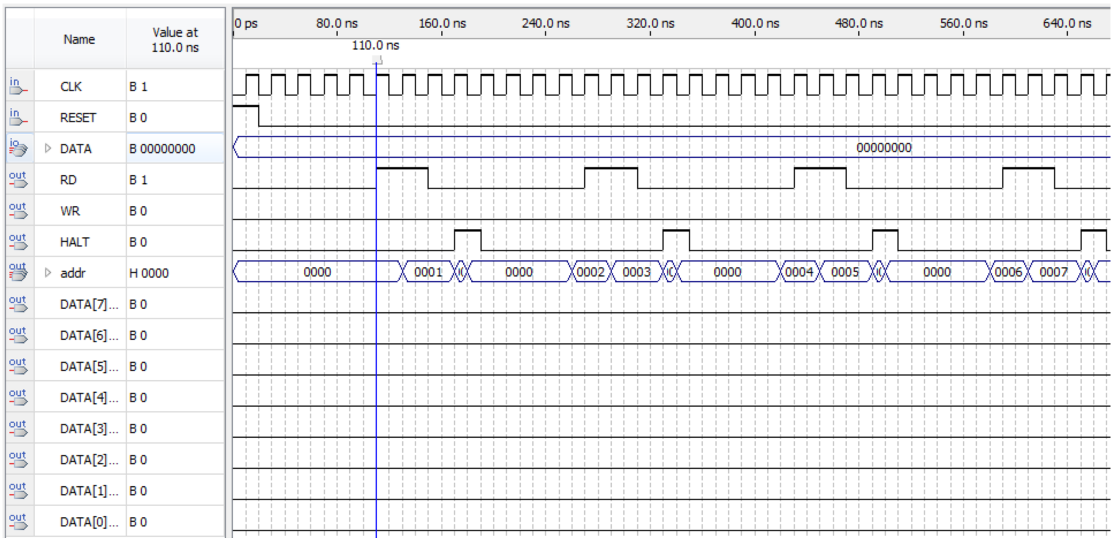
RISC_CPU rst rst

RISC_CPU rst CPU

CPU

000H Rst fetch

RISC_CPU ROM 000H



Name	Value at 110.0 ns
CLK	B 1
RESET	B 0
DATA	B 00000000
RD	B 1
WR	B 0
HALT	B 0
addr	H 0000
DATA[7]...	B 0
DATA[6]...	B 0
DATA[5]...	B 0
DATA[4]...	B 0
DATA[3]...	B 0
DATA[2]...	B 0
DATA[1]...	B 0
DATA[0]...	B 0

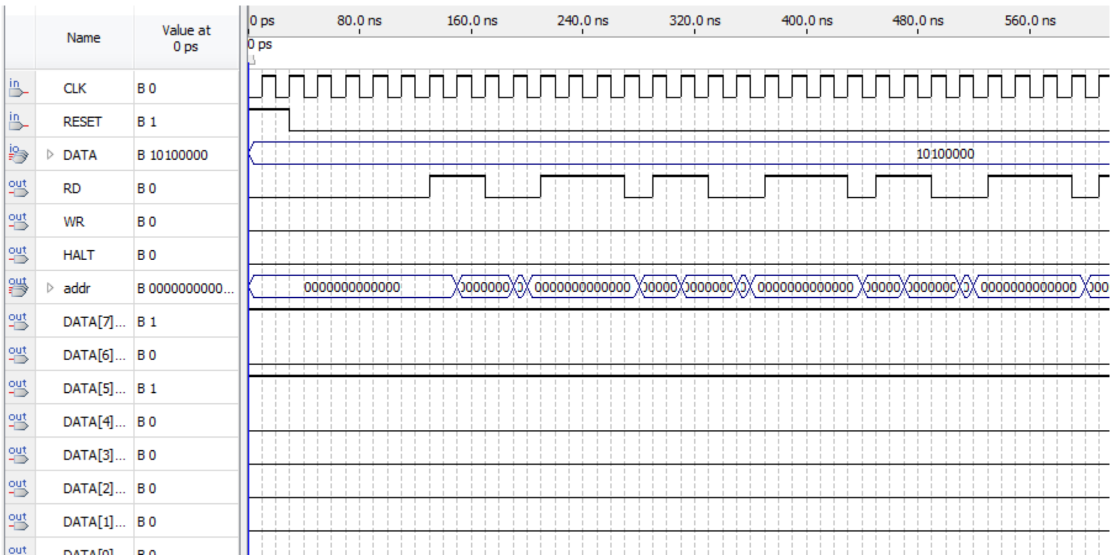
1

0-3 3.5

4-6 rd

7 7.5

PC

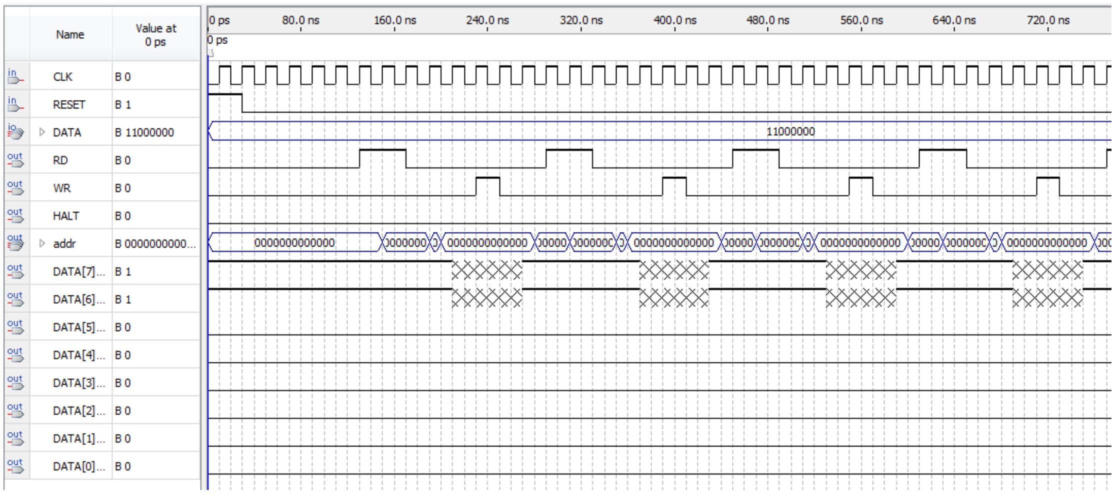


Name	Value at 0 ps
CLK	B 0
RESET	B 1
DATA	B 10100000
RD	B 0
WR	B 0
HALT	B 0
addr	B 0000000000...
DATA[7]...	B 1
DATA[6]...	B 0
DATA[5]...	B 1
DATA[4]...	B 0
DATA[3]...	B 0
DATA[2]...	B 0
DATA[1]...	B 0
DATA[0]...	B 0

2

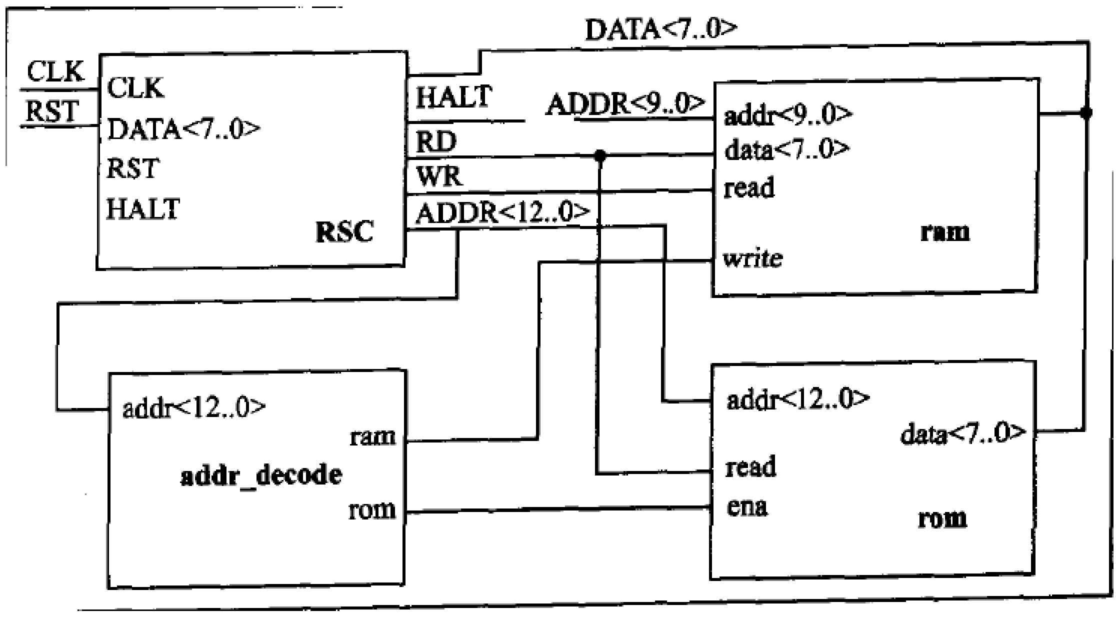
3.5 4

5 6 7.5 PC

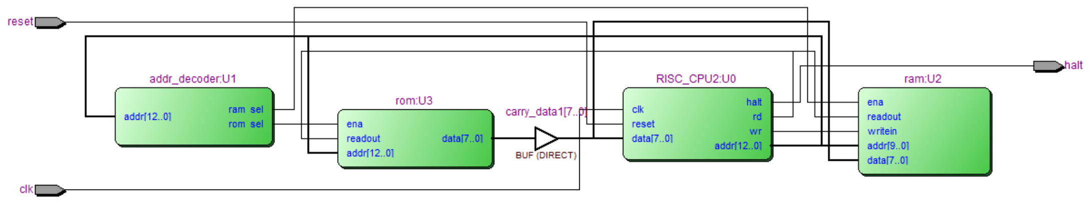

Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
CLK B 0
RESET B 1
DATA B 11000000
RD B 0
WR B 0
HALT B 0
addr B 0000000000...
DATA[7]... B 1
DATA[6]... B 1
DATA[5]... B 0
DATA[4]... B 0
DATA[3]... B 0
DATA[2]... B 0
DATA[1]... B 0
DATA[0]... B 0
11000000
0000000000000

3

➢


DATA<7..0>
CLK
RST
CLK
DATA<7..0>
RST
HALT
RSC
HALT
RD
WR
ADDR<9..0>
ADDR<12..0>
addr<9..0>
data<7..0>
read
write
ram
addr<12..0>
ram
addr_decode
rom
addr<12..0>
data<7..0>
read
ena
rom

➢ RTL


reset
clk
halt
addr_decoder:U1
addr[12..0]
ram_sel
rom_sel
rom:U3
ena
readout
addr[12..0]
data[7..0]
carry_data1[7..0]
BUF (DIRECT)
RISC_CPU2:U0
clk
reset
data[7..0]
halt
rd
wr
addr[12..0]
ram:U2
ena
readout
writein
addr[9..0]
data[7..0]

➢

```
library ieee;
use ieee.std_logic_1164.all;
entity RISC_CPU_TPO1 is
port( clk : in std_logic;
          reset : in std_logic;
halt : out std_logic);
end RISC_CPU_TPO1;
architecture instu of RISC_CPU_TPO1 is
component RISC_CPU is
port(clk,reset :in std_logic;
         data : inout std_logic_vector(7 downto 0);
          rd,wr,halt : out std_logic;
          addr: buffer std_logic_vector(10 downto 0));
end component;
component addr_decodeer is
port(addrin : in std_logic_vector(10 downto 0);
          addrout : out std_logic_vector(9 downto 0);
         rom_sel,ram_sel : out std_logic);
end component;
component ram1 is
port (rst : in std_logic;
          ena : in std_logic;
          writein,readout : in std_logic;
            data : inout std_logic_vector(7 downto 0);
            addr : in std_logic_vector(9 downto 0));
end component;
component rom1 is
port (ena : in std_logic;
          readout : in std_logic;
            data : out std_logic_vector(7 downto 0);
            addr : in std_logic_vector(9 downto 0));
end component;
signal carry_rom_sel,carry_ram_sel : std_logic;
signal carry_write,carry_read : std_logic;
signal carry_data1 : std_logic_vector(7 downto 0);
signal carry_addr : std_logic_vector (10 downto 0);
signal carry_addrout : std_logic_vector (9 downto 0);
begin
U0:RISC_CPU port map (clk,reset,carry_data1,carry_read,carry_write,halt,carry_addr);
U1:addr_decodeer port map (carry_addr,carry_addrout,carry_rom_sel,carry_ram_sel);
U2:ram1 port map (reset,carry_ram_sel,carry_write,carry_read,carry_data1,carry_addrout);
U3:rom1 port map (carry_rom_sel,carry_read,carry_data1,carry_addrout);
end instu;
```

ROM RISC_CPU

Program1 RISC_CPU

RISC_CPU RISC_CPU

2E HLT

Program1 ROM1 RAM1

- **ROM**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rom1 is
port (ena : in std_logic;
        readout : in std_logic;
          data : out std_logic_vector(7 downto 0);
          addr : in std_logic_vector(12 downto 0));
end rom1;
architecture behav of rom1 is
type rom_array is array(0 to 1023) of std_logic_vector (7 downto 0);
signal   mem:rom_array;
begin
mem(0)<="11100000"; --00 begin:JMP TST_JMP
mem(1)<="00111100";
mem(2)<="00000000"; --02 HLT//JMP did not work
mem(3)<="00000000";
mem(4)<="00000000"; --04 HLT//JMP did not load PC, it skipped
mem(5)<="00000000";
mem(6)<="10111000"; --06 JMP_OK:LDA DATA_1
mem(7)<="00000000";
mem(8)<="00100000"; --08 SKZ
mem(9)<="00000000";
mem(10)<="00000000"; --0a HLT //SKZ or LDA did not work
mem(11)<="00000000";
mem(12)<="10111000"; --0c LDA DATA_2
mem(13)<="00000001";
mem(14)<="00100000"; --0e SKZ
mem(15)<="00000000";
mem(16)<="11100000"; --10 JMP    SKZ_OK
mem(17)<="00010100";
mem(18)<="00000000"; -- 12 HLT        //SKZ or LDA did not work
mem(19)<="00000000";
mem(20)<="11011000";    --14 SKZ_OK:    STO TEMP //store non-zero value in TEMP
mem(21)<="00000010";
mem(22)<="10111000";    --16 LDA    DATA_1
mem(23)<="00000000";
mem(24)<="11011000";    --18 STO    TEMP    //store zero value in TEMP
mem(25)<="00000010";
mem(26)<="10111000";    --1a LDA TEMP
mem(27)<="00000010";
mem(28)<="00100000";    --1c SKZ         //check to see if STO worked
mem(29)<="00000000";
mem(30)<="00000000";    --1e HLT        //STO did not work
mem(31)<="00000000";
mem(32)<="10011000";    --20 XOR DATA_2
mem(33)<="00000001";
mem(34)<="00100000";    --22 SKZ        //check to see if XOR worked
mem(35)<="00000000";
mem(36)<="11100000";    --24 JMP XOR_OK
mem(37)<="00101000";
mem(38)<="00000000";    --26 HLT      //XOR did not work at all
mem(39)<="00000000";
mem(40)<="10011000";    --28 XOR_OK:    XOR DATA_2
mem(41)<="00000001";
mem(42)<="00100000";    --2a SKZ
mem(43)<="00000000";
mem(44)<="00000000";    --2c HLT      //XOR did not switch all bits
mem(45)<="00000000";
mem(46)<="00000000";    --2e    END: HLT      //CONGRATULATIONS - TEST1 PASSED!
mem(47)<="00000000";
mem(48)<="11100000";    --30    JMP BEGIN    //run test again
mem(49)<="00000000";
--------------------@3c--------------------
mem(60)<="11100000";    --3c TST_JMP:    JMP JMP_OK
mem(61)<="00000110";
mem(62)<="00000000";    --3e HLT        //JMP is broken
process(addr,ena,readout)
begin
if(readout='1') then
if(ena='1') then
data<=mem(conv_integer(addr));
else
```

```
data<="ZZZZZZZZ";
end if;
end if;
end process;
end behav;
```

➢ **RAM**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram1 is
port (rst : in std_logic;
        ena : in std_logic;
        writein,readout : in std_logic;
          data : inout std_logic_vector(7 downto 0);
          addr : in std_logic_vector(9 downto 0));
end ram1;
architecture behav of ram1 is
type ram_array is array(0 to 255) of std_logic_vector (7 downto 0);
signal   mem:ram_array;
begin
process(readout,data,addr,ena)
begin
if(readout='1' and ena='1') then
data<=mem(conv_integer(addr));
else
data<="ZZZZZZZZ";
end if;
end process;
process(writein,rst)
begin
if(rst='1') then
mem(0)<="00000000";--1800H DATA_1;
mem(1)<="11111111";--1801H DATA_2;
mem(2)<="10101010";
elsif(rising_edge(writein)) then
mem(conv_integer(addr))<=data;
end if;
end process;
end behav;
```

➢

2E HALT

Program2 RISC_CPU

RISC_CPU RISC_CPU

20 HEX HLT

Program2 ROM2

RAM2

- **ROM**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity rom2 is
port (ena : in std_logic;
        readout : in std_logic;
          data : out std_logic_vector(7 downto 0);
          addr : in std_logic_vector(12 downto 0));
end rom2;
architecture behav of rom2 is
type rom_array is array(0 to 1023) of std_logic_vector (7 downto 0);
signal   mem:rom_array;
begin
mem(0)<="10111000";--          //   00    BEGIN:   LDA DATA_2
mem(1)<="00000001";
mem(2)<="01111000";--          //   02             AND DATA_3
mem(3)<="00000010";
mem(4)<="10011000";--          //   04           XOR DATA_2
mem(5)<="00000001";
mem(6)<="00100000";--          //   06           SKZ
mem(7)<="00000000";
mem(8)<="00000000";--          //   08           HLT                  //AND doesn't work
mem(9)<="00000000";
mem(10)<="01011000";--         //   0a           ADD DATA_1
mem(11)<="00000000";
mem(12)<="00100000";--         //   0c           SKZ
mem(13)<="00000000";
mem(14)<="11100000";--         //   0e           JMP ADD_OK
mem(15)<="00010010";
mem(16)<="00000000";--         //   10           HLT                  //ADD doesn't work
mem(17)<="00000000";
mem(18)<="10011000";--         //   12   ADD_OK:   XOR DATA_3
```

```
    mem(19)<="00000010";
    mem(20)<="01011000";--          //   14       ADD DATA_1              //FF plus 1 makes
-1
    mem(21)<="00000000";
    mem(22)<="11011000";--          //   16       STO TEMP
    mem(23)<="00000011";
    mem(24)<="10111000";--          //   18       LDA DATA_1
    mem(25)<="00000000";
    mem(26)<="01011000";--          //   1a         ADD TEMP          //-1   plus   1   should
make zero
    mem(27)<="00000011";
    mem(28)<="00100000";--          //   1c       SKZ
    mem(29)<="00000000";
    mem(30)<="00000000";--          //   1e             HLT         //ADD Doesn't work
    mem(31)<="00000000";
    mem(32)<="00000000";--          //   20   END:      HLT      //CONGRATULATIONS  -  TEST2
PASSED!
    mem(33)<="00000000";
    mem(34)<="11100000";--          //   22       JMP BEGIN           //run test again
    mem(35)<="00000000";
    process(addr,ena,readout)
    begin
    if(readout='1') then
    if(ena='1') then
    data<=mem(conv_integer(addr));
    else
    data<="ZZZZZZZZ";
    end if;
    end if;
    end process;
    end behav;
```

➢ **RAM**

```
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_unsigned.all;
    entity ram2 is
    port (rst : in std_logic;
            ena : in std_logic;
            writein,readout : in std_logic;
              data : inout std_logic_vector(7 downto 0);
              addr : in std_logic_vector(12 downto 0));
    end ram2;
    architecture behav of ram2 is
    type ram_array is array(0 to 255) of std_logic_vector (7 downto 0);
```

```
signal   mem:ram_array;
begin
process(readout,data,addr,ena)
begin
if(readout='1' and ena='1') then
data<=mem(conv_integer(addr));
else
data<="ZZZZZZZZ";
end if;
end process;
process(writein,rst)
begin
if(rst='1') then
  mem(0)<="00000001";--          //   1800           DATA_1:
//constant   1(hex)
  mem(1)<="10101010";--          //   1801           DATA_2:
//constant AA(hex)
  mem(2)<="11111111";--          //   1802           DATA_3:
//constant FF(hex)
  mem(3)<="00000000";--          //   1803           TEMP:
elsif(rising_edge(writein)) then
mem(conv_integer(addr))<=data;
end if;
end process;
end behav;
```

➢

20H HALT

ROM CPU_TEST.vwf, ACCUM DATA

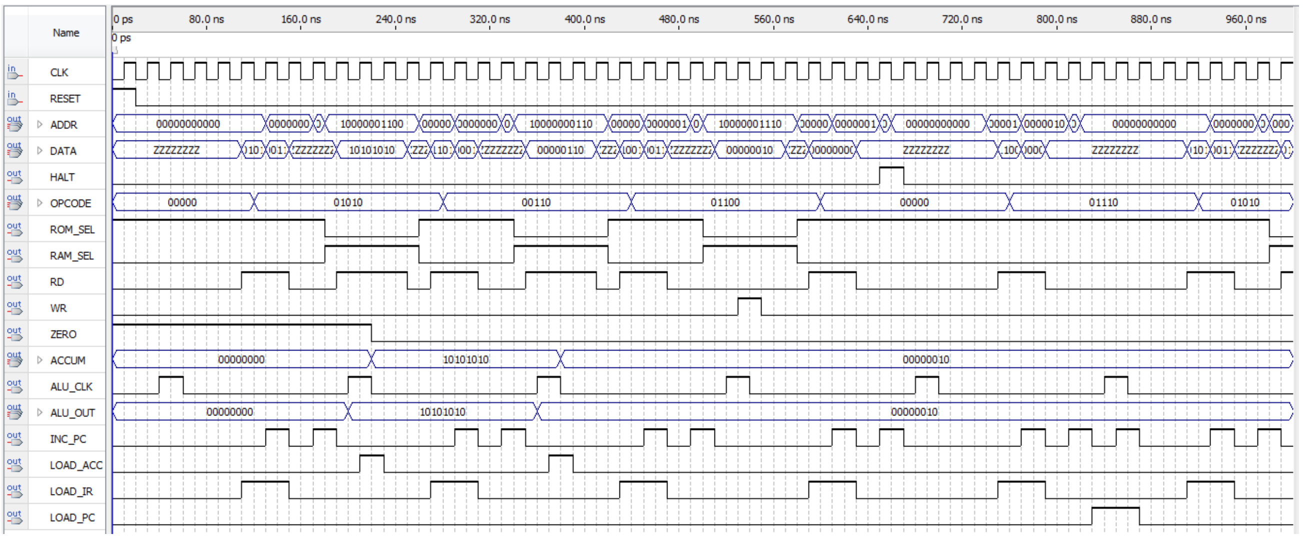
RAM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ram1 is
port (rst : in std_logic;
          ena : in std_logic;
          writein,readout : in std_logic;
             data : inout std_logic_vector(7 downto 0);
             addr : in std_logic_vector(9 downto 0));
end ram1;
```

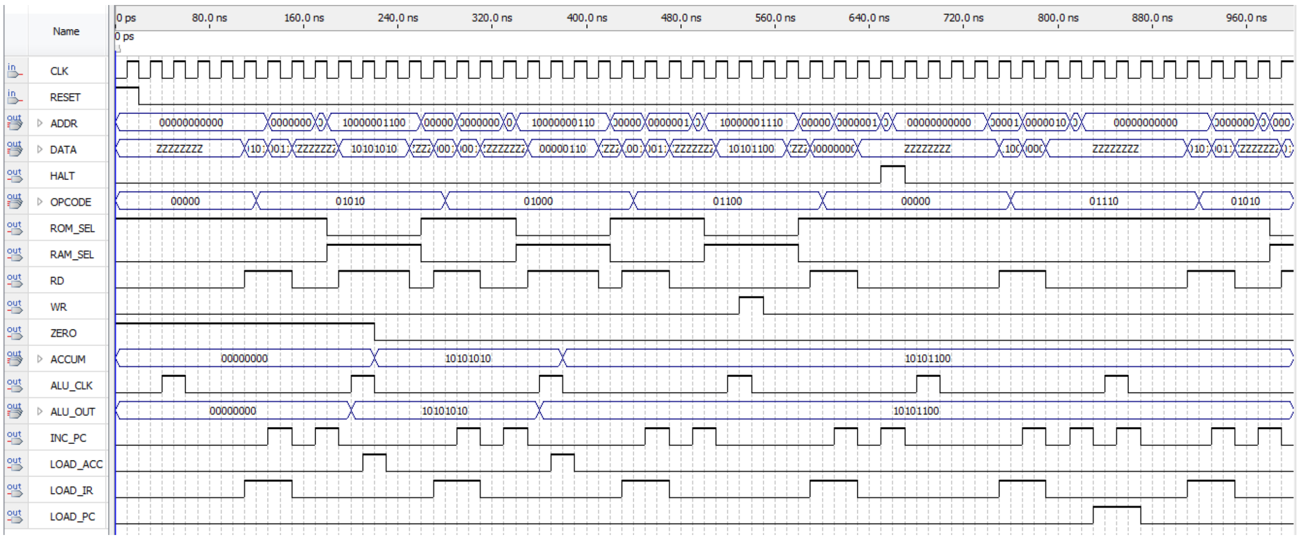
```
architecture behav of ram1 is
type ram_array is array(0 to 255) of std_logic_vector (7 downto 0);
signal   mem:ram_array;
begin
process(readout,data,addr,ena)
begin
if(readout='1' and ena='1') then
data<=mem(conv_integer(addr));
else
data<="ZZZZZZZZ";
end if;
end process;
process(writein,rst)
begin
if(rst='1') then
mem(0)<="00000000";--800H DATA_0;
mem(1)<="00000001";--801H DATA_1;
mem(2)<="00000010";--802H DATA_2;
mem(3)<="00000011";--803H DATA_3;
mem(4)<="00000100";--804H DATA_4;
mem(5)<="00000101";--805H DATA_5;
mem(6)<="00000110";--806H DATA_6;
mem(7)<="00000111";--807H DATA_7;
mem(8)<="00001000";--808H DATA_8;
mem(9)<="00001001";--809H DATA_9;
mem(10)<="00001010";--80aH DATA_10;
mem(11)<="11111111";--80bH DATA_11;
mem(12)<="10101010";--80cH DATA_12;
mem(13)<="01010101";--800H DATA_13;
elsif(rising_edge(writein)) then
mem(conv_integer(addr))<=data;
end if;
end process;
end behav;
```

ROM

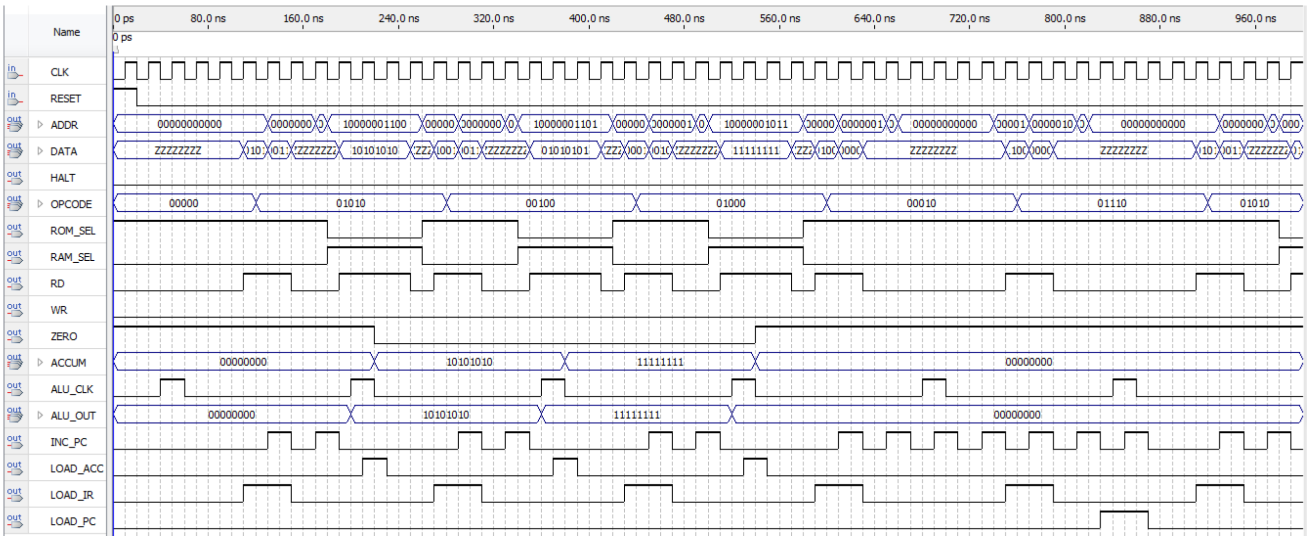
```
-------------------      -LDA/AND/STO/HALT/JMP----------------------
mem(0)<="01010100"; --00 begin:LDA DATA_12 "10101010"
mem(1)<="00001100";
mem(2)<="00110100"; --02 AND DATA_6             "00000110"
mem(3)<="00000110";
mem(4)<="01100100"; --04 STO DATA_14            "00000010"
mem(5)<="00001110";
mem(6)<="00000000"; --04 HALT
mem(7)<="00000000";
mem(8)<="01110000"; --05 JMP begin
mem(9)<="00000000";
```



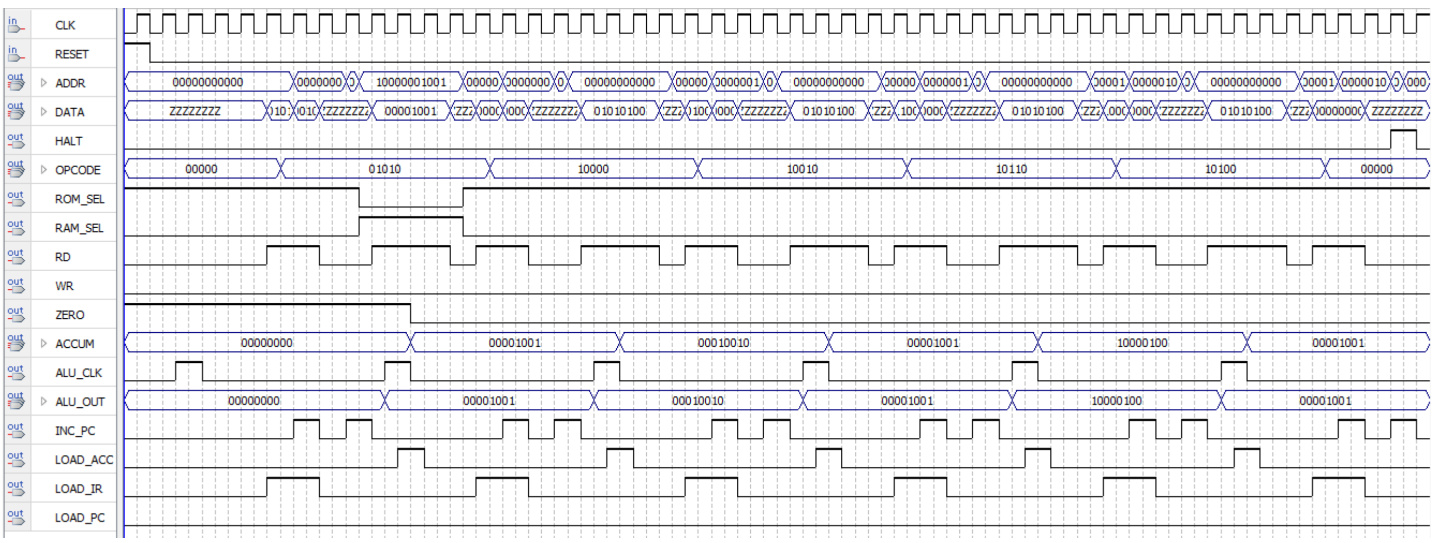
```
-----------------          LDA/XORR/STO/HALT/JMP--------------------
mem(0)<="01010100"; --00 begin:LDA DATA_12 "10101010"
mem(1)<="00001100";
mem(2)<="01000100"; --02 XORR DATA_6             "00000110"
mem(3)<="00000110";
mem(4)<="01100100"; --04 STO DATA_14             "10101100"
mem(5)<="00001110";
mem(6)<="00000000"; --04 HALT
mem(7)<="00000000";
mem(8)<="01110000"; --05 JMP begin
mem(9)<="00000000";
```



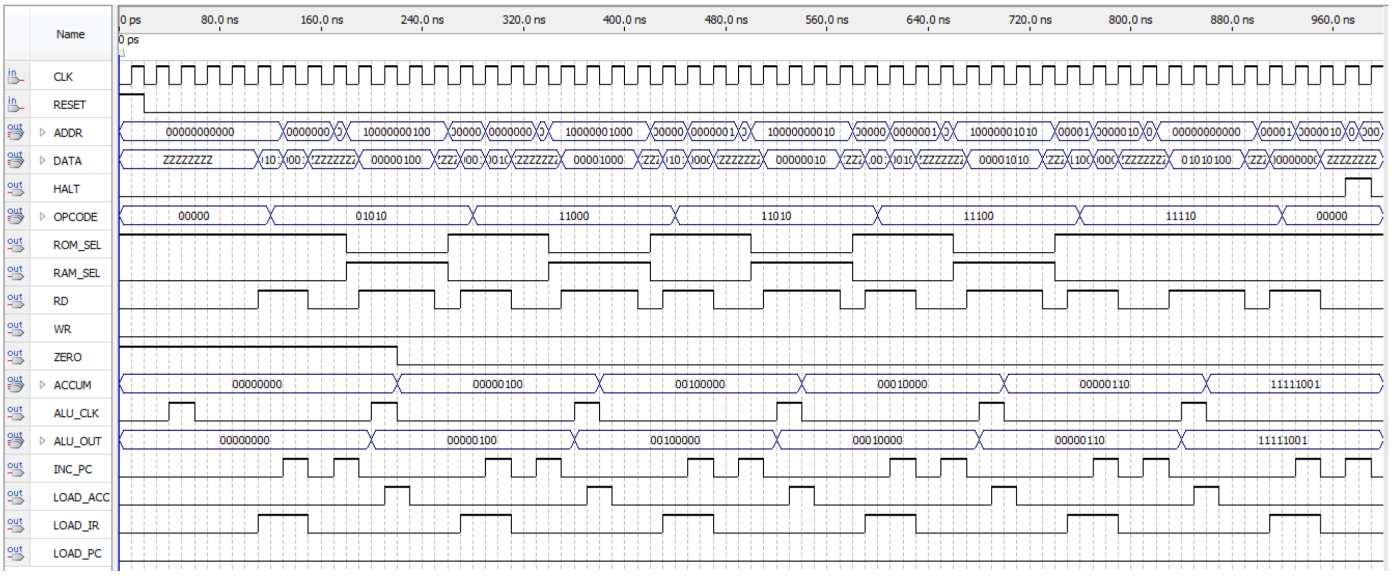
```
-----------------          LDA/XORR/SKZ/HALT/JMP--------------------
mem(0)<="01010100"; --00 begin:LDA DATA_12 "10101010"
mem(1)<="00001100";
mem(2)<="00100100"; --02 ADD DATA_13            "01010101"
mem(3)<="00001101";
mem(4)<="01000100"; --06 ANDD DATA_11           "11111111"
mem(5)<="00001011";
mem(6)<="00010000"; --08 SKZ
mem(7)<="00000000";
mem(8)<="00000000"; --08 HALT
mem(9)<="00000000";
mem(10)<="01110000"; --0a JMP begin
mem(11)<="00000000";
```



```
--------------          LDA/SLL/SRL/ROR/ROL/HALT---------------
mem(0)<="01010100"; --00 begin:LDA DATA_9      "00001001"
mem(1)<="00001001";
mem(2)<="10000000"; --02 SLL                   "00010010"
mem(3)<="00000000";
mem(4)<="10010000"; --06 SRL                   "00001001"
mem(5)<="00000000";
mem(6)<="10110000"; --08 ROR                   "10000100"
mem(7)<="00000000";
mem(8)<="10100000"; --08 ROL                   "00001001"
mem(9)<="00000000";
mem(10)<="00000000"; --0a HALT
mem(11)<="00000000";
```

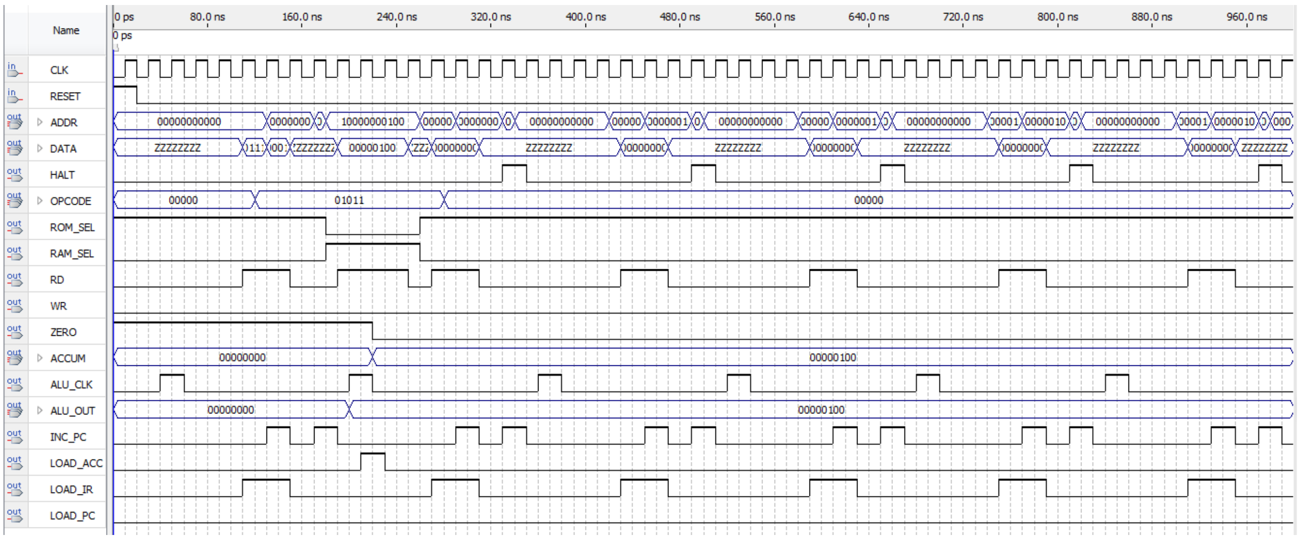


```
--------------        LDA/MUL/DIV/SUB/NOTT/HALT---------------
mem(0)<="01010100"; --00 begin:LDA DATA_4              "00000100"
mem(1)<="00000100";
mem(2)<="11000100"; --02 MUL DATA_8                    "00100000"
mem(3)<="00001000";
mem(4)<="11010100"; --06 DIV DATA_2                    "00010000"
mem(5)<="00000010";
mem(6)<="11100100"; --08 SUB DATA_10                   "00000110"
mem(7)<="00001010";
mem(8)<="11110000"; --08 NOTT                          "11111001"
mem(9)<="00000000";
mem(10)<="00000000"; --0a HALT
mem(11)<="00000000";
```



------------------------   LDA#/HLT   -----------------------

```
mem(0)<="01011100"; --00 begin:LDA# DATA_4                    "00000100"
mem(1)<="00000100";
mem(2)<="00000000"; --02 HALT                            "00100000"
mem(3)<="00000000";
```



1 2 3

RISC_CPU

[1].   .Verilog

[2].   ,   ,   .CPU