Faculty of Engineering

Computer Engineering Department

Computer Architecture Lab Project ” MIPS processor Simulation”

Submitted to:

Dr. Khaled Badran

Faculty of Engineering, Computer Engineering Department

Computer Architecture (21COMP05I)

Submitted by:

Name: Mahmoud Ashraf Mahmoud Ahmed, ID: 204858, Group: A2

Name: Abdelrahman Mostafa, ID: 206395, Group: A2

Name: Pierre Tamer, ID: 198987, Group: A2

Name: Tomas Ehab, ID: 181954, Group: A2

April 2022, Cairo

Table of Contents

I. Introduction

The MIPS processor, developed by Stanford University researchers in 1984, is a RISC (Reduced Instruction Set Computer) processor.

RISC processors usually support fewer and simpler instructions than CISC (Complex Instruction Set Computer) counterparts (such as Intel Pentium processors).

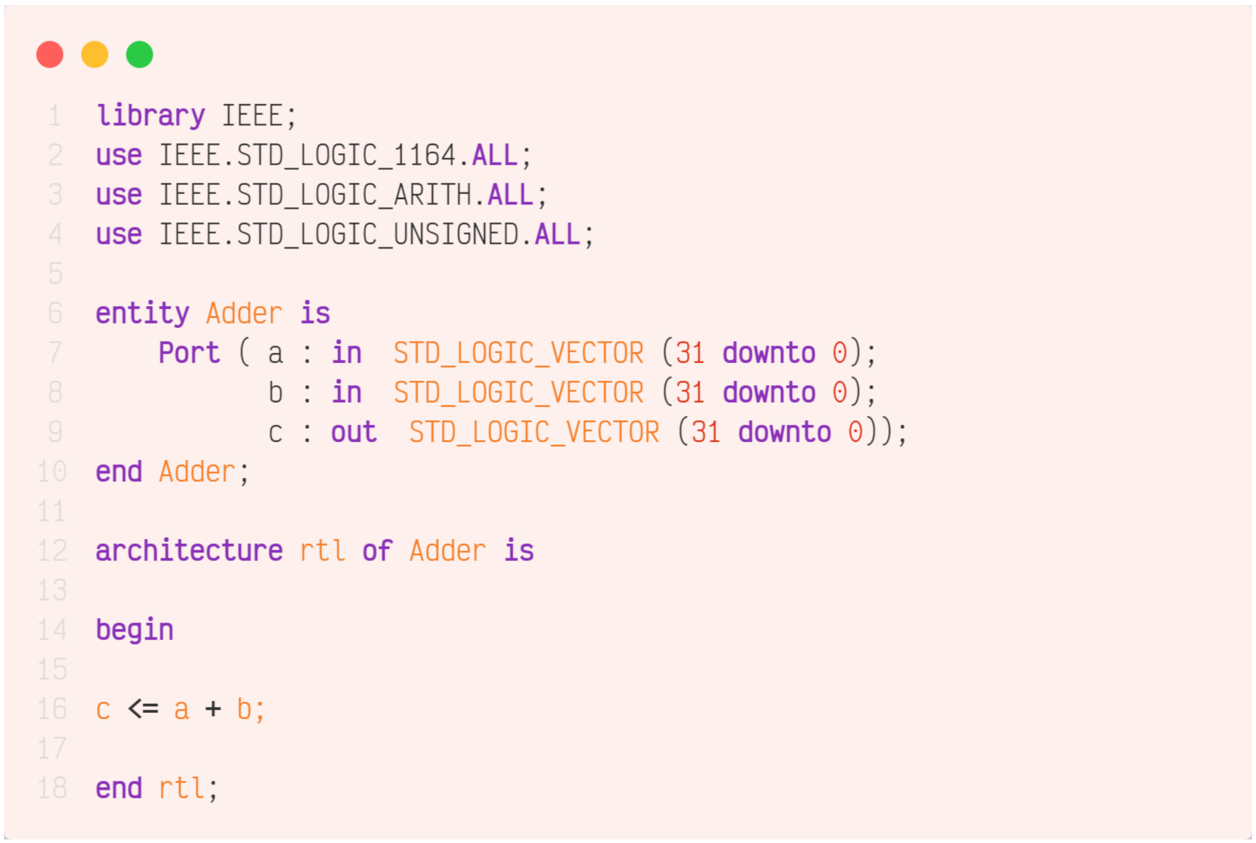
The concept is, however, that because of its simpler design, a RISC processor can be made much faster than a CISC processor.

Nowadays, it is widely accepted that RISC processors are more efficient than CISC processors; in fact, the only popular CISC processor still in use (Intel Pentium) internally translates CISC instructions into RISC instructions before they are executed.

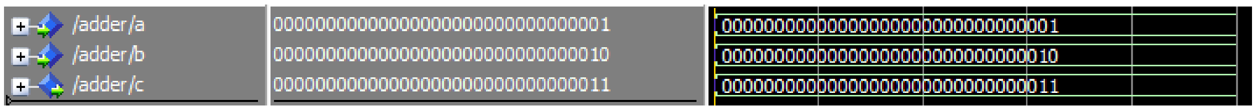
All code for this project can be accessed from: https://github.com/uptotec/comparch-project-s2

II. VHDL Coding & Simulation of Single Blocks

1. Adder Block



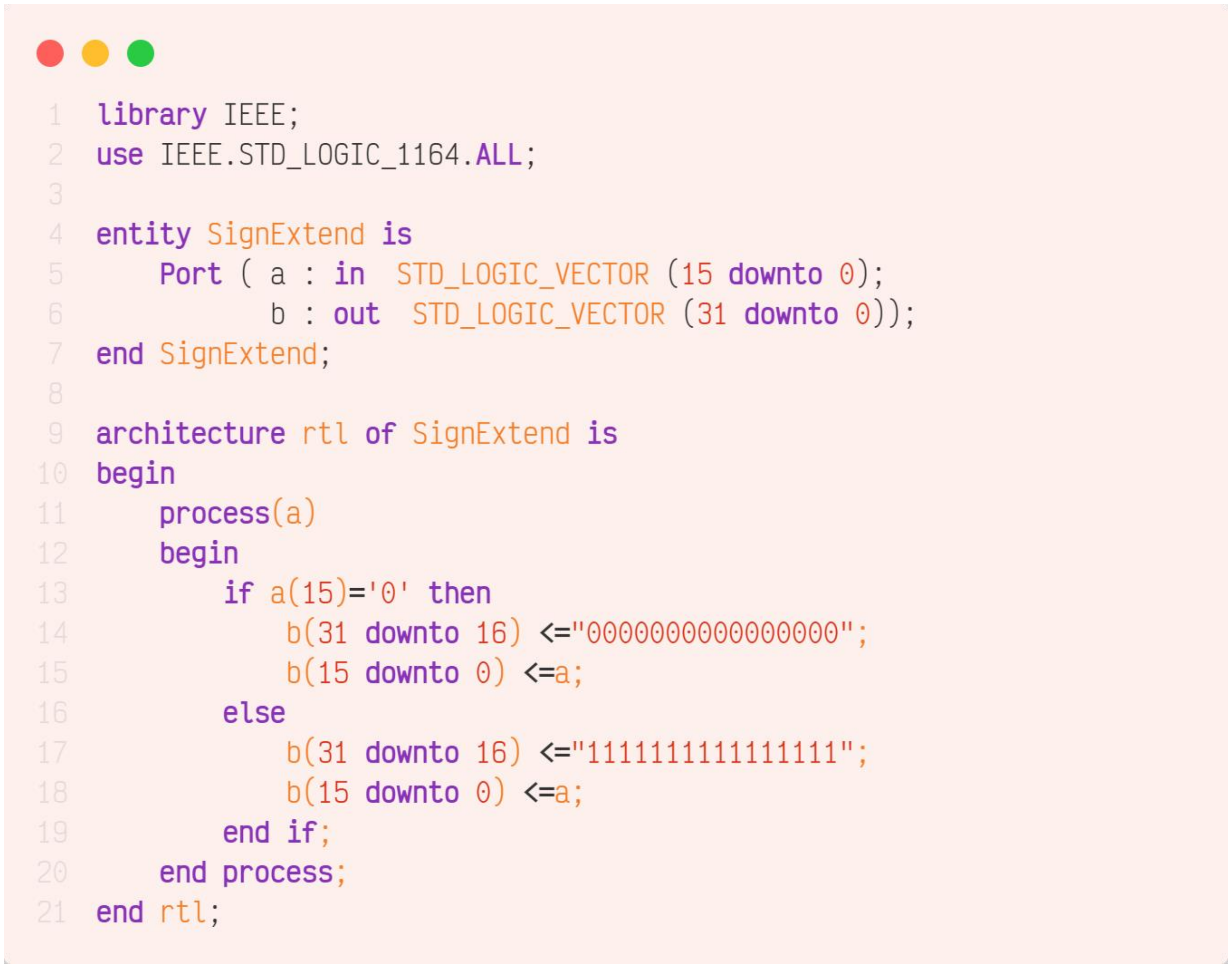
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Adder is
    Port ( a : in  STD_LOGIC_VECTOR (31 downto 0);
           b : in  STD_LOGIC_VECTOR (31 downto 0);
           c : out  STD_LOGIC_VECTOR (31 downto 0));
end Adder;

architecture rtl of Adder is

begin

c <= a + b;

end rtl;
```



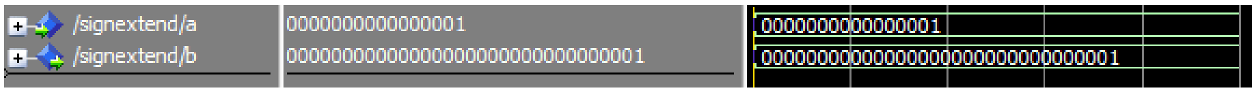
```
/adder/a   00000000000000000000000000000001   00000000000000000000000000000001
/adder/b   00000000000000000000000000000010   00000000000000000000000000000010
/adder/c   00000000000000000000000000000011   00000000000000000000000000000011
```

```
force -freeze sim:/adder/a 16#00000001 0
force -freeze sim:/adder/b 16#00000002 0
run
```

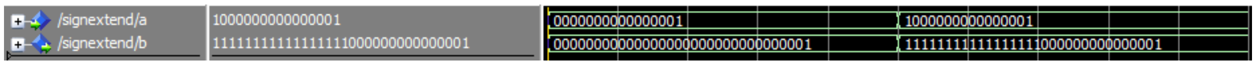
2. Sign Extend



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SignExtend is
    Port ( a : in  STD_LOGIC_VECTOR (15 downto 0);
           b : out  STD_LOGIC_VECTOR (31 downto 0));
end SignExtend;

architecture rtl of SignExtend is
begin
    process(a)
    begin
        if a(15)='0' then
            b(31 downto 16) <="0000000000000000";
            b(15 downto 0) <=a;
        else
            b(31 downto 16) <="1111111111111111";
            b(15 downto 0) <=a;
        end if;
    end process;
end rtl;
```



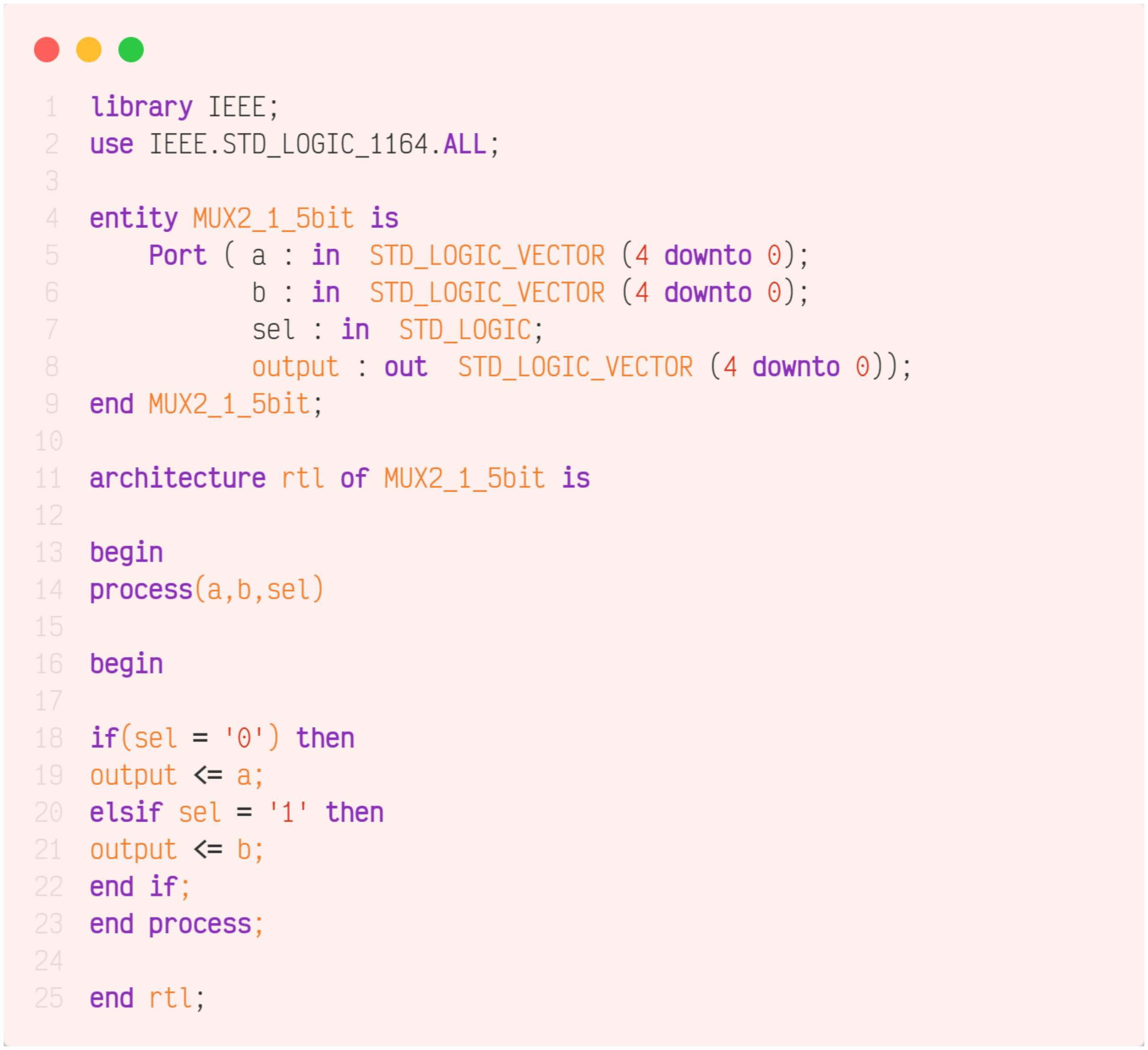
```
/signextend/a   0000000000000001                   0000000000000001
/signextend/b   00000000000000000000000000000001   00000000000000000000000000000001
```



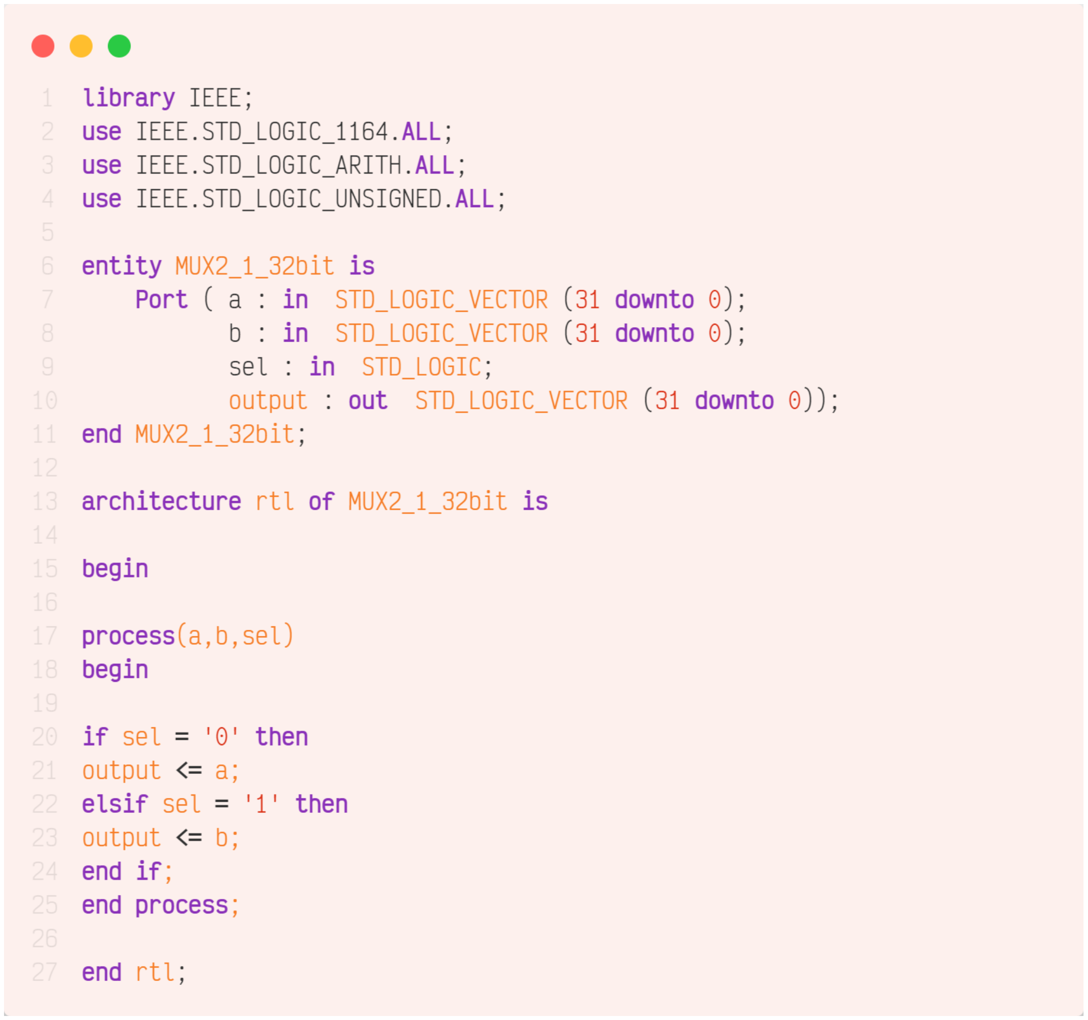
```
/signextend/a   1000000000000001                   0000000000000001                   1000000000000001
/signextend/b   11111111111111111000000000000001   00000000000000000000000000000001   11111111111111111000000000000001
```

```
force -freeze sim:/signextend/a 16#0001 0

run

force -freeze sim:/signextend/a 1000000000000001 0

run
```

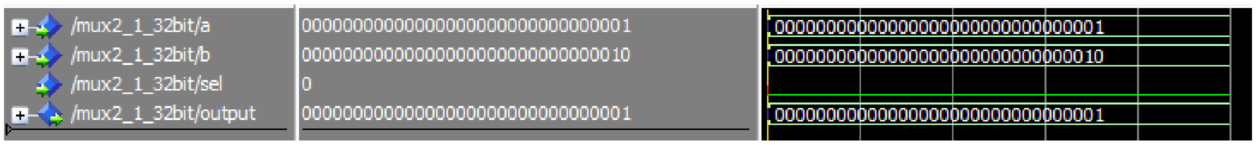
3. 2x1 MUX 32 bits & 5 bits



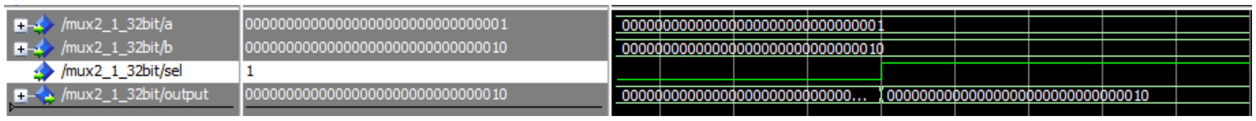
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX2_1_5bit is
    Port ( a : in  STD_LOGIC_VECTOR (4 downto 0);
           b : in  STD_LOGIC_VECTOR (4 downto 0);
           sel : in  STD_LOGIC;
           output : out  STD_LOGIC_VECTOR (4 downto 0));
end MUX2_1_5bit;

architecture rtl of MUX2_1_5bit is

begin
process(a,b,sel)

begin

if(sel = '0') then
output <= a;
elsif sel = '1' then
output <= b;
end if;
end process;

end rtl;
```



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUX2_1_32bit is
    Port ( a : in  STD_LOGIC_VECTOR (31 downto 0);
           b : in  STD_LOGIC_VECTOR (31 downto 0);
           sel : in  STD_LOGIC;
           output : out  STD_LOGIC_VECTOR (31 downto 0));
end MUX2_1_32bit;

architecture rtl of MUX2_1_32bit is

begin

process(a,b,sel)
begin

if sel = '0' then
output <= a;
elsif sel = '1' then
output <= b;
end if;
end process;

end rtl;
```



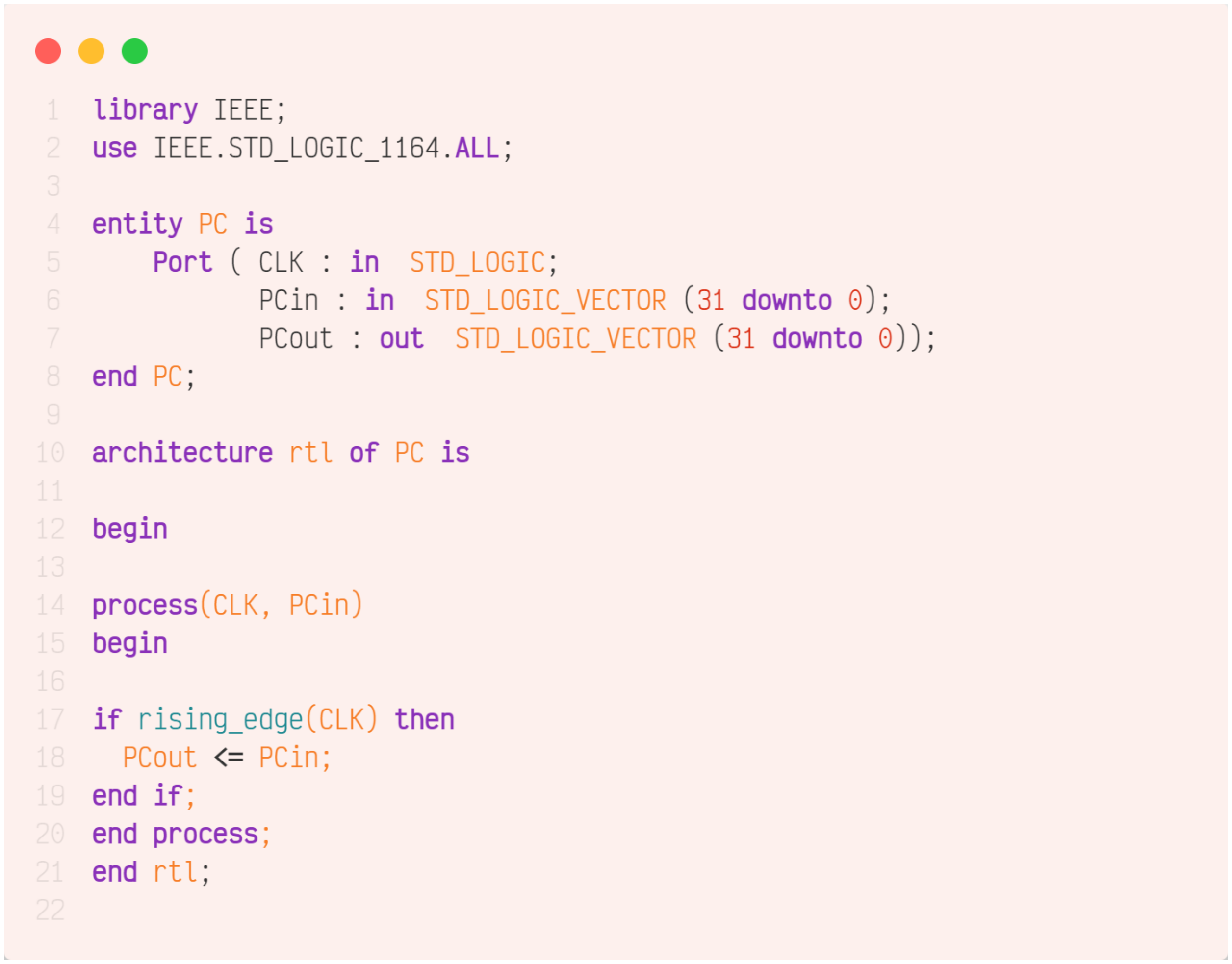
```
/mux2_1_32bit/a       00000000000000000000000000000001   00000000000000000000000000000001
/mux2_1_32bit/b       00000000000000000000000000000010   00000000000000000000000000000010
/mux2_1_32bit/sel     0
/mux2_1_32bit/output  00000000000000000000000000000001   00000000000000000000000000000001
```



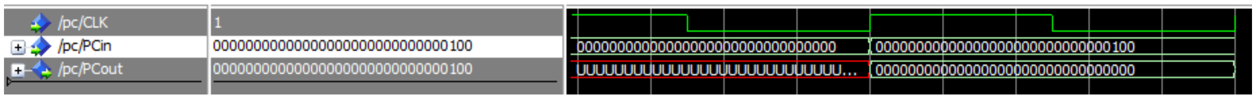
```
/mux2_1_32bit/a       00000000000000000000000000000001   00000000000000000000000000000001
/mux2_1_32bit/b       00000000000000000000000000000010   00000000000000000000000000000010
/mux2_1_32bit/sel     1
/mux2_1_32bit/output  00000000000000000000000000000010   000000000000000000000000000...   00000000000000000000000000000010
```

```
force -freeze sim:/mux2_1_32bit/a 16#00000001 0
force -freeze sim:/mux2_1_32bit/b 16#00000002 0
force -freeze sim:/mux2_1_32bit/sel 0 0
run
force -freeze sim:/mux2_1_32bit/sel 1 0
run
```

4. Program Counter



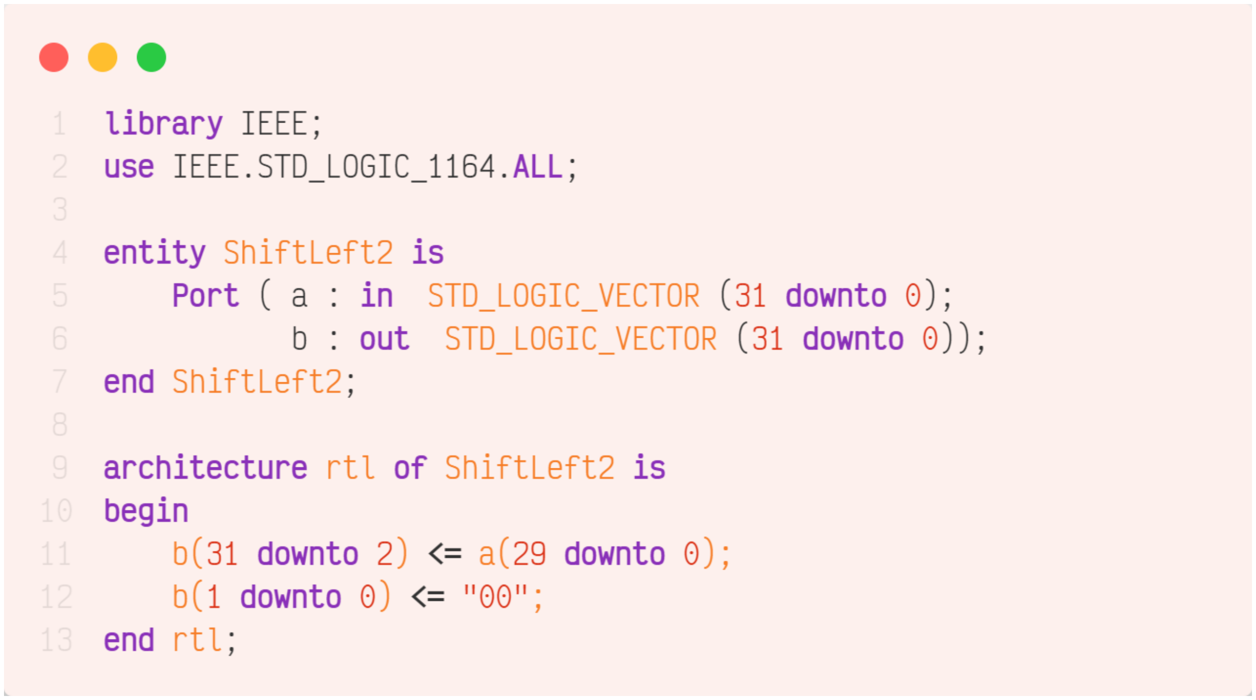
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PC is
    Port ( CLK : in  STD_LOGIC;
           PCin : in  STD_LOGIC_VECTOR (31 downto 0);
           PCout : out  STD_LOGIC_VECTOR (31 downto 0));
end PC;

architecture rtl of PC is

begin

process(CLK, PCin)
begin

if rising_edge(CLK) then
  PCout <= PCin;
end if;
end process;
end rtl;

```



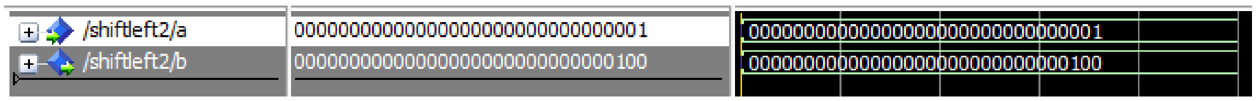
```
/pc/CLK     1
/pc/PCin    00000000000000000000000000000100    00000000000000000000000000000000    00000000000000000000000000000100
/pc/PCout   00000000000000000000000000000100    UUUUUUUUUUUUUUUUUUUUUUUUUUUUU...    00000000000000000000000000000000
```

```
force -freeze sim:/pc/CLK 1 0, 0 {50 ps} -r 100

force -freeze sim:/pc/PCin 16#00000000 0

run

force -freeze sim:/pc/PCin 00000000000000000000000000000100 0

run
```

5. Shift Left 2



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ShiftLeft2 is
    Port ( a : in  STD_LOGIC_VECTOR (31 downto 0);
           b : out  STD_LOGIC_VECTOR (31 downto 0));
end ShiftLeft2;

architecture rtl of ShiftLeft2 is
begin
    b(31 downto 2) <= a(29 downto 0);
    b(1 downto 0) <= "00";
end rtl;
```

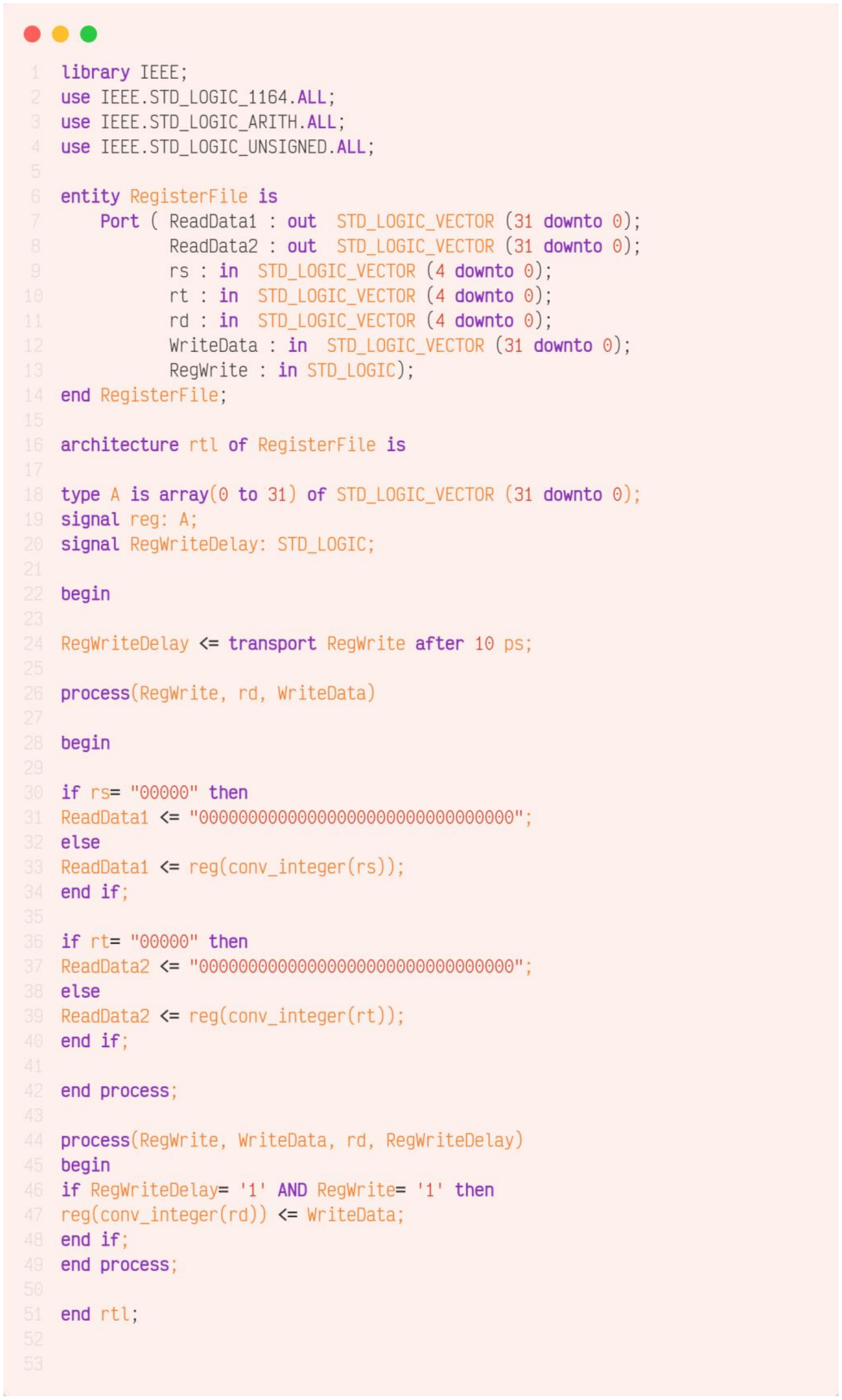


```
/shiftleft2/a   00000000000000000000000000000001   00000000000000000000000000000001
/shiftleft2/b   00000000000000000000000000000100   00000000000000000000000000000100
```

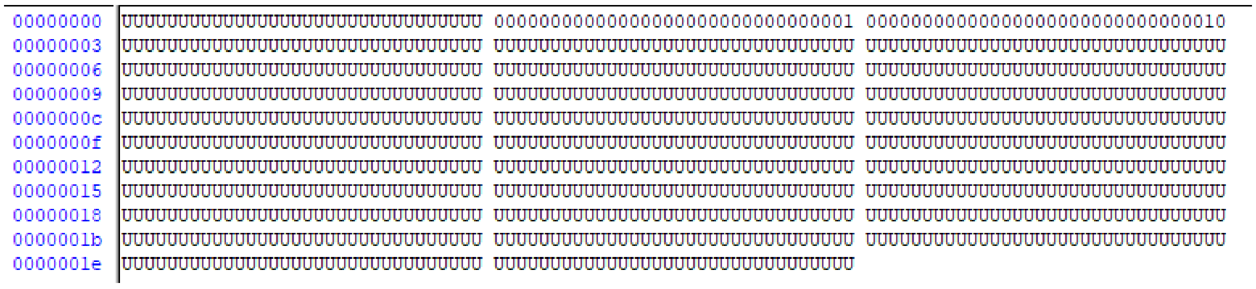
force -freeze sim:/shiftleft2/a 16#00000001 0

run

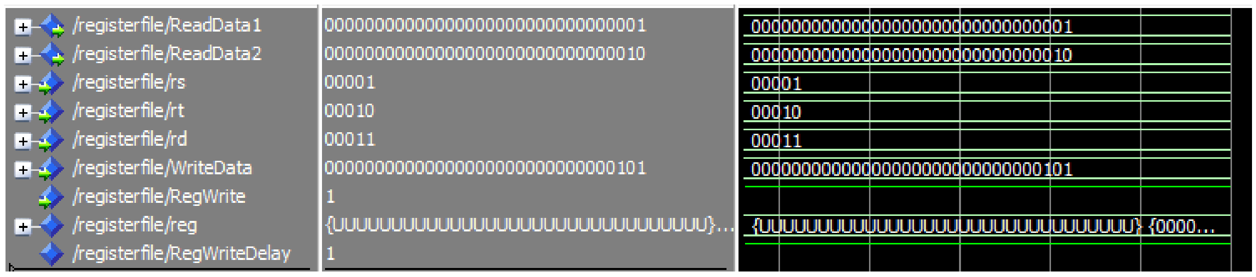
6. Register File



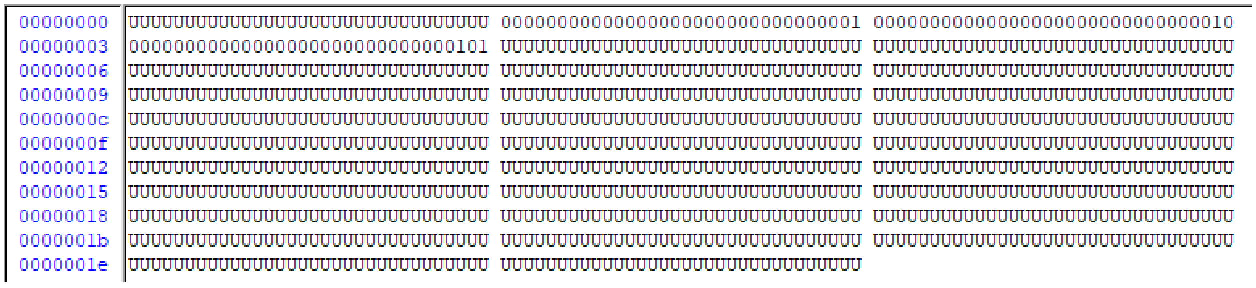
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RegisterFile is
    Port ( ReadData1 : out  STD_LOGIC_VECTOR (31 downto 0);
           ReadData2 : out  STD_LOGIC_VECTOR (31 downto 0);
           rs : in  STD_LOGIC_VECTOR (4 downto 0);
           rt : in  STD_LOGIC_VECTOR (4 downto 0);
           rd : in  STD_LOGIC_VECTOR (4 downto 0);
           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
           RegWrite : in STD_LOGIC);
end RegisterFile;

architecture rtl of RegisterFile is

type A is array(0 to 31) of STD_LOGIC_VECTOR (31 downto 0);
signal reg: A;
signal RegWriteDelay: STD_LOGIC;

begin

RegWriteDelay <= transport RegWrite after 10 ps;

process(RegWrite, rd, WriteData)

begin

if rs= "00000" then
ReadData1 <= "00000000000000000000000000000000";
else
ReadData1 <= reg(conv_integer(rs));
end if;

if rt= "00000" then
ReadData2 <= "00000000000000000000000000000000";
else
ReadData2 <= reg(conv_integer(rt));
end if;

end process;

process(RegWrite, WriteData, rd, RegWriteDelay)
begin
if RegWriteDelay= '1' AND RegWrite= '1' then
reg(conv_integer(rd)) <= WriteData;
end if;
end process;

end rtl;
```



```
00000000 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU 00000000000000000000000000000001 00000000000000000000000000000010
00000003 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000006 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000009 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000000c UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000000f UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000012 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000015 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000018 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000001b UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000001e UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```



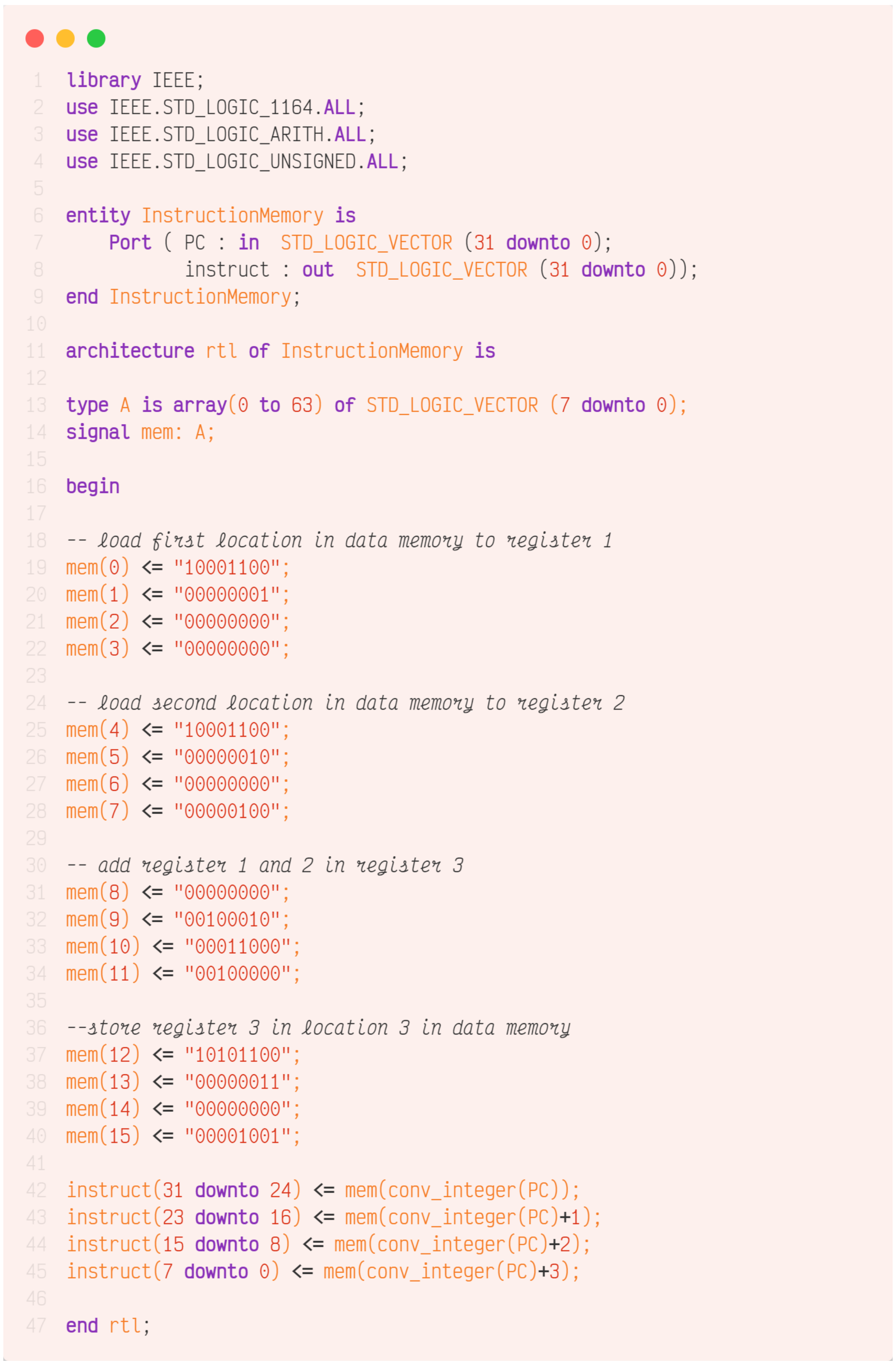
```
/registerfile/ReadData1     00000000000000000000000000000001     00000000000000000000000000000001
/registerfile/ReadData2     00000000000000000000000000000010     00000000000000000000000000000010
/registerfile/rs            00001                                00001
/registerfile/rt            00010                                00010
/registerfile/rd            00011                                00011
/registerfile/WriteData     00000000000000000000000000000101     00000000000000000000000000000101
/registerfile/RegWrite      1
/registerfile/reg           {UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU}...  {UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU} {0000...
/registerfile/RegWriteDelay 1
```



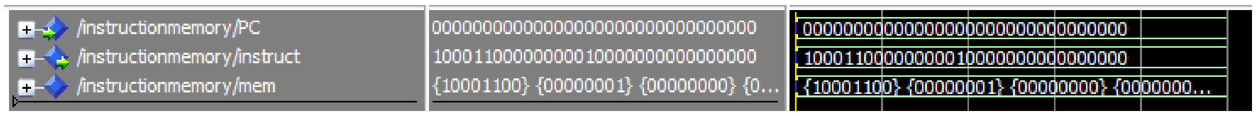
```
00000000 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU 00000000000000000000000000000001 00000000000000000000000000000010
00000003 00000000000000000000000000000101 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000006 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000009 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000000c UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000000f UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000012 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000015 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000018 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000001b UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000001e UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

```
mem load -filltype value -filldata 16#00000001 -fillradix symbolic /registerfile/reg(1)
mem load -filltype value -filldata 16#00000002 -fillradix symbolic /registerfile/reg(2)
force -freeze sim:/registerfile/rs 00001 0
force -freeze sim:/registerfile/rt 00010 0
force -freeze sim:/registerfile/rd 00011 0
force -freeze sim:/registerfile/WriteData 16#00000005 0
force -freeze sim:/registerfile/RegWrite 1 0
force -freeze sim:/registerfile/RegWriteDelay 1 0
run
```

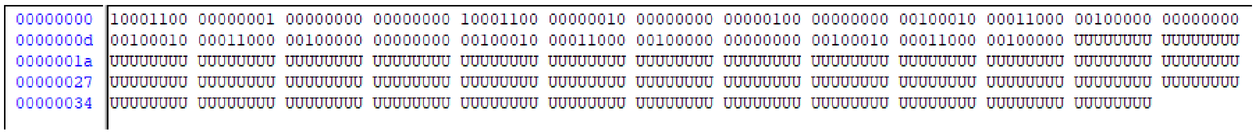
7. Instruction Memory



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity InstructionMemory is
    Port ( PC : in  STD_LOGIC_VECTOR (31 downto 0);
           instruct : out  STD_LOGIC_VECTOR (31 downto 0));
end InstructionMemory;

architecture rtl of InstructionMemory is

type A is array(0 to 63) of STD_LOGIC_VECTOR (7 downto 0);
signal mem: A;

begin

-- load first location in data memory to register 1
mem(0) <= "10001100";
mem(1) <= "00000001";
mem(2) <= "00000000";
mem(3) <= "00000000";

-- load second location in data memory to register 2
mem(4) <= "10001100";
mem(5) <= "00000010";
mem(6) <= "00000000";
mem(7) <= "00000100";

-- add register 1 and 2 in register 3
mem(8) <= "00000000";
mem(9) <= "00100010";
mem(10) <= "00011000";
mem(11) <= "00100000";

--store register 3 in location 3 in data memory
mem(12) <= "10101100";
mem(13) <= "00000011";
mem(14) <= "00000000";
mem(15) <= "00001001";

instruct(31 downto 24) <= mem(conv_integer(PC));
instruct(23 downto 16) <= mem(conv_integer(PC)+1);
instruct(15 downto 8) <= mem(conv_integer(PC)+2);
instruct(7 downto 0) <= mem(conv_integer(PC)+3);

end rtl;
```



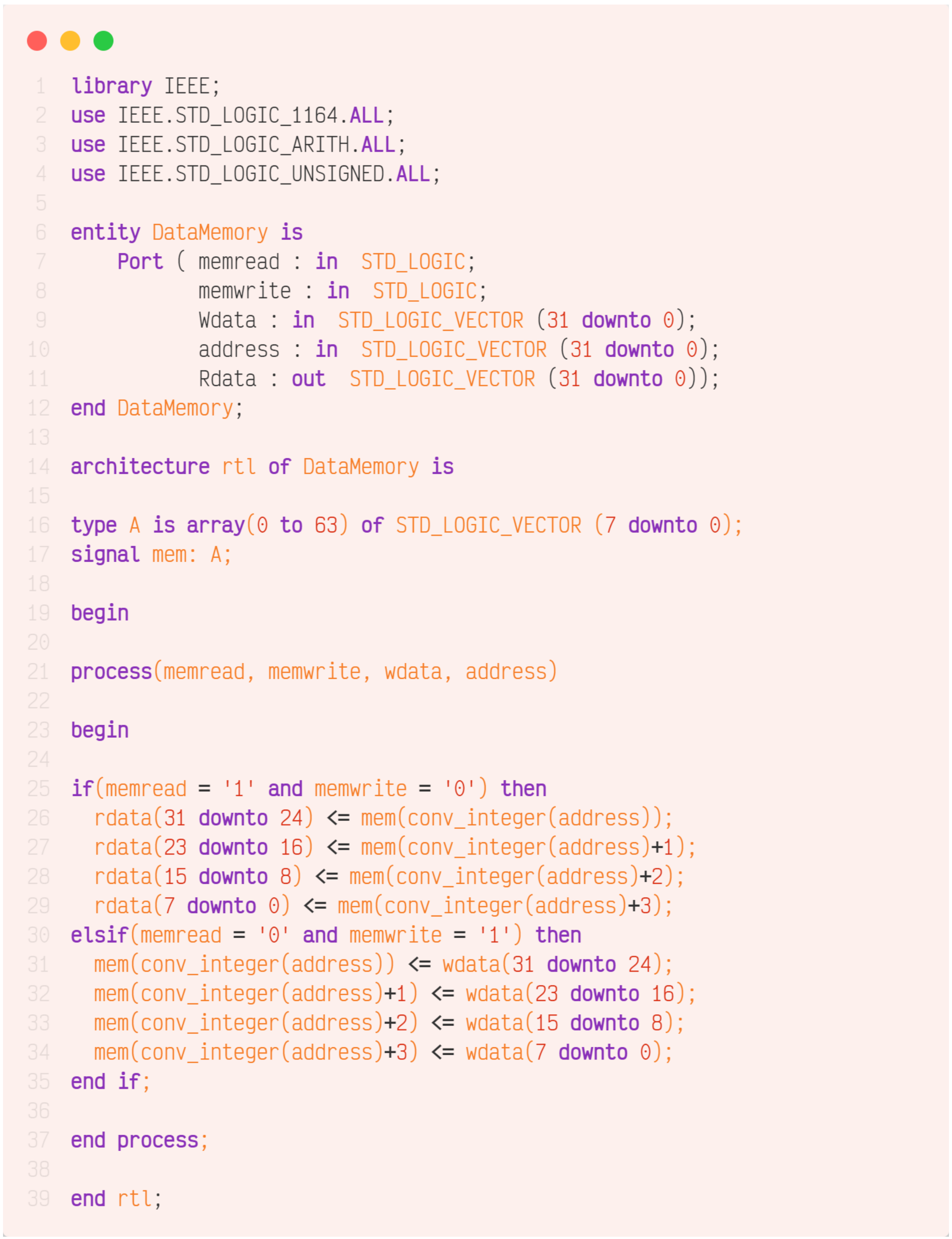
```
/instructionmemory/PC        00000000000000000000000000000000        00000000000000000000000000000000
/instructionmemory/instruct  10001100000000010000000000000000        10001100000000010000000000000000
/instructionmemory/mem       {10001100} {00000001} {00000000} {0...  {10001100} {00000001} {00000000} {0000000...
```



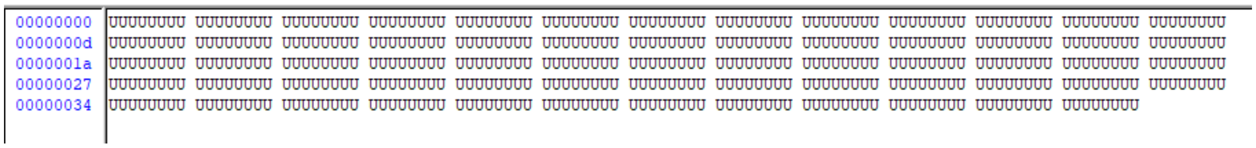
```
00000000 10001100 00000001 00000000 00000000 10001100 00000010 00000000 00000100 00000000 00100010 00011000 00100000 00000000
0000000d 00100010 00011000 00100000 00000000 00100010 00011000 00100000 00000000 00100010 00011000 00100000 UUUUUUUU UUUUUUUU
0000001a UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000027 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000034 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
```

```
force -freeze sim:/instructionmemory/PC 16#00000000 0

run
```

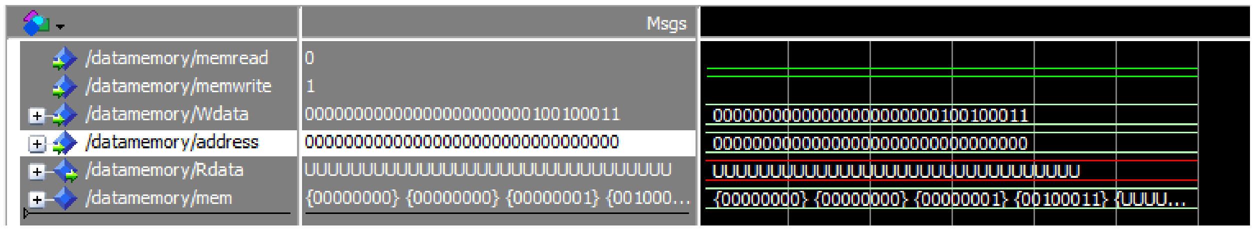
8. Data Memory



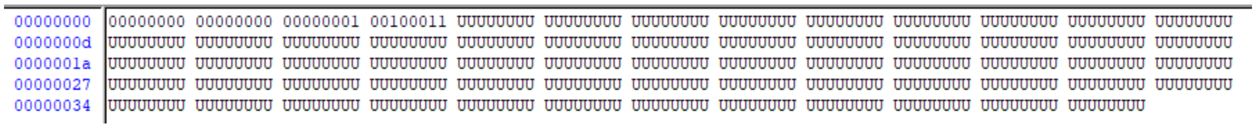
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DataMemory is
    Port ( memread : in  STD_LOGIC;
           memwrite : in  STD_LOGIC;
           Wdata : in  STD_LOGIC_VECTOR (31 downto 0);
           address : in  STD_LOGIC_VECTOR (31 downto 0);
           Rdata : out  STD_LOGIC_VECTOR (31 downto 0));
end DataMemory;

architecture rtl of DataMemory is

type A is array(0 to 63) of STD_LOGIC_VECTOR (7 downto 0);
signal mem: A;

begin

process(memread, memwrite, wdata, address)

begin

if(memread = '1' and memwrite = '0') then
  rdata(31 downto 24) <= mem(conv_integer(address));
  rdata(23 downto 16) <= mem(conv_integer(address)+1);
  rdata(15 downto 8) <= mem(conv_integer(address)+2);
  rdata(7 downto 0) <= mem(conv_integer(address)+3);
elsif(memread = '0' and memwrite = '1') then
  mem(conv_integer(address)) <= wdata(31 downto 24);
  mem(conv_integer(address)+1) <= wdata(23 downto 16);
  mem(conv_integer(address)+2) <= wdata(15 downto 8);
  mem(conv_integer(address)+3) <= wdata(7 downto 0);
end if;

end process;

end rtl;
```



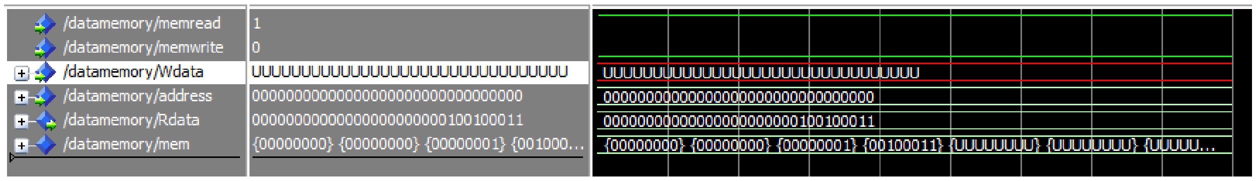
```
00000000 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
0000000d UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
0000001a UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000027 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000034 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
```



```
                       Msgs
/datamemory/memread    0
/datamemory/memwrite   1
/datamemory/Wdata      00000000000000000000000100100011
/datamemory/address    00000000000000000000000000000000
/datamemory/Rdata      UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/datamemory/mem        {00000000} {00000000} {00000001} {001000...
```



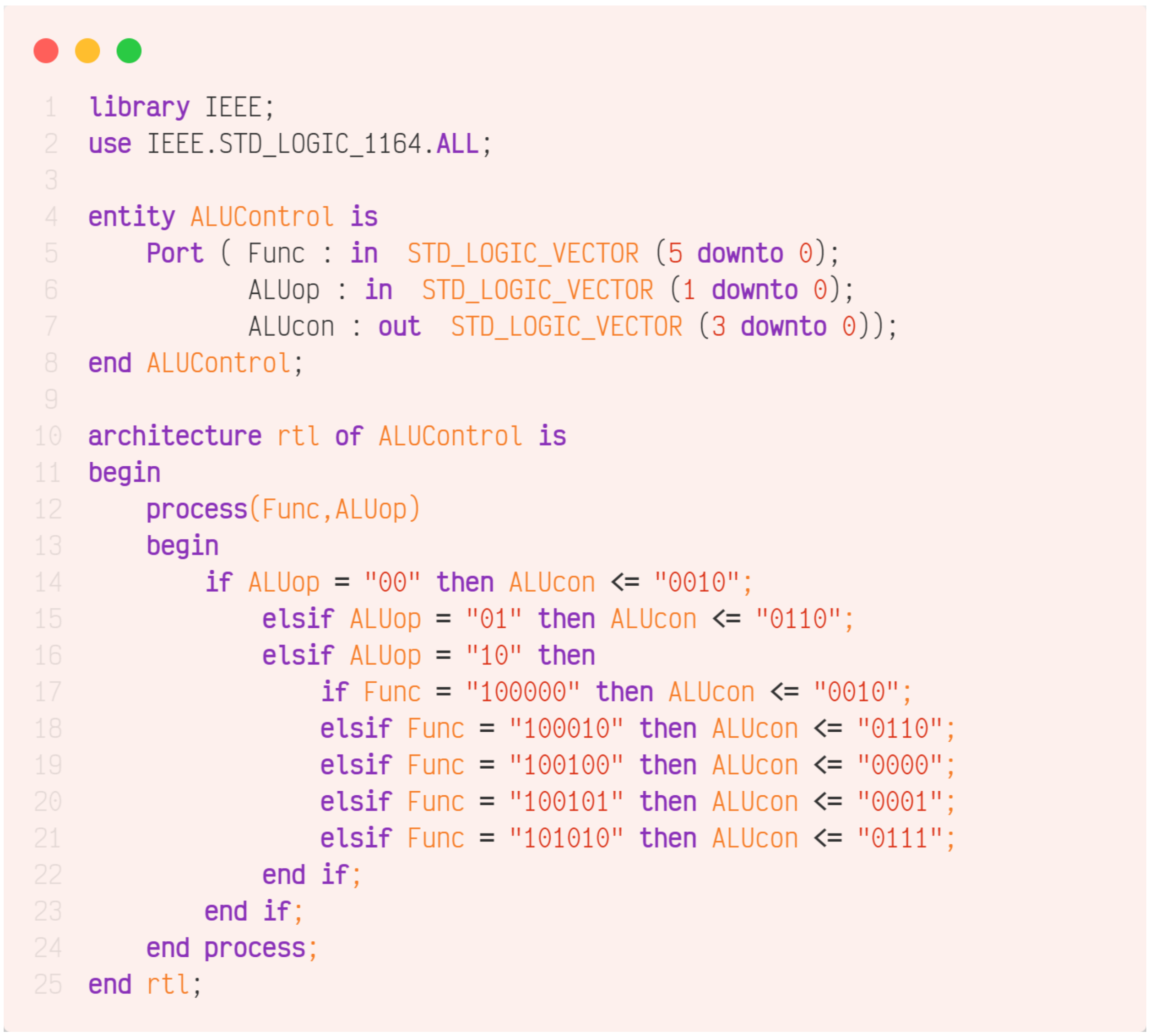
```
00000000 00000000 00000000 00000001 00100011 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
0000000d UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
0000001a UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000027 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000034 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
```



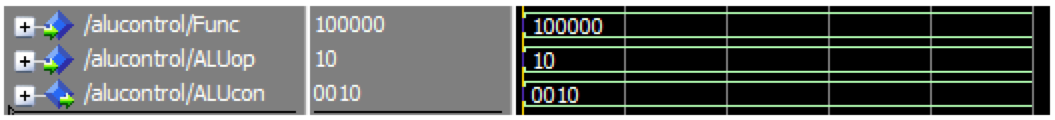
```
/datamemory/memread    1
/datamemory/memwrite   0
/datamemory/Wdata      UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/datamemory/address    00000000000000000000000000000000
/datamemory/Rdata      00000000000000000000000100100011
/datamemory/mem        {00000000} {00000000} {00000001} {001000...
```

```
force -freeze sim:/datamemory/memwrite 1 0
force -freeze sim:/datamemory/memread 0 0
force -freeze sim:/datamemory/Wdata 16#000000123 0
force -freeze sim:/datamemory/address 16#00000000 0
run
force -freeze sim:/datamemory/memread 1 0
force -freeze sim:/datamemory/memwrite 0 0
noforce sim:/datamemory/Wdata
run
```

9. ALU Control



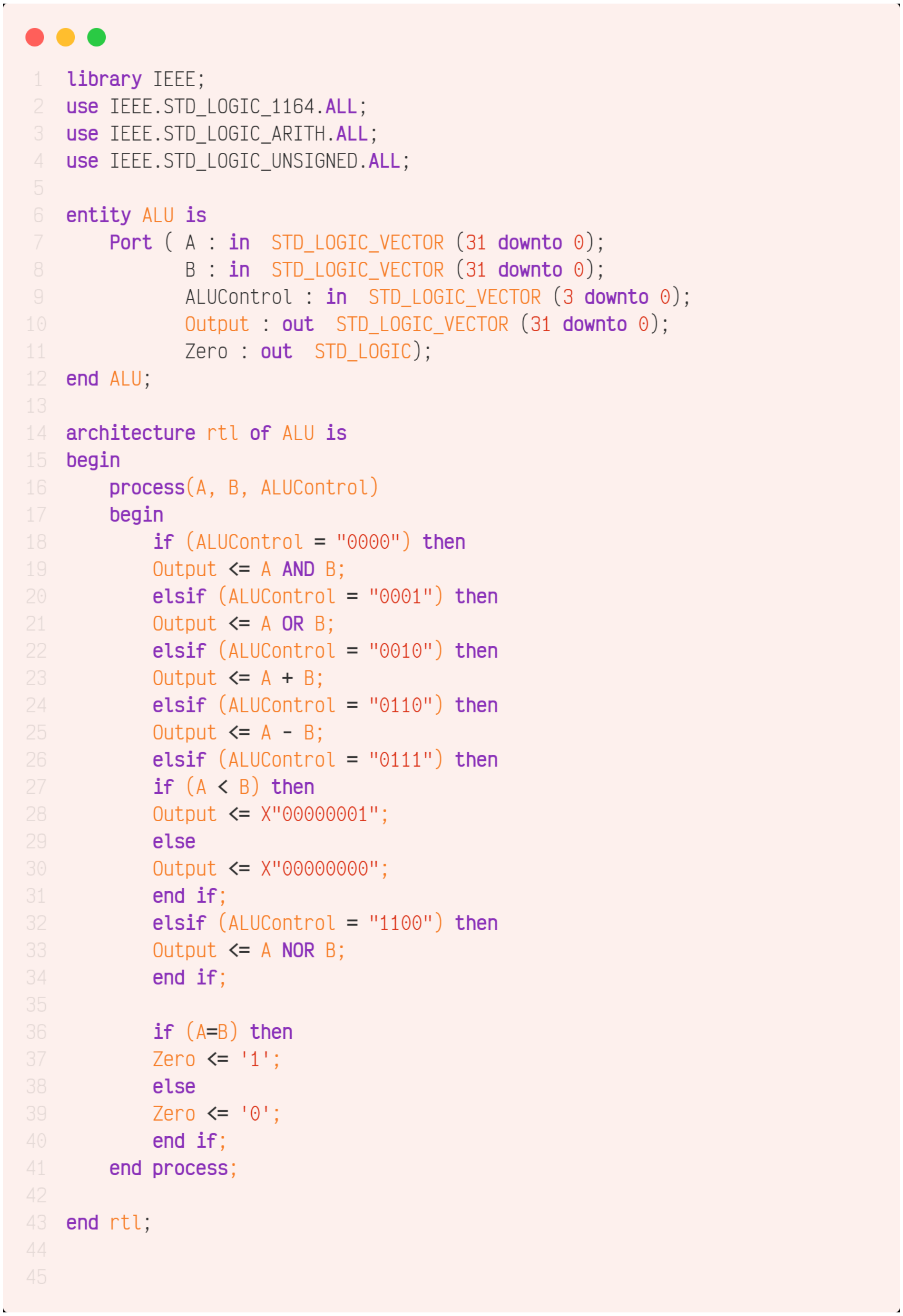
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ALUControl is
    Port ( Func : in  STD_LOGIC_VECTOR (5 downto 0);
           ALUop : in  STD_LOGIC_VECTOR (1 downto 0);
           ALUcon : out  STD_LOGIC_VECTOR (3 downto 0));
end ALUControl;

architecture rtl of ALUControl is
begin
    process(Func,ALUop)
    begin
        if ALUop = "00" then ALUcon <= "0010";
            elsif ALUop = "01" then ALUcon <= "0110";
            elsif ALUop = "10" then
                if Func = "100000" then ALUcon <= "0010";
                elsif Func = "100010" then ALUcon <= "0110";
                elsif Func = "100100" then ALUcon <= "0000";
                elsif Func = "100101" then ALUcon <= "0001";
                elsif Func = "101010" then ALUcon <= "0111";
            end if;
        end if;
    end process;
end rtl;
```



```
/alucontrol/Func    100000   100000
/alucontrol/ALUop   10       10
/alucontrol/ALUcon  0010     0010
```

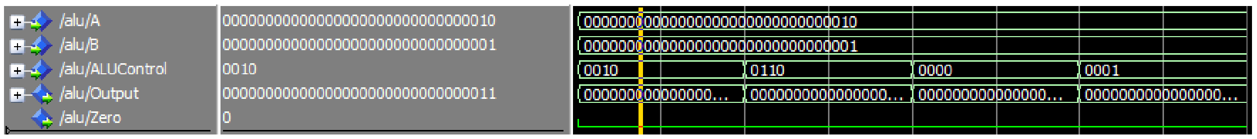
```
force -freeze sim:/alucontrol/Func 100000 0
force -freeze sim:/alucontrol/ALUop 10 0
run
```

10. ALU



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
    Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
           B : in  STD_LOGIC_VECTOR (31 downto 0);
           ALUControl : in  STD_LOGIC_VECTOR (3 downto 0);
           Output : out  STD_LOGIC_VECTOR (31 downto 0);
           Zero : out  STD_LOGIC);
end ALU;

architecture rtl of ALU is
begin
    process(A, B, ALUControl)
    begin
        if (ALUControl = "0000") then
        Output <= A AND B;
        elsif (ALUControl = "0001") then
        Output <= A OR B;
        elsif (ALUControl = "0010") then
        Output <= A + B;
        elsif (ALUControl = "0110") then
        Output <= A - B;
        elsif (ALUControl = "0111") then
        if (A < B) then
        Output <= X"00000001";
        else
        Output <= X"00000000";
        end if;
        elsif (ALUControl = "1100") then
        Output <= A NOR B;
        end if;

        if (A=B) then
        Zero <= '1';
        else
        Zero <= '0';
        end if;
    end process;

end rtl;
```

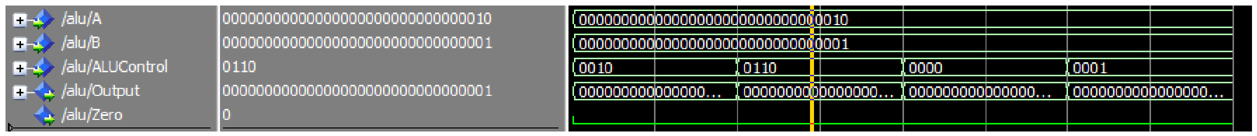
Add



```
/alu/A           00000000000000000000000000000010
/alu/B           00000000000000000000000000000001
/alu/ALUControl  0010
/alu/Output      00000000000000000000000000000011
/alu/Zero        0
```

```
00000000000000000000000000000010
00000000000000000000000000000001
0010              0110              0000              0001
0000000000000000... 0000000000000000... 000000000000000... 0000000000000000...
```

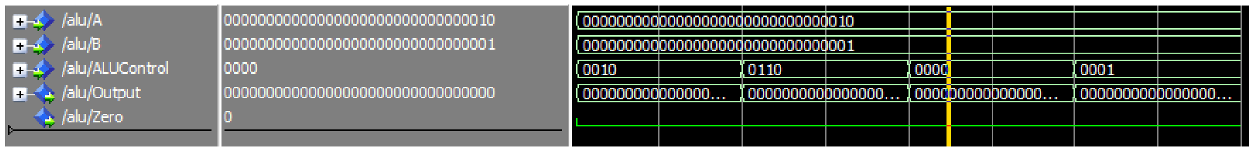
Subtract



```
/alu/A           00000000000000000000000000000010
/alu/B           00000000000000000000000000000001
/alu/ALUControl  0110
/alu/Output      00000000000000000000000000000001
/alu/Zero        0
```

```
00000000000000000000000000000010
00000000000000000000000000000001
0010              0110              0000              0001
0000000000000000... 0000000000000000... 000000000000000... 0000000000000000...
```

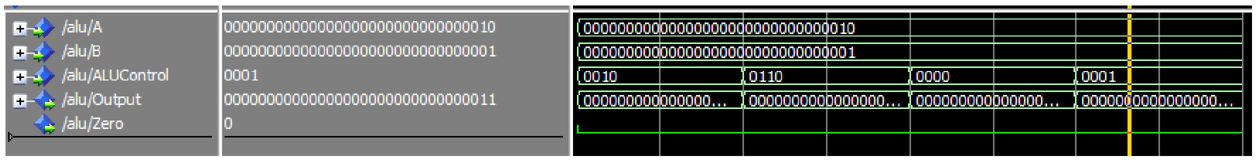
AND



```
/alu/A           00000000000000000000000000000010
/alu/B           00000000000000000000000000000001
/alu/ALUControl  0000
/alu/Output      00000000000000000000000000000000
/alu/Zero        0
```

```
00000000000000000000000000000010
00000000000000000000000000000001
0010              0110              0000              0001
0000000000000000... 0000000000000000... 000000000000000... 0000000000000000...
```

OR

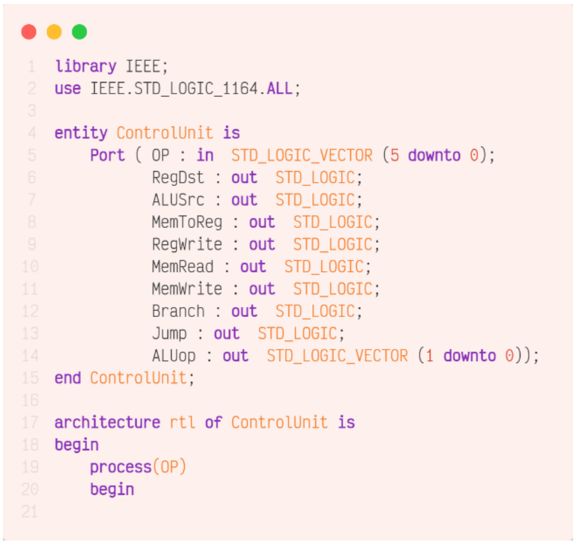


```
/alu/A           00000000000000000000000000000010
/alu/B           00000000000000000000000000000001
/alu/ALUControl  0001
/alu/Output      00000000000000000000000000000011
/alu/Zero        0
```

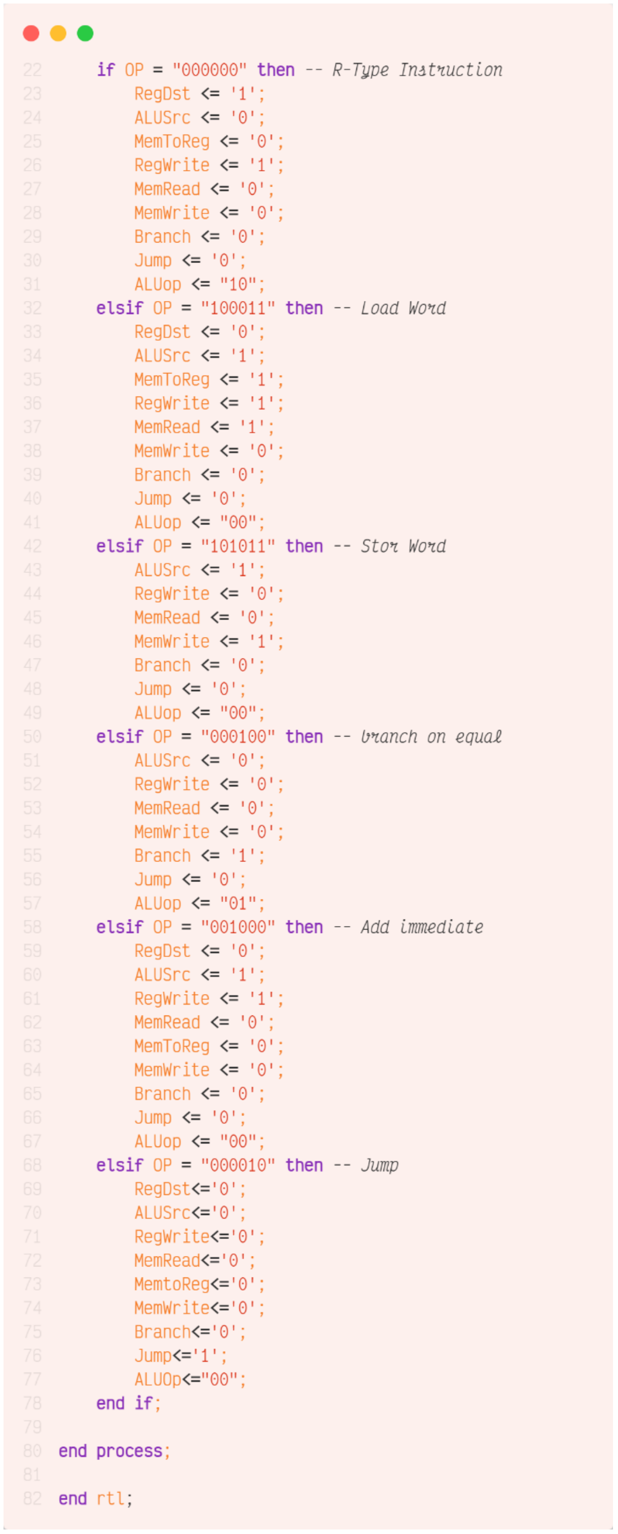
```
00000000000000000000000000000010
00000000000000000000000000000001
0010              0110              0000              0001
0000000000000000... 0000000000000000... 000000000000000... 0000000000000000...
```

```
force -freeze sim:/alu/A 16#00000002 0
force -freeze sim:/alu/B 16#00000001 0
force -freeze sim:/alu/ALUControl 0010 0
run
force -freeze sim:/alu/ALUControl 0110 0
run
force -freeze sim:/alu/ALUControl 0000 0
run
force -freeze sim:/alu/ALUControl 0001 0
run
```

11. Control Unit

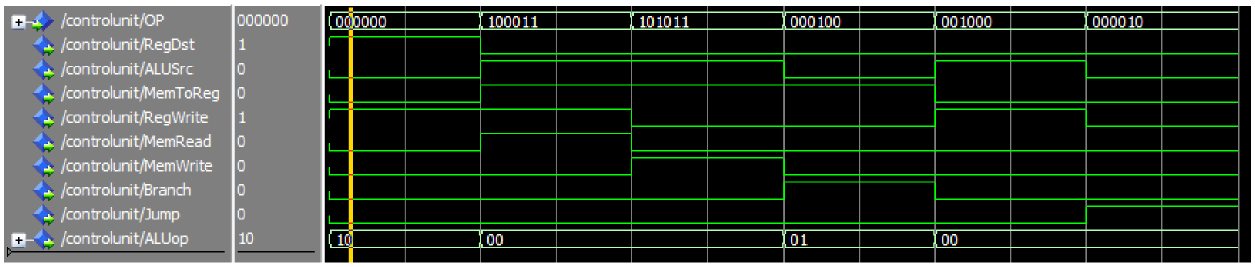


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ControlUnit is
    Port ( OP : in  STD_LOGIC_VECTOR (5 downto 0);
           RegDst : out  STD_LOGIC;
           ALUSrc : out  STD_LOGIC;
           MemToReg : out  STD_LOGIC;
           RegWrite : out  STD_LOGIC;
           MemRead : out  STD_LOGIC;
           MemWrite : out  STD_LOGIC;
           Branch : out  STD_LOGIC;
           Jump : out  STD_LOGIC;
           ALUop : out  STD_LOGIC_VECTOR (1 downto 0));
end ControlUnit;

architecture rtl of ControlUnit is
begin
    process(OP)
    begin

```



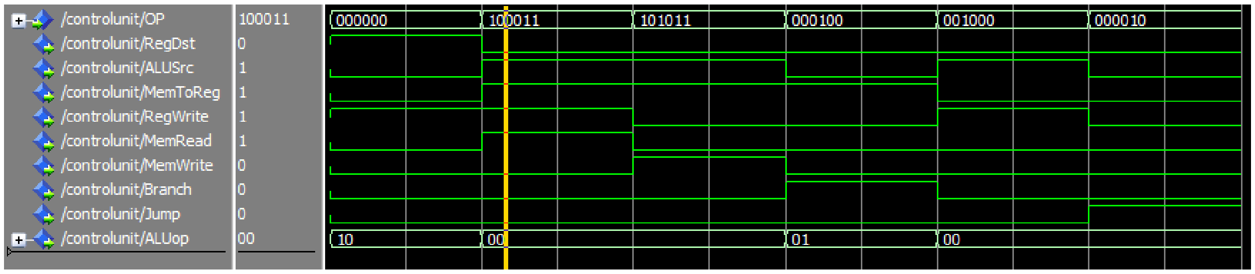
```
        if OP = "000000" then -- R-Type Instruction
            RegDst <= '1';
            ALUSrc <= '0';
            MemToReg <= '0';
            RegWrite <= '1';
            MemRead <= '0';
            MemWrite <= '0';
            Branch <= '0';
            Jump <= '0';
            ALUop <= "10";
        elsif OP = "100011" then -- Load Word
            RegDst <= '0';
            ALUSrc <= '1';
            MemToReg <= '1';
            RegWrite <= '1';
            MemRead <= '1';
            MemWrite <= '0';
            Branch <= '0';
            Jump <= '0';
            ALUop <= "00";
        elsif OP = "101011" then -- Stor Word
            ALUSrc <= '1';
            RegWrite <= '0';
            MemRead <= '0';
            MemWrite <= '1';
            Branch <= '0';
            Jump <= '0';
            ALUop <= "00";
        elsif OP = "000100" then -- branch on equal
            ALUSrc <= '0';
            RegWrite <= '0';
            MemRead <= '0';
            MemWrite <= '0';
            Branch <= '1';
            Jump <= '0';
            ALUop <= "01";
        elsif OP = "001000" then -- Add immediate
            RegDst <= '0';
            ALUSrc <= '1';
            RegWrite <= '1';
            MemRead <= '0';
            MemToReg <= '0';
            MemWrite <= '0';
            Branch <= '0';
            Jump <= '0';
            ALUop <= "00";
        elsif OP = "000010" then -- Jump
            RegDst<='0';
            ALUSrc<='0';
            RegWrite<='0';
            MemRead<='0';
            MemtoReg<='0';
            MemWrite<='0';
            Branch<='0';
            Jump<='1';
            ALUOp<="00";
        end if;

    end process;

    end rtl;
```

R-Type



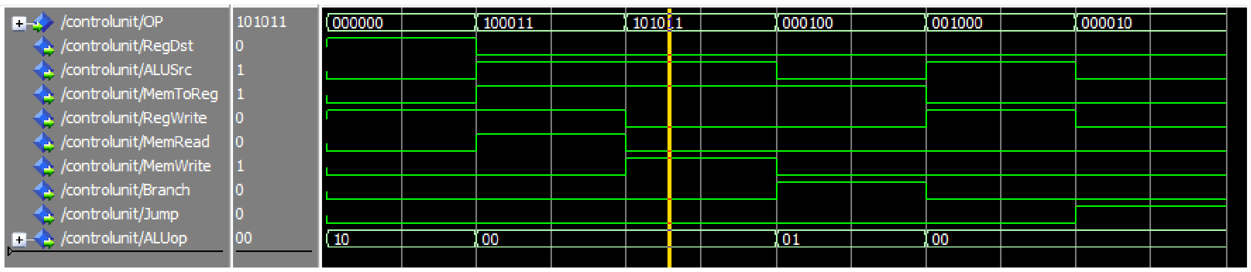
```
/controlunit/OP         000000
/controlunit/RegDst     1
/controlunit/ALUSrc     0
/controlunit/MemToReg   0
/controlunit/RegWrite   1
/controlunit/MemRead    0
/controlunit/MemWrite   0
/controlunit/Branch     0
/controlunit/Jump       0
/controlunit/ALUop      10
```

Load Word



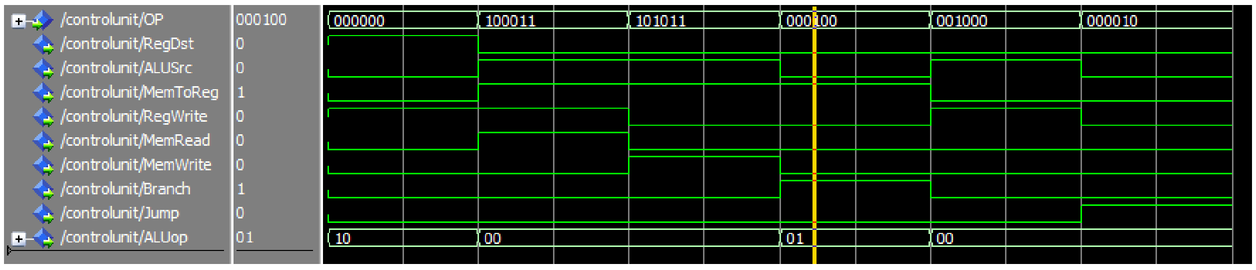
```
/controlunit/OP         100011
/controlunit/RegDst     0
/controlunit/ALUSrc     1
/controlunit/MemToReg   1
/controlunit/RegWrite   1
/controlunit/MemRead    1
/controlunit/MemWrite   0
/controlunit/Branch     0
/controlunit/Jump       0
/controlunit/ALUop      00
```

Store Word



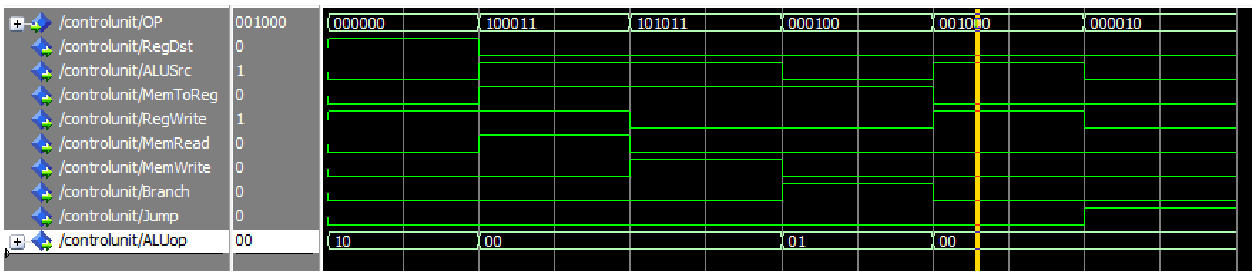
```
/controlunit/OP         101011
/controlunit/RegDst     0
/controlunit/ALUSrc     1
/controlunit/MemToReg   1
/controlunit/RegWrite   0
/controlunit/MemRead    0
/controlunit/MemWrite   1
/controlunit/Branch     0
/controlunit/Jump       0
/controlunit/ALUop      00
```

Branch on Equal

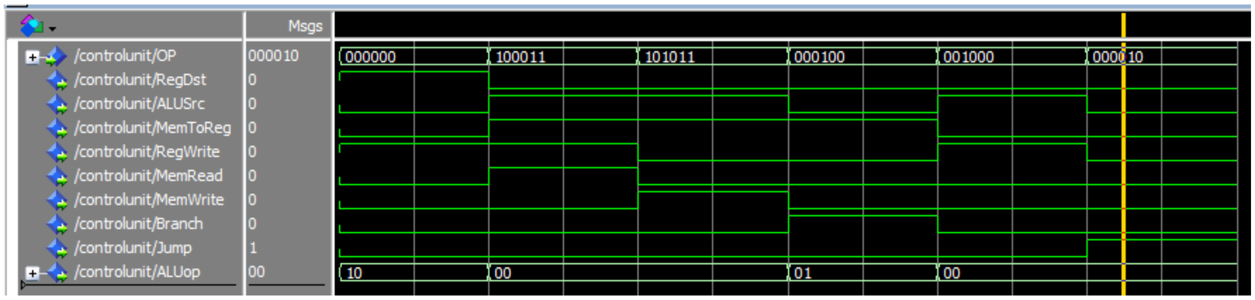


```
/controlunit/OP         000100
/controlunit/RegDst     0
/controlunit/ALUSrc     0
/controlunit/MemToReg   1
/controlunit/RegWrite   0
/controlunit/MemRead    0
/controlunit/MemWrite   0
/controlunit/Branch     1
/controlunit/Jump       0
/controlunit/ALUop      01
```

Add Immediate

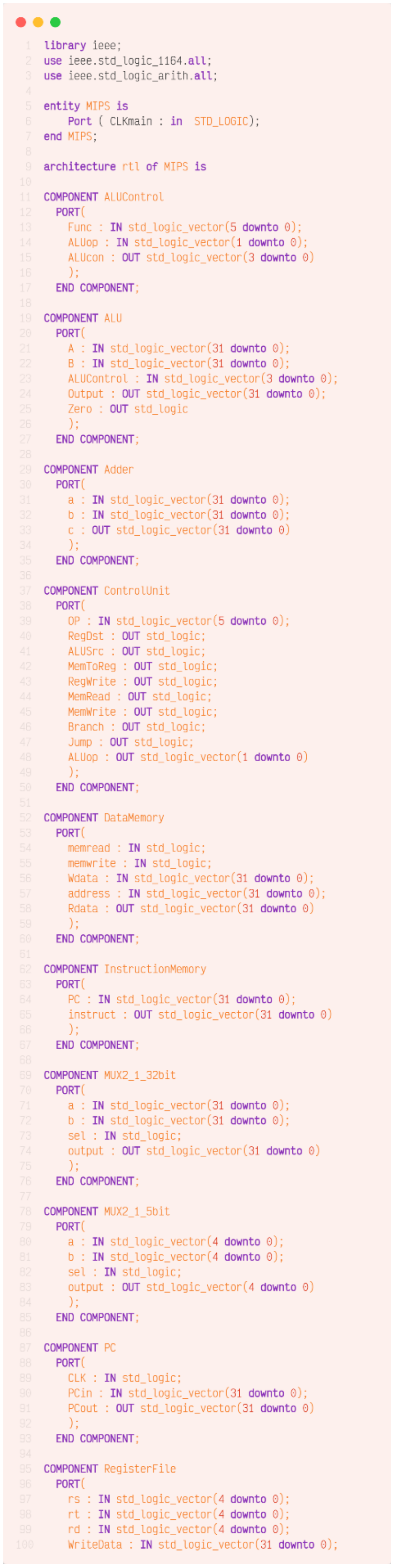


Jump

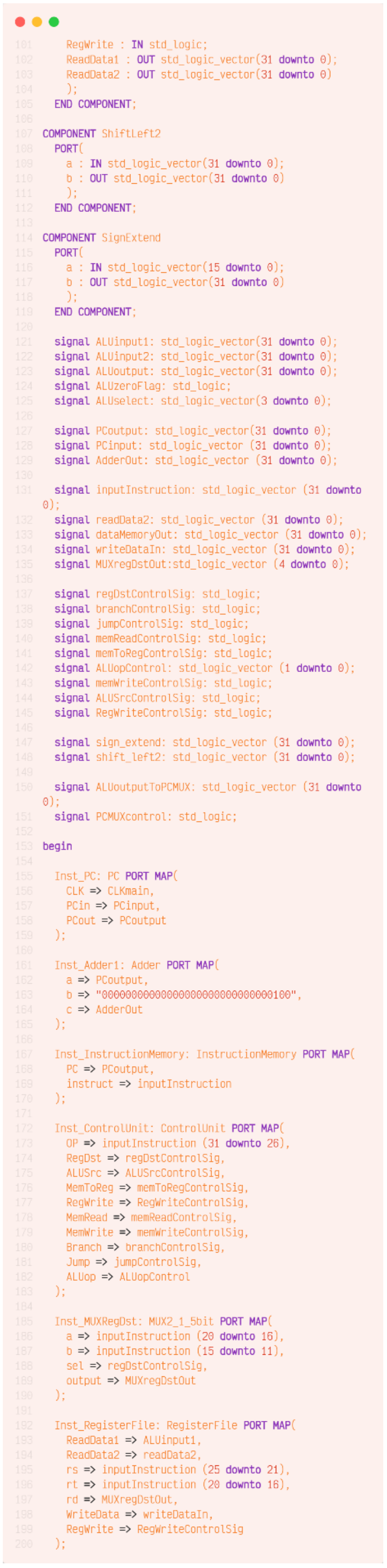


```
force -freeze sim:/controlunit/OP 000000 0
run
force -freeze sim:/controlunit/OP 100011 0
run
force -freeze sim:/controlunit/OP 101011 0
run
force -freeze sim:/controlunit/OP 000100 0
run
force -freeze sim:/controlunit/OP 001000 0
run
force -freeze sim:/controlunit/OP 000010 0
run
```

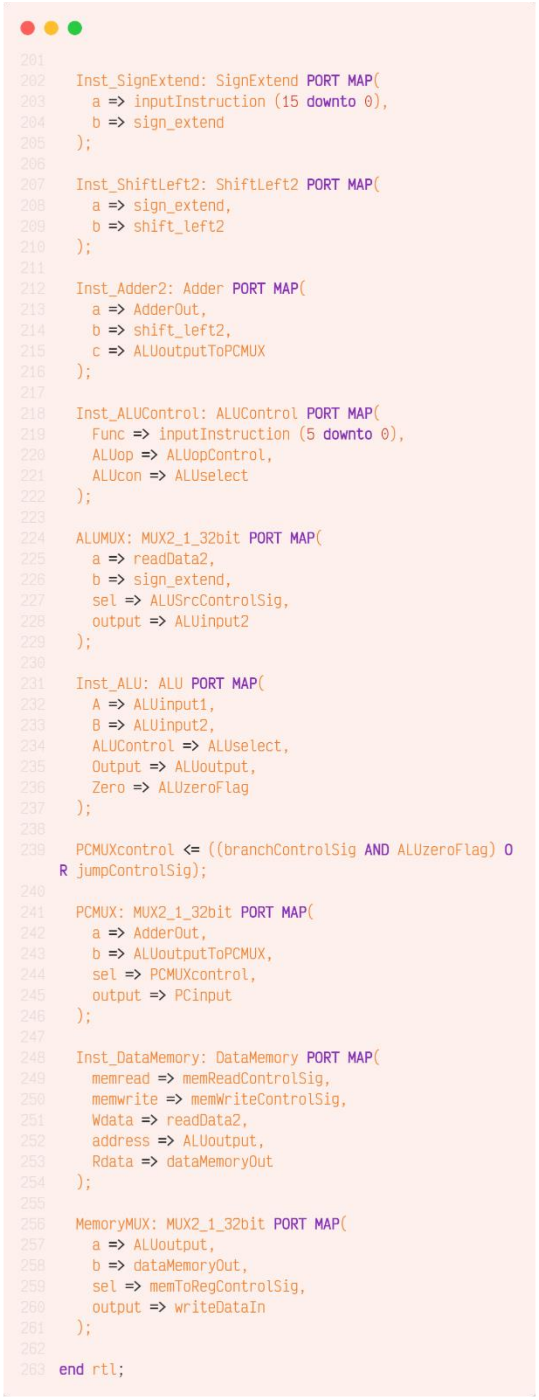
III. VHDL Coding & Simulation of MIPS CPU



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

entity MIPS is
    Port ( CLKmain : in  STD_LOGIC);
end MIPS;

architecture rtl of MIPS is

COMPONENT ALUControl
  PORT(
    Func : IN std_logic_vector(5 downto 0);
    ALUop : IN std_logic_vector(1 downto 0);
    ALUcon : OUT std_logic_vector(3 downto 0)
    );
  END COMPONENT;

COMPONENT ALU
  PORT(
    A : IN std_logic_vector(31 downto 0);
    B : IN std_logic_vector(31 downto 0);
    ALUControl : IN std_logic_vector(3 downto 0);
    Output : OUT std_logic_vector(31 downto 0);
    Zero : OUT std_logic
    );
  END COMPONENT;

COMPONENT Adder
  PORT(
    a : IN std_logic_vector(31 downto 0);
    b : IN std_logic_vector(31 downto 0);
    c : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;

COMPONENT ControlUnit
  PORT(
    OP : IN std_logic_vector(5 downto 0);
    RegDst : OUT std_logic;
    ALUSrc : OUT std_logic;
    MemToReg : OUT std_logic;
    RegWrite : OUT std_logic;
    MemRead : OUT std_logic;
    MemWrite : OUT std_logic;
    Branch : OUT std_logic;
    Jump : OUT std_logic;
    ALUop : OUT std_logic_vector(1 downto 0)
    );
  END COMPONENT;

COMPONENT DataMemory
  PORT(
    memread : IN std_logic;
    memwrite : IN std_logic;
    Wdata : IN std_logic_vector(31 downto 0);
    address : IN std_logic_vector(31 downto 0);
    Rdata : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;

COMPONENT InstructionMemory
  PORT(
    PC : IN std_logic_vector(31 downto 0);
    instruct : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;

COMPONENT MUX2_1_32bit
  PORT(
    a : IN std_logic_vector(31 downto 0);
    b : IN std_logic_vector(31 downto 0);
    sel : IN std_logic;
    output : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;

COMPONENT MUX2_1_5bit
  PORT(
    a : IN std_logic_vector(4 downto 0);
    b : IN std_logic_vector(4 downto 0);
    sel : IN std_logic;
    output : OUT std_logic_vector(4 downto 0)
    );
  END COMPONENT;

COMPONENT PC
  PORT(
    CLK : IN std_logic;
    PCin : IN std_logic_vector(31 downto 0);
    PCout : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;

COMPONENT RegisterFile
  PORT(
    rs : IN std_logic_vector(4 downto 0);
    rt : IN std_logic_vector(4 downto 0);
    rd : IN std_logic_vector(4 downto 0);
    WriteData : IN std_logic_vector(31 downto 0);
```



```vhdl
    RegWrite : IN std_logic;
    ReadData1 : OUT std_logic_vector(31 downto 0);
    ReadData2 : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;

COMPONENT ShiftLeft2
  PORT(
    a : IN std_logic_vector(31 downto 0);
    b : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;

COMPONENT SignExtend
  PORT(
    a : IN std_logic_vector(15 downto 0);
    b : OUT std_logic_vector(31 downto 0)
    );
  END COMPONENT;

  signal ALUinput1: std_logic_vector(31 downto 0);
  signal ALUinput2: std_logic_vector(31 downto 0);
  signal ALUoutput: std_logic_vector(31 downto 0);
  signal ALUzeroFlag: std_logic;
  signal ALUselect: std_logic_vector(3 downto 0);

  signal PCoutput: std_logic_vector(31 downto 0);
  signal PCinput: std_logic_vector (31 downto 0);
  signal AdderOut: std_logic_vector (31 downto 0);

  signal inputInstruction: std_logic_vector (31 downto
0);
  signal readData2: std_logic_vector (31 downto 0);
  signal dataMemoryOut: std_logic_vector (31 downto 0);
  signal writeDataIn: std_logic_vector (31 downto 0);
  signal MUXregDstOut:std_logic_vector (4 downto 0);

  signal regDstControlSig: std_logic;
  signal branchControlSig: std_logic;
  signal jumpControlSig: std_logic;
  signal memReadControlSig: std_logic;
  signal memToRegControlSig: std_logic;
  signal ALUopControl: std_logic_vector (1 downto 0);
  signal memWriteControlSig: std_logic;
  signal ALUSrcControlSig: std_logic;
  signal RegWriteControlSig: std_logic;

  signal sign_extend: std_logic_vector (31 downto 0);
  signal shift_left2: std_logic_vector (31 downto 0);

  signal ALUoutputToPCMUX: std_logic_vector (31 downto
0);
  signal PCMUXcontrol: std_logic;

begin

  Inst_PC: PC PORT MAP(
    CLK => CLKmain,
    PCin => PCinput,
    PCout => PCoutput
  );

  Inst_Adder1: Adder PORT MAP(
    a => PCoutput,
    b => "00000000000000000000000000000100",
    c => AdderOut
  );

  Inst_InstructionMemory: InstructionMemory PORT MAP(
    PC => PCoutput,
    instruct => inputInstruction
  );

  Inst_ControlUnit: ControlUnit PORT MAP(
    OP => inputInstruction (31 downto 26),
    RegDst => regDstControlSig,
    ALUSrc => ALUSrcControlSig,
    MemToReg => memToRegControlSig,
    RegWrite => RegWriteControlSig,
    MemRead => memReadControlSig,
    MemWrite => memWriteControlSig,
    Branch => branchControlSig,
    Jump => jumpControlSig,
    ALUop => ALUopControl
  );

  Inst_MUXRegDst: MUX2_1_5bit PORT MAP(
    a => inputInstruction (20 downto 16),
    b => inputInstruction (15 downto 11),
    sel => regDstControlSig,
    output => MUXregDstOut
  );

  Inst_RegisterFile: RegisterFile PORT MAP(
    ReadData1 => ALUinput1,
    ReadData2 => readData2,
    rs => inputInstruction (25 downto 21),
    rt => inputInstruction (20 downto 16),
    rd => MUXregDstOut,
    WriteData => writeDataIn,
    RegWrite => RegWriteControlSig
  );
```

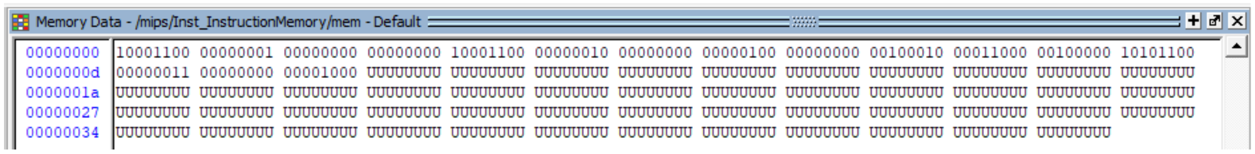


```vhdl

  Inst_SignExtend: SignExtend PORT MAP(
    a => inputInstruction (15 downto 0),
    b => sign_extend
  );

  Inst_ShiftLeft2: ShiftLeft2 PORT MAP(
    a => sign_extend,
    b => shift_left2
  );

  Inst_Adder2: Adder PORT MAP(
    a => AdderOut,
    b => shift_left2,
    c => ALUoutputToPCMUX
  );

  Inst_ALUControl: ALUControl PORT MAP(
    Func => inputInstruction (5 downto 0),
    ALUop => ALUopControl,
    ALUcon => ALUselect
  );

  ALUMUX: MUX2_1_32bit PORT MAP(
    a => readData2,
    b => sign_extend,
    sel => ALUSrcControlSig,
    output => ALUinput2
  );

  Inst_ALU: ALU PORT MAP(
    A => ALUinput1,
    B => ALUinput2,
    ALUControl => ALUselect,
    Output => ALUoutput,
    Zero => ALUzeroFlag
  );

  PCMUXcontrol <= ((branchControlSig AND ALUzeroFlag) O
R jumpControlSig);

  PCMUX: MUX2_1_32bit PORT MAP(
    a => AdderOut,
    b => ALUoutputToPCMUX,
    sel => PCMUXcontrol,
    output => PCinput
  );

  Inst_DataMemory: DataMemory PORT MAP(
    memread => memReadControlSig,
    memwrite => memWriteControlSig,
    Wdata => readData2,
    address => ALUoutput,
    Rdata => dataMemoryOut
  );

  MemoryMUX: MUX2_1_32bit PORT MAP(
    a => ALUoutput,
    b => dataMemoryOut,
    sel => memToRegControlSig,
    output => writeDataIn
  );

end rtl;
```

Test is running four instructions

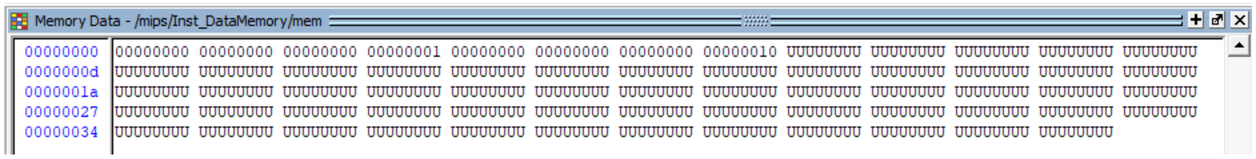
1. Load the first 32bit word from Data memory to Register 1 (value = 1)
2. Load the second 32bit word from Data memory to Register 2 (value = 2)
3. Adding Register 1 and 2 to Register 3 (value = 1 + 2 = 3)
4. Store Register 3 value in the third 32bit location in data memory

After first run



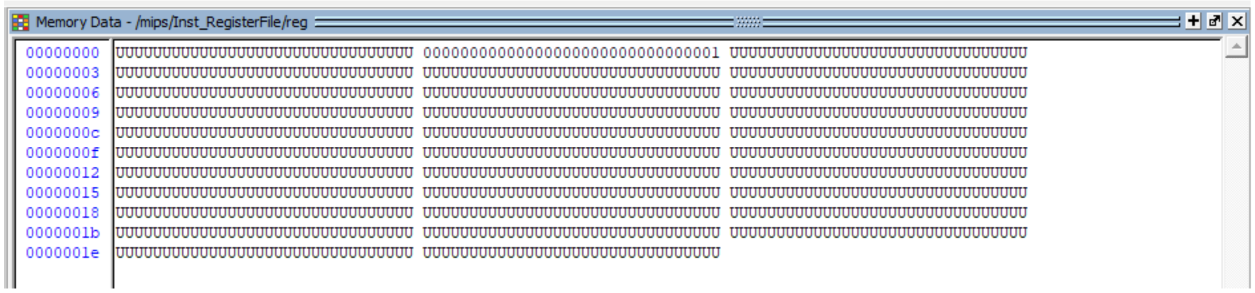
Memory Data - /mips/Inst_InstructionMemory/mem - Default

```
00000000  10001100 00000001 00000000 00000000 10001100 00000010 00000000 00000100 00000000 00100010 00011000 00100000 10101100
0000000d  00000011 00000000 00001000 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
0000001a  UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000027  UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000034  UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
```



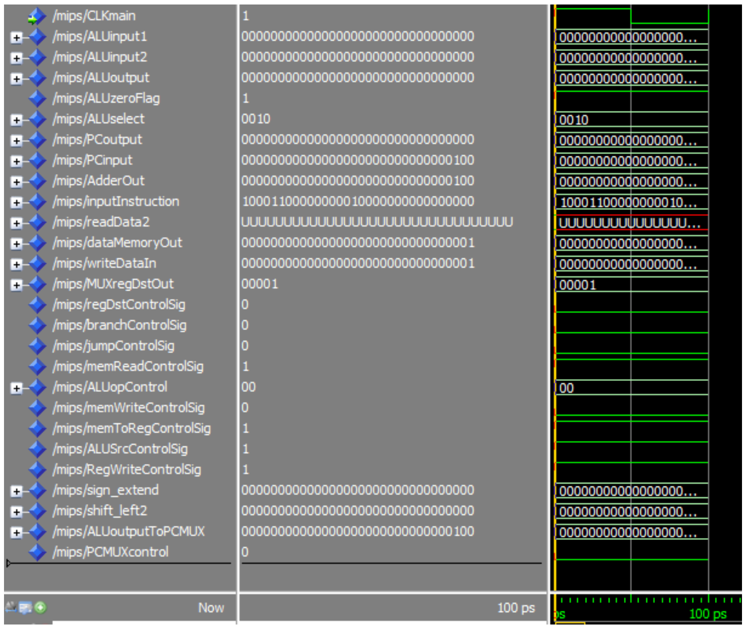
Memory Data - /mips/Inst_DataMemory/mem

```
00000000  00000000 00000000 00000000 00000001 00000000 00000000 00000000 00000010 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
0000000d  UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
0000001a  UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000027  UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000034  UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
```



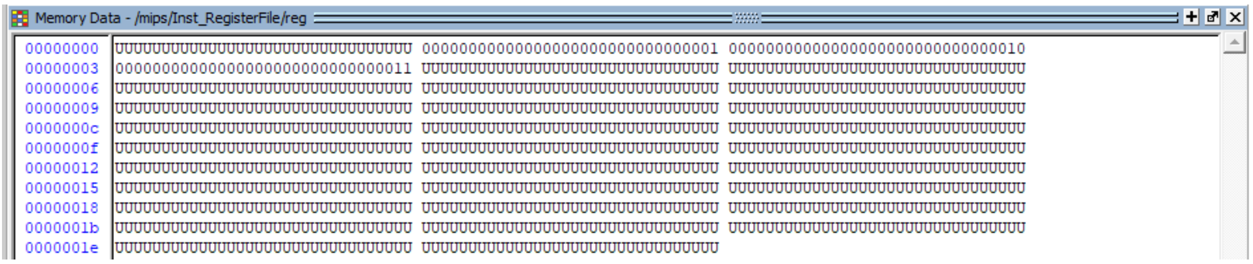
Memory Data - /mips/Inst_RegisterFile/reg

```
00000000  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU 00000000000000000000000000000001 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000003  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000006  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000009  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000000c  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000000f  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000012  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000015  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000018  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000001b  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000001e  UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```

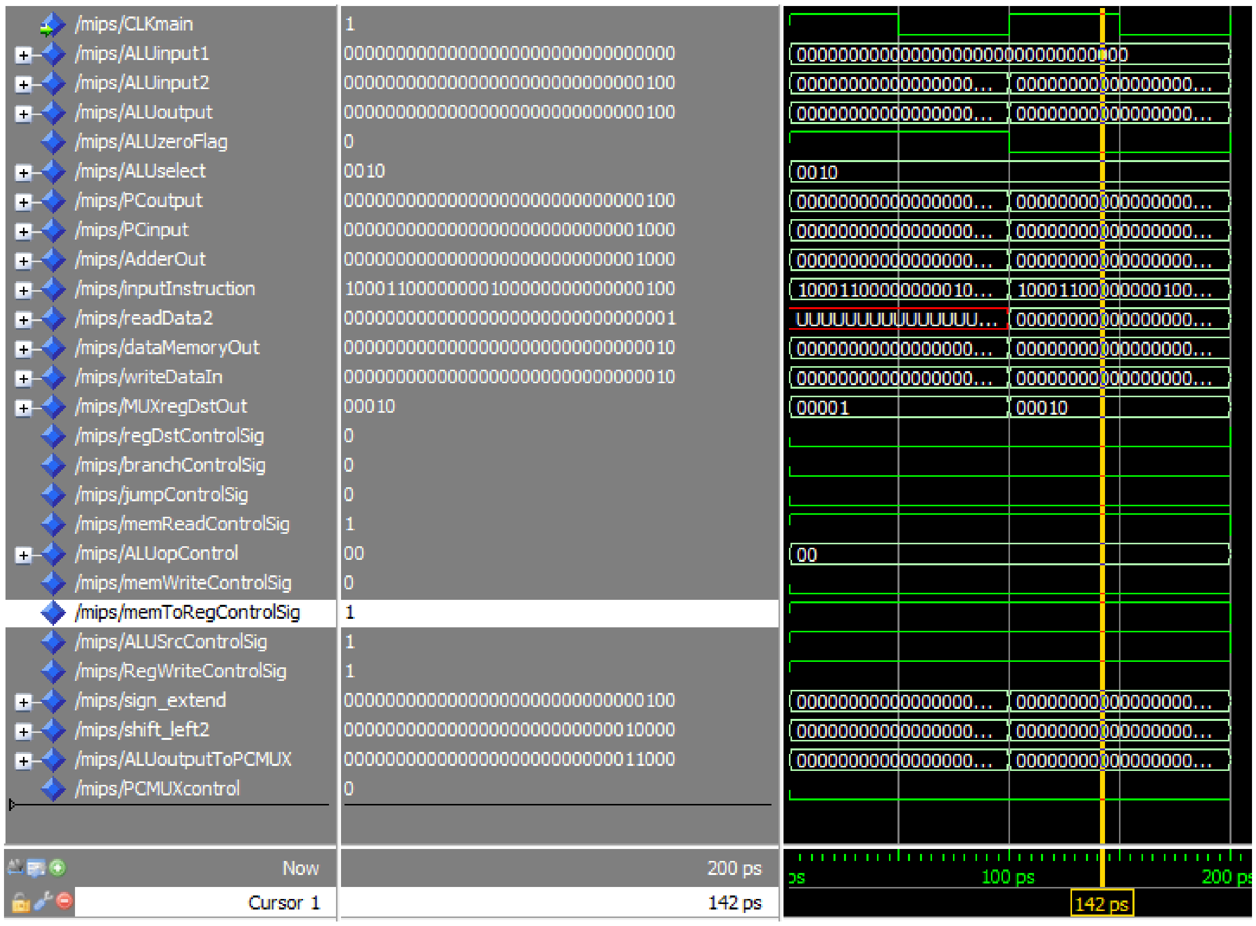


```
/mips/CLKmain                1
/mips/ALUinput1              00000000000000000000000000000000
/mips/ALUinput2              00000000000000000000000000000000
/mips/ALUoutput              00000000000000000000000000000000
/mips/ALUzeroFlag            1
/mips/ALUselect              0010
/mips/PCoutput               00000000000000000000000000000000
/mips/PCinput                00000000000000000000000000000100
/mips/AdderOut               00000000000000000000000000000100
/mips/inputInstruction       10001100000000010000000000000000
/mips/readData2              UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/mips/dataMemoryOut          00000000000000000000000000000001
/mips/writeDataIn            00000000000000000000000000000001
/mips/MUXregDstOut           00001
/mips/regDstControlSig       0
/mips/branchControlSig       0
/mips/jumpControlSig         0
/mips/memReadControlSig      1
/mips/ALUopControl           00
/mips/memWriteControlSig     0
/mips/memToRegControlSig     1
/mips/ALUSrcControlSig       1
/mips/RegWriteControlSig     1
/mips/sign_extend            00000000000000000000000000000000
/mips/shift_left2            00000000000000000000000000000000
/mips/ALUoutputToPCMUX       00000000000000000000000000000100
/mips/PCMUXcontrol           0
Now                          100 ps
```

After Second Run

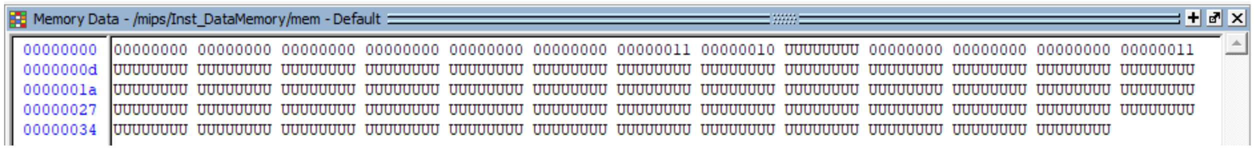


```
Memory Data - /mips/Inst_RegisterFile/reg
00000000 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU 00000000000000000000000000000001 00000000000000000000000000000010
00000003 00000000000000000000000000000011 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000006 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000009 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000000c UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000000f UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000012 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000015 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00000018 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000001b UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
0000001e UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
```



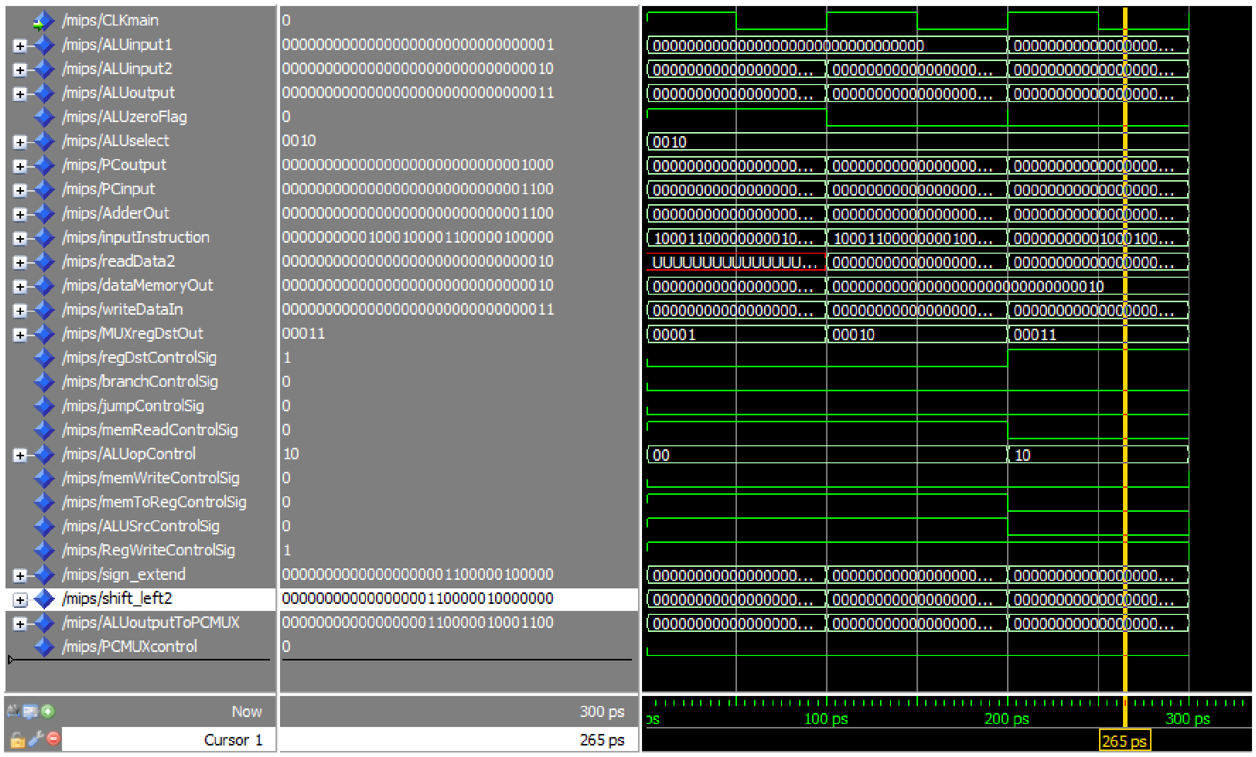
```
/mips/CLKmain             1
/mips/ALUinput1           00000000000000000000000000000000
/mips/ALUinput2           00000000000000000000000000000100
/mips/ALUoutput           00000000000000000000000000000100
/mips/ALUzeroFlag         0
/mips/ALUselect           0010
/mips/PCoutput            00000000000000000000000000000100
/mips/PCinput             00000000000000000000000000001000
/mips/AdderOut            00000000000000000000000000001000
/mips/inputInstruction    10001100000000100000000000000100
/mips/readData2           00000000000000000000000000000001
/mips/dataMemoryOut       00000000000000000000000000000010
/mips/writeDataIn         00000000000000000000000000000010
/mips/MUXregDstOut        00010
/mips/regDstControlSig    0
/mips/branchControlSig    0
/mips/jumpControlSig      0
/mips/memReadControlSig   1
/mips/ALUopControl        00
/mips/memWriteControlSig  0
/mips/memToRegControlSig  1
/mips/ALUSrcControlSig    1
/mips/RegWriteControlSig  1
/mips/sign_extend         00000000000000000000000000000100
/mips/shift_left2         00000000000000000000000000010000
/mips/ALUoutputToPCMUX    00000000000000000000000000011000
/mips/PCMUXcontrol        0

Now       200 ps
Cursor 1  142 ps
```

After Third Run



Memory Data - /mips/Inst_DataMemory/mem - Default

```
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000011 00000010 UUUUUUUU 00000000 00000000 00000000 00000011
0000000d UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
0000001a UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000027 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
00000034 UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU UUUUUUUU
```



```
/mips/CLKmain                0
/mips/ALUinput1              00000000000000000000000000000001
/mips/ALUinput2              00000000000000000000000000000010
/mips/ALUoutput              00000000000000000000000000000011
/mips/ALUzeroFlag            0
/mips/ALUselect              0010
/mips/PCoutput               00000000000000000000000000001000
/mips/PCinput                00000000000000000000000000001100
/mips/AdderOut               00000000000000000000000000001100
/mips/inputInstruction       00000000001000100001100000100000
/mips/readData2              00000000000000000000000000000010
/mips/dataMemoryOut          00000000000000000000000000000010
/mips/writeDataIn            00000000000000000000000000000011
/mips/MUXregDstOut           00011
/mips/regDstControlSig       1
/mips/branchControlSig       0
/mips/jumpControlSig         0
/mips/memReadControlSig      0
/mips/ALUopControl           10
/mips/memWriteControlSig     0
/mips/memToRegControlSig     0
/mips/ALUSrcControlSig       0
/mips/RegWriteControlSig     1
/mips/sign_extend            00000000000000000001100000100000
/mips/shift_left2            00000000000000000110000010000000
/mips/ALUoutputToPCMUX       00000000000000000110000010001100
/mips/PCMUXcontrol           0

Now        300 ps
Cursor 1   265 ps
```

```
force -freeze sim:/mips/CLKmain 1 0, 0 {50 ps} -r 100
force -freeze sim:/mips/PCoutput 16#00000000 0 -cancel 100ps
mem load -filltype value -filldata 16#00 -fillradix symbolic /mips/Inst_DataMemory/mem(0)
mem load -filltype value -filldata 16#00 -fillradix symbolic /mips/Inst_DataMemory/mem(1)
mem load -filltype value -filldata 16#00 -fillradix symbolic /mips/Inst_DataMemory/mem(2)
mem load -filltype value -filldata 16#01 -fillradix symbolic /mips/Inst_DataMemory/mem(3)
mem load -filltype value -filldata 16#00 -fillradix symbolic /mips/Inst_DataMemory/mem(4)
mem load -filltype value -filldata 16#00 -fillradix symbolic /mips/Inst_DataMemory/mem(5)
mem load -filltype value -filldata 16#00 -fillradix symbolic /mips/Inst_DataMemory/mem(6)
mem load -filltype value -filldata 16#02 -fillradix symbolic /mips/Inst_DataMemory/mem(7)
run
run
run
```