Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Computer Organization and Architecture
Course Number:	COE628
Semester/Year (e.g.F2016)	W2024

Instructor:	Khalid Abdel Hafeez

Assignment/Lab Number:	1
Assignment/Lab Title:	Program Counter and Register Set Design

Submission Date:	2024/01/26
Due Date:	2024/02/01

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Sarim	Shahwar	501109286	12	SS

Lab Objective:

The primary objective of this lab is to design, implement, simulate, and test two types of registers using VHDL and the Waveform generator. These registers are fundamental components in the architecture of a 32-bit CPU. The two types of registers that were created are the following:

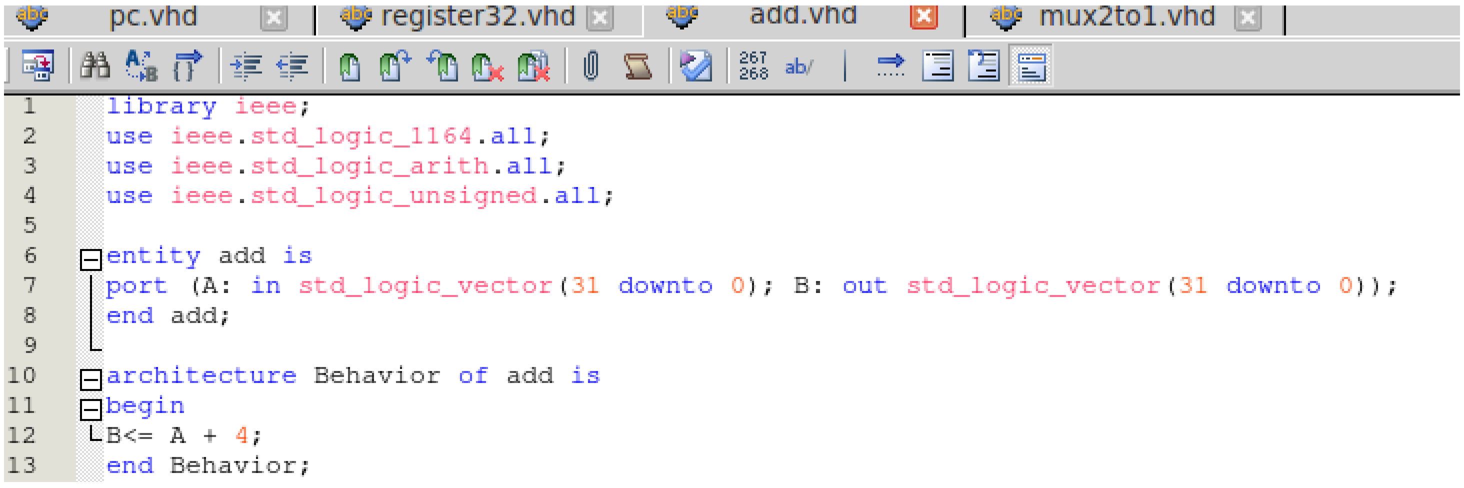
1. A 1-bit Register: This register is the most basic form of storage in digital circuits. It holds a single bit of data, which can be either 0 or 1.
2. A 32-bit Register: This register is capable of holding 32 bits of data, making it suitable for use in a 32-bit CPU architecture. The 32-bit register is crucial for operations that involve larger data types, such as integers and addresses.

Both registers have the following inputs and outputs:

- Clock (clk): A signal that synchronizes data transfer and operations within the register.
- Clear (clr): A signal used to reset or clear the register's contents.
- Load/Enable (ld): A control signal that determines when the register should load (or accept) the input data.
- Data Input (d): Represents the n-bit data to be stored in the register, where n is 1 for the 1-bit register and 32 for the 32-bit register.
- Data Output (q): The n-bit output that reflects the data currently stored in the register.

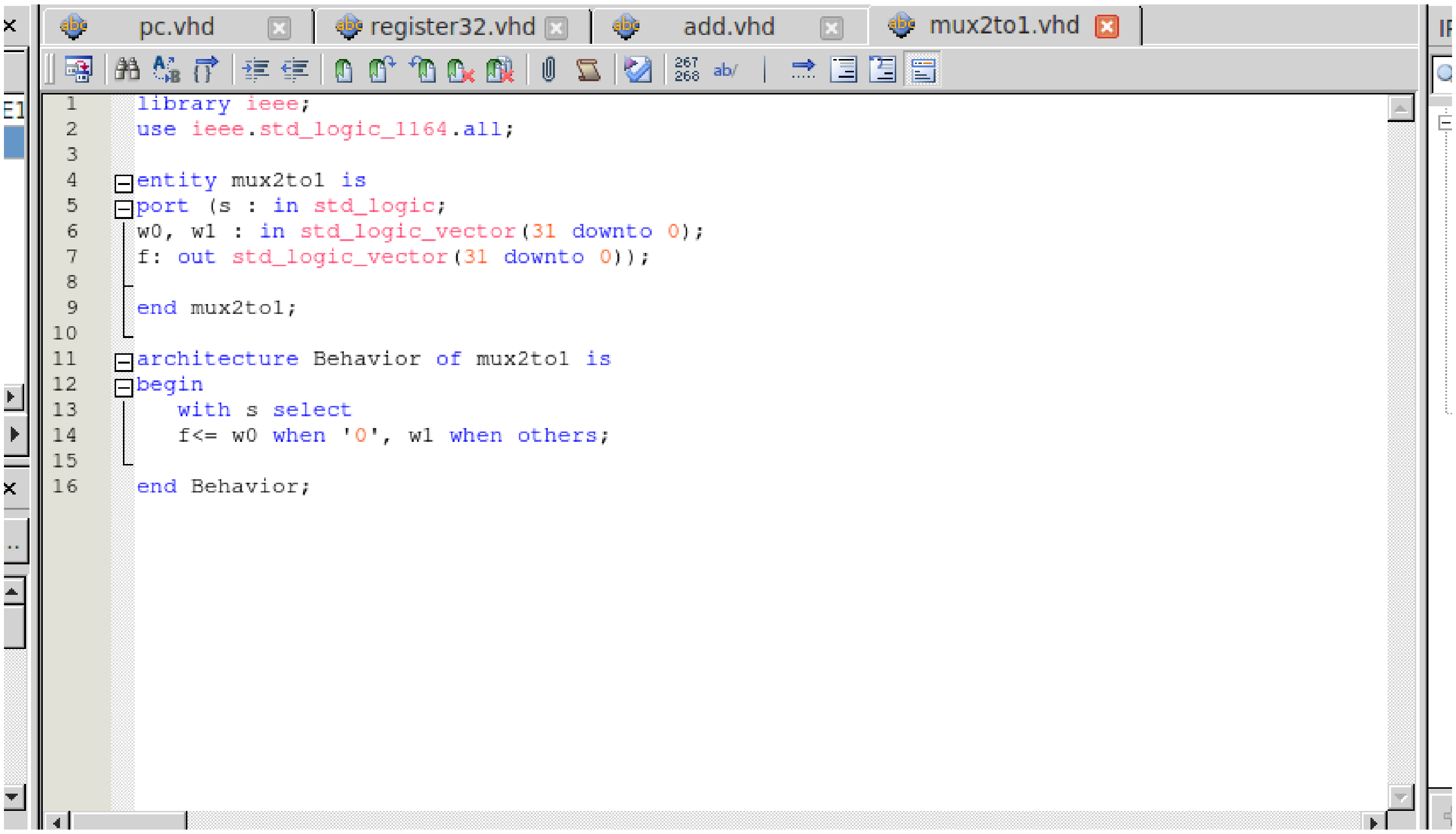
Experiment Details:

Add: This is the add function VHDL Code. This code takes a 32-bit input, adds the constant value 4 to it, and outputs the result on a 32-bit output port.



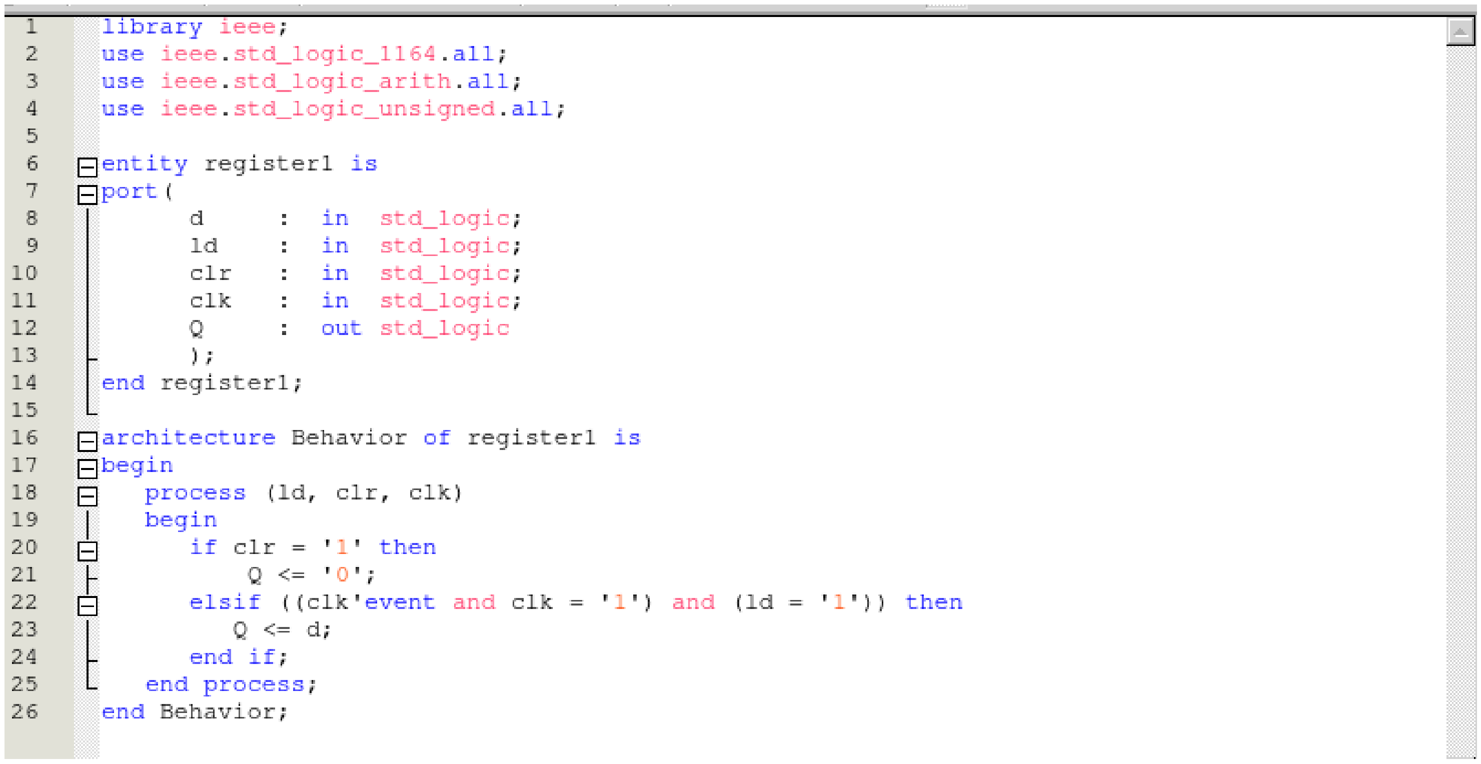
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

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

architecture Behavior of add is
begin
B<= A + 4;
end Behavior;
```

Mux2to1: This is a simple digital circuit is designed to choose between two 32-bit data inputs, referred to as w0 and w1, using a single-bit selector labeled as s. When the selector s is in the '0' state, the output, denoted as f, is assigned the value of w0. Conversely, if the selector s is in the '1' state (or any state other than '0'), the output f takes the value of w1.



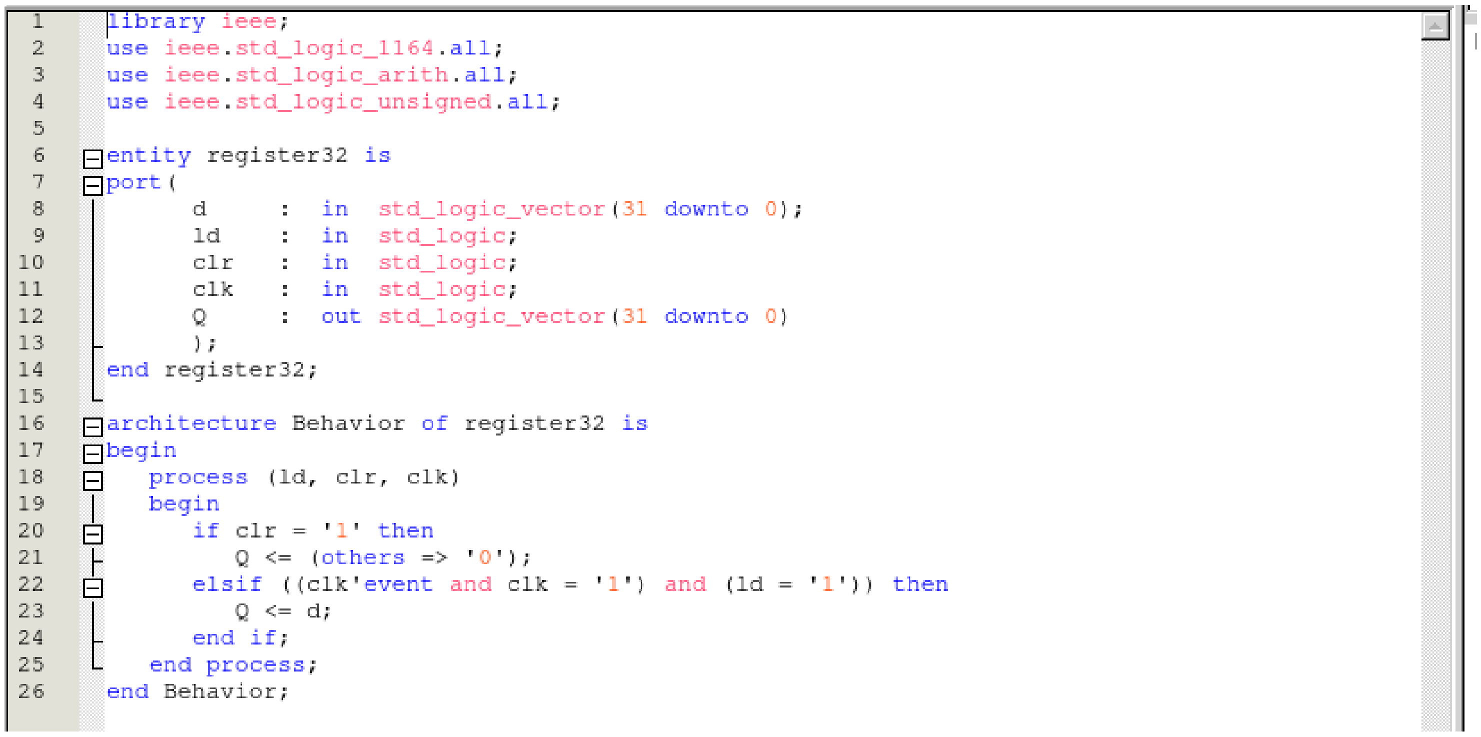
```
library ieee;
use ieee.std_logic_1164.all;

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

end mux2to1;

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

end Behavior;
```

Register 1:
This code is a procedure that reacts to alterations in the ld, clr, and clk signals. Within this procedure:
- When clr equals '1', the register's output, Q, is reset to '0', thereby clearing the register.
- Upon observing a rising edge on the clk signal (as shown by clk'event and clk being '1') and with the load signal ld set to '1', the data input d is transferred into the register, with its value then reflected in the output Q.
- If neither condition is satisfied, the output Q maintains its last state.



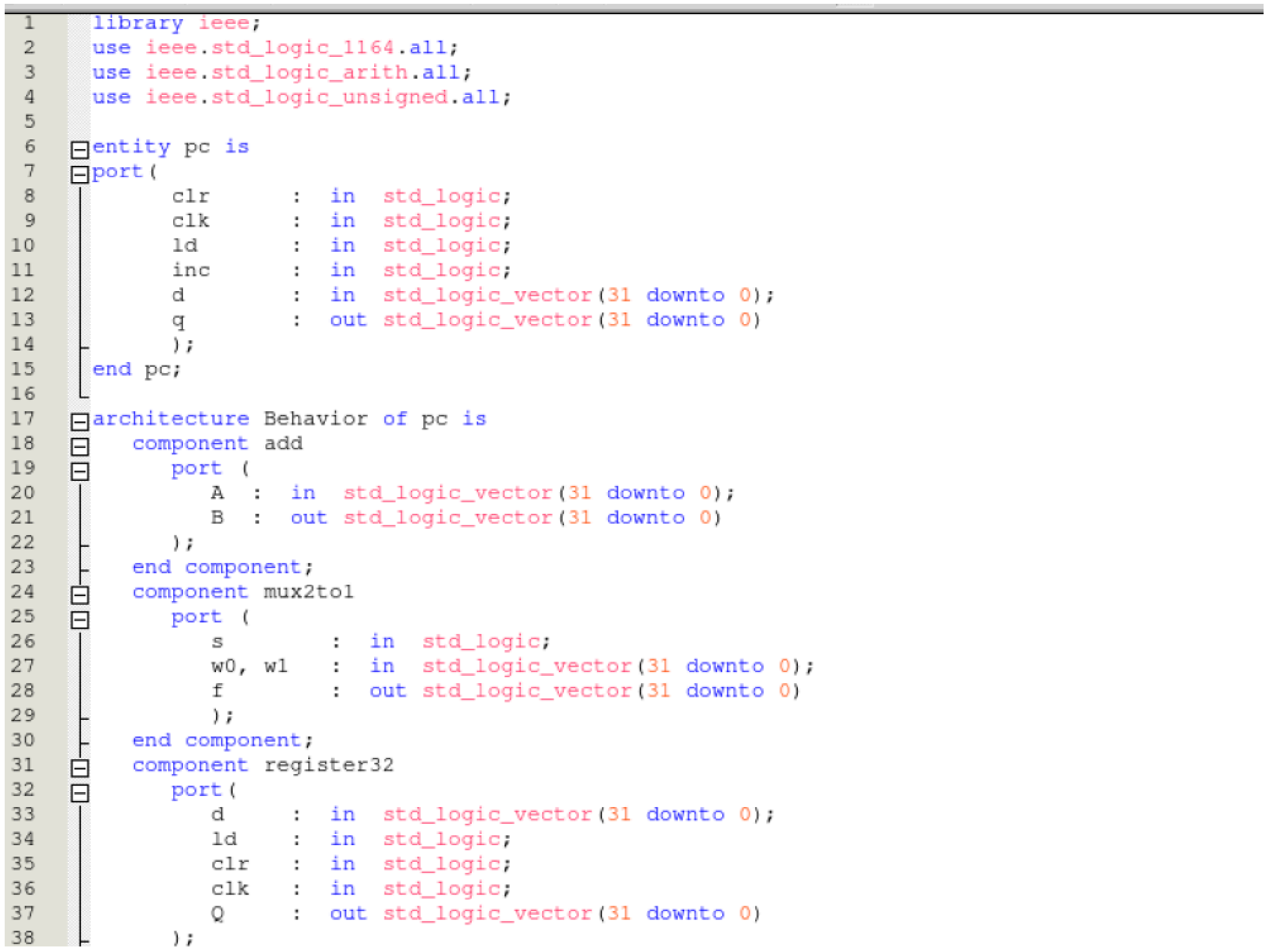
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity register1 is
port(
        d       :  in   std_logic;
        ld      :  in   std_logic;
        clr     :  in   std_logic;
        clk     :  in   std_logic;
        Q       :  out  std_logic
        );
end register1;

architecture Behavior of register1 is
begin
   process (ld, clr, clk)
   begin
      if clr = '1' then
         Q <= '0';
      elsif ((clk'event and clk = '1') and (ld = '1')) then
         Q <= d;
      end if;
   end process;
end Behavior;
```

Register 32: This code describes a 32-bit register with load and clear functionality. The register updates its 32-bit output Q either on the rising edge of the clock signal (if the load signal ld is active) or when the clear signal clr is activated

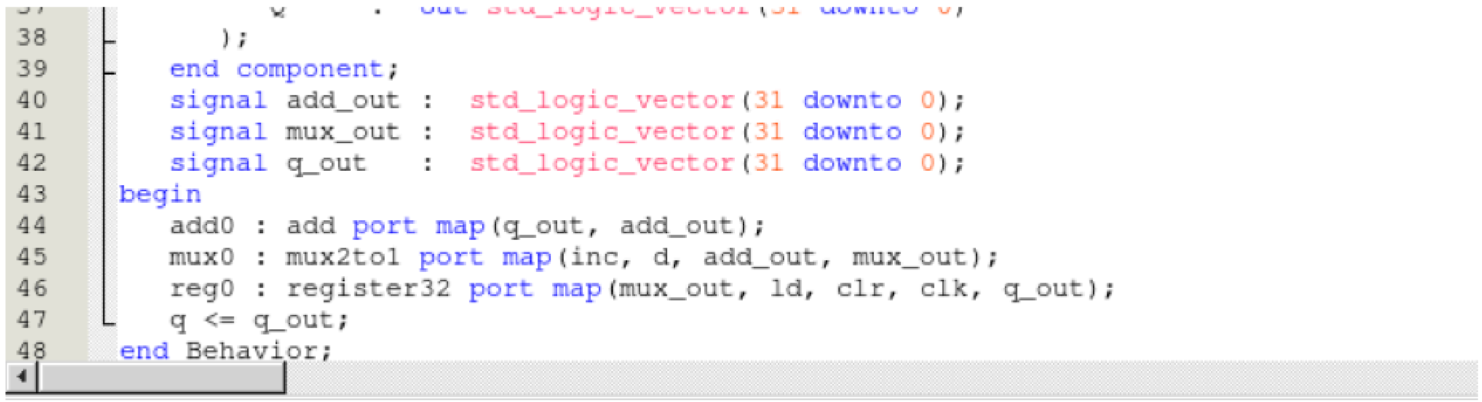


```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

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

architecture Behavior of register32 is
begin
   process (ld, clr, clk)
   begin
      if clr = '1' then
         Q <= (others => '0');
      elsif ((clk'event and clk = '1') and (ld = '1')) then
         Q <= d;
      end if;
   end process;
end Behavior;
```

PC: This code describes a program counter with the capability to either increment its current value (by a fixed amount, which is 4, to move to the next instruction address in a sequence) or load a new value based on external signals. The inc signal determines whether the PC increments its value or loads a new address from d. The clr and ld signals control clearing and loading the register, respectively, while clk provides synchronization. The q output reflects the current value of the program counter.



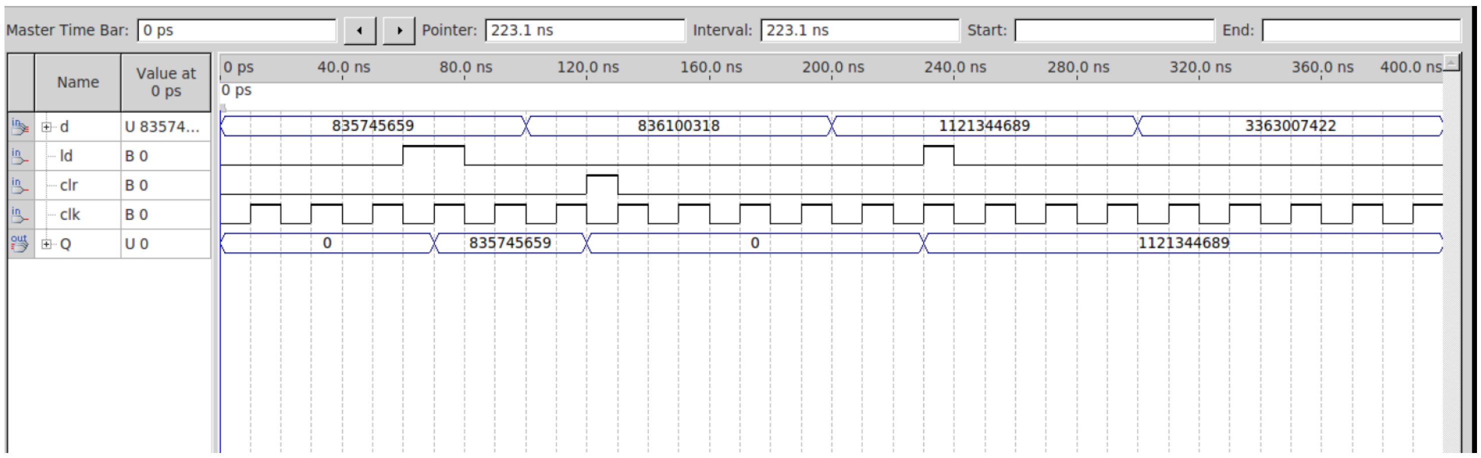
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

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

architecture Behavior of pc is
   component add
      port  (
         A  :  in   std_logic_vector(31 downto 0);
         B  :  out  std_logic_vector(31 downto 0)
      );
   end component;
   component mux2to1
      port  (
         s         :  in  std_logic;
         w0, w1    :  in  std_logic_vector(31 downto 0);
         f         :  out std_logic_vector(31 downto 0)
         );
   end component;
   component register32
      port(
         d     :  in  std_logic_vector(31 downto 0);
         ld    :  in  std_logic;
         clr   :  in  std_logic;
         clk   :  in  std_logic;
         Q     :  out std_logic_vector(31 downto 0)
      );
```



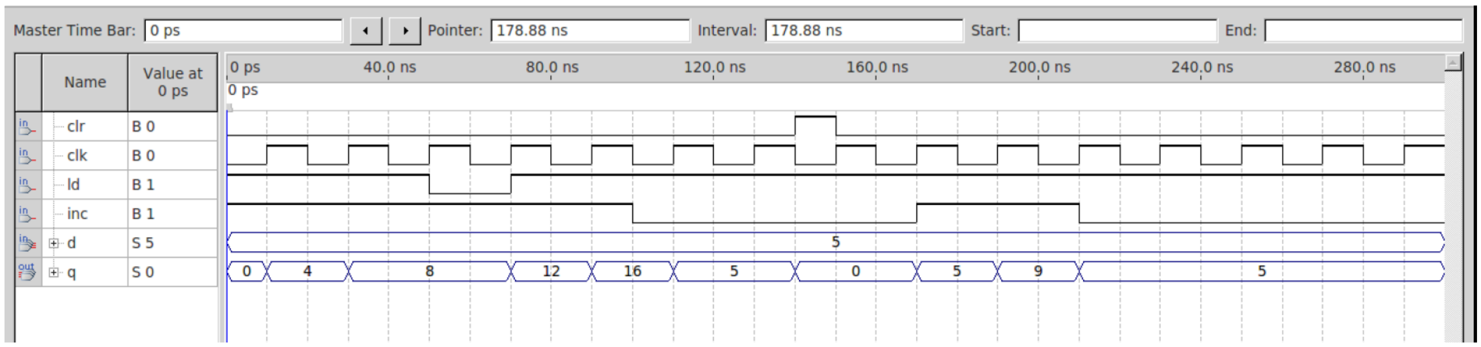
```
      );
   end component;
   signal add_out  :  std_logic_vector(31 downto 0);
   signal mux_out  :  std_logic_vector(31 downto 0);
   signal q_out    :  std_logic_vector(31 downto 0);
begin
   add0 : add port map(q_out, add_out);
   mux0 : mux2to1 port map(inc, d, add_out, mux_out);
   reg0 : register32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

Results:

Register 32:
When ld is high at the initial rising edge of the clock, the register takes in the value of d and stores it in q. This value in q remains unchanged until ld is high again on a subsequent rising clock edge, at which time q updates to d's new value. This cycle of updating continues each time there's a rising edge on the clock and ld is high, indicating that the register operates as intended: it transfers the input d to the output q upon receiving the ld command. In this waveform, the clr signal is not activated, so there's no instance where the output q resets to zero.


Master Time Bar: 0 ps Pointer: 223.1 ns Interval: 223.1 ns Start: End:

PC: Initially, with ld set high, q is configured to load the value from d on the first rising edge of clk. If ld is set low while inc is high and the PC design increments on a rising clock edge, then q should increment on each rising edge where inc remains high. The value of q changes only during the rising edges of the clock, and this occurs when either ld or inc is high. In scenarios where ld is high, q adopts the value from d. Conversely, when inc is high, the value of q increases incrementally.


Master Time Bar: 0 ps Pointer: 178.88 ns Interval: 178.88 ns Start: End:

Discussion:

The PC's primary role is to track the address of the next instruction, and its correct functioning is crucial for sequential instruction execution and control flow. If the PC's behaviour–incrementing after each instruction or jumping to specific addresses during branch or jump instructions—aligns with expected outcomes, it indicates a successful implementation, crucial for the system's overall functionality. Deviations from expected behaviour are equally significant, as they highlight potential errors or areas for improvement. The output values increment by 4 when the ‘inc’ is set high, and when the ‘ld’ value is set to low, the number stays constant until the value returns to being high. The rising edge value is 5, and it takes over the output once the ‘inc’ is set to low. A reset takes place and sets the output value to 0. After that, the process continues with the rising edge value and the increment of 4, which goes back to the rising edge value of 5. The PC code, VHDL code and the waveform function are correct according to the objective of this lab.