```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;
-- Declaration the inputs and the outputs of synchronous ROM
entity program_memory is
port (clk,reset   : in std_logic;
 load_IR          : in std_logic;
 address_load     : in std_logic;
 pc               : in std_logic_vector(4 downto 0);
 instruction      : out  std_logic_vector(7 downto 0));
end program_memory;

architecture Behavioral of program_memory is
-- BUILDING MEMORY STORAGE BY USING VHDL ARRAY INDEX
-- THE ARRAY WILL BE 0 TO 15 AS THERE ARE 16 LOCATIONS ADDRESS OF DATA IN THE MEMORY
-- THE WIDTH OF THE ARRAY IS 8 BIT
-- THE 16 INSTRUCTIONS REGISTER DATA LINE OF MEMORY WAS DESIGNED INTENTIALLY TO
PROVIDE A PARITICULAR COMMANDS
type ROM_type is array (0 to 31 )of std_logic_vector(7 downto 0);
 signal rom_addr: std_logic_vector(4 downto 0);  -- Defining a 4 bit rom address
signal
 constant rom_data: ROM_type:=(
 "00000100",                              ---  the instruction is to load x and y
from data memory
 "10001110",                              ---  the instruction is to do branch
operation
 "10101000",                              ---  the instruction is to do x-y
 "10101100",                              ---  the instruction is to do (a*2)
 "01100001",                              ---  the instruction is to store accumlato
 "00001110",                              ---  the instruction is to load w and z
from data memory
 "10100100",                              ---  the instruction is to do a+b
 "10110100",                              ---  the instruction is to a/2
 "01010000",                              ---  the instruction is to load (x-y)*2 on
B register
 "10111000",                              ---  the instruction is to do xor bitwise
operation
 "01010100",                              ---  the instruction is to load v on B
register from data_memory
 "10111100",                              ---  the instruction is to do a*b
 "01111100",                              ---  the instruction is to store the value
of accumlator
 "11110000",                              ---  the instruction is to load accumlator
toward output port
 "00100000",                              ---  the instruction is to load external
data from input port
```

```
 "11010000",                          ---  the instruction is to jump the
program counter to the first address
 "00000100",                          ---  the instruction is to load x and y
from data memory
 "10100100",                          ---  the instruction is to perform additio
 "10101100",                          ---  the instruction is to perform
multiplication by 2
 "01111000",                          ---  the instruction is to store the value
of accumlator
 "00010001",                          ---  the instruction is to load w and v
from data memory
 "10111100",                          ---  the instruction is to perform
multiplication
 "01011000",                          ---  the instruction is to load temp from
data memory to b register
 "10101000",                          ---  the instruction is to perform
subtraction
 "01111000",                          ---  the instruction is to store the value
of accumlator
 "00001110",                          ---  the instruction is to load z and v
from data memory
 "10100100",                          ---  the instruction is to perform additio
 "01011000",                          ---  the instruction is to load temp from
data memory to b register
 "10100100",                          ---  the instruction is to perform additio
 "01111100",                          ---  the instruction is to store the value
of accumlator
 "11100000",                          ---  the instruction is to load accumlator
toward output port
 "00100000");                         ---  the instruction is to load external
data from input port

begin

process(clk,reset)                              ------------  begin process
begin
if(reset = '1') then                            -------------- check if reset = 1
instruction <= x"00";                           -------------- reset the output
value of the memory
rom_addr   <= "00000";
elsif(rising_edge(clk)) then                    -------------- when rising edge is
detected
if(load_IR = '1') then                          -------------- when load signal is
set
instruction <= rom_data(to_integer(unsigned(rom_addr))); -------------- read the
instruction register of the memory based on the given address input
```

```
elsif(address_load = '1') then                          -------------- check if
load_address is set
rom_addr <= pc;                                        -------------- if load_address is
set, the program address will be loaded
end if;
end if;
end process;

end Behavioral;
```