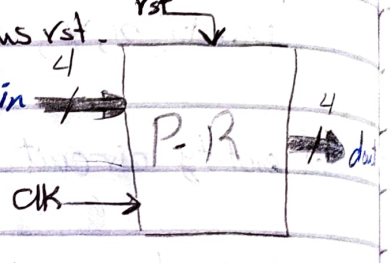
Subject : ..... Date: / /

Registers:-

Parallel Register is a flipflop, but it takes input of n-bit.



* VHDL Code of Register with Asynchronous rst.

```
entity reg is
  Port( din  : in std_logic_vector(3 downto 0);
        clk, rst : in std_logic;
        dout : out std_logic_vector(3 downto 0));
end entity reg;

architecture rtl of reg is
begin
    process(rst, clk)
    begin
       if(rst = '1') then
          dout <= (others => '0'); -- Initial state

       elsif(rising_edge(clk)) then
          dout <= din;
       endif;

    endprocess;
end rtl;
```

Subject : ..... Date: / /

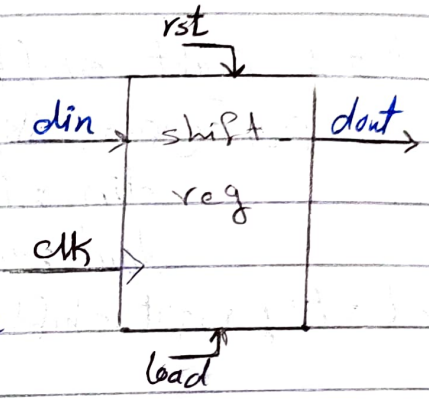
Shift Registers:-

① shift right register. ② Shift left register.

shift register performs two operations:-

① Load data

② shift data → left / right



we use control signal, "load", to choose which operation will be performed.

if load <= '1' → load data.

if load <= '0' → shift data.

* shift right with n bits ≃ divide by 2^n
* shift left with n bits ≃ multiply by 2^n

mode: in = internal signal	mode: out = internal signal

in assign operator "=":-

Signal which is defined as input, should be written in Left, and it can't be in Right. and the opposite for output signal. dout <= dout (✗ dout: 0);

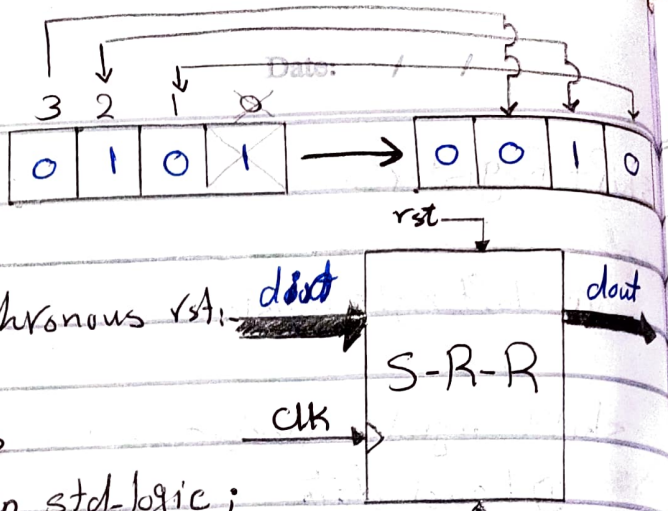
So, we will use temp internal signal to be able to use value of output signal in the right.

We can't use buffer mode because connected other circuits.

O/P signal type = input Port type. "mode".

Subject : ..... Date: / /

Shift Right Register :-



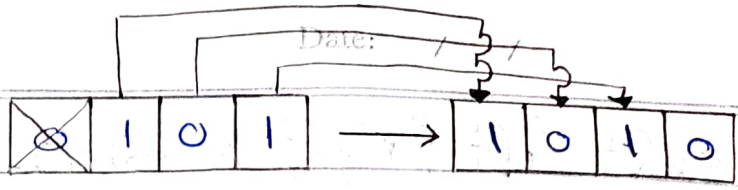
VHDL Code with Asynchronous rst :-

```
entity right_shift_reg is
    port(rst, clk, load : in std_logic;
         din : in std_logic_vector(3 downto 0); load
         dout : out std_logic_vector(3 downto 0));
end entity right_shift_reg;

architecture rtl of right_shift_reg is
  signal d_temp : std_logic_vector(3 downto 0);
  begin
    process(rst, clk)
     begin
       if(rst = '1') then
         d_temp <= (others => '0');

       elsif(rising_edge(clk)) then
         if(load = '1') then      * Read data operation.
           d_temp <= din;

         else                     * shift data operation.
           d_temp <= '0' & d_temp(3 downto 1);
         endif;
       end if;
     end process;
     dout <= d_temp;
end architecture rtl;
```

Subject : .....

Date: / /

shift left Register:-



VHDL-Code with Asynchronous rst:- "shift left 3bits"

```
entity left_shift_reg is
    Port(rst,clk,load: in std_logic;
         din: in std_logic_vector(7 downto 0);
         dout: out std_logic_vector(7 downto 0));
end entity left_shift_reg;

architecture rtl of left_shift_reg is
signal d_temp: std_logic_vector(7 downto 0);
 begin
  process(rst,clk)
begin  if(rst='1') then
        d_temp<=(others=>'0');
      elsif(rising_edge(clk)) then
        if(load='1') then
            d_temp<=din;
        else
            d_temp<=d_temp(4 downto 0) & "000";
        end if;
      endif;
  endprocess;
  dout<=d_temp;
end architecture rtl;
```

Subject : ..... Date: / /

shift left & right register:-

load "00" → load data
"01" → shift data to right
"10" → shift data to left.

VHDL Code with Asynchronous rst:-

```
entity shift_reg is
    Port( rst, clk, load : in std_logic;
          load           : in std_logic_vector(1 downto 0);
          din            : in std_logic_vector(3 downto 0);
          dout           : out std_logic_vector(3 downto 0));
end entity shift_reg;

architecture rtl of shift_reg is
signal d_temp : std_logic_vector(3 downto 0);
begin
 process(rst, clk)
    if(rst = '1') then
d_temp <= (others => '0');
    elsif (rising_edge(clk)) then
        if(load = "00") then -- load data
            d_temp <= din;
        elsif(load = "01") then -- shift data to right
            d_temp <= '0' & d_temp(3 downto 1);
        elsif(load = "10") then -- shift data to left
            d_temp <= d_temp(2 downto 0) & '0';
        endif;
    endif;
end process;
 dout <= d_temp;
end architecture rtl;
```