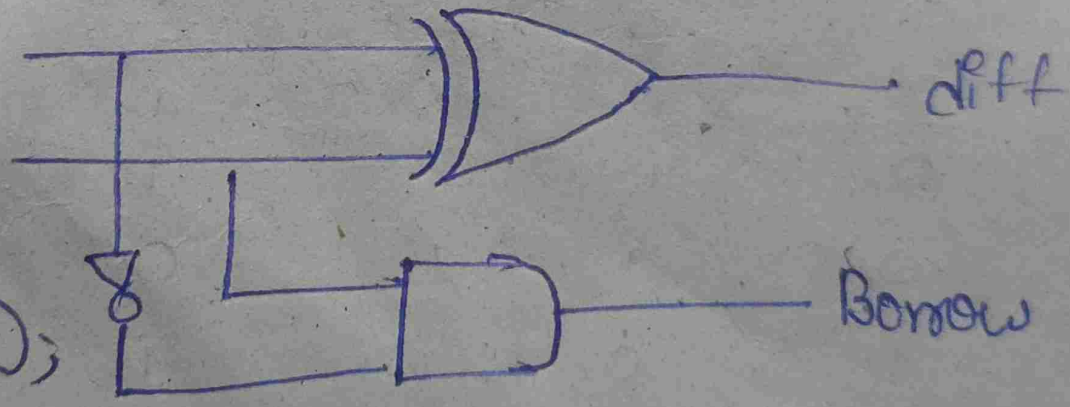
VHDL for half subtractor [data flow style]

A	B	diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

It can subtract two binary numbers. It has two inputs & two outputs.

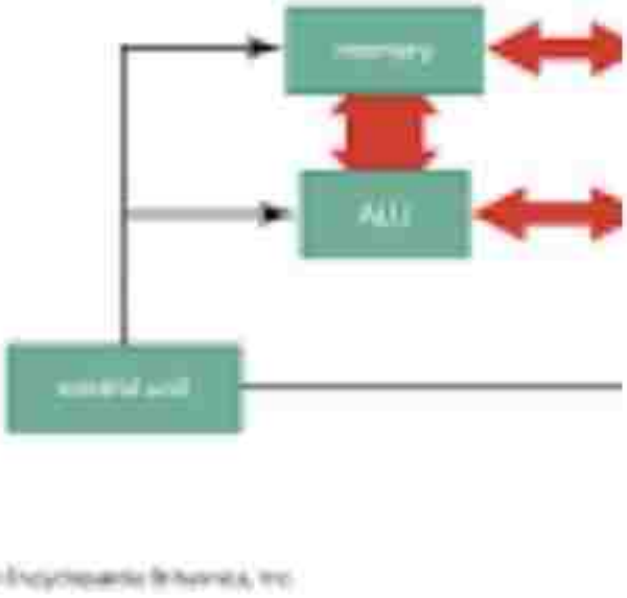


diff <= (A XOR B)

Borrow <= ((NOT A) AND B);

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity and_gate is
Port (A, B : in std_logic;
      Diff, Borrow : out std_logic);
end and_gate;
Architecture and_arch of and_gate is
begin
diff <= (A xOR B);
Borrow <= ( (NOT A) AND B);
end and_arch;
```

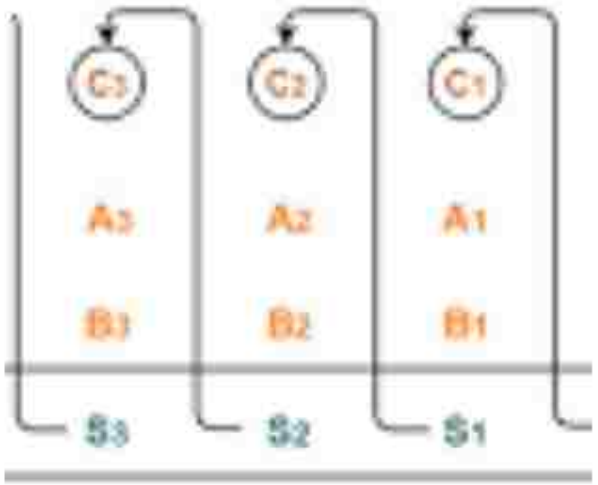
function in digital computer



The ALU performs **simple addition, subtraction, multiplication, division, and logic operations, such as OR and AND**. The memory stores the program's instructions and data.

operation	Inputs S2	S1	S0	outputs F
clear	0	0	0	F=0000
B-A	0	0	1	B-A
A-B	0	1	0	A-B
ADD	0	1	1	$A+B+C_{IN}$
XOR	1	0	0	A XOR B
OR	1	0	1	A OR B
AND	1	1	0	A AND B
PRESET	1	1	1	F=1111

Disadvantages of Ripple Carry Adder-



Adding two 4-bit Numbers

- Ripple Carry Adder does not allow to use all the full adders simultaneously.
- Each full adder has to necessarily wait until the carry bit becomes available from its adjacent full adder.
- This increases the propagation time.
- Due to this reason, ripple carry adder becomes extremely slow.

BCD to 7 segment decoder:

```
library IEEE;
use IEEE.STD_LOGIC_U64.all;
Entity codeconvestes is
Port (bcd : in std_logic_vector [3 downto 0];
      Leds : out std_logic_vector(6 downto 0));
end codeconvestes;

Architecture codeconverter_arch of codeconverter is
begin
process (bcd)
begin
 case bcd is
     when "0000" => leds <= "1111110";
     when "0001" => leds <= "0110000";
     when "0010" => leds <= "1101101";
     when "0011" => leds <= "1111001";
     when "0100" => leds <= "0110011";
     when "0101" => leds <= "1011011";
     when "0110" => leds <= "0011111";
     when "0111" => leds <= "1110000";
     when "1000" => leds <= "1111111";
     when "1001" => leds <= "1111011";
     when others => leds <= "-------";
 end case;
end process;
end code.converter_arch;
```

VHDL Code For 4 to 1 Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity mux_4to1 is
 port(

      A,B,C,D : in STD_LOGIC;
      S0,S1: in STD_LOGIC;
      Z: out STD_LOGIC
   );
end mux_4to1;

architecture bhv of mux_4to1
is
begin
process (A,B,C,D,S0,S1) is
begin
  if (S0 ='0' and S1 = '0')
then
       Z <= A;
  elsif (S0 ='1' and S1 =
'0') then
       Z <= B;
  elsif (S0 ='0' and S1 =
'1') then
       Z <= C;
  else
       Z <= D;
  end if;

end process;
end bhv;
```

VHDL Code for 1 to 4 Demux

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity demux_1to4 is
 port(

 F : in STD_LOGIC;
 S0,S1: in STD_LOGIC;
 A,B,C,D: out STD_LOGIC
 );
end demux_1to4;

architecture bhv of
demux_1to4 is
begin
process (F,S0,S1) is
begin
 if (S0 ='0' and S1 = '0')
then
 A <= F;
 elsif (S0 ='1' and S1 = '0')
then
 B <= F;
 elsif (S0 ='0' and S1 = '1')
then
 C <= F;
 else
 D <= F;
 end if;

end process;
end bhv;
```

Circuit Design of 4 to 16 Decoder Using 3 to 8 Decoder

A decoder circuit of the higher combination is obtained by adding two or more lower combinational circuits. 4 to 16 decoder circuit is obtained from two 3 to 8 decoder circuits or three 2 to 4 decoder circuits.

When two 3 to 8 Decoder circuits are combined the enable pin acts as the input for both the decoders. When enable pin is high at one 3 to 8 decoder circuits then it is low at another 3 to 8 decoder circuit.

VHDL Code for 4 to 2 encoder using logic gates

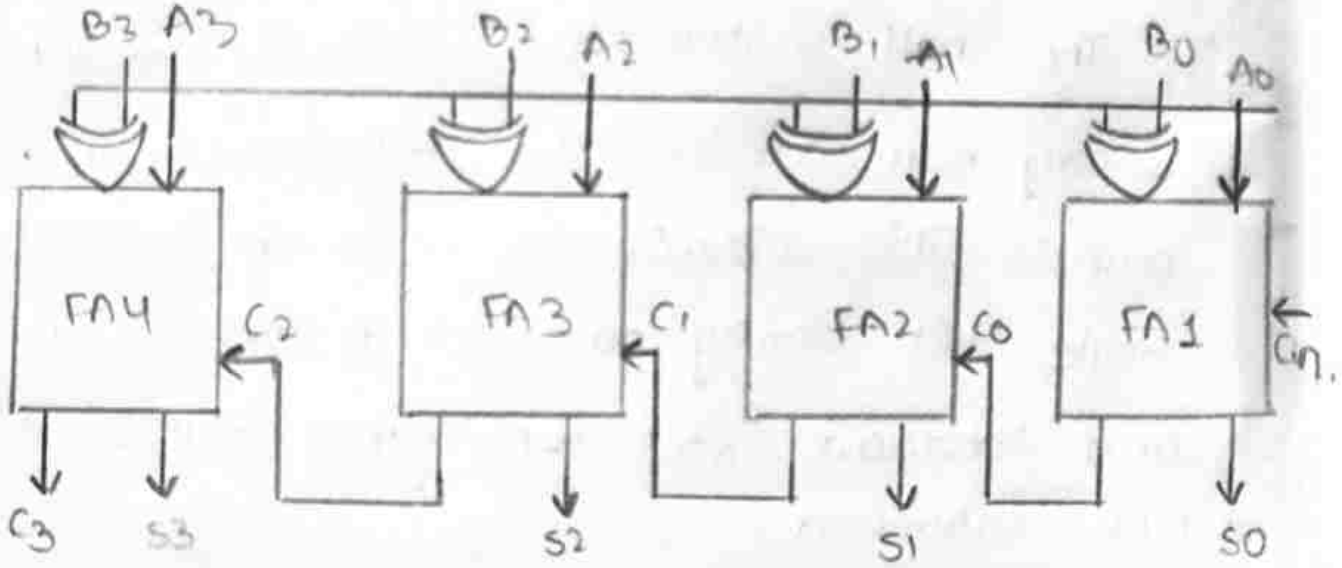
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity encoder2 is
 port(
 a : in STD_LOGIC_VECTOR(3
downto 0);
 b : out STD_LOGIC_VECTOR(1
downto 0)
 );
end encoder2;

architecture bhv of encoder2
is
begin

b(0) <= a(1) or a(2);
b(1) <= a(1) or a(3);

end bhv;
```

VHDL Code for 2 to 4 decoder using logic gates

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decoder2 is
 port(
 a : in STD_LOGIC_VECTOR(1
downto 0);
 b : out STD_LOGIC_VECTOR(3
downto 0)
 );
end decoder2;

architecture bhv of decoder2
is
begin

b(0) <= not a(0) and not
a(1);
b(1) <= not a(0) and a(1);
b(2) <= a(0) and not a(1);
b(3) <= a(0) and a(1);

end bhv;
```

b] Draw the circuit and explain the operation of binary adder subtractor.

Ans:



A Binary Adder subtractor is one which is capable of both addition and subtraction of binary numbers in one circuit itself. The operation being performed depends upon the binary value the control signal holds. It is one of the components of the ALU

This circuit requires Exor gate, full adder, Binary addition and subtraction

- As shown in fig, the first fulladder has control line directly as its input [Cin], The input A0 is directly input in FA 1. The 3rd input is the EXOR of B0 and M. The two outputs produced are S0 and C0

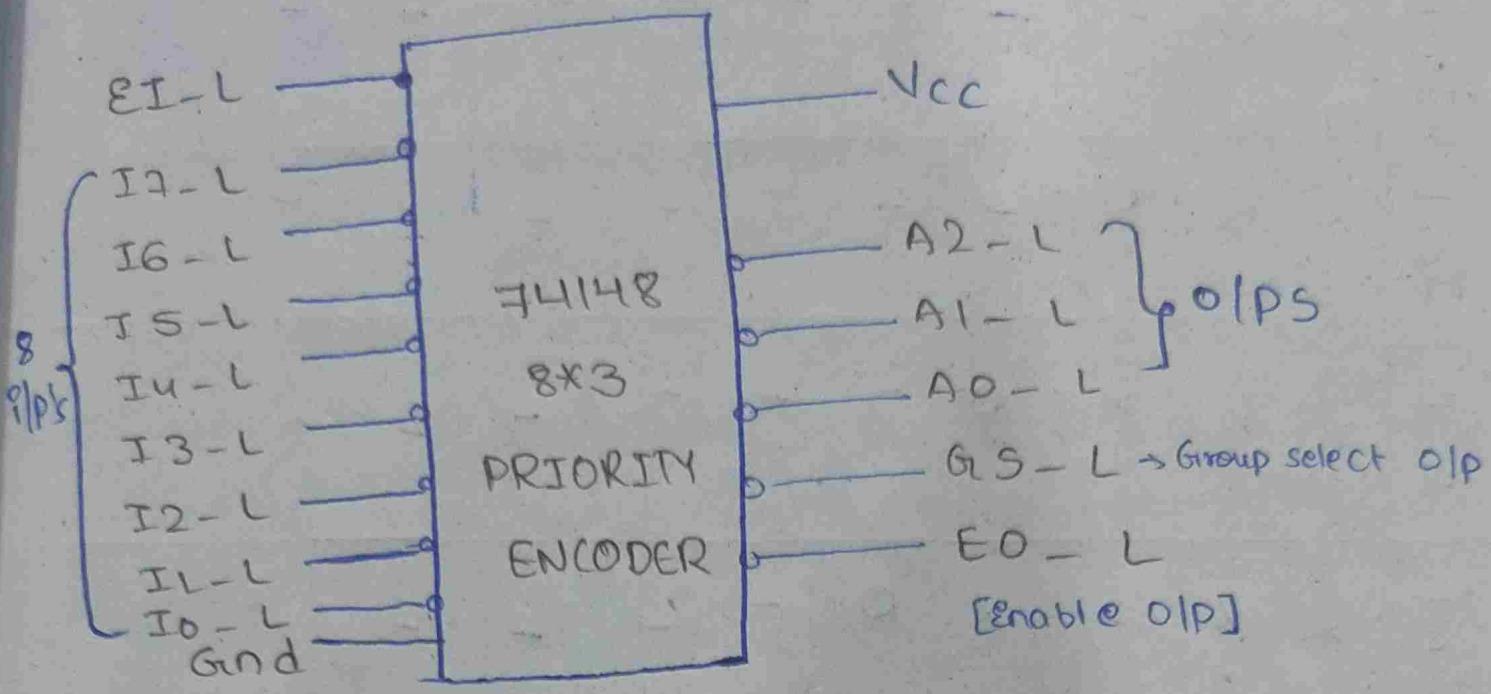
If the value of M is 1 the output of B0+B0' Thus the operation would be A+(B0') .Now 2's comp subtraction for two numbers A & B is given by A+B' This suggests that when K=1, the operation being performed on the four bit numbers is subtraction

parallel Binary Adder / subtractor:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
Entity PBAS is
 Port (A : in std_logic_vector (0 to 3);
       B in std_logic_vector (0 to 3);
       Cin, m : in std_logic;
       S : out std_logic_vector (0 to 3);
       C : out std_logic_vector (0 to 3));
End PBAS;
Architecture PBAS_arch of PBAS is
signal D0, D1, D2, D3, D4, D5, D6, D7 : std_logic;
component FA
 Port (I0, I1, I2 : in std_logic;
       sum, carry : out std_logic);
End component;
component XOR
 Port (I3, I4 : in std_logic;
       I5 : out std_logic);
End component;
begin
```

```
L1 : XOR portmap (B(0), D0, D1);
L2 : XOR portmap (B(1), D2, D3);
L3 : XOR portmap (B(2), D4, D5);
L4 : XOR portmap (B(3), D6, D7);
L5 : FA portmap (A(0), D1, Cin, S(0), C(0));
L6 : FA portmap (A(1), D3, C(0), S(1), C(1));
L7 : FA portmap (A(2), D5, C(1), S(2), C(2));
L8 : FA portmap (A(3), D7, C(2), S(3), C(3));

End PBAS_arch;
```

74148 [PRIORITY ENCODER] → (8*3)



GS-L [when more than 1 i/p is asserted]

$H_7 = I_7$

$H_6 = I_6 \cdot I_7'$

$H_5 = I_5 \cdot I_6' \cdot I_7'$

$H_4 = I_4 \cdot I_5' \cdot I_6' \cdot I_7'$

$H_3 = I_3 \cdot I_4' \cdot I_5' \cdot I_6' \cdot I_7'$

$H_2 = I_2 \cdot I_3' \cdot I_4' \cdot I_5' \cdot I_6' \cdot I_7'$

$H_1 = I_1 \cdot I_2' \cdot I_3' \cdot I_4' \cdot I_5' \cdot I_6' \cdot I_7'$

$H_0 = I_0 \cdot I_1' \cdot I_2' \cdot I_3' \cdot I_4' \cdot I_5' \cdot I_6' \cdot I_7'$

$A_2\text{-}L = H_4 + H_5 + H_6 + H_7$

$A_1\text{-}L = H_2 + H_3 + H_6 + H_7$

$A_0\text{-}L = H_1 + H_3 + H_5 + H_7$

EI-L	I_0-L	I_1-L	I_2-L	I_3-L	I_4-L	I_5-L	I_6-L	I_7-L	A_2-L	A_1-L	A_0-L	GS-L	EO-L
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	x	x	x	x	x	x	x	0	0	0	0	0	1
0	x	x	x	x	x	x	0	1	0	0	1	0	1
0	x	x	x	x	x	0	1	1	0	1	0	0	1
0	x	x	x	x	0	1	1	1	0	1	1	0	1
0	x	x	x	0	1	1	1	1	1	0	0	0	1
0	x	x	0	1	1	1	1	1	1	0	1	0	1
0	x	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
Entity priority_encoder is
Port(I_L: in std_logic_vector(7 downto 0);
     EI_L: in std_logic;
     A_L: out std_logic_vector(2 downto 0);
     GS_L, EO_L:out std_logic);
end priority_encoder;
Architecture priority_Encoder_arch of priority_
                                    Encoder is

signal EI,EO, GS: std_logic;
signal A: std_logic_vector(2 downto 0):="000";
signal I: std_logic_vector(7 downto 0);
 begin
 EI<=(not EI_L);
 I <=(not I_L);
 EO<='1';
 GS<='0';
 if EI='0' then EO<='0';
 else
 for j in 7 downto 0 loop
 if   I(j)='1' then
 GS<='1';
 EO<='0';
 A<= conv_std_logic_vector(j,0);
```

```
        exit;
    end if;
end loop;
end if;
EO_L <= (not EO);
GS_L <= (not GS);
A_L <= (not A);
end process;
end priority_Encoder_arch;
```

$P_3P_2P_1G_0 + P_3P_2G_1 + P_3G_2 + G_3$

Program: 4-BIT CLA Program

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
Entity cla_block is
Port (P, G : in std_logic_vector (3 down to 0);
      Cin : in std_logic;
      C : out std_logic_vector (4 down to 0));
end cla_block;
Architecture cla_arch of cla_block is
begin
C(0) <= Cin;
C(1) <= ((P(0) AND C(0)) OR G(0));
C(2) <= ((P(1) AND P(0) AND C(0)) OR ((P(1) AND G(0))
                                    OR G(1));
C(3) <= ((P(2) AND P(1) AND P(0) AND C(0)) OR ((P(2) AND P(1)
    AND G(0)) OR (P(2) AND G(1)) OR G(1));
```

C(4) <= (CP(3) AND P(2) AND P(1) AND P(0) AND C(0))
OR (P(3) AND P(2) AND P(1) AND G(0)) OR
(P(3) AND P(2) AND G(1)) OR (P(3) AND G(2))
OR (G(3));

4-CLA ADDER PROGRAM:

A[A3 A2 A1 A0] | S
B[B3 B2 B1 B0] | Cout
Co

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity adder is
Port (A,B: in std_logic_vector(3 downto 0);
      Co : in std_logic;
       s : out std_logic_vector(3 downto 0);
    Cout : out std_logic);
end adder;
Architecture adder_arch of adder is
signal P,G : std_logic_vector (3 downto 0):="0000";
signal C : std_logic_vector (4 downto 0):="0000";
component cla_block
 Port (P,G : in std_logic_vector (3 downto 0);
       Cin : in std_logic;
         C : out std_logic_vector (4 downto 0));
end component;
```

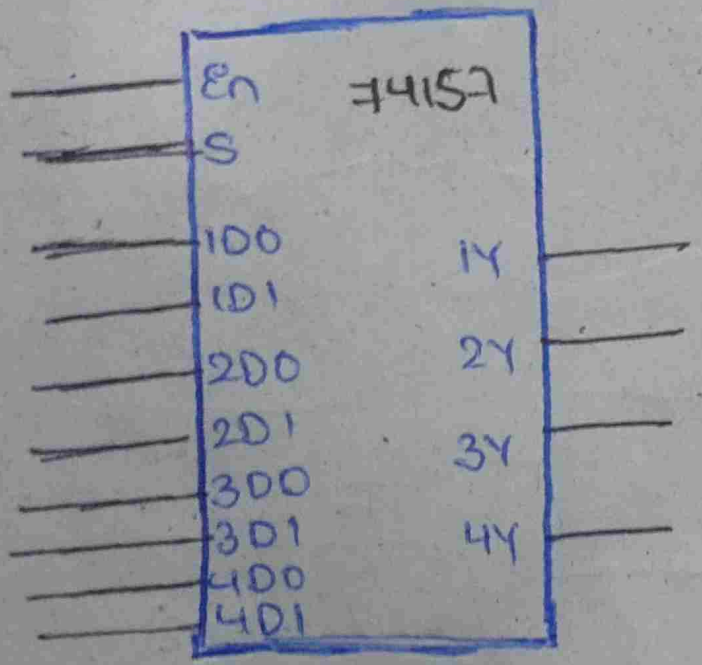
```
begin
P <= (A XOR B);
G <= (A AND B);

L1: cla_block port map (P, G, C0, C);
  S <= (P XOR C (3 down to 0));
  Cout <= C(4);

end adder_arch;
```

a) 2-INPUT 4-BIT MULTIPLEXER

The MSI, 74X157 is a 2-input, 4-bit Multiplexer. This multiplexer has two sets of 4-bit inputs. It also has 4-bit outputs. The single select input line allows the first set of four inputs or the second set of 4-inputs to be connected to the output. Thus four-bits of data from two sources are routed to the output. The function table and the circuit of the multiplexer are shown. table 18.1, figure 18.1

The multiplexer has two sets of 4-bit active-high inputs 1A, 2A, 3A, 4A and 1B, 2B, 3B, 4B respectively. The multiplexer has 4-bit active-high outputs 1Y, 2Y, 3Y 4Y. The single select input allows either the 4-bit input A or the 4-bit input B to be connected to the 4-bit output Y. The G active-low pin enables or disables the Multiplexer.

74157 [MULTIPLEXER]:



Inputs		Output
En_l	S	1Y 2Y 3Y 4Y
1	x	0 0 0 0
0	0	1D0 2D0 3D0 4D0
0	1	1D1 2D1 3D1 4D1

Asseted as → [1D0 2D0 3D0 4D0] → [A_3 A_2 A_1 A_0]

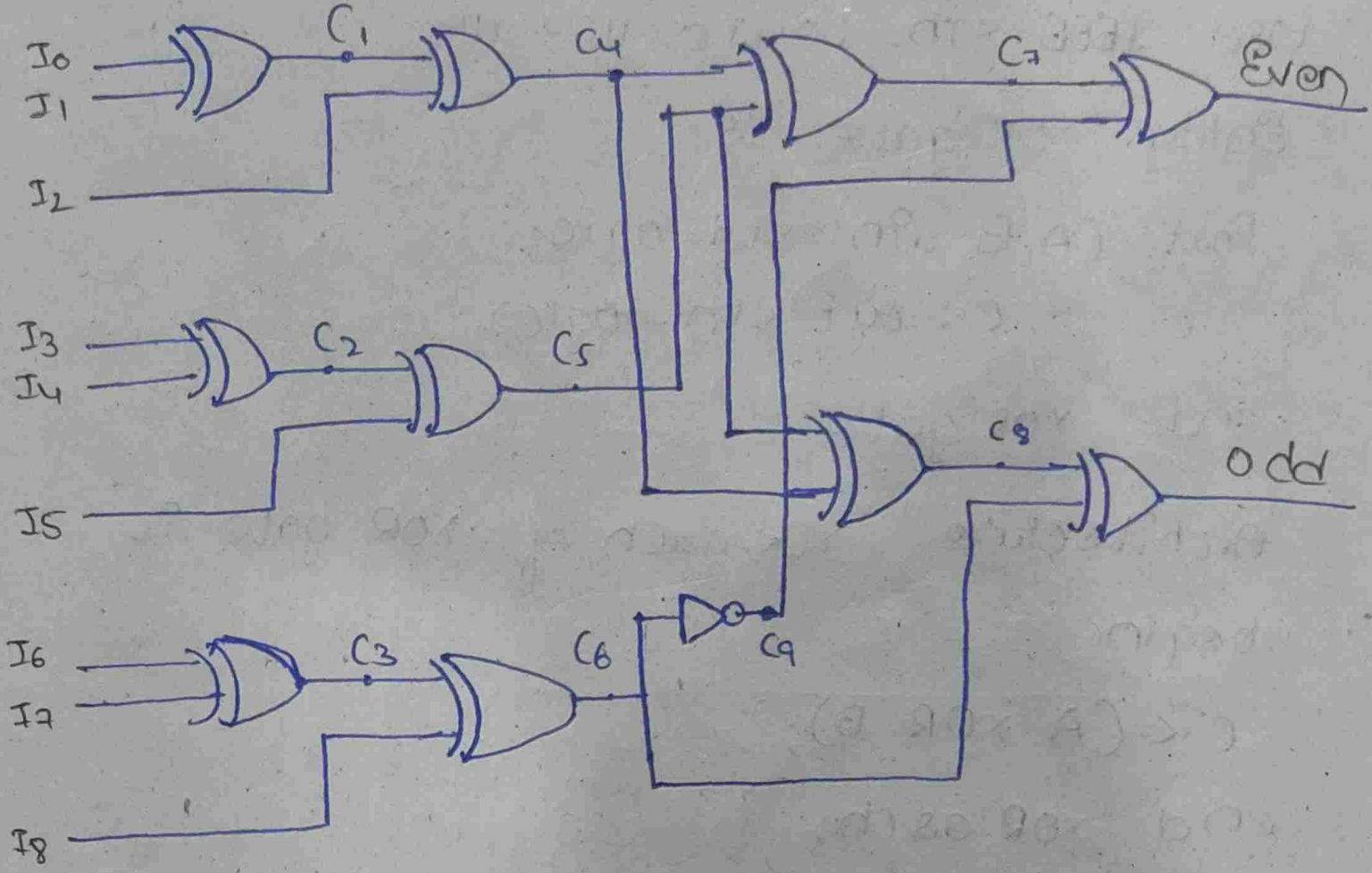
[1D1 2D1 3D1 4D1] → [B_3 B_2 B_1 B_0]

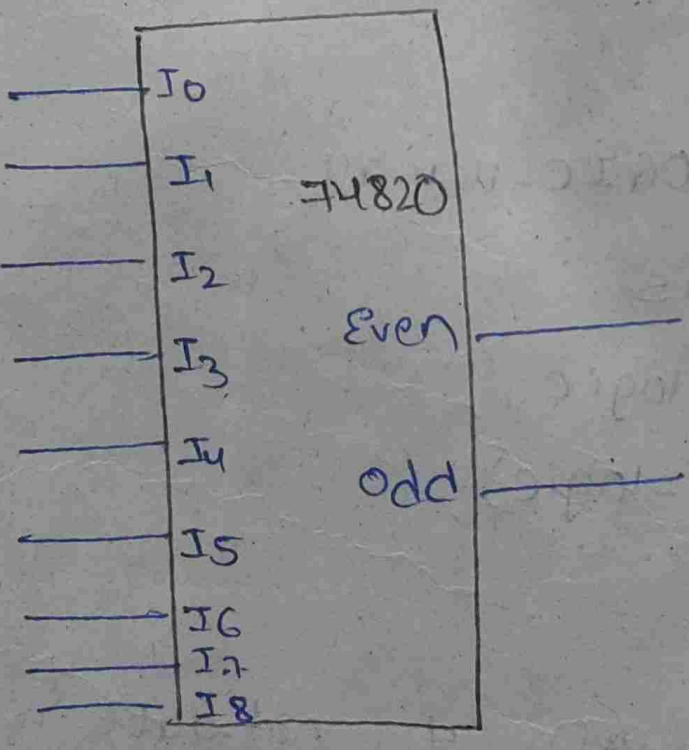
Y(3) Y(2) Y(1) Y(0)

Y → [1Y 2Y 3Y 4Y]

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
Entity mux is
Port (En_l : in std_logic;
      s : in std_logic;
      Y : out std_logic_vector(3 downto 0);
      A,B : in std_logic_vector(3 downto 0));
end mux;
Architecture mux_arch of mux is
signal k: std_logic_vector(3 downto 0);
begin
with s select
      k <= A when '0';
           B when '1';
           '0' when others;
      Y <= k when En_l = '0' else Y <= "000";
end mux;
```

74820 [9-bit parity Generator]:





XOR Gate:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
Entity XOR_gate is
Port (A,B : in std_logic;
      C : out std_logic);
end XOR_gate;
Architecture XOR_arch of XOR gate is
begin
C <=(A XOR B);
end XOR_arch;
```

NOT gate:

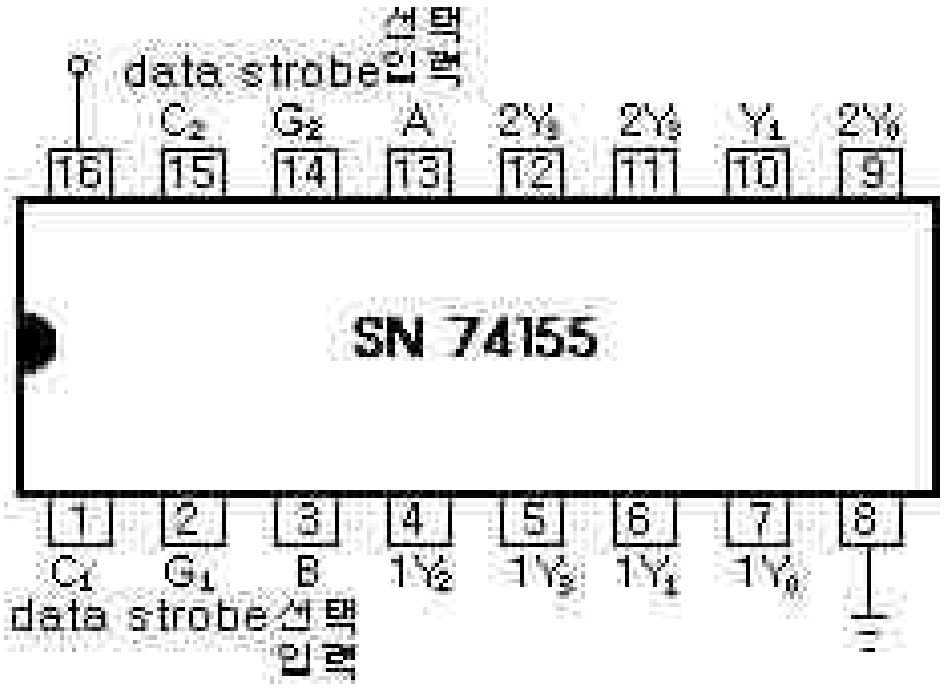
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
Entity NOT_gate is
Port (D: in std_logic;
      Y : out std_logic);
end NOT_gate;
Architecture NOT_arch of not_gate is
begin
Y <= (NOT D);
end NOT_arch;
```

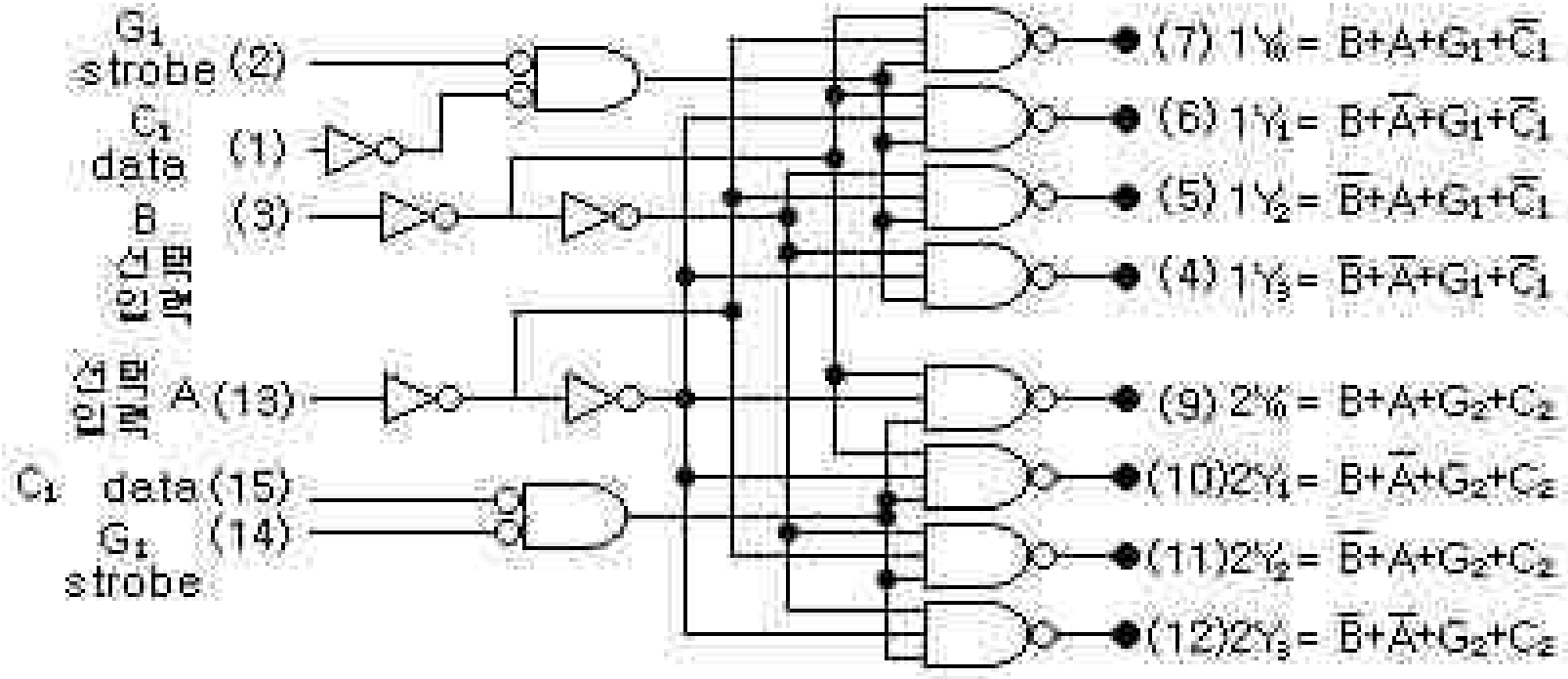
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

Entity 9-bit parity is
Port ( I: in  std_logic_vector(8 downto 0);
       Even,odd:out std_logic);
end 9-bit parity;

Architecture parity_arch of 9bitparity is
signal c:std_logic_vector(9 downto 0);
component XOR_gate
Port (A,B: in std_logic;
      C: out std_logic);
end component;
component NOT_gate
Port (D:in std_logic;
      Y: out std_logic);
end component;
begin
L1: XOR1  port map( I(0), I(1), C(1));
L2 : XOR2  port map( I(3), I(4), C(2));
L3 : XOR3  port map ( I(6), I(7), C(3));
L4 : XOR4  port map (C(1), I(2), C(4));
L5 : XOR5  port map (C(2), I(5), C(5));
L6 : XOR6  port map (C(3), I(8), C(6));
L7 : XOR7  port map (C(4), C(5), C(7));
```

```
L8 : XOR8  port map (c(4), c(5), c(8));
L9 : NOT1  port map (c(6), c(9));
L10: XOR9  port map (c(7), c(9), even);
L11 : XOR10 port map (c(8), c(6), odd);

end parity_arch;
```

VHDL Code for 2 to 4 decoder using case statement

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decoder is
port(
a : in STD_LOGIC_VECTOR(1
downto 0);
b : out STD_LOGIC_VECTOR(3
downto 0)
);
end decoder;
architecture bhv of decoder
is
begin

process(a)
begin
case a is
when "00" => b <= "0001";
when "01" => b <= "0010";
when "10" => b <= "0100";
when "11" => b <= "1000";
end case;
end process;

end bhv;
```


선택
입력
data strobe
C₂
G₂
A
2Y₃
2Y₂
Y₁
2Y₀
16
15
14
13
12
11
10
9
SN 74155
1
2
3
4
5
6
7
8
C₁
G₁
B
1Y₂
1Y₃
1Y₁
1Y₀
data strobe
선택
입력


G₁
strobe (2)
C₁
data (1)
B (3)
선택입력
A (13)
선택입력
C₁ data (15)
G₁ (14)
strobe
(7) 1Y₀ = B+A+G₁+C̄₁
(6) 1Y₁ = B+Ā+G₁+C̄₁
(5) 1Y₂ = B̄+A+G₁+C̄₁
(4) 1Y₃ = B̄+Ā+G₁+C̄₁
(9) 2Y₀ = B+A+G₂+C₂
(10) 2Y₁ = B+Ā+G₂+C₂
(11) 2Y₂ = B̄+A+G₂+C₂
(12) 2Y₃ = B̄+Ā+G₂+C₂