پارسا محمدپور – ۹۸۲۴۳۰۵۰

پویا جهانگیری – ۹۸۲۴۳۰۷۶

گزارش آزمایش:

در این جلسه با توجه به اینکه در پیش گزارش آزمایش قبلی، کار پیاده‌سازی کد vhdl و تست کردن ماژول alu در مدل‌سیم انجام شد، بعد از توضیح کوتاهی در باب پیاده‌سازی آن، به سراغ کارهای صورت گرفته در این جلسه می‌رویم.

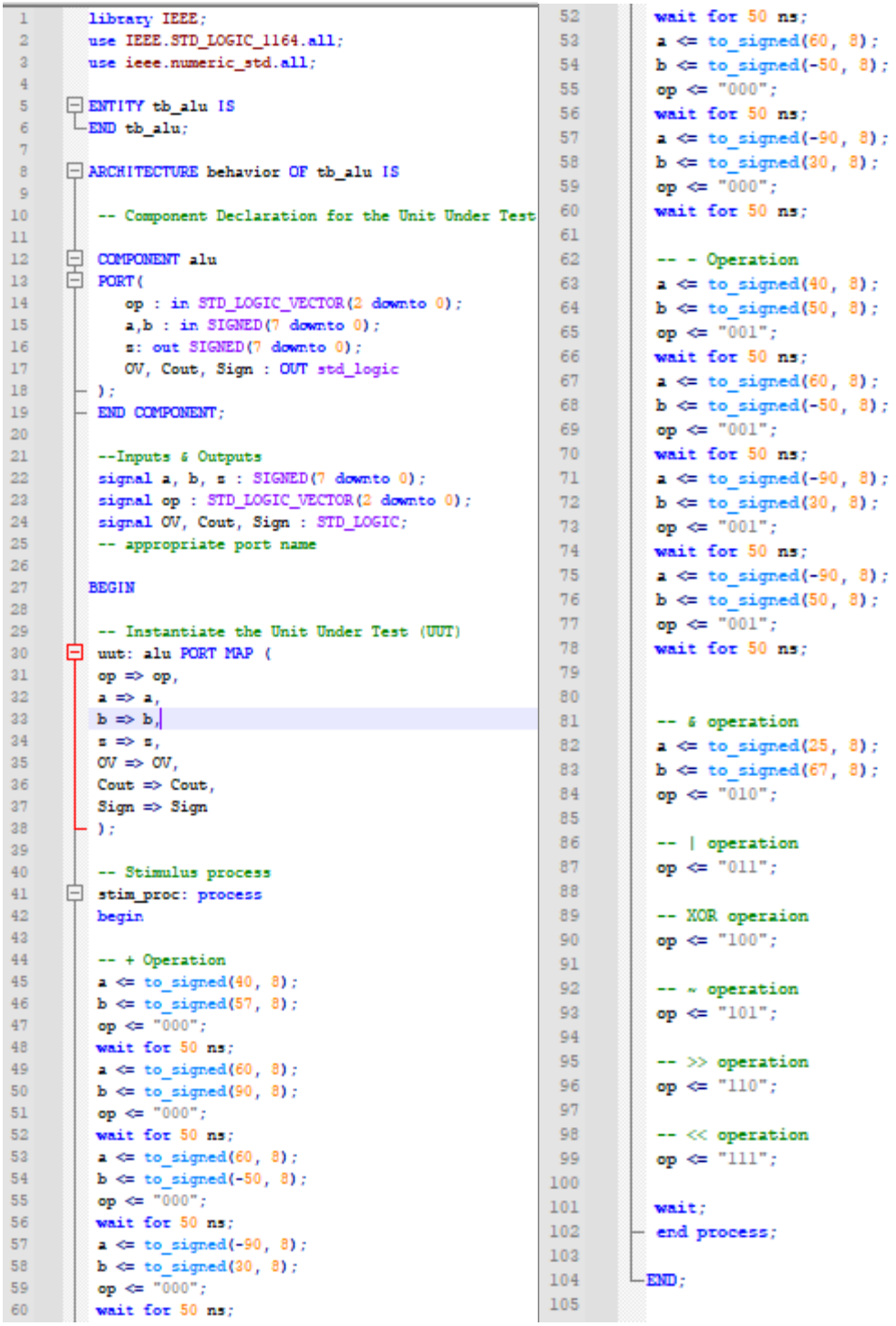
۱- پیاده‌سازی ماژول alu:

با توجه به صورت سوال، ماژول alu را پیاده‌سازی کردیم. طوریکه ورودی آن، دو تا عدد هشت بیتی علامت دار به همراه یه بیت Operation باشد، و همچنین خروجی آن نیز یک عدد هشت بیتی خروجی به همراه سه بیت گفته شده دیگر باشد. سپس به ازای تمام خروجی‌های ممکن، یک سیگنال تعریف کردیم و هرکدام از این سیگنال‌های مورد نظر را به نحو گفته شده پر کردیم. سپس با توجه به اینکه ورودی operation چه حالتی داشته است، یکی از این سیگنال‌ها را به خروجی assign کردیم. کدهای مربوطه در فایل مربوط به جلسه قبلی نیز قرار داده شده است، اما دوباره آنها را در اینجا می‌آوریم:

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

entity alu is
 port(
    op : in STD_LOGIC_VECTOR(2 downto 0);
    a,b : in SIGNED(7 downto 0);
    z: out SIGNED(7 downto 0);
    OV, Cout, Sign : OUT std_logic
 );
end alu;

architecture bhv of alu is
 -- Component Declaration for the Unit Under Test (UUT)

 signal a_plus_b, a_minus_b, a_and_b, a_or_b, a_xor_b, a_not, a_one_shift_right, b_one_shift_left, z_res : SIGNED(7 downto 0);

begin
    a_plus_b <= a + b;
    a_minus_b <= a - b;
    a_or_b <= a or b;
    a_xor_b <= a xor b;
    a_and_b <= a and b;
    a_not <= not a;
    a_one_shift_right <= a srl 1;
    b_one_shift_left <= b sll 1;
```

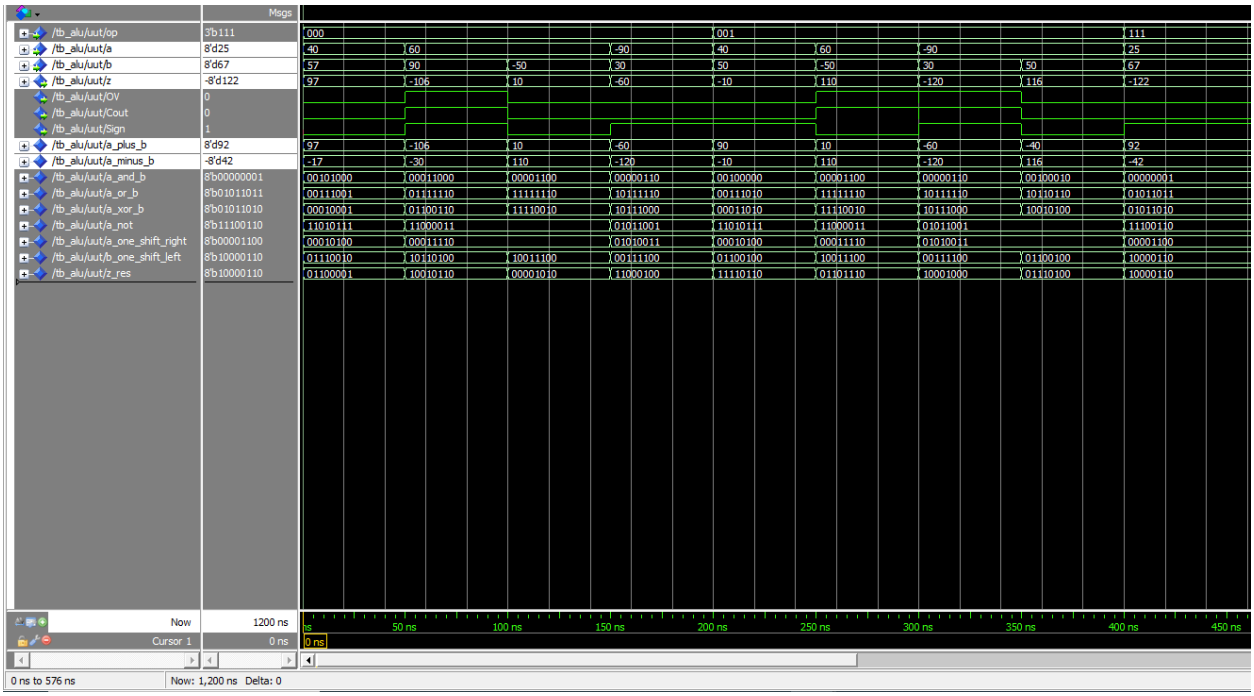
```
begin
    a_plus_b <= a + b;
    a_minus_b <= a - b;
    a_or_b <= a or b;
    a_xor_b <= a xor b;
    a_and_b <= a and b;
    a_not <= not a;
    a_one_shift_right <= a srl 1;
    b_one_shift_left <= b sll 1;

    z_res <= a_plus_b when op = "000" else
        a_minus_b when op = "001" else
        a_and_b when op = "010" else
        a_or_b when op = "011" else
        a_xor_b when op = "100" else
        a_not when op = "101" else
        a_one_shift_right when op = "110" else
        b_one_shift_left when op = "111";

    Cout <=  ((a(7) and b(7) and (not z_res(7))) or ((not a(7)) and (not b(7)) and z_res(7))) when op = "000" else
          ((a(7) and (not b(7)) and z_res(7)) or ((not a(7)) and b(7) and (not z_res(7))))when op = "001" else '0';
    z <= z_res(7 downto 0);
    OV <= ((a(7) and b(7) and (not z_res(7))) or ((not a(7)) and (not b(7)) and z_res(7))) when op = "000" else
          ((a(7) and (not b(7)) and z_res(7)) or ((not a(7)) and b(7) and (not z_res(7))))when op = "001" else '0';

    Sign <= z_res(7);

end bhv;
```

در نوشتن فایل تست‌بنچ به یک چالش برخورد کردیم و آن هم نحوه ورودی دادن برای ورودی‌های signed بود. که برای آن با توجه به سرچ‌های صورت گرفته، دیدیم باید مقدار ورودی را برحسب int به تابعی برای تبدیل int به unsigned دهیم که در ادامه کدهای مربوط به این بخش نیز قرار گرفته است. حال در ادامه کدهای فایل مربوط به تست‌بنچ را قرار می‌دهیم:



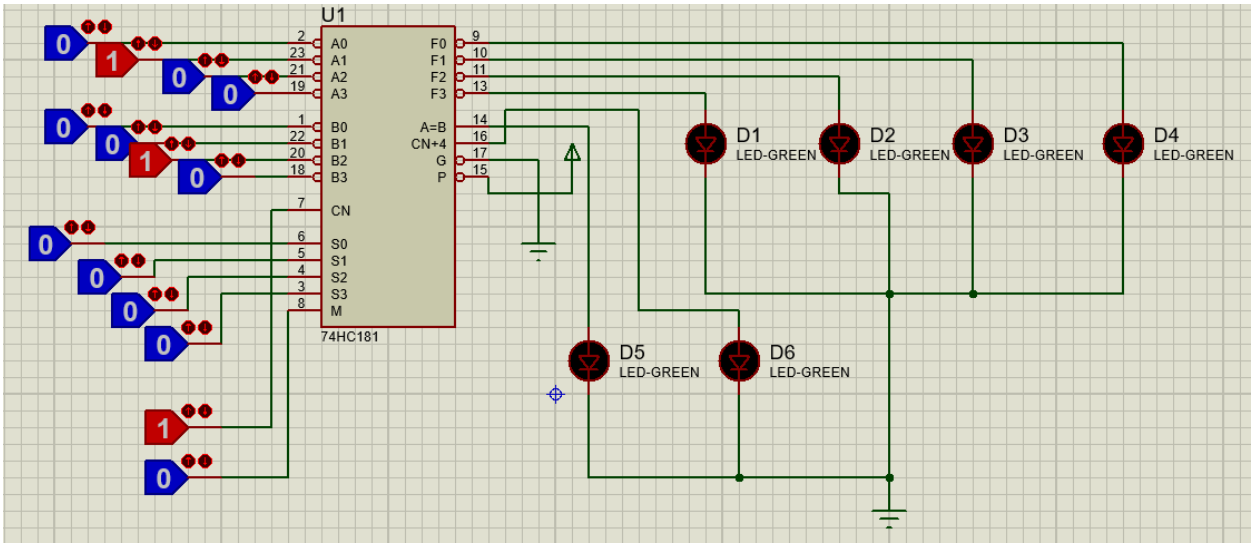
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

ENTITY tb_alu IS
END tb_alu;

ARCHITECTURE behavior OF tb_alu IS

 -- Component Declaration for the Unit Under Test

 COMPONENT alu
 PORT(
    op : in STD_LOGIC_VECTOR(2 downto 0);
    a,b : in SIGNED(7 downto 0);
    z: out SIGNED(7 downto 0);
    OV, Cout, Sign : OUT std_logic
 );
 END COMPONENT;

 --Inputs & Outputs
 signal a, b, z : SIGNED(7 downto 0);
 signal op : STD_LOGIC_VECTOR(2 downto 0);
 signal OV, Cout, Sign : STD_LOGIC;
 -- appropriate port name

 BEGIN

 -- Instantiate the Unit Under Test (UUT)
 uut: alu PORT MAP (
 op => op,
 a => a,
 b => b,
 z => z,
 OV => OV,
 Cout => Cout,
 Sign => Sign
 );

 -- Stimulus process
 stim_proc: process
 begin

 -- + Operation
 a <= to_signed(40, 8);
 b <= to_signed(57, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(60, 8);
 b <= to_signed(90, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(60, 8);
 b <= to_signed(-50, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(-90, 8);
 b <= to_signed(30, 8);
 op <= "000";
 wait for 50 ns;
 a <= to_signed(-90, 8);
 b <= to_signed(-50, 8);
 op <= "000";
 wait for 50 ns;

 -- - Operation
 a <= to_signed(40, 8);
 b <= to_signed(50, 8);
 op <= "001";
 wait for 50 ns;
 a <= to_signed(60, 8);
 b <= to_signed(-50, 8);
 op <= "001";
 wait for 50 ns;
 a <= to_signed(-90, 8);
 b <= to_signed(30, 8);
 op <= "001";
 wait for 50 ns;
 a <= to_signed(-90, 8);
 b <= to_signed(50, 8);
 op <= "001";
 wait for 50 ns;


 -- & operation
 a <= to_signed(25, 8);
 b <= to_signed(67, 8);
 op <= "010";

 -- | operation
 op <= "011";

 -- XOR operaion
 op <= "100";

 -- ~ operation
 op <= "101";

 -- >> operation
 op <= "110";

 -- << operation
 op <= "111";

 wait;
 end process;

END;
```

در ادامه نتیجه حاصل از شبیه‌سازی را قرار می‌دهیم:



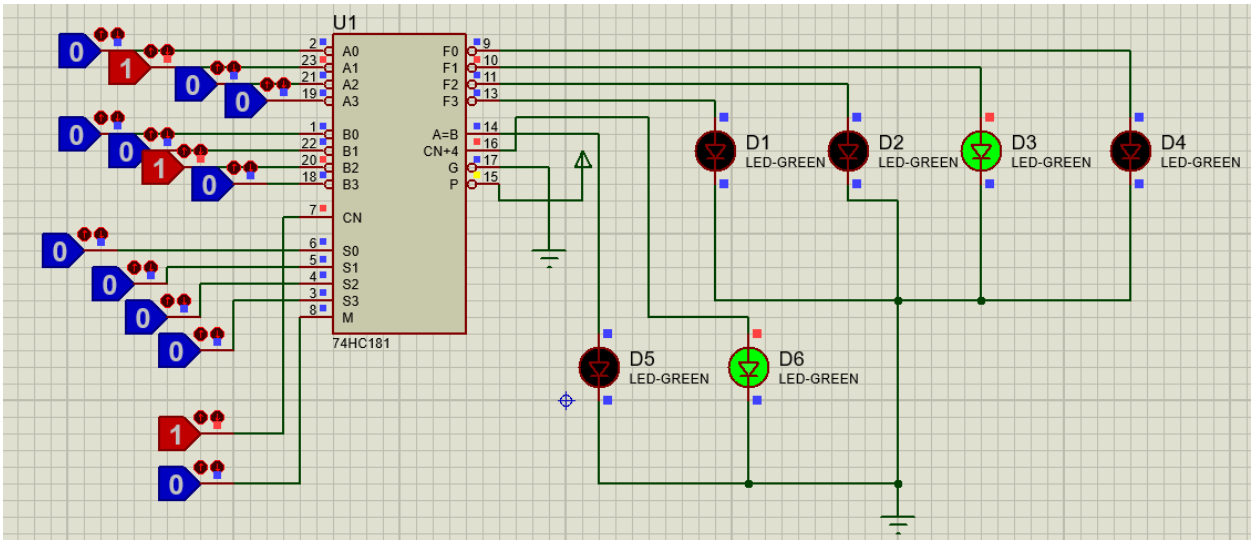
۱- پیاده‌سازی شبیه‌سازی پروتئوس برای ماژول ALU:

حال به سراغ دیگر کارهای صورت گرفته در این جلسه می‌رویم. در این جلسه بعد از اینکه مقداری به بقیه بچه‌ها کمک کردیم و کارهای چند نفر را تحویل گرفتیم، به سراغ تسک بعدی رفتیم. در این جلسه از ما خواسته شده بود تا از ماژول آماده alu در پروتئوس استفاده کنیم و چند تا از ورودی‌های ممکن را ب آن بدهیم و در دو حالت آن را تست کنیم. در ادامه عکس‌های مربوط به این فعالیت را خواهیم آورد. ابندا عکس مربوط به شاتیک مدار را می‌آوریم که به صورت زیر می‌باشد:

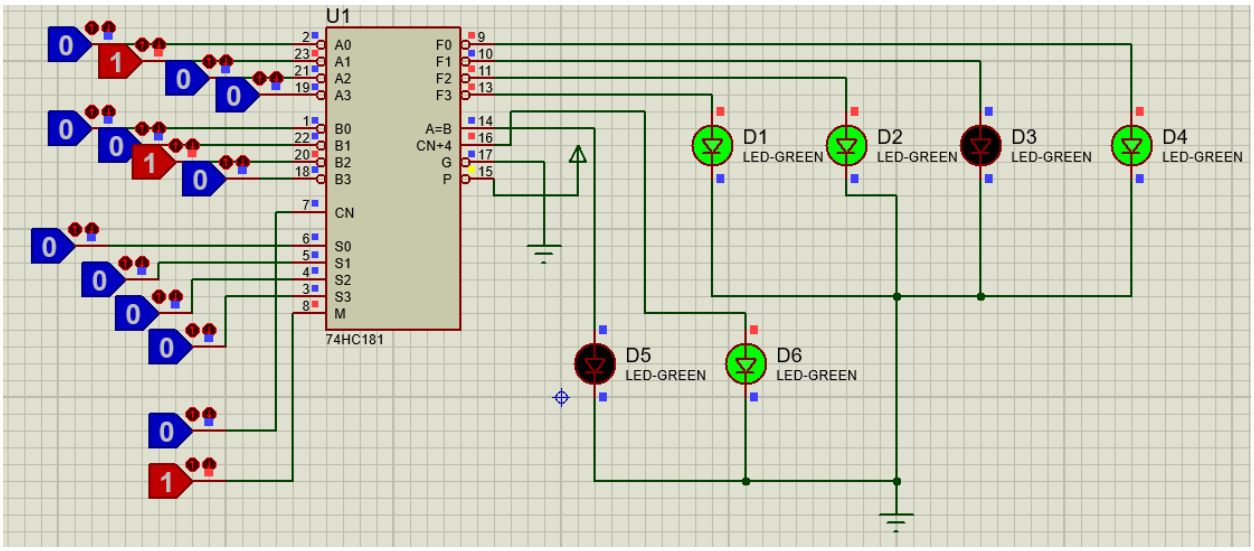


حال در ادامه به سراغ اعمال ورودی‌های مختلف operation یا همانs (در اینجا s می‌باشد) می‌رویم و برای هر دو حالت arithmetic و logic خروجی مورد نظر را قرار خواهیم داد. همچنین شایان ذکر است که برای استفاده از این ماژول (ماژول 74HC181) ابتدا سرچ کردیم و سپس چند فایل پی‌دی‌اف برای دیتاشیت ابن ماژول پیدا کردیم و سپس با کمک گرفتن از آن و همچنین کمک‌های استاد، به پیاده‌سازی آن پرداختیم. در ادامه عکس حالت‌های مختلف operation در خروجی‌های متناظر با آن قرار داده شده است:

- ورودی 0000:
  - Arithmetic:

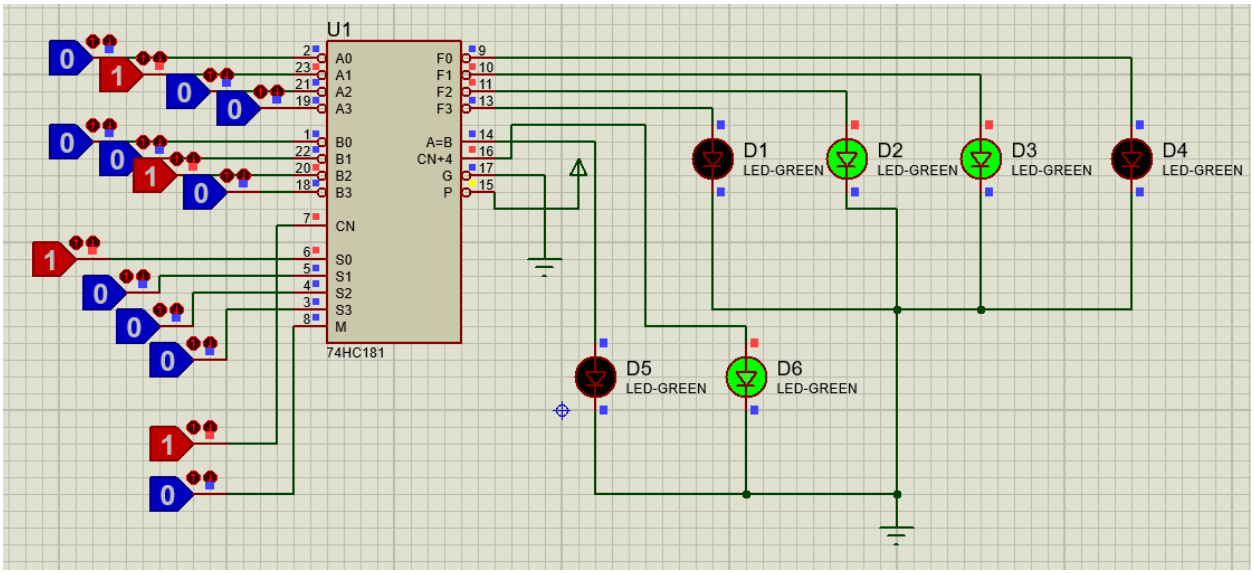


- Logic:

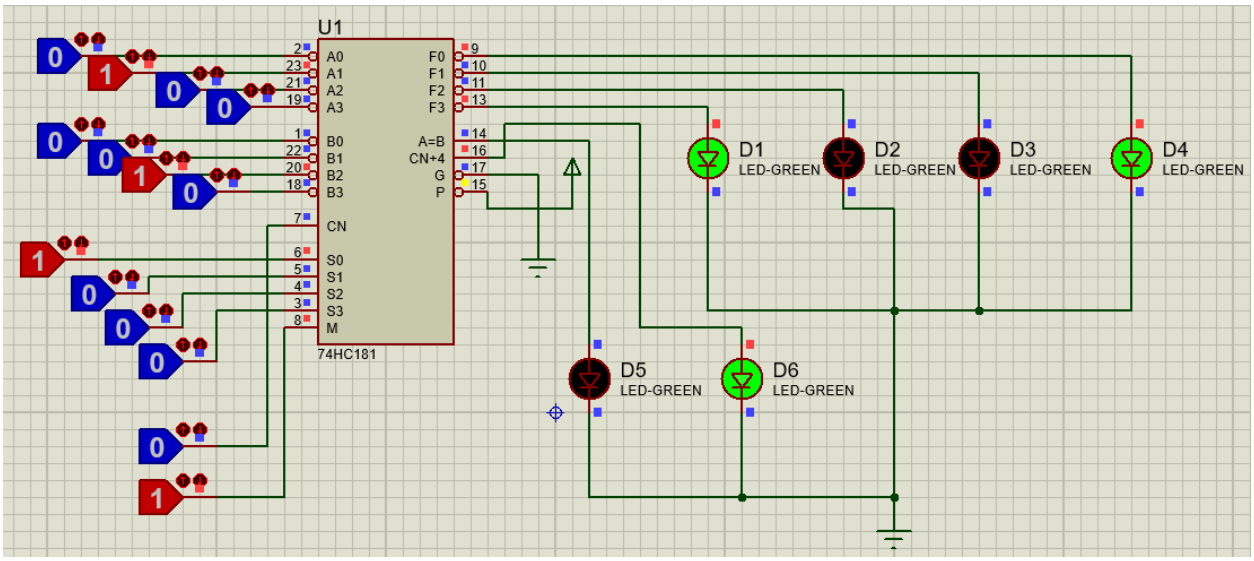


ورودی 0001:

- Arithmetic:

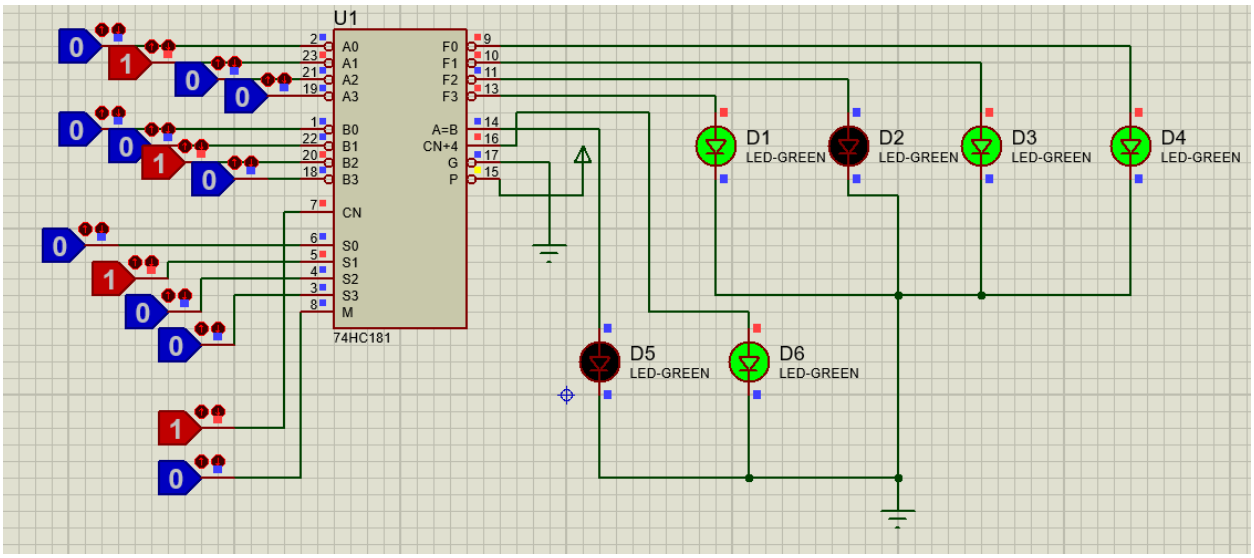


- Logic:

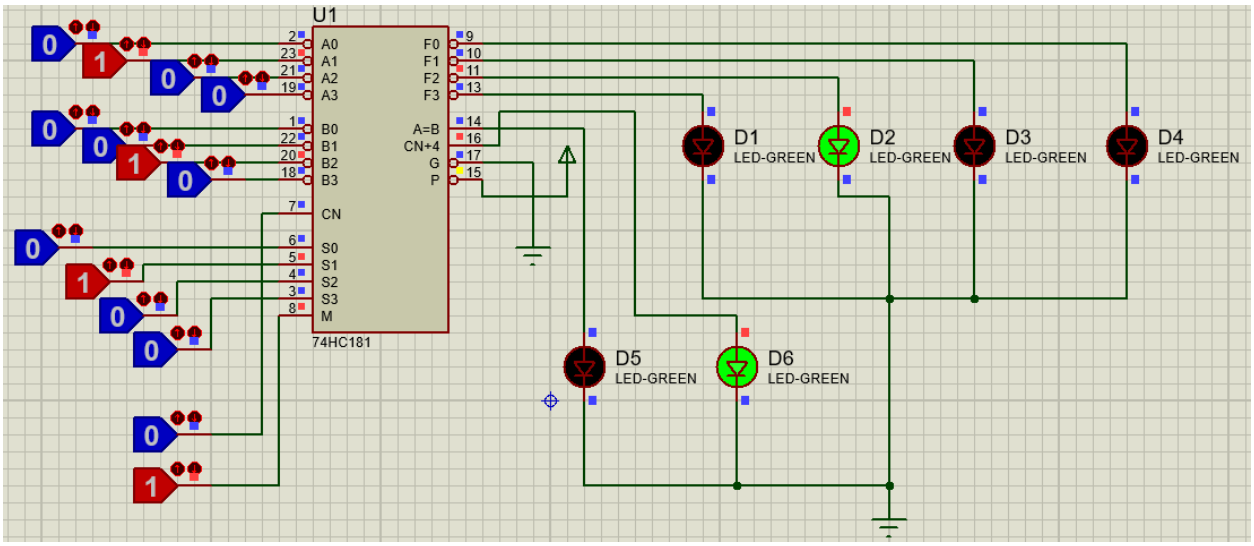


ورودی 0010:

- Arithmetic:

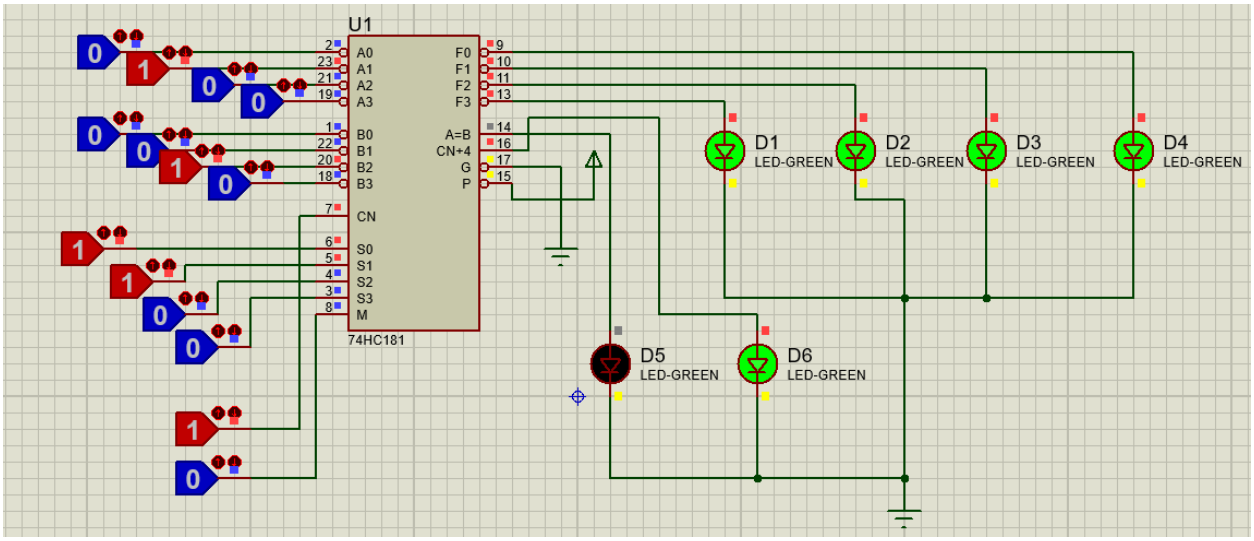


- Logic :

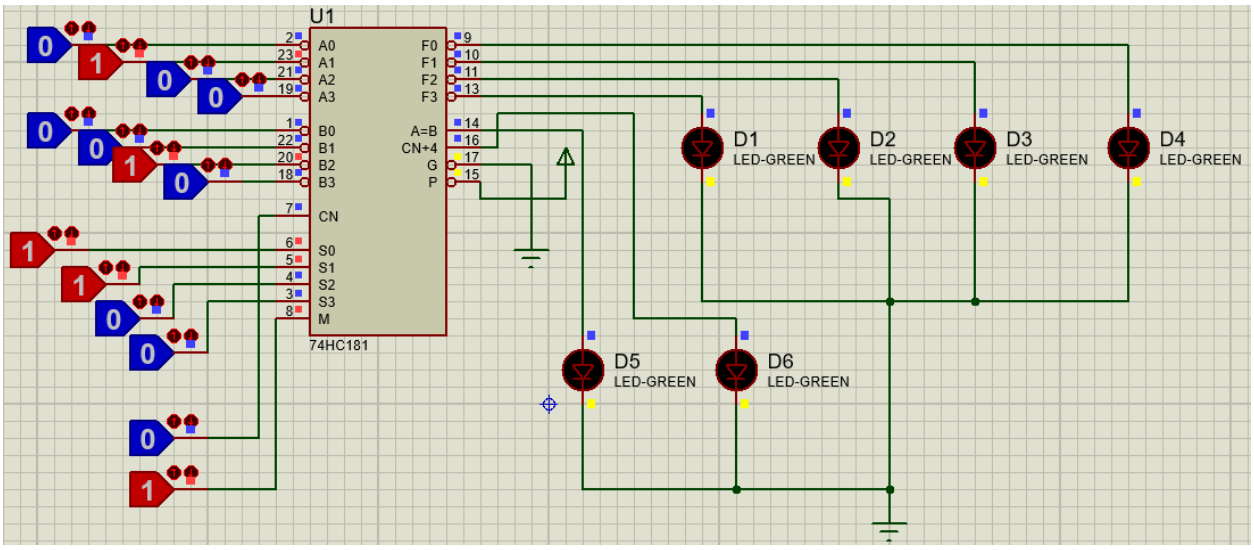


- ورودی 0011:

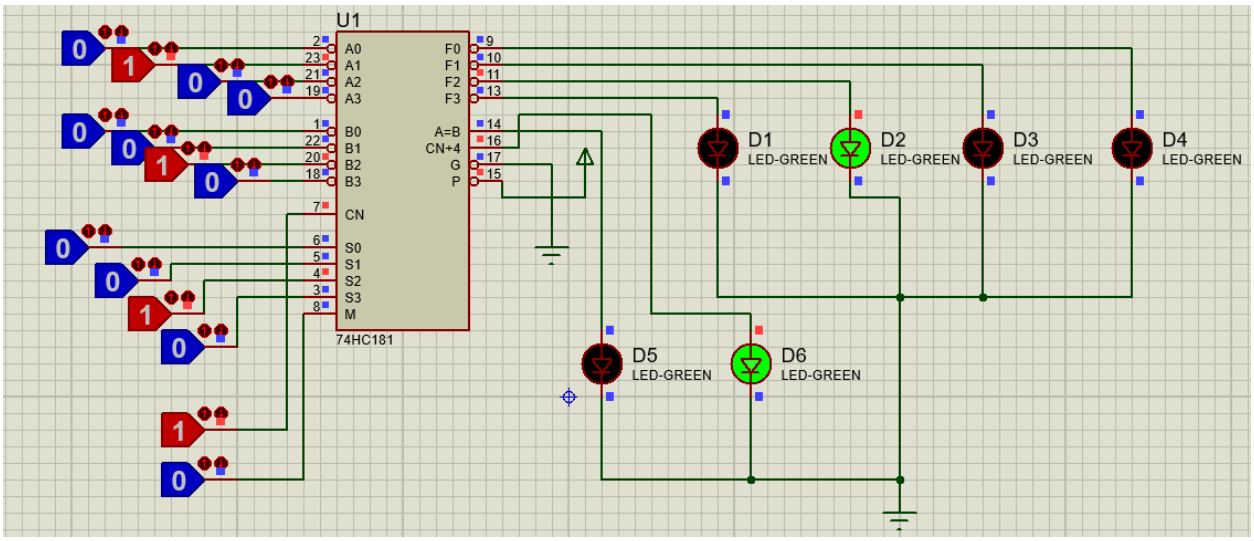
- Arithmetic :



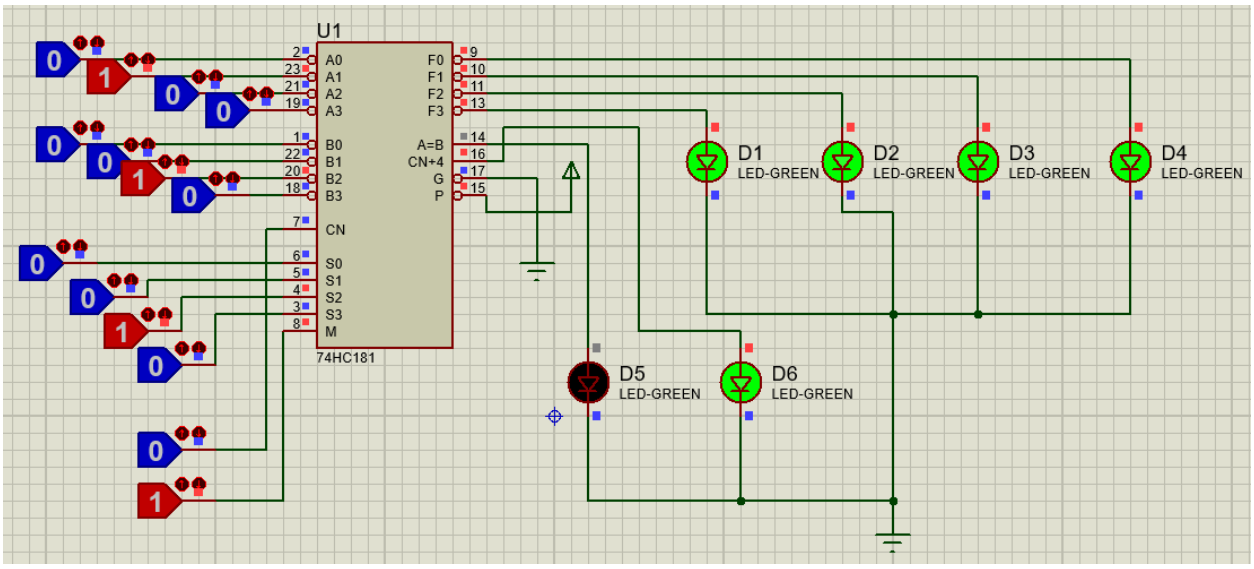
- Logic:



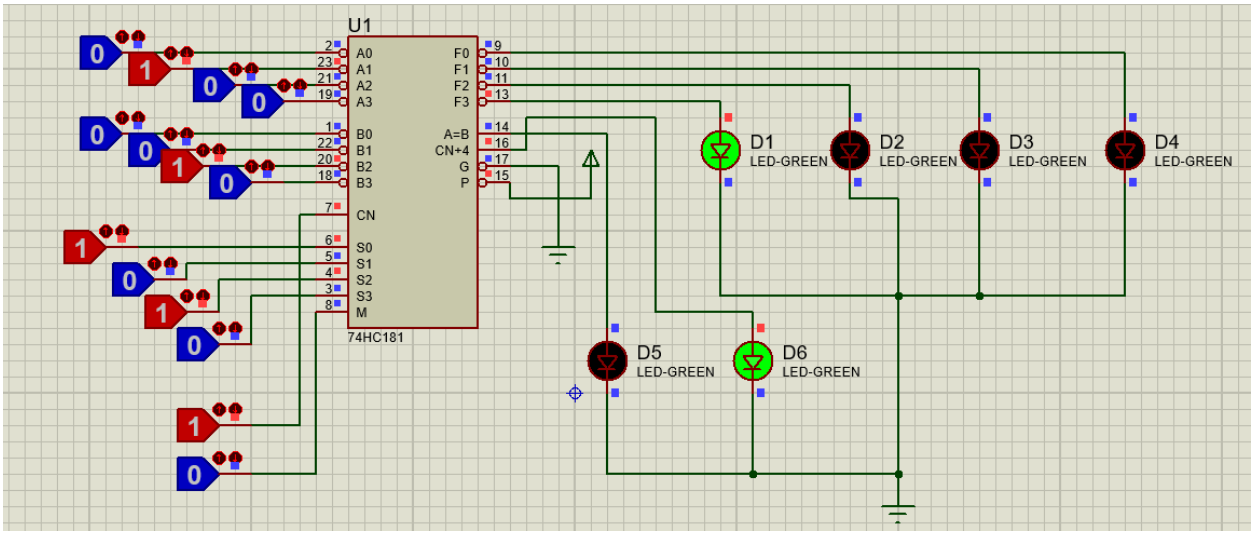
- ورودی 0100:
- Arithmetic:



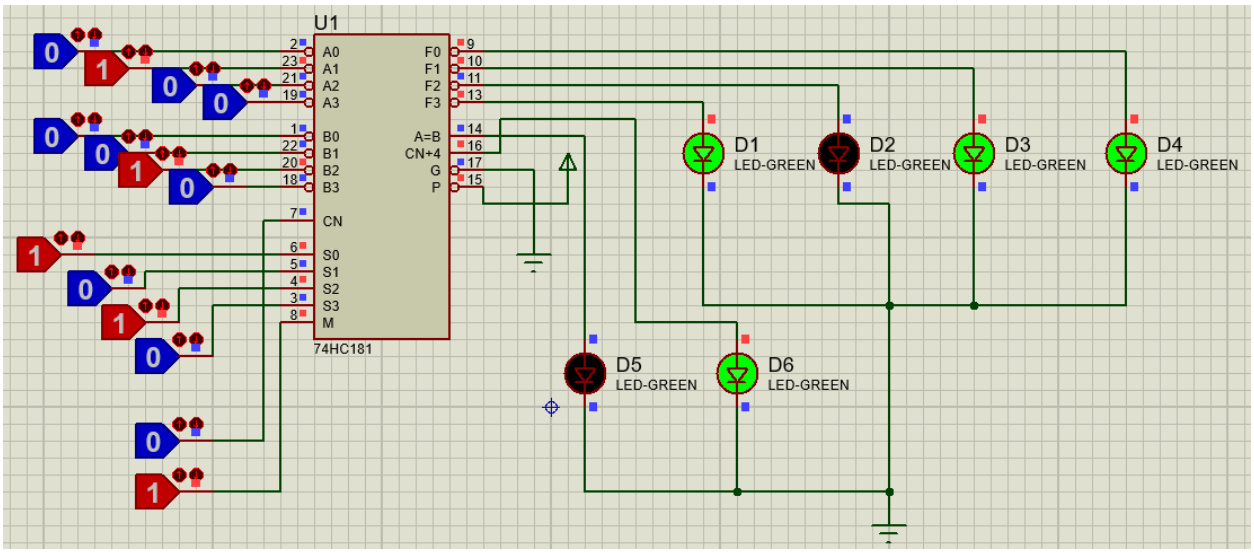
- Logic :



- ورودی 0101:
- Arithmetic :

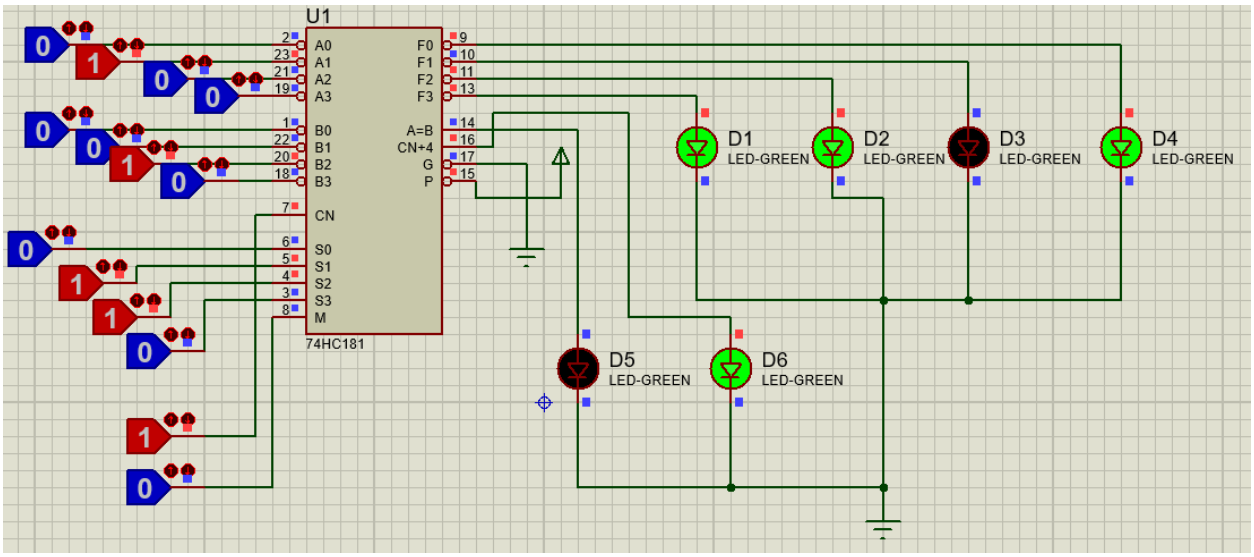


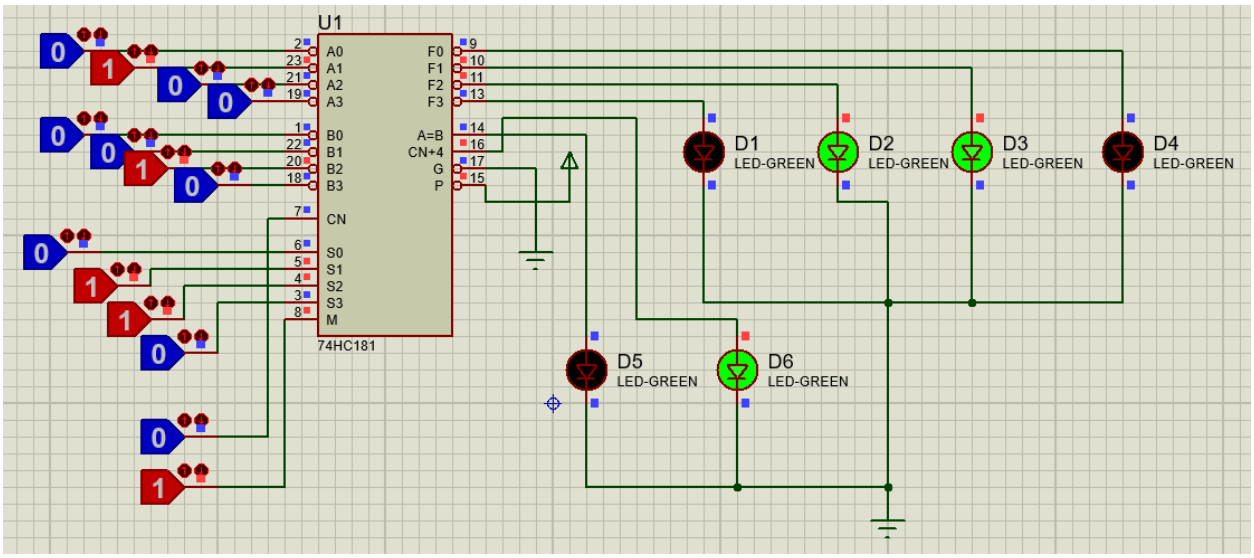
- Logic :

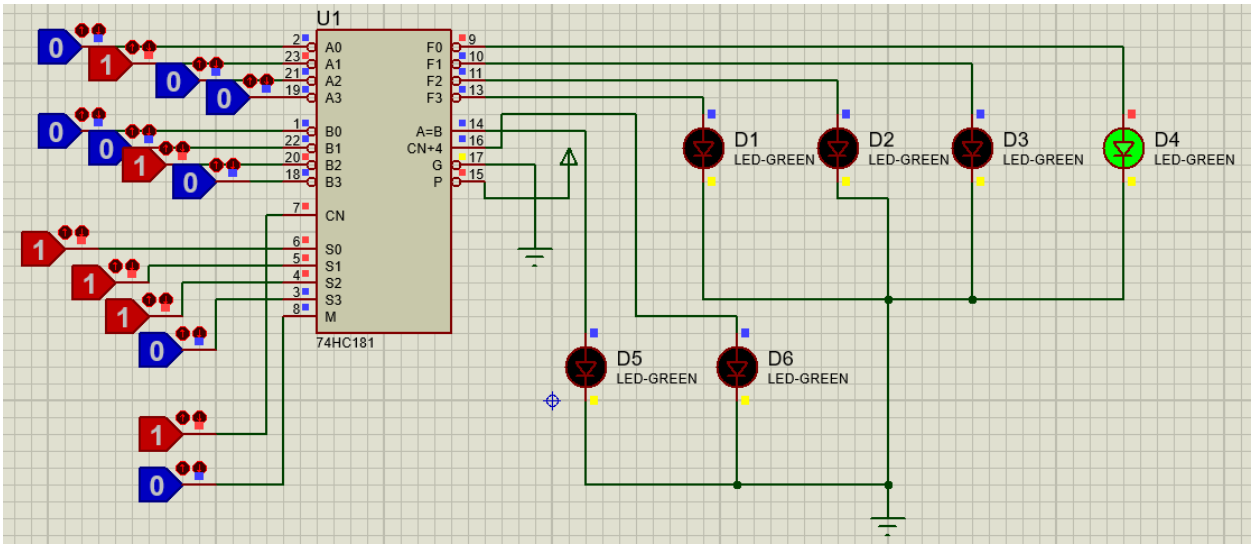


- ورودی 0110:

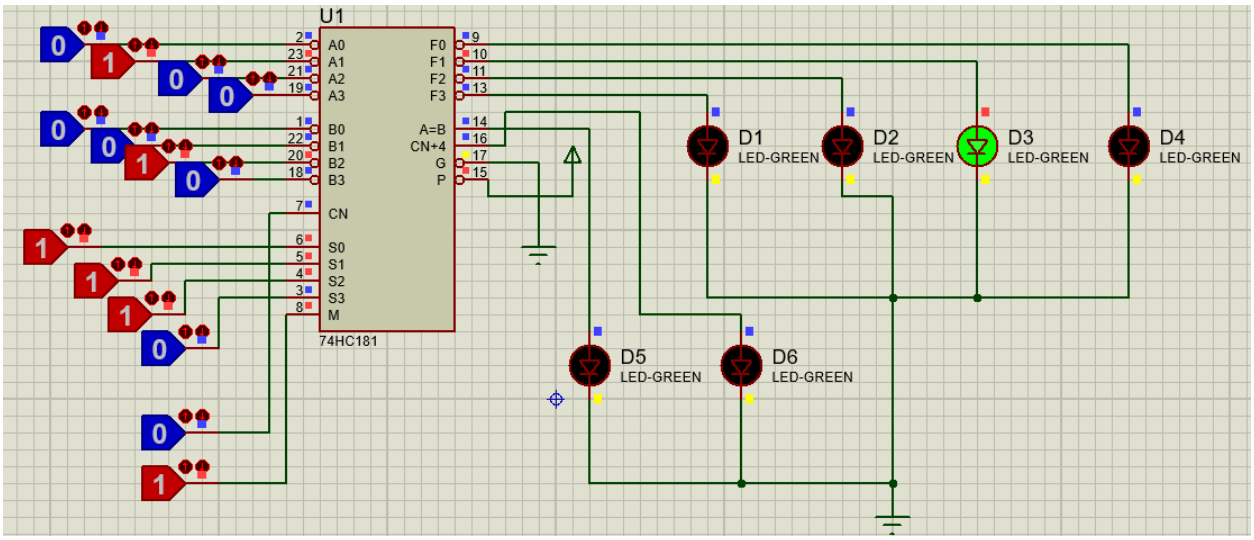
- Arithmetic :



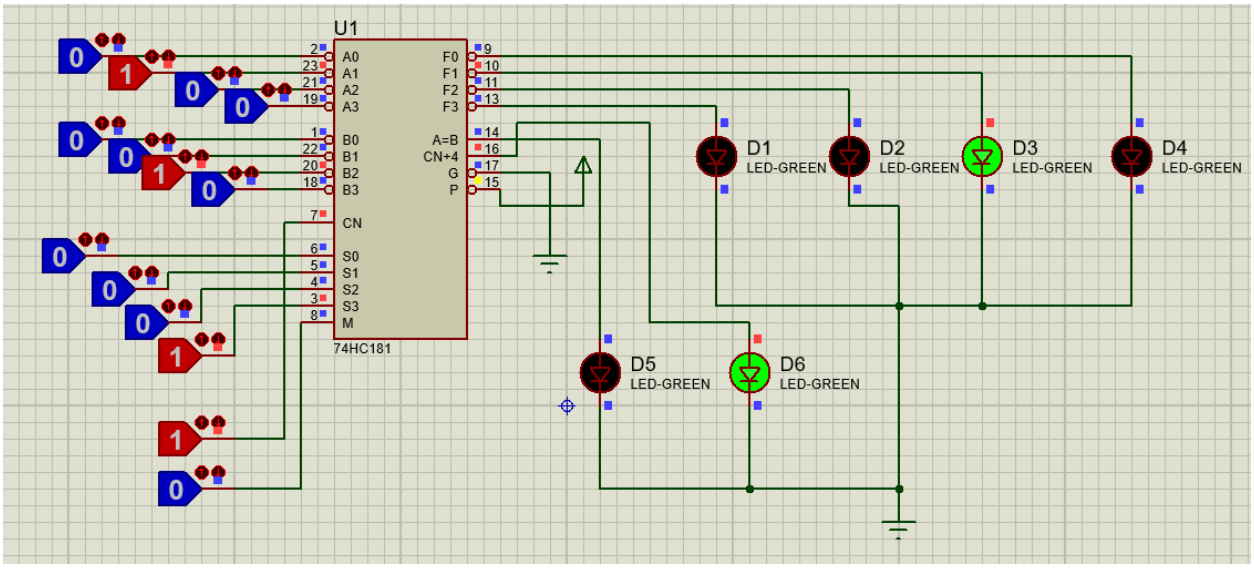
- Logic :


- ورودی 0111:
- Arithmetic :


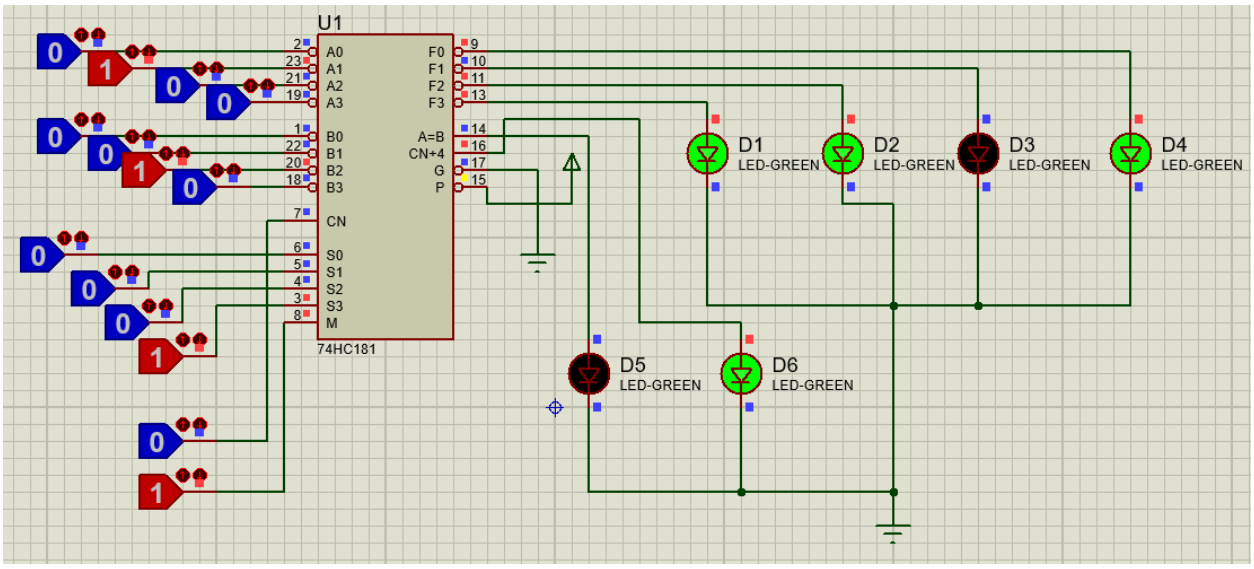
- Logic:



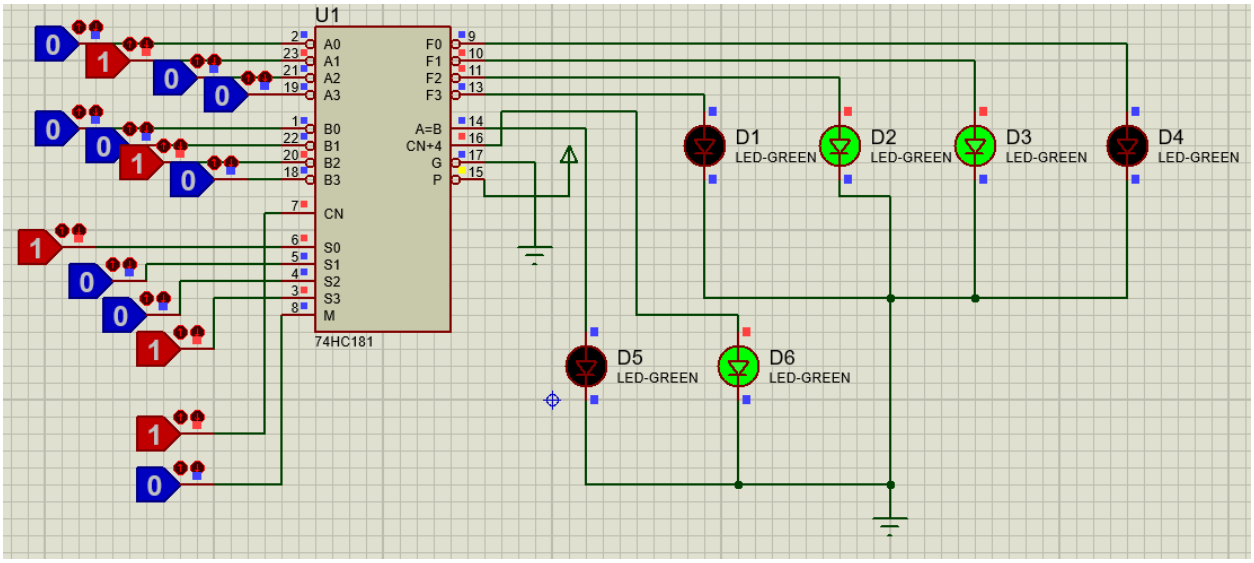
- ورودی 1000:
- Arithmetic:



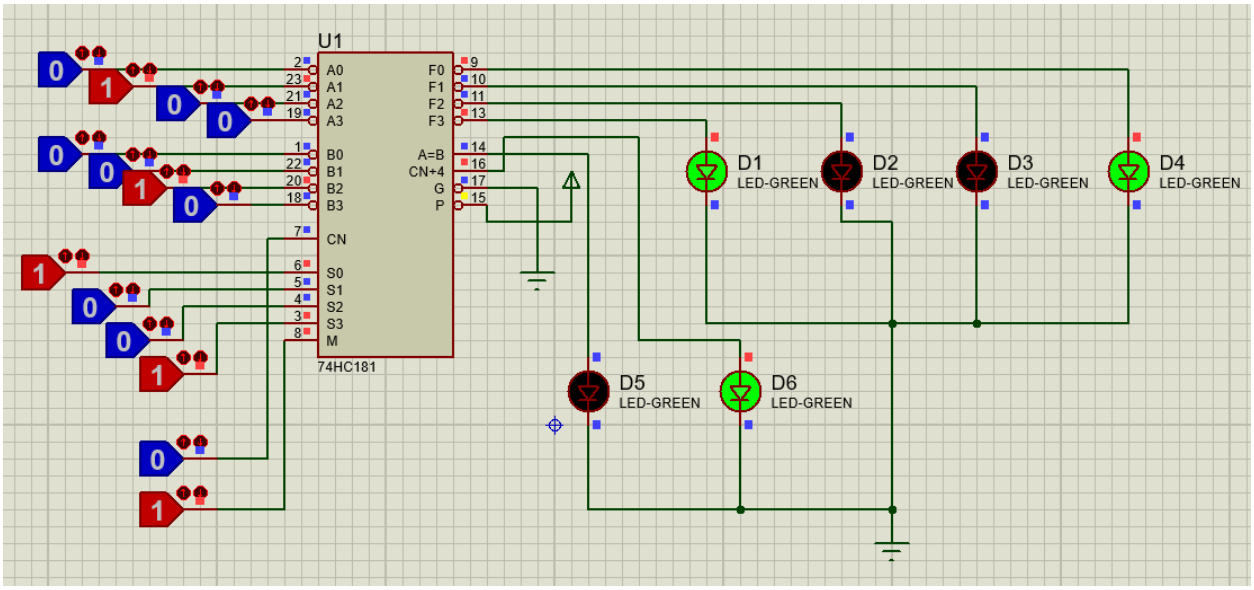
- Logic :



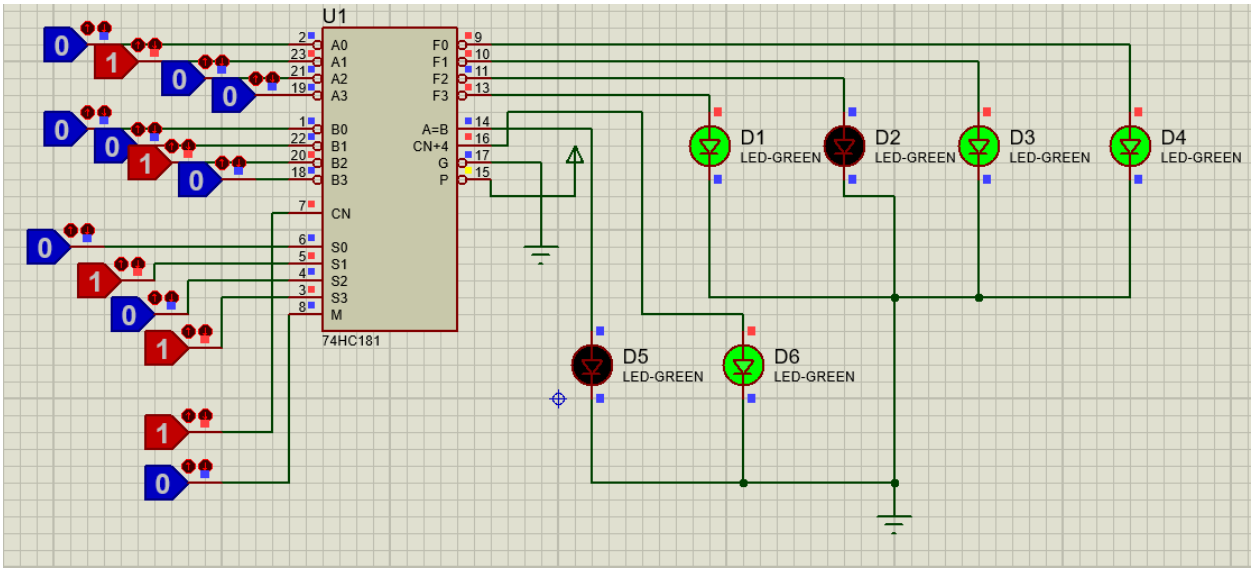
- ورودی 1001:
- Arithmetic :



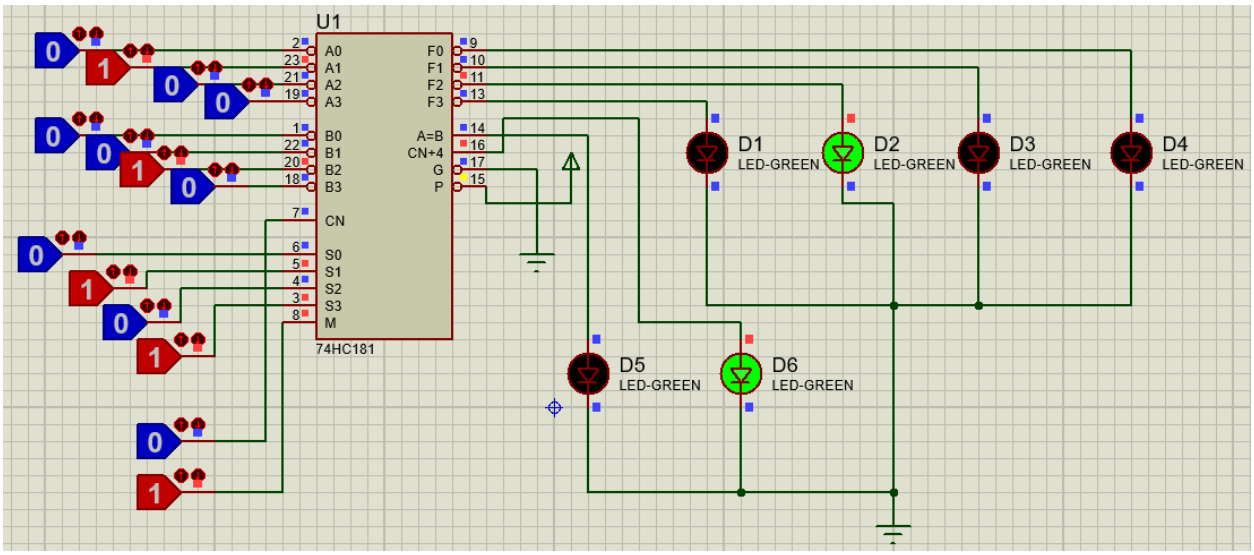
- Logic:



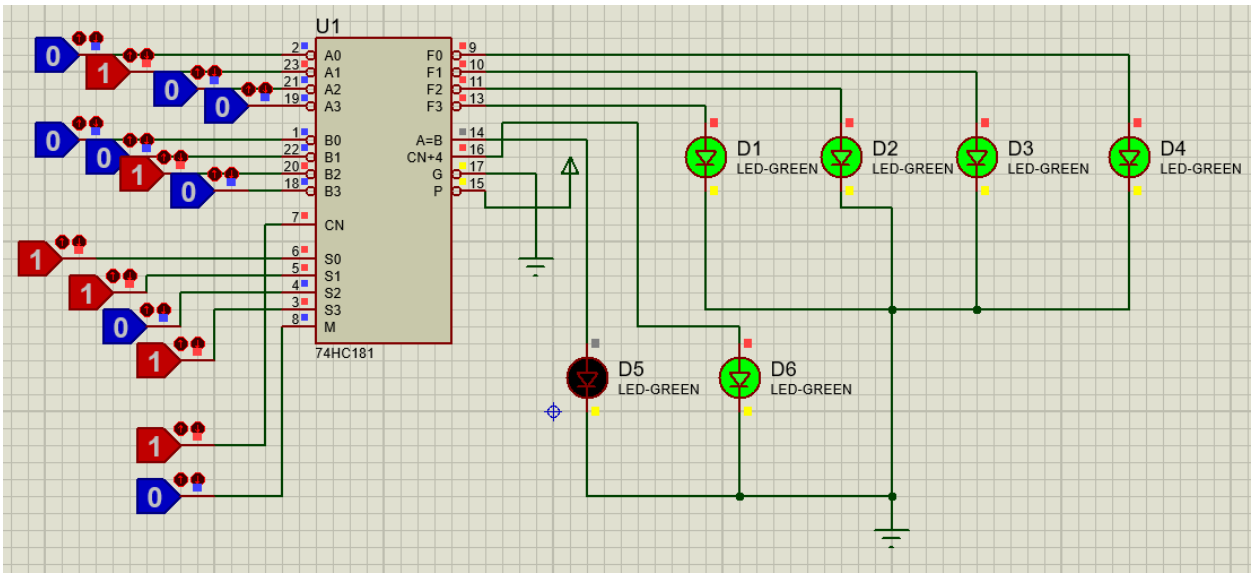
- ورودی 1010:
- Arithmetic:



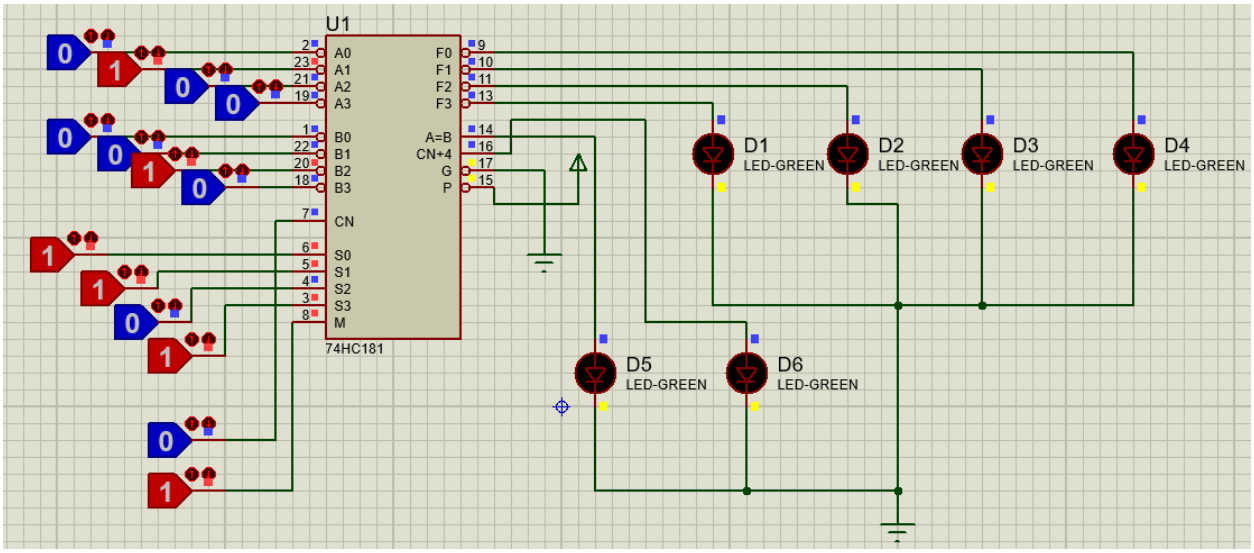
- Logic :



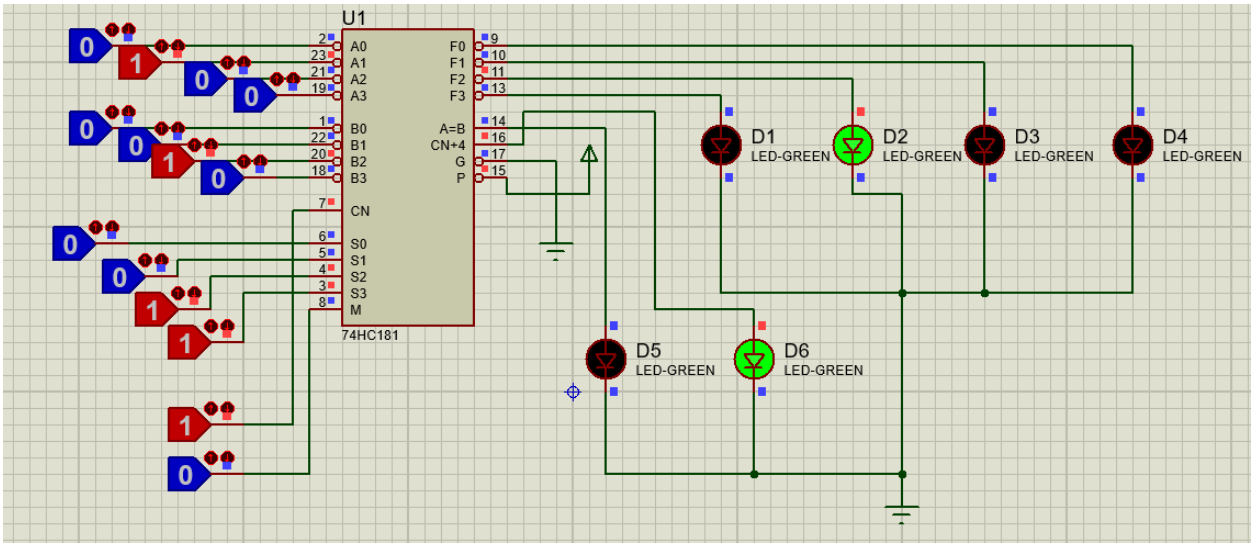
- ورودی 1011:
- Arithmetic :

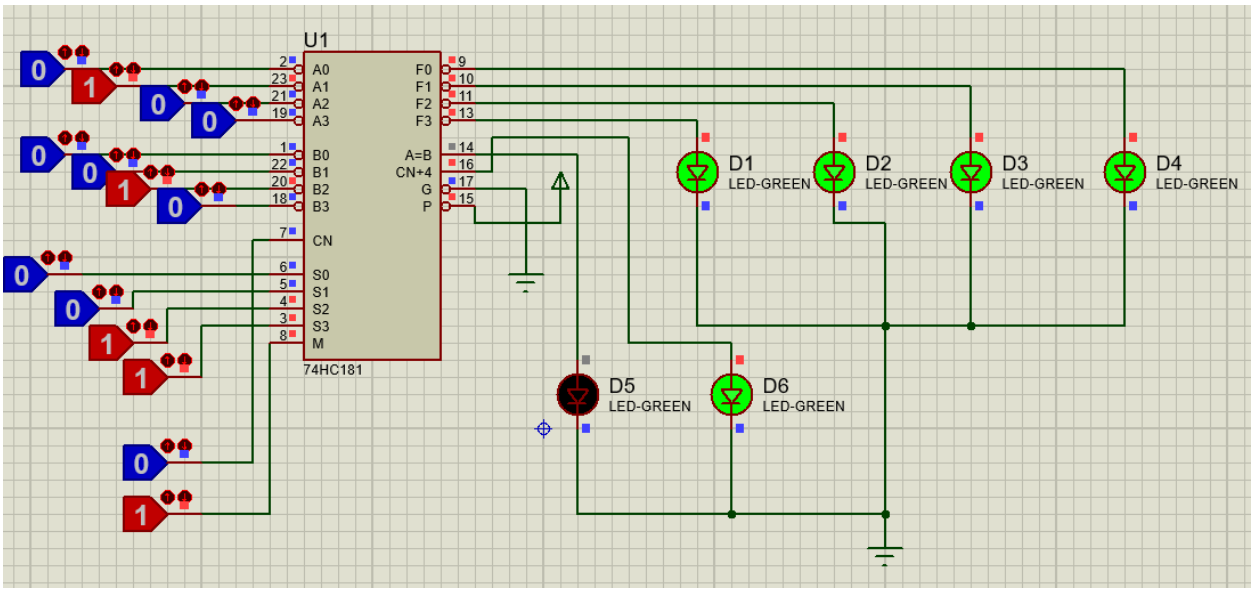


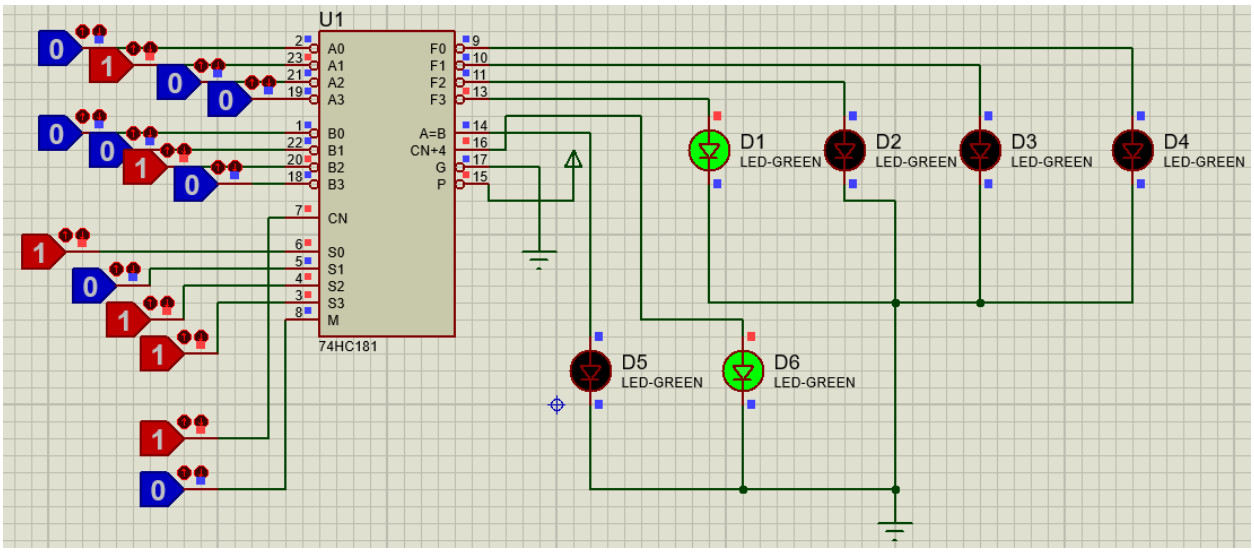
- Logic:



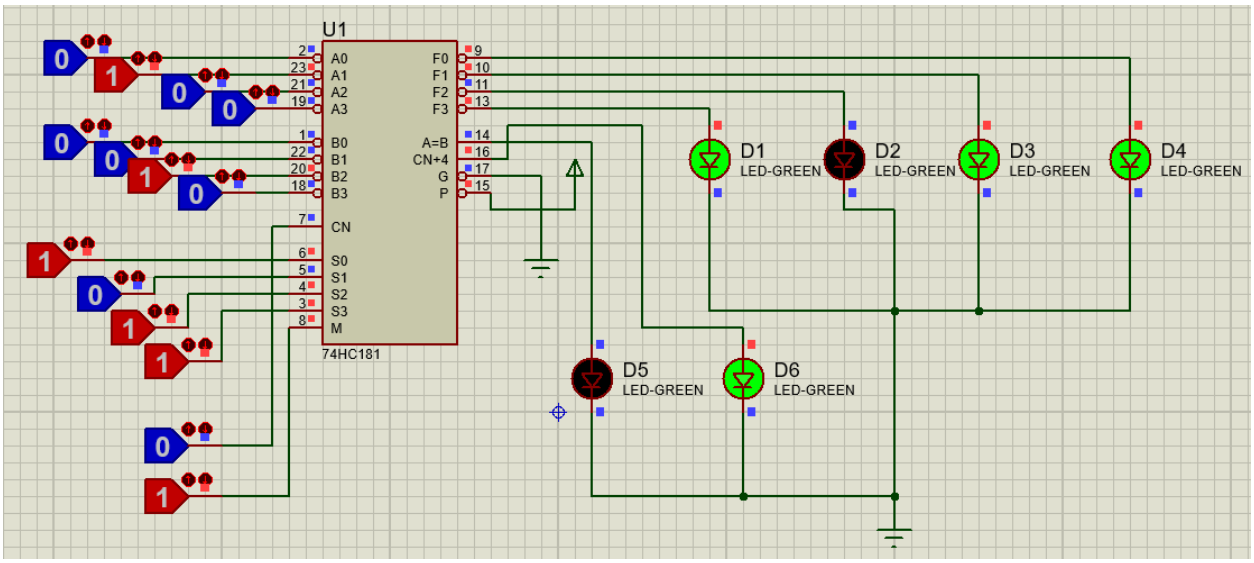
- ورودی 1100:
- Arithmetic:



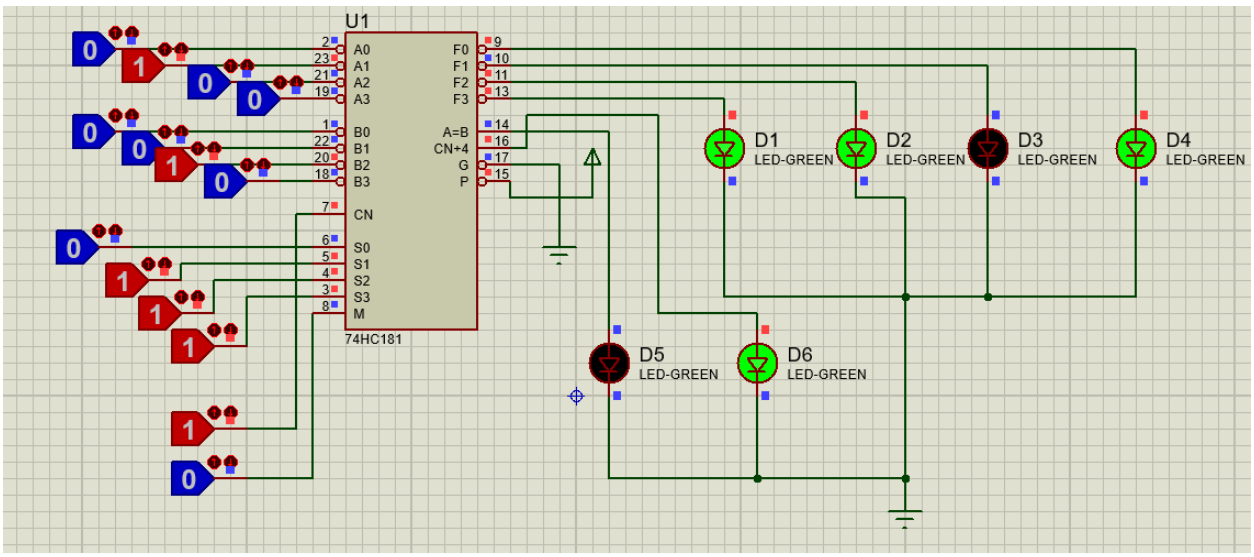
- Logic :
  - 

- ورودی 1101:
  - Arithmetic :
    - 

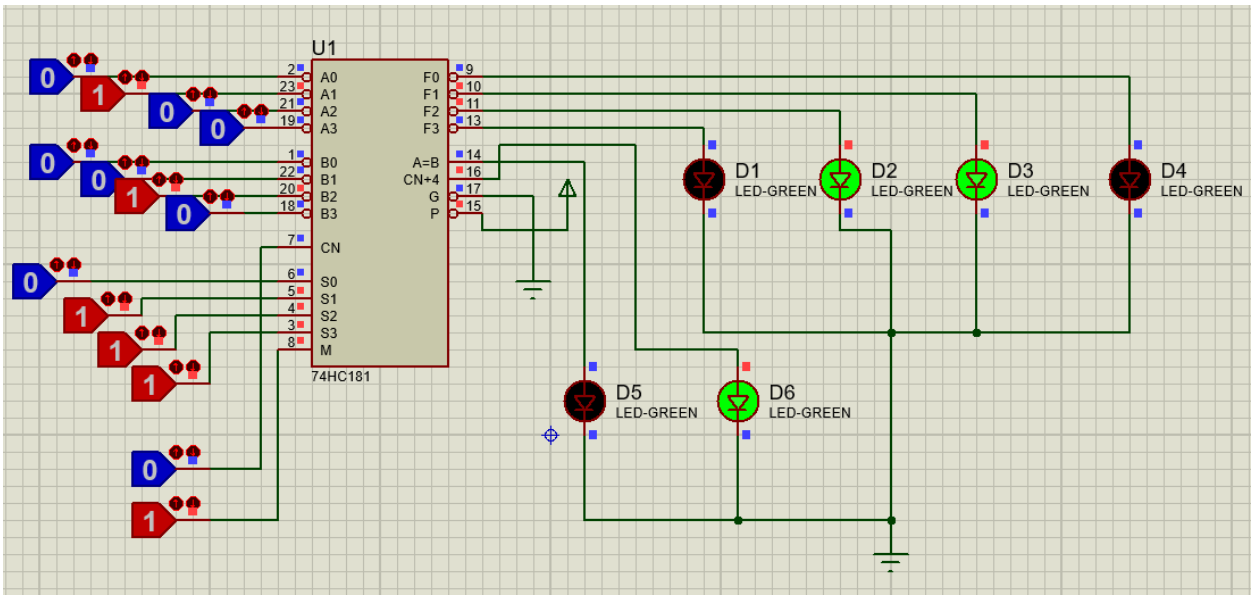
- Logic :



- ورودی 1110:
- Arithmetic :

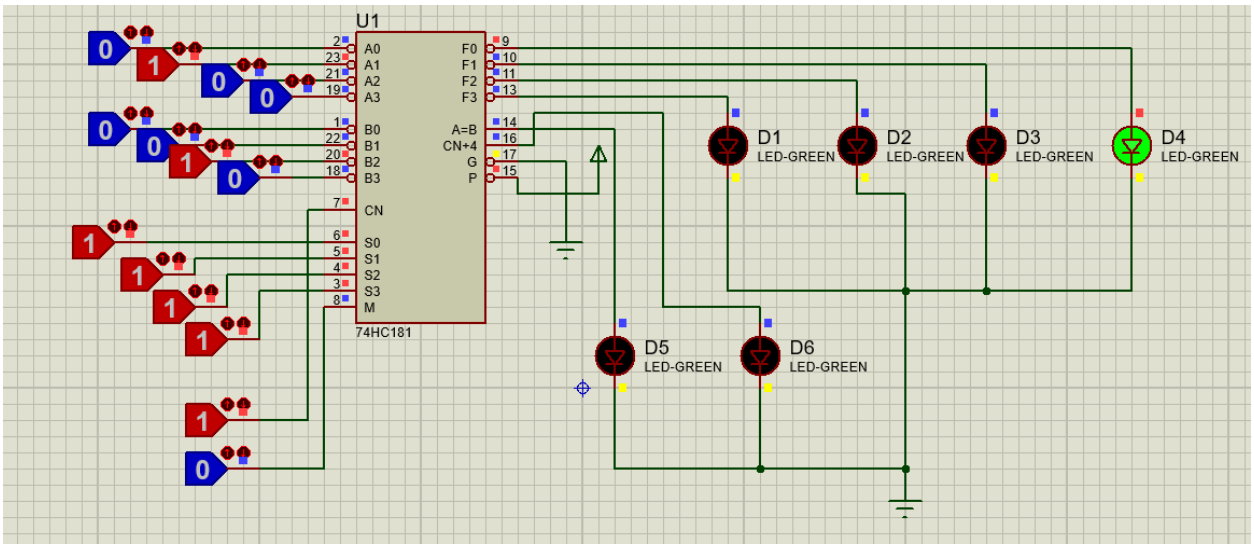


Logic :○

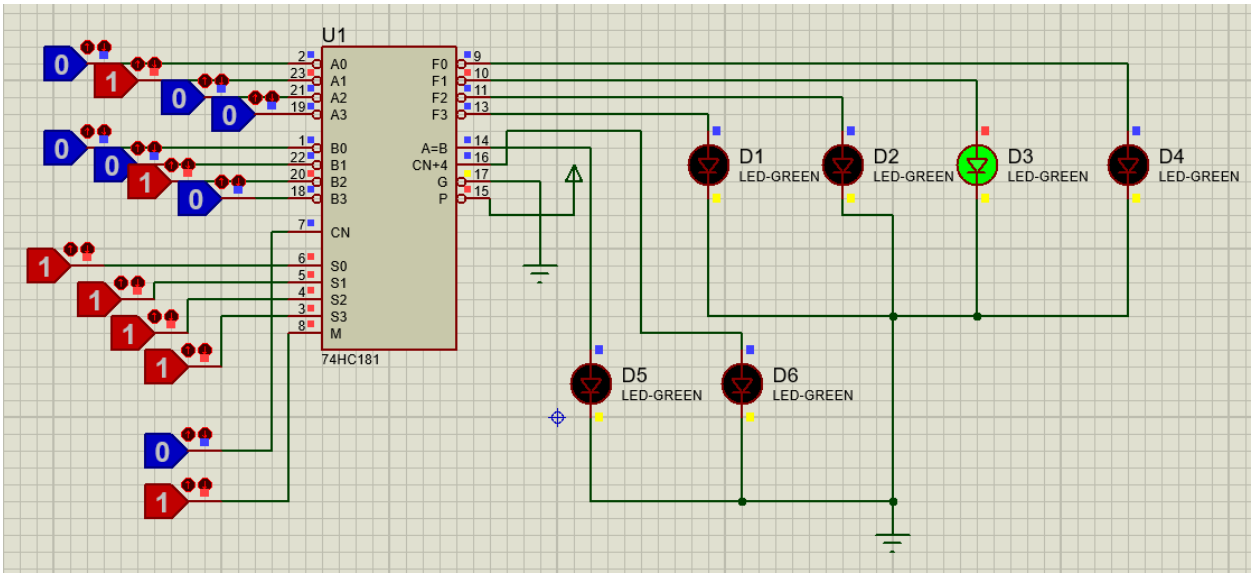


- ورودی 1111:

Arithmetic :○



Logic ○


U1
A0
A1
A2
A3
B0
B1
B2
B3
CN
S0
S1
S2
S3
M
F0
F1
F2
F3
A=B
CN+4
G
P
74HC181
D1
LED-GREEN
D2
LED-GREEN
D3
LED-GREEN
D4
LED-GREEN
D5
LED-GREEN
D6
LED-GREEN

گزارش بستن مدار alu (تراشه 74181 LS):

در تسک بعدی‌ای که انجام دادیم، سپس به سراغ پیاده‌سازی مداری رفتیم که در آن از تراشه 74181 LS استفاده شده بود. در این مرحله، ابتدا مطابق با آزمایش‌های قبلی، بردبورد را متصل کردیم و منبع تغذیه و ولتاژ GND را به آن متصل کردیم، همینطورولتمتر را نیز متصل کردیم. سپس به تعداد چهار تا LED برداشتیم و متصل کردیم به خروجی‌ها تا در صورت یک شدن هر خروجی، متوجه یک شدن آن بشویم و همچنین قبل از متصل کردن خروجی‌ها به LEDها، یک مقاومت قرار دادیم تا LEDها آسیب نبینند. سپس پس از متصل کردن مدار، آن را برای دو تا ورودی مختلف تست کردیم و نتیجه را چک کردیم که درست بود.

آموخته‌های این جلسه:

در این جلسه نحوه کارکردن با تراشه‌های سطح بالاتری مانند ALU، یادگرفیتم و یادآوری شد. سپس استفاده از شیت‌های هرکدام از این ماژول‌ها و نحوه عملکردشان آشنا شدیم. همچنین استفاده از المان‌های مختلف در پروتئوس را یادگرفتیم و نحوه متصل کردن آن را نیز از روی شیت متناظر با آن دیدیم. همچنین در این جلسه نحوه استفاده از ورودی داینامیک در پروتئوس (ورودی‌های logic state) را یاد گرفتیم و در مدار از آن استفاده کردیم و کار را به مقدار خیلی زیادی راحت تر کردیم.