

هم طراحی سخت افزار- نرم افزار

HW/SH Codesign

طراحی پردازنده 6 بیتی

استاد : دکتر صالحی

اعضا گروه :

یاسان حسن زاد 980122680044

سروش قلی زاده 980122680095

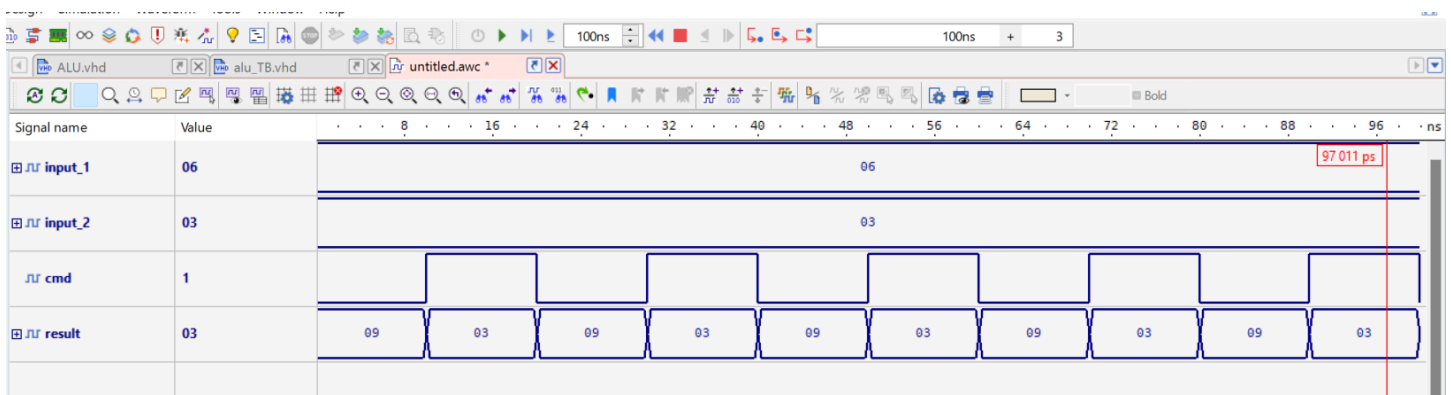
- [ALU](#)
- [IR PC](#)
- [MainRegisters](#)
- [MUX](#)
- [ROM](#)
- [ControlUnit](#)
- [Main](#)(خروجی اصلی خواسته شده در پروژه)
- [Assembler](#)

ALU

Alu دارای دو ورودی 6 بیتی ، یک ورودی 2 بیتی (cmd) select ، یک خروجی 6 بیتی است .

```
33 port(  
34     input_1,input_2 : in std_logic_vector(5 downto 0);  
35     cmd : in std_logic;  
36     result : out std_logic_vector(5 downto 0)  
37 );  
38 end ALU;  
39  
40 architecture ALU of ALU is  
41     signal aluresult : unsigned(5 downto 0);  
42     signal a,b : unsigned(5 downto 0);  
43  
44     begin  
45         a <= unsigned(input_1);  
46         b <= unsigned(input_2);  
47  
48         process(cmd,a,b)  
49             begin  
50                 case cmd is  
51                     when '0' => aluresult <= a + b;  
52                     when '1' => aluresult <= a - b;  
53                     when others => aluresult <= (others => 'X');  
54                 end case;  
55             end process;  
56  
57     result <= std_logic_vector(aluresult);  
58  
59 end ALU;
```

در کد بالا مشاهده میکنیم که هر زمان که cmd صفر باشد alu ورودی ها رو با هم جمع میکنه و هر وقت یک باشه اونارو از هم کم میکنه .



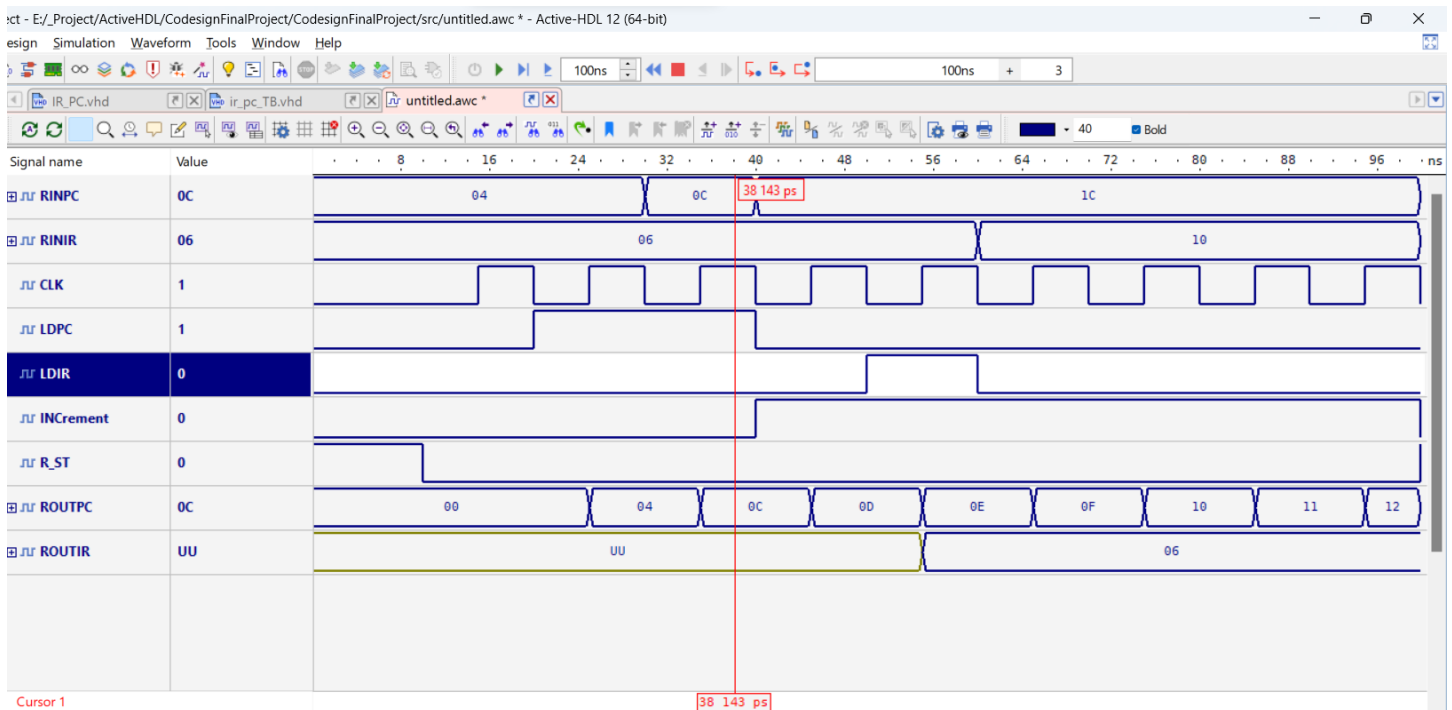
Testbench بالا هم نشون میده که دو ورودی 6 و 3 در صورتی که cmd یک باشه از هم کم میشن (خروجی 3 مشه) و اگه cmd صفر باشه با هم جمع میشن (خروجی 9 میشه).

IR_PC

registerIR دارای یک ورودی 6 بیتی RINIR و ورودی های clk و loadIR و یک خروجی 6 بیتی ROUTIR است .

```
48 ----- IR register -----
49
50
51 IR: process(CLK)
52 begin
53     if(rising_edge(CLK)) then
54         if(LDIR='1') then
55             ROUTIR <= RINIR ;
56         end if;
57     end if;
58 end process IR;
59
```

همانطور که در کد بالا مشاهده میکنید هر زمان که CLK این ثبات به لبه بالا رونده خودش رسید و load فعال بود ورودی و میریزه تو خروجی .



با توجه به TestBench شکل بالا ورودی ما که 12 بود با فعال شدن load تو لبه بالا رونده ، خروجی هم 12 میشه .

RegisterPC دارای یک ورودی 6 بیتی RINPC و ورودی های Clk ، LoadPC ، Increment و Reset و یک خروجی 6 بیتی ROUTPC است.

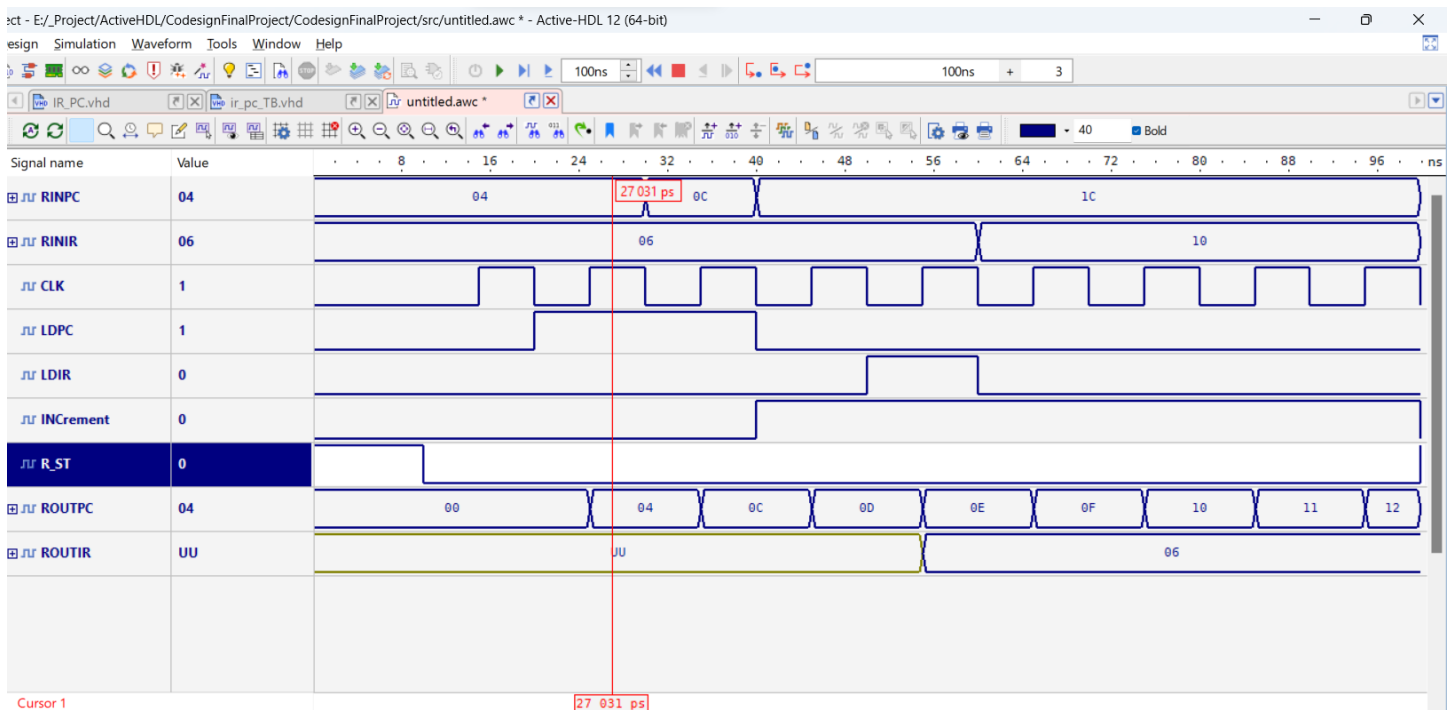
```

59 ----- PC register -----
60
61
62 PC: process(CLK,R_ST)
63 begin
64
65     if(R_ST='1') then
66         pc1 <= (others => '0');
67     elsif(rising_edge(CLK)) then
68         if(LDPC='1') then
69             pc1 <= RINPC;
70             end if;
71             if(INCremen='1') then
72                 pc1 <= pc1 + 1;
73             end if;
74         end if;
75     end process PC;
76     ROUTPC <= pc1;
77

```

همانطور که در کد بالا مشاهده میکنید اگر Reset یک باشد ورودی صفر به خروجی میدیم و اگر CLK این ثبات به لبه ی بالا رونده ی خودش رسید و load فعال بود ورودی رو به خروجی انتقال می دیم و اگر لبه ی بالارونده بود و Increment فعال بود ، به ورودی یکی اضافه میکنه و به خروجی انتقال می ده .

نتیجه رو تو Testbench پایین میبینید :



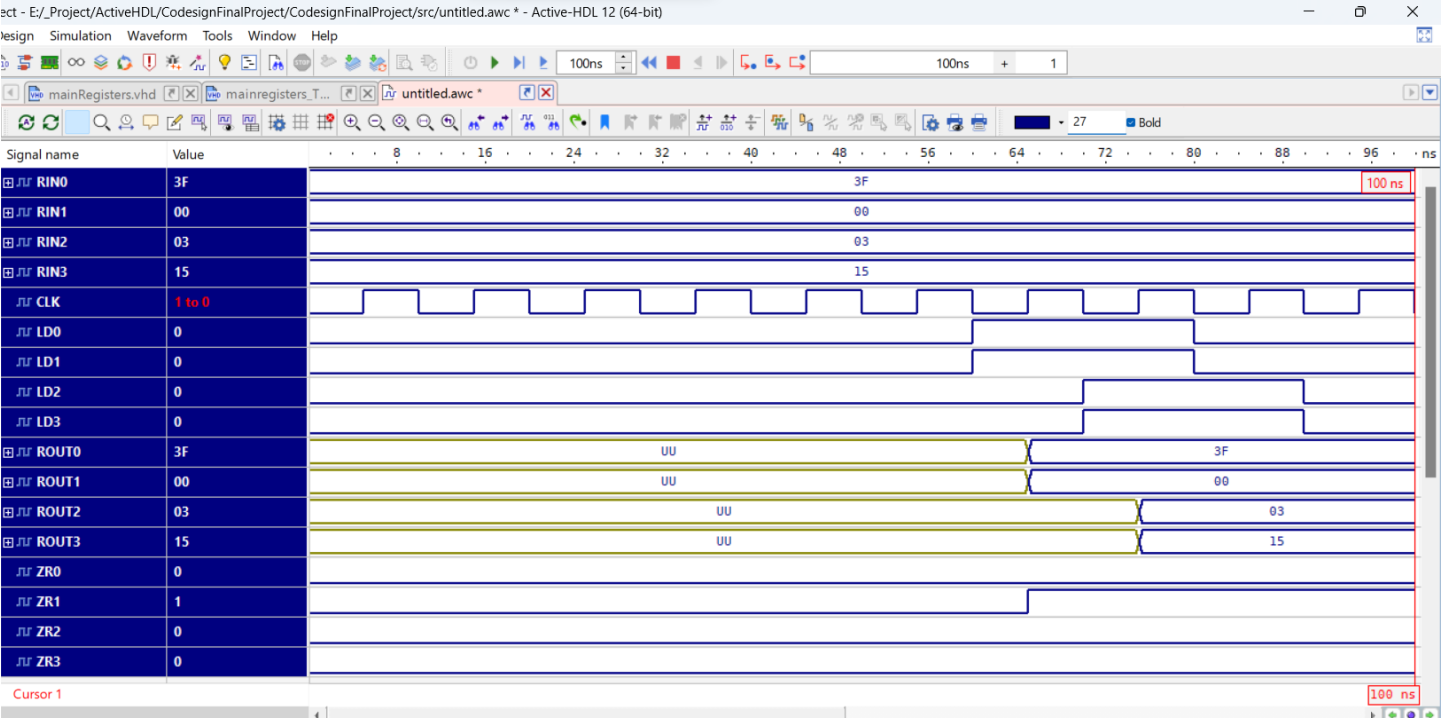
MainRegisters

Mainregisters دارای 4 ثبات اصلی ، که دارای یک ورودی 6 بیتی RIN و ورودی های Clk و Load و خروجی های 6 بیتی ROUT و ZR است .

```
----- MAIN registers -----  
RN0 <= RIN0 when (LD0='1') else R_0;  
RN1 <= RIN1 when (LD1='1') else R_1;  
RN2 <= RIN2 when (LD2='1') else R_2;  
RN3 <= RIN3 when (LD3='1') else R_3;  
  
MAIN: process(CLK)  
begin  
    if(rising_edge(CLK)) then  
        R_0 <= RN0;  
        R_1 <= RN1;  
        R_2 <= RN2;  
        R_3 <= RN3;  
    end if;  
end process MAIN;  
  
ROUT0 <= R_0;  
ROUT1 <= R_1;  
ROUT2 <= R_2;  
ROUT3 <= R_3;  
  
ZR0 <= '1' when (R_0="000000") else '0';  
ZR1 <= '1' when (R_1="000000") else '0';  
ZR2 <= '1' when (R_2="000000") else '0';  
ZR3 <= '1' when (R_3="000000") else '0';  
  
end mainregisters;
```

همانطور که در کد بالا مشاهده میکنید اگر Load یک باشد و لبه ی بالا رونده ی کلاک باشد ورودی رو به خروجی انتقال میدهیم و اگر ZR ثبات ها فعال باشد ، به ثبات ها خروجی صفر می دهد.

نتیجه رو تو Testbench صفحه بعد میبینید :



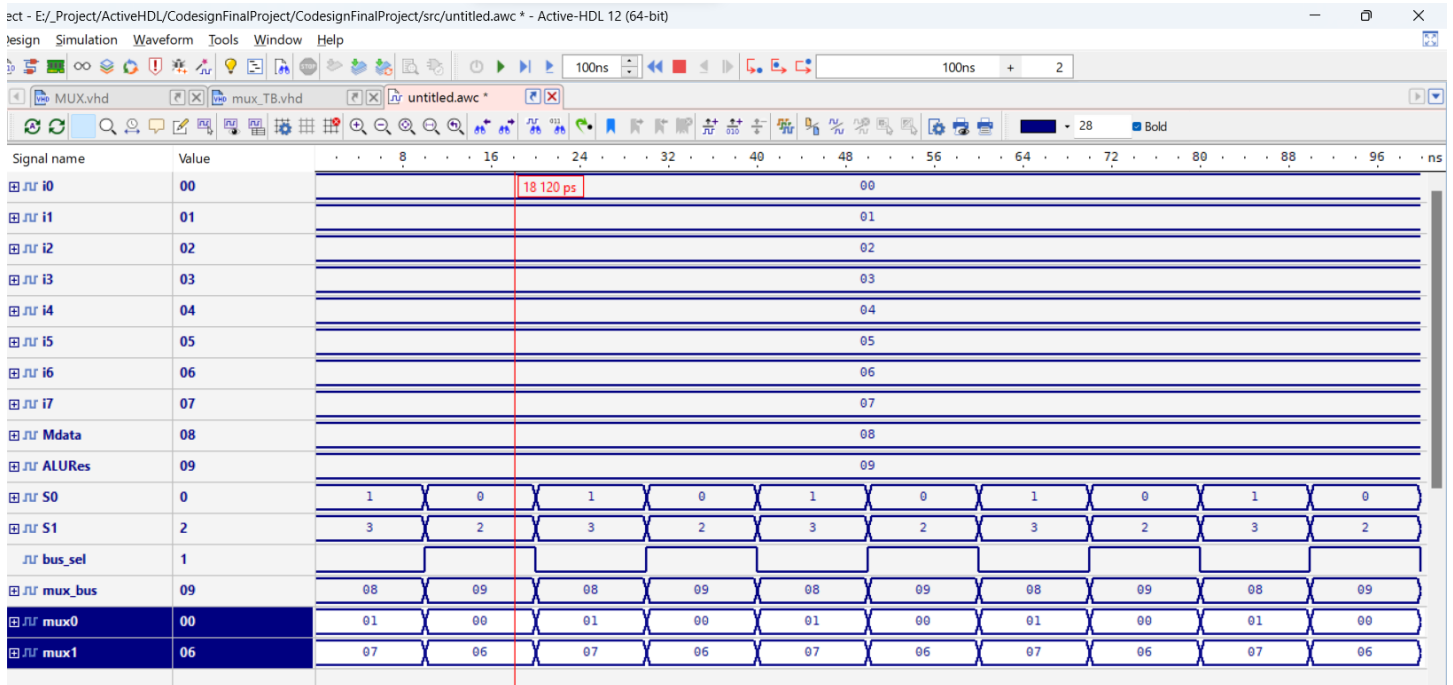
MUX

دو عدد **MUXALU** داریم که دارای چهار ورودی 6 بیتی و یک خروجی 6 بیتی و دو خط select 2 بیتی است .

```
----- alu muxes -----  
M_0: process(S0,i0,i1,i2,i3)  
begin  
  case S0 is  
    when "00" => mux0 <= i0;  
    when "01" => mux0 <= i1;  
    when "10" => mux0 <= i2;  
    when "11" => mux0 <= i3;  
    when others => mux0 <= (others => 'X');  
  end case;  
end process M_0;  
  
M_1: process(S1,i4,i5,i6,i7)  
begin  
  case S1 is  
    when "00" => mux1 <= i4;  
    when "01" => mux1 <= i5;  
    when "10" => mux1 <= i6;  
    when "11" => mux1 <= i7;  
    when others => mux1 <= (others => 'X');  
  end case;  
end process M_1;
```

همانطور که در کد بالا مشاهده میکنید ماکس اول با توجه ب خط select خود ورودی را به خروجی منتقل میکند و در ماکس دوم نیز به هم صورت با توجه به select دوم مقدار ورودی را به خروجی ماکس دوم منتقل میکند .

نتیجه رو تو Testbench صفحه بعد میبینید :



Muxbus دارای 2 ورودی mdata و ALUres و یک خروجی bus می باشد.

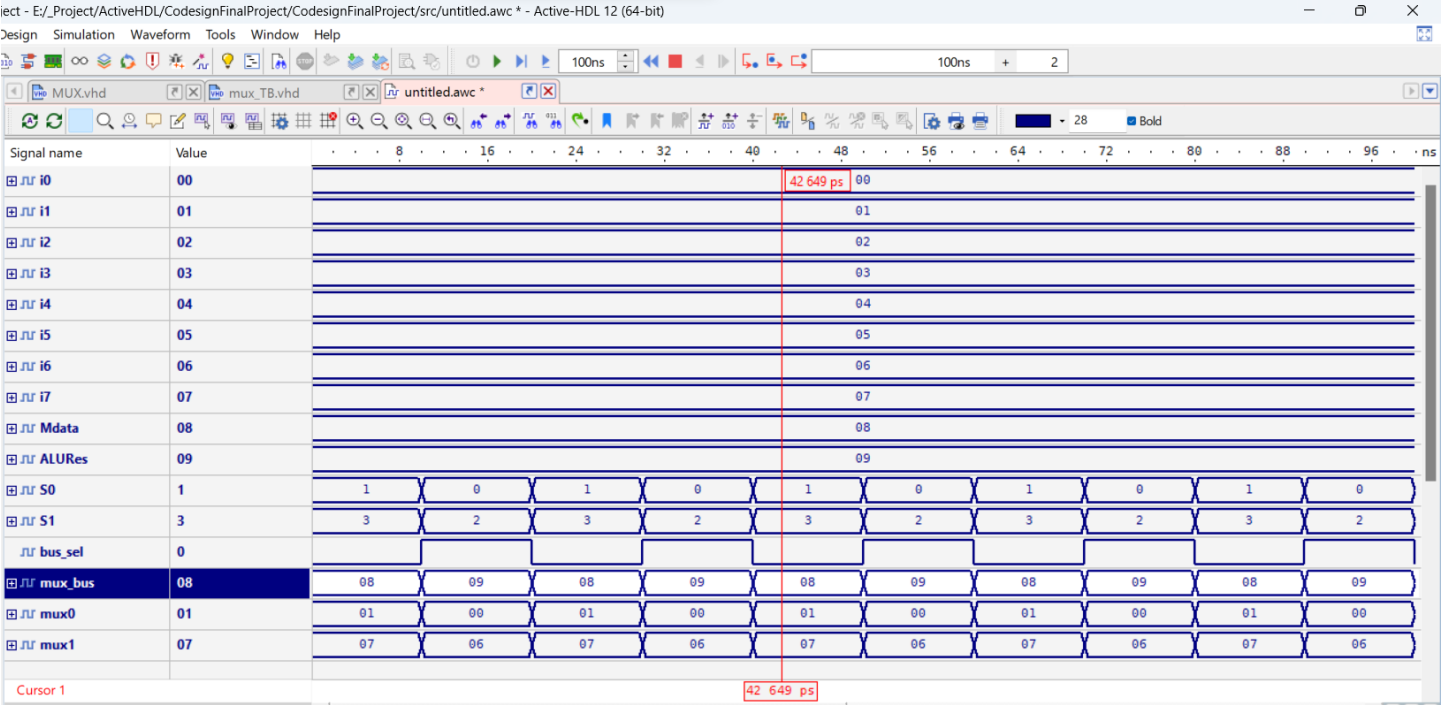
```

----- bus mux -----
MUXBUS: process(bus_sel,Mdata,ALUres)
begin
  case bus_sel is
    when '0' => mux_bus <= Mdata;
    when '1' => mux_bus <= ALUres;
    when others => mux_bus<= ('X');
  end case;
end process MUXBUS;

```

همانطور که در کد بالا مشاهده میکنید اگر bus_sel صفر باشد ، مقدار Mdata را در خروجی ماکس می ریزیم و اگر bus_sel یک باشد ، مقدار ALUres را در خروجی قرار می دیم .

نتیجه رو تو Testbench صفحه بعد میبینید :

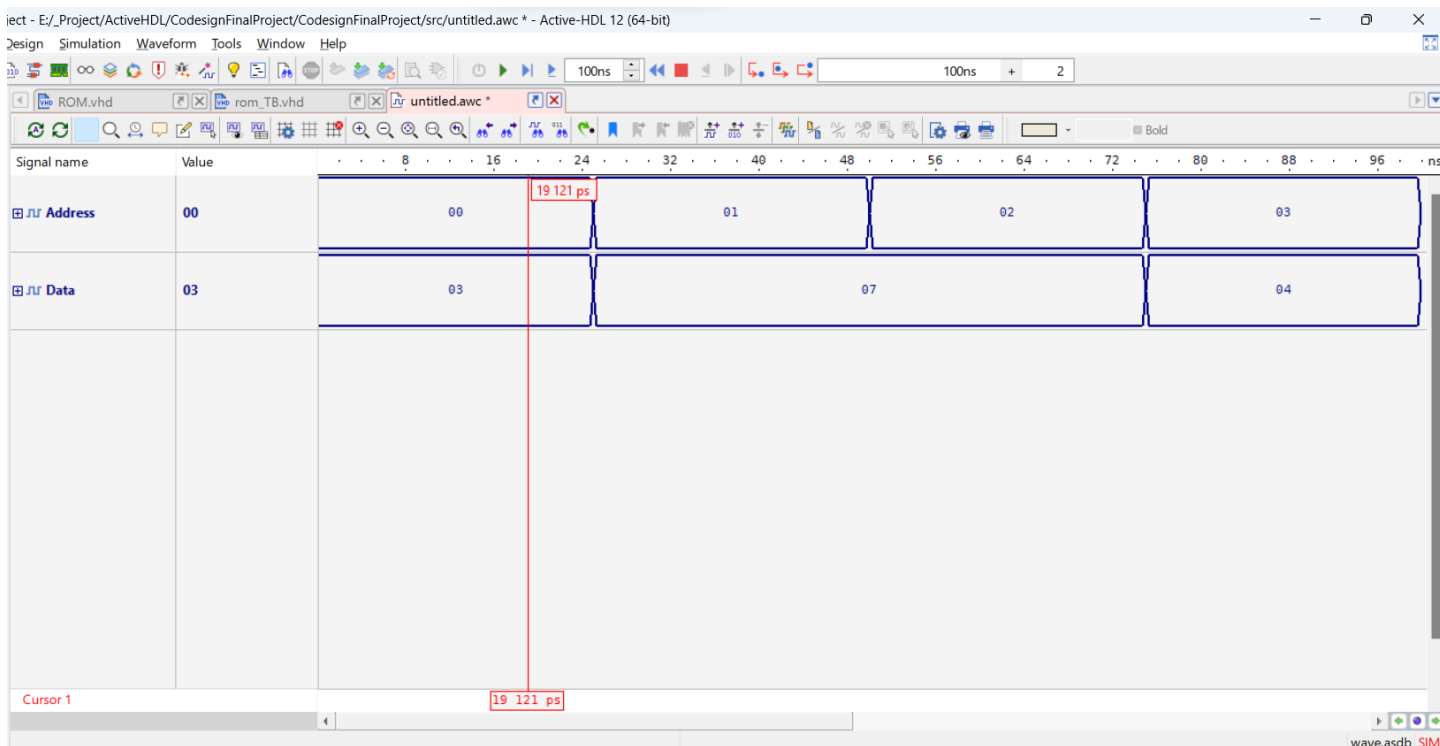


ROM

حافظه از نوع ROM و دارای یک ورودی 6 بیتی Address و یک خروجی 6 بیتی Data هست .

```
37
38 type ROM_type is array(0 to 63) of std_logic_vector(5 downto 0);
39 signal m : ROM_type;
40
41 begin
42 m(0) <= "000011" ;
43 m(1) <= "000111" ;
44 m(2) <= "000111" ;
45 m(3) <= "000100" ;
46 m(4) <= "010001" ;
47 m(5) <= "000000" ;
48
49 Data <= m(to_integer(unsigned(Address)));
50
```

همانطور که در کد بالا مشاهده میکنید یک پهنا و عمق به حافظمون اختصاص میدیم و یکی یکی به مقدار دیتا مورد نظرمون آدرس می دهیم .



با توجه به TestBench شکل بالا آدرس خانه اول مقدار 3 را نشان می دهد و به ترتیب بقیه خانه های آدرس مقدار بقیه دیتا ها را نمایش میدهند.

ControlUnit

CONTROLUNIT دارای چهار ورودی 6 بیتی ZR ، یک ورودی CLK ، و 6 بیتی ROUTIR و 6 خروجی LD ریجسترها ، خروجی selectBus ، selects ، cmd ، rst ، increment ، است .

```
entity ControlUnit is
    port(
        clock,rst: in std_logic;
        ZR0,ZR1,ZR2,ZR3 : in std_logic;
        ROUT_IR : in std_logic_vector(5 downto 0);
        LD0,LD1,LD2,LD3 : out std_logic;
        LD_PC , LD_IR :out std_logic;
        Sel0: out std_logic_vector(1 downto 0);
        Sel1: out std_logic_vector(1 downto 0);
        BusSelect , ALU_CMD : out std_logic;
        INC , CLR : out std_logic
    );
end ControlUnit;
```

در ادامه تمام state ها به تفکیک مشخص شده اند :

```
78 ----- s0 -----
79 process(PSTATE,ROUT_IR,ZR0,ZR1,ZR2,ZR3)
80     begin
81         case PSTATE is
82             when S0 => CLR <= '1';
83                 LD_IR <= '0';
84                 INC <= '0';
85                 LD_PC <= '0';
86                 BusSelect <= '0';
87                 LD0 <= '0';
88                 LD1 <= '0';
89                 LD2 <= '0';
90                 LD3 <= '0';
91                 LD_PC <= '0';
92                 BusSelect <= '0';
93                 ALU_CMD <= '0';
94                 Sel0 <= "00";
95                 Sel1 <= "00";
96                 NSTATE <= S1;
97 ----- s1 -----
98             when S1 => LD_IR <= '1';
99                 INC <= '1';
100                 LD_PC <= '0';
101                 BusSelect <= '0';
102                 CLR <= '0';
103                 LD0 <= '0';
104                 LD1 <= '0';
105                 LD2 <= '0';
106                 LD3 <= '0';
107                 ALU_CMD <= '0';
108                 Sel0 <= "00";
109                 Sel1 <= "00";
110                 NSTATE <= HLTSTATE;
```

```

111 ----- hltstate -----
112
113 when HLTSTATE => if(ROUT_IR = "000000") then NSTATE <= S2 ;
114 else
115   if(ROUT_IR(5 downto 4) = "00") then
116     NSTATE <= S3;
117   elsif(ROUT_IR(5 downto 4) = "01") then
118     NSTATE <= S4;
119   elsif(ROUT_IR(5 downto 4) = "10") then
120     NSTATE <= S5;
121   elsif(ROUT_IR(5 downto 4) = "11") then
122     if(R(temp) = '1') then
123       NSTATE <= S7;
124     else
125       NSTATE <= S6;
126   end if;
127 end if;
128 end if;
129
130 CLR <= '0';
131 BusSelect <= '0';
132 LD0 <= '0';
133 LD1 <= '0';
134 LD2 <= '0';
135 LD3 <= '0';
136 ALU_CMD <= '0';
137 Sel0 <= "00";
138 Sel1 <= "00";
139 LD_PC <= '0';
140 LD_IR <= '0';
141 INC <= '0';

```

```

142 ----- s2 -----
143
144 when S2 =>
145   CLR <= '0';
146   BusSelect <= '0';
147   LD0 <= '0';
148   LD1 <= '0';
149   LD2 <= '0';
150   LD3 <= '0';
151   ALU_CMD <= '0';
152   Sel0 <= "00";
153   Sel1 <= "00";
154   LD_PC <= '0';
155   LD_IR <= '0';
156   NSTATE <= S2;
157 ----- s3 -----
158
159 when S3 => if(ROUT_IR(3 downto 2) = "00") then LD0 <= '1';
160   elsif(ROUT_IR(3 downto 2) = "01") then LD1 <= '1';
161   elsif(ROUT_IR(3 downto 2) = "10") then LD2 <= '1';
162   elsif(ROUT_IR(3 downto 2) = "11") then LD3 <= '1';
163   end if;
164
165   INC <= '1';
166   LD_PC <= '0';
167   BusSelect <= '0';
168   CLR <= '0';
169   LD_IR <= '0';
170   ALU_CMD <= '0';
171   Sel0 <= "00";
172   Sel1 <= "00";
173   NSTATE <= S1;

```

```

172 ----- s4 -----
173
174 when S4 => Sel0 <= ROUT_IR(3 downto 2);
175     Sel1 <= ROUT_IR(1 downto 0);
176
177     if(ROUT_IR(3 downto 2) = "00") then LD0 <= '1';
178     elsif(ROUT_IR(3 downto 2)= "01") then LD1 <= '1';
179     elsif(ROUT_IR(3 downto 2)= "10") then LD2 <= '1';
180     elsif(ROUT_IR(3 downto 2)= "11") then LD3 <= '1';
181     end if;
182
183     ALU_CMD <= '0';
184     BusSelect <= '1';
185     CLR <= '0';
186     LD_IR <= '0';
187     INC <= '0';
188     LD_PC <= '0';
189     NSTATE <= S1;
190

```

```

191 ----- s5 -----
192
193 when S5 => Sel0 <= ROUT_IR(3 downto 2);
194     Sel1 <= ROUT_IR(1 downto 0);
195
196     if(ROUT_IR(3 downto 2) = "00") then LD0 <= '1';
197     elsif(ROUT_IR(3 downto 2)= "01") then LD1 <= '1';
198     elsif(ROUT_IR(3 downto 2)= "10") then LD2 <= '1';
199     elsif(ROUT_IR(3 downto 2)= "11") then LD3 <= '1';
200     end if;
201
202     ALU_CMD <= '1';
203     BusSelect <= '1';
204     CLR <= '0';
205     LD_IR <= '0';
206     INC <= '0';
207     LD_PC <= '0';
208     NSTATE <= S1;

```

```

209 ----- s6 -----
210
211 when S6 => LD_PC <= '1';
212     INC <= '0';
213     BusSelect <= '0';
214     CLR <= '0';
215     LD0 <= '0';
216     LD1 <= '0';
217     LD2 <= '0';
218     LD3 <= '0';
219     LD_IR <= '0';
220     Sel0 <= "00";
221     Sel1 <= "00";
222     ALU_CMD <= '0';
223     NSTATE <= S1;
224

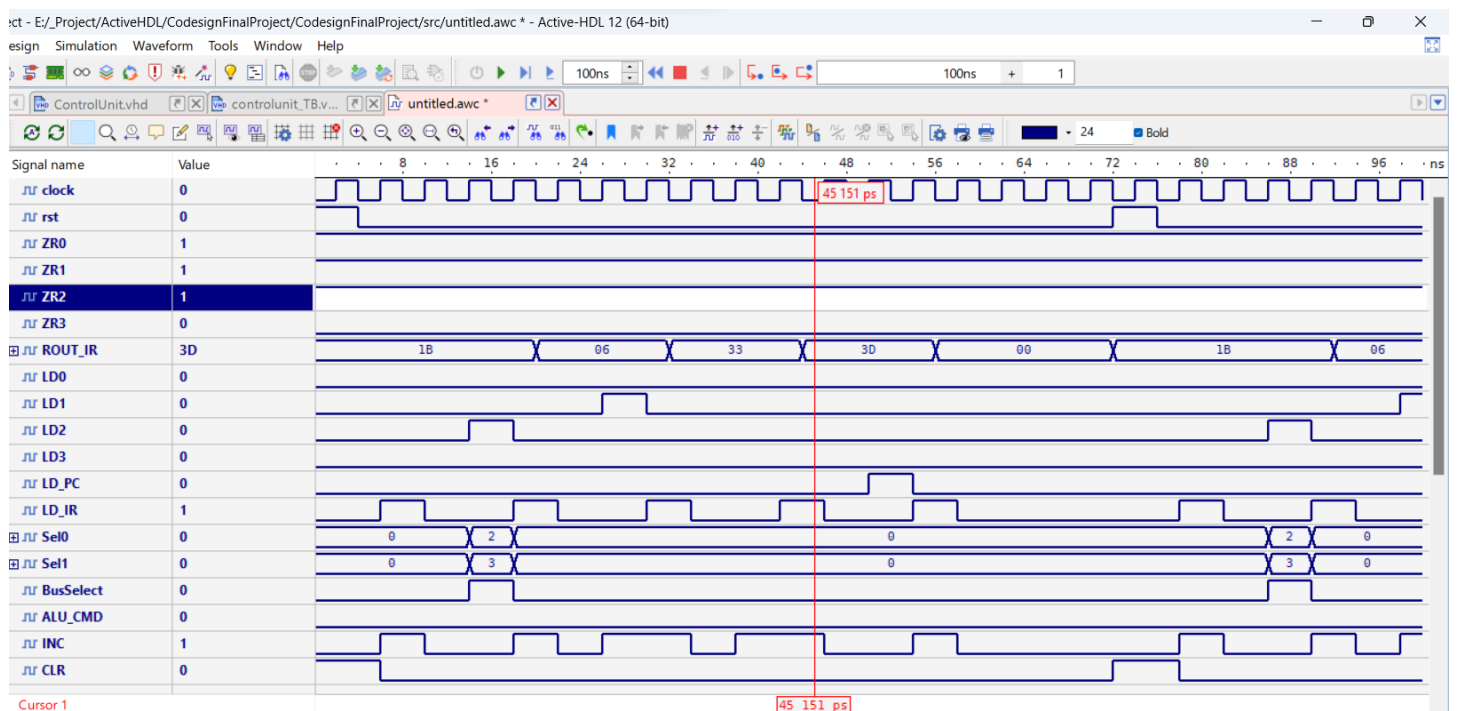
```

```

225
226
227 when S7 => INC <= '1';
228     LD_PC <= '0';
229     CLR <= '0';
230     LD0 <= '0';
231     LD1 <= '0';
232     LD2 <= '0';
233     LD3 <= '0';
234
235     LD_IR <= '0';
236     Sel0 <= "00";
237     Sel1 <= "00";
238     ALU_CMD <= '0';
239     BusSelect <= '0';
240     NSTATE<= S1;
241
242 end case;

```

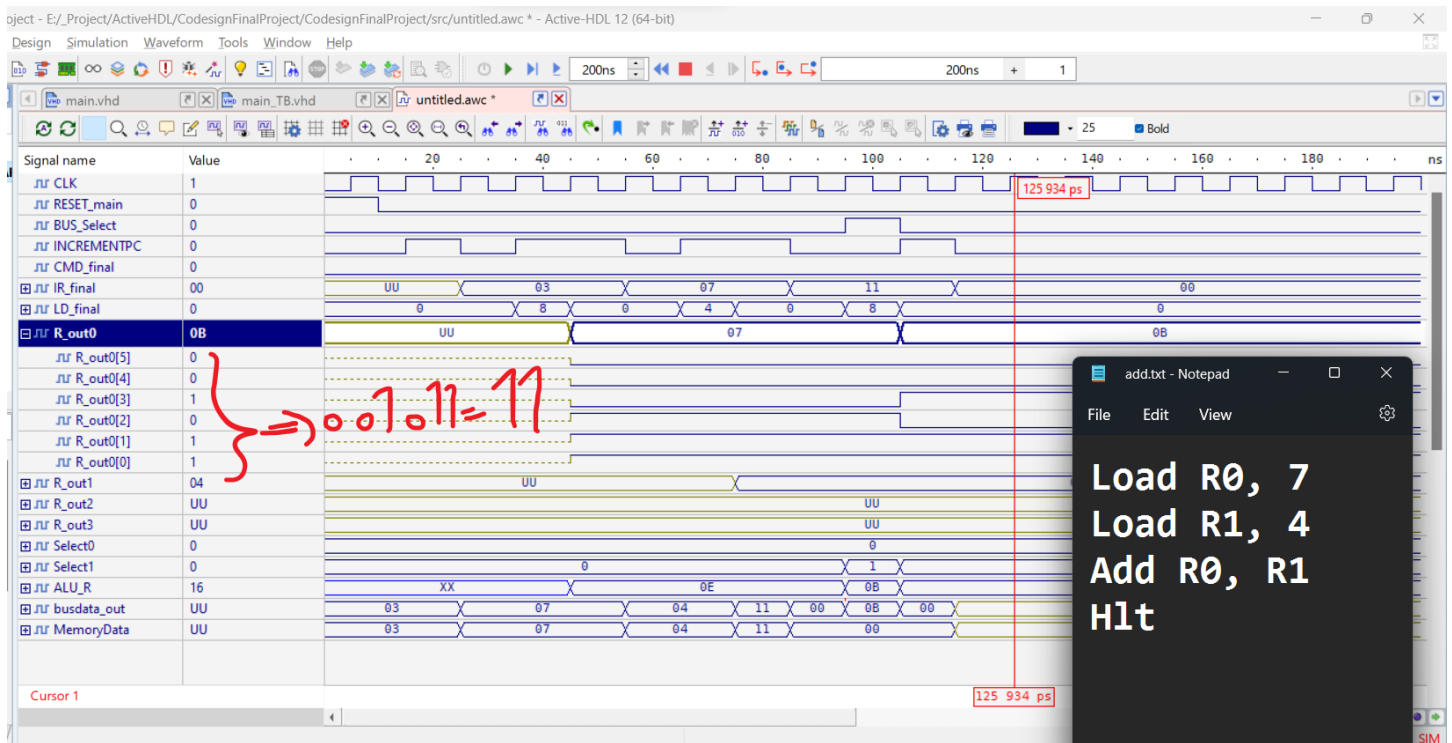
نتیجه رو تو Testbench پایین میبینید :



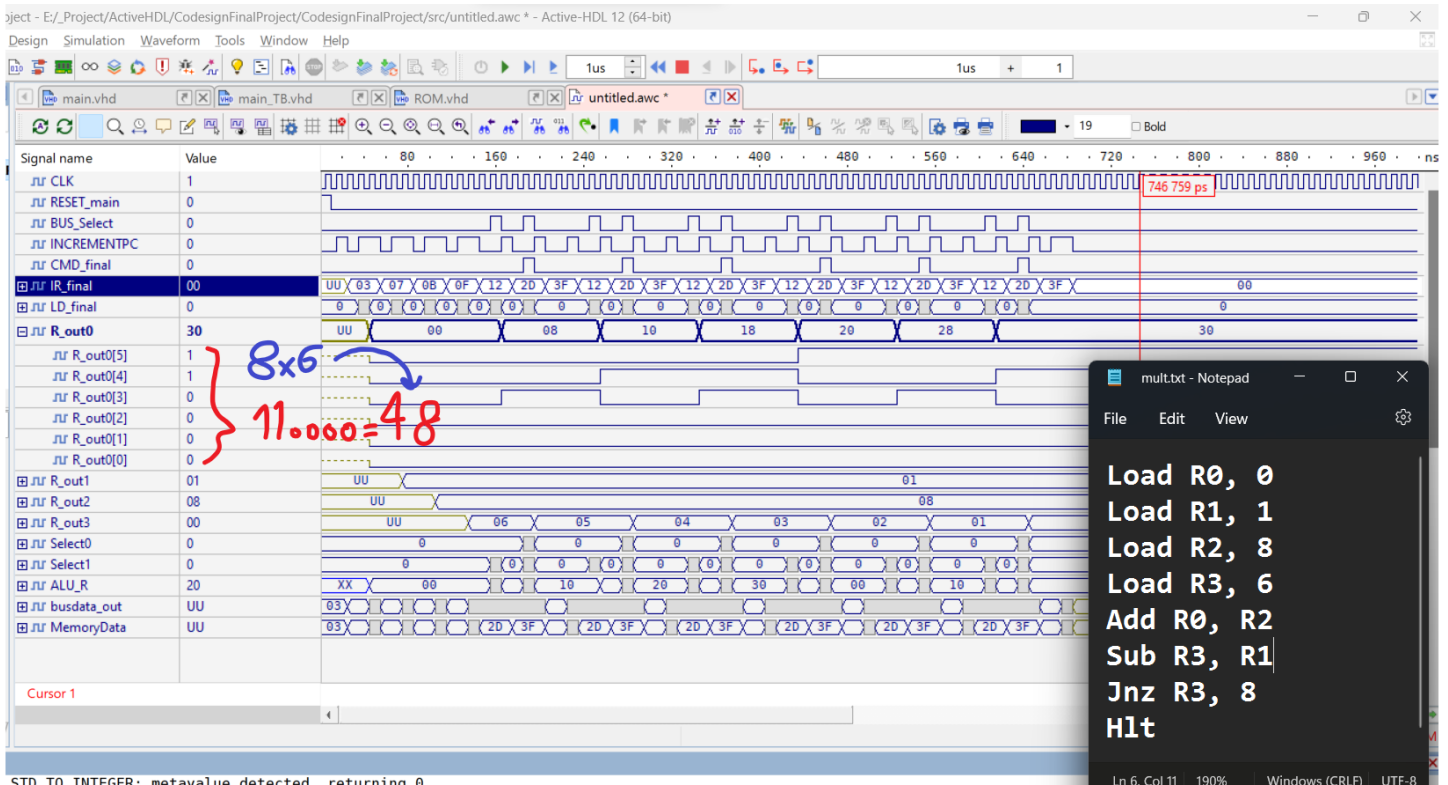
Main

خروجی های جمع و ضرب

خروجی جمع دو عدد 7 و 4 خواسته شده در پروژه :



خروجی ضرب 8 در 6 خواسته شده در پروژه :



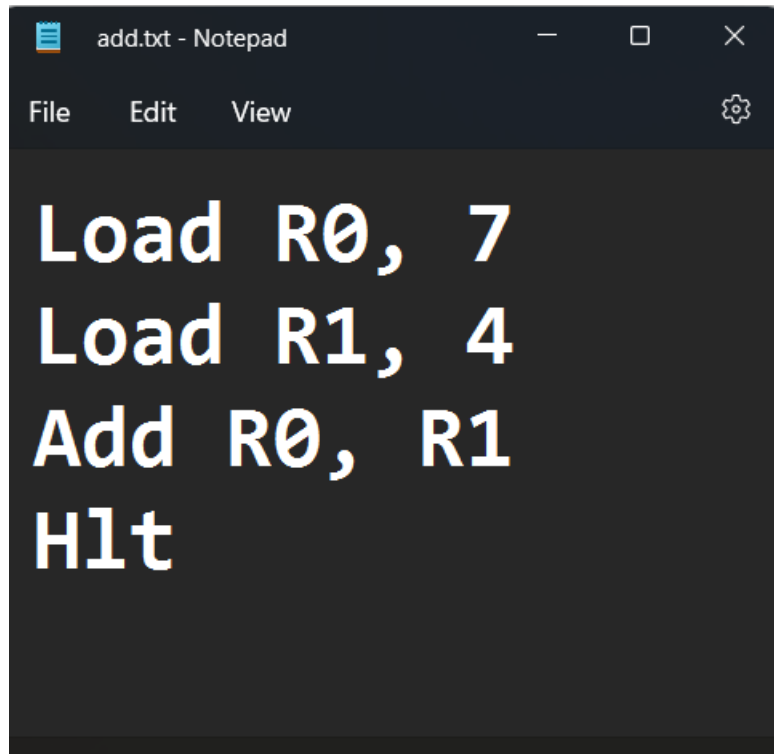
الگوریتم خواسته شده در قسمت دوم و سوم پروژه:

در این الگوریتم ابتدا مقدار صفر را در ثبات R0 قرار می دهیم تا بتوانیم عدد مورد نظر را هر بار با خودش جمع کنیم و در ادامه در ثبات R1 مقدار 1 را قرار می دهیم تا هر بار که یکی از اعداد با خودش جمع می شود عدد دیگر یکی از خودش کم کند و در ثبات R2 و R3 مقادیری که قرار است ضرب شوند را قرار می دهیم و حلقه ی for ما عملیات جمع عدد با خود و کم کردن عدد دیگر را انجام می دهیم تا عدد مورد نظر که تفریق بر روی آن انجام می شود صفر شود و مقدار خروجی را در ثبات R0 قرار می دهد بعد وارد دستور halt می شود و خروجی را نشان می دهد .

Assembler

جمع :

فایل txt ورودی به assembler :



```
add.txt - Notepad
File Edit View
Load R0, 7
Load R1, 4
Add R0, R1
Hlt
```

خروجی کد که در rom قرار میدادیم :

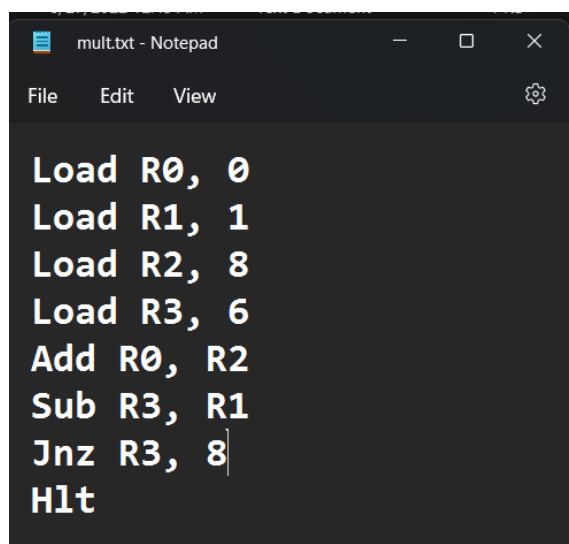
```
run:
  ROM Address is :
m(0) <= "000011" ;
m(1) <= "000111" ;
m(2) <= "000111" ;
m(3) <= "000100" ;
m(4) <= "010001" ;
m(5) <= "000000" ;

BUILD SUCCESSFUL (total time: 0 seconds)
```

```
FileReader file_reader = new FileReader("add.txt");
```

ضرب :

فایل txt ورودی به assembler :



```
Load R0, 0
Load R1, 1
Load R2, 8
Load R3, 6
Add R0, R2
Sub R3, R1
Jnz R3, 8
Hlt
```

خروجی کد که در rom قرار میدادیم :

```
run:
  ROM Address is :
m(0) <= "000011" ;
m(1) <= "000000" ;
m(2) <= "000111" ;
m(3) <= "000001" ;
m(4) <= "001011" ;
m(5) <= "001000" ;
m(6) <= "001111" ;
m(7) <= "000110" ;
m(8) <= "010010" ;
m(9) <= "101101" ;
m(10) <= "111111" ;
m(11) <= "001000" ;
m(12) <= "000000" ;

BUILD SUCCESSFUL (total time: 0 seconds)
```

```
FileReader file_reader = new FileReader("mult.txt");
```