موضوع پروژه: طراحی یک cpu ساده

اعضای گروه: سارا کاویانی –950122681020

امیر مدوی ثابت- 950122681008

توضیحات:

قسمت ممووری ROM از یک آرایه 12 تایی تشکیل شده که به صورت ثابت در آن مقادیری نوشته و ثبت شده‌اند. این مقادیر غیر قابل تغییر هستند که از جدول زیر پیروی میکنند

opcode	کد اسمبلی
001	LOAD
010	ADD
011	SUB
100	JNZ
101	MUL
111	HLT

حافظه ROMبرای قسمت اول پروژه به صورت زیر پیاده سازی شده است

توضیحات	کد دودویی	کد اسمبلی	NUMBER
	0010000	LOAD R0,A	0
Value A=5	0000101		1
	0010100	LOAD R1 , B	2
Value B =3	0000011		3
A=A+B	0100001	ADD R0,R1	4
FINISH	1110000	HLT	5

حافظه ROM برای قسمت دوم پروژه به صورت زیر پیاده سازی شده است

توضیحات	کد دودویی	کد اسمبلی	NUMBER
Sum	0010000	LOAD R0 ,0	0
Value sum=0	0000000		1
A	0010100	LOAD R1 ,A	2
Value A=7	0000111		3
B	0011000	LOAD R2 , B	4
Value B=8	0001000		5
index	0011100	LOAD R3 , i	6
Value index=1	0000001		7
L0: SUM=SUM+B	0100010	ADD R0,R2	8
A=A-1	0110111	SUB R1 , R3	9
REPEAT FROM L0	1000100	JNZ R1 ,L0	10
L0=8	0001000		11
FINISH	1110000	HLT	12

پیاده سازی حافظه ROM

```
type memory is array ( 12 downto 0) of std_logic_vector( 6  downto 0);
constant my_Rom: memory :=(
--load r0
0 =>"0010000",
--value 0
1 =>"0000000",
--load r1
2 =>"0010100",
--value 2
3 =>"0000010",
-- load r2
4 =>"0011000",
-- value 3
5 =>"0000011",
-- load r3
6 =>"0011100",
--value 1
7 =>"0000001",
--add r0 , r2
--L0
8=>"0100010",
--sub r1 ,r3
9=>"0110111",
--jnz r1,L0
10=>"1000100",
--L0
11=>"0001000",
--HLT
12=>"1111111");
```

چگونه به خانه های حافظه دسترسی پیدا می‌کنیم؟

```
------------------------ ROM memory ---------------------------------
process(pc)
begin
    case pc is
        when "0000000" => MData <=my_Rom(0);
        when "0000001" => MData <=my_Rom(1);
        when "0000010" => MData <=my_Rom(2);
        when "0000011" => MData <=my_Rom(3);
        when "0000100" => MData <=my_Rom(4);
        when "0000101" => MData<=my_Rom(5);
        when "0000110" => MData<=my_Rom(6);
        when "0000111" => MData<=my_Rom(7);
        when "0001000" => MData<=my_Rom(8);
        when "0001001" => MData<=my_Rom(9);
        when "0001010" => MData<=my_Rom(10);
        when "0001011" => MData<=my_Rom(11);
        when "0001100" => MData<=my_Rom(12);
        when others =>    MData <="0000000";
    end case;
end process;
```

- **Registers**

پیاده سازی رجیستر ها

رجیستر ها یک ورودی به نام LDX دارند که اگر در لبه بالا رونده کلاک این ورودی یک بود مقدار ورودی از روی خط باس خوانده می‌شود و اگر ورودی LDX یک نبود همان مقدار قبلی رجیستر به آن اعمال می‌شود.

رجیستر ها یک خروجی دیگر به نام ZRX دارند که اگر مقدار LDX یک بود (ورودی جدیدی در حال وارد شدن است ونیاز به بررسی مقدار ورودی است) و مقدار ورودی صفر بود، خروجی ZRX یک می‌شود

```
----------------------R0-------------------
process(clk,LD0,rin_r0,busline,zr0)
begin
        rin_r0<=busline;
    if (rising_edge(clk)) then
        if (LD0='1') then
            rout_r0<=rin_r0;
        else
            rout_r0<=rout_r0;
        end if;
    end if;
        if LD0='1' then
            if rin_r0="000000" then
                zr0<='1';
            end if;

        end if;


end process;
```

- **Ir register**

این رجیستر با لبه بالا رونده کلاک و load=1 مقدار قرار داده شده روی bus را میخواند.

در غیر این صورت مقدار جدیدی روی آن ذخیره نمی‌شود.

```
----------------------IR REG--------------------------
process(clk,LD_IR,rin_ir,busline)
begin
    rin_ir<=busline;
    if (rising_edge(clk)) then
          if  LD_IR ='1' then
            rout_ir <=rin_ir ;
          else
              rout_ir<=rout_ir;


        end if;
    end if;


end process;
```

- **Pc register**

این رجیستر در هر لبه بالا رونده کلاک سه روند مختلف را ممکن است دنبال کند

1) اگر سیگنال increment=1 باشد و در یکی از stateهای state1یا state3 قرار گرفته باشیم به مقدار pc یکی اضافه می‌شود
State1 مربوط به روند کلی برنامه است که پس از خواندن دستور از مموری به خانه بعدی حافظه اشاره می‌کند
State3 مربوط به دستور load هستش که پس از خواندن دستور مقداری که باید در رجیستر ذخیره شود در خانه بعدی حافظه قرار دارد پس pc باید یکی اضافه شود تا به خانه بعدی حافظه که دستور بعدی قرار دارد اشاره کند.
2) اگر سیگنال clear=1باشد مقدار pc ریست می‌شود
3) اگر سیگنال load =1باشد مقداری که از حافظه خوانده شده و روی باس قرار گرفته در pc ذخیره می‌شود این حالت در دستور jnz رخ می‌دهد.

```
-------------------- PC REG----------------------------



process(clk,LD_PC,busline,clr_pc,pc,inc_pc,pr_state)
begin

    if rising_edge(clk) then

        if  inc_pc='1'and (pr_state=state1 or pr_state=state3) then
            pc<=pc + "0000001";


        elsif clr_pc='1' then
            pc<= "0000000";

        elsif (LD_PC='1') then

            pc <=busline ;
            end if;
    end if;
end process;

-----------------------------------------------------
```

- **Mux 0 – Mux 1**

سیگنال reg_select_mux1وreg_select_mux0 در برنامه به صورت زیر مقدار دهی شده است

reg_select_mux0<=rout_ir(3)&rout_ir(2 -- reg source

reg_select_mux1<=rout_ir(1)&rout_ir(0)—reg destination

این مقدار در دستور اسمبلی برابر است با regsource وreg destination

```
------mux1 ----------------------------------------
process (reg_select_mux1,rout_r0,rout_r1,rout_r2,rout_r3)

begin

    case reg_select_mux1 is
        when "00" => mux1<=rout_r0;
        when "01" => mux1<=rout_r1;
        when "10" => mux1<=rout_r2;
        when "11" => mux1<=rout_r3;
        when others =>mux1<="1111111";
    end case;
end process;
--------mux0 ----------------------------------------
process (reg_select_mux0,rout_r0,rout_r1,rout_r2,rout_r3)

begin

    case reg_select_mux0 is
        when "00" => mux0<=rout_r0;
        when "01" => mux0<=rout_r1;
        when "10" => mux0<=rout_r2;
        when "11" => mux0<=rout_r3;
        when others =>mux0<="1111111";
    end case;
end process;
```

- **ALU**

اگر مقدار command یک باشد : جمع
اگر مقدار command دو باشد : تفریق

```
----------------ALU-----------------------------------
process(cmd,mux0,mux1)
begin

    case cmd is
        when "01" => aluresult<= mux0 + mux1;
        when "10" => aluresult<= mux0 - mux1;
        ---multiply
        --when "11" => aluresult<= mux0 mux1;
        when others => aluresult <= "0000000";
    end case;
end process;
-------------------------output---------------------------
```

- **Bus mux**

اگر مقدار bus_select صفر باشد از روی حافظه مقدار مورد نظر را می‌خواند و روی خط bus قرار می‌دهد
اگر مقدار bus_select یک باشد خروجی واحد alu را روی خط bus قرار می‌دهد

```
------------------- bus mux ---------------------------------
process(bus_sel,MData,aluresult)
begin
    case bus_sel is
        when '0' => busline <= MData ;
        when '1'=> busline<=aluresult;
        when others => busline <="0000000";
    end case;
end process;
```

- **Control unit**

کنترل یونیت از دو بخش تشکیل شده است

بخش اول کنترل pr_state و nx_state را به عهده دارد و با کلاک کار میکند

```
-------------------------- control unit -----------------------------
process(reset,clk)
begin
    if reset='1' then
        pr_state<=state0;
    elsif (rising_edge(clk)) then
        pr_state<= nx_state;
    end if;
end process;
```

بخش دوم مدیریت state ها را به عهده دارد و بدون کلاک بر اساس op code وضعیت بعدی را مشخص می‌کند

ir_select<=rout_ir(6)&rout_ir(5)&rout_ir(4)

1) State0 – State1

ماشین حالت از state0 شروع به کار می‌کند و هر بار که دوباره به این state وارد میشویم مقدار pc صفر می‌شود

State 1: مقدار LD_IR یک می‌شود تا رجیستر IR کد دودویی مربوط به دستور مورد نظر را از حافظه دریافت کند

اگر در این state مقدار pc برابر 12 یا بیشتر شد به یکی از شرط های خاتمه برنامه رسیده ایم و دستور دیگری در حافظه 12 خانه ای وجود ندارد. و به state0 میرویم تا دوباره از اول شروع کنیم

در غیر این صورت به state8 می‌رویم تا opcode را بررسی کنیم

```
process(nx_state,pr_state,rout_ir,pc,busline, cmd)
begin
    reg_select_mux0<=rout_ir(3)&rout_ir(2);
    reg_select_mux1<=rout_ir(1)&rout_ir(0);
    ir_select<=rout_ir(6)&rout_ir(5)&rout_ir(4);
    case pr_state is
        when state0 =>
            clr_pc <='1';
            nx_state<=state1;
        when state1 =>
            LD_IR   <='1';
            LD0<='0';
            LD1<='0';
            LD2<='0';
            LD3<='0';
            LD_pc<='0';
            inc_pc<='1';
            bus_sel<='0';
    -------------- it is the end of rom memory----------------------------
            if pc >"0001100" then
                nx_state<=state0;
            else
                nx_state<=state8;
            end if;
```

State8

در این state بر اساس opcode تصمیم میگیریم که به هر کدام از statesهای 2 تا 7 برویم

State 4: add

State 2 : hlt

State3 :load

State 5: sub

State 6 : jnz

State7: mul

```
----  wait for one clk --------
        when state8 =>
        inc_pc<='0';
        LD_IR<='0';
        case ir_select is
            when "111" =>
            nx_state<=state2;  --hlt
            when "001" =>
            nx_state<=state3;   --load
            when "010" =>
            nx_state<=state4; --add
            when "011" =>
            nx_state<=state5; --sub
            when "100"=>
            --checking zrx------------
```

قسمت دوم state8

اگر به هیچکدام از stateهای 2تا5 وارد نشدیم در شرایطی که ir_select ="100" باشد

بر اساس مقدار reg_select_mux0 بررسی میکنیم که شرط دستور jnz برقرار است یا خیر و بر اساس آن به state 6یا state9 وارد می‌شویم

reg_select_mux0<=rout_ir(3)&rout_ir(2 -- reg source

reg_selct_mux0 مشخص می‌کند صفر بودن مقدار کدام رجیستر باید چک شود

```
    --checking zrx------------
            case reg_select_mux0 is
            --enabling reg0 first input
            when "00" =>
            if zr0='1' then
                nx_state <=state9;  ---do nothing
            else
                nx_state<=state6;    ---jnz
            end if;
            --checking register2
            when "01" =>
            if zr1='1' then
                nx_state<=state9;---do nothing
            else
                nx_state<=state6;---jnz
            end if;
            --enabling reg2 first input
            when "10"=>
            if zr2='1' then
                nx_state<=state9;--- do nothing
            else
                nx_state<=state6;---jnz
            end if;
            --checking register3
            when "11" =>
            if zr3='1' then
                nx_state<=state9;-- do nothing
            else
                nx_state<=state6;--jnz
            end if;
            when others =>
        end case;
    when "101"=>
    nx_state<=state7; --mul
    when others=>
end case;
```

State2- State3

در state2 به وضعیت اتمام برنامه میرویم و در آن state می‌مانیم

در state3 میخواهیم مقدار رجیستر sourceرا به روز رسانی کنیم و مقدار جدیدی در آن ذخیره کنیم
به همین دلیل از reg_select_mux0استفاده می‌کنیم تا شماره رجیستر مورد نظر را پیدا کنیم و ورودی load آن را یک می‌کنیم تا بتواند از روی خط باس مقدار جدید را ذخیره کند

reg_select_mux0<=rout_ir(3)&rout_ir(2 -- reg source

```
-----------------------------------------
        when state2=>
            nx_state<=state2;
---------------------------load----------------
        when state3=>
        nx_state<=state1;

            inc_pc<='1' ;
            bus_sel<='0';

        case reg_select_mux0 is
            --enabling reg0 first input
            when "00" =>
                LD0<='1';
            --enabling reg1 first input
            when "01" =>
                LD1<='1';
            --enabling reg2 first input
            when "10"=>
                LD2<='1';
            --enabling reg3 first input
            when "11" =>
            LD3<='1';
            when others =>

        end case;
```

State4

در این state عملیات جمع انجام می‌شود و ما نیاز داریم تا مشخص کنیم حاصل عملیات جمع در کدام رجیستر ذخیره شود. به همین دلیل از reg_select_mux0 استفاده می‌کنیم تا رجیستر source را شناسایی کنیم و ورودی load آن را فعال کنیم تا حاصل جمع در آن ذخیره و مقدار جدیدی به آن رجیستر داده شود.

reg_select_mux0<=rout_ir(3)&rout_ir(2 -- reg source

```
when state4 =>
        nx_State<=state1;
        cmd<="01";
        bus_sel<='1';

case reg_select_mux0 is
    --enabling reg0 first input
    when "00" =>
    LD0<='1';

    --enabling reg1 first input
    when "01" =>
    LD1<='1';

    --enabling reg2 first input
    when "10"=>
    LD2<='1';

    --enabling reg3 first input
    when "11" =>
    LD3<='1';

    when others =>

end case;
```

State5

در این state عملیات تفریق انجام می‌شود و ما نیاز داریم تا مشخص کنیم حاصل عملیات تفریق در کدام رجیستر ذخیره شود. به همین دلیل از reg_select_mux0 استفاده می‌کنیم تا رجیستر source را شناسایی کنیم و ورودی load آن را فعال کنیم تا حاصل تفریق در آن ذخیره و مقدار جدیدی به آن رجیستر داده شود.

reg_select_mux0<=rout_ir(3)&rout_ir(2 -- reg source

```
------------------------sub---------------------------------
        when state5=>
                nx_state<=state1;
                cmd<="10";
                bus_sel<='1';
            case reg_select_mux0 is
            --enabling reg0 first input
            when "00" =>
            LD0<='1';


            --enabling reg1 first input
            when "01" =>
            LD1<='1';

            --enabling reg2 first input
            when "10"=>
            LD2<='1';

            --enabling reg3 first input
            when "11" =>
            LD3<='1';

            when others =>

        end case;
```

State6 – State 7

در state6 اگر شرط دستور jnz برقرار بود و مقدار رجیستر مورد نظر صفر نبود به ادرسی که دستور اشاره می‌کند می‌رویم و دوباره فرایند fetch ,decode انجام می‌شود.

state7

در این state عملیات ضرب انجام می‌شود و ما نیاز داریم تا مشخص کنیم حاصل عملیات ضرب در کدام رجیستر ذخیره شود. به همین دلیل از reg_select_mux0 استفاده می‌کنیم تا رجیستر source را شناسایی کنیم و ورودی load آن را فعال کنیم تا حاصل تفریق در آن ذخیره و مقدار جدیدی به آن رجیستر داده شود.

reg_select_mux0<=rout_ir(3)&rout_ir(2 -- reg source

```
-----------------------------jnz-------------------------------
        when state6=>
            LD_PC<='1' ;
            bus_sel<='0';
            nx_state<=state1;
---------------------------mul ---------------------------------------
        when state7=>
                nx_state<=state1;
                cmd<="11";
                bus_sel<='1';
                --enabling reg0 first input
        case reg_select_mux0 is
            when "00" =>
            LD0<='1';

            --enabling reg1 first input
            when "01" =>
            LD1<='1';

            --enabling reg2 first input
            when "10"=>
            LD2<='1';

            --enabling reg3 first input
            when "11" =>
            LD3<='1';

            when others =>

        end case;
```

State9

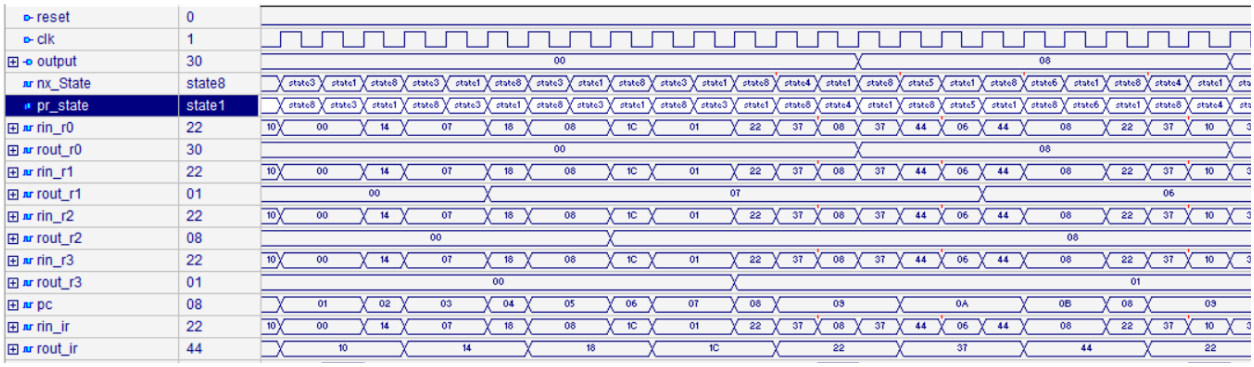
در صورتی که شرط jnz برقرار نباشد به این state وارد می‌شویم
چون در حالت عادی pc به خانه بعدی حافظه اشاره می‌کند و زمانی که به دستور jnz میرسیم
مقدار بعدی که pcبه آن اشاره می‌کند آدرس بعدی پرش می‌باشد باید از این خانه حافظه رد شویم
تا دستور بعدی را fetch و سپس decodeکنیم
به همین دلیل ورودی increment pc را یک می‌کنیم تا مقدار pcافزایش یابد

```
-------------------------------------------------nothing? ------------------
    when state9=>
        inc_pc<='1';
        nx_state<=state1;
    end case;
end process;

-------------------------------------------------------------------
```

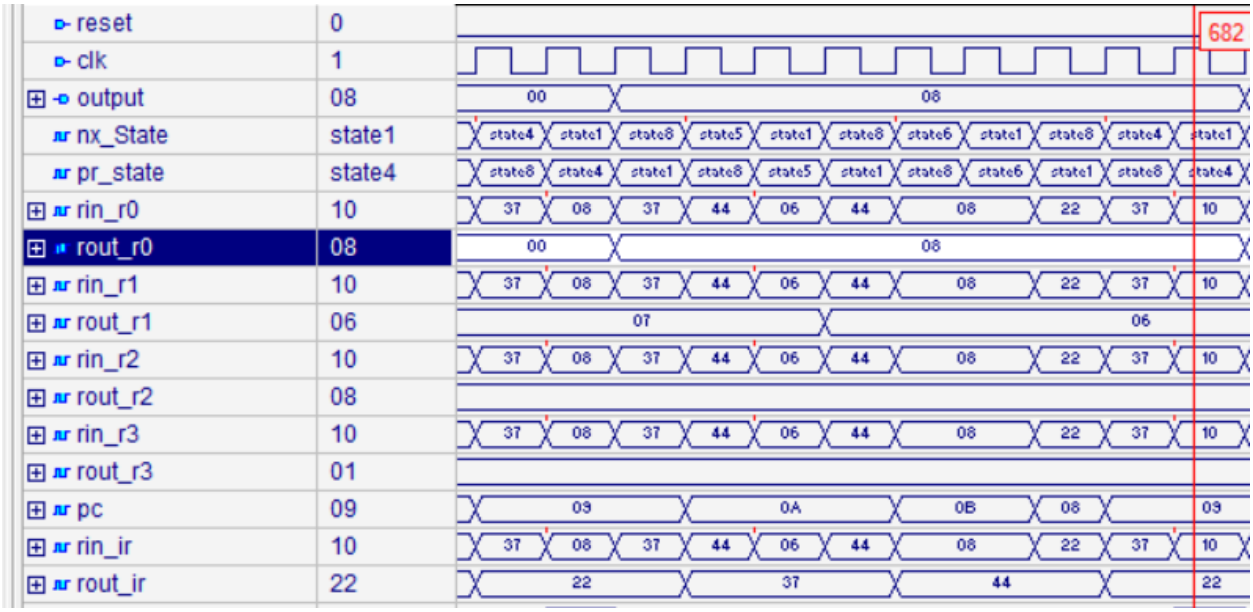
خروجی

اولین دور اجرای برنامه (مقدار دهی به رجیستر ها) و اولین دور اجرای حلقه for برای محاسبه ضرب دو عدد از طریق جمع تکراری



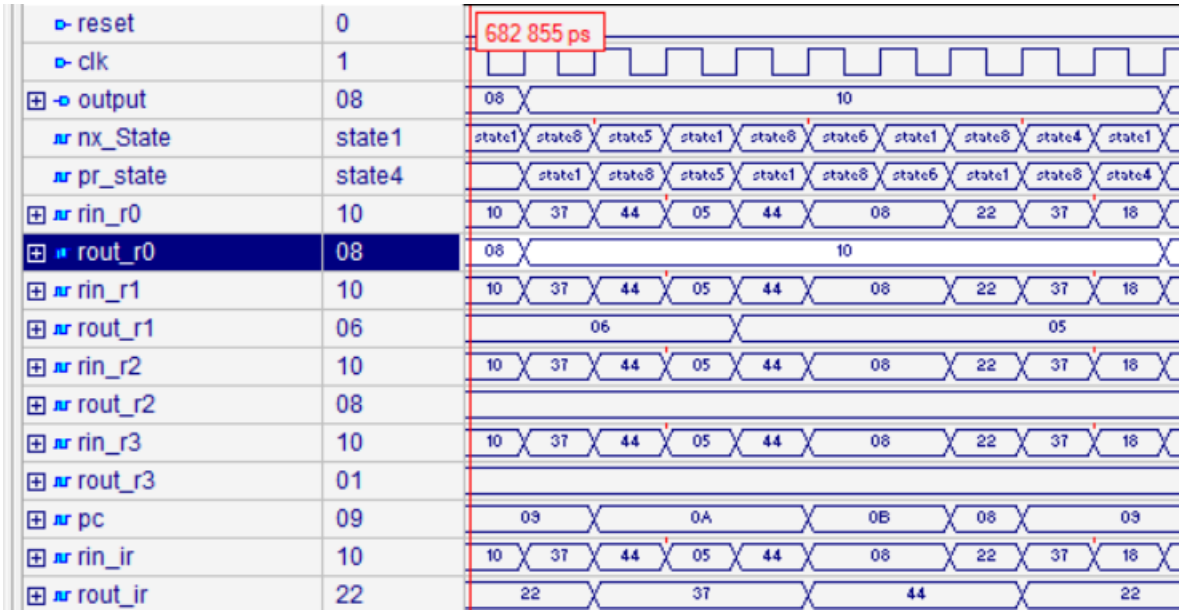
به ترتیب اولین دور اجرای حلقه for

8=0+8



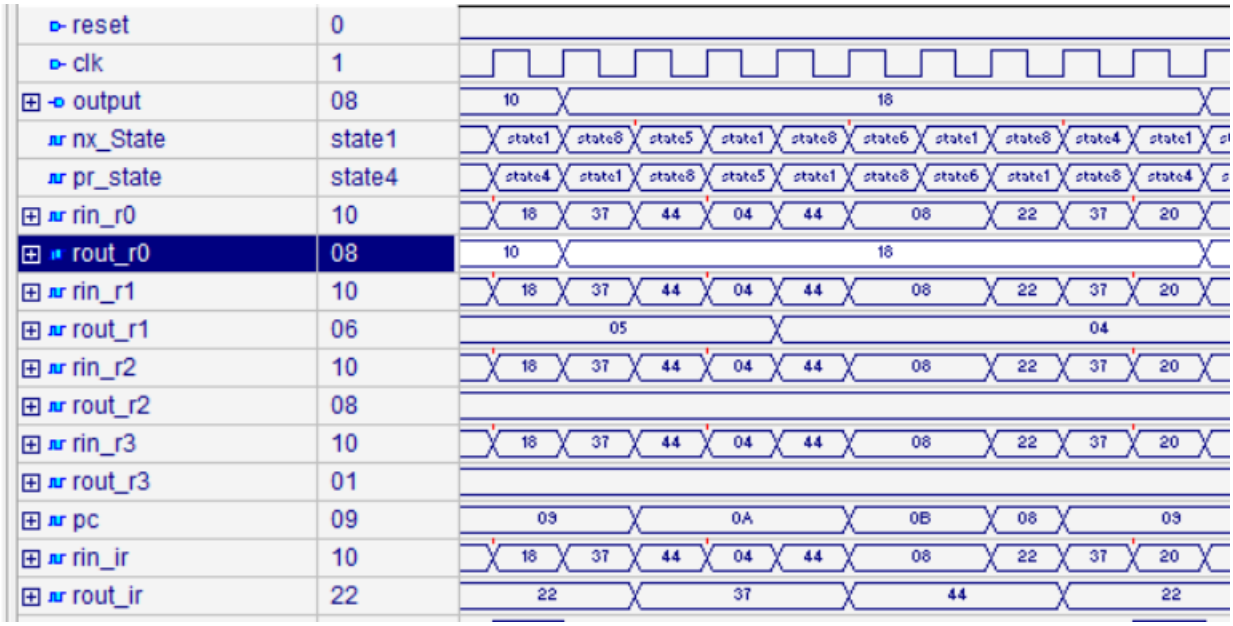
دومین دور اجرای حلقه for

8+8=16 => hex :10



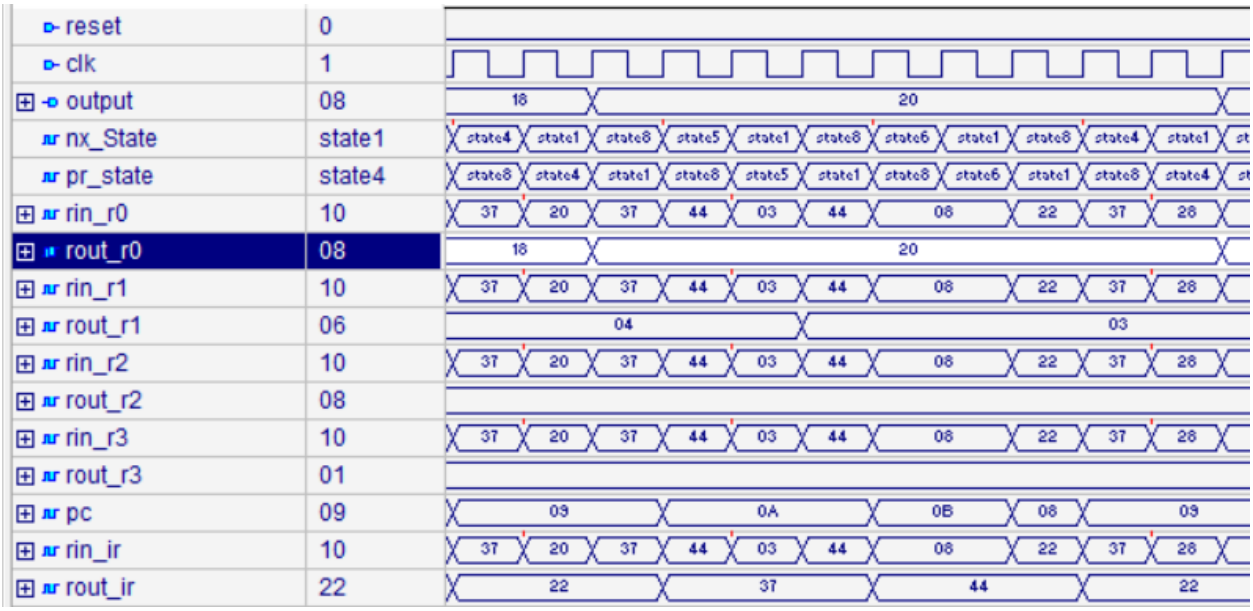
سومین دور اجرای حلقه for

16+8=24 => hex:18



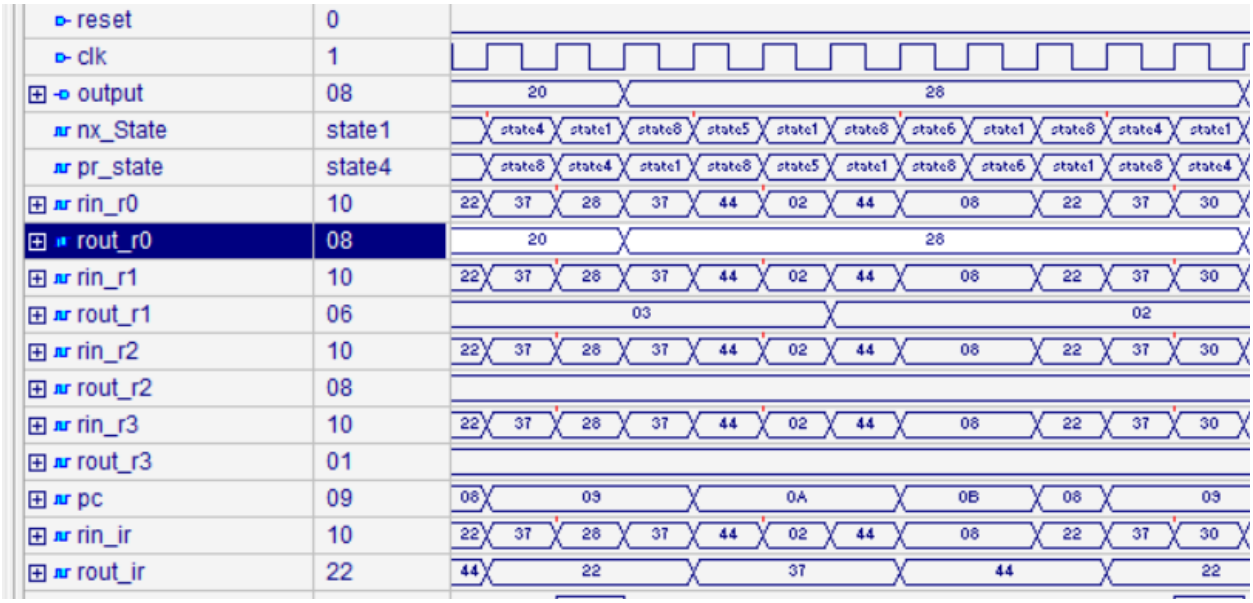
چهارمین دور اجرای حلقه for

24+8=32=> hex: 20



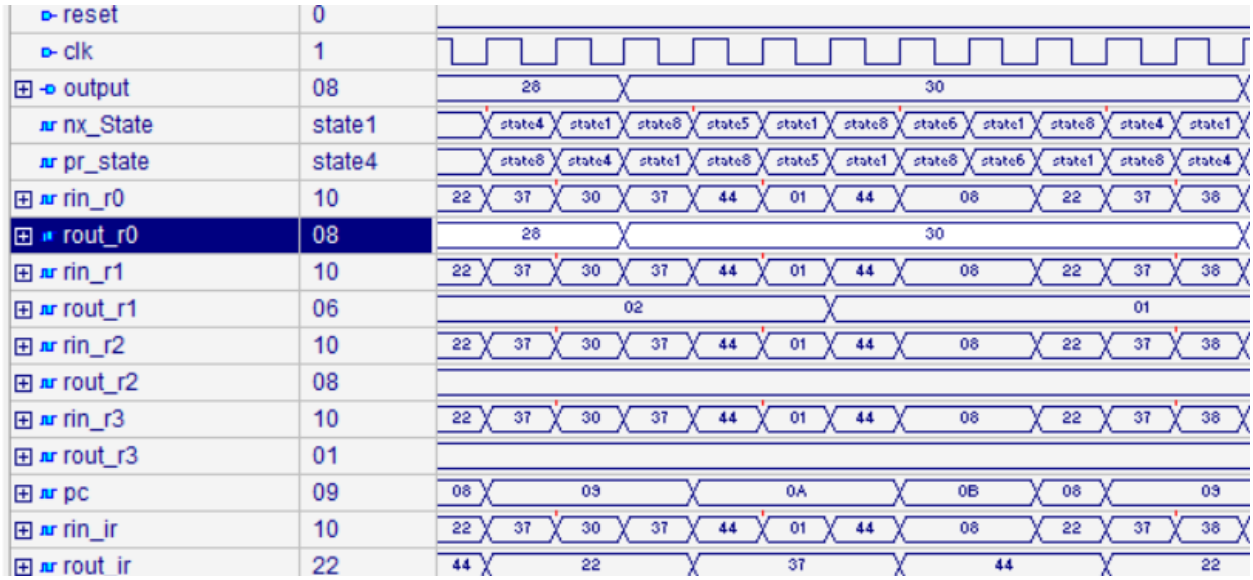
پنجمین دور اجرای حلقهfor

32+8=40=>hex:28



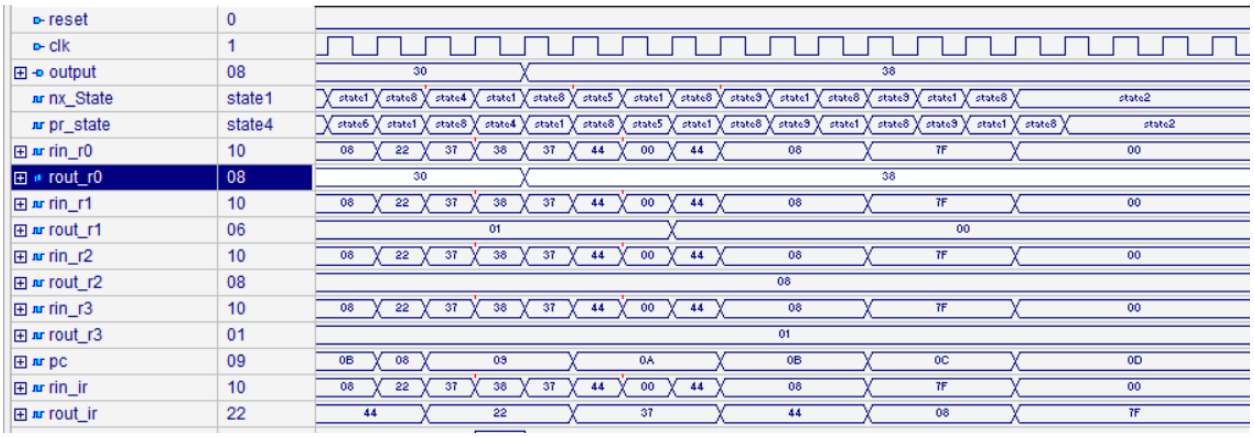
شیشمین دور اجرای حلقهfor

40+8=48 =>hex:30



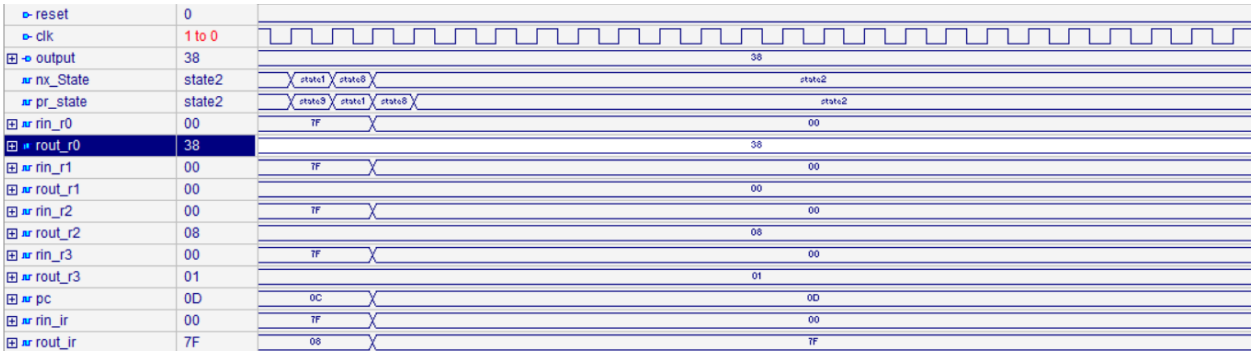
هفتمین دور اجرای حلقهfor

48+8=56 hex:38



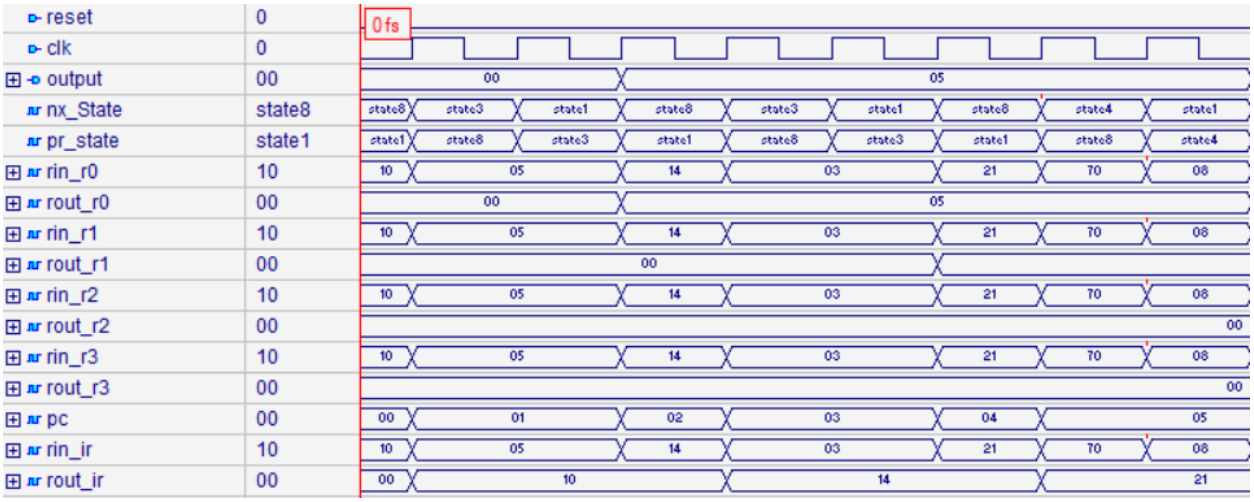
مرحله آخر اجرای برنامه

شرط خاتمه برنامه رسیدن به state2 و ماندن در همین state است

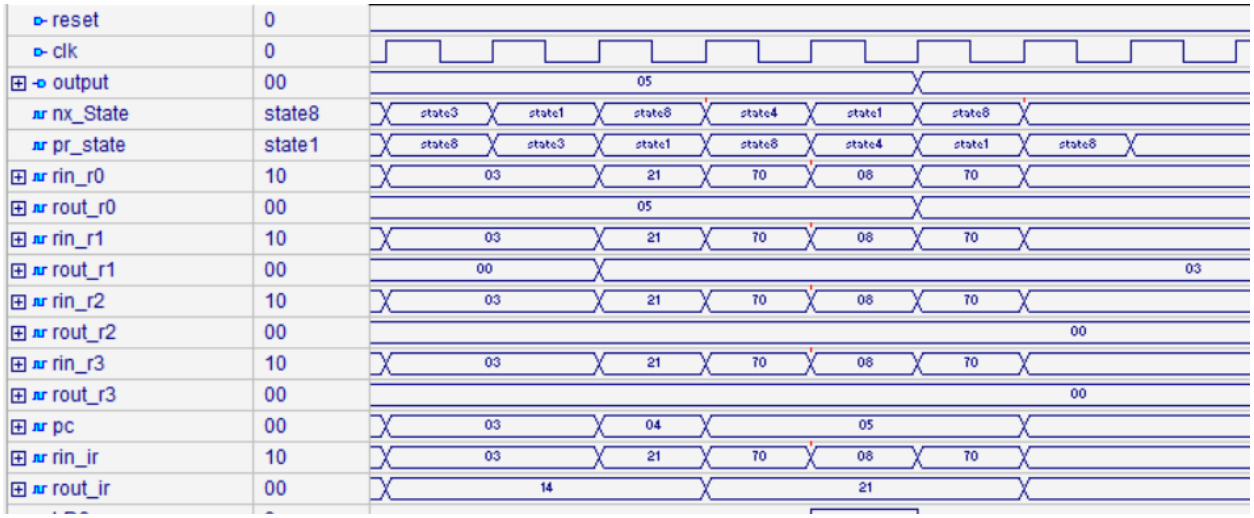


انجام عمل جمع

Load r0



Loadr1



Adding r0,r1