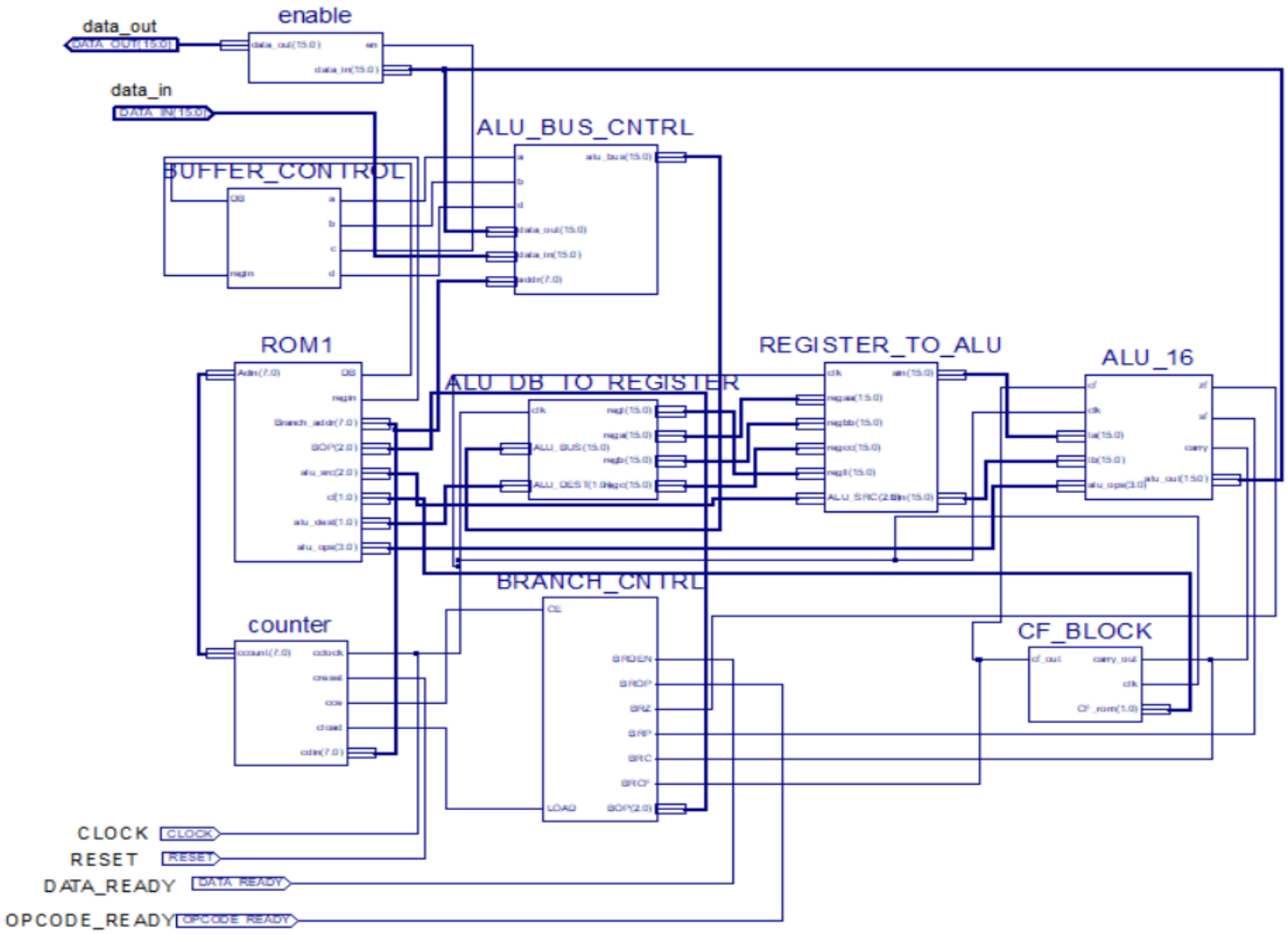
PROJECT-2 REPORT

NAME: SHARATH VENKATESH

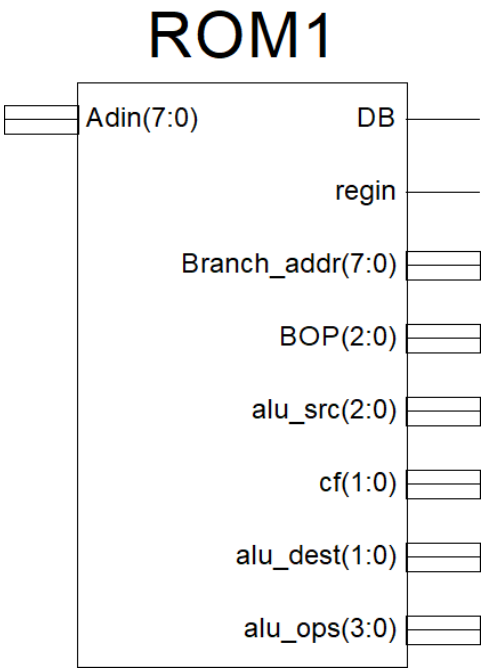
CWID: 893324566

OVERALL BLOCK DIAGRAM



Discussion on each component

ROM: This is where the microcode resides, the ROM is a component used to store data that controls every component of the processor.



VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ROM1 is
port(Adin:in std_logic_vector(7 downto 0);
        Branch_addr: out std_logic_vector(7 downto 0);
        DB,regin: out std_logic;
```

```
        BOP ,alu_src :out std_logic_vector(2 downto 0);
        cf,alu_dest:out std_logic_vector(1 downto 0);
        alu_ops:out std_logic_vector(3 downto 0));
end ROM1;
architecture archROM1 of ROM1 is
--microcode for ADD16u-----------------------------------------------------------------------
constant data1:STD_LOGIC_VECTOR(23 downto 0):="000000000111010000000001";
constant data2:STD_LOGIC_VECTOR(23 downto 0):="000000000111000000000000";
constant data3:STD_LOGIC_VECTOR(23 downto 0):="000000000111010000000011";
constant data4:STD_LOGIC_VECTOR(23 downto 0):="010000000111000000000000";
constant data5:STD_LOGIC_VECTOR(23 downto 0):="101000000110001100000001";

type rom_array is array(NATURAL range<>) of STD_LOGIC_VECTOR(23 downto 0);
constant rom:rom_array:=(data1,data2,data3,data4,data5);
begin
process(Adin)
variable j:integer;
variable M: std_logic_vector(23 downto 0);
begin
j:=conv_integer(Adin);
M:=rom(j);
Branch_addr<=M(7 downto 0);
DB<=M(8);
BOP<=M(11 downto 9);
regin<=M(12);
cf<=M(14 downto 13);
alu_ops<=M(18 downto 15);
```

```
alu_src<=M(21 downto 19);
alu_dest<=M(23 downto 22);
end process;
end archROM1;
```

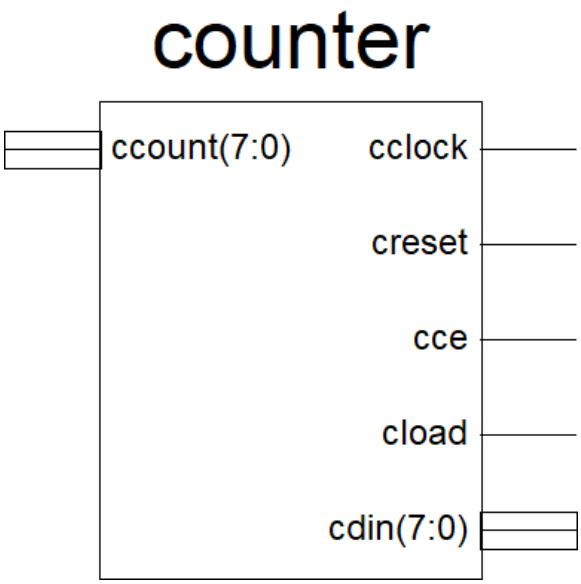
ROM SIMULATION

Explanation:

The ROM moves to the location pointed by the adin [7:0] and outputs the data in the specified location.

In this case adin is “00000100”, so the ROM outputs data in the location “00000100” and branches to Branch location specified in the data, in this case Branch address is “00000000”.

COUNTER: The counter is responsible for propagating through the microcode inside the ROM.

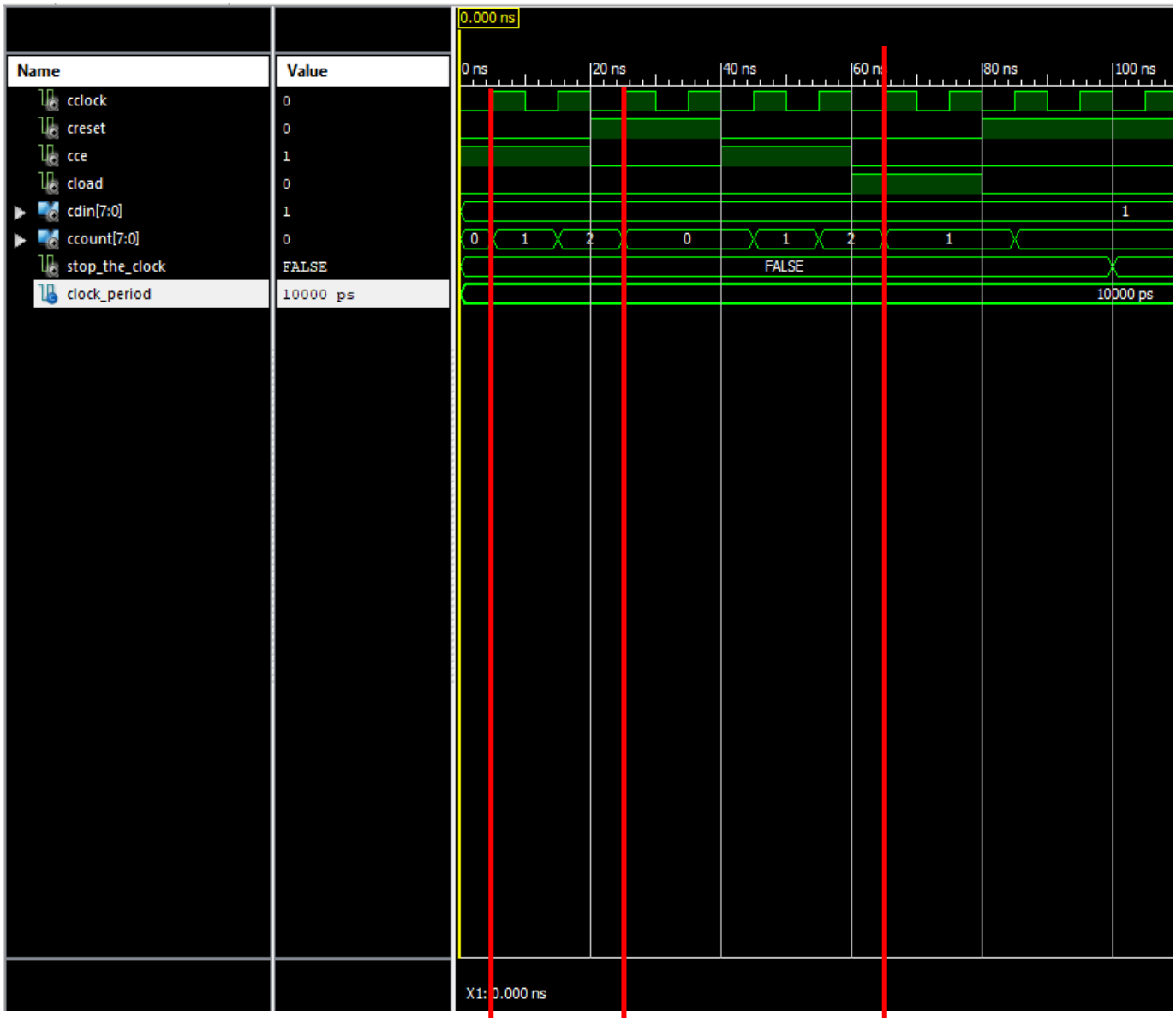


VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity counter is
port(cclock,creset,cce,cload:in std_logic;
                cdin:in std_logic_vector(7 downto 0);
                ccount:out std_logic_vector(7 downto 0));
                end counter;
architecture arch_counter of counter is
begin
process(cclock,creset)
```

```
variable tmp : std_logic_vector(7 downto 0):=x"00";
begin
  if rising_edge(cclock) then
        if creset='1'then
    tmp:=x"00";
        end if;
  if(cce='0' and cload='0') then
  tmp:=tmp;
  elsif(cce='0' and cload='1') then
  tmp:=cdin;
  elsif(cce='1' and cload='0') then
  tmp:=tmp+1;
  elsif(cce='1' and cload='1') then
  tmp:=cdin;
  end if;
end if;
ccount<= tmp;
end process;
end arch_counter;
```

COUNTER SIMULATION

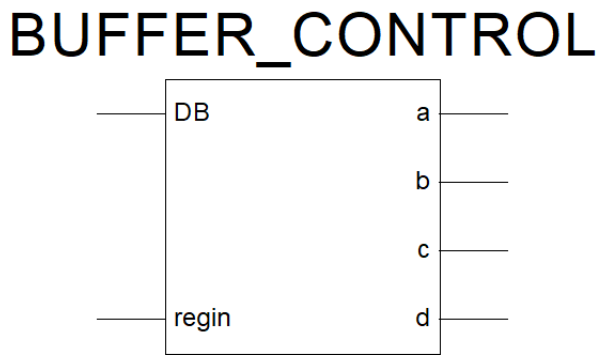


When Count enable is high and Count Load is low, at the rising edge of clock the counter counts

When Count reset is high the count output is zero

When Count enable is low and Count Load is high, at the rising edge of clock cdin is loaded to counter output

BUFFER CONTROL: The Buffer control is a component used to generate control signals to ALU_BUS_CONTROL. i.e. a, b, c, d. based on the input signal DB, Regin.

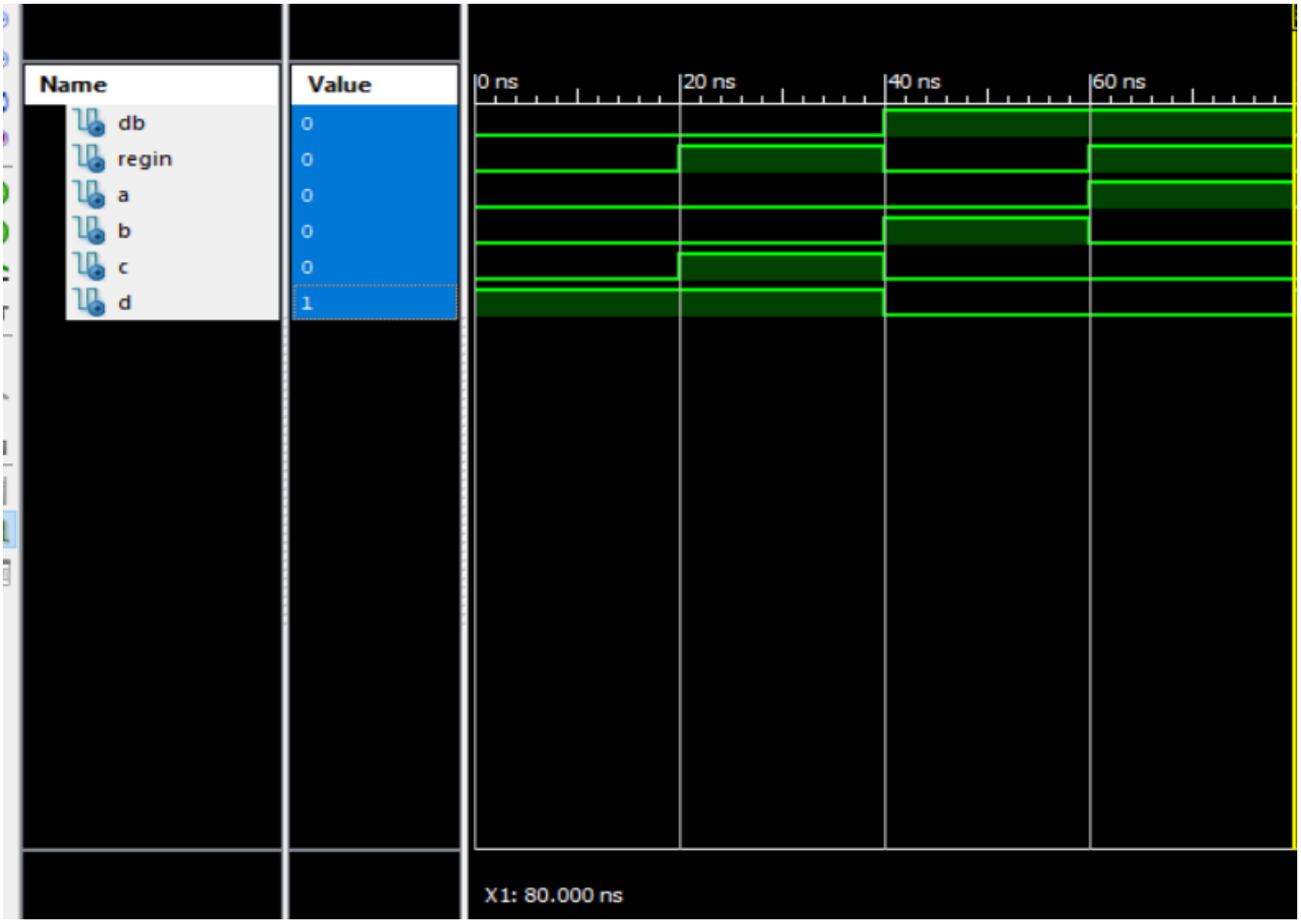


VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity BUFFER_CONTROL is
port(DB,regin :in std_logic;
a,b,c,d :out std_logic);
end BUFFER_CONTROL;
architecture Behavioral of BUFFER_CONTROL is
begin
process(DB,regin)
begin
if(DB='0') and (regin='0') then
a<='0';
```

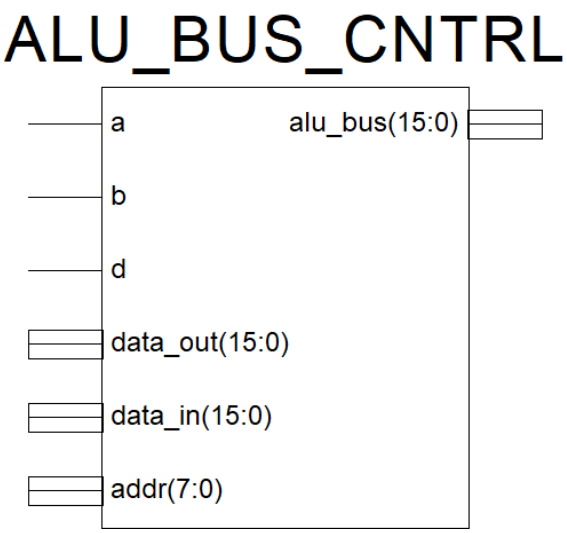
```
b<='0';
c<='0';
d<='1';
elsif(DB='0') and (regin='1') then
a<='0';
b<='0';
c<='1';
d<='1';
elsif(DB='1') and (regin='0') then
a<='0';
b<='1';
c<='0';
d<='0';
elsif(DB='1') and (regin='1') then
a<='1';
b<='0';
c<='0';
d<='0';
end if;
end process;
end Behavioral;
```

BUFFER CONTROL SIMULATION



Regin	DB	a	b	c	d
0	0	0	0	0	1
0	1	0	0	1	1
1	0	0	1	0	0
1	1	1	0	0	0

ALU BUS CONTROL: This component determines the source of data to the Registers. Data can be sent from the ALU output(data_out), Data bus input(data_in), Opcode input(addr). These sources are selected according to the control input signal a,b,d.

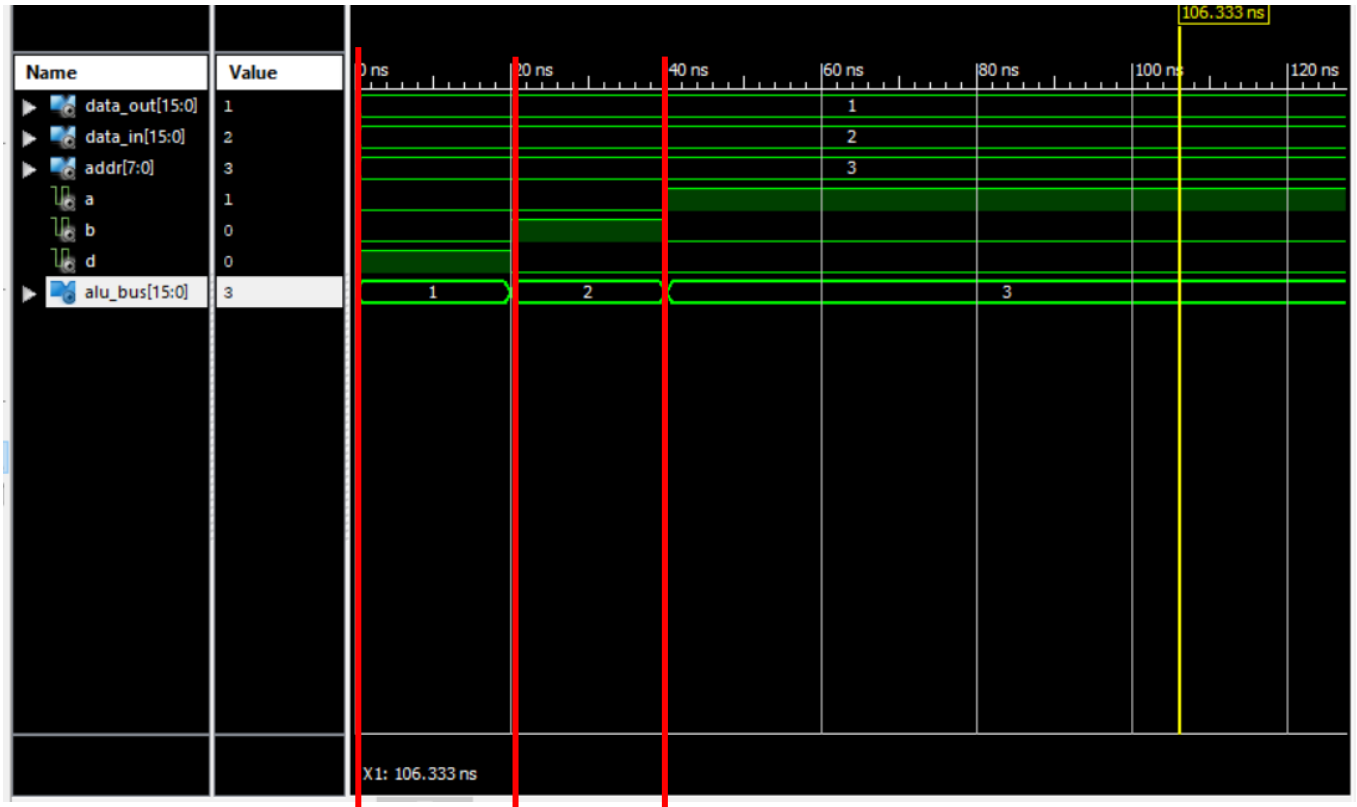


VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity ALU_BUS_CNTRL is
port( data_out,data_in:in std_logic_vector(15 downto 0);
a,b,d:in std_logic;
addr: in std_logic_vector(7 downto 0);
alu_bus:inout std_logic_vector(15 downto 0));
```

```
end ALU_BUS_CNTRL;
architecture Behavioral of ALU_BUS_CNTRL is
begin
process(a,b,d,alu_bus,data_in,data_out,addr)
begin
if b = '1' then
        alu_bus<=data_in;
elsif a = '1' then
        alu_bus<=x"0000";
        alu_bus(7 downto 0)<=addr;
elsif d = '1' then
        alu_bus<=data_out;
else
        alu_bus<=alu_bus;
end if;
end process;
end Behavioral;
```

ALU BUS CONTROL SIMULATION

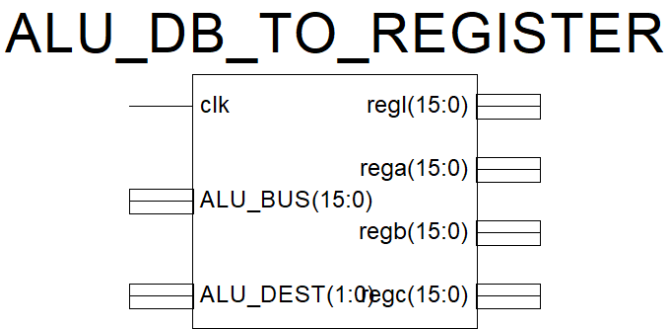


When a=’1’.
alu_bus=addr

When d=’1’.
alu_bus=data_out

When b=’1’.
alu_bus=data_in

ALU/DB to Register: This component determines the destination register to the input ALU_BUS. The destination register is controlled by the control signal Alu_dest.



VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity ALU_DB_TO_REGISTER is
port( ALU_BUS :in std_logic_vector(15 downto 0);
ALU_DEST :in std_logic_vector(1 downto 0);
clk :in std_logic;
regl,rega,regb,regc :out std_logic_vector(15 downto 0):=x"0000");
end ALU_DB_TO_REGISTER;
architecture Behavioral of ALU_DB_TO_REGISTER is
begin
process(ALU_BUS,ALU_DEST,clk)
begin
```

```
if(rising_edge(clk) ) then
if( ALU_DEST = "00")then
rega<=ALU_BUS;
elsif(ALU_DEST = "01")then
regb<=ALU_BUS;
elsif(ALU_DEST = "10")then
regc<=ALU_BUS;
elsif(ALU_DEST = "11")then
regl<=ALU_BUS;
end if;
end if;
end process;
end Behavioral;
```

ALU/DB to Register simulation:

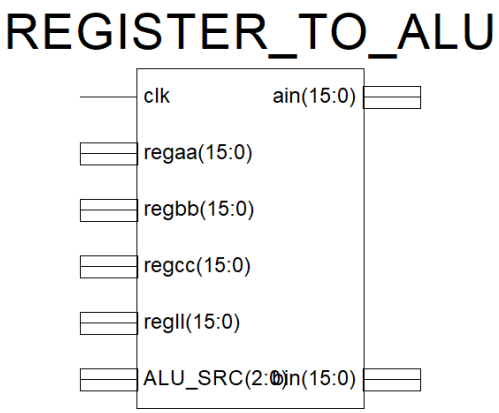
When alu_dest=”11”, then output In Reg L

When alu_dest=”00”, then output In Reg A

When alu_dest=”10”, then output In Reg C

When alu_dest=”01”, then output In Reg B

REGISTER to ALU: This component is responsible to control the input data to the ALU. The input data are from four source registers and the Register to ALU component selects any two of the four source registers as ALU input for binary operations, and one of the four source registers for unary operations.



VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity REGISTER_TO_ALU is
port(regaa,regbb,regcc,regll :in std_logic_vector(15 downto 0);
ALU_SRC :in std_logic_vector(2 downto 0);
ain,bin :inout std_logic_vector(15 downto 0);
clk: in std_logic);
end REGISTER_TO_ALU;
```

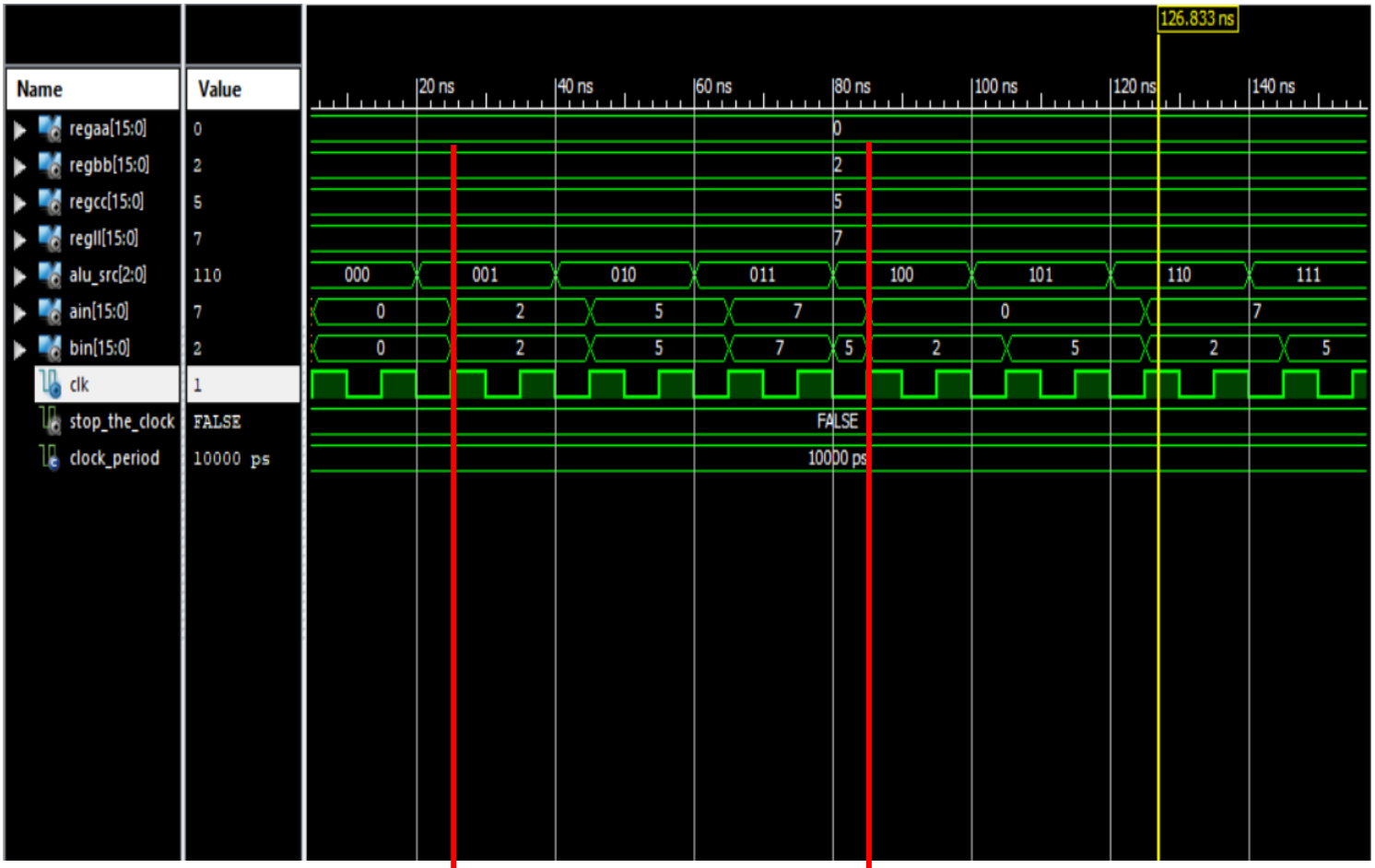
```
architecture Behavioral of REGISTER_TO_ALU is
----------decoder----------
component decoder is
port(ALU_SRCd :in std_logic_vector(2 downto 0);
D1,D2,D3:out std_logic;
clk :in std_logic);
end component;
----------mux2_1-------------
component mux2_1 is
port(in1_1,in1_2 :in std_logic_vector(15 DOWNTO 0);
sl1:in std_logic;
out1:out std_logic_vector(15 DOWNTO 0));
end component;
---------ALL_SIGNALS-----------
signal s1,s2,s5 :std_logic;
signal s3,s4,s6 :std_logic_vector(15 downto 0);
--------------------------------
begin
G1:decoder port map(ALU_SRC,s1,s2,s5,clk);
G2:mux2_1 port map(regcc,regbb,s1,s3);
G3:mux2_1 port map(regaa,regll,s2,s4);
G4:mux2_1 port map(s3,s4,s5,s6);
G5:mux2_1 port map(s6,s4,ALU_SRC(2),ain);
G6:mux2_1 port map(ain,s3,ALU_SRC(2),bin);
end Behavioral;

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

```
---Entity decoder----------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity decoder is
port(ALU_SRCd :in std_logic_vector(2 downto 0);
D1,D2,D3 :out std_logic;
clk: in std_logic);
end decoder;
architecture archdecoder of decoder is
begin
process(clk,AlU_SRCd)
begin
if(rising_edge(clk))then
        if(ALU_SRCd ="000")then
        D1<='X';
        D2<='0';
        D3<='1';
        elsif(ALU_SRCd ="001")then
        D1<='1';
        D2<='X';
        D3<='0';
        elsif(ALU_SRCd ="010")then
        D1<='0';
        D2<='X';
```

```
        D3<='0';
        elsif(ALU_SRCd ="011")then
        D1<='X';
        D2<='1';
        D3<='1';
        elsif(ALU_SRCd ="100")then
        D1<='1';
        D2<='0';
        D3<='1';
        elsif(ALU_SRCd ="101")then
        D1<='0';
        D2<='0';
        D3<='1';
        elsif(ALU_SRCd ="110")then
        D1<='1';
        D2<='1';
        D3<='1';
        elsif(ALU_SRCd ="111")then
        D1<='0';
        D2<='1';
        D3<='1';
end if;
end if;
end process;
end archdecoder;

------------------Entity MUX2_1----------------------
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity MUX2_1 is
port(in1_1,in1_2 :in std_logic_vector(15 DOWNTO 0);
sl1:in std_logic;
out1:out std_logic_vector(15 DOWNTO 0));
end MUX2_1;
architecture archMUX2_1 of MUX2_1 is
begin
process(in1_1,in1_2,sl1)
begin
if(sl1='0')then
out1<=in1_1;
elsif(sl1='1')then
out1<=in1_2;
end if;
end process;
end archMUX2_1;
```

REGISTER to ALU simulation



When alu_src=”001”,

Ain and Bin of ALU = Reg A

Ain=2, Bin=2, for Unary Operation

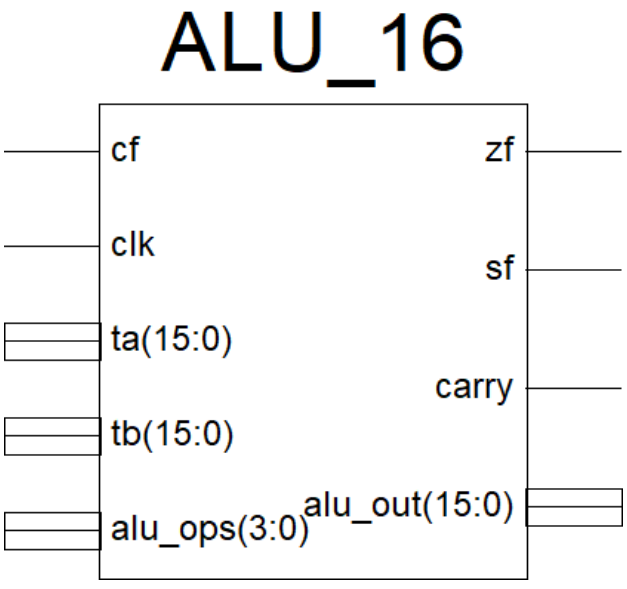
When alu_src=”100”,

Ain =Reg A, Bin = Reg B

Ain=0, Bin=2, for Binary Operation

ALU: This component is responsible to perform all the arithmetic and logical operations for the processor. The ALU performs**,**

Arithmetic ops ADD SUB MULT INCR DECR CLR	Signed, Unsigned, 16-bit, 32-bit
Logical Ops AND OR NOT XOR	16-bit, 32-bit
Shift Ops Logical Arithmetic	16-bit, 32-bit Signed, Unsigned, 16-bit, 32-bit



VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

entity ALU_16 is
port(ta,tb :in std_logic_vector(15 downto 0);
cf,clk : in std_logic;
alu_ops:in std_logic_vector(3 downto 0);
zf,sf,carry: out std_logic;
alu_out:out std_logic_vector(15 downto 0));
end ALU_16;

architecture arch_ALU of ALU_16 is
signal temp_aluout :std_logic_vector(15 downto 0):="0000000000000000";
signal xc: std_logic;
begin
process(clk,ta,tb,cf)
begin
if(rising_edge(clk))then
---ADD----
if(alu_ops="0000")then
        temp_aluout<=ta+tb;
        if(not ta)<tb then xc<='1';end if;
---SUB----
```

```
elsif(alu_ops="0001")then
        temp_aluout<=ta+(not tb)+1;
        if ta<tb then xc<='1';end if;
---AND----
elsif(alu_ops="0010")then
        for i in 0 to 15 loop
        temp_aluout(i)<=ta(i) and tb(i);
        end loop;
---OR-----
elsif(alu_ops="0011")then
        for i in 0 to 15 loop
        temp_aluout(i)<=ta(i) or tb(i);
        end loop;
---XOR----
elsif(alu_ops="0100")then
        for i in 0 to 15 loop
        temp_aluout(i)<=ta(i) xor tb(i);
        end loop;
---PASS---
elsif(alu_ops="0101")then
        temp_aluout<=temp_aluout;
---NOT----
elsif(alu_ops="0110")then
        for i in 0 to 15 loop
        temp_aluout(i)<=not ta(i);
        end loop;
---SHLCF---
```

```
elsif(alu_ops="0111")then
        temp_aluout<= ta(14 downto 0) & cf;
        xc<=ta(15);
---SHRCF---
elsif(alu_ops="1000")then
        temp_aluout<= cf & ta(15 downto 1);
        xc<=ta(0);
---INCR----
elsif(alu_ops="1001")then
        temp_aluout<=ta+x"0001";
        if temp_aluout = x"0000" then  xc<='1'; end if;
---DECR----
elsif(alu_ops="1010")then
        temp_aluout<=ta-x"0001";
        if temp_aluout = x"FFFF" then  xc<='1';end if;
---CLEAR---
elsif(alu_ops="1011")then
        temp_aluout<="0000000000000000";
end if;
alu_out<=temp_aluout(15 downto 0);
if(temp_aluout=x"0000")then
zf<='1';
else
zf<='0';
end if;
if(temp_aluout(15)='1')then
sf<='1';
```

```
else
sf<='0';
end if;
carry<=xc;
end if;
end process;
end arch_ALU;
```

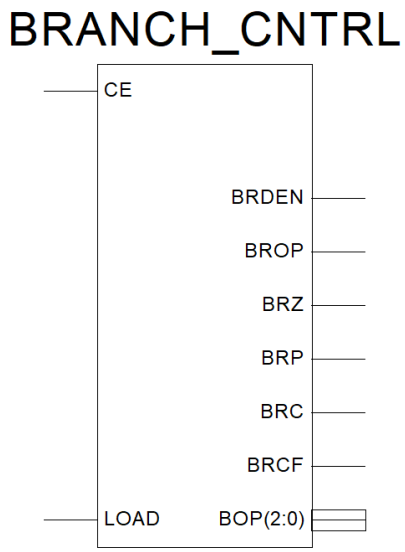
ALU SIMULATION:

Alu_out= ta+tb, when Alu_ops="0000"

Alu_out= Shift right with CF (ta), when Alu_ops="1000"

Alu_out= not ta, when Alu_ops="0110"

BRANCH CONTROL: This component determines which branch operation to perform from the microcode. The control signal to the component is the BOP from the Microprogram (ROM).



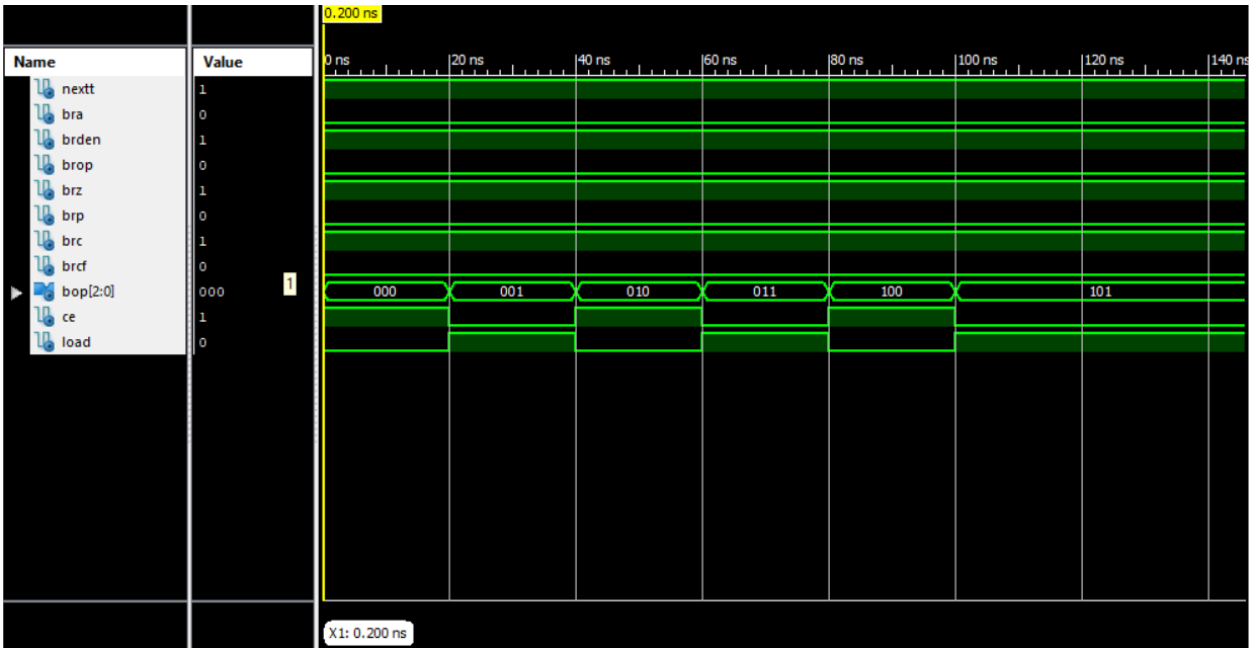
VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity BRANCH_CNTRL is
port(BRDEN,BROP,BRZ,BRP,BRC,BRCF: in std_logic;
BOP: in std_logic_vector(2 downto 0);
CE,LOAD :out std_logic);
end BRANCH_CNTRL;
```

```
architecture Behavioral of BRANCH_CNTRL is
begin
process(BOP)
begin
if(BOP = "000") then
        CE<='1';
        LOAD<='0';
elsif(BOP ="001") then
        CE<='0';
        LOAD<='1';
elsif(BOP ="010") then
        CE<=BRDEN;
        LOAD<=NOT BRDEN;
elsif(BOP ="011") then
        CE<=BROP;
        LOAD<=NOT BROP;
elsif(BOP ="100") then
        CE<=BRZ;
        LOAD<=NOT BRZ;
elsif(BOP ="101") then
        CE<=BRP;
        LOAD<=NOT BRP;
elsif(BOP ="110") then
        CE<=BRC;
        LOAD<=NOT BRC;
elsif(BOP ="111") then
        CE<=BRCF;
```

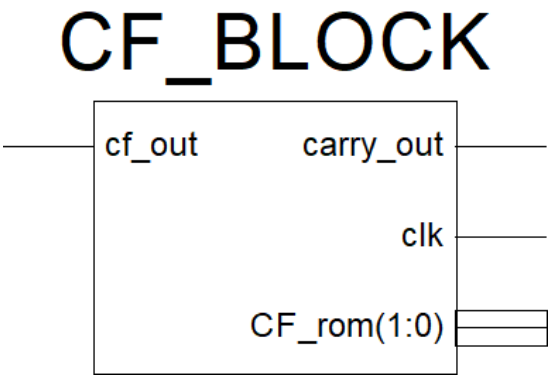
```
        LOAD<=NOT BRCF;
end if;
end process;
end Behavioral;
```

BRANCH CONTROL SIMULATION



BOP (2)	BOP (1)	BOP (0)	
0	0	0	NEXT
0	0	1	BRA
0	1	0	BRDEN
0	1	1	BROP
1	0	0	BRZ
1	0	1	BRP
1	1	0	BRC
1	1	1	BRCF

CF BLOCK: This component performs the carry flag set, carry, clear, No change operations.

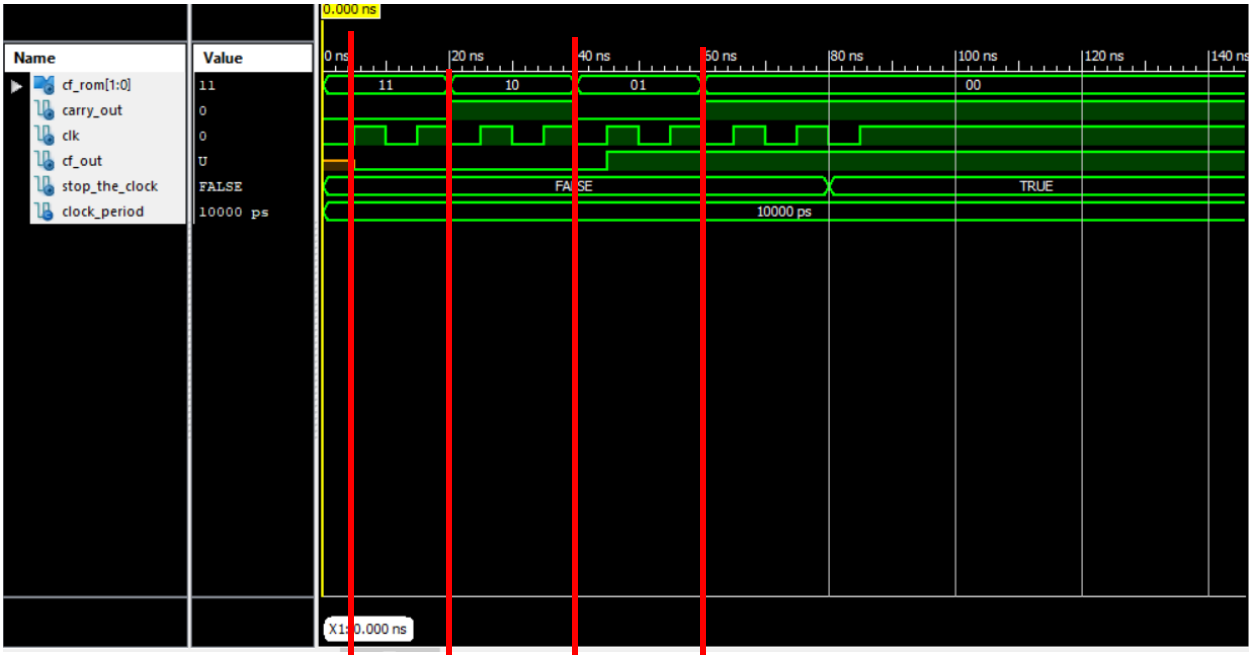


VHDL CODE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity CF_BLOCK is
port(CF_rom :in std_logic_vector(1 downto 0);
        carry_out,clk: in std_logic;
        cf_out:inout std_logic);
end CF_BLOCK;
architecture Behavioral of CF_BLOCK is
begin
process(clk,CF_rom,carry_out)
begin
if rising_edge(clk) then
if(CF_rom = "00") then
```

```
        cf_out<=cf_out;
elsif(CF_rom = "01") then
        cf_out<='1';
elsif(CF_rom = "10") then
        cf_out<='0';
elsif(CF_rom = "11") then
        cf_out<=carry_out;
end if;
end if;
end process;
end Behavioral;
```

CF SIMULATION



When Bit 14,13 is “11”

CF_OUT=CARRY_OUT

When Bit 14,13 is “10”

CF_OUT=CLR (0)

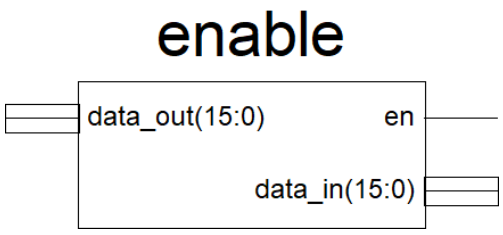
When Bit 14,13 is “01”

CF_OUT=SET (1)

When Bit 14,13 is “00”

CF_OUT=No change

ENABLE: To enable ALU output onto the data bus



VHDL CODE:

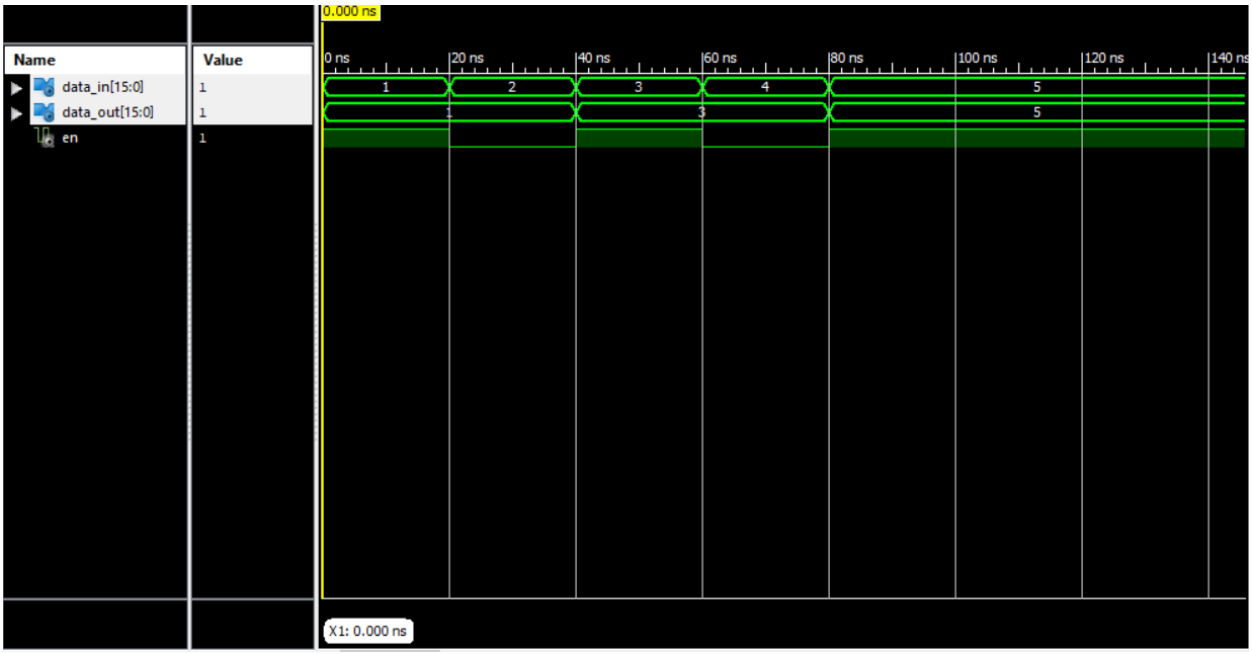
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;
entity enable is
port(data_in:in std_logic_vector(15 downto 0);
        data_out:inout std_logic_vector(15 downto 0);
        en :in std_logic);
end enable;
architecture Behavioral of enable is
begin
process(en)
begin
if(en = '1')then
data_out<=data_in;
elsif(en = '0')then
data_out<=data_out;
```

end if;

end process;

end Behavioral;

ENABLE SIMULATION



MICROCODE

ADD 32 u

Offset	Alu_dest	Alu_src	Alu_ops	Cf	Reg_in	Bop	Db	Addr	Description
10	XX	XX	XX	NC	DB	BRDBN	IN	10	Wait for 1stlow 16-bit
11	L	XX	XX	NC	DB	NEXT	IN	XX	DB to L
12	XX	XX	XX	NC	DB	BRDBN	IN	12	Wait for 1sthigh 16-bit
13	A	XX	XX	NC	DB	NEXT	IN	XX	DB to A
14	XX	XX	XX	NC	DB	BRDBN	IN	14	Wait for 2nd low 16-bit
15	B	XX	XX	NC	DB	NEXT	IN	XX	DB to B
16	XX	XX	XX	NC	DB	BRDBN	IN	16	Wait for 2nd high 16-bit
17	C	XX	XX	CLR	DB	NEXT	IN	XX	DB to C
18	B	LB	ADD	CAR	ALU	NEXT	OUT	XX	B+L+CF to B
19	C	AC	ADD	CAR	ALU	BRA	OUT	00	C+A+CF to C & go to “00”

ADD 16 s

Offset	Alu_dest	Alu_src	Alu_ops	Cf	Reg_in	Bop	Db	Addr	Description
20	XX	XX	XX	NC	DB	BRDBN	IN	20	Wait for 1st data
21	A	XX	XX	NC	DB	NEXT	IN	XX	DB to A
22	XX	XX	XX	NC	DB	BRDBN	IN	22	Wait for 2nd data
23	B	XX	XX	NC	DB	BRDBN	IN	XX	DB to B
24	C	AB	XOR	CLR	ALU	BRP	IN	26	If AB>0 go to “26” Else next
25	A	AB	ADD	CAR	ALU	BRA	OUT	00	A+B+CF to A& go to “00”
26	A	AB	ADD	CAR	ALU	NEXT	OUT	XX	A+B+CF to A & next
27	C	AC	XOR	NC	ALU	BRP	OUT	00	If AC>0 go to “00” Else next
28	X	X	XX	NC	ALU	BRA	OUT	F0	Overflow, go to “F0”

MULT 16u

Offset	Alu_dest	Alu_src	Alu_ops	Cf	Reg_in	Bop	Db	Addr	Description
30	PASS	PASS	PASS	NC	DB	BRDBN	IN	30	Wait for m'ier
31	C	C	PASS	NC	DB	NEXT	IN	XX	DB to C (m'ier)
32	PASS	PASS	PASS	NC	DB	BRDBN	IN	32	Wait for m'cnd
33	A	A	PASS	NC	DB	NEXT	IN	XX	DB to A (m'cnd)
34	PASS	PASS	PASS	NC	DB	BROP	OUT	34	Wait for op counter
35	L	L	PASS	NC	DB	NEXT	OUT	10	OP to L(counter)
36	B	B	CLR	CLR	ALU	NEXT	IN	XX	Clear B
37	L	L	DECR	CLR	ALU	BRZ	IN	3E	If L-1=0 goto "3E"
38	B	B	SHLCF	CAR	ALU	NEXT	IN	XX	B to B(14:0)+'0', CF=B(15)
39	C	C	SHLCF	CAR	ALU	BRC	IN	3B	C□C(14:0)+CF,CF=C(15) If Carry=1 goto "3B"
3A	L	L	PASS	CLR	ALU	BRA	IN	37	Clear CF, goto "37"
3B	B	AB	ADD	CAR	ALU	BRC	IN	3D	B to A+B If Carry=1 goto "3D"
3C	C	C	PASS	CLR	ALU	BRA	IN	37	Clear CF, goto "37"
3D	C	C	INCR	CLR	ALU	BRA	IN	37	C+1 to C, goto "37"
3E	L	L	PASS	CLR	ALU	BRA	IN	3E	HOLD

ABS 32 s

Offset	Alu_dest	Alu_src	Alu_ops	Cf	Reg_in	Bop	Db	Addr	Description
40	XX	XX	XX	NC	DB	BRDBN	IN	40	Wait for 1st data
41	A	XX	XX	NC	DB	NEXT	IN	XX	DB to A
42	XX	XX	XX	NC	DB	BRDBN	IN	42	Wait for 2nd data
43	B	XX	XX	NC	DB	NEXT	IN	XX	DB to B
44	L	XX	CLR	NC	ALU	NEXT	OUT	XX	Clear L
45	C	B	PASS	NC	ALU	BRP	OUT	00	B to C, If C>0 goto “00” Else next
46	B	B	NOT	NC	ALU	NEXT	OUT	XX	~B to B
47	A	A	NOT	CLR	ALU	NEXT	OUT	XX	~A to A
48	A	A	INC	CAR	ALU	NEXT	OUT	XX	A+1 to A
49	B	LB	ADD	CAR	ALU	BRA	OUT	00	B+L+CF to B& goto “00”