Univerzitet u Novom Sadu
FAKULTET TEHNIČKIH NAUKU
Departman za Elektroniku
Mikroprocesorska Elektronika

PROJEKTOVANJE ARM PROCESORA (VHDL)

Mentor:
Ivan Mezei

Studenti:
Nemanja Sibinčić
Aleksandar Komazec
Milan Božić

Uvod

Tokom projekta biće prikazani osnovni i napredniji aspekti VHDL-a kao i projektovanje ARM(**Advanced RISC Machines**) procesora koji svoje instrukcije izvršava u jednom taktu, takozvani **Single Cycle.**Zatim će biti izvršena verifikacija zadatog procesora.

Takođe sa aspekta broja taktova u kojima ARM izvršava zadate inskrukcije postoje i ARM procesori čije se instrukcije izvršavaju u više taktova za redom ili čak i paralelno.

Cilj ovog projekta je da članovi tima pokažu svoja znanja iz oblasti VHDL-a kao i osnovna znanja o procesorima opšte i unapred određene namene.Tokom projekta biće opisana detaljna funkcionalnost svakog bloka,njegova verifikacija i nacin rada čitavog sistema.
Procesor koji se obrađuje unutar ovog projekat je delimično obrađen na predavanjima.
Kompletna funkcionalnost sledi u ovom elaboratu.

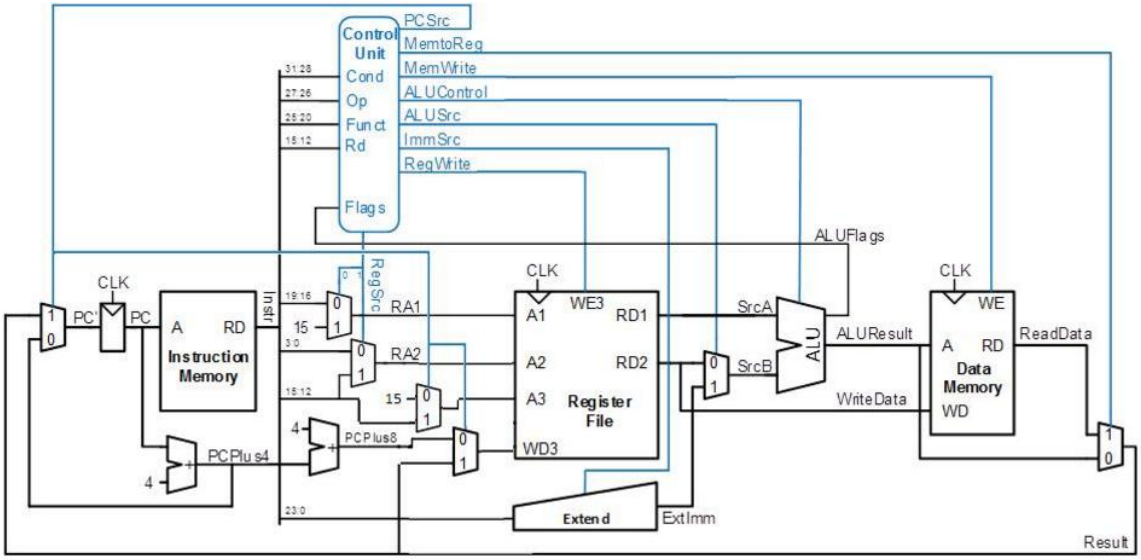
OPŠTI PRIKAZ ARM-a i podela procesora

1.Data Path koga čine sledeći blokovi:

-Multiplekseri 2 na 1 ->7
-Dvoulazni sabirači ->2
-Registri opšte namene -> 1
-Memorijski blokovi tipa RAM ->2
-Memorijski blokovi tipa Rom ->1
-Blok za proširenje vrednosti ->1

2.Control Unit koga čine sledeći blokovi

-Kontrolna jedinica je urađena preko memorijskog bloka (ROM)

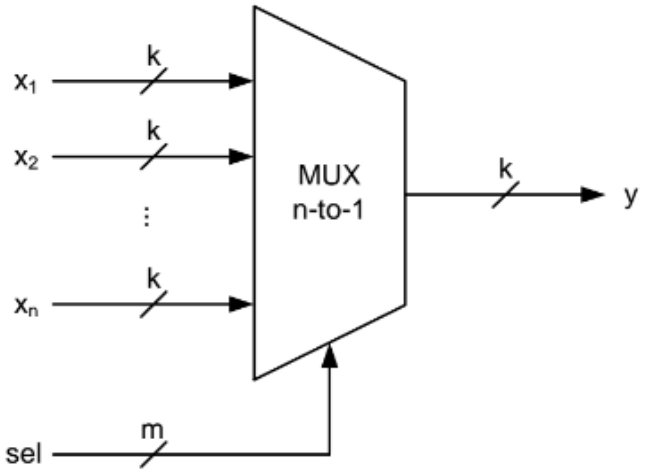


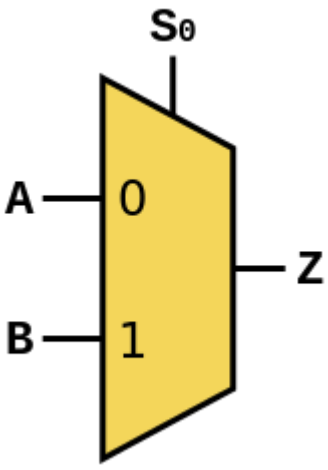
Slika 1: Blok šema ARM_SC procesora

DETALJAN PRIKAZ BLOKOVA ARM-a

Multiplekser

Multiplekser je kombinaciona mreža sa n ulaza za podatke, x_1, x_2, ..., x_n, jednog selekcionog ulaza, *sel*, i jednim izlazom za podatke, y. Širina svakog od n ulaza za podatke je jednaka i iznosi k bita. Širina izlaza za podatke mora biti jednaka širini ulaza za podatke, k bita. Širina selekcionog ulaza jednaka je m bita i mora biti dovoljno velika da omogući jednoznačnu selekciju svakog od raspoloživih n ulaza za podatke. Interfejs generičkog multipleksera prikazan je na slici 1.





Slika 2:Multiplekser 2 na1 ,blok šema

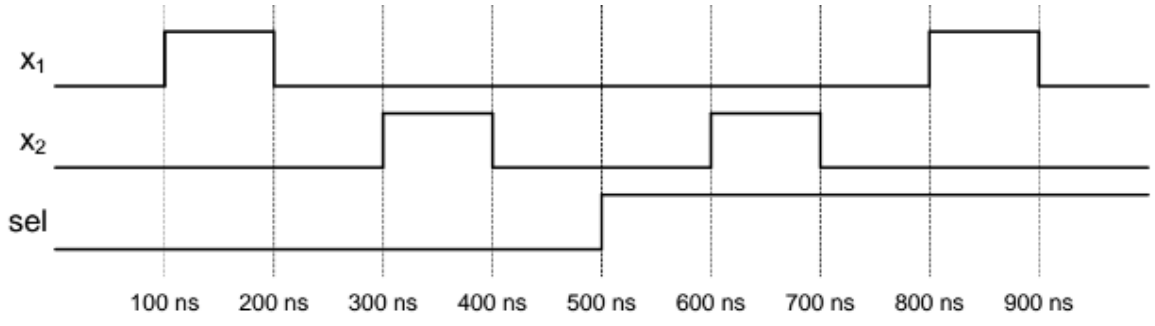
VHDL kod Multipleksera 2 na 1:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity demux is
    Port ( x1 : in  STD_LOGIC_VECTOR (3 downto 0);
                            x2 : in  STD_LOGIC_VECTOR (15 downto 12);
           sel : in  STD_LOGIC;
           outp : out  STD_LOGIC_VECTOR (3 downto 0));
end demux;




architecture Behavioral of demux is
begin
outp <= x1 when sel='0' else x2;

end Behavioral;
```

TEST BENCH

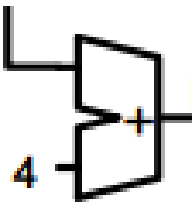
```
entity mux2na1_tb is
end entity mux2na1_tb;

architecture beh of mux2na1_tb is

-- Deklaracija komponente VHDL modula koji se verifikuje (DUV)

-- U ovom slučaju je to multiplekser 2-na-1
component mux2na1 is

                        port (x1: in std_logic; -- ulazni port podataka 1
                        x2: in std_logic; -- ulazni port podataka 2
                        sel: in std_logic; -- selekcioni ulaz
                        y: out std_logic);-- izlazni port podataka

end component mux2na1;


-- Deklaracija unutrašnjih signala potrebnih za povezivanje stimulus
```

```
-- generatora sa ulazima DUV-a

                signal x1_s, x2_s, sel_s: std_logic;
                signal y_s: std_logic;

begin

-- Komponenta koja se verifikuje

duv: mux2na1

port map (
        x1 => x1_s,
        x2 => x2_s,
        sel => sel_s,
        y => y_s);

-- Stimulus generator koji generise potrebne vrednosti na

-- ulaznim portovima DUV-a na osnovu kojih ce biti moguce

-- proveriti da li DUV implementira potrebnu funkcionalnost

stim_gen: process

begin

x1_s <= ‘0’, ‘1’ after 100 ns, ‘0’ after 200 ns,‘1’ after 800 ns, ‘0’ after 900 ns;
x2_s <= ‘0’, ‘1’ after 300 ns, ‘0’ after 400 ns, ‘1’ after 600 ns, ‘0’ after 700 ns;
sel_s <= ‘0’, ‘1’ after 500 ns;

wait;
end process;
end architecture beh;
```



Slika3: Talasni oblik multipleksera tokom simulacije u VHDL-u.

Sabirač



Slika3:Dvoulazni sabirač

Sabiračko kolo će nam služiti za uvećavanje PC registra za 4 kako bi sledeca instrukcija na koju ce pokazivati PC bila sledeca instrukcija u InstRegistru.

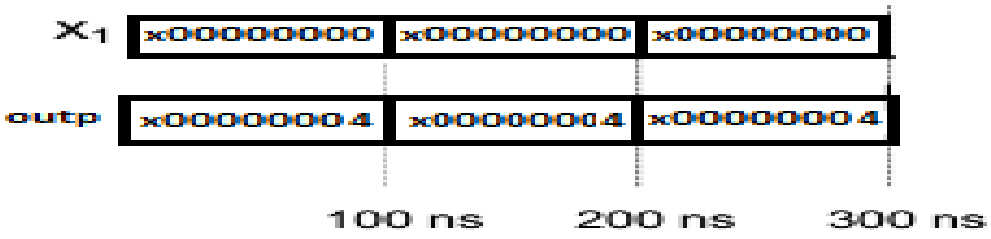
VHDL kod

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

entity Addr is
    Port ( x1 : in  STD_LOGIC_VECTOR (31 downto 0);
           outp : out  STD_LOGIC_VECTOR (31 downto 0));
end Addr;

architecture Behavioral of Addr is
begin

        outp <= x1 + x"00000004";
end Behavioral;
```

TESTBENCH

```
entity mux2na1_tb is
end entity mux2na1_tb;

architecture beh of mux2na1_tb is

component mux2na1 is

                port ( x1 : in  STD_LOGIC_VECTOR (31 downto 0);
                      Outp: out STD_LOGIC_VECTOR(31 downto 0) );
```

```
end component mux2na1;

signal x1_s,outp_s: std_logic;

begin

                port map (
                        x1 => x1_s,
                        outp => outp_s);

stim_gen: process
begin
                x1_s <= '0';
wait;
end process;
end architecture beh;
```



Registri opšte namene

Memorije

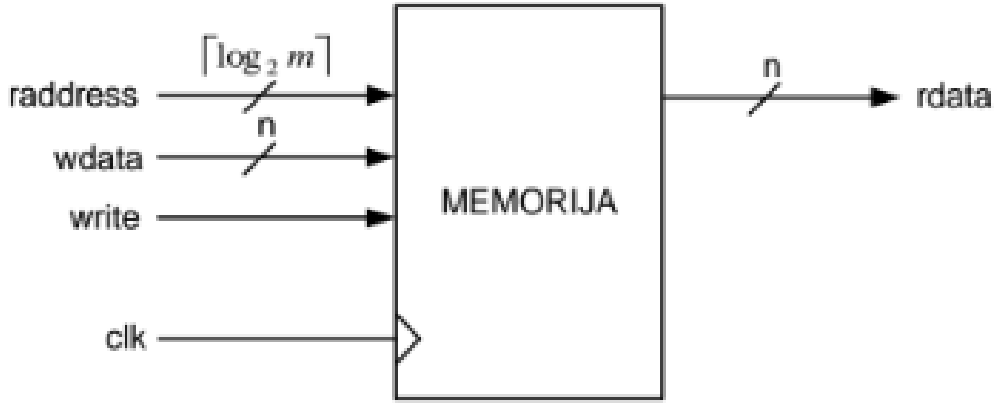
Memorija predstavlja veliki broj individualnih registara organizovanih u jednu celinu kojima se pristupa preko zajedničkih portova. U tom smislu memorija se može posmatrati kao velika registrska banka. Ovo je samo donekle tačno, jer postoje bitne razlike između memorije i registarske banke:

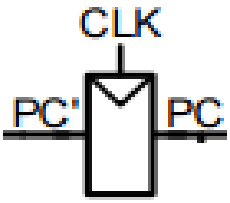
- Kapacitet memorije je znatno veći od kapaciteta registarske banke. Kapacitet registrske banke iznosi nekoliko desetina registara (tipično je reč o 32 registra), dok se kapacitet memorije meri milionima, pa i milijardama registara.
- Registarska banka po pravilu ima veći broj pristupa za upis i čitanje podataka (tipično 2 pristupa za čitanje i 1 pristup za upis), dok memorija gotovo uvek ima samo 1 pristup koji se koristi i za čitanje i za upis. U jednom trenutku može se ili čitati sadržaj iz memorije ili vršiti upis novog sadržaja, ali nije moguć istovremeni upis i čitanje.
- Tehnološki, registarska banka je gotovo uvek „izgrađena" od flip flopova. U slučaju memorija tehnologija izrade memorijskih ćelija može biti raznolika. Upravo zbog ovoga, individualni registri od kojih je memorija sastavljena se u slučaju memorije ne nazivaju registrima već *memorijskim lokacijama*.

Takođe, za razliku od registarskih banki u koje je uvek moguće i upisivati i čitati podatke, memorije se mogu podeliti u dve veliku grupe u zavisnosti od toga da li je upis podataka moguć ili nije:

- RAM memorije – memorije kod kojih je moguć upis podataka u memoriju tokom rada
- ROM memorije – memorije kod kojih je moguće samo čitati podatke

Razmotrimo prvo RAM memorije. Osnovni interfejs RAM memorije sastavljene od m n-bitnih lokacija prikazan je na slici 1.





Slika 4:Registar PC sa Clock-om koji se koristi u šemi

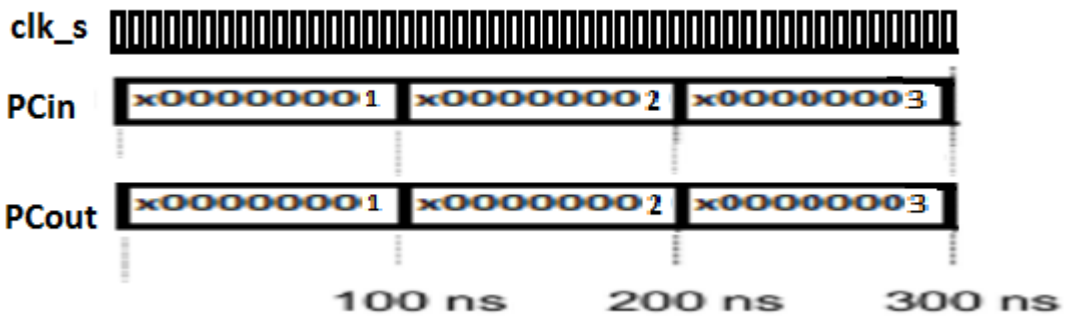
VHDL kod

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PCreg is
    Port ( PCin : in  STD_LOGIC_VECTOR (31 downto 0);
           PCout : out  STD_LOGIC_VECTOR (31 downto 0);
           clk : in  STD_LOGIC
           --reset : in  STD_LOGIC
                        );
end PCreg;
architecture Behavioral of PCreg is
           --signal temp: std_logic_vector(31 downto 0);
begin
process (clk) is
           begin
                    if(clk'event and clk='1' ) then
                                --temp<=PCin;
                                PCout<=PCin;       else
                            --        PCout<=temp;
                            PCout <= (others => '0');
                    end if;

end process;

end Behavioral;
```

TESTBENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;

-- entity declaration for your testbench.Dont declare any ports here
ENTITY test_tb IS
END test_tb;

ARCHITECTURE behavior OF test_tb IS
   -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT test  --'test' is the name of the module needed to be tested.
--just copy and paste the input and output ports of your module as such.
    PORT(
          PCin : in  STD_LOGIC_VECTOR (31 downto 0);
         PCout : out  STD_LOGIC_VECTOR (31 downto 0);
         clk : in  STD_LOGIC

       );
    END COMPONENT;

   --declare inputs and initialize them
   signal clk_s : std_logic := '0';
   Signal PCin_s: std_logic_vector(31 downto 0):= (others=>x"00000000");

 --declare outputs and initialize them
   signals PCout_: std_logic_vector(31 downto 0);

-- Clock period definitions
   constant clk_period : time := 10 ns;
BEGIN
    -- Instantiate the Unit Under Test (UUT)
   uut: test PORT MAP (
        clk => clk_s,
        PCin=>PCin_s.
        PCout=> PCout_s

       );

   -- Clock process definitions( clock with 50% duty cycle is generated here.
   clk_process :process
   begin
```

```
        clk <= '0';
        wait for clk_period/2;  --for.5 ns signal is '0'.
        clk <= '1';
        wait for clk_period/2;  --for next 5 ns signal is '1'.
    end process;

    -- Stimulus process
   stim_proc: process
    begin
PCin_s<=x”00000001”, x”00000002” after 100 ns, x”00000003 ”after 200 ns ;
        wait;
   end process;

END;
```



Memorijski blokovi tipa RAM

RAM memorija u opštem slučaju poseduje sledeće portove:

- 1 pristupa za čitanje ili upis podataka upisanih u memorijske lokacije,
- *clk* ulaz za sinhronizaciju rada registarske banke

Opciono, RAM memorija može posedovati i sledeće portove:

- *reset* ulazni port za inicijalizaciju sadržaja memorijskih lokacija
- *clock enable* ulazni port za selekciju rastućih ivica *clk* porta na koje memorija treba da se aktivira
- *enable* ulazni port za dozvolu rada memorije

Pristup za čitanje ili upis podataka sastoji se iz sledećih portova:

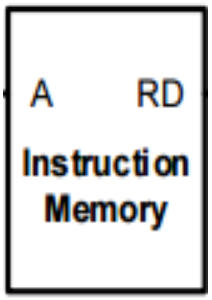
- *address*, ulazne adresne magistrale koja služi za adresiranje memorijske lokacije čiji sadržaj se želi pročitati ili se u nju želi upisati novi sadržaj; moguće vrednosti adresa su iz opsega [0 *m*-1] celih brojeva
- *rdata*, izlazne magistrale podataka preko koje se dobija trenutni sadržaj adresirane memorijske lokacije
- *wdata*, ulazne magistrale podataka preko koje se prosleđuje podatak koji je potrebno upisti u adresiranu memorijsku lokaciju
- *write*, ulaznog porta dozvole upisa novog podatka u adresiranu memorijsku lokaciju
- *read*, opcioni ulazni port koji inicira proces čitanja podatka iz adresirane memorijske lokacije

Kao i u slučaju registarskih banki, u zavisnosti od trenutka pojavljivanja sadržaja adresirane memorijske lokacije na *rdata* portu razlikujemo dva načina čitanja podataka iz memorije:

- **sinhrono čitanje** - kod kojega se sadržaj adresirane memorijske lokacije pojavlje na *rdata* portu prilikom nailaska sledeće rastuće ivice *clk* signala,
- **asinhrono čitanje** - kod kojega se sadržaj adresirane memorijske lokacije pojavljuje odmah nakon stabilizacije nove adrese na *raddress* portu.

Upis podataka u memoriju vrši se preko pristupa za upis. Postavljanjem adrese registra u koji želimo upisati podatak na *address* ulazni port, adresira se željena memorijska lokacija. Podatak koji se želi upisati u adresiranu memorijsku lokaciju postavlja se na *wdata* ulazni port. Sam upis podatka je uvek sinhroni sa *clk* signalom, i aktivira se kada se *write* ulazni port postavi na vrednost 1.

Na primer, vremenski dijagram rada 8-bitne RAM memorije kapaciteta 1024 lokacije, sa asinhronim čitanjem, prikazan je na slici 2.



Slika 5: Instrukcioni registar,blok šema

Unutar ovog registra će se nalaziti instrukcije koje procesor izvršava.

VHDL KOD

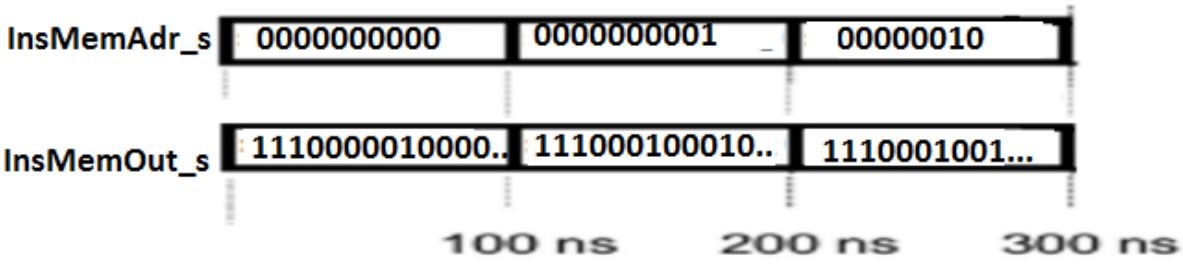
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity InsMem is
port (
InsMemAdr : in std_logic_vector(9 downto 0);
InsMemOut: out std_logic_vector(31 downto 0)
);
end InsMem;


architecture Behavioral of InsMem is

type InsMemorija is array (0 to 1023) -- 2^31-1
of std_logic_vector(31 downto 0);

constant Memorija: InsMemorija:=(
                "11100000100001100101000000000111",--add
                "11100010001001010000000000000101",--sub5-5
                "11100010010001010001000000001010",--and5i10
                "11100010011001010010000000000101",--or5i5
                "11100100000001010011000000011010",--mem imm, podatak 5, destinacija 3,
store
                "11100100000101110100000000010011",--mem imm, podatak 7, destinacija 4, load
                "11101110000000000000000000000011",--branch na 3ce mesto u memoriji
                others => (x"00000000")
                );
begin
                InsMemOUT<=Memorija(conv_integer(unsigned(InsMemAdr)));
```

```
end Behavioral;
```

TESTBENCH

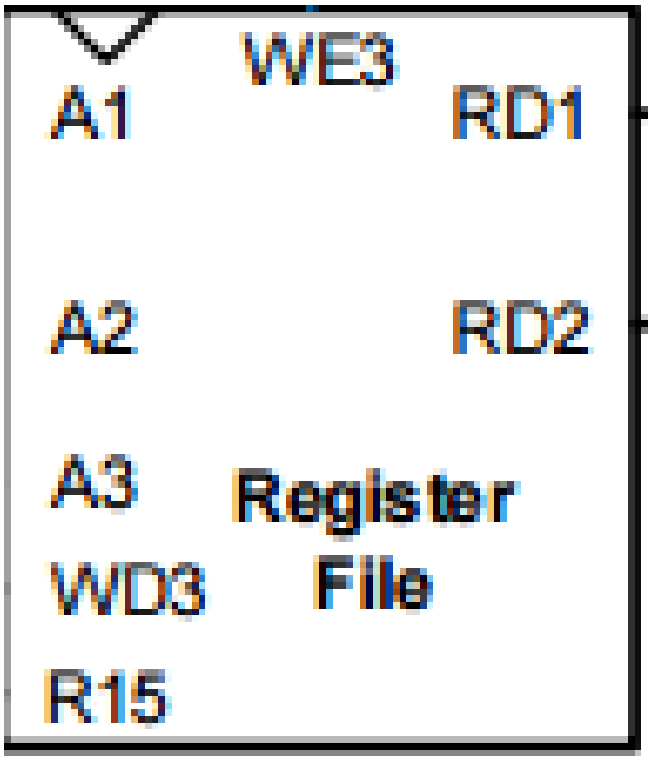
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;

-- entity declaration for your testbench.Dont declare any ports here
ENTITY test_tb IS
END test_tb;

ARCHITECTURE behavior OF test_tb IS
   -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT test  --'test' is the name of the module needed to be tested.
--just copy and paste the input and output ports of your module as such.
    PORT(
       InsMemAdr : in std_logic_vector(9 downto 0);
      InsMemOut: out std_logic_vector(31 downto 0)

        );
    END COMPONENT;

   --declare inputs and initialize them
   signal InsMemAdr_s: std_logic_vector :=”0000000000”;


 --declare outputs and initialize them
   signal InsMemOut_s: std_logic_vector(31 downto 0);


BEGIN
    -- Instantiate the Unit Under Test (UUT)
   uut: test PORT MAP (
         InsMemAdr=> InsMemAdr_s.
         InsMemOut=> InsMemOut_s

        );

   -- Clock process definitions( clock with 50% duty cycle is generated here.


   -- Stimulus process
  stim_proc: process
```

```
  begin
InsMemAdr_s<= ”00000001”after 100 ns, ”00000010” after 200 ns;
       wait;
 end process;

END;
```



Register file

Ovaj registar ima tri ulaza za učitavanje unapred zadate adresne lokacije unutar memorije. (A1,A2,A3).Tu se nalazi i ulaz WD3 na koji se dovodi vrednost sa magistrale(podatak). Memorija je sinhrona tako da postoji clock ulaz kao i dozvola za pisanje. Dva izlaza omogućavaju da se sa određene memorijske lokacije dovede vrednost koja se šalje u ostatak kola asinhrono.
.



Slika 6: Blok šema Register File

VHDL KOD

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

entity FileRegistar is
	port (
	clk: in std_logic;
	RW3 : in std_logic;
	RD1 : out STD_logic_vector(31 downto 0);
	RD2 : out STD_logic_vector(31 downto 0);
	A1 : in STD_logic_vector(3 downto 0);
	A2 : in STD_logic_vector(3 downto 0);
	A3 : in STD_logic_vector(3 downto 0);
	WD3 : in STD_logic_vector(31 downto 0)
);
end FileRegistar;

architecture Behavioral of FileRegistar is
	type FR is array (0 to 15) of std_logic_vector(31 downto 0);
	signal FRMem : FR:=(
		X"00000003",--0
		X"00000005",--1
		X"00000000",--2
		X"00000000",--3
		X"00000000",--4
		X"00000000",--5
		X"00000000",--6
		X"00000000",--7
		X"00000000",--8
		X"00000000",--9
		X"00000000",--10
		X"00000000",--11
		X"00000000",--12
		X"00000000",--13
		X"00000000",--14
		X"00000000"--15
	);
```

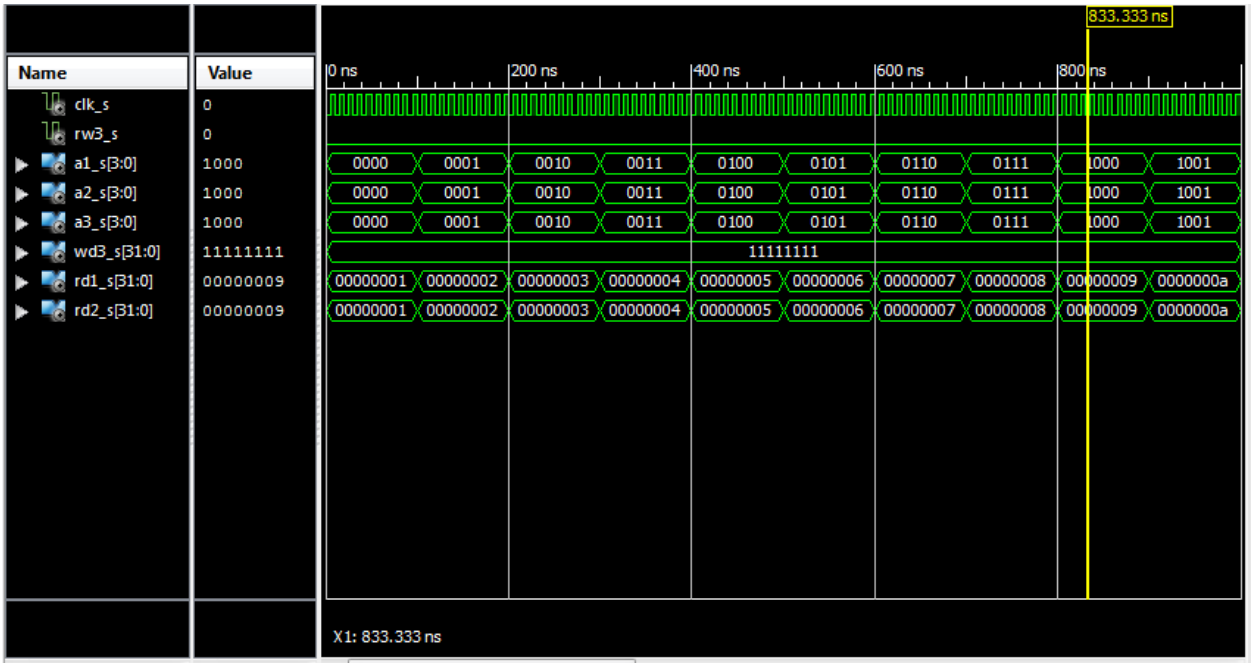
```
begin
	prvi:process(clk,RW3) is
	begin
		if (clk'event and clk='1') then
			if(RW3 ='1') then
				FRmem(conv_integer(unsigned(A3)))<=WD3;
			end if;
		end if;
	end process;
	RD1<=FRMem(conv_integer(unsigned(A1)));
	RD2<=FRMem(conv_integer(unsigned(A2)));
end Behavioral;
```

TESTBENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY RegFile_TB IS
END RegFile_TB;

ARCHITECTURE behavior OF RegFile_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT FileRegistar
    PORT(
         clk : IN  std_logic;
         RW3 : IN  std_logic;
         RD1 : OUT  std_logic_vector(31 downto 0);
         RD2 : OUT  std_logic_vector(31 downto 0);
         A1 : IN  std_logic_vector(3 downto 0);
         A2 : IN  std_logic_vector(3 downto 0);
         A3 : IN  std_logic_vector(3 downto 0);
         WD3 : IN  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk_s : std_logic := '0';
   signal RW3_s : std_logic := '0';
   signal A1_s : std_logic_vector(3 downto 0) := (others => '0');
```

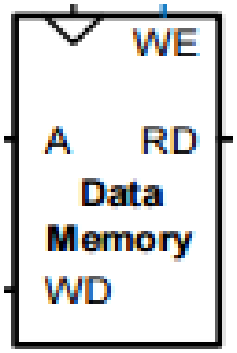
```
   signal A2_s : std_logic_vector(3 downto 0) := (others => '0');
   signal A3_s : std_logic_vector(3 downto 0) := (others => '0');
   signal WD3_s : std_logic_vector(31 downto 0) :=X"11111111";

         --Outputs
   signal RD1_s : std_logic_vector(31 downto 0);
   signal RD2_s : std_logic_vector(31 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

         -- Instantiate the Unit Under Test (UUT)
   uut: FileRegistar PORT MAP (
         clk => clk_s,
         RW3 => RW3_s,
         RD1 => RD1_s,
         RD2 => RD2_s,
         A1 => A1_s,
         A2 => A2_s,
         A3 => A3_s,
         WD3 => WD3_s
       );

   -- Clock process definitions
   clk_process :process
   begin
                  clk_s <= '0';
                  wait for clk_period/2;
                  clk_s <= '1';
                  wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
         A1_s<=X"1" after 100 ns,X"2" after 200 ns,X"3" after 300 ns,X"4" after 400 ns,X"5" after
500 ns,X"6" after 600 ns,X"7" after 700 ns,X"8" after 800 ns,X"9" after 900 ns,X"A" after 1000
ns,X"B" after 1100 ns,X"C" after 1200 ns,X"D" after 1300 ns,X"E" after 1400 ns,X"F" after 1500 ns;
```

```
   A2_s<=X"1" after 100 ns,X"2" after 200 ns,X"3" after 300 ns,X"4" after 400 ns,X"5" after 500
ns,X"6" after 600 ns,X"7" after 700 ns,X"8" after 800 ns,X"9" after 900 ns,X"A" after 1000 ns,X"B"
after 1100 ns,X"C" after 1200 ns,X"D" after 1300 ns,X"E" after 1400 ns,X"F" after 1500 ns;

   A3_s<=X"1" after 100 ns,X"2" after 200 ns,X"3" after 300 ns,X"4" after 400 ns,X"5" after 500
ns,X"6" after 600 ns,X"7" after 700 ns,X"8" after 800 ns,X"9" after 900 ns,X"A" after 1000 ns,X"B"
after 1100 ns,X"C" after 1200 ns,X"D" after 1300 ns,X"E" after 1400 ns,X"F" after 1500 ns;
         --RW3_s<='1' after 20 ns;

      wait;
   end process;

END;
```



Data Memory

Je sinhrona memorija koja će nam služiti da za upis podataka na memorijske lokacije. Preko ulaza A dobijamo informaciju na koju memorijsku lokaciju će se upisati vrednost sa ulaza WD. Kontrolni ulaz WE omogucava upis. Dok se na RD izlazu pojavljuje informacija sa memorijske lokacije A.



VHDL kod

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




entity Datamem is
    Port ( clk : in  STD_LOGIC;
           WE : in  STD_LOGIC;
           A : in  STD_LOGIC_VECTOR (9 downto 0);
           rd : out  STD_LOGIC_VECTOR (31 downto 0);
           wd : in  STD_LOGIC_VECTOR (31 downto 0));
end Datamem;

architecture Behavioral of Datamem is
        type FR is array (0 to 1023) of std_logic_vector(31 downto 0);
        signal FRMem : FR:=(
                X"00000099",--0
                X"00000088",--1
                others => x"00000000"
        );
begin
        process (clk) is
        begin
                if (clk'event and clk='1') then
                        if WE='1' then
```

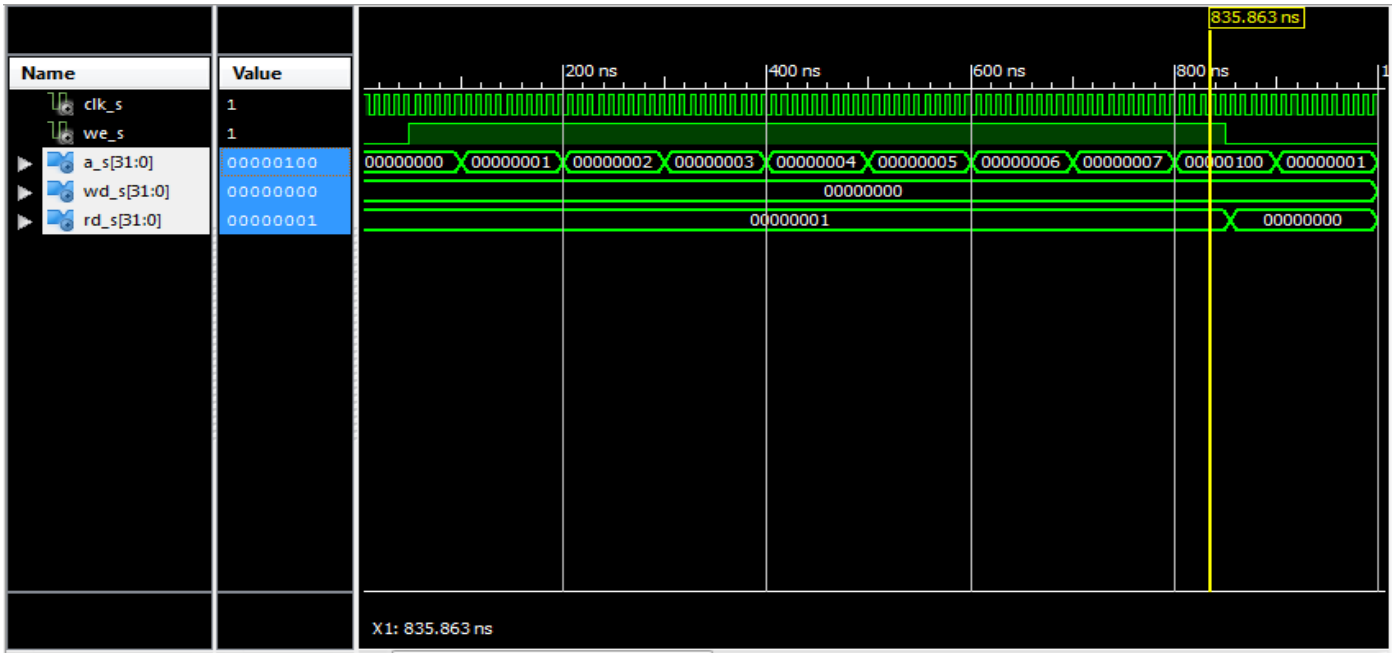
```
                    FRMem(conv_integer(A(9 downto 0))) <= wd;
                end if;
            end if;
        end process;
        rd <= FRMem(conv_integer(A(9 downto 0)));
end Behavioral;
```

TESTBENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


ENTITY Datamem_TB IS
END Datamem_TB;

ARCHITECTURE behavior OF Datamem_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Datamem
    PORT(
         clk : IN  std_logic;
         WE : IN  std_logic;
         A : IN  std_logic_vector(31 downto 0);
         rd : OUT  std_logic_vector(31 downto 0);
         wd : IN  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk_s : std_logic := '0';
   signal WE_s : std_logic := '0';
   signal A_s : std_logic_vector(31 downto 0) := (others => '0');
   signal wd_s : std_logic_vector(31 downto 0) := (others => '0');

        --Outputs
```

```
   signal rd_s : std_logic_vector(31 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: Datamem PORT MAP (
        clk => clk_s,
        WE => WE_s,
        A => A_s,
        rd => rd_s,
        wd => wd_s );

   -- Clock process definitions
   clk_process :process
   begin
                clk_s <= '0';
                wait for clk_period/2;
                clk_s <= '1';
                wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
        A_s<=X"00000001" after 100 ns,X"00000002" after 200 ns,X"00000003" after 300
ns,X"00000004" after 400 ns,X"00000005"after 500 ns ,X"00000006"after 600
ns,X"00000007"after 700 ns,X"00000100"after 800 ns,X"00000001" after 900 ns,X"00000002"
after 1000 ns,X"00000003" after 1200 ns,X"00000004" after 1300 ns,X"00000005"after 1400 ns
,X"00000006"after 1500 ns,X"00000007"after 1600 ns,X"00000100"after 1700 ns;
        WE_s<='1' after 50 ns ,'0' after 850 ns;

      wait;
   end process;
```

```
END;
```



Rezultat simulacije:
Sa ove simulacije može se primetiti ispravnu funkcionalnost date memorije.

Ekstender

Blok koji služi sa proširenje adresnih bitova nulama do 32 bita.
Blok radi konkatenaciju bitova nula na određeni opseg pristigle 32bitne vrednosti.
(8,12,24).

Slika 9: Extendet blok šema

VHDL KOD

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

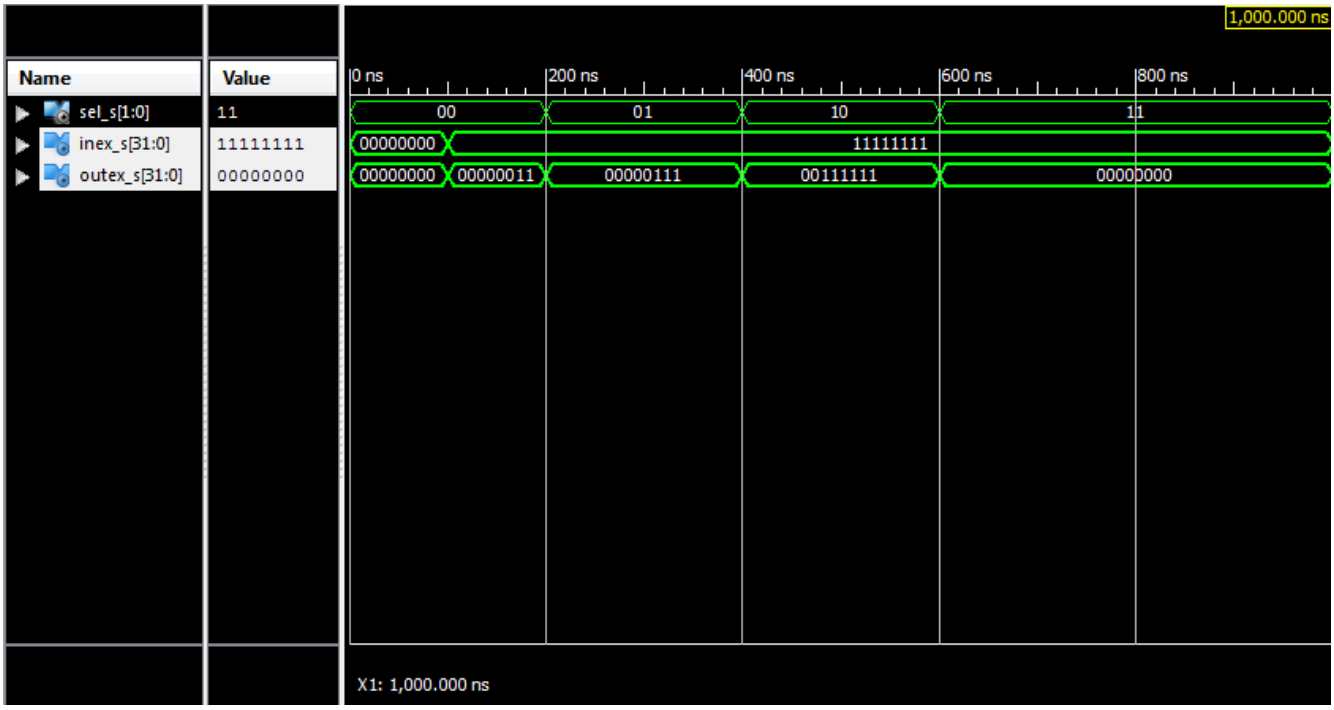
```
entity extender is
    Port ( sel : in  STD_LOGIC_VECTOR (1 downto 0);
           inex : in  STD_LOGIC_VECTOR (31 downto 0);
           outex : out  STD_LOGIC_VECTOR (31 downto 0));
end extender;

architecture Behavioral of extender is
           signal outp : std_logic_vector(31 downto 0);
begin
           process(inex,sel) is
           begin
                    case sel is
                             when "00" => outp <=(x"000000" & inex(7 downto 0)); -- za 8bitni ulaz
                             when "01" => outp <=(x"00000" & inex(11 downto 0)); -- za 12bitni ulaz
                             when "10" => outp <=(x"00" & inex(23 downto 0)); -- za 24bitni ulaz
                             when others => outp<=(others=>'0');
                    end case;

           end process;
           outex <= outp;
end Behavioral;
```

TESTBENCH

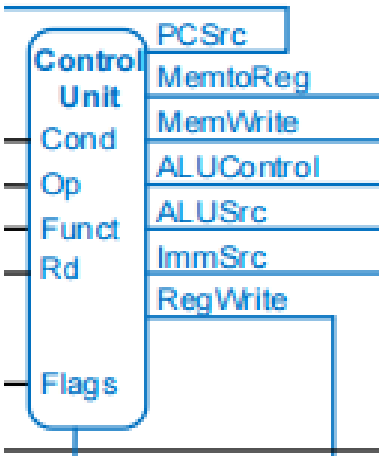
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
  ENTITY EXT_TB IS
END EXT_TB;

ARCHITECTURE behavior OF EXT_TB IS
   COMPONENT extender
    PORT(
         sel : IN  std_logic_vector(1 downto 0);
         inex : IN  std_logic_vector(31 downto 0);
         outex : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;

  --Inputs
   signal sel_s : std_logic_vector(1 downto 0) := (others => '0');
   signal inex_s : std_logic_vector(31 downto 0) := (others => '0');

           --Outputs
   signal outex_s : std_logic_vector(31 downto 0);
```

```
BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: extender PORT MAP (
          sel => sel_s,
          inex => inex_s,
          outex => outex_s
        );
   stim_proc: process
   begin
                        inex_s<=X"11111111" after 100 ns;
                        sel_s<="01" after 200 ns,"10" after 400 ns,"11" after 600 ns;

      wait;
   end process;

END;
```

Rezultat simulacije:



Kontrolna jedinica (Control Unit)



Upravljačka jedinica, **Upravljački sklop** ili **kontrolna jedinica** je deo središnje jedinice (na engleskom central processing unit CPU) koja usmerava radnju i ima izravnu kontrolu preko ostalih delova CPU-a: memorije, aritmetičko-logičke jedinice, instrukcijskih memorija, sabirnica, ulazno/izlazih memorija, spremnik prekida itd registara. Preko izlaznih signala upravljački sklopa usmeruju radnje ostalih delova CPU uređaja.

Upravljačka jedinica je centralni deo mikroprocesora koji reguliše izvršavanje naredbi koji su sastavni deo nekog procesora. Mnogi procesori sastavni su deo većih celina, a i sami procesor se sastoji od raznih delova kao: ALU, registara,PC, memorija stanja, IR i tako dalje. Ovi svi delovi ne mogu raditi u isto vreme već mora postojati neki redosled kojim se ti pod delovi: uključe, isključe i kada koji deo procesora obavlja svoj rad. Postoje razna rešenja kako je izvedena upravljačka jedinica od kojih su dva najzastupljenija riješenja: mikroprogramska i FSM ili ROM. Ta dva spomenuta rešenja uslovljena tehnologijom i vremenom kada su izrađeni neki procesor, i naravno izborima tehnološkog tima prilikom dizajna tog procesora. Kod mikroprograma upravljanjem ostalih delova CPU-a obavlja se izvršavanjem skupa manjih radnji koje su kodirane u mikroprogram. Mikroprogram je pohranjen u kontrolnoj memoriji dok redosled narebi mikrorprograma održava posebna jedinica nazvan mikrosledbenik. Mikrosedbenik izvršava naredbe u mikroprogramu u onom redosledu u kojemu su one napisane.

Naša kontrolna jedinica je napisana u obliku ROM memorije.

VHDL KOD

```
entity cu is
```

```
    Port ( instr : in  STD_LOGIC_VECTOR (5 downto 0);
           flags : in  STD_LOGIC_VECTOR (3 downto 0);
           PCSrc : out  STD_LOGIC;
           MemToReg : out  STD_LOGIC;
           MemWrite : out  STD_LOGIC;
           AluControl : out  STD_LOGIC_VECTOR (1 downto 0);
           AluSrc : out  STD_LOGIC;
           ImmSrc : out  STD_LOGIC_VECTOR (1 downto 0);
           RegWrite : out  STD_LOGIC;
           novimux : out  STD_LOGIC;
                        flagsoutput: out std_logic_vector(3 downto 0);
           RegSrc : out  STD_LOGIC_VECTOR (1 downto 0));
end cu;

architecture Behavioral of cu is
           type FR is array (0 to 15) of std_logic_vector(9 downto 0);
           constant FRMem : FR := (
                                                                  "0000001001",--dpreg
                                                                    "0000001001",--dpreg
                                                                    "0001001001",--dpimm
                                                                    "0001001001",--dpimm
                                                                    "0011010100",--str
                                                                    "0101011000",--ldr
                                                                    "0011010100",--str
                                                                    "0101011000",--ldr
                                                                    "0000000000",
                                                                    "0000000000",
                                                                    "0000000000",
                                                                    "0000000000",
                                                                    "1001100010",--B
                                                                    "1001100010",--B
                                                                    "1001100010",--B
                                                                    "1001100010");--B
           signal code: std_logic_vector(9 downto 0);
begin
           code <= FRMem(conv_integer(instr(5 downto 3) & instr(0))); --op&I&S
           PCSrc<=code(9);
           novimux<=code(9);
           MemToReg<=code(8);
           MemWrite<=code(7);
           ALUSrc<=code(6);
           ImmSrc<=code(5 downto 4);
           RegWrite<=code(3);
```

```
	RegSrc<=code(2 downto 1);
	process(instr)is
	begin
		if(instr(5 downto 4)="00") then
			AluControl<=instr(2 downto 1);
		else
			AluControl<="00";
		end if;
	end process;
	flagsoutput <= flags;
end Behavioral;
```

ORGANIZACIJA DATAPATH-a

```
ibrary IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dp is
	port (clk : in std_logic;
			reset: in std_Logic;
			PCsrc : in std_logic;
			MemToReg: in std_logic;
			MemWrite: in std_logic;
			ALUCONTROL: in std_logic_vector(1 downto 0);
			ALUsrc: in std_logic;
			Immsrc: in std_logic_vector(1 downto 0);
			RegWrite: in std_logic;
			RegSrc: in std_logic_vector(1 downto 0);
			Instr: out std_logic_vector(5 downto 0);
			Flags: out std_logic_vector(3 downto 0);
			novimux: in std_logic;
```

```
                resultdp: out std_logic_vector(31 downto 0)
                );
end dp;

architecture Behavioral of dp is
        signal
pcinp,pcoutp,instroutp,rd1outp,rd2outp,result,PCPlus8,exoutp,SrcB,ALUresult,ReadData,PCPlus4
,WD3input : std_logic_vector(31 downto 0);
        signal a1inp,a2inp,izlnovimux: std_logic_vector(19 downto 16);
begin

        PC: entity work.PCreg (Behavioral) port map (
                PCin => pcinp,
       PCout => pcoutp,
       clk => clk,
                reset =>reset
                );

        IRreg: entity work.InsMem (Behavioral) port map (
                InsMemAdr => pcoutp(9 downto 0),
                InsMemOut => instroutp
                );

        Demux0: entity work.demux0 (Behavioral) port map (
                x1 => instroutp(19 downto 16),
                sel => RegSrc(0),
                outp => a1inp
                );

        Demux1: entity work.demux (Behavioral) port map (
                x1 => instroutp(3 downto 0),
                x2 => instroutp(15 downto 12),
                sel => RegSrc(1),
                outp => a2inp
                );

        RegFile: entity work.FileRegistar (Behavioral) port map (
                clk => clk,
                RW3 => RegWrite,
                RD1 => rd1outp,
                RD2 => rd2outp,
                A1 => a1inp,
                A2 => a2inp,
```

```
                A3 => izlnovimux,
                WD3 => WD3input
                );

  Extender: entity work.extender (Behavioral) port map (
                sel => ImmSrc,
inex => instroutp(23 downto 0),
outex => exoutp
                );

  Demux2: entity work.demux32 (Behavioral) port map (
                x1 => rd2outp,
                x2 => exoutp,
                sel => ALUsrc,
                outp => SrcB
                );

  ALU: entity work.alu (Behavioral) port map (
                ALUa => rd1outp,
                ALUb => SrcB,
                ALUout => ALUresult,
                ALUsel => ALUcontrol,
                ALUflags => Flags
                );

  Datamem: entity work.Datamem (Behavioral) port map (
                clk => clk,
WE => MemWrite,
A => ALUresult(9 downto 0),
rd => ReadData,
wd => rd2outp
                );

  Demux3: entity work.demux32 (Behavioral) port map (
                x1 => ALUresult,
                x2 => ReadData,
                sel => MemToReg,
                outp => result
                );

  Demux4: entity work.demux32 (Behavioral) port map (
                x1 => PCPlus4,
                x2 => result,
```

```
                sel => PCSrc,
                outp => pcinp
                );
        Addr1: entity work.Addr (Behavioral) port map (
                x1 => pcoutp,
                outp => PCPlus4
                );
        Addr2: entity work.Addr (Behavioral) port map (
                x1 => PCPlus4,
                outp => PCPlus8
                );
        Demux5: entity work.demux32 (Behavioral) port map (
                x2 => PCPlus8,
                x1 => result,
                sel => novimux,
                outp => WD3input
                );
        Demux6: entity work.demux (Behavioral) port map (
                x2 => "1111",
                x1 => instroutp(15 downto 12),
                sel => novimux,
                outp => izlnovimux
                );

        resultdp <= result;
        instr <= instroutp(27 downto 25)&instroutp(22 downto 20);
end Behavioral;
```

KOMPLETAN STRUKTURNI MODEL ARM_SC procesora

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity topfile is
    Port ( clk : in std_logic;
                reset: in std_logic;
                result : out  STD_LOGIC_VECTOR (31 downto 0);
                flags : out std_logic_vector (3 downto 0)
                );
end topfile;

architecture Behavioral of topfile is
        signal instrSig :std_logic_vector(5 downto 0);
```

```
        signal flagsSig :std_logic_vector(3 downto 0);
        signal PCSrcSig, memtoregSig, memwriteSig, alusrcSig, regwriteSig, novimuxSig:std_logic;
        signal alucnSig, immsrcSig , regsrcSig :std_logic_vector(1 downto 0);
begin
        CU: entity work.cu port map(
                instr =>  instrSig,
                flags => flagsSig,
                PCSrc => PCSrcSig,
                MemToReg => memtoregSig,
                MemWrite => memwriteSig,
                AluControl => alucnSig,
                AluSrc => alusrcSig,
                ImmSrc => immsrcSig,
                RegWrite => regwriteSig,
                novimux => novimuxSig,
                flagsoutput => flags,
                RegSrc => regsrcSig
        );

        DP: entity work.dp port map(
                        clk => clk,
                        reset => reset,
                        PCsrc => PCSrcSig,
                        MemToReg => memtoregSig,
                        MemWrite => memwriteSig,
                        ALUCONTROL => alucnSig,
                        ALUsrc => alusrcSig,
                        Immsrc => immsrcSig,
                        RegWrite => regwriteSig,
                        RegSrc => regsrcSig,
                        Instr => instrSig,
                        Flags => flagsSig,
                        novimux => novimuxSig,
                        resultdp => result
                        );

end Behavioral;
```

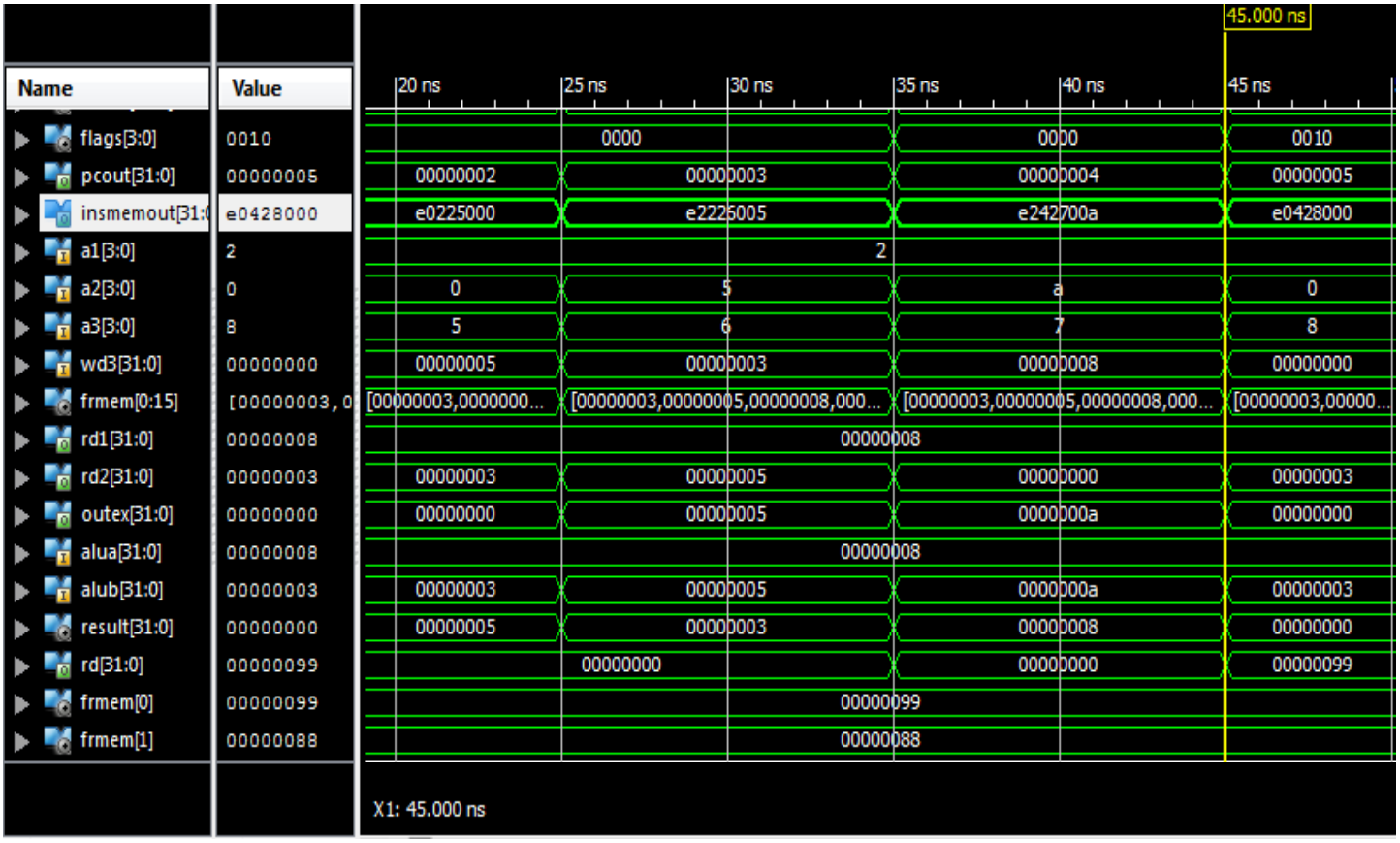
TESTBENCH

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

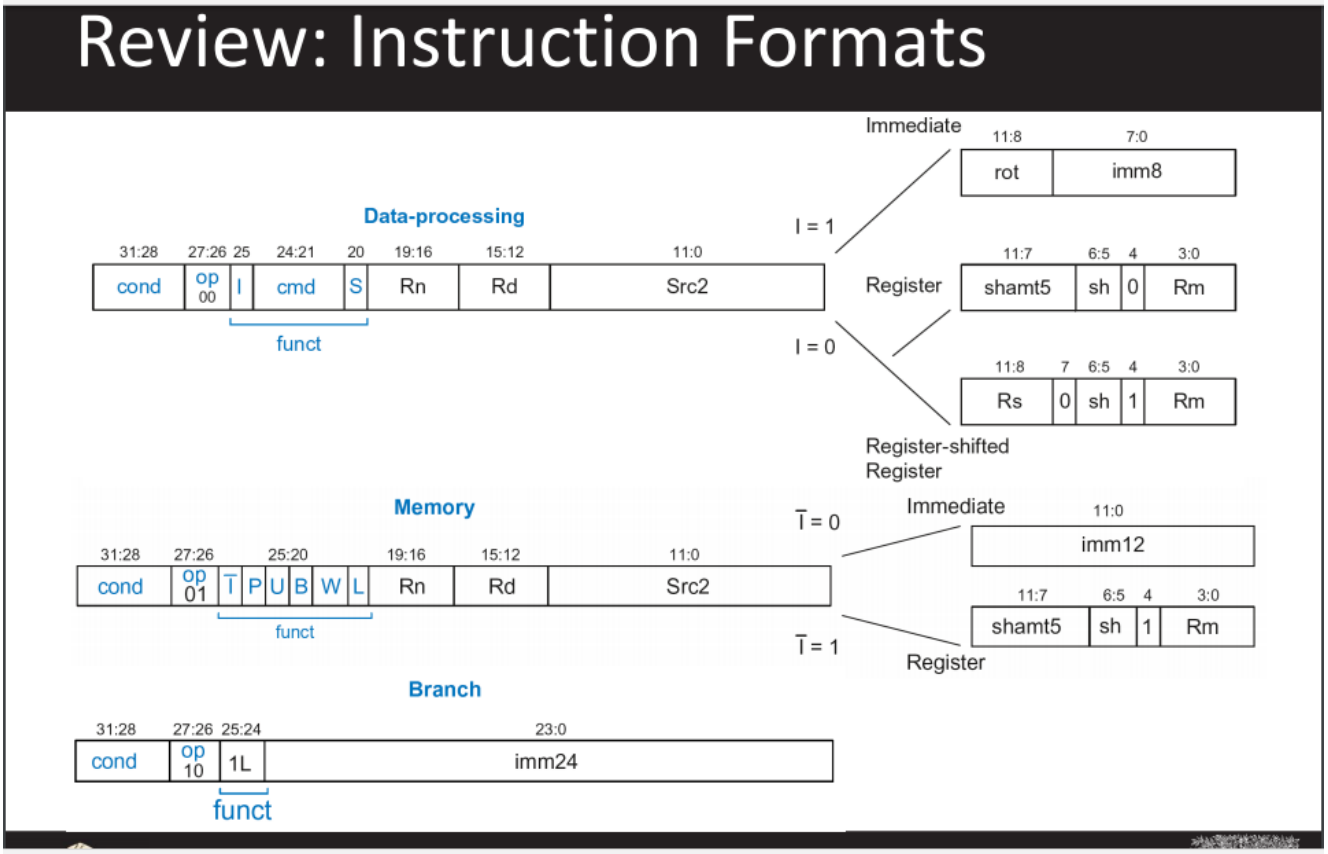
```
ENTITY tb IS
END tb;

ARCHITECTURE behavior OF tb IS

    COMPONENT topfile
    PORT(
         clk : IN  std_logic;
         reset : IN  std_logic;
         result : OUT  std_logic_vector(31 downto 0);
         flags : OUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal reset : std_logic := '0';
   signal result : std_logic_vector(31 downto 0);
   signal flags : std_logic_vector(3 downto 0);
   constant clk_period : time := 10 ns;
BEGIN
 -- Instantiate the Unit Under Test (UUT)
   uut: topfile PORT MAP (
        clk => clk,
          reset => reset,
          result => result,
          flags => flags
        );
   clk_process :process
   begin
                    clk <= '0';
                    wait for clk_period/2;
                    clk <= '1';
                    wait for clk_period/2;
   end process;
            stim_proc: process
  begin
 hold reset state for 100 ns.
     wait for clk_period*10;
 insert stimulus here
wait;
   end process;
```

END;

Rezultat simulacije:



OBJAŠNJENJE INSTRUKCIJA



Tri vrste instrukcija:

-Data-processing

-Memory

-Branch

Data processing:

Cond-Nebitna 4 bitava

OP-2 bita potrebna za odlučivanje vrste instrukcije(Data-Processing(00),Memory(01),Branch(10))

I-Immediate je kontrolni bit koji određuje da li se drugi podatak uzima iz memorije ili ekstendera.

cmd-Određuje vrstu funkcije (ADD,SUB,AND,OR) (“0000”,”0001”,”0010”,”0011”)

s-Nebitno u Data-processing

Rn-adresa prvog operanda u registar fajlu

Rd-adresa destinacije u registar fajlu

Src2 ima dva moda:

i=1 onda rot-nebitno,imm8 nalazi se vrednost podatka koji se proširi ekstenderom

i=0 onda su svi bitovi od 11 do 4 nebitni ,zadnja 4 bita su adresa drugog podatka (Rm) u registar fajlu

MEMORY:
Cond-nebitno
OP(01)
I,P,U,B,W nebitno
L=1 onda je LOAD
L=0 onda je STORE

STORE:Rn- je adresa Registar File-a sa koje se podatak sabira sa vrednosti Imm12 i to Predstavlja adresu u DataMemory u koju će se učitati podatak sa adrese Rd iz RegistarFile-a.

LOAD:Isto kao STORE samo se podatak čita iz DataMemory i upisuje u RegisterFile

BRANCH:
Cond-nebitno
OP-10
Funct-10
Imm24 se proširi ekstenderom i sabere se sa vrednošću sa adrese 15 iz RegisterFile-a
I to se upiše u PC registar.

Da bi mogle biti ispunjene predhodne instrukcije potrebno je da kontrolna jedinica poštuje parametre date u sledećoj tabeli.

Op	$Funct_5$	$Funct_0$	Type	Branch	MemtoReg	MemW	ALUSrc	ImmSrc	RegW	RegSrc	ALUOp
00	0	X	DP Reg	0	0	0	0	XX	1	00	1
00	1	X	DP Imm	0	0	0	1	00	1	X0	1
01	X	0	STR	0	X	1	1	01	0	10	0
01	X	1	LDR	0	1	0	1	01	1	X0	0
11	X	X	B	1	0	0	1	10	0	X1	0

ZAKLJUČAK

Single Cycle izvršava instrukcije u jednom taktu.To mu omogućava relativno lako razumevanje rada i to mu jedna od prednosti.Maksimalna frekfencija našeg ARM-a
je 249 MHZ ,a ipak može izvršavati instrukcije nego MULTI_CYCLE na višim frekfencijama jer celu instrukciju izvršava u jednom taktu.
SingleCycle je mama :)

LITERATURA

Digital Design and Computer Architecture ARM Edition 2015
Wikipedia
Google.com
Xilinx.com