**Univerzitet u Novom Sadu**

**FAKULTET TEHNIČKIH NAUKU**

**Departman za Elektroniku**

**Mikroprocesorska Elektronika**

PROJEKTOVANJE

ARM

PROCESORA

(VHDL)

Mentor: Studenti:

**Ivan Mezei** **Nemanja Sibinčić**

**Aleksandar Komazec**

**Milan Božić**

Sadržaj

Uvod 2

Opšti prikaz ARM-a i podela procesora 3

Detaljan prikaz blokova ARM-a 4

Multiplekser 4

Sabirač 7

Registri opšte namene 9

Memorijski blokovi tipa RAM 13

Registar fajl 16

Data Memory 20

Extender 24

Kontrolna jedinica 27

Organizacija DataPath-a 29

Kompletan strukturni model ARM \_SC procesora 32

Objašnjenje instrukcija 36

Zaključak 38

Literatura 38

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

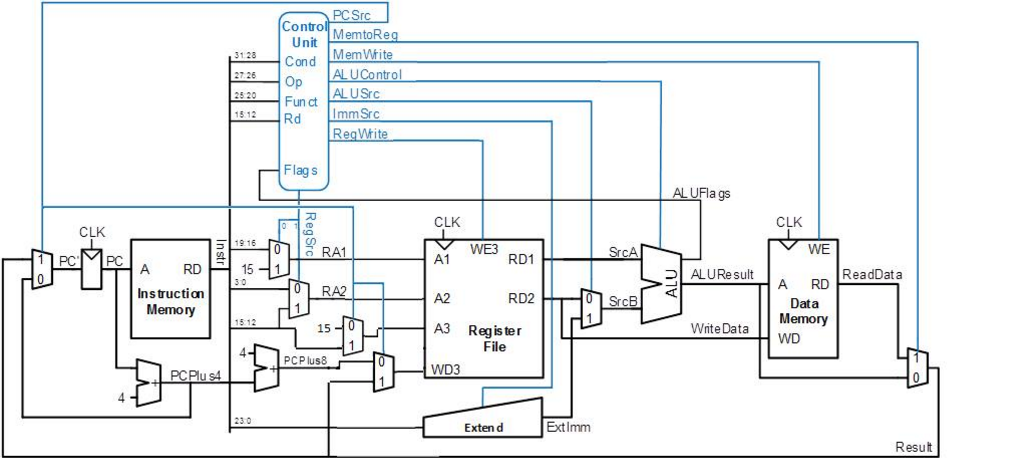
-Memorijski blokovi tipa RAM ->2

-Memorijski blokovi tipa Rom ->1

-Blok za proširenje vrednosti ->1

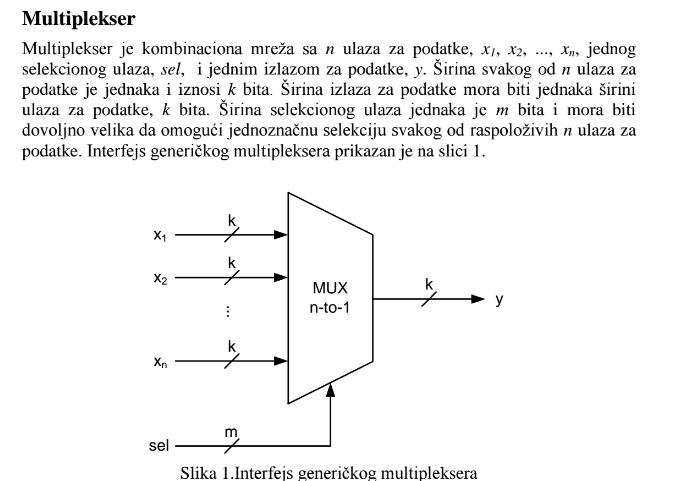
2.Control Unit koga čine sledeći blokovi

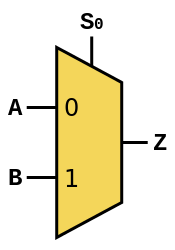
-Kontrolna jedinica je urađena preko memorijskog bloka (ROM)



Slika 1: Blok šema ARM\_SC procesora

DETALJAN PRIKAZ BLOKOVA ARM-a





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;

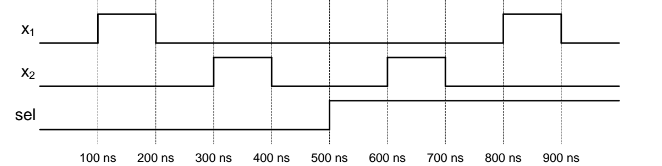
x2\_s <= ‘0’, ‘1’ after 300 ns, ‘0’ after 400 ns, ‘1’ after 600 ns, ‘0’ after 700 ns;

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

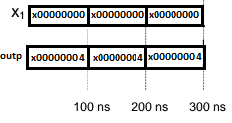
begin

x1\_s <= ‘0’;

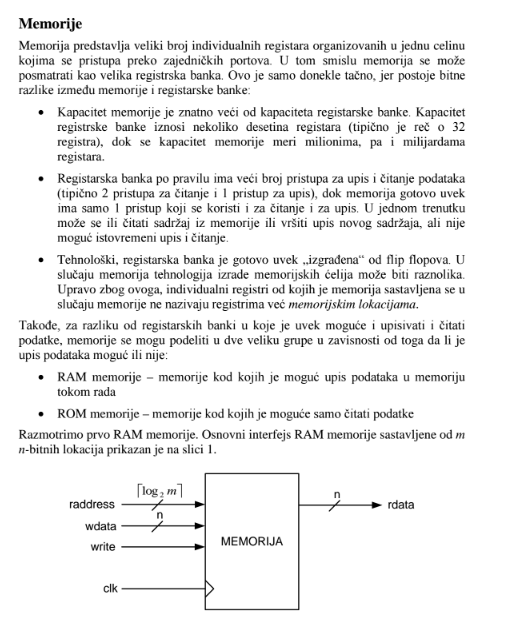
wait;

end process;

end architecture beh;



**Registri opšte namene**





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

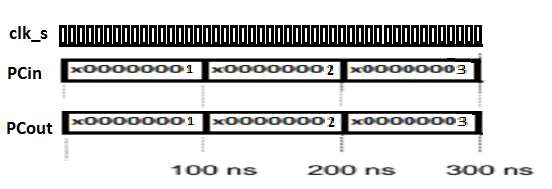
begin

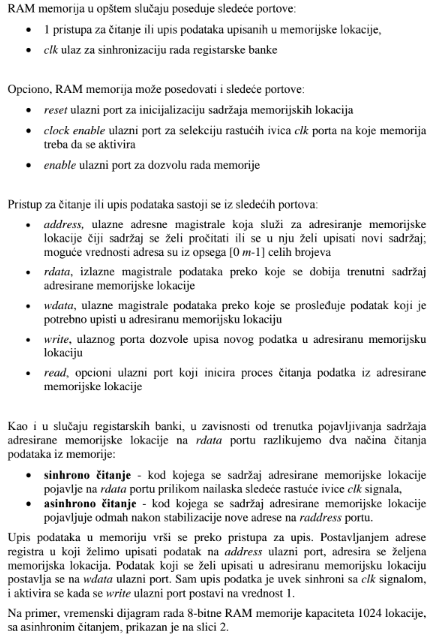
PCin\_s<=x”00000001”, x”00000002” after 100 ns, x”00000003 ”after 200 ns ;

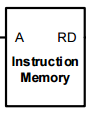
wait;

end process;

END;



**Memorijski blokovi**



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

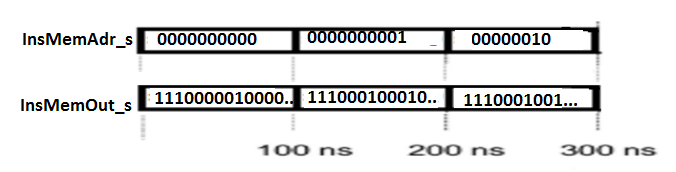
begin

InsMemAdr\_s<= ”00000001”after 100 ns, ”00000010” after 200 ns;

wait;

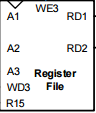
end process;

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;

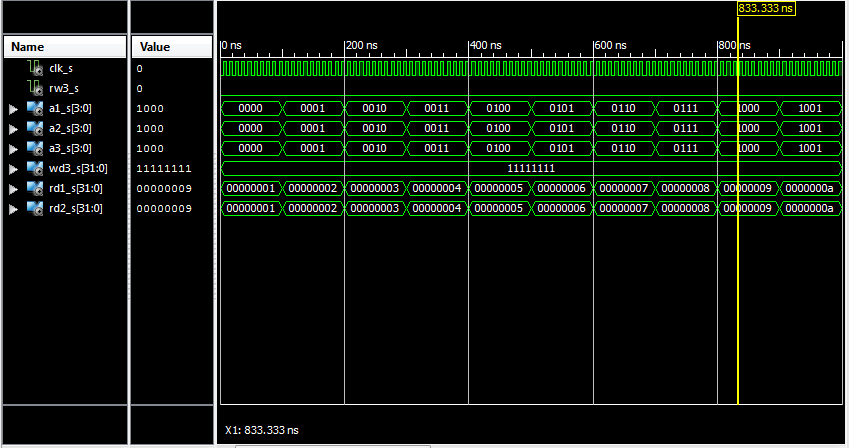
A3\_s<=X"1" after 100 ns,X"2" after 200 ns,X"3" after 300 ns,X"4" after 400 ns,X"5" after 500 ns,X"6" after 600 ns,X"7" after 700 ns,X"8" after 800 ns,X"9" after 900 ns,X"A" after 1000 ns,X"B" after 1100 ns,X"C" after 1200 ns,X"D" after 1300 ns,X"E" after 1400 ns,X"F" after 1500 ns;

--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

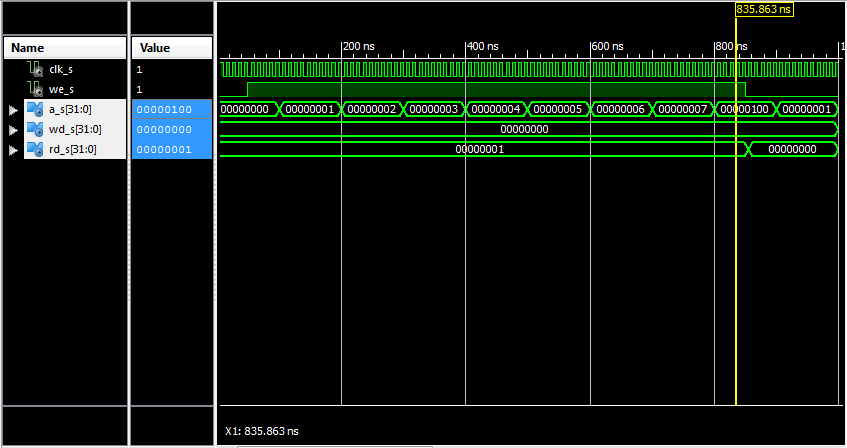
begin

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;

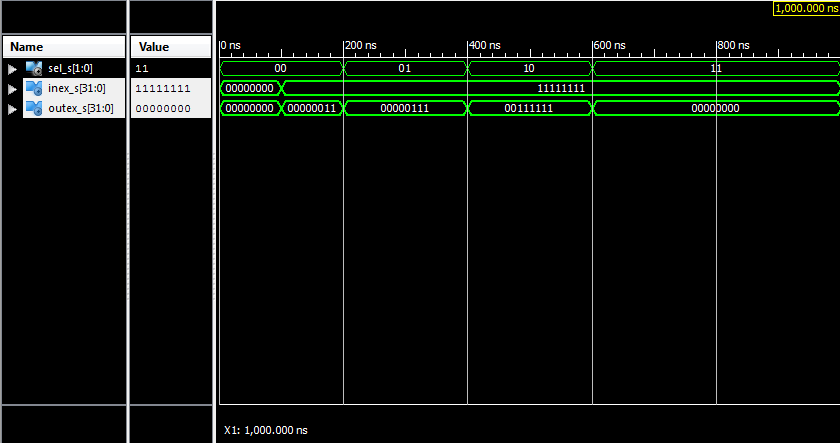
sel\_s<="01" after 200 ns,"10" after 400 ns,"11" after 600 ns;

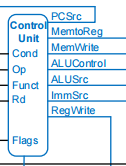
wait;

end process;

END;

Rezultat simulacije:



**Kontrolna jedinica (Control Unit)**

**Upravljačka jedinica**, **Upravljački sklop** ili **kontrolna jedinica** je deo [središnje jedinice](https://hr.wikipedia.org/wiki/CPU) (na [engleskom](https://hr.wikipedia.org/wiki/Engleski_jezik) central processing unit [CPU](https://hr.wikipedia.org/wiki/CPU)) koja usmerava radnju i ima izravnu kontrolu preko ostalih delova CPU-a: memorije, aritmetičko-logičke jedinice, instrukcijskih memorija, [sabirnica](https://hr.wikipedia.org/wiki/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](https://hr.wikipedia.org/wiki/Mikroprocesor) 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](https://hr.wikipedia.org/wiki/ALU), [registara](https://hr.wikipedia.org/wiki/Me%C4%91uspremnik),PC, memorija [stanja](https://hr.wikipedia.org/w/index.php?title=Spremnik_stanja&action=edit&redlink=1), [IR](https://hr.wikipedia.org/w/index.php?title=Adresni_me%C4%91uspremnik&action=edit&redlink=1) 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](https://hr.wikipedia.org/wiki/Mikroprogram) upravljanjem ostalih delova CPU-a obavlja se izvršavanjem skupa manjih radnji koje su kodirane u mikroprogram. Mikroprogram je pohranjen u [kontrolnoj memoriji](https://hr.wikipedia.org/w/index.php?title=Kontrolna_memorija&action=edit&redlink=1) 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;

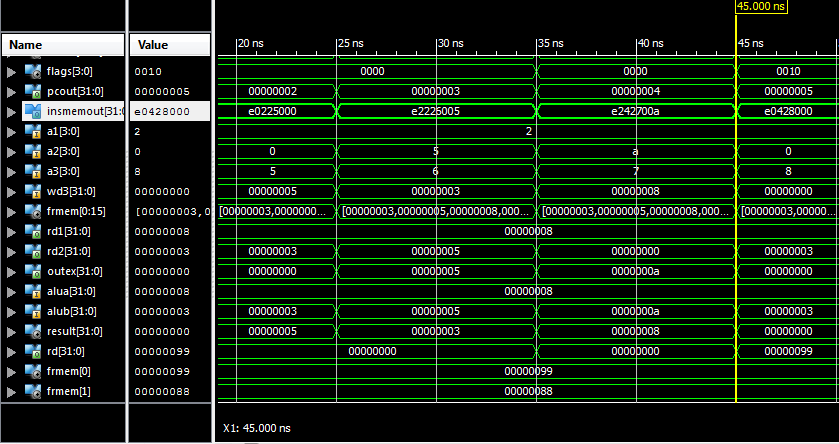
insert stimulus here

wait;

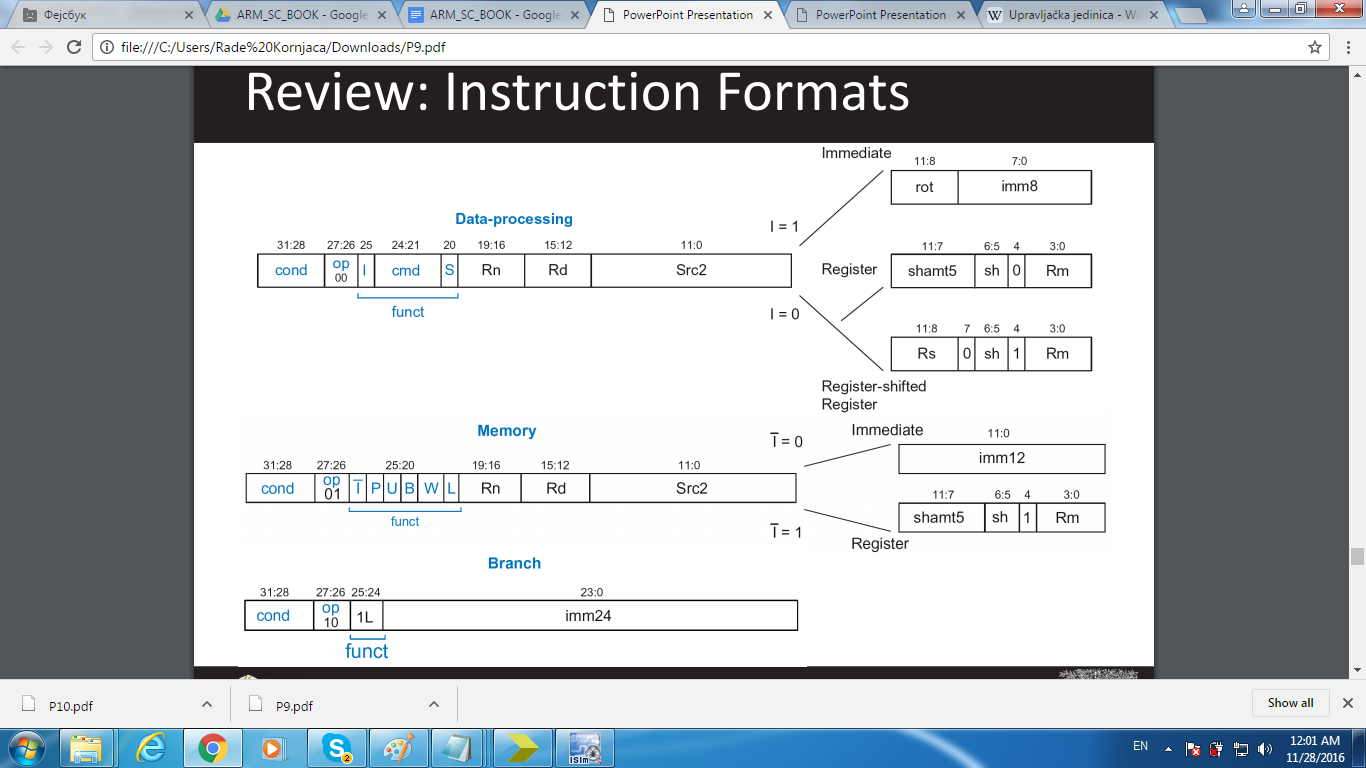
end process;

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

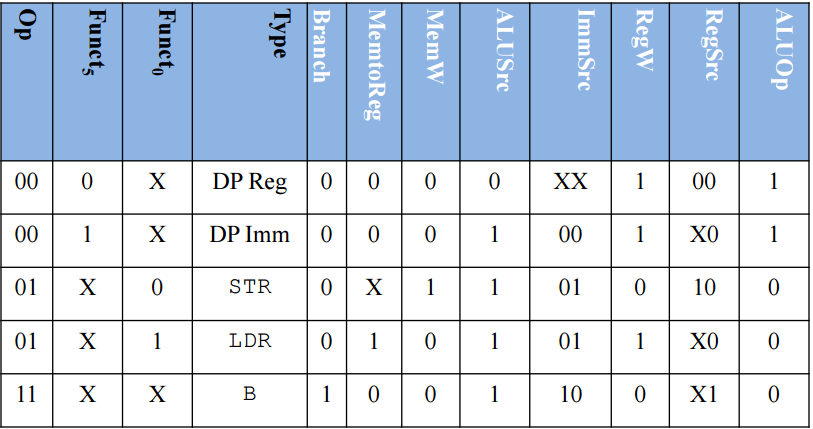
OP-10

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.



**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