Universitatea Tehnica din Cluj-Napoca

Catedra de Calculatoare

Asamblor pentru un procesor RISC

Profesor coordonator: Daniela Fati

Numele studentului: Simina Monica Balint

Grupa: 30236

Data: 03/01/2020

Cuprins:

1. Rezumat

Acest proiect are ca si scop realizarea unui asamblor pentru un procesor **RISC**(Reduced Instruction Set Computer – Procesor cu Set Instructiuni Reduse). Principalele obeiective ale acestei lucrari sunt urmatoarele: implementarea propriu zisa, intr-un mediu de dezvoltare hardware a procesorului cerut, avand un set prestabilit de instructiuni si realizarea unei aplicatii Windows cu meniuri, ceea ce tine loc de interfata cu utilizatorul. Pentru a putea discuta despre atingerea acestor obiective principale, se observa inca de la inceput ca cele doua vor fi separate in 2 parti, fiecare obiectiv apartinand uneia: partea hardware, pentru primul obiectiv si respectiv partea software pentru al doilea. Mediul de dezvoltare hardware ales este Vivado versinea 2016.1 , iar limbajul utilizat este VHDL. Pentru partea software am optat pentru mediul de dezvoltare Eclipse, impreuna cu limbajul Java. Interfata cu utilizatorul este una simpla, avand mai multe butoane, care intruneste toate componentele hardware ale procesorului. Pentru a putea testa functionalitatea programului am utilizat mediul de simulare oferit de Vivado, iar prin intermediul acestei simulari s-au putut urmarii rezultatele si pe baza acestora s-au tras concluzile finale. Procesorul **RISC** are ca principala structura interna o memorie ROM(Read Only Memory), in care au fost introduse instructiunile procesorului hardcodate. Pe baza simularii se poate observa foarte usor ca se trece de la o intructiune la alta la fiecare semnal de ceas(functionalitate corecta).

2. Introducere

In informatica, un **procesor** este un dispozitiv hardware al unui computer care pornind de la un set de instructiuni efectueaza operatiuni pe o sursa externa de date. Termenul este frecvent utilizat pentru a face referire la unitatea centrala de procesare dintr-un sistem.

Procesorul este elementul principal al unui sistem de calcul si incorporeaza functiile unitatii centrale de prelucrare a informatiei a unui calculator sau a unui sistem electronic structurat functional (care coordoneaza sistemul).

De obicei, fizic procesorul se prezinta sub forma unui microprocesor, care este fabricat pe un singur cip de circuit integrat metal-oxid-semiconductor (MOS). Reprezinta forma structurala cea mai complexa pe care o pot avea circuitele integrate. Cipul semiconductor, care este plasat pe placa de baza, este foarte complex, putand ajunge sa contina milioane de microtranzistoare- dispozitiv electronic din categoria semiconductoarelor care are cel putin trei terminale (borne sau electrozi), care fac legatura la regiuni diferite ale cristalului semiconductor. El controleaza activitatile intregului sistem in care este integrat și poate prelucra datele furnizate de utilizator[1].

Microprocesoarele sunt utilizate in orice arhitectura de calcul: PC -uri, servere, laptop -uri, sisteme inglobate etc, unde compatibilitatea software, performanta, generalitatea și flexibilitatea sunt importante. In prezent, aproape toate procesoarele de uz general sunt microprocesoare, ceea ce face ca cei doi termeni sa fie practic sinonimi.

Arhitectura **RISC –** calculator cu set de instructiuni reduse, apartine unui miscroprocesor si calculator cu un set de comenzi simple si rapide, in care viteza creste datorita simplificarii instructiunilor, astfel incat decodificarea acestora este mai simpla , pentru a oferii performante superioare. Arhitectura **RISC** mai este cunoscuta si sub numele de "arhitectura Load/Store", deoarece instrucțiunile care

se refera la memoria principala sunt limitate la o singura operatie - fie incarcare, fie stocare. Conceptul **RISC** a dus la un design mai atent al microprocesoarelor, pe 32-bit sau pe 64-bit spatiu de adresare. **RISC** este o evolutie si alternativa la Complesc Instructins Set Computer (CISC) si pana in prezent, este cea mai eficienta tehnologie a arhitecturii CPU. Incepand cu anul 2018, 99% din toate cipurile noi utilizeaza arhitectura **RISC**.

Obiectivele principale ale arhitecturilor **RISC** pot fi considerate universale si sunt acceptate de cate toti proiectantii de calculatoare: minimizarea timpului de executie si minimizarea costului de proiectare. Metode adoptate pentru indeplinirea obiectivelor: imbunatatirea tehnologiei componentelor, minimizarea numarului mediu al ciclurilor de ceas pe instructiune si executia simultana a mai multor instructiuni[3].

Caracteristicile principale ale procesorului **RISC:**

- unitatea de control este realizata prin "logica de tip cablat";
- Instructiunile se desfasoara intr-o singura stare si durate de timp identice;
- numarul de instructiuni este mic, sub 128;
- dimensiunea este fixa pentru tot formatul instructiunilor;
- instructiunile au format uniform, nu mai mult de 4 tipuri de formate;
- numarul modurilor de adresare este mic (sub 4) și nu neaparat simple;
- accesul memoriei se face numai cu instructiuni de transferuri simple de date intre registre si memorie (tip "LOAD" /STORE");
- include numai instructiuni simple, executate rapid, de obicei intr-un singur tact de ceas, datorita tehnicii pipeline;
- tendinta de accentuare pe parte software;
- mai putine tipuri de date hardware;
- prezinta un format uniform al instructiunilor, ceea ce conduce la o codificare mai simpla;
- minimalizeaza utilizarea memoriei prin existenta unui numar mare de registri (datele sunt aduse in si din memorie prin intermediul acestor registri) si permiterea utilizarii multiple a registrilor generali, care sunt identici.

In sectiunile ce urmeaza voi prezenta: fundamentarea teoretica, metodele alese pentru proiectare si implementare cu 3 subcapitole, rezultatele experimentale, concluzile trase si bibliografia.

3. Fundamentare teoretica

Scopul acestui proiect este de a proiecta si a implementa un asamblor pentru un procesor RISC.

Modelul principal dupa care m- am orientat in realizarea proiectului este microprocesorul pipeline pe care l-am rezolvat in urma cu un an in cadrul materiei Arhitectura Calculatoarelor. Procesorul este stucturat pe mai multe componente si module, fiecare modul la randul lui fiind iimplementat printr-o arhitectura comportamentala. Totodata cu ajutorul materialelor de la laboratorul materiei Structura Sistemelor de Calcul, am reusit implementarea din punct de vedere hardware a procesorului.

Cand vine vorba despre interfata grafica, am apelat la cunostiintele acumulate de-a lungul anilor si in special anul trecut de studiu, pentru a modela o aplicatie windows in limbajul de programare Java.

4. Proiectare si implementare

4.1. Metode experimentale utilizate

Am folosit in realizarea proiectului 2 metode experimentale in functie de scopurile dorite.

Pentru implementarea hardware am ales utilizarea mediului de programare Vivado. **Vivado Design Suite** este o suita software produsa de Xilinx pentru sinteza si analiza proiectelor HDL , inlocuind Xilinx ISE cu caracteristici suplimentare pentru dezvoltarea unui sistem pe cip si sinteza la nivel inalt. Vivado reprezinta o rescriere fundamentala si regandirea intregului flux de proiectare (comparativ cu ISE). Iar pentru testarea proiectului am folosit modulul de simulare oferit de Vivado.

Pentru implementarea sofware am ales utilizarea mediului de programare Eclipse. **Eclipse** este un mediu de dezvoltare open-source scris preponderent in Java. Acesta poate fi folosit pentru a dezvolta aplicatii Java si, prin intermediul unor plug-in-uri, in alte limbaje, cum ar fi C, C++, COBOL, Python, Perl și PHP [1].

4.2. Solutia aleasa

Acesta solutie prezentata este cea mai usora de abordat pe baza cunostiintelor acumulate pana acum. Aceasta aplicatie reprezinta o versiune minimalista a unui asamblor pentru un procesor RISC: reuneste componentele principale ale procesorului intr-o interfata grafica prietenoasa pentru un utilizator.

La un nivel mai inalt putem vorbi despre conectarea acesteia la un compilator care sa verifice corectitudinea codului introdus si conversia acestuia in limbajul calculatorului.

4.3. Module utilizate si scheme bloc

In realizarea procesorului se opteaza pentru o implementare pipeline prin intermediul careia se pot distinge 4 etaje:

- **IF**- contine doar elementele unitatii de control; aici se extrage instructiunea din memoria de instructiuni si se actualizeaza controlul de program PC;
- **ID**- realizeaza decodificarea instructiunii din registrul dde instructiuni si generarea semnalelor de comanda necesare executiei instructiunii; semnalele DateSA si DateSB din acest etaj sunt utilizate ca registre pipeline;
- **EX**- pentru majoritatea inscuctiunilor, in acest etaj se executa o operatie aritmetrica, o operatie logica sau o operatie cu memoria; semnalele de comanda utilizate aici sunt MxB, OpUAL si MemWR, restul semnalelor sunt inscrise in registrul pipeline;
- **WB**- ultimul etaj; selecteaza cu ajutorul multiplexorului MUXD, valoarea care va fi inscrisa in registrul destinatie din setul de registre[2].

Componente importante:

1. Uitatea aritmetrica si logica prezentata in figura 1, contine modulele FUNC si DEPL, unde primul modul are rolul de a implementa toate operatiile artimetrice si logice- mai putin operatiile de deplasare- iar cel de-al doilea modul implementeaza operatiile de deplasare stanga- dreapta. Multiplexorul MUXF selecteaza care dintre cele 2 iesiri trebuie trimisa mai departe.

UAL

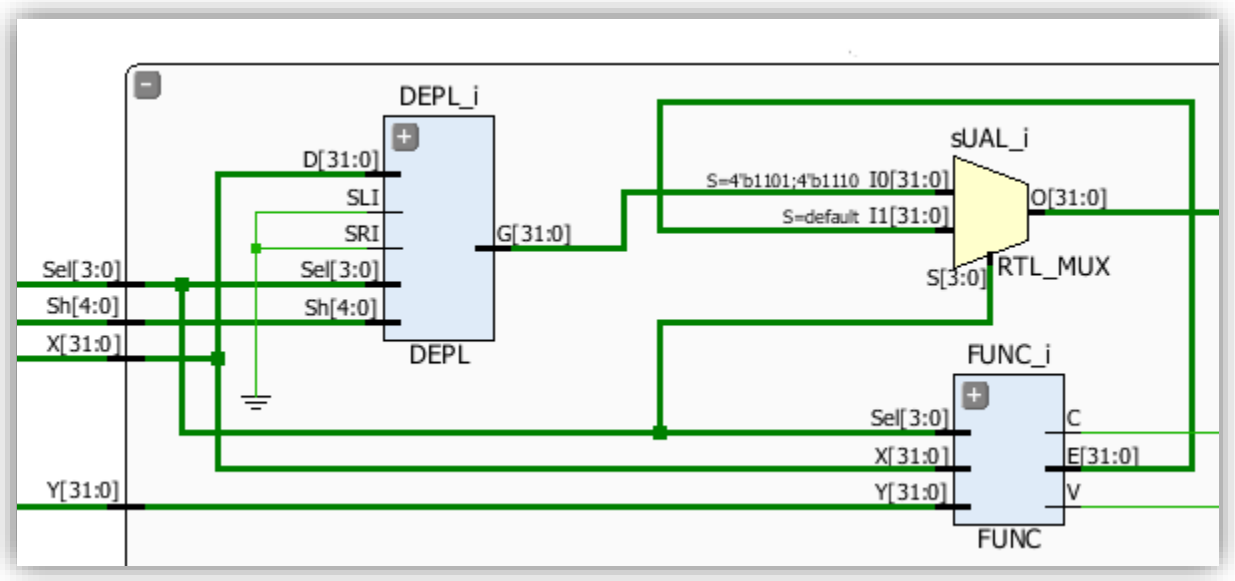

Figura 1: schema bloc a UAL genrata de Vivado

2. Unitatea de control prezentata , este reprezentata de cel mai importat circuit al sau si anume, decodificatorul de instructiuni, prezentat in figura 2. Acesta este un circuit combinational care genereaza toate semnalele de comanda necesare functionarii procesorului pe baza diferitelor campuri ale instructiunii aflate in registrul de instructiuni RI. Pentru incarcarea contorului de program PC cu adresa urmatoarei instructiuni executate, se utilizeaza multiplexorul MUXC.

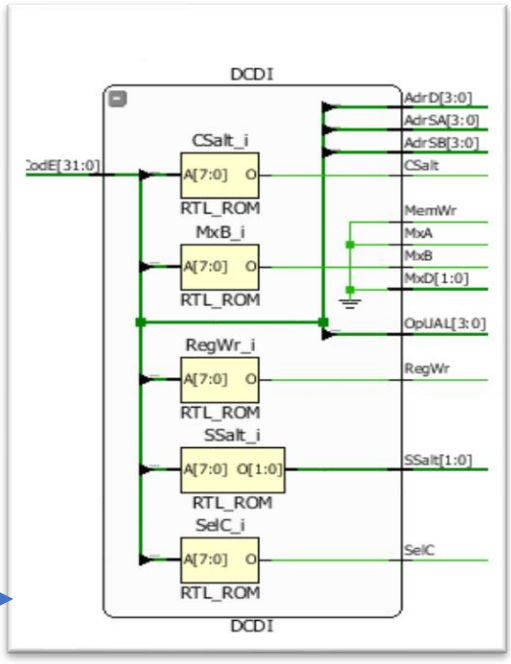

Figura 2: schema decodificator generata de Vivado

In figura 3 este reprezentata schema bloc restransa a procesorului RISC

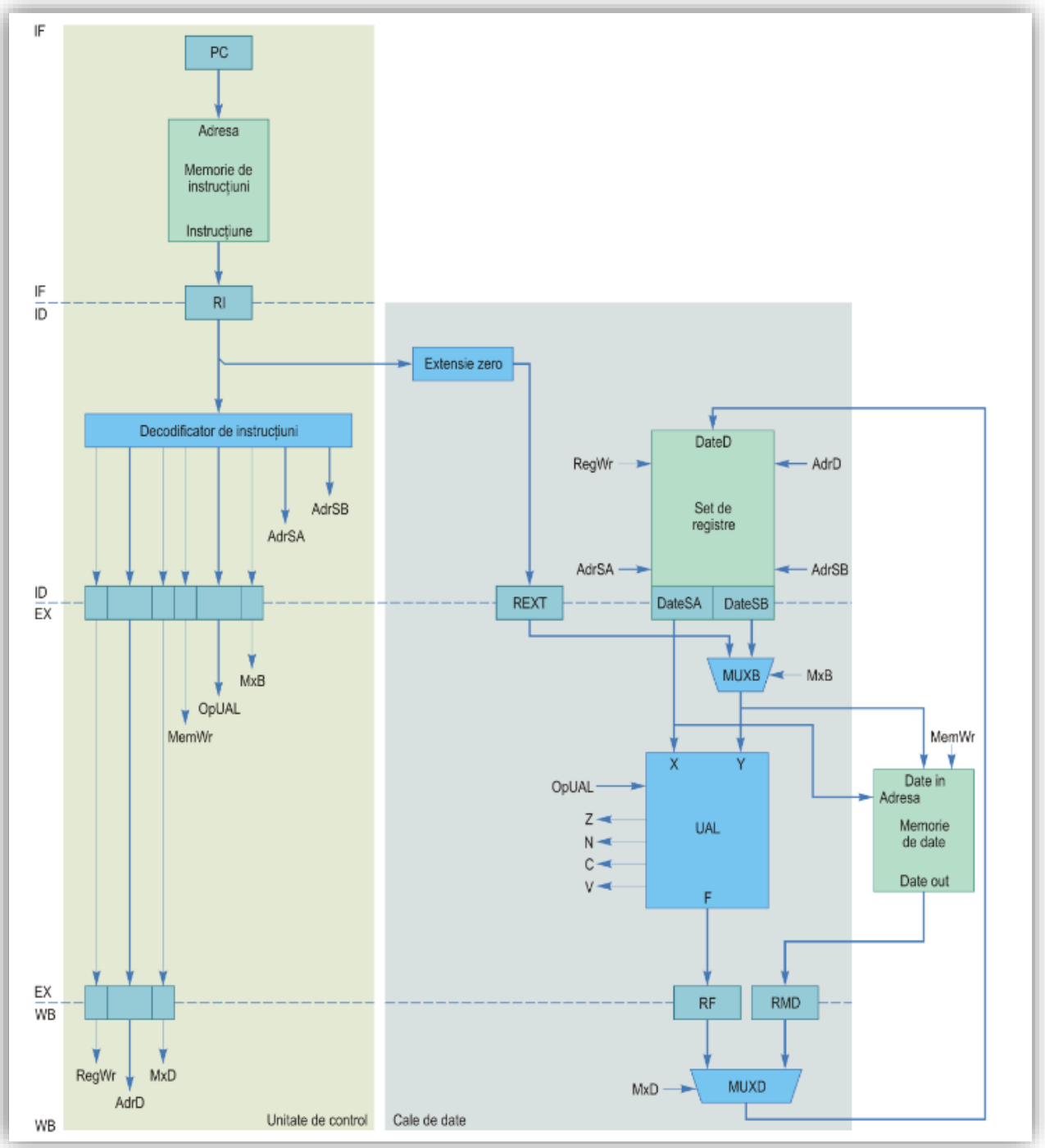

Figura 3: schema simplificata a unitatii de control si a caii de date a procesorului [2]

Din punct de vedere al interfetei , codul este unul simplu si usor de inteles, aceasta fiind foarmata din 12 butoane cu nume speicfice(figura 4) care fiecare la randul lui, lanseaza cate un cod executabil al unei componente ce apartine procesorului. Fereastra lansata se deschide in mod de editare si atunci codul poate fi modificat(Figura 5).

Figura 4: aplicatie cu 12 butoane

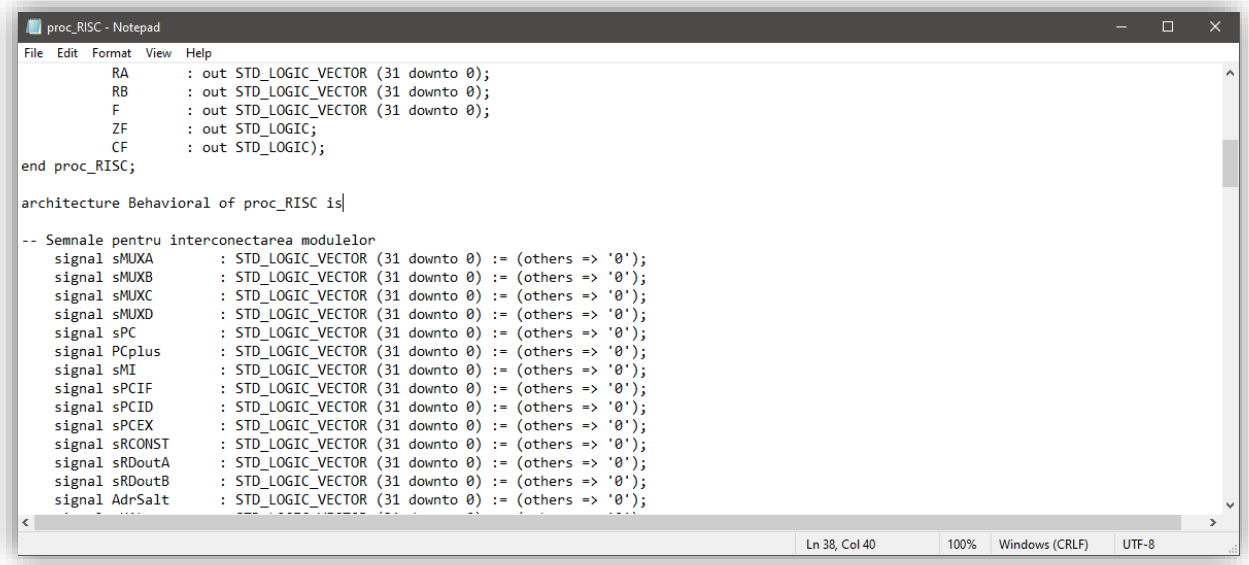

Figura 5: fisierul editabil si codul executabil al unei componente

Un buton a fost realizat utilizant urmatoarea secventa de cod din figura de mai jos:

```
JButton btnNewButton_6 = new JButton("mux_4_1");
btnNewButton_6.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        try {
            Desktop.getDesktop().open(new java.io.File("File/mux_4_1.txt"));
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
});
btnNewButton_6.setBounds(391, 320, 189, 48);
frame.getContentPane().add(btnNewButton_6);
```

Figura 6: codul unui buton

In aceasta secventa de cod se verifica daca exista o cale valida la fisierul cautat, iar in caz contrar se arunca o exceptie.

5. Rezultate experimentale

Cel mai mare rol din acest capitol il are simulatorul Vivado, cu ajutorul caruia am reusit sa testez cu succes implementarea procesorului.

Pentru a putea testa acest proiect a fost necesara implementarea unui banc de test care are srtuctura urmatoare:

- partea declarativa prezentata in figura 7.

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity simulare is
--  Port ( );
end simulare;
```

Figura 7: bibliotecile folosite si antetul simularii procesorului

- declararea semnalelor necesare,generarea unui semnal de clock si instantierea ca si componenta a procesorului in arhitectura simularii propriu zise(figura 8).

```
rchitecture Behavioral of simulare is

   constant CLK_PERIOD : TIME := 20 ns;
   signal Clk      : std_logic;
   signal Rst      : STD_LOGIC;
   signal AdrInstr : STD_LOGIC_VECTOR (31 downto 0);
   signal Instr    : STD_LOGIC_VECTOR (31 downto 0);
   signal Data     : STD_LOGIC_VECTOR (31 downto 0);
   signal RA       : STD_LOGIC_VECTOR (31 downto 0);
   signal RB       : STD_LOGIC_VECTOR (31 downto 0);
   signal F        : STD_LOGIC_VECTOR (31 downto 0);
   signal ZF       : STD_LOGIC;
   signal CF       : STD_LOGIC;

egin

   gen_clk: process
       begin
            Clk <= '0';
            wait for (CLK_PERIOD/2);
            Clk <= '1';
            wait for (CLK_PERIOD/2);
   end process gen_clk;

UT: entity work.proc_RISC
       generic map (DIM_MI => 256,
                    DIM_MD => 256)
       port map( Clk => Clk,
                 Rst => Rst,
                 AdrInstr => AdrInstr,
                 Instr => Instr,
                 Data => Data,
                 RA => RA,
                 RB => RB,
                 F  => F,
                 ZF => ZF,
                 CF => CF);
```

Figura 8

- generarea unui proces pentru testarea propiu-zisa- figura 9.

```
gen_test: process
begin
    Rst <= '1';
    wait for 20ns;
    Rst <= '0';
    wait for 20ns;
    wait;
end process;
```

Figura 9.

In urma rularii acestui banc de test am obtinut urmatoarele rezultate din figura 10:

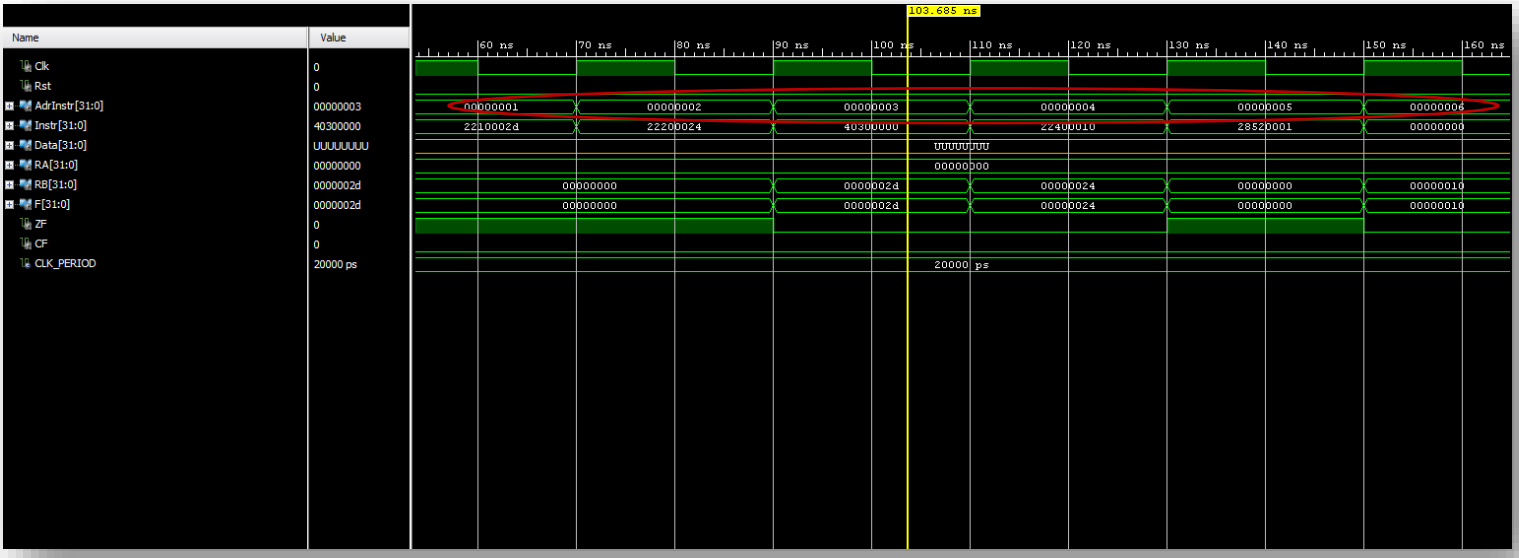

Figura 10: rezultatul simularii

Se poate observa cu usurinta ca rezultatele obtinute, cele incercuite cu rosu, sunt cele bune pe baza codului din memoria ROM, prezentat in figura 11. Campul data are valoare a nedefinita deoarece in programul din memorie nu se folosesc instructiuni care utilizeaza memoria.

```
type MI_TYPE is array (0 to DIM_MI-1) of STD_LOGIC_VECTOR (31 downto 0);
signal MI : MI_TYPE := (
                        x"0A00_0000",       -- 00 XOR  R0,R0,R0
                        x"2210_002D",       -- 01 ADDI R1,R0,45 (0x2D)
                        x"2220_0024",       -- 02 ADDI R2,R0,36 (0x24)
                        x"4030_0000",       -- 03 MOVA R3,R0
                        x"2240_0010",       -- 04 ADDI R4,R0,16 (0x10)
                        x"2852_0001",       -- 05 ANDI R5,R2,1
                        x"0000_0000",       -- 06 NOP
                        x"0000_0000",       -- 07 NOP

                        x"6005_0003",       -- 08 BZ   R5,3
                        x"0000_0000",       -- 09 NOP
                        x"0000_0000",       -- 0A NOP
                        x"0233_1000",       -- 0B ADD  R3,R3,R1
                        x"0E11_0001",       -- 0C SHL  R1,R1,1
                        x"0D22_0001",       -- 0D SHR  R2,R2,1
                        x"2544_0001",       -- 0E SUBI R4,R4,1
                        x"0000_0000",       -- 0F NOP

                        x"0000_0000",       -- 10 NOP
                        x"5004_FFF3",       -- 11 BNZ  R4,-13 (0xFFF3)
                        x"4003_0000",       -- 12 MOVA R0,R3
                        x"6900_0000",       -- 13 HALT
                        x"6900_0000",       -- 14 HALT
                        x"6900_0000",       -- 15 HALT
                        x"6900_0000",       -- 16 HALT
                        x"6900_0000",       -- 17 HALT
```

Figura 11: o parte din continutul memoriei ROM

6. Concluzii

Studierea si realizarea acestui proiect m-a ajutat la aprofundarea cunostiintelor acumulate pana acum in domeniul de programare hardware. Pot spune ca mi-am dezolvatat abilitatile de modelare in limbajele de programare: VHDL si Java.

In final, se poate spune ca aplicatia realizata, separata in cele 2 parti,este una usor de intles.

Din cauza situatiei in care ne aflam proiectul nu a putut fi testat pe o placa fizica, asa ca testarea acestuia s-a bazat in totalitate pe rezultatele simulatorului.

Unele dintre cele mai bune dezvoltari pe viitor, ar fi conectarea interfetei grafice la un compilator pentru a putea compila si traduce codul executabil, in cod masina.

7. Bibliografie

[1] www.wikipedia.org

[2] laborator numar 12 – dr. Zoltan Baruch

[3] https://rum.kagutech.com/4129808-risc-processor-architecture