Universitatea Tehnica

din Cluj-Napoca

Catedra de Calculatoare

Microcontroler simplu

Numele indrumatorului:

Daniela Fati

Data: 11 noiembrie 2020

Numele studentului:

Dascal Raluca-Georgiana

Grupa: 30326

Cuprins:

1. Rezumat

Un microcontroler este în esență o versiune mai mică si mai ieftina a computerelor personale. Acest dispozitiv reprezinta un computer autonom, cu un singur chip, care incorporează toate componentele de bază ale unui computer personal la o scară mult mai mică. Microcontrolerele sunt de obicei utilizate ca si controlere incorporate care controlează unele părți ale unui sistem mai mare, cum ar fi roboții mobili, perifericele computerului etc.

Acest proiect a avut ca scop proiectarea structurala a unui microcontroler pe o placa de devoltare FPGA utilizand limbajul VHDL. Acesta trebuia sa defineasca atat instructiunile generale ale unui asemenea dispozitiv cat si cateva instructiuni speciale pentru controlul resurselor placii specificate mai sus. Programul executat de catre unitatea centrala a microcontrolerului a continut un set de instructiuni (in cod masina) care au facut posibila calcularea sumei numerelor pare dintr-un sir si determinarea faptului ca aceasta a fost sau nu multiplu de un numar dat.

Metoda de rezolvare aleasa a fost descrierea in limbaj hardware VHDL a cinci module elementare care au stocat setul de instructiuni din programul (firmware) al microcontrolerului, au analizat codul masina al acestora (fiecare instructiune a fost descrisa in memorie ROM sub forma precizata) si au extras informatiile necesare efectuarii tuturor operatiile logice si aritmetice necesare.

Rezultatele obtinute au fost corecte din punct de vedere matematic (au fost comparate rezultatele obtinute de catre simulatorul aplicatiei Vivado Xilinx cu rezultatele obtinute in urma parcugerii ‘pe foaie’ a programului din memoria ROM). Datele necesare verificarii corectitudinii au fost intocmai cele prezentate mai sus (suma numerele pare dintr-un sir si valoarea 0 stocata intr-un registru in cazul in care suma obtinuta nu era multiplu de numar dat sau 1 in caz contrar).

Principalele concluzii deduse din studierea si realizarea acestui proiect au fost constientizarea importantei si a vastei utilizari a microcontroalelor in viata de zi cu zi si aprofundarea cunostiintele dobandite pana acum atat in ceea ce priveste limbajul hardware VHDL cat si logica de proiectare structurala.

2. Introducere

Microcontrolerul este o structura electronica care permite controlul unui proces sau, in termeni mai generali, a unei interactiuni specifice cu mediul exterior, fara a fi nevoie de interventia activitatii umane. Scopul microcontroalelor difera in functie de modul de functionare. Acesta poate fi modificat prin programarea cu un soft special creat intocmai pentru a face posibila indeplinirea sarcinile cerute. Acest program instalat si rulat intr-un microcontroler se numeste firmware.

Structura de baza a unui microcontroler cuprinde : o unitate centrala (CPU) - denumit si creierul microcontroalelor - dispozitivul care este utilizat pentru preluarea datelor din program, decodarea si finalizarea sarcinilor atribuite realizand operatiile aritmetice si logice necesare ; o memorie- chip-ul de memorie stocheaza toate programele - (seturile de instructiuni) pe care unitatea centrala (CPU) il executa - si datele - aceasta poate fi de trei tipuri : RAM (Random Acces Memory) – locatiile din aceasta memorie pot fi accesate in orice ordine, este memoria vie a calculatorului; ROM (Read Only Memory)- continutul lor nu poate fi modificat, aceasta nu e o memorie volatila ; sau memoria flash - aceasta celula de memorie este similara cu celula de memorie EPROM, in care stergerea se realizeaza la nivel de bloc. Oricare dintre aceste tipuri de memorie pot fi folosite in implementarea unui microcontroler simplu ; si dispozitive periferice- aceste dispozitive sunt utilizate cu scopul de a realiza functionalitati diferite si de a permite o usoara interactiune cu mediul exterior).

Utilizarea microcontrolerului in sisteme incapsulate-integrate este primordiala, si totodata transparenta pentru un utilizator normal. Folosirea acestui sistem in controlarea caracteristicilor si actiunilor echipamentului fizic a ajutat in dezvoltarea a foarte multor industrii care imbunatesc viata omului si totodata care ajuta la evolutia stiintei. Acest dispozitiv a fost de folos in industria automobilelor - controlul climatizarii, sisteme de alarma; in industria electronica de consum – sisteme audiu - video, GPS-uri; - in realizarea perifericelor pentru calculatoare, chiar si in medicina.

Un microcontroler simplu este in esenta o versiune mai mica a computerelor personale, acesta incorporeaza toate componentele de baza ale unui computer personal la o scara mult mai mica. Microcontrolerul, in terminologie moderna, este vazut ca fiind similar unui chip, insa mai putin sofisticat, care controleaza majoritatea dispozitivelor electronice complexe pe care oamenii le folosesc zi de zi. Raspandirea microcontroalelor a fost una vasta si rapida datorita costului sau redus si a domeniului larg de utilizare a acestuia, fiind gasit aproape in orice industrie.

Desi putin cunoscut ca si terminologie si caracteristici, microcontrolerul este utilizat de catre majoritatea populatiei globului pamanteasc. Folosim microcontrolere pe scară largă în proiectarea sistemelor incorporate. Ii putem observa existenta intr-un numar imens de aplicatii si dispozitive din jurul nostru. Un microcontroler poate prezenta o eficienta si o performantă mai ridicata in anumite aplicatii decat orice computer pe care il folosim. Cele mai simple microcontrolere facilitează funcționarea sistemelor electromecanice care se gasesc in articolele de zi cu zi, cum ar fi cuptoare, frigidere, prăjitoare de pâine, dispozitive mobile, chei , sisteme de jocuri video, televizoare si sisteme de udare a gazonului. Acestea sunt, de asemenea, frecvente in masinile de birou, cum ar fi fotocopiatoarele, scanerele, aparatele de fax și imprimantele, precum si contoarele inteligente , bancomatele și sistemele de securitate. Microcontrolere mai sofisticate indeplinesc functii critice in aeronave, nave spațiale, nave oceanice, vehicule, sisteme medicale și de sustinere a vietii, precum si in roboti. In scenariile medicale, microcontrolerele pot regla operatiile unei inimi artificiale, rinichi sau alte organe. Ele pot fi, de asemenea, esențiale in functionarea dispozitivelor protetice.

Cerinta acestui proiect este implementarea pe o placa de dezvoltare FPGA, folosind proiectarea structurala a unui microcontroler simplu folosind limbajul hardware VHDL. Este necesara definirea atat a unor instructiuni generale, cat si a unor instructiuni speciale pentru controlul resurselor placii specificate. Acest proiect are ca obiectiv final determinarea unui rezultat corect matematic in urma parcurgerii unui set de instructiuni (descrise in cod masina) care defines operatii logice si aritmetice stocate intr-o memorie de tip ROM.

Programul executat de catre unitatea centrala (CPU) a microcontrolerului simplu implementat este alcatuit dintr-un set de instructiuni (descrierea acestora s-a realizat utilizand un cod masina standard, specific fiecarei operatie logica sau aritmetica necesara) a caror ordine si functionalitate permite calcularea sumei numerelor pare dintr-un sir - stocat intr-o memorie de tip RAM-. Mai mult, se cere sa se determine daca aceasta suma este sau nu un multiplu al unui numar dat stocat intr-un registru, iar rezultatul conditiei prezentate anterior sa fie retinut de asemenea intr-un registru (1 in cazul in care suma este multiplul numarului dat, 0 in caz contrar).

Pentru implemenarea structurala a microcontrolerului am ales utilizarea a cinci componente relativ elementare, pentru a putea realiza implementarea scopului fiecarei instructiuni reutilizand anumite blocuri (componente) astfel obtinandu-se o solutie optima, atat din punct de vedere al resurselor folosite, cat si din punct de vedere al intelegerii schemei.

Componente utilizate :

- un bistabil de tip D- pentru parcurgerea instructiunilor programului ;
- o memorie ROM- contine programul (setul de instructiuni) care permite realizarea cerintelor matematice din enunt;
- o memorie RAM – stocheaza sirul numerelor amintite mai sus;
- bloc de registre- permite stocarea unor «valori de lucru » si returnarea acestora la nevoie ;
- doua sumatoare -unul pentru calcularea adresei de salt, celalalt pentru generarea adresei urmatoarei instructiuni;
- o unitatea de extensia a unui numar ;
- o unitate aritmetica-logica- aceasta realizeaza implementarea fiecarei intructiuni gasite in memoria ROM.

Prezentarea pe scurt al capitolelor documentatiei:

- Rezumat- cuprinde descrierea pe scurt atat al conceptului de microcontroler simplu cat si a modalitatii de implementare aleasa.
-
- Fundamentare teoretica – contine explicarea in termini mai stiintifici a unor concepte intalnite atat in timpul studierii microcontrolerului, cat si in timpul parcurgerii implementarii acestui dispozitiv. Acest capitol vine in ajutorul intelegerii functionalitatilor microcontrolerului, si a termenilor de specialitate utilizati.
- Proiectare si implementare- aceast capitol va fi partea principala a raportului. El va contine toate detaliile care tin atat de alegerile facute pe parcursul realizarii proiectului (metoda experimentala, solutia), cat si de implementare si schema.
- Rezultatele experimentale- in acest capitol va fi necesara demonstratia functionarii corecte a microcontrolerului implementat si modul in care s-a facut testarea.

- Concluzii – capitolul va contine rezumatul experientei mele in urma realizarii acestui proiect : problemele intampinate pe parcurs, motivarea alegerilor mele si sustinerea modului in care am implementat dispozitivul.
- Bibliografie- contine toate resursele folosite.
- Anexe- contine codul sursa si toate schemele realizate.

3. Fundamentare teoretica

Microcontrolerul este un mic computer compact, fabricat in interiorul uni chip si utilizat in système de control automat. Este un control logic programabil economic.

Primul microcontroler a fost realizat de Michael Cochran si Gary Boone.

Tehnologia a fost dezvoltata intr-un mod uimitor datorita acestui dispozitiv si ne-a facut viata mai usoara ca niciodata, dezvoltand majoritatea industriilor.

In tehnologiile moderne, unele dispozitive cu microcontrolere constituie un design complex si sunt cababile sa aiba o lungime a cuvantului mai mare de 64 de biti. Microcontrolerele moderne sunt proiectate folosind arhitectura CISC (Complex Instruction Set Computer) care implica instructiuni de tip macro – aceasta este utilizata pentru a inlocui numarul de instructiuni mici-. Aceste microcontrolere functioneaza la un consum de energie mult mai mic in comparatie cu cele vechi.

O lista scurta a modelelor de microcontrolere ar fi : Altera, Atmel, ELAN, EPSON, Fujitsu, Maxim Integrated, NEC .

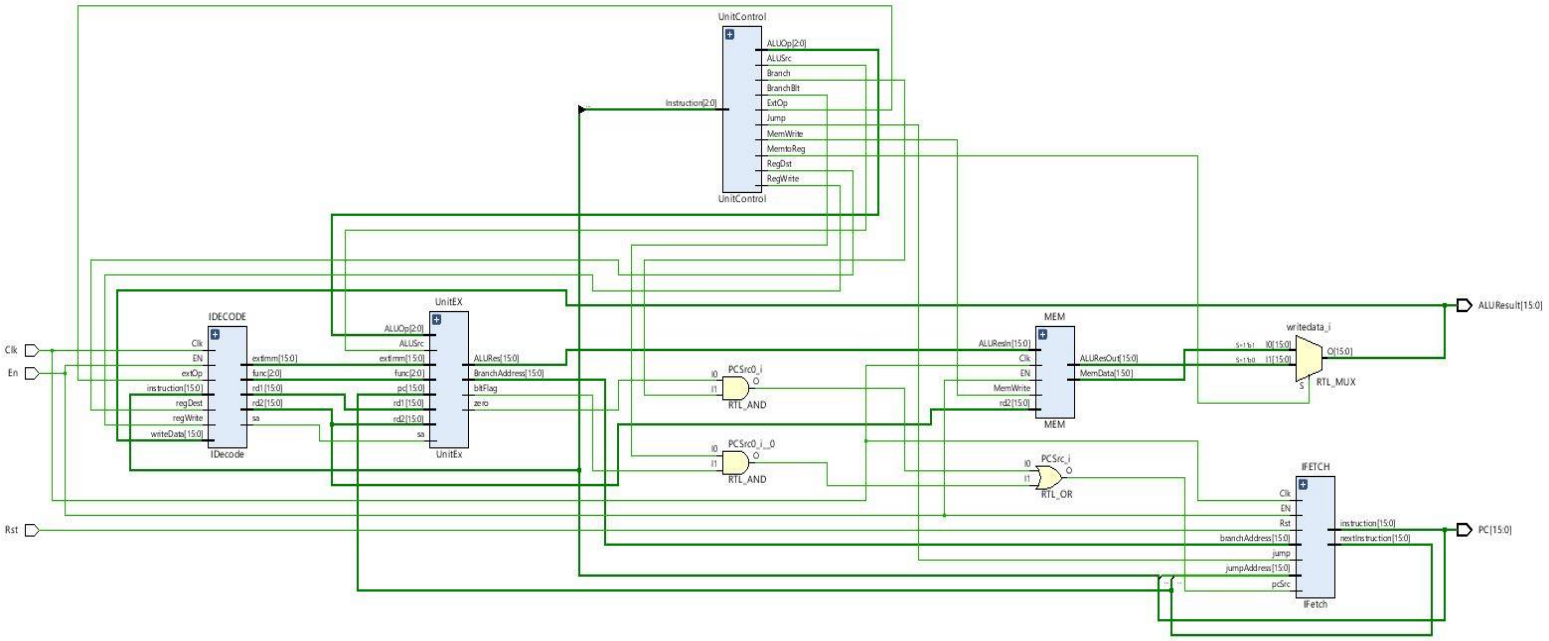
Pentru realizarea implementarii microcontrolerului simplu am ales descrierea a cinci module in limbaj VHDL. In urma studierii cartii “De la bit la procesor” – Florin Oniga si mai mult a subiectelor propuse si detaliate de John Hennessy și David Patterson in seria de carti Computer organization and design: the hardware/software interface am ajuns la concluzia ca solutia aleasa este una optima din mai multe puncta de vedere, principalul obiectiv fiind folosirea redusa a resurselor.

4. Proiectare si implementare

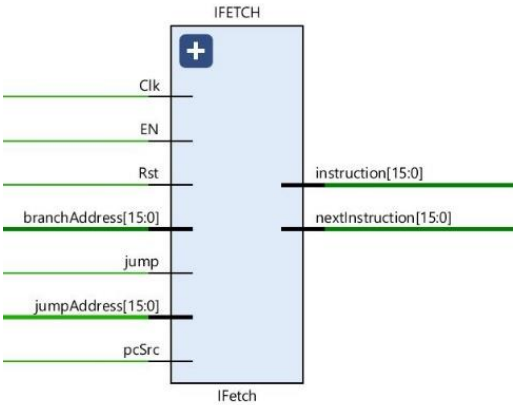
Functionarea corecta a microcontrolerul simplu, dispozitivul pe care a trebuit sa il implementez pentru acest proiect a fost demonstrata cu ajutorul simulatorului Vivado. Pentru a asigura corectitudinea datelor obtinute in simulator a fost necesara compararea acestora si cu rezultatele obtinute prin folosirea unui model analitic aplicat asupra programului (firmware) stocat in memorie ROM.

Am ales implementarea microcontrolerului folosind cinci componente elementare pentru a putea fi folosite in citirea si decodarea fiecarei instructiuni si realizarea scopului acestora. Caracterul general al componentelor permite utilizarea acestora de catre fiecare instructiune, fara a fi necesara

descrierea unui set componente specifice fiecarei instructiuni. Solutie aleasa reduce atat resursele necesare ale realizarii dispozitivului, cat si timpul necesar implementarii acestuia.

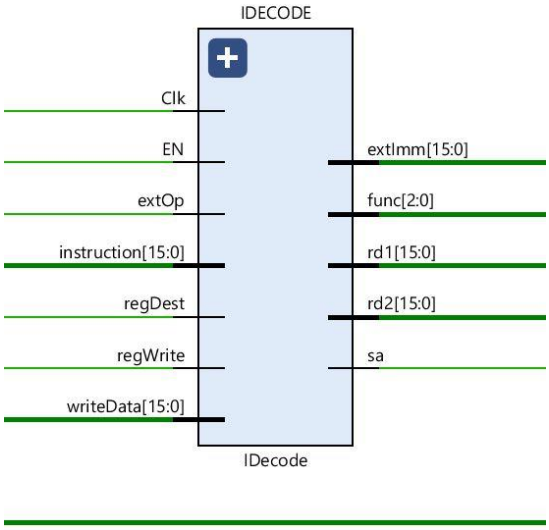


4.1 Instruction fetch



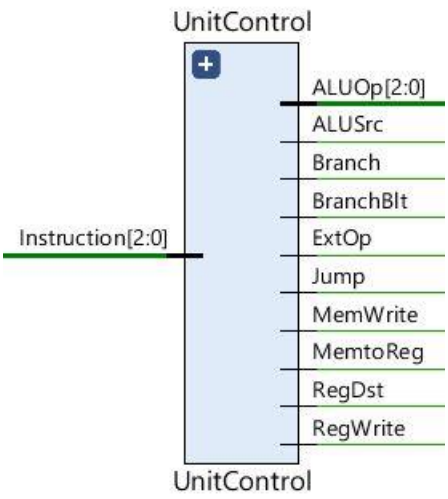
Componenta instruction fetch este componenta care extrage din memoria ROM instructiunea care trebuie executata. Semnalul de iesire instruction reprezeinta instructiune curenta si semnalul nextInstruction reprezinta urmatoarea instructiune in cazul in care semnalel de control jump si pcSrc sunt 0, branchAdress in cazul in care pcSrc este 1 (o adresa pentru saltul conditionat) sau jumpAdress in cazul in care jump este 1 (salt neconditionat).

4.2 Instruction decode



Componenta instruction decode reprezinta blocul de registre. In functie de tipul instructiunii regasite pe semnalul de intrare instruction blocul de registre are mai multe comportamente. Acest comportement este definit de catre semnalele de control determinate in urmatoare componenta. Toate informatiile necesare extragerii datelor se regasesc in codul masina din semnalul instruction. La nivel general pentru o instructiune de tip R, semnalele de iesire rd1 si rd2 reprezinta valorile registrelor asupra carora se vor executa operatiile logice sau aritmetice. In cazul in care intructiunea este de tip I valorile folosite sunt semnalele de iesire rd1 si extImm. Semnalele func si sa vor fi folosite pentru realizarea operatiilor. In cazul in care instructiunea necesita schimbarea valorii unui registru in functie de valoarea lui regDest (care alege registrul ce trebuie modificat) acel registru va lua valoarea writeData.

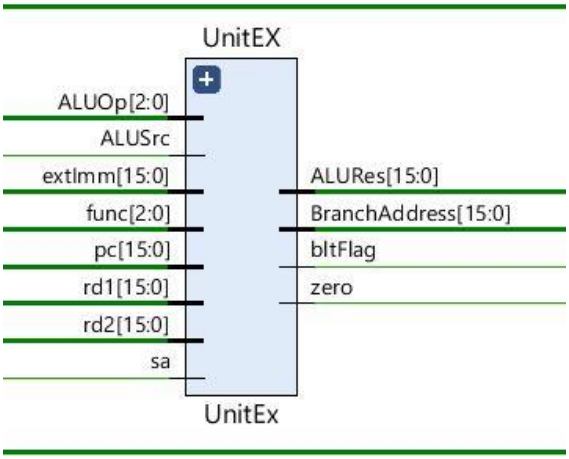
4.3 Unit Control



Unitatea de control genereaza semnalele de control necesare bunei functionari a microcontrolerului. In functie de ultimii 3 biti se determina aceste semnalele. Prin conventie instructiunile de tip R au ultimii 3 biti ‘000’ si au acelasi semnale de iesire. Urmatorul tabel va contine toate aceste semnale.

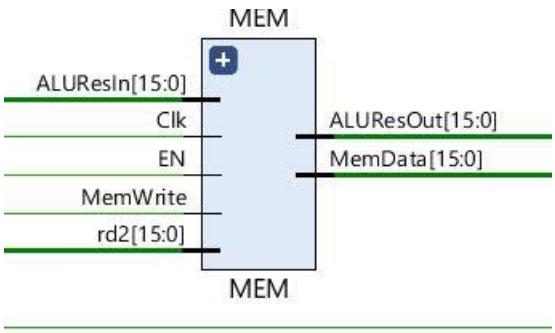
	RegDst	ExtOp	ALUSrc	Branch	Jump	MemWrite	MemtoReg	RegWrite	BranchBlt	AluOP
Tip R	1	0	0	0	0	0	0	1	0	000
ADDI	0	1	1	0	0	0	0	0	0	001
LW	0	1	1	0	0	0	1	1	0	010
ANDI	0	0	1	0	0	0	0	1	0	011
BEQ	0	1	0	1	0	0	0	0	0	100
J	0	0	0	0	1	0	0	0	0	000
BLT	0	1	0	0	0	0	0	0	1	110
SW	0	1	1	0	0	1	0	0	0	111

4.5 Unit execution



Unitatea de executie este componenta care realizeaza operatia logica sau aritmetica in functie de informatiile dobandite de componentele precedente. ALURes este rezultatul matematic, in aceasta componenta se realizeaza si calcularea adresei de salt conditionat, iar rezultatul ‘1’ unei porti si dintre bltFlag si BranchBlt va semnifica faptul ca se realizeaza o instructiune de tip BLT, iar rezultatul ‘1’ unei porti si dintre zero si Branch va semnifica faptul ca se realizeaza o instructiune de tip BEQ.

4.6 Memory



Aceasta componenta este necesara doar pentru intructiunile de tip LW si SW. Modulul foloseste o memorie de tip RAM pentru a stoca diferite informatii. In cazul in care avem o intructiune de tip LW se va stoca intr-un registru valoarea din memorie aflata la adresa indicata, in cazul in care avem o intrctiune de tip SW la adresa indicate va fi retinuta valoarea de pe semnalul de intare ALUResIN.

5. Rezultate experimentale

Mediul software utilizat in proiectarea microcontrolerului simplu a fost Vivado Design Suite 2020.1. Descrierea modulelor necesarea a fost realizata prin limbajul hardware VHDL. Sistemul de operare folosit a fost Windows 10 datorita experientei pe care am dobandit-o de-a lungul anilor, acest sistem de operare fiind dominant in muca mea. Simulatorul utilizat pentru demonstrarea functionalitatii corecta a proiectului a fost simulatorul mediului de dezvoltare prezentat mai sus.

Pentru o urmarire mai usoara a componentelor utilizate vor fi descrise toate modulele.

IFetch- contine o memorie ROM, un bistabil D, precum si doua multiplexoarea pentru alegerea urmatoarei instructiuni.

IDecode- contine un registru , un multiplexor si de asemena o unitate de extensie a unui numar cu semn.

UnitControl- nu contine nici o componenta bine definita.

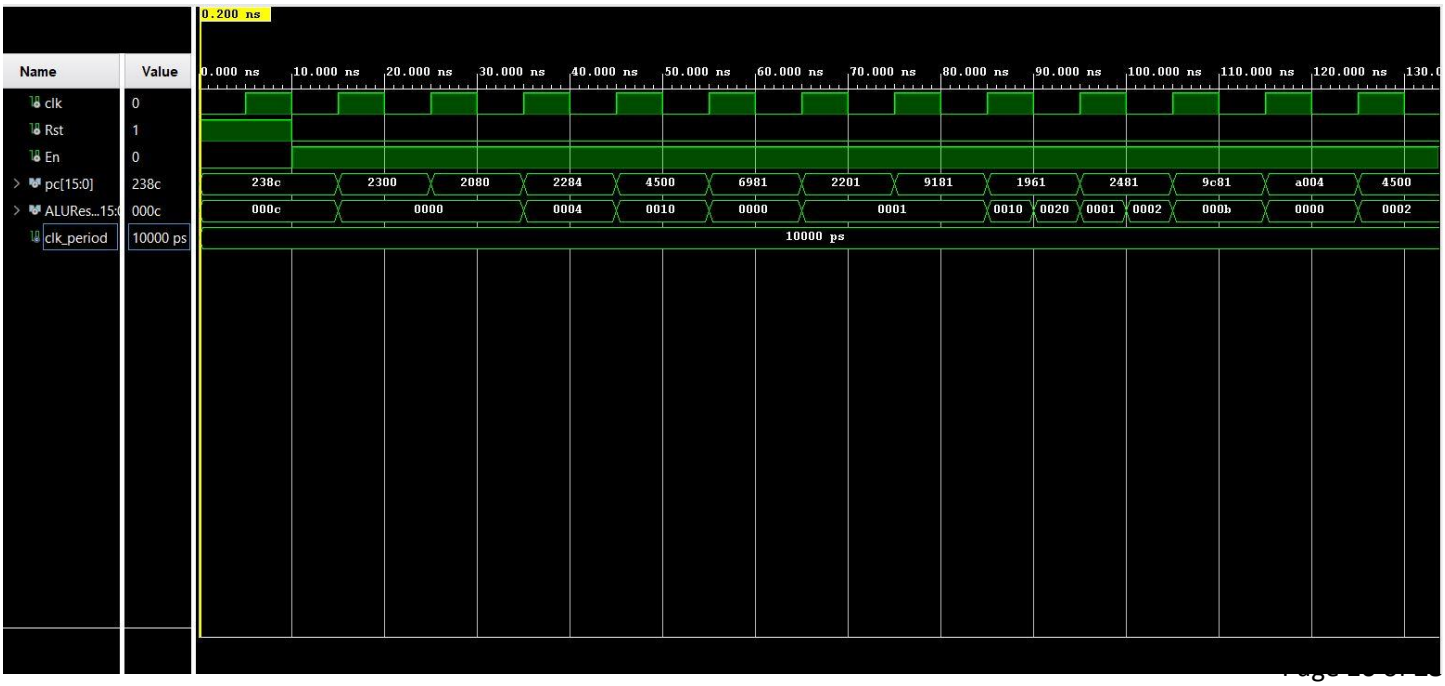
UnitEX- contine pe langa logica de baza a componentei si un multiplexor.

Mem- contine o memorie de tip RAM.

Modulul principal- continue pe langa instantierea modulelor prezentate mai sus, doua porti si a caror rezultate vor ajunge intr-o poarta de tip sau si un registru de deplasare.

Procedura de testare utilizata a fost crearea unui banc de test. Valorile introduse pe semnalele de intrare a fost asignate astfel incat sa se poata demonstra bunul mers al dispozitivului.

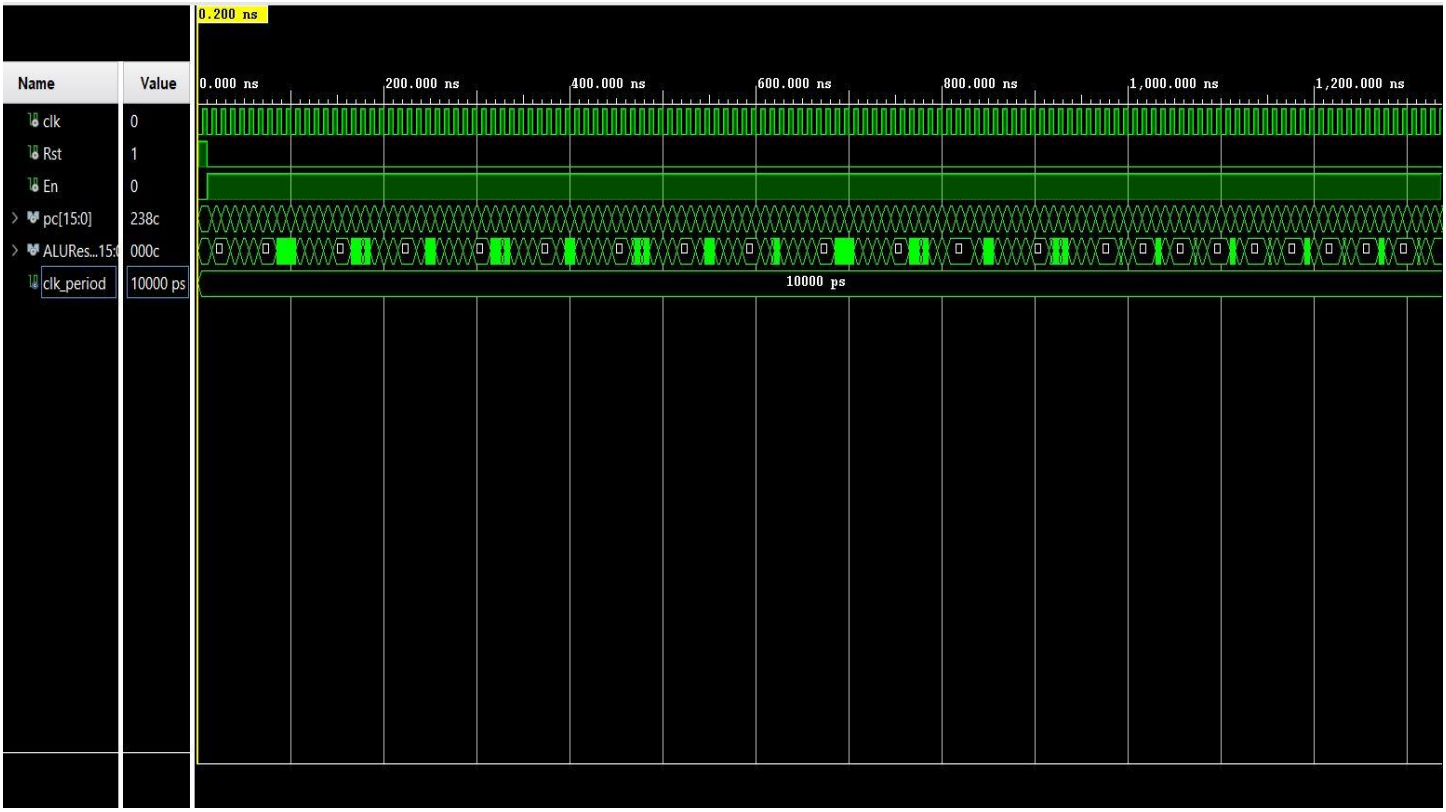
Rezultatul simularii pentru primele instructiuni, acestea pot fi verificate :



Pentru verificarea corectitudinii voi prezenta si un tabel a caror rezultate au fost deduse aplicand in model analitic pentru primele intrusctiuni.

Instructiune	Cod masina(binar) (PC)	AluRes(hexa)
addi $7,$0,12	001_000_111_0001100	C
addi $6,$0,0	001_000_110_0000000	0
addi $1,$0,0	001_000_001_0000000	0
addi $5,$0,4	001_000_101_0000100	4
lw $2,0($1)	010_001_010_0000000	10
andi $3,$2,1	011_010_011_0000001	0
addi $4,$0,1	001_000_100_0000001	1
beq $3,$4,1	100_100_011_0000001	1
add $6,$6,$2	000_110_010_110_0_001	10
addi $1,$1,1	001_001_001_0000001	1
beq $1,$7,1	100_111_001_0000001	B
j 4	101_0000000000100	0
lw $2,0($1)	010_001_010_0000000	2

O privere de ansamblu a intregii simulari :



6. Concluzii

Cerinta acestui proiect este implementarea pe o placa de dezvoltare FPGA, folosind proiectarea structurala a unui microcontroler simplu folosind limbajul hardware VHDL. Este necesara definirea atat a unor instructiuni generale, cat si a unor instructiuni speciale pentru controlul resurselor placii specificate. Acest proiect are ca obiectiv final determinarea unui rezultat corect matematic in urma parcurgerii unui set de instructiuni (descrise in cod masina) care defines operatii logice si aritmetice stocate intr-o memorie de tip ROM.

Studierea si implementarea microcontrolerului simplu mi-a dezvoltat abilitatea de a avea o logica rationala si de a scrie cod in limbaj hardware. Am aflat vasta utilizare si importnata acestui mic dispozitiv, acesta fiind indispensabil in viata omului.

Principalul avantaj al metodei de implementare alese este posibiliatatea de a reutiliza componentele pentru intreg setul de instructiuni (scris in cod masina), fara a fi necesar descrierea componentelor in mod particular. Un dezavantaj ar fi faptul ca fiecare instructiune se desfasoara pe un ciclu de ceas.

Acest microcontroler simplu poate fi utilizat pentr diverse calcula matematice, sau cu o logica buna in descrierea programlui (firmaware) poate aduce mari contributii si in domenii mai complexe.

Pentru dezvoltarea acestui proiect o sugestie ar fi marirea numarului de biti pe instructiune (fiind posibile instructiuni chiar mai mari de 64 de biti). O alta sugestie ar fi diversificarea setului de intructiuni introducand noi functrionalitati (se pastreaza structura, generalitatea componentelor permitand si executarea altor instructiuni).

7. Bibliografie

- https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf
- https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/366-0.pdf
- https://en.wikipedia.org/wiki/List_of_common_microcontrollers#:~:text=%20List%20of%20common%20microcontrollers%20%201%20Altera.,18%20Xilinx.%20%2019%20XMOS.%20%20More%20
- https://internetofthingsagenda.techtarget.com/definition/microcontroller
- https://simple.wikipedia.org/wiki/Microcontroller

8. Anexe

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


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.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 IFetch is
 Port (Clk: in std_logic;
       Rst: in std_logic;
       EN: in std_logic;
       branchAddress: in std_logic_vector(15 downto 0);
       jumpAddress: in std_logic_vector(15 downto 0);
       jump: in std_logic;
       pcSrc: in std_logic;
       instruction: out std_logic_vector(15 downto 0);
       nextInstruction:  out std_logic_vector(15 downto 0));
end IFetch;

architecture Behavioral of IFetch is
type MEM is array (0 to 22) of std_logic_vector(15 downto 0);
signal ROM:MEM:=(B"001_000_111_0001100", -- addi $7,$0,12 - 12 numere in sir
                 B"001_000_110_0000000", -- addi $6,$0,0 - suma numerelor
                 B"001_000_001_0000000", -- addi $1,$0,0 - counter in sir
                 B"001_000_101_0000100", -- addi $5,$0,4 - multiplu

                 B"010_001_010_0000000", -- lw $2,0($1) - ia cate un element din sir
                 B"011_010_011_0000001", -- andi $3,$2,1 - daca e numar par reg3 va avea valoarea 0,
altfel 1
                 B"001_000_100_0000001", -- addi $4,$0,1 - in reg4 incarca valoarea 1
                 B"100_100_011_0000001", -- beq $3,$4,1 - daca numarul e impar va sari peste suma
                 B"000_110_010_110_0_001", -- add $6,$6,$2 - daca numarul e par se va aduna la suma
```

```
            B"001_001_001_0000001", -- addi $1,$1,1 - se incrementeaza pozitia in sir
            B"100_111_001_0000001", -- beq $1,$7,1 - in cazul in care pozitia e 13 inseamna ca am evaluat cele 12 numere din sir
            B"101_0000000000100", -- j 4 - in cazul in care nu am ajuns la 13 se sare la citirea numerelor din memorie

            B"001_110_001_0000000", -- addi $1,$6,0 - se incarca in reg1 suma, deoarece aceasta va suferi scaderi succesive si astfel se va pierde valoarea
            B"100_000_001_0000100", -- beq $1,$0,4 - daca suma e egala cu 0 inseamna ca e multiplu si se sare la validarea cond suma e un multiplu de numar dat
            B"100_001_101_0000011", -- beq $5,$1,3 - daca suma e egala cu numarul dat se sare la validarea cond suma e un multiplu de numar dat
            B"110_001_101_0000110", -- blt $5,$1,6 - daca suma ramasa e mai mica decat numarul dat inseamna ca aceasta nu e multiplu si se sare la negarea conditiei (adica valoarea 0-False in reg7)
            B"000_001_101_001_0_010", -- sub $1,$1,$5 - in cazul in care e mai mare se face scaderea numarului dat din suma ramasa
            B"101_0000000001110", -- j 14 - se sare la compararea sumei ramase cu multiplu

            B"001_000_111_0000001", -- addi $7,$0,1 - se incarca valoare 1 in cazul in care e multiplu de numar dat
            B"001_000_001_0000001", -- addi $1,$0,1 - se incarca valoarea 1 in reg1
            B"111_001_110_0000000", -- sw $6 0($1) - daca e suma e multiplu de numar dat se incarca suma la adresa 1 in memorie
            B"100_111_001_0000001", -- beq  $1,$7,1 - in cazul in care a fost multiplu de numar dat se sare peste setarea reg7 cu valoarea 0 specifica lui False.
            B"001_000_111_0000000"); -- addi $7,$0,0  - se incarca valoarea 0 in caz ca nu e multiplu
signal pc : std_logic_vector(15 downto 0) := (others => '0');
signal pcAux,mux1,mux2: std_logic_vector(15 downto 0);
begin
process(Clk)
   begin
      if rising_edge(Clk) then
         if Rst = '1' then
            pc <= (others => '0');
         elsif EN = '1' then
            pc <= mux2;
         end if;
      end if;
end process;

instruction <= ROM(conv_integer(pc(15 downto 0)));

pcAux <= pc + 1;
nextInstruction <= pcAux;
```

```
    process(pcSrc, pcAux, BranchAddress)
       begin
          if (pcSrc='1') then
             mux1 <= branchAddress;
          else
             mux1 <= pcAux;
         end if;
    end process;

    process(jump, mux1, jumpAddress)
       begin
          if (jump='1') then
             mux2 <= jumpAddress;
          else
             mux2 <= mux1;
          end if;
    end process;

    end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.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 IDecode is
```

```
 Port ( Clk: in std_logic;
        EN: in std_logic;
        instruction: in std_logic_vector(15 downto 0);
        writeData: in std_logic_vector(15 downto 0);
        regWrite: in std_logic;
        regDest: in std_logic;
        extOp: in std_logic;
        rd1: out std_logic_vector(15 downto 0);
        rd2: out std_logic_vector(15 downto 0);
        extImm: out std_logic_vector(15 downto 0);
        func: out std_logic_vector(2 downto 0);
        sa: out std_logic);
end IDecode;

architecture Behavioral of IDecode is
signal writeAddress: std_logic_vector(2 downto 0);
type reg_array is array(0 to 7) of std_logic_vector(15 downto 0);
signal reg_file : reg_array := (others => X"0000");
begin

 with regDest select
        writeAddress <= instruction(6 downto 4) when '1', -- rd
                        instruction(9 downto 7) when '0', -- rt
                        (others => '0') when others; -- unknown

 process (Clk)
  begin
   if falling_edge(Clk) then
    if EN='1' and regWrite='1' then
     reg_file(conv_integer(writeAddress)) <= writeData;
```

```
    end if;
   end if;
 end process;

 rd1 <= reg_file(conv_integer(instruction(12 downto 10)));
 rd2 <= reg_file(conv_integer(instruction(9 downto 7)));

 extImm(6 downto 0) <= instruction(6 downto 0);
 with extOp select
     extImm(15 downto 7) <= (others => instruction(6)) when '1',
                            (others => '0') when '0',
                            (others => '0') when others;

sa<=instruction(3);
func<=instruction(2 downto 0);
end Behavioral;

----------------------------------------------------------------------------------


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 UnitControl is
  Port (   Instruction : in std_logic_vector(2 downto 0);
           RegDst : out std_logic;
           ExtOp : out std_logic;
           ALUSrc : out std_logic;
           Branch : out std_logic;
           Jump : out std_logic;
           ALUOp : out std_logic_vector(2 downto 0);
           MemWrite : out std_logic;
           MemtoReg : out std_logic;
           RegWrite : out std_logic;
           BranchBlt: out std_logic);
end UnitControl;

architecture Behavioral of UnitControl is

begin
 process (Instruction)
  begin
   RegDst <= '0';
   ExtOp <= '0';
   ALUSrc <= '0';
   Branch <= '0';
   Jump <= '0';
   MemWrite <= '0';
   MemtoReg <= '0';
   RegWrite <= '0';
   BranchBlt <='0';
```

```
ALUOp <= "000";
case (Instruction) is
 when "000" => --R Type
  RegDst <= '1';
  RegWrite <='1';
  ALUOp <= "000";
 when "001" => --ADDI
  ExtOp <= '1';
  ALUSrc <= '1';
  RegWrite <= '1';
  ALUOp <= "001";
 when "010" => --LW
  ExtOp <= '1';
  ALUSrc <= '1';
  MemtoReg <= '1';
  RegWrite <= '1';
  ALUOp <= "010";
 when "011" => --ANDI
  ALUSrc <= '1';
  RegWrite <= '1';
  ALUOp <= "011";
 when "100" => --BEQ
  ExtOp <= '1';
  Branch <= '1';
  ALUOp <= "100";
 when "101" => -- J
  Jump <= '1';
 when "110" => --BLT
  ExtOp <= '1';
  BranchBlt <= '1';
```

```
     ALUOp <= "110";
    when "111" => --SW
     ExtOp <= '1';
     ALUSrc <= '1';
     MemWrite <= '1';
     ALUOp <= "111";
    when others =>
     RegDst <= '0';
     ExtOp <= '0';
     ALUSrc <= '0';
     Branch <= '0';
     Jump <= '0';
     MemWrite <= '0';
     MemtoReg <= '0';
     RegWrite <= '0';
     BranchBlt <= '0';
     ALUOp <= "000";
    end case;
  end process;
end Behavioral;

----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.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 UnitEx is
 Port (  pc: in std_logic_Vector (15 downto 0);
        rd1: in std_logic_Vector (15 downto 0);
        rd2: in std_logic_Vector (15 downto 0);
        extImm:  in std_logic_Vector (15 downto 0);
        func:  in std_logic_Vector (2 downto 0);
        sa:  in std_logic;
        ALUSrc: in std_logic;
        ALUOp: in std_logic_Vector (2 downto 0);
        zero: out  std_logic;
        BranchAddress: out std_logic_Vector (15 downto 0);
        ALURes: out std_logic_Vector (15 downto 0);
        bltFlag: out  std_logic);
end UnitEx;

architecture Behavioral of UnitEx is
signal muxAux:  std_logic_Vector (15 downto 0);
signal result:  std_logic_Vector (15 downto 0);
signal ALUCtrl:  std_logic_Vector (2 downto 0);
begin

with ALUSrc select
```

```
    muxAux <= RD2 when '0',
                extImm when '1',
                (others => '0') when others;


process(ALUOp, func)
    begin
      case ALUOp is
        when "000" => --R-Type
                case func is
                  when "001" => ALUCtrl <= "000"; -- ADD
                  when "010" => ALUCtrl  <= "001"; -- SUB
                  when "011" => ALUCtrl  <= "011"; -- SRL
                  when "100" => ALUCtrl  <= "100"; -- SLL
                  when "101" => ALUCtrl  <= "010"; -- AND
                  when "110" => ALUCtrl  <= "101"; -- OR
                  when "111" => ALUCtrl  <= "110"; -- XOR
                  when others => ALUCtrl <= "111";
                end case;
        when "001" => ALUCtrl  <="000";--addi
        when "010" => ALUCtrl  <= "000";--LW
        when "011" => ALUCtrl  <="010";--andi
        when "100" => ALUCtrl  <="001";--beq
        when "110" => ALUCtrl  <="001";--blt
        when "111" => ALUCtrl  <="000";--sw
        when others => ALUCtrl  <="111";
      end case;
    end process;
  ALU : process (ALUCtrl, muxAux, rd1, sa)
        begin
          case ALUCtrl is
```

```
            when "000" => result <= rd1 + muxAux; --  +
            when "001" => result <= rd1 - muxAux; -- -
            when "011" => result <= '0' & rd1(15 downto 1); --SRL
            when "100" => result <= rd1(14 downto 0) & '0'; --SLL
            when "010" => result<= rd1 and muxAux; -- AND
            when "101" => result<= rd1 or muxAux; -- OR
            when "111" => result <= rd1 xor  muxAux; -- XOR
            when others => result <= x"0000";
            end case;
        end process;
        zero<='1' when result=x"0000" else '0';
        bltFlag<='1' when result (15)='1' else '0';


ALURes <=result;
BranchAddress<=pc+extImm;

end Behavioral;

----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.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 MEM is
  Port ( Clk : in std_logic;
         EN : in std_logic;
         MemWrite: in std_logic;
         ALUResIn: in std_logic_vector (15 downto 0);
         rd2: in std_logic_vector (15 downto 0);
         MemData: out std_logic_vector (15 downto 0);
         ALUResOut: out std_logic_vector (15 downto 0));
end MEM;

architecture Behavioral of MEM is
type RAM is array (0 to 15) of std_logic_vector(0 to 15);
signal mem_RAM: RAM := (
   X"0010",--16
   X"0002",--2
   X"0005",--5
   X"0014",--20
   X"002B",--43
   X"000A",--10
   X"001B",--27
   X"0013",--19
   X"0018",--24
   X"001E",--30
   X"0001",--1
```

```
    X"000A",--10
    X"0007",--7
    others =>X"FFFF");

begin
 process (clk)
  begin
    if rising_edge(clk)then
     if EN='1' and MemWrite='1' then
      mem_RAM(conv_integer(ALUResIN(3 downto 0)))<=RD2;
     end if;
    end if;
 end process;

 MemData <= mem_RAM(conv_integer(ALUResIn(3 downto 0)));
 ALUResOut<=ALUResIn;

end Behavioral;

----------------------------------------------------------------------------------



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 ModulPrincipal is
  Port (Clk: in std_logic;
        Rst: in std_logic;
        En: in std_logic;
        PC:  out std_logic_vector (15 downto 0);
        ALUResult: out std_logic_vector (15 downto 0));
end ModulPrincipal;

architecture Behavioral of ModulPrincipal is

signal Instruction, nextInstruction, RD1, RD2, writedata, extImm : STD_LOGIC_VECTOR(15 downto 0);
signal JumpAddress, BranchAddress, ALURes, ALUResAUX, MemData : STD_LOGIC_VECTOR(15 downto
0);
signal func : STD_LOGIC_VECTOR(2 downto 0);
signal sa, zero : STD_LOGIC;
signal PCSrc,BranchBlt,bltFlag : STD_LOGIC;
signal RegDest, ExtOp, ALUSrc, Branch, Jump, MemWrite, MemtoReg, RegWrite : STD_LOGIC;
signal ALUOp :  STD_LOGIC_VECTOR(2 downto 0);

begin
IFETCH: entity work.IFetch
Port map (Clk=> Clk,
          Rst=> Rst,
          EN=> En,
          branchAddress=>  branchAddress,
          jumpAddress=>    jumpAddress,
          jump=> jump,
```

```
        pcSrc=> pcSrc,
        instruction=> instruction,
        nextInstruction=> nextInstruction);
IDECODE: entity work.IDecode
 Port map (Clk=> Clk,
        EN=> En,
        instruction=> instruction,
        writeData=> writeData,
        regWrite=> regWrite,
        regDest=> regDest,
        extOp=> extOp,
        rd1=> rd1,
        rd2=> rd2,
        extImm=> extImm,
        func=> func,
        sa=> sa);
UnitControl: entity work.UnitControl
  Port map (Instruction=> instruction (15 downto 13),
         RegDst => regDest,
         ExtOp=> extOp,
         ALUSrc=> ALUSrc,
         Branch=> branch,
         Jump=> jump,
         ALUOp=> ALUOp,
         MemWrite=> MemWrite,
         MemtoReg=> MemtoReg,
         RegWrite=> RegWrite,
         BranchBlt=>BranchBlt );
UnitEX: entity work.UnitEX
    Port map ( pc=> nextInstruction,
```

```
            rd1=> rd1,
            rd2=> rd2,
            extImm => extImm,
            func=> func,
            sa=> sa,
            ALUSrc=> ALUSrc,
            ALUOp=> ALUOp,
            zero=> zero,
            BranchAddress=> BranchAddress,
            ALURes=> ALURes,
            bltFlag=> bltFlag);
MEM: entity work.MEM
  Port map ( Clk=> Clk,
            EN=> En,
            MemWrite=> MemWrite,
            ALUResIn=> ALURes,
            rd2=> rd2,
            MemData=> MemData,
            ALUResOut=> ALUResAUX);

 with MemtoReg select
    writedata <= MemData when '1',
     ALUResAUX when '0',
     (others => '0') when others;

PCSrc<= (zero and branch) or (BranchBlt and bltFlag);
jumpAddress <=nextinstruction(15 downto 13) & instruction(12 downto 0);
pc<=instruction;
ALUResult<=writeData;
 end behavioral;
```