**Proiectarea unei unități centrale**

**Proiect Structura sistemelor de calcul**

**Student: Teocan Patricia Claudia**

**Grupa: 30234**

Cuprins

[1. Introducere 3](#_Toc121327469)

[1.1 Context 3](#_Toc121327470)

[1.2 Specificații 3](#_Toc121327471)

[1.3 Obiective 4](#_Toc121327472)

[2. Studiu bibliografic 4](#_Toc121327473)

[3. Design 5](#_Toc121327474)

[4. Implementare 7](#_Toc121327475)

[a. Instruction fetch 7](#_Toc121327476)

[b. Instruction decode 8](#_Toc121327477)

[c. Instruction execute 9](#_Toc121327478)

[d. Memory 10](#_Toc121327479)

[e. Test env 10](#_Toc121327480)

[5. Testare și implementare 11](#_Toc121327481)

[6. Concluzii 11](#_Toc121327482)

# Introducere

## Context

Principalul scop al proiectului este implementarea și funcționarea ale unui circuit cu 20 instrucțiuni ce respectă standardul IEEE configurate pe 16 biți. Principalele funcționalități ale circuitului sunt efectuarea operațiilor de tip aritmetic, logic, de transfer, de salturi condiționate sau necondiționate, rotiri. Se pot realiza operații de adunare și scădere, de tip shift și mai multe tipuri de rotiri la baza biților, la stânga sau la dreapta, aritmetice sau logice, salturi la alte instrucțiuni sau operațiile de tip AND, OR, XOR, NOT, de comparație, întregul cod fiind scris in limbajul de descriere hardware VHDL si utilizându-se IDE-ul VIVADO si plăcuța BASYS 3.

Scopurile acestui dispozitiv sunt variate. Principalul mod în care acesta poate fi folosit este în varianta lui inițiala, drept circuit propriu-zis pentru a efectua operațiile pe care acesta le are implementate: operații pe 16 biți ce respecta formatul IEEE. O altă opțiune ar fi utilizarea drept aparat periferic și utilizarea funcționalităților acestuia de către al dispozitiv. În plus, mai există varianta de integrare a acestuia într-un alt circuit si folosirea instrucțiunilor in cadrul lui, mapându-se foarte ușor în stil structural, utilizând codul VHDL.

## Specificații

Pentru început, menționez modul de implementare și utilizare al circuitului. Întregul cod este scris în limbajul bine cunoscut de descriere hardware, și anume VHDL în care am realizat multiple proiecte în anii trecuți de facultate la alte materii de tip hardware, precum Proiectarea Sistemelor Numerice, Arhitectura Calculatoarelor. Instrucțiunile descrise respectă standardul IEEE. Implementarea si simularea circuitului este realizată cu ajutorul IDE-ului de la XILINX, VIVADO. Pentru utilizarea dispozitivului și demonstrarea funcționalităților acestuia se folosește plăcuța din gama XILINX, BASYS 3. Codul VHDL se încărcă pe aceasta și circuitul poate fi utilizat folosind butoanele, switch-urile, afișoarele cu 7 segmente.

Pentru ca dispozitivul să fie mai ușor de folosit de către utilizatori, atât datele de intrare, introduse de acesta, cât și datele de ieșire sunt într-un limbaj înțeles de către om, întrucât se folosește sistemul hexa, cu cel binar cu care lucrează circuitul. La nivelul executării operațiilor, datele sunt interpretate în sistem binar, cum este specific tuturor circuitelor hardware. Astfel, este nevoie de 2 componente de conversie a datelor: unul pentru input din hexa în binar și unul pentru output din binar în hexa. În plus, utilizatorul are la dispoziție o funcție de tip RESTART ce poate fi folosită în orice moment al utilizării dispozitivului în caz că acesta ar fi ales o operație nedorita sau a introdus greșit unul dintre numere. Odată cu apelarea acestei funcționalități se vor șterge toate datele introduse sau selectate de utilizator în cadrul ultimei utilizări a circuitului. Numere introduse de acesta vor fi șterse complet din memoria de care se folosește dispozitivul,

dar la fel si operațiile selectate pentru returnarea unui rezultat dorit. Ceea ce se întâmplă la nivel intern, este revenirea la o stare inițială setata de programator în implementarea device-ului. După folosirea acestei funcții, circuitul poate fi folosit în mod obișnuit.

Vorbind despre nivelul intern al circuitului, modul în care acesta este implementat este cel structural. Se folosesc bistabile și porți logice de diferite tipuri care vor fi mapate. Modul de lucru este cel de împachetare, întrucât se începe de la cele mai mici circuite care sunt utilizate drept componente propriu-zise. În cadrul implementării se folosesc 9 registre interne, dintre care unul are rolul de acumulator, iar celelalte 8 sunt registre generale. În plus, tipul de adresare este cea indirectă și principiul care sta la baza acesteia este cel de stivă care funcționează în mod LIFO. Mai exact, ultimul element introdus este primul extras din structura de date.

## Obiective

Obiectivul acestui proiect este implementarea, simularea și asigurarea funcționalității corecte unui circuit ce are implementate 20 de instrucțiuni cu numere întregi și în complement față de 2 pe 16 biți. Pe lângă buna funcționare a dispozitivului și corectitudinea rezultatelor returnate de circuit, este important ca utilizatorul să îl poată folosi ușor iar rezultatul să fie înțeles de către acesta.

Modul în care utilizatorul folosește circuitul este cu ajutorul plăcuței BASYS 3. Rezultatul este vizibil pe afișoarele cu 7 segmente. Valorile suportate de dispozitiv sunt limitate de cei 16 biți. La nivel hexa, se pot folosi numere cu în complement față de 2 cu valori cuprinse în intervalul de la −32,768 și până la 32,767.

# Studiu bibliografic

În cadrul acestui proiect, numerele sunt stocate pe 16 biți astfel: primul bit este rezervat pentru stocarea semnului numărului (0 pentru numerele pozitive, respectiv 1 pentru cele negative) iar restul biților reprezintă valoarea în modul. Pentru ca valorile să poate fi interpretate corect de către VHDL, datele de intrare vor fi convertite în complement față de 2. Așadar, valorile respectă standardul IEEE al limbajului de descriere hardware VHDL.

|  |  |
| --- | --- |
| semn | valoare |

1 bit 15 biți

Exemple

|  |  |
| --- | --- |
| zecimal | 26,013 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| hexa | 6 | 5 | 9 | D |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| binar | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

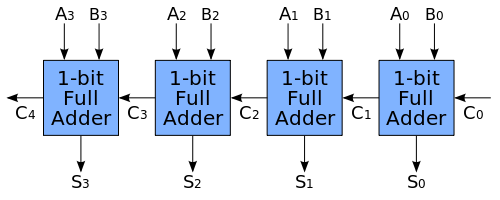
* Operații aritmetice -3
* De adunare și scădere pe numere de 16 biți, cu semn implementate prin cascadare de sumatoare complete (2)
* De shiftare aritmetică la dreapta în care pe poziția rămasă liberă se pune 1 în cazul în care numărul a fost negativ și 0 în caz contrar (1)
* Operații logice -11
* De shiftare logică stânga și la dreapta în care pe poziția rămasă liberă se pune 0 (2)
* De rotire logică la stânga și la dreapta în care se shiftează toți biții iar pe poziția rămasă liberă se pune valoarea bitului care a fost eliminat (2)
* Clasice pe biți (AND, OR, XOR, NOT, NAND, NOR, NXOR) (7)
* Salturi necondiționate
* Salturi condiționate
* Operații de comparare
* Realizate cu ajutorul operațiilor de adunare și scădere
* Operații de testare
* Test dacă un număr este pozitiv sau negativ verificând primul bit

Implementarea operațiilor menționate mai sus se realizează cu bistabile și porți logice. Se începe de la cea mică componentă, crescând treptat complexitatea componentelor. În componența fiecărei porți se vor regăsi componente mai micuțe implementate anterior, respectând acest principiu. Pentru realizarea fiecărei componente, codul va fi descris în mod structural fără a avea descrieri comportamentale. La fiecare pas se vor mapa componente anterioare.

# Design

* Operații aritmetice
* Adunare

Pentru realizarea adunării a două numere pe 16 biți în complement față de 2, sunt cascadate 4 sumatoare complete pe 4 biți ce sunt implementate la rândul lor prin cascadarea a 4 sumatoare complete pe 1 bit. Transportul de intrare al sumatorului pe 16 biți este deja setat ca mereu să fie valoarea ‘0’. Astfel, utilizatorul nu trebuie să trimită niciun fel de semnal pentru această intrare pentru a ne asigura că nu se strecoară greșeli în procesul de efectuare al acestei operații.



Sumator pe 4 biți prin cascadarea

a 4 sumatoare pe 1 bit

* Scădere

Modul de implementare folosește circuitul de adunare în întregime. Singura diferență dintre adunare și scădere este reprezentată de valorile de intrare ale sumatorului. Circuitul de intrare are integrată și o componentă care înmulțește cel de-al doilea operand cu -1 pentru putea fi realizată adunarea cu opusul. Astfel este realizată operația de scădere a 2 numere.

* Shiftare aritmetică la dreapta

Este realizată cu ajutorul unui registru pe 16 biți, aceștia fiind deplasați cu atâtea poziții câte au fost precizate cu ajutorului celui de-al doilea termen la dreapta, iar pentru determinarea valorii care va veni pe primii biți, se folosește un multiplexor pe 1 bit a cărui selecție va intra vechiul prim bit al numărului.

* Operații logice
* Shiftare logică la stânga și dreapta

Sunt realizate cu ajutorul unui registru pe 16 biți, aceștia fiind deplasați cu atâtea poziții câte au fost precizate cu ajutorului celui de-al doilea termen la stânga sau la dreapta, iar pe biții rămași liberi se va pune valoarea ‘0’.

* Rotații logice la stânga și la dreapta

Sunt realizate cu ajutorul unui registru pe 16 biți, aceștia fiind rotiți de atâtea ori câte au fost precizate cu ajutorului celui de-al doilea termen la stânga sau la dreapta.

* Operații logice clasice (AND, OR, XOR, NOT, NAND, NOR, NXOR)

Sunt implementate cu ajutorul porților logice clasice specifice. Valorile de intrare și de ieșire sunt pe 16 biți.

* Salt necondiționat

Saltul necondiționat se execută cu ajutorul unui registru în care se stochează adresă dorită.

* Operații de testare
* De egalitate

Cu ajutorului circuitului de scădere si celui care verifică dacă un număr este egal cu zero, utilizatorul va putea primi o confirmare sau o infirmare pentru egalitatea celor 2 numere introduse.

* Mai mic

Cu ajutorului circuitului de scădere si celui care precizează dacă un număr este pozitiv sau nu, utilizatorul va putea primi o confirmare sau o infirmare pentru verificarea daca primul număr introdus este mai mic decât cel de-al doilea.

* Mai mare

Cu ajutorului circuitului de scădere si celui care precizează dacă un număr este pozitiv sau nu, utilizatorul va putea primi o confirmare sau o infirmare pentru verificarea daca primul număr introdus este mai mare decât cel de-al doilea.

* Salt condiționat

Saltul implementat este BLE ce compară două numere, iar în cazul în care primul este mai mic decât al doilea, se executa saltul modificându-se contorul programului.

* Test număr pozitiv sau negativ

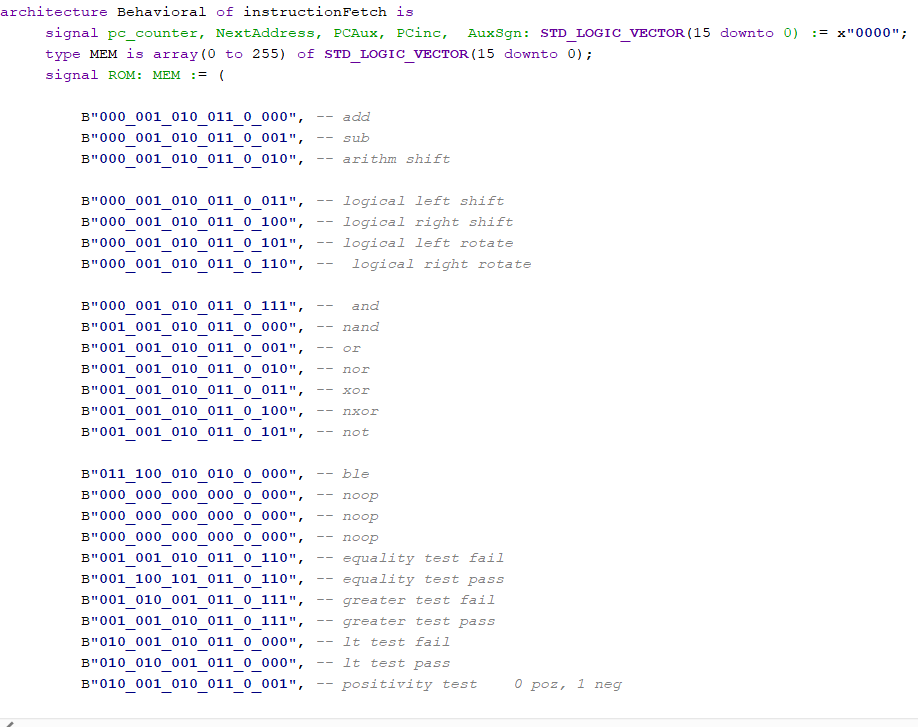
În cazul acestei operații, doar primul număr introdus de utilizator va fi luat în considerare. Folosind un multiplexor ce va avea pe selecție primul bit al numărului, se va afla dacă numărul este pozitiv sau negativ.

# Implementare

## Instruction fetch

Instruction fetch este prima componenta a mips-ului. Aici, în memoria de tip ROM denumită specific ROM sunt stocate instrucțiunile care se vor executa folosind plăcuța. În plus, tot în cadrul acestei părți se incrementează și counter-ul “pc\_counter” la fiecare semnal de tact. Un alt rol este de a implementa mux-urile pentru instrucțiunile de tip BRANCH și JUMP pe baza semnalelor de tip selecție, și anume PCsrc și Jump.

O imagine care conține text

Descriere generată automat

## O imagine care conține text Descriere generată automatInstruction decode

În aceasta componentă sunt stocate registrele care păstrează la fiecare pas valorile asupra cărora se vor efectua operațiile implementate. Avem 8 registre generale care sunt inițializate cu valori aleatorii pentru a putea detecta greșelile de implementare care s-ar putea strecura și unul acumulator în care se va stoca rezultatul operației.

## Instruction execute



O imagine care conține text

Descriere generată automatRolul acestei componente, după cum îi spune și numele, este de a executa operațiile implementate. Sunt mapate toate instrucțiunile implementate pentru a putea fi folosite în mod structural în cadrul proiectului: cele aritmetice, logice și, respectiv, de testare. După cum se vede și în poză, sunt doar 18 operații mapate, celelalte două sunt operații BRANCH și JUMP în cazul cărora nu trebuie implementate suplimentar, întrucât au fost deja codificate in componenta Instruction decode, sub formă de multiplexoare.

O imagine care conține masă

Descriere generată automat Selecția operației care se execută se face în funcție de semnalele ALUCTRL si FUNC care sunt trimise din componenta de control Main Unit. În plus, tot aici se setează și bitul Zero din cadrul operației de tip BRANCH. Acesta se activează dacă este îndeplinită condiția impusă și daca semnalul ALUCTRL ilustrează faptul că operația curentă este de acest tip.

## O imagine care conține text Descriere generată automatMemory

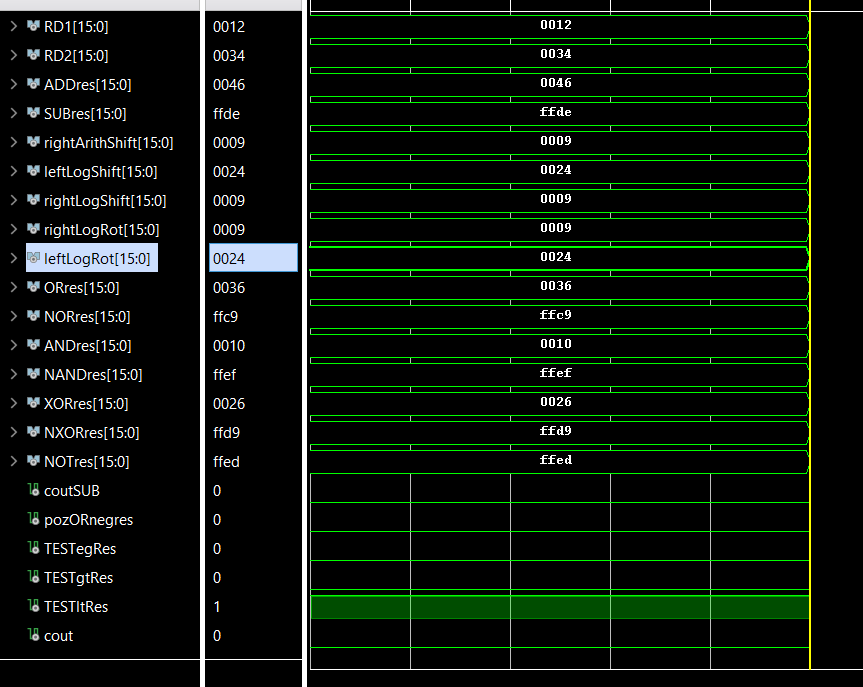
La fel ca celelalte componente, și aceasta are un nume sugestiv. Aici este implementată memoria RAM denumita specific, RAM, care stochează toate numerele pentru operații. Astfel, tipul de adresare în cadrul acestui proiect este indirectă. Pe ieșirile acestei componente sunt informații din memoria RAM, de la adresele cerute de instrucțiunea curentă.

## Test env

O imagine care conține text

Descriere generată automatAceastă componentă reprezintă partea principală a proiectului mips. Aici sunt apelate toate componentele implementate și descrise mai sus. În plus, tot aici se setează si adresa pentru instrucțiunea de tip JUMP și semnalul de selecție pentru operațiile de tip Branch. Pentru a putea fi folosită plăcuța BASYS 3 cu ușurință, a fost implementat un debouncer pentru butoane și afișarea pe afișoarele cu 7 segmente integrate.

# Testare și implementare

Următoarele exemple dovedesc funcționalitatea corecta ale operațiilor implementate. RD1 și RD2 reprezintă termenii asupra cărora se aplică operațiile. În cazul operațiilor care au o singură valoare de intrare, se va alege RD1.

# Concluzii

Rolul acestui proiect a fost proiectarea unei unități centrale cu aproximativ 20 de instrucțiuni( aritmetice, logice, de transfer si de salt). În implementare au fost necesare 9 registre, 1 de acumulator și 8 registre generale. Tipul de adresare a trebuit sa fie indirectă, așadar am decis ca numerele pentru operațiile descrise să fie stocate în memoria RAM din

cadrul componentei de Memory ilustrată mai sus în capitolul de implementare. Pentru memoria în care sunt stocate instrucțiunile, a fost necesar ca adresarea să fie de tip stivă. La nivel intern, implementarea operațiilor a fost realizată prin cascadări, întrucât a fost necesar să folosesc porți logice și bistabile.