

**Facultatea de Automatica și Calculatoare**

**MIPS CICLU DE CEAS UNIC**

Student:Ilieș Oana-Elena

Grupa:30215

**Cuprins**

1.Rezumat

2.Introducere

2.1. Obiectivul proiectului

2.2.FPGA

3.Fundamentare teoretică

4.Proiectare și Implementare

5.Concluzii

Bibliografie

**1.Rezumat**

Proiectul prezinta sinteza de circuit a unui unui MIPS (Microprocessor without Interlocked Pipelined Stages) în ciclu unic constă în executarea tuturor fazelor procesorului (preluare instrucțiune, decodare, executare, accesare memorie și scriere înapoi) pentru fiecare instrucțiune în parte, înainte de a trece la următoarea instrucțiune. Aceasta este o abordare simplă și ușor de implementat, dar nu beneficiază de avantajele procesării în paralel, ceea ce poate reduce performanța în cazul unor programe complexe. Pe langa implementarea clasica a unui procesor MIPS s-a adaugat un program pentru media artimetica ponderata.

**2.Introducere**

2.1 Obiectivul proiectului

Obiectivul implementării unui procesor MIPS este acela de a crea o unitate hardware capabilă să execute instrucțiuni conform arhitecturii MIPS (Microprocessor without Interlocked Pipeline Stages). Acest procesor este proiectat pentru a avea o performanță ridicată și este utilizat în multe domenii, inclusiv în dispozitive mobile, routere, camere video și alte echipamente.

Implementarea codului pentru calculul mediei aritmetice ponderate ar adăuga o funcționalitate utilă procesorului MIPS. Acesta ar putea fi implementat în modulul de instrucțiuni al procesorului, astfel încât să poată fi executat în timp real. Pentru a efectua calculele, ar trebui să se aloce un registru pentru a stoca suma ponderată și să se implementeze operațiile de înmulțire și împărțire în instrucțiunile de calcul. În acest fel, procesorul ar putea efectua calculele necesare pentru medie direct în timpul execuției programelor. Implementarea acestui cod ar crește astfel performanța și utilitatea procesorului.

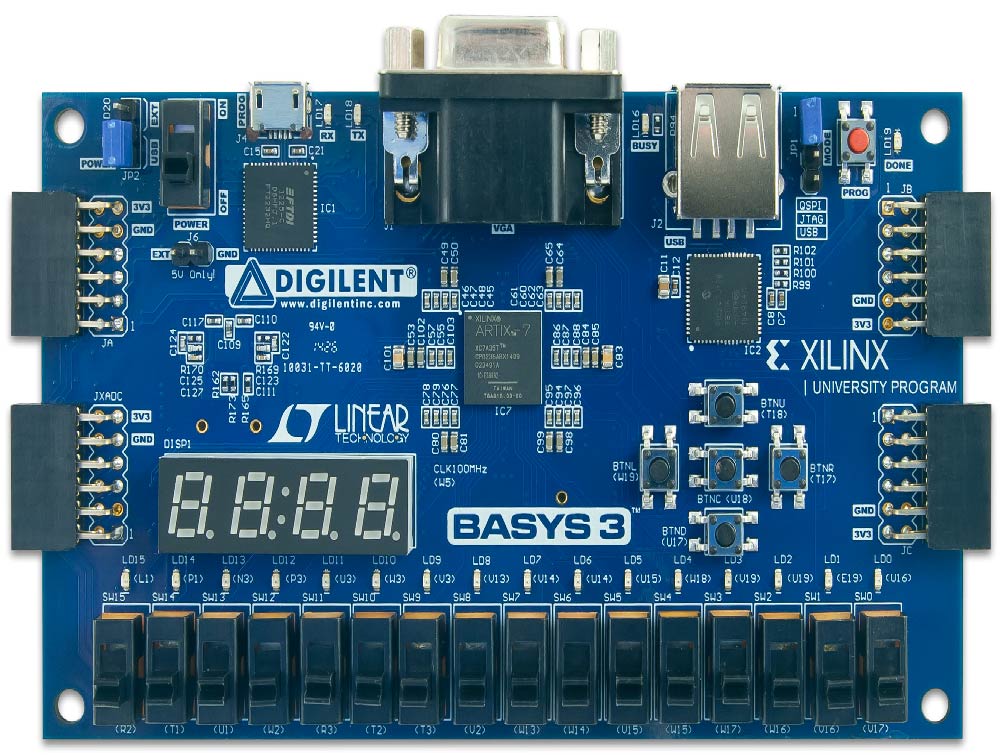
Implementarea unui procesor MIPS cu funcționalitatea de calcul al mediei aritmetice ponderate poate fi îmbunătățită prin stocarea rezultatului în memorie. Aceasta ar putea fi realizată prin introducerea unei instrucțiuni specifice care să permită transferul valorii calculate într-un registru specific către o adresă de memorie specificată. În plus, pentru a putea compara valoarea mediei cu un prag, acesta ar trebui să fie de asemenea stocat în memorie, de preferat la o adresă predefinită.

Proiectul a fost implementat cu ajutorul codului VHDL, care a apărut în anul 1987 și se află în forma actuală încă din anul 1993. Este utilizat în concepția asistată pe calculator (CAD=Computer Aided Design), a circuitelor integrare (de exemplu ASIC) sau, în cazul nostru, pentru configurarea FPGA-urilor.

2.2.FPGA

Progresele în tehnologia digitală au fost uimitoare de-a lungul anilor și au dat naștere designului sistemelor digitale, care continuă să servească drept o mare sursă de ajutor și confort pentru omenire în multe feluri. În zilele noastre, numeroase aplicații în electronică și alte tehnologii folosesc tehnici digitale pentru a efectua operațiuni care cândva au fost efectuate prin metode analogice. Sistemele digitale ofera versatilitate și superioritate față de metodele analogice datorita faptului că nu sunt afectate de fluctuații false ale tensiunii, au o mai mare precizie și acuratețe și pot stoca miliarde de biți de informații într-un spațiu relativ mic. Mai mulți autori au făcut diferite cercetări pentru a minimiza cerințele de putere în fabricarea unui cip integrat și pentru a avea flexibilitate și performanță ridicate. Dezvoltarea plăcii FPGA este o parte a cercetării.

Field-programmable gate array (FPGA) este un circuit integrat proiectat pentru a fi configurat de către un client sau un proiectant după fabricație - de unde și termenul programabil în câmp. Configurația FPGA este în general specificată folosind un limbaj de descriere hardware (HDL), similar cu cel utilizat pentru un circuit integrat specific aplicației (ASIC).



**3.Fundamentare teoretică**

Pentru a implementa un procesor MIPS (Microprocessor without Interlocked Pipeline Stages) cu ciclu unic, sunt necesare următoarele elemente principale:

Unitatea de control (CU) - Aceasta este componenta care se ocupă de controlul fluxului de date și de instrucțiunile care sunt procesate de către procesor. Unitatea de control generează semnalele de control care controlează modulele de execuție din procesor, precum și semnalele de control pentru memoria sistemului.

Unitatea aritmetică și logică (ALU) - Această unitate efectuează operațiile aritmetice și logice necesare pentru procesarea datelor, inclusiv operațiile de adunare, scădere, înmulțire și împărțire.

Register File - Acesta este un set de registre folosite pentru a stoca datele și instrucțiunile care trebuie procesate de către procesor. Registrele sunt utilizate pentru a transfera date între diferitele module ale procesorului.

Memorie - Procesorul trebuie să aibă acces la memoria sistemului pentru a stoca și recupera datele. Acesta poate fi implementat folosind o memorie RAM (Random Access Memory) sau o memorie ROM (Read-Only Memory) în funcție de cerințele sistemului.

Busuri de date și adrese - Acestea sunt căile prin care datele și adresele sunt transmise între modulele procesorului și memoria sistemului.

Instrucțiuni - Procesorul trebuie să fie capabil să execute un set de instrucțiuni MIPS pentru a putea efectua diferite operații aritmetice și logice. Aceste instrucțiuni sunt stocate în memoria sistemului și sunt citite și executate de către procesor.

Pentru implementarea procesorului MIPS ciclu unic, au fost utilizate, printre altele, un display cu 7 segmente și un generator monopuls.

**4.Proiectare și Implementare**

Primul pas în implementarea funcționalității de calcul al mediei aritmetice ponderate este de a determina codul corespunzător acestei operații. Acest cod va fi integrat în modulul de instrucțiuni al procesorului, astfel încât să poată fi executat în timp real de către procesor. În cadrul codului, va fi necesar să se aloce un registru special pentru a stoca suma ponderată, iar apoi să se efectueze operații de împărțire și înmulțire pentru a obține valoarea finală a mediei aritmetice ponderate.

După ce codul a fost determinat, acesta va trebui să fie integrat în modulul de instrucțiuni al procesorului, care va avea rolul de a prelua și executa aceste instrucțiuni. Pentru aceasta, vor fi necesare modificări ale design-ului procesorului, pentru a putea integra noile funcționalități.

Pe lângă acest pas, implementarea procesorului MIPS va necesita și utilizarea altor componente hardware și software, cum ar fi un memory controller, o unitate de control, o unitate de decodare, un interfață de comunicare cu alte dispozitive, precum și alte componente specifice. De asemenea, pentru testarea procesorului vor fi necesare instrumente speciale, precum un simulator de procesor sau un debuger.

**Pas1:Implementare cod in C**

**float calc\_media\_pond(float val[], float pond[], int n, int prag) {**

**float suma\_pond = 0, suma\_val\_pond, medie = 0;**

**for (int i = 0; i < n; i++) {**

**suma\_val\_pond += val[i] \* pond[i];**

**suma\_pond += pond[i];**

**}**

**medie=suma\_val\_pond / suma\_pond;**

**if(medie<prag)**

**return 0;**

**else**

**return 1;**

**}**

**int main() {**

**float val[] = {2, 2, 2, 2}; float pond[] = {1, 1, 1, 1};**

**int n = sizeof(val) / sizeof(val[0]);**

**float media\_pond = calc\_media\_pond(val, pond, n);**

**printf("Media aritmetica ponderata este: %f", media\_pond);}**

**Pas2.:Implementare cod masina:**

1. **addi $0, $0, 0 –init reg in care se stocheaza sum ponderata cu 0**
2. **addi $1, $0, 0 –init reg in care se stocheaza suma de ponderati cu 0**
3. **addi $2, $0,5-nr de elemente din vectori**
4. **addi$3, $0, 9-pragul de comparatie pentru rezultat**
5. **loop:**
6. **lw $4, offset($0)-adresa unde se gaseste vectorul**
7. **lw $5, offset($1)-adresa unde se gaseste vectorul de ponderi**
8. **mul $4, $4, $5-inmultesc elementele vectorului**
9. **add $0, $0, $4-pun suma ponderata in $0**
10. **add $0, $0, $5-pun suma de ponderi**
11. **addi $0, $0, 1-inc vector**
12. **addi $1, $0, 1-inc vector ponderi**
13. **beq $2, $1, 15**
14. **j 5**
15. **end:**
16. **srl $0, $0, 4-impartire**
17. **sw $0, ($0)-aici se stochea media**
18. **verificare prag:**

**19.bgt $3, $0, 21**

**20.addi $0, $0, 8-add 8 daca ii mai mare dacat prag**

**21.mai mare:**

**22.addi $0, $0, 9-add 8 daca ii mai mare dacat prag**

**B"011\_000\_000\_0000000", -- X"2080" -- ADDI $1, $0, 0 --1**

**B"011\_000\_001\_0000000",**

**B"011\_000\_010\_0000101",**

**B"011\_000\_011\_0001010",**

**B"100\_001\_100\_0000000",**

**B"100\_010\_101\_0000000",**

**B"000\_100\_101\_100\_0\_011",**

**B"000\_000\_100\_000\_0\_001",**

**B"011\_000\_000\_0000001",**

**B"011\_000\_001\_0000001",**

**B"001\_001\_010\_0001111",**

**B"111\_0000000000101",**

**B"000\_000\_111\_000\_1\_100",**

**B"110\_000\_000\_0000000",**

**B"010\_000\_011\_0000111",**

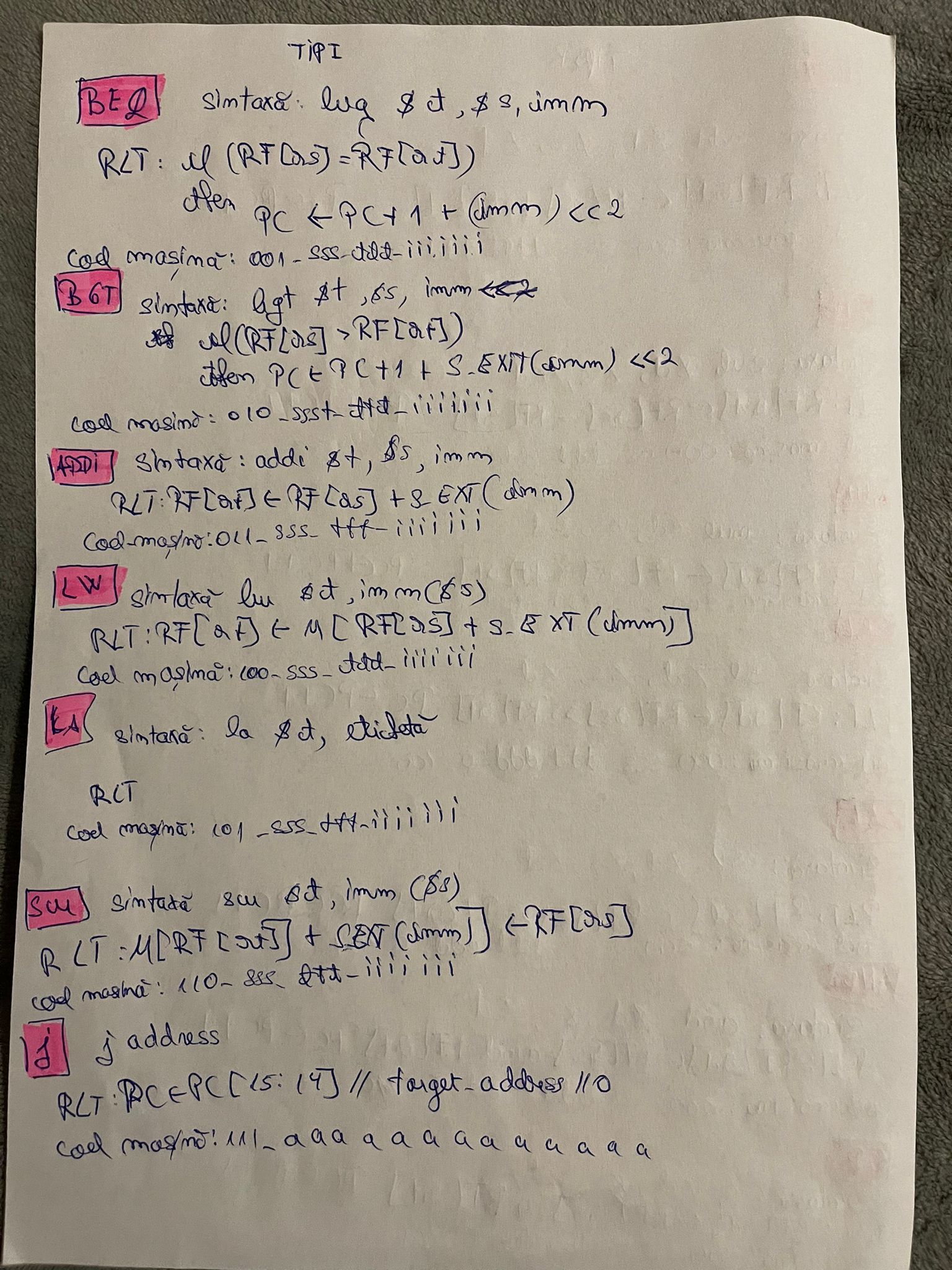
**B"001\_000\_000\_0000000",**

**B"111\_0000000010011",**

**B"001\_000\_000\_0000001",**

**Pas3.Identificare instruciuni necesare si codificarea lor**

Penrtu acest pas am codificat intructiunile de tip R, I si J sub urmatorul format:



Pe langa sintaxa instruciunii, am optat si pentru exemplificarea RTL plus codul masina specific

Pas4: Crearea tabelului de control specific in urma intructiunilor

Cod VHDL

1. **Intruction Fetch**

*signal clk: in std\_logic;*

*signal jump: in std\_logic;*

*signal jump\_adress: in std\_logic\_vector(15 downto 0);*

*signal branch\_adress:in std\_logic\_vector(15 downto 0);*

*signal PC\_SRC:in std\_logic\_vector(1 downto 0);*

*signal rst:in std\_logic;*

*signal en:in std\_logic;*

*signal PC\_NEXT:out std\_logic\_vector(15 downto 0);*

*signal instruction: out std\_logic\_vector(15 downto 0)*

În arhitectura MIPS Single Cycle, procesul de fetch (preluare) a instrucțiunilor este realizat într-un singur ciclu de ceas. Acest proces constă în următoarele etape:

Citirea adresei instrucțiunii: Adresa instrucțiunii următoare de executat este citită din registrul PC (Program Counter).

Actualizarea PC-ului: Adresa următoarei instrucțiuni este calculată prin adăugarea a la valoarea curentă a PC-ului și stocată din nou în PC pentru a pregăti fetch-ul următoarei instrucțiuni.

Procesul de fetch se repetă în continuu, procesorul preluând și executând instrucțiuni în mod secvențial. Odată ce instrucțiunea este preluată în registrul de instrucțiune, următoarea etapă este decodificarea și executarea instrucțiunii respective.

Aici se afla si codul binar reprezentativ pentru instructiunea nou adaugata

1. **Intruction Decode**

*signal RegWrite:in std\_logic;*

*signal Instr:in std\_logic\_vector(15 downto 0 );*

*signal RegDst :in std\_logic;*

*signal ExtOp:in std\_logic;*

*signal rd1:out std\_logic\_vector(15 downto 0);*

*signal rd2:out std\_logic\_vector(15 downto 0);*

*signal wd:in std\_logic\_vector(15 downto 0);*

*signal ExtImm:out std\_logic\_vector(15 downto 0);*

*signal func:out std\_logic\_vector(2 downto 0);*

*signal sa :out std\_logic;*

*signal clk:in std\_logic;*

*signal en:in std\_logic);*

În arhitectura MIPS, după preluarea (fetch) instrucțiunii din memoria de program, urmează etapa de decodificare (instruction decode) în care procesorul extrage informații importante despre instrucțiunea preluată. Aceste informații includ:

Tipul instrucțiunii: Instrucțiunile MIPS sunt împărțite în trei categorii principale: instrucțiuni de tip R (registru), instrucțiuni de tip I (imediat) și instrucțiuni de tip J (salt). Procesorul determină tipul instrucțiunii preluate pe baza primelor câtorva biți ai codului op.

Registre implicați: Majoritatea instrucțiunilor MIPS utilizează registre pentru a stoca sau accesa date. Procesorul identifică registrele utilizate în instrucțiune, care pot fi registre de sursă sau registre de destinație.

Operanzii instrucțiunii: Instrucțiunile MIPS implică adesea doi sau mai mulți operanzi (operandi). Procesorul extrage informațiile referitoare la operanzi din interiorul instrucțiunii.

Adresa de salt: Instrucțiunile de tip J permit procesorului să efectueze salturi la alte adrese de program. Adresa de salt este e

1. **Intruction Execute**

*signal ALUOp : in STD\_LOGIC\_VECTOR (2 downto 0);*

*signal ALUSrc : in STD\_LOGIC;*

*signal ALURes : out STD\_LOGIC\_VECTOR (15 downto 0);*

*signal PC: in std\_logic\_vector(15 downto 0);*

*signal rd1 : in STD\_LOGIC\_VECTOR (15 downto 0);*

*signal rd2 : in STD\_LOGIC\_VECTOR (15 downto 0);*

*signal Ext\_imm : in STD\_LOGIC\_VECTOR (15 downto 0);*

*signal sa : in STD\_LOGIC;*

*signal func : in STD\_LOGIC\_VECTOR (2 downto 0);*

*signal branchAddress:out std\_logic\_vector(15 downto 0)*

În arhitectura MIPS, etapa de execuție (instruction execute) urmează după etapa de decodificare. În această etapă, procesorul efectuează operația specifică instrucțiunii, în funcție de tipul și operandii instrucțiunii.

În funcție de tipul instrucțiunii, operațiile efectuate în etapa de execuție pot fi foarte variate. Iată câteva exemple:

Instrucțiunile aritmetice și logice (de tip R și I) efectuează operații matematice, precum adunarea, scăderea, înmulțirea, împărțirea, șiftarea sau operații logice și bit a bit. Aceste operații sunt realizate pe baza valorilor din registrele implicate în instrucțiune.

Instrucțiunile de acces la memorie (de tip I) efectuează citirea sau scrierea datelor în memoria principală. Adresele de memorie sunt calculate în funcție de valorile din registrele implicate în instrucțiune.

Instrucțiunile de salt (de tip J și I) pot schimba adresa următoarei instrucțiuni ce urmează să fie executate. Adresa de salt este calculată în funcție de valorile din registrele implicate sau poate fi specificată în mod explicit în instrucțiune.

1. **Main Control(UC)**

*Instruction: in std\_logic\_vector(15 downto 0);*

*RegDst : out STD\_LOGIC;*

*ExtOp : out STD\_LOGIC;*

*ALUSrc : out STD\_LOGIC;*

*Branch : out STD\_LOGIC\_VECTOR(1 downto 0);*

*Jump : out STD\_LOGIC;*

*ALUOp : out STD\_LOGIC\_VECTOR (2 downto 0);*

*MemWrite : out STD\_LOGIC;*

*MemToReg : out STD\_LOGIC;*

*RegWrite : out STD\_LOGIC*

UC joacă un rol crucial în funcționarea procesorului MIPS, permițându-i să execute instrucțiunile într-un mod eficient și sincronizat. Prin intermediul semnalelor de control pe care le generează, UC asigură că instrucțiunile sunt preluate, decodate și executate corect, iar procesorul funcționează într-un mod optim.

Generarea semnalelor de control: UC generează semnalele de control necesare pentru a dirija circuitele procesorului în etapele de fetch, decode și execute, în funcție de tipul instrucțiunii și operația care trebuie efectuată.

Actualizarea registrului PC: UC este responsabilă pentru actualizarea registrului PC (Program Counter) cu adresa instrucțiunii următoare care trebuie executată. Acest lucru se face în funcție de tipul instrucțiunii și adresa de salt (dacă există).

Gestionarea ciclurilor de ceas: UC dirijează ciclurile de ceas ale procesorului, asigurându-se că toate blocurile de circuite funcționează în sincron.

**5.Memorie**

*signal clk:in std\_logic;*

*signal MemWrite: in std\_logic;*

*signal en: in std\_logic;*

*signal rd2:in std\_logic\_vector(15 downto 0);*

*signal ALUResI:in std\_logic\_vector(15 downto 0);*

*signal MemData: out std\_logic\_vector(15 downto 0);*

*signal ALUResO: out std\_logic\_vector(15 downto 0)*

Sursa de memorie joacă un rol important în procesul de execuție a instrucțiunilor, deoarece majoritatea instrucțiunilor MIPS implică citirea sau scrierea datelor din memoria principală. Prin intermediul accesului rapid la memoria cache și a implementării unor tehnici de optimizare, cum ar fi prefetching-ul de date, sursa de memorie ajută la minimizarea timpului de execuție al programelor, permițând procesorului să efectueze operațiile necesare cât mai rapid posibil.

**5.Concluzii**

În concluzie, implementarea unui procesor MIPS single-cycle pe 16 biți și documentarea sa necesită o abordare sistematică, o cunoaștere profundă a arhitecturii MIPS și o planificare atentă a tuturor componentelor necesare. Documentația ar trebui să fie completă și să ofere o explicație detaliată a procesului de implementare și a interconexiunilor dintre blocurile de circuite.

**6.Bibliografie**

1. OpenCores: <https://opencores.org/>
2. VHDL World: <https://www.vhdlworld.com/>
3. FPGA4Student: <https://www.fpga4student.com/>
4. Fpga4fun: <https://www.fpga4fun.com/>
5. Site-ul oficial MIPS: <http://www.mips.com/products/architectures/mips32-16-bit/>
6. Mips-tutorial.com: <http://www.mips-tutorial.com/mips32-instruction-set/>
7. Universitatea din New South Wales: <https://www.cse.unsw.edu.au/~cs1521/20T2/>
8. Digital Design and Computer Architecture (2nd Edition): <https://www.elsevier.com/books/digital-design-and-computer-architecture/harris/978-0-12-394424-5>