SIMULATOR DE UCP IN ARHITECTURA PIPELINE

CIUPE DAVID ROBERT
UNIVERSITATEA TEHNICA CLUJ-NAPOCA
2022-2023
GRUPA: 30231/1

Simulator de UCP in arhitectura pipeline

Cuprins

1.Introducere

Scopul acestui proiect este de a dezvolta un simulator de procesor pe 32 de biti in arhitectura pipeline. Acest simulator poate fi folosit pentru a vizualiza cu usurinta instructiunile si calea de date din interiorul procesorului la un moment de timp. Voi realiza o aplicatie cu o interfata grafica pentru a simula functionarea procesorului pipeline.

2.Studiu bibliografic

Microprocessor without Interlocked Pipeline Stage (MIPS) este o arhitectura de procesor care utilizeaza setul de instructiuni RISC simplificata si foarte scalabila dezvoltata de MIPS Computer Systems in 1985. . Procesoarele cu aceasta arhitectura sunt utilizate in: sisteme inglobate, smartphone, tablete, routere, statii de lucru, console de jocuri, autovehicule (Tesla Model S).

Procesoarele MIPS pipeline sunt o imbunatatire a procesoarelor MIPS ciclu unic. Acestea au o linie de instructiuni in 5 etape pentru a executa mai multe instructiuni in acelasi timp. Pipeline-urile permit exploatarea paralelismului la nivel de instructiuni, efectuand in paralel, in etaje diferite, operatii pentru instructiuni diferite.
"Pipelining" nu imbunatateste durata de executie a instructiunilor individuale dar creste productivitatea.

MIPS are 32 de registre fiecare putand inmagazina un numar de 32 de biti(un cuvant).

Exista trei tipuri de formate de instructiuni MIPS: R, I si J. Fiecare instructiune incepe cu un cod optional de 6 biti:

Formatul R(registru): corespunde instructiunilor aritmetice si logice avand ca operanzi trei registre.

- Op : codul instructiunii(opcode), 6 biti
- Rs: primul operand sursa, 5 biti
- Rt: al doilea operand sursa 5 biti
- Rd: operandul destinatie, 5 biti
- Sa, shift amount, 5 biti
- Funct: functia operatiei(selecteaza varianta specifica de operatie pentru un anumit opcode), 6 biti

Formatul I(instructiune): dispunde doar de patru campuri, doua registre si o valoare imediata de 16 bit. Este folosit de doua categorii de instructiuni: instructiunile load-store(de access la memorie) sii instructiunile de salt conditionat.

- Op: opcode, 6 biti
- Rs: primul operand sau registru de baza, 5 biti
- Rt: al doilea operand sau registru destinatie, 5 biti
- Adresa: adresa de memories au de salt, 16 biti.

Formatul J(jump): corespunde instructiunilor de salt neconditionat(jump) si contine doar doua campuri.

- Op: opcode, 6 biti
- Adresa: adresa care este adaugata la PC + 4 pentru a forma adresa de salt neconditionat, 26 biti.

Caile de date MIPS sunt sectionate in 5 etaje: IF, ID, EX, MEM, WB. Se introduce register intre etaje care transfera valori de date si semnale de control de la etaj la etaj.

IF – Aducerea instructiunii

Se citeste instructiunea de executat, adresa PC se incrementeaza cu 4 si se scrie in PC adresa PC incrementata sau adresa de ramificare.

ID – Decodificarea instructiunii si citirea operanzilor

Se decodifica instructiunea adusa
Se citesc operanzii din Blocul de Registre.
Campul imediat se extinde cu semn sau cu zero.
Se transmit adresele care pot fi folosite ca adrese de scriere in Blocul de Registre.
PC incrementat se transmite la etajul urmator.

EX – Etajul de executie

Se calculeaza adresa efectiva pentru instructiunile care acceseaza memoria.
Se transmite data de stocat pentru instructiunea SW
Se transmit instructiuni de tip R, I.
Se transmit instructiuni de ramificare, test de egalitate in ALU.
Adresa de ramificare se calculeaza in sumatorul suplimentar.
Se transmite mai departe adresa potentiala de scriere in blocul de registre

MEM – Memoria

Load (LW) – se citeste din memoria de date, comform adresei calculate in etajul EX
Store (SW) – se scrie in memoria de date comform adresei calculate in etajul EX, valoarea citita din RF[rt] in etajul ID
Se transmite mai departe adresa de scriere in Blocul de Registre
Se transmit rezultatele operatiilor ALU / adresa de ramificare

WB – Write back

Operatii tip ALU pentru instructiunile tip R si I:

- Rezultatul obtinut in ALU se scrie in Blocul de Registre

LW : Valoarea citita din Memoria de Date se scrie in Blocul de Registre

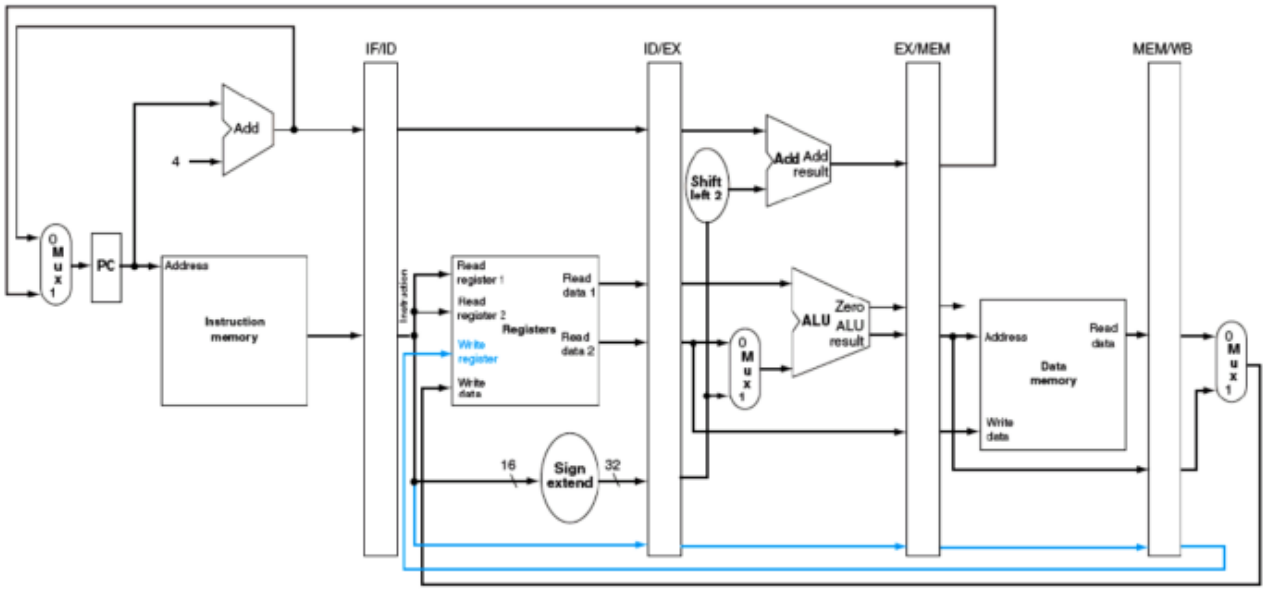

Figura 1 : Mips Pipeline

3. Analiza proiect

Acest proiect are ca scop dezvoltarea unui simulator de procesor pe 32 de biti in arhitectura pipeline. Simulatorul este folosit pentru a vizualiza in detaliu cum functioneaza un mips pipeline si care sunt pasii pe care acest procesor ii urmeaza atunci cand executa anumite instructiuni.

Aplicatia are urmatoarele functionalitati:

- Utilizatorul poate introduce cod in limbaj de asamblare
- Dupa ce a scris codul in limbaj de asamblare va putea vedea instructiunea tradusa in cod masina
- Utilizatorul poate sa vizualizeze la un anumit pas datele din registrele pipeline care transfera date de la un etaj la altul(IF/ID, ID/EX, EX/MEM, MEM/WB): cum ar fi : semnalele de control, calea de date.
- Utilizatorul poate vizualiza datele din Instruction Fetch:
  - PC-ul curent
  - PC-ul urmator
  - Instructiunea curenta in cod masina
- Utilizatorul poate vizualiza datele stocate in toate cele 32 de registre si tot odata operanzii din blocul de registre(si rs, rt, rd) si imediatul extins.
- Se poate vizualiza si adresa efectiva pentru instructiunile care acceseaza memoria.
- Se poate vizualiza rezultatul din ALU (atat operatorii cat si rezultatul)
- Se poate vizualiza adresa de ramificare.
- Memoria poate fi vizualizata. Se poate vizualiza o locatie din memorie aflata la o anumita adresa, sau toate locatiile din memorie deodata.

Aplicatia functioneaza in felul urmator. Dupa ce se scrie codul in limbaj de asamblare, se pot executa instructiunile pas cu pas. Si cu o apasare de buton, simularea trece la pasul urmator iar datele se transfera din registre la urmatorul etaj de registre.

Tot odata aplicatia functioneaza exact ca si un procesor pipeline adevarat, avand aceleasi funtionalitati ca si acesta. Se pot realiza aceleasi operatii ca pe mips iar aplicatia incearca sa simuleze procesorul cat mai bine.

Se poate executa un algoritm predefinit sau se poate scrie un program de catre utilizator. Simularea se poate face pas cu pas sau se poate executa automat singura.

Dupa cum spuneam si mai sus, exista trei tipuri de instructiuni: de tip R, tip I si tip J. Instructiunile de tip R corespund instructiunilor aritmetice si logice. Cu acest tip de instructiune se vor putea realiza tot felul de operatii aritmetice cum ar fi: adunare, scadere. Operatii logice : AND, XOR, OR. Si alte operatii cum ar fi shiftarea.

Instructiunile de tip I folosesc o valoare imediata si se pot realiza urmatoarele operatii cu ele: adunare/scadere cu imediat. Operatii logice cu un imediat; AND, XOR, OR, etc. Se pot realiza load si store si salt conditionat.

Instructiunile de tip J sunt folosite pentru salturi neconditionate.

4. Design

Acestea sunt codificarile pentru instructiuni, si semnalele de control pentru fiecare tip de instructiune.

Instrucțiune	**Opcode**	**RegDst**	**ExtOp**	**ALUSrc**	**Branch**	**Jump**	**JmpR** (opțional)	**MemWrite**	**MemtoReg**	**RegWrite**	**ALUOp(2:0)**	**FUNC instr**
ADD	000000	1	X	0	0	0		0	0	1	000	000000
SUB	000000	1	X	0	0	0		0	0	1	001	000001
AND	000000	1	X	0	0	0		0	0	1	010	000010
OR	000000	1	X	0	0	0		0	0	1	011	000011
XOR	000000	1	X	0	0	0		0	0	1	100	000100
SLT	000000	1	X	0	0	0		0	0	1	101	000101
JR	000000	1	X	0	0	1	1	0	0	1	110	000110
ADDI	001000	0	1	1	0	0		0	0	1	000	X
LW	100011	0	1	1	0	0		0	1	0	X	X
SW	101011	0	1	1	0	0		1	0	0	X	X
BEQ	000100	X	1	0	1	0		0	0	0	001	X
BNE	000101	X	1	0	1	0		0	0	0	001	X
ANDI	001100	0	0	1	0	0		0	0	1	010	X
ORI	001110	0	0	1	0	0		0	0	1	011	X
XORI	000110	0	0	1	0	0		0	0	1	100	X
JUMP	000010	X	X	X	X	1		X	X	X	X	X

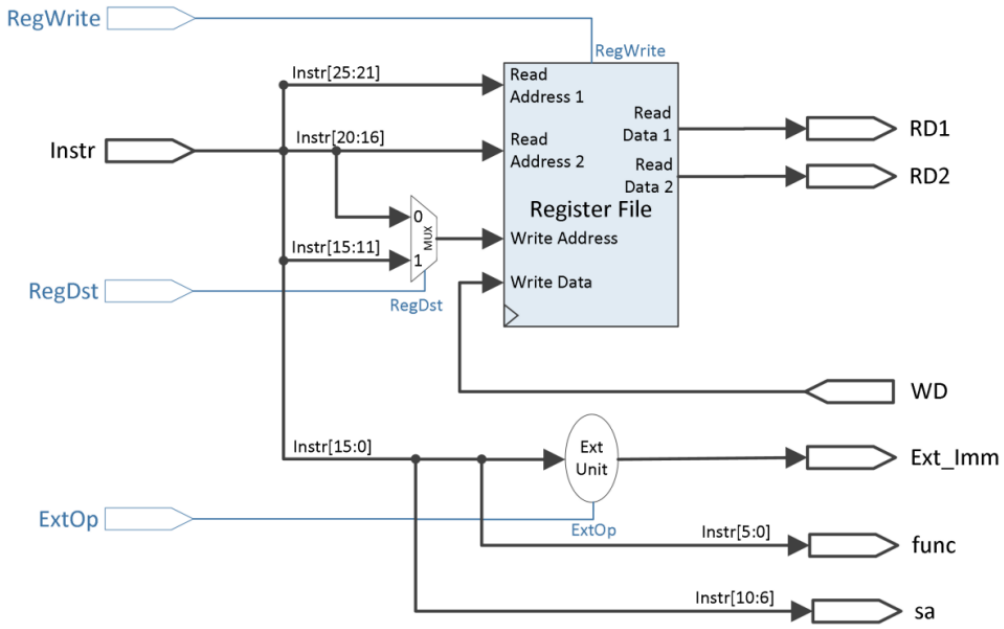

Figura 2 : Bloc registre

5. Implementare

Pentru implementarea aplicatiei am folosit limbajul de programare Java.

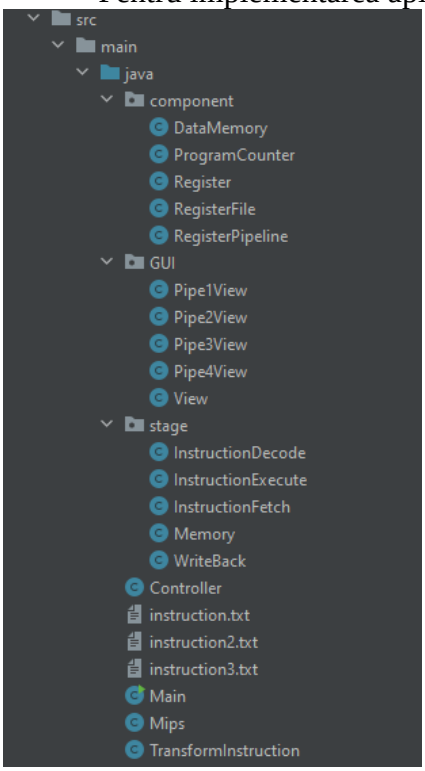

Figura 3 : Pachetele si clasele proiectului

Pentru implementarea proiectului am folosit principiile OOP. Am realizat o clasa pentru fiecare componenta mips.

Avem clasa Mips, aici sunt create si instantiate toate componentele mips si sunt legate intre ele.

In pachetul component avem clasa Register. Aceasta este registrul de baza care poate stoca o valoare. Clasa RegisterFile contine o lista cu mai multi registrii, iar in acesti registrii se vor stoca datele din blocul de registrii. RegisterPipeline este o clasa care extinde RegisterFile, iar aceasta este folosita pentru a stoca registrii pipeline si datele din acestia de la fiecare stadiu (if/id, id/ex, ex/mem, mem/wb). Avem si clasa ProgramCounter care extinde clasa Register. Acesta simuleaza program counter-ul si este incrementat la fiecare clock.

In pachetul stage, avem fiecare stadiu pipeline. Pentru fiecare stadiu am implementat metoda run() in care se executa ce este de executat in acel stagiu. In InstructionFetch se citeste din memorie instructiunea si se incrementeaza PC. In InstructionDecode se decodifica instructiunea si se transmit semnalele de control mai departe. In InstructionExecute se executa efectiv instructiunea si se transmit mai departe semnalele. In Memory se scrie sau citeste din memoria programului in functie de semnalele de control. In WriteBack se scrie in blocul de registre in functie de semnalele de control.

Fiecare clasa are metoda clock(), care simuleaza un tic de ceas al procesorului.

In clasa TransformInstruction am transformat instructiunea din limbaj de asamblare in cod mips.

6. Testare

Pentru testare avem o interfata grafica in care putem scrie instructiuni in limbaj de asamblare. Si se pot vizualiza blocul de registrii, memoria, PC-ul si fiecare registru pipeline cu semnalele de control.

7. Concluzii

In concluzie am realizat un program care simuleaza un procesor MIPS pipeline pe 32 de biti cu o interfata grafica in care se poate vedea pas cu pas cum se modifica datele si semnalele din fiecare stagiu pipeline.

Bibliografie:

[1] Florin Oniga, “De la bit la procesor. Introducere în arhitectura calculatoarelor”, Editura U.T. Press, Cluj-Napoca, 2019, ISBN 978-606-737-366-0

[2] https://ro.wikipedia.org/wiki/Arhitectur%C4%83_MIPS

[3] https://users.utcluj.ro/~onigaf/files/AC.html