

**Facultatea de Automatica și Calculatoare**

**MIPS CICLU DE CEAS PIPELINE**

Student:Ilieș Oana-Elena

Grupa:30222

**Cuprins**

1.Rezumat

2.Introducere

2.1. Obiectivul proiectului

2.2.FPGA

3.Fundamentare teoretică

4.Proiectare și Implementare

5.Concluzii

Bibliografie

**1.Rezumat**

Documentația prezintă un studiu detaliat al arhitecturii și funcționării unui procesor MIPS (Microprocessor without Interlocked Pipelined Stages) cu tehnica de pipelining. Pipelining-ul este o tehnică utilizată în proiectarea procesorului pentru a accelera execuția instrucțiunilor prin împărțirea procesului în mai multe etape simultane. Implementarea sa a pornit in urma implementarii unui MIPS în ciclu unic.

**2.Introducere**

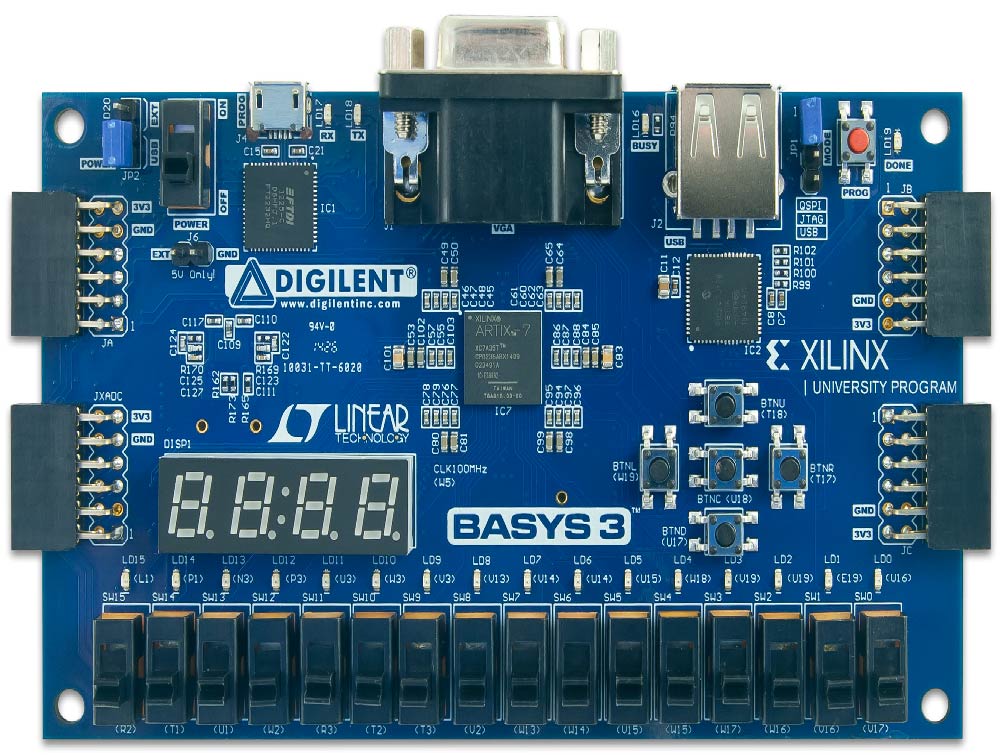
2.1 Obiectivul proiectului

Obiectivul unui procesor MIPS Pipeline este de a îmbunătăți performanța și eficiența procesării instrucțiunilor într-un procesor, prin împărțirea execuției instrucțiunilor în mai multe etape simultane. In urma analizei realizate, implementarea este efectuata sub forma unui program Fibonacci.

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 pipeline 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.

Registrii pentru stocarea stadiilor pipeline-ului:

PC (Program Counter): Registru special care conține adresa instrucțiunii următoare care va fi preluată.

IF/ID (Instruction Fetch/Instruction Decode): Registru de transfer care stochează instrucțiunea preluată din memoria de instrucțiuni și alte informații relevante pentru decodare.

ID/EX (Instruction Decode/Execution): Registru de transfer care conține datele decodate ale instrucțiunii și semnalele de control necesare pentru etapa de execuție.

EX/MEM (Execution/Memory Access): Registru de transfer care stochează rezultatele etapei de execuție și semnalele de control asociate accesului la memorie.

MEM/WB (Memory Access/Write Back): Registru de transfer care conține datele și semnalele de control pentru accesul la memorie și operațiile de scriere înapoi în registrele de destinație.

Hazardul de date este o problemă în cadrul procesorului când există conflicte între accesul la date. Aceste conflicte pot duce la rezultate greșite și pot afecta corectitudinea execuției programului.

Există trei tipuri principale de hazarduri de date: hazardul de date structural, hazardul de date de control și hazardul de date de dependență.

1. Hazardul de date structural:

- Apare atunci când o instrucțiune nu poate fi executată în același ciclu cu o altă instrucțiune din cauza limitărilor structurale ale procesorului.

- Exemple de hazarduri de date structurale includ accesarea simultană a aceluiași modul de memorie, utilizarea simultană a aceluiași port de înregistrare sau a aceluiași modul de execuție a unei operații.

2. Hazardul de date de control:

- Apare atunci când o instrucțiune de branch (săritură) schimbă fluxul de execuție și afectează instrucțiunile ulterioare.

- Exemple de hazarduri de date de control includ deciziile de branch care nu sunt cunoscute înainte de execuție și necesită modificarea fluxului normal de instrucțiuni.

3. Hazardul de date de dependență:

- Apare atunci când o instrucțiune depinde de rezultatele unei alte instrucțiuni care nu s-au finalizat încă.

- Există două tipuri de hazarduri de date de dependență:

a) Hazard de date de dependență RAW (Read-After-Write): O instrucțiune citește o valoare dintr-un registru pe care o instrucțiune anterioară tocmai l-a scris.

b) Hazard de date de dependență WAW (Write-After-Write): Două instrucțiuni încearcă să scrie în același registru în aceeași perioadă de timp.

- Pentru a rezolva hazardul de date de dependență, se pot folosi tehnici precum reordonarea instrucțiunilor, utilizarea registrelor de citire/scrise (register renaming), sau introducerea de nop-uri (instrucțiuni goale) pentru a se asigura că dependențele sunt rezolvate corect.

Pentru a gestiona hazardurile de date într-un procesor MIPS Pipeline, pot fi implementate tehnici precum forwardare (forwarding), nop-uri și interlock-uri (bubble-uri). Aceste tehnici ajută la evitarea și rezolvarea conflictelor de date și asigură execuția corectă a instrucțiunilor în cadrul pipeline-ului.

**4.Proiectare și Implementare**

Cod VHDL

1. **Intruction Fetch**

Modulul IFetch are următoarele porturi:

clk: semnalul de ceas

rst: semnalul de reset

en: semnalul de enable

BranchAddress: vectorul de 16 biți pentru adresa de branch

JumpAddress: vectorul de 16 biți pentru adresa de jump

Jump: semnalul de jump

PCSrc: semnalul pentru selectarea sursei PC-ului

Instruction: semnalul de ieșire pentru instrucțiune (vector de 16 biți)

PCinc: semnalul de ieșire pentru PC-ul incrementat (vector de 16 biți)

Modulul IFetch are un registru PC care reține adresa următoarei instrucțiuni. Pe baza valorii PC-ului, instrucțiunea corespunzătoare este citită dintr-o memorie ROM.

PC-ul este actualizat în fiecare ciclu de ceas pe baza semnalelor de control și instrucțiunilor de branch și jump. Instrucțiunea curentă este transmisă ca ieșire, iar PC-ul incrementat este calculat pentru următorul ciclu de ceas.

In memoria ROM aflata in cadrul acestuia se afla codul fibonacii, implemenat fara hazarduri:

B"001\_000\_001\_0000000", -- X"2080" -- ADDI $1, $0, 0 --0

B"001\_000\_010\_0000001", -- X"2101" -- ADDI $2, $0, 1 --1

B"001\_000\_011\_0000000", -- X"2180" -- ADDI $3, $0, 0 --2

B"001\_000\_100\_0000001", -- X"2201" -- ADDI $4, $0, 1 --3

B"000\_000\_000\_0000000", --4

B"011\_011\_001\_0000000", -- X"6C80" -- SW $1, 0($3) --5

B"011\_100\_010\_0000000", -- X"7100" -- SW $2, 0($4) --6

B"010\_011\_001\_0000000", -- X"4C80" -- LW $1, 0($3) --7

B"010\_100\_010\_0000000", -- X"5100" -- LW $2, 0($4) --8

B"000\_000\_000\_0000000", --9

B"000\_000\_000\_0000000", --10

B"000\_001\_010\_101\_0\_000", -- X"0550" -- ADD $5, $1, $2 --11

B"000\_000\_000\_0000000", --12

B"000\_000\_010\_001\_0\_000", -- X"0110" -- ADD $1, $0, $2 --13

B"000\_000\_101\_010\_0\_000", -- X"02A0" -- ADD $2, $0, $5 --14

B"111\_0000000001011", -- X"E008" -- J 11 --15

B"000\_000\_000\_0000000", --16

1. **Intruction Decode**

Modulul IDecode primește următoarele semnale și porturi:

clk: semnalul de ceas

en: semnalul de enable

Instr: vectorul de 13 biți pentru instrucțiune

WD: vectorul de 16 biți pentru datele scrise în registrul de destinație

WA: vectorul de 3 biți pentru adresa de înregistrare de destinație (registru scris)

RegWrite: semnalul pentru activarea scrierii în registrul de destinație

ExtOp: semnalul pentru activarea operațiilor de extensie (semnul extins și zero extension)

RD1: vectorul de 16 biți pentru datele citite din primul registru (registru de sursă 1)

RD2: vectorul de 16 biți pentru datele citite din al doilea registru (registru de sursă 2)

Ext\_Imm: vectorul de 16 biți pentru constanta sau valoarea extinsă

func: vectorul de 3 biți pentru codul funcție al instrucțiunii

sa: semnalul pentru specificarea câmpului SA al instrucțiunii

Modulul IDecode este responsabil pentru decodificarea instrucțiunilor din pipeline-ul MIPS. Acesta primește instrucțiunea și alte semnale de control și produce semnalele necesare pentru citirea datelor din registrele de sursă, scrierea datelor în registrul de destinație și extensia valorilor.

Principalele funcționalități ale modulului includ:

Citirea datelor din registrele de sursă în baza indicilor specificați în instrucțiune.

Scrierea datelor în registru de destinație (dacă semnalul RegWrite este activat).

Extensia semnului sau extensia cu zero a unei valori în funcție de instrucțiune și semnalul ExtOp.

Generarea semnalelor pentru codul funcție și câmpul SA al instrucțiunii.

Modulul IDecode utilizează un registru de fișiere pentru a stoca valorile registrilor generale din procesorul MIPS.

1. **Main Control**

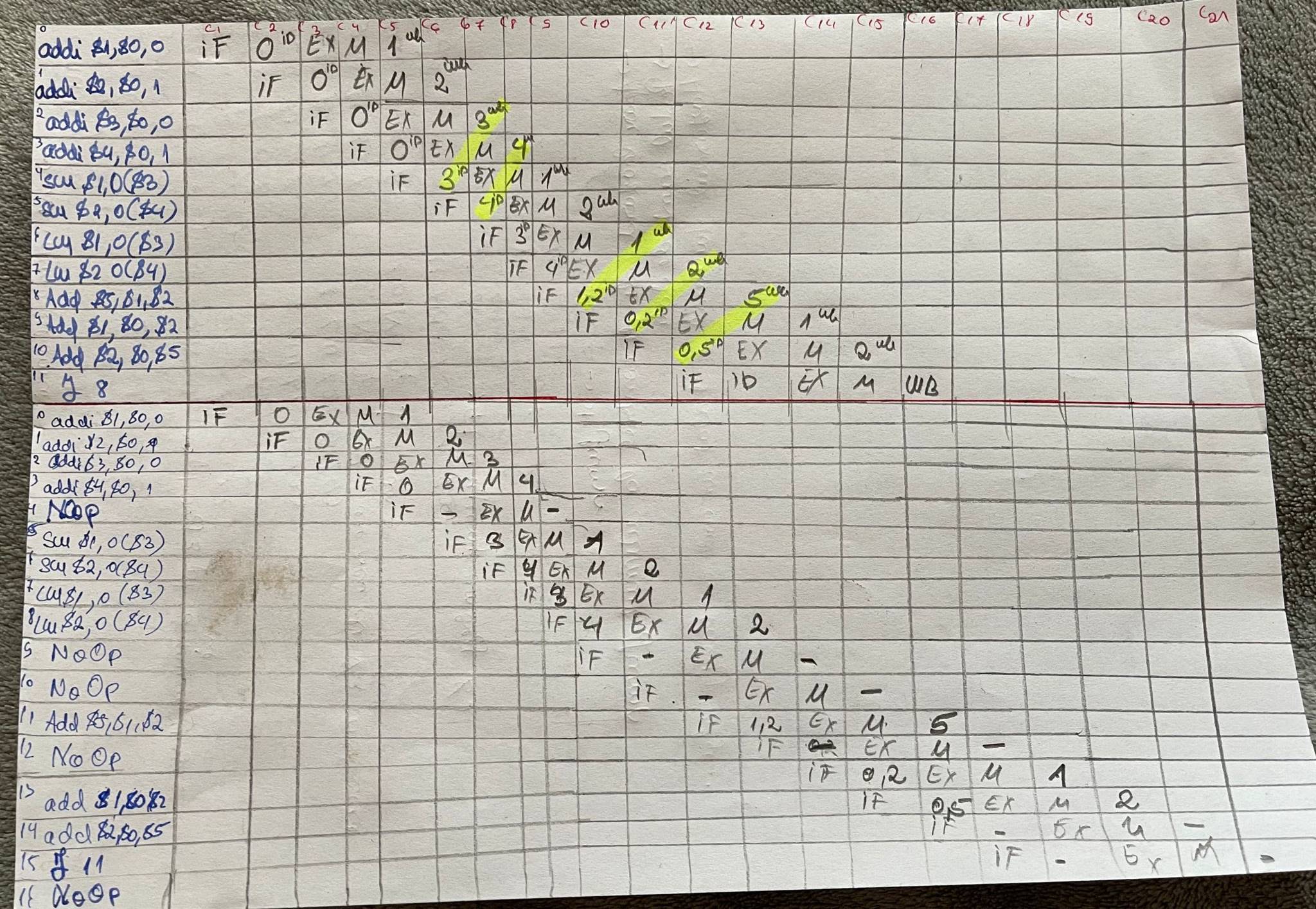
Modulul MainControl este responsabil pentru generarea semnalelor de control în funcție de instrucțiunea primită. Acesta determină tipul instrucțiunii și setează semnalele corespunzătoare pentru celelalte module ale procesorului.

1. **Intruction Execute**

Acest cod reprezintă implementarea în VHDL a unei unități de execuție (Execution Unit) pentru un procesor MIPS Pipeline. Unitatea primește semnale de intrare precum adrese de registri, date, semnale de control și instrucțiuni, și produce rezultatele operațiilor ALU, adresele pentru scrierea înapoi în registre și adresa de săritură. Principalele funcționalități includ selectarea sursei de date pentru ALU, determinarea adresei de scriere în registre, controlul ALU și detectarea valorii zero a rezultatului ALU. Unitatea de execuție este un component important al procesorului MIPS Pipeline.

**5.Memorie**

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.



Apare hazardul de control in cazul ultimului Jump, in cadrul registrelor subliniate apare hazardul de date, iar hazardul structural e tratat in ID prin utilizarea lui falling\_edge(clk).

**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>