Logo

Description automatically generated

**DOCUMENTAȚIE:**

**MIPS 16**

**PIPELINE**

STUDENT: SABĂU OANA-MARIA

GRUPA: 30222

AN UNIVERSITAR: 2022/2023

**Cuprins**

[1. Obiectiv 3](#_Toc135397351)

[2. Prezentare și analiză problemă 3](#_Toc135397352)

[3. Definire instrucțiuni, identificare hazarduri și trasare program 4](#_Toc135397353)

[4. Implementare 8](#_Toc135397354)

[5. Scheme incluse 9](#_Toc135397355)

[6. Concluzii 13](#_Toc135397356)

[7. Resurse necesare 13](#_Toc135397357)

[8. Bibliografie 13](#_Toc135397358)

# **1. Obiectiv**

Obiectivul proiectului constă în realizarea unui microprocesor MIPS pipeline pe 16 biți care creează un model digital al unui procesor care poate executa instrucțiuni MIPS la viteze ridicate, folosind limbajul de descriere hardware VHDL. Acest microprocesor se obține prin extinderea microprocesorului MIPS 16 cu ciclu unic realizat anterior prin adăugarea unor regiștri adiționali (bistabile de tip D) ce sporesc viteza de execuție a programului, după cum se va prezenta mai jos.

# **2. Prezentare și analiză problemă**

MIPS Pipeline este conceput pentru a minimiza dependențele de date între instrucțiuni și a maximiza utilizarea resurselor hardware, permițând astfel un nivel ridicat de performanță și eficiență în procesarea instrucțiunilor. Una dintre caracteristicile cheie ale MIPS Pipeline este divizarea procesului de execuție al instrucțiunilor într-o succesiune de etape independente. Aceasta înseamnă că instrucțiunile sunt împărțite în diferite etape, iar fiecare etapă prelucrează o instrucțiune diferită la un moment dat. Astfel, într-un moment dat, mai multe instrucțiuni se află în diferite etape ale pipeline-ului.

Divizarea instrucțiunilor în etape independente are mai multe avantaje. În primul rând, permite ca mai multe instrucțiuni să fie prelucrate în același timp, ceea ce duce la o creștere a vitezei de execuție a programelor. În loc să aștepte finalizarea completă a unei instrucțiuni înainte de a trece la următoarea, pipeline-ul permite ca instrucțiunile să progreseze simultan prin diferitele etape ale procesului de execuție.

Cu toate acestea, pipeline-ul MIPS nu este lipsit de provocări. Unele instrucțiuni pot avea dependențe de date care nu pot fi evitate, ceea ce duce la încetinirea execuției. Acestea poartă numele de *hazarduri*.

Hazardurile sunt situații în care o instrucțiune următoare nu poate fi executată în următoarea perioadă de ceas. Există 3 tipuri de hazarduri:

* ***Hazard structural (dependență de resurse)***: două instrucțiuni încearcă să folosească simultan o resursă, în scopuri diferite; constrângeri de resurse
* ***Hazard de date (dependenta de date)***: încercare de a folosi date înainte să fie disponibile; pentru o instrucțiune care a ajuns în faza ID, operanzii sunt în curs de prelucrare în alte etaje de pipeline
* ***Hazard de control/comandă(dependenta de condiții, control)***: decizia despre ramificarea unui program nu se știe înainte de evaluarea condițiilor salturilor și calcularea adresei de ramificare pentru PC; trecerea prin pipeline a instrucțiunilor care influențează PC(ramificări, salturi)

Programul ales pentru a fi implementat de microprocesor determină cel mai mare divizor comun – CMMDC (eng. GCD) dintre două elemente dintr-un array a căror poziții au fost date. Programul inițial este scris în limbajul de programare C, apoi este transformat în cod de asamblare, iar apoi în cod mașină (respectând convențiile MIPS 16).

# **3. Definire instrucțiuni, identificare hazarduri și trasare program**

*Programul MIPS asamblare descrie găsirea celui mai mare divizor comun – CMMDC (eng. GCD) între 2 elemente dintr-un array de pe pozițille date:*

**0 LW $1, 1($0)**

**1 LW $2, 5($0)**

**2 BEQ $1, $2, 6**

**3 SLT $3, $1, $2**

**4 BEQ $3, $0, 2**

**5 SUB $2, $2, $1**

**6 J 2**

**7 SUB $1, $1, $2**

**8 J 2**

**9 SW $1, 7($0)**

Programul inițial nu se execută corect din cauza hazardurilor întâmpinate. Acestea sunt detaliate și descrise mai jos:

Întâmpinăm **hazarduri structurale** (de exemplu, instrucțiunea 0 și 2 încearcă să folosească în același timp Register File). Pentru a evita acest fenomen, soluția este de a modifica blocul de regiștri RFdin MIPS cu ciclu unic: scriere în RF pe frontul descrescător (falling\_edge) al perioadei de ceas deoarece citirea din RF este asincronă, se face până la sfârșitul perioadei de ceas.

Cu toate că tratăm hazardul structural, ne confruntăm cu cel de date. Prin folosirea tehnicii de stalling (adăugarea de No Operations care nu modifică programul, doar au ca efect întârzierea unor instrucțiuni), reușim să rezolvăm și **hazardul de date** precum mai jos:

* între instrucțiunile 0 și 2 (registrul $1) – adăugăm 1 No Op
* între instrucțiunile 1 și 2 (registrul $2) – adăugăm 1 No Op
* între instrucțiunile 3 și 4 (registrul $3) – adăugăm 2 No Op

Ca să ne asigurăm că nu există hazard de date la instrucțiunile 5 și 7 (sunt operații de tip R – scădere), adăugam câte 2 No Op după fiecare instrucțiune pentru a fi siguri că am stocat rezultatul în registrul $1, respectiv registrul $2 (regiștrii destinație).

**Hazardul de control** apare în cazul instrucțiunilor branch (BEQ) și jump (J) deoarece se modifică calcularea adresei pentru Program Counter. Avem hazard de control la instrucțiunile:

* instrucțiunea 2: se execută un salt condiționat (beq) la etajul MEM – adăugăm 3 No Op
* instrucțiunea 4: se execută un salt condiționat (beq) la etajul MEM – adăugăm 3 No Op
* instrucțiunea 6: se execută un salt necondiționat (j) la etajul ID – adăugăm 1 No Op
* instrucțiunea 6: se execută un salt necondiționat (j) la etajul ID – adăugăm 1 No Op

Urmărind acestea, ar trebui să obținem un program în cod mașină funcțional.

*Programul soluționat fără hazarduri este descris mai jos:*

**0 LW $1, 1($0)**

**1 LW $2, 5($0)**

**2 NO OP**

**3 NO OP**

**4 BEQ $1, $2, 6**

**5 NO OP**

**6 NO OP**

**7 NO OP**

**8 SLT $3, $1, $2**

**9 NO OP**

**10 NO OP**

**11 BEQ $3, $0, 2**

**12 NO OP**

**13 NO OP**

**14 NO OP**

**15 SUB $2, $2, $1**

**16 NO OP**

**17 NO OP**

**18 J 4**

**19 NO OP**

**20 SUB $1, $1, $2**

**21 NO OP**

**22 NO OP**

**23 J 4**

**25 NO OP**

**25 SW $1, 7($0)**

*Program cod mașină:*

**B"010\_000\_001\_0000001", -- X"4081" -- lw $1, 1($0) --0**

**B"010\_000\_010\_0000101", --X"4105" -- lw $2, 5($0) --1**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --2**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --3**

**B"100\_010\_001\_0010100", --X"8896" -- beq $1, $2, 20 --4**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --5**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --6**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --7**

**B"000\_001\_010\_011\_0\_111", --X"0537" -- slt $3, $1, $2 --8**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --9**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --10**

**B"100\_000\_011\_0001000", --X"8188" -- beq $3, $0, 8 --11**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --12**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --13**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --14**

**B"000\_010\_001\_010\_0\_001", --X"08A1" -- sub $2, $2, $1 --15**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --16**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --17**

**B"111\_0000000000100", --X"E004" -- j 4 --18**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --19**

**B"000\_001\_010\_001\_0\_001", --X"0511" -- sub $1, $1, $2 --20**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --21**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --22**

**B"111\_0000000000100", --X"E004" -- j 4 --23**

**B"000\_000\_000\_000\_0\_000", --X"0000" --NoOp (ADD $0, $0, $0) --24**

**B"011\_000\_001\_0000111", --X"6087" --sw $1, 7($0) --25**

# **4. Implementare**

Ciclul de execuție a unei instrucțiuni MIPS are următoarele etape / faze:

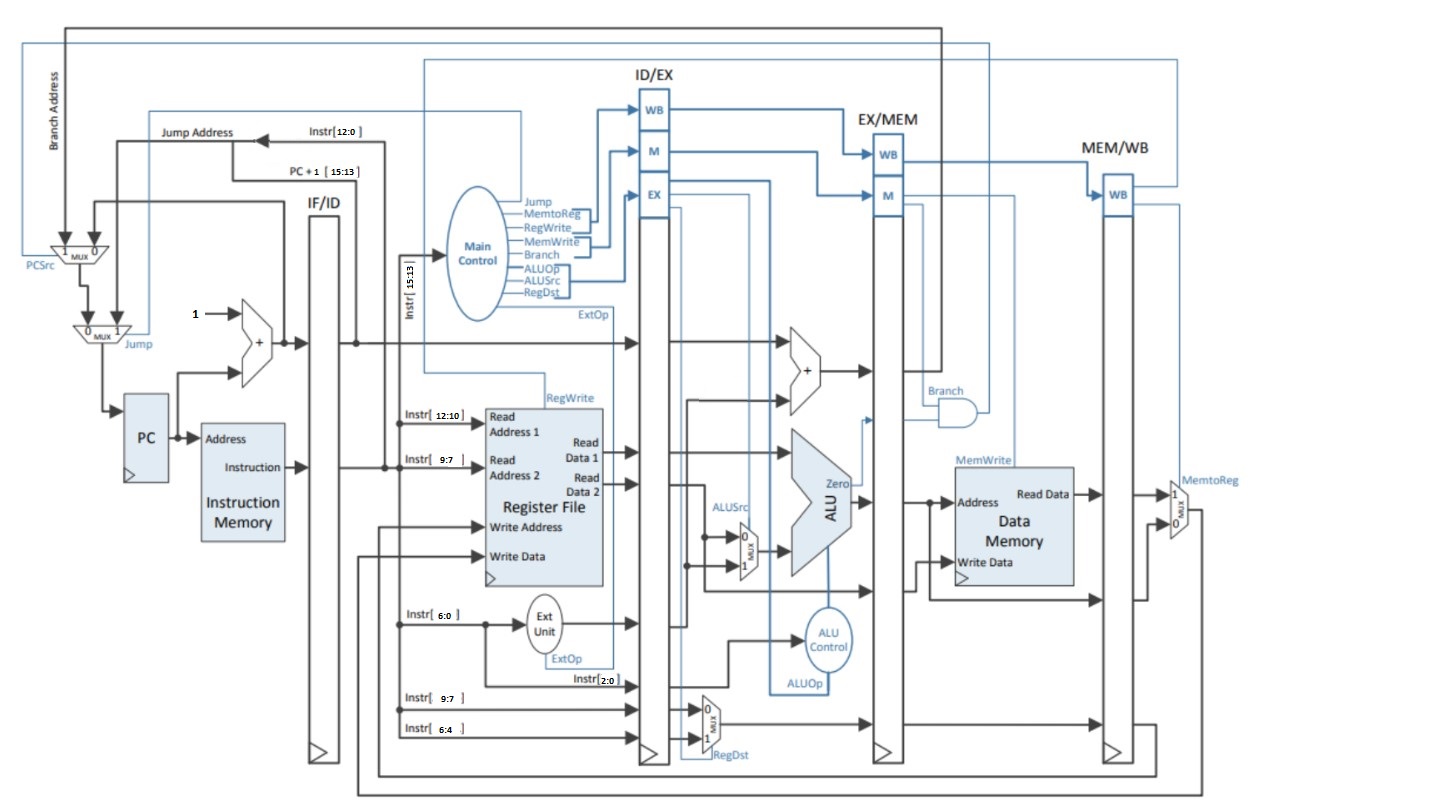
* ***IF – Extragerea Instrucțiunii / Instruction Fetch*** - procesorul extrage instrucțiunea din memoria programului. Adresa instrucțiunii este stocată în registrul Program Counter (PC), iar PC-ul este incrementat pentru a indica adresa următoarei instrucțiuni din program.
* ***ID/OF – Decodificarea Instrucțiunii / Extragerea Operanzilor*** - acum, procesorul decodează instrucțiunea extrasă în faza IF și extrage orice operanzi necesari pentru execuție. În funcție de tipul instrucțiunii, se poate lua decizia de a transfera datele din registrele de intrare ale procesorului în unitatea de execuție (EX), sau de a citi datele din memoria sistemului.
* ***Instruction Decode / Operand Fetch*** - în această fază, procesorul efectuează operația specifică instrucțiunii pe operanzii decodați în faza ID/OF
* ***EX – Execuție / Execute*** - procesorul accesează memoria sistemului pentru a citi sau scrie datele. Această fază este necesară doar pentru instrucțiunile care accesează memoria.
* ***MEM – Memorie / Memory*** - procesorul accesează memoria sistemului pentru a citi sau scrie datele. Precum faza EX, este necesară doar pentru instrucțiunile care accesează memoria.
* ***WB – Scriere Rezultat / Write Back*** - procesorul scrie rezultatul operației înapoi în registrele de ieșire ale procesorului. Acest lucru este necesar pentru a păstra consistența datelor și pentru a permite instrucțiunilor ulterioare să utilizeze valorile corecte.

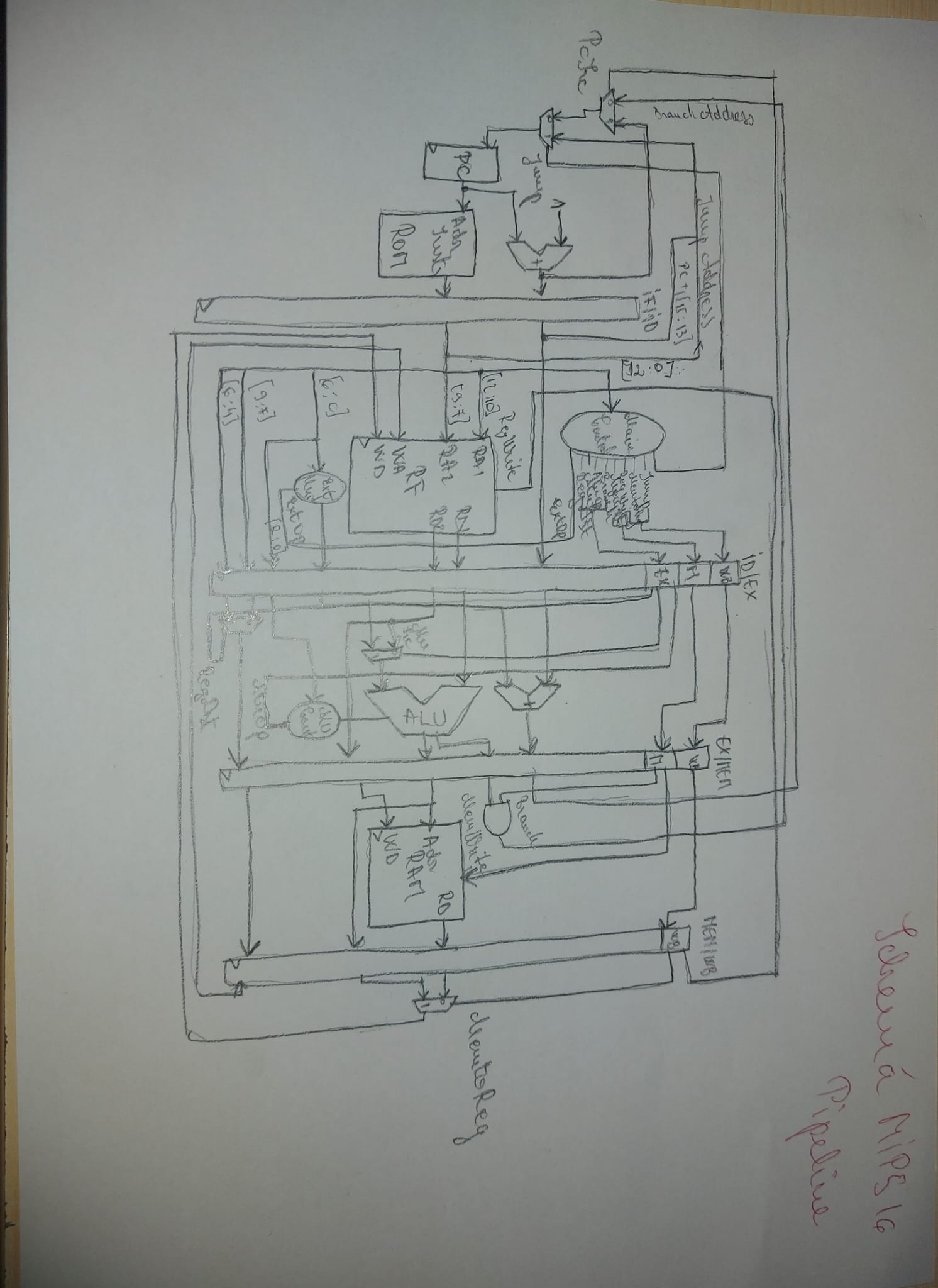
Prin împărțirea procesului de execuție în etape, pipeline-ul permite ca mai multe instrucțiuni să fie în diferite etape ale procesului în același timp, ceea ce duce la o creștere a performanței. De exemplu, în timp ce o instrucțiune este în etapa de execuție, o altă instrucțiune poate fi în etapa de decodare, iar o a treia instrucțiune poate fi în etapa de preluare. Aceasta permite o suprapunere eficientă a execuției instrucțiunilor și reduce timpul total de execuție.

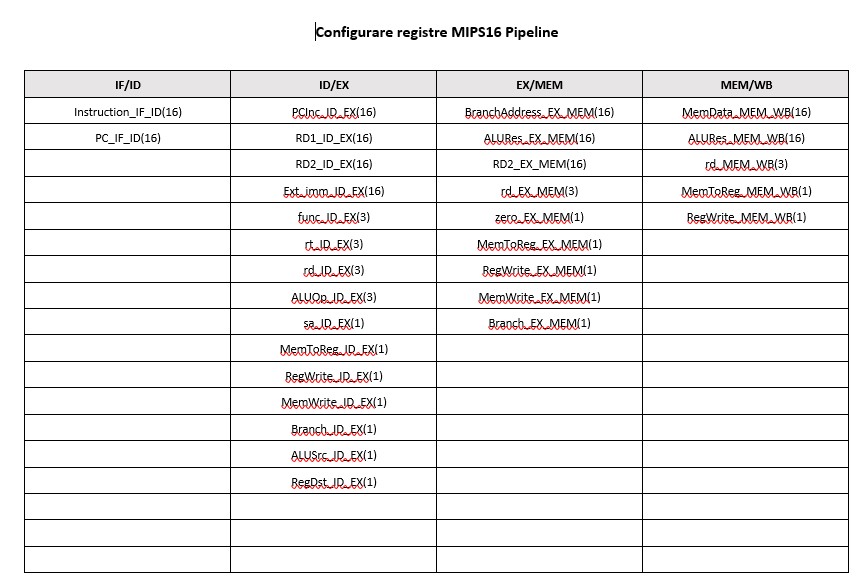
Pentru fiecare dintre aceste faze s-a creat o sursă separată în proiectul VHDL, iar componentele au fost incluse în main-ul *test\_env* (exceptând WB – care este implementată ca un multiplexor MUX 2:1). De asemenea, pentru a evita fenomenul de bounce și pentru a afișa rezultatele pe SSD-ul plăcii FPGA Basys 3, au fost implementate și incluse un MPG (Monopulse Generator), respectiv și un SSD (Seven Segment Display) în cadrul primelor laboratoare.

# **5. Scheme incluse**

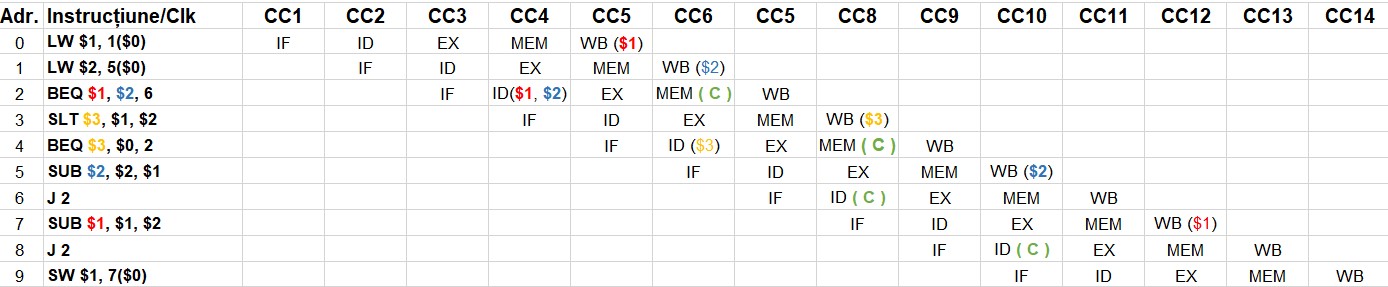
**Schema MIPS 16 Pipeline**

****

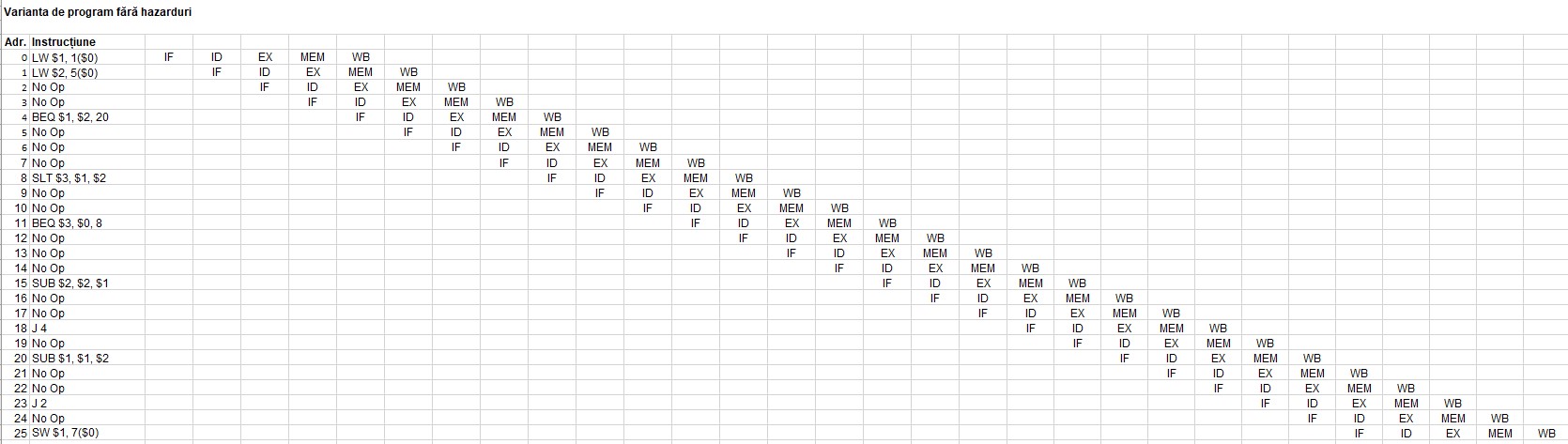
****

****

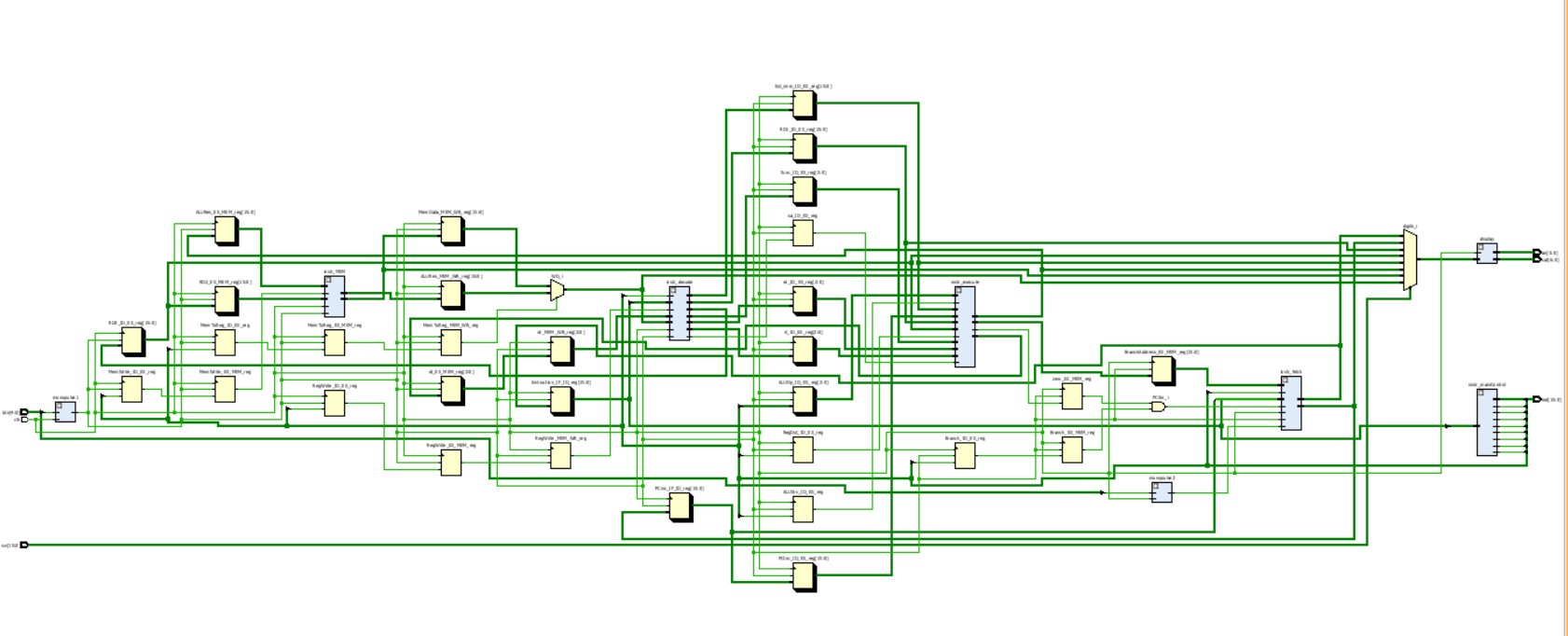
**Varianta de pipeline cu hazarduri identificate**

****

**Varianta de pipeline fără hazarduri**

****

**Schemă RTL MIPS 16 Pipeline**

****

# **6. Concluzii**

Cunoștințele dobândite în cadrul laboratoarelor 4-10 la Arhitectura Calculatoarelor au condus la întocmirea microprocesorului MIPS 16 Pipeline care realizează parțial corect programul întocmit în cod mașină și tratează hazardurile întâmpinate. Acesta a fost testat și verificat pe placa Basys 3 pentru corectitudine și certitudine, însă am întâmpinat probleme la Write Back la finalul programului.

# **7. Resurse necesare**

* Cunoștinte generale de hardware și programare dobândite la materiile Proiectare Logică, Proiectarea Sistemelor Numerice, Arhitectura Calculatoarelor, Programare în Limbaj de Asamblare
* Mediul de dezvoltare Xilinx VIVADO HL WebPACK (am folosit versiunea 2016.4)
* Basys 3 Artix-7 FPGA Trainer Board - Digilent

# **8. Bibliografie**

* <https://en.wikipedia.org/wiki/Stanford_MIPS>
* <https://www.cs.umd.edu/~meesh/411/CA-online/chapter/pipelining-mips-implementation/index.html>
* <https://users.utcluj.ro/~onigaf/files/teaching/AC/AC_indrumator_laborator.pdf>
* <https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/366-0.pdf>
* <https://users.utcluj.ro/~onigaf/files/AC.html>
* <https://users.utcluj.ro/~vcristian/AC.html>
* <https://www.youtube.com/@mirceapaulmuresan>