DOCUMENTATIE

MIPS16 P

Logo

Description automatically generated

NUME STUDENT: VOICU LAURA-LUISA

GRUPA: 30226

***CUPRINS***

1. **Modificari particulare pentru cele 4 instructiuni suplimentare**
2. **Tabel cu descrierea registrilor pipeline**
3. **Schema procesorului mips pipeline**
4. **Analiza hazardurilor**
5. **Activitati incomplete**
6. **RTL schematic**
7. **Testare placuta**

Tranzitia procesorului MIPS16 ciclu unic in pipeline s-a realizat prin intermediul a 4 registre, impartind procesorul in 5 zone , astfel incat executia instructiunilor sa se realizeze pe decursul a mai 5 perioade de ceas (pentru o singura instructiune) . Astfel, numarul total de timpi de ceas ar fi 5\*n(???) , insa putem reduce acest numar daca facem ca fiecare instructiune sa inceapa la exact urmatoarea peroada de ceas dupa inceperea instructiunii anterioare. Problemele ce apar la aceasta abordare sunt posibilitatea de a folosi un registru in care nu s-au completat datele de iesire (abia in ciclul 5 are loc acest proces) ca si operand pentru o instructiune urmatoare, sau citirea unui registru inainte ca acesta sa fie scris . Astfel apare problema hazardurilor (de date, structurale, de control) ce este rezolvata prin introducerea NOOP.

**1.Modificari particulare aduse celor 4 instructiuni**

🡪 Pentru instructiunea BNE semnificatia semnalului “zero” a fost inversata ( adica atunci cand rezultatul RF[rs]-RF[rt]!=0 zero<=1 ) pentru a putea trece la instructiunea pc+1+s\_ext(imm) 🡪 deci se comporta exact opus instructiunii beq ( fiind cea mai simpla abordare ).

🡪 Pentru instructiunea SLTI a fost adaugata o noua instructiune in ALU (aluCTRL=1000) ce compara (dpdv matematic cu ajutorul bibliotecii IEEE.STD\_LOGIC\_ARITH.ALL) continutul registrelor si seteaza rezultatul corespunzator ( deci nu voi avea componente “vizibile” pe schema, ci doar in intreriorul ALU).

**2.Tabel cu descrierea registrilor pipeline**

|  |  |  |  |
| --- | --- | --- | --- |
| **IF/ID** | **ID/EX** | **EX/MEM** | **MEM/WB** |
| pcPlus\_ID(16) | jmp(1) | memToReg\_mem(1) | memToReg\_wb(1) |
| instr\_ID(16) | memToReg\_ex(1) | regWrite\_mem(1) | regWrite\_wb(1) |
|  | regWrite\_ex(1) | branch\_mem(1) | memData\_wb(16) |
|  | branch\_ex(1) | aluSrc\_mem(1) | aluRes\_wb(16) |
|  | aluSrc\_ex(1) | memWrite\_mem(1) | result\_mux\_wb(3) |
|  | memWrite\_ex(1) | regDst\_mem(1) |  |
|  | regDst\_ex(1) | zero\_mem(1) |  |
|  | aluOp\_ex(3) | aluOp\_mem(3) |  |
|  | ext\_imm\_ex(16) | result\_mux\_mem(3) |  |
|  | func\_ex(3) | brAddress\_mem(16) |  |
|  | sa\_ex(1) | alures\_mem(16) |  |
|  | result\_mux\_ex(3) | rd2\_mem(16) |  |
|  |  |  |  |

**3.Schema procesorului MIPS16 Pipeline**

Observatie: desi au fost aduse modificari in cadrul instructiunilor bne si slti, nu a fost nevoie sa adaug componente noi, deoarece schimbarile s-au realizat la nivel de ALU (deci au fost introduse operatii noi in ALU);

A picture containing text, diagram, plan, handwriting

Description automatically generated

**4. Analiza hazardurilor**



Tabelul in dimensiunea originala format excel se gaseste in arhiva

Programul scris in urma adaugari de NOOP-urilor (obs: in cod exista mici modificari la final pentru a verifica corectitudinea programului):



Tabelul in format text se gaseste in arhiva.

**5. Activitati incomplete**

Procesorul are toate partile complete

**6.RTL schematic**

A picture containing diagram, plan, technical drawing, schematic

Description automatically generated

**7.Testare.Functionalitate**

Programul a fost testat pe placuta si functioneaza. In registrii $3 si $4 sunt calculate sumele corespunzatoare ( in $3 suma elementelor de pe indice impar si in $4 suma celor de pe indice par) si ulterior verificat daca acestea sunt egale (in registrul $1 se va scrie valoarea 1 daca sunt egale si 0 altfel 🡪 au fost verificate ambele cazuri)

b"001\_000\_001\_0000001", -- 2081 0. addi $1,$0,1 --i=1

b"001\_000\_010\_0000101" ,-- 2105 1. addi $2,$0,5 --j=5 -- la 5 se opreste ( functioneaza ca while)

b"001\_000\_011\_0000000", -- 2180 2. addi $3,$0,0 --sum1=0

b"001\_000\_100\_0000000", -- 2200 3. addi $4,$0,0 --sum2=0

b"000\_010\_001\_101\_0\_000", -- 08D0 4. add $5,$2,$1 --n = i+j

--

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

--

b"000\_000\_101\_101\_1\_011", -- 02DB 5. srl $5,$5,1 --n/2 -- marchez jumatatea vectorului

b"100\_010\_001\_0100000", -- 88A0 6. loop\_st: beq $1,$2,loop\_end = 32 -- deschide bucla

---

b"000\_000\_000\_000\_0\_000", --noop 7.

b"000\_000\_000\_000\_0\_000", --noop 8.

b"000\_000\_000\_000\_0\_000", --noop 9.

---

b"001\_000\_110\_0000001",-- 2301 10. addi $6,$0,1 -- $6=0001

---

b"000\_000\_000\_000\_0\_000", --noop 11.

b"000\_000\_000\_000\_0\_000", --noop 12.

---

b"000\_110\_001\_111\_0\_100", -- 18F4 13. and $7,$6,$1 $7 = 1 daca e i impar, 0 altfel

---

b"000\_000\_000\_000\_0\_000", --noop 14.

b"000\_000\_000\_000\_0\_000", --noop 15.

---

b"100\_111\_000\_000101", -- 9C09 16. beq $7,$0,par=11 -- verifica daca i par

---

b"000\_000\_000\_000\_0\_000", --noop 17.

b"000\_000\_000\_000\_0\_000", --noop 18.

b"000\_000\_000\_000\_0\_000", --noop 19.

---

b"010\_001\_101\_0000000", --4680 20. lw $5, $1

---

b"000\_000\_000\_000\_0\_000", --noop 21.

b"000\_000\_000\_000\_0\_000", --noop 22.

---

b"000\_101\_000\_101\_0\_000", -- 1450 23. add $5,$5,$0 --n = i+j

---

b"000\_000\_000\_000\_0\_000", --noop 24.

b"000\_000\_000\_000\_0\_000", --noop 25.

---

b"000\_011\_101\_011\_0\_000", -- 0EB0 26. impar: add $3,$3,$5

b"111\_0000000100110", -- E026 27. jmp incr = 38

---

b"000\_000\_000\_000\_0\_000", --noop 28. -- de aici incep sa calculez pt cel cu index par

---

b"010\_001\_101\_0000000", --4780 lw $5, $1

---

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

---

b"000\_101\_000\_101\_0\_000", -- 1C70 4. add $5,$5,$0 --n = i+j

---

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

---

b"000\_100\_101\_100\_0\_000", -- 12C0 12. par: add $4,$4,$5

b"001\_001\_001\_0000001",-- 2481 13. incr: addi $1,$1,1

b"111\_0000000001000", -- E008 14. jmp 8 --loop\_st

--

b"000\_000\_000\_000\_0\_000", --noop

---

b"100\_011\_100\_0000110", -- 8E06 15. loop\_end: beq $3,$4, equal

---

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

---

b"001\_000\_001\_0000000",-- 2080 13. addi $1,$0,0 -- $6=0001

b"111\_0000000110011", -- E033 14. jmp done=51

---

b"000\_000\_000\_000\_0\_000", --noop

---

b"001\_000\_001\_0000001",-- 2081 13. equal: addi $1,$0,1 -- $6=0001

---

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

---

b"000\_001\_000\_001\_0\_000", -- 0410 4. --done add $1,$1,$0 --n = i+j

---

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

---

b"100\_001\_001\_0000100", --8484 sw $1, $1

--- verific daca sumele retinute in $

--ma asigur ca am terminat cu sw (noopx5 safest)

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

b"000\_000\_000\_000\_0\_000", --noop

--fac un addi $3,0,0 sa vad ce imi afiseaza pe rd1

b"001\_000\_011\_0000000", --2180 addi $3,$0,0 --i=1 --if

--ma asigur ca termin cu cea de sus

b"000\_000\_000\_000\_0\_000", --noop --aici imi da rd1,rd2 pt aia de sus --id

b"000\_000\_000\_000\_0\_000", --noop --ex

b"000\_000\_000\_000\_0\_000", --noop --mem

b"000\_000\_000\_000\_0\_000", --noop --wb

--fac un addi $4,0,0 sa vad ce imi afiseaza pe rd1

b"001\_000\_100\_0000000", --2200 addi $4,$0,0 --i=1

b"000\_000\_000\_000\_0\_000", --noop --aici imi da rd1,rd2 pt aia de sus --id

b"000\_000\_000\_000\_0\_000", --noop --ex

b"000\_000\_000\_000\_0\_000", --noop --mem

b"000\_000\_000\_000\_0\_000", --noop --wb

---