**Mips pipeline – 16 bit**

1. **Modificari pentru instructiunile alese**

Singura modificare la calea de date adusa pentru instructiunile alese de mine a fost pentru instructiunea bgetz (branch grater tan or equal to zero) unde a trebuit sa mai generez in uc un semnal pentru detectarea operatiei pe care l-am transmis mai apoi prin registrele intermediare sis a generez un semnal in alu care este active daca primul operand este mai mare decat 0, pe care se asemenea l-am trimis in register pana in MEM unde am generat semnalul PCSrc pentru muxul din IF care determina daca se face sau nu saltul la adresa de branch.

**xor $rs $rt $rd**

**format:** 000\_sss\_ttt\_ddd\_0\_110 (sa nu conteaza)

RF[rd] <= RF[rs] xor Rf[rt]

nu au trebuit aduse modificari deoarece e ca toate operatiile r si inca mai erau “locuri” in alu pt operatii cu un semnal de control pe 3 biti

**sra $rs $rd sa**

**format:** 000\_sss\_ttt\_ddd\_sa\_111

RF[rd] <= RF[rs] >>>sa;

nu au trebuit aduse multe modificari deoarece este ca orice alta instructiune r

**andi $rs $rt imm**

**format:** 101\_sss\_ttt\_iiiiiii

RF[rt] <= RF[rs] and Zero\_ext(imm)

este similar cu addi dar este diferit semnalul de alu op deci nu au fost aduse modificari la calea de date

**bgez $rs imm**

**format:** 110\_sss\_000\_iiiiiii

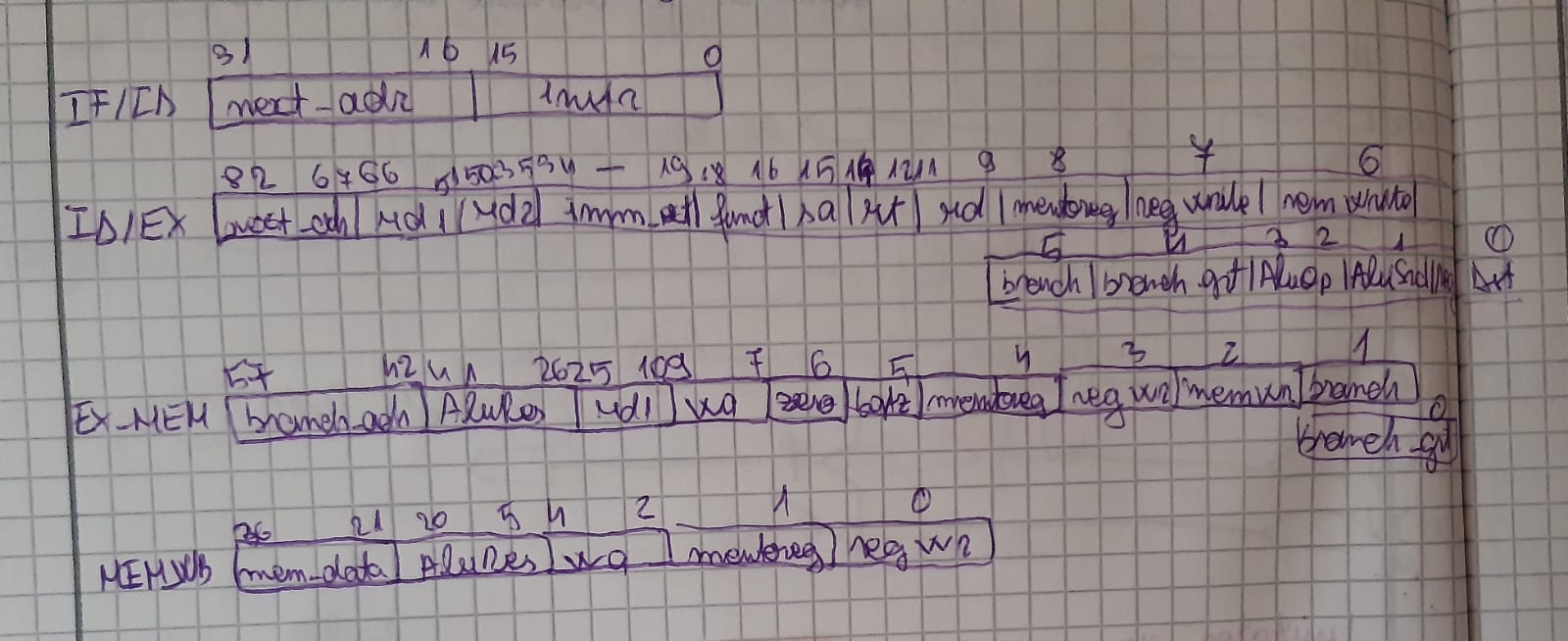
if(RF[rs]>0)

pc<pc+1+sign\_ext(imm)

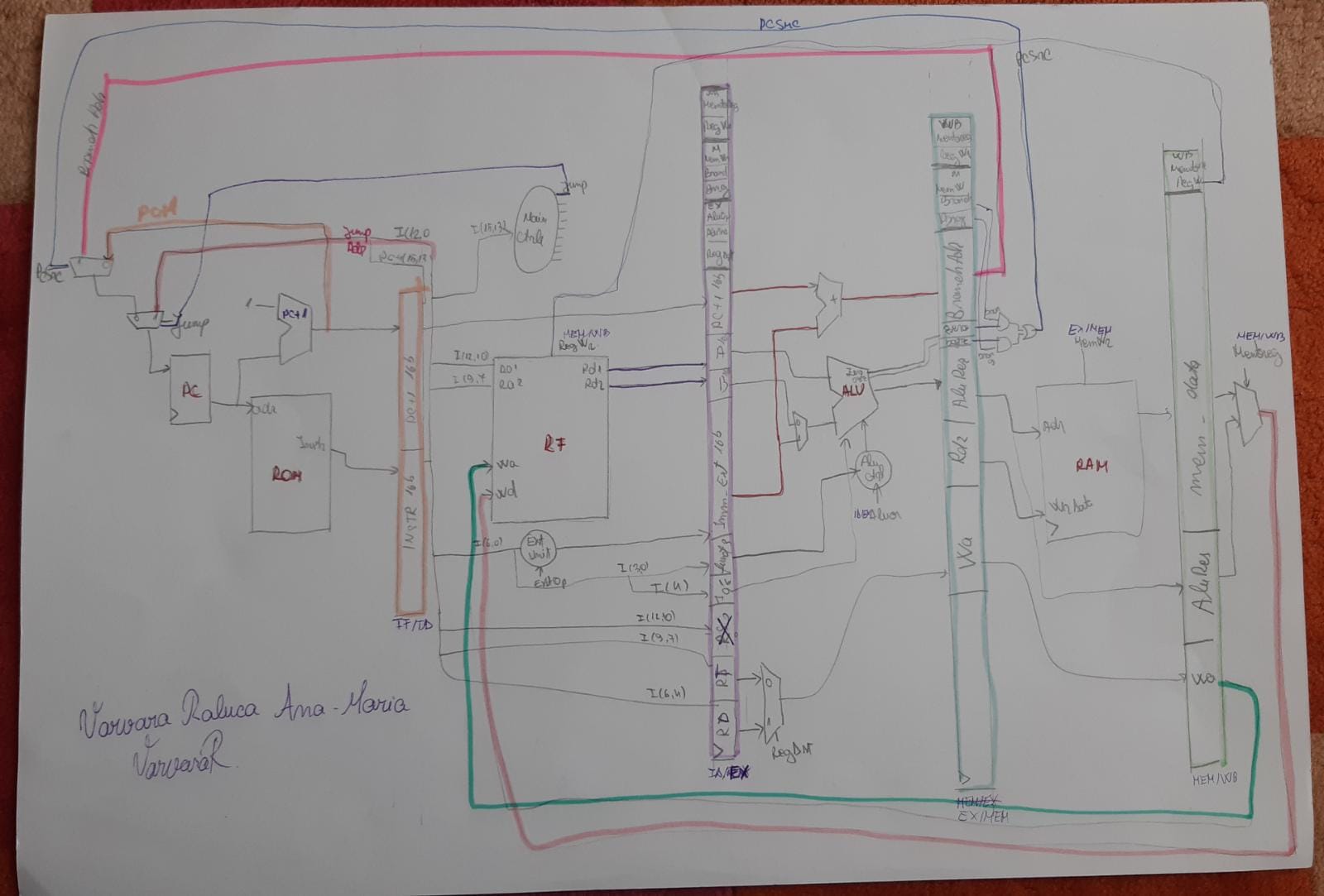
am adaugat un nou flag ca iesire la alu si un nou semnal de control branch\_greater si l-am adaugat ca semnal de control la intrarea in muxul care allege dintre pc+1 si branch address, (adica semnalul pt mux va fi (zero&branch)|(branch\_greater&bgtez)

1. **Registrii pipeline**

|  |  |  |  |
| --- | --- | --- | --- |
| **REG\_IF\_ID(31** – **0)** | **REG\_ID\_EX(<msb>** – **0>)** | **REG\_EX\_MEM(<msb>** – **0)** | **REG\_MEM\_WB(<msb>** – **0)** |
| PC+1(31 – 16) | PC+1(82 – 67) | Branch\_address(57 – 42) | Mem\_data(36 – 21) |
| Instruction(15 – 0) | RD1(66 - 51) | AluRes(41 – 26) | AluRes(20 – 5) |
|  | RD2(50 – 35) | RD1(25 – 10) | Write\_address(4 – 2) |
|  | Imm\_ext(34 – 19) | Write\_address(9 – 7) | Memtoreg(1) |
|  | Funct(18 – 16) | Zero(6) | Regwrite(0) |
|  | sa(15) | Bgtz(5) |  |
|  | Rt(14 – 12) | Memtoreg(4) |  |
|  | Rd(11 – 9) | Regwrite(3) |  |
|  | Memtoreg(8) | Memwrite(2) |  |
|  | Regwrite(7) | Branch(1) |  |
|  | Memwrite(6) | Branch\_grt(0) |  |
|  | Branch(5) |  |  |
|  | Branch\_grt(4) |  |  |
|  | AluOp(3-2) |  |  |
|  | AluSrc(1) |  |  |
|  | Regdst(0) |  |  |



1. **Calea de date**



1. **Rezolvarea hazardurilor**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cc1 | Cc2 | Cc3 | Cc4 | Cc5 | Cc6 | Cc7 | Cc8 | Cc9 | Cc10 | Cc11 | Cc12 | Cc13 | Cc14 | Cc15 |
| 1.add $1, $0 $0 | IF | ID $0 | EX | MEM | WB($1) |  |  |  |  |  |  |  |  |  |  |
| 2.addi $3 $0 10 |  | IF | ID $0 | EX | MEM | WB($3) |  |  |  |  |  |  |  |  |  |
| 3.lw $4 0($1) |  |  | IF | ID($1) | EX | MEM | WB($4) |  |  |  |  |  |  |  |  |
| 4.lw $5 0($1) |  |  |  | IF | ID($1) | EX | MEM | WB($5) |  |  |  |  |  |  |  |
| 5.sw $4 11($0) |  |  |  |  | IF | ID($4) | EX | MEM | WB |  |  |  |  |  |  |
| 6.sw $5 12($0) |  |  |  |  |  | IF | ID($5) | EX | MEM | WB |  |  |  |  |  |
| 7.beq $1 $3 11 |  |  |  |  |  |  | IF | ID($1$3) | EX | MEM | WB |  |  |  |  |
| 8.lw $6 0($1) |  |  |  |  |  |  |  | IF | ID($1) | EX | MEM | WB($6) |  |  |  |
| 9.sub $7 $5 $6 |  |  |  |  |  |  |  |  | IF | ID($5$6) | EX | MEM | WB($7) |  |  |
| 10.bgez $7 2 |  |  |  |  |  |  |  |  |  | IF | ID($7) | EX | MEM | WB |  |
| 11.add $5 $0 $6 |  |  |  |  |  |  |  |  |  |  | IF | ID($6) | EX | MEM | WB($5) |

Pana la iteratia 10

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cc8 | Cc9 | Cc10 | Cc11 | Cc12 | Cc13 | Cc14 | Cc15 | Cc16 | Cc17 | Cc18 | Cc19 | Cc20 | Cc21 | Cc22 |
| 10.bgez $7 2 | IF | ID($7) | EX | MEM | WB |  |  |  |  |  |  |  |  |  |  |
| 11.add $5 $0 $6 |  | IF | ID($6) | EX | MEM | WB($5) |  |  |  |  |  |  |  |  |  |
| 12.sw $6 12($0) |  |  | IF | ID($6) | EX | MEM | WB |  |  |  |  |  |  |  |  |
| 13.sub $7 $6 $4 |  |  |  | IF | ID($6$4) | EX | MEM | WB($7) |  |  |  |  |  |  |  |
| 14.bgez $7 2 |  |  |  |  | IF | ID($7) | EX | MEM | WB |  |  |  |  |  |  |
| 15.add $4 $0 $6 |  |  |  |  |  | IF | ID($6) | EX | MEM | WB($4) |  |  |  |  |  |
| 16.sw $6 11($0) |  |  |  |  |  |  | IF | ID($6) | EX | MEM | WB |  |  |  |  |
| 17.addi $1 1 |  |  |  |  |  |  |  | IF | ID($1) | EX | MEM | WB($1) |  |  |  |
| 18.j 6 (in prog instr sunt num de la 0 deci aici se sare la 7) |  |  |  |  |  |  |  |  | IF | ID | EX | MEM | WB |  |  |
| 7.beq $1 $3 11 |  |  |  |  |  |  |  |  |  | IF | ID($1) | EX | MEM | WB |  |

La iteratia 10

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cc16 | Cc17 | Cc18 | Cc19 | Cc20 | Cc21 | Cc22 | Cc23 | Cc24 | Cc25 | Cc26 | Cc27 | Cc20 | Cc21 | Cc22 |
| 18.j 6 | IF | ID | EX | MEM | WB |  |  |  |  |  |  |  |  |  |  |
| 19.lw $4 11($0) |  | IF | ID | EX | MEM | WB($4) |  |  |  |  |  |  |  |  |  |
| 20.lw $5 12($0) |  |  | IF | ID | EX | MEM | WB($5) |  |  |  |  |  |  |  |  |
| 21.add $6 $4 $5 |  |  |  | IF | ID($4$5) | EX | MEM | WB($6) |  |  |  |  |  |  |  |
| 22.sra $6 1 |  |  |  |  | IF | ID($6) | EX | MEM | WB($6) |  |  |  |  |  |  |

**Programul inainte de rezolvarea hazardurilor**

0 add $1, $0 $0

1 addi $3 $0 10

2 lw $4 0($1) -- minim

3 lw $5 0($1) --maxim

4 sw $4 11($0) --11 minim

5 sw $5 12($0) --12 maxim

6 beq $1 $3 11 --if(i==10)jump

7 lw $6 0($1) --elem curent

8 sub $7 $5 $6 --max

9 bgez $7 2

10 add $6 $0 $5 5 = 6+0

11 sw $6 12($0)

12 sub $7 $6 $4 --min

13 bgez $7 2

14 add $4 $0 $6 4 = 6 + 0

15 sw $4 11($0)

16 addi $1 1

17 j 6

18 lw $4 11($0)

19 lw $5 12($0)

20 add $6 $4 $5

21 sra $6 1

**Program dupa rez hazardurilor**

0. add $1, $0 $0

1. addi $3$0 10

2. noop

3. lw $4 0($1)

4. lw $5 0($1)

5. noop

6. sw $4 11($0)

7. sw $5 12($0)

8. beq $1 $3 27 --grija la chimbat

9. noop

10. noop

11. noop

12. lw $6 0($1)

13. noop

14. noop

15. sub $7 $5 $6

16. noop

17. noop

18. bgez $7 5 --grija la schimbat

19. noop

20. noop

21. noop

22. add $5 $0 $6

23. sw $6 12($0)

24. sub $7 $6 $4

25. nnop

26. noop

27. bgez $7 5 --grija la schimbat

28. noop

29. noop

30. noop

31. add $4 $0 $6

32. sw $6 11($0) --grija la schimbat

33. addi $1 1

34. noop

35. j 8 --grija la schimbat

36. noop

37. lw $4 11($0)

38. lw $5 12($0)

39. noop

40. noop

41. add $6 $4 $5

42. noop

43. noop

44. sra $6 1

**Hazardul structural** (care in acest caz e mai mult de date) in care se vrea accesul in acelasi timp la registrii (de ex intre 1 si 4 la ciclul 5 de ceas se vrea citirea si scrierea in acelasi timp) l-am rezolvat modificand scrierea in registru sa fie pe frontal descrescator astfel incat se citesc operanzii asincron in a doua partea a tactului, dup ace se scrie.

**Hazard de date**

Am intalnit hazard de date conform tabelului facut mai sus intre instructiuniile urmatoare:

* 1-3, hazard raw pe registrul 1
* 3-5, hazard raw pe registrul 4
* 4-6, hazard raw pe registrul 5
* 8-9, hazard raw pe registrul 6
* 9-10, hazard raw pe registrul 7
* 13-14, hazard raw pe registrul 7
* 17-7(este hazard daca se sare – aici am rezolvat adaugand un noop fix dupa 17), hazard raw pe registrul 1
* 19-21, hazard raw pe registrul 4
* 20-21, hazard raw pe registrul 5
* 21-22., hazard raw pe registrul 6

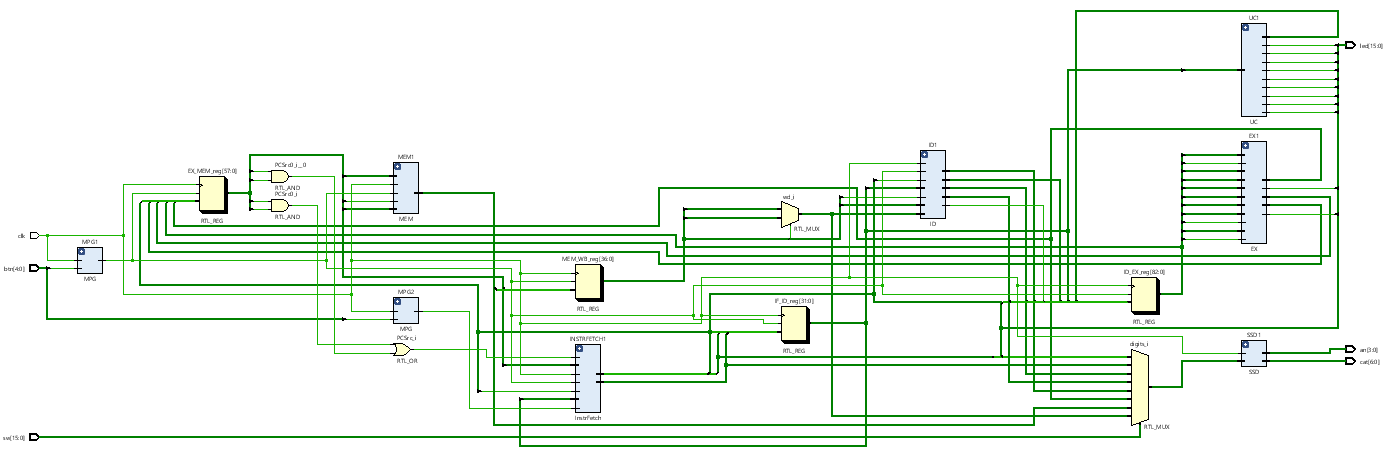
Pt a rezolva acest tip de hazard am adaugat cate noopuri a fost nevoie in functie de cate cicluri de ceas sunt intre wb al primei instructiuni si id al celei de-a doua ( de exemplu daca wb al primei instr este cu 2 cicluri de ceas dupa id-ul celei de-a doua am adaugat 2 noop intre ele, dar daca este dupa un ciclu de ceas am adaugat numai un noop dupa prima). In multe cazuri rezolvarea hazardului dintre 2 instr a rezolvat hazardul dintre alte 2 instructiuni (de exemplu rezolvarea 4-6 cu punerea unui noop dupa 4 a rezolvat 3-5, iar 20-21 a rezolvat 19-21).

**Hazard de control**

Pt a rezolva acest tip de hazard care apare la fiecare instructiune de branch sau de jump, unde daca se face saltul vor intra in executie un anumit numar de instructiuni fara sa trepuiasca (3 la branch una la jump), am adaugat cate 3 noop -uri dupa fiecare branch si unul dupa jump (as fi putut pune instructiunea de incrementare a reg $1 dupa jump deoarece oricum trebuia executata dar aveam nevoie de rezultatul din registrul 1 in prima instructiune la care se sare asa ca am pus un noop).

1. **Descriere VHDL**

Toate partile din descrierea vhdl sunt corecte si functionale, nu am nimic lipsa, exista doar warninguri de genul celor prezentate la mips single cycle, dar arhitectura este functionala si testata pe placuta.



1. **Testare pe placuta**

Nu am mai schimbat case-ul de afisare pe afisoare si legarea la leduri si se vor vedea rezultate de la diferite instructiuni din etajul de executie la care sunt in ciclul de ceas curent.

Linkul catre videoclip: <https://youtu.be/JnpfCcUcSak>