DOCUMENTATIE

MIPS16 P

UNIVERSITATEA
TEHNICĂ
DIN CLUJ-NAPOCA

NUME STUDENT: VOICU LAURA-LUISA
GRUPA: 30226

CUPRINS

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).

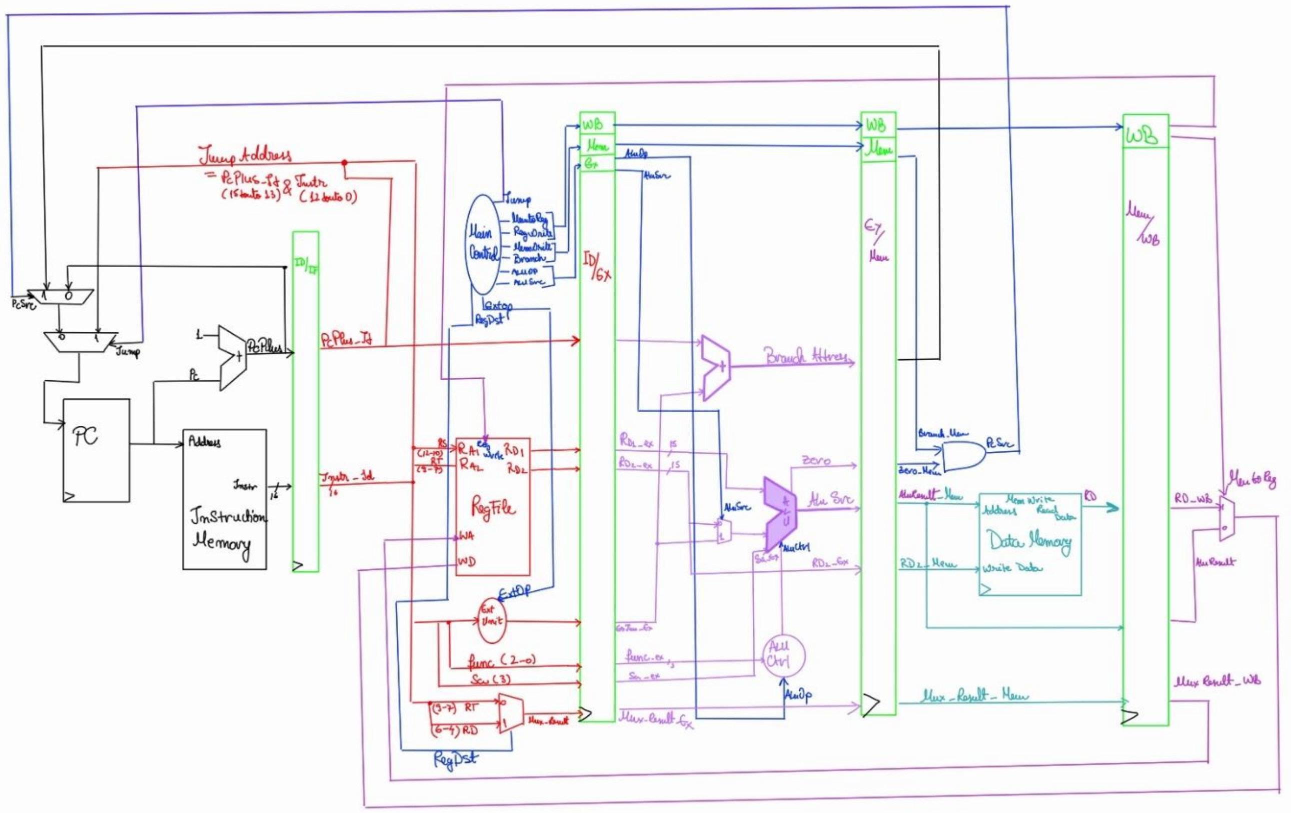
→ Pentru instructiunea SLTI a fost adaugata o noua instructiune in ALU (aluCTRL=1000) ce compara (dpdv matematic cu ajutorul bibliotecii IEEE.STD_LOGIC_ARITH.ALL) continutul registrelor si seteaza rezultatul corespunzator (deci nu voi avea componente "vizibile" pe schema, ci doar in intreriorul ALU).

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

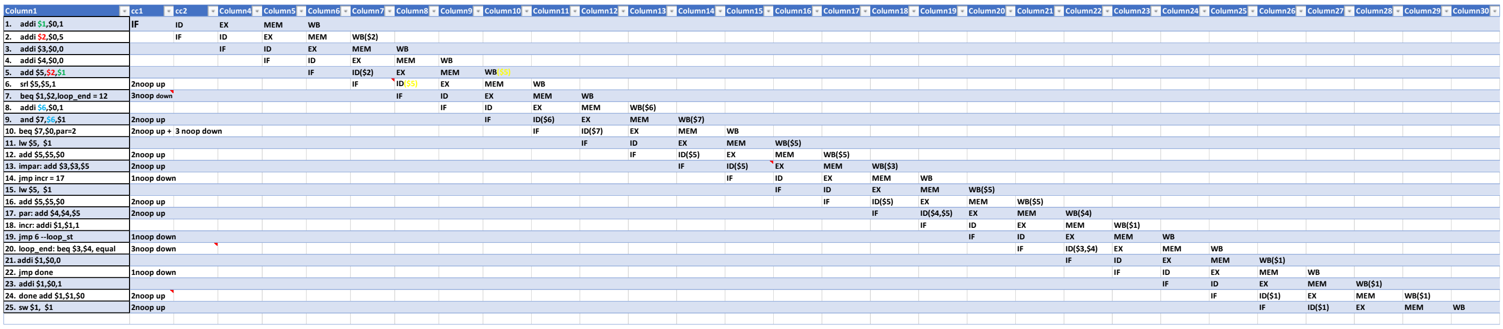
Observatie: desi au fost aduse modificari in cadrul instructiunilor bne si slti, nu a fost nevoie sa adaug componente noi, deoarece schimbarile s-au realizat la nivel de ALU (deci au fost introduse operatii noi in ALU);



Mips 16 Pipeline

Voicu LAURA-LUISA , grupa 30226

4. Analiza hazardurilor



Column1	cc1	cc2	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12	Column13	Column14	Column15	Column16	Column17	Column18	Column19	Column20	Column21	Column22	Column23	Column24	Column25	Column26	Column27	Column28	Column29	Column30
1. addi $1,$0,1	IF	ID	EX	MEM	WB																								
2. addi $2,$0,5		IF	ID	EX	MEM	WB($2)																							
3. addi $3,$0,0			IF	ID	EX	MEM	WB																						
4. addi $4,$0,0				IF	ID	EX	MEM	WB																					
5. add $5,$2,$1					IF	ID($2)	EX	MEM	WB($5)																				
6. srl $5,$5,1	2noop up					IF	ID($5)	EX	MEM	WB																			
7. beq $1,$2,loop_end = 12	3noop down						IF	ID	EX	MEM	WB																		
8. addi $6,$0,1								IF	ID	EX	MEM	WB($6)																	
9. and $7,$6,$1	2noop up								IF	ID($6)	EX	MEM	WB($7)																
10. beq $7,$0,par=2	2noop up +	3 noop down								IF	ID($7)	EX	MEM	WB															
11. lw $5, $1											IF	ID	EX	MEM	WB($5)														
12. add $5,$5,$0	2noop up											IF	ID($5)	EX	MEM	WB($5)													
13. impar: add $3,$3,$5	2noop up												IF	ID($5)	EX	MEM	WB($3)												
14. jmp incr = 17	1noop down													IF	ID	EX	MEM	WB											
15. lw $5, $1															IF	ID	EX	MEM	WB($5)										
16. add $5,$5,$0	2noop up															IF	ID($5)	EX	MEM	WB($5)									
17. par: add $4,$4,$5	2noop up																IF	ID($4,$5)	EX	MEM	WB($4)								
18. incr: addi $1,$1,1																		IF	ID	EX	MEM	WB($1)							
19. jmp 6 --loop_st	1noop down																		IF	ID	EX	MEM	WB						
20. loop_end: beq $3,$4, equal	3noop down																			IF	ID($3,$4)	EX	MEM	WB					
21. addi $1,$0,0																					IF	ID	EX	MEM	WB($1)				
22. jmp done	1noop down																					IF	ID	EX	MEM	WB			
23. addi $1,$0,1																							IF	ID	EX	MEM	WB($1)		
24. done add $1,$1,$0	2noop up																							IF	ID($1)	EX	MEM	WB($1)	
25. sw $1, $1	2noop up																								IF	ID($1)	EX	MEM	WB

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):

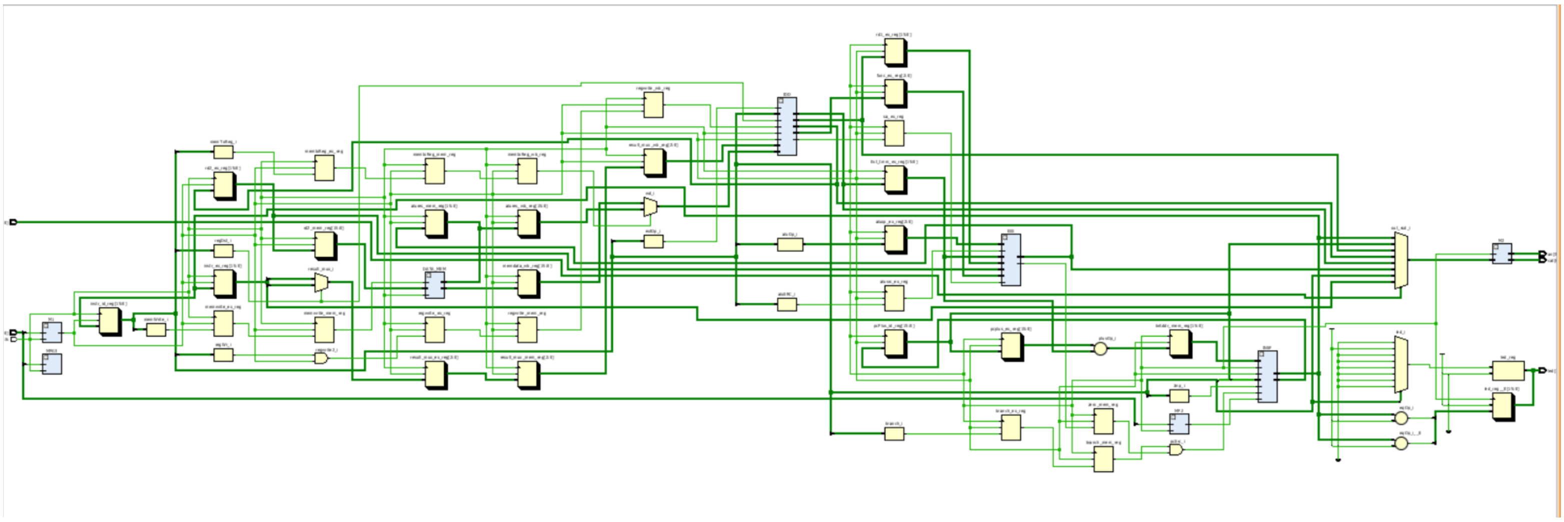
Instructiune
0. addi $1,$0,1
1. addi $2,$0,5 --j=5
2. addi $3,$0,0
3. addi $4,$0,0
4. add $5,$2,$1
5. add $0,$0,0
6. add $0,$0,0
7. srl $5,$5,1
8. loop_start : beq $1,$2,loop_end = 32 !!!!! –
9. add $0,$0,0
10. add $0,$0,0
11. add $0,$0,0
12. addi $6,$0,1
13. add $0,$0,0
14. add $0,$0,0
15. and $7,$6,$1
16. add $0,$0,0
17. add $0,$0,0
18. beq $7,$0,par=11
19. add $0,$0,0
20. add $0,$0,0
21. add $0,$0,0
22. lw $5, $1
23. add $0,$0,0
24. add $0,$0,0
25. add $5,$5,$0
26. add $0,$0,0
27. add $0,$0,0
28. impar: add $3,$3,$5
29. jmp incr = 38
30. add $0,$0,0 de aici calculez pt cel cu index par
31. lw $5, $1
32. add $0,$0,0
33. add $0,$0,0
34. add $5,$5,$0
35. add $0,$0,0
36. add $0,$0,0
37. par: add $4,$4,$5
38. incr: addi $1,$1,1
39. jmp 8 --loop_st
40. add $0,$0,0
41. loop_end: beq $3,$4, equal=6
42. add $0,$0,0
43. add $0,$0,0
44. add $0,$0,0
45. addi $1,$0,0
46. jmp done=51
47. add $0,$0,0
48. equal: addi $1,$0,1
49. add $0,$0,0

Tabelul in format text se gaseste in arhiva.

5. Activitati incomplete

Procesorul are toate partile complete

6.RTL schematic



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
---
```