MIPS (MICROPROCESOR) CICLU UNIC

Studentă: Alina Aurică

Profesor îndrumător: Vlad Miclea

Materie:Arhitectura Calculatoarelor

CUPRINS:

1. INSTRUCȚIUNI:

Instrucțiunile care pot fi executate pe un microprocesor ciclu unic sunt de 3 tipuri:

- De tip R: dintre care aveam obligativitatea să folosim instrucțiunile add, sub, sll, srl, and, or. Pe lângă acestea am ales să folosesc instrucțiunile xor și nor.

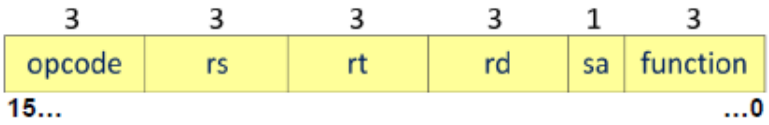
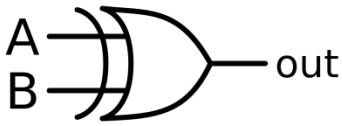


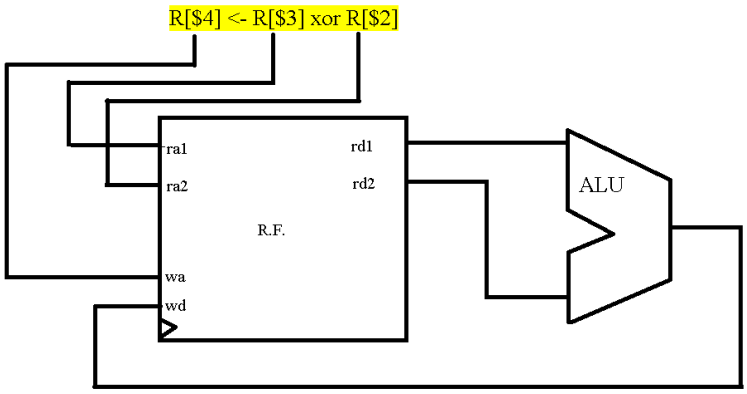
Figura 5-2: Instrucțiune de tip R

XOR reprezintă sau_exclusiv, având următorul tabel de adevăr:

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

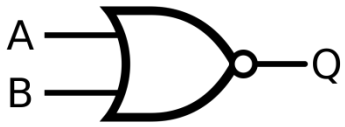


În microprocesor este de forma: xor $4 $3 $2, codificat cu următoarea adresă de memorie: b"000_011_010_100_0_110", și se traduce în: R[$4] <- R[$3] xor R[$2], unde $3 este registrul sursă, $4 este registrul destinație, iar $2 este registrul targhet.

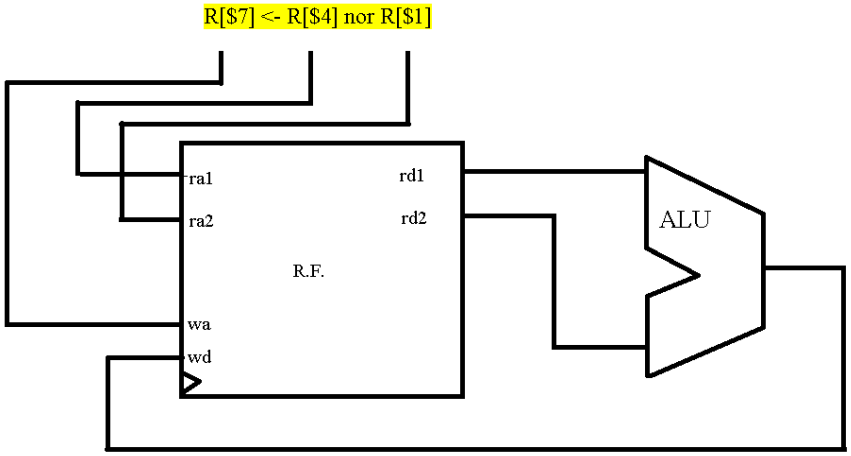


NOR reprezintă sau negat și are următorul tabel de adevăr

A	B	S
0	0	1
0	1	0
1	0	0
1	1	0



În microprocesor este de forma: nor $7 $4 $1, codificat cu următoarea adresă de memorie: b"000_100_001_111_0_111" și se traduce în: R[$7] <- R[$4] nor R[$1], unde $4 este registrul sursă, $7 este registrul destinație, iar $1 este registrul targhet.



- De tip I: dintre care aveam obligativitatea să folosim instrucțiunile addi, lw, sw, bqe. Pe lângă acestea am ales să folosesc instrucțiunile xori și ori.

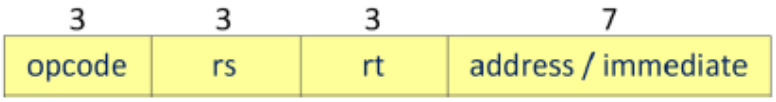
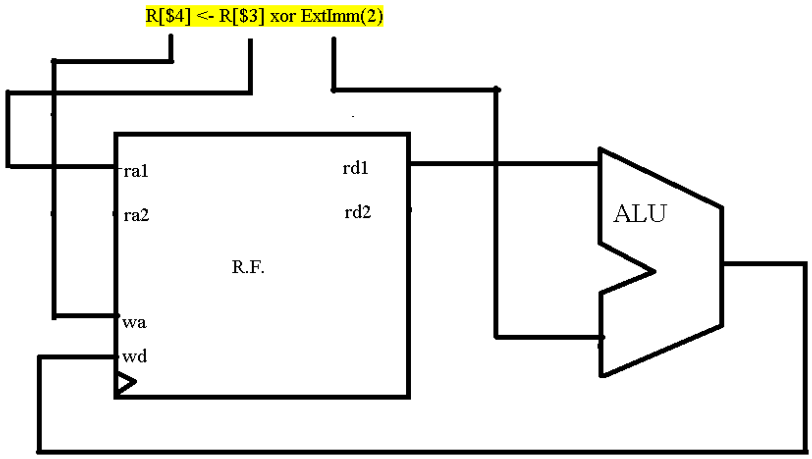


Figura 5-3: Instrucțiune de tip I

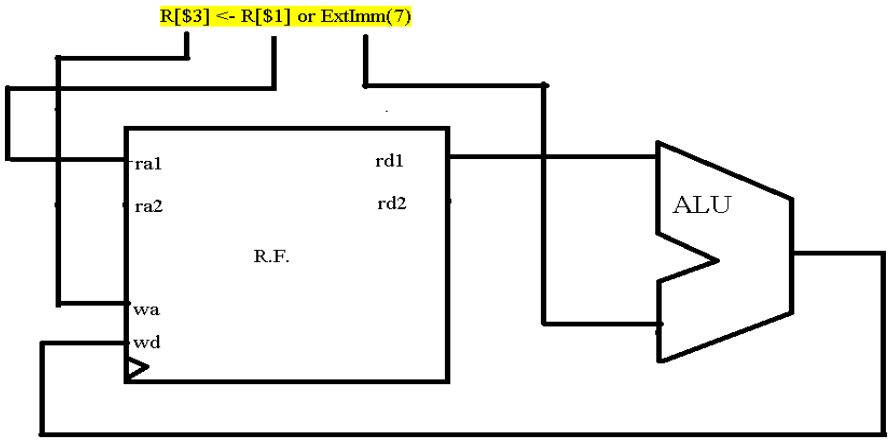
XORI se comportă exact ca și xor-ul, singura diferență este că operația se face între registrul sursă și immediate (care este o valoare pe care noi o impunem), iar rezultatul se reține în registrul targhet.

În microprocesor este de forma: xori $4 $3 2, codificat cu următoarea adresă de memorie: b"110_011_100_0000010" și se traduce în: R[$4] <- R[$3] xor ExtImm(2), unde $3 este registrul sursă, iar $4 este registrul targhet.



ORI se comportă exact ca și or-ul, singura diferență este că operația se face între registrul sursă și immediate (care este o valoare pe care noi o impunem), iar rezultatul se reține în registrul targhet.

În microprocesor este de forma: ori $3 $1 7, codificat cu următoarea adresă de memorie: b"101_001_011_0000111" și se traduce în: R[$3] <- R[$1] or ExtImm(7), unde $3 este registrul sursă, iar $4 este registrul targhet.



- De tip J: dintre care instrucțiunea jump, care face un salt la adresa pe care o stabilim

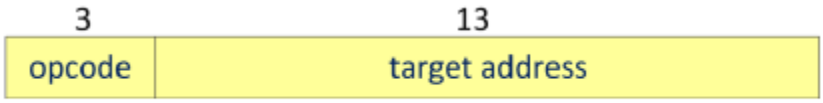


Figura 5-4: Instrucțiune de tip J

2. TABELE CU VALORI:

Instrucțiunea	Opcode	RegDst	ExtOp	ALUSrc	Branch	Jump	MemWrite	MemtoReg	RegWrite	ALUOp	func	ALUCtrl
Add	000	1	X	0	0	0	0	0	1	000	000	000
Sub	000	1	X	0	0	0	0	0	1	000	001	001
Sll	000	1	X	0	0	0	0	0	1	000	010	010
Srl	000	1	X	0	0	0	0	0	1	000	011	011
And	000	1	X	0	0	0	0	0	1	000	100	100
Or	000	1	X	0	0	0	0	0	1	000	101	101
Xor	000	1	X	0	0	0	0	0	1	000	110	110
Nor	000	1	X	0	0	0	0	0	1	000	111	111
Addi	001	0	1	1	0	0	0	0	1	001	xxx	000
Lw	010	0	1	1	0	0	0	1	1	010	xxx	000
Sw	011	0	1	1	0	0	1	0	0	011	xxx	000
Bqe	100	0	1	1	1	0	0	0	0	100	xxx	100
Ori	101	0	1	1	0	0	0	0	1	101	xxx	101
Xori	110	0	1	1	0	0	0	0	1	110	xxx	110
Jump	111	0	X	X	0	1	0	0	0	xxx	xxx	xxx

3. PROGRAMUL CARE RULEAZĂ PE MICROPROCESOR:

a=3, b=2, c=6

r=((a+b)*4-c)/4+7

if r == 0

r = 7; (ori)

r = r-2; (xori)

else

r = 0; (and)

r = 16; (or)

break;

r = 27; (xor)

r = FFE4; //in hexazecimal (nor) –incercam să-l ducem cât mai aproape de maxim

Având niște valori date, vom încerca să rezolvăm o problemă de forma ((a+b)*4-c)/4+7, după care să aflăm dacă rezultatul e zero sau nu. În funcție de răspuns, se vor executa diferite calcule cu acesta.

4. TRASAREA EXECUȚIEI PROGRAMULUI:

Add $5 $1 $2

Sll $3 $5 $0

Sub $4 $3 $6

Srl $7 $4 $0

Addi $1 $7 7

Sw $5 $1 0

Lw $2 $5 0

Beq $0 $1 3

And $6 $1 $4

Or $3 $6 $4

Jump 13

Ori $3 $1 7

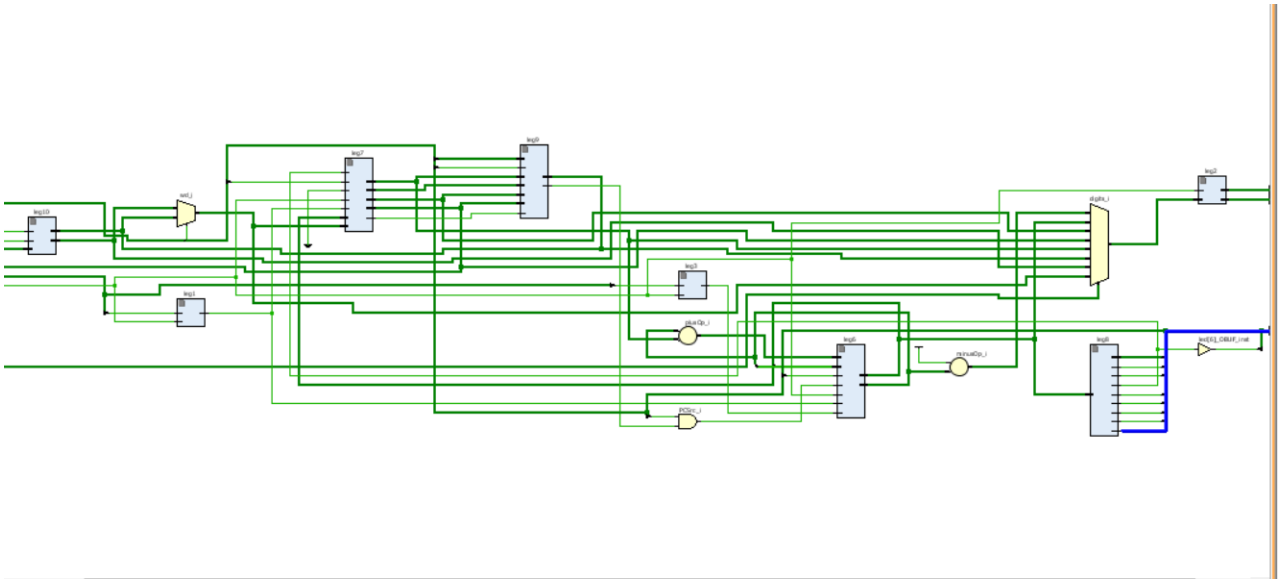
Xori $4 $3 2

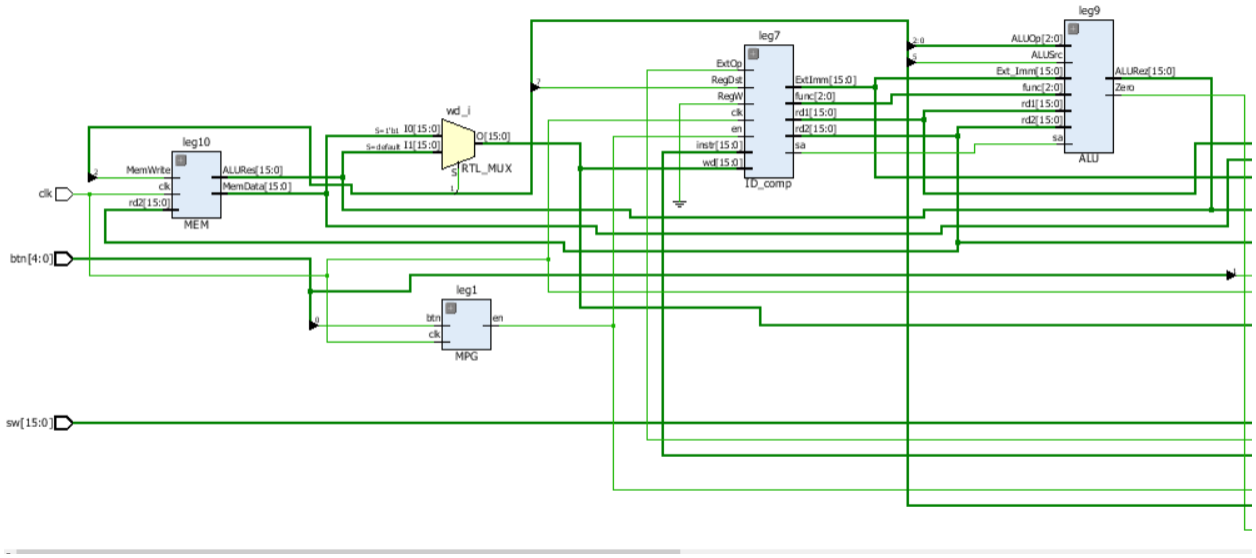
Xor $4 $3 $2

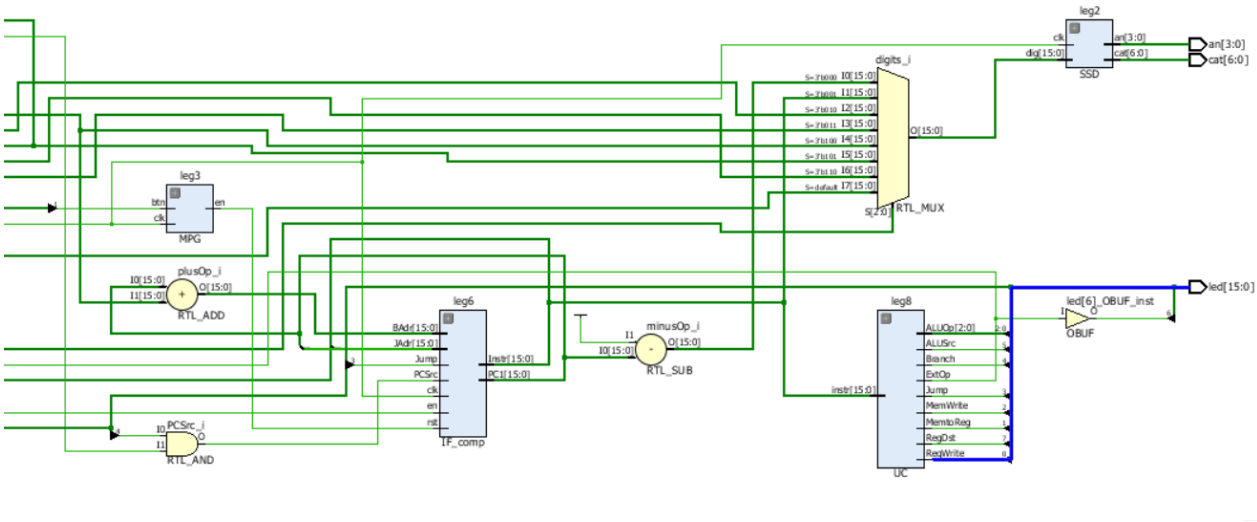
Nor $7 $4 $1

```
signal ROM: Rom_type := (b"000_001_010_101_0_000", --add 0550 rd1=3 rd2=2 ALURez=5 (wd)
                         b"000_101_000_011_1_010", --sll 143A rd1=5 rd2=0 ALURez=14(hexazecimal, 20(zecimal) (wd)
                         b"000_011_110_100_0_001", --sub 1f41 rd1=14 rd2=4 ALURez=10(hezazecimal), 16(zecimal) (wd)
                         b"000_100_000_111_1_011", --srl 107b rd1=10 rd2=0 ALURez=4 (wd)
                         b"001_111_001_0000111", --addi 3c87 rd1=4 ExtImm=7 ALURez=b(hexazecimal), 11(zecimal) (wd)
                         b"011_001_101_0000000", --sw 6680 rd1=b rd2=b ALURez=b, in memorie, tot b
                         b"010_101_010_0000000", --lw 5500 rd1=b rd2=b, in wd ecrie b
                         b"100_001_000_0000011", --bqe 8403 rd1=b rd2=0 ExtImm=3
                         b"000_001_100_110_0_100", --and 0664 rd1=b rd2=10 ALURez=0 (wd)
                         b"000_110_100_011_0_101", --or 1a35 rd1=0 rd2=10 ALURez=10 (wd)
                         b"111_0000000001101", --j e00d
                         b"101_001_011_0000111", --ori a587 rd1=0 rd2=7 ExtImm=7 ALURez=7 (wd)
                         b"110_011_100_0000010", --xori ce02 rd1=7 rd2=5 ExtImm=2 ALURez=5 (wd)
                         b"000_011_010_100_0_110", --xor 0d46 rd1=10 rd2=2 ALURez=1b(hexazecimal), 27(zecimal) (wd)
                         others => b"000_100_001_111_0_111"); --nor 10f7 rd1=1b rd2=4 ALURez=FFE4(hexazecimal) (wd)
```

5. RTL Schematic:







6. FUNCȚIONALITATE:

Proiectul a fost testat pe plăcuță, iar conform datelor de intrare, acesta execută corect codul.