Logo

Description automatically generated

**Procesorul MIPS ciclu unic- 16 biti**

**Student: Sandor Alexia**

**An:2**

**Grupa: 30228**

MIPS-ul are ca si componente urmatoarele:

* MPG
* 7segment
* Instruction\_Fetch
* Register\_File
* Instruction Decode
* Mem
* Unit Control UC
* Unit Execute UE

1. ***Cele 4 instructiuni suplimentare***
2. **Tipul R**

**- xor $rd, $rs, $rt =>** RF[rd] 🡨RF[rs] ^ RF[rt] //Exclusive OR

**- slt $rd, $rs,$rt =>** RF[rd] 🡨1 if RF[rt] < RF[rs] else 0 // – Set on less than (signed)

**2. Tipul I**

**- bgt** If(RF[rs] >RF[rt]) then PC 🡨PC + S\_Ext(imm) // Branch Greater than

**- ori** RF[rt] ⇓RF[rs] | Z\_Ext(imm) // Logical OR unsigned constant

1. **Semnale control MIPS16 pentru Anexa 5**

<?> ϵ {\_gez, \_ne, \_gtz}

*Tipuri de operații care se pun în paranteză la ALUOp si ALUCtrl:* {(+), (-), (&), (|), (^), (<<*l*), (<<*lv*), (>>*l*), (>>*a*), (<)}, & - AND, | - OR, ^ *- XOR, l* *- logic, a - aritmetic, v - cu variabilă*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instrucțiune** | **Opcode** *Instr(15-13)* | **RegDst** | **ExtOp** | **ALUSrc** | **Branch** | **<Bgt>** | **Jump** | **JmpR** (opțional) | **MemWrite** | **MemtoReg** | **Reg Write** | **ALUOp (1:0)** | **func**  *Instr(2-0)* | **ALUCtrl (2:0)** |
| ADD | 000 | 1 | X | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 00 (+) | 000 | 000 (+) |
| SUB | 000 | 1 | X | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 00 (-) | 001 | 001(-) |
| SLL | 000 | 1 | X | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 00(<<) | 010 | 010(<<) |
| SRL | 000 | 1 | X | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 00(>>) | 011 | 011(>>) |
| AND | 000 | 1 | X | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 00(and) | 100 | 100(and) |
| OR | 000 | 1 | X | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 00 (or) | 101 | 101(or) |
| XOR | 000 | 1 | X | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 00(xor) | 110 | 110(xor) |
| SLT | 000 | 1 | X | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 00 (slt) | 111 | 111(slt) |
| ADDI | 001 | 0 | 1 | 1 | 0 | 0 | 0 |  | 0 | 0 | 1 | 01 | X | 000 |
| LW | 010 | 0 | 1 | 1 | 0 | 0 | 0 |  | 0 | 1 | 1 | 01 | X | 000 |
| SW | 011 | X | 1 | 1 | 0 | 0 | 0 |  | 1 | 0 | 0 | 01 | X | 000 |
| BEQ | 100 | X | 1 | 0 | 1 | 0 | 0 |  | 0 | 0 | 0 | 10 | X | 001 |
| BGT | 101 | X | 1 | 0 | 0 | 1 | 0 |  | 0 | 0 | 0 | 10 | X | 001 |
| ORI | 110 | 0 | 1 | 1 | 0 | 0 | 0 |  | 0 | 0 | 1 | 11 | X | 110 |
| J | 111 | X | X | X | X | 0 | 1 |  | 0 | 0 | 0 | XX | X | XXX |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. ***Program- memoria ROM***

***Programul prezent in memoria ROM sorteaza un sir de 10 elemente in ordine descrescatoare.***

***Codul in C***

***Programul gaseste minimul dintr-un sir si il ridica la patrat.***

int main() {

int arr[10] = { 8,5,41,7,3,20,52,18,37,10 };

int min = arr[0];

for (int j = 1; j < 10; j++) {

if (arr[j] < min) {

min = arr[j];

}

}

min = min \* min;

return 0;

}

b"001\_000\_001\_0000001"

b"001\_000\_100\_0001001"

b"000\_000\_000\_010\_0\_000"

b"010\_010\_101\_0000000"

b"100\_001\_100\_0000101"

b"001\_010\_010\_0000001"

b"010\_010\_110\_0000000"

b"101\_110\_101\_0000001"

b"010\_010\_101\_0000000

b"001\_001\_001\_0000001"

b"111\_0000100"

b"000\_000\_000\_111\_0\_000"

b"000\_000\_000\_011\_0\_000"

b"100\_111\_101\_0000011"

b"000\_011\_101\_011\_0\_000"

b"001\_111\_111\_0000001"

b"111\_0001010"

b"011\_010\_011\_0000000"

1. Trasarea programului

***0. addi $1, $0, 1 #i=0, contorul buclei RD1=0, Ext\_imm=1, AluRes=1***

***1. addi $4, $0, 10 #se salveaza numarul maxim de iteratii RD1=0, Ext\_imm=10, AluRes=10***

***2. add $2, $0, $0 #initializarea indexului locatiei de memorie RD1=0, Ext\_imm=20, AluRes=0***

***3. lw $5,0($2) #min=arr[0] RD1=0, Ext\_imm=0, AluRes=0***

***4. beq $1, $4, 6 #s-au facut 10 iteratii? Daca da, sari la RD1=i, Ext\_imm=6, AluRes=5***

***5. addi $2,$2,1 #creste indexu*** ***RD1=elem, Ext\_imm=0, AluRes=1***

***6. lw $6, 0($2) #salveaza in $6 urmatorul elem RD1=elem+1, Ext\_imm=1, AluRes=1***

***7. bgt $6, $5, 1 #arr[i]>min? Daca da, sari RD1=i, Ext\_imm=1, AluRes=2***

***8. add $5, $6, $0 #daca nu, salveaza in $5 arr[i] RD1=0, Ext\_imm=4, AluRes=4***

***9. addi $1, $1, 1 #creste contorul RD1=index+1, Ext\_imm=6, AluRes=FFFE***

***10. j 4 #jump RD1=0, Ext\_imm=4, AluRes=4***

***11. add $7, $0, $0 #pune in $7 0 RD1=0, Ext\_imm=0, AluRes=0***

***12. add $3,$0, $0 #pune in $3 valoarea 0 RD1=0, Ext\_imm=0, AluRes=0***

***13. beq $7, $5,3 #este $7=$5? daca da sari RD1=i, Ext\_imm=3, AluRes=min***

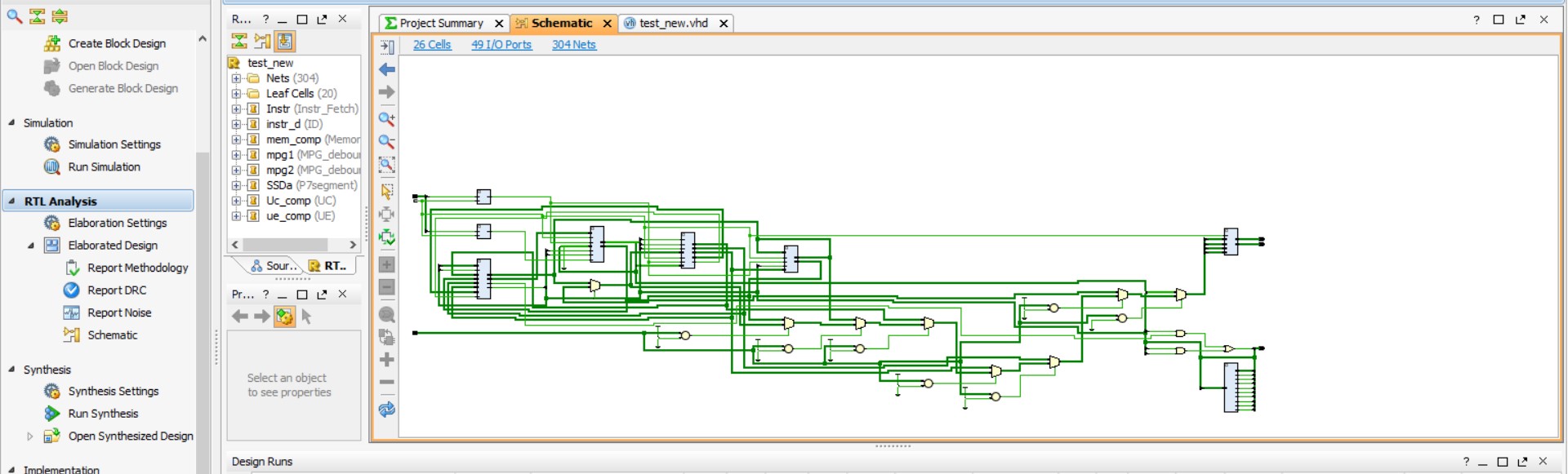
***14. add $3, $3, $5 #daca nu adauga in $3 valoarea lui $5 RD1=3, Ext\_imm=0, AluRes=1***

***15. addi $7, $7,1 #creste contorul RD1=index+1, Ext\_imm=1, AluRes=2***

***16. j 12 #jump RD1=0, Ext\_imm=12, AluRes=0***

***17. sw $3,0($2) #salveaza minimul la patrat RD1=3, Ext\_imm=0, AluRes=3***

1. **Nu exista parti din procesor incomplete din laboratoarele 4-7.**
2. **Schema**

****

1. **Programul a fost testat pe placa, functioneaza doar daca cel mai mic element este ultimul sau primul**

**h)** 