**Procesorul MIPS, ciclu unic – versiune pe 16 biți**

Student: Vesa Bianca

Grupa: 30227

1. **Instructiuni alese**

Pe langa setul minimal cu cele 11 instructiuni impuse in laboratorul 4, procesorul implementat mai executa inca doua instructiuni de tip R (xor si nand), si doua de tip I (bne si xori), care au urmatoarele descrieri RTL:

* xor $rd, $rs, $rt | RF[rd] <- RF[rs] ^ RF[rt]
* nand $rd, $rs, $rt | RF[rd]<- RF[rs] nand RF[rt]
* bne $rt, $rs, imm | if ( RF[rs] != RF[rt] ) then PC<-PC + S\_ext(imm)
* xori $rd, $rs, imm | RF[rt] <- RF[rs] ^ Z\_ext(imm)

**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** | **Br<?>**(opțional) | **Jump** | **JmpL**(opțional) | **MemWrite** | **MemtoReg** | **RegWrite** | **ALUOp(2:0)** | **func***Instr(2-0)* | **ALUCtrl(2:0)** | **JmpR** (opțional) |
| Add | 000 | 1 | X | 0 | 0 |  | 0 |  | 0 | 0 | 1 | 000(+) | 000 | 000(+) |  |
| Sub | 000 | 1 | X | 0 | 0 |  | 0 |  | 0 | 0 | 1 | 001(-) | 001 | 001(-) |  |
| Sll | 000 | 1 | X | 0 | 0 |  | 0 |  | 0 | 0 | 1 | 010(<<l) | 010 | 010(<<l) |  |
| Srl | 000 | 1 | X | 0 | 0 |  | 0 |  | 0 | 0 | 1 | 011(>>l) | 011 | 011(>>l) |  |
| And | 000 | 1 | X | 0 | 0 |  | 0 |  | 0 | 0 | 1 | 100(&) | 100 | 100(&) |  |
| Or | 000 | 1 | X | 0 | 0 |  | 0 |  | 0 | 0 | 1 | 101(|) | 101 | 101(|) |  |
| Xor | 000 | 1 | X | 0 | 0 |  | 0 |  | 0 | 0 | 1 | 110(^) | 110 | 110(^) |  |
| Nand | 000 | 1 | X | 0 | 0 |  | 0 |  | 0 | 0 | 1 | 111(nand) | 111 | 111(nand) |  |
| Addi | 001 | 0 | 1 | 1 | 0 |  | 0 |  | 0 | 0 | 1 | 001(+) | X | 000(+) |  |
| Lw | 010 | 0 | 1 | 1 | 0 |  | 0 |  | 0 | 1 | 1 | 010(-) | X | 000(+) |  |
| Sw | 011 | X | 1 | 1 | 0 |  | 0 |  | 1 | 0 | 0 | 011(+) | X | 000(+) |  |
| Beq | 100 | X | 1 | 0 | 1 |  | 0 |  | 0 | 0 | 0 | 100(-) | X | 001(-) |  |
| Bne | 101 | X | 1 | 1 | 1 | <NE> | 0 |  | 0 | 0 | 0 | 101(-) | X | 001(-) |  |
| Xori | 110 | 0 | 1 | 1 | 0 |  | 0 |  | 0 | 0 | 1 | 110(^) | X | 110(^) |  |
| J | 111 | X | X | X | X |  | 1 |  | 0 | 0 | 0 | 111 | X | X |  |

1. **Program – memorie ROM**

Programul din memoria ROM calculeaza al zecelea numar din sirul lui Fibonacci, il stocheaza in memoria RAM, la adresa 4, cu ajutorul instructiunii sw, dupa care verifica daca acesta a fost scris, cu ajutorul instructiunii lw.

Codul C corespunzator programului:

*int first = 0, second = 1, i=0, next, result;*

*int n=9; // al 10-lea numar din sir*

*do*

*{*

*next = first + second;*

*first = second;*

*second = next;*

*result = next;*

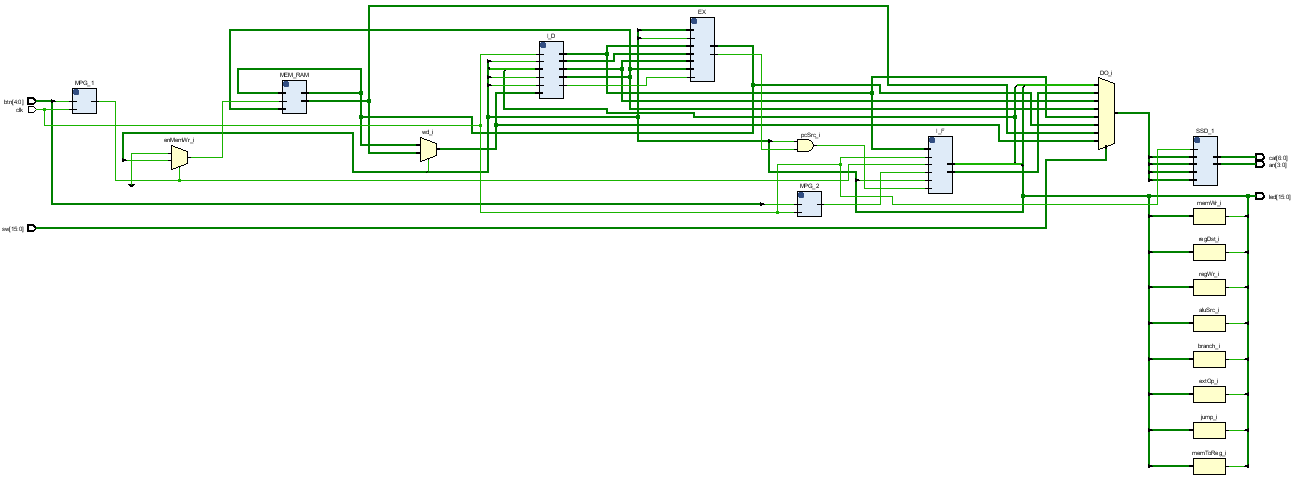
*} while ( i != n)*

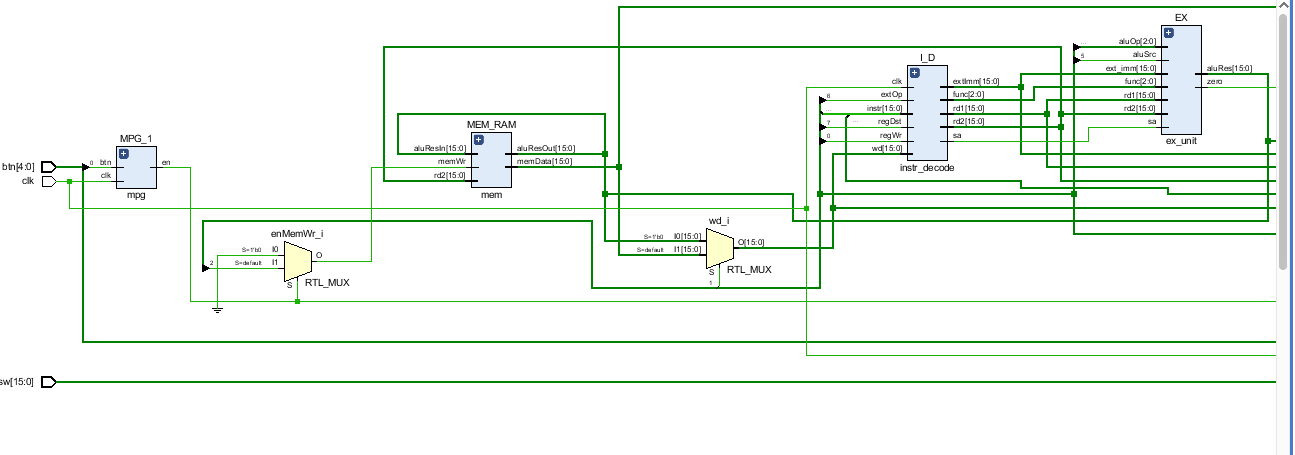
*printf(“%d”, result);*

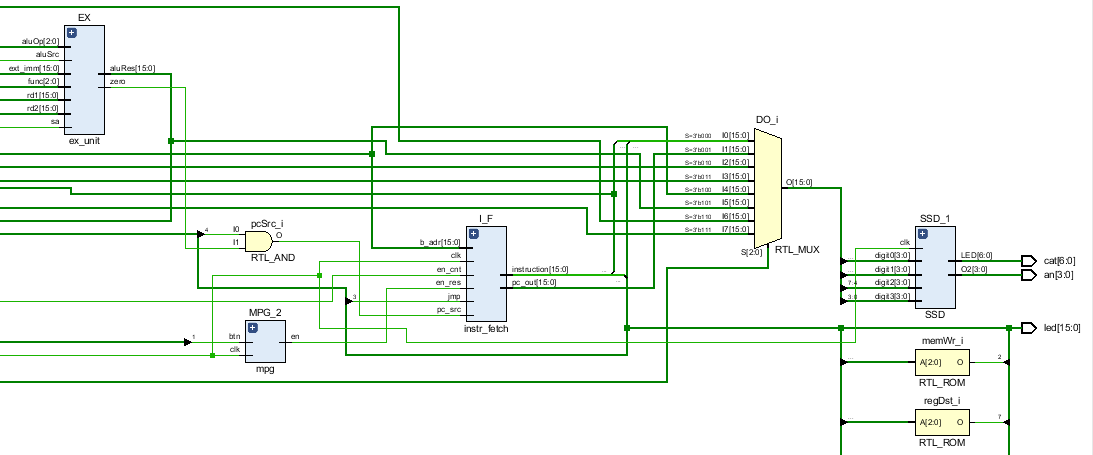
1. **Trasarea programului**
2. addi $1, $0, 0 RD1 = 0, RD2 = 0, Ext\_Imm = 0, ALURes = 0, memData = 0, WD = 0
3. addi $2, $0, 1 RD1 = 0, RD2 = 1, Ext\_Imm = 1, ALURes = 1, memData = 0, WD = 1
4. addi $7, $0, 9 RD1 = 0, RD2 = 9, Ext\_Imm = 9, ALURes = 9, memData = 0, WD = 9
5. and $3, $0, $0 RD1 = 0, RD2 = 0, Ext\_Imm = x, ALURes = 0, memData = 0, WD = 0
6. add $4, $1, $2 RD1 = 0, RD2 = 1, Ext\_Imm = x, ALURes = 1, memData = 0, WD = 1
7. add $1, $0, $2 RD1 = 0, RD2 = 1, Ext\_Imm = x, ALURes = 1, memData = 0, WD = 1
8. add $2, $0, $4 RD1 = 0, RD2 = 1, Ext\_Imm = x, ALURes = 1, memData = 0, WD = 1
9. add $6, $0, $3 RD1 = 0, RD2 = 0, Ext\_Imm = x, ALURes = 0, memData = 0, WD = 0
10. addi $3, $6, 1 RD1 = 0, RD2 = 1, Ext\_Imm = 1, ALURes = 1, memData = 0, WD = 1
11. beq $3, $7, 1 RD1 = 9, RD2 = 1, Ext\_Imm = 1, ALURes = 8, memData = 0, WD = 8
12. j 4 RD1 = x, RD2 = x, Ext\_Imm = x, ALURes = x, memData = x, WD = x
13. xor $5, $0, $4 RD1 = 0, RD2 = 55 (x37), Ext\_Imm = x, ALURes = 55(x37), memData = 0, WD = 55(x37)
14. addi $6, $0, 4 RD1 = 0, RD2 = 4, Ext\_Imm = 4, ALURes = 4, memData = 0, WD = 4
15. sw $5, 0($6) RD1 = 4, RD2 = 55(x37), Ext\_Imm = 0, ALURes = 4, memData = 55(x37), WD = 4
16. addi $4, $0, 0 RD1 = 0, RD2 = 0, Ext\_Imm = 0, ALURes = 0, memData = 0, WD = 0
17. lw $1, 0($6) RD1 = 4, RD2 = 55(x37), Ext\_Imm = 0, ALURes = 4, memData = 55(x37), WD = 55(x37)
18. j 0 RD1 = x, RD2 = x, Ext\_Imm = x, ALURes = x, memData = x, WD = x
19. Nu exista parti lipsa din procesor, sau activitati incomplete din laboratoarele anterioare
20. **RTL schematic**

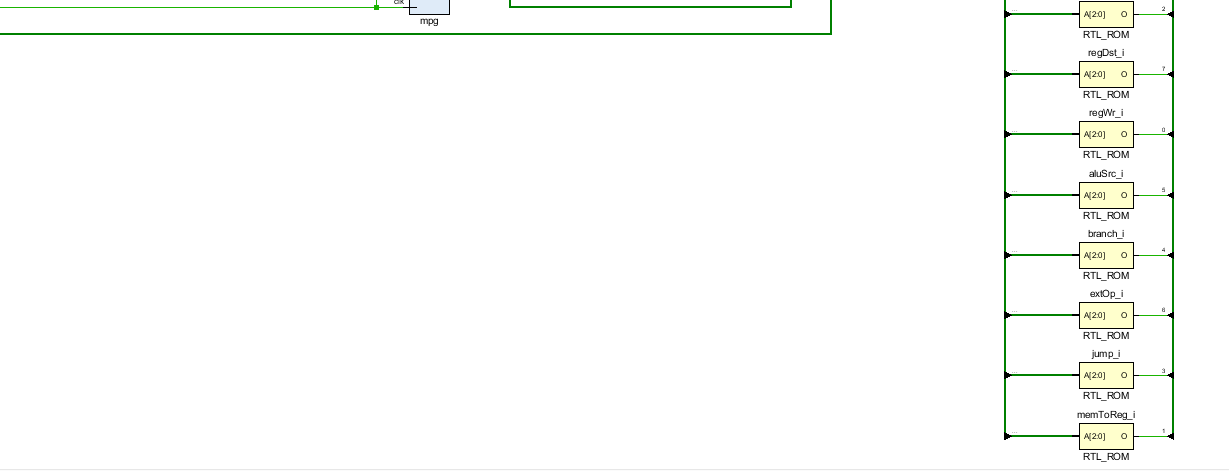
In cele ce urmeaza voi atasa imaginile cu RTL schematic pentru MIPS 16, atat partea de ansamblu (in intregime si pe bucati, pentru ca schema este prea mare pentru a se vedea clar), cat si fiecare componenta in parte.

MIPS 16:

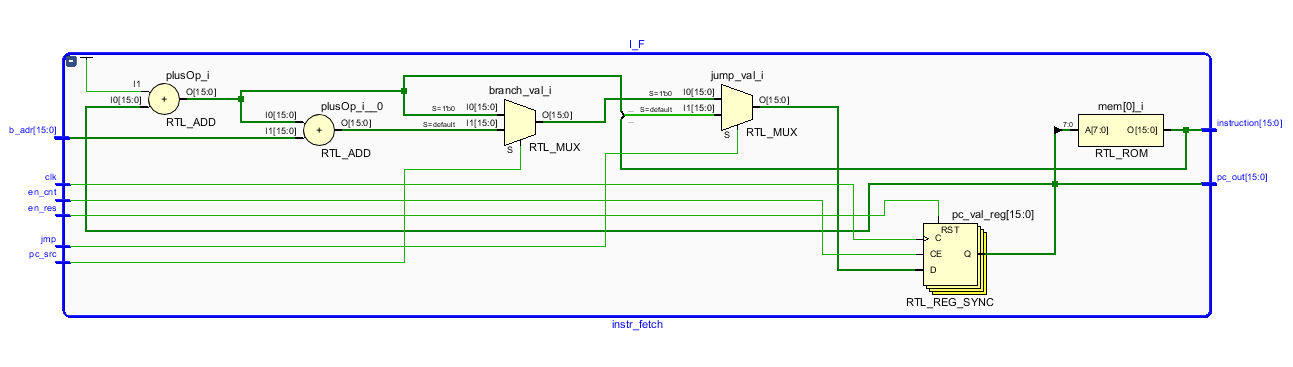




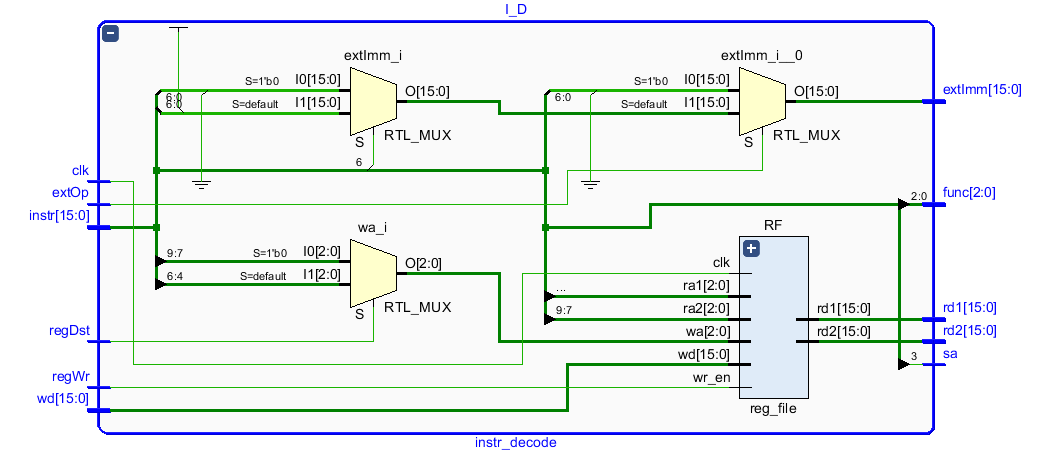




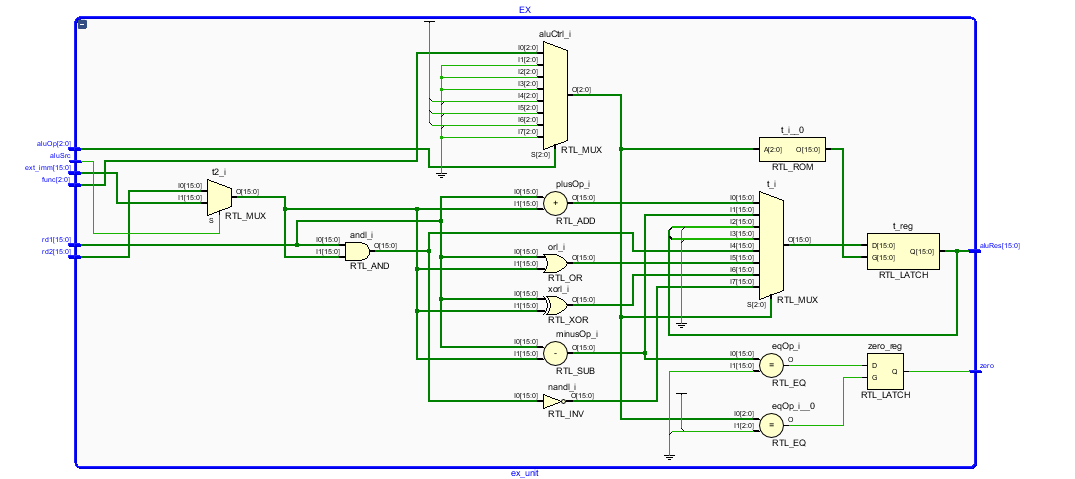
Instruction fetch:



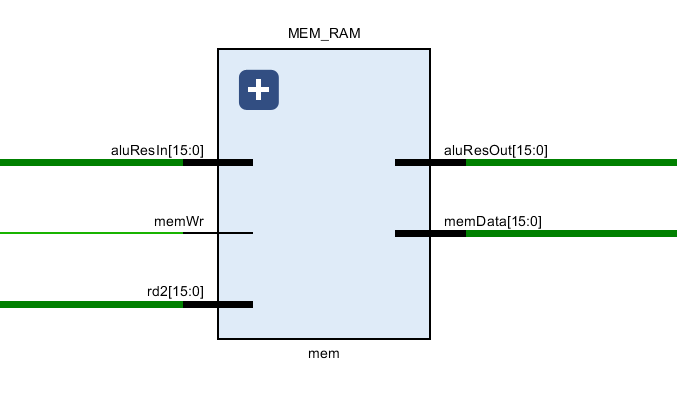
Instruction decode:



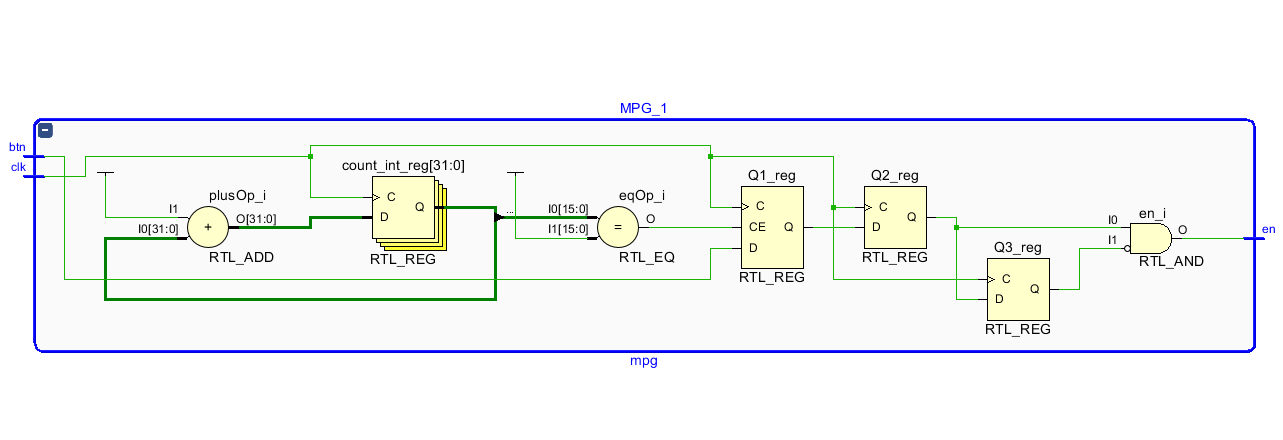
Execution unit:



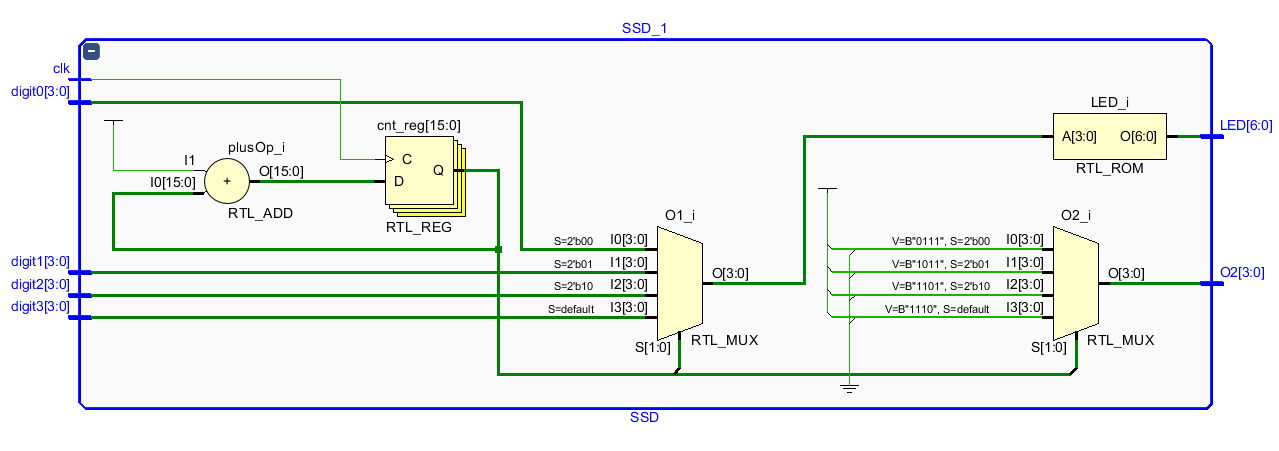
Memory block:



Monopulse generator:



Seven segment display:



1. **Testare**

Microprocesorul a fost testat pe placa Basys 3, verificand la fiecare iteratie a programului din memorie toate valorile semnalelor afisate pe SSD si pe LED-uri si este complet functional.