RAPORT MIPS 16 PIPELINE

STUDENT : VESA BIANCA

GRUPA : 30227

# Registre pipeline

Pentru transformarea procesorului MIPS 16 cu ciclu unic într-un procesor de tip pipeline am definit registrele de pipeline, sectionand componentele MIPS-ului cu ciclu unic implementat anterior (IF/ID, ID/EX, EX/MEM, MEM/WB). De asemenea, am efectuat modificarile necesare fiecarei componente, pentru a corespunde noii scheme pentru calea de date + control.

Mai jos se afla descrierea fiecarui registru:

**Configurare registreMIPS16 Pipeline – varianta 1**

Se introduc pe coloane semnalele de date și control mapate la registre, de sus în jos, începând de la biții cei mai semnificativi ai registrului către cei mai puțin semnificativi. Se introduc în paranteză biții din registru alocați pentru fiecare semnal în parte. În dreptul numelui registrelor din primul rând se introduce în paranteză poziția bitului cel mai semnificativ (<msb>) din care reiese dimensiunea totală alocată registrului.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **REG\_IF\_ID(31**– **0)** | **REG\_ID\_EX(81**– **0)**   |  |  |  |  | | --- | --- | --- | --- | | ID/EX\_IN(72-0) | WB(1-0) | M(1-0) | EX(4-0) | | **REG\_EX\_MEM(55**– **0)**   |  |  |  | | --- | --- | --- | | EX/MEM\_IN(51-0) | WB(1-0) | M(1-0) | | **REG\_MEM\_WB(36**– **0)**   |  |  | | --- | --- | | MEM/WB\_IN(34-0) | WB(1-0) | |
| PC + 1(31– 16) | PC + 1(72 - 57) memToReg(1) memWr(1) aluOp(4-2) | br\_address(51-36) memToReg(1) memWr(1) | memData(34-19) memToReg(1) |
| Instruction(15 – 0) | RD1(56 – 41) regWr(0) branch(0) aluSrc(1) | zero(35) regWr(0) branch(0) | aluRes(18-3) regWr(0) |
|  | RD2(40 – 25) regDst(0) | aluRes(34-19) | write\_add(2-0) |
|  | ext\_imm(24 – 9) | RD2(18-3) |  |
|  | function(8 – 6) | write\_add(2-0) |  |
|  | RT(5 – 3) |  |  |
|  | RD(2 – 0) |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

URL: <https://drive.google.com/open?id=1Yw18tfjFjo-v897vVpd8KNGnpx5U8i9n>

# Identificarea hazardurilor

Programul executat de procesorul MIPS ciclu unic este urmatorul:

|  |
| --- |
| 1. addi $1, $0, 0 |
| 1. addi $2, $0, 1 |
| 1. addi $7, $0, 9 |
| 1. and $3, $0, $0 |
| 1. add $4, $1, $2 |
| 1. add $1, $0, $2 |
| 1. add $2, $0, $4 |
| 1. add $6, $0, $3 |
| 1. addi $3, $6, 1 |
| 1. beq $3, $7, 1 |
| 1. j 5 |
| 1. xor $5, $0, $4 |
| 1. addi $6, $0, 4 |
| 1. sw $5, 0($6) |
| 1. addi $4, $0, 0 |
| 1. lw $1, 0($6) |
| 1. j 0 |

In urma realizarii diagramei de pipeline, prezenta in documentul cu numele *Identificarea Hazardurilor* am identificat cele 3 tipuri de hazarduri care apar in program.

1. Hazard structural

Am identificat acest tip de hazard la nivelul perechilor de instructiuni care incearca sa scrie in, respectiv citeasca din acelasi registru, in aceeasi perioada de ceas:

* Ciclu de ceas 5 : instructiunea 1 scrie in $2, instructiunea 4 citeste din $2
* Ciclu de ceas 17 : instructiunea 12 scrie in $6, instructiunea 15 citeste din $6

1. Hazard de date

Hazardul de date RAW este prezent in instrucțiunile care au ajuns în etajul ID , iar operanzii sursă sunt în curs de calculare în etajele următoare de pipeline (de către instrucțiuni anterioare):

* Instructiunea 6 are ca operand sursa $4, in ciclul de ceas 8, a carui valoare va fi calculata de instructiunea 4 in ciclul de ceas 9
* Instructiunea 8 are ca operand sursa $6 in ciclul de ceas 10, a carui valoare va fi calculata de instructiunea 7 in ciclul de ceas 12
* Instructiunea 9 are ca operand sursa $3 in ciclul de ceas 11, a carui valoare va fi calculata de instructiunea 8 in ciclul de ceas 13
* Instructiunea 13 are ca operand sursa $6 in ciclul de ceas 15, a carui valoare va fi calculata de instructiunea 12 in ciclul de ceas 17

1. Hazard de control

Acest tip de hazard se gaseste la nivelul instructiunilor de salt (beq si j).

Pentru instructiunea beq vor intra implicit în execuție următoarele 3 instrucțiuni => Am inserat 3 NoOp in continuarea instructiunii

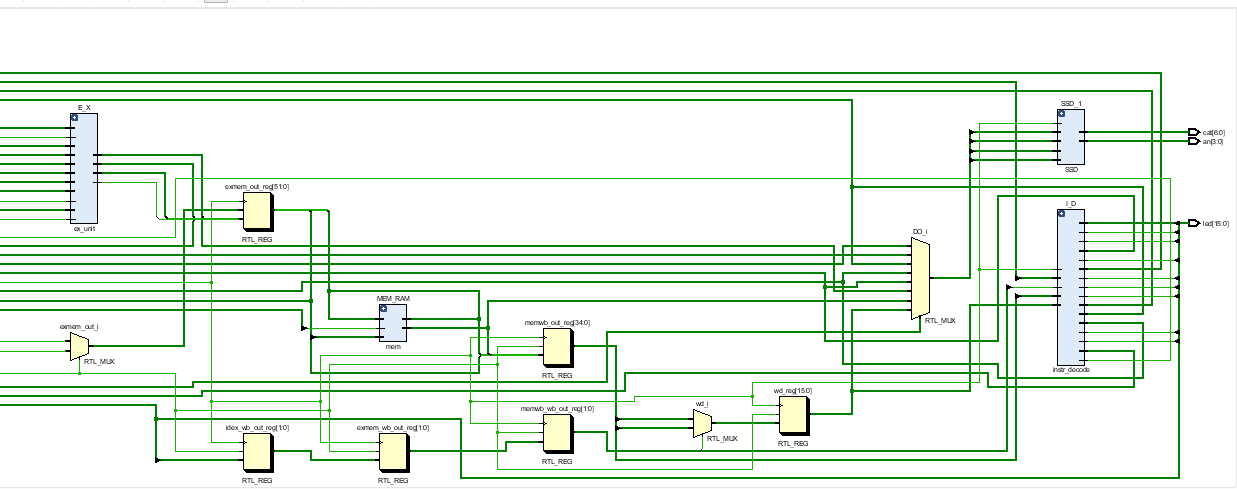
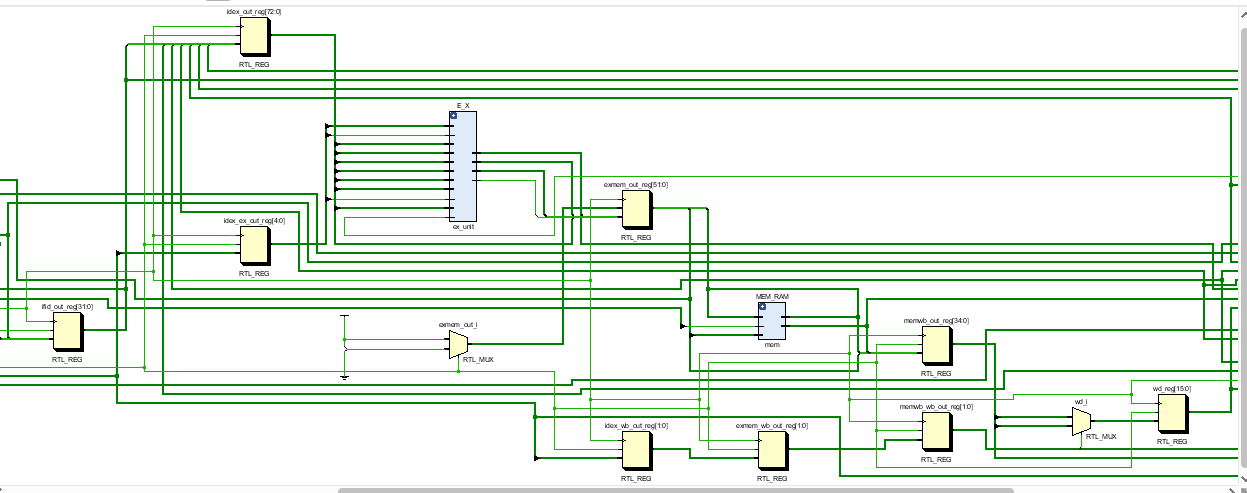
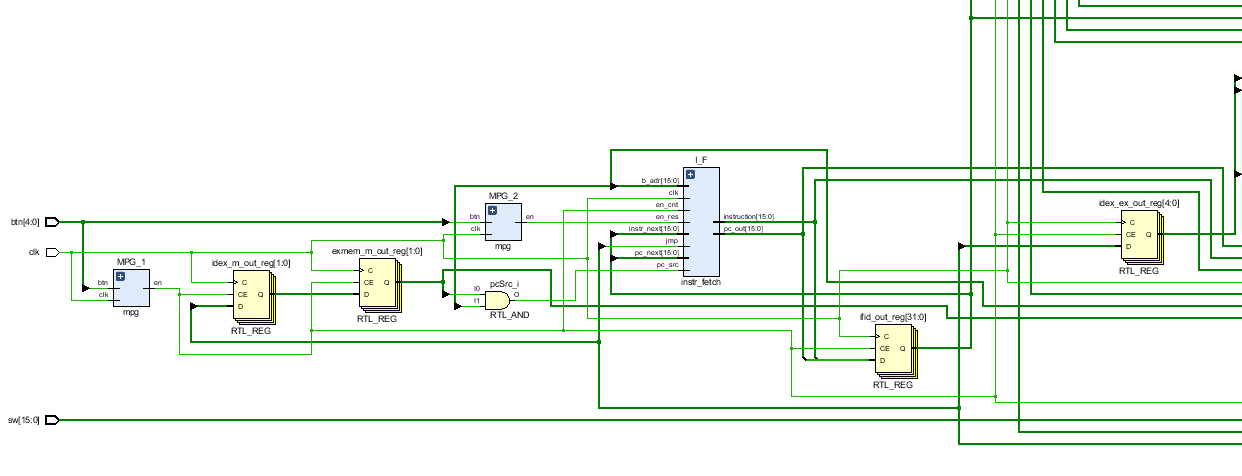
Instructiunile j executa saltul când sunt în etajul ID, ceea ce inseamna că intră în execuție doar instrucțiunea imediat următoare => Am inserat un NoOp in continuarea instructiunii

Programul rezultat in urma inserarii NoOp necesare este urmatorul, iar diagrama pipeline modificata este in documentul *Rezolvare Hazarduri*:

|  |
| --- |
| 0. addi $1, $0, 0 |
| 1.addi $2, $0, 1 |
| 2.addi $7, $0, 9 |
| 3.and $3, $0, $0 |
| 4.and $0, $0, $0 |
| 5.add $4, $1, $2 |
| 6.add $1, $0, $2 |
| 7.and $0, $0, $0 |
| 8.and $0, $0, $0 |
| 9.add $2, $0, $4 |
| 10.add $6, $0, $3 |
| 11.and $0, $0, $0 |
| 12.and $0, $0, $0 |
| 13.and $0, $0, $0 |
| 14.addi $3, $6, 1 |
| 15.and $0, $0, $0 |
| 16.and $0, $0, $0 |
| 17.and $0, $0, $0 |
| 18.beq $3, $7, 5 |
| 19.and $0, $0, $0 |
| 20.and $0, $0, $0 |
| 21.and $0, $0, $0 |
| 22.j 5 |
| 23.and $0, $0, $0 |
| 24.xor $5, $0, $4 |
| 25addi $6, $0, 4 |
| 26.and $0, $0, $0 |
| 27and $0, $0, $0 |
| 28.and $0, $0, $0 |
| 29.sw $5, 0($6) |
| 30.addi $4, $0, 0 |
| 31.lw $1, 0($6) |
| 32.j 0 |
| 33.and $0, $0, $0 |

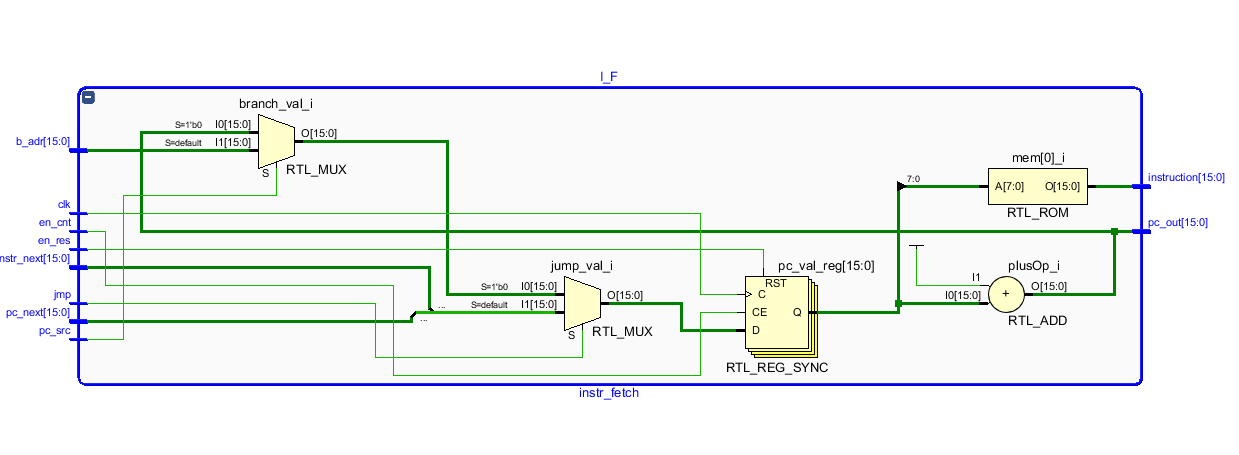
# RTL schematic

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, pentru a vedea modificarile care apar fata de MIPS ciclu unic.

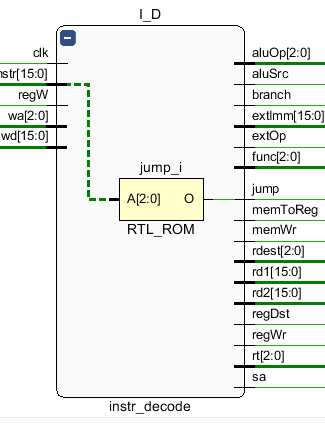


MIPS 16

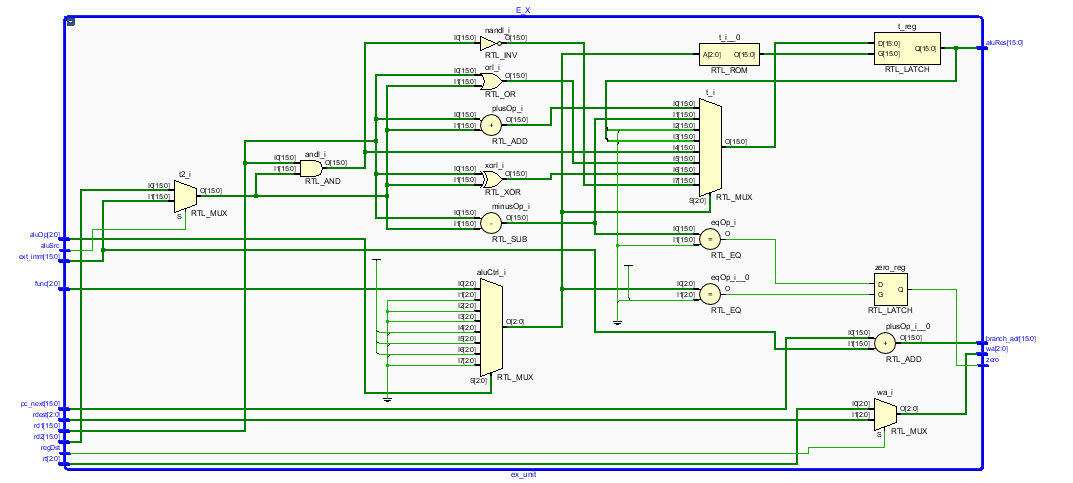
INSTRUCTION FETCH



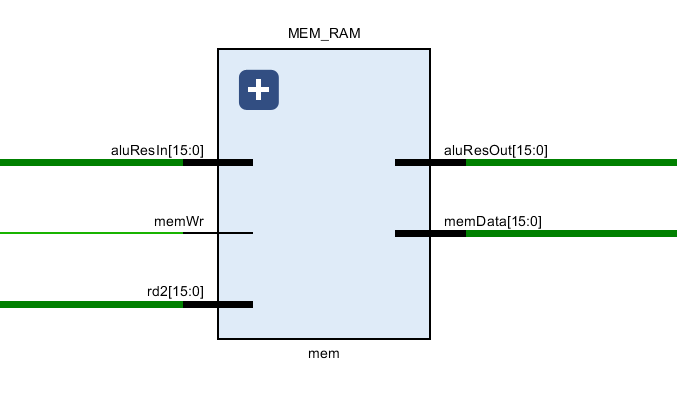
INSTRUCTION DECODE



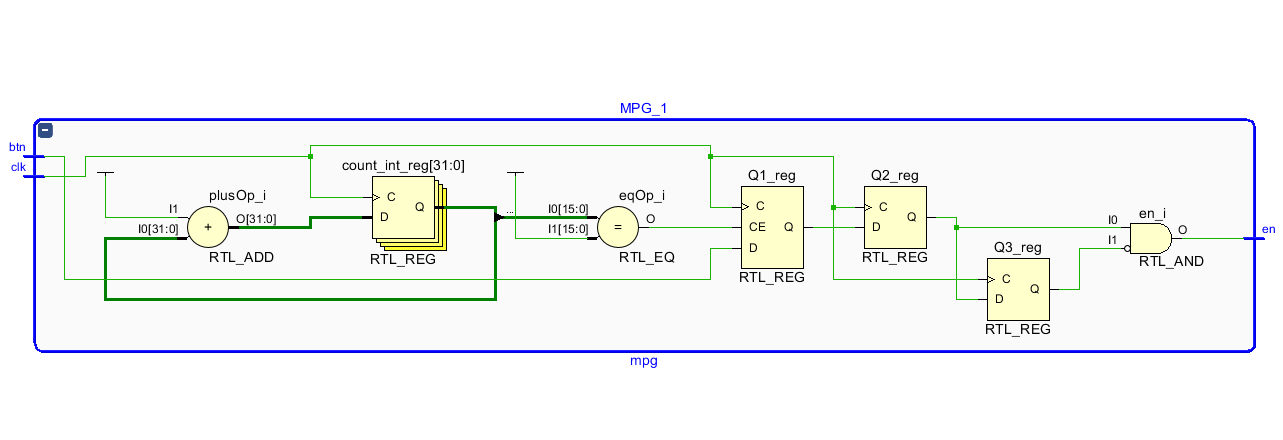
EXECUTION UNIT



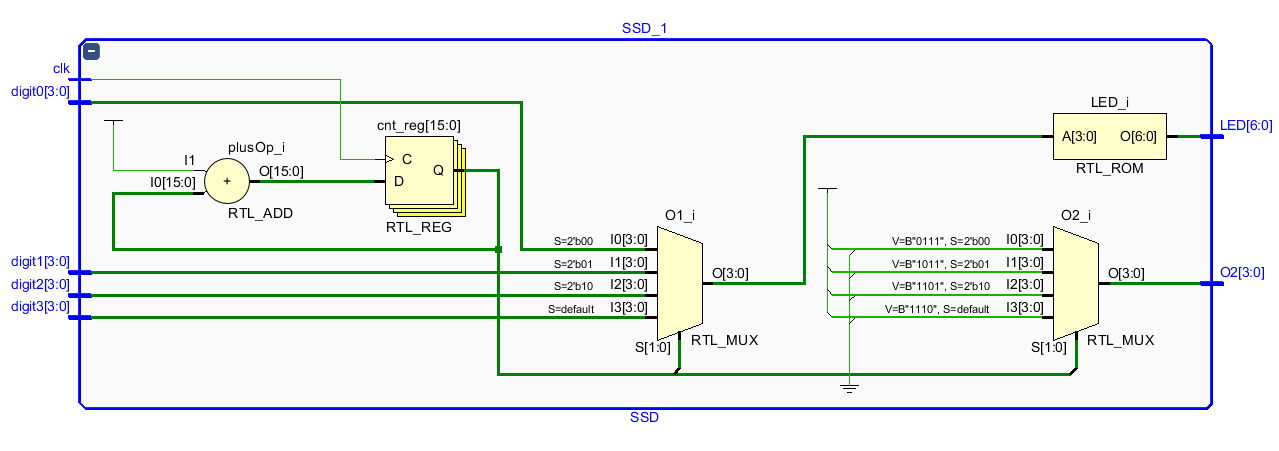
MEMORY BLOCK



MONOPULSE GENERATOR



SEVEN SEGMENT DISPLAY



# 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.