PROCESORUL MIPS

RAPORT

Nume student: Molnár Zsanett-Ingrid

Grupa: 302210

CUPRINS

1. Instrucțiuni…………………………………………………………………3

2. Valorile semnalelor de control…………………………………………….6

3. Descrierea programului ales……………………………………………….6

4. Trasarea execuției programului....................................................................8

5. Schema procesorului MIPS pe 16 biți..........................................................8

6. Concluzii......................................................................................................9

**1. Instrucțiuni:**

* **Instrucțiuni de tip R:**

ADD: $d $s + $t; PC  PC + 1;

add $d, $s, $t

000 sss ttt ddd 0 000

Ex. add $1, $2, $5

000 010 101 001 0 000

SUB: $d $s - $t; PC PC + 1;

sub $d, $s, $t

000 sss ttt ddd 0 001

Ex. sub $1, $5, $2

000 101 010 001 0 001

SLL: $d $t << h; PC PC + 1;

sll $d, $t, h

000 sss ttt ddd h 010

Ex. sll $1, $3, 1

000 000 011 001 1 010

SRL: $d $t >> h; PC PC + 1;

srl $d, $t, h

000 000 ttt ddd h 011

Ex. srl $1, $2, 1

000 000 010 001 1 011

AND: $d $s & $t; PC PC + 1;

and $d, $s, $t

000 sss ttt ddd 0 100

Ex. and $2, $1, $4

000 001 100 010 0 100

OR: $d $s | $t; PC PC + 1;

or $d, $s, $t

000 sss ttt ddd 0 101

Ex. or $3, $2, $1

000 010 001 011 0 101

XOR: $d $s ^ $t; PC PC + 4;

xor $d, $s, $t

000 sss ttt ddd 0 110

Ex. xor $1, $2, $3

000 010 011 001 0 110

SLLV: $d $t << $s; PC PC + 4;

sllv $d, $t, $s

000 sss ttt ddd 0 110

Ex. sllv $2, $1, $5

000 101 001 010 0 111

* **Instrucțiuni de tip I:**

ADDI: $t $s + imm; PC PC + 1;

addi $t, $s, imm

001 sss ttt iiiiiii

Ex. addi $2, $1, 3

001 001 010 0000011

LW: $t MEM[$s + offset]; PC PC + 1;

lw $t, offset($s)

010 sss ttt iiiiiii

Ex. lw $1, 4($3)

010 011 001 0000100

SW: MEM[$s + offset] $t; PC PC + 1;

sw $t, offset($s)

011 sss ttt iiiiiii

Ex. sw $5, 3($2)

011 010 101 0000011

BEQ: if $s == $t then PC PC + 1 + (offset << 2); else PC PC + 1;

beq $s, $t, offset

100 sss ttt iiiiiii

Ex. beq $1, $3, 3

100 001 011 0000011

BGTZ: if $s > 0 PC PC + 4 + (offset << 2); else PC PC + 4;

bgtz $s, offset

101 sss 000 iiiiiii

Ex. bgtz $2, 6

101 010 000 0000110

BNE: if $s != $t PC PC + 1 + (offset << 2); else PC PC + 1;

bne $s, $t, offset

110 sss ttt iiiiiii

Ex. bne $2, $3, 4

110 010 011 0000100

* **Instrucțiuni de tip J:**

J: PC ((PC +4) & 0xf0000000) | (target << 2);

j target

111 iiiiiii

Ex. j 1

111 0000000000001

**2. Valorile semnalelor de control:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | Reg  dest | Reg  write | Alu  src | Ext  Op | Alu  Ctrl | Mem  write | Mem  to reg | Branch  equal | Branch  not equal | Branch  greater  than zero | Jump |
| ADD | 1 | 1 | 0 | X | 00 | 0 | 0 | 0 | 0 | 0 | 0 |
| SUB | 1 | 1 | 0 | X | 00 | 0 | 0 | 0 | 0 | 0 | 0 |
| SLL | 1 | 1 | 0 | X | 00 | 0 | 0 | 0 | 0 | 0 | 0 |
| SRL | 1 | 1 | 0 | X | 00 | 0 | 0 | 0 | 0 | 0 | 0 |
| AND | 1 | 1 | 0 | X | 00 | 0 | 0 | 0 | 0 | 0 | 0 |
| OR | 1 | 1 | 0 | X | 00 | 0 | 0 | 0 | 0 | 0 | 0 |
| XOR | 1 | 1 | 0 | X | 00 | 0 | 0 | 0 | 0 | 0 | 0 |
| SLLV | 1 | 1 | 0 | X | 00 | 0 | 0 | 0 | 0 | 0 | 0 |
| ADDI | 0 | 1 | 1 | 1 | 01 | 0 | 0 | 0 | 0 | 0 | 0 |
| LW | 0 | 1 | 1 | 1 | 01 | 0 | 1 | 0 | 0 | 0 | 0 |
| SW | X | 0 | 1 | 1 | 01 | 1 | X | 0 | 0 | 0 | 0 |
| BEQ | X | 0 | 0 | 1 | 10 | 0 | X | 1 | 0 | 0 | 0 |
| BGTZ | X | 0 | 0 | 1 | 10 | 0 | X | 0 | 0 | 1 | 0 |
| BNE | X | 0 | 0 | 1 | 10 | 0 | X | 0 | 1 | 0 | 0 |
| J | X | 0 | X | X | 11 | 0 | X | X | X | X | 1 |

**3. Descrierea programului ales:**

Programul ales de mine calculează suma elementelor diferite de 3 dintr-un vector dat. Vectorul dat este A[10], și conține 10 numere întregi.

int A[10]={1,3,2,4,3,1,7,3,9,2};

int sum=0;

for(int i=0; i<n; i++)

{

if(A[i]!=3)

{

sum=sum+A[i];

}

}

Programul este constituit dintr-un for, care parcurge fiecare element din vector. În interiorul for-ului, pentru fiecare element se testează condiția de a fi diferit de 3 cu ajutorul unui if. Dacă elementul curent este diferit de 3, se adună la suma declarată și inițializată la începutul programului. În cazul în care elementul este egal cu 3, acesta nu se adaugă la sumă.

Pentru ca programul nostru să poată fi pus pe plăcuța FPGA, acesta trebuie tradus în cod de asamblare, apoi în cod mașină.

add $1, $0, $0

addi $4, $0, 10

addi $6, $0, 3

add $2, $0, $0

add $5, $0, $0

begin\_loop: beq $1, $4, end\_loop

lw $3, A\_addr($2)

beq $3, $6, next

add $5, $5, $3

next: addi $2, $2, 1

addi $1, $1, 1

j begin\_loop

end\_loop: sw $5, sum\_addr($0)

Cod de asamblare: Cod mașină:

add $1, $0, $0 000 000 000 001 0 000 => 0010

addi $4, $0, 10 001 000 100 0001010 => 220A

addi $6, $0, 3 001 000 010 0000011 => 2103

add $2, $0, $0 000 000 000 010 0 000 => 0020

add $5, $0, $0 000 000 000 101 0 000 => 0050

beq $1, $4, 6 100 001 100 0000110 => 8606

lw $3, 10($2) 010 010 011 0001010 => 498A

beq $3, $6, 1 100 011 110 0000001 => 8F01

add $5, $5, $3 000 101 011 101 0 000 => 15D0

addi $2, $2, 1 001 010 010 0000001 => 2901

addi $1, $1, 1 001 001 001 0000001 => 2481

j 5 111 0000000000101 => E005

sw $5, 20($0) 011 000 101 0010100 => 6294

**4. Trasarea execuției programului:**

add $1, $0, $0 RD1=0, RD2=0, Ext\_imm=0, AluRes=0

addi $4, $0, 10 RD1=0, RD2=0, Ext\_imm=10, AluRes=10

addi $6, $0, 3 RD1=0, RD2=0, Ext\_imm=3, AluRes=3

add $2, $0, $0 RD1=0, RD2=0, Ext\_imm=0, AluRes=0

add $5, $0, $0 RD1=0, RD2=0, Ext\_imm=0, AluRes=0

beq $1, $4, 6 RD1=0, RD2=10, Ext\_imm=0, AluRes=0

lw $3, 10($2) RD1=2, RD2=3, Ext\_imm=0, AluRes=0, wd=1, mem\_data=1 (A[0])

beq $3, $6, 1 RD1=1, RD2=3, Ext\_imm=1, AluRes=0

add $5, $5, $3 RD1=0, RD2=1, Ext\_imm=0, AluRes=1

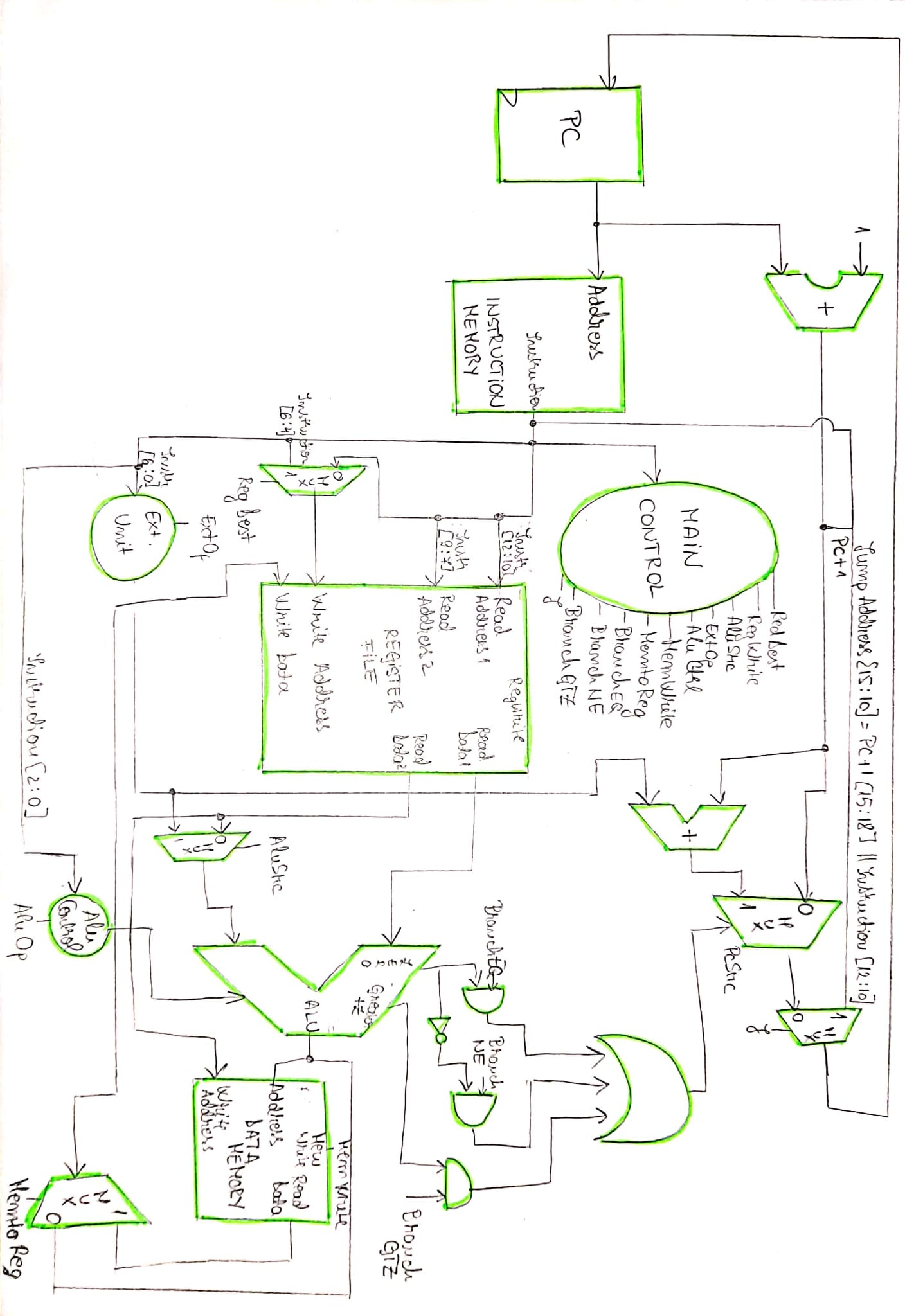
addi $2, $2, 1 RD1=0, RD2=0, Ext\_imm=1, AluRes=1

addi $1, $1, 1 RD1=0, RD2=0, Ext\_imm=1, AluRes=1

j 5 RD1=0, RD2=0, Ext\_imm=5, AluRes=0

sw $5, 20($0) RD1=0, RD2=5, Ext\_imm=0, AluRes=0

**5. Schema procesorului MIPS pe 16 biți:**



**6. Concluzii:**

În ceea ce privește descrierea procesorului MIPS în VHDL, toate componentele acestuia au fost implementate. Proiectul a fost testat pe placa FPGA și este parțial funcțional. Bucla princpală a programului se oprește după 10 iterații, dar nu execută al doilea branch equal (BEQ) în cazul în care se găsește un element egal cu 3.