Procesorul MIPS ciclu unic

(versiune pe 16 biți)

-RAPORT-

Nume: Buda Andreea-Rodica

Grupa: 30221

1. Operații suplimentare

- Exclusive OR (XOR)
  - Operație de tip R.
  - Operație logică prin care se salvează în $rd rezultatul XOR-ului logic între registrele $rs și $rt.
  - Assembly: xor $rd, $rs, $rt

- Set on Less Than (SLT)
  - Operație de tip R.
  - Compară conținutul a două registre, $rs și $rt, și pune rezultatul comparării în registrul $rd.
  - Assembly: slt $rd, $rs, $rt

- Set on Less Than Immediate (SLTI)
  - Operație de tip I.
  - Compară conținutul unui registru, $rs, cu o valoare imediată (imm) și pune rezultatul comparării în registru $rt.
  - Assembly: slti $rt, $rs, imm

- Logical AND unsigned constant (ANDI)
  - Operație de tip I.
  - Operație care face AND între registrul $rs și valoarea imediatei extinse, și salvează rezultatul în registrul $rt.
  - Assembly: andi $rt, $rs, imm

2. Tabel cu semnale de control

Semnale control MIPS16 pentru Anexa 5

Instrucțiune	**Opcode** *Instr(15-13)*	**RegDst**	**ExtOp**	**ALUSrc**	**Branch**	**Jump**	**Mem Write**	**Memto Reg**	**Reg Write**	**ALUOp (2:0)**	**func** *Instr(2-0)*	**ALUCtrl (2:0)**
ADD	000	1	0	0	0	0	0	0	1	000	000	000(+)
SUB	000	1	0	0	0	0	0	0	1	000	001	001(-)
SLL	000	1	0	0	0	0	0	0	1	000	010	010(<<l)
SRL	000	1	0	0	0	0	0	0	1	000	011	011(>>L)
AND	000	1	0	0	0	0	0	0	1	000	100	100(AND)
OR	000	1	0	0	0	0	0	0	1	000	101	101(OR)
XOR	000	1	0	0	0	0	0	0	1	000	110	110(XOR)
SLT	000	1	0	0	0	0	0	0	1	000	111	111(a)
ADDI	001	0	1	1	0	0	0	0	1	001	-	000(+)
LW	010	0	1	1	0	0	0	1	1	010	-	000(+)
SW	011	0	1	1	0	0	1	0	0	011	-	000(+)
BEQ	100	0	1	0	1	0	0	0	0	100	-	010(SUB)
ANDI	101	0	0	1	0	0	0	0	1	101	-	100(I-AND)
SLTI	110	0	0	1	0	0	0	0	1	110	-	111(a)
J	111	0	0	0	0	1	0	0	0	X	-	X

3. Program executat de procesor

Programul strochează în memoria RAM un șir de n numere naturale. Se face suma a doua câte două numere consecutive din șir și se determină cea mai mare sumă dintre toate. La final se afișează această sumă.

❖ Pseudocod

```
1 max=0
2 suma=0
3 i=1
4 n=6
5 v[n] = {2,3,4,5,2,4}
6 max=v[0]+v[1]
7 for  i = 2 to n
8      suma = v[i] + v[i + 1]
9      if suma > max then
10          max = suma
11     end if
12     i=i+1
13 end for
```

❖ Codul mașină al instrucțiunilor

```
--zona declarativa
B"001_000_001_0000000", --ADDI $1,$0,0  (int max=0) #2080
B"001_000_010_0000000", --ADDI $2,$0,0  (int suma=0) #2100
B"001_000_011_0000001", --ADDI $3,$0,1  (int i=1) #2181
B"001_000_100_0000101", --ADDI $4,$0,0  (int n=5) #2205
B"010_000_101_0000000", --LW $5,0($0) (v[0])(memorie[0]) #4280
B"010_000_110_0000001", --LW $6,1($0) (v[1])(memorie[1]) #4301
B"000_101_110_001_0_000", --ADD $1,$5,$6  (max=v[0]+v[1]) #1710
--bucla for
B"010_011_101_0000000", --LW $5,0($3) (v[i]) #4E80
B"010_011_110_0000001", --LW $6,1($3) (v[i+1]) #4F01
B"000_101_110_010_0_000", --ADD $2,$5,$6  (suma=v[i]+v[i+1]) #1720
B"000_010_001_001_0_111", --SLT $1,$2,$1 (if(suma>max)then max=suma) #0897
B"001_011_011_0000001", --ADDI $3,$3,1  (int i=i+1) #2D81
B"100_011_100_0000001", --BEQ $3,$4,1 (sare peste adresa de jump daca i=n) #8E01
B"111_0000000000111", --JMP 7 #E007
--
X"00DA",
```

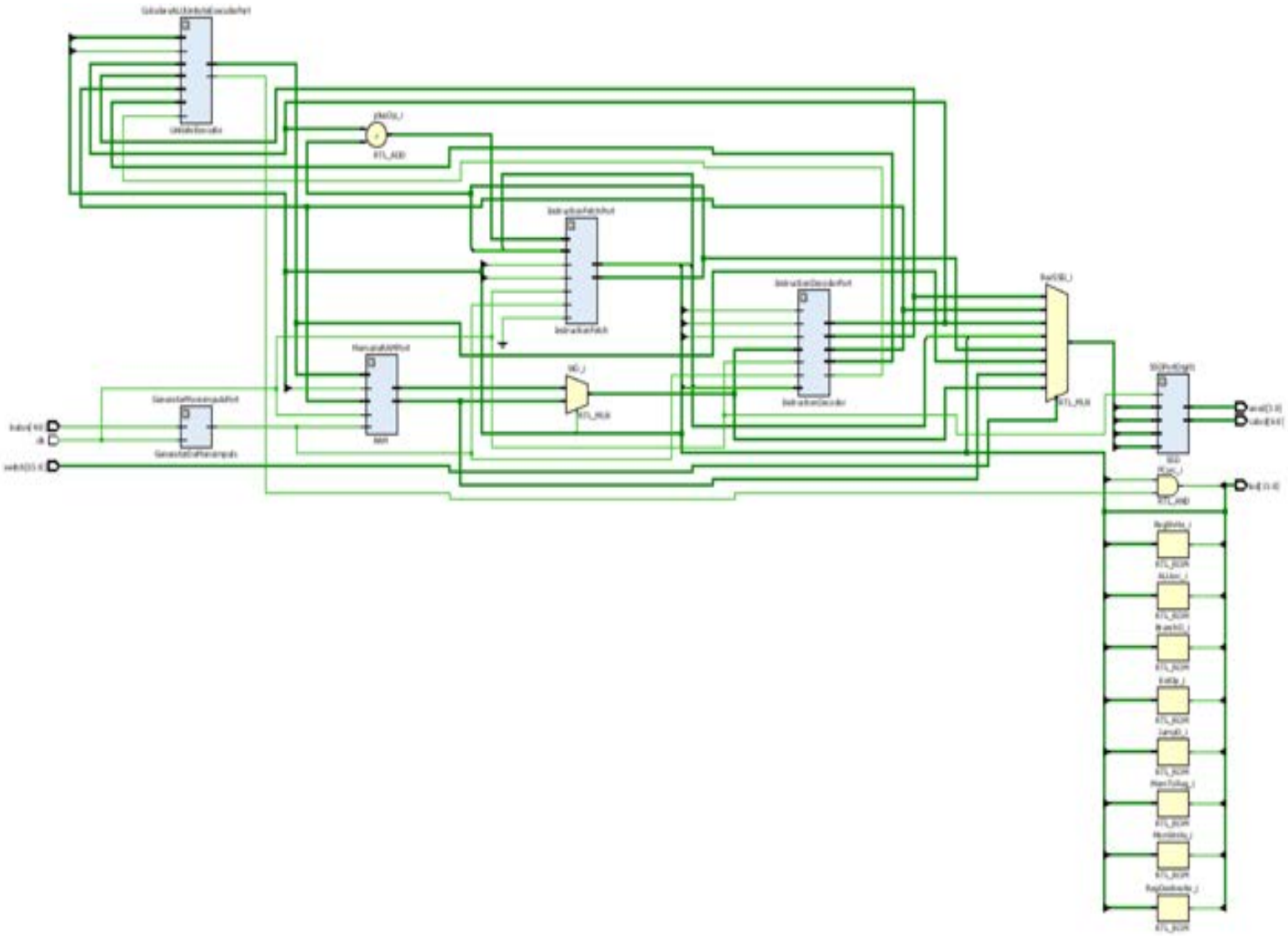
4. Trasarea programului

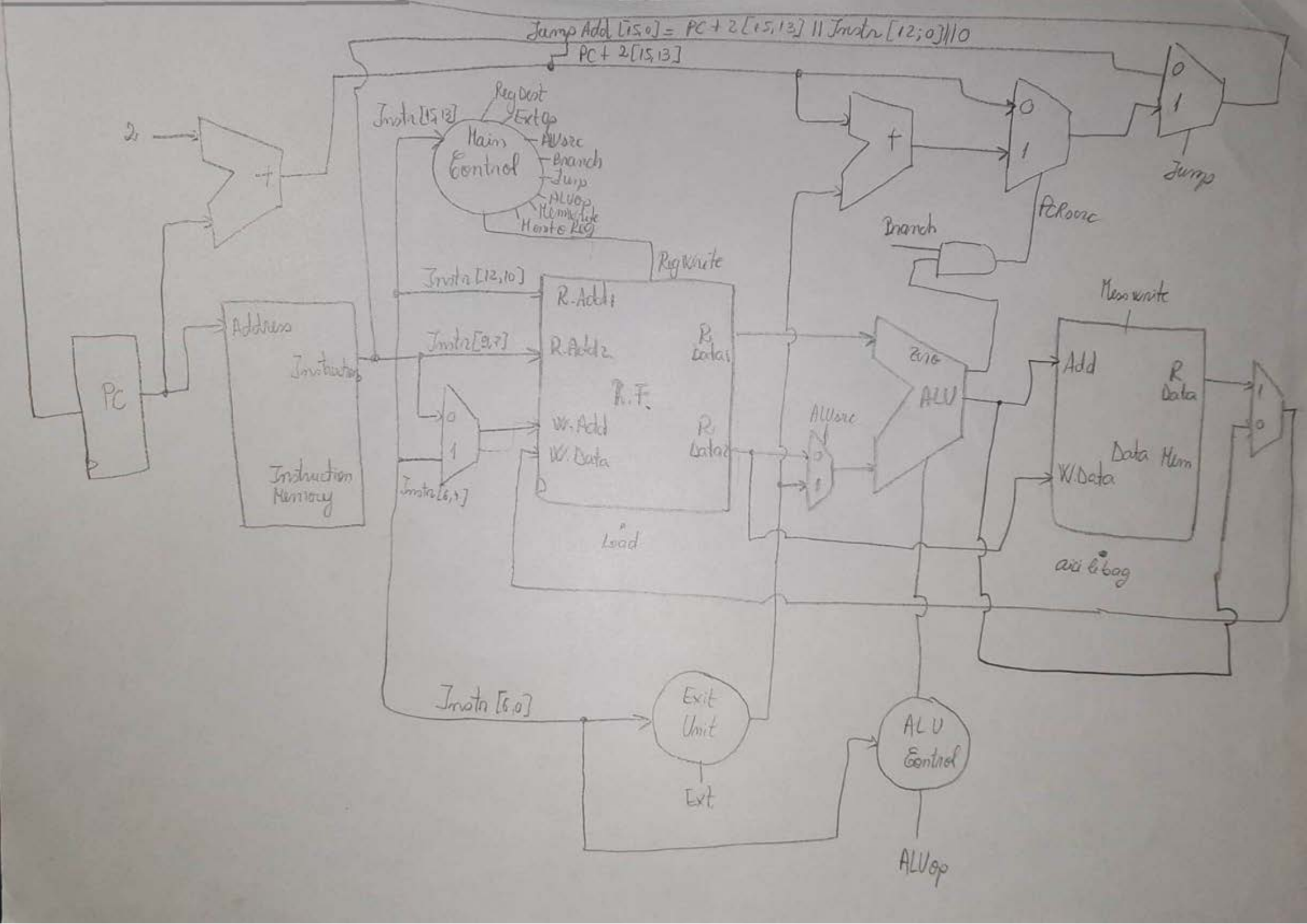
addi $1,$0,0 =>RD1=0, ExtImm=0, ALUres=0
addi $2,$0,0 =>RD1=0, ExtImm=0, ALUres=0
addi $3,$0,1 =>RD1=0, ExtImm=1, ALUres=1
addi $4,$0,0 =>RD1=0, ExtImm=5, ALUres=5
lw $5,0($0) =>RD1=0, RD2=0, ExtImm=0, ALUres=0, DataIn=2,WD=2
lw $6,1($0) => RD1=0, RD2=0, ExtImm=1, ALUres=1, DataIn=3,WD=3

add $1,$5,$6 =>RD1=2, RD2=3,ALUres=5
lw $5,0($3) => RD1=1, RD2=2, ExtImm=0, ALUres=1, DataIn=$3,WD=$3
lw $6,1($3) => RD1=1, RD2=3, ExtImm=1, ALUres=2, DataIn=$4,WD=$4

add $2,$5,$6 =>RD1=$5, RD2=$6,ALUres=$5+$6
slt $1,$2,$1 =>RD1=$2, RD2=$1, ALUres=RD1 if RD1>RD2, else RD2
addi $3,$3,1 =>RD1=$3, ExtImm=1, ALUres=$3+1
beq $3,$4,1 =>RD1=$3, RD2=$4, ExtImm=1, zero=0, Pcout = Pc + 2
jmp 7 =>ExtImm=7 se efectueaza salt la adresa 7 din memorie

5. Corectitudinea descrierii în VHDL




Jump Add [15:0] = PC + 2 [15,13] || Instr [12:0] || 0
PC + 2 [15,13]
2
+
PC
Address
Instruction
Instruction Memory
Instr [15,13]
Reg Dest
Ext Op
Main Control
ALUsrc
Branch
Jump
ALUop
Mem write
Mem to Reg
Reg Write
Instr [12,10]
Instr [9,7]
Instr [6,1]
R. Add 1
R. Add 2
W. Add
W. Data
R. F.
R Data 1
R Data 2
Load
+
Branch
PCSrc
Jump
ALUsrc
Zero
ALU
Mem write
Add
R Data
Data Mem
W. Data
Instr [6,0]
Exit Unit
Ext
ALU Control
ALUop