UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Procesorul MIPS, ciclu unic

– versiune pe 16 biți –

MOȘILĂ LUCIANA

30226

ARHITECTURA CALCULATOARELOR

2023

Instrucțiuni alese suplimentar

Instrucțiunea bgez (Branch on Greater Than or Equal to Zero)

```
bgez $rs, offset
```

opcode	rs	rt	Imm
101	sss	000	iiiiiii

RTL abstract:

If (RF[rs] >= 0) then PC ← PC + 2 + (offset << 1)

Else PC ← PC + 2

Instrucțiune de tip I. Se efectueaza o operație între conținutul unui registru și o valoare imediată;

Efectuează un salt condiționat la o adresă relativă la adresa instrucțiunii următoare dacă registrul sursă are conținutul mai mare sau egal cu zero;

Instrucțiunea bltz (Branch on Less Than Zero)

```
bltz $rs, offse
```

opcode	rs	rt	imm
110	sss	000	iiiiiii

RTL abstract:

If (RF[rs] < 0) then PC ← PC + 2 + (offset << 1)

Else PC ← PC + 2

Instrucțiune de tip I. Se efectueaza o operație între conținutul unui registru și o valoare imediată;

Efectuează un salt condiționat la o adresă relativă la adresa instrucțiunii următoare dacă registrul sursă are conținutul mai mic decât zero;

Instrucțiunea xor (Sau-exclusiv)

```
xor $rd, $rs, $rt
```

opcode	rs	rt	rd	sa	func
000	sss	ttt	ddd	0	110

RTL abstract:

RF[rd] ← RF[rs] ^ RF[rt];

Instrucțiune de tip R. Operațiile se efectuează asupra conținutului unor registre;

Realizează operația de sau-exclusiv între conținutul a două registre;

Instrucțiunea slt (Set on Less Than)

```
slt $rd, $rs, $rt
```

opcode	rs	rt	rd	sa	func
000	sss	ttt	ddd	0	111

RTL abstract: **If (RF[rs] < RF[rt]) then RF[rd] ← 1**

Else RF[rd] ← 0

Instrucțiune de tip R – operațiile se efectuează asupra conținutului unor registre;

Setează registrul destinație atunci când conținutul primului registru sursă este mai mic decât conținutul celui de-al doilea registru sursă;

Tabel cu valorile semnalelor de control pentru setul de instrucțiuni selectat

Instrucțiune	Reg Dst	Reg Write	ALU Src	ALU Ctrl	Ext Op	Mem Write	Memto Reg	Slt	Branch	Bgez	Bltz	Jump
add	1	1	0	000 (+)	X	0	0	0	0	0	0	0
sub	1	1	0	001 (-)	X	0	0	0	0	0	0	0
sll	1	1	0	010 (<<)	X	0	0	0	0	0	0	0
srl	1	1	0	011 (>>)	X	0	0	0	0	0	0	0
and	1	1	0	100 (and)	X	0	0	0	0	0	0	0
or	1	1	0	101 (or)	X	0	0	0	0	0	0	0
xor	1	1	0	110 (xor)	X	0	0	0	0	0	0	0
slt	1	1	0	111 (cmp)	X	0	0	1	0	0	0	0
addi	0	1	1	000 (+)	1	0	0	0	0	0	0	0
lw	0	1	1	000 (+)	1	0	1	0	0	0	0	0
sw	X	0	1	000 (+)	1	1	X	0	0	0	0	0
beq	X	0	0	001 (-)	1	0	X	0	1	0	0	0
bgez	X	0	0	001 (-)	1	0	X	0	0	1	0	0
bltz	X	0	0	001 (-)	1	0	X	0	0	0	1	0
j	X	0	X	XXX	X	0	X	0	0	0	0	1

In componenta ALU se executa urmatoarele operatii ce corespund tabelului cu valori:

(+) – în ALU are loc o operație de adunare
(<<) – în ALU are loc o deplasare logică la stânga cu o poziție
(>>) – în ALU are loc o deplasare logică la dreapta cu o poziție
(and) – în ALU are loc o operație de și-logic
(or) – în ALU are loc o operație de sau-logic
(xor) – în ALU are loc o operație de sau-exclusiv
(cmp) – în ALU are loc o operație de comparare (folosită doar în cazul instrucțiunii *Set on Less Than*)

Descrierea în cuvinte, cod C şi cod maşină a programului încărcat în memoria ROM.

Codul ales pentru a demonstra corectitudinea functionarii procesorului, reprezinta operatia de adunare a primelor n numere impare. Bucla "while", va continua atâta timp cât valoarea variabilei "j" este mai mică decât valoarea variabilei "n". In variabila s va fi stocata suma primelor n numere impare.

Cod in C:

```c
int j, i, n, s;
j = 0;
i = 0;
n = 5;
s = 0;
while (j < n)
{
    s = s + i;
    i = i + 2;
    j++;
}
```

Codul masina si codul de asamblare:

```
B"001_000_001_0000000",
B"001_000_010_0000001",
B"001_000_011_0000101",
B"001_000_100_0000000",

B"100_001_011_0000100",
B"000_100_010_100_0_000",
B"001_010_010_0000010",
B"001_001_001_0000001",
B"111_0000000000100",
B"011_000_100_0010100",
others => x"1111");
```

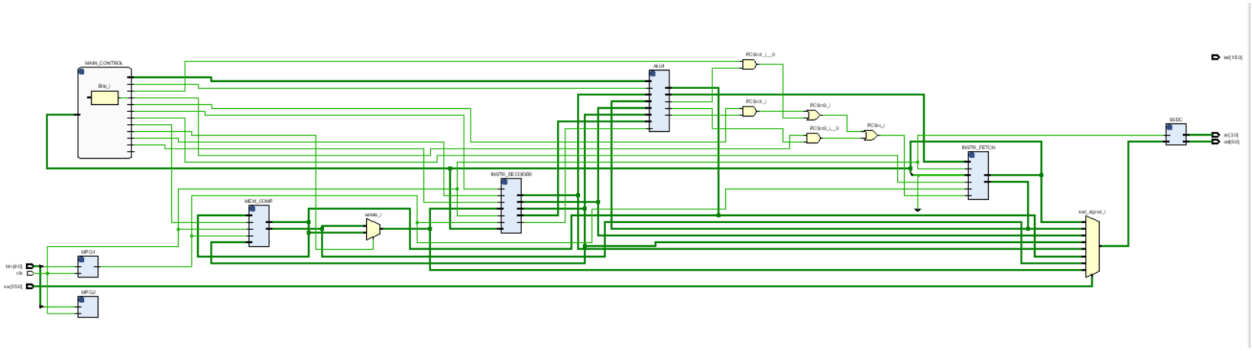
```
    RF[0] = 0;
    RF[2] = RF[0] + 1;
    RF[3] = RF[0] + 5;
    RF[4] = RF[0] + 0;
begin:
    if (RF[1] == RF[3])
        goto et_final;
    RF[4] = RF[2] + RF[4];
    RF[2] = RF[2] + 2;
    RF[1] = RF[1] + 1;
    goto begin;
et_final:
    s = RF[4];
```

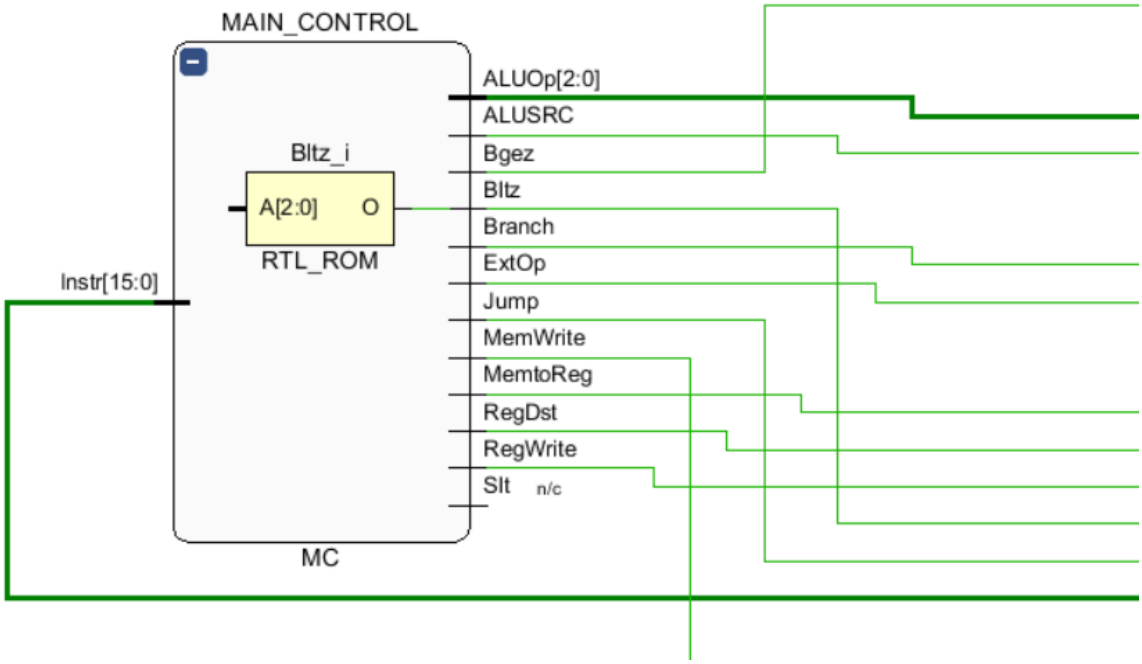
Trasarea programului:

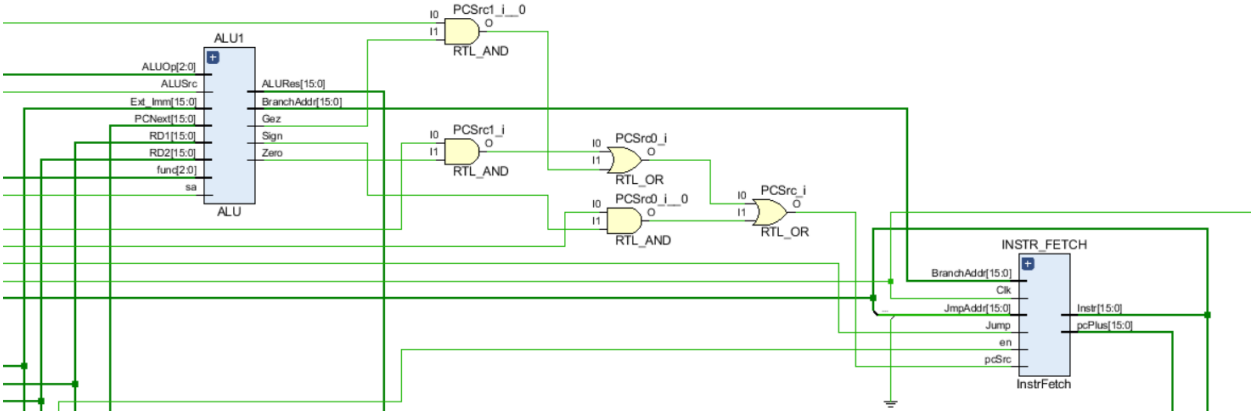
```
A =  (X"000A", X"0001", X"0110", X"B001", X"0C60", X"1743", X"3F10", X"2008", X"0403",  X"6005")
```

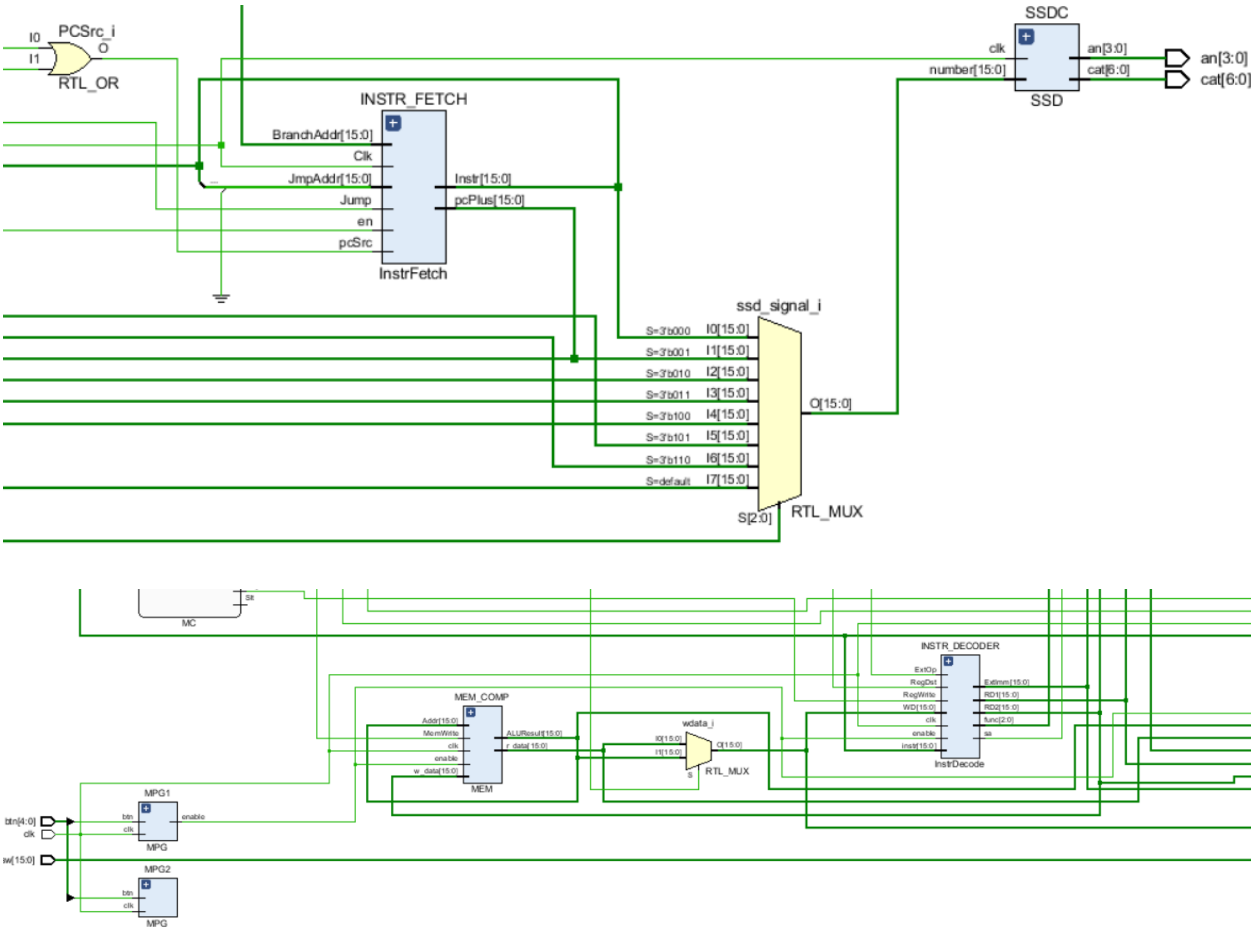
```
addi $1, $0, 0  -- RD1 = 2  RD2 = 0901  Ext_Imm = 0  ALU_Res = 2  Zero = 0
addi $2, $0, 1  -- RD1 = 2 RD2 = 2002  Ext_Imm = 1  ALU_Res = 3 Zero = 0
addi $3, $0, 5  -- RD1 = 2  RD2 = 00CD Ext_Imm = 5 ALU_Res = 7  Zero = 0
addi $4, $0, 0  -- RD1 = 2 RD2 = 34CA Ext_Imm = 0  ALU_Res = 2  Zero= 0

beq $1, $3, 4    -- nu se executa saltul => Zero = 0
add $4, $2, $4   -- RD1 = 2  RD2 = 3   ALU_Res = 5  Zero = 1
addi $2, $2, 2   -- RD1 = 3  RD2 = 3  Ext_Imm = 2  ALU_Res = 5  Zero = 0
addi $1, $1, 1   -- RD1 = 2  RD2 = 2  Ext_Imm = 1  ALU_Res = 3  Zero = 0
j 4              -- sare la adresa 4
sw $4, 20 ($0)   -- prima iteratie => nu se executa
```

RTL schemati:




MAIN_CONTROL
Bltz_i
A[2:0]
O
RTL_ROM
Instr[15:0]
ALUOp[2:0]
ALUSRC
Bgez
Bltz
Branch
ExtOp
Jump
MemWrite
MemtoReg
RegDst
RegWrite
Slt
n/c
MC


ALU1
ALUOp[2:0]
ALUSrc
Ext_Imm[15:0]
PCNext[15:0]
RD1[15:0]
RD2[15:0]
func[2:0]
sa
ALURes[15:0]
BranchAddr[15:0]
Gez
Sign
Zero
ALU
PCSrc1_i__0
RTL_AND
PCSrc1_i
RTL_AND
PCSrc0_i
RTL_OR
PCSrc0_i__0
RTL_AND
PCSrc_i
RTL_OR
INSTR_FETCH
BranchAddr[15:0]
Clk
JmpAddr[15:0]
Jump
en
pcSrc
Instr[15:0]
pcPlus[15:0]
InstrFetch



Testarea a fost efectuata pe placuta FPGA. Nu am observat erori in timpul acesteia.