MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Procesorul MIPS – ciclu unic 16 biti

Nume: Pinciuc Darius-Eduard

Grupa: 30226

a. Instrucțiuni suplimentare

1. Instrucțiunea xor (Sau-exclusiv)

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

– realizeaza sau-exclusiv intre registri

– Sintaxa: `xor $rd, $rs, $rt`

– Formatul:

opcode	rs	rt	rd	sa	func
000	rs	rt	rd	0	110

– RTL abstract: RF[rd] ← RF[rs] ^ RF[rt];

2. Instrucțiunea slt (Set on Less Than)

– Instrucțiune de tip R

–Registrul destinatie este setat pe 1, cand registrul sursa este mai mic decat registrul target

–Sintaxa: `slt $rd, $rs, $rt`

–Formatul instrucțiunii:

opcode	rs	rt	rd	sa	func
000	rs	rt	rd	0	111

–RTL abstract:

If (RF[rs] < RF[rt]) then RF[rd] ← 1

Else RF[rd] ← 0

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

– Instrucțiune de tip I – operație între conținutul unui registru și o valoare imediată;

– Efectuează un salt condiționat la o adresă dacă valoarea din registrul sursa este >= 0;

– Sintaxa: `bgez $rs, offset`

– Formatul:

opcode	rs	rt	imm
101	rs	rt	imm

– RTL abstract:

$$\text{If } (RF[rs] >= 0) \text{ then } PC \leftarrow PC + 2 + (offset << 1)$$
$$\text{Else } PC \leftarrow PC + 2$$

4. Instrucțiunea bltz (Branch on Less Than Zero)

– Instrucțiune de tip I

– Similar cu bgez, efectuat doar daca registrul sursa <=0;

– Sintaxa: `bltz $rs, offset`

– Formatul:

opcode	rs	rt	imm
110	sss	000	iiiiii

– RTL abstract:

$$\text{If } (RF[rs] < 0) \text{ then } PC \leftarrow PC + 2 + (offset << 1)$$
$$\text{Else } PC \leftarrow PC + 2$$

b. Tabel cu valorile semnalelor de control

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

(+) – în ALU => adunare

(–) – în ALU => scădere

xor – în ALU => sau-exclusiv

and – în ALU => și-logic

or – în ALU => sau-logic

<< – în ALU => deplasare logică la stânga cu o poziție

>> – în ALU => deplasare logică la dreapta cu o poziție

cmp – în ALU => comparatie

c. Cod C și cod mașina

Pentru a demonstra functionalitatea procesorului am ales un program ce determina suma numerelor de la 1 la n. Scrierea sumei se face la adresa addr = 1.

In pozele urmatoare sunt ilustrate atat codul in C, cat si codul masina, iar pentru o intelegere mai buna programul a fost rescris astfel incat sa reflecte instructiunile programului in limbaj de asamblare.

```
B"001_000_010_0000000", -- addi $2, $0, 0    #2100  -- s = 0;
B"001_000_011_0000001", -- addi $3, $0, 1    #2181  -- i = 1;
B"001_000_100_0000011", -- addi $4, $0, 10   #2203  -- n = 10;
B"100_011_100_0000011", -- beq $3, $4, 3     #8E03  -- while(i<n){
B"000_010_011_010_0_000", -- add $2, $3, $2  #09A0  -- s = s+i;
B"001_011_011_0000001", -- addi $3, $3, 1    #2d81
B"111_0000000000011", -- j 3                 #E003
B"011_000_010_0000001", -- sw $5, 1          #6101
others => x"1111"
```

```
int s = 0;
int i = 1;
int n = 3;
while (i < n)
{
    s = s + i;
    i = i + 1;
}
```

```
    RF[0] = 0;
    RF[2] = RF[0] + 1;
    RF[4] = RF[0] + 10;
loop:
    if (RF[3] == RF[4])
        goto final_loop;
    RF[2] = RF[3] + RF[2];
    RF[3] = RF[3] + 1;
    goto loop;
final_loop:
    addr(1) = RF[2];
```

d. Trasarea executiei

```
                          --     RD1 RD2 ALURes, ExtImm,  Sign, Zero
 0 addi $2, $0, 0     #2100  -- 1;   3;   1;       0      0      0
 1 addi $3, $0, 1     #2181  -- 1;   4;   2;       1      0      0
 2 addi $4, $0, 3     #2203  --  1;  0A00; 4;      3      0      0
 3 beq $3, $4, 3      #8E03  --  nu se face saltul              0
 4 add $2, $3, $2     #09A0  -- 1;   2;   3;              0      1
 5 addi $3, $3, 1     #2d81  -- 2;   2;   3;       1      0      0
 6 j 3                #E003  -- sare la adresa 3 din memorie
 7 sw $2, 1           #6101  -- nu se executa la prima iteratie
```

Am testat pe placuta. Nu am intampinat erori, nici probleme. S-ar putea sa existe mici probleme pe care le-am sarit din vedere.

e. RTL schematic