**Raport Procesor MIPS-16 biti**

Moldovan Adelin

Grupa 302210

1. Cele 4 instructiuni alese suplimentar:

**Slt**-compar 2 registri.

**Xor**-pentru a veadea daca doi registri au aceasi valoarea.

**Andi**-il folosesc sa vad daca un numar este par sau impar(face SI pe biti: registrul rs si valoarea 1, daca e 0 e numar par, iar daca e 1 numar par.

**Ori**-pun o valoare in registrul destinatie o valoare imediata.

* De tip R:

**XOR**: $d <- $s ^ $t, PC <- PC + 1

xor $d, $s, $t

000 sss ttt ddd 0 110

Ex. xor $1, $2, $3

000 010 011 001 0 110

**SLT**: if $s < $t, $d <- 1 else $d <-0, PC <- PC + 1

slt $d, $s, $t

000 sss ttt ddd 0 111

Ex. slt $1, $2, $3

000 011 010 001 0 111

* De tip I:

**ORI**: $t <- $s | imm, PC <- PC + 1

ori $t, $s, imm

000 sss ttt iiii 0 110

Ex. ori $1, $2, $3

000 010 011 001 0 110

**ANDI**: $t <- $s & imm

andi $t, $s, imm

000 ttt sss iiii 0 101

Ex. andi $1, $2, 3

000 011 001 001 0 110

b.Tabelul cu valorile semnalelor de control pentru toate instructiunile:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | Opcode | Reg  Dst | Ext  Op | ALU  Src | Branch | Jump | Mem  Write | MemtoReg | Reg  Write | ALUOp | Func | ALUCtrl |
| Instructiuni de tip R | | | | | | | | | | | | |
| ***ADD*** | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000(funct) | 000(+) | 000(+) |
| ***SUB*** | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000(funct) | 001(-) | 011(-) |
| ***SLL*** | 000 | X | X | X | 0 | 0 | 0 | 0 | 1 | 000(funct) | 010(<<|) | 010(<<|) |
| ***SRL*** | 000 | X | X | X | 0 | 0 | 0 | 0 | 1 | 000(funct) | 011(>>|) | 011(>>|) |
| ***AND*** | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000(funct) | 100(&) | 100(&) |
| ***OR*** | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000(funct) | 101(|) | 101(|) |
| ***XOR*** | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000(funct) | 110(^) | 110(^) |
| ***SLT*** | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000(funct) | 111(-) | 001(-) |
| Instructiuni de tip I | | | | | | | | | | | | |
| ***ADDI*** | 001 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 001(+) | xxx | 000(+) |
| ***LW*** | 010 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 010(+) | xxx | 000(+) |
| ***SW*** | 011 | X | 1 | 1 | 0 | 0 | 1 | X | 0 | 011(0) | xxx | 000(+) |
| ***BEQ*** | 100 | X | 1 | 0 | 1 | 0 | 0 | X | 0 | 100(-) | xxx | 001(-) |
| ***ANDI*** | 101 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 101(&) | xxx | 100(&) |
| ***ORI*** | 110 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 110(|) | xxx | 101(|) |
| Instructiuni de tip J | | | | | | | | | | | | |
| ***J*** | 111 | x | x | x | x | 1 | 0 | x | 0 | xxx | xxx | xxx |

c.Descrierea in cuvinte a programului scris

addi $1,$0,1 - pun in registrul 1 valoarea 1 (limita inferioara a sirului)

ori $2,$0,10 - in registrul 2 adaug 10 (limita superioara a sirului)

addi $5,$0,0 - cu ajutorul reg. $5 imi parcurg memoria

ori $4,$0,0 - in reg. $4- fac suma elementelor din memorie

et: andi $7,$1,1 - verific daca numarul din $1 e par sau impar

beq $7,$0,3 - daca e par sar la eticheta “par” pentru a-l inmulti cu “2”

addi $7,$1,-1 - daca e impar, atunci scad din acesta valoarea “1”

sw $7,0($5) - salvez valoarea actuala (daca e nr par, “nr \*2”, daca e impar,” nr-1”) in memorie

j 12 (et2) - sar la instructiunea 12 din program

par : sll $6,$1,1 - daca numarul e par il inmultesc cu “2”

sw $6,0($5) - salvez numarul par inmultit cu “2” in memorie

et2: lw $3,0($5) - scot valoarea actuala din memorie si o pun registrul $3

add $4,$4,$3 - adun la suma($4) valoarea din memorie extrasa din instructiunea precedenta

addi $1,$1,1 - trec la urmatorul element din sir

addi $5,$5,1 - cresc valoarea adresei memoriei

beq $1,$2,1 (end) - daca am ajuns la limita superioara, merg la ultima instructiune(programul s-a terminat)

j 4 (et) - daca nu am ajuns la limita superioara, imi parcurg iar pasii anteriori

end: sw $4,0($5) - la finalul programului, salvez si suma elementelor sirului “modificate” in memorie

Parcurg elementele de la 1 la 9 inclusiv, elementele pare le inmulteste cu 2,

iar pe cele impare le scad cu valoarea 1, valorile le adaug in memorie. Pe parcurs fac o

suma a elementelor din memorie, si la final o adaug si pe aceasta in memorie.

d.Trasarea executiei programului.

addi $1,$0,1 - Instr = x"2081", RD1=0, Ext\_Imm=1, ALURes=1

ori $2,$0,10 - Instr= x"C10A", RD1=0, Ext\_Imm=A, ALURes=A

addi $5,$0,0 - Instr= x"2280, RD1=0, Ext\_Imm=0, ALURes=0

ori $4,$0,0 - Instr= x"C200", RD1=0, Ext\_Imm=0, ALURes=0

andi $7,$1,1 - Instr= x"A781, RD1=1, Ext\_Imm=1, ALURes=1

beq $7,$0,3 - Instr= x"9C03", RD1=1, RD2=0, ALURes=1, Ext\_Imm=3, branchAddress=9

addi $7,$1,-1 - Instr= x"27FF”, RD1=1, Ext\_Imm=FFFF, ALURes=0

sw $7,0($5) - Instr= x"7780”, RD1=0, RD2=0, Ext\_Imm=0, ALURes=0

j 12 - Instr= x"E00C", RD1=0, RD2=0, Ext\_Imm=C, ALURes=0, jumpAddress=C

sll $6,$1,1 - Instr= x"046A”, RD1=2, RD2=0, ALURes=4

sw $6,0($5) - Instr= x"7700", RD1=1, RD2=4, Ext\_Imm=0, ALURes=1

lw $3,0($5) - Instr= x"5580", RD1=0, RD2=0, Ext\_Imm=0, ALURes=0,MemData=0(0 pentru prima iteratie, la a doua 4, apoi 2, 8, 5 etc)

add $4,$4,$3 - Instr= x"11C0", RD1=0, RD2=0, Ext\_Imm=0, ALURes=0

addi $1,$1,1 - Instr= x"2481", RD1=1, RD2=0, Ext\_Imm=1, ALURes=2

addi $5,$5,1 - Instr= x"3681" , RD1=0, RD2=0, Ext\_Imm=1, ALURes=1

beq $1,$2,1 - Instr= x"8501" , RD1=2, RD2=A, Ext\_Imm=1, ALURes=FFF8, branchAdd=x”11”(17)

j 4 - Instr= x"E004" , RD1=0, RD2=0, Ext\_Imm=4, ALURes=0, jumpAddress=4

**.**

**.**

**.**

sw $4,0($5) - Instr= x"7600" , RD1=9, RD2=x”3C” (=60), Ext\_Imm=0, ALURes=9

e. Nu exista activitati incomplete.

f. Este totul descris in vhdl. Nu exista erori.

RTL schematic:

Diagram

Description automatically generated

g. A fost testat pe placa si este functional.