

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**~SPECIALIZAREA: CALCULATOARE SI TEHNOLOGIA INFORMATIEI~**

RAPORT MIPS 16

**Student: Arseniuc Anamaria**

**Grupa 30226**

**An 2, Semestrul 2**

1. Cele 4 instructiuni alese suplimentar

* Instructiuni de tip R:

***xor*** (mi se pare util deoarece putem cu ajutorul lui sa vedem daca 2 registre au aceeasi valoare, sau sa setam un registru la 0, facand operatia de “xor” cu el insusi)

RTL: RF[rd] <- RF[rs] ^ RF[rt]

***slt*** (ne ajuta sa comparam usor 2 registri, setand in registrul destinatie valoarea 1 sau 0)

RTL: if ( RF[rs] < RF[rt]) then

RF[rd] <-1

Else

RF[rd] <-0

* Instructiuni de tip I

***andi*** (l-am folosit si in “programelul” meu pentru a vedea daca un numar este par sau impar; facand “SI” pe biti a registrului rs cu valoarea imediata “1”, valoarea rezultata poate fi “0” (numar par) sau “1” (pentru numar impar) )

RTL: RF[rt]<-RF[rs] & Z\_Ext(imm)

***ori*** (si pe acesta l-am folosit in program; in loc de addi, se poate folosi si aceasta instructiune pentru a pune in registrul destinatie o valoare imediata (ex. addi $3,$0,7 ~ ori $3,$0,7) )

RTL: RF[rt]<-RF[rs] | Z\_Ext(imm)

1. tabelul cu valorile semnalelor de control pentru toate instructiunile

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instr** | **Opcode** | **Reg**  **Dst** | **Ext**  **Op** | **ALU**  **Src** | **Branch** | **Jump** | **Mem**  **Write** | **Memto**  **Reg** | **Reg**  **Write** | **ALUOp** | **Func** | **ALUCtrl** |
| ***ADD*** | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 1 | 000(func) | 000(+) | 000(+) |
| ***SUB*** | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 1 | 000(func) | 001(-) | 001(-) |
| ***SLL*** | 000 | x | x | x | 0 | 0 | 0 | 0 | 1 | 000(func) | 010(<<l) | 010(<<l) |
| ***SRL*** | 000 | x | x | x | 0 | 0 | 0 | 0 | 1 | 000(func) | 011(>>l) | 011(>>l) |
| ***AND*** | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 1 | 000(func) | 100(&) | 100(&) |
| ***OR*** | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 1 | 000(func) | 101(|) | 101(|) |
| ***XOR*** | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 1 | 000(func) | 110(^) | 110(^) |
| ***SLT*** | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 1 | 000(func) | 111(-) | 001(-) |
| ***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(+) | 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(|) |
| ***J*** | 111 | x | x | x | x | 1 | 0 | X | 0 | xxx | xxx | xxx |

1. descrierea in cuvinte a programului scris si executat de procesor (cel incarcat in memoria rom)

Parcurg elementele de la 1 la 9, inclusiv; elementele pare le inmultesc cu 2, iar pe cele impare le scad cu valoarea “1”, valorile rezultatele adaugandu-le in memorie; pe parcurs fac o suma a elementelor din memorie disponibile, iar la final o adaug si pe aceasta in memorie.

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 11 (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

1. trasarea executiei programului dacă există un ciclu repetitiv – se traseaza prima iterație. Aici trebuie explicat numeric ce valori se calculeaza la fiecare instructiune si ce valori relevante apar in calea de date pe campurile afisate pe SSD

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. daca exista activitati (parti din procesor) incomplete din laboratoarele 4-7 se va mentiona explicit acest lucru in raport

-nu exista

f. corectitudinea descrierii vhdl

* este totul descris in vhdl: DA
* exista erori? : NU
* print screen cu RTL schematic la nivelul de sus unde se vad unitatile implementate IF, ID, EX, etc)

RTL schematic 