MIPS

PIPELINE

Horga Ana-Maria

Grupa : 30225

Pentru testarea micro procesorului pipeline, am implementat un program in care am folosit cateva operatii din setul de instructiuni pe care le poate face microprocesorul: add, addi, sll,beq,lw,sw,j. Programul realizeaza suma elemntelor unui vector, mai inati fiecare element fiid shiftat la stanga cu 2 poziti (sau \*4). Toate elementele sunt functionale, suma finala ar trebui sa fie 112 ceea ce in hexa este X“00000070”. Am testat pe placa, iar toate valorile au fost cele asteptate. Singurele probleme intampinate au fost de neatentie in scrierea codului, dar am reusit sa le gasesc si corectez. Astfel, microprocesorul ar trebui sa functioneze pentru orice program de testare care contine instructiunile:add, sub, sll, srl, and, or, mult, xor, addi, lw, sw, beq, bne, ori, j.

Instructiunile alese suplimentar sunt: mult, xor, bne, ori.

Instructiunea Multiplication realizeaza inmultirea a doua numere.

ASM: mult $d, $s, $t

RTL: $d<- $s \* $t

Format: 000000 sssss ttttt ddddd 00000 000111

Semnale de control:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RegDst** | **ExtOp** | **ALUSrc** | **Branch** | **BR**\_NOT | **Jump** | **MemWrite** | **MemtoReg** | **Reg Write** | **ALUOp[2:0]** | **ALUCtrl[2:0]** |
| 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 100 |

Instructiunea XOR realizeaza sau exclusiv pe biti.

ASM: xor $d, $s, $t

RTL: $d<- $s ^ $t

Format: 000000 sssss ttttt ddddd 00000 001000

Semnale de control:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RegDst** | **ExtOp** | **ALUSrc** | **Branch** | **BR**\_NOT | **Jump** | **MemWrite** | **MemtoReg** | **Reg Write** | **ALUOp[2:0]** | **ALUCtrl[2:0]** |
| 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 101 |

Instructiunea Branch on not equal, este asemanatoare cu beq, singura diferenta este ca daca operanzii care se compara sunt diferiti sare la adresa de offset, altfel trece la instructiuea urmatoare. Pnetru aceasta instructiune am adaugat un semnal branch not dat de Main Control. Am adaugat si un semnal not\_zero dat de ALU, iar la final PCSrc(cel care decide daca sa se faca salt beq sau bne) a primit valoarea (branch and zero) or (branch\_not and not\_zero).

ASM: bne $s, $t, offset

RTL: if $s!=$t then PC<-(PC+4)+S\_EXT(offset)<<2

Else PC<-PC+4

Format: 001101 sssss ttttt offset(16 biti)

Semnale de control:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RegDst** | **ExtOp** | **ALUSrc** | **Branch** | **BR**\_NOT | **Jump** | **MemWrite** | **MemtoReg** | **Reg Write** | **ALUOp[2:0]** | **ALUCtrl[2:0]** |
| x | 1 | 0 | 0 | 1 | 0 | 0 | x | x | 010 | 010 |

Instructiunea Or immediate, este asemanatoare cu or, singura diferenta este ca un operand este registru, iar celalalt este imediata extensa.

ASM: ori $t, $s , imm

RTL: $t<-$s | Z\_EXT(imm)

Format: 001110 sssss ttttt imm(16 biti)

Semnale de control:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RegDst** | **ExtOp** | **ALUSrc** | **Branch** | **BR**\_NOT | **Jump** | **MemWrite** | **MemtoReg** | **Reg Write** | **ALUOp[2:0]** | **ALUCtrl[2:0]** |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 011 | 011 |