Documentatie processor Mips

Instructiunile suplimentare alese de mine sunt :bsez,subi,not, si xor.  
De asemenea am modifcat putin Mips-ul astfel incat rotatia sa se faca cu semn sau fara semn ptr sll , resprectiv slr. Si am reusit sa fac si un compilator de cod in java 😊 care traduce din asm in masina.

add $1 $2 $1 ==> 000\_010\_001\_001\_0\_000 -> 0

sub $1 $2 $1 ==> 000\_010\_001\_001\_0\_001 -> 1

sll $4 $5 1 ==> 000\_101\_000\_100\_1\_010 -> 2

slr $4 $5 0 ==> 000\_101\_000\_100\_0\_011 -> 3

and $1 $2 $1 ==> 000\_010\_001\_001\_0\_100 -> 4

or $4 $5 $6 ==> 000\_101\_110\_100\_0\_101 -> 5

xor $1 $2 $3 ==> 000\_010\_011\_001\_0\_110 -> 6

not $1 $2 ==> 000\_010\_000\_001\_0\_111 -> 7

addi $1 $0 24 ==> 001\_000\_001\_0011000 -> 8

lw $1 $2 ==> 010\_010\_001\_0000000 -> 9

sw $1 $2 ==> 011\_010\_001\_0000000 -> 10

beq $1 $2 113 ==> 100\_001\_010\_1110001 -> 11

subi $1 $0 24 ==> 101\_000\_001\_0011000 -> 12

bsez $3 $2 20 ==> 110\_011\_010\_0010100 -> 13

j 123 ==> 111\_0000001111011 -> 14

**Semnale control MIPS16 pentru Anexa 5**

<?> ϵ {\_gez, \_ne, \_gtz}

*Tipuri de operații care se pun în paranteză la ALUOp si ALUCtrl:* {(+), (-), (&), (|), (^), (<<*l*), (<<*lv*), (>>*l*), (>>*a*), (<)}, & - AND, | - OR, ^ *- XOR, l* *- logic, a - aritmetic, v - cu variabilă*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instrucțiune** | **Opcode** *Instr(15-13)* | **RegDst** | **ExtOp** | **ALUSrc** | **Branch** | **bsez** | **Jump** | **MemWrite** | **MemtoReg** | **Reg Write** | **ALUOp (1:0)** | **func**  *Instr(2-0)* | **ALUCtrl (2:0)** |
| add | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 | 000 | 000 |
| sub | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 | 001 | 001 |
| sll | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 | 010 | 010 |
| slr | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 | 011 | 011 |
| and | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 | 100 | 100 |
| Or | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 | 101 | 101 |
| Nor | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 | 110 | 110 |
| not | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 | 111 | 111 |
| addi | 001 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 00 | -- | 000 |
| Lw | 010 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 00 | -- | 000 |
| sw | 011 | X | 1 | 1 | 0 | 0 | 0 | 1 | X | 0 | 00 | -- | 000 |
| beq | 100 | X | 1 | 0 | 1 | 0 | 0 | 0 | X | 0 | 10 | -- | 001 |
| subi | 101 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 10 | -- | 001 |
| bsez | 110 | X | 1 | 0 | 0 | 1 | 0 | 0 | X | 0 | 10 | -- | 001 |
| j | 111 | X | X | X | X | X | 1 | 0 | X | X | XX | X | X |

**c. descrierea in cuvinte (si cod C daca e disponibil) a programului scris si executat de procesor (cel incarcat in memoria rom)**

ALgoritmul de mai jos poate fi urmarit mult mai bine in fisierele CiurErastene.asm si CodMasina.asm din folderul COD ASM folosind Notepad++ sau Notepad;

//Programul calculeaza numarul de numere prime din intervalul [a;b] cu ajutorul ciurului lui Erstotene

//Programul functioneaza oricare ar fi a si b < 100 dar poate fi ajustat usor sa duca mult mai mult

xor $0 $0 $0  
  
addi $1 $0 24 //a  
addi $2 $0 35 //b  
  
addi $7 $0 1 //incarcam adresa de memorie  
sw $1 $7 // pe pozitiile 1 si 2 am pus intervalul  
addi $7 $0 2  
sw $2 $7  
  
  
addi $7 $0 3 //initializam v[0] si v[1] cu 0 unde v = 3 (POINTER)  
sw $0 $7 //STOCAM 0 LA POTIZIA \*$7  
addi $7 $0 4  
sw $0 $7  
  
addi $7 $0 1 // 1 vom pune pentru a marca ca numarul e posibil prim  
addi $1 $0 105  
addi $5 $0 6 ///acum avem 4 pe reg5 si 103 pe reg1 si 1 pe $7  
  
  
subi $1 $1 1 //bucla1: //for ($1 = 104 ;$1 <= 5 ;$1--)  
 sw $7 $1 // initializam tot de la v[2] la v[100] cu 1 (nu uita ca v = 3)  
BSEZ $5 $1 -3 // => bucla1  
  
//$1,2 indecsi  
//$3 - scrieri citiri memorie  
//$7 - folosit pentru adresa memoriei  
//$4 - stocheaza marimea vectorului  
//$6 - variabila ajutatoare  
  
addi $1 $0 2  
addi $4 $0 100  
  
addi $7 $1 3 //loop1: for ($1 = 2 ; $1 <= 100 ;$1 ++) $7 = $1 + v  
 lw $3 $7 //se citeste v[$1]  
 BEQ $3 $0 5 //if (v[$1] == 0) sarim peste loop2  
  
 add $2 $1 $0  
 add $2 $2 $1 //loop2: for ($2 = $1 ;$2 <=100; $2 += $1 )  
 addi $7 $2 3 //$7 = &v[$2]  
 sw $0 $7 //v[$2] = 0  
 bsez $2 $4 -4 // =>loop2  
  
addi $1 $1 1 //$1 ++  
bsez $1 $4 -10 //=>loop1  
  
  
addi $7 $0 1  
lw $1 $7 // initializam i cu a din [a b]  
addi $7 $0 2  
lw $4 $7  
xor $5 $5 $5 //$5 = 0 //$5 va fi contorul  
  
addi $7 $1 3 //loop3: for ($1 = a ; $1<=b; $1++)  
 lw $3 $7  
 BEQ $3 $0 1 //if ($3 == 0) sarim peste adunare  
 addi $5 $5 1 //numaram  
 //final if  
addi $1 $1 1  
bsez $1 $4 -6 //=>loop3  
  
  
addi $5 $5 0 // afisam $5 la infinit ,programul va sari aici  
j 40 //Febra placuta la degete de la apasat pe buton la placuta

**d)Trasarea executiei programului:**

xor $0 $0 $0 // RD1 = 0 , Rd2 = 0, ExtImm =6 , AluRes =0 , RamData =1 , WD=0

addi $1 $0 24 //a // RD1 = 0 , Rd2 = 24, ExtImm =24 , AluRes =24 , RamData =0 , WD=24

addi $2 $0 35 //b // RD1 = 0 , Rd2 = 35, ExtImm =35 , AluRes =35 , RamData =0 , WD=35

addi $7 $0 1 //incarcam adresa de memorie // RD1 = 0 , Rd2 = 1, ExtImm =1 , AluRes =1 , RamData =2 , WD=1

sw $1 $7 // pe pozitiile 1 si 2 am pus intervalul // RD1 = 1 , Rd2 = 24, ExtImm =0, AluRes =1 , RamData =24 , WD=1

addi $7 $0 2 // RD1 = 0 , Rd2 = 2, ExtImm =2 , AluRes =2 , RamData =3 , WD=2

sw $2 $7 // RD1 = 2 , Rd2 = 35, ExtImm =0 , AluRes =2 , RamData =35 , WD=2

addi $7 $0 3 //initializam v[0] si v[1] cu 0 unde v = 3 (POINTER) // RD1 = 0 , Rd2 = 3, ExtImm =3 , AluRes =3 , RamData =0 , WD=3

sw $0 $7 //STOCAM 0 LA POTIZIA \*$7 // RD1 = 3 , Rd2 = 0, ExtImm =0 , AluRes =3 , RamData =0 , WD=3

addi $7 $0 4 // RD1 = 0 , Rd2 = 4, ExtImm =4 , AluRes =4 , RamData =0 , WD=4

sw $0 $7 // RD1 = 4 , Rd2 = 0, ExtImm =0 , AluRes =4 , RamData =0 , WD=4

addi $7 $0 1 // 1 vom pune pentru a marca ca numarul e posibil prim // RD1 = 0 , Rd2 = 1, ExtImm =1 , AluRes =1 , RamData =24 , WD=1

addi $1 $0 105 // RD1 = 0 , Rd2 = 103, ExtImm =103 , AluRes =103 , RamData =0 , WD=103

addi $5 $0 6 ///acum avem 4 pe reg5 si 103 pe reg1 si 1 pe $7 // RD1 = 4 , Rd2 = 4, ExtImm =4 , AluRes =4 , RamData =0 , WD=4

addi $6 $1 0 //bucla1: //for ($1 = 104 ;$1 <= 5 ;$1--) // RD1 = 103 , Rd2 = 103, ExtImm =0 , AluRes =103 , RamData =0, WD=103

Prima bucla incepe aici

subi $1 $6 1 // RD1 = 103 , Rd2 = 102, ExtImm =1 , AluRes =102 , RamData =0 , WD=102

sw $7 $1 // initializam tot de la v[2] la v[100] cu 1 (nu uita ca v = 3) // RD1 = 102, Rd2 = 1, ExtImm =0, AluRes =1 , RamData =1102 , WD=0

BSEZ $5 $1 -4 // => bucla1 // RD1 = 4 , Rd2 = 102, ExtImm =FFFC , AluRes =FF9E , RamData =0 , WD=FF9E

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

**Am reusit sa rezolv toate erorile**

**f. corectitudinea descrierii vhdl (este totul descris in vhdl, exista erori? Daca da, explicatii!, print screen cu RTL schematic la nivelul de sus unde se vad unitatile implementate IF, ID, EX, etc)**

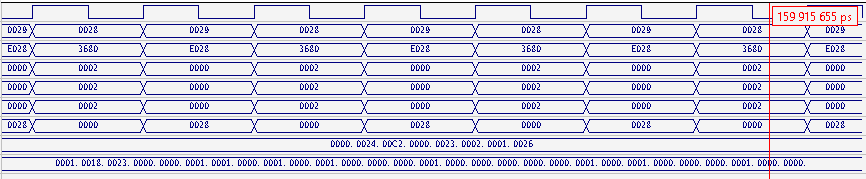
**Da, am cateva warning-uri dar nu par sa fie ceva important .(Fie semnale declarate nefolosite fie altele)**

**g. optional! daca a fost testat pe placa si este functional (complet?) sau daca a fost testat cu simulatorul – detalii despre testare/simulare**

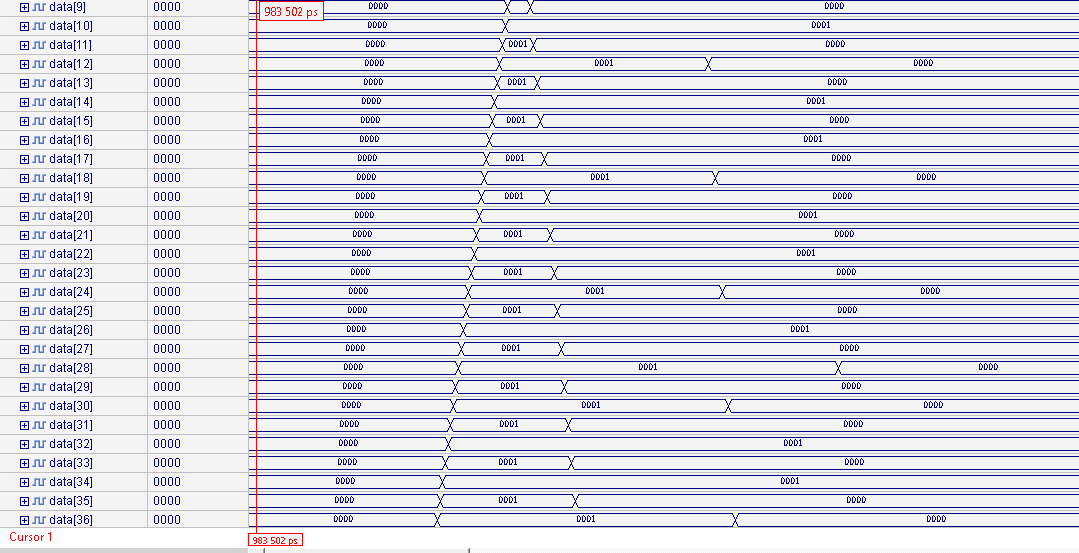
Da , a fost testat si pe placa si pot spune ca primele 300 de instructiuni functioneaza perfect , dar am calculat ca programul trebuie sa ruleze cateva zeci de mii de instructiuni .

Pentru a Simula programul se vor seta semnalele clk si ok ca fiind ambele clock la 100MHZ si se va rula timp de 160us.

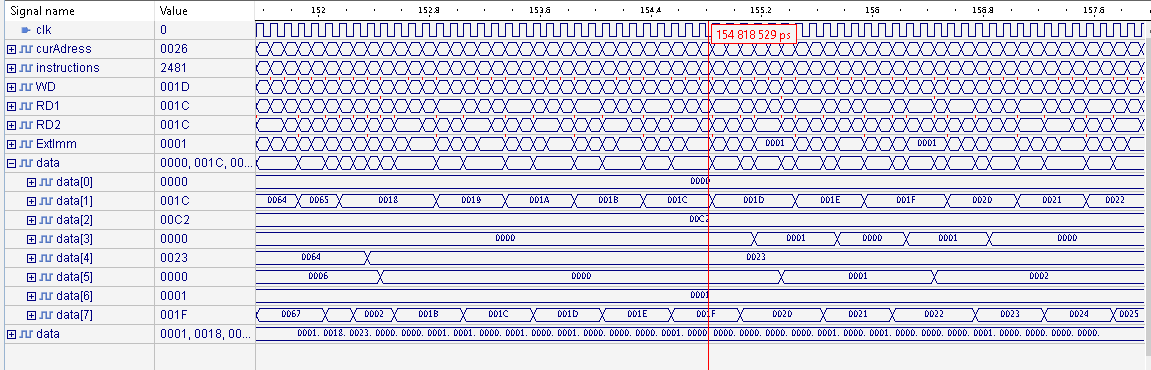
LA finalul simularii ar trenuii sa se repete ultimile 2 instructiuni la nesfarsit , iar la instructiunea x”28” va arata in RD1 respectiv RD2 rezzultatul final , mai exact 2 (programul calculeaza numerele prime din intervalul [24,35])



**Mai jos se poate vedea cum memoria RAM este initializa cu 1 insemnand faptul ca la inceput consideram toate numerele ca fiind prime(Setarea are loc de la final la inceput ,Iar apoi incepem sa le setam cu 0 pe cele care sunt multipli de alte numere , numerele pare fiind primele care cad.Atentie v[0] corespunde lui data[3] deci la data(25) e stocat daca numrul 22 e prim sau nu l finalul algorimului**



**Mai jos se poate vedea cum se Numara daca numarul e prim sau nu (asta se face in registrul 5 care e initial e initializat cu 0.In registrul 1 se pot vedea numerele prime la care se face verificarea daca sunt prime sau nu ,iar in registrul 7 se poate vedea zona de memorie corespunzatorare numarului.**

****

**V-am lasat si Waveform-ul pentru mai multe detalii**