RAPORT MIPS CICLU UNIC

Hulea Andrei-Florin
Grupa 30225

1.Instructiuni suplimentare: xor si slt de tip R, ori si bne de tip I

- XOR : xor $rd,$rs,$rt
  - Face operatia de sau exclusiv intre registrii rs si rt, rezultatul fiind salvat in registrul rd
  - RTL abstract : RF[rd] <- RF[rs] ^ RF[rt]
  - 000_sss_ttt_ddd_0_110

- SLT : slt $rd,$rs,$rt
  - Daca rs este mai mic decat rt, atunci rd va primi 1, altfel 0
  - RTL abstract: if(RF[rs] < RF[rt]) then RF[rd] <- 1 else RF[rd] <- 0
  - 000_sss_ttt_ddd_0_111

- ORI : ori $rt,$rs,imm
  - rt primeste rezultatul aplicarii unui sau logic intre rs si imediat extins cu 0
  - RTL abstract: RF[rt] <- RF[rs] | Z_Ext(imm)
  - 101_sss_ttt_iiiiiii

- LUI : lui $rt,imm
  - rt primeste jumatatea de sus a imediatului
  - RTL abstract: RF[rt] <- imm || 0x0000
  - 110_000_ttt_iiiiiii

2.Semnale de control:

Semnale control MIPS16 pentru Anexa 5

<?> ∈ {GEZ, NE, GTZ}

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

Instrucțiune	**Opcode** *Instr(15-13)*	**RegDst**	**ExtOp**	**ALUSrc**	**Branch**	**Br<?>** (opțional)	**Jump**	**MemWrite**	**MemtoReg**	**RegWrite**	**ALUOp (1:0)**	**func** *Instr(2-0)*	**ALUCtrl (2:0)**	**JmpR** (opțional)
1.add	000	1	0	0	0		0	0	0	1	000	000	000	
2.sub	000	1	0	0	0		0	0	0	1	000	001	001	
3.sll	000	1	0	0	0		0	0	0	1	000	010	010	
4.srl	000	1	0	0	0		0	0	0	1	000	011	011	
5.and	000	1	0	0	0		0	0	0	1	000	100	100	
6.or	000	1	0	0	0		0	0	0	1	000	101	101	
7.xor	000	1	0	0	0		0	0	0	1	000	110	110	
8.slt	000	1	0	0	0		0	0	0	1	000	111	111	
9.addi	001	0	1	1	0		0	0	0	1	001(+)	xxx	000	
10.lw	010	0	1	1	0		0	0	1	1	001(+)	xxx	000	
11.sw	011	0	1	1	0		0	1	0	0	001(+)	xxx	000	
12.beq	100	0	1	0	1		0	0	0	0	010(-)	xxx	001	
13.ori	101	0	1	1	0		0	0	0	1	101(|)	xxx	101	
14.lui	110	0	1	1	0		0	0	0	1	001(+)	xxx	001	
15.j	111	0	1	0	0		1	0	0	0	Xxx	xxx	000	

URL: https://drive.google.com/open?id=1SI7x2Gp_2m3SEkwnXuGt4ns4voYzpGBH

3.Cod

Programul trebuie sa returneze daca suma numerelor din sirul lui Fibonacci dintr-un interval (a,b) este numar prim.

Cod C:

```
#include <iostream>

#include <stdlib.h>

using namespace std;

int main()

{

        cin>>a>>b;

   int t1=0, t2=1, nextTerm=0,suma = 0;;

        while(t2<=b) {

        nextTerm = t1 + t2;

        t1 = t2;

         t2 = nextTerm;

        if(t2>=a) suma = suma + t2;

}


   for (int i = 2; i < suma; i++){

        for (int j = i; j <= suma; j = j + i)

                if (j == n) ok = 0;

        }

        if (ok == 1)

                printf(" e prim\n");

        else printf("nu e prim\n");

}
```

Cod Assembly:

```
0       ADDI $s1, $s0, 1 #ultimul
1       ADDI $s2, $s0, 0 #penultimul
2       ADDI $s3, $s0, 6 #a
3        ADDI $s4, $s0, 30 #b
4       ADDI $s5, $s0, 0 #suma
5       ADDI $s7, $s0, 1 #pt verificare
6       loop:     ADDI $s6, $s0, 0 #verificare interval
7       SLT $s6, $s4, $s1 # daca a iesit din interval
8       BEQ $s6, $s7, verif_prim # sare la verif_prim
9       ADDI $s6, $s0, 0 #verificare interval
10      SLT $s6, $s3, $s1 # daca a intrat in interval
11      BEQ $s6, $s7, add_to_sum
12      ADD $s1, $s1, $s2 #ultimul
13      SUB $s2, $s1, $s2 #penultimul
14       J loop
15      add_to_sum: ADD $s1, $s1, $s2 #ultimul
16      SUB $s2, $s1, $s2 #penultimul
17      ADD $s5, $s5, $s2 #suma
18      J loop
19      verif_prim: ADDI $s1, $s0, 1 #counter
20      ADDI $s2, $s0, 0 #j
21      ADDI $s4, $s0, 0 #rez
22      ADDI $s7, $s0, 1 #verif
23      BEQ $s5, $s1, nu_e_prim
24      loop2     :ADDI $s1, $s1, 1
25      BEQ $s1, $s5, end # for i,n
```

```
26      ADD $s2, $s0, $s1 # for j=i,n,j=j+i
27      loop3:    BEQ $s2, $s5, nu_e_prim
28      ADD $s2, $s2, $s1 # j = j + i
29      ADDI $s6, $s0, 0 #verificare interval
30      SLT $s6, $s5, $s2
31      BEQ $s6, $s0, aici
32      J loop2
32      aici: J loop3
33      nu_e_prim: ADDI $s4, $s0, 0 # NOT_PRIME
34      J FINAL
35      end: ADDI $s4, $s0, 1 # IS_PRIME
36      FINAL: sw $s5,10($s0)
37      sw $s4,11($s0)
```

Trasarea executiei programului:

```
ADDI $s1, $s0, 1            -- RD1 = 0 , Ext_imm = 1 , ALURes = 1
ADDI $s2, $s0, 0            -- RD1 = 0 , Ext_imm = 0 , ALURes = -1
ADDI $s3, $s0, 6 #a         -- RD1 = 0 , Ext_imm =6, ALURes = 6
ADDI $s4, $s0, 30 #b        -- RD1 = 0 , Ext_imm = 30 , ALURes = 30
ADDI $s5, $s0, 0 #suma      -- RD1 = 0 , Ext_imm = 0 , ALURes = 0
ADDI $s7, $s0, 1 #pt verificare      -- RD1 = 0 , Ext_imm = 1 , ALURes = 1
loop:
   ADDI $s6, $s0, 0 #verificare interval       -- RD1 = 0 , Ext_imm = 0 , ALURes = 0
   SLT $s6, $s4, $s1 # daca a iesit din interval          -- RD1 = 30 , RD2 = 1 , ALURes = 0
   BEQ $s6, $s7, verif_prim # sare la verif_prim       -- RD1 = 0 , RD2= 1 , ALURes=-1 => next address
   ADDI $s6, $s0, 0 #verificare interval       -- RD1 = 0 , Ext_imm = 0 , ALURes = 0
   SLT $s6, $s3, $s1 # daca a intrat in interval          -- RD1 = 6 , RD2 = 1 , ALURes = 4
   BEQ $s6, $s7, add_to_sum        -- RD1 = 0 , RD2 = 1 , ALURes = -1 => next address
```

ADD $s1, $s1, $s2 #ultimu -- RD1 = 1 , RD2 = 0, ALURes = 1

SUB $s2, $s1, $s2 #penultimu -- RD1 = 1, RD2 =0 , ALURes = 1

J loop -- PC = offset(loop) = 6

loop:

ADDI $s6, $s0, 0 #verificare interval -- RD1 = 0 , Ext_imm = 0 , ALURes = 0

** SLT $s6, $s4, $s1 # daca a iesit din interval -- RD1 = 30 , RD2 = 0 , ALURes = 0

BEQ $s6, $s7, verif_prim # sare la verif_prim -- RD1 = 0 , RD2= 1 , ALURes=-1 => next address

ADDI $s6, $s0, 0 #verificare interval -- RD1 = 0 , Ext_imm = 0 , ALURes = 0

* SLT $s6, $s3, $s1 # daca a intrat in interval -- RD1 = 6 , RD2 = 0 , ALURes = 0

BEQ $s6, $s7, add_to_sum -- RD1 = 0 , RD2 = 1 , ALURes = -1 => next address

ADD $s1, $s1, $s2 #ultimul -- RD1 = 0 , RD2 = 1 , ALURes = 1

SUB $s2, $s1, $s2 #penultimul -- RD1 = 1 , RD2 =0 , ALURes = 1

J loop -- PC = offset(loop) = 6

.....

* SLT $s6, $s3, $s1 # daca a intrat in interval -- RD1 = 6 , RD2 = 8 , ALURes = 1

BEQ $s6, $s7, add_to_sum -- RD1 = 1 , RD2 = 1 , ALURes = 0 => add_to_sum

J loop -- sarim peste

add_to_sum:

ADD $s1, $s1, $s2 #ultimu -- RD1 = 8 , RD2 = 5 , ALURes = 13

SUB $s2, $s1, $s2 #penultimu -- RD1 = 13 , RD2 = 5 , ALURes = 8

ADD $s5, $s5, $s2 #suma -- RD1 = 0 , RD2 = 8 , ALURes = 8

J loop -- PC = offset(loop) = 6

.....

** SLT $s6, $s4, $s1 # daca a iesit din interval -- RD1 = 30 , RD2 = 34 , ALURes = 1

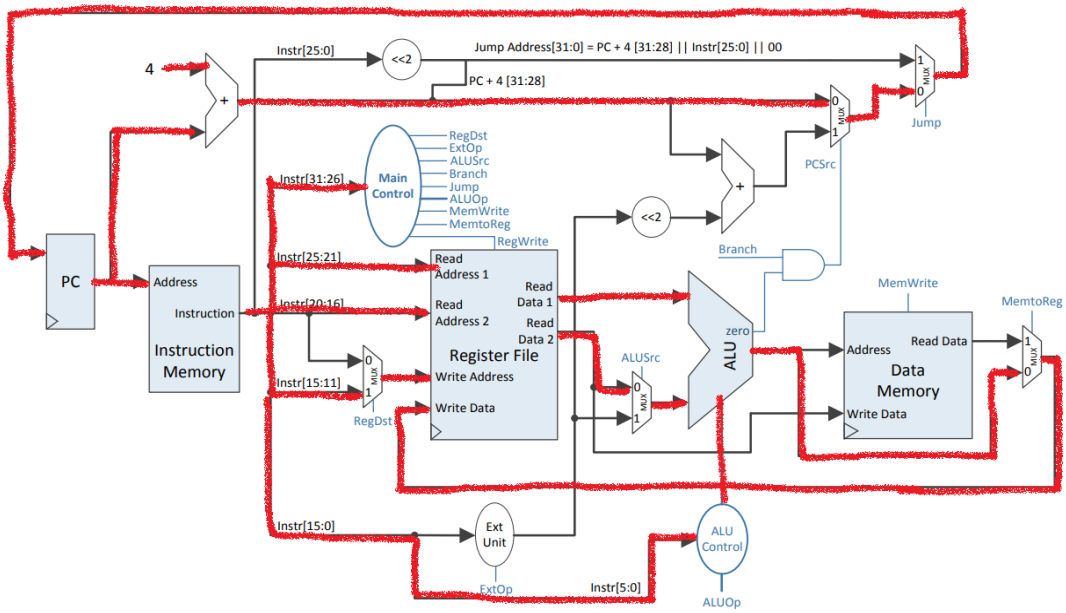
BEQ $s6, $s7, verif_prim # sare la verif_prim -- RD1 = 1 , RD2= 1 , ALURes=0 => verif_prim

verif_prim: (in acest moment suma din $s5 va fi 34, adica 13 + 21 si vom verifica daca e numar prim)

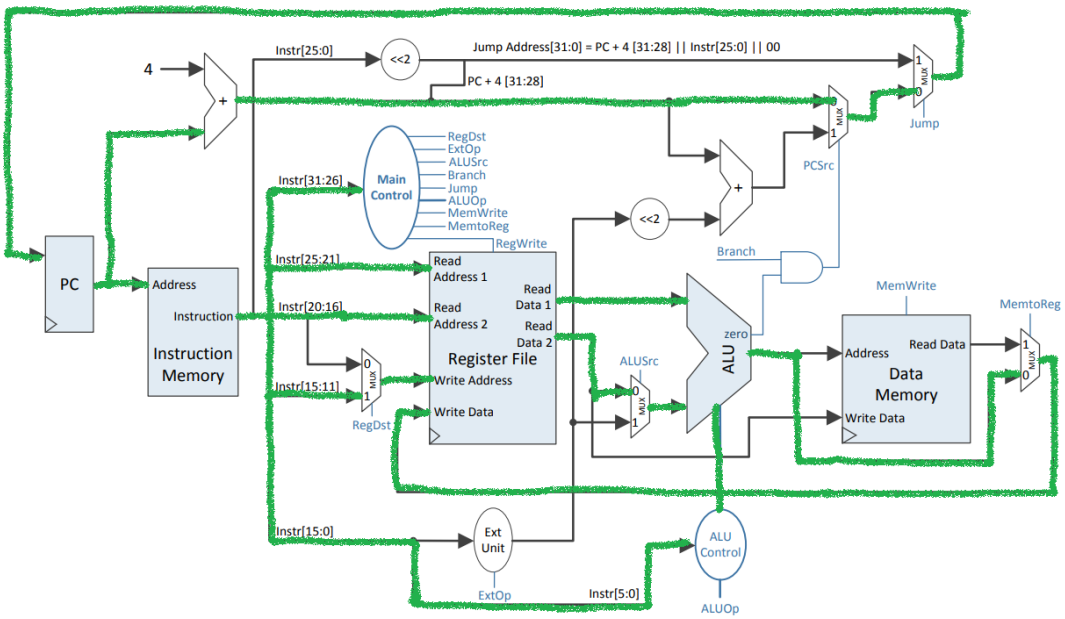
ADDI $s1, $s0, 1 #counter -- RD1 = 0 , Ext_imm = 1 , ALURes = 1

```
    ADDI $s2, $s0, 0 #j       -- RD1 = 0 , Ext_imm = 0 , ALURes = 1
    ADDI $s4, $s0, 0 #rez   --RD1 = 0 , Ext_imm = 0, ALURes = 0;
    ADDI $s7, $s0, 1 #verif --RD1 = 0 , Ext_imm = 1, ALURes = 1;
    BEQ $s5, $s1, nu_e_prim           --RD1 = 34 , RD2 = 1, ALURes = 33;
loop2:
    ADDI $s1, $s1, 1          --RD1 = 1 , Ext_imm = 1, ALURes = 2;
    BEQ $s1, $s5, end # for i,n         --RD1 = 2 , Ext_imm = 34, ALURes = -32;
    ADD $s2, $s0, $s1 # for j=i,n,j=j+i          --RD1 = 0 , RD2 = 2, ALURes = 2;
    loop3:
        BEQ $s2, $s5, nu_e_prim       --RD1 = 2 , RD2 = 34, ALURes = -32;
        ADD $s2, $s2, $s1 # j = j + i  --RD1 = 2 , RD2 = 2, ALURes = 4;
       ADDI $s6, $s0, 0 #verificare interval     --RD1 = 0 , Ext_imm = 0, ALURes = 0;
       SLT $s6, $s5, $s2       --RD1 = 34 , RD2 = 4, ALURes = 0;
       BEQ $s6, $s0,aici       --RD1 = 0 , RD2 = 0, ALURes = 0 => next address = aici
      J loop3 – PC = offset(loop3) = 27
aici :  J loop2 –PC=offset(loop2) = 24
.....
Programul continua sa verifice daca suma noastra are multiplii
Se va opri cand $s2 va ajunge la valoarea 34, deci numarul nu este prim deoarece e divizibil cu 2
.....
nu_e_prim:
            ADDI $s4, $s0, 0 # NOT_PRIME --RD1 = 0 , Ext_imm = 0, ALURes = 0;
    J FINAL
end:
            ADDI $s4, $s0, 1 # IS_PRIME        --sarim peste
FINAL:  sw $s5,10($s1)    --stocam suma din $s5 pentru a o putea verifica la sfarsitul programului
Sw $s4,11($s0)    --stocam si pe $s4 pentru a verifica daca e prim. $s4=0, deci numarul nu este prim
```

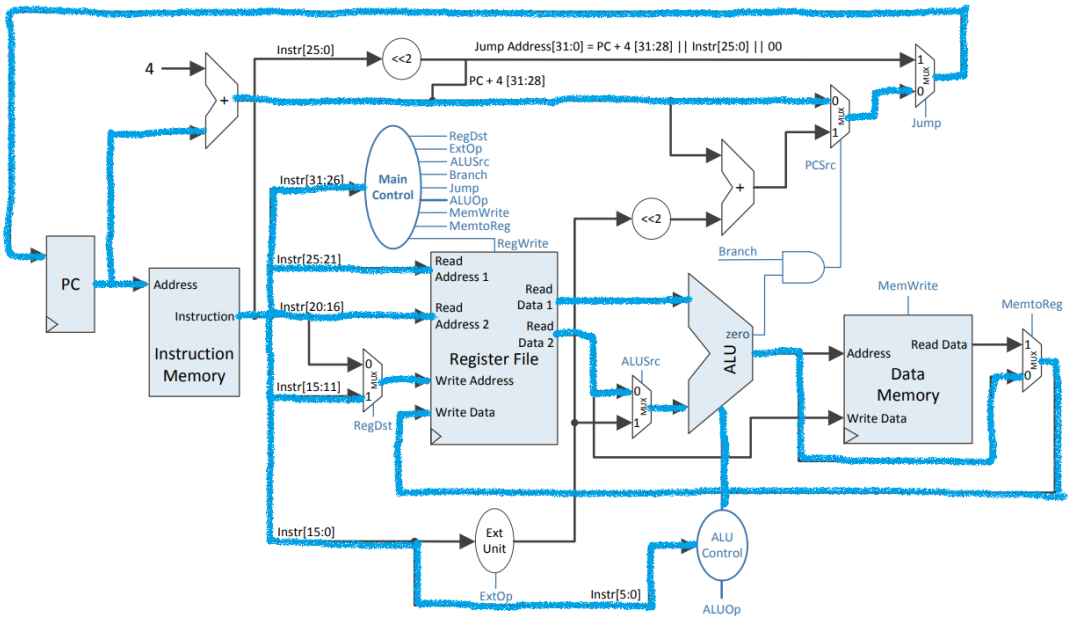
add


4
+
Instr[25:0]
<<2
Jump Address[31:0] = PC + 4 [31:28] || Instr[25:0] || 00
PC + 4 [31:28]
MUX
Jump
PCSrc
Main Control
RegDst
ExtOp
ALUSrc
Branch
Jump
ALUOp
MemWrite
MemtoReg
RegWrite
Instr[31:26]
Branch
PC
Address
Instruction
Instruction Memory
Instr[25:21]
Instr[20:16]
Instr[15:11]
Read Address 1
Read Address 2
Write Address
Write Data
Read Data 1
Read Data 2
Register File
RegDst
ALUSrc
zero
ALU
MemWrite
MemtoReg
Address
Read Data
Data Memory
Write Data
Instr[15:0]
Ext Unit
ExtOp
Instr[5:0]
ALU Control
ALUOp

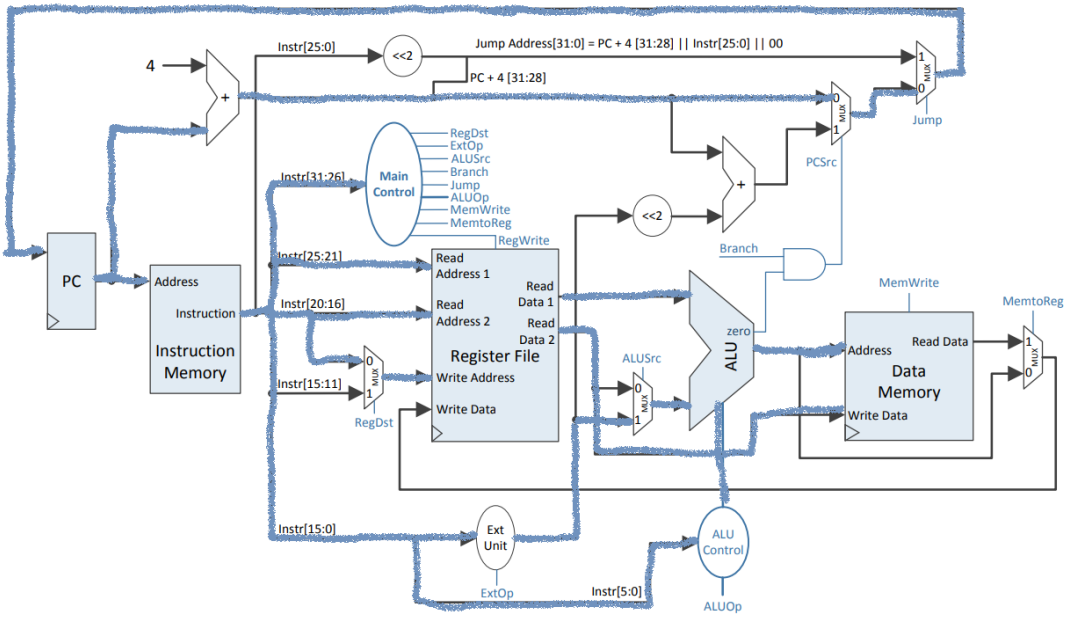
sub


4
+
Instr[25:0]
<<2
Jump Address[31:0] = PC + 4 [31:28] || Instr[25:0] || 00
PC + 4 [31:28]
MUX
Jump
PCSrc
Main Control
RegDst
ExtOp
ALUSrc
Branch
Jump
ALUOp
MemWrite
MemtoReg
RegWrite
Instr[31:26]
Branch
PC
Address
Instruction
Instruction Memory
Instr[25:21]
Instr[20:16]
Instr[15:11]
Read Address 1
Read Address 2
Write Address
Write Data
Read Data 1
Read Data 2
Register File
RegDst
ALUSrc
zero
ALU
MemWrite
MemtoReg
Address
Read Data
Data Memory
Write Data
Instr[15:0]
Ext Unit
ExtOp
Instr[5:0]
ALU Control
ALUOp

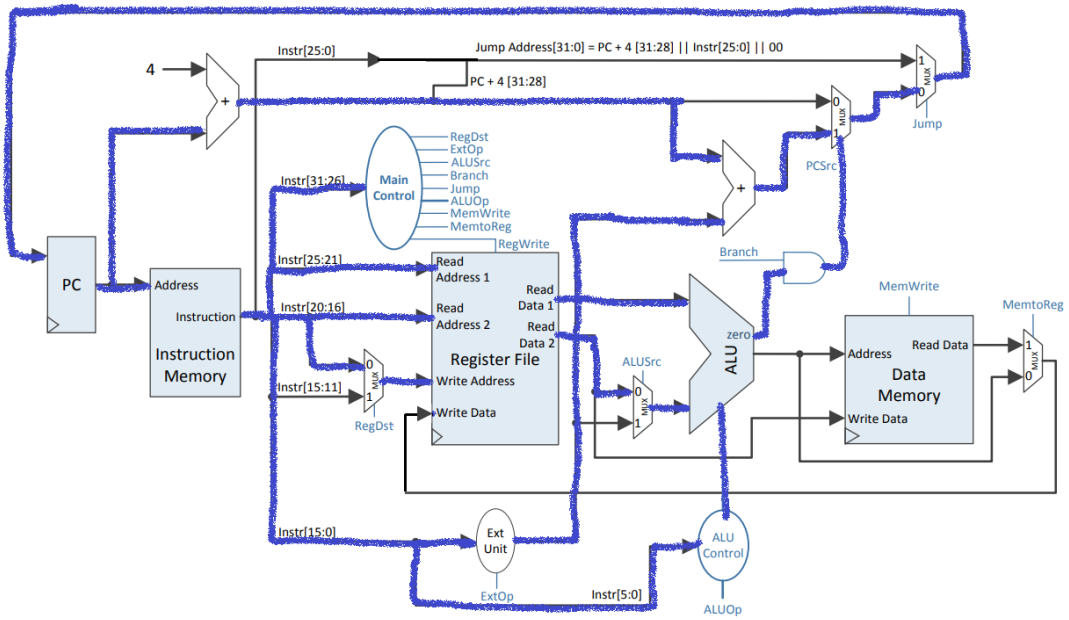
slt


4
+
Instr[25:0]
<<2
Jump Address[31:0] = PC + 4 [31:28] || Instr[25:0] || 00
PC + 4 [31:28]
Jump
PCSrc
Main Control
RegDst
ExtOp
ALUSrc
Branch
Jump
ALUOp
MemWrite
MemtoReg
RegWrite
Instr[31:26]
Instr[25:21]
Instr[20:16]
Instr[15:11]
Instr[15:0]
Instr[5:0]
PC
Address
Instruction
Instruction Memory
Read Address 1
Read Address 2
Write Address
Write Data
Read Data 1
Read Data 2
Register File
RegDst
Branch
ALUSrc
zero
ALU
Address
Read Data
Write Data
Data Memory
MemWrite
MemtoReg
Ext Unit
ExtOp
ALU Control
ALUOp
MUX

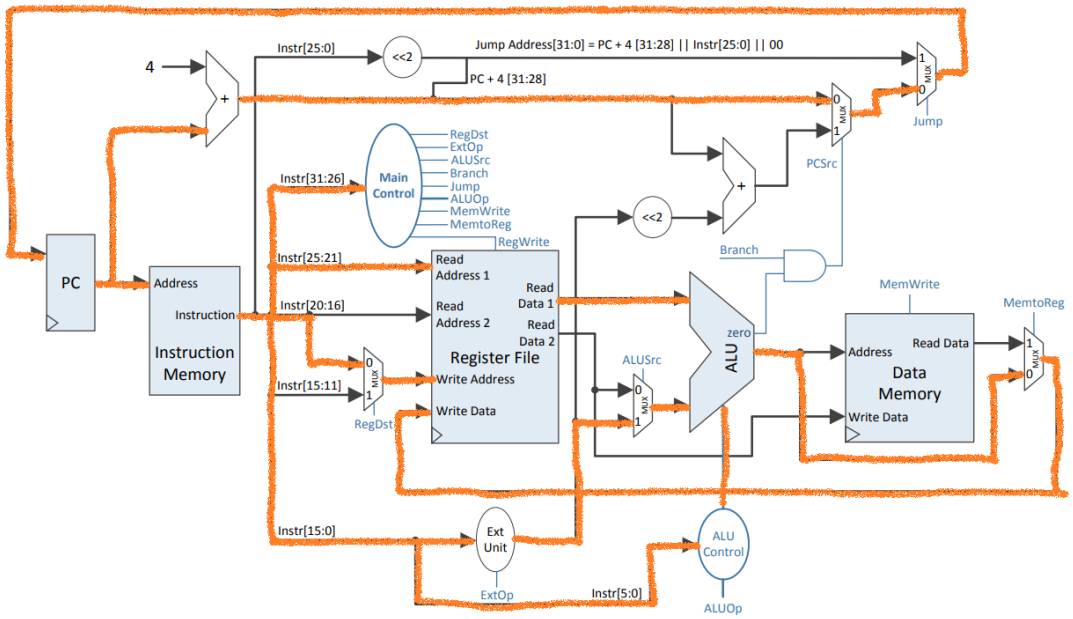
sw


4
+
Instr[25:0]
<<2
Jump Address[31:0] = PC + 4 [31:28] || Instr[25:0] || 00
PC + 4 [31:28]
Jump
PCSrc
Main Control
RegDst
ExtOp
ALUSrc
Branch
Jump
ALUOp
MemWrite
MemtoReg
RegWrite
Instr[31:26]
Instr[25:21]
Instr[20:16]
Instr[15:11]
Instr[15:0]
Instr[5:0]
PC
Address
Instruction
Instruction Memory
Read Address 1
Read Address 2
Write Address
Write Data
Read Data 1
Read Data 2
Register File
RegDst
Branch
ALUSrc
zero
ALU
Address
Read Data
Write Data
Data Memory
MemWrite
MemtoReg
Ext Unit
ExtOp
ALU Control
ALUOp
MUX

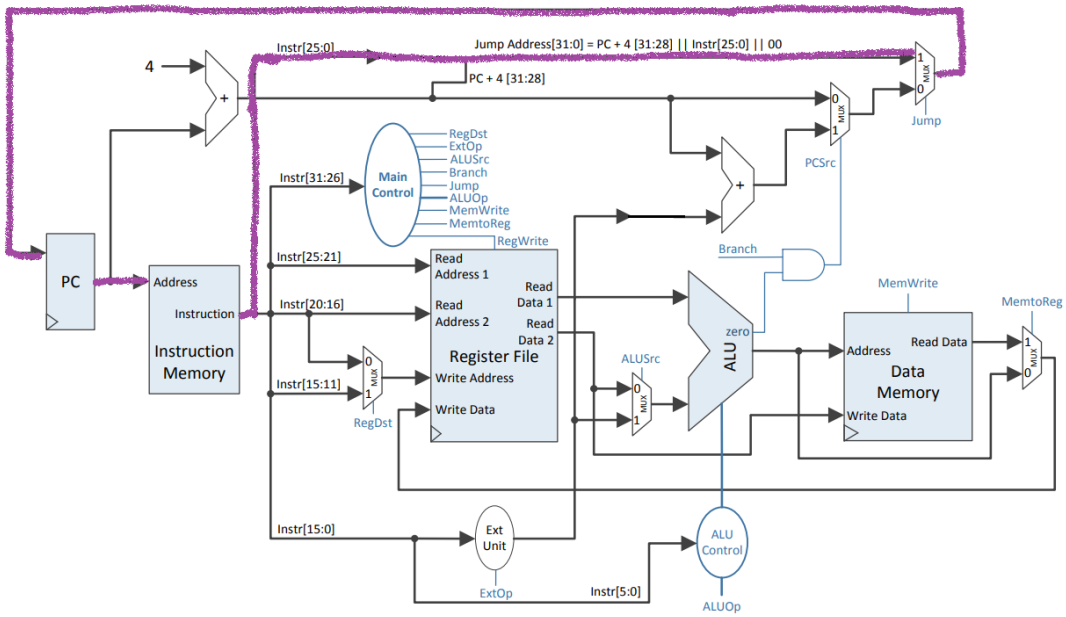
beq


4
Instr[25:0]
Jump Address[31:0] = PC + 4 [31:28] || Instr[25:0] || 00
PC + 4 [31:28]
Jump
PCSrc
Branch
RegDst
ExtOp
ALUSrc
Branch
Jump
ALUOp
MemWrite
MemtoReg
RegWrite
Main Control
Instr[31:26]
PC
Address
Instruction
Instruction Memory
Instr[25:21]
Instr[20:16]
Instr[15:11]
Read Address 1
Read Address 2
Write Address
Write Data
Read Data 1
Read Data 2
Register File
RegDst
ALUSrc
zero
ALU
MemWrite
Address
Read Data
Data Memory
Write Data
MemtoReg
Instr[15:0]
Ext Unit
ExtOp
ALU Control
Instr[5:0]
ALUOp

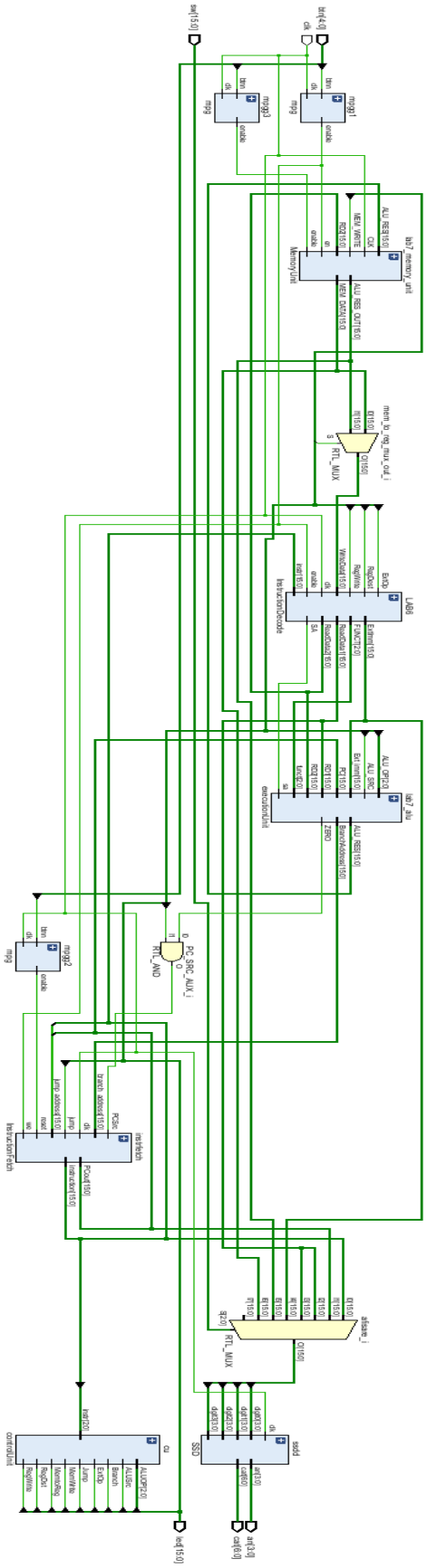
addi


4
Instr[25:0]
<<2
Jump Address[31:0] = PC + 4 [31:28] || Instr[25:0] || 00
PC + 4 [31:28]
Jump
PCSrc
Branch
RegDst
ExtOp
ALUSrc
Branch
Jump
ALUOp
MemWrite
MemtoReg
RegWrite
Main Control
Instr[31:26]
PC
Address
Instruction
Instruction Memory
Instr[25:21]
Instr[20:16]
Instr[15:11]
Read Address 1
Read Address 2
Write Address
Write Data
Read Data 1
Read Data 2
Register File
RegDst
ALUSrc
zero
ALU
MemWrite
Address
Read Data
Data Memory
Write Data
MemtoReg
Instr[15:0]
Ext Unit
ExtOp
ALU Control
Instr[5:0]
ALUOp

j



Observatie: in codul din vivado am pus intervalul (a,b) mai mic pentru a putea fi mai usor verificat.


sw[15:0]
btn[4:0]
clk
mpg3
mpg1
mpg2
ALU_RES[15:0]
MEM_WRITE
CLK
MEM_DATA[15:0]
RegWrite
RegDst
MemToReg
Jump
MemWrite
Branch
ExtOp
ALUSrc
ALUOP[2:0]
RD1[15:0]
RD2[15:0]
ZERO
PC_SRC_AUX_i
RTL_AND
RTL_MUX
PCout[15:0]
instruction[15:0]
branch_address[15:0]
jump_address[15:0]
an[3:0]
cat[6:0]
led[15:0]