**Configurare registre MIPS16 Pipeline – varianta 1**

Se introduc pe coloane semnalele de date și control mapate la registre, de sus în jos, începând de la biții cei mai semnificativi ai registrului către cei mai puțin semnificativi. Se introduc în paranteză biții din registru alocați pentru fiecare semnal în parte. În dreptul numelui registrelor din primul rând se introduce în paranteză poziția bitului cel mai semnificativ (<msb>) din care reiese dimensiunea totală alocată registrului.

|  |  |  |  |
| --- | --- | --- | --- |
| **REG\_IF\_ID(31** – **0)** | **REG\_ID\_EX(<75>** – **0>)** | **REG\_EX\_MEM(<52>** – **0)** | **REG\_MEM\_WB(<33>** – **0)** |
| PC + 1(31 – 16) | MemtoReg(75) | MemtoReg(52) | MemtoReg(33) |
| Instruction(15 – 0) | RegWrite (74) | RegWrite (51) | RegWrite (32) |
|  | MemWrite(73) | MemWrite(50) | MemData (31 downto 16) |
|  | Branch(72) | Branch(49) | alures2 (15 downto 0) |
|  | ALUOp(71 downto 69) | BranchAdress (48 downto 33) |  |
|  | ALUSrc(68) | zero (32) |  |
|  | PC + 1 (67 downto 52) | alures (31 downto 16) |  |
|  | RD1 (51 downto 36) | Rd2 (15 downto 0) |  |
|  | RD2 (35 downto 20) |  |  |
|  | ext\_imm (19 downto 4) |  |  |
|  | Func (3 downto 1) |  |  |
|  | Sa(0) |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

URL: <https://drive.google.com/file/d/1Yw18tfjFjo-v897vVpd8KNGnpx5U8i9n/view?usp=sharing>

InF

In inf avem registrul PC , memoria ROM un ALU care adauga 1 la iesirea registrului , si doua MUX uri , pentru BranchAdress respectiv JumpAdress

Are ca intrari clk Jenable PCsrc reset enable JAdress BAdress

Are ca iesiri PCplus instruction

PCplus returneaza outputul registrului PC + 1

Instruction reprezinta informatia stocata in ROM

ID( si RF)

In ID se afla RF-ul ,operatia de extindere si un mux pentru Write Adress

Are ca intrari clk RegWrite Instr RegDst ExtOp WD

Are ca iesiri Ext\_imm func sa RD1 RD2

Ext\_imm returneaza instr(15 downto 9) extins cu 0 sau cu semn in functie de valoarea lui ExtOp

Func returnaza Instr(2 downto 0)

Sa returneaza Instr(12);

RD1 si RD2 vor primi valorile aflate la Read adress 1 si read adress 2 prin intermesiul RF

EX

EX este entitatea responsabila cu efectuarea operatilor de catre procesor. In aceasta se afla AluCtrl care primeste valori in functie de valorile lui AluOp si func astfel codificand fiecare operatie, dupa care ALU efectueaza operatiile in functie codul primit de la ALUCtrl

RAM (MEM)

Aici se afla memoria RAM

Are ca intrari PCinc Ext\_Imm func sa RD1 RD2 ALUSrc ALUOp

Are ca iesiri zero ALURes BranchAdress

Lui branch adress i se atribuie PCinc + Ext\_imm

Alures primeste valori in functie de AluCtrl care este un semnal care primeste valori la randul lui in functie de ALUOp si func

In functie de valorile lui AluCtrl ALU efectueaza urmatoarele operatii

Add adunare intre RD1 si RD2

Sub scadere intre RD1 si RD2

Sll shiftare la stanga cu 1 bit

Srl shiftare la dreapta cu 1 bit

And functia and intre RD1 si RD2

Or functia or intre RD1 si RD2

Xor functia xor intre RD1 si RD2

Sllv functia and intre RD1 si Ext\_imm

Addi adunare intre intre RD1 si Ext\_imm

Lw load word

Sw store word

Beq salt conditionat daca este egalitate intre doua registre

Xori xor intre RD1 si Ext\_imm

Ori or intre RD1 si Ext\_imm

UC

Entitatea UC este responsabila cu valorile semnalelor de control in functie de bitii 15 downto 13 a instructiunilor primite de la PC

Are ca intrare Instr

Are ca iesiri RegDst ExtOp ALUSrc Branch Jump ALUOp MemWrite MemToReg RegWrite

MIPS

Este entitatea in care legam entitatile intre ele . de asemenea aici ii atribuim lui PCsrc branch and zero , calculam jump adress ,ii atribuim lui Write data alures sau MemData in functie de valoarea semnalului MemtoReg ,hotaram ce avem afisat pe SSD in functie de bitii 7 downto 5 a lui sw si atribuim ledurilor cate un semnal pentru a putea tine evidenta lor mai usor

Diferenta dintre acest mips si mips pipeline este faptul ca in entitatea mips(la pipeline) avem prezente 4 registre

IF\_and\_ID: std\_logic\_vector(31 downto 0);

signal MEM\_and\_WD: std\_logic\_vector(33 downto 0);

signal EX\_and\_MEM: std\_logic\_vector(20 downto 0);

signal ID\_and\_EX: std\_logic\_vector(76 downto 0);

care au rolul de a memora anumite semnale si iesiri intre etaje

--B"001\_000\_001\_0010100" --ADDI $1, $0, 20

--B"001\_000\_010\_0000101" --ADDI $2, $0, 5

--B"001\_000\_100\_0001110" --ADDI $4, $0, 14

--B"000\_000\_000\_0000000", -- NoOp

--B"000\_000\_000\_0000000", -- NoOp

--B"000\_001\_010\_011\_0\_001" --SUB $3, $1, $2

--B"011\_000\_010\_0000000" --SW $3, 0($0)

--B"010\_000\_101\_0000000" --LW $5,0($0)

--B"000\_000\_000\_0000000", -- NoOp

--B"000\_000\_000\_0000000", -- NoOp

--B"000\_101\_100\_110\_0\_100" --AND $6,$5,$4

--B"000\_000\_000\_0000000", -- NoOp

--B"110\_110\_111\_0000101" --XORi $7 $6 5

--B"000\_000\_000\_0000000", -- NoOp

--B"000\_111\_010\_110\_0\_001" --SUB $6, $7, $2

--B"111\_0000000000001" --j 1