Documentatie proiect – implementare MIPS 32, ciclu unic

1. Descriere generala

Proiectul consta in implementarea unui procesor MIPS simplificat utilizand limbajul VHDL, testat si sintetizat pe placa de dezvoltare Basys3. Proiectul urmeaza modelul arhitecturii clasice MIPS pe 5 etape: IF (Instruction Fetch), ID (Instruction Decode), EX (Execute), MEM (Memory Access) si WB (Write Back).

2. Componente functionale

Implementarea este structurata modular, utilizand urmatoarele fisiere:

2.1 Ifetch.vhd – Instruction Fetch

ROL: aduce instructiunea curenta din memoria de instructiuni

Elemente:

- Registru PC (Program Counter) care se incrementeaza cu 4 la fiecare ciclu de ceas
- Memorie ROM ce contine instructiunile programului

Ofera ca iesire instructiunea catre etapa de decodificare si noua valoare a PC-ului.

2.2 ID.vhd – Instruction Decode

ROL: decodifica instructiunea, extrage campurile importante (OpCode, registre, valori imediate)

Primeste instructiunea de 32 de biti de la IF si semnalele de control de la UC si trimite catre EX operanzii si semnalele relevante.

2.3 RegFile.vhd – Register File

- Contine 32 de registre de cate 32 de biti
- Suporta citirea a doua registre si scrierea intr-unul singur in acelasi ciclu de ceas

2.4 UC.vhd – Unit of Control

ROL: genereaza semnale de control in functie de opcode-ul instructiunii

Converteste instructiunea in semnale precum: RegWrite, ALUSrc, MemRead, Branch etc. Astfel permite selectarea si coordonarea fluxului de date

2.5 MUX.vhd – Multiplexor

Un multiplexor parametrizat pentru a putea fi folosit oriunde este nevoie a alege intre mai multe surse de date.

Exemple:

- Selectarea intre valoarea imediata si registru pentru ALU
- Selectarea intre valoarea din memorie si rezultatul ALU pentru scrierea in registru

2.6 EX.vhd – Execute

- Contine ALU care executa operatii aritmetice si logice
- Primeste datele de la ID si controleaza semnalul Zero pentru instructiunea de tip branch
- Efectueaza si calculul adresei de memorie pentru lw/sw

2.7 MEM.vhd – Memory Access

Simuleaza o memorie RAM pentru date.

Permite:

- Citirea: pentru lw, datele sunt trimise catre WB
- Scrierea: pentru sw, datele din registru sunt stocate in memorie

2.8 MPG.vhd – Debounce pentru butoane (Multiple Pulse Generator)

ROL: elimina efectul de bouncing la apasarea butoanelor fizice de pe placa. De asemenea este folosit pentru a inregistra tranzitii clare de la utilizator (ex: trecerea la urmatoarea instructiune)

2.9 View_instructions_on_basys.vhd – Afisare instructiuni

Converteste instructiunea MIPS intr-un format afisabil pe cele 4 afisoare cu 7 segmente.

Utilizeaza un switch pentru a selecta partea inferioara (biti 15 – 0) sau superioara (biti 31 – 16) ai instructiunii, astfel intregul cuvant de 32 de biti sa poata fi vizualizat in doua parti.

Usureaza depanarea si monotorizarea instrucituniilor direct pe placa.

2.10 Test_env.vhd – Mediu de test

Integreaza toate componentele de mai sus intr-o schema de ansamblu si permite simularea si verificarea functionarii corecte a instructiuniilor.

3. Probleme intampinate

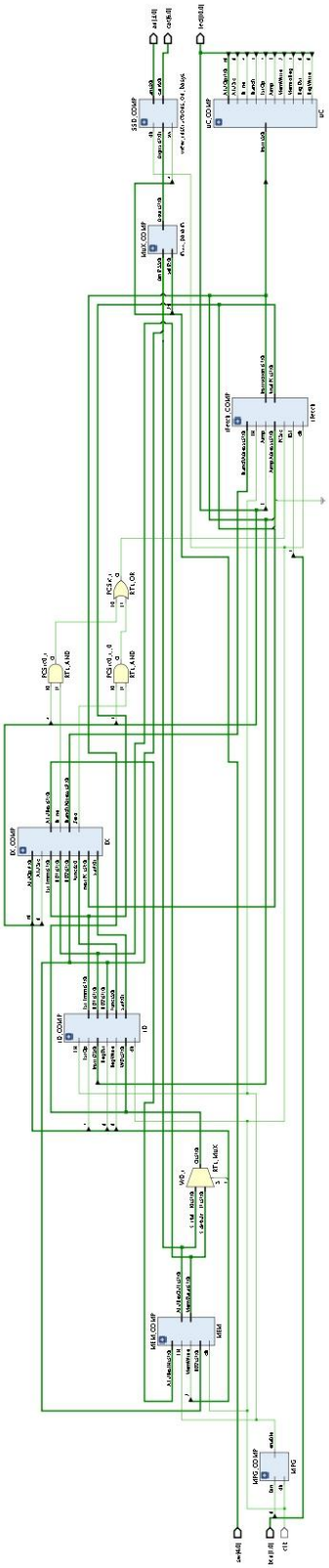
- **Limitarea afisajului:** Basys3 dispune de 4 afisoare cu 7 segmente ceea ce a impus implementarea unui mecanism de comutare pentru vizualizarea completa a instructiunii
- **Sincronizare cu ceasul:** a fost necesar un modul de debounce pentru butoane (MPG.vhd) deoarece semnalele de intrare generate manual pot provoca glitch-uri
- **Decodificarea instrucțiunilor:** Unele erori inițiale în semnalele de control au dus la comportamente neașteptate, rezolvate prin debug
- **Scrierea incorecta in registre:** Semnalul RegWrite nu era activat corect sau adresa de scriere nu era valida. Am testat fiecare instructiune R-type si am verificat ca semnalele de adresare, date si control erau corect propagate.

4. Semnale de control

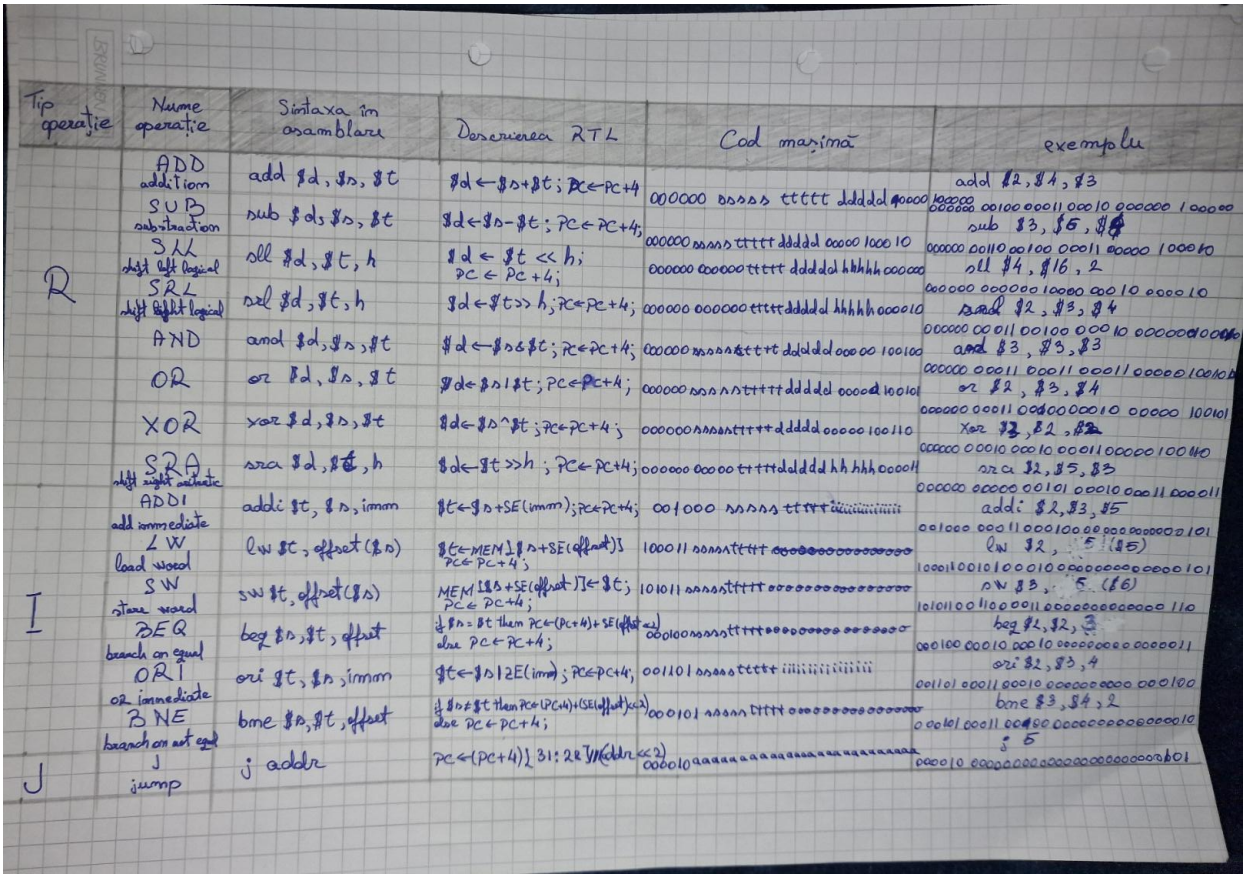
Unitatea de control este responsabila pentru generarea semnalelor care ghideaza executia fiecarei instructiuni. Aceste semnale influenteaza selectarea datelor, activarea scrierilor si citirilor in registre/memorie si determinarea salturilor conditionate.

SEMNAL	DESCRIERE
RegWrite	Activeaza scrierea in registrul destinatie in etapa WB
MemRead	Permite citirea din memoria de date (util pentru instructiunea lw)
MemWrite	Permite scrierea in memoria de date (util pentru instructiunea sw)
MemtoReg	Selecteaza daca valoarea scrisa in registru vine din memorie sau din ALU
ALUSrc	Alege intre operandul 2 din registru si o valoare imediata pentru ALU
ALUOp	Codifica tipul de operatie aritmetica/logica
Branch	Activeaza logica de salt conditionat
Jump	Activeaza saltul neconditionat
Br_ne	Semnal specific pentru branch daca nu sunt egale. Se activeaza daca operand1 != operand2

5. RTL schematic generat in vivado



6. Descriere instructiuni



Tip operație	Nume operație	Sintaxa în asamblare	Descrierea RTL	Cod mașină	exemplu
R	ADD addition	add $d, $s, $t	$d ← $s + $t; PC ← PC+4	000000 sssss ttttt ddddd 00000 100000	add $2, $1, $3
	SUB substraction	sub $d, $s, $t	$d ← $s − $t; PC ← PC+4;	000000 sssss ttttt ddddd 00000 100010	sub $3, $5, $6
	SLL shift left logical	sll $d, $t, h	$d ← $t << h; PC ← PC+4;	000000 00000 ttttt ddddd hhhhh 000000	sll $4, $16, 2
	SRL shift right logical	srl $d, $t, h	$d ← $t >> h; PC ← PC+4;	000000 00000 ttttt ddddd hhhhh 000010	srl $2, $3, $4
	AND	and $d, $s, $t	$d ← $s & $t; PC ← PC+4;	000000 sssss ttttt ddddd 00000 100100	and $3, $3, $3
	OR	or $d, $s, $t	$d ← $s | $t; PC ← PC+4;	000000 sssss ttttt ddddd 00000 100101	or $2, $3, $4
	XOR	xor $d, $s, $t	$d ← $s ^ $t; PC ← PC+4;	000000 sssss ttttt ddddd 00000 100110	xor $3, $2, $2
	SRA shift right aritmetic	sra $d, $t, h	$d ← $t >> h; PC ← PC+4;	000000 00000 ttttt ddddd hhhhh 000011	sra $2, $5, 3
I	ADDI add immediate	addi $t, $s, imm	$t ← $s + SE(imm); PC ← PC+4;	001000 sssss ttttt iiiiiiiiiiiiiiii	addi $2, $3, 5
	LW load word	lw $t, offset($s)	$t ← MEM[$s + SE(offset)]; PC ← PC+4;	100011 sssss ttttt oooooooooooooooo	lw $2, 5($5)
	SW store word	sw $t, offset($s)	MEM[$s + SE(offset)] ← $t; PC ← PC+4;	101011 sssss ttttt oooooooooooooooo	sw $3, 5($6)
	BEQ branch on equal	beq $s, $t, offset	if $s = $t then PC ← (PC+4) + SE(offset << 2) else PC ← PC+4;	000100 sssss ttttt oooooooooooooooo	beq $2, $2, 3
	ORI OR immediate	ori $t, $s, imm	$t ← $s | ZE(imm); PC ← PC+4;	001101 sssss ttttt iiiiiiiiiiiiiiii	ori $2, $3, 4
	BNE branch on not equal	bne $s, $t, offset	if $s ≠ $t then PC ← (PC+4) + (SE(offset) << 2) else PC ← PC+4;	000101 sssss ttttt oooooooooooooooo	bne $3, $4, 2
J	J jump	j addr	PC ← (PC+4)[31:28] ∥ (addr << 2)	000010 aaaaaaaaaaaaaaaaaaaaaaaaaa	j 5

SRA – Shift Right Aritmetic: realizeaza o deplasare aritmetica la dreapta, pastrand semnul

XOR – Exclusive OR: realizeaza XOR bit cu bit intre doua registre

ORI – OR Immediate: realizeaza un OR logic intre un registru si o valoare imediata zero-extinsa

BNE – Branch on Not Equal: sare la o eticheta daca doua registre NU sunt egale

7. Viitoare imbunatatiri

- Implementarea hazardelor și forwarding pentru execuție mai eficientă în pipeline.
- Adăugarea unor instrucțiuni suplimentare (and, or, slt, bne etc.).
- Extinderea memoriei pentru programe mai complexe.