Raport

1. Modificări suplimentare

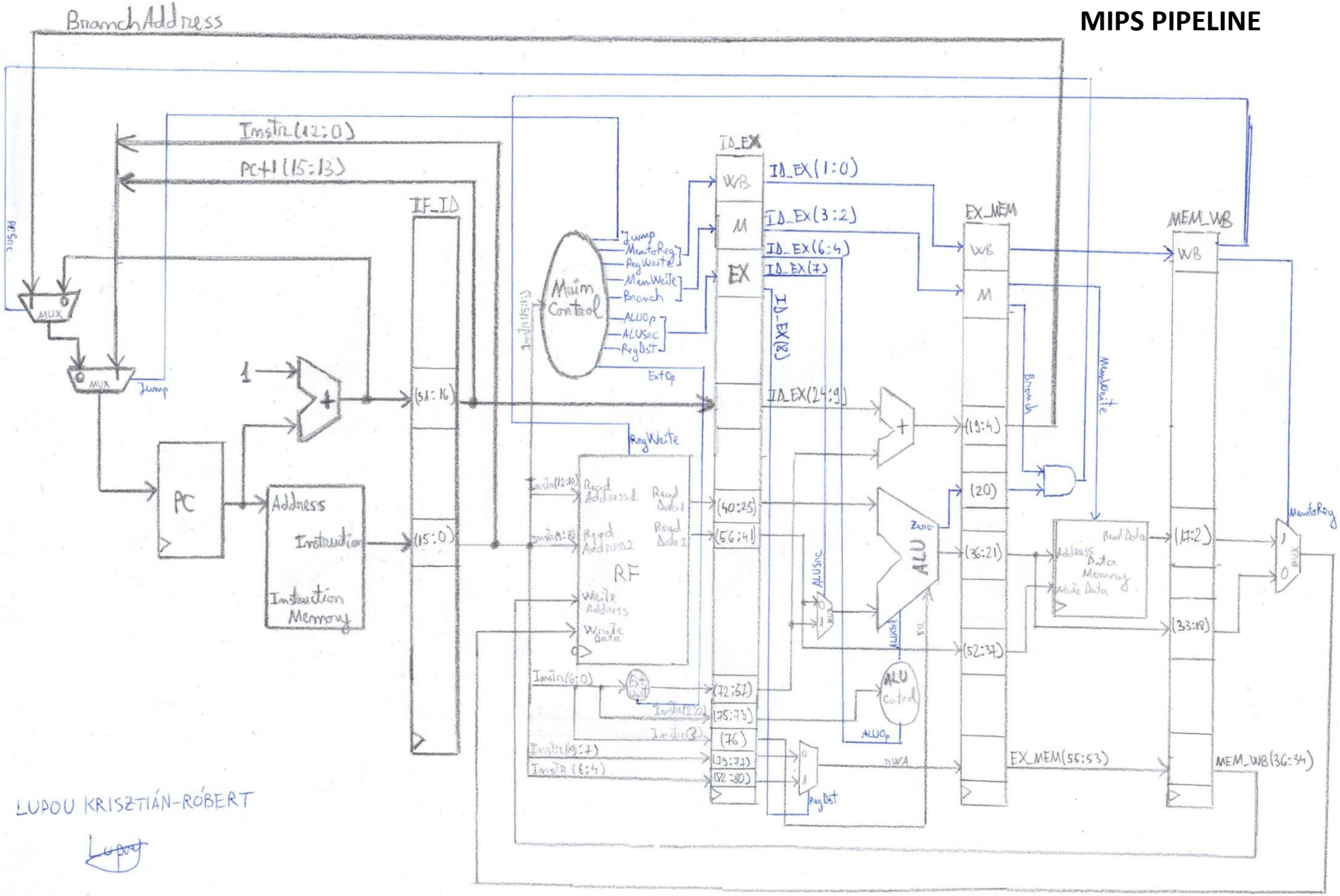
Nu au fost făcute modificări suplimentare pentru cele 4 instrucțiuni alese (XOR, SRA, ANDI, ORI).

2. 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(82 – 0)**	**REG_EX_MEM(55 – 0)**	**REG_MEM_WB(36 – 0)**
Instruction(31 – 16)	rd(82 – 80)	rWA(55 – 53)	rWA(36 – 34)
PC + 1(15 – 0)	rt(79 – 77)	RD2(52 – 37)	ALUResOut(33 – 18)
	sa(76)	ALURes(36 – 21)	MemData(17 – 2)
	func(75 – 73)	Zero(20)	RegWrite(1)
	Ext_Imm(72 – 57)	branchAddr(19 – 4)	MemToReg(0)
	RD2(56 – 41)	Branch(3)	
	RD1(40 – 25)	MemWrite(2)	
	PC + 1(24 – 9)	RegWrite(1)	
	RegDst(8)	MemToReg(0)	
	ALUSrc(7)		
	ALUOp(6 – 4)		
	Branch(3)		
	MemWrite(2)		
	RegWrite(1)		
	MemToReg(0)		

3. Schema procesorului MIPS PIPELINE



LUPOU KRISZTIÁN-RÓBERT

4. Identificarea hazardurilor

Programul executat de procesor face suma primelor 6 numere impare. Rezultatul va fi reținut în register file la adresa 7.
Programul inițial în MIPS 16 ciclu unic:

Adr.	Instrucțiune
0	addi $1,$0,2
1	addi $2,$0,1
2	addi $3,$0,3
3	addi $4,$0,5
4	sw $2,0($2)
5	sw $3,0($1)
6	lw $5,0($1)
7	lw $6,0($2)
8	add $7,$5,$6
9	addi $6,$0,1
10	add $5,$3,$1
11	add $7,$7,$5
12	addi $3,$5,0
13	addi $6,$6,1
14	beq $6,$4,1
15	j 10

Hazarduri identificate:

- între instrucțiunile 1 și 4 este hazard structural după reg $2
- între instrucțiunile 2 și 5 este hazard structural după reg $3
- între instrucțiunile 6 și 8 este hazard RAW după reg $5
- între instrucțiunile 7 și 8 este hazard RAW după reg $6
- între instrucțiunile 8 și 11 este hazard structural după reg $7
- între instrucțiunile 10 și 11 este hazard RAW după reg $5
- între instrucțiunile 10 și 12 este hazard RAW după reg $5
- între instrucțiunile 13 și 14 este hazard RAW după reg $6
- la instrucțiunea 14 este hazard de control
- la instrucțiunea 15 este hazard de control

Diagrama de pipeline:

Instr/Clk	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
addi $1,$0,2	IF	ID	EX	MEM	WB				
addi $2,$0,1		IF	ID	EX	MEM	WB			
addi $3,$0,3			IF	ID	EX	MEM	WB		
addi $4,$0,5				IF	ID	EX	MEM	WB	
sw $2,0($2)					IF	ID	EX	MEM	WB

Instr/Clk	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12
addi $4,$0,5	IF	ID	EX	MEM	WB				
sw $2,0($2)		IF	ID	EX	MEM	WB			
sw $3,0($1)			IF	ID	EX	MEM	WB		
lw $5,0($1)				IF	ID	EX	MEM	WB($5)	
lw $6,0($2)					IF	ID	EX	MEM	WB($6)

Instr/Clk	CC7	CC8	CC9	CC10	CC11	CC12	CC13	CC14	CC15
lw $5,0($1)	IF	ID	EX	MEM	WB($5)				
lw $6,0($2)		IF	ID	EX	MEM	WB($6)			
add $7,$5,$6			IF	ID($5,$6)	EX	MEM	WB		
addi $6,$0,1				IF	ID	EX	MEM	WB	
add $5,$3,$1					IF	ID	EX	MEM	WB($5)

Instr/Clk	CC10	CC11	CC12	CC13	CC14	CC15	CC16	CC17	CC18
addi $6,$0,1	IF	ID	EX	MEM	WB				
add $5,$3,$1		IF	ID	EX	MEM	WB($5)			
add $7,$7,$5			IF	ID($5)	EX	MEM	WB		
addi $3,$5,0				IF	ID($5)	EX	MEM	WB	
addi $6,$6,1					IF	ID	EX	MEM	WB

Instr/Clk	CC13	CC14	CC15	CC16	CC17	CC18	CC19	CC20
addi $3,$5,0	IF	ID	EX	MEM	WB			
addi $6,$6,1		IF	ID	EX	MEM	WB($6)		
beq $6,$4,1			IF	ID($6)	EX	MEM	WB	
j 10				IF	ID	EX	MEM	WB

Programul rescris, fără hazarduri:

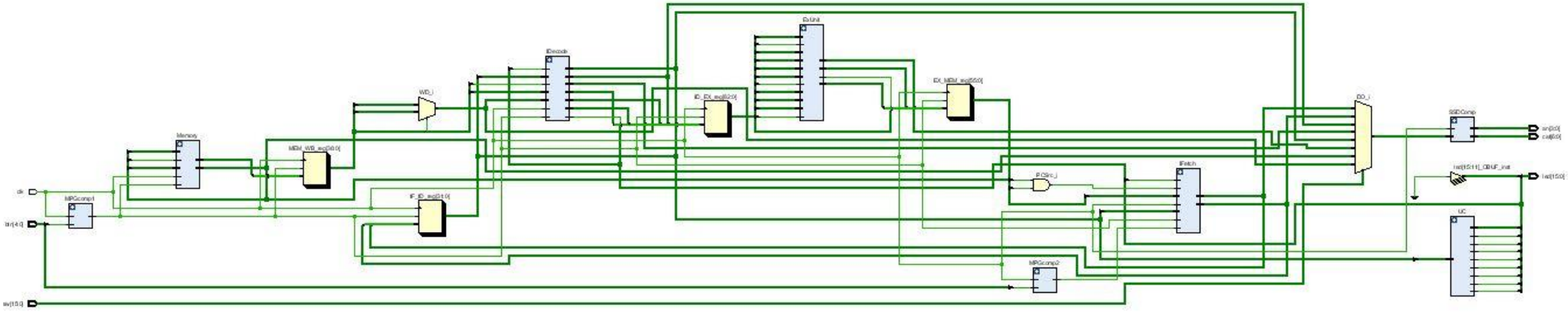
Adr.	Instrucțiune
0	addi $1,$0,2
1	addi $2,$0,1
2	addi $3,$0,3
3	addi $4,$0,5
4	sw $2,0($2)
5	sw $3,0($1)
6	lw $5,0($1)
7	lw $6,0($2)
8	NoOp
9	NoOp
10	add $7,$5,$6
11	addi $6,$0,1
12	add $5,$3,$1
13	NoOp
14	NoOp
15	add $7,$7,$5
16	addi $3,$5,0
17	addi $6,$6,1
18	NoOp
19	NoOp
20	beq $6,$4,4
21	NoOp
22	NoOp
23	NoOp
24	j 12
25	NoOp

5. Corectitudinea

Tot proiectul este descris în limbaj VHDL, în afară de fișierul de constrângeri.

Nu am întâmpinat probleme și erori, fișierul bit generându-se cu succes.

Pentru corectitudine am verificat schematicul proiectul (imaginea de jos), iar căile de date corespund.



6. Testarea

Pentru testare am folosit simulatorul Vivado, nu s-a testat pe placă.
Am comparat rezultatele simulării cu valorile rezultate la procesorul MIPS cu ciclu unic, iar aceste valori au fost egale.
Am aplicat Force Clock pentru btn[0] (butonul de enable) pentru a simularea apăsării unui buton cu o perioadă de 10 ms, și un Force Clock pentru clk cu o perioadă de 10 ns, cu Leading edge value: 0, Trailing edge value: 1 pentru cele două semnale.
Am aplicat Force Constant pentru btn[1] (butonul de reset), constanta fiind 0 logic, iar pentru switch-uri nu contează ce constantă se aplică deoarece în simulare există posibilitatea de a adăuga semnalele dorite de la diferite componente pentru a vizualiza valorile acestora.
Mai jos am atașat o imagine cu rezultatul simulării unde se poate observa rezultatul așteptat (36):

În imagine rezultatul hașurat reprezintă valoarea registrului intermediar MEM_WB care reprezintă valorile din etapa de scriere pentru instrucțiunea add $7,$7,$5, unde MEM_WB(36 – 34) = 7 e adresa de scriere și MEM_WB(33 – 18) = 36 care e valoarea care se scrie la adresa 7.