Raport

1. Instrucțiuni

* XOR -instrucțiune de tip R

SAU EXCLUSIV logic între două registre și memorează rezultatul în al treilea.

-ASSEMBLY: XOR $rd, $rs, $rt

-RTL Abstract: RF[rd] ← RF[rs] ^ RF[rt]

-Operație: $rd ← $rs ^ $rt, PC ← PC + 1

-Cod mașină: 000_sss_ttt_ddd_0_110

-Exemplu: XOR $3, $2, $1

000_010_001_011_0_110

* SRA -instrucțiune de tip R

Deplasare aritmetică la dreapta pentru un registru, rezultatul este memorat în altul. Se repetă valoarea bitului de semn.

-ASSEMBLY: SRA $rd, $rt, h

-RTL Abstract: RF[rd] ← RF[rt] >> sa

-Operație: $rd ← $rt >> h, PC ← PC + 1

-Cod mașină: 000_sss_ttt_ddd_h_111

-Exemplu: SRA $3, $1, 1

000_000_001_011_1_111

* ANDI -instrucțiune de tip I

AND logic între un registru și o valoare imediată și memorează rezultatul în al doilea registru.

-ASSEMBLY: ANDI $rt, $rs, imm

-RTL Abstract: RF[rt] ← RF[rs] & s_ext(imm)

-Operație: $rt ← $rs ^ imm, PC ← PC + 1

-Cod mașină: 101_sss_ttt_iiiiiii

-Exemplu: ANDI $4, $3, 7

101_011_100_0000111

* ORI -instrucțiune de tip I

OR logic între un registru și o valoare imediată și memorează rezultatul în al doilea registru.

-ASSEMBLY: ORI $rt, $rs, imm

-RTL Abstract: RF[rt] ← RF[rs] | s_ext(imm)

-Operație: $rt ← $rs | imm, PC ← PC + 1

-Cod mașină: 110_sss_ttt_iiiiiii

-Exemplu: ORI $5, $2, 8

110_010_101_0001000

2. Semnale control MIPS16

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

Instrucțiune	**Opcode** *Instr(15-13)*	**RegDst**	**ExtOp**	**ALUSrc**	**Branch**	**Br<?>**	**Jump**	**MemWrite**	**MemtoReg**	**RegWrite**	**ALUOp (2:0)**	**func** *Instr(2-0)*	**ALUCtrl (2:0)**	**JmpR**
ADD	000	1	Any	0	0		0	0	0	1	001 (R)	000	000 (+)	
SUB	000	1	Any	0	0		0	0	0	1	001 (R)	001	001 (-)	
SLL	000	1	Any	0	0		0	0	0	1	001 (R)	010	010 (<<l)	
SRL	000	1	Any	0	0		0	0	0	1	001 (R)	011	011 (>>l)	
AND	000	1	Any	0	0		0	0	0	1	001 (R)	100	100 (&)	
OR	000	1	Any	0	0		0	0	0	1	001 (R)	101	101 (|)	
XOR	000	1	Any	0	0		0	0	0	1	001 (R)	110	110 (^)	
SRA	000	1	Any	0	0		0	0	0	1	001 (R)	111	111 (>>a)	
ADDI	001	0	1	1	0		0	0	0	1	010 (+)	X	000 (+)	
LW	010	0	1	1	0		0	0	1	1	010 (+)	X	000 (+)	
SW	011	X	1	1	0		0	1	0	0	010 (+)	X	000 (+)	
BEQ	100	X	1	0	1		0	0	0	0	011 (-)	X	001 (-)	
ANDI	101	0	0	1	0		0	0	0	1	100 (&)	X	100 (&)	
ORI	110	0	0	1	0		0	0	0	1	101 (|)	X	101 (|)	
J	111	X	X	X	X		1	0	X	0	Any	X	Any	

3. Descrierea programului executat de procesor

Programul executat de procesor face suma primelor 6 numere impare.

Rezultatul va fi reținut în register file la adresa 7.

Inițial, în register file se vor memora valoarea 2 la adresa 1, valoare care se adună pentru a obține noul număr impar, valoarea 1 la adresa 2 care reprezintă prima cifră impară, valoarea 3 la adresa 3 care reprezintă prima cifră impară, valoarea 5 la adresa 4 care reprezintă numărul de adunări.

În memorie la adresa 1 (valoarea lui RF(2)) se stochează valoarea 1 (valoarea lui RF(2)), iar la adresa 2 (valoarea lui RF(1)) se stochează valoarea 3 (valoarea lui RF(3)), după care din memorie se încarcă valorile de la adresele 2 și 1 în register file la adresele 5, respective 6.

În register file la adresa 7 se încarcă suma valorilor de la adresele 5 și 6 (1 + 3) și se modifică valoarea din register file de la adresa 6 cu 1, în RF(6) fiind contorul adunărilor efectuate.

În buclă se va tot adăuga la valoarea din RF(5) valoarea din RF(1) care este 2 pentru a obține numărul impar care se va aduna la valoarea din RF(7), iar după această adunare valoarea din RF(6) se va incrementa. Cu beq se va verifica dacă valoarea din RF(6) e egală cu valoarea din RF(4), dacă sunt egale atunci se iese din buclă, altfel se va efectua în continuare bucla cu ajutorul instrucțiunii de jump.

4. Trasarea execuției programului

Este făcută pentru toate iterațiile și am făcut-o sub formă de tabel.
Unde este - într-o celulă, acela este un semnal irelevant pentru instrucțiunea respectivă.

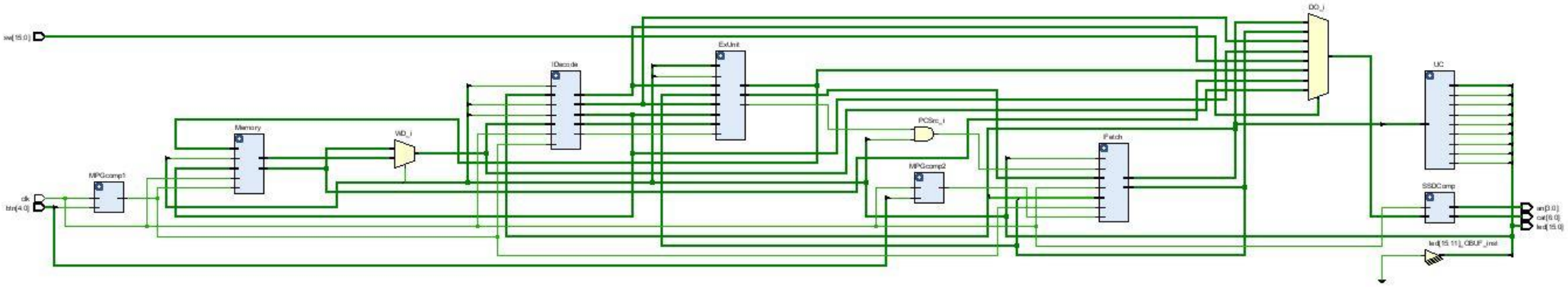
Trasarea execuției programului de test pentru MIPS16

Index Instr	SW(7:5)	"000"	"001"	"010"	"011"	"100"	"101"	"110"	"111"		
	Instr	Instr (*hexa*)	PC+1	RD1	RD2	Ext_Imm	ALURes	MemData	WD	BranchAddr	JumpAddr
0	addi $1, $0, 2	X"2082"	X"0001"	X"0000"	–	X"0002"	X"0002"	–	X"0002"	–	–
1	addi $2, $0, 1	X"2101"	X"0002"	X"0000"	–	X"0001"	X"0001"	–	X"0001"	–	–
2	addi $3, $0, 3	X"2183"	X"0003"	X"0000"	–	X"0003"	X"0003"	–	X"0003"	–	–
3	addi $4, $0, 5	X"2205"	X"0004"	X"0000"	–	X"0005"	X"0005"	–	X"0005"	–	–
4	sw $2, 0($2)	X"6900"	X"0005"	X"0001"	–	X"0000"	X"0001"	–	–	–	–
5	sw $3, 0($1)	X"6580"	X"0006"	X"0002"	–	X"0000"	X"0002"	–	–	–	–
6	lw $5, 0($1)	X"4680"	X"0007"	X"0002"	–	X"0000"	X"0002"	X"0003"	X"0003"	–	–
7	lw $6, 0($2)	X"4B00"	X"0008"	X"0001"	–	X"0000"	X"0001"	X"0001"	X"0001"	–	–
8	add $7, $5, $6	X"1770"	X"0009"	X"0003"	X"0001"	–	X"0004"	–	X"0004"	–	–
9	addi $6, $0, 1	X"2301"	X"000A"	X"0000"	–	X"0001"	X"0001"	–	X"0001"	–	–
10	add $5, $3, $1	X"0CD0"	X"000B"	X"0003"	X"0002"	–	X"0005"	–	X"0005"	–	–
11	add $7, $7, $5	X"1EF0"	X"000C"	X"0004"	X"0005"	–	X"0009"	–	X"0009"	–	–
12	addi $3, $5, 0	X"3580"	X"000D"	X"0005"	–	X"0000"	X"0005"	–	X"0005"	–	–
13	addi $6, $6, 1	X"3B01"	X"000E"	X"0001"	–	X"0001"	X"0002"	–	X"0002"	–	–
14	beq $6, $4, 1	X"9A01"	X"000F"	X"0002"	X"0005"	X"0001"	–	–	–	X"0010"	–
15	j 10	X"E00A"	X"0010"	–	–	X"000A"	–	–	–	–	X"000A"
10	add $5, $3, $1	X"0CD0"	X"000B"	X"0005"	X"0002"	–	X"0007"	–	X"0007"	–	–
11	add $7, $7, $5	X"1EF0"	X"000C"	X"0009"	X"0007"	–	X"0010"	–	X"0010"	–	–

12	addi $3, $5, 0	X"3580"	X"000D"	X"0007"	–	X"0000"	X"0007"	–	X"0007"	–	–
13	addi $6, $6, 1	X"3B01"	X"000E"	X"0002"	–	X"0001"	X"0003"	–	X"0003"	–	–
14	beq $6, $4, 1	X"9401"	X"000F"	X"0003"	X"0005"	X"0001"	–	–	–	X"0010"	–
15	j 10	X"E00A"	X"0010"	–	–	X"000A"	–	–	–	–	X"000A"
10	add $5, $3, $1	X"0CD0"	X"000B"	X"0007"	X"0002"	–	X"0009"	–	X"0009"	–	–
11	add $7, $7, $5	X"1EF0"	X"000C"	X"0010"	X"0009"	–	X"0019"	–	X"0019"	–	–
12	addi $3, $5, 0	X"3580"	X"000D"	X"0009"	–	X"0000"	X"0009"	–	X"0009"	–	–
13	addi $6, $6, 1	X"3B01"	X"000E"	X"0003"	–	X"0001"	X"0004"	–	X"0004"	–	–
14	beq $6, $4, 1	X"9401"	X"000F"	X"0004"	X"0005"	X"0001"	–	–	–	X"0010"	–
15	j 10	X"E00A"	X"0010"	–	–	X"000A"	–	–	–	–	X"000A"
10	add $5, $3, $1	X"0CD0"	X"000B"	X"0009"	X"0002"	–	X"000B"	–	X"000B"	–	–
11	add $7, $7, $5	X"1EF0"	X"000C"	X"0019"	X"000B"	–	X"0024"	–	X"0024"	–	–
12	addi $3, $5, 0	X"3580"	X"000D"	X"000B"	–	X"0000"	X"000B"	–	X"000B"	–	–
13	addi $6, $6, 1	X"3B01"	X"000E"	X"0004"	–	X"0001"	X"0005"	–	X"0005"	–	–
14	beq $6, $4, 1	X"9401"	X"000F"	X"0005"	X"0005"	X"0001"	–	–	–	X"0010"	–
15		X"0000"	X"0011"								

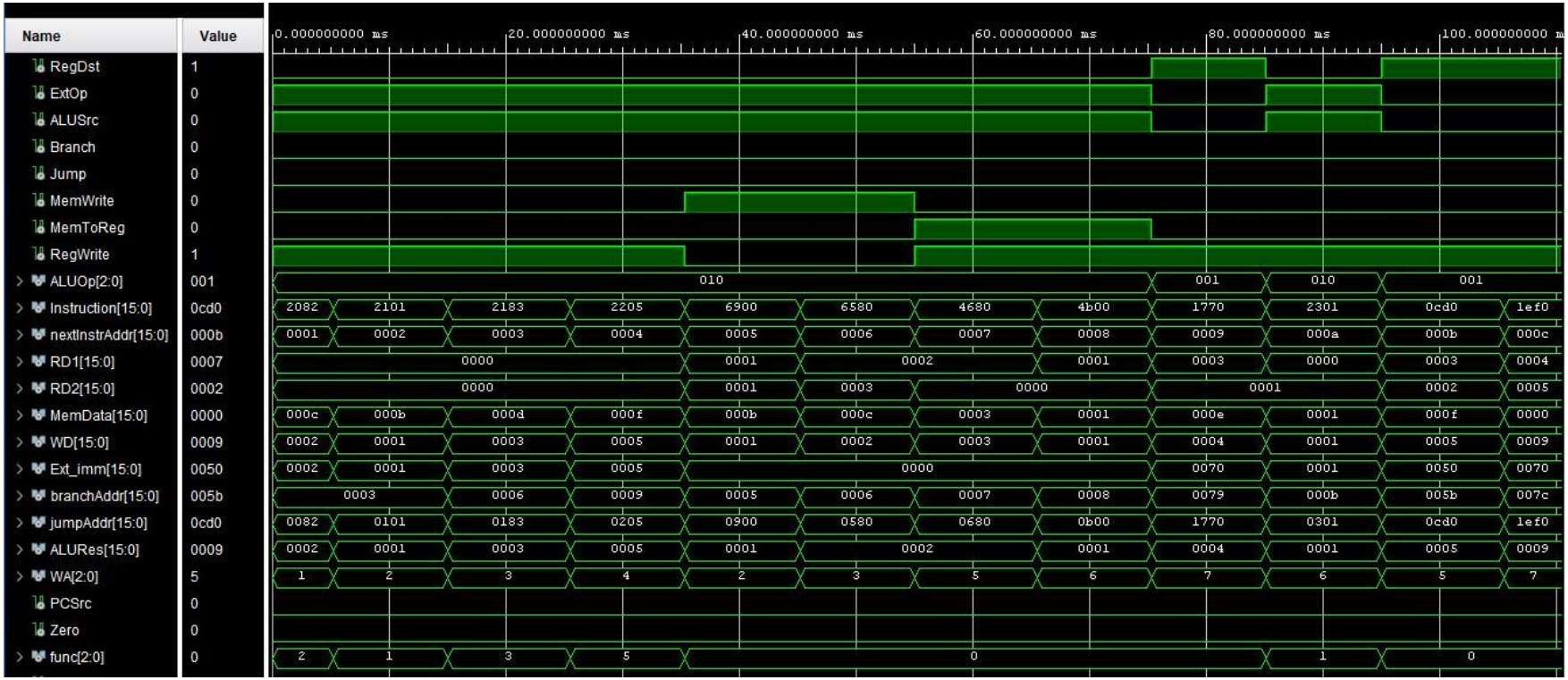
5. Corectitudinea

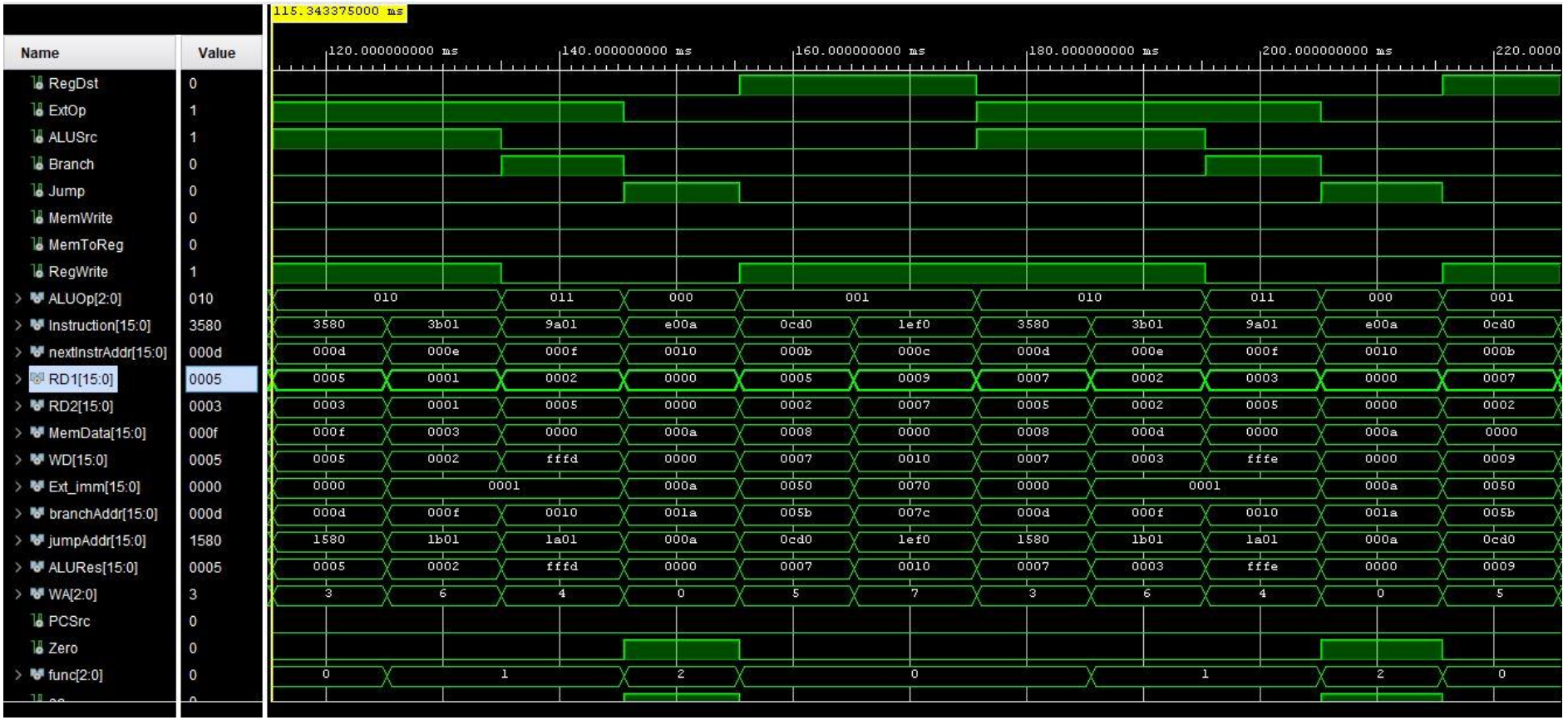
Tot proiectul este descris în limbaj VHDL, în afară de fișierul de constrângeri.
Nu am întâmpinat probleme și erori, fișierul bit generându-se cu succes.
Pentru corectitudine am verificat schematicul proiectul (imaginea de jos), iar căile de date corespund.

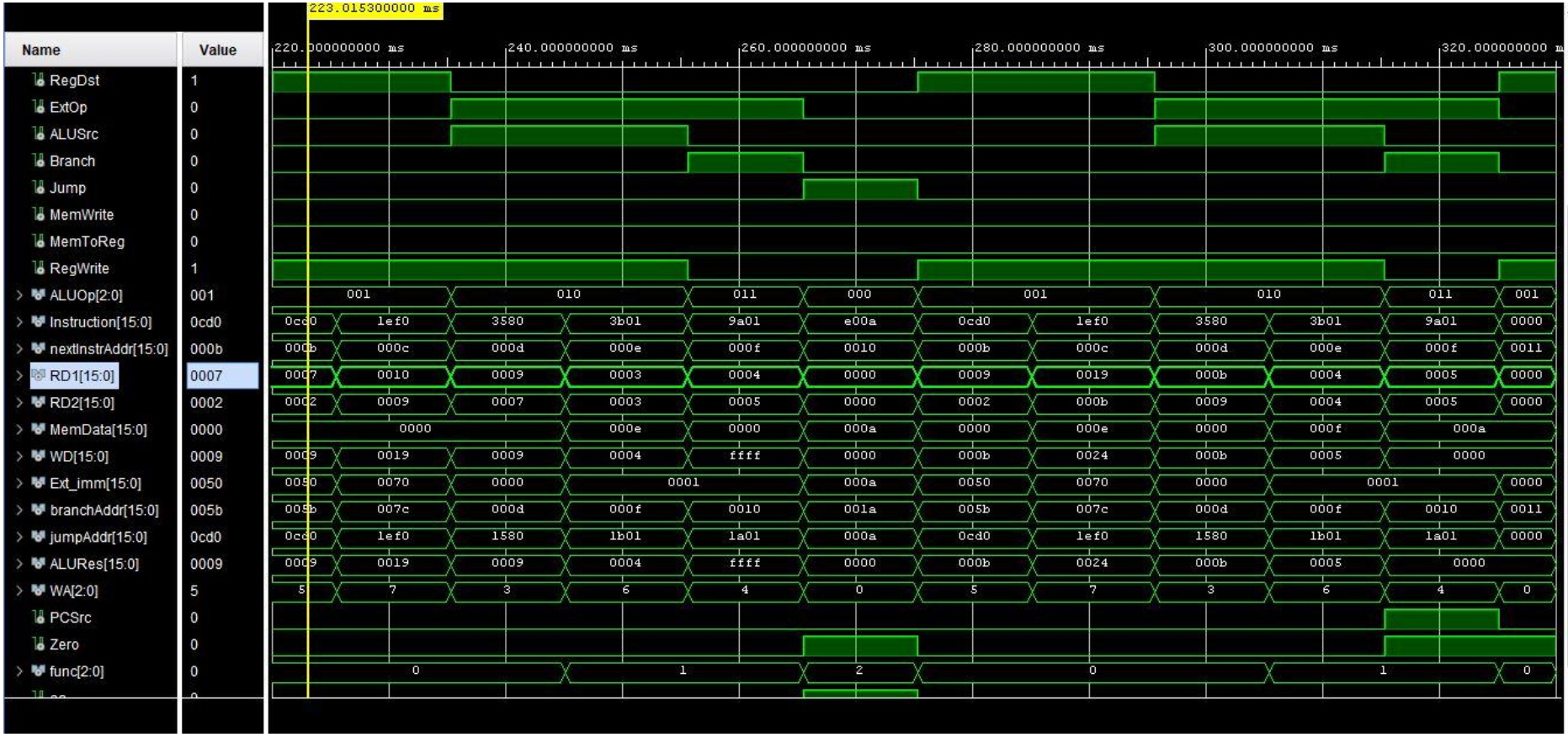


6. Testarea

Pentru testare am folosit simulatorul Vivado.
Am comparat rezultatele simulării cu valorile semnalelor de la trasarea execuției programului, 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 imagini cu rezultatele simulării pentru tot programul.




115.343375000 ms
120.000000000 ms
140.000000000 ms
160.000000000 ms
180.000000000 ms
200.000000000 ms
220.0000
Name
Value
RegDst 0
ExtOp 1
ALUSrc 1
Branch 0
Jump 0
MemWrite 0
MemToReg 0
RegWrite 1
ALUOp[2:0] 010 010 011 000 001 010 011 000 001
Instruction[15:0] 3580 3580 3b01 9a01 e00a 0cd0 1ef0 3580 3b01 9a01 e00a 0cd0
nextInstrAddr[15:0] 000d 000d 000e 000f 0010 000b 000c 000d 000e 000f 0010 000b
RD1[15:0] 0005 0005 0001 0002 0000 0005 0009 0007 0002 0003 0000 0007
RD2[15:0] 0003 0003 0001 0005 0000 0002 0007 0005 0002 0005 0000 0002
MemData[15:0] 000f 000f 0003 0000 000a 0008 0000 0008 000d 0000 000a 0000
WD[15:0] 0005 0005 0002 fffd 0000 0007 0010 0007 0003 fffe 0000 0009
Ext_imm[15:0] 0000 0000 0001 000a 0050 0070 0000 0001 000a 0050
branchAddr[15:0] 000d 000d 000f 0010 001a 005b 007c 000d 000f 0010 001a 005b
jumpAddr[15:0] 1580 1580 1b01 1a01 000a 0cd0 1ef0 1580 1b01 1a01 000a 0cd0
ALURes[15:0] 0005 0005 0002 fffd 0000 0007 0010 0007 0003 fffe 0000 0009
WA[2:0] 3 3 6 4 0 5 7 3 6 4 0 5
PCSrc 0
Zero 0
func[2:0] 0 0 1 2 0 1 2 0


223.015300000 ms
220.000000000 ms
240.000000000 ms
260.000000000 ms
280.000000000 ms
300.000000000 ms
320.000000000 m
Name
Value
RegDst 1
ExtOp 0
ALUSrc 0
Branch 0
Jump 0
MemWrite 0
MemToReg 0
RegWrite 1
ALUOp[2:0] 001
Instruction[15:0] 0cd0
nextInstrAddr[15:0] 000b
RD1[15:0] 0007
RD2[15:0] 0002
MemData[15:0] 0000
WD[15:0] 0009
Ext_imm[15:0] 0050
branchAddr[15:0] 005b
jumpAddr[15:0] 0cd0
ALURes[15:0] 0009
WA[2:0] 5
PCSrc 0
Zero 0
func[2:0] 0