Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών
Σχεδίαση Ψηφιακών Συστημάτων (ICE – 4005) Project_MIPS _2020
Γκούσαρης Κωνσταντίνος – 711171073 – cs171073@uniwa.gr

6: MIPS R TYPE

Τρόπος Σχεδίασης

Το κύκλωμα έχει αρχικά υλοποιηθεί και κατασκευαστεί με την βοήθεια και τις οδηγίες που το βιβλίο θεωρίας **"Τεχνολογία και Σχεδίαση Ψηφιακών Συστημάτων", Ιωάννης Βογιατζής, 3η Έκδοση** και τις διαφάνειες του μαθήματος.

Αρχικά προστέθηκαν κάποια εξαρτήματα-κυκλώματα τα οποία υλοποιήθηκαν στα παραπάνω βήματα αλλά διαπιστώθηκε ότι δεν ταίριαζαν με την συνολική υλοποίηση και έπρεπε να παραμετροποιηθούν κατάλληλα ώστε να μπορέσει να γίνει λειτουργικό το κύκλωμα. Τα εξαρτήματα τα οποία κατασκευάστηκαν εκ νέου ή αλλάχτηκαν σε αυτά κάποιες τιμές είναι το Instruction Memory, ο Program Counter, το Register File και ο Full Adder. Όπως προαναφέρθηκε το βιβλίο της θεωρίας αποτέλεσε Manual στο πως πρέπει να γίνουν οι αλλαγές και οι συνδέσεις των εξαρτημάτων. Η σχεδίαση του MIPS έγινε σταδιακά, αρχικά υλοποιήθηκε το πρώτο μέρος του το οποίο αποτελείται από την IM(Instruction Memory), τον PC(program counter) και τον FA(Full Adder). Ελέγχθηκε η έξοδος της IM ήταν ορθή και η οποία αποτελεί είσοδο για τον Register File και την Alu Control. Στην συνεχεία συνδέθηκαν και τα υπόλοιπα components μεταξύ τους συμφωνά με το παρακάτω σχήμα σχεδίασης. Επίσης το τελικό κύκλωμα του περιέχει σήματα και έξοδοι εξαρτημάτων(components) τα οποία δεν χρησιμοποιούνται στην υλοποίησή των R TYPE εντολών αλλά υλοποιήθηκαν σε περίπτωση κατά την οποία γίνει επέκταση ή ολοκλήρωση του κυκλώματος.

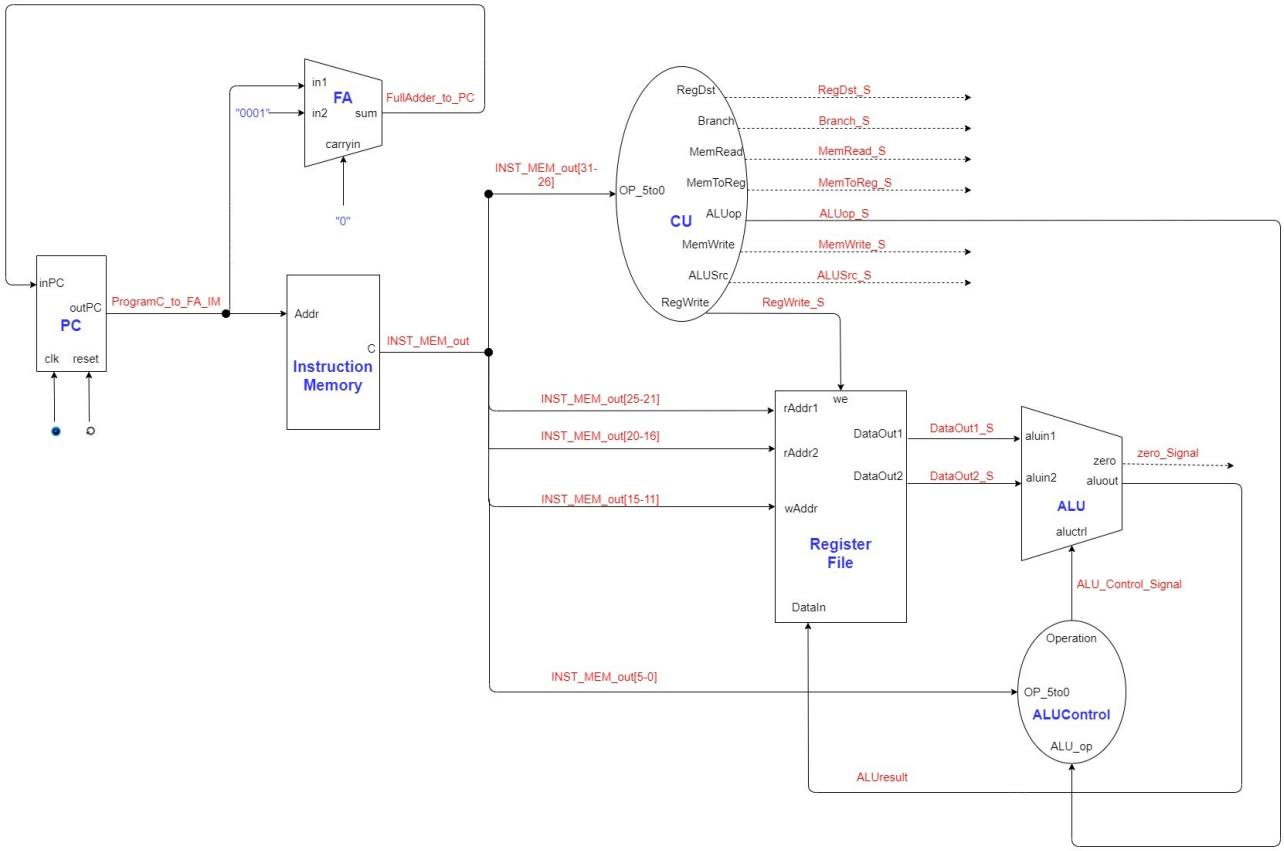
Τα περιεχόμενα του Register File και του Instruction Memory κατασκευάστηκαν manually και τοποθετήθηκαν τιμές σε αυτά συμφωνά με τα δεδομένα τα οποία η άσκηση ζητούσε ως προς επίδειξη καλής λειτουργίας των R TYPE εντολών ενός επεξεργαστή MIPS.

Τέλος το testbench το οποίο υλοποιήθηκε, σαν έξοδο του MIPS(outMIPS) περιέχει το αποτέλεσμά της ALU(aluout), δηλαδή έχει γίνει σύνδεση της εξόδου του κυκλώματος την έξοδό της ALU όπου ταυτόχρονα συνδέεται και με την είσοδο εγγραφής(DataIn) του Register File.

Σχήμα Σχεδίασης

R TYPE υλοποίηση του επεξεργαστή MIPS

*με μπλε χρωμα ειναι το ονομα του καθε εξαρτηματος
*με κοκκινο χρωμα ειναι τα εξωτερίκα σηματα
*με μαυρο οι εισοδοι και οι εξοδοι του κάθε εξαρτηματος.



6.extras_FullAdder.vhd

```
--MIPS Part_6
--FULL ADDER
--27/05/2020, Konstantinos Gkousaris, 711171073, UniWA

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY Fulladder32bit IS PORT (
      in1, in2  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      carryin   : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      sum     : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      carryout  : OUT STD_LOGIC);
END  Fulladder32bit;

ARCHITECTURE FA32bit OF Fulladder32bit IS

      SIGNAL tmp : STD_LOGIC_VECTOR(4 DOWNTO 0); --SUM WITH CARRY ON (32)
BEGIN
      tmp <= STD_LOGIC_VECTOR(to_signed(to_integer(signed(in1)) +
            to_integer(signed(in2)) + to_integer(signed(carryin)),5));
      carryout <= tmp(4);
      sum      <= tmp(3 DOWNTO 0);
END FA32bit;
```

6.extras_ProgramCounter.vhd

```
--MIPS Part_6
--PROGRAM COUNTER,
--27/05/2020, Konstantinos Gkousaris, 711171073, UniWA

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY programCounter IS PORT (
      inPC  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      outPC : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      clk     : IN STD_LOGIC;
      reset   : IN STD_LOGIC);
END programCounter;

ARCHITECTURE programCounter_1 OF programCounter IS
BEGIN
      reg : PROCESS(clk)
      BEGIN
            IF( reset = '1' ) THEN
                  outPC <= STD_LOGIC_VECTOR(to_signed(-1,4));
            END IF;
            IF RISING_EDGE(clk) THEN
                  outPC <= inPC;
            END IF;
      END PROCESS;
END programCounter_1;
```

6.extras_InstructionMemory.vhd

```
--MIPS Part_6
--MemoryInstruction_MIPS
--Memory Set specialy for the implementation of FINAL_PROJECT
--27/05/2020, Konstantinos Gkousaris, 711171073, UniWA

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;


ENTITY instrMemoryMIPS IS PORT (
            Addr : IN STD_LOGIC_VECTOR(3 downto 0);
            C : out STD_LOGIC_VECTOR(31 downto 0));
END instrMemoryMIPS;

ARCHITECTURE behavioral OF instrMemoryMIPS IS

TYPE rom16x32 IS ARRAY (0 TO 15) OF STD_LOGIC_VECTOR(31 downto 0);

        --give default
      SIGNAL instrmem : rom16x32 := (                 --Instruction Set
            "00000000010001100010000000100000",       --add $4 $2 $6
            "00000000010001100010100000100010",       --sub $5 $2 $6
            "00000000000000000000000000000000",
            "00000000100000010000000000000000",
            "11111111111111111111111111111111",
            "00000000000000000000000000000000",
            "11111111111111111111111111111111",
            "00000000100000010000000000000000",
            "11111111111111111111111111111111",
            "00000000101001100010000000100000",
            "11111111111111111111111111111111",
            "11111111111111111111111111111111",
            "11111111111111111111111111111111",
            "11111111111111111111111111111111",
            "11111111111111111111111111111111",
            "11111111111111111111111111111111");
BEGIN
      C <= instrmem(to_integer(unsigned(Addr)));

END;
```

6.extras_ProgramCounter.vhd

```
--MIPS Part_6
--RegisterFileFull_for_MIPS
--29/05/2020, Konstantinos Gkousaris, 711171073, UniWA

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;

ENTITY registerfile_full_for_MIPS IS
      GENERIC (   dw : natural := 32;
                  size : natural := 32;
                  adrw : natural := 5);
```

```
PORT  (     Datain : IN STD_LOGIC_VECTOR(dw-1 downto 0);
                        rAddr1: IN STD_LOGIC_VECTOR(adrw-1 downto 0);
                        rAddr2: IN STD_LOGIC_VECTOR(adrw-1 downto 0);
                        wAddr : IN STD_LOGIC_VECTOR(adrw-1 downto 0);
                        we : IN STD_LOGIC;
                        clk :  IN STD_LOGIC;
                        reset : IN STD_LOGIC;
                        Dataout1 : OUT STD_LOGIC_VECTOR(dw-1 downto 0);
                        Dataout2 : OUT STD_LOGIC_VECTOR(dw-1 downto 0));
end registerfile_full_for_MIPS;

ARCHITECTURE behavioral OF registerfile_full_for_MIPS IS

TYPE regArray IS ARRAY(0 to size-1) OF std_logic_vector(dw-1 DOWNTO 0);

--maping of register file, for our demonstration we will use the registers
--$2, $6 which contains values 10 and 7 and $5, $4 which store the results
--there are plenty of values in other registers for testing purposes and also
--the registers have not the same names and same causes with a real MIPS
--implemantation because this implemantion is about only for R TYPE
--instructions.
signal regfileb : regArray := (          --register name (it's not full_MIPS)
            x"00000000",                        --$0
            x"00000000",                        --$1
            x"0000000a",--contains value 10     --$2
            x"00000000",                        --$3
            x"00000000",                        --$4
            x"00000000",                        --$5
            x"00000007",--contains value 7      --$6
            x"00000000",                        --$7
            x"00000000",                        --$8
            x"00000000",                        --$9
            x"0000000a",--contains value 10     --$10
            x"00000000",                        --$11
            x"00000007",--contains value 7      --$12
            x"00000005",--contains value 5      --$13
            x"00000000",                        --$14
            x"00000000",                        --$15
            x"00000000",                        --$16
            x"00000000",                        --$17
            x"00000000",                        --$18
            x"00000000",                        --$19
            x"00000000",                        --$20
            x"00000000",                        --$21
            x"00000000",                        --$22
            x"00000000",                        --$23
            x"00000000",                        --$24
            x"00000000",                        --$25
            x"00000000",                        --$26
            x"00000000",                        --$27
            x"10008000",                        --$28
            x"7FFFF1EC",                        --$29
            x"eeeeeeee",                        --$30
            x"ffffffff"                         --$31
      );
```

```
BEGIN
	PROCESS(clk)
	BEGIN
	IF reset= '1' THEN --reset entire circuit
		Dataout1 <= x"FFFFFFFF" ;
		Dataout2 <= x"FFFFFFFF" ;
	ELSIF (clk'event AND clk='0') then
		IF we='1' then

		Dataout1 <= regfileb(to_integer(unsigned(rAddr1)));
		Dataout2 <= regfileb(to_integer(unsigned(rAddr2)));

		END IF;
	END IF;
		Dataout1 <= regfileb(to_integer(unsigned(rAddr1)));
		Dataout2 <= regfileb(to_integer(unsigned(rAddr2)));
	END PROCESS;
END behavioral;
```

6.1_MIPS.vhd

```
--MIPS Part_6
--MIPS, main implementation
--14/05/2020, Konstantinos Gkousaris, 711171073, UniWA
--Architecture based, code base on implementation of book
--"Τεχνολογία και Σχεδίαση Ψηφιακών Συστημάτων",Third Edition, Ioannis Vogiatzis

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY mips IS PORT (
	clk   : IN STD_LOGIC;
	Reset       : IN STD_LOGIC;
	outMIPS : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
END mips;

ARCHITECTURE mips_ar_1 OF mips IS
--for implementation to execute the R type instruction we need the components:
--ALU 32bit
--Register File
--Instruction Memory
--Control Unit
--ALU Control Unit
--Program Counter(PC)

--(1)ALU 32bit
COMPONENT alu32 PORT (
	aluin1      : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
	aluin2      : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
	aluctrl     : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
	aluout      : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
	zero        : OUT STD_LOGIC);
END COMPONENT;
```

```
--(2)Register File
COMPONENT registerfile_full_for_MIPS PORT(
      Datain       : IN STD_LOGIC_VECTOR(31 downto 0);
      rAddr1       : IN STD_LOGIC_VECTOR(4 downto 0);
      rAddr2       : IN STD_LOGIC_VECTOR(4 downto 0);
      wAddr        : IN STD_LOGIC_VECTOR(4 downto 0);
      we           : IN STD_LOGIC;
      clk          : IN STD_LOGIC;
      reset        : IN STD_LOGIC;
      Dataout1     : OUT STD_LOGIC_VECTOR(31 downto 0);
      Dataout2     : OUT STD_LOGIC_VECTOR(31 downto 0));
END COMPONENT;

--(3)Instruction Memory
COMPONENT instrMemoryMIPS PORT (
      Addr  : IN STD_LOGIC_VECTOR(3 downto 0);
      C     : OUT STD_LOGIC_VECTOR(31 downto 0));
END COMPONENT;

--(4)Control Unit
COMPONENT Control IS PORT (
      OP_5to0 : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
      RegDst, RegWrite, ALUSrc, Branch : OUT STD_LOGIC;
      MemRead, MemWrite, MemtoReg : OUT STD_LOGIC;
      ALU_op: OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END COMPONENT;

--(5)ALU Control Unit
COMPONENT Alu_Control PORT(
      Funct       : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
      ALU_op      : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
      Operation   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END COMPONENT;

--(6)Full Adder 4bit
COMPONENT Fulladder32bit PORT (
      in1, in2    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      carryin     : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      sum         : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      carryout    : OUT STD_LOGIC);
END  COMPONENT;

--(7)Program Counter 4bit
COMPONENT programCounter PORT (
      inPC        : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      outPC       : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      clk         : IN STD_LOGIC;
      reset       : IN STD_LOGIC);
END COMPONENT;

--signal between blocks
--first step design, PC,FA,IM with out to RegFile
signal ProgramC_to_FA_IM  : std_logic_vector(3 DOWNTO 0):=(others => '0');
--signal from the output of PC-->FA/IM
signal FullAdder_to_PC    : std_logic_vector(3 DOWNTO 0):= (others => '0');
```

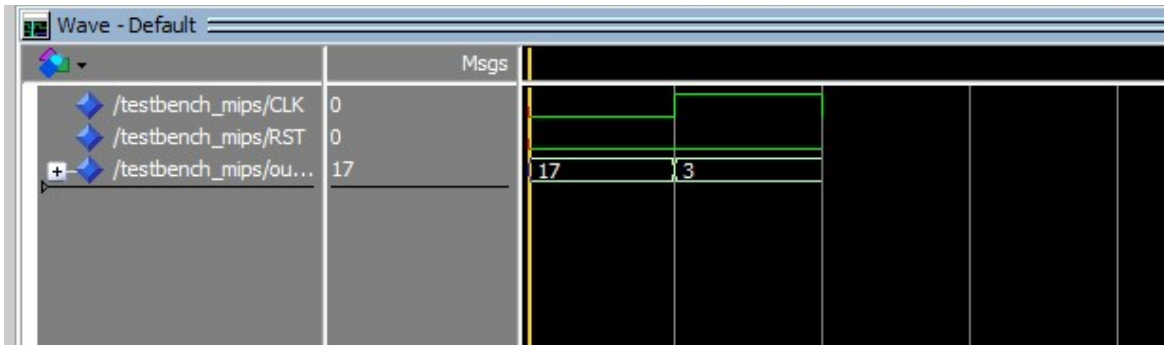
```
--signal the result of Fulladder to PC
signal INST_MEM_out       : std_logic_vector(31 DOWNTO 0):= (others => '0');
--SIGNAL which is result from Instruction Memory

--second step design control Unit
signal RegWrite_S       : std_logic:='1';
signal ALUSrc_S         : std_logic:='0';
signal MemWrite_S       : std_logic:='0';
signal MemRead_S        : std_logic:='0';
signal RegDist_S        : std_logic:='0';
signal MemToReg_S       : std_logic:='0';
signal Branch_S         : std_logic:='0';
signal ALU_op_S         : std_logic_vector(1 DOWNTO 0):=(others => '0');
--out to ALU_control

--third step design RegisterFile
signal DataOut1_S : std_logic_vector(31 DOWNTO 0):=(others => '0');
--SIGNAL from register to ALUin1
signal DataOut2_S : std_logic_vector(31 DOWNTO 0):=(others => '0');
--SIGNAL from register to ALUin2


--fourth step design ALU
signal ALUresult              : std_logic_vector(31 DOWNTO 0):=(others => '0');
--SIGNAL from ALU to REGFILE
signal zero_Signal            : std_logic:='0';
--ZERO
signal ALU_Control_Signal     : std_logic_vector(3 DOWNTO 0):=(others => '0');
--SIGNAL from Alu Control to ALU32
signal ALUoperation           : std_logic_vector(1 DOWNTO 0):=(others => '0');
--SIGNAL For ALU Operations

BEGIN

FULLADDER_CON         : Fulladder32bit PORT MAP(
                  in1         => ProgramC_to_FA_IM,
                  in2         => "0001",
                  carryin     => "0",
                  sum         => FullAdder_to_PC
                  );
INSTMEM_CON           : instrMemoryMIPS PORT MAP(
                  Addr        =>  ProgramC_to_FA_IM,
                  C           =>  INST_MEM_out --outMIPS
                  );
PROGRAMCOUNTER_CON  : programCounter PORT MAP(
                  inPC        => FullAdder_to_PC,
                  outPC       => ProgramC_to_FA_IM,
                  clk         => clk,
                  reset       => reset
                  );
REGISTER_FILE_CON : registerfile_full_for_MIPS PORT MAP (
                  Datain      => ALUresult,
                  rAddr1      => INST_MEM_out(25 DOWNTO 21),
                  rAddr2      => INST_MEM_out(20 DOWNTO 16),
                  wAddr     => INST_MEM_out(15 DOWNTO 11),
```

```
                we        => RegWrite_S,
                clk       => clk,
                Reset     => Reset,
                Dataout1  => DataOut1_S,
                Dataout2  => DataOut2_S
                );
CONTROLUNIT_CON   : Control PORT MAP(
                OP_5to0     => INST_MEM_out(31 DOWNTO 26),      --it is used
                RegDst      => RegDist_S,
                RegWrite    => RegWrite_S,                      --it is used
                ALUSrc      => ALUSrc_S,
                Branch      => Branch_S,
                MemRead     => MemRead_S,
                MemWrite    => MemWrite_S,
                MemtoReg    => MemtoReg_S,
                ALU_op      => ALU_op_S                         --it is used
                );

ALU_CONTROL_CON   : Alu_Control PORT MAP(
                Funct       =>  INST_MEM_out(5 DOWNTO 0),
                ALU_op      =>  ALU_op_S,
                Operation   =>  ALU_Control_Signal
                );

ALU_32_CONNECCT   : alu32 PORT MAP (
                aluin1      => DataOut1_S,
                aluin2      => DataOut2_S,
                aluctrl     => ALU_Control_Signal,
                aluout      => ALUresult,
                zero        => zero_Signal
                );

outMIPS <= ALUresult;

END;
```

6.2_MIPS_testbench.vhd

```
--MIPS Part_6
--MIPS, test bench
--02/06/2020, Konstantinos Gkousaris, 711171073, UniWA
--Architecture based, code base on implementation of book
--"Τεχνολογία και Σχεδίαση Ψηφιακών Συστημάτων",Third Edition, Ioannis Vogiatzis


LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;


ENTITY testBench_MIPS IS
END testBench_MIPS;

ARCHITECTURE behavioral OF testBench_MIPS IS
```

```
	COMPONENT mips PORT (
		CLK	: IN STD_LOGIC;
		Reset		: IN STD_LOGIC;
		outMIPS : OUT STD_LOGIC_VECTOR(31 DOWNTO 0));
	END COMPONENT;

	SIGNAL CLK	: STD_LOGIC;
	SIGNAL RST	: STD_LOGIC;
	SIGNAL outMIPS	: std_logic_vector(31 DOWNTO 0);

	constant clk_period : time := 100 ns;

BEGIN
	UUT : mips PORT MAP (CLK, RST, outMIPS);
	--process for clock
	clk_process :PROCESS
	BEGIN
		clk<= '0'; wait for clk_period/2;
		clk <= '1'; wait for clk_period/2;
	END PROCESS;

	simulation_mips : PROCESS
	BEGIN
		--Rst <= '1'; wait for 10 ns;
		Rst <= '0'; wait for 400 ns;
		Rst <= '1'; wait for 50 ns;
	END PROCESS;

END;
```



Πραγματοποιήθηκε δοκιμή ενός κύκλου βάζοντας στους καταχωρητές $2 και $6 τις τιμές 10 και 7 αντίστοιχα όπως αναγράφεται και στην υλοποίηση των Instruction Memory και Register File και δοκιμαστικά πραγματοποιείται η πράξη της πρόσθεσης(add) και της αφαίρεσης(sub) η οποία επιβεβαιώνει και την ορθή λειτουργία του κυκλώματος του επεξεργαστή MIPS και συγκεκριμένα την εκτέλεση των εντολών τύπου R.