ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΟΙΚΙΝΩΝΙΩΝ

Αρχιτεκτονική Υπολογιστών Εργαστήριο

Τελική Εργασία Εξαμήνου

Στοιχεία Φοιτητή

Ονοματεπώνυμο: Κατόπης Δημήτριος

ΑΜ:2124

Email:int02124@uoi.gr

1η Άσκηση – Wiring Arduino

Σε αυτό το κύκλωμα χρησιμοποιούμε ένα Arduino το οποίο σε συνεργασία με έναν αισθητήρα Ultrasonic Sensor HC-Sr04 μετράει την απόσταση αντικείμενων . Σε συνδυασμό με 8 led , παίρνουμε πληροφορίες σχετικά με την απόσταση η οποία μετρείται σε εκατοστά και ανάλογα με την απόσταση αναβοσβήνουν τα led καθώς όσο μικραίνει η απόσταση αλλάζει ο ρυθμός με τον οποίο αναβοσβήνουν και ο προειδοποιητικός ήχος γίνεται όλο και πιο έντονος (με την βοήθεια ενός αισθητήρα piezo) .

Απόσταση	LED
1-5 cm	LED 1
6-7 cm	LED 2
7-8 cm	LED 3
8-10 cm	LED 4
10-11 cm	LED 5
11-15 cm	LED 6
15-20 cm	LED 7
>20 cm	LED 8

Χρησιμοποιούμε : 1. Arduino Uno

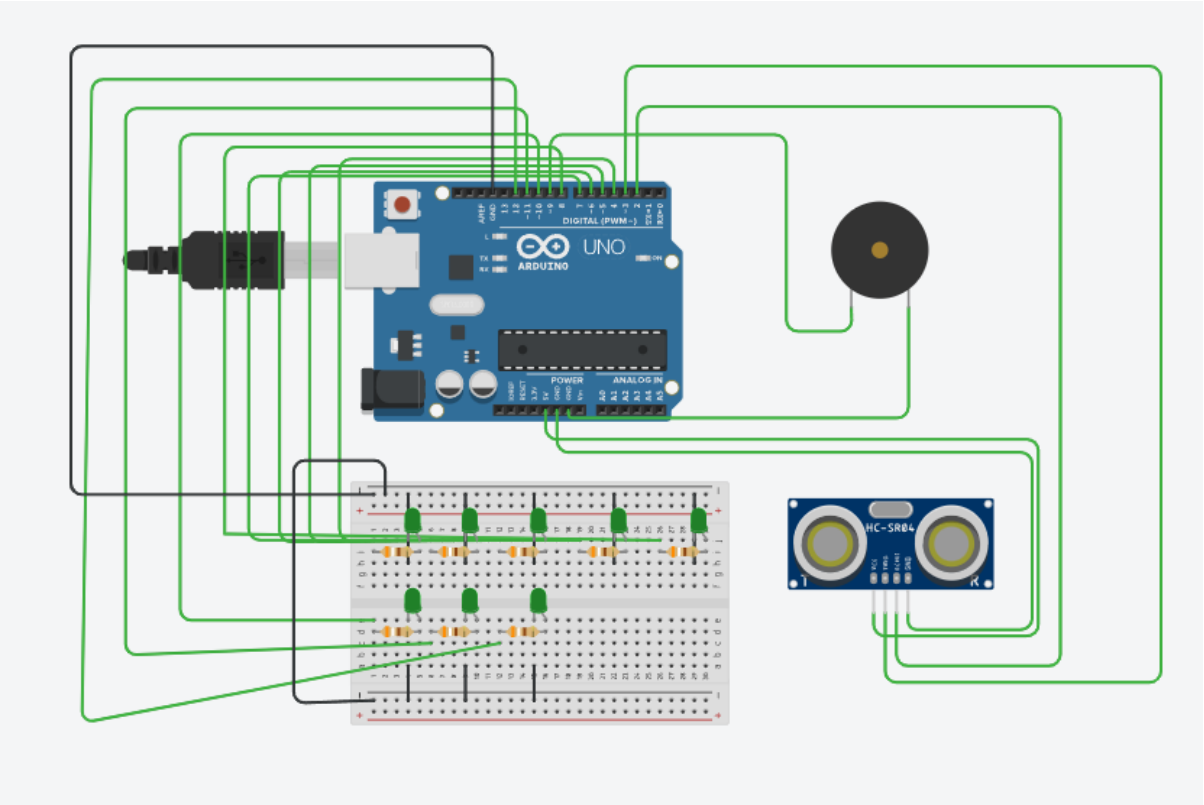
2. Breadboard

3.8 led

4. R=390Ω για κάθε λεντ

5. Ultrasonic sensor hc-sr04

6. 1 piezo



```
/*
This code was designed by Katopis Dimitrios.
It is intended for academic use at the Department of Informatics and Telecommunication
of University of Ioannina
*/
/*
This code was designed and compiled at Arduino ver. 1.8.13
Αυος ο κωδικας μετραει την αποσταση αντικειμων χρησιμοποιωντας εναν αισθητηρα Ulstrasonic
*/



#define echoPin 2 // συνδέστε την ακίδα D2 Arduino στην ακίδα Echo του HC-SR04
#define trigPin 3 // συνδέστε την ακίδα D3 Arduino στην ακίδα Trig του HC-SR04

long duration; // μεταβλητή για τη διάρκεια της διαδρομής των ηχητικών κυμάτων
double distance; // μεταβλητή για τη μέτρηση της απόστασης
const int buzzer = 9; //buzzer στο pin 9
int led1=12;
int led2=11;
int led3=10;
int led4=8;
int led5=7;  //αρχικοποιηση λεντ αποστασεων
int led6=6;
int led7=5;
int led8=4;

void setup() {
  pinMode(trigPin, OUTPUT); // Ορίζει το trigPin ως OUTPUT
  pinMode(echoPin, INPUT); // Ορίζει το echoPin ως INPUT
  Serial.begin(9600);     // Η σειριακή επικοινωνία ξεκινά με ταχύτητα baudrate 9600
  Serial.println("Ultrasonic Sensor HC-SR04");
  pinMode(buzzer, OUTPUT); // buzzer - pin 9 ως output

  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
  pinMode(led4,OUTPUT);
  pinMode(led5,OUTPUT);
  pinMode(led6,OUTPUT);
  pinMode(led7,OUTPUT);
  pinMode(led8,OUTPUT);

}
```

```
void loop() {
  // Αρχικοποίηση του trigPin ως LOW
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);

  // Θέτουμε το trigPin HIGH (ACTIVE) για 10 μικροδευτερόλεπτα
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Διαβάζει το echoPin, επιστρέφει το χρόνο διαδρομής του ηχητικού κύματος σε μικροδευτερόλεπτα
  duration = pulseIn(echoPin, HIGH);

  // Υπολογισμός της απόστασης
  distance = duration * 0.034 / 2; // Ταχύτητα ηχητικού κύματος διαιρούμενη με 2 (πηγαινει και πίσω

  // Εμφανίζει την απόσταση στη σειριακή οθόνη
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");


  if (distance>=1.0 && distance<=5.0 )
  {
    digitalWrite(led1,HIGH);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
    digitalWrite(led4,LOW);
    digitalWrite(led5,LOW);
    digitalWrite(led6,LOW);
    digitalWrite(led7,LOW);
    digitalWrite(led8,LOW);
    tone(buzzer, 1000);  // Αποστολή 1KHz ηχητικού σήματος...
    delay(100);         // για 1sec
    noTone(buzzer);     // Σταματάμε τον ήχο
    delay(100);
  }
  else if (distance>5.0 && distance<=7.0)
  {
    digitalWrite(led1,LOW);
    digitalWrite(led2,HIGH);
    digitalWrite(led3,LOW);
```

```
  else if (distance>7.0 && distance<=8.0)
  {
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,HIGH);
    digitalWrite(led4,LOW);
    digitalWrite(led5,LOW);
    digitalWrite(led6,LOW);
    digitalWrite(led7,LOW);
    digitalWrite(led8,LOW);
    tone(buzzer, 1000);
    delay(300);
    noTone(buzzer);
    delay(300);
  }
  else if(distance>8.0 && distance<=10.0)
  {
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
    digitalWrite(led4,HIGH);
    digitalWrite(led5,LOW);
    digitalWrite(led6,LOW);
    digitalWrite(led7,LOW);
    digitalWrite(led8,LOW);
    tone(buzzer, 1000);
    delay(400);
    noTone(buzzer);
    delay(400);
  }
  else if (distance>10.0 && distance<=11.0)
  {
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
    digitalWrite(led4,LOW);
    digitalWrite(led5,HIGH);
    digitalWrite(led6,LOW);
    digitalWrite(led7,LOW);
    digitalWrite(led8,LOW);
    tone(buzzer, 1000);
    delay(500);
    noTone(buzzer);
    delay(500);
```

```
  else if (distance>11.0 && distance <=15.0)
  {
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
    digitalWrite(led4,LOW);
    digitalWrite(led5,LOW);
    digitalWrite(led6,HIGH);
    digitalWrite(led7,LOW);
    digitalWrite(led8,LOW);
    tone(buzzer, 1000);
    delay(600);
    noTone(buzzer);
    delay(600);
  }
  else if (distance>15.0 && distance<=20.0)
  {
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
    digitalWrite(led4,LOW);
    digitalWrite(led5,LOW);
    digitalWrite(led6,LOW);
    digitalWrite(led7,HIGH);
    digitalWrite(led8,LOW);
    tone(buzzer, 1000);
    delay(700);
    noTone(buzzer);
    delay(700);

  }
  else if(distance>20.0)
  {
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
    digitalWrite(led4,LOW);
    digitalWrite(led5,LOW);
    digitalWrite(led6,LOW);
    digitalWrite(led7,LOW);
    digitalWrite(led8,HIGH);
    tone(buzzer, 1000);
    delay(800);
    noTone(buzzer);
```

2η Άσκηση- Assembly Mips

Εδώ γράφουμε έναν κώδικα σε γλώσσα μηχανής (assembly mips) το οποίο υλοποιεί μια αριθμομηχανή .

Η αριθμομηχανή κάνει τις εξής πράξεις

1.Πρόσθεση

2.Αφαίρεση

3.Πολλαπλασιασμό

4.Διαίρεση

5.Ύψωση σε δύναμη του 2 του 1ου αριθμού που έδωσε ο χρήστης

6.Ύψωση σε δύναμη του 2 του 2ου αριθμού που έδωσε ο χρήστης

7.Έξοδος

Ο χρήστης αρχικά εισάγει δύο αριθμούς και μετά μπορεί να επιλέξει ποια ενέργεια θα κάνει πατώντας τον κατάλληλο αριθμό.

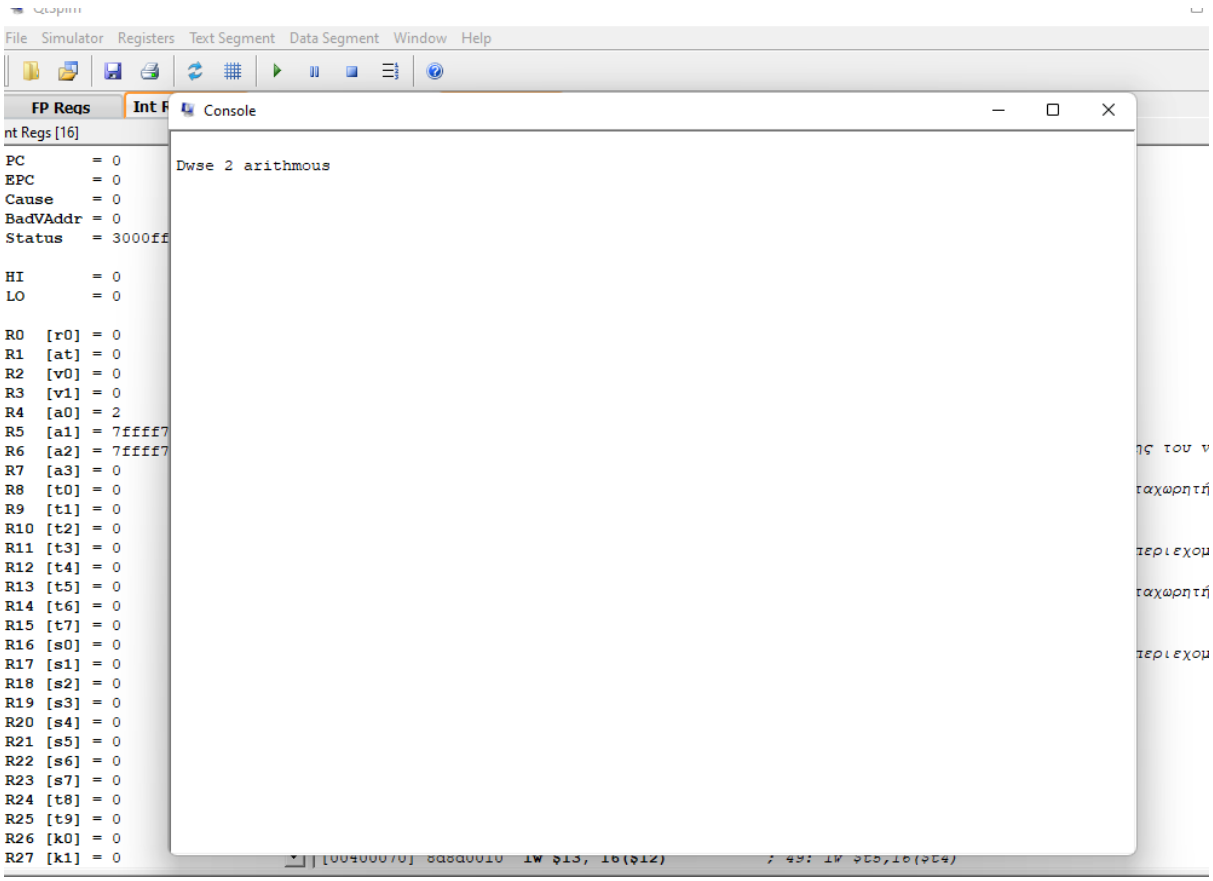
```
# Προσθήκη δύο αριθμών απο τον χρήστη

        .data
                value: .word 0, 0, 0
                msg1:  .asciiz "Sum="
                msg2:  .asciiz "afairesi="
                msg3:  .asciiz "pollaplasioasmos="
                msg4:  .asciiz "Diairesi="
                msg5:  .asciiz "Ypsosi se dunami tou 2 ston 1o aritmo="
                msg55:  .asciiz "Ypsosi se dunami tou 2 ston 2o aritmo="
                msg6:  .asciiz "Lathos epilogi, dwse ksana\n"
                msg0: .asciiz "\nDwse 2 arithmous\n"
                msg00:  .asciiz "Dwse
epilogi\n1.Προσθεση\n2.Αφαιρεση\n3.Πολλαπλασιασμος\n4.Διαιρεση\n5.Υψωση σε
δυναμη του 2 του 1ου αριθμου\n6.Υψωση σε δυναμη του 2 του 2ου
αριθμου\n7.EXIT\n"

        .text           # Μέρος text προγράμματος
        .globl main # Κύριο πρόγραμμα

main:

while:

        li $v0, 4
        la $a0, msg0
        syscall

        la $t0, value # Φόρτωση διεύθυνσης του value για αρχικοποίηση των
καταχωρητών
        li $v0, 5         # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 5
(syscall για read_int)
        syscall           # Κλήση syscall
        sw $v0, 0($t0)  # Αποθήκευση του περιεχομένου του καταχωρητή $v0 στην
θέση 0 του καταχωρητή $t0

# Σε αυτό το σημείο έχουμε διαβάσει την πρώτη τιμή

        li $v0, 5         # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 5
(syscall για read_int)
        syscall           # Κλήση syscall
        sw $v0, 4($t0)     # Αποθήκευση του περιεχομένου του καταχωρητή $v0
στην θέση 4 του καταχωρητή $t0


# TIP: αποθηκεύουμε +4 θέσεις επειδή η μνήμη μας είναι οργανωμένη σε byte
# Σε αυτό το σημείο έχουμε διαβάσει την δεύτερη τιμή
```

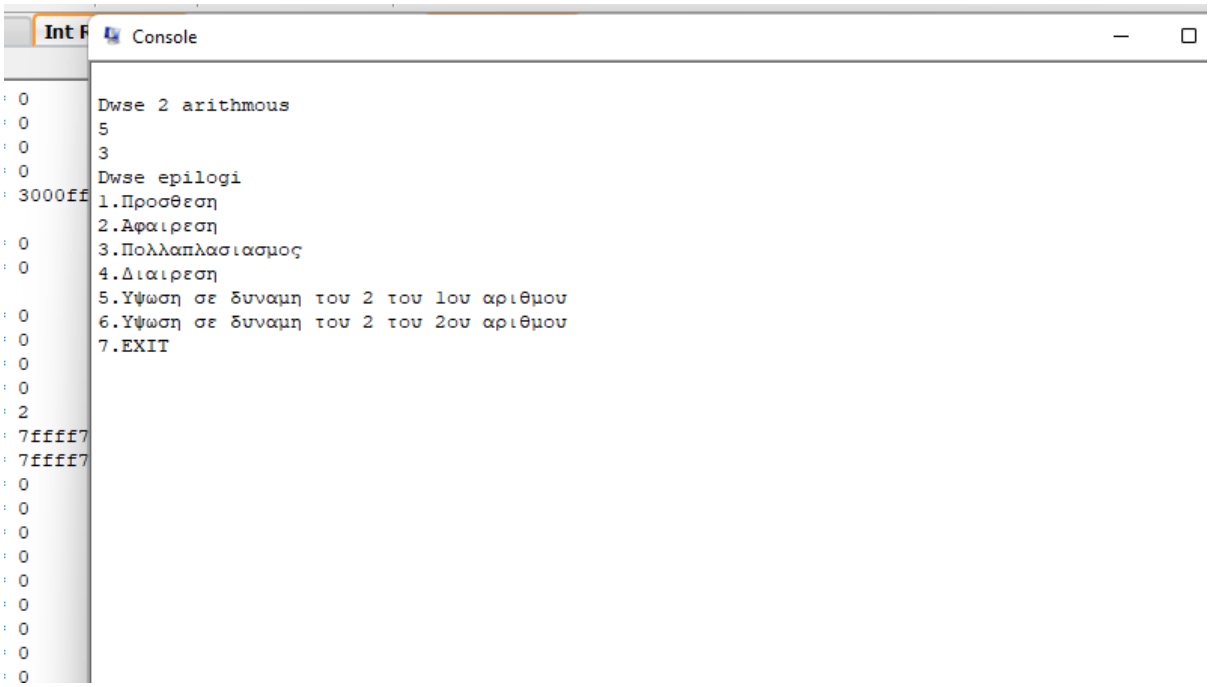
```
epilogi:
        li $v0, 4
        la $a0, msg00
        syscall
        la $t4,value
        li $v0,5
        syscall
        sw $v0,16($t4)
        lw $t5,16($t4)

        bne $t5,1,else



# Τώρα αρκεί να φορτώσουμε τα δεδομένα μας για να κάνουμε την πρόσθεση

        lw $t1, 0($t0)  # Φόρτωση των δεδομένων απο την θέση 0 του καταχωρητή
$t0 και αποθήκευση αυτου στον καταχωρητή $t1
        lw $t2, 4($t0)  # Φόρτωση των δεδομένων απο την θέση 4 του καταχωρητή
$t0 και αποθήκευση αυτου στον καταχωρητή $t2
        add $t3, $t1, $t2       # Πρόσθεση των τιμών των καταχωρητών $t1 και
$t2 και αποθήκευση αυτου στο καταχωρητή $t3
        sw $t3, 8($t0)  # Αποθήκευση του περιεχομένου του καταχωρητή $t3 στην
θέση 8 του καταχωρητή $t0

# Σε αυτο το σημείο έχει ολοκληρωθεί η πρόσθεση
# Εμφάνιση αποτελέσματος:

   li $v0, 4   # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 4 (syscall για
print_string)
   la $a0, msg1  # Φόρτωση της διεύθυνσης του msg1 στο καταχωρητή $a0
(argument to print_string call)
   syscall  # Κλήση syscall

# Μέχρι εδώ έχουμε εμφανίσει το μήνυμα Sum=

   li $v0, 1   # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 1 (syscall για
print_int)
   move $a0, $t3 # Φόρτωση στο καταχωρητή $a0 το περιεχόμενο του $t3
   syscall     # Κλήση syscall
   j while
else:

        bne $t5,2,else1

        lw $t1, 0($t0)
        lw $t2, 4($t0)
        sub $t3, $t1, $t2
        sw $t3, 8($t0)
```

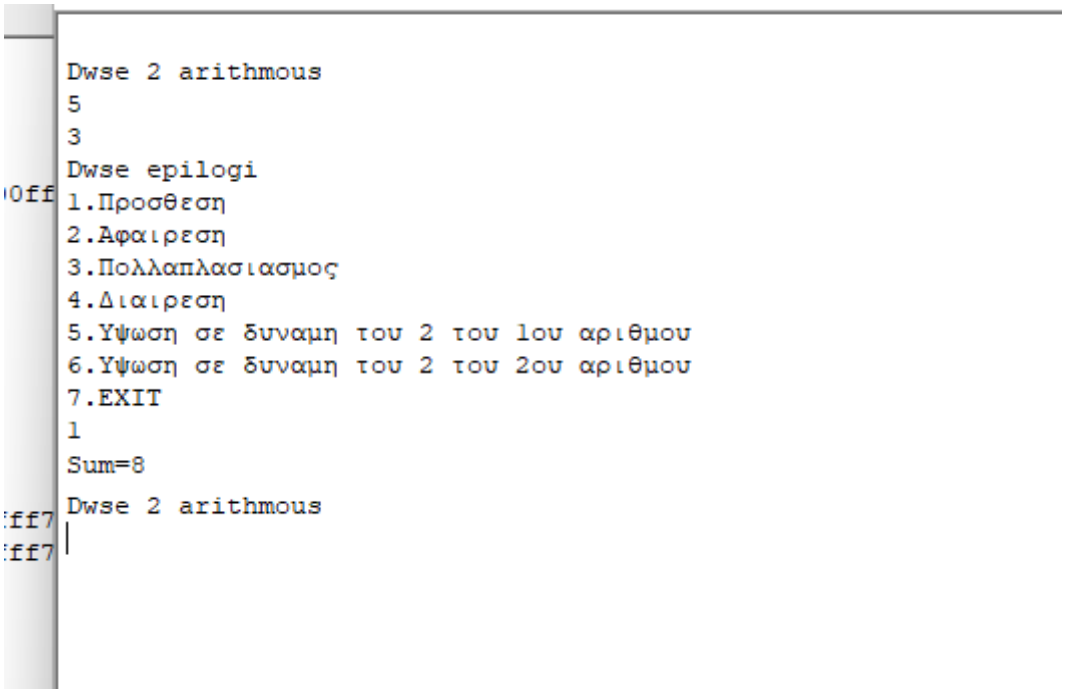
```
        li $v0, 4
        la $a0, msg2
        syscall

        li $v0, 1
        move $a0, $t3
        syscall
        j while
else1:
        bne $t5,3,else2

        lw $t1, 0($t0)
        lw $t2, 4($t0)
        mul  $t3, $t1, $t2
        sw $t3, 8($t0)

        li $v0, 4
        la $a0, msg3
        syscall

        li $v0, 1
        move $a0, $t3
        syscall
        j while

else2:
        bne $t5,4,else3

        lw $t1, 0($t0)
        lw $t2, 4($t0)
        div  $t3, $t1, $t2
        sw $t3, 8($t0)

        li $v0, 4
        la $a0, msg4
        syscall

        li $v0, 1
        move $a0, $t3
        syscall
        j while
else3:
        bne $t5,5,else4

        lw $t1, 0($t0)
        lw $t2, 4($t0)
        mul  $t3, $t1, $t1
```

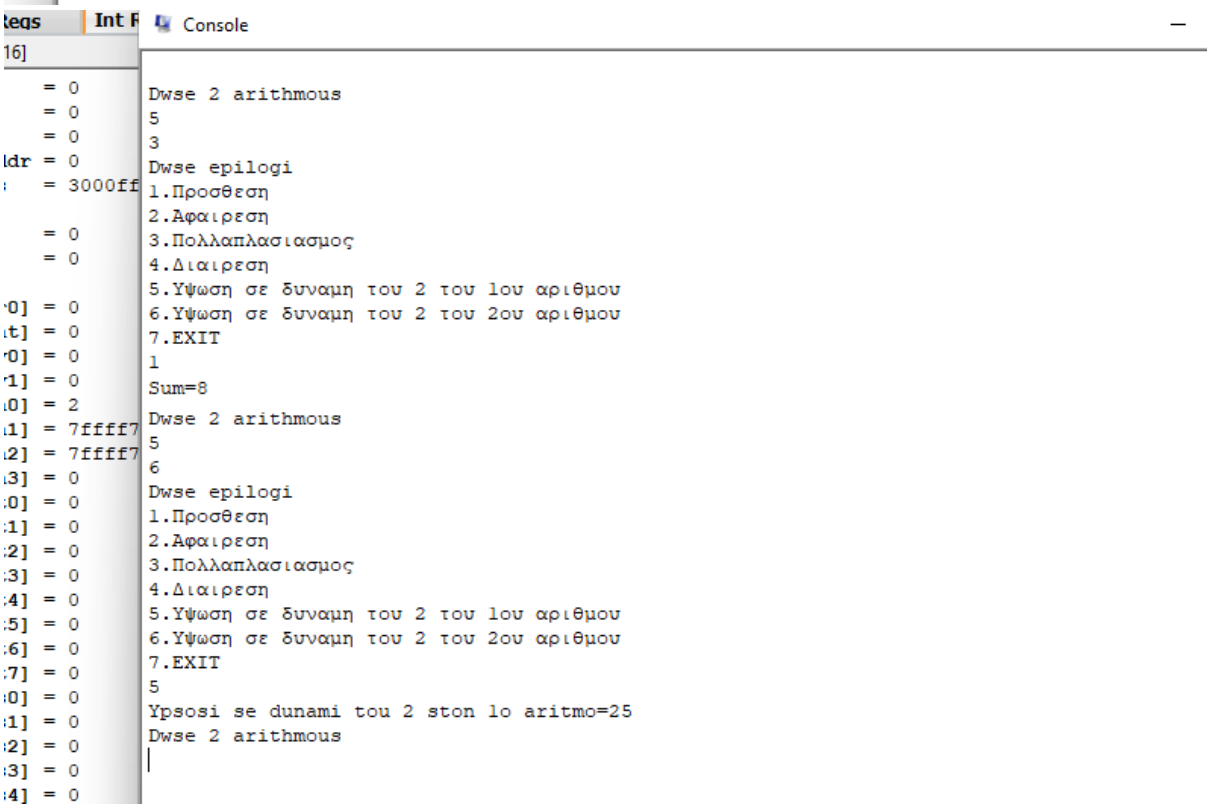
```
        sw $t3, 8($t0)

        li $v0, 4
        la $a0, msg5
        syscall

        li $v0, 1
        move $a0, $t3
        syscall
        j while


else4:
        bne $t5,6,else5

        lw $t1, 0($t0)
        lw $t2, 4($t0)
        mul  $t3, $t2, $t2
        sw $t3, 8($t0)

        li $v0, 4
        la $a0, msg55
        syscall

        li $v0, 1
        move $a0, $t3
        syscall
        j while

else5:
        bne $t5,7,else6
        li  $v0, 10 # Άμεση φόρτωση στο καταχωρητή $v0 την τιμή 10 (syscall
για exit)
        syscall  # Κλήση syscall

else6:  li $v0, 4
        la $a0, msg6
        syscall
        j epilogi
```

Μέσω του Qtspim τρέχουμε τον κώδικα






Dwse 2 arithmous
5
3
Dwse epilogi
1.Προσθεση
2.Αφαιρεση
3.Πολλαπλασιασμος
4.Διαιρεση
5.Υψωση σε δυναμη του 2 του 1ου αριθμου
6.Υψωση σε δυναμη του 2 του 2ου αριθμου
7.EXIT
1
Sum=8
Dwse 2 arithmous


Console
Dwse 2 arithmous
5
3
Dwse epilogi
1.Προσθεση
2.Αφαιρεση
3.Πολλαπλασιασμος
4.Διαιρεση
5.Υψωση σε δυναμη του 2 του 1ου αριθμου
6.Υψωση σε δυναμη του 2 του 2ου αριθμου
7.EXIT
1
Sum=8
Dwse 2 arithmous
5
6
Dwse epilogi
1.Προσθεση
2.Αφαιρεση
3.Πολλαπλασιασμος
4.Διαιρεση
5.Υψωση σε δυναμη του 2 του 1ου αριθμου
6.Υψωση σε δυναμη του 2 του 2ου αριθμου
7.EXIT
5
Ypsosi se dunami tou 2 ston 1o aritmo=25
Dwse 2 arithmous

3η Άσκηση -VHDL Mips

Στην τελευταία άσκηση φτιάχνουμε έναν μικροεπεξεργαστή 8-bit ο οποίος κάνει τις παρακάτω λειτουργίες

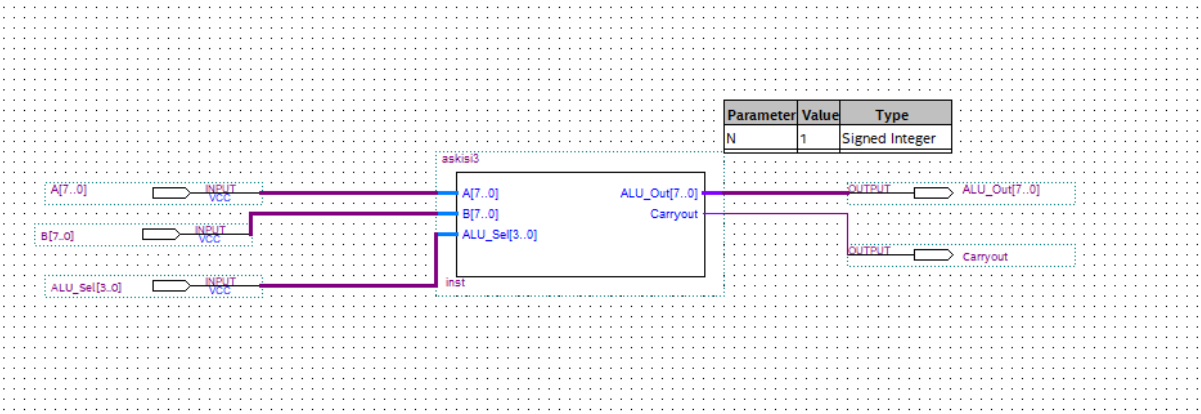
Είσοδος Ελέγχου	Λειτουργία:	Εξήγηση:
0000	A + B	Πρόσθεση
0001	A - B	Αφαίρεση
0010	A * B	Πολλαπλασιασμός
0011	A / B	Διαίρεση
0100	A sll N	Αριστερή ολίσθηση
0101	A srl N	Δεξιά ολίσθηση
0110	A rol N	Περιστροφή αριστερά
0111	A ror N	Περιστροφή δεξιά
1000	A and B	Λογική πράξη AND
1001	A or B	Λογική πράξη OR
1010	A xor B	Λογική πράξη XOR
1011	A nor B	Λογική πράξη NOR
1100	A nand B	Λογική πράξη NAND
1101	A xnor B	Λογική πράξη XNOR
1110	A > B	Σύγκριση αν A > B
1111	A = B	Σύγκριση αν A = B

Γράφουμε τον κώδικα σε γλώσσα VHDL με την βοήθεια του Quartus .

```
library IEEE;
use IEEE.STD_logic_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;    -- καλουμε τις καταλληλες βιβλιοθηκες
use ieee.NUMERIC_STD.all;


entity askisi3 is   -- οντοτητα
generic(
      constant N:natural:= 1
      );

      Port(
      A, B     : in  STD_LOGIC_VECTOR(7 downto 0);       -- εδω φτιαχνουμε τις εισοδους οι οποιοι θα εναι των 8 bit
      ALU_Sel  : in  STD_LOGIC_VECTOR(3 downto 0);       -- η εισοδος για την επιλογη της πραξης
      ALU_Out   : out  STD_LOGIC_VECTOR(7 downto 0); -- 1 output 8-bit
      Carryout : out std_logic
      );
end askisi3;

architecture askisi3 of askisi3 is   -- αρχιτεκτινικη και δηλωση σηματων για την μεταβλητη της εξοδου


signal ALU_Result : std_logic_vector (7 downto 0);
signal tmp: std_logic_vector (8 downto 0);


begin
process (A,B,ALU_Sel)
begin
   case(ALU_SEl)is     -- εδω ξεκιναει ο κωδικας και με την βοηθεια των ελεγχων μεσω case κανουμε την καθε πραξη
    when "0000"=>
       ALU_Result <= A + B ;
    when "0001" => -- Subtraction
    ALU_Result <= A - B ;
   when "0010" => -- Multiplication
    ALU_Result <= std_logic_vector(to_unsigned((to_integer(unsigned(A)) * to_integer(unsigned(B))),8)) ;
   when "0011" => -- Division
    ALU_Result <= std_logic_vector(to_unsigned(to_integer(unsigned(A)) / to_integer(unsigned(B)),8)) ;
   when "0100" => -- Logical shift left
    ALU_Result <= std_logic_vector(unsigned(A) sll N);
   when "0101" => -- Logical shift right
    ALU_Result <= std_logic_vector(unsigned(A) srl N);
   when "0110" => --  Rotate left
    ALU_Result <= std_logic_vector(unsigned(A) rol N);
   when "0111" => -- Rotate right
    ALU_Result <= std_logic_vector(unsigned(A) ror N);
   when "1000" => -- Logical and
    ALU_Result <= A and B;
   when "1001" => -- Logical or
    ALU_Result <= A or B;
   when "1010" => -- Logical xor
    ALU_Result <= A xor B;
   when "1011" => -- Logical nor
    ALU_Result <= A nor B;
   when "1100" => -- Logical nand
    ALU_Result <= A nand B;
```

```
    ALU_Result <= A xor B;
   when "1011" => -- Logical nor
    ALU_Result <= A nor B;
   when "1100" => -- Logical nand
    ALU_Result <= A nand B;
   when "1101" => -- Logical xnor
    ALU_Result <= A xnor B;
   when "1110" => -- συγκριση
    if(A>B) then
     ALU_Result <= x"01" ;
    else
     ALU_Result <= x"00" ;
    end if;
   when "1111" => --ισοτητα
    if(A=B) then
     ALU_Result <= x"01" ;
    else
     ALU_Result <= x"00" ;
    end if;
   when others => ALU_Result <= A + B ;
   end case;
  end process;

    ALU_Out <= ALU_Result; -- ALU out
  tmp <= ('0' & A) + ('0' & B);
  Carryout <= tmp(8); -- Carryout flag
    end askisi3;
```

Έπειτα δημιουργούμε το μπλοκ και συνδέουμε κατάλληλα τις εισόδους και τις εξόδους με σωστά καλώδια .



Τέλος μέσω της κυματομορφής δίνουμε μέσω των A και B τους αριθμούς της αρέσκειας μας και με το ALU SEL επιλέγουμε την πράξη που επιθυμούμε.Μερικά παραδείγματα