1η Ομάδα Ασκήσεων στα Συστήματα Μικροϋπολογιστών

6ο Εξάμηνο, Ακαδημαϊκή Περίοδος 2020 – 2021

Ονοματεπώνυμο	Αριθμός Μητρώου
Μπούφαλης Οδυσσεύς – Δημήτριος	el18118
Στεφανάκης Γεώργιος	el18436

Άσκηση 1η

Κάνοντας χρήση του πίνακα αναφοράς οδηγιών Assembly 8085 που αναφέρεται στην εκφώνηση της άσκησης, καταφέραμε να αποκωδικοποιήσουμε το δοθέν σε γλώσσα μηχανής πρόγραμμα ως εξής:

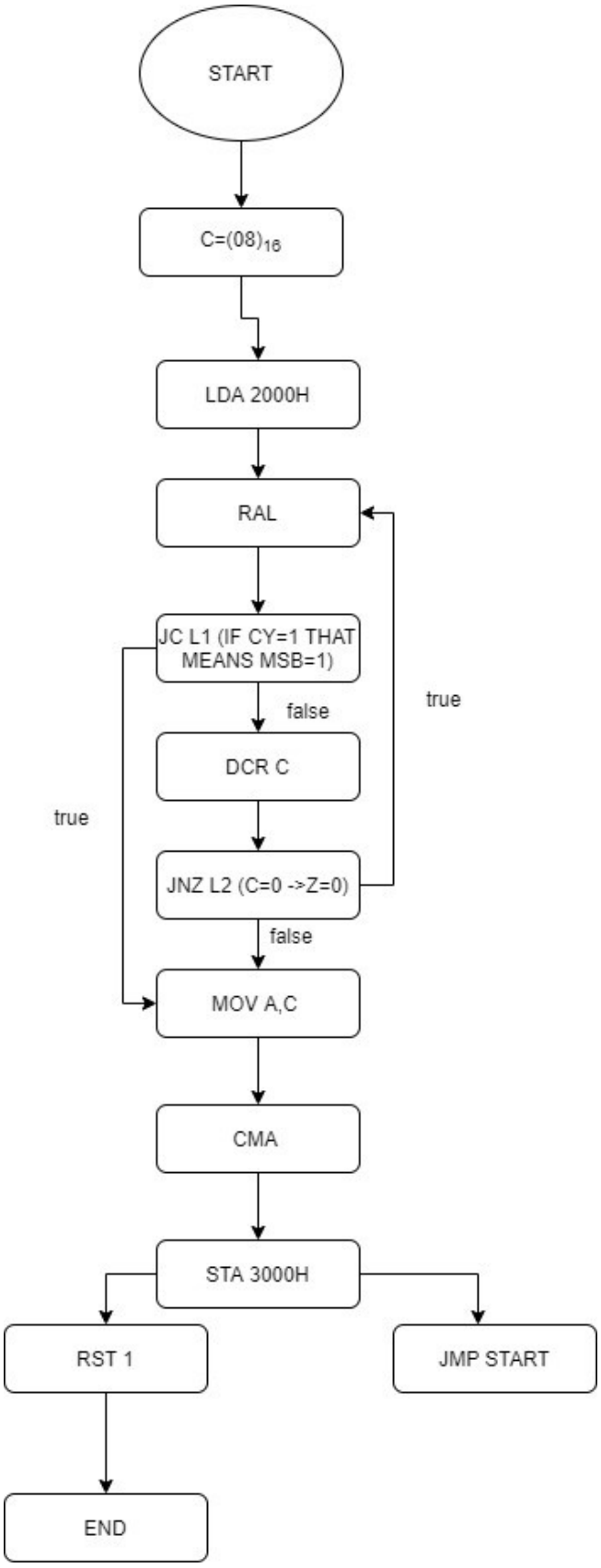
```
0E 08        ⇒        MVI C,08H
3A 00 20     ⇒        LDA 2000H
17           ⇒  L2:   RAL
DA 0D 08     ⇒        JC L1
0D           ⇒        DCR C
C2 05 08     ⇒        JNZ L2
79           ⇒  L1:   MOV A,C
2F           ⇒        CMA
32 00 30     ⇒        STA 3000H
CF           ⇒        RST 1
```

Στην παραπάνω αποκωδικοποίηση έχουμε χρησιμοποιήσει τις ετικέτες L1, L2 ώστε να κάνουμε πιο κατανοητή τη ροή του προγράμματος καθώς δεν είναι προφανές σε ποιο σημείο του κώδικα η εντολή άλματος μεταφέρει τον program counter. Το συμπέρασμα αυτό προέκυψε μελετώντας τη συνολική μνήμη που δεσμεύουν οι εντολές και παρατηρώντας ποια εντολή αντιστοιχεί στη θέση μνήμης που υπαγορεύει η εντολή JC να μεταφερθεί η εκτέλεση του προγράμματος. Παρακάτω δίνεται η έξοδος του MicroLab Simulator, για τον δεδομένο κώδικα, η οποία μας οπτικοποιεί το πως οι παραπάνω οδηγίες είναι αποθηκευμένες στη μνήμη του μικροεπεξεργαστή.

Διεύθυνση	Περιεχόμενο	Εντολή
0800	0E-opcode	MVI C,08H
0801	8-data	
0802	3A-data	LDA 2000H
0803	0-address	
0804	20-address	
L2:		
0805	17-opcode	RAL
0806	DA-opcode	JC L1
0807	0D-address	
0808	8-address	
0809	0D-opcode	DCR C
080A	C2-opcode	JNZ L2
080B	5-address	
080C	8-address	
L1:		
080D	79-opcode	MOV A,C
080E	2F-opcode	CMA
080F	32-opcode	STA 3000H
0810	0-address	
0811	30-address	
0812	CF-opcode	RST 1

Το πρόγραμμα που παραθέσαμε παραπάνω αρχικοποιεί το register C με την τιμή 8 (δηλ. το πλήθος των bits που υπάρχουν σε μία θέση μνήμης), φορτώνει το περιεχόμενο της θέσης μνήμης 2000H στον accumulator και μπαίνει σε έναν βρόχο. Σε αυτό τον βρόχο κάνει αριστερή κύλιση στο περιεχόμενο του A με αποτέλεσμα το μέχρι πρότινος MSB να αποθηκευτεί στο register CY. Ύστερα, μειώνεται μοναδιαία ο καταχωρητής C και ελέγχεται αν το περιεχόμενο του κρατουμένου είναι μηδέν ή ένα. Εάν είναι μηδέν ξαναμπαίνουμε στον βρόχο. Εάν είναι ένα, αποθηκεύουμε την τρέχουσα τιμή του C στο A και την συμπληρώνουμε ως προς ένα ώστε να την εμφανίσουμε στα LED's καθώς αυτά ακολουθούν αρνητική λογική. Έπειτα, το πρόγραμμα τερματίζει. Συμπερασματικά, το παραπάνω πρόγραμμα έχει ως έξοδο το πλήθος των σημαντικών ψηφίων (δηλ. το πλήθος των ψηφίων του αριθμού δίχως τα μηδενικά που προηγούνται πριν από τον πρώτο άσσο) της εισόδου. Για να πετύχουμε τη συνεχόμενη μορφή του παραπάνω αλγόριθμου μπορούμε να προσθέσουμε μία ετικέτα START στην πρώτη γραμμή του κώδικα και με εντολή άλματος σε εκείνη αμέσως πριν από την εντολή RST 1, να μεταφέρουμε το program counter στην εκκίνηση του προγράμματος. Το αρχείο πηγαίου κώδικα *ask1.8085* έχει επισυναπτεί μαζί με την παρούσα αναφορά και μπορεί να εκτελεστεί στον TSIK.

Παραθέτουμε επιπλέον το διάγραμμα ροής του αλγορίθμου που μόλις περιεγράφηκε.



Άσκηση 2η

```
START:
      IN   10H          ; input to accumulator
      MVI  A,01H        ; first led to be lit is 01H
      CMA               ; negative logic leds
      STA  3000H        ; display output
      CMA
      MOV  D,A          ; D reg represents current lighting led

      LXI  B,01F4H      ; B <- 0226H (550 dec)
      CALL DELB         ; delay 550 ms

INPUT:
      LDA  2000H        ; load input
      ANI  01H          ; mask lsb
      JZ   STANDBY      ; if lsb == 0 goto STANDBY else check msb

      LDA  2000H        ; else load input again
      ANI  80H          ; mask msb
      JNZ  RIGHT        ; if msb == 1 goto RIGHT
      JMP  LEFT         ; else goto LEFT

RIGHT:
      MOV  A,D
      RRC               ; shift current led one position right
      CMA
      STA  3000H        ; display output
      CALL DELB         ; delay
      CMA
      MOV  D,A          ; move A to D
      JMP  INPUT        ; goto INPUT

LEFT:
      MOV  A,D
      RLC               ; shift current led one position left
      CMA
      STA  3000H
      CALL DELB
      CMA
      MOV  D,A
      JMP  INPUT
```

```
STANDBY:
      MOV  A,D          ; assign current led to A
      CMA               ; negative logic led's
      STA  3000H        ; display current led
      CALL DELB         ; delay
      CMA
      JMP  INPUT        ; continue program
END
```

“ask2.8085”

Άσκηση 3η

```
START:
      IN 10H
      LXI B,01F4H       ; put in BC delay equal to 500

L2:
      LDA 2000H
      CPI 64H           ; compare A with 100
      JNC L1            ; if A >= 100 jump to L1
      MVI D,FFH         ; put FFH( whose 2's complement is -1)  to D

DECA:
      INR D             ; increase D by 1 in order to keep track of the num of 10's
      SUI 0AH           ; substitute 10 from A
      JNC DECA          ;
      ADI 0AH           ; when A becomes negative add 10 to obtain number of units
      MOV E,A           ; store at E number of units
      MOV A,D           ; store at A number of tens
      RLC               ; shift left 4 times in order to put the tens in the 4 MSB
      RLC
      RLC
      RLC
      ADD E             ; add number of units to A so that they appear at the 4 LSB
      CMA               ; complement because leds use negative logic
      STA 3000H         ; output to leds
      JMP L2            ; repeat

L1:
      CPI C8H           ; compare A with 200
      JNC L3            ; if A >= 200 jump to L3

      MVI A,F0H         ; put 0 to the 4 LSB
      STA 3000H         ; the 4 Least significant leds light up
      CALL DELB         ; delay
      MVI A,FFH         ; put 1 to all bits
      STA 3000H         ; turn off all the leds
      CALL DELB         ; delay
      JMP L2            ; repeat

L3:                     ; if A >= 200
      MVI A,0FH         ; put 1 to the 4 LSB
      STA 3000H         ; light up the 4 most significant leds
      CALL DELB         ; delay
      MVI A,FFH         ; put 1 to all bits
      STA 3000H         ; turn off all the leds
      CALL DELB         ; delay
      JMP L2            ; repeat
END
```

"ask3.8085"

Άσκηση 4η

1) Τεχνολογία 1 (Διακριτά Στοιχεία):
    - Αρχικό Κόστος Σχεδίασης: 20.000 €
    - Κόστος I.C ανά τεμάχιο: 10 €
    - Κόστος συναρμολόγησης ανά τεμάχιο: 10 €

    Η συνάρτηση Κόστους συναρτήσει των x τεμαχίων είναι:

    $\mathrm{K(x)} = 20.000 + 20\ \mathrm{x}$ και η συνάρτηση κόστους ανά τεμάχιο είναι η εξής: $\frac{\mathrm{K(x)}}{\mathrm{x}} = \frac{20.000}{\mathrm{x}} + 20.$

2) Τεχνολογία 2 (FPGAs):
    - Αρχικό Κόστος Σχεδίασης: 10.000 €
    - Κόστος I.C ανά τεμάχιο: 30 €
    - Κόστος συναρμολόγησης ανά τεμάχιο: 10 €

    Η συνάρτηση Κόστους συναρτήσει των x τεμαχίων είναι:

    $\mathrm{K(x)} = 10.000 + 40\ \mathrm{x}$ και η συνάρτηση Κόστους ανά τεμάχιο είναι η εξής: $\frac{\mathrm{K(x)}}{\mathrm{x}} = \frac{10.000}{\mathrm{x}} + 40.$

3) Τεχνολογία 3 (SoC-1):
    - Αρχικό Κόστος Σχεδίασης: 100.000 €
    - Κόστος I.C ανά τεμάχιο: 2 €
    - Κόστος συναρμολόγησης ανά τεμάχιο: 2 €

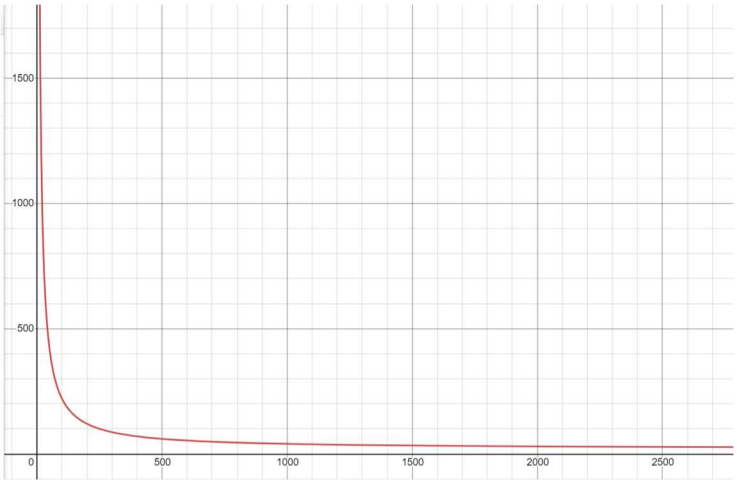
    Η συνάρτηση Κόστους συναρτήσει των x τεμαχίων είναι:

    $\mathrm{K(x)} = 100.000 + 4\ \mathrm{x}$ και η συνάρτηση Κόστους ανά τεμάχιο είναι η εξής: $\frac{\mathrm{K(x)}}{\mathrm{x}} = \frac{100.000}{\mathrm{x}} + 4.$

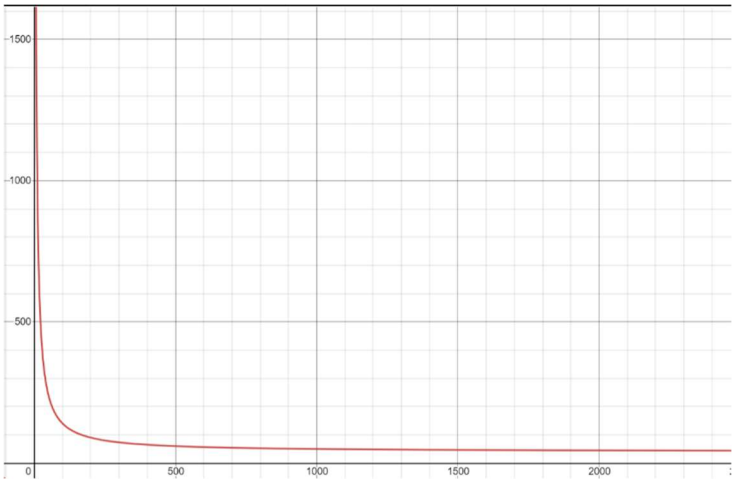
4) Τεχνολογία 4 (Soc-2):
    - Αρχικό Κόστος Σχεδίασης: 200.000 €
    - Κόστος I.C ανά τεμάχιο: 1 €
    - Κόστος συναρμολόγησης ανά τεμάχιο: 1 €

    Η συνάρτηση Κόστους συναρτήσει των x τεμαχίων είναι:

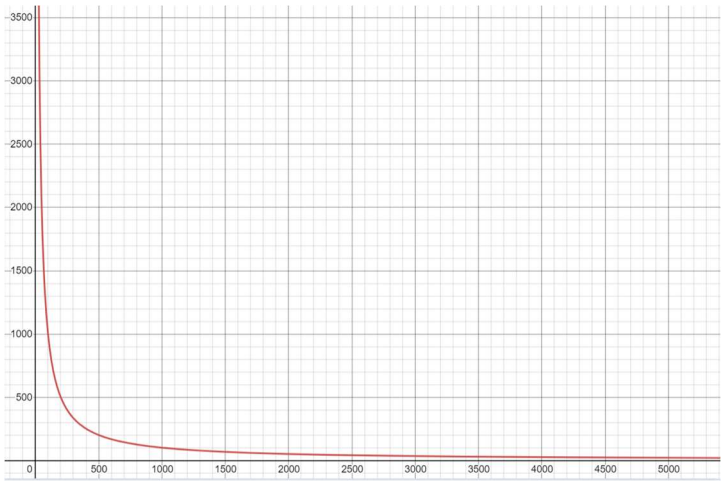
    $\mathrm{K(x)} = 200.000 + 2\ \mathrm{x}$ και η συνάρτηση Κόστους ανά τεμάχιο είναι η εξής: $\frac{\mathrm{K(x)}}{\mathrm{x}} = \frac{200.000}{\mathrm{x}} + 2.$



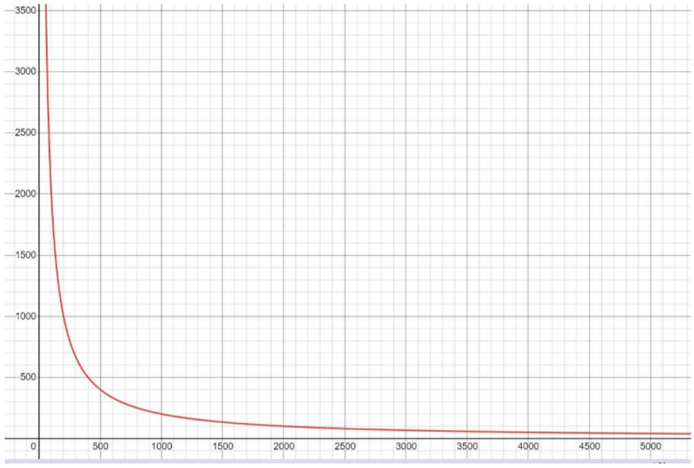
Εικόνα 1. Γράφημα κόστους ανά τεμάχιο Τεχνολογία 1



Εικόνα 2. Γράφημα κόστους ανά τεμάχιο Τεχνολογία 2

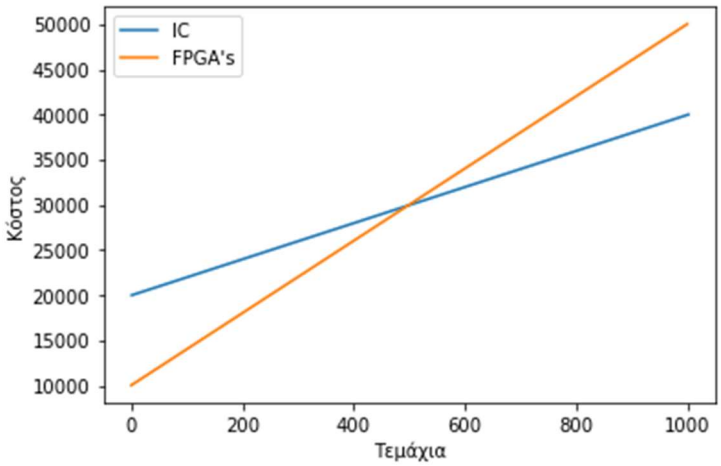


Εικόνα 3. Γράφημα κόστους ανά τεμάχιο Τεχνολογία 3

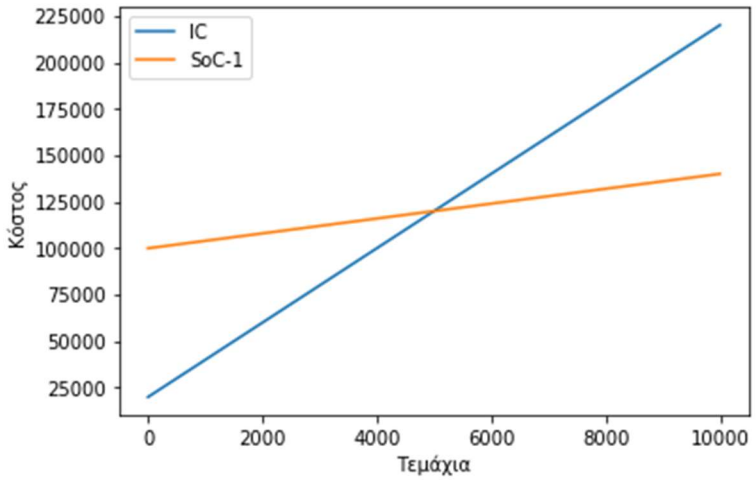


Εικόνα 4. Γράφημα κόστους ανά τεμάχιο Τεχνολογία 4

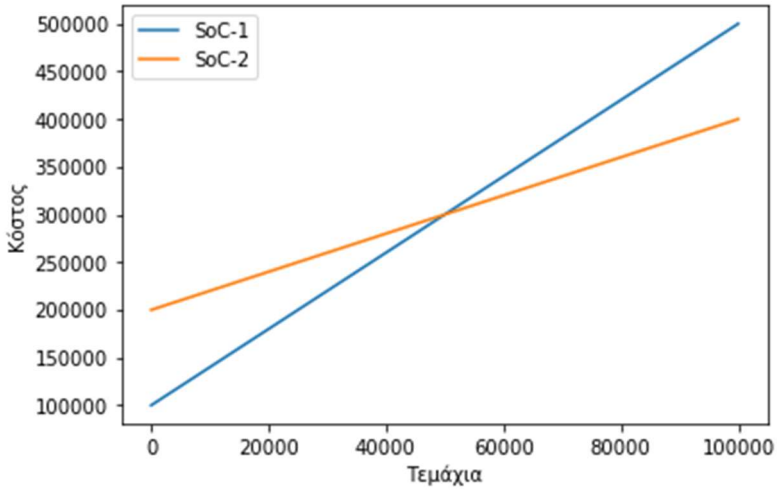
Παρακάτω υπολογίζουμε τις περιοχές πλήθους τεμαχίων για τις οποίες κάθε τεχνολογία θεωρείται συμφερότερη.



- Οι τεχνολογίες 1 και 2 εμφανίζουν ίδιο κόστος όταν: $20.000 + 20\text{x} = 10.000 + 40\text{x} \Rightarrow \text{x} = 500$ τεμάχια. Η καμπύλη της 2ης τεχνολογίας εμφανίζει μεγαλύτερη κλίση και μικρότερο αρχικό κόστος όπως φαίνεται από το γράφημα παραπάνω. Συνεπώς, για πλήθος τεμαχίων μικρότερο από 500 συμφέρει η 2η τεχνολογία ενώ για τεμάχια άνω των 500 συμφέρει η 1η τεχνολογία.
- Οι τεχνολογίες 1 και 3 εμφανίζουν ίδιο κόστος όταν: $20.000 + 20\text{x} = 100.000 + 4\text{x} \Rightarrow \text{x} = 5000$ τεμάχια. Η καμπύλη της 1ης τεχνολογίας έχει μικρότερο αρχικό κόστος και μεγαλύτερη κλίση όπως φαίνεται από το γράφημα. Συνεπώς, για πλήθος τεμαχίων κάτω από 5000 συμφέρει η 1η τεχνολογία ενώ για τεμάχια άνω των 5000 συμφέρει η 3η τεχνολογία.



- Οι τεχνολογίες 3 και 4 εμφανίζουν ίδιο κόστος όταν: $200.000 + 2x = 100.000 + 4x \Rightarrow x = 50.000$ τεμάχια. Η καμπύλη της 3ης τεχνολογίας έχει μικρότερο αρχικό κόστος και μεγαλύτερη κλίση όπως φαίνεται από το γράφημα. Συνεπώς, για πλήθος τεμαχίων κάτω από 50.000 συμφέρει η 3η τεχνολογία ενώ για τεμάχια άνω των 50.000 συμφέρει η 4η τεχνολογία.



Επομένως, συνολικά η 1η τεχνολογία προτιμάται για πλήθος πλακετών μεγαλύτερο από 500 και μικρότερο από 5000. Η 2η τεχνολογία προτιμάται για πλήθος πλακετών μικρότερο από 500. Η 3η τεχνολογία προτιμάται για πλήθος πλακετών μεγαλύτερο από 5000 και μικρότερο από 50.000. Τέλος, η 4η τεχνολογία προτιμάται για πλήθος πλακετών μεγαλύτερο από 50.000. Παρατηρούμε ότι όσο αυξάνεται το αρχικό κόστος σχεδίασης και μειώνεται το κόστος πλακέτας (I.C και συναρμολόγηση) τόσο πιο συμφέρουσα είναι η τεχνολογία για μεγαλύτερο πλήθος τεμαχίων το οποίο είναι λογικό αφού το αρχικό κόστος μοιράζεται σε περισσότερα τεμάχια.

Τέλος, πρέπει να διερευνήσουμε για ποια τιμή των I.C ανά τεμάχιο στην 2η Τεχνολογία (FPGA's) θα μπορούσε να εξαφανιστεί η επιλογή της 1η Τεχνολογίας. Σύμφωνα με την κατάταξη που κάναμε παραπάνω αρκεί η 2η τεχνολογία να είναι καλύτερη από την 1η για οποιοδήποτε αριθμό τεμαχίων x: $10.000 + (y + 10)x \leq 20.000 + 20x \Rightarrow (y - 10)x \leq 10.000$ για κάθε x συνεπώς $y - 10 \leq 0 \Rightarrow y \leq 10$.

Άσκηση 5η

i)

$F_1 = A\,(BC + D) + B'C'D$

```
module Circuit_A (F1, A, B, C, D);
    Input A, B, C, D;
    Output F1;
    wire w, x, y, z, a, d;
    and (w, B , C);
    or (x, w, D);
    and (y , x , A);
    not (z, B);
    not (a, C);
    and (d , D , z, a);
    or (F1, d , y);
endmodule
```

$F_2(A, B, C, D) = \Sigma\,(0, 2, 3, 5, 7, 9, 10, 11, 13, 14)$

```
primitive UDP_F2 (F2, A, B, C, D);
    Output F2;
    Input A, B, C, D;
    table
       0 0 0 0 : 1;
       0 0 0 1 : 0;
       0 0 1 0 : 1;
       0 0 1 1 : 1;
       0 1 0 0 : 0;
       0 1 0 1 : 1;
       0 1 1 0 : 0;
       0 1 1 1 : 1;
       1 0 0 0 : 0;
       1 0 0 1 : 1;
       1 0 1 0 : 1;
       1 0 1 1 : 1;
       1 1 0 0 : 0;
       1 1 0 1 : 1;
       1 1 1 0 : 1;
       1 1 1 1 : 0;
    endtable
endprimitive
```

$F_3 = ABC + (A + BC)D + (B + C)DE$

```
module Circuit_C (F3, A, B, C, D, E);
    Input A, B, C, D, E;
    Output F3;
    wire w, x, y, z, a, b, c, d;
    and (w, B, C); // w = BC
    and (x, w, A); // x = ABC
    or (y, w, A); // y = A+BC
    and (z, y, D); // z = (A+BC)D
    or (a, B, C); // a = B+C
    and (b, D, E); // b = DE
    and (c, a, b); // c = DE(B+C)
    or (d, x, z); // d = ABC + (A+BC)D
    or (F3, d, c); // F3 = ABC + (A+BC)D + DE(B+C)
endmodule
```

$F_4 = A(B + CD + E) + BCDE$

```
module Circuit_D (F4, A, B, C, D, E);
    Input A, B, C, D, E;
    Output F4;
    wire w, x, y, z, a, b;

    and (w, C, D); // w = CD
    or (x, w, B); // x = CD+B
    or (y, x, E); // y = CD+B+E
    and (z, y, A); // z = A(CD+B+E)

    and (a, B, E); // a = BE
    and (b, a, w); // b = BECD
    or (F4, b, z); // F4 = BECD+A(CD+B+E)
endmodule
```

ii)

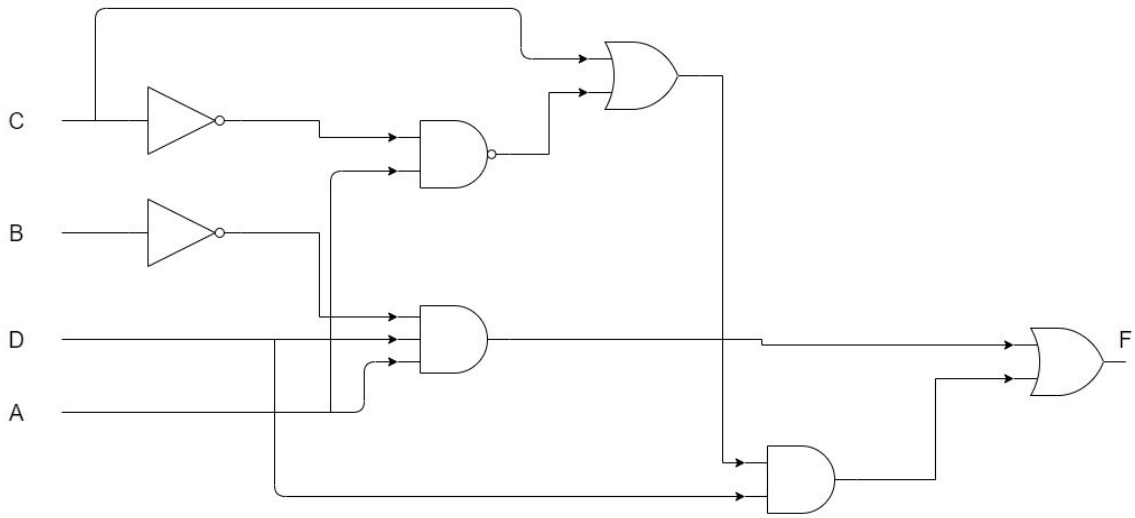
```
module Circuit_A (F1, A, B, C, D);
    Input A, B, C, D;
    Output F1;
    Assign F1 = ((B & C) | D) & A) | (~B & ~C & D);
endmodule
```

```
module Circuit_B (F2, A, B, C, D);
    Input A, B, C, D;
    Output F2;
    Assign F2 = (~A & ~B & ~C & ~D) | (~A & ~B & C & ~D) |
          (~A & ~B & C & D) | (~A & B & ~C & D) |
          (~A & B & C & D) | (A & ~B & ~C & D) |
          (A & ~B & C & ~D) | (A & ~B & C & D) |
          (A & B & ~C & D) | (A & B & C & ~D);
endmodule

module Circuit_C (F3, A, B, C, D, E);
    Input A, B, C, D, E;
    Output F3;
    Assign F3 = (A&B&C) | ((A | B&C) & D) | ((B | C) & (D & E));
endmodule

module Circuit_D (F4, A, B, C, D, E);
    Input A, B, C, D, E;
    Output F4;
    Assign F4 = (A & (B | (C & D) | E)) | (B & C & D & E);
endmodule
```

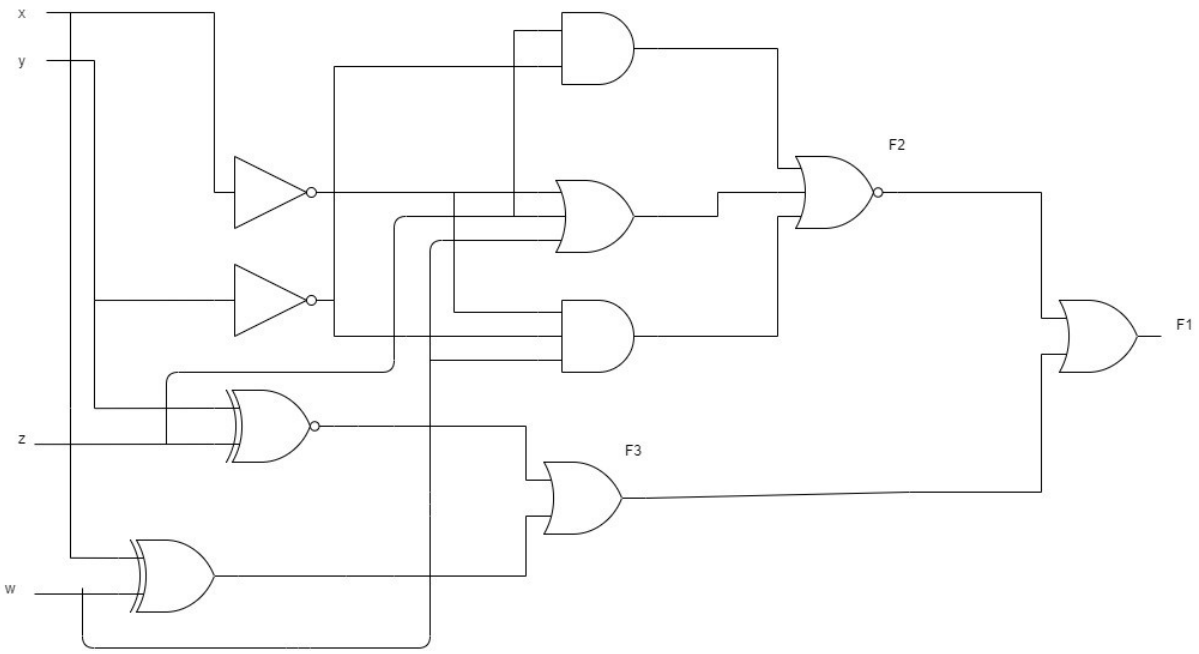
Άσκηση 6η

i)

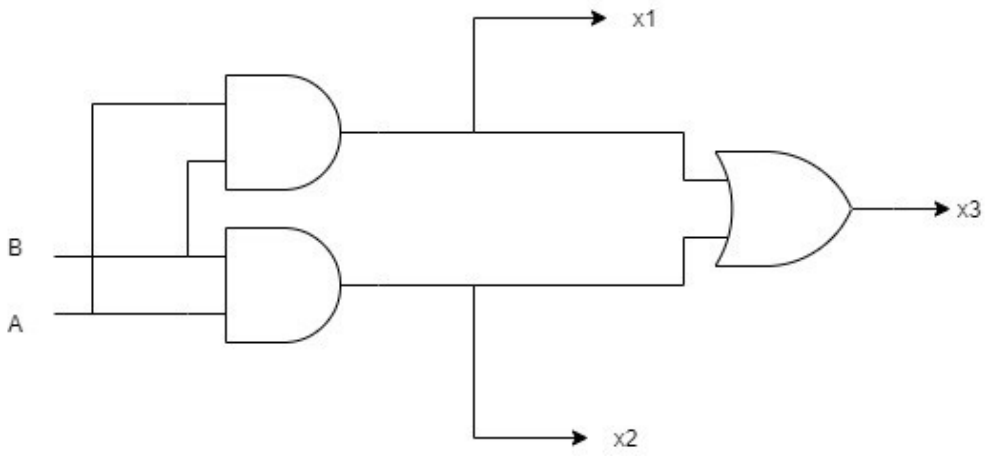
a)



b)



c)



ii)

```
module half_adder (output S, C, input x, y);
    xor(S,x,y);
    and(C,x,y);
endmodule

module full_adder (output S, C, input x, y, z);
    wire S1,C1,C2;
    half_adder HA1(S1,C1,x,y);
               HA2(S,C2,S1,z);
    or G1(C,C2,C1);
endmodule

module four_bit_adder_subtracter (Sum, C4, A, B, M);
    output [3:0] Sum;
    output C4;
    input[3:0] A, B;
    input M; // if M=1 then subtracter else if M=0 adder, M=C0
    wire C1,C2,C3;
    xor (Y0,M,B[0]),
        (Y1,M,B[1]),
        (Y2,M,B[2]),
        (Y3,M,B[3]);
    full_adder FA0(Sum[0],C1,A[0],Y0,M),
               FA1(Sum[1],C2,A[1],Y1,C1),
               FA0(Sum[2],C3,A[2],Y0,C2),
               FA0(Sum[3],C4,A[3],Y0,C3);
endmodule



module four_bit_adder_subtracter_dataflow (Sum, Cout, A, B, M);
    output [3:0] Sum;
    output Cout;
    input [3:0] A,B;
    input M;
    assign {Cout,Sum} = (M) ? A-B+M : A+B+M;
endmodule
```

Άσκηση 7η

i)

```
module Mealy_Model (out, x, clock, reset);
    output out;
    input x, clock, reset;
    reg [1: 0] state;
    reg out
    parameter a = 2'b00,
              b = 2'b01,
              c = 2'b10,
              d = 2'b11;
    always @ (posedge clock, negedge reset)
    if (reset == 0) state <= a;
    else case (state)
        a: if (~x) begin
                out=1;state <= d;
                end
        else begin
             out=0;state <= a;
             end
        b: if (~x) begin
                out=1;state <= c;
                end
        else begin
                out=0;state <= a;
                end
        c: if (~x) begin
                out=1;state <= d;
                end
        else begin
                out=0;state <= b;
                end
        d: if (~x) begin
                out=0;state <= c;
                end
        else begin
                out=1;state <= d;
                end
    endcase
endmodule
```

ii)

```
module Moore_Model (y, x, clock, reset);
     output y;
     input x, clock, reset;
     reg [1: 0] state;
     parameter a = 2'b00,
                 b = 2'b01,
                 c = 2'b10,
                 d = 2'b11;
     always @ (posedge clock, negedge reset)
     if (reset == 0) state <= a;
          else case (state)
          a: if (~x) state <= d; else state <= a;
          b: if (~x) state <= c; else state <= a;
          c: if (~x) state <= b; else state <= d;
          d: if (~x) state <= c; else state <= d;
     endcase
     assign y = state[0]^state[1];
endmodule
```