# Εθνικό Μετσόβιο Πολυτεχνείο

# Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

### 1η Ομάδα Ασκήσεων

**Μάθημα:** Συστήματα Μικροϋπολογιστών

**Εξάμηνο:** 6ο

**Ονοματεπώνυμα:** Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

## ***Άσκηση 1***

Το πρόγραμμα που δόθηκε σε γλώσσα μηχανής είναι το εξής:

**0E** 08 **3A** 00 20 **17 DA** 0D 08 **0D C2** 05 08 **79 2F 32** 00 30 **CF**

Για να αποκωδικοποιήσουμε το πρόγραμμα, χρησιμοποιούμε τον πίνακα 2 του παραρτήματος 2 των σημειώσεις.

Μεταφράζοντας το πρόγραμμα της άσκησης από γλώσσα μηχανής σε γλώσσα Assembly λαμβάνουμε το παραπάνω, το οποίο περιέχεται στο αρχείο askisi1a.8085. Αντίστοιχα με την μετατροπή Πρόγραμμα → Μνήμη και εντολές λαμβάνουμε το παρακάτω:

```
MVI C,08H  ; Load 08h into register C
LDA 2000H  ; Load contents of memory location 2000h into accumulator
RAL        ; Rotate accumulator left
JC 080DH   ; Jump to 080Dh if carry flag is set
DCR C      ; Decrement the value in register C
JNZ 0805H  ; Jump to 0805h if zero flag is not set
MOV A,C    ; Move the value of C into accumulator
CMA        ; Complement accumulator
STA 3000h  ; Store contents of accumulator into memory location 3000h
RST 1      ; Restart the program from address 0000h

END
```

```
0800   0E     MVI C,08H
0801   08
0802   3A     LDA 2000H
0803   00
0804   20
0805   17     RAL
0806   DA     JC RESET
0807   00
0808   00
0809   0D     DCR C
080A   C2     JNZ RESET
080B   00
080C   00
080D   79     MOV A,C
080E   2F     CMA
080F   32     STA 3000H
0810   00
0811   30
0812   CF     RST 1
```

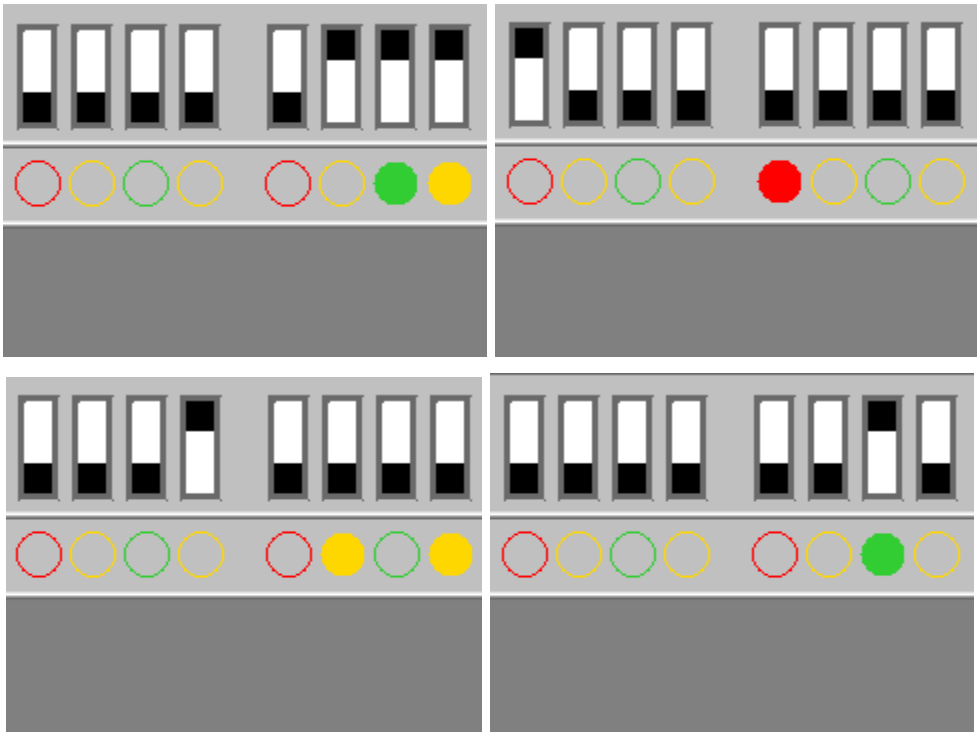
Το πρόγραμμα διαβάζει την τιμή του MSB από τους διακόπτες εισόδου και την μετατρέπει σε δυαδική μορφή για να εμφανιστεί στις λυχνίες εξόδου. Για να εκτελείται συνεχώς, πρέπει να προστεθεί μια εντολή άλματος στο τέλος του προγράμματος που θα μεταφέρει την εκτέλεση στην αρχή του προγράμματος. Επίσης, προκειμένου να μεταφραστεί και να εκτελεστεί το παραπάνω πρόγραμμα, θα πρέπει να αντκαταστήσουμε τις διευθύνσεις των συνθηκών άλματος με ετικέτες, όποως φαίνεται και στο τελικό πρόγραμμα (αρχείο askisi1b.8085)

```
START:
       MVI C,08H
       LDA 2000H
POINT1:
       RAL
       JC POINT2
       DCR C
       JNZ POINT1
POINT2:
       MOV A,C
       CMA
       STA 3000H
       JMP START

       END
```

Η λειτουργία του προγράμματος αυτού είναι να εμφανίζει σε δυαδική αναπαράσταση στα φωτάκια LED (αριθμημένα από δεξιά προς τα αριστερά κατά 1, 2,… ,8) τον αριθμό της θέσης

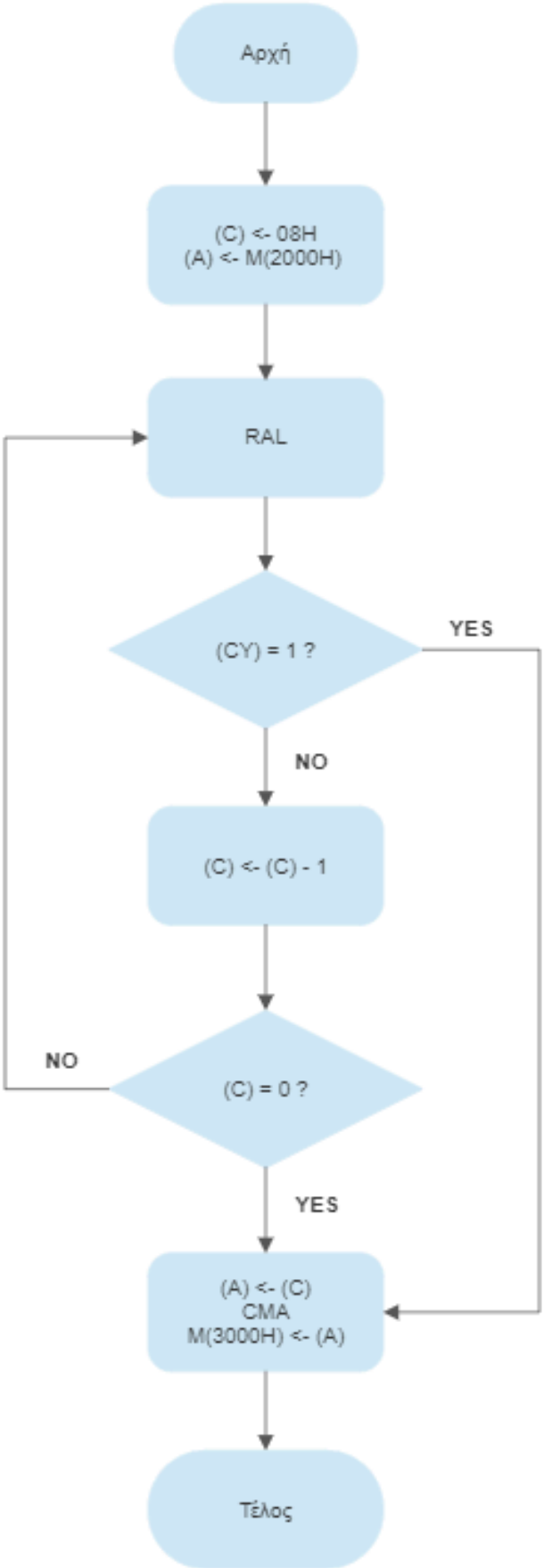
του πιο αριστερού dip switch που είναι ON ή, από άλλη οπτική, επιστρέφει το συμπλήρωμα ως προς 1 (αριθμημένα από αριστερά προς τα δεξιά κατά 0, 1,… ,7) του αριθμού της θέσης του πιο αριστερού dip switch που είναι ON. Η τιμή αυτή αναπαριστάται δυαδικά από τα αναμένα LEDs (Αντίστροφη λογική, 0 → *ON*, 1 → *OFF*.)



Παρακάτω φαίνεται το διάγραμμα ροής του προγράμματος που απεικονίζει τη λειτουργία ενός προγράμματος που ξεκινά από το σύμβολο Αρχή. Στην αρχή του προγράμματος, η τιμή 08H φορτώνεται στον καταχωρητή C και φορτώνονται τα περιεχόμενα της θέσης μνήμης 2000H στον συσσωρευτή χρησιμοποιώντας την οδηγία LDA.

Στη συνέχεια, το πρόγραμμα εισάγει έναν βρόχο στο POINT1 όπου ο συσσωρευτής περιστρέφεται αριστερά χρησιμοποιώντας την εντολή RAL. Εάν έχει οριστεί το carry flag, το πρόγραμμα μεταβαίνει στο POINT2 χρησιμοποιώντας την εντολή JC. Διαφορετικά, το πρόγραμμα μειώνει την τιμή στον καταχωρητή C χρησιμοποιώντας την εντολή DCR και μεταπηδά πίσω στο POINT1 εάν το zero flag δεν έχει οριστεί χρησιμοποιώντας την εντολή JNZ.

Στο POINT2, το πρόγραμμα μετακινεί την τιμή του καταχωρητή C στον συσσωρευτή χρησιμοποιώντας την εντολή MOV και στη συνέχεια συμπληρώνει τον συσσωρευτή χρησιμοποιώντας την εντολή CMA. Τέλος, το πρόγραμμα αποθηκεύει τα περιεχόμενα του συσσωρευτή στη θέση μνήμης 3000H χρησιμοποιώντας την εντολή STA.


Αρχή
(C) <- 08H
(A) <- M(2000H)
RAL
(CY) = 1 ?
YES
NO
(C) <- (C) - 1
(C) = 0 ?
NO
YES
(A) <- (C)
CMA
M(3000H) <- (A)
Τέλος

## ***Άσκηση 2***

Το πρόγραμμα που ζητείται στην άσκηση αυτή παρατίθεται παρακάτω και περιέχεται στο αρχείο askisi2.8085.

```
IN 10H
LXI B,01F4H
MVI E,01H

START:
  ; Load input from memory address 2000H into accumulator A
  LDA 2000H
  ; Copy value of A to register D
  MOV D,A
  ; Rotate right through carry flag, checking carry to see if MSB is set
  RRC
  JC START
  ; Call delay subroutine DELB
  CALL DELB
  ; Copy value of D back to A
  MOV A,D
  ; Rotate left through carry flag, checking carry to see if MSB is set
  RLC
  JC RIGHT

LEFT:
  ; Move value of E into A and then take the 1's complement
  MOV A,E
  CMA
  ; Store complement of A in memory location 3000H
  STA 3000H
  ; Take 1's complement of A again
  CMA
  ; Rotate left through carry flag
  RLC
  ; Move result back into E
  MOV E,A
  ; Jump back to start of loop
  JMP START

RIGHT:
  ; Move value of E into A and then take the 1's complement
  MOV A,E
  CMA
  ; Store complement of A in memory location 3000H
  STA 3000H
  ; Take 1's complement of A again
  CMA
  ; Rotate right through carry flag
  RRC
  ; Move result back into E
  MOV E,A
  ; Jump back to start of loop
  JMP START

END
```

Το πρόγραμμα λειτουργεί ως εξής:

- Η πρώτη γραμμή του προγράμματος αρχικοποιεί τις θύρες εισόδου/εξόδου του συστήματος εκπαίδευσης μLAB.

- Το πρόγραμμα χρησιμοποιεί καταχωρητές B και C για να εισάγει μια χρονική καθυστέρηση μεταξύ κάθε κίνησης LED.

- Το πρόγραμμα ορίζει την αρχική θέση του αναμμένου LED ως το λιγότερο σημαντικό bit (LSB) της θύρας εξόδου, φορτώνοντας την τιμή 1 στον καταχωρητή E.

- Το πρόγραμμα εισάγει έναν άπειρο βρόχο που διαβάζει την τιμή της θύρας εισόδου του dip switch (2000H) και τη μετατοπίζει δεξιά κατά ένα bit (χρησιμοποιώντας την εντολή RRC). Εάν έχει οριστεί το carry flag (που υποδεικνύει ότι το πιο σημαντικό bit είναι 1), το πρόγραμμα μεταβαίνει στην ετικέτα RIGHT. Διαφορετικά, το πρόγραμμα συνεχίζει στην επόμενη οδηγία.

- Το πρόγραμμα καλεί την υπορουτίνα DELB για να εισάγει μια καθυστέρηση πριν μετακινήσετε το LED στην επόμενη θέση.

- Το πρόγραμμα φορτώνει την τρέχουσα θέση LED (αποθηκευμένη στον καταχωρητή D), τη μετατοπίζει προς τα αριστερά κατά ένα bit (χρησιμοποιώντας την οδηγία RLC) και ελέγχει το carry flag. Εάν έχει οριστεί το carry flag, το πρόγραμμα μεταβαίνει στην ετικέτα RIGHT. Διαφορετικά, το πρόγραμμα μετακινείται στην επόμενη εντολή.

- Εάν το πρόγραμμα φτάσει στην ετικέτα LEFT, φορτώνει την τιμή στον καταχωρητή E, παίρνει το συμπλήρωμά του (χρησιμοποιώντας την εντολή CMA) και το αποθηκεύει στη θύρα εξόδου (3000H). Στη συνέχεια, χρειάζεται το συμπλήρωμα της σημαίας μεταφοράς για να αποθηκεύσει την τρέχουσα θέση του LED στον καταχωρητή E. Τέλος, το πρόγραμμα μεταπηδά πίσω στην ετικέτα START για να συνεχίσει τον βρόχο.

- Εάν το πρόγραμμα φτάσει στην ετικέτα RIGHT, φορτώνει την τιμή στον καταχωρητή E, παίρνει το συμπλήρωμά του (χρησιμοποιώντας την εντολή CMA), το αποθηκεύει στη θύρα εξόδου (3000H) και παίρνει το συμπλήρωμα του σημαία μεταφοράς για να αποθηκεύσει το ρεύμα θέση του LED στον καταχωρητή E. Στη συνέχεια, το πρόγραμμα μεταβαίνει πίσω στην ετικέτα START για να συνεχίσει τον βρόχο.

***Ασκηση 3***

```
START:
    LDA 2000H         ; Load input binary number from memory
    ANI 0FH          ; Mask to get the lower 8 bits only
    CPI 64H           ; Compare with 100 (decimal)
    JNC FLASH_LSD     ; If greater than or equal to 100,
flash LSDs
    MVI C,0AH        ; Set up divisor for decimal conversion
    MOV B,A          ; Copy input number to B
    LXI H,0000H ; Load memory location 0000H into the H-L
pair
    MVI M,00H    ; Move 0 into memory location 0000H
    MOV A,M      ; Move the value of memory location 0000H
to the A register
             ; Clear accumulator
    MVI D,00H        ; Set MSD to "0"
DIV_LOOP:
    CMP D         ; Compare input number with divisor
    JC END_DIV_LOOP ; If B < C, exit loop
    SUB C            ; Subtract divisor from B
    INR D            ; Increment MSD
    JMP DIV_LOOP
END_DIV_LOOP:
    MOV A,D          ; Move MSD to accumulator
    ADI 30H          ; Convert MSD to ASCII character
    STA 3001H        ; Store MSD in memory
    MOV A,B          ; Move LSD to accumulator
    ADI 30H          ; Convert LSD to ASCII character
    STA 3000H        ; Store LSD in memory
    JMP END          ; Done, halt
FLASH_LSD:
    MVI A,0FH       ; Set up LSD flash pattern
    MVI B,0AH        ; Set up divisor for decimal conversion
```

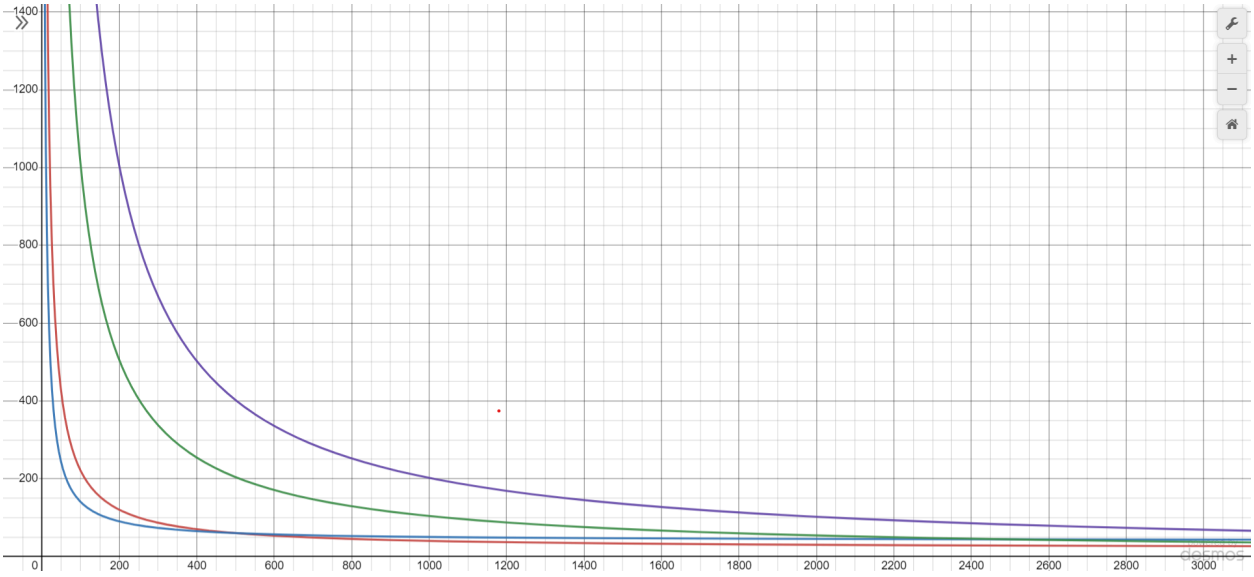
```
    LXI H,4000H     ; Set up delay counter in memory
location 4000H
    MVI M,0FH      ; Set the MSB of the delay counter to
FFH
    MVI A,C8H      ; Set the LSB of the delay counter to
C8H (200 in decimal)
    STA 4001H       ; Store the LSB of the delay counter in
memory location 4001H
    JMP FLASH       ; Jump to flash routine
FLASH_MSD:
    MVI A,20H      ; Set up MSD flash rate delay
    MVI B,F0H       ; Set up MSD flash pattern
    JMP FLASH        ; Jump to flash routine
FLASH:
    STA 3000H       ; Output MSD
    MOV D,A         ; Load flash rate delay
LOOP1:
    DCX D            ; Decrement delay counter
    JNZ LOOP         ; Loop until delay is complete
    STA 3000H           ; Output flash pattern
    DCX D            ; Reset delay counter
    JNZ LOOP1         ; Loop until delay is complete
    JMP START        ; Jump back to main loop
END:
    END
```

## ***Άσκηση 4***

Για κάθε διαφορετική τεχνολογία που χρησιμοποιείται, η σχέση κόστους ανά τεμάχιο διαφοροποιείται, με βάση τα δεδομένα της άσκησης. Έτσι, έχουμε:

1. Για χρήση διακριτών στοιχείων και I.C.:
   $f_1(x) = \frac{20.000}{x} + 20$
2. Για χρήση FPGAs και μικρού αριθμού περιφερειακών:
   $f_2(x) = \frac{10.000}{x} + 40$
3. Για σχεδίαση ειδικού SoC-1:
   $f_3(x) = \frac{100.000}{x} + 4$
4. Για σχεδίαση ειδικού SoC-2:
   $f_4(x) = \frac{200.000}{x} + 2$

Θα σχεδιάσουμε καθεμία από τις παραπάνω συναρτήσεις κόσμου στο ίδιο διάγραμμα (για $x \geq 0$), ώστε να διευκολύνεται και η σύγκρισή τους:

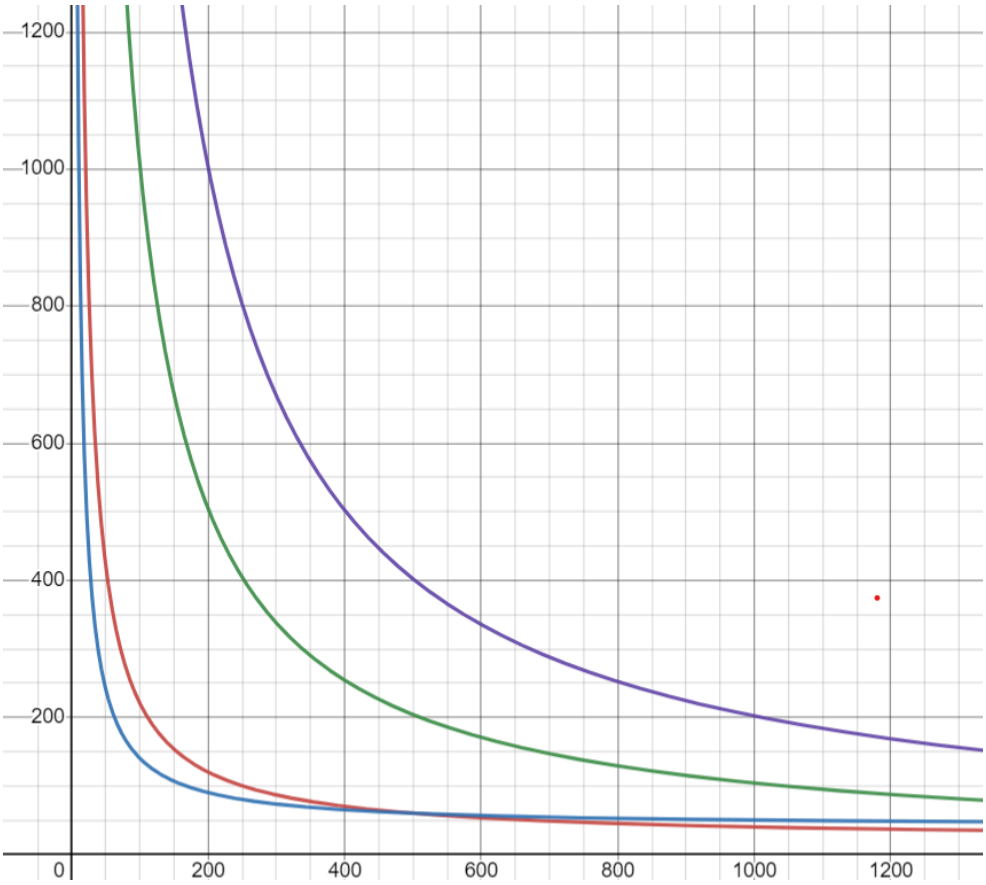



* όπου $f_1(x) =$ κόκκινο, $f_2(x) =$ μπλε, $f_3(x) =$ πράσινο και $f_4(x) =$ μωβ.

Κάθε τεχνολογία έχει μια περιοχή συμφέροντος κόστους, η οποία μπορεί να εντοπιστεί μέσω τις εξίσωσης των τεσσάρων παραπάνω συναρτήσεων:

- $f_1(x) = f_2(x) \leftrightarrow \frac{20.000}{x} + 20 = \frac{10.000}{x} + 40 \leftrightarrow \frac{10.000}{x} = 20 \leftrightarrow x = 500$
- $f_1(x) = f_3(x) \leftrightarrow \frac{20.000}{x} + 20 = \frac{100.000}{x} + 4 \leftrightarrow \frac{80.000}{x} = 16 \leftrightarrow x = 5.000$
- $f_1(x) = f_4(x) \leftrightarrow \frac{20.000}{x} + 20 = \frac{200.000}{x} + 2 \leftrightarrow \frac{180.000}{x} = 18 \leftrightarrow x = 10.000$
- $f_2(x) = f_3(x) \leftrightarrow \frac{10.000}{x} + 40 = \frac{100.000}{x} + 4 \leftrightarrow \frac{90.000}{x} = 36 \leftrightarrow x = 2.500$

- $f_2(x) = f_4(x) \leftrightarrow \frac{10.000}{x} + 40 = \frac{200.000}{x} + 2 \leftrightarrow \frac{190.000}{x} = 38 \leftrightarrow x = 5.000$
- $f_3(x) = f_4(x) \leftrightarrow \frac{100.000}{x} + 4 = \frac{200.000}{x} + 2 \leftrightarrow \frac{100.000}{x} = 2 \leftrightarrow x = 50.000$




Σύμφωνα με τα αποτελέσματα των εξισώσεων, καθώς και τη διαγραμματική απεικόνιση των συναρτήσεων, η περιοχές συμφέροντος κόστους για κάθε τεχνολογία είναι οι εξής:

1. $f_1$: $500 < x < 5.000$
2. $f_2$: $0 < x < 500$
3. $f_3$: $5.000 < x < 50.000$
4. $f_4$: $50.000 < x$

Θεωρούμε τη μεταβλητή κ ως το κόστος ανά τεμάχιο I.C. Προκειμένου η τεχνολογία FPGA να εξαλείψει την τεχνολογία I.C., αρκεί:

$$f_2(x) < f_1(x) \leftrightarrow \frac{10.000}{x} + 10 + \kappa < \frac{20.000}{x} + 20 \leftrightarrow \kappa < \frac{10.000}{x} + 10$$

$$\lim_{x \to \infty} \left(\frac{10.000}{x} + 10\right) = 10 \text{ άρα } \kappa < 10$$