NATIONAL TECHNICAL UNIVERSITY OF ATHENS SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

1η Σειρά Ασκήσεων

Συστήματα Μικροϋπολογιστών

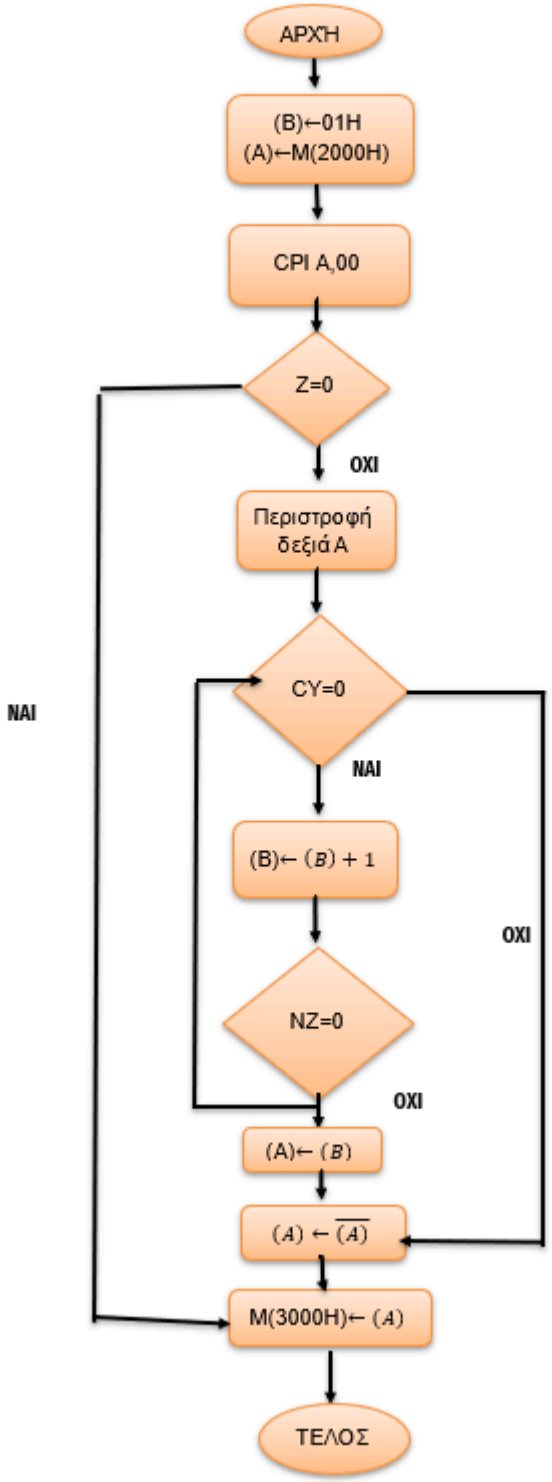
1η Άσκηση:

Αρχικά συμβουλευόμενοι τις οδηγίες της άσκησης ανατρέξαμε στον πίνακα 2 και προέκυψε η εξής αποκωδικοποίηση του κώδικα 16δικής μορφής:

```
        MVI B,01H
        LDA 2000H
        CPI 00H
        JZ LABEL_3
LABEL_1:
        RAR
        JC LABEL_2
        INR B
        JNZ LABEL_1
LABEL_2:
        MOV A,B
LABEL_3:
        CMA
        STA 3000H
        RST 1
END
```

```
0800   06      MVI B,01H
0801   01
0802   3A      LDA 2000H
0803   00
0804   20
0805   FE      CPI 00H
0806   00
0807   CA      JZ LABEL_3
0808   13
0809   08

LABEL_1:
080A   1F      RAR
080B   DA      JC LABEL_2
080C   12
080D   08
080E   04      INR B
080F   C2      JNZ LABEL_1
0810   0A
0811   08

LABEL_2:
0812   78      MOV A,B

LABEL_3:
0813   2F      CMA
0814   32      STA 3000H
0815   00
0816   30
0817   CF      RST 1
```

Πριν εξηγήσουμε τον κώδικα, γνωρίζοντας ότι οι εντολές LDA,STA και αλμάτων υπό συνθήκη είναι των 3 bytes και όλες οι υπόλοιπες που χρησιμοποιούνται στον κώδικα είναι του 1 byte εκτός της MVI που είναι δύο προκύπτει η “φωτογραφία” της αρχιτεκτονικής κατάστασης του συστήματος που περιέχει τις εντολές.Ο κώδικας αρχικοποιεί τον καταχωρητή B την τιμή 1 και ακολούθως φορτώνει στον συσσωρευτή το περιεχόμενο της μνήμης με διεύθυνση 2000H(πόρτα εισόδου). Εν συνεχεία, ο αριθμός που φορτώθηκε στον A συγκρίνεται με το 0. Εάν το αποτέλεσμα της σύγκρισης είναι 0 τότε μεταφερόμαστε σε κώδικα που βρίσκεται στην διεύθυνση 0813H(LABEL_1) της μνήμης.Μετά εκτελείται δεξιά περιστροφή του accumulator με συμμετοχή κρατουμένου. Πρακτικά, κάθε bit του καταχωρητή A και το MSB καταλήγει στο Carry και το κρατούμενο στο LSB. Ακολούθως, εάν προκύψει κρατούμενο μεταφερόμαστε την διεύθυνση 0812(LABEL_2). Αν όχι αυξάνουμε το περιεχόμενο του καταχωρητή B. Έπειτα, αν το αποτέλεσμα της προηγούμενης πράξης είναι μη μηδενικό μεταφερόμαστε στην 080AH(LABEL_3). Μεταφέρουμε την τιμή του B στον accumulator και συμπληρώνουμε αυτό το αποτέλεσμα. Τέλος, αποθηκεύουμε το αποτέλεσμα του accumulator στην διεύθυνση μνήμης της πόρτας εξόδου 30H. Για να εκτελεστεί η ρουτίνα, εφαρμόζουμε το RST 1 έτσι ώστε να μεταφερθεί ο έλεγχος και ο PC στην διεύθυνση 0800H, δηλαδή εκεί που είναι γραμμένος ο κώδικας της ρουτίνας. Πρακτικά, ο κώδικας είναι μια υπορουτίνα η οποία αλλάζει την τιμή του B σύμφωνα με την τιμή του περιεχομένου μνήμης 2000H (πόρτα εισόδου) και μετά το αποτέλεσμα της τελικής τιμής αποθηκεύεται στον συσσωρευτή και εν τέλει τοποθετείται στην πόρτα εξόδου 0030H. Τόσο οι ετικέτες όσο και το END έχουν χρησιμοποιηθεί για να μπορέσει ο compiler να τρέξει το πρόγραμμα και να γνωρίζει μέχρι που να κάνει assemble τον κώδικα. Το διάγραμμα ροής είναι αυτό που ακολουθεί:



Για να τρέχει συνέχεια το πρόγραμμα συνεχώς το μόνο που πρέπει να αλλάξουμε είναι ότι θα πρέπει αντί για την διακοπή RST_1 να έχουμε ένα jump στην αρχή του προγράμματος για αυτό και δημιουργούμε το label ΑΡΧΗ και ο κώδικας είναι ο εξής:

```
ARXH:
       MVI B,01H
       LDA 2000H
       CPI 00H
       JZ LABEL_3
LABEL_1:
       RAR
       JC LABEL_2
       INR B
       JNZ LABEL_1
LABEL_2:
       MOV A,B
LABEL_3:
       CMA
       STA 3000H
       JMP ARXH
END
```

Πρακτικά έχουμε έναν κώδικα ο οποίος αποθηκεύει στον καταχωρητή B την θέση από δεξιά προς τα αριστερά την πρώτη θέση εμφάνισης του 1 του αριθμου στην θύρα εισόδου και μετά μεσω του accumulator A το αποθηκεύουμε στην θύρα εισόδου 30H.

2η Άσκηση: Πρακτικά για την ορθή λειτουργία του προγράμματος πρέπει να διακρίνουμε περιπτώσεις ανάλογα με τις τιμές των δύο LSB:

- 01: Το led εκτελεί την λειτουργία μετακίνησης από την μία στην άλλη πλευρά αέναως, δηλαδή αρχικά πηγαίνει προς τα αριστερά και μόλις συναντήσει την τελευταία θέση (8) γυρνάει προς την αντίστροφη κατεύθυνση και μόλις βρει το led 1 πηγαίνει προς τα αριστερά και τουμπαλιν.
- 11: Το led σταματάει στην θέση όπου βρισκόταν όταν πατήσαμε και συνεχιζει αν τεθεί OFF το 2ο LSB ξεκινά με την κατεύθυνση που είχε πριν.
- 00: Το κύκλωμα εκτελεί κυκλική ολίσθηση του led
- 10: Σταματα το led στην θέση που ήταν και συνεχίζει με βάσει το αν είναι set το LSB

Παρακάτω έχουμε ένα screenshot του κώδικα με τις επεξηγήσεις του:

```
IN 10H
LXI B,01F4H ; BC has the value of 500 because so are the ms of delay we want
MVI A,FEH ; LED starting position(11111110 for negative logic)

PRINT_RESULT:
STA 3000H
MOV D,A ; D contains previous LED position

START:
       CALL DELB ; 0.5s delay
       LDA 2000H ; load the input port to the accumulator
       ANI 03H ; we only keep the last two digits
       CPI 01H ; we see if the values of them are 01 so we must perform the desired oper
       JZ SHIFT_LEFT ; we move to the move left label
RETURN:
       CPI 00H ; we check if the two last LSBs are 00 and if so
       JZ CYCLE_LEFT ; we should perform the cycle left
       MVI A,FEH ; turn on LED 0
       STA 3000H ; if not we simply set the LED 0 ON and
       JMP START ; jump to start

CYCLE_LEFT:
       MOV A,D ; A now contains previous LED position
       RLC ; we just perform a left shifting in order to perform the cycle
       JMP PRINT_RESULT ; we print the result

SHIFT_LEFT:
       MOV A,D ; the accumulator has now the previous position of the led
       CPI 7FH ; we check if the led has reached the leftmost led
       JZ SHIFT_RIGHT ; and if it does we should move right
       RLC ; else we perform a left shifting to the previous led in order to move one po
       JMP PRINT_RESULT ; we print the result in the output port

SHIFT_RIGHT:
       MOV A,D ; as always we have the previous led in the accumulator
       CPI FEH ; proportionally to the previous case we check if we are in the rightmost
       JZ SHIFT_LEFT ; if we do we should move to the opposite direction
       RRC ; if not we should moove the led one position to the right
       STA 3000H ; we save the content of the accumulator in the output port
       MOV D,A ; again the D has the previous led
       CALL DELB ; 0.5s delay
       LDA 2000H ; we load the input port to the accumulator
       ANI 03H ; nullifies all bits except the 2 LSBs
       CPI 01H ; we recompare the LSBs with 01H
       JZ SHIFT_RIGHT
       JMP RETURN
END
```

3η Άσκηση:

Για τον σκοπό της άσκησης διακρίναμε τρεις περιπτώσεις αναφορικά με το input:

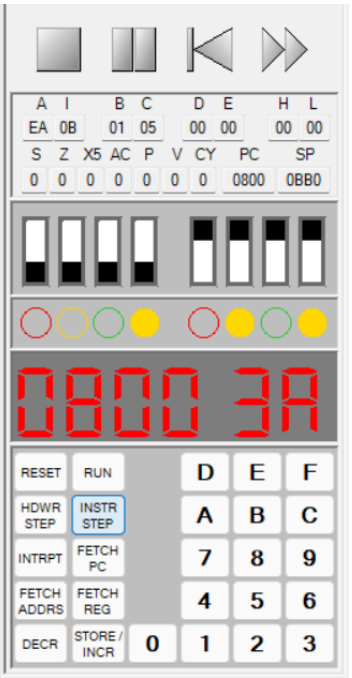
- input>199
- input>99 && input<199
- input<99

Για την πρώτη περίπτωση χρησιμοποιώντας την εντολή CPI και τις σημαίες που προκύπτουν, θέτουμε το περιεχόμενο του συσσωρευτή ίσο με 1111111 και το συμπληρώνουμε γιατι τα LED είναι αρνητικής λογικής. Γενικά, σε κάθε περίπτωση το αποτέλεσμα προκύπτει στον συσσωρευτή και συμπληρώνεται για να φανεί το σωστό αποτέλεσμα στο console και πάντα γυρνάμε στην αρχή του κώδικα έτσι ώστε να τρέχει συνέχεια. Ακολούθως, αν το input<199 έχουμε την επόμενη σύγκριση η οποία ελέγχει εάν input>99. Αν ισχύει αφαιρούμε από τον accumulator το αντίστοιχο 100 στο δεκαεξαδικό και εν συνεχεία τυπώνουμε στα LEDs. Αν δεν συμβαίνει τίποτα από τα παραπάνω έχουμε έναν αριθμό μικρότερο του 100 και έχουμε πρακτικά τον ίδιο κώδικα με την

σελίδα 84 του βιβλίου αλλά με μια διαφοροποίηση. Εδώ, δεν αποθηκεύουμε σε διαφορετικές θέσεις μνήμης τις δεκάδες και τις μονάδες, αλλά στην ίδια θύρα εξόδου. Για να γίνει τούτο, έχουμε ότι κρατάμε σε έναν καταχωρητή τις μονάδες και μετά στον accumulator τις δεκάδες και έτσι εκτελώντας 4 φορές αριστερή περιστροφή πηγαίνουν στα MSBs. Μετά, απλά προσθέτουμε τις μονάδες προσθέτοντας τον καταχωρητή που τις περιέχει στον A και τυπώνουμε όπως πριν. Έπεται ο κώδικας της άσκησης μαζί με snippets για την εκτέλεση αυτού για input 15, 107 και 203.

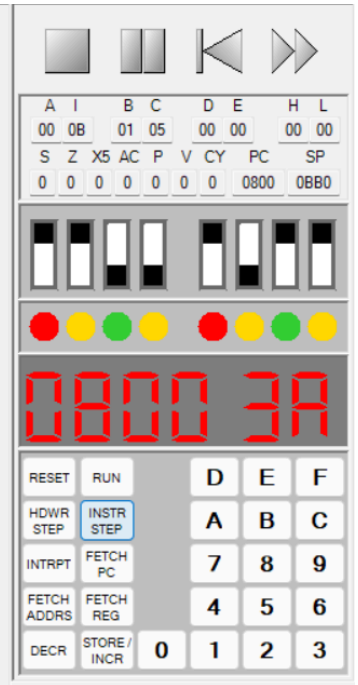
```
ARXH:
       LDA 2000H ;we load the input to the accumulator
       CPI C7H ; we compare it with 199 and if it larger than that
       JNC CASE_1 ; we jump into the first case handling
       CPI 63H ; if it is not then we check if it is greater than 99
       JNC CASE_2 ; if it is we jump in the second case
       MVI B,FFH ; else we have the third case where the number is <99
DECA:
       INR B ; the assembly code is the same with the problem in page 84
       SUI 0AH ; but the only difference is that we save the units
       JNC DECA ; and tens in the same register
       ADI 0AH ; to do so we save the units into reg C
       MOV C,A ; and the tens to Accumulator like done in the next command
       MOV A,B ; Then in order to have them in the MSBs we shift them
       RLC ; four times to the left
       RLC
       RLC
       RLC
       ADD C ; and then add the units
       CMA ; we have the complement of A beacuse the LEDS are of negative logic
       STA 3000H ; we save the result and then we jump into the ARXH
       JMP ARXH

CASE_1:
       MVI A,FFH ; we set all the LEDS and we complement them to be properly shown
       CMA
       STA 3000H
       JMP ARXH
CASE_2:
       SUI 64H ; we remove 100 from the final result
       CMA
       STA 3000H
       JMP ARXH
END
```

```
ARXH:
0800   3A     LDA 2000H
0801   00
0802   20
0803   FE     CPI C7H
0804   C7
0805   D2     JNC CASE_1
0806   25
0807   08
0808   FE     CPI 63H
0809   63
080A   D2     JNC CASE_2
080B   2E
080C   08
080D   06     MVI B,FFH
080E   FF

DECA:
080F   04     INR B
0810   D6     SUI 0AH
0811   0A
0812   D2     JNC DECA
0813   0F
0814   08
0815   C6     ADI 0AH
0816   0A
0817   4F     MOV C,A
0818   78     MOV A,B
0819   07     RLC
081A   07     RLC
081B   07     RLC
081C   07     RLC
081D   81     ADD C
081E   2F     CMA
081F   32     STA 3000H
0820   00
0821   30
0822   C3     JMP ARXH
0823   00
0824   08

CASE_1:
0825   3E     MVI A,FFH
0826   FF
0827   2F     CMA
0828   32     STA 3000H
0829   00
```



```
0800   3A     LDA 2000H
0801   00
0802   20
0803   FE     CPI C7H
0804   C7
0805   D2     JNC CASE_1
0806   25
0807   08
0808   FE     CPI 63H
0809   63
080A   D2     JNC CASE_2
080B   2E
080C   08
080D   06     MVI B,FFH
080E   FF

DECA:
080F   04     INR B
0810   D6     SUI 0AH
0811   0A
0812   D2     JNC DECA
0813   0F
0814   08
0815   C6     ADI 0AH
0816   0A
0817   4F     MOV C,A
0818   78     MOV A,B
0819   07     RLC
081A   07     RLC
081B   07     RLC
081C   07     RLC
081D   81     ADD C
081E   2F     CMA
081F   32     STA 3000H
0820   00
0821   30
0822   C3     JMP ARXH
0823   00
0824   08

CASE_1:
0825   3E     MVI A,FFH
0826   FF
0827   2F     CMA
0828   32     STA 3000H
0829   00
082A   30
```

```
0800   3A     LDA 2000H
0801   00
0802   20
0803   FE     CPI C7H
0804   C7
0805   D2     JNC CASE_1
0806   25
0807   08
0808   FE     CPI 63H
0809   63
080A   D2     JNC CASE_2
080B   2E
080C   08
080D   06     MVI B,FFH
080E   FF

DECA:
080F   04     INR B
0810   D6     SUI 0AH
0811   0A
0812   D2     JNC DECA
0813   0F
0814   08
0815   C6     ADI 0AH
0816   0A
0817   4F     MOV C,A
0818   78     MOV A,B
0819   07     RLC
081A   07     RLC
081B   07     RLC
081C   07     RLC
081D   81     ADD C
081E   2F     CMA
081F   32     STA 3000H
0820   00
0821   30
0822   C3     JMP ARXH
0823   00
0824   08

CASE_1:
0825   3E     MVI A,FFH
0826   FF
0827   2F     CMA
0828   32     STA 3000H
0829   00
082A   30
```

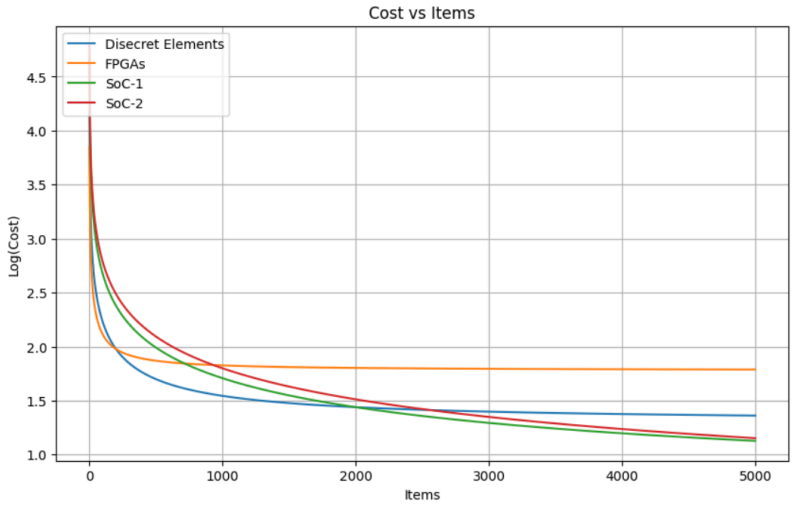


4η Άσκηση:

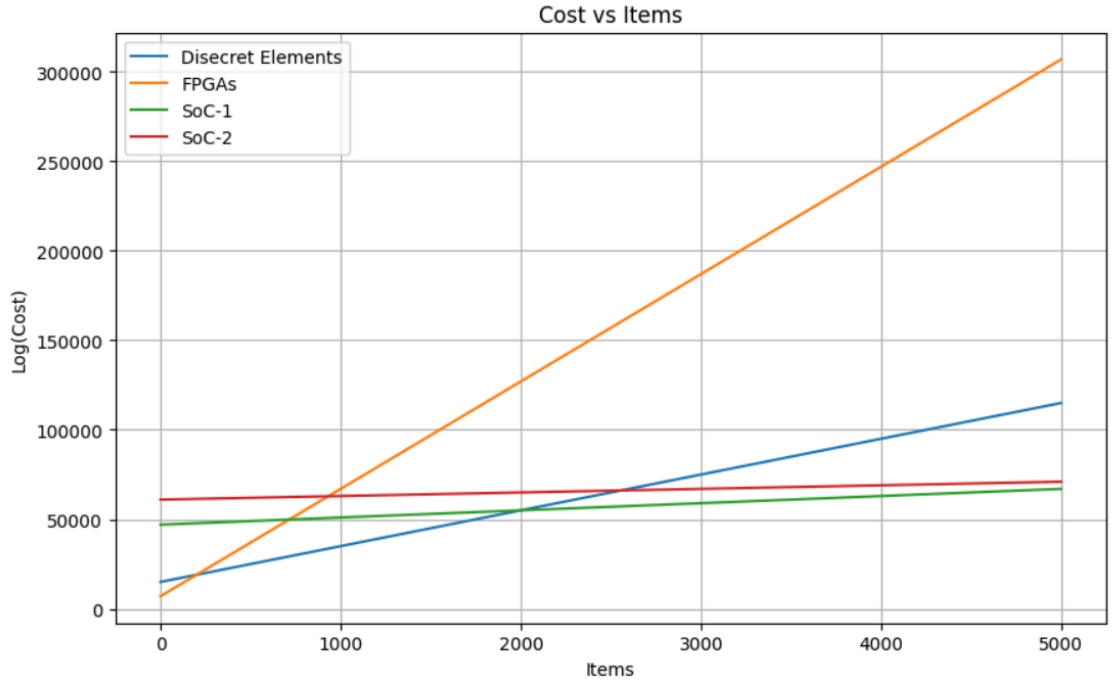
Για την παρούσα άσκηση παρουσιάζουμε την συνάρτηση κόστους για κάθε μέθοδο κατασκευής και προκύπτουν τα εξής:

- Διακριτά Στοιχεία: K(x) = 15000 + 20x, Cost_per_item = K(x)/x=15000/x +20
- FPGAs: K(x) = 7000 + 60x, Cost_per_item = K(x)/x=7000/x +60
- SoC-1:K(x) = 47000 + 4x, Cost_per_item = K(x)/x=47000/x +4
- Soc-2:K(x) = 61000 + 2x, Cost_per_item = K(x)/x=61000/x +2

Για την καμπύλη κάθε συνάρτησης έχουμε πρώτα το plot για τα κόστη ανά τεμάχιο σε log scale για να μπορέσουμε να διακρίνουμε καλύτερα τα σημεία τομής:



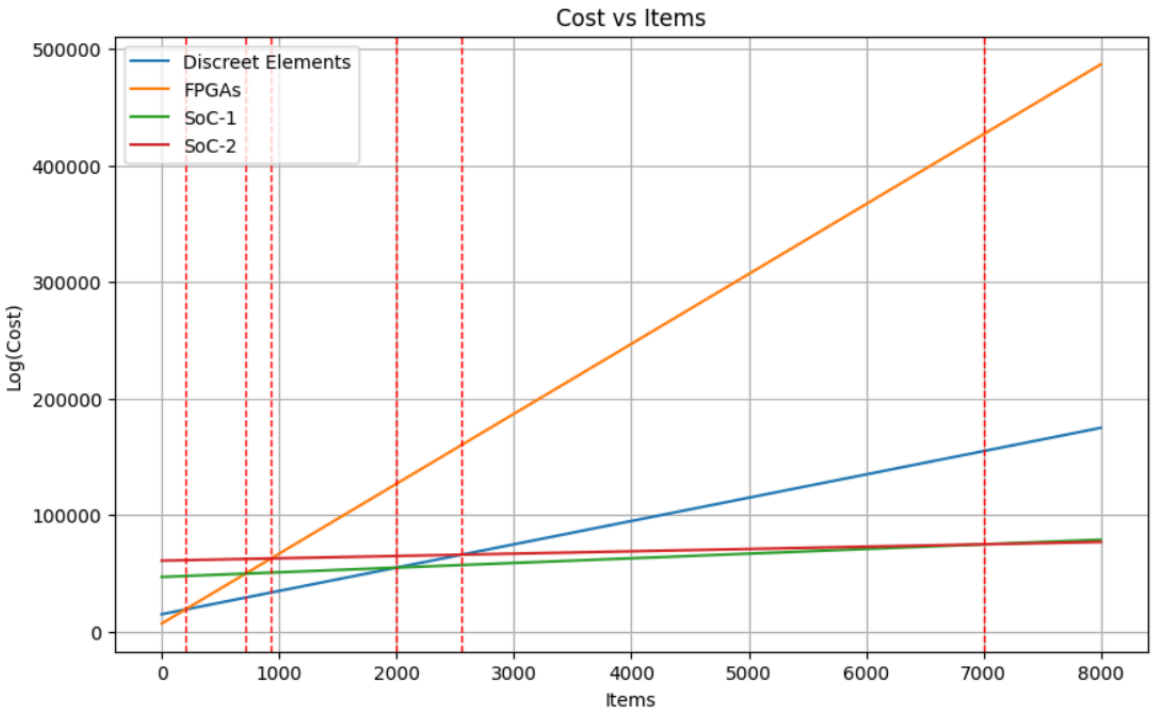
Ακολούθως, απεικονίσαμε τα διαγράμματα συνολικού κόστους για κάθε μέθοδο κατασκευής:



Οι περιοχές συμφέρουσας επιλογής μεθόδου καθορίζονται μέσω των σημείων τομής των τεσσάρων συναρτήσεων κόστους(θεωρούμε Ki(x) την αντίστοιχη συνάρτηση κόστους):

- K1(x)=K2(x): x=200
- K1(x)=K3(x): x=2000
- K1(x)=K4(x): x=2556(κατόπιν στρογγυλοποίησης)
- K2(x)=K3(x): x=714
- K2(x)=K4(x): x=931
- K3(x)=K4(x): x=7000

Για την οπτικοποίηση του αποτελέσματος χρησιμοποιήσαμε κατακόρυφες γραμμές στα αντίστοιχα σημεία τομής και παρατηρήσαμε ποιά είναι η καλύτερη συνάρτηση κόστους:

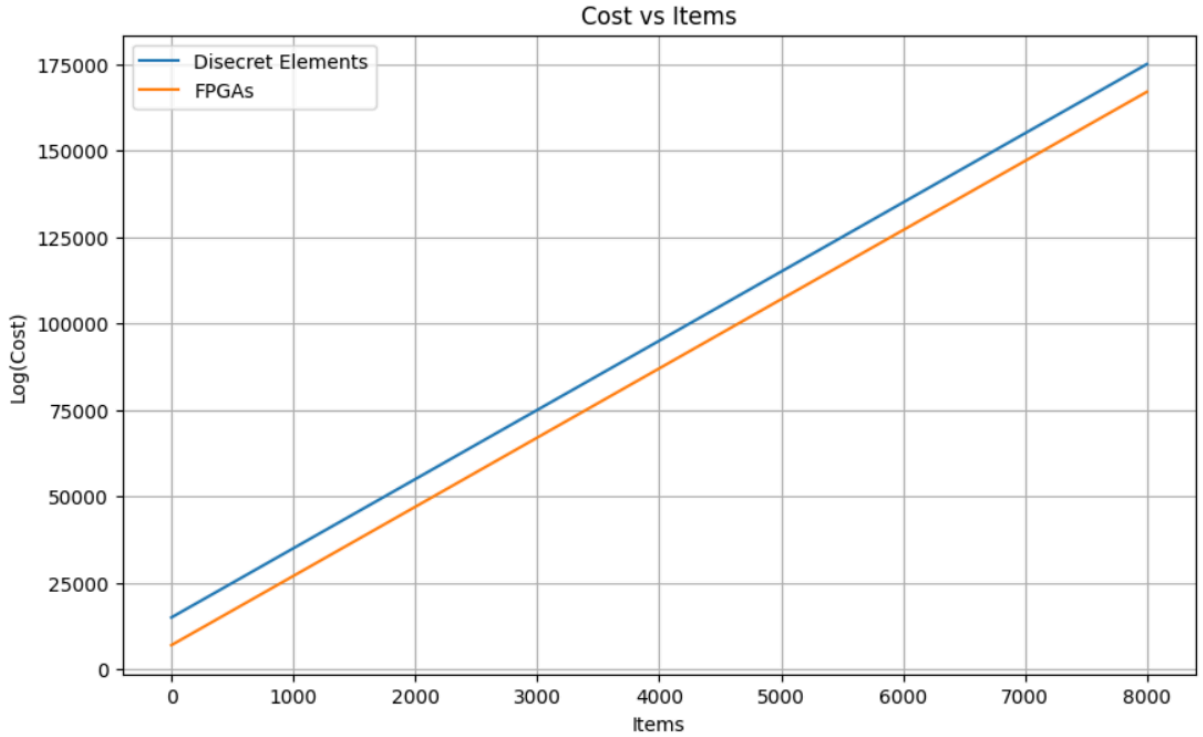


Τα αποτελέσματα που προκύπτουν είναι:

- 0-200 items: Discrete Elements
- 200-2000 items: FPGAs
- 2000-7000 items: SoC-1
- 7000-... items: SoC-2

Αυτό που εντοπίσαμε είναι ότι οι τεχνολογίες SoC είναι καλύτερες και οικονομικότερες για μεγάλες τιμές των τεμαχίων. Αναφορικά με το τελευταίο ζητούμενο, θέλουμε η νεα συνάρτηση κόστους του FPGA να είναι μικρότερη για όλες τις τιμές των τεμαχίων απο την συναρτηση κόστους των διακριτών στοιχείων, δηλαδή: $(a+10)x+7000<15000+20x \Rightarrow a<10+(8000/x)$. Καθώς το x αυξάνει το κλάσμα τείνει στο 0 άρα εν γένει η ελάχιστη τιμή του α=10 και αυτό μπορεί να διαπιστωθεί απο το παρακάτω figure στο οποίο βλέπουμε ότι πάντα τα FPGA

είναι κάτω απο τα διακριτά στοιχεία:



(ο κώδικας μεσω του οποίου παράχθηκαν τα παραπάνω είναι συνημμένος στην αναφορά μας!)