ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ

1η ομάδα ασκήσεων

Χιώτης Χαράλαμπος (03122142)
Τσουρούφλη Ζωή (03122062)

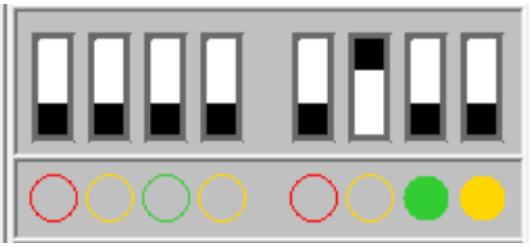
ΑΣΚΗΣΕΙΣ ΠΡΟΣΟΜΟΙΩΣΗΣ

1η ΑΣΚΗΣΗ

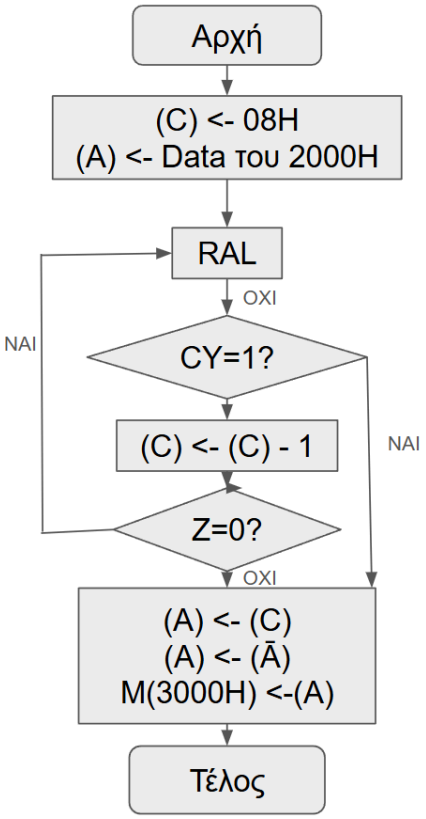
0800	OE	MVI C,08H	φορτώνει την τιμή 08H στον register C
0801	08		
0802	3A	LDA 2000H	φορτώνει στον A από το περιεχόμενο της μνήμης που βρίσκεται στην θέση 2000H
0803	00		
0804	20		
0805	17	RAL	κυκλική αριστερά μετατόπιση (μέσω κρατούμενου) του περιεχομένου του A
0806	DA	JC 080D	αν το carry flag είναι CY=1 κάνει jump στη διεύθυνση 080D
0807	0D		
0808	08		
0808	OD	DCR C	μείωση της τιμής του καταχωρητή C κατά 1
080A	C2	JNZ 0805	στην περίπτωση μη μηδενικού αποτελέσματος (Z=0) κάνει jump στη διεύθυνση 0805
080B	05		
080C	08		
080D	79	MOV A,C	μεταφορά δεδομένων από τον καταχωρητή C στον A
080E	2F	CMA	συμπλήρωμα των bits του περιεχομένου του καταχωρητή A
080F	32	STA 3000H	αποθήκευση του περιεχομένου του καταχωρητή A προς τη θέση μνήμης 3000H
0810	00		
0811	30		
0812	CF	RST 1	software interrupt

Τι κάνει το πρόγραμμα αυτό πρακτικά;
Φορτώνει στον A καταχωρητή ένα περιεχόμενο και βλέπει αν το MSB στις πρώτες 8 μετατοπίσεις γίνεται 1. Αν κάποια στιγμή το MSB είναι 1, ο A κρατάει πόσες μετατοπίσεις (εκ των 8) απέμειναν, τις αντιστρέφει και αποθηκεύει το αποτέλεσμα στη μνήμη. Αν στις 8 μετατοπίσεις το MSB δεν γίνει ποτέ 1 αποθηκεύει αντίστοιχα τις υπόλοιπες ανεστραμμένες μετατοπίσεις (8) και τερματίζει το πρόγραμμα.

Τι περιμένω να δω στον simulator;
Θυμόμαστε ότι η θέση 2000H είναι το input από τα dip switches επομένως μπορώ να ελέγχω από εκεί το input που βάζω. Για παράδειγμα με input στα dip switches 4, το πρόγραμμα μου εκτελεί 5 περιστροφές μέχρι να εντοπίσει κρατούμενο=1, υπολογίζει τις μετατοπίσεις που απέμειναν 8-5=3 και μου πετάει αυτό το output στα λαμπάκια.



Διάγραμμα Ροής



Τι αλλαγές πρέπει να γίνουν για να έχουμε συνεχή λειτουργία του παραπάνω προγράμματος, δηλαδή να επαναλαμβάνεται χωρίς τέλος;

Προκειμένου το πρόγραμμα μας να επαναλαμβάνεται μέχρι τέλους θα πρέπει απλώς να βάλουμε πριν την RST 1 και την end την εντολή jump στη main, με main να είναι ετικέτα στην αρχή του προγράμματος. Τώρα έχουμε τη δυνατότητα να δίνουμε διαφορετικά input στα dip

switches και να παιρνουμε κατευθειαν το αντιστοιχο αποτέλεσμα, χωρις να πρεπει να σταματησουμε το προγραμμα, να αλλαξουμε το ινπουτ και να το ξανατρεξουμε .

2η Άσκηση

MAIN:

Φορτώνουμε το πρώτο λαμπάκι στην αρχή του προγράμματος και το αποθηκεύω στον B. Φορτωνω το περιεχόμενο των dip switches στον καταχωρητή A. Με λογική πράξη ελεγχω το MSB και αν ειναι 0 (OFF) παω στην LEFT, η οποία περιέχει την υλοποίηση για να πηγαίνουν τα λαμπάκια προς τα αριστερά. Αν δεν ειναι 0 (ON) συνεχιζει κανονικά στην RIGHT και τα λαμπάκια θα πηγαίνουν προς τα δεξια.

```
MVI A,01H ; φορτωνουμε το πρωτο λαμπακι

MAIN:
MVI B,10H ; για την καθυστερηση της DELB
CALL DELB
MOV B,A
LDA 2000H ; φερνουμε ινπουτ

ANI 80H ; Ελεγχουμε MSB
JZ LEFT ; αν το MSB ειναι 0 (δηλαδη OFF) πας αριστερα

RIGHT:
```

Η LEFT και η RIGHT έχουν πάνω κάτω την ιδια υλοποίηση.

Ας πάρουμε την **RIGHT:**.

```
RIGHT:
CALL CHECK_LSB ; παμε να τσεκαρουμε το LSB
MOV A,B ; φερνουμε σωστη τιμη
RRC ; πας δεξια
CALL OUTPUT ; καλεις την αουτπουτ
JMP MAIN ; ξανα
```

Πρώτα καλούμε τη ρουτίνα CHECK_LSB η οποία ελέγχει το LSB των dip switches. Αν ειναι ON γυρνάμε στην RIGHT , φέρνουμε τη σωστή τιμή στον A και κανουμε RRC , έτσι πανε τα bits του A μια θέση δεξιά. Έπειτα καλούμε την ρουτίνα OUTPUT. Τέλος, κάνουμε JMP MAIN για να ξανατρέξει το πρόγραμμα. Γιαυτό και στην αρχή της main έχουμε DELB (ορίσαμε την καθυστέρηση στον B) , διαφορετικά τα λαμπάκια θα άναβαν πολύ γρήγορα και δε θα βλέπαμε το επιθυμητό αποτέλεσμα.

Λεπτομέρεια: στην LEFT βάλαμε το RLC μετά το call output ώστε να ανάψει το πρώτο λαμπάκι (πρώτη εντολή), διαφορετικά θα μετακινούνταν πριν ανάψει το πρώτο και θα άρχιζε από το δεύτερο.

CHECK_LSB:

```
CHECK_LSB:
LDA 2000H
ANI 01H ; αν LSB 1 παιρνω 1 αρα z=0
JNZ CONTINUE ; ON=LSB 1
PUSH B
MVI B,10H
CALL DELB ; για να προλαβει το νεο ινπουτ και να συνε
POP B ; για να εχουμε και εδω την ιδια καθυστερηση (οι
CALL CHECK_LSB ; ξαναελεγχε μεχρι να αλλαξει
CONTINUE:
RET
```

Φορτώνουμε πάλι το περιεχόμενο των switches, με λογική πράξη κρατάμε το LSB και αν ειναι 1 (ON) συνεχίζουμε και κανει return. Αν είναι όμως 0 (OFF) πρέπει να σταματήσει εκεί που βρίσκεται. Γ'ιαυτό ξανακαλούμε τη ρουτίνα και μόνο όταν αλλάξει το LSB θα βγει από τη λούπα και θα συνεχίσει. Έχουμε βάλει και DELB γιατί προσέξαμε ότι κόλλαγε όταν αλλάζαμε το input, καθώς δεν προλάβαινε να πάρει τη νέα τιμή. Το push και το pop είναι επειδή δεν πρέπει να αλλάξει η τιμή του B, στον οποίο έχουμε σώσει το αρχικό περιεχόμενο του A, δηλαδή ποιο λαμπάκι ανάβει.

OUTPUT:

```
OUTPUT:
MOV B,A ; κραταμε κανονικη τιμη
CMA ; συμπληρωμα
STA 3000H ; εξοδος
MOV A,B ; επαναφορα
RET
```

Εκεί, αποθηκεύουμε τον A, καθώς μετά κάνουμε συμπλήρωμα του A (αντίστροφη λογική απεικόνισης) και το στέλνουμε στη διευθυνση που ειναι τα λαμπάκια. Επαναφέρουμε τον A και κάνουμε return.

3η Άσκηση

MAIN:

```
MAIN:
IN 10H
LDA 2000H ; φορτωνω περιεχομενο
CPI C8H
JNC LEFT
CPI 64H ; συγκριση με 100
JNC RIGHT
MVI B,FFH ; B=-1
```

Φορτώνουμε το περιεχόμενο των switches, δηλαδή τον αριθμό που θα μετατρέψουμε σε δεκαδικό και έπειτα το συγκρίνουμε με το 200. Αν είναι μεγαλύτερος πάει στον LEFT, όπου αναβοσβήνουν ολα τα MSB λαμπάκια. Διαφορετικά συνεχίζει στην επόμενη σύγκριση με το 100. Αν είναι μεγαλύτερο του 100 (εχουμε εξασφαλίσει n<200) τότε πάει στο RIGHT, όπου αναβοσβήνουν ολα τα LSB. Διαφορετικά, αν είναι δηλαδή από 0 έως και 99, πάει στην DECA και υπολογίζει δεκάδες και μονάδες.

LEFT:

```
LEFT:
LDA 2000H ; ξανατσεκαρω ινπουτ
CPI C8H
JC MAIN
MVI A,F0H
CMA
STA 3000H
MVI B,10H
CALL DELB
MVI A,00H
CMA
STA 3000H
MVI B,10H
CALL DELB
JMP LEFT
```

φορτώνουμε το input και το συγκρίνουμε με το 200 ώστε αν αλλάξει να βγούμε από τη λούπα. Φέρνουμε την τιμή F0H κάνουμε συμπλήρωμα και το στέλνουμε στην έξοδο. Έτσι άναψαν τα MSB. Κάνουμε DELB για να δούμε τα λαμπάκι, μηδενίζουμε τον A, συμπλήρωμα και το στέλνουμε στην έξοδο. Έτσι έσβησαν τα MSB. Και πάλι LEFT.

RIGHT:

```
RIGHT:
LDA 2000H ; ξανατσεκαρω ινπουτ
CPI 64H ; αν ειναι μικροτερο του 100 βγες
JC MAIN
CPI C8H ; αν ειναι μεγαλυτερο του 200 βγες
JNC MAIN
MVI A,0FH ; αναβω LSB
CMA
STA 3000H
MVI B,10H
CALL DELB ; delay
MVI A,00H ; σβηνω
CMA
STA 3000H
MVI B,10H
CALL DELB
JMP RIGHT
```

Ίδια υλοποίηση, μόνο που εδώ θέλει δύο συγκρίσεις. Μια για αν είναι μικρότερο του 100 και μια για αν είναι μεγαλύτερο του 200. Αν δεν είναι τίποτα απο τα δυο, τότε μένουμε στην RIGHT.

DECA:

```
DECA:
INR B
SUI 0AH ; αφαιρω 10
JNC DECA ; αν θετικο ξανα
ADI 0AH ; επαναφορα του A αν αρνητικο
STA 0900H ; μοναδες
MOV A,B
STA 0901H ; δεκαδες
CALL OUTPUT
JMP MAIN
```

Όσον αφορά τον τρόπο υπολογισμού των δεκάδων και τον μονάδων προφανώς βασιζόμαστε στο παράδειγμα 4. Έπειτα, αποθηκεύουμε τις μονάδες και τις δεκάδες σε θέσεις μνήμης που διαθέτει ο simulator. Μετά καλούμε την OUTPUT και μετά την MAIN για να έχουμε συνεχή λειτουργία.

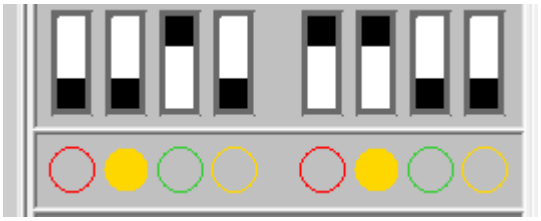
OUTPUT:

```
OUTPUT:
LDA 0901H     ; Φόρτωσε τις δεκάδες στον A
RLC           ; Περιστροφή αριστερά 1 bit
RLC           ; (x4 φορές) για να πάνε οι δεκάδες στα MSB
RLC
RLC

MOV B,A      ; Βάλε το αποτέλεσμα στον B (MSB = δεκάδες)

LDA 0900H     ; Φόρτωσε τις μονάδες στον A
ANI 0FH       ; Καθάρισε τα 4 MSB, κράτα μόνο τα 4 LSB (0-9)

ORA B         ; Συνδύασε A και B: A = A OR B
CMA
STA 3000H

RET
```

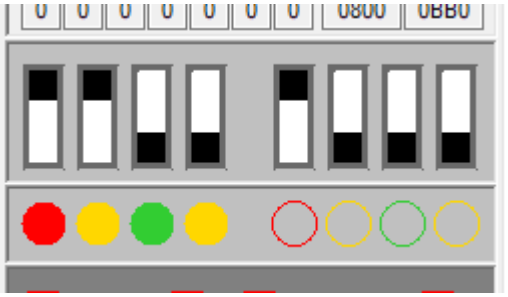
Φέρνουμε τις δεκάδες, κάνουμε 4 RLC , αφού θέλουμε να απεικονιστούν στα MSB και αποθηκεύουμε στον B. Φέρνουμε τις μονάδες, κρατάμε LSB και κάνουμε OR A με τον B. ‘Ετσι, αποθηκεύουμε στον A 8 bits, όπου τα 4 πρώτα (απο τα αριστερα) είναι οι δεκάδες και τα 4 τελευταία οι μονάδες.

Σενάρια:

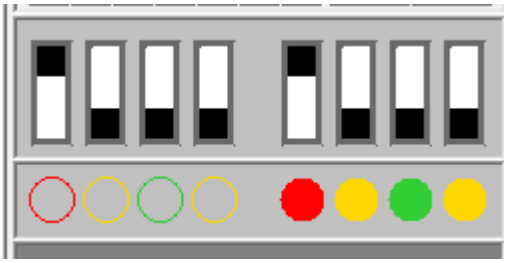
1. Αριθμός 44. Πράγματι, ανάβει στα MSB το 4 και στα LSB επίσης.



2. Αριθμός 200. Πράγματι αναβοσβήνουν τα MSB



3. Αριθμός 136. Πράγματι αναβοσβήνουν τα LSB.

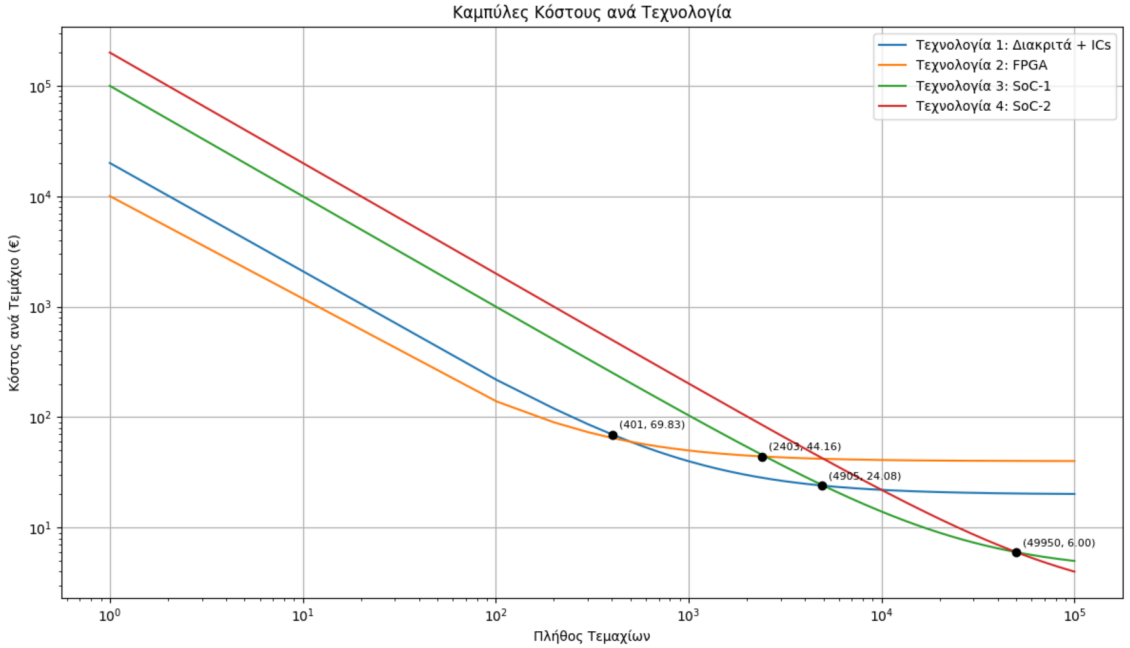


4η Άσκηση

Υπολογίζουμε αρχικά τις σχέσεις κόστους ανά τεμάχιο x, για κάθε μία από τις τεχνολογίες.
Ο γενικός τύπος που θα ακολουθήσουμε είναι ο:
Κόστος= Αρχικό + (Κόστος-ICs + Κόστος-κατασκευής)×Πλήθος τεμαχίων

c1=20.000+10*x+10*x
c2=10.000+30*x+10*x
c3=100.000+2*x+2*x
c4=200.000+1*x+1*x

Τις εμφανίζουμε τώρα αυτές τις σχέσεις σε ένα κοινό διάγραμμα:



Αναζητάμε τώρα τις περιοχές πλήθους τεμαχίων που είναι οικονομικότερες για την κάθε τεχνολογία:

- Η τεχνολογία 2 παρατηρούμε ότι μας συμφέρει μέχρι και τα 401 τεμάχια με μέγιστο κόστος ανά τεμάχιο στα 69.83 ευρώ.
- Η τεχνολογία 1 μας συμφέρει για παραγωγή από 401 τεμάχια έως και 4905
- Η τεχνολογία 3 μας συμφέρει για παραγωγή από 4905 έως και 49950 τεμάχια
- Ενώ η τεχνολογία 4 είναι οικονομικότερη για περισσότερα από 49950 τεμάχια άρα για πολύ μεγάλες παραγωγές.

Τα συμπεράσματα αυτά προκύπτουν απευθείας από τις γραφικές παραστάσεις που παρατίθονται από πάνω (plots προέκυψαν από python σε jupiter notebook).

Τέλος, θέλουμε να διερευνήσουμε για ποια τιμή κόστους ανά τεμάχιο των I.C. στην τεχνολογία των FPGAs (αντί των 30€) θα μπορούσε να εξαφανιστεί η επιλογή της 1ης τεχνολογίας.
Άρα ψάχνουμε τη μέγιστη τιμή ανά τεμάχιο των I.C., τέτοια ώστε να εξαλείφει το πλεονέκτημα κόστους της τεχνολογίας 1 (το οποίο βρίσκεται έως τα 401 τεμάχια).

Θέλουμε δηλαδή c2’<c1 δηλαδή 10.000+(νέο I.C. κόστος)*x+10*x<20.000+10*x+10*x και άρα προκύπτει νέο I.C. κόστος<10.