ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ - 6ο ΕΞΑΜΗΝΟ
ΑΝΑΦΟΡΑ 1ης ΟΜΑΔΑΣ ΑΣΚΗΣΕΩΝ

ΓΕΩΡΓΑΚΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ 03120827
ΓΕΩΡΓΙΑΔΗ ΔΑΦΝΗ 03120189

ΑΣΚΗΣΕΙΣ ΠΡΟΣΟΜΟΙΩΣΗΣ

Disclaimer: ο προσομοιωτής 8085 τρέχει σε Windows 7 μέσα σε virtual box.

ΑΣΚΗΣΗ 1

Το πρόγραμμα που δίνεται σε γλώσσα μηχανής αποκωδικοποιείται, με βάση τον πίνακα 2 του παραρτήματος 2 των σημειώσεων για το mLAB ως εξής:

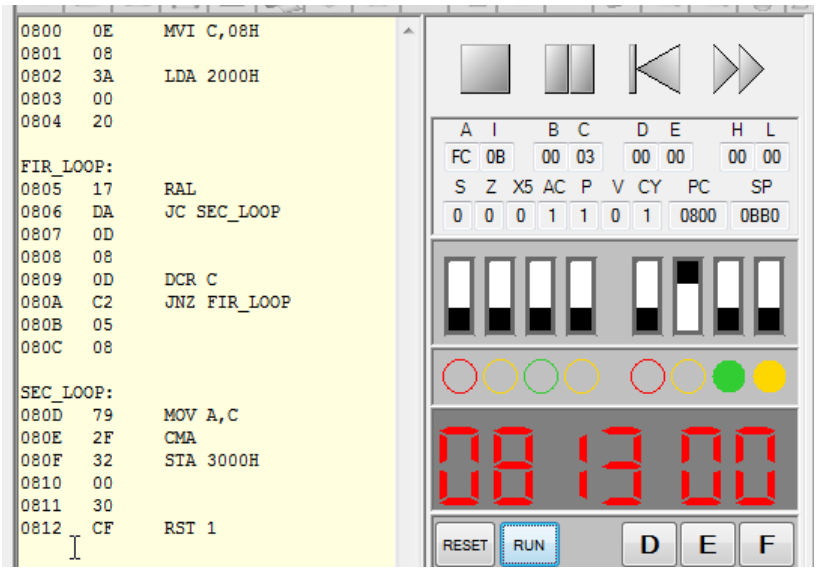
```
 1 MVI C,08H
 2 LDA 2000H
 3 RAL
 4 JC 080DH
 5 DCR C
 6 JNZ 0805H
 7 MOV A,C
 8 CMA
 9 STA 3000H
10 RST 1
```

```
0800   0E     MVI C,08H
0801   08
0802   3A     LDA 2000H
0803   00
0804   20

FIR_LOOP:
0805   17     RAL
0806   DA     JC SEC_LOOP
0807   0D
0808   08
0809   0D     DCR C
080A   C2     JNZ FIR_LOOP
080B   05
080C   08

SEC_LOOP:
080D   79     MOV A,C
080E   2F     CMA
080F   32     STA 3000H
0810   00
0811   30
0812   CF     RST 1
```

Για να δουλέψει σωστά το πρόγραμμα χρειαζόμαστε 2 labels στα 080DH και 0805H.

Η λειτουργία του προγράμματος αυτού είναι η εξής:
Εμφανίζει σε δυαδική μορφή στις λυχνίες εξόδου, τον αριθμό που αντιστοιχεί στην θέση του αριστερότερου dip switch που είναι ON.

Πράγματι:



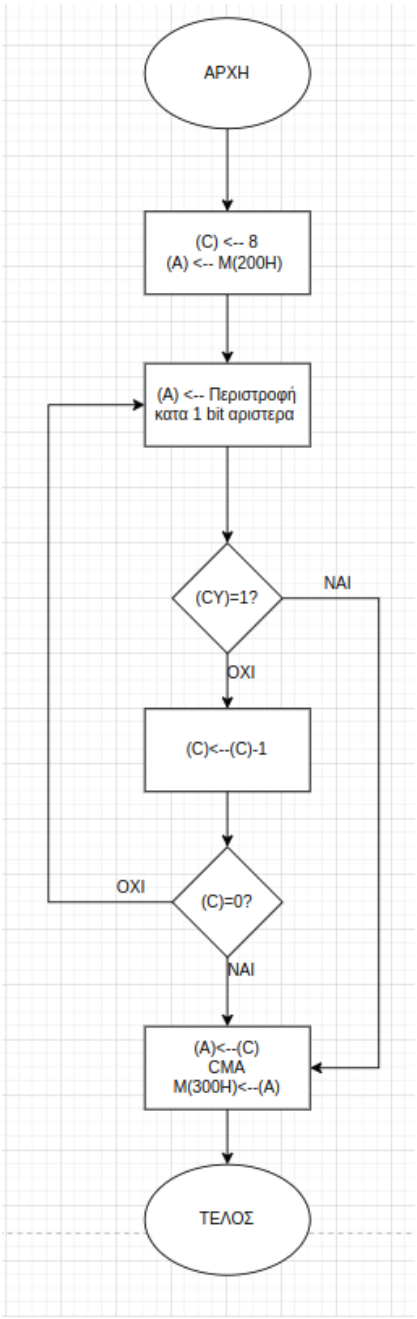
Σκοπός είναι να έχουμε συνεχή λειτουργία του παραπάνω προγράμματος, για να το πετύχουμε αυτό θα πρέπει στο τέλος του προγράμματος αντί για RST 1, να έχουμε ένα jump που να ξαναπηγαίνει στην αρχή του προγράμματος.

Το πρόγραμμα με το οποίο καταλήγουμε είναι το εξής:

```
START:
        MVI C,08H
        LDA 2000H
FIR_LOOP:
        RAL             ; αριστερη περιστροφη
        JC SEC_LOOP     ; jump αν υπαρχει κρατουμενο
        DCR C
        JNZ FIR_LOOP    ; αν c!=0 ξανα το loop
SEC_LOOP:
        MOV A,C
        CMA
        STA 3000H
        JMP START       ; Το προγραμμα τρεχει συνεχως

END
```

ΠΡΟΣΟΧΗ:
Για να τρέξει το πρόγραμμα αφού το κάνουμε assemble, πατάμε RUN. Το πρόγραμμα θα σταματήσει μόνο του. Πρέπει να πατήσουμε το πλήκτρο FETCH PC και έπειτα ξανά το RUN.

Τέλος δίνεται το διάγραμμα ροής του προγράμματος:



!!Στο παραδοτέο zip έχουμε συμπεριλάβει ένα βίντεο στο οποίο ελέγχουμε την ορθότητα του προγράμματος. !!
Επεξήγηση βίντεο:
Στην αρχή ανοίγουμε το δεύτερο dip switch, με αποτέλεσμα στα LEDs της εξόδου ορθώς να εμφανιστεί η δυαδική αναπαράσταση του αριθμού «2» (00000010). Έπειτα ανοίγουμε το τρίτο dip switch παράλληλα και παρατηρούμε ότι τα LEDs της εξόδου εμφανίζουν δυαδικά τον αριθμό «3» (00000011) που είναι το αριστερότερο dip switch που είναι ON. Τέλος ανοίγοντας το πέμπτο dip switch αγνοούνται όλα τα προηγούμενα και εμφανίζεται στην έξοδο δυαδικά ο αριθμός «5» (00000101).

ΑΣΚΗΣΗ 2

Το πρόγραμμα που ζητείται σε Assembly είναι το εξής:

```
        IN 10H
        LXI B,01F4H     ; Αποθηκευση στο B πόσο καθυστέρηση θέλουμε (0.5s = 500ms = 1F4H)
        MVI E,01H       ; Αρχικό LED το LSB

START:
        LDA 2000H
        MOV D,A         ; Στον D(βοηθητικος) η θέση μνήμης της θύρας εισόδου dip switch
        RRC             ; Ολίσθηση δεξιά ωστε να δούμε εαν CY=1 (CY=A0))
        JC START        ; Αν ναι τοτε το LSB ειναι OFF αρα ξαναξεκιναω μεχρι τελικα να γινει ON
        CALL DELB       ; Καθυστέρηση 0.5s
        MOV A,D         ; (A)=2000H
        RLC             ; Ολίσθηση αριστερά ωστε να δούμε αν CY=1 (CY=A7)
        JC RIGHT_ROT    ; Αν ναι τοτε κίνηση των LED δεξιά, αν οχι τοτε αριστερα

LEFT_ROT:                       ; Αριστερη Κίνηση LED
        MOV A,E         ; (A) = θεση αρχικου LED
        CMA             ; Συμπλήρωμα του A
        STA 3000H       ; Αποθήκευση του A (οπου δειχνει ποιο LED πρεπει να αναψει)
        CMA
        RRC
        MOV E,A         ; (A) = θέση επόμενου LED
        JMP START       ; Πισω στην αρχή για έλεγχο

RIGHT_ROT:                      ; Δεξιά Κίνηση LED
        MOV A,E         ; Ιδιο σκεπτικό με αριστερή κίνηση αλλα με RRC αντι για RLC
        CMA
        STA 3000H
        CMA
        RLC
        MOV E,A
        JMP START

        END
```

ΠΡΟΣΟΧΗ:
Για να τρέξει το πρόγραμμα αφού το κάνουμε assemble, πατάμε RUN. Το πρόγραμμα θα σταματήσει μόνο του. Πρέπει να πατήσουμε το πλήκτρο FETCH UP και έπειτα ξανά το RUN.

Παρατηρήσεις:

1. !! Στο παραδοτέο zip έχουμε συμπεριλάβει ένα βίντεο στο οποίο ελέγχουμε την ορθότητα του προγράμματος. !!
   Επεξήγηση βίντεο:
   Στο βίντεο φαίνεται ότι έχοντας το LSB σε θέση ON, όπου στην δική μας υλοποίηση να είναι κάτω, και το MSB σε θέση OFF, όπου στην δική μας υλοποίηση να είναι πάνω, παρατηρούμε ένα αναμμένο LED να κινείται αριστερά και να συνεχίζει να κινείται κυκλικά (0123456701…κλπ). Στην συνέχεια ανοίγουμε το MSB, και βλέπουμε το LED να κινείται δεξιόστροφα από την θέση που έμεινε. Τέλος κλείνοντας το LSB το LED σταθεροποιείται αναμμένο στην θέση που είναι.
2. Η καθυστέρηση γίνεται με την εντολή CALL DELB, κάθε φορά που ξανακάνω τους απαραίτητους ελέγχους , εφόσον έχω φορτώσει στην αρχή στο B το πόσο χρόνο καθυστέρηση θέλουμε.
3. Πριν την αποθήκευση του A στην θέση μνήμης 3000H (θύρα εξόδου των LEDs) δημιουργώ το συμπλήρωμα του A και αποθηκεύω αυτό, καθώς στα LEDs χρησιμοποιείται αντίστροφη λογική
4. Στην περίπτωση της δεξιάς κίνησης, όπου το MSB των dip switch είναι άσσος, για τον υπολογισμό της επόμενης θέσης του ανοιχτού LED χρειάζεται η εντολή RLC(η

οποία κάνει δεξιά ολίσθηση κατά 1), με αυτό τον τρόπο ολισθαίνουν προς την σωστή κατεύθυνση. Η παρατήρηση αυτή είναι συμπληρωματική με την από πάνω.

5. Διαφορετικός τρόπος υλοποίησης είναι να ορίσουμε αρχική θέση των LEDs την 11111110 και να χρησιμοποιούμε κανονικά τις εντολές RLC και RRC χωρίς να υπολογίζουμε τα συμπληρώματα. Ωστόσο σε αυτήν την περίπτωση η διάταξη των διακοπτών είναι αντεστραμμένη (δηλαδή πάνω ON και κάτω OFF) Στην δική μας υλοποίηση είναι πάνω OFF και κάτω ON. [OFF:το μαύρο πλαίσιο είναι πάνω]

ΑΣΚΗΣΗ 3

Το πρόγραμμα σε Assembly είναι το εξής:

```
        IN 10H
        LXI B,0320H     ;Καθυστέρηση 800ms
START:
        LDA 2000H
        CPI 63H         ;ελεγχος αν ειναι μεγαλύτερο του 99
        JNC GREATER_THAN_99
        MVI D,FFH       ; Το συμπληρωμα ως προς 2 ειναι το -1

DECA:
        INR D
        SUI 0AH
        JNC DECA        ; Ελέγχουμε αν εχουμε αρνητικό αριθμό
        ADI 0AH
        MOV E,A         ; Κρατάμε τις μονάδες
        MOV A,D         ; Παιρνάω τις δεκαδες στο A
        RLC             ; 4 Περιστροφές για να μεταφέρουμε τις δεκάδες απο τα
        RLC             ;       LSB στα MSB
        RLC
        RLC
        ADD E           ; Προσθεση στις δεκάδες τις μονάδες
        CMA
        STA 3000H       ; Αποθηκευση στην διευθυνση που αντιστοιχει στις θύρες εξόδου LEDs
        JMP START

GREATER_THAN_99:
        CPI C7H
        JNC GREATER_THAN_199    ; Ελεγχος αν ειναι μεγαλύτερο απο 199
        MVI A,F0H               ; Αναβοσβηνουν τα LSB LEDs
        STA 3000H               ; με καθυστερηση 800 ms
        CALL DELB
        MVI A,FFH
        STA 3000H
        CALL DELB
        JMP START

GREATER_THAN_199:
        MVI A,0FH               ; Αναβοσβηνουν τα MSB LEDs
        STA 3000H               ; με καθυστερηση 800 ms
        CALL DELB
        MVI A,FFH
        STA 3000H
        CALL DELB
        JMP START

        END
```

ΠΡΟΣΟΧΗ:
Για να τρέξει το πρόγραμμα αφού το κάνουμε assemble, πατάμε RUN. Το πρόγραμμα θα σταματήσει μόνο του. Πρέπει να πατήσουμε το πλήκτρο FETCH UP και έπειτα ξανά το RUN.

Παρατηρήσεις:

1. !! Στο παραδοτέο zip έχουμε συμπεριλάβει ένα βίντεο στο οποίο ελέγχουμε την ορθότητα του προγράμματος. !!
   Επεξήγηση βίντεο:
   Στο βίντεο φαίνεται ότι στην αρχή ανοίγουμε το τέταρτο dip switch που αντιστοιχεί σε δυαδική μορφή στον αριθμό 8_{10}, αφού έχοντας ON μόνο το τέταρτο dip switch, δηλαδή «1» και όλα τα άλλα «0», είναι $00001000_2 = 8_{10}$, αφού 8<100, το αποτέλεσμα της εξόδου είναι το αναμενόμενο αφού εμφανίζεται ο αριθμός 8_{10} με τον τρόπο που ζητείται, οι δεκάδες με δυαδική μορφή στα 4 MSB και οι μονάδες με δυαδική μορφή στα 4 LSB. Στην συνέχεια ανοίγουμε και το τρίτο dip switch, με αυτό τον τρόπο έχουμε στο input τον αριθμό 12_{10}, καθώς είναι $00001100_2 = 12_{10}$, και 12<100, άρα όμοια με προηγουμένως στο output παρατηρώ στα 4 MSB τις δεκάδες, δηλαδή 0001 και στα 4 LSB τις μονάδες του αριθμού, δηλαδή 0010. Έπειτα, ανοίγουμε και το έκτο dip switch, με αυτό τον τρόπο έχουμε στο input τον αριθμό 44_{10}, καθώς είναι $00101100_2 = 44_{10}$, και 44<100, άρα όμοια με προηγουμένως στο output παρατηρώ στα 4 MSB τις δεκάδες, δηλαδή $0100_2 = 4_{10}$ και στα 4 LSB τις μονάδες του αριθμού, δηλαδή $0100_2 = 4_{10}$. Ανοίγοντας και το έβδομο dip switch, στο input έχουμε τον αριθμό 108_{10}, αφού $01101100_2 = 108_{10}$, με 100<108<200, άρα επιθυμώ να αναβοσβήνουν διαρκώς τα 4 LSB των LEDs το οποίο και συμβαίνει. Τέλος ανοίγοντας και το όγδοο dip switch, στο input έχουμε τον αριθμό 236_{10}, αφού $11101100_2 = 236_{10}$, με 200 < 236, άρα επιθυμώ να αναβοσβήνουν διαρκώς τα 4 MSB των LEDs το οποίο και παρατηρώ. Άρα το πρόγραμμα λειτουργεί με τον τρόπο που ζητείται.
2. Σχετικά με την αναπαράσταση στα LEDs έχουμε τις ίδιες παρατηρήσεις με την προηγούμενη άσκηση (συγκεκριμένα παρατηρήσεις 3 και 4)

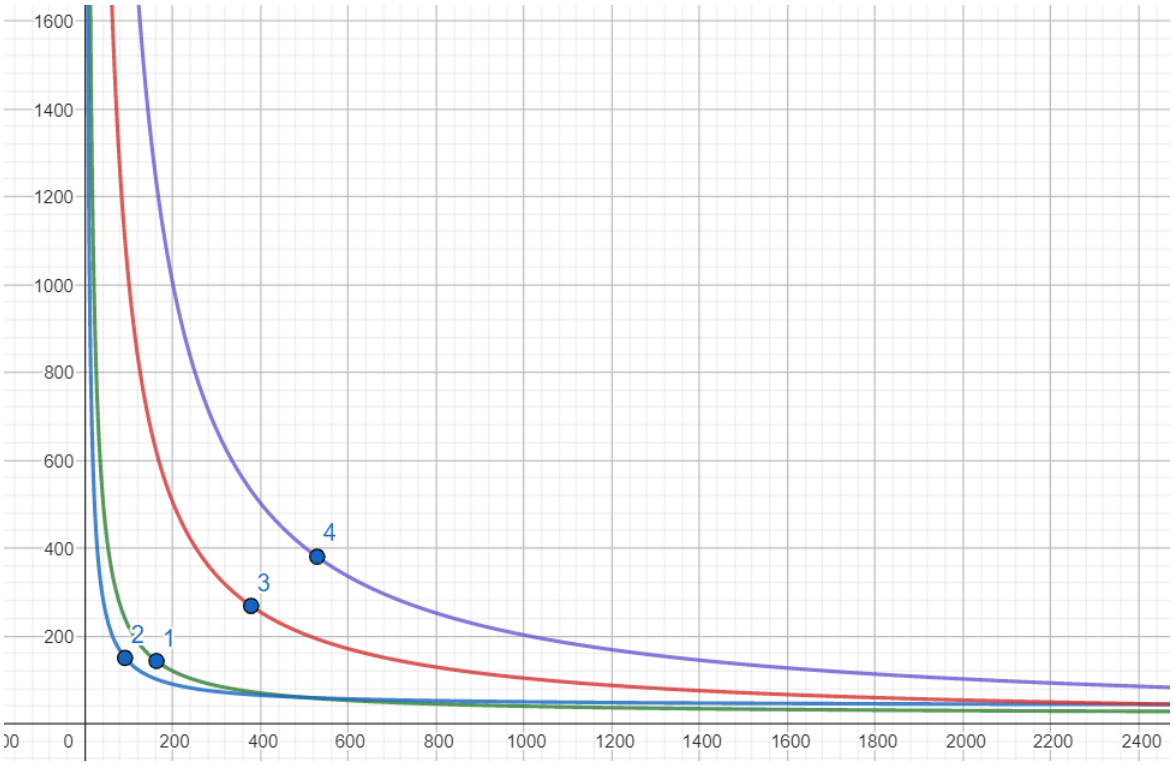
ΘΕΩΡΗΤΙΚΕΣ ΑΣΚΗΣΕΙΣ

ΑΣΚΗΣΗ 4

Οι συναρτήσεις κόστους ανά τεμάχιο για κάθε τεχνολογία φαίνονται παρακάτω:

1. $f_1(x) = \frac{20000+20x}{x}$
2. $f_2(x) = \frac{10000+40x}{x}$
3. $f_3(x) = \frac{100000+4x}{x}$
4. $f_4(x) = \frac{200000+2x}{x}$

Όπου οι αντίστοιχες καμπύλες που προκύπτουν είναι οι ακόλουθες:



Στην συνέχεια εξισώνω την κάθε μία από τις συναρτήσεις κόστους σε ζεύγη και λαμβάνουμε τα ακόλουθα σημεία τομής των καμπυλών:

(1η με 2η) – $f_1(x) = f_2(x) \Rightarrow x = 500$
(1η με 3η) – $f_1(x) = f_3(x) \Rightarrow x = 5000$
(1η με 4η) – $f_1(x) = f_4(x) \Rightarrow x = 10000$
(2η με 3η) – $f_2(x) = f_3(x) \Rightarrow x = 2500$
(2η με 4η) – $f_2(x) = f_4(x) \Rightarrow x = 5000$
(3η με 4η) – $f_3(x) = f_4(x) \Rightarrow x = 50000$

Λαμβάνοντας τα παραπάνω αποτελέσματα και συνδυάζοντάς τα με τις γραφικές παραστάσεις που βρίσκονται πιο πάνω, αντιλαμβανόμαστε ότι οι συμφερότερες (χαμηλότερου κόστους) περιοχές κάθε τεχνολογίας είναι:

1. I.C: $500 < x < 5000$
2. FPGA: $0 < x < 500$
3. SoC-1: $5000 < x < 50000$
4. SoC-2: $50000 < x$

Παρατηρώ ότι οι τεχνολογίες που έχουν μεγαλύτερο κόστος σχεδίασης, SoC-1 και SoC-2, είναι πιο κερδοφόρες για υψηλότερους αριθμούς τεμαχίων.

Έστω w το κόστος ανά στοιχείων IC για την τεχνολογία των FPGAs, τότε το συνολικό κόστος κατασκευής για την δεύτερη τεχνολογία θα είναι μικρότερο της πρώτης όταν:

$$f_2'(x) = \frac{10000 + (10 + w)x}{x} < \frac{20000 + 20x}{x} = f_1(x) \Rightarrow w < \frac{10000}{x} + 10$$

Για $x \to \infty$ θα πρέπει να ισχύει $w < 10$, στην πραγματικότητα είναι αδύνατο να παρασκευαστούν άπειρα τεμάχια επομένως αρκεί να ισχύει $w \geq 10$. Άρα όποια τιμή και να έχει το νέο κόστος μεταξύ 0 και 10 οδηγείται στον αποκλεισμό της πρώτης τεχνολογίας.