

Συστήματα Μικροϋπολογιστών
6ο Εξάμηνο
1η Ομάδα Ασκήσεων

Χαρδούβελης Γεώργιος-Ορέστης
el15100
6ο Εξάμηνο

Κυριάκου Αθηνά
el17405
6ο Εξάμηνο

Άσκηση 1

Ο κώδικας που δόθηκε σε γλώσσα μηχανής για τη συγκεκριμένη άσκηση, όπου με **Bold** επισημαίνονται οι εντολές, είναι:

06 01 3A 00 20 FE 00 CA 13 08 1F DA 12 08 04 C2 0A 08 78 2F 32 00 30 CF

Χρησιμοποιώντας τον πίνακα 2 του παραρτήματος 2 των σημειώσεων *Εισαγωγή στο Εκπαιδευτικό Σύστημα mLAB* έγινε συμβολομετάφραση του παραπάνω κώδικα. Η αντιστοιχία εντολών κώδικα μηχανής με Assembly είναι:

Κωδικός γλώσσας μηχανής	Εντολή Assembly
06	MVI B
3A	LDA
FE	CPI
CA	JZ
1F	RAR
DA	JC
04	INR B
C2	JNZ
78	MOV A,B
2F	CMA
32	STA (address)
CF	RST 1

Παρακάτω φαίνεται το πρόγραμμα σε Assembly που προέκυψε (**Bold**), σε αντιστοιχία με τη γλώσσα μηχανής (*Italic*) και τη συμβολική διεύθυνση μνήμης που έχει αποθηκευτεί (*Italic grey*).

0800 06 **MVI B, 01H**
0801 01
0802 3A **LDA 2000H**
0803 00
0804 20
0805 FE **CPI 00**
0806 00

0807 CA JZ 0813H

0808 13

0809 08

080A 1F RAR

080B DA JC 0812H

080C 12

080D 08

080E 04 INR B

080F C2 JNZ 08AH

0810 0A

0811 08

0812 78 MOV A, B

0813 2F CMA

0814 32 STA 3000H

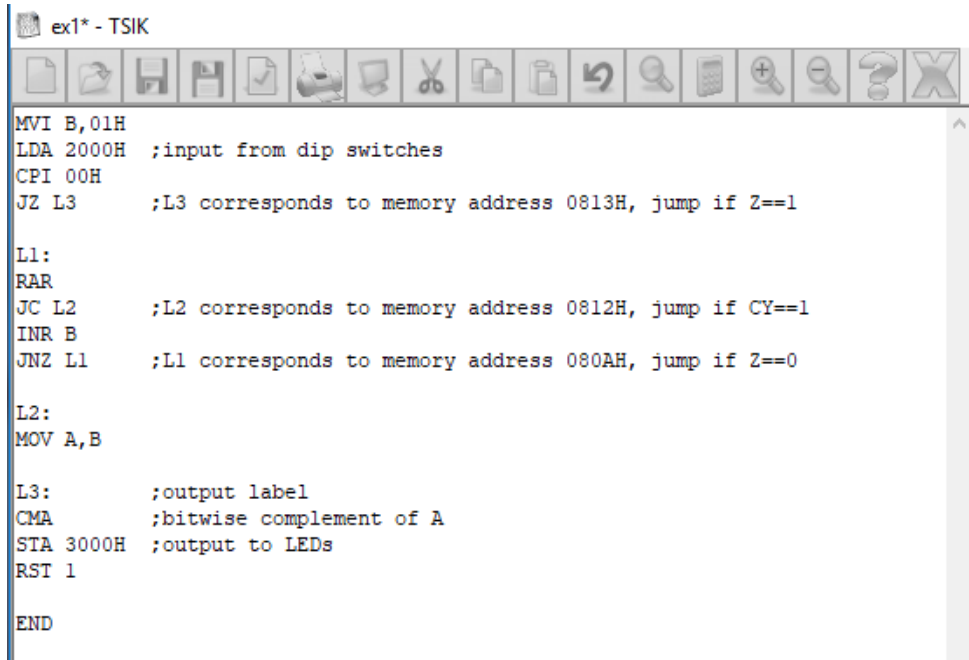
0815 00

0816 30

0817 CF RST 1

Προσομοίωση:

Για να δοκιμαστεί ο παραπάνω κώδικας Assembly στο πρόγραμμα προσομοίωσης του εκπαιδευτικού συστήματος μLAB, χρειάστηκε να γίνουν κάποιες τροποποιήσεις Συγκεκριμένα, να τοποθετηθούν **Labels** στις διευθύνσεις που παραπέμπουν οι εντολές άλματος καθώς και να προστεθεί η εντολή **END** στο τέλος του προγράμματος (Σχήμα 1). Η αντιστοιχία της συμβολικής γλώσσας με τον κώδικα μηχανής καθώς και οι θέσεις μνήμης στις οποίες αποθηκεύονται οι εντολές, επαληθεύτηκαν και από το πρόγραμμα προσομοίωσης, όπως φαίνεται στο Σχήμα 2.



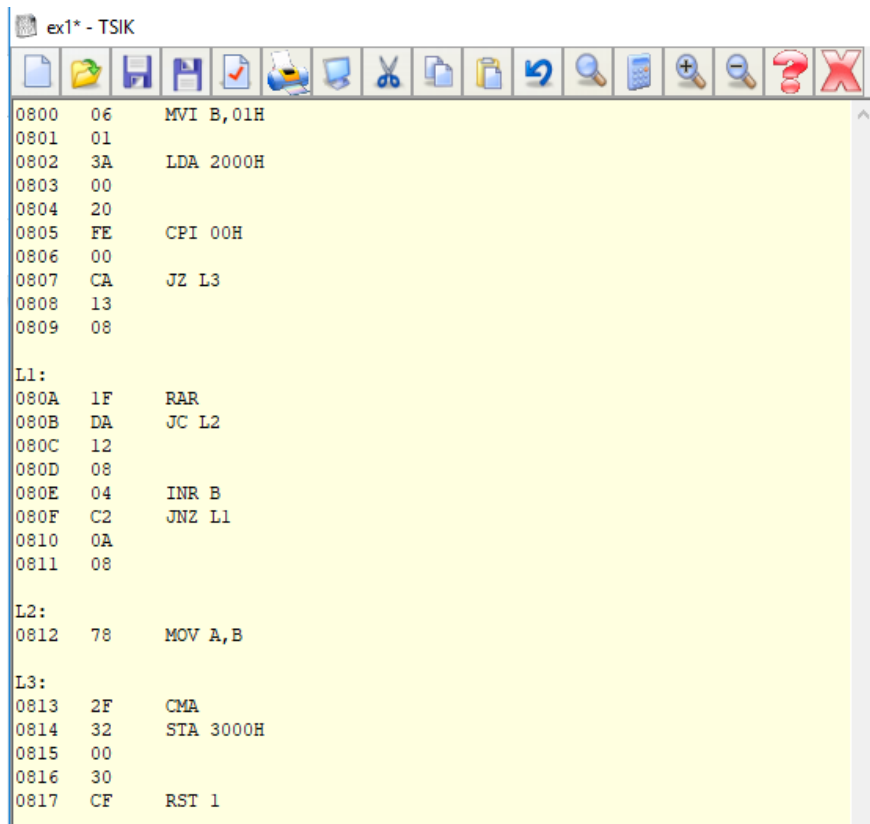
```
ex1* - TSIK
MVI B,01H
LDA 2000H ;input from dip switches
CPI 00H
JZ L3      ;L3 corresponds to memory address 0813H, jump if Z==1

L1:
RAR
JC L2      ;L2 corresponds to memory address 0812H, jump if CY==1
INR B
JNZ L1     ;L1 corresponds to memory address 080AH, jump if Z==0

L2:
MOV A,B

L3:      ;output label
CMA      ;bitwise complement of A
STA 3000H ;output to LEDs
RST 1
END
```

Σχήμα 1



Address	Hex	Instruction
0800	06	MVI B,01H
0801	01	
0802	3A	LDA 2000H
0803	00	
0804	20	
0805	FE	CPI 00H
0806	00	
0807	CA	JZ L3
0808	13	
0809	08	
L1:		
080A	1F	RAR
080B	DA	JC L2
080C	12	
080D	08	
080E	04	INR B
080F	C2	JNZ L1
0810	0A	
0811	08	
L2:		
0812	78	MOV A,B
L3:		
0813	2F	CMA
0814	32	STA 3000H
0815	00	
0816	30	
0817	CF	RST 1

Σχήμα 2

Συνοπτικά η **λειτουργία** του προγράμματος είναι:

Το περιεχόμενο του καταχωρητή B αρχικοποιείται σε 1 και στον καταχωρητή A φορτώνεται το περιεχόμενο της θέσης μνήμης 2000H, δηλαδή η 8-bit είσοδος από τα dip switches. Στην περίπτωση που η **είσοδος έχει τιμή 0** γίνεται άλμα στην υπορουτίνα **L3** διαφορετικά εκτελείται σειριακά η **L1**.

L1: Το περιεχόμενο του A περιστρέφεται δεξιά κατά μία θέση μέσω της σημαίας κρατουμένου. Δηλαδή γίνεται δεξιά ολίσθηση του περιεχομένου του A, το A7 (MSB) παίρνει την τιμή της CY και η CY παίρνει την τιμή του A0 (LSB) πριν την ολίσθηση. Αν CY==1 (A0==1), εκτελείται η L2 διαφορετικά το περιεχόμενο του B αυξάνεται κατά 1. Αν το νέο αυτό περιεχόμενο του B δεν είναι 0, εκτελείται η ξανά L1 αλλιώς σειριακά η L2.

L2: Ο καταχωρητής A παίρνει την τιμή του B.

L3: Πρόκειται για την υπορουτίνα εξόδου. Το τελικό περιεχόμενο του A αντικαθίσταται από το bit προς bit συμπλήρωμά του και η τιμή αυτή εμφανίζεται στα LEDS (θέση μνήμης 3000H).

Παραδείγματα εξόδων από προσομοίωση:

1) Είσοδος A=0H - Έξοδος A=FFH → όλα τα LEDs σε κατάσταση OFF (Σχήμα 3)

The screenshot displays the TSIM software interface. On the left, the assembly code is listed with addresses from 0800 to 0817. The code includes instructions like MVI, LDA, CPI, JZ, RAR, JC, INR, JNZ, MOV, CMA, STA, and RST. On the right, the hardware status is shown, including registers A, B, C, D, E, H, L, and flags S, Z, X5, AC, P, V, CY, PC, SP. The status LEDs are shown as OFF, and the output display shows 00 18 00.

Address	Op Code	Instruction
0800	06	MVI B, 01H
0801	01	
0802	3A	LDA 2000H
0803	00	
0804	20	
0805	FE	CPI 00H
0806	00	
0807	CA	JZ L3
0808	13	
0809	08	
L1:		
080A	1F	RAR
080B	DA	JC L2
080C	12	
080D	08	
080E	04	INR B
080F	C2	JNZ L1
0810	0A	
0811	08	
L2:		
0812	78	MOV A, B
L3:		
0813	2F	CMA
0814	32	STA 3000H
0815	00	
0816	30	
0817	CF	RST 1

A	I	B	C	D	E	H	L
FF	0B	01	00	00	00	00	00

S	Z	X5	AC	P	V	CY	PC	SP
0	1	0	0	1	0	0	0800	08B0

RESET	RUN	D	E	F	
HW/R STEP	INSTR STEP	A	B	C	
INTRPT	FETCH PC	7	8	9	
FETCH ADDR	FETCH REG	4	5	6	
DECR	STORE/INCR	0	1	2	3

Σχήμα 3

ex1* - TSIM

0800 06 MVI B,01H
 0801 01
 0802 3A LDA 2000H
 0803 00
 0804 20
 0805 FE CPI 00H
 0806 00
 0807 CA JZ L3
 0808 13
 0809 08

L1:
 080A 1F RAR
 080B DA JC L2
 080C 12
 080D 08
 080E 04 INR B
 080F C2 JNZ L1
 0810 0A
 0811 08

L2:
 0812 78 MOV A,B

L3:
 0813 2F CMA
 0814 32 STA 3000H
 0815 00
 0816 30
 0817 CF RST 1

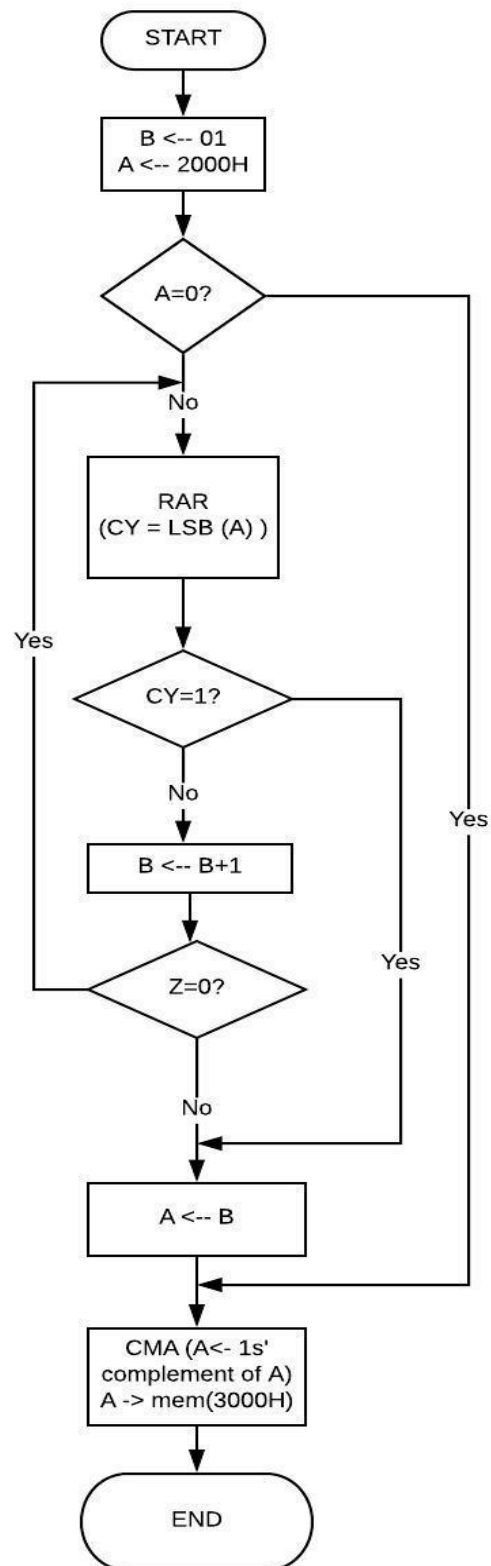
Registers: A I B C D E H L
 FF 0B 01 00 00 00 00 00
 S Z X5 AC P V CY PC SP
 0 1 0 0 1 0 0 0800 0BB0

7-segment display: 001800

Controls: RESET RUN
 HOLD STEP INSTR STEP
 INTRPT FETCH PC
 FETCH ADDR FETCH REG
 DEC STORE/INCR 0 1 2 3

Σχήμα 4

Το διάγραμμα ροής είναι:



Σε περίπτωση που επιθυμούμε να έχουμε **συνεχόμενη μορφή** του προγράμματος, δηλαδή να επαναλαμβάνεται χωρίς τέλος, μπορούμε να εκτελέσουμε το πρόγραμμα σε έναν βρόχο, αφαιρώντας την συνθήκη τερματισμού RST 1.

LOOP:

MVI B, 01H

LDA 2000H

CPI 00

JZ L3

L1:

RAR

JC L2

INR B

JNZ L1

L2:

MOV A, B

L3:

CMA

STA 3000H

JMP LOOP

END

Άσκηση 2

Το ζητούμενο πρόγραμμα σε assembly έχει ως εξής:

```
IN 10H
MVI B,01H      ; φτιάχνω καθυστέρηση για τη DELB 0,5sec
MVI C,F4H      ; αφού (01F4)16=500ms =0.5sec
MVI D,01H      ; στον D θα έχουμε αποθηκευμένο το LEDακι που θέλουμε να ανάψει και
                ; αρχικοποιείται στο 1
MVI E,01H      ; στον E θα έχουμε αποθηκευμένο ένα flag για να γνωρίζουμε
                ; αν κινούμαστε δεξιά ή αριστερά όταν το LSB είναι ON
```

```
MAIN:
LDA 2000H
ANI 02H        ; συγκρίνουμε να δούμε αν το 2ο LSB ψηφίο είναι αναμμένο
JZ SKIP        ; αν δεν είναι συνεχίζει
MVI A,01H      ; αν δεν είναι,, ανάβουμε το λαμπάκι στη θέση μηδέν
CALL TADA
JMP MAIN       ; επιστρέφουμε στο loop και ξανασυγκρίνουμε
```

```
SKIP:
MOV A,D
CALL TADA      ; ανάβουμε το LED
LDA 2000H
ANI 01H        ; ελέγχουμε αν το LSB είναι ON
JNZ WEIRDSTUFF ; αν είναι ON πάμε και ελέγχουμε τι κίνηση θα κάνει
```

```
CIRCLE_LOOP:  ; αλλιώς, αν είναι OFF κάνω απλά κυκλική περιστροφή
MOV A,D
RLC           ; κάνει κυκλική κίνηση 012345670123
MOV D,A
JMP MAIN      ; επιστρέφω και κάνω σύγκριση
```

```
WEIRDSTUFF:
MOV A,E
ANI 01H       ; αν το E είναι 1 κινείται αριστερά
JNZ WL        ; πάμε στο loop στο οποίο κινείται αριστερά
```

```
WR:           ; αλλιώς κινείται δεξιά
MOV A,D
RRC
MOV D,A
MOV A,D
ANI 01H       ; ελέγχουμε αν το D είναι ίσο με 1 (ώρα δηλαδή να κινηθούμε ανάποδα)
JZ MAIN       ; αν όχι επιστρέφουμε στην MAIN
MVI E,01H     ; αν ναι, αλλάζουμε το flag μας στο E σε 1
```


JMP MAIN

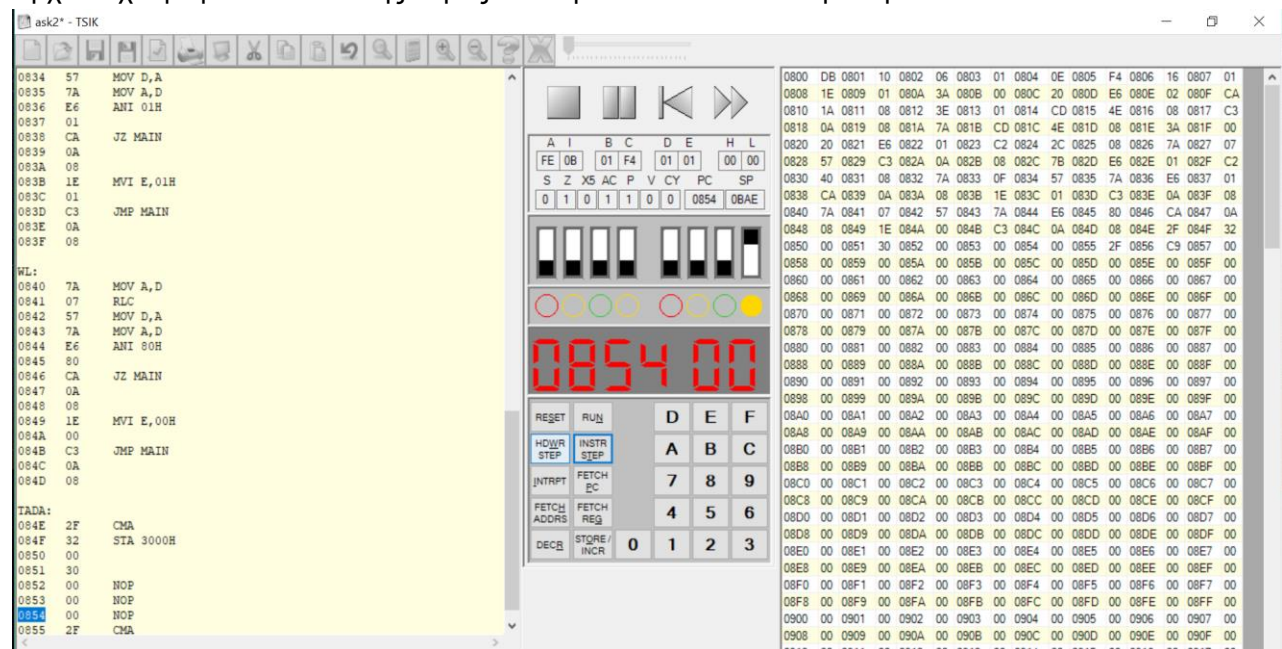
WL: ;loop ώστε να κινείται αριστερά
MOV A,D
RLC
MOV D,A
MOV A,D
ANI 08H ;ελέγχουμε αν το D είναι ίσο με 8 (ώρα δηλαδή να κινηθούμε ανάποδα)
JZ MAIN ;αν όχι επιστρέφουμε στην MAIN
MVI E,00H ;αν ναι, αλλάζουμε το flag μας στο E σε 0
JMP MAIN

TADA: ;η συνάρτηση που ανάβει το επιθυμητό λαμπάκι
CMA ;συμπληρώνω ως προς 1 εφόσον έχω αρνητική λογική
STA 3000H ;βάζω την καθυστέρηση για να μείνει αναμμένο το λαμπάκι για 0.5sec
CMA
RET

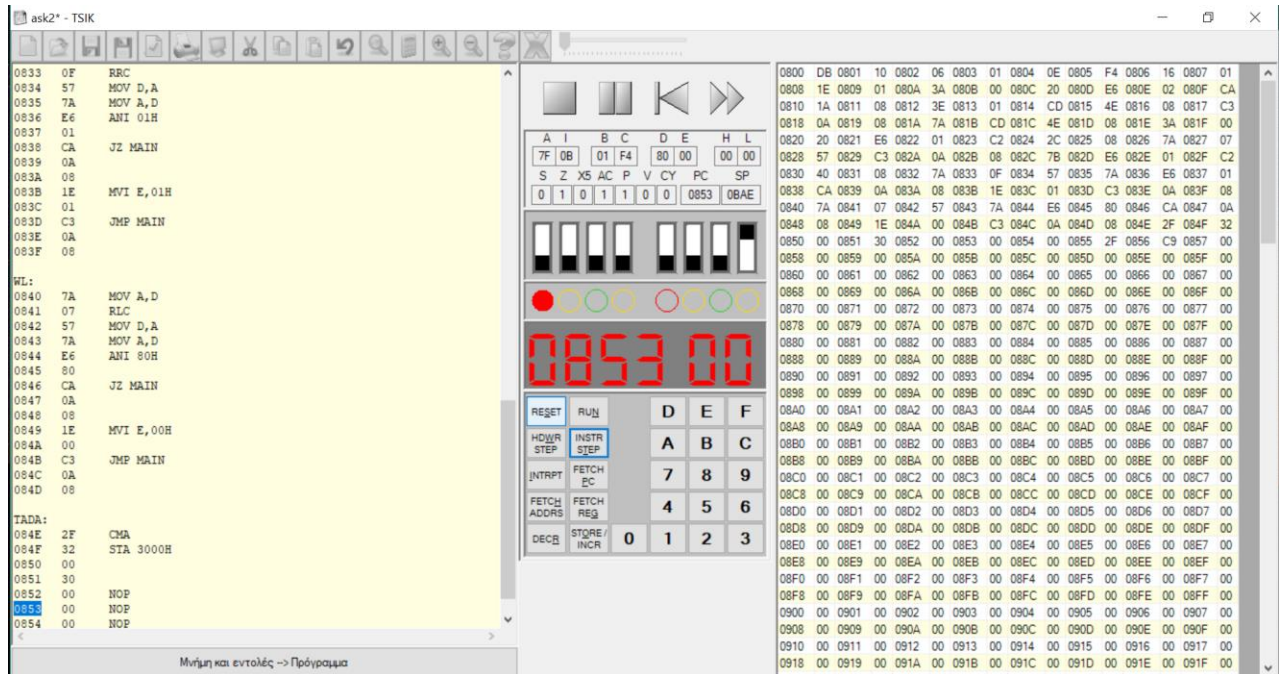
END

Επαληθεύοντας μέσω της προσομοίωσης / βηματική εκτέλεση του προγράμματός σας (αντικαθιστώντας τη ρουτίνα DELB με 3 εντολές NOP) έχουμε:

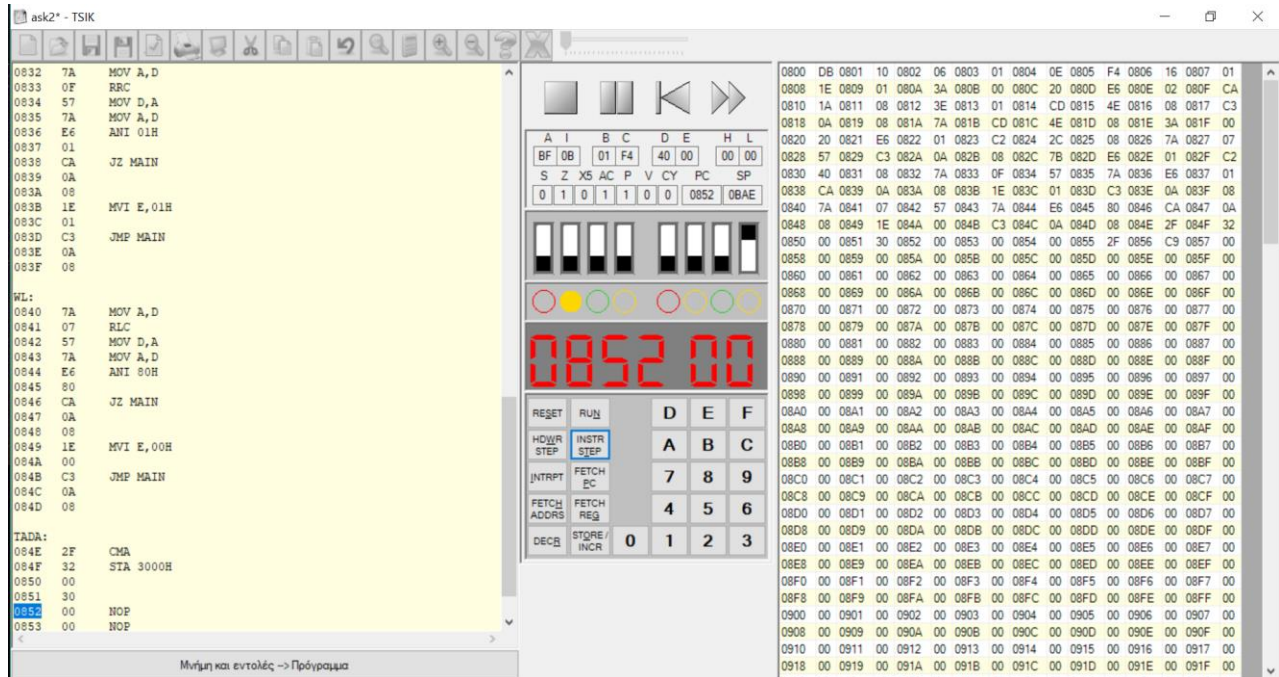
Αρχικά έχουμε μόνο το LSB της θύρας των dip switch ON και ανάβει πρώτα το led 0.



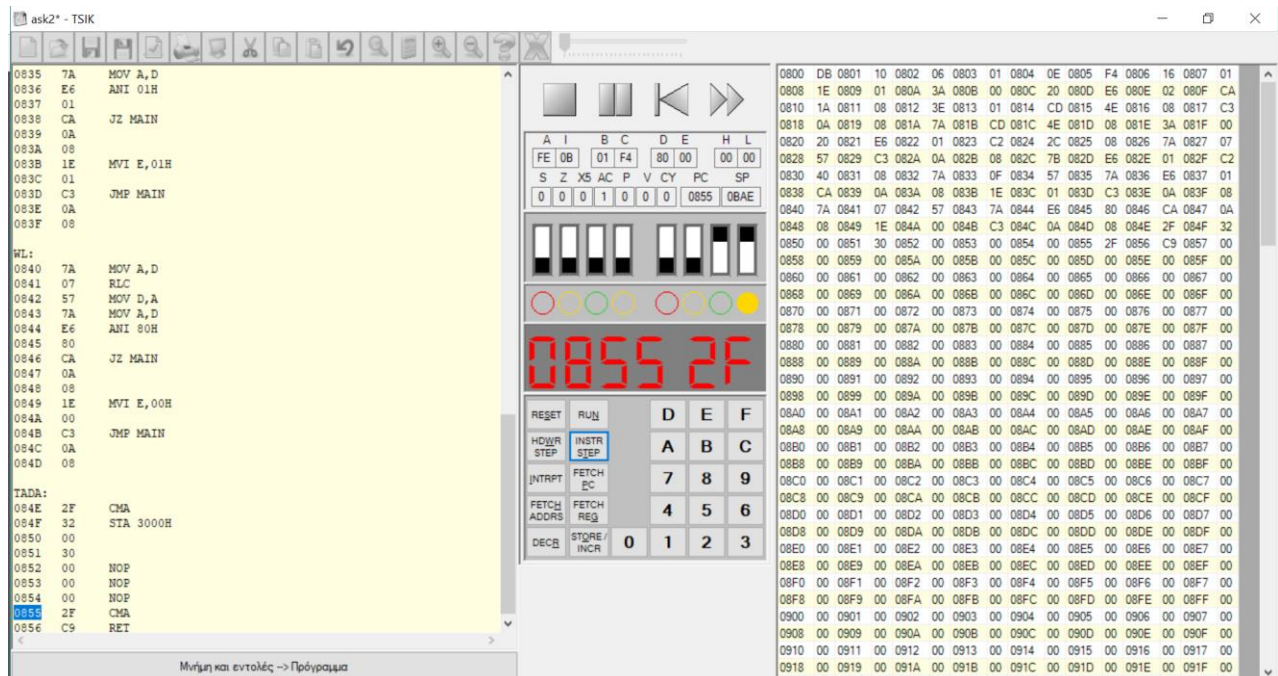
Στη συνέχεια συνεχίζοντας με μόνο το LSB της θύρας των dip switch ON(άρα και το LSB και το 2o LSB), συνεχίζει κίνηση προς τα δεξιά και ύστερα από 7 επαναλήψεις έχουμε αναμμένο το 8o led:



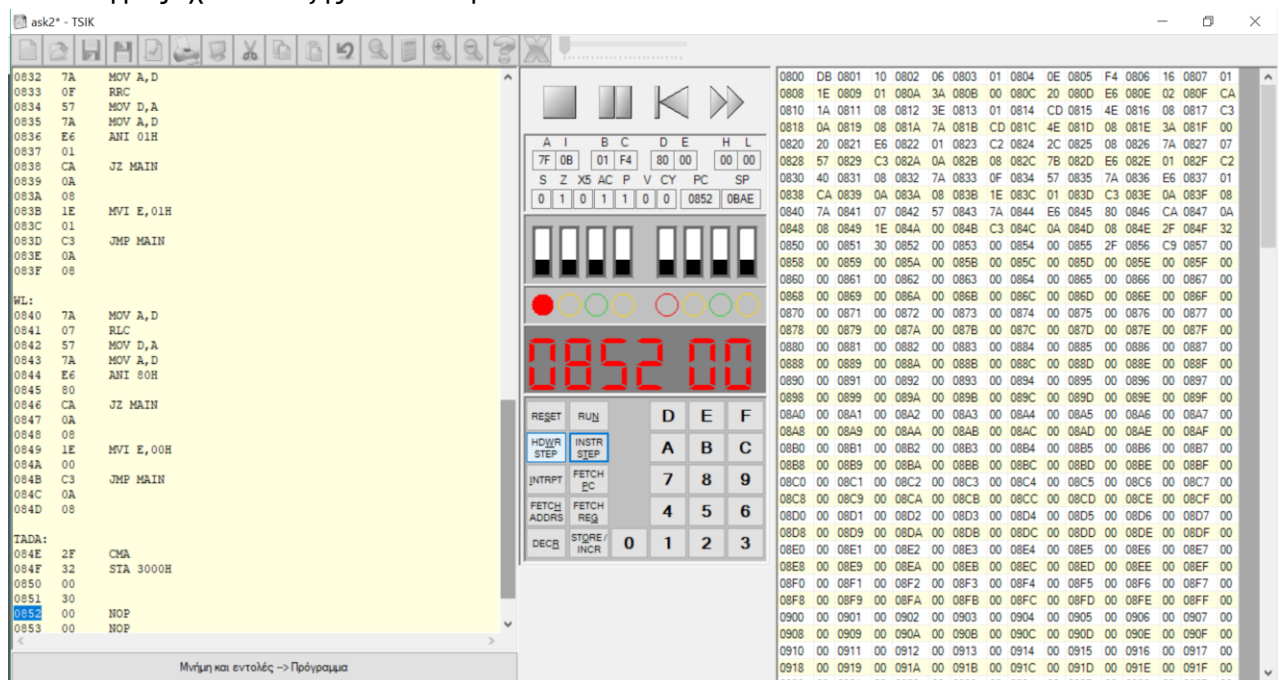
Εφόσον έφτασε στο 80, αμέσως μετά συνεχίζει ανάποδα κίνηση προς τα αριστερά οπότε στην επόμενη επανάληψη έχουμε αναμμένο το 7ο led:

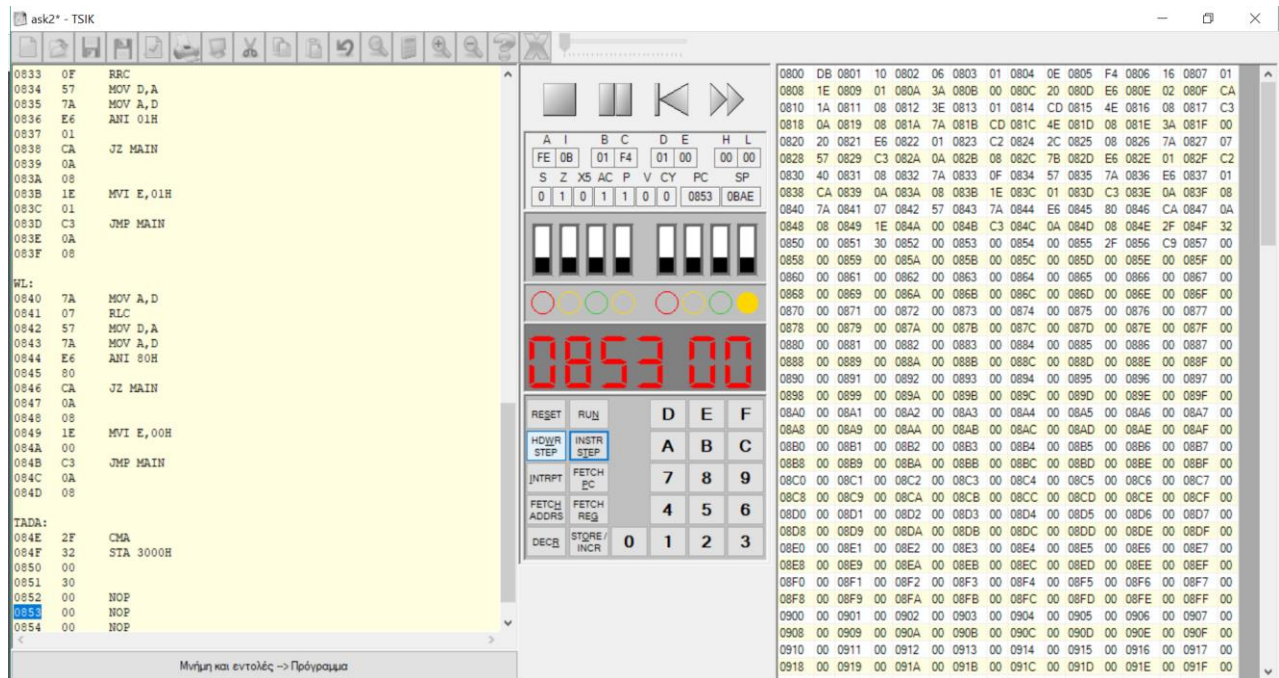


Στη συνέχεια θέτουμε το 2ο LSB στοιχείο της θύρας των dip switch ON. Εκεί έπειτα από όσες επαναλήψεις και να έχουμε, μένει αναμμένο το πρώτο led.



Στη συνέχεια, θέτουμε όλα τα bit της θύρας των dip switch OFF. Έτσι, καταρχάς, η κίνηση συνεχίζει από το σημείο που είχαμε μείνει πριν, ενώ έχουμε πλέον κυκλική κίνηση. Οπότε οι δύο επόμενες επαναλήψεις έχουν το εξής αποτέλεσμα:





Βλέπουμε λοιπόν πως σε κάθε περίπτωση τηρήθηκαν οι οδηγίες που δώθηκαν στην εκφώνηση.

Άσκηση 3

Ο κώδικας για την υλοποίηση της άσκησης είναι:

```
LDA 2000H    ;input from input ports corresponding to memory address 200H
MVI B,FFH    ;B register for counting dozens, initialized to -1
MVI C,FFH    ;C register for counting hundreds, initialized to -1
```

```
L1:          ;loop for the hundreds
INR C
SUI 64H      ;subtract 100 from the number
JNC L1
ADI 64H      ;add 100 to find the modulo
```

```
L2:          ;loop for the dozens
INR B
SUI 0AH      ;subtract 10 from the number
JNC L2
ADI 0AH      ;add 10 to find the modulo
```

```
;dozens in register B and units in register A
;total result needs to be in register A
```

```
MOV C,A
MOV A,B
```

```
RLC
RLC
RLC
RLC
ORI 0FH
MOV B,A      ;dozens in MSB of register B
```

```
MOV A,C
ORI F0H      ;units in LSB of register A
```

```
ANA B        ;bitwise AND operation (A)<-(A)AND(B), final result in A
STA 3000H    ;output in LEDs (memory address 3000H)
              ;dozens in MSB, units in LSB
```

```
HLT
END
```

Κάποια σημεία που χρειάζεται να επισημανθούν για τις συσκευές E/E που χρησιμοποιούνται είναι:
Η είσοδος της **συσκευής εισόδου** (dip switches) αποθηκεύεται στη διεύθυνση μνήμης 2000H για αυτό και ο αριθμός αρχικά διαβάζεται από αυτή τη θέση μνήμης και αποθηκεύεται στον καταχωρητή A.

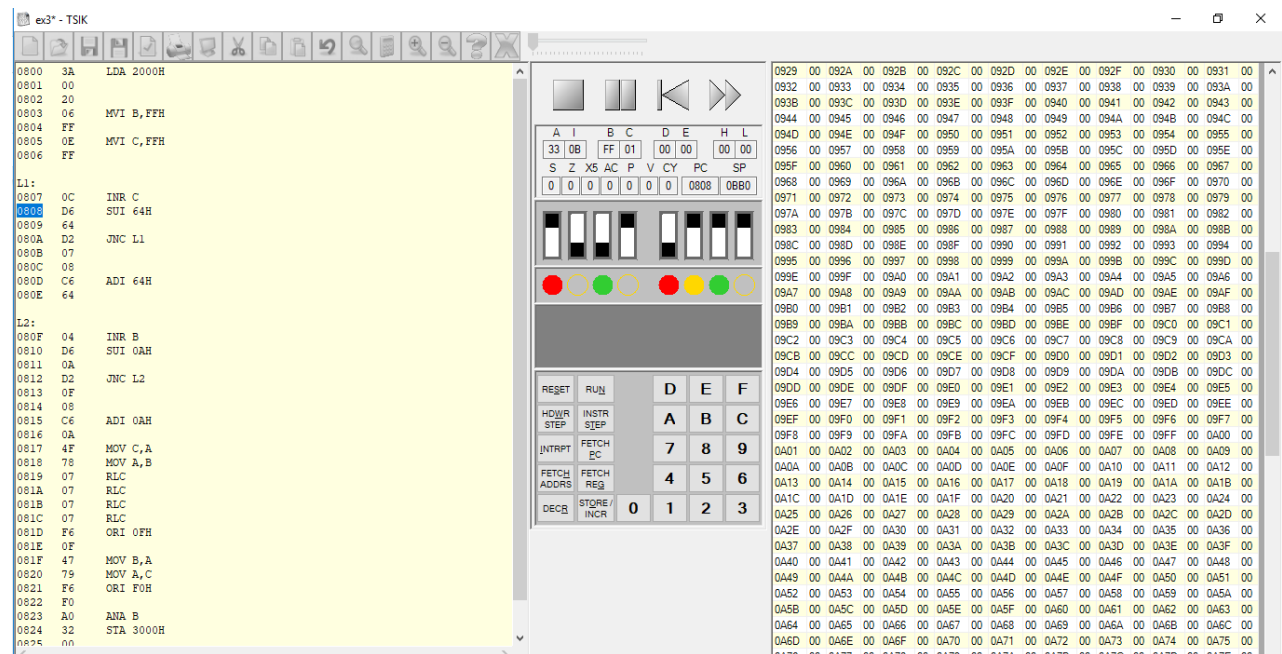
Μία από τις **συσκευές εξόδου** του προσομοιωτή είναι τα LEDs. Για να αναπαρασταθεί ένα 8bits αποτέλεσμα στα LEDs πρέπει να αποθηκευτεί στη θέση μνήμης 3000H, όπως συμβαίνει μετά τον υπολογισμό του τελικού αποτελέσματος. Επίσης, τα LEDs είναι συνδεδεμένα σε αρνητική λογική, δηλαδή όταν ένα LED είναι σε κατάσταση HIGH (αντίστοιχο bit 1), είναι σβηστό OFF.

Σημειώνεται τέλος ότι εφόσον ο δυαδικός αριθμός που μετατρέπεται σε δεκαδικό είναι αποθηκευμένος σε 8bits και θεωρούμε βάσει του λυμένου παραδείγματος ότι είναι μη προσημασμένος, ανήκει στο διάστημα [0,255].

Παράδειγμα εκτέλεσης προσομοίωσης:

Έστω ότι θέλουμε να εμφανίσουμε στην έξοδο τις δεκάδες και τις μονάδες του αριθμού (151)10. Η αναπαράσταση του αντίστοιχου δυαδικού αριθμού σε 8bits είναι: $(1001\ 0111)_2 = 97H$ και εισάγεται στα dip switches. Έχει δηλαδή 5 δεκάδες $(0101)_2$ και 1 μονάδα $(0001)_2$. Οπότε το τελικό αποτέλεσμα που αποθηκεύεται στη διεύθυνση 3000H πρέπει να είναι $(0101\ 0001)_2$ και λόγω αρνητικής λογικής να ανάβουν τα LED A7,A5,A3,A2,A1 όπου A7 το LED που αντιστοιχεί στο περισσότερο σημαντικό bit.

Αφού εκτελέσουμε τον κώδικα στην προσομοίωση επαληθεύεται ότι ανάβουν τα LED που προαναφέρθηκαν (Σχήμα 5).

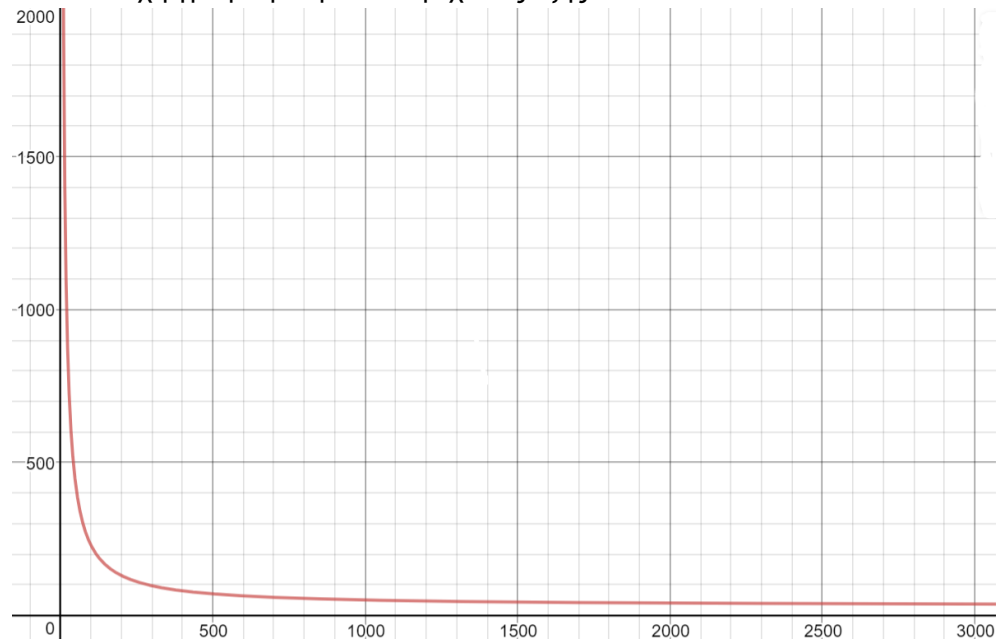


Σχήμα 5

Άσκηση 4

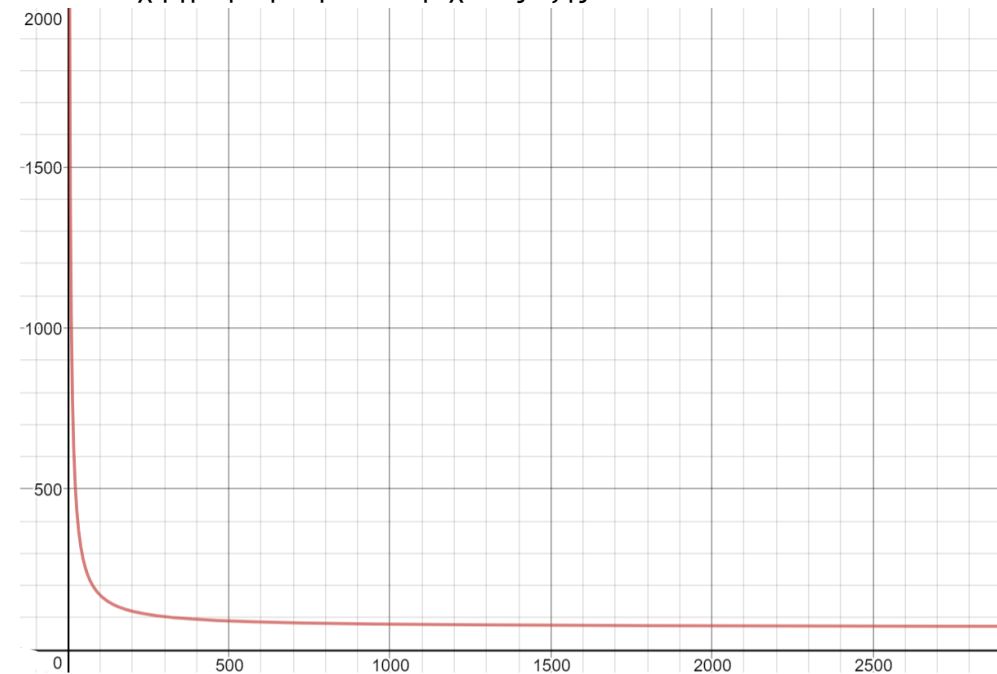
Στην 1η τεχνολογία έχουμε κόστος ανά τεμάχιο: $(20000 + 30X) / X$

Η αντίστοιχη γραφική παράσταση έχει ως εξής:



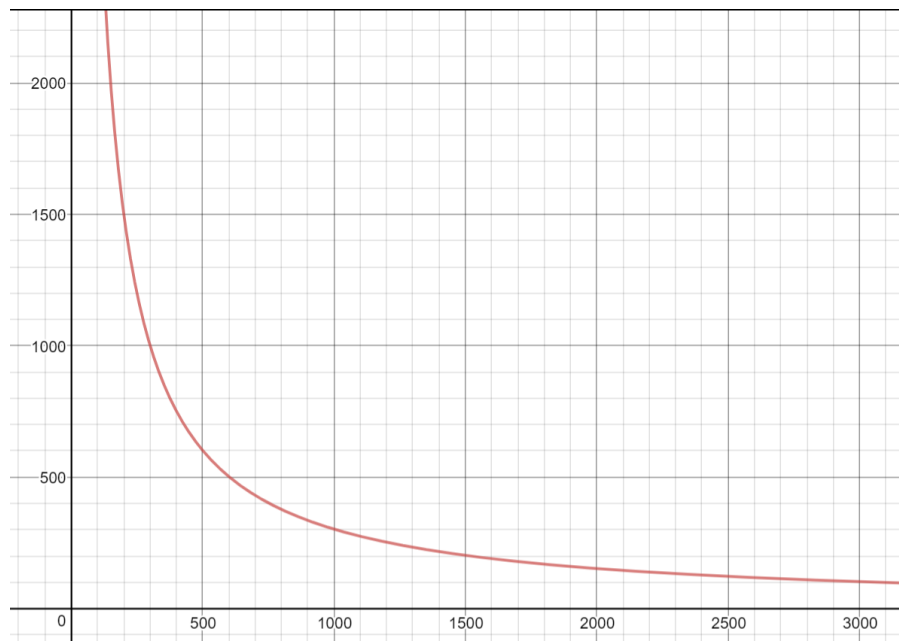
Στην 2η τεχνολογία έχουμε κόστος ανά τεμάχιο: $(10000 + 70X) / X$

Η αντίστοιχη γραφική παράσταση έχει ως εξής:



Στην 3η τεχνολογία έχουμε κόστος ανά τεμάχιο: $(300000 + 2X) / X$

Η αντίστοιχη γραφική παράσταση έχει ως εξής:



Από τις εξισώσεις που έχουν δημιουργηθεί παρατηρούμε πως η 1η και η 2η τεχνολογία έχουν ίδιο κόστος για 250 τεμάχια και η 1η και η 3η για 10000 τεμάχια. (ή 2η και η 3η για περίπου 4265 τεμάχια).

- Επομένως, **μέχρι και τα πρώτα 250** τεμάχια συμφέρει η **2η τεχνολογία**.
- **Από 250 έως 10000** τεμάχια συμφέρει η **1η τεχνολογία**
- Και τέλος **από 10000 τεμάχια και πάνω** συμφέρει η **3η τεχνολογία**

Για να εξαφανιστεί η πρώτη επιλογή θα έπρεπε (αν θεσουμε w = κόστος I.C των FPGAs) :
 $10000+(w+10)x < 20000+30x$ για κάθε $x \leq 10000$ (αφού από εκεί και έπειτα ούτως ή άλλως η 3η τεχνολογία είναι καλύτερη από την 1η).

Επομένως για $x=10000$ έχουμε :

$$10000+(w+10)*10000 < 320000$$

$$1+w+10 < 32$$

$$w \leq 21$$

Επομένως για κόστος **μικρότερο ή ίσο των 21€** η τεχνολογία 1 θα εξαφανιστεί η επιλογή της πρώτης τεχνολογίας.

Άσκηση 5

(a) $F1 = A(CD+B)+BC'D'$

Δομική περιγραφή

```
module F1_structural(F1,A,B,C,D);
    output F1;
    input A,B,C,D;
    wire w1,w2,w3,w4,w5,w6;
    not
        G4(w1,C),
        G5(w2,D);
    and
        G1(w4,C,D),
        G3(w6,A,w5),
        G6(w3,w2,w1,B);
    nor
        G2(w5,B,w4),
        G7(F1,w6,w3);
endmodule
```

Περιγραφή ροής δεδομένων

```
module F1_dataflow(F1,A,B,C,D);
    output F1;
    input A,B,C,D;
    assign F1=(A&((C&D)||B))||(B&(!C)&(!D));
endmodule
```

(b) $F2(A,B,C,D) = \Sigma(0,2,3,5,7,8,10,11,14,15)$

Με χρήση πίνακα Karnaugh και τεχνικής ελαχιστοποίησης λογικής συνάρτησης, η F2 σε μορφή αθροίσματος γινομένων είναι: **$F2=B'D'+CD+A'BD+AC$**

Δομική περιγραφή

```
module F2_structural(F2,A,B,C,D);
    output F2;
    input A,B,C,D;
    wire w1,w2,w3,w4,w5,w6,w7;
    not
        G1(w1,B),
        G2(w2,D),
        G5(w5,A);
```

```

        and
            G3(w3,w1,w2),
            G4(w4,C,D),
            G6(w6,w5,B,D),
            G7(w7,A,C);
        or
            G8(F2,w3,w4,w6,w7);
    endmodule

```

Περιγραφή ροής δεδομένων

```

module F2_dataflow(F2,A,B,C,D);
    output F2;
    input A,B,C,D;
    assign F2=((!B)&(!D))||((C&D)||(!A)&B&D)||((A&C);
endmodule

```

(c) $F3=ABC+(A+B)CD+(B+CD)E$

Δομική περιγραφή

```

module F3_structural(F3,A,B,C,D,E);
    output F3;
    input A,B,C,D,E;
    wire w1,w2,w3,w4,w5,w6;
    and
        G1(w1,A,B,C),
        G3(w3,w2,C,D),
        G4(w4,C,D),
        G6(w6,w5,E);
    or
        G2(w2,A,B),
        G5(w5,w4,B),
        G7(F3,w1,w3,w6);
endmodule

```

Περιγραφή ροής δεδομένων

```

module F3_dataflow(F3,A,B,C,D,E);
    output F3;
    input A,B,C,D,E;
    assign F3=(A&B&C)||((A||B)&C&D)||((B||(C&D))&E);
endmodule

```

(d) $F4=A(BC+D+E)+CDE$

Δομική περιγραφή

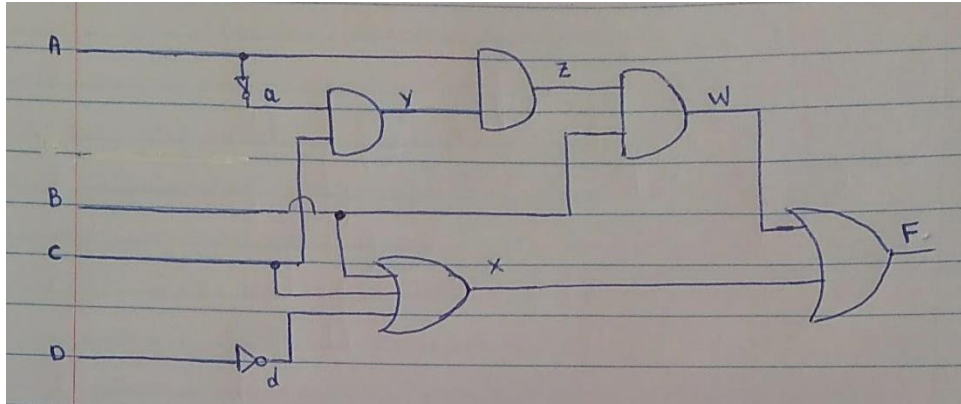
```
module F4_structural(F4,A,B,C,D,E);
    output F4;
    input A,B,C,D,E;
    wire w1,w2,w3,w4;
    and
        G1(w1,B,C),
        G3(w3,A,w2),
        G4(w4,C,D,E);
    or
        G2(w2,w1,D,E),
        G5(F4,w4,w3);
endmodule
```

Περιγραφή ροής δεδομένων

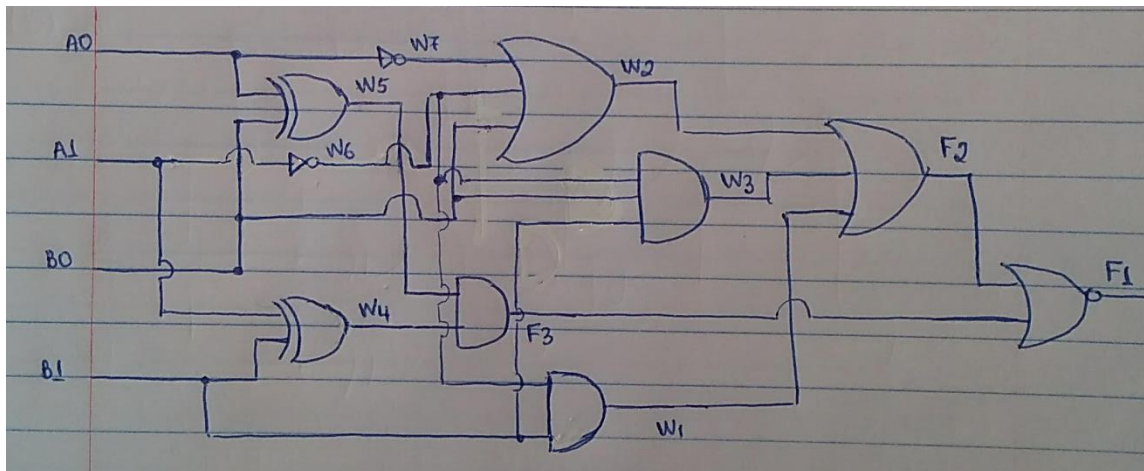
```
module F4_dataflow(F4,A,B,C,D,E);
    output F4;
    input A,B,C,D,E;
    assign F4=(A&((B&C)||D||E))||(C&D&E);
endmodule
```

Άσκηση 6

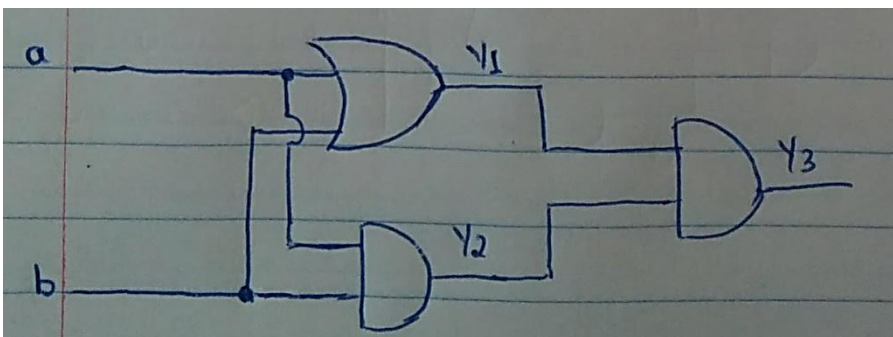
3.36 (a)



(b)

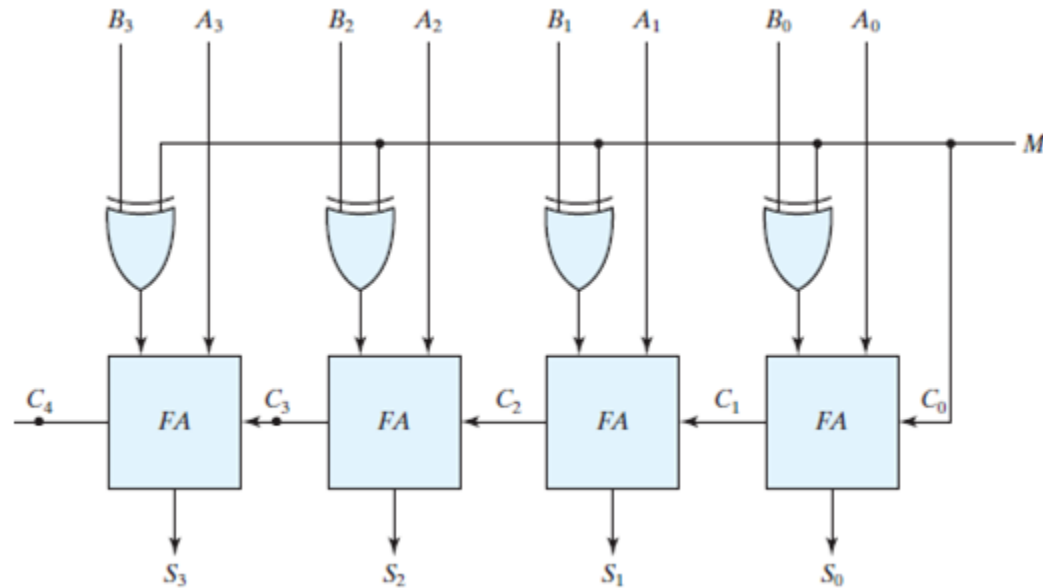


(c)



4.37 Bottom-up gate level hierarchical descriptor of a four-bit adder-subtractor for unsigned binary numbers

Η υλοποίηση του ψηφιακού κυκλώματος φαίνεται στο σχήμα.



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

```
module full_adder(S,C,x,y,z);
    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 uns_4_bit_add_sub(result,C4,A,B,M);
    output [3:0] result;
    output C4;
    input [3:0] A,B;
    input M;
    wire C1,C2,C3,X0,X1,X2,X3;
    xor
```

```

        G0(X0,M,B[0]),
        G1(X1,M,B[1]),
        G2(X2,M,B[2]),
        G3(X3,M,B[3]);
    full_adder
        FA0(result[0],C1,X0,A[0],M),
        FA1(result[1],C2,X1,A[1],C1),
        FA2(result[2],C3,X2,A[2],C2),
        FA1(result[3],C4,X3,A[3],C3);
endmodule

```

4.40 Dataflow of a four-bit adder-subtractor for unsigned binary numbers

```

module uns_4_bit_add_sub_dataflow(result,C4,A,B,M);
    output [3:0] result;
    output C4;
    input[3:0] A,B;
    input M;
    assign {result,C4}=(M)?A-B:A+B;
endmodule

```

Άσκηση 7

5.48

```
module ask_5_48 (  
    output reg y_out;  
    input x_in, clock reset  
);  
  
reg [1:0] state, next_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 state<=next_state;  
  
always @(state, x_in)  
    case(state)  
        A:    if (x_in) next_state=c; else next_state=b;  
        B:    if (x_in) next_state=d; else next_state=c;  
        C:    if (x_in) next_state=d; else next_state=b;  
        D:    if (x_in) next_state=a; else next_state=c;  
    endcase  
  
always @(state, x_in)  
    case(state)  
        a,d: y_out=~x_in;  
        b,c: y_out=x_in;  
    endcase  
  
endmodule
```

5.49

```
module ask_5_49 (  
    output [1,0] y_out,  
    input x_in, 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_in) state <= c; else state <= b;  
        b:    if (x_in) state <= c; else state <= d;  
        c:    if (x_in) state <= d; else state <= b;  
        d:    if (x_in) state <= a; else state <= c;  
    endcase
```

```

assign y_out = state;
endmodule

```

6.35 f

```

module ask_6_35f (
    output reg      [3:0] A_par,
    input   [3:0] I_par,
    input    s1,s0,
            CLK, Clear_b
);
always @(posedge CLK, negedge Clear_b)
    (if Clear_b==0) A_par <= 4'b0000;
    else
        case ({s1,s0})
            2'b01: A_par<=4'b0000;
            2'b10: A_par<=~A_par;
            2'b11: A_par<=I_par;
        endcase
endmodule

```

6.35 I

```

module ask_6_35I (
    output reg [3:0]      A_count,
    input          Up,Down,
            CLK, Clear_b
);
always @(posedge CLK, negedge Clear_b)
    if(~Clear_b)  A_count <= 4'b0000
    else if (Up)   A_count<=A_count + 1'b1; //έχει προτεραιότητα η είσοδος ελέγχου
                                                    // μέτρηση προς τα επάνω
    else if (Down) A_count<=A_count - 1'b1;
    else          A_count <= A_count //πλεονάζουσα εντολή
endmodule

```