

Συστήματα Μικροϋπολογιστών
6ο Εξάμηνο
1η Ομάδα Ασκήσεων

Χαρδούβελης Γεώργιος-Ορέστης
el15100
6ο Εξάμηνο

Κυριάκου Αθηνά
el17405
6ο Εξάμηνο

Ασκηση 1

Ο κώδικας Assembly για την υλοποίηση όλης της άσκησης βρίσκεται στο αρχείο ex1.8085.

(α) Το πρόγραμμα για την αποθήκευση των αριθμών 0-255 κατά φθίνουσα σειρά σε διαδοχικές θέσεις μνήμης ξεκινώντας από την 0900H, φαίνεται στο παρακάτω σχήμα.



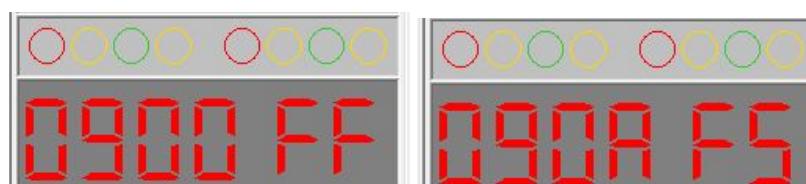
```
IN 10H
LXI H,0900H      ;starting address
MVI B,FFH        ;(B)=255

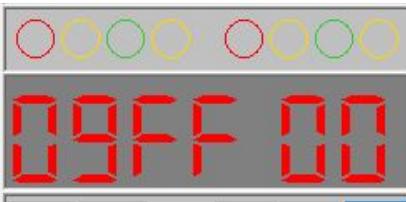
L1:
    MOV M,B
    INX H          ;next address
    DCR B          ;decrease the input until reaching zero
    JNZ L1         ;if it is not zero

    MOV M,B        ;adding zero
END
```

Για να ελέγξουμε αν η ζητούμενη λειτουργία έγινε σωστά, τρέχουμε το πρόγραμμα και στη συνέχεια με τη χρήση της ειδικής λειτουργίας **FETCH ADDRS** στο πρόγραμμα προσομοίωσης βλέπουμε αν έχει αποθηκευθεί το επιθυμητό αποτέλεσμα σε κάποιες ενδεικτικές διευθύνσεις μνήμης.

Για **FETCH ADDRS=0900H** (255), **FETCH ADDRS=090AH** (245), και **FETCH ADDRS=09FFH** (0) έχουμε τα σωστά αποτελέσματα.





(β) Ο κώδικας Assembly για την υλοποίηση φαίνεται παρακάτω (υλοποιήθηκε πρώτα το ερώτημα γ και μετά το β).

```

IN 10H

;;;;;;;;;;
;storing the numbers in memory
;counting the numbers between [20H,70H], result in C

LXI H,0900H      ;starting address
MVI A,FFH        ;(A)=255

MVI B,20H        ;(A)>=32=(20H)
MVI D,71H        ;(A)=<112=(70H)
MVI C,00H        ;C initialized to 0

LOOP1:
    MOV M,A

COMPR:
    CMP D
    JNC CONT1      ;if (A)>=71H => CY=0, CONTINUE
    CMP B
    JC CONT1      ;if (A)<20H => CY=1, CONTINUE
    INR C

CONT1:
    INX H          ;next address
    DCR A          ;decrease the input until reaching zero
    JNZ LOOP1      ;if it is not zero

    MOV M,A        ;adding zero

;;;;;;;;;;
;counting the number of zeros, result in registers D,E

LXI H,0900H      ;starting address
MVI D,00H
MVI E,00H        ;counter of zeros in D-E

MOV A,M

```

```

LOOP2: ;looping for all the addresses
    MVI B,08H           ;B is the counter, 8 times repetition

SUB_LOOP: ;looping for 8 times
    RRC                 ;right shift so that (CY)<-(A0)
    JC CONT2            ;if CY=1, continue
    INX D               ;if CY=0, augment the counter of zeros

CONT2:
    DCR B
    JNZ SUB_LOOP
    INX H               ;show to the next address
    MOV A,M              ;the next number to be processed
    CPI 00H              ;check if zero is reached
    JZ CONT3
    JMP LOOP2

CONT3: MVI B,08H

EXIT: ;when the result is zero, increase the D-E by 8zeros and exit
    INX D
    DCR B
    JNZ EXIT

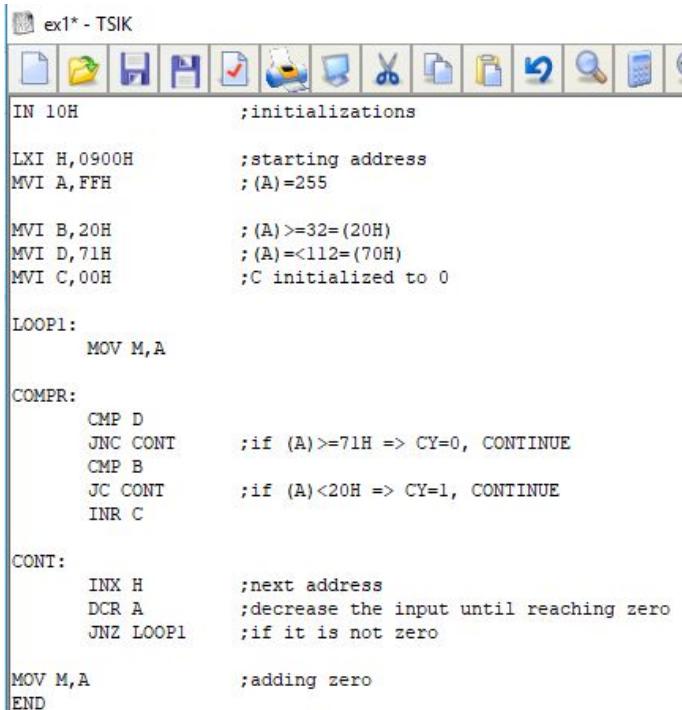
;;;;;;;;;;RESULTS;;;;;;;;;;
;D-E number of zeros
;C numbers in [20H,70H]

END

```

Το πλήθος των δυαδικών αναπαραστάσεων των αριθμών 255-0 είναι X=1024 όπως επαληθεύεται και από την εκτέλεση του προγράμματος εφόσον D=04H και E=0H.

(γ) Το πρόγραμμα για τον υπολογισμό του πλήθους των αριθμών $x_m \in [0,255]$ για τους οποίους ισχύει $20H \leq x_m \leq 70H$, προσδιορίστηκε επεκτείνοντας το προηγούμενο πρόγραμμα όπως φαίνεται στο σχήμα. Το αναμενόμενο αποτέλεσμα είναι $112-32+1=81=51H$, το οποίο μετά την εκτέλεση του προγράμματος αποθηκεύτηκε στον καταχωρητή C.



```

ex1* - TSIK
IN 10H          ;initializations

LXI H,0900H      ;starting address
MVI A,FFH        ;(A)=255

MVI B,20H        ;(A)>=32=(20H)
MVI D,71H        ;(A)=<112=(70H)
MVI C,00H        ;C initialized to 0

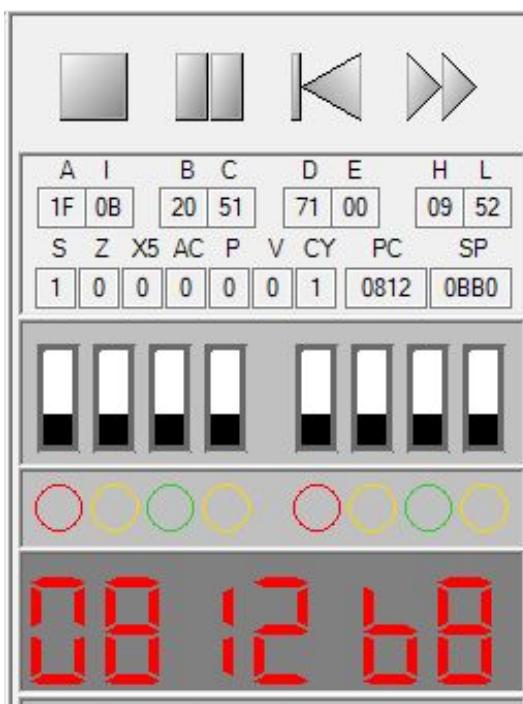
LOOP1:
    MOV M,A

COMPR:
    CMP D
    JNC CONT      ;if (A)>=71H => CY=0, CONTINUE
    CMP B
    JC CONT       ;if (A)<20H => CY=1, CONTINUE
    INR C

CONT:
    INX H          ;next address
    DCR A          ;decrease the input until reaching zero
    JNZ LOOP1      ;if it is not zero

MOV M,A          ;adding zero
END

```



(δ) Το τελικό πρόγραμμα που επιτελεί όλες τις παραπάνω λειτουργίες είναι:

```

IN 10H

;;;;;;;;;;;;;;
;storing the numbers in memory
;counting the numbers between [20H,70H], result in C

LXI H,0900H      ;starting address
MVI A,FFH        ;(A)=255

MVI B,20H        ;(A)>=32=(20H)
MVI D,71H        ;(A)=<112=(70H)
MVI C,00H        ;C initialized to 0

LOOP1:
    MOV M,A

COMPR:
    CMP D
    JNC CONT1      ;if (A)>=71H => CY=0, CONTINUE
    CMP B
    JC CONT1       ;if (A)<20H => CY=1, CONTINUE
    INR C

CONT1:
    INX H          ;next address
    DCR A          ;decrease the input until reaching zero
    JNZ LOOP1      ;if it is not zero

    MOV M,A        ;adding zero

;;;;;;;;;;;;;;
;counting the number of zeros, result in registers D,E

LXI H,0900H      ;starting address
MVI D,00H        ;counter of zeros in D-E
MVI E,00H

MOV A,M

```

```

LOOP2: ;looping for all the addresses
    MVI B,08H           ;B is the counter, 8 times repetition

SUB_LOOP: ;looping for 8 times
    RRC                 ;right shift so that (CY)<-(A0)
    JC CONT2            ;if CY=1, continue
    INX D               ;if CY=0, augment the counter of zeros

CONT2:
    DCR B
    JNZ SUB_LOOP
    INX H               ;show to the next address
    MOV A,M             ;the next number to be processed
    CPI 00H              ;check if zero is reached
    JZ CONT3
    JMP LOOP2

CONT3: MVI B,08H

EXIT: ;when the result is zero, increase the D-E by 8zeros and exit
    INX D
    DCR B
    JNZ EXIT

;;;;;;;;;; RESULTS - OUTPUT ;;;;;;;;;;;;;;;
;D-E number of zeros
;C numbers in [20H,70H]

LDA 2000H           ;input from dip switches in register A
MVI B,06H           ;B is the counter, 6 times repetition

LOOP3: ;6 right shifts needed
    RRC
    DCR B
    JNZ LOOP3
    JNC OUT_E          ;if the 6th switch is LOW, check if the 7th is HIGH

OUT_C:
    MOV A,C
    JMP FINAL

OUT_E:
    RRC
    JNC OUT_D          ;if the 7th is LOW, check if the 8th is HIGH
    MOV A,E
    JMP FINAL

OUT_D:
    RRC                ;assume that there is always an input from dip switches
    MOV A,D
    JMP FINAL

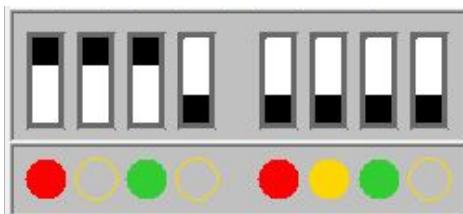
FINAL:
    STA 3000H
    END

```

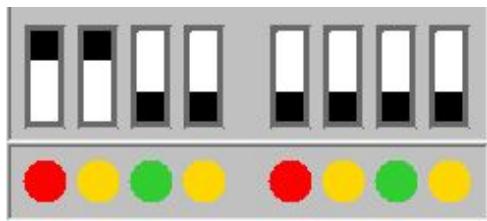
Για τους εισαχθέντες αριθμούς [0,255], όπως προκύπτει από τα παραπάνω ισχυεί **C=51H**, **E=0H** και **D=04H**.

Ελέγχοντας από τα LSB προς τα MSB τα dip switches με διαδοχικές εντολές RRC αν είναι HIGH:

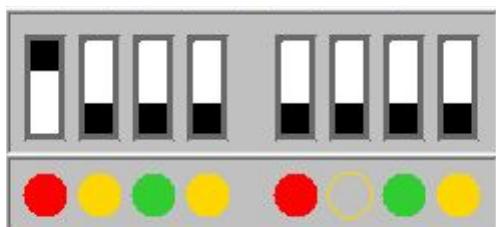
- Το **6o bit**, στα LEDs εμφανίζεται το περιεχόμενο του καταχωρητή **C**.



- Το **7o bit**, στα LEDs εμφανίζεται το περιεχόμενο του καταχωρητή **E**.



- Το **8o bit**, στα LEDs εμφανίζεται το περιεχόμενο του καταχωρητή **D**.



Αξίζει να σημειωθεί σε αυτό το σημείο ότι τα LEDs είναι **συνδεδεμένα με αρνητική λογική**, δηλαδή το λογικό 1 (HIGH) αντιστοιχεί σε σβηστό LED και το λογικό 0 (LOW) σε αναμμένο LED.

Άσκηση 2

Ο κώδικας που προκύπτει είναι ο εξής:

MVI B,00H	;BC =100 IN DECIMAL, SO 1/10 SEC TIME DELAY
MVI C,64H	;FOR DELB

START:

LDA 2000H	;READ INPUT
RAR	;CY=LSB
JC START	;IF LSB!=0 READ INPUT AGAIN

STEP2: ;LSB WAS 0 BEFORE

LDA 2000H	;READ INPUT
RAR	;CY=LSB
JNC STEP2	;IF LSB!=1 READ INPUT AGAIN

STEP3:

LDA 2000H	
RAR	
JC STEP3	;SAME OLD STUFF

GETREADYFORTHIS:

MVI E,96H ;E=150 IN DECIMAL
MVI D,00H ;USED AS COUNTER

ALLTHESIGNLELIGHTS:

CALL TICK_TOCK

MVI A,00H
STA 3000H ;LIGHT ALL THE LIGHTS

CALL TICK_TOCK ;EVERY TIME WE CHECK LSB WE HAVE CALLED
;TICK_TOCK TWICE, THUS $\frac{1}{2}$ SEC HAVE PASSED

MVI A,00H
CMA
STA 3000H ;TURN OFF ALL THE LIGHTS

MOV A,D
CPI 00H ;IF SWITCH WAS DOWN
JZ THING1 ;CHECK WHAT'S HAPPENING NOW
CPI 01H ;IF SWITCH WAS UP AFTER BEING DOWN
JZ THING2 ;CHECK WHAT'S HAPPENING NOW

THING1:

LDA 2000H
RAR
JNC ALLTHESIGNLELIGHTS ;SWITCH NOT ON YET
INR D ;SWITCH IS ON, D=1
JMP ALLTHESIGNLELIGHTS

THING2:

LDA 2000H
RAR
JC ALLTHESIGNLELIGHTS ;SWITCH NOT DOWN YET
JMP GETREADYFORTHIS ;SWITCH IS DOWN

TICK_TOCK:

CALL DELB
DCR E ; $150 * 1 / 10 = 15\text{sec}$
MOV A,E
CPI 00H ;CHECK IF TIME'S UP
JZ START ;IF YES START OVER WITH ALL LEDS OFF
RET

END

Τρέχοντας το πρόγραμμα με τον προσομοιωτή (αρχείο ex2.8085), έχοντας αντικαταστίσει την DELB με 3 NOP βλέπουμε τα εξής:

Αρχικά για να περάσει από START, STEP2 και STEP3 πρέπει αντίστοιχα ο LSB διακόπτης να μεταβεί από OFF σε ON και ξανά σε OFF, αλλιώς επαναλαμβάνει το ίδιο βήμα.

Αν περάσει από όλα τα παραπάνω στάδια, αρχικοποιεί τους καταχώρητες E και D στις τιμές 150 (που χρησιμοποιείται για να μετρήσει τα δευτερόλεπτα) και 0 αντίστοιχα.

The screenshot shows a Z80 emulator interface. On the left, the assembly code is displayed:

```

0800 06 MVI B,00H
0801 00
0802 0E MVI C,64H
0803 64

START:
0804 3A LDA 2000H
0805 00
0806 20
0807 1F RAR
0808 DA JC START
0809 04
080A 08

STEP2:
080B 3A LDA 2000H
080C 00
080D 20
080E 1F RAR
080F D2 JNC STEP2
0810 0B
0811 08

STEP3:
0812 3A LDA 2000H
0813 00
0814 20
0815 1F RAR
0816 DA JC STEP3
0817 12
0818 08

GETREADYFORTHIS:
0819 1E MVI E,96H
081A 96
081B 16 MVI D,00H
081C 00

```

On the right, the status window shows the following details:

- Registers:** A, I, B, C, D, E, H, L, S, Z, X5, AC, P, V, CY, PC, SP. Values: A=80, I=OB, B=00, C=64, D=00, E=96, H=00, L=00. S=0, Z=0, X5=0, AC=0, P=0, V=0, CY=0, PC=081B, SP=0B80.
- DIP Switches:** Eight switches labeled A through H, all in the off position.
- LEDs:** Four red LEDs labeled A, B, C, D, and four green LEDs labeled E, F, G, H.
- Display:** Shows the time as 08 16 16.
- Control Buttons:** RESET, RUN, HDWR STEP, INSTR STEP, INTRPT, FETCH PC, FETCH ADDRS, FETCH REG, DECR, STORE / INCR. The INSTR STEP button is highlighted.
- Memory Dump:** A grid showing memory locations from 0 to 3. Rows A, B, C, D show values 7, 8, 9; rows E, F, G, H show values 4, 5, 6.

Η συνάρτηση TICK_TOCK ουσιαστικά είναι ένα buffer 1/10 sec και καλείται 2 φορές πριν εξεταστεί το LSB των dip switches άρα έχουν περάσει $\frac{1}{5}$ sec. Καλείται 2 φορές, μία πριν ανάψουν όλα τα LEDs και μία πριν σβήσουν για να φαίνεται η αλλαγή (εφόσον εξομοιώνουμε φώτα ενός χώρου).

```

0820 3E MVI A,00H
0821 00
0822 32 STA 3000H
0823 00
0824 30
0825 CD CALL TICK_TOCK
0826 4F
0827 08
0828 3E MVI A,00H
0829 00
082A 2F CMA
082B 32 STA 3000H
082C 00
082D 30
082E 7A MOV A,D
082F FE CPI 00H
0830 00
0831 CA JZ THING1
0832 39
0833 08
0834 FE CPI 01H
0835 01

```

Συνολικά καλείται 150 φορές άρα σύνολο υπάρχει καθυστέρηση 15 sec.

To loop αυτό (ALLTHESINGLELIGHTS) συνολικά εκτελείται για 15sec και μετά γυρνάμε στην αρχή εκτός από την περίπτωση που ο διακόπτης του LSB μεταβεί από OFF ξανά σε ON και μετά OFF.

Δεδομένου πως ήταν OFF από πριν, αρχικά μεταβαίνει στην ετικέτα THING1 (εφόσον το D είναι ίσο με 0) και ελέγχει αν το LSB switch είναι ON. Αν είναι OFF ακόμη επιστρέφουμε στο tag ALLTHESINGLELIGHTS και αναβοσβήνουμε τα φώτα από εκεί που είχαμε μείνει. Άλλιώς κάνουμε το D=1 και επιστρέφουμε ξανά στο flag.

```

0835 01
0836 CA JZ THING2
0837 44
0838 08

THING1:
0839 3A LDA 2000H
083A 00
083B 20
083C 1F RAR
083D D2 JNC ALLTHESINGLELIGHTS
083E 1D
083F 08
0840 14 INR D
0841 C3 JMP ALLTHESINGLELIGHTS
0842 1D
0843 08

```

Τώρα με D=1 ξέρουμε πως ο LSB switch έχει μεταβεί ήδη από OFF σε ON οπότε αν μεταβεί ξανά σε OFF ανανεώνεται ο χρόνος των 15 sec.

Με D=1 στην επόμενη κλήση του ALLTHESINGLELIGHTS καλείται η THING2. Αν ο εν λόγω διακόπτης έχει παραμείνει ON επιστρέφουμε στην ALLTHESINGLELIGHTS και επαναλαμβάνουμε. Άλλιώς, επιστρέφουμε στο label GETREADYFORTHIS και αρχικοποιούμε ξανά τις τιμές D και E, επαναλαμβάνοντας όλη την παραπάνω διαδικασία ανανεώνοντας τον χρόνο των 15sec.

```

082B 32 STA 3000H
082C 00
082D 30
082E 7A MOV A,D
082F FE CPI 00H
0830 00
0831 CA JZ THING1
0832 39
0833 08
0834 FE CPI 01H
0835 01
0836 CA JZ THING2
0837 44
0838 08

THING1:
0839 3A LDA 2000H
083A 00
083B 20
083C 1F RAR
083D D2 JNC ALLTHESIGNLELIGHTS
083E 1D
083F 08
0840 14 INR D
0841 C3 JMP ALLTHESIGNLELIGHTS
0842 1D
0843 08

THING2:
0844 3A LDA 2000H
0845 00
0846 20
0847 1F RAR
0848 DA JC ALLTHESIGNLELIGHTS
0849 1D
084A 08
084B C3 JMP GETREADYFORTHIS
084C 19

```

The screenshot shows a Z80 assembly debugger interface. On the left is a memory dump window with the following assembly code:

```

082B 32 STA 3000H
082C 00
082D 30
082E 7A MOV A,D
082F FE CPI 00H
0830 00
0831 CA JZ THING1
0832 39
0833 08
0834 FE CPI 01H
0835 01
0836 CA JZ THING2
0837 44
0838 08

THING1:
0839 3A LDA 2000H
083A 00
083B 20
083C 1F RAR
083D D2 JNC ALLTHESIGNLELIGHTS
083E 1D
083F 08
0840 14 INR D
0841 C3 JMP ALLTHESIGNLELIGHTS
0842 1D
0843 08

THING2:
0844 3A LDA 2000H
0845 00
0846 20
0847 1F RAR
0848 DA JC ALLTHESIGNLELIGHTS
0849 1D
084A 08
084B C3 JMP GETREADYFORTHIS
084C 19

```

The right side of the interface contains several control and status elements:

- Top right: Control buttons for memory dump (square), registers (vertical bars), and navigation (left, right).
- Registers section (A-I, B-C, D-E, H-L):

A	I	B	C	D	E	H	L
00	0B	00	64	01	90	00	00
S	Z	X5	AC	P	V	CY	PC
0	1	0	0	1	0	0	084B
084B	OBBO						
- Memory dump section (8x8 grid of memory bytes):

1	0	0	1	0	0	084B	OBBO
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
- Control buttons (Reset, Run, Step, Fetch PC, Fetch Reg, Decr) and a numeric keypad (0-9) for address and value entry.
- LED indicators for various system states.
- Large digital display showing the value 084B C3.

```

GETREADYFORTHIS:
0819 1E MVI E,96H
081A 96
081B 16 MVI D,00H
081C 00

ALLTHESIGNLELIGHTS:
081D CD CALL TICK_TOCK
081E 4E
081F 08
0820 3E MVI A,00H
0821 00
0822 32 STA 3000H
0823 00
0824 30
0825 CD CALL TICK_TOCK
0826 4E

```

The screenshot shows a Z80 assembly debugger interface. On the left is a memory dump window with the following assembly code:

```

GETREADYFORTHIS:
0819 1E MVI E,96H
081A 96
081B 16 MVI D,00H
081C 00

ALLTHESIGNLELIGHTS:
081D CD CALL TICK_TOCK
081E 4E
081F 08
0820 3E MVI A,00H
0821 00
0822 32 STA 3000H
0823 00
0824 30
0825 CD CALL TICK_TOCK
0826 4E

```

The right side of the interface contains several control and status elements:

- Top right: Control buttons for memory dump (square), registers (vertical bars), and navigation (left, right).
- Registers section (A-I, B-C, D-E, H-L):

A	I	B	C	D	E	H	L
00	0B	00	64	01	90	00	00
S	Z	X5	AC	P	V	CY	PC
0	1	0	0	1	0	0	0819
0819	OBBO						
- Memory dump section (8x8 grid of memory bytes):

1	0	0	1	0	0	0819	OBBO
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
0	1	0	0	1	0		
- Control buttons (Reset, Run, Step, Fetch PC, Fetch Reg, Decr) and a numeric keypad (0-9) for address and value entry.
- LED indicators for various system states.

Άσκηση 3

Το πρόγραμμα για την υλοποίηση της άσκησης 3 είναι στο αρχείο ex3.

```
;initializations
MVI E,00H
MVI C,03H      ;AND_mask in the 2 LSB
MVI H,00H      ;OR_mask in the 2 LSB
MVI D,02H      ;counter of the main loop

LDA 2000H      ;input from dip switches in register A
```

L1:

```
;CALCULATING THE OR GATE

MOV B,A          ;storing in B before the changes, changes in A
ANA C            ;keeping only the last 2 digits
CMP H            ;compare with the OR_mask
JNZ FORM1_OR    ;OR output 0 only if
MVI A,00H        ;both of the inputs are 0
JMP FORM0_OR
```

FORM1_OR:

```
MVI A,10H
```

FORM0_OR:

```
ORA E            ;adding the previous results in A
RRC
MOV E,A
MOV A,B
RRC
RRC              ;calculations in the next 2 LSB
```

```
;CALCULATING THE AND GATE
```

```
MOV B,A
ANA C            ;keeping only the last 2 digits
CMP C            ;compare with the AND_mask
JZ FORM1_AND    ;AND output 1 only if
MVI A,00H        ;both of the inputs are 1
JMP FORM0_AND
```

FORM1_AND:

```
MVI A,10H
```

FORM0_AND:

```
ORA E
```

```
RRC  
MOV E,A  
  
DCR D  
JZ CONTINUE  
MOV A,B  
RRC  
RRC  
JMP L1 ;repeating the loop twice
```

CONTINUE:

```
MVI H,01H ;XOR_mask1  
MVI L,02H ;XOR_mask2  
MVI E,00H  
MVI C,0CH ;mask to keep bits X3,X2 0000 1100  
  
;CALCULATING THE XOR GATE  
MOV B,A ;store the result so far in register B  
ANI 07H ;0000 0111, A3<-0  
MOV E,A  
MOV A,B ;calculations in register A  
  
ANA C ;keeping only the last 2 digits  
RRC  
RRC  
  
CMP H ;compare with XOR_mask1  
JZ FORM1_XOR  
CMP L ;compare with XOR_mask2  
JZ FORM1_XOR ;XOR output 1 only if only one of the inputs is 1  
  
MVI A,00H  
JMP FORM0_XOR
```

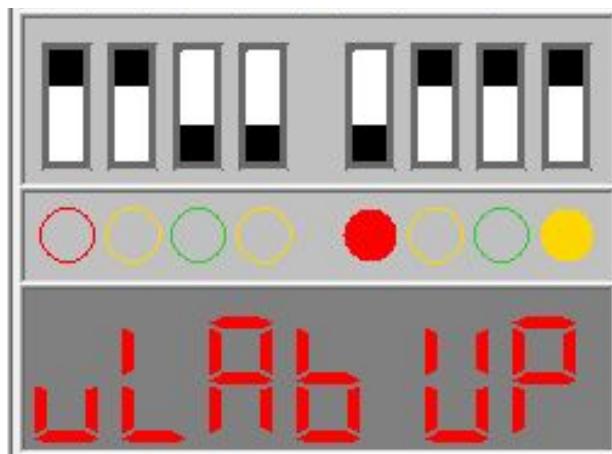
FORM1_XOR: MVI A,08H

```
FORM0_XOR:  
ORA E ;final result in register A  
  
;negative logic of the LEDs  
CMA ;bitwise complement of A  
;logic '1'-> ON LED  
;logic '0'-> OFF LED  
  
STA 3000H ;output in LEDS from register A  
END
```

Παράδειγμα προσομοίωσης:

Για είσοδο στα dip switches: **1100 0111** η αναμενόμενη έξοδος είναι 1001 και άρα τα 4 MSB των LED πρέπει να είναι OFF (αδιάφορες θέσεις εξόδου) και τα 4 LSB από αριστερά προς τα δεξιά **ON - OFF - OFF - ON**.

Όπως φαίνεται στην εικόνα, το αποτέλεσμα είναι το επιθυμητό.



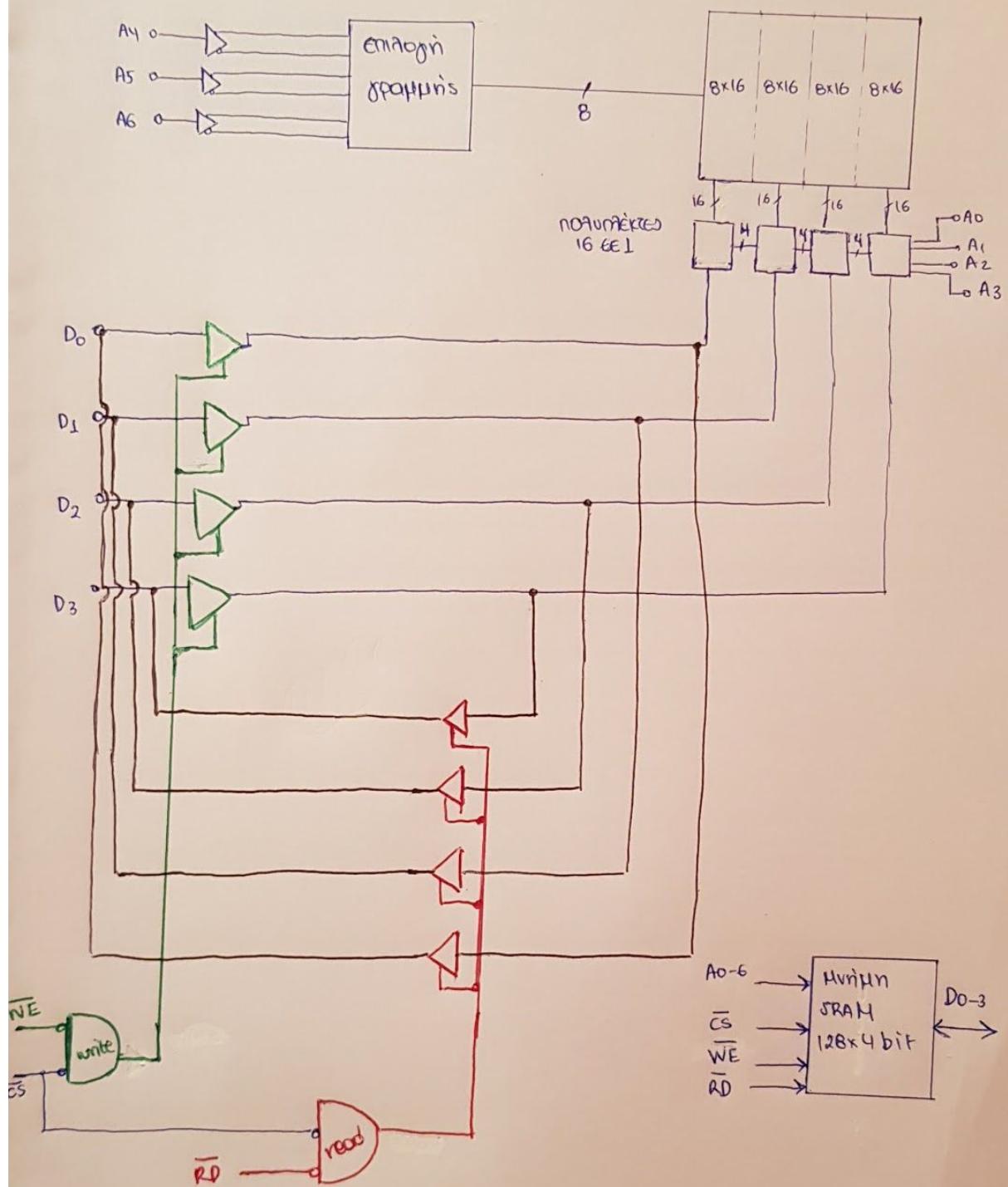
Άσκηση 4

SRAM

Μνήμη SRAM : 128x4 bit

Διεύθυνση A6-A0 = 7 bit

Λέπτη μήκους 4 bit (D3-D0)



Περιγραφή λειτουργίας

Η τετραγωνική διάταξη της μνήμης SRAM αποτελείται από 8x64 στοιχεία μνήμης. Τα bits A4-A6 της διεύθυνσης επιλέγεται μία από τις 8 γραμμές της διάταξης. Η επιλογή της στήλης από κάθε επιμέρους τετράδα στις οποίες χωρίζεται καθέτως η διάταξη, γίνεται μέσω των πολυπλεκτών 16 σε 1 με τα bit A0-A3 της διεύθυνσης. Οι πολυπλέκτες υλοποιούνται με διακόπτες και επιτρέπουν τη διέλευση των δεδομένων και προς τις δύο κατευθύνσεις.

Το δεδομένο που διαβάστηκε από τη μνήμη (έξοδος) ή αυτό που πρόκειται να εγγραφεί (είσοδος) αποτελείται από τα 4 bit D0-D3.

Όταν το σήμα **CS** (συμπλήρωμα) είναι 1, τότε έχουμε απομόνωση της εισόδου από την έξοδο.

Όταν το σήμα **CS** είναι 0, τότε έχουμε εγγραφή ή ανάγνωση στη μνήμη, ανάλογα με τις τιμές των σημάτων **WE** και **RD**. Συγκεκριμένα, όταν **WE**=0 έχουμε εγγραφή και όταν **RD**=0, έχουμε ανάγνωση. Θεωρούμε ότι τα σήματα **WE** και **RD** δεν παίρνουν ταυτόχρονα τις τιμές 0 και 1.

Για παράδειγμα, έστω ότι θέλουμε να διαβάσουμε από τη μνήμη, οπότε εισάγουμε την επιθυμητή διεύθυνση στην οποία είναι αποθηκευμένο το δεδομένο στα bits A0-A6. Τότε **CS**=0, **RD**=0 και **WE**=1.

Με τα bits A4-A6 επιλέγεται η γραμμή της διάταξης στην οποία είναι αποθηκευμένο το δεδομένο και με τα bits A0-A3 η στήλη. Τα επιθυμητά bits εμφανίζονται στις εξόδους των πολυπλεκτών και επειδή η έξοδος της πύλης AND_write είναι 0, ενώ της AND_read είναι 1, τα bits εμφανίζονται στις εξόδους D0-D3 μέσω των τρισταθών buffers ανάγνωσης (οι τρισταθείς buffers εγγραφής είναι σε αποκοπή).

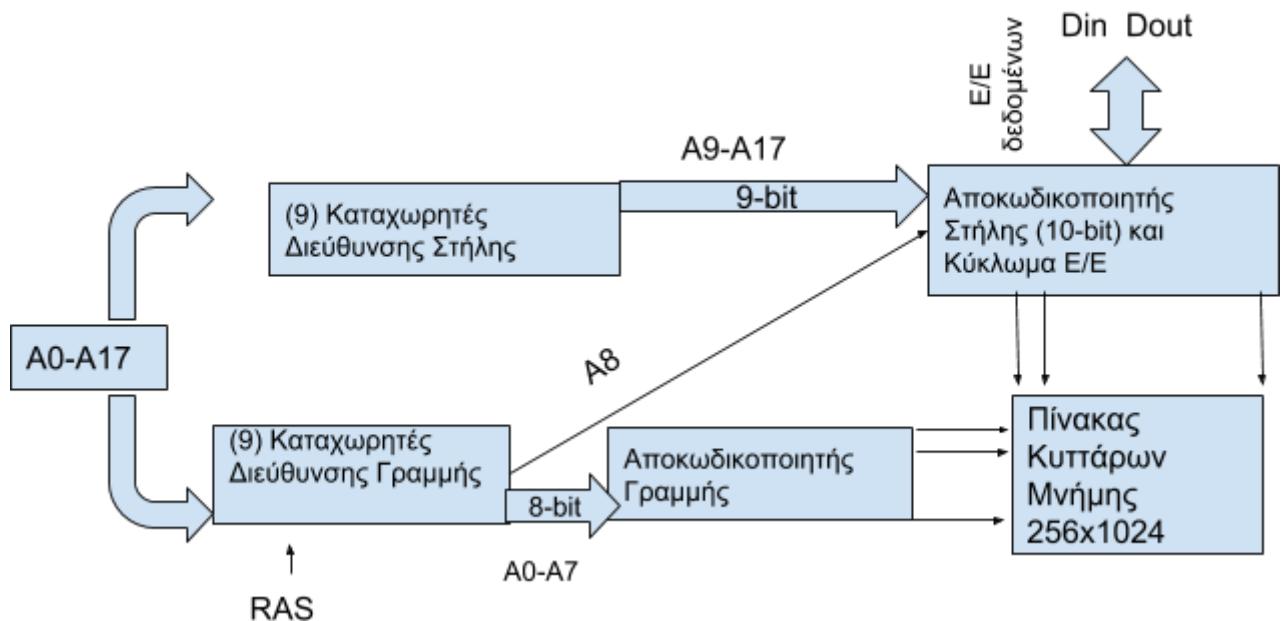
DRAM

Η μνήμη έχει μέγεθος $256K = 2^{18}$ bits. Άρα, το μήκος μιας διεύθυνσης είναι 19 bits. Από αυτά, τα 9 είναι (A9 - A17) χρησιμοποιούνται για τον προσδιορισμό της διεύθυνσης στήλης (CAS - Column Address). Τα υπόλοιπα 9 (A0 - A8) χρησιμοποιούνται για το προσδιορισμό γραμμής της μνήμης (RAS - Row Address).

Βέβαια, το bit A8 στέλνεται στον αποκωδικοποιητή στήλης για την επιτέλεση της προφανούς λειτουργίας. Άρα η διεύθυνση στήλης έχεις μήκος 10 bits και άρα έχουμε $2^{10} = 1024$ στήλες ενώ η διεύθυνση γραμμής έχει 8 bits και άρα $2^8 = 256$ γραμμές.

Αρχικά, μεταφέρονται τα 8 λιγότερα σημαντικά ψηφία και κρατούνται στη μνήμη με το σήμα RAS. Στη συνέχεια μεταφέρονται τα υπόλοιπα που συγκρατούνται συνολικά από τον παλμό CAS. Η εγγραφή της μνήμης γίνεται στο αρνητικό μέτωπο του παλμού W ενώ το G ισούται με 1. Η ανάγνωση της μνήμης γίνεται στο αρνητικό μέτωπο του παλμού G ενώ το W ισούται με 1.

Αυτά φαίνονται και από το παρακάτω σχήμα.



Άσκηση 5

Στην παρούσα άσκηση έχουμε ROM μεγέθους $4\text{K} \times 8 \text{ bits}$ που αποτελείται από 2 ολοκληρωμένα μνήμης $2\text{K} \times 8 \text{ bits}$, ενώ ξεκινάει από την διεύθυνση 0000H .

Ο χάρτης μνήμης που προκύπτει για την ROM είναι ο εξής:

Αρχή Διευθύνσεων			
BIN	0000	0000	0000 0000
HEX	0	0	0 0
Τέλος Διευθύνσεων			
BIN	0000	1111	1111 1111
HEX	0	F	F F

Συγκεκριμένα για τα δύο ολοκληρωμένα μνήμης ROM1 και ROM2 έχουμε:

ROM1:

Αρχή Διευθύνσεων			
BIN	0000	0000	0000 0000
HEX	0	0	0 0
Τέλος Διευθύνσεων			
BIN	0000	0111	1111 1111
HEX	0	7	F F

ROM2:

Αρχή Διευθύνσεων
BIN 0000 1000 0000 0000
HEX 0 8 0 0
Τέλος Διευθύνσεων
BIN 0000 1111 1111 1111
HEX 0 F F F

Η RAM έχει μέγεθος 10Kbytes (=4K * 8b bits) και υλοποιείται από 2 SRAM 2Kbytes και 8Kbytes.

Χάρτης μνήμης SRAM:

Αρχή Διευθύνσεων
BIN 0001 0000 0000 0000
HEX 1 0 0 0
Τέλος Διευθύνσεων
BIN 0011 0111 1111 1111
HEX 3 7 F F

Ενώ ο χάρτης διευθύνσεων των 2 SRAMs έχει ως εξής:

SRAM1(2 Kbytes):

Αρχή Διευθύνσεων
BIN 0001 0000 0000 0000
HEX 1 0 0 0
Τέλος Διευθύνσεων
BIN 0001 0111 1111 1111
HEX 1 7 F F

SRAM2(8 Kbytes):

Αρχή Διευθύνσεων
BIN 0001 1000 0000 0000
HEX 1 8 0 0
Τέλος Διευθύνσεων
BIN 0011 0111 1111 1111
HEX 3 7 F F

Τα ψηφία A15, A14 χρησιμεύουν ως είσοδοι επίτρεψης του αποκαδικοποιητή και τα ψηφία A11-A13 χρειάζονται για την επιλογή της κατάλληλης μνήμης.

Συγκεκριμένα, όταν $A_{13}=0$ & $A_{12}=0$ & $A_{11}=0$ επιλέγεται η ROM1, όταν $A_{13}=0$ & $A_{12}=0$ & $A_{11}=1$ η ROM2, όταν $A_{13}=0$ & $A_{12}=1$ & $A_{11}=0$ ή SRAM1 και για κάθε άλλη περίπτωση (πέρα από το να είναι όλα μονάδες) η SRAM2.

Άρα, έστω πως S_0, S_1, S_2 επιλογείς του αποκωδικοποιητή. Ο πίνακας αληθείας έχει ως εξής:

A_{13}	A_{12}	A_{11}		S_2	S_1	S_0
0	0	0		0	0	0
0	0	1		0	0	1
0	1	0		0	1	0
0	1	1		0	1	1
1	0	0		0	1	1
1	0	1		0	1	1
1	1	0		0	1	1
1	1	1		X	X	X

Για να αποφύγουμε απαγορευμένη πρόσβαση σε μνήμη μπορούμε να θέσουμε $S_2=0$.

Απόρροια όλων των παραπάνω είναι τα παρακάτω σχήματα:

