Συστήματα Μικροϋπολογιστών

Απαντήσεις των ασκήσεων στην 2η ομάδα ασκήσεων

4 ασκήσεις που ακολουθούν είναι ασκήσεις προσομοίωσης και εν συνεχεία, ακολουθούν θεωρητικές ασκήσεις και προσομοιώσεις / κυκλώματα με την χρήση του προσομοιωτή σχημάτων diagrams.net, logisim και ενισχυτικά το MS Paint των Windows

ΦΟΙΤΗΤΕΣ-ΕΠΙΜΕΛΕΙΑ: Αλκιβιάδης Μιχαλίτσης και Βόικος Στέφανος

4/5/2021

Στοιχεία φοιτητών

Ονοματεπώνυμα: Αλκιβιάδης Μιχαλίτσης και Στέφανος Βόικος

Α.Μ: el18868 και el18162 Ακαδημαϊκό εξάμηνο: 6ο

Σχολή: Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (Η.Μ.Μ.Υ)

ΑΠΑΝΤΗΣΕΙΣ ΤΩΝ ΑΣΚΗΣΕΩΝ ΤΗΣ ΟΜΑΔΑΣ 2 ΣΤΟ ΜΑΘΗΜΑ «ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ»

1η ΑΣΚΗΣΗ:

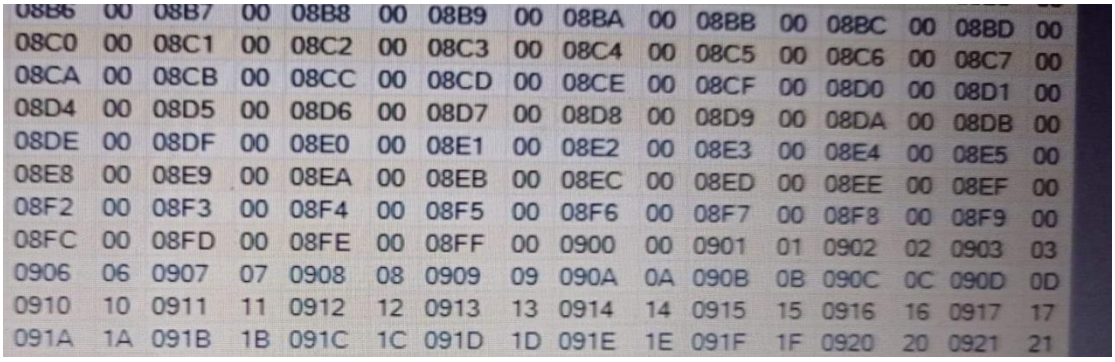
Παρακάτω ακολουθεί ο κώδικας του προγράμματος καθώς και ένα στιγμιότυπο όπου φαίνονται οι αποθηκεύσεις των ζητούμενων αριθμών:

```
        IN 10H
        MVI A,00H       ;NUMBERS(0<=A<=255)
        LXI H,0900H     ;ADRESSES
        LXI B,0000H     ;MONADES
        MVI D,00H       ;NUMBERS [10H,60H]
START:
        MOV M,A         ;SAVE AT MEMORY
        MOV E,A         ;RECENT SAVE
        JMP ONES
COUNTONES:
        MOV A,E
        JMP NUMBERS
COUNTNUM:
        INX H           ;NEXT MEMORY LOCATION
        INR A           ;NEXT NUMBER
        CPI 00H         ;A=00H ?
        JZ FINISH       ;FINISH CHECK
        JMP START

ONES:                   ;MONDES COUNTING (BINARY 1)
        STC             ;CY=0
        CMC             ;
        RAR             ;
        JNC ONE         ;MONADA
        INX B
ONE:
        CPI 00H;A=00H ?
        JZ COUNTONES            ;FINISH CHECK□
        JMP ONES

NUMBERS:                ;NUMBER COUNTING AT [10H,60H]
        CPI 10H
        JC COUNTNUM ;A<10H ?
```

```
        CPI 60H
        JC NUMBER      ;A<60H ?
        JNZ COUNTNUM           ;A=60H ?
NUMBER:
        INR D
        JMP COUNTNUM
FINISH:
        END
```

08FA	00	08FB	00	08FC	00	08FD	00	08FE	00	08FF	00	0900	00	0901	00	0902	00	0903	00
0904	00	0905	00	0906	00	0907	00	0908	00	0909	00	090A	00	090B	00	090C	00	090D	00
090E	00	090F	00	0910	00	0911	00	0912	00	0913	00	0914	00	0915	00	0916	00	0917	00
0918	00	0919	00	091A	00	091B	00	091C	00	091D	00	091E	00	091F	00	0920	00	0921	00
0922	00	0923	00	0924	00	0925	00	0926	00	0927	00	0928	00	0929	00	092A	00	092B	00
092C	00	092D	00	092E	00	092F	00	0930	00	0931	00	0932	00	0933	00	0934	00	0935	00
0936	00	0937	00	0938	00	0939	00	093A	00	093B	00	093C	00	093D	00	093E	00	093F	00
0940	00	0941	00	0942	00	0943	00	0944	00	0945	00	0946	00	0947	00	0948	00	0949	00
094A	00	094B	00	094C	00	094D	00	094E	00	094F	00	0950	00	0951	00	0952	00	0953	00
0954	00	0955	00	0956	00	0957	00	0958	00	0959	00	095A	00	095B	00	095C	00	095D	00
095E	00	095F	00	0960	00	0961	00	0962	00	0963	00	0964	00	0965	00	0966	00	0967	00
0968	00	0969	00	096A	00	096B	00	096C	00	096D	00	096E	00	096F	00	0970	00	0971	00
0972	00	0973	00	0974	00	0975	00	0976	00	0977	00	0978	00	0979	00	097A	00	097B	00
097C	00	097D	00	097E	00	097F	00	0980	00	0981	00	0982	00	0983	00	0984	00	0985	00
0986	00	0987	00	0988	00	0989	00	098A	00	098B	00	098C	00	098D	00	098E	00	098F	00
0990	00	0991	00	0992	00	0993	00	0994	00	0995	00	0996	00	0997	00	0998	00	0999	00
099A	00	099B	00	099C	00	099D	00	099E	00	099F	00	09A0	00	09A1	00	09A2	00	09A3	00
09A4	00	09A5	00	09A6	00	09A7	00	09A8	00	09A9	00	09AA	00	09AB	00	09AC	00	09AD	00
09AE	00	09AF	00	09B0	00	09B1	00	09B2	00	09B3	00	09B4	00	09B5	00	09B6	00	09B7	00
09B8	00	09B9	00	09BA	00	09BB	00	09BC	00	09BD	00	09BE	00	09BF	00	09C0	00	09C1	00
09C2	00	09C3	00	09C4	00	09C5	00	09C6	00	09C7	00	09C8	00	09C9	00	09CA	00	09CB	00
09CC	00	09CD	00	09CE	00	09CF	00	09D0	00	09D1	00	09D2	00	09D3	00	09D4	00	09D5	00
09D6	00	09D7	00	09D8	00	09D9	00	09DA	00	09DB	00	09DC	00	09DD	00	09DE	00	09DF	00
09E0	00	09E1	00	09E2	00	09E3	00	09E4	00	09E5	00	09E6	00	09E7	00	09E8	00	09E9	00
09EA	00	09EB	00	09EC	00	09ED	00	09EE	00	09EF	00	09F0	00	09F1	00	09F2	00	09F3	00
09F4	00	09F5	00	09F6	00	09F7	00	09F8	00	09F9	00	09FA	00	09FB	00	09FC	00	09FD	00
09FE	00	09FF	00	0A00	00	0A01	00	0A02	00	0A03	00	0A04	00	0A05	00	0A06	00	0A07	00
0A08	00	0A09	00	0A0A	00	0A0B	00	0A0C	00	0A0D	00	0A0E	00	0A0F	00	0A10	00	0A11	00

Τρέχουμε αρκετά steps και παράγουμε όντως τα νούμερα από 0 έως 255 σε δεκαεξαδικό σύστημα. Η αρχή του αποτελεί η παρακάτω εικόνα.



08B6	00	08B7	00	08B8	00	08B9	00	08BA	00	08BB	00	08BC	00	08BD	00				
08C0	00	08C1	00	08C2	00	08C3	00	08C4	00	08C5	00	08C6	00	08C7	00				
08CA	00	08CB	00	08CC	00	08CD	00	08CE	00	08CF	00	08D0	00	08D1	00				
08D4	00	08D5	00	08D6	00	08D7	00	08D8	00	08D9	00	08DA	00	08DB	00				
08DE	00	08DF	00	08E0	00	08E1	00	08E2	00	08E3	00	08E4	00	08E5	00				
08E8	00	08E9	00	08EA	00	08EB	00	08EC	00	08ED	00	08EE	00	08EF	00				
08F2	00	08F3	00	08F4	00	08F5	00	08F6	00	08F7	00	08F8	00	08F9	00				
08FC	00	08FD	00	08FE	00	08FF	00	0900	00	0901	01	0902	02	0903	03				
0906	06	0907	07	0908	08	0909	09	090A	0A	090B	0B	090C	0C	090D	0D				
0910	10	0911	11	0912	12	0913	13	0914	14	0915	15	0916	16	0917	17				
091A	1A	091B	1B	091C	1C	091D	1D	091E	1E	091F	1F	0920	20	0921	21				

(β) Για να υπολογίσουμε το πλήθος των μηδενικών χρησιμοποιήσαμε δεξιές περιστροφές και έλεγχο για την τιμή του CY, 8 φορές για κάθε αριθμό. Για να ελέγξουμε την ορθότητα του αποτελέσματος του πλήθους των μηδενικών, βάζουμε

το πρόγραμμα να τυπώσει τον D και τον E στα LED εξόδου (αυτό το κομμάτι του κώδικα έχει αντικατασταθεί με σχόλια και υπάρχει μόνο για έλεγχο). Βλέπουμε ότι ο D είναι ίσος με 00000100 και ο E ίσος με 00000000, το οποίο είναι σωστό, καθώς 0400 HEX = 1024, το οποίο είναι όντως το πλήθος των μηδενικών των αριθμών 0-255 (2048 ψηφία / 2 = 1024).

(γ) Οι αριθμοί αυτοί συνολικά είναι 81 = 51 HEX, άρα αναμένουμε ότι η σωστή τιμή του C είναι 51. Πράγματι, ενεργοποιώντας τις σχολιασμένες εντολές στο τέλος του προγράμματος, τυπώνεται στα LED εξόδου ο αριθμός 51, επομένως ο κώδικάς μας είναι σωστός.

2η ΑΣΚΗΣΗ:

Παρακάτω ακολουθεί ο κώδικας του προγράμματος:

```
        MVI A,FFH
        STA 3000H
        MVI D,64H           ;20 Seconds Counter(100*0.2) 100=64H
        LXI B,0064H         ;wait 100 ms
START:
        LDA 2000H           ;Latches status reading
        ANI 80H             ;MSB
        CPI 00H             ;OFF ?
        JZ OFF1
        JMP START
OFF1:                       ;FIRST OFF
        LDA 2000H
        ANI 80H
        CPI 80H         ;ON ?
        JZ ON1
        JMP OFF1
ON1:                        ;FIRST ON
        LDA 2000H
        ANI 80H
        CPI 00H             ;OFF ?
        JZ OFF2
        JMP ON1
OFF2:                       ;SECOND OFF
        LDA 2000H
        ANI 80H
        CPI 80H         ;ON ?
        JZ ON2
        MVI A,00H           ;OPEN
        STA 3000H           ;
        CALL DELB
        DCR D               ;TIME
        MOV A,D
        CPI 00H             ;TIME EXPIRATION ?
        JNZ OFF2            ;CHECK TIME EXPIRATION
        MVI A,FFH           ;CLOSE
        STA 3000H           ;
        MVI D,96H           ;RESET TIMER
        JMP OFF1
```

```
ON2:                    ;SECOND ON
        LDA 2000H
        ANI 80H
        CPI 00H         ;OFF ?
        JZ RESTART      ;CHECK TIME REFRESH
        MVI A,00H
        STA 3000H
        CALL DELB
        DCR D
        MOV A,D
        CPI 00H
        JNZ ON2
        MVI A,FFH
        STA 3000H
        MVI D,96H
        JMP OFF1
RESTART:
        MVI D,96H       ;RESET TIMER
        JMP OFF2

        END
```

3η ΑΣΚΗΣΗ:

3.1) Για να πραγματοποιήσουμε το ζητούμενο της άσκησης θα προσθέσουμε στο 0 δυνάμεις του 2, ξεκινώντας από το 27 και κατεβαίνοντας μέχρι τη θέση του αριστερότερου 1

Παρακάτω ακολουθεί ο κώδικας του προγράμματος:

```
START:
        LDA 2000H
        CPI 00H
        JZ ZERO         ;CHECK IF INPUT IS EQUAL TO ZERO, THEN PRINT ZERO
        MVI B,00H

ROTATE:
        RAL
        INR B           ;FINAL VALUE OF B = POSITION OF THE LEFTMOST 1
        JNC ROTATE

        MVI C,00H
        MVI D,80H       ;D IS 2^7 AND IN EACH LOOP IT WILL BE DIVIDED BY 2

ADDINGPOWERS:
        MOV A,B
        CPI 00H
        JZ FINISH
        MOV A,C
        ADD D           ;ADD POWER OF 2
        MOV C,A
        DCR B           ;(ADD POWER OF 2)*B
        MOV A,D
        RRC             ;DIVIDE D BY 2 --> NEXT POWER OF 2
        MOV D,A
```

```
        JMP ADDINGPOWERS

FINISH:
        MOV A,C

ZERO:
        CMA
        STA 3000H
        JMP START
        END
```

3.2)

Παρακάτω ακολουθεί ο κώδικας του προγράμματος:

```
START:
        MVI B,80H       ;LED
        CALL KIND       ;KEYBOARD READ
        CPI 00H;A=0?
        JZ TURNOFF      ;CHECK FOR ZERO
        CPI 09H;A=9?
        JNC TURNOFF  ;CHECK FOR KEY PRESSED >= 9
        CPI 01H;A=1?
        JZ ONE ;CHECK FOR ONE
        CPI 02H;A=2?
        JZ TWO          ;CHECK FOR TWO
        CPI 03H;A=3?
        JZ THREE        ;CHECK FOR THREE
        CPI 04H;A=4?
        JZ FOUR         ;CHECK FOR FOUR
        CPI 05H;A=5?
        JZ FIVE ;CHECK FOR FIVE
        CPI 06H;A=6?
        JZ SIX          ;CHECK FOR SIX
        CPI 07H;A=7?
        JZ SEVEN        ;CHECK FOR SEVEN
        CPI 08H;A=8?
        JZ EIGHT        ;CHECK FOR EIGHT
ONE:
        MVI B,FFH
        JZ SHOW
TWO:
        MVI B,FEH
        JZ SHOW
THREE:
        MVI B,FCH
        JZ SHOW
FOUR:
        MVI B,F8H
        JZ SHOW
FIVE:
        MVI B,F0H
        JZ SHOW
SIX:
        MVI B,E0H
        JZ SHOW
SEVEN:
        MVI B,C0H
        JZ SHOW
EIGHT:
```

```
        MVI B,80H
        JZ SHOW
TURNOFF:            ;LED OFF
        MVI B,00H   ;KEY PRESSED >=9 OR EQUAL TO ZERO
        JZ SHOW
SHOW:               ;LED ON
        MOV A,B
        CMA
        STA 3000H
        JMP START
        END
```

3.3) Κάθε φορά ελέγχουμε ποιο πλήκτρο είναι πατημένο. Αυτό επιτυγχάνεται ελέγχοντας ξεχωριστά την κάθε γραμμή του πληκτρολογίου και προσπαθώντας να βρει τη στήλη στην οποία βρίσκεται το πατημένο πλήκτρο. Έπειτα φορτώνουμε στη μνήμη τον κωδικό του, ο οποίος τυπώνεται στα 7-SGM DISPLAYS με τη βοήθεια των ρουτινών STDM και DCD. Σημειώνεται ότι το πρόγραμμα είναι συνεχούς λειτουργίας και επομένως τυπώνει τον κωδικό του κάθε πλήκτρου για όση ώρα αυτό παραμένει πατημένο. Επίσης, σημειώνεται ότι στην αρχή το πρόγραμμα τυπώνει τον κωδικό της εντολής run (84), μόλις αυτό πατιέται για την εκκίνηση του προγράμματος. Αν αυτό θέλαμε να το αποφύγουμε, θα μπορούσαμε να βάλουμε μια CALL DELB στην αρχή του προγράμματος έτσι ώστε να αγνοηθεί το αρχικό run.

4η ΑΣΚΗΣΗ:

Παρακάτω ακολουθεί ο κώδικας του προγράμματος:

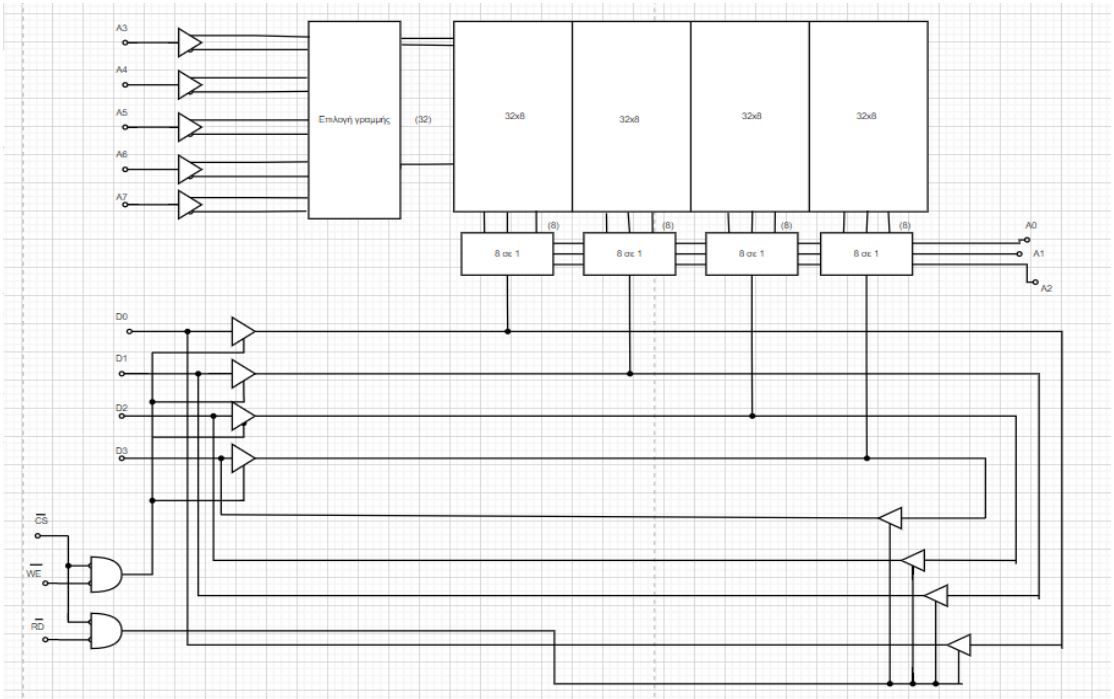
```
START:
       MVI D,00H     ;LEDS
       ;X3
       LDA 2000H     ;A3
       ANI 80H       ;10000000
       RRC           ;SLIDING TO THE NEXT DIGIT
       MOV B,A       ;RECENT SAVE
       LDA 2000H     ;B3
       ANI 40H       ;01000000
       ANA B         ;A3 AND B3
       RRC
       RRC
       MOV E,A       ;RECENT SAVE
       RLC
       RRC
       RRC
       MOV D,A       ;RECENT SAVE
       ;X2
       LDA 2000H     ;A2
       ANI 20H       ;00100000
       RRC
       MOV B,A
       LDA 2000H     ;B2
       ANI 10H       ;00010000
       ANA B         ;A2 AND B2
       ORA E         ;(A2 AND B2)OR X3
       RRC
       RRC
```

```
ORA D          ;ADD
MOV D,A
;X1
LDA 2000H      ;A1
ANI 08H        ;00001000
RRC
MOV B,A
LDA 2000H      ;B1
ANI 04H        ;00000100
ORA B          ;A1 OR B1
RRC
RRC
MOV C,A        ;RECENT SAVE
RLC
ORA D
MOV D,A
;X0
LDA 2000H      ;A0
ANI 02H        ;00000010
RRC
MOV B,A
LDA 2000H      ;B0
ANI 01H        ;00000001
ORA B          ;A0 OR B0
XRA C          ;(A0 OR B0) XOR X1
ORA D
CMA
STA 3000H
JMP START
END
```

Χρησιμοποιώντας εντολές ANI ώστε να απομονώνουμε τα ψηφία που θέλουμε σε κάθε περίπτωση, αφού τα έχουμε φέρει στην επιθυμητή θέση με εντολές RLC, RRC. Με τις εντολές ORA, ANA, XRA εκτελούμε τις κατάλληλες λογικές πράξεις μεταξύ των ψηφίων και προσθέτουμε τα επιμέρους αποτελέσματα με την εντολή ADD για να παραχθεί το τελικό αποτέλεσμα.

5η ΑΣΚΗΣΗ:

Η εσωτερική οργάνωση της μνήμης SRAM με τα χαρακτηριστικά που περιγράφονται στην εκφώνηση της άσκησης, αποτελεί η παρακάτω:



Για να γίνει εγγραφή στη μνήμη πρέπει να είναι ενεργοποιημένα (δηλαδή 0, αφού είναι αντεστραμμένα με άρνηση NOT) τα σήματα CS’ και WE’. Σε αυτή την περίπτωση γίνεται 1 η έξοδος της πάνω πύλης AND, η οποία δίνει έξοδο λογικό 1 στα τέσσερα αριστερά τρισταθή buffers. Έτσι, τα D0, D1, D2, D3 πηγαίνουν στους τέσσερις πολυπλέκτες (που σε αυτή την περίπτωση λειτουργούν ως αποπλέκτες). Τέλος, τα σήματα A0, A1, A2 ως επιλογείς των πολυπλεκτών καθορίζουν σε ποια εκ των 8 στηλών του κάθε μπλοκ θα γίνει η εγγραφή, και τα σήματα A3, A4, A5, A6 και A7 επιλέγουν μέσω ενός αποκωδικοποιητή μία από τις 32 γραμμές. Έτσι, τα D0, D1, D2, D3 γράφονται όλα ταυτόχρονα σε τέσσερα κελιά της μνήμης (ένα σε κάθε μπλοκ). Για να γίνει ανάγνωση από τη μνήμη πρέπει να είναι ενεργοποιημένα (δηλαδή 0, αφού είναι αντεστραμμένα) τα σήματα CS’ και RD’. Σε αυτή την περίπτωση γίνεται 1 η έξοδος της κάτω πύλης AND, η οποία δίνει ως έξοδο λογικό 1 στα τέσσερα δεξιά τρισταθή buffers. Έτσι, οι έξοδοι των τεσσάρων πολυπλεκτών πηγαίνουν στα D0, D1, D2, D3. Ο κάθε πολυπλέκτης διαβάζει ένα bit του αντίστοιχου μπλοκ, του οποίου η στήλη καθορίζεται από τους επιλογείς A0, A1, A2 και η γραμμή από τα A3, A4, A5, A6. Έτσι, τα τέσσερα bits (ένα από το κάθε μπλοκ) που επιλέχθηκαν από τη μνήμη περνιούνται στα D0, D1, D2, D3. Επιβεβαιώνω ότι πρόκειται για 256 bit το καθένα «κομμάτι» καθώς ισχύει το γινόμενο 32 * 8 = 256.Θα μπορούσε πάντως να γίνει και με κάποιον άλλον συνδυασμό όπως 16*16, δηλαδή να διαθέτω 16 γραμμές ως έξοδο από το τμήμα επιλογέα γραμμής και αντί για χρήση πολυπλεκτών που βγάζουν 8 bit, θα χρησιμοποιήσουμε πολυπλέκτες που θα δέχονται ως είσοδο 4 bit και θα βγάζουν 16 επιλογές.

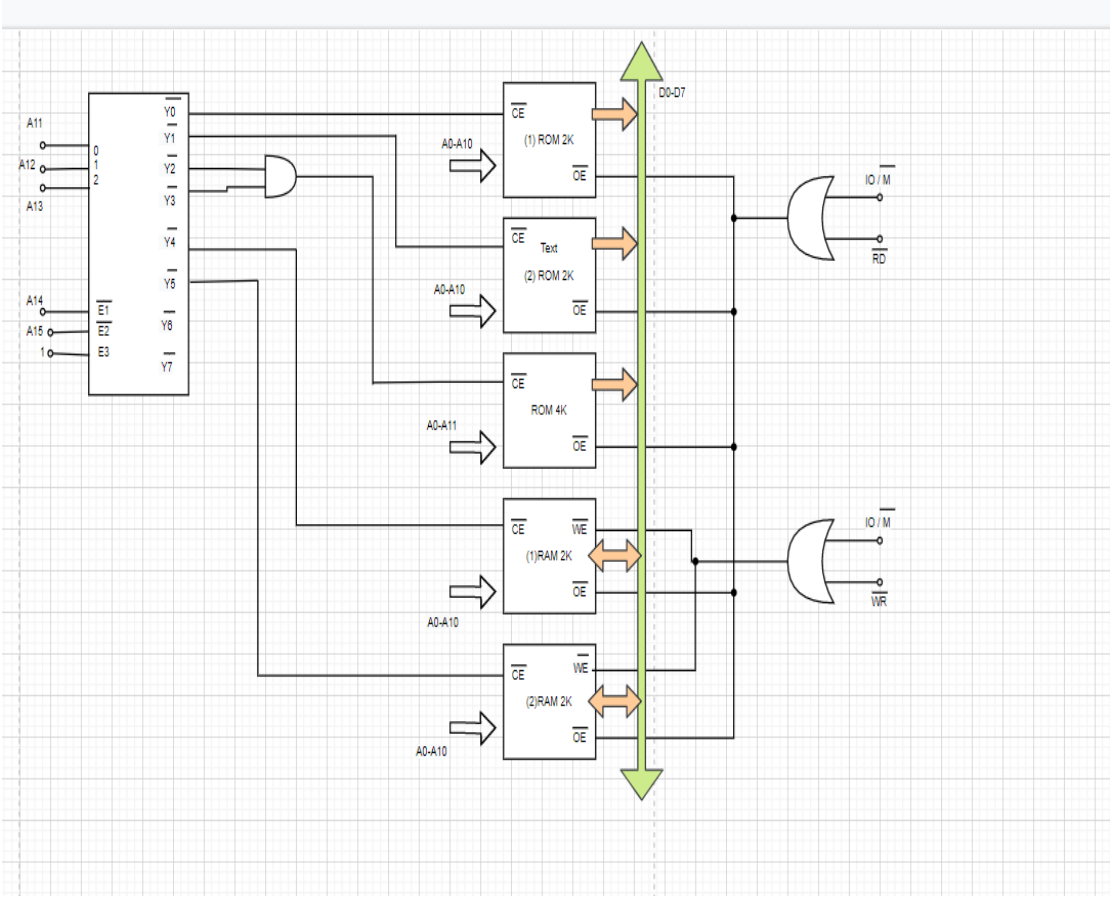
6η ΑΣΚΗΣΗ:

Για το εν λόγω πρόβλημα, ο χάρτης μνήμης που παράγεται με βάση τα ζητούμενα της εκφώνησης και τις ιδιότητες του κυκλώματος που θα σχεδιάσουμε και είναι:

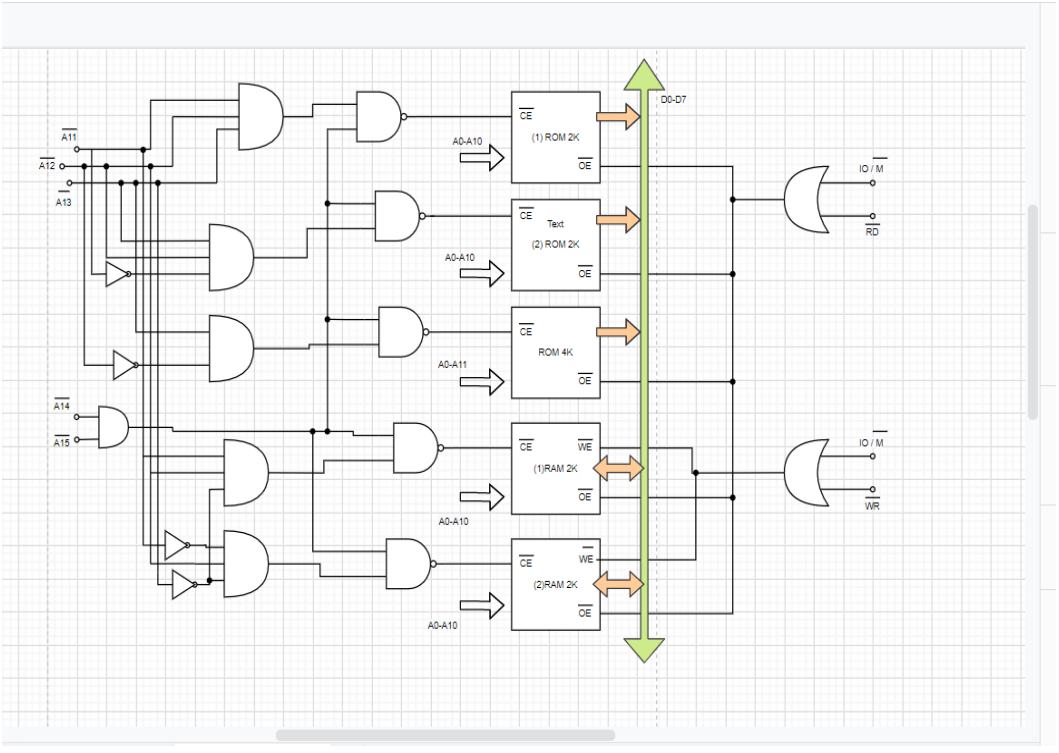
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Memory
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	(1) ROM 2K
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	07FFH	(1) ROM 2K
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0800H	(2) ROM 2K
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH	(2) ROM 2K
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H	ROM 4K
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH	ROM 4K
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H	(1)RAM 2K
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	27FFH	(1)RAM 2K
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2800H	(2)RAM 2K
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	2FFFH	(2)RAM 2K

Έτσι, έχοντας τώρα βρει τον χάρτη της μνήμης, μπορώ να σχεδιάσω τα ζητούμενα κυκλώματα και παραθέτω τα σχέδια παρακάτω:

α) Για την πρώτη περίπτωση, η σχεδίαση γίνεται με τη χρήση αποκωδικοποιητή 3:8 (74LS138) και λογικές πύλες, οπότε και το κύκλωμα σχεδιάστηκε, με την βοήθεια του προσομοιωτή diagrams.net και αποτελεί το κάτωθι:



β)Αν η σχεδίαση τώρα γίνει μόνο με την χρήση λογικών πυλών, τότε η σχεδίαση που προκύπτει αποτελεί η ακόλουθη:



7η ΑΣΚΗΣΗ:

Στην 7η και τελευταία άσκηση, καλούμαστε να σχεδιάσουμε ένα μΥ-Σ 8085 που να χαρακτηρίζεται από χάρτη μνήμης, Ο οποίος πίνακας που αντιστοιχεί στο εν λόγω περιβάλλον-πρόβλημα αποτελεί ο παρακάτω:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address	Memory
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	ROM 16K
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	2FFFH	ROM 16K
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3000H	RAM 4K
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH	RAM 4K
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H	RAM 4K
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	4FFFH	RAM 4K
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	5000H	RAM 4K
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFFH	RAM 4K
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000H	ROM 16K
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	6FFFH	ROM 16K

Υπογραμμίζω και επαναλαμβάνω το δεδομένο που υπάρχει στην εκφώνηση της άσκησης. Και είναι το ότι η διεύθυνση της θύρας εισόδου είναι η 70 Hex ενώ η διεύθυνση της θύρας εξόδου είναι η 7000 Hex

Επομένως και με βάση όλα τα παραπάνω, το αποτέλεσμα που προκύπτει είναι το κάτωθι: