Συστήματα Μικροϋπολογιστών 2023-24 — 2η Ομάδα Ασκήσεων

Κόρδας Νικόλαος - Α.Μ.: 03121032
Κριθαρίδης Κωνσταντίνος - Α.Μ.: 03121045

17 Απριλίου 2024

1η ΑΣΚΗΣΗ

Ερώτημα (α)

Στο ερώτημα αυτό καλούμαστε να αποθηκεύσουμε τους αριθμούς 0 - 127 σε διαδοχικές θέσεις μνήμης, ξεκινώντας από την 0900H (δίνεται στην εκφώνηση). Αυτό το πετυχαίνουμε με τον κώδικα:

```
IN 10H
MVI A,00H ; A register is our counter
LXI B,0900H ; Ad counter

L1:
STAX B
INR A
INR C
CPI 80H
JC L1

END
```

Πιο συγκεκριμένα, αρχικά απενεργοποιούμε την προστασία της μνήμης (IN 10H) και στη συνέχεια αρχικοποιούμε τον μετρητή από το 0 έως το 127 στον καταχωρητή A και τον μετρητή για τις διευθύνσεις μνήμης στο ζεύγος καταχωρητών B-C (διότι οι διευθύνσεις έχουν μέγεθος 2 byte). Στην συνέχεια, στην αρχή κάθε επανάληψης που ξεκινά στην ετικέτα L1 αποθηκεύουμε το περιεχόμενο φορτώνουμε το περιεχόμενο του A (μετρητή 0 - 127) στην θέση μνήμης που έχει αποθηκεύσει το ζεύγος καταχωρητών B-C, και στη συνέχεια αυξάνονται και τα 2 κατά 1. Προσοχή, για να αυξηθούν σωστά οι θέσεις μνήμης, αρκεί να αυξήσω μόνο το κάτω byte, τον C δηλαδή. Τέλος, ελέγχουμε αν ο μετρητής A είναι μικρότερος από 128 και εφόσον είναι εκτελούμε πάλι την ίδια διαδικασία, ενώ αν όχι τερματίζουμε το πρόγραμμα. Επιβεβαιώνουμε την ορθότητα του προγράμματός μας ελέγχοντας τις αλλαγές που επιτελεί στη RAM. Συγκεκριμένα:

0900	00	0901	01	0902	02	0903	03
0904	04	0905	05	0906	06	0907	07
0908	08	0909	09	090A	0A	090B	0B
090C	0C	090D	0D	090E	0E	090F	0F
0910	10	0911	11	0912	12	0913	13
0914	14	0915	15	0916	16	0917	17
0918	18	0919	19	091A	1A	091B	1B
091C	1C	091D	1D	091E	1E	091F	1F
0920	20	0921	21	0922	22	0923	23
0924	24	0925	25	0926	26	0927	27
0928	28	0929	29	092A	2A	092B	2B
092C	2C	092D	2D	092E	2E	092F	2F
0930	30	0931	31	0932	32	0933	33
0934	34	0935	35	0936	36	0937	37
0938	38	0939	39	093A	3A	093B	3B
093C	3C	093D	3D	093E	3E	093F	3F
0940	40	0941	41	0942	42	0943	43
0944	44	0945	45	0946	46	0947	47
0948	48	0949	49	094A	4A	094B	4B
094C	4C	094D	4D	094E	4E	094F	4F
0950	50	0951	51	0952	52	0953	53
0954	54	0955	55	0956	56	0957	57
0958	58	0959	59	095A	5A	095B	5B
095C	5C	095D	5D	095E	5E	095F	5F
0960	60	0961	61	0962	62	0963	63
0964	64	0965	65	0966	66	0967	67
0968	68	0969	69	096A	6A	096B	6B
096C	6C	096D	6D	096E	6E	096F	6F

0970	70	0971	71	0972	72	0973	73
0974	74	0975	75	0976	76	0977	77
0978	78	0979	79	097A	7A	097B	7B
097C	7C	097D	7D	097E	7E	097F	7F
0980	00	0981	00	0982	00	0983	00
0984	00	0985	00	0986	00	0987	00

Βλέπουμε πως πράγματι, το πρόγραμμά μας είναι ορθό.

Ερώτημα (β)

Στο ερώτημα αυτό μετράμε πόσους άσσους έχουμε σε όλους τους δυαδικούς αριθμούς από το 0 - 127. Θεωρητικά, αυτός ο αριθμός είναι 448, αφού οι αριθμοί από το 0 - 127 περιέχουν 7 δυαδικά ψηφιά ο καθένας (σύνολο $7 * 127 = 896$) εκ των οποίων τα μισά είναι άσσοι. Για να τεστάρω το αποτέλεσμα του προγράμματός μου, φορτώνω τους B και C στη μνήμη και κατόπιν τους διαβάζω. Ο κώδικας που χρησιμοποιήθηκε φαίνεται στην επόμενη σελίδα:

```
IN 10H
LXI B,0000H ; Our 1s counter
MVI D,00H ; 0-127 Counter
MVI E,01H ; 1 tester


L2:
MOV A,D
ANA E
CPI 00H
JZ IFEND ; If 0
INX B
IFEND: MOV A,E
        RLC
        MOV E,A
        JNC L2
INR D
MVI E,01H
MOV A,D
CPI 80H
JC L2

; We print the result
MOV A,C
STA 0A20H
MOV A,B
STA 0A21H

END
```

Πιο συγκεκριμένα, εκτελώ την πράξη AND με ένα-ένα τα ψηφία κάθε αριθμού κάθε φορά με τη μάσκα 00000001, 00000010, ... και αν το αποτέλεσμα είναι 1 αυξάνω τον μετρητή ενώ αν όχι συνεχίζω τις επαναλήψεις μέχρι να έρθει η ώρα του επόμενου αριθμού.
Παρακάτω, παραθέτουμε τα περιεχόμενα των θέσεων μνήμης 0A20 και 0A21 που είναι και το αποτέλεσμά μας.

0A18	00	0A19	00	0A1A	00	0A1B	00
0A1C	00	0A1D	00	0A1E	00	0A1F	00
0A20	C0	0A21	01	0A22	00	0A23	00
0A24	00	0A25	00	0A26	00	0A27	00
0A28	00	0A29	00	0A2A	00	0A2B	00

Άρα, σε 16αδική μορφή ο μετρητής μας είναι 01C0 που στο δεκαδικό σύστημα είναι πράγματι το 448.

Ερώτημα (γ)

Στο ερώτημα (γ) καλούμαστε να γράψουμε κώδικα assembly που εξετάζει πόσοι αριθμοί των δεδομένων βρίσκονται στο διάστημα 10H έως 60H. Θεωρητικά, το πλήθος αυτό είναι 60H - 10H + 1H = 51H. Δηλαδή, 81 στο δεκαδικά σύστημα. Ο κώδικας που συντάξαμε φαίνεται παρακάτω:

```
IN 10H
MVI B,00H ; 0-127 counter
MVI D,00H ; Counter of nums

L1:
MOV A,B
CPI 10H ; Is B < 10H?
JC ISEND ; If yes, loop if not end
CPI 60H ; Is B < 60H?
JC YES ; If yes, 10H<=X<60H
JZ YES ; X = 60H
JNZ ISEND ; else loop

YES:
INR D

ISEND:
INR B
MOV A,B
CPI 80H
JC L1

; VISUALIZATION OF RESULT IN MEMORY

MOV A,D
STA 0830H

END
```

Τέλος, μετά την εκτέλεση του προγράμματος βλέπουμε πως στην μνήμη μας στην διεύθυνση 0830H υπάρχει πράγματι ο αριθμός 51H.

0828	00	0829	00	082A	00	082B	00
082C	00	082D	00	082E	00	082F	00
0830	51	0831	00	0832	00	0833	00
0834	00	0835	00	0836	00	0837	00
0838	00	0839	00	083A	00	083B	00

2η ΑΣΚΗΣΗ

```
       MVI C,64H ; Timer initialization C = 100 (total ms in 0.1 seconds)
       MVI D,00H ; Counter initialisation D = 0
INIT_ON: LDA 2000H ; Read the input from the dip switches
       CPI 7FH  ; 0111 1111b
       JNC INIT_ON ; If MSB set, switch initially ON
OFF_STATE: LDA 2000H ; Read the input from the dip switches
       CPI 7FH  ; 0111 1111b
       JC OFF_STATE  ; if MSB unset, repeat OFF_STATE
ON_STATE: MOV A,D ; Move counter D to A
       CPI 00H ; Check if D is equal to 0
       JZ SKIP_DEL ; If yes, skip delay
       PUSH D  ; Push D to stack in case it changes in the call
       CALL DELB ; Delay for 0.1 seconds
       POP D  ; Pop D from stack
       DCR D  ; Decrease counter D
       JNZ SKIP_DEL ; If D != 0, skip updating the LEDs
       MVI A,FFH  ; Set A to 1111 1111b to clear all LEDs
       STA 3000H ; Output A to the (negative logic) LED's
SKIP_DEL: LDA 2000H ; Read the input from the dip switches
       CPI 7FH  ; 0111 1111b
       JNC ON_STATE  ; if MSB set, repeat ON_STATE
       MVI D,32H ; Counter initialisation D = 50 (50*0.1 = 20 seconds)
DELAY:  XRA A  ; Clear A to open all LED's
       STA 3000H ; Output A to the (negative logic) LED's
       PUSH D  ; Push D to stack in case it changes in the call
       CALL DELB ; Delay for 0.1 seconds
       POP D  ; Pop D from stack
       LDA 2000H ; Read the input from the dip switches
       CPI 7FH  ; 0111 1111b
       JNC ON_STATE ; If MSB set, repeat ON_STATE (and continue delaying)
       DCR D  ; Decrease the counter
       JNZ DELAY ; If not reached 10 seconds, repeat DELAY state
       MVI A,FFH  ; Set A to 1111 1111b to clear all LEDs
       STA 3000H ; Output A to the (negative logic) LED's
       JMP OFF_STATE ; Stop delaying and go to OFF_STATE
END
```

Στην αρχή, αρχικοποιούμε τις μεταβλητές του timer και του μετρητή. Αν αρχικά είναι ON παραμένουμε στο ON μέχρι να αλλάξει το dip switch του MSB. Μόλις αλλάξει, προχωράμε στο κύριο μέρος του κώδικα και όπου η κατάσταση ξεκινάει ως OFF. Όταν η κατάσταση αλλάξει σε ON, αν δεν είχε ήδη συμβεί OFF-ON-OFF, οπότε ο μετρητής είναι 0, τότε επαναλαμβάνουμε. Αν δεν είναι 0, τότε περιμένουμε 0.1sec, την διακριτική ικανότητα που ζητείται, και μειώνουμε τον μετρητή. Αν ο μετρητής έγινε 0, τότε κάνουμε clear τα LEDs. Στη συνέχεια ελέγχουμε αν η κατάσταση άλλαξε σε OFF (οπότε έγινε OFF-ON-OFF), και αν ναι, αρχικοποιούμε τον μετρητή στην κατάλληλη τιμή για να έχουμε ανοιχτά τα LEDs για 20 δευτερόλεπτα. Αν η κατάσταση αλλάξει σε ON, επαναλαμβάνουμε ό,τι αναφέραμε προηγουμένως και επανεκινούμε τον μετρητή αν αυτό χρειαστεί με νέα μετάβαση σε OFF. Αλλιώς, αν απλά περάσουν τα 20 δευτερόλεπτα ενώ βρισκόμαστε στο OFF, σβήνουμε τα λαμπάκια και γυρίζουμε στο label OFF_STATE, επαναλαμβάνοντας τον όλο αλγόριθμο.

3η ΑΣΚΗΣΗ

Ερώτημα i.

Ο κώδικας assembly για το ερώτημα αυτό φαίνεται παρακάτω:

```
MVI B,00H

SWITCH_LIGHTS:
MOV A,B
RRC
CMA
STA 3000H

INPUT:
LDA 2000H ; Read input in register A
MOV C,A ; Store input
MVI B,01H ; Position of open switch

SEARCH: ;Rotate right till overflow
MOV A,B
RLC ; Position update
MOV B,A
JC SWITCH_LIGHTS
MOV A,C
RRC
MOV C,A
JC SWITCH_LIGHTS
JNC SEARCH


END
```

Η ιδέα είναι να στρίβουμε το input δεξιόστροφα μέχρι να υπερχειλίσει, κάνοντας το ανάποδο σε έναν άλλο καταχωρητή που θα μας υποδείξει τι θα ανάψουμε. Το πρόγραμμα αυτό είναι συνεχούς λειτουργίας και έτσι αλλάζει κατάσταση κάθε φορά που πειράζουμε τους διακόπτες του simulator.

Ερώτημα ii.

Ο κώδικας assembly για το ερώτημα αυτό φαίνεται παρακάτω:

```
MVI A,00H

SWITCH_LIGHTS:
STA 3000H

INPUT:
CALL KIND
MOV B,A ; Store input in B register
CPI 00H ; Comp input with zero
JZ INPUT ; If zero, new input
CPI 09H ; Comp with 9
JC L1 ; If 1 <= input < 9, valid
JNC INPUT ; Else new input

L1:
MVI A,00H ; Init A
L2:
DCR B
JZ SWITCH_LIGHTS
RLC
INR A ; Next led on too
JMP L2

END
```

Αρχικά ξεκινάμε με όλα τα λαμπάκια αναμμένα (δε διευκρινίζεται κάτι στην εκφώνηση περί αρχικής κατάστασης οπότε επιλέγουμε τυχαία αυτή). Το πρόγραμμα είναι συνεχούς λειτουργίας. Οπότε κάθε φορά περιμένει το input του χρήστη από το πληκτρολόγιο και αρχικά εκτελεί ελέγχους για το αν αυτό είναι επιτρεπτό. Κατόπιν, αφαιρούμε συνεχώς 1 από την είσοδο (που έχει αποθηκευτεί στον καταχωρωτή B) για να δούμε πόσα led θα ανάψουμε (κάθε μείωση είναι ένα rotation RLC). Μόλις βρούμε το αποτέλεσμα, η εκτέλεση μεταβαίνει στο άναμμα των LED και στην συνέχει πάλι στο input... Ο κώδικας περιέχει και σχόλια.

Ερώτημα iii.

Ο κώδικας για το ερώτημα αυτό φαίνεται παρακάτω:

```
1  IN 10H ; No memory protection
2
3  START:
4  LXI D,0B00H ; Big address, no conflict
5  MVI A,10H ; Idle char (fist four pos)
6  STA 0B00H
7  STA 0B01H
8  STA 0B02H
9  STA 0B03H ; Idle is one byte long
10
11 ; NOW READING
12
13 LINE_0:
14     MVI A,FEH ; Line 11111110
15     STA 2800H
16     LDA 1800H ; Read column
17     MVI B,07H
18     ANA B ; 0 the first 5 MSB
19     MVI B,86H
20     CPI 06H ; INSTR_STEP???
21     JZ SHOW
22     MVI B,85H
23     CPI 05H ; FETCH PC???
24     JZ SHOW
25     ; Ignore HDWR_STEP
```

```
26
27  LINE_1:
28      MVI A,FDH ; Line 11111101
29      STA 2800H
30      LDA 1800H ; Read column
31      MVI B,07H
32      ANA B ; 0 the first 5 MSB
33      MVI B,84H
34      CPI 06H ; INSTR_STEP???
35      JZ SHOW
36      MVI B,80H
37      CPI 05H ; FETCH PC???
38      JZ SHOW
39      MVI B,82H
40      CPI 03H ; FETCH ADRS???
41      JZ SHOW
42
43  LINE_2:
44      MVI A,FBH ; Line 11111011
45      STA 2800H
46      LDA 1800H ; Read column
47      MVI B,07H
48      ANA B ; 0 the first 5 MSB
49      MVI B,00H
50      CPI 06H ; 0???
51      JZ SHOW
52      MVI B,83H
53      CPI 05H ; STORE/INCR???
54      JZ SHOW
55      MVI B,81H
56      CPI 03H ; INCR???
57      JZ SHOW
58
59
60  LINE_3:
61      MVI A,F7H ; Line 11110111
62      STA 2800H
63      LDA 1800H ; Read column
64      MVI B,07H
65      ANA B ; 0 the first 5 MSB
66      MVI B,01H
67      CPI 06H ; 1???
68      JZ SHOW
69      MVI B,02H
70      CPI 05H ; 2???
71      JZ SHOW
72      MVI B,03H
73      CPI 03H ; 3???
74      JZ SHOW
75
76  LINE_4:
77      MVI A,EFH ; Line 11101111
78      STA 2800H
79      LDA 1800H ; Read column
80      MVI B,07H
81      ANA B ; 0 the first 5 MSB
82      MVI B,04H
83      CPI 06H ; 4???
84      JZ SHOW
85      MVI B,05H
86      CPI 05H ; 5???
87      JZ SHOW
88      MVI B,06H
89      CPI 03H ; 6???
90      JZ SHOW
91
92  LINE_5:
93      MVI A,DFH ; Line 11011111
94      STA 2800H
95      LDA 1800H ; Read column
96      MVI B,07H
97      ANA B ; 0 the first 5 MSB
98      MVI B,07H
99      CPI 06H ; 7???
100     JZ SHOW
101     MVI B,08H
102     CPI 05H ; 8???
```

```
    JZ SHOW
    MVI B,09H
    CPI 03H ; 9???
    JZ SHOW

LINE_6:
    MVI A,BFH ; Line 10111111
    STA 2800H
    LDA 1800H ; Read column
    MVI B,07H
    ANA B ; 0 the first 5 MSB
    MVI B,0AH
    CPI 06H ; A???
    JZ SHOW
    MVI B,0BH
    CPI 05H ; B???
    JZ SHOW
    MVI B,0CH
    CPI 03H ; C???
    JZ SHOW

LINE_7:
    MVI A,7FH ; Line 01111111
    STA 2800H
    LDA 1800H ; Read column
    MVI B,07H
    ANA B ; 0 the first 5 MSB
    MVI B,0DH
    CPI 06H ; D???
    JZ SHOW
    MVI B,0EH
    CPI 05H ; E???
    JZ SHOW
    MVI B,0FH
    CPI 03H ; F???
    JZ SHOW

JMP START ; If nothing pressed, again

SHOW:
    LXI H,0B04H ; Continue store block
    MOV A,B
    ANI 0FH ; 4 LSB
    MOV M,A
    INX H
    MOV A,B
    ANI F0H ; 4 MSB
    RLC
    RLC
    RLC
    RLC
    MOV M,A

    CALL STDM
    CALL DCD
    JMP START

END
```

Πρώτα από όλα αφαιρούμε την προστασία της μνήμης. Στην συνέχεια αποθηκεύουμε στα 4 LSB τον κενό χαρακτήρα (από αυτά που θα προβάλλουμε στην οθόνη). Ουσιαστικά, όπως λέει και η εκφώνηση θέλουμε μόνο τα 2 τελευταία. Στην συνέχεια, εφαρμόζουμε τις οδηγίες του εγχειριδίου του μLAB για την κάθε γραμμή του πληκτρολογίου. Αφού την επιλέξουμε (STA με το κατάλληλο byte), διαβάζουμε τις στήλες και αποθηκεύουμε κάθε πιθανό πλήκτρο στον καταχωρητή B. Ελέγχουμε αν όντως το αντίστοιχο πλήκτρο πιέστηκε και κατόπιν μεταφέρουμε την εκτέλεση στην ετικέτα που είναι υπεύθυνη για την εμφάνιση μηνυμάτων στην οθόνη. Εκεί, πάλι σύμφωνα με το εγχειρήδιο του προσομοιωτή εκτελούμε τα κατάλληλα βήματα προκειμένου να εμφανιστεί ο χαρακτήρα που πιέστηκε στην οθόνη.
ΣΗΜΕΙΩΣΗ: Οι χαρακτήρες τύπου FETCH_PC αναπαρίστανται από τους δεκαεξαδικούς αριθμούς που τους αντιστοιχούν.

4η ΑΣΚΗΣΗ

```
    LXI SP,0BB0H ; Initialize Stack Pointer
    LDA 2000H

```

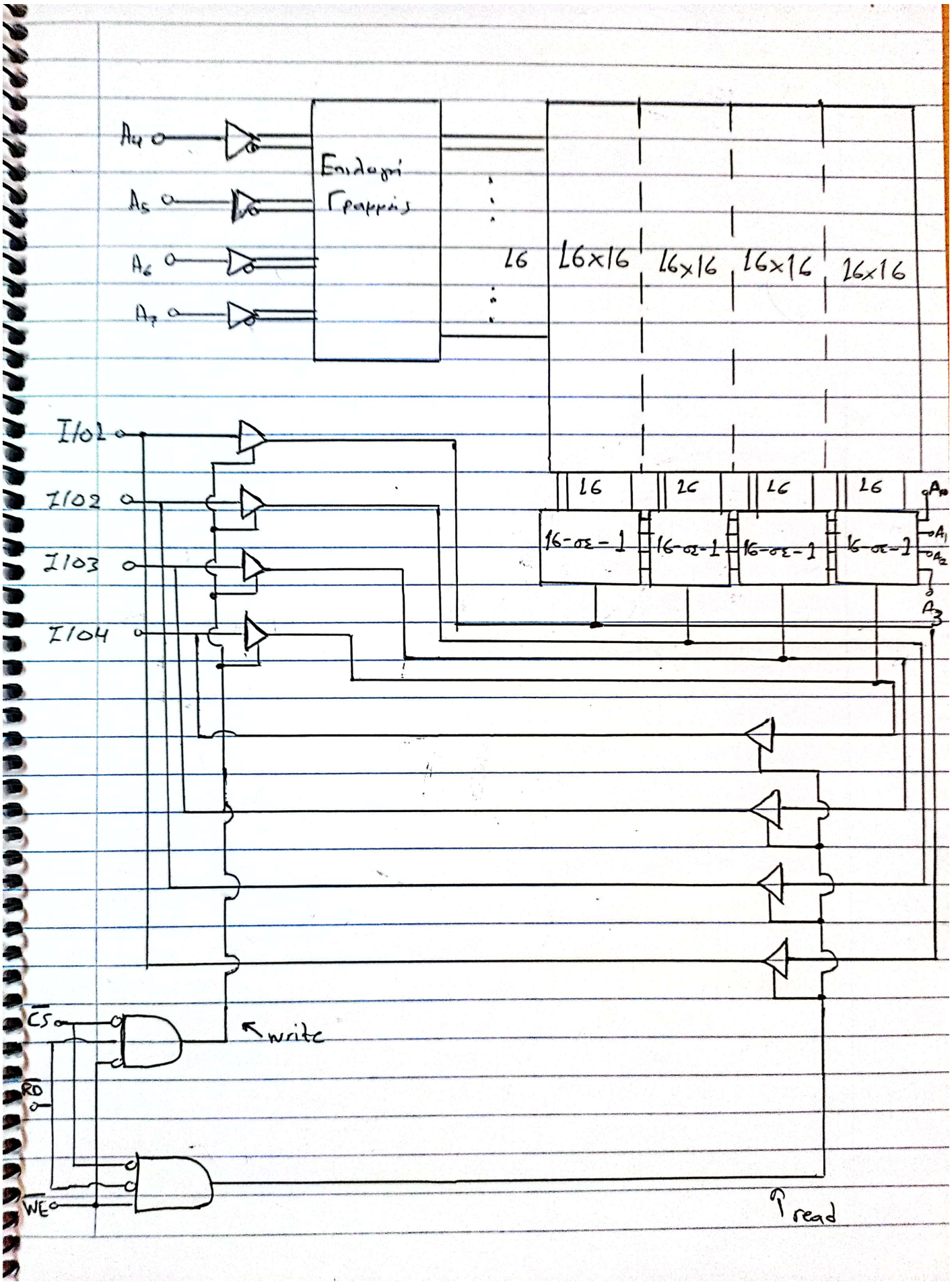
```
    MOV B,A  ; B[0] = B0
    RRC  ; A[0] = A0
    MOV C,A ; C = (2000H)>>2
    ANA B  ; A[0] = A0 AND B0
    MOV D,A ; D[0] = A0 AND B0. STORED IN D (for now)
    MOV A,C ; A = (2000H)>>2
    RRC  ; A = (2000H)>>3

    MOV B,A ; B[0] = B1
    RRC  ; A[0] = A1
    MOV C,A ; C = (2000H)>>4
    ANA B  ; A[0] = A1 AND B1
    MOV E,A ; E[0] = A1 AND B1. STORED IN E
    MOV A,C ; A = (2000H)>>5
    RRC  ; A = (2000H)>>6

    MOV B,A ; B[0] = B2
    RRC  ; A[0] = A2
    MOV C,A ; C = (2000H)>>6
    XRA B  ; A[0] = A2 XOR B2
    MOV H,A ; H[0] = A2 XOR B2. STORED IN H (for now)
    MOV A,C ; A = (2000H)>>5
    RRC  ; A = (2000H)>>6

    MOV B,A ; B[0] = B3
    RRC  ; A[0] = A3
    MOV C,A ; C = (2000H)>>7
    XRA B  ; A[0] = A3 XOR B3
    MOV L,A ; L[0] = A3 XOR B3. STORED IN L

    MOV A,D ; A[0] = A0 AND B0
    ORA E  ; A[0] = (A0 AND B0) OR (A1 AND B1)
    MOV D,A ; D[0] = (A0 AND B0) OR (A1 AND B1). STORED IN D

    MOV A,H ; A[0] = A2 XOR B2
    ORA L  ; A[0] = (A2 XOR B2) OR (A3 XOR B3)
    MOV H,A ; H[0] = (A2 XOR B2) OR (A3 XOR B3). STORED IN H

    MVI B,01H ; B = 0000 0001b. Initialise Mask
    MVI C,00H ; C = 0000 0000b. Initialise Result

    MOV A,L ; A[0] = X_3
    CALL LOAD1  ; C[1] = X_3, C[else] = 0

    MOV A,H ; A[0] = X_2
    CALL LOAD1 ; C[1] = X_2, C[2] = X_3, C[else] = 0

    MOV A,E ; A[0] = X_1
    CALL LOAD1 ; C[1] = X_1, C[2] = X_2, C[3] = X_3

    MOV A,D ; A[0] = X_3
    CALL LOAD2 ; C[0..3] = X_[0..3], C[else] = 0

    MOV A,C ; Move result to A
    CMA  ; Complement A to get result in negative logic
    STA 3000H ; Output the result to the LEDs

LOAD1:   ; C<<=1, C[1] = A[0], C[0] = 0
    ANA B  ; Get bit
    ORA C  ; Add to result. Updated result in A
    RLC  ; Shift A 1 position to the left
    MOV C,A ; Updated result in C
    RET  ; Return to main program

LOAD2:   ; C[0] = A[0]

    ANA B  ; Get bit
    ORA C  ; Add to result. Updated result in A
    MOV C,A ; Updated result in C
    RET  ; Return to main program

END
```

Υπολογίζουμε τη λογική συνάρτηση για το κάθε bit εξόδου, ξεκινώντας τόσο στο input όσο και στο output τους υπολογισμούς από δεξιά προς τα αριστερά. Αποθηκεύουμε τις τιμές εισόδου που διαβάζουμε σε διαφορετικούς καταχωρητές, κάνοντας κατάλληλες ολισθήσεις ώστε οι τιμές αληθείας που θέλουμε να βρίσκονται πάντα στο LSB των καταχωρητών. Όταν

διαβάσουμε τα $A_0,\ B_0,\ A_1,\ B_1$ υπολογίζουμε παράλληλα τα $X_0,\ X_1$. Ομοίως, όταν διαβάσουμε τα $A_2,\ B_2,\ A_3,\ B_3$, υπολογίζουμε τα $X_2,\ X_3$. Επαναχρησιμοποιούμε όποιον καταχωρητή πλέον δεν χρειαζόμαστε. Στο τέλος, έχοντας σε διαφορετικούς καταχωρητές τα 4 bits της εξόδου, τα φορτώνουμε όλα με τη σωστή σειρά σε έναν καταχωρητή, τον οποίο στέλνουμε με ανάποδη λογική στα LEDs της εξόδου.

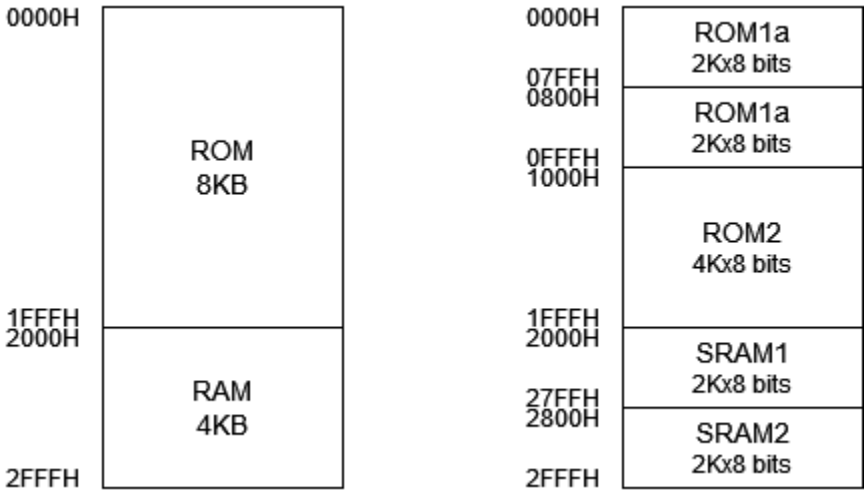
5η ΑΣΚΗΣΗ

Στο σχήμα που παραθέτουμε παραπάνω φαίνεται η εσωτερική οργάνωση μίας μνήμης SRAM 256x64 bit. Ισχύει ότι, 256x4 = 16x16x4 = 2^4x2^4x4. Συνεπώς, προκειμένου να επιλέξουμε μία από τις 16 γραμμές της μνήμης χρειαζόμαστε 4 bit (A4, A5, A6, A7). Καθένα από τα bit εισόδου/εξόδου (I/O1, I/O2, I/O3, I/O4) είναι συνδεδεμένο σε έναν πολυπλέκτη 16-σε-1, οι οποίοι είναι υπεύθυνοι για την επιλογή μίας εκ των 16 τετράδων στις οποίες θα εγγραφούν ή από τις οποίες θα αναγνωστούν. Πιο συγκεκριμένα, η επιλογή των τετράδων ελέγχεται από τα 4 LSB της διεύθυνσης μνήμης A0, A1, A2, A3.
Μένει να μπορούμε να επιλέξουμε πότε θα γίνεται ανάγνωση και πότε εγγραφή στον πίνακα της μνήμης. Αυτό γίνεται με την βοήθεια απομονωτών και των σημάτων $\overline{CS}, \overline{RD}, \overline{WE}$. Λεπτομερέστερα, μόλις το $\overline{CS}$ γίνει 0, η μνήμη ενεργοποιείται (το 1 εκ των τριών ορισμάτων των πυλών AND γίνεται 1). Εφόσον, πλέον ($\overline{CS} = 0$) τα σήματα $\overline{WE} = 1$ και $\overline{RD} = 0$, ενεργοποιούνται οι απομονωτές των γραμμών σημασμένων με read και συνεπώς θα έχουμε ανάγνωση από τη μνήμη. Τέλος, στην περίπτωση που τα σήματα $\overline{WE} = 0$ και $\overline{RD} = 1$, ενεργοποιούνται οι απομονωτές των γραμμών σημασμένων με write και συνεπώς θα έχουμε εγγραφή από στη μνήμη.


A4
A5
A6
A7
Επιλογή Γραμμής
16
16x16
16x16
16x16
16x16
I/O1
I/O2
I/O3
I/O4
16
16
16
16
16-στ-1
16-στ-1
16-στ-1
16-στ-1
A0
A1
A2
A3
CS
RD
WE
write
read

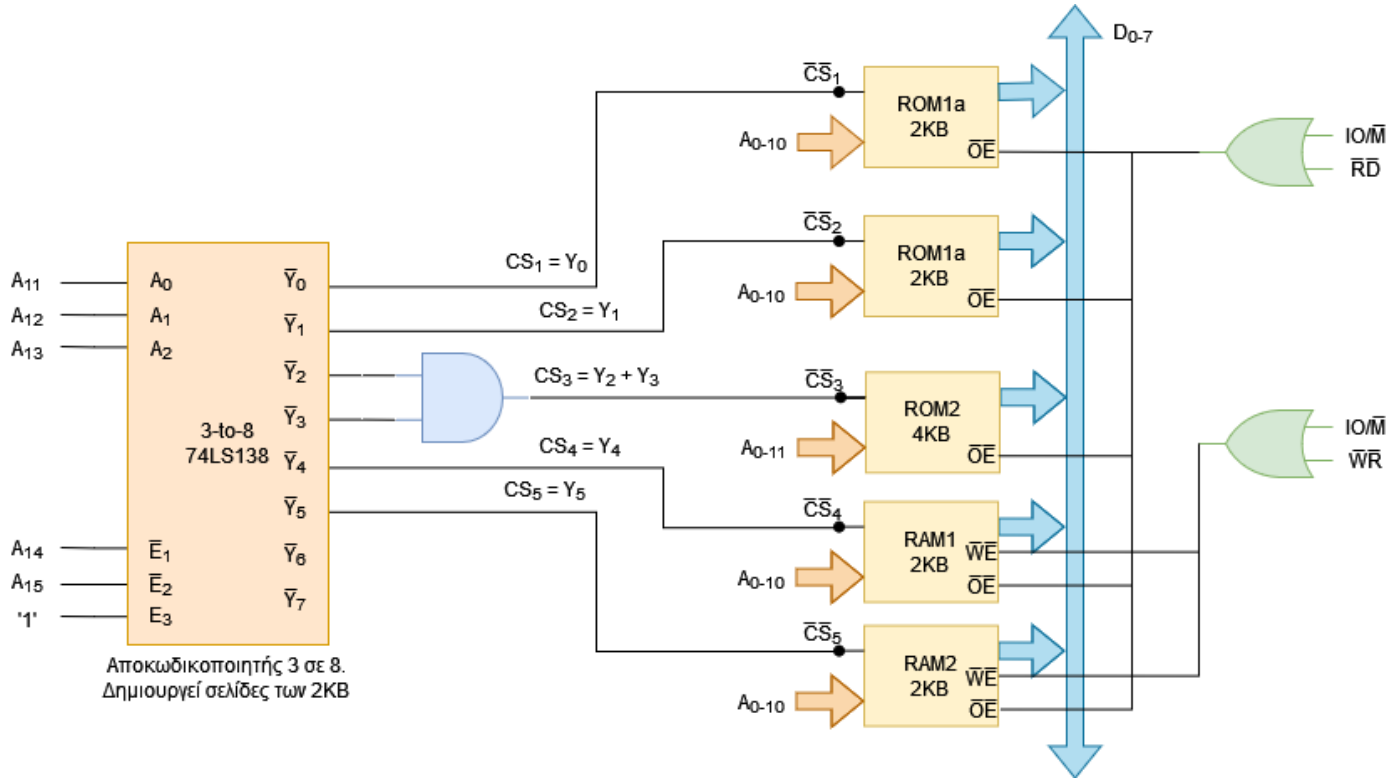
6η ΑΣΚΗΣΗ

Αρχικά, παρουσιάζουμε παρακάτω τον χάρτη μνήμης του συστήματος που δίνεται στην εκφώνηση.

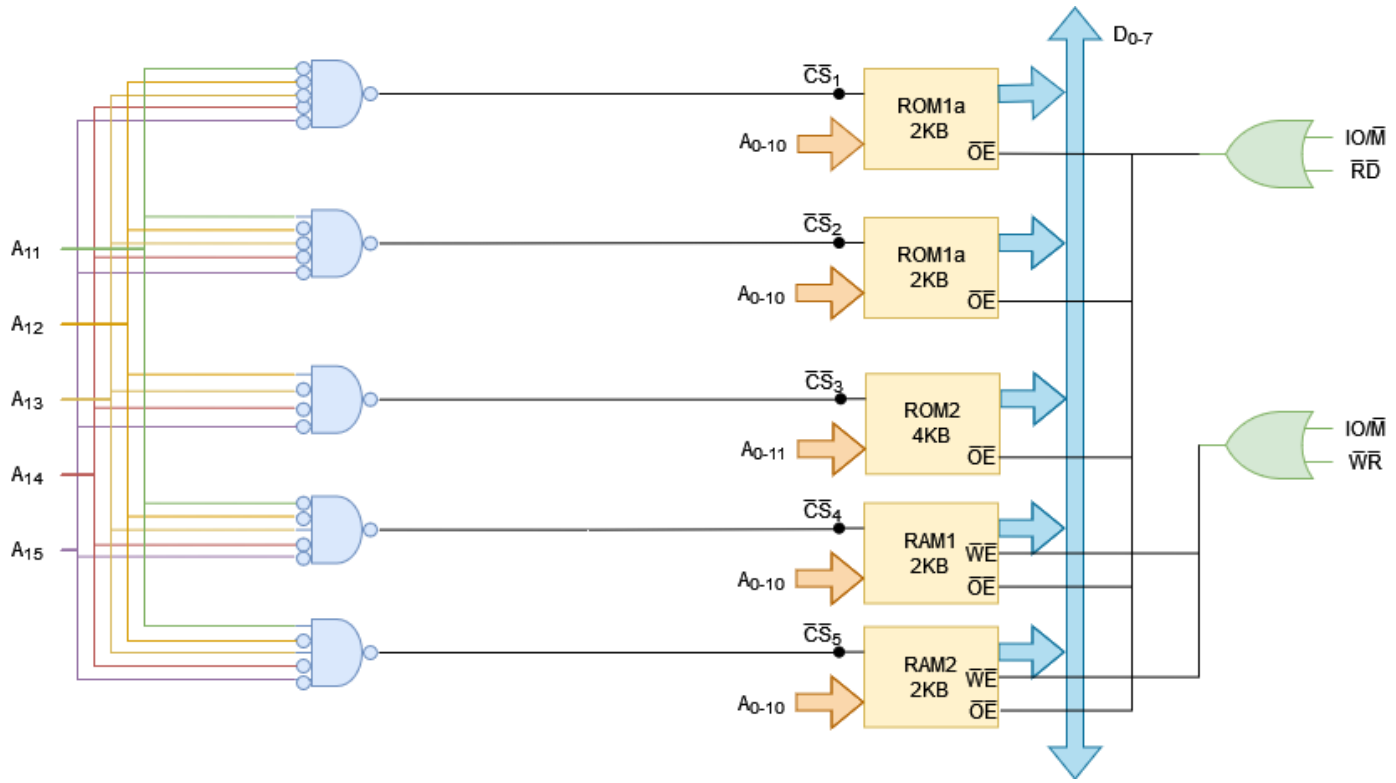


Χάρτης Μνήμης																	
Περιοχή ROM																	
Memory	**Address**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ROM1a	0000	0	0	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0
(2KB)	07FF	0	0	**0**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1
ROM1b	0800	0	0	**0**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0
(2KB)	0FFF	0	0	**0**	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1
ROM2	1000	0	0	**0**	**1**	**0**	0	0	0	0	0	0	0	0	0	0	0
(4KB)	1FFF	0	0	**0**	**1**	**1**	1	1	1	1	1	1	1	1	1	1	1
Περιοχή ROM																	
Memory	**Address**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RAM1	2000	0	0	**1**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0
(2KB)	27FF	0	0	**1**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1
RAM2	2800	0	0	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0
(2KB)	2FFF	0	0	**1**	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1

α) Το λογικό διάγραμμα της μνήμης και η συνδεσμολογία με τα απαιτούμενα σήματα από το system bus του μΕ 8085, όταν η αποκωδικοποίηση των διευθύνσεων γίνεται με αποκωδικοποιητή 3:8 (74LS138) φαίνεται παρακάτω.



β) Ενώ όταν η αποκωδικοποίηση γίνεται μόνο με λογικές πύλες, το διάγραμμα και οι αντίστοιχες διασυνδέσεις φαίνονται παρακάτω.

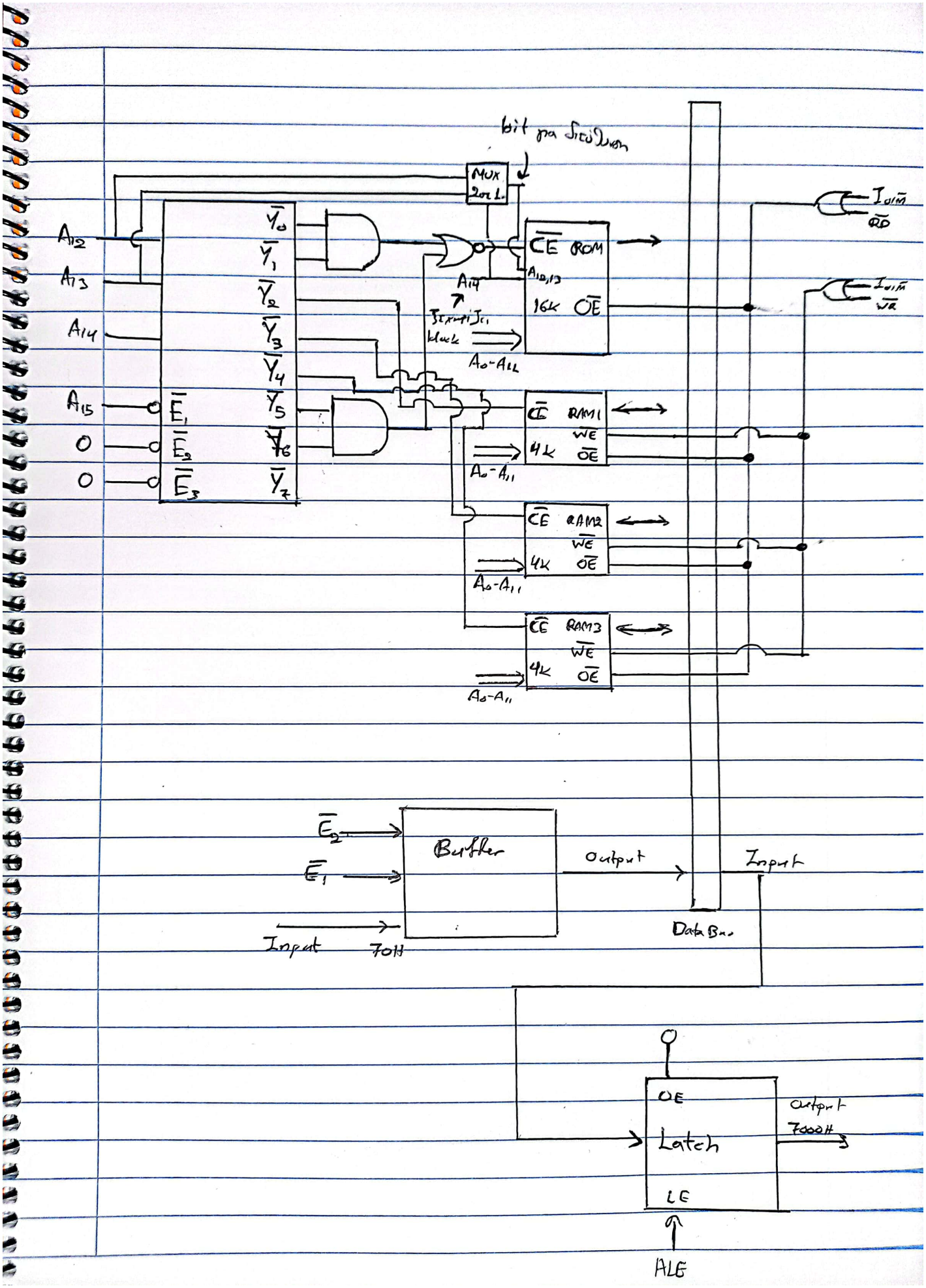


7η ΑΣΚΗΣΗ

Πρώτα από όλα, παρουσιάζουμε τον χάρτη μνήμης μαζί τα bits των διευθύνσεων του χάρτη μνήμης.

Χάρτης Μνήμης																	
Memory	**Address**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ROM1	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8KBytes)	1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM1	2000	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
(4KBytes)	2FFF	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
RAM2	3000	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
(4KBytes)	3FFF	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM3	4000	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4KBytes)	4FFF	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
ROM2	5000	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
(8KBytes)	6FFF	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1

Παρατηρώντας το χάρτη, βλέπουμε πως το MSB A_{15} δεν μεταβάλλεται. Επομένως θα το χρησιμοποιήσουμε στην επίτρεψη. Κατόπιν, βλέπουμε πως τα bit $A_0 - A_{11}$ παίρνουν όλες τις δυνατές τους τιμές, δημιουργώντας σελίδες των 4 KByte. Έτσι, μένουν 3 bit, τα A_{12}, A_{13}, A_{14} τα οποία θα επιλέγουν μεταξύ των σελίδων αυτών. Οι πρώτες 2 σελίδες αφορούν τα πρώτα 8 KByte της ROM, οι επόμενες 3 τις 3 RAM και οι 2 ακόμα τα υπόλοιπα 8 KByte της ROM. Η τελευταία σελίδα μπορεί να χρησιμοποιηθεί και για το input 70H καθώς και για την memory-mapped I/O στην 7000H. Για την ROM, επειδή την χωρίζουμε εικονικά, πρέπει κάπως να διαχωρίσουμε τις 2 περιοχές της. Για αυτό χρησιμοποιούμε το bit A14 το οποίο της το περνάμε και αυτό σαν είσοδο. Για να ολοκληρθεί η διευθυνσιοδότηση εκτός των bit A0-A11 χρησιμοποιούμε το A12 για την πρώτη περιοχή και το A13 για την δεύτερη επιλέγοντας τα κάθε φορά με βάση ποια περιοχή ενεργοποιείται (MUX με selector το A14). Το κύκλωμα απεικονίζεται παρακάτω:


MUX
2or L
A_{12}
A_{13}
A_{14}
A_{15}
0
0
$\bar{E}_1$
$\bar{E}_2$
$\bar{E}_3$
$\bar{Y}_0$
$\bar{Y}_1$
$\bar{Y}_2$
$\bar{Y}_3$
$\bar{Y}_4$
$\bar{Y}_5$
$\bar{Y}_6$
$\bar{Y}_7$
A_{14}
$\overline{CE}$ ROM
$A_{10,13}$
16K $\overline{OE}$
A_0-A_{LL}
$\overline{CE}$ RAM1
$\overline{WE}$
4K $\overline{OE}$
A_0-A_{11}
$\overline{CE}$ RAM2
$\overline{WE}$
4K $\overline{OE}$
A_0-A_{11}
$\overline{CE}$ RAM3
$\overline{WE}$
4K $\overline{OE}$
A_0-A_{11}
$I_{o/\overline{M}}$
$\overline{RD}$
$I_{o/\overline{M}}$
$\overline{WR}$
$\bar{E}_2$
$\bar{E}_1$
Buffer
Output
Input
Input
70H
Data Bus
OE
Latch
LE
Output
7000H
ALE