ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΣΥΣΤΗΜΑΤΑ ΜΙΚΟΫΠΟΛΟΓΙΣΤΩΝ - 6ο ΕΞΑΜΗΝΟ
ΑΝΑΦΟΡΑ 2ης ΟΜΑΔΑΣ ΑΣΚΗΣΕΩΝ

ΓΕΩΡΓΙΑΔΗ ΔΑΦΝΗ 03120189
ΓΕΩΡΓΑΚΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ 03120827

ΑΣΚΗΣΕΙΣ ΠΡΟΣΟΜΟΙΩΣΗΣ

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

ΑΣΚΗΣΗ 1

Ερώτημα Α

```
; ΕΡΩΤΗΜΑ Α

        IN 10H
        MVI A,00H       ;Αριθμοί που θα αποθηκευουμε
        LXI H,0900H     ;Διευθυνσεις που θα χρησιμοποιήσουμε
        MOV M,A         ;Αποθηκευση του 0
STORE:
        INR A           ;Αυξηση και του A και του H
        INX H
        MOV M,A         ;Αποθηκευση του A
        CPI 7FH         ;Loop μεχρι A>127
        JNZ STORE
```

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

Παρατηρήσεις:

1. Για να πιστοποιήσουμε ότι το πρόγραμμα ήταν επιτυχές και δούλεψε όπως θέλαμε κοιτάμε τον χώρο μνήμης του μικροεπεξεργαστή και παρατηρούμε ότι σε κάθε διεύθυνση μνήμης έχουμε την τιμή που ζητάμε

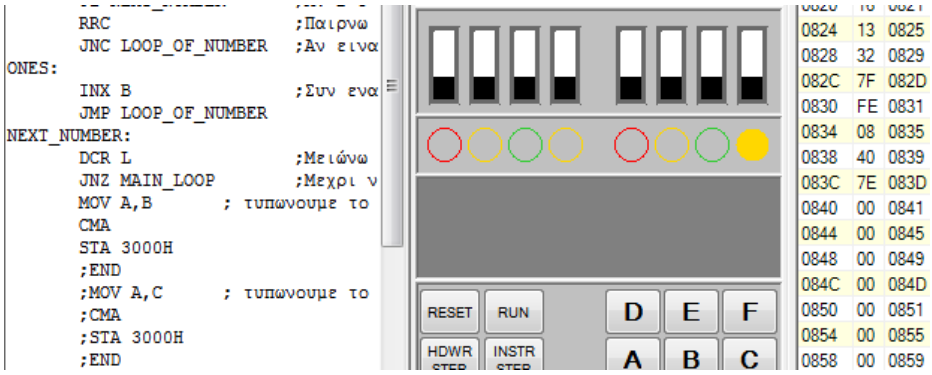
Ερώτημα Β

```
14 ; ΕΡΩΤΗΜΑ Β
15         LXI B,0000H             ;Αποθηκευω στο BC την τιμή 0
16 MAIN_LOOP:
17         MOV A,M                 ;Φορτωνω στο A το τελευταιο στοιχείο (127 θεση M=097FH)
18         MVI D,09H               ;Θετω το D=9 για το loop του καθε αριθμού
19 LOOP_OF_NUMBER:
20         DCR D
21         JZ NEXT_NUMBER          ;Αν D=0 παω στον επόμενο αριθμό (ο οποιος ειναι κατα 1 μικρλότερος)
22         RRC                     ;Παιρνω το τελυταιό ψηφιο απο τον αριθμό
23         JNC LOOP_OF_NUMBER      ;Αν ειναι ασσος το μετραω, αλλιως προχωραω
24 ONES:
25         INX B                   ;Συν ενας δυαδικος ασσος
26         JMP LOOP_OF_NUMBER
27 NEXT_NUMBER:
28         DCR L                   ;Μειώνω κατα 1 την διευθυνση (επόμενος αριθμος)
29         JNZ MAIN_LOOP           ;Μεχρι να φτάσω στο 0
30         ;MOV A,B         ; τυπωνουμε το B
31         ;CMA
32         ;STA 3000H
33         ;END
34         ;MOV A,C         ; τυπωνουμε το C
35         ;CMA
36         ;STA 3000H
37         ;END
```

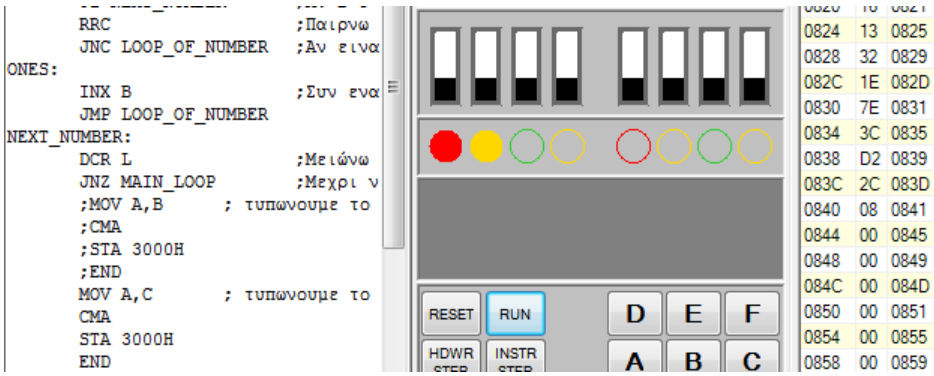
Παρατηρήσεις:

1. Η λογική είναι ότι για κάθε αριθμό/καταχωρητή, ξέρω ότι αποτελείται από 8bits (1 byte) Άρα κάνω ένα μεγάλο loop καλύπτοντας όλους τους αριθμούς με αντίστροφη σειρά που τους έβαλα στην μνήμη και για κάθε έναν κάνω 8 επαναλήψεις κοιτάζοντας κάθε bit.
2. Το αποτέλεσμα που ψάχνουμε αποθηκεύεται στους καταχωρητές B και C. Με τις εντολές των γραμμών 30 έως 33 και 34 έως 37 εμφανίζουμε στα LEDs τις τιμές των B και C αντίστοιχα:

B:



C:



3. Το αποτέλεσμα που βρήκαμε είναι σωστό. Οι συνολικοί άσσοι υπολογίζονται ως εξής:

Από συμμετρία κάθε bit από τα 8 bit είναι 1 τις μισές φορές. Αυτό συμβαίνει διότι το πλήθος των αριθμών μορφής xxx1xxxx είναι ίδιο με το πλήθος των αριθμών της μορφής xxx0xxxx)
Άρα από 0 έως 127 ο συνολικός αριθμών άσσων είναι

$$\frac{1}{2} \cdot 7 \cdot 128 \; = \; 448$$

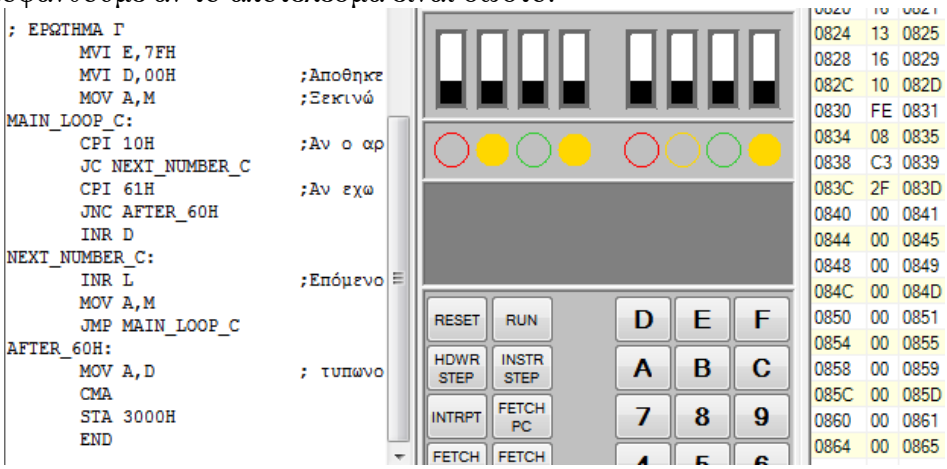
Το 448 σε δεκαεξαδική μορφή είναι: 01C0
Ο αριθμός αυτός χωρίζεται ως εξής στους καταχωρητές B και C:
B: 01_{16} = 0000 0001
C: $C0_{16}$ = 1100 0000
Επομένως το αποτέλεσμα είναι σωστό!

Ερώτημα Γ

```
; ΕΡΩΤΗΜΑ Γ
        MVI E,7FH
        MVI D,00H               ;Αποθηκευω στο D το πλήθος
        MOV A,M                 ;Ξεκινώ απο το 1ο στοιχείο (0 θεση M=0900H)
MAIN_LOOP_C:
        CPI 10H                 ;Αν ο αριθμός ειναι μικρότερος του 10H πηγαινε στον επόμενο
        JC NEXT_NUMBER_C
        CPI 61H                 ;Αν εχω προσπεράσει τον 60H ουσιαστικα εχω τελειώσει
        JNC AFTER_60H
        INR D
NEXT_NUMBER_C:
        INR L                   ;Επόμενος αριθμός
        MOV A,M
        JMP MAIN_LOOP_C
AFTER_60H:
        MOV A,D         ; τυπωνουμε το D
        CMA
        STA 3000H
        END
```

Παρατηρήσεις:

1. Το τελικό αποτέλεσμα αποθηκεύεται στον καταχωρητή D. Επομένως όπως και στο προηγούμενο ερώτημα εμφανίζουμε στα LEDs το περιεχόμενο του D για να αποφανθούμε αν το αποτέλεσμα είναι σωστό:



2. Οι αριθμοί 10H και 60H είναι στο δεκαδικό σύστημα οι αριθμοί 16 και 96 αντίστοιχα. Επομένως το πλήθος των αριθμών που περιλαμβάνονται ανάμεσα σε αυτούς τους δύο συμπεριλαμβανομένων αυτών είναι 96-16+1 = 81. Το 81 στο δυαδικό είναι: 01010001.

Επομένως το αποτέλεσμα είναι σωστό!

Ο κώδικας για το συνολικό ερώτημα είναι ο εξής:

```
; ΕΡΩΤΗΜΑ Α
        IN 10H
        MVI A,00H       ;Αριθμοί που θα αποθηκευουμε
        LXI H,0900H     ;Διευθυνσεις που θα χρησιμοποιήσουμε
        MOV M,A         ;Αποθηκευση του 0
STORE:
        INR A           ;Αυξηση και του Α και του Η
        INX H
        MOV M,A         ;Αποθηκευση του Α
        CPI 7FH         ;Loop μεχρι Α>127
        JNZ STORE

; ΕΡΩΤΗΜΑ Β
        LXI B,0000H             ;Αποθηκευω στο BC την τιμή 0
MAIN_LOOP:
        MOV A,M                 ;Φορτωνω στο Α το τελευταιο στοιχείο (127 θεση M=097FH)
        MVI D,09H               ;Θετω το D=9 για το loop του καθε αριθμού
LOOP_OF_NUMBER:
        DCR D
        JZ NEXT_NUMBER          ;Αν D=0 παω στον επόμενο αριθμό (ο οποιος ειναι κατα 1 μικρλότερα
        RRC                     ;Παιρνω το τελυταιό ψηφιο απο τον αριθμό
        JNC LOOP_OF_NUMBER      ;Αν ειναι ασσος το μετραω, αλλιως προχωραω
ONES:
        INX B                   ;Συν ενας δυαδικος ασσος
        JMP LOOP_OF_NUMBER
NEXT_NUMBER:
        DCR L                   ;Μειώνω κατα 1 την διευθυνση (επόμενος αριθμος)
        JNZ MAIN_LOOP           ;Μεχρι να φτάσω στο 0
        ;MOV A,B        ; τυπωνουμε το Β
        ;CMA
        ;STA 3000H
        ;END
        ;MOV A,C        ; τυπωνουμε το C
        ;CMA
        ;STA 3000H
        ;END


; ΕΡΩΤΗΜΑ Γ
        MVI E,7FH
        MVI D,00H               ;Αποθηκευω στο D το πλήθος
        MOV A,M                 ;Ξεκινώ απο το 1ο στοιχείο (0 θεση M=0900H)
MAIN_LOOP_C:
        CPI 10H                 ;Αν ο αριθμός ειναι μικρότερος του 10Η πηγαινε στον επόμενο
        JC NEXT_NUMBER_C
        CPI 61H                 ;Αν εχω προσπεράσει τον 60Η ουσιαστικα εχω τελειώσει
        JNC AFTER_60H
        INR D
NEXT_NUMBER_C:
        INR L                   ;Επόμενος αριθμός
        MOV A,M
        JMP MAIN_LOOP_C
AFTER_60H:
        MOV A,D         ; τυπωνουμε το D
        CMA
        STA 3000H
        END
```

ΑΣΚΗΣΗ 2

Το πρόγραμμα που υλοποιήσαμε είναι το ακόλουθο:

```
        LXI B,0064H             ; 64H = 100 -> 100ms αρα 0.1s
        MVI D,C8H               ; D=200 για να έχω 200επανάληψεις->20δευτερόλεπτα
START:
        LDA 2000H               ;Φορτωνω στο A τα switches
        RLC                     ;Τσεκαρω το MSB επαναληπτικά μεχρι να ειναι OFF
        JNC EXER2_OFF           ;Αν CY=A7=0 πήγαινε στο OFF
        JMP START
EXER2_OFF:                      ;Τωρα ειναι OFF - Περιμένω το πρωτο ON οπου μετά θα περιμένω το OFF
        LDA 2000H
        RLC
        JC EXER2_ON             ;Τωρα περιμένω να γίνει ON και αν ναι πάω στο ON
        JMP EXER2_OFF
EXER2_ON:
        MVI D,C8H               ; D=200 για να έχω 200επανάληψεις->20δευτερόλεπτα
        LDA 2000H               ;Τωρα περιμένω το OFF για να ενεργοποιηθεί το push-button
        RLC
        JNC LOOP_20S
        JMP EXER2_ON

LOOP_20S:                       ;Loop για 20 δευτερόλεπτα
        LDA 2000H
        RLC
        JC ON_WHILE_LEDS
        JMP LED_AND_CHECK

LED_AND_CHECK:
        MVI A,00H               ;A=0 ωστε να ανάψουν όλα τα LEDs (ανάποδη λογική)
        STA 3000H
        CALL DELB               ;Καθυστέρηση
        DCR D
        JNZ LOOP_20S
        MVI A,FFH
        STA 3000H
        JMP EXER2_OFF           ; Αν τελειώσουν τα 20s κλείσε τα LED και πήγαινε στο OFF

ON_WHILE_LEDS:
        LDA 2000H
        RLC
        JNC RESTART_LOOP        ; Αν ξανέρθει OFF ξεκίνα τον timer απο την αρχή απο το RESTART_LOOP
        MVI A,00H               ; Αλλιώς συνεχίζει η λειτουργία του LOOP_20s, χωρίς το LED_AND_CHECK
        STA 3000H
        CALL DELB
        DCR D
        JNZ ON_WHILE_LEDS
        MVI A,FFH
        STA 3000H
        JMP EXER2_OFF
RESTART_LOOP:                   ;Απο την αρχή το delay routine των 20 δευτερολεπτων
        MVI D,C8H
        JMP LOOP_20S
        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 και 2 βίντεο στα οποία ελέγχουμε την ορθότητα του προγράμματος!!
   Επεξήγηση βίντεο:
   Στο 1ο βίντεο έχουμε ορίσει την ταχύτητα του προγράμματος πολύ γρήγορη, από την μπάρα πάνω δεξιά. Έτσι κάθε delay διαρκεί περίπου 3-4 δευτερόλεπτα και βλέπουμε ότι ενεργοποιώντας το push button πολλές φορές τα λαμπάκια δεν σταματάνε παρότι περνάνε 4 δευτερόλεπτα καθώς ο χρόνος ξεκινάει από την αρχή. Όταν σταματάμε να πειράζουμε το switch τότε μετά από 4 δευτερόλεπτα κλείνουν τα λαμπάκια. Στο 2ο βίντεο έχουμε ορίσει την ταχύτητα του προγράμματος περίπου κανονική και έτσι βλέπουμε περίπου 20 δευτερόλεπτα delay όπου τα λεντάκια είναι αναμμένα.

2. Κάθε φορά που ενεργοποιείται το push button γίνεται χρήση μιας ρουτίνας καθυστέρησης. Αυτή η ρουτίνα λειτουργεί εκτελώντας ένα μεγάλο loop (200 επαναλήψεις) όπου κάθε επανάληψη προσδίδει περίπου 0.1s. Έτσι συνολικά παίρνουμε τον χρόνο που ζητάμε (20s).

ΑΣΚΗΣΗ 3

ΕΡΩΤΗΜΑ i)

Ο κώδικας για το ερώτημα i είναι ο ακόλουθος:

```
START:
        MVI C,08H
        LDA 2000H
        MVI B,00H       ; B ειναι η θέση η οποία πρεπει να ανάψει
FIR_LOOP:
        RAR             ; δεξιά περιστροφη
        JC SEC_LOOP_HELP        ; jump αν βρηκαμε ασσο
        INR B
        DCR C
        JNZ FIR_LOOP    ; αν c!=0 ξανα το loop
RESTART:
        MVI A,FFH
        STA 3000H       ;Κανένα LED ανοικτό
        JMP START
SEC_LOOP_HELP:
        MVI A,FEH
SEC_LOOP:
        ;Τωρα το B δειχνει την θέση που πρέπει να ανάψει
        RLC
        DCR B
        JNZ SEC_LOOP
        ;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 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 το πρόγραμμα ήδη τρέχει και όλοι οι διακόπτες είναι στο off. Ανοίγουμε με την σειρά τους διακόπτες 8,6,5,3 σχηματίζοντας το παράδειγμα της εκφώνησης. Κάθε φορά που ανοίγουμε έναν διακόπτη παρατηρούμε ότι η αντίστοιχη θέση LED ανάβει, καθώς αυτός ο νέος διακόπτης είναι ο δεξιότερος. Στο τέλος έχουμε σχηματίσει τον αριθμό 10110100 και παρατηρούμε στα LED: OOOO OXOO όπου O=σβηστό και X=ανοικτό, όπως ακριβώς δείχνει το παράδειγμα.

ΕΡΩΤΗΜΑ ii)
Ο κώδικας για το ερώτημα ii είναι ο ακόλουθος:

```
START:
        CALL KIND
        CPI 00H         ;Ελεγχος =0
        JZ WRONG
        CPI 09H         ;Ελεγχος <9
        JNC WRONG
        MOV B,A
        MVI A,00H       ;A = 0
FIR_LOOP:
        DCR B
        JZ LEDS
        RLC             ;Βαζω 1 στις θέσεις του A που πρέπει
        INR A
        JMP FIR_LOOP
LEDS:
        STA 3000H
        JMP START
WRONG:
        MVI A,FFH       ;Κανένα LED ανοικτό
        STA 3000H
        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 το πρόγραμμα ήδη τρέχει και κανένα LED δεν είναι ανοικτό. Πατώντας τον αριθμό 4 του πληκτρολογίου παρατηρούμε ότι ανοίγουν τα LEDs της αντίστοιχης θέσης και όλα τα LED υψηλότερης θέσης από αυτό, δηλαδή τα LED 4,5,6,7,8. Έπειτα πατώντας τον αριθμό 9, όλα τα LEDs σβήνουν καθώς θέλουμε να βλέπουμε αποτέλεσμα μόνο για τους αριθμούς 1-8. Έπειτα πατάμε τους αριθμούς 6,5 όπου πάλι βλέπουμε σωστό αποτέλεσμα. Στην συνέχεια ελέγχουμε τις ακριανές περιπτώσεις, πατώντας στην αρχή το 8, όπου ανοίγει μόνο το 8ο (MSB) LED και τον αριθμό 1 όπου ανοίγουν όλα τα LEDs. Τέλος με τον αριθμό 0 που είναι εκτός του πεδίου μας, όλα τα LEDs σβήνουν.
2. Στο πρόγραμμα χρησιμοποιούμε την ρουτίνα KIND με την εντολή CALL KIND

ΕΡΩΤΗΜΑ iii)

Ο κώδικας για το ερώτημα iii είναι ο ακόλουθος:

```
START:
        IN 10H
        LXI H,0A00H     ;Θεση μνήμης 0A00H
        MVI B,04H       ;B=4
FIR_LOOP:               ;Αρχικοποίηση της μνήμης / Χωρίς αυτο εμφανίζει και τα 6 display
        MVI M,10H                               ; Εγω θέλω μόνο τα πρώτα 2
        INX H
        DCR B
        JNZ FIR_LOOP
;Σαρωση για κάθε γραμμή
LINE0:
        MVI A,F7H       ;11110111 : πορτα σάρωσης
        STA 2800H
        LDA 1800H       ;Διαβάζω τις στήλες
        ANI 07H         ;Κρατάω τα 3 LSB, τα υπόλοιπα δεν με ενδιαφέρουν
        MVI D,01H       ;(1)
        CPI 06H         ;xxxxx110
        JZ DISPLAY
        MVI D,02H       ;(2)
        CPI 05H         ;xxxxx101
        JZ DISPLAY
        MVI D,03H       ;(3)
        CPI 03H         ;xxxxx011
        JZ DISPLAY
LINE1:
        MVI A,EFH       ;11101111
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI D,04H
        CPI 06H         ;(4)
        JZ DISPLAY
        MVI D,05H
        CPI 05H         ;(5)
        JZ DISPLAY
        MVI D,06H
        CPI 03H         ;(6)
        JZ DISPLAY
LINE2:
        MVI A,DFH       ;11011111
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI D,07H
        CPI 06H         ;(7)
        JZ DISPLAY
        MVI D,08H
        CPI 05H         ;(8)
        JZ DISPLAY
        MVI D,09H
        CPI 03H         ;(9)
        JZ DISPLAY
LINE3:
        MVI A,BFH       ;10111111
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI D,0AH
        CPI 06H         ;(A)
        JZ DISPLAY
        MVI D,0BH
        CPI 05H         ;(B)
        JZ DISPLAY
        MVI D,0CH
        CPI 03H         ;(C)
        JZ DISPLAY
```

```
LINE4:
        MVI A,7FH       ;01111111
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI D,0DH
        CPI 06H         ;(D)
        JZ DISPLAY
        MVI D,0EH
        CPI 05H         ;(E)
        JZ DISPLAY
        MVI D,0FH
        CPI 03H         ;(F)
        JZ DISPLAY

;Πρεπει να κάνω το ίδιο και για τα υπόλοιπα κουμπιά
LINE5:
        MVI A,FBH       ;11111011
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI D,00H
        CPI 06H         ;(0)
        JZ DISPLAY
        MVI D,10H
        CPI 05H         ;(STORE/INCR)
        JZ DISPLAY
        MVI D,11H
        CPI 03H         ;(DECR)
        JZ DISPLAY
LINE6:
        MVI A,FEH       ;11111110
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI D,12H       ;(INSTR_STEP)
        CPI 06H
        JZ DISPLAY
        MVI D,13H       ;(FETCH_PC)
        CPI 05H
        JZ DISPLAY

LINE7:
        MVI A,FDH       ;11111101
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI D,14H
        CPI 06H         ;(RUN)
        JZ DISPLAY
        MVI D,15H
        CPI 05H         ;(FETCH_REG)
        JZ DISPLAY
        MVI D,16H
        CPI 03H         ;(FETCH_ADDRS)
        JZ DISPLAY
        JMP START
DISPLAY:
        LXI H,0A04H     ;θεση των LSB
        MOV A,D
        ANI 0FH         ;4LSB
        MOV M,A
        INX H
        MOV A,D
        ANI F0H         ;4MSB
        RLC
        RLC
        RLC
        RLC
        MOV M,A
        LXI D,0A00H
        CALL STDM
        CALL DCD
        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 τρέχουμε το πρόγραμμα από την αρχή πατώντας FETCH PC και μετά RUN, και κατευθείαν φαίνεται ο αριθμός 14 στον οποίο αντιστοιχεί. Έπειτα πατάμε στο πληκτρολόγιο τους αριθμούς 4,8,D,C,6 και 0 και βλέπουμε να εμφανίζονται σωστά στο 7-segment-display. Έπειτα πατάμε τα κουμπιά STORE/INC, FETCH REG, RUN, DECR και FETCH PC και βλέπουμε να εμφανίζονται οι αριθμοί 10, 15, 14, 11 και 13 τους οποίους εμείς αντιστοιχίσαμε σε αυτά τα πλήκτρα.
2. Το πρόγραμμα είναι τόσο μεγάλο καθώς δεν χρησιμοποιούμε την ρουτίνα KIND. Υλοποιούμε ολόκληρη την διαδικασία ανάγνωσης με βάση το εγχειρίδιο χρήσης του προσομοιωτή (συγκεκριμένα σελίδες 76 έως 82). Χρησιμοποιώντας αυτή την προσέγγιση χωρίζουμε το πρόγραμμα σε 3 μέρη.
   1ο: Ανάγνωση γραμμής
   Το πληκτρολόγιο χωρίζεται σε γραμμές όπου κάθε γραμμή περιέχει συγκεκριμένα πλήκτρα. Σε κάθε επανάληψη ελέγχουμε όλες τις γραμμές για να δούμε σε ποια από αυτές υπήρξε χρήση του πληκτρολογίου
   2ο: Εύρεση στήλης
   Σε κάθε γραμμή ελέγχουμε όλες τις στήλες, ελέγχοντας ουσιαστικά κάθε πλήκτρο. Όταν βρούμε σε ποια στήλη αντιστοιχίζεται το πλήκτρο που πατήθηκε αποθηκεύουμε μια συγκεκριμένη τιμή σε έναν καταχωρητή
   3ο: Εμφάνιση αποτελέσματος
   Εμφανίζουμε στο 7-segment-display το πλήκτρο το οποίο πατήθηκε. Αν το πλήκτρο αυτό δεν ήταν κάποιο από τα αριθμήσιμα στο 16 σύστημα πλήκτρα, τότε εκτυπώνουμε έναν άλλο αριθμό τον οποίο εμείς ορίζουμε, φτιάχνοντας μια αντιστοιχία με τα εξτρά πλήκτρα και κάποιους αριθμούς.
3. Για την εύρεση στήλης χρησιμοποιούμε μόνο τα 3LSB, με βάση το εγχειρίδιο, επομένως ελέγχουμε τις στήλες με τους αριθμούς 06H, 05H, 03H. Επομένως σε κάθε γραμμή μπορούμε να κάνουμε 3 ελέγχους
4. Τα πλήκτρα εκτός των δεκαεξαδικων αριθμών (πέρα του 0) ανήκουν σε διαφορετικές γραμμές
5. Για το 7-segment-display χρησιμοποιούμε τις ρουτίνες STDM και DCD με βάση την θέση μνήμης 0A00H. Επίσης εφόσον θέλουμε μόνο τα πρώτα 2 7-segment-diplays χρησιμοποιούμε ένα loop στην αρχή όπου μηδενίζουμε όλα τα υπόλοιπα

ΑΣΚΗΣΗ 4

Ο κώδικας για την άσκηση 4 είναι ο εξής:

```
START:
        LDA 2000H
        MOV B,A         ;Αποθηκεύω το input απο τα switches
CHECK_1:        ;A7xorA6
        XRI 80H         ;Axor10000000
        MOV C,A         ;C = A
        MOV A,B
        XRI 40H         ;A
        RLC
        XRA C           ;A XOR C
        ANI 80H
        MOV D,A         ;D7
CHECK_2:        ;A5xorA4
        MOV A,B
        XRI 20H         ;
        MOV C,A         ;C = A = xxAxxxx
        MOV A,B
        XRI 10H         ;A
        RLC
        XRA C           ;A XOR C
        ANI 20H
        RLC
        RLC
        MOV E,A
        JMP OR1
CHECK_3:        ;A3andA2
        MOV A,B
        ANI 08H         ;
        MOV C,A         ;C = A
        MOV A,B         ;A is equal to the input of dip switches
        ANI 04H         ;A
        RLC
        ANA C           ;A3 AND A2 / RESULT IN A3
        ANI 08H
        RLC
        RLC
        XRA D
        MOV D,A         ;D5 OK

CHECK_4:        ;A1andA0
        MOV A,B
        ANI 02H         ;
        MOV E,A         ;C = A
        MOV A,B         ;A is equal to the input of dip switches
        ANI 01H         ;A
        RLC
        ANA E           ;A1 AND A0
        ANI 02H
        MOV E,A ;E1
        JMP OR2
OR1:    ;Αποθηκευση D7 και D6
        MOV A,E
        ANI 80H
        ORA D           ; A7
        RRC
        XRA D
        MOV D,A
        JMP CHECK_3
OR2:    ;Αποθηκευση D5 και D4
        MOV A,E         ;D5 ηδη αποθηκευμενο
        ANI 02H
        RLC
        RLC
        RLC
        RLC             ; GO TO D5
        ANI 20H
        ORA D
        RRC
        ANI 10H
        XRA D
        MOV D,A

LEDS:
        MOV A,D         ;Μετακίνηση των D7D6D5D4 στα LSB
        RLC
        RLC
        RLC
        RLC
        ANI 0FH
        CMA
        STA 3000H
        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 το πρόγραμμα τρέχει ήδη με αρχική διάταξη bit εισόδου 0000 0000. Έτσι κανένα LED δεν είναι ανοικτό. Στην συνέχεια πηγαίνουμε από πύλη σε πύλη και δοκιμάζουμε και τις 4 πιθανές περιπτώσεις για τα bit εισόδου: 00, 01, 11, 10 και παρατηρούμε το αποτέλεσμα στα LEDs. Το πρόγραμμα δουλεύει ακριβώς όπως ορίζεται από το I.C. της εκφώνησης.
   Από αριστερά προς τα δεξιά:
   Στην 1η πύλη: Δίνει 1 όταν η είσοδος είναι 01 ή 10 και ανάβουν το 4ο και το 3ο LED
   Στην 2η πύλη: Δίνει 1 όταν η είσοδος είναι 01 ή 10 και ανάβει το 3ο LED
   Στην 3η πύλη: Δίνει 1 όταν η είσοδος είναι 11 και ανάβουν το 2ο και το 1ο LED
   Στην 4η πύλη: Δίνει 1 όταν η είσοδος είναι 11 και ανάβει το 1ο LED
2. Αναλύοντας το υποθετικό I.C. καταλήγουμε ότι χρειάζεται να υλοποιήσουμε 4 ελέγχους: 1. A3xorB3 2. A2xorB2 3. A1andB1 4. A0andB0. Έπειτα πρέπει να υλοποιήσουμε και τις 2 OR πύλες. Κάθε πύλη επομένως εξετάζεται και σε διαφορετικό LABEL του προγράμματος.
3. Η λογική του προγράμματος είναι σε κάθε έλεγχο η απομόνωση των bit που θα χρησιμοποιήσουμε για τις πράξεις και η αποθήκευση του αποτελέσματος σε έναν καταχωρητή που κάθε φορά τον ανανεώνουμε κάνοντας xor με τον εαυτό του.
4. Τα αποτελέσματα από τους ελέγχους 1 και 3 αποθηκεύονται κατευθείαν στο καταχωρητή D στα D7 και D5 ενώ τα D6 και D4 προκύπτουν από τις 2 OR. Έτσι ο D περιέχει στα MSB του το αποτέλεσμα που πρέπει να εμφανιστεί στα LED. Πριν το εμφανίσουμε ωστόσο το μεταφέρουμε στα 4 LSB σύμφωνα με την εκφώνηση.

Σχόλιο:
Όλοι οι κώδικες περιέχονται σε αρχεία .8085 στο παραδοτέο zip.

ΘΕΩΡΗΤΙΚΈΣ ΑΣΚΗΣΕΙΣ

ΑΣΚΗΣΗ 5

Η οργάνωση που παρουσιάζεται αφορά μνήμη SRAM με μέγεθος 256x4bit. Άρα καταλαβαίνουμε ότι η μνήμη θα έχει χώρο για 256 λέξεις με μέγεθος 4bit. Για αυτό μπορούμε να χωρίσουμε την μνήμη σε 4 τμήματα, το κάθε τμήμα θα έχει μέγεθος 256bit. Με αυτόν το σχεδιασμό κάθε λέξη (4bit) θα διαμοιράζεται και στα 4 αυτά τμήματα, ένα bit σε κάθε τμήμα.
Το κάθε τμήμα θα έχει δύο καταστάσεις α και β, τέτοιες ώστε αxβ=256. Επιλέγουμε α=16 και β=16 (16x16) με σκοπό να επιτύχουμε τετραγωνικό σχήμα.

Για επιλογή γραμμής χρειάζομαι: $\log_2 16$=4bit διεύθυνσης όμοια και
Για επιλογή στήλης χρειάζομαι: $\log_2 16$=4bit διεύθυνσης

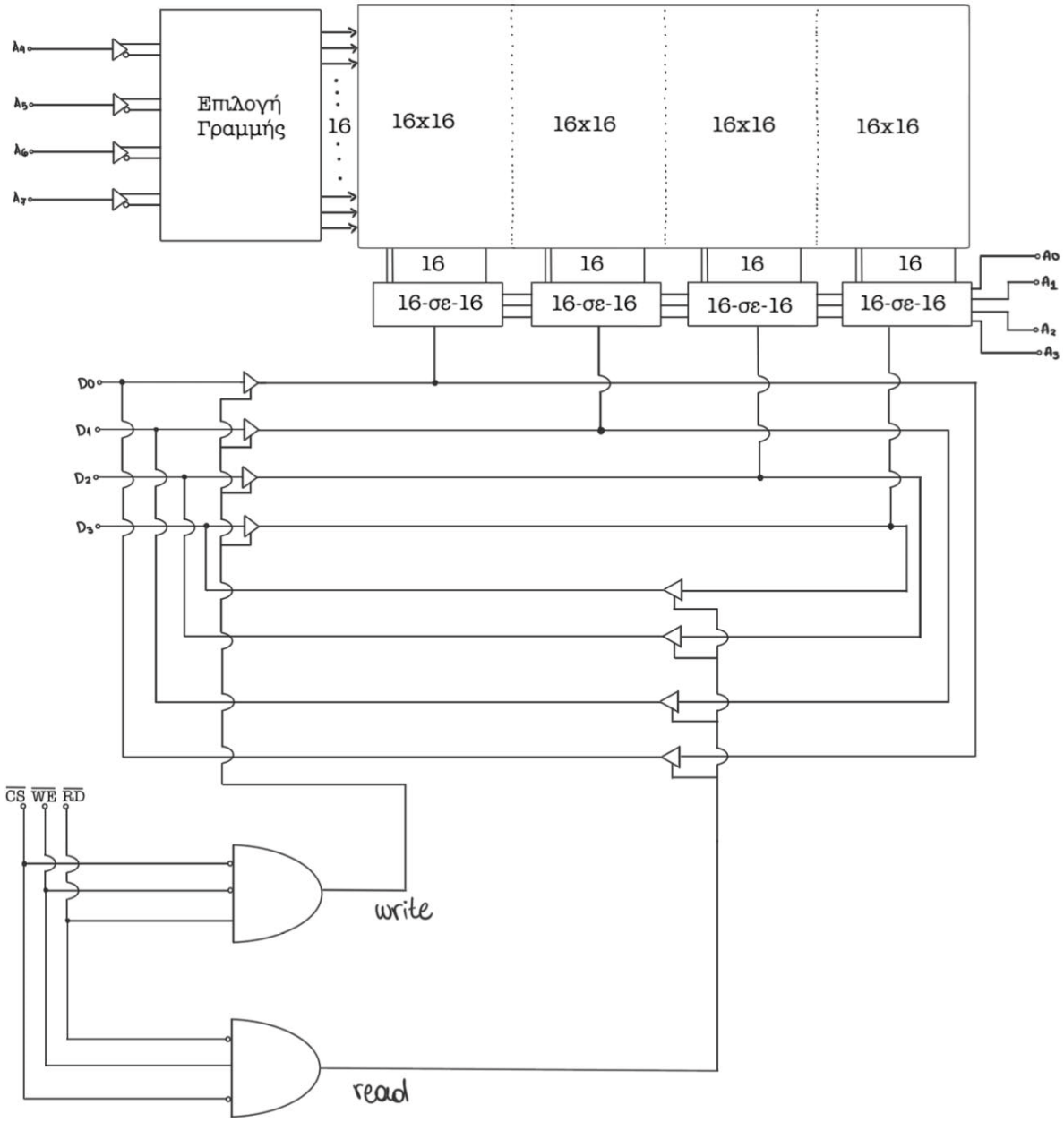
Οι ακροδέκτες διευθύνσεων A_0 μέχρι A_3 χρησιμοποιούνται για την επιλογή στήλης με χρήση πολυπλεκτών-αποπλεκτών 4-σε-16 και οι ακροδέκτες διευθύνσεων A_4 μέχρι A_7 χρησιμοποιούνται για την επιλογή γραμμής με χρήση πολυπλεκτών-αποπλεκτών 4-σε-16. Οι ακροδέκτες D_0 μέχρι D_3 είναι οι ακροδέκτες εισόδου/εξόδου 4bit, ενώ τα σήματα, μέσω των αντίστοιχων ακροδεκτών επιτρέπουν ή αποτρέπουν τις λειτουργίες εγγραφής και ανάγνωσης.

Παράδειγμα:
Εγγραφή της λέξης 1011 στην θέση μνήμης 01110010 και διάβασμα της λέξης στην θέση μνήμης 01000100.

Για την εγγραφή:
Αρχικά, στις εισόδους διεύθυνσης A_0 μέχρι A_7 εφαρμόζεται η διεύθυνση εγγραφής (01110010). Εφαρμόζοντας τα δεδομένα για εγγραφή (λέξη -> 1011) στους ακροδέκτες της εισόδου D_0 μέχρι D_3.
Με σκοπό να γίνει εγγραφή στην μνήμη πρέπει τα σήματα CS' και WE' να είναι ενεργοποιημένα, δηλαδή αρνητικό σήμα 0 αφού είναι αντεστραμμένα, ενώ το σήμα RD' είναι απενεργοποιημένο δηλαδή θετικό σήμα 1. Με αυτό τον τρόπο η έξοδο της πάνω πύλη AND (write) γίνεται 1 και η έξοδο της κάτω πύλης AND (write) γίνεται 0. Επιλέγεται με αυτό τον τρόπο ότι θα γίνει εγγραφή στην μνήμη και όχι ανάγνωση από αυτή. Η πάνω πύλη δίνει επίτρεψη 1 στα τέσσερα αριστερά τρισταθή buffers, έτσι τα D_0, D_1, D_2, D_3 πηγαίνουν στους αντίστοιχους πολυπλέτες, που εδώ λειτουργούν ως αποπλέκτες. Τα σήματα επιλογής A_0 μέχρι A_3 οδηγούν το καθένα στην στήλη 0111, δηλαδή το δεκαδικό 7 και τα σήματα επιλογής A_4 μέχρι A_7 οδηγούν το καθένα στην γραμμή 0010, δηλαδή το δεκαδικό 2.
Άρα το δεδομένο D_0, δηλαδή το πρώτο bit της λέξης, εγγράφεται στο πρώτο τμήμα της μνήμης στην θέση (2,7), το δεδομένο D_1, δηλαδή το δεύτερο bit της λέξης, εγγράφεται στο δεύτερο τμήμα της μνήμης στην θέση (2,7), το δεδομένο D_2, δηλαδή το τρίτο bit της λέξης, εγγράφεται στο τρίτο τμήμα της μνήμης στην θέση (2,7), το δεδομένο D_3, δηλαδή το τρίτο bit της λέξης, εγγράφεται στο τέταρτο τμήμα της μνήμης στην θέση (2,7).

Για το διάβασμα:
Αρχικά, στις εισόδους διεύθυνσης A_0 μέχρι A_7 εφαρμόζεται η διεύθυνση διαβάσματος (01000100). Στην συγκεκριμένη περίπτωση πρέπει τα σήματα CS' και RD' να είναι ενεργοποιημένα, δηλαδή αρνητικό σήμα 0 αφού είναι αντεστραμμένα, ενώ το σήμα WE' είναι απενεργοποιημένο δηλαδή θετικό σήμα 1. Με αυτό τον τρόπο η έξοδο της πάνω πύλη AND (write) γίνεται 0 και η έξοδο της κάτω πύλης AND (write) γίνεται 1. Επιλέγεται με αυτό τον τρόπο ότι θα γίνει ανάγνωση από την μνήμη και όχι εγγραφή σε αυτή.
Τα σήματα A_4 μέχρι A_7 επιλέγουν τη αντιστοιχεί στον ελαχιστόρο 0100, δηλαδή στην σειρά 4 στο δεκαδικό, ενώ τα σήματα επιλογής A_0 μέχρι A_3 επιλέγουν τη αντιστοιχεί στον 0100, δηλαδή στην γραμμή 4 στο δεκαδικό. Με αυτό τον τρόπο τα σήματα των bits της 4η γραμμής της 4η στήλης φτάνουν στις εξόδους των πολυπλεκτών-αποπλέκτων (που λειτουργούν ως πολυπλέκτη), και φτάνουν στις εξόδους D_0 μέχρι D_3.

Σε περίπτωση σφάλματος, όπου στέλνεται αρνητικός παλμός ταυτόχρονα στους ακροδέκτες και , τότε τα σήματα αλληλοακυρώνονται.


Επιλογή
Γραμμής
16
16x16
16x16
16x16
16x16
16
16
16
16
16-σε-16
16-σε-16
16-σε-16
16-σε-16
A0
A1
A2
A3
D0
D1
D2
D3
CS WE RD
write
read

ΑΣΚΗΣΗ 6

Για τον Χάρτης Μνήμης:
ROM1: 2k x 8bit = 2k byte
ROM2: 2k x 8bit = 2k byte
ROM3: 4k x 8bit = 4k byte
SRAM1: 2k x 8bit = 2k byte
SRAM2 2k x 8bit = 2k byte

Εφόσον, στο σχεδιαζόμενο σύστημα συμμετέχει ο μΕ 8085, η αναπαράσταση των δεδομένων απαιτεί 8bits, οι λέξεις έχουν μήκος 8bit = 1byte.

- Οι λέξεις που αποθηκεύονται στην ROM1 είναι σε πλήθος 2k = 2^{11} λέξεις, επομένως απαιτούνται 11bytes ($A_0 – A_{10}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στην ROM2 είναι σε πλήθος 2k = 2^{11} λέξεις, επομένως απαιτούνται 11bytes ($A_0 – A_{10}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στην ROM3 είναι σε πλήθος 4k = 2^{12} λέξεις, επομένως απαιτούνται 12bytes ($A_0 – A_{11}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στην SRAM1 είναι σε πλήθος 2k = 2^{11} λέξεις, επομένως απαιτούνται 11bytes ($A_0 – A_{10}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στην SRAM1 είναι σε πλήθος 2k = 2^{11} λέξεις, επομένως απαιτούνται 11bytes ($A_0 – A_{10}$) για την διευθυνσιοδότησή τους

Με βάση αυτά τα δεδομένα σχεδιάζεται ο χάρτης μνήμης

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	0000	ROM1 – 2k
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	07FF	
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0800	ROM2 – 2k
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFF	
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000	ROM3 – 4k
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000	SRAM1 – 2k
0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	27FF	
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2800	SRAM2 – 2k
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	2FFF	

Παρατηρούμε ότι τα bit A_{15} και A_{14} δεν χρησιμοποιούνται για τον προσδιορισμό κάποιας θέσης μνήμης, έχουν για όλες την τιμή 00. Χρησιμοποιούνται, λοιπόν, ως επίτρεψη στον αποκωδικοποιητή (1η υλοποίηση) ή στις λογικές πύλες του δευτέρου τρόπου.

Τα bit A_{11}, A_{12} και A_{13} χρησιμοποιούνται για την επιλογή του επιθυμητού ολοκληρωμένου (ROM1, ROM2,ROM3, SRAM1 ή SRAM2) καθώς συνδυασμοί αυτών προσδιορίζουν μοναδικά τις περιοχές μνήμης που αντιστοιχούν στο κάθε ολοκληρωμένο.

$A_{11}A_{12}A_{13}$ = 000 -> ROM1
$A_{11}A_{12}A_{13}$ = 001 -> ROM2
$A_{11}A_{12}A_{13}$ = 010 ή $A_{11}A_{12}A_{13}$ = 011 -> ROM3

$A_{11}A_{12}A_{13}$ = 100 ->SRAM1
$A_{11}A_{12}A_{13}$ = 101 -> SRAM2

α) Χρήση αποκωδικοποιητή και πυλών:
Γίνεται χρήση αποκωδικοποιητή 3-σε-8 [74LS138]. Ο αποκωδικοποιητής παίρνει ως εισόδους τα A_{11}, A_{12}, A_{13} και τα A_{14}, A_{15} ως επιτρέψεις. Έτσι, μόνο όταν $A_{14} = A_{15} = 0$ θα μπορεί μέσω του αποκωδικοποιητή να γίνει επιλογή μνήμης.

Ανάλογα, με τον συνδυασμό των A_{11}, A_{12}, A_{13} όλες οι έξοδοι του αποκωδικοποιητή γίνονται 1 εκτός από αυτή που αντιστοιχεί στο συμπλήρωμα του ελαχιστόρου που καθορίζει η είσοδος.

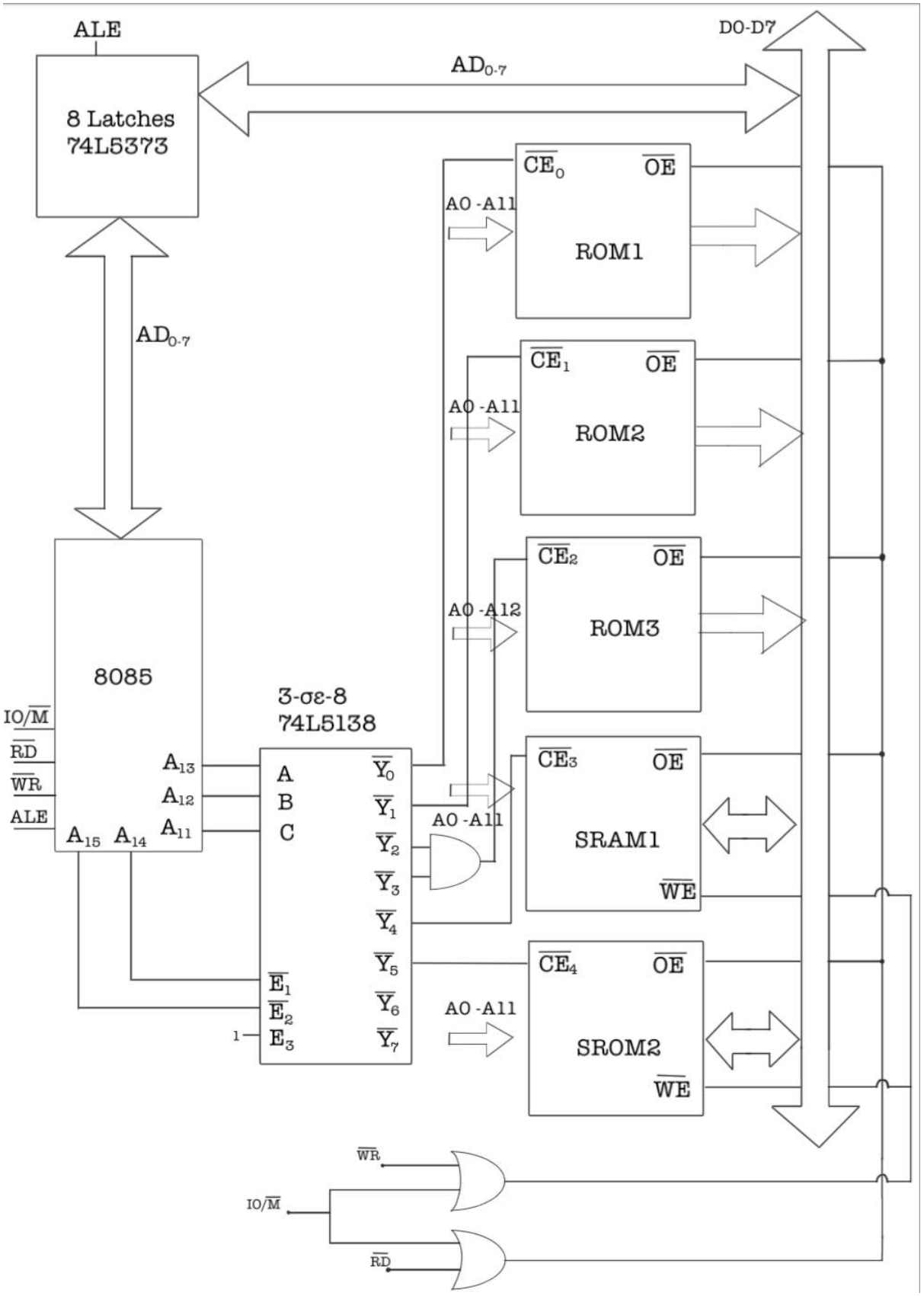
$CE_0 = 1$ ($\overline{CE_0} = 0$) όταν $A_{11}A_{12}A_{13} = 000$, δηλαδή όταν $Y_0 = 1 \Rightarrow \bar{Y}_0 = 0$
$\Rightarrow \overline{CE_0} = \bar{Y}_0$

$CE_1 = 1$ ($\overline{CE_1} = 0$) όταν $A_{11}A_{12}A_{13} = 001$, δηλαδή όταν $Y_1 = 1 \Rightarrow \bar{Y}_1 = 0$
$\Rightarrow \overline{CE_1} = \bar{Y}_1$

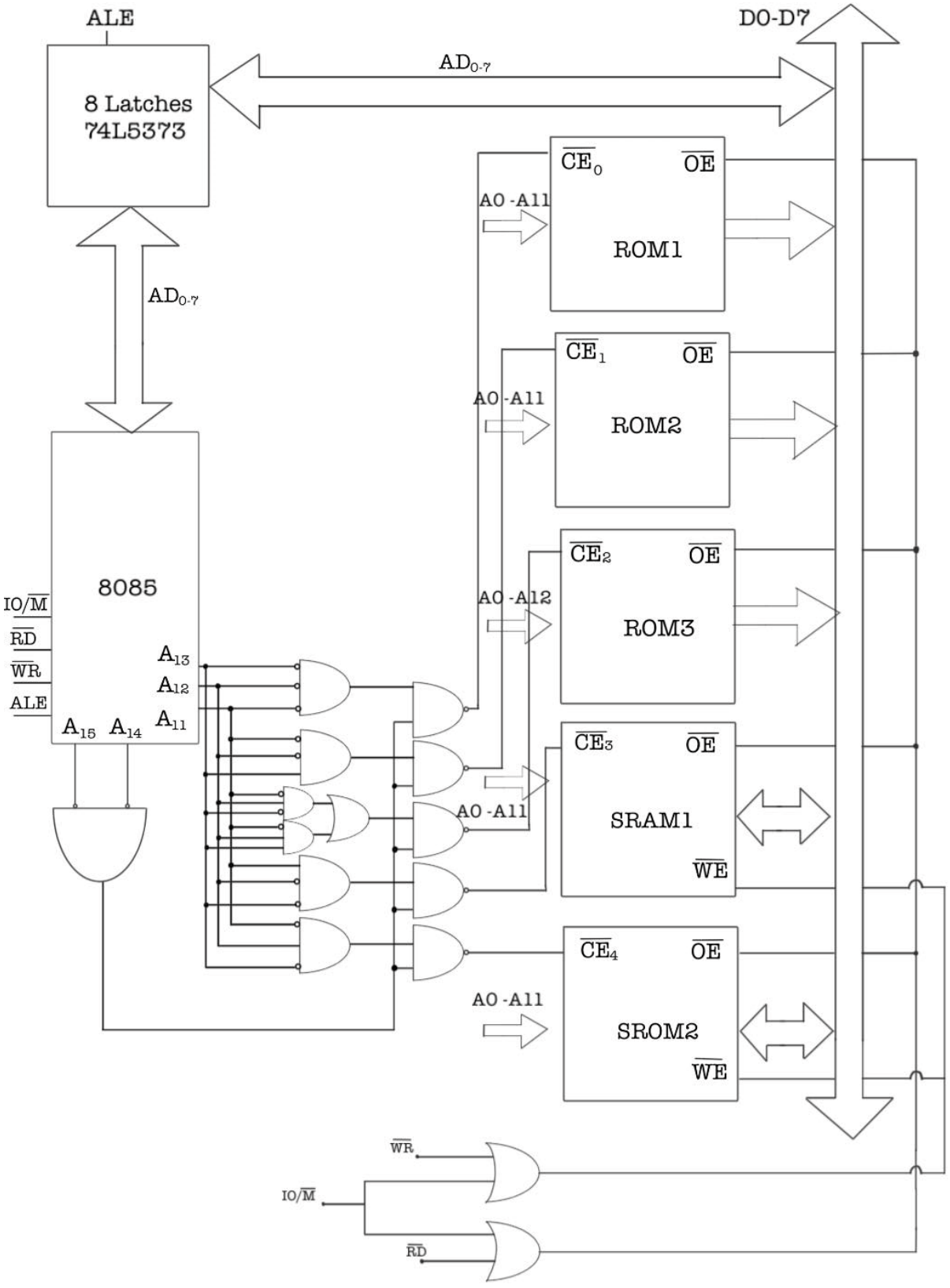
$CE_2 = 1$ ($\overline{CE_2} = 0$) όταν $A_{11}A_{12}A_{13} = 010$ και $A_{11}A_{12}A_{13} = 011$, δηλαδή όταν
$Y_2 + Y_3 = 1 \Rightarrow \bar{Y}_2\bar{Y}_3 = 0$
$\Rightarrow \overline{CE_2} = \bar{Y}_2\bar{Y}_3$

$CE_3 = 1$ ($\overline{CE_3} = 0$) όταν $A_{11}A_{12}A_{13} = 100$, δηλαδή όταν $Y_4 = 1 \Rightarrow \bar{Y}_4 = 0$
$\Rightarrow \overline{CE_3} = \bar{Y}_4$

$CE_4 = 1$ ($\overline{CE_4} = 0$) όταν $A_{11}A_{12}A_{13} = 101$, δηλαδή όταν $Y_5 = 1 \Rightarrow \bar{Y}_5 = 0$
$\Rightarrow \overline{CE_4} = \bar{Y}_5$


ALE
8 Latches
74L5373
AD0-7
D0-D7
CE0
OE
A0 -A11
ROM1
AD0-7
CE1
OE
A0 -A11
ROM2
CE2
OE
A0 -A12
ROM3
8085
IO/M
RD
WR
ALE
A15
A14
A13
A12
A11
3-σε-8
74L5138
A
B
C
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
E1
E2
E3
1
A0 -A11
CE3
OE
SRAM1
WE
CE4
OE
A0 -A11
SROM2
WE
WR
IO/M
RD

β) Χρήση αποκλειστικά λογικών πυλών:
Χρήση κατάλληλου συνδυασμού πυλών ώστε να φτάνει αρνητικός παλμός μόνο στο chip που θέλουμε κάθε φορά να προσπελάσουμε



ΑΣΚΗΣΗ 7

Δίνονται τα ακόλουθα ολοκληρωμένα:

- ROM 16kbytes - εικονικά θα μοιραστεί σε δύο περιοχές των 8kbytes
- RAM 12kbytes - 3 RAM των 4kbytes

Μνήμη ROM των 16kB, της οποίας τα πρώτα 8kB θα αντιστοιχιστούν στις θέσεις μνήμης 0000H έως 1FFFH και τα υπόλοιπα 8kB θα αντιστοιχιστούν στις θέσεις μνήμης 4000H έως 6FFFH.
Μνήμη RAM των 4kB αντιστοιχιζόμενη στις θέσεις μνήμης 1000H έως 1FFFH.
Μνήμη RAM των 8kB αντιστοιχιζόμενη στις θέσεις μνήμης 2000H έως 3FFFH.

- Οι λέξεις που αποθηκεύονται στην ROM1 είναι σε πλήθος 8k = 2^{13} λέξεις, επομένως απαιτούνται 13bytes ($A_0 – A_{12}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στην RAM1 είναι σε πλήθος 4k = 2^{12} λέξεις, επομένως απαιτούνται 12bytes ($A_0 – A_{11}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στην RAM2 είναι σε πλήθος 4k = 2^{12} λέξεις, επομένως απαιτούνται 12bytes ($A_0 – A_{11}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στην RAM3 είναι σε πλήθος 4k = 2^{12} λέξεις, επομένως απαιτούνται 12bytes ($A_0 – A_{11}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στην ROM2 είναι σε πλήθος 8k = 2^{13} λέξεις, επομένως απαιτούνται 13bytes ($A_0 – A_{12}$) για την διευθυνσιοδότησή τους

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	0000	ROM – 8k
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000	RAM1 – 4k
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	2FFF	
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3000	RAM2 – 4k
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2000	RAM3 – 4k
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	4FFF	
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	5000	ROM – 8k
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	6FFF	

Παρατηρούμε ότι το bit A_{15} δεν χρησιμοποιείται για τον προσδιορισμό κάποιας θέσης μνήμης (έχει για όλες τις θέσεις μνήμης τιμή 0). Μπορούμε λοιπόν να το χρησιμοποιήσουμε ως επίτρεψη στον αποκωδικοποιητή που επιλέγει το ολοκληρωμένο για προσπέλαση.
Τα bit A_{12}, A_{13}, A_{14} χρησιμοποιούνται για την επιλογή του επιθυμητού ολοκληρωμένου (ROM, RAM1 ή RAM2) καθώς συνδυασμοί αυτών προσδιορίζουν μοναδικά τις περιοχές μνήμης που αντιστοιχούν στο κάθε ολοκληρωμένο.

$A_{12}A_{13}A_{14}$ = 000 και $A_{12}A_{13}A_{14}$ = 001 και $A_{12}A_{13}A_{14}$ = 101 -> ROM
$A_{12}A_{13}A_{14}$ = 010 -> RAM1
$A_{12}A_{13}A_{14}$ = 011 -> RAM2
$A_{12}A_{13}A_{14}$ = 100 και $A_{12}A_{13}A_{14}$ = 110 ->RAM3
Ο αποκωδικοποιητής παίρνει ως εισόδους τα A_{12}, A_{13}, A_{14} και τα A_{15} ως επίτρεψη. Έτσι, μόνο όταν A_{15} = 0 θα μπορεί μέσω του αποκωδικοποιητή να γίνει επιλογή μνήμης. Ανάλογα,

με τον συνδυασμό των A_{12}, A_{13}, A_{14} όλες οι έξοδοι του αποκωδικοποιητή γίνονται 1 εκτός από αυτή που αντιστοιχεί στο συμπλήρωμα του ελαχιστόρου που καθορίζει η είσοδος.

$CE_0 = 1$ ($\overline{CE_0} = 0$) όταν $A_{12}A_{13}A_{14} = 000$, $A_{12}A_{13}A_{14} = 001$ και $A_{12}A_{13}A_{14} = 101$, δηλαδή όταν $Y_0 + Y_1 + Y_4 = 1 \Rightarrow \bar{Y}_0\bar{Y}_1\bar{Y}_4 = 0$

$\Rightarrow \overline{CE_0} = \bar{Y}_0\bar{Y}_1\bar{Y}_4$

$CE_1 = 1$ ($\overline{CE_1} = 0$) όταν $A_{12}A_{13}A_{14} = 010$, δηλαδή όταν $Y_2 = 1 \Rightarrow \bar{Y}_2 = 0$

$\Rightarrow \overline{CE_1} = \bar{Y}_2$

$CE_2 = 1$ ($\overline{CE_2} = 0$) όταν $A_{12}A_{13}A_{14} = 011$, δηλαδή όταν $Y_3 = 1 \Rightarrow \bar{Y}_3 = 0$

$\Rightarrow \overline{CE_2} = \bar{Y}_3$

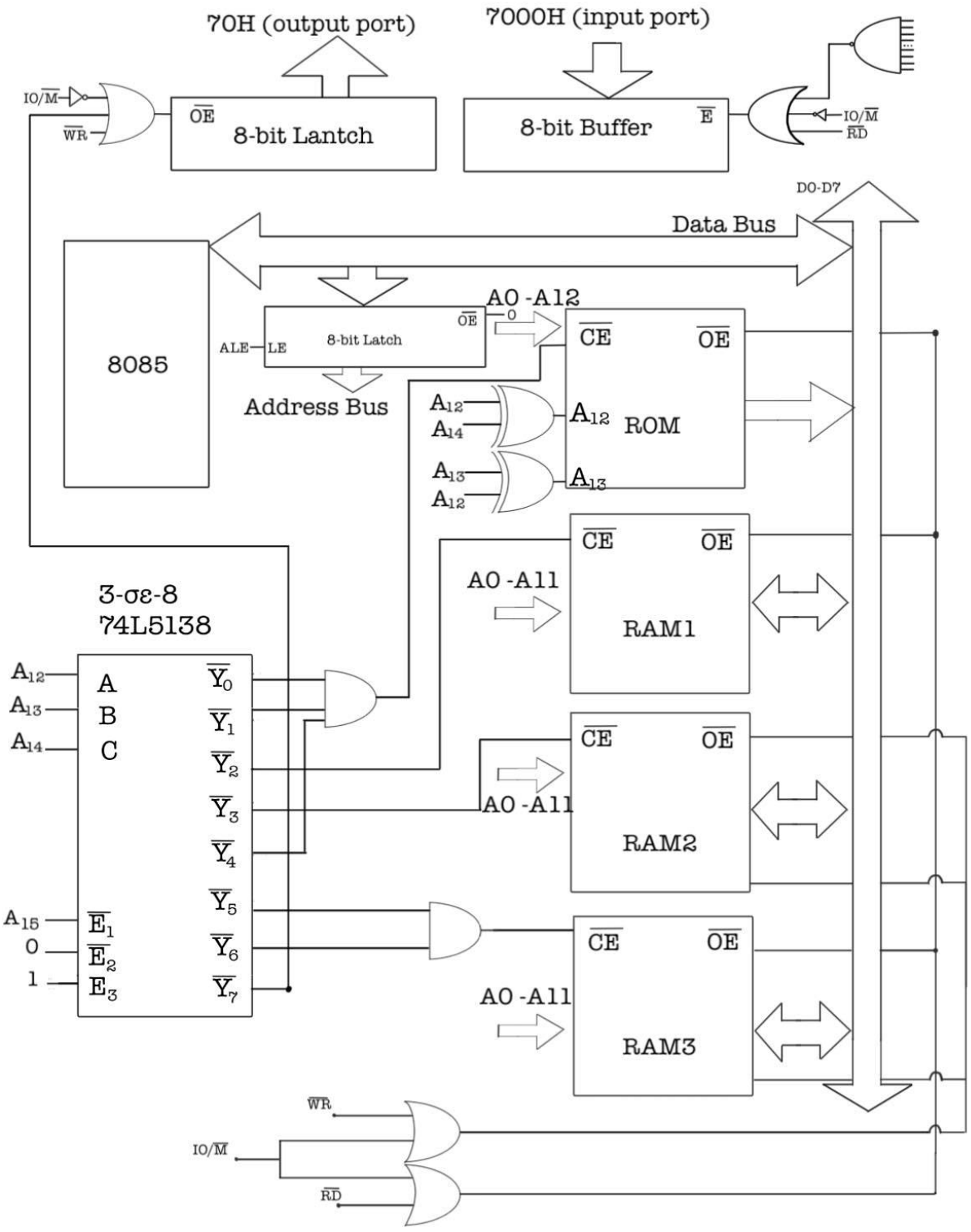
$CE_3 = 1$ ($\overline{CE_3} = 0$) όταν $A_{12}A_{13}A_{14} = 100$ και $A_{12}A_{13}A_{14} = 110$,
δηλαδή όταν $Y_5 + Y_6 = 1 \Rightarrow \bar{Y}_5\bar{Y}_6 = 0$

$\Rightarrow \overline{CE_3} = \bar{Y}_5\bar{Y}_6$

Για την επίτρεψη του Latch της θύρας εισόδου 7000H κάνουμε χρήση πύλης AND με 16 εισόδους, η οποία δίνει στην έξοδο 1 αν:
$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0 = 0111000000000000$

Για την επίτρεψη του Latch της θύρας εισόδου 70H κάνουμε χρήση του Y_7. Η μνήμη ROM λαμβάνει τα bits A_0 έως A_{11} από το address bus και τα bit A_{12} και A_{13} από τις εξόδους των πυλών XOR που φαίνονται στο παρακάτω σχήμα.

Αυτό συμβαίνει, ώστε τα bit A_{12} και A_{13} να μετατρέπουν τις διευθύνσεις του χάρτη μνήμης που αντιστοιχούν σε θέσεις της ROM που δεν είναι στο πρώτο τμήμα της (0000H – 1FFFH) σε συνεχόμενες θέσεις εσωτερικά στο ολοκληρωμένο (δηλαδή γίνεται αντιστοίχιση της διεύθυνσης 5000H έως 6FFFH του χάρτη μνήμης στις διευθύνσεις 2000H έως 3FFFH της ROM).


70H (output port)
7000H (input port)
IO/M
WR
OE
8-bit Lantch
8-bit Buffer
E
IO/M
RD
D0-D7
Data Bus
8085
A0 -A12
OE
0
ALE
LE
8-bit Latch
Address Bus
CE
OE
A12
A14
A12
ROM
A13
A12
A13
3-σε-8
74L5138
A0 -A11
RAM1
A12
A13
A14
A
B
C
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
A15
0
1
E1
E2
E3
A0 -A11
RAM2
A0 -A11
RAM3
WR
IO/M
RD