ΣΗΜΜΥ ΕΜΠ
Συστήματα Μικροϋπολογιστών / 2η ομάδα ασκήσεων
Νικόλαος Πηγαδάς / el18445
2020-2021 / 6ο εξάμηνο

Άσκηση 1

Κώδικας

```
IN 10H
;1A)
LXI H,0900H    ; Αρχικοποιώ τον καταχωρητή HL με την πρώτη διεύθυνση, 0900 H
MVI A,00H      ; Αρχικοποίηση μετρητή που δημιουργεί της ζητούμενες τιμές προς αποθήκευση

ITERATION:
       MOV M,A      ; Σώζω την τιμή στην θέση μνήμης που της αντιστοιχεί
       CPI FFH      ; Ελέγχω αν η τιμή αυτή είναι 255 και άρα η τελευταία προς αποθήκευση
       JZ CONTINUE  ; Αν είναι πάμε στο ζητούμενο (β) της άσκησης
                    ; αλλιώς:
       INR A        ; Αύξηση τιμής μετρητή κατά 1/Δημιουργία επόμενης τιμής προς   >>
       INX H        ; Πηγαίνω την αντίστοιχη επιθυμητή διεύθυνση μνήμης
       JMP ITERATION  ; Επαναλαμβάνω την διαδικασία


;1B)
CONTINUE:
       LXI H,09FFH      ; Επιλέγω την προσπέλαση στοιχείων από το τέλος για ευκολία
       LXI B,0000H      ; Αρχικοποιώ τον μετρητή μου των άσσων

NEXT_NUMBER:
         MOV A,M        ; Φέρνω αριθμό από την μνήμη
         MVI D,09H      ; Δίνω στον μετρητή των bit αριθμού, τον D, την τιμή 9 (*)

NEXT_BIT:
         DCR D        ; Αφαιρώ 1 από τον αριθμό των bit που δεν έχουν ελεγχθεί
         JZ NEXT_ADDR ; Αν μένουν 0 bits που δεν έχουν ελεγχθεί, πηγαίνω σε νέα διεύθυνση
         RRC          ; Δεξιά ολίσθηση/Μεταφορά LSB στην θέση κρατούμενου
         JNC NEXT_BIT ; Αν CY = 0, τότε έλεγξε επόμενο bit
         INX B        ; Αλλιώς, αυξάνω τον μετρητή άσσων μου
         JMP NEXT_BIT ; και κατόπιν ελέγχω το επόμενο bit

NEXT_ADDR:
         DCR L  ; Εφόσον ξεκινάω από το τέλος, αφαιρώ 1 για την προσπέλαση των αριθμών
         JNZ NEXT_NUMBER ; Έλεγχος επόμενου αριθμού

LXI H,0A08H ; Ορισμός αυθαίρετης θέσης στην μνήμη για την αποθήκευση των 4 MSB του result
MOV M,B     ; και πραγματοποίηση αποθήκευσης
LXI H,0A09H ; Ορισμός αυθαίρετης θέσης στην μνήμη για την αποθήκευση των 4 LSB του result
MOV M,C     ; και πραγματοποίηση αποθήκευσης


;1C)
MVI D,00H     ; Αρχικοποίηση μετρητή πλήθους αριθμών
LXI H,09FFH   ; Επιλέγω την προσπέλαση στοιχείων από το τέλος για ευκολία

CHECK_10:        ; Έλεγχος αν ο αριθμός είναι μεγαλύτερος του 10
        MOV A,M  ; Τον φέρνω από την μνήμη αρχικά
        CPI 10H  ; Σύγκριση με το 10
        JNC CHECK_60   ; Αν είναι μεγαλύτερος, σύγκρινε σε σχέση με το 60
        JMP NEXT_NUM   ; Αλλιώς έλεγξε τον επόμενο αριθμό

CHECK_60:        ; Έλεγχος αν ο αριθμός είναι μικρότερος του 60
        CPI 61H  ; Σύγκριση με το 61 H
        JC COUNTER   ; Αύξησε τον μετρητή αριθμών, αν xn < 61
        JMP NEXT_NUM ; Αλλιώς πήγαινε στον επόμενο αριθμό
```

```
COUNTER:          ; Βοηθητική ετικέτα
        INR D     ; που αυξάνει τον μετρητή

NEXT_NUM:
          DCR L  ; Εφόσον ξεκινάω από το τέλος, αφαιρώ 1 για την προσπέλαση των αριθμών
          JNZ CHECK_10 ; Ξεκινάω έλεγχο για τον νέο αριθμό

LXI H,0A0CH  ; Ορισμός αυθαίρετης θέσης στην μνήμη για την αποθήκευση του αποτελέσματος
MOV M,D      ; και πραγματοποίηση αποθήκευσης
END          ; Τέλος προγράμματος
```

(*)1B) Αφαιρώ απευθείας από το 9 του μετρητή bit αριθμού(D), οπότε τελικά ελέγχονται τα 8 bits κάθε αριθμού.

08FA	00	08FB	00	08FC	00	08FD	00	08FE	00	08FF	00	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	80	0981	81	0982	82	0983	83	0984	84	0985	85
0986	86	0987	87	0988	88	0989	89	098A	8A	098B	8B	098C	8C	098D	8D	098E	8E	098F	8F
0990	90	0991	91	0992	92	0993	93	0994	94	0995	95	0996	96	0997	97	0998	98	0999	99
099A	9A	099B	9B	099C	9C	099D	9D	099E	9E	099F	9F	09A0	A0	09A1	A1	09A2	A2	09A3	A3
09A4	A4	09A5	A5	09A6	A6	09A7	A7	09A8	A8	09A9	A9	09AA	AA	09AB	AB	09AC	AC	09AD	AD
09AE	AE	09AF	AF	09B0	B0	09B1	B1	09B2	B2	09B3	B3	09B4	B4	09B5	B5	09B6	B6	09B7	B7
09B8	B8	09B9	B9	09BA	BA	09BB	BB	09BC	BC	09BD	BD	09BE	BE	09BF	BF	09C0	C0	09C1	C1
09C2	C2	09C3	C3	09C4	C4	09C5	C5	09C6	C6	09C7	C7	09C8	C8	09C9	C9	09CA	CA	09CB	CB
09CC	CC	09CD	CD	09CE	CE	09CF	CF	09D0	D0	09D1	D1	09D2	D2	09D3	D3	09D4	D4	09D5	D5
09D6	D6	09D7	D7	09D8	D8	09D9	D9	09DA	DA	09DB	DB	09DC	DC	09DD	DD	09DE	DE	09DF	DF
09E0	E0	09E1	E1	09E2	E2	09E3	E3	09E4	E4	09E5	E5	09E6	E6	09E7	E7	09E8	E8	09E9	E9
09EA	EA	09EB	EB	09EC	EC	09ED	ED	09EE	EE	09EF	EF	09F0	F0	09F1	F1	09F2	F2	09F3	F3
09F4	F4	09F5	F5	09F6	F6	09F7	F7	09F8	F8	09F9	F9	09FA	FA	09FB	FB	09FC	FC	09FD	FD
09FE	FE	09FF	FF	0A00	00	0A01	00	0A02	00	0A03	00	0A04	00	0A05	00	0A06	00	0A07	00
0A08	04	0A09	00	0A0A	00	0A0B	00	0A0C	51	0A0D	00	0A0E	00	0A0F	00	0A10	00	0A11	00

Καταγραφή οθόνης του στιγμιότυπου μνήμης που δείχνει ότι:
(α) είναι αποθηκευμένες οι τιμές 0-255 στις διευθύνσεις 0900 H – 09FF H.

(β) είναι αποθηκευμένο στις θέσεις 0A08(B) και 0A09(C) το σύνολο των 1 των παραπάνω δεδομένων, όταν αυτά είναι σε δυαδική μορφή. Όπως αναμέναμε, το αποτέλεσμα είναι 0400 H, δηλαδή 1024_{10}. Το αποτέλεσμα αυτό προκύπτει θεωρητικά ως εξής:
Έχουμε τους αριθμούς 0000 0000 μέχρι 1111 1111. Παρατηρούμε ότι υπάρχει συμμετρία στην εμφάνιση μηδενικών και άσσων, δηλαδή ο αριθμός δυαδικών της μορφής x1xx xxxx είναι ίσος με τον αριθμό δυαδικών της μορφής x0xx xxxx. Επαγωγικά καταλήγουμε στο συμπέρασμα ότι από όλα τα ψηφία περιμένουμε τα μισά να είναι 1 και τα μισά 0. Άρα ο αριθμός άσσων είναι:

$$½ (8 \cdot 256) = 1024$$

όπου 8 ο αριθμός bits ενός αριθμού και 256 το σύνολο των αριθμών.

(γ) είναι αποθηκευμένο το πλήθος των αριθμών με (10 H) ≤ x_n ≤ (60 H), στο δεκαδικό 16 ≤ x_n ≤ 96, δηλαδή 51H (81_{10}) στην θέση 0A0C της μνήμης.

Άσκηση 2

```
LXI B,0140H          ; (B) <- 320 (ορισμός καθυστέρησης από trial and error)

OFF1:                ; OFF για πρώτη φορά
       LDA 2000H     ; διάβασμα εισόδου
       ANI 80H       ; Απομόνωση MSB
       CPI 80H       ; έλεγχος τιμής MSB
       JZ ON1        ; Αν MSB = 1, το MSB είναι ON για πρώτη φορά

       JMP OFF1      ; Αλλιώς MSB = 0 και παραμένω σε αυτήν την ετικέτα

ON1:                 ; ON για πρώτη φορά
       LDA 2000H     ; Διάβασμα εισόδου
       ANI 80H       ; Απομόνωση MSB
       CPI 00H       ; Αν MSB = 0, πάμε στην RESTART για να καταλήξουμε στην OFF2, αφού
       JZ RESTART    ; έχουμε τον συνδυασμό OFF-ON-OFF
       JMP ON1       ; Αλλιώς MSB = 1 και παραμένω σε αυτήν την ετικέτα

RESTART:
       LXI D,FFFFH   ; (D) <- 65535 (από trial and error για τα 20 sec. Ορίζει πόσες φορές
                                    θα κληθούν οι CALL DELB)
       MVI L,00H     ; FLAG

OFF2:                ; OFF για δεύτερη φορά
       LDA 2000H
       ANI 80H
       CPI 80H
       JZ ON2              ; Αν MSB = 1, έχουμε ON για δεύτερη φορά
       MVI A,00H           ; Άναμμα LEDs (αντίστροφη λογική)
       STA 3000H           ;   >>    >>
       DCR D               ; Μέσω αυτής της εντολής ελέγχουμε το πέρας της καθυστέρησης
       JNZ STOP            ; Καθυστέρηση
       MVI A,FFH           ; Σβήσιμο των LEDs
       STA 3000H           ;    >>    >>  >>
       MOV M,L             ; Πρόσβαση στο L ώστε να συνδεθεί με την παρακάτω εντολή
       JZ OFF1             ; L = 0, MSB off με LEDs σβηστά, πάμε στην ετικέτα OFF1
       JMP ON1             ; L = 1, MSB on με LEDs σβηστά, πάμε στη ετικέτα ON1
STOP:                      ; Καθυστέρηση/ ώρα που τα LEDs παραμένουν ανοιχτά
       CALL DELB
       CALL DELB
       JMP OFF2

ON2:
       INR L           ; (L) <- 1
       LDA 2000H
       ANI 80H
       CPI 00H
       JZ RESTART       ; OFF-ON-OFF με LEDs αναμμένα, ανανεώνω την καθυστέρηση
       DCR D            ; αποκτώ πρόσβαση στον καταχωρητή D
       INR D            ; αλλά δεν αλλοιώνω το περιεχόμενό του
       JNZ OFF2         ; Αν OFF-ON-OFF-ON και D != 0, συνεχίζω την καθυστέρηση στην OFF2
       JMP ON1          ; Αν OFF-ON-OFF-ON και D = 0, έχει τελειώσει η καθυστέρηση/ ο
                        ; χρόνος που τα LEDs είναι αναμμένα, πάμε στην ετικέτα ON1
END
```

Επεξήγηση

Η <u>γενική ιδέα</u> της λύσης είναι ότι μέσω της εντολής ANI απομονώνουμε το MSB κάθε φορά και πάμε σε ετικέτα ON ή OFF. Έχουμε ετικέτα για κάθε περίπτωση:

- την OFF1 θεωρώντας ότι είναι OFF για πρώτη φορά, οπότε τα φώτα είναι σβηστά
- την ON1 θεωρώντας ότι ο διακόπτης MSB ανέβηκε για πρώτη φορά, φώτα ακόμα σβηστά
- την OFF2, που εφόσον έχει προηγηθεί η ON1, ανοίγουν τα φώτα για 20 sec(έχουμε OFF-ON-OFF)
- την ON2, χάρη στην οποία μπορούμε να ανανεώσουμε τα 20 sec, αν ξαναπατηθεί ο συνδυασμός OFF-ON-OFF, ενώ δεν έχουν σβήσει τα φώτα (έχει ξαναπατηθεί ο συνδυασμός ON-OFF-ON)

Καταχωρητής D: ορίζει πόσες φορές θα πάμε στην ετικέτα STOP, η οποία ορίζει την χρονική περίοδο που τα λαμπάκια θα είναι αναμμένα, μέσω της CALL DELB. Είναι πρακτικά σαν την μεταβλητή (i) που ορίζει πόσες φορές θα γίνει ένα for loop.

Καταχωρητής L: Εδώ ο καταχωρητής L λειτουργεί ως flag. Συγκεκριμένα, όταν τελειώνει η OFF2 και σβήσουν τα LEDs, έχει διαφορά από πού ήρθαμε. Αν παραμείναμε στην OFF2 καθόλη την διάρκεια της καθυστέρησης, δηλαδή αν δεν είχαμε συνδυασμό OFF-ON-OFF όσο τα φώτα ήταν αναμμένα, τότε το MSB είναι OFF με τα φώτα σβηστά. Έτσι, πάμε στην ετικέτα OFF1.

Από την άλλη, αν ενδιάμεσα πήγαμε στην ετικέτα ON2 επειδή το MSB ξαναπήγε στην θέση ON, αλλά δεν έχουμε ξανά OFF για ανανέωση του χρόνου, θέλουμε να τελειώσουν τα 20 sec καθυστέρησης που είχαν ξεκινήσει. Άρα, επιστρέφουμε στην OFF2 για να τα ολοκληρώσει. Όμως, πρέπει να ξέρουμε όταν τελειώσουν τα LEDs αν ήρθαμε από την ON2 και συνεπώς να πάμε στην ON1, εφόσον το MSB είναι ON με σβηστά τα LEDS.

Συνοψίζοντας,
λοιπόν, έχω ορίσει ότι για L = 1 έχουμε έρθει από την ON2 (συνδυασμός OFF-ON-OFF-ON, ενώ τα LEDs είναι αναμμένα. Έχουμε MSB ON, σβηστά τα LEDs, πάμε στην ON1) και για L = 0 έχουμε παραμείνει στην OFF2 και για τα 20 sec (Δεν είχαμε συνδυασμό OFF-ON-OFF, όσο τα LEDs ήταν αναμμένα. Έχουμε MSB OFF, σβηστά τα LEDs, πάμε στην OFF1).

Η καθυστέρηση γίνεται μέσω του διπλού καταχωρητή BC. Θέλουμε να ανανεώνεται συνεχώς ο χρόνος για νέα πατήματα του συνδυασμού OFF-ON-OFF και η διακριτική ικανότητα του συστήματος είναι περίπου 1/10 sec. Συνεπώς, επιδιώκουμε η καθυστέρηση να μην γίνεται συνεχόμενα, αλλά μέσω βρόχου που κάνει συνεχείς καθυστερήσεις πολύ πολύ μικρές. Αυτό μας επιτρέπει να ελέγχουμε συνεχώς αν έχει αλλάξει η τιμή του MSB.
- Αν όχι, συνεχίζουμε την καθυστέρηση των ~20 sec, κανονικά στην ετικέτα OFF2.
- Αν ναι, πάμε στην ετικέτα ON2.

Η ετικέτα OFF2:
- Ελέγχει την είσοδο μήπως χρειαστεί να ανανεωθούν τα 20 sec, αν ναι μας στέλνει στην ON2
- Ανάβει τα LEDs και ελέγχει την καθυστέρηση μέσω του καταχωρητή D και της ετικέτας STOP
- Σβήνει τα LEDs
- Στο τέλος της ετικέτας έχουμε σβηστά LEDs. Αν MSB ON πάμε στην OFF1, αλλιώς στην ON1

Η ετικέτα ON2:
- Θέτει το flag L ίσο με 1
- 1) Ελέγχει την είσοδο, η οποία αν είναι MSB OFF ξανά, δηλαδή ενώ τα LEDs είναι αναμμένα έχουμε πάλι OFF-ON-OFF, τότε ανανεώνει τον χρόνο των 20 sec. Αυτό το επιτυγχάνει οδηγώντας το πρόγραμμα στην ετικέτα RESTART, η οποία αρχικοποιεί ξανά το L (= 0) και το D.
  2) Αν έχουμε OFF-ON-OFF-ON, μας στέλνει πίσω στην OFF2, ώστε να μην σβήσουν πρόωρα τα LEDs.
  3) Αλλιώς, έχουμε σβηστά LEDs και MSB ON, οπότε μας στέλνει στην ετικέτα ON1.

Η υλοποίηση της καθυστέρησης των **~20 sec**, **επιτεύχθηκε μέσω δοκιμών**, καθώς υπήρξαν παρεκκλίσεις μεταξύ θεωρίας και πράξης. Συγκεκριμένα, θεωρητικά θέλουμε:
- 20 sec καθυστέρηση
- Β κοντά στην διακριτική ικανότητα του συστήματος (100 ms) και μέσω loop φτάνουμε στην επιθυμητή καθυστέρηση. Έτσι, μπορούμε συνεχώς να ελέγχουμε της αλλαγές στην είσοδο. Άρα, ιδανικά υπολογίζουμε ότι:
  Β ≈ 100, $\frac{20\,000\text{ms}}{1\,ms\cdot B}$ = 200 για τον καταχωρητή D, δηλαδή κλήσεις της ετικέτας STOP, η οποία σε αυτήν την περίπτωση περιέχει μια μόνο κλήση της CALL DELB, η οποία προδίδει καθυστέρηση 1ms.

Τέλος, αξίζει να σημειωθεί ότι το trial and error έγινε σε Η/Υ μέσων δυνατοτήτων, ενώ η ρύθμιση ταχύτητας στο μLAB ήταν στο μέγιστο. Με τις παραπάνω θεωρητικές τιμές, τα LEDs έσβηναν πρόωρα. Η καθυστέρηση έχει να κάνει με τις δυνατότητες του εκάστοτε συστήματος.

Άσκηση 3

(α)

```
MVI E,09H          ;Χρησιμοποιείται σαν όριο στον έλεγχο των bit
START:             ;Αν η είσοδος γίνει 0 σβήσε ό,τι έχεις γράψει
       MVI A,00H
       CMA
       STA 3000H
INPUT:             ;Διάβασμα εισόδου και έλεγχός της. Αρχικοποίηση του μετρητή bit
       LDA 2000H
       CPI 00H
       JZ START
       MVI C,00H   ;Μετρητής θέσης δεξιότερου bit
       JMP LOOP1

LOOP1:
       RRC         ;Δεξιά ολίσθηση μέχρι να βρω άσσο
       JC HELP     ;Όταν βρω τον δεξιότερο, φεύγω
       INR C       ;Βρίσκει την θέση του δεξιότερου bit
       CMP E
       JNZ LOOP1
       JMP INPUT

HELP:
       MVI A,01H
; κάνουμε στο 01H τόσες ολισθήσεις, όσες η θέση στην οποία βρέθηκε ο δεξιότερος άσσος
;(μετρητής C)

LOOP2:
       RLC    ;αριστερή ολίσθηση για να ανάψει το κατάλληλο LED
       DCR C
       JZ PRINT
       JMP LOOP2

PRINT:            ;Εκτύπωση
       CMA
       STA 3000H
       JMP INPUT

END
```

(β)

```
IN 10H
MVI E,00H
MVI A,00H
MVI C,00H

; Προσδιορισμός εισόδου και ανακατεύθυνση στην αντίστοιχη ετικέτα
IN:
       CALL KIND
       CMP E
       JZ IN
       MOV E,A
       CPI 09H
       JNC IN      ; >8
       CPI 00H
       JZ IN       ; ==0

; Επιθυμητές περιπτώσεις
       CPI 01H
       JZ ONE
       CPI 02H
       JZ TWO
       CPI 03H
       JZ THREE
       CPI 04H
       JZ FOUR
       CPI 05H
       JZ FIVE
       CPI 06H
       JZ SIX
       CPI 07H
       JZ SEVEN
       MVI A,00H
       CMA
       JMP END

ONE: MVI A,01H
     JMP END

TWO: MVI A,03H
     JMP END

THREE: MVI A,07H
       JMP END

FOUR: MVI A,0FH
      JMP END

FIVE: MVI A,1FH
      JMP END

SIX: MVI A,3FH
     JMP END

SEVEN: MVI A,7FH
       JMP END

END:   RRC
       ADI 80H

; Output
SHOW:  STA 3000H
       JMP IN

END
```

(γ)

```
START:
       IN 10H
       LXI H,0A00H
       MVI B,04H     ; Για την παρακάτω επανάληψη

L1:                  ; αποθήκευση κενού x4
       MVI M,10H
       INX H
       DCR B
       JNZ L1

; Γραμμή 0
       MVI A,FEH     ; πόρτα σάρωσης = 11111110
       STA 2800H
       LDA 1800H     ; διάβασμα στηλών των πλήκτρων
       ANI 07H       ; κρατάμε μόνο τα 3 LSB, αυτά μας ενδιαφέρουν για την στήλη

       MVI C,86H     ; πιθανό κουμπί
       CPI 06H       ; έλεγχος
       JZ SHOW       ; αν ναι τύπωσέ το στο 7-segment display

       MVI C,85H     ; ομοίως για όλα τα πιθανά κουμπιά
       CPI 05H
       JZ SHOW

; ομοίως και για τις υπόλοιπες γραμμές
; Γραμμή 1
       MVI A,FDH
       STA 2800H
       LDA 1800H
       ANI 07H

       MVI C,84H
       CPI 06H       ; RUN
       JZ SHOW
       MVI C,80H
       CPI 05H       ; FETCH_REG
       JZ SHOW
       MVI C,82H
       CPI 03H       ; FETCH_ADDRS
       JZ SHOW

; Γραμμή 2
       MVI A,FBH
       STA 2800H
       LDA 1800H
       ANI 07H

       MVI C,00H
       CPI 06H       ; 0
       JZ SHOW
       MVI C,83H
       CPI 05H       ; STORE/INCR
       JZ SHOW
       MVI C,81H
       CPI 03H       ; DECR
       JZ SHOW

; Γραμμή 3
       MVI A,F7H
       STA 2800H
       LDA 1800H
       ANI 07H

       MVI C,01H     ; 1
       CPI 06H
       JZ SHOW
       MVI C,02H     ; 2
```

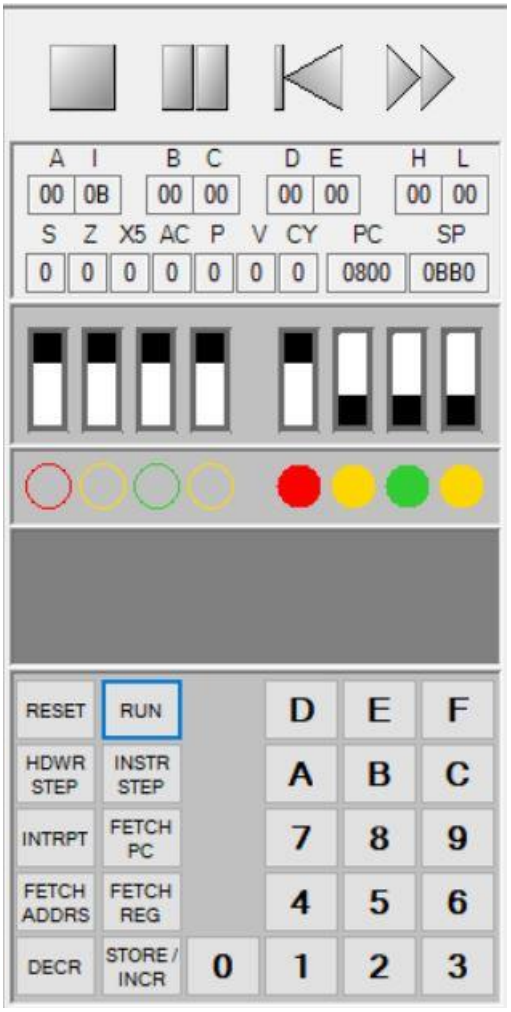
```
        CPI 05H
        JZ SHOW
        MVI C,03H     ; 3
        CPI 03H
        JZ SHOW

; Γραμμή 4
        MVI A,EFH
        STA 2800H
        LDA 1800H
        ANI 07H

        MVI C,04H
        CPI 06H       ; 4
        JZ SHOW
        MVI C,05H
        CPI 05H       ; 5
        JZ SHOW
        MVI C,06H
        CPI 03H       ; 6
        JZ SHOW

; Γραμμή 5
        MVI A,DFH
        STA 2800H
        LDA 1800H
        ANI 07H

        MVI C,07H
        CPI 06H       ; 7
        JZ SHOW
        MVI C,08H
        CPI 05H       ; 8
        JZ SHOW
        MVI C,09H
        CPI 03H       ; 9
        JZ SHOW
; Γραμμή 6
        MVI A,BFH
        STA 2800H
        LDA 1800H
        ANI 07H

        MVI C,0AH
        CPI 06H       ; A
        JZ SHOW
        MVI C,0BH
        CPI 05H       ; B
        JZ SHOW
        MVI C,0CH
        CPI 03H       ; C
        JZ SHOW

; Γραμμή 7
        MVI A,7FH
        STA 2800H
        LDA 1800H
        ANI 07H

        MVI C,0DH
        CPI 06H       ; D
        JZ SHOW
        MVI C,0EH
        CPI 05H       ; E
        JZ SHOW
        MVI C,0FH
        CPI 03H       ; F
        JZ SHOW

        JMP START     ; αν δεν πατήθηκε κουμπί, κάνουμε τους ελέγχους από την αρχή
```

```
; Εμφάνιση στο 7-segment display
SHOW:
       LXI H,0A04H
       MOV A,C             ; (A) <- πληροφορία κουμπιού
       ANI 0FH             ; απομόνωση 4 LSBs
       MOV M,A             ; μετακίνηση στο πέμπτο ψηφίο των 7-segment display

       INX H               ; επόμενη θέση μνήμης
       MOV A,C
       ANI F0H             ; κρατάμε τα 4 MSBs
       RLC                 ; Μετατροπή σε 4 LSBs
       RLC
       RLC
       RLC

       MOV M,A             ; μετακίνηση στο έκτο ψηφίο του 7-segment display
       LXI D,0A00H         ; μετακίνηση του block 0A00H - 0A05H
                           ; στο σημείο που διαβάζει η ρουτίνα DCD
       CALL STDM
       CALL DCD            ; ρουτίνα που εμφανίζει το αποτέλεσμα στο 7-segment display

       JMP START           ; Ξανά από την αρχή

                                          END
```

Άσκηση 4

```
IC:

       LDA 2000H     ; (A) <- A3B3 A2B2 A1B1 A0B0 (INPUT)
       MOV B,A       ; (B) <- A3B3 A2B2 A1B1 A0B0
       RRC           ; (A) <- B0A3 B3A2 B2A1 B1A0
       MOV C,A       ; (C) <- B0A3 B3A2 B2A1 B1A0

       XRA B         ; (A) <- XY3 XY2 XY1 XY0 , where Yi = XOR(Ai,Bi), i=0,1,2,3
       RRC           ; (A) <- Y0X Y3X Y2X Y1X
       MOV D,A       ; (D) <- Y0X Y3X Y2X Y1X
       ANI 02H       ; (A) <- 00  00  00  Y10
       MOV E,A       ; (E) <- 00  00  00  X10 , Y1 == X1, E contains A1 XOR B1 in correct
position
       MOV A,D       ; (A) <- Y0X Y3X Y2X Y1X
       RLC           ; (A) <- XY3 XY2 XY1 XY0
       ANI 01H       ; (A) <- 00  00  00  0Y0
       MOV D,A       ; (D) <- 00  00  00  0Y0

       MOV A,C       ; (A) <- B0A3 B3A2 B2A1 B1A0
       ANA B         ; (A) <- B0A3 B3A2 B2A1 B1A0 AND A3B3 A2B2 A1B1 A0B0 (B AND A==C)
       MOV C,A       ; (C) <- XZ3 XZ2 XZ1 XZ0, where Zi = AND(Ai,Bi), i=0,1,2,3
       RRC           ; (A) <- Z0X Z3X Z2X Z1X
       RRC           ; (A) <- XZ0 XZ3 XZ2 XZ1
       ANI 04H       ; (A) <- 00  00  0Z2  00
       MOV B,A       ; (B) <- 00  00  0Z2  00 , B contains A2 AND B2 in correct position

       MOV A,C       ; (A) <- XZ3 XZ2 XZ1 XZ0
       RRC           ; (A) <- Z0X Z3X Z2X Z1X
       RRC           ; (A) <- XZ0 XZ3 XZ2 XZ1
       RRC          ; (A) <- Z1X Z0X Z3X Z2X
       ANI 08H       ; (A) <- 00  00  Z30  00
       MOV C,A       ; (C) <- 00  00  X30  00 , Z3 == X3, C contains A3 AND B3 in correct
position

       MOV A,C       ; (A) <- 00  00  Z30  00
       RRC           ; (A) <- 00  00  0Z3  00
       ORA B         ; (A) <- 00  00  0Z3  00 OR 00  00  0Z2  00  (B OR A) == ((A2 AND
B2) OR (A3 AND B3))
       MOV B,A       ; (B) <- 00  00  0X2  00 , X2, B contains ((A2 AND B2) OR (A3 AND
B3))

       MOV A,E       ; (A) <- 00  00  00  X10
       RRC           ; (A) <- 00  00  00  0X1
       XRA D         ; (A) <- 00  00  00  0X1 XOR 00  00  00  0Y0 (D XOR E) == ((A0 XOR
B0) XOR (A1 XOR B1))
       MOV D,A       ; (D) <- 00  00  00  0X0 , X0, D contains ((A0 XOR B0) XOR (A1 XOR
B1))

       MOV A,C       ; 0 0 0 0 X3  0  0  0
       ADD B         ; 0 0 0 0 X3 X2  0  0
       ADD E         ; 0 0 0 0 X3 X2 X1  0
       ADD D         ; 0 0 0 0 X3 X2 X1 X0

       CMA           ; (A) <- (A') (Reverse logic)
       STA 3000H     ; OUTPUT
       JMP IC        ; NEW INPUT
END
```

Παρατίθεται και ένα παράδειγμα εκτέλεσης του προγράμματος:



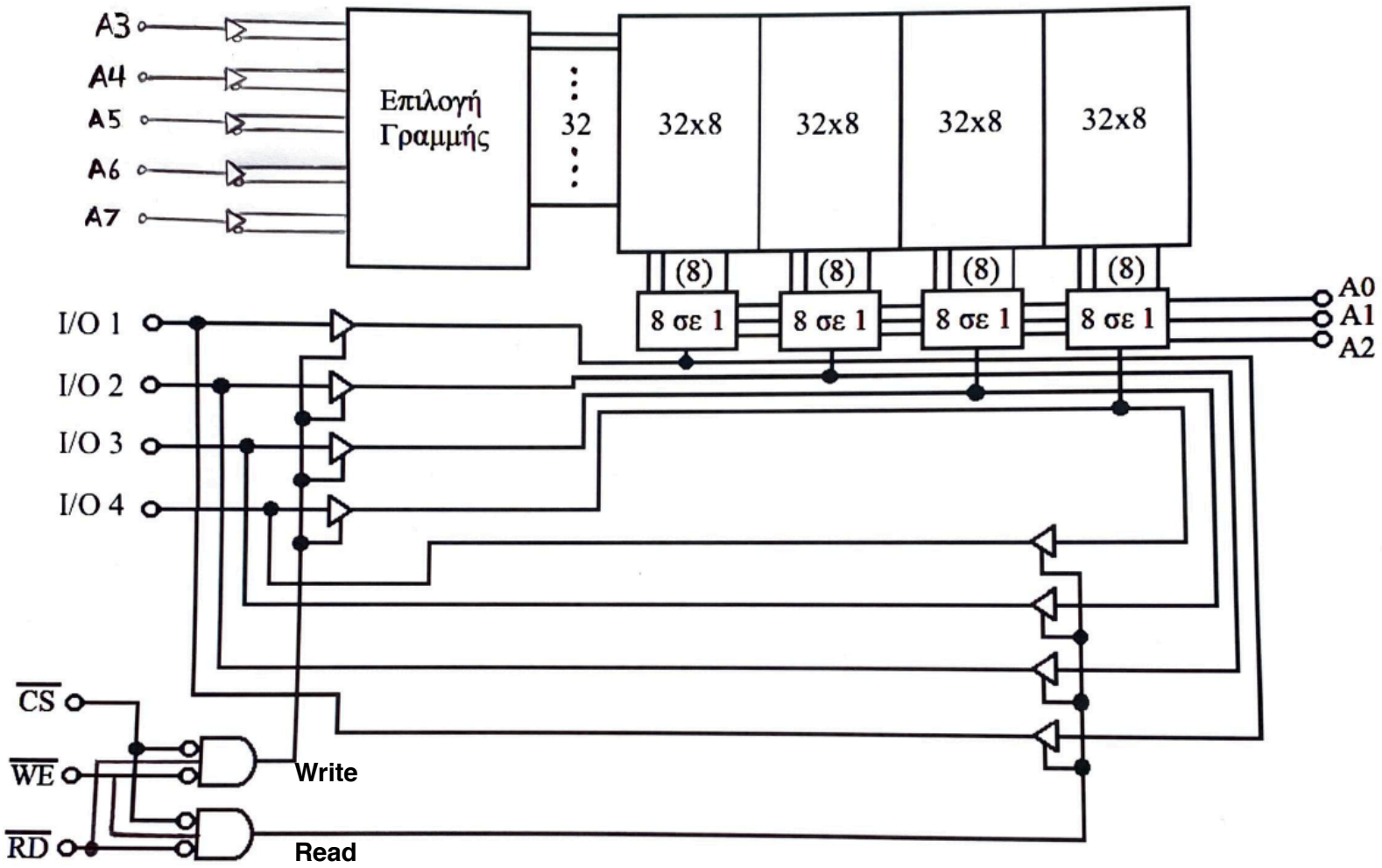
Έχουμε:
A3 = 1, B3 = 1, A2 = 1, B2 = 1, A1 = 1, B1 =0, A0 = 0, B0 = 0
και
X0 = ((A0 XOR B0) XOR (A1 XOR B1)) = ((0 XOR 0) XOR (1 XOR 0)) = 1
X1 = (A1 XOR B1) = (1 XOR 0) = 1
X2 = ((A2 AND B2) AND (A3 AND B3)) = ((1 AND 1) AND (1 AND 1)) = 1
X3 = (A3 AND B3) = (1 AND 1) = 1

Οπότε επαληθεύεται η λειτουργία του προγράμματος.

Άσκηση 5

Εσωτερική οργάνωση μνήμης SRAM 256x4 bits

Σχεδίαση



Γενικά

Η ζητούμενη οργάνωση είναι για SRAM 256x4 bits, δηλαδή έχουμε χώρο 256 λέξεων μεγέθους 4 bit. Χωρίζουμε, λοιπόν, την μνήμη σε 4 τμήματα, που το καθένα έχει μέγεθος 256 bits. Για λόγους απόδοσης θέλουμε η διάταξη της μνήμης να είναι τετραγωνική, οπότε επιλέγω τον χωρισμό 32x8 για κάθε τμήμα. Άρα, προκύπτει η τετραγωνική διάταξη με 32 γραμμές και 8x4=32 στήλες.

Έχουμε:

$\log_2 32 = 5$ bits για τον προσδιορισμό της γραμμής μιας διεύθυνσης μνήμης, τα A_{3-7}

$\log_2 8 = 3$ bits για τον προσδιορισμό της στήλης μιας διεύθυνσης μνήμης, τα A_{0-2}

Οι ακροδέκτες I/O_{1-4} χρησιμεύουν ως είσοδοι ή έξοδοι, ανάλογα αν έχουμε εγγραφή ή ανάγνωση, αντίστοιχα. Ο CS' καθορίζει αν έχουμε λειτουργία της συγκεκριμένης SRAM, ενώ οι RD' και WE' αν έχουμε ανάγνωση ή εγγραφή, αντίστοιχα. Αξίζει να αναφερθεί εδώ ότι ακολουθείται η αντίστροφη λογική για τα CS', RD' και WE', αφού ενεργοποιούνται με την τιμή 0. Αυτό επιλέγεται, επειδή μέσω της αντίστροφης λογικής έχουμε μικρότερο θόρυβο και λιγότερη κατανάλωση. Καταληκτικά, για να ενεργοποιηθεί η μνήμη για εγγραφή πρέπει να έχουμε:

CS' = 0, WE' = 0 και RD' = 1

ενώ για ανάγνωση:

CS' = 0, RD' = 0 και WE' = 1

ενώ στις περιπτώσεις

CS' = 0, RD' = 0 και WE' = 0

και

CS' = 0, RD' = 1 και WE' = 1

και

CS' = 1

η μνήμη δεν λειτουργεί. Η πρώτη περίπτωση μάλιστα, ελέγχεται από τους μη ανεστραμμένους ακροδέκτες των RD' και WE' που συνδέονται με τις πύλες AND.

Στην περίπτωση εγγραφής οι ακροδέκτες I/O λειτουργούν σαν input και μέσω των bit διεύθυνσης και των πολυπλεκτών, βρίσκεται η επιθυμητή θέση μνήμης και γίνεται η ζητούμενη εγγραφή. Η είσοδος διέρχεται στην αρχή από απομονωτές ελέγχου ροής δεδομένων.

Στην περίπτωση ανάγνωσης τα bits διεύθυνσης με την βοήθεια πολυπλεκτών μας δίνουν πρόσβαση στην επιθυμητή θέση μνήμης. Κατόπιν, με την βοήθεια απομονωτών ελέγχου ροής δεδομένων οδηγούνται στους ακροδέκτες I/O, οι οποίοι λειτουργούν σαν output.

Εξήγηση μέσω παραδείγματος

Έστω ότι επιθυμούμε να:

1) Διαβάσουμε την θέση μνήμης 10010001
2) Γράψουμε την λέξη 1010 στην θέση μνήμης 01101110

1) Τα bits διεύθυνσης A_{0-7} λαμβάνουν τις τιμές 10010001 ως εξής:

$A_7 = 1, A_6 = 0, A_5 = 0, A_4 = 1, A_3 = 0$ (σειρά)

$A_2 = 0, A_1 = 0, A_0 = 1$ (στήλη)

Στέλνονται αρνητικοί παλμοί στους ακροδέκτες CS' και RD', ενώ ο WE' λαμβάνει θετικό. Η πύλη AND για την ανάγνωση (κάτω πύλη) έχει έξοδο 1 και δίνει σήμα στους αντίστοιχους απομονωτές ελέγχου ροής δεδομένων να επιτρέψουν την διεύλεση δεδομένων. Παράλληλα, η πύλη AND για την εγγραφή (πάνω πύλη) έχει έξοδο 0 και οι αντίστοιχοι απομονωτές δεν επιτρέπουν την διεύλεση δεδομένων.

Από τα A_{3-7} επιλέγεται μέσω πολυπλεκτών η σειρά της θέσης μνήμης, έχουμε 10010, δηλαδή ψάχνουμε στην 18η σειρά. Έχοντας τα A_{0-2} επιλέγουμε μέσω πολυπλεκτών την στήλη, εδώ 001 και άρα την 1η στήλη. Η ζητούμενη θέση έχει πλέον προσδιοριστεί και τα δεδομένα, διερχόμενα μέσα από τους απομονωτές ροής δεδομένων, φτάνουν στις I/O εξόδους και διαβάζονται.

2) Στα bits διεύθυνσης A_{0-7} έχουμε την θέση μνήμης προς εγγραφή. Τα δεδομένα προς εγγραφή μπαίνουν από τις I/O σαν input. Έχουμε αρνητικό παλμό στους ακροδέκτες CS', WE', ενώ θετικό στον RD'. Η πύλη AND για την εγγραφή έχει output 1, ενώ η πύλη AND για την ανάγνωση έχει output 0. Οι απομονωτές ελέγχου ροής δεδομένων της εγγραφής επιτρέπουν την διεύλεση πληροφορίας, ενώ αυτοί της ανάγνωσης όχι.

Μέσω πολυπλεκτών εντοπίζουμε την θέση προς εγγραφή. Συγκεκριμένα:

$A_7 = 0, A_6 = 1, A_5 = 1, A_4 = 0, A_3 = 1$ (γραμμή)

$A_2 = 1, A_1 = 1, A_0 = 0$ (στήλη)

Δηλαδή θέλουμε να γράψουμε στην θέση που βρίσκεται στην γραμμή 13 (01101) και στην στήλη 6 (110). Η θέση είναι προσδιορισμένη και ο αριθμός 1010 εγγράφεται.

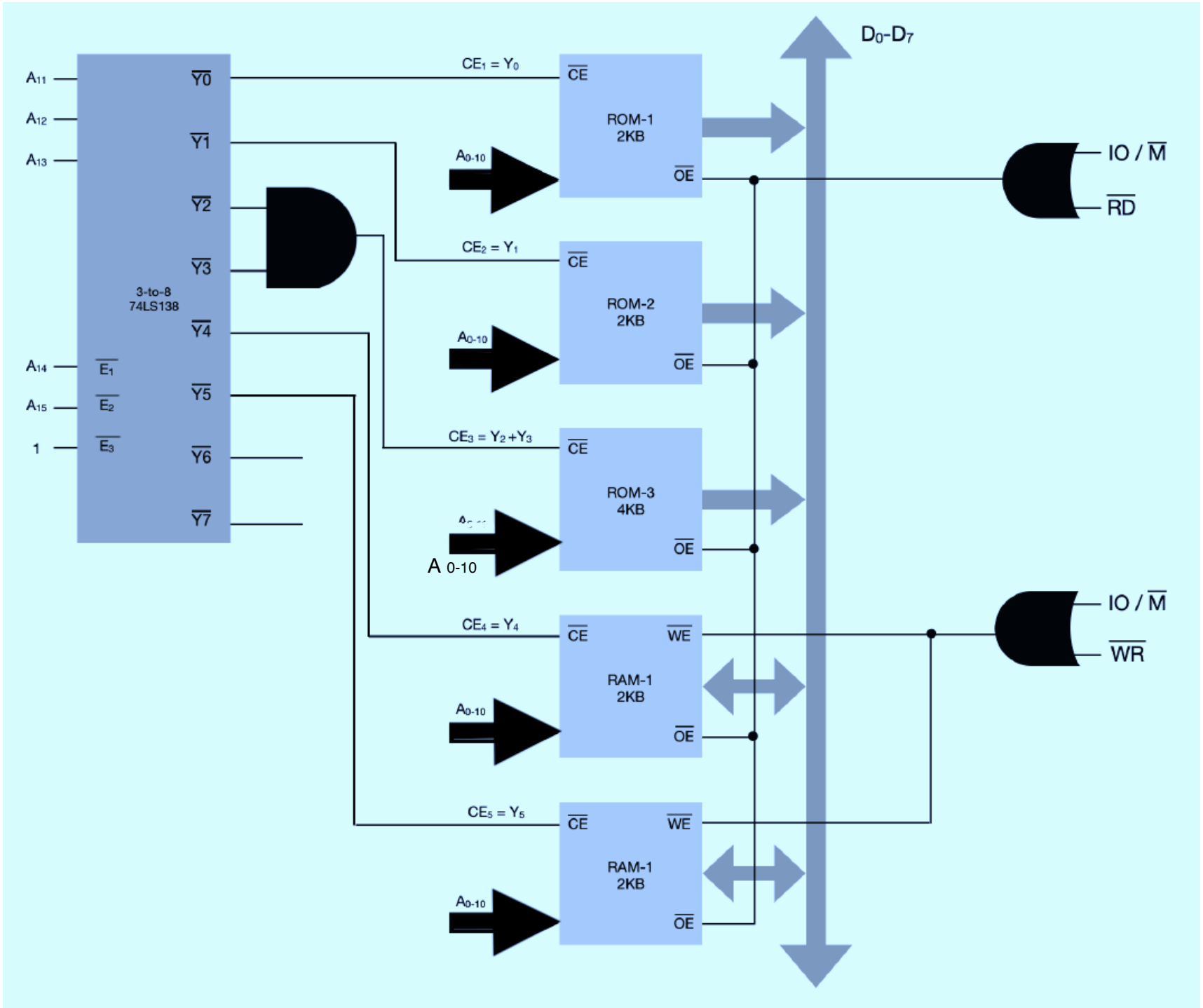
Άσκηση 6

Χάρτης μνήμης

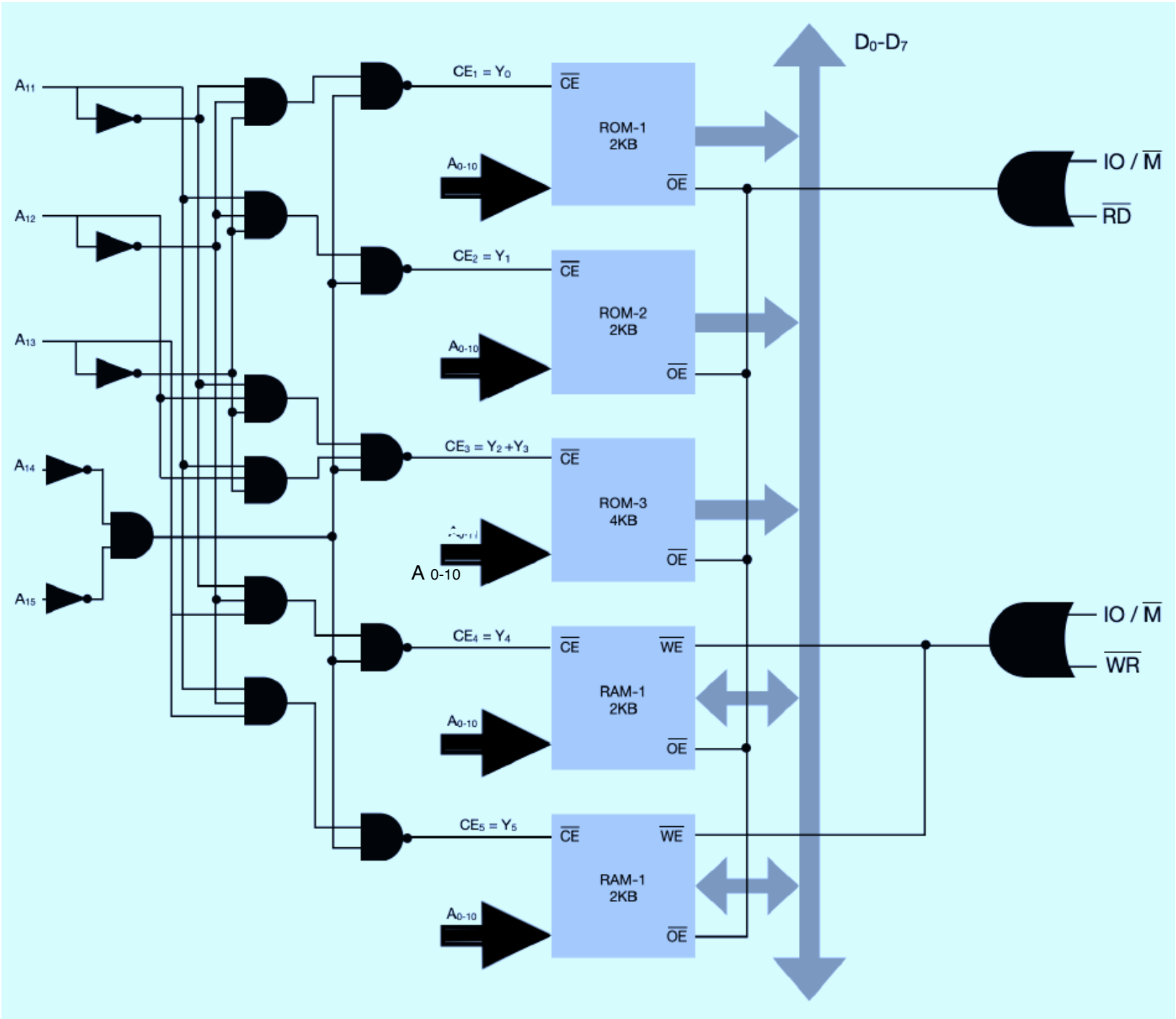
MEM	ADDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM 2K	0000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	07FFH	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
ROM 2K	0800H	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	0FFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
ROM 4K	1000H	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	1FFFH	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM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

Έχουμε συνολικά 8K ROM και 4K RAM. Συμφέρει η σελιδοποίηση 2K και άρα έχουμε 6 σελίδες. Η μνήμη ROM 2K έχει 2 σελίδες. Τα A_{15} και A_{14} είναι συνεχώς 0, οπότε βολεύει να τα ορίσουμε ως enable bits, E_1 και E_2, αντίστοιχα. Τα $A_{0\text{-}10}$ βοηθούν στον προσδιορισμό θέσης, εσωτερικά στην μνήμη. Τέλος, τα $A_{11\text{-}13}$ προσδιορίζουν την σελίδα μνήμης. Διαλέξαμε αυτά καθώς μεταβάλλονται και μας δίνουν 8 συνδυασμούς σελίδων και εμείς χρησιμοποιούμε 6.

(α) Αποκωδικοποίηση με αποκωδικοποιητή 3:8 (74LS138) και λογικές πύλες



(β) Αποκωδικοποίηση με λογικές πύλες



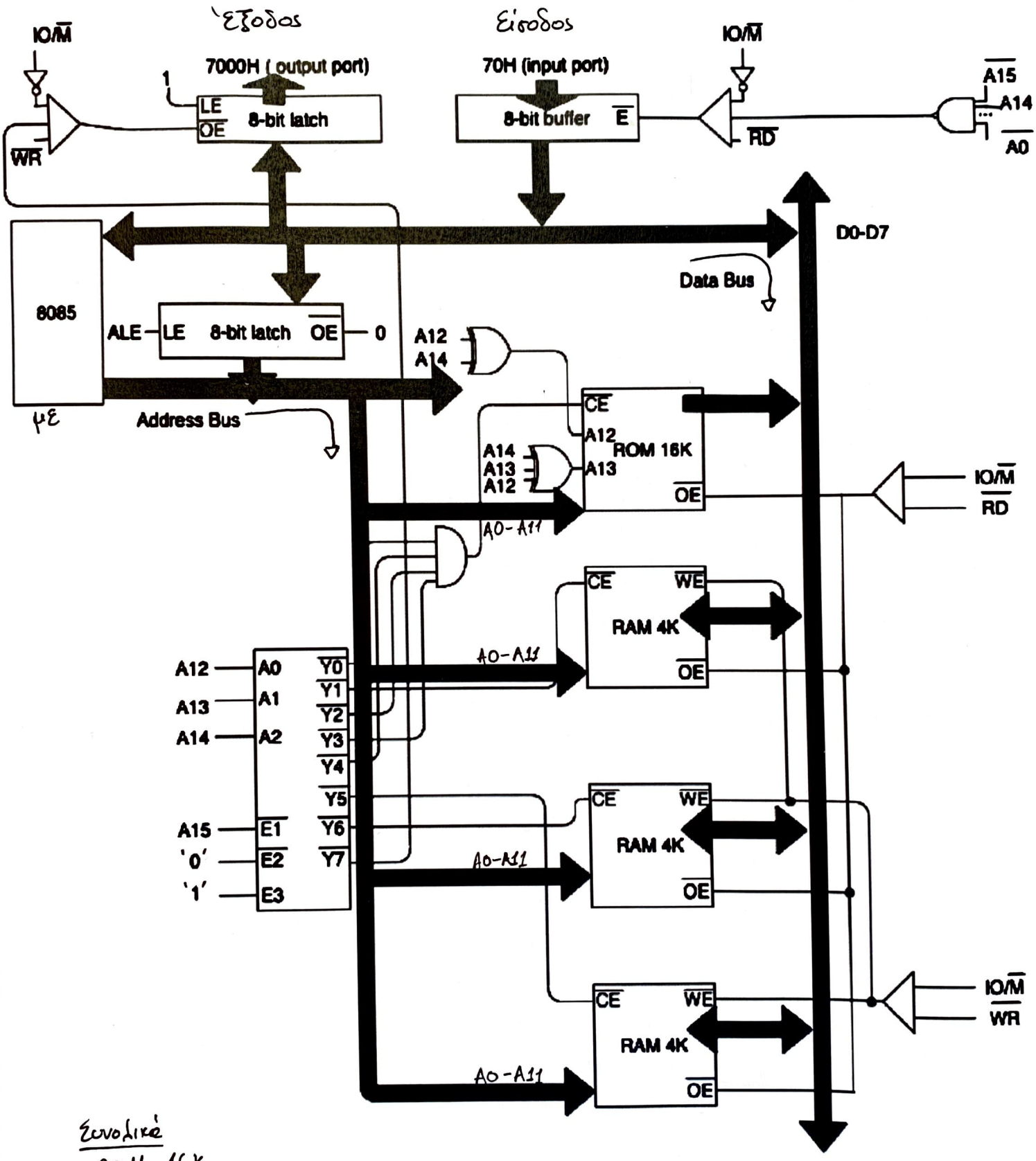
Άσκηση 7

Χάρτης μνήμης μΥ-Σ 8085

MEM	ADDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM 16K	0000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2FFFH	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
RAM 4K	3000H	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	3FFFH	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM 4K	4000H	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4FFFH	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RAM 4K	5000H	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	5FFFH	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM 16K	6000H	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	6FFFH	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
MAP I/O	7000H	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
STD I/O	0070H	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0

Έχουμε συνολικά 32K ROM και 12K RAM. Συμφέρει η σελιδοποίηση 4K και άρα έχουμε 11 σελίδες. Οι μνήμες ROM 16K έχουν 4 σελίδες. Το A_{15} είναι συνεχώς 0, οπότε βολεύει να τα ορίσουμε ως enable bit E_1, ενώ $E_2 = 0$. Τα $A_{0\text{-}11}$ βοηθούν στον προσδιορισμό θέσης, εσωτερικά στην μνήμη. Τέλος, τα $A_{12\text{-}14}$ προσδιορίζουν την σελίδα μνήμης. Διαλέξαμε αυτά καθώς μεταβάλλονται και επίσης μας δίνουν 8 συνδυασμούς σελίδων και εμείς χρησιμοποιούμε 6.

μΥ-Σ 8085



Συνολικά
ROM 16K
RAM 12K