2η Σειρά Συστημάτων Μικροεπεξεργαστών

1η Άσκηση:

Ερωτήματα α,β): Για τα ερωτήματα α,β παράλληλα με την φόρτωση των δεδομένων μετρήσαμε και το πλήθος των 1 που υπάρχουν στους δυαδικούς αριθμούς από 0-127. Για αυτούς τους σκοπούς, πέρα από την εισαγωγή της εντολής IN 10H για πλήρη πρόσβαση στην μνήμη, έχουμε την αρχικοποίηση του A στο 0(τον χρησιμοποιούμε ως ενδιάμεσο καταχωρητή αποθήκευσης του i-οστου αριθμού), τον E στο 8 για λόγους που θα γινουν κατανοητοι αργότερα και τον διπλό καταχωρητή στην διεύθυνση 0900 από όπου αρχίζουμε να αποθηκεύουμε τους αριθμούς. Η διαδικασία είναι απλή, αρχικά συγκρίνουμε τον αριθμό με το 128 και αν είναι μικρότερος συνεχίζουμε ενώ αν όχι μεταβαίνουμε στο END_LABEL του οποίου η λειτουργία θα επεξηγηθεί στην συνέχεια. Ακολούθως, αποθηκεύουμε το περιεχόμενο του A στην μνήμη και μετράμε τους άσσους της i-οστης εισόδου. Αυτό γίνεται μέσω δεξιάς ολίσθησης, για αυτό και ορίσαμε τον E στα 8 καθώς κάθε καταχωρητής έχει 8 bits. Εκτελώντας ισάριθμες φορές περιστροφή μεσω του κρατουμένου ελέγχουμε εάν κάθε bit είναι άσσος. Αν είναι αυξάνουμε τον διπλό καταχωρητή BC αλλιώς απλά μειώνουμε την τιμή του E. Έπειτα εάν τουτη είναι μη μηδενική συνεχίζουμε για να σαρώσουμε όλα τα bits και μόλις μηδενιστεί έχουμε τελειώσει την διαδικασία εύρεσης του αθροίσματος των άσσων μέχρι το i-οστο στοιχείο και μετά απλώς αυξάνουμε την τιμή του A για να λάβουμε τον επόμενο αριθμό και μετακινούμε τον “pointer” στην μνήμη μια θέση πιο κάτω. Τέλος, στο end_label απλώς για να δούμε την ορθή λειτουργία του κώδικα αποθηκεύουμε τον BC στις θέσεις 0B00-0B01 . Το snippet του κώδικα φαίνεται παρακάτω:

```
IN 10H ; command in order to have access to the whole memory
MVI A,00H ; initialize A to 00H
MVI E,08H ; initialize E to 8
LXI B,0000H ; initialize the counter of 1s to 0
LXI H,0900H ; load the address in the double register HL
KATAXWRHSH:
       CPI 80H ; compare it with 128 and if it is equal jump to the end
       JZ END_LABEL ; jump to the end label
       MOV M,A ; move the content of A to the memory location
       JMP COUNT_ACES ; jump to the counting subroutine
COUNT_ACES:
       RRC ; rotate right through carry
       JC INCREMENT_BC ; jump if carry is set
       JMP CONTINUE_2 ; jump to the next step
INCREMENT_BC:
       INX B ; increment B register
       JMP CONTINUE_2 ; jump to the next step
CONTINUE_1:
       MVI E,08H ; we reset E to 8 for the next iteration
       INX H ; move one position "down" in memory
       INR A ; increment the value of D
       JMP KATAXWRHSH ; loop until all the numbers are placed correctly
CONTINUE_2:
       DCR E ; decrement E
       JNZ COUNT_ACES ; if not zero, continue counting aces
       JMP CONTINUE_1 ; if zero, continue placing numbers
END_LABEL:
       MOV A,B
       STA 0B00H
       MOV A,C
       STA 0B01H
       END
```

Παραθέτουμε ακόμη και screenshot της αρχιτεκτονικής κατάστασης του συστήματος για να αποδείξουμε ότι τόσο οι αριθμοί έχουν φορτωθεί σωστά αλλά και ότι οι άσσοι είναι

448(01C0 στο δεκαεξαδικό):

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	00	0981	00	0982	00	0983	00	0984	00	0985	00

0B00	01	0B01	C0

Ερώτημα γ): Για αυτό το ερώτημα θα χρησιμοποιήσουμε την εισαγωγή των αριθμών και νέο κώδικα καθώς ο καταχωρητής D χρησιμοποιούνταν για άλλο σκοπό στην άσκηση μέχρι στιγμής. Όπως και πριν, παράλληλα με την φόρτωση των αριθμών έχουμε τον έλεγχο αν το input είναι >=10H και <=60H. Μέσω προγράμματος σε C υπολογίσαμε ότι το ζητούμενο πλήθος είναι 81(δηλαδή 51H) γεγονός που επιβεβαιώθηκε καθώς αποθηκεύσαμε τον D στην θέση μνήμης 0B00 H και συμφωνούσε με την θεωρητική ανάλυση:

```
IN 10H ; command in order to have access to the whole memory
MVI A,00H ; initialize A to 00H
MVI D,00H ; initialize the counter to 0
LXI H,0900H ; load the address in the double register HL
KATAXWRHSH:
       CPI 80H ; compare it with 128 and if it is equal jump to the end
       JZ END_LABEL ; jump to the end label
       CPI 10H ; we check if the number is more or equal to 10H
       JZ INCR_D ; if it is equal to 10H we increment the counter
       JNC CHECK ; if it is more we keep on checking
CONTINUE:
       MOV M,A ; move the content of A to the memory location
       INX H ; move one position "down" in memory
       INR A ; increment the value of D
       JMP KATAXWRHSH ; loop until all the numbers are placed correctly
INCR_D:
       INR D ; increment D
       JMP CONTINUE  ; move on with the next inputs
CHECK:
       CPI 60H ; we check if it less or equal to 60H and if so we increase D
       JZ INCR_D
       JC INCR_D
END_LABEL:
       MOV A,D ; we save the content of D into the accumulator
       STA 0B00H ; we save the count into 0B00 in order to see the result
       END
```

0B00	51

2η Άσκηση:

Για την λύση του προβλήματος αρχικά θα χρειαστούμε στον καταχωρητή D ένα χρονόμετρο και θα το αρχικοποιήσουμε στα 20 δευτερόλεπτα και για την ρουτίνα καθυστέρησης θέτουμε στον B την τιμή των 200ms. Ακολούθως, με κατάλληλη μάσκα ελέγχουμε το MSB και ανιχνεύουμε το pattern 010 και αν κάποια στιγμή δεν αλλάξει κάτι επαναφέρουμε τον έλεγχο στο κατάλληλο σημείο. Μετά, ελέγχουμε εάν το MSB είναι εκ νέου 1 και αν είναι μειώνουμε το χρονόμετρο και ελέγχουμε εάν έχουμε μη μηδενικό χρόνο. Αν ναι, συνεχίζουμε για την αναγνώριση του μοτίβου 010 εκ νέου αλλιώς επαναλαμβάνουμε και επαναρχικοποιούμε το ρολόι. Τέλος, εάν αναγνωρίσουμε εν τέλει την ακολουθία 010 στο πιο σημαντικό bit τότε έχουμε ανανέωση του χρόνου. Η υλοποίηση μας αποτυπώνεται στον παρακάτω κώδικα:

```
        MVI A,FFH ; INITIALIZE THE ACCUMULATOR TO 11111111
        STA 3000H ; SET OFF ALL THE LEDS DUE TO NEGATIVE LOGIC
        MVI D,64H ; TIMER OF 20 SECS = 100*0,2 SECS (100=64H)
        LXI B,00C8H ;B CONTAINS THE DELAY OF 200ms
START:
        LDA 2000H ; WE READ THE INPUT AND WE PLACE IT TO THE ACCUMULATOR
        ANI 80H ; RETAIN ONLY THE MSB
        CPI 00H ; CHECK IF THE FIRST BIT IS 0
        JZ OFF1 ; IF IT IS 0 WE MOVE TO THE LABEL OFF1
        JMP START ; ELSE ITERATE AGAIN
OFF1:
        LDA 2000H ; WE READ THE INPUT
        ANI 80H ; WE RETAIN ONLY THE FIRST BIT(MSB)
        CPI 80H ; WE CHECK IF IT IS 1 AND IF IT IS
        JZ ON1 ; WE JUMP ON LABEL ON1
        JMP OFF1 ; ELSE WE JMP BACK TO OFF1 WAITING FOR THE MSB TO BE 1
ON1:
        LDA 2000H ; WE READ AGAIN THE INPUT
        ANI 80H ; WE RETAIN THE FIRST BIT
        CPI 00H ; WE CHECK IF IT IS 0
        JZ OFF2 ; IF IT IS 0 WE MOVE TO OFF 2
        JMP ON1 ; IF NOT WE REREAD UNTIL IT BECOMES 0
        ; AT THIS POINT WE HAVE RECOGNIZED THE PATTERN 010 FOR THE MSB
OFF2:
        LDA 2000H
        ANI 80H
        CPI 80H ;WE REREAD THE INPUT AND WE SEE IF THE MSB IS 1
        JZ ON2 ;IF IT IS WE MOVE INTO ON2 LABEL
        MVI A,00H ; ELSE WE KEEP ALL THE LEDS ON
        STA 3000H ; AND WE SHOW THEM
        CALL DELB ; WE CALL DELAY ROUTINE
        DCR D ; WE DECREMENTCTHE 20 secs TIMER
        MOV A,D ; WE MOVE THE RESULT TO THE ACCUMULATOR
        CPI 00H ; WE CHECK IF THE TIME HAS RUN OUT
        JNZ OFF2 ;IF THE RESULT IS NOT 0 WE ITERATE TO OFF2
        MVI A,FFH ;IF IT IS ZERO WE SET ALL THE LEDS OFF
        STA 3000H ;WE SHOW THEM
        MVI D,64H ;RESET THE TIMER
        JMP OFF1 ; RESTART THE PROCESS
```

```
ON2:
       LDA 2000H
       ANI 80H
       CPI 00H ; WE HAVE THE MSB AND WE CHECK IF IT ZERO
       ;WE HAVE SEEN UP TO NOW THE 01010
       JZ RESTART ; WE CHEKK IF THE RESULT IS O AND IF SO WE MOVE TO RESTART WHERE
       ;WE REINITIALIZE THE TIMER
       MVI A,00H ; WE SET ALL THE LEDS ON
       STA 3000H ; WE SHOW THEM
       CALL DELB ; WE CALL THE ROUTINE OF DELAY
       DCR D ; WE DECREMENT THE TIMER
       MOV A,D
       CPI 00H
       JNZ ON2 ; WE CHECK THAT THE TIMER IS NOT OUT
       MVI A,FFH ;WE SET ALL THE LEDS OFF
       STA 3000H
       MVI D,64H ; WE RESTART THE TIMER
       JMP OFF1 ; ITERATE OVER AGAIN
RESTART:
       MVI D,64H ; WE REINITIALIZE THE TIMER
       JMP OFF2 ; JMP TO OFF2
END
```

3η Άσκηση:

α) Για την υλοποίηση του πρώτου ζητουμένου έχουμε ότι θα πρέπει να λαμβάνουμε το input(αποθηκεύοντας το στον συσσωρευτή) και εν συνεχεία εκτελώντας δεξιά ολίσθηση με συμμετοχή του κρατουμένου και ουσιαστικά κάθε φορά μετακινούμε στην θέση του κρατουμένου την τιμή του τρέχοντος LSB μέχρι αυτό να γίνει ένα. Ο κώδικας είναι ο εξής:

```
IN 10H ; FULL ACCESS TO MEMORY
ARXH:
       MVI A,FFH ;INITIALIZE THE LEDS TO BE ALL CLOSED
       MVI C,01H ;INITIALIZE THE FIRST POSSIBLE LED TO BE ONE
       STA 3000H ;BY STORING THE VALUE OF A WE SET OFF THE LEDS
       LDA 2000H ;READ INPUT
       CPI 00H ;WE CHECK IF THE INPUT IS 00H
       JZ BASE_CASE ;IF NONE OF THE BITS OF THE INPUT IS ONE JMP TO BASE CASE

LOOP_1:
       RAR ; RIGHT SHIFT WITH THE CARRY ON
       MOV B,A ;WE KEEP A COPY OF THE SHIFTER REG A
       JC SET_LEDS ; MOVE TO THE SHOW LEDS IF WE HAVE A CURRY IN THE CURRENT "INDEX"
       MOV A,C ; WE MOVE THE CONTENT OF C TO A
       RLC ; WE SHIFT IT ONE POSITION TO THE LEFT BECAUSE C SHOWS WHICH LED SHOULD BE ON
       MOV C,A ; RECOVER THE VALUE OF C
       MOV A,B ; RECOVER THE VALUE OF A
       JMP LOOP_1 ; ITERATE SINCE NO CURRY FOUND(NO 1 BIT YET)
BASE_CASE:
       MVI A,FFH ; WE SECURE THAT THE VALUE OF THE ACCUMULATOR IS SETTING ALL THE LEDS 1
       STA 3000H ; DUE TO THE NEGATIVE LOGIC WE SET THEM OFF
       JMP ARXH ; RESTART
SET_LEDS:
       MOV A,C ; C IS MOVING INTO THE ACCUMULATOR
       CMA ; COMPLEMENT THE VALUE OF THE ACCUMULAGTOR TO THE NEGATIVE LOGIC OF LEDS
       STA 3000H ;PRINT THE OUTPUT
       JMP ARXH ;RESTART
END
```

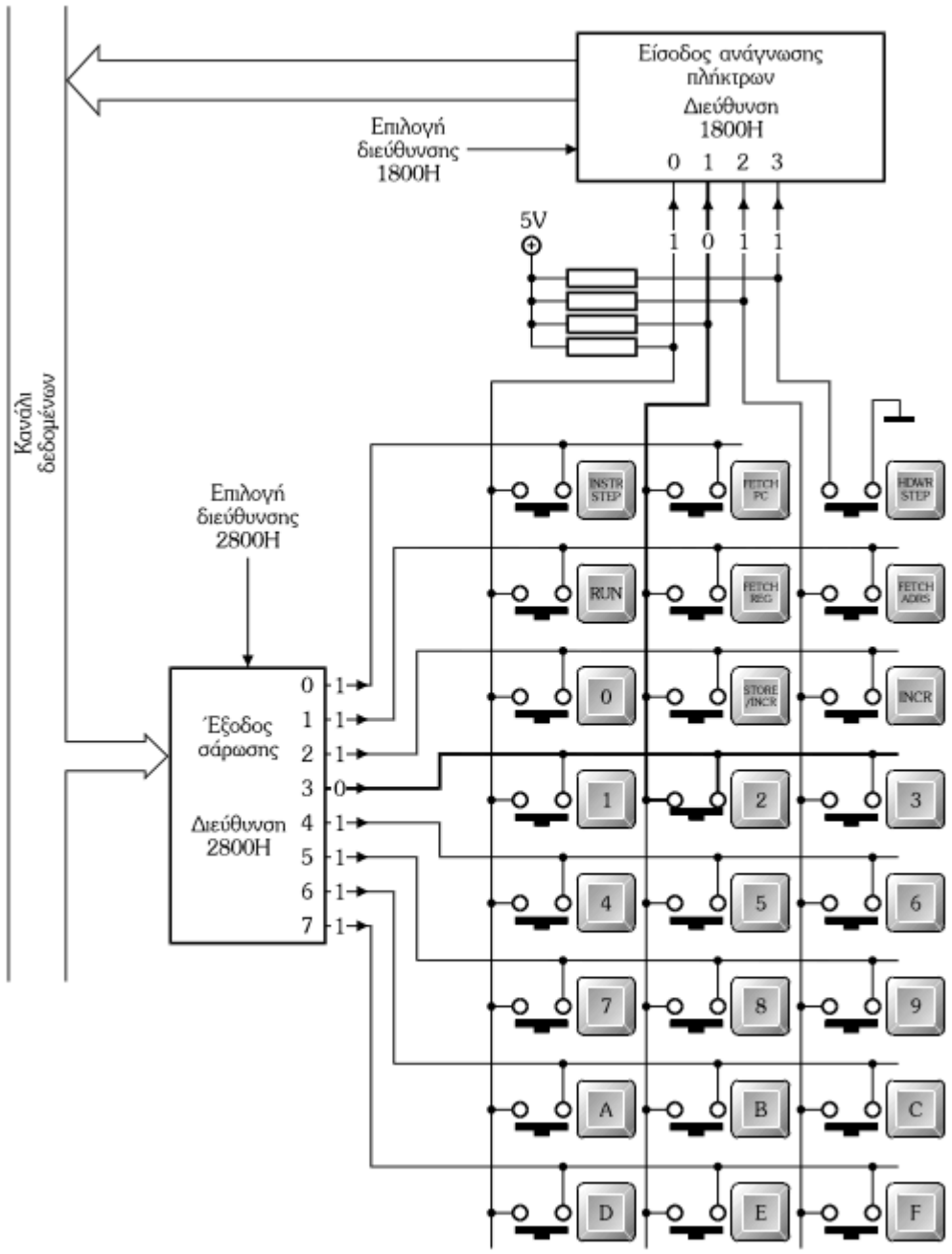
Αρχικά τοποθετούμε στον A την τιμή FF έτσι ώστε όλα τα leds να είναι OFF και αρχικοποιούμε την τμή 1 στον C που δηλώνει το πρώτο Led που είναι πιθανό να ανάψει.

Ακολούθως, εάν το αποτέλεσμα είναι 0 τότε μεταπηδάμε στην βασική περίπτωση όπου σιγουρεύουμε ότι όλα τα led είναι 0 και κάνουμε διαρκώς επανάληψη. Σε αντίθετη περίπτωση, εκτελούμε δεξιά ολίσθηση με συμμετοχή κρατουμένου και κρατάμε ένα αντίγραφο του συσσωρευτή στον καταχωρητή B. Εαν έχουμε κρατούμενο πηγαίνουμε στην ρουτίνα εξυπηρέτησης και ανοίγουμε όλα τα κατάλληλα led αλλιώς θέτουμε τον A ίσο με την τωρινή πιθανή θέση του led που είναι ON εκτελούμε μια αριστερή ολίσθηση και επαναφέρουμε την τιμή του C και A και εν τέλει στο print απλώς τυπώνουμε την τιμή του C.
β) Αρχικά, καλούμε την ρουτίνα KIND έτσι ώστε στον συσσωρευτή να αποθηκευτεί το αποτέλεσμα της εισόδου απο το πληκτρολόγιο και ελέγχουμε εάν είναι μηδεν και αν είναι πηγαίνουμε στο label NOT_ALLOWED και ομοίως αν το input είναι μεγαλύτερο του 9 επίσης πηγαίνουμε στην ρουτίνα εξυπηρέτησης του μη επιτρεπόμενου αποτελέσματος. Εν συνεχεία εργαζόμαστε χρησιμοποιώντας τις δυνάμεις του δύο. Στον καταχωρητή C το τελικό αποτέλεσμα ενώ το περιεχόμενο του D είναι η τρέχουσα δύναμη του 2, ενώ παράλληλα έχουμε ένα αντίγραφο του accumulator στον καταχωρητή B. Στην ρουτίνα POWERS ελέγχουμε εάν η τρέχουσα του B είναι 1 τότε σταματάμε. Αυτό συμβαίνει διότι πρέπει να τυπώνουμε από το τρέχον led μέχρι και το πιο αριστερό που υπάρχει. Έπειτα, μεταφέρουμε το τελικό αποτέλεσμα του C στον συσσωρευτή και του προσθέτουμε την τρέχουσα τιμή της δύναμης του 2 και στην συνέχεια ελαττώνουμε την τιμή του B και για τον D εκτελούμε αριστερή ολίσθηση και ουσιαστικά τον διπλασιάζουμε και κάνουμε iterate. Τέλος, στην finish αποθηκεύουμε την τιμή του τελικού αποτελέσματος στον συσσωρευτή και

τυπώνουμε το αποτέλεσμα στην θύρα εισόδου.

```
ARXH:
       CALL KIND ; CALL THE ROUTINE KIND TO GET INPUT FROM KEYBOARD
       CPI 00H ;CHECK IF THE INPUT IS ZERO
       JZ NOT_ALLOWED ;IF IT IS JMP TO ZERO WERE WE JUST OSET OFF ALL THE LIGHTS
       CPI 09H ;WE CHECK IF THE INPUT IS LESS THAN 9
       JNC NOT_ALLOWED ;IF IT NOT WE SET ALL THE LIGHTS OFF ELSE WE MOVE ON
       MOV B,A ;GET A COPY OF A INTO REG B
       MVI C,00H ; C CONTAINS THE FINAL RESULT AND IT USES THE POWERS OF TWO
       MVI D,01H ;WE INITIALIZE D TO 1 WHICH IS THE FIRST POWER OF 2


POWERS:
       MOV A,B ; MOVE THE VALUE OF THE ACCUMULATOR BACK TO A
       CPI 01H ; CHECK IF THE RESULT IS ONE AND IF SO WE JUMP TP FINISH
       JZ FINISH ; WE CHOOSE 1 BECASUE WE WANT THE LEDS FROM THE INPUT
       ;LOCATION AND THEN TO BE ON
       MOV A,C ; A GETS THE CONTENT OF REGISTER C
       ADD D ; WE ADD THE NEXT POWER OF TWO
       MOV C,A ; AND WE STORE THE NEW VALUE TO C
       DCR B ;B = B-1
       MOV A,D ; A TAKES THE CURRENT POWER OF AND IT SHIFTS IT ONE POSITION LEFT
       RLC ; PRACTICALLY IN DECIMAL D=2*D ;
       MOV D,A ; WE MOVE THE FRSH RESULT TO REGISTER D
       JMP POWERS ; ITERATE

FINISH:
       MOV A,C ; SAVE THE FINAL RESULT TO A
       JMP OUTPUT

OUTPUT:
       STA 3000H ; SAVE THE RESULT IN THE OUTPUT PORT
       JMP ARXH ; RESTART THE PROGRAM


NOT_ALLOWED:
       MVI A,FFH ; SET ALL THE LEDS OFF
       STA 3000H ; SAVE THE RESULT
       JMP ARXH ; IN ORDER PROGRAM TO RUN FOREVER
END
```

γ) Πριν παραθέσουμε τον κώδικα της άσκησης θα εξηγήσουμε πώς γίνεται η ανάγνωση του πληκτρολογίου απο το microlab.



Σχήμα 1. Διάγραμμα interface του πληκτρολογίου

Πρακτικά, ενεργοποιούμε κάθε γραμμή μηδενίζοντας κατάλληλα αυτή που επιθυμούμε και θέτοντας ένα όλες τις υπόλοιπες. Αυτό γίνεται χρησιμοποιώντας την διεύθυνση 2800H απο την οποία αποκωδικοποιείται το αποτέλεσμα και μετά κάθε πλήκτρο επιλέγεται αγνοώντας τα πρώτα 5 MSB και αν έχουμε το i-οστο πλήκτρο εκ νέου με αρνητική λογική ενεργοποιούμε την κατάλληλη στήλη. Τέλος αποθηκεύουμε το αποτέλεσμα στην διεύθυνση στην οποία η ρουτίνα εξυπηρέτησης της απεικόνισης στο 7 segment screen led. Ο κώδικας

είναι ο εξής:

```
ARXH:
       IN 10H ; HAVE FULL ACCESS TO THE WHOLE MEMORY
       LXI H,0A00H ; STARTING ADDRESS OF SAVING THE BLOCK OF DATA
       MVI B,04H ; INITIALIZATION OF B TO 4
L1:
       MVI M,10H ;
       INX H ; INCREMENT THE HL REGISTER
       DCR B ; B = B-1
       JNZ L1 ; ITERATE TILL B IS ZERO
LINE0:
       MVI A,FEH ; SCAN PORT 11111110 -LINE SELECT
       STA 2800H
       LDA 1800H ; READ THE COLUMNS
       ANI 07H ; KEEP ONLY THE LAST THREE LSBS THE OTHER HAVE UNDETERMINED VALUES
       MVI C,86H ; C IS THE CODE OF EACH BUTTON
       CPI 06H ; BUTTON INSTR STEP
       JZ SHOW
       MVI C,85H
       CPI 05H       ; BUTTON FETCH PC
       JZ SHOW
                     ; IGNORING THE HDWR_STEP BUTTON
LINE1:
       MVI A,FDH ; SCAN PORT 11111101 - LINE SELECT
       STA 2800H
       LDA 1800H ; READ_COLUMNS
       ANI 07H ; RETAIN ONLY THE 3 MSBS
       MVI C,84H
       CPI 06H ; BUTTON RUN
       JZ SHOW
       MVI C,80H
       CPI 05H ; BUTTON FETCH_REG
       JZ SHOW
       MVI C,82H
       CPI 03H ; BUTTON FETCH_ADDRS
       JZ SHOW
```

```
LINE2:
        MVI A,FBH ; SCAN PORT 11111011 - LINE SELECT
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,00H
        CPI 06H ; BUTTON 0
        JZ SHOW
        MVI C,83H
        CPI 05H ; BUTTON STORE/INCR
        JZ SHOW
        MVI C,81H
        CPI 03H ; BUTTON DECR
        JZ SHOW
LINE3:
        MVI A,F7H
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,01H ; BUTTON 1
        CPI 06H
        JZ SHOW
        MVI C,02H ; BUTTON 2
        CPI 05H
        JZ SHOW
        MVI C,03H ; BUTTON 3
        CPI 03H
        JZ SHOW
LINE4:
        MVI A,EFH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,04H
        CPI 06H ; BUTTON 4
        JZ SHOW
        MVI C,05H
        CPI 05H ; BUTTON 5
        JZ SHOW
        MVI C,06H
        CPI 03H ; BUTTON 6
        JZ SHOW
```

```
LINE5:
       MVI A,DFH
       STA 2800H
       LDA 1800H
       ANI 07H
       MVI C,07H
       CPI 06H ; BUTTON 7
       JZ SHOW
       MVI C,08H
       CPI 05H ; BUTTON 8
       JZ SHOW
       MVI C,09H
       CPI 03H ; BUTTON 9
       JZ SHOW
LINE6:
       MVI A,BFH
       STA 2800H
       LDA 1800H
       ANI 07H
       MVI C,0AH
       CPI 06H ;BUTTON A
       JZ SHOW
       MVI C,0BH
       CPI 05H ;BUTTON B
       JZ SHOW
       MVI C,0CH
       CPI 03H ;BUTTON C
       JZ SHOW
LINE7:
       MVI A,7FH
       STA 2800H
       LDA 1800H
       ANI 07H
       MVI C,0DH
       CPI 06H ;BUTTON D
       JZ SHOW
       MVI C,0EH
       CPI 05H ;BUTTON E
       JZ SHOW
       MVI C,0FH
       CPI 03H ; BUTTON F
       JZ SHOW
       JMP ARXH ; IF NONE OF THE BUTTON IS PRESSED THEN PERFORM THE SCAN AGAIN
```

```
SHOW:
       LXI H,0A04H ; LOAD THE DESIRED ADDRESS TO HL REGISTER
       MOV A,C ; WE STORE THE VALUE OF C(CODE) INTO THE ACCUMULATOR
       ANI 0FH ; WE KEEP ONLY THE LAST 4 BITS
       MOV M,A ; WE PLACE THEM IN THE MEMORY LOCATION MEANING THE FIFTH BIT OF THE SEVEN
       INX H  ; WE MOVE INTO THE NEXT MEMORY LOCATION
       MOV A,C ; WE PUT THE VALUE OF CODE INTO THE ACCUMULATOR
       ANI F0H ; WE KEEP THE 4 MSBS
       RLC ; BY PERFORMING 4 SHIFTS TO THE LEFT WE MAKE THEM LSBS
       RLC
       RLC
       RLC
       MOV M,A ; WE PRACTICALLY MOVE THEM TO THE 6TH BIT OF THE SCREEN LED
       LXI D,0A00H ; NECESSARY TRANSFER OF THE BLOCK IN ORDER TO BE READ BY DCD
       CALL STDM
       CALL DCD ; ROUTINES FOR REPRESENTING THE OUTPUT
       JMP ARXH ; ITERATE AGAIN AND AGAIN
END
```

4η 'Ασκηση:

Ακολουθούμε την λογική της ασκήσης του βιβλίου όπου ζητούταν η υλοποίηση του ενός intergrated με 4 πύλες NAND και τώρα το μόνο που αλλάζει είναι οι συνθήκες οι οποίες θέτουν ένα το output των πυλών. Πρακτικά, διαβάζουμε τα δύο bits και σώζουμε το αποτέλεσμα σε έναν καταχωρητή και στην συνέχεια shiftαρουμε το αποτέλεσμα(ενώ κρατάμε μέρος του αποτέλεσματος για την OR) και συνεχίζουμε κάνοντας επανάληψη μέχρι να δημιουργηθούν όλα τα outputs.

```
LDA 2000H ; LOAD THE INPUT PORT TO THE ACCUMULATOR
MOV D,A ; COPY OF INPUT INTO D
MVI C,03H ; MASK FOR MAINTAIN ONLY THE TWO LSBS
ANA C ; RETAIN ONLY THE TWO LAST DIGITS
CMP C ; THE ONLY WAY THE OUTPUT OF AND TO BE ONE
JZ FIRST_AND_RESULT
MVI B,00H
MVI E,00H
JMP CONTINUE
FIRST_AND_RESULT:
       MVI B,01H
       JMP CONTINUE
CONTINUE:
       MOV A,D ; RETAIN THE INPUT
       RRC
       RRC ; IN ORDER TO GTE THE NEXT TWO BITS
       ANA C
       CMP C
       JZ SECOND_AND_RESULT
       JMP CONTINUE_2
SECOND_AND_RESULT:
       MVI E,01H
       JMP CONTINUE_2
CONTINUE_2:
       MOV A,B ; FIRST RESULT OF AND IN ACCUMULATOR
       ORA E ;
       MOV H,A ; H HAS THE FIRST CORRECT INPUT
       MOV A,E ;
       RLC
       ORA H
       MOV L,A ; L HAS THE TWO LSB BITS
       MVI B,00H ;REINITIALIZE TO 0
       MVI E,00H ;REINITIALIZE TO 0
       MOV A,D
       ANI 30H
       RRC
       RRC
       RRC
       RRC
       CPI 01H
       JZ FIRST_XOR_RESULT
       CPI 02H
       JZ FIRST_XOR_RESULT
       JMP CONTINUE_3
```

```
CONTINUE_3:
        MOV A,D
        ANI C0H
        RRC
        RRC
        RRC
        RRC
        RRC
        RRC
        CPI 01H
        JZ SECOND_XOR_RESULT
        CPI 02H
        JZ SECOND_XOR_RESULT
        JMP CONTINUE_4
SECOND_XOR_RESULT:
        MVI E,01H
        JMP CONTINUE_4
CONTINUE_4:
        MOV A,B
        ORA E
        MOV H,A ; H HAS THE FIRST OUTPUT
        MOV A,E
        RLC
        ORA H
        RLC
        RLC
        ORA L
        CMA
        STA 3000H
END
```

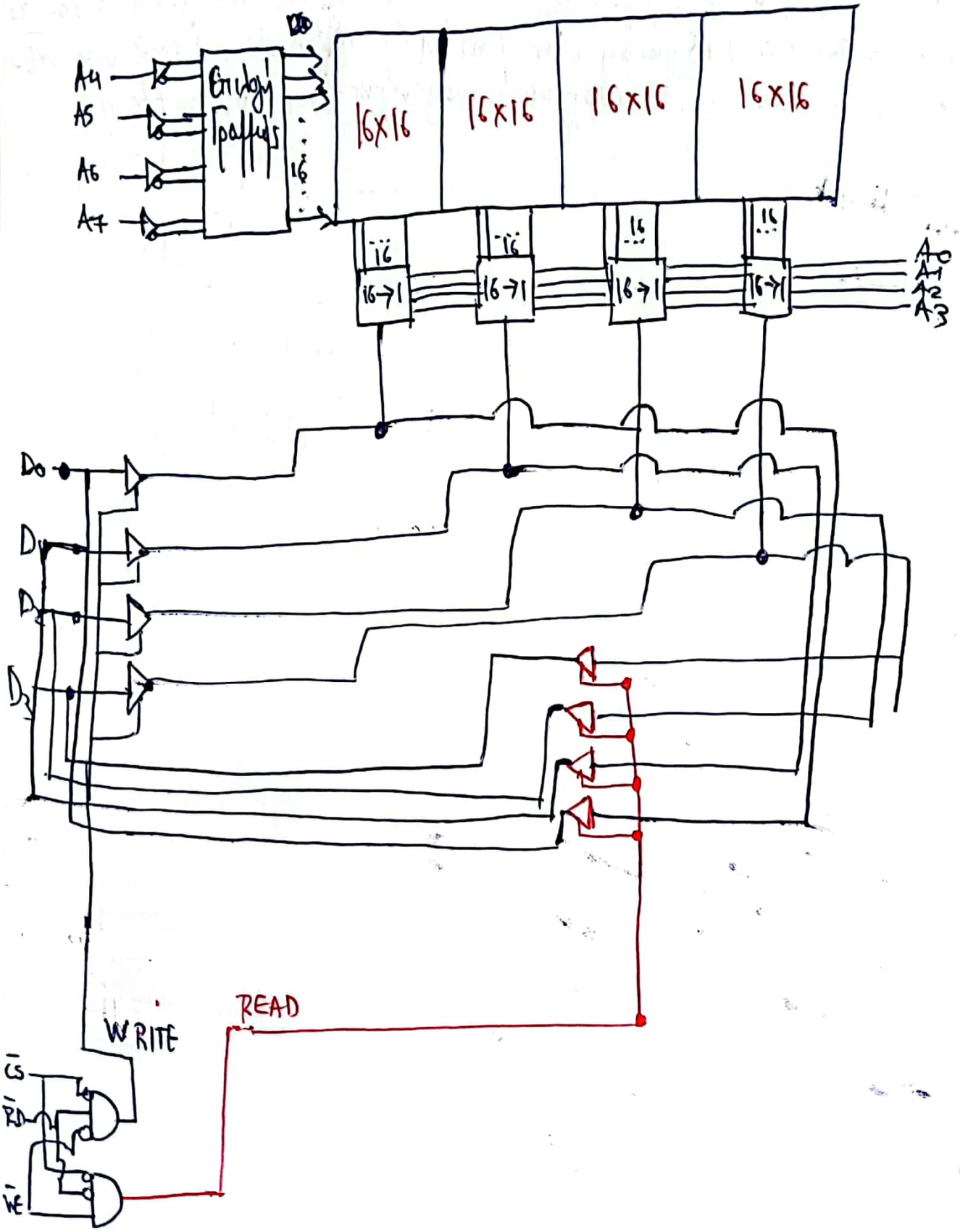
5η Άσκηση

Από την άσκηση και την εκφώνησή της διαπιστώνουμε ότι έχουμε 4 bits εισόδου-εξόδου δεδομένων και 256 cells μνήμης. Για να φτιάξουμε μια τέτοια SRAM πρέπει να τις οργανώσουμε το μέγεθός της έτσι ώστε ο επιλογέας γραμμής να μην έχει πολλά bits έτσι ώστε να μην επιβαρυνθούμε χρονικά. Άρα μας να μην έχει πολλά bits έτσι ώστε να μην επιβαρυνθούμε χρονικά. Άρα

$$256 = 2^8 = 2^4 \cdot 2^4$$

Άρα 4 bits για επιλογή γραμμής έστω τα $A_4 - A_7$ και τα $A_0 - A_3$ τα υπόλοιπα για αυτολειτουργοποίηση. Έχουμε τον εξής:

ΣΧΗΜΑ 4



Τώρα θα επεξηγηθεί η λειτουργία της SRAM. Μέσω της επιλογής γραμμής των τεσσάρων bits (MSB) επιλέγουμε γραμμή και μέσω των τεσσάρων LSB από φράζουμε στήλη. Από την άλλη, εάν έχουμε λειτουργία εγγραφής ενεργοποιούνται οι buffers εισόδου και τα δεδομένα D0-D3 και μέσω των MUX (είναι προς τις δυο κατευθύνσεις τόσο για εγγραφή όσο και για ανάγνωση) και περνάνε τα δεδομένα στις επιλεγμένες θέσεις (γραμμή, στήλη). Προφανώς όταν έχουμε λειτουργία εγγραφής, οι αποκωδικοποιητές της αναγνώσης είναι off. Σε περίπτωση αναγνώσης, οι buffers εγγραφής off και χρησιμοποιώντας την επιλογή γραμμής και τα LSBs αποφασίζουμε ποια στοιχεία δεδομένων πηγαίνει στο output D0 - D3. Όταν το σήμα $\overline{CS}$ γίνει μηδέν ενεργοποιείται η λειτουργία μνήμης. Εν συνεχεία, αν το $\overline{WE}$ γίνει 0 και το $\overline{RD}$ γίνει ένα τότε έχουμε εγγραφή στη μνήμη (μπλε buffers) ενώ αν $\overline{WE}=1$ και $\overline{RD}=0$ τότε έχουμε λειτουργία ανάγνωσης (κόκκινοι buffers)

6η Άσκηση

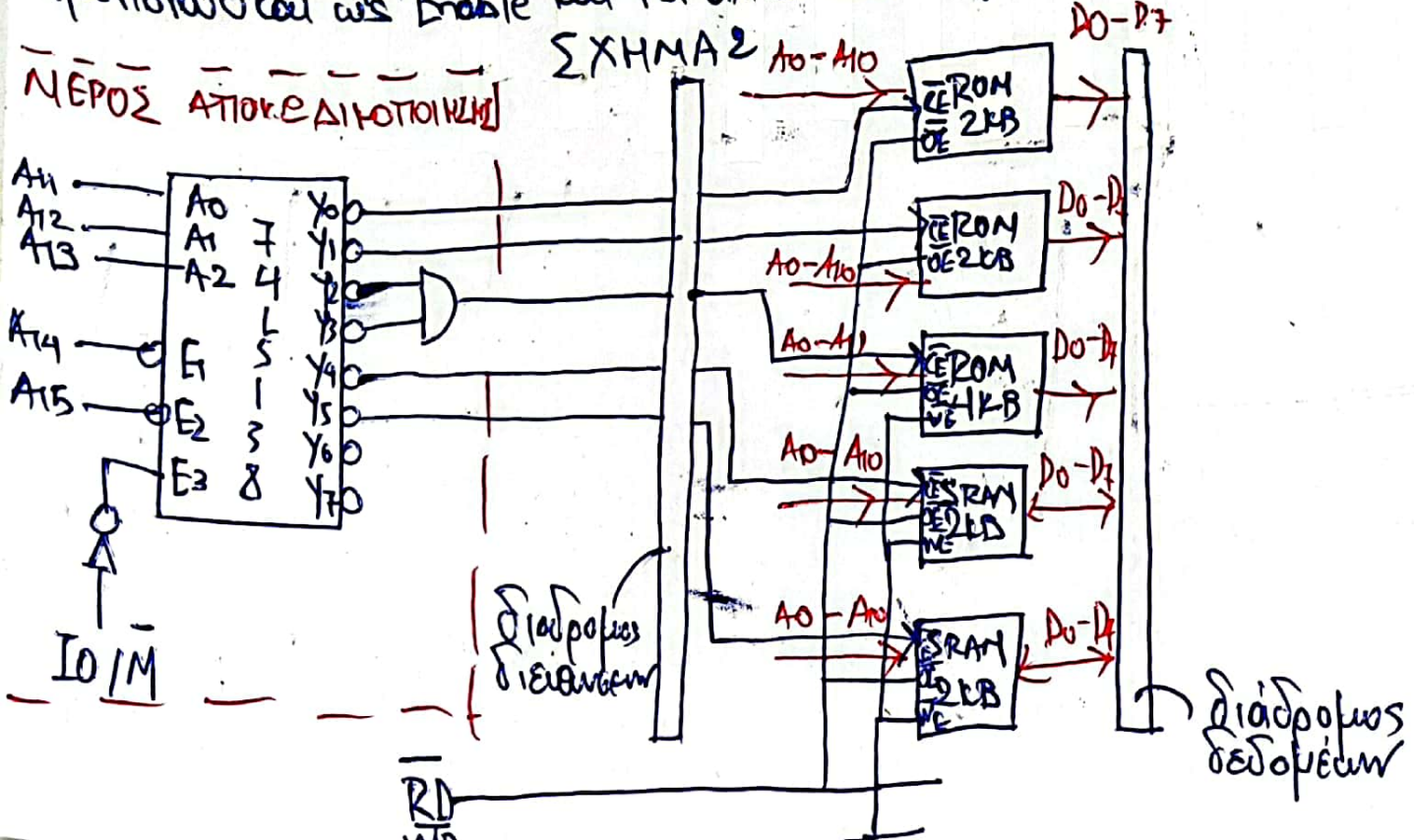
Για την υλοποίηση της άσκησης έχουμε ότι η ROM αρχίζει από την διεύθυνση 0000H και καταλαμβάνει 8KB και ακολουθείται από RAM 4KB (θεωρούμε ότι έχουμε δύο ολοκληρωμένα των 2KB για την ROM και ένα 4KB με δύο ολοκληρωμένα των 2KB. Αρχή ομοίως, για την RAM θεωρούμε ότι έχουμε δύο ολοκληρωμένα των 2KB. Αρχή και έχουμε τον χάρτη μνήμης (θεωρούμε ότι έχουμε για την ROM πρώτα τα 2 ICs των 2KB και εν συνεχεία το 4KB). Προφανώς επιλέγουμε την μη πιο κάτω οπτικοποίηση:

παραμένουν σχήμα 1

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1η θέση 2KB
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	
0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	2η θέση 2KB
0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	3η & 4η θέση 2KB
0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	1	
0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	5η θέση 2KB
0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	
0	0	1	0	0	1	1	1	1	1	0	0	0	0	0	0	6η θέση 2KB
0	0	1	0	1	0	0	0	0	0	1	1	1	1	1	1	
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	

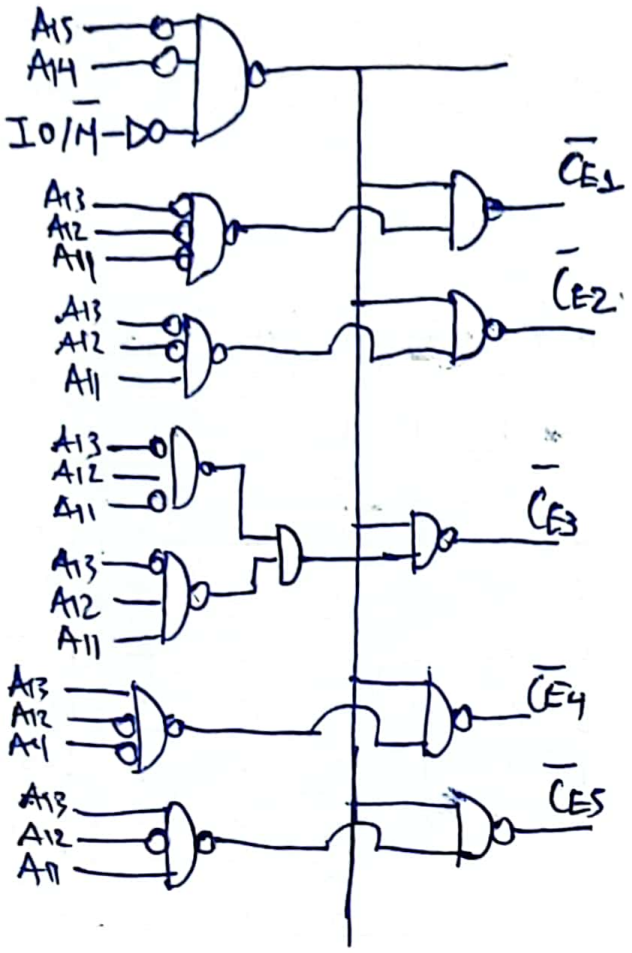
α) Για την υλοποίηση με αποκωδικοποίηση έχουμε τα δύο bits A_{14}-A_{15} χρησιμοποιούνται ως Enable και τα άλλα ως Input

ΣΧΗΜΑ 2



Το μόνο που απαιτείται στο ερώτημα β είναι το μέρος της αποκωδικοποίησης (2) και δίνεται ως εξής:

ΣΧΗΜΑ 3



Με $\overline{CE_i}$ ορίζουμε το CE του i-οστού στοιχείου μνήμης. Έχουμε και στα δυο είδη αποκωδικοποιήσεων το IO/M αναγραμμένο δείχνοντας ότι αυτοί αναφέρονται στην μνήμη. Στο ΣΧΗΜΑ 3 προφανώς ακολουθείται το σπαστό υποσύνολο του σχήματος 2 αλλά η δεύτερη αλοποίηση είναι πλουσιότερη σε πύλες NAND δεδομένου ότι καταλήγει σε περισσότερη σε όρους πυλών από την πρώτη.

7η Άσκηση

Αρχικά το concept είναι ότι θα χρησιμοποιήσουμε δύο αποκωδικοποιητές 3-8 έναν για τις μνήμες και έναν για τις ζητούμενες θύρες εισόδου - εξόδου. Για την χαρτογράφιση έχουμε μια μνήμη αρχικά των 8K ROM, ακολουθούν τρεις 4KB RAM και μια 8K ROM. A priori η χαρτογράφιση που θα επιλέξω θα χρησιμοποιεί βελιδοποίηση των 4KB (δηλαδή το μικρότερο resolution). Επομένως, ο χαρτης μνήμης είναι ο εξής:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8KB
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	4KB
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	4KB
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4KB
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	8KB
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	

Όπως αναφέραμε και ήταν λογικό οι πρώτες δύο σελίδες (4KB) αυτού του πρώτη ROM, η τρίτη, η τέταρτη και η πέμπτη στις τρεις RAM αντίστοιχα και οι 6η-7η του 8KB ROM. Ακόμη ένα τεχνικό θέμα που μπορούμε να χρησιμοποιήσουμε είναι ότι η 7η σελίδα μπορεί να εφαρμοστεί για την πόρτα 70Η. Για την memory mapped - Io θύρα 700Η και το συνολικό κύκλωμα

ΣΧΗΜΑ 5