# 2η Σειρά Ασκήσεων

Ειρήνη Στρουμπάκου 03121183

Χαράλαμπος Παπαδόπουλος 03120199

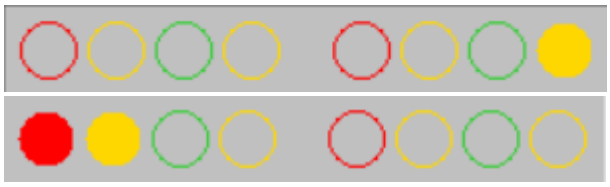
1) Το πρόγραμμα σε γλώσσα assembly είναι το εξής

```
;a
IN 10H
MVI A,00H           ;Store the value 00H in A
LXI H,0900H         ;Store the value 0900H in HL

START:
       MOV M,A      ;Store the (A) to (H)(L)
       INX H
       INR A
       CPI 80H      ;Check if it has reached 128, so it's greater than 127
       JC START



;b
       LXI B,0000H  ;Initialise the counter of ones
       MVI D,00H    ;Initialise the counter of 0 to 127 to 0 value

START2:
       INR D        ;Increase the counter of 0 to 127
       MOV A,D      ;Store each number we use in A
       CPI 80H      ;Check if it has reached 128, so it's greater than 127
       JNC ENDING
       MVI E,08H    ;Initialize a counter for the shifts
       JMP CHECK

CHECK:
       DCR E        ;Check if 7 shifts have been done
       JZ START2    ;If yes go to the next number
       RRC
       JNC CHECK    ;If the LSB was not 1 then continue the shifts
       INX B        ;Else increase the ones counter
       JMP CHECK    ;And continue with the shifts



;c
       MVI D,00H    ;Initialize the counter
       MVI A,00H    ;Set the first number in 0
       CHECK2:
       INR A
       CPI 80H      ;Check if it has reached 128, so it's greater than 127
       JNC ENDING
       CPI 10H      ;Check if greater than 10H
       JC STOP      ;If not, do not increase and go to STOP which jumps to
CHECK
       CPI 61H      ;Check if smaller than 61H, to include 60H
       JNC STOP     ;If not, do not increase and go to STOP which jumps to
CHECK
       INR D        ;If it passes all checks, increase counter D
       JMP CHECK2
       STOP:
       JMP CHECK2

ENDING:
       END
```

a)Τρέχουμε το πρόγραμμα και πράγματι παρατηρούμε τις τιμές που περιμέναμε στις αντίστοιχες διευθύνσεις:

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

b)Ελέγχουμε αν το πρόγραμμα λειτούργησε σωστά φορτώνοντας στη θέση 3000H αρχικά την τιμή του B και έπειτα του C. Καθώς έχουμε φορτώσει το συμπλήρωμα του A, βλέπουμε ότι πράγματι εμφανίζεται ο αριθμός 0000000111000000, που αντιστοιχεί στον δεκαδικό 448, που ήταν και ο επιθυμητός.



c)Ελέγχουμε αν το πρόγραμμα λειτουργεί και φορτώνουμε στη θέση 3000H το συμπλήρωμα του D και παίρνουμε τον αριθμό 01010001 που αντιστοιχεί στον δεκαδικό 81, που ήταν ο επιθυμητός.

2)

```
START:
       IN 10H
       LXI D,FFFFH          ;set delay
       MVI L,06H            ;multiply delay by L
;for 20 seconds L=16H and program virtual delay cursor at 2nd position

INPUT:
       LDA 2000H            ;first input

PUSH_BUTTON:
       ANI 01H              ;check LSB
       JNZ HIGH

LOW:                        ;LSB 0 subroutine
       MOV A,C              ;check counter
       CPI 02H              ;if counter=2 that means the previous state was
HIGH
       JZ LIGHTS_ON
```

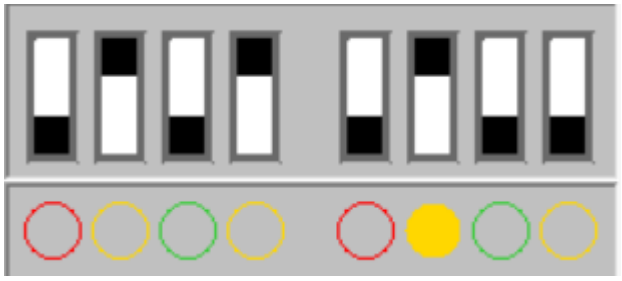
```
       JNZ LOOP_LOW

LOOP_LOW:                  ;stay here while LSB doesn't change from 0
       LDA 2000H
       CPI 01H
       JNZ LOOP_LOW
       INR C
       JMP HIGH

HIGH:
       MOV A,C
       CPI 00H             ;in case the first state was LSB=1. avoid tunging
on the_
       JZ INPUT            ;_lights after just HIGH-LOW sequence
       CPI 01H             ;if counter=1 that means the previous state was LOW
       JNZ LOOP_HIGH

LOOP_HIGH:                 ;stay here while LSB doesn't change from 1
       LDA 2000H
       CPI 01H
       JZ LOOP_HIGH
       INR C
       JMP LOW

LIGHTS_ON:
       MVI A,FFH
       CMA
       STA 3000H
       MVI C,00H           ;reset the counter for future inputs
       JMP DELAY

DELAY:
       DCR L
       JNZ SUBDELAY
       JMP END1

SUBDELAY:
       CALL CHECK_INPUT    ;keep checking for PUSH sequence
       DCX D
       MOV A,D
       ORA E
       JNZ SUBDELAY
       JMP DELAY

CHECK_INPUT:
       LDA 2000H
       CPI 00H
       JZ LOW2
       JNZ HIGH2

;same logic as before but now we need to return to the delay since we need to
keep
;counting down inbetween possible inputs.

HIGH2:
       MOV A,C
       CPI 01H             ;if counter=1 that means the previous state was LOW

       JZ HELP_HIGH2
       RET

HELP_HIGH2:
       INR C
       RET

LOW2:
       MOV A,C
       CPI 00H             ;if counter=0 that means the previous state was LOW
```

```
       JZ HELP_LOW2
       CPI 02H             ;if counter=2 that means the previous state was
HIGH
       JZ REFRESH_DELAY
       RET

HELP_LOW2:
       INR C
       RET

REFRESH_DELAY:
       LXI D,FFFFH
       MVI L,16H
       MVI C,00H
       JMP DELAY

END1:
       MVI A,00H
       CMA
       STA 3000H           ;turn off the lights
       END
```

3)

i)

```
LOOP1:
       MVI B,00H     ;Initialize the counter of shifts
       MVI C,09H     ;Initialize another counter to find if the input is 0
       LDA 2000H
START:
       INR B         ;For each shift, we increase our counter
       DCR C         ;We decrease the initialized to 9 counter and check if it
has become 0
       JZ ENDING1    ;If yes, A=0 and we go to ending1
       RRC           ;We make a right shift till we find 1
       JNC START     ;If we don't find 1, loop
FOUND1:
       MVI A,01H
       DCR B         ;We want to store 2^(B-1)
MAKE:
       RLC           ;We move A left till we reach the desirable bit position
       DCR B         ;We decrease the shift counter for each left shift
       JNZ MAKE      ;If the shifts counter doesn't reach 0, loop
       JMP ENDING2   ;If we find 1, got to ending2
ENDING1:
       MVI A,00H
       CMA
       STA 3000H
       JMP LOOP1     ;The program doesn't stop

ENDING2:
       CMA
       STA 3000H
       JMP LOOP1
       END
```



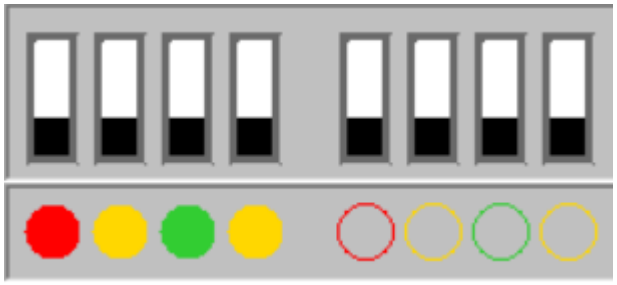
Πράγματι παίρνουμε ως αναμμένο μόνο το φως, που αντιστοιχεί στον πρώτο άσσο.

ii)

```
LOOP1:
       CALL KIND
       CPI 00H       ;If 0, go to ending1
       JZ ENDING1
       CPI 09H
       JZ ENDING1    ;If 9, go to ending1
       MOV B,A       ;Load in B the given number A
       MVI A,01H
       DCR B
       JZ NUM1       ;If B-1 is 0, then the given number is 1

MAKE:
       DCR B         ;Decrease B till it reaches 0, so I have made all the
       JZ ENDING2    ;necessary shifts
       RLC           ;Move left and add 1, so I create
       ADI 01H       ;my number in a form 00111
```

```
       JMP MAKE

NUM1:
       MVI A,00H     ;If given number is 1, turn on all LEDs
       JMP ENDING2

ENDING1:
       MVI A,00H     ;For 0 or 9 turn off all LEDs
       CMA
       STA 3000H
       JMP LOOP1

ENDING2:
       STA 3000H
       JMP LOOP1
       END
```

Με είσοδο 5, πράγματι ανάβουν τα λεντάκια 5,6,7,8:



iii)

```
IN 10H
LOOP1:
MVI A,10H
LXI D,0B00H          ;I create 4 empty spaces, because I only
STA 0B00H            ;want to show 2 numbers
STA 0B01H
STA 0B02H
STA 0B03H

MVI A,FEH            ;The 0 line has code 11111110
STA 2800H
LDA 1800H
ANI 07H              ;I want to keep the last 3 digits
CPI 06H              ;If 110 INSTR STEP
JZ INSTR_STEP
CPI 05H              ;If 101 FETCH PC
JZ FETCH_PC

MVI A,FDH            ;The 1 line has code 11111101
STA 2800H
LDA 1800H
ANI 07H              ;I want to keep the last 3 digits
CPI 06H              ;If 110 RUN
JZ RUNS
CPI 05H              ;If 101 FETCH REG
JZ FETCH_REG
CPI 03H              ;If 011 FETCH ADDRS
JZ FETCH_ADDRS

MVI A,FBH             ;The 2 line has code 11111011
STA 2800H
LDA 1800H
ANI 07H              ;I want to keep the last 3 digits
CPI 06H              ;If 110 ZERO
```

```
JZ ZERO
CPI 05H             ;If 101 STORE/INCR
JZ STORE_INCR
CPI 03H              ;if 011 DECR
JZ DECR

MVI A,F7H            ;The 3 line has code 11110111
STA 2800H
LDA 1800H
ANI 07H             ;I want to keep the last 3 digits
CPI 06H             ;If 110 ONE
JZ ONE
CPI 05H             ;If 101 TWO
JZ TWO
CPI 03H             ;If 011 THREE
JZ THREE

MVI A,EFH           ; The 4 line has code 11101111
STA 2800H
LDA 1800H
ANI 07H             ;I want to keep the last 3 digits
CPI 06H             ;If 110 FOUR
JZ FOUR
CPI 05H             ;If 101 FIVE
JZ FIVE
CPI 03H             ;If 011 SIX
JZ SIX

MVI A,DFH           ;The 5 line has code 11011111
STA 2800H
LDA 1800H
ANI 07H             ;I want to keep the last 3 digits
CPI 06H             ;If 110 SEVEN
JZ SEVEN
CPI 05H             ;If 101 EIGHT
JZ EIGHT
CPI 03H             ;If 011 NINE
JZ NINE

MVI A,BFH           ;The 6 line has code 10111111
STA 2800H
LDA 1800H
ANI 07H             ;I want to keep the last 3 digits
CPI 06H             ;If 110 A
JZ BUT_A
CPI 05H             ;If 101 B
JZ BUT_B
CPI 03H             ;If 011 C
JZ BUT_C

MVI A,7FH           ;The 7 line has code 01111111
STA 2800H
LDA 1800H
ANI 07H             ;I want to keep the last 3 digits
CPI 06H             ;If 110 D
JZ BUT_D
CPI 05H             ;If 101 E
JZ BUT_E
CPI 03H             ;If 011 F
JZ BUT_F
JMP LOOP1

INSTR_STEP:
       MVI A,06H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,08H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
```

```
       JMP ENDING

FETCH_PC:
       MVI A,05H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,08H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

RUNS:
       MVI A,04H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,08H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

FETCH_REG:
       MVI A,00H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,08H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

FETCH_ADDRS:
       MVI A,02H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,08H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

ZERO:
       MVI A,00H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       STA 0B05H           ;We then store in the next position the second
                    ;digit of the 2 digits number
       JMP ENDING

STORE_INCR:
       MVI A,03H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,08H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

DECR:
       MVI A,01H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,08H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

ONE:
       MVI A,01H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H
       JMP ENDING

TWO:
```

```
       MVI A,02H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

THREE:
       MVI A,03H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

FOUR:
       MVI A,04H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

FIVE:
       MVI A,05H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

SIX:
       MVI A,06H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

SEVEN:
       MVI A,07H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

EIGHT:
       MVI A,08H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

NINE:
       MVI A,09H           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

BUT_A:
       MVI A,0AH           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
```

```
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

BUT_B:
       MVI A,0BH           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

BUT_C:
       MVI A,0CH           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

BUT_D:
       MVI A,0DH           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

BUT_E:
       MVI A,0EH           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

BUT_F:
       MVI A,0FH           ;We first store the last digit of the 2 digits
number
       STA 0B04H
       MVI A,00H           ;We then store in the next position the second
       STA 0B05H           ;digit of the 2 digits number
       JMP ENDING

ENDING:
       CALL STDM           ;We portray the number we want in
       CALL DCD            ;the screen above the keyboard
       JMP LOOP1
       END
```

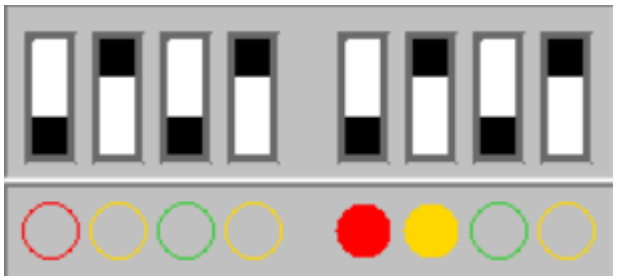
Το πρόγραμμα λειτουργεί ως εξής:

Για κάθε αριθμό γραμμής, υπάρχουν 3 κωδικοί ανάλογα τα 3 τελευταία ψηφία:06H,05H,03H. Ανάλογα τον κωδικό που λαμβάνουμε, μέσω jump πηγαίνουμε στη συνάρτηση του αντίστοιχου κουμπιού. Κάθε συνάρτηση έχει ως στόχο το να φορτώσει στην οθόνη μας τον επιθυμητό διψήφιο κωδικό κάθε εντολής για σύντομο χρονικό διάστημα. Για να πάρουμε λοιπόν 2 ψηφία, αρχικά στις 4 τελευταίες θέσεις φορτώνουμε το κενό(10H). Εάν δεν έχει εφαρμοστεί η εντολή κάποιου κουμπιού, τότε η οθόνη μας δεν δείχνει τίποτα.

4)

```
START:
       IN 10H
       LDA 2000H
       MVI C,00H
       MOV B,A

LOOP1:
       MOV A,C
       CPI 00H
       JZ AND0
       CPI 01H
       JZ AND1
       CPI 02H
       JZ XOR0
       CPI 03H
       JZ XOR1


AND0:
       MOV A,B
       ANI 01H             ;B0
       MOV D,A
       MOV A,B
       ANI 02H             ;A0
       RRC                 ;we bring A0 and B0 on the same position to compare
       ANA D               ;AND between A0 and B0
       MOV D,A             ;store the result in register D
       INR C
       JMP LOOP1

AND1:
       MOV A,B
       ANI 04H             ;B1
       MOV E,A
       MOV A,B
       ANI 08H             ;A1
       RRC                 ;we bring A1 and B1 on the same position to compare
       ANA E
       RRC                 ;X1 output
       MOV E,A             ;store the result in register E
       MOV A,D             ;second level OR gate comparison
       RLC                 ;we bring D and E on the same position
       ORA E               ;OR between D and E
       RRC                 ;move the result to output X0
       MOV D,A             ;store the result in register D
       INR C
       JMP LOOP1

XOR0:
       MOV A,B
       ANI 10H             ;B2
       MOV H,A
       MOV A,B
       ANI 20H             ;A2
       RRC                 ;we bring A2 and B2 on the same position to compare
       XRA H
       RRC
       RRC                 ;X2 output
       MOV H,A             ;store the result in register H
       INR C
       JMP LOOP1

XOR1:
       MOV A,B
       ANI 40H             ;B3
       MOV L,A
```

```
        MOV A,B
        ANI 80H             ;A3
        RRC                 ;we bring A3 and B3 on the same position to compare
        XRA L
        RRC
        RRC
        RRC                 ;X3 output
        MOV L,A             ;store the result in register L
        MOV A,H             ;second level XOR comparison
        RLC                 ;we bring H and L on the same position
        ORA L               ;OR between H and L
        RRC                 ;move the result to output X2
        MOV H,A             ;store the result in register H

END:
        MOV A,D
        ORA E
        ORA H
        ORA L
        CMA
        STA 3000H
        JMP START
        END
```

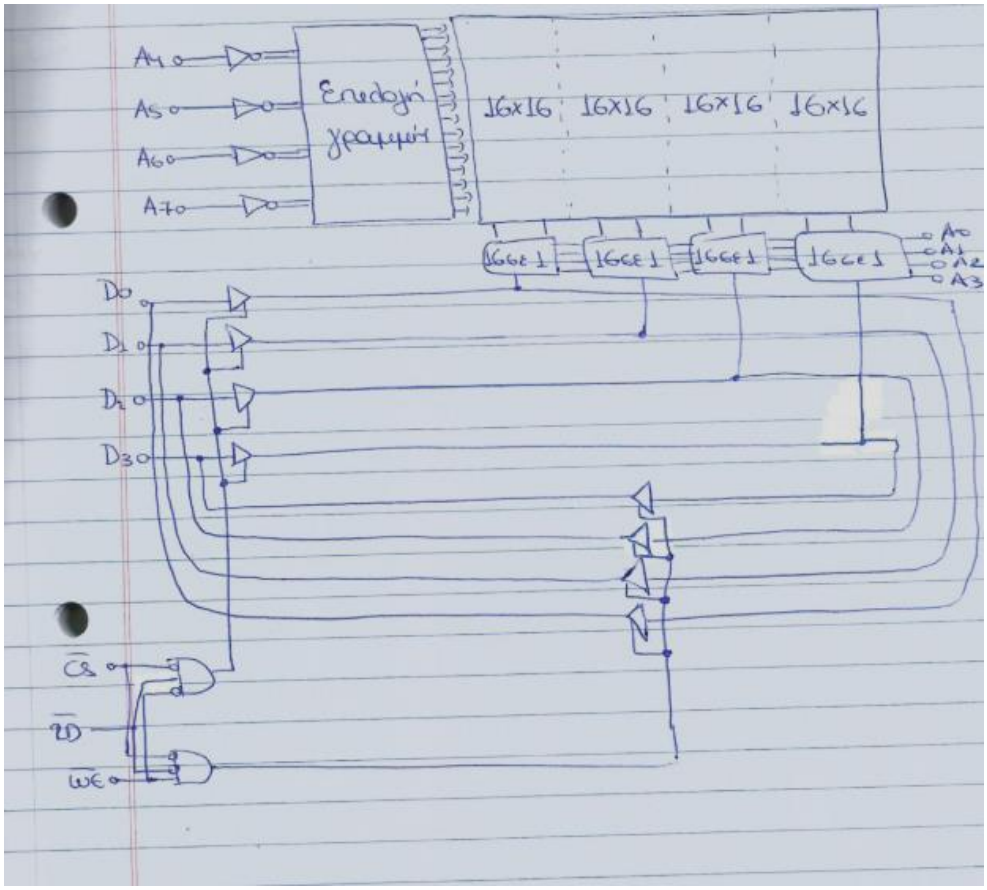


Στο παραπάνω παράδειγμα θέτουμε ως είσοδο το 01010101. Οπότε η πρώτη πύλη XOR επιστρέφει 1, η δεύτερη πύλη XOR επιστρέφει 1 και οι 2 τελευταίες AND επιστρέφουν 0. Επομένως, οι τιμές που πρέπει να λάβουμε είναι 1 από την 1η XOR, 1 από το OR των 2 XOR, 0 από την 1η AND και 0 από το OR των 2 XOR. Τα πρώτα 4 ψηφία πρέπει να είναι σβηστά. Οπότε, το επιθυμητό αποτέλεσμα είναι το 00001100, το οποίο και παίρνουμε ως έξοδο.

5) Για την οργάνωση μιας 256x4 bit SRAM μνήμης, υπάρχουν πολλοί συνδυασμοί στηλών και γραμμών. Εμείς στο ακόλουθο σχήμα, επιλέγουμε τετραγωνική οργάνωση, η οποία στη συγκεκριμένη περίπτωση αντιστοιχεί σε 16 γραμμές x 16 στήλες. Η επιλογή των γραμμών γίνεται με βάση τα σήματα A4-A7, ενώ η επιλογή στηλών με βάση τα σήματα A0-A3, που λειτουργούν ως επιλογείς των πολυπλεκτών. Για παράδειγμα, αν έχω A0A1A2A3A4A5A6A7=11101101, τότε θα επιλεγεί η 14η στήλη και η 13η γραμμή.

Για να γίνει είτε ανάγνωση από την μνήμη, είτε εγγραφή στην μνήμη, πρέπει καταρχάς να είναι ενεργοποιημένο δηλαδή με τιμή 0, το σήμα CS’. Ειδικότερα, για να γίνει ανάγνωση από τη μνήμη πρέπει να είναι ενεργοποιημένο, δηλαδή με τιμή 0 το σήμα RD’ και απενεργοποιημένο, δηλαδή με τιμή 1 το σήμα WE’. Από την άλλη για να γίνει εγγραφή στη μνήμη, πρέπει το σήμα WE’ να είναι ενεργοποιημένο, δηλαδή με τιμή 0 και το σήμα RD’ απενεργοποιημένο, δηλαδή με τιμή 1.

Το κύκλωμα είναι το ακόλουθο:



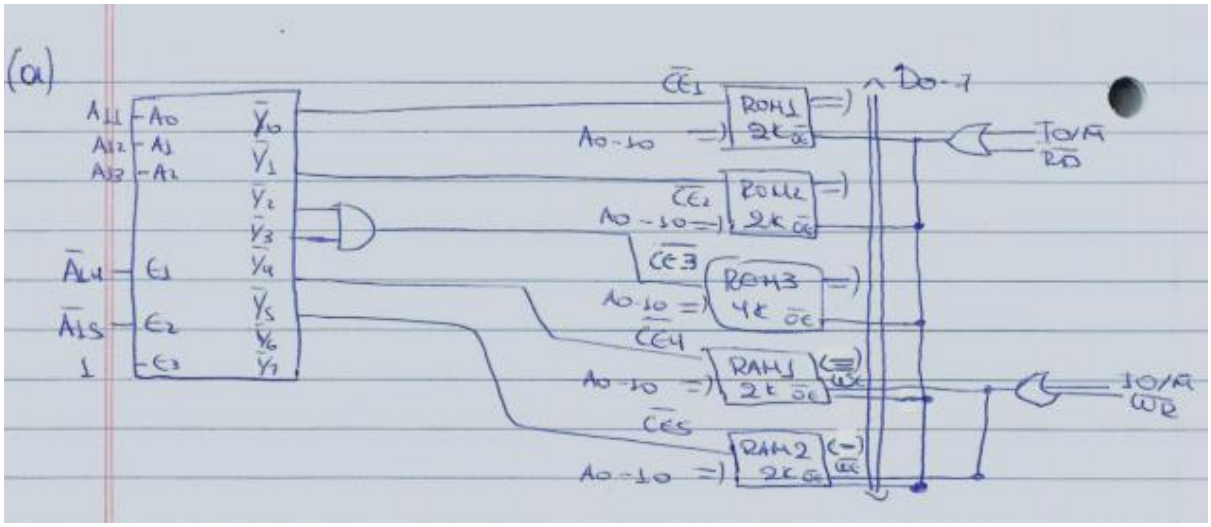

6) Ο ζητούμενος χάρτης μνήμης είναι ο ακόλουθος:

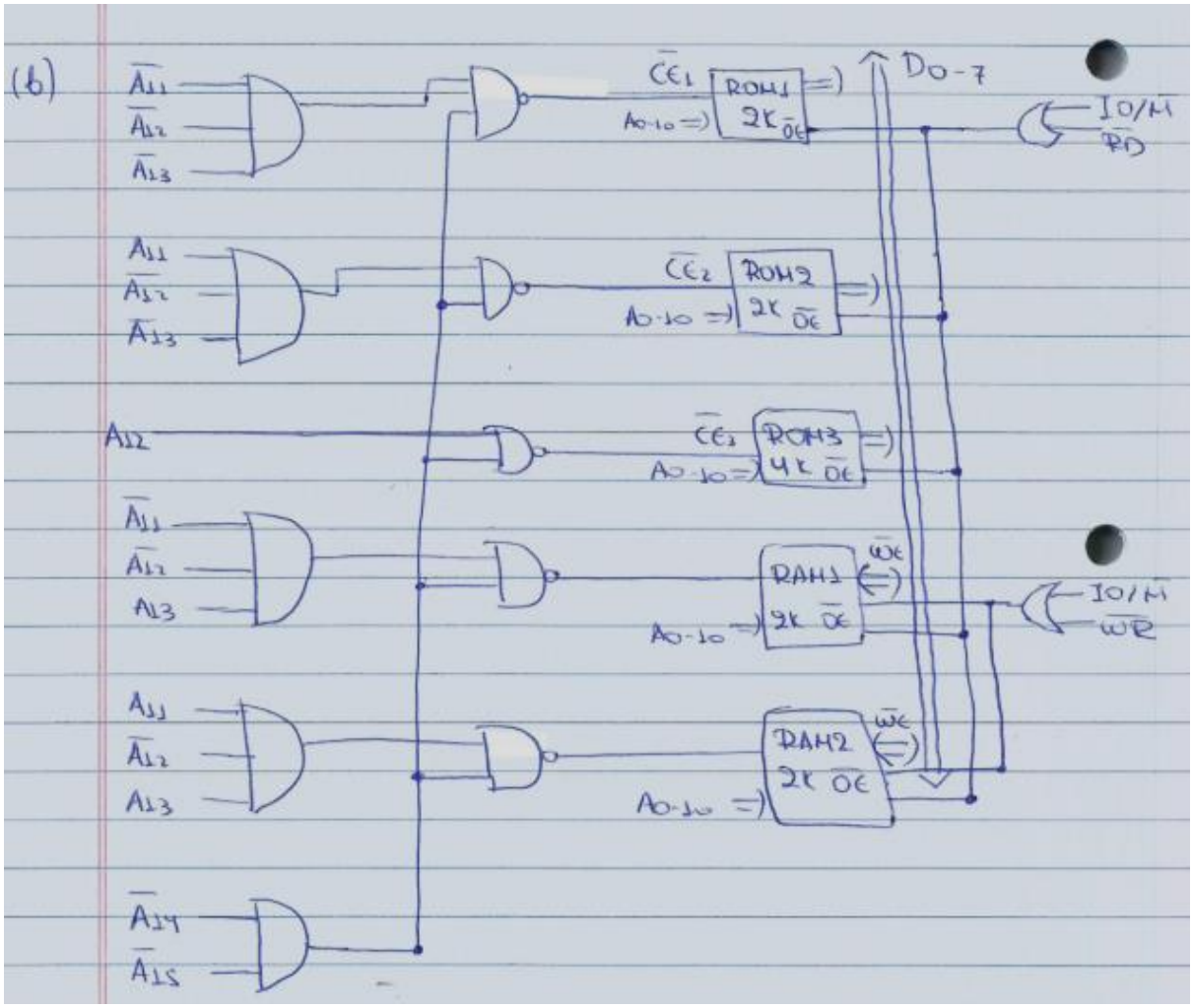
| Μνήμη | Διεύθυνση | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ROM1 | 0000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2K | 07FF | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ROM2 | 0800 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2K | 0FFF | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ROM3 | 1000 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4K | 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 |
| 2K | 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 |
| 2K | 2FFF | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Παρατηρώ ότι τα ψηφία 15,14 είναι 0 για όλες τις μνήμες, οπότε δεν θα καθορίσουν την επιλογή μνημών. Θα χρησιμοποιήσω επομένως αποκωδικοποιητή, ο οποίος έχει ως εισόδους τα ψηφία A13, A12, A11, τα οποία κυρίως επηρεάζουν την απόφασή μας για τη μνήμη και ως enable τα A14',A15'. Με βάση τον χάρτη μνήμης προκύπτει ότι για το (α) ερώτημα $CE_{ROM1}$=Y0, $CE_{ROM2}$=Y1, $CE_{ROM3}$=Y2+Y3, $CE_{RAM1}$=Y4 και $CE_{RAM2}$=Y5. Για το (β) ερώτημα ομοίως παρατηρούμε ότι η ROM1 ενεργοποιείται για A13'A12'A11', δηλαδή για 000, οπότε χρησιμοποιώ AND με τα συμπληρώματα των A13A12A11. Για την ROM2 παρατηρούμε ότι ενεργοποιείται για A13'A12'A11, οπότε χρησιμοποιώ AND με τα συμπληρώματα των A13A12 και το A11. Η ROM3 παρατηρούμε ότι είναι η μόνη που ενεργοποιείται με A12=1, οπότε δεν επηρεάζεται από κάτι άλλο. Η RAM1 ενεργοποιείται με

A13A12’A11’, οπότε χρησιμοποιώ μία AND για το A13 και τα συμπληρώματα των A12A11. Τέλος η RAM2 ενεργοποιείται με A13A12’A11, οπότε χρησιμοποιώ AND για το A13, το A11 και το συμπλήρωμα του A12. Βέβαια, πριν το CE κάθε μνήμη έχει και μία NAND με το AND του A14’,A15’, που πρέπει να είναι πάντα στο 0. Εάν αυτά είναι διάφορα του 0, δεν επιλέγεται καμία μνήμη.

Τα 2 κυκλώματα είναι τα ακόλουθα:



7) Ο χάρτης μνήμης είναι ο εξής:

| Μνήμη | Διεύθυνση | 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 |
| | 07FF | 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 |
| | 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 |
| | 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 |
| | 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 |
| | 5FFF | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Παρατηρώ ότι το ψηφίο 15 είναι για όλες τις μνήμες 0, οπότε θα μπει σαν enable στον αποκωδικοποιητή, ενώ τα ψηφία που κυρίως επηρεάζουν την επιλογή μνημών είναι τα A14,A13,A12. Η ROM1 ενεργοποιείται για A14'A13'A12' και A14'A13'A12, οπότε συνδέεται με Y0+Y1. Η RAM1 ενεργοποιείται για A14'A13A12', οπότε συνδέεται με Y2. Η RAM2 ενεργοποιείται για A14'A13A12, οπότε συνδέεται με Y3. Η RAM3 ενεργοποιείται για A14A13'A12', οπότε συνδέεται με Y4. Τέλος, η ROM2 ενεργοποιείται με A14A13'A12 και A14A13A12', οπότε συνδέεται με Y5+Y6. Όμως, τελικά οι ROM1 και ROM2 συνδέονται σε μία ενιαία ROM, η οποία ενεργοποιείται με Y0+Y1+Y5+Y6.

Το κύκλωμα είναι το ακόλουθο: