2η Ομάδα Ασκήσεων στα Συστήματα Μικροϋπολογιστών

6ο Εξάμηνο, Ακαδημαϊκή Περίοδος 2020 – 2021

Ονοματεπώνυμο	Αριθμός Μητρώου
Μπούφαλης Οδυσσεύς – Δημήτριος	el18118
Στεφανάκης Γεώργιος	el18436

Άσκηση 1η

```
EXERCISE_1:
1.A:
            IN  10H             ; remove memory protection
            LXI H,0900H         ; initial memory index 0900H
            MVI E,00H           ; E <- 0 (number to be stored in HL)

START:
            MOV M,E             ; move E to HL memory location
            INR E               ; E <- E + 1
            INX H               ; next memory location
            MOV A,E             ; A <- E
            CPI FFH             ; if A = 255 then Z = 0
            JNZ START           ; if A != 255 then jump to START
            MOV M,E             ; store 255 to memory

1.B:
            LXI B,0000H         ; BC counts num of ones
            LXI H,0900H         ; HL points first stored number

LOOP_ONES:
            MOV D,M             ; D <- current number
            MOV A,M             ; A <- current number

            MVI E,00H           ; E <- 00H
            CALL COUNT_ONES     ; count ones in current number
            MOV A,M             ; A <- current number
            INX H               ; HL points to next memory block
            CPI FFH             ; if current number != 255 then
            JNZ LOOP_ONES       ; loop
            JMP 1.c             ; end of 1.b

COUNT_ONES:
            INR E               ; COUNT_ONES loops while 1 <= E <= 8
            MOV A,D             ; A <- shifted number
            ANI 01H             ; mask LSB
            JZ  SKIP            ; if LSB is not 1 then goto SKIP
            INX B               ; else increase BC
SKIP:
            MOV A,D
            RRC                 ; right shift number
            MOV D,A             ; and store to D
            MOV A,E
            CPI 08H             ; if E != 8 then goto COUNT_ONES
            JNZ COUNT_ONES
            RET
```

1.C:

```
            LXI H,0900H
            MVI D,00H          ; counter of numbers in [10H, 60H]
LOOP1:
            MOV A,M            ; A <- current number
            INX H              ; HL <- next memory block
            CPI 10H            ; if num < 10H then
            JC  LOOP1          ; goto LOOP1
            CPI 61H            ; if num > 61H then
            JNC EXIT           ; exit
            INR D              ; else D <- D + 1
            JMP LOOP1          ; loop

EXIT:
            RST 1
            END
```

"ask1.8085"

Άσκηση 2η

```
EXERCISE_2:
            IN  10H             ; remove memory protection
            LXI B,0064H         ; BC <- 64H (100 dec) delay constant 100 ms
            MVI D,C8H           ; D <- C8H (200 dec) for long delay 200*100 ms = 20 sec

START:
            MVI A,00H
            CMA                 ; negative logic output
            STA 3000H           ; turn off all leds

                                ; the MSB is currently OFF
OFF_TO_ON:                      ; check for ON
            CALL DELB           ; delay
            LDA 2000H           ; load input on A
            ANI 80H             ; mask MSB
            CPI 80H             ; if MSB = 1 then goto ON_TO_OFF
            JNZ OFF_TO_ON       ; else repeat

ON_TO_OFF:                      ; check for OFF
            CALL DELB           ; delay
            LDA 2000H
            ANI 80H
            CPI 00H             ; if MSB = 1 then repeat else
            JNZ ON_TO_OFF       ; the push button is ON so we light the output byte
            MVI A,FFH           ; move the number FF hex = 255 decimal to register A
            CMA
            STA 3000H           ; turn on all the leds

CONTINUE:
            DCR D
            MOV A,D
            CPI 00H
            JZ  START           ; if D == 0 then jump to START else
            CALL DELB
            LDA 2000H           ; check for OFF-->ON
            ANI 80H
            CPI 80H             ; if MSB is ON then goto WAIT_FOR_OFF
            JNZ CONTINUE        ; jump to the subroutine CONTINUE

WAIT_FOR_OFF:                   ; check for OFF-->ON
            CALL DELB
            DCR D               ; delay 20 sec while waiting for MSB to become ON
            LDA 2000H
            ANI 80H
            CPI 00H             ; if MSB = 1 then wait for OFF
            JNZ WAIT_FOR_OFF
            MVI D,C8H           ; else reset constant D <- C8H (200 dec)
            JMP CONTINUE        ; jump to CONTINUE
            END
```

“ask2.8085”

Άσκηση 3η

```
EXERCISE_3.1:
            IN  10H             ; remove memory protection
START:
            LDA 2000H           ; load input
            MVI D,00H           ; D <- 00H
            MOV E,A             ; E stores initial input
            RAR                 ; shift right until first
            JC  D_1             ; one appears in CY
            RAR                 ; we jump to label D_i
            JC  D_2             ; if we find the first bit "1"
            RAR                 ; at index i of the dip switches
            JC  D_3
            RAR
            JC  D_4
            RAR
            JC  D_5
            RAR
            JC  D_6
            RAR
            JC  D_7
            RAR
            JC  D_8

SHOW:                           ; SHOW outputs on negative
            MOV A,D             ; logic leds the content of
            CMA                 ; A
            STA 3000H
            JMP START

D_8:        MVI D,80H
            MOV A,D             ; A <- 10000000
            JMP SHOW
D_7:        MVI D,40H           ; A <- 01000000
            JMP SHOW
D_6:        MVI D,20H           ; A <- 00100000
            JMP SHOW
D_5:        MVI D,10H           ; A <- 00010000
            JMP SHOW
D_4:        MVI D,08H           ; A <- 00001000
            JMP SHOW
D_3:        MVI D,04H           ; A <- 00000100
            JMP SHOW
D_2:        MVI D,02H           ; A <- 00000010
            JMP SHOW
D_1:        MVI D,01H           ; A <- 00000001
            JMP SHOW
EXIT:
            END
```

“ask3.1.8085”

```
EXERCISE_3.2:
            IN  10H             ; remove memory protection
START:
            CALL KIND           ; KIND waits for input from keyboard
            MOV E,A             ; E <- keyboard input
            CPI 08H             ; comparing input with numbers 1-8
            JZ  D_8
            CPI 07H
            JZ  D_7
            CPI 06H
            JZ  D_6
            CPI 05H
            JZ  D_5
            CPI 04H
            JZ  D_4
            CPI 03H
            JZ  D_3
            CPI 02H
            JZ  D_2
            CPI 01H
            JZ  D_1
            MVI A,00H           ; if input is not in range 1-8 then
            CMA                 ; A <- 00H so we don't turn on any
            STA 3000H           ; leds
            JMP START

SHOW:       MOV A,D
            CMA
            STA 3000H
            JMP START

D_1:        MVI D,FFH           ; we jump to label D_i if user input
            JMP SHOW            ; is equal to i and we turn on all the
D_2:        MVI D,FEH           ; leds from index i up to 8
            JMP SHOW
D_3:        MVI D,FCH
            JMP SHOW
D_4:        MVI D,F8H
            JMP SHOW
D_5:        MVI D,F0H
            JMP SHOW
D_6:        MVI D,E0H
            JMP SHOW
D_7:        MVI D,C0H
            JMP SHOW
D_8:        MVI D,80H
            JMP SHOW
EXIT:
            END
```

"ask3.2.8085"

```
EXERCISE_3.3:
START:
            IN  10H           ; turn memory protection off
            LXI H,0810H
            MVI M,10H         ; 10H is code for ' '
            INX H
            MVI M,10H
            INX H
            MVI M,10H
            INX H
            MVI M,10H
            INX H
            MVI M,10H
            INX H
            MVI M,10H

0LINE:
            MVI A,FEH         ; select line 0 (11111110)
            STA 2800H
            LDA 1800H         ; key input
            ANI 07H           ; keep 3 LSBs
            MVI B,85H         ; FETCH PC code
            CPI 05H           ; if FETCH PC button is pressed
            JZ  PRINT         ; go to PRINT
            MVI B,86H         ; INSTR STEP code
            JZ  PRINT         ; go to OUTPUT else
            CPI 06H           ; if INSTR STEP then
            JZ  PRINT         ; go to PRINT

1LINE:
            MVI A,FDH
            STA 2800H
            LDA 1800H
            ANI 07H
            MVI B,80H         ; FETCH REG
            CPI 05H
            JZ  PRINT
            MVI B,82H         ; FETCH ADDRS
            CPI 03H
            JZ  PRINT
            MVI B,84H         ; RUN
            CPI 06H
            JZ  PRINT

2LINE:
            MVI A,FBH
            STA 2800H
            LDA 1800H
            ANI 07H
            MVI B,00H         ; 0
            CPI 06H
            JZ PRINT
            MVI B,81H         ; DECR
            CPI 03H
```

```
            JZ  PRINT
            MVI B,83H         ; STORE/INCR
            CPI 05H
            JZ  PRINT


3LINE:
            MVI A,F7H
            STA 2800H
            LDA 1800H
            ANI 07H
            MVI B,01H         ; 1
            CPI 06H
            JZ  PRINT
            MVI B,02H         ; 2
            CPI 05H
            JZ PRINT
            MVI B,03H         ; 3
            CPI 03H
            JZ  PRINT

4LINE:
            MVI A,EFH
            STA 2800H
            LDA 1800H
            ANI 07H
            MVI B,04H         ; 4
            CPI 06H
            JZ  PRINT
            MVI B,05H         ; 5
            CPI 05H
            JZ  PRINT
            MVI B,06H         ; 6
            CPI 03H
            JZ  PRINT

5LINE:
            MVI A,DFH
            STA 2800H
            LDA 1800H
            ANI 07H
            MVI B,07H         ; 7
            CPI 06H
            JZ  PRINT
            MVI B,08H         ; 8
            CPI 05H
            JZ  PRINT
            MVI B,09H         ; 9
            CPI 03H
            JZ  PRINT

6LINE:
            MVI A,BFH
            STA 2800H
```

```
            LDA 1800H
            ANI 07H
            MVI B,0AH         ; A
            CPI 06H
            JZ  PRINT
            MVI B,0BH         ; B
            CPI 05H
            JZ  PRINT
            MVI B,0CH         ; C
            CPI 03H
            JZ  PRINT

LINE_7:
            MVI A,7FH
            STA 2800H
            LDA 1800H
            ANI 07H
            MVI B,0DH         ; D
            CPI 06H
            JZ  PRINT
            MVI B,0EH         ; E
            CPI 05H
            JZ  PRINT
            MVI B,0FH         ; F
            CPI 03H
            JZ  PRINT

            JMP START

PRINT:
            LXI H,0815H       ; output addr "0810-0815", 0815: left digit of led screen
            MOV A,B           ; code of the button pressed
            CPI 80H           ; if it is less than 80H: "0123456ABCDEF" then
            JC  0TOF          ; go to 0TOF
            MVI M,08H         ; else another button is pressed, all their codes
            LXI H,0814H       ; start with 8 which is represented by 08/LOAD NEXT ADRSS
            SUI 80H           ; SUB 80, only next digit remains
            MOV M,A           ; move LSB to required address
            JMP SHOW
      0TOF: MVI M,00H
            LXI H,0814H
            MOV M,B
      SHOW: LXI D,0810H       ; load address of the message
            CALL STDM
            CALL DCD
            JMP START
            END
```

“ask3.3.8085”

Άσκηση 4η

```
EXERCISE_4:
            JMP START

; Routines that implement the logical gates
; needed for the IC (XOR, OR, AND).
; Input registers: D, E
; Output register: C
; Usage: Align bits to be compared on the same column and store
;        them in input registers D and E. Result will be stored
;        in register C on the same column as the input bits.

XOR_OP:
            MOV A,D
            XRA E
            MOV C,A
            RET

OR_OP:
            MOV A,D
            ORA E
            MOV C,A
            RET

AND_OP:
            MOV A,D
            ANA E
            MOV C,A
            RET

            ; Driver code
START:
            LDA 2000H           ; load input
            MOV H,A             ; H stores input for later use

            ; First XOR gate between A0 and B0
            ANI 01H             ; mask B0 and
            MOV D,A             ; store it in input reg D
            MOV A,H             ; retrieve user input
            ANI 02H             ; mask A0
            RRC                 ; right shift to align bits to be compared
            MOV E,A             ; store A0 in input reg E
            CALL XOR_OP
            MOV L,C             ; L <- A0 XOR B0

            ; Second XOR gate between A1 and B1
            MOV A,H
            ANI 04H             ; mask third input bit, that is B1
            MOV D,A             ; D <- B1
            MOV A,H
            ANI 08H             ; mask fourth input bit, that is A1
            RRC                 ; bit alignment
            MOV E,A             ; E <- A1
            CALL XOR_OP
            MOV B,C             ; B <- A1 XOR B1
```

```
; Third XOR gate between (A0 XOR B0) and (A1 XOR B1)
MOV D,L           ; store A0 XOR 01 in input reg D
MOV A,B
RRC               ; A1 XOR B1 has to be shifted right twice
RRC               ; so that the output bit aligns with LSB
MOV E,A           ; store A1 XOR B1 in input reg E
CALL XOR_OP
MOV L,C           ; L <- (A0 XOR B0) XOR (A1 XOR B1) in LSB
MOV A,B           ; A1 XOR B1 also has to be given as output
RRC               ; of the second led
ADD L             ; so we add it to the final reg
MOV L,A           ; L <- result of first 2 leds

; First AND gate between A3 and B3
MOV A,H
ANI 40H           ; mask to retrieve bit B3
MOV D,A           ; store B3 in input reg D
MOV A,H
ANI 80H           ; mask to retrieve A3
RRC               ; right shift to align it with B3
MOV E,A           ; store A3 in input reg E
CALL AND_OP
MOV B,C           ; B <- A3 AND B3
MOV A,B           ; right shift result three times
RRC               ; to align it with output X3
RRC
RRC
ADD L             ; store result in output reg
MOV L,A

; Second AND gate between A2 and B2
MOV A,H
ANI 10H           ; mask to retrieve B2
MOV D,A           ; store B2 in input reg D
MOV A,H
ANI 20H           ; mask to retrieve A2
RRC               ; align A2 with B2
MOV E,A
CALL AND_OP
MOV H,C           ; H <- A2 AND B2 (running out of regs...)

; OR gate between (A3 AND B3) and (A2 AND B2)
MOV A,B
RRC               ; right shift twice A3 AND B3 to
RRC               ; align it with A2 AND B2
MOV D,A           ; store A3 AND B3 in input reg D
MOV E,H           ; store A2 AND B2 in input reg E
CALL OR_OP
MOV H,C           ; H <- (A3 AND B3) OR (A2 AND B2)
MOV A,H
RRC               ; right shift result twice so that it
RRC               ; aligns with X2
ADD L             ; store result in output reg L
```

```
; Output the result on negative logic leds
MOV L,A
CMA
STA 3000H
JMP START
END
```

"ask4.8085"

Όλα τα αρχεία πηγαίου κώδικα που παρουσιάστηκαν παραπάνω έχουν επισυναπτεί μαζί με την παρούσα αναφορά και μπορούν να εκτελεστούν στο σύστημα TSIK MicroLab Simulator .

Άσκηση 5η

Η μνήμη SRAM 256x4 bits που μας ζητείται να υλοποιήσουμε έχει χώρο για 256 λέξεις μεγέθους 4 bits. Χωρίζουμε λοιπόν την μνήμη σε 4 τμήματα όπου το καθένα έχει μέγεθος 256 bits. Κάθε λέξη μοιράζει από ένα bit της σε κάθε τμήμα από τα τέσσερα. Το κάθε τμήμα είναι ένας πίνακας με διαστάσεις α και β τέτοιες ώστε α · β = 256. Επιλέγουμε διαστάσεις 32x8 έτσι ώστε να πετύχουμε τετραγωνικό σχήμα. Για να επιλέξουμε γραμμή χρειαζόμαστε $\log_2 32 = 5$ bits και χρειαζόμαστε και άλλα $\log_2 8 = 3$ bits για να επιλέξουμε στήλη. Για αυτούς τους λόγους οι ακροδέκτες διευθύνσεων A0-A2 χρησιμοποιούνται για την επιλογή στήλης με χρήση πολυπλεκτών 8-σε-1 και οι ακροδέκτες διευθύνσεων A3-A7 χρησιμοποιούνται για την επιλογή γραμμής με χρήση αποκωδικοποιητή 5-σε-32. Οι ακροδέκτες D0-D3 είναι οι ακροδέκτες εισόδου/εξόδου των 4 bit και τα σήματα $\overline{\mathrm{WE}}$, $\overline{\mathrm{CS}}$, και $\overline{\mathrm{RD}}$ μέσω των αντίστοιχων ακροδεκτών επιτρέπουν ή αποτρέπουν λειτουργίες εγγραφή και ανάγνωσης. Τα τρία σήματα ελέγχου έχουν αρνητική πολικότητα: ενεργοποιούν τη λειτουργία τους, το καθένα, όταν είναι μηδέν. Το σήμα $\overline{\mathrm{CS}}$ (αρνητικό ChipSelect) ενεργοποιεί ή αδρανοποιεί ολόκληρο το chip και προορίζεται για χρήση όταν φτιάχνουμε μια μεγάλη μνήμη από πολλά chips, για να επιλέγουμε σε ποιο chip απευθυνόμαστε κάθε φορά. Όταν $\overline{\mathrm{CS}} = 0$ (ενεργό chip), το σήμα $\overline{\mathrm{WE}}$ (αρνητικό WriteEnable) ενεργοποιεί την εγγραφή ενώ το σήμα $\overline{\mathrm{RD}}$ ενεργοποιεί την ανάγνωση.

Για να γίνει η εγγραφή στη μνήμη:

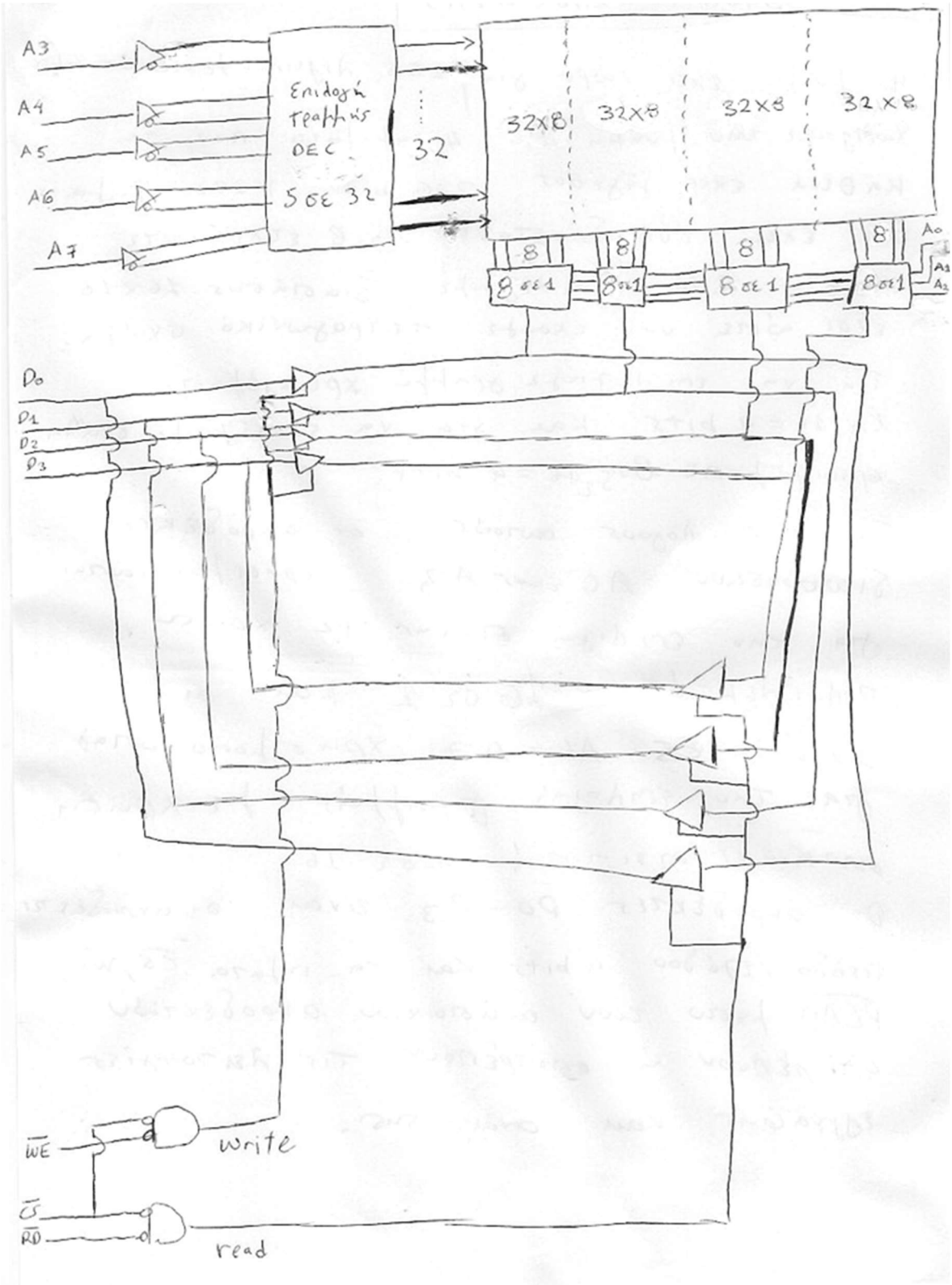
1. Εφαρμόζουμε στις γραμμές A_0-A_7 τη διεύθυνση που θέλουμε να γράψουμε.
2. Η $\overline{\mathrm{CS}}$ τίθεται στο λογικό 0 μέσω αρνητικού παλμού και σταματάει η απομόνωση εισόδου και εξόδου της μνήμης.
3. Έρχεται αρνητικός παλμός στον $\overline{\mathrm{WE}}$ ενώ στον ακροδέκτη $\overline{\mathrm{RD}}$ έρχεται θετικός παλμός και ξεκινάει η εγγραφή στη μνήμη αφού η έξοδος της πύλης AND (write) γίνεται 1 και της AND (read) γίνεται 0. Οι A_0-A_2 προσδιορίζουν σε ποια στήλη από τις 8 να οδηγήσουν την είσοδο και οι A_3-A_7 σε ποια γραμμή θα γίνει η εγγραφή των 4 bits. Στη συνέχεια το σύρμα write, οδηγείται στο enable των τρισταθών buffers που κοιτούν προς τα δεξιά και ελέγχουν το πέρασμα πληροφορίας από τις εισόδους D_0, D_1, D_2, D_3 οι οποίοι ενεργοποιούνται. Έτσι η πληροφορία τους περνάει στην έξοδο των τρισταθών buffers και ανανεώνει το περιεχόμενο της κατάλληλης θέσης μνήμης η οποία έχει επιλεχθεί όπως περιγράφηκε παραπάνω μέσω των ακροδεκτών A_0-A_7. Οι τρισταθείς buffers που κοιτούν προς τα αριστερά τίθενται σε κατάσταση υψήλης αντίστασης.
4. Τελειώνει ο αρνητικός παλμός $\overline{\mathrm{WE}}$,
5. Επανέρχεται το $\overline{\mathrm{CS}}$ στο λογικό 1.

Για να γίνει η ανάγνωση από τη μνήμη:

1. Εφαρμόζουμε στις γραμμές A0-A7 τη διεύθυνση που θέλουμε να διαβάσουμε.
2. Η $\overline{\mathrm{CS}}$ τίθεται στο λογικό 0 μέσω αρνητικού παλμού και σταματάει η απομόνωση εισόδου και εξόδου της μνήμης.
3. Έρχεται αρνητικός παλμός στον $\overline{\mathrm{RD}}$ ενώ στον ακροδέτη $\overline{\mathrm{WE}}$ έρχεται θετικός παλμός και ξεκινάει η εγγραφή στη μνήμη αφού η έξοδος της πύλης AND (read) γίνεται 1 και της AND (write) γίνεται 0. Οι ακροδέκτες A3-A7 καθορίζουν ποια γραμμή θα επιλεχθεί για την ανάγνωση των 4 bits ενώ οι ακροδέκτες A0-A2 επιτρέπουν στα bit της επιλεγμένης στήλης να φτάσουν στις εξόδους των πολυπλεκτών. Στη συνέχεια το σύρμα read, οδηγείται στο

enable των τρισταθών buffers που κοιτούν προς τα αριστερά και έτσι επιτρέπεται το πέρασμα πληροφορίας μέσα από τους D0, D1, D2, D3. Έτσι η πληροφορία τους περνάει στην έξοδο των τρισταθών buffers και έτσι διαβάζουμε την επιθυμητή διεύθυνση. Οι τρισταθείς buffers που κοιτούν προς τα δεξιά τίθενται σε κατάσταση υψήλης αντίστασης.

4. Τελειώνει ο αρνητικός παλμός $\overline{\mathrm{RD}}$.
5. Επανέρχεται το $\overline{\mathrm{CS}}$ στο λογικό 1.



Άσκηση 6η

Το ζητούμενο σύστημα μνήμης προς σχεδίαση συνίσταται από τα εξής ολοκληρωμένα που βρίσκονται σε διαδοχικές θέσεις χωρίς κενά:

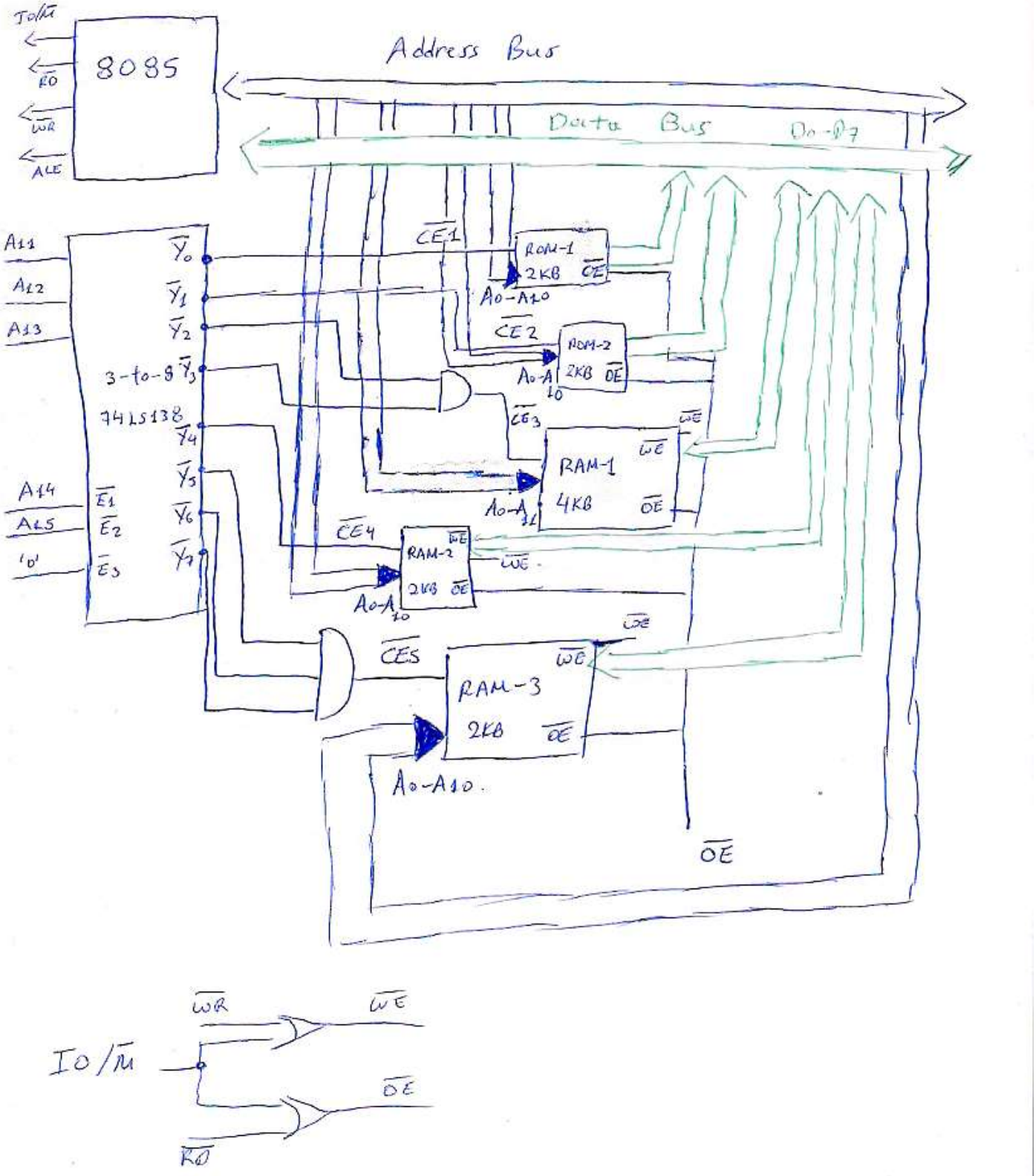
- **0000H-07FFH**: 1 ολοκληρωμένο μνήμης ROM 2K X 8bit
- **0800H-07FFH:** 1 ολοκληρωμένο μνήμης ROM 2K X 8bit
- **1000H-1FFFH:** 1 4K X 8 bit SRAM
- **2000H-27FFH:** 1 2K X 8 bit SRAM
- **2800H-2FFFH:** 1 2K X 8 bit SRAM

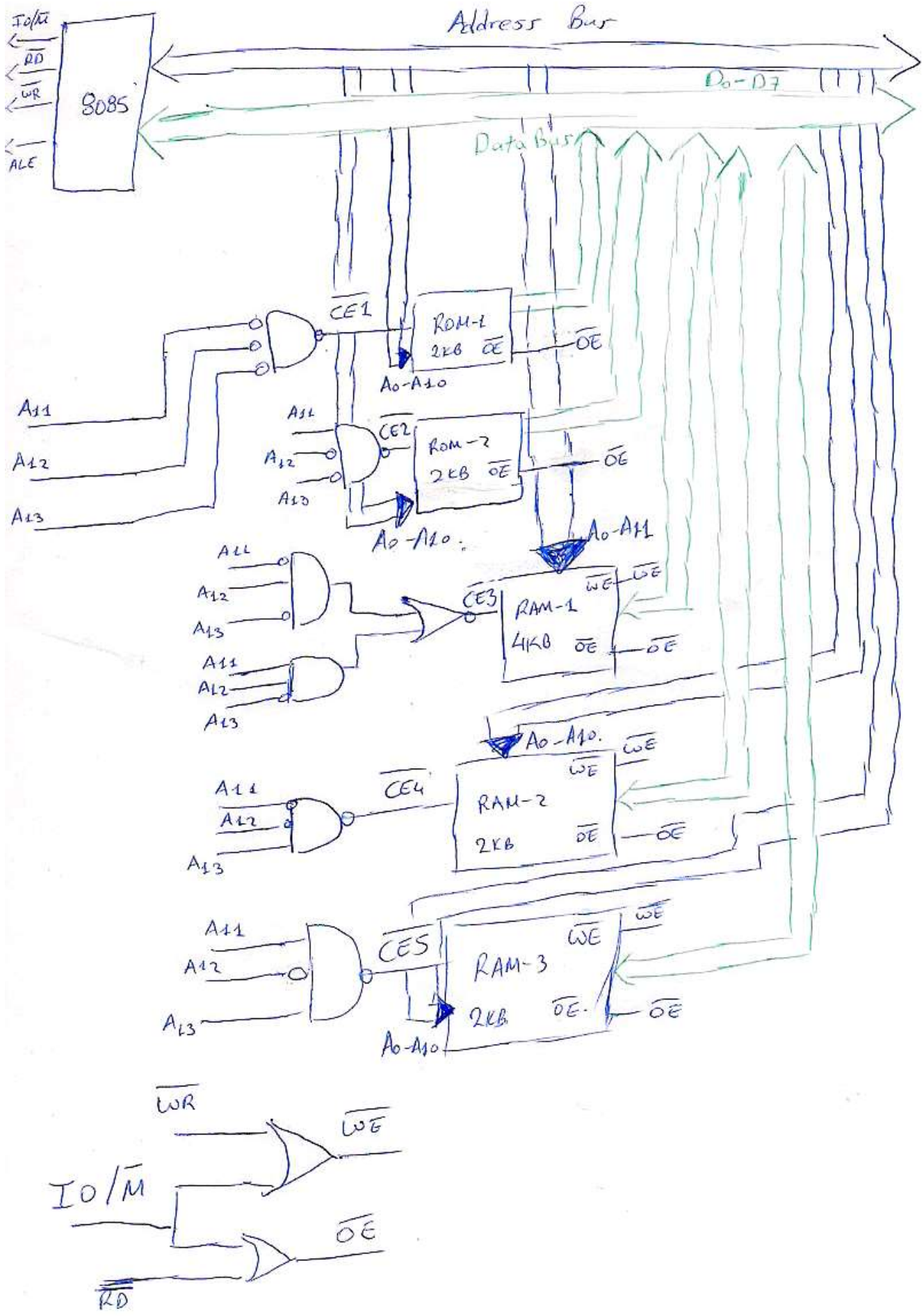
Με χρήση του μΕ 8085 η αναπαράσταση των δεδομένων απαιτεί 8 bits και οι διευθύνσεις αποτελούνται από το πολύ 16 bits.

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

Παρατηρούμε ότι τα bit A_{15}, A_{14} δεν χρησιμοποιούνται για τον προσδιορισμό καμίας θέσης μνήμης και έχουν για όλες τις θέσεις μνήμης τιμή 00. Χρησιμοποιούνται λοιπόν ως επίτρεψη στον αποκωδικοποιητή ή στις λογικές πύλες της 2ης υλοποίησης. Τα bit A_{11}, A_{12}, A_{13} χρησιμοποιούνται για την επιλογή του επιθυμητού ολοκληρωμένου. Συγκεκριμένα:

- $A_{13}, A_{12}, A_{11} = 000 \rightarrow \text{ROM1}$
- $A_{13}, A_{12}, A_{11} = 001 \rightarrow \text{ROM2}$
- $A_{13}, A_{12}, A_{11} = 010,\ A_{13}, A_{12}, A_{11} = 011 \rightarrow \text{ROM3}$
- $A_{13}, A_{12}, A_{11} = 100 \rightarrow \text{RAM1}$
- $A_{13}, A_{12}, A_{11} = 101 \rightarrow \text{RAM2}$


8085
Address Bus
Data Bus D0-D7
RD
WR
ALE
A11
A12
A13
A14
A15
'0'
3-to-8
74LS138
RAM-1
2KB
4KB
RAM-2
RAM-3
A0-A10
A0-A11
CE1
CE2
CE3
CE4
CE5
WE
OE
IO/M


IO/M
RD
WR
ALE
8085
Address Bus
D0-D7
Data Bus
CE1
ROM-1
2KB
OE
A0-A10
A11
A12
A13
CE2
ROM-2
2KB
OE
A0-A10
A0-A11
CE3
RAM-1
4KB
WE
OE
A0-A10
CE4
RAM-2
2KB
WE
OE
CE5
RAM-3
2KB
WE
OE
A0-A10
WR
WE
IO/M
OE
RD

Άσκηση 7η

- **0000H-2FFFH:** ROM (12KB)
- **3000H-5FFFH:** RAM (12KB)
- **6000H-6FFFH:** ROM (4KB)
- **7000H:** θύρα εξόδου (Memory map I/O)
- **70H:** θύρα εισόδου (Standard I/O)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Address
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	2FFFH
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	3000H
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	4FFFH
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	5000H
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFFH
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000H
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	6FFFH

Από το παραπάνω πίνακα παρατηρούμε ότι η διαφοροποίηση των θέσεων μνήμης που καταλαμβάνουν οι ROM, RAM γίνεται μέσω των θέσεων 12, 13, 14. Οπότε αυτά τα bits χρησιμοποιήθηκαν σαν είσοδοι προκειμένου να επιλεχθεί το κατάλληλο ολοκληρωμένο μνήμης. Επιπλέον από τις προδιαγραφές του προβλήματος ορίζεται η κατάληψη των θέσεων μνήμης με αλληλουχία ROM, RAM, ROM. Δηλαδή, η ενιαία μνήμη ROM που διαθέτουμε που έχει μέγεθος 16KB ίσο με το άθροισμα των μεγεθών των επιμέρους 4KB και οι 3 RAM των 4KB με συνολικό μέγεθος 12KB θα πρέπει να παρεμβάλλονται. Οπότε το πρόβλημα ανάγεται στο πως θα μπορέσουμε να διαχωρίσουμε τις διευθύνσεις που αντιστοιχούν σε κάθε κομμάτι της ROM. Αναλυτικά έχουμε:

- Οι λέξεις που αποθηκεύονται στην ROM είναι σε πλήθος 16KB = 2^{14} και συνεπώς χρειάζονται 14 bit ($A_0 - A_{13}$) για την διευθυνσιοδότησή τους
- Οι λέξεις που αποθηκεύονται στις RAM 1, RAM 2, RAM 3 μεγέθους 4KB είναι σε πλήθος 4K =2^{12} λέξεις και συνεπώς απαιτούνται 12 bit ($A_0 - A_{11}$) για την διευθυνσιοδότησή τους.

Τα bit A_{12}, A_{13}, A_{14} χρησιμοποιούνται για την επιλογή του επιθυμητού ολοκληρωμένου (ROM, RAM1, RAM2 ή RAM3) καθώς ένας ή περισσότεροι συνδυασμοί αυτών προσδιορίζουν μοναδικά τις περιοχές μνήμης που αντιστοιχούν σε κάθε ολοκληρωμένο. Συγκεκριμένα:

- $A_{14}A_{13}A_{12} = 000$ και $A_{14}A_{13}A_{12} = 001$ και $A_{14}A_{13}A_{12} = 010$ και $A_{14}A_{13}A_{12} = 110$ για τη ROM
- $A_{14}A_{13}A_{12} = 011$ για την RAM 1
- $A_{14}A_{13}A_{12} = 100$ για την RAM 2
- $A_{14}A_{13}A_{12} = 101$ για την RAM 3

Η μνήμη ROM λαμβάνει τα bits $A_0 - A_{11}$ από το address bus ενώ το bit A_{12} από την έξοδο της πύλης XOR και το A_{13} αυτούσιο όπως φαίνεται στο παρακάτω σχήμα. Αυτό συμβαίνει έτσι 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-2FFFH) σε συνεχόμενες θέσεις εσωτερικά στο ολοκληρωμένο (γίνεται δηλαδή αντιστοίχιση των διευθύνσεων 6000H έως 6FFFH του χάρτη μνήμης στις διευθύνσεις 3000H-3FFFH της ROM). Τέλος, για την επίτρεψη του latch της θύρας

εξόδου 7000H κάνουμε χρήση πύλης AND με 16 εισόδους (γιατί είναι memory map I/O), η οποία αντιστοιχεί σε $(A_0'A_1'A_2'A_3'A_4'A_5'A_6'A_7'A_8'A_9'A_{10}'A_{11}'A_{12}A_{13}A_{14}A_{15}')'$ ενώ για την επίτρεψη του latch εισόδου κάνουμε χρήση του $\overline{Y_7}$ (αφού είναι από το standard I/O) που δεν χρησιμοποιείται κάπου αλλού.