Ομάδα: Χρίστος Χατζηχριστοφή(03117711)

Δημήτρης Λάμπρος(03117070)

Μάθημα: Συστήματα Μικροϋπολογιστών

Σχολή – Εξάμηνο: ΗΜΜΥ 6ο

Δεύτερο σετ ασκήσεων

Άσκηση 1:

a) Επιλέξαμε καθυστέρηση 3sec ώστε να εμφανίζεται καλύτερα το αποτέλεσμα.

```
ARXH:  IN 10H
       MVI A,FFH      ; FFH=255 decimal
       LXI D,0900H    ; DE pair has memory location 0900H
       MVI H,00H      ; H is a counter
LOOPA: STAX D         ; store accumulator to register pair DE
       DCR A          ; A--;
       INX D          ; Move one memory space down
       CPI 00H
       JNZ LOOPA
       LXI D,0900H    ; DE pair has memory location 0900H
PRINT: LDAX D         ; load contents of DE mem loc to A
       LXI B,0BB8H    ; 3 sec delay
       CALL DELB      ; to see result
       CMA
       STA 3000H      ; print result to LEDs
       INX D          ; Move one memory space down
       INR H          ; H++
       MOV A,H
       CPI 00H        ;compare H to 0 (256)
       JNZ PRINT      ;IF H>0 PRINT next
END
```

b) Στην άσκηση αυτή για διασταύρωση του αποτελέσματος έγινε το πρόγραμμα σε c++. Έτσι ελέξαμε την ορθότητα του.

```
ARXH:       IN 10H
            MVI A,FFH     ; FFH=255 decimal
            LXI B,0900H   ; BC pair has memory location 0900H
            LXI D,0000H   ; INIT DE PAIR
            MVI L,00H     ;L IS A COUNTER

LOOPA:      STAX B        ;store accumulator to register pair HL
            DCR A         ; A--;
            INX B         ;Move one memory space down
            CPI 00H
            JNZ LOOPA
            ;FINISHED INIT MATRIX STARTIN IN 0900H

            LXI B,0900H   ; HL pair has memory location 0900H
DATA:       LDAX B        ;load contents of HL mem loc to A
            MVI H,08H     ; H=8;
CNTZEROS:   RAL           ; MOVE MSB TO Carry in order to check it.
            JC NEXTBIT
            INX D
NEXTBIT:    DCR H         ; H--;
            STA 08FFH     ; store A in memory loc x08ff
            MOV A,H
            CPI 00H
            LDA 08FFH     ; Load A
            JNZ CNTZEROS

            INX B         ;Move one memory space down
            INR L         ; L++
            MOV A,L
            CPI 00H       ;compare L to 0 (256)
            JNZ DATA

END
```

```
#include <iostream>
#include <cmath>
using namespace std;

#define BITS 8

int main(){

    int counterZeros=0,n;

    for (int j = 0; j < 256; j ++){
        n = j;
        int binary[BITS];

        for (int k = 0; k<BITS; k++)
            binary[k] = 0;

        for(int i=0; n>0; i++)
        {
            binary[i] = n%2;
            n = n/2;
        }

        for(int l = 0 ; l < BITS ; l ++)
            if(binary[l] == 0) counterZeros++;

    }
    cout<< "Result is: "<<counterZeros<<endl;
}
```

c) Για το ερώτημα αυτό μπορούμε πολύ εύκολα να διασταυρώσουμε το αποτέλεσμα, καθώς μας ζητείται το σύνολο των αριθμών που είναι μεταξύ 20H και 70H.

```
ARXH:       IN 10H
            MVI A,FFH           ; FFH=255 decimal
            LXI B,0900H         ; BC pair has memory location 0900H
            LXI D,0000H         ; INIT DE PAIR
            MVI L,00H           ; L IS A COUNTER

LOOPA:      STAX B              ; store accumulator to register pair HL
            DCR A               ; A--;
            INX B               ; Move one memory space down
            CPI 00H
            JNZ LOOPA
            ;FINISHED INIT MATRIX STARTING IN 0900H

            LXI D,0900H         ; DE pair has memory location 0900H
DATA:       LDAX D              ;load contents of DE mem loc to A
            CPI 20H             ; Comparing A with 20H
            JC NEXT             ; If Carry = 1 then A < 20H
            CPI 70H             ; Comparin A with 70H
            JC IT_IS            ; If Carry = 1 then A < 70H.
MIGHTBE:    JZ IT_IS            ; If not might be equal to 70H.
            JMP NEXT            ; If Z = 1 then A = 70H. Else jump to next
IT_IS:      INR C               ; If all checks passed, then counter ++.
NEXT:       INX D               ;Move one memory space down
            INR L               ; L++
            MOV A,L
            CPI 00H             ;compare L to 0 (256)
            JNZ DATA
            MOV A,C             ; Load content of C to A.
            CMA                 ; Getting A ready for printing.
            STA 3000H           ; Printing content of A which is C.
END
```

Άσκηση 2:

```
                IN 10H
                MVI D,00H           ; Initiate D to zero.
                LXI B,0064H         ; 100ms.Initiate E to 75dec.
START:          MVI E,4BH           ; Because 75*(100ms+100ms) = 15s
                LDA 2000H           ; Reading from dip switch.
                CMP D               ; Compare A to previous value.
                MOV H,A             ; Save A to H.
                JC MUST_FLICKER     ; If C = 1 then switch went from ON --> OFF
                MOV D,A             ; Keeping previous read of LSB Dip switch
                JMP START           ; Start over, if switch didnt go from ON-->OFF

INIT_E:MVI E,4BH                    ; Initiate E to 75dec.

MUST_FLICKER:   MVI A,00H           ; Making sure A is Zero, so flicker is right.
                STA 3000H           ; print complement of A(11111111)
                CALL DELB           ; delay to see result
                CMA                 ; complement zero so in outport 00000000 is
                STA 3000H           ; shown.
                CALL DELB           ; delay to see result
                MOV A,H             ; Restore A's current value.It was saved to H
                MOV D,A             ; Keeping previous read of LSB Dip switch.
                LDA 2000H           ; Reading from port.
                MOV H,A             ; Save A to H.
                CMP D               ; Compare A to previous value.
                JC INIT_E           ; If C = 1 then switch went from ON --> OFF.
                                    ; So restarting the timer.
                DCR E               ; Decreasing the timer by one.
                MOV A,E             ; Moving E to A, to compare if E reached
                CPI 00H             ; zero. If yes then go back to start and
                JZ START            ; wait for an other input
                JMP MUST_FLICKER    ; else continue the flickering ;)
END
```

Άσκηση 3:

```
; The idea of this program is to find out the position of the first one.
; We start the counter from 8 and decreasing every time until the position
; is found. When found, we jump to FOUND label where we do exactly
; B RALS. I.E if B is equal to 0 then 0 RALS are done, if B is 7, 7 RALS
; are done. Each time a RAL is done then we decrease B by one and continue
; until B reaches zero, and we go to PRINT label where we print the
; output to the leds. Note that every time we complement carry, cause
; if not carry will keep coming from MSB to LSB.

            IN 10H
START:      MVI B,08H    ; Counter
            LDA 2000H    ; Reading from dip switches.

FIRST_ONE:  DCR B        ; Decrease counter by one
            RAL          ; Bringing MSB to Carry flag
            JC FOUND     ; First one found
            MOV L,A      ; saving A to L
            MOV A,B      ; move B to A so we make
            CPI 00H      ; a comparison if B reached 0
            JZ PRINT     ; no more digits to check
            MOV A,L      ; return the value of A
            JMP FIRST_ONE      ; continue the process.

FOUND:      MVI A,FFH    ; Filling A with ones
LOOP_A:     MOV L,A      ; saving A to L
            MOV A,B      ; move B to A so we make
            CPI 00H      ; a comparison if B reached 0
            JZ PRINT     ; If its equal to zero then print
            MOV A,L      ; else moving back A from L and cont
            STC          ; Making sure Carry is eq to 1
            CMC          ; Complementing Carry
            RAL          ; moving MSB to carry
            DCR B        ; Decreasing B each time
            JMP LOOP_A   ; jumping to loop until B == 0

PRINT:      MOV A,L      ; Moving back A from L
            CMA          ; and get the complement so as
            STA 3000H    ; we print the right value to the leds.
            JMP START    ; Jumping back to start.Wait for input
END
```

(a)

```
IN 10H
INIT:       MVI L,00H     ; L is a counter for flicker times
            LXI B,1388H   ; 5000 ms delay
            CALL KIND     ; read input from keyboard
            CPI 00H       ; IF A==0 then error-read again
            JZ INIT
            CPI 08H       ; IF A>8 then error-read again
            JZ CONTINUE   ; else continue
            JNC INIT
CONTINUE:   CPI 05H       ; (A >= 5) ? MSB : LSB
            JNC MSB
LSB:        MVI A,F0H     ; A is not(0000 1111)
            STA 3000H     ; print A in 3000H
            CALL DELB     ; delay to see result
            MVI A,FFH     ; A is not (0000 0000)
            STA 3000H     ; print A in 3000H
            CALL DELB     ; delay to see result
            INR L         ; L++
            MOV A,L       ; Move L to A so as to compare
            CPI 04H       ; compare A(which is L) with 4
            JZ INIT       ; repeat program
            JMP LSB       ; else jump to LSB, to flicker L times
MSB:        MVI A,0FH     ; A IS NOT(1111 0000)
            STA 3000H     ; print A in 3000H
            CALL DELB     ; delay to see result
            MVI A,FFH     ; A is not (0000 0000)
            STA 3000H     ; print A in 3000H
            CALL DELB     ; delay to see result
            INR L         ; L++
            MOV A,L       ; Move L to A so as to compare
            CPI 04H       ; compare A(which is L) with 4
            JZ INIT       ; repeat program
            JMP MSB       ; else jump to MSB, to flicker L times

END
```

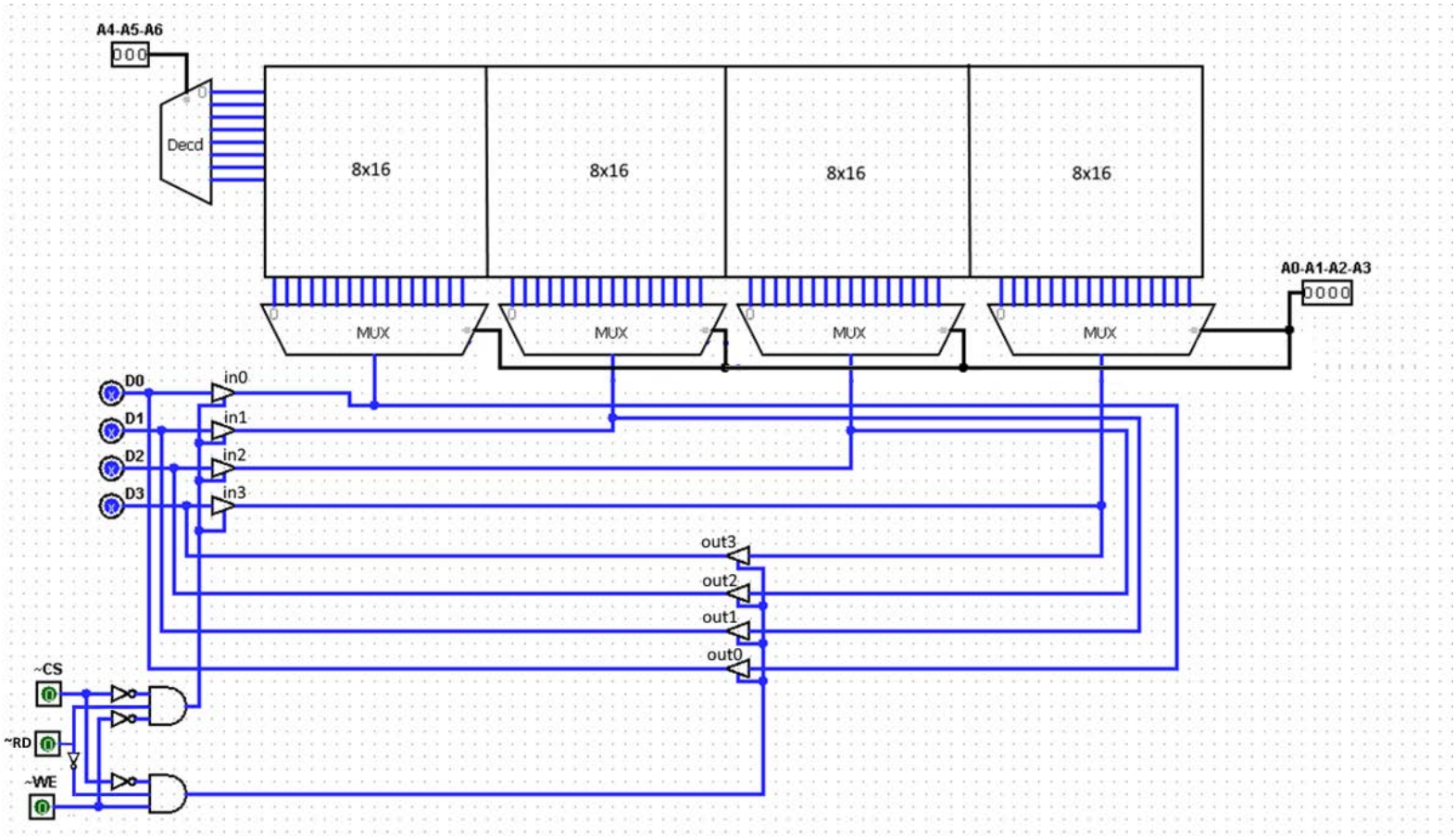
(b)

(γ) Το τρίτο ερώτημα παραβλέπεται απο την αναφορά λόγω της έκτασης του. Βρίσκεται κανονικά μέσα στον φάκελο μαζί με τα υπόλοιπα .8085 files και την αναφορά.

Άσκηση 4:

```
                    IN 10H
READ_BITS:          MVI B,01H           ; B is pointer to bits
                    LXI H,0A10H         ; HL is where results are stored
                    LDA 2000H           ; Reading switches
                    JMP FIRST_RUN ; Need this tag to avoid wrong store
TRAVERSE_BITS:      CPI 00H             ; When A is zero, all bits traversed
                    JZ  LAST_XOR        ; So we need to do the XOR
                    MOV A,C             ; Restore A's value which was stored in C
FIRST_RUN:          MOV C,A             ; Save A's value to C
                    ANA B               ; Keep Bi
                    MOV D,A             ; D has Bi
                    MOV A,B             ; Getting B to A so as to move to next bit
                    RAL                 ; which is Ai. This is done by
                    MOV B,A             ; multiplying B by 2
                    CPI 08H             ; If A is equal to 8 then AND gate
                    JZ AND_GATE         ; so we jump to AND_GATE
                    CPI 80H             ; If A is equal to 128 then AND gate
                    JZ AND_GATE         ; so we jump to AND_GATE
;-------------------------or gate-------------------------
                    MOV A,C             ; Restore A's value which was stored in C
                    ANA B               ; Keep Ai
                    RAR                 ; Moving Ai to the same position as Bi
                    ORA D               ; OR Ai,Bi
                    JMP SAVE            ; Proceed to save
;-------------------------endgate-------------------------
;-------------------------and gate-------------------------
AND_GATE:           MOV A,C             ; Restore A's value which was stored in C
                    ANA B               ; Keep Ai
                    RAR                 ; Moving Ai to the same position as Bi
                    ANA D               ; AND Ai,Bi
;-------------------------end gate-------------------------
SAVE:               MOV M,A             ; Save result in memory
                    INX H               ; Move the pointer to next pos
                    MOV A,B             ; Getting B to A so as to move to next bit
                    RAL                 ; which B(i+1). This is done by
                    MOV B,A             ; multiplying B by 2
                    JMP TRAVERSE_BITS   ; Continue traverse of bits

LAST_XOR:           LDA 0A12H           ; If we jumped here then last action
                    MOV B,A             ; must be operated. Which is XOR
                    LDA 0A13H           ; So we load the last 2 results which
                    RAR                 ; are stored in 0A13 and 0A12. Two RARS
                    RAR                 ; must be done so as digits are in same
                    XRA B               ; position.
                    STA 0A13H           ; Store the result

;------------------------------------------------------------
; B is an auxilary register so as to get the result ready.
; Loading to H the start of the "stack" that the results
; are in.Moving whatever is inside Memory to A.Adding to
; the auxilary Register (B). Move to next elem of the "stack".
; Note that every RAR is done so as the bits are shown in the
; correct position(led).
;------------------------------------------------------------
PRINT:              MVI B,00H
                    LXI H,0A10H
                    MOV A,M
                    ADD B
                    MOV B,A
                    INX H

                    MOV A.M
                    RAR
                    ADD B
                    MOV B,A
                    INX H

                    MOV A,M
                    RAR
                    RAR
                    ADD B
                    MOV B,A
                    INX H

                    MOV A,M
                    RAR
                    ADD B
                    CMA                 ; A = A' so as to print the right result
                    STA 3000H           ; Printing.
                    JMP READ_BITS       ; Moving to next input
END
```

Άσκηση 5:



Χρειαζόμαστε 4 blocks των 128 bits. Αποφασίσαμε να έχουμε τρείς επιλογείς γραμμής($A_4 – A_6$) και 4 επιλογείς για τους για τους πολυπλέκτες, αφού το κάθε data block έχει χωριστεί σε 8x16. Θα χρησιμοποιηθούν πύλες AND τριών εισόδων για τον καθορισμό της πράξης ανάγνωσης ή εγγραφής. Το ξεχωριστό σήμα ~RD πρέπει να ενωθεί με τις AND με τον εξής τρόπο: Για την AND που είναι υπεύθυνη για την εγγραφή θα ενωθεί αυτούσιο το σήμα ενώ για την AND που είναι υπεύθυνη για την ανάγνωση θα ενωθεί μέσω μιας NOT.

<u>Παράδειγμα:</u> Έστω ο αριθμός 1010. Το CS πρέπει να είναι low τόσο κατα την φόρτωση όσο και την ανάγνωση ώστε το chip να δέχεται σε κάθε περίπτωση τα input signals των επιλογών.

<u>Φόρτωση:</u> Τα δεδομένα θα φορτωθούνε στο D_0 εώς το D_3(απο LSB σε MSB). Το ~WE πρέπει να είναι low και το ~RD να είναι high. Σε αυτή την περίπτωση οι buffers in0-in3 που είναι τρισταθείς θετικής λογικής, θα μπούνε σε κατάσταση Z-high οπότε θα περάσουν τα σήματα $D_0 – D_3$ ώς είσοδοι στους MUX ώστε να καταχωρηθούν τα $D_0 – D_3$ στη μνήμη, ανάλογα πάντα με τους επιλογείς γραμμών και στηλών. Επειδή το ~WE θα είναι low και το ~RD high, οι out0 – out3 που είναι τρισταθείς θετικής λογικής, θα είναι σε κατάσταση Z-low οπότε δε θα υπάρχει βραχυκύκλωμα.

<u>Ανάγνωση:</u> Το ~WE πρέπει να είναι high και το ~RD να είναι low. Στην περίπτωση αυτή οι buffers out0 – out3 θα μπούνε σε κατάσταση Z-high αφού είναι τρισταθείς θετικής λογικής. Συνεπώς, ανάλογα με τους επιλογείς τόσο τους $A_4 – A_6$, αλλά και τους επιλογείς των MUX $A_0 – A_3$, θα περνάει στις εξόδους $D_0 – D_3$ τα επιθυμητά δεδομένα απο την μνήμη. Επειδή το ~WE θα είναι high και το ~RD low, οι buffers in0-in3 που είναι τρισταθείς θετικής λογικής, θα μπούν σε κατάσταση Z-low αφού δέχονται σήμα 0 απο την AND και συνεπώς αποφεύγεται ξανά το βραχυκύκλωμα.

Άσκηση 6:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ADDRESS	MEMORY
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	ROM_1
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	07FFH	2Kx8
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0800H	ROM_2
0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	17FFH	4Kx8
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1800H	SRAM_1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH	2Kx8
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H	SRAM_2
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH	8Kx8

Χάρτης Μνήμης:

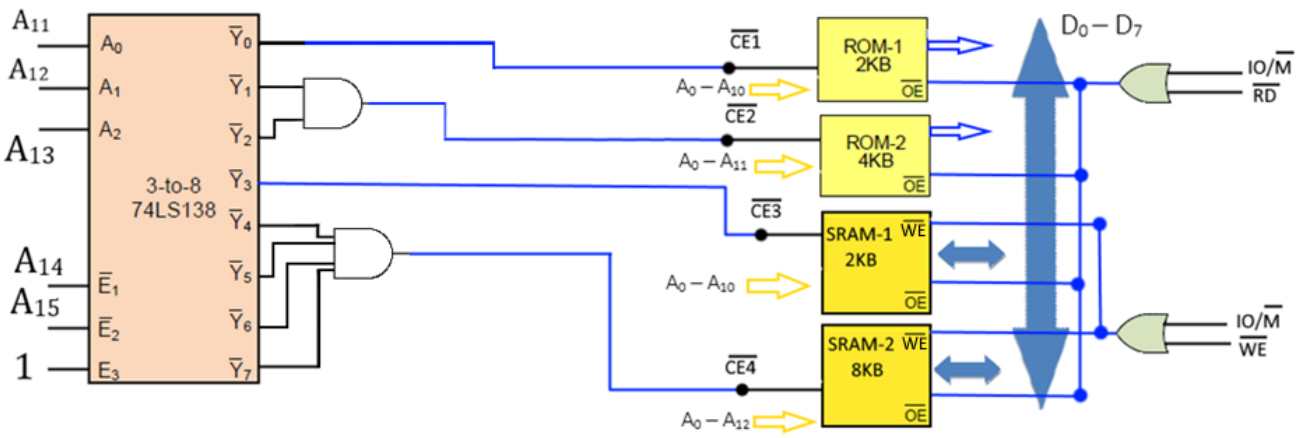
Θα χρησιμοποιήσουμε ως bits επιλογής τα $A_{11} - A_{13}$ (LSB TO MSB) και ως επιλογείς τους A_{15}, A_{14} για τους ACTIVE LOW και θα βραχυκυκλώσουμε το E3 (active high) στο 1 (VCC). Από τον χάρτη μνήμης παρατηρούμε ότι:

$$ROM_1 = 000 + 000, ROM_2 = 000 + 001, SRAM_1 = 011 + 011, SRAM_2 = 100 + 101 + 110 + 111$$

Επειδή ο 74LS138 είναι αρνητικής λογικής, τελικά:

$$\overline{CE}_1 = \bar{Y}_0, \overline{CE}_2 = \bar{Y}_1\bar{Y}_2, \overline{CE}_3 = \bar{Y}_3, \overline{CE}_4 = \bar{Y}_4\bar{Y}_5\bar{Y}_6\bar{Y}_7$$

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



Αποκωδικοποίηση με μόνο λογικές πύλες:

Κοιτάμε ποια bits θα διαφοροποιήσουν τις διευθύνσεις που χρησιμοποιεί κάθε μνήμη. Εξετάζουμε λοιπόν τον ελάχιστο αριθμό bits ο οποίος είναι κοινός για όλες τις διευθύνσεις της εκάστοτε μνήμης, προσθέτοντας bits, αν χρειάζεται, σε περίπτωση που συνειδητοποιήσουμε ότι δυο μνήμες έχουν τον ίδιο συνδυασμό bits, οπότε χρειάζονται παραπάνω bits για να διαφοροποιηθούν. Καταλήγουμε ότι χρειαζόμαστε 5 bits A_{11} - A_{15} και την επίτρεψη $IO/\overline{M}$. Τα A_{15}, A_{14} δεν προσθέτουν κάποια πληροφορία, παρόλα αυτά θα τα χρησιμοποιήσουμε ώστε να είμαστε σίγουροι οτι σε κάποια περίπτωση σφάλματος δεν θα αποκτηθεί πρόσβαση στη μνήμη.

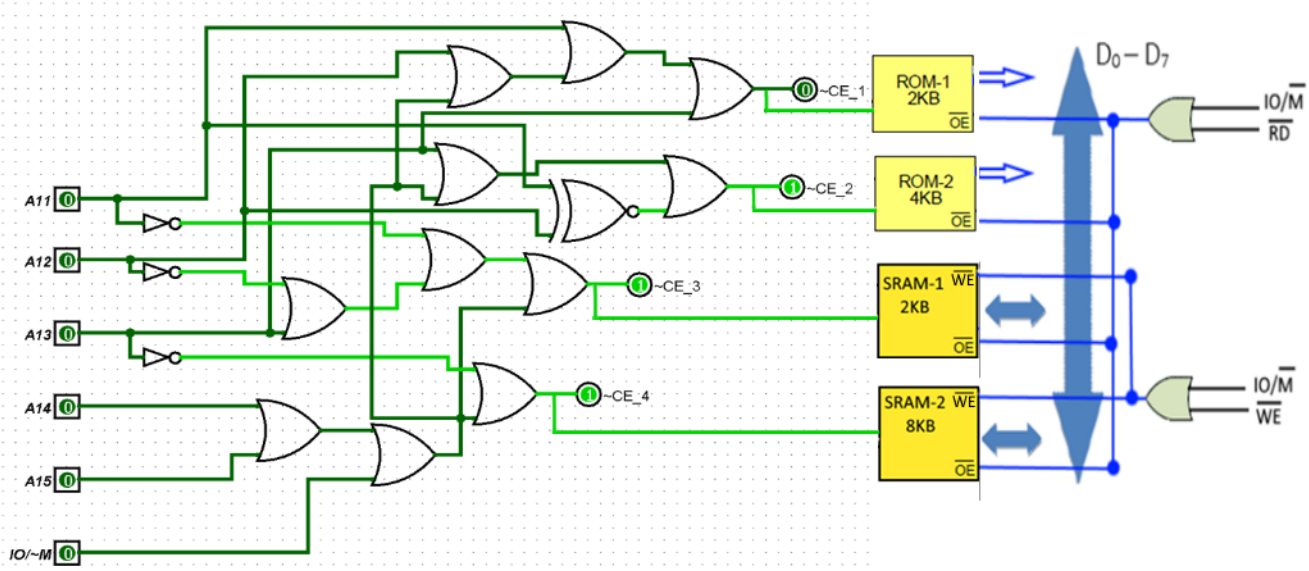
Τελικά:

$$\overline{ROM_1} = \overline{[\bar{A}_{11}\bar{A}_{12}\bar{A}_{13}\bar{A}_{14}\bar{A}_{15}\bar{\bar{M}}]} \overset{DeMorgan}{\Longrightarrow} [A_{11} + A_{12} + A_{13} + A_{14} + A_{15} + \bar{M}]$$

$$\overline{ROM_2} = [A_{13} + A_{14} + A_{15} + \bar{M} + \overline{[\bar{A}_{12}A_{11} + A_{12}\bar{A}_{11}]}]$$

$$\overline{SRAM_1} = [\bar{A}_{11} + \bar{A}_{12} + A_{13} + A_{14} + A_{15} + \bar{M}]$$

$$\overline{SRAM_2} = [\bar{A}_{13} + A_{14} + A_{15} + \bar{M}]$$



Παρατήρηση:

- Θα προσπαθήσουμε να υλοποιήσουμε τις παραπάνω συναρτήσεις με τις ίδιες πύλες, για λόγους οικονομίας. Έτσι διαλέγουμε OR 2 εισόδων , NOT, και μια XOR. Ο αποκωδικοποιητής μας σε σχέση με τον 74LS138 έχει λιγότερες πύλες NOT αλλά παραπάνω OR από τις αντίστοιχες NAND. Σαφώς, αν έχουμε στη διάθεσή μας πύλες OR ή NAND παραπάνω εισόδων μπορούμε να χρησιμοποιήσουμε μέχρι και μια πύλη με κάποιους αντιστροφείς. Η υλοποίηση εξαρτάται από τις προδιαγραφές που έχουμε.
- Στο πιο πάνω σχήμα φαίνεται πως είναι ενεργοποιημένη η ROM_1. Αυτό ήταν μια «τυχαία συγκυρία» καθώς η είσοδος $A_{11}A_{12}A_{13}A_{14}A_{15}$ = 00000 και το $\overline{M} = 0$.

Άσκηση 7:

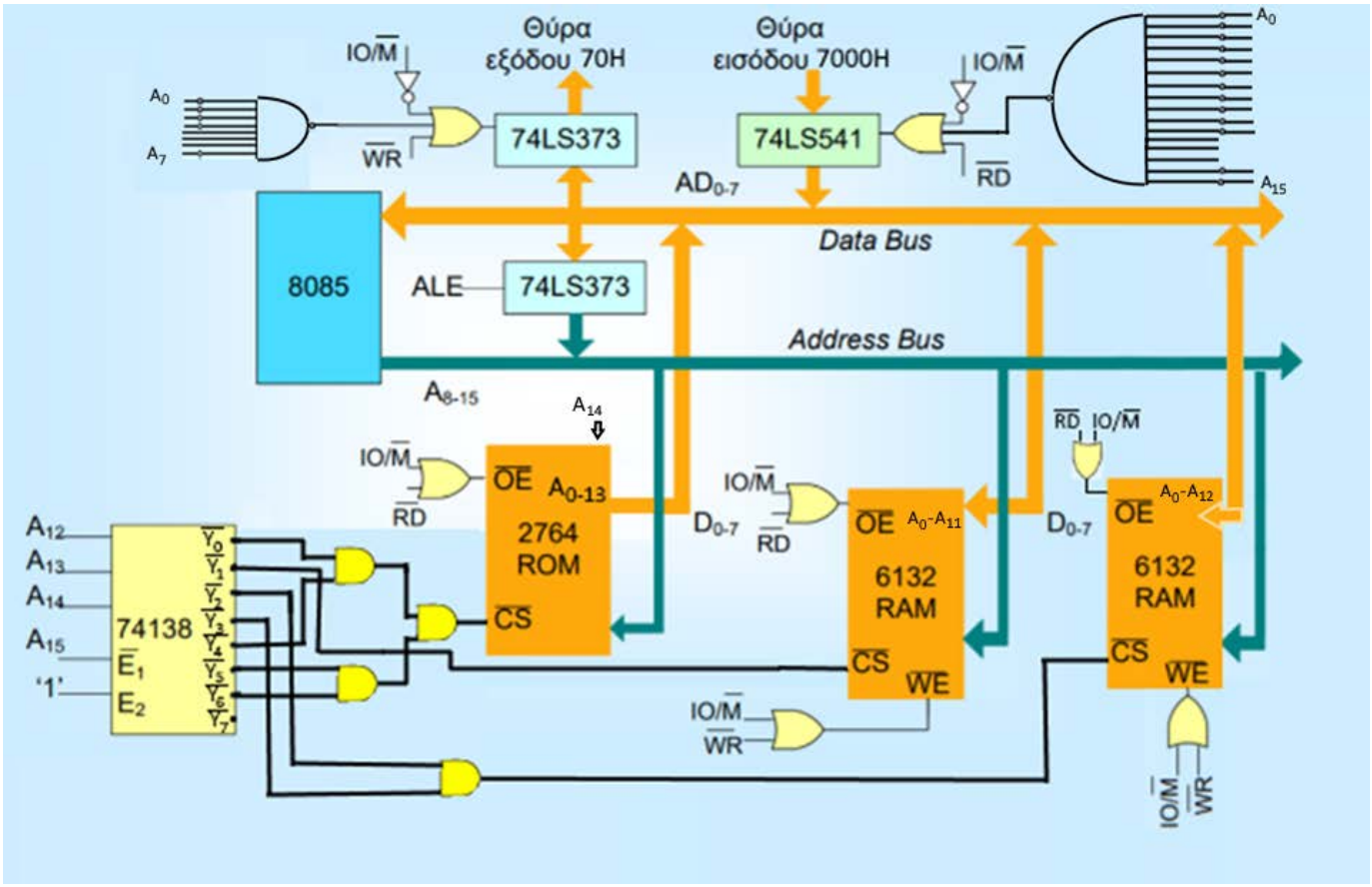
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ADDRESS	MEMORY
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	ROM_1 FIRST 4KB
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH	
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H	RAM_1 4KBx8
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H	RAM_2 8KBx8
0	0	1	1	1	1	1	1	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	ROM_1 LAST 12KB
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	6FFFH	

- Τα Chip Enables έχουν ώς εξής: $\overline{CE}_1 = \bar{Y}_0\bar{Y}_4\bar{Y}_5\bar{Y}_6, \overline{CE}_2 = \bar{Y}_1, \overline{CE}_3 = \bar{Y}_2\bar{Y}_3$

- Για το active low enable του ολοκληρωμένου των Flip Flops(74LS373 συσκευή εξόδου) θέλουμε είσοδο $\overline{IO} + NAND_1 + \overline{WR}$, όπου: $NAND_1 = \overline{\bar{A}_7 A_6 A_5 A_4 \bar{A}_3 \bar{A}_2 \bar{A}_1 \bar{A}_0}$

- Για το active low enable του 74LS541 (buffer εισόδου) θέλουμε είσοδο $\overline{IO} + NAND_2 + \overline{RD}$, όπου $NAND_2 = \overline{\bar{A}_{15} A_{14} A_{13} A_{12} \bar{A}_{11} \bar{A}_{10} \bar{A}_9 \bar{A}_8 \bar{A}_7 \bar{A}_6 \bar{A}_5 \bar{A}_4 \bar{A}_3 \bar{A}_2 \bar{A}_1 \bar{A}_0}$

- <u>Παρατήρηση:</u>
  - Αν υλοποιούσαμε την επιλογή διεύθυνσης για την περίπτωση εξόδου (STD I/O) με τα MSBs, δηλαδή, την έξοδο του DEC $\bar{Y}_7$, θα γλιτώναμε μεν μια NAND, αλλά περίπτωση που για κάποιο λόγο $\overline{RD} = 0$ και $\overline{WR} = 0$ θα ενεργοποιούνταν τόσο η θύρα εξόδου όσο και η θύρα εισόδου. Αυτό θα συνέβαινε, καθώς στην περίπτωση που θα φορτώναμε από την είσοδο 7000H, τα $A_0 – A_{15}$ = 0111 0000 0000 0000, τα $A_8 – A_{15}$ θα ήταν 0111 0000 οπότε θα ενεργοποιόταν και η έξοδος του DEC $\bar{Y}_7$.
  - Στην ROM, όπου διαχωρίζεται σε δύο μέρη, τα οποία είναι ασυνεχή στον χάρτη μνήμης παρατηρήθηκε ένα «πρόβλημα» το οποίο έχει να κάνει με το πως θα γνωρίζει η ROM σε ποιο κομμάτι της βρίσκεται. Έτσι ίσως να μπορούμε να χρησιμοποιήσουμε ένα παραπάνω Address bit, απ ότι είναι ο ελάχιστος αριθμός για να καλύψει το μέγεθος της μνήμης(A_{14} που φαίνεται στο σχήμα). Το A_{14} δε θα χρησιμοποιηθεί σαν επιλογέας στήλης ή γραμμής αλλά θα είναι ένας επιπλέον σηματοδότης ο οποίος θα διαχωρίζει το διάστημα 0000H – 0FFFH απο το 4000H – 6FFFH(μιλώντας πάντα για το εσωτερικό της ROM).


Θύρα εξόδου 70H
Θύρα εισόδου 7000H
IO/M
WR
RD
74LS373
74LS541
AD0-7
Data Bus
Address Bus
8085
ALE
74LS373
A8-15
A14
A0
A7
A15
2764 ROM
6132 RAM
6132 RAM
A0-13
A0-A11
A0-A12
D0-7
CS
OE
WE
74138
A12
A13
A14
A15
'1'
E1
E2