Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

2η Ομάδα Ασκήσεων

Μάθημα: Συστήματα Μικροϋπολογιστών

Εξάμηνο: 6ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Άσκηση 1

α) Ο ζητούμενος κώδικας είναι ο εξής (*'Άσκηση 1α.8085'*):

```
ORG 0900H    ; set the origin of the program to 0900H

    IN 10H    ; disable memory protection
    MVI A,00H ; initialize the accumulator with 0
    LXI H,0900H  ; set the memory pointer to 0900H
    MVI C,80H ; set the loop counter to 80H

LOOP1:
    MOV M,A ; store the value in the accumulator at the memory
address pointed to by H
    INR A    ; increment the value in the accumulator
    DCR C    ; decrement the loop counter
    JNZ LOOP1 ; jump to the loop label if the loop counter is not
zero

    HLT      ; halt the program

    END      ; end of the program
```

Ας εξηγήσουμε εντολή-εντολή τον κώδικα:

- **ORG 0900H:** Θέτουμε ως πηγή του προγράμματος τη θέση μνήμης 0900H, από την οποία το πρόγραμμα θα φορτώνει.
- **MVI A,00H:** Αρχικοποιούμε τον καταχωρητή στην τιμή 0.
- **LXI H,0900H:** Θέτουμε τον δείκτη μνήμης στη θέση 0900H που, όπως προαναφέρθηκε, είναι η διεύθυνση μνήμης στην οποία το πρόγραμμα θα ξεκινήσει να αποθηκεύει τους αριθμούς.
- **MVI C,80H:** Θέτουμε τον loop counter στην τιμή 128_{10} (= $80H_{16}$), δηλαδή τον αριθμό των τιμών που θέλουμε να αποθηκεύσουμε στη μνήμη.
- **LOOP:** Το πρόγραμμα μπαίνει σε ένα loop, το οποίο ακολουθεί τα επόμενα βήματα.
- **MOV M,A:** Αποθηκεύουμε την τιμή του καταχωρητή στη διεύθυνση μνήμης στην οποία δείχνει ο H.
- **INR A:** Αυξάνουμε την τιμή του καταχωρητή κατά 1.
- **DCR C:** Μειώνουμε τον loop counter κατά 1.
- **JNZ LOOP:** Ελέγχουμε αν ο loop counter ισούται με την τιμή 0. Αν όχι, τότε το πρόγραμμα μπαίνει ξανά στην ετικέτα **LOOP** και επαναλαμβάνει την ίδια διαδικασία. Αν ναι, τότε το πρόγραμμα ακολουθεί την επόμενη εντολή.
- **HLT:** Ο καταχωρητής κάνει halt.
- **END:** Τέλος προγράμματος.

β) Για το παραπάνω πρόγραμμα, καλούμαστε να υπολογίσουμε τον αριθμό των άσσων που εμφανίζονται στα δεδομένα. Για τον σκοπό αυτό κατασκευάζουμε τον παρακάτω κώδικα (*'Άσκηση 1β.8085'*):

```
ORG 0900H    ; set the origin of the program to 0900H

    IN 10H    ; disable memory protection
    LXI H,0900H  ; set the memory pointer to 0900H
    MVI B,00H ; initialize the high byte of BC with 0
    MVI C,80H ; set the loop counter to 80H

COUNT_ONES:
    MOV A,M ; move the value at the memory location pointed to by H
into the accumulator
    ANI 7FH ; clear the MSB of the accumulator (i.e., set it to 0)
    ADD B   ; add the high byte of BC to the accumulator
    MOV B,A ; move the result from the accumulator into the high byte
of BC
    XRA A   ; clear the accumulator
    MOV A,M ; move the value at the memory location pointed to by H
into the accumulator
```

```
    ANI 01H ; check the LSB of the accumulator (i.e., check if it's 1
or 0)
    RLC     ; rotate the accumulator left, shifting the LSB to the
MSB position
    JNC SKIP_INC ; jump to SKIP_INC label if the LSB is 0
    INX B   ; increment the high byte of BC if the LSB is 1
SKIP_INC:
    INR C   ; increment the loop counter
    INX H   ; increment the memory pointer
    JNZ COUNT_ONES ; jump back to COUNT_ONES label if the loop
counter is not zero

    HLT     ; halt the program

    END     ; end of the program
```

Η λειτουργία του κώδικα εξηγείται μέσα στα σχόλια. Γνωρίζουμε πως από τον αριθμό 0 έως τον αριθμό 127, ο συνολικός αριθμός των δυαδικών άσσων είναι ίσος με 896. Η ένδειξη των LED μάς δίνει τον αριθμό 0380 στο δεκαεξαδικό σύστημα, δηλαδή:

$$\left(0380_{16}\right) = \left(896_{10}\right)$$

γ) Ο κώδικας για τον υπολογισμό των αριθμών 0-127 που βρίσκονται μεταξύ και συμπεριλαμβανομένων των αριθμών 10H και 60H φαίνεται παρακάτω (*'Άσκηση 1γ.8085'*):

```
ORG 0900H    ; set the origin of the program to 0900H

    IN 10H    ; disable memory protection
    LXI H,1000H  ; set the memory pointer to 1000H
    MVI C,50H ; set the loop counter to 50H
    MVI D,00H ; initialize register D with 0

LOOP2:
    MOV A,M ; load the value from memory into the accumulator
    CPI 10H ; compare the value in the accumulator with 10H
    JC SKIP ; jump to SKIP if the value is less than 10H
    CPI 60H ; compare the value in the accumulator with 60H
    JNC SKIP ; jump to SKIP if the value is greater than 60H
    INR D ; increment register D if the value is between and
including 10H and 60H
SKIP:
    INX H    ; increment the memory pointer
    DCR C    ; decrement the loop counter
```

```
    JNZ LOOP2 ; jump to the loop label if the loop counter is not
zero

    HLT       ; halt the program

    END       ; end of the program
```

Γνωρίζουμε πως μεταξύ των δοσμένων τιμών υπάρχουν:

$$(60_{16}) - (10_{16}) = (96_{10}) - (16_{10}) = (80_{10})$$

80 αριθμοί, κάτι που επιβεβαιώνεται ελέγχοντας την τιμή του καταχωρητή D από τα LED.

Άσκηση 2

Στην δεύτερη άσκηση ζητείται να γραφεί ένα πρόγραμμα που όταν το MSB της θύρας εισόδου dip switch από OFF γίνει ON και ξανά OFF τότε να ανάβουν όλα τα LED της πόρτας εξόδου. Αν όμως ενδιάμεσα ξαναενεργοποιηθεί το push-button (OFF - ON - OFF το MSB των dip switch) να ανανεώνεται ο χρόνος.

Οπότε προκύπτει το παρακάτω πρόγραμμα (το οποίο βρίσκεται στο αρχείο askisi2.8085):

```
LXI B,0064H ;We call delb with 64Hms = 1/10 sec delay

START:
      LDA 2000H   ;Load input from dip switches to A
      RLC         ;Rotate left to check MSB
      JNC OFF     ;If MSB is off then go to OFF
      JMP START

OFF:              ;We check if the switch is turned on
      LDA 2000H
      RLC
      JC ON_FIRST ;If it's on then go to ON1
      JMP OFF     ;Else wait until it's on

ON_FIRST:               ;In order to light up the LEDs we have to switch
off
                  ;the MSB, but when that happens we'll have to call delb
      MVI D,C8H   ;200 times(we want them on for 20s), that's why D = 200
      LDA 2000H
      RLC
      JNC OPEN    ;If MSB turned off then push button(off-on-off) is
activated
      JMP ON_FIRST

OPEN:
```

```
      LDA 2000H   ;If MSB turns on the 20s timer keeps on going, but if
that
      RLC         ;happens then we have to check whether it turns off or
not.
      JC ON_AGAIN ;If it does so, then we have to reset the timer
      MVI A,00H
      STA 3000H   ;Turn on all LEDs
      CALL DELB
      DCR D       ;Decrease D
      JNZ OPEN    ;If D = 0 then 20s passed
      MVI A,FFH
      STA 3000H   ;Turn off all LEDs and start checking again
      JMP OFF

ON_AGAIN:         ;Getting here means that if the MSB switch turns off then
the
      LDA 2000H   ;timer has to reset.If the MSB stays on the whole time
then we
      RLC         ;repeat the same process as above
      JNC RESTART
      MVI A,00H
      STA 3000H
      CALL DELB
      DCR D
      JNZ ON_AGAIN
      MVI A,FFH
      STA 3000H
      JMP OFF

RESTART:          ;If on-off occured while we had the LEDs on then reset
the timer
      MVI D,C8H
      JMP OPEN

END
```

Η επεξήγηση του προγράμματος γίνεται με σχόλια πάνω στον κώδικα.

Άσκηση 3

Ζητείται να γραφούν σε assembly 8085 και να εκτελεστούν στο μLAB, 4 προγράμματα με τις εξής λειτουργίες:

1) Να διαβάζει την πόρτα εισόδου των dip switches και με βάση το 1ο δεξιότερο ON, να ανάβει το αντίστοιχης τάξης led (π.χ. για 1011 0100 => OOOO OXOO, όπου O=σβηστό led, X=αναμμένο led). Το πρόγραμμα να είναι συνεχούς λειτουργίας.

Το πρόγραμμα είναι αποθηκευμένο στο αρχείο askisi3i.8085 και η λειτουργία του εξηγείται σε σχόλια πάνω στον κώδικα.

```
START:
      LDA 2000H   ;Load input from dip switches to A
      MVI B,00H   ;Set B to 0, B is the current position being checked
      CPI 01H     ;Check if far-left dip-switch is on
      JZ TURNON   ;If it's on, turn on far-left LED

CHECK:
      RRC         ;Rotate right through carry, this will move the rightmost
bit of A into the carry flag
      DCR D       ;Decrement D, which starts at 8, to keep track of the
position being checked
      JZ TURNOFF  ;If D reaches 0, it means no dip switch is on so turn off
all LEDs and start again
      INR B       ;Increase B, which is the current position being checked
      JNC CHECK   ;If the carry flag is 0, it means the current bit being
checked is 0 so continue checking
      MVI A,FEH   ;Set A to 11111110 in binary, to turn on the LED at the
current position being checked
      DCR B       ;Decrease B to move to the correct bit position for the
LED

TURNON:
      RLC         ;Rotate left through carry, this moves the bit in the
carry flag to the leftmost position of A
      DCR B       ;Decrease B to move to the correct bit position for the
LED
      JNZ TURNON  ;If B is not 0, keep rotating left to reach the correct
bit position for the LED
      STA 3000H   ;Store the value of A to memory location 3000H to turn on
the LED at the current position
      JMP START   ;Start again from the beginning, to keep checking the dip
switches

TURNOFF:
      MVI A,FFH   ;Set A to 11111111 in binary, to turn off all LEDs
      STA 3000H   ;Store the value of A to memory location 3000H to turn
off all LEDs
      JMP START   ;Start again from the beginning, to keep checking the dip
switches

END
```

2) Να αναμένει το πάτημα του δεκαεξαδικού πληκτρολογίου και μόνο των αριθμών 1 έως 8. Κάθε φορά να ανάβει το led της αντίστοιχης θέσης και όλα τα υψηλότερης τάξης led μετά από αυτό (π.χ. για 3 => XXXX XXOO, όπου O=σβηστό led, X=αναμμένο led). Να γίνει χρήση της ρουτίνας KIND που υπάρχει στο παράρτημα 1 των σημειώσεων του μLAB. Το πρόγραμμα να είναι συνεχούς λειτουργίας.

Το πρόγραμμα είναι αποθηκευμένο στο αρχείο askisi3ii.8085 και η λειτουργία του εξηγείται σε σχόλια πάνω στον κώδικα.

```
START:
      CALL KIND   ;Wait for input from hexadecimal keyboard
      CPI 00H     ;Check if input is 1
      JC TURN_OFF       ;If input is less than 1, turn off LEDs and wait
for valid input
      CPI 09H     ;Check if input is 8
      JNC TURN_OFF      ;If input is greater than 8, turn off LEDs and wait
for valid input
      MOV B,A     ;Save input in B register
      MVI A,00H   ;Initialize A register with 0
      DCR B             ;Decrement the input
      JZ OPEN_ALL       ;If input is 1, open all LEDs and skip the loop
      INR A             ;Increment A register

REPEAT:
      DCR B             ;Decrement the input
      JZ OPEN     ;If input is 0, go to OPEN with the current value of A
      RLC         ;Rotate left through carry
      INR A             ;Increment A register
      JMP REPEAT  ;Repeat until the input is zero

OPEN:
      STA 3000H   ;Open the LEDs starting from the number that we pressed
up to the MSB
      JMP START   ;Start checking again

OPEN_ALL:
      MVI A,FFH   ;Set A register to FFH (turn on all LEDs)
      STA 3000H   ;Turn on all LEDs
      JMP START   ;Start checking again

TURN_OFF:
      MVI A,FFH   ;Set A register to FFH (turn off all LEDs)
      STA 3000H   ;Turn off all LEDs
      JMP START   ;Start checking again

END               ;End of program
```

3) Με βάση την ύλη των σελ. 76 – 79 (των σημειώσεων του μLAB) να γίνει απευθείας ανάγνωση του πληκτρολογίου χωρίς τη χρήση της ρουτίνας KIND. Το αποτέλεσμα του κωδικού (βάσει του πίνακα 1 της σελ. 74) να εμφανίζεται στα 2 αριστερότερα 7-segment display με βάση τις ρουτίνες DCD (Display Character Decoder) και STDM (Store Display Message), σελ. 80-82.

Το πρόγραμμα είναι αποθηκευμένο στο αρχείο askisi3iii.8085 και η λειτουργία του εξηγείται σε σχόλια πάνω στον κώδικα.

```
START:
      IN 10H      ; remove memory protection
      LXI H,0A00H ; load HL with the start of the memory
      MVI B,04H   ; simple repeater

LL:
      MVI M,10H   ; store 'nothing' (4 times)
      INX H
      DCR B
      JNZ LL

L0:
      MVI A,FEH   ;FEH = 11111110, means select line 0
      STA 2800H   ;store it in 2800H and declare you are processing line 0
      LDA 1800H   ;read the column
      ANI 07H     ;reset the 5 MSBs, because the last 3 have the
information of the column
      MVI C,86H   ;if A read is 00000110 then button
      CPI 06H     ;from column 1 is pressed. So go to
      JZ DISP     ;DISPLAY with C having the code for INSTR_STEP
      MVI C,85H   ;same for FETCH PC
      CPI 05H     ;but now I check if A = 00000101
      JZ DISP     ;means 2nd button is pressed

L1:
      MVI A,FDH
      STA 2800H
      LDA 1800H
      ANI 07H
      MVI C,84H
      CPI 06H     ; RUN
      JZ DISP
      MVI C,80H
      CPI 05H     ; FETCH_REG
      JZ DISP
      MVI C,82H
      CPI 03H     ; FETCH_ADDRS
      JZ DISP
```

```
L2:
      MVI A,FBH
      STA 2800H
      LDA 1800H
      ANI 07H
      MVI C,00H
      CPI 06H     ; 0
      JZ DISP
      MVI C,83H
      CPI 05H     ; STORE/INCR
      JZ DISP
      MVI C,81H
      CPI 03H     ; DECR
      JZ DISP

L3:
      MVI A,F7H
      STA 2800H
      LDA 1800H
      ANI 07H
      MVI C,01H   ; 1
      CPI 06H
      JZ DISP
      MVI C,02H   ; 2
      CPI 05H
      JZ DISP
      MVI C,03H   ; 3
      CPI 03H
      JZ DISP

L4:
      MVI A,EFH
      STA 2800H
      LDA 1800H
      ANI 07H
      MVI C,04H
      CPI 06H     ; 4
      JZ DISP
      MVI C,05H
      CPI 05H     ; 5
      JZ DISP
      MVI C,06H
      CPI 03H     ; 6
      JZ DISP

L5:
      MVI A,DFH
      STA 2800H
      LDA 1800H
```

```
      ANI 07H
      MVI C,07H
      CPI 06H     ; 7
      JZ DISP
      MVI C,08H
      CPI 05H     ; 8
      JZ DISP
      MVI C,09H
      CPI 03H     ; 9
      JZ DISP

L6:
      MVI A,BFH
      STA 2800H
      LDA 1800H
      ANI 07H
      MVI C,0AH
      CPI 06H     ; A
      JZ DISP
      MVI C,0BH
      CPI 05H     ; B
      JZ DISP
      MVI C,0CH
      CPI 03H     ; C
      JZ DISP

L7:
      MVI A,7FH
      STA 2800H
      LDA 1800H
      ANI 07H
      MVI C,0DH
      CPI 06H     ; D
      JZ DISP
      MVI C,0EH
      CPI 05H     ; E
      JZ DISP
      MVI C,0FH
      CPI 03H     ; F
      JZ DISP

      JMP START   ; if no button is pressed, check again

DISP:
      LXI H,0A04H
      MOV A,C     ;C has the button's code
      ANI 0FH     ;isolate the 4 LSBs
      MOV M,A     ;store them to 0A00H(first digit from left)
      INX H       ;HL++
```

```
      MOV A,C
      ANI F0H     ;isolate the 4 MSBs
      RLC         ;shift them to the 4 LSBs
      RLC
      RLC
      RLC
      MOV M,A     ;store them to 0A01(second digit from left)

      LXI D,0A00H ;move the block 0A00 to 0A05 to the memory

      CALL STDM   ;where the DCD routine reads

      CALL DCD    ;print


      JMP START

END
```

Άσκηση 4

Το πρόγραμμα που περιγράφει το λογικό σχήμα της εκφώνησης φαίνεται παρακάτω και είναι αποθηκευμένο στο αρχείο *'Άσκηση 4.8085'*:

```
START:
            LDA 2000H  ;Load input from dip switches to A
            MOV B,A    ;Save A to register B
      A0_B0:
            ANI 01H    ;A = A AND 00000001
            MOV C,A    ;C = B0
            MOV A,B    ;A is equal to the input of dip switches
            ANI 02H    ;A = A AND 00000010 = A0
      AND0:
            RRC        ;Rotate right to get the output at LSB
            MOV B,A
            ANA C      ;A AND C
            MOV D,A    ;Save the answer D = A0 AND B0

      A1_B1:
            MOV A,B
            ANI 02H    ;A = A AND 00000010 = B1
            MOV C,A    ;C = B1
            MOV A,B
```

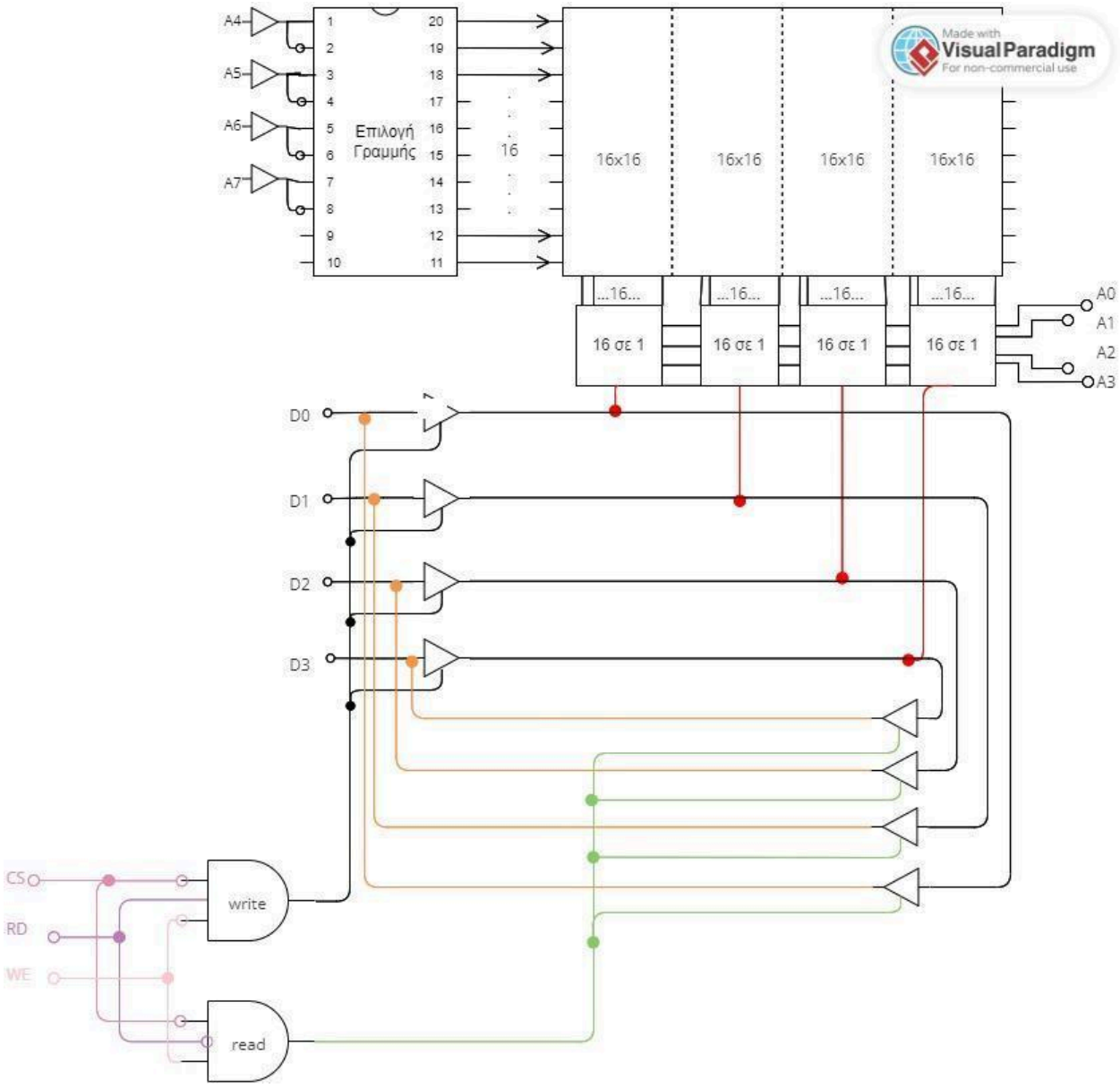
```
        ANI 04H     ;A = A AND 00000100 = A1

AND1:
        RRC
        ANA C       ;X1 = A1 AND B1
        MOV E,A     ;Save X1 at LSB

OUTPUT0:
      MOV A,D
      ORA E
      MOV D,A      ;apothikevw to xo
      MOV A,E
      RLC
      ADD D
      MOV D,A     ;apothikevw to x1xo


A2_B2:
        MOV A,B
        RRC
        RRC
        ANI 01H     ;A = A AND 00000001
        MOV C,A     ;C = B2
        MOV A,B
        ANI 02H     ;A = A AND 00000010 = A2
        RRC
XOR1:
        XRA C       ;A = A2 XOR B2
        MOV E,A     ;E = A XOR C (output of A2_B2)
A3_B3:
        MOV A,B
        RRC
        RRC
        ANI 01H     ;A = A AND 00000001
        MOV C,A     ;C = B3
        MOV A,B
        ANI 02H     ;A = A AND 00000010 = A3
        RRC
XOR2:
        XRA C       ;X3 = A XOR C (output of A3_B3)
        MOV B,A

OUTPUT2:
        ORA E       ;A = X2
        RLC
```

```
        RLC
        ADD D
        MOV D,A
        MOV A,B
        RLC
        RLC
        RLC
        ADD D

LED:
        STA 3000H  ;Turn on the correct LEDs
        JMP START  ;Start checking again

        END
```

Άσκηση 5

Στο παρακάτω σχήμα του βιβλίου φαίνεται η εσωτερική δομή μιας SRAM 256x4 bit. Θα αναλύσουμε τη σχήμα μας από τα αριστερά προς τα δεξιά. Αρχικά, εχουμε έναν πίνακα μνήμης από τον οποίο επιλέγουμε μίας από τις 16 εξόδους με βάση τις γραμμές διεύθυνσης A4-A7. Οι είσοδοι D0-D3 είναι οι γραμμές δεδομένων μας και συνδέονται με τις εξόδους των 4 πολυπλεκτών 16x1. Καθένας από τους τέσσερις πολυπλέκτες επιλέγει μίας από τις 16 στήλες του πίνακα μνήμης, βάσει των διευθύνσεων A0-A3. Σε αυτές τις γραμμές διευθύνσεων (A0-A3) είτε καταχωρούνται τα δεδομένα των γραμμών δεδομένων D0-D3, είτε το αντίστροφο.

Έτσι, αν η διεύθυνση A0A1A2A3 A4A5A6A7 ισούται με 0001 0010, τότε επιλέγεται η 1η (0001) γραμμή του πίνακα μνήμης και η 2η (0010) τετράδα του.



Αναφορικά στις εγγραφές και τις αναγνώσεις, καθεμία από αυτές καθορίζονται από τα τρια σήματα ελέγχου, CS, RD WE. Για CS = 0, ενεργοποιείται η λειτουργία της μνήμης. Για WE = 0 και RD = 1, ενεργοποιούνται οι απομονωτές της εξόδου της write και πραγματοποιείται εγγραφή στη μνήμη. Αντίθετα, για WE = 1 και RD = 0, ενεργοποιούνται οι απομονωτές της εξόδου της read (πράσινο χρώμα) και πραγματοποιείται ανάγνωση από τη μνήμη.

Άσκηση 6

Στην εκφώνηση δίνεται ως προδιαγραφή η χρήση 2 ολοκληρωμένων κυκλωμάτων μνήμης RAM μεγέθους 2K x 8 bits το καθένα και 1 ολοκληρωμένο μνήμης RAM μεγέθους 4K x 8 bits ώστε να επιτευχθεί συνολική μνήμη 8KB.

Δεδομένου ότι στο μΥ σύστημα συμμετέχει ο μΕ 8085 για την αναπαράσταση των δεδομένων απαιτούνται 8 bits. Για την αναπαράσταση των διευθύνσεων απαιτούνται το πολύ 16 bits. Η ROM έχει μέγεθος 8KB = 8K x 8 bits = 2^13 x 8 bits. Άρα για τις διευθύνσεις της εν λόγω μνήμης γίνεται χρήση 13 bits, από A0 - A12. Αντίστοιχα, για την μνήμη RAM γίνεται χρήση 12 bits, από A0 - A11 αφού το μέγεθος της ανέρχεται στα 4KB = 4K x 8 bits = 2^12 x 8 bits. Προφανώς και για τις δύο μνήμες το πλήθος των bits για την αναπαράσταση των δεδομένων είναι. Παρατηρείται οτι 3 (A13 - A15) από τα 16 bits (A0 - A15) μένουν αχρείαστα. Αυτά τα bits λοιπόν χρειάζονται στον αποκωδικοποιητή για την επιλογή του κατάλληλου ολοκληρωμένου. Ο αποκωδικοποιητής όχι μόνο επιλέγει κάθε φορά ανάμεσα στα δύο είδη μνήμης του συστήματος (ROM - RAM) αλλά επιλέγει κι ανάμεσα στα ολοκληρωμένα που αφορούν στο ίδιο είδος μνήμης. Αυτό συμβαίνει, γιατί από τις προδιαγραφές της εκφώνησης η μνήμη ROM συγκροτείται από δύο ολοκληρωμένα κυκλώματα ενώ η RAM συγκροτείται από 3 ολοκληρωμένα κυκλώματα.

Μνήμη	Διεύθυνση μνήμης	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM#1 - 2K	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	07FF	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
ROM#2 - 2K	0800	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	0FFF	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
ROM #3 - 4K	1000	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM #1 - 2K	2000	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	27FF	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1
RAM #2 - 2K	2800	0	0	1	0	1	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

α) Στους παραπάνω χάρτες μνήμης κάποια ψηφία έχουν χρωματιστεί για να αναδειχθεί ο ιδιαίτερος ρόλος τους στην αποκωδικοποίηση. Ειδικότερα, τα μπλε ψηφία A15, A14 χρησιμεύουν ως είσοδοι επίτρεψης του αποκωδικοποιητή. Τα κόκκινα ψηφία, από A11 - A13 χρειάζονται για την επιλογή της κατάλληλης μνήμης. Ας παρατηρηθεί οτι κάθε μνήμη προσδιορίζεται μοναδικά από έναν ή δύο συνδυασμούς των ψηφίων αυτών. Για παράδειγμα, ο συνδυασμός A13A12A11 = 000 καθορίζει μοναδικά τη ROM#1, ενώ ο συνδυασμός A13A12A11 = 100 ορίζει την μνήμη RAM#1. Βέβαια, τα ψηφία αυτά πρέπει με κάποιο τρόπο να αντιστοιχιστούν στις εισόδους του αποκωδικοποιητή. Αυτό σημαίνει ότι οι είσοδοι του αποκωδικοποιητή αποτελούν συναρτήσεις των κόκκινων ψηφίων. Απόρροια των παραπάνω συνιστά ο πίνακας αληθείας των συναρτήσεων αυτών:

Τα X στον πίνακα αληθείας αναφέρονται σε αδιάφορους όρους, αφού ο συνδυασμός στον οποίο αντιστοιχούν δεν έχει κάποια πρακτική σημασία. Χρειάζεται όμως προσοχή στο τι τελικά θα επιλεχθεί για τα X. Δύνανται άσχετοι συνδυασμοί των A11A12A13 με κατάλληλη τιμή του X να οδηγούν σε

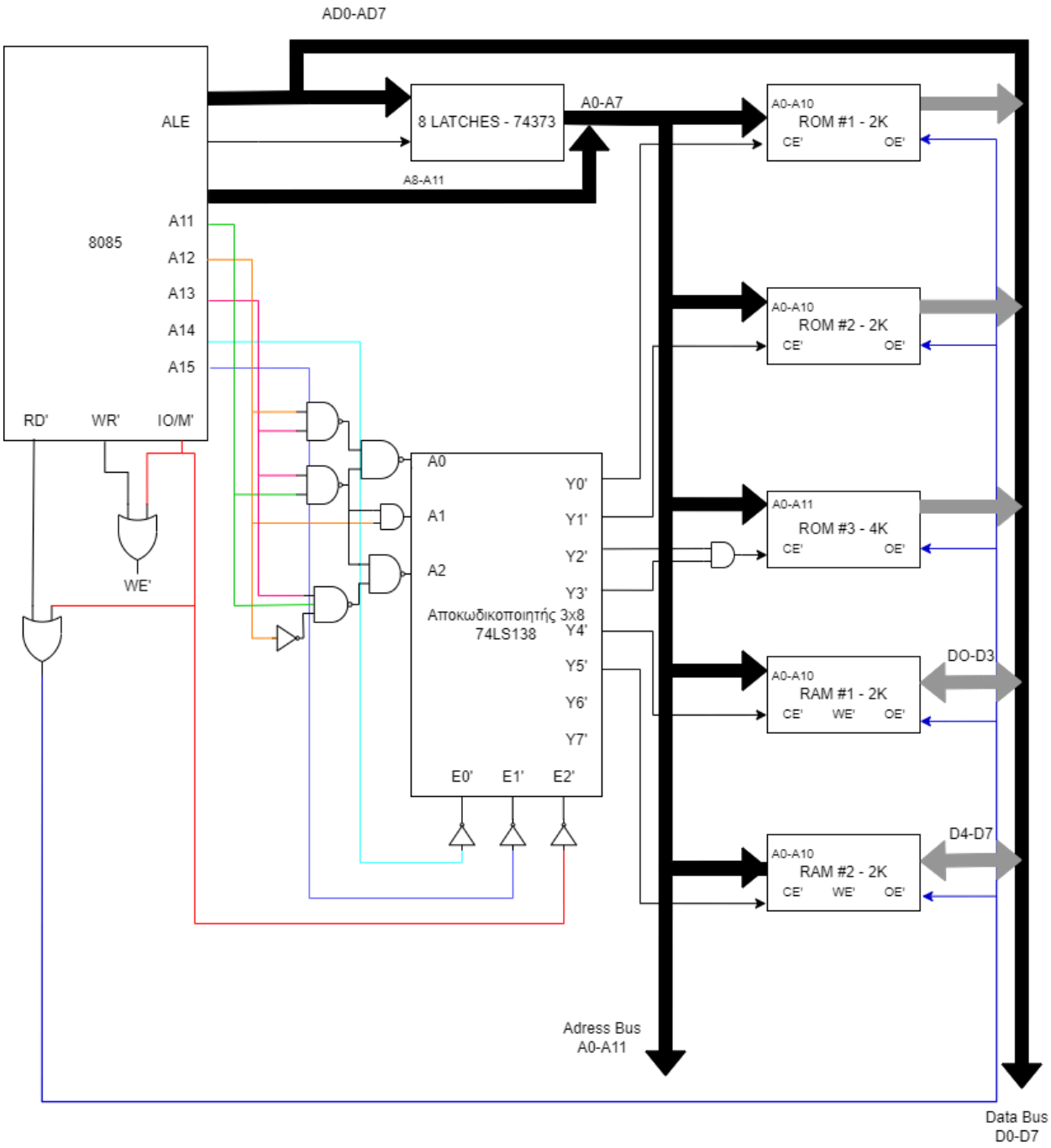
A13	A12	A11	A	B	C
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

A13	A12	A11	A	B	C
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	1	X
1	1	1	1	1	X

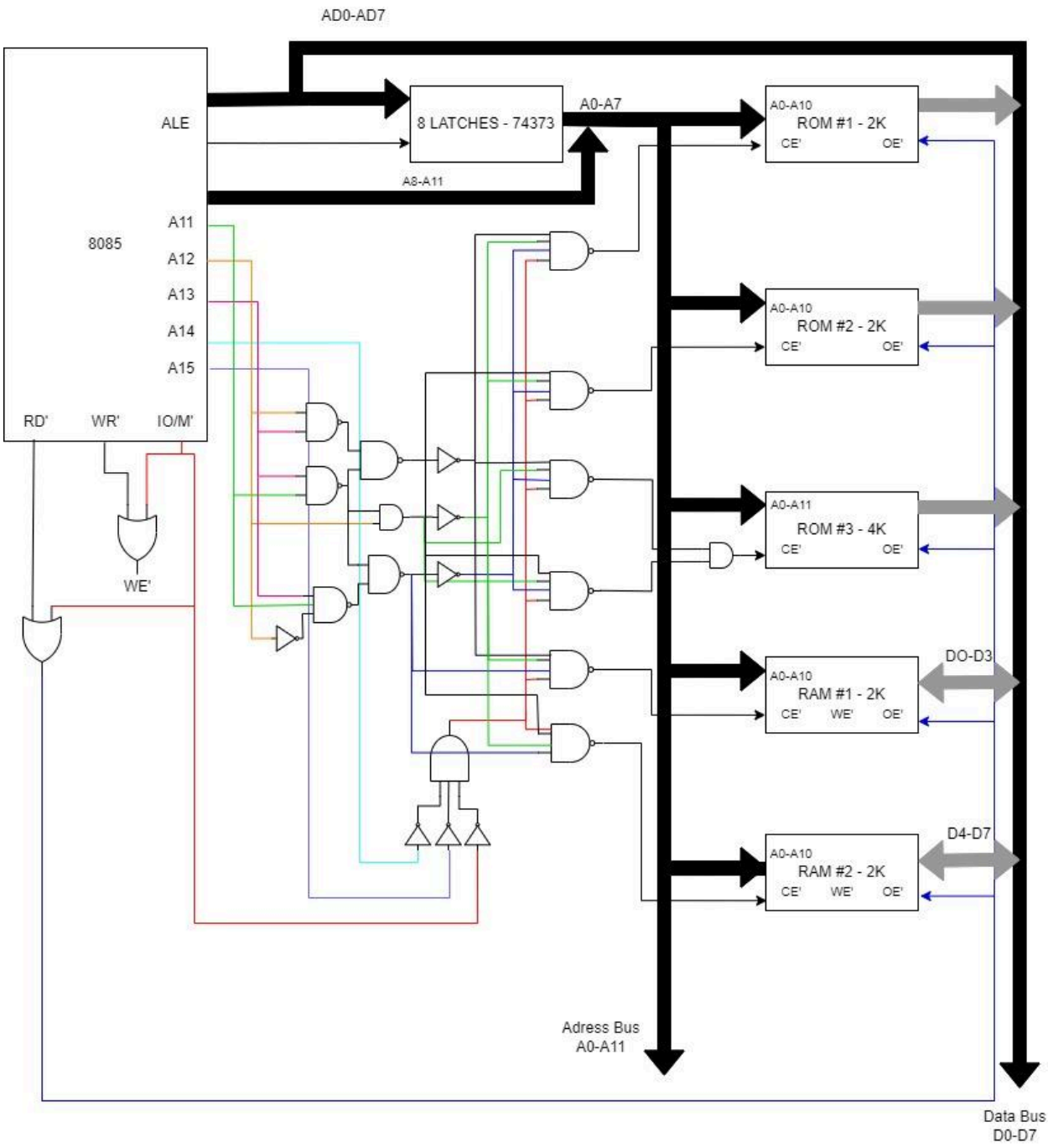
Μετά από πράξεις και μεθόδους της σχεδίασης λογικών κυκλωμάτων προκύπτουν:

- A = A13*A11+A13*A12
- B = A12+A13*A12
- C = A13*A12*A11 + A13*A12

Όλα τα παραπάνω τελικά οδηγούν στην ακόλουθη σχεδίαση:



β) Έπειτα, θα υλοποιήσουμε το ίδιο κύκλωμα χρησιμοποιώντας μόνο λογικές πύλες:


AD0-AD7
ALE
8 LATCHES - 74373
A0-A7
A8-A11
8085
A11
A12
A13
A14
A15
RD'
WR'
IO/M'
WE'
A0-A10
ROM #1 - 2K
CE'
OE'
A0-A10
ROM #2 - 2K
CE'
OE'
A0-A11
ROM #3 - 4K
CE'
OE'
DO-D3
A0-A10
RAM #1 - 2K
CE'
WE'
OE'
D4-D7
A0-A10
RAM #2 - 2K
CE'
WE'
OE'
Adress Bus
A0-A11
Data Bus
D0-D7

<u>Άσκηση 7</u>

Οι προδιαγραφές προσδιορίζουν την σχεδίαση. Ο μΕ 8085 συνεπάγεται διευθύνσεις 16 bits και δεδομένα 8 bits.

Μνήμη	Διεύθυνση μνήμης	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM#1 - 8K	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM#1 - 4K	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
RAM #2 - 4K	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
RAM #3 - 4K	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
ROM #2 - 8K	5000	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	6FFF	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1

Η μνήμη RAM δίνεται ως μία 3άδα από RAMs των 4KBytes x 8 bits σύμφωνα με τα διαθέσιμα υλικά. Η μνήμη ROM δίνεται με μέγεθος 16KBytes αφού συγκροτείται από δύο περιοχές στον χάρτη μνήμης, την περιοχή 0000-1FFFH και την περιοχή 5000-6FFFH, μεγέθους 8KBytes η καθεμία.

Από τον χάρτη μνήμης προκύπτει το A_{15} να χρησιμεύει ως είσοδος επίτρεψης, ενώ τα bits από A_{12}-A_{14} χρειάζονται για τον μοναδιαίο προσδιορισμό κάθε ολοκληρωμένου κυκλώματος μνήμης. Επομένως, βγαίνει ο εξής πίνακας αποκωδικοποίησης.

α/α	14	13	12	A	B	C	Μνήμη

Τα X στον πίνακα αληθείας αναφέρονται σε αδιάφορους όρους, αφού ο συνδυασμός στον οποίο αντιστοιχούν δεν έχει κάποια πρακτική σημασία. Χρειάζεται όμως προσοχή στο τι θα επιλεχθεί για τα X. Για παράδειγμα η στήλη A περιέχει μηδενικά και τον αδιάφορο όρο. Θα μπορούσε για εξοικονόμηση χώρου στην

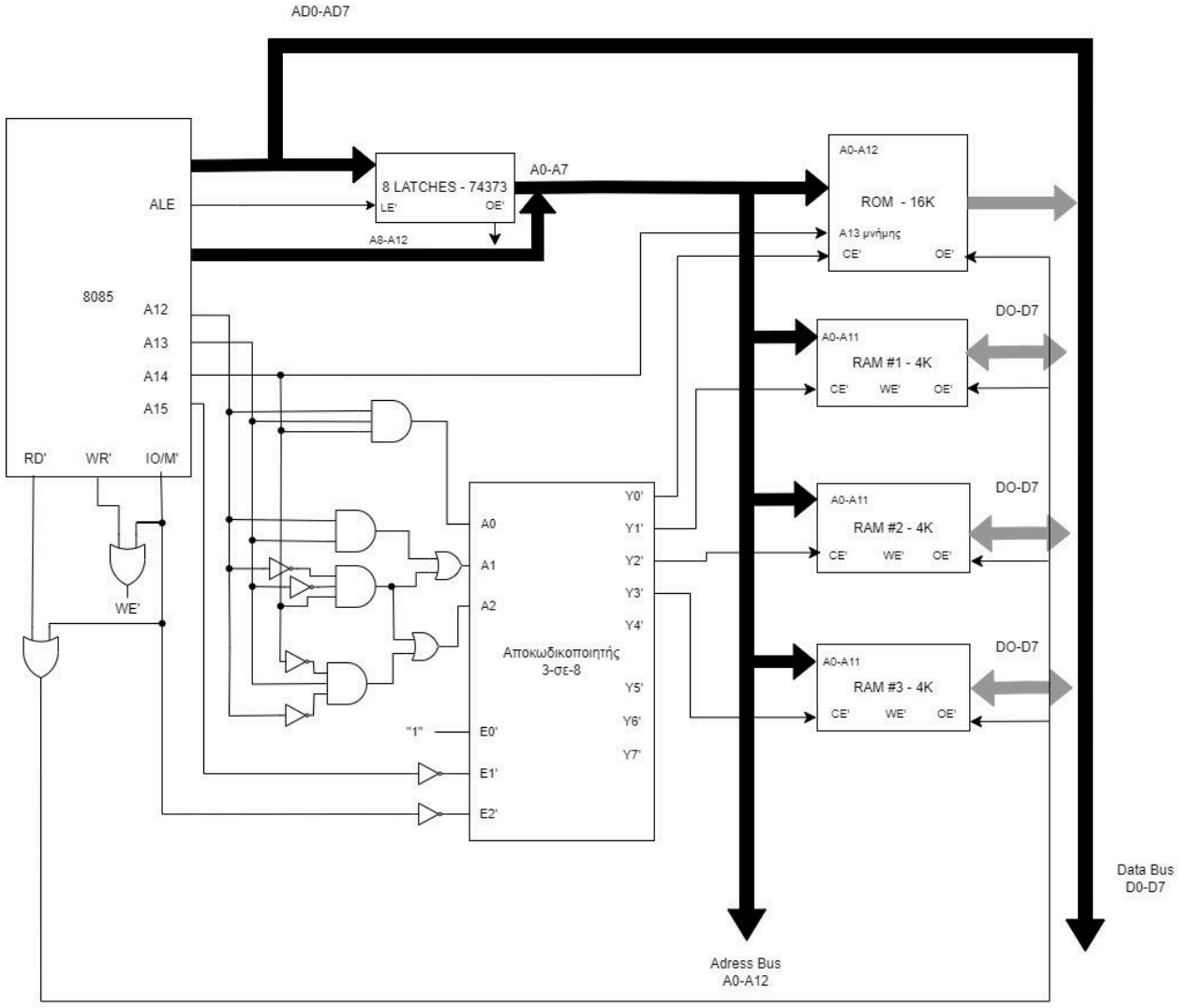
0	0	0	0	0	0	0	ROM
1	0	0	1	0	0	0	ROM
2	0	1	0	0	0	1	RAM1
3	0	1	1	0	1	0	RAM2
4	1	0	0	0	1	1	RAM3
5	1	0	1	0	0	0	ROM
6	1	1	0	0	0	0	ROM
7	1	1	1	X	X	X	-

A = A_{14} A_{13} A_{12} B = A_{13} A_{12} + A_{14} A_{13}' A_{12}', C = A_{14}' A_{13} A_{12}' + A_{14} A_{13}' A_{12}'

Όπως αναφέρθηκε η ROM αποτελείται από δύο περιοχές του χάρτη μνήμης. Προφανώς η δεύτερη περιοχή μετά την RAM, δεν αντιστοιχεί σε μια μνήμη των 8KBytes αλλά σε μία μνήμη 28KBytes. Όμως, επειδή η διαθέσιμη μνήμη ROM που έχουμε είναι μία των 16KBytes, γίνεται μετατόπιση της δεύτερης περιοχής μνήμης στα όρια της μνήμης των 16KBytes. Συγκεκριμένα, στις διευθύνσεις 0000-3FFFH της 16KB μνήμης, η πρώτη περιοχή του χάρτη μνήμης και της ROM, αντιστοιχεί επακριβώς στην περιοχή 0000-1FFF και η τρίτη περιοχή του χάρτη μνήμης από 5000-6FFFH, αντιστοιχίζεται στην περιοχή 2000-3FFFH της μνήμης. Το θέμα που προκύπτει τώρα είναι πως θα διακρίνονται οι διευθύνσεις της πρώτης περιοχής του χάρτη μνήμης από την τρίτη. Παρατηρώντας τον αρχικό χάρτη μνήμης, διαπιστώνουμε ότι οι δύο περιοχές της ROM ως προς τα bits A_{12}-A_{14}, διαφέρουν κυρίως ως προς το A_{14}. Όταν αυτό ισούται με 0, τότε λαμβάνεται η πρώτη περιοχή μνήμης, ενώ όταν είναι 1, λαμβάνεται η δεύτερη περιοχή. Το ίδιο προκύπτει και από τον χάρτη μνήμης των 16KBytes με την διαφορά ότι τον ρόλο του A_{14} έχει το A_{13}, γεγονός αναμενόμενο, γιατί μια μνήμη 16KB απαιτεί 14 bits από A_0-A_{13} για τον προσδιορισμό όλων των δυνατών διευθύνσεων. Στην σχεδίαση λοιπόν του συστήματος στο A_{13} που αφορά την μνήμη εισέρχεται το A_{14} από τον 8085. Στον επόμενο χάρτη τα ονόματα ROM1 και ROM2 αναφέρονται στις δύο περιοχές της ROM του αρχικού χάρτη.

Χάρτης Μνήμης 16KB

Μνήμη	Διεύθυνση μνήμης	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM#1 - 8K	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM #2 - 8K	2000	0	0	1	0	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



Τέλος για την είσοδο και την έξοδο του συστήματος απορρέει ο χάρτης μνήμης,

Διεύθυνση μνήμης	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7000H	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
70H	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0

Τα bits χωρίζονται σε τρεις ομάδες, κάθε μια εκ των οποίων χρωματίζεται διαφορετικά. Η ροζ ομάδα χρησιμεύει ως είσοδος επίτρεψης όπως φαίνεται στο σχήμα. Η μπλέ ομάδα χρειάζεται ώστε μετά την αποκωδικοποίηση να επιλέγεται η σωστή θύρα εξόδου σύμφωνα με τις τιμες A_{12}-A_{14}. Τέλος η πορτοκαλί ομάδα καθορίζει τις τιμές στην είσοδο του αποκωδικοποιητή. Η ξεχωριστή παρουσίαση γίνεται για να δοθεί η προσοχή που χρειάζεται σε κάθε μέρος του μΥ συστήματος.