Συστήματα Μικρουπολογιστών - 2ή Σειρά Ασκήσεων

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Ακαδημαϊκό έτος : 2018 – 2019 Εξάμηνο : 6ό
Μέλη ομάδας : Βόσινας Κωνσταντίνος ΑΜ : 03116435
Ανδριόπουλος Κωνσταντίνος ΑΜ : 03116023

Ασκήσεις Προσωμοίωσης

Άσκηση 1η

Τα παρακάτω προγράμματα επαληθεύτηκαν ως προς την ορθότητα τους με χρήση του προσωμοιωτή του x8085

Το πρόγραμμα που ακολουθεί υλοποιεί τα 3 ερωτήματα της άσκησης.

```
START:

    IN 10H      ;Initially diasble memory protection
    LXI H,0900H ;Initialize address
    MVI A,00H       ;Initialize number to be stored
L1:
    MOV M,A     ;Store number
    INX H       ;Increment address
    INR A       ;Increment Number
    CPI 00H     ;In case of A=00H, aka overflow
    JNZ L1      ;Do not repead, else do repeat

TASK_B:

    LXI H,0900H ;Initialize address BC
    MVI B,00H
    MVI C,00H

ONES:
    MVI D,00H       ;D is the number of digits checked, initially 0
    MOV A,M     ;Fetch number from memory

COUNT_BITS:

    MOV E,A     ;Store acc temporarily
    MOV A,D     ;Check if D=08H, if all digits have been checked
    CPI 08H
    JZ NEXT     ;If yes, check next number
    INR D       ;Else increment D
    MOV A,E     ;Fetch next digit of A
    RAL         ;Using CY from RAL
    JNC COUNT_BITS
    INX B       ;IF CY=1 then increment BC
    JMP COUNT_BITS ;Loop until D=08H

NEXT:
    INX H       ;Increment address
    MOV A,H     ;Check if address is past 09FFH
    CPI 0AH
    JZ TASK_C       ;If yes, begin next task
    JMP ONES        ;else get next number
```

```
TASK_C:
    LXI H,0900H
    MVI D,00H

IS_BETWEEN:
    MOV A,M     ;Fetch a number
    CPI 10H     ;Check if it is less than 10
    JC NEXT_NUM ;If yes, check next number
    CPI 60H     ;Check if greater than 60
    JC NEXT_NUM
    INR D       ;If neither, number is in range, increment D

NEXT_NUM:
    INX H       ;Increment address
    MOV A,H     ;Check if address is past 09FFH
    CPI 0AH
    JNZ IS_BETWEEN  ;If NOT, get next number
END
```

Άσκηση 2η

```
START:

    LXI B,00C8H     ;Set time unit equal to 200ms
                    ;BC = 00C8H = 200 DEC
    MVI D,96H       ;D is a counter = 150 in decimal
                    ;150*200ms = 30 sec

INPUT_LOOP:
    LDA 2000H       ;Read input
    RAL             ;Using carry check MSB
    CALL DELB       ;Delay by one time unit
    JC INPUT_LOOP   ;If MSB==1 keep reading, else
                    ;go to state off1

STATE_OFF1:
    LDA 2000H       ;Similarily, read until MSB=1
    RAL
    CALL DELB
    JNC STATE_OFF1

STATE_ON1:
    LDA 2000H       ;Read until MSB=0 again
    RAL             ;OFF->ON->OFF
    CALL DELB
    JC STATE_ON1

STATE_OFF2:
    CALL TURNON_LED ;Turn on MSB led

    LDA 2000H       ;Read input again
    RAL
    CALL DELB       ;Check if MSB=1, in which case
    JC STATE_ON2    ;Go to state ON2
    DCR D           ;Else, decrease counter
    MOV A,D         ;Check if counter=0
    CPI 00H         ;Meaning 30 seconds have passed
    JNZ STATE_OFF2  ;If not repeat
    CALL TURNOFF_LED
    MVI D,96H       ;Else, reset counter and restart process
    JMP INPUT_LOOP

STATE_ON2:
    CALL TURNON_LED ;Similarily as state off2
    LDA 2000H
    RAL
    CALL DELB
    JNC STATE_OFF3
    DCR D
    MOV A,D
    CPI 00H
    JNZ STATE_ON2
    CALL TURNOFF_LED
    MVI D,96H
    JMP INPUT_LOOP

STATE_OFF3:
    MVI D,96H       ;If another OFF->ON->OFF detected
    JMP STATE_OFF2  ;Reset timer and return to off2
```

```
TURNON_LED:
    MVI A,00H
    STA 3000H
    RET

TURNOFF_LED:
    MVI A,FFH
    STA 3000H
    RET
END
```

Άσκηση 3η

(i)

```
START:

    LDA 2000H   ;LOAD input
    MVI D,00H   ;D is a counter, position of LSB which is ON
                ;Initially at 0
    MOV E,A     ;E is a temporary register

LOOP1:
    MOV A,E     ;Load data to accumulator
    RAR         ;Check LSB with RAR
    JC OUTPUT   ;If LSB=1, output result using D register
    INR D       ;Else, increment D
    MOV E,A
    MOV A,D     ;Check if D<8, in which case restart loop
    CPI 08H
    JNZ LOOP1
    JMP OUTPUT  ;If D=8 output result also


OUTPUT:
    MVI B,00H   ;B is a counter
    MVI C,FFH   ;C containts the data to be printed on the leds
                ;Initially set at FF, all leds on
LOOP2:
    MOV A,B     ;Check if B has reached D's value
    CMP D       ;If yes, fo to RESULT
    JZ RESULT   ;Which turns leds on, using C's value
    MOV A,C     ;Else, shift C left once
    RLC
    MOV C,A     ;By decrementing C, LSB becomes 0
    DCR C       ;Example C=11111000 => C=11110000
    INR B       ;Increment B and loop again
    JMP LOOP2

RESULT:
    MOV A,C     ;C contains the correct value
    CMA
    STA 3000H
    JMP START

END
```

(ii)

```
START:

    CALL KIND   ;Read input and decrement by 1
    DCR A       ;So that it is in range [0,7]

    MVI D,00H   ;D is a counter, position of LSB which is ON
                ;Initially at 0
    MOV E,A     ;E is a temporary register

LOOP1:
    MOV A,E     ;Load data to accumulator
    CMP D
    JZ OUTPUT   ;If LSB=1, output result using D register

    INR D       ;Else, increment D
    MOV E,A
    MOV A,D     ;Check if D<8, in which case restart loop
    CPI 08H
    JNZ LOOP1
    MVI A,FFH   ;If D=8 output zeros
    STA 3000H
    JMP START

OUTPUT:
    MVI B,00H   ;B is a counter
    MVI C,FEH   ;C containts the data to be printed on the leds
                ;Initially set at 11111110, only led 0 is on
LOOP2:
    MOV A,B     ;Check if B has reached D's value
    CMP D       ;If yes, go to RESULT
    JZ RESULT   ;Which turns leds on, using C's value
    MOV A,C     ;Else, shift C left once
    RLC
    MOV C,A
    INR B       ;Increment B and loop again
    JMP LOOP2

RESULT:
    MOV A,C     ;C contains the correct value
    STA 3000H
    JMP START

END
```

(iii)

```
START:
    IN 10H

    MVI B,01H       ;B is the output of display, increments starting from 1
    MVI D,F7H       ;Initial position of line 3
    MVI C,00H       ;C is a counter, in order to read 5 final lines

LINES3_TO_7:
                ;These lines contain the numbers between 1 and F
                ;Three numbers on each line, starting from line 3

    MOV A,D       ;Read line through accumulator
    STA 2800H
    LDA 1800H
    ANI 07H       ;07H = 000000111, so sets 5 MSB's to 0
    CPI 06H       ;Compare with 00000110, so 1,4,7 etc
    JZ DISPLAY    ;Call DISPLAY if equal, print based on B
    INR B         ;Increment B
    CPI 05H       ;Compare with 00000101, so 2,5,8 etc
    JZ DISPLAY
    INR B
    CPI 03H       ;Compare with 00000011, so 3,7,9 etc
    JZ DISPLAY
    INR B
    MOV A,D       ;Shift D's value once to the left
    RLC           ;So that the mask now points to next line
    MOV D,A
    INR C          ;Increment C, repeat 5 times, once for each line
    MOV A,C
    CPI 05H
    JNZ LINES3_TO_7

;No discernible pattern for lines 0->2
;As such, they are handled manually
LINE2:
    MVI A,FBH     ;Line 2 contains 0, STORE/INCR, INCR
    STA 2800H
    LDA 1800H
    ANI 07H
    MVI B,00H     ;Checking 0
    CPI 06H
    JZ DISPLAY
    MVI B,83H     ;Checking STORE/INCR
    CPI 05H
    JZ DISPLAY
    MVI B,81H     ;Checking ICR
    CPI 03H
    JZ DISPLAY

LINE1:
    MVI A,FDH     ;Line 1 contains RUN, FETCH/REG, FETCH ADDR
    STA 2800H
    LDA 1800H
    ANI 07H
    MVI B,84H     ;Checking RUN
    CPI 06H
    JZ DISPLAY
    MVI B,80H     ;Checking FETCH/REG
    CPI 05H
    JZ DISPLAY
    MVI B,82H     ;Checking FETCH ADDR
    CPI 03H
    JZ DISPLAY
```

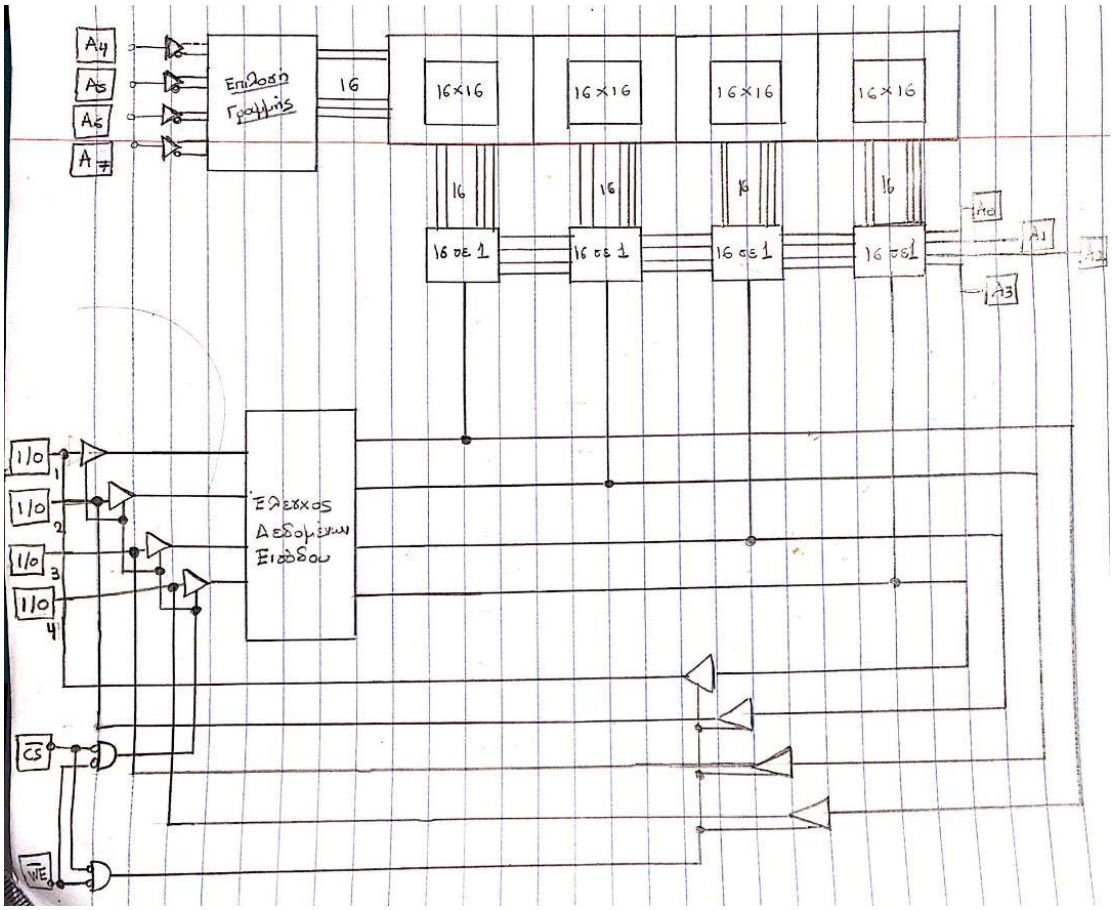
```
LINE0:
    MVI A,FEH     ;Line 0 contains INSTR STEP, FETCH PC, HDWR STEP
    STA 2800H
    LDA 1800H
    ANI 07H
    MVI B,86H     ;Checking INSTR STEP
    CPI 06H
    JZ DISPLAY
    MVI B,85H     ;Checking FETCH PC
    CPI 05H
    JZ DISPLAY

    JMP START     ;No keys pressed, repeat process

DISPLAY:
    MOV A,B       ;Move code to accumulatoe
    ANI 0FH       ;Keep only 4 LSB's
    STA 0900H     ;Store to adress 0900, from where it will be displayed

    MOV A,B
    ANI F0H       ;Keep only 4 MSB's
    RRC           ;Shift 4 times, so they are in the place of 4 LSB's
    RRC
    RRC
    RRC
    STA 0901H     ;Store in next memory location
    LXI D,0900H   ;Point DE to memory location 0900

    CALL STDM     ;Call 7-segment display processes
    CALL DCD
    JMP START
END
```

Άσκηση 4η

```
START:
    IN 10H
    MVI E,00H       ;E contains the result, initially set at 0

    LDA 2000H       ;Load input

CALCULATE_X3:
    RLC             ;Rotate once, now next digit we want to use is in LSB position
    MOV D,A         ;Save current

    ANI 01H         ;Set all digits except LSB 0

    MOV B,A         ;Store in B, it is now either 00H, or O1H

    CALL NEXT_DIGIT ;Get next digit
    ANA B           ;A (and) B -> A, now A is either 00H, or 01H
    ADD E           ;Add the result to E, essentially setting its LSB to result
    RLC             ;Rotate left once, so next result can be added
    MOV E,A         ;Update E


CALCULATE_X2:
    CALL NEXT_DIGIT     ;Similarily as X3
    MOV B,A
    CALL NEXT_DIGIT
    ANA B
    ADD E
    RLC
    MOV E,A

CALCULATE_X1:
    CALL NEXT_DIGIT
    MOV B,A
    CALL NEXT_DIGIT
    ORA B
    MOV C,A         ;Result is needed on X1, so we save it
    ADD E
    RLC
    MOV E,A

CALCULATE_X0:
    CALL NEXT_DIGIT
    MOV B,A
    CALL NEXT_DIGIT
    ORA B           ;First A or B is calculated
    XRA C           ;Then the result is used on the XOR gate
    ADD E
    MOV E,A

RESULT:
    MOV A,E         ;E is now ready to be displayed
    CMA
    STA 3000H
    JMP START

NEXT_DIGIT:
    MOV A,D         ;Get current state of input
    RLC             ;Rotate once so the digit we want is on LSB position
    MOV D,A         ;Save current state
    ANI 01H         ;Use mask, so everything except LSB is set to 0
    RET
END
```

Άσκηση 5η

Η μνήμη είναι 256 x 4 bits. Ο δεύτερος αριθμός, το 4, δηλώνει σε πόσα ίσα μέρη θα είναι χωρισμένη η μνήμη κι ο πρώτος αριθμός, το 256, δηλώνει πόση χωρητικότητα μνήμης έχει το κάθε μέρος. Άρα η μνήμη θα κατακερματίζεται σε 4 μέρη, ισοδύναμα τράπεζες, των 256 bits το καθένα. Κάθε τράπεζα από την μεριά της είναι ένας διδιάστατος πίνακας με γραμμές και στήλες. Προφανώς το μέγεθος του πίνακα αυτού, πρέπει να ισούται σταθερά με 256 ανεξάρτητα από την κατανομή των γραμμών και των στηλών. Ο λόγος είναι οτι το 256 συνιστά προδιαγραφή.
Γίνεται φανερό πως οι γραμμές κι οι στήλες μπορούν να κατανεμηθούν με πολλούς τρόπους. Επιλέχθηκε τετραγωνική διάταξη για την κάθε τράπεζα. Το πλήθος των γραμμών ισούται με αυτό των στηλών, δηλαδή, 16. Η επιλογή αυτή επέφερε την χρήση πολυπλεκτών 16 - σε - 1. Επίσης επέφερε την χρήση 4 bits τόσο για τον προσδιορισμό της διεύθυνσης των γραμμών όσο και των στηλών. Τέλος, το γεγονός των 4 τραπεζών επιβάλλει την ύπαρξη και 4 σημάτων I/O για κάθε μία από αυτές.
Για να γίνει ανάγνωση από την μνήμη αρχικά εφαρμόζεται στις εισόδους διεύθυνσης A0 - A7 η διεύθυνση. Έπειτα επιλέγεται το κατάλληλο ολοκληρωμένο κύκλωμα SRAM μέσω του ακροδέκτη επιλογής CS. Στην συνέχεια, μετά από πάροδο χρόνου όσος κι ο χρόνος προσπέλασης εμφανίζονται τα δεδομένα στις εξόδους δεδομένων I/O. Αντίστοιχα για να γίνει εγγραφή πάλι στις εισόδους διεύθυνσης εφαρμόζεται η διεύθυνση κι επιλέγεται το κατάλληλο ολοκληρωμένο κύκλωμα SRAM. Μετά τα δεδομένα προς εγγραφή εφαρμόζονται στις εισόδους δεδομένων I/O και στον ακροδέκτη WE στέλνεται αρνητικός παλμός ώστε να επιτραπεί και να εκτελεστεί η λειτουργία της εγγραφής των δεδομένων.



Άσκηση 6η

Το σύστημα μνήμης της άσκησης χρησιμοποιείται από επεξεργαστή x8085, όπου η αναπαράσταση των δεδομένων είναι 8-bit και η διευθυνσιοδότηση απαιτεί 16-bit.

Συνολικό μέγεθος ROM είναι 10KBytes = (8Kbytes + 2Kbytes) = $(2^{13}+2^{11})$*8bits, άρα χρησιμοποιούνται 14 bits, δηλαδή για τη ROM έχουμε τα A_0-A_{13}.

Συνολικό μέγεθος RAM είναι 10KBytes = (4Kbytes + 2Kbytes) = $(2^{12}+2^{11})$*8bits, άρα χρησιμοποιούνται 13 bits, δηλαδή για τη ROM έχουμε τα A_0-A_{12}.

Για τον αποκωδικοποιητή, πρέπει να επιλέγει μεταξύ των μνημών RAM και ROM καθώς και μεταξύ των διαφορετικών Ics.

ROM				Χάρτης Μνήμης		RAM			
Αρχή Διευθύνσεων									
0000 0	0000 0	0000 0	0000 0	BIN HEX	0010 2	1000 8	0000 0	0000 0	BIN HEX
Τέλος Διευθύνσεων									
0010 2	0111 7	1111 F	1111 F	BIN HEX	0111 3	1111 F	1111 F	1111 F	BIN HEX

Χάρτης μνήμης											
ROM 1				ROM 2				ROM 3			
Αρχή Διευθύνσεων											
0000 0	0000 0	0000 0	0000 0	0001 1	0000 0	0000 0	0000 0	0010 2	0000 0	0000 0	0000 0
Τέλος Διευθύνσεων											
0000 0	1111 F	1111 F	1111 F	0001 1	1111 F	1111 F	1111 F	0010 2	0111 7	1111 F	1111 F

SRAM1				Χάρτης Μνήμης		SRAM2			
Αρχή Διευθύνσεων									
0010 2	1000 8	0000 0	0000 0	BIN HEX	0011 3	0000 0	0000 0	0000 0	BIN HEX
Τέλος Διευθύνσεων									
0010 2	0111 7	1111 F	1111 F	BIN HEX	0111 3	1111 F	1111 F	1111 F	BIN HEX

Επομένως, σύμφωνα με τους παραπάνω χάρτες μνήμης, έχουμε ότι με τα A_{11}-A_{13} προσδιορίζεται η κατάλληλη μνήμη, από κάποιο μοναδικό συνδυασμό των ψηφίων αυτών. Επομένως προκύπτει ο πίνακας αλήθειας.

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

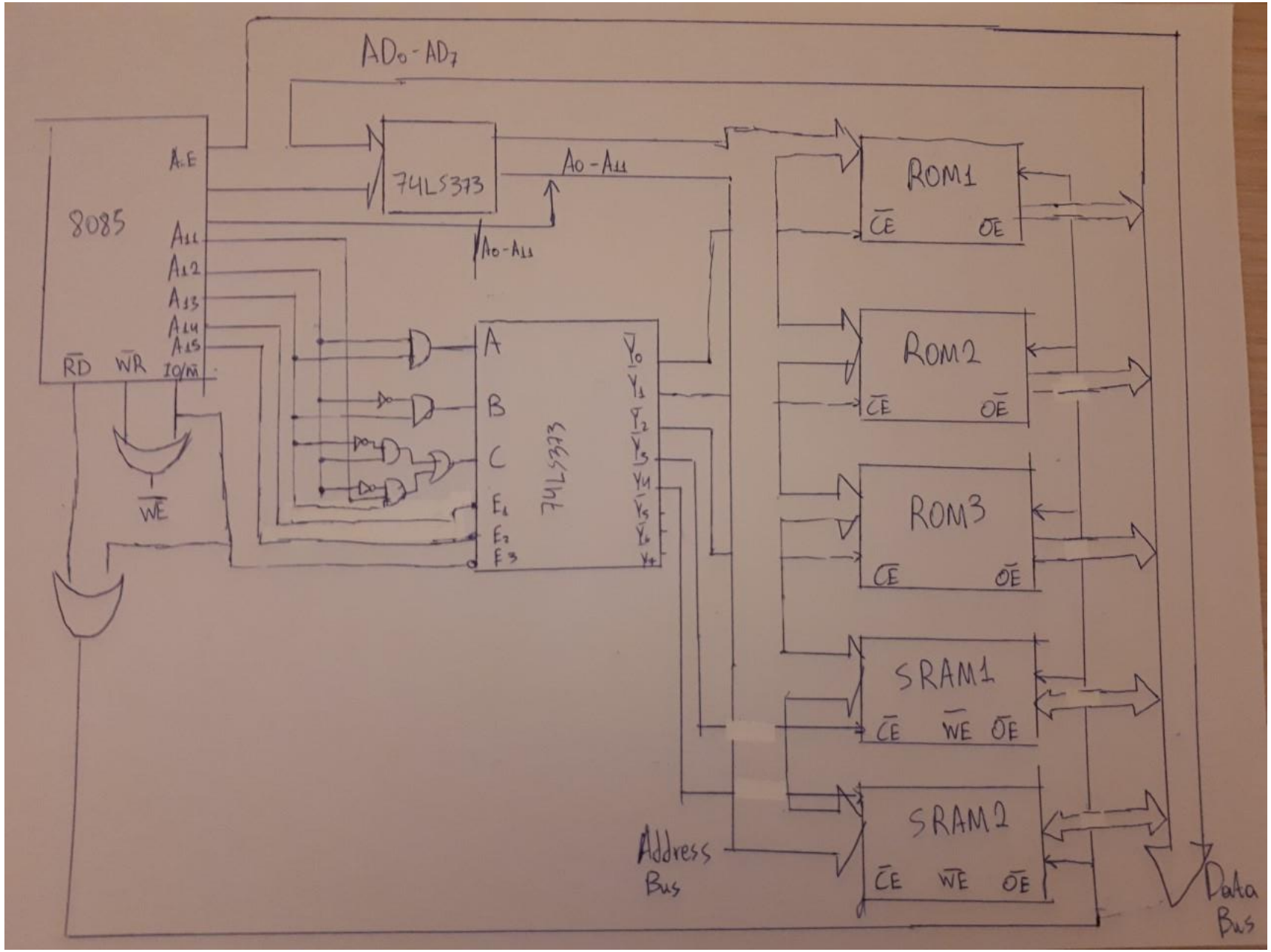
Επομένως προκύπτουν :

$$A = A_{13}A_{12}$$

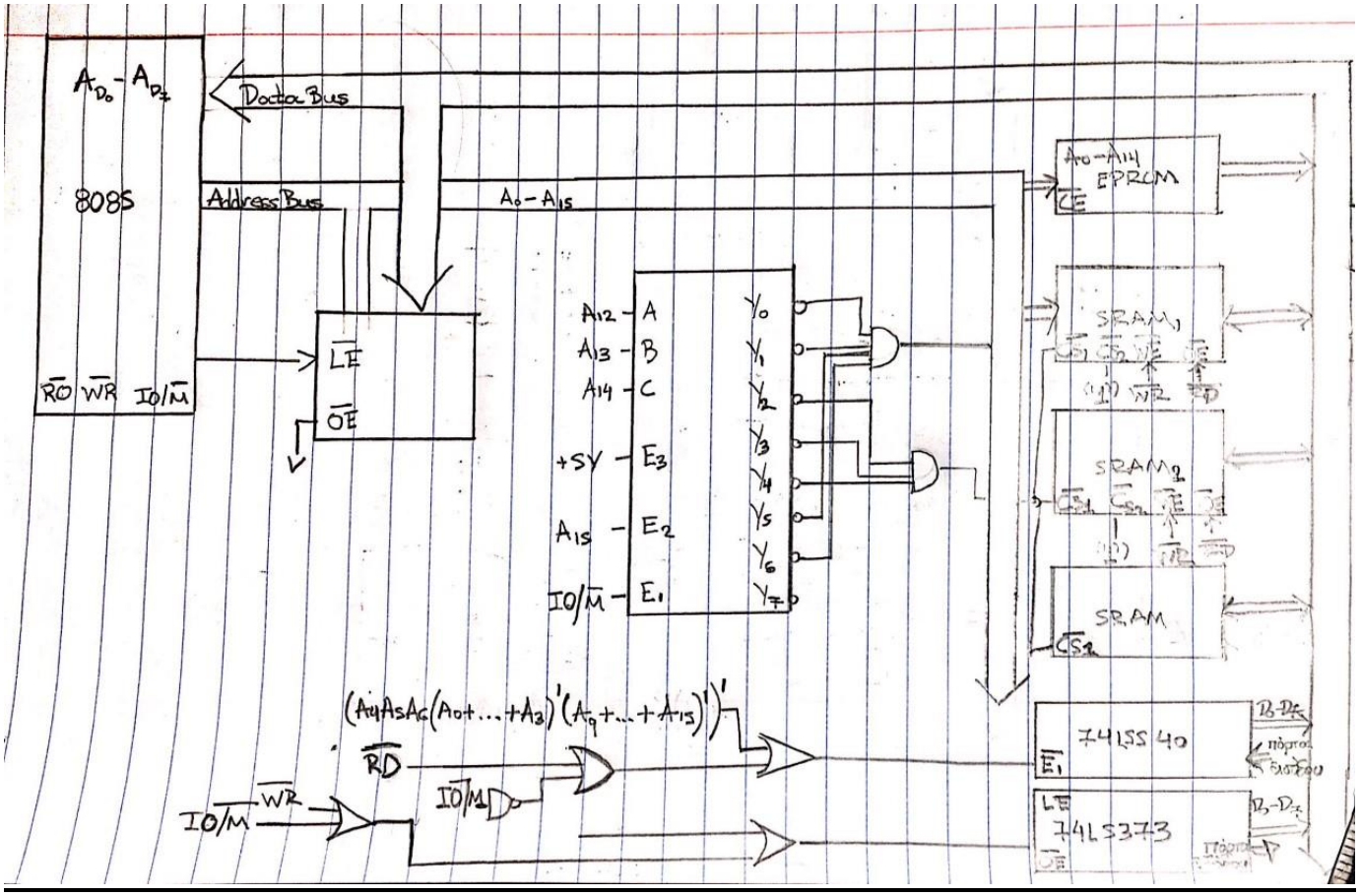
$$B = A_{13}A_{12}$$

$$C = A'_{13}A_{12} + A_{13}A'_{12}A_{11}$$

Με βάση τα παραπάνω λοιπόν θα σχεδιάσουμε τον αποκωδικοποιητή, ο οποίος θα επιλέγει το κατάλληλο ολοκληρωμένο.


AD0 - AD7
8085
A.E
A11
A12
A13
A14
A15
RD
WR
IO/M
WE
74LS373
A0 - A11
A0 - A11
74LS373
A
B
C
E1
E2
E3
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
ROM1
CE
OE
ROM2
CE
OE
ROM3
CE
OE
SRAM1
CE
WE
OE
SRAM2
CE
WE
OE
Address Bus
Data Bus

Άσκηση 7


8085
Data Bus
Address Bus
A0 - A15
RD WR IO/M
LE
OE
A12
A13
A14
+5V
A15
IO/M
A
B
C
E3
E2
E1
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
A0-A11
EPROM
CE
SRAM
SRAM
SRAM
WR
74155 40
74LS373
RD
IO/M
WR
IO/M