ΣΥΣΤΗΜΑΤΑ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ

2η Εργαστηριακή Άσκηση

Ακαδημαϊκό έτος 2020-2021

Χριστίνα Προεστάκη | ΑΜ : 03118877

Νικόλαος Μπέλλος | ΑΜ : 03118183

1η Άσκηση

Κώδικας ASSEMBLY για 8085

```
      IN 10H          ;disable memory protection
      MVI A,00H       ;counter for question (a)
      LXI H,0900H     ;address
      LXI B,0000H     ;counter for question (b)
      MVI D,00H       ;counter for question (c)
START:
      MOV M,A         ;moves A to memory (H)
      MOV E,A         ;stores A to E temporarily
      JMP QUEST_B
LOOP_NUM:
      INX H           ;increase address (H) by 1
      INR A           ;A = A + 1
      CPI 00H         ;checks if A is 256 (if true CY=0)
      JZ FINISH       ;if CY=0 goto FINISH
      JMP START       ;else start process with new number
GOTO_C:
      MOV A,E
      JMP QUEST_C
QUEST_B:
      RAR             ;CY = LSB(A)
      JNC CHECK_A     ;if CY = 0 check if end of number
      INX B           ;else increase B
CHECK_A:
      CPI 00H         ;check if A = OOH
      JZ  GOTO_C      ;if A = 00H goto question (b)
      JMP QUEST_B     ;else continue count of ones
QUEST_C:
      CPI 10H         ;compare A with 10H
```

```
      JC LOOP_NUM      ;if A < 10H continue to next number
      CPI 60H          ;compare A with 60H
      JC INC_D         ;if A < 60H increase D
      JNZ LOOP_NUM     ;else continue with new A
INC_D:
      INR D            ;D = D + 1
      JMP LOOP_NUM     ;continue with new A
FINISH:
      RST 1
      END
```

2η Άσκηση

```
      MVI   D,C8H           ;200 * 0.1s = 20s
      LXI   B,0064H         ;100*10^(-3) = 0.1s

FIRST_CHECK:
      LDA   2000H
      RAL                   ;check MSB -> CY
      JC    FIRST_CHECK     ;if ON repeat
      JMP   FIRST_OFF

FIRST_OFF:
      LDA   2000H
      RAL
      JC    FIRST_ON        ;if ON
      JMP   FIRST_OFF       ;if OFF, repeat until ON

FIRST_ON:
      LDA   2000H
      RAL
      JC    FIRST_ON        ;if ON, repeat until OFF
      JMP   SECOND_OFF      ;if OFF

SECOND_OFF:
      LDA    2000H
      RAL
      JC    SECOND_ON       ;if ON, check for next OFF
      MVI   A,00H           ;if else
      STA   3000H           ;turn on LED
      CALL  DELB            ;wait 0.1sec
      DCR   D               ;decrease D, hence the time
```

```
      MOV   A,D
      CPI   00H             ;if D = 0, time is up (Z = 1)
      JZ    AGAIN           ;repeat
      JMP   SECOND_OFF      ;repeat until time is up

SECOND_ON:
      LDA   2000H
      RAL
      JNC   RESTART_TIME    ;if OFF, restart time
      MVI   A,00H
      STA   3000H           ;if ON, turn on LED
      CALL  DELB            ;wait 0.1sec
      DCR   D               ;decrease D
      MOV   A,D
      CPI   00H             ;if D = 0, time is up
      JZ    AGAIN           ;repeat
      JMP   SECOND_ON       ;repeat until time is up


AGAIN:
      MVI   A,FFH           ;turn off LED
      STA   3000H
      MVI   D,C8H           ;restart D = 200
      JMP   FIRST_OFF       ;repeat from the start

RESTART_TIME:
      MVI D,C8H             ;restart time
      JMP SECOND_OFF

      END
```

3η Άσκηση

i. Κώδικας ASSEMBLY για 8085

```
 START:
      MVI C,01H         ;Initialize counter to 1
      LDA 2000H         ;Load input from switches
      CPI 00H           ;Check if input is zeros
      JZ DISPLAY_ZEROS
```

```
GOTO_LOOP:
      RAR              ;take LSB(A)
      JC GOTO_DISPLAY  ;if LSB(A) = 1 print counter
      MOV E,A          ;store A temporarily
      MOV A,C
      RAL              ;rotate left the counter
      MOV C,A
      MOV A,E          ;restore A
      JNZ GOTO_LOOP    ;repeat for next digit
GOTO_DISPLAY:
      MOV A,C
      CMA              ;reverse bits (for LEDs)
      STA 3000H        ;output counter to LEDs
      JMP START
DISPLAY_ZEROS:
      MVI A,FFH
      STA 3000H
      JMP START

      END
```

ii. Κώδικας ASSEMBLY για 8085

```
START:
      MVI C,FFH        ;Initialize counter to 1
      CALL KIND        ;Load input from switches
      CPI 00H          ;Check if input is 0
      JZ DISPLAY_ZEROS
      CPI 09H          ;Check if input >=9
      JNC DISPLAY_ZEROS
GOTO_LOOP:
      DCR A            ;Decrease counter
      CPI 00H          ;Check if counter = 0
      JZ GOTO_DISPLAY  ;if true print counter (C)
      MOV E,A          ;store A temporarily
      MOV A,C
      RAL              ;rotate left the counter (C)
      MOV C,A
      MOV A,E          ;restore A
      JNZ GOTO_LOOP    ;repeat
GOTO_DISPLAY:
      MOV A,C
```

```
        CMA             ;reverse bits (for LEDs)
        STA 3000H       ;output counter to LEDs
        JMP START
DISPLAY_ZEROS:
        MVI A,FFH
        STA 3000H
        JMP START

        END
```

iii. Κώδικας ASSEMBLY για 8085

```
        IN 10H          ;switch off all LEDs
        LXI H,0AA0H     ;address of first digit
        MVI A,06H
        MVI B,07H       ;use register B as mask (0000 0111)
LOOP_BLANK:
        MVI M,10H
        INX H
        DCR A
        CPI 00H
        JNZ LOOP_BLANK

START:
        MVI C,FEH       ;ROW 1
        CALL TAKE_ROW
        CPI 03H
        JZ KEY_INST_STEP
        CPI 05H
        JZ KEY_FETCH_PC
        CPI 06H
        JZ KEY_HDWR_STEP

        MVI C,FDH       ;ROW 2
        CALL TAKE_ROW
        CPI 06H
        JZ KEY_RUN
        CPI 05H
        JZ KEY_FETCH_REG
        CPI 03H
        JZ KEY_FETCH_ADRS
```

```
MVI C,FBH       ;ROW 3
CALL TAKE_ROW
CPI 06H
JZ KEY_00
CPI 05H
JZ KEY_STORE
CPI 03H
JZ KEY_INCR

MVI C,F7H       ;ROW 4
CALL TAKE_ROW
CPI 06H
JZ KEY_01
CPI 05H
JZ KEY_02
CPI 03H
JZ KEY_03

MVI C,EFH       ;ROW 5
CALL TAKE_ROW
CPI 06H
JZ KEY_04
CPI 05H
JZ KEY_05
CPI 03H
JZ KEY_06

MVI C,DFH       ;ROW 6
CALL TAKE_ROW
CPI 06H
JZ KEY_07
CPI 05H
JZ KEY_08
CPI 03H
JZ KEY_09

MVI C,BFH       ;ROW 7
CALL TAKE_ROW
CPI 06H
JZ KEY_0A
CPI 05H
JZ KEY_0B
CPI 03H
```

```
      JZ KEY_0C

      MVI C,7FH        ;ROW 8
      CALL TAKE_ROW
      CPI 06H
      JZ KEY_0D
      CPI 05H
      JZ KEY_0E
      CPI 03H
      JZ KEY_0F

      CALL DISPLAY_KEY
      JMP START

TAKE_ROW:
      MOV A,C
      STA 2800H
      LDA 1800H
      ANA B
      RET

DISPLAY_KEY:
      LXI D,0AA0H
      CALL STDM
      CALL DCD
      RET

KEY_INST_STEP:
      LXI H,0AA5H
      MVI M,08H
      LXI H,0AA4H
      MVI M,06H
      CALL DISPLAY_KEY
      JMP START

KEY_FETCH_PC:
      LXI H,0AA5H
      MVI M,08H
      LXI H,0AA4H
      MVI M,05H
      CALL DISPLAY_KEY
      JMP START
```

```
KEY_HDWR_STEP:
      LXI H,0AA5H
      MVI M,0FH
      LXI H,0AA4H
      MVI M,07H
      CALL DISPLAY_KEY
      JMP START

KEY_RUN:
      LXI H,0AA5H
      MVI M,08H
      LXI H,0AA4H
      MVI M,04H
      CALL DISPLAY_KEY
      JMP START

KEY_FETCH_REG:
      LXI H,0AA5H
      MVI M,08H
      LXI H,0AA4H
      MVI M,00H
      CALL DISPLAY_KEY
      JMP START

KEY_FETCH_ADRS:
      LXI H,0AA5H
      MVI M,08H
      LXI H,0AA4H
      MVI M,02H
      CALL DISPLAY_KEY
      JMP START

KEY_STORE:
      LXI H,0AA5H
      MVI M,08H
      LXI H,0AA4H
      MVI M,03H
      CALL DISPLAY_KEY
      JMP START

KEY_INCR:
      LXI H,0AA5H
      MVI M,08H
```

```
      LXI H,0AA4H
      MVI M,01H
      CALL DISPLAY_KEY
      JMP START

KEY_00:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,00H
      CALL DISPLAY_KEY
      JMP START

KEY_01:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,01H
      CALL DISPLAY_KEY
      JMP START

KEY_02:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,02H
      CALL DISPLAY_KEY
      JMP START

KEY_03:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,03H
      CALL DISPLAY_KEY
      JMP START

KEY_04:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,04H
      CALL DISPLAY_KEY
```

```
      JMP START

KEY_05:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,05H
      CALL DISPLAY_KEY
      JMP START

KEY_06:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,06H
      CALL DISPLAY_KEY
      JMP START

KEY_07:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,07H
      CALL DISPLAY_KEY
      JMP START

KEY_08:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,08H
      CALL DISPLAY_KEY
      JMP START

KEY_09:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,09H
      CALL DISPLAY_KEY
      JMP START

KEY_0A:
```

```
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,0AH
      CALL DISPLAY_KEY
      JMP START

KEY_0B:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,0BH
      CALL DISPLAY_KEY
      JMP START

KEY_0C:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,0CH
      CALL DISPLAY_KEY
      JMP START

KEY_0D:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,0DH
      CALL DISPLAY_KEY
      JMP START

KEY_0E:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
      MVI M,0EH
      CALL DISPLAY_KEY
      JMP START

KEY_0F:
      LXI H,0AA5H
      MVI M,00H
      LXI H,0AA4H
```

```
      MVI M,0FH
      CALL DISPLAY_KEY
      JMP START

      END
```

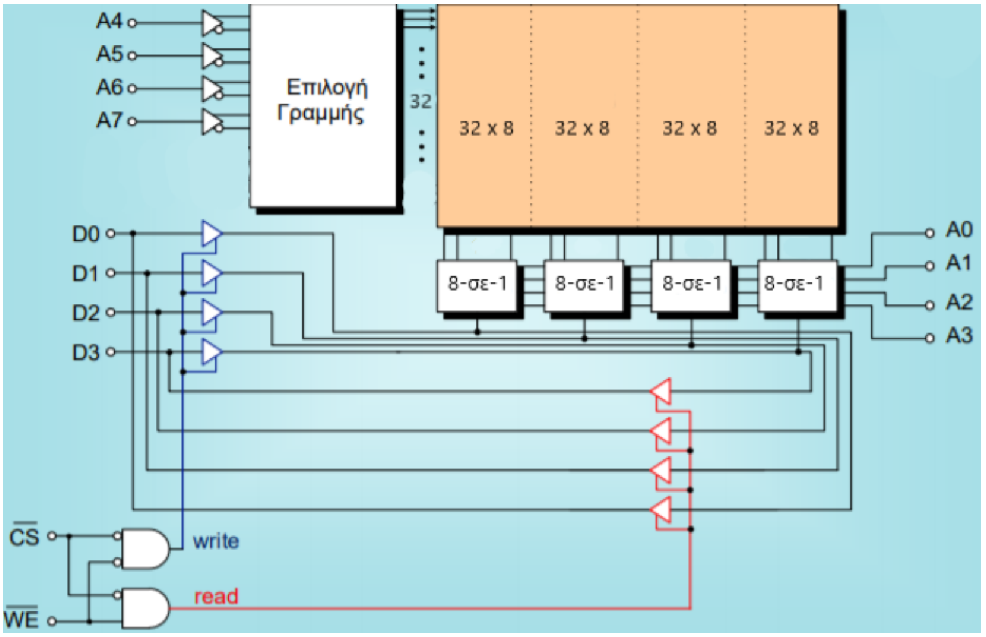
4η Άσκηση

```
START:
      LDA   2000H
      ANI   80H         ;A3 (8th switch)
      RRC               ;move A3 to 7th switch
      MOV   B,A         ;save A3(7th) -> B
      LDA   2000H
      ANI   40H         ;B3 (7th switch)
      ANA   B           ;A3 and B3 -> A
      RRC               ;(6th)
      RRC               ;(5th)
      RRC               ;(4th)
      RRC               ;(3rd)
      MOV   C,A         ;save (A3 and B3)
      RLC               ;(4th)
      MOV   D,A         ;save A(4th)

      LDA   2000H
      ANI   20H         ;A2 (6th)
      RRC               ;5th
      MOV   B,A         ;save A2(5th) -> B
      LDA   2000H
      ANI   10H         ;B2 (5th)
      ANA   B           ;B2 and A2 -> A
      RRC               ;(4th)
      RRC               ;(3rd)
      ORA   C           ;C(A3 and B3) or A(B2 and A2)
      ORA   D           ;[(A3 and B3) or (A2 and B2)](3rd) or D(4th)
      MOV   D,A         ;save A(3rd and 4th)

      LDA   2000H
      ANI   08H         ;A1 (4th)
      RRC               ;(3rd)
      MOV   B,A         ;save A1(3rd) -> B
```

```
LDA   2000H
ANI   04H         ;B1(3rd)
XRA   B           ;A1 xor B1
RRC               ;(2nd)
RRC               ;(1st)
MOV   C,A         ;save A1 xor B1
RLC               ;(2nd)
ORA   D           ;D(3rd, 4th) or (A1 xor B1) (2nd)
MOV   D,A

LDA   2000H
ANI   02H         ;A0 (2nd)
RRC               ;A0 (1st)
MOV   B,A         ;save A0(1st) -> B
LDA   2000H
ANI   01H         ;B0 (1st)
XRA   B           ;A0 xor B0
XRA   C           ;(A0 xor B0) xor C
ORA   D
CMA
STA   3000H
JMP   START
END
```

5η Άσκηση

Παρακάτω παρατίθεται αντίστοιχο σχήμα του σχήματος 3.2 (βιβλίο θεωρίας) που αναπαριστά την εσωτερική οργάνωση μιας μνήμης SRAM 256x4 bit. Η μνήμη αυτή έχει 1024 bits = 2^10 = 2^5 * 2^5 = 32 * 32. Επομένως, χωρίζουμε τις θέσεις μνήμης σε 32 γραμμές των 32 στοιχείων. Εφόσον τα στοιχεία είναι 4 bit χωρίζουμε κάθε γραμμή σε 4 κομμάτια, 8 bits το κάθε ένα.



Η ανάλυση της διαδικασίας εγγραφής και ανάγνωσης πραγματοποιείται παρακάτω μέσω παραδείγματος.

Έστω η διεύθυνση μνήμης 00011011 στην οποία θέλουμε να γράψουμε το στοιχείο 0110 και έπειτα να το διαβάσουμε.

Τα 4 MSB της διεύθυνσης μνήμης (0001) εισάγονται στις εισόδους διεύθυνσης A4 - A7, ενώ τα 4 LSB (1011) στις εισόδους διεύθυνσης A0 - A3, ώστε διαδοχικά στοιχεία να ανήκουν στην ίδια γραμμή. Με βάση τα MSB επιλέγεται η κατάλληλη γραμμή μέσω ενός επιλογέα (4-σε-32). Τα LSB θα προσδιορίζουν τη θέση της λέξης, ενεργοποιώντας την ίδια γραμμή των 4 πολυπλεκτών, εφόσον κάθε bit λέξεις αποθηκεύεται σε 1 απο τα 4 κομμάτια. Ο πολυπλέκτης, σε αντίθεση με τον αποκωδικοποιητή μας επιτρέπει τόσο την εγγραφη, όσο και την ανάγνωση.

Στον διάδρομο των δεδομένων (D0 - D3), εισάγονται τα bits του στοιχείου προς εγγραφή (0110), για την οποία πρέπει να ενεργοποιηθούν οι buffers μπλε χρώματος (προς τα δεξιά), το οποίο επιτυγχάνεται θέτωντας CS = 0, WE = 0. Τα δεδομένα, περνάνε από τους buffers, τους πολυπλέκτες και εγγράφονται στην ενεργοποιημένη γραμμή.

Αντιστοίχως, για την ανάγνωση της συγκεκριμένης θέσης μνήμης, ακολουθούμε τα ίδια βήματα με τη διαφορά ότι θέτουμε WE = 1, ώστε να ενεργοποιηθούν οι buffers κόκκινου χρώματος και αποθηκεύεται το κάθε bit της λέξης που έχει επιλεχθεί από τους πολυπλέκτες στους D0 - D1.

6η Άσκηση

Συνολικά θα χρησιμοποιήσουμε:

ROM: 2K x 8 bit

2K x 8 bit

4K x 8 bit

RAM: 2K x 8 bit

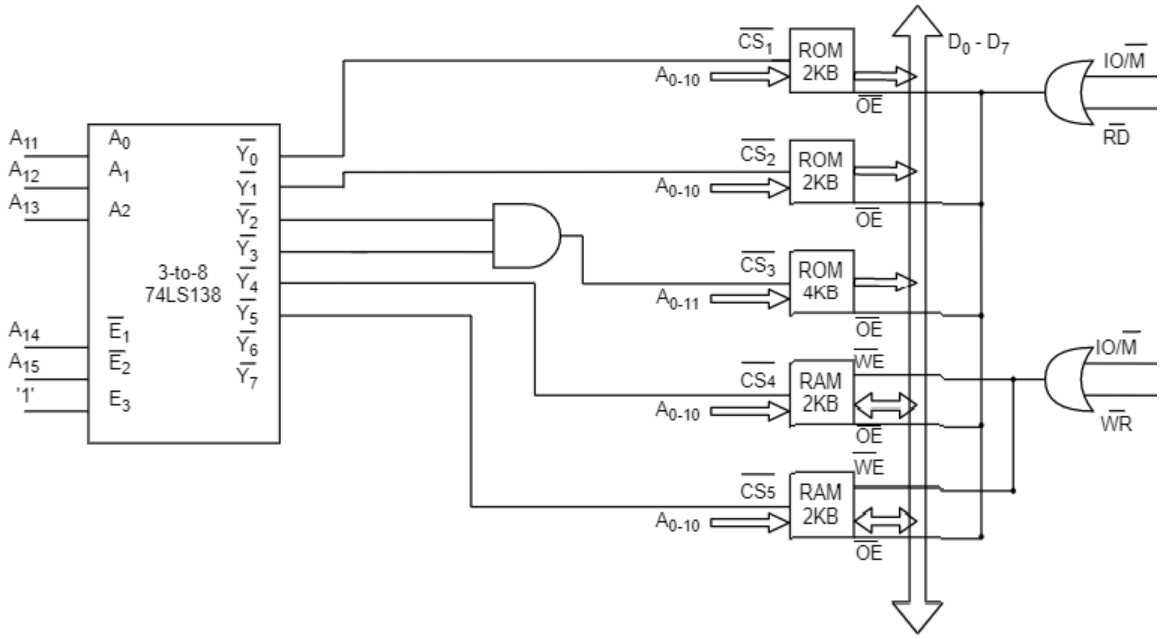
2K x 8 bit

Παρακάτω παρατίθεται ο χάρτης μνήμης.

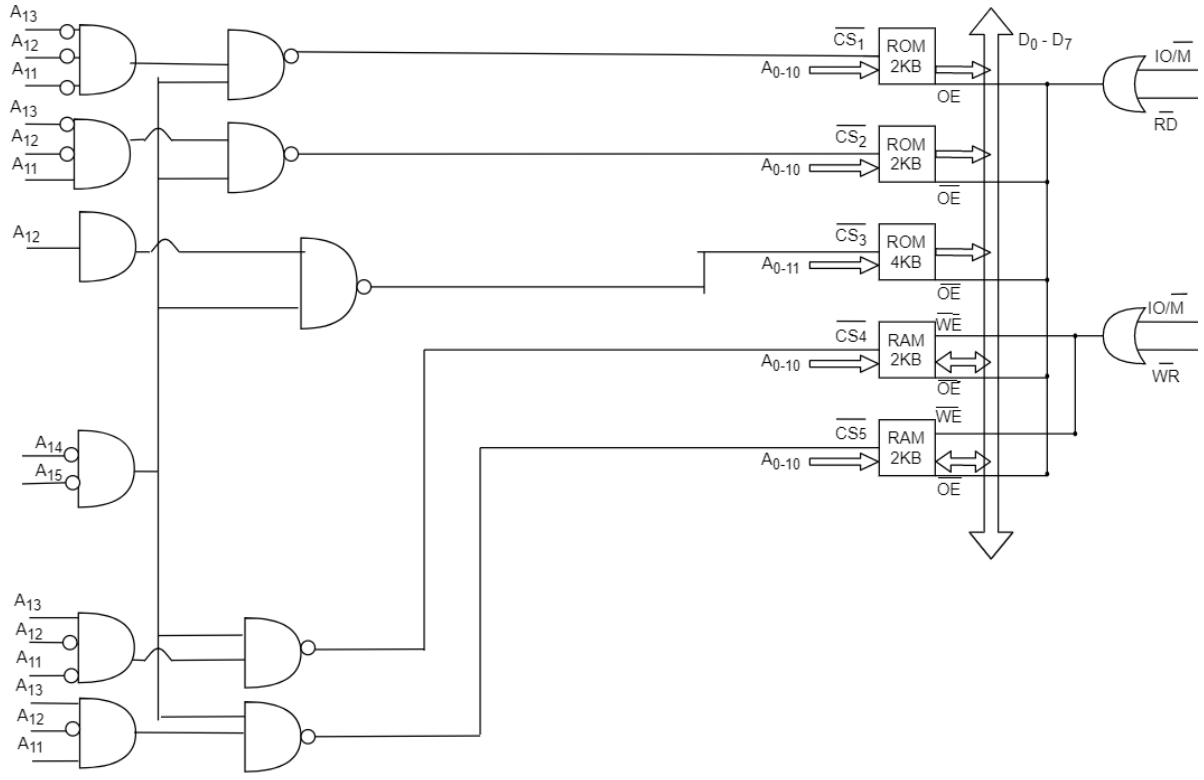
Memory	*Address*	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0000*H*	0	0	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0
	07*FFH*	0	0	**0**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1
ROM	0800*H*	0	0	**0**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0
	0*FFFH*	0	0	**0**	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1
	1000*H*	0	0	**0**	**1**	**0**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

Παρατηρούμε ότι τα bit A_{11}, A_{12}, A_{13} χρησιμοποιούνται για την επιλογή του ολοκληρωμένου.

α) Χρήση αποκωδικοποιητή 3:8 και λογικών πυλών.



β) Με χρήση μόνο λογικών πυλών.



7η Άσκηση

Δημιουργούμε τον παρακάτω χάρτη μνήμης :

Memory	Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM(12K)	0000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2FFFH	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
RAM(4K)	3000H	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	3FFFH	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM(4k)	4000H	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4FFFH	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
RAM(4K)	5000H	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	5FFH	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM(4K)	6000H	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	6FFFH	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1

Τα κυκλώματα διασύνδεσης απεικονίζονται παρακάτω.