Συστήματα Μικροϋπολογιστών

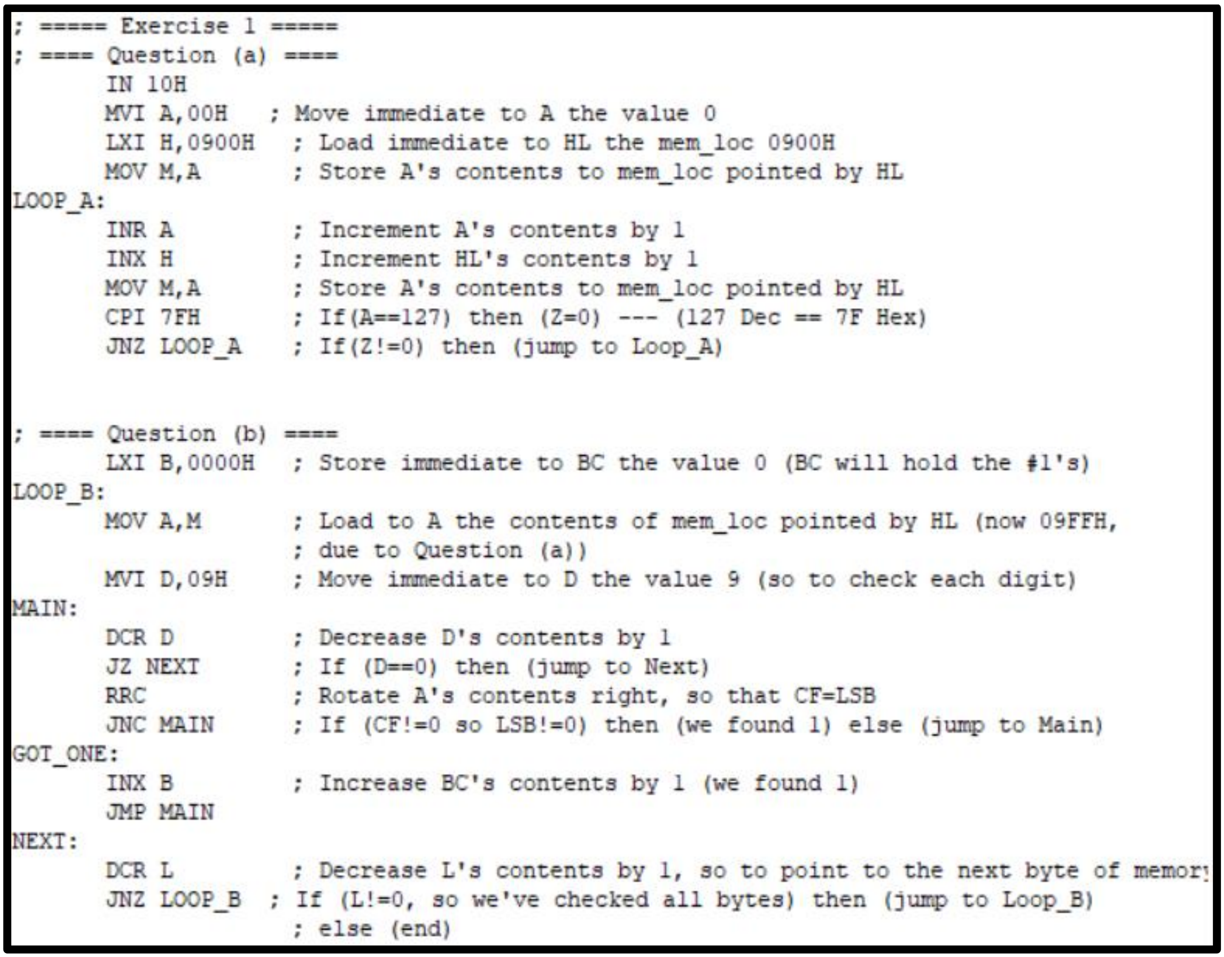
2η Ομάδα Ασκήσεων

Συμμετέχοντες:

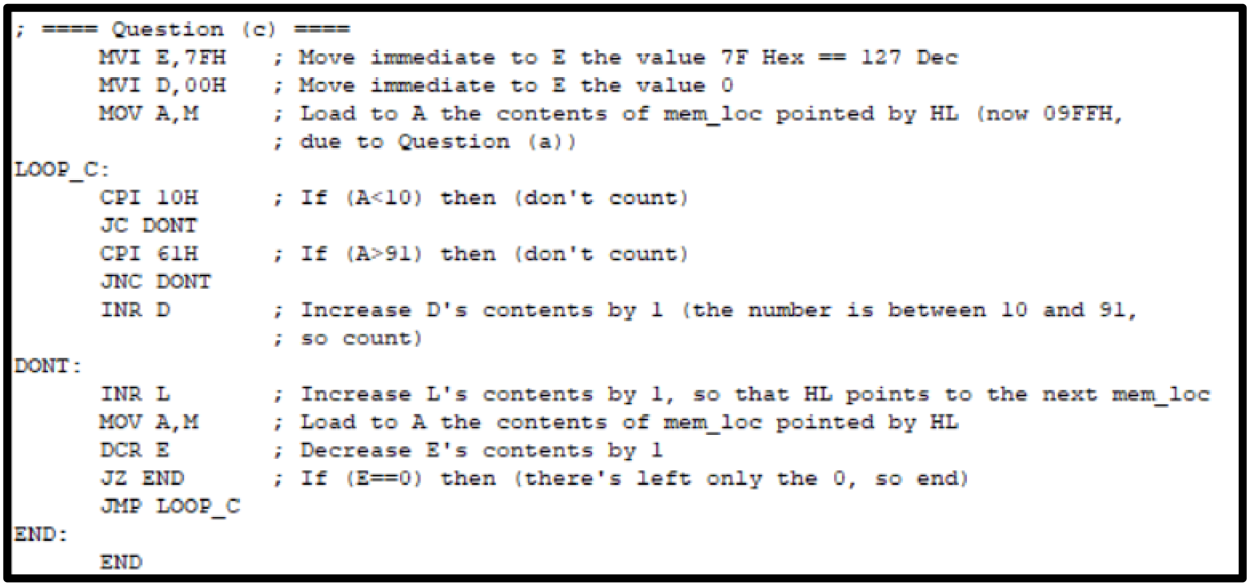
Κωνσταντίνος Σωφρονιάδης, 03120642, (Ασκήσεις: 3,4,6)

Ιωάννης Τσαντήλας, 03120883, (Ασκήσεις: 1,2,5)

1η Άσκηση

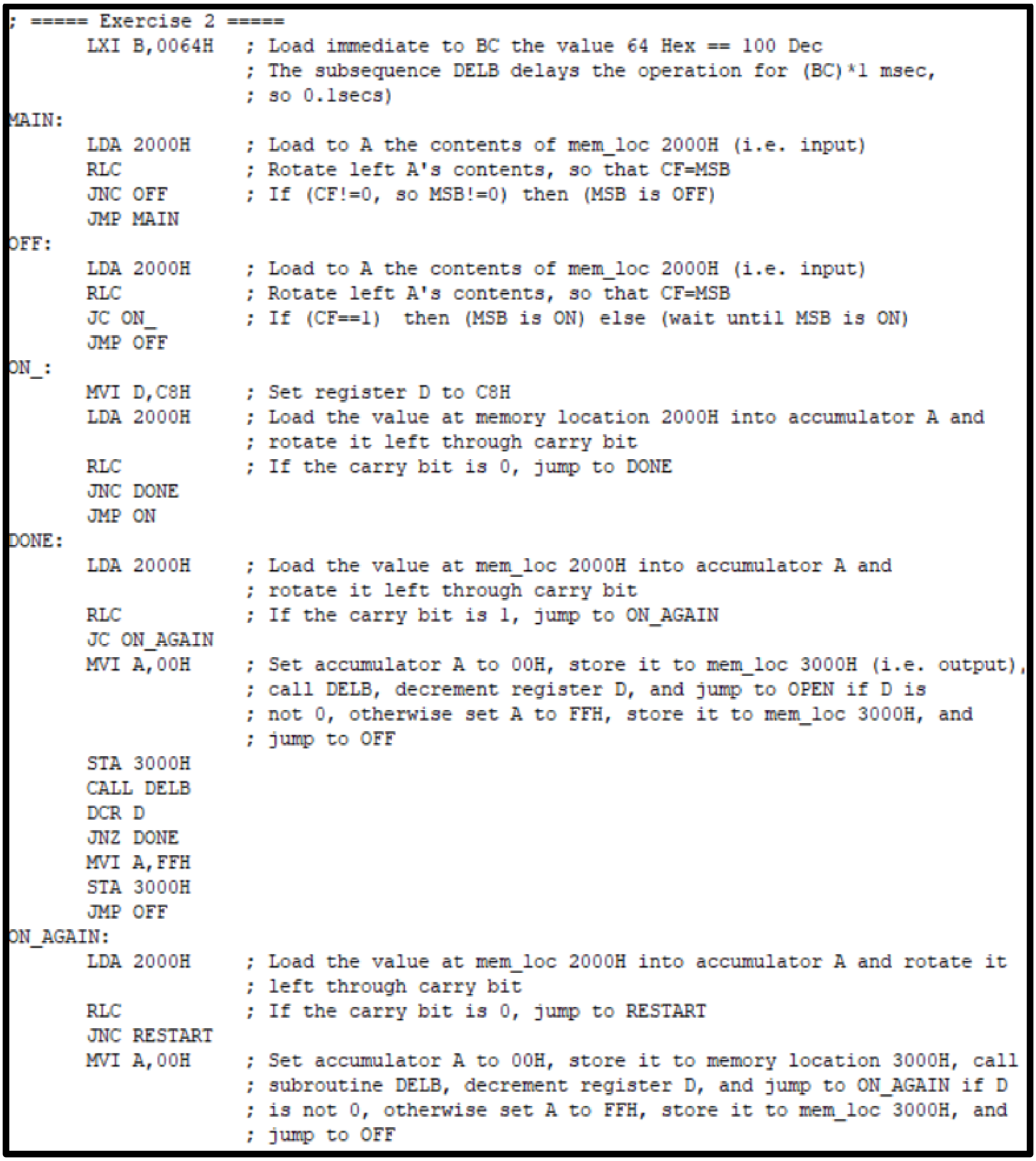


```
; ===== Exercise 1 =====
; ==== Question (a) ====
       IN 10H
       MVI A,00H   ; Move immediate to A the value 0
       LXI H,0900H   ; Load immediate to HL the mem_loc 0900H
       MOV M,A       ; Store A's contents to mem_loc pointed by HL
LOOP_A:
       INR A         ; Increment A's contents by 1
       INX H         ; Increment HL's contents by 1
       MOV M,A       ; Store A's contents to mem_loc pointed by HL
       CPI 7FH       ; If(A==127) then (Z=0) --- (127 Dec == 7F Hex)
       JNZ LOOP_A    ; If(Z!=0) then (jump to Loop_A)


; ==== Question (b) ====
       LXI B,0000H   ; Store immediate to BC the value 0 (BC will hold the #1's)
LOOP_B:
       MOV A,M       ; Load to A the contents of mem_loc pointed by HL (now 09FFH,
                     ; due to Question (a))
       MVI D,09H     ; Move immediate to D the value 9 (so to check each digit)
MAIN:
       DCR D         ; Decrease D's contents by 1
       JZ NEXT       ; If (D==0) then (jump to Next)
       RRC           ; Rotate A's contents right, so that CF=LSB
       JNC MAIN      ; If (CF!=0 so LSB!=0) then (we found 1) else (jump to Main)
GOT_ONE:
       INX B         ; Increase BC's contents by 1 (we found 1)
       JMP MAIN
NEXT:
       DCR L         ; Decrease L's contents by 1, so to point to the next byte of memory
       JNZ LOOP_B  ; If (L!=0, so we've checked all bytes) then (jump to Loop_B)
                     ; else (end)
```

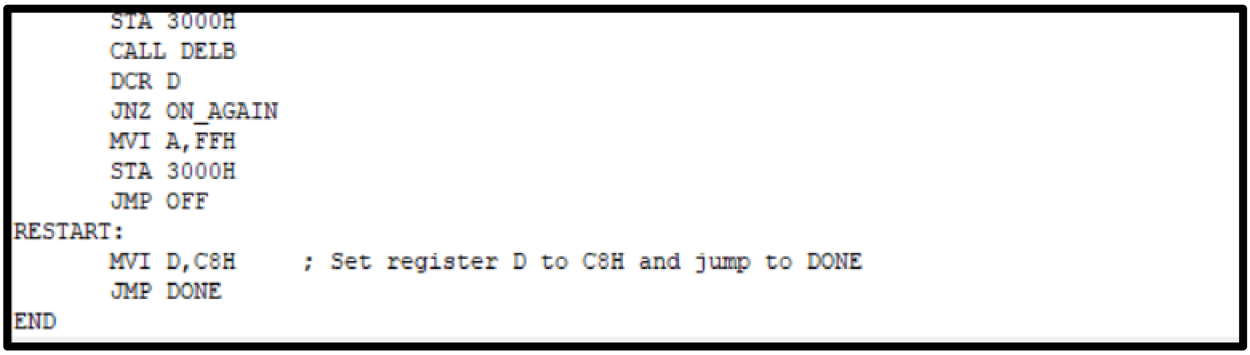


```
; ==== Question (c) ====
      MVI E,7FH    ; Move immediate to E the value 7F Hex == 127 Dec
      MVI D,00H    ; Move immediate to E the value 0
      MOV A,M      ; Load to A the contents of mem_loc pointed by HL (now 09FFH,
                   ; due to Question (a))
LOOP_C:
      CPI 10H      ; If (A<10) then (don't count)
      JC DONT
      CPI 61H      ; If (A>91) then (don't count)
      JNC DONT
      INR D        ; Increase D's contents by 1 (the number is between 10 and 91,
                   ; so count)
DONT:
      INR L        ; Increase L's contents by 1, so that HL points to the next mem_loc
      MOV A,M      ; Load to A the contents of mem_loc pointed by HL
      DCR E        ; Decrease E's contents by 1
      JZ END       ; If (E==0) then (there's left only the 0, so end)
      JMP LOOP_C
END:
      END
```

2η Άσκηση



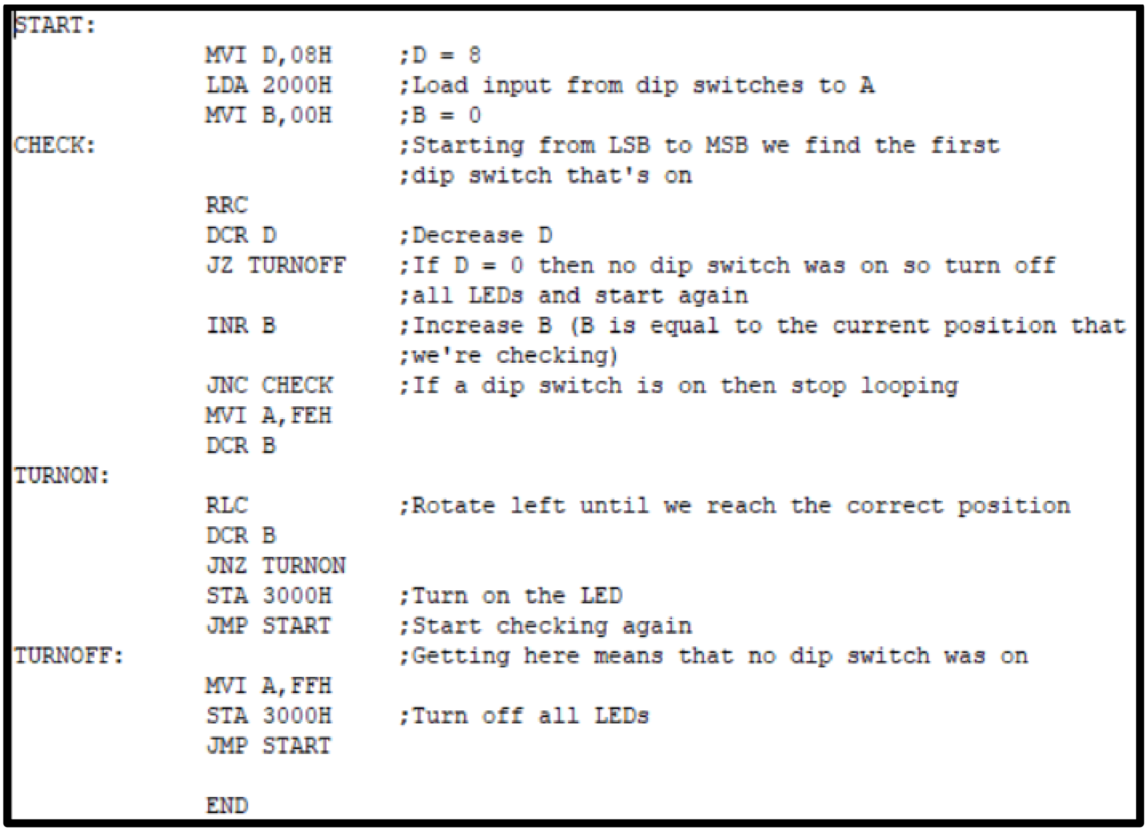
```
; ===== Exercise 2 =====
        LXI B,0064H     ; Load immediate to BC the value 64 Hex == 100 Dec
                        ; The subsequence DELB delays the operation for (BC)*1 msec,
                        ; so 0.1secs)
MAIN:
        LDA 2000H       ; Load to A the contents of mem_loc 2000H (i.e. input)
        RLC             ; Rotate left A's contents, so that CF=MSB
        JNC OFF         ; If (CF!=0, so MSB!=0) then (MSB is OFF)
        JMP MAIN
OFF:
        LDA 2000H       ; Load to A the contents of mem_loc 2000H (i.e. input)
        RLC             ; Rotate left A's contents, so that CF=MSB
        JC ON_          ; If (CF==1)  then (MSB is ON) else (wait until MSB is ON)
        JMP OFF
ON_:
        MVI D,C8H       ; Set register D to C8H
        LDA 2000H       ; Load the value at memory location 2000H into accumulator A and
                        ; rotate it left through carry bit
        RLC             ; If the carry bit is 0, jump to DONE
        JNC DONE
        JMP ON
DONE:
        LDA 2000H       ; Load the value at mem_loc 2000H into accumulator A and
                        ; rotate it left through carry bit
        RLC             ; If the carry bit is 1, jump to ON_AGAIN
        JC ON_AGAIN
        MVI A,00H       ; Set accumulator A to 00H, store it to mem_loc 3000H (i.e. output),
                        ; call DELB, decrement register D, and jump to OPEN if D is
                        ; not 0, otherwise set A to FFH, store it to mem_loc 3000H, and
                        ; jump to OFF
        STA 3000H
        CALL DELB
        DCR D
        JNZ DONE
        MVI A,FFH
        STA 3000H
        JMP OFF
ON_AGAIN:
        LDA 2000H       ; Load the value at mem_loc 2000H into accumulator A and rotate it
                        ; left through carry bit
        RLC             ; If the carry bit is 0, jump to RESTART
        JNC RESTART
        MVI A,00H       ; Set accumulator A to 00H, store it to memory location 3000H, call
                        ; subroutine DELB, decrement register D, and jump to ON_AGAIN if D
                        ; is not 0, otherwise set A to FFH, store it to mem_loc 3000H, and
                        ; jump to OFF
```



```
        STA 3000H
        CALL DELB
        DCR D
        JNZ ON_AGAIN
        MVI A,FFH
        STA 3000H
        JMP OFF
RESTART:
        MVI D,C8H       ; Set register D to C8H and jump to DONE
        JMP DONE
END
```

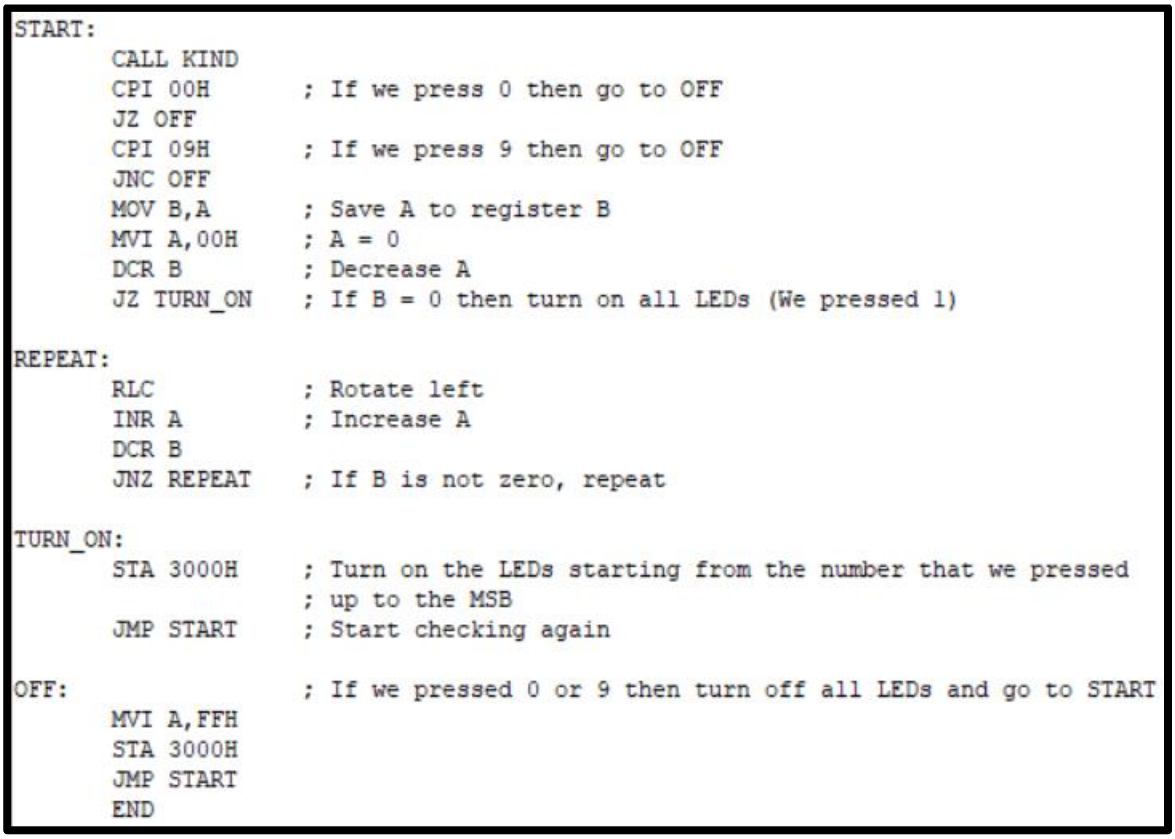
3η Άσκηση

Ερώτημα (i)



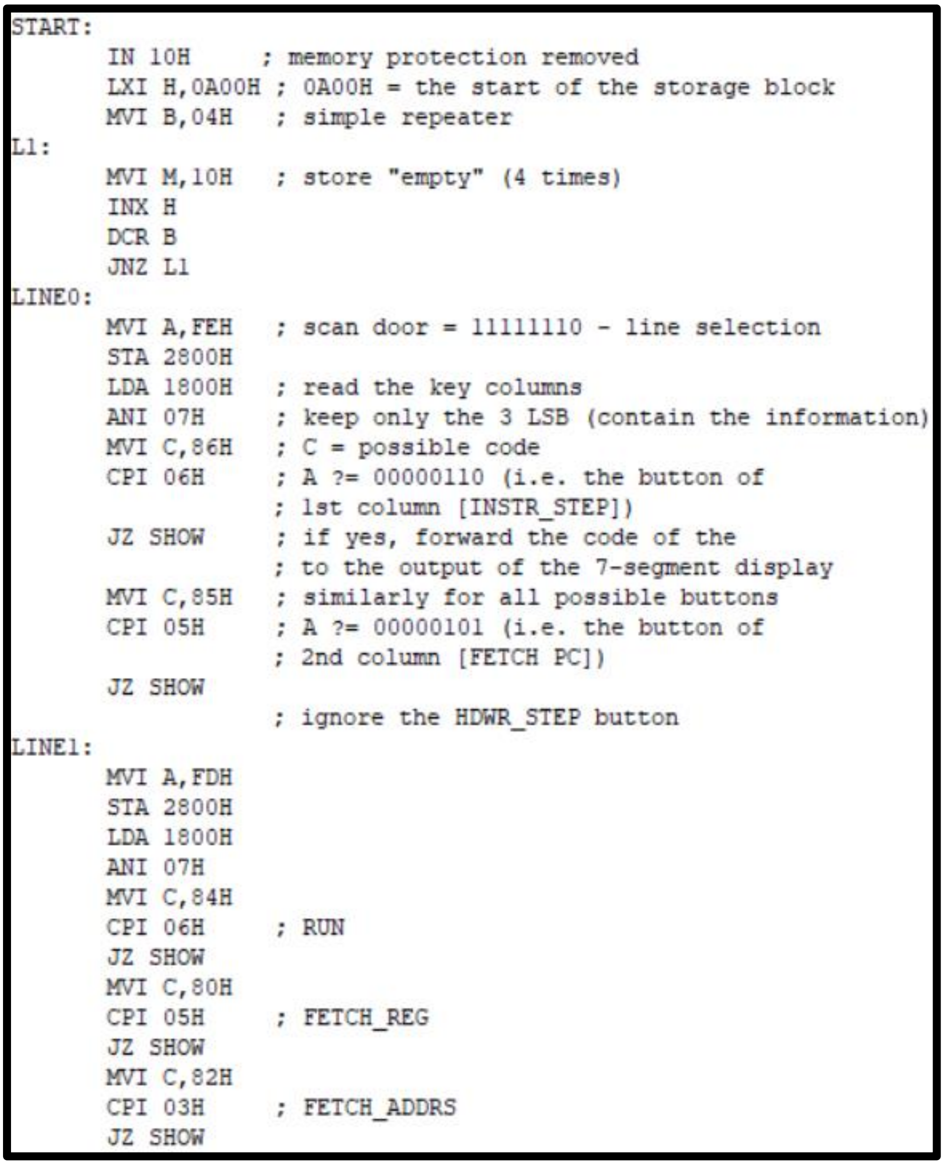
```
START:
             MVI D,08H     ;D = 8
             LDA 2000H     ;Load input from dip switches to A
             MVI B,00H     ;B = 0
CHECK:                     ;Starting from LSB to MSB we find the first
                           ;dip switch that's on
             RRC
             DCR D         ;Decrease D
             JZ TURNOFF    ;If D = 0 then no dip switch was on so turn off
                           ;all LEDs and start again
             INR B         ;Increase B (B is equal to the current position that
                           ;we're checking)
             JNC CHECK     ;If a dip switch is on then stop looping
             MVI A,FEH
             DCR B
TURNON:
             RLC           ;Rotate left until we reach the correct position
             DCR B
             JNZ TURNON
             STA 3000H     ;Turn on the LED
             JMP START     ;Start checking again
TURNOFF:                   ;Getting here means that no dip switch was on
             MVI A,FFH
             STA 3000H     ;Turn off all LEDs
             JMP START

             END
```

Ερώτημα (ii)

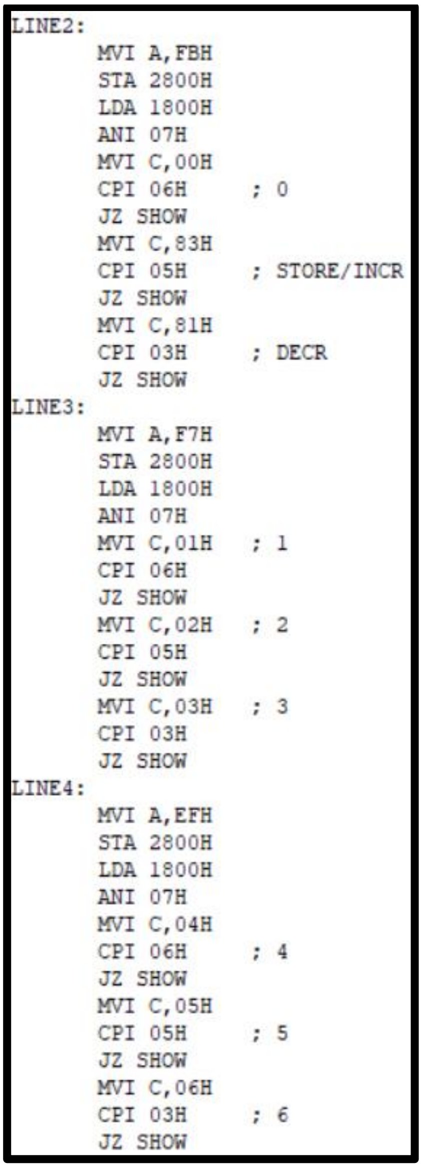


```
START:
       CALL KIND
       CPI 00H       ; If we press 0 then go to OFF
       JZ OFF
       CPI 09H       ; If we press 9 then go to OFF
       JNC OFF
       MOV B,A       ; Save A to register B
       MVI A,00H     ; A = 0
       DCR B         ; Decrease A
       JZ TURN_ON    ; If B = 0 then turn on all LEDs (We pressed 1)

REPEAT:
       RLC           ; Rotate left
       INR A         ; Increase A
       DCR B
       JNZ REPEAT    ; If B is not zero, repeat

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

OFF:                 ; If we pressed 0 or 9 then turn off all LEDs and go to START
       MVI A,FFH
       STA 3000H
       JMP START
       END
```

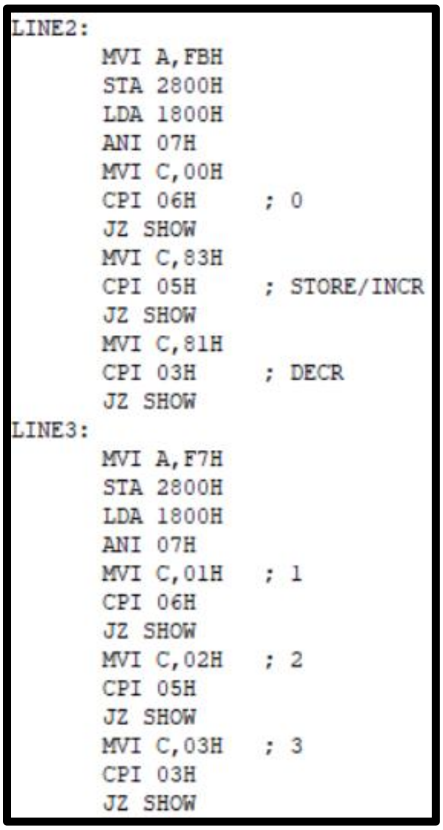
Ερώτημα (iii)



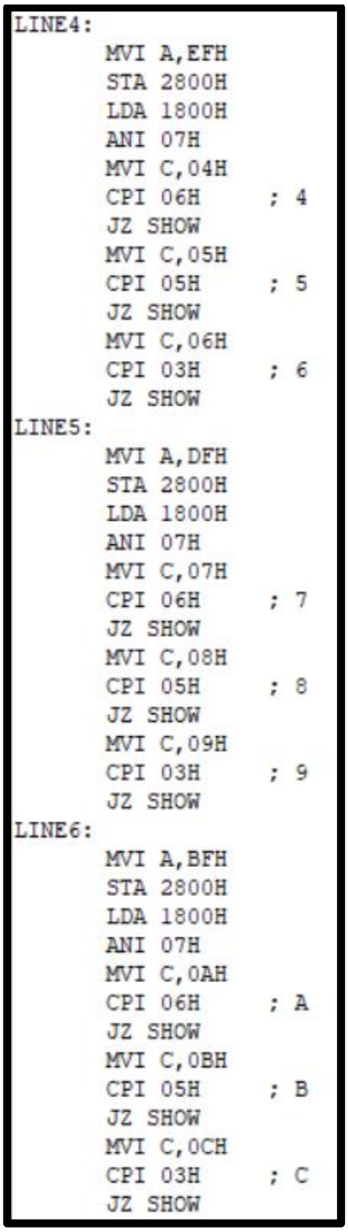
```
START:
        IN 10H     ; memory protection removed
        LXI H,0A00H ; 0A00H = the start of the storage block
        MVI B,04H   ; simple repeater
L1:
        MVI M,10H   ; store "empty" (4 times)
        INX H
        DCR B
        JNZ L1
LINE0:
        MVI A,FEH   ; scan door = 11111110 - line selection
        STA 2800H
        LDA 1800H   ; read the key columns
        ANI 07H     ; keep only the 3 LSB (contain the information)
        MVI C,86H   ; C = possible code
        CPI 06H     ; A ?= 00000110 (i.e. the button of
                    ; 1st column [INSTR_STEP])
        JZ SHOW     ; if yes, forward the code of the
                    ; to the output of the 7-segment display
        MVI C,85H   ; similarly for all possible buttons
        CPI 05H     ; A ?= 00000101 (i.e. the button of
                    ; 2nd column [FETCH PC])
        JZ SHOW
                    ; ignore the HDWR_STEP button
LINE1:
        MVI A,FDH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,84H
        CPI 06H     ; RUN
        JZ SHOW
        MVI C,80H
        CPI 05H     ; FETCH_REG
        JZ SHOW
        MVI C,82H
        CPI 03H     ; FETCH_ADDRS
        JZ SHOW
```



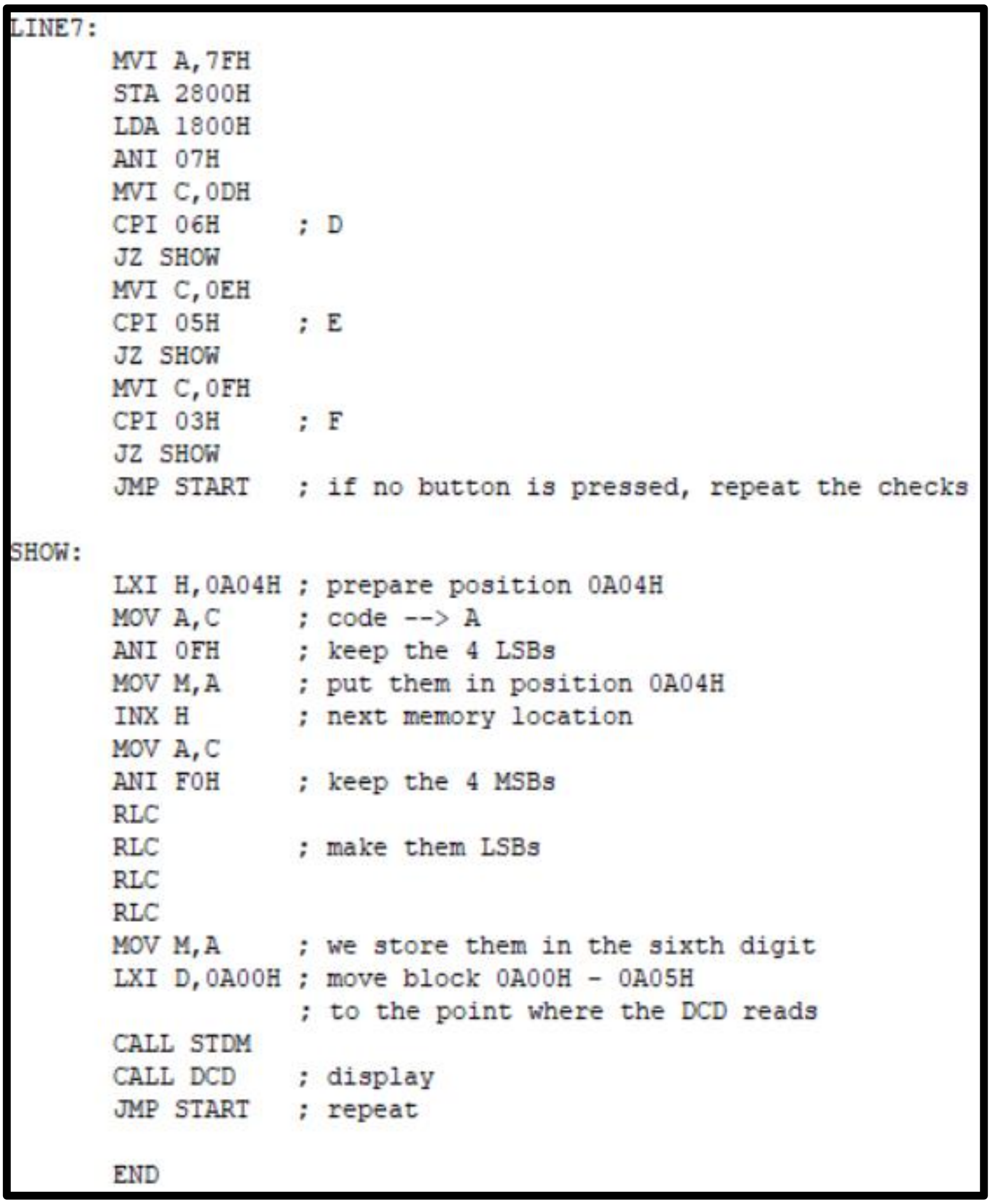
```
LINE2:
        MVI A,FBH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,00H
        CPI 06H     ; 0
        JZ SHOW
        MVI C,83H
        CPI 05H     ; STORE/INCR
        JZ SHOW
        MVI C,81H
        CPI 03H     ; DECR
        JZ SHOW
LINE3:
        MVI A,F7H
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,01H   ; 1
        CPI 06H
        JZ SHOW
        MVI C,02H   ; 2
        CPI 05H
        JZ SHOW
        MVI C,03H   ; 3
        CPI 03H
        JZ SHOW
LINE4:
        MVI A,EFH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,04H
        CPI 06H     ; 4
        JZ SHOW
        MVI C,05H
        CPI 05H     ; 5
        JZ SHOW
        MVI C,06H
        CPI 03H     ; 6
        JZ SHOW
```



```
LINE2:
        MVI A,FBH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,00H
        CPI 06H     ; 0
        JZ SHOW
        MVI C,83H
        CPI 05H     ; STORE/INCR
        JZ SHOW
        MVI C,81H
        CPI 03H     ; DECR
        JZ SHOW
LINE3:
        MVI A,F7H
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,01H   ; 1
        CPI 06H
        JZ SHOW
        MVI C,02H   ; 2
        CPI 05H
        JZ SHOW
        MVI C,03H   ; 3
        CPI 03H
        JZ SHOW
```

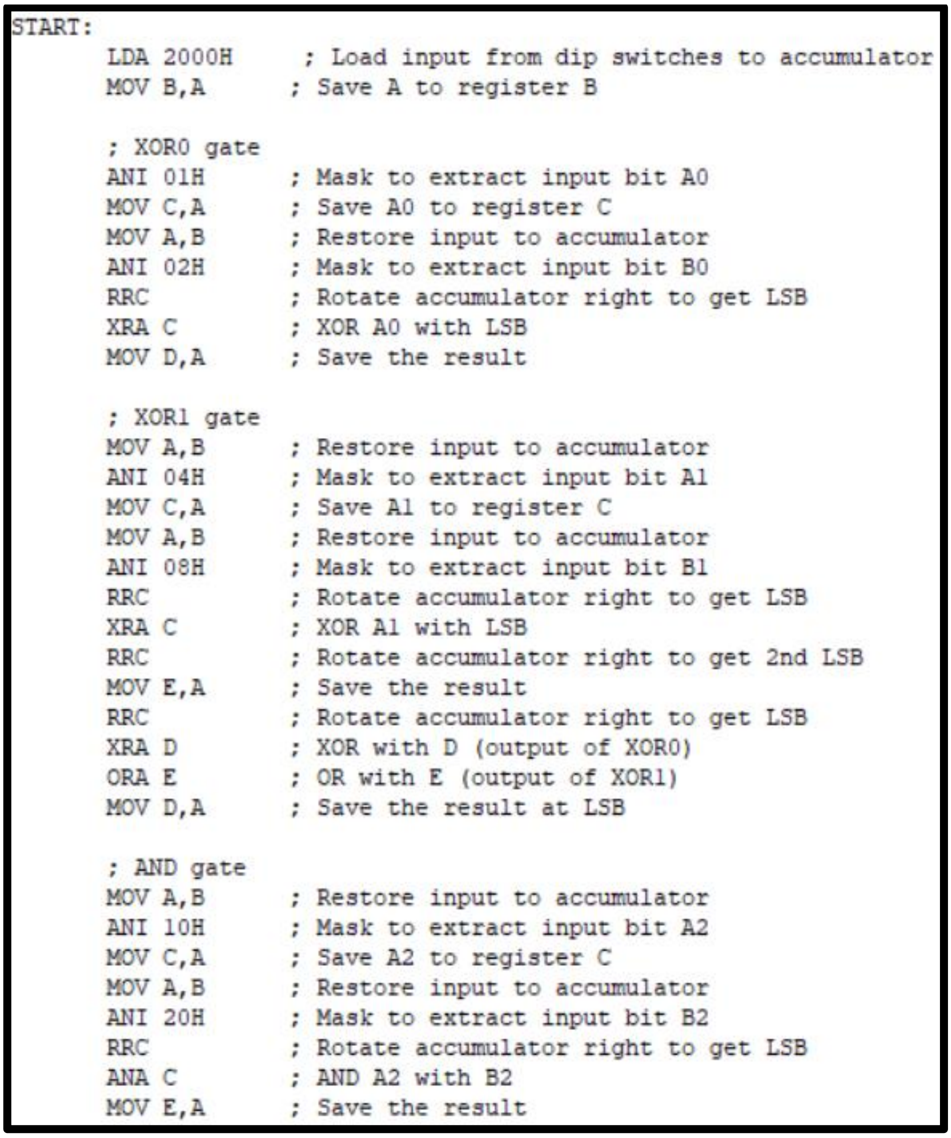


```
LINE4:
        MVI A,EFH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,04H
        CPI 06H     ; 4
        JZ SHOW
        MVI C,05H
        CPI 05H     ; 5
        JZ SHOW
        MVI C,06H
        CPI 03H     ; 6
        JZ SHOW
LINE5:
        MVI A,DFH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,07H
        CPI 06H     ; 7
        JZ SHOW
        MVI C,08H
        CPI 05H     ; 8
        JZ SHOW
        MVI C,09H
        CPI 03H     ; 9
        JZ SHOW
LINE6:
        MVI A,BFH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,0AH
        CPI 06H     ; A
        JZ SHOW
        MVI C,0BH
        CPI 05H     ; B
        JZ SHOW
        MVI C,0CH
        CPI 03H     ; C
        JZ SHOW
```

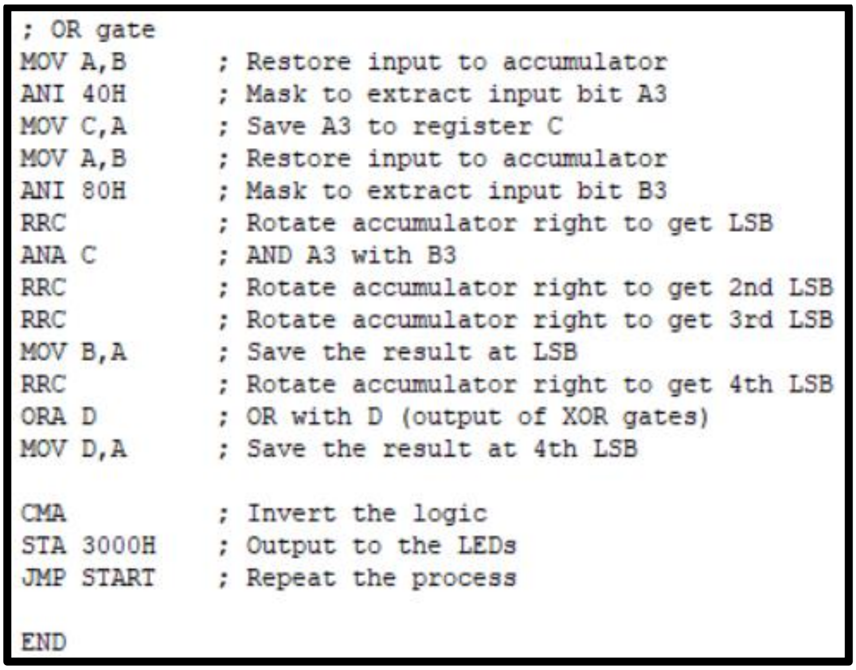


```
LINE7:
        MVI A,7FH
        STA 2800H
        LDA 1800H
        ANI 07H
        MVI C,0DH
        CPI 06H     ; D
        JZ SHOW
        MVI C,0EH
        CPI 05H     ; E
        JZ SHOW
        MVI C,0FH
        CPI 03H     ; F
        JZ SHOW
        JMP START   ; if no button is pressed, repeat the checks

SHOW:
        LXI H,0A04H ; prepare position 0A04H
        MOV A,C     ; code --> A
        ANI 0FH     ; keep the 4 LSBs
        MOV M,A     ; put them in position 0A04H
        INX H       ; next memory location
        MOV A,C
        ANI F0H     ; keep the 4 MSBs
        RLC
        RLC         ; make them LSBs
        RLC
        RLC
        MOV M,A     ; we store them in the sixth digit
        LXI D,0A00H ; move block 0A00H - 0A05H
                    ; to the point where the DCD reads
        CALL STDM
        CALL DCD    ; display
        JMP START   ; repeat

        END
```

4η Άσκηση



```
START:
      LDA 2000H     ; Load input from dip switches to accumulator
      MOV B,A      ; Save A to register B

      ; XOR0 gate
      ANI 01H      ; Mask to extract input bit A0
      MOV C,A      ; Save A0 to register C
      MOV A,B      ; Restore input to accumulator
      ANI 02H      ; Mask to extract input bit B0
      RRC          ; Rotate accumulator right to get LSB
      XRA C        ; XOR A0 with LSB
      MOV D,A      ; Save the result

      ; XOR1 gate
      MOV A,B      ; Restore input to accumulator
      ANI 04H      ; Mask to extract input bit A1
      MOV C,A      ; Save A1 to register C
      MOV A,B      ; Restore input to accumulator
      ANI 08H      ; Mask to extract input bit B1
      RRC          ; Rotate accumulator right to get LSB
      XRA C        ; XOR A1 with LSB
      RRC          ; Rotate accumulator right to get 2nd LSB
      MOV E,A      ; Save the result
      RRC          ; Rotate accumulator right to get LSB
      XRA D        ; XOR with D (output of XOR0)
      ORA E        ; OR with E (output of XOR1)
      MOV D,A      ; Save the result at LSB

      ; AND gate
      MOV A,B      ; Restore input to accumulator
      ANI 10H      ; Mask to extract input bit A2
      MOV C,A      ; Save A2 to register C
      MOV A,B      ; Restore input to accumulator
      ANI 20H      ; Mask to extract input bit B2
      RRC          ; Rotate accumulator right to get LSB
      ANA C        ; AND A2 with B2
      MOV E,A      ; Save the result
```

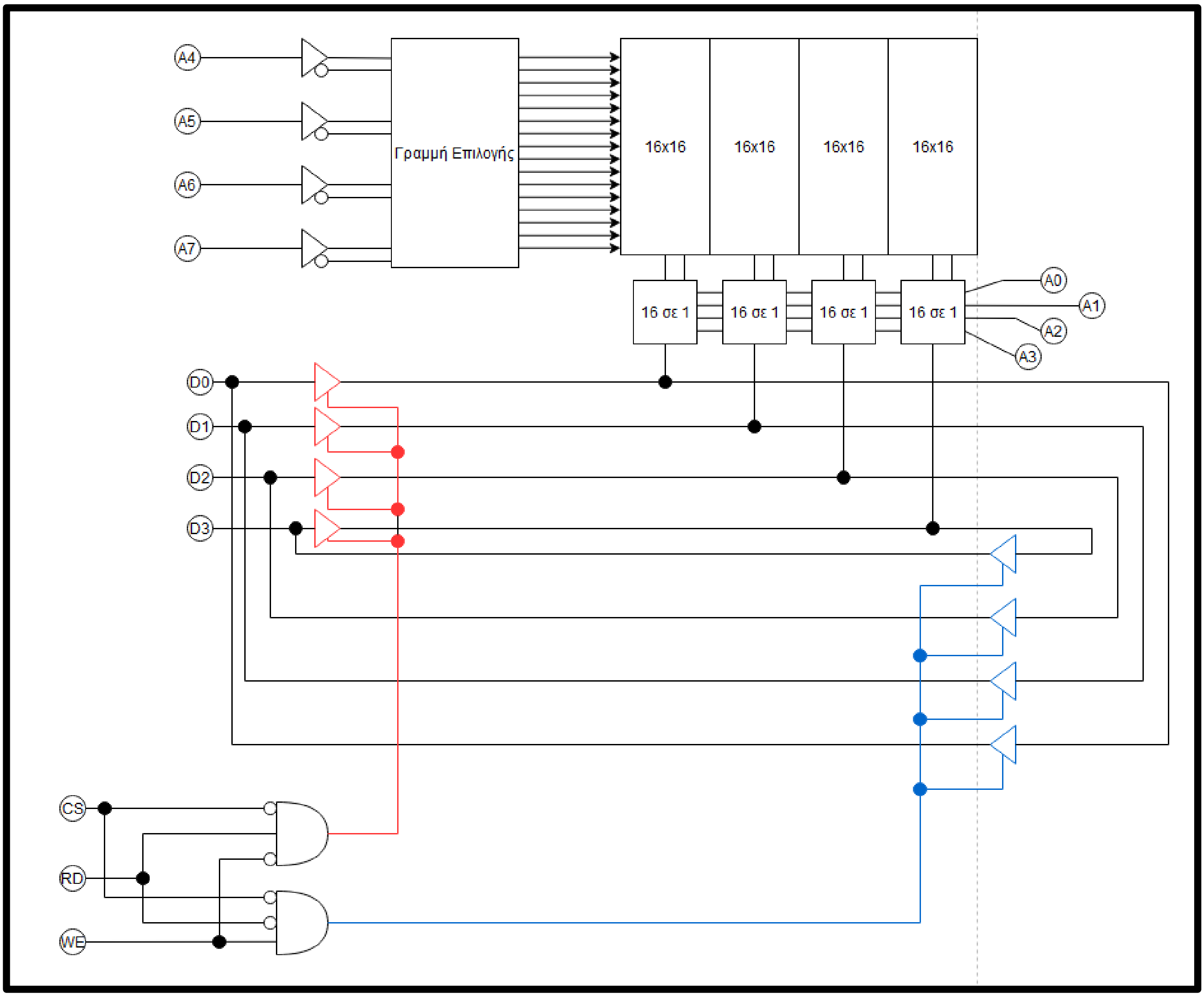


```
; OR gate
MOV A,B      ; Restore input to accumulator
ANI 40H      ; Mask to extract input bit A3
MOV C,A      ; Save A3 to register C
MOV A,B      ; Restore input to accumulator
ANI 80H      ; Mask to extract input bit B3
RRC          ; Rotate accumulator right to get LSB
ANA C        ; AND A3 with B3
RRC          ; Rotate accumulator right to get 2nd LSB
RRC          ; Rotate accumulator right to get 3rd LSB
MOV B,A      ; Save the result at LSB
RRC          ; Rotate accumulator right to get 4th LSB
ORA D        ; OR with D (output of XOR gates)
MOV D,A      ; Save the result at 4th LSB

CMA          ; Invert the logic
STA 3000H    ; Output to the LEDs
JMP START    ; Repeat the process

END
```

$5^{η}$ Άσκηση

Παραθέτουμε την δομή της **SRAM**, όπως και μία ενδεικτική εικόνα της:

- **A4-A7:** γραμμές **διεύθυνσης** (address), βάσει των οποίων επιλέγονται μία από τις δεκαέξι γραμμές του πίνακα μνήμης.
- **D0-D3:** γραμμές **δεδομένων** (data), οι οποίες συνδέονται με τον πίνακα της μνήμης μέσω τεσσάρων πολυπλεκτών 16 σε 1.
- **A0-A3:** γραμμές **διεύθυνσης** (address), βάσει των οποίων ο καθένας από τους προαναφερθέντες πολυπλέκτες επιλέγει μία από τις δεκαέξι τετράδες-στήλες του πίνακα μνήμης.
- **D0-D3:** γραμμές **δεδομένων** (data), τα οποία γράφονται ή διαβάζονται από τις προαναφερθέντες τετράδες/στήλες σε συνδυασμό με την αντίστοιχη γραμμή του πίνακα διευθύνσεων.



Η ανάγνωση και η εγγραφή ελέγχονται από τα σήματα **CS**, **RD**, **WE**:

- **CS=0:** Ενεργοποίηση λειτουργίας μνήμης
- **WE=0, RD=1:** Ενεργοποιούνται οι **κόκκινοι** απομονωτές, υπεύθυνοι για την εγγραφή.
- **WE=1, RD=0:** Ενεργοποιούνται οι **μπλε** απομονωτές, υπεύθυνοι για την ανάγνωση.

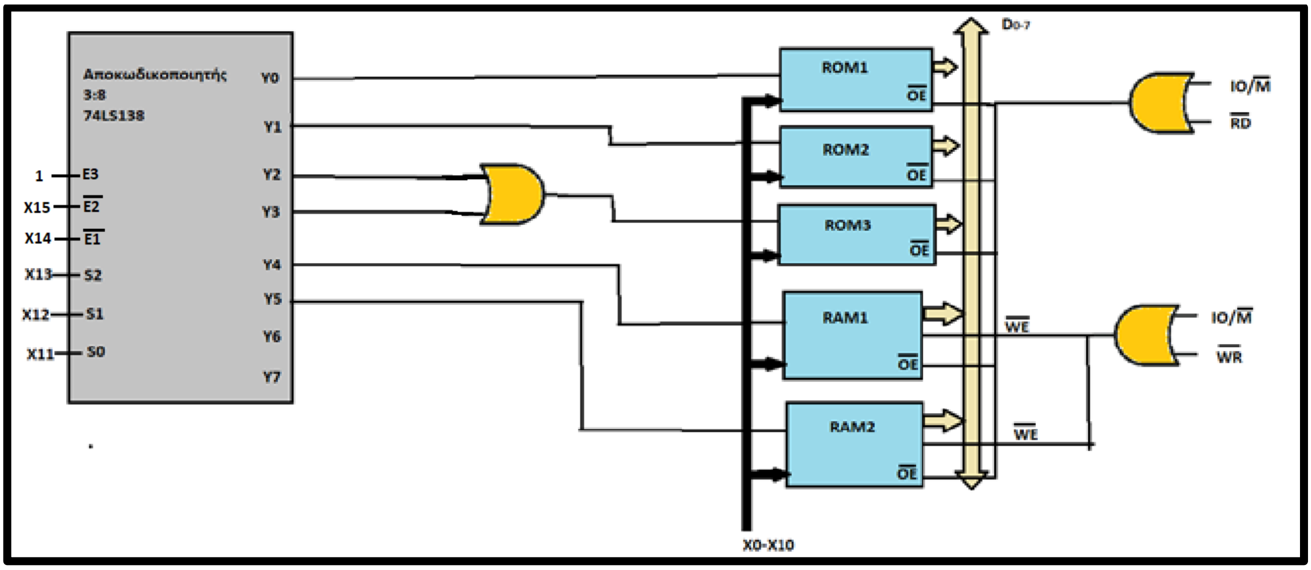
6η Άσκηση

Έχουμε τον χάρτη μνήμης:

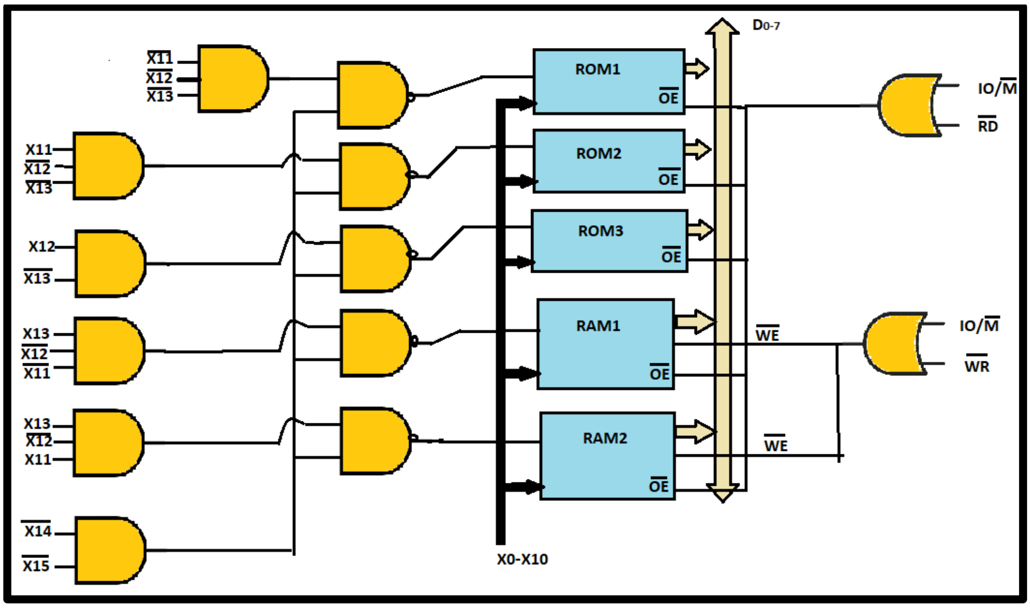
Μνήμη	Διεύθυνση	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ROM1 2K	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2K-1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
ROM2 2K	2K	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	4K-1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
ROM3 4K	4K	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	8K-1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM1 2K	8K	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	10K-1	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1
RAM2 2K	10K	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	12K-1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

Θα περάσουμε από αποκωδικοποιητή τις τιμές των ψηφίων 13, 12, 11 για να αποφανθούμε σε ποιο ολοκληρωμένο μνήμης θα πάει η κάθε διεύθυνση.

a) Με ένα αποκωδικοποιητή 3:8 (74LS138) και λογικές πύλες έχουμε:



b) Με μόνο λογικές πύλες έχουμε:


$\overline{X11}$
$\overline{X12}$
$\overline{X13}$
X11
$\overline{X12}$
$\overline{X13}$
X12
$\overline{X13}$
X13
$\overline{X12}$
$\overline{X11}$
X13
$\overline{X12}$
X11
$\overline{X14}$
$\overline{X15}$
D0-7
ROM1
$\overline{OE}$
ROM2
$\overline{OE}$
ROM3
$\overline{OE}$
RAM1
$\overline{OE}$
RAM2
$\overline{OE}$
$\overline{WE}$
$\overline{WE}$
IO/$\overline{M}$
$\overline{RD}$
IO/$\overline{M}$
$\overline{WR}$
X0-X10