**Εθνικό Μετσόβιο Πολυτεχνείο**

**Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών**

**2η Ομάδα Ασκήσεων**

**Μάθημα:** Συστήματα Μικροϋπολογιστών

**Εξάμηνο:** 6ο

**Ονοματεπώνυμα:** Αλεξοπούλου Γεωργία (ΑΜ: 03120164), Γκενάκου Ζωή (ΑΜ: 03120015)

***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 (= , 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.

Στη συνέχεια τοποθετούμε τα δεδομένα στη RAM και λαμβάνουμε τα αποτελέσματα της παρακάτω φωτογραφίας:

Η ΜΑΛΑΚΙΑ ΔΕΝ ΜΟΥ ΔΕΙΧΝΕΙ ΣΩΣΤΑ ΤΗ ΡΑΜ

**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:

**γ)** Ο κώδικας για τον υπολογισμό των αριθμών 0-127 που βρίσκονται μεταξύ και συμπεριλαμβανομένων των αριθμών 10Η και 60Η 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:

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 ΜSB 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 ΟΝ, 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 |

1. 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 |

1. 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***

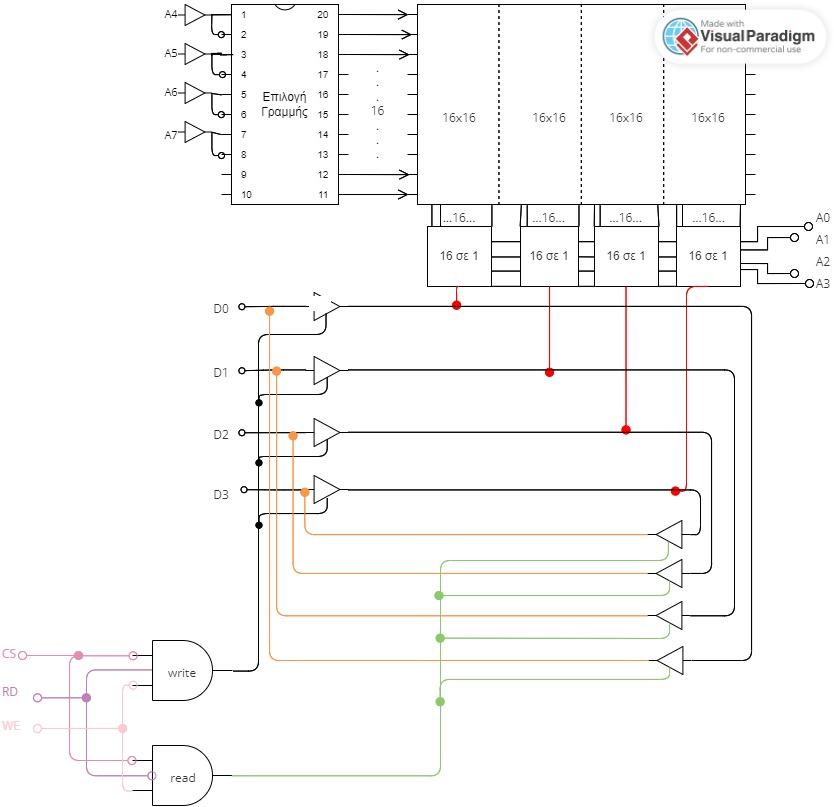
Το πρόγραμμα που περιγράφει το λογικό σχήμα της εκφώνησης φαίνεται παρακάτω και είναι αποθηκευμένο στο αρχείο *‘Άσκηση 4.8085’*:

|  |
| --- |
| 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 = A  MOV A,B ;A is equal to the input of dip switches  ANI 02H ;A = A AND 00000010  XOR0:  RRC ;Rotate right to get the output at LSB  XRA C ;A XOR C  MOV D,A ;Save the answer  A1\_B1:  MOV A,B  ANI 04H ;A = A AND 00000100  MOV C,A ;C = A  MOV A,B  ANI 08H ;A = A AND 00001000  XOR1:  RRC  XRA C ;A XOR C  RRC ;Rotate right to get X1 at 2nd LSB  MOV E,A ;E = A XOR C (output of XOR1)  RRC ;Rotate right to get X0 at LSB  XRA D ;(A XOR C) XOR D  ORA E ;Save X1 at 2nd LSB  MOV D,A ;Save X0 at LSB  A2\_B2:  MOV A,B   ANI 10H ;A = A AND 00010000  MOV C,A ;C = A  MOV A,B  ANI 20H ;A = A AND 00100000  RRC  AND:  ANA C ;A AND C  MOV E,A ;E = A AND C (output of A2\_B2)  A3\_B3:  MOV A,B  ANI 40H ;A = A AND 01000000  MOV C,A ;C = A  MOV A,B  ANI 80H ;A = A AND 10000000  RRC  AND1:  ANA C ;A AND C (output of A3\_B3)  RRC  RRC  MOV B,A  RRC  ORA D  MOV D,A ;Save X3 at 4th LSB  OR:  MOV A,B  ORA E ;(A AND C(output of A3\_B3)) OR E  RRC  RRC  ORA D ;Save X2 at 3rd LSB  LED:  CMA ;Inverse logic  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 Α4-Α7. 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 (Α0-Α3) είτε καταχωρούνται τα δεδομένα των γραμμών δεδομένων D0-D3, είτε το αντίστροφο.

Έτσι, αν η διεύθυνση Α0Α1Α2Α3 Α4Α5Α6Α7 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***

Αρχικά θα κατασκευάσουμε τον χάρτη μνήμης. Γνωρίζουμε πως η ROM ξεκινά από τη θέση 0000Η και υλοποιείται με δύο 2Κx8bit ROMs και μία 4Κx8bit ROM. Η RAM υλοποιείται με δύο 2Κx8bit ICs. Συνεπώς:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
|  | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ROM#2  - 2K |  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ROM #3 - 4K |  | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| RAM #1 - 2K |  | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| RAM #2 - 2K |  | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

***Άσκηση 7***

Ζωή