Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

3η Ομάδα Ασκήσεων

Μάθημα: Συστήματα Μικροϋπολογιστών

Εξάμηνο: 6ο

Ονοματεπώνυμο: Αλεξοπούλου Γεωργία (ΑΜ: 03120164), Γκενάκου Ζωή (ΑΜ: 03120015)

Άσκηση 1

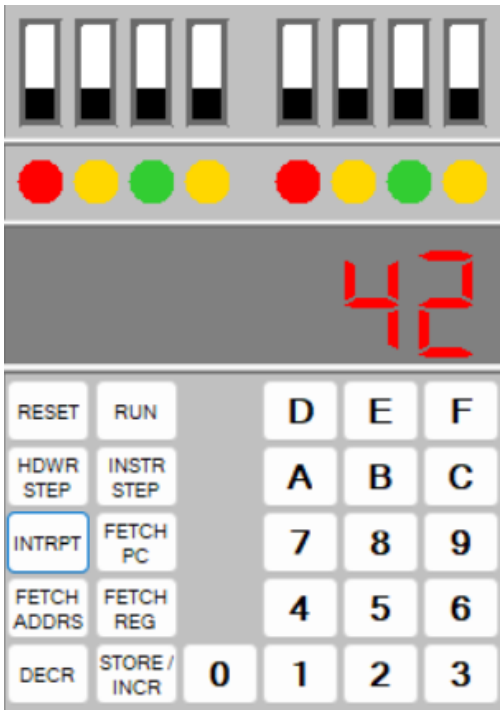
α) Ο ζητούμενος κώδικας είναι ο εξής (*'Askisi1.8085'*):

```
LXI B,0032H ;used for 0.05s delay
MVI A,10H ;4 left-most 7-segment displays need to be empty
STA 0B02H
STA 0B03H
STA 0B04H
STA 0B05H

START:
     MVI A,0DH ;interrupt mask to allow rst 6.5
     SIM
     EI ;enable interrupts

WAIT: JMP WAIT

INTR_ROUTINE:
     POP H ;reduce stack
     MVI A,00H
```

```
      STA 3000H
      MVI D,3CH ;60 seconds
      MVI E,00H ;E will count up to 5 to reverse lights every 5*0.05
= 0.25s
      MVI H,00H ;H will count up to 4 light reverses to reduce D by 1
(1s)
      MVI L,FFH ;L will contain current light condition (00 or FF)
      EI ;enable interrupts to allow timer renewing

LIGHTS:
      INR E
      MOV A,E
      CPI 05H
      JNZ SKIP

      MOV A,L
      CMA
      MOV L,A
      STA 3000H
      INR H ;increase H every 5 loops
      MVI E,00H
      MOV A,H
      CPI 04H
      JNZ SKIP

      DCR D ;1 second has passed
      MVI H,00H
      MOV A,D
      CPI 60H ;60 seconds have passed if D is 60
      JZ START

SKIP:
      CALL DELB
      PUSH H ;store HL
      MOV A,D ;seconds remaining
      MVI B,00H

      ;bring number in decimal form (A contains ones and B contains
tens)
```

```
      SUB10:
      CPI 0AH
      JC FINISH
      SUI 0AH
      INR B
      JMP SUB10

FINISH:
      STA 0B00H ;print ones in right-most 7-segment display

      MOV A,B
      STA 0B01H ;print tens in second 7-segment display

      PUSH D ;store DE
      LXI D,0B00H ;used for STDM
      CALL STDM
      CALL DCD
      MOV D,B
      MOV E,C ;restore D and E
      LXI B,0032H ;used for 0.05s delay
      POP D ;restore DE
      POP H ;restore HL
      JMP LIGHTS

END
```

Πράγματι, εκτελώντας τον κώδικα πραγματοποιείται η ζητούμενη λειτουργία:



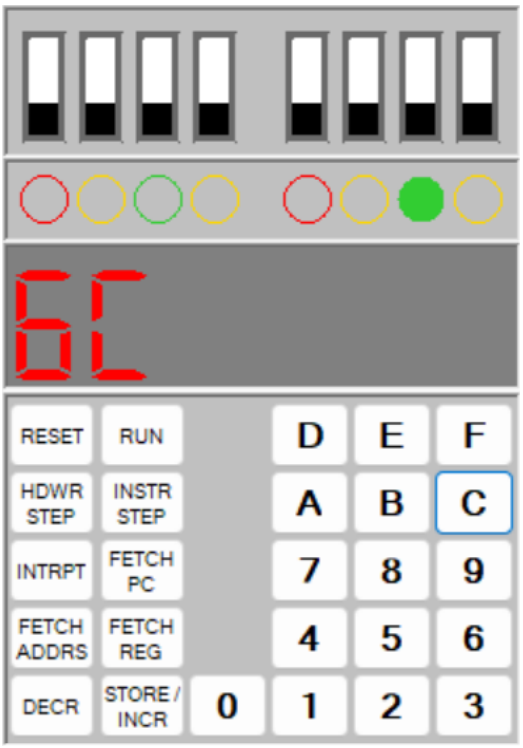
Άσκηση 2

Ο ζητούμενος κώδικας είναι ο εξής (*'askisi2.8085'*):

```
; Initializations
MVI D,40H ; Set threshold K1
MVI E,80H ; Set threshold K2
INR D
INR E
MVI A,10H ; 4 right-most 7-segment displays need to be empty
STA 0B00H
STA 0B01H
STA 0B02H
STA 0B03H

MVI A,0DH ; interrupt mask to allow RST 6.5
SIM
EI ; enable interrupts

WAIT:
  JMP WAIT

INTR_ROUTINE:
  POP H ; reduce stack
  CALL KIND ; read keyboard
  STA 0B05H
  RLC
  RLC
  RLC
  RLC ; multiply by 16
  MOV B,A
  CALL KIND
  STA 0B04H
  ADD B ; get total number
  MOV B,A
  PUSH D ;temporarily save D and E
  LXI D,0B00H ;address for STDM
  CALL STDM
  CALL DCD
  POP D ;restore D and E
  MOV A,B
```

```
  CMP D
  JC LED0
  CMP E
  JC LED1
  MVI A,FBH ; LED2
  JMP FINISH
LED0:
  MVI A,FEH ; LED0
  JMP FINISH
LED1:
  MVI A,FDH ; LED1
FINISH:
  STA 3000H
  EI
  JMP WAIT
END
```

Μέσα στον κώδικα, χρησιμοποιούνται οι εντολές SIM και EI για την επίτρεψη των διακοπών RST 6.5, καθώς και οι ρουτίνες STDM και DCD για το τύπωμα στις δεκαεξαδικές οθόνες. Στην αρχή του προγράμματος, δίνουμε ενδεικτικές τιμές στους καταχωρητές D και E, για να ορίσουμε τα κατώφλια K1 και K2. Μετά από τις εντολές αρχικοποίησης το πρόγραμμα αναμένει τη διακοπή RST 6.5 για να δεχτεί την είσοδο του δεκαεξαδικού πληκτρολογίου με τη ρουτίνα KIND, να την τυπώσει στις δεκαεξαδικές οθόνες και να κάνει τις κατάλληλες συγκρίσεις για να ανάψει το αντίστοιχο LED. Αυτό επαναλαμβάνεται με την αναγνώριση της επόμενης διακοπής.
Παρακάτω φαίνεται ένα παράδειγμα λειτουργίας του προγράμματός μας:



Άσκηση 3

Η λειτουργία καθεμίας από τις ζητούμενες διεργασίες εξηγείται αναλυτικά στα σχόλια του κώδικα.

α) Τη μακροεντολή SWAP Nible Q που εναλλάσσει το χαμηλότερης αξίας HEX ψηφίο με το υψηλότερης των καταχωρητών γενικού σκοπού B, C, D, E, H και L καθώς και της θέσης μνήμης που ‘δείχνει’ ο διπλός καταχωρητής H-L

```
SWAP_Nibble_Q:
  ; Save the original values of the registers
  PUSH BC
  PUSH DE
  PUSH HL

  ; Swap the lowest and highest nibbles of B register
  LD A,B
  AND 0FH      ; Mask the lowest nibble
  RRCA         ; Rotate right to get the highest nibble as the lowest
nibble
  RRCA         ; Rotate right again to restore the original value
  OR A,B      ; Combine the modified and original nibbles
  LD B,A

  ; Repeat the same process for C, D, E, H, and L registers
  LD A,C
  AND 0FH
  RRCA
  RRCA
  OR A,C
  LD C,A

  LD A,D
  AND 0FH
  RRCA
  RRCA
  OR A,D
  LD D,A

  LD A,E
  AND 0FH
  RRCA
  RRCA
```

```
  OR A,E
  LD E,A

  LD A,H
  AND 0FH
  RRCA
  RRCA
  OR A,H
  LD H,A

  LD A, L
  AND 0FH
  RRCA
  RRCA
  OR A,L
  LD L,A

  ; Swap the lowest and highest nibbles of the memory location pointed by
HL
  LD A,(HL)
  AND 0FH
  RRCA
  RRCA
  OR A,(HL)
  LD (HL),A

  ; Restore the original values of the registers
  POP HL
  POP DE
  POP BC

  ; Return from the macro
  RET
  END
```

β) Τη μακροεντολή FILL RP, X, K, γεμίζει ένα τμήμα μνήμης με μια σταθερά K (0-255).

```
FILL_RP_X_K MACRO RP, X, K
    PUSH AF                  ; Save AF register
    LD A,X                  ; Load X into A register
    OR A                     ; Check if X is zero
    JP Z, FILL_ZERO_SIZE    ; Jump to FILL_ZERO_SIZE if X is zero
```

```
    DEC A                    ; Decrement X
    LD BC,0                 ; Initialize counter BC
    LD B,A                  ; Load X into B register

FILL_LOOP:
    LD (RP),K               ; Store constant K at memory location pointed
by RP
    INC RP                   ; Increment RP
    DJNZ FILL_LOOP           ; Decrement B and loop back if not zero
    POP AF                   ; Restore AF register
    RET

FILL_ZERO_SIZE:
    LD BC,256               ; Set segment size to 256
    LD (RP),K               ; Store constant K at memory location pointed
by RP
    LDIR                     ; Fill remaining 255 locations with K
    POP AF                   ; Restore AF register
    RET

ENDM
```

γ) Η μακροεντολή RHLR, περιστρέφει τα περιεχόμενα των καταχωρητών H και L κατά μια θέση δεξιά.

```
RHLR MACRO
  PUSH AF                ; Save AF register
  PUSH HL                ; Save HL register

  RRC L                 ; Rotate right L register
  RRA                   ; Rotate right H register through carry
  RLC L                 ; Rotate left L register to move L0 to CY

  POP HL                 ; Restore HL register
  POP AF                 ; Restore AF register

  RET
ENDM
```

Άσκηση 4

Από τα δεδομένα της εκφώνησης έχουμε τις αρχικές τιμές τόσο του Program Counter (PC) όσο και του Stack Pointer (SP), 0840H και 3000H αντίστοιχα. Εκτελώντας την εντολή **CALL 0900H** γίνονται τα εξής:

- Η εντολή CALL σπρώχνει τη διεύθυνση της επόμενης εντολής (0842H) στη στοίβα και κάνει jump στην υπο-ρουτίνα που βρίσκεται στη διεύθυνση 0900H.
- Η τιμή του PC γίνεται ίση με 0900H.
- Η τιμή του SP μειώνεται κατά δύο, δηλαδή είναι ίση με 2FFEH, αφού οι στοίβα μεγαλώνει προς τα κάτω.
- Η διεύθυνση 0842H αποθηκεύεται στη διεύθυνση μνήμης που δείχνει ο SP (δηλ. 2FFEH) και ο SP συγχρονίζεται κατάλληλα.

Όταν καλείται το RST 5.5 Interrupt, ο μικροεπεξεργαστής ακολουθεί τα interrupt handling procedures του. Έτσι, η νέα τιμή του PC (0900H) σπρώχνεται στη στοίβα και το PC τίθεται στην αρχική διεύθυνση Interrupt Service Routine (ISR), δηλαδή 0028H (για τον RST 5.5).
Οι τιμές του μετρητή προγράμματος και του δείκτη σωρού ανανεώνονται έτσι ώστε PC = 0028H και SP = FFCH. Η προηγούμενη τιμή του PC, δηλ. 0900H, αποθηκεύεται στη διεύθυνση μνήμης που δείχνει ο δείκτης στοίβας (2FFCH). Ο τελευταίος συγχρονίζεται, και πάλι, κατάλληλα.
Σχετικά με την εκτέλεση της ρουτίνας ISR, πρόκειται για μια ρουτίνα που είναι σχεδιασμένη για τη διαχείριση των interrupts (εν προκειμένω του RST 5.5). Οι εντολές που εκτελούνται στην αρχή και επιστρέφονται από την ISR εξαρτώνται από τις ειδικές προϋποθέσεις του interrupt handler. Κατά την επιστροφή από την ISR, η διεύθυνση επιστροφής (0900H) εξάγεται από τη στοίβα στον PC, ενώ ο ίδιος ο PC επιστρέφει στην τιμή 0900H. Ο SP αυξάνεται κατά 2 και λαμβάνει την τιμή 2FFEH, επαναφέροντας τον δείκτη στοίβας στην αρχική του τιμή.

Παρακάτω φαίνεται ο πίνακας με τις τιμές των PC και SP κατά τη διάρκεια της διαδικασίας που περιγράψαμε:

Step	Instruction/Event	PC	SP
1	CALL 0900H	0840H	3000H
2	RST 5.5 Interrupt Occurs	0840H	2FFEH

3	Push PC (0840H) to Stack	0840H	2FFCH
4	Load PC with Interrupt Vec	0028H	2FFCH
5	Decrement SP by 2	0028H	2FFEH
6	Push PC (0028H) to Stack	0028H	2FFCH
7	Load PC with Subroutine	0900H	2FFCH
8	Service Routine Execution	-	-
9	RET Instruction Executed	-	-
10	Pop PC from Stack	0028H	2FFCH
11	Increment SP by 2	0028H	2FFEH

12	Pop PC from Stack	0840H	2FFEH

Μετά τη διακοπή και τη ρουτίνας, οι τελικές τιμές είναι:

- PC: 0840H
- SP: 2FFEH

Λάβετε υπόψη ότι τα περιεχόμενα της στοίβας και του σωρού δεν αναφέρονται ρητά ούτε επηρεάζονται από το δεδομένο σενάριο, επομένως δεν περιλαμβάνονται στον πίνακα.

Άσκηση 5

α) Το πρόγραμμα που ζητείται στην άσκηση παρατίθεται παρακάτω:

```
MVI A, 0DH   ; Load value 0DH into accumulator A
SIM          ; Set interrupt mask
LXI H, 0000H ; Initialize HL register pair to 0000H
MVI C, 64    ; Number of steps (64)
EI           ; Enable interrupts

ADDR:        ; Address loop for delaying
MVI A, C     ; Load value of C into accumulator A
CPI 00H      ; Compare accumulator A with 00H
JNZ ADDR     ; Jump to ADDR if not zero
DI           ; Disable interrupts
DAD H        ; Add HL to itself
DAD H        ; Add HL to itself
DAD H        ; Add HL to itself
MOV A, L     ; Move contents of L register to accumulator A
ANI 80H      ; Mask the most significant bit of accumulator A
MVI L, 00H   ; Clear L register
CPI 00H      ; Compare accumulator A with 00H
JNZ ROUNDING ; Jump to ROUNDING if not zero

BACK:        ; Loop for halting execution
HLT          ; Halt execution
```

```
ROUNDING:    ; Rounding subroutine
INR H        ; Increment contents of H register
JMP BACK     ; Jump to BACK

0034:        ; Address for RST6.5 interrupt
JMP RST6_5   ; Jump to RST6.5 subroutine

RST6_5:      ; RST6.5 interrupt subroutine
  PUSH PSW   ; Push program status word onto stack
  MOV A, C   ; Move value of C into accumulator A
  ANI 01H    ; Perform bitwise AND with 01H
  JPO FOUR_MSB ; Jump to FOUR_MSB if parity is odd (LSB is 1)
  IN 20H     ; Read 4 LSBs from device into accumulator A
  ANI 0FH    ; Mask the lower nibble of accumulator A
  MOV B, A   ; Move accumulator A into B register
  JMP FOUR_LSB ; Jump to FOUR_LSB

FOUR_MSB:    ; Subroutine for handling 4 MSBs
  IN 20H     ; Read 4 MSBs from device into accumulator A
  ANI 0FH    ; Mask the lower nibble of accumulator A
  RLC        ; Rotate accumulator A left through carry
  RLC        ; Rotate accumulator A left through carry
  RLC        ; Rotate accumulator A left through carry
  RLC        ; Rotate accumulator A left through carry
  ORA B      ; Perform logical OR with B register
  MVI D, 00H ; Clear D register
  MOV E, A   ; Move accumulator A into E register
  DAD D      ; Add DE to HL

FOUR_LSB:    ; Subroutine for handling 4 LSBs
  POP PSW    ; Pop program status word from stack
  DCR C      ; Decrement C
  EI         ; Enable interrupts
  RET        ; Return from subroutine
```

β) Και αντίστοιχα για το δεύτερο μέρος της εκφώνησης:

```
LXI H, 00H    ; Initialize HL register pair to 00H
MVI C, 64     ; Number of steps (64)

MAIN:
IN 20H        ; Read PORT_IN to check Data Ready line
ANI 80H       ; Mask other bits except X7
JP MAIN       ; Jump to MAIN if X7 is high (waiting for it to go low)

MOV A, C      ; Move value of C into accumulator A
ANI 01H       ; Perform bitwise AND with 01H
JPO FOUR_MSB ; Jump to FOUR_MSB if parity is odd (LSB is 1)
IN 20H        ; Read 4 LSBs from device into accumulator A
ANI 0FH       ; Mask the lower nibble of accumulator A
MOV B, A      ; Move accumulator A into B register
JMP FOUR_LSB ; Jump to FOUR_LSB

FOUR_MSB:     ; Subroutine for handling 4 MSBs
IN 20H        ; Read 4 MSBs from device into accumulator A
ANI 0FH       ; Mask the lower nibble of accumulator A
RLC           ; Rotate accumulator A left through carry
RLC           ; Rotate accumulator A left through carry
RLC           ; Rotate accumulator A left through carry
RLC           ; Rotate accumulator A left through carry
ORA B         ; Perform logical OR with B register
MVI D, 00H    ; Clear D register
MOV E, A      ; Move accumulator A into E register
DAD D         ; Add DE to HL

FOUR_LSB:     ; Subroutine for handling 4 LSBs
DCR C         ; Decrement C
JZ ADDR       ; Jump to ADDR if C is zero

CHECK:
IN 20H        ; Read PORT_IN to check Data Ready line
ANI 80H       ; Mask other bits except X7
JM CHECK      ; Jump to CHECK if X7 is high (waiting for it to go low)
```

```
JMP MAIN      ; Jump to MAIN

ADDR:         ; Address loop for delaying
DAD H         ; Add HL to itself
DAD H         ; Add HL to itself
DAD H         ; Add HL to itself
MOV A, L      ; Move contents of L register to accumulator A
ANI 80H       ; Mask the most significant bit of accumulator A
MVI L, 00H    ; Clear L register
CPI 00H       ; Compare accumulator A with 00H
JNZ ROUNDING ; Jump to ROUNDING if not zero

BACK:         ; Loop for halting execution
HLT           ; Halt execution

ROUNDING:     ; Rounding subroutine
INR H         ; Increment contents of H register
JMP BACK      ; Jump to BACK
```