ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

Συστήματα Μικροϋπολογιστών

3η Ομάδα Ασκήσεων

Αναστάσιος Λαγός - el13531
Κωνσταντίνος Βασιλάκης - el16504

Ασκηση 1

Ο κώδικας είναι ο παρακάτω

```
DATA: MVI A,10H ;Only middle segs should display
STA 0B00H
STA 0B01H
STA 0B04H
STA 0B05H

LXI B,0064H
MVI A,0DH               ;Unmask RST6.5 - Mask RST7.5,RST5.5
SIM
LXI D,0B00H             ;Load address of digits for 7-seg display
EI

RESET1:
CALL BLNK
MVI A,FFH
OUT 30H

WAIT_LOOP:              ;Waiting for interrupt
JMP WAIT_LOOP

INTR_ROUTINE:
POP H                   ;Don't need to return
MVI A,00H               ;Light LEDs
OUT 30H

LXI B,0064H             ;100ms
LXI D,0B00H             ;Load address of digits for 7-seg display

RESET_TIMER:
LXI H,0600H
SHLD 0B02H
CALL DISPLAY_SEG
EI

DECIMAL_CHECK:
MOV A,L
CPI 00H         ;If L = 0 then we need to DCR the decimals.
JZ DCR_DECIMAL
DCR L                   ;Else DCR L
SHLD 0B02H
CALL DISPLAY_SEG
JMP DECIMAL_CHECK

DCR_DECIMAL:
DCR H
MOV A,H
CPI 00H         ;If H = 0 then we reached the final countdown.
MVI L,09H
SHLD 0B02H
```

```
CALL DISPLAY_SEG
JZ END_COND
JMP DECIMAL_CHECK


DISPLAY_SEG:            ;Routine that displays timer on screen
PUSH PSW                ;Need to store 8085 current state
                        ;to call STDM and DCD
PUSH H
PUSH D
PUSH B
CALL STDM
MVI A,0AH               ;10 * 100ms = 1 sec of displaying the
                         ;number on the screen
DISPLAY_LOOP:           ;If pair BC is loaded with 03E8 and we just
                        ;CALL DELB without looping the call to DCD
CALL DCD                ;then the numbers appear on the screen for
                        ;a brief moment before disappearing
POP B
CALL DELB               ;Note: 1 sec varies cause simulation
PUSH B
DCR A
CPI 00H
JNZ DISPLAY_LOOP
POP B                   ;Retrive 8085 status
POP D
POP H
POP PSW
RET

END_COND:               ;If decimal is 0 then count down from 9
                        ;and be done.
MOV A,L
CPI 00H
DCR L
SHLD 0B02H
CALL DISPLAY_SEG
JNZ END_COND
JMP RESET1

END
```

Ασκηση 2

Ο κώδικας είναι ο παρακάτω

```
EQU K1_VALUE,10H
EQU K2_VALUE,20H

START:
    ; Enable Rst 6.5 interrupt
    MVI A,0DH
    SIM
    EI

; Wait for the interrupt to happen
WAIT_INTR:
    JMP WAIT_INTR

INTR_ROUTINE:
    ; Save register state. HL are not used so
    ; they are not stored in the stack
    PUSH PSW
    PUSH B
    PUSH D

    LXI D,0B50H     ; Arbitrary memory address we write
                    ; the screen data

    ; Get the first input and write it to the memory
    ; location 0B50H (1st digit on the screen)
    CALL KIND
    MOV C,A
    STA 0B50H

    ; Get the second input and write it to the memory
    ; location 0B51H (2nd digit on the screen)

    CALL KIND
    STA 0B51H

    ; Combine the second input (4 MSB) with the first (4 LSB) store
    ; it on C
    RLC
    RLC
    RLC
    RLC
    ADD C
    MOV C,A

    ; Make the rest of the screen bits zero
    MVI A,00H
    STA 0B52H
    STA 0B53H
    STA 0B54H
```

```
    STA 0B55H

    ; Display the two inputs on the screen
    CALL STDM
    CALL DCD


    ; Get the combined 8 bit number
    MOV A,C
    STA 0B26H

    MVI D,K1_VALUE
    MVI E,K2_VALUE

    ; Increase the value so the first and second intervals are
    ; K1 and K2 inclusive respectively
    INR D
    INR E

    ; Right led if for (0,K1]
    MVI B,01H
    CMP D
    JC SWITCH_LEDS_ON

    ; Middle led is for (K1,K2]
    MVI B,02H
    CMP E
    JC SWITCH_LEDS_ON

    ; Left led is for (K2,FFH)
    MVI B,04H


SWITCH_LEDS_ON:
    MOV A,B
    CMA
    STA 3000H
    EI

    ; Load previous register state
    POP B
    POP D
    POP PSW
    RET
END
```

Ασκηση 3

α)

Η παρακάτω μακροεντολή δουλεύει ως εξής. Αποθηκεύσει τον A και τα flags στην στοίβα. Μεταφέρει τον Q στον A και τον περιστρέφει 4 φορές και μετά τον αποθηκεύει πάλι πίσω στον Q. Αντίστοιχα αποθηκεύει τα περιεχόμενα του HL στον A τα περιστρέφει 4 φορές και τα σώνει πάλι στην ίδια διεύθυνση. Τέλος ανακτά από την στοίβα τις αποθηκευμένες τιμές του A και των flags.

```
SWAP Nimble MACRO Q

PUSH PSW

MOV A,Q
RLC
RLC
RLC
RLC
MOV Q,A

MOV A,M
RRC
RRC
RRC
RRC
MOV M,A

POP PSW

ENDM
```

β)

Στην παρακάτω μαρκοεντολή αποθηκεύουμε τα περιεχόμενα των flags, A και H στην στοίβα. Τοποθετούμε τον αριθμό των στοιχείων που θέλουμε να κάνουμε ίσα με K στον A και μεταφέρουμε στο HL τις τιμές του RP. Γράφουμε K σε αυτήν τη θέση μνήμης και με ένα loop αυξάνουμε την θέση μνήμης κατά ένα και αφαιρούμε ένα από τον counter X που είναι στον A μέχρι να γράψουμε τις απαιτούμενες τιμές. Τέλος ανακτάμε από την στοίβα τα H,A και flags.

```
FILL MACRO RP,X,K

PUSH PSW
PUSH H
MOV A,X
MOV H,R
MOV L,P

LOOP:
```

```
MVI M,K
INR M
DCR A
JNZ LOOP

POP H
POP PSW

ENDM
```

γ)

Στην παρακάτω μαρκοεντολή αρχικά αποθηκεύουμε στην στοίβα τον A και flags. Κάνουμε με την σειρά ένα δεξιό rotate μέσα από το carry τον H και μετά τον L όσες φορές έχει επιλεγεί. Στο τέλος ο HL έχει τις shifted τιμές και ο καταχωρητής B έχει την τιμή του carry.

```
RHLR MACRO n

PUSH PSW
MOV B,n

LOOP:
MOV A,B
CPI 00H
JZ STOP
MOV A,H
RAR
MOV H,A
MOV A,L
RAR
MOV L,A

DCR B
JMP LOOP

STOP:
JC STORE
MVI B,00H
JMP ENDING

STORE:
MVI B,01H

ENDING:
POP PSW

ENDM
```

Ασκηση 4

Ο 8085 ως απάντηση σε μια διακοπή αρχικά ολοκληρώνει την τρέχουσα εντολή (CALL 0880H). Πριν την ολοκληρώσει οι PC,SP και η stack φαίνονται στην πρώτη στήλη. Με την εκτέλεση της CALL 0880H αποθυκεύεται στην stack το προηγούμενο PC και η κατάσταση του μΕ (PUSH PSW) ενώ το PC γίνεται 0880, όπως στην δεύτερη στήλη. Εδώ γίνεται η αναγνώριση της συσκευής που προκάλεσε την διακοπή και αν υπήρχαν πολλαπλές διακοπές η εκτέλεση του ISR θα γινόνταν με βάση την προτεραιότητα TRAP -> RST7.5 -> RST6.5 -> RST5.5. Όμως έχουμε μια διακοπή οπότε απλώς θα εκτελεστεί η ISR. Στην συνέχεια θα αποθυκευτεί στην stack το PC και έπειτα θα μεταβεί στην διεύθυνση της υπορουτίνας της εκάστοτε διακοπής. Στην συγκεκριμένη περίπτωση στην διεύθυνση 003C.Εδώ θα γίνει πάλι αποθύκευση της κατάστασης του μΕ και των καταχωρητων (B,C,D,E,H,L) στην stack.Θα εκτελεστεί το κύριο σώμα της ρουτίνας εξυπηρέτησης, θα ελευθερωθούν από την stack το PSW και οι καταχωρητές, θα γίνει επίτρεψη νέων διακοπών και εν συνεχεία το PC θα επιστρέψει στην διεύθυνση που βρισκόταν πριν την διακοπή(0880). Αντίστοιχα και για την εκτέλεση της ρουτίνας στην θέση 0880.

PC	0800	0880	003C
SP	00	PSW	80
SP + 1	30	00	08
SP + 2		08	PSW
SP + 3		00	00
SP + 4		30	08
SP + 5			00
SP + 6			30

Ασκηση 5

α)

Στον παρακάτω κώδια περιμένουμε ένα interrupt 6.5. Όταν έρθει ανάλογα αν έχουμε LSB ή MSB στην είσοδο (το βρίσκουμε ανάλογα με το αν το counter που έχουμε είναι μονό ή ζυγό αφού αυτά έρχονται εναλλάξ) τα προσθέτουμε σε ένα ολικό άθροισμα στο HL επειδή οι αριθμοί είναι 8bit οπότε το συνολικό άθροισμα μπορεί να ξεπεράσει τα 8 bit. Στο τέλος διαιρούμε με 32 κάνοντας 4 shift right του HL. Το τελικό αποτέλεσμα θα βρίσκεται στον L. Γενικά στον παρακάτω κώδικα αφού η διακοπή μπορεί να γίνει οποτεδήποτε σώνουμε τα δεδομένα για ασφάλεια στην μνήμη μεταξύ των διακοπών και τα ανακτάμαι από εκεί για την επεξεργασία.

```
START:
    ; Store in memory the counter at 0B10H and
    ; the current 16 bit sum in 0B11H,0B12H to
    ; retrieve them between interrupts
    ; Counter starts from 64 = 40H
    MVI A,40H
    STA 0B10H

    ; Sum starts from 0
    MVI A,00H
    STA 0B11H
    STA 0B12H

    ; Enable Rst 6.5 interrupt
    MVI A,0DH

    SIM
    EI

; Wait for the interrupt to happen
WAIT_INTR:
    JMP WAIT_INTR

RST6.5:
    ; Save register state
    PUSH PSW
    PUSH B
    PUSH D
    PUSH H

    JMP RST6.5_ROUTINE

    ; Load previous register state
    POP PSW
    POP B
    POP D
    POP H

RST6.5_ROUTINE:
    IN 20H      ; Get input to A
```

```
    ANI 0FH    ; Keep the 4 LSB
    MOV C,A

    ; Load Current Counter and save it to D
    LDA 0B10H
    MOV D,A

    ; If it is even it is LSB bytes so no shift
    ; is needed
    ANI 01H
    CPI 01H
    JZ ADD_RESULTS

GET_4_MSB:
    MOV A,C
    RLC
    RLC
    RLC
    RLC
    MOV C,A

ADD_RESULTS:
    ; Load Current Sum
    LHLD 0B11H

    ; 16 bit addition of BC to HL
    ; C containes the new bytes
    MVI B,00H
    DAD B

    ; Decrease the counter
    DCR D
    JZ FIND_MEAN
    ; If counter is not yet 0 update it in
    ; memory along with the current sum and
    ; return waiting for the next interrupt
    MOV A,D
    STA 0B10H
    SHLD 0B11H
    EI
    RET

FIND_MEAN:
    ; We shift 4 times to the right the register HL
    ; in order to divide the sum of the 32 numbers
    ; by 32 and find the mean value. We first set
    ; the carry to 0 and we rotate right the H and
    ; then the L registers. We do this 4 times
    STC
    CMC
    MOV A,H
    RAL
    MOV H,A
    MOV A,L
    RAL
    MOV L,A
```

```
    STC
    CMC
    MOV A,H
    RAL
    MOV H,A
    MOV A,L
    RAL
    MOV L,A

    STC
    CMC
    MOV A,H
    RAL
    MOV H,A
    MOV A,L
    RAL
    MOV L,A

    STC
    CMC
    MOV A,H
    RAL
    MOV H,A
    MOV A,L
    RAL
    MOV L,A


    ; The result is in L. We also store it
    ; in 0B13H and finish the program
    MOV A,L
    LDA 0B13H

    ; Load previous register state
    POP PSW
    POP B
    POP D
    POP H
END
```

β)

Ο παρακάτω κώδικας είναι παρόμοιος με αυτόν του ερωτήματος α. Έχουν αφαιρεθεί οι διακοπές και στην θέση τους έχει μπει μία λογική που ελέγχει αν έχουμε μετάβαση από 0 σε ένα στο x7. Αν ναι κάνουμε ένα κύκλο επεξεργασίας αλλιώς συνεχίζουμε το loop. Επίσης έχουν αφαιρεθεί οι αποθηκεύσεις στην μνήμη αφου τώρα το πρόγραμμα θα τρέξει συνεχόμενα και αν μία διακοπή ή άλλη διεργασία αναλάβει τον έλεγχο θα πρέπει να αποθηκεύσει τα δεδομένα στο stack πρώτα και να τα ανακτήσει όταν επιστρέψει.

```
START:
    ; Initialization of counter in D and sum in HL
    MVI D,40H
    MVI H,00H
    MVI L,00H
    ; C has the new bytes
    MVI C,00H
    ; E has the previous x7 value
    MVI E,00H
    MVI B,00H

PROCESS_DATA:
    ; Update previous x7 value
    MOV E,B
    IN 20H      ; Get input to A
    ; Temporarily store A to C
    MOV C,A

    ANI 80H
    ; Temporarily store current x7 value to B
    MOV B,A

    CPI 01H
    ; If x7 = 0 continue looping
    JNZ PROCESS_DATA
    ; If x7 = 1 check x7 previous value
    ; stored in E
    MOV A,E
    CPI 00H
    ; If previous value is also 1 continue looping
    ; else process new data
    JNZ PROCESS_DATA

CONTINUE_PROCESSING:
    MOV A,C
    ANI 0FH     ; Keep the 4 LSB
    MOV C,A
    ; If it is even it is LSB bytes so no shift
    ; is needed
    ANI 01H
    CPI 01H
    JZ ADD_RESULTS

GET_4_MSB:
    MOV A,C
    RLC
    RLC
    RLC
    RLC
    MOV C,A

ADD_RESULTS:
    ; 16 bit addition of BC to HL
    ; C containes the new bytes
```

```
    MVI B,00H
    DAD B
    ; Decrease the counter
    DCR D
    JZ FIND_MEAN
    ; If counter is not yet 0 update loop
    JMP PROCESS_DATA

FIND_MEAN:
    ; We shift 4 times to the right the register HL
    ; in order to divide the sum of the 32 numbers
    ; by 32 and find the mean value. We first set
    ; the carry to 0 and we rotate right the H and
    ; then the L registers. We do this 4 times
    STC
    CMC
    MOV A,H
    RAL
    MOV H,A
    MOV A,L
    RAL
    MOV L,A

    STC
    CMC
    MOV A,H
    RAL
    MOV H,A
    MOV A,L
    RAL
    MOV L,A

    STC
    CMC
    MOV A,H
    RAL
    MOV H,A
    MOV A,L
    RAL
    MOV L,A

    STC
    CMC
    MOV A,H
    RAL
    MOV H,A
    MOV A,L
    RAL
    MOV L,A

    ; The result is in L. We also store it
    ; in 0B13H and finish the program
    MOV A,L
    LDA 0B13H
END
```