Συστήματα Μικροϋπολογιστών

3η Σειρά Ασκήσεων

Τσιλιμιγκουνάκης Μιχάλης (el19001)

Στεφανής Παναγιώτης (el19096)

Ασκήσεις Προσομοίωσης:

1η Άσκηση

Ζητούμενο:

Το πρόγραμμα, μέσω διακοπής RST6.5 ελέγχει τα φώτα ενός χώρου. Όταν λάβει διακοπή αναβοσβήνει τα LED για 45 δευτερόλεπτα με διάρκεια άναψε-σβήσε 0.5 δευτερόλεπτα. Αν λάβει πάλι διακοπή όσο γίνεται αυτό κάνει reset τον timer των 45 δευτερολέπτων. Σε όλη την διάρκεια που αναβοσβήνει τα LED, εμφανίζει στα 2 LSB των 7-segment πόσα δευτερόλεπτα μένουν ακόμα.

Η ροή του προγράμματος έχει ως εξής:

Στην αρχή του προγράμματος κάνουμε τις απαραίτητες αρχικοποιήσεις, δηλαδή βγάζουμε την προστασία της μνήμης, αρχικοποιούμε την οθόνη με κενούς χαρακτήρες και ενεργοποιούμε τις διακοπές. Μετά υπάρχει ένας ατέρμονος βρόγχος που απλά λουπάρει στο εαυτό του (περιμένει δηλαδή εκεί να έρθει διακοπή). Όταν έρθει διακοπή, πάμε στην ρουτίνα εξυπηρέτησης της, η οποία αρχικοποιεί το delay και το πόσα δευτερόλεπτα θα αναβοσβήσουν τα LEDs και ξαναενεργοποιεί τις διακοπές ώστε να μπορεί να έρθει πάλι διακοπή και θα ξαναξεκινήσει η όλη διαδικασία από την αρχή. Στην συνέχεια ανάβει και σβήνει τα LED, προσθέτοντας και μισό δευτερόλεπτο καθυστέρηση ενδιάμεσα και καλεί την συνάρτηση “COUNTDONW” η οποία φορτώνει στην μνήμη τον μετρητή πόσων δευτερολέπτων απομένουν και κάνει τις κατάλληλες μετατροπές ώστε να εμφανιστεί στα 7 segment displays. Στην συνέχεια καλείται η ρουτίνα “SHOW” η οποία εμφανίζει με τις κατάλληλες κλήσεις τα εναπομείναντα δευτερόλεπτα στην οθόνη. Επαναλαμβάνουμε την όλη παραπάνω διαδικασία 45 φορές.

Ο κώδικας (επόμενη σελίδα)

```
IN 10H
MVI B,06H
LXI H,0A00H
FOR:
    MVI M,10H
    INX H
    DCR B
    JNZ FOR

    MVI A,0DH   ;load value in A for mask interrupt
    SIM     ;load interrupt mask with A
    EI      ;enable interrupt RST 6.5

WAIT_UNTIL_INT:
    JMP WAIT_UNTIL_INT

INTR_ROUTINE:
    MVI E,2DH       ;E=45 for LED ON/OFF duration
    LXI B,01F4H ;B = 500 (500 ms in DELB)
    EI

START:
    MVI D,02H   ;D=2 to simulate 1 sec time
    MVI A,00H   ;00000000--> A
    STA 3000H   ;LED ON
    CALL DELB
    MVI A,FFH   ;11111111--> A
    STA 3000H   ;LED OFF
    CALL DELB
    CALL COUNTDOWN

LOOP1:
    CALL SHOW   ;show the remain time
    ;CALL DELB
    DCR D
    JNZ LOOP1   ;loop twice (2*500 ms = 1 sec)
    DCR E   ;decrease the time by 1
    JNZ START   ;loop to start

    JMP WAIT_UNTIL_INT

COUNTDOWN:
    PUSH PSW
    PUSH B
    PUSH H
    MOV A,E ;Load with A with remaining time in Decimal
    MVI B,FFH   ;Initialize C as "-1"
L1: INR B
    SUI 0AH
    JNC L1  ;While A>0 loop to L1 to compute decades
    ADI 0AH ;Add 10 to A to compute the units
    LXI H,0A04H
    MOV M,A ;Store in fifth seg-disp the units
    INX H
    MOV M,B ;Store in sixth seg-disp the decades
    POP H
    POP B
    POP PSW
    RET

SHOW:
    PUSH PSW
    PUSH D
    LXI D,0A00H ;Load D with the first memory position
    CALL STDM
    CALL DCD        ;display the time in the last two seg-disp
    POP D
    POP PSW
    RET
END
```

2η Άσκηση

Ζητούμενο:

Το πρόγραμμα, όταν προκαλείται διακοπή RST 6.5, διαβάζει 2 δεκαεξαδικά ψηφία από το πληκτρολόγιο και τα απεικονίζει στα 2 μεσαία 7 segment display. Στην συνέχεια συγκρίνει τον αριθμό αυτό με 3 κατώφλια και ανάλογα το διάστημα στο οποίο ανήκει ο αριθμός ανάβει το πρώτο, δεύτερο, τρίτο ή τέταρτο LED (από δεξιά, δηλαδή τα LSB).

Η ροή του προγράμματος έχει ως εξής:

Ξεκινάμε με τις αρχικοποιήσεις (μνήμη, κατώφλια, αρχικοποίηση της οθόνης με κενούς χαρακτήρες). Στην συνέχεια μπαίνουμε σε μια ατέρμονη λούπα, η οποία αποτελεί το κύριο πρόγραμμα (label “WAIT”). Εκεί εκτελείται κώδικας στο μικροεπεξεργαστή περιμένοντας να έρθει κάποιο interrupt. Η ρουτίνα εξυπηρέτησης του interrupt αρχικά διαβάζει από το πληκτρολόγιο και τους 2 χαρακτήρες και τους αποθηκεύει προσωρινά στον A. Στην συνέχεια κάνει συγκρίσεις για να δει σε ποιο διάστημα ανήκει ο αριθμός και ανάβει το αντίστοιχο LED. Στην συνέχεια επιστρέφει από το την ρουτίνα διαχείρισης της διακοπής στο κύριο πρόγραμμα.

Ο κώδικας: (επόμενη σελίδα)

```
    IN 10H          ;remove memory protection
    MVI A,0DH       ;insert interupt mask into A
    SIM             ;move A into interupt mask
    EI              ;enable interupt based on mask

    MVI C,3CH       ;first threshold
    MVI D,78H       ;second threshold
    MVI E,B4H       ;third threshold

    MVI B,06H       ;iterator
    LXI H,0A00H         ;first memory location
DISP_INIT:
    MVI M,10H       ;store to memory blank character code
    INX H           ;next memory position
    DCR B           ;iterator--
    JNZ DISP_INIT   ;repeat

WAIT:
    JMP WAIT            ;main program (waiting for an interupt)

INTR_ROUTINE:
    CALL KIND       ;get keyboard input
    STA 0A03H       ;store in position of msb in 7seg display
    RLC                 ;4 left shifts to make it msb (4 shifts because it is in hex)
    RLC
    RLC
    RLC
    MOV B,A             ;move to b to get next digit
    CALL KIND       ;get next digit
    STA 0A02H       ;store in lsb position in 7srg display
    ORA B           ;add digits together
    CMP C           ;compare if it is less than threshold1
    JC FIRST_LED        ;turn on first led if <threshold1
    JZ FIRST_LED        ;turn on first led if =threshold1
    CMP D           ;compare if it is less than threshold2
    JC SECOND_LED   ;turn on second led if <threshold2
    JZ SECOND_LED       ;turn on second led if =threshold2
    CMP E           ;compare if it is less than threshold3
    JC THIRD_LED        ;turn on third led if <threshold3
    JZ THIRD_LED        ;turn on third led if =threshold3
    JMP FOURTH_LED  ;turn on last led
CONT:
    EI              ;re-enable interrupt
    RET             ;return

FIRST_LED:
    MVI A,FEH       ;turn on first led
    JMP     OUTPUT      ;jump to output to output into leds and display
SECOND_LED:
    MVI A,FDH       ;turn on second led
    JMP OUTPUT      ;jump to output to output into leds and display

THIRD_LED:
    MVI A,FBH       ;turn of third led
    JMP OUTPUT      ;jump to output to output into leds and display

FOURTH_LED:
    MVI A,F7H       ;turn on fourth led

OUTPUT:
    STA 3000H       ;output into leds
    PUSH D          ;push d into stack
    LXI D,0A00H     ;D = &(what to output)
    CALL STDM       ;correct location of 7seg chars
    CALL DCD            ;output into 7seg display
    POP D           ;retrieve D
    JMP CONT            ;jump to continue to finish intreupt handling

END
```

Θεωρητικές Ασκήσεις:

3η Άσκηση

a) Φτιάχνουμε μια μακροεντολή που να αυξάνει κατά 1, ένα δεκαεξάμπιτο αριθμό αποθηκευμένο σε 2 διαδοχικές θέσεις στην μνήμη.

```
MACRO INR16 ADDR
PUSH PSW       ;pwd into stack
PUSH H         ;h into stack
LXI H,ADDR     ;mov addr into h-l register pair
MOV A,M        ;get memory content into A
INR M          ;increase memory content
CMP 00H        ;check if overflow occured
JZ INCR_MSB    ;if yes increment also the next digit
JMP L1         ;if no go to l1 to exit

INCR_MSB:
INX H          ;increase pointer to memory
INR M          ;increaase memory content

L1:
POP H          ;retrieve H
POP PSW        ;retireve PWD
ENDM           ;end macro
```

Το μάκρο αυξάνει κατά 1 την τιμή της πρώτης θέσης στην μνήμη και ελέγχει αν συνέβη overflow. Αν συνέβη, αυξάνει και την τιμή στην επόμενη θέση μνήμης κατά 1.

b) Φτιάχνουμε μια μακροεντολή η οποία γεμίζει ένα τμήμα της μνήμης (μεγέθους K) με τους αριθμούς K, K-1, K-2, …, 1. Το μέγεθος είναι μεταξύ 1 και 256.

```
MACRO FILLD ADDR, K
     PUSH PSW
     PUSH B
     PUSH H
     PUSH L
     MVI B, K
     LXI H,ADDR
LOOP:
     MOV M,B
     INX H
     DCR B
     JZ END
END:
     POP L
     POP H
     POP B
     POP PSW
```

Το μάκρο αποθηκεύει προσωρινά στον B την τιμή K και τρέχει έναν βρόχο που αποθηκεύει την τιμή του K όπου δείχνει ο H-L, αυξάνει κατά 1 τον pointer στην μνήμη, μειώνει κατά 1 το B και επαναλαμβάνει όσο το B δεν είναι 0.

c) Φτιάχνουμε μια μακροεντολή η οποία περιστρέφει κατά μία θέση αριστερά τα περιεχόμενα ενός «εικονικού» καταχωρητή 17 bit που αποτελείται από τους καταχωρητές CY-Q-R.

```
MACRO RHLL Q,R
MOV A,R   ;move R into A
RLC       ;left shift it (it took CY as LSB and given its MSB to CY)
MOV R,A   ;store R back into R
MOV A,Q   ;move Q into A
RLC       ;left shift it (it took CY as LSB (R's MSB) and given its MSB to CY)
MOV Q,A   ;store A back int Q
ENDM
```

Εδώ φορτώνουμε τον R στο A και τον ολισθαίνουμε αριστερά 1 θέση. Άρα έχει πάρει το CY για LSB και το MSB του το έχει δώσει στο CY. Γυρνάμε τον A πίσω στον R. Κάνουμε την ίδια διαδικασία για το Q και τώρα έχει πάρει για LSB του το CY που είχε δώσει ο R και έχει δώσει το δικό του MSB στο CY.

4η Άσκηση

Η RST6.5 είναι hardware διακοπή και θα πάρει προτεραιότητα έναντι της εντολής JMP 2200H, εφόσον οι διακοπές είναι ενεργοποιημένες και η μάσκα διακοπών επιτρέπει την συγκεκριμένη διακοπή.

- Αρχικά το PC είναι ίσο με 2000H και το SP ίσο με 3000H
- Πραγματοποιείται η εντολή JMP 2200H άρα το PC γίνεται ίσο με 2200H ενώ το stack pointer δεν αλλάζει αφού έχουμε ένα απλό jump και όχι κάποια κλήση για να μας χρειάζεται η διεύθυνση επιστροφής
- Έρχεται η διακοπή RST6.5 και η τρέχουσα τιμή του PC αποθηκεύεται στην στοίβα. Πιο συγκεκριμένα ο stack pointer γίνεται 2FFEH, δηλαδή αυξάνεται κατά δύο θέσεις, στις οποίες μπαίνει η high και low τιμή του PC. Άρα (2FFF) = 22H και (2FFE) = 00H.
- Ο PC παίρνει την διεύθυνση εξυπηρέτησης αυτής της διακοπής, δηλαδή την τιμή 0034H.
- Μόλις τελειώσει η ρουτίνα εξυπηρέτησης ο PC θα ανακτήσει την προηγούμενη τιμή του από το stack, δηλαδή PC = 2200H
- Ο SP θα αυξηθεί κατά 2 ώστε να αποδεσμευτεί ο χώρος που είχε δεσμευτεί για να αποθηκευτεί προσωρινά το PC
- Η εκτέλεση θα συνεχίσει από την θέση 2200^H

5η Άσκηση

a) Στην άσκηση αυτή θα γράψουμε ένα πρόγραμμα το οποίο θα λαμβάνει 16 δεδομένα των 8 μπιτ από μια συσκευή, από την θύρα 20^H. Θα τα λαμβάνει σε κομμάτια των 4 μπιτ και θα ενημερώνεται ότι αυτά είναι έτοιμα να τα παραλάβει μέσω μια διακοπής RST5.5. Αφού λάβει όλα τα δεδομένα, πρέπει να υπολογίσουμε τον μέσο όρο τους.

Η ροή του προγράμματος:

Κάνουμε τις αρχικοποιήσεις (αριθμό μνημάτων που περιμένουμε να λάβουμε, ενεργοποίηση διακοπών) και μετά μπαίνουμε σε ένα κυρίως πρόγραμμα που απλά τσεκάρουμε ατέρμονα αν έχουμε λάβει και τα 32 κομμάτια πληροφορίας. Όταν έρθει διακοπή πάμε ασύγχρονα στην ρουτίνα εξυπηρέτησης της, μέσα στην οποία τσεκάρουμε αν περιμένουμε τα 4 LSB ή τα 4 MSB ενός από τους 16 8μπιτους αριθμούς. Ανάλογα την περίπτωση διαχειριζόμαστε τα 4 ληφθέντα μπιτ διαφορετικά. Αν είναι τα MSB (θεωρούμε

πως πρώτα έρχονται τα LSB) πάμε και αθροίζουμε τον ολόκληρο πια 8μπιτο αριθμό στο ζεύγος καταχωρητών H-L. Η ρουτίνα εξυπηρέτησης την διακοπής μόλις λάβει τα δεδομένα και τα αποθηκεύσει επιστρέφει στην κύρια συνάρτηση που ελέγχει ατέρμονα αν έχουμε λάβει όλα τα δεδομένα. Μόλις τα λάβουμε υπολογίζει το 4*(H-L) το οποίο ισοδυναμεί με 4 shift αριστερά. Αυτό μας επιτρέπει να πάρουμε στον H τα 4 MSB του L και τα 4 LSB του H. Άρα στον H, αυτή την στιγμή, έχουμε το αρχικό άθροισμα shifted 4 θέσεις δεξιά, διαιρεμένο δηλαδή με το 16. Άρα έχουμε τον μέσο όρο των 16 αυτών αριθμών.

Ο κώδικας:

```
IN 10H          ;unlock memory
MVI A,0EH           ;load in A appropriate interupt mask
SIM             ;move A into interupt mask
MVI C,20H           ;C=32d (32 steps)
EI              ;enable interupt

ADDR:
    MOV A,C         ;move remaining steps into A
    CPI 00H     ;check if steps have finished
    JNZ ADDR        ;if not, go to ADDR to wait for a interupt
    DI          ;disable interupt to calculate average
    DAD H       ;4 double additions equals to 4 left shifts
    DAD H       ;so now H has the 8 bit result (the mean value)
    DAD H
    DAD H
    MOV A,H     ;put result into A

002C:
    JMP RST5.5 ;jump to interupt handler

RST5.5:
    PUSH PSW    ;store state
    MOV A,C     ;move C into A to check remaining steps
    ANI 01H     ;if last digit is 1 then we have odd numbers
                ;of remaining steps so we have to get msb
    JZ GET_4LSB
    JMP GET_4MSB
GET_4LSB:
    IN 20H      ;get input
    ANI F0H     ;isolate x7-x4
    RRC         ;4 right shifts to make them LSB
    RRC
    RRC
    RRC
    MOV B,A     ;move them into B to store them temporarily
                ;until we get MSBs
    JMP RETURN  ;return from interupt handling
GET_4MSB:
    IN 20H      ;get input
    ANI F0H     ;isolate x7-x4
    ORA B       ;add them to 4 LSBs
    MVI D,00H       ;zeros into D so that it does not interfere
                ;with the addition
    MOV E,A     ;move them into E
    DAD D       ;add D-E into H-L
RETURN:

    DCR C       ;decrease number of remaining steps
    POP PSW     ;retrieve state
    EI          ;re-enable interupts
    RET         ;return from interupt handling
```

b) Παραποιούμε τον προηγούμενα κώδικα ώστε ο μικροεπεξεργαστής να ενημερώνεται για έτοιμα δεδομένα με το LSB της θύρας 20^H (ενεργό χαμηλά).

Η ροή του προγράμματος:

Η αλλαγή στον κώδικα είναι ότι ο κορμός του προγράμματος ελέγχει πότε θα πέσει στο 0 το x0 και όταν γίνει αυτό πάει και τρέχει το ίδιο κομμάτι κώδικα που έτρεχε και στο προηγούμενο ερώτημα η εξυπηρέτηση της διακοπής. Απλά τώρα δεν είναι ρουτίνα αλλά ένα απλό branch. Στο τέλος του διαβάσματος των 4 ληφθέντων μπιτ ελέγχουμε αν έχουμε λάβει και τους 16 αριθμούς. Αν ναι πάμε να υπολογίσουμε τον μέσο όρο, αλλιώς πάμε να περιμένουμε τον επόμενο αριθμό. Αφού επιστρέψουμε στην κεντρική ροή, ελέγχουμε ατέρμονα αν το x0 επανήλθε από 0 στο 1. Όταν συμβεί αυτό πάμε και περιμένουμε να πέσει πάλι στο 0 (πότε δηλαδή θα έρθει το επόμενο πακέτο δεδομένων). Αυτό προϋποθέτει ότι ο μικροεπεξεργαστής διαβάζει και διαχειρίζεται το πακέτο δεδομένων και επιστρέφει στο κεντρικό πρόγραμμα γρηγορότερα από τον ρυθμό που στέλνει δεδομένα η εξωτερική συσκευή. Ο υπολογισμός του μέσο όρου γίνεται με τον ίδιο τρόπο με το α) ερώτημα.

Ο κώδικας:

```
IN 10H          ;unlock memory
MVI A,0EH       ;load in A appropriate interupt mask
SIM             ;move A into interupt mask
MVI C,20H       ;C=32d (32 steps)
EI              ;enable interupt

MAIN:
    IN 20H      ;get input
    ANI 01H     ;isolate x0
    JNZ MAIN    ;wait until x0 = 0
    JMP GET_DATA
GOT_DATA:
    IN 20H      ;get input
    ANI 01H     ;isolate x0
    JZ GOT_DATA ;loop in here until x0 goes back to 1
    JMP MAIN    ;if it goes to 1 go to main to wait for it
                ;to go to 0 again
GET_RESULTS
    DAD H       ;4 double additions equals to 4 left shifts
    DAD H       ;so now H has the 8 bit result (the mean value)
    DAD H
    DAD H
    MOV A,H     ;put result into A

GET_DATA:
    MOV A,C     ;move C into A to check remaining steps
    ANI 01H     ;if last digit is 1 then we have odd numbers
                ;of remaining steps so we have to get msb
    JZ GET_4LSB
    JMP GET_4MSB
GET_4LSB:
    IN 20H      ;get input
    ANI F0H     ;isolate x7-x4
    RRC         ;4 right shifts to make them LSB
    RRC
    RRC
    RRC
    MOV B,A     ;move them into B to store them temporarily
                ;until we get MSBs
    JMP RETURN  ;return from interupt handling
GET_4MSB:
    IN 20H      ;get input
    ANI F0H     ;isolate x7-x4
    ORA B       ;add them to 4 LSBs
    MVI D,00H   ;zeros into D so that it does not interfere
                ;with the addition
    MOV E,A     ;move them into E
    DAD D       ;add D-E into H-L
RETURN:

    DCR C       ;decrease number of remaining steps
    JZ  GET_RESULTS;if we got every number goto calculate mean
    JMP GOT_DATA;return from interupt handling
```