Συστήματα Μικρουπολογιστών

Εργαστηριακή άσκηση 1

Ομάδα 12

Κυπριανίδης Αλέξανδρος 8012

Μάστορας Ραφαήλ Ευάγγελος 7918

Τμήμα 1

Αρχικά εισάγαμε με την μορφή δεκαεξαδικού, ακολουθώντας κωδικοποίηση Ascii, τα ΑΕΜ μας στην μνήμη του προγράμματος. Το aem1 αντιστοιχεί στο 8012 και το aem2 στο 7918, καθώς ζητήθηκε να καταχωρηθούν με αλφαβητική σειρά. Επιπλέον για λόγους ευκολίας ορίσαμε τους καταχωρητές R17-R24 να αντιστοιχούν στα aem1a-aem1d και aem2a-aem2d, τα οποία αντιπροσωπεύουν τα αντίστοιχα ψηφία του κάθε ΑΕΜ. Μετά αρχικοποιήσαμε τον Stack Pointer, ούτως ώστε να αποθηκευτούν σωστά οι διαδικασίες. Έπειτα ανακτήσαμε από την μνήμη του προγράμματος τα ψηφία των ΑΕΜ και τα περάσαμε μέσα στις παραπάνω μεταβλητές. Σε επόμενο βήμα μετατρέψαμε τα εκάστοτε ψηφία των ΑΕΜ από δεκαεξαδικά σε BCD αφαιρώντας την τιμή 0x30. Αμέσως μετά κάναμε κλήση της συνάρτησης compare η οποία συγκρίνει τα δύο ΑΕΜ και αν το aem1 είναι μεγαλύτερο του aem2 καλεί την συνάρτηση led, αλλιώς καλεί την συνάρτηση led2. Η συνάρτηση led αφού θέσει τον καταχωρητή R25 ίσο με 1, (ώστε να γνωρίζουμε αργότερα οτι aem1>aem2 και να το χρησιμοποιήσουμε όπου χρειαστεί) με διαδοχικά shift και την χρήση μιας OR εμφανίζει στα LED 7-4 το τρίτο ψηφίο του μεγαλύτερου ΑΕΜ, ενώ στα LED 3-0 το τέταρτο ψηφίο του μεγαλύτερου ΑΕΜ. Καλεί επίσης μια συνάρτηση delay 10 δευτερολέπτων ώστε να είναι εμφανής η αλλαγή της κατάστασης των LED, ενώ αφού τελειώσει η delay επιστρέφει στο σημείο μετά την κλήση της compare. Η συνάρτηση led2 απλά θέτει την τιμή του καταχωρητή R25 ίση με 0 για μελλοντική χρήση και επιστρέφει πίσω στην τελευταία κλίση (compare). Σε εκείνο το σημείο καλείται η συνάρτηση III η οποία εξετάζει ποιο είναι το μεγαλύτερο ΑΕΜ και ανάλογα καλεί τις maxaem1 ή maxaem2 με την χρήση των breq kai brne αντίστοιχα. Στις συναρτήσεις αυτές απομονώνουμε το τελευταίο bit των δύο ΑΕΜ με την χρήση μασκών και δυαδικών πράξεων και ανάλογα αν τα ΑΕΜ είναι περιττά ή άρτια ανάβουν τα ζητούμενα από την άσκηση LED 1 και 0. Τέλος, οι συναρτήσεις αυτές επιστρέφουν στην τελευταία κλήση (δηλαδή μετά την κλήση της συνάρτησης III) και αφού σβήσουν τα LED καλούν την συνάρτηση part2 που θα εξηγηθεί παρακάτω.

Δυσκολία συναντήσαμε κυρίως στο κομμάτι που έπρεπε να περαστούν στην μνήμη του προγράμματος με μορφή ASCII τα ΑΕΜ μας, καθώς και στο κομμάτι

της μετατροπής των δεκαεξαδικών τιμών σε BCD. Επιπλέον στο σημείο που έπρεπε να κατηγοριοποιήσουμε τα ΑΕΜ ως άρτια ή περιττά παρουσιάστηκε δυσκολία.

Τμήμα 2

Στο $2^{ο}$ μέρος του προγράμματος ορίζουμε ως έξοδο το port D στο οποίο είναι συνδεδεμένοι οι διακόπτες. Στην συνέχεια δημιουργούμε έναν ατέρμων βρόγχο, στον οποίον περιμένουμε να πατηθεί ένας από τους διαθέσιμους διακόπτες(SW0-SW3,SW7). Στην περίπτωση που επιλεγεί ένας εκ των 4 διακοπτών με δείκτες 0-3 μεταφερόμαστε σε ένα κομμάτι του προγράμματος περιμένοντας να απελευθερωθεί ο διακόπτης. Με την απελευθέρωση του διακόπτη ελέγχουμε ποιο από τα διαθέσιμα ΑΕΜ είναι μεγαλύτερο και οδηγούμαστε στην κατάλληλη περίπτωση. Αν το ΑΕΜ1 είναι μεγαλύτερο του ΑΕΜ2 τότε αντιγράφουμε το πρώτο εκ των δύο ψηφίων, που προκύπτουν ανάλογα την περίπτωση, στον καταχωρητή r28. Με 4 διαδοχικές αριστερές μετατοπίσεις του καταχωρητή οδηγούμε τα bit που μας ενδιαφέρουν στα 4 μεγαλύτερα bit του καταχωρητή. Έτσι με χρήση της OR ανάμεσα στον r28 και στον καταχωρητή, που είναι αποθηκευμένο το άλλο ψηφίο, καταφέρνουμε να συνδυάσουμε σε ένα καταχωρητή τα 2 ψηφία. Στη συνέχεια ανάβουμε τα led με την γνωστή διαδικασία. Αυτή η υλοποίηση ακολουθείτε και στην περίπτωση που το ΑΕΜ2 είναι μεγαλύτερο του ΑΕΜ1, απλά χρησιμοποιώντας τα ψηφία του ενδιαφερόμενου ΑΕΜ. Στην περίπτωση που πατηθεί ο διακόπτης 7, αφού περιμένουμε να απελευθερωθεί, καλούμε την συνάρτηση III που υλοποιήθηκε στο πρώτο τμήμα, με πρόβλεψη ώστε να μην σβήνουν τα led. Τέλος, το πρόγραμμα απελευθερώνει τις μεταβλητές που ορίσαμε για τα ψηφία των ΑΕΜ και οδηγείται στον ατέρμον βρόγχο end. Στο συγκεκριμένο τμήμα η μεγαλύτερη δυσκολία παρατηρήθηκε στο διαχωρισμό των εκάστοτε περιπτώσεων, που έπρεπε να υλοποιηθούν, για κάθε διακόπτη. Επιπλέον ήταν αρκετά απαιτητική η υλοποίηση του πρώτου ατέρμονα βρόγχου καθώς θέλαμε να δίνεται η δυνατότητα να επιλεγεί ένας εκ των διαθέσιμων διακοπτών.

Breakpoints

Για τον επιτυχή έλεγχο του προγράμματος χρησιμοποιήσαμε ποικίλα breakpoints. Τοποθετήσαμε στο σημείο που διαβάζουμε τα ΑΕΜ από την μνήμη του προγράμματος καθώς και στο σημείο που τα μετατρέπουμε σε μορφή bcd. Επίσης βάλαμε στα διάφορα σημεία που εμφανίζονται τα led ώστε να ελέγχουμε κάθε φορά άμα ανάβουν τα προβλεπόμενα led.

Κώδικας Προγράμματος

```
.include "m16def.inc"
.org 0

.cseg


table:
.db 0x38,0x30,0x31,0x32                ;AEM1=8012 (KYPRIANIDIS)
.db 0x37,0x39,0x31,0x38                ;AEM2=7918 (MASTORAS)

.def aem1a=r17
.def aem1b=r18
.def aem1c=r19
.def aem1d=r20                         ;define variables for better use
.def aem2a=r21
.def aem2b=r22
.def aem2c=r23
.def aem2d=r24

sp_init:
        ldi r16,low(RAMEND)            ;stack pointer initialization
        out spl,r16
        ldi r16,high(RAMEND)
        out sph,r16



rjmp part1                             ;start 1st part

part1:
        ldi ZL,LOW(2*table)            ;seperate z memory poiner into high and low
        ldi ZH,HIGH(2*table)
        lpm                            ;read lsb from program memory
        mov aem1a,r0                   ;transfer r0 to aem1a
        adiw ZL,1                      ;add 1 to low bit (Zl) in order to take the next bit
from r0
        lpm
        mov aem1b,r0
        adiw ZL,1
        lpm
        mov aem1c,r0
        adiw ZL,1
        lpm
```

```
        mov aem1d,r0

        adiw ZL,1
        lpm
        mov aem2a,r0
        adiw ZL,1
        lpm
        mov aem2b,r0
        adiw ZL,1
        lpm
        mov aem2c,r0
        adiw ZL,1
        lpm
        mov aem2d,r0

        ;convert from ascii to bcd
        subi aem1a,0x30
        subi aem1b,0x30
        subi aem1c,0x30
        subi aem1d,0x30
        subi aem2a,0x30
        subi aem2b,0x30
        subi aem2c,0x30
        subi aem2d,0x30

        rcall compare                   ;calls the 1st function in order to compare the
AEM's values
        ser r16                         ;turn off leds
        out PORTB,r16
        rcall III                   ;calls the  III function which has even and odd compares
        ser r16
        out PORTB,r16
        rjmp part2                      ;go to part 2



compare:


        cp aem2a,aem1a                  ;if aem2a<aem1a brlo becomes available and go
to led function
        brlo led                        ;else brne becomes available and go to led2
function
        brne led2
        cp aem2b,aem1b
```

```
        brlo led
        brne led2
        cp aem2c,aem1c
        brlo led
        brne led2
        cp aem2d,aem1d
        brlo led
        brne led2




led:
        ldi r25,1                       ;define r25=1 if aem1>aem 2
        ser r16                         ;initiallize
        out DDRB,r16

        ser r16
        out PORTB,r16
        ;led 7-4 aem1
        mov r28,aem1c
        lsl r28                 ;shift left 4 times in order to turn on 7-4 led
        lsl r28
        lsl r28
        lsl r28
        or r28,aem1d                    ;combine of aem1c and aem1d after shifting
        com r28
        out PORTB,r28
        rcall delay
        ret



led2:
        ldi r25,0                       ;if aem2>aem1 r1 =0
        ret

lll:
        cpi r25,1               ;if r25=1 (aem1>aem2) breq will be available else brne
will be
        breq maxaem1
        brne maxaem2
```

```
maxaem1:
        mov r28,aem1d
        andi r28,0b00000001             ;mask in order to insulate last bit
        adiw r28,1
        andi r28,0b00000001
        lsl r28                         ;shift it one bit left because we want it on 1 led
        mov r27,r28

        mov r28,aem2d
        andi r28,0b00000001
        adiw r28,1
        andi r28,0b00000001
        or r28,r27                      ;combine the states of leds into one byte

        ser r16
        out PORTB,r16
        com r28                                 ;turn on leds using byte
        out PORTB,r28
        rcall delay
        ret

maxaem2:
        mov r28,aem2d                           ;same with maxaem1, if aem2>aem1
        andi r28,0b00000001
        adiw r28,1
        andi r28,0b00000001
        lsl r28
        mov r27,r28

        mov r28,aem1d
        andi r28,0b00000001
        adiw r28,1
        andi r28,0b00000001
        or r28,r27

        ser r16
        out PORTB,r16
        com r28
        out PORTB,r28
        rcall delay
        ret
```

```
;delay:

        ;ldi r31,0x2
                                        ;we used this for debugging
;d7:                                    ;so we dont have to waste time on more delay
steps
        ;dec r31
        ;brne d7
        ;ret


delay:
        ldi r31, 0xed
        d3:
        ldi r30, 0xed                   ;237*237*238 =~10sec (cycle time =
0.25ìs)
        d2:
        ldi r29, 0xee
        d1:
        dec r29
        brne d1
        dec r30
        brne d2
        dec r31
        brne d3
        ret

part2:                                  ;start part 2
        clr r16
        out DDRD,r16
        ser r16                         ;initialization of switches
        out PIND,r16

switch_unpressed:                       ;its an infinite loop until one switch is pressed
        sbis PIND,0x00                  ;if(Port D,pin0=0) then dont execute next
command
        rjmp sw0_pressed                ;calls the sw0_pressed function
        sbis PIND,0x01
        rjmp sw1_pressed
        sbis PIND,0x02
        rjmp sw2_pressed
```

```
        sbis PIND,0x03
        rjmp sw3_pressed
        sbis PIND,0x07
        rjmp sw7_pressed
        rjmp switch_unpressed

sw0_pressed:
        sbis PIND,0x00                  ;check till switch0  is unpressed
        rjmp sw0_pressed
                                        ; if released continue
        cpi r25,1                       ; if r25=1 (aem1>aem2) then breq will be called
        breq max1min2sw0                ;else brne will be called
        brne max2min1sw0

max1min2sw0:
        mov r28,aem2c                   ;copy aem2c to r28
        lsl r28                         ;shift 4 times left r28 in order to combine
        lsl r28                         ;the two last digits
        lsl r28
        lsl r28
        or r28,aem2d                    ;combination of last digits
        ser r16
        out PORTB,r16
        com r28                                 ;turn on leds
        out PORTB,r28
        rcall delay
        rjmp free                       ;calls free

max2min1sw0:
        mov r28,aem1c
        lsl r28
        lsl r28                         ;same as before , if aem2>aem1
        lsl r28
        lsl r28
        or r28,aem1d
        ser r16
        out PORTB,r16
        com r28
        out PORTB,r28
        rcall delay
        rjmp free


sw1_pressed:
        sbis PIND,0x01
```

```
        rjmp sw1_pressed

        cpi r25,1
        breq max1min2sw1
        brne max2min1sw1



max1min2sw1:
        mov r28,aem2a
        lsl r28
        lsl r28
        lsl r28
        lsl r28
        or r28,aem2b
        ser r16
        out PORTB,r16
        com r28
        out PORTB,r28
        rcall delay
        rjmp free



max2min1sw1:
        mov r28,aem1a
        lsl r28
        lsl r28
        lsl r28
        lsl r28
        or r28,aem1b
        ser r16
        out PORTB,r16
        com r28
        out PORTB,r28
        rcall delay
        rjmp free

sw2_pressed:
        sbis PIND,0x02
        rjmp sw2_pressed

        cpi r25,1
        breq max1min2sw2
```

```
        brne max2min1sw2

max1min2sw2:
        mov r28,aem1c
        lsl r28
        lsl r28
        lsl r28
        lsl r28
        or r28,aem1d
        ser r16
        out PORTB,r16
        com r28
        out PORTB,r28
        rcall delay
        rjmp free

max2min1sw2:
        mov r28,aem2c
        lsl r28
        lsl r28
        lsl r28
        lsl r28
        or r28,aem2d
        ser r16
        out PORTB,r16
        com r28
        out PORTB,r28
        rcall delay
        rjmp free

sw3_pressed:
        sbis PIND,0x03
        rjmp sw3_pressed

        cpi r25,1
        breq max1min2sw3
        brne max2min1sw3

max1min2sw3:
        mov r28,aem1a
        lsl r28
        lsl r28
        lsl r28
        lsl r28
        or r28,aem1b
```

```
        ser r16
        out PORTB,r16
        com r28
        out PORTB,r28
        rcall delay
        rjmp free

max2min1sw3:
        mov r28,aem2a
        lsl r28
        lsl r28
        lsl r28
        lsl r28
        or r28,aem2b
        ser r16
        out PORTB,r16
        com r28
        out PORTB,r28
        rcall delay
        rjmp free

sw7_pressed:
        sbis PIND,0x07
        rjmp sw7_pressed

        rcall III
        rjmp free


free:
        .undef aem1a                    ;undefine the variables
        .undef aem1b
        .undef aem1c
        .undef aem1d
        .undef aem2a
        .undef aem2b
        .undef aem2c
        .undef aem2d
        rjmp end                        ;calls end

end:
        rjmp end                        ;infinite loop
```