ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣ/ΝΙΚΗΣ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 2

GROUP 3, ΟΜΑΔΑ 5

Παπαδόπουλος Κωνσταντίνος (8677)

Τοπαλίδης Ευθύμιος (8417)

Ζητούμενα

Α)

➔ Λειτουργία του προγράμματος (αλγόριθμος και συγκεκριμένα βήματα που απαιτούνται):
Η υλοποίηση του προγράμματος ακολούθησε τη σχεδιαστική πορεία που καθόριζε η εκφώνηση της άσκησης αυτής καθαυτής.
Σημεία που θα έπρεπε να τονίσουμε, εκτός των σχολίων υπάρχουν στον πηγαίο κώδικα είναι:

- Η *rjmp RESET* είναι η πρώτη εντολή του κώδικα, ώστε ο PC να διαβάσει πράγματι εντολές και όχι άλλα τυχόν δεδομένα.

- Η αρχικοποίηση του stack pointer.

- Η αρχικοποίηση (μία φορά) των PORT εξόδου και εισόδου για τα LED και SWITCH αντίστοιχα.

- Ο έλεγχος πίεσης ενός πλήκτρου ή όχι γίνεται με την κλασική μεθοδολογία (σελ. 154 Προγραμματίζοντας τον Μικροελεγκτή AVR).

- Για την αποθήκευση της σειράς των πλήκτρων, πολύ απλά τα αποθηκεύουμε, δηλαδή αποθηκεύουμε τον δυαδικό αριθμό που αντιστοιχεί στο άναμμα του κατάλληλου LED όταν αυτό δοθεί σαν όρισμα στο *out PORTB, sw,* σε διαδοχικές θέσεις στη μνήμη (στην SRAM αν και στη συνέχεια αναφέρουμε και άλλη υλοποίηση).

- Για τον έλεγχο αν πατήθηκε παραπάνω από μία φορές ένα πλήκτρο υλοποιήσαμε την εξής τεχνική: φτιάξαμε μια μάσκα με 11111111 η οποία γέμιζε με 0 στην αντίστοιχη θέση που είχε πατηθεί το πλήκτρο, ελέγχαμε αν το *or* της μάσκας με το πλήκτρο που πατήθηκε ήταν μεγαλύτερο ή όχι από την μάσκα και έτσι ξέραμε αν είχε πατηθεί (για την καλύτερη κατανόηση του αλγορίθμου που περιγράφεται βοηθάει η ανάγνωση του κώδικα).

➔ Κώδικας με σχόλια και καθορισμός της ονομασίας των μεταβλητών που χρησιμοποιούνται:
Παρακάτω δίνεται το ζητούμενο πρόγραμμα με αναλυτικά σχόλια, στα σημεία που απαιτείται, για την κατανόηση λειτουργίας του κώδικα.

➔ Δυσκολίες που αντιμετωπίσαμε:
Η εύρεση ενός τρόπου αποθήκευσης της σειράς με την οποία πατήθηκαν τα πλήκτρα, καθώς και το να αγνοείται αν πατηθεί δεύτερη φορά το ίδιο πλήκτρο.

➔ Διαφορές στην αποσφαλμάτωση στον προσομοιωτή και στην αναπτυξιακή κάρτα:
Στον προσομοιωτή δεν διαπιστώσαμε την λάθος θέση αποθήκευση κάποιων εκ των δεδομένων μας στην SRAM (data memory). Επίσης, χρειάστηκε μια μικρή αλλαγή της τοποθέτησης της υπο-ρουτίνας *blink* που ήταν υπεύθυνη για να αναβοσβήνουν τα LED.

➔ Επιπρόσθετα σχόλια για την παραπέρα βελτίωση του κώδικα που αναπτύχθηκε:
Στο τέλος της άσκησής μας διαπιστώσαμε ότι αντί να αποθηκεύουμε στην SRAM (data memory) τη σειρά των πλήκτρων που πατήθηκαν αυξάνοντας έτσι το χρόνο εκτέλεσης, μπορούμε να την αποθηκεύουμε στους καταχωρητές εργασίας, αναφερόμενοι στη θέση μνήμης τους (με τον καταχωρητή Z) ως 0, 1, 2 ... για τους καταχωρητές R0, R1, R2 … αντίστοιχα.

Β)

Σημεία εισαγωγής breakpoint ώστε η παρακολούθηση διαδοχικών λειτουργιών να είναι πιο αποδοτική:

- Μετά το πάτημα του διακόπτη SW0

- Μετά το πάτημα και των 7 διακοπτών, για να διαπιστώσουμε αν αποθηκεύτηκαν στη σωστή θέση μνήμης τα σωστά δεδομένα από την ενέργεια αυτή.

- Αντίστοιχα μετά το πάτημα ενός συγκεκριμένου SW για να ελέγχουμε την κατάσταση του προγράμματος.

```
;ergasia2.asm
;Papadopoulos Konstantinos, AEM 8677
;Topalidhs Efthymis, AEM 8417
;Group 3, Team 5
;Purpose: ...

.include "m16def.inc"


.cseg  ;tells the assembler that the following code is to be put into program memory
          ;This is necessary when the .dseg directive was used before

;.org $100    ;0x0100 ;set program memory address counter to 0x0100,
                        ;set the program counter to a specific value



rjmp RESET         ; ----first command of program

;---defining aliases for registers
.def mask = r17     ;the mask for checking if switch has been pressed
.def swcounter = r18       ;how many switches have been pressed until now
.def sw1 = r8         ;switch that is pressed first
.def sw2 = r9         ;switch that is pressed second and so on...
.def sw3 = r10       ;never actually used
.def sw4 = r11       ;never actually used
.def sw5 = r12       ;never actually used
.def sw6 = r13       ;never actually used
.def sw7 = r14       ;never actually used
;---


RESET:                                        ; set initial value of stack pointer
        ldi r16, low(RAMEND)

        out SPL,r16

        ldi r16, HIGH(RAMEND)
        out SPH, r16
        rjmp main
```

```
main:

;initialization code
;---setting PORTB as output port
ldi r16, 0b11111111
out DDRB, r16              ;The data direction register of port B is named DDRB

;---setting PORTD as input port
ldi r16, 0b00000000
out DDRD, r16

;---setting Z register to show address (beginning from 0x60)
ldi ZL, 0x66
ldi ZH, 0x00


;main program
;--A) waiting loop, checking as flag SW0
get_switch0:
        in r16, PIND        ;copy state of SW0 to PORTD (when pressed is 0b11111110)
        andi r16,$0F        ;clear upper nibble of r16 (0b00001111)
        cpi r16, 0b00001110       ;checks if SW0 is pressed
        brne get_switch0

;--B) wait for every switch to be pressed at least once and save the order they are pressed
;          if a switch is pressed more than once save only the first value
;          when all sw are pressed blink with 1sec frequency
ldi mask, 255                     ;make mask 0b11111111
ldi swcounter, 8     ;SW must be pressed 7 times
get_switch_all:
        in r16, PIND
        mov r15,r16         ;save initial state of r16, input of SW
        or r15, mask;logic or r15 with mask
        cp r15, r16         ;if r15 is equal to r16 the SW has already been pressed before
        breq get_switch_all       ;return and wait for other switch
        and mask, r16       ;updating mask by puting zero in the right bit if SW is pressed
        st Z+, r16          ;saving SW pressed in DATA MEMORY
        dec swcounter;
        brne get_switch_all       ;if swcounter is 0 then all 7 SW have been pressed
```

```
;--C)
;Z register shows address and decreases by one to get all SW values from DATA
MEMORY


get_switch:
        rcall blink
        in r22, PIND        ;copy state of SW0 to PORTD (when pressed is 0b11111110)
        andi r22,$0F        ;clear upper nibble of r16 (0b00001111)
        cpi r22, 0b00001101     ;checks if SW1 is pressed
        breq SW_1
        cpi r22, 0b00001011     ;checks if SW2 is pressed
        breq SW_2
        cpi r22, 0b00000111     ;checks if SW3 is pressed
        breq SW_3
        brne get_switch         ;if none is pressed keep waiting


SW_1:

                ldi swcounter, 7    ;all 7 LEDs must light
                ldi ZL, 0x66        ;starts again at the beggining of where the SWs were
stored
                inc ZL                          ;first SW is stored in Z+1
                ld sw1, Z+
                out PORTB, sw1
check:          in r22, PIND
                andi r22,$0F
                cpi r22, 0b00001011     ;checks if SW2 is pressed
                breq SW_2
                brne check


SW_2:
        in r22, PIND        ;copy state of SW0 to PORTD (when pressed is 0b11111110)
        andi r22,$0F        ;clear upper nibble of r16 (0b00001111)
        cpi r22, 0b00000111     ;checks if SW3 is pressed
        breq SW_3
        ld sw2, Z+              ;it is sw2 only the first time, then it is sw3, sw4...
        out PORTB, sw2
        rcall delay5calc
        dec swcounter           ;counts how many lights have left to light
```

```
        breq SW_4
        brne SW_2



SW_3:
        rcall blink
        in r22, PIND        ;copy state of SW0 to PORTD (when pressed is 0b11111110)
        andi r22,$0F        ;clear upper nibble of r16 (0b00001111)
        cpi r22, 0b00001011     ;checks if SW2 is pressed
        breq SW_2
        cpi r22, 0b00001101     ;checks if SW1 is pressed
        breq SW_1
        brne SW_3               ;if none is pressed keep blinking


;--D) the program has ENDED, open all LEDs
SW_4:
        ldi r22, 0b00000000
        out PORTB, r22    ;program has ended, all the lights are on

end:
        rjmp end      ;end of program



;--subroutine that keeps lights open for 1sec and then closed for 1sec
blink:
        ldi r16, 0 ; open led lights for 1 sec (space betwenn characters)
        out PORTB, r16
        rcall delay1calc
        ldi r16, 255 ; close led lights for 1 sec (space betwenn characters)
        out PORTB, r16
        rcall delay1calc
ret



; delay subroutine
```

```
; Delay 3 999 997 cycles
; 999ms 999us 250 ns at 4.0 MHz -- approx 1 sec
;--as seen in the calculator
delay1calc:
                ldi  r19, 21
                ldi  r20, 75
                ldi  r21, 190
L1:             dec  r21
                brne L1
                dec  r20
                brne L1
                dec  r19
                brne L1
                nop

        ret



; Delay 19 999 997 cycles
; 4s 999ms 999us 250 ns at 4.0 MHz -- approx 5 sec
;--as seen in the calculator
delay5calc:
                  ldi  r19, 102
                  ldi  r20, 118
                  ldi  r21, 193
L2:               dec  r21
                  brne L2
                  dec  r20
                  brne L2
                  dec  r19
                  brne L2
        ret
```