ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 1

GROUP 3
ΟΜΑΔΑ 5

ΠΑΠΑΔΟΠΟΥΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ (8677)

ΤΟΠΑΛΙΔΗΣ ΕΥΘΥΜΙΟΣ (8417)

ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:

Αρχικά, αποθηκεύουμε σε μορφή Look Up Table τη δυαδική και BCD μορφή των AEM στην Program Memory. Πριν από αυτό προσδιορίζουμε την αρχή του program counter στη θέση 0X0100 με το προσδιοριστικό .org και με το .cseg προσδιορίζουμε την αρχή του κώδικα . Έπειτα δίνουμε ονόματα στους καταχωρητές ώστε να διακρίνουμε τη λειτουργία τους κατά την ανάγνωση του κώδικα Προσέχουμε ώστε ως πρώτη εντολή να εκτελεστεί αυτή του κυρίου προγράμματος που αρχίζει με τη RESET(εδώ έχουμε αρχικοποίηση του stack pointer) και στη συνέχεια με τη main (κυρίως πρόγραμμα).

ΤΜΗΜΑ1

Στην αρχή της main εκτελούμε δεικτοδοτούμενη προσπέλαση της program memory με τον καταχωρητή Z. Σε αυτό το σημείο χρειάζεται να προσέξουμε ότι επειδή κάνουμε προσπέλαση 16-Bit διευθύνσεων από 8-Bit καταχωτρητή χρειάζεται να πολλαπλασιάσουμε επί 16-Bit διεύθυνση του Look Up Table που έχουμε αποθηκεύση τα AEM . Αυτή ήταν και η πρώτη δυσκολία που αντιμετωπίσαμε για την ανάγνωση τιμών από την program memory με τη χρήση της lpm και post-increment του Z register. Έπειτα φωρτόνουμε στους αντίστοιχους καταχωρητές τα AEM έχοντας όμως κατά νου ότι τα δεδομένα στην program memory καταλαμβάνουν 16-bit και έχουμε στη διάθεσή μας 8-bit καταχωρητές για να τα χειριστούμε, για αυτό και χρησιμοποιούμε ζεύγη καταχωρητών.

Στην συνέχεια εμφανίζουμε τα δύο AEM σε μορφή BCD με τον κάθε χαρακτήρα να εμφανίζεται για 5sec και μεταξύ τους να μεσολαβεί διάστημα 3sec . Οι χρονικές καθυστερήσεις εισάγονται με τις συναρτήσεις delay που υλοποιήσαμε. Για την υλοποίηση των καθυστερήσεων (delays) δημιουργήσαμε τρεις εμφωλευμένους βρόχους επανάληψης μειώνοντας έναν μετρητή κάθε φορά έτσι ώστε ο συνολικός χρόνος εξόδου από την υπορουτίνα καθυστέρησης να είναι ίσος με τη ζητούμενη από την άσκηση καθυστέρηση, κάθε φορά. Υλοποιούμε τις συναρτήσεις delay σαν υπορουτίνες με σκοπό να αποφύγουμε την εγγραφή του ίδιου κώδικα στο κυρίως πρόγραμμα. Χρειάστηκε να αφιερώσουμε κάποιο χρόνο ώστε να κατανοήσουμε τον τρόπο λειτουργίας των LED και των διακοπτών. Ορίζουμε τη σύμβαση ώστε ο καταχωρητής PORTB να συνδέεται με την έξοδο (δηλαδή με τα LED) ,πράγμα το οποίο επιτυγχάνεται μέσω του καταχωρητή κατεύθυνσης της θύρας PORTB , του DDRB. Όταν φορτώσουμε στον καταχωρητή αυτό σε όλα τα bit του 1 τότε όλες οι θύρες του PORTB λειτουργούν σε κατεύθυνση write-έξοδοι. Αντίστοιχα σκεπτόμαστε και για την είσοδο PORTD που σθνδέεται με την είσοδο. Η ανάγνωση και εγγραφή σε αυτούς τους τους I/O καταχωρητές πραγματοποιείται με τις εντολές IN , OUT αντίστοιχα . Στη συνέχεια συγκρίνουμε τα δύο AEM με την εντολή cp(compare) ξεκινώντας από τα high bytes τους κι αν αυτή παράξει Z=0 στον status register τότε πηγαινουμε τον έλεγχο στην σύγκριση των low bytes.

Ακολούθως , χρειάστηκε να αντιμετωπίσουμε το πρόβλημα της ανάγνωσης

διακοπτών και της εμφάνισης στα led του AEM1-AEM2 αν πατήσουμε το SW7 και του AEM2-AEM1 αν πατήσουμε το SW6. Για την ανάγνωση διακοπτών χρησιμοποιήσαμε την εντολή in ώστε μέσα στον επαναληπτικό βρόχο να διαβάζουμε την κατάσαση της εισόδου-διακόπτες και να κάνουμε τις αντίστοιχες διακλαδώσεις μέσω εντολών branch στα label difaem1(AEM1-AEM2),difaem2(AEM2-AEM1) .Χρειαζόμαστε τον βρόχο αναμονής λόγω τις διαφοράς στις ταχύτητες εκτέλεσης του προγράμματος και χειρισμού των πλήκτρων κάτι που λαμβάνεται υπόψη και στο τμήμα2 της άσκησης.

ΤΜΗΜΑ2

Στο τμήμα2 χρειάζεται το πρόγραμμά μας να χειριστεί διαδοχικά πατήματα διακοπτών και μεταβάσεις στα αντίστοιχα τμηματα κώδικα για την εκτέλεση των λειτουργιών που περιγράφηκαν παραπάνω. Κι εδώ όπως προαναφέραμε χρησιμοποιούμε έναν επαναληπτικό βρόχο όπου διαβάζουμε συνεχώς την κατασταση των PIN εισόδου . Εδώ αντιμέτωπίσαμε το πρόβλημα που αφορά την εμβέλεια της branch εντολής καθώς προσπαθήσαμε να μεταφερθούμε σε labels που ήταν περα της εμβέλειάς της. Γιαυτό πρώτα κάνουμε branch σε ένα label που βρίσκεται σε κοντινή απόσταση και στη συνέχεια κάνουμε χρήση της jump που μπορεί να αναφέρεται σε όλο το τομήμα μνήμης του προγράμματος. Επίσης χρησιμοποιήθηκε ένας counter-cntr που αυξάνει την τιμή του κατά ένα κάθε φορά που εισέρχεται σε μία από τις προαναφερθείσες λειτουργίες που γίνονται μεταξύ των AEM .Όταν ο counter πάρει τη μέγιστη τιμή του τότε δεν θα γίνει δεκτό κάποιο επιπλέον πάτημα διακόπτη παρά μόνο θα τερματίσει το πρόγραμμα με αναμένα όλα τα led και είσοδο στον αέναο βρόχο hi .

```
;ergasia1.aps
;Papadopoulos Konstantinos, AEM 8677
;Topalidhs Efthymios, AEM 8417
;GROUP_3


.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 the program

Table:.dw 8677, 8417, 34423, 33815 ;Store the decimal and BCD values of aem's


;--------------------
.def aem1_H=R16 ; (aem1H_byte)
.def aem1_L=R17 ; (aem1L_byte)
.def aem2_H=R18 ; (aem2H_byte)
.def aem2_L=R19 ; (aem2L_byte)
.def cntr=R23   ; COUNTER FOR TMHMA_2

.def aem1_Hb=R6 ; (aem1H_byte BCD)
.def aem1_Lb=R7 ; (aem1L_byte BCD)
.def aem2_Hb=R8 ; (aem2H_byte BCD)
.def aem2_Lb=R9 ; (aem2L_byte BCD)


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

        out SPL,r16

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



main:

ldi ZH, high(Table<<1) ; Initialize Z pointer
ldi ZL, low(Table<<1)
lpm aem1_L, Z+; Load constant from program
lpm aem1_H, Z+


lpm aem2_L, Z+; Load constant from program
lpm aem2_H, Z+;


lpm aem1_Lb, Z+; Load constant from program for BDC
lpm aem1_Hb, Z+
```

```
lpm aem2_Lb, Z+; Load constant from program for BCD
lpm aem2_Hb, Z+


;------------------------ DISPLAY OF AEM1 BCD

ldi r21, 0b11111111
out DDRB, r21 ;The data direction register of port B is named DDRB

aem1:
        mov r21, aem1_Lb ;We display aem's characters for 5sec and with a gap
                         ;of 3sec between each other

        rcall display_delay


;----------------------
        mov r21, aem1_Hb

        rcall display_delay


        cpi cntr, 0b00000001 ;this compare and jump is for TMHMA_2
        breq bcksw1

bcksw1: jmp SWITCHES

;-----------------------

aem2: mov r21, aem2_Lb

        rcall display_delay


;----------------------


        mov r21, aem2_Hb

        rcall display_delay

        cpi cntr, 0b00000010
        breq bcksw2

bcksw2:jmp SWITCHES
```

```
;------------------ COMPARE TWO AEM


cmp: cp aem1_H, aem2_H ;check if aem1>aem2
        breq check          ; go to check if equal
        brge led0           ; if > go to led0



led1: ldi r21, 0b11111111

        out DDRB, r21 ;The data direction register of port B is named DDRB
        ldi r21, 0b11111110 ; open only led 1

        out PORTB, r21 ;PORTB is connected with leds

        rcall Delay3

        cpi cntr, 0b00000011
        breq bcksw3

bcksw3:jmp SWITCHES

        rjmp outside




check:      cp aem2_L, aem1_L
            brge led1

led0: ldi r20, 0b11111111

        out DDRB, r20
        ldi r20, 0b11111110      ; open only led 0

        out PORTB, r20

        rcall Delay3

        cpi cntr, 0b00000011
        breq bcksw4
```

```
bcksw4:jmp SWITCHES

        rjmp outside



outside:

;------------------SW6(AEM2-AEM1)_SW7(AEM1-
AEM2)_OPERATIONS----------------

            mov r3, aem1_L    ;These registers will be used for subtraction
            mov r4, aem2_L
            mov r5, aem1_H
            mov r6, aem2_H


            ldi r21, 0b11111111         ; make PORTB output port

        out DDRB, r21
        ldi r21, 0b00000000    ; make PORTD input port. PORTD is connected with
                               ;switches

        out DDRD, r21

swagain:    in r21, PIND        ; copy state of pressed switches to PORTD

                    andi r21, 0b11111111 ; mask

                    cpi r21, 0b01111111        ; check if switch 7 is pressed
                    breq difaem1
                    cpi r21, 0b10111111 ; check id switch 6 is pressed
                    breq difaem2
                    rjmp swagain



difaem1:    clc
                    sub r3, r4
                    sbc r5, r6
                    mov r21, r3          ; display in binary the aem difference
                    rcall display_delay
                    mov r21, r5          ; display in binary the aem difference
                    rcall display_delay
                    cpi cntr, 0b00000100
```

```
                breq bcksw5

bcksw5:     jmp SWITCHES

                rjmp hi

difaem2:    clc
                sub r4, r3
                sbc r6, r5
                mov r21, r4         ; display in binary the aem difference
                rcall display_delay
                mov r21, r6         ; display in binary the aem difference
                rcall display_delay
                cpi cntr, 0b00000101
                breq bcksw6

bcksw6:     jmp SWITCHES

                rjmp hi


close:          ldi r22, 255
                out PORTB, r22    ; close led lights



        ;-------------TMHMA_2----------------

;cntr register is a counter that help us to know if our program counter has passed
;from sections aem1 , aem2 , cmp , difaem1 , difaem2.These sections are in progress
;when we press switch0 , switch1 , switch2 , switch6 , switch7 respectively

aem1new:inc cntr ;we increase cntr so as to know which section is in progress
            jmp aem1


aem2new:inc cntr
            jmp aem2


cmp1:       inc cntr
            jmp cmp


difaem1new:     inc cntr
```

```
                jmp difaem1


difaem2new: inc cntr
                jmp difaem2


LED_ON:   ldi r21, 0    ;we turn all LEDs ON when all the operations take place and
the
                                                ;counter has the max value
                out PORTB, r21
                clr cntr
                rjmp hi
;In the section below we check in a loop which switch has been pressed and we
transfer
;the program counter in the corresponding section

SWITCHES:
            cpi cntr, 0b00000101 ;check if all characters have been displayd
            breq LED_ON
            ldi r21, 0b11111111         ; make PORTB output port
        out DDRB, r21
        ldi r21, 0b00000000    ; make PORTD input port
    out DDRD, r21
            in r21, PIND
            cpi r21, 0b11111110         ;check if sw0 is pressed
            breq aem1new
            cpi     r21, 0b11111101    ;check if sw1 is pressed
            breq aem2new
            cpi r21, 0b11111011 ;check if sw2 is pressed
            breq cmp1
            cpi r21, 0b01111111 ;check if sw7 is pressed
            breq difaem1new
            cpi r21, 0b10111111         ;check if sw6 is pressed
            breq difaem2new
            rjmp SWITCHES


            ;maybe nope is used before in r21, PIND



hi:             ;In the end of the program we transfer the program counter in a infinite
loop
```

```
rjmp hi




;------------

display_delay: com r21
                    out PORTB, r21 ; display sequence of bytes and delay
                    rcall Delay5          ; keep every character on for 5 secs
                    ldi r21, 0 ; close led lights for 3 secs (space betwenn characters)
                    out PORTB, r21
                    rcall Delay3
                    ret


; Delay 19 999 996 cycles
; 4s 999ms 999us at 4.0 MHz

Delay5:   ldi  r18, 102
          ldi  r19, 118
          ldi  r20, 192
L1:       dec  r20
          brne L1
         dec  r19
         brne L1
         dec  r18
         brne L1
         nop
                ret
```

```
; Delay 11 999 996 cycles
; 2s 999ms 999us at 4.0 MHz

Delay3:ldi  r18, 61
      ldi  r19, 225
      ldi  r20, 63
L2:    dec  r20
      brne L2
      dec  r19
      brne L2
      dec  r18
      brne L2
      nop
           ret



;Delay:              LDI outer_L, low(N0)          ; is is going to be called with
RCALL
;                   LDI outer_H, high(N0)
;outer_loop: ldi inner_L, low(N1)
;                   LDI inner_H, high(N1)
;inner_loop: nop
;                   sbiw inner, 1
;                   brne inner_loop
;                   sbiw outer,1
;                   brne outer_loop
;                   ret                                   ; Ttot = 5*Ni*No +
5*No + 8
```