ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 1

Ομάδα 7
Χατζηθωμά Αντρέας 8026
Νήρας Δημήτρης 8057

<u>ΤΜΗΜΑ 1</u>

Αρχικά κάναμε definition τους καταχωρητές R26-R29 για λόγους ευκολίας και αποθηκεύσαμε τα ΑΕΜ μας εκεί. Έπειτα αρχικοποιήσαμε τον Stack Pointer ούτως ώστε να αποθηκευτεί σωστά το άθροισμα των ΑΕΜ στην SRAM και να αποθηκευτούν σωστά οι διαδικασίες. Στην πρόσθεση του ΑΕΜ παρατηρήσαμε από τον STATUS REGISTER ότι έχουμε υπερχείλιση και επομένως κρατήσαμε σε ακόμα ένα καταχωρητή το κρατούμενο. Ακολούθως ορίσαμε ως έξοδο το PORTB για το άναμμα ων LED. Έπειτα εμφανίζονται διαδοχικά τα ΑΕΜ και το άθροισμα τους στα LEDs όπως περιγράφεται στην εκφώνηση με μια καθυστέρηση 10 δευτερολέπτων η οποία παράγεται από την διαδικασία delay. Η διαδικασία delay αποτελείται από 3 επαναληπτικές διαδικασίες (for loop). Υπολογίστηκε πως μια επανάληψη αντιστοιχεί σε 3 κύκλους μηχανής (κύκλος μηχανής = 0,25 μs) και επομένως τα όρια τέθηκαν με τρόπο ώστε να παράγεται καθυστέρηση περίπου 10 δευτερολέπτων.

Δυσκολευτήκαμε στο κομμάτι της πρόσθεσης των BCD ΑΕΜ καθώς υπήρχε υπερχείλιση, αλλά και στον υπολογισμό της καθυστέρησης καθώς έπρεπε να τρέξουμε αρκετές φορές το πρόγραμμα για να ελέγξουμε τη σωστή λειτουργία του.

Το πρόγραμμα στην πλακέτα έτρεξε χωρίς προβλήματα όπως ακριβώς και στο simulator.

<u>ΤΜΗΜΑ 2</u>

Στο 2ο μέρος του προγράμματος ορίζουμε ως έξοδο το port D στο οποίο είναι συνδεδεμένοι οι διακόπτες. Έπειτα, μέσω διαδικασιών delay ελέγχουμε αν ο διακόπτης SW1 έχει πατηθεί και ακολούθως αν έχει αφαιθεί. Όταν γίνει αυτό (ενεργοποίηση – απενεργοποίηση διακόπτη), εμφανίζεται στα LEDs το ΑΕΜ του 1ου φοιτητή. Ακολούθως, με την ίδια διαδικασία, εμφανίζεται το ΑΕΜ του 2ου φοιτητή χρησιμοποιώντας όμως τον διακόπτη SW2. Τέλος, με διαδοχικά πατήματα του διακόπτη SW3, εμφανίζονται διαδοχικά στα LEDs 0–3 όλα τα ψηφία του αθροίσματος (σε μορφή BCD) των 2 ΑΕΜ. Αυτό επιτυγχάνεται χρησιμοποιώντας διαδικασίες delay για τον έλεγχο της ενεργοποίησης - απενεργοποίησης διακόπτη, όπως γίνεται και με τους προηγούμενους διακόπτες, και ακολούθως εφαρμόζεται κ κατάλληλη ολίσθηση στον register τον οποίο θέλουμε να εμφανίσουμε στα LEDs ούτως ώστε να χρησιμοποιούνται μόνο τα LEDs 0-3 και όχι όλα.

Το τμήμα 2 δεν είχε ιδιαίτερη δυσκολία συγκριτικά με το τμήμα 1, αλλά αν υπήρχε κάτι το οποία μας πήρε περισσότερο χρόνο από τα υπόλοιπα, αυτό είναι ο έλεγχος των διακοπτών μέσω των διαδικασιών delay.

BREAKPOINTS

Για τον έλεγχο της σωστής λειτουργίας του προγράμματος επιλέξαμε να βάλουμε breakpoints στο σημείο της πρόσθεσης των ΑΕΜ ούτως ώστε να ελέγξουμε την ορθότητα του αποτελέσματος. Ακολούθως, βάλαμε breakpoints στο σημείο που εμφανίζονται τα ΑΕΜ μας και η πρόσθεση τους στα LEDs ούτως ώστε να δούμε αν ανάβουν τα σωστά LEDs.

ΚΩΔΙΚΑΣ ΠΡΟΓΡΑΜΜΑΤΟΣ

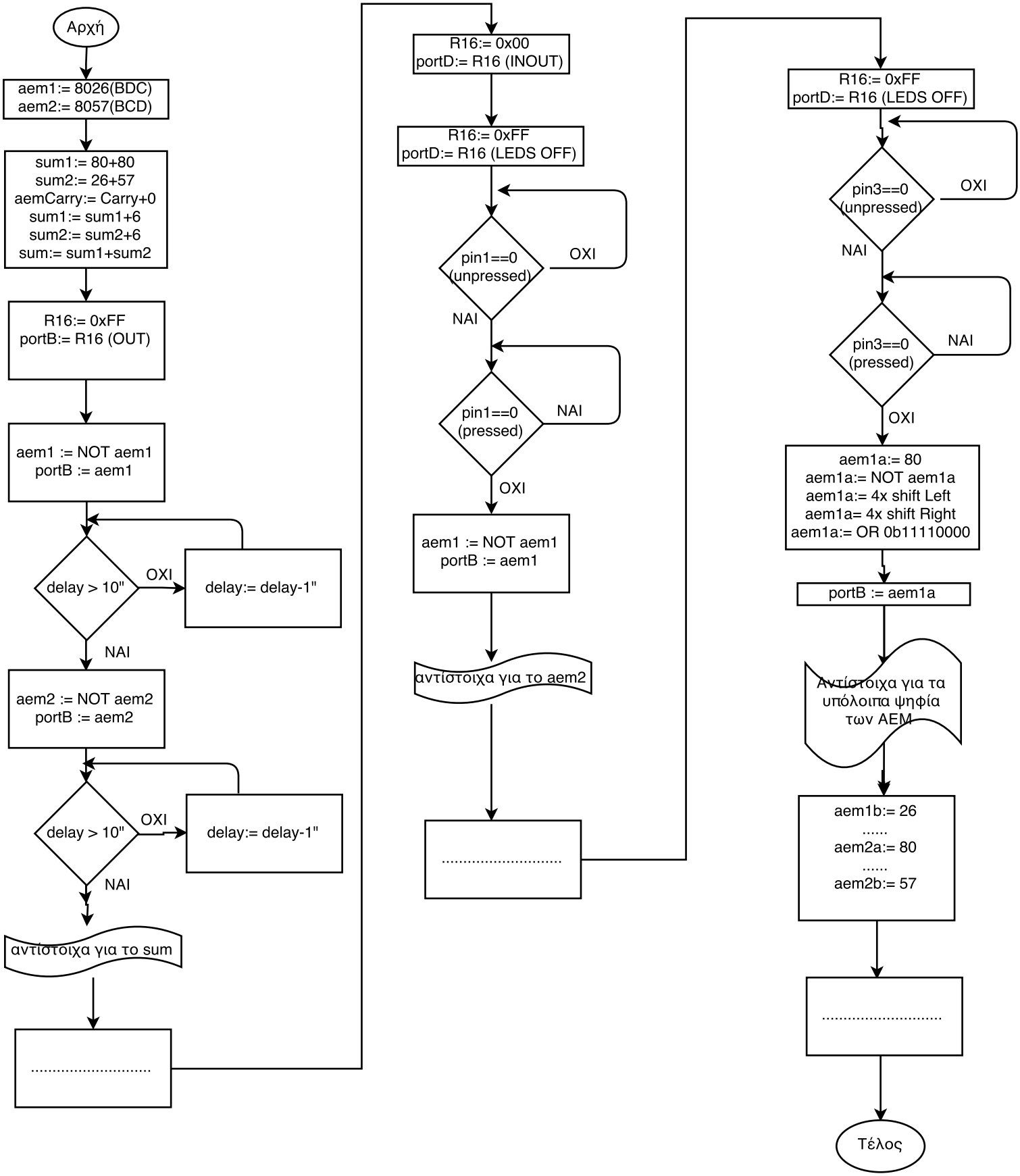
```
;lab1
;Χατζηθωμά Αντρέας 8026
;Νήρας Δημήτρης 8057
;Στο τμήμα 1 αρχικοποιούμε τον stack pointer, εισάγουμε τα αεμ και υπολογίζουμε την πρόσθεση.
;Ακολούθως στα LEDs εμφανίζονται τα ΑΕΜ μας και το αποτέλεσμα της πρόσθεσης τους, ανα 10 δευτερόλεπτα το καθένα.
;Στο τμήμα 2 τα ΑΕΜ και το άθροισμα τους, εμφανίζονται στα LEDs αν πατηθούν οι διακόπτες SW1, SW2 και SW3 αντίστοιχα.

.include "m16def.inc"
.org 0
.cseg

rjmp part1

.def aem1a = r27
.def aem1b = r26
.def aem2a = r29
.def aem2b = r28

part1:
;initialize the stack
sp_init:
        ldi r16, low(RAMEND)
        out spl, r16
        ldi r16, high(RAMEND)
        out sph, r16

main:
        ldi aem1a, 0b10000000        ;BCD num 80
        ldi aem1b, 0b00100110        ;BCD num 26
        ldi aem2a, 0b10000000        ;BCD num 80
        ldi aem2b, 0b01010111        ;BCD num 57
        ldi r20, 0b00000110   ;6 number to add if BCD addition is upper 9 in second byte array addition
        ldi r21, 0b01100000   ;6 number to add if BCD addition is upper 9 in fisrt byte array addition

        mov r17, aem1b
        mov r18, aem2b
        add r17, r18            ;add 26+57
        add r17, r20            ;add 6 to result of (26+57)
        sts 0x0060, r17                 ;store result to SRAM address 0x0060
```

```
        mov r17, aem2a
        mov r18, aem2a
        adc r17, r18        ;add 80+80 and the contents of C flag (carry)

        ldi r24, $00        ;load 0 to register
        ldi r25, $00        ;load 0 to register
        adc r24, r25        ;add zero to keep the carry
        sts 0x0070, r24             ;store result to SRAM address 0x0070

        add r17, r21        ;add 6 to result of (80+80)
        sts 0x0080, r17             ;store result to SRAM address 0x0080

setDDRBouput:
        ser r16                     ;all "1"
        out DDRB, r16       ;set portB (exit)

led_blinking:
        ser r16                     ;all "1"
        out PORTB, r16              ;turn off LEDs
        rcall aem_1b
        rcall aem_1a
        rcall aem_2b
        rcall aem_2a
        rcall add0
        rcall add1
        rcall add2
        ser r16
        out PORTB, r16
        rjmp part2

;LED functions
aem_1b:
        mov r30, aem1b
        com r30                     ;NOT(register) because of the ACTIVE LOW logic of LEDs
        out PORTB, r30
        rcall delay
        ret

aem_1a:
        mov r30, aem1a
        com r30
        out PORTB, r30
        rcall delay
        ret

aem_2b:
        mov r30, aem2b
        com r30
        out PORTB, r30
        rcall delay
        ret
```

```
aem_2a:
        mov r30, aem2a
        com r30
        out PORTB, r30
        rcall delay
        ret

add0:
        lds r30, 0x0060
        com r30
        out PORTB, r30
        rcall delay
        ret

add1:
        lds r30, 0x0080
        com r30
        out PORTB, r30
        rcall delay
        ret

add2:
        lds r30, 0x0070
        com r30
        out PORTB, r30
        rcall delay
        ret

delay:
        ldi r17, 0x2
d7:
        dec r17
        brne d7
        ret


bdelay:
        ldi r17, 0xed                   ;237x237x238 loops for 10sec delay (cycle time = 0.25us)
d3:
        ldi r18, 0xed
d2:
        ldi r19, 0xee
d1:
        dec r19
        brne d1
        dec r18
        brne d2
        dec r17
        brne d3
        ret
```

```
part2:
        clr r16                 ;all "0"
        out DDRD, r16           ;set portD (input)

        ser r16
        out PIND, r16           ;turn off leds

;switch 1
;checks if switch is unpressed and waits until is pressed
sw1_unpressed:
        sbic PIND, 0x01         ;Skip next instruction if (Port D, pin1 == 0)
        rjmp sw1_unpressed
;checks if switch is pressed and waits until is released
sw1_pressed:
        sbis PIND, 0x01         ;If (Port D, pin1 == 0) do
        rjmp sw1_pressed

        rcall aem_1b
        rcall aem_1a

        ser r16
        out PORTB, r16

;switch 2
sw2_unpressed:
        sbic PIND, 0x02         ;Skip next instruction if (Port D, pin2 == 0)
        rjmp sw2_unpressed
sw2_pressed:
        sbis PIND, 0x02         ;If (Port D, pin2 == 0) do
        rjmp sw2_pressed

        rcall aem_2b
        rcall aem_2a

        ser r16
        out PORTB, r16

        rcall press
        lds r30, 0x0060
        rcall left_right

        rcall press
        lds r30, 0x0060
        rcall right

        rcall press
        lds r30, 0x0080
        rcall left_right

        rcall press
        lds r30, 0x0080
```

```
        rcall right

        rcall press
        rcall add2

        ser r16
        out PORTB, r16

end:
        rjmp end

;release variables
.undef aem1a
.undef aem1b
.undef aem2a
.undef aem2b

;switch 3
press:
sw3_unpressed:
        sbic PIND, 0x03         ;Skip next instruction if (Port D, pin3 == 0)
        rjmp sw3_unpressed
sw3_pressed:
        sbis PIND, 0x03         ;If (Port D, pin3 == 0) do
        rjmp sw3_pressed
        ret

;shift register 4 times left/right inj order to light up leds 0-3 only
left_right:
        com r30
        lsl r30
        lsl r30
        lsl r30
        lsl r30
        lsr r30
        lsr r30
        lsr r30
        lsr r30
        ori r30, 0b11110000
        out PORTB, r30
        ret

right:
        com r30
        lsr r30
        lsr r30
        lsr r30
        lsr r30
        ori r30, 0b11110000
        out PORTB, r30
        ret
```


Αρχή
aem1:= 8026(BDC)
aem2:= 8057(BCD)
sum1:= 80+80
sum2:= 26+57
aemCarry:= Carry+0
sum1:= sum1+6
sum2:= sum2+6
sum:= sum1+sum2
R16:= 0xFF
portB:= R16 (OUT)
aem1 := NOT aem1
portB := aem1
delay > 10"
OXI
delay:= delay-1"
NAI
aem2 := NOT aem2
portB := aem2
delay > 10"
OXI
delay:= delay-1"
NAI
αντίστοιχα για το sum
.....
R16:= 0x00
portD:= R16 (INOUT)
R16:= 0xFF
portD:= R16 (LEDS OFF)
pin1==0
(unpressed)
OXI
NAI
pin1==0
(pressed)
NAI
OXI
aem1 := NOT aem1
portB := aem1
αντίστοιχα για το aem2
.....
R16:= 0xFF
portD:= R16 (LEDS OFF)
pin3==0
(unpressed)
OXI
NAI
pin3==0
(pressed)
NAI
OXI
aem1a:= 80
aem1a:= NOT aem1a
aem1a:= 4x shift Left
aem1a= 4x shift Right
aem1a:= OR 0b11110000
portB := aem1a
Αντίστοιχα για τα
υπόλοιπα ψηφία
των AEM
aem1b:= 26
.....
aem2a:= 80
.....
aem2b:= 57
.....
Τέλος