ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

Αναφορά

1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Ανάπτυξη κώδικα για τον μικροελεγκτή ATmega328PB και προσομοίωση της εκτέλεσης του στο αναπτυξιακό περιβάλλον MPLAB X

Ομάδα 1

Ξανθόπουλος Παναγιώτης (03119084)

Παπαναστάσης Αθανάσιος (03113197)

Ζήτημα 1.1

Αρχικά, το πρόγραμμα αρχικοποιεί τον καταχωρητή r24. Εισάγουμε για τυχαία δοκιμή τον αριθμό 17 στη δεκαεξαδική μορφή του στο ζεύγος μεταβλητών r24(low byte), r25(high byte) και καλούμε τη συνάρτηση wait_x_msec, η οποία θα δημιουργήσει χρονική καθυστέρηση x ms με απόλυτη ακρίβεια, στο συγκεκριμένο παράδειγμα 17 ms.

Στη συνέχεια έχουμε επανάληψη x φορές (όσες και η τιμή των r25:r24), όπου κάθε φορά καλούμε τη delay_986 που διαρκεί 986 cc ακριβώς. Μειώνουμε την τιμή των r25:r24 κατά 1 με τη χρήση της εντολής sbiw (η οποία χειρίζεται ζεύγος καταχωρητών) και αν γίνει 0, γίνεται branch (με την εντολή breq, που βλέπει αν η προηγούμενη εντολή οδήγησε σε μηδενικό αποτέλεσμα μέσω του bit Z του καταχωρητή σημαιών) και επιστρέφουμε, αλλιώς συνεχίζουμε την επανάληψη.

Για τις πρώτες x-1 (η τιμή των r25-r24) επαναλήψεις, έχουμε τις καθυστερήσεις:

- rcall delay_986 -> 986cc (ρουτίνα) + 3cc (εντολή rcall)
- sbiw -> 2cc
- breq -> 1cc, αφού δεν γίνεται branch
- Στη συνέχεια έχουμε 4 rjmp, άρα σύνολο 8cc

Οπότε (x-1)*(986+3+2+1+8)=1000*(x-1)cc

Για την τελευταία επανάληψη έχουμε:

- rcall delay_986 -> 986cc (ρουτίνα) + 3cc (εντολή rcall)
- sbiw -> 2cc
- breq -> 2cc, εφόσον γίνεται branch
- ret -> 4

Οπότε σύνολο 997cc. Αν προσθέσουμε και την αρχική κλήση στη wait_x_msec (3cc) έχουμε 1000cc. Άρα σύνολο (x-1)*1000+1000=1000*x cc. Εφόσον το ρολόι είναι στο 1MHz, 1cc διαρκεί 1μs άρα 1000*x cc διαρκούν x ms.

```
.include "m328PBdef.inc"

reset:
   ldi r24, low(RAMEND)
   out SPL, r24
   ldi r24, high(RAMEND)
   out SPH, r24

main:
   ldi r24, 0x11
   ldi r25, 0x00
   rcall wait_x_msec
   rjmp main

wait_x_msec:
   rcall delay_986u
   sbiw r24, 1
   breq end

   rjmp help1                ;2cc

help1:
   rjmp help2                ;2 cc

help2:
   rjmp help3                ;2 cc
```

```
help3:
   rjmp wait_x_msec          ;2 cc

end:
   ret

delay_986u:
   ldi r26, 98

loop_986u:
   rcall wait_4u
   dec r26
   brne loop_986u

   nop
   nop
   ret

wait_4u:
   ret
```

Ζήτημα 1.2

Το πρόγραμμα αρχικά ορίζει τους καταχωρητές r25, r24, r23, r22 στους οποίους αναθέτουμε τις δοσμένες από την εκφώνηση τιμές των μεταβλητών, A, B, C, D. Στη συνέχεια ορίζουμε τον καταχωρητή r21 να ισούται με 6, τον αριθμό των επαναλήψεων που θέλουμε να υπολογιστούν οι λογικές συναρτήσεις. Μετά ορίζουμε τα increments, τις τιμές με τις οποίες θα αυξάνεται η κάθε μεταβλητή, όπως έχουν δοθεί από την εκφώνηση (r26 για τη μεταβλητή A, r27 για τη μεταβλητή B, r28 για τη μεταβλητή C, r29 για τη μεταβλητή D).

Ορίζουμε επίσης τους καταχωρητές r18, r19, r20 για να υπολογίζουν τις προσωρινές τιμές των συναρτήσεων και των επιμέρους πράξεων.

Αντιγράφουμε στον καταχωρητή r20 τη μεταβλητή A (mov r20, r25), υπολογίζουμε το συμπλήρωμά του (com r20). Επαναλαμβάνουμε τη διαδικασία με τον r19 για το συμπλήρωμα του B. Υπολογίζουμε το AND των A', B' (and r20, r19), και κάνουμε το ίδιο για τις μεταβλητές B', D (and r19, r22). Υπολογίζουμε το OR των r19, r20 (or r20, r19). Τέλος, υπολογίζουμε το συμπλήρωμά του αποτελέσματος, το οποίο ισούται με τη συνάρτηση F0.

Αντίστοιχες πράξεις υλοποιούμε για τον υπολογισμό της F1 στη συνέχεια, όπως φαίνεται στο πρόγραμμα παρακάτω.

Στη συνέχεια, προσθέτουμε στους καταχωρητές που περιέχουν τις τιμές των μεταβλητών (r25, r24, r23, 22) τα αντίστοιχα increments και αφού μειωθεί κατά 1 η τιμή του καταχωρητή r21, αν δεν ισούται με το 0, τότε το πρόγραμμα συνεχίζει με την επόμενη επανάληψη, αλλιώς σταματά.

Οι τιμές των συναρτήσεων, καθώς και των μεταβλητών σε κάθε κύκλο επαναλήψεων, φαίνονται στον παρακάτω πίνακα.

A	B	C	D	F0	F1
0x55	0x43	0x22	0x02	0x57	0x77
0x57	0x46	0x26	0x07	0x56	0x76

0x59	0x49	0x2A	0x0C	0x59	0x7B
0x5B	0x4C	0x2E	0x11	0x4E	0x6E
0x5D	0x4F	0x32	0x16	0x4F	0x6F
0x5F	0x52	0x36	0x1B	0x56	0x76

```
.include "m328PBdef.inc"

reset:
   ldi r24, low(RAMEND)
   out SPL, r24
   ldi r24, high(RAMEND)
   out SPH, r24

main:
   ldi r25, 0x55       ;A
   ldi r24, 0x43       ;B
   ldi r23, 0x22       ;C
   ldi r22, 0x02       ;D

   ldi r21, 0x06       ;counter

   ldi r26, 0x02       ;increments
   ldi r27, 0x03
   ldi r28, 0x04
   ldi r29, 0x05

loop:

   ;compute F0 first

   mov r20, r25        ;r20 = A
   com r20             ;r20 = A'

   mov r19, r24
   com r19             ;r19 = B'

   and r20, r19        ;r20 = A' * B'

   and r19, r22        ;r19 = B' * D

   or r20, r19
   com r20             ;r20 = F0

   ;compute F1

   mov r19, r25        ;r19 = A
   or r19, r23         ;r19 = A + C

   mov r18, r22        ;r18 = D
   com r18             ;r18 = D'
   or r18, r24         ;r18 = B + D'

   and r19, r18        ;r19 = F1

   ;change A,B,C,D

   add r25, r26
   add r24, r27
```

```
    add r23, r28
    add r22, r29

    ;loop condition

    subi r21, 0x01
    brne loop
```

Ζήτημα 1.3

Αρχικά, ορίζουμε τη θύρα εξόδου PORTD ως output. Στη συνέχεια, εφόσον ο r23 θα θέτει την έξοδο της PORTD, τον αρχικοποιούμε στην τιμή 0x01 (LSB).

Ορίζουμε τη συνάρτηση check_T, στην οποία ελέγχουμε το T flag του SREG, στο οποίο έχουμε αποθηκεύσει την κατεύθυνση κίνησης του βαγονέτου. Όταν το T είναι 1 το βαγονέτο κινείται αριστερά ενώ όταν το T είναι 0, το βαγονέτο κινείται δεξιά. Αν το T είναι 1 γίνεται branch στη συνάρτηση shift_left και ξεκινά η κίνηση προς τα αριστερά, αλλιώς συνεχίζουμε στη συνάρτηση shift_right και το βαγονέτο κινείται δεξιά.

Στη συνάρτηση shift_right θέτουμε την έξοδο του PORTD ίση με την τιμή του r23. Θέτουμε το ζεύγος καταχωρητών ίσο με 500 και καλούμε τη συνάρτηση wait_x_msec, που παρουσιάστηκε στο Ζήτημα 1.1, για να δημιουργήσουμε την επιθυμητή χρονική καθυστέρηση των 500ms, την παύση δηλαδή του βαγονέτου στην κάθε στάση. Στη συνέχεια, ο καταχωρητής ολισθαίνει δεξιά. Αν προηγουμένως ο καταχωρητής r23 ήταν στο αριστερό άκρο (LSB), με την δεξιά ολίσθηση γίνεται 0. Οπότε, ελέγχουμε αν ο r23 ισούται με 0, δηλαδή αν η έξοδος του PORTD βρίσκεται στο LSB. Αν δεν είναι 0, συνεχίζουμε την κίνηση στα δεξιά, αλλιώς συνεχίζουμε στη συνάρτηση change_to_left ώστε να αλλάξουμε κατεύθυνση κίνησης.

Στη συνάρτηση change_to_left το bit που συμβολίζει το βαγονέτο αλλάζει κατεύθυνση, ορίζουμε δηλαδή το T flag ίσο με 1 και δημιουργούμε έξτρα καθυστέρηση 1s, λόγω της πρόσθετης στάσης. Θέτουμε τον r23 ίσο με 0x02, ώστε το bit που ισούται με 1 να βρίσκεται αριστερά από το lsb, για να ξεκινήσει η κίνηση προς την αριστερή κατεύθυνση και συνεχίζουμε το πρόγραμμα με τη συνάρτηση check_T.

Στη συνέχεια, ορίζουμε την κίνηση στα αριστερά με αντίστοιχο τρόπο και ορίζουμε τη συνάρτηση που δημιουργεί χρονικές καθυστερήσεις, όπως στο Ζήτημα 1.

```
.include "m328PBdef.inc"

reset:
    ldi r24, low(RAMEND)
    out SPL, r24
    ldi r24, high(RAMEND)
    out SPH, r24

init:
    ser r23
    out DDRD, r23     ;PORTD as output

    ldi r23, 0b00000001          ;LED initialized at LSB
    set
```

```
;if T=1, move left
;if T=0, move right

check_T:
   brts shift_left      ;if T is set (1), branch to shift left routine

shift_right:
   out PORTD, r23   ;output
   ldi r24, 0xf4
   ldi r25, 0x01
   rcall wait_x_msec            ;wait 500ms
   lsr r23              ;shift right
   cpi r23, 0x00        ;check if the LED is at LSB
   brne check_T         ;if not, continue shifting right

change_to_left:
   set                          ;T becomes 1, so we shift left
   ldi r24, 0xe8
   ldi r25, 0x03
   rcall wait_x_msec            ;wait extra 1000ms
   ldi r23, 0x02        ;new led position
   rjmp check_T         ;continue

shift_left:
   out PORTD, r23   ;output
   ldi r24, 0xf4
   ldi r25, 0x01
   rcall wait_x_msec            ;wait 500ms
   lsl r23              ;shift left
   cpi r23, 0x00        ;check if the LED is at MSB
   brne check_T         ;if not, continue shifting left

change_to_right:
   clt                          ;T becomes 0, so we shift right
   ldi r24, 0xe8
   ldi r25, 0x03
   rcall wait_x_msec            ;wait extra 1000ms
   ldi r23, 0x40        ;new led position
   rjmp check_T         ;continue

wait_x_msec:
   rcall delay_986u
   sbiw r24, 1
   breq end

   rjmp help1

help1:
   rjmp help2                   ;2 cc

help2:
   rjmp help3                   ;2 cc

help3:
   rjmp wait_x_msec             ;2 cc

end:
   ret
```

```
 delay_986u:
    ldi r26, 98

loop_986u:
    rcall wait_4u
    dec r26
    brne loop_986u

    nop
    nop
    ret

wait_4u:
    ret
```