



Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Μικροϋπολογιστών, ακ. έτος 2024 - 2025

Λύσεις στην 1η Εργαστηριακή Άσκηση

Όνομα: Ειρήνη	Όνομα: Γεώργιος
Επώνυμο: Σιμιτζή	Επώνυμο: Οικονόμου
ΑΜ: 03121063	ΑΜ: 03121103
Εργαστηριακή Ομάδα: 30	

Ζήτημα 1.1

Στο ζήτημα αυτό υλοποιούμε κώδικα Assembly ο οποίος παράγει ρυθμιζόμενες χρονικές καθυστερήσεις της τάξης των x msec, όπου x είναι ένας αριθμός από 1 έως 65.535, ή στο δυαδικό σύστημα από 1 σε 1111111111111111. Ο αριθμός x αποθηκεύεται στο ζεύγος των καταχωρητών r24 (low byte) και r25 (high byte).

```
.include "m328PBdef.inc"

.equ FOSC_MHZ = 16      ; MHz
.equ DEL_ms = 1000     ; x ms ; x = 1, ..., 65535
                        ; x ms ; x = 1, ..., 0b1111111111111111
                        ; highest number that can be represented
                        ; by an unsigned 16-bit binary

.equ F1 = FOSC_MHZ*DEL_ms ; F1 = 16M*xm = 16*1000*x cycles

    ldi r24, LOW(RAMEND)    ; initialize stack pointer
    out SPL, r24
    ldi r24, HIGH(RAMEND)
    out SPH, r24

    ; 16 = 2^4 = 10000, *16 will shift left 4 bits binary
    ; 12 bits binary left so highest number that is safe
    ; to not overflow is 4095 = 111111111111
    ; x is 1, ..., 4095 to not overflow

start:
    ldi r24, low(DEL_ms)    ; Load DEL_ms (lower byte)
    ldi r25, high(DEL_ms)   ; Load DEL_ms (upper byte)
    rcall wait_x_msec
    rjmp start

    ; delay_inner gives in total ~1000 cycles
    ; it is the internal 1000 in 16*1000*x
delay_inner:
    ldi r23, 246           ; 1 cycle
loop3:    ; this produces 4*245 + 8 = 988 cycles
    dec r23                ; 1 cycle
    nop                    ; 1 cycle
    brne loop3             ; 2 cycle if r23 != 0 else 1 cycle
    nop                    ; 1 cycle
    ret                    ; 4 cycles

wait_x_msec:    ; wait_x_msec gives in total 5 cycles
    push r24        ; 2 cycles
    push r25        ; 2 cycles
```

```

    ldi r22, 16          ; 1 cycle

outer_loop:

    ldi r24, low(DEL_mS) ; 1 cycle
    ldi r25, high(DEL_mS) ; 1 cycle

    ; outer_loop gives in total 2 cycles

    ; loop4 gives in total 994 * x - 1 cycles
    ; it is the internal 1000*x in 16*1000*x
loop4:
    rcall delay_inner    ; 988+1+3 cycles
    sbiw r24, 1          ; (2 cycles)
    brne loop4           ; (1 or 2 cycles)

    ; it is the 16 in 16*1000*x
    dec r22              ; (1 cycle)
    brne outer_loop      ; 2 cycle if r22 != 0 else 1 cycle

    ; we have achieved 15.904*x + c cycles of delay up to this point

extra:  pop r25           ; (2 cycles)
        pop r24          ; (2 cycles)

        ret

```

Η συχνότητα του μικροελεγκτή είναι 16MHz και το πλήθος των κύκλων που γίνεται η μέτρηση του χρόνου είναι $FI = FOSC_MHZ * DEL_mS = 16MHz * xmsec = 16 * 1.000 * x$. Εξαιτίας αυτού, δημιουργούμε φωλιασμένες επαναλήψεις (nested loops) και υπολογίζουμε τους κύκλους κάθε εντολής προκειμένου να καταλήξουμε στο απαραίτητο ποσό. Αναλυτικά στον κώδικα σημειώνονται το πλήθος των κύκλων σε κάθε υπορουτίνα wait_x_msec, outer_loop, loop4, delay_inner, και loop3.

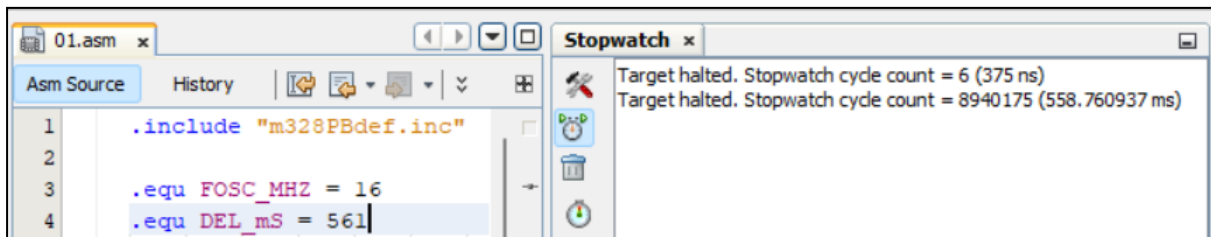
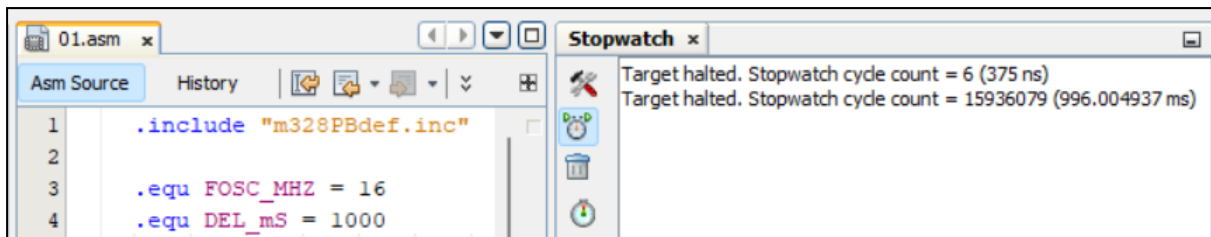
Τελικά, καταλήγουμε σε πλήθος $15.904 * x + const$ κύκλων και μετράμε τον χρόνο εκτέλεσης με το εργαλείο stopwatch, θέτοντας διαδοχικά breakpoints κατά και έπειτα της κλήσης της εντολής rcall wait_x_msec.

```

21  start:
22      ldi r24, low(DEL_mS)    ; Load DEL_mS (lower byte)
23      ldi r25, high(DEL_mS)  ; Load DEL_mS (upper byte)
    rcall wait_x_msec
    rjmp start

```

Επίσης, δοκιμάζουμε τον κώδικα για τιμές καθυστερήσεων $x = 1.000$ και $x = 561$.



Παρατηρούμε ότι η διαφορά, ή το "σφάλμα", είναι της τάξης του 0.40%. Για να διορθωθεί, μπορεί να προστεθεί αριθμός εντολών nop ίσος με τους υπολειπόμενους κύκλους, δηλαδή 96 κύκλοι.

Τέλος, όπως αναφέραμε, το 65.535 στο δυαδικό σύστημα είναι το 1111111111111111, δηλαδή, είναι ο μεγαλύτερος αριθμός που μπορεί να αναπαρασταθεί από έναν 16-bit δυαδικό αριθμό χωρίς πρόσημο. Πολλαπλασιάζοντας το x στο ζεύγος των καταχωρητών r24 και r25 με το 16, τα bits μετατοπίζονται κατά 4 θέσεις αριστερά. Ως αποτέλεσμα, υπερβαίνουμε το μέγεθος της λέξης του μικροελεγκτή, γεγονός που προκαλεί overflow (υπερχείλιση). Οπότε, ο μεγαλύτερος αριθμός msec που μπορούμε να έχουμε με ασφάλεια ως καθυστέρηση είναι ο μεγαλύτερος αριθμός που μπορεί να αναπαρασταθεί από έναν 12-bit δυαδικό αριθμό χωρίς πρόσημο, δηλαδή 4.095. Κώδικας Assembly που υπολογίζει ακριβώς 16.000 κύκλους παρουσιάζεται στο Ζήτημα 1.3.

Ζήτημα 1.2

Στο ζήτημα αυτό υλοποιούμε κώδικα Assembly ο οποίος υπολογίζει τις 2 λογικές συναρτήσεις F0 και F1. Παρακάτω παρουσιάζονται τα καίρια κομμάτια του κώδικα.

```
main:
; calculate F0 = (A*B_ + B_*D)_
com r_b      ; B is now B_
mov f_0, r_a  ; temporary save A
and f_0, r_b  ; f_0 is now A*B_
mov r23, r_d  ; temporary save D
and r23, r_b  ; r23 is now B_*D
or f_0, r23   ; f_0 is now F0
com f_0
com r_b
```

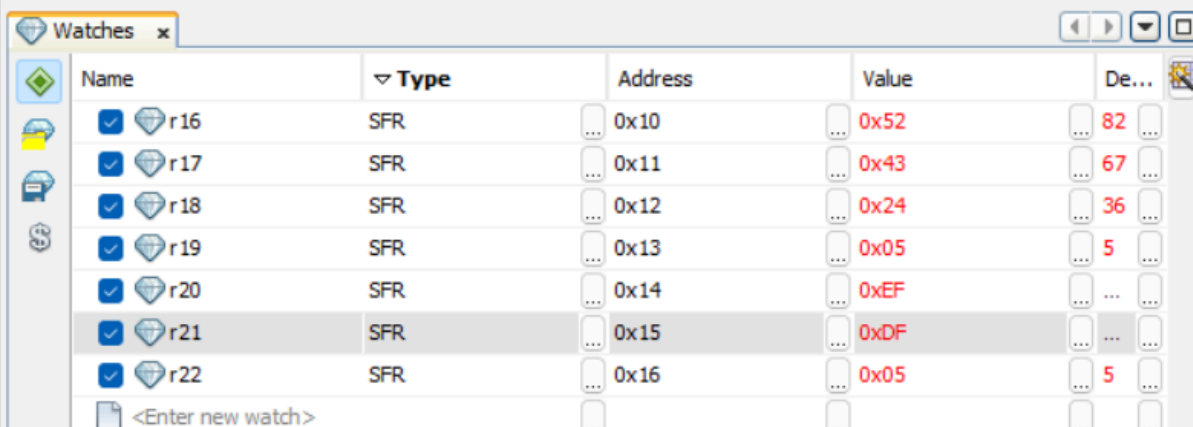
```
; calculate F1 = (A+C_) * (B*D_)
com r_c      ; C is now C_
mov f_1, r_a  ; temporary save A
or f_1, r_c   ; f_1 is now A+C_
com r_d
mov r23, r_b  ; temporary save B
or r23, r_d
and f_1, r23  ; f_1 is now F1
com r_c
com r_d
```

Θέτοντας breakpoint κατά την κλήση της εντολής brne main, εκτελούμε τον κώδικα διαδοχικά 6 φορές, όσες και ο αριθμός που είναι αποθηκευμένος στον καταχωρητή r22.

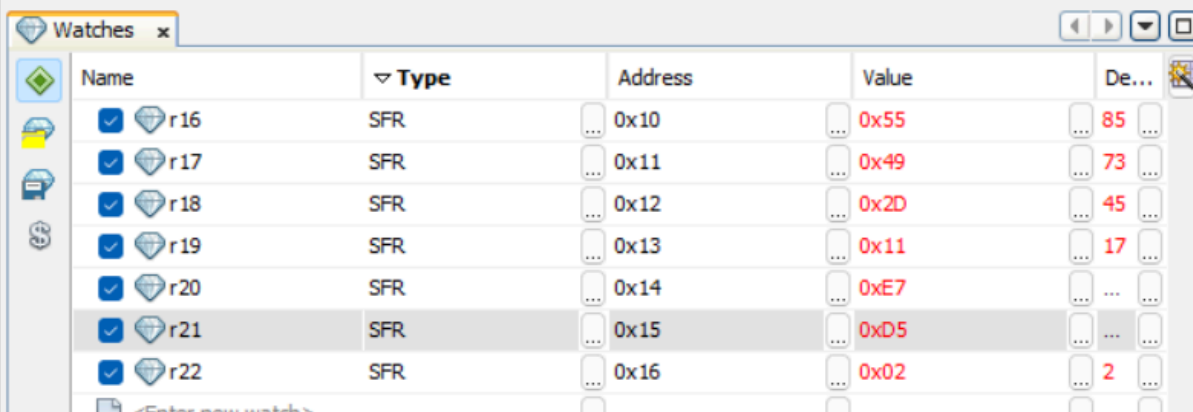
Χρησιμοποιούμε κατάλληλα watches για όλους τους καταχωρητές r16 - r22 και με τη διαδικασία του debugging συμπληρώνουμε τον πίνακα.

A	B	C	D	F0	F1
0x51	0x41	0x21	0x01	0xEF	0xDF
0x52	0x43	0x24	0x05	0xEB	0xDB
0x53	0x45	0x27	0x09	0xE5	0xD3
0x54	0x47	0x2A	0x0D	0xE7	0xD5
0x55	0x49	0x2D	0x11	0xEB	0xC7
0x56	0x4B	0x30	0x15	0xEB	0xCB

Παραθέτουμε δύο τυχαία στιγμιότυπα του παραθύρου watches κατά την εκτέλεση του κώδικα, ένα κατά την πρώτη επανάληψη ($i = 6$) και ένα κατά την τρίτη επανάληψη ($i = 3$).



Name	Type	Address	Value	De...
<input checked="" type="checkbox"/> r16	SFR	0x10	0x52	82
<input checked="" type="checkbox"/> r17	SFR	0x11	0x43	67
<input checked="" type="checkbox"/> r18	SFR	0x12	0x24	36
<input checked="" type="checkbox"/> r19	SFR	0x13	0x05	5
<input checked="" type="checkbox"/> r20	SFR	0x14	0xEF	...
<input checked="" type="checkbox"/> r21	SFR	0x15	0xDF	...
<input checked="" type="checkbox"/> r22	SFR	0x16	0x05	5
<Enter new watch>				



Name	Type	Address	Value	De...
<input checked="" type="checkbox"/> r16	SFR	0x10	0x55	85
<input checked="" type="checkbox"/> r17	SFR	0x11	0x49	73
<input checked="" type="checkbox"/> r18	SFR	0x12	0x2D	45
<input checked="" type="checkbox"/> r19	SFR	0x13	0x11	17
<input checked="" type="checkbox"/> r20	SFR	0x14	0xE7	...
<input checked="" type="checkbox"/> r21	SFR	0x15	0xD5	...
<input checked="" type="checkbox"/> r22	SFR	0x16	0x02	2
<Enter new watch>				

Τέλος, επαληθεύσαμε τις τιμές των συναρτήσεων χρησιμοποιώντας το παρακάτω python script:

```
F0 = ~((A & ~B) | (~B & D))
print("F0", hex(F0 & 0xFF))
F1 = (A | ~C) & (B | ~D)
print("F1", hex(F1 & 0xFF))
```

Ζήτημα 1.3

Στο ζήτημα αυτό υλοποιούμε κώδικα Assembly ο οποίος ελέγχει ένα αυτοματισμό βαγονέτου που κινείται συνεχώς, αρχικά από δεξιά προς τα αριστερά και στη συνέχεια αντίστροφα. Μοντελοποιούμε την κατεύθυνση της κίνησης του βαγονέτου χρησιμοποιώντας το T flag του SREG (Status Register). Η λογική λειτουργία είναι η εξής:

- Όταν το T flag είναι 0, το βαγονέτο κινείται προς τα δεξιά.
- Όταν το T flag είναι 1, το βαγονέτο κινείται προς τα αριστερά.

Παρακάτω παρουσιάζονται τα καίρια κομμάτια του κώδικα.

```
left:
    out PORTD, wagon; show current value
    clt; clear T flag

    ldi r24, low(delay) ; load delay
    ldi r25, high(delay)

    rcall wait_x_msec; call delay function
    lsl wagon; logical shift left of wagon
    cpi wagon, 0x00; compare with zero

    brne left; if it is not zero then go to left
; (it moves to the left until it reaches the end)

rotateright;; end of the row, we have to turn to the right
    ldi r24, low(1000); extra 1 sec
    ldi r25, high(1000)

    rcall wait_x_msec; call the delay function
    clt; T flag set to zero
    ldi wagon, 0x40;01000000
```

Αντίστοιχο κομμάτι κώδικα γράφουμε για την κίνηση προς τα δεξιά. Η καθυστέρηση κάθε βήματος υλοποιείται καταλλήλως με τον κώδικα του Ζητήματος 1.1