



1^η Εργαστηριακή Άσκηση

Ομάδα 17

Κωνσταντίνος Πίκουλας 03120112

Ανδρέας Στάμος 03120018

1^η Άσκηση

Στην 1^η Άσκηση υλοποιήσαμε μια ρουτίνα χρονοκαθυστέρησης *wait_x_msec* η οποία καθυστερεί την εκτέλεση του προγράμματος κατά *x msec*, όπου *x* ο αριθμός που είναι αποθηκευμένος στον διπλό καταχωρητή *r24:r25*.

Κώδικας 1^{ης} άσκησης

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=7    ;MHz

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

loop1:
    ldi r24, low(1256)
    ldi r25, high(1256)

    rcall wait_x_msec    ; 3 cycles
    rjmp loop1

wait_x_msec:
    ldi r16, FOSC_MHZ    ; 1 cycle
                        ; load frequency (7 MHz) THIS IS THE COUNTER FOR THE LOOP
                        ; r27:r26 will hold the result of the multiplication of FOSC_MHZ * r25:r24
                        ; multiplying as follows: (r27 * 2^8 + r26) * r16 = (r27 * r16 * 2^8) + (r26 * r16)

    mul r16, r24    ; 1 cycle
    movw r26, r0    ; 1 cycle
    mul r16, r25    ; 1 cycle
    add r27, r0     ; 1 cycle

    sbiw r26, 1     ; 2 cycles, subtract one to account for the overhead

                        ; need 994-7=987=246*4+3 more cycles (overhead)
    ldi r23, 246    ; (1 cycle)
loop0:
    dec r23         ; 1 cycle
    nop            ; 1 cycle
```

```

    brne loop0          ; 1 or 2 cycles
    nop                 ; 1 cycle
    nop                 ; 1 cycle
    nop                 ; 1 cycle
                        ;total group delay 996 cycles
delay_inner:
    ldi    r23, 249     ; (1 cycle)
loop_inn:
    dec r23             ; 1 cycle
    nop                 ; 1 cycle
    brne loop_inn       ; 1 or 2 cycles

    sbiw r26,1          ; 2 cycles
    brne delay_inner    ; 1 or 2 cycles

    ret                 ; 4 cycles

```

Εξήγηση

Αρχικά έχουμε γνωστά μόνο την συχνότητα στην οποία τρέχει ο μικροελεγκτής μας (στον κώδικα είναι ***FOSC_MHZ=7 MHz***) καθώς και την καθυστέρηση που επιθυμούμε να προκαλέσουμε (και είναι αποθηκευμένη στον διπλό καταχωρητή r24:r25).

Επομένως όταν κληθεί η εντολή *rcall wait_x_msec* υπολογίζουμε πόσους κύκλους πρέπει να εκτελέσουμε προκειμένου η καθυστέρηση που θα προκληθεί να είναι απόλυτα ακριβής. Η κύκλοι είναι $FOSC_HZ * delay$ όπου delay η καθυστέρηση σε second και FOSC_HZ η συχνότητα σε Hz.

Επειδή ο υπολογισμός αυτός χρησιμοποιεί κάποιος κύκλους ρολογιού αφαιρούμε κάποιους κύκλους ρολογιού από τα loop μας που βρίσκονται εντός της ρουτίνας χρονοκαθυστερήσης και προκειμένου να καλύψουμε την διαφορά προσθέτουμε κάποιες κενές εντολές *nop*.

Αποτέλεσμα

Εκτελώντας το πρόγραμμα στο simulator του MPLAB, στα 7MHz (ή σε οποιαδήποτε συχνότητα αρκεί η συχνότητα που είναι ορισμένη στο πρόγραμμα να είναι ίδια με την συχνότητα του simulator) και με επιθυμητή καθυστέρηση 1256msec (αποθηκευμένη στον διπλό καταχωρητή r24:r25) παίρνουμε το παρακάτω αποτέλεσμα:

Target halted. Stopwatch cycle count = 8792000 (1.256 s)

2^η Άσκηση

Στην 2^η Άσκηση υλοποιήσαμε ένα πρόγραμμα το οποίο υπολογίζει 2 λογικές συναρτήσεις:

- $F0 = (A' \cdot B' \cdot C' + D)'$
- $F1 = (A' + C) \cdot (B' + D')$

Κώδικας 2^{ης} άσκησης

```
.include "m328PBdef.inc"

.def A=r16
.def B=r17
.def C=r18
.def D=r19
.def F0=r20
.def F1=r21
.def temp=r22
.def counter=r23

reset:
    ldi A, 0x45 ; load initial values to registers
    ldi B, 0x23
    ldi C, 0x21
    ldi D, 0x01
    ldi counter, 5 ; init the counter (5 times)
loop:
    com A
    com B
    com C

    mov F0, A
    and F0, B
    and F0, C
    or F0, D
    com F0 ; F1 holds the appropriate value

    ; A,B,C commed

    com C
```

```

mov F1, A
or F1, C
mov temp, D
com temp
or temp, B
and F1, temp

```

```

; A,B commed

```

```

com A
com B

```

```

dec counter
breq exit

```

```

subi A, -0x01
subi B, -0x02
subi C, -0x04
subi D, -0x05
rjmp loop

```

```

exit:
rjmp exit

```

Εξήγηση

Το πρόγραμμα μας υπολογίζει τις λογικές συναρτήσεις F0 και F1 και τις αποθηκεύει στους καταχωρητές r20 και r21 αντίστοιχα. Σε κάθε loop προσθέτει την ζητούμενη σταθερά στον αντίστοιχο καταχωρητή.

Αποτέλεσμα

Από την εκτέλεση του προγράμματος μας και με χρήση breakpoints παίρνουμε τα παρακάτω αποτελέσματα:

A	B	C	D	F0	F1
0x45	0x23	0x21	0x01	0x66	0xBA
0x46	0x25	0x25	0x06	0x61	0xB9
0x47	0x27	0x29	0x0B	0x64	0xB8
0x48	0x29	0x2D	0x10	0x6D	0xBF
0x49	0x2B	0x31	0x15	0x6A	0xB6

3^η Άσκηση

Στην 3^η Άσκηση υλοποιήσαμε ένα πρόγραμμα το οποίο ελέγχει ένα αυτοματισμό βαγονέτου που κινείται συνεχώς, αρχικά από δεξιά προς τα αριστερά και στην συνέχεια αντίστροφα. Η κίνηση του βαγονέτου κατά μια θέση γίνεται κάθε 1.5sec (κατά προσέγγιση), εκτός από όταν βρίσκεται σε ακριανή θέση και πρέπει να αλλάξει κατεύθυνση, στην οποία περίπτωση κάνει πρόσθετη στάση 2sec.

Κώδικας 3^{ης} άσκησης

```
.include "m328PBdef.inc"

.equ FOSC_MHZ=16 ;MHz
.equ NORMAL_TIMEOUT=1500 ;ms
;Big timeout is the extra time we have to wait when changing rotation
.equ BIG_TIMEOUT=3500-NORMAL_TIMEOUT ;ms
.def rail=r21

reset:
    ldi r16,low(RAMEND)
    out SPL,r16
    ldi r16,high(RAMEND)
    out SPH,r16

    ser r16
    out DDRD, r16 ; set PORTD as output

; T=1 means we are going to the left
; T=0 means we are going to the right

set ; in the beginning we are going to the left so T=1
ldi rail, 0x01
out PORTD, rail
ldi r24, low(NORMAL_TIMEOUT)
ldi r25, high(NORMAL_TIMEOUT)

main_loop:
    rcall wait_x_msec
check_right_edge:
    cpi rail, 0x01 ; check if we are on the right edge
    brne check_left_edge ; if we are not then check if we are on the left edge
    brbc 6, change_direction_to_left ; if we are going to right change direction
    jmp rotate
```

```

check_left_edge:
    cpi rail, 0x80 ; check if we are on the left edge
    brne rotate ; if we are not then rotate
    brbs 6, change_direction_to_right ; if we are heading right then change direction
    jmp rotate
change_direction_to_left:
    set ; T = 1
    ldi r24, low(BIG_TIMEOUT) ; make additional timeout
    ldi r25, high(BIG_TIMEOUT)
    rcall wait_x_msec
    lsl rail ; shift left
    out PORTD, rail
    ldi r24, low(NORMAL_TIMEOUT) ; restore normal timeout
    ldi r25, high(NORMAL_TIMEOUT)
    jmp main_loop
change_direction_to_right:
    clt ; T = 0
    ldi r24, low(BIG_TIMEOUT) ; make additional timeout
    ldi r25, high(BIG_TIMEOUT)
    rcall wait_x_msec
    lsr rail ; shift right
    out PORTD, rail
    ldi r24, low(NORMAL_TIMEOUT) ; restore old timeout
    ldi r25, high(NORMAL_TIMEOUT)
    jmp main_loop

rotate:
    brtc rotate_right ; if T = 0 rotate right
    lsl rail
    out PORTD, rail
    jmp main_loop
rotate_right:
    lsr rail
    out PORTD, rail
    jmp main_loop

wait_x_msec:
    ldi r16, FOSC_MHZ    ; 1 cycle
                        ; load frequency (16 MHz) THIS IS THE COUNTER FOR THE LOOP
                        ; r27:r26 will hold the result of the multiplication of FOSC_MHZ * r25:r24
                        ; multiplying as follows: (r27 * 2^8 + r26) * r16 = (r27 * r16 * 2^8) + (r26 *
r16)
    mul r16, r24    ; 1 cycle
    movw r26, r0    ; 1 cycle
    mul r16, r25    ; 1 cycle

```


add r27, r0	; 1 cycle
sbiw r26, 1	; 2 cycles, subtract one to account for the overhead
; need $994-7=987=246*4+3$ more cycles (overhead)	
ldi r23, 246	; (1 cycle)
loop0:	
dec r23	; 1 cycle
nop	; 1 cycle
brne loop0	; 1 or 2 cycles
nop	; 1 cycle
nop	; 1 cycle
nop	; 1 cycle
;total group delay 996 cycles	
delay_inner:	
ldi r23, 249	; (1 cycle)
loop_inn:	
dec r23	; 1 cycle
nop	; 1 cycle
brne loop_inn	; 1 or 2 cycles
sbiw r26,1	; 2 cycles
brne delay_inner	; 1 or 2 cycles
ret	; 4 cycles

Εξήγηση

Η κατεύθυνση του βαγονέτου αποθηκεύεται στο T flag του καταχωρητή SREG του AVR. Η σύμβαση που πήραμε είναι ότι εάν:

- T = 0, η κίνηση είναι προς τα δεξιά
- T = 1, η κίνηση είναι προς τα αριστερά

Η κίνηση προς δεξιά ή αριστερά γίνεται με βάση το T flag. Κάθε φορά που πρέπει να κινηθεί το βαγονέτο αρχικά καθυστερούμε 1.5sec, και έπειτα ελέγχουμε εάν έχουμε φτάσει σε ακριανή θέση. Εάν έχουμε φτάσει σε ακριανή θέση τότε περιμένουμε extra 2sec (ώστε συνολικά να καθυστερήσει η εκτέλεση 3.5sec), αλλάζουμε το T flag σε 0 ή 1 αναλόγως την φορά της κίνησης και μετά ολισθαίνουμε το βαγονέτο προς την ανάλογη κατεύθυνση.

Η ρουτίνα χρονοκαυστέρησης είναι ακριβώς η ίδια με αυτήν της 1^{ης} Άσκησης.



--