Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών
Εργαστήριο Μικροϋπολογιστών

Έκθεση 1ης Εργαστηριακής Άσκησης

Στοιχεία:
Ομάδα: **39**
Μέλη: **Ευάγγελος Μυργιώτης, Αριστοτέλης Γρίβας**

Ζήτημα 1.1 :

Ο παρακάτω κώδικας υλοποιεί την χρονοκαθυστέρηση 1ms με τον εξής τρόπο:

Η κλήση της rcall είναι **3** κύκλοι

Μέσα στο loop κάνει 99 επαναλήψεις. Εκ των οποίων:

1η: Συνολικά 11 κύκλους (λόγω του ldi r26, 99)
2η-98η: Συνολικά 10 κύκλους
99η: Συνολικά 9 κύκλους (λόγω του λιγότερου κύκλου από την brne

Με άθροισμα (10+1)+97(10)+(10-1) = 99*10 = **990** Κύκλους ρολογιού

Τα υπόλοιπα 10 είναι από την sbiw, brne wait (με τα 6 nop)

Για κάθε ms εκτος του τελευταίου δηλαδή R25:R24 != 1 έχουμε άθροισμα 1000 κύκλοι(1ms για 1mhz)

Για το τελευταίο ms (R25:R24 = 1) δεν θα μπει μέσα στην wait από το brne wait και μαζί με την return, οι κύκλοι αντί για 10 θα είναι **7**.
με σύνολο 997.

Άρα για X ms ο κώδικας εκτελεί 3 + (X-1)* 1000 + 997 = X*1000 κύκλοι (X ms για 1 mhz)

Βάζοντας Break point στις εξής εντολές:

```
main:
    rcall  wait_x_msec
    rjmp end
```

Με input 1000ms 1sec βλέπουμε στο stop watch την εξής καθυστέρηση:
Target halted. Stopwatch cycle count = 1000000 (1 s)

Αντίστοιχα για 500 ms:
Target halted. Stopwatch cycle count = 500000 (500 ms)

Κώδικας :

```
.include "m328PBdef.inc"

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

    ldi r24, low(1000)
    ldi r25, high(1000)
 main:
     rcall  wait_x_msec
     rjmp end


wait4:
    ret                    ;4 cycles

wait:
    nop                   ;6 cycles
    nop
    nop
    nop
    nop
    nop
wait_x_msec:
    ldi r26, 99           ;1 cycles

loop:

    rcall wait4        ;3+4 = 7 cycles
    dec r26                ;1 cycle
    brne loop              ;1 or 2 cycles
    sbiw r25 : r24 , 1      ;2 cycles
    brne wait              ;1 or 2 cycles


    ret                   ;4cycles



end:
```

Ζήτημα 1.2 :

Ο τελικός πίνακας έχει αυτή τη μορφή:

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

Ο κώδικας υλοποιεί τις λογικές συναρτήσεις αποθηκεύοντας τα A,B,C,D στα register r16, r17, r18, r19 και κάνει λογικές πράξεις ανά bit, και παρουσιάζονται στα r20,21 (αντίστοιχα το F0 και F1)

Παρακάτω παραθέτουμε ενδεικτικά τα δυο πρώτα test cases:

R16	0x55	85	01010101
R17	0x43	67	01000011
R18	0x22	34	00100010
R19	0x02	2	00000010
R20	0x57	87	01010111
R21	0x77	119	01110111

R16	0x57	87	01010111
R17	0x46	70	01000110
R18	0x26	38	00100110
R19	0x07	7	00000111
R20	0x56	86	01010110
R21	0x76	118	01110110

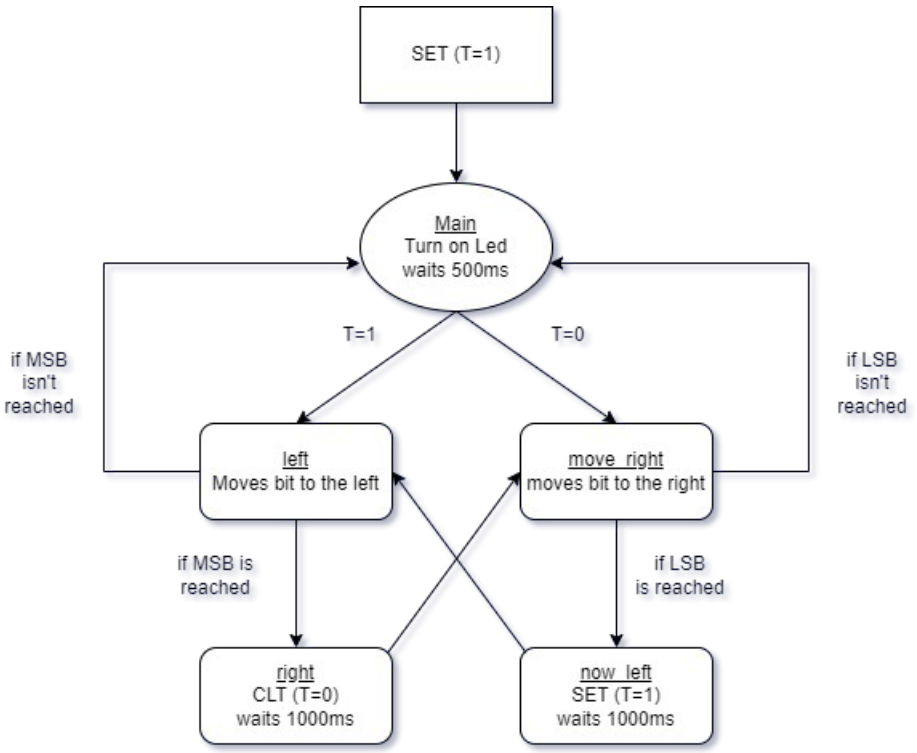
Κώδικας :

```
.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 var = r23

ldi A, 0x55          ;inital values
ldi B, 0x43
ldi C, 0x22
ldi D, 0x02
ldi var, 0x06        ;number of loops in var

make:
    mov F0, A        ;F0 = A
    or F0, B         ;F0 = (A+B)
    mov temp, D      ;temp = D
    com temp         ;temp = D'
    or temp, B       ;temp = (B+D')
    and F0, temp     ;F0 = F0*temp = (A+B)*(B+D')

    mov F1, A        ;F1 = A
    or F1, C         ;F1 = (A+C)
    and F1, temp     ;F1 = F1*temp = (A+C)*(B+D')
```

```
subi A, -2      ;add constants and loop again
subi B, -3
subi C, -4
subi D, -5
dec var             ;decrease var
brne make       ;if we've done the 6 loops end else loop again

end:
```

Ζήτημα 1.3 :

Ο κώδικας είναι η υλοποίηση αυτού του flowchart σε AVR assembly.



Κώδικας :

```
.include "m328PBdef.inc"

reset:
      ldi r24 , low(RAMEND)
      out SPL , r24
      ldi r24 , high(RAMEND)
      out SPH , r24
      ser r24                 ;PORTD output
      out DDRD , r24
      ldi r27 , 1             ;set lsb  for first output
      SET               ;set T flag = 1 for left rotation
      rjmp main

now_left:
      SET   ;if we've reached LSB we set T flag to 1 and start left rotation
      ldi r24 , low(1000)   ;extra 1ms latency
      ldi r25 , high(1000)
      rcall wait_x_msec
      rjmp left

main:
      out PORTD , r27       ;show LED
      ldi r24 , low(500)    ;wait 0.5s
      ldi r25 , high(500)
      rcall wait_x_msec
      BRTC move_right       ;if T flag = 0 move right else left

left:
      cpi r27 , 0x80        ;if we reach MSB its time to rotate right
      breq right
      lsl r27               ;else we continue left
      rjmp main             ;and print it
right:
      CLT   ;if we've reached MSB,we clear the T flag and start rotating right
      ldi r24 , low(1000)   ;we also need to wait 1ms extra
      ldi r25 , high(1000)
      rcall wait_x_msec

move_right:
      cpi r27 , 1           ;if we reach LSB its time to rotate left
      breq now_left
      lsr r27               ;else shift right and print it
      rjmp main



wait4:
```

```
    ret

wait:
    nop
    nop
    nop
    nop
    nop
    nop
wait_x_msec:

    ldi r26, 99

loop:
    rcall wait4
    dec r26
    brne loop
    sbiw r25 : r24 , 1
    brne wait

    ret
```