Σχολή Ηλεκτρολόγων Μηχανικών

&

Μηχανικών Υπολογιστών

Εργαστήριο Μικρουπολογιστών

5η Εργαστηριακή Άσκηση

Λαμπράκος Χρήστος Α.Μ. : 03112062

Μανδηλαράς Νικηφόρος, Α.Μ: 03112012

Σπαθαράκης Δημήτρης Α.Μ:03112523

Ομάδα : C05

Έβδομο Εξάμηνο

29 /11 /2015

Ζήτημα Πρώτο

Αρχικά ο κώδικάς μας, δίνει ονόματα σε τέσσερις καταχωρητές και συνεχίζει με την αρχικοποίηση της στοίβας, των θυρών εξόδου εισόδου και ανάβει το bit 0 της εξόδου. Στη μεταβλητή status κρατάμε την έξοδο τη δεδομένη στιγμή. Στη συνέχεια ελέγχει αν είναι πατημένο το bit 7 και αν ναι περιμένει να ελευθερωθεί για να εκτελέσει τη λειτουργία του. Μόλις το πλήκτρο βρεθεί μη ενεργοποιημένο θα τυπώσει την έξοδο και θα προετοιμάσει την επόμενη κατάσταση περιστρέφοντας την τρέχουσα έξοδο κατά μία θέση αριστερά. Στη συνέχεια προσθέτει στους καταχωρητές r24 , r25 την ζητούμενη καθυστέρηση και καλείται η συνάρτηση των διαφανειών wait_msec. Στη συνέχεια ελέγχει αν βρίσκεται στην αριστερότερη θέση και αν ναι τότε συνεχίζει στην ετικέτα WAITR διαφορετικά θα ξαναεκτελέσει τον βρόχο της WAITL. Όσο αφορά το βρόχο της WAITR εκτελεί ακριβώς την ίδια διεργασία απλά τώρα περιστρέφει δεξιά την έξοδο και ελέγχει στο τέλος αν βρίσκεται στην δεξιότερη θέση της συστοιχίας των LED ώστε να μεταβεί στην ετικέτα WAITL.

Κώδικας:

```
.include "m16def.inc"

.def status = r16
.def min    = r17
.def max    = r18
.def check  = r19

RESET:
   ldi status, low(RAMEND)         ; Initialize stack pointer.
   out SPL, status
   ldi status, high(RAMEND)
   out SPH, status
   clr status
   out DDRA, status       ; PORTA as input.
   out PORTA, status
   ser status
   out DDRB, status        ; PORTB as output.
   out PORTB, status      ; Enable pull-up resistors (for LEDs).
   ldi min, 0x01           ; Rightmost position.
   ldi max, 0x80           ; Leftmost position.
   mov status, min         ; Initial state: only LED A0 is on.

WAITL:
   in check, PINA          ;  Spin until PB0 is pressed. in mean read from PIN
   cp check, max
   breq WAITL
   out PORTB, status      ; Light LEDs.
   lsl status            ; Shift register to the left.
   ldi r24, low(500)
   ldi r25, high(500)
   rcall wait_msec        ; Delay for 500msec.
   cpse status, max       ; Break loop if leftmost bit is reached.
   rjmp WAITL

WAITR:
   in check, PINA         ; Spin until PB0 is pressed. in mean read from PIN
   cp check, max
```

```
    breq WAITR
    out PORTB, status      ; Light LEDs.
    lsr status             ; Shift register to the right.
    ldi r24, low(500)
    ldi r25, high(500)
    rcall wait_msec        ; Delay for 500msec.
    cpse status, min       ; Break loop if rightmost bit is reached.
    rjmp WAITR
    rjmp WAITL             ; Loop endlessly...

; == wait_msec ==
; Treats (r25:r24) as a 16-bit value K.
; Causes a delay almost equal to K msec. Calls wait_usec.
; MODIFIES: r24, r25, SPL, SPH
.
wait_msec:
    push r24               ; Save r24 on the stack.
    push r25               ; Save r25 on the stack.
    ldi r24, low(998)      ; (r25:r24) = 998
    ldi r25, high(998)
    rcall wait_usec        ; Cause a ~1msec delay.
    pop r25                ; Restore r25 from stack.
    pop r24                ; Restore r24 from stack.
    sbiw r24, 1            ; Decrease (r25:r24).
    brne wait_msec         ; Loop until (r25:r24) == 0.
    ret                    ; Return to caller.

; == wait_usec ==
; Treats r24 as a 8-bit value K.
; Causes a delay almost equal to K usec.
; MODIFIES: r24, r25.

wait_usec:
    sbiw r24, 1
    nop
    nop
    nop
    nop
    brne wait_usec
    ret
```

Ζήτημα Δεύτερο

Η υλοποίησή μας ξεκινά με την αρχικοποίηση της στοίβας και των θυρών εξόδου εισόδου. Στη συνέχεια διαβάζει από την είσοδο έναν αριθμό του οποίου απομονώνει τα 4 πρώτα και 4 τελευταία ψηφία στις μεταβλητές r15 και r16 αντίστοιχα, μετατοπίζοντας επίσης τα 4 MSB κατά 4 θέσεις δεξιά. Στη συνέχεια δημιουργεί το χρόνο της καθυστέρησης για κάθε μία από τις λειτουργίες ON και OFF που πρόκειται να εκτελέσουν τα LED ακολουθώντας την συνάρτηση που μας δίνεται (2*x+1)*50 όπου x τα r15 , r16 κατά περίπτωση. Το αποτέλεσμα του τελευταίου πολλαπλασιασμού αποθηκεύεται στους καταχωρητές r0 και r1 και μεταφέρεται αντίστοιχα στους καταχωρητές r24, r25 ώστε να καλέσουμε στην συνέχεια την wait_m_sec με την κατάλληλη καθυστέρηση ανά περίπτωση.

Κώδικας:

```
.include "m16def.inc"
ldi r24,low(RAMEND) ;initialize stack pointer
out spl,r24
ldi r24,high(RAMEND)
out sph,r24
clr r17             ; αρχικοποίηση της PORTB
out DDRB , r17      ; για είσοδο
ser r26             ; αρχικοποίηση της PORTA
out DDRA , r26      ; για έξοδο
flash:
  in r17,PINB
  mov r16,r17
  andi r16,0x0f ; mask for 4 LSB
  andi r17,0xf0 ;mask for 4 MSB
  mov r15,r17
  lsr r15       ;or swap r15
  lsr r15
  lsr r15
  lsr r15
  add r15,r15
  inc r15
  add r16,r16
  inc r16

  rcall on          ; Άναψε τα LEDs
  rcall off           ; Σβήσε τα LEDs
  rjmp flash          ; Επανέλαβε
                    ; Υπορουτίνα για να ανάβουν τα LEDs
on:
 ldi r20,50
 mul r15,r20
 mov r24 ,r0
 mov r25 ,r1
 ser r26         ; θέσε τη θύρα εξόδου των LED
 out PORTA , r26
 rcall wait_msec
 ret                ; Γύρισε στο κύριο πρόγραμμα
                    ; Υπορουτίνα για να σβήνουν τα LEDs
off:
 mul r16,r20
 mov r24 ,r0
 mov r25 ,r1
 clr r26        ; μηδένισε τη θύρα εξόδου των LED
 out PORTA , r26
 rcall wait_msec
 ret                ; Γύρισε στο κύριο πρόγραμμα

wait_usec:
```

```
sbiw r24 ,1 ; 2 κύκλοι (0.250 μsec)
nop ; 1 κύκλος (0.125 μsec)
nop ; 1 κύκλος (0.125 μsec)
nop ; 1 κύκλος (0.125 μsec)
nop ; 1 κύκλος (0.125 μsec)
brne wait_usec ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
ret

wait_msec:
 push r24 ; 2 κύκλοι (0.250 μsec)
 push r25 ; 2 κύκλοι
 ldi r24 , low(998) ; φόρτωσε τον καταχ. r25:r24 με 998 (1 κύκλος - 0.125 μsec)
 ldi r25 , high(998) ; 1 κύκλος (0.125 μsec)
 rcall wait_usec ; 3 κύκλοι (0.375 μsec), προκαλεί συνολικά καθυστέρηση 998.375 μsec
 pop r25 ; 2 κύκλοι (0.250 μsec)
 pop r24 ; 2 κύκλοι
 sbiw r24 , 1 ; 2 κύκλοι
 brne wait_msec ; 1 ή 2 κύκλοι (0.125 ή 0.250 μsec)
 ret ; 4 κύκλοι (0.500 μsec)
```

Ζήτημα Τρίτο

Το πρόγραμμά μας αφού αρχικοποιήσει τις θύρες εξόδου, εισόδου καθώς και τη στοίβα, δημιουργεί τις μεταβλητές s1, s2 που αντιπροσωπεύουν την τωρινή και την αμέσως προηγούμενη είσοδο αντίστοιχα. Στη συνέχεια μπαίνει σε μια επαναληπτική διαδικασία όπου συγκρίνει το τέταρτο bit τωρινής εισόδου με την προηγούμενη και αν η προηγούμενη τιμή του τετάρτου bit βρεθεί μεγαλύτερη (δηλ. 1 η προηγούμενη, μηδενική η τωρινή είσοδος) σημαίνει ότι είχαμε πάτημα και άφημα του κουμπιού οπότε εκτελείται η λειτουργία που αντιστοιχεί στο κουμπί 4 δηλαδή το άναμμα του τελευταίου bit. Στη συνέχεια απομονώνει και τα υπόλοιπα bit της εισόδου μέχρι το μηδενικό και ελέγχει κατά τον ίδιο τρόπο αν θα εκτελεστεί η εκάστοτε λειτουργία. Οι μετατοπίσεις κατά μία θέση αριστερά ή δεξιά πραγματοποιούνται με πολλαπλασιασμό / διαίρεση κατά δύο αντίστοιχα, ενώ για μετατοπίσεις δύο θέσεων κατά 4. Επιπλέον στις περιπτώσεις μετατόπισης των LED κατά δύο θέσεις απαιτείται να γίνει έλεγχος για το αν βρισκόμαστε τόσο στην τελευταία θέση, όπως συμβαίνει και για τις μετατοπίσεις της μιας θέσης, όσο και για το αν είμαστε στην προτελευταία. Στις ειδικές αυτές περιπτώσεις τίθεται καταλλήλως ανά περίπτωση ποιο θα είναι το επόμενο bit που θα ανάψει ώστε να επιτυγχάνεται η κυκλική λειτουργία.

Κώδικας:

```
#include <avr/io.h>

int main ()
{
  SP = RAMEND;
  int outpt=0, inpt=0, s1=0, s2=0, past;
  // Set Port D bits as inputs
  DDRB = 0xff;
  PORTB=0x80;
  outpt = 0x80;
  DDRD = 0x00;
  inpt = PIND;
  past=inpt;

  while (1) {
       // mask bit 4 of current and past input
       s1 = inpt & 0x10;
       s2 = past & 0x10;
       if ((s2 - s1) == (0x10)) {
```

```
                outpt = 0x80;
            PORTB = outpt;
        }
        // now mask bit 3...
        s1 = inpt & 0x08;
        s2 = past & 0x08;
        if ((s2 - s1) == (0x08)) {
                if (outpt <= 0b00100000)
                        outpt = outpt * 4;
                else if (outpt == 0x40)
                        outpt = 0x01;
                else
                        outpt = 0x02;
                PORTB = outpt;
                //past = inpt;
                //inpt = PIND;
                //continue;
        }
        // now mask bit 2...
        s1 = inpt & 0x04;
        s2 = past & 0x04;
        if ((s2 - s1) == (0x04)) {
                if (outpt >= 0b00000100)
                        outpt = outpt / 4;
                else if (outpt == 0x02)
                        outpt = 0x80;
                else
                        outpt = 0x40;
                PORTB = outpt;
        }
        // now mask bit 1...
        s1 = inpt & 0x02;
        s2 = past & 0x02;
        if ((s2 - s1) == (0x02)) {
                if (outpt <= 0b01000000)
                        outpt = outpt * 2;
                else
                        outpt = 0x01;
                PORTB = outpt;
        }
        // now mask the LSB...
        s1 = inpt & 0x01;
        s2 = past & 0x01;
        if ((s2 - s1) == (0x01)) {
                if (outpt >= 0b00000010)
                        outpt = outpt / 2;
                else
                        outpt = 0x80;
                PORTB = outpt;
        }
    past = inpt;
        inpt = PIND;
  }
  return 0;
}
```