Σχολή Ηλεκτρολόγων Μηχανικών

&

Μηχανικών Υπολογιστών

Εργαστήριο Μικρουπολογιστών

7η Εργαστηριακή Άσκηση

Λαμπράκος Χρήστος Α.Μ. : 03112062

Μανδηλαράς Νικηφόρος, Α.Μ: 03112012

Σπαθαράκης Δημήτρης Α.Μ:03113523

Ομάδα : C05

Έβδομο Εξάμηνο

14/12 /2015

Ζήτημα Πρώτο

Ο κώδικάς μας αρχικοποιεί τη στοίβα, τις θύρες A,C ως θύρες εισόδου και τέλος την θύρα B ως θύρα εξόδου. Έχοντας λάβει την είσοδο εισέρχεται σε μια επαναληπτική δομή την οποία πρόκειται να εκτελέσει 4 φορές και στην οποία σε κάθε επανάληψη απομονώνει τα δύο τελευταία bit της PORTA, περιστρέφοντάς την επίσης δύο θέσεις δεξιά. Έπειτα από κάθε περιστροφή το τελευταίο bit αποθηκεύεται στο κρατούμενο C που αποτελεί το τελευταίο bit του καταχωρητή Sreg. Απομονώνουμε το τελευταίο αυτό bit και το κρατάμε στον καταχωρητή r16 αν πρόκειται για το πρώτο bit της λούπας και στον r19 αν πρόκειται για το δεύτερο. Στη συνέχεια ανάλογα σε ποια επανάληψη βρισκόμαστε μετακινούμαστε στο τμήμα κώδικα εκείνο που υλοποιεί την κατάλληλη πύλη. Στον καταχωρητή r21 διατηρούμε κάθε στιγμή την υπό διαμόρφωση έξοδο. Στην ετικέτα T1 υλοποιείται η πρώτη πύλη XOR συγκρίνοντας το περιεχόμενο των καταχωρητών r16 , r19 και αν το βρει διαφορετικό αυξάνει κατά ένα την τιμή του r21. Στην ετικέτα T2, υλοποιούμε πύλη OR, συγκρίνουμε τώρα αν κάποιος καταχωρητής έχει τιμή “ένα” προκειμένου να θέσουμε “ένα” και στην έξοδο. Το αποτέλεσμα αυτό το κατευθύνουμε τόσο στην έξοδο B2 όσο και σε μια πύλη AND μαζί με το αποτέλεσμα της προηγούμενης ετικέτας για να καθορίσουμε έτσι και την τιμή της B1. Κατά τον ίδιο τρόπο με προηγουμένως υλοποιούμε στην ετικέτα T3 μια πύλη NOR με την προσθήκη μιας εντολής αντιστροφής του αποτελέσματος που προέκυψε. Τέλος για την υλοποίηση της πύλης NXOR χρησιμοποιούμε την ίδια διαδικασία σύγκρισης με την T1. Η αντιστροφή του κάθε ψηφίου ανάλογα με την αντίστοιχη είσοδο C υλοποιείται στην αντίστοιχη ετικέτα ελέγχοντας πρώτα αν το ψηφίο είναι και κατάλωντας έπειτα τη συνάρτηση _reverse_ που αντιστρέφει τον καταχωρητή r18 και έπειτα απομονώνει το τελευταίο του bit.

Κώδικας:

```
.include "m16def.inc"

    ldi r24,low(RAMEND) ;initialize stack pointer
    out spl,r24
    ldi r24,high(RAMEND)
    out sph,r24
    clr r17
    out DDRA, r17   ; PORTA as input.
    clr r17
    out DDRC, r17   ; PORTC as input.
    ser r26         ; αρχικοποίηση της PORTB
    out DDRB , r26  ; για έξοδο

flash:
       in r17,PINA       ; input in r17
       in r22,PINC       ; input in r22
       ldi r20 , 0X04  ; counter
LOOPA:
       ldi r18,0X00
       lsr r17         ; shift right
       in r16,sreg
       andi r16, 0x01  ; mask C flag
       lsr r17         ; shift right
       in r19,sreg
       andi r19, 0x01  ; mask C flag
       ;lsr r22
       ;in r23,sreg
       ;andi r23, 0x01
       dec r20
       cpi r20, 0x03
       breq T1
       cpi r20, 0x02
       breq T2
       cpi r20, 0x01
       breq T3
       cpi r20, 0x00
```

```
       breq T4

T1:                    // Gate XOR
       cp  r16,r19      ; Compare r16 to r19
       breq C01         ; Branch if r16==r19
       inc r18
C01:
       mov r21,r18
       rjmp LOOPA

T2:
   cpi r16,0x01       // Gate OR
   brne C02
   inc r18
   rjmp C03
C02:
   cpi r19,0x01
   brne C03
   inc r18
C03:
       mov r16,r18
       sbrc r22,1       ; check for C input
       rcall _reverse_   ; reverse
       lsl r18
       and r16,r21
       mov r14,r18      ; temp values in order to
       mov r18,r16      ; be compatible with _reverse_
       sbrc r22,0
       rcall _reverse_
       mov r16,r18      ; undo changes
       mov r18,r14
       sbrc r16,0
       inc r18
       mov r21, r18
       rjmp LOOPA

T3:                    // Gate NOR
   cpi r16,0x01
   brne C04
   inc r18
   rjmp C05
C04:
   cpi r19,0x01
   brne C05
   inc r18
C05:
   com r18
   andi r18, 0x01
   sbrc r22,2       ; check for C input
   rcall _reverse_   ; reserve
   lsl r18
   lsl r18
   add r21,r18
   rjmp LOOPA

T4:
       cp  r16,r19      ; Compare r16 to r19
       brne C06         ; Branch if r16!=r19
       inc r18
C06:
       sbrc r22,3       ; check for C input
       rcall _reverse_  ; reserve
       lsl r18
       lsl r18
       lsl r18
       add r21,r18
       andi r22, 0xf0
```

```
        add r21,r22
        out PORTB , r21
        rjmp flash


_reverse_:
    com r18
    andi r18, 0x01
    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 5 πρώτα ψηφία τα οποία και αποθηκεύει στις μεταβλητές a-e. Έπειτα υλοποιεί με τις κατάλληλες λογικές πράξεις τις τρεις ζητούμενες συναρτήσεις προσέχοντας όπου γίνεται αντιστροφή κάποιας μεταβλητής στη συνέχεια να απομονώνεται το τελευταίο bit. Τέλος προσθέτει την τιμή κάθε συνάρτησης στη μεταβλητή result την οποία έπειτα περιστρέφει μία θέση αριστερά προκειμένου στη συνέχεια να αποθηκευτεί η επόμενη συνάρτηση στο LSB. Οι f_0, f_1, f_2 εμφανίζονται εν τέλει κατά σειρά στα τρία MSB της θύρα C.

Κώδικας:

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

int main ()
{
        SP = RAMEND;
        uint8_t inpt, result, a, b, c, d, e, f_0, f_1, f_2;
        //PORT A's bits 0-4 as inputs
        DDRA = 0x00;
        //PORT C's bits 5-7 as outputs
        DDRC = 0xFF;
        while (1) {
                inpt = PINA;
                result = 0;
                //locate and align critical bits...
                a = inpt & 0x01;
                b = inpt & 0x02;
                b = b >> 1;
                c = inpt & 0x04;
                c = c >> 2;
                d = inpt & 0x08;
                d = d >> 3;
                e = inpt & 0x10;
                e = e >> 4;
                f_0 = ~((a & b & c) | (c & d) | (d & e));
                f_0 = f_0 & 0x01;
                //update result and shift...
                result = result + f_0;
                result = result << 1;
                f_1 = (a & b) | (c & d & ((~e) & 0x01));
                //same here
                result = result + f_1;
                result = result << 1;
                f_2 = f_0 | f_1;
                result = result + f_2;
                //final shift is five bits
                //to get to the MSB's
                result = result << 5;
                PORTC = result;
        }
        return 0;
```

```
}
```

Ζήτημα Τρίτο

Το πρόγραμμά μας αφού προβεί στις απαραίτητες αρχικοποιήσεις περιμένει μέχρι να διαβάσει το πρώτο νούμερο της ομάδας μας το “0”, οπότε και θα εισέλθει στη λούπα second number. Εκεί σε περίπτωση που δωθεί πάλι “0” ή τίποτα (σκέτο 0) ξαναγυρίζει στη δεύτερη λούπα. Αν δοθεί “5” που αποτελεί το δεύτερο νούμερο της ομάδας μας ανάβει όλα τα LED και καλή τη συνάρτηση καθυστέρησης. Σε περίπτωση που δοθεί κάτι διαφορετικό επιστρέφει στη λούπα first number. Για την ανάγνωση από το keypad χρησιμοποιήθηκαν οι δοσμένες συναρτήσεις `scan_keypad_rising_edge,keypad_to_ascii,scan_keypad.`

Κώδικας:

```
.include "m16def.inc"

.def temp = r16

.DSEG
_tmp_: .byte 2

.CSEG
.org 0x00
    rjmp RESET              ; Start of main program.

RESET:
    ldi r24, LOW(RAMEND)    ; Initialise stack pointer.
    out SPL, r24
    ldi r24, HIGH(RAMEND)
    out SPH, r24
    ser r23
    out DDRB, r23           ; PORTB as output.
    ldi r24, 0xf0           ; Turn off low nibble of r24.
    out DDRC, r24          ; High nibble of PORTC: output, low nibble: input.
    clr r23
    out PORTB, r23          ; Turn B-LEDs off.


FIRST_NUMBER:
    ldi r24,10             ; Allow for 10ms debouncing time.
    rcall scan_keypad_rising_edge ; Read pressed buttons in r25:r24.
    rcall keypad_to_ascii
    ldi temp, '0'
    cp r24, temp            ; Skip loop-back if 0 was pressed.
    brne FIRST_NUMBER

SECOND_NUMBER:
    ldi r24, 10                   ; Allow for 10ms debouncing time.
    rcall scan_keypad_rising_edge ; Read pressed buttons in r25:r24.
    rcall keypad_to_ascii         ; Convert to ASCII
    cpi r24, 0
    breq SECOND_NUMBER
    cpi r24, '0'            ; Skip loop-back if 0 was pressed.
    breq SECOND_NUMBER
    ldi temp, '5'
    cp r24, temp            ; Skip loop-back if 5 was pressed.
    brne FIRST_NUMBER

CODE_VERIFIED:
    ser r23                 ; Light all B-LEDs.
    out PORTB, r23
    ldi r24, LOW(3000)      ; Pause for 3sec.
    ldi r25, HIGH(3000)
    rcall wait_msec
    clr r23                             ; Turn-off all A-LEDs
```

```
    out PORTB, r23
    rjmp FIRST_NUMBER              ; Loop endlessly...


scan_keypad_rising_edge:
       mov r22 ,r24
       rcall scan_keypad
       push r24
       push r25
       mov r24 ,r22
       ldi r25 ,0
       rcall wait_msec
       rcall scan_keypad
       pop r23
       pop r22
       and r24 ,r22
       and r25 ,r23
       ldi r26 ,low(_tmp_)
       ldi r27 ,high(_tmp_)
       ld r23 ,X+
       ld r22 ,X
       st X ,r24
       st -X ,r25
       com r23
       com r22
       and r24 ,r22
       and r25 ,r23
       ret

scan_keypad:
       ldi r24 ,0x01
       rcall scan_row
       swap r24
       mov r27 ,r24
       ldi r24 ,0x02
       rcall scan_row
       add r27 ,r24
       ldi r24 ,0x03
       rcall scan_row
       swap r24
       mov r26 ,r24
       ldi r24 ,0x04
       rcall scan_row
       add r26 ,r24
       movw r24 ,r26
       ret

keypad_to_ascii:
       movw r26 ,r24
       ldi r24 ,'*'
       sbrc r26 ,0
       ret
       ldi r24 ,'0'
       sbrc r26 ,1
       ret
       ldi r24 ,'#'
       sbrc r26 ,2
       ret
       ldi r24 ,'D'
       sbrc r26 ,3
       ret
       ldi r24 ,'7'
       sbrc r26 ,4
       ret
       ldi r24 ,'8'
       sbrc r26 ,5
       ret
```

```
        ldi r24 ,'9'
        sbrc r26 ,6
        ret
        ldi r24 ,'C'
        sbrc r26 ,7
        ret
        ldi r24 ,'4'
        sbrc r27 ,0
        ret
        ldi r24 ,'5'
        sbrc r27 ,1
        ret
        ldi r24 ,'6'
        sbrc r27 ,2
        ret
        ldi r24 ,'B'
        sbrc r27 ,3
        ret
        ldi r24 ,'1'
        sbrc r27 ,4
        ret
        ldi r24 ,'2'
        sbrc r27 ,5
        ret
        ldi r24 ,'3'
        sbrc r27 ,6
        ret
        ldi r24 ,'A'
        sbrc r27 ,7
        ret
        clr r24
        ret

wait_msec:
        push r24
        push r25
        ldi r24 , low(998)
        ldi r25 , high(998)
        rcall wait_usec
        pop r25
        pop r24
        sbiw r24 , 1
        brne wait_msec
        ret

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

scan_row:
        ldi r25 ,0x08
back_: lsl r25
        dec r24
        brne back_
        out PORTC ,r25
        nop
        nop
        in r24 ,PINC
        andi r24 ,0x0f
        ret
```