ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Αναφορά Πρώτου Εργαστήριου RISC-V
Μάθημα: Εργαστήριο Μικροϋπολογιστών

Ονοματεπώνυμο: Μάρκος Γκέργκες
Αριθμός Μητρώου: 03117870

Ακαδημαϊκό Έτος 2020-21

Ερώτημα 1

Για το διάβασμα των διακοπτών, και την απεικόνιση των leds, χρησιμοποίηθηκαν οι ίδιες συναρτήσεις που ορίστηκαν στα πλαίσια του μαθήματος. Πιο συγκεκριμένα, η READ_GPIO είναι ένα macro το οποίο επιστρέφει το περιεχόμενο ενός δείκτη που παίρνει σαν παράμετρο. Παρόμοια, η WRITE_GPIO θέτει το περιεχόμενο ενός δείκτη με μια τιμή. Γίνεται typecasting σε 32bit unsigned, ενώ η δεσμευμένη λέξη volatile διασφαλίζει ότι ο μεταγλωττιστής θα διαβάζει κάθε φορά την τιμή της μεταβλητής από τη μνήμη. Έτσι, και σε περίπτωση διακοπής, η τιμή της μετβλητής θα είναι η σωστή.

Κώδικας C

```c
// From memory-map
#define GPIO_SWs     0x80001400
#define GPIO_LEDs    0x80001404
#define GPIO_INOUT   0x80001408

//define basic read-write macros
#define READ_GPIO(addr) (*(volatile unsigned*)addr)
#define WRITE_GPIO(addr, value) { (*(volatile unsigned *)addr) = (value);}

int main(void)
{
  volatile unsigned ddr_value=0xFFFF, msb_val, lsb_val, sum;

  WRITE_GPIO(GPIO_INOUT, ddr_value);              //set leds as output
  while (1) {
      msb_val = READ_GPIO(GPIO_SWs);
      msb_val = msb_val >> 28;                    //move 4msb of switches to lsb
      lsb_val = READ_GPIO(GPIO_SWs);
      lsb_val = lsb_val >> 16;                    //move 4lsb of switches to lsb
      sum = (msb_val & 0xF) + (lsb_val & 0xF);    //mask bits and add them
      if (sum < 16) {
      WRITE_GPIO(GPIO_LEDs, sum);
      } else {
        WRITE_GPIO(GPIO_LEDs, 0x10);              //switch 5th led ON(index-1)
      }

  }
  return 0;
}
```

Κώδικας Assembly

Επειδή το πρόγραμμα εκτελείται σε ένα SoC, πρέπει πρώτα να εκτελεστούν κάποιες εντολές αρχικοποίησης. Ο κώδικας που αφορά το πρόγραμμα μας βλέπουμε ότι ξεκινά στη θέση μνήμης 0x90. Για αποθήκευση στην μνήμη χρησιμοποιείται η στοίβα. Η κορυφή της στοίβας βρίσκεται στον καταχωρητή x2("sp"), ενώ το περιεχόμενο της στοίβας αυξάνεται προς τα κάτω. Ο δείκτης της στοίβας($x2$/sp) είναι κατά σύμβαση ευθυγραμμισμένος να δείχνει πάντα σε θέσεις μνήμης πολλαπλάσια του τεσσάρων λέξεων (quadword 16 byte aligned). Όποτε χρειάζεται νέος χώρος για προσωρινή αποθήκευση στην μνήμη, μειώνεται ο δείκτης στοίβας κατά ένα πολλαπλάσιο του 16, και στη συνέχεια με τη χρήση ενός θετικού offset από τον δείκτη μπορούμε να αναφερθούμε σε μια συγκεκριμένη θέση μνήμης.
Το extension που χρησιμοοποιείται επιτρέπει compression των κωδικών κάποιων εντολών από 4 bytes σε 2. Για λόγους "καθαρότητας" και κατανόησης του κώδικα, έχουν αφαιρεθεί οι κωδικοί των εντολών.

```
1
2  Disassembly of section .text.init:
3
4  00000000 <_start>:
5     0:   csrw   minstret,zero
6     4:   csrw   minstreth,zero
7     8:   li   ra,0        #Αρχικοποιούνται οι καταχωρητές
8     a:   li   sp,0
9     c:   li   gp,0
10    e:   li   tp,0
11   10:   li   t0,0
12   12:   li   t1,0
13   14:   li   t2,0
14   16:   li   s0,0
15   18:   li   s1,0
16   1a:   li   a0,0
17   1c:   li   a1,0
18   1e:   li   a2,0
19   20:   li   a3,0
20   22:   li   a4,0
21   24:   li   a5,0
22   26:   li   a6,0
23   28:   li   a7,0
24   2a:   li   s2,0
25   2c:   li   s3,0
26   2e:   li   s4,0
27   30:   li   s5,0
28   32:   li   s6,0
29   34:   li   s7,0
30   36:   li   s8,0
31   38:   li   s9,0
32   3a:   li   s10,0
33   3c:   li   s11,0
34   3e:   li   t3,0
35   40:   li   t4,0
36   42:   li   t5,0
37   44:   li   t6,0
38   46:   lui  t1,0x55555
39   4a:   addi   t1,t1,1365 # 55555555 <_sw_int_mem_ctrl+0x51555555>
40   4e:   csrw   0x7c0,t1
41   52:   auipc gp,0x3
42   56:   addi   gp,gp,-1778 # 2960 <__global_pointer$>
43   5a:   auipc sp,0x3
44   5e:   addi   sp,sp,262 # 3160 <_sp>
45   62:   auipc a0,0x2
46   66:   addi   a0,a0,254 # 2160 <__bss_start>
47   6a:   auipc a1,0x2
48   6e:   addi   a1,a1,246 # 2160 <__bss_start>
49   72:   bgeu   a0,a1,80 <_start+0x80>
```

```
   76:    sw   zero,0(a0)
   7a:    addi   a0,a0,4
   7c:    bltu   a0,a1,76 <_start+0x76>
   80:    jal  ea <__libc_init_array>
   82:    li   a0,0
   84:    li   a1,0
   86:    jal  90 <main>
   88:    j  88 <_start+0x88>
   8a:    unimp
   8c:    unimp
   ...

Disassembly of section .text:
#Εδώ ξεκινάει το πρόγραμμα μας
00000090 <main>:
   90:    addi   sp,sp,-16    #αυξάνουμε τη στοίβα κατά 4 λέξεις
   92:    lui  a5,0x10        #φορτώνουμε την τιμη 0x10 στα άνω 20 bits του a5
   94:    addi   a5,a5,-1     #αφαιρώντας 1, ο a5 παίρνει την τιμή 0xFFFF
   96:    sw   a5,12(sp)      #αποθηκεύουμε τον a5 στην στοίβα
   98:    lw   a4,12(sp)      #δίνουμε στον a4 την τιμή που είχε ο a5(0xFFFF)
   9a:    lui  a5,0x80001     #φορτώνουμε την σταθερά στα άνω 20 bits, a5=0x80001000
          # mem[a5+1032] = a4 , ισχύει πως 1032 = 0x408, άρα mem[0x800010408] = 0xFFFF [GPIO_INOUT]
   9e:    sw   a4,1032(a5) # 80001408 <OVERLAY_END_OF_OVERLAYS+0xa0001408>
   a2:    j  ae <main+0x1e>   #πραγματοποιούμε jump στην διεύθυνση a4, αντιστοιχεί στην αρχή της while
   a4:    lui  a5,0x80001     #αντιστοιχεί στο "else branch", a5 = 0x80001000
   a8:    li   a4,16          #στον a4 φορτώνουμε τη σταθερά για να ανάψει το 5ο led
          #χρησιμοποιώντας offset ανάβουμε το 5ο led, αντιστοιχεί σε "WRITE_GPIO(GPIO_LEDs,0x10)"
   aa:    sw   a4,1028(a5) # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
   ae:    lui  a4,0x80001     #φορτώνουμε τη σταθερά στα άνω 20 bits, τα υπόλοιπα=0
          # a5=mem[a4+1024], a5=mem[0x80001400], αντιστοιχεί στην msb_val=READ_GPIO(GPIO_SWs)
   b2:    lw   a5,1024(a4) # 80001400 <OVERLAY_END_OF_OVERLAYS+0xa0001400>
   b6:    sw   a5,8(sp)       #αποθηκεύουμε την τιμή των διακοπτών στη στοίβα
   b8:    lw   a5,8(sp)       #την ξαναφορτώνουμε στον a5 από την στοίβα
   ba:    srli   a5,a5,0x1c   #δεξιά ολίσθηση του a5 0x1c φορές-αντοιτοιχεί στην "msb_val >> 28"
   bc:    sw   a5,8(sp)       #αποθηκεύουμε την ολισθημένη τιμή των διακοπτών στη στοίβα
   be:    lw   a5,1024(a4)    #a5=mem[0x80001400], αντιστοιχεί στην lsb_val=READ_GPIO(GPIO_SWs)
   c2:    sw   a5,4(sp)       #αποθηκεύουμε την τιμή των διακοπτών στη στοίβα
   c4:    lw   a5,4(sp)       #την ξαναφορτώνουμε στον a5 από την στοίβα(λόγω volatile)
   c6:    srli   a5,a5,0x10   #δεξιά ολίσθηση του a5 0x10 φορές-αντοιτοιχεί στην "lsb_val >> 16"
   c8:    sw   a5,4(sp)       #αποθηκεύουμε την τιμή των lsb_val στη στοίβα
   ca:    lw   a5,8(sp)       #φορτώνουμε στον καταχωρητή a5 την τιμή της msb_val
   cc:    andi   a5,a5,15     #κάνουμε mask τα 4 lsb του a5, αντοιτοιχεί στο "msb_val & 0xF"
   ce:    lw   a4,4(sp)       #φορτώνουμε στον a4 την lsb_val από τη στοίβα
   d0:    andi   a4,a4,15     #επίσης mask τα 4 lsb, αντιστοιχεί στο "lsb_val & 0xF"
   d2:    add  a5,a5,a4       #προσθέτουμε τις 2 τιμές, αντιστοιχεί στην ανάθεση της sum(γραμμή 20 C)
   d4:    sw   a5,0(sp)       #αποθηκεύουμε το άθροισμα στην στοίβα
   d6:    lw   a4,0(sp)       #το φορτώνουμε στον a4 από τη στοίβα
   d8:    li   a5,15          #δίνουμε τη σταθερά 15 στον a5, θα χρησιμοποιηθεί για σύγκριση
          #Αν 15 < άθροισμα(a4), τότε jump στη διεύθυνση a4
   da:    bltu   a5,a4,a4 <main+0x14>
   de:    lw   a4,0(sp)       #φορτώνουμε την τιμή του αθροίσματος απ'τη στοίβα
   e0:    lui  a5,0x80001     #δίνουμε την τιμή x80001000 στον a5
          #και με ένα offset, ανάβουμε τα leds που πρέπει,όπως στην "WRITE_GPIO(GPIO_LEDs ,sum)"
   e4:    sw   a4,1028(a5) # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
   e8:    j  ae <main+0x1e>     #συνεχίζουμε το while loop

000000ea <__libc_init_array>:    #και άλλες αρχικοποιήσεις που αφορούν πιθανώς το SoC
   ea:    addi   sp,sp,-16
   ec:    sw   s0,8(sp)
   ee:    sw   s2,0(sp)
   f0:    auipc  s0,0x0
   f4:    addi   s0,s0,-240 # 0 <__comrv_align_size>
   f8:    auipc  s2,0x0
```

```
 fc:   addi  s2,s2,-248 # 0 <__comrv_align_size>
100:   sub s2,s2,s0
104:   sw  ra,12(sp)
106:   sw  s1,4(sp)
108:   srai  s2,s2,0x2
10c:   beqz  s2,11e <__libc_init_array+0x34>
110:   li  s1,0
112:   lw  a5,0(s0)
114:   addi  s1,s1,1
116:   addi  s0,s0,4
118:   jalr  a5
11a:   bne s2,s1,112 <__libc_init_array+0x28>
11e:   auipc s0,0x0
122:   addi  s0,s0,-286 # 0 <__comrv_align_size>
126:   auipc s2,0x0
12a:   addi  s2,s2,-294 # 0 <__comrv_align_size>
12e:   sub s2,s2,s0
132:   srai  s2,s2,0x2
136:   beqz  s2,148 <__libc_init_array+0x5e>
13a:   li  s1,0
13c:   lw  a5,0(s0)
13e:   addi  s1,s1,1
140:   addi  s0,s0,4
142:   jalr  a5
144:   bne s2,s1,13c <__libc_init_array+0x52>
148:   lw  ra,12(sp)
14a:   lw  s0,8(sp)
14c:   lw  s1,4(sp)
14e:   lw  s2,0(sp)
150:   addi  sp,sp,16
152:   ret
```

Ερώτημα 2

Για την υλοποίηση της καθυστέρησης, ορίζεται αυθαίρετα μια μεγάλη σταθερά η οποία καθυστερεί την εκτέλεση της ροής του προγράμματος, προσθέτοντας επιπλέον κύκλους ρολογιού όταν μια μεταβλητή αυξάνεται επαναληπτικά μέχρι να γίνει ίση της σταθεράς.

Για την εμφάνιση της άρνησης των διακοπτών στα LEDS εξόδου, απαιτείται κάθε φορά η αντιστροφή του σημαντικότερου bit απο 1 σε 0, και από 0 σε 1 αντίστοιχα. Στη συγκεκριμένη υλοποίηση χρησιμοποιείται μια μεταβλητή "previous_msb" για το λόγο αυτό.

Κώδικας C

```
1
2  // From memory-map
3  #define GPIO_SWs     0x80001400
4  #define GPIO_LEDs    0x80001404
5  #define GPIO_INOUT   0x80001408
6
7  #define DELAY    0x200000
8
9  //define basic read-write macros
10 #define READ_GPIO(addr) (*(volatile unsigned*)addr)
11 #define WRITE_GPIO(addr, value) { (*(volatile unsigned *)addr) = (value);}
12
13 int main(void)
14 {
15     volatile unsigned ddr_value=0xFFFF, switches_val, temp;
16     volatile unsigned ones_sum = 0, previous_msb = 0;
17     volatile unsigned i, timer;
18     WRITE_GPIO(GPIO_INOUT, ddr_value);          //set leds as output
19     previous_msb = READ_GPIO(GPIO_SWs);         //read switches
20     previous_msb &= 0x80000000;                 //keep msb for comparison
21     while (1) {
22         do {                                 //continue reading untill msb changes
23         switches_val = READ_GPIO(GPIO_SWs);
24         ;
25         } while ((switches_val & 0x80000000) == previous_msb);
26         previous_msb = switches_val & 0x80000000;      //Update msbit
27         switches_val = switches_val >> 16;
28         switches_val &= 0xFFFF;
29         switches_val ^= 0xFFFF;                        //flip bits using XOR
30         temp = switches_val;
31         ones_sum = 0;
32         for (i=0; i<16; i++) {                         //shift and count 1-bits
33             if (temp & 0x1) {
34                 ones_sum ++;
35             }
36             temp =  temp >> 1;
37         }
38
39         for (i=1; i<=ones_sum; i++) {              //switch ON and OFF with delay
40             timer = 0;
41             WRITE_GPIO(GPIO_LEDs, switches_val);
42             while(timer < DELAY) {
43                 timer++;
44             }
45             timer = 0;
46             WRITE_GPIO(GPIO_LEDs, 0);
47             while(timer < DELAY) {
48                 timer++;
49             }
50         }
```

```
    }
    return 0;
}
```

Κώδικας Assembly

```
Disassembly of section .text.init:

00000000 <_start>:     #Αρχικοποιούνται οι καταχωρητές
   0: csrw   minstret,zero
   4: csrw   minstreth,zero
   8: li   ra,0
   a: li   sp,0
   c: li   gp,0
   e: li   tp,0
  10: li   t0,0
  12: li   t1,0
  14: li   t2,0
  16: li   s0,0
  18: li   s1,0
  1a: li   a0,0
  1c: li   a1,0
  1e: li   a2,0
  20: li   a3,0
  22: li   a4,0
  24: li   a5,0
  26: li   a6,0
  28: li   a7,0
  2a: li   s2,0
  2c: li   s3,0
  2e: li   s4,0
  30: li   s5,0
  32: li   s6,0
  34: li   s7,0
  36: li   s8,0
  38: li   s9,0
  3a: li   s10,0
  3c: li   s11,0
  3e: li   t3,0
  40: li   t4,0
  42: li   t5,0
  44: li   t6,0
  46: lui t1,0x55555
  4a: addi  t1,t1,1365 # 55555555 <_sw_int_mem_ctrl+0x51555555>
  4e: csrw  0x7c0,t1
  52: auipc gp,0x3
  56: addi  gp,gp,-1650 # 29e0 <__global_pointer$>
  5a: auipc sp,0x3
  5e: addi  sp,sp,390 # 31e0 <_sp>
  62: auipc a0,0x2
  66: addi  a0,a0,382 # 21e0 <__bss_start>
  6a: auipc a1,0x2
  6e: addi  a1,a1,374 # 21e0 <__bss_start>
  72: bgeu  a0,a1,80 <_start+0x80>
  76: sw  zero,0(a0)
  7a: addi  a0,a0,4
  7c: bltu  a0,a1,76 <_start+0x76>
  80: jal 16a <__libc_init_array>
  82: li   a0,0
  84: li   a1,0
```

```
  86: jal 90 <main>
  88: j 88 <_start+0x88>
  8a: unimp
  8c: unimp
  ...

Disassembly of section .text:

00000090 <main>:
  90: addi  sp,sp,-32    #αυξάνουμε τη στοίβα κατά 8 λέξεις
  92: lui a5,0x10        #φορτώνουμε τη σταθερά στα άνω 20 bits του a5, a5= 0x00010000
  94: addi  a5,a5,-1     #μειώνοντας 1, γίνεται a5=0x0000FFFF
  96: sw  a5,28(sp)      #αποθηκεύουμε την τιμή του a5 στη στοίβα
  98: sw  zero,16(sp)    #αποθηκεύουμε την τιμή 0 στη στοίβα
  9a: sw  zero,12(sp)    #αποθηκεύουμε την τιμή 0 στη στοίβα
  9c: lw  a4,28(sp)      #φορτώνουμε την τιμή 0x0000FFFF από τη στοίβα στον a5
  9e: lui a5,0x80001     #φορτώνουμε τη σταθερά στα άνω 20 bits του a5, a5= 0x80001000
      #με ένα offset 0x408 από τον a5, θέτουμε τα leds σαν έξοδο στη διεύθυνση 0x80001408
  a2: sw  a4,1032(a5) # 80001408 <OVERLAY_END_OF_OVERLAYS+0xa0001408>
  a6: lw  a5,1024(a5)    #διαβάζουμε τους διακόπτες απτή θέση μνήμης 0x80001400
  aa: sw  a5,12(sp)      #αποθηκεύουμε την τιμή των διακοπτών στη στοίβα
  ac: lw  a5,12(sp)      #ξανα φορτώνουμε την τιμή των διακοπτών από τη στοίβα
  ae: lui a4,0x80000     #φορτώνουμε τη σταθερά στα άνω 20 bits του a4, a4= 0x80000000
  b2: and a5,a5,a4       #κάνουμε mask την τιμή των διακοπτών με τη σταθερά 0x80000000
  b4: sw  a5,12(sp)      # αποθηκεύουμε την τιμή του σημαντικότερου bit στη στοίβα
  b6: j e6 <main+0x56>   #κάνουμε jump στη διεύθυνση 0xe6
  b8: lw  a5,20(sp)      #φορτώνουμε την μεταβλητή temp από τη στοίβα
  ba: srli  a5,a5,0x1    #την ολισθαίνουμε δεξιά κατά 1
  bc: sw  a5,20(sp)      #ενημερώνουμε την τιμή της στην στοίβα
  be: lw  a5,8(sp)       #φορτώνουμε τον μετρητή i από τη στοίβα
  c0: addi  a5,a5,1      #τον αυξάνουμε κατά 1
  c2: sw  a5,8(sp)       #τον ενημερώνουμε στην στοίβα
  c4: lw  a4,8(sp)       #φορτώνουμε τον μετρητή i του for-loop από τη στοίβα
  c6: li  a5,15          #θέτουμε τον a=15
      #αν i>15 σπάμε το loop με άλμα στην θέση μνήμης da
  c8: blt a5,a4,da <main+0x4a>
  cc: lw  a5,20(sp)      #φορτώνουμε την μεταβλητή temp από τη στοίβα
  ce: andi  a5,a5,1      #κάνουμε mask κρατώντας μόνο το lsb
      #αν ισούται με 0, κάνουμε άλμα και δεν αυξάνουμε την ones_sum
  d0: beqz  a5,b8 <main+0x28>
  d2: lw  a5,16(sp)      #αλλιώς φορτώνουμε από τη στοίβα την ones_sum
  d4: addi  a5,a5,1      #και την αυξάνουμε κατά 1 επειδή βρήκαμε άσσο
  d6: sw  a5,16(sp)      #ενημερώνουμε την τιμή της στη στοίβα
  d8: j b8 <main+0x28>   #κάνουμε άλμα στη θέση b8, όπου γίνεται η ολίσθηση
  da: li  a5,1           #αρχικοποιούμε τον μετρητή i=1
  dc: sw  a5,8(sp)       #τον ενημερώνουμε στην στοίβα
  de: lw  a4,8(sp)       #φορτώνουμε τον i από την στοίβα(volatile)
  e0: lw  a5,16(sp)      #θέτουμε τον a5=ones_sum για την σύγκριση του forloop
      #αν ones_sum>= i κάνουμε άλμα στην θέση 128
  e2: bgeu  a5,a4,128 <main+0x98>
  e6: lui a5,0x80001     #φορτώνουμε τη σταθερά στα άνω 20 bits του a5, a5= 0x80001000
      #διαβάζουμε τους διακόπτες απτή θέση μνήμης 0x80001400
  ea: lw  a5,1024(a5) # 80001400 <OVERLAY_END_OF_OVERLAYS+0xa0001400>
  ee: sw  a5,24(sp)      #αποθηκεύουμε την τιμή των διακοπτών στη στοίβα
  f0: lw  a5,24(sp)      #ξανα φορτώνουμε την τιμή των διακοπτών από τη στοίβα(volatile)
  f2: lui a4,0x80000     #φορτώνουμε τη σταθερά στα άνω 20 bits του a4, a4= 0x80000000
  f6: and a5,a5,a4       #κάνουμε mask τη τιμή των διακοπτών κρατώντας το σημαντικότερο bit
  f8: lw  a4,12(sp)      #φορτώνουμε από τη στοίβα τη τιμή του προηγούμενου σημαντικότερου bit
      #αν είναι ίσες οι τιμές, jump στην θέση μνήμης 0xE6 και συνέχιση του do while()
  fa: beq a5,a4,e6 <main+0x56>  #αλλιώς είχαμε αλλαγή του σημαντικότερου bit
  fe: lw  a5,24(sp)      # φόρτωση της τιμής των διακοπτών από τη στοίβα
 100: lui a4,0x80000     #φορτώνουμε τη σταθερά στα άνω 20 bits του a4, a4= 0x80000000
 104: and a5,a5,a4       #κάνουμε mask τη τιμή των διακοπτών κρατώντας το σημαντικότερο bit
```

```
 106: sw   a5,12(sp)       #και ενημερώνουμε την τιμή του προηγούμενου σημαντικότερου bit στη στοίβα
 108: lw   a5,24(sp)       #φόρτωση της τιμής των διακοπτών από τη στοίβα
      #κάνουμε δεξιά ολίσθηση την τιμή κατά 16 φορές, για να τη φέρουμε στα 16 lsb
 10a: srli  a5,a5,0x10
 10c: sw   a5,24(sp)       #αποθηκεύουμε την ολισθημένη τιμή στη στοίβα
 10e: lw   a4,24(sp)       #την φορτώνουμε στον a4 από τη στοίβα(volatile)
 110: lui  a5,0x10         # φορτώνουμε τη σταθερά στα άνω 20 bits του a5, a5= 0x00010000
 112: addi  a5,a5,-1       # αφαιρώντας 1, ο a5 γίνεται 0x0000FFFF
 114: and  a4,a4,a5        # κάνουμε mask τα 16 lsb
 116: sw   a4,24(sp)       # τα αποθηκεύουμε στη στοίβα
 118: lw   a4,24(sp)       # τα φορτώνουμε από τη στοίβα
 11a: xor  a5,a5,a4        # αντιστρέφουμε τα 16 lsb, και τα αποθηκεύουμε στον a5
 11c: sw   a5,24(sp)       # αποθηκεύουμε την αντεστραμμένη τιμή στη στοίβα
 11e: lw   a5,24(sp)       # φορτώνουμε την αντεστραμμένη τιμή από τη στοίβα(volatile)
 120: sw   a5,20(sp)       # την αποθηκεύουμε στη θέση μνήμης mem[sp+20]
 122: sw   zero,16(sp)     #αποθηκεύουμε το 0 στη θέση μνήμης mem[sp+16]
 124: sw   zero,8(sp)      #αποθηκεύουμε το 0 στη θέση μνήμης mem[sp+8]-μετρητής loop
 126: j  c4 <main+0x34>    #πραγματοποιούμε άλμα στη θέση μνήμης c4
 128: sw   zero,4(sp)      #αρχικοποιούμε την mem[sp+4] = 0, θέση μεταβλητής timer
 12a: lw   a4,24(sp)       #φορτώνουμε την τιμή των διακοπτών στον a4
 12c: lui  a5,0x80001      #φορτώνουμε την τιμή 0x80001000
      #και με offset ανάβουμε τα leds, mem[0x80001404]=switches_val
 130: sw   a4,1028(a5) # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
 134: lw   a4,4(sp)        #φορτώνουμε την τιμή του timer
 136: lui  a5,0x200        #φορτώνουμε την σταθερά καθυστέρησης
      #Αν τελειώσει η καθυστέρηση, άλμα στην θέση 146
 13a: bge  a4,a5,146 <main+0xb6>
 13e: lw   a5,4(sp)        #φορτώνουμε την τιμή του timer
 140: addi  a5,a5,1        #αυξάνουμε κατά 1
 142: sw   a5,4(sp)        #την ενημερώνουμε στην στοίβα
 144: j  134 <main+0xa4> #και συνεχίζουμε το loop καθυστέρησης
 146: sw   zero,4(sp)      #έδώ τελείωσε η καθυστέρηηση, αρχικοποιούμε πάλι τον timer
 148: lui  a5,0x80001      #a5 = 0x80001000
      #θέτουμε τη θέση μνήμης των led=0
 14c: sw   zero,1028(a5) # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
 150: lw   a4,4(sp)        #φορτώνουμε τον timer
 152: lui  a5,0x200        #θέτουμε την σταθερά της σύγκρισης
      #Αν τελειώσει η καθυστέρηση, άλμα στην θέση 162
 156: bge  a4,a5,162 <main+0xd2>
 15a: lw   a5,4(sp)        #αλλιώς φορτώνουμε την τιμή του timer
 15c: addi  a5,a5,1        #τον αυξάνουμε κατά 1
 15e: sw   a5,4(sp)        #και τον ενημερώνουμε στην στοίβα
 160: j  150 <main+0xc0> #άλμα και συνέχιση του loop καθυστέρησης
 162: lw   a5,8(sp)        #φορτώνουμε τον μετρητή του εξωτερικού loop
 164: addi  a5,a5,1        #τον αυξάνουμε κατά 1
 166: sw   a5,8(sp)        #και τον ενημερώνουμε στην στοίβα
 168: j  de <main+0x4e>  #άλμα στη θέση de

0000016a <__libc_init_array>:
 16a: addi  sp,sp,-16
 16c: sw   s0,8(sp)
 16e: sw   s2,0(sp)
 170: auipc s0,0x0
 174: addi  s0,s0,-368 # 0 <__comrv_align_size>
 178: auipc s2,0x0
 17c: addi  s2,s2,-376 # 0 <__comrv_align_size>
 180: sub  s2,s2,s0
 184: sw   ra,12(sp)
 186: sw   s1,4(sp)
 188: srai  s2,s2,0x2
 18c: beqz  s2,19e <__libc_init_array+0x34>
 190: li   s1,0
 192: lw   a5,0(s0)
```

```
194: addi  s1,s1,1
196: addi  s0,s0,4
198: jalr  a5
19a: bne s2,s1,192 <__libc_init_array+0x28>
19e: auipc s0,0x0
1a2: addi  s0,s0,-414 # 0 <__comrv_align_size>
1a6: auipc s2,0x0
1aa: addi  s2,s2,-422 # 0 <__comrv_align_size>
1ae: sub s2,s2,s0
1b2: srai  s2,s2,0x2
1b6: beqz  s2,1c8 <__libc_init_array+0x5e>
1ba: li  s1,0
1bc: lw  a5,0(s0)
1be: addi  s1,s1,1
1c0: addi  s0,s0,4
1c2: jalr  a5
1c4: bne s2,s1,1bc <__libc_init_array+0x52>
1c8: lw  ra,12(sp)
1ca: lw  s0,8(sp)
1cc: lw  s1,4(sp)
1ce: lw  s2,0(sp)
1d0: addi  sp,sp,16
1d2: ret
```