Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εργαστήριο Μικροϋπολογιστών

$7^ο$ Εξάμηνο - Ροή Υ

Πρώτο Εργαστήριο **RISC-V**

Αμπατζή Ναυσικά - 031 17 198

Δήμος Δημήτρης - 031 17 165

Αθήνα
Ιανουάριος, 2021

1 Ερώτημα 1

Το πρώτο ζητούμενο της εργαστηριακής άσκησης αφορούσε τον σχεδιασμό προγράμματος σε γλώσσα C, το οποίο εμφανίζει στα 4 λιγότερο σημαντικά LEDs το άθροισμα των 4 πιο και των 4 λιγότερο σημαντικών bits των διακοπτών. Εάν προκύψει υπερχείλιση, τότε ανάβει το πέμπτο bit των LEDs. Ζητείται να θεωρηθούν μόνο θετικοί ακέραιοι και συμβολισμοί χωρίς πρόσημο.

Παραθέτουμε τον κώδικα C, ο οποίος εξηγείται μέσα από σύντομα σχόλια.

```c
#define GPIO_SWs    0x80001400
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408

#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }

int main (void) {
  int En_Value=0xFFFF, switches_value;
  unsigned int four_MSB, four_LSB, result, overflow = 0;

  WRITE_GPIO(GPIO_INOUT, En_Value);

  while(1) {
    switches_value = READ_GPIO(GPIO_SWs);   // read switches

    four_MSB = (switches_value & 0xF0000000) >> 28;  // isolate 4 MSBs
    four_LSB = (switches_value & 0x000F0000) >> 16;  // isolate 4 LSBs

    result = four_MSB + four_LSB;           // calculate result
    overflow = (result & 0x00000010) >> 4;  // check overflow

    if(overflow) {   // if we got an overflow: LED4 = ON (and only)
      WRITE_GPIO(GPIO_LEDs, 0x10);
    }
    else            // else show result
      WRITE_GPIO(GPIO_LEDs, result);
  }

  return(0);
}
```

Στη συνέχεια, παρατίθεται η disassembled εκδοχή του κώδικα όπως την παρήγαγε ο debugger. Σε αυτή έχουν προστεθεί επεξηγηματικά σχόλια αναφορικά με τη χρησιμότητα της κάθε εντολής.

```
0x00000090: 37 17 00 80       lui   a4,0x80001    # 0x80001000 is assigned to a4

0x00000094: c1 67             lui   a5,0x10       # (0x10 << 12 =) 0x00010000 is assigned to a5

0x00000096: fd 17             addi  a5,a5,-1      # a5 value changes to 0x00010000 - 0x00000001 =
                        # 0x0000FFFF

0x00000098: 23 24 f7 40       sw a5,1032(a4)      # a5 register content (0x0000FFFF) is stored in
                        # memory cell: base + offset = a4 + 1032 =
                        # 0x80001000 + 0x00000408 = 0x80001408
                        # This is to implement "WRITE_GPIO(GPIO_INOUT, En_Value)",
                        # which sets switches to inputs
                        # (zeros to 16 MSB of memory cell: 0x80001408)
                        # and LEDs to outputs (ones to 16 LSB of memory cell 0x80001408)

0x0000009c: 29 a0             j  0xa6 <main+22>   # Jump to 0xa6 = 0x000000a6,
                        # <main+22>: 22 lines "after" main = start
                        # Practically, PC gets 0x000000a6

                        # === This part is executed if NO overflow occurs ===
0x0000009e: 37 17 00 80       lui   a4,0x80001    # 0x80001000 is assigned to a4

0x000000a2: 23 22 f7 40       sw a5,1028(a4)      # Store a5 (result) to: 0x80001000 + 1028 =
                        # 0x80001000 + 0x0404 = 0x80001404
                        # Output result on corresponding LEDs
                        # ====================

0x000000a6: b7 17 00 80       lui   a5,0x80001    # 0x80001000 is assigned to a5

0x000000aa: 83 a7 07 40       lw a5,1024(a5)      # a5 = 0x80001000 + 1024 =
                        # 0x80001000 + 0x00000400 = 0x80001400 (switch value),
                        # ("switches_value = READ_GPIO(GPIO_SWs);")

0x000000ae: 13 d7 c7 01       srli  a4,a5,0x1c    # a4 gets switches value
                        # (logical) right shift by 0x1c = 28 positions (isolate 4 MSB switch)

0x000000b2: c1 87             srai  a5,a5,0x10    # a5 gets switches value
                        # (arithmetic) right shift by 0x10 = 16 positions (isolate 4 LSB switch)

0x000000b4: bd 8b             andi  a5,a5,15      # Keep only the 4 hex LSBs

0x000000b6: ba 97             add   a5,a5,a4      # a5 gets the result of a5 + a4

0x000000b8: 13 d7 47 00       srli  a4,a5,0x4     # Overflow check: right logical shift
                        # of the result (a5) by 4 positions
                        # and store in a4
                        # (if overflow, bit0 = 1)

0x000000bc: 05 8b             andi  a4,a4,1       # isolate overflow bit

0x000000be: 65 d3             beqz  a4,0x9e <main+14>  # if a4 = 0 (no overflow)
                        # jump to address 0x9e

0x000000c0: b7 17 00 80       lui   a5,0x80001    # otherwise, a5 <- 0x80001000

0x000000c4: 41 47             li a4,16      # a4 = 0x10

0x000000c6: 23 a2 e7 40       sw a4,1028(a5)      # store a4 (=0x10) to: 0x80001000 + 1028 =
                        # 0x80001000 + 0x0404 = 0x80001404
                        # LED5 = ON

0x000000ca: f1 bf             j  0xa6 <main+22>   # return to main loop
```

2 Ερώτημα 2

Το δεύτερο ζητούμενο της εργαστηριακής άσκησης αφορούσε τον σχεδιασμό προγράμματος σε γλώσσα C, το οποίο απεικονίζει επανειλημμένα στα LEDs (παρεμβάλλοντας αυθαίρετη καθυστέρηση) την άρνηση της τιμής των 16 διακοπτών, τόσες φορές όσοι άσσοι περιέχονται στην εν λόγω άρνηση. Κατά την επανειλημμένη απεικόνιση της άρνησης, οποιαδήποτε νέα είσοδος στους διακόπτες αγνοείται. Μετά και την τελευταία απεικόνιση, τα LEDs παραμένουν σβηστά μέχρι να αλλάξει τιμή ο πιο σημαντικός διακόπτης. Μέχρι τότε, ο χρήστης μπορεί να μεταβάλλει τους υπόλοιπους διακόπτες ως επιθυμεί.

Παραθέτουμε τον κώδικα C, ο οποίος εξηγείται μέσα από σύντομα σχόλια.

```c
#define GPIO_SWs    0x80001400
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408

#define READ_GPIO(dir) (*(volatile unsigned *)dir)
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value); }

int main (void) {
    int En_Value = 0xFFFF;
    unsigned int switches_value, negate, temp, count, MSB;

    WRITE_GPIO(GPIO_INOUT, En_Value);       // IO Set
    switches_value = READ_GPIO(GPIO_SWs);   // Read switches value

    while(1) {

        MSB = switches_value >> 31;                     // keep the MSB switch value
        negate = (switches_value >> 16) ^ (0xFFFF);     // keep and negate the 16 MSB
                                                        // (y XOR 1 = -y)
        temp = negate;                                  // temporary variable
        count = 0;                                      // counts the 1s in the negation

        // counting 1s process
        while(temp) {           // process ends when there are no more 1s
            if(temp & 1)        // if LSB == 1 then count++
                count++;
            temp = temp >> 1;   // get rid of the LSB by shifting left
        }

        while(count--) {        // loop as many times as the 1s (= count)
            WRITE_GPIO(GPIO_LEDs, negate);  // show negation on LEDs
            // delay
            WRITE_GPIO(GPIO_LEDs, 0);       // turn LEDs OFF
        }

        do {    // keep reading switches until MSB changes from the previous value
            switches_value = READ_GPIO(GPIO_SWs);
        } while(((switches_value & 0x80000000)>>31) == MSB); // checks if new MSB == previous MSB

    }
    return(0);
}
```

Στη συνέχεια, παρατίθεται η disassembled εκδοχή του κώδικα όπως την παρήγαγε ο debugger. Σε αυτή έχουν προστεθεί επεξηγηματικά σχόλια αναφορικά με τη χρησιμότητα της κάθε εντολής.

```
            # ===== Like ex1 (sum.c): "WRITE_GPIO(GPIO_INOUT, En_Value);"
            # Sets switches as inputs and LEDs as outputs
0x00000090: 37 17 00 80       lui   a4,0x80001
0x00000094: c1 67             lui   a5,0x10
0x00000096: fd 17             addi  a5,a5,-1
0x00000098: 23 24 f7 40       sw a5,1032(a4)
            # ======================================================
```

```
0x0000009c: 83 26 07 40       lw a3,1024(a4) # a3 = mem[a4 + 1024]
                                             # = mem[0x80001000 + 0x400]
                                             # = mem[0x80001400]
                                             # = switches value ==>
                                             # a3 = switches value

0x000000a0: 1d a8             j  0xd6 <main+70> # jump unconditionally to address
                                             # 0xd6 (== 70 memory cells after beginning)


            # ===== THIS IS THE LOOPS SECTION: WHILEs ARE IMPLEMENTED =====
0x000000a2: 05 83             srli  a4,a4,0x1   # a4 = a4 >> 1 ("temp = temp >> 1;")

            # ===== "while(temp)" loop =====
0x000000a4: 11 c7             beqz  a4,0xb0 <main+32> # if a4 = 0 (temp = 0)
                                             # then jump to 0xb0 (end loop)
0x000000a6: 13 76 17 00       andi  a2,a4,1     # a2 = LSB(a4)
0x000000aa: 65 de             beqz  a2,0xa2 <main+18> # if it's zero
                                             # then jump to 0xa2 (continue looping)
0x000000ac: 85 07             addi  a5,a5,1      # otherwise, it's an 1, so count++
0x000000ae: d5 bf             j  0xa2 <main+18>  # jump to 0xa2, to continue looping


            # ===== "while(count--)" loop =====
0x000000b0: 13 87 f7 ff       addi  a4,a5,-1    # a4 = count - 1

0x000000b4: 89 cb             beqz  a5,0xc6 <main+54>  # if count (= a5) == 0, then end loop

0x000000b6: b7 17 00 80       lui   a5,0x80001  # a5 = 0x80001000

0x000000ba: 23 a2 d7 40       sw a3,1028(a5)    # a3 -> mem[a5 + 1028] = mem[0x80001404]: LEDs ON

0x000000be: 23 a2 07 40       sw zero,1028(a5)  # 0  -> mem[a5 + 1028] = mem[0x80001404]: LEDs OFF

0x000000c2: ba 87             mv a5,a4    # a5 gets the reduced counter value from line 0x000000b0

0x000000c4: f5 b7             j  0xb0 <main+32> # continue looping from address 0xb0


            # ===== "do - while" loop =====
0x000000c6: b7 17 00 80       lui   a5,0x80001  # a5 = 0x80001000

0x000000ca: 83 a6 07 40       lw a3,1024(a5)    # a3 = mem[0x80001400]
                                             # reads switch value

0x000000ce: 93 d7 f6 01       srli  a5,a3,0x1f  # shifts right by 31 positions:
                                             # keep MSB switch value in a5

0x000000d2: e3 8a b7 fe       beq   a5,a1,0xc6 <main+54>  # if MSB(new) == MSB(prev), loop again
                                        # else, continue program with the new switches value in a3
            # ========================================================

            # this implements: "MSB = switches_value >> 31;"
0x000000d6: 93 d5 f6 01       srli  a1,a3,0x1f  # a1 = a3 >> 31
                                             # = MSB(a3)
                                             # = MSB(switches value)

            # this implements: "negate = (switches_value >> 16) ^ (0xFFFF);"
0x000000da: c1 82             srli  a3,a3,0x10  # a3 = a3 >> 16

0x000000dc: c1 67             lui   a5,0x10     # a5 = (0x10 << 12) = 0x00010000

0x000000de: fd 17             addi  a5,a5,-1    # a5 = 0x0000FFFF

0x000000e0: bd 8e             xor   a3,a3,a5    # a3 = a3 XOR 0xFFFF = -a3 (1s complement)
            # ============================

            # this implement: "temp = negate;"
0x000000e2: 36 87             mv a4,a3    # (temp =) a4 <- switch value negation

```

```
79              # this implements: "count = 0;"
80 0x000000e4: 81 47              li a5,0      # a5 = 0
81
82 0x000000e6: 7d bf              j  0xa4 <main+20> # jump unconditionally to address 0xa4
```