ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ

(2020-2021)

5η ΟΜΆΔΑ ΑΣΚΉΣΕΩΝ

Ονοματεπώνυμο:

- Χρήστος Τσούφης

Αριθμός Μητρώου:

- 03117176

Ομάδα Εργαστηρίου:

- B19

Εξέταση – Επίδειξη:

- 13/1/2021

Στοιχεία Επικοινωνίας:

- el17176@mail.ntua.gr

1η Άσκηση

Ο πηγαίος κώδικας, μαζί με τα απαραίτητα σχόλια:

Κώδικας σε **C**:

```c
#define GPIO_SWs    0x80001400 // defines the memory position of the switches
#define GPIO_LEDs   0x80001404 // defines the memory position of the LEDs
#define GPIO_INOUT  0x80001408 // defines the memory position of input (switches) /
output (leds)

#define READ_GPIO(dir) (*(volatile unsigned *)dir) // reads from dir and returns the
 memory position of dir
#define WRITE_GPIO(dir, value) {(*(volatile unsigned *)dir) = (value);} // finds the
 memory position of dir and writes on it the new value

int main (void){
    // initializations
    int En_Value = 0xFFFF;
    int switches_val, first, second;
    // configuration part
    WRITE_GPIO(GPIO_INOUT, En_Value); // it is used to declare that LEDs will be use
d for ouput and switches as input

    while (1){
        switches_val = READ_GPIO(GPIO_SWs); // read the value of switches
        // it is shifted to the right because the value of switches is in MSB
        first = (0x000f0000 & switches_val) >> 16; // mask
        second = (0xf0000000 & switches_val) >> 28; // mask
        switches_val = first + second; // add 4 LSB with 4 MSB

        // write to LEDs
        if (switches_val <= 15){WRITE_GPIO(GPIO_LEDs, switches_val);}
        else{WRITE_GPIO(GPIO_LEDs, 16);} // overflow
    }
    return(0);
}
```

Κώδικας σε **assembly** (με τα απαραίτητα σχόλια κάτω από κάθε γραμμή):

```
0x00000090: 37 17 00 80          lui   a4,0x80001

# load upper immediate / 0x80001 is placed in the leftmost 20 bits of reg
a4 and the rightmost 12 bits are set to zero

0x00000094: c1 67                lui   a5,0x10

# load upper immediate / 0x10 is placed in the leftmost 20 bits of reg a5
and the rightmost 12 bits are set to zero

0x00000096: fd 17                addi  a5,a5,-1

# add immediate / a5 = a5 - 1

0x00000098: 23 24 f7 40          sw    a5,1032(a4) # 0x80001408

# store word / a 32-bit value is copied from a5 to a4

0x0000009c: 31 a0                j     0xa8 <main+24>

# jump / jump to 0xa8

0x0000009e: b7 17 00 80          lui   a5,0x80001

# load upper immediate / 0x80001 is placed in the leftmost 20 bits of reg
a4 and the rightmost 12 bits are set to zero

0x000000a2: 41 47                li    a4,16

# load immediate / immediate value 16 is copied into reg a4

0x000000a4: 23 a2 e7 40          sw    a4,1028(a5) # 0x80001404

# store word / a 32-bit value is copied from a5 to a4

0x000000a8: b7 17 00 80          lui   a5,0x80001

# load upper immediate / 0x80001 is placed in the leftmost 20 bits of reg
a5 and the rightmost 12 bits are set to zero

0x000000ac: 03 a7 07 40          lw    a4,1024(a5) # 0x80001400

# load word / a 32-bit value is fetched from memory and moved into reg a4

0x000000b0: 93 57 07 41          srai  a5,a4,0x10

# shift right arithmetic immediate / the contents of a4 is shifted right
0x10 bits and the result is placed in a5

0x000000b4: bd 8b                andi  a5,a5,15

# and immediate / 15 is logically ANDed to a5 and the result is placed in
a5
```

```
0x000000b6: 71 83                srli  a4,a4,0x1c
```

shift right logical immediate / the contents of a4 is shifted right 0x1c bits and the result is placed in a4

```
0x000000b8: ba 97                add   a5,a5,a4
```

add / a5 = a5 + a4

```
0x000000ba: 3d 47                li    a4,15
```

load immediate / immediate value 15 is copied into reg a4

```
0x000000bc: e3 41 f7 fe          blt   a4,a5,0x9e <main+14>
```

branch if not equal / the contents of a4 is compared to the contents of a5 and if a4 is less than a5, control jumps to a PC-relative target address which is 0x9e

```
0x000000c0: 37 17 00 80          lui   a4,0x80001
```

load upper immediate / 0x80001 is placed in the leftmost 20 bits of reg a4 and the rightmost 12 bits are set to zero

```
0x000000c4: 23 22 f7 40          sw    a5,1028(a4) # 0x80001404
```

store word / a 32-bit value is copied from a4 to a5

```
0x000000c8: c5 b7                j     0xa8 <main+24>
```

jump / jump to 0xa8

2η Άσκηση

Ο πηγαίος κώδικας, μαζί με τα απαραίτητα σχόλια:

Κώδικας σε **C**:

```
#define GPIO_SWs    0x80001400 // defines the memory position of the switches
#define GPIO_LEDs   0x80001404 // defines the memory position of the LEDs
#define GPIO_INOUT  0x80001408 // defines the memory position of input (switches) /
output (leds)

#define READ_GPIO(dir) (*(volatile unsigned *)dir) // reads from dir and returns the
 memory position of dir
#define WRITE_GPIO(dir, value) { (*(volatile unsigned *)dir) = (value);} // finds th
e memory position of dir and writes on it the new value

// function that lights ON/OFF the LEDs
void leds(int complem_val, int ones){ // the arguments are the switches that will sw
itch on and the sum
    int cntr = 65000; // virtual delay
    while(ones > 0){
        WRITE_GPIO(GPIO_LEDs, complem_val); // switch on the LEDs of complement
        while(cntr > 0){cntr--;} // loop that reduces counter
        WRITE_GPIO(GPIO_LEDs, 0); // switch off LEDs due to 0
        cntr = 6500; // virtual delay
        while(cntr > 0){cntr--;} // loop that reduces counter
        cntr = 6500; // virtual delay
        ones--; // reduce sum
    }
}

int main (void){
    // initializations
    int En_Value = 0xFFFF;
    WRITE_GPIO(GPIO_INOUT, En_Value);
    unsigned int comp_val, switches_val, prev_val = 0, switches_val_initial = 0;
    int cntr = 0, sum = 0, last_bit = 0;

    while(1){ // for continuous operation
        switches_val = READ_GPIO(GPIO_SWs); // read the value of switches & place th
em accordingly for the LEDs
        switches_val >>= 16; // shift what you read 16 positions to the right (LSB)
so that LEDs can 'understand'
        switches_val_initial = (switches_val << 16); // shift 16 positions to the le
ft to take the initial value
        comp_val = ~switches_val; // calculate the complement value of switches
```

```
        // calculation of ones
        while(cntr <= 15){ // it counts 16 times since the value of the switches is
on the 16 LSB
            if((last_bit = (comp_val >> cntr) & 1) == 1){ // initially it does 0 shi
fts, then 1 shift etc
                sum++; // calculats the number of ones
            }
            cntr++; // counter is used for the number of loops
        }

        leds(comp_val, sum); // go to the function above

        prev_val = switches_val_initial;
        switches_val = switches_val_initial;
        // for as long as the MSB of the switches remains the same, just run
        // but if the MSB of the switches value changes, repeat the whole process
        while((switches_val & 0x80000000) == (prev_val & 0x80000000)){ // mask these
 values
            switches_val &= 0x80000000;
        }

        sum = 0 ;
        cntr = 0;
    }
    return(0);
}
```

Κώδικας σε **assembly** (με τα απαραίτητα σχόλια κάτω από κάθε γραμμή):

```
0x00000090: c1 67                lui   a5,0x10
# load upper immediate / 0x10 is placed in the leftmost 20 bits of reg a5
and the rightmost 12 bits are set to zero
0x00000092: 93 87 87 de          addi  a5,a5,-536 # 0xfde8
# add immediate / a5 = a5 - 536
0x00000096: 1d a0                j     0xbc <leds+44>
# jump / jump to 0xbc
0x00000098: fd 17                addi  a5,a5,-1
# add immediate / a5 = a5 - 1
0x0000009a: e3 4f f0 fe          bgtz  a5,0x98 <leds+8>
# blt x0, a5, 0x98 / which means: Branch if > zero
0x0000009e: b7 17 00 80          lui   a5,0x80001
# load upper immediate / 0x80001 is placed in the leftmost 20 bits of reg
a5 and the rightmost 12 bits are set to zero
0x000000a2: 23 a2 07 40          sw    zero,1028(a5) # 0x80001404
# store word / a 32-bit value is copied from zero to a5
0x000000a6: 89 67                lui   a5,0x2
# load upper immediate / 0x2 is placed in the leftmost 20 bits of reg a5
and the rightmost 12 bits are set to zero
0x000000a8: 93 87 47 96          addi  a5,a5,-1692 # 0x1964
# add immediate / a5 = a5 -1692
0x000000ac: 63 54 f0 00          blez  a5,0xb4 <leds+36>
#  bge x0, a5, 0xb4 / which means: Branch if ≤ zero
0x000000b0: fd 17                addi  a5,a5,-1
# add immediate / a5 = a5 - 1
0x000000b2: ed bf                j     0xac <leds+28>
# jump / jump to 0xac
0x000000b4: fd 15                addi  a1,a1,-1
# add immediate / a1 = a1 - 1
```

```
0x000000b6: 89 67                lui   a5,0x2

# load upper immediate / 0x2 is placed in the leftmost 20 bits of reg a5
and the rightmost 12 bits are set to zero

0x000000b8: 93 87 47 96          addi  a5,a5,-1692 # 0x1964

# add immediate / a5 = a5 -1692

0x000000bc: 63 57 b0 00          blez  a1,0xca <leds+58>

#  bge x0, a5, 0xb4 / which means: Branch if ≤ zero

0x000000c0: 37 17 00 80          lui   a4,0x80001

# load upper immediate / 0x2 is placed in the leftmost 20 bits of reg a5
and the rightmost 12 bits are set to zero

0x000000c4: 23 22 a7 40          sw    a0,1028(a4) # 0x80001404

# store word / a 32-bit value is copied from a0 to a4

0x000000c8: c9 bf                j     0x9a <leds+10>

# jump / jump to 0x9a

0x000000ca: 82 80                ret

# return / returns from a subroutine/function
```