$5^{η}$ Εργαστηριακή Άσκηση στο Εργαστήριο Μικροεπεξεργαστών

Ομάδα Β 3

Αλέξανδρος Κυριακάκης (03112163),
Ιωάννης Αλεξόπουλος (03117001)

Ιανουάριος 2021

$1^{η}$ Άσκηση

C Code

```c
// memory-mapped I/O addresses
#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, high, low, sum;

  WRITE_GPIO(GPIO_INOUT, En_Value);

  while (1) {
    switches_value = READ_GPIO(GPIO_SWs) >> 16;   // read value on switches and
    ↪  shift
    low = switches_value & 0xF;
    high = (switches_value & 0xF000) >> 12;
    sum = high + low;
    if (sum < 16) {
      WRITE_GPIO(GPIO_LEDs, sum);  // display switch value on LEDs
    }
    else {
      sum = 16;
      WRITE_GPIO(GPIO_LEDs, sum);  // display switch value on LEDs

    }
```

```
28
29    }
30
31    return(0);
32  }
```

Assembly Code

```
1  0x00000090: 37 17 00 80      lui a4,0x80001 # load 0x80001 on 20 most significant
   ↪  bits of a4
2  0x00000094: c1 67            lui a5,0x10    # load 0x10 on 20 most significant
   ↪  bits of a5
3  0x00000096: fd 17            addi a5,a5,-1 # subtract 1
4  0x00000098: 23 24 f7 40      sw a5,1032(a4) # store value of a5 (0xFFFF) to memory
   ↪  address 0x80001408
5  0x0000009c: 31 a0            j 0xa8 <main+24> # jump to address 0xa8
6  0x0000009e: b7 17 00 80      lui a5,0x80001  # load 0x80001 on 20 most significant
   ↪  bits of a5
7  0x000000a2: 41 47            li a4,16 # a4 = 16
8  0x000000a4: 23 a2 e7 40      sw a4,1028(a5) # store value of a4 (16 = 0x10) to
   ↪  memory address 0x80001404
9  0x000000a8: b7 17 00 80      lui a5,0x80001  # load 0x80001 on 20 most significant
   ↪  bits of a5
10 0x000000ac: 83 a7 07 40      lw a5,1024(a5) # a5 = mem[0x80001400] - load swithes
11 0x000000b0: c1 83            srli a5,a5,0x10 # shift switches value logically 16
   ↪  times
12 0x000000b2: 13 f7 f7 00      andi a4,a5,15 # andi to keep 4 LSB
13 0x000000b6: b1 83            srli a5,a5,0xc #  shift switches value logically 12
   ↪  times so original 4 MSB in position of 4 LSB
14 0x000000b8: ba 97            add a5,a5,a4 # add 4 LSB and 4 MSB
15 0x000000ba: 3d 47            li a4,15 # load immediate 15 = 0xFFFF
16 0x000000bc: e3 41 f7 fe      blt a4,a5,0x9e <main+14> # if less sum > 15 take the
   ↪  branch that stores 0x10 to leds
17 0x000000c0: 37 17 00 80      lui a4,0x80001 # # load 0x80001 on 20 most
   ↪  significant bits of a4
18 0x000000c4: 23 22 f7 40      sw a5,1028(a4) # mem[0x80001404] = a5 - sum is stored
   ↪  in leds
19 0x000000c8: c5 b7            j 0xa8 <main+24> # jump to loop again
```

2^{η} Άσκηση

C Code

```
1  // memory-mapped I/O addresses
2  #define GPIO_SWs    0x80001400
3  #define GPIO_LEDs   0x80001404
```

```c
#define GPIO_INOUT  0x80001408

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

int do_the_job (void){
  int number_of_ace = 0, switches_value;
  switches_value = READ_GPIO(GPIO_SWs);
  msb = switches_value & 0x8000;
  for (int i = 0; i < 16; i++){
    if ((switches_value >> i) & 0x1) { // an vreis asso
      WRITE_GPIO(GPIO_LEDs, switches_value ^ 0xffff); // grapse thn arnhsh sta led
      for(int j =0; j < 1000; j++) if (j > 998 ) number_of_ace++; // delay!!! we use number_of_ace just to avoid optimization
      WRITE_GPIO(GPIO_LEDs, 0x0);  // svise ta
      for(int j =0; j < 1000; j++) if (j > 998 ) number_of_ace++; // delay!!! we use number_of_ace just to avoid optimization
    }
  }
  return(number_of_ace/2); // Ayto to kanoume mono gia apofygh tou optimization apo ton compiler
}

int main ( void )
{
  int En_Value=0xFFFF;

  WRITE_GPIO(GPIO_INOUT, En_Value);

  do_the_job(); // arxikopoiei to msb
  while (1) {
    if ((READ_GPIO(GPIO_SWs) & 0x8000) != msb) do_the_job(); // an allaksei to msb kane thn do the job.
  }

  return(0);
}
```

Assembly Code

Do the job (function)

```
0x00000090: b7 17 00 80      lui a5,0x80001
0x00000094: 03 a6 07 40      lw a2,1024(a5) # Read Switches value and write it to a2
0x00000098: a1 67            lui a5,0x8
0x0000009a: f1 8f            and a5,a5,a2 # Logical AND
```

```
0x0000009c: 09 67          lui a4,0x2
0x0000009e: 23 2c f7 1a    sw a5,440(a4)  # Write the Switches-MSB to <msb>
0x000000a2: 81 46          li a3,0  # a3 =:= i
0x000000a4: 81 45          li a1,0  # a1 =:= number of aces
0x000000a6: 35 a8          j 0xe2 <do_the_job+82> # jump to (line 29)
0x000000a8: 85 07          addi a5,a5,1 # Move on to the 1000 loops
0x000000aa: 13 07 70 3e    li a4,999 # Delay "for loop"
0x000000ae: 63 48 f7 00    blt a4,a5,0xbe <do_the_job+46> # if 999 < j go to
    ↪  line 17
0x000000b2: 13 07 60 3e    li a4,998
0x000000b6: e3 59 f7 fe    bge a4,a5,0xa8 <do_the_job+24>
0x000000ba: 85 05          addi a1,a1,1
0x000000bc: f5 b7          j 0xa8 <do_the_job+24> # go to line 10
0x000000be: b7 17 00 80    lui a5,0x80001 # switch off the leds
0x000000c2: 23 a2 07 40    sw zero,1028(a5) # 0x80001404
0x000000c6: 81 47          li a5,0 # init a5
0x000000c8: 11 a0          j 0xcc <do_the_job+60> # go to line 22
0x000000ca: 85 07          addi a5,a5,1
0x000000cc: 13 07 70 3e    li a4,999 # Next delay
0x000000d0: 63 48 f7 00    blt a4,a5,0xe0 <do_the_job+80> # if 999 < j go to 28
0x000000d4: 13 07 60 3e    li a4,998
0x000000d8: e3 59 f7 fe    bge a4,a5,0xca <do_the_job+58>
0x000000dc: 85 05          addi a1,a1,1
0x000000de: f5 b7          j 0xca <do_the_job+58>
0x000000e0: 85 06          addi a3,a3,1 # i++ move on to the next bit
0x000000e2: bd 47          li a5,15 # max(i) =:= a5 = 15
0x000000e4: 63 cf d7 00    blt a5,a3,0x102 <do_the_job+114> # if i < 15 go to
    ↪  0x102 (line 41)
0x000000e8: b3 57 d6 40    sra a5,a2,a3 # a5 = switches_value >> i
0x000000ec: 85 8b          andi a5,a5,1 # a5 = switches_value >> i) & 0x1
0x000000ee: ed db          beqz a5,0xe0 <do_the_job+80> # if a5 == 0 go to (line
    ↪  28)
0x000000f0: c1 67          lui a5,0x10 # else grapse thn arnhsh sta led
0x000000f2: fd 17          addi a5,a5,-1 # a5 == 0xffff
0x000000f4: b1 8f          xor a5,a5,a2 # a5 = switches_value ^ 0xffff
0x000000f6: 37 17 00 80    lui a4,0x80001
0x000000fa: 23 22 f7 40    sw a5,1028(a4) # Mem(Leds) <- a5
0x000000fe: 81 47          li a5,0 # a5 = 0
0x00000100: 6d b7          j 0xaa <do_the_job+26> # jump to line 11
0x00000102: 93 d7 f5 01    srli a5,a1,0x1f # Right shift: a5 = a1 >> 31
0x00000106: be 95          add a1,a1,a5 # a1 += a5
0x00000108: 13 d5 15 40    srai a0,a1,0x1 # arithmetic shift right a0 = a1 >> 1
0x0000010c: 82 80          ret
```

Main (function)

```
# Main function

addi sp,sp,-16
sw ra,12(sp) # fort
lui a4,0x80001 # a4 = GPIO_INOUT
lui a5,0x10 # a5 = En_Value
addi a5,a5,-1 # a5 = 0xffff
sw a5,1032(a4) # a5 -> Mem(0x80001408)
jal 0x90 <do_the_job> # PC-relative jump to do_the_job and save PC on ra
j 0x124 <main+22>  # jump to the "while"
jal 0x90 <do_the_job> # PC-relative jump to do_the_job and save PC on ra
lui a5,0x80001
lw a5,1024(a5) # Read Switches
lui a4,0x8
and a5,a5,a4 # Hold the MSB
lui a4,0x2  # Bring the already saved <msb> on a4
lw a4,440(a4) # 0x21b8 <msb>
beq a5,a4,0x124 <main+22> # if <msb> == Switches_MSB then jump to while inside
↪  main
j 0x122 <main+20> # else jump to do_the_job function
```