SOLUZIONE COMPLETA ESERCIZIO 1 DEL 30/06/2016

Il codice dovra' essere organizzato in tre moduli: A) codice utente; B) codice libreria (libreria arduino, composta essenzialmente da syscall wrappers); C) codice kernel (mini-drivers delle periferiche coinvolte).

PARTE A (CODICE UTENTE)

```
.data
led: .word 13
pushButton: .word 2
buttonState: .word 0

.extern pinMode
.extern Serial.begin
.extern Serial.println
.extern digitalRead
.extern digitalWrite
.extern delay

.text
#--------------------------------------
# SETUP
setup:
   addi $sp, $sp, -4  #alloc stackspace
   sw   $ra, 0($sp)   #save old-ra

   la   $t0, led      #punta a led
   lw   $a0, 0($t0)   #legge led
   add  $a1, $0, $0   #setta II param.
   jal  pinMode

   addi $a0, $0, 9600 #setta il I param.
   jal  Serial.begin

   la   $t0, pushButton #punta a pushButton
   lw   $a0, 0($t0)   #legge pushButton
   addi $a1, $0, 1    #setta II param.
   jal  pinMode

   lw   $ra, 0($sp)   #restore old-ra
   addi $sp, $sp, 4   #deallocate stack space
   jr $ra

#--------------------------------------
# LOOP
loop:
   addi $sp, $sp, -4  #alloc stackspace
   sw   $ra,0($sp)    #save old-ra

   la   $t0, led      #punta a led
   lw   $a0, 0($t0)   #legge led
   addi $a1, $0, 1    #setta II param.
   jal  digitalWrite

   addi $a0, $0, 1000 #setta I param.
   jal  delay

   la   $t0,pushButton
   lw   $a0, 0($t0)
   jal  digitalRead

   la $t0, buttonState
   sw   $v0, 0($t0)

   add  $a0, $0, $v0  #prep. I param.

   la   $t0, led      #punta a led
   lw   $a0, 0($t0)   #legge led
   addi $a1, $0, 0    #setta il II param.
   jal  digitalWrite

   addi $a0, $0, 1000 #setta I param.
   jal  delay

   lw   $ra, 0($sp)   #restore old-ra
   addi $sp, $sp, 4   #deallocate stack space
   jr   $ra
```

PARTE B (CODICE LIBRERIA)

```
.text
.globl pinMode
.globl Serial.begin
.globl Serial.println
.globl digitalRead
.globl digitalWrite
.globl delay

Serial.begin:
   addi $sp, $sp -4
   sw   $ra 0($sp)
   addi $v0, $0, 21 #syscall 21
   teq  $0,$0     ###SIM:emulo syscall
   lw   $ra 0($sp)
   addi $sp, $sp, 4
   jr   $ra

Serial.println:
   addi $sp, $sp -4
   sw   $ra 0($sp)
   addi $v0, $0, 22 #syscall 22
   teq  $0,$0     ###SIM:emulo syscall
   lw   $ra 0($sp)
   addi $sp, $sp, 4
   jr   $ra

digitalRead:
   addi $sp, $sp -4
   sw   $ra 0($sp)
   addi $v0, $0, 23 #syscall 23
   teq  $0,$0     ###SIM:emulo syscall
   lw   $ra 0($sp)
   addi $sp, $sp, 4
   jr   $ra

digitalWrite:
   addi $sp, $sp -4
   sw   $ra 0($sp)
   addi $v0, $0, 24 #syscall 24
   teq  $0,$0     ###SIM:emulo syscall
   lw   $ra 0($sp)
   addi $sp, $sp, 4
   jr   $ra

pinMode:
   addi $sp, $sp -4
   sw   $ra 0($sp)

   lw   $ra 0($sp)
   addi $sp, $sp, 4
   jr   $ra

delay:
   addi $sp, $sp -4
   sw   $ra 0($sp)
   addi $v0, $0, 26 #syscall 26
   teq  $0,$0     ###SIM:emulo syscall
   lw   $ra 0($sp)
   addi $sp, $sp, 4
   jr   $ra
```

PARTE C(CODICE KERNEL)

```
#####################################
# EXCEPTION HANDLER (OPZIONALE PER SPIM TEST)
# (per comodit□ettere opzione QUIET di SPIM)
# NOTA: non 和n exc. handler generale,
# questo vale solo per questo esercizio
.kdata
saveframe: .space(4*8) # per v0,a0,a1,t0...t4
_syscallmsg: .asciiz "Syscall "
_nl: .asciiz "\n"

.ktext 0x80000180
   la   $k0, saveframe
   sw   $v0, 0($k0)
   sw   $a0, 4($k0)
   sw   $a1, 8($k0)
   sw   $t0, 12($k0)
   sw   $t1, 16($k0)
   sw   $t2, 20($k0)
   sw   $t3, 24($k0)
   sw   $t4, 28($k0)

   # Print a message (optional)
   addi $v0, $0, 4  # print_str
   la   $a0, _syscallmsg
   syscall
   lw   $a0, 0($k0) # pop $v0
   addi $v0, $0, 1  # print_int
   syscall
   addi $v0, $0, 4  # print_str
   la   $a0, _nl
   syscall

   # Switch to mysyscall code
   lw   $v0, 0($k0) # pop $v0
   lw   $a0, 4($k0) # pop $a0
   lw   $a1, 8($k0) # pop $a1
   addi $t0, $0, 21
   beq  $t0, $v0, syscall21
   addi $t0, $0, 22
   beq  $t0, $v0, syscall22
   addi $t0, $0, 23
   beq  $t0, $v0, syscall23
   addi $t0, $0, 24
   beq  $t0, $v0, syscall24
   addi $t0, $0, 25
   beq  $t0, $v0, syscall25
   addi $t0, $0, 26
   beq  $t0, $v0, syscall26

isr_ret:
   # in case of mysyscall
   # do not overwrite
   # $v0 (can be mysyscall output)
   lw   $k1, 0($k0)  # pop v0
   addi   $t0, $0,13
   beq  $t0, $k1,dno_v0
   lw   $v0, 0($k0)  # pop v0
dno_v0:
   lw   $a0, 4($k0)  # pop a0
   lw   $a1, 8($k0)  # pop a1
   lw   $t0, 12($k0) # pop t0
   lw   $t1, 16($k0) # pop t1
   lw   $t2, 20($k0) # pop t2

   lw   $t3, 24($k0) # pop t3
   lw   $t4, 28($k0) # pop t4

   # standard exception exit
   mfc0    $k0, $14  # update EPC
   addiu   $k0, $k0, 4
   mtc0    $k0, $14
   mtc0    $0,  $13  # update CAUSE

   mfc0    $k0, $12  # update STATUS
   andi    $k0, 0xfffd
   ori     $k0, 0x0001# Set int. enable bit
   mtc0    $k0, $12
   eret

#####################################
#  KERNEL CODE (DRIVERS)
.kdata
fc1: .word 1190000
fc2: .word 1843200

.ktext
syscall21: #Serial.begin # a0=BR
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x03F4
   # LCR (DLAB=1)
   addi $t1,$0,0xAB #10101011
                    #->LCRval.(DLAB=1)
   #bit6=nobreak,bit5-4=#disp.di 0,bit3=parity
   #bit2=1stopbit,bit1-0=8bitframe
   sb   $t1, 3($t0) # LCR (offset 3)
   #calculate the time constant
   sll  $t2, $a0, 4 # BR*16
   la   $t1, fc2    # Carica &fc2
   lw   $t1,0($t1)  #Carica val di fc2
   div  $t1, $t2    # C=fc1/(BR*16)
   mflo $t1         # Valore DL
   sb   $t1, 0($t0) # va in DLL
                    #(byte -signif)
   srl  $t1, $t1, 8 # Valore DLM
                    # nel byte +sig.
   sb   $t1, 0($t0) # va in DLM (byte +signif.)
   # LCR (DLAB=0)
   addi $t1,$0,0x2B #00101011
                    #->LCRval.(DLAB=0)
   sb   $t1, 3($t0)
   j    isr_ret

syscall22: #Serial.println
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x03F4
   sb   $a0,0($t0)      #write data byte
   j    isr_ret

syscall23: #digitalRead
   addi $t0, $0, 13     #LED
   beq  $a0, $t1,isled1#isled
   addi $t0, $0, 2      #PUSHBUTTON
   beq  $a0, $t1,ispsh1#ispushbutton
   j    iserror1
isled1:
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x5678
   lb   $t2, 0($t0)     #read byte
   andi $t2, $t2, 0x10 #mask bit 4
   srl  $v0, $t2, 4
   j fine1
ispsh1:
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x4321
   lb   $t2, 0($t0)     #read byte
   andi $t2, $t2, 0x80 #mask bit 7
   srl  $v0, $t2, 7
   j    fine1
iserror1:
   #do nothing for now
fine1:
   j    isr_ret

syscall24: #digitalWrite
           #a0=reg.no. a1=value(0 o 1)
   addi $t0, $0, 13     #LED
   beq  $a0, $t1,isled2#isled
   addi $t0, $0, 2      #PUSHBUTTON
   beq  $a0, $t1,ispsh2#ispushbutton
   j iserror2
isled2:
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x5678
   andi $t1, $a1, 1  #mask bit 0 of a1
   srl  $t1, $t1, 4  #select bit4
   lb   $t2, 0($t0)  #read before write
   andi $t2, $t2, 0xEF #clear bit4
   or   $t3, $t2, $t1#set bit4
   sb   $t3, 0($t0)  #store
   j fine2
ispsh2:
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x4321
   andi $t1, $a1, 1  #mask bit 0 of a1
   srl  $t1, $t1, 7  #select bit7
   lb   $t2, 0($t0)  #read before write
   andi $t2, $t2, 0x7F #clear bit7
   or   $t3, $t2, $t1#set bit7
   sb   $t3, 0($t0)  #store
   j fine2
iserror2:
   #do nothing for now
fine2:
   j  isr_ret

syscall25: #pinMode
           # a0=reg.no  a1=value
           #assume that modifies bit 0
           # of each status reg.
   addi $t0, $0, 13     #LED
   beq  $a0, $t1,isled3#isled
   addi $t0, $0, 2      #PUSHBUTTON
   beq  $a0, $t1,ispsh3#ispushbutton
   j iserror3
isled3:
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x5678
   lb   $t2, 0($t0)     #read byte
   andi $t1, $a1, 1     #mask bit0 of a1
   andi $t2, $t2, 0xFE #clear bit0
   or   $t3, $t2, $t1  #set bit0
   sb   $t3, 0($t0)
   j fine3
ispsh3:
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x4321
   lb   $t2, 0($t0)     #read byte
   andi $t1, $a1, 1     #mask bit0 of a1
   andi $t2, $t2, 0xFE #clear bit0
   or   $t3, $t2, $t1  #set bit0
   j fine3
iserror3:
   #do nothing for now
fine3:
   j  isr_ret

syscall26: #delay  # a0=delay
   addi $t0, $0,  0x900
   sll  $t0, $t0, 20
   addi $t0, $t0, 0x0040
   la   $t0, fc1     #1.19 MHz
   lw   $t0, 0($t0)
   mult $a0, $t0     #count=delta*fc
   mflo $t2
   addi $t2, $t2, -1 #count=delta*fc-1
   andi $t3, $t2, 0xFF #LSB
   srl  $t4, $t2, 8    #MSB
   andi $t4, $t4, 0xFF
   addi $t1, $0,  0x30 #bit7-6=counter0,
                       #bit5-4=LSB+MSB(16bit)
                       #bit3-1=mode-0,
                       #bit0=binary
counter
   la   $t0, fc1    #1.19 MHz
   sb   $t1, 3($t0) #write value in CWR
   sb   $t3, 0($t0) #write LSB in CR0
   sb   $t4, 0($t0) #write MSB in CR0
   #start counting
   addi $t1,$0,0xC2 #11000010 read CR0
                    # counter latch cmd
 sw   $a0, 0($t0)   ### SIMULA CONTATORE (init)
 add  $t2, $0, $a0
check:
 addi $t2, $t2, -1  ### SIMULA CONTATORE (dec)
 sw   $t2, 0($t0)    ### SIMULA CONTATORE (upd)
   sb   $t1, 3($t0)    #CR0 counter latch cmd
   lh   $t2, 0($t0)    #CR0-LSB (SIM:lh GIUSTO lb)
   lh   $t3, 0($t0)    #CR0-MSB (SIM:lh GIUSTO:lb)
   bne  $t3, $0, check
   bne  $t2, $0, check
   # count finished CR0==0
   j  isr_ret
```