COMPITO di ARCHITETTURA DEI CALCOLATORI del 30-06-2016 (v1.1) **MATRICOLA**_____

DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI
→ NON USARE FOGLI NON TIMBRATI
→ ANDARE IN BAGNO PRIMA DELL'INIZIO DELLA PROVA
→ NO APPUNTI O FOGLI PERSONALI, NO TELEFONI, SMARTPHONE, ETC

COGNOME_____

NOME_____

SVOLGIMENTO DELLA PROVA:
PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16": es. N.1+2+3+7
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1+2+3+4+6
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo CALCOLATORI es. N.1+2+3+4+5.
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.6+7

NOTA: per l'esercizio 7 dovranno essere consegnati due files: il file del programma VERILOG e il file relativo all'output (screenshot o copy/paste)

1) [16] Trovare il codice assembly MIPS corrispondente del seguente programma (**utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** riportate qua sotto, per riferimento). Inoltre si relizzino sempre in assembly MIPS le funzioni esterne della libreria "arduino", ipotizzando di utlizzare per la comunicazione seriale il chip 16550A mappato ad indirizzo 0x9000'03F4 per generare ritardi il chip 8254 mappato ad indirizzo 0x9000'0040 (default trasmissione: 8 bit dati, parita' dispari di zeri, 1 bit di stop) e per mantenere lo stato del led il bit4 di una porta posta ad indirizzo 0x9000'5678, per mantenere lo stato del pulsante il bit 7 di una porta posta ad indirizzo 0x9000'4321. Notare che le funzioni di tale libreria devono risiedere tutto nello spazio Kernel. Si ricorda inoltre che il 16550A e' temporizzato con una frequenza Fc=1.8432MHz, mentre l'8254 con una frequenza Fc=1.19MHz. (Nota: svolgere l'esercizio solo su carta SENZA l'ausilio del simulatore).

```
#include <arduino.h>
#define INPUT 1
#define OUTPUT 0
#define HIGH 1
#define LOW 0

// Pin 13 has an LED connected
int led = 13;

// digital pin 2 has a pushbutton attached to it.
int pushButton = 2;
int buttonState = 0;

void setup() {
  pinMode(led, OUTPUT);

  // initialize serial communication
  // at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

void loop() {
  digitalWrite(led, HIGH); // turn the LED
  delay(1000);             // wait for a second

  // read the input pin:
  buttonState = digitalRead(pushButton);

  // print out the state of the button:
  Serial.println(buttonState);
  delay(10);               // delay 10ms

  digitalWrite(led, LOW);  // turn the LED off
  delay(1000);             // wait for a second
}
```

2) [8] Si consideri una cache di dimensione 32B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 434, 737, 441, 745, 449, 753, 457, 749, 234, 750, 754, 758, 762, 434, 837, 435, 841, 445, 849, 457. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso alla cache, riportare i tag contenuti in cache al termine, i bit di modifica (se presenti) e la lista dei blocchi (ovvero il loro indirizzo) via via eliminati durante il rimpiazzamento ed inoltre in corrispondenza di quale riferimento il blocco e' eliminato sia nel caso di politica rimpiazzamento LRU che nel caso di politica di rimpiazzamento FIFO.

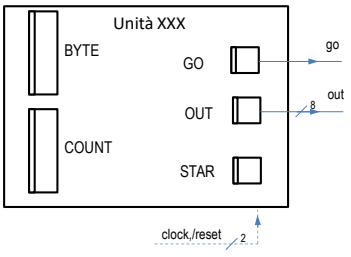
3) [8] Calcolare e confrontare i tempi di esecuzione in spazio Kernel (comprensivi dei tempi di setup dei controller e di gestione della stampa stessa) della stampa di un testo di lunghezza 1024 byte nei tre casi in cui si gestisca l'operazione con la tecnica di: i) polling; ii) interrupt; iii) DMA. Si utilizzino i seguenti tempi: A) per il setup del DMA controller 20 cicli; B) per acknowledge di interrupt 4 cicli; C) per abilitazione di interrupt, per ritorno a User space e per ritorno da interrupt 2 cicli; D) per sbloccaggio utente 15 cicli; E) per passaggio di controllo allo scheduler e per riconoscimento dell'interrupt e lancio della routine di gestione dell'interrupt 3 cicli; F) nell'accesso ai registri di I/O della periferica (status, control, data): 2 cicli per ogni scrittura e 2 cicli per ogni lettura; F) ogni variabile temporanea e allocata in un registro del processore e ogni operazione del processore impiega sempre un ciclo (ALUJ, BRANCH, LOAD/STORE); G) si supponga che letture successive al registro di stato abbiano successo una volta ogni 10 accessi.

4) [4] Spiegare tramite un diagramma architetturale il funzionamento della paginazione inversa per la gestione della memoria virtuale assumendo di avere come ingresso un indirizzo di pagina virtale VPN e come uscita un indirizzo di pagina fisica PPN.

5) [4] Spiegare tramite un diagramma architetturale il funzionamento della paginazione a tre livelli per la gestione della memoria virtuale assumendo di avere come ingresso un indirizzo di pagina virtale VPN e come uscita un indirizzo di pagina fisica PPN.

6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Moore con un ingresso X su tre bit e una uscita Z su tre bit che funziona nel seguente modo: l'uscita rappresenta un numero binario naturale tale che Z=(cx2+cx1+cx0) mod 5 essendo cx2, cx1, cx0 il numero degli 1 logici che sono stati presentati fino all'istante considerato agli ingressi X[2], X[1], X[0] rispettivamente. Rappresentare la macchina a stati finiti per tale rete di Moore, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.

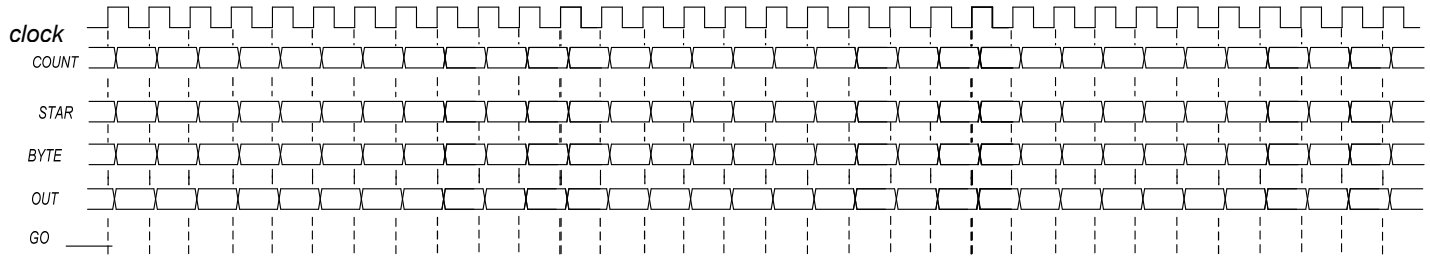
7) [8] Descrivere e sintetizzare l'Unità *XXX* che emette un byte generato in accordo alla legge di cui sotto. Il byte deve permanere all'uscita *out* di *XXX* per un numero di clock esattamente pari a *numero_clock* = *byte* * 3 e deve essere notificato dal fatto che la variabile *go* passa da 0 ad 1 per un ciclo di clock. I *byte* generati soddisfano la doppia condizione di essere numeri *dispari* e *multipli di tre.* **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità *XXX* (il modulo TopLevel e' riportato in calce)



COMPITO di ARCHITETTURA DEI CALCOLATORI del 30-06-2016 (v1.1) **MATRICOLA**_____

COGNOME_____

NOME_____



Instructions

Instruction	Example		Meaning	Comments
add	add/addu	$1,$2,$3	$1 = $2 + $3	(signed/unsigned) 3 operands; exception possible
subtract	sub/subu	$1,$2,$3	$1 = $2 - $3	(signed/unsigned) 3 operands; exception possible
add immediate	addi/addiu	$1,$2,100	$1 = $2 + 100	(signed/unsigned) + constant ; exception possible
multiplication	mult/multu	$1, $2	Hi,Lo= $1 x $2	(signed/unsigned) 64-bit Product ; result in Hi,Lo
division	div/divu	$1, $2	Hi= $1 % $2, Lo = $1 / $2	(signed/unsigned) division
move from Hi	mfhi	$1	$1 = Hi	Create copy of Hi
move from Lo	mflo	$1	$1 = Lo	Create copy of Lo
and	and	$1,$2,$3	$1 = $2 & $3	3 register operands; Logical AND
or	or	$1,$2,$3	$1 = $2 | $3	3 register operands; Logical OR
nor	nor	$1,$2,$3	$1 = !($2 | $3)	3 register operands; Logical NOR
xor	xor	$1,$2,$3	$1 = $2 ^ $3	3 register operands; Logical XOR
and immediate	andi	$1,$2,100	$1 = $2 & 100	Logical AND register, constant
or immediate	ori	$1,$2,100	$1 = $2 | 100	Logical OR register, constant
xor immediate	xori	$1,$2,100	$1 = $2 ^ 100	Logical XOR register, constant
shift left logical	sll	$1,$2,10	$1 = $2 << 10	Shift left by constant
shift right logical	srl	$1,$2,10	$1 = $2 >> 10	Shift right by constant
load word	lw	$1,100($2)	$1 = Memory[$2+100]	Data from memory to register
load byte	lb	$1,100($2)	$1 = Memory[$2+100]	Data from memory to register
load byte unsigned	lbu	$1,100($2)	$1 = Memory[$2+100]	Data from mem. To reg.; no sign extension
store word	sw	$1,100($2)	Memory[$2+100] = $1	Data from register to memory
store byte	sb	$1,100($2)	Memory[$2+100] = $1	Data from register to memory
load address	la	$1,var	$1 = &var	Load variable address
branch unconditional	b	100	go to PC+4+100	PC relative branch
branch on equal	beq	$1,$2,100	if ($1 = = $2) go to PC+4+100	Equal test; PC relative branch
branch on not equal	bne	$1,$2,100	if ($1 != $2) go to PC+4+100	Not equal test; PC relative
set on less than	slt	$1,$2,$3	if ($2 < $3) $1 = 1; else $1 = 0	Compare less than; 2's complement
set on less than immediate	slti	$1,$2,100	if ($2 < 100) $1 = 1; else $1 = 0	Compare < constant; 2's complement
set on less than unsigned	sltu	$1,$2,$3	if ($2 < $3) $1 = 1; else $1 = 0	Compare less than; natural number
set on less than imm.unsigned	sltiu	$1,$2,100	if ($2 < 100) $1 = 1; else $1 = 0	Compare constant; natural number
jump	j	10000	go to 10000	Jump to target address
jump register	jr	$31	go to $31	For switch, procedure return
jump and link	jal	10000	$31 = PC + 4;go to 10000	For procedure call
no operation	nop		Do nothing	Do nothing
load-linked	ll	$1,100($2)	$1=Memory[$2+100]	Read and start to monitor the given memory location
store-conditional	sc	$1,100($2)	Memory[$2+100]=$1 or →	return 0 if a coherence action happens since the previous ll ($1 must be different from 0)
add.s add.d	add.x	$f0,f2,$f4	$f0=$f2+$f4	Single and double precision add
sub.s sub.d	add.x	$f0,f2,$f4	$f0=$f2-$f4	Single and double precision subtraction
mul.s mul.d	mul.x	$f0,f2,$f4	$f0=$f2*$f4	Single and double precision multiplication
div.s div.d	div.x	$f0,f2,$f4	$f0=$f2/$f4	Single and double precision division
mov.s mov.d	mov.x	$f0,$f2	$f0←$f2	Single and double precision move
abs.s abs.d	abs.x	$f0,$f2	$f0=ABS($f2)	Single and double precision absolute value
neg.s neg.d	neg.x	$f0,$f2	$f0= – ($f2)	Single and double precision opposite value
c.lt.s c.lt.d (eq,ne,le,gt,ge)	c.lt.x	$f0,$f2	Temp=($f0<$f2)	Single and double: compare $f0 and $f2 <,=,!=,<=,>,>=
mtc1	mtc1	$1,$f2	$f2=$1	Data from gen.reg. $1 to C1 reg. $f2 (no conversion)
mfc1	mfc1	$f1,$1	$1=$f2	Data from gen.reg. to C1 reg. (no conversion)
branch on false	bc1f	label	If (Temp = = false) go to label	Temp is 'Condition-Code'
branch on true	bc1t	label	If (Temp = = true) go to label	Temp is 'Condition-Code'
load floating point (32bit)	lwc1	$f0,0($1)	$f0←Memory[$1]	Data from FP (C1) register to memory
store floating point (32bit)	swc1	$f0,0($1)	Memory[$1]←$f0	Data from memory to FP (C1) register
convert single into double	cvt.d.s	$f0,$f2	$f0=(double)$f2	Also cvt.s.d (viceversa)
convert single into integer	cvt.w.s	$f1,$f0	$f1=(int)$f0	Also cvt.s.w (viceversa)

Register Usage

Name	Reg. Num.	Usage
$zero	0	The constant value 0
$s0-$s7	16-23	Saved
$t0-$t9	8-15,24-25	Temporaires
$a0-$a3	4-7	Arguments

Name	Reg.Num.	Usage
$v0-$v1	2-3	Results
$fp, $sp	30,29	frame pointer, stack pointer
$ra, $gp	31,28	return address, global pointer
$k0-$k1	26,27	Kernel usage

Reg. Num.	Usage
$f0, $f2	Return values
$f12,$f14	Function arguments
$f20,$f22,$f24,$f26,$f28,$f30	Saved registers
$f4,$f6,$f8,$f10,$f16,$f18	Temporaries registers

System calls

Service Name	Service Num. ($v0)	INPUT Arguments	OUTPUT Arguments
print_int	1	$a0=integer to print	---
print_float	2	$f12=float to print	---
print_double	3	($f12,$f13)=double to print	---
print_string	4	$a0=address of ASCIIZ string to print	---
read_int	5	---	$v0=integer
read_float	6	---	$f0=float
read_double	7	---	$f0-f1=double
read_string	8	$a0=address of input buffer, $a1=max characters to read	---
sbrk	9	$a0=Number of bytes to be allocated	$v0=pointer to the allocated memory
exit	10	---	---

```
module TopLevel;
  reg reset_; initial begin reset_=0;  #22 reset_=1; #300; $stop; end
  reg clock ; initial clock =0; always #5 clock <=(!clock);
  wire[8:0] COUNT=Xxx.COUNT;
  wire[7:0] BYTE=Xxx.BYTE;
  wire STAR=Xxx.STAR;
  wire GO=Xxx.go;
  wire[7:0] OUT=Xxx.out;
 XXX Xxx(go,out, clock,reset_);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 30-06-2016 (v1.1)

SOLUZIONE

ESERCIZIO 1

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.
   jal  Serial.println

   addi $a0, $0, 10    #setta I param.
   jal  delay

   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)
   addi $v0, $0, 25 #syscall 25
   teq  $0,$0      ###SIM:emulo syscall
   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
```

Output:

```
Syscall 25
Syscall 21
Syscall 25
Syscall 24
Syscall 26
Syscall 23
Syscall 22
Syscall 26
Syscall 24
Syscall 26
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 30-06-2016 (v1.1)

SOLUZIONE

ESERCIZIO 3 (REVISIONE 2.1)

STAMPA A POLLING → 8'195+50'177+2=**58'374**

- copy_from_user(buffer, p, count); → * } 8'195
  for (k=0; k<count; ++k) { → 1 ciclo per statment iniziale +2 cicli (SLT+BNE)
  while (*printer_status_reg != READY); → 4(2 per lettura I/O+BNE+J)*10 (tentativi [H])
  *printer_data_register = p[k]; → 3(2 per calcolo offset+1 lettura)+ 2 (scrittura I/O) cicli
  } → 2 cicli (ADDI+J) → 1 + (2+40+5+2)*1024=50'177 cicli
  return_to_user(); → ([C]) 2 cicli

* copy_from_user:

```
(a0=&buffer,a1=&p,a2=count)
      ADD  t1, x0, x0     →1
LOOP: SLT  t0, t1, a2     →1
      BNE  t0, x0,FINE    →1
      ADDI t1, t1, 1      →1
      ADD  t0, a0, t1     →1
      ADD  t1, a1, t1     →1
      LW   t2, 0(t0)      →1
      SW   t2, 0(t1)      →1
      J LOOP              →1
FINE:  → 1024*8+2+1 = 8'195 cicli
```

STAMPA A INTERRUPT → 8'195+49+20'469=**28'713**

- Implementazione dalla system call:
  copy_from_user(buffer, p, count); → * } 8'195
  while (*printer_status_reg != READY); → 4(2 per lettura I/O+BNE+J)*10(tentativi)
  *printer_data_register = p[0]; → 2(scrittura I/O)+1(lettura) cicli
  count = count - 1; → (ADDI) 1 ciclo
  enable_interrupts(); → ([C]) 2 cicli
  scheduler(); → ([E]) 3 cicli } 49
- Implementazione della routine di servizio: → 1023 * (3(ricon [E])+3(lanc.int [E])) + 1022*14 + 1*23=20'469
  if (count == 0) { → (BNE) 1 ciclo
  unblock_user(); → ([D]) 15 cicli
  } else { → (J jump) 1 ciclo
  *printer_data_register = p[k]; → 3(2 per calcolo offset+1 lettura)+ 2 (scrittura I/O) cicli } 14
  count = count - 1; → (ADDI) 1 ciclo
  k = k + 1; → (ADDI) 1 ciclo
  }
  acknowledge_interrupt(); → ([B]) 4cicli
  return_from_interrupt(); → ([C]) 2 cicli } 23

STAMPA A DMA → 8'195+23+3(ricon [E])+3(lanc.int [E])+21=**8'245**

- Implementazione dalla system call:
  copy_from_user(buffer, p, count); → * } 8'195
  setup_DMA_controller(); → ([A]) 20 cicli
  scheduler(); → ([E]) 3 cicli
- Implementazione della routine di servizio:
  acknowledge_interrupt(); → ([B]) 4 cicli
  unblock_user(); → ([D]) 15 cicli
  return_from_interrupt(); → ([C]) 2 cicli

NOTA:
PER LE SOLUZIONI DEI RESTANTI ESERCIZI CHIEDERE DIRETTAMENTE AL DOCENTE