COMPITO di ARCHITETTURA DEI CALCOLATORI del 07-07-2017 **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 (selezionare una delle seguenti 4 opzioni):
☐ PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16 e 16/17": 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) [14] 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).

```
int count = 0;
void solve(int n, int col, int *hist) {
  int i, j;
  char s[2];
  s[1] = '\0';
  if (col == n) {
    print_str("\nNo. "); print_int(++count); print_str("\n-----\n");
    for (i = 0; i < n; i++, print_str("\n"))
      for (j = 0; j < n; j++, print_str(s))
        s[0] = j == hist[i] ? 'Q' : ((i + j) & 1) ? ' ' : '.';
    return;
  } // fineif
  for (i = 0; i < n; i++) { //inizio for
    for (j = 0; j < col && !(hist[j] == i || abs(hist[j] - i) == col - j); j++);
    if (j < col) continue;
    hist[col] = i;
    solve(n, col + 1, hist);
  } //fine for
} //fine solve
```

```
int main() {
  int hist[8];
  solve(8, 0, hist);
}
```

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, la politica di rimpiazzamento e' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 355, 373, 315, 319, 322, 347, 318, 349, 334, 348, 377, 319, 283, 243, 391, 344, 370, 345, 61, 394. 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..

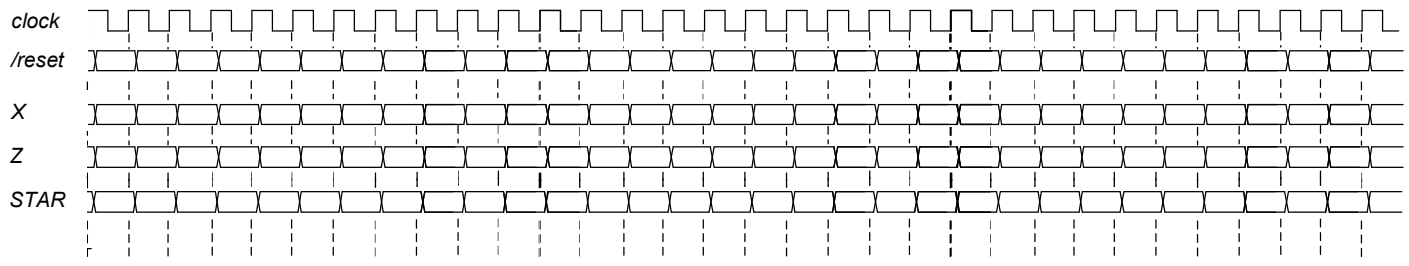
3) [8] Calcolare e confrontare i tempi di esecuzione in spazio Kernel (comprensivi dei tempi di setup dei controller e di gestione della stampa stessa) della stampa di un testo di lunghezza 1024 byte nei tre casi in cui si gestisca l'operazione con la tecnica di: i) polling; ii) interrupt; iii) DMA. 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) [10] Descrivere e sintetizzare in Verilog la rete sequenziale descritta nell'esercizio 6 e il modulo TopLevel. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità. Nota: si puo' svolgere l'esercizio su carta oppure con ausilio del simulatore salvando una copia dell'output (diagramma temporale) e del programma Verilog su USB-drive del docente.



COMPITO di ARCHITETTURA DEI CALCOLATORI del 07-07-2017 **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 / move from Lo	`mfhi/mflo $1`	$1 = Hi ($1 = Lo)	Create copy of Hi (Create a 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 (l=logical,a=arithmetic)	`srl/sra $1,$2,10`	$1 = $2 >> 10	Shift right by constant (in the arithmetic case, the sign is always preserved)
load word / load byte	`lw/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 / store byte	`sw/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/mfc1	`mtc1/mfc1 $1,$f2`	$f2=$1 / $1=$f2	Data from gen.reg. $1 to C1 reg. $f2 (no conversion) / and viceversa
ctc1/cfc1	`ctc1/cfc1 $1,$cf2`	$cf2=$1 / $1=$cf2	Data from gen.reg. to C1 CONTROL reg. (no conversion)) / and viceversa
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	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 07-07-2017 **MATRICOLA**_____

TRACCIA DELLA SOLUZIONE **COGNOME**_____

NOME_____

ESERCIZIO 1

Una possibile soluzione e' la seguente:

```
.data
count: .word 0
hdr: .asciiz "\nNo. "
sep: .asciiz "\n-----\n"
nl1: .asciiz "\n"

.text
.globl main

solve:
        addi $sp,$sp,-32 # allocate frame+loc.var
        sw $a0,0($sp)    # save n
        sw $a1,4($sp)    # save col
        sw $a2,8($sp)    # save *hist
        sw $fp,12($sp)   # save old fp
        sw $ra,16($sp)   # save old ra
                         # 20($sp) is for 's'
        sw $s0,24($sp)   # saved i
        sw $s1,28($sp)   # saved j
        add $fp,$sp,$0
        sb   $0, 21($fp) # s[1]=`\0`
                         # i:s0, j:s1
        bne $a1,$a0,endif#if (col==n)
        la $a0,hdr       # print the header
        addi $v0,$0,4    # print_str
        syscall
        la $t0,count     # &count
        lw $a0,0($t0)            # read count
        addi $a0,$a0,1           #++count
        sw $a0,0($t0)    # store count
        addi $v0,$0,1    # print_str
        syscall
        la $a0,sep       # print the separator
        addi $v0,$0,4    # print_str
        syscall
        lw $a0,0($sp)    #restore a0

        addi $s0,$0,0            # i=0
fori1:
        slt $t9,$s0,$a0
        beq $t9,$0,endfori1
        #----- fori1 body start
        addi $s1,$0,0            # j=0
forj1:
        slt $t9,$s1,$a0
        beq $t9,$0,endforj1
        #----- forj1 body start
        sll $t2,$s0,2    # i*4
        add $t2,$a2,$t2  # &hist+i*4
        lw  $t3,0($t2)   # t3:hist[i]
        bne $t3,$s1,ter2 # i!=j -> ter2
        addi $t4,$0,'Q'
        sb  $t4,20($fp)  #s[0]='Q'
        j   nextforj1
ter2:
        add $t2,$s0,$s1  # i+j
        andi $t3,$t2,1   # (i+j)&1
        bne  $t3,$0,ter3
        addi $t4,$0,' '
        sb  $t4,20($fp)  #s[0]=' '
        j   nextforj1
ter3:
        addi $t4,$0,'.'
        sb  $t4,20($fp)  #s[0]='.'
        #----- forj1 body end
nextforj1:
        addi $s1,$s1,1   # ++j
        addi $a0,$fp,20          # &s
        addi $v0,$0,4    # print_str
        syscall
        lw $a0,0($sp)    #restore a0
        j forj1
endforj1:
        #----- fori1 body end
        addi $s0, $s0, 1 # i++
        la $a0,nl1       # print just 1 newline
        addi $v0,$0,4    # print_str
        syscall
        lw $a0,0($sp)    #restore a0
        j fori1
endfori1:
        j goreturn
endif:
        addi $s0,$0,0            # i=0
fori2:
        slt $t9,$s0,$a0  # i<?n
        beq $t9,$0,endfori2 #ifnot-->endfori2
        #----- fori1 body start
        addi $s1,$0,0            # j=0
forj2:
        slt $t9,$s1,$a1  # j<?col
        beq $t9,$0,endforj2 #ifnot endforj2
        sll $t2,$s1,2    # j*4
        add $t2,$a2,$t2  # &hist+j*4
        lw  $t3,0($t2)   # t3:hist[j]
        beq $t3,$s0,endforj2 #hist[j]==?i,ifyes endforj2
        sub $t4,$t3,$s0  # hist[j]-i
        slt $t5,$t4,$0   # (hist[j]-i)<?0
        beq $t5,$0,posit
        sub $t4,$s0,$t3  # i-hist[j]
posit:
        sub $t5,$a1,$s1  # col-j
        beq $t4,$t5,endforj2
        #----- forj2 body start
        #----- forj2 body end
        addi $s1, $s1, 1 # j++
        j forj2
endforj2:
        slt $t9,$s1,$a1  # j<?col
        bne $t9,$0,gocont # continue
        sll $t2,$a1,2    # col*4
        add $t2,$a2,$t2  # &hist+col*4
        sw  $s0,0($t2)   # hist[col]:i
        # prep. input param.
        addi $a1,$a1,1           # col+1
        jal solve
        lw   $a1,4($fp)  # restore a1 (col)
        #----- fori1 body end
gocont:
        addi $s0, $s0, 1 # i++
        j fori2
endfori2:

goreturn:
        lw $a0,0($sp)
        lw $a1,4($sp)
        lw $a2,8($sp)
        lw $fp,12($sp)
        lw $ra,16($sp)
        lw $s0,24($sp)
        lw $s1,28($sp)
        addi $sp,$sp,32
        j $ra
        #--------
main:
        addi $sp,$sp,-32 # allocate "int hist[8]"
        addi $a0,$0,8            # n
        addi $a1,$0,0            # col
        add  $a2,$sp,$0  # *hist
        jal solve
        addi $sp,$sp,32  # deallocate "int hist[8]"
        addi $v0,$0,10
        syscall
```

ESERCIZIO 2

```
A = 4
B = 8
C = 32
RP = FIFO
Thit = 4
Tpen = 40
File: c1170707.sh_001000.din
Read 20 references.
=== T X XM XT XS XB H [SET]:USAGE [SET]:MODIF [SET]:TAG
=== R 355 44 44 0 3 0 [0]:3,0,0,0 [0]:0,0,0,0 [0]:44,-,-,-
=== W 373 46 46 0 5 0 [0]:2,3,0,0 [0]:0,0,0,0 [0]:44,46,-,-
=== R 315 39 39 0 3 0 [0]:1,2,3,0 [0]:0,0,0,0 [0]:44,46,39,-
=== W 319 39 39 0 7 1 [0]:1,2,3,0 [0]:0,0,1,0 [0]:44,46,39,-
=== R 322 40 40 0 2 0 [0]:0,1,2,3 [0]:0,0,1,0 [0]:44,46,39,40
=== W 347 43 43 0 3 0 [0]:3,0,1,2 [0]:0,0,1,0 [0]:43,46,39,40 (out: XM=44 XT=44 XS=0 )
=== R 318 39 39 0 6 1 [0]:3,0,1,2 [0]:0,0,1,0 [0]:43,46,39,40
=== W 349 43 43 0 5 1 [0]:3,0,1,2 [0]:1,0,1,0 [0]:43,46,39,40
=== R 334 41 41 0 6 0 [0]:2,3,0,1 [0]:1,0,1,0 [0]:43,41,39,40 (out: XM=46 XT=46 XS=0 )
=== W 348 43 43 0 4 1 [0]:2,3,0,1 [0]:1,0,1,0 [0]:43,41,39,40
=== R 377 47 47 0 1 0 [0]:1,2,3,0 [0]:1,0,0,0 [0]:43,41,47,40 (out: XM=39 XT=39 XS=0 )
=== W 319 39 39 0 7 0 [0]:0,1,2,3 [0]:1,0,0,0 [0]:43,41,47,39 (out: XM=40 XT=40 XS=0 )
=== R 283 35 35 0 3 0 [0]:3,0,1,2 [0]:0,0,0,0 [0]:35,41,47,39 (out: XM=43 XT=43 XS=0 )
=== W 243 30 30 0 3 0 [0]:2,3,0,1 [0]:0,0,0,0 [0]:35,30,47,39 (out: XM=41 XT=41 XS=0 )
=== R 391 48 48 0 7 0 [0]:1,2,3,0 [0]:0,0,0,0 [0]:35,30,48,39 (out: XM=47 XT=47 XS=0 )
=== W 344 43 43 0 0 0 [0]:0,1,2,3 [0]:0,0,0,0 [0]:35,30,48,43 (out: XM=39 XT=39 XS=0 )
=== R 370 46 46 0 2 0 [0]:3,0,1,2 [0]:0,0,0,0 [0]:46,30,48,43 (out: XM=35 XT=35 XS=0 )
=== W 345 43 43 0 1 1 [0]:3,0,1,2 [0]:0,0,0,1 [0]:46,30,48,43
=== R 61 7 7 0 5 0 [0]:2,3,0,1 [0]:0,0,0,1 [0]:46,7,48,43 (out: XM=30 XT=30 XS=0 )
=== W 394 49 49 0 2 0 [0]:1,2,3,0 [0]:0,0,0,1 [0]:46,7,49,43 (out: XM=48 XT=48 XS=0 )
---------------------------------------
P1 Nmiss=15 Nhit=5 Nref=20 mrate=0.750000 AMAT=34
```

ESERCIZIO 3

STAMPA A POLLING → 6146+26626+2=**32774**

- copy_from_user(buffer, p, count); → *
  for (k=0; k<count; ++k) { → 3 cicli
    while (*printer_status_reg != READY); → 2(lettura I/O)*10(tentativi)
    *printer_data_register = p[k]; → 2 (scrittura I/O)+2(lettura+calcolo offset) cicli
  } → 1 ciclo → 2 + (1+20+4+1)*1024=26626 cicli
  return_to_user(); → 2 cicli

COMPITO di ARCHITETTURA DEI CALCOLATORI del 07-07-2017 **MATRICOLA**_____

TRACCIA DELLA SOLUZIONE **COGNOME**_____

NOME_____

STAMPA A INTERRUPT→ 6146+29+19460=**25635**

- Implementazione dalla system call:
  copy_from_user(buffer, p, count); → *
  while (*printer_status_reg != READY); → 2(lettura I/O)*10(tentativi)
  *printer_data_register = p[0]; → 2(scrittura I/O)+1(lettura) cicli
  count = count - 1; → 1 ciclo
  enable_interrupts(); → 2 cicli
  scheduler(); → 3 cicli
- Implementazione della routine di servizio: → 1023 * (13+3(ricon)+3(lanc.int)) + 23=19460
  if (count == 0) { → 1 ciclo
  unblock_user(); → 15 cicli
  } else { → 1 ciclo
  *printer_data_register = p[k]; → 2 (scrittura I/O)+2(lettura+calcolo offset) cicli
  count = count - 1; → 1 ciclo
  k = k + 1; → 1 ciclo
  }
  acknowledge_interrupt(); → 4cicli
  return_from_interrupt(); → 2 cicli

STAMPA A DMA → 6146+23+3(ricon)+3(lanc.int)+21=**6175**

- Implementazione dalla system call:
  copy_from_user(buffer, p, count); → *
  setup_DMA_controller(); → 20 cicli
  scheduler(); → 3 cicli
- Implementazione della routine di servizio:
  acknowledge_interrupt(); → 4 cicli
  unblock_user(); → 15 cicli
  return_from_interrupt(); → 2 cicli

* LOOP: SLT $t0, $t1, $s2; BNE $t0,$0,FINE; ADDI $t1, $t1, 1; ADD $s0, $a0, $t1; ADD $s1, $a1, $t1; LW $t2, 0($s0); SW $t2, 0($s1); J LOOP; FINE: → 2+ 1024*6 = 6146 cicli

ESERCIZIO 4

```
module TopLevel;
reg reset_;initial begin reset_=0; #22 reset_=1; #300; $stop; end
reg clock; initial clock=0;always #5 clock <=(!clock);
reg [2:0] X;
wire [2:0] z=Xxx.z;
wire [2:0] STAR=Xxx.STAR;
initial begin X=0;
wait(reset_==1); #5
@(posedge clock); X<=2; @(posedge clock); X<=4; @(posedge clock); X<=2; @(posedge clock); X<=5;
@(posedge clock); X<=4; @(posedge clock); X<=0; @(posedge clock); X<=4; @(posedge clock); X<=0;
@(posedge clock); X<=4; @(posedge clock); X<=1; @(posedge clock); X<=5; @(posedge clock); X<=7;
@(posedge clock); X<=1; @(posedge clock); X<=2; @(posedge clock); X<=3; @(posedge clock); X<=0;
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=0;
$finish;
end
XXX Xxx(X,Z,clock,reset_);
endmodule


module XXX(x,z,clock, reset_);
input clock, reset_;
input [2:0]x;
output [2:0]z;
reg [2:0] STAR;
always@(reset_==0) #1 begin STAR <= 0; end
assign z = STAR;
always @(posedge clock) if (reset_==1) #3
   STAR<=(STAR+x[0]+x[1]+x[2])%5;
endmodule
```