COMPITINO #1 di *CALCOLATORI ELETTRONICI 1* del 27-10-06 **MATRICOLA**_____

COGNOME_____

NOME_____

1a) [18/40] Trovare il codice assembly MIPS corrispondente del seguente programma (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS (riportate in calce, per riferimento).
1b) [18/40] Calcolare il tempo di esecuzione nel caso di un calcolatore con frequenza di clock pari a 4 GHz. Si assuma che il CPI di ciascuna categoria di istruzioni sia: aritmetico-logiche-salti 1, branch 3, load-store 100.
1c) [4/40] Evidenziare (numericamente) il contributo del tempo di esecuzione dovuto agli accessi in memoria dati.

```
typedef struct mylistTAG{
   struct mylistTAG *next;
   int data;
} mylist;

int k, v, m;
mylist *ml;

int readlast(mylist *p, int *d)
{
   int i = 0;
   mylist *p0 = p, *p1 = p;

   while (p1 != NULL) {
      p0 = p1;
      p1 = p1->next;
      ++i;
   }

   if (p0 != NULL) { *d = p0->data; } else { *d = -1; }
   return (i);
}
```

```
mylist *headpush(int d, mylist *p) {
   mylist *n = malloc(sizeof(mylist));
   n->next = p;
   n->data = d;
   return(n);
}

main()
{
   for (k = 1; k <= 100; k+=2) {
      ml = headpush(k, ml);
      m = readlast(ml, &v);
      printf("%d - %d\n", m, v);
   }
}
```

MIPS instructions

Instruction	Example		Meaning	Comments
add	`add`	`$1,$2,$3`	$1 = $2 + $3	3 operands; exception possible
subtract	`sub`	`$1,$2,$3`	$1 = $2 - $3	3 operands; exception possible
add immediate	`addi`	`$1,$2,100`	$1 = $2 + 100	+ constant; exception possible
subtract immediate	`subi`	`$1,$2,100`	$1 = $2 - 100	- constant; exception possible
multiplication	`mult`	`$1, $2`	Hi,Lo= $1 x $2	64-bit Signed Product ; result in Hi,Lo
division	`div`	`$1, $2`	Hi= $1 % $2, Lo = $1 / $2	Signed 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 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

Register Usage

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

Name	Register 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

Name	Usage
$f0, $f1, …, $f31	Single precision floating point registers
$f0, $f2, …, $f30	Double precision floating point registers

System calls

Service Name	Service Num. ($v0)	INPUT Arguments	OUTPUT Arguments
print_int	1	$a0=integer	---
print_string	4	$a0=string pointer	---
Sbrk	9	$a0=Number of bytes to be allocated	$v0=pointer to the allocated memory

Una possibile soluzione per il programma assembly e' (in grigio chiaro sono inserite parti opzionali):

```
.data
k:  .word 0
v:  .word 0
m:  .word 0
ml: .word 0
sep: .asciiz " - "
ret: .asciiz "\n"

.text
.globl main

readlast:
# $v0=i, $t1=p0, $t2=p1
#_____________________________________BB-RL1
        add  $v0, $0, $0  # i = 0
        add  $t0, $0, $a0         # p0 = p
        add  $t1, $0, $a0         # p1 = p
#_____________________________________BB-RL2
w1:     beq  $t1, $0, end_w1
#_____________________________________BB-RL3
        add  $t0, $0, $t1         # p0 = p1
        lw   $t3, 0($t1)  # p1->next
        add  $t1, $0, $t3         # p1 = p1->next
        addi $v0, $v0, 1  # ++i
        j    w1
#_____________________________________BB-RL4
end_w1: beq  $t0, $0, else_if1
#_____________________________________BB-RL5
        lw   $t3, 4($t0)  # p0->data
        j   end_if1
#_____________________________________BB-RL6
else_if1:addi $t3, $0, -1 # -1
#_____________________________________BB-RL7
end_if1: sw    $t3, 0($a1) # *d = ...
        jr    $ra         # restituisce i in $v0

headpush:
# $t0=n, $s6=p, $s7=d
#_____________________________________BB-HP1
        add   $s6, $0, $a0 # salva $a0 in $s6
        add   $s7, $0, $a1 # salva $a1 in $s7
        addi  $v0, $0, 9
        addi  $a0, $0, 8
        syscall           # $v0 = n
        sw    $s7, 0($v0) # n->next = p
        sw    $s6, 4($v0) # n->data = d
        jr  $ra           # restituisce n in $v0

main:
# $s0=&k,  $s1=&ml, $s2=&v, $s3=&m
# $s4=100, $s5=tmp1(k)
#_____________________________________BB-MA1
        la    $s0, k
        la    $s1, ml
        la    $s2, v
        la    $s3, m
        addi  $s5, $0, 1   # tmp1 = 1
        sw      $s5, 0($s0)  # k = 1
        addi  $s4, $0, 100 # $s4 = 100
#_____________________________________BB-MA2
for1:   lw      $s5, 0($s0)  # k
        slt   $t0, $s4, $s5     # 100 <? n
        bne   $t0, $0, end_for1 # se VERO, end_for1
#_____________________________________BB-MA3
        lw    $a0, 0($s0)  # arg0 = k
        lw    $a1, 0($s1)  # arg1 = ml
        jal headpush
        sw    $v0, 0($s1)  # ml = risultato
        lw    $a0, 0($s1)         # arg0 = ml
        add   $a1, $0, $s2 # arg1 = &v
        jal readlast
        sw    $v0, 0($s3)  # m = risultato
        add   $a0, $0, $v0 # m
        addi  $v0, $0, 1
        syscall
        addi  $v0, $0, 4
        la    $a0, sep     # separatore
        syscall
        addi  $v0, $0, 1
        lw    $a0, 0($s2)  # v
        syscall
        addi  $v0, $0, 4   # a capo
        la    $a0, ret
        syscall
        addi  $s5, $s5, 2         # k+=2
        sw    $s5, 0($s0)  # memorizza k
        j   for1
#_____________________________________BB-MA4

end_for1:addi  $v0, $0, 10  #USCITA
        syscall
```

Il partizionamento del programma in basic block e' fatto prendendo sequenze di una o piu' istruzioni consecutive che o terminano con una istruzione di branch condizionale o una jump incondizionata (ma non una jal), oppure terminano subito prima di un'etichetta di salto. L'istruzione syscall esegue 1 ciclo in modalita' utente e il resto del tempo in modalita' kernel, per cui e' assimilata ad una jump. La gestione della printf, per semplicita' poteva essere fatta con una semplice syscall 'print_string'.

Bi	***Ni***	***ALJ***	***B***	***LS***	cBi_{CPU}	$cBi_{CPU,MEM}$	$Cbi_{CPU}= cBi_{CPU}*Ni$	$Cbi_{CPU,MEM}= cBi_{CPU,MEM}*Ni$
BB-RL1	50	3	0	0	3	0	150	0
BB-RL2	1325	0	1	0	3	0	3975	0
BB-RL3	1275	4	0	1	104	100	132600	127500
BB-RL4	50	0	1	0	3	0	150	0
BB-RL5	50	1	0	1	101	100	5050	5000
BB-RL6	0	1	0	0	1	0	0	0
BB-RL7	50	1	0	1	101	100	5050	5000
BB-HP1	50	6	0	2	206	200	10300	10000
BB-MA1	1	10	0	1	110	100	110	100
BB-MA2	51	1	1	1	104	100	5304	5100
BB-MA3	50	18	0	7	718	700	35900	35000
BB-MA4	1	2	0	0	2	0	2	0
Ccpu (in cicli di clock)							**198591**	**187700**

Per il calcolo di Ni si osserva in particolare che:

i) il blocco BB-MA3 che costituische il corpo del ciclo for, viene eseguito 50 volte
ii) il blocco BB-MA2, contenendo anche il confronto finale del for viene eseguito 51 volte
iii) le funzioni readlast e headpush vengono chiamate 50 volte
iv) il blocco BB-RL3 viene eseguto 1 volta alla prima chiamata della readlast, 2 volte alla seconda, …
in totale quindi $\sum_{i=1}^{50} i = 25*51 = 1275$ volte
v) il blocco BB-RL2 viene eseguito sempre una volta in piu' di RL3

Si ha quindi che:

$$T_{CPU} = \frac{C_{CPU}}{f_{CPU}} = \frac{198591}{4 \cdot 10^9} \cong 49.548ms$$

$$T_{CPU,MEM} = \frac{C_{CPU,MEM}}{f_{CPU}} = \frac{187700}{4 \cdot 10^9} = 46.925ms$$

Ovvero circa il 95% del tempo di esecuzione e' speso per accessi in memoria.