COMPITINO #1 di *CALCOLATORI ELETTRONICI 1* del 02-11-07

MATRICOLA___________________

COGNOME_____________________

NOME_____________________

1) [5/40] Ricavare la rappresentazione binaria in formato IEEE-754 singola precisione del numero 0.045 supponendo di effettuare un arrotondamento al piu' vicino valore in virgola mobile rappresentabile nel suddetto formato.

2) [35/40] 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). La funzione exp() riceve un numero di tipo "float" e ne restituisce l'esponenziale (e' una funzione esterna da dichiarare opportunamente, in particolare $v0=exp($a0)...).

```
int num_cond(float T[3][], int n, float *nc)
{
   int r = 0, j=1;
   int k;

   while (j <= n) {
      for (k = 0; k < n; ++k) {
          if (T[j-1][k] != 0) {
             *nc += 1 / exp(-T[j-1][k]);
          } else {
             r = 1;
          }
      }
      ++j;
   }
   return (r);
}
```

```
float f = 0.0;
float A[][] =
{{1.0,2.0,3.0},{4.0,5.0,6.0},{7.0,8.0,9.0}};

main()
{
    int ec;

    ec = num_cond(A, 3, &f);

    printf("esito=");
    printf(ec);
    printf("   n.cond=");
    printf(f);
    printf("\n");
}
```

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
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 absolute 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 $1,$f2`	$f2=$1	Data from gen.reg. to C1 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]	
store floating point (32bit)	`swc1 $f0,0($1)`	Memory[$1]←$f0	
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	Register 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	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 to print	---
print_float	2	$f12=float to print	---
print_string	4	$a0=address of ASCIIZ string to print	---
Sbrk	9	$a0=Number of bytes to be allocated	$v0=pointer to the allocated memory

COMPITINO #1 di *CALCOLATORI ELETTRONICI 1* del 02-11-07 **SOLUZIONE**

1) Normalizzando 0.045 si ottiene: 1.44*2^-5. L' "uno" iniziale non viene rappresentato nel formato IEEE-754. Successivamente si puo' ricavare la rappresentazione binaria di 0.44 con 23 bit moltiplicando per 2 e ricavando la n-esima cifra piu' significativa della mantissa M:

M = 0111 0000 1010 0011 1101 011

(es. 0.44 *2 =0.88 → 0, 0.88*2 =1.76 → 1, 0.76*2 = 1.52 → 1, 0.52*2= 1.04 → 1, 0.04 *2 = 0.08 → 0, ...)

Per l'esponente, ricordando che nel caso di singola precisione il valore della polarizzazione e' 127:
E = -5+127=122 ovvero 01111010

Osservando che le cifre binarie di M si ripetono dalla 21-esima cifra e in particolare la 24-esima cifra e' un 1 si desume che la rappresentazione piu' vicina e' quella superiore. Quindi la rappresentazione cercata e':
0 0111 1010 0111 0000 1010 0011 1101 100

2) Una possibile soluzione e' riportata nel file ncond.s.

```
.data
f:      .float 0.0
A:      .float 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0
ret:    .asciiz "\n"
esi:    .asciiz "esito="
nco:    .asciiz "   n.cond="


.text
.globl main
.extern exp

# $a0=**T
# $a1=n
# $a2=*nc
# $v0=r
# $t0=j
# $t1=k
num_cond:
        addi    $sp, $sp, -24 # spazio per $a0,$v0,$t0,$t1
        sw      $ra, 0($sp)   # salva $ra perche' usa altra f.
        sw      $fp, 4($sp)   # salva $fp perche' usa frame
        add     $fp, $sp, $0

        add     $v0, $0, $0
        addi    $t0, $0, 1    # j = 1

        addi    $t2, $0, 1    # costante f.p. 1
        mtc1    $t2, $f1
        cvt.s.w $f1, $f1
        addi    $t2, $0, 0    # costante f.p. 0
        mtc1    $t2, $f0      # la codifica di 0.0 e' 0...0

while_ini:
        slt     $t2, $a1, $t0 # j>?n
        bne     $t2, $0, while_end # se si while_end

        add     $t1, $0, $0   # k = 0
for_ini:
        slt     $t2, $t1, $a1 # k<?n
        beq     $t2, $0, for_end # se no for_end

        addi    $t3, $t0, -1  # j -1
        add     $t2, $t3, $t3
        add     $t2, $t2, $t3 # (j-1)*3
        add     $t2, $t2, $t1 # (j-1)*3+k
        sll     $t2, $t2, 2   # *4
        add     $t2, $t2, $a0 # &T[j-1][k]
        lwc1    $f2, 0($t2)   # T[j-1][k]

        c.eq.s $f2, $f0      # == 0.0
        bc1t    ramo_else

        sw      $a0, 8($fp)   # salva $a0
        sw      $v0, 12($fp)  # salva $v0
        sw      $t0, 16($fp)  # salva $t0
        sw      $t1, 20($fp)  # salva $t1

        neg.s   $f2, $f2      # cambio segno
        mfc1    $a0, $f2
        jal     exp           # $v0=exp($a0)
        mtc1    $v0, $f2
        lw      $t1, 20($fp)  # salva $t1
        lw      $t0, 16($fp)  # salva $t0
        lw      $v0, 12($fp)  # ripristina
$v0
        lw      $a0, 8($fp)   # ripristina
$a0

        div.s   $f2, $f1, $f2 # 1/sqr(-
T[][])

        lwc1    $f3, 0($a2)   # *nc
        add.s   $f3, $f3, $f2 # +=
        swc1    $f3, 0($a2)   # *nc=
        j       if_end

ramo_else:
        addi    $v0, $0, 1    # r = 1

if_end:
        addi    $t1, $t1, 1   # ++k
        j       for_ini

for_end:
        addi    $t0, $t0, 1   # ++j
        j       while_ini

while_end:
        lw      $ra, 0($fp)
        lw      $fp, 4($fp)
        addi    $sp, $sp, 24
        jr      $ra           # ritorna $v0

main:
        la      $a0, A        # param.1
        addi    $a1, $0, 3    # param.2
        la      $a2, f        # param.3

        jal     num_cond

        add     $s0, $v0, $0  # salva ec

        la      $a0, esi      #print "esi...
        addi    $v0, $0, 4
        syscall
        add     $a0, $s0, $0  # ripristina
ec
        addi    $v0, $0, 1
        syscall
        la      $a0, nco      # print
"nco...
        addi    $v0, $0, 4
        syscall
        la      $a0, f        # print f
        lwc1    $f12, 0($a0)
        addi    $v0, $0, 2
        syscall
        la      $a0, ret     # print ret
        addi    $v0, $0, 4
        syscall
        addi    $v0, $0, 10  # exit
        syscall
```