

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	bolf label	If (Temp == false) go to label	Temp is 'Condition-Code'
branch on true	bolt 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	Name	Register Num.	Usage	Name	Usage
\$zero	0	The constant value 0	\$v0-\$v1	2-3	Results	\$f0, \$f1, ..., \$f31	Single precision floating point registers
\$s0-\$s7	16-23	Saved	\$fp, \$sp	30,29	frame pointer, stack pointer	\$f0, \$f2, ..., \$f30	Double precision floating point registers
\$t0-\$t9	8-15,24-25	Temporaires	\$ra, \$gp	31,28	return address, global pointer		
\$a0-\$a3	4-7	Arguments	\$k0-\$k1	26,27	Kernel usage		

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 ASCII string to print	---
Sbrk	9	\$a0=Number of bytes to be allocated	\$v0=pointer to the allocated memory