COMPITINO #1 DI CALCOLATORI ELETTRONICI 1 del 12-11-02 **MATRICOLA**___________________

(versione riveduta da errori di stampa) **COGNOME**____________________

NOME____________________

1) Si consideri il seguente frammento di codice C e si scriva il corrispondente codice Assembly MIPS. (NOTE: 1) UTILIZZARE SOLTANTO LE ISTRUZIONI RIPORTATE NELLA TABELLA SOTTOSTANTE; 2) PER DICHIARARE LE COSTANTI 2.0 E 0.0005 USARE LA DIRETTIVA .double)

```
main()
{
   double d1, d2, r;
   r = newsqrt(d1) + newsqrt(d2);
}

double newsqrt(double v)
{
   double x1, x0 = v / 2;
   int finito = 0;

   while (! finito) {
      x1 = (x0 + (v / x0)) / 2;
      if ((x1 > x0) && (x1 – x0) < 0.0005) finito = 1;
      else if ((x0 > x1) && (x0 – x1) < 0.0005) finito = 1;
      x0 = x1;
   }
   return (x1);
}
```

2) Descrivere brevemente 3 vantaggi della architettura RISC rispetto a quella CISC.

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
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 <,=,!=,<=,>,>=
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	

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