COMPITINO #1 di CALCOLATORI ELETTRONICI 1 del 22-02-05

MATRICOLA____________________

COGNOME____________________

NOME____________________

1) Trovare il codice assembly MIPS corrispondente al 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).

```
char buff[80] = “Britney Spears\n”;

char to_upper(char c)
{
   if (c >= ‘a’ && c <= ‘z’) c -= 0x20;
   return (c);
}

char *myfun(int n, char *p, char c, float f, double d)
{
   char *r, *s = p;

   while (*s++ != ‘\0’ && n-- > 0) {
      *s = to_upper(*s);
      if (*s == c) r = s;
   }

   if (f * d < 0) {
      f = -f;
      r = myfun(7, r, c, f, d);
   }
   return (r);
}

main()
{
   char *p;

   printf(buff);
   p = myfun(7, buff, ‘E’, 1, -1);
   printf(p);
}
```

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	
convert single into double	`cvt.d.s $f0,$f2`	$f0=(double)$f2	Also cvt.s.d (viceversa)
convert single into integer	`cvt.w.s $t0,$f0`	$t0=(int)$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	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

Si presenta una possible soluzione:

```
.data
buff: .asciiz "Britney Spears\n"
      .space 64
zerof: .float 0.0
punof: .float 1.0
munod: .double -1.0
tempf: .float 0.0

.text
.globl main

to_upper:
      slti  $t0, $a0, 'a'
      bne   $t0, $0, ret_to_upper
      addi  $t1, $0,  'z'
      slt   $t2, $t1, $a0
      bne   $t0, $0, ret_to_upper
      addi  $a0, $a0, -0x20
ret_to_upper:
      add   $v0, $a0, $0
      jr    $ra

myfun:
      addi  $sp, $sp, -32 # 8 per var.locali, 4 per $ra, 4 $fp, 16 arg.
      sw    $fp, 12($sp)
      add   $fp, $sp, $0  # inizializza il nuovo frame pointer
                          # d -->inserito dal chiamante
      sw    $a3, 28($fp)  # f
      sw    $a2, 24($fp)  # c
      sw    $a1, 20($fp)  # p
      sw    $a0, 16($fp)  # n
      sw    $ra,  8($fp)  # ind. ritorno
      sw    $s1,  4($fp)  # $s1 = s
      sw    $s0,  0($fp)  # $s0 = r

      add   $s1, $a1, $0  # s = p
while_start:
      lb    $t0, 0($s1)   # *s
      addi  $s1, $s1, 1   # s++
      beq   $t0, $0, while_end # salta se la prima cond. e' falsa
      add   $t1, $a0, $0  # n
      addi  $a0, $a0, -1  # n--
      slt   $t2, $0, $t1  # 0 <? n
      beq   $t2, $0, while_end # salta se la seconda cond. e' falsa

      sw    $a0, 16($fp)  # save $a0, richiesto da to_upper
      lb    $t0, 0($s1)   # nuovo *s
      add   $a0, $t0, $0  # setup del parametro di input
      jal   to_upper      # chiama to_upper
      lw    $a0, 16($fp)  # ripristina $a0
      sb    $v0, 0($s1)   # memorizza resultato, *s = ...
      lb    $t0, 0($s1)   # nuovo *s
      bne   $t0, $a2, while_start

      add   $s0, $s1, $0  # r = s
      j     while_start


while_end:
      lwc1  $f0, 28($fp)  # carica f in $f0
      lwc1  $f2, 32($fp)  # carica d in ($f2,$f3)
      lwc1  $f3, 36($fp)  # 64 bit...
      cvt.d.s $f4, $f0    # converti f in double
      la    $t0, zerof
```

```
      lwc1  $f30, 0($t0)
      cvt.d.s $f8, $f30   # ($f8,$f9) = 0.0 (double)

      mul.d $f6, $f2, $f4 # (double)f * d
      c.lt.d $f6, $f8     # f*d <? 0
      bc1f  if_end

      sub.s $f12, $f30, $f12 # f = -f
      addi  $a0, $0, 7    # $a0 = 7
      add   $a1, $s0, $0  # $a1 = r
                          # $a2 e' a posto = c
      la    $t0, tempf    # espediente per non usare
      swc1  $f12, 0($t0)  #   la mfc1 $a3, $f12
      lw    $a3, 0($t0)   #   cosi' sistemo $a3

      addi  $sp, $sp, -8
      swc1  $f2, 0($sp)
      swc1  $f3, 4($sp)   # quinto parametro = d

      jal   myfun

      add   $s0, $v0, $0  # r = risultato di myfun

if_end:
      add   $v0, $s0, $0  # parametro restituito

      lw    $a0, 16($fp)  # ripristina a0, a1, a2, a3
      lw    $a1, 20($fp)  # (necessario a causa della chiamata ricorsiva)
      lw    $a2, 24($fp)  #
      lw    $a3, 28($fp)  #
      lw    $ra, 8($fp)   # ripristina $ra, $fp
      lw    $fp, 12($fp)  # (necessario a causa della chiamata ricorsiva)
      addi  $sp, $sp, 40  # ripristina lo stack e butta i parametri locali r e s
      jr    $ra

main:
      la    $a0, buff
      addi  $v0, $0, 4
      syscall

      addi  $a0, $0, 7
      la    $a1, buff
      addi  $a2, $0, 'E'
      la    $t0, punof
      lw    $a3, 0($t0)
      addi  $sp, $sp, -8
      la    $t0, munod
      lw    $t1, 0($t0)
      sw    $t1, 0($sp)
      lw    $t2, 4($t0)
      sw    $t2, 4($sp)

      jal   myfun

      add   $a0, $v0, $0
      addi  $v0, $0, 4
      syscall

      addi  $v0, $0, 10
      syscall
```