COMPITINO #1 di *CALCOLATORI ELETTRONICI 1* del 03-11-08

MATRICOLA____________________

COGNOME______________________

NOME______________________

1) [40/40] 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). Come ipotesi di lavoro si supponga inoltre che NON SIA POSSIBILE UTILIZZARE I REGISTRI $t0,$t1,...,$t9, $s1,$s2,...,$s7,$k0,$k1.

```
int A[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 8}};

int detmin(int i, int j, int o, int o0)
{
   int s0, s1, d0, d1, d2, k, k1, k2, k3, i3, j3, d =
0, i1 = i + 1, i2, j1 = j + 1, j2, o1 = i1 + o;
   i2 = i1 % o0;
   j2 = j1 % o0;

   if (o == 1) d = A[i2][j2];
   else {
      for (k = i1; k < o1; ++k) {
         k1 = k % o0;
         k2 = (k1 + 1) % o0;
         k3 = ((k2 != i) ? k1 : ((k1 + 1) % o0));
         d0 = detmin(k3, j2, o - 1, o0);
         d1 = A[k1][j2]*d0;
         s0 = (k+j2) % 2;
         s1 = s0 ? -1 : 1;
         d2 = s1*d1;
         d += d2;
      }
   }
   return(d);
}
```

```
int det(int o)
{
    int i, s, d = 0;
    for (i = 0; i < o; ++i) {
       s = 1 - (i % 2) * 2;
       d += s*A[i][0]*detmin(i, 0, o - 1, o);
    }
    return (d);
}


main()
{
    int dt = det(3);

    print_string("det(A)=");
    print_int(dt);
    exit();
}
```

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	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
exit	10	---	---

1) Una possibile soluzione:

```
            .data
A:          .word       1, 2, 3, 4, 5, 6, 7, 8, 8
str:        .asciiz "det(A)="

            .text
            .globl main

detmin:
            addi        $sp,$sp,-92
            sw          $ra,72($sp)
            sw          $fp,68($sp)
            add         $fp,$0,$sp

            sw          $a0,76($fp) # salva i
            sw          $a1,80($fp) # salva j
            sw          $a2,84($fp) # salva o
            sw          $a3,88($fp) # salva o0
            sw          $0,20($fp)  # d=0
            lw          $v0,76($fp) # i
            addi        $v0,$v0,1   # i+1
            sw          $v0,16($fp) # i1 = i+1
            lw          $v0,80($fp) # j
            addi        $v0,$v0,1   # j+1
            sw          $v0,8($fp)  # j1 = j+1
            lw          $v1,16($fp) # i1
            lw          $v0,84($fp) # o
            add         $v0,$v1,$v0 # i1+o
            sw          $v0,0($fp)  # o1 = i1+o
            lw          $v1,16($fp) # i1
            lw          $v0,88($fp) # o0
            div         $v1,$v0     # HI = i1 % o0
            mfhi        $v0
            sw          $v0,12($fp) # i2= i1 % o0
            lw          $v1,8($fp)  # j1
            lw          $v0,88($fp) # o0
            div         $v1,$v0     # HI = j1 % o0
            mfhi        $v0
            sw          $v0,4($fp)  # j2= j1 % o0
            lw          $v1,84($fp) # o
            addi        $v0,$0,1    # 1
            bne         $v1,$v0,else # o!=1 --> else
            lw          $v1,12($fp) # i2
            lw          $a0,4($fp)  # j2
            la          $a1,A       # &A
            add         $v0,$0,$v1  # i2
            sll         $v0,$v0,1   # i2*2
            add         $v0,$v0,$v1 # i2*3
            add         $v0,$v0,$a0 # i2*3+j2
            sll         $v0,$v0,2   # *4
            add         $v0,$v0,$a1 # &A[i2][j2]
            lw          $v0,0($v0)  # A[i2][j2]
            sw          $v0,20($fp) # d=
            j           fine_if
else:
            lw          $v0,16($fp) # i1
            sw          $v0,44($fp) # k
            j           ini_for
corpo_for:
            lw          $v1,44($fp) # k
            lw          $v0,88($fp) # o0
            div         $v1,$v0     # HI = k % o0
            mfhi        $v0
            sw          $v0,40($fp) # k1=
            lw          $v0,40($fp)
            addi        $v1,$v0,1   # k1+1
            lw          $v0,88($fp) # o0
            div         $v1,$v0     # HI = (k1+1) % o0
            mfhi        $v0
            sw          $v0,36($fp) # k2=
            lw          $v1,36($fp) # k2
            lw          $v0,76($fp) # i
            bne         $v1,$v0,espr1 # k2 != i --> espr1

            lw          $v0,40($fp) # k1
            addi        $v1,$v0,1
            lw          $v0,88($fp) # o0
            div         $v1,$v0     # HI = (k1+1) % o0
            mfhi        $v0
            sw          $v0,32($fp) # k3=
            j           fine_espr
espr1:
            lw          $v0,40($fp) # k1
            sw          $v0,32($fp) # k3=
fine_espr:
            lw          $v1,32($fp) # k3
            sw          $v1,32($fp) # k3=
            lw          $v0,84($fp) # o
            addi        $v0,$v0,-1  # o-1
            lw          $a0,32($fp) # k3
            lw          $a1,4($fp)  # j2
            add         $a2,$0,$v0  # o-1
            lw          $a3,88($fp) # o0
            jal         detmin
            sw          $v0,56($fp) # d0=
            lw          $v1,40($fp) # k1
            lw          $a0,4($fp)  # j2
            la          $a1,A       # &A
            add         $v0,$0,$v1  # k1
            sll         $v0,$v0,1   # k1*2
            add         $v0,$v0,$v1 # k1*3
            add         $v0,$v0,$a0 # k1*3+j2
            sll         $v0,$v0,2   # *4
            add         $v0,$v0,$a1 # &A[k1][j2]
            lw          $v1,0($v0)  # A[k1][j2]
            lw          $v0,56($fp) # d0
            mult        $v1,$v0     # A[k1][j2]*d0
            mflo        $v0
            sw          $v0,52($fp) # d1=
            lw          $v1,44($fp) # k
            lw          $v0,4($fp)  # j2
            add         $v1,$v1,$v0 # k+j2
            addi        $v0,$0,2    # 2
            div         $v1,$v0     # HI = (k+j2) % 2
            mfhi        $v0
            sw          $v0,64($fp) # s0=
            lw          $v0,64($fp) # s0
            beq         $v0,$0,espr4# s0==0 --> espr4
            addi        $v0,$0,-1
            sw          $v0,60($fp) # s1=-1
            j           espr3
espr4:
            addi        $v0,$0,1
            sw          $v0,60($fp) # s1=1
espr3:
            lw          $v0,60($fp) # s1
            sw          $v0,60($fp) # s1=
            lw          $v1,60($fp) # s1
            lw          $v0,52($fp) # d1
            mult        $v1,$v0     # s1*d1
            mflo        $v0
            sw          $v0,48($fp) # d2
            lw          $v1,20($fp) # d
            lw          $v0,48($fp) # d2
            add         $v0,$v1,$v0 # d+d2
            sw          $v0,20($fp) # d=
            lw          $v0,44($fp) # k
            addi        $v0,$v0,1   # ++k
            sw          $v0,44($fp) # k=
ini_for:
            lw          $v0,44($fp) # k
            lw          $v1,0($fp)  # o1
            slt         $v0,$v0,$v1 # k<?o1
            bne         $v0,$0,corpo_for # SI-->corpo_for
fine_if:
            lw          $v0,20($fp) # restituisce d
            add         $sp,$0,$fp
            lw          $ra,72($sp)
            lw          $fp,68($sp)
            addi        $sp,$sp,92
            j           $ra
det:
            addiu       $sp,$sp,-28
            sw          $ra,20($sp)
            sw          $fp,16($sp)
            sw          $s0,12($sp) # salva s0
            add         $fp,$0,$sp
            sw          $a0,24($fp) # salva o
            sw          $0,8($fp)   # i=0
            sw          $0,0($fp)   # d=0
            j           ini2_for
c2_for:
            lw          $v0,8($fp)  # i
            addi        $v1,$0,2    # 2
            div         $v0,$v1     # HI = i % 2
            mfhi        $v0
            sll         $v1,$v0,1   # *2
            addi        $v0,$0,1    # 1
            sub         $v0,$v0,$v1 # 1-...
            sw          $v0,4($fp)  # s=
            lw          $v0,8($fp)  # i
            la          $a0,A       # &A
            sll         $v1,$v0,2   # i*4
            sll         $v0,$v1,2   # i*16
            sub         $v0,$v0,$v1 # i*12 (offset di [i][0])
            add         $v0,$v0,$a0 # &A[i][0]
            lw          $v1,0($v0)  # A[i][0]
            lw          $v0,4($fp)  # s
            mult        $v1,$v0     # s*A[i][0]
            mflo        $s0         # salva in s0
            lw          $v0,24($fp) # o
            addi        $v0,$v0,-1  # o-1
            lw          $a0,8($fp)  # i
            add         $a1,$0,$0   # 0
            add         $a2,$0,$v0  # o-1
            lw          $a3,24($fp) # o
            jal         detmin
            mult        $s0,$v0     # (detmin)*s*A[i][0]
            mflo        $v1
            lw          $v0,0($fp)  # d
            add         $v0,$v0,$v1 # d+...
            sw          $v0,0($fp)  # d=
            lw          $v0,8($fp)  # i
            addiu       $v0,$v0,1   # ++i
            sw          $v0,8($fp)  # i=
ini2_for:
            lw          $v0,8($fp)   # i
            lw          $v1,24($fp)  # o
            slt         $v0,$v0,$v1  # i<?o
            bne         $v0,$0,c2_for # SI-->c2_for

            lw          $v0,0($fp)  # restituisce d
            add         $sp,$0,$fp
            lw          $ra,20($sp) # ripristina ra
            lw          $fp,16($sp) # ripristina fp
            lw          $s0,12($sp) # ripristina s0
            addi        $sp,$sp,28  # ripristina sp
            j           $ra

main:
            addi        $a0, $0, 3
            jal         det
            add         $s0, $0, $v0 #salva dt

            la          $a0, str
            addi        $v0, $0, 4
            syscall     # print_string

            add         $a0, $0, $s0 #ripristina dt
            addi        $v0, $0, 1
            syscall     # print _int

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