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	---	---