COMPITINO di *CALCOLATORI ELETTRONICI 1* del 11-06-13 **MATRICOLA**_____

COGNOME_____

NOME_____

1) [30/40] Trovare il codice assembly MIPS corrispondente del seguente programma (**utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** riportate qua sotto, per riferimento).

```
/* Finds the index of element having max. absolute value.
** Jack Dongarra, linpack, 3/11/78.
*/
static int idamax(int n, float *dx,int incx)

    {
    Float dmax;
    int i, ix, itemp;

    if (n < 1)
        return(-1);
    if (n ==1 )
        return(0);
    if(incx != 1)
        {

        /* code for increment not equal to 1 */

        ix = 1;
        dmax = fabs((double)dx[0]);
        ix = ix + incx;
        for (i = 1; i < n; i++)
            {
            if(fabs((double)dx[ix]) > dmax)
                {
                itemp = i;
                dmax = fabs((double)dx[ix]);
                }
            ix = ix + incx;
            }
        }
    else
        {

        /* code for increment equal to 1 */

        itemp = 0;
        dmax = fabs((double)dx[0]);
        for (i = 1; i < n; i++)
            if(fabs((double)dx[i]) > dmax)
                {
                itemp = i;
                dmax = fabs((double)dx[i]);
                }
        }
    return (itemp);
    }

int main()
{
    float a[100];
    int l, k, n, lda;

    l = idamax(n-k,&a[lda*k+k],1) + k;
}
```

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	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	$f1,$f0	$f1=(int)$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	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

2) [10/40] Si consideri una cache di dimensione 64B e a 4 vie di tipo write-back. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 7773, 7756, 7777, 7790, 7774, 7789, 7773, 7788, 7779, 5390, 7777, 7772, 7703, 5270, 7760, 7775, 7787, 7771, 7707, 5378, 7712.

Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso alla cache, riportare i tag contenuti in cache al termine e la lista dei blocchi (ovvero il loro indirizzo) via via eliminati durante il rimpiazzamento ed inoltre in corrispondenza di quale riferimento il blocco e' eliminato.