

MODULO RETI LOGICHE:

I SEGUENTI ESERCIZI VALGONO 50% DEL VOTO FINALE (40/80) PER GLI INFORMATICI (ARCHITETTURA 1) E (1 E 2) IL 33% DEL VOTO FINALE (20/60) PER GLI ALTRI (ARCHITETTURA 1A)

**Esercizio 1**

Progettare un contatore modulo 13 incrementante e decrementante.

**Esercizio 2**

Progettare un circuito combinatorio che accetta in ingresso tre numeri interi  $A, B, C$ , di  $n$  bit e produce in uscita il minimo  $m$  tra il massimo  $M_1$  di  $A$  e  $B$ , ed il massimo  $M_2$  di  $B$  e  $C$ ; inoltre se  $M_1 < M_2$  calcola in uscita la differenza  $A - M_1$ , altrimenti calcola la differenza  $B - M_2$ .

MODULO CALCOLATORI ELETTRONICI:

I SEGUENTI ESERCIZI VALGONO 50% DEL VOTO FINALE (40/80) PER ARCHITETTURA 1 E 66% DEL VOTO FINALE (40/60) PER ARCHITETTURA 1A. VALGONO 40/40 PER GLI ALTRI.

- [18] 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). In alternativa, si usi l'assembly x86 anziche' MIPS. Le funzioni non definite sono da considerare funzioni esterne al programma Per la funzione fabs usare l'istruzione floating point corrispondente.

```
int check_matrix(double *lu, double *m, int s) {
    int r,c,k,limit; double s=0.0, rr, l, u;
    for(r=0; r<s; r++){
        for(c=0; c<s; c++){
            rr=0.0;
            if (r-1<c) limit=r-1; else limit=c;
            for(k=0; k<=limit; k++){
                l=lu[r*s+k];
                u=lu[k*s+c];
                rr=rr+l*u;
            }
            if(c>=r){
                l=1.0;
                u=lu[r*s+c];
                rr=rr+l*u;
            }
            r=r-m[r*s+c];
            r=fabs(r);
            if(r<1.0E-3) r=0.0;
            s=s+r;
        }
    }
    return(s<1.0E-3);
}

void decompose_matrix(double *m, int s) {
    int k, r, c; double f;
    for(k=0; k<s-1; k++){
        for(r=k+1; r<s; r++){
            f=m[r*s+k]/m[r*k+k];
            for(c=k+1; c<s; c++){
                m[r*s+c]=m[r*s+c]-f*m[k*s+c];
            }
            m[r*s+k]=f;
        }
    }
}
```

- [8] Si consideri una cache di dimensione 128B e a 2 vie di tipo write-back. La dimensione del blocco e' 32 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: 6781, 6761, 6711, 1191, 6712, 6117, 1181, 1791, 1187, 2198, 6185, 6180, 6111, 2178, 6168, 6783, 6795, 6779, 6715, 6316, 6710. 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.
- [4] Secondo la legge di Amdahl, calcolare lo speedup di un programma in cui il 90% del tempo e' speso nelle due funzioni del punto 1 e nell'ipotesi che la prima funzione possa essere accelerata di un fattore 5 con hardware specifico.
- [6] Programmare il timer 8254 in modo che sull'uscita 1 venga generata una forma d'onda con periodo 10ms in "modo 3" e sull'uscita due una forma d'onda con periodo 20ms in "modo 2".
- [4] Spiegare il funzionamento dello schema del delay slot nella pipeline 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 <,<=,>,>=
mtcl (mfc1)	mtcl	\$1, \$f2	\$f2 = \$1	Data from gen.reg. to C1 reg. (no conversion) (and viceversa)
branch on false	bclf	label	If (Temp == false) go to label	Temp is 'Condition-Code'
branch on true	bclt	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
\$\$0-\$\$7	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_double	3	(\$12,\$13)=double 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