

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 una rete sequenziale con due ingressi x_1 e x_2 ed una uscita z , la quale riconosce la sequenza di ingresso 01,11,10,00. Realizzare la rete sia in forma asincrona che in forma sincrona basata su flip-flop JK.

Esercizio 2

Progettare un contatore modulo 10 e modulo 8, nel quale il modulo viene commutato tra i due richiesti ogni volta che si presenta su un ingresso m un impulso, il quale ha anche l'effetto di azzerare il conteggio corrente, indipendentemente dal segnale da contare x .

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.

- [16] 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.

```
float ddot(int n,double dx[],int incx,double dy[],int incy);
daxpy(int n,double da,double dx[],int incx, double dy[],int incy)
}
for (kb = 0; kb < n; kb++) {
    k = n - (kb + 1);
    b[k] = b[k]/a[lda*k+k];
    t = -b[k];
    daxpy(k,t,&a[lda*k+0],1,&b[0],1);
}
}
main ()
{
    double a[100], b[100];
    int ipvt[10], lda, n;
    dgesl(a,lda,n,ipvt,b,0);
}

float ddot(int n,double dx[],int incx,double dy[],int incy);
daxpy(int n,double da,double dx[],int incx, double dy[],int incy)
}
for (kb = 0; kb < n; kb++) {
    k = n - (kb + 1);
    b[k] = b[k]/a[lda*k+k];
    t = -b[k];
    daxpy(k,t,&a[lda*k+0],1,&b[0],1);
}
}
main ()
{
    double a[100], b[100];
    int ipvt[10], lda, n;
    dgesl(a,lda,n,ipvt,b,0);
}

dgesl(double a[],int lda, int n, int ipvt[], double b[], int job)
{
    double t;
    int k,kb,l,nml;

    if (nml >= 1) {
        for (k = 0; k < nml; k++) {
            l = ipvt[k];
            t = b[l];
            if (l != k){
                b[l] = b[k];
                b[k] = t;
            }
            daxpy(n-(k+1),t,&a[lda*k+k+1],1,&b[k+1],1);
        }
    }
}
```

- [8] Si consideri una cache di dimensione 64B e a 2 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: 7781, 7764, 7785, 7798, 7782, 7797, 7781, 7796, 7787, 5398, 7785, 7780, 7711, 5278, 7768, 7783, 7795, 7779, 7715, 5386, 7720. 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.
- [6] Ricavare la rappresentazione binaria in formato IEEE-754 doppia precisione del numero 2/3 supponendo di effettuare un arrotondamento verso $+\infty$.
- [4] In quali casi lo schema di propagazione della pipeline non permette di risparmiarne alcun ciclo anche se abilitato?
- [6] Spiegare il funzionamento dello schema di paginazione inversa.

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,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	Name	Register Num.	Usage	Name	Usage
\$zero	0	The constant value 0	\$v0-\$v1	2-3	Results	\$f0, \$f1, ..., \$f31	Single precision floating point registers
\$.s0-\$.s7	16-23	Saved	\$.fp, \$.sp	30,29	frame pointer, stack pointer	\$f0, \$f2, ..., \$f30	Double precision floating point registers
\$.t0-\$.t9	8-15,24-25	Temporaries	\$.ra, \$.gp	31,28	return address, global pointer		
\$.a0-\$.a3	4-7	Arguments	\$.k0-\$.k1	26,27	Kernel usage		

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 ASCII string to print	---
Sbrk	9	\$a0=Number of bytes to be allocated	\$v0=pointer to the allocated memory