

## 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**

Date le funzioni:

$$\Sigma_4(4,5,6,7,13)+\Delta_4(10,11,14,15)$$

$$\Sigma_4(0,4,5,13)+\Delta_4(2,6,7)$$

costruire la corrispondente rete nei tre seguenti modi:

- PLA in forma minima;
- utilizzando un solo multiplexer a quattro ingressi, ciascuno di due bit;
- memoria di sola lettura.

**Esercizio 2**

Una rete sequenziale con un ingresso  $x$  ed una uscita  $z$  funziona nel seguente modo. L'uscita normalmente ha valore 1, ma se in ingresso viene riconosciuta la sequenza 1011, l'uscita viene portata a 0 e rimane tale finché entro i tre periodi di clock successivi in ingresso si presenta la coppia 01, oppure se dopo tali tre periodi di clock si presentano in ingresso due 1 consecutivi: in corrispondenza di una di queste due situazioni l'uscita ritorna ad 1 e la rete riprende il funzionamento considerando l'ultimo 1 acquisito come l'inizio di una nuova sequenza da riconoscere. Progettare la rete utilizzando come elementi di memoria flip-flop tutti diversi.

## 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 anziché MIPS. Le funzioni non definite sono da considerare funzioni esterne al programma.

```
FILE1.c:
#define A(i,j) a[(i)*n+(j)]
#define N 5

double x[N*N];
double *ssyrk(double *a, double *r, double alpha, double beta, int n);

void intrandmat(double *a, int n) {
    int i, j, k, v;
    for (i = 0; i < n; ++i) {
        for (j = 0; j < n; ++j) {
            k = i * n + j;
            v = (k * k + 3) % 199;
            A(i,j) = (double)(1 + k + v);
            if ((k % (n+1)) == 0) A(i,j) = 400 + v;
        }
    }
}

int main() {
    double *r, alpha = 7.0, beta = 3.0;
    int i, j, k;
    r = (double*)sbrk(N * N * sizeof(double));
    intrandmat(x, N);
    r = ssyrk(x, r, alpha, beta, N);
}
```

```
FILE2.c:
#define A(i,j) a[(i)*n+(j)]
#define R(i,j) r[(i)*n+(j)]

double *ssyrk(double *a, double *r, double alpha, double beta, int n)
{
    int i, j, p;
    double temp;

    for (j = 0; j < n; ++j) {
        for (i = 0; i < n; ++i) {
            temp = 0.0;
            for (p = 0; p < n; ++p) {
                temp += A(p,i) * A(p,j);
            }
            if (beta == 0.0) {
                R(i,j) = alpha * temp;
            } else {
                R(i,j) = alpha * temp + beta * R(i,j);
            }
        }
    }
    return(r);
}
```

- [7] Si consideri una cache di dimensione 96B e a 3 vie di tipo write-back. La dimensione del blocco è 8 byte, il tempo di accesso alla cache è 4 ns e la penalità in caso di miss è pari a 40 ns, la politica di rimpiazzamento è FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 843, 649, 847, 649, 848, 619, 843, 644, 145, 894, 142, 844, 949, 846, 944, 844, 244, 849, 144, 848, 844. 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 è eliminato.
- [8] Per la funzione ssyrk della domanda 1, calcolare il tempo di esecuzione nell'ipotesi di  $n=5$ , frequenza di clock pari a 1GHz e cicli necessari (processore senza pipeline) per eseguire le istruzioni: aritmetico-logiche-jump  $C_{ALJ}=1$ , per i branch  $C_B=3$ , per le load-store (anche floating point)  $C_{LS}=5$ , per le operazioni floating point  $C_{FP}=2$ ;
- [4] Rappresentare in single precision IEEE-754, il valore  $289/3$  arrotondato al valore più vicino.
- [3] Produrre la symbol table per il FILE1.c e per il FILE2.c del codice proposto nella domanda 1.

## 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 unconditional	b 100	go to PC+4+100	PC relative branch
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	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
\$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_double	3	(\$f12,\$f13)=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