

## 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 PLA costituito interamente da porte NAND per le funzioni:

$$f_1 = \Sigma_4(0, 2, 3, 6, 7, 8, 10, 11)$$

$$f_2 = \Sigma_4(0, 6, 7, 8, 15)$$

## Esercizio 2

Progettare una rete sequenziale che in un flusso continuo di bit in ingresso riconosce la sequenza xxx100lxxx oppure la sequenza xxx001lxxx.

Nel progetto fare uso di flip flop T.

## 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.

```

char daytab[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
};
int m;

int day_of_year(int year, int month, int day)
{
    int i, leap;
    if (year < 1752 || month < 1 || month > 12 || day < 1)
        return -1;
    leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;
    if (day > daytab[leap][month])
        return -1;
    for (i = 1; i < month; i++)
        day += daytab[leap][i];
    return day;
}

main() {
    m=day_of_year(2013, 4, 9);
    print_int(m);
    exit;
}

```

- [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: 5473, 5356, 5377, 5290, 5374, 5389, 5373, 5388, 5379, 5390, 5277, 5272, 5203, 5270, 5260, 5375, 5387, 5371, 5307, 5378, 5312. 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.
- [6] Ricavare la rappresentazione binaria in formato IEEE-754 doppia precisione del numero 4/3 supponendo di effettuare un arrotondamento al piu' vicino valore in virgola mobile rappresentabile nel suddetto formato.
- [4] Illustrare lo schema di arbitraggio in Daisy-Chain supponendo di dover collegare ad un bus comune 4 dispositivi chs utilizzino tale schema (indicare anche i segnali del bus comune).
- [6] Disegnare lo schema dettagliato dell'organizzazione fisica in banchi di una DRAM ad 1Mbit assumendo banchi da 32Kbit raggruppati in 4 blocchi.

**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
\$\$0-\$\$9	8-15,24-25	Temporaires
\$\$0-\$\$3	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