COMPITO di ARCHITETTURA DEI CALCOLATORI 1 del 26-01-2015 **MATRICOLA**_____

COGNOME_____

NOME_____

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

Quattro flip-flop T devono essere interconnessi con opportuna circuiteria logica per realizzare un dispositivo che opera in tre diversi modi selezionabili attraverso due bit di controllo m_1 ed m_2:

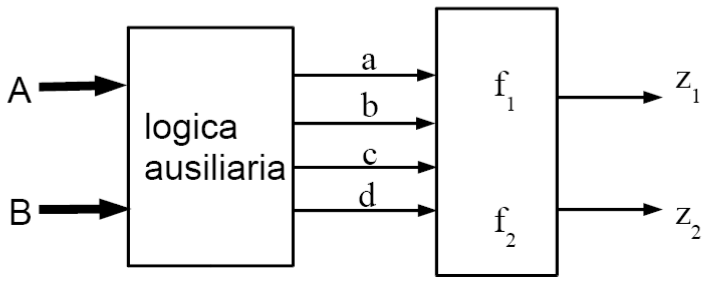
a) registro di rotazione aritmetica sinistra di due bit ($m_1 = 0$, $m_2 = 1$);
b) contatore modulo 31 con passo 3 ($m_1 = 1$, $m_2 = 0$);
c) sequenziatore che percorre ciclicamente la successione 0, 4, 8, 12, 10, 8, 6, 4, 2, 0, … ($m_1 = 1$, $m_2 = 1$).

Per il punto b) si osservi che il generico valore y del conteggio ($0 \le y \le 30$) si può pensare ottenuto dalla relazione $y = 2x+x$ con $0 \le x \le 10$.

Esercizio 2

Dati due numeri relativi A e B in complemento a 2, di n bit ciascuno, definire quattro variabili a, b, c, d, e due funzioni booleane di tali variabili $z_1 = f_1(a,b,c,d)$ e $z_2 = f_2(a,b,c,d)$ tali che se A e B sono entrambi positivi o nulli, $z_1z_2 = 00$; se sono entrambi negativi, $z_1z_2 = 11$; se sono discordi e quello di valore assoluto maggiore è il numero positivo, $z_1z_2 = 10$; se infine i due numeri sono discordi e quello di valore assoluto maggiore è il numero negativo, $z_1z_2 = 10$. Disegnare inoltre la rete combinatoria corrispondente.

Suggerimento: le variabili a, b, c, d riassumono le condizioni in cui si trovano i due numeri (concordi, discordi, ecc.) e saranno quindi prodotte attraverso una opportuna logica ausiliaria, pure da progettare, come schematizzato in figura:



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.

1. [8] Si consideri una cache di dimensione 320B e a 5 vie di tipo write-back. La dimensione del blocco e' 64 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: 127, 113, 163, 121, 140, 161, 115, 1124, 2122, 3141, 4116, 113, 116, 123, 8191, 4116, 2131, 3110, 5111, 118, 131, 121. 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.
2. [4] Rappresentare in double precision IEEE-754, il valore 589/3 arrotondato al valore piu' vicino.
3. [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). In alternativa, si usi l'assembly x86 anziche' MIPS. Le funzioni non definite sono da considere esterne al programma.

FILE1.c:

```
#define N 5
float x[N*N];
int spotf2(int n, float *a, int lda);
#define A(i,j) a[(i)*N+(j)]

void initsimmrandmat(float *a, int n) {
   int i, j, k, v;
   for (i = 0; i < n; ++i) {
      for (j = i; j < n; ++j) {
         k = i * n + j;
         v = (k * k + 3) % 199;
         A(i,j) = (float)(1 + k + v);
         A(j,i) = (float)(1 + k + v);
         if ((k % (n+1)) == 0) A(i,j) = n * 200 + v;
      }
   }
}

int main() {
    float *x;
    int i, j, k;
    x = (float*)sbrk(N * N * sizeof(float));
    initsimmrandmat(x, N);
    spotf2(n, x, N);
}
```

FILE2.c:

```
#define A(i,j) a[(i)*lda+(j)]

int spotf2(int n, float *a, int lda) {
   int i,j;
   float ajj, alpha=-1.0, beta=1.0, gamma;
   for (j = 0; j < n; ++j) {
      ajj = A(j,j) - sdot(j, &A(j, 0), 1, &A(j, 0), 1);
      if (ajj > 0.0f) {
         ajj = sqrt(ajj);
         A(j,j) = ajj;
         if (j + 1 < n) {
            sgemv(j, n - j - 1, &alpha, &A(j + 1, 0),
               lda, &A(j, 0), 1, &beta, &A(j + 1, j), lda);
            gamma = 1.0 / ajj;
            sscal(n - j - 1, &gamma, &A(j + 1, j), lda);
         }
      } else {
         A(j,j)=ajj;
         return (1+j);
      }
      for (i=j+1; i<n; ++i) { A(j,i)=0.0; }
   }
   return 0;
}
```

4. [8] Per la funzione initsimmrandmat della domanda 3, 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;
5. [4] Produrre la symbol table per il FILE1.c e per il FILE2.c del codice proposto nella domanda 3.

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=$1	Data from gen.reg. to C1 reg. (no conversion) (and viceversa)
branch on false	`bc1f`	`label`	If (Temp = = false) go to label	Temp is 'Condition-Code'
branch on true	`bc1t`	`label`	If (Temp = = true) go to label	Temp is 'Condition-Code'
load floating point (32bit)	`lwc1`	`$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	$f12=float to print	---
print_double	3	($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