COMPITO di ARCHITETTURA DEI CALCOLATORI 1 del 03-02-2014 **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 (ARCHTIETTURA 1A)

Esercizio 1
Progettare una rete sequenziale con due ingressi x_1, x_2 ed una uscita z, la quale funziona nel seguente modo. Se $x_2 = 0$ e all'ingresso x_1 si presenta la successione di quattro bit 0101, l'uscita diventa 1; se $x_2 = 1$ e all'ingresso x_1 si presenta la successione di quattro bit 1011, l'uscita diventa 1. In tutti gli altri casi l'uscita è 0. Se x_2 cambia da 0 ad 1 o viceversa, la rete abbandona l'esame della successione corrente e passa ad esaminare l'altra.

Esercizio 2 (solo per Reti Logiche da 6 CFU)
Progettare una ASM con due registri A e B, in grado di eseguire le seguenti operazioni:

OP1: se una variabile binaria x letta in ingresso vale 0, carica in A il risultato dell'espressione [B]^2, dove ^ indica l'innalzamento a potenza; se x vale 1, carica in A il risultato dell'espressione ([A]+[B])/2.

OP2: Trasferisce in A il risultato dell'espressione [B]+d*2, dove d è un intero letto in ingresso.

OP3: Tasferisce in B il contenuto di A e azzera A.

Per il progetto della parte di controllo è ammissibile sia una soluzione hardware, sia una soluzione firmware, a scelta.

Esercizio 3 (solo per Reti Logiche da 3 CFU)
Progettare una rete combinatoria che effettua i calcoli specificati dall'esercizio 2 e che al posto dei registri A e B possiede due ingressi con lo stesso nome e due uscite per i risultati.

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. [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 considere funzioni esterne al programma.

```
double x[4][5];

void gep(float c[SIZE][SIZE+1],int n, int *res)
{
    int i,j,k,p,q,m;
    float temp,x[SIZE],sum,max;

    for(j=0;j<=n-1;j++)
    {
        max = fabs(c[j][j]);
        p=j;
        for(m=j+1;m<=n;m++)
        {
            if(fabs(c[m][j]) >= max)
            {
                max = c[m][j];
                p = m;
            }
        }
        if(p != j)
        {
            for(q=j;q<=n+1;q++)
            {
                temp = c[j][q];
                c[j][q] = c[p][q];
                c[p][q] = temp;
            }
        }
        for(i=j+1;i<=n;i++)
        {
            temp = c[i][j] / c[j][j];
            for(k=j;k<=n;k++)
            {
                c[i][k] = c[i][k] - (temp * c[j][k]);
            }
        }
    }

    x[n-1] = c[n-1][n] / c[n-1][n-1];

    for(i=n-2;i>=0;i--)
    {
        sum = 0;
        for(j=i+1;j<=n;j++)
        {
            sum = sum + (c[i][j] * x[j]);
        }
        x[i] = (c[i][n] - sum)/c[i][i];
    }
}

int main () {
   int out;
   gep(x,4 &out);
}
```

2. [8] Si consideri una cache di dimensione 32B e a 4 vie di tipo write-back. La dimensione del blocco e' 4 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: 4123, 4339, 4327, 4339, 4328, 4139, 4333, 4354, 4325, 4354, 4322, 4354, 4339, 4126, 4354, 4324, 4354, 4329, 4354, 4328, 4354. 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.
3. [4] Qual e' lo scopo della tabella dei simboli nell'assemblatore a due passate? Indicare un esempio di ogni tipo possibile di elementi contenuti in tale tabella.
4. [4] Illustrare il funzionamento di una DRAM da 1 Gbit, disegnando i blocchi architetturali interni e spiegando gli eventi conseguenti all'attivazione dei segnali RAS e CAS.
5. [6] Spiegare il funzionamnto del meccanismo di paginazione inversa nella memoria virtuale, utilizzando un esempio numerico per illustrarne il funzionamento.

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 <,=,!=,<=,>,>=
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	($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