COMPITO di ARCHITETTURA DEI CALCOLATORI 1 del 20-02-2015 **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

Una rete combinatoria con due uscite è stata progettata per le funzioni $z_1 = \sum_4(1,2,3,4,5,6,7,8,12)$ e $z_2 = \sum_4(2,6,8,10,14,15)$. Disegnarne la struttura globalmente ottima a livello di porte NAND. Durante il funzionamento, a causa di un guasto, la rete cessa di comportarsi come previsto, ma calcola le funzioni $z'_1 = \sum_4(1,3,4,5,6,7,8,11,12)$ e $z'_2 = \sum_4(2,6,7,8,10,14)$. Non potendo sostituirla con una uguale ben funzionante, verificare la possibilità di forzarne gli ingressi, in modo da farla rispondere come previsto dal progetto, mediante una ROM di cui devono essere specificate la struttura interna a livello porte e la configurazione binaria delle parole.

Esercizio 2

Progettare la parte di controllo cablata di un sistema PO-PC avente il seguente repertorio di istruzioni.

CMP: verifica se [A] > [B], se [A] < [B], oppure se [A] = [B], dove A e B sono due registri di 16 bit, esaminando in successione coppie di bit di pari posizione, a partire dalla coppia più significativa. Il risultato del confronto è codificato con 10, 01, 11 rispettivamente nei tre casi e scritto in un registro di uscita RIS.

SWP: memorizza nei registri A e B due vettori binari *a* e *b* di 16 bit letti dall'ingresso e scambia la metà meno significativa di A con la metà più significativa di B e viceversa la metà più significativa di A con la metà meno significativa di B.

NOTA: ricordare che la scrittura in parallelo di un registro richiede che ***tutti*** i suoi bit siano scritti simultaneamente, quindi ...

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 768B e a 6 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: 177, 1163, 223, 2181, 200, 3221, 175, 1184, 2182, 3201, 4176, 8173, 2176, 9183, 8251, 4176, 2201, 3180, 5171, 7178, 3191, 181. 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 single precision IEEE-754, il valore 460/7 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.

```
void QuickSort(float *array, int from, int to)
{
   if(from>=to)return;
   int pivot = array[from];
   int i = from, j;
   float temp;
   for(j = from + 1;j <= to;j++) {
      if(array[j] < pivot) {
         i = i + 1;
         temp = array[i];
         array[i] = array[j];
         array[j] = temp;
      }
   }
   temp = array[i];
   array[i] = array[from];
   array[from] = temp;
   QuickSort(array,from,i-1);
   QuickSort(array,i+1,to);
}
```

```
int main()
{
   int number_of_elements;
   read_int(&number_of_elements);
   float array[number_of_elements];
   int iter;
   for(iter = 0;iter < number_of_elements;iter++) {
      read_float(&array[iter]);
   }
   QuickSort(array,0,number_of_elements-1);
   for(iter = 0;iter < number_of_elements;iter++) {
      print_float(array[iter]);
      print_string(" ");
   }
   print_string("\n");
   return 0;
}
```

4. [8] Per la funzione QuickSort della domanda 3, calcolare il tempo di esecuzione nell'ipotesi di 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 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	---
read_int	5	---	$v0=integer
read_float	6	---	$f0=float
read_double	7	---	$f0-f1=double
read_string	8	$a0=address of input buffer, $a1=max characters to read	
sbrk	9	$a0=Number of bytes to be allocated	$v0=pointer to the allocated memory
exit	10	---	---