COMPITO di *ARCHITETTURA DEI CALCOLATORI 1* del 11-09-2012 **MATRICOLA**_____

COGNOME_____

NOME_____

Esercizio1

Si progetti un dispositivo digitale che effettua il confronto tra due vettori binari A e B di otto bit e presenta su due uscite z_1 e z_2 le seguenti configurazioni a seconda degli ingressi. Mediante un bit di controllo m, i due vettori sono interpretati come due numeri naturali ($m = 0$), oppure come due numeri in complemento a due ($m = 1$). Nel primo caso, se $A = B$, $z_1 = 1$, $z_2 = 1$; se $A > B$, $z_1 = 1$, $z_2 = 0$; se $A < B$, $z_1 = 0$, $z_2 = 1$. Nel secondo caso, se $A = B$, $z_1 = 0$, $z_2 = 0$; se $A > B$, $z_1 = 0$, $z_2 = 1$; se $A < B$, $z_1 = 1$, $z_2 = 0$.

Il dispositivo può essere progettato sia come rete combinatoria sia come ASM, a piacere, ma non sono accettate soluzioni banali, per quanto corrette.

Esercizio 2

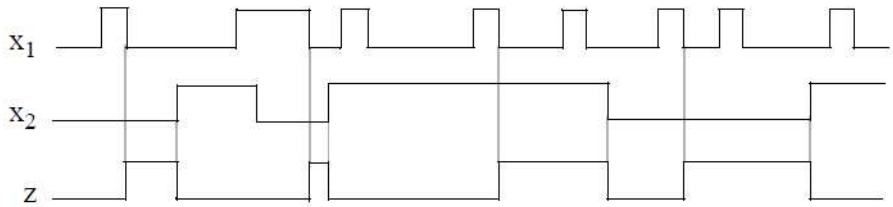
Un circuito sequenziale possiede due ingressi x_1 ed x_2 ed una uscita z ed opera nel seguente modo.

Quando $x_2 = 0$, z rimane 0 finchè si presenta su x_1 un impulso alla fine del quale z diventa 1, mantenendo poi tale valore finchè x_2 diventa 1; allora z ritorna a 0 e vi rimane.

Quando $x_2 = 1$ e all'ingresso x_1 si presenta la sequenza 101, z diventa 1 e mantiene tale valore finchè x_2 diventa 0; allora anche z ritorna a 0 e vi rimane.

In ogni altro caso z rimane inalterata.

Si progetti il circuito utilizzando, se necessario, una memoria di sola lettura per la parte combinatoria .



I precedenti esercizi valgono 50/100 del punteggio totale della prova scritta di Architettura dei Calcolatori 1.

I seguenti esercizi valgono 50/100 della prova scritta di Architettura dei Calcolatori 1, ovvero i 50/50 della prova scritta di Calcolatori Elettronici 1 ovvero Calcolatori Elettronici TLC/GES.

3) [13] Si scriva in assembly (MIPS o x86) il codice della routine di gestione a polling per ricevere dati a 8 bit (parita' pari, 1 bit di stop) sulla porta seriale 16550A. Il carattere deve essere letto supponendo che la porta seriale sia mappata in memoria all'indirizzo 0x02F8 e il dato letto deve essere trasferito in un buffer che si trova a partire dall'indirizzo 0x8001'0000. Tale buffer puo' contenere al massimo 32 bytes.

4) [12] Si consideri una cache di dimensione 512B e a 2 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: 3123, 7456, 4567, 1190, 7264, 2789, 2893, 9088, 2019, 1290, 2227, 3902, 8903, 8890, 3160, 3189, 3197, 3201, 3207, 3208, 3212.
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.

5 [25] (v. pagina successiva)

6) [25] 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.

```
typedef struct {
 int mp
 int dataitem;
 struct listelement *link;
} listelement;

main () {
 listelement listmember, *listpointer;
 int data, choice;
 listpointer = NULL;

 do {
   Menu(&choice);
   switch (choice) {
      case 1:
        print_string("Enter value to add  ");
        read_int(&data);
        listpointer =
           AddItem(listpointer, data);
        break;
      case 2:
        if (listpointer == NULL)
           print_string("Queue empty!\n");
        else
           listpointer =
             RemoveItem (listpointer);
        break;
      case 3:
           PrintQueue (listpointer); break;
      case 4:
        break;
      default:
           print_string("Invalid choice - try again\n");
        break;
    }
  } while (choice != 4);
  ClearQueue (listpointer);
  exit(0);
}

void lock (int *m) {
   while (*m == 1);
   --*m;
}

void unlock (int *m) {
   ++*m;
}

listelement *RemoveItem (listelement * listpointer)
{

    listelement *tempp;

    lock(&listpointer->mp);
    print_string("Element removed is");
    print_int(listpointer -> dataitem);
    tempp = listpointer -> link;
    unlock(&listpointer->mp);
    return (tempp);

}
```

MIPS 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 memory 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

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_string	4	$a0=address of ASCIIZ string to print	---
read_int	5	---	$v0=read integer
sbrk	9	$a0=Number of bytes to be allocated	$v0=pointer to the allocated memory
exit	10	---	---