COMPITO di ARCHITETTURA DEI CALCOLATORI 1 del 23-06-2014 **MATRICOLA**_____

COGNOME_____

NOME_____

MODULO RETI LOGICHE:

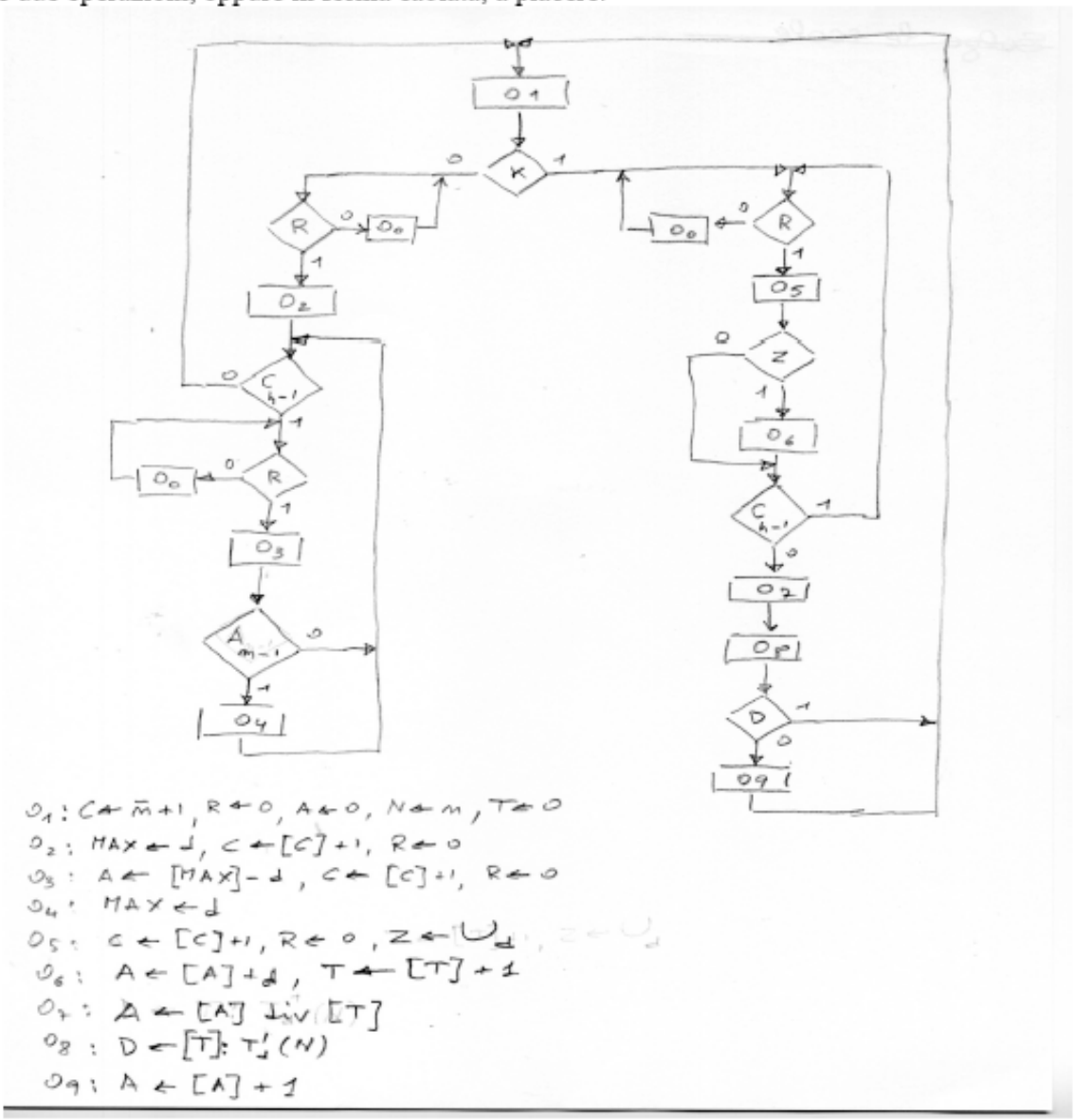
I SEGUENTI ESERCIZI VALGONO 50% DEL VOTO FINALE (40/80) PER GLI INFORMATICI (ARCHITETTURA 1) E (1 E 2) IL 33% DEL VOTO FINALE (20/60) PER GLI ALTRI (ARCHTIETTURA 1A)

Esercizio 1

Una rete sequenziale possiede due ingressi, x_1 e x_2, ed una uscita z e funziona nel seguente modo. L'uscita normalmente ha valore 0, ma quando su x_1 si succedono due impulsi di durata qualsiasi (purché sufficiente per la stabilizzazione della rete) mentre x_2 è 0 e successivamente un impulso si verifica su x_2 mentre x_1 è 0, l'uscita diventa 1 al termine dell'impulso su x_2 e rimane tale fino alla prima transizione da 0 ad 1 su x_1, mentre x_2 è 0, allorché ritorna a zero. Progettare la rete.

Esercizio 2

Dato il sistema descritto dal seguente diagramma di flusso progettarne la PO e la PC in forma microprogrammata in linguaggio *ts* con separazione del codice di operazione dalle microroutine delle due operazioni, oppure in forma cablata, a piacere.



Nota: *d* è un dato di ingresso al sistema, R è un flag che informa il sistema che un nuovo dato è pronto. L'operatore ":" indica il confronto, l'operatore "div" la divisione intera e l'operatore "U" l'or bit a bit del dato specificato come pedice.

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. sqrt e' una funzione di una libreria esterna.

```
typedef unsigned char us8;
typedef struct {
  us8 exit;
  char side;
} QueueType;

#define QueueSize 20
QueueType QueueTypeInit = { 255, 'C' };

typedef enum {
  qeNONE = 0,
  qeOUT_OF_RANGE = 1
} QueueError;

QueueType *table[QueueSize];
us8 head;
us8 tail;
void clear();

Queue() {
    clear();
}

void index_rollover(us8 index) {
    index++;
    if(index >= QueueSize) {
      index = 0;
    }
}

us8 size() {
    return QueueSize;
}

us8 count() {
    if( head >= tail ) {
      return (head - tail);
    }
    else {
      return (head + QueueSize - tail);
    }
}
```

```
void remove(us8 index) {
    if(index < QueueSize) {
        table[index] = &QueueTypeInit;
    }
}

void clear() {
    us8 i;
    for(i = 0; i < QueueSize; i++) {
      remove(i);
    }
    head = 0;
    tail = 0;
}

QueueType *at(us8 index) {
    QueueType *rval = &QueueTypeInit;
    if(index < QueueSize) {
      return table[index];
    }
    return rval;
}

QueueError push(QueueType *position) {
    if( count() >= (QueueSize-1) ) {
      return qeOUT_OF_RANGE;
   }

    table[head] = position;
    index_rollover(head);
    return qeNONE;
}

QueueType pop() {
    QueueType *temp = &QueueTypeInit;
    temp = at(tail);
    remove(tail);
    index_rollover(tail);
    return *temp;
}
```

2. [7] Si consideri una cache di dimensione 64B e a 4 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: 4423, 4356, 4357, 4290, 4364, 4389, 4393, 4388, 4319, 4390, 4227, 4202, 4203, 4290, 4260, 4385, 4387, 4301, 4307, 4308, 4312. 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. [5] Spiegare il funzionamento della paginazione inversa, facendo riferimento ad un diagramma architetturale dettagliato e ad un esempio numerico.
4. [4] Spiegare il significato del "machine epsilon" e fornirne il valore nel caso di IEEE-754 doppia precisione.
5. [6] Descrivere in formalismo C-like o Assembly MIPS come avviene l'operazione di ingresso di un pacchetto dati da rete in modalita' DMA.

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	$f12=float to print	---
print_double	3	($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