COMPITO di ARCHITETTURA DEI CALCOLATORI del 15-09-2016

MATRICOLA_____

COGNOME_____

NOME_____

DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI
→ NON USARE FOGLI NON TIMBRATI
→ ANDARE IN BAGNO PRIMA DELL'INIZIO DELLA PROVA
→ NO APPUNTI O FOGLI PERSONALI, NO TELEFONI, SMARTPHONE, ETC

SVOLGIMENTO DELLA PROVA:
PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16": es. N.1+2+3+7
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1+2+3+4+6
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo CALCOLATORI es. N.1+2+3+4+5.
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.6+7

NOTA: per l'esercizio 7 dovranno essere consegnati due files: il file del programma VERILOG e il file relativo all'output (screenshot o copy/paste)

1) [14] Trovare il codice assembly MIPS corrispondente del seguente programma (**utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri e l'ABI dell'assembly MIPS** riportate qua sotto, per riferimento). Nota: svolgere direttamente su carta (senza il simulatore).

```
struct Edge {
    int vx;
    struct Edge *next;
};

struct Edge *AddEdge(struct Edge *curHead, int newVx)
{
    struct Edge *newHead = malloc(sizeof(struct Edge));
    newHead->vx = newVx;
    newHead->next = curHead;
    return newHead;
}

void BFSearch(struct Edge *adjList[], int vertices,
              int parent[], int level[], int startVx)
{
    struct Edge *tv;
    int i, par, lev=0, flag = 1;
    level[startVx] = lev;

    while (flag) {
        flag = 0;
        for (i = 1; i <= vertices; ++i) {
            if (level[i] == lev) {
                flag = 1;
                tv = adjList[i];
                par = i;

                while (tv != NULL) {
                    if (level[tv->vx] != -1) {
                        tv = tv->next;
                        continue; /*va a fine while*/
                    }
                    level[tv->vx] = lev + 1;
                    parent[tv->vx] = par;
                    tv = tv->next;
                }
            }
        }
        ++lev;
    }
}
```

2) [4] Per la funzione BFSearch in linguaggio C indicata al precedente esercizio, immaginando che il puntatore adjList valga 0x1000, che il puntatore level valga 0x2000, che il puntatore parent valga 0x3000, che le variabili local (allocate nel frame) tv, i, par, lev, flag risiedano solo in memoria a partire dall'indirizzo 0x7000 (verso indirizzi alti), che il vettore adjList conenga i valori [0]:{1,0x5000}, [1]:{2,0x5100}, [2]:{3,0x5200}, ... e cosi' via e che le condizioni degli if, for e while siano sempre vere: produrre la lista degli indirizzi dei primi 20 riferimenti alla memoria dati (startVx si puo' assumere pari a zero e la memoria si puo' assumere tutta inizializzata a zero).

3) [6] Si consideri una cache di tipo write-back/write-non-allocate fully associative, con 8 elementi. La dimensione del blocco e' 8 byte e il tempo di accesso e' 1 ns mentre la penalita' in caso di miss e' pari a 99 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache (primo livello), ad indirizzi al byte: 434, 737, 441, 745, 449, 753, 457, 749, 234, 750, 754, 758, 762, 434, 837, 435, 841, 445, 849, 457. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso finale, riportare per ciascun livello di cache al termine: i tag della configurazione finale, i bit di modifica (se presenti) 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.

4) [4] Rappresentare in double precision IEEE-754, il valore epsilon di macchina (nota bene: 64 bit).

5) [4] Discutere il grafico della legge di Amdhal al variare di 'p' parte da accelerare nell'intervallo [0,1] e con 'a' pari a 1 e 10 rispettivamente. Mostrare poi cosa accade al variare di a nell'intervallo [1,10] e per p fisso pari a 0.5.

6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Mealy con un ingresso X su un bit e una uscita Z su due bit che funziona nel seguente modo: devono essere riconosciute le sequenze anche interallacciate 0,1,0,0 e 0,1,0,1; l'uscita Z[1] va a 1 se si presente una delle due sequenze mentre Z[0] dice quale sequenza si e' presentata (Z[0]=1 se si presenta 0,1,0,0; Z[0]=0 altrimenti. Rappresentare la macchina a stati finiti per tale rete di Mealy, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.

7) [8] Descrivere e sintetizzare in Verilog la rete sequenziale descritta nel'esercizio 6. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità *XXX* (il modulo TopLevel e' riportato in calce). Nota: svolgere l'esercizio con ausilio del simulatore e salvare una copia dell'output e del programma nella cartella ARCAL160915.

COMPITO di ARCHITETTURA DEI CALCOLATORI del 15-09-2016 **MATRICOLA**_____

COGNOME_____

NOME_____

Instructions

Instruction	Example	Meaning	Comments
add	`add/addu $1,$2,$3`	$1 = $2 + $3	(signed/unsigned) 3 operands; exception possible
subtract	`sub/subu $1,$2,$3`	$1 = $2 - $3	(signed/unsigned) 3 operands; exception possible
add immediate	`addi/addiu $1,$2,100`	$1 = $2 + 100	(signed/unsigned) + constant ; exception possible
multiplication	`mult/multu $1, $2`	Hi,Lo= $1 x $2	(signed/unsigned) 64-bit Product ; result in Hi,Lo
division	`div/divu $1, $2`	Hi= $1 % $2, Lo = $1 / $2	(signed/unsigned) 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
no operation	`nop`	Do nothing	Do nothing
load-linked	`ll $1,100($2)`	$1=Memory[$2+100]	Read and start to monitor the given memory location
store-conditional	`sc $1,100($2)`	Memory[$2+100]=$1 or →	return 0 if a coherence action happens since the previous ll ($1 must be different from 0)
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 opposite 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	`mtc1 $1,$f2`	$f2=$1	Data from gen.reg. $1 to C1 reg. $f2 (no conversion)
mfc1	`mfc1 $f1,$1`	$1=$f2	Data from gen.reg. to C1 reg. (no conversion)
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]	Data from FP (C1) register to memory
store floating point (32bit)	`swc1 $f0,0($1)`	Memory[$1]←$f0	Data from memory to FP (C1) register
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	Reg. 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	Reg.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

Reg. Num.	Usage
$f0, $f2	Return values
$f12,$f14	Function arguments
$f20,$f22,$f24,$f26,$f28,$f30	Saved registers
$f4,$f6,$f8,$f10,$f16,$f18	Temporaries 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	---	---

```
module TopLevel;
  reg reset_; initial begin reset_=0;  #22 reset_=1; #300; $stop; end
  reg clock ; initial clock =0; always #5 clock <=(!clock);
  wire X=Xxx.X;
  wire STAR=Xxx.STAR;
  wire Z=Xxx.Z;
  XXX Xxx(x,z, clock,reset_);
endmodule
```