COMPITO di ARCHITETTURA DEI CALCOLATORI del 15-07-2016 (*)

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) [16] 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 dell'assembly MIPS** riportate qua sotto, per riferimento). Nota: svolgere l'esercizio con ausilio del simulatore e salvare una copia dell'output e del programma nella cartella ARCAL160715.

```
FILE1.c::
double arr[20] = 20, 19, 18, 17, 16, 15,
14, 13, 12, 11, 10, 9, 8, 7,6,5,4,3,2,1};
int merge(double arr[],int l,int m,int h){
  double arr1[11],arr2[11];
  int n1,n2,i,j,k;
  n1=m-l+1; n2=h-m;
  for(i=0; i<n1; i++) arr1[i]=arr[l+i];
  for(j=0; j<n2; j++) arr2[j]=arr[m+j+1];
  arr1[i]=9999; arr2[j]=9999;
  i=0; j=0;
  for(k=l; k<=h; k++) {
    if(arr1[i]<=arr2[j])
      arr[k]=arr1[i++];
    else
      arr[k]=arr2[j++];
  }

  return 0;
}
```

```
FILE2.c::
int merge(double arr[],int l,int m,int h);
extern double arr[20];
int merge_sort(double arr[],int low,int
high)
{
  int mid;
  if(low<high) {
    mid=(low+high)/2;
    merge_sort(arr,low,mid);
    merge_sort(arr,mid+1,high);
    merge(arr,low,mid,high);
  }
  return 0;
}
```

```
int main()
{
  int n=20,i;

  merge_sort(arr,0,n-1);

  print_string("Sorted array:");
  for(i=0; i<n; i++)
    print_double(arr[i]);

  exit(0);
}
```

2) [8] Si consideri una cache a 2 livelli in cui il primo livello ha dimensione 64B e a 2 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 16 byte sia per il primo che per il secondo livello, il tempo di accesso alla cache e' 1 ns e la penalita' in caso di miss (per accedere al livello 2) e' pari a 5 ns, la politica di rimpiazzamento e' LRU. Il secondo livello ha dimensione 64B e a 2 vie, ancora di tipo write-back/write-non-allocate e la penalita' in caso di miss e' 20. Il processore effettua i seguenti accessi in cache (primo livello), ad indirizzi al byte: 611 91 61 191 113 141 17 113 171 190 10 12 10 15 91 71 61 251 131 61 71 21. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso finale per ciascun livello e quello effettivo visto dal processore, 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.

3) [8] Calcolare e confrontare i tempi di esecuzione in spazio Kernel (comprensivi dei tempi di setup dei controller e di gestione della stampa stessa) della stampa di un testo di lunghezza 1024 byte nei tre casi in cui si gestisca l'operazione con la tecnica di: i) polling; ii) interrupt; iii) DMA. Si utilizzino i seguenti tempi: A) per il setup del DMA controller 20 cicli; B) per acknowledge di interrupt 4 cicli; C) per abilitazione di interrupt, per ritorno a User space e per ritorno da interrupt 2 cicli; D) per sbloccaggio utente 15 cicli; E) per passaggio di controllo allo scheduler e per riconoscimento dell'interrupt e lancio della routine di gestione dell'interrupt 3 cicli; F) nell'accesso ai registri di I/O della periferica (status, control, data): 2 cicli per ogni scrittura e 2 cicli per ogni lettura; F) ogni variabile temporanea e allocata in un registro del processore e ogni operazione del processore impiega sempre un ciclo (ALUJ, BRANCH, LOAD/STORE); G) si supponga che letture successive al registro di stato abbiano successo una volta ogni 30 accessi.

4) [4] Spiegare tramite un diagramma temporale (preciso) e la specifica di tutti i segnali necessari come funziona l'operazione di lettura dell'handshake asincrono.

5) [4] Discutere, tramite un esempio ciascuno i concetti di endianess dei byte a allineamento degli indirizzi.

6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Moore 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 squenze 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 Moore, 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 ARCAL160715.

(*) versione riveduta da alcuni errori tipografici

COMPITO di ARCHITETTURA DEI CALCOLATORI del 15-07-2016 (*) **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
```

(*) versione riveduta da alcuni errori tipografici