COMPITO di ARCHITETTURA DEI CALCOLATORI del 15-02-2017 **MATRICOLA**_____

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

COGNOME_____

NOME_____

SVOLGIMENTO DELLA PROVA (selezionare una delle seguenti 4 opzioni):
☐ PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16 e 16/17": 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.

1) [12] Scrivere in assembly MIPS l'implementazione della funzione char *atox(char *s) che trasforma la stringa puntata dal parametro di ingresso s – contenente i caratteri ASCII di un numero naturale (ovvero un intero >=0) in una stringa, il cui puntatore e' restituito come parametro di ritorno che rappresenta in esadecimale lo stesso numero preceduto da "0x" (es. la stringa "123" dovra' diventare "0x7B"). La funzione dovra' anche provvedere all'allocazione della memoria dinamica per memorizzare la stringa di uscita, inoltre dovra' essere realizzato un semplice programma main che legga da tastiera la stringa di ingresso e stampi la stringa di uscita chiamando la funzione "atox" (promemoria: i caratteri ASCII delle cifre sono "0"->0x30, "1"-> 0x31,... "A"->0x41, "B"->0x42, ... "x"->0x78).

2) [9] Si consideri una cache a 2 livelli in cui il primo livello ha dimensione 128B, fully-associative, 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' 2 ns e la penalita' in caso di miss (per accedere al livello 2) e' pari a 8 ns, la politica di rimpiazzamento e' LRU. Il secondo livello ha dimensione 128B, fully-associative, ancora di tipo write-back/write-non-allocate e la penalita' in caso di miss e' 90 ns. Il processore effettua i seguenti accessi in cache (primo livello), ad indirizzi al byte: 10 12 10 15 91 71 61 251 131 611 91 61 191 113 141 17 113 171 190 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) [7] Dato il seguente frammento di codice, in esecuzione sulla classica pipeline MIPS a 5-stadi, con 1 delay-slot, accesso sovrapposto ai registri nelle fasi di decodifica/write-back, propagazione abilitata, valori iniziali dei registri $8=4, $9=40, determinare quale istruzione e' in esecuzione in ciascuno degli stadi F,D,X,M,W al ciclo numero 18 (il primo ciclo parte da 1):

```
Loop:  sub $9,   $9, $8
       add $10, $16, $9
       lw  $11, -4($10)
       add $17, $17, $11
       bne $9,   $0, Loop
```

4) [4] Scrivere la codifica in linguaggio macchina (parole da 32 bit) del programma in assembly MIPS indicato all'esercizio 3 (si ricorda che per le istruzioni add,sub,lw,bne i campi opcode/funct valgono rispettivamente: 0/0x20,0/0x22,0x23/-,0x5/-).

5) [4] Spiegare il funzionamento della paginazione a tre livelli con un diagramma ed un esempio numerico nel caso di spazio di indirizzamento virtuale a 64 bit, spazio di indirizzamento fisico a 40 bit, pagine di 4K e 8 bit per l'offset di ciascuno dei tre livelli.

6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Mealy con un ingresso X su un bit e una uscita Z su un bit che funziona nel seguente modo: devono essere riconosciute le sequenze interallacciate del tipo 1,1,0,1; l'uscita Z va a 1 se è presente tale sequenza. Rappresentare per tale macchina a stati finiti, 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 nell'esercizio 6. Tracciare il diagramma di temporizzazione come verifica della correttezza dell'unità XXX (il modulo TopLevel e' riportato in calce). Nota: si puo' svolgere l'esercizio su carta oppure con ausilio del simulatore salvando una copia dell'output (diagramma temporale) e del programma Verilog su USB-drive del docente.

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 / move from Lo	mfhi/mflo	$1	$1 = Hi ($1 = Lo)	Create copy of Hi (Create a 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 (l=logical,a=arithmetic)	srl/sra	$1,$2,10	$1 = $2 >> 10	Shift right by constant (in the arithmetic case, the sign is always preserved)
load word / load byte	lw/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 / store byte	sw/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

COMPITO di ARCHITETTURA DEI CALCOLATORI del 15-02-2017 **MATRICOLA**_____

COGNOME_____

NOME_____

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/mfc1	`mtc1/mfc1 $1,$f2`		$f2=$1 / $1=$f2	Data from gen.reg. $1 to C1 reg. $f2 (no conversion) / and viceversa
ctc1/cfc1	`ctc1/cfc1 $1,$cf2`		$cf2=$1 / $1=$cf2	Data from gen.reg. to C1 CONTROL 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]	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);
reg X;
wire [1:0] Z;
wire [2:0] STAR=Xxx.STAR;
wire Z1=Xxx.z[1];
wire Z0=Xxx.z[0];
initial begin X=0;
wait(reset_==1); #5
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=1;
@(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=1; @(posedge clock); X<=0;
@(posedge clock); X<=1; @(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=1;
@(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=1; @(posedge clock); X<=0;
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=0;
$finish;
end
XXX Xxx(X,Z,clock,reset_);
endmodule
```