

DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI  
 → NON USARE FOGLI NON TIMBRATI  
 → ANDARE IN BAGNO PRIMA DELL'INIZIO DELLA PROVA

**SVOLGIMENTO DELLA PROVA:**

PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16": es. N.1+2+3+5

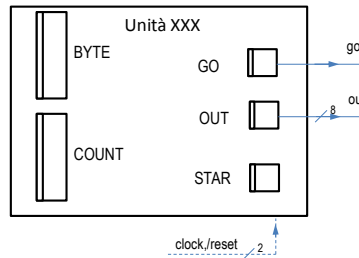
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1+2+3+4+6

PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo CALCOLATORI es. N.1+2+3+4.

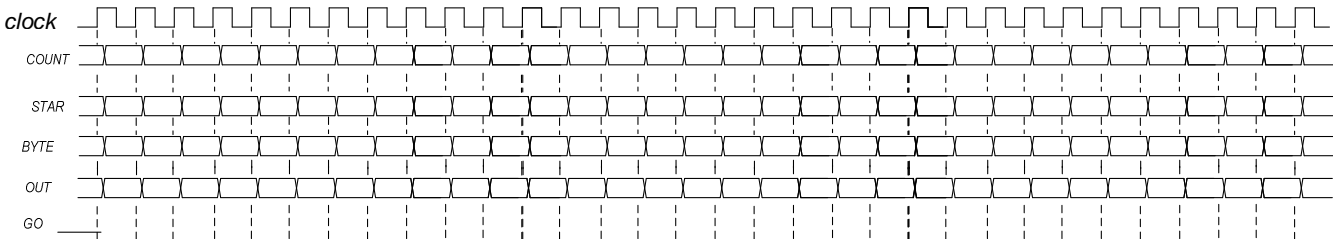
PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.5+6

NOTA: per gli esercizi 1e 2 (e analogamente per l'esercizio 5) dovranno essere consegnati due files: il file del programma MIPS (ovvero VERILOG) e il file relativo all'output (screenshot o copy/paste)

1. [6] Tramite il simulatore SPIM, (**utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS riportate in calce**), scrivere in assembly MIPS:
  - i) scrivere in assembly MIPS una funzione che svolga una moltiplicazione di due operandi interi con segno a 64 bit per ottenere un risultato su 64 bit. Gli operandi sono inizialmente nei registri (\$a1:\$a0) e (\$a3:\$a2). Il risultato deve trovarsi in (\$t1:\$t0)
  - ii) un chiamante che stampi su schermo in esadecimale il prodotto di 0x12345678 per se stesso e il prodotto di tale numero per il suo negato (in complemento a due).
  
2. [13] Tramite il simulatore SPIM (**utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS riportate in calce**), scrivere in assembly MIPS:
  - i) una funzione che svolga una moltiplicazione di due operandi interi con segno a 64 bit per ottenere un risultato su 128 bit. Gli operandi sono inizialmente nei registri (\$a1:\$a0) e (\$a3:\$a2). Il risultato deve trovarsi in (\$t3:\$t2:\$t1:\$t0)
  - ii) un chiamante che stampi su schermo in esadecimale il prodotto di 0x1234567890ABCDEF per se stesso e il prodotto di tale numero per il suo negato (in complemento a due) utilizzando la precedente funzione.
  
3. [7] Si consideri una cache di dimensione 96B e a 3 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 16 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: 155, 173, 115, 119, 122, 947, 318, 449, 234, 748, 377, 319, 283, 243, 391, 144, 770, 945, 61, 194. 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, 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 11/13 arrotondato al valore piu' vicino.
  
5. [10] Descrivere e sintetizzare l'Unità XXX che emette un byte generato in accordo alla legge di cui sotto. Il byte deve permanere all'uscita *out* di XXX per un numero di clock esattamente pari a  $numero\_clock = byte * 2$  e deve essere notificato dal fatto che la variabile *go* passa da 0 ad 1 per un ciclo di clock. I *byte* generati soddisfano la doppia condizione di essere numeri *dispari* e *multipli di tre*.



Tracciare il diagramma di temporizzazione come verifica della correttezza della descrizione dell'unità XXX (il modulo TopLevel e' riportato in calce)



6. [8] Sintetizzare una rete sequenziale utilizzando il modello di Moore con un ingresso X su due bit e una uscita Z su singolo bit che riconosca la sequenza di ingresso 01,11,10,00. Rappresentare la macchina a stati finiti per tale riconoscitore, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.

**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)
mfcl	mfcl \$F1,\$1	\$1=\$F2	Data from gen.reg. to C1 reg. (no conversion)
branch on false	bclf label	If (Temp == false) go to label	Temp is 'Condition-Code'
branch on true	bclt 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	Name	Reg. Num.	Usage	Reg. Num.	Usage
\$zero	0	The constant value 0	\$v0-\$v1	2-3	Results	\$f0, \$f2	Return values
\$\$0-\$7	16-23	Saved	\$fp, \$sp	30,29	frame pointer, stack pointer	\$f12,\$f14	Function arguments
\$\$0-\$19	8-15,24-25	Temporaries	\$ra, \$gp	31,28	return address, global pointer	\$f20,\$f22,\$f24,\$f26,\$f28,\$f30	Saved registers
\$\$a0-\$a3	4-7	Arguments	\$\$k0-\$k1	26,27	Kernel usage	\$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[8:0] COUNT=Xxx.COUNT;
  wire[7:0] BYTE=Xxx.BYTE;
  wire STAR=Xxx.STAR;
  wire GO=Xxx.go;
  wire[7:0] OUT=Xxx.out;
  XXX Xxx(go,out, clock,reset_);
endmodule
    
```