

DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI
--

**SVOLGIMENTO DELLA PROVA:**

PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16": es. N.1 + es. N.3 + es. N.4

PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1,2,3,5

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

PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.4,5

NOTA: per l'esercizio 1 (e analogamente per l'esercizio 4) dovranno essere consegnati due files: il file del programma MIPS (ovvero VERILOG) e il file relativo all'output (screenshot o copy/paste)

1. [18] Utilizzando il simulatore SPIM, codificare in assembly MIPS il seguente codice (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), **rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** (riportate in calce). Al termine della codifica consegnare 2 files: il programma in MIPS e l'output relativo.

```

#define N 4
#define MAXITER 20

double A[N*N] = {
    3.0, -1.0, 1.0, 0.0,
    0.0, 5.0, 0.0, -1.0,
    1.0, 0.0, -3.0, 1.0,
    0.0, 2.0, 1.0, 4.0
};

double b[N] = { 2.0, 9.0, -1.0, 9.0 };

void seidel(double *x, double *y, double *a, double c) {
    int j;
    *x = c;
    for (j = 0; j < N; ++j) *x -= a[j] * y[j];
}

void setup() {
    int i, j;
    for (i = 0; i < N; ++i) {
        c[i] = b[i] / A[i+i*N];
        for (j = 0; j < N; ++j) {
            R[i*N+j] = (i==j) ? 0 : A[i*N+j] / A[i*N+i];
        }
    }
}

int report() {
    int i;
    print_string("X: ");
}

for (i = 0; i < N; ++i) {
    print_double(x[i]); print_string(" ");
}
print_string(" - iter="); print_int(iter);
print_string(" e="); print_double(e);
print_string("\n");
}

int test_convergence() {
    int j, r;
    ++iter; e = 0;
    for (j = 0; j < N; ++j)
        e += (y[j] - x[j]) * (y[j] - x[j]);
    r = (e < 0.0001 * 0.0001 || iter == MAXITER);
    report();
    return (r);
}

int compute() {
    int i;
    do {
        for (i = 0; i < N; ++i) y[i] = x[i];
        for (i = 0; i < N; ++i) seidel(&x[i], x, &R[N*i], c[i]);
    } while (!test_convergence());
}

int main()
{
    setup(); compute(); report(); exit(0);
}

```

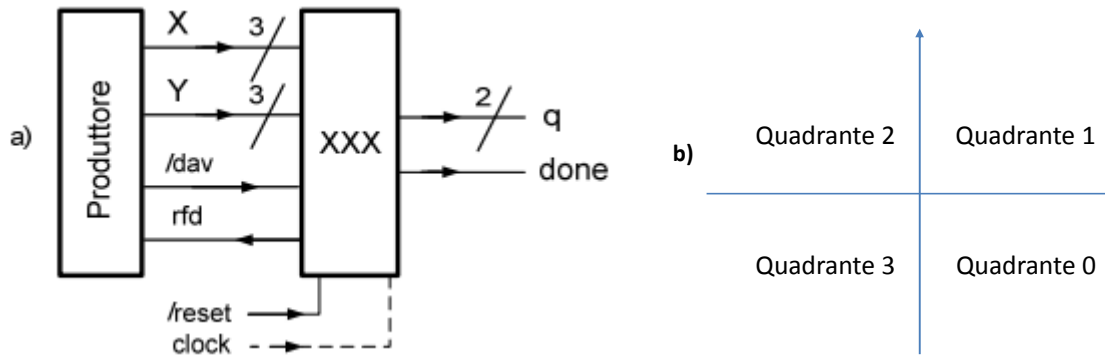
2. [7] Si consideri una cache di dimensione 128B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 16 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 124, 169, 167, 245, 183, 119, 235, 163, 288, 309, 310, 308, 213, 196, 377, 166, 362, 233, 163, 169. 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.
3. [5] In un processore MIPS con pipeline determinare i cicli necessari per eseguire due iterazioni per il seguente frammento di codice sia nel caso di propagazione abilitata che di propagazione disabilitata. Nota: e' presente 1 delay-slot e l'accesso ai registri nella fase di decodifica e write-back possono essere sovrapposte.

```

        add $1, $2, $3
L1: lw  $4,0($1)
        add $6, $1, $4
        lw  $5,0($1)
        bne $6, $5, L1
        nop

```

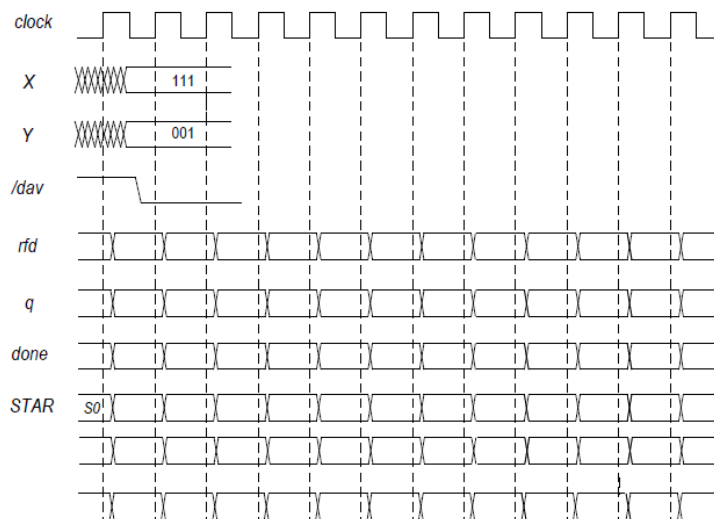
4) [10] L'unità XXX (Fig. a) preleva dal Produttore i due numeri naturali X e Y e li utilizza come le rappresentazioni (in complemento a due) di due numeri interi x e y. Interpreta x e y come le coordinate di un punto nel piano cartesiano ed emette tramite l'uscita q il numero d'ordine (0, 1, 2, 3) del quadrante a cui appartiene il punto (Fig. b) tenendo done ad 1 per un ciclo di clock.



Si specifichino i numeri d'ordine dei quadranti in modo da ottimizzare la rete combinatoria che produce q e si riportino tali numeri d'ordine nella Fig. b. Sempre in questa ottica, si considerino i semiassi come opportunamente appartenenti ai quadranti. Per chiarirsi le idee è utile completare preliminarmente la seguente tabella

x	y	X	Y	q
+1	+1	001	001	1
+1	-1			
-1	-1			
-1	+1			
+0	+0	000	000	1
+0	+1			
+0	-1			
+1	+0			
-1	+0			

Si descriva e si sintetizzi l'unità XXX e se ne tracci l'evoluzione nell'ipotesi che il Produttore fornisca le rappresentazioni di  $(x,y) = (-1,+1), (0,+2), (1,3), (2,-4), (3,-3), (-4,-2)$  indicando chiaramente nel grafico tali rappresentazioni (il codice Verilog del produttore e del top-module è fornito nell'ultima pagina del testo).



5) [12] Sintetizzare un riconoscitore di sequenze 11-01-10 utilizzando il modello di Moore. 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 \$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 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
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 CI 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	Name	Register Num.	Usage	Name	Usage
\$zero	0	The constant value 0	\$v0-\$v1	2-3	Results	\$f0, \$f1, ..., \$f31	Single precision floating point registers
\$s0-\$s7	16-23	Saved	\$fp, \$sp	30,29	frame pointer, stack pointer	\$f0, \$f2, ..., \$f30	Double precision floating point registers
\$t0-\$t9	8-15,24-25	Temporaires	\$ra, \$gp	31,28	return address, global pointer		
\$a0-\$a3	4-7	Arguments	\$k0-\$k1	26,27	Kernel usage		

## 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; #1 reset_=1; #300; $stop; end
  reg clock ; initial clock =0; always #5clock <=!clock);
  wire[2:0] X,Y;
  wire rfd, dav_;
  wire[1:0]q; wire done;
  wire[1:0] STAR=Xxx.STAR;
  XXX Xxx(rfd, dav_,X,Y, q,done, clock,reset_);
  Produttore PRO(rfd,dav_,X,Y);
endmodule

```

```

module Produttore(rfd,dav_,X,Y);
  input          rfd;
  output         dav_;
  output [3:0]   X,Y;
  reg DAV_;      assign dav_=DAV_;
  reg [2:0]      APP1_X, APP2_X, APP1_Y, APP2_Y; assign X=APP1_X, Y=APP1_Y;
  initial begin APP2_X='B111; APP2_Y=001; DAV_=1; end
  always
    begin #5; wait(rfd==1); #3 APP1_X=APP2_X; APP2_X=APP2_X+1;
APP1_Y=APP2_Y; APP2_Y=APP2_Y+1;
          #5 DAV_=0; wait(rfd==0); #1 APP1_X='HXX; APP1_Y='HXX; #9
    DAV_=1;end
endmodule

```