COMPITO di ARCHITETTURA DEI CALCOLATORI del 19-06-2019 **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 FOGLI PERSONALI, NO TELEFONI, SMARTPHONE/WATCH, ETC

COGNOME_____

NOME_____

SVOLGIMENTO DELLA PROVA:

☐ PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16, 16/17, 17/18, 18/19": es. N.1+2+3+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) [12/30] Trovare il codice assembly MIPS corrispondente al seguente programma **(usando solo e unicamente istruzioni della tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** riportate qua sotto per riferimento).

```
float q[8] = { 0.6931472, 0.9333737, 0.9888778, 0.9984959,
               0.9998293, 0.9999833, 0.9999986, 0.9999999 };
unsigned int mz = 11111;
float rnd(void) {
  mz = 31883 * (mz & 65535) + (mz >> 16);
  return (1.0 + mz) * 2.3283062e-10;
}
float gexp(float avg) {
  int i; float a, u, umin, ustar;

  for (a=0.0,u=rnd(); ; a = a + q[0]) {
    u = u + u;
    if ( 1.0 < u ) break;
  }

  u = u - 1.0;
  if (u <= q[0]) return (a + u) / avg;

  for (i=0,ustar = rnd(),umin = ustar; u > q[i]; ++i) {
    ustar = rnd();
    if (ustar < umin) umin = ustar;
  }

  return (a + umin * q[0]) / avg;
}

int main() {
  for (int n = 0; n < 10; ++n) {
    print_float(gexp(1.0/970000));
    print_string(" ");
  }
  exit(0);
}
```

Instructions

Opcode+Funct (hexadecimal)	Instruction	Example	Meaning	Comments
00+20/00+21	**add**	`add/addu $1,$2,$3`	$1 = $2 + $3	(signed/unsigned) 3 operands; exception possible
00+22/00+23	**subtract**	`sub/subu $1,$2,$3`	$1 = $2 - $3	(signed/unsigned) 3 operands; exception possible
08/09	**add immediate**	`addi/addiu $1,$2,100`	$1 = $2 + 100	(signed/unsigned) + constant ; exception possible
00+18/00+19	**multiplication**	`mult/multu $1, $2`	Hi,Lo= $1 x $2	(signed/unsigned) 64-bit Product ; result in Hi,Lo
00+1A/00+1B	**division**	`div/divu $1, $2`	Hi= $1 % $2, Lo = $1 / $2	(signed/unsigned) division
00+10/00+12	**move from Hi / move from Lo**	`mfhi/mflo $1`	$1 = Hi ($1 = Lo)	Create copy of Hi (Create a copy of Lo)
00+2A/00+2B	**set on less than**	`slt/sltu $1,$2,$3`	if ($2 < $3) $1 = 1; else $1 = 0	(signed/unsigned) compare $2 and $3 (less than)
0A/0B	**set on less than immediate**	`slti/sltiu $1,$2,100`	if ($2 < 100) $1 = 1; else $1 = 0	(signed/unsigned) compare $2 and constant (less than)
00+24/25/26/27	**and / or / xor / nor**	`and/or/xor/nor $1,$2,$3`	$1=$2&$3 / $2|$3 / $2^$3 / !($2|$3)	3 register operands; Logical AND/OR/XOR/NOR
0C/0D/0E	**and /or / xor immediate**	`andi/ori/xori $1,$2,100`	$1 = $2 & 100 / $2 | 100 / $2 ^100	Logical AND/OR/XOR register, constant
00+00	**shift left logical**	`sll $1,$2,10`	$1 = $2 << 10	Shift left by constant
00+02/00+03	**shift right (l=logical,a=arithmetic)**	`srl/sra $1,$2,10`	$1 = $2 >> 10	Shift right by constant (for arithmetic: sign is preserved)
00+04	**shift left logical**	`sllv $1,$2,10`	$1 = $2 << $3	Shift left by variable
00+06/00+07	**shift right (l=logical,a=arithmetic)**	`srlv/srav $1,$2,10`	$1 = $2 >> $3	Shift right by variable (for arithmetic: sign is preserved)
23/20	**load word / load byte**	`lw/lb $1,100($2)`	$1 = Memory[$2+100]	Data from memory to register
24	**load byte unsigned**	`lbu $1,100($2)`	$1 = Memory[$2+100]	Data from mem. To reg.; no sign extension
2B/28	**store word / store byte**	`sw/sb $1,100($2)`	Memory[$2+100] = $1	Data from register to memory
0F	**load upper immediate**	`lui $1,0x1234`	$1=0x1234'0000	load most significant 16 bits
PSEUDOINSTRUCTION	**load address**	`la $1,var`	$1 = &var	Load address of var (lui $1,H16(&var);ori $1, L16(&var)) H16/L16=high/low 16 bits of &var
02	**jump**	`j 10000`	go to 10000	Jump to target address
00+08	**jump register**	`jr $31`	go to $31	For switch, procedure return
03	**jump and link**	`jal 10000`	$31 = PC + 4;go to 10000	For procedure call
04	**branch on equal**	`beq $1,$2,100`	if ($1 = = $2) go to PC+4+100	Equal test; PC relative branch
05	**branch on not equal**	`bne $1,$2,100`	if ($1 != $2) go to PC+4+100	Not equal test; PC relative
00+0C	**syscall**	`syscall`	call OS service $v0	See table of system calls below
10+10,rs=10	**rfe**	`rfe`	shift right (k,e) bits in STATUS reg	Exit Kernel Mode, Enable Interrupts
PSEUDOINSTRUCTION	**branch unconditional**	`b 100`	go to PC+4+100	PC relative branch (e.g., beq $0,$0,100)
PSEUDOINSTRUCTION	**no operation**	`nop`	do nothing	Do nothing (e.g. sll $0,$0,0)
30	**load-linked**	`ll $1,100($2)`	$1=Memory[$2+100]	Read and start to monitor the given memory location
38	**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)
11+00 fmt=10/11	**add.s / add.d**	`add.x $f0,$f2,$f4`	$f0=$f2+$f4	Single and double precision add
11+01 fmt=10/11	**sub.s / sub.d**	`sub.x $f0,$f2,$f4`	$f0=$f2-$f4	Single and double precision subtraction
11+02 fmt=10/11	**mul.s / mul.d**	`mul.x $f0,$f2,$f4`	$f0=$f2*$f4	Single and double precision multiplication
11+03 fmt=10/11	**div.s / div.d**	`div.x $f0,$f2,$f4`	$f0=$f2/$f4	Single and double precision division
11+05 fmt=10/11	**abs.s / abs.d**	`abs.x $f0,$f2`	$f0=ABS($f2)	Single and double precision absolute value
11+06 fmt=10/11	**mov.s / mov.d**	`mov.x $f0,$f2`	$f0←$f2	Single and double precision move
11+07 fmt=10/11	**neg.s / neg.d**	`neg.x $f0,$f2`	$f0= – ($f2)	Single and double precision opposite value
11+3C(31,32,3D,3E,3F) fmt=10/11	**c.lt.s / c.lt.d (ne,eq,gt,le,ge)**	`c.lt.x $f0,$f2`	Temp=($f0<$f2)	Single and double: compare $f0 and $f2 <,=,!=,>,<=>=
11+00 fmt=4/0	**move to/from coprocessor 1**	`mtc1/mfc1 $1,$f2`	$f2=$1 / $1=$f2	Move $1 to/from C1reg. $f2 (no conversion)
10+00 fmt=4/0	**move to/from coprocessor 0**	`mtc0/mfc0 $1,$2`	$c2=$1 / $1=$c2	Move $1 to/from C0 reg. $f2 (no conversion)
11+00 fmt=6/2	**move to/from control reg of cop.1**	`ctc1/cfc1 $1,$cf2`	$cf2=$1 / $1=$cf2	Move $1 to/from C1-CONTROL register
11 fmt=8,ft=1/0	**branch on true/false**	`bc1t/bc1f label`	If (Temp = = true/false) go to label	Temp is 'Condition-Code'
31/39	**load/store floating point (32bit)**	`lwc1/swc1 $f0,0($1)`	$f0←Memory[$1] / Memory[$1]←$f0	Data from FP (C1) register to memory
11+21,fmt=10/11+22,fmt=11	**convert from/to single to/from double**	`cvt.d.s/cvt s.d $f0,$f2`	$f0=(double)$f2/$f0=(single)$f2	Type conversion
11+24,fmt=11/11+20	**convert from/to single to/from integer**	`cvt.w.s/cvt s.w $f1,$f0`	$f1=(int)$f0 / $f0=(single)$f2	Type conversion

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	Serv.No.($v0)	INPUT Arguments	OUTPUT Args
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

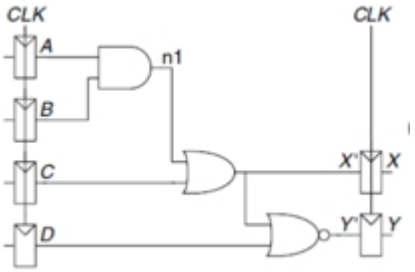
Service Name	Serv.No.($v0)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	$f0=float
read_double	7	---	$f0-f1=double
read_string	8	$a0=address of input buffer, $a1=max chars to read	---
sbrk	9	$a0=Number of bytes to be allocated	$v0=pointer to allocated memory
exit	10	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 19-06-2019

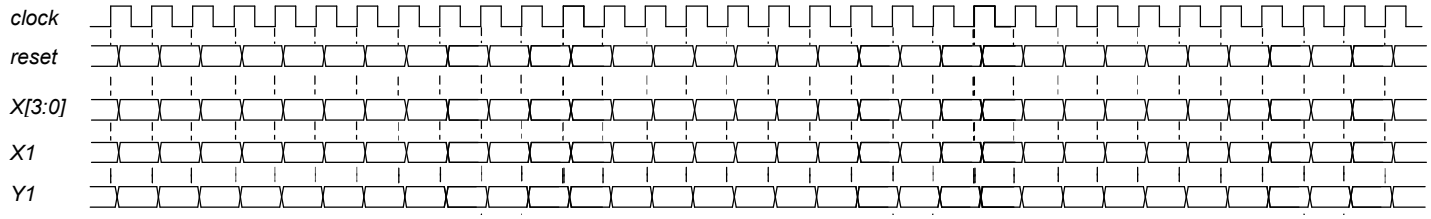
2) [5/30] Si consideri una cache di dimensione 128B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 16 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 226, 412, 262, 610, 239, 260, 214, 323, 321, 340, 315, 412, 515, 622, 790, 815. 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) [4/30] Spiegare il funzionamento del FLIP-FLOP-SR negative edge-triggered: riportare tabella della verita' e sintesi a livello di porte e per ogni riga della tabella della verita' dimostrare perché si ottiene quella data uscita usando il digramma temporale (in particolare per quanto riguarda la sensibilità sui fronti in discesa).

7) [9/30] Descrivere e sintetizzare in Verilog il modulo XXX rappresentato in figura. Tracciare il diagramma di temporizzazione come verifica della correttezza dell'unità XXX (il modulo Testbench e' fornito). Nota: si può 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.




```
`timescale 1ns/1ps
module TopLevel;
reg reset_;initial begin reset_=0; #22 reset_=1; #300; $stop; end
reg clock ;initial clock =0; always #5 clock <=(!clock);
reg[3:0] X;
wire X1=Xxx.x;
wire Y1=Xxx.y;
initial begin X='B000;
 wait(reset_==1);
 @(posedge clock); X<='B0000;@(posedge clock); X<='B0001;@(posedge clock); X<='B0010;@(posedge clock); X<='B0011;
 @(posedge clock); X<='B0100;@(posedge clock); X<='B0101;@(posedge clock); X<='B0110;@(posedge clock); X<='B0111;
 @(posedge clock); X<='B1000;@(posedge clock); X<='B1001;@(posedge clock); X<='B1010;@(posedge clock); X<='B1011;
 @(posedge clock); X<='B1100;@(posedge clock); X<='B1101;@(posedge clock); X<='B1110;@(posedge clock); X<='B1111;
 #10
 $finish;
end
XXX Xxx(clock,reset_,X[3],X[2],X[1],X[0], X1,Y1);
endmodule
```



COMPITO di ARCHITETTURA DEI CALCOLATORI del 19-06-2019

SOLUZIONE

ESERCIZIO 1

```
.data
q:      .float 0.6931472, 0.9333737,
0.9888778, 0.9984959, 0.9998293,
0.9999833, 0.9999986, 0.9999999
mz:     .word 11111
cf1:    .float 1.0
c970:   .word 970000
sp:     .asciiz " "
cf232:  .float 2.3283062e-10
c31883: .word 31883
c65535: .word 65535

.globl main
.text
#-----------------------------------
rnd:
  la   $t2,cf232    # &cf232
  la   $t3,c31883   # &c31883
  la   $t4,c65535   # &c65535
  la   $t5,mz       # &mz
  lwc1 $f0,0($t2)   # 2.32...
  lw   $t6,0($t3)   # 31883
  lw   $t7,0($t4)   # 65535
  lw   $t9,0($t5)   # mz
  and  $t8,$t9,$t7  # mz&65535
  multu $t8,$t6     # ()*31883
  mflo $t8
  srl  $t9,$t9,16   # mz>>16
  addu $t9,$t9,$t8  # ()+()
  sw   $t9,0($t5)   # update mz
  mtc1 $t9,$f4      # mz into c1
  cvt.s.w $f4,$f4   #(float)mz
  la   $t5,cf1      # &cf1
  lwc1 $f6,0($t5)   # 1.0
  add.s $f4,$f4,$f6 # 1.0+mz
  mul.s $f0,$f4,$f0 # result
  jr $ra
#-----------------------------------
# f12=avg, s0=i, f20=a,
# f22=u,f24=umin,f26=ustar,f28=1.0
gexp:
#-----------------------------------
# PROLOGO
   addi $sp,$sp,-32    #alloco frame
   sw   $ra, 28($sp)   #salvo old-ra
   sw   $fp, 24($sp)   #salvo old-fp
   sw   $s0, 20($sp)   #salvo s0
   swc1 $f20,16($sp)   #salvo f20
   swc1 $f22,12($sp)   #salvo f22
   swc1 $f24, 8($sp)   #salvo f24
   swc1 $f26, 4($sp)   #salvo f26
   swc1 $f28, 0($sp)   #salvo f28
   add  $fp,$0,$sp     #NUOVO fp
   la   $t1,cf1        # &cf1
   lwc1 $f28,0($t1)    # f28=1.0

   mtc1 $0,$f20        # a=0.0
   jal  rnd            # f0=rnd()
   mov.s $f22,$f0      # u=(.)
   la   $t0,q          # &q
   lwc1 $f4,0($t0)     # q[0]
forlini:
   add.s $f22,$f22,$f22# u=u+u
   c.lt.s $f28,$f22    # 1.0<u
   bc1t  for1break
   add.s $f20,$f20,$f4 # a=a+(.)
   j    forlini
for1break:
   sub.s $f22,$f22,$f28# u=u-1.0
   c.le.s $f22,$f4     # u<=?q[0]
   bc1f fineif2
   add.s $f0,$f20,$f22 # a+u
   div.s $f0,$f0,$f12  # (.)/avg
   j gexp_epilogo
fineif2:
   add  $s0,$0,$0      # i=0
   jal  rnd
   mov.s $f26,$f0      # ustar=rnd()
   mov.s $f24,$f26     # umin=ustar
for2ini:
   jal   rnd
   mov.s  $f26,$f0     # ustar=rnd()
   c.lt.s $f26,$f24    # ustar<?umin
   bc1f  fineif3
   mov.s $f24,$f26     # umin=ustar
fineif3:
   add  $t0,$s0,$s0
   add  $t0,$t0,$t0    # i*4
   la   $t1,q          # &q
   add  $t1,$t1,$t0    # &q[i]
   lwc1 $f4,0($t1)     # q[i]
   c.lt.s $f4,$f22     # q[i]<?u
   bc1f for2end
   addi $s0,$s0,1      # ++i
   j    for2ini
for2end:
   la    $t0,q         # &q
   lwc1  $f4,0($t0)    # q[0]
   mul.s $f0,$f24,$f4  # umin*q[0]
   add.s $f0,$f20,$f22 # a+(.)
   div.s $f0,$f0,$f12  # (.)/avg
#-----------------------------------
# EPILOGO
gexp_epilogo:
   lw   $ra, 28($sp)#ripristi. old-ra
   lw   $fp, 24($sp)#ripristi. old-fp
   lw   $s0, 20($sp)#ripristi. s0
   lwc1 $f20,16($sp)#ripristi. f20
   lwc1 $f22,12($sp)#ripristi. f22
   lwc1 $f24, 8($sp)#ripristi. f24
   lwc1 $f26, 4($sp)#ripristi. f26
   lwc1 $f26, 4($sp)#ripristi. f28
   addi $sp,$sp,32  #dealloco frame
   jr $ra
#-----------------------------------
main:
   add  $s0,$0,$0  # n=0
   la   $t1,cf1    # &cf1
   la   $t2,c970   # &c970
   lwc1 $f20,0($t1)# f20=1.0
   lw   $s3,0($t2) # s3=970000
   addi $s4,$0,10  # s4=10
main_forini:
   slt  $t9,$s0,$s4# n<?10
   beq  $t9,$0,main_forend

   mtc1 $s3,$f4    # 970000
   cvt.s.w $f6,$f4 # 970000.0
   div.s $f12,$f20,$f6#1.0/970000.0
   jal  gexp       # result in f0
   mov.s $f12,$f0  # number to print
   addi $v0,$0,2   # serv.2
   syscall
   la   $a0,sp     # &sp
   addi $v0,$0,4   # serv.4
   syscall
   addi $s0,$s0,1  # n++
   j    main_forini
main_forend:
   addi $v0,$0,10  # serv.10
   syscall
```

Console

```
2327159.00000000 2099008.75000000 1803283.87500000 1326078.75000000 1416302.25000000
818284.87500000 1403477.37500000 1246176.37500000 1503524.75000000 1456143.50000000
```

ESERCIZIO 2

Sia X il generico riferimento, A=associativita', B=dimensione del blocco, C=capacita' della cache.

Si ricava S=C/B/A=# di set della cache=128/16/4=2, XM=X/B, XS=XM%S, XT=XM/S:

```
A=4, B=16, C=128, RP=FIFO, Thit=4, Tpen=40, 16 references:
=== T    X    XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  226   14   7   0   2  0 [0]:3,0,0,0 [0]:0,0,0,0  [0]:7,-,-,-
=== W  412   25  12   1  12  0 [1]:3,0,0,0 [1]:0,0,0,0  [1]:12,-,-,-
=== R  262   16   8   0   6  0 [0]:2,3,0,0 [0]:0,0,0,0  [0]:7,8,-,-
=== W  610   38  19   0   2  0 [0]:1,2,3,0 [0]:0,0,0,0  [0]:7,8,19,-
=== R  239   14   7   0  15  1 [0]:1,2,3,0 [0]:0,0,0,0  [0]:7,8,19,-
=== W  260   16   8   0   4  1 [0]:1,2,3,0 [0]:0,1,0,0  [0]:7,8,19,-
=== R  214   13   6   1   6  0 [1]:2,3,0,0 [1]:0,0,0,0  [1]:12,6,-,-
=== W  323   20  10   0   3  0 [0]:0,1,2,3 [0]:0,1,0,0  [0]:7,8,19,10
=== R  321   20  10   0   1  1 [0]:0,1,2,3 [0]:0,1,0,0  [0]:7,8,19,10
=== W  340   21  10   1   4  0 [1]:1,2,3,0 [1]:0,0,0,0  [1]:12,6,10,-
=== R  315   19   9   1  11  0 [1]:0,1,2,3 [1]:0,0,0,0  [1]:12,6,10,9
=== W  412   25  12   1  12  1 [1]:0,1,2,3 [1]:1,0,0,0  [1]:12,6,10,9
=== R  515   32  16   0   3  0 [0]:3,0,1,2 [0]:0,1,0,0  [0]:16,8,19,10
=== W  622   38  19   0  14  1 [0]:3,0,1,2 [0]:0,1,1,0  [0]:16,8,19,10
=== R  790   49  24   1   6  0 [1]:3,0,1,2 [1]:0,0,0,0  [1]:24,6,10,9
=== W  815   50  25   0  15  0 [0]:2,3,0,1 [0]:0,0,1,0  [0]:16,25,19,10
--------------------------------------
P1 Nmiss=11   Nhit=5  Nref=16  mrate=0.687500   AMAT=th+mrate*tpen=31.5
```

CONTENTUTI dei 4 SET al termine:

LISTA BLOCCHI USCENTI:

```
(out: XM=14 XT=7  XS=0 )

(out: XM=25 XT=12 XS=1 )
(out: XM=16 XT=8  XS=0 )
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 19-06-2019

SOLUZIONE

ESERCIZIO 3

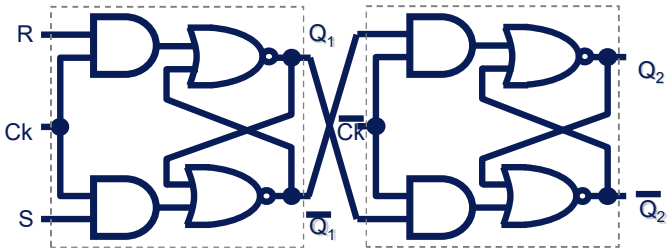
Quando CK=1, il master Clocked-SR latch e' trasparente e lo slave e' opaco. Quindi i valori S e R si propagano su Q1. Quando CK=0, il master diventa opaco e lo slave trasparente: il valore di Q1 si propaga su Q, ma Q1 resta scollegato da S e R. QUINDI: qualunque valore sia presente su S e R un momento PRIMA del fronte in discesa del clock da 1 a 0, condiziona Q immediatamente DOPO la discesa del clock. In tutti gli altri momenti Q mantiene il suo vecchio valore, perche' c'e' sempre un Clocked-SR latch opaco che blocca il cammino da S e R a Q.

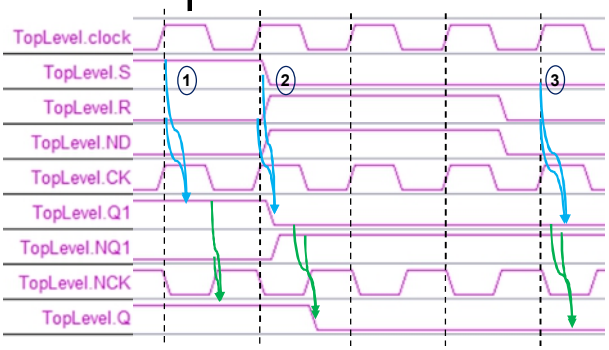
S-R negative edge triggered

Tabella di Verità

Ck	S	R	Q	/Q
↓	0	0	Q	/Q
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	---	---
0	X	X	Q	/Q
1	X	X	Q	/Q
↑	X	X	Q	/Q

Schema logico





1) SR='b10 → set → Q=1
2) SR='b01 → reset → Q=0
3) SR='b00 → hold → Q=Q

ESERCIZIO 7

Codice VERILOG:

```
module chain(clk,rst_,a,b,c,d,  x,y);
input clk,rst_,a,b,c,d;
output x,y;
reg x,y;
wire n1, n2;
reg areg, breg, creg, dreg;
always @ (posedge clk)  if(rst_==1) #3
begin
  areg <= a;
  breg <= b;
  creg <= c;
  dreg <= d;
  x <= n2;
  y <= ~(dreg | n2);
end
assign n1 = areg & breg;
assign n2 = n1 | creg;
endmodule
```

Diagramma temporale: