COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-04-2016

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

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)

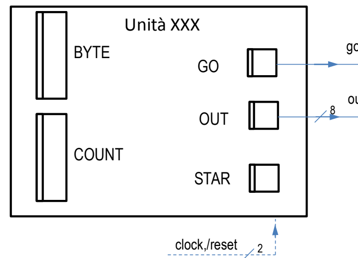
1. [6] Tramite il simulatore SPIM, (**utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS riportate in calce**), scrivere in assembly MIPS:
   i) scrivere in assembly MIPS una funzione che svolga una moltiplicazione 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 complento 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 complento a due) utilizzando la precedente funzione.

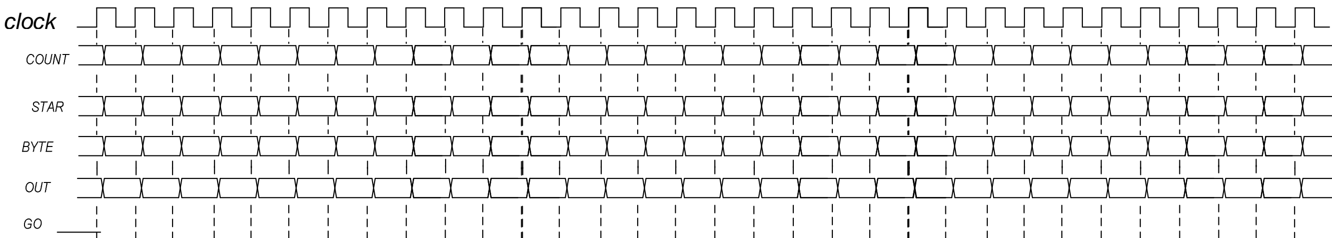
3. [7] Si consideri una cache di dimensione 96B e a 3 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 16 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi 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.

COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-04-2016 **MATRICOLA**____________________

(SOLUZIONE) **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[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
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-04-2016

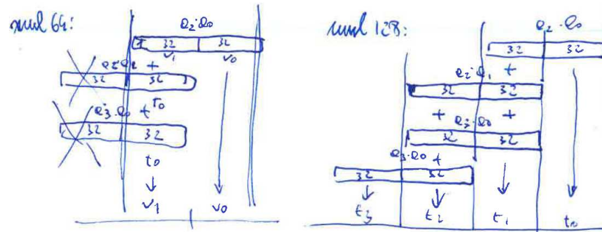
MATRICOLA____________________

(SOLUZIONE)

COGNOME______________________

NOME______________________

ESERCIZIO 1) ESERCIZIO 2)



Attenzione a tenere conto dei carry che vengono generati nelle somme.

```
                .data
x0:             .word 0x12345678
x1: .word 0x00000000
x2: .word 0x90ABCDEF
x3: .word 0x12345678
nl: .ascii "\n"
m64: .asciiz "mul64:\n"
m128: .asciiz "mul128:\n"
buf:.space(34)
                .text
                .globl         main

changesign:
                nor   $v0, $a0, $0
                nor   $v1, $a1, $0
                addiu $t0, $v0, 1
                sltu  $t1, $t0, $v0
                addu  $v1, $v1, $t1
                add   $v0, $t0, $0
                jr    $ra

mul64:
                addi  $sp, $sp, -8   # space for ra and 5th param of hexprint
                sw    $ra, 4($sp)
                multu $a0, $a2       # a0*a2
                mflo  $v0            # lo(a0*a2)
                mfhi  $v1            # hi(a0*a2)
                multu $a1, $a2       # a1*a2
                mflo  $t0            # lo(a1*a2)
                addu  $v1, $v1, $t0  # hi(a0*a2)+lo(a1*a2)
                multu $a0, $a3       # a0*a3
                mflo  $t0            # lo(a0*a3)
                addu  $v1, $v1, $t0  # hi(a0*a2)+lo(a1*a2)+lo(a0*a3)
                add   $a0, $0, $v0
                add   $a1, $0, $v1
                addi  $t9, $0, 16    # 64 bit printing
                sw    $t9, 0($sp)    # push onto the stack the 5th param
                jal   hexprint       # print result
                lw    $ra, 4($sp)
                addi  $sp, $sp, 8
                jr    $ra

mul128:
                addi  $sp, $sp, -8   # space for ra and 5th param of hexprint
                sw    $ra, 4($sp)
                slt   $t9, $a3, $0   # sgn(a3:a2)
                slt   $t8, $a1, $0   # sgn(a1:a0)
                xor   $t7, $t9, $t8  # sgn(a3:a2) XOR sgn(a1:a0)

                beq   $t8, $0, noch1
                jal   changesign
                add   $a0, $v0, $0
                add   $a1, $v1, $0
noch1:
                beq   $t9, $0, noch2
                addi  $sp, $sp, -8
                sw    $a0, 0($sp)    # save $a0, $a1
                sw    $a1, 4($sp)
                add   $a0, $a2, $0
                add   $a1, $a3, $0
                jal   changesign
                add   $a2, $v0, $0
                add   $a3, $v1, $0
                lw    $a0, 0($sp)    # restore $a0, $a1
                lw    $a1, 4($sp)
                addi  $sp, $sp, 8
noch2:

                multu $a0, $a2       # a0*a2
                mflo  $t0            # lo(a0*a2)
                mfhi  $t1            # hi(a0*a2)
                nor   $t5, $t1, $0   #
                multu $a1, $a2       # a1*a2
                mflo  $t4            # lo(a1*a2)
                mfhi  $t2            # hi(a1*a2)
                sltu  $t5, $t5, $t4  # carry(hi(a0*a2)+lo(a1*a2)
                addu  $t1, $t1, $t4  # hi(a0*a2)+lo(a1*a2)
                multu $a0, $a3       # a0*a3
                add   $t2, $t2, $t5  # hi(a1*a2)+carry
                nor   $t5, $t1, $0   #
                mflo  $t4            # lo(a0*a3)
                mfhi  $t3            # hi(a0*a3)
                sltu  $t5, $t5, $t4  # carry(hi(a0*a2)+lo(a1*a2)+lo(a0*a3))
                addu  $t1, $t1, $t4  # hi(a0*a2)+lo(a1*a2)+lo(a0*a3)
                multu $a1, $a3       # a1*a3
                add   $t2, $t2, $t5  # hi(a1*a2)+carry+carry
                nor   $t4, $t2, $0   #
                sltu  $t4, $t4, $t3  # carry(hi(a1*a2)+hi(a0*a3))
                addu  $t2, $t2, $t3  # hi(a1*a2)+hi(a0*a3)
                nor   $t5, $t2, $0   #
                mfhi  $t3            # hi(a1*a3)
                add   $t3 $t3, $t4   # hi(a1*a3)+carry(hi(a1*a2)+hi(a0*a3))
                mflo  $t4            # lo(a1*a3)
                sltu  $t5, $t5, $t4  # carry(hi(a1*a2)+hi(a0*a3)+lo(a3*a0))
                add   $t3, $t3, $t5  # hi(a1*a3)+carry+carry
                addu  $t2, $t2, $t4  # hi(a1*a2)+hi(a0*a3)+lo(a3*a0)

                beq   $t7, $0, noch3
                nor   $t0, $t0, $0
                nor   $t1, $t1, $0
                nor   $t2, $t2, $0
                nor   $t3, $t3, $0

                addiu $t4, $t0, 1
                sltu  $t5, $t4, $t0
                add   $t0, $t4, $0

                addu  $t4, $t1, $t5
                sltu  $t5, $t4, $t1
                add   $t1, $t4, $0

                addu  $t4, $t2, $t5
                sltu  $t5, $t4, $t2
                add   $t2, $t4, $0

                addu  $t3, $t3, $t5
noch3:
                add   $a0, $0, $t0
                add   $a1, $0, $t1
                add   $a2, $0, $t2
                add   $a3, $0, $t3
                addi  $t9, $0, 32    # 128 bit printing
                sw    $t9, 0($sp)    # push onto the stack the 5th param
                jal   hexprint       # print result

                lw    $ra, 4($sp)
                addi  $sp, $sp, 8
                jr    $ra

hexprint:
                lw    $t9, 0($sp)    # read the lenght of the hex string
                la    $t0, buf       # buffer address
                add   $t1, $t0, $t9  #
                addi  $t1, $t0, 1    # point ot the end of the buffer
                sb    $0,  0($t1)    # end of string
                la    $t1, nl        # read the new line character
                lb    $t1, 0($t1)    #
                add   $t0, $t0, $t9  # point to the last character of the buf
                addi  $t8, $0, 8     # constant 8

loop1:
                sb    $t1, 0($t0)    # store character
                addi  $t9, $t9, -1   # update buffer counter
                addi  $t0, $t0, -1   # update buffer pointer
                slti  $t3, $t9, 0
                andi  $t1, $a0, 0xF  # get last hex digit
                srl   $a0, $a0, 4    # shift right the other digits
                addi  $t1, $t1, 0x30 # transform into a ASCII char
                slti  $t2, $t1, 0x3A #
                bne   $t2, $0,  step1# if greater than 9
                addi  $t1, $t1, 7    # add 7 --> A, B, C, D, E, F
step1:
                div   $t9, $t8       # check if done with 8 digits
                mfhi  $t4            #
                bne   $t4, $0,  step2# if so:
                add   $a0, $a1, $0   # shift right the other registers
                add   $a1, $a2, $0   #
                add   $a2, $a3, $0   #
step2:
                beq   $t3, $0,  loop1# loop on all digits
                addi  $a0, $t0, 1    # point to the beginning of the buf
                addi  $v0, $0,  4    # print the hex string
                syscall
                jr    $ra

main:
                la    $a0, m64
                addi  $v0, $0, 4
                syscall              # print "mul64:"

                la    $s0, x0        # (s1:s0)=(x1:x0)
                lw    $s0, 0($s0)
                la    $s1, x1
                lw    $s1, 0($s1)

                add   $a0, $s0, $0   # print 1st operand
                add   $a1, $s1, $0
                addi  $t9, $0, 16    # 64 bit printing
                addi  $sp, $sp, -4
                sw    $t9, 0($sp)    # push onto the stack the 5th param
                jal   hexprint       # print operand
                addi  $sp, $sp 4     # restore the stack

                add   $a0, $s0, $0
                add   $a1, $s1, $0
                jal   changesign
                add   $s2, $v0, $0
                add   $s3, $v1, $0

                add   $a0, $v0, $0   # print 2nd operand
                add   $a1, $v1, $0
                addi  $t9, $0, 16    # 64 bit printing
                addi  $sp, $sp, -4
                sw    $t9, 0($sp)    # push onto the stack the 5th param
                jal   hexprint       # print operand
                addi  $sp, $sp 4     # restore the stack

                add   $a0, $s0, $0
                add   $a1, $s1, $0
                add   $a2, $s0, $0
                add   $a3, $s1, $0
                jal   mul64          # do 64-bit multiplication

                add   $a0, $s0, $0
                add   $a1, $s1, $0
                add   $a2, $s2, $0
                add   $a3, $s3, $0
                jal   mul64          # do 64-bit multiplication

                la    $a0, m128
                addi  $v0, $0, 4
                syscall              # print "mul128:"

                la    $s0, x2        # (s1:s0)=(x3:x2)
                lw    $s0, 0($s0)
                la    $s1, x3
                lw    $s1, 0($s1)

                add   $a0, $s0, $0   # print 1st operand
                add   $a1, $s1, $0

                addi  $t9, $0, 16    # 64 bit printing
                addi  $sp, $sp, -4
                sw    $t9, 0($sp)    # push onto the stack the 5th param
                jal   hexprint       # print operand
                addi  $sp, $sp 4     # restore the stack

                add   $a0, $s0, $0
                add   $a1, $s1, $0
                jal   changesign
                add   $s2, $v0, $0   # (s3:s2)= - (x3:x2)
                add   $s3, $v1, $0

                add   $a0, $v0, $0   # print 2nd operand
                add   $a1, $v1, $0

                addi  $t9, $0, 16    # 64 bit printing
                addi  $sp, $sp, -4
                sw    $t9, 0($sp)    # push onto the stack the 5th param
                jal   hexprint       # print operand
                addi  $sp, $sp 4     # restore the stack

                add   $a0, $s0, $0
                add   $a1, $s1, $0
                add   $a2, $s0, $0
                add   $a3, $s1, $0
                jal   mul128          # do 128-bit multiplication

                add   $a0, $s0, $0
                add   $a1, $s1, $0
                add   $a2, $s2, $0
                add   $a3, $s3, $0
                jal   mul128          # do 128-bit multiplication

                addi  $v0, $0, 10
                syscall
```

Output:

Console

```
mul64:
0000000012345678
FFFFFFFFEDCBA988
014B66DC1DF4D840
FEB49923E20B27C0
mul128:
1234567890ABCDEF
EDCBA9876F543211
014B66DC328828BCA6475F09A2F2A521
FEB49923CD77D74359B8A0F65D0D5ADF
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-04-2016 **MATRICOLA**____________________

(SOLUZIONE) **COGNOME**______________________

NOME________________________

ESERCIZIO 3)
A=3,B=16,C=96.
Si ricava S=C/B/A=# di set della cache=128/16/4=2, XM=X/B, XS=XM%S, XT=XM/S:

```
=== T    X    XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  155    9   4   1  11  0 [1]:2,0,0 [1]:0,0,0  [1]:4,-,-
=== W  173   10   5   0  13  0 [0]:2,0,0 [0]:0,0,0  [0]:5,-,-
=== R  115    7   3   1   3  0 [1]:1,2,0 [1]:0,0,0  [1]:4,3,-
=== W  119    7   3   1   7  1 [1]:1,2,0 [1]:0,1,0  [1]:4,3,-
=== R  122    7   3   1  10  1 [1]:1,2,0 [1]:0,1,0  [1]:4,3,-
=== W  947   59  29   1   3  0 [1]:0,1,2 [1]:0,1,0  [1]:4,3,29
=== R  318   19   9   1  14  0 [1]:2,0,1 [1]:0,1,0  [1]:9,3,29 (out: XM=9  XT=4  XS=1 )
=== W  449   28  14   0   1  0 [0]:1,2,0 [0]:0,0,0  [0]:5,14,-
=== R  234   14   7   0  10  0 [0]:0,1,2 [0]:0,0,0  [0]:5,14,7
=== W  748   46  23   0  12  0 [0]:2,0,1 [0]:0,0,0  [0]:23,14,7 (out: XM=10 XT=5  XS=0)
=== R  377   23  11   1   9  0 [1]:1,2,0 [1]:0,0,0  [1]:9,11,29 (out: XM=7  XT=3  XS=1 )
=== W  319   19   9   1  15  1 [1]:2,1,0 [1]:1,0,0  [1]:9,11,29
=== R  283   17   8   1  11  0 [1]:1,0,2 [1]:1,0,0  [1]:9,11,8 (out: XM=59 XT=29 XS=1)
=== W  243   15   7   1   3  0 [1]:0,2,1 [1]:1,0,0  [1]:9,7,8 (out: XM=23 XT=11 XS=1 )
=== R  391   24  12   0   7  0 [0]:1,2,0 [0]:0,0,0  [0]:23,12,7 (out: XM=28 XT=14 XS=0 )
=== W  144    9   4   1   0  0 [1]:2,1,0 [1]:0,0,0  [1]:4,7,8 (out: XM=19 XT=9  XS=1 )
=== R  770   48  24   0   2  0 [0]:0,1,2 [0]:0,0,0  [0]:23,12,24 (out: XM=14 XT=7  XS=0 )
=== W  945   59  29   1   1  0 [1]:1,0,2 [1]:0,0,0  [1]:4,7,29 (out: XM=17 XT=8  XS=1 )
=== R   61    3   1   1  13  0 [1]:0,2,1 [1]:0,0,0  [1]:4,1,29 (out: XM=15 XT=7  XS=1 )
=== W  194   12   6   0   2  0 [0]:2,0,1 [0]:0,0,0  [0]:6,12,24 (out: XM=46 XT=23 XS=0 )
-------------------------------------
P1 Nmiss=17   Nhit=3  Nref=20  mrate=0.85   AMAT=38
```

Situazione finale cache:

```
[0]:6,12,24
[1]:4,1,29
```

Lista blocchi uscenti:

```
out: XM=9  XT=4  XS=1
out: XM=10 XT=5  XS=0
out: XM=7  XT=3  XS=1
out: XM=59 XT=29 XS=1
out: XM=23 XT=11 XS=1
out: XM=28 XT=14 XS=0
out: XM=19 XT=9  XS=1
out: XM=14 XT=7  XS=0
out: XM=17 XT=8  XS=1
out: XM=15 XT=7  XS=1
out: XM=46 XT=23 XS=0
```

ESERCIZIO 3)
Normalizzando 11/13 si ottiene: 22/13*2^-1 ovvero m=22/13, e=-1. Ricaviamo quindi S,M,E.

L' "uno" iniziale non viene rappresentato nel formato IEEE-754. Essendo m = 22/13=$1.\overline{692307}$ si ha:
M = m-1= $0.\overline{692307}$
Successivamente si puo' ricavare la rappresentazione binaria di M con 52 bit moltiplicando per 2 e ricavando via via la n-esima cifra piu' significativa:
$0.\overline{692307}$ *2 = $1.\overline{384615}$ → **1**, $0.\overline{384615}$ * 2 = $0.\overline{769230}$ → **0**, $0.\overline{769230}$*2= $1.\overline{538461}$ → **1**, ...
(notare che NON si deve mai troncare il numero: occorre mantenere tutte le cifre periodiche).
Dopo le prime 12 cifre binarie le cifre si ripetono (ribadendo la periodicita' del numero), quindi e'facilmente predicibile il resto delle 40 cifre binarie della mantissa. Inoltre la 53-esima cifra della mantissa corrisponde ad uno 0 quindi il piu'vicino numero rappresentabile in IEEE-754 doppia precisione e'quello che si ottiene troncando cosi'com'e'il valore ottenuto per M.
Ovvero:
M = 1011 0001 0011 1011 0001 0011 1011 0001 0011 1011 0001 0011 1011

Per l'esponente, ricordando che nel caso di doppia precisione il valore della polarizzazione e' 1023:
E = e+1023 = -1+1023=1022 ovvero 0111 1111 110

Inoltre per il segno S = 0

Quindi la rappresentazione cercata e':
0011 1111 1110 1011 0001 0011 1011 0001 0011 1011 0001 0011 1011 0001 0011 1011

COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-04-2016 MATRICOLA________________

(SOLUZIONE) COGNOME________________

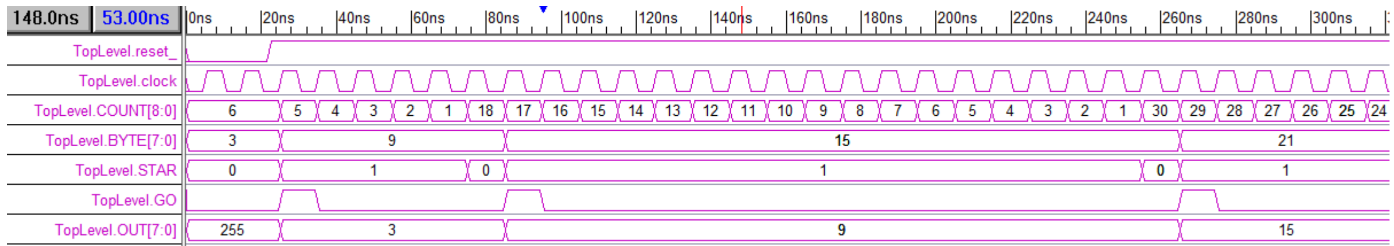
NOME________________

ESERCIZIO 5)

Codice Verilog del modulo da realizzare (possibile soluzione con Mealy-Ritardato):

```
module XXX(go,out,  clock,reset_);
input         clock, reset_;
output        go;
output [7:0] out;                          //go,out fanno le veci di 'z'

reg STAR;   parameter S0=0,S1=1;           //STAR, COUNT, BYTE: definizione stato
reg [8:0]  COUNT;                          //
reg [7:0]  BYTE;                           //

reg        GO;           assign go=GO;
reg [7:0]  OUT;          assign out=OUT;   //GO,OUT fanno le veci di 'OUTR'

wire[7:0]  new_byte = (BYTE==255)?3:(BYTE+6); // legge f(X,S) (in questo caso X non c'è)
wire[8:0]  Num_Cicli = {BYTE,1'B0};           //

always @(reset_==0) begin STAR=S0; GO<=0; OUT=255; BYTE=3; COUNT<=6; end
always @(posedge clock) if (reset_==1) #0
  casex(STAR)
   S0:  begin GO<=1; OUT<=BYTE;  BYTE<=new_byte; COUNT<=(COUNT-1);      STAR<=S1; end
   S1:  begin GO<=0;             COUNT<=(COUNT==1)?Num_Cicli:(COUNT-1); STAR<=(COUNT==1)?S0:S1; end
  endcase
endmodule
```

Diagramma di Temporizzazione:



ESERCIZIO 6)

Macchina a Stati Finiti:

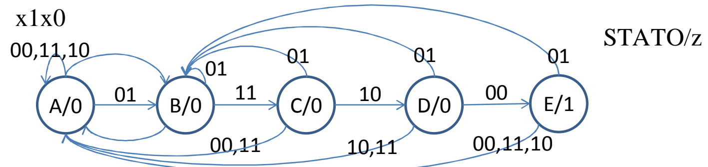


Tabella delle transizioni (e sua possibile codifica):

x_1x_0	00	01	11	10	z
A	A	B	A	A	0
B	A	B	C	A	0
C	A	B	A	D	0
D	E	B	A	A	0
E	A	B	A	A	1

$y_2y_1y_0$ \ x_1x_0	00	01	11	10	z
000	000	001	000	000	0
001	000	001	011	000	0
011	000	001	000	010	0
010	100	001	000	000	0
100	000	001	000	000	1
101	---	---	---	---	---
111	---	---	---	---	---
110	---	---	---	---	---

a_2,a_1,a_0

Equazioni booleane:

Per CN1:

a2=y1/y0/x1/x0

a1=/y1y0x1x0+ y1y0x1/x0

a0=/x1x0+/y2y1x1x0

Per CN2:

z=y2

Circuito sequenziale sincronizzato basato su FF-D: