COMPITO di ARCHITETTURA DEI CALCOLATORI del 17-01-2018 (rev1.2) **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, 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.

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) [18] 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). Il tipo 'int64_t' e' un intero che occupa due registri interi a 32 bit; le operazioni di shift e somma devono essere adattate.

```
double A[9]={10.0,1.0,2.0,1.0,20.0,3.0,2.0,3.0,30.0};

void printmat(char *name, double *X, int m) {
   int i;
   print_string(name); print_string("[");
   print_int(m); print_string("]=\n");
   for (i=0; i<m; ++i)
     { print_double(X[i]); print_string(" "); }
   print_string("\n");
}

double mysqrt(double x) {
   int_64t vint = *(int64_t*)&x; // stessi bit visti come int64_t
   vint = (1 << 61) + (vint >> 1) - (1 << 51);
   return *(double*)&vint; // Interpreto di nuovo come double
}
```

```
double *cholesky(double *A, int n) {
   double *L, s; int i, j, k;

   L = (double*)sbrk(n * n * 8);
   for (j=0; j<n; j++) {
      for (s=0, k=0; k<j; k++) s += L[j*n+k] * L[j*n+k];
      L[j*n+j] = mysqrt(A[j*n+j] - s);
      for (i=j+1; i<n; i++) {
         for (s=0, k=0; k<j; k++) s += L[i*n+k] * L[j*n+k];
         L[i*n+j] = (1.0 / L[j*n+j] * (A[i*n+j] - s));
      }
   }
   return L;
}

int main() {
   double *R;
   printmat("X", A, 9);
   R = cholesky(A, 3);
   printmat("R", R, 9);
}
```

2) [8] Si consideri una cache di dimensione 64B e a 2 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 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: 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) [4] Assemblare il seguente programma MIPS, utilizzando la tabella sottostante e riportando il formato utilizzato e i valori in esadecimale di ciascun campo di quel formato (es. LW $2, 0($1) → FORMATO I: 23 1 2 0); il programma viene caricato all'indirizzo standard 0x0040'0000:

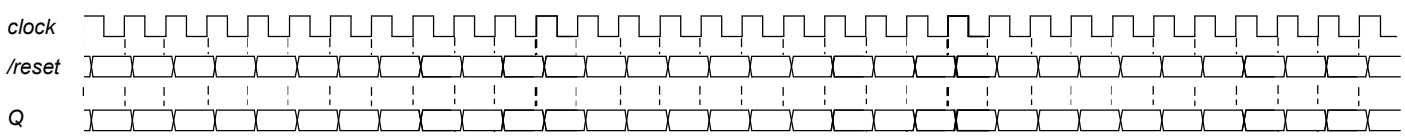
```
lab:  LW  $2, 0($1)
      LW  $4, 0($3)
      SLL  $5, $2, 10
      SRL  $6, $4, 12
      JAL  fun
      SLT  $7, $5, $6
      BNE  $7, $0, lab
      JR  $31
fun:  DIV  $10, $11
      JR  $31
```

4) (non assegnato)

5) (non assegnato)

6) (non assegnato)

7) [8] **Realizzare** in Verilog (per studenti 2014 e anni precedenti --> v.nota finale) sia un ripple counter a 4-bit che il relativo testbench: il clock ha un periodo di 10ns; il segnale _reset e' attivo basso: resta alto per 5ns, basso per 20ns, e poi ritorna alto per 600ns. Il contatore inizia il conteggio producendo sull'uscita Q il valore binario 0000 quando il segnale di /reset e'attivato, appena disattivato il /reset il conteggio prosegue. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità riportando i segnali clock, /reset, uscita Q per la durata complessiva (625ns). 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. (Per studenti 2014 e anni precedenti descrivere il comportamento di questa rete e disegnare líntero diagramma di temporizzazione, come sopra specificato).



COMPITO di ARCHITETTURA DEI CALCOLATORI del 17-01-2018 (rev1.2) **MATRICOLA**_____

SOLUZIONE **COGNOME**_____

NOME_____

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)
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	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	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 17-01-2018 (rev1.2) **MATRICOLA**_____

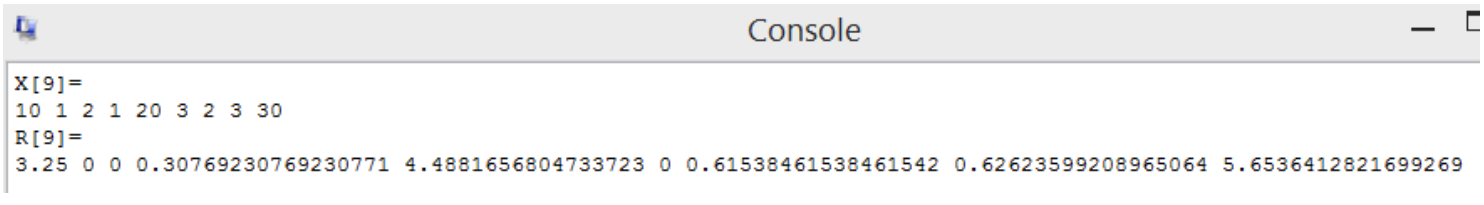
SOLUZIONE **COGNOME**_____

NOME_____

ESERCIZIO 1)

```
.data
A:      .double 10.0,  1.0,   2.0
      .double  1.0, 20.0,   3.0
      .double  2.0,  3.0,  30.0
uno:    .double 1.0
paropen:  .asciiz "["
parclose:  .asciiz "]=\n"
newline:  .asciiz "\n"
spc:    .asciiz " "
namer:    .asciiz "R"
namex:     .asciiz "X"

.text
.globl main

main:
  #________NO CALL FRAME___________
  # R e' una var. temporanea che coincide
  # col valore di ritorno della f. cholesky
  la   $a0, namex
  la   $a1, A
  addi $a2, $0, 9
  jal  printmat
  la   $a0, A
  addi $a1, $0, 3
  jal  cholesky
  la   $a0, namer
  add  $a1, $0, $v0
  addi $a2, $0, 9
  jal  printmat
  addi $v0, $0, 10
  syscall

printmat:
  #________NO CALL FRAME___________
  # i associato a t0
  addi $v0, $0, 4 # stampa 'name'= a0
  syscall
  la   $a0, paropen
  addi $v0, $0, 4
  syscall          # stampa parent. aperta
  add  $a0, $0, $a2
  addi $v0, $0, 1
  syscall      # stampa m
  la   $a0, parclose
  addi $v0, $0, 4
  syscall          # stampa parent. chiusa
  add  $t0, $0, $0  # i=0
pmfor:
  slt   $t1, $t0, $a2  # i<?m
  beq  $t1, $0, pmfinefor
  sll   $t1, $t0, 3  # i*8
  add  $a0, $a1, $t1  # &X+i*8
  lwc1 $f12,0($a0)  # X[i]
  lwc1 $f13,4($a0)
  addi $v0, $0, 3
  syscall        # stampa X[i]
  la   $a0, spc
  addi $v0, $0, 4
  syscall              # stampa 1 spazio
  addi $t0, $t0, 1  # ++i
  j pmfor
pmfinefor:
  la   $a0, newline
  addi $v0, $0, 4
  syscall          # stampa parent. aperta
  jr $ra

mysqrt:
  #________NO CALL FRAME___________#
  mfc1  $t1,  $f12
  mfc1  $t0,  $f13

  andi  $t2,  $t0, 1
  sll    $t2,  $t2, 31 # MS-bit
  sra    $t0,  $t0, 1  # vint>>1
  srl    $t1,  $t1, 1
  or    $t1,  $t1, $t2

  addi    $t2,  $0,  1
  sll    $t2,  $t2, 29
  add    $t0,  $t0, $t2

  addi    $t2,  $0,  1
  sll    $t2,  $t2, 19
  sub    $t0,  $t0, $t2

  mtc1  $t1,  $f12
  mtc1  $t0,  $f13
  jr    $ra

cholesky:
  #________CALL FRAME___________
  # saved variables: s0    4B
  # saved variables: s1=j    4B
  # saved variables: s2=k    4B
  # saved variables: s3=i    4B
  # saved variables: s4    4B
  # ra             4B
  #____________________Totale 8B
  addi   $sp, $sp, -24
  sw      $ra, 0($sp)
  sw      $s0, 4($sp)
  sw      $s1, 8($sp)
  sw      $s2, 12($sp)
  sw      $s3, 16($sp)
  sw      $s4, 20($sp)
  add    $s0, $0, $a0  # salvo a0 in s0

  mult  $a1, $a1    # n*n
  mflo  $a0
  sll    $a0, $a0, 3    # n*n*8
  addi  $v0, $0, 9    # sbrk, v0=&L
  syscall

  add    $s1, $0, $0    # j=0
f1:
  slt    $t9, $s1, $a1  # j<?n
  beq    $t9, $0, finef1
  mtc1  $0, $f0      # s=0.0
  mtc1  $0, $f1
  add   $s2, $0, $0    # k=0
f2:
  slt    $t9, $s2, $s1  # k<?j
  beq    $t9, $0, finef2

  mult  $s1, $a1
  mflo  $t8
  add    $t8, $t8, $s2  # j*n+k
  sll    $t8, $t8, 3    # (j*n+k)*8
  add    $t8, $v0, $t8  # &L[j,k]
  lwc1  $f2, 0($t8)    # L[j,k]
  lwc1  $f3, 4($t8)
  mul.d  $f4, $f2, $f2  # L*L
  add.d  $f0, $f0, $f4  # s+=L*L

  addi  $s2, $s2, 1   # ++k
  j    f2
finef2:

  mult  $s1, $a1
  mflo  $s4
  add    $s4, $s4, $s1  # j*n+j
  sll    $s4, $s4, 3    # (j*n+j)*8
  add    $t8, $s0, $s4  # &A[j,j]
  lwc1  $f4, 0($t8)    # A[j,j]
  lwc1  $f5, 4($t8)
  sub.d  $f12, $f4, $f0  # A[]-s
  jal    mysqrt
  add    $t8, $v0, $s4  # &L[j,j]
  swc1  $f12, 0($t8) # scrivo in L[j,j]
  swc1  $f13, 4($t8)

  addi  $s3, $s2, 1    # i=j+i

f3:
  slt    $t9, $s3, $a1  # i<?n
  beq    $t9, $0, finef3
  mtc1  $0, $f0      # s=0.0
  mtc1  $0, $f1
  add   $s2, $0, $0    # k=0
f4:
  slt    $t9, $s2, $s1  # k<?j
  beq    $t9, $0, finef4

  mult  $s3, $a1
  mflo  $t8
  add    $t8, $t8, $s2  # i*n+k
  sll    $t8, $t8, 3    # (i*n+k)*8
  add    $t8, $v0, $t8  # &L[i,k]
  lwc1  $f6, 0($t8)    # L[i,k]
  lwc1  $f7, 4($t8)

  mult  $s1, $a1
  mflo  $t8
  add    $t8, $t8, $s2  # j*n+k
  sll    $t8, $t8, 3    # (j*n+k)*8
  add    $t8, $v0, $t8  # &L[j,k]
  lwc1  $f8, 0($t8)    # L[j,k]
  lwc1  $f9, 4($t8)

  mul.d  $f4, $f6, $f8   # L[i,k]*L[j,k]
  add.d  $f0, $f0, $f4  # s+=L[]*L[]

  addi  $s2, $s2, 1   # ++k
  j    f4

finef4:
  la    $t8, uno    # 1.0
  lwc1  $f10,0($t8)
  lwc1  $f11,4($t8)
  div.d  $f14,$f10,$f12  # 1.0 / L[j,j]

  mult  $s3, $a1
  mflo  $s4
  add    $s4, $s4, $s1  # i*n+j
  sll    $s4, $s4, 3    # (i*n+j)*8
  add    $t7, $s0, $s4  # &A[i,j]
  lwc1  $f4, 0($t7)    # A[i,j]
  lwc1  $f5, 4($t7)
  sub.d  $f16, $f4, $f0  # A[]-s

  mult  $s1, $a1
  mflo  $t8
  add    $t8, $t8, $s2  # j*n+k
  sll    $t8, $t8, 3    # (j*n+k)*8
  add    $t8, $v0, $t8  # &L[j,k]
  lwc1  $f6, 0($t8)    # L[j,k]
  lwc1  $f7, 4($t8)
  mul.d   $f18, $f14,$f16 # 1/L[]*(A[]-s)

  add    $t8, $v0, $s4  # &L[i,j]
  swc1  $f18, 0($t8)  # scrivo in L[]
  swc1  $f19, 4($t8)

  addi  $s3, $s3, 1    # ++i
  j     f3

finef3:
  addi  $s1, $s1, 1   # ++j
  j     f1

finef1:
  # v0 already contains &L
  lw    $s4, 20($sp)
  lw    $s3, 16($sp)
  lw    $s2, 12($sp)
  lw    $s1, 8($sp)
  lw    $s0, 4($sp)
  lw    $ra, 0($sp)
  addi   $sp, $sp, 24
  jr $ra
```

OUTPUT:



COMPITO di ARCHITETTURA DEI CALCOLATORI del 17-01-2018 (rev1.2) **MATRICOLA**_____

SOLUZIONE **COGNOME**_____

NOME_____

```
ESERCIZIO 2)
A = 2
B = 8
C = 64
RP = FIFO
Thit = 4
Tpen = 40
File: c1180117.sh_001000.din
Read 20 references.
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  124   15   3   3   4  0 [3]:1,0 [3]:0,0  [3]:3,-
=== W  169   21   5   1   1  0 [1]:1,0 [1]:0,0  [1]:5,-
=== R  167   20   5   0   7  0 [0]:1,0 [0]:0,0  [0]:5,-
=== W  245   30   7   2   5  0 [2]:1,0 [2]:0,0  [2]:7,-
=== R  183   22   5   2   7  0 [2]:0,1 [2]:0,0  [2]:7,5
=== W  119   14   3   2   7  0 [2]:1,0 [2]:0,0  [2]:3,5 (out: XM=30 XT=7  XS=2 )
=== R  235   29   7   1   3  0 [1]:0,1 [1]:0,0  [1]:5,7
=== W  163   20   5   0   3  1 [0]:1,0 [0]:1,0  [0]:5,-
=== R  288   36   9   0   0  0 [0]:0,1 [0]:1,0  [0]:5,9
=== W  309   38   9   2   5  0 [2]:0,1 [2]:0,0  [2]:3,9 (out: XM=22 XT=5  XS=2 )
=== R  310   38   9   2   6  1 [2]:0,1 [2]:0,0  [2]:3,9
=== W  308   38   9   2   4  1 [2]:0,1 [2]:0,1  [2]:3,9
=== R  213   26   6   2   5  0 [2]:1,0 [2]:0,1  [2]:6,9 (out: XM=14 XT=3  XS=2 )
=== W  196   24   6   0   4  0 [0]:1,0 [0]:0,0  [0]:6,9 (out: XM=20 XT=5  XS=0 )
=== R  377   47  11   3   1  0 [3]:0,1 [3]:0,0  [3]:3,11
=== W  166   20   5   0   6  0 [0]:0,1 [0]:0,0  [0]:6,5 (out: XM=36 XT=9  XS=0 )
=== R  362   45  11   1   2  0 [1]:1,0 [1]:0,0  [1]:11,7 (out: XM=21 XT=5  XS=1 )
=== W  233   29   7   1   1  1 [1]:1,0 [1]:0,1  [1]:11,7
=== R  163   20   5   0   3  1 [0]:0,1 [0]:0,0  [0]:6,5
=== W  169   21   5   1   1  0 [1]:0,1 [1]:0,0  [1]:11,5 (out: XM=29 XT=7  XS=1 )
-------------------------------------
P1 Nmiss=15   Nhit=5  Nref=20  mrate=0.750000   AMAT=34
```

ESERCIZIO 3)

```
lab: LW  $2, 0($1)      FORMATO I (op,rs,rd,im):        23  1  2  0000
     LW  $4, 0($3)      FORMATO I (op,rs,rd,im):        23  3  4  0000
     SLL $5, $2, 10     FORMATO R (op,rs,rt,rd,sh,fu): 00  2  0  5 0A 00
     SRL $6, $4, 12     FORMATO R (op,rs,rt,rd,sh,fu): 00  4  0  6 0C 02
     JAL fun            FORMATO J (op,im)              03  100009          # indirizzo(fun)=400024 → (400024/4)=100009
     SLT $7, $5, $6     FORMATO R (op,rs,rt,rd,sh,fu): 00  5  6  7 00 2A
     BNE $8, $0, lab    FORMATO I (op,rs,rd,im):        05  7  0  FFF9      # branch di -7 istruzioni
     JR  $31            FORMATO R (op,rs,rt,rd,sh,fu): 00 1F  0  0 00 08
fun: DIV $10, $11       FORMATO R (op,rs,rt,rd,sh,fu): 00  A  B  0 00 1A
     JR  $31            FORMATO R (op,rs,rt,rd,sh,fu): 00 1F  0  0 00 08
```

ESERCIZIO 7)

```
`timescale 1ns/1ps
module TopLevel;
  reg clock; reg reset_;
  wire[3:0] Q;
  always #10 clock<=(!clock);
  initial begin
    $display ("time, \t clock, \t reset_, \t QQQQ");
    $monitor ("%g, \t %b, \t %b, \t %b",
          $time, clock, reset_, Q);
    reset_ = 1'b1; clock = 0;
    #5   reset_ = 1'b0;
    #20  reset_ = 1'b1;
    #600 $finish;
  end
  reg T; initial begin T=1; end
  Ripple_Counter ripc(Q,T,clock,reset_);
  //debug:
  wire q0=ripc.q0,q1=ripc.q1,q2=ripc.q2,q3=ripc.q3;
  wire[3:0] q= {ripc.q3,ripc.q2,ripc.q1,ripc.q0};
endmodule

// Flip-Flop T sensibile al fronte in discesa
module FFTn(q,t,clock,reset_);
  input clock,reset_,t; output q; reg STAR;
  parameter S0=0,S1=1;
  assign q=(STAR==S0)?0:1;
  always @(reset_==0) #1 STAR <= S0;
  always @(negedge clock) if (reset_==1) #3
    casex(STAR)
      S0: STAR <= (t==0)?S0:S1;
      S1: STAR <= (t==1)?S0:S1;
    endcase
endmodule

module Ripple_Counter(Q,T,clock,reset_);
  input clock, reset_,T;
  wire q0,q1,q2,q3;
  reg[3:0] Q1;
  output[3:0] Q; assign Q=Q1;
  FFTn rc0(q0,T,clock,reset_);
  FFTn rc1(q1,T,q0,reset_);
  FFTn rc2(q2,T,q1,reset_);
  FFTn rc3(q3,T,q2,reset_);
  //notare che per non avere stati spuri in uscita
conviene bufferizzare in un registro
  always @(negedge clock) begin Q1[0]<=q0; Q1[1]<=q1;
Q1[2]<=q2; Q1[3]<=q3; end
endmodule
```