COMPITO di ARCHITETTURA DEI CALCOLATORI del 11-04-2018 (rev1.1) **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, 16/17, 17/18": es. N.1+2+3+7.

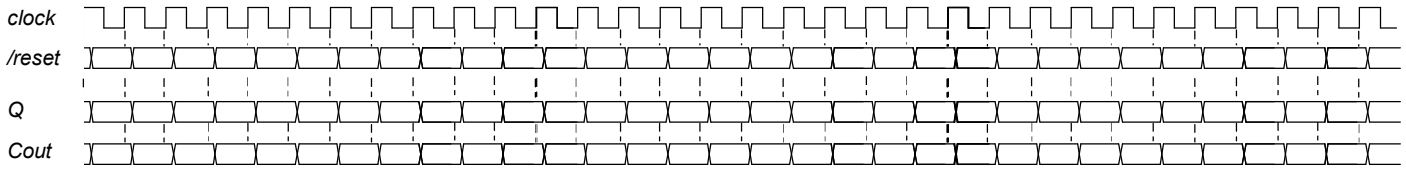
NOTA: per l'esercizio 7 dovranno essere consegnati DUE files: il file del programma VERILOG e il file relativo all'output (screenshot o copy/paste)

1) [19/38] 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).

```
typedef struct node {
    struct node *next;
    int vertex;
}node;

int vi[10]={0,0,0,0,1,2,3,4,5,6};
int vj[10]={1,2,3,4,5,5,6,6,7,7};
node *G[20];
int visited[20];
int n=8;

void DFS(int i) {
    node *p;
    print_int(i);
    print_string(" ");
    p=G[i];
    visited[i]=1;
    while(p!=NULL) {
       i=p->vertex;
       if(!visited[i]) DFS(i);
       p=p->next;
    }
}

void insert(int vi,int vj) {
    node *p,*q;
    q=(node*)sbrk(sizeof(node));
    q->vertex=vj;
    q->next=NULL;
    if(G[vi]==NULL)
        G[vi]=q;
    else {
        p=G[vi];
        while(p->next!=NULL) p=p->next;
        p->next=q;
    }
}

void read_graph() {
    int i;
    for(i=0;i<n;i++) {
        G[i]=NULL;
        for(j=0;j<10;j++) insert(vi[j],vj[j]);
    }
}

int main() {
    int i;
    read_graph();
    for(i=0;i<n;i++) visited[i]=0;
    DFS(0);
    print_string("\n");
    exit(0);
}
```

2) [7/38] Si consideri una cache di dimensione 160B e a 5 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' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 927, 413, 63, 11, 40, 61, 15, 124, 822, 141, 16, 113, 16, 23, 91, 216, 31, 210, 11, 18, 31, 21. 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) [4/38] Rappresentare il numero 5/7 in un formato simile a IEEE-754 singola precisione ma supponendo di avere solo 4 bit per l'esponente anziche' 8 e solo 3 bit per la mantissa anziche' 23 (e un bit per il segno). L'arrotondamento deve essere effettuato al numero piu' vicino rappresentabile e in caso di equidistanza arrotondare al valore pari (round to nearest, ties to even).
4) Non assegnato
5) Non assegnato
6) Non assegnato
7) [8/38] **Realizzare** in Verilog (per studenti 2014 e anni precedenti --> v.nota finale) sia un contatore look-ahead-carry a 4-bit ottenuto collegando l'uscita S (opportunamente memorizzata su flip-flop-D) all'ingresso B di un sommatore look-ahead-carry mentre l'ingresso A e' fissato a 4'b0001. Realizzare la descrizione del circuito in Verilog e 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 e Cout (carry in uscita al contotare) 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 11-04-2018

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 11-04-2018

SOLUZIONE

ESERCIZIO 1

```
.data
sp1: .asciiz " "
nl1: .asciiz "\n"
vi:  .word 0, 0, 0, 0, 1, 2, 3, 4, 5, 6
vj:  .word 1, 2, 3, 4, 5, 5, 6, 6, 7, 7
G:   .space 80
visited: .space 80
n:   .word 8

.text
.globl main

DFS:
  #________CALL FRAME___________
  # saved variables: fp,ra     8B
  # saved variables: s0        4B
  # saved variables: s1        4B
  #____________________Totale 16B
  # p in s0, i in s1
  addi  $sp,$sp,-16
  sw    $fp, 0($sp)
  add   $fp, $sp, $0
  sw    $ra, 4($sp)
  sw    $s0, 8($sp)
  sw    $s1,12($sp)

  add    $s1, $a0, $0 # s1 = i
  addi   $v0, $0, 1   # print_int (a0)
  syscall
  la     $a0, sp1
  addi   $v0, $0, 4   # print_string
  syscall
  la     $t0, G       # &G
  add    $t1, $s1,$s1
  add    $t1, $t1,$t1
  add    $t0, $t0,$t1 # &G + i*4
  lw     $s0, 0($t0)  # p (=G[i])
  addi   $t2, $0,1    # 1
  la     $t0, visited # & visited
  add    $t0, $t0,$t1 # &visited+i*4
  sw     $t2, 0($t0)  # visited[i]=1
dfs_iniwhile:
  beq    $s0,$0, dfs_finewhile
  lw     $a0, 4($s0)  # i=p->vertex
  la     $t0, visited # & visited
  add    $t4, $a0,$a0
  add    $t4, $t4, $t4
  add    $t0, $t0, $t4
  lw     $t3, 0($t0)  # visited[i]
  bne    $t3, $0,dfs_dopoif
  jal    DFS
dfs_dopoif:
  lw     $s0,0($s0)   # p=p->next
  j      dfs_iniwhile
dfs_finewhile:
  lw     $s1,12($sp)
  lw     $s0, 8($sp)
  lw     $ra, 4($sp)
  lw     $fp, 0($sp)
  addi    $sp,$sp,16
  jal $ra

insert:
  #________NO CALL FRAME_________
  # vi(a0) in s0, vj(a1) in s1
  add    $t9,$0,$a0
  addi   $a0,$0,8
  addi   $v0,$0,9
  syscall             # v0= q
  sw     $a1,4($v0)  # q->vertex=vj
  sw     $0, 0($v0)  # q->next=NULL
  la     $t0,G      # &G
  add    $t4,$t9,$t9
  add    $t4,$t4,$t4
  add    $t0,$t0,$t4 # &G+4*vi
  lw     $t1,0($t0)  # G[vi]
  bne    $t1,$0,i_else
  sw     $v0,0($t0)  # G[vi]=q
  j      i_dopoif
i_else:
  # t1 is p=G[vi]
i_while:
  lw     $t2,0($t1)  # p->next
  beq    $t2,$0, i_dopowhile
  add    $t1, $0,$t2 # p=p->next
  j     i_while
i_dopowhile:
  sw     $v0,0($t1)  # p->next=q
i_dopoif:
  jal    $ra

read_graph:
  #________CALL FRAME___________
  # saved variables: fp,ra  8B
  # saved variables:s0,s1,s2 12B
  #____________________Totale 20B
  # i in s0, j in s1, n in s2
  addi   $sp,$sp,-20
  sw     $fp, 0($sp)
  add    $fp, $sp, $0
  sw     $ra, 4($sp)
  sw     $s0, 8($sp)
  sw     $s1,12($sp)
  sw     $s2,16($sp)

  la     $t1, n      # &n
  lw     $s2, 0($t1) # n

  addi   $s0, $s0,0  # i=0
rg_inifor1:
  slt    $t2, $s0,$s2# i<?n
  beq    $t2,$0, rg_finefor1
  la     $t0, G      # &G
  add    $t4, $s0,$s0
  add    $t4, $t4,$t4
  add    $t0, $t0,$t4# &G+4*i
  sw     $0,  0($t0) # G[i]=0

  addi   $s1, $0,0  # j=0
rg_inifor2:
  addi   $t3, $0, 10 # 10
  slt    $t2, $s1,$t3# i<?n
  beq    $t2,$0, rg_finefor2
  la     $t5, vi
  la     $t6, vj
  add    $t4, $s1,$s1
  add    $t4, $t4,$t4
  add    $t5, $t5,$t4# &vi[j]
  add    $t6, $t6,$t4# &vj[j]
  lw     $a0, 0($t5) # vi[j]
  lw     $a1, 0($t6) # vj[j]
  jal    insert

  addi   $s1, $s1,1  # ++j
  j       rg_inifor2
rg_finefor2:

  addi   $s0, $s0,1  # ++i
  j       rg_inifor1
rg_finefor1:

  lw     $s2,16($sp)
  lw     $s1,12($sp)
  lw     $s0, 8($sp)
  lw     $ra, 4($sp)
  lw     $fp, 0($sp)
  addi   $sp,$sp,20
  jal    $ra

#-----------------------------------
main:
    jal    read_graph
  add    $t0, $0, $0  # i=0
  la     $t1, n       # &n
  lw     $t1,0($t1)   # n
  la     $t3, visited # & visited
inifor:
  slt    $t2, $t0, $t1
  beq    $t2, $0, finefor
  add    $t4, $t0, $t0
  add    $t4, $t4, $t4
  add    $t3, $t3, $t4# &t3 + i*4
  sw     $0,0($t3)    # visited[i]
  addi   $t0, $t0, 1  # ++i
  j     inifor
finefor:
  add    $a0, $0, $0  # 1st param = 0
  jal   DFS

  la     $a0, nl1
  addi   $v0, $0, 4  #print_string
  syscall

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

Console

```
0 1 5 7 2 3 6 4
```

ESERCIZIO 2

```
A = 5, B = 8, C = 160, RP = LRU, Thit = 4, Tpen = 40;

=== T X XM XT XS XB H [SET]:USAGE [SET]:MODIF [SET]:TAG
=== R 927 115 28 3 7 0 [3]:4,0,0,0,0 [3]:0,0,0,0,0 [3]:28,-,-,-,-
=== W 413 51 12 3 5 0 [3]:3,4,0,0,0 [3]:0,0,0,0,0 [3]:28,12,-,-,-
=== R 63 7 1 3 7 0 [3]:2,3,4,0,0 [3]:0,0,0,0,0 [3]:28,12,1,-,-
=== W 11 1 0 1 3 0 [1]:4,0,0,0,0 [1]:0,0,0,0,0 [1]:0,-,-,-,-
=== R 40 5 1 1 0 0 [1]:3,4,0,0,0 [1]:0,0,0,0,0 [1]:0,1,-,-,-
=== W 61 7 1 3 5 1 [3]:2,3,4,0,0 [3]:0,0,1,0,0 [3]:28,12,1,-,-
=== R 15 1 0 1 7 1 [1]:4,3,0,0,0 [1]:0,0,0,0,0 [1]:0,1,-,-,-
=== W 124 15 3 3 4 0 [3]:1,2,3,4,0 [3]:0,0,1,0,0 [3]:28,12,1,3,-
=== R 822 102 25 2 6 0 [2]:4,0,0,0,0 [2]:0,0,0,0,0 [2]:25,-,-,-,-
=== W 141 17 4 1 5 0 [1]:3,2,4,0,0 [1]:0,0,0,0,0 [1]:0,1,4,-,-
=== R 16 2 0 2 0 0 [2]:3,4,0,0,0 [2]:0,0,0,0,0 [2]:25,0,-,-,-
=== W 113 14 3 2 1 0 [2]:2,3,4,0,0 [2]:0,0,0,0,0 [2]:25,0,3,-,-
=== R 16 2 0 2 0 1 [2]:2,4,3,0,0 [2]:0,0,0,0,0 [2]:25,0,3,-,-
=== W 23 2 0 2 7 1 [2]:2,4,3,0,0 [2]:0,1,0,0,0 [2]:25,0,3,-,-
=== R 91 11 2 3 3 0 [3]:0,1,2,3,4 [3]:0,0,1,0,0 [3]:28,12,1,3,2
=== W 216 27 6 3 0 0 [3]:4,0,1,2,3 [3]:0,0,1,0,0 [3]:6,12,1,3,2 (out: XM=115 XT=28 XS=3 )
=== R 31 3 0 3 7 0 [3]:3,4,0,1,2 [3]:0,0,1,0,0 [3]:6,0,1,3,2 (out: XM=51 XT=12 XS=3 )
=== W 210 26 6 2 2 0 [2]:1,3,2,4,0 [2]:0,1,0,0,0 [2]:25,0,3,6,-
=== R 11 1 0 1 3 1 [1]:4,2,3,0,0 [1]:0,0,0,0,0 [1]:0,1,4,-,-
=== W 18 2 0 2 2 1 [2]:1,4,2,3,0 [2]:0,1,0,0,0 [2]:25,0,3,6,-
=== R 31 3 0 3 7 1 [3]:3,4,0,1,2 [3]:0,0,1,0,0 [3]:6,0,1,3,2
=== W 21 2 0 2 5 1 [2]:1,4,2,3,0 [2]:0,1,0,0,0 [2]:25,0,3,6,-
-------------------------------------
P1 Nmiss=14 Nhit=8 Nref=22 mrate=0.636364 AMAT=29.4545
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 11-04-2018

SOLUZIONE

ESERCIZIO 3

Il formato con 1 bit di segno, 4 bit di esponente e 3 bit di mantissa e' una proposta per uno standard "quarter-IEEE-754".
Normalizzando 5/7 si ottiene: 10/7*2^-1 ovvero m=10/7, e=-1. Ricaviamo quindi S,M,E.

L' "uno" iniziale non viene rappresentato nel formato quarter-IEEE-754. Essendo m = 10/7=1.$\overline{428571}$ si ha:

M = m-1= 0.$\overline{428571}$

Successivamente si puo' ricavare la rappresentazione binaria di M con 3 bit moltiplicando per 2 e ricavando via via la n-esima cifra piu' significativa:

0. $\overline{428571}$ *2 = 0.$\overline{857142}$ → **0**, 0.$\overline{857142}$ *2 = 1.$\overline{714285}$ → **1**, 0.$\overline{714285}$ *2= 1.$\overline{428571}$ → **1**, ...e questo punto si ripete 0, 1, 1, 0, 1, 1, e cosi' via...
(notare che NON si deve mai troncare il numero: occorre mantenere tutte le cifre periodiche).
Dopo le prime 3 cifre binarie le cifre si ripetono (ribadendo la periodicita' del numero), quindi e'facilmente predicibile il resto delle cifre binarie della mantissa. Inoltre la quarta cifra della mantissa corrisponde ad uno 0 quindi il piu'vicino numero rappresentabile in IEEE-754 singola precisione e'quello che si ottiene troncando cosi' com'e' il valore ottenuto per M. Quindi abbiamo ottenuto:
011 ... ovvero M = 011

Per l'esponente, ricordando che nel caso di singola precisione il valore della polarizzazione e' 7:

E = e+7 = -1+7=6 ovvero 0110

Inoltre per il segno S = 0

Quindi la rappresentazione cercata e':

0b 0011 0011

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; #650 T=0; end
  LAC_Carry_Counter lcc(Q,T,clock,reset_);
  //debug:
  wire q0=lcc.Q[0], q1=lcc.Q[1], q2=lcc.Q[2], q3=lcc.Q[3];
  wire[3:0] q= {lcc.Q[3],lcc.Q[2],lcc.Q[1],lcc.Q[0]};
  wire cout=lcc.cout;
endmodule

// LAC adder
module lac_full_adder(a, b, c, g, p, s);
  input a, b, c; output g, p, s;
  assign g = a & b;
  assign p = a ^ b;
  assign s = a ^ (b ^ c);
endmodule

// LAC Adder 4-bit
module lac_adder_4bit(a, b, cin, s, cout);
    input [3:0] a, b; input cin; output cout; output [3:0] s;
    wire [4:0] c; wire [3:0] g, p;

    assign c[0] = cin;
    assign cout = c[4];

    lac_full_adder add0(a[0], b[0], c[0], g[0], p[0], s[0]);
    assign c[1] = g[0] | (p[0] & c[0]);

    lac_full_adder add1(a[1], b[1], c[1], g[1], p[1], s[1]);
    assign c[2] = g[1] | (p[1] & g[0]) | (p[1] & p[0] & c[0]);

    lac_full_adder add2(a[2], b[2], c[2], g[2], p[2], s[2]);
    assign c[3] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) |
(p[2] & p[1] & p[0] & c[0]);

    lac_full_adder add3(a[3], b[3], c[3], g[3], p[3], s[3]);
    assign c[4] = g[3] | (p[3]&g[2]) | (p[3]& p[2]&g[1]) |
(p[3]&p[2]&p[1]&g[0]) | (p[3]&p[2]&p[1]&p[0]&c[0]);
endmodule

module LAC_Carry_Counter(Q,T,clock,reset_);
  input clock, reset_,T; output[3:0] Q;
  reg[3:0] A, S; reg cin;
  wire[3:0] S1;
  wire cout;
  assign Q=S;
  always @(reset_==1) begin if (T==0) A <= 4'b0000; else A <=
4'b0001; end
  always @(reset_==0) #1 begin S <= 4'b0000; cin <= 1'b0; end
  lac_adder_4bit lacadd(A, S, cin, S1, cout);
  always @(negedge clock) if (reset_==1) #3 begin S<=S1; end
endmodule
```