COMPITO di ARCHITETTURA DEI CALCOLATORI del 08-02-2019

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
→ NO FOGLI PERSONALI, NO TELEFONI, SMARTPHONE, ETC

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) [14/30] 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).

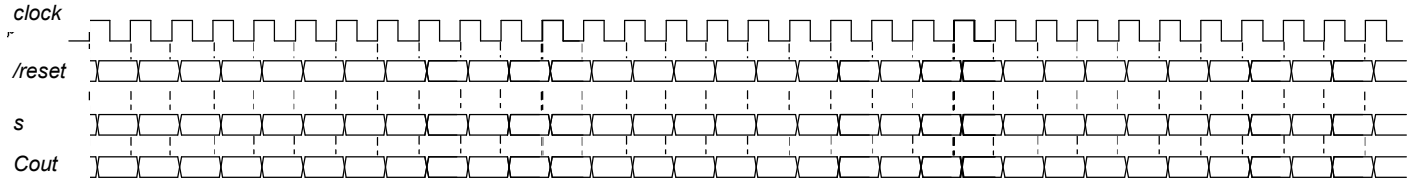
```
unsigned int reverseBits(unsigned int num)
{
    unsigned int  NO_OF_BITS = 32; //sizeof(num) * 8;
    unsigned int reverse_num = 0, i, temp;

    for (i = 0; i < NO_OF_BITS; i++)
    {
        temp = (num & (1 << i));
        if(temp)
            reverse_num |= (1 << ((NO_OF_BITS - 1) - i));
    }

    return reverse_num;
}
```

```
int main()
{
    unsigned int x = 2;
    unsigned int y = reverseBits(x);
    print_int(y);
    print_string("\n");
    print_int(reverseBits(y));
}
```

2) [5/30] Si consideri una cache di dimensione 256B 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: 126, 112, 162, 110, 139, 160, 114, 223, 221, 240, 215, 312, 415, 522, 690, 715, 830, 911, 1010, 1117, 1230, 122. 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) [2/30] Rappresentare la tabella di verità di un encoder a 4 ingressi e 2 uscite e sintetizzarlo tramite mappe di Karnaugh.

7) [9/30] **Realizzare** in Verilog un sommatore a 4-bit di tipo carry-look-ahead. Il testbench e' dato. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità riportando i segnali clock, /reset, uscita S e Cout (carry in uscita al sommatore) per la durata complessiva (45ns). 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



Testbench:

```
`timescale 1ns/1ps
module cla4_testbench;
   reg reset_; initial begin reset_=0; #2 reset_=1; #600; $stop; end
   reg clock;  initial clock<=0;  always #5 clock<=(!clock);
   reg[3:0] a, b; reg cin; wire cout; wire [3:0] s;
   initial begin cin <=0;
      @(posedge clock); a<=7; b<=7; @(posedge clock); a<=7; b<=9;
      @(posedge clock); a<=9; b<=2; @(posedge clock); a<=7; b<=3;
      $finish;
   end
   cla_adder_4bit CLA4(a,b,cin,s,cout);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 08-02-2019

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	**ift 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	**ift 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**	`Syscalls`	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

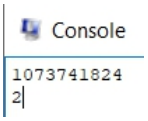
Service Name	Service Num. ($v0)	INPUT Arguments	OUTPUT Arguments
print_int	1	$a0=integer to print	---
print_float	2	$f12=float to print	---
print_double	3	($f12,$f13)=double to print	---
print_string	4	$a0=address of ASCIIZ string to print	---
read_int	5	---	$v0=integer
read_float	6	---	$f0=float
read_double	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 08-02-2019

SOLUZIONE

ESERCIZIO 1

```
.data
RTN: .asciiz "\n"

.globl main
.text
#-----------------------------------
#
a0=num,t0=NO_OF_BITS,t1=reverse_num,
# t2=i,t3=temp
reversebits:
   addiu $t0, $0, 32 # t0=32
   addu  $t1, $0, $0 # t1=0
   addu  $t2, $0, 0  # i=0
for_ini:
   sltu  $t4,$t2,$t0 # i<?NO_OF_BITS
   beq   $t4,$0,for_end
   addiu $t4,$0,1    # t4=1
   sllv  $t3,$t4,$t2 # 1<<i
   and   $t3,$a0,$t3 # num & (.)

   beq   $t3,$0,if_end
   addu  $t4,$t0,$0  # t4=NO_OF_BITS
   addiu $t4,$t4,-1  # t4=t4-1
   subu  $t4,$t4,$t2 # t4=(t4-i)
   addiu $t5,$0,1    # t5=1
   sllv  $t5,$t5,$t4 # 1<<(.)
   or    $t1,$t1,$t5
if_end:
   addiu $t2, $t2, 1 # i++
   j for_ini
for_end:
   addu  $v0,$t1,0
   jr  $ra

#-----------------------------------
main: #s0=x, s1=y
#-----------------------------------
# PROLOGO
   addi  $sp,$sp,-16 # alloco frame
   sw      $ra,12($sp) # salvo OLD-
ra
   sw      $fp, 8($sp) # salvo OLD-
fp
   add   $fp,$0,$sp  # NUOVO fp
   sw    $s1, 4($fp) # salvo s1
   sw    $s0, 0($fp) # salvo s0

   addiu $s0, $0, 2  #s0 = x = 2
   addu  $a0, $s0, $0#a0 = x
   jal   reversebits
   addu  $s1, $v0, $0#s1 = y

   addu  $a0, $s1, $0
   addiu $v0, $0, 1  #serv.1
   syscall
   la     $a0, RTN    # carriage
return
   addi  $v0, $0, 4  #serv.4
   syscall

   addu  $a0, $s1, $0 # y
   jal   reversebits

   addu  $a0, $v0, $0
   addi  $v0, $0, 1  #serv.1
   syscall

#-----------------------------------
FINEFUN: # EPILOGO
   lw     $s0, 0($fp) # ripristina
s0
   lw     $s1, 4($fp) # ripristina
s1
   lw    $ra, 8($fp) # RIPRISTINA ra
   lw    $fp,12($fp) # RIPRISTINA fp
   addi  $sp,$sp,16  # DEALLOCO
FRAME
   addi  $v0,$0,10   # serv.10
   syscall           # exit
```



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=256/16/4=4, XM=X/B, XS=XM%S, XT=XM/S:

```
A=4, B=16, C=256, RP=FIFO, Thit=4, Tpen=40, 22 references:
=== T    X    XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  126    7   1   3  14  0 [3]:3,0,0,0 [3]:0,0,0,0  [3]:1,-,-,-
=== W  112    7   1   3   0  1 [3]:3,0,0,0 [3]:1,0,0,0  [3]:1,-,-,-
=== R  162   10   2   2   2  0 [2]:3,0,0,0 [2]:0,0,0,0  [2]:2,-,-,-
=== W  110    6   1   2  14  0 [2]:2,3,0,0 [2]:0,0,0,0  [2]:2,1,-,-
=== R  139    8   2   0  11  0 [0]:3,0,0,0 [0]:0,0,0,0  [0]:2,-,-,-
=== W  160   10   2   2   0  1 [2]:2,3,0,0 [2]:1,0,0,0  [2]:2,1,-,-
=== R  114    7   1   3   2  1 [3]:3,0,0,0 [3]:1,0,0,0  [3]:1,-,-,-
=== W  223   13   3   1  15  0 [1]:3,0,0,0 [1]:0,0,0,0  [1]:3,-,-,-
=== R  221   13   3   1  13  1 [1]:3,0,0,0 [1]:0,0,0,0  [1]:3,-,-,-
=== W  240   15   3   3   0  0 [3]:2,3,0,0 [3]:1,0,0,0  [3]:1,3,-,-
=== R  215   13   3   1   7  1 [1]:3,0,0,0 [1]:0,0,0,0  [1]:3,-,-,-
=== W  312   19   4   3   8  0 [3]:1,2,3,0 [3]:1,0,0,0  [3]:1,3,4,-
=== R  415   25   6   1  15  0 [1]:2,3,0,0 [1]:0,0,0,0  [1]:3,6,-,-
=== W  522   32   8   0  10  0 [0]:2,3,0,0 [0]:0,0,0,0  [0]:2,8,-,-
=== R  690   43  10   3   2  0 [3]:0,1,2,3 [3]:1,0,0,0  [3]:1,3,4,10
=== W  715   44  11   0  11  0 [0]:1,2,3,0 [0]:0,0,0,0  [0]:2,8,11,-
=== R  830   51  12   3  14  0 [3]:3,0,1,2 [3]:0,0,0,0  [3]:12,3,4,10
=== W  911   56  14   0  15  0 [0]:0,1,2,3 [0]:0,0,0,0  [0]:2,8,11,14
=== R 1010   63  15   3   2  0 [3]:2,3,0,1 [3]:0,0,0,0  [3]:12,15,4,10
=== W 1117   69  17   1  13  0 [1]:1,2,3,0 [1]:0,0,0,0  [1]:3,6,17,-
=== R 1230   76  19   0  14  0 [0]:3,0,1,2 [0]:0,0,0,0  [0]:19,8,11,14
=== W  122    7   1   3  10  0 [3]:1,2,3,0 [3]:0,0,0,0  [3]:12,15,1,10
-------------------------------------
P1 Nmiss=17   Nhit=5  Nref=22  mrate=0.772727   AMAT=th+mrate*tpen=34.9091
```

CONTENTUTI dei 4 SET al termine:

LISTA BLOCCHI USCENTI:

```
(out: XM=7  XT=1  XS=3 )
(out: XM=15 XT=3  XS=3 )
(out: XM=8  XT=2  XS=0 )
(out: XM=19 XT=4  XS=3 )
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 08-02-2019

SOLUZIONE

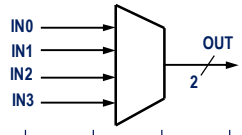
ESERCIZIO 3

Encoder da 4 a 2

- L'encoder effettua l'operazione inversa del decoder
- Posto un 1 sul k-esimo dei 2^n ingressi (gli altri sono posti a 0):le n uscite producono i bit corrispondenti alla rappresentazione in base due di k

TABELLA DI VERITÀ (COMPLETA):

IN3	IN2	IN1	IN0		OUT1	OUT0
0	0	0	1		0	0
0	0	1	0		0	1
0	1	0	0		1	0
1	0	0	0		1	1
0	0	0	0		X	X
0	0	1	1		X	X
0	1	0	1		X	X
0	1	1	0		X	X
0	1	1	1		X	X
1	0	0	1		X	X
1	0	1	0		X	X
1	0	1	1		X	X
1	1	0	0		X	X
1	1	0	1		X	X
1	1	1	0		X	X
1	1	1	1		X	X



IN3,IN2 \ IN1,IN0	00	01	11	10
00	X	0	X	1
01	0	X	X	X
11	X	1	X	X
10	1	X	X	X

OUT1

IN3,IN2 \ IN1,IN0	00	01	11	10
00	X	0	X	0
01	1	X	X	X
11	X	1	X	X
10	1	X	X	X

OUT0

OUT0=IN1+IN3 OUT1=IN2+IN3

ESERCIZIO 7

Codice VERILOG:

```
module cla_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

module cla_adder_4bit(a,b,cin,s,cout);
  input [3:0]a,b; input  cin;
  output [3:0]s;  output cout;
  wire [4:0] c;
  wire [3:0] g, p;
  assign c[0] = cin;
  assign cout = c[4];
  cla_full_adder add0(a[0],b[0],c[0],  g[0],p[0],s[0]);
  assign c[1] = g[0] | (p[0] & c[0]);
  cla_full_adder add1(a[1],b[1],c[1],  g[1],p[1],s[1]);
  // assign c[2] = g[1] | (p[1] & c[1]);
  assign c[2] = g[1] | (p[1] & g[0]) | (p[1] & p[0] & c[0]);
  cla_full_adder add2(a[2],b[2],c[2],g[2],p[2],s[2]);
  // assign c[3] = g[2] | (p[2] & c[2]);
  assign c[3] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0])
  | (p[2] & p[1] & p[0] & c[0]);
  cla_full_adder add3(a[3],b[3],c[3],g[3],p[3],s[3]);
  // assign c[4] = g[3] | (p[3] & c[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
```

Diagramma temporale: