COMPITO di ARCHITETTURA DEI CALCOLATORI del 31-01-2022

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/WATCH, ETC

NOTA: dovrà essere consegnato l'elaborato dell'es.1 come file <COGNOME>.s e quelli dell'es. 4 come files <COGNOME>.v e <COGNOME>.png

1) [12/30] Trovare il codice assembly RISC-V corrispondente al seguente micro-benchmark (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), rispettando le convenzioni di uso dei registri dell'assembly (riportate qua sotto, per riferimento).

```
float *X, *L;

void hilb1(int n) {
    int i, j, l; float f;
    X = (float *)sbrk(n*n*sizeof(float));
    L = (float *)sbrk(n*n*sizeof(float));
    for (i = 0; i < n; ++i) {
        f = (float) 1 / (i + 1);
        for (j = 0; j < n; ++j) {
            l = i * n + j;
            X[l] = n * n * (n * n / (float)(i + j + 1));
            L[l] = 0;
            if (i>=j) {
                f = f * ((float) (i - j + 1) / (i + j + 1));
                L[l] = n * n * sqrt(2*j+1) * f;
            }
        }
    }
}
```

```
int main() {
    float s = 0; int k;
    hilb1(4);
    for (k = 0; k < 16; ++k) s += L[k] + X[k];
    print_float(s);
    exit(0);
}
```

RISCV Instructions (RV64IMFD) v210622

Instruction coding (hexadecimal) opcode+funct3+{funct7,imm}	Instruction	Example	Register operation	Meaning (** instructions available only in RV64, i.e. 64-bit case)
33+0+00/3b+0+00	add	add/addw x5,x6,x7	x5 ← x6 + x7	Add two operands; exception possible (addw**)
33+0+20/3b+0+20	subtract	sub/subw x5,x6,x7	x5 ← x6 – x7	Subtracts two operands; exception possible (subw**)
13+0+imm/1b+0+imm	add immediate	addi/addiw x5,x6,100	x5 ← x6 + 100	Add a constant ; exception possible (addiw**)
33+0+01/3b+0+01	multiply	mul/mulw x5,x6, x7	x5 ← x6 * x7	(signed/word) Lower 64 bits of 128-bits product (mulw**)
33+01+01	multiply high	mulh x5,x6,x7	x5 ← x6 * x7	Higher 64bits of 128-bits product
33+4+01/3b+4+01	division	div/divw x5,x6,x7	x5 ← x6/x7	(signed/word) division (divw**)
33+6+01/3b+6+01	reminder	rem/remw x5,x6,x7	x5 ← x6 % x7	Reminder of the division (remw**)
33+2+0/33+3+0	set on less than	slt/sltu x5,x6,x7	if (x6 < x7) x5 ← 1; else x5 ← 0	(signed/unsigned) compare x6 and x7 (less than)
13+2+imm/13+3+imm	set on less than immediate	slti/sltiu x5,x6,100	if (x6 < 100) x5← 1; else x5 ← 0	(signed/unsigned) compare x6 and 100 (less than)
33+7+0/33+6+0/33+4+0	and / or / xor	and/or/xor x5,x6,x7	x5← x6&x7 / x6|x7 / x6^ x7	Logical AND/OR/XOR
13+7+imm/13+6+imm/13+4+imm	and /or / xor immediate	andi/ori/xori x5,x6,100	x5 ← x6&100 / x6|100 / x6^100	Logical AND/OR/XOR register, constant
33+1+0/3b+1+0	shift left logical	sll/sllw x5,x6,x7	x5 ← x6 << x7	Shift left by register (sllw**)
13+1+imm/1b+1+imm	shift left logical immediate	slli/slliw x5,x6,10	x5 ← x6 << 10	Shift left by the immediate value (slliw**)
33+5+0/3b+5+0	shift right logical	srl/srlw x5,x6,x7	x5 ← x6 >> x7	Shift right by register (srlw**)
13+5+imm/1b+5+imm	shift right logical immediate	srli/srliw x5,x6,10	x5 ← x6 >> 10	Shift left by immediate value (srliw**)
33+5+20/3b+5+20	shift right arithmetic	sra/sraw x5,x6,x7	x5 ← x6 >> x7 (arith.)	Shift right by register (sign is preserved) (sraw**)
13+5+imm/1b+5+imm	shift right arithmetic immediate	srai/sraiw x5,x6,10	x5 ← x6 >> 10 (arith.)	Shift right by immediate value (sraiw**)
03+3+imm/03+2+imm/03+0+imm	load dword / word / byte	ld/lw/lb x5,100(x6)	x5 ← MEM[x6+100]	Data from memory to register
03+6+imm/03+4+imm	load word / byte unsigned	lwu/bu x5,100(x6)	x5 ← MEM[x6+100]	Data from mem. To reg.; no sign extension (lwu**)
23+3+imm/23+2+imm/23+0+imm	store dword / word / byte	sd/sw/sb x5,100(x6)	MEM[x6+100] ← x5	Data from register to memory (sw**)
37+imm[31:12] (no funct3)	load upper immediate	lui x5,0x12345	x5 ← 0x1234'5000	Load most significant 20 bits
PSEUDOINSTRUCTION	load address	la x5,var	x5 ← &var (PSEUDO INST.) load address of 'var' in x5	REAL: lui x5,H20(&var);ori x5, L12(&var) INST. (H20=high 20 bit of &var; L12=low 12 bits of &var)
6f+imm[31:12](rd=0) 63+0+imm[11:0](rs1=rs2=0)	jump/branch	j/b label	PC+=off (off=PC-&label) (PS.INST.)	REAL INST.: jal x0,offset/beq x0,x0,offset
6f+0+imm[31:12](rd=1,no funct3)	jump and link (offset)	jal label	x1←(PC+4);PC+=offset (PS. INST.)	REAL INST.: jal x1,offset (offset=PC-&label)
67+0+imm (rd=0,rs1=1)	return from procedure	ret	PC←x1 (PSEUDO INST.)	REAL INST.: jalr x0,0(x1)
67+0+imm	jump and link register	jalr x1, 100(x5)	x1 ← (PC + 4);PC=x5+100	Procedure return; indirect call
63+0+(imm÷2)/63+1+(imm÷2)	branch on equal / not-equal	beq/bne x5,x6,100	if (x5 = =/!= x6) PC=PC+100	Equal / Not-equal test; PC relative branch
73+0+0 (rs1=0,rs2=0,rd=0)	ecall	ecall	SEPC←PC;PC←STVEC;save PL/IE;PL=1;IE=0	Call OS (service number in a7); PL= privilege lev; IE=int.en.
73+0+8 (rs1=0,rs2=2,rd=0)	sret	sret	PC←SEPC; restore PL/IE	Exit supervisor mode; PL= privilege lev; IE=int.en.
PSEUDOINSTRUCTION	move	mv x5,x6	x5 ← x6 (PSEUDO INST.)	REAL INST.: add x5,x0,x6
PSEUDOINSTRUCTION	load immediate	li x5,100	x5 ← 100 (PSEUDO INST.)	REAL INST.: addi x5,x0,100
PSEUDOINSTRUCTION	no operation (nop)	nop	do nothing (PSEUDO INST.)	REAL INST.: addi x0,x0,0
53+0+{0,1}/53+0+{4,5}	floating point add/sub	fadd/fsub.{s,d} f0,f1,f2	f0←f1+f2 / f0←f1-f2	Single or double precision add / subtract
53+0+{8,9}/53+0+{c,d}	floating point multiplication/division	fmul/fdiv.{s,d} f0,f1,f2	f0←f1*f2 / f0←f1/f2	Single or double precision multiplication / division
PSEUDOINSTRUCTION	floating point move between f-regs	fmv.{s,d} f0,f1	f0←f1 (PSEUDO INST.)	REAL INST.: fsgnj.{s,d} f0,f1,f1
PSEUDOINSTRUCTION	floating point negate	fneg.{s,d} f0,f1	f0← – (f1) (PSEUDO INST.)	REAL INST.: fsgnjn.{s,d} f0,f1,f1
PSEUDOINSTRUCTION	floating point absolute value	fabs.{s,d} f0,f1	f0← | f1 | (PSEUDO INST.)	REAL INST.: fsgnjx.{s,d} f0,f1,f1
53+0/1/2+{50,51}	floating point compare	fle/flt/feq.{s,d} x5,f0,f1	x5← (f0<f1)	Single and double: compare f0 and f1 <=,<,==
53+0+{70,71} (rs2=0)	move between x (integer) and f regs	fmv.x.{s,d} x5,f0	x5←f0 (no conversion)	Copy (no conversion)
53+0+{78,79} (rs2=0)	move between f and x regs	fmv.{s,d}.x f0,x5	f0←x5 (no conversion)	Copy (no conversion)
7+2+imm/27+2+imm	load/store floating point (32bit)	flw/fsw f0,0(x5)	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
7+3+imm/27+3+imm	load/store floating point (64bit)	fld/fsd f0,0(x5)	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
53+7+21(rs2=0)/53+7+20(rs2=0)	convert to/from double from/to single	fcvt.d.s/fcvt.s.d f0,f1	f0← (double)f1 / f0← (single)f1	Type conversion
53+7+{60,61} (rs2=0)	convert to integer from {single,double}	fcvt.w.{s,d} x5,f0	x5← (int)f0	Type conversion
53+7+{68,69} (rs2=0)	convert to {single,double} from integer	fcvt.{s,d}.w f0,x5	f0← ({single,double})x5	Type conversion
53+0+{2c,2d} (rs2=0)	square root	fsqrt.{s,d} f0,f1	f0← square root of f1	Single or double square root
53+0/1/2+{10,11}	sign injection	fsgnj/jn/jx.{s,d} f0,f1,f2	f0←sgn(f2)|f1|/–sgn(f2)|f1|/sgn(f2)f1	Extract the mantissa and exp. from f1 and sign from f2

Register Usage

Register	ABI Name	Usage
x10-x11	a0-a1	arguments and results
x9, x18-x27	s1, s2-s11	Saved
x5-7, x28-x31	t0-t2, t3-t6	Temporaries
x12-x17	a2-a7	Arguments

Register	ABI Name	Usage
x0	zero	The constant value 0
x8, x2	s0/fp, sp	frame pointer, stack pointer
x1, x3	ra, gp	return address, global pointer
x4	tp	thread pointer

Register	ABI Name	Usage
f10-f11	fa0-fa1	Argument and Return values
f8-f9, f18-f27	fs0-fs1, fs2-fs11	Saved registers
f0 – f7, f28-f31	ft0-ft7, ft8-ft11	Temporaries registers
f12-17	fa2-fa7	Function arguments

System calls

Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Args
print_int	1	a0=integer to print	---
print_float	2	fa0=float to print	---
print_double	3	fa0=double to print	---
print_string	4	a0=address of ASCIIZ string to print	---
read_int	5	---	a0=integer

Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	fa0=float
read_double	7	---	fa0=double
read_string	8	a0=address of input buffer, a1=max chars to read	---
sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
exit	10	---	---

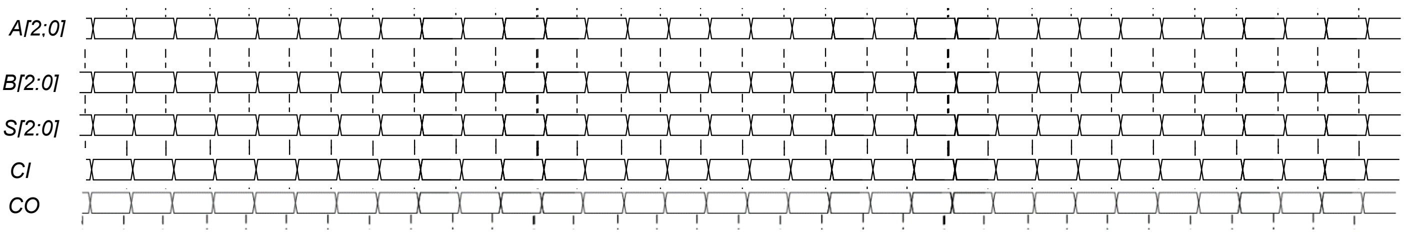
COMPITO di ARCHITETTURA DEI CALCOLATORI del 31-01-2022

2) [5/30] 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' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 2023, 2139, 2427, 2439, 2428, 2439, 2433, 2454, 2425, 2454, 2422, 2454, 2439, 2126, 2454, 2424, 2554, 2629, 2754, 2828. 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) [3/30] Spiegare perché è problematico creare chip di grosse dimensioni ricavando la relazione fra costo di un chip (C_{DIE}), l'area del chip (A_{DIE}) e l'area del wafer (A_{WAFER}).

4) [10/30] Descrivere e sintetizzare in Verilog una rete combinatoria che realizzi la sottrazione fra due interi in complemento a due su 3 bit utilizzando moduli da 1 bit collegati in riporto seriale. Gli stimoli di ingresso sono dati dal seguente modulo Verilog Testbench. Devono essere realizzati i due moduli indicati FULL_SUB e RC_SUBTRACTOR corrispondenti rispettivamente ad un modulo che effettua la sottrazione su 1 bit e al modulo che realizza la sottrazione su 3 bit.

Tracciare il diagramma di temporizzazione [5/10 punti] come verifica della correttezza dell'unità. Nota: si puo' 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. Modello del diagramma temporale da tracciare:




```
module Testbench;
   reg[2:0] A, B;
   wire[2:0] S;
   reg CI;
   wire CO;
   initial begin CI<=1;
      A<=6; B<=1; #20
      A<=5; B<=3; #20
      A<=4; B<=6; #20
      A<=7; B<=7; #20
      $finish;
   end
   RC_SUBTRACTOR rcs(A, B, CI,  S,CO);
endmodule

module FULL_SUB(a, b, ci,  s,c);
   …
endmodule

module RC_SUBTRACTOR(A, B, CI,  S,CO);
   …
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 31-01-2022

SOLUZIONE

ESERCIZIO 1

```
.data
X: .space 4
L: .space 4
.text
.globl main
hilb1:
        mv      a6,a0       # n
        mulw    t0,a6,a6    # n*n
        slli    a0,t0,2     # * sizeof(float)
        li        a7,9      # sbrk
        ecall
        mv      a1,a0       # a1=&L
        la      t6,L        # &L
        sw      a1,0(t6)    # store &L
        slli    a0,t0,2     # * sizeof(float)
        li        a7,9        # sbrk
        ecall               # a0=&X
        la      t6,X        # &X
        sw      a0,0(t6)    # store &X
        #fori
        add     t0,x0,x0    # i=0
h1_fori_start:
        slt     t6,t0,a6    # i<?n
        beq     t6,x0,h1_fori_end
        # f = (float) 1 / (i + 1)
        addi    t6,t0,1     # i+1
        fcvt.s.w ft0,t6     # (float)(.)
        li      t6,1        # 1
        fcvt.s.w ft1,t6     # (float)1
        fdiv.s  ft9,ft1,ft0# f=1/(i+1)
        # forj
        add     t1,x0,x0    # j=0
h1_forj_start:
        slt     t6,t1,a6    # j<?n
        beq     t6,x0,h1_forj_end
        # prepara &X[i,j]
        mul         t6,t0,a6# i*n
        add         t6,t6,t1# l=i*n+j
        slli    t6,t6,2     # 4*(i*n+j)
        add     a5,t6,a0    # a5=&X[i,j]
        # prepara &L[i,j]
        add         a4,t6,a1# a4=&L[i,j]
        # prepara valore da assegnare
        add     t6,t0,t1    # i+j
        addi    t6,t6,1     # i+j+1
        fcvt.s.w ft0,t6     #(float)(.)
        mul     t6,a6,a6    # n*n
        fcvt.s.w ft1,t6     # (float)(.)
        fdiv.s  ft2,ft1,ft0# (n*n)/(i+j+1)
        fmul.s  ft3,ft1,ft2# (n*n)*(.)
        # assegna valori
        fsw     ft3,0(a5)   # X[i,j]=(.)
        sw      x0,0(a4)    # L[i,j]=0
        # if (i>=j) {
        slt         t6,t1,a6    # j<?l
        beq         t6,x0,h1_end_if
h1_if1:
        sub     t6,t0,t1    # i-j
        addi    t6,t6,1     # i-j+1
        fcvt.s.w ft4,t6     # float(.)
        fdiv.s  ft5,ft4,ft0# (.)/(i+j+1)
        fmul.s  ft9,ft9,ft5# f*(.)
        add     t6,t1,t1    # 2*j
        addi    t6,t6,1     # 2*j+1
        fcvt.s.w ft6,t6     # (float)(.)
        fsqrt.s ft6,ft6     # sqrt(.)
        fmul.s  ft6,ft1,ft6# n*n*(.)
        fmul.s  ft6,ft6,ft9# (.)*f
        fsw     ft6,0(a4)   # L[i,j]=(.)
h1_end_if:
        addi         t1,t1,1# ++j
        b          h1_forj_start
h1_forj_end:
        addi         t0,t0,1# ++i
        b       h1_fori_start
h1_fori_end:
        ret

main:
        addi    sp,sp,-4
        fsw     fs0,0(sp)   # save fs0
        fmv.s.x fs0,zero    # s=fs0 = 0
        li      a0,4        # a0=4
        call    hilb1

        la      a0,L        # &L
        la      a1,X        # &X
        lw      a4,0(a0)    # *L
        lw      a5,0(a1)    # *X
        li      t0,0        # t0=k = 0
main_for_ini:
        slti    t1, t0, 16
        beq     t1, x0, main_for_end

        flw     ft0,0(a4)   # L[k}
        flw     ft1,0(a5)   # X[k]
        fadd.s  fs0,fs0,ft0
        fadd.s  fs0,fs0,ft1
        addi    a4,a4,4     # next elem
        addi    a5,a5,4     # next elem

        addi    t0,t0,1     # ++k
        b       main_for_ini
main_for_end:
        fmv.s   fa0,fs0     # fa0=s
        li        a7,2      # print_float
        ecall
        li        a7,10     # exit
        ecall
        flw     fs0,0(sp)
        addi    sp,sp,4
```

Messages | Run I/O

```
1349.5162
-- program is finished running (0) --
```

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=64/8/2,     XM=X/B,    XS=XM%S,    XT=XM/S.
A=2, B=8, C=64, RP=LRU, Thit=4, Tpen=40, 20 references:
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R 2023  252  63   0   7  0 [0]:1,0 [0]:0,0  [0]:63,-
=== W 2139  267  66   3   3  0 [3]:1,0 [3]:0,0  [3]:66,-
=== R 2427  303  75   3   3  0 [3]:0,1 [3]:0,0  [3]:66,75
=== W 2439  304  76   0   7  0 [0]:0,1 [0]:0,0  [0]:63,76
=== R 2428  303  75   3   4  1 [3]:0,1 [3]:0,0  [3]:66,75
=== W 2439  304  76   0   7  1 [0]:0,1 [0]:0,1  [0]:63,76
=== R 2433  304  76   0   1  1 [0]:0,1 [0]:0,1  [0]:63,76
=== W 2454  306  76   2   6  0 [2]:1,0 [2]:0,0  [2]:76,-
=== R 2425  303  75   3   1  1 [3]:0,1 [3]:0,0  [3]:66,75
=== W 2454  306  76   2   6  1 [2]:1,0 [2]:1,0  [2]:76,-
=== R 2422  302  75   2   6  0 [2]:0,1 [2]:1,0  [2]:76,75
=== W 2454  306  76   2   6  1 [2]:1,0 [2]:1,0  [2]:76,75
=== R 2439  304  76   0   7  1 [0]:0,1 [0]:0,1  [0]:63,76
=== W 2126  265  66   1   6  0 [1]:1,0 [1]:0,0  [1]:66,-
=== R 2454  306  76   2   6  1 [2]:1,0 [2]:1,0  [2]:76,75
=== W 2424  303  75   3   0  1 [3]:0,1 [3]:0,1  [3]:66,75
=== R 2554  319  79   3   2  0 [3]:1,0 [3]:0,1  [3]:79,75 (out: XM=267 XT=66 XS=3 )
=== W 2629  328  82   0   5  0 [0]:1,0 [0]:0,1  [0]:82,76 (out: XM=252 XT=63 XS=0 )
=== R 2754  344  86   0   2  0 [0]:0,1 [0]:0,0  [0]:82,86 (out: XM=304 XT=76 XS=0 )
=== W 2828  353  88   1   4  0 [1]:0,1 [1]:0,0  [1]:66,88
--------------------------------------
P1 Nmiss=11   Nhit=9  Nref=20  mrate=0.550000   AMAT=th+mrate*tpen=26
```

CONTENUTI dei SET al termine

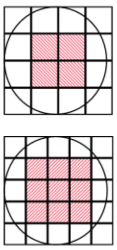
LISTA BLOCCHI USCENTI:

ESERCIZIO 3

Perche' non si fanno chip "grossi"

$$C_{DIE} = \frac{C_{WAFER}}{N_{DIE} \cdot Y_{WAFER}}$$

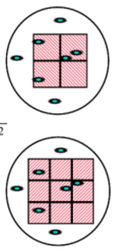
- C_{DIE} = Costo del 'die' (del 'chip')
- C_{WAFER} = Costo del wafer
- N_{DIE} = Numero di 'die' in un wafer
- Y_{WAFER} = 'Yield' o Resa del wafer (numero 'die' per unita' di sup.)
- D_{WAFER} = Diametro del wafer
- A_{DIE} = Area del 'die'
- A_{WAFER} = Area del wafer=$\pi \cdot (D_{WAFER}/2)^2$
- N_{TEST} = Numero di 'die' usati per test
- F = Difetti per unita' di superfice



N. di chip "teorici" sul wafer

N. di chip sul bordo del wafer

N. di chip usati per test "distruttivi"



$$N_{DIE} = \frac{\pi \cdot \left(D_{WAFER}/2\right)^2}{A_{DIE}} - \frac{\pi \cdot D_{WAFER}}{\sqrt{2 \cdot A_{DIE}}} - N_{TEST} \cong \frac{A_{WAFER}}{A_{DIE}}$$

Inoltre: $Y_{WAFER} = \dfrac{1}{1 + (F \cdot A_{DIE}/2)^2}$

$$\Rightarrow C_{DIE} = \frac{C_{WAFER} \cdot A_{DIE} \cdot \left[1 + (F \cdot A_{DIE}/2)^2\right]}{A_{WAFER}} \propto \frac{A_{DIE}^3}{A_{WAFER}}$$

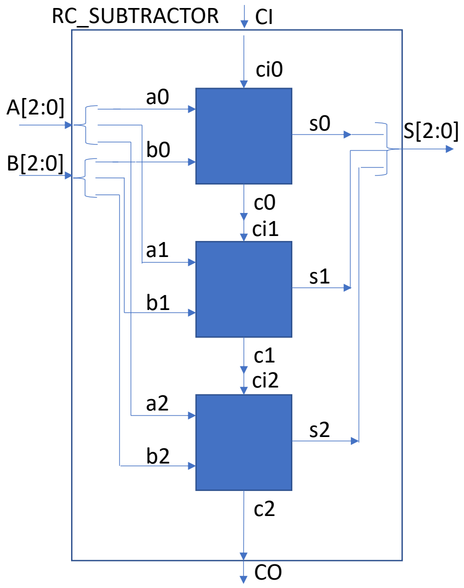
Il costo di un chip e' proporzionale a circa il cubo della sua area!

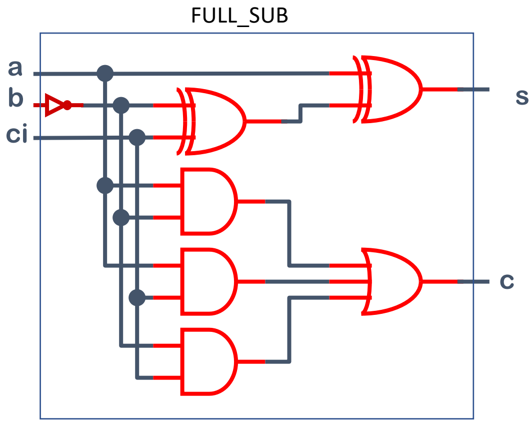
COMPITO di ARCHITETTURA DEI CALCOLATORI del 31-01-2022

SOLUZIONE

ESERCIZIO 4

E’ conveniente rappresentarsi uno schema dei moduli da realizzare:





Possibile codice Verilog dei moduli da realizzare:

```
module FULL_SUB(a, b, ci,  s,c);
   input  a, b, ci;
   output s, c;
   assign s = a ^ ((~b) ^ ci);
   assign c = (a & (~b)) | (ci & (a | (~b)));
endmodule

module RC_SUBTRACTOR(A, B, CI,  S,CO);
   input[2:0] A;
   input[2:0] B;
   input CI;
   output[2:0] S;
   output CO;
   wire[2:0] c;
   assign CO=c[2];
   FULL_SUB s0(A[0],B[0],CI,    S[0],c[0]);
   FULL_SUB s1(A[1],B[1],c[0],  S[1],c[1]);
   FULL_SUB s2(A[2],B[2],c[1],  S[2],c[2]);
Endmodule
```

Diagramma di Temporizzazione: