COMPITO di ARCHITETTURA DEI CALCOLATORI del 25-07-2023

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) [10/30] 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).

```
int hcf(int x, int y) {
  int t;
  while (y != 0) {
    t = x % y; x = y; y = t;
  }
  return x;
}

int relprime(int x, int y) {
  return hcf(x, y) == 1;
}

int euler(int n) {
  int length = 0, i;
  for (i = 1; i < n; i++)
    if (relprime(n, i)) length++;
  return length;
}

int sumTotient(int lower, int upper) {
  int sum = 0, i;
  for (i = lower; i <= upper; i++)
    sum = sum + euler(i);
  return sum;
}

int main() {
  int n = sumTotient(1,30);
  print_int(n);
  exit(0);
}
```

Nota: 'int' è un intero a 64 bit.

RISCV Instructions (RV64IMFD) v221117

Instruction coding (hexadecimal) funct7/imm	funct3	opcode	Instruction	Example	Register operation	Meaning (** instructions available only in RV64, i.e. 64-bit case)
00	0	33/3b	add	add/addw x5,x6,x7	x5 ← x6 + x7	Add two operands; exception possible (addw**)
20	0	33/3b	subtract	sub/subw x5,x6,x7	x5 ← x6 – x7	Subtracts two operands; exception possible (subw**)
imm	0	13/1b	add immediate	addi/addiw x5,x6,100	x5 ← x6 + 100	Add a constant ; exception possible (addiw**)
01	0	33/3b	multiply	mul/mulw x5,x6, x7	x5 ← x6 * x7	(signed/word) Lower 64 bits of 128-bits product (mulw**)
01	1	33	multiply high	mulh x5,x6,x7	x5 ← x6 * x7	Higher 64bits of 128-bits product
01	4	33/3b	division	div/divw x5,x6,x7	x5 ← x6/x7	(signed/word) division (divw**)
01	6	33/3b	reminder	rem/remw x5,x6,x7	x5 ← x6 % x7	Reminder of the division (remw**)
00	2/3	33	set on less than	slt/sltu x5,x6,x7	if (x6 < x7) x5 ← 1; else x5 ← 0	signed compare x6 and x7 (less than)
imm	2/3	13	set on less than immediate	slti/sltiu x5,x6,100	if (x6 < 100) x5← 1; else x5 ← 0	unsigned compare x6 and 100 (less than)
00	7/6/4	33	and / or / xor	and/or/xor x5,x6,x7	x5← x6&x7 / x6|x7 / x6^ x7	Logical AND/OR/XOR register operand
imm	7/6/4	13	and /or / xor immediate	andi/ori/xori x5,x6,100	x5 ← x6&100 / x6|100 / x6^100	Logical AND/OR/XOR constant operand
0	1	33/3b	shift left logical	sll/sllw x5,x6,x7	x5 ← x6 << x7	Shift left by register (sllw**)
imm	1	13/1b	shift left logical immediate	slli/slliw x5,x6,10	x5 ← x6 << 10	Shift left by the immediate value (slliw**)
0	5	33/3b	shift right logical	srl/srlw x5,x6,x7	x5 ← x6 >> x7	Shift right by register (srlw**)
imm	5	13/1b	shift right logical immediate	srli/srliw x5,x6,10	x5 ← x6 >> 10	Shift left by immediate value (srliw**)
20	5	33/3b	shift right arithmetic	sra/sraw x5,x6,x7	x5 ← x6 >> x7 (arith.)	Shift right by register (sign is preserved) (sraw**)
imm	5	13/1b	shift right arithmetic immediate	srai/sraiw x5,x6,10	x5 ← x6 >> 10 (arith.)	Shift right by immediate value (sraiw**)
imm	3/2/0	03	load dword / word / byte	ld/lw/lb x5,100(x6)	x5 ← MEM[x6+100]	Data from memory to register
imm	6/4	03	load word / byte unsigned	lwu/lbu x5,100(x6)	x5 ← MEM[x6+100]	Data from mem. To reg.; no sign extension (lwu**)
imm	3/2	23	store dword / word / byte	sd/sw/sb x5,100(x6)	MEM[x6+100] ← x5	Data from register to memory (sw**)
imm[31:12]	-	37	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 INST.: lui x5,H20(&var);ori x5, L12(&var) (H20=high 20 bits of &var; L12=low 12 bits of &var)
imm[31:12] (rd=0) imm[11:0] (rs1=rs2=0)	- 0	6f/63	jump/branch	j/b label	PC+=off (off=PC-&label) (PS.INST.)	REAL INST.: jal x0,offset/beq x0,x0,offset
imm[31:12] (rd=1)	-	6f	jump and link (offset)	jal label	x1←(PC+4); PC+=offset (PS. INST.)	REAL INST.: jal x1,offset (offset=PC-&label)
Imm (rd=0,rs=1)	0	67	return from procedure	ret	PC←x1 (PSEUDO INST.)	REAL INST.: jalr x0,0(x1)
imm	0	67	jump and link register	jalr x1, 100(x5)	x1 ← (PC + 4); PC=x5+100	Procedure return; indirect call
imm÷2	0/1	63	branch on equal / not-equal	beq/bne x5,x6,100	if (x5 = =/!= x6) PC=PC+100	Equal / Not-equal test; PC relative branch
00 (rs1=0,rs2=0,rd=0)	0	73	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.
08 (rs1=0,rs2=2,rd=0)	0	73	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
{0,1} / {4,5}	0	53	floating point add/sub	fadd/fsub.{s,d} f0,f1,f2	f0←f1+f2 / f0←f1-f2	Single or double precision add / subtract
{8,9} / {c,d}	0	53	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
{50,51}	0/1/2	53	floating point compare	fle/flt/feq.{s,d} x5,f0,f1	x5← (f0<f1)	Single and double: compare f0 and f1 <=,<,==
{70,71} (rs2=0)	0	53	move between x (integer) and f regs	fmv.x.{s,d} x5,f0	x5←f0 (no conversion)	Copy (no conversion)
{78,79} (rs2=0)	0	53	move between f and x regs	fmv.{s,d}.x f0,x5	f0←x5 (no conversion)	Copy (no conversion)
imm	2	7	load/store floating point (32bit)	flw/fsw f0,0(x5)	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
imm	3	7	load/store floating point (64bit)	fld/fsd f0,0(x5)	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
21/20 (rs2=0)	7	53	convert to/from double from/to single	fcvt.d.s/fcvt.s.d f0,f1	f0← (double)f1 / f0← (single)f1	Type conversion
{60,61} (rs2=0)	7	53	convert to integer from {single,double}	fcvt.w.{s,d} x5,f0	x5← (int)f0	Type conversion
{68,69} (rs2=0)	7	53	convert to {single,double} from integer	fcvt.{s,d}.w f0,x5	f0← ({single,double})x5	Type conversion
{2c,2d} (rs2=0)	0	53	square root	fsqrt.{s,d} f0,f1	f0← square root of f1	Single or double square root
{10,11}	0/1/2	53	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

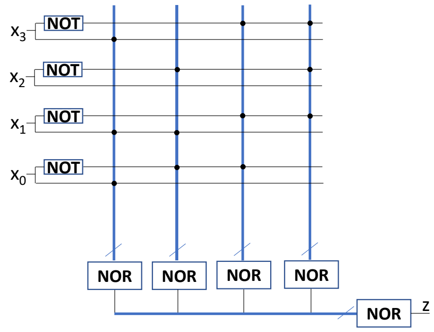
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	fa0=float
read_double	7	---	fa0=double
read_string	8	a0=address of input buffer, a1=max chars to read	---
sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
exit	10	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 25-07-2023

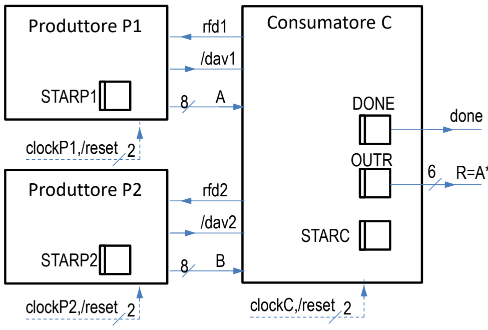
2) [5/30] Calcolare il tempo di esecuzione (TCPU) del seguente frammento di codice, ipotizzando che vengano eseguiti su un processore RISC-V (ideale) con frequenza di clock pari a 1 GHz, assumendo i seguenti valori per il CPI di ciascuna categoria di istruzioni: aritmetico-logiche-salti 1, branch 5, load-store 10 cicli di clock (cc):

```
main:    addi t1,x0,100
loop:    mul  t2,t1,t1
         xor  a0,t2,t1
         jal  funz1
         sd   a0,0(t2)
         addi t0,t0,1
         bne  t0,t1,loop
funz1:   ld   a0,0(a0)
         addi a0,a0,123
         ret
```

3) [4/30] Data la seguente rete combinatoria: i) disegnare la mappa di Karnaugh; ii) inserendo in tale mappa dei non-specificato (simbolo 'X') in corrispondenza degli ingressi $x_3\, x_2\, \overline{x_1}\, x_0$ $x_3\, \overline{x_2}\, x_1\, x_0$, ricavare un'equazione booleana in forma "somma di prodotti" che descriva la nuova mappa in modo da usare sottocubi di dimensione maggiore possibile:

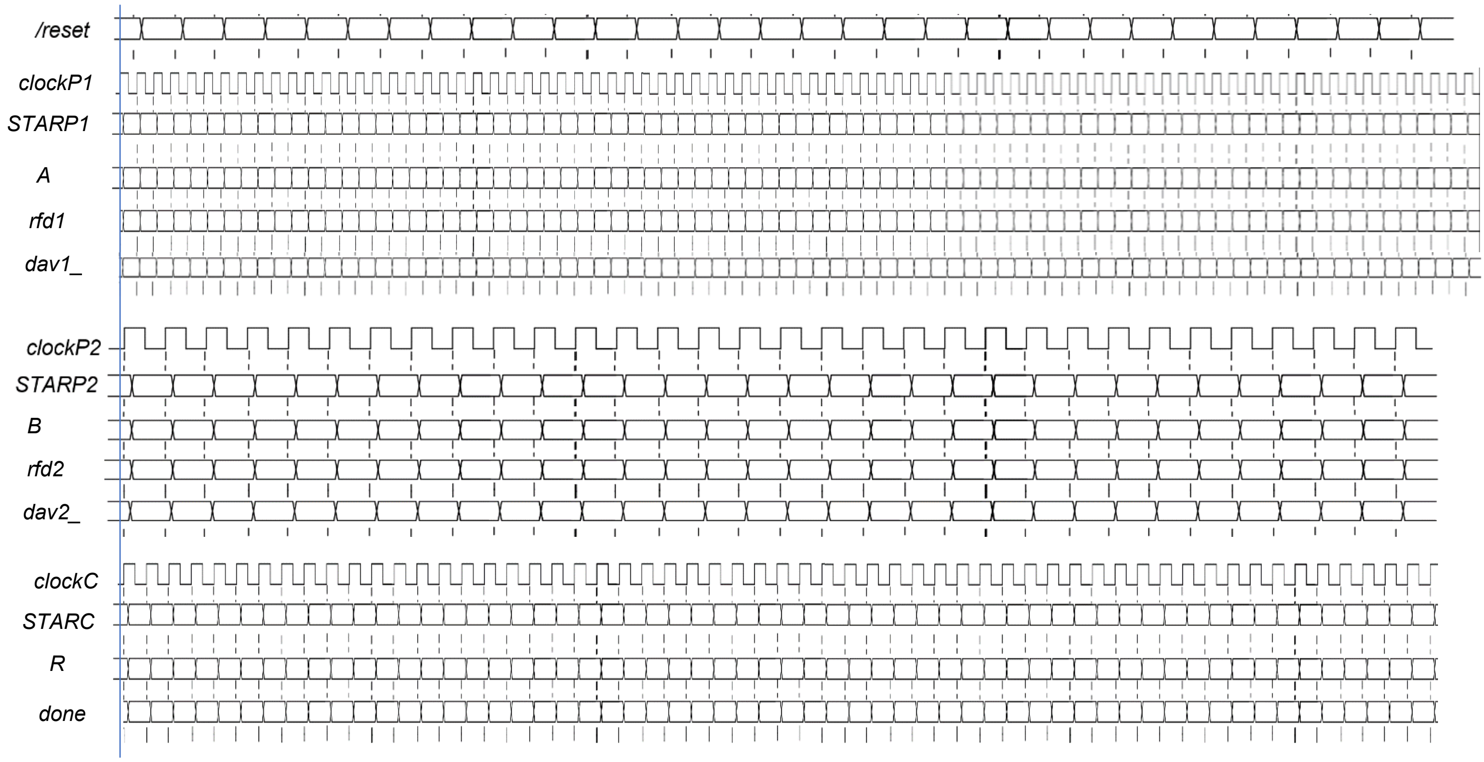


4) [11/30] Descrivere e sintetizzare in Verilog il modulo **C** di figura che funziona nel seguente modo: richiede 2 interi a 8 bit rispettivamente da due produttori **P1** e **P2** attraverso l'usuale protocollo produttore-consumatori, gestendo in modo asincrono i segnali **rfd** e **/dav** di ciascuno (v. figura).
Una volta che i due interi sono acquisiti nei registri interni A e B ne viene effettuato il prodotto e viene presentato in uscita il risultato R=A*B per 5 cicli di clock di **C**; il segnale "done" resta alto per i suddetti 5 cicli per indicare la validità del dato di uscita. Il segnale di reset è comune a tutti e tre i moduli e ognuno di essi lavora in maniera localmente sincronizzata con periodo di clock: clockP1=2ns, clockP2=5ns, clockC=3ns. **Tracciare il diagramma di temporizzazione (punti 5/11)** come verifica della correttezza del modulo realizzato.




```
module testbench;
  reg [7:0] seed1,seed2; initial begin seed1=13; seed2=17; end
  reg reset_; initial begin reset_=0;#5 reset_=1;#200;$stop; end
  reg clockP1; initial clockP1 =0; always #1 clockP1<=(!clockP1)
  wire[1:0] STARP1=P1.STAR;
  wire[7:0] A; wire rfd1, dav1_;
  reg clockP2; initial clockP2 =0; always #2.5 clockP2 <=(!clock
  wire[1:0] STARP2=P2.STAR;
  wire[7:0] B; wire rfd2, dav2_;
  reg clockC; initial clockC =0; always #1.5 clockC <=(!clockC);
  wire[1:0] STARC=C.STAR;
  wire[7:0] R; wire done;
  PROD P1(seed1,rfd1,clockP1,reset_,     dav1_,A);
  PROD P2(seed2,rfd2,clockP2,reset_,     dav2_,B);
  CONS C(dav1_,A,dav2_,B,clockC,reset_,  rfd1,rfd2,done,R);
endmodule
```

```
module PROD(seed,rfd,clock,reset_,     dav_,Z);
  input[7:0] seed; output[7:0] Z;
  input rfd,clock,reset_; output dav_;
  reg DAV_; assign dav_=DAV_;
  reg[7:0] N; assign Z=N;
  reg[1:0] STAR; parameter S0=0, S1=1, S2=2;
  reg t;

  always @(reset_==0) begin DAV_<=1; STAR<=S0; N=seed; end
  always @(posedge clock) if (reset_==1) #0.1
    casex (STAR)
      S0: begin DAV_=1; STAR<=(rfd==1)?S1:S0; end
      S1: begin DAV_=0; //generate a pseudo-random number via LFSR
          t = N[0] ^ N[1] ^ N[3] ^N[4]; N = {N[6:0], t};
          STAR<=S2; end
      S2: begin STAR<=(rfd==1)?S2:S0;end
    endcase
endmodule
```



COMPITO di ARCHITETTURA DEI CALCOLATORI del 25-07-2023

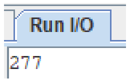
SOLUZIONE

ESERCIZIO 1

```
 .text
#-------------------------------------------
hcf:
# a0: x, a1: y
hcf_start:
   beq   a1,zero,hcf_end# y==?0 --> fine
   rem   t0,a0,a1      # t= x % y
   mv    a0,a1         # x=y
   mv    a1,t0         # y=t
   b       hcf_start
hcf_end:
   ret

#-------------------------------------------
relprime:
# a0: x, a1: y
   addi sp,sp,-8 # allocate frame
   sd   ra,0(sp)
   call hcf       # hcf(x,y)
   mv   t0,a0
   li   a0,1
   beq  t0,a0,rp_end
   li   a0,0
rp_end:
   ld   ra,0(sp)
   addi sp,sp,8  # deallocate frame
   ret

#-------------------------------------------
euler:
# a0: n
# returns: a0: length
   addi sp,sp,-32 # allocate frame
   sd   ra,24(sp)
   sd   s0,16(sp)  # s0: length
   sd   s1,8(sp)   # s1: i
   sd   s2,0(sp)  # s2: n
```

```
   li   s0,0       # length=0
   li   s1,1       # i=1
   mv   s2,a0      # save n
e_for_start:
   beq  s1,s2,e_for_end
   mv   a0,s2      # 1st parm: n
   mv   a1,s1      # 2nd parm: i
   call relprime
   beq  a0,x0,e_if_end
   addi s0,s0,1   # length++
e_if_end:
   addi s1,s1,1   # i++
   b e_for_start
e_for_end:
   mv   a0,s0      # length
   ld  ra,24(sp)
   ld  s0,16(sp)
   ld  s1,8(sp)
   ld  s2,0(sp)
   addi sp,sp,32  # deallocate frame
   ret

#-------------------------------------------
sumTotient:
# a0: lower, a1: upper
   addi    sp,sp,-32 # allocate frame
   sd    ra,24(sp)
   sd    s0,16(sp)   # i
   sd    s1,8(sp)    # sum
   sd    s2,0(sp)    # upper

   li    s1,0        # sum=0
   mv       s0,a0    # i=lower
   mv       s2,a1    # save upper
```

```
st_for_start:
   slt      t0,s2,s0   # i >? upper
   bne    t0,x0,st_for_end #true-->end

   mv    a0,s0        # 1st param: i
   call    euler
   add    s1,a0,s1    # sum += ...
   addi     s0,s0,1   # i++
   b st_for_start
st_for_end:

   mv    a0,s1
   ld    ra,24(sp)
   ld    s0,16(sp)
   ld    s1,8(sp)
   ld    s2,0(sp)
   addi     sp,sp,32    # deallocate frame
   ret

.globl main
#-------------------------------------------
main:
   li    a1,30
   li    a0,1
   call    sumTotient
   li    a7,1
   ecall
   li    a7,10
   ecall
```



ESERCIZIO 2

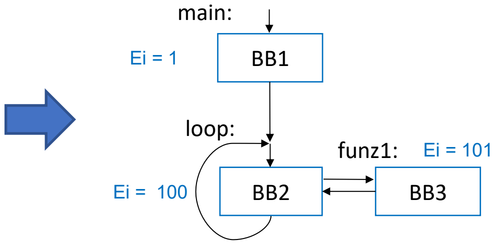
Il codice in esame è composto da 3 Basic-Block (BB) come qua illustrato e si può calcolare il tempo di esecuzione usando l'equazione delle prestazioni:

$$T_{CPU} = N_{CPU} \cdot \overline{CPI} \cdot T_C = T_C \cdot \sum_{i=1}^{N_{BB}} E_i \cdot \left(\sum_{T=1}^{ALJ,B,LS} N_{CPU,T} \cdot CPI_T \right)_i$$

essendo:

$T_C = 1\ GHz$ $N_{BB} = numero\ di\ basic\ blocks = 3$	$N_{CPU,T} = numero\ di\ istruzioni\ della\ categoria\ T\ (ALJ, B\ o\ LS)$ $CPI_T = CPI\ della\ categoria\ T\ (ALJ, B\ o\ LS)$ $E_i = numero\ di\ volte\ che\ un\ dato\ BB_{i-esimo}$ *è eseguito*

```
main:   addi t1,x0,100       BB1
loop:   mul  t2,t1,t1
        xor  a0,t2,t1
        jal  funz1
        sd   a0,0(t2)        BB2
        addi t0,t0,1
        bne  t0,t1,loop
funz1:  ld   a0,0(a0)
        addi a0,a0,123       BB3
        ret
```



Quindi:

BBi	$N_{CPU,ALJ}$ ($CPI_{ALJ} = 1$)	$N_{CPU,B}$ ($CPI_B = 5$)	$N_{CPU,LS}$ ($CPI_{LS} = 10$)	$(N_{CPU,T} \times CPI_T)_i$	E_i	$C_{CPU,i} = (N_{CPU,T} \times CPIT_T)_i * E_i$
BB1	1	0	0	1	1	1 cc
BB2	4	1	1	19	100	1900 cc
BB3	2	0	1	12	101	1212 cc
					CCPU (in cicli di clock)	**3113 cc**

ovvero:

$T_{CPU} = 10^{-9} \cdot 3113 =$
$= 3.113\ \mu s$

ESERCIZIO 3

La rete combinatoria si sintetizza con: $z = \overline{\overline{(x_3 + x_1 + x_0)} + \overline{(\overline{x_2} + x_1 + \overline{x_0})} + \overline{(\overline{x_3} + \overline{x_1} + \overline{x_0})} + \overline{(\overline{x_3} + \overline{x_2} + \overline{x_1})}}$ ovvero $\bar{z} = (\overline{x_3}\,\overline{x_1}\,\overline{x_0}) + (x_2\overline{x_1}x_0) + (x_3x_1x_0) + (x_3x_2x_1)$ →disegniamo la mappa di Karnaugh osservando che dove la precedente espressione a destra vale 1 dovrò invece riportare uno 0 in quanto sto calcolando $\bar{z}$

$x_1x_0 \backslash x_3x_2$	00	01	11	10
00	0	0	1	1
01	1	0	0	1
11	1	1	0	0
10	1	1	0	1

← mappa di Karnaugh per z.

ovvero, inserendo i non specificati in corrisponenza degli ingressi:

$x_3\,x_2\,\overline{x_1}\,x_0 \quad x_3\,\overline{x_2}\,x_1\,x_0$ →

$x_1x_0 \backslash x_3x_2$	00	01	11	10
00	0	0	1	1
01	1	0	X	1
11	1	1	0	X
10	1	1	0	1

E usando sottocubi di dimensione maggiore possibile: $z = x_3\,\overline{x_1} + x_3\,\overline{x_2} + \overline{x_3}\,x_1 + \overline{x_2}\,x_0$

COMPITO di ARCHITETTURA DEI CALCOLATORI del 25-07-2023

SOLUZIONE

ESERCIZIO 4

Questa è una possibile soluzione del modulo "Consumatore C".

Codice Verilog del modulo da realizzare:

```
module CONS(dav1_,A,dav2_,B,clock,reset_,  rfd1,rfd2,done,R);
  output[7:0] R; input[7:0] A, B;
  input dav1_,dav2_,clock, reset_; output rfd1,rfd2,done;
  reg[7:0] OUTR; assign R=OUTR;

  reg RFD1,RFD2,DONE; assign rfd1=RFD1, rfd2=RFD2, done=DONE;
  reg[1:0] STAR; parameter S0=0,S1=1,S2=2;
  reg[2:0] COUNT;

  always @(reset_==0) begin STAR<=S0; RFD1<=0; RFD2<=0; DONE<=0; OUTR<=0; end
  always @(posedge clock) if (reset_==1) #0.1
    casex (STAR)
      S0: begin RFD1<=1; RFD2<=1; COUNT<=0; STAR<=(dav1_==0 & dav2_==0)?S1:S0; end
      S1: begin OUTR<=A*B; RFD1<=0; RFD2<=0; DONE<=1; COUNT=COUNT+1; STAR=(COUNT<5)?S1:S2; end
      S2: begin DONE<=0; STAR<=(dav1_==0 | dav2_==0)?S2:S0; end
    endcase
endmodule
```

Diagramma di Temporizzazione: