COMPITO di ARCHITETTURA DEI CALCOLATORI del 28-02-2023

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**
NOTA2: per il recupero della prima prova in itinere svolgere gli esercizi 3 e 4; per il recupero della seconda, esercizi 1 e 2.

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).

```
void swap(char *v[], int i, int j) {
   char *temp;
   temp = v[i]; v[i] = v[j];  v[j] = temp;
}
```

Nota: 'int' è un intero a 64 bit.

```
void qsort1(char *v[], int left, int right) {
   int i, last;
   if (left >= right) return;
   swap(v, left, (left+right)/2);
   last = left;
   for (i = left+1; i <= right; i++)
      if (*v[i] < *v[left]) swap(v, ++last, i);
   swap(v, left, last);
   qsort1(v, left, last-1);
   qsort1(v, last+1, right);
}
```

```
int main () {
   int k;
   char w[] = "siena";
   char *p[] = { w, w+1, w+2, w+3, w+4 };
   char buf[2] = " ";
   qsort1(p, 0, 4);
   for (k = 0; k < 5; ++k) { buf[0] = *p[k]; print_string(buf); }
   exit (0);
}
```

RISCV Instructions (RV64IMFD) **v221117**

Instruction coding (hexadecimal)			Instruction	Example	Register operation	Meaning (** instructions available only in RV64, i.e. 64-bit case)
funct7/imm	funct3	opcode				
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: lui x5,H20(&var);ori x5, L12(&var) INST.` (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|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 28-02-2023

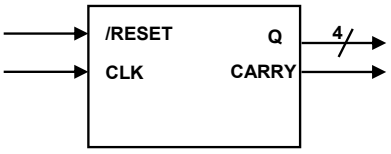
SOLUZIONE

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' 4 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: 126, 120, 127, 125, 184, 120, 215, 143, 287, 308, 290, 298, 212, 195, 357, 146, 342, 213, 143, 149. 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.

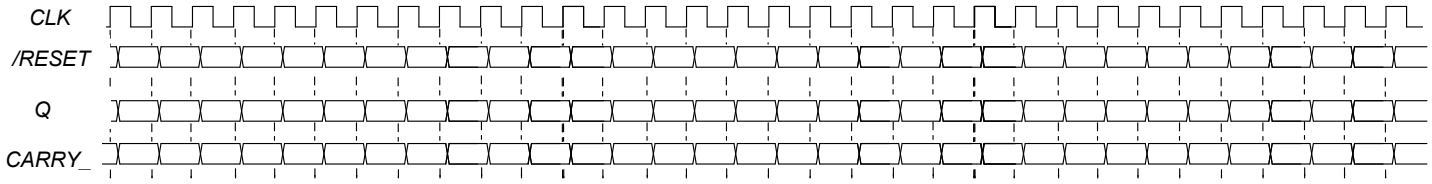
3) [4/30] Dato il full adder di figura (A e B sono i due bit da sommare e C_{IN} il riporto in ingresso; S è il risultato della somma, G è il segnale di generate e P il segnale di propagate) sintetizzare la rete logica di S, G e P in forma di somma di prodotti, ovvero utilizzando due soli livelli di logica in termini di porte AND e OR. Si può scrivere la soluzione esprimendo per S, G e P le rispettive equazioni booleane.

4) **[11/30] Realizzare** in Verilog un contatore verso l'alto ad una cifra decimale (cioè che conta 0, 1, …, 9, 0, …) che viene incrementato ad ogni colpo di clock CLK e basato su Flip-Flop T (che commutano sul fronte in salita del clock). Il conteggio comparirà sull'uscita Q. Al raggiungimento del valore 0 l'uscita CARRY passerà ad 1 (altrimenti vale 0). Il conteggio continuerà poi ciclicamente. Gli stimoli di ingresso sono dati dal seguente modulo Verilog Testbench. Viene inoltre fornito il codice del FF-Tp.
Suggerimento: costruire la seguente tabella e derivare le relative reti mancanti.

Stato attuale	Stato successivo	Ingressi del FF-Tp	CARRY
q3 q2 q1 q0	q3' q2' q1' q0'	t3 t2 t1 t0	C
0 0 0 0	0 0 0 1	0 0 0 1	1
…	…	…	…



Tracciare il diagramma di temporizzazione [punti 5/30 degli 11 di questo esercizio] come verifica della correttezza dell'unità riportando i segnali CLK, /RESET, uscita Q e uscita CARRY per la durata complessiva.
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.




```
module TopLevel;
  reg CLK; reg RESET_; always #10 CLK<=(!CLK);
  wire[3:0] Q; wire OUT;
  initial begin
    RESET_ = 1'b1; CLK = 0;
    #2 RESET_ = 1'b0; #5 RESET_ = 1'b1; #320 $finish;
  end
  MyCounter mc(Q,OUT,  CLK,RESET_);
  //debug:
  wire q0=mc.q0, q1=mc.q1, q2=mc.q2, q3=mc.q3;
endmodule
```

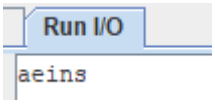
```
// Flip-Flop T sensibile al fronte in salita
module FFTp(q,qbar,clock,reset_,  t);
  input clock, reset_;
  input t;
  output q,qbar;
  reg STAR;
  parameter S0=0,S1=1;
  assign q=(STAR==S0)?0:1, qbar=(STAR==S0)?1:0;
  always @(reset_==0) #1 STAR <= S0;
  always @(posedge clock) if (reset_==1) #1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

COMPITO di ARCHITETTURA DEI CALCOLATORI del 28-02-2023

SOLUZIONE

ESERCIZIO 1

```
.data
w: .asciz "siena"
buf: .asciz " "

.text
.globl main
swap:
# a0=v, a1=i, a2=j
  slli a1,a1,3  # i byte offset
  slli a2,a2,3  # j byte offset
  add t0,a0,a1 # &v[i]
  add t1,a0,a2 # &v[j]
  ld  t2,0(t0) # v[i]
  ld  t3,0(t1) # v[j]
  sd  t3,0(t0) # v[i]=v[j]
  sd  t2,0(t1) # v[j]=v[i]
ret
#-----------------------------
qsort1:
  # a0=v, a1=left, a2=right
  addi sp,sp,-48 #
  sd   ra, 0(sp)
  sd   s0, 8(sp)
  sd   s1,16(sp)
  sd   s2,24(sp)
  sd   s3,32(sp) #last
  sd   s4,40(sp) #i
  mv   s0,a0
  mv   s1,a1
  mv   s2,a2
  slt  t0,a1,a2  # left<?right
  beq  t0,x0,q_end#false: end

  # a0=v e' gia' a posto
  # a1=left e' gia' a posto
  add  t0,a1,a2 #left+right
  srai a2,t0,1  #(.)/2
  call swap
  mv   s3,s1    #last=left
  addi s4,s1,1  #i=left+1
q_for_ini:
  slt  t6,s2,s4 #right<?i
  bne  t6,x0,q_for_end
     slli t0,s4,3  #byte offset i
     add  t1,s0,t0 # &v[i]
     ld   t2,0(t1) # v[i]
     lb   t3,0(t2) # *v[i]
     slli t0,s1,3  #byte offset left
     add  t1,s0,t0 # &v[left]
     ld   t2,0(t1) # v[left]
     lb   t4,0(t2) # *v[left]
     slt  t6,t3,t4 # (..)<(.)
     beq  t6,x0,q_if_end
     mv   a0,s0    #v
     addi s3,s3,1  #++last
     mv   a1,s3    # nuovo last
     mv   a2,s4    #i
     call swap
q_if_end:
  addi s4,s4,1 # i++
  b q_for_ini
q_for_end:
  mv   a0,s0 #v
  mv   a1,s1 #left
  mv   a2,s3 #last
  call swap
  mv   a0,s0 #v
  mv   a1,s1 #left
  addi a2,s3,-1 #last-1
  call qsort1
  mv   a0,s0 #v
  addi a1,s3,1 #last+1
  mv   a2,s2 #right
  call qsort1
q_end:
  ld   ra, 0(sp)
  ld   s0, 8(sp)
  ld   s1,16(sp)
  ld   s2,24(sp)
  ld   s3,32(sp)
  ld   s4,40(sp)
  addi sp,sp,+48 #
ret
#-----------------------------
main:
  # t0=k, t1=w, t2=buf, a0=p
  addi sp,sp,-40 #allocate p[5]

  # initializza il vettore p
  la t1, w    # &w
  mv a0,sp    # a2=&p
  sd t1,0(a0) # p[0]=&w
  addi t1,t1,1 # next char in w
  sd t1,8(a0) # p[1]=&(w+1)
  addi t1,t1,1 # next char in w
  sd t1,16(a0) # p[2]=&(w+2)
  addi t1,t1,1 # next char in w
  sd t1,24(a0) # p[3]=&(w+3)
  addi t1,t1,1 # next char in w
  sd t1,32(a0) # p[4]=&(w+4)

  # chiama qsort1
  li a1, 0  # 2nd param
  li a2, 4  # 3rd param
  call qsort1
  mv a1,sp    # ri-inizializza a1=&p
              # per uso successivo
  la a0,buf   # a0=&buf

  # stampa contenuto finale di p
  li t0,0   # k =0
  for_m_ini:
  slti t6,t0,5 # k<?5
  beq t6,x0,for_m_end # se falso: fine
    slli t1,t0,3 # byte offset k
    add  t1,a1,t1# &p[k]
    ld t6,0(t1)  # p[k]
    lb t5,0(t6)  # *p[k]
    sb t5,0(a0)  # buf[0]=(.)
    li a7,4 # print_string
    ecall
  addi t0,t0,1 # ++k
  b for_m_ini
  for_m_end:
  li a7,10
  ecall
```



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=64/4/2,     XM=X/B,    XS=XM%S,   XT=XM/S.
A=2, B=4, C=64, RP=LRU, Thit=4, Tpen=40, 20 references:
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  126   31   3   7   2  0 [7]:1,0 [7]:0,0  [7]:3,-
=== W  120   30   3   6   0  0 [6]:1,0 [6]:0,0  [6]:3,-
=== R  127   31   3   7   3  1 [7]:1,0 [7]:0,0  [7]:3,-
=== W  125   31   3   7   1  1 [7]:1,0 [7]:1,0  [7]:3,-
=== R  184   46   5   6   0  0 [6]:0,1 [6]:0,0  [6]:3,5
=== W  120   30   3   6   0  1 [6]:1,0 [6]:1,0  [6]:3,5
=== R  215   53   6   5   3  0 [5]:1,0 [5]:0,0  [5]:6,-
=== W  143   35   4   3   3  0 [3]:1,0 [3]:0,0  [3]:4,-
=== R  287   71   8   7   3  0 [7]:0,1 [7]:1,0  [7]:3,8
=== W  308   77   9   5   0  0 [5]:0,1 [5]:0,0  [5]:6,9
=== R  290   72   9   0   2  0 [0]:1,0 [0]:0,0  [0]:9,-
=== W  298   74   9   2   2  0 [2]:1,0 [2]:0,0  [2]:9,-
=== R  212   53   6   5   0  1 [5]:1,0 [5]:0,0  [5]:6,9
=== W  195   48   6   0   3  0 [0]:0,1 [0]:0,0  [0]:9,6
=== R  357   89  11   1   1  0 [1]:1,0 [1]:0,0  [1]:11,-
=== W  146   36   4   4   2  0 [4]:1,0 [4]:0,0  [4]:4,-
=== R  342   85  10   5   2  0 [5]:0,1 [5]:0,0  [5]:6,10
=== W  213   53   6   5   1  1 [5]:1,0 [5]:1,0  [5]:6,10
=== R  143   35   4   3   3  1 [3]:1,0 [3]:0,0  [3]:4,-
=== W  149   37   4   5   1  0 [5]:0,1 [5]:1,0  [5]:6,4
-------------------------------------
P1 Nmiss=14   Nhit=6  Nref=20  mrate=0.700000   AMAT=th+mrate*tpen=32
```

CONTENUTI dei SET al termine

LISTA BLOCCHI USCENTI:

```
(out: XM=77 XT=9  XS=5 )

(out: XM=85 XT=10 XS=5 )
```

ESERCIZIO 3

Per il segnale di generate G si ha banalmente $G = A \,\&\, B$
Per il segnale di propagate si tratta di una porta XOR che si può sintetizzare così in somma di prodotti: $P = A\bar{B} + \bar{A}B$
Per il segnale della somma basta scrivere la tabella di verità, es. in forma di mappa di Karnagh:

C_{IN} \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Quindi $S = \bar{A}\bar{B}C_{IN} + \bar{A}B\overline{C_{IN}} + ABC_{IN} + A\bar{B}\overline{C_{IN}}$

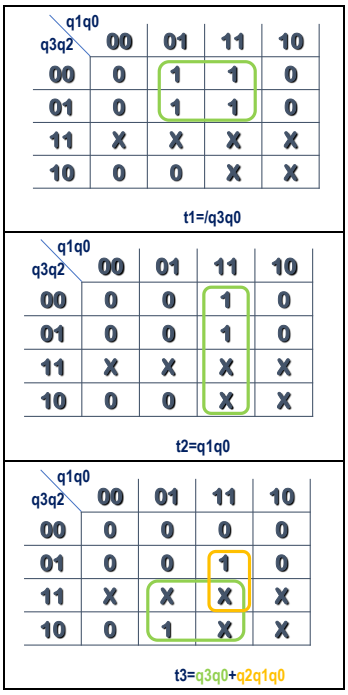
COMPITO di ARCHITETTURA DEI CALCOLATORI del 28-02-2023

SOLUZIONE

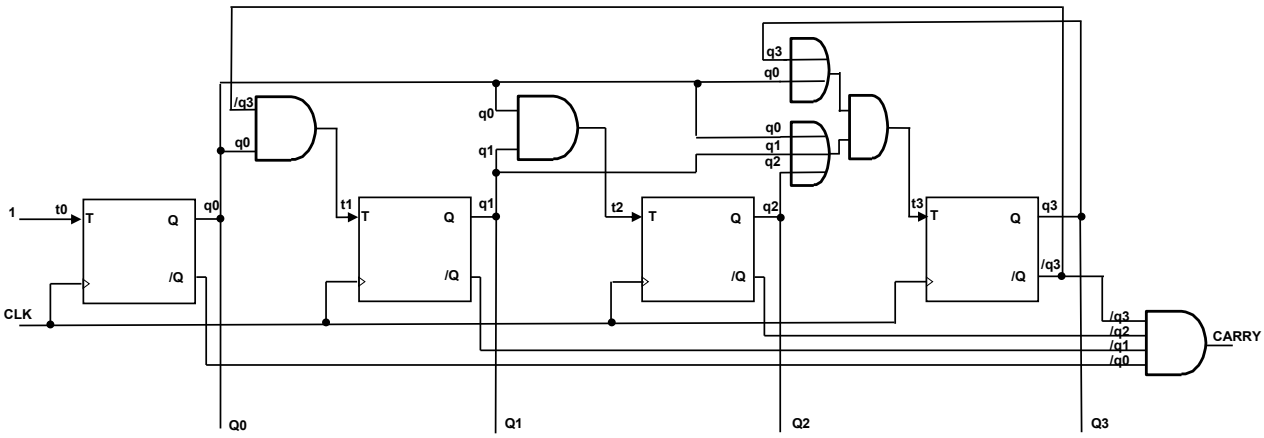
ESERCIZIO 4

Completando la tabella che mette in relazione stato presente e successivo si possono derivare i valori da inserire sugli ingressi dei FF-Tp. Da questa tabella si possono facilmente ricavare le reti combinatorie che mettono in relazione gli Q con T (basta guardare la colonna 'stato successivo' e notare i bit che ddevono commutare ("Toggle"). Per quanto riguarda t0 si vede facilmente che t0-1 (sempre) mentre per c=/q3/q2/q1/q0.

Stato attuale (Q)	Stato successivo	Ingressi del FF-Tp	CARRY
q3 q2 q1 q0	q3' q2' q1' q0'	t3 t2 t1 t0	C
0 0 0 0	0 0 0 1	0 0 0 1	1
0 0 0 1	0 0 1 0	0 0 1 1	0
0 0 1 0	0 0 1 1	0 0 0 1	0
0 0 1 1	0 1 0 0	0 1 1 1	0
0 1 0 0	0 1 0 1	0 0 0 1	0
0 1 0 1	0 1 1 0	0 0 1 1	0
0 1 1 0	0 1 1 1	0 0 0 1	0
0 1 1 1	1 0 0 0	1 1 1 1	0
1 0 0 0	1 0 0 1	0 0 0 1	0
1 0 0 1	0 0 0 0	1 0 0 1	0
1 0 1 0	X X X X	X X X X	X
1 0 1 1	X X X X	X X X X	X
1 1 0 0	X X X X	X X X X	X
1 1 0 1	X X X X	X X X X	X
1 1 1 0	X X X X	X X X X	X
1 1 1 1	X X X X	X X X X	X



Si può quindi far riferimento al seguente schema:



Codice Verilog:

```
module MyCounter(Q,c, clock,reset_);
  input clock, reset_;
  wire q0,q1,q2,q3,q0bar,q1bar,q2bar,q3bar,t0,t1,t2,t3;
  output[3:0] Q; output c;
  assign t0=1, t1=q3bar&q0, t2=q1&q0, t3=(q3&q0)|(q2&q1&q0), c=q3bar&q2bar&q1bar&q0bar;
  FFTp fp0(q0,q0bar,clock,reset_,  t0);
  FFTp fp1(q1,q1bar,clock,reset_,  t1);
  FFTp fp2(q2,q2bar,clock,reset_,  t2);
  FFTp fp3(q3,q3bar,clock,reset_,  t3);
  assign Q[3]=q3, Q[2]=q2, Q[1]=q1, Q[0]=q0;
endmodule
```

Diagramma di Temporizzazione: