COMPITO di ARCHITETTURA DEI CALCOLATORI del 16-02-2021

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 come file <COGNOME>.s all'indirizzo di posta giorgi@unisi con subject: C1210126)

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

```
char buff[80] = "2*3+4/5";

char find_op(char *s) {
  char op = '\0';
  while (*s != '\0') {
    if (*s < '0' || *s > '9') {
      op = *s; break;
    } else {
      ++s;
    }
  }
  return (op);
}

void printnl(char *b) {
  print_string(b); print_string("\n");
}
```

```
int main() {
  int a, b=2, d=3; double f=4, g=5, h;
  float w, x=7, y=8; char c, *p = buff;
  do {
    switch (c = find_op(p++)) {
      case '+': a = b + d; break;
      case '*': f = g * x; break;
      case '/': w = x / y; break;
    }
  } while (c != '\0');
  h = a * w;
  printnl(buff);
  print_double(h);
  exit(0);
}
```

RISCV Instructions (RV64IMFD) **v191222**

Instruction coding (hexadecimal) opcode+funct3+{funct7,imm}	Instruction	Example		Meaning	Comments (** 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/lbu**	**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	Load address of var (lui x5,H20(&var);ori x12, L12(&var)) H20=high 20 bit of &var; L12=low 12 bits of &var
PSEUDOINSTRUCTION	**jump**	**j/b**	**1000**	go to 1000	**(PSEUDO) INSTR. IS: jal x0,offset/beq x0,x0,offset**
PSEUDOINSTRUCTION	**jump and link (offset)**	**jal**	**100**	x1← (PC + 4); go to PC+100	**(PSEUDO) INSTR. IS: jal x1,offset**
PSEUDOINSTRUCTION	**return from procedure**	**ret**		PC←x1	**(PSEUDO) INSTR. IS: jalr x0,0(x1)**
67+0+imm	**jump and link register**	**jalr**	**x1, 100(x5)**	x1 ← (PC + 4); go to 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**		call OS service number in a7	See table of system calls below
73+0+8 (rs1=0,rs2=2,rd=0)	**sret**	**sret**		Exit Supervisor mode	-
PSEUDOINSTRUCTION	**move**	**mv**	**x5,x6**	x5 ← x6	**(PSEUDO) INSTR. IS: add x5,x0,x6**
PSEUDOINSTRUCTION	**load immediate**	**li**	**x5,100**	x5 ← 100	**(PSEUDO) INSTR. IS: addi x5,x0,100**
PSEUDOINSTRUCTION	**no operation (nop)**	**nop**		do nothing	**(PSEUDO) INSTR. IS: addi x0,x0,0**
53+0+{0,1}/53+0+{4,5}	**floating point add/sub**	**fadd.{s,d}/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.{s,d}/fdiv.{s,d}**	**f0,f1,f2**	f0←f1*f2 / f0←f1/f2	Single or double precision multiplication / division
53+2+{10,11}	**floating point absolute value**	**fabs.{s,d}**	**f0,f1**	f0← | f1 |	**(PSEUDO) INSTR. IS: fsgnjx.{s,d} f0,f1**
53+0+{10,11}	**floating point move between f-regs**	**fmv.{s,d}**	**f0,f1**	f0←f1	**(PSEUDO) INSTR. IS: fsgnj.{s,d} f0,f1**
53+1+{10,11}	**floating point negate**	**fneg.{s,d}**	**f0,f1**	f0← – (f1)	**(PSEUDO) INSTR. IS: fsgnjn.{s,d} f0,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}	**move between x (integer) and f regs**	**fmv.x.{s,d}**	**x5,f0**	x5←f0 (no conversion)	Copy (no conversion)
53+0+{78,79}	**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=1)	**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}	**convert to integer from {single,double}**	**fcvt.w.{s,d}**	**x5,f0**	x5← (int)f0	Type conversion
53+7+{68,69}	**convert to {single,double} from integer**	**fcvt.{s,d}.w**	**f0,x5**	f0← ({single,double})x5	Type conversion

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

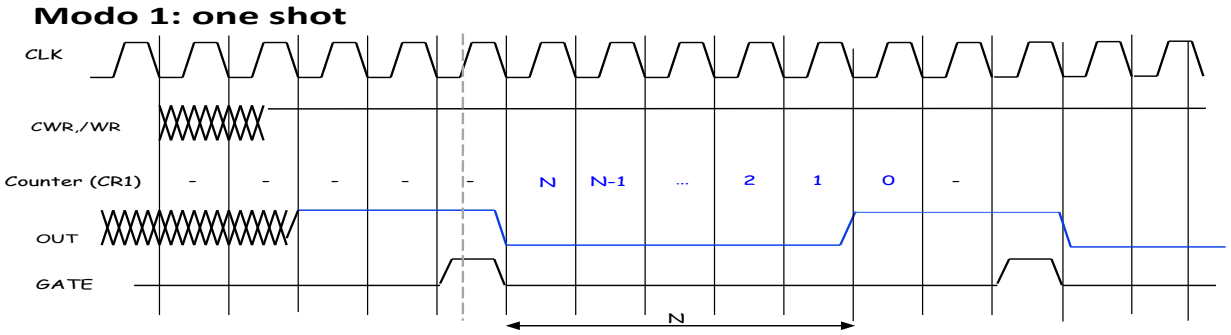
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Args
print_int	1	a0=integer to print	---
print_float	2	fa0=float to print	---
print_double	3	fa0=double to print	---
print_string	4	a0=address of ASCIIZ string to print	---
read_int	5	---	a0=integer

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	---	---

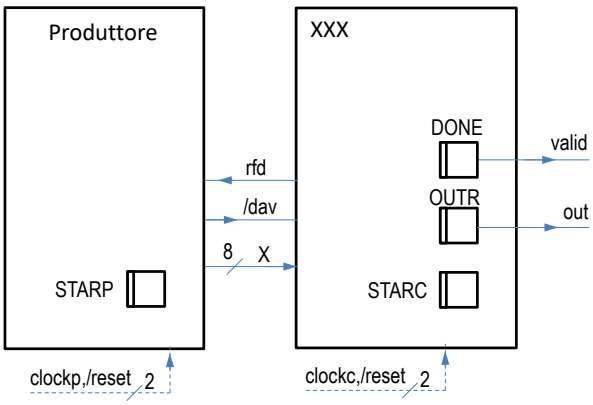
COMPITO di ARCHITETTURA DEI CALCOLATORI del 16-02-2021

2) [5/30] Si consideri una cache di dimensione 96B e a 3 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' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 723, 339, 327, 379, 778, 139, 333, 754, 725, 354, 322, 354, 739, 1, 26, 754, 324, 354, 729, 354, 328, 354. 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] Spiegare con proprie parole il funzionamento del "Modo 1" del timer 8254, il cui diagramma temporale è riportato in figura. Inoltre, indicare con precisione: i) il significato dei segnali rappresentati in tale diagramma, ii) come deve essere impostata la parola di controllo CWR e il relativo registro di conteggio per ottenere questo diagramma supponendo di utilizzare N=64001, il contatore n.1 in conteggio binario.

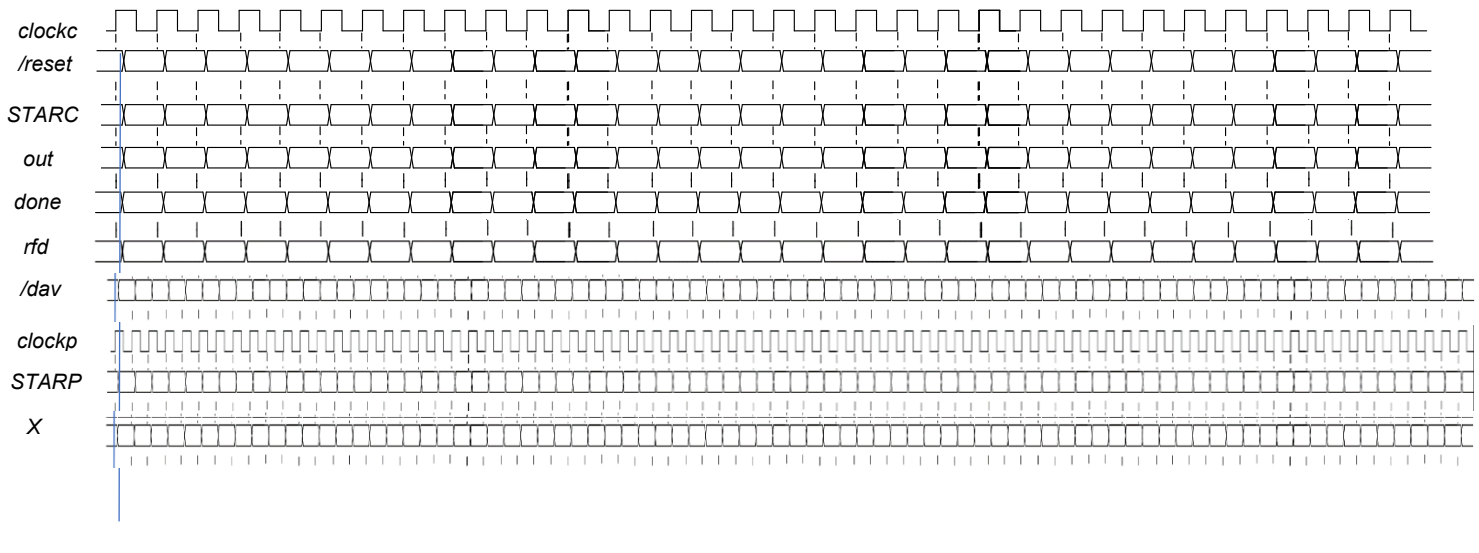


4)[9/30] Descrivere e sintetizzare in Verilog il modulo `XXX` di figura che funziona nel seguente modo: riceve un byte (`X`) dal modulo produttore col quale colloquia tramite i segnali `rfd` e `/dav`; ad ogni byte (X) ricevuto il modulo presenta sull'uscita `out` gli 8 bit di cui è costituito – dal più significativo al meno significativo – uno dopo l'altro per ogni ciclo di clock di `XXX` e indicandone la disponibilità abilitando il segnale `valid` per 8 cicli di clock di `XXX`. Il modulo `XXX` opera con un `clock` di periodo 4ns mentre il modulo Produttore, con `clockp`, puo' avere periodo sia 40ns (attuale codice) che 2ns: verificare il corretto funzionamento per entrambi i valori di `clockp`. Il codice del produttore e del testbench e' dato qua sotto. **Tracciare il diagramma di temporizzazione** come verifica della correttezza del modulo realizzato.




```
module testbench;
 reg reset_;
 initial begin reset_=0; #1 reset_=1; #300; $stop; end
 reg clockp;
 initial clockp =0; always #20 clockp <=(!clockp);
 wire[1:0] STARC=Xxx.STAR;
 wire[7:0]X; wire rfd, dav_;
 wire[1:0] STARP=PRO.STAR;
 wire[7:0] SX=Xxx.S, CX=Xxx.C;
 wire valid;
 wire out;
 reg clockc;
 initial clockc =0; always #2 clockc <=(!clockc);
 XXX        Xxx(dav_,X,clockc,reset_,    rfd,out,valid);
 Produttore PRO(rfd,clockp,reset_,       dav_,X);
endmodule
```

```
module Produttore(rfd,clock,reset_,   dav_,X);
 input rfd,clock,reset_; output dav_; output[7:0] X;
 reg DAV_; assign dav_=DAV_;
 reg[7:0] XP;   assign X=XP;
 reg[1:0] STAR; parameter S0=0, S1=1, S2=2, Q0=173;
 always @(reset_==0) begin STAR<=S0; end
 always @(posedge clock) if (reset_==1)  #0.1
  casex (STAR)
    S0: begin        DAV_=1; STAR<=(rfd==1)?S1:S0; end
    S1: begin XP=Q0; DAV_=0; STAR<=S2; end
    S2: begin                STAR<=(rfd==1)?S2:S0;end
 endcase
endmodule
```



COMPITO di ARCHITETTURA DEI CALCOLATORI del 16-02-2021

SOLUZIONE

ESERCIZIO 1

```
#char buff[80] = "2*3+4/5";
.data
buff: .asciz "2*3+4/5"
      .space  72
nl:   .asciz "\n"

.text
.globl main
#char find_op(char *s) {
# a0=s
find_op:
   mv t0, a0  # s
#  char op = '\0';
   li   a0, 0   # op = '\0'
#  while (*s != '\0') {
fo_while:
   lb     t6, 0(t0)
   # carica il carattere *s
   beq    t6, x0, fo_while_end
   # se *s==0 termina il ciclo

#    if (*s < '0' || *s > '9') {
   slti   t1, t6, '0'         # *s <? '0'
   li   t2, '9'
   slt  t3, t2, t6      # *s >? '9'
   or     t4, t1, t3      # t1 || t3
   beq    t4, x0, fo_if_else
   # se la cond. e' vera ho finito
#      op = *s; break;
   mv   a0, t6          # op=*s
   b    fo_while_end
#    } else {
#      ++s;
fo_if_else:
   addi t0, t0, 1  # ...altrimenti s++
#    }
#  }
   b      fo_while   # e continuo il while
fo_while_end:
#  return (op); op # sta gia in a0
   ret
#}

#void printnl(char *b) {
printnl:
#  print_string(b); print_string("\n");
   li a7,4
   ecall
   la  a0, nl
   ecall
   ret
#}
```

```
#int main() {
main:
   addi sp, sp, -56
   sw   ra, 0(sp)
   sw   s0, 4(sp) # p
   sw   s1, 8(sp) # a
   sw   s2,12(sp) # b
   sw   s3,16(sp) # d
   fsd fs0,20(sp) # f
   fsd fs1,28(sp) # g
   fsd fs2,36(sp) # h
   fsw fs3,44(sp) # w
   fsw fs4,48(sp) # x
   fsw fs5,52(sp) # y

#  int a, b=2, d=3;
#  s3 = a, s1 = b, s2 = d
   li   s2, 2         # b=2
   li   s3, 3         # d=3
#  double f=4, g=5, h;
#  fs0 = f, fs1 = g, fs2 = h
   li   t0, 4
   fcvt.d.w fs0,t0   # f=4
   li   t0, 5
   fcvt.d.w fs1,t0   # g=5

#  float w, x=7, y=8;
#  fs3 = w, fs4 = x, fs5 = y
   li   t0, 7
   fcvt.s.w fs4,t0   # x=7
   li   t0, 8
   fcvt.s.w fs5,t0   # y=8

#  char c, *p = buff;
#  t0=c, s0=p
   la   s0, buff      # p = buff

#  do {
m_dowhile:
#    switch (c = find_op(p++)) {
   mv   a0, s0        # p
   addi s0, s0, 1     # p++
   jal  find_op       # ritorna c=a0
#      case '+': a = b + d; break;
m_sw1:
   li   t0, '+'       # case '+' ?
   bne  a0,t0,m_sw2  # se no caso succ.
   add  s1, s2, s3    # se si', a=b+d
   b      m_sw_end     # break
#      case '*': f = g * x; break;
```

```
m_sw2:
   li   t0, '*'          # case '*' ?
   bne  a0,t0,m_sw3  # se no caso succ.
   fcvt.d.s ft4,fs4  # (double)x
   fmul.d fs0,fs1,ft4# se si', f=g*x
   b      m_sw_end     # break
#      case '/': w = x/y; break;
m_sw3:
   li   t0, '/'          # case '/' ?
   bne a0,t0,m_sw_end# se no fine switch
   fdiv.s fs3,fs4,fs5# se si', w=x/y
                         # break
#    }
m_sw_end:
#  } while (c != '\0');
   bne a0,x0,m_dowhile
   # c !=? '\0', se si' dowhile

#  h = a * w;
   fcvt.s.w ft0,s1   # (float)a
   fmul.s ft0,ft0,fs3# a*w
   fcvt.d.s fs2,ft0 # (double)h
#  printnl(buff);
   la a0, buff
   jal printnl
#  print_double(h);
   fmv.d fa0,fs2
   li a7,3  # print_double
   ecall
#  exit(0);
   li a7,10 # exit (servizio #10)
   ecall

   lw   ra, 0(sp)
   lw   s0, 4(sp)
   lw   s1, 8(sp)
   lw   s2,12(sp)
   lw   s3,16(sp)
   fld fs0,20(sp)
   fld fs1,28(sp)
   fld fs2,36(sp)
   flw fs3,44(sp)
   flw fs4,48(sp)
   flw fs5,52(sp)
   addi sp,sp,56
   ret
#}
```

```
Run I/O
2*3+4/5
4.375
-- 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=96/16/3, XM=X/B, XS=XM%S, XT=XM/S.
A=3, B=16, C=96, RP=LRU, Thit=4, Tpen=40, 22 references:

```
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  723   45  22   1   3  0 [1]:2,0,0 [1]:0,0,0  [1]:22,-,-
=== W  339   21  10   1   3  0 [1]:1,2,0 [1]:0,0,0  [1]:22,10,-
=== R  327   20  10   0   7  0 [0]:2,0,0 [0]:0,0,0  [0]:10,-,-
=== W  379   23  11   1  11  0 [1]:0,1,2 [1]:0,0,0  [1]:22,10,11
=== R  778   48  24   0  10  0 [0]:1,2,0 [0]:0,0,0  [0]:10,24,-
=== W  139    8   4   0  11  0 [0]:0,1,2 [0]:0,0,0  [0]:10,24,4
=== R  333   20  10   0  13  1 [0]:2,0,1 [0]:0,0,0  [0]:10,24,4
=== W  754   47  23   1   2  0 [1]:2,0,1 [1]:0,0,0  [1]:23,10,11
=== R  725   45  22   1   5  0 [1]:1,2,0 [1]:0,0,0  [1]:23,22,11
=== W  354   22  11   0   2  0 [0]:1,2,0 [0]:0,0,0  [0]:10,11,4
=== R  322   20  10   0   2  1 [0]:2,1,0 [0]:0,0,0  [0]:10,11,4
=== W  354   22  11   0   2  1 [0]:1,2,0 [0]:0,1,0  [0]:10,11,4
=== R  739   46  23   0   3  0 [0]:0,1,2 [0]:0,1,0  [0]:10,11,23
=== W    1    0   0   0   1  0 [0]:2,0,1 [0]:0,1,0  [0]:0,11,23
=== R   26    1   0   1  10  0 [1]:0,1,2 [1]:0,0,0  [1]:23,22,0
=== W  754   47  23   1   2  1 [1]:2,0,1 [1]:1,0,0  [1]:23,22,0
=== R  324   20  10   0   4  0 [0]:1,2,0 [0]:0,0,0  [0]:0,10,23
=== W  354   22  11   0   2  0 [0]:0,1,2 [0]:0,0,0  [0]:0,10,11
=== R  729   45  22   1   9  1 [1]:1,2,0 [1]:1,0,0  [1]:23,22,0
=== W  354   22  11   0   2  1 [0]:0,1,2 [0]:0,0,1  [0]:0,10,11
=== R  328   20  10   0   8  1 [0]:0,2,1 [0]:0,0,1  [0]:0,10,11
=== W  354   22  11   0   2  1 [0]:0,1,2 [0]:0,0,1  [0]:0,10,11
---------------------------------------
P1 Nmiss=14   Nhit=8  Nref=22  mrate=0.636364   AMAT=th+mrate*tpen=29.4545
```

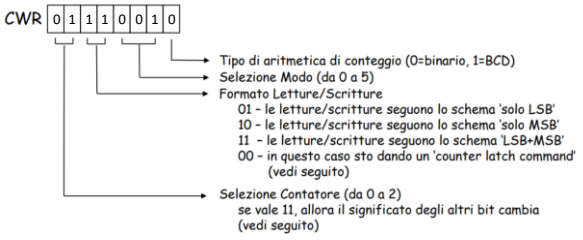
```
(out: XM=45 XT=22 XS=1 )
(out: XM=21 XT=10 XS=1 )
(out: XM=48 XT=24 XS=0 )

(out: XM=8  XT=4  XS=0 )
(out: XM=20 XT=10 XS=0 )
(out: XM=23 XT=11 XS=1 )

(out: XM=22 XT=11 XS=0 )
(out: XM=46 XT=23 XS=0 )
```

LISTA BLOCCHI USCENTI:

CONTENUTI dei SET al termine

ESERCIZIO 3

Il modo 1 viene utilizzato per realizzare sull'uscita OUT (es. OUT1 per CR1) una commutazione da "0" verso "1" dopo un ritardo temporale pari a N/fc essendo N la costante di tempo scritta nel registro di conteggio (es. CR1), mentre fc è la frequenza applicata sul piedino CLK corrispondente al contatore di interesse (es. CLK1 per CR1); il ritardo viene conteggiato a partire da un istante di innesco che parte dal fronte in discesa del clock successivo ad un impulso di valore "1" applicato sul piedino GATE (es. GATE1 per CR1).



i) In figura sono rappresentati i segnali appena discussi (OUT, GATE, CLK); inoltre, "Counter" (CR1) indica il valore assunto dal contatore durante il conteggio, mentre CWR indica il valore impostato nel registro CWR e /WR è il segnale di scrittura applicato per poter scrivere nei registri CR1 e CWR.

ii) La parola di controllo deve valere 0111'0010=0x72, essendo necessario effettuare due scritture da 8 bit per scrivere i 16 bit della costante N=64001=0xFA01, ovvero basta scrivere 0x72 in CRW e 0xFA seguito da 0x01 in CR1.

COMPITO di ARCHITETTURA DEI CALCOLATORI del 16-02-2021

SOLUZIONE

ESERCIZIO 4

DIAGRAMMA DEGLI STATI DEL PRODUTTORE (codice dato)

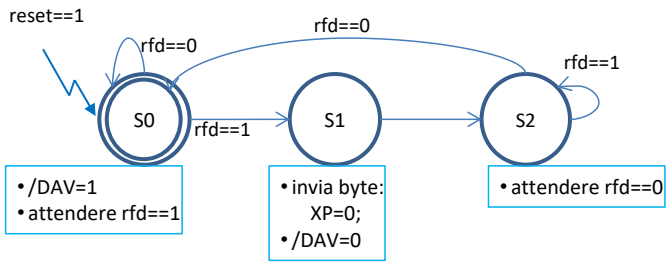
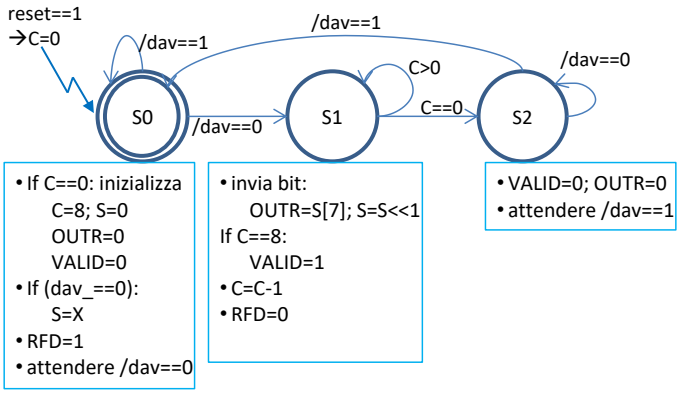


DIAGRAMMA DEGLI STATI DEL CONSUMATORE

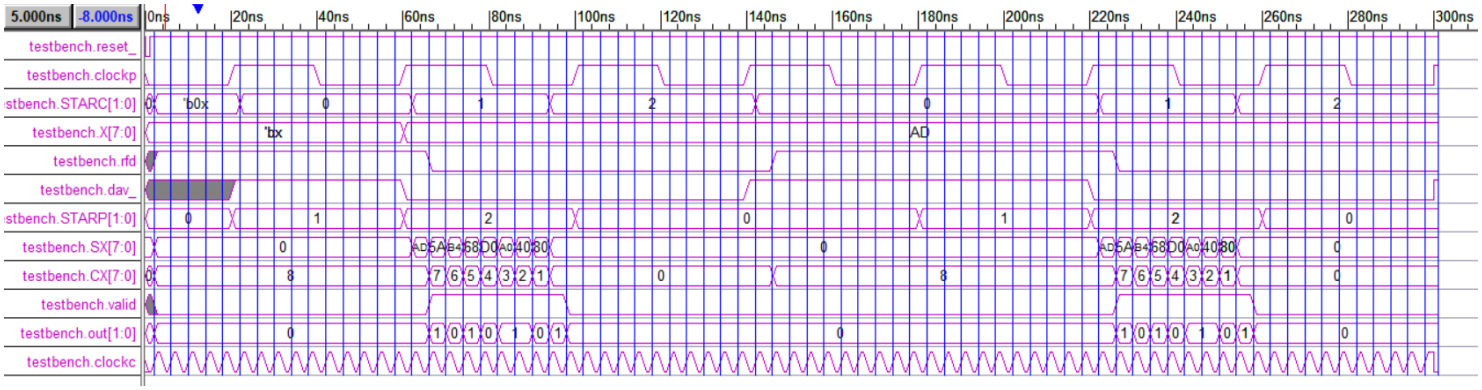


Codice Verilog del modulo da realizzare (consumatore XXX)

```
module XXX(dav_,X,clock,reset_,  rfd,out,valid);
 input       dav_,clock, reset_; input[7:0] X;
 output      rfd,valid,out;
 reg         OUTR; assign out=OUTR;
 reg[7:0]    S;
 reg         RFD,VALID; assign rfd=RFD, valid=VALID;
 reg[1:0]    STAR;  reg[7:0] C;
 parameter   S0=0,S1=1,S2=2,S3=3;

 always @(reset_==0) begin STAR<=S0; C<=0; end

 always @(posedge clock) if (reset_==1)  #0.1
  casex (STAR)
   S0: begin if (C==0) begin C=8; OUTR=0; VALID=0; S<=0; end
       RFD=1; STAR<=(dav_==0)?S1:S0; if (dav_==0) begin S=X; end end
   S1: begin OUTR=S[7]; S=S<<1; if (C==8) VALID=1;  C=C-1;
       RFD=0; STAR<=(C==0)?S2:S1; end
   S2: begin  VALID=0; OUTR=0; STAR<=(dav_==0)?S2:S0; end
  endcase
endmodule
```

Diagramma di Temporizzazione:

T_{CLOCKC}=4ns, T_{CLOCKP}=40ns



T_{CLOCKC}=4ns, T_{CLOCKP}=2ns