COMPITO di ARCHITETTURA DEI CALCOLATORI del 18-07-2022

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) [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).

```
#define SAMPLES 10
#define MATSIZE 16
int mat[MATSIZE][MATSIZE];

void transpose() {
    for (int i = 0 ; i < MATSIZE ; i++) {
        for (int j = i+1 ; j < MATSIZE ; j++) {
            int aux = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = aux;
        }
    }
}
```

Nota: 'int' è un intero a 64 bit.

```
int main() {
    for (int i = 0 ; i < MATSIZE ; i++)
        for (int j = 0 ; j < MATSIZE ; j++)
            mat[i][j] = i+j;
    for (int k = 0 ; k < SAMPLES ; k++)
        transpose();
    int r = 0;
    for (int i = 0 ; i < MATSIZE ; i++)
        r = r + mat[0][i];
    print_int(r);
}
```

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

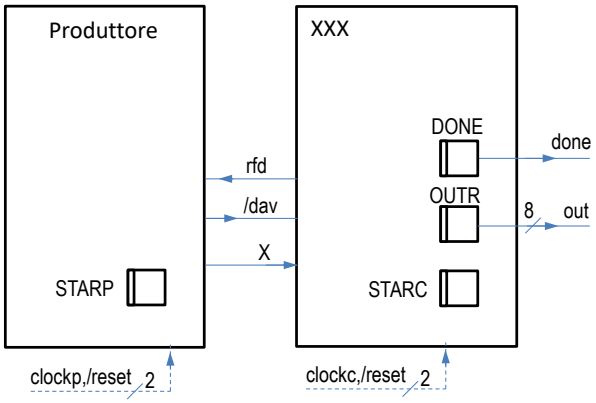
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	fa0=float
read_double	7	---	fa0=double
read_string	8	a0=address of input buffer, a1=max chars to read	---
sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
exit	10	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 18-07-2022

2) [5/30] Si consideri una cache di dimensione 32B 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: 7123, 7339, 7327, 7339, 7328, 7139, 7333, 7354, 7325, 7354, 7322, 7354, 7339, 7126, 7354, 7324, 7354, 7329, 7354, 7328, 7354. 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.

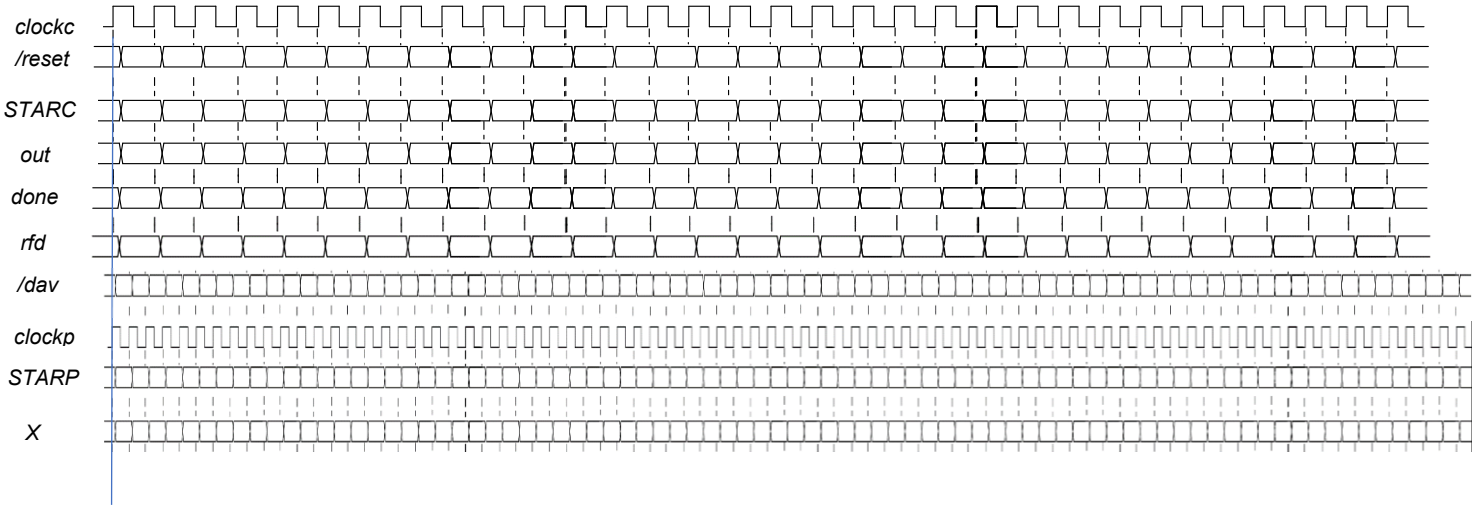
3) [5/30] Disegnare l'organizzazione fisica di una memoria DRAM da 16Mbit indicando i collegamenti fra i blocchi CTRL, ROW_LATCHES, COL_LATCHES, ROW_DECODER, COL_DECODER, ROW_BUFFERS, PRECHARGE, TRISTATE e spiegare lo svolgimento delle operazioni di accesso ad un singolo bit.

4) [10/30] Descrivere e sintetizzare in Verilog il modulo `XXX` di figura che funziona nel seguente modo: riceve un bit (`X`) dal modulo produttore col quale colloquia tramite i segnali `rfd` e `/dav`; ogni otto bit (Xi) il modulo presenta sull'uscita `out` un byte (8-bit), indicandone la disponibilita' abilitando il segnale `done` per 1 ciclo di clock di `XXX`. Il modulo `XXX` opera con un `clockc` di periodo 4ns mentre il modulo Produttore, con `clockp`, ha un periodo di 2ns: verificare il corretto funzionamento. Il codice del produttore e del testbench e' dato qua sotto. **Tracciare il diagramma di temporizzazione (punti 5/10)** come verifica della correttezza del modulo realizzato.




```
module testbench;
 reg reset_;
 initial begin reset_=0; #1 reset_=1; #400; $stop; end
 reg clockc;
 initial clockc =0; always #2 clockc <=(!clockc);
 wire[1:0] STARC=Xxx.STAR;
 wire[7:0]out; wire done, rfd, dav_;
 wire[1:0] X;
 reg clockp;
 initial clockp =0; always #1 clockp <=(!clockp);
 wire[1:0] STARP=PRO.STAR;
 wire[7:0] QP=PRO.QP, CP=PRO.C, SX=Xxx.S, CX=Xxx.C;
 XXX        Xxx(dav_,X,clockc,reset_,    rfd,out,done);
 Produttore PRO(rfd,clockp,reset_,       dav_,X);
endmodule
```

```
module Produttore(rfd,clock,reset_,   dav_,X);
 input rfd,clock,reset_; output dav_,X;
 reg DAV_; assign dav_=DAV_;
 reg XP;   assign X=XP;
 reg[7:0] QP, C;
 reg[1:0] STAR; parameter S0=0, S1=1, S2=2, Q0=173;
 always @(reset_==0) begin C<=0; STAR<=S0; end
 always @(posedge clock) if (reset_==1)  #0.1
  casex (STAR)
   S0: begin if (C==0)begin C=8; QP=Q0; XP=QP[7]; end
       DAV_=1; STAR<=(rfd==1)?S1:S0; end
   S1: begin XP=QP[7]; QP=QP<<1; C=C-1;
       DAV_=0; STAR<=S2; end
   S2: begin   STAR<=(rfd==1)?S2:S0;end
 endcase
endmodule
```



COMPITO di ARCHITETTURA DEI CALCOLATORI del 18-07-2022 (REV.231205)

SOLUZIONE

ESERCIZIO 1

```
.data
mat: .space 2048

.text
.globl main
transpose:
    la   a0,mat       # &mat
    li   a1,16        # MATSIZE
    li   t0,0         # i=0
tr_for1_ini:
    slt  t3,t0,a1     # i<?MATSIZE
    beq  t3,zero,tr_for1_end # false->end
    #----- for1-body start
    addi t1,t0,1      # j=i+1
tr_for2_ini:
    slt  t3,t1,a1     # j<?MATSIZE
    beq  t3,zero,tr_for2_end # false->end
    #----- for2-body start
    #calc &mat+(i*16+j)*8
    slli t4,t0,4      # i*MATSIZE
    add  t4,t4,t1     # i*MTATSIZE+j
    slli t4,t4,3      # (.)*8
    add  t4,a0,t4     # &mat[i][j]
    #calc &mat+(j*16+i)*8
    slli t5,t1,4      # j*MATSIZE
    add  t5,t5,t0     # j*MATSIZe+i
    slli t5,t5,3      # (.)*8
    add  t5,a0,t5     # &mat[j][i]

    ld   a2,0(t4)     # temp=mat[i][j]
    ld   a3,0(t5)     # temp2=mat[j][i]
    sd   a3,0(t4)     # mat[i][j]=temp2
    sd   a2,0(t5)     # mat[j][i]=temp

    #----- for2-body end
    addi t1,t1,1      # ++j
    b    tr_for2_ini
tr_for2_end:
    #----- for1-body end
    addi t0,t0,1      # ++i
    b    tr_for1_ini
tr_for1_end:
    ret
```

```
main:
    addi sp,sp,-8     # allocate frame
    sd   s0,0(sp)     # save s0

    la   a0,mat       # &mat
    li   a1,16        # MATSIZE
    li   t0,0         # i=0
for1_ini:
    slt  t3,t0,a1     # i<?MATSIZE
    beq  t3,zero,for1_end # false->end
    #----- for1-body start
    li   t1,0         # j=0
for2_ini:
    slt  t3,t1,a1     # j<?MATSIZE
    beq  t3,zero,for2_end # false->end
    #----- for2-body start
    #calc &mat+(i*16+j)*8
    slli t4,t0,4      # i*MATSIZE
    add  t4,t4,t1     # i*MATSIZE+j
    slli t4,t4,3      # (.)*8
    add  t4,a0,t4     # &mat[i][j]
    add  t5,t0,t1     # i+j
    sd   t5,0(t4)     # mat[i][j]=i+j

    #----- for2-body end
    addi t1,t1,1      # ++j
    b    for2_ini
for2_end:
    #----- for1-body end
    addi t0,t0,1      # ++i
    b    for1_ini
for1_end:

    li   s0,0         # k=0
for3_ini:
    slti t3,s0,10     # k<?10
    beq  t3,zero,for3_end # false->end
    #----- for3-body start
    jal transpose
    #----- for3-body end
    addi s0,s0,1      # ++i
    b    for3_ini
for3_end:
```

```
    li   t0,0         # i=0
    li   a0,0         # r=0
    li   a1,16        # MATSIZE
    la   a2,mat       # &mat
for4_ini:
    slt  t3,t0,a1     # i<?MATSIZE
    beq  t3,zero,for4_end # false->end
    #----- for4-body start
    #calc &mat+(0*MATSIZE+i)*8
    slli t4,zero,4    # 0*MATSIZE
    add  t4,t4,t0     # 0*MATSIZE+i
    slli t4,t4,3      # (.)*8
    add t4,a2,t4      # &mat[0][i]
    ld   t5,0(t4)     # mat[0][i]
    add a0,a0,t5      # r=r+(.)

    #----- for4-body end
    addi t0,t0,1      # ++i
    b    for4_ini
for4_end:
    li   a7,1         # print_int
    ecall

    ld   s0,0(sp)     # restore s0
    addi sp,sp,8      # free frame

    li   a7,10        # exit
    ecall
```

```
Run I/O
120
-- 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=32/4/2, XM=X/B, XS=XM%S, XT=XM/S.
A=2, B=4, C=32, RP=LRU, Thit=4, Tpen=40, 21 references:

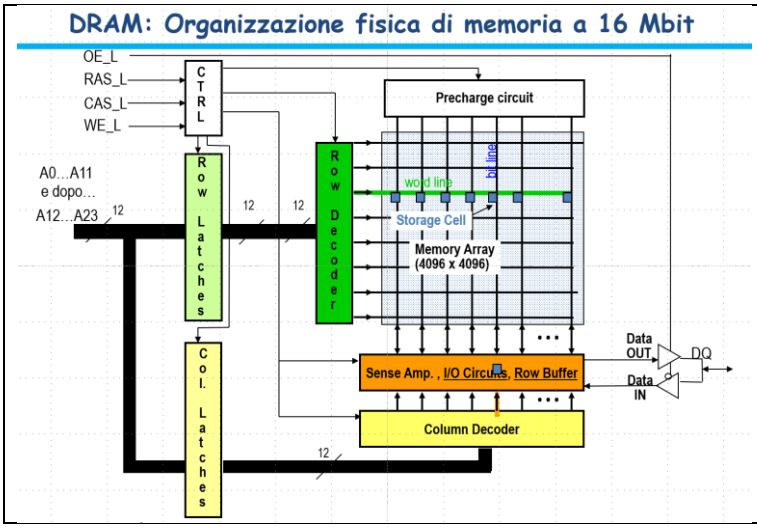
```
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R 7123 1780 445   0   3  0 [0]:1,0 [0]:0,0  [0]:445,-
=== W 7339 1834 458   2   3  0 [2]:1,0 [2]:0,0  [2]:458,-
=== R 7327 1831 457   3   3  0 [3]:1,0 [3]:0,0  [3]:457,-
=== W 7339 1834 458   2   3  1 [2]:1,0 [2]:1,0  [2]:458,-
=== R 7328 1832 458   0   0  0 [0]:0,1 [0]:0,0  [0]:445,458
=== W 7139 1784 446   0   3  0 [0]:1,0 [0]:0,0  [0]:446,458 (out: XM=1780 XT=445 XS=0 )
=== R 7333 1833 458   1   1  0 [1]:1,0 [1]:0,0  [1]:458,-
=== W 7354 1838 459   2   2  0 [2]:0,1 [2]:1,0  [2]:458,459
=== R 7325 1831 457   3   1  1 [3]:1,0 [3]:0,0  [3]:457,-
=== W 7354 1838 459   2   2  1 [2]:0,1 [2]:1,1  [2]:458,459
=== R 7322 1830 457   2   2  0 [2]:1,0 [2]:0,1  [2]:457,459 (out: XM=1834 XT=458 XS=2 )
=== W 7354 1838 459   2   2  1 [2]:0,1 [2]:0,1  [2]:457,459
=== R 7339 1834 458   2   3  0 [2]:1,0 [2]:0,1  [2]:458,459 (out: XM=1830 XT=457 XS=2 )
=== W 7126 1781 445   1   2  0 [1]:0,1 [1]:0,0  [1]:458,445
=== R 7354 1838 459   2   2  1 [2]:0,1 [2]:0,1  [2]:458,459
=== W 7324 1831 457   3   0  1 [3]:1,0 [3]:1,0  [3]:457,-
=== R 7354 1838 459   2   2  1 [2]:0,1 [2]:0,1  [2]:458,459
=== W 7329 1832 458   0   1  1 [0]:0,1 [0]:0,1  [0]:446,458
=== R 7354 1838 459   2   2  1 [2]:0,1 [2]:0,1  [2]:458,459
=== W 7328 1832 458   0   0  1 [0]:0,1 [0]:0,1  [0]:446,458
=== R 7354 1838 459   2   2  1 [2]:0,1 [2]:0,1  [2]:458,459
-------------------------------------
P1 Nmiss=10   Nhit=11   Nref=21   mrate=0.476190   AMAT=th+mrate*tpen=23.0476
```

LISTA BLOCCHI USCENTI:

CONTENUTI dei SET al termine

Sia X il generico riferimento, A=associativita', B=dimensione del blocco, C=capacita' della cache.

ESERCIZIO 3



- I 24 bit che compongono l'indirizzo vengono inviato in due gruppi di 12 bit ciascuno e memorizzando i 12 bit più significativi ad es. nei ROW_LATCHES e I 12 meno significative nei COL_LATCHES
- **In ogni caso i 24 bit non possono essere utilizzati simultaneamente, in quanto a causa della struttura compatta della cella DRAM deve innanzitutto essere letta una intera riga della matrice 4096x4096 e bufferizzata nel ROW_BUFFER**
- In una fase successiva si possono usare I bit del COL_LATCHES per Selezionare il singolo bit all'interno della riga di 4096 bit bufferizzata
- Una volta selezionato un bit questo può essere letto dirigendolo sull'uscita DQ attraverso opportuna selezione dei tristate buffer (DATA_IN, DATA_OUT)
- **L'operazione di lettura si svolge in modo analogo ma andando preventivamente a scrivere nel ROW_BUFFER il bit di interesse e poi riscrivendo l'intera riga letta all'interno della matrice di bit.**

COMPITO di ARCHITETTURA DEI CALCOLATORI del 18-07-2022 (REV.231205)

SOLUZIONE

ESERCIZIO 4

DIAGRAMMA DEGLI STATI DEL PRODUTTORE (codice dato)

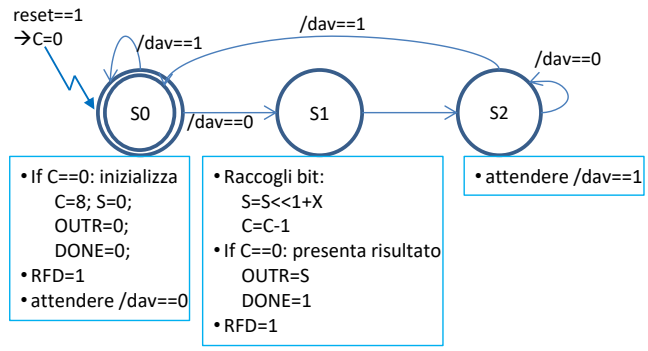
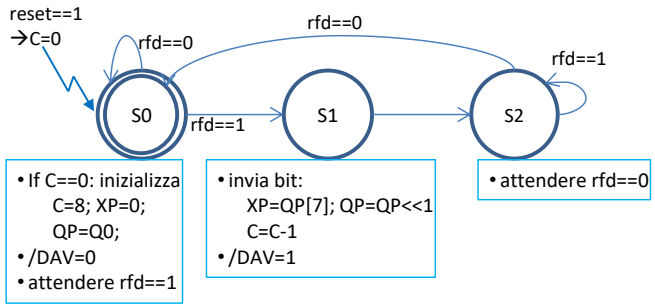


DIAGRAMMA DEGLI STATI DEL CONSUMATORE (da trovare)



Codice Verilog del modulo da realizzare (consumatore XXX)

```
module XXX(dav_,X,clock,reset_,  rfd,out,done);
 input       dav_,X,clock, reset_;
 output[7:0] out; output rfd,done;
 reg[7:0]    OUTR,S; assign out=OUTR;
 reg         RFD,DONE; assign rfd=RFD, done=DONE;
 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; DONE=0; S<=0; end
       RFD=1; STAR<=(dav_==0)?S1:S0; end
   S1: begin S=(S<<1)+X; C=C-1; if (C==0) begin OUTR=S; DONE=1; end
       RFD=0; STAR<=S2; end
   S2: begin  STAR<=(dav_==0)?S2:S0; end
  endcase
endmodule
```

Diagramma di Temporizzazione:

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