COMPITO di ARCHITETTURA DEI CALCOLATORI del 27-06-2022

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) [11/30] 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 A(i,j) A[(i)*(n)+(j)]
#define C(i,j) C[(i)*(n)+(j)]
float A[9] = { 1, 2, 3, 4, 5, 6, 7, 8, 8 };

float *ssyrk(float *A, float *C, float alpha, float beta, int n) {
   int i, j, p; float temp;

   for (j = 0; j < n; ++j) {
      for (i = 0; i < n; ++i) {
         temp = 0.0;
         for (p = 0; p < n; ++p) {
            temp += A(p,i) * A(p,j);
         }
         C(i,j) = alpha * temp + beta * C(i,j);
      }
   }
   return(C);
}
```

```
int main() {
    float *C, alpha = 7.0, beta = 3.0;
    int n = 3;

    C = (float*)sbrk(n*n * sizeof(float));
    C = ssyrk(A, C, alpha, beta, n);
    print_float(C(2,2));
    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	---	---

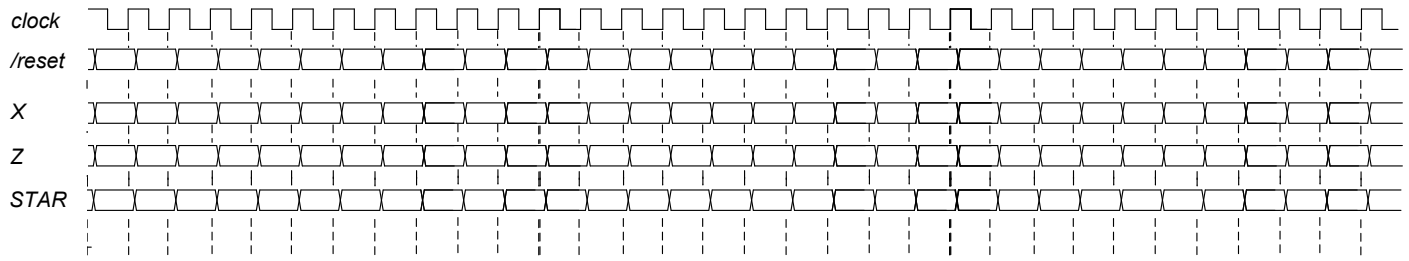
COMPITO di ARCHITETTURA DEI CALCOLATORI del 27-06-2022

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: 6123, 5339, 6327, 5339, 6328, 5139, 6333, 5354, 6325, 5354, 6322, 5354, 6339, 5126, 6354, 5324, 6354, 5329, 6354, 5328, 6354. 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 il diagramma di handshake asincrono in scrittura indicando i segnali di indirizzamento (A), dati (D) lettura/scrittura (R/W), richiesta (REQ) e approvazione (ACK) e spiegare in dettaglio i sei istanti rilevanti di tale diagramma (cosa accade, quali sono i prerequisiti).

4) [9/30] Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Moore con un ingresso X su un bit e una uscita Z su un bit che funziona nel seguente modo: devono essere riconosciute le sequenze non-interallacciate 1,1,0 e 1,0,1; l'uscita Z va a 1 (per 1 ciclo di clock) se è presente una delle due sequenze. Gli stimoli di ingresso sono dati dal seguente modulo Verilog Testbench.

Tracciare il diagramma di temporizzazione [4/9 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 TopLevel;
reg reset_;initial begin reset_=0; #22 reset_=1; #300; $stop; end
reg clock ;initial clock=0; always #5 clock <=(!clock);
reg X;
wire Z;
wire [2:0] STAR=Xxx.STAR;
initial begin X=0;
wait(reset_==1); #5
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=1;
@(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=0; @(posedge clock); X<=1;
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=0;
@(posedge clock); X<=1; @(posedge clock); X<=1; @(posedge clock); X<=1; @(posedge clock); X<=0;
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=0;
$finish;
end
XXX Xxx(X,Z,clock,reset_);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 27-06-2022

SOLUZIONE

ESERCIZIO 1

```
.data
A: .float 1, 2, 3, 4, 5, 6, 7, 8, 8
a1: .float 7.0
be: .float 3.0
.text
.globl main
#float *ssyrk(float *A, float *C, float alpha,
float beta, int n) {
ssyrk: #a0=&A, a1=&C, a2=n, fa0=alpha, fa1=beta
    # int i, j, p; float temp;
    # t0=i, t1=j, t2=p, ft0=temp
    li   t1,0          #j=0
for1_ini:
    slt  t3,t1,a2      # j<?n
    beq  t3,zero,for1_end # false->end
    #----- for1-body start
    li   t0,0          # i=0
for2_ini:
    slt  t3,t0,a2      # i<?n
    beq  t3,zero,for2_end # false->end
    #----- for2-body start
    fmv.s.x ft0,zero   # temp=0.0
    li   t2,0          # p=0
for3_ini:
    slt  t3,t2,a2      # p<?n
    beq  t3,zero,for3_end # false->end
    #----- for3-body start
    #calc &A+(p*n+i)*4
    mul t4,t2,a2       # p*n
    add t4,t4,t0       # p*n+i
    slli t4,t4,2       # (.)*4
    add t4,a0,t4       # &A(p,i)
    flw ft1,0(t4)      # A(p,i)
    #calc &A+(p*n+j)*4
    mul t4,t2,a2       # p*n
    add t4,t4,t1       # p*n+j
    slli t4,t4,2       # (.)*4
    add t4,a0,t4       # &A(p,j)
    flw ft2,0(t4)      # A(p,j)
    fmul.s ft1,ft1,ft2# A(p,i) * A(p,j)
    fadd.s ft0,ft0,ft1# temp += (.)
    #----- for3-body end
    addi t2,t2,1       # ++p
    b    for3_ini
for3_end:
    #calc &C+(i*n+j)*4
    mul t4,t0,a2       # i*n
    add t4,t4,t1       # p*n+j
    slli t4,t4,2       # (.)*4
    add t4,a1,t4       # &C(i,j)
    flw ft1,0(t4)      # C(i,j)
    fmul.s ft1,ft1,fa1# (.)*beta
    fmul.s ft2,fa0,ft0# alpha*temp
    fadd.s ft1,ft1,ft2# (..)+(.)
    fsw ft1,0(t4)      # C(i,j)=(.)
    #----- for2-body end
    addi t0,t0,1       # ++i
    b    for2_ini
for2_end:
    #----- for1-body end
    addi t1,t1,1       # ++j
    b    for1_ini
for1_end:
    mv   a0,a1         # a0=&C
    ret

main:
    li      a0,3       # n
    mul     a0,a0,a0   # n*n
    slli    a0,a0,2    # n*n *4 (4=sizeof(float))
    li      a7,9       # sbrk
    ecall              # allocate 36 bytes
    mv      a1,a0      # 2nd param. = &C
    la      a0,A         # 1st param. = &A
    la      t0,a1      # &alpha
    flw     fa0,0(t0) # 3rd param. alpha=3.0
    la      t0,be      # &beta
    flw     fa1,0(t0) # 4th param. beta=7.0
    li      a2,3       # 5th param. n=3
    jal     ssyrk      # returns &C in a0
    li      a2,3       # n=3
    li      t0,2
    mul     t0,t0,a2   # 2*n
    addi    t0,t0,2    # 2*n+2
    slli    t0,t0,2    # (.) *4 (4=sizeof(float))
    add     t0,a0,t0   # &C(2,2)
    flw     fa0,0(t0) # C(2,2)
    li      a7,2       # print_float
    ecall
    li      a7,10
    ecall
```

```
Run I/O
763.0
-- program is finished running (0) --
```

ESERCIZIO 2

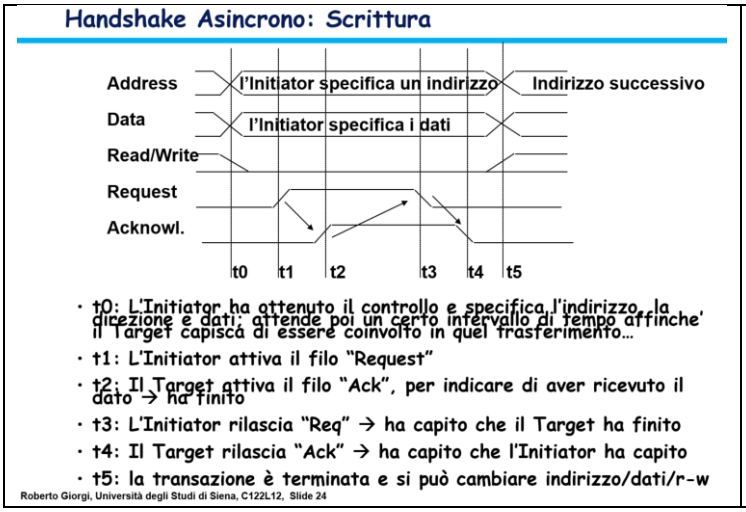
Sia X il generico riferimento, A=associativita', B=dimensione del blocco, C=capacita' della cache.
Si ricava S=C/B/A=# di set della cache=64/8/2, XM=X/B, XS=XM%S, XT=XM/S.
A=2, B=8, C=64, RP=LRU, Thit=4, Tpen=40, 21 references:

```
=== T    X    XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R 6123  765 191   1   3  0 [1]:1,0 [1]:0,0  [1]:191,-
=== W 5339  667 166   3   3  0 [3]:1,0 [3]:0,0  [3]:166,-
=== R 6327  790 197   2   7  0 [2]:1,0 [2]:0,0  [2]:197,-
=== W 5339  667 166   3   3  1 [3]:1,0 [3]:1,0  [3]:166,-
=== R 6328  791 197   3   0  0 [3]:0,1 [3]:1,0  [3]:166,197
=== W 5139  642 160   2   3  0 [2]:0,1 [2]:0,0  [2]:197,160
=== R 6333  791 197   3   5  1 [3]:0,1 [3]:1,0  [3]:166,197
=== W 5354  669 167   1   2  0 [1]:0,1 [1]:0,0  [1]:191,167
=== R 6325  790 197   2   5  1 [2]:1,0 [2]:0,0  [2]:197,160
=== W 5354  669 167   1   2  1 [1]:0,1 [1]:0,1  [1]:191,167
=== R 6322  790 197   2   2  1 [2]:1,0 [2]:0,0  [2]:197,160
=== W 5354  669 167   1   2  1 [1]:0,1 [1]:0,1  [1]:191,167
=== R 6339  792 198   0   3  0 [0]:1,0 [0]:0,0  [0]:198,-
=== W 5126  640 160   0   6  0 [0]:0,1 [0]:0,0  [0]:198,160
=== R 6354  794 198   2   2  0 [2]:0,1 [2]:0,0  [2]:197,198 (out: XM=642 XT=160 XS=2 )
=== W 5324  665 166   1   4  0 [1]:1,0 [1]:0,1  [1]:166,167 (out: XM=765 XT=191 XS=1 )
=== R 6354  794 198   2   2  1 [2]:0,1 [2]:0,0  [2]:197,198
=== W 5329  666 166   2   1  0 [2]:1,0 [2]:0,0  [2]:166,198 (out: XM=790 XT=197 XS=2 )
=== R 6354  794 198   2   2  1 [2]:0,1 [2]:0,0  [2]:166,198
=== W 5328  666 166   2   0  1 [2]:1,0 [2]:1,0  [2]:166,198
=== R 6354  794 198   2   2  1 [2]:0,1 [2]:1,0  [2]:166,198
-------------------------------------
P1 Nmiss=11   Nhit=10  Nref=21  mrate=0.523810   AMAT=th+mrate*tpen=24.9524
```

CONTENUTI dei SET al termine

LISTA BLOCCHI USCENTI:

ESERCIZIO 3



- I sei istanti rilevanti sono t0-t5 e il funzionamento è dettagliato nella slide
- Il colloquio inizia da un pre-identificato "initiator" e si rivolge ad un "target" identificato dall'indirizzo "address"
- **Notare che in tale intervallo [t0-t5] il dispositivo deve restare sempre indirizzato altrimenti gli altri segnali (in particolare D e R/W, possono essere erroneamente ricevuti da altri dispositive)**

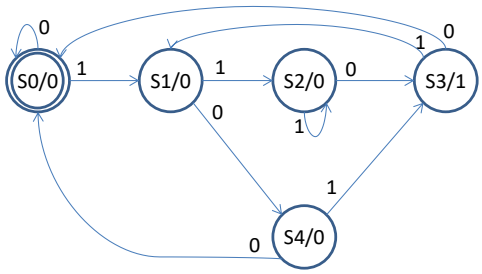
COMPITO di ARCHITETTURA DEI CALCOLATORI del 27-06-2022

SOLUZIONE

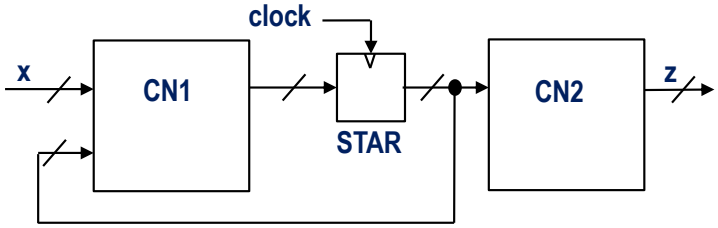
ESERCIZIO 4

In corrispondenza del pattern $X_{t-2},X_{t-1},X_t = 1,1,0$ oppure $1,0,1$ ottengo → $Z_{t+1} = 1$;
(ricordare che e' richiesto Moore).

Diagramma degli Stati



Schema di riferimento per Moore



Codice Verilog del modulo da realizzare (possibile soluzione con Moore):

```
module XXX(x,z,clock,reset_);
input        clock,reset_,x;
output z;
reg     [2:0] STAR;
parameter S0='B000,S1='B001,S2='B010,S3='B011,S4='B100;
always @(reset_==0) #1 begin STAR<=S0; end
assign z=(STAR==S3)?1:0;

always @(posedge clock) if(reset_==1) #3
casex(STAR)
      S0:begin STAR<=(x==1)?S1:S0; end
      S1:begin STAR<=(x==1)?S2:S4; end
      S2:begin STAR<=(x==1)?S2:S3; end
      S3:begin STAR<=(x==1)?S1:S0; end
      S4:begin STAR<=(x==1)?S3:S0; end
endcase
endmodule
```

Diagramma di Temporizzazione: