COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-02-2024

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).

```
double X[3][3]={{1.0, 2.0, 3.0},{4.0, 5.0, 6.0},{7.0, 8.0, 8.0}};
double mattrace(double A[3][3]) {
    double t=0.0;
    for (int i = 0; i < 3; i++) t+=A[i][i];
    return(t);
}
int main() {
    double X2[3][3];
    matsquare(X, X2);
    print_double(mattrace(X));
    print_double(mattrace(X2));
    exit(0);
}
```

Nota: 'int' è un intero a 64 bit.

```
void matsquare(double A[3][3], double R[3][3]) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            R[i][j] = 0.0;
            for (int k = 0; k < 3; k++)
                R[i][j] +=A[i][k] * A[k][j];
        }
    }
}
```

RISCV Instructions (RV64IMFD) **v230703**

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	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	---	---

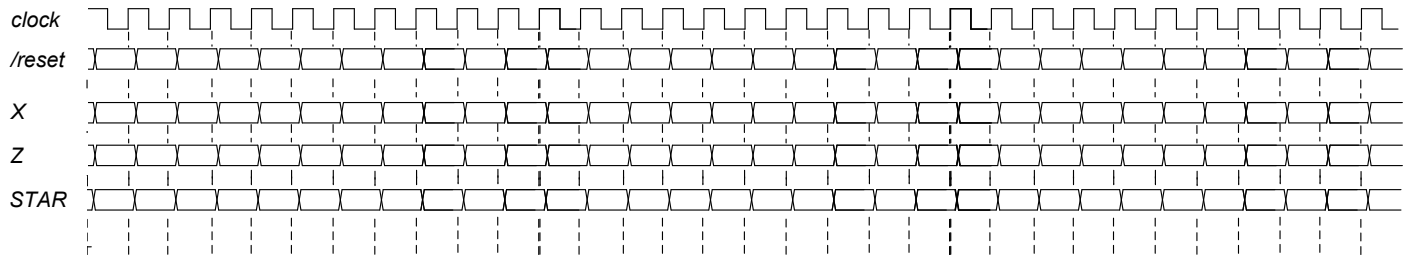
COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-02-2024

2) [5/30] Si consideri una cache di dimensione 48B e a 3 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: 477, 363, 223, 281, 400, 321, 275, 284, 482, 301, 276, 273, 476, 383, 251, 276, 401, 380. 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] Descrivere l'algoritmo CSMA/CD per l'accesso al mezzo trasmissivo nello standard Ethernet e il relativo algoritmo di back off.

4) Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Moore con un ingresso X su due bit e una uscita Z su un bit che funziona nel seguente modo: devono essere riconosciute le sequenze non-interallacciate 11,10,00,01,11; l'uscita Z va a 1 (per 1 ciclo di clock) se è presente tale sequenza. Gli stimoli di ingresso sono dati dal seguente modulo Verilog Testbench [11/30 di cui 4/30 per il diagramma di temporizzazione corretto].

   Tracciare il diagramma di temporizzazione [4/10 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[1:0] 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<=3; @(posedge clock); X<=2;
@(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=3; @(posedge clock); X<=1;
@(posedge clock); X<=1; @(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=3;
@(posedge clock); X<=2; @(posedge clock); X<=1; @(posedge clock); X<=3; @(posedge clock); X<=2;
@(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=3; @(posedge clock); X<=0;
$finish;
end
XXX Xxx(X,Z,clock,reset_);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-02-2024

SOLUZIONE

ESERCIZIO 1

```
.data
X:.double 1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,8.0
.text
.globl main
#-----------------------------------------
mattrace: # in: a0=&A[3][3], out: fa0=t
  fmv.d.x fa0,x0     # t=0.0
  li   t0,0          # i = 0
  li   t6,3          # loop limit
mt_for_start:
  beq  t0,t6, mt_for_end  # if i == 3, exitfor
  fld  ft0,0(a0)     # load A[i][i]
  addi a0,a0,32      # next &A[i][i]
  fadd.d fa0,fa0,ft0# t+=A[i][i]
  addi t0, t0, 1     # i++
  b mt_for_start
mt_for_end:
  ret                # return fa0=t
#-----------------------------------------
matsquare:#in: a0=&A[3][3] a1=&R[3][3]
  li   t0,0          # i = 0
  li   t6,3          # loop limit
ms_for1_start:
  beq  t0,t6,ms_for1_end  # if i == 3, for1end
  li   t1,0          # j = 0
ms_for2_start:
  beq  t1,t6,ms_for2_end  # if j == 3, for2end
  # R[i][j]=0.0
  add  t3,t0,t0      # 2*i
  add  t3,t3,t0      # 3*i
  add  t3,t3,t1      # 3*i+j
  slli t3,t3,3       # offset=(3i+j)*8
  add  t3,t3,a1      # &R[i][j]
  sd   x0,0(t3)      # R[i][j]=0.0
  # R[i][j] += A[i][k] * A[k][j] for k = 0 to 2
  li t2,0            # k = 0
ms_for3_start:
  beq  t2,t6,ms_for3_end  # if k == 3, for3end
  add  t4,t0,t0      # 2*i
  add  t4,t4,t0      # 3*i
  add  t4,t4,t2      # 3*i+k
  slli t4,t4,3       # offset=(3i+k)*8
  add  t4,t4,a0      # &A[i][k]
  fld  ft0,0(t4)     # A[i][k]
  add  t4,t2,t2      # 2*k
  add  t4,t4,t2      # 3*k
  add  t4,t4,t1      # 3*k+j
  slli t4,t4,3       # offset=(3k+j)*8
  add  t4,t4,a0      # &A[k][j]
  fld  ft1,0(t4)     # A[k][j]
  fmul.d ft0,ft0,ft1 #A ik*Akj
  # Add A[i][k] * A[k][j] to R[i][j]
  fld  ft2,0(t3)     # R[i][j]
  fadd.d ft2,ft2,ft0 #Rij+=Aik*Akj
  fsd  ft2,0(t3)
  addi t2, t2,1      # k++
  b ms_for3_start
ms_for3_end:
  addi t1,t1,1       # j++
  b ms_for2_start
ms_for2_end:
  addi t0,t0,1       # i++
  b ms_for1_start
ms_for1_end:
  ret
#-----------------------------------------
main:
  addi sp,sp,-72# allocate frame (for X2[3][3]
  la   a0,X
  mv   a1,sp    # X2 is allocated in the frame
  call matsquare
  la   a0,X
  call mattrace
  li   a7,3     # print double in fa0
  ecall
  mv   a0,sp    # X2 is allocated in the frame
  call mattrace
  li   a7,3     # print double in fa0
  ecall
  li   a7,10    # exit
  ecall         # desregarding the frame
```

Run I/O

14.0244.0

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=48/8/3, XM=X/B, XS=XM%S, XT=XM/S.
A=3, B=8, C=48, RP=LRU, Thit=4, Tpen=40, 18 references:

=== T	X	XM	XT	XS	XB	H	[SET]:USAGE	[SET]:MODIF	[SET]:TAG
=== R	477	59	29	1	5	0	[1]:2,0,0	[1]:0,0,0	[1]:29,-,-
=== W	363	45	22	1	3	0	[1]:1,2,0	[1]:0,0,0	[1]:29,22,-
=== R	223	27	13	1	7	0	[1]:0,1,2	[1]:0,0,0	[1]:29,22,13
=== W	281	35	17	1	1	0	[1]:2,0,1	[1]:0,0,0	[1]:17,22,13
=== R	400	50	25	0	0	0	[0]:2,0,0	[0]:0,0,0	[0]:25,-,-
=== W	321	40	20	0	1	0	[0]:1,2,0	[0]:0,0,0	[0]:25,20,-
=== R	275	34	17	0	3	0	[0]:0,1,2	[0]:0,0,0	[0]:25,20,17
=== W	284	35	17	1	4	1	[1]:2,0,1	[1]:1,0,0	[1]:17,22,13
=== R	482	60	30	0	2	0	[0]:2,0,1	[0]:0,0,0	[0]:30,20,17
=== W	301	37	18	1	5	0	[1]:1,2,0	[1]:1,0,0	[1]:17,18,13
=== R	276	34	17	0	4	1	[0]:1,0,2	[0]:0,0,0	[0]:30,20,17
=== W	273	34	17	0	1	1	[0]:1,0,2	[0]:0,0,1	[0]:30,20,17
=== R	476	59	29	1	4	0	[1]:0,1,2	[1]:1,0,0	[1]:17,18,29
=== W	383	47	23	1	7	0	[1]:2,0,1	[1]:0,0,0	[1]:23,18,29
=== R	251	31	15	1	3	0	[1]:1,2,0	[1]:0,0,0	[1]:23,15,29
=== W	276	34	17	0	4	1	[0]:1,0,2	[0]:0,0,1	[0]:30,20,17
=== R	401	50	25	0	1	0	[0]:0,2,1	[0]:0,0,1	[0]:30,25,17
=== W	380	47	23	1	4	1	[1]:2,1,0	[1]:1,0,0	[1]:23,15,29

LISTA BLOCCHI USCENTI:

(out: XM=59 XT=29 XS=1)

(out: XM=50 XT=25 XS=0)
(out: XM=45 XT=22 XS=1)

(out: XM=27 XT=13 XS=1)
(out: XM=35 XT=17 XS=1)
(out: XM=37 XT=18 XS=1)

(out: XM=40 XT=20 XS=0)

CONTENUTI dei SET al termine

P1 Nmiss=13 Nhit=5 Nref=18 mrate=0.722222 AMAT=th+mrate*tpen=32.8889

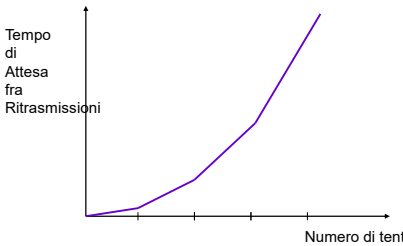
ESERCIZIO 3

CSMA/CD, Carrier Sense Multiple Access with Collision Detection

- Un nodo "ascolta prima di parlare" (carrier sense)
- Se il mezzo e' libero si attende un tempo "Defer Time"
- Se ci sono collisioni (multiple access) l'energia sul mezzo aumenta
- Un nodo cerca di rilevare collisioni durante la trasmissione (collision detect)
- Se un nodo rileva una collisione, continua a trasmettere 4-6 bytes per essere sicuro che la collisione venga rilevata dagli altri nodi e poi "lascia" il mezzo
- Se un nodo ha rilevato collisione, la trasmissione viene ripetuta dopo R*Δt per altri 15 tentativi
- Dopo 16 tentativi l'errore viene segnalato ai livelli superiori
- Δt e' fisso mentre R viene calcolato secondo un "algoritmo di back off"

Algoritmo di back off per il calcolo di R

- Sia n l'n-esimo tentativo di trasmissione (n=1,2,…, 15)
- K= min(n,10)
- R e' un numero casuale t.c. 0 <= R <= 2^K



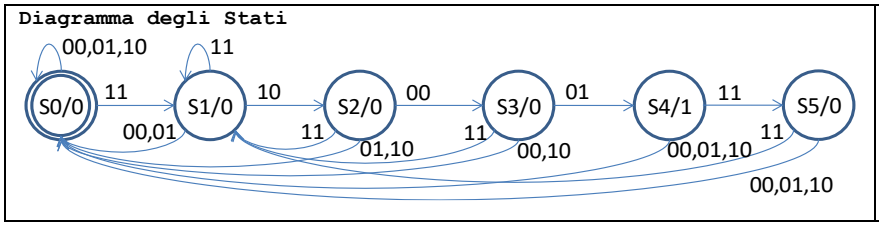
Risolve in maniera probabilisticamente accettabile i conflitti fino a 1024 nodi…

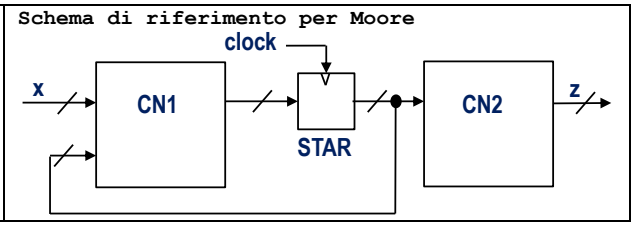
COMPITO di ARCHITETTURA DEI CALCOLATORI del 14-02-2024

SOLUZIONE

ESERCIZIO 4

In corrispondenza del pattern X_{t-4}, X_{t-3}, X_{t-2},X_{t-1},X_t = 11,10,00,01,11 NON INTERALLACCIATO ottengo → Z_{t+1} = 1; (ricordare che e' richiesto Moore).





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

```
module XXX(X,z,clock,reset_);
  input[1:0] X;
  input clock,reset_;
  output z;
  reg[2:0] STAR;
  parameter S0=0,S1=1,S2=2,S3=3,S4=4,S5=5;

  always @(reset_==0)#1 begin STAR<=S0; end
  assign z=(STAR==S5)?1:0;

  always @(posedge clock) if(reset_==1) #3
  casex(STAR)
    S0: begin STAR<=(X==2'b11)?S1:S0; end
    S1: begin STAR<=(X==2'b10)?S2:(X==2'b11)?S1:S0; end
    S2: begin STAR<=(X==2'b00)?S3:(X==2'b11)?S1:S0; end
    S3: begin STAR<=(X==2'b01)?S4:(X==2'b11)?S1:S0; end
    S4: begin STAR<=(X==2'b11)?S5:S0; end
    S5: begin STAR<=(X==2'b11)?S1:S0; end
  endcase
endmodule
```

Diagramma di Temporizzazione: