COMPITO di ARCHITETTURA DEI CALCOLATORI del 26-06-2024

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) [12/30] 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).

```
struct Node { int data; struct Node* next; };
struct Node* head = NULL;

void swap(struct Node* a, struct Node* b) {
    int temp = a->data; a->data = b->data; b->data = temp;
}
void insert(struct Node** head, int data) {
    struct Node* newNode = sbrk(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

void bubbleSort(struct Node* head) {
    int swapped;
    struct Node *ptr1, *ptr0 = NULL;
    if (head == NULL) return;
    do {
        swapped = 0; ptr1 = head;
        while (ptr1->next != ptr0) {
            if (ptr1->data > ptr1->next->data) {
                swap(ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        ptr0 = ptr1;
    } while (swapped);
}

int main() {
    insert(&head, 62); insert(&head, 14); insert(&head, 25);
    bubbleSort(head);
    print_int(head->data); exit(0);
}
```

Nota: 'int' è un intero a 64 bit.

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)*sgn(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

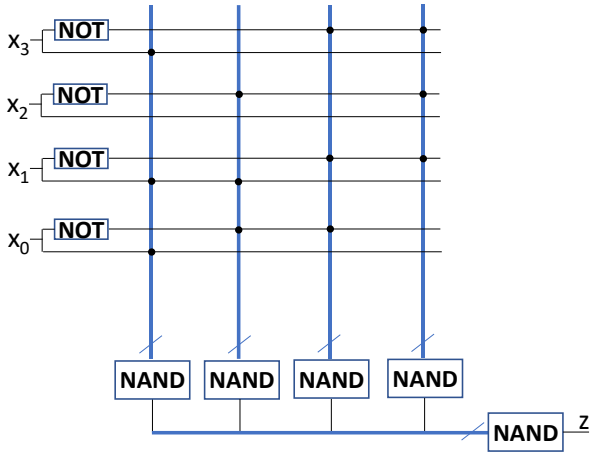
System calls

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	---	---

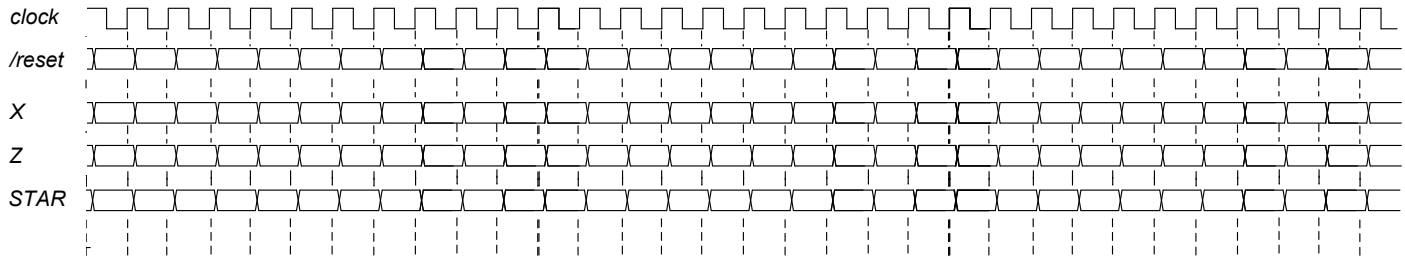
2) [5/30] Si consideri una cache di dimensione 48B e a 3 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 27, 13, 63, 11, 40, 61, 15, 124, 822, 141, 16, 113, 16, 23, 91, 216, 31, 210. 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] Data la seguente rete combinatoria: i) disegnare la mappa di Karnaugh; ii) inserendo in tale mappa dei non-specificato (simbolo 'X') in corrispondenza degli ingressi $x_3\ x_2\ \overline{x_1}\ \overline{x_0}$ $x_3\ \overline{x_2}\ \overline{x_1}\ x_0$, ricavare un'equazione booleana in forma "somma di prodotti" che descriva la nuova mappa in modo da usare sottocubi di dimensione maggiore possibile:



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 interallacciate 1,1,1,1, e 1,0,0,1; 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<=1; @(posedge clock); X<=1; @(posedge clock); X<=0; @(posedge clock); X<=1;
@(posedge clock); X<=1; @(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=1;
@(posedge clock); X<=1; @(posedge clock); X<=1; @(posedge clock); X<=1; @(posedge clock); X<=1;
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=0;
$finish;
end
XXX Xxx(X,Z,clock,reset_);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 26-06-2024

SOLUZIONE

ESERCIZIO 1

```
.data
head: .dword 0
.text
.globl main
#-----------------------------------------
swap:#in: a0=a, a1=b
  # Load a->data into t0
  ld   t0,0(a0) # a->data
  ld   t1,0(a1) # b->data
  sd   t1, 0(a0)# b->data=a->data
  sd   t0, 0(a1)# a->data=b->data
  ret
#-----------------------------------------
insert:#in: a0=&head, a1=data
  mv   t0,a0    # save a0=head
  li   a0, 16   # sizeof(struct Node)
  li   a7, 9    # Allocate memory for a new Node
  ecall
  sd   a1,0(a0) # Store data in the new Node
  ld   t2,0(t0) # *head
  sd   t2,8(a0) # Store next pointer of the new
Node
  sd   a0,0(t0) # Link the new Node on head
  ret
#-----------------------------------------
bubblesort:#in: a0=head
  addi sp,sp,-48# allocate frame
  sd   s0, 0(sp)
  sd   s1, 8(sp)
  sd   s2,16(sp)
  sd   s3,24(sp)
  sd   s9,32(sp)
  sd   ra,40(sp)
  li   s0,0     # s0 = ptr0 = NULL
  beq  a0,x0,bs_end # if head == NULL -->end
  mv   s9,a0    # save s9=a0=head
bs_do_start:
  li   s2,0     # s2 = swapped = 0
  mv   s1,s9    # s1 = ptr1 = head
bs_while_start:
  ld   s3,8(s1) # s3 = ptr1->next
  beq  s3,s0,bs_while_end # ptr1->next!=ptr0 --
>whend
  ld   t0,0(s1) # t0 = ptr1->data
  ld   t1,0(s3) # t1 = ptr1->next->data
  slt  t6,t1,t0 # t1<?t0 false-->ifend
  beq  t6,x0,bs_if_end
  mv   a0,s1    # a0=ptr1
  mv   a1,s3    # a1=ptr1->next
  call swap
  li   s2,1     # swapped = 1
bs_if_end:
  mv   s1,s3    # ptr1 = ptr1->next
  b bs_while_start
bs_while_end:
  mv   s0,s1    # ptr0 = ptr1
  bne  s2,x0,bs_do_start #swapped!= ->dostart
bs_end:
  ld   s0, 0(sp)
  ld   s1, 8(sp)
  ld   s2,16(sp)
  ld   s3,24(sp)
  ld   s9,32(sp)
  ld   ra,40(sp)
  addi sp,sp,48  # deallocate frame
  ret
#-----------------------------------------
main:
  la   a0,head
  li   a1,62
  call insert
  la   a0,head
  li   a1,14
  call insert
  la   a0,head
  li   a1,25
  call insert
  la   a0,head   # &ead
  ld   a0,0(a0)  # head
  call bubblesort
  la   a0,head   # &head
  ld   a0,0(a0)  # head
  ld   a0,0(a0)  # head->data
  li   a7,1      # print int
  ecall
  li   a7,10     # exit
  ecall
```

Run I/O

```
14
```

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   27    3   1   1   3  0 [1]:2,0,0 [1]:0,0,0  [1]:1,-,-
=== W   13    1   0   1   5  0 [1]:1,2,0 [1]:0,0,0  [1]:1,0,-
=== R   63    7   3   1   7  0 [1]:0,1,2 [1]:0,0,0  [1]:1,0,3
=== W   11    1   0   1   3  1 [1]:0,2,1 [1]:0,1,0  [1]:1,0,3
=== R   40    5   2   1   0  0 [1]:2,1,0 [1]:0,1,0  [1]:2,0,3
=== W   61    7   3   1   5  1 [1]:1,0,2 [1]:0,1,1  [1]:2,0,3
=== R   15    1   0   1   7  1 [1]:0,2,1 [1]:0,1,1  [1]:2,0,3
=== W  124   15   7   1   4  0 [1]:2,1,0 [1]:0,1,1  [1]:7,0,3
=== R  822  102  51   0   6  0 [0]:2,0,0 [0]:0,0,0  [0]:51,-,-
=== W  141   17   8   1   5  0 [1]:1,0,2 [1]:0,1,0  [1]:7,0,8
=== R   16    2   1   0   0  0 [0]:1,2,0 [0]:0,0,0  [0]:51,1,-
=== W  113   14   7   0   1  0 [0]:0,1,2 [0]:0,0,0  [0]:51,1,7
=== R   16    2   1   0   0  1 [0]:0,2,1 [0]:0,0,0  [0]:51,1,7
=== W   23    2   1   0   7  1 [0]:0,2,1 [0]:0,1,0  [0]:51,1,7
=== R   91   11   5   1   3  0 [1]:0,2,1 [1]:0,0,0  [1]:7,5,8
=== W  216   27  13   1   0  0 [1]:2,1,0 [1]:0,0,0  [1]:13,5,8
=== R   31    3   1   1   7  0 [1]:1,0,2 [1]:0,0,0  [1]:13,5,1
=== W  210   26  13   0   2  0 [0]:2,1,0 [0]:0,1,0  [0]:13,1,7
-------------------------------------
P1 Nmiss=13   Nhit=5   Nref=18   mrate=0.722222   AMAT=th+mrate*tpen=32.8889
```

LISTA BLOCCHI USCENTI:

```
(out: XM=3   XT=1   XS=1 )

(out: XM=5   XT=2   XS=1 )

(out: XM=7   XT=3   XS=1 )

(out: XM=1   XT=0   XS=1 )
(out: XM=15  XT=7   XS=1 )
(out: XM=17  XT=8   XS=1 )
(out: XM=102 XT=51  XS=0 )
```

CONTENUTI dei SET al termine

ESERCIZIO 3

La rete combinatoria si sintetizza con: $z = \overline{(\overline{x_3}\ x_1\ x_0)\ (\overline{x_2}\ x_1\ \overline{x_0})\ (\overline{x_3}\ \overline{x_1}\ \overline{x_0})\ (\overline{x_3}\ \overline{x_2}\ \overline{x_1})}$ ovvero $\bar{z} = (\overline{x_3} + \overline{x_1} + \overline{x_0})\ (x_2 + \overline{x_1} + x_0)\ (x_3 + x_1 + x_0)\ (x_3 + x_2 + x_1)$ →disegniamo la mappa di Karnaugh osservando che la precedente espressione a destra individua i maxtermini ovvero degli " 0" in quanto sto calcolando $\bar{z}$

$x_1 x_0$ \ $x_3 x_2$	00	01	11	10
00	0	0	1	1
01	0	1	1	1
11	1	1	0	0
10	0	1	1	0

← mappa di Karnaugh per $\bar{z}$.

ovvero, passando a z e inserendo i non specificati in corrisponenza degli ingressi:

$x_3\ x_2\ \overline{x_1}\ \overline{x_0}$ $x_3\ \overline{x_2}\ \overline{x_1}\ x_0$ →

$x_1 x_0$ \ $x_3 x_2$	00	01	11	10
00	1	1	X	0
01	1	0	0	X
11	0	0	1	1
10	1	0	0	1

3 4 1 2

I non-specificato quindi non aiutano per niente.
E usando sottocubi di dimensione maggiore possibile la funzione rimane la stessa:

$z = x_3 x_1\ x_0 + \overline{x_2}\ x_1 \overline{x_0} + \overline{x_3}\ \overline{x_1}\ \overline{x_0} + \overline{x_3}\ \overline{x_2}\ \overline{x_1}$

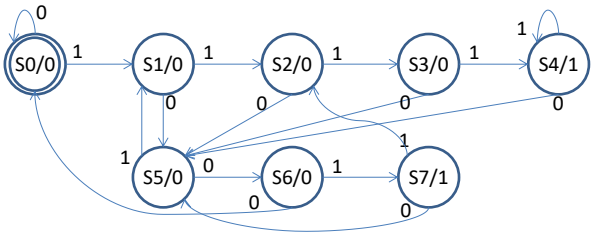
COMPITO di ARCHITETTURA DEI CALCOLATORI del 26-06-2024

SOLUZIONE

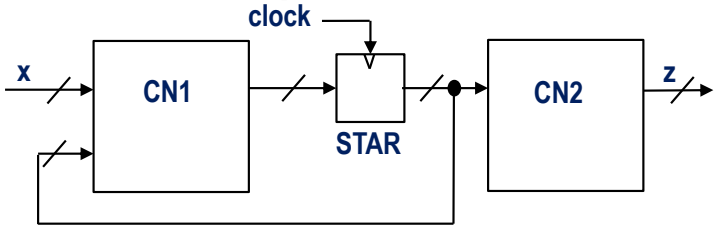
ESERCIZIO 4

In corrispondenza del pattern $X_{t-3}, X_{t-2}, X_{t-1}, X_t$ = 1,1,1,1 oppure 1,0,0,1 INTERALLACCIATI 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 X;
  input clock,reset_;
  output z;
  reg[2:0] STAR;
  parameter S0=0,S1=1,S2=2,S3=3,S4=4,S5=5,S6=6,S7=7;

  always @(reset_==0)#1 begin STAR<=S0; end
  assign z=(STAR==S4)?1:(STAR==S7)?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:S5; end
    S2: begin STAR<=(X==1)?S3:S5; end
    S3: begin STAR<=(X==1)?S4:S5; end
    S4: begin STAR<=(X==1)?S4:S5; end
    S5: begin STAR<=(X==1)?S1:S6; end
    S6: begin STAR<=(X==1)?S7:S0; end
    S7: begin STAR<=(X==1)?S2:S5; end
  endcase
endmodule
```

Diagramma di Temporizzazione: