COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-04-2023

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).

```
void qsort2(float *array, int from, int to) {
   if(from>=to)return;
   float pivot = array[from];
   int i = from, j;
   float temp;
   for(j = from + 1; j <= to; j++) {
      if(array[j] < pivot) {
         i = i + 1;
         temp = array[i];
         array[i] = array[j];
         array[j] = temp;
      }
   }
   temp = array[i];
   array[i] = array[from];
   array[from] = temp;
   qsort2(array,from,i-1);
   qsort2(array,i+1,to);
}
```

Nota: 'int' è un intero a 64 bit.

```
int main() {
   int number_of_elements;
   read_int(&number_of_elements);
   float array[number_of_elements];
   int iter;
   for(iter = 0;iter < number_of_elements;iter++) {
      read_float(&array[iter]);
   }
   qsort2(array,0,number_of_elements-1);
   for(iter = 0;iter < number_of_elements;iter++) {
      print_float(array[iter]);
      print_string(" ");
   }
   print_string("\n");
   exit(0);
}
```

RISCV Instructions (RV64IMFD) **v221117**

Instruction coding (hexadecimal)			Instruction	Example	Register operation	Meaning
funct7/imm	funct3	opcode				(** instructions available only in RV64, i.e. 64-bit case)
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 INST.:** `lui x5,H20(&var);ori x5, L12(&var)` (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 21-04-2023

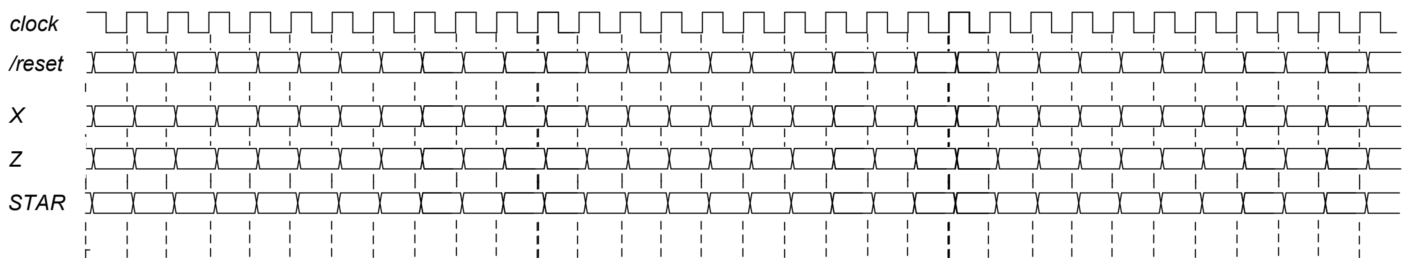
SOLUZIONE

2) [5/30] Rappresentare il numero 3.4 in un formato IEEE-754 singola precisione. L'arrotondamento deve essere effettuato al numero più vicino rappresentabile e in caso di equidistanza arrotondare al valore pari (round to nearest, ties to even); sviluppare il calcolo illustrando come si ottiene il risultato.

3) [5/30] Descrivere la procedura di programmazione della porta seriale 16550A, mappata a partire dall'indirizzo 0x900003E8, per ottenere 1 bit di stop, 8 bit di trama, parità dispari di uni (no break), baud rate pari a 19200. Imponendo una frequenza esterna di clock di tale chip pari a 1.8432 MHz, quale sono gli indirizzi a 32 bit (in esadecimale) dei registri da programmare secondo queste specifiche? E quali i valori da scrivere in tali registri?

4) [10/30] 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 1,1,1,1, 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 [4/10 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 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<=0; @(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 21-04-2023

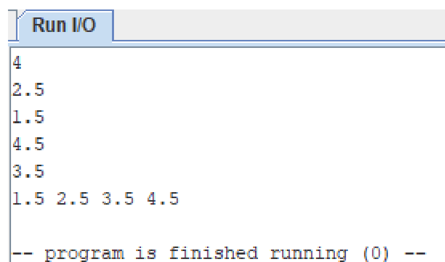
SOLUZIONE

ESERCIZIO

```
.data
newline: .asciz "\n"
space:   .asciz " "
.text
.globl  main

# a0=array, a1=from, a2=to (tutti int64)
qsort2:
    addi sp,sp,-40 # alloca frame
    sd   ra,32(sp) # salva ra (f. ricorsiva)
    sd   s0,24(sp) # salva gli sx che uso
    sd   s1,16(sp)
    sd   s2, 8(sp)
    sd   s3, 0(sp)
    slt  t0,a1,a2  # from <? to
    beq  t0,x0,q_fine # esci se falso
    mv   s0,a0     # salva &array
    mv   s1,a1     # salva 'from'
    mv   s2,a2     # salva 'to'
    slli t0,s1,2   # 'from' * sizeof(float)
    add  t0,s0,t0  # &array[from]
    flw  fa0,0(t0) # pivot= array[from]
    mv   s3,s1     # s3=i = from
    addi t2,s1,1   # j = from + 1
q_for_start:
    slt  t0,s2,t2  # to <? j
    bne  t0,x0,q_for_end # salta se vero
    slli t1,t2,2   # j * sizeof(float)
    add  t1,s0,t1  # &array[j]
    flw  fa1,0(t1) # fa1= array[j]
    flt.s t0,fa1,fa0# array[j] <? pivot
    beq  t0,x0,q_if_end
    addi s3,s3,1   # i=i+1
    slli t3,s3,2   # i * sizeof(float)
    add  t3,s0,t3  # &array[i]
    flw  fa2,0(t3) # fa0=temp= array[i]

    fsw  fa1,0(t3) # array[i]= array[j]
    fsw  fa2,0(t1) # array[j]= temp
q_if_end:
    addi t2,t2,1   #j++
    b q_for_start
q_for_end:
    slli t3,s3,2   # i * sizeof(float)
    add  t3,s0,t3  # &array[i]
    flw  fa2,0(t3) # fa0=temp= array[i]
    slli t0,s1,2   # from * sizeof(float)
    add  t0,s0,t0  # &array[from]
    flw  fa1,0(t0) # fa1= array[from]
    fsw  fa1,0(t3) # array[i]= array[from]
    fsw  fa2,0(t0) # array[from]= temp
    mv   a0,s0     # &array
    mv   a1,s1     # from
    addi a2,s3,-1  # i-1
    call qsort2
    mv   a0,s0     # &array
    addi a1,s3,1   # i+1
    mv   a2,s2     # to
    call qsort2
q_fine:
    ld   ra,32(sp) # ripristina ra
    ld   s0,24(sp) # ripristana gli sx
    ld   s1,16(sp)
    ld   s2, 8(sp)
    ld   s3, 0(sp)
    addi sp,sp,40  # dealloca frame
    ret
#-----------------------------------------------
main:
    li   a7,5      # leggi int --> a0
    ecall
    mv   s1,a0     # salva number_of_elements
    slli s3,s1,2   # (.) * sizeof(float)
    sub  sp,sp,s3  # alloca 'array nello stack
    mv   s2,sp     # salva ind. base di array
    li   t1,0         # iter = 0
m_for1_start:
    slt  t0,t1,s1  # iter <? number_of...
    beq  t0,x0,m_for1_end
    slli t2,t1,2   # iter * sizeof(float)
    add  t2,s2,t2  # &array[iter]
    li   a7,6      # leggi float --> fa0
    ecall
    fsw  fa0,0(t2) # array[iter]= fa0
    addi t1,t1,1   # iter++
    b m_for1_start
m_for1_end:
    mv   a0,s2     # &array
    li   a1,0      # 2nd param.
    addi a2,s1,-1  # number_of... - 1
    call qsort2
    li   t1,0      # iter = 0
m_for2_start:
    slt  t0,t1,s1  # iter <? number_of...
    beq  t0,x0,m_for2_end
    slli t2,t1,2   # iter * sizeof(float)
    add  t2,s2,t2  # &array[iter]
    flw  fa0,0(t2) # fa0= array[iter]
    li   a7,2      # stampa float
    ecall
    la   a0,space  # " "
    li   a7,4      # stampa stringa
    ecall
    addi t1,t1,1   # iter++
    b m_for2_start
m_for2_end:
    la   a0,newline# \n
    li   a7,4      # stampa stringa
    ecall
    add  sp,sp,s3  # dealloca array da stack
    li   a7,10     # exit
    ecall
```



ESERCIZIO 2

Il formato IEEE-754 singola precision prevede 1 bit di segno, 8 bit di esponente e 23 bit di mantissa . Normalizzando 3.4 si ottiene: 3.4=1.7*2^(+1) ovvero m=1.7, e=+1. Ricaviamo quindi S,M,E.

L' "uno" iniziale non viene rappresentato nel formato quarter-IEEE-754. Essendo m = 1.7=si ha:

M = m-1= 0.7

Successivamente si puo' ricavare la rappresentazione binaria di M con 23 bit moltiplicando per 2 e ricavando via via la n-esima cifra più significativa: 0.7*2=1.4 → **1**, 0.4*2=0.8→ **0**, 0.8*2=1.6 → **1**, 0.6*1=1.2 → **1**, 0.2*2=0.4 → **0**, ...e questo punto si ripete 0,1,1,0 e così via...
(notare che NON si deve mai troncare il numero: occorre mantenere tutte le cifre periodiche); in altri termini, dopo la prima cifra binaria le cifre si ripetono, quindi e' facilmente predicibile il resto delle cifre binarie della mantissa. Inoltre, la 24-esima cifra della mantissa corrisponde ad uno 1 quindi il più vicino numero rappresentabile in IEEE-754 singola precisione è quello che si ottiene aggiungendo un 1 alla 23-esima cifra, ovvero:
M(da arrotondare)=1 0110 0110 0110 0110 0110 0110... ovvero M = 1 0110 0110 0110 0110 0110 10

Per l'esponente, ricordando che nel caso di singola precisione il valore della polarizzazione e' 127:

E = e+127 = +1+127=128 ovvero 1000 0000

Inoltre, per il segno S = 0

Quindi la rappresentazione cercata e':

0b 0100 0000 0101 1001 1001 1001 1001 1010 (in esadecimale 0x4059 999A)

ESERCIZIO 3

Visione logica: DLM e DLL

- **Avendo posto a 1 il bit di accesso ai Divisor Latches (DLAB=1) posso scrivere/leggere in DLM,DLL il valore della costante C che determina il bit-rate**
  - DLM,DLL vengono interpretati come un intero senza segno a 16 bit
  - Detta FCK la frequenza del clock esterno e BR il bit-rate desiderato, si ha

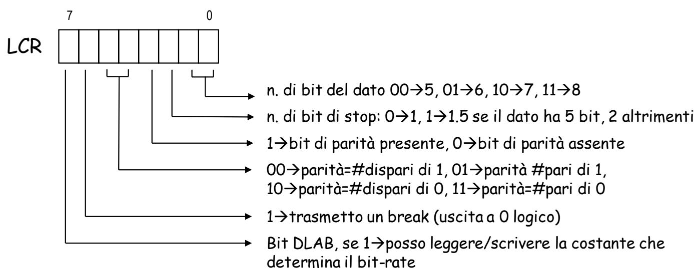
$$C = \frac{Fc}{16 \cdot BR}$$

Nota: nei PC Fc=1.8432MHz
Es. per ottenere un BR=9600 →C=12

BR	C
300	384
1200	96
4800	24
9600	12
115200	1

- **DLAB non è altro che il bit 7 di LCR**
- **I registri RBR, THR, IER, IIR, FCR si accedono ponendo DLAB=0**
- **Per i restanti registri il valore di DLAB è ininfluente**

Visione logica: LCR, Line Control Register



Ricordando che è necessario programmare i registri LCR, DLL e DLM che hanno un offset pari a 3, 0 e 1 rispettivamente rispetto all'indirizzo base della porta seriale, i rispettivi indirizzi a 32 bit risultano 0x9000'03EB, 0x9000'03E8, 0x9000'03E9, rispettivamente. Per il registro LCR il valore da scrivere risulta in binario 1000'1011 (0x8B); successivamente andrà scritta la costante di tempo C=1843200/16/19200=6 (0x0006) ovvero dovremo scrivere il valore 0x00 in DLM e il valore 0x06 in DLL.

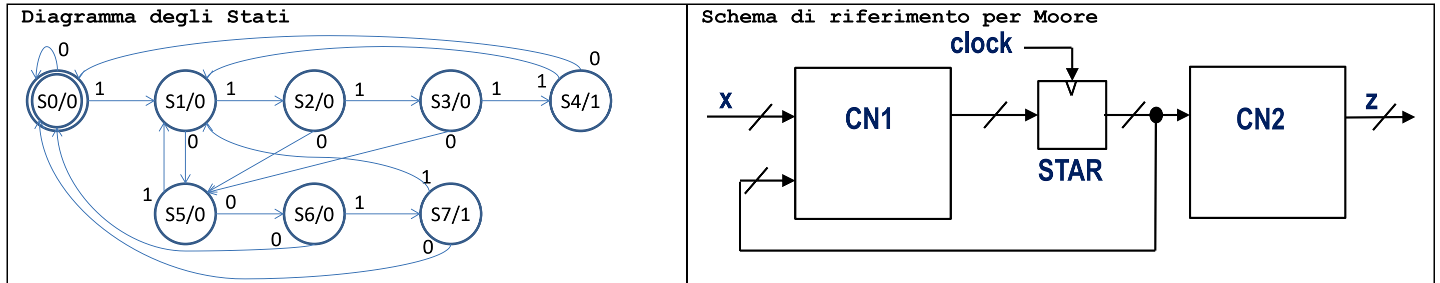
COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-04-2023

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 ottengo → Z_{t+1} = 1;
(ricordare che e' richiesto Moore).

NOTA: altre soluzioni anche piu' ottimizzate sono possibili, ma le ottimizzazioni FSM non sono nel programma di questo insegnamento.



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)?S1:S0; end
    S5: begin STAR<=(X==1)?S1:S6; end
    S6: begin STAR<=(X==1)?S7:S0; end
    S7: begin STAR<=(X==1)?S1:S0; end
  endcase
endmodule
```

Diagramma di Temporizzazione: