COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-04-2021

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) [9/30] 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).

```
static int X[200], Z[200];
static int Loop3(int n, int *X, int *Y) {
 int i, Q;
 Q = 0;
 for (i = 0, i < n; ++i) {
    Q += X[i] * Y[i];
 }
 return Q;
}
```

```
main()
{
 int n, k, i, V;
 n = 100; V = 0;
 for (k = 0; k < n; ++k) {
    X[k] = 222; Z[k] = 222;
 }
 for (i = 0; i < n; ++i) {
    V += Loop3(n, X, Z);
 }
 print_int(V);
 print_string("\n");
}
```

RISCV Instructions (RV64IMFD) **v191222**

Instruction coding (hexadecimal) opcode+funct3+{funct7,imm}	Instruction	Example		Meaning	Comments (** 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/lbu`	`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	Load address of var (lui x5,H20(&var);ori x12, L12(&var)) H20=high 20 bit of &var; L12=low 12 bits of &var
PSEUDOINSTRUCTION	**jump**	`j/b`	`1000`	go to 1000	`(PSEUDO) INSTR. IS: jal x0,offset/beq x0,x0,offset`
PSEUDOINSTRUCTION	**jump and link (offset)**	`jal`	`100`	x1← (PC + 4); go to PC+100	`(PSEUDO) INSTR. IS: jal x1,offset`
PSEUDOINSTRUCTION	**return from procedure**	`ret`		PC←x1	`(PSEUDO) INSTR. IS: jalr x0,0(x1)`
67+0+imm	**jump and link register**	`jalr`	`x1, 100(x5)`	x1 ← (PC + 4); go to 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`		call OS service number in a7	See table of system calls below
73+0+8 (rs1=0,rs2=2,rd=0)	**sret**	`sret`		Exit Supervisor mode	-
PSEUDOINSTRUCTION	**move**	`mv`	`x5,x6`	x5 ← x6	`(PSEUDO) INSTR. IS: add x5,x0,x6`
PSEUDOINSTRUCTION	**load immediate**	`li`	`x5,100`	x5 ← 100	`(PSEUDO) INSTR. IS: addi x5,x0,100`
PSEUDOINSTRUCTION	**no operation (nop)**	`nop`		do nothing	`(PSEUDO) INSTR. IS: addi x0,x0,0`
53+0+{0,1}/53+0+{4,5}	**floating point add/sub**	`fadd.{s,d}/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.{s,d}/fdiv.{s,d}`	`f0,f1,f2`	f0←f1*f2 / f0←f1/f2	Single or double precision multiplication / division
53+2+{10,11}	**floating point absolute value**	`fabs.{s,d}`	`f0,f1`	f0← | f1 |	`(PSEUDO) INSTR. IS: fsgnjx.{s,d} f0,f1`
53+0+{10,11}	**floating point move between f-regs**	`fmv.{s,d}`	`f0,f1`	f0←f1	`(PSEUDO) INSTR. IS: fsgnj.{s,d} f0,f1`
53+1+{10,11}	**floating point negate**	`fneg.{s,d}`	`f0,f1`	f0← – (f1)	`(PSEUDO) INSTR. IS: fsgnjn.{s,d} f0,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}	**move between x (integer) and f regs**	`fmv.x.{s,d}`	`x5,f0`	x5←f0 (no conversion)	Copy (no conversion)
53+0+{78,79}	**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=1)	**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}	**convert to integer from {single,double}**	`fcvt.w.{s,d}`	`x5,f0`	x5← (int)f0	Type conversion
53+7+{68,69}	**convert to {single,double} from integer**	`fcvt.{s,d}.w`	`f0,x5`	f0← ({single,double})x5	Type conversion

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

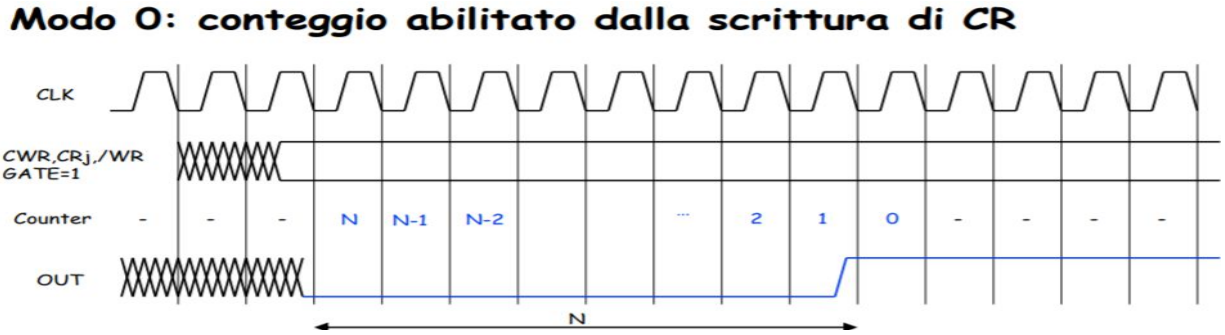
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Args
print_int	1	a0=integer to print	---
print_float	2	fa0=float to print	---
print_double	3	fa0=double to print	---
print_string	4	a0=address of ASCIIZ string to print	---
read_int	5	---	a0=integer

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 22-04-2021

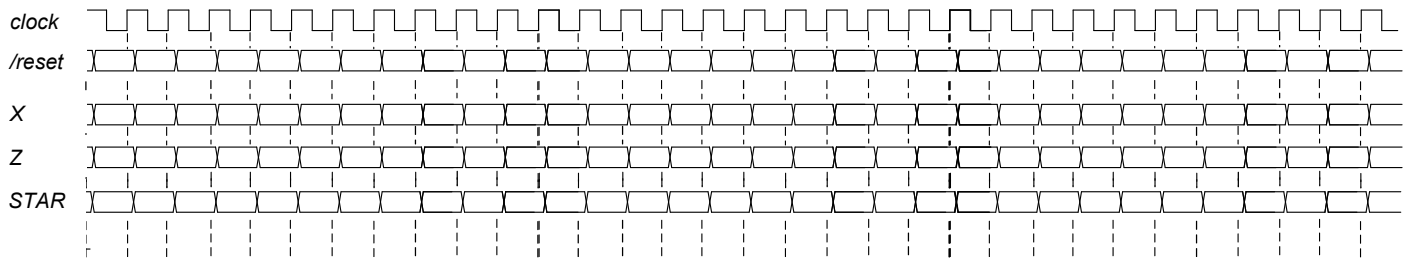
2) [6/30] Si consideri una cache di dimensione 64B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 125, 170, 167, 245, 183, 119, 235, 163, 288, 309, 310, 308, 213, 196, 377, 166, 362, 233, 163, 169. 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] Spiegare con proprie parole il funzionamento del "Modo 0" del timer 8254, il cui diagramma temporale è riportato in figura. Inoltre, indicare con precisione: i) il significato dei segnali rappresentati in tale diagramma, ii) come deve essere impostata la parola di controllo CWR e il relativo registro di conteggio per ottenere questo diagramma supponendo di utilizzare N=64001, il contatore n.0 in conteggio binario.



4) [10/30] Descrivere e sintetizzare in Verilog una rete sequenziale basata sul modello di Mealy-Ritardato con un ingresso X su un bit e una uscita Z su due bit che funziona nel seguente modo: devono essere riconosciute le sequenze non interallacciate 1,1,1,0 e 0,1,0,1; l'uscita Z[1] va a 1 se si presenta una delle due sequenze mentre Z[0] dice quale sequenza si e' presentata (Z[0]=1 se si presenta 1,1,1,0; Z[0]=0 altrimenti). Rappresentare la macchina a stati finiti per tale rete di Mealy-Ritardato e t**racciare il diagramma di temporizzazione** come verifica della correttezza dell'unità. Nota: si puo' svolgere l'esercizio su carta oppure (preferibilmente) con ausilio del simulatore salvando una copia dell'output (diagramma temporale) e del programma Verilog su USB-drive del docente.

```
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 [1:0] Z;
  wire [2:0] STAR=Xxx.STAR;
  wire Z1=Xxx.z[1];
  wire Z0=Xxx.z[0];
  initial begin X=0;
  wait(reset_==1); #5
  @(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;
  @(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; @(posedge clock); X<=1;
  @(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 22-04-2021 **(rev. 22/6/2021)**

SOLUZIONE

ESERCIZIO 1

```
.data
X:    .space    800        # 200 interi
Z:    .space    800        # 200 interi
nl:   .asciz    "\n"       # new-line

.text
.globl main

Loop3:
    # a3=a0=n, a1=X, a2=Y
    # t0=i, new a0=Q
    # t1=tmp, t2=tmp
    mv   a3,a0      # n
    li   a0, 0      # Q=0
    li   t0, 0      # i=0
l1:
    slt  t2, t0, a3 # t2=(i<n)
    beq  t2, zero, endl1 #Cond.fal ->termine ciclo
    slli t1, t0, 2  # t1=i<<2 (=i*4)
    add  t2, a1, t1 # t2=&X[i]
    lw   t2, 0(t2)  # t2=X[i]
    add  t3, a2, t1 # t3=&Y[i]
    lw   t3, 0(t3)  # t3=Y[i]
    mul  t4,t2, t3  # t4=X[i]*Y[i] (su 32 bit)
    add  a0, a0, t4 # Q+=X[i]*Y[i]
    addi   t0, t0, 1# i++
    b      l1       # Ripeti il ciclo
endl1:
    ret             # Esci da funzione
main:
    # s0=n, s1=k, s2=i, s3=V, s4=X, s5=Z
    # t0=tmp, t2=222
    la     s4, X    # s4=&X[0]
    la     s5, Z    # s5=&Z[0]
    li     s0, 100  # n=100
    li     s3, 0    # V=0
    li     t2, 222  # t2=222
    li     s1, 0    # k=0
l2:
    slt    t0, s1, s0# t0=(k<n)
    beq    t0, zero, endl2 #C.falsa->termine ciclo
    slli   t0, s1, 2    # t0=k<<2 (=k*4)
    add    t1, s4, t0# t1=&X[k]
    sw     t2, 0(t1) # X[k]=222
    add    t1, s5, t0# t1=&Z[k]
    sw     t2, 0(t1) # Z[k]=222
    addi   s1, s1, 1 # k++
    b      l2       # Ripeti il ciclo
endl2:
    li    s2, 0     # i=0
l3:
    slt   t0, s2, s0 # t0=i<n
    beq   t0, zero, endl3#C.falsa -> termine ciclo
    mv    a0, s0     # I argomento per Loop3
    mv    a1, s4     # II argomento per Loop3
    mv    a2, s5     # III argomento per Loop3
    jal   Loop3      # Loop3(n,X,Z)
    add   s3, s3, a0 # V+=Loop3(n,X,Z)
    addi  s2, s2, 1  # i++
    b     l3         # Ripeti il ciclo
endl3:
    mv    a0, s3     # I argomento
    li    a7, 1      # print_int()
    ecall
    la    a0, nl
    li    a7, 4      # print_str()
    ecall
    li    a7, 10     # a7=codice per la exit
    ecall            # exit()
```

Run I/O

```
492840000

-- 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=64/8/4, XM=X/B, XS=XM%S, XT=XM/S.
A=4, B=8, C=64, RP=LRU, Thit=4, Tpen=40, 20 references:

	T	X	XM	XT	XS	XB	H	[SET]:USAGE	[SET]:MODIF	[SET]:TAG	
===	R	125	15	7	1	5	0	[1]:3,0,0,0	[1]:0,0,0,0	[1]:7,-,-,-	
===	W	170	21	10	1	2	0	[1]:2,3,0,0	[1]:0,0,0,0	[1]:7,10,-,-	
===	R	167	20	10	0	7	0	[0]:3,0,0,0	[0]:0,0,0,0	[0]:10,-,-,-	
===	W	245	30	15	0	5	0	[0]:2,3,0,0	[0]:0,0,0,0	[0]:10,15,-,-	
===	R	183	22	11	0	7	0	[0]:1,2,3,0	[0]:0,0,0,0	[0]:10,15,11,-	
===	W	119	14	7	0	7	0	[0]:0,1,2,3	[0]:0,0,0,0	[0]:10,15,11,7	
===	R	235	29	14	1	3	0	[1]:1,2,3,0	[1]:0,0,0,0	[1]:7,10,14,-	
===	W	163	20	10	0	3	1	[0]:3,0,1,2	[0]:1,0,0,0	[0]:10,15,11,7	
===	R	288	36	18	0	0	0	[0]:2,3,0,1	[0]:1,0,0,0	[0]:10,18,11,7	(out: XM=30 XT=15 XS=0)
===	W	309	38	19	0	5	0	[0]:1,2,3,0	[0]:1,0,0,0	[0]:10,18,19,7	(out: XM=22 XT=11 XS=0)
===	R	310	38	19	0	6	1	[0]:1,2,3,0	[0]:1,0,0,0	[0]:10,18,19,7	
===	W	308	38	19	0	4	1	[0]:1,2,3,0	[0]:1,0,1,0	[0]:10,18,19,7	
===	R	213	26	13	0	5	0	[0]:0,1,2,3	[0]:1,0,1,0	[0]:10,18,19,13	(out: XM=14 XT=7 XS=0)
===	W	196	24	12	0	4	0	[0]:3,0,1,2	[0]:0,0,1,0	[0]:12,18,19,13	(out: XM=20 XT=10 XS=0)
===	R	377	47	23	1	1	0	[1]:0,1,2,3	[1]:0,0,0,0	[1]:7,10,14,23	
===	W	166	20	10	0	6	0	[0]:2,3,0,1	[0]:0,0,1,0	[0]:12,10,19,13	(out: XM=36 XT=18 XS=0)
===	R	362	45	22	1	2	0	[1]:3,0,1,2	[1]:0,0,0,0	[1]:22,10,14,23	(out: XM=15 XT=7 XS=1)
===	W	233	29	14	1	1	1	[1]:2,0,3,1	[1]:0,0,1,0	[1]:22,10,14,23	
===	R	163	20	10	0	3	1	[0]:2,3,0,1	[0]:0,0,1,0	[0]:12,10,19,13	
===	W	169	21	10	1	1	1	[1]:1,3,2,0	[1]:0,1,1,0	[1]:22,10,14,23	

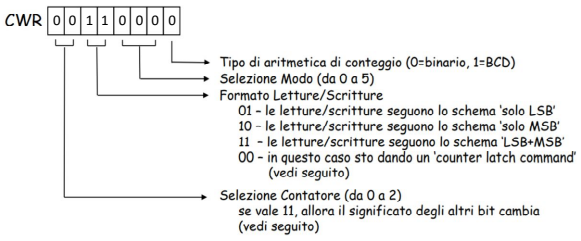
LISTA BLOCCHI USCENTI:

CONTENUTI dei SET al termine

```
---------------------------------------
P1 Nmiss=14   Nhit=6  Nref=20  mrate=0.700000   AMAT=th+mrate*tpen=32
```

ESERCIZIO 3

Il modo 0 viene utilizzato per realizzare sull'uscita OUT (es. OUT0 per CR0) una commutazione da "0" verso "1" dopo un ritardo temporale pari a N/fc essendo N la costante di tempo scritta nel registro di conteggio (es. CR0), mentre fc è la frequenza applicata sul piedino CLK corrispondente al contatore di interesse (es. CLK0 per CR0).



i) In figura sono rappresentati i segnali appena discussi (OUT, GATE, CLK); inoltre, "Counter" (CR0) indica il valore assunto dal contatore durante il conteggio, mentre CWR indica il valore impostato nel registro CWR e /WR è il segnale di scrittura applicato per poter scrivere nei registri CR0 e CWR.

ii) La parola di controllo deve valere 0011'0000=0x30, essendo necessario effettuare due scritture da 8 bit per scrivere i 16 bit della costante N=64001=0xFA01, ovvero basta scrivere 0x30 in CRW e 0xFA seguito da 0x01 in CR0.

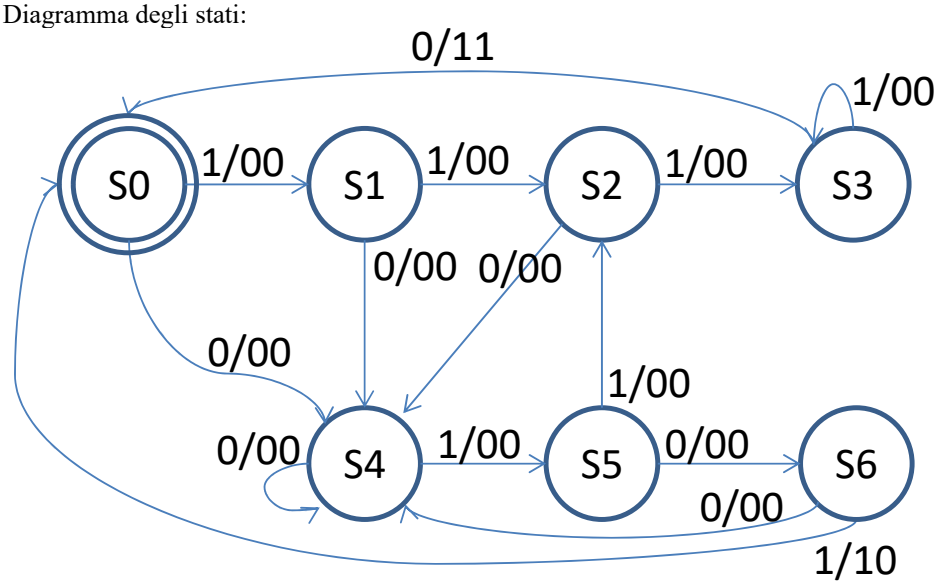
COMPITO di ARCHITETTURA DEI CALCOLATORI del 22-04-2021 **(rev. 22/6/2021)**

SOLUZIONE

ESERCIZIO 4

In corrispondenza del pattern $X_{t-3},X_{t-2},X_{t-1},X_t$ = 1,1,1,0 ottengo → Z_{t+1} = 11; (ricordare che e' richiesto Mealy ritardato).
In corrispondenza del pattern $X_{t-3},X_{t-2},X_{t-1},X_t$ = 0,1,0,1 ottengo → Z_{t+1} = 10.

Diagramma degli stati:



Codice Verilog del modulo da realizzare (possibile soluzione con Mealy-Ritardato):

```
module XXX(x,z,clock,reset_);
input        clock,reset_,x;
output [1:0] z;
reg    [1:0] OUTR;
reg    [2:0] STAR;
parameter S0='B000,S1='B001,S2='B010,S3='B011,S4='B100,S5='B101,S6='B110;
always @(reset_==0) #1 begin STAR<=S0; OUTR<=0; end
assign z=OUTR;

always @(posedge clock) if(reset_==1) #3
casex(STAR)
        S0:begin STAR<=(x==0)?S4:S1; OUTR<=0; end
        S1:begin STAR<=(x==0)?S4:S2; OUTR<=0; end
        S2:begin STAR<=(x==0)?S4:S3; OUTR<=0; end
        S3:begin STAR<=(x==0)?S0:S3; OUTR<=(x==0)?'B11:'B00; end
        S4:begin STAR<=(x==0)?S4:S5; OUTR<=0; end
        S5:begin STAR<=(x==0)?S6:S2; OUTR<=0; end
        S6:begin STAR<=(x==0)?S4:S0; OUTR<=(x==1)?'B10:'B00; end
endcase
endmodule
```

Diagramma di Temporizzazione: (template)