COMPITO di ARCHITETTURA DEI CALCOLATORI del 10-11-2021

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) [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).

```
void caesar1(char *s, char *c) {
  int k = 7;
  while (*c != 0) {
    if ('A' <= *c && *c <= 'A' + 25) {
      *s = i4modp(*c + k - 'A', 26) + 'A';
    }
    else if ('a' <= *c && *c <= 'a' + 25) {
      *s = i4modp(*c + k - 'a', 26) + 'a';
    } else {
      *s = *c;
    }
    c++;
    s++;
  }
  *s = '\0';
}
```

```
int i4modp (int i, int j) {
  int v;
  v = i % j;
  if (v < 0) v = v + j;
  return v;
}


int main() {
    char s[32];
    caesar1(s, "Architettura dei Calcolatori");
    print_string(s);
    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

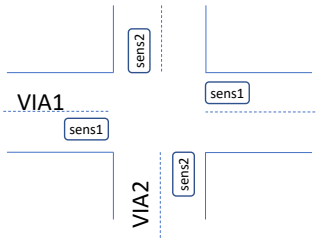
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	fa0=float
read_double	7	---	fa0=double
read_string	8	a0=address of input buffer, a1=max chars to read	---
sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
exit	10	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 10-11-2021

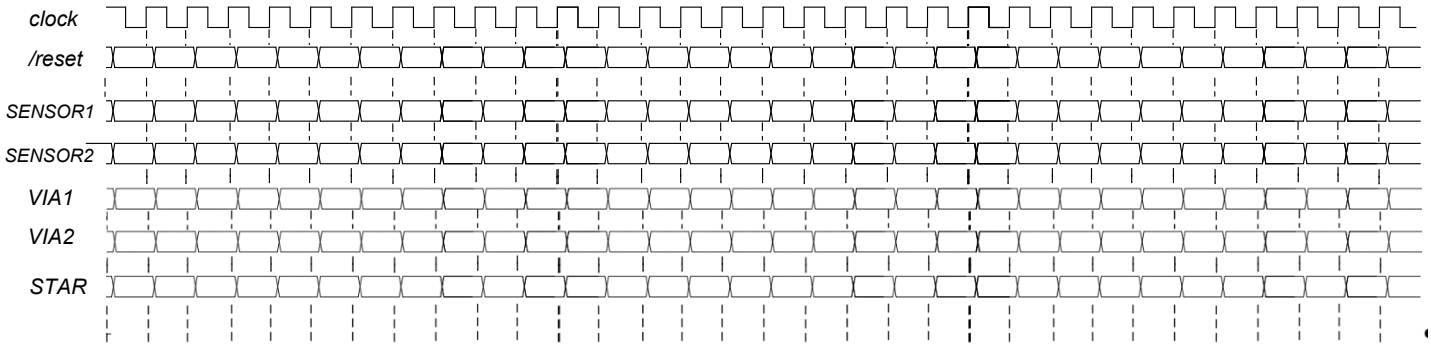
2) [5/30] Si consideri una cache di dimensione 128B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 823, 449, 447, 429, 878, 239, 433, 854, 825, 344, 422, 454, 839, 1, 26, 854, 424, 454, 829, 154, 428, 454. 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 il criterio con cui sono definiti i livelli di tensione VMIN, VOL, VOH, VMAX che designano gli intervalli di tensione corrispondenti allo ZERO logico e UNO logico nei circuiti digitali. (Suggerimento: utile una rappresentazione grafica).

4) [10/30] Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Moore che realizzi un controllore semaforico che agisce nel modo seguente. Ad un incrocio sono presenti due vie con dei sensori che rilevano la presenza dei veicoli in prossimità dell'incrocio come illustrato in figura. Ogni 5 secondi i semafori possono cambiare a seconda di quello che indicano tali sensori. Se ci sono veicoli su una via soltanto, il semaforo resta verde su tale via e rosso sull'altra. Il semaforo commuta dal verde al giallo solo se non passano più veicoli su uno dei sensori, poi commuta dal giallo al rosso e poi ritorna verde sempre in corrispondenza di un fronte in salita del clock (con periodo 5) secondi e in funzione della presenza o meno di veicoli sui sensori. Gli stimoli di ingresso sono dati dal seguente modulo Verilog Testbench. (Suggerimento: il sistema ha solo 4 stati).



Tracciare il diagramma di temporizzazione [5/10 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:




```
`timescale 1s/10ms
module Testbanch;
reg reset_; initial begin reset_=0; #6 reset_=1; #300; $stop; end
reg clock; initial clock=1; always #2.5 clock<=(!clock);
reg       SENSOR1,SENSOR2;
wire[1:0] VIA1=sem.via1;
wire[1:0] VIA2=sem.via2;
wire[1:0] STAR=sem.STAR;
initial begin
   SENSOR1<=0; SENSOR2<=0; #0.5
   SENSOR1<=1; SENSOR2<=0; #10
   SENSOR1<=1; SENSOR2<=1; #5
   SENSOR1<=0; SENSOR2<=1; #15
   SENSOR1<=1; SENSOR2<=0; #15
   SENSOR1<=0; SENSOR2<=0; #5
   $finish;
   end
   SEMAFORO sem(VIA1,VIA2,  SENSOR1,SENSOR2,clock,reset_);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 10-11-2021

SOLUZIONE

ESERCIZIO 1

```
.data
c: .asciz "Architettura dei Calcolatori"

.text
.globl main

i4modp:
   rem a0,a0,a1      # a0=v = i%j
   slt t1,a0,x0      # v<?0
   beq t1,x0,i4modp_if_end
   add a0,a0,a1      # v+=j
i4modp_if_end:
   ret

caesar1:
   addi sp,sp,-20     #alloca stack
   sw   ra,0(sp)     # save ra
   sw   s0,4(sp)     # save s0=k
   sw   s1,8(sp)     # save s1=s
   sw   s2,12(sp)    # save s2=c
   sw   s3,16(sp)    # save s3=*c

   mv   s2,a1        # s2=c
   mv   s1,a0        # s1=s
   li   s0,7         # s0=k = 7

c1_wh_ini:
   li   t1,'A'       # t1='A'
   lb   s3,0(s2)     # *c
   beq  s3,x0,c1_wh_end # *c==0 -->

   slt  t0,s3,t1    # *c<?'A'
   bne  t0,x0,c1_else1
   addi t2,t1,25    # 'A'+25
   slt  t0,t2,s3    # 'A'+25<?*c
   bne  t0,x0,c1_else1

   add  a0,s3,s0    # *c+k
   sub  a0,a0,t1    # (.)-'A'
   li   a1,26       # secondo param.
   jal  i4modp
   addi a0,a0,'A'   # risultato+'A'
   sb   a0,0(s1)    # *s=(.)
   b    c1_if_end

c1_else1:
   li   t1,'a'      # t1='a'
   slt  t0,s3,t1    # *c<?'a'
   bne  t0,x0,c1_else2
   addi t2,t1,25    # 'a'+25
   slt  t0,t2,s3    # 'a'+25<?*c
   bne  t0,x0,c1_else2

   add  a0,s3,s0    # *c+k
   sub  a0,a0,t1    # (.)-'a'
   li   a1,26       # secondo param.
   jal  i4modp
   addi a0,a0,'a'   # risultato+'a'
   sb   a0,0(s1)    # *s=(.)
   b    c1_if_end

c1_else2:
   sb s3,0(s1)      # *s = *c;

c1_if_end:
   addi s2,s2,1     # ++c
   addi s1,s1,1     # ++s
   b c1_wh_ini
c1_wh_end:
   sb  x0,0(s1)     # *s='\0'

   lw   s3,16(sp)   # ripristina reg.s
   lw   s2,12(sp)
   lw   s1,8(sp)
   lw   s0,4(sp)
   lw   ra,0(sp)    # ripristina ra
   addi sp,sp,20    # ripristina stack
   ret

main:
   addi  sp,sp,-32  # alloca s nello stack
   mv    a0,sp      # primo param.
   la    a1,c       # second param.
   jal   caesar1
   mv    a0,sp      # primo param.
   li    a7,4
   ecall            # print the string
   li    a7,10
   ecall            # exit
   addi  sp,sp,32   # ripristina stack
```

Run I/O

```
Hyjopalaabyh klp Jhsjvshavyp
-- 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=128/8/4, XM=X/B, XS=XM%S, XT=XM/S.
A=4, B=8, C=128, RP=LRU, Thit=4, Tpen=40, 22 references:

===	T	X	XM	XT	XS	XB	H	[SET]:USAGE	[SET]:MODIF	[SET]:TAG
===	R	823	102	25	2	7	0	[2]:3,0,0,0	[2]:0,0,0,0	[2]:25,-,-,-
===	W	449	56	14	0	1	0	[0]:3,0,0,0	[0]:0,0,0,0	[0]:14,-,-,-
===	R	447	55	13	3	7	0	[3]:3,0,0,0	[3]:0,0,0,0	[3]:13,-,-,-
===	W	429	53	13	1	5	0	[1]:3,0,0,0	[1]:0,0,0,0	[1]:13,-,-,-
===	R	878	109	27	1	6	0	[1]:2,3,0,0	[1]:0,0,0,0	[1]:13,27,-,-
===	W	239	29	7	1	7	0	[1]:1,2,3,0	[1]:0,0,0,0	[1]:13,27,7,-
===	R	433	54	13	2	1	0	[2]:2,3,0,0	[2]:0,0,0,0	[2]:25,13,-,-
===	W	854	106	26	2	6	0	[2]:1,2,3,0	[2]:0,0,0,0	[2]:25,13,26,-
===	R	825	103	25	3	1	0	[3]:2,3,0,0	[3]:0,0,0,0	[3]:13,25,-,-
===	W	344	43	10	3	0	0	[3]:1,2,3,0	[3]:0,0,0,0	[3]:13,25,10,-
===	R	422	52	13	0	6	0	[0]:2,3,0,0	[0]:0,0,0,0	[0]:14,13,-,-
===	W	454	56	14	0	6	1	[0]:3,2,0,0	[0]:1,0,0,0	[0]:14,13,-,-
===	R	839	104	26	0	7	0	[0]:2,1,3,0	[0]:1,0,0,0	[0]:14,13,26,-
===	W	1	0	0	0	1	0	[0]:1,0,2,3	[0]:1,0,0,0	[0]:14,13,26,0
===	R	26	3	0	3	2	0	[3]:0,1,2,3	[3]:0,0,0,0	[3]:13,25,10,0
===	W	854	106	26	2	6	1	[2]:1,2,3,0	[2]:0,0,1,0	[2]:25,13,26,-
===	R	424	53	13	1	0	1	[1]:3,1,2,0	[1]:0,0,0,0	[1]:13,27,7,-
===	W	454	56	14	0	6	1	[0]:3,0,1,2	[0]:1,0,0,0	[0]:14,13,26,0
===	R	829	103	25	3	5	1	[3]:0,3,1,2	[3]:0,0,0,0	[3]:13,25,10,0
===	W	154	19	4	3	2	0	[3]:3,2,0,1	[3]:0,0,0,0	[3]:4,25,10,0
===	R	428	53	13	1	4	1	[1]:3,1,2,0	[1]:0,0,0,0	[1]:13,27,7,-
===	W	454	56	14	0	6	1	[0]:3,0,1,2	[0]:1,0,0,0	[0]:14,13,26,0

CONTENUTI dei SET al termine

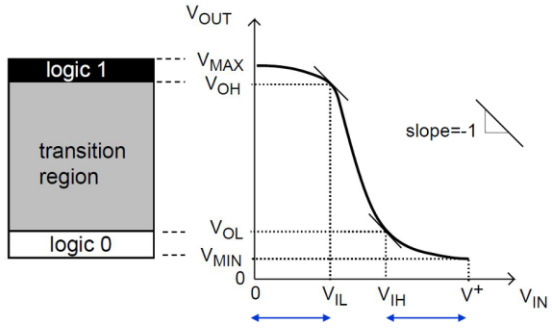
LISTA BLOCCHI USCENTI:

out: XM=55 XT=13 XS=3)

P1 Nmiss=15 Nhit=7 Nref=22 mrate=0.681818 AMAT=th+mrate*tpen=31.2727

ESERCIZIO 3

Definizione dei livelli logici 1 e 0



Intervallo di valori di ingresso che produce un valido 1 logico

Intervallo di valori di ingresso che produce un valido 0 logico

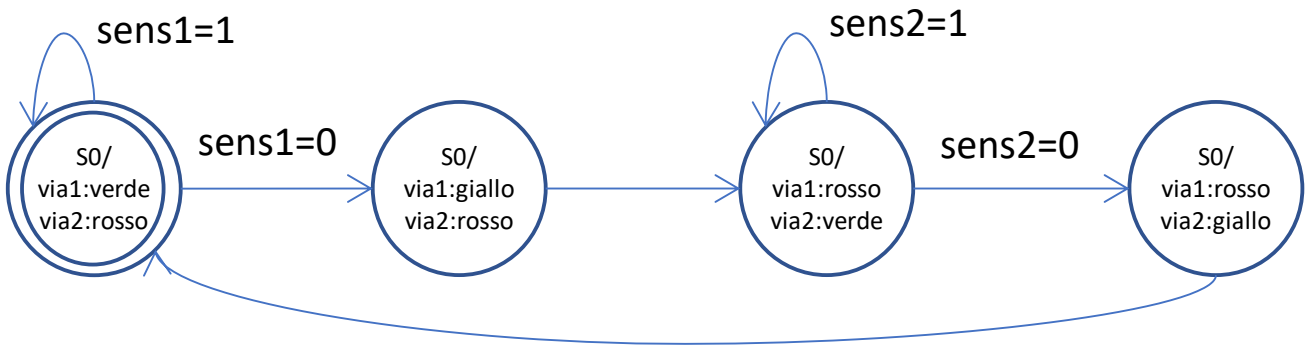
- 0 logico: intervallo di valori di uscita compreso fra VMIN e VOL
  - VMIN = tensione di uscita per la quale VIN=V+
  - VOL = la piu' piccola tensione di uscita per la quale la pendenza e' -1
- 1 logico: intervallo di valori di uscita compreso fra VOH e VMAX
  - VOH = la piu' grande tensione di uscita per la quale la pendenza e' =-1
  - VMAX = tensione di uscita per la quale VIN=0

COMPITO di ARCHITETTURA DEI CALCOLATORI del 10-11-2021

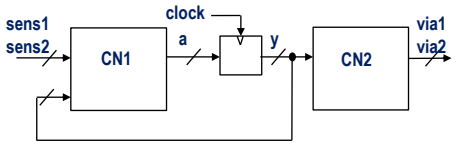
SOLUZIONE

ESERCIZIO 4

Macchina a stati finiti:



Modello di Moore:



Possibile implementazione in codice Verilog del modulo da realizzare (soluzione con Moore):

```
module SEMAFORO(via1,via2,  sens1,sens2,clock,reset_);
  input  clock, reset_;
  input  sens1, sens2;
  output [1:0]  via1, via2;
  reg    [1:0]  STAR;
  parameter S0=0,S1=1,S2=2,S3=3;
  parameter ROSSO=0, GIALLO=1, VERDE=2;
  always @(reset_==0) #1 STAR<=S0;

  assign via1=(STAR==S0)?VERDE:
              (STAR==S1)?GIALLO:
                         ROSSO;
  assign via2=(STAR==S2)?VERDE:
              (STAR==S3)?GIALLO:
                         ROSSO;

  always @(posedge clock) if (reset_==1) #0.5
    casex(STAR)
       S0: STAR<=(sens1==1)?S0:S1;
       S1: STAR<=S2;
       S2: STAR<=(sens2==1)?S2:S3;
       S3: STAR<=S0;
    endcase
endmodule
```

Diagramma di Temporizzazione: