COMPITO di ARCHITETTURA DEI CALCOLATORI del 04-07-2023

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) [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).

```
int A[3 * 4] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
int B[4 * 3] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
int C[3 * 3] = {0};

int main() {
    igemm(A, B, C, 3, 4);
    for (int i = 0; i < 3; i++) print_int(C[4*i]);
    exit(0);
}

void loadRow(int* matrix, int n, int i, int* v) {
    for (int j = 0; j < n; j++) v[j] = matrix[n * i + j];
}
```

Nota: 'int' è un intero a 64 bit.

```
void igemm(int* A, int* B, int* C, int n, int m) {
    int *vc = sbrk(8*n); int *vb = sbrk(8*m);

    for (int i = 0; i < n; i++) {
        loadRow(C, n, i, vc);
        for (int j = 0; j < m; j++) {
            int a = A[i * m + j];
            loadRow(B, n, j, vb);
            for (int k = 0; k < m; ++k) vc[k] += a * vb[k];
        }
        for (int j = 0; j < n; j++) C[n * i + j] = vc[j];
    }
}
```

RISCV Instructions (RV64IMFD) **v221117**

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 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

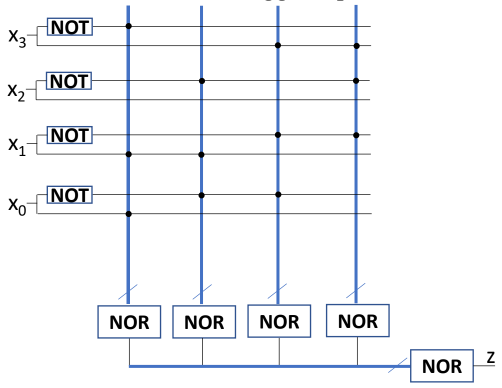
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Args
print_int	1	a0=integer to print	---
print_float	2	fa0=float to print	---
print_double	3	fa0=double to print	---
print_string	4	a0=address of ASCIIZ string to print	---
read_int	5	---	a0=integer

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 04-07-2023

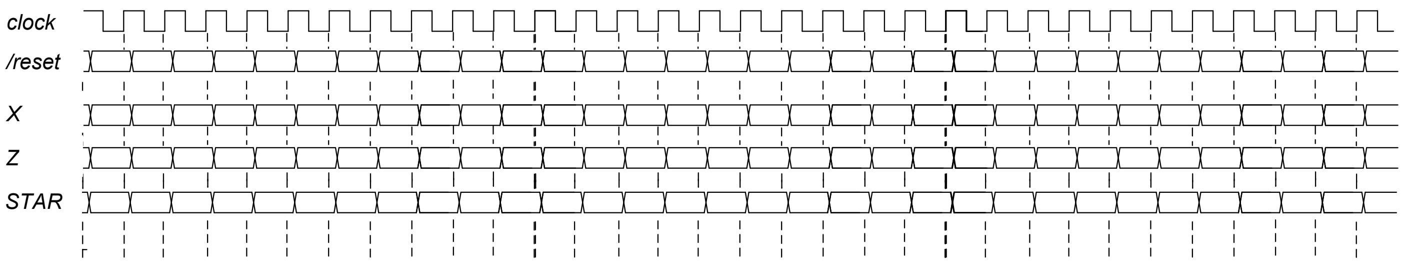
2) [5/30] Si consideri una cache di dimensione 64B e a 2 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 4 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 177, 1163, 223, 2181, 200, 3221, 175, 1184, 2182, 3201, 4176, 8173, 2176, 9183, 8251, 4176, 2201, 3180, 5171, 7178. 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] 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 $\overline{x_3}\ x_2\ \overline{x_1}\ x_0$ $\overline{x_3}\ \overline{x_2}\ x_1\ x_0$, ricavare un'equazione booleana in forma "somma di prodotti" che descriva la nuova mappa in modo da usare sottocubi di dimensione maggiore possibile:



4) [8/10] Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Mealy con un ingresso X su un bit e una uscita Z su un bit che funziona nel seguente modo: devono essere riconosciute le sequenze <u>non-interallacciate</u> 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 04-07-2023

SOLUZIONE

ESERCIZIO 1

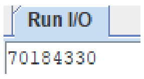
```
.data
A: .dword 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
B: .dword 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
C: .dword 0, 0, 0, 0, 0, 0, 0, 0, 0

.text
.globl main
#-------------------------------------------------
loadRow:
# a0: &matrix, a1: n=matrix size, a2: i=row index
# a3: Pointer to store row values (v)
  li   t0, 0          # Initialize counter j
lr_loop:
  beq  t0, a1, lr_end# End loop if j >= n
  mul  t1, a1, a2     # n*i
  add  t1, t1, t0     # n*i+j
  slli t1, t1, 3      # offset: 8 * (n*i+j)
  add  t1, a0, t1     # &matrix[n*i+j]
  ld   t2, 0(t1)      # Load value matrix[n*i+j]
  slli t3, t0, 3      # offset: 8 * j
  add  t3, a3, t3     # &v[j]
  sd   t2, 0(t3)      # Store value in v[j]
  addi t0, t0, 1      # Increment counter j
  j lr_loop           # Repeat loop
lr_end:
  ret
#-------------------------------------------------
igemm:
# a0: &A, a1: &B, a2: &C, a3: n=rows, a4: m=cols
  addi sp, sp,-88     # allocate frame
  sd   ra,  0(sp)
  sd   s0,  8(sp)
  sd   s1, 16(sp)
  sd   s2, 32(sp)
  sd   s3, 40(sp)
  sd   s4, 48(sp)
  sd   s5, 56(sp)
  sd   s6, 64(sp)
  sd   s7, 72(sp)
  sd   s10,80(sp)
  sd   s11,88(sp)
  mv   s0, a0         # Save pointer to A in s0
  mv   s1, a1         # Save pointer to B in s1
  mv   s2, a2         # Save pointer to C in s2
  mv   s3, a3         # Save n in s3
  mv   s4, a4         # Save m in s4
  slli a0, s3, 3      # Calculate size of vc: 8 * n
  li   a7,9          # Allocate memory for vc
  ecall
  mv   s10,a0        # Save vc
  slli a0, s4, 3      # Calculate size of vb: 8 * m
  li   a7,9          # Allocate memory for vb
  ecall
  mv   s11,a0        # Save vb
li s5, 0            # Initialize counter i
ig_for1_start:
  beq s5, s3, ig_for1_end # End loop if i >= n
  mv a0, s2         # &C
  mv a1, s3         # Size of C
  mv a2, s5         # Row index i
  mv a3, s10        # Pointer to vc
  call loadRow
  li s6, 0           # Initialize counter j
  ig_for2_start:
    beq  s6, s4, ig_for2_end# End loop if j >= m
    mul  t1, s4, s5 # m*i
    add  t1, t1, s6 # m*i+j
    slli t1, t1, 3  # offset: 8 * (m*i+j)
    add  t1, s0, t1 # &A[m*i+j]
    ld   s7, 0(t1)  # a=A[m*i+j]
    mv   a0, s1     # &B
    mv   a1, s3     # Size of B
    mv   a2, s6     # Row index j
    mv   a3, s11    # Pointer to vb
    call loadRow
    li t0, 0         # Initialize counter k (vc)
    ig_for3_start:
      beq  t0,s4,ig_for3_end# End loop if j >= m
      slli t1, t0, 3     # offset: 8 * k
      add  t5, s10, t1   # &vc[k]
      add  t6, s11, t1   # &vb[k]
      ld   t2, 0(t5)     # Load value from vc[k]
      ld   t3, 0(t6)     # Load value from vb[k]
      mul  t4, s7, t3    # a * vb[k]
      add  t2, t2, t4    # Accumulate in vc[k]
      sd   t2, 0(t5)     # Store updated vc[k]
      addi t0, t0, 1     # Increment counter k
      b ig_for3_start    # Repeat loop
    ig_for3_end:
    addi s6, s6, 1       # Increment counter j
    b ig_for2_start      # Repeat loop
  ig_for2_end:
  li t3, 0               # Initialize counter j
  ig_for4_start:
    beq  t3, s3, ig_for4_end# End loop if j >= n
    slli t4, t3, 3       # offset: 8 * j
    add  t4, t4, s10     # &vc[j]
    mul  t5, s3, s5           # n*i
    add  t5, t5, t3      # n*i+j
    slli t5, t5, 3       # 8 * (n*i+j)
    add  t5, t5, s2      # &C[n*i+j]
    ld   t6, 0(t4)       # Load value from vc[j]
    sd   t6, 0(t5)       # Store in C[n*i+j]
    addi t3, t3, 1       # Increment counter j
    j ig_for4_start      # Repeat loop
  ig_for4_end:
  addi s5, s5, 1         # Increment counter i
  b ig_for1_start        # Repeat loop
ig_for1_end:
  ld   ra,  0(sp)
  ld   s0,  8(sp)
  ld   s1, 16(sp)
  ld   s2, 24(sp)
  ld   s3, 32(sp)
  ld   s4, 40(sp)
  ld   s5, 48(sp)
  ld   s6, 56(sp)
  ld   s7, 64(sp)
  ld   s10,72(sp)
  ld   s11,80(sp)
  addi sp,sp,88   # deallocate frame
  ret

# Function: main
main:
  la   a0, A                 # Pointer to matrix A
  la   a1, B                 # Pointer to matrix B
  la   a2, C                 # Pointer to matrix C
  li   a3, 3                 # n (matrix size)
  li   a4, 4                 # m (matrix size)
  call igemm                 # Call igemm function

  li   t0, 0                 # Initialize counter i
  la   a2, C                 # &C
  li   a3, 3                 # n (matrix size)
    main_loop:
    beq  t0, a3, main_end# End loop if i >= n

    slli t1,t0,5             # 8*(4*i)=32*i
    add  t1,a2,t1            # &C[4*i]
    ld   a0, 0(t1)           # Load value from C[4*i]
    li   a7, 1               # printing integer
    ecall
    addi t0, t0, 1           # Increment counter i
    b main_loop              # Repeat loop
  main_end:
  li   a7, 10                #
  ecall                      # Call exit function
```



```
70184330
```

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=64/4/2,     XM=X/B,    XS=XM%S,    XT=XM/S.
A=2, B=4, C=64, RP=LRU, Thit=4, Tpen=40, 20 references:
=== T    X    XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  177    44   5   4   1  0 [4]:1,0 [4]:0,0  [4]:5,-
=== W 1163   290  36   2   3  0 [2]:1,0 [2]:0,0  [2]:36,-
=== R  223    55   6   7   3  0 [7]:1,0 [7]:0,0  [7]:6,-
=== W 2181   545  68   1   1  0 [1]:1,0 [1]:0,0  [1]:68,-
=== R  200    50   6   2   0  0 [2]:0,1 [2]:0,0  [2]:36,6
=== W 3221   805 100   5   1  0 [5]:1,0 [5]:0,0  [5]:100,-
=== R  175    43   5   3   3  0 [3]:1,0 [3]:0,0  [3]:5,-
=== W 1184   296  37   0   0  0 [0]:1,0 [0]:0,0  [0]:37,-
=== R 2182   545  68   1   2  1 [1]:1,0 [1]:0,0  [1]:68,-
=== W 3201   800 100   0   1  0 [0]:0,1 [0]:0,0  [0]:37,100
=== R 4176  1044 130   4   0  0 [4]:0,1 [4]:0,0  [4]:5,130
=== W 8173  2043 255   3   1  0 [3]:0,1 [3]:0,0  [3]:5,255
=== R 2176   544  68   0   0  0 [0]:1,0 [0]:0,0  [0]:68,100
=== W 9183  2295 286   7   3  0 [7]:0,1 [7]:0,0  [7]:6,286
=== R 8251  2062 257   6   3  0 [6]:1,0 [6]:0,0  [6]:257,-
=== W 4176  1044 130   4   0  1 [4]:0,1 [4]:0,1  [4]:5,130
=== R 2201   550  68   6   1  0 [6]:0,1 [6]:0,0  [6]:257,68
=== W 3180   795  99   3   0  0 [3]:1,0 [3]:0,0  [3]:99,255
=== R 5171  1292 161   4   3  0 [4]:1,0 [4]:0,1  [4]:161,130
=== W 7178  1794 224   2   2  0 [2]:1,0 [2]:0,0  [2]:224,6
-------------------------------------
P1 Nmiss=18   Nhit=2  Nref=20  mrate=0.900000   AMAT=th+mrate*tpen=40
```

CONTENUTI dei SET al termine

LISTA BLOCCHI USCENTI:

```
(out:  XM=296 XT=37 XS=0 )
(out:  XM=43 XT=5   XS=3 )
(out:  XM=44 XT=5   XS=4 )
(out:  XM=290 XT=36 XS=2 )
```

ESERCIZIO 3

La rete combinatoria si sintetizza con: $z = \overline{\overline{(\overline{x_3} + x_1 + x_0)} + \overline{(\overline{x_2} + x_1 + \overline{x_0})} + \overline{(x_3 + \overline{x_1} + \overline{x_0})} + \overline{(x_3 + \overline{x_2} + \overline{x_1})}}$ ovvero $\overline{z} = (x_3\, \overline{x_1}\, \overline{x_0}) + (x_2\, \overline{x_1}\, x_0) + (\overline{x_3}\, x_1\, x_0) + (\overline{x_3}\, x_2\, x_1) \rightarrow$disegniamo la mappa di Karnaugh osservando che dove la precedente espressione a destra vale 1 dovrò invece riportare uno 0 in quanto sto calcolando $\overline{z}$

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

ovvero, inserendo i non specificati in corrispondenza degli ingressi:

$\overline{x_3}\; x_2\; \overline{x_1}\; x_0 \quad \overline{x_3}\; \overline{x_2}\; x_1\; x_0, \quad \rightarrow$

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

E usando sottocubi di dimensione maggiore possibile: $z = x_3 x_1 + \overline{x_3}\,\overline{x_2} + \overline{x_3}\,\overline{x_1} + \overline{x_2} x_0$

COMPITO di ARCHITETTURA DEI CALCOLATORI del 04-07-2023

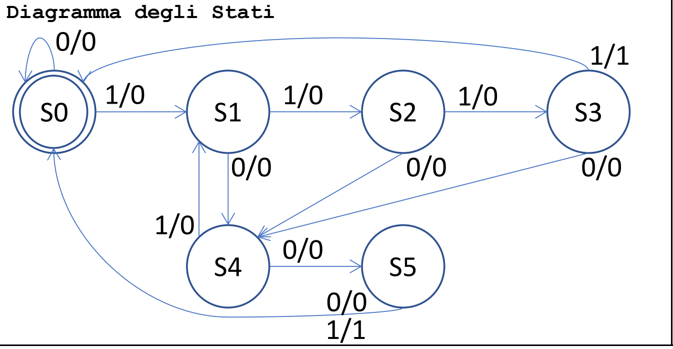
SOLUZIONE

ESERCIZIO 4

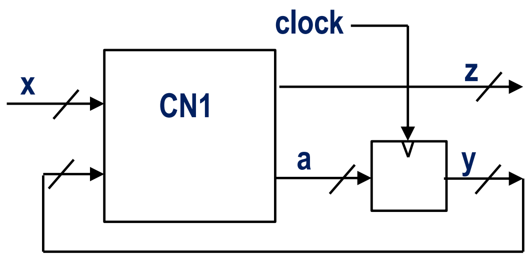
In corrispondenza del pattern X_{t-3}, X_{t-2},X_{t-1},X_t = 1,1,1,1 oppure 1,0,0,1 ottengo → Z_{t+1} = 1;
(ricordare che e' richiesto Mealy).

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

Diagramma degli Stati



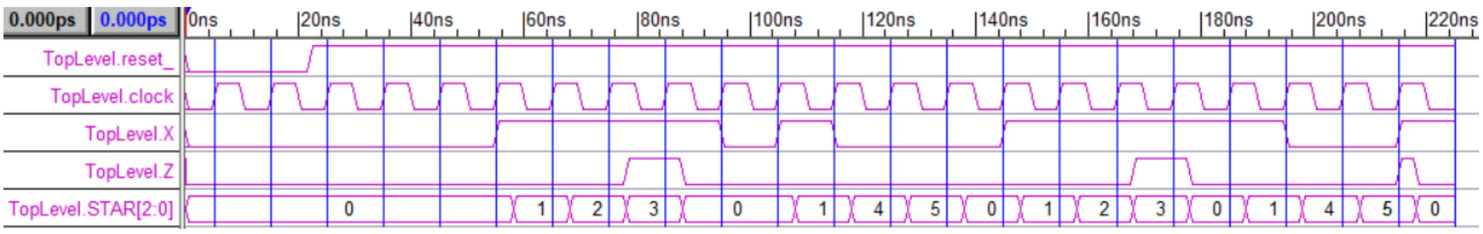
Schema di riferimento per Mealy



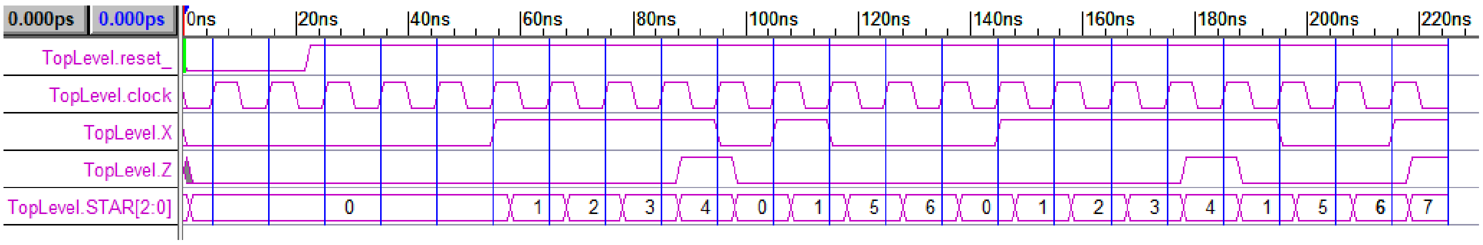
Codice Verilog del modulo da realizzare (possibile soluzione con Mealy):

```
module XXX(x,z,clock,reset_);
input clock,reset_,x;
output z;
reg [2:0] STAR;
parameter S0='B000 , S1='B001 , S2='B010 , S3='B011, S4='B100, S5='B101;
always @(reset_==0) begin STAR<=0 ; end
assign z=(STAR==S3 && x==1) + (STAR==S5 && x==1);
always @(posedge clock) if (reset_==1) #3
  casex(STAR)
    S0: begin STAR<=(x==0)?S0:S1; end
    S1: begin STAR<=(x==0)?S4:S2; end
    S2: begin STAR<=(x==0)?S4:S3; end
    S3: begin STAR<=(x==0)?S4:S0; end
    S4: begin STAR<=(x==0)?S5:S1; end
    S5: begin STAR<=(x==0)?S0:S0; end
  endcase
endmodule
```

Diagramma di Temporizzazione:



Confronto con Moore:


0.000ps
0.000ps
0ns
20ns
40ns
60ns
80ns
100ns
120ns
140ns
160ns
180ns
200ns
220ns
TopLevel.reset_
TopLevel.clock
TopLevel.X
TopLevel.Z
TopLevel.STAR[2:0]