COMPITO di ARCHITETTURA DEI CALCOLATORI del 24-11-2022

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).

```
typedef struct node { struct node *next; int vertex; }node;
int vi[12]={0,1,1,1,2,2,5, 5,4,4, 7, 7};
int vj[12]={1,2,3,4,5,6,9,10,7,8,11,12};
node *G[13]; int visited[13];
void BFS(int i) {
    node *p; int x=i; p=G[i]; int stop=0;
    do {
        if(!visited[i]) {
            visited[i]=1;
            print_int(i); print_string(" ");
        }
        if (p!= NULL) {
            i=p->vertex; p=p->next;
        }
        else stop = 1;
    } while(!stop);
    if (i < 12) BFS(++x);
}
```

Nota: 'int' è un intero a 64 bit.

```
void insert(int vi,int vj) {
    node *p,*q;
    q=(node*)sbrk(sizeof(node));
    q->vertex=vj;
    q->next=NULL;
    if(G[vi]==NULL)
        G[vi]=q;
    else {
        p=G[vi];
        while(p->next!=NULL) p=p->next;
        p->next=q;
    }
}
int main() {
    for(int i=0;i<12;i++) insert(vi[i],vj[i]);
    BFS(0);
    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	---	---

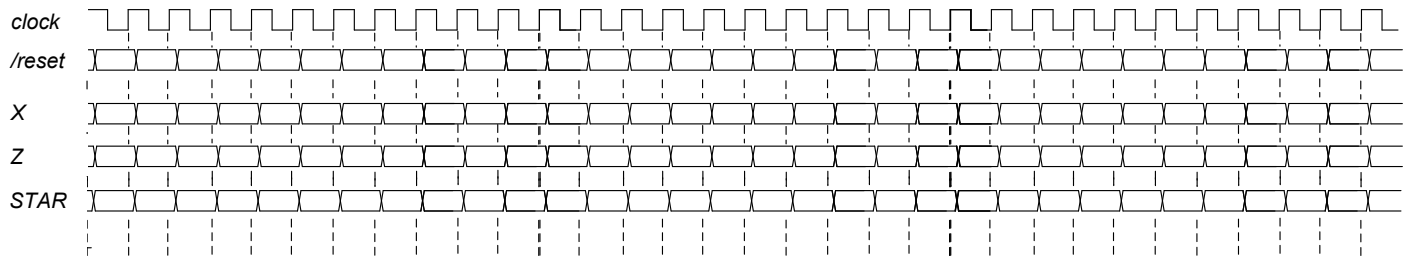
COMPITO di ARCHITETTURA DEI CALCOLATORI del 24-11-2022

2) [5/30] Si consideri una cache di dimensione 64B e a 4 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' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 1, 105, 240, 378, 492, 597, 678, 712, 850, 976, 597, 1123, 1233, 1377, 678, 1512, 1613, 1714, 1844, 1911. 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) [6/30] Illustrare i seguenti tre metodi di comunicazione fra processore e dispositivo ed evidenziarne vantaggi e svantaggi: a) spazi di I/O e memoria separati; b) spazio di I/O interamente mappato in memoria; c) spazio di indirizzamento ibrido. Esemplificare tramite possibili istruzioni macchina in ognuno dei tre casi.

4) [9/30] Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Mealy ideale (cioè senza ritardi interni) con un ingresso X su tre bit e una uscita Z su tre bit che funziona nel seguente modo: l'uscita rappresenta un numero binario naturale tale che Z=(cx2+cx1+cx0) mod 5 essendo cx2, cx1, cx0 il numero degli 1 logici che sono stati presentati fino all'istante considerato agli ingressi X[2], X[1], X[0] rispettivamente (cx2=# di "1" vista da X[2], etc.). 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  [2:0] X;
wire [2:0] Z=Xxx.z;
wire [2:0] STAR=Xxx.STAR;
initial begin X=0;
wait(reset_==1); #5
@(posedge clock); X<=2; @(posedge clock); X<=4; @(posedge clock); X<=2; @(posedge clock); X<=5;
@(posedge clock); X<=4; @(posedge clock); X<=0; @(posedge clock); X<=4; @(posedge clock); X<=0;
@(posedge clock); X<=4; @(posedge clock); X<=1; @(posedge clock); X<=5; @(posedge clock); X<=7;
@(posedge clock); X<=1; @(posedge clock); X<=2; @(posedge clock); X<=3; @(posedge clock); X<=0;
@(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 24-11-2022

SOLUZIONE

ESERCIZIO 1

```
.data
sp1: .asciz " "
vi:  .dword 0,1,1,1,2,2,5, 5,4,4, 7, 7
vj:  .dword 1,2,3,4,5,6,9,10,7,8,11,12
visited: .space 104
G:   .space 104

.text
.globl main
#-----------------------------------
BFS:# a0=i
    #____CALL FRAME_____________
    addi  sp,sp,-32  # Totale   32B
    sd    ra, 0(sp)  # save ra   8B
    sd    s0, 8(sp)  # save p    8B
    sd    s1,16(sp)  # save stop 8B
    sd    s2,24(sp)  # save x    8B
    #___________________________
    la    t0,G       # &G
    slli  t1,a0,3    # i*8
    add   t0,t0,t1   # &G[i]
    ld    s0,0(t0)   # p=G[i]
    li    s1,0       # stop=0
    mv    s2,a0      # x=i
bfs_inidowh:
    slli  t1,a0,3    # i*8
    la    t2,visited # &visited
    add   t2,t2,t1   # &visited[i]
    ld    t3,0(t2)   # visited[i]=1
    bne   t3,zero,bfs_fineif1
    li    t3,1       # 1
    sd    t3,0(t2)   # visited[i]=1
    li    a7,1       # print_int (a0)
    ecall
    la    a0, sp1    # UNO SPAZIO
    li    a7,4       # print_string
    ecall
    mv    a0,s2      # restore i
bfs_fineif1:
    beq   s0,zero,bfs_elseif2
    ld    a0,8(s0)   # a0=i=p->vertex
    ld    s0,0(s0)   # p=p->next
    b     bfs_fineif2
bfs_elseif2:
    li    s1,1       # stop=1
bfs_fineif2:
    beq   s1,zero,bfs_inidowh# stop=?0->inidow

    slti  t3,a0,12   # i<?12 NO-->fineif3
    beq   t3,zero,bfs_fineif3
    mv    a0,s2      # x
    addi  a0,a0,1    # ++x
    jal   BFS
bfs_fineif3:
    ld    ra, 0(sp)
    ld    s0, 8(sp)
    ld    s1,16(sp)
    ld    s2,24(sp)
    addi  sp,sp,32   # dealloca frame
    ret
#-----------------------------------
insert:
    # a0=vi a1=vj
    mv    t0,a0      # t0=vi
    li    a0,16      # sizeof(node)
    li    a7,9       # sbrk
    ecall            # a0=q
    sd    a1,8(a0)   # q->vertex=vj
    sd    zero,0(a0) # q->next=NULL
    la    t1,G           # &G
    slli  t0,t0,3    # vi*8
    add   t0,t1,t0   # &G[vi]
    ld    t1,0(t0)   # G[vi]
    bne   t1,zero,i_else
    sd    a0,0(t0)   # G[vi]=q
    b     i_dopoif
i_else:
          # t1 is p=G[vi]
i_while:
    ld    t2,0(t1)   # p->next
    beq   t2,zero,i_dopowhile
    mv    t1,t2      # p=p->next
    b     i_while
i_dopowhile:
    sd    a0,0(t1)   # p->next=q
i_dopoif:
    ret
#-----------------------------------
main:
    addi  sp,sp,-8
    sd    s0,0(sp)
    li    s0,0       # i=0
inifor:
    slti  t3,s0,12   # i<?12
    beq   t3,zero,finefor
    slli  t0,s0,3    # i*8
    la    t1,vi      # &vi
    add   t1,t1,t0   # &vi[i]
    ld    a0,0(t1)   # v[i]
    la    t2,vj      # &vj
    add   t2,t2,t0   # &vj[i]
    ld    a1,0(t2)   # vj[i]
    jal   insert
    addi  s0,s0,1    #++i
    b     inifor
finefor:
    li    a0,0       # 1st param.
    jal   BFS
    ld    s0,0(sp)
    addi  sp,sp,8
    li    a7,10      # exit
    ecall
```

```
Run I/O
0 1 0 1 2 3 4 5 6 7 8 9 10 11 12
-- 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=64/4/4,     XM=X/B,     XS=XM%S,    XT=XM/S.
A=4, B=4, C=64, RP=FIFO, Thit=4, Tpen=40, 20 references:
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF   [SET]:TAG
=== R    1    0   0   0   1  0 [0]:3,0,0,0 [0]:0,0,0,0   [0]:0,-,-,-
=== W  105   26   6   2   1  0 [2]:3,0,0,0 [2]:0,0,0,0   [2]:6,-,-,-
=== R  240   60  15   0   0  0 [0]:2,3,0,0 [0]:0,0,0,0   [0]:0,15,-,-
=== W  378   94  23   2   2  0 [2]:2,3,0,0 [2]:0,0,0,0   [2]:6,23,-,-
=== R  492  123  30   3   0  0 [3]:3,0,0,0 [3]:0,0,0,0   [3]:30,-,-,-
=== W  597  149  37   1   1  0 [1]:3,0,0,0 [1]:0,0,0,0   [1]:37,-,-,-
=== R  678  169  42   1   2  0 [1]:2,3,0,0 [1]:0,0,0,0   [1]:37,42,-,-
=== W  712  178  44   2   0  0 [2]:1,2,3,0 [2]:0,0,0,0   [2]:6,23,44,-
=== R  850  212  53   0   2  0 [0]:1,2,3,0 [0]:0,0,0,0   [0]:0,15,53,-
=== W  976  244  61   0   0  0 [0]:0,1,2,3 [0]:0,0,0,0   [0]:0,15,53,61
=== R  597  149  37   1   1  1 [1]:2,3,0,0 [1]:0,0,0,0   [1]:37,42,-,-
=== W 1123  280  70   0   3  0 [0]:3,0,1,2 [0]:0,0,0,0   [0]:70,15,53,61
=== R 1233  308  77   0   1  0 [0]:2,3,0,1 [0]:0,0,0,0   [0]:70,77,53,61
=== W 1377  344  86   0   1  0 [0]:1,2,3,0 [0]:0,0,0,0   [0]:70,77,86,61
=== R  678  169  42   1   2  1 [1]:2,3,0,0 [1]:0,0,0,0   [1]:37,42,-,-
=== W 1512  378  94   2   0  0 [2]:0,1,2,3 [2]:0,0,0,0   [2]:6,23,44,94
=== R 1613  403 100   3   1  0 [3]:2,3,0,0 [3]:0,0,0,0   [3]:30,100,-,-
=== W 1714  428 107   0   2  0 [0]:0,1,2,3 [0]:0,0,0,0   [0]:70,77,86,107
=== R 1844  461 115   1   0  0 [1]:1,2,3,0 [1]:0,0,0,0   [1]:37,42,115,-
=== W 1911  477 119   1   3  0 [1]:0,1,2,3 [1]:0,0,0,0   [1]:37,42,115,119
---------------------------------------
P1 Nmiss=18   Nhit=2   Nref=20   mrate=0.900000   AMAT=th+mrate*tpen=40
```

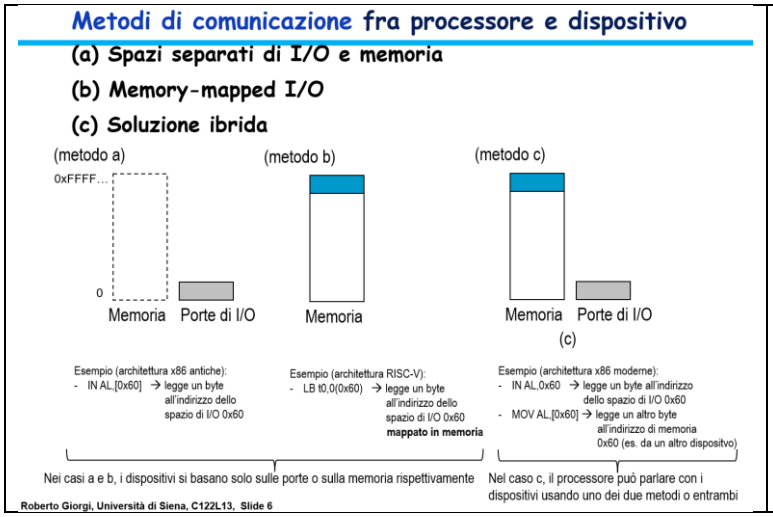
LISTA BLOCCHI USCENTI:

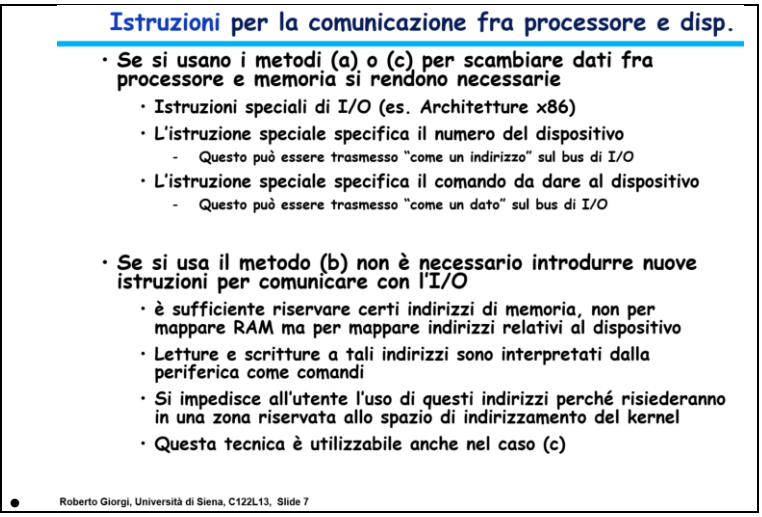
```
(out: XM=0   XT=0   XS=0 )
(out: XM=60  XT=15  XS=0 )
(out: XM=212 XT=53  XS=0 )

(out: XM=244 XT=61  XS=0 )
```

CONTENUTI dei SET al termine

ESERCIZIO 3



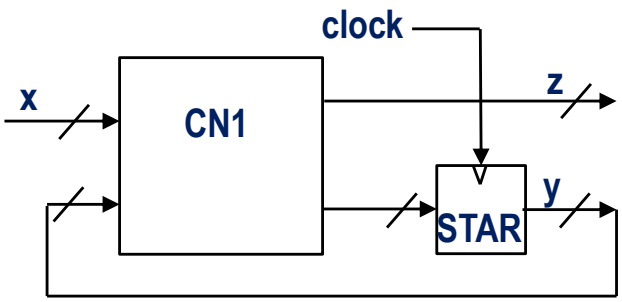


COMPITO di ARCHITETTURA DEI CALCOLATORI del 24-11-2022

SOLUZIONE

ESERCIZIO 4

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

Conviene rappresentare direttamente l'equazione richiesta per il calcolo dello stato interno. In Mealy l'uscita sarà anticipata rispetto al calcolo dello stato interno.	Schema di riferimento per Mealy Ritardato 

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

```
module XXX(x,z,clock, reset_);
input clock, reset_;
input [2:0]x;
output [2:0]z;
reg [2:0] STAR;
always@(reset_==0)  begin STAR <= 0; end
assign z = (STAR+x[0]+x[1]+x[2])%5;
always @(posedge clock) if (reset_==1)
   STAR<=(STAR+x[0]+x[1]+x[2])%5;
Endmodule
```

Diagramma di Temporizzazione: