

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

COGNOME _____

NOME _____

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).

```
float x[3][3] = {{2,7,1},{3,-2,0},{1,5,3}}, y[3][3], z[3][3];
int main() { print_float(LU(x, 3, y, z)); }
```

```
float LU(float a[][3], int n, float L[][3], float U[][3]) {
    float r = 1;
    for (int k=0; k<n; k++) {
        L[k][k] = 1;
        for (int i = k+1; i<n; i++) {
            L[i][k] = a[i][k] / a[k][k];
            a[i][k] = L[i][k];
            for (int j=k+1; j<n; j++)
                a[i][j] = a[i][j] - L[i][k] * a[k][j];
        }
        for (int j=k; j<n; j++) U[k][j] = a[k][j];
        r *= U[k][k];
    }
    return (r);
}
```

Nota: 'int' è un intero a 64 bit.

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	remainder	rem/remw x5,x6,x7	x5 ← x6 % x7	Remainder 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	srlw/srlw x5,x6,10	x5 ← x6 >> 10	Shift left by immediate value (srlw**)
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	sraiw/sraiw x5,x6,10	x5 ← x6 >> 10 (arith.)	Shift right by immediate value (sraw**)
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 ← 0x12345000	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 bits of &var; L12=low 12 bits of &var)
imm[31:12] (rd=0)	-	61/63	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

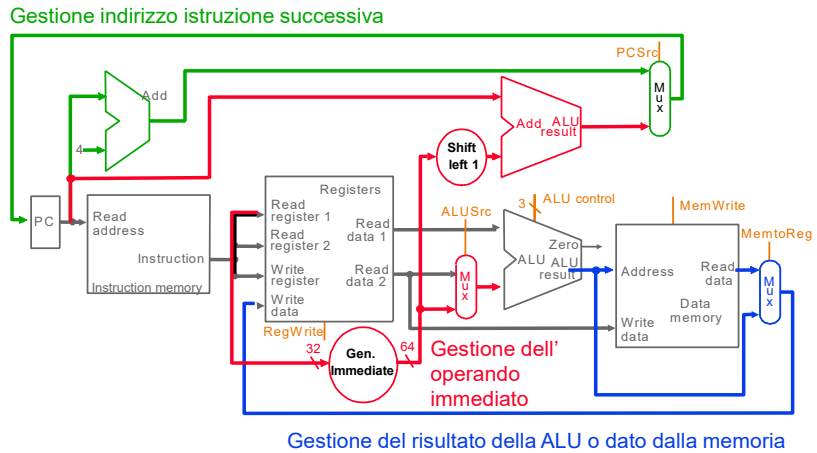
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Args
print_int	1	a0=integer to print	---
print_float	2	fa0=float to print	---
print_double	3	fa0=double to print	---
print_string	4	a0=address of ASCIIZ string to print	---
read_int	5	---	a0=integer

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	---	---

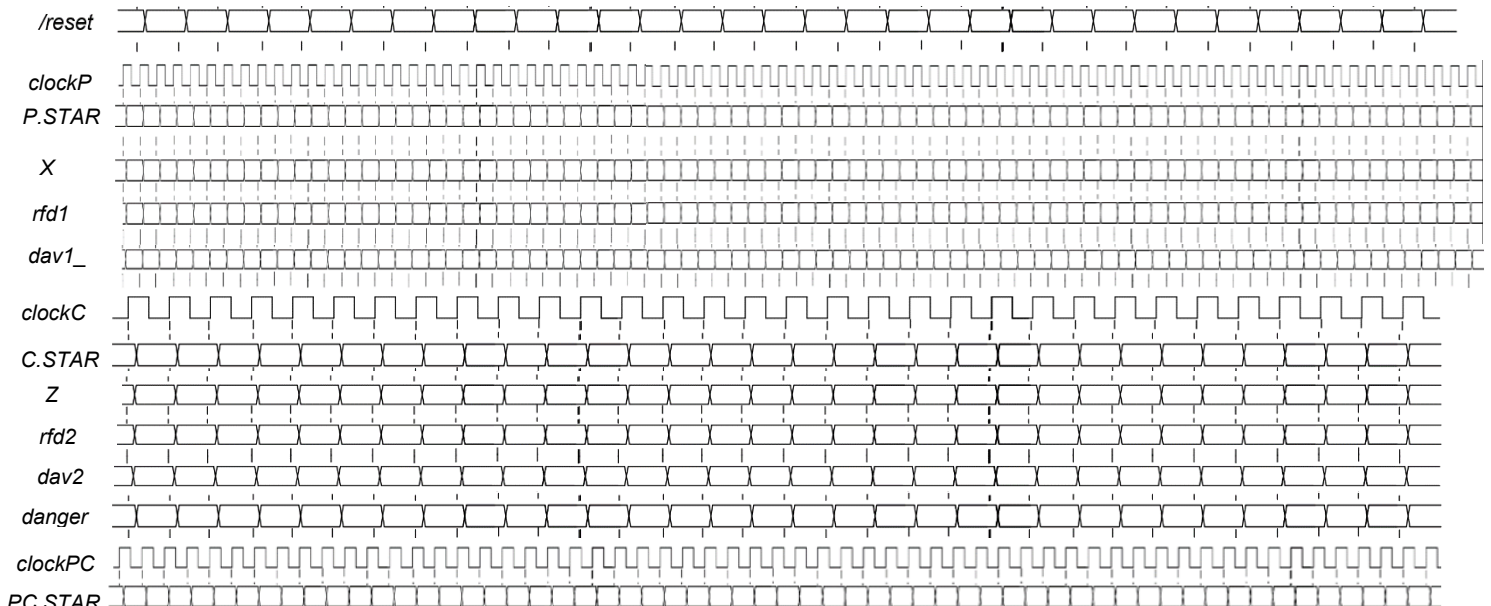
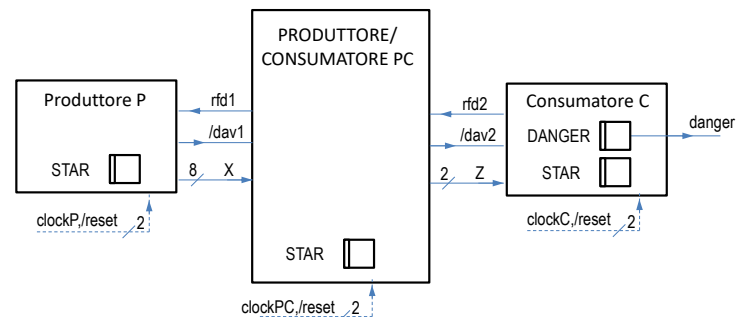
2) [5/30] Si consideri una cache di dimensione 32B e a 2 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 4 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 27, 13, 63, 11, 40, 61, 15, 124, 822, 141, 16, 113, 16, 23, 91, 216, 31, 210, 11, 18. 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] Con riferimento al processore RISC-V di figura; quali valori devono assumere i segnali di controllo per eseguire la seguente istruzione?

ld x5,0(x6)



4) [11/30] Descrivere e sintetizzare in Verilog il modulo PC di figura, che si comporta da consumatore verso il modulo P e da produttore verso il modulo C. Tutto ciò che PC deve fare è ricevere X e mandare sull'uscita Z i due bit X[4:3]. Il colloquio fra i 3 moduli P, PC, C avviene in maniera asincrona tramite i segnali rfd e /dav rispettivi; ognuno dei moduli lavora in maniera localmente sincronizzata come specificato in figura e stabilito dal qui allegato testbench. **Tracciare il diagramma di temporizzazione (punti 5/11)** come verifica della correttezza del modulo realizzato.



```

module testbench;
  reg reset; initial begin reset = 0; #5 reset = 1; #200; $stop; end
  reg clockP; initial clockP = 0; always #1 clockP <= (!clockP);
  wire[1:0] STARP=P.STAR;
  reg[7:0] X; wire rfd1, dav1;
  reg clockPC; initial clockPC = 0; always #2.5 clockPC <= (!clockPC);
  wire[2:0] Z; wire rfd2, dav2;
  reg clockC; initial clockC = 0; always #1.5 clockC <= (!clockC);
  wire[1:0] STARC=C.STAR;
  wire DANGER;
  PROD P (rfd1, clockP, reset, dav1, X);
  PRODCONS PC (dav1, rfd2, clockPC, reset, X, rfd1, dav2, Z);
  CONS C (dav2, clockC, reset, Z, rfd2, DANGER);
endmodule
    
```

```

always @(posedge clock) if (reset == 1) #0.1
  casex (STAR)
    S0: begin DAV = 1; STAR <= (rfd == 1) ? S1 : S0; end
    S1: begin DAV = 0;
        //generate a pseudo-random number via LFSR
        t = X[0] ^ X[1] ^ X[3] ^ X[4]; X = {X[6:0], t};
        STAR <= S2; end
    S2: begin STAR <= (rfd == 1) ? S2 : S0; end
  endcase
endmodule

module CONS(dav, clock, reset, Z, rfd, danger);
  input dav, clock, reset; output rfd, danger;
  input[2:0] Z;
  reg RFD, DANGER; assign rfd = RFD, danger = DANGER;
  reg[1:0] STAR; parameter S0 = 0, S1 = 1, S2 = 2;
  always @(reset == 0) begin STAR <= S0; RFD <= 0; DANGER <= 0; end
  always @(posedge clock) if (reset == 1) #0.1
    casex (STAR)
      S0: begin RFD <= 1; STAR <= (dav == 0) ? S1 : S0; end
      S1: begin RFD <= 0; if (Z == 2) DANGER <= 1; STAR <= S2; end
      S2: begin DANGER <= 0; STAR <= (dav == 0) ? S2 : S0; end
    endcase
endmodule
    
```

```

module PROD(rfd, clock, reset, dav, X);
  input rfd, clock, reset; output dav;
  output[7:0] X; reg[7:0] X; reg t;
  reg DAV; assign dav = DAV;
  reg[1:0] STAR; parameter S0 = 0, S1 = 1, S2 = 2;
  always @(reset == 0) begin DAV <= 1; STAR <= S0; X = 13; end
endmodule
    
```

SOLUZIONE

ESERCIZIO 1

```
.data
x: .float 2, 7, 1, 3, -2, 0, 1, 5, 3
y: .float 0, 0, 0, 0, 0, 0, 0, 0, 0
z: .float 0, 0, 0, 0, 0, 0, 0, 0, 0
one: .float 1.0
.text
.globl main
LU: # a0: &a, a1: n, a2: L, a3: U
la t6,one # &one
flw fa0,t6 # r = 1.0
li t3,3 # 3
li t0,0 # k = 0
LU_for1_start:
beq t0,a1,LU_for1_end# k == n --> exitfor
la t6,one # &one
flw ft1,t6 # 1.0
slli t6,t0,4 # offset:(3*k+k) * 4
add t6,t6,a2 # t0 = &L[k][k]
fsw ft1,t6 # L[k][k] = 1.0

addi t1,t0,1 # i = k + 1
LU_for2_start:
beq t1,a1,LU_for2_end# i == n --> exitfor
# L[i][k] = a[i][k] / a[k][k]
mul t6,t1,t3 # i*3
add t6,t6,t0 # i*3+k
slli t6,t6,2 # offset:(i*3+k)*4
add t4,t6,a0 # t4 = &a[i][k]
flw ft2,t4 # a[i][k]
slli t5,t0,4 # offset:(3*k+k)*4
add t5,t5,a0 # t5 = &a[k][k]
flw ft3,t5 # a[k][k]
fdiv.s ft4,ft2,ft3 # aik/akk
add t5,t6,a2 # &L[i][k]
fsw ft4,t5 # Lik=aik/akk
fsw ft4,t4 # aik=Lik

addi t2,t0,1 # j = k + 1
LU_for3_start:
beq t2,a1,LU_for3_end# j==n --> exitfor
# a[i][j] = a[i][j] - L[i][k] * a[k][j]
mul t6,t1,t3 # i*3
add t6,t6,t2 # i*3+j
slli t5,t6,2 # offset:(i*3+j)*4
add t4,t5,a0 # t4 = &a[i][j]
flw ft2,t4 # a[i][j]
mul t6,t1,t3 # i*3
add t6,t6,t0 # i*3+k
slli t5,t6,2 # offset:(i*3+k)*4
add t5,t5,a2 # t5 = &L[i][k]
flw ft3,t5 # L[i][k]
mul t5,t0,t3 # k*3
add t5,t5,t2 # k*3+j
slli t5,t5,2 # offset:(k*3+j)*4
add t5,t5,a0 # t5 = &a[k][j]
flw ft4,t5 # a[k][j]
fmul.s ft5,ft3,ft4 # Lik*akj
fsub.s ft5,ft2,ft5 # aij - (.)
fsw ft5,t4 # a[i][j] = (.)
addi t2,t2,1 # j++
b LU_for3_start
LU_for3_end:
addi t1,t1,1 # i++
b LU_for2_start
LU_for2_end:
mv t2,t0 # j=k
LU_for4_start:
beq t2,a1,LU_for4_end# j==n --> exitfor
# U[k][j] = a[k][j]
mul t5,t0,t3 # k*3
add t5,t5,t2 # k*3+j
slli t1,t5,2 # offset:(k*3+j)*4
add t5,t1,a0 # t5 = &a[k][j]
flw ft4,t5 # U[k][j] = akj
addi t2,t2,1 # j++
b LU_for4_start
LU_for4_end:
Zet
#-----
main:
la a0, x
li a1, 3
la a2, y
la a3, z
call LU # result in fa0
li a7,2 # print float in fa0
ecall
li a7, 10 # exit
ecall
```

Run I/O
-58.0

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=32/4/2, XM=X/B, XS=XM*S, XT=XM/S.

A=2, B=4, C=32, RP=FIFO, Thit=4, Tpen=40, 20 references:

====	T	X	XM	XT	XS	XB	H	[SET]:USAGE	[SET]:MODIF	[SET]:TAG
====	R	27	6	1	2	3	0	[2]:1,0	[2]:0,0	[2]:1,-
====	W	13	3	0	3	1	0	[3]:1,0	[3]:0,0	[3]:0,-
====	R	63	15	3	3	3	0	[3]:0,1	[3]:0,0	[3]:0,3
====	W	11	2	0	2	3	0	[2]:0,1	[2]:0,0	[2]:1,0
====	R	40	10	2	2	0	0	[2]:1,0	[2]:0,0	[2]:2,0
====	W	61	15	3	3	1	1	[3]:0,1	[3]:0,1	[3]:0,3
====	R	15	3	0	3	3	1	[3]:0,1	[3]:0,1	[3]:0,3
====	W	124	31	7	3	0	0	[3]:1,0	[3]:0,1	[3]:7,3
====	R	822	205	51	1	2	0	[1]:1,0	[1]:0,0	[1]:51,-
====	W	141	35	8	3	1	0	[3]:0,1	[3]:0,0	[3]:7,8
====	R	16	4	1	0	0	0	[0]:1,0	[0]:0,0	[0]:1,-
====	W	113	28	7	0	1	0	[0]:0,1	[0]:0,0	[0]:1,7
====	R	16	4	1	0	0	1	[0]:0,1	[0]:0,0	[0]:1,7
====	W	23	5	1	1	3	0	[1]:0,1	[1]:0,0	[1]:51,1
====	R	91	22	5	2	3	0	[2]:0,1	[2]:0,0	[2]:2,5
====	W	216	54	13	2	0	0	[2]:1,0	[2]:0,0	[2]:13,5
====	R	31	7	1	3	3	0	[3]:1,0	[3]:0,0	[3]:1,8
====	W	210	52	13	0	2	0	[0]:1,0	[0]:0,0	[0]:13,7
====	R	11	2	0	2	3	0	[2]:0,1	[2]:0,0	[2]:13,0
====	W	18	4	1	0	2	0	[0]:0,1	[0]:0,0	[0]:13,1

LISTA BLOCCHI USCENTI:

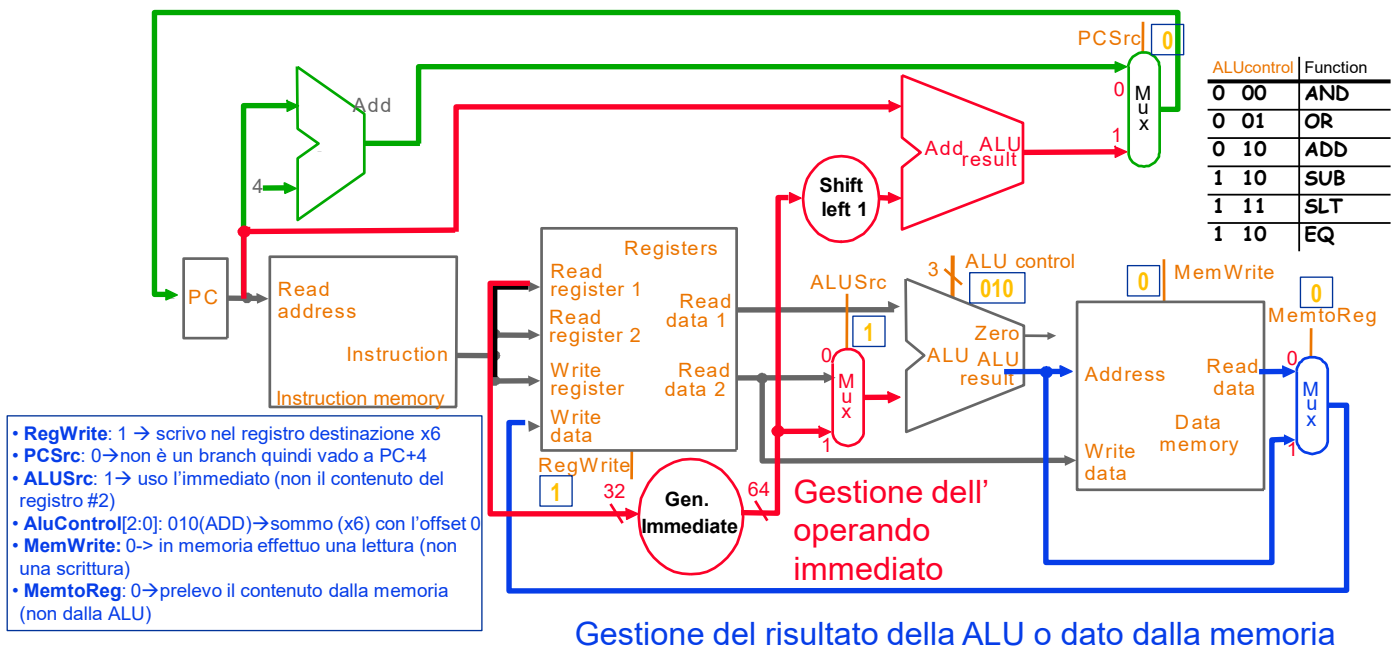
(out: XM=6 XT=1 XS=2)
(out: XM=3 XT=0 XS=3)
(out: XM=15 XT=3 XS=3)
(out: XM=2 XT=0 XS=2)
(out: XM=10 XT=2 XS=2)
(out: XM=31 XT=7 XS=3)
(out: XM=4 XT=1 XS=0)
(out: XM=22 XT=5 XS=2)
(out: XM=28 XT=7 XS=0)

CONTENUTI dei SET al termine

P1 Nmiss=17 Nhit=3 Nref=20 mrate=0.850000 AMAT=th+mrate*tpen=38

ESERCIZIO 3

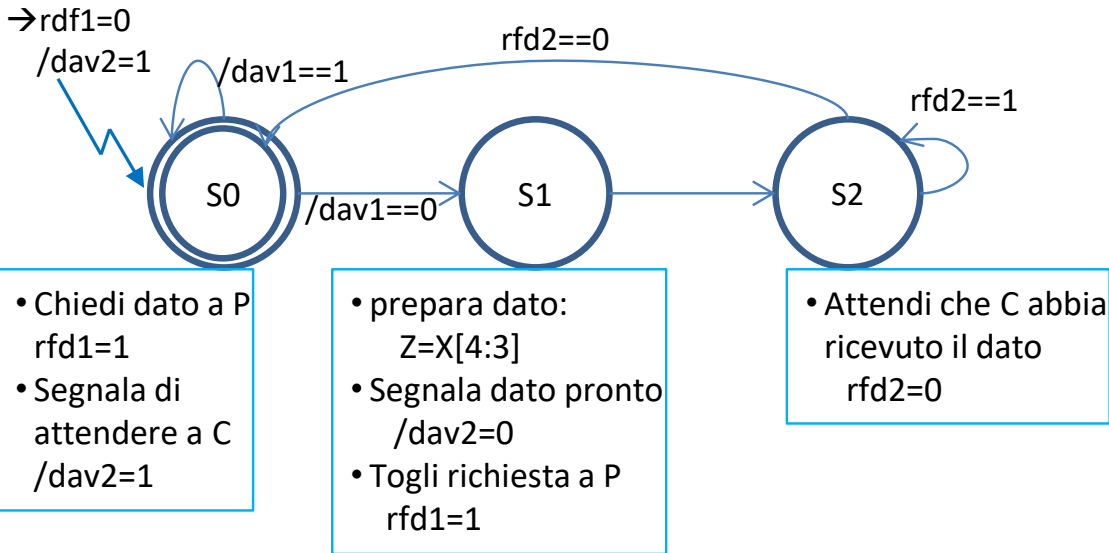
Gestione indirizzo istruzione successiva



Gestione del risultato della ALU o dato dalla memoria

SOLUZIONE

ESERCIZIO 4



Questa è una possibile soluzione del modulo “Produttore/Consumatore PC”:

```

module PRODCONS(dav1_,rfd2_,clock,reset_,X, rfd1,dav2_,Z);
input dav1_,rfd2_,clock,reset_; input[7:0] X;
output rfd1, dav2_; output[1:0] Z; reg[1:0]Z;
reg[1:0] STAR; parameter S0=0,S1=1,S2=2;
reg RFD1,DAV2_; assign rfd1=RFD1,dav2_=DAV2_;
always @(reset_==0) begin STAR<=S0; RFD1<=0; DAV2_<=0; end
always @(posedge clock) if (reset_==1) #0.1
    casex (STAR)
        S0: begin RFD1<=1; DAV2_<=1; STAR<=(dav1_==0)?S1:S0; end
        S1: begin RFD1<=0; Z<=X[4:3]; DAV2_<=0; STAR<=S2; end
        S2: begin STAR<=(rfd2==1)?S2:S0; end
    endcase
endmodule
    
```

Diagramma di Temporizzazione:

