

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) [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)	Instruction	Example	Meaning	Comments
33+0+00/3b+0+0	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+1mm/1b+0+1mm	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+0+1+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	remainder	rem/remw x5,x6,x7	x5 ← x6 % x7	Remainder 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+1mm/13+3+1mm	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+1mm/13+6+1mm/13+4+1mm	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+1mm/1b+1+1mm	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+1mm/1b+5+1mm	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+1mm/1b+5+1mm	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+1mm/03+4+1mm	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+1mm[31:12] (no funct3)	load upper immediate	lui x5,0x12345	x5 ← 0x12345000	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-reg	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+1mm/27+2+1mm	load/store floating point (32bit)	flw/fsw f0,0(x5)	f0 ← MEM[x5] / MEM[x5] ← f0	Data from FP register to memory
7+3+1mm/27+3+1mm	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

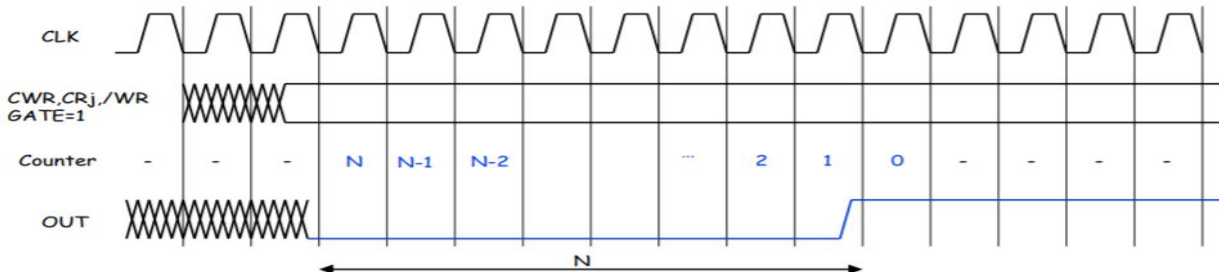
System calls

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 ASCII 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) [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.

Modo 0: conteggio abilitato dalla scrittura di CR



- 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 tracciare 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
    
```

