COMPITO di ARCHITETTURA DEI CALCOLATORI del 26-01-2021

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 come file <COGNOME>.s all'indirizzo di posta giorgi@unisi con subject: C1210126)

1) [12/30] Trovare il codice assembly RISC-V corrispondente dei seguenti micro-benchmark (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), rispettando le convenzioni di uso dei registri dell'assembly (riportate qua sotto, per riferimento).

```
char buff[80] = "Architettura dei Calcolatori\n";

char to_upper(char c) {
  if (c >= 'a' && c <= 'z') c -= 0x20;
  return (c);
}

char *myfun(int n, int *m, char *p, char c, float f,
double d) {
  char *r, *s = p;
  if (n == 0) return (p);
  while (*s != '\0' && n-- > 0) {
    *s = to_upper(*s);
    if (*s == c) r = s;
    s++;
  }
  if (f * d < 0) {
    f = -f;
    r = myfun(n/2, m, r, c, f, d);
  }
  ++*m;
  return (r);
}

int main() {
  char *p; int z = 1;
  print_string(buff);
  p = myfun(28, &z, buff, 'E', 1, -1);
  print_string(p);
  print_int(z);
  exit(0);
}
```

Nota: le costanti 'a', 'z', -0x20 sono accettate dal RARS come se fossero numeri interi

RISCV Instructions (RV64IMFD) **v191222**

Instruction coding (hexadecimal) opcode+funct3+{funct7,imm}	Instruction	Example	Meaning	Comments (** 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/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+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	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-regs**	`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+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=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

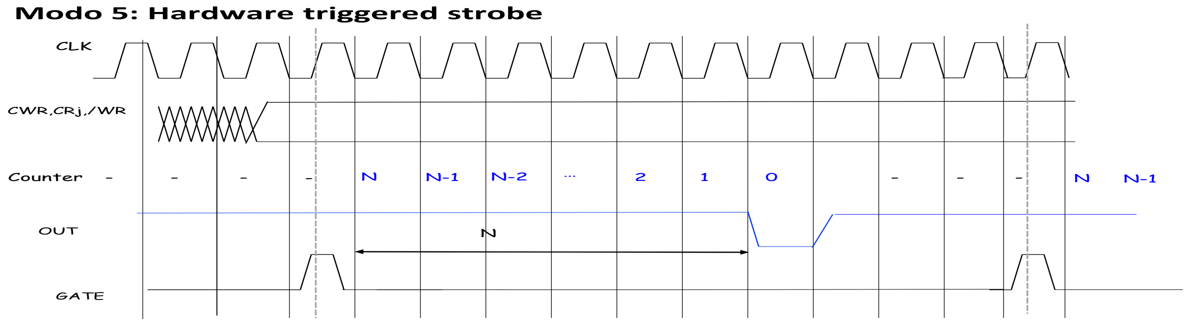
System calls

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 ASCIIZ string to print	---
read_int	5	---	a0=integer

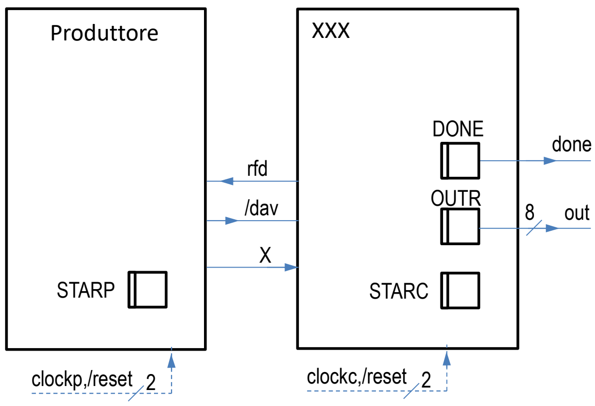
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	fa0=float
read_double	7	---	fa0=double
read_string	8	a0=address of input buffer, a1=max chars to read	---
sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
exit	10	---	---

2) [5/30] Si consideri una cache di dimensione 128B e a 4 vie di tipo write-back. La dimensione del blocco e' 32 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: 99, 104, 140, 118, 112, 197, 178, 112, 250, 176, 125, 223, 133, 277, 256, 212, 163, 174, 184. 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 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] Spiegare con proprie parole il funzionamento del "Modo 5" 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=64000, il contatore n.2 in conteggio binario.



4) [9/30] Descrivere e sintetizzare in Verilog il modulo XXX di figura che funziona nel seguente modo: riceve un bit (X) dal modulo produttore col quale colloquia tramite i segnali rfd e /dav; ogni otto bit (Xi) il modulo presenta sull'uscita out un byte (8-bit), indicandone la disponibilita' abilitando il segnale done per 1 ciclo di clock di XXX. Il modulo XXX opera con un clockc di periodo 4ns mentre il modulo Produttore, con clockp, puo' avere periodo sia 6ns (attuale codice) che 2ns: verificare il corretto funzionamento per entrambi i valori di clockp. Il codice del produttore e del testbench e' dato qua sotto. **Tracciare il diagramma di temporizzazione** come verifica della correttezza del modulo realizzato.




```
module testbench;
 reg reset_;
 initial begin reset_=0; #1 reset_=1; #400; $stop; end
 reg clockc;
 initial clockc =0; always #2 clockc <=(!clockc);
 wire[1:0] STARC=Xxx.STAR;
 wire[7:0]out; wire done, rfd, dav_;
 wire[1:0] X;
 reg clockp;
 initial clockp =0; always #3 clockp <=(!clockp);
 wire[1:0] STARP=PRO.STAR;
 wire[7:0] QP=PRO.QP, CP=PRO.C, SX=Xxx.S, CX=Xxx.C;
 XXX        Xxx(dav_,X,clockc,reset_,   rfd,out,done);
 Produttore PRO(rfd,clockp,reset_,      dav_,X);
endmodule
```

```
module Produttore(rfd,clock,reset_,   dav_,X);
 input rfd,clock,reset_; output dav_,X;
 reg DAV_; assign dav_=DAV_;
 reg XP;   assign X=XP;
 reg[7:0] QP, C;
 reg[1:0] STAR; parameter S0=0, S1=1, S2=2, Q0=173;
 always @(reset_==0) begin C<=0; STAR<=S0; end
 always @(posedge clock) if (reset_==1)  #0.1
  casex (STAR)
   S0: begin if (C==0)begin C=8; QP=Q0; XP=QP[7]; end
       DAV_=1; STAR<=(rfd==1)?S1:S0; end
   S1: begin XP=QP[7]; QP=QP<<1; C=C-1;
       DAV_=0; STAR<=S2; end
   S2: begin   STAR<=(rfd==1)?S2:S0;end
 endcase
endmodule
```