

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: per l'esercizio 4 consegnare DUE files: il file del programma VERILOG e il file del diagramma temporale (screenshot o copy/paste)

1) [12/30] Trovare il codice assembly RISC-V/MIPS 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).

```
typedef struct header {
    struct header *ptr; unsigned size;
} Header;
static Header base = {NULL, 0};
static Header *freep = NULL;

void myfree(void *ap) {
    Header *bp, *p; bp = (Header *)ap - 1;
    for (p = freep; !(bp > p && bp < p->ptr); p = p->ptr) {
        if (p >= p->ptr && (bp > p || bp < p->ptr)) break;
    }
    if (bp + bp->size == p->ptr) {
        bp->size += p->ptr->size;
        bp->ptr = p->ptr->ptr;
    } else bp->ptr = p->ptr;
    if (p + p->size == bp) {
        p->size += bp->size;
        p->ptr = bp->ptr;
    } else p->ptr = bp;
    freep = p;
}

void *alloc_and_print_pun(int sz) {
    void *p = sbrk(sz);
    print_string("p=");
    print_int(p);
    print_string("\n");
    return (p);
}

int main() {
    void *p0, *p1, *p2, *p3;
    base.ptr = &base; freep=&base;
    p0 = alloc_and_print_pun(0);
    p1 = alloc_and_print_pun(256);
    p2 = alloc_and_print_pun(256);
    p3 = alloc_and_print_pun(256);
    myfree(p1); myfree(p2); myfree(p3);
    p0 = alloc_and_print_pun(0);
}
```

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+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+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-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+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/tp, sp	frame pointer, stack pointer
x1, x3	ra, gp	return address, global pointer
x4	tp	thread pointer

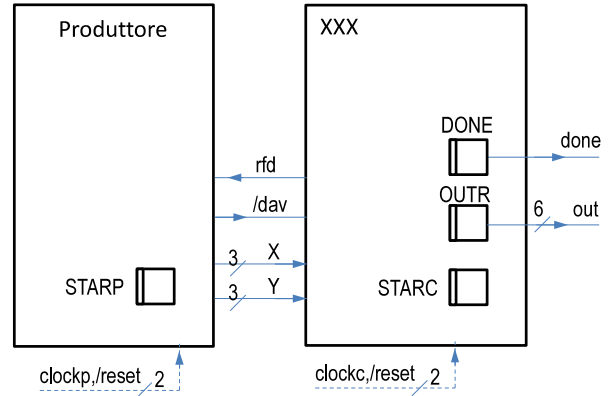
Register	ABI Name	Usage
f10-f11	fa0-fa1	Argument and Return values
f8-f9, f18-f27	fs0-fs1, fs2-fs11	Saved registers
f0-f7, f28-f31	ft0-ft7, ft8-ft11	Temporaries registers
f12-17	fa2-fa7	Function arguments

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) [4/30] Descrivere in dettaglio la pseudo-istruzione JAL del RISC-V e il formato J, evidenziando le differenze rispetto al formato U (per MIPS: descrivere la pseudo-istruzione LA); discutere le istruzioni reali usate per implementare questa pseudo-istruzione.
- 3) [4/30] Descrivere il “Read-Back command”, utilizzato nella lettura dei contatori del timer 8254: quali informazioni debbono essere scritte nel registro di controllo e quali vantaggi comporta?
- 4) [10/30] Descrivere e sintetizzare in Verilog il modulo XXX di figura che funziona nel seguente modo: riceve due interi con segno su tre bit (X e Y) dal modulo produttore col quale colloquia tramite i segnali `rfd` e `/dav`; ogni tre coppie (Xi,Yi) il modulo presenta sull’uscita `out` il prodotto scalare $\sum_{i=1}^3 X_i \cdot Y_i$, indicandone la disponibilita’ abilitando il segnale `done` per 1 ciclo di clock di XXX. Il modulo XXX opera con un `clockc` di periodo 10ns mentre il modulo Produttore, con `clockp`, puo’ avere periodo sia 4ns (attuale codice) che 12ns: 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. 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.



```

module testbench;
  reg reset_; initial begin reset_=0; #1 reset_=1; #350; $stop;
end
  reg clockc; initial clockc =0; always #5 clockc <=!clockc;
  wire[2:0] STARC=XXX.STAR;
  wire[5:0]out; wire done, rfd, dav;
  reg clockp; initial clockp =0; always #2 clockp <=!clockp;
  wire[2:0] X,Y;
  wire[2:0] STARP=PRO.STAR;
  Consumatore Xxx(dav_,X,Y, rfd,out,done, clockc,reset_);
  Produttore PRO(rfd, dav_,X,Y, clockp,reset_);
endmodule
    
```

```

module Produttore(rfd, dav_,X,Y, clock,reset_);
  input rfd, clock,reset_;
  output dav_;
  output [2:0] X,Y;
  reg DAV_; assign dav_ =DAV_;
  reg [2:0] PX,PY,QX,QY; assign X=PX, Y=PY;
  reg [1:0] STAR; parameter S0=0, S1=1, S2=2;

  always @(reset_==0) begin QX='B010; QY='B111; DAV_=1; STAR=S0; end
  always @(posedge clock) if (reset_==1) #0.1
  caseX (STAR)
    S0: begin DAV_ <= (rfd==1)?0:1; PX<=QX; PY<=QY; STAR<= (rfd==0)?S0:S1; end
    S1: begin DAV_ <= (rfd==1)?0:1; QX<=PX+1; QY<=PY+3; STAR<= (rfd==1)?S1:S2; end
    S2: begin DAV_ <=1; STAR<= (rfd==0)?S2:S0;end
  endcase
endmodule
    
```

