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

1) **[30/30]** 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).

```c
int N = 4;
float f;

float celem(int i, int j, float *X) {
    int l = i + N * j; float c = 0;
    if (i >= j) {
        c = f;
        f = f * sqrt(((float) (N + j - 1) / (N + j + 1)));
    }
    if (i == j) c *= (N + j);
    X[l] = c;
    return(fabs(c));
}

float cholg(float *X, int n) {
    int i, j;
    float sum1, max1 = 0;
```

```c
    for (i = 0; i < n; ++i) {
        sum1 = 0;
        f = (float) 1 / sqrt(n);
        for (j = 0; j < n; ++j) sum1 += celem(i, j, X);
        if (sum1 > max1) max1 = sum1;
    }
    return(max1);
}

int main() {
    float max;
    float *X = (float*)malloc(N * N * sizeof(float));
    max = cholg(X, N);
    print_float(max);
    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 bit 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 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	Register	ABI Name	Usage	Register	ABI Name	Usage
x10-x11	a0-a1	arguments and results	x0	zero	The constant value 0	f10-f11	fa0-fa1	Argument and Return values
x9, x18-x27	s1, s2-s11	Saved	x8, x2	s0/fp, sp	frame pointer, stack pointer	f8-f9, f18-f27	fs0-fs1, fs2-fs11	Saved registers
x5-7, x28-x31	t0-t2, t3-t6	Temporaries	x1, x3	ra, gp	return address, global pointer	f0 – f7, f28-f31	ft0-ft7, ft8-ft11	Temporaries registers
x12-x17	a2-a7	Arguments	x4	tp	thread pointer	f12-17	fa2-fa7	Function arguments

System calls

Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Args	Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
print_int	1	a0=integer to print	---	read_float	6	---	fa0=float
print_float	2	fa0=float to print	---	read_double	7	---	fa0=double
print_double	3	fa0=double to print	---	read_string	8	a0=address of input buffer, a1=max chars to read	---
print_string	4	a0=address of ASCIIZ string to print	---	sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
read_int	5	---	a0=integer	exit	10	---	---