COMPITINO di ARCHITETTURA DEI CALCOLATORI del 17-12-2021

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 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).

```
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;
    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 bits 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 from/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
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	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 17-12-2021

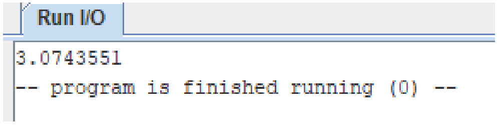
SOLUZIONE

ESERCIZIO 1

```
.data
#int N = 4;
#float f;
N: .word 4
f: .float

.text
.globl main
#float celem(int i, int j, float *X) {
celem:
#  int l = i + N * j; float c = 0;
   la       t0,N        # &N
   lw       t0,0(t0)    # N
   mul      t1,t0,a1    # N*j
   add      t1,t1,a0    # l=i+ N*j
   fmv.s.x  ft0,x0      # c=0
#  if (i >= j) {
   slt      t2,a0,a1    # i<?j
   bne      t2,x0,ce_if1_end
#    c = f;
   la       t3,f        # &f
   flw      ft1,0(t3)   # f
   fmv.s    ft0,ft1     # c=f
#    f = f * sqrt(((float) (N + j - 1) / (N + j +
1)));
   add      t2,t0,a1    # N+j
   addi     t4,t2,-1    # N+j-1
   fcvt.s.w ft2,t4      # (float)
   addi     t4,t2,1     # N+j+1
   fcvt.s.w ft3,t4      # (float)
   fdiv.s   ft2,ft2,ft3 # (N+j-1)/(N+j+1)
   fsqrt.s  ft2,ft2     # sqrt(.)
   fmul.s   ft1,ft1,ft2 # f*(.)
   fsw      ft1,0(t3)   # f=(.)

#  }
ce_if1_end:
#  if (i == j) c *= (N + j);
   bne      a0,a1,ce_if2_end # i!=?j -->
   add      t2,t0,a1    # N+j
   fcvt.s.w ft2,t2      # (float)
   fmul.s   ft0,ft0,ft2 # c*=(.)
ce_if2_end:
#  X[l] = c;
   slli     t1,t1,2     # offset=l*4
   add      a2,a2,t1    # &X[l]
   fsw      ft0,0(a2)   # X[l]=c
#  return(fabs(c));
   fabs.s   fa0,ft0     # fa0=fabs(c)
#}
ret
```

```
#int cholg(float *X, int n) {
cholg:
   addi     sp,sp,-28
   sw       ra,0(sp)    # salva ra
   sw       s0,4(sp)    # salva s0 -->i
   sw       s1,8(sp)    # salva s1 -->j
   fsw      fs0,12(sp)  # salva fs0 -->max1
   fsw      fs1,16(sp)  # salva fs1 -->sum1
   sw       s2,20(sp)   # salva s2 -->n
   sw       s3,24(sp)   # salva s3 -->X
#  int i, j;
#  float sum1, max1 = 0;
   fmv.s.x  fs0,x0      # max1=0
   mv       s2,a1       # n
   mv       s3,a0       # &X

#  for (i = 0; i < n; ++i) {
   mv       s0,x0       # i=0
for1_ini:
   slt      t0,s0,s2    # i<?n
   beq      t0,x0,for1_end
#     sum1 = 0;
   fmv.s.x  fs1,x0      # sum1=0
#     f = (float) 1 / sqrt(n);
   fcvt.s.w ft0,s2      # (float(n)
   fsqrt.s  ft0,ft0     # sqrt(n)
   li       t0,1
   fcvt.s.w ft1,t0      # (float) 1
   fdiv.s   ft2,ft1,ft0 # 1/sqrt(n)
   la       t0,f        # &f
   fsw      ft2,0(t0)   # f=(.)
#     for (j = 0; j < n; ++j) ...
   mv       s1,x0       # j=0
for2_ini:
   slt      t0,s1,s2    # j<?n
   beq      t0,x0,for2_end

#  ... sum1 += celem(i, j, X);
   mv       a0,s0       # a0=i
   mv       a1,s1       # a1=j
   mv       a2,s3       # t0=&X
   jal      celem
   fadd.s   fs1,fs1,fa0

   addi     s1,s1,1     # ++j
   b        for2_ini
for2_end:
```

```
#     if (sum1 > max1) max1 = sum1;
   flt.s    t0,fs0,fs1  # max1<?sum1
   beq      t0,x0,if_end
   fmv.s    fs0,fs1     # max1=sum1
if_end:
#  }
   addi     s0,s0,1
   b        for1_ini
for1_end:
#  return(max1);
   fmv.s    fa0,fs0     # fa0=max1
#}
   lw       s3,24(sp)   # ripristina stack
   lw       s2,20(sp)
   flw      fs1,16(sp)
   flw      fs0,12(sp)
   lw       s1,8(sp)
   lw       s0,4(sp)
   lw       ra,0(sp)
   addi     sp,sp,28
ret

#int main() {
main:
#  float max;
#  fa0=max

#  float *X = (float*)malloc(N * N *
sizeof(float));
   la   t0, N       # &N
   lw   a1,0(t0)    # N
   mul  a0,a1,a1    # N*N
   slli a0,a0,2     # N*N*sizeof(float)
   li   a7,9        # sbrk
   ecall
#  max = cholg(X, N);
   jal  cholg       # a0=X,a1=N -->fa0
#  print_float(max);
   li   a7,2        # print_float
   ecall
#  exit(0);
   li   a7,10       #exit
   ecall
#}
```



```
3.0743551
-- program is finished running (0) --
```