COMPITINO2 di ARCHITETTURA DEI CALCOLATORI del 17-12-2020

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: C1201217)

1) [30/30] 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).

```
float f(float x) {return x*x + 3*x - 2;}

float df(float x){return (f(x-0.1)-f(x+0.1))/0.2;}

void myprintf(float f, float x, float c, float d){
    print_float(f); print_string(" ");
    print_float(x); print_string(" ");
    print_float(c); print_string(" ");
    print_float(d); print_string("\n");
}

float bisect(float a, float b) {
    float xmd, epserr=0.0001, fm;
    do {
        xmd=(a+b) / 2;
        fm=f(xmd);
        if (fabs(fm) < epserr) {
            return xmd;
        } else {
            if (f(a)*fm < 0)
                b=xmd;
            else
                a=xmd;
        }
        myprintf(fm, xmd, a, b);
    }
    while (fabs(fm) >= epserr);
    return xmd;
}

int main() {
    float dy = 0, x = bisect(-2, 1);
    if (x != 0) dy=df(x);
    print_float(dy); print_string("\n");
    return 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+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	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 17-12-2020

SOLUZIONE

ESERCIZIO 1

```
.data
delta:  .float 0.1
epserr: .float 0.0001
space:  .asciz " "
acapo:  .asciz "\n"

.text
#-----------------------------------
#float f (float x) {
f: #fa0=x
#    return (x*x + 3*x - 1.75);
#}
fmul.s ft0,fa0,fa0 #x*x
addi   t0,x0,3
fcvt.s.w ft1,t0   #3.0
fmul.s ft2,ft1,fa0#3*x
addi   t0,x0,2
fcvt.s.w ft3,t0   #2.0
fadd.s fa0,ft0,ft2#x*x+3*x
fsub.s fa0,fa0,ft3#(.)-2
ret

#-----------------------------------
#float df(float x) {
df:
addi   sp,sp,-16  #alloco frame
sw     ra,0(sp)   #salvo ra
fsw    fs0,4(sp)  #salvo prec fs0
fsw    fs1,8(sp)  #salvo prec fs1
fsw    fs2,12(sp) #salvo prec fs2
fmv.s  fs0,fa0    #salvo x
la     t0,delta   #&delta
flw    fs1,0(t0)  #0.1
fsub.s fa0,fs0,fs1#x-0.1
jal    f          #f(x-0.1)
fmv.s  fs2,fa0    #salvo f(x-0.1)
fadd.s fa0,fs0,fs1#x+0.1
jal    f          #f(x+0.1)
fsub.s fa0,fs2,fa0#f(.)-f(.)
fadd.s fs1,fs1,fs1#0.2
fdiv.s fa0,fa0,fs1#(.)/0.2
#    return ((f(x-0.01)-f(x+0.01))/0.02);
lw     ra,0(sp)   #ripristino ra
flw    fs0,4(sp)  #ripristino prec fs0
flw    fs1,8(sp)  #ripristino prec fs1
flw    fs2,12(sp) #ripristino prec fs2
addi   sp,sp,16   #dealloco frame
ret
#}

#-----------------------------------
# void myprintf(float fm1, float xmd,
#               float a, float b) {
myprintf:
#fa0 gia' a posto #stampa fm1
li a7,2           #print_float
ecall
la a0,space       #spazio
li a7,4           #print_string
ecall
fmv.s fa0,fa1     #stampa xmd
li a7,2           #print_float
ecall
la a0,space       #spazio
li a7,4           #print_string
ecall
fmv.s fa0,fa2     #stampa a
li a7,2           #print_float
ecall
la a0,space       #spazio
li a7,4           #print_string
ecall
fmv.s fa0,fa3     #stampa b
li a7,2           #print_float
ecall
la a0,acapo       #a capo
li a7,4           #print_string
ecall
ret

#-----------------------------------
#float bisect(float a, float b) {
bisect:
#   fs0=xmd fs1=a fs2=b fs3=epserr fs4=fm
addi   sp,sp,-24  #alloco frame
sw     ra,0(sp)   #salvo ra
fsw    fs0,4(sp)  #salvo prec fs0
fsw    fs1,8(sp)  #salvo prec fs1
fsw    fs2,12(sp) #salvo prec fs2
fsw    fs3,16(sp) #salvo prec fs3
fsw    fs4,20(sp) #salvo prec fs4

fmv.s  fs1,fa0    #a
fmv.s  fs2,fa1    #b
#   epserr=0.0001
la     t0,epserr  #&epserr
flw    fs3,0(t0)  #epserr=0.0001

dowhile_ini:   #  do {
#xmd=(a+b) / 2
fadd.s fa0,fs1,fs2
addi   t2,x0,2
fcvt.s.w ft0,t2   #2.0
fdiv.s fs0,fa0,ft0#xmd=(a+b)/2
fmv.s  fa0,fs0    #prep. arg
#  fm=f(xmd);
jal    f          #chiamo f()
fmv.s  fs4,fa0    #fm=f()
#  if (fabs(fm) < epserr) {
fabs.s ft0,fs4    #fabs(fm)
flt.s  t3,ft0,fs3 #fabs()<epserr
beq    t3,x0,if1_else #se falso-->else
#    return 0;
fmv.s  fa0,fs0    #ritorno xmd
j      fine
#} else  {
if1_else:
#    if (f(a)*fm < 0)
fmv.s  fa0,fs1    #preparo arg
jal    f          #chiamo f()
fmul.s ft1,fa0,fs4#f(a)*fm
fcvt.s.w ft0,x0   #0.0
flt.s  t0,ft1,ft0 #f(a)*fm < 0
beq    t0,x0,if2_else#se < e' falso: else2
#       b=xmd;
fmv.s  fs2,fs0    #b=xmd
j      if2_fine
#    else
if2_else:
fmv.s  fs1,fs0    #a=xmd;
if2_fine:

fmv.s  fa0,fs4    #fm
fmv.s  fa1,fs0    #xmd
fmv.s  fa2,fs1    #a
fmv.s  fa3,fs2    #b
jal    myprintf

# while (fabs(fm) >= epserr);
fabs.s ft4,fs4    #fabs(fm)
flt.s  t0,ft4,fs3 #fabs(fm)<epserr
beq    t0,x0,dowhile_ini#se < e' falso: >= e' vero:
while_ini

# return xmd;
fmv.s  fa0,fs0    # return xmd

fine:
lw     ra,0(sp)   #ripristino ra
flw    fs0,4(sp)  #ripristino prec fs0
flw    fs1,8(sp)  #ripristino prec fs1
flw    fs2,12(sp) #ripristino prec fs2
flw    fs3,16(sp) #ripristino prec fs3
flw    fs4,20(sp) #ripristino prec fs4
addi   sp,sp,24   #dealloco frame
ret

#-----------------------------------
#int main() {
.globl main
main:
addi sp,sp,-8     #alloco frame
sw   ra,0(sp)     #salvo ra
fsw  fs0,4(sp)    #salvo prec fs0

#    float dy=0;
fmv.s.x fs0,x0    #0.0

#    x = bisect(-2, 1);
li     t0,-2
fcvt.s.w fa0,t0   #-2.0
li     t0,1
fcvt.s.w fa1,t0   #1.0
jal  bisect       #out: fa0=x

#    if (x != 0) dy=df(x);
fmv.s.x ft0,x0    # 0.0
feq.s  t0,fa0,ft0# x==?0.0
bne    t0,x0,m_if_fine#se vero: m_if_fine
jal    df         #in: fa0=x, out: fa0=dy
fmv.s  fs0,fa0    #dy=(.)
m_if_fine:

#    print_float(dy); print_string("\n");
fmv.s  fa0,fs0    #dy
li     a7,2
ecall             #print_float dy
la     a0,acapo
li     a7,4
ecall             #print_str acapo

#return 0;
li     a0,0       #0
#}
lw     ra,0(sp)   #ripristino ra
flw    fs0,4(sp)  #ripristino prec fs0
addi   sp,sp,8    #dealloco frame
ret
```

OUTPUT

Run I/O

```
-3.25 -0.5 -0.5 1.0
-1.1875 0.25 0.25 1.0
0.265625 0.625 0.25 0.625
-0.49609375 0.4375 0.4375 0.625
-0.12402344 0.53125 0.53125 0.625
0.068603516 0.578125 0.53125 0.578125
-0.028259277 0.5546875 0.5546875 0.578125
0.02003479 0.56640625 0.5546875 0.56640625
-0.004146576 0.5605469 0.5605469 0.56640625
0.007935524 0.56347656 0.5605469 0.56347656
0.0018923283 0.5620117 0.5605469 0.5620117
-0.0011277199 0.5612793 0.5612793 0.5620117
3.8218498E-4 0.5616455 0.5612793 0.5616455
-3.7276745E-4 0.5614624 0.5614624 0.5616455
-4.12311
```