COMPITINO di ARCHITETTURA DEI CALCOLATORI del 20-12-2022

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

MATRICOLA____________________

COGNOME____________________

NOME____________________

NOTA: dovrà essere consegnato l'elaborato dell'es.1 come file **<COGNOME>.s** e quelli dell'es. 4 come files **<COGNOME>.v** e **<COGNOME>.png**

1) [10/30] 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).

Nota: 'int' è un intero a 64 bit.

```
typedef struct { int idx; float val; char name[32]; } element;
element container[1000];
int hist[100];

void mystrcpy(char *s, char *t) {
    while ((*s++ = *t++) != '\0');
}

void histogram(element *A, int *bin, int length) {
   int i;
   for (i = 0; i < length; i++){
     if (A[i].val > 3.1) bin[A[i].idx]++;
     else mystrcpy(A[i].name, "Bad element");
   }
}
```

```
int main () {
   int k=0, n;
   read_int(&n);
   histogram(container, hist, n);
   while (k < n) {
     print_string(container[k].name);
     print_float(container[k].val);
     ++k;
   }
   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

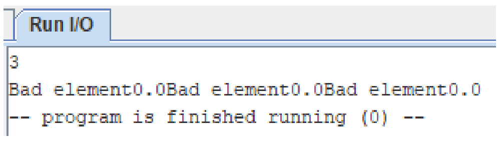
Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	fa0=float
read_double	7	---	fa0=double
read_string	8	a0=address of input buffer, a1=max chars to read	---
sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
exit	10	---	---

COMPITINO di ARCHITETTURA DEI CALCOLATORI del 20-12-2022

SOLUZIONE

ESERCIZIO 1

```
.data
container: .space 44000
hist: .space 800
c31: .float 3.1
badelem: .asciz "Bad element"

.text
.globl main

mystrcpy: # a0=s, a1=t
#___NO__CALL_FRAME___
lbu  t0,0(a1)# *s
sb   t0,0(a0)# *t
addi a1,a1,1 # t++
addi a0,a0,1 # s++
bne  t0,x0,mystrcpy # t0 !=? 0 -->cicla
ret

histogram: # a0=A, a1=bin, a2=length, s1=i
addi sp,sp,-52 # allocate frame
sw   ra,44(sp) # save ra
sw   s1,36(sp) # save all used sx
sw   s2,28(sp)
sw   s3,20(sp)
sw   s4,12(sp)
sw   s5, 4(sp)
fsw  fs0,0(sp)

mv   s3,a0     # s3=A
mv   s4,a1     # s4=bin
mv   s5,a2     # s5=lenght
la   t0,c31    # &c31
flw  fs0,0(t0) # fs0=3.1

li   s1,0      # i=0
hfor_ini:
  slt t0,s1,s5 # i<? length
  beq t0,x0,hfor_end
    # if (A[i].val > 3.1)
    li   t0,44      #sizeof(elem)
    mul  t1,s1,t0   # [i]
    add  t1,s3,t1   # &A[i]
    flw  ft0,8(t1)  # A[i].val
    flt.s t0,fs0,ft0# 3.1 <? (.)
    beq  t0,x0, hif_else
      # bin[A[i].idx]++;
      ld   t2,0(t1) # A[i].idx
      slli t2,t2,3  # *8
      add  t2,s4,t2 # &bin[.]
      ld   t3,0(t2) # bin[.]
      addi t3,t3,1  # ++
      sd   t3,0(t2) # bin[.]++
      b hif_end
    hif_else:
      addi a0,t1,12  # &A[i].name
      la   a1,badelem# &"Bad element"
      call mystrcpy
    hif_end:

  addi s1,s1,1 # i++
  b hfor_ini
hfor_end:
flw  fs0,0(sp)
lw   s5, 4(sp)
lw   s4,12(sp)
lw   s3,20(sp)
lw   s2,28(sp)
lw   s1,36(sp)
lw   ra,44(sp)
addi sp,sp,+52
ret

main:
  addi sp,sp,-16
  sd   ra, 8(sp)
  sd   s1, 0(sp)  # n

  li  a7,5        # read_int
  ecall           # a0=n

  #   histogram(container, hist, n);
  mv  s1,a0       # s1=n
  mv  a2,a0       # a2=n
  la  a0,container
  la  a1,hist
  call histogram

li  t1,0   #k
mv  t2,s1  #n
la  t3,container
mwhi_ini:
  #   while (k < n) {
  slt t0,t1,t2  # k<?n
  beq t0,x0,mwhi_end
    #  print_string(container[k].name);
    li  t0,44   # sizeof(element)
    mul t4,t1,t0# k-th eleme
    add t4,t3,t4# &container[k]
    addi a0,t4,12#&container[k].name
    li   a7,4   # print_string
    ecall
    # print_float(container[k].val);
    flw  fa0,8(t4)#container[k].val
    li   a7,2   # print_float
    ecall
  addi t1,t1,1 # ++k
  b mwhi_ini

mwhi_end:
ld   s1, 0(sp)
ld   ra, 8(sp)
addi sp,sp,+16

li a0,0
li a7,10 #exit(0)
ecall
```



```
3
Bad element0.0Bad element0.0Bad element0.0
-- program is finished running (0) --
```