COMPITINO di ARCHITETTURA DEI CALCOLATORI del 10-12-2024

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) [22/30] 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 matrix[3][3] = { 1, 2, 3, 4, 5, 6 , 7, 8, 9 };

int compute_1_norm(const int matrix[][3]) {
    int max_sum = 0;
    for (int j = 0; j < 3; ++j) {
        int column_sum = 0;
        for (int i = 0; i < 3; ++i)
            column_sum += abs(matrix[i][j]);
        if (column_sum > max_sum)
            max_sum = column_sum;
    }
    return max_sum;
}

int main() {
    int norm_1;

    norm_1 = compute_1_norm(matrix); // Compute 1-norm
    print_string("1-norm of the matrix = ");
    print_int(norm_1);
    exit(0);
}
```

Nota: 'int' è un intero a 64 bit.

RISCV Instructions (RV64IMFD) **v230703**

Instruction coding (hexadecimal)			Instruction	Example		Register operation	Meaning (** instructions available only in RV64, i.e. 64-bit case)
funct7/imm	funct3	opcode					
00	0	33/3b	**add**	`add/addw`	`x5,x6,x7`	x5 ← x6 + x7	Add two operands; exception possible (addw**)
20	0	33/3b	**subtract**	`sub/subw`	`x5,x6,x7`	x5 ← x6 – x7	Subtracts two operands; exception possible (subw**)
imm	0	13/1b	**add immediate**	`addi/addiw`	`x5,x6,100`	x5 ← x6 + 100	Add a constant ; exception possible (addiw**)
01	0	33/3b	**multiply**	`mul/mulw`	`x5,x6, x7`	x5 ← x6 * x7	(signed/word) Lower 64 bits of 128-bits product (mulw**)
01	1	33	**multiply high**	`mulh`	`x5,x6,x7`	x5 ← x6 * x7	Higher 64bits of 128-bits product
01	4	33/3b	**division**	`div/divw`	`x5,x6,x7`	x5 ← x6/x7	(signed/word) division (divw**)
01	6	33/3b	**reminder**	`rem/remw`	`x5,x6,x7`	x5 ← x6 % x7	Reminder of the division (remw**)
00	2/3	33	**set on less than**	`slt/sltu`	`x5,x6,x7`	if (x6 < x7) x5 ← 1; else x5 ← 0	signed compare x6 and x7 (less than)
imm	2/3	13	**set on less than immediate**	`slti/sltiu`	`x5,x6,100`	if (x6 < 100) x5← 1; else x5 ← 0	unsigned compare x6 and 100 (less than)
00	7/6/4	33	**and / or / xor**	`and/or/xor`	`x5,x6,x7`	x5← x6&x7 / x6|x7 / x6^ x7	Logical AND/OR/XOR register operand
imm	7/6/4	13	**and /or / xor immediate**	`andi/ori/xori`	`x5,x6,100`	x5 ← x6&100 / x6|100 / x6^100	Logical AND/OR/XOR constant operand
0	1	33/3b	**shift left logical**	`sll/sllw`	`x5,x6,x7`	x5 ← x6 << x7	Shift left by register (sllw**)
imm	1	13/1b	**shift left logical immediate**	`slli/slliw`	`x5,x6,10`	x5 ← x6 << 10	Shift left by the immediate value (slliw**)
0	5	33/3b	**shift right logical**	`srl/srlw`	`x5,x6,x7`	x5 ← x6 >> x7	Shift right by register (srlw**)
imm	5	13/1b	**shift right logical immediate**	`srli/srliw`	`x5,x6,10`	x5 ← x6 >> 10	Shift left by immediate value (srliw**)
20	5	33/3b	**shift right arithmetic**	`sra/sraw`	`x5,x6,x7`	x5 ← x6 >> x7 (arith.)	Shift right by register (sign is preserved) (sraw**)
imm	5	13/1b	**shift right arithmetic immediate**	`srai/sraiw`	`x5,x6,10`	x5 ← x6 >> 10 (arith.)	Shift right by immediate value (sraiw**)
imm	3/2/0	03	**load dword / word / byte**	`ld/lw/lb`	`x5,100(x6)`	x5 ← MEM[x6+100]	Data from memory to register
imm	6/4	03	**load word / byte unsigned**	`lwu/lbu`	`x5,100(x6)`	x5 ← MEM[x6+100]	Data from mem. To reg.; no sign extension (lwu**)
imm	3/2	23	**store dword / word / byte**	`sd/sw/sb`	`x5,100(x6)`	MEM[x6+100] ← x5	Data from register to memory (sw**)
imm[31:12]	-	37	**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 bits of &var; L12=low 12 bits of &var)
imm[31:12] (rd=0) imm[11:0] (rs1=rs2=0)	- 0	6f/63	**jump/branch**	`j/b`	`label`	PC+=off (off=PC-&label) **(PS.INST.)**	`REAL INST.: jal x0,offset/beq x0,x0,offset`
imm[31:12] (rd=1)	-	6f	**jump and link (offset)**	`jal`	`label`	x1←(PC+4); PC+=offset **(PS. INST.)**	`REAL INST.: jal x1,offset` (offset=PC-&label)
Imm (rd=0,rs=1)	0	67	**return from procedure**	`ret`		PC←x1 **(PSEUDO INST.)**	`REAL INST.: jalr x0,0(x1)`
imm	0	67	**jump and link register**	`jalr`	`x1, 100(x5)`	x1 ← (PC + 4); PC=x5+100	Procedure return; indirect call
imm÷2	0/1	63	**branch on equal / not-equal**	`beq/bne`	`x5,x6,100`	if (x5 = =/!= x6) PC=PC+100	Equal / Not-equal test; PC relative branch
00 (rs1=0,rs2=0,rd=0)	0	73	**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.
08 (rs1=0,rs2=2,rd=0)	0	73	**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`
{0,1} / {4,5}	0	53	**floating point add/sub**	`fadd/fsub.{s,d}`	`f0,f1,f2`	f0←f1+f2 / f0←f1-f2	Single or double precision add / subtract
{8,9} / {c,d}	0	53	**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`
{50,51}	0/1/2	53	**floating point compare**	`fle/flt/feq.{s,d}`	`x5,f0,f1`	x5← (f0<f1)	Single and double: compare f0 and f1 <=,<,==
{70,71} (rs2=0)	0	53	**move between x (integer) and f regs**	`fmv.x.{s,d}`	`x5,f0`	x5←f0 (no conversion)	Copy (no conversion)
{78,79} (rs2=0)	0	53	**move between f and x regs**	`fmv.{s,d}.x`	`f0,x5`	f0←x5 (no conversion)	Copy (no conversion)
imm	2	7	**load/store floating point (32bit)**	`flw/fsw`	`f0,0(x5)`	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
imm	3	7	**load/store floating point (64bit)**	`fld/fsd`	`f0,0(x5)`	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
21/20 (rs2=0)	7	53	**convert to/from double from/to single**	`fcvt.d.s/fcvt.s.d`	`f0,f1`	f0← (double)f1 / f0← (single)f1	Type conversion
{60,61} (rs2=0)	7	53	**convert to integer from {single,double}**	`fcvt.w.{s,d}`	`x5,f0`	x5← (int)f0	Type conversion
{68,69} (rs2=0)	7	53	**convert to {single,double} from integer**	`fcvt.{s,d}.w`	`f0,x5`	f0← ({single,double})x5	Type conversion
{2c,2d} (rs2=0)	0	53	**square root**	`fsqrt.{s,d}`	`f0,f1`	f0← square root of f1	Single or double square root
{10,11}	0/1/2	53	**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(0)	10	---	---

2) [8/30] Si consideri una cache di dimensione 32B e a 2 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 4 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 1433, 1454, 1425, 1454, 1422, 1454, 1639, 1726, 1854, 1424. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso alla cache, riportare i tag contenuti in cache al termine, i bit di modifica (se presenti) e la lista dei blocchi (ovvero il loro indirizzo) via via eliminati durante il rimpiazzamento ed inoltre in corrispondenza di quale riferimento il blocco e' eliminato.

COMPITINO di ARCHITETTURA DEI CALCOLATORI del 10-12-2024

SOLUZIONE

ESERCIZIO 1

```
.data
matrix: .dword 1, 2, 3, 4, 5, 6, 7, 8, 9
msg: .asciz "1-norm of the matrix = "
.text
.globl main
#-----------------------------------------
compute_1_norm: # a0 = &matrix
# Initialize registers
li t0, 0            # t0: max_sum = 0
li t4, 3            # t4: SIZE (constant 3)

# Outer loop over columns (j)
li t3, 0            # t3: j index (column)
loop_j:
  slt a6, t3, t4
  beq a6, zero, fine_loop_j # loop_j if j<SIZE

  li t1, 0          # t1: column_sum = 0
  li t2, 0          # t2: i index (row)
  # Inner loop over rows (i)
  loop_i:
    slt a6, t2, t4
    beq a6, zero, fine_loop_i # loop_i if i<SIZ
    mul t5, t2, t4 # t5 = i*SIZE
    add t5, t5, t3 # t5 = i*SIZE+j
    slli t5, t5, 3 # t5 = (.)*8 (dword off.)
    add t5, t5, a0 # t5 = &matrix[i][j]
    ld t6, 0(t5)   # t6 = matrix[i][j]

    # Take absolute value of matrix[i][j]
    slt a6, t6, zero
    beq a6, zero, pos # if false -> positive
    neg t6, t6
  pos:
    add t1, t1, t6 # column_sum += abs(.)

    addi t2, t2, 1 # i++
         b loop_i
fine_loop_i:

    # Compare column_sum with max_sum
    slt a6, t0, t1
    bne a6, zero, update_max_sum
    b continue_outer_loop

update_max_sum:
  mv t0, t1         # max_sum = column_sum

continue_outer_loop:
  addi t3, t3, 1    # j++
  b loop_j
fine_loop_j:

  # Return to caller
  mv a0, t0     # return value
  ret

main:
  la a0, matrix # address of matrix into a0
  call compute_1_norm # result in a0

  # Print the result
  mv t0, a0     # save a0 in t0
  la a0, msg    # address of message
  li a7, 4      # service 4 to print string
  ecall
  mv a0, t0     # retore a0 (result)
  li a7, 1      # service 1 for printing int
  ecall         # make ecall

  # Exit program
  li a7, 10     # service 10 for exit(0)
  ecall
```

```
Run I/O
1-norm of the matrix = 18
-- program is finished running (0) --
```

ESERCIZIO 2

Sia X il generico riferimento, A=associativita', B=dimensione del blocco, C=capacita' della cache.
Si ricava S=C/B/A=# di set della cache=32/4/2=4, XM=X/B, XS=XM%S, XT=XM/S.
A=2, B=4, C=32, RP=FIFO, Thit=4, Tpen=40, 10 references:

```
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R 1433  358  89   2   1  0 [2]:1,0 [2]:0,0  [2]:89,-
=== W 1454  363  90   3   2  0 [3]:1,0 [3]:0,0  [3]:90,-
=== R 1425  356  89   0   1  0 [0]:1,0 [0]:0,0  [0]:89,-
=== W 1454  363  90   3   2  1 [3]:1,0 [3]:1,0  [3]:90,-
=== R 1422  355  88   3   2  0 [3]:0,1 [3]:1,0  [3]:90,88
=== W 1454  363  90   3   2  1 [3]:0,1 [3]:1,0  [3]:90,88
=== R 1639  409 102   1   3  0 [1]:1,0 [1]:0,0  [1]:102,-
=== W 1726  431 107   3   2  0 [3]:1,0 [3]:0,0  [3]:107,88
=== R 1854  463 115   3   2  0 [3]:0,1 [3]:0,0  [3]:107,115
=== W 1424  356  89   0   0  1 [0]:1,0 [0]:1,0  [0]:89,-
-------------------------------------
P1 Nmiss=7   Nhit=3  Nref=10  mrate=0.700000   AMAT=th+mrate*tpen=32ns
```

LISTA BLOCCHI USCENTI:

```
(out: XM=363 XT=90 XS=3 )
(out: XM=355 XT=88 XS=3 )
```

CONTENUTI dei SET al termine