

Several Web links appear below and at least one is long, to avoid the tedium of typing them view the assignment in Adobe reader and click.

**Problem 1:** Consider these add instructions in three common ISAs. (Use the ISA manuals linked to the references page, <http://www.ece.lsu.edu/ee4720/reference.html>.)

```
# MIPS64
addu r1, r2, r3
```

```
# SPARC V9
add g2, g3, g1
```

```
# PA RISC 2
add r1, r2, r3
```

- Show the encoding (binary form) for each instruction. Show the value of as many bits as possible.
- Identify the field or fields in the SPARC add instruction which are the closest equivalent to MIPS' `func` field. (A field is a set of bits in an instruction's binary representation.)
- Identify the field or fields in the PA-RISC add instruction which are the closest equivalent to MIPS' `func` field. *Hint: Look at similar PA-RISC instructions such as `sub` and `xor`.*
- The encodings of the SPARC and MIPS add instructions have *unused fields*: non-opcode fields that must be set to zero. Identify them.

**Problem 2:** Read the Overview section of the PA-RISC 2.0 Architecture manual, [http://h21007.www2.hp.com/dspp/files/unprotected/parisc20/PA\\_1\\_overview.pdf](http://h21007.www2.hp.com/dspp/files/unprotected/parisc20/PA_1_overview.pdf). (If that link doesn't work find the overview section from the course references page, <http://www.ece.lsu.edu/ee4720/reference.html>.)

A consequence of the unused fields in MIPS and SPARC add instructions and RISC's fixed-width instructions is that the instructions are larger than they need to be.

The PA-RISC overview explains how PA-RISC embodies important RISC characteristics, as do other RISC ISAs, but also has unique features of its own.

- It is because of one of those class of features that the PA-RISC 2.0 add instruction lacks an unused field. What is PA-RISC's catchy name for those features?
- Provide an objection (from the RISC point of view) to the added functionality of PA-RISC's add instruction. If possible, find places in the overview that provide or at least hint at counterarguments to those objections.

**Problem 3:** Many ISAs today started out as 32-bit ISAs and were extended to 64 bits. Two examples are SPARC (v8 is 32 bits, v9 is 64 bits) and MIPS (MIPS32 and MIPS64). One important goal is that code compiled for the 32-bit version should run unchanged on the 64-bit version. Another important goal is to add as little as possible in the 64-bit version. For example, it would be easy maintain compatibility by adding a new set of 64-bit integer registers and new 64-bit integer instructions, but that would inflate the cost of the implementation. Another approach would be to extend the existing 32-bit integer registers to 64 bits and change the existing instructions so they now operate on 64-bit quantities, but that would break 32-bit code (consider `sll` followed by `srl`).

(a) Does a MIPS32 `add` instruction, for example, `add $s1, $s2, $s3`, perform 64-bit arithmetic when run on a MIPS64 implementation? If not, what instruction should be used to perform 64 bit integer arithmetic?

(b) Does a SPARC v8 `add` instruction, for example, `add %g2, %g3, %g1`, perform 64-bit arithmetic when run on a v9 implementation? If not, what instruction should be used?

**Problem 4:** Continuing with techniques for extending 32-bit ISAs to 64 bits, consider the problem of floating-point registers. Both MIPS32 and SPARC v8 have 32 32-bit FP registers that can be used in pairs to perform 64-bit FP arithmetic. Both 64-bit versions effectively have 32 64-bit FP registers, but using different approaches.

(a) Describe the different approaches.