LSU EE 4720 **Homework 4** Solution **Due: 4 September 2010**

Questions in this assignment are about VAX, an ISA that was mentioned in class but for which no details were given. Use the VAX-11 Architecture Reference Manual (Cover, 1982; text, 1980), which is linked to the course references page, as a reference for this assignment. (The VAX MACRO and Instruction Set Reference Manual can be used as a secondary reference; you may also use any other resources that you can find.) Chapter and section numbers in this assignment refer to the VAX-11 manual, not to the VAX MACRO manual.

Problem 1: Compare the design goals for VAX as described in Section 1.1 to the design goals for SPARC as described in the SPARC Architecture Manual V8 Section 1.1 (also linked to the course references page).

(*a*) List the design goals for each architecture that are considered defining elements of the respective ISA family (CISC and RISC). Explain whether the design goals in VAX and SPARC are mutually exclusive (meaning you can't easily do both).

The solution below is based on the EE 4720 ISA Families Overview notes. One VAX goals that is consistent with defining elements of CISC ISAs is "High bit efficiency," which implies variable instruction size. Another "Systematic, elegant instruction set . . . " can be interpreted to mean a large variety of immediate sizes and addressing modes, which is a CISC characteristic. (Of course, with no context "systematic, elegant" can mean anything.) If instead one interprets the "systematic, elegant" goal only to mean that there are few special-purpose registers and data types, then the goal is consistent with both CISC and RISC.

The SPARC goal of being "Easily pipelined" matches a goal and characteristic of RISC ISAs.

Mutual exclusivity will be discussed for family-consistent goals (the ones mentioned above): High bit efficiency, which implies variable instruction size, is not consistent with easy pipelining because instruction fetch of one instruction would depend upon the decoding of the prior instruction, making it cumbersome to do both (fetch and decode) at the same time.

(*b*) List a feature or design goal for each ISA that is unrelated to the features of the respective ISA family. Briefly explain why it is unrelated.

For VAX: *Extensibility*, because that only indicates that new instructions can be added, it says nothing about what those instructions might be.

For SPARC: *Register Windows*, because that would be just as useful on a CISC ISA.

Problem 2: Answer the following questions about VAX and RISC instruction formats.

(*a*) MIPS has three instruction formats for the integer instructions, SPARC has from three to five (depending on how you count). The VAX ISA seems to have a simpler format, according to Section 2.6 (it takes just half a page to describe). Even if the VAX format is conceptually simpler (and many would dispute that), why is it more complex in a way that is important to implementers. *Hint: This is an easy question.*

A VAX instruction can have zero to six operands, and each operand can be a variety of sizes. In a RISC ISA given a format, one knows exactly which bits a particular piece of information (say, a register number) will occupy. That's why the address inputs of the register file in our MIPS implementation can connect directly to the instruction register, no decode logic is needed. In VAX to find, say, a register number for the third operand one must look at the opcode, and the first two operands just to determine where to look. For this one needs decode logic and a shifter or multiplexor to extract the bits that are needed.

(*b*) In class each operand of a typical CISC instruction had a *type* and *info* field to describe its addressing mode. What are the corresponding VAX field names?

The *type* field is part of the *operand specifier* in VAX, and *info* field is part of the *operand specifier* field and the *specifier extension* field.

Grading Note: Many answered access type *and* data type *for this question. Those refer to the two general pieces of information that an operand specifier conveys, they do not refer to specific fields in the instruction.*

(*c*) Some RISC instructions have something like a type field, though not capable of specifying the wide range of operand types as the VAX type fields (see the previous problem). Find two examples of MIPS instructions that have an equivalent of a type field. Identify the field and explain what operand types it specifies. *Hint: Consider instructions that deal with floating-point numbers.*

The format for floating-point operate instructions, such as `add.d`, has a `fmt` field which indicates whether the operand is single- or double-precision.

(*d*) Both MIPS and SPARC have an opcode field that appears in every instruction format and some kind of an opcode extension field that appears in some of the formats. Name the opcode extension fields in MIPS and SPARC. What is the closest equivalent to an opcode extension field in VAX?

The opcode extension in MIPS is called `func`, the opcode extension in SPARC is called `op2` in format 2 and `op3` in format 3. The closest VAX equivalent is the second byte of a two-byte opcode.

Problem 3: Find the VAX addressing modes requested in the problems below. The term *addressing mode* can refer to registers, immediates, as well as memory addresses.

(*a*) Find the VAX addressing modes corresponding to the addressing mode used by the indicated operands in each instruction below.

Name the mode, and show how the operand would be encoded in the instruction (there is no need to show the entire instruction).

```
# SOLUTION
addi r1, r2, 3      # Both source operands.
#
# r2:                       Register mode.
#     7 6 5 4 3 2 1 0   <- Bit positions
#    !   5   !   2   !  <- Field values.
#
# 3:                        Literal Mode
#     7 6 5 4 3 2 1 0   <- Bit positions
#    ! 0 !   3       !  <- Field values.

lw r1, 0(r2)        # Source operand. Note that the displacement is zero.
#
# 0(r2):                   Register Deferred
#     7 6 5 4 3 2 1 0   <- Bit positions
#    !   6   !   2   !  <- Field values.

lw r3, 4(r4)        # Source operand
#
# 4(r2):                   Displacement Mode, with a byte displacement
#     First Byte         Second Byte
#     7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0   <- Bit positions
#    !  10   !   2   !  !       4       !  <- Field values.

ld [l1+l2], l3      # SPARC insn, source operand
# No equivalent VAX addressing mode. (Sorry)
```

(*b*) Find the VAX addressing mode that can be used in place of the three instructions below. Name the mode, and show how it is encoded.

```
sll r1, r2, 2
add r3, r1, r4
lw r5, 0(r3)


# Solution

# Indexed addressing: VAX assembler: (r4)[r2]
#
#     7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0   <- Bit positions
#    !  6    !   4   !  !   4   !   2   !  <- Field values.
#
# Note: The amount by which r2 is multiplied is determined by the
# instruction that uses the operand. For example for ADDB3 (add byte
# with two source operands) the value of r2 is multiplied by 1, for
# ADDL3 the value is multiplied by 4 (a VAX longword).
```