Quiz 6a

1. (3 points) Ben Bitdiddle is designing a brand new application. He knows about certain

data types and operations that will be needed, but he expects that more may

be added later, once people start using the program. He is leaning toward

using data-directed programming rather than message passing. Why?

\_\_\_\_\_ Message-passing style is inherently inefficient.

\_\_\_\_\_ Message-passing style requires the user to change pre-existing code

to add new functionality to a type.

\_\_\_\_\_ Message-passing style does not handle a large number of types well.

\_\_\_\_\_ Message-passing style requires the use of a central data structure

to hold Ben's type information.

\_\_\_\_\_ Message-passing style makes it difficult to add new types.

2. (3 points) In the first homework assignment you wrote a plural function that

takes an English noun as argument, and returns its plural. Suppose you

now want to generalize that function to work with other languages. You

rename the original function to English-plural:

(define (english-plural noun)

(cond ((member? (last noun) '(o x s)) (word noun 'es))

((equal? (last noun) 'y)

(if (vowel? (last (bl noun)))

(word noun 's)

(word (bl noun) 'ies)))

(else (word noun 's)) ))

Someone else in your project group has written analogous functions

called French-plural and Italian-plural. (You don't have

to write these!) Your plan is to type-tag each word

and use the tag to figure out which plural function to use.

Write the function plural that takes a type-tagged word as

its argument and returns the correct plural form, using these other

procedures to handle the individual languages.

Use the constructor attach-tag and selectors type-tag and contents

3. (4 points) Computers don't directly understand languages like Scheme. Instead each

computer model has a *machine language* in which instructions are

represented as sequences of numbers. We'll simulate a machine language

instruction as a list of numbers, although that isn't the actual format.

Since sequences of numbers aren't very readable, we generally represent

instructions in an *assembly* language that's slightly easier

for a human to understand. In the simplified assembly language of

this problem, the first number is replaced by an operation name, and

the other numbers may be rearranged.

Here are a few typical instructions in both forms:

machine language --> assembly language

(2 8 9 10) --> (sub 10 8 9)

(5 29 10 4) --> (store 10 4 (29))

(6 10 8 400) --> (beq 10 8 400)

The first number in each machine language instruction is the *opcode*,

which represents the operation to be performed. The other numbers are

operands.

We are going to write a disassembler --- a translator that takes a

**machine** language instruction as its argument and returns the

corresponding **assembly** language instruction. We are going to use

data-directed programming. You are given the following instruction

formats:

(define register '(0 3 1 2))

(define memory '(0 2 3 (1)))

(define branch '(0 1 2 3))

(define immediate '(0 2 1 3))

In each of these format lists, the number 0 represents the operation name,

and other numbers represent the positions of operands in the machine

language instruction. For example, the machine instruction (2 8 9 10)

is translated by finding out that the opcode 2 has the name sub

and is of register type, so its operands should be presented in

the order third, first, second.

The program must find out that opcode 2 means sub and is of register

type. You are given the following table of instruction names and types:

(define ops (list (list 'add register)

(list 'addi immediate)

(list 'sub register)

(list 'subi immediate)

(list 'load memory)

(list 'store memory)

(list 'beq branch)

(list 'bne branch)))

In this list, the *nth* element (counting from zero) represents

opcode *n*.

You are also given a procedure that takes a machine language instruction

with the opcode replaced by the operation name as argument, and returns

the assembly language form:

(define (printer instr format)

(map (lambda (piece)

(if (list? piece)

(printer instr piece)

(list-ref instr piece)))

format))

> (printer '(store 29 10 4) '(0 2 3 (1)))

(store 10 4 (29))

Your job is to write the procedure assembly that takes one

argument, a machine language instruction, and returns the assembly

language equivalent.