50006 Compilers Exercise 3: Code Generation for functions—ASSESSED

This coursework's primary objective is to help prepare you for WACC and the exams. It is to be done INDIVIDUALLY. *Please submit via LabTS*

This exercise concerns a simple programming language with functions and arithmetic expressions. Programs are represented using a Haskell abstract syntax tree:

```
type Prog = [Function]
data Function = Defun String String Exp
data Exp = Const Int | Var String | Plus Exp Exp | Minus Exp Exp | Apply String Exp
```

For example, the Haskell expression

```
[Defun "dec" "x" (Minus (Var "x") (Const 1)),
 Defun "main" "x" (Minus (Const 2) (Apply "dec" (Minus (Const 3)(Var "x"))))]
```

represents this program:

```
dec(x) { return x - 1; }
main(x) { return 2 - dec(3 - x); }
```

Your task is to write a code generator for this language. Your code generator should produce code for a 68000. Instructions are represented in Haskell using this data type:

```
data Instr = Define String       -- "label:"
           | Jsr String          -- jump to subroutine, push PC
           | Ret                 -- return from subroutine, pop PC from stack
           | Mov Operand Operand -- "mov.l xxx yyy" (yyy:=xxx)
           | Add Operand Operand -- "add.l xxx yyy" (yyy:=yyy+xxx)
           | Sub Operand Operand -- "sub.l xxx yyy" (yyy:=yyy-xxx)
data Operand = Reg Register -- specifies data or address register
             | Push         -- "-(a7)" (as in "mov.w d0,-(a7)" to push d0)
             | Pop          -- "(a7)+" (so "Mov Pop (Reg D0)" = mov.w (a7)+,d0)
             | ImmNum Int   -- "#n"
data Register = D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | A7
```

(a) Using Haskell, write a code generator for `Function`s. Assume the existence of a code generation function for expressions, `transExp`, which you will define shortly. Functions should return their result in register `D0`.

(b) Write down a code generation function `saveRegs`, which identifies which registers are currently in use, and generates code to push them onto the stack. Write another function, `restoreRegs`, that generates code to restore them from the stack.

(c) Write down a code generation function `transExp`, which generates code for an expression (`Exp`), given a list of available registers. The expression's result is to be left in the first register in the list. Assume that functions can have just one parameter (`Var "x"`), which is stored in register `D1`.

You do need to be careful to make sure registers are saved and restored properly and are not accidentally overwritten.

Your code generator need not handle running out of registers, but should avoid using registers in a way that leads to avoidable pushing and popping.

The exercise is intended to be completed in one or at most two pages of Haskell code. Full marks can be achieved by getting all the details right - there are no additional marks for going further with a more elaborate solution. Solutions to similar exercises from previous years may be found on the internet - but we will be personally offended if you expect us to provide feedback on work done by someone else. The point of the exercise is to help you learn.