221 Compilers Exercise 3: Code Generation for functions—ASSESSED

Please submit via CATE. Your solution should be in a file called "Ex3FunctionsCodeGenerator.hs", or "Ex3FunctionsCodeGenerator.lhs". You can find some Haskell code to help you get started at

`http://www.doc.ic.ac.uk/~phjk/CompilersCourse/SampleCode/Ex3-AssessedCodeGenForFunctionCalls/`

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 | 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)
           | 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 `D1`.

(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 `D0`. Your code generator need not handle running out of registers, but should use as few as possible.

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

Paul H J Kelly, Imperial College London 2013