module Compile where

import HERA.Base
import HERA.Assembly

import Data.Int
import Data.Word

data Exp
  = Var String       -- string representing an object-language variable
  | Const Int16      -- a constant (signed) 16-bit value
  | Add Exp Exp      -- sum of two expressions
  | Mult Exp Exp     -- product of two expressions
  | Neg Exp          -- negation of an expression
  deriving (Eq, Show)

-- values for variables
type Ctxt = [(String, Word16)]

compile :: Ctxt -> Exp -> AssemblyProgram
compile ctxt exp = data_insns ++ exp_insns ++ [AHalt]
  where
    data_insns = compileCtxt ctxt
    exp_insns  = compileExp exp

-- Put each variable in memory with its own data label.
compileCtxt :: Ctxt -> AssemblyProgram
compileCtxt ctxt = concatMap mk_data ctxt
  where
    mk_data (var_name, value) = [ADlabel var_name, AInteger value]

-- Compile an expression into an assembly program that puts
-- its result in R1
compileExp :: Exp -> AssemblyProgram
compileExp (Var s) = [ ASetl R1 s
                      , ALoad R1 0 R1 ]
compileExp (Const n) = [ ASet R1 (fromIntegral n) ]
compileExp (Add e1 e2) = concat [ compileExp e1
                                  , push R1
                                  , compileExp e2
                                  , pop R2
                                  , [AAdd R1 R1 R2] ]
compileExp (Mult e1 e2) = concat [ compileExp e1
                                  , push R1
                                  , compileExp e2
                                  , pop R2
                                  , [AMul R1 R1 R2] ]
compileExp (Neg e) = concat [ compileExp e
                            , [ANeg R1 R1] ]

-- Push a register value onto the stack
push :: Register -> AssemblyProgram
push reg = [ AStore reg 0 sp 
            , AInc sp 1 ]

-- Pop a value off the stack into a register
pop :: Register -> AssemblyProgram
pop reg = [ ADec sp 1 
           , ALoad reg 0 sp ]