Assignment

This assignment is worth 15% of the final mark of the declarative programming course. The code and documentation explaining your results are to be handed to the departmental secretary by Monday 16th May 2004. Assignments handed in late will be marked down by 3 marks (out of 15) per day. No assignments will be accepted after Friday 20st May 2004.

The Department of Computer Science and AI takes a very serious view on plagiarism. Refer to the departmental website on plagiarism for more details:

http://www.cs.um.edu.mt/resources/plagiarism/

Answer either question 1 or question 1. You may answer both questions.

1. The aim of this assignment is to build a simulator of a computer running assembly instructions. The computer we will be simulating will have a very simple instruction set. All data values in the machine are simple integers. Furthermore, references to data and instruction addresses are also specified as integers:

   type Data = Integer

   type DataAddress = Integer

   type InstructionAddress = Integer

   (a) The computer has three registers named A, B and C. Define a new datatype Register which can store a register name.

   (b) The registers all store a number. The state of the registers can be stored in the following type:

   type RegisterValues = [(Register, Data)]

   Write two functions to get and set register values:

   readReg :: RegisterValues -> Register -> Data
   writeReg :: RegisterValues -> Register -> Data -> RegisterValues

   (c) The computer also has an internal data memory which can store a list of numbers. Data stored in memory can be read and written by specifying its address. Create a type synonym for a block of memory, and write the following functions to read and write to memory:

   readMem :: Memory -> DataAddress -> Data
   writeMem :: Memory -> DataAddress -> Data -> Memory

   (d) The instruction set on the machine is very limited. The following instructions can be accepted:
Complete the datatype `Instruction` which describes an instruction:

```haskell
data Instruction =
    MOVrr Register Register
  | MOVrn Register Data
  | ...
```

type Program = [Instruction]

The meaning of the instructions is quite straightforward.

- **MOVrr R1 R2**: copies the value stored in register R2 to register R1.
- **MOVrn R n**: writes value n in register R.
- **MEMREAD R a**: reads the data stored at address a into register R.
- **MEMWRITE R a**: writes the value of register R at address a in the memory.
- **JUMP a**: jumps to the instruction at address a.
- **JZ R a**: if register R has value zero, jump to the instruction at address a, otherwise do nothing and continue to the next instruction.
- **INC R**: increments the value of register R.
- **DEC R**: decrements the value of register R.
- **END**: marks the end of the program.

The state of the machine contains a number of pieces of information, all of which are needed to be able to run a program:

1. The program being run;
2. The address of the next instruction to run;
3. The values of the registers;
4. The state of the memory.

```haskell
type ProgramState = (Program, InstructionAddress, RegisterValues, Memory)
```

Write a function `displayState` which translates a program state to a human readable string:

```haskell
displayState :: ProgramState -> String
```

Write a function `step` which performs one instruction, changing the program state accordingly:

```haskell
step :: ProgramState -> ProgramState
```

Write a function `hasTerminated` which returns whether the program in the given state has terminated (the next instruction to run is an `END` instruction).

```haskell
hasTerminated :: ProgramState -> Bool
```

Write a function which runs a program from a given state until termination:
run :: ProgramState -> ProgramState

(i) Write a function which runs a given program from a state with all registers and memory
values set to zero until termination:

execute :: Program -> ProgramState

(j) Write and test an assembly program for this machine to add the contents of registers \( A \)
and \( B \), leaving the result in \( A \).

(k) Write and test an assembly program for this machine to multiply the contents of reg-
isters \( A \) and \( B \), leaving the result in \( A \).