Exercise 6

Due 7/7/2017

1 Introduction

This exercise implements the code generation part of compiling StarKist programs to MIPS assembly. The details of the StarKist programming language (including its syntax and semantics) were described previously in exercise 5. The code generation module traverses the AST built by Bison, and builds an intermediate representation (IR) tree. The IR tree is then scanned in a post order fashion to further translate it into MIPS assembly. So roughly, the code

Phase(1)	AST $\rightarrow$ IR tree
Phase(2)	IR tree $\rightarrow$ ASM file

Table 1: The LIR tree nodes.

generation can divided into two parts: AST to IR tree, and IR tree into MIPS assembly. Choosing an IR language specification inevitably affects which of the two phases becomes more tedious. By that we mean that there are very few IR commands. Specifically, Table 1 describes the entire set of IR commands.

2 IR Specification

we will use a low level IR (LIR) tree contains a very limited set of commands. Table

BINOP	MOVE	MEM
CJUMP	JUMP	CONST
LABEL	TEMP	

Table 2: The LIR tree nodes.

3 Translation from LIR tree to ASM file

As mentioned earlier, the advantage of choosing a lower level IR tree is an easier translation from IR tree to assembly, and a harder translation from AST to IR tree. To create a graph visualization of the AST, please install graphviz and run

```
        $ dot -Tjpeg -o ./AST_Graph.jpeg ./AST_Graph.txt
```

from `EX5/LINUX_GCC_MAKE`

4 Input

The input for this exercise is a single text file, the input StarKist program.

5 Output

The output is a MIPS asm text file that contains the compiled StarKist program. Currently, the supplied makefile uses (UBUNTU's native MIPS simulator) spim to run your compiled program and save results in the output directory:

```
spim -file compiledProgram.s > Output.txt
```

Please make sure that *every* StarKist program given to you does what is expected to do (printing primes between 2 and 100, merging sorted lists, checking out of bound array accesses etc.)

6 Submission Guidelines

The code for this exercise resides as usual in subdirectory EX6 of the course GitHub. Next, you need to add the relevant derivation rules and AST constructors for classes. Last, you should implement the missing parts of the Starkist semantic analyzer. The semantic analyzer resides in the file semant.c, and this is where most of your changes will occur. Please submit your exercise in your GitHub repository under COMPILATION/EX5, and have a makefile there to build a runnable program called compiler. Make sure that compiler is created in the same level as the makefile: inside EX5. To avoid the pollution of EX5, please remove all *.o files once the target is built. The next paragraph describes the execution of compiler.

Execution parameters compiler receives 2 input file names:

InputStarkistProgram.txt
OutputMipsAsm.s