Compiler Project Language

Amit Zohar, 313307720

Ofek Ben Atar 322208430

1. Introduction

The CPL (Compiler Project Language) is a compiler that translates source code written in the CPL language into executable machine code in the form of MIPS assembly language file. The CPL compiler is implemented using the lexical analyzer Flex and parser generator Bison. The program's symbol table implementation uses a hash table of size 500 to store variables. The CPL compiler generates two output files: a <filename>.lst file containing the listing of the source code with error messages and a <filename>.s file containing the compiled assembly code.

2. System Requirements

- A C compiler that supports the C18 standard
- Flex and Bison lexical analyzer and parser generator tools

3. Installation

Clone the project from git:

```
> git clone https://github.com/AmitGZ/Compilation-Project.git
> cd Compilation-Project
```

To compile the CPL compiler and run it on <filename>.cpl (the input file shall be in the bin directory same as the cpm.exe)
Either use the makefile provided as such:

```
Compilation-Project > cd src
src > make all
src > make compile file=<filename>.cpl
```

Or do it manually:

```
Compilation-Project > cd src
src > flex cpm.l
src > bison -t -d cpm.y
src > gcc -o ../bin/cpm cpm.tab.c lex.yy.c MipsWriter.c Hash.c
src > cd ../bin
bin > ./cpm “<filename>.cpl”
```

4. Symbol Table Implementation

The CPL compiler's symbol table implementation uses a hash table of size 500 to store variables. The hash table implementation is located in the Hash file.
The reason we chose a hash table for the symbol table implementation is because it provides fast and efficient lookup and insertion of symbols. When a symbol is added to the symbol table, its name is hashed to obtain an index in the hash table, and the symbol is stored at that index. When a symbol needs to be looked up, its name is hashed again to find the index, and the symbol is retrieved from that index.
Hash table implementation has an average lookup and insertion time complexity of O(1), which is very efficient for large symbol tables. It's also very scalable, as the hash table can be resized dynamically as the number of symbols grows, we were informed that the program will have up to 250 variables so we chose double the expected size of variables to ensure the performance of the hash table.
The hash table is updated each time a variable is declared, each node in the hash table contains the variable's type, name, is constant.
In summary, hash table implementation is a good choice for symbol table implementation because of its fast lookup and insertion times, scalability, and efficient use of memory.

5. Error Handling

The CPL compiler reports all the errors that exist in the source code. If the program does not compile, the SW shall not create a .s file but the SW shall continue to compile the whole program to report all the errors that exist. Function inputs' are validated using a macro MY_ASSERT which receives a boolean and asserts it. If it is false the SW shall return (exit from the function) and report to stdout the SW exception.

6. Code Description

6.1. cpm.y

Contains the main method, calls the yyparse() method, opens and the creates output files.

6.2. MipsWriter

The MipsWriter file is used by cpm.y bison parser, it is responsible for writing and translating all the parsed language to assembly language. This module manages the allocation and deallocation of temporary registers throughout the assembly program, striving to use the minimal number of registers for each operation.

6.3. MyStructs

The MyStructs file contains all the global structs, macros, enums, and error reporting method used by all the files in the CPL compiler project.

6.4. Hash

The Hash file contains the implementation of Hash table, used by the compiler’s symbol table. The bison parser updates the symbol table each time it recognizes a variable declaration.