CSE305 Computer Architecture
201911036 Chanjung Kim
Professor Daehoon Kim
Project #1 – Simple MIPS Assembler

Introduction

An assembler is a program that compiles the given assembly code into an object file, which contains some constants and machine codes. Although modern compiler toolchains generate a symbol table and a relocation table within the output object file, this assembler does not generate them.

This assembler uses nine types of instructions internally for the consistency of the parsing logic. The format is as follows (unnamed fields are automatically initialized to zero by the assembler):

R format: `ADDU AND NOR OR SLTU SUBU`

6 bits	source1: 5 bits	source2: 5 bits	destination: 5 bits	5 bits	function: 6bits

`op destination, source1, source2`

JR format: `JR`

6 bits	source: 5 bits	15 bits	function: 6 bits

`op source`

SR format: `SLL SRL`

11 bits	source: 5 bits	destination: 5 bits	shiftAmount: 5 bits	Function: 6bits

`op destination, source, shiftAmount`

I format: `ADDIU ANDI ORI SLTIU`

operation: 6 bits	source: 5 bits	destination: 5 bits	immediate: 16 bits

`op destination, source, immediate`

BI format: `BEQ, BNE`

operation: 6 bits	source: 5 bits	destination: 5 bits	offset: 16 bits

`op source, destination, offset`

II format: `LUI`

operation: 6 bits	5 bits	destination: 5 bits	immediate: 16 bits

`op destination, immediate`

OI format: `LB LW SB SW`

operation: 6 bits	operand1: 5 bits	operand2: 5 bits	offset: 16 bits

`op operand2, offset(operand1)`

J format: `J JAL`

operation: 6 bits	target: 26 bits

`op target`

LA format (pseudo instructions): `LA`

`op destination, target`

Project Structure

The project contains three directories: `Public`, `Source`, and `Tests`. `Public` contains header files, `Source` contains C++ source files, and `Tests` contains unit tests. Source files in Tests are not necessary to build `runfile`.

Source contains 5 source files, whose roles are as the following:

- `Tokenization.cc` implements source tokenization.
- `Parsing.cc` parses tokenization result.
- `Genertion.cc` implements machine code generation.
- `File.cc` contains file I/O functions.
- `Main.cc` contains the entry point and converts errors into strings.

Build Instructions

This assembler is written in C++17. Although GCC 7 supports C++17, it lacks some standard C++17 headers (e.g. `<filesystem>` or `<charconv>`). To properly build the program, please use GCC 9 or higher. To install GCC 9 on Ubuntu 18.04, run:

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt update
sudo apt install gcc-9 g++-9
```

This project uses CMake 3.13. To build the program with CMake, run:

```
mkdir build
cd build
cmake -DCMAKE_CXX_COMPILER=g++-9 ..
cmake --build .

./runfile input.s
```

If CMake 3.13 or higher is not available in your system, you can use this command instead:

```
g++-9 -std=c++17 -I./Public  \
      -o runfile             \
      ./Source/File.cc       \
      ./Source/Generation.cc \
      ./Source/Main.cc       \
      ./source/Parsing.cc    \
      ./Source/Tokenization.cc

./runfile input.s
```

Acknowledgements

The parsing logic was inspired by nom, a Rust parser combinators library.