Project 1: Simple MIPS assembler Report

추성재 (201911185)

1. 개발 환경

개발에 사용한 언어는 C++이다. VMware workstation player에서 실행되는 Ubuntu 22.04에서 주로 개발했다. 사용한 컴파일러는 g++ 11.3.0 version이며, make를 이용해 빌드했다. 또한 최종결과를 VMware workstation player에서 실행되는 Ubuntu 20.04에서 g++ 9.4.0 version을 이용해 빌드해보았고 성공적으로 빌드되었다.

제출한 압축 파일 속에는 소스 파일과 그에 상응하는 헤더 파일, 리포트 파일이 있다. 추가로 lms에 올라와있는 sample.s, sample2.s도 있다.

2. 컴파일 방법

"00_FinalProject" 폴더 내에 들어온 뒤 shell에서 다음과 같이 컴파일한다.

```
g++ 03_CodeManage.cpp 02_InstructionConvert.cpp 01_DataManage.cpp 00_main.cpp -o mipsAssemblerFinal
```

컴파일을 완료하면 mipsAssemblerFinal 이라는 이름의 실행파일이 나온다.

3. 실행 방법

다음과 같이 변환하고자 하는 파일을 뒤에 인자로 넣어서 실행하면 된다.

```
./mipsAssemblerFinal sample.s
```

실행 후 인자로 넣은 파일의 이름과 동일한 형태의 *.o 형태의 바이너리 파일이 어셈블러와 동일한 폴더에 생성된다.

4. 코드의 flow

이번 프로젝트의 코드는 다음과 같이 네 개의 소스코드로 이루어져있다.

- 00_main.cpp

  어셈블러의 전반적인 흐름을 담당한다. 어셈블리 코드 파일을 읽어서 라인별로 저장한 후, 코드 속 data와 Instruction들을 분리해 16진수 문자열로 해석한 후 *.o 형태의 바이너리파일로 저장한다.

- 01_DataManage.cpp

  데이터를 변환하고 판별하는 함수들이 있다.

Function	Work
DecimalToHex	10진수 integer를 받아 16진수 문자열로 반환한다.

SplitLine	seperator 문자를 기준으로 문자열을 분리해 std::vector<std::string>형태로 반환한다.
StrHaveChar	입력된 문자열이 특정 문자를 가지고 있는지 판별해 bool 값으로 결과를 반환한다.
NumToBit	std::string 형태의 수와 크기를 받으면 이에 해당하는 이진수 값의 std::string 형태를 크기에 맞춰서 반환한다.

- 02_InstructionConvert

  instruction을 16진수 문자열로 변환하는 함수와 명령어들의 op, funct 정보를 담은 std::map container가 있다.

Function	Work
InstrConvert_R	R형식의 Instruction을 16진수 문자열로 반환한다.
InstrConvert_I	I형식의 Instruction을 16진수 문자열로 반환한다.
InstrConvert_J	J형식의 Instruction을 16진수 문자열로 반환한다.
InstrConvert_Control	main에서 저장한 Instruction을 받아 명령어 형식 별로 변환 함수를 불러온다.

- 03_CodeManage.cpp

  어셈블리 코드 속 label들의 주소를 매칭시키고, main에서 읽어온 data와 Instruction들을 모두 16진수 문자열로 변환하는 함수들이 있다.

Function	Work
DataLabeling	main에서 저장한 .data에서 .text 사이의 코드들을 읽은 후 label이 있으면 label에 해당하는 메모리 주소를 label을 key 값으로 사용하는 map container에 저장한다.
TextLabeling	main에서 저장한 .text 이후의 코드들을 읽은 후 label이 있으면 label에 해당하는 메모리 주소를 label을 key값으로 사용하는 map container에 저장한다.
DataSave	코드 속 데이터 값들을 16진수 문자열로 변환한다.
InstrSave	Instruction 속 label들을 앞서 구한 주소들로 바꾼 후 InstrConvert_Control에 Instruction들을 넘긴다.

어셈블러 코드의 flow는 다음과 같다.

1. main에서 어셈블리 코드를 읽어서 라인별로 vector에 저장한다. 이때, .data와 .text를 이용해 data를 표현하는 코드와 instruction을 표현하는 코드를 나누

어서 저장한다. 라인별로 들어온 코드들은 저장될 때 SplitLine에서 단어 단위로 나누어서 저장된다.

2. 그 후 코드 속에 있는 label들이 나타내는 주소들을 DataLabeling과 TextLabeling을 이용해 label을 key로 하고 label이 나타내는 주소를 value로 하는 map에 저장한다.

3. 그 후 .data 아래에 있는 .word의 값들과 .text 아래에 있는 instruction들을 DataSave와 InstrSave를 이용해 16진수 문자열로 바꾼다. 이때, InstrSave는 Instruction의 주소를 PC 변수를 이용해 추적해서 InstrConvert_Control에 넘긴다.

4. 그 후 16진수 문자열들을 *.o 형식의 이름을 가진 바이너리 파일로 저장해 출력한다.