Project 1: Simple MIPS assemble Report

201811061 박동진

1. 소스 코드에 대한 간략한 설명

이 소스코드는 C++로 작성되었습니다.
Part는 소스코드의 주석으로도 표시하겠습니다.

Part 1.
ifstream으로 파일을 입력받습니다.
.data, .text로 문자열을 구분하여 vector<string> datas, vector<string> texts으로 각각 저장합니다.

Part 2.
datas를 colon이 있으면 isLabel = true로 지정해 조건문으로 구별을 하였고 label과 그 주소값을 map<string, unsigned int> dataLabel에 저장하였습니다. 또한, ".word"를 기준으로 data를 구분하여 bitset STL을 사용해 16진수와 10진수를 2진수로 바꾸고 이를 string으로 변환해 vector<string> variables에 저장하였습니다. 각 data의 주소값을 addr2(data들의 주소 : 0x10000000)에 저장하고 반복문이 끝날때마다 addr2 = addr2 + 4를 통해 4바이트씩 늘어나는 주소값을 표현하였습니다.

Part 3.
마찬가지로 texts도 colon을 기준으로 isLabel = true인 instruction과 그 주소값을 map<string, unsigned int> textLabel에 저장하였습니다. 각각의 instruction의 주소를 addr(text들의 주소 : 0x00400000)에 저장하고 반복문이 끝날때마다 addr = addr + 4를 통해 4바이트씩 늘어나는 주소값을 표현하였습니다. 그리고 texts를 한단어씩 읽어들여 instruction의 종류별로 조건문을 만들었습니다. 각 instruction 별로 다른 op, shamt, funct를 조건문 안에서 설정해주었습니다. 또한, format별로 처리하는 함수를 만들었습니다.

Part 4.
addiu 같은 instruction rt, rs, imm의 형식을 twoRegImm 함수를 따로 만들어 처리하였습니다.
addu 같은 instructions rd, rs, rt의 형식을 threeReg 함수를 따로 만들어 처리하였습니다.
beq 같은 instructions rs, rt, label의 형식을 twoReg 함수를 따로 만들어 처리하였습니다.
J, JAL은 특이한 케이스라 생각하여 함수를 만들지 않고 조건문안에서 구현하였습니다.
JR 같은 instructions rs의 형식을 oneReg 함수를 따로 만들어 처리하였습니다.
lw 같은 instruction rt, address의 형식을 oneRegAddr 함수를 따로 만들어 처리하였습니다.
sll 같은 instruction rd, rt, shamt의 형식을 twoRegShift 함수를 따로 만들어 처리하였습니다.

Part 5-1, 5-2
이렇게 반복문이 끝나면 instructions에는 32개의 이진수가 들어있는 string이 저장되어있습니다. 하지만 label의 주소값들은 이진수가 아닌 "Label의 이름:" 이 저장되어있습니다. 이들은 모두 string의 끝부분에 append하도록 하였습니다. 여기서 문자열을 처음부터 끝까지 순차적으로 처리하기 때문에 아직 처리되지 않은 label의 주소값이 인스터럭션에 쓰이는 경우가 있었습니다. 이런 경우, 특정 instructions 값 뒤에 "label의 이름:" 과 그 label의 주소값을 append하였습니다. Beq와 bne 조건문에 구현하였습니다.

(Part 5-3)
특이한 형식인 la는 상위 16비트는 lui instruction으로 저장을 하고 bool checkZero로 하위 16비트가 모두 0이 아닌 경우만 ori instruction으로 저장을 하는 형식으로 처리하였습니다.

Part 6.
또 다른 반복문을 사용해 instructions의 label 이름을 textLabel에서 찾아서 주소로 바꾸어주었습니다. label이름을 instructions에서 지우고 대신 주소를 이진수로 표현하여 append하였습니다. Label을 이용하는 인스터럭션중 label의 주소를 저장하는 비트를 26개 사용하는 경우, 비트를 16개로 사용하는 경우로 나누어 구분하였습니다. exit label을 사용하는 경우는 label의 주소 비트를 26개 사용하는 J format이라는 것을 생각해 조건문을 추가하여 exit label 주소를 처리하였습니다.

Part 7-1, 7-2
instructions과 variables는 2진수로 표현된 string입니다. 이를 vector<unsigned int> instructionsInt, vector<unsigned int> variablesInt 로 10진수 변환을 합니다. instructionsInt와 variablesInt의 size가 각각 text section size와 data section size입니다. sstreamstring의 hex변환을 이용해 각 데이터를 10진수에서 16진수로 바꿉니다. 그 이후 string이 된 결과값을 ofstream으로 text를 .o 파일로 출력합니다.

2. 컴파일/실행 방법, 환경

-환경
이 소스코드(main.cpp)는 Ubuntu 20.04.4 64bit 환경(VirtualBox) 에서 작성되었습니다. 컴파일러는 g++ 9.4.0 을 사용하였습니다.

-컴파일/실행
터미널에서 main.cpp와 input 파일이 있는 디렉토리로 이동합니다. 터미널에 g++ -o runfile main.cpp 를 입력합니다. 터미널에 ./runfile <input 파일의 이름.s>를 입력합니다. 예를 들어 input파일이 sample.s 일 경우, 터미널에 ./runfile sample.s 를 입력하면 됩니다. 그러면 같은 디렉토리에 sample.o 파일이 생기게 됩니다. sample.o 파일이 output 파일입니다.