컴퓨터 구조 Project1 보고서

202011125 유선우

1. 컴파일/실행 방법, 환경

코딩과 테스트는 docker를 이용한 ubuntu 20.04에서 진행하였다.

해당 환경에서 g++의 버전은 9.4.0을 사용하였고,

다음 command를 통해서 컴파일과 실행을 할 수 있었다.

$ g++ assembler.cpp -o runfile

$ ./runfile <assembly file>

2. 코드 작동 Flow

Assembler의 동작을 하기 위해서는

1) Input file parsing
2) Each Label address Checking (label 주소를 저장하는 instruction을 위해)
3) Data size, Text size 저장
4) Instruction을 line by line으로 변환해서 저장
5) Data값 저장

의 순서를 거쳐야 한다.

이 과정에서 유의해야할 점은 'la' instruction은 2개의 instruction으로 해석될 수도 있다는 점이다.

2개의 instruction으로 해석되는지 여부에 따라 'la' instruction 뒤에 위치한 label의 주소도 달라지게 되고, j Format instruction이나 bne, beq와 같은 주소를 사용하는 instruction이 바이너리 코드로 변환될 때의 값이 달라지게 된다.

따라서 위의 과정에서 추가적으로 instruction을 한 줄씩 읽으며 각 'la' instruction

에 대해 2개의 instruction으로 분리되어야 하는지 여부를 판단하고 label 주소와 data size를 재조정하는 함수를 추가하였다. (pseudoTrans 함수)

따라서 assembler.cpp 코드는 다음과 같이 작동한다.

1) parser함수에서 input file을 line by line으로 읽어 data는 data vector에 text는 text vector에 push_back() 한다.
2) labelChecker함수에서 label의 주소를 map 구조를 갖는 data_list에 저장하고, data, text size를 저장한다.
3) pseudoTrans함수에서 line by line으로 instruction을 읽으며 la를 label 주소에 맞게 lui instruction 또는 lui + ori instruction으로 변환하여 다시 text에 저장한다.

   여기서 lui와 ori는 상, 하위 16비트를 사용하는데 pseudoTrans함수가 실행되면서 lui + ori instruction으로 바뀌는 경우 해당 la보다 밑에 있는 label 주소를 +4 해주기 때문에 lui와 ori를 다시 text에 저장할 때, imm값이 아닌 #label을 넣어 후에 #을 발견하면 label 주소의 상, 하위 16비트를 사용하도록 하였다.

   또한 이후 assembler 함수에서 진행하는 과정에서 label이 읽히지 않도록 하기 위해 text vector에서 label은 모두 제거해준다.
4) 이후 assembler함수에서 text vector와 data vector를 word별로 쪼개서 instrucion을 binary하게 변환한 후 bin vector에 저장하였다. 이 과정에서 instruction마다 조건에 맞게 binary code로 변환해야하기에 binTransformer함수를 호출해 처리하였다.
5) binTransformer함수에서는 assembler함수에서 받아온 word vector를 binary code로 변환한다.

   이 과정에서 각 instruction에 특성에 맞게 변환이 되도록 조건문을 이용하여 조절하였다.

모든 과정에서 int값을 2진수로 변환하는 과정이 너무 많이 사용되어 추가적으로 getBinData함수를 정의하였다. 이 함수는 특정 숫자를 이진수로 변환하여 하위 n(파라미터로 입력받음)개의 bit만을 string으로 return하는 함수이다.