**컴파일러의 기초 프로젝트 #4 결과 보고서**

전기정보공학부 2016-10769 이권형

**1. 전체적인 디자인에 대한 설명**

각각의 operation이나 stack의 구조, calling convention은 ppt를 참고하여 구현하였다. calling convention은 Example의 2번째르 사용하였다.

project3에서 사용하였던 code를 기반으로 imbedded action에서 해당하는 code를 output file에 print하도록 하였다.

세부적인 구현은 다음과 같다.

(1) variable

모든 variable은 stack상에서 변수에 대한 주소로 존재한다. 이때 변수의 값을 참조하거나 arithmetic operation이 있으면 변수의 주소에 대해 fetch한 뒤 그 값을 사용한다. pointer와 & operand의 경우 decl struct에 새로 추가된 fetched 변수를 통해 구현되어 있다. & operand는 변수의 주소를 나타내므로, fetched에 표시를 하여 다음에 사용될 시 fetch를 바로 하지 않고 한번의 fetch는 무시한다. 이로써 변수의 주소를 참조하는 것과 같은 효과를 가진다.

global local variable은 모두 offset을 통해 참조된다. 각각의 decl struct에 offset이라는 변수가 있고, 이는 현재 scope에서의 variable size의 합에 따라 결정된다.

(2) Arithmetic/Logic instruction

arithmetic/logic instruction의 경우 stack에 미리 존재하는 variable 두 개에 대해 단순히 operation을 실행한다.

(2) Conditional jump

branch나 for, while 등의 control flow를 조작하는 경우 jump instruction을 사용한다. 이 때 label이 필요한데, label\_counter 변수를 통해 이를 구현하였다. label\_counter는 global 변수로 label의 번호를 저장하고, 사용될 때마다 1씩 증가한다. 이를 통해 conditional jump와 함께 채 control flow instruction을 구현하였다.

while의 경우 loop\_label과 end\_label을 따로 저장하고, loop\_label에서 조건을 check하고 end\_label로 conditional jump하도록 되어 있다. nested문을 구현하기 위해 WHILE을 label struct 형식으로 지정하고, 각각의 label number를 WHILE token struct에 저장하는 식으로 되어 있다.

for문의 경우 nested는 고려할 필요가 없어 global label을 이용한다. 먼저 initialize code와 check code를 실행하고, statement로 jump한다. statement의 실행이 끝나면 for문의 마지막 stmt가 실행되고, 이후 다시 check label로 jump하는 것을 반복한다.

(3) I/O

write\_함수의 경우 stack에 있는 값을 fetch한 뒤 print한다.

(4) Function call

calling convention의 경우 ppt에서 제공된 것 중 calling convention 2를 사용하였다.

(5) struct의 연산

struct의 경우 다른 variable과 다르게 size가 1이 아니므로 추가적인 구현이 필요하다. struct의 모든 변수는 다른 것과 같이 stack에 쌓여 있고, struct의 연산의 경우 size만큼 recursive하게 variable에서의 code가 반복된다. return, parameter의 경우도 위와 같다.

(6) 기타

const\_expr->expr reduction의 경우 array에서 사용되는데, array에서의 index에서는 expr로 reduction되는 과정에서 생성된 code가 사용되지 않는다. 따라서 이 경우 expr 이전에 output file의 File descriptor를 잠시 linux에서 제공하는 null device로 설정한 뒤, reduction이 일어날 때 원래의 file descriptor로 복구한다. 이로써 사용되지 않는 code를 출력하지 않을 수 있다.

INCOP나 DECOP의 code는 ppt와 example로 주어진 code를 활용하여 구현하였고, basic implementation은 모두 주어진 example code를 참고하여 구현하였다.

**2. 완성도에 대한 설명**

주어진 grammar에 대한 모든 operation을 구현하였다. struct assign, return, parameter 이외에도 for, while, if, continue, break 등의 과제를 해결하였다.