Design and Development of Compiler for C- Language

Phase 4: Code Generation

과목명: [CSE4120] 기초컴파일러구성

담당교수: 서강대학교 컴퓨터공학과 정성원

개발자: 24 조

정서원(20150959)

Project 4 결과 보고서

프로젝트 제목: Design and Development of Compiler for C- language

Phase 3: Code Generation

제출일: 2019. 6. 25

참여조원: 정서원(20150959)

I. 개발 목표

Semantic analyzer 로 수정된 C 파일의 syntax tree 를 기반으로 TM 코드를 생성하는 Code Generator 를 구현한다.

II. 개발 범위 및 내용

가. 개발 범위

- ✓ pc, mp, fp, ac, ac1 레지스터들을 활용하여 syntax tree 의 각 node 에 알맞게 코드를 생성한다.
- ✓ code.c 와 code.h 는 레지스터와 메모리를 이용하여 tm 코드를 생성하도록 하는 코드이며, 별도로 수정할 필요가 없었다.
- ✓ cgen.c 와 cgen.h 에서는 code.c, code.h 에서 작성한 함수들을 적절히 활용하여 C 코드를 tm 코드로 적절히 변환할 수 있도록 한다.

나. 개발 내용

SPIM Simulator 또는 TINY Machine Simulator 로 실행했을 때 실행되는 파일을 생성할 수 있도록 하는 것이 이번 프로젝트의 요구사항이었는데, 후자를 선택하여 .tm 파일을 생성하는 code generator 를 구현하였다.

III. 추진 일정 및 개발 방법

가. 추진 일정

- ✓ 6/22: 교재를 참고하여 프로젝트 이해
- ✓ 6/23: code.c 와 code.h 분석
- ✓ 6/24: cgen.c 와 cgen.h 작성
- ✓ 6/25: 결과 보고서 작성 및 제출

나. 개발 방법

이전 프로젝트들과 같이 Linux 환경에서 개발하며, 교재를 참고한다.

다. 연구원 역할 분담: 20150959 정서원 100%

IV. 연구 결과

1. 합성 내용:

1 차와 2 차 프로젝트에서 lexical analyzer 와 LALR parser 를 구현했었고, 3 차 프로젝트에서는 2 차 프로젝트에서 구성한 abstract syntax tree 를 이용하는 semantic analyzer 를 구현하였다. 이번 프로젝트에서는 이를 바탕으로 C- 코드를 three-address code 로 변환할 수 있도록 하는 code generator 를 구현한다. Three-address code 란 컴파일러의 코드 변환 기능을 최적화하기 위한 intermediate code 로, 각각의 instruction 은 최대 세 개의 항을 가진다. 사용하는 opcode 로는 LDA, LDC, LD, ST, ADD, SUB, MUL, DIV, JEQ, IN, OUT 이 있다.

- ✓ LDA: 메모리의 주소를 레지스터에 load
- ✓ LDC: constant 를 레지스터에 load
- ✓ LD: 메모리에 저장된 값을 레지스터에 load
- ✓ ST: 레지스터에 저장된 값을 메모리에 load
- ✓ ADD: 덧셈
- ✓ SUB: 뺄셈

- ✓ MUL: 곱셈
- ✓ DIV: 나눗셈
- ✓ JEQ: 레지스터의 값이 0 일 때 특정 위치로 jump
- ✓ IN: 입력
- ✓ OUT: 출력

2. 분석 내용

1) Stmt node

① **IfK:** C language 에서의 selection statement 에는 if-else if-else 문과 switch-case 문이 있지만, C-에서는 if-else 만 사용한다. 또한 short-circuit evaluation 을 채택한다.

```
if (E) S1 else S2:
〈code to evaluate E〉
fjp L1
〈code for S1〉
ujp L2
lab L1
〈code for S2〉
lab L2
```

② **WhileK:** C language 에서의 iteration statement 에는 while 문, do-while 문, for 문이 있지만, C-에서는 while 문만 사용한다.

```
while (E) S:
llab L1
〈code to evaluate E〉
fjp L2
〈code for S〉
ujp L1
lab L2
```

③ **ReturnK:** 해당 노드에 child 가 있으면 code 를 생성하고, 없으면 pass 한다.

2) Exp node

① OpK

attr.op 이 ASSIGN 인 경우와 나머지 경우로 나눠야 한다. 전자에서는 lhs 의 메모리에 rhs 의 값을 저장해야 한다. Lhs 가 array 인 경우 먼저 메모리에 ac 를 load 한다. 그리고 변수의 index 를 load 하기 위한 코드를 생성하여 그 값을 ac 에 저장하고, 주소값을 ac1 에 저장한다. 그 다음 할당하려는 식별자의 주소에 ac1 에 저장된 주소를 더하여 위치를 구한다. Lhs 가 int 인 경우에는 식별자의 주소를 찾아 그 위치에 값을 저장하면 된다.

attr.op 이 사칙연산(+, -, *, /)이거나 비교 연산자(〉, 〉=, 〈, 〈=, ==, !=)일 경우에는 기존 코드를 그대로 적용하면 된다.

② **ConstK:** LDC 를 이용해 constant 를 메모리에 load 한다.

③ **IdK:** 이 node 도 OpK 의 assign 과 같이 array 인 경우와 int 인 경우로 분리해야 한다. Array 인 경우에는 index 를 이용해 주소값을 구한 다음 그 주소에 있는 값을 load 하지만, int 인 경우에는 그 메모리에 저장된 값을 바로 load 한다.

④ **CallK:** 함수를 호출하는 statement 의 node 이다. 첫 번째로 post-order traversal 을 통해 argument 들에 대한 코드를 생성하고, argument 의 개수를 구한다. 그 다음 argument 의 개수만큼 sp 를 조정하고 그 아래 fp 를 저장한다. Fp 를 sp 의 위치에 가져온 다음 return address 를 저장하고, syntax tree 에 저장되어 있는 해당 함수의 location 을 이용해 함수를 호출한다.

3) Decl node

① **FuncK:** 함수를 선언하는 statement 의 node 이다. 함수의 location 을 load 하고 메모리에 추가하여 할당한다. Main 함수는 그 location 을 미리 저장해서 나중에 코드를 생성할 때 가장 먼저 불러올 수 있도록 한다. 또한 input 함수와 output 함수도 구현하여 그 기능을 하도록 한다. Parameter 에 대해서도 CallK 의 argument 들과 같이 post-order traversal 을 이용해 코드를 생성한다.

② **CmpndK:** Compound statement 가 선언됐을 때의 node 이다. 선언된 local 변수들이 있을 때 메모리에 영역을 할당해준다.

3. 제작 내용 및 시험 내용

```
void main(void){
    int a;
    int b;
    int c;
    int d;
    a =1;
    b=2;
    c= 3;
    d = 4;
    a = a+b+((c+d)+5+6)+7+8*9;
    output(a);
    b = c*d -b;
    output(b);
    c = d/(c-1);
    output(c);
    d = a/b - c*c;
    output(d);
}
```

```
cse20150959@cspro7:~/project4_24$ ./tm test2.tm
TM  simulation (enter h for help)...
Enter command: g
OUT instruction prints: 100
OUT instruction prints: 10
OUT instruction prints: 2
OUT instruction prints: 6
HALT: 0,0,0
Halted
```

5. 평가 내용

위의 test2.c(error-free) 파일을 .tm 파일로 변환했을 때 잘 실행되는 것을 확인할 수 있다.

6. 기타

혼자 진행하는 프로젝트라 협업에 쏟아야 하는 시간은 절약되었지만, 이해해야 하는 내용이 많아 오히려 더 많은 시간을 투자해야 했다.