Design and Development of Compiler for C- Language
(설계 프로젝트 수행 결과)

Phase 4: Design and Implementation of Code Generator

과목명: [CSE4120] 기초컴파일러구성
담당교수: 서강대학교 컴퓨터공학과 정 성 원
개발자: 6조
서현규 (20161596)
이동승 (20161616)
개발기간: 2019. 6. 13 - 2019. 6. 25

프로젝트 제목: Design and Development of Compiler for C-Language:
Phase 4: A Code Generator
제출일: 2019. 06. 25. 화
개발자: 6조 - 서현규 (20161596), 이동승 (20161616)

※ 참고사항

1. 본 프로젝트는 **SPIM**으로 실행했습니다. SPIM으로 실행해주시면 감사하겠습니다.

2. 아래와 같이 C-minus 파일을 컴파일 하시면

   ```
   ./project4_6 test.tny
   ```

   생성되는 SPIM 코드 파일은 **test.tm** 입니다. spim -file test.tm으로 실행하실 수 있습니다.

3. 만약 컴파일하려는 C-minus 파일의 파일명 또는 경로 맨 앞에 점 "**.**" 이 있으면 SPIM 코드 파일이 숨김파일로 생성됩니다. 예를들어, 아래와 같이 test.tny를 컴파일 하시면,

   ```
   ./project4_6  ../SOME_PATHS/test.tny
   ```

   경로 "**..**/SOME_PATHS/test.tny" 맨 앞에 점이 있기 때문에,
   생성되는 SPIM 코드 파일의 파일명은 test.tm 이 아니라 **.tm**입니다.
   즉 숨김파일로 생성됩니다. 이 경우 ls 커맨드로는 파일이 생성된 것을 확인할 수 없기 때문에 **ls -a** 커맨드로 파일이 생성된 것을 확인해주시면 감사하겠습니다. 이 경우 실행은 동일하게 spim -file **.tm**으로 해주시면 감사하겠습니다.
   (C-minus 파일의 파일명이나 경로 맨 앞에 점이 없으면 이 3번 참고사항은 무시하셔도 괜찮습니다.)

I. 개발 목표

이번 프로젝트에서는 project 1,2,3에서 구현한 lexical analyzer, parser, semantic analyzer를 이용해서 SPIM architecture에 맞는 머신 코드를 생성하는 것을 목표로 한다.

II. 개발 범위 및 내용

가. 개발 범위

SPIM 머신 Code Generator를 개발한다.

나. 개발 내용

C- 소스코드를 해석하여 SPIM 머신에서 실행 가능한 머신 코드를 생성하는 기능을 컴파일러에 추가한다. 그리고 input, output 내장 함수를 추가한다.

III. 추진 일정 및 개발 방법

가. 추진 일정

6월 12~13일 : 프로젝트 전반적인 이해
6월 14~18일 : cgen.c, cgen.h를 수정하여
Compound body의 statement와 expression을 구현함.
6월 19일~22일: cgen.c의 function declaration과 call procedure를 구현함
6월 23일 : testcase를 만들어보고 실행
6월 24일 : 보고서 작성

나. 개발 방법

앞선 프로젝트에서 구축한 AST를 순회하며 node 타입에 따라 해당하는 spim code를 기계적으로 생성한다. SPIM system call을 사용해서 input, output 내장 함수를 구현한다. Runtime environment는 SPIM 메뉴얼에서 제시하는 바와 유사하게 구축한다.

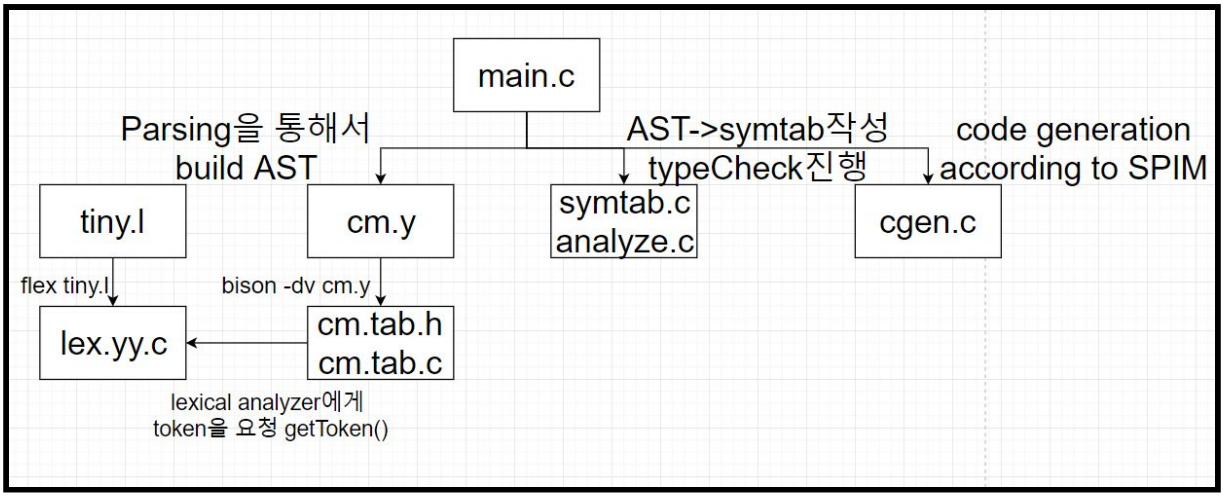
다. 연구원 역할 분담

서현규 :
code.c, code.h, cgen.c cgen.h, genStmtCode, genExpCode, genDeclCode 구현(**50%**)

이동승 :
cgen.c의 genDeclCode와 function call 에 대한 recursive구현 및 보고서 작성 (**50%**)

IV. 연구 결과

1. 합성 내용



Project4에 대한 Flow는 위와 같다. cm.y 에서 생성한 AST가 analyze.c 에서 오류가 없다고 판단하면 cgen.c는 AST를 순회하며 node type에 따라 적절한 SPIM 머신 코드를 생성한다. 생성된 코드는 SPIM simulator 를 통해 실행 가능하다.
개발 결과물의 사용 방법은 다음과 같다.

```
$ ./project4_6 <tny 형식 파일>
$ spim -file <tm 형식 파일>
```

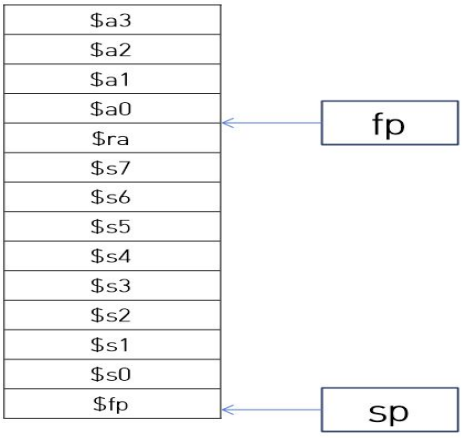
먼저 project4_6 실행 파일을 이용해서 tny 확장자 파일을 컴파일한다. 컴파일 결과 tm 확장자 파일이 생성되는데, 이 파일을 spim 머신에 넣으면 실행할 수 있다.

2. 분석 내용

code.c 와 code.h 에는 두 가지 API 함수 emitCode와 emitComment를 개발했다. emitCode는 문자열을 인수로 받아 SPIM 코드 파일에 출력 한다. 머신코드만 있으면 가독성이 떨어진다는 것을 개발 중간에 발견하여 주석을 작성해주는 함수 emitComment를 개발했다. emitCode를 통해 생성된 코드는 symbol table의 scoping 을 따르는데, 앞선 프로젝트에서 static scope rule 에 따라 symbol table을 구축했기 때문에 emitCode 는 **static scope rule**을 지원한다.

cgen.c 와 cgen.h 에는 AST를 순회하며 코드를 생성하는 cGen 함수를 개발했다. cGen은 AST를 node 별로 sibling을 따라 순회하며 node 의 종류 (statement, declaration, expression) 에 따라 getStmtCode, genDeclCode, genExpCode 함수를 호출해서 코드를 생성하도록 한다. 코드 생성시 emitCode api 함수를 이용해서 코드 파일에 코드를 생성했다. 그리고 genStmtCode, genExpCode, genDeclCode 는 다시 cGen 함수를 재귀적으로 호출하여 child 노드에 대한 코드를 생성할 수 있다.

genDeclCode()는 declaration 노드에 대응하는 머신 코드를 생성하는 함수다. variable declaration일 경우 global, local 여부를 구분한다. global 변수의 주소는 0x10000000 부터 시작하며 global 변수가 선언 될 때 마다 global 영역의 마지막 주소를 해당 변수에 할당하고, global 영역을 변수 크기만큼 증가 시킨다. 이 과정은 allocGlobal 함수로 구현했다. 지역 변수의 경우 addiu $sp, $sp, -(변수크기) 와 같이 변수 크기만큼 스택 영역을 할당하는 식으로 구현했다. function declaration일 경우 먼저 함수이름과 동일한 라벨을 생성하고, stack frame을 형성했다. stack frame 구조를 그림으로 나타내면 다음과 같다.



재귀 호출을 대비해서 $a0 ~ a3 register와 ra register를 저장한 뒤 $s0 ~ s7 register들을 저장하고 마지막으로 $fp를 저장했다. 그리고 fp는 ra와 a0의 사이를 가리키게 했다. $s0 ~ s7 register를 저장한 것은 spim manual에 이 register들이 preserve 되어야 한다고 명시되어 있었기 때문이다. stack frame을 구성하는 코드를 생성하면 함수 body에 해당하는 코드를 생성한다. 함수 몸체 코드 생성이 끝나면 stack frame을 해제하는 코드를 생성해야 한다. stack frame 할당의 역순으로 stack frame을 해제하는 코드를 생성했다. 그리고 stack frame 해제하는코드에는 clean_label이라는 label을 붙여서 return statement 호출 시 해당 label로 바로 jump할 수 있도록 구현했다.

getStmtCode()는 statement 노드에 대응하는 머신 코드를 생성하는 함수다. Compound statement의 경우 Compound statement의 declaration part에 해당하는 코드를 먼저 생성하고

code part에 해당하는 코드는 그 다음에 생성했다. 코드 생성이 끝나면 할당된 stack 을 해제했다.

If statement는 수업시간에 배운대로 코드를 생성했다. expression 에 해당하는 코드를 생성한 뒤 expression이 false일 경우 else 로 jump하는 코드를 생성했다. 그리고 if의 body에 해당하는 코드를 생성 한 뒤 if문을 바깥으로 jump 하는 코드를 생성하고 마지막으로 else의 body에 해당하는 코드를 생성했다. 적절한 위치에 label들을 생성하여 jump가 원활히 이루어지도록 했다.

While statement는 먼저 expression을 체크하는 부분으로 jump 하는 코드를 생성 하고 while 의 body에 해당하는 코드를 생성했다. 그리고 expression에 해당하는 코드를 생성하고 while의 body로 jump하는 코드를 생성했다. 적절한 위치에 label들을 생성하여 jump가 원활히 이루어지도록 했다.

Return statement는 return하는 expression에 해당하는 코드를 생성한 뒤 함수종료 지점 (clean_label) 으로 jump 하는 코드를 생성했다.

genExpCode()는 expression 노드에 대응하는 머신 코드를 생성하는 함수다. Assign operation은 먼저 right operand의 코드를 생성한다. Left operand가 배열이 아니고 전역변수 일 경우, 해당 변수의 전역 주소를 계산하여 그곳에서 right operand의 결과값을 저장한다. (전역주소는 **0x10000000** 부터 시작한다고 가정했다.) Left operand가 배열이 아니고 지역변수일 경우 left operand의 fp 기준 상대 주소로 right operand의 결과값을 저장한다. 배열일 경우 array subscription에 해당하는 코드를 생성하고 array subscription만큼 떨어진 상대 주소 또는 전역 주소에 right operand 결과값을 저장했다.

일반 연산자의 경우, left operand의 코드를 먼저 생성한 뒤 스택 공간을 4byte만큼 할당하고 할당 된 곳에 left operand의 결과값(v0 register)을 저장했다. 그리고 right operand의 코드를 생성하여 결과를 v0 register에 저장한 뒤 스택에 저장된 값을 s0 register 로 옮겼다. 결국 left operand 값은 s0에, right operand의 값은 v0에 있기 때문에 이를 각각의 operation에 따라 처리하면 된다. 계산이 끝나면 addiu $sp, $sp, 4를 해서 할당된 stack을 해제했다.

Id expression에 대해서는 assign operation에서 left operand를 구현할 때와 비슷하다. array가 아닌 경우 global 과 local variable로 나눠서 처리하고, array인 경우 subscription 코드를 생성한 뒤 referencing 하고 마지막에 lw $v0, 0($v0) 로 referencing한 값을 다시 $v0에 저장되게끔 구현했다. Const expression 에 대해서는 값 자체를 $v0에 올리면 된다.

Call expression은 argument에 해당하는 코드를 생성하고 각 argument를 $a0 ~ $a3 register에 저장한 뒤 jal로 해당 함수 영역으로 jump한다. 이는 일반적인 함수의 이야기고, input, output인 경우 SPIM manual 을 참조해서 구현했다. output 함수는 syscall을 활용해서 output_message를 출력하고, 첫번째 argument의 코드를 생성한 뒤 결과값을 syscall로 출력한 뒤 다시 new line을 출력하도록 구현했다. input도 마찬가지로 input message를 system call 로 출력 한 뒤 v0에 5를 저장해서 input system call로 입력 받을 수 있도록 구현했다.

3. 제작 내용

code.c, code.h
-emitCode(): 문자열을 입력받아 해당 문자열을 코드 파일에 쓴다.
-emitComment(): 문자열을 입력받아 해당 문자열을 코드파일에 주석으로 쓴다.
cgen.c, cgen.h
-codeGen(): '.data', '.text' 같은 기본적인 spim directive를 생성한다. AST를 인수로 받아 cGen()을 호출한다.
-cGen(): tree node를 인수로 받아 node 종류에 따라 genStmtCode, genExpCode, genDeclCode 함수를 호출한다.
- genStmtCode, genExpCode, genDeclCode: 분석 항목에서 밝힌 바대로 statement, expression , declaration 노드에 대응하는 코드를 생성한다.
-insertIOFunctions: input() 함수와 output() 함수 선언 노드들을 AST에 삽입하는 함수다.

4. 시험 내용

테스트에 사용한 **test1.tny, test2.tny**와 그 실행 결과는 다음과 같다.

(test1.tny)

```
int globalArray[100];

int func (int x) {
    return (x * 2) + 1;
}

void main (void) {
    int i;
    int x;

    x = input();

    if (x > 100)
        x = 100;

    i = 0;
    while(i < x) {
        globalArray[i] = func(i);
        i = i + 1;
    }

    i = 0;
    while (i < x) {
        if ( globalArray[i] < x)
            output(globalArray[i]);
        else
            output(globalArray[i] - 2 * globalArray[i]);
        i = i + 1;
    }
}
```

```
Input : 10
Output : 1
Output : 3
Output : 5
Output : 7
Output : 9
Output : -11
Output : -13
Output : -15
Output : -17
Output : -19
```

(test2.tny)

```
int calc (int x, int y) {
    return x * y + 1;
}

int func (int x) {
    if (x > 1)
        return calc(x, 2) + func(x - 1);
    else
        return 1;
}

void main (void) {
    int req;
    int sol;
    int i;
    int arr[3];

    i = 0;
    while ( i < 3) {
        req = input();
        sol = func(req);
        arr[i] = sol;
        output(arr[i]);
        i = i + 1;
    }
}
```

```
Input : 10
Output : 118
Input : 100
Output : 10198
Input : 2
Output : 6
```

다음은 C- language로 작성된 bubblesort를 SPIM으로 구동한 결과다.

.

```
cse20161616@cspro9:~/Compiler2019/project4_6$ spim -file bubble.tm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Input : 4
Input : 100
Input : 50
Input : 30
Input : 10
Output : 10
Output : 30
Output : 50
Output : 100
```

유명한 dynammic programming문제인 LIS(Longest Increasing Subsequence)를 구하는 문제를 C-에 맞게 구현하여 실행해 보았다. 아래는 그 결과다.

```
cse20161616@cspro9:~/Compiler2019/project4_6$ spim -file test_lis.tm
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Input : 5
Input : 3
Input : 1
Input : 2
Input : 4
Input : 5
Output : 4
```

5. 평가 내용

project의 목표인 C- source code에 대한 SPIM code generation을 완성하였다고 평가할 수 있겠다. 그리고 기본적인 variable참조와 function에 대한 recursive call 또한 구현할 수 있게 되었다고 생각한다.

V. 기타

1. 연구 조원 기여도

서현규: 50%,
이동승: 50%

2. 자체 평가

여러가지 복잡한 코드를 테스트했을 때도 결과가 정확히 나오는 것을 확인 할 수 있었기 때문에 과제를 올바르게 수행했다고 생각한다.

3. 느낀점

컴파일러의 마지막 프로젝를 수행함으로써 code generation까지 마무리 할 수 있어서 기뻤다.