Design and Development of Compiler for C- Language

Phase 4: Design and Implementation of Code Generator

과목명 : [CSE4120] 기초 컴파일러 구성

담당교수 서강대학교 컴퓨터공학과 정 성 원

5조

김민우(20141510)

정재훈(20151607)

**Project4 결과 보고서**

**1. 개발 목표**

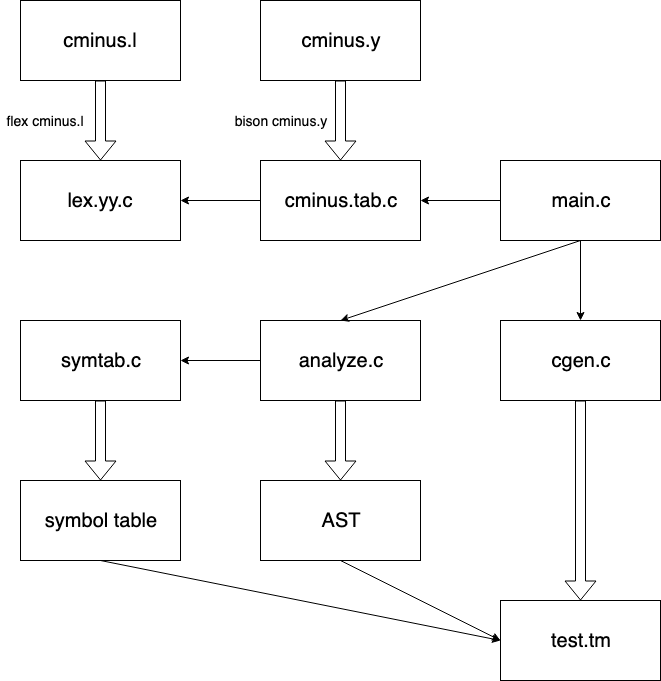
Project4의 개발 목표는 C- language에 대한 code generator를 구현하는 것이 목적이다. Code generator는 C- language로 작성된 소스코드를 MIPS Processor를 위한 assembly code로 translate 할 수 있어야 한다.

**2. 개발 범위 및 내용**

**가 개발 범위**

Project4는 Project3에서 구현한 syntax tree와 symbol table을 이용하여 code generator를 구현을 해야한다. Symbol table의 정보를 MIPS machine에서 직접적으로 사용할 수 있도록 하기 위해서 MIPS machine의 system call, register, system call, 함수 호출 규약 등을 고려해서 code generator를 구현해야 한다.

**나 개발 내용**



전체 소프트웨어 분석도

|  |
| --- |
| Stack Data |
|  |
| Dynamic Data |
| Static Data  (0x10000000~) |
| Text Segment |
| Reserved |

|  |
| --- |
| Argument1 |
| .  .  . |
| Control Link |
| Return Address |
| Local Variable |
| Temporary |

$sp

$fp

Activation Record & Memory Architecture

C- language로 작성된 소스코드를 구현한 컴파일러로 MIPS assembly code로 변환시키는 과정을 살펴보면, main함수에서 생성된 AST를 매개변수로 하여 genCode()를 호출한다. genCode()에서는 Scope가 전역인 경우를 처리하고 data segment와 text segment부분을 나눈 다음 cGen()함수를 호출한다. cGen()함수에서는 AST를 순회하면서 Node의 종류에 따라서 assembly code로 치환하는 작업을 수행한다.

**MIPS architecture 특징**

1. MIPS의 경우에는 0x10000000에서 증가하는 방향으로 global variable이 쌓이고, local variable의 경우 $(sp)에서 감소하는 방향으로 쌓인다.
2. 사용할 수 있는 레지스터의 수가 32개이다.
3. spim에서 MIPS assembly code를 실행할 경우, main 함수가 implicit하게 호출되므로 entry point는 main label이다.

**Static Scope 구현**

Static Scope를 구현하기 위해 기존 생성하였던 symbol table을 이용하였으며, TreeNode에 Symbol table node를 연결시켜 location이나 global 여부를 참조하도록 한다. 또한 Nested Function 선언이 없기 때문에 fp에 대한 offset 또는 global memory에 대한 offset만 유지하면 static scope를 구현 가능하다.

**genLabel()**

if문, while문에서 unique하게 label을 생성하기 위해서 호출하는 함수

**genCode()**  
Data segment와 code segment를 구분하고 cGen함수를 호출해서 MIPS assembly code를 생성한다. input\_prompt와 output\_prompt를 Data segment에 저장한다.

Exp, Variable, Const등의 표현식을 계산 후, 이를 temporary register 대신 stack을 임시변수 저장공간으로 사용한다. 따라서 마치 P machine 처럼 연산 결과는 stack( $sp )에 남아 있다.

**cGen(TreeNode \*tree, int isAddr, FILE \*codeFile)**

해당 변수가 Address로 표현되어야 하는지 Value로 표현되어야 하는지 나타내는 isAddr 변수를 전달한다.

1. node의 종류가 FactorK(식별자, 배열, 상수)인 경우:

상수, 배열과, 변수를 구분한 다음, global영역과 local영역에 따라서 다르게 stack에 저장한다. global 영역은 static 영역 0x10000000을 기준으로 저장하며 local 영역은 $fp를 기준으로 저장한다. 주의해야 할것은 array가 parameter로 쓰였을 경우인데 이때는 해당 parameter를 array pointer 변수로 생각하여 참조를 다르게 해줘야 한다.

1. node의 종류가 StmtK(if, while, return, compound, exp)인 경우:

if, while 경우에는 일정한 패턴의 assembly 코드로 표현이 될 수 있다. 조건문에 대한 코드를 생성한 후 beq 명령어를 통해 branch 한다. 이때 label 생성이 병행되어야 한다.

return의 경우에는 $ra값을 return address을 통해 복구하고 $sp를 $fp로 초기화 한다. control link를 $fp에 집어넣고 마지막에 jr명령어로 ra주소로 jump한다. parameter에 대한 pop은 해당 function에 대한 총 parameter size를 기억하고 있다가 이를 토대로 pop해줘야 한다. return 값이 존재할 경우, 이를 $t0에 담아 전달한다.

Compound의 경우에는 child들을 호출하는 것으로 마무리한다.

Expression의 경우에는 Right handside를 먼저 code를 생성하고 그 후 Left handside code를 생성하고 해당하는 연산과 유사한 MIPS assembly 명령어를 사용하여 연산을 수행한다. 만약 Assignment( = )의 경우 lefthand side에서는 주소가 계산되어야 하므로 isAddr을 1로 켜준다.

1. node의 종류가 DeclareK(선언문)인 경우:

선언문은 단지 공간을 할당하는 용도 이므로 Local 일 경우에만 stack pointer($sp)를 밀어 공간을 확보해야한다.

1. node의 종류가 FuncK(함수 정의문)인 경우:

FuncK에서는 Call당시 argument가 stack에 제대로 들어왔다는 가정하에 activation record를 생성한다. 먼저 $fp에 $sp를 저장하고, return address 영역에 $ra를 넣는다. 그 후 해당 노드의 자식들에 대한 코드를 생성한다.

Function이 종료 될때, parameter에 대한 node를 traverse하면서 구한 Parameter size를 통해 stack을 pop한다.

void return의 경우 return문이 존재하지 않아, return sequence가 없기 때문에 default void return sequence를 삽입한다.

1. node의 종류가 ParamK(함수 parameter)인 경우:

해당 node에서는 단지 parameter size만 계산한다.

C- langauge에서 parameter로 배열의 주소나 int값이 들어올 수 있으므로 4byte를 parameter size에 항상 추가시킨다.

1. node의 종류가 CallK(함수 호출)인 경우:

input()과 output() 제외한 함수를 호출하면, 함수를 호출하기 위해서 준비하는 과정을 실행한다.

먼저 argument에 대한 computation을 진행한다. computation 마다 이는 자동으로 stack에 저장되므로 따로 stack에 저장하지않아도 상관없다. control link를 new activation code에 저장후 해당 함수 label로 jump한다. return 받을 값이 존재한다면 $t0에 담겨 오므로, 해당 값을 stack에 저장한다.

**input & output**

입출력함수의 경우, 적절한 register값을 대입하고 syscall을 호출하면된다. synatx tree를 생성하는 과정에서 input(), output()함수를 호출했을 때 발생할 수 있는 semantic error를 막기 위해서 symbol table에 input함수와 output함수를 st\_insert()로 직접 추가하였다.

**input()**

input함수는 input\_prompt(“Enter value for IN instruction : ”)를 출력하고, 사용자로부터 입력을 받는 함수이다.

v0 register에 system call code번호 4를 대입하고 a0 register에 input\_prompt 주소를 대입해서 syscall을 호출하면, 문자열 출력에 대한 system call이 호출된다.

v0 register에 system call code번호 5를 대입하고 syscall을 호출하면, 정수를 입력받는 system call이 호출된다. 사용자가 숫자를 입력하면, 숫자를 v0 register에 저장한다.

**output()**

output함수는 output\_prompt(“OUT instruction prints : ”)를 출력하고, 전달된 argument의 값을 출력하는 함수이다.

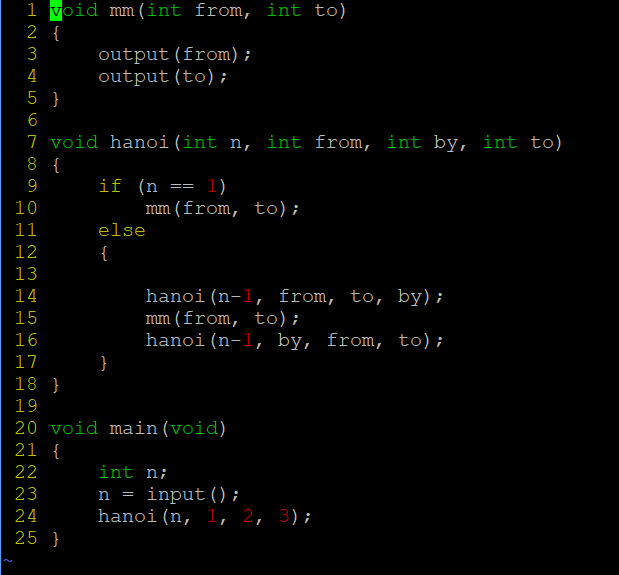
v0 register에 system call code번호 4를 대입하고 a0 register에 output\_prompt 주소를 대입해서 syscall을 호출하면, 문자열 출력에 대한 system call이 호출된다.

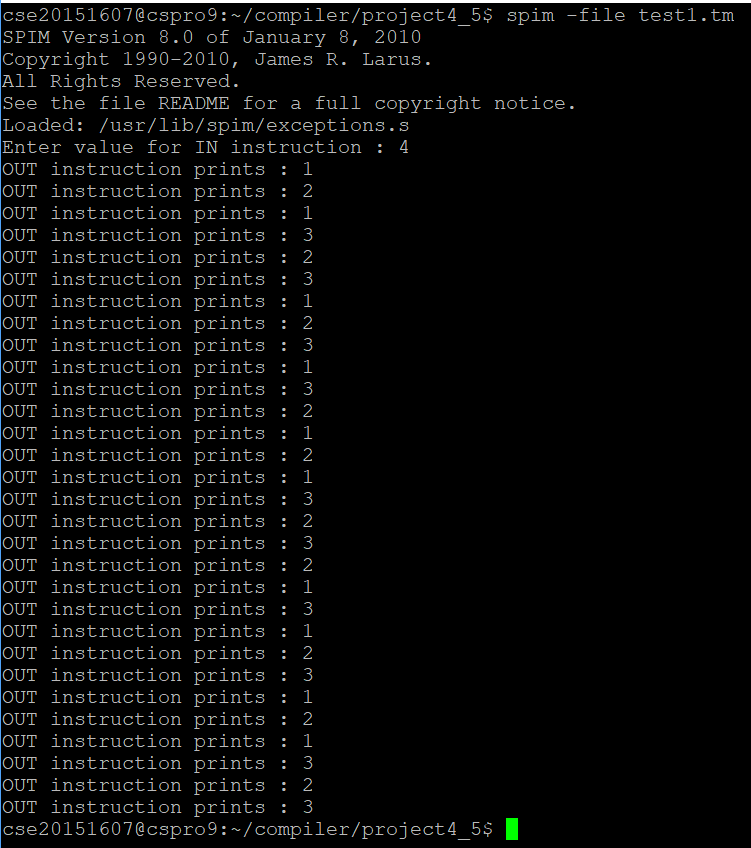
v0 register에 system call code번호 1를 대입하고 a0 register에 원하는 값을 대입하고 syscall을 호출하면, a0 register의 값을 출력하는 system call이 호출된다.

**3. 시험**

(test1.c) 하노이의 탑

하노이의 탑 n개를 입력받아 이를 재귀로 풀어 어디서 어디로 이동해야하는지 출력한다

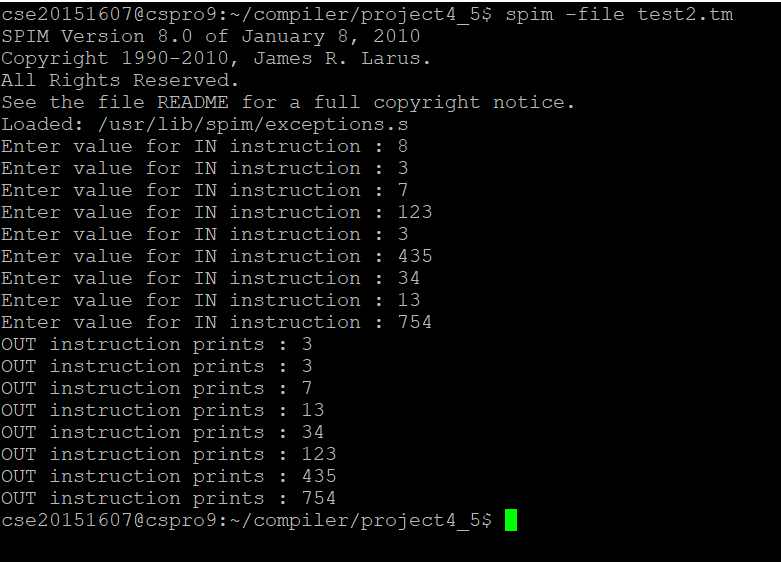




(test2.c) 이진 트리 삽입, Inorder 순회

N을 입력받아 총 N개의 postive value를 입력받는다. 이를 int array로 구현되어있는 tree에 insert하고, inorder 순회를 통해 print한다. 그러면 sorted된 상태여야한다.





**5. 기타**

1. 연구 조원 기여도

20141510 김민우 (50%)

20151607 정재훈 (50%)

2. 자체 평가

컴파일러 수업에서 배웠던 runtime environment를 완벽히 활용하고, 이를 MIPS instruction을 통해 자체적으로 구현하여 Compiler의 기본 개념을 총체적으로 이해하였다.

3. 느낀 점

Scanner, Parser, Semantic Analyzer, Code Generator까지 차근차근 진행해오면서 최종적으로 코드가 생성되는 것을 보니 매우 뿌듯하였다.