**Design and Development of Compiler**

**For C- Language**

**(Project 4 Design and Implementation of Code Generator**

**for TM simulator)**

**결과 보고서**

**과목명: [CSE4120] 기초 컴파일러 구성**

**담당교수: 서강대학교 컴퓨터공학과 정 성 원**

**개발자: 15조 (20141540 심규진, 20141602 황기덕)개발기간: 2019. 06. 21 ~ 2019. 06. 25**

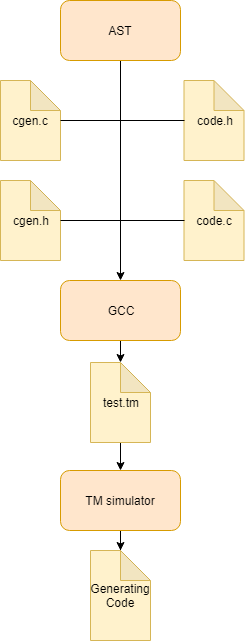
**프로젝트 제목 : Design and Development of Compiler for C-Language:**

**Phase 4: Design and Implementation of Code Generator**

**제출일 : 2019. 06. 25**

**조원 : 20141540 심규진, 20141602 황기덕**

**I. 개발 목표**

****

* 프로젝트 3에서 구현한 C Minus언어 Semantic Analyzer를 활용하여 SPIM machine(TM simulator 사용하였다.)에서 동작하는 C Minus언어용 code generator를 개발한다. Stack 기반으로 memory를 access하며 global 변수와 local 변수는 stack에 쌓는 방식이 다르므로 다르게 접근해야 한다. 또한, 해당 scope가 종료되었을 때 stack의 memory를 지워주어야 한다.

**II. 개발 범위 및 내용**

1. **개발 범위**

* 교재에 나와있는 appendix의 코드는 Tiny언어용이므로, C Minus 언어에 맞게 함수와 자료구조를 수정해주어야 한다.
* code.h, code.c, cgen.h, cgen.c를 C Minus 언어에 맞게 수정하여 TM simulator에서 구동 가능한 [test].tm code를 생성한다.

1. **개발 내용**

* symtab.h, symtab.c
  + st\_set\_emit\_loc : 함수의 emitloc을 설정하는 함수
  + st\_get\_emit\_loc : 함수의 emitloc을 가져오는 함수
  + st\_get\_target\_loc : 변수의 relative location을 가져오는 함수
  + st\_gen\_start : 변수의 location을 찾기 위해 초기화 작업을 진행하는 함수
  + st\_gen\_up, st\_gen\_up : 변수의 location을 찾기 위해 scope level을 이동하는 함수
* analyze.c
  + CallK일때, TM상에서 input, output함수를 처리할 수 있도록 수정한다.
  + Input은 값을 assign할 수 있도록 type을 integer로 설정한다.
* cgen.h, cgen.c
  + 해당 node마다 맞는 역할을 수행하여 C Minus 언어를 TM instruction으로 만들어준다.
    - emitRO (opcode r, s, t) : register only
    - emitRM (opcode r, d(s)) : register and memory, d는 offset
    - emitRM\_Abs(opcode r, a) : RM과 같지만 a는 absolute location
* code.h, code.c
  + #define ac2 2 // acc를 위한 register

#define fp 3 // frame pointer register

#define sp 4 // stack pointer register

* + cgen.c에서 코드를 생성할 때 도움을 주는 유틸리티 함수
  + 각 kind에 맞게 SPIM machine(TM simulator)에서 generate되는 instruction을 출력한다.

**III. 추진 일정 및 개발 방법**

1. **추진 일정**

-2019/06/21 : 명세서 확인 및 개발목표 수립

-2019/06/22 : Code generator 코드 작성

-2019/06/23 : 기존 코드와의 합성 및 TM code input, output 구현

-2019/06/24 : testing 및 결과 보고서 작성

-2019/06/25 : 프로젝트 제출

1. **개발 방법**

* 프로젝트3에서 제작한 Semantic Analyzer를 바탕으로 C Minus 언어를 위한 code generator를 작성한다.
* Semantic error check를 모두 통과하면 codeGen함수를 호출하여 code generation을 시작한다.
* cgen.c에서는 실질적으로 코드를 생성하고,

code.c에서는 코드 출력에 사용되는 fprintf을 수행한다 (utility 함수).

* code generator를 통해 .tm 파일을 생성하여 TM simulator로 test 한다.

|  |  |
| --- | --- |
| **Opcode Format : Opcode r, s, t** | |
| **HALT** | 프로그램 수행 중단 (operand 무시) |
| **IN** | reg[r] ← 정수 값을 standard input을 통해 받음 (s, t 무시) |
| **OUT** | reg[r] → standard output을 통해 출력 (s, t 무시) |
| **ADD** | reg[r] = reg[s] + reg[t] |
| **SUB** | reg[r] = reg[s] – reg[t] |
| **MUL** | reg[r] = reg[s] \* reg[t] |
| **DIV** | reg[r] = reg[s] / reg[t] (ZERO\_DIV 발생가능) |

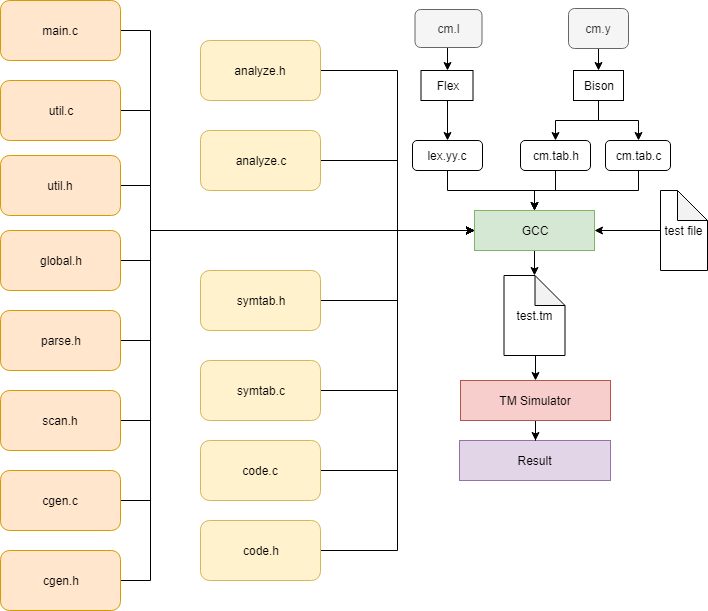
|  |  |
| --- | --- |
| **Opcode Format : Opcode r, d(s)** | |
| **LD** | reg[r] = dMem[a] (data memory a번지의 값을 r 레지스터에 저장) |
| **LDA** | reg[r] = a (a의 주소를 바로 r 레지스터에 저장) |
| **LDC** | reg[r] = d (d값을 바로 r 레지스터에 저장) |
| **ST** | dMem[a] = reg[r] (레지스터 r의 값을 메모리 a번지에 저장) |
| **JLT** | if ( reg[r] < 0 ) reg[PC\_REG] = a (만약 r레지스터의 값이 0보다 작으면 a번 명령행으로 이동한다. 이하 동일) |
| **JLE** | if ( reg[r] ≤ 0 ) reg[PC\_REG] = a |
| **JGE** | if ( reg[r] ≤ 0 ) reg[PC\_REG] = a |
| **JGT** | if ( reg[r] ≥ 0 ) reg[PC\_REG] = a |
| **JEQ** | if ( reg[r] == 0 ) reg[PC\_REG] = a |
| **JNE** | if ( reg[r] != 0 ) reg[PC\_REG] = a |

**IV. 연구 결과**

1. **합성 내용**

프로젝트3번 Symantic Analyzer로 만든 AST를 기반으로 code generator를 수행하여 TM assemble code를 생성하여 TM simulator로 실행한다.

**[Code Generator의 파일 구성도는 다음과 같다]**



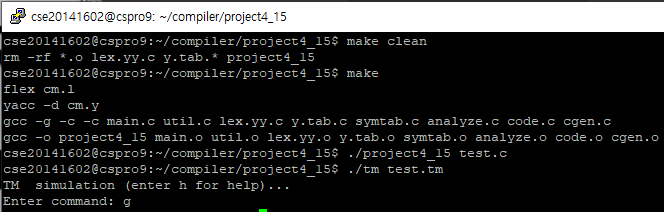
1. **분석 내용**

|  |  |  |
| --- | --- | --- |
| **Node Kind** | **Sub Kind** | **Action** |
| **genStmt** | **CompoundK** | 자식 노드에 대한 code를 generate한다. |
| **SelectionK** | If문인 경우, 자식 노드 2개(ELSE가 있을 경우 3개)에 대한 code를 generate한다.  이때, 조건식에 대한 code를 generate한 뒤, fjp L1과 같은 역할을 수행하게 하기 위하여 바로 다음 emit을 저장하여 skip한 후(emitskip(1)) if의 statement를 생성한다.  그 후 p3이 NULL이면(else문이 없다면) 현재 emit location을 저장한 후 condition 뒤 emit location을 불러와 그 위치에서 fjp를 시켜주고 emit location을 복귀시킨다.  p3이 NULL이 아니라면(else문이 있다면) ujp L2, lab L2와 같은 역할을 해 주어야 하기 때문에 또다시 emitSkip(1)을 해 주어 location을 저장시키고, if statement에 대한 코드를 만든 후 else statement를 만들고 저장시킨 location들을 불러와준다. |
| **ElseK** | Else문은 자식 노드에 대한 code를 generate한다. |
| **IterationK** | While문인 경우, 자식에 대한 code를 generate하기 전에 emitSkip(0)을 통하여 본인의 위치를 저장시키고 condition code를 generate 한다. 그 후 emitSkip(1)을 통하여 조건이 만족하지 않았을 때(while을 빠져나갈 때) 를 대비하여 location을 저장하고, Statement code를 generate한 다음 위에서 저장한 location을 알맞게 이용해 lab L1, lab L2, fjp L2, ujp L1등을 만든다. |
| **ReturnK** | 자식 노드에 대한 code를 generate한다. Return후에는 mian 함수에서의 호출 위치로 돌아가야 하므로 LD opcode를 사용하여 fp 바로 밑에 있는 return address 값을 pc에 넣어준다. |
| **genExp** | **ConstK** | 상수의 경우, ac에 값을 저장한다. |
| **IdK** | 변수의 경우, st\_get\_target\_loc 함수를 통해 해당 변수가 global 변수인지 확인하고 loc(offset)의 정보를 가져온다.  isAddr이 True일 경우 lvalue이고, isAddr이 False인 경우 rvalue이므로 각각의 경우에 주소를 저장할지, 값을 저장할지를 정해서 알맞게 처리해 준다. |
| **IdarrayK** | Array의 경우, store\_addr\_lev()로 현재 isAddr 상태를 저장하고 A[num] 형태이면 num 부분의 code를 generate해 준다. 그 후 restore\_addr\_lev()로 저장한 isAddr을 가져온 다음 st\_get\_target\_loc()함수를 통해 location과 global 변수인지에 대한 정보를 가져온다. 그 후 isAddr이 true이면, 즉 lvalue이면 ac에 주소값을 저장해주고 isAddr이 Falsue이면 ac에 그 변수의 값을 저장해준다. |
| **OpK** | Operator의 경우, 자식 노드 2개(lhs, rhs)값을 각각 ac, ac1에 저장한 후에, operation 또는 비교연산자를 수행한다. 이 때 op의 결과값이 false이면 ac에 0을 저장하고, ture이면 ac에 1을 저장해준다. |
| **AssignK** | 우선 store\_addr\_lev()를 통해 isAddr의 정보를 저장하고 isAddr을 False로 만든 후, p2, 즉 rvalue에 대한 code를 generate한다. p2의 대한 정보가 ac에 저장되어잇으므로 그것을 mp에 저장하고, isAddr을 True로 만든 후 p1, 즉 lvalue에 대한 code를 generate 한다.  mp에 임시 저장했던 rvalue값을 꺼내 온 후 ac의 주소에 rvalue를 넣은 후, restore\_addr\_lev()를 통해 isAddr 정보를 복구해준다. |
| **CallK** | CallK는 input인 경우, output인 경우, 그 외의 함수인 경우로 나누어서 처리한다. Input과 output은 이미 구현된 코드(IN, OUT)를 이용하여 처리하고, 그 외의 user defined function가 호출되었을 때 control link와 return address를 저장하고 호출한 user defined function으로 jump해주며, 호출된 위치로 돌아오면 fp와 sp를 restore 해주어 처리한다. |
| **genDec** | **IntK** | global변수 선언 시, stack에 공간을 할당해준다.  (sp를 한칸 내려준다.) |
| **LocalK** | 변수 선언 시, stack에 공간을 할당해준다. (sp를 한칸 내려준다.) |
| **ArrayK** | array 선언 시, array index만큼 stack에 공간을 할당해준다. (sp를 array index만큼 내려준다.) |
| **FunK** | 함수의 선언 시, 해당 함수의 이름으로 만들어지는 symbol table 노드의 emitloc 정보를 업데이트 한다.  Main function이며 user define function이 하나 이상 존재하는 경우 global 변수 저장 이후에 user define function에 대한 code가 쌓일 것이기 때문에 첫 번째 함수 code generate시 저장해 두었던 emit location에서 바로 현재 location으로 jump할 수 있게 만들어 준 후 함수 내부에 대한 code를 generate 해 준다.  Main function이며 user define function이 없을 경우엔 global 변수 선언 뒤 바로 main 함수가 나오기 때문에 위처럼 jump를 뛸 필요가 없으니 바로 함수 내부에 대한 code를 generate 해 준다.  Main function이 아니며 첫번째 user define function일 경우엔 현재 emit location을 저장하여 main function에서 jump할 수 있 해 준다.  그 후 function 내부에 대한 code를 generate 해 주고, code가 다 끝나면 return addr의 정보를 pc에 저장해 준다. |

1. **제작 내용**

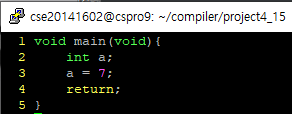
* **프로그램 수행**

완성된 프로그램으로 실행을 하면 (./project4\_15 test.c) test.tm파일이 생성되고, 이를 TM simulator로 수행한다.

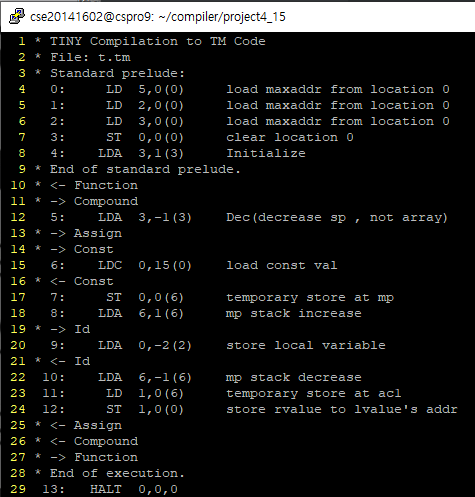


**<프로그램 실행 장면>**

* **TM code 생성**



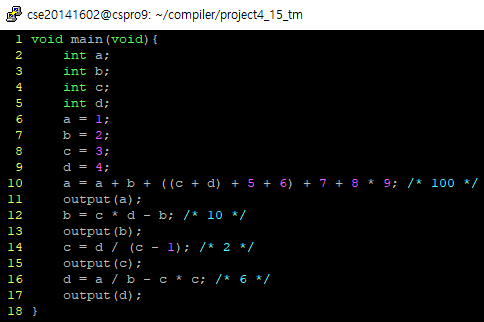
**<C minus 코드>**



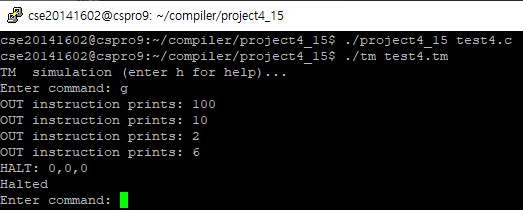
**<TM code 생성>**

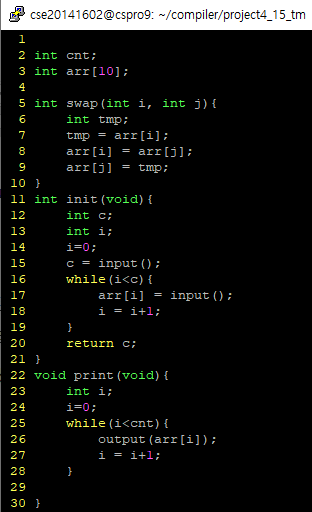
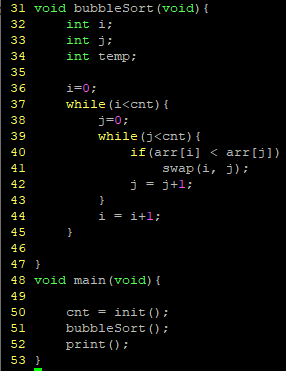
**4. 시험 내용**

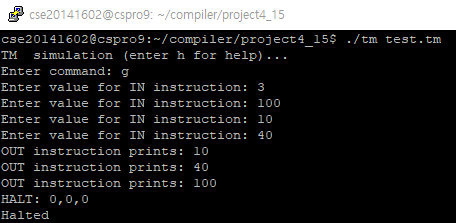
* test 파일에 대해서 생성된 TM code를 TM simulator로 실행하여 적절한 output을 갖는지 테스트해본다.
  + 출력 예시 프로그램 1 (조교 제공 코드1 [test4.c])
    - input : none
    - output : a, b, c, d의 결과값 (100, 10, 2, 6)



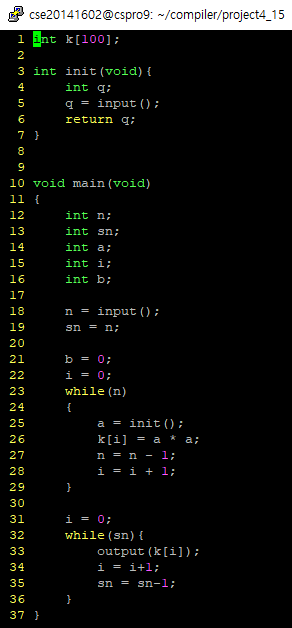
* + - 수행 결과



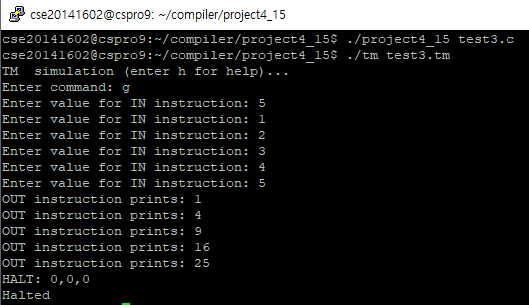
* + 출력 예시 프로그램 2 (조교 제공 코드2 [test.tm])
    - input : array index
    - output : bubble sort를 수행한 결과값  
       
    - 수행 결과



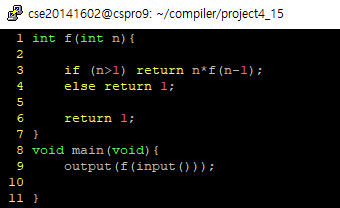
* + 출력 예시 프로그램 3 (직접 구현한 코드1 [test3.c])
    - input : 먼저 n을 입력받고, n개의 수를 입력받는다.
    - output : 입력받은 n개의 수를 제곱한다.



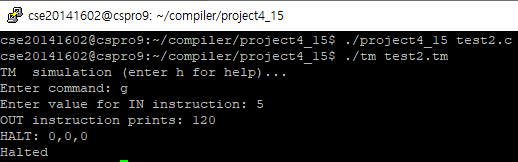
* + - 수행 결과



* + 출력 예시 프로그램 4 (직접 구현한 코드2 [test2.c])
    - input : n
    - output : n!



* + - 수행 결과



**5. 평가 내용**

* TM Code 생성 시 기존 Project3 과정에서 Validation check를 통하여 type check, scope check를 거쳐 테스트 파일에 대한 에러 여부를 검증하였고, 이를 TM code로 generation 시키기 때문에 안정성을 확보하였다.
* TM 머신 내에서 step by step으로 Debug를 수 차례 진행하여 단계별 TM Instruction에 대한 생성 및 Register 에 대한 Value값을 확인하였다. 정상적인 Case의 코드로 개인당 10회 이상의 다양한 테스트를 진행하였을 때 프로그램 실행 시 TM 프로그램 내에서 Memory overflow 등의 상황을 모두 검증하였으므로 안정성과 신뢰성을 확인하였다.

**V.연구 결과**

1. **역할 분담**

|  |  |  |
| --- | --- | --- |
|  | **심규진** | **황기덕** |
| **명세서 확인 및 개발 목표 수립** | 50% | 50% |
| **Code generator 코드 작성** | 20% | 80% |
| **기존 코드와의 합성 및 구현** | 50% | 50% |
| **코드 testing 및 보고서 작성** | 80% | 20% |

1. **자체 평가**

문제가 발생하면, 팀원들과 상의를 하며 각자 방법의 타당성을 서로 검증하며 가장 적합한 방법으로 문제를 해결하였다. TM 코드로 변한 후에 디버깅을 위해서 TM 코드로 나오는 부분에 대해서 공부한 내용을 모두 합하여 조원과 함께 수정이 필요한 부분을 수정하였다.

1. **느낀점**

처음 프로젝트4에 대한 공지를 하였을때 당연히 pcode를 구현할 것이라 생각했었는데 예상과 달리 three address code를 구현하는 프로젝트라 적잖이 당황했다. 그러나 4차 프로젝트를 진행하며 하나의 작은 컴파일러를 만드는 과정속에서 컴파일러의 작동 원리를 이해할 수 있었던 좋은 경험이었다.