编译原理调研报告
简单汇编编程及编译器功能设定

姓名：周辰霏
学号：1712991
专业：计算机科学与技术

摘要

本文参考 gcc 支持的 C 语言特性, 定义和使用上下文无关文法描述了一个简单的自定义编译器所需编译的 C 子集; 通过将简单 C 语言程序改写为汇编语言程序, 在调试和运行的过程中更深层次的体会编译器是如何解决“翻译”中的种种问题的, 并对编译器程序中所 需要的数据结构和算法设计进行调研, 更多地了解在编译器的实现过程中亟待解决和完善的问题。

关键字：上下文无关文法；编译器实现；gcc 编译器；汇编编程；C 语言

2019 年 9 月 29 日

目录

一、 自定义 C 编译器

（一） GCC 支持的语言特性

GCC 支持多种版本的 C 语言标准，默认使用 C11 加 GCC extension 的组合，支持的主要特性如下：

- **表达式**
  - 基本表达式: 由标识符、常量、字符串等直接构成的表达式
  - 前缀表达式: 无括号且将运算符写在操作数前面的与中缀表达式相对的表达式
  - 一元表达式: 包含一元运算符的表达式
  - 强制类型转换表达式: 将操作数转化为所需要的类型的表达式
  - 算术表达式: 包括乘法表达式、加法表达式、移位表达式、位操作表达式等
  - 关系表达式: 用关系运算符将两个表达式连接起来的表达式
  - 等号表达式: 用 == 或!= 将两个表达式连接起来的表达式
  - 逻辑表达式: 用逻辑运算符将关系表达式或逻辑量连接起来的表达式
  - 条件表达式: 由条件运算符构成的表达式, 并常用来构成赋值语句
  - 赋值表达式: 用来赋给某变量一个具体值的表达式
- **声明及外部定义** 用于向程序表明变量的类型和名称, 包括结构体、常量、变量、函数、指针等的声明初始化或外部定义
- **语句**
  - 标记语句: 在语句前有可引用的标识符
  - 复合语句: 把多个语句用括号括起来组成的一个语句称复合语句
  - 条件分支语句: 用来判断给定的条件是否满足, 并根据判断的结果决定执行的语句
  - 循环语句: 一组被重复执行的语句, 其是否继续重复执行取决于循环终止条件
  - 跳转语句: 从程序的一个位置跳转到另一个位置的语句
- **预处理指令** C 语言的预处理主要有: 宏定义、文件包含、条件编译

以上为常见的 C 语言特性，其它补充性不常用的特性则不一一列举。

（二） 自定义编译器支持的 C 语言子集

基于上述主要特性，初步自定义的编译器支持下述特性

- **表达式** 不包含前缀表达式
- **声明** 只包括变量和函数的声明
- **条件分支 & 循环结构 & 跳转语句**

（三） CFG描述的C语言子集

1. 表达式

- 基本表达式

$$primary\_expr \rightarrow id \mid const \mid string \mid (expr)$$

id表示标识符,const表示常量,string表示字符串,expr表示表达式

- 一元表达式

$$\begin{aligned} unary\_expr \rightarrow\ & ++ unary\_expr \mid --unary\_expr \mid unary\_operator\ cast\_expr \\ & \mid sizeof\ unary\_expr \mid sizeof(type\_name) \end{aligned}$$

$$unary\_operator \rightarrow \& \mid * \mid + \mid - \mid \sim \mid !$$

unary_expr表示一元表达式,cast_expr表示强制转换表达式,type_name表示类型名,unary_operator表示一元运算符

- 强制类型转换表达式

$$cast\_expr \rightarrow unary\_expr \mid (type\_name)\ cast\_expr$$

- 算术表达式
  - 乘法表达式

$$\begin{aligned} mul\_expr \rightarrow\ & cast\_expr \mid mul\_expr * cast\_expr \\ & \mid mul\_expr / cast\_expr \mid mul\_expr\ \%\ cast\_expr \end{aligned}$$

  - 加法表达式

$$add\_expr \rightarrow mul\_expr \mid add\_expr + mul\_expr \mid add\_expr - cast\_expr$$

  - 移位表达式

$$sh\_expr \rightarrow add\_expr \mid sh\_expr << add\_expr \mid sh\_expr >> add\_expr$$

- 关系表达式

$$\begin{aligned} rela\_expr \rightarrow\ & sh\_expr \mid rela\_expr < sh\_expr \mid rela\_expr > sh\_expr \\ & \mid rela\_expr <= sh\_expr \mid rela\_expr >= sh\_expr \end{aligned}$$

- 等号表达式

$$eq\_expr \rightarrow rela\_expr \mid eq\_expr == rela\_expr \mid eq\_expr\ != rela\_expr$$

- 逻辑表达式
  - 与表达式

$$and\_expr \rightarrow eq\_expr \mid and\_expr \& eq\_expr$$

- 异或表达式

$$xor\_expr \rightarrow and\_expr \mid xor\_expr \hat{} and\_expr$$

- 或表达式

$$or\_expr \rightarrow xor\_expr \mid or\_expr|xor\_expr$$

- 逻辑与表达式

$$land\_expr \rightarrow or\_expr \mid land\_expr \&\& or\_expr$$

- 逻辑或表达式

$$lor\_expr \rightarrow land\_expr \mid lor\_expr||land\_expr$$

- 条件表达式

$$con\_expr \rightarrow lor\_expr \mid lor\_expr?expr : con\_expr$$

- 赋值表达式

$$assign\_expr \rightarrow con\_expr \mid unary\_expr \ \ assign\_operator \ \ assign\_expr$$
$$assign\_operator \rightarrow \ = \mid *= \mid /= \mid += \mid -= \mid <<= \mid >>= \mid \&= \mid \hat{}= \mid |=$$
$$expr \rightarrow assign\_expr \mid expr, assign\_expr$$

2. 声明

- 变量声明

$$type \rightarrow void \mid int \mid float \mid double \mid char$$
$$idlist \rightarrow idlist, id \mid id$$
$$decl \rightarrow type \ \ idlist$$

id 表示标识符,idlist 标识符列表,type 变量类型,decl 声明语句

- 函数定义及声明

$$funcdef \rightarrow type \ funcname \ (paralist) \ stmt$$
$$paralist \rightarrow paralist, paradef \mid paradef \mid \epsilon$$
$$paradef \rightarrow type \ id$$

funcdef 表示函数声明语句,funcname 函数表,paralist 参数列表,stmt 语句,paradef 参数声明

3. 条件分支 & 循环结构 & 跳转语句

- 条件分支

$$label\_stmt \rightarrow id \ : \ stmt \mid case \ const \ : \ stmt \mid default \ : \ stmt$$
$$select\_stmt \rightarrow if \ (expr) \ stmt \ else \ stmt \mid if \ (expr) \ stmt \mid switch \ (expr) \ stmt$$

label_stmt 表示标记语句,select_stmt 表示分支语句

- 循环语句

$$iteration\_stmt \rightarrow for\ (expr; expr; expr)\ stmt \mid while\ (expr)\ stmt \mid do\ stmt\ while\ (expr)$$

- 跳转语句

$$jump\_stmt \rightarrow goto\ id; \mid continue; \mid break; \mid return\ \ expr;$$

综上全部即为自定义 C 语句子集的 CFG 表示

二、 通过等价汇编代码编写探究编译器的实现

(一) 阶乘程序

阶乘程序主要包含了自定义 C 语言子集中的赋值、算术、关系、等号表达式, 变量声明, 循环语句, 跳转语句

factorial

```
.486;表示使用486处理器
.model flat,stdcall
include \masm32\include\msvcrt.inc
includelib \masm32\lib\msvcrt.lib

.data
result dd 1    ;最终结果
datan dd ?;输入n
typen db '%d',0
data1 db 'Hello World',0dh,0ah,0   ;13,10表示换行,0表示终止
data2 db 'Press any key to continue...',0


.code
start:
    ;————————————————输入————————————————
    invoke crt_scanf,addr typen,addr datan ;输入n scanf("%d",&n)
    mov eax,result    ;eax=result
    mov ecx,2d        ;ecx计数寄存器ecx=2
    push eax
    ;————————————————循环————————————————
    @@:   ;发现一个很好用的跳转,@B寻找前一个最近的@@,@F寻找后一个最近的
    pop eax
    mul cx       ;最大计算到17
    inc ecx
    push eax
    cmp ecx,datan     ;datan-ecx 改变标志位,ZF=1则二者相等
    jle @B            ;≤时跳转到前一个最近的@@
    ;————————————————输出————————————————
    pop eax
```

```
        mov result,eax
        invoke crt_printf,addr typen,result
        invoke crt_printf, 0dh,0ah,0
        invoke crt_printf,addr data2
        ret
    end start
```

（二） 斐波那契数列

斐波那契数列程序主要包含了自定义 C 语言子集中的赋值、算术、关系、等号表达式, 变量声明, 循环语句, 跳转语句

fibonacci

```
.486;表示使用486处理器
.model flat,stdcall
include \masm32\include\msvcrt.inc
includelib \masm32\lib\msvcrt.lib

.data
    f0 dd 0
    f1 dd 1
    i dd 1;不知道为什么直接赋值ecx令其自增会报错,引入中间变量解决
    temp dd ?     ;上一个result
    typen db '%d',0
    datan dd ?
    space db ' ',0
    data1 db 'Press any key to continue...',0

.code
start:
    ;————————————————输入————————————————
    invoke crt_scanf,addr typen,addr datan;输入n
    invoke crt_printf,addr typen,f1   ;输出fib(1)
    invoke crt_printf,addr space
    mov ecx,i
    cmp ecx,datan
    jl @F       ;<向下跳转
    ;————————————————迭代————————————————
    @@:
    mov ebx,f1
    mov temp,ebx
    mov eax,f0
    add ebx,eax
    mov f1,ebx
    invoke crt_printf,addr typen,f1
    invoke crt_printf,addr space
    mov eax,temp
    mov f0,eax
```

```
        inc i                                ;inc ecx
        mov ecx,i
        cmp ecx,datan
        jl @B               ;< 向上跳转
        ;————————————————————防闪退————————————————————
        invoke crt_printf,0dh,0ah,0
        invoke crt_printf,addr data1
        ret
    end start
```

（三） 前 n 项和计算

计算公差为 1, 首项为 1 的等差数列的前 n 项和的程序, 它主要包含了自定义 C 语言子集中的赋值、算术、关系、等号表达式, 变量声明, 循环语句, 跳转语句, 函数声明及调用

sum

```
int sum(int n)
{
    int ans=0,i=0;
    do{
        ans+=i;
    }while(i<=n)
    return ans;
}
int main()
{
    int n;
    scanf("%d",&n);
    printf("%d\n",sum(n));
    return 0;
}
```

sum

```
    .486;表示使用486处理器
    .model flat,stdcall
    option casemap :none

    include \masm32\include\msvcrt.inc
    includelib \masm32\lib\msvcrt.lib

    .data
    result dd 0    ;最终结果
    datan dd ?;输入n
    typen db '%d',0
    i dd 1
    data1 db 'Hello World',0dh,0ah,0   ;13,10表示换行,0表示终止
    data2 db 'Press any key to continue...',0
```

```
15
16
17      .code
18      sum:
19          add eax,i
20          inc i
21          cmp i,ecx
22          jle sum
23          ret
24      start:
25          ;————————————————————————输入————————————————————————
26          invoke crt_scanf,addr typen,addr datan ;输入n scanf("%d",n)
27          mov eax,0    ;eax=result
28          mov ecx,datan
29          ;————————————————————————调用————————————————————————
30          call sum
31          ;————————————————————————输出————————————————————————
32          mov result,eax
33          invoke crt_printf,addr typen,result
34          invoke crt_printf, 0dh,0ah,0
35          invoke crt_printf,addr data2
36          ret
37      end start
```

（四） 条件判断位运算

一个简单的条件判断并执行位运算操作的程序, 它主要包含了自定义 C 语言子集中的赋值、算术、关系、等号表达式, 变量声明, 条件判断语句, 循环语句, 跳转语句

ifbit

```
1       int main()
2       {
3           int datan;
4           printf("1.and2.or3.not4.xor5.shl6.shr\n");
5           scanf("%d",&datan);
6           if(datan==1) printf("%d",99&59);
7           else if(datan==2) printf("%d",99|59);
8           else if(datan==3) printf("%d",99|59);
9           else if(datan==4) printf("%d",99^59);
10          else if(datan==5) printf("%d",99<<1);
11          else if(datan==6) printf("%d",99>>1);
12          else printf("error!");
13          return 0;
14      }
```

ifbit

```
1       .486;表示使用486处理器
```

```
.model flat,stdcall
include \masm32\include\msvcrt.inc
includelib \masm32\lib\msvcrt.lib

.data
typen db '%d',0
datan dd ?
result dd ?
data1 db '1.and2.or3.not4.xor5.shl6.shr',0dh,0ah,0
data2 db 'Press any key to continue...',0
data3 db 'error!'

.code
start:
;————————————————————输入————————————————————
invoke crt_printf,addr data1
invoke crt_scanf,addr typen,addr datan
mov eax,datan
;————————————————————判断————————————————————
cmp eax,1d
je iand
cmp eax,2d
je ior
cmp eax,3d
je inot
cmp eax,4d
je ixor
cmp eax,5d
je ishl
cmp eax,6d
je ishr
jmp error
;————————————————————条件————————————————————
iand:
mov ebx,01100011b
and ebx,00111011b
mov result,ebx
jmp output
ior:
mov ebx,01100011b
or ebx,00111011b
mov result,ebx
jmp output
inot:
mov ebx,01100011b
not ebx
mov result,ebx
jmp output
```

```
    ixor:
    mov ebx,01100011b
    xor ebx,00111011b
    mov result,ebx
    jmp output
    ishl:
    mov ebx,01100011b
    shl ebx,1
    mov result,ebx
    jmp output
    ishr:
    mov ebx,01100011b
    shr ebx,1
    mov result,ebx
    jmp output
    ;————————————————————————输出————————————————————————
    output:
    invoke crt_printf,addr typen,result
    invoke crt_printf,0dh,0ah,0
    invoke crt_printf,addr data2
    ret
    error:
    invoke crt_printf,addr data3
    invoke crt_printf,addr data2
    ret
    end start
```

（五） 总述——编译器如何实现将 C 转换为汇编程序

由上述三个简单程序可基本涵盖除强制转换和标记语句外的自定义编译器功能, 但是这是人的“手工编译”, 所以与编译器的转换还是有些微的不同。从人的角度来说, 只要按照 C 语言的思想将之“翻译”为汇编语言即可, 其中的重点在于寄存器堆的合理利用以及语句之间逻辑关系的梳理。故而其实人“手工编译”的汇编程序与真正编译器“翻译地并不尽相同, 原因在于:

1. 有时为了更好更快更直接的书写汇编代码, 主体程序会被扩写或缩写——如在斐波那契数列程序中 C 语言使用的是简洁的递归形式, 而汇编代码则是选择了拆开的迭代形式;
2. 有时为了节省寄存器, 可能会重复使用同一个寄存器, 另一方面由于立即数之间不能直接相加减, 所以需要先将其中一个立即数存入寄存器, 这就增加了汇编程序的代码量

编译器不能只会翻译一个源程序, 而是要有能力将所有合法的 C 程序转换为汇编程序, 而如果通过编译器来实现这一过程, 不可能让编译器像人一样针对 每一个不同的程序有着不同的想法, 即穷举所有的 C 程序是不可能的也不现实的, 那么要如何去做呢? 基本方向是将 C 程序中包含的每个语言特性正确翻译! 即编译过程中的语法分析、语义分析到代码生成部分。但是这时仍然有一个问题——每个语言特性仍然有无穷多种合法的呈现 (a=1，b=2.0，...), 可行的解决方法是将同一类型的语言特性一般化符号化。

总结下来, 整个编译过程应该包括:

- **词法分析:** 将字符序列转换为 token 的过程

- **语法分析:** 利用 token 构造语法树
- **语义分析:** 进行类型检查; 借助符号表检查语法树的语义是否与既定的语言特性一致
- **中间代码生成:** 生成介于源程序语言和机器瞬间之间的一种结构简单、含义明确的记号系统
- **代码优化:** 通过优化步骤试图改进中间代码, 以便生成更好的目标代码
- **代码生成:** 将优化后的中间代码映射到目标语言

所使用到的数据结构包括:

- **记号 (token):** 用以分类表示字符
- **语法树:** 它的构造通常为基于指针的标准结构, 在进行分析时动态分配该结构, 整棵树可作为一个指向根节点的单个变量保存, 结构中的每一个节点都是一个记录，它的域表示由分析程序和之后的语义分析程序收集的信息。[1]
- **符号表:** 该数据结构中的信息与函数、变量、常量以及数据类型这些标识符有关。符号表几乎在编译的每一个阶段都要出现
- **常数表:** 用于存放程序中使用到的常量和字符串, 常数表对于缩小程序在存储器中的大小显得非常重要 [1]
- **中间代码:** 该代码可以是文本串的数组、临时文本文件或是结构的连接列表
- **临时文件:** 用来保存翻译时中间步骤的结果

所需要的算法设计有:

- **词法分析算法:** 词法分析过程需要 Thompson 算法 (从正则表达式到非确定有限状态有限机)、子集构造算法 (从非确定有限状态有限机到确定有限状态有限机) 以及 Hopcroft 最小化算法最终生成传给语法分析器的代码
- **递归下降分析算法:** 是一种自顶向下的语法分析方法, 它使用一组递归过程来处理输入。文法的每一个非终结符都有一个相关联的过程, 可分为预测分析和回溯分析两类 [2]

根据上述基本编译流程以及数据结构和算法设计可以将编译过程描述为以下几种程序的顺序工作:

1. **扫描程序:** 编译器阅读源程序 (字符流形式), 扫描程序执行的是词法分析部分——即将字符序列收集到 token 中, 该过程可类比拼写分类汇总, 该过程使用各词法分析算法, 使用符号表常熟文字表, 生成记号
2. **语法分析程序:** 语法分析程序实现定义程序结构的语法分析, 多用到自顶向下的递归下降分析算法, 用到扫描程序给出的记号、符号表等, 结果表示为语法树
3. **语义分析程序:** 语义分析程序的作用是分析每个特性所定义的语义——称之为“静态语义”, 它确定程序的运行。一般的程序设计语言的典型静态语义包括声明和类型检查。[1] 由语义分析程序计算的额外信息, 如数据类型则被称为属性, 它们通常存储于语法树和符号表
4. **源代码优化程序:** 最早的优化步骤在语义分析之后开始
5. **代码生成器:** 代码生成器得到中间代码 (IR)), 并生成目标机器代码
6. **目标代码优化程序:** 这一优化包括选择编址模式以提高性能、将速度慢的指令更换成速度快的、删除多余无用的操作

三、 结论

对自定义 C 语言编译器的设定和上下文无关文法描述, 更直接地感受一种语言的各种特性应该怎样被叙述和怎样被实现, 加之对简单 C 代码的汇编语言改写 可以更好的体会编译器所完成的工作, 在此基础上对编译器的各道工序研究, 了解编译器所运用到的基本数据结构和算法, 更深的体会到了编译器需要解决的问题 ——如何将各种特性归纳成各自的符号、如何翻译和检查这些符号、如何存储和翻译源代码等等。

参考文献

[1] 编译器实现 (一)[EB/OL].https://www.cnblogs.com/x-police/p/10859240.html

[2] Alfred V.Aho,Monica S.Lam. 编译原理 (第二版)[M]. 北京: 机械工业出版社,2009.39.