CASL 汇编器的设计与实现

刘福江，柴树杉

（中国地质大学 信息工程学院，武汉 430074）

摘　要：介绍了一种教学实验系统 CASL 汇编器的设计过程。使用了一种改进了的两次扫描算法实现 CASL 汇编器。实验表明，对两遍扫描算法的改进确实可行，CASL 汇编器的工作状态良好。

关键字：CASL；汇编器；扫描算法；向前引用

实用的汇编语言虽然功能非常强大，但是结构也很复杂，不利于初学者学习。由国家软件考试中心定义用于学习的CASL 汇编语言功能单一，程序格式规范，具有汇编语言的重要特征，是初学者学习汇编语言的最佳选择。作为一种教学实验系统，实现CASL汇编器具有很好的教学意义：首先，使用CASL汇编器可以方便学生学习CASL语言；其次，实现CASL汇编器可以帮助学生了解汇编器的工作原理[1,2]。

1 汇编器功能简介

汇编语言是一种面向机器、低级的计算机语言。汇编器的主要任务是将汇编语言的程序源程序翻译为功能等价的特定机器的目标程序[2]。

1.1 实用汇编器的功能

实用的汇编语言翻译器除了将汇编源程序翻译为机器的语言外，还依据汇编语言不同的特征提供其他的功能。实用汇编器的一般工作包括 :处理伪命令 ;定义和替换宏；进行错误检查；生成机器代码[2]。

1.2 CASL 汇编器的功能

CASL汇编器运行在普通的PC机上，产生COMET虚拟机的目标代码（CASL汇编语言是建立在COMET机器上的虚拟语言），是一种交叉汇编的汇编器。因为CASL语言功能单一的特点，CASL汇编器的工作也相对较少[4,5]。

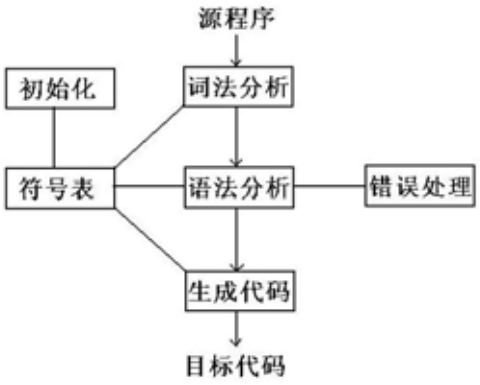
CASL 汇编器的主要工作有：

1） 处理 DC、DS 伪命令定义的变量
2） 将定义的标号替换为机器地址
3） 替换 IN、OUT、EXIT 三个宏
4） 对 CASL 源程序进行错误检查
5） 生成 COMET 虚拟机的目标代码

2 CASL 汇编器设计

2.1 功能模块

根据Alfred.V.Aho所描述的编译器模块结构，我们将CASL汇编器设计为类似的几个功能模块：词法分析器；语法分析器；代码生成器；符号表；错误处理。各个功能模块之间的相互关系如图 1 所示[3,5]。



2.1.1 词法分析器 词法分析器将 CASL 源程序解释成一个个独立的记号，然后将记号的类型以及记号所对应的值返回给语法分析器。词法分析器一般不产生错误信息。

2.1.2 语法分析器 语法分析器依据 CASL 语言的相关规则，对由词法分析器返回的记号序列进行分析。在语法分析的过程中，一般会生成相应的符号表，同时调用代码生成器生成目标代码。如果在语法分析过程中遇到任何错误语句，则调用错误处理模块。

2.1.3 代码生成器 代码生成器负责生成每个机器指令和最终目标代码的存储工作。在 CASL 源程序太长，或者指令参数使用错误时，代码生成器将发生错误。

2.1.4 符号表 符号表是一个辅助功能模块，

主要用于处理标号信息。该模块提供必要的符号操作：定义符号，查询符号等。

2.1.5 错误处理模块 错误处理模块输出相关的错误信息和发生错误的位置，然后退出程序。错误处理模块不会分析产生错误原因，也不会修复任何错误。

2.2 数据结构

CASL汇编器采用了改进了的两次扫描算法，使用的数据和普通两次扫描算法相比要也有一些区别。CASL汇编器主要的数据结构包括：地址计数器LC，符号表，未知标号表[1,2,5]。

2.2.1 地址计数器LC 地址计数器LC用于计算每个标号对应的地址和生成目标代码的存储位置[2,4]。

2.2.3 符号表 CASL 汇编器采用散列表来组织符号信息，同时使用链表来处理散列冲突的情况。

2.2.3 未知标号表 未知标号表记录在第一次扫描中遇到的向前引用标号的信息，然后在第二次扫描中用于修正目标代码。CASL汇编器中未知标号结构定义如下：

```
static struct { int i, max;
        struct { struct id *p;
                int off; } *stack;
} unknow_id;
```

其中 i 为表中标号数目，max 为表的容量，p 指向符号表中符号的结构，off 为标号对应目标代码的存放位置。

3 核心算法简介

3.1 传统的两遍汇编算法

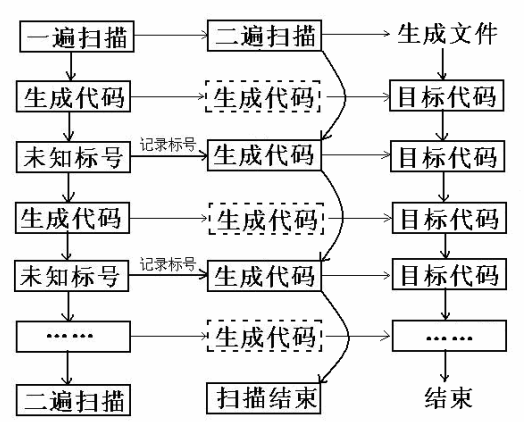
传统的汇编器采用两次汇编来处理标号向前引用问题。在第一遍扫描先确定所有的标号的地址，并保存在符号表中。第二遍扫描依据第一遍扫描所生成的符号表来汇编每一条汇编语句[2,4]。

由于在一遍扫描中并没有完成足够两的工作，因此传统的两遍汇编无法达到最佳的扫描效果。

3.2 改进的两遍扫描算法

根据多遍操作的原理可知，之前的遍应当尽量告诉稍后的遍做什么，而不是简单地向它提供简单的事实[1]。

CASL汇编器对算法改进的基本思路是，如果可能就在第一遍扫描时完成所有的工作，从而达到简化稍后遍的操作。CASL汇编器扫描算法的具体流程如图 2 所示[6]。



一遍扫描先生成绝大部分目标代码，二遍扫描再生成剩下小部分的目标代码，最后得到完整的目标代码。

3.3 两种算法的比较

为了量化两种算法的时间，我们定性假设扫描每一条指令所花时间为常数 t。汇编源程序的语句总数为 n，有标号向前引用的语句数为 x。

扫描整个源程序，传统的两遍扫描算法所花时间 T1 = (n + n) * t；改进后的算法所花时间 T2 = (n + x) * t。两种算法所花时间差 T2 – T1 = (n+n)*t - (n+x)*t = (n-x)*t。

根据经验可知，向前引用标号的数量一般远小于汇编程序的语句总数，即 x << n。因此，改进后的扫描算法比传统的的两遍扫描算法的效率要高出许多。

4 CASL 汇编器的代码实现

4.1 第一遍扫描的代码实现

```
while(lookahead != END
    && lookahead != EOF) {
    if(lookahead == '\n') {
        /* 如果是空行则继续循环 */
        match('\n'); continue;
    }
    if(lookahead == LABEL) {
        /* 有标号则定义并跳过标号 */
        def_id(lexbuf, addr);
        match(LABEL);
```

```
    }
    /* 汇编当前指令行，跳过'\n' */
    emitter(); match('\n');
}
```

源程序未结束时执行一个大的循环，每次循环判断并汇编每一个语句。lookahead 是记录当前记号的类型；match 函数匹配并跳过当前记号；def_id 记录标号；emitter 生成目标代码。

在 emitter 生成目标代码时，如果遇到未定义的标号，由 get_id 负责记录该标号的相关信息（保存在 unknow_id 体结构中），以便在第二次扫描时使用。

4.2 第二遍扫描的代码实现

```
/* 扫描未知标号 */
void patch_id(short code[])
{
    int i;
    for(i = 0; i < unknow_id.i; ++i) {
        int addr =
            unknow_id.stack[i].p->addr;
        int off  = unknow_id.stack[i].off;
        if(addr == -1) error("unknow id");
        code[off] = addr;
    }
}
```

第二遍扫描的对象是第一遍扫描中记录的未知标号包，在此对应为 unknow_id 结构。patch_id 遍历所有的未知标号，直到生成完整的目标代码。

如果标号的地址 p->addr 为-1（标号地址不可能为负数），则发生标号未定义错误。

5 结语

实践证明，实现CASL语言汇编器，可以更方便学习CASL汇编语言，同时也有助于了解汇编器的工作原理[6]。

参考文献

[1] Donald E.Knuth 著。苏运霖 译。《计算机程序设计艺术》第 1 卷 189-190，219-220

[2] Andrew S. Tanenbaum 著。 刘卫东 等译。 结构化计算机组成 第 7 章 机械工业出版社 2001

[3] Alfred.V.Aho Ravi Seth Jeffrey D.Ullman 著。Charpter 2 A simple one-pass compiler

[4] 杨光 冉峰 单片机汇编器的设计与实现 微计算机应用 2005-3。

[5] 程林辉 张凯 基于 SAAM 方法的 CASL 编译程序软件体系结构分析 中南民族大学学报(自然科学版) 第 23 卷 第 4 期 2004 年 12 月

[6] http://www.comet.ful.cn