目标代码生成

杨科迪 费迪

2022 年 11 月 22 日

目录

1 实验描述

经过一个学期的学习，编译器的构造到了令人激动的最终阶段，这将是这学期编译原理课程最具成就感的一步，实现编译器的程序员就可以称为一个“浪漫”的程序员了。在本次实验中，同学们需要将中间代码转化为目标代码，最后运行你生成的目标代码，检测其正确性。

2 实验要求

1. 完成目标代码生成工作，输出目标代码，并在希冀平台上提交进行测试。
2. 本次需要撰写完整研究报告，研究报告内容是你构建编译器，从词法分析模块开始，一直到目标代码生成的全过程。
3. 在雨课堂的提交栏中提交本次实验的 gitlab 链接，并在附件栏中提交研究报告。
4. 上机课时，以小组为单位，向助教讲解程序。

3 实验效果

以如下 SysY 语言为例：

```
int main()
{
    int a;
    int b;
    int min;
    a = 1 + 2 + 3;
    b = 2 + 3 + 4;
    if (a < b)
        min = a;
    else
        min = b;
    return min;
}
```

需要提示一点，为避免提供给同学们过多的提示，提供的代码框架并不能对 example.sy 文件直接生成指导书中这样完整的汇编代码，需要大家自行完善。

执行 make run 命令会生成对应的目标代码：

```
    .arch armv8-a
    .arch_extension crc
    .arm
    .global main
    .type main , %function
```

```
main:
.L17:
    push {fp}
    mov fp, sp
    sub sp, sp, #12
    ldr v26, =1
    add v4, v26, #2
    add v5, v4, #3
    str v5, [fp, #-12]
    ldr v27, =2
    add v7, v27, #3
    add v8, v7, #4
    str v8, [fp, #-8]
    ldr v9, [fp, #-12]
    ldr v10, [fp, #-8]
    cmp v9, v10
    blt .L21
    b .L24
.L21:
    ldr v13, [fp, #-12]
    str v13, [fp, #-4]
    b .L23
.L22:
    ldr v15, [fp, #-8]
    str v15, [fp, #-4]
    b .L23
.L23:
    ldr v16, [fp, #-4]
    mov r0, v16
    add sp, sp, #12
    pop {fp}
    bx lr
.L24:
    b .L22
```

进行寄存器分配之后

```
    .arch armv8-a
    .arch_extension crc
    .arm
    .global main
    .type main , %function
```

```
main:
.L17:
    push {r4, r5, fp}
    mov fp, sp
    sub sp, sp, #12
    ldr r4, =1
    add r4, r4, #2
    add r4, r4, #3
    str r4, [fp, #-12]
    ldr r4, =2
    add r4, r4, #3
    add r4, r4, #4
    str r4, [fp, #-8]
    ldr r4, [fp, #-12]
    ldr r5, [fp, #-8]
    cmp r4, r5
    blt .L21
    b .L24
.L21:
    ldr r5, [fp, #-12]
    str r5, [fp, #-4]
    b .L23
.L22:
    ldr r5, [fp, #-8]
    str r5, [fp, #-4]
    b .L23
.L23:
    ldr r5, [fp, #-4]
    mov r0, r5
    add sp, sp, #12
    pop {r4, r5, fp}
    bx lr
.L24:
    b .L22
```

4 实验流程

4.1 代码框架

本次实验框架代码的目录结构如下：

```
./
├── include
│   ├── Ast.h
│   ├── SymbolTable.h
│   ├── Type.h
│   ├── IRBuilder.h
│   ├── Unit.h
│   ├── Function.h
│   ├── BasicBlock.h
│   ├── Instruction.h
│   ├── Operand.h
│   ├── AsmBuilder.h.....................................汇编代码构造辅助类
│   ├── MachineCode.h....................................汇编代码构造相关类
│   ├── LinearScan.h.....................................线性扫描寄存器分配相关类
│   └── LiveVariableAnalysis.h...........................活跃变量分析相关类
├── src
│   ├── Ast.cpp
│   ├── BasicBlock.cpp
│   ├── Function.cpp
│   ├── Instruction.cpp
│   ├── lexer.l
│   ├── main.cpp
│   ├── Operand.cpp
│   ├── parser.y
│   ├── SymbolTable.cpp
│   ├── Type.cpp
│   ├── Unit.cpp
│   ├── MachineCode.cpp
│   ├── LinearScan.cpp
│   └── LiveVariableAnalysis.cpp
├── sysyruntimelibrary
├── test
├── .gitignore
├── example.sy
└── Makefile
```

- AsmBuilder.h 中为汇编代码构造辅助类。其主要作用就是在中间代码向目标代码进行自顶向下的转换时，记录当前正在进行转换操作的对象，以进行函数、基本块及指令的插入。

- MachineCode.h 中为汇编代码构造相关的框架，大体的结构和中间代码是类似的，只有具体到汇编指令和对应操作数时有不同之处。其框架大致如下
  - MachineUnit
  - MachineFunction
  - MachineBlock
  - MachineInstruction

    LoadMInstruction 从内存地址中加载值到寄存器中。

    StoreMInstruction 将值存储到内存地址中。

    BinaryMInstruction 二元运算指令，包含一个目的操作数和两个源操作数。

    CmpMInstruction 关系运算指令。

    MovMInstruction 将源操作数的值赋值给目的操作数

    BranchMInstruction 跳转指令。

    StackMInstruction 寄存器压栈、弹栈指令。
  - MachineOperand

    IMM 立即数。

    VREG 虚拟寄存器。在进行目标代码转换时，我们首先假设有无穷多个寄存器，每个临时变量都会得到一个虚拟寄存器。

    REG 物理寄存器。在进行了寄存器分配之后，每个虚拟寄存器都会分配得到一个物理寄存器。

    LABEL 地址标签，主要为 BranchMInstruction 及 LoadMInstruction 的操作数。
- LiveVariableAnalysis.h 为活跃变量分析，用于寄存器分配过程。
- LinearScan.h 为线性扫描寄存器分配算法相关类，为虚拟寄存器分配物理寄存器。

4.2 生成汇编代码

在中间代码生成之后，大家就可以对中间代码进行自顶向下的遍历，从而生成目标代码。整个目标代码的框架和中间代码的框架是比较类似的，只有在指令和操作数的设计上有所不同。

在代码框架中，我们已经给出了较为完整的代码示例，同学们只需要完成一些基础表达式的翻译，再将汇编代码打印出来即可。

4.2.1 访存指令

对于访存指令，框架代码中已经给出了对于 LoadInstruction 的翻译，具体可以分为以下三种情况：

1. 加载一个全局变量或者常量

   对于这种情况，同学们需要生成两条加载指令，首先在全局的地址标签中将其地址加载到寄存器中，之后再从该地址中加载出其实际的值。

2. 加载一个栈中的临时变量

   由于在 AllocaInstruction 指令中，已经为所有的局部变量申请了栈内空间，并将其相对 FP 寄存器的栈内偏移存在了符号表中，同学们只需要以 FP 为基址寄存器，根据其栈内偏移生成一条加载指令即可。

3. 加载一个数组元素

   数组元素的地址存放在一个临时变量中，只需生成一条加载指令即可。

大家只需要模仿完成 StoreInstruction 的翻译即可。

4.2.2 内存分配指令

框架代码中已经完成了对于 AllocaInstruction 的翻译，具体思路就是为指令的目的操作数在栈内分配空间，将其相对于 FP 寄存器的偏移存在符号表中。

4.2.3 二元运算指令

对 BinaryInstruction 的翻译应该是最简单的，框架代码里已经给出了加法运算的翻译，同学们只需要仿照即可。需要注意的一点是，中间代码中二元运算指令的两个操作数都可以是立即数，但这一点在汇编指令中是不被允许的。对这种情况，同学们需要根据汇编指令的规则，提前插入 LOAD 汇编指令，来将其中的某个操作数加载到寄存器中。需要注意的是，当第二个源操作数是立即数时，其数值范围有一定限制。

4.2.4 控制流指令

1. UncondBrInstruction

   对于 UncondBrInstruction，同学们只需要生成一条无条件跳转指令即可，至于跳转目的操作数的生成，大家只需要调用 genMachineLabel() 函数即可，参数为目的基本块号；

2. CondBrInstruction

对于 CondBrInstruction，首先明确在中间代码中该指令一定位于 CmpInstruction 之后，对 CmpInstruction 的翻译比较简单，相信大家都能完成。对 CondBrInstruction，同学们首先需要在 AsmBuilder 中添加成员以记录前一条 CmpInstruction 的条件码，从而在遇到 CondBrInstruction 时生成对应的条件跳转指令跳转到 True Branch，之后需要生成一条无条件跳转指令跳转到 False Branch。

3. RetInstruction

对于 RetInstruction 同学们需要考虑的情况比较多。首先，当函数有返回值时，我们需要生成 MOV 指令，将返回值保存在 R0 寄存器中；其次，我们需要生成 ADD 指令来恢复栈帧，（如果该函数有 Callee saved 寄存器，我们还需要生成 POP 指令恢复这些寄存器），生成 POP 指令恢复 FP 寄存器；最后再生成跳转指令来返回到 Caller。

4.2.5 函数定义

在目标代码中，在函数开头需要进行一些准备工作。首先需要生成 PUSH 指令保存 FP 寄存器及一些 Callee Saved 寄存器，之后生成 MOV 指令令 FP 寄存器指向新的栈底，之后需要生成 SUB 指令为局部变量分配栈内空间。分配栈空间时需要注意，一定要在完成寄存器分配后再确定实际的函数栈空间，因为有可能会有某些虚拟寄存器被溢出到栈中。一种思路是不在目标代码生成时插入 SUB 指令，而是在后续调用 output() 函数打印目标代码时直接将该条指令打印出来，因为在打印时已经可以获取到实际的栈内空间大小；另一种思路是先记录操作数还没有确定的指令，在指令的操作数确定后进行设置[1]。至此，就可以继续转换函数中对应的其他指令了。

4.2.6 函数调用指令

在进行函数调用时，对于含参函数，需要使用 R0-R3 寄存器传递参数，如果参数个数大于四个还需要生成 PUSH 指令来传递参数；之后生成跳转指令来进入 Callee 函数；在此之后，需要进行现场恢复的工作，如果之前通过压栈的方式传递了参数，需要恢复 SP 寄存器；最后，如果函数执行结果被用到，还需要保存 R0 寄存器中的返回值。

4.3 实现寄存器分配

在本次实验中我们需要完成线性扫描寄存器分配算法 [1]，单趟遍历每个活跃区间 (Interval)，为其分配物理寄存器。其具体分为以下三个步骤。

4.3.1 活跃区间分析

在前一步的目标代码生成过程中，已经为所有临时变量分配了一个虚拟寄存器 VREG。在这一步需要为每个 VREG 计算活跃区间。这一步在课程中应该已经有所讲解，具体算法可参照龙书第二版 P391。在框架代码中，我们已经完成了这一步，同学们只需要使用其分析结果即可。

4.3.2 寄存器分配

在一步我们涉及到了两个集合：

[1] 翻译 RetInstruction 时，也可以采取相同的思路

intervals 表示还未分配寄存器的活跃区间，其中所有的 interval 都按照开始位置进行递增排序；active 表示当前正在占用物理寄存器的活跃区间集合，其中所有的 interval 都按照结束位置进行递增排序。

现在开始遍历 intervals 列表进行寄存器分配，这一步需要大家完善 linearScanRegisterAllocation() 函数。对任意一个 unhandled interval 都进行如下的处理：

1. 遍历 active 列表，看该列表中是否存在结束时间早于 unhandled interval 的 interval（即与当前 unhandled interval 的活跃区间不冲突），若有，则说明此时为其分配的物理寄存器可以回收，可以用于后续的分配，需要将其在 active 列表删除；

2. 判断 active 列表中 interval 的数目和可用的物理寄存器数目是否相等，

   (a) 若相等，则说明当前所有物理寄存器都被占用，需要进行寄存器溢出操作。具体为在 active 列表中最后一个 interval 和当前 unhandled interval 中选择一个 interval 将其溢出到栈中，选择策略就是看谁的活跃区间结束时间更晚，如果是 unhandled interval 的结束时间更晚，只需要置位其 spill 标志位即可，如果是 active 列表中的 interval 结束时间更晚，需要置位其 spill 标志位，并将其占用的寄存器分配给 unhandled interval，再按照 unhandled interval 活跃区间结束位置，将其插入到 active 列表中。

   (b) 若不相等，则说明当前有可用于分配的物理寄存器，为 unhandled interval 分配物理寄存器之后，再按照活跃区间结束位置，将其插入到 active 列表中即可。

4.3.3 生成溢出代码

在上一步寄存器分配结束之后，如果没有临时变量被溢出到栈内，那寄存器分配的工作就结束了，所有的临时变量都被存在了寄存器中；若有，就需要在操作该临时变量时插入对应的 LoadMInstruction 和 StoreMInstruction，其起到的实际效果就是将该临时变量的活跃区间进行了切分，以便重新为其进行寄存器分配。这一步需要大家完善 LinearScan::spillAtInterval(Interval *interval) 函数。具体分为以下三个步骤：

1. 为其在栈内分配空间，获取当前在栈内相对 FP 的偏移；

2. 遍历其 USE 指令的列表，在 USE 指令前插入 LoadMInstruction，将其从栈内加载到目前的虚拟寄存器中；

3. 遍历其 DEF 指令的列表，在 DEF 指令后插入 StoreMInstruction，将其从目前的虚拟寄存器中存到栈内；

插入结束后，会迭代进行以上过程，重新活跃变量分析，进行寄存器分配，直至没有溢出情况出现。

5 评分标准

完成基本要求后，大家就可以生成全部 level1 级别样例的汇编代码了。评测时，需要提交项目 gitlab 链接到希冀 OJ 平台上进行测试。

5.1 目标代码生成

5.1.1 基本要求

1. 实现基本的 IR 指令到汇编指令的翻译，完善 genMachineCode() 函数
   (a) 数据访存指令的翻译，主要只需要完成 StoreInstruction；
   (b) 二元运算指令的翻译，BinaryInstrction；
   (c) 比较指令的翻译；
   (d) 控制流指令的翻译，UncondBrInstr 语句、CondBrInstr 语句、RetInstru 语句等；
   (e) 函数定义及函数调用的翻译。
2. 实现汇编指令的打印
   (a) 实现变量及常量数据的声明函数（即需要在目标代码开头打印出全局变量及常量的声明伪指令，在 lab2 中对该部分已经有所讲解）；
   (b) 完善基础汇编指令的 Output() 函数。
   (c) 完善寄存器分配算法中 linearScanRegisterAllocation() 函数;
   (d) 完善寄存器分配算法中 spillAtInterval() 函数。

5.1.2 进阶要求

1. 实现数组的翻译；
2. 实现浮点类型的翻译；
3. 实现 break、continue 语句的翻译；
4. 实现非叶函数的翻译。

5.2 完整研究报告

5.2.1 报告要求

1. 实验报告内容应包括是你构建编译器，从词法分析模块一直到目标代码生成的全过程；
2. 符合科技论文写作规范，包含完整结构：题目、摘要、关键字、引言、你的工作和结果的具体介绍、结论、参考文献，文字、图、表符合格式规范；
3. 建议使用 latex 撰写。

5.3 评分标准公式

记“本次实验最终得分”为 $Score$，“OJ 通过样例及级别三的必做项中你完成的部分对应成绩加权和”为 $base$(该加权和计算标准可见“构建语法分析器”实验指导书，满分 12 分)，“本次作业讲解得分”为 w_a(满分 100 分)，“本次实验报告得分”为 w_b(满分 100 分)，“额外加分”记为 $bonus$(额外加分标准可见“构建语法分析器”实验指导书)，“迟交罚分比例”为 $penalty$。

评分标准公式如下:

$$Score = (base * (w_a/100 * 80\% + w_b/100 * 20\%) + bonus) * (1 - penalty)$$

6 结语

课程实验转眼间进入尾声。一个学期，七次实验，我们动手实现了属于自己的编译器。或许这是你第一次面对如此规模巨大的工程，相信这对于正在阅读的你是一件很有成就感且“浪漫”的挑战，也相信你在丰富的课程实验中学有所获。最后，希望同学们虽面对繁重的课程、生活压力，也仍要保持探索知识的好奇心，以及对生活的无限热爱。

——孙一丁

希望同学们能喜欢编译原理这门课程，享受实验过程。每次实验都是助教们精心准备的，然而助教水平有限，恳请同学批评指正。最后，祝大家期末顺利！

——杨科迪

祝大家期末愉快！

——当了一学期学长的费迪学姐

祝大家期末考试顺利！祝大家未来健康快乐，一往无前！！

——潘宇

学没学会编译不知道，但是 Linux 不能再不会了！由于大多数人都会及格就不祝你们期末顺利了，还是祝大家财源滚滚吧。希望能有更多对系统方向工作感兴趣的同学，在此打一个广告：对南开百度联合实验室感兴趣的同学，可以联系王刚老师；对嵌入式系统实验室感兴趣的同学，可以联系宫晓利老师！

——李世阳

参考文献

[1] MASSIMILIANO POLETTO. Linear scan register allocation. https://c9x.me/compile/bib/linearscan.pdf.