課題 2:codegen　コード生成器

22B30862　情報工学系　山口友輝

2024 年 8 月 20 日

1　はじめに

私は今回、XC のコード生成器を実装するために、codegen.c を拡張、修正した。レベル 1,2 は完成し、レベル 3 は途中まで実装できた。本レポートでは、中でも特徴的だった実装について述べたいと思う。

2　レベル 1

2.1　long 型大域変数

long 型大域変数については、codegen_dec 関数内に大域変数宣言としてのコードを追加し、大域変数の値を使うときは rip レジスタにロードすることで、大域変数を使えるようにした。

2.2　代入式

代入式については、codegen_exp 関数内で実装した。基本的な実装方法としては、右辺値を codegen_exp 関数でスタックにプッシュし、左辺値は変数が大域変数であるか局所変数であるかを場合わけして、アドレスをスタックにプッシュした。そのあと、プッシュした値やアドレスをポップし、右辺値を左辺値のアドレスが表すメモリに注に代入した。また、スタックポインタを管理するために、右辺値の値が格納されているレジスタをプッシュすることにも気を付けた。

2.3　while 文

while 文の実装については、まず、while 文が始まるときのラベル (WHILE_START) にジャンプし、while 文の条件式を満たさないときは、while 文が終わる時のラベル (WHILE_END) にジャンプし、ジャンプしなかったときは while 文の処理を WHILE_START に記述し、処理が終わったら WHILE_START にジャンプというような方法で、実装した。

また、条件文が真であるか偽であるかの判別には 0 で初期化された rsi レジスタが 0 を超えている場合は真、超えていない場合は偽として、考えた。この処理については後程記述する。

2.4　二項演算子 (<,+,-)

二項演算子は基本的に、2 つの子を codegen_exp 関数によって、計算、プッシュし、その値をポップしてから処理を行った。+,-についてはポップした値を add 命令や sub 命令で処理した後に、その値をスタック

にプッシュした。これは、レベル 2 の二項演算子 (*,/.) でも同様の流れで処理を実装している。

<については、式の左側の値が式の右側の値よりも小さかったら、rsi レジスタをインクリメントし、条件文が真になるようにした。レベル 2 の==演算子の処理も同様で、式の左の値と右の値が等しいときに rsi レジスタをインクリメントした。

3 レベル 2

3.1 if 文、if-else 文、return 文

ifelse 文については、if 文の中身の式を評価し、rsi レジスタが 0 より大きければ if 文のラベルにジャンプ、rsi レジスタが 0 以下であれば else 文のラベルにジャンプし、ジャンプ左 k の処理が終わったら、ifelse 文の終わりを表すラベルにジャンプした。

return 文の実装については、返り値を codegen_exp によって計算し、その値をポップすることで実装した。

3.2 二項演算子 (||,&&)

|| については、式の左側の式と右側の式をそれぞれ処理することで、どちらかの式が正しければ、rsi レジスタがインクリメントされ、while 文や、ifelse 文で式の真偽を評価できるようになっている。

&&については、右側の式を処理してから rsi をデクリメントしてから、左側の式を処理することで、左右の式がどちらも真でないと rsi レジスタが 1 になっていないようにした。

4 レベル 3

レベル 3 では単項演算子*の処理まで行えた。

基本的には*がついた式の右辺値の処理を行ってからアドレスや、そのアドレスが指すメモリ値を格納してからスタックにプッシュするという流れだが、二項演算子=の左辺値に*の式がある場合は rax レジスタに値をロードする代わりに*の式の右辺値を codegen_exp で処理することで、実装した。

ポインタ型の足し算についても、TYPEKIND を利用して、実装を試みたが、こちらは上手くいかなかった。

5 考察

今回は使う値を全てスタックにプッシュしてからポップするという実装方針でコンパイラを作成したが、この処理方法だと、命令数が多くなり、無駄な実行時間がかかってしまう。実行時間がより早いコンパイラを作成するためには、このような余分な命令を減らすことで実現可能だと考えた。

しかし、今回の実装方法はスタックを最大限に用いることで機械的な実装が容易であった。これはスタックフレームを利用することの利点の 1 つだと考えた。