情報科学特別講義A

C++を用いた関数実行可能なスタックマシンの実装と評価

第一回発表資料

発表者 阿部 碧音

目次

1. コンセプト
2. 要件定義
3. 実装について
4. デモ
5. 評価
6. 今後の展望
7. 質問

コンセプト

・できればどのような環境でも動くべき
　→ Gnu コンパイラを使った C++ での実装

・命令をすべて初めに読み込む
　→ 読み込み・実行を往復するようなモデルだとコンテキスト
　　スイッチの管理が大変であると予想
　　また、あらかじめある程度のエラーチェックが可能

・ある程度の可読性を確保
　→ 現段階では高級言語の実装をしていないため

要件定義

・計算リソースとしてスタックが存在

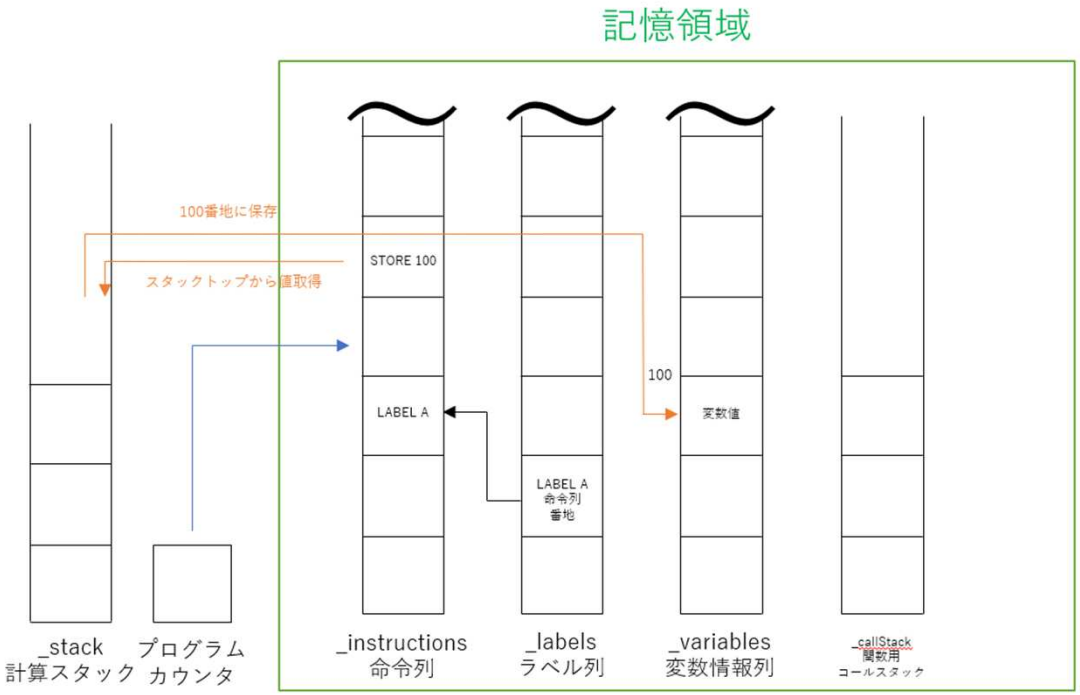
・命令列を記憶領域に保持

・プログラムカウンタにより命令列を逐次的に実行していく

・ラベル列により関数呼び出しに対応

・変数情報列に変数を保持可能

・コールスタックにより関数のネストが可能



実装について

実行可能な命令セット

- ADD, SUB, MUL, DIV … 算術演算
- PUSH, POP … スタック操作
- STORE, LOAD … メモリ操作
- LABEL, JUMP, JPEQ … 命令列操作
- FUNC, CALL, RET … 関数操作
- END, PRINT … その他

実装について (算術演算)

ADD, SUB, MUL, DIV

説明 :

スタックの上二つを用いて算術演算を行い、結果をスタックにPUSHする。

詳細 :

演算は必ず (スタックの上から二番目の値) op (スタックの一番上の値)で行うものとする。
DIV の0割りに関しては、エラーメッセージを出し強制終了とする。

実装について (算術演算)

ADD, SUB, MUL, DIV

説明 :

スタックの上二つを用いて算術演算を行い、結果をスタックにPUSHする。

詳細 :

演算は必ず (スタックの上から二番目の値) op (スタックの一番上の値)で行うものとする。
DIV の0割りに関しては、エラーメッセージを出し強制終了とする。

スタック操作

PUSH op1

説明：

スタックに op1 で指定した値を投入する。

POP

説明：

スタックの一番上の値を破棄する。

メモリ操作

STORE op1

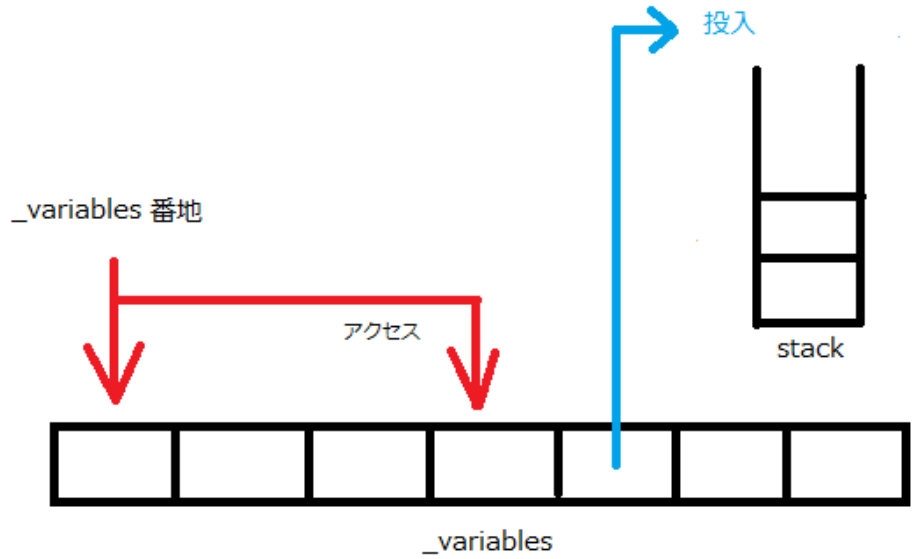
説明：

op1で指定したメモリ番地にスタックの
一番上の値を保存する。

LOAD op1

説明：

Op1で指定したメモリ番地の値を
スタックの一番上にPUSHする。



LOAD命令のイメージ図

命令列操作

LABEL op1

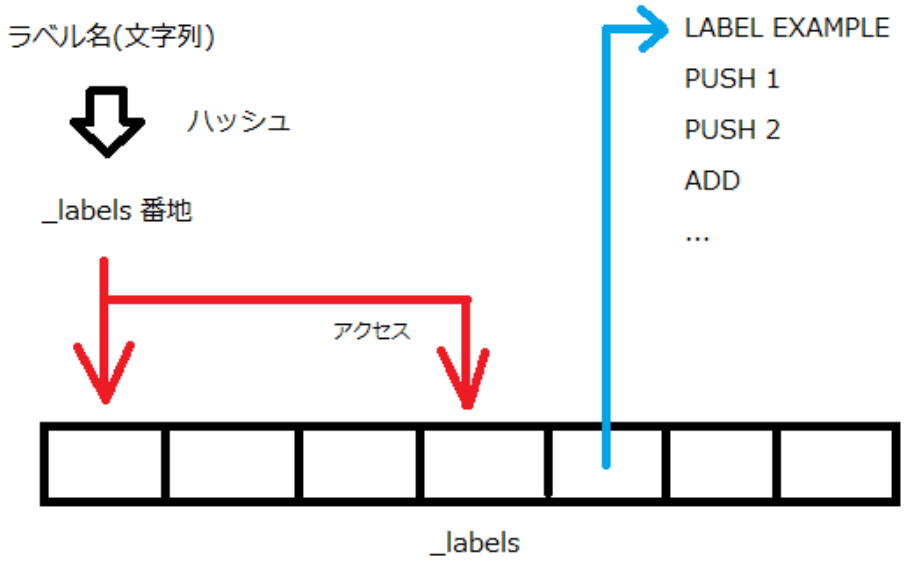
説明：

命令列にラベルを宣言、ラベルは文字列として
指定が可能で、ハッシュ化され_labelsに
命令番号とともに保持される。

JUMP op1，JPEQ0 op1

説明：

op1で指定したラベルにプログラムカウンタ
を設定する。また、JPEQ0 では、スタックの
一番上の値が0だった場合のみJUMP
を実行する。



LOAD命令のイメージ図

関数操作

FUNC op1, RET

説明：
関数の宣言および終了位置を指定する。
Op1では関数に名前を付けることができ、その名前はラベルと同様のメモリ空間に保持される。

注意：
ラベルと同じメモリ空間であり、ハッシュの値が short (2 byte) であるため、宣言が多い場合衝突する可能性がある。

関数操作

FUNC op1, RET, CALL op1

説明：
関数の宣言および終了位置を指定する。
Op1では関数に名前を付けることができ、その名前はラベルと同様のメモリ空間に
保持される。CALL では、op1で指定した関数の位置にプログラムカウンタをセットする。

注意：
ラベルと同じメモリ空間であり、ハッシュの値が short (2 byte) であるため、宣言が
多い場合衝突する可能性がある。また、命令列をあらかじめすべて読み込むため、
宣言されていない関数名を検出可能。

その他

PRINT

説明：
スタックの一番上の値を出力する。

END

説明：
プログラムの終了位置を規定する。

デモ

評価

良い点

- 関数実装によりある程度処理をまとめることができる
- ラベルのオペランドに文字列が使用できることにより可読性が向上

悪い点

- メモリ番地を自身で把握しなくてはならない
- コード全体の量が多くなる傾向にある。

今後の展望

・変数名や関数名などは、もう少し高級な言語を作成したほうが良いと思われる。

名称テーブル・アドレス番地変換などを導入すれば、人間としての可読性向上が見込める。

ただし、前処理の時間は増大すると考えられる。

また、アドレス変換をすることにより、現定義アーキテクチャであるラベル・関数などの要素ごとのメモリ空間分割が必要なくなる。

今後の展望

・並列化

小規模スタックを複数並べることによるSIMD演算化

・命令オペランドの型汎用化

現状数値型にしか対応していないため、それらの汎用化

質問

- もう少しオペコード分岐をうまく記述する方法はないのか?
  → 現状かなり if が並ぶあまりよくないコード
-