ネットワーク系演習III
コンパイラ
コールスタック

2017年度

齋藤彰一@ネットワーク系

手続き呼び出し処理の課題

- 手続き宣言
  - どのようにして手続きが宣言済みか否かを確認するか？
- 引数
  - どのように引き渡すか？
  - どのようにして手続き内で変数として使えるようにするか？
  - どのようにして大域変数と区別するか？
  - どのようにして手続き呼び出し毎に別々に割り当てるのか？
- 局所変数
  - どのようにして大域変数と区別するか？
  - どのようにして手続き呼び出し毎に別々に割り当てるのか？

手続き宣言

- 概要
  - 手続き名だけチェックする（簡単）
- 考え方
  - 手続きはプログラムファイル内ならどこからでも呼び出せる
  - つまり、手続き呼び出しがあった時点で、それ以前に手続き宣言が必要
- 手順
  - １）手続き宣言(procedure)を見つけた時点でテーブルに登録する
  - ２）手続き呼び出しを見つけた時点で、手続き名がテーブルに登録されているか確認する
    - 登録されていれば、呼び出し可能
    - 登録されていなければ、エラー
  - テーブル
    - 大域変数の記号表もしくは手続き名専用の記号表
- 拡張オプション
  - 引数の数をチェックする

引数と局所変数

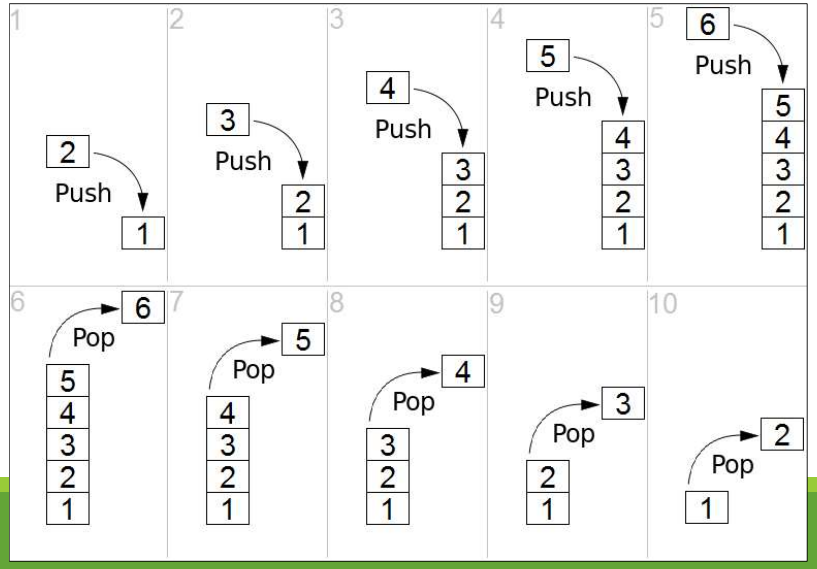
- 共通点
  - 手続き内のみで利用可能
  - 大域変数と同じ名前の場合は、手続き内で宣言した引数・局所変数が有効
  - 手続き実行毎に新しいアドレス（記憶場所）が必要
- 相違点
  - 変数宣言の方法
    - 引数　procedure(a,b,c)
    - 局所変数　var a,b,c
  - 引数は、呼び出し元が値を定める

記号表

- 手続き内のみで利用可能
  - 該当する手続きをコンパイル中だけ記号表で検索できればいい
    - 手続きをコンパイル中だけ、引数名・局所変数名を記号表に登録
    - 手続きのコンパイルが終わったら、引数名と局所変数名を破棄する
  - ⇒大域変数用記号表と、引数・局所変数用記号表を別々にすると良い
- 大域変数と同じ名前の場合は、手続き内で宣言した引数・局所変数が有効
  - 大域変数は記号表で名前とアドレスを管理している
    - 大域変数が使用された場合の処理
      1. 変数名と同じ名前が記号表に登録されているかを、大域変数記号表1つ1つを調べる
      2. 最初に一致した変数のアドレスを使用する
  - 「最初に一致」したものを使うのだから…
    - 大域変数用記号表より前に引数・局所変数用記号表を検索すれば良い
  - ⇒変数検索は、まず引数・局所変数用記号表を検索し、ない場合だけ大域変数用記号表を検索する。最初に一致した変数のアドレスを使用する。

引数・局所変数の管理

- 手続き実行毎に新しいアドレス（記憶場所）が必要
  - 大域変数は固定的なアドレス(0,1,2,…)を割り当てていた
  - 固定的アドレスでは、「手続き実行毎」に新しいアドレスを割り当てることはできない
    - 手続きが何回実行されるかコンパイル時点ではわからないので、何回か分をまとめてコンパイル時に割り当てることもできない
  - 手続き呼び出し毎にアドレスを定める（しかない）
    - ポイント：呼び出し毎にアドレスが必要だが、使うのは最新の呼び出し分のみ
    - つまり、最初に割り当てたアドレスは最後に使い、最後に割り当てられたアドレスは最初に使う　⇒　First In Last Out (FILO)　⇒　スタック構造！



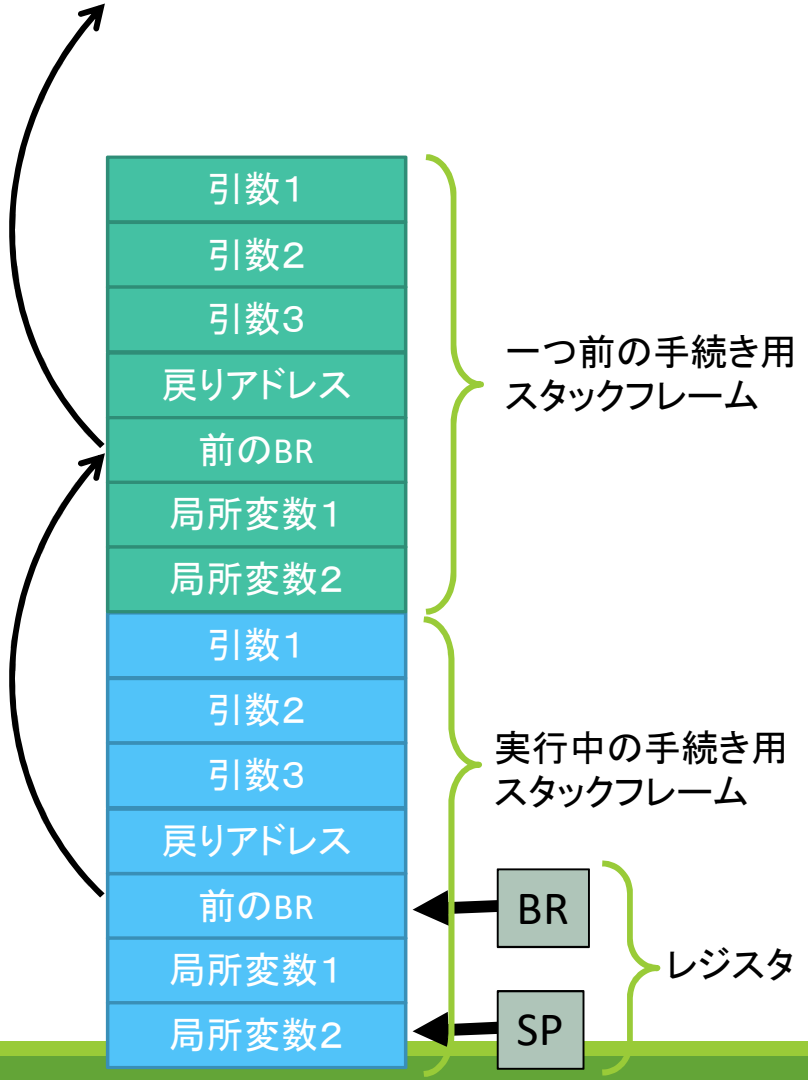
https://ja.wikipedia.org/wiki/スタック

引数・局所変数の管理２

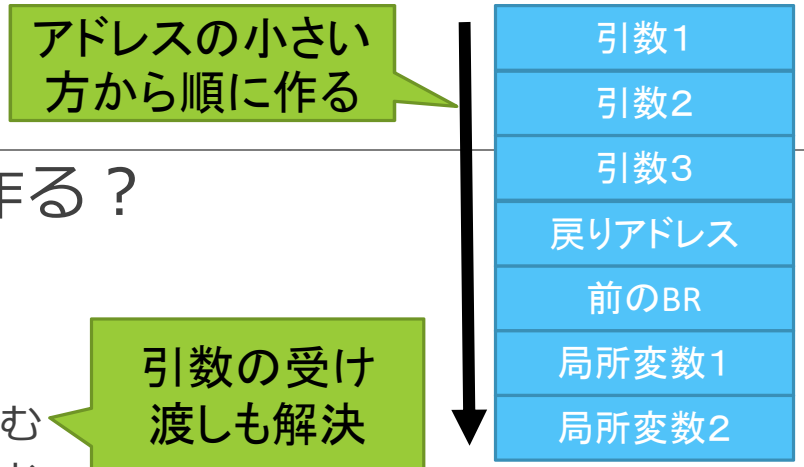
- 手続き実行毎に新しいアドレス（記憶場所）が必要：続き
- 引数・局所変数の管理はスタック構造が良さそう
  - では、スタックをどう使うか？
- スタックの1つ分（スタックフレーム）に何を含めるか？
  - 各手続きの実行に必要なデータ
    - 引数
    - 局所変数
  - 手続き終了後に戻るアドレス
    - 手続きの呼び出し元アドレス
  - ひとつ前のスタックフレームのアドレス
    - 呼び出し元手続きに戻ったときに使用するスタックフレームのアドレスのこと
- 引数・局所変数記号表にはどうやって登録する？
  - 手続き毎にアドレスが変化するので絶対アドレス(固定アドレス)は使えない
  - スタックフレーム内に引数・局所変数が配置されるので、スタックフレーム内での相対アドレスを登録する
  - ⇒スタックフレームのアドレスが分かれば、引数・局所変数のアドレスは計算できる

引数・局所変数の管理

- 手続き実行毎に新しいアドレス（記憶場所）が必要：続き
- スタックフレームの構造
  - BR：ベースレジスタ
    - スタックフレームのアドレスを示すレジスタ
    - マニュアル図20の動的リンク
  - SP：スタックポインタ
    - スタックの先端のアドレスを示す
- 引数・局所変数用記号表への登録
  - BRからの相対アドレスを登録
    - 引数１：-4、引数２：-3、引数３：-2
    - 局所変数１：1、局所変数２：2
  - アセンブリ命令の記述例
    - 引数１をレジスタ0にロード
      - load r0, -4(BR)
  - BRを実行中のスタックフレームに合わせることで、引数・局所変数のアドレスを求めることができる

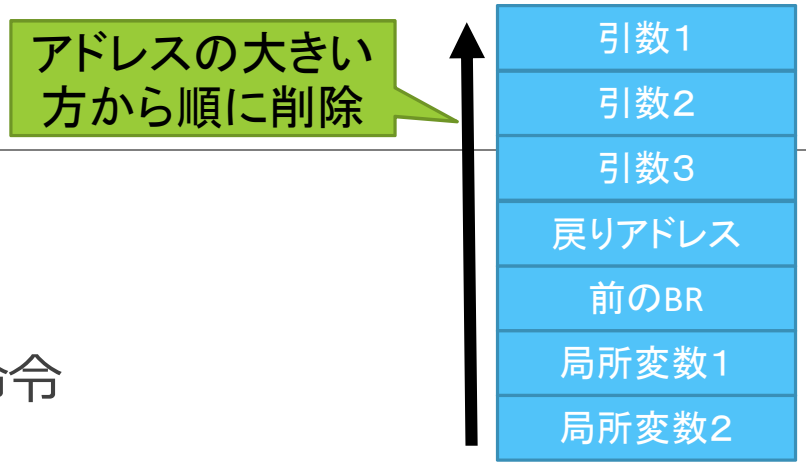


命令１



- どうやってスタックフレームを作る？
- 手続き呼び出し元での処理
  - 引数：proc(a,b,c)
    - a を求め、その値をpush命令でスタックに積む
    - b を求め、その値をpush命令でスタックに積む
    - c を求め、その値をpush命令でスタックに積む
    - 第1引数から順に値を求めて、それぞれpush命令でスタックに積む
  - 戻りアドレス：call命令
    - Call命令を使うことで、手続きへのジャンプと戻りアドレスをスタックに積むという2つの動作が実行される
- 呼び出された手続きでの処理
  - BRをpush命令でスタックに積む
    - この時点でのBRは、読み出し元（一つ前）のスタックフレームを指している
  - SPをBRにコピーする
    - 現在実行中の手続きのスタックフレームを指すようにする
  - 局所変数領域を割り当てる
    - SP = SP + 局所変数の数：SPを直接

命令2



- 手続きの終了の仕方
- 呼び出された手続きでの処理
  - return文を見つけた時のアセンブリ命令
  - 局所変数領域を削除する
    - SP = SP – 局所変数の数
  - BRを呼び出し元手続きを指すように戻す
    - BR = SPが指す値
  - 処理を呼び出し元手続きに戻す
    - ret命令
- 呼び出し元での処理
  - call命令の次のアセンブリ命令
  - 引数を削除する
    - SP = SP – 引数の数