コンピュータ科学実験3
課題4, 5

2020年01月09日

課題4

- 四則演算を行うソースプログラムに対応するコンパイラを作成
  - 出力ファイル名は result.ll とすること
- ソースプログラム ex1.p を入力とし，
  生成したコードをLLVM IR（lliコマンド）で実行して結果を確認

課題5

- PL-0コンパイラを完成
  - 手続き
  - 制御文
  - 入出力処理
- pl0a.p, pl0b.p, pl0c.p, pl0d.pを
  ソースプログラムとして実行し，結果を確認
- 出力ファイル名：result.ll

コードの内部表現

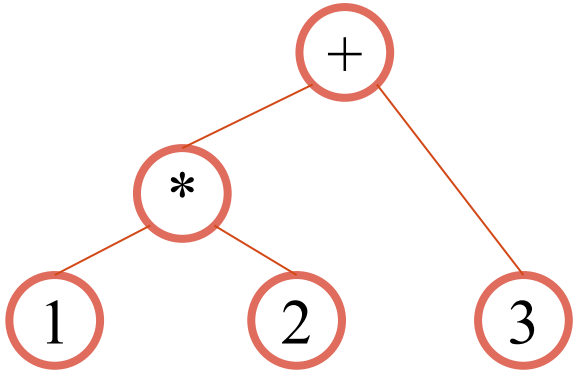
- 生成するコードは関数定義の列
  - Fundecl型のリストで各関数定義を保持
    - Fundecl型は，内部に命令の列を持つ
- 命令の列
  - LLVMCode型のリストで保持
    - 内部では，各命令の内容を共用体で保持

計算の手順

- 計算の例

  1 * 2 + 3

  %1 = mul nsw i32 1, 2
  %2 = add nsw i32 %1, 3



- +演算は，木の左側の演算結果と右側の演算結果の和を求める
  - 左右の演算結果を保持したレジスタ番号が必要
  - 数値リテラルの場合はその値

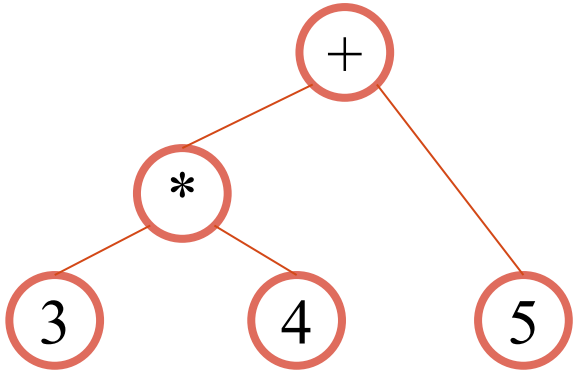
Factor型（のスタックFactorstack）で保持しておく

計算の手順

- 計算の例

  3 * 4 + 5

  %1 = mul nsw i32 3, 4
  %2 = add nsw i32 %1, 5

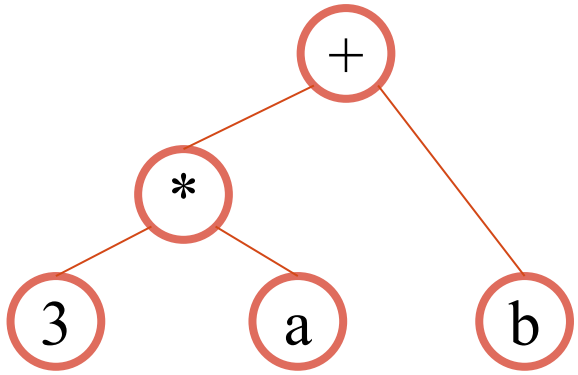


構文解析して
3 を発見→ factorpush
4 を発見→ factorpush
mulの計算（コード生成）
　　2回factorpopして2つのオペランドを得る
　　mulを計算した結果を格納するレジスタ番号（ここでは%1）をfactorpush
5 を発見→factorpush
addの計算（コード生成）
　　2回factorpopして2つのオペランドを得る
　　addを計算した結果を格納するレジスタ番号（ここでは%2）をfactorpush

計算の手順（課題5の場合）

- 計算の例

  3 * a + b



%1 = load i32, i32* @a, align 4
%2 = mul nsw i32 3, %1
%3 = load i32, i32* @b, align 4
%4 = add nsw i32 %2, %3

構文解析して
3 を発見→ factorpush
a を発見→ loadのコード生成をしてレジスタ番号（ここでは%1）を factorpush
mulの計算（コード生成）
　　2回factorpopして2つのオペランドを得る
　　mulを計算した結果を格納するレジスタ番号（ここでは%2）をfactorpush
b を発見→ loadのコード生成をしてレジスタ番号（ここでは%3） factorpush
addの計算（コード生成）
　　2回factorpopして2つのオペランドを得る
　　addを計算した結果を格納するレジスタ番号（ここでは%4）をfactorpush

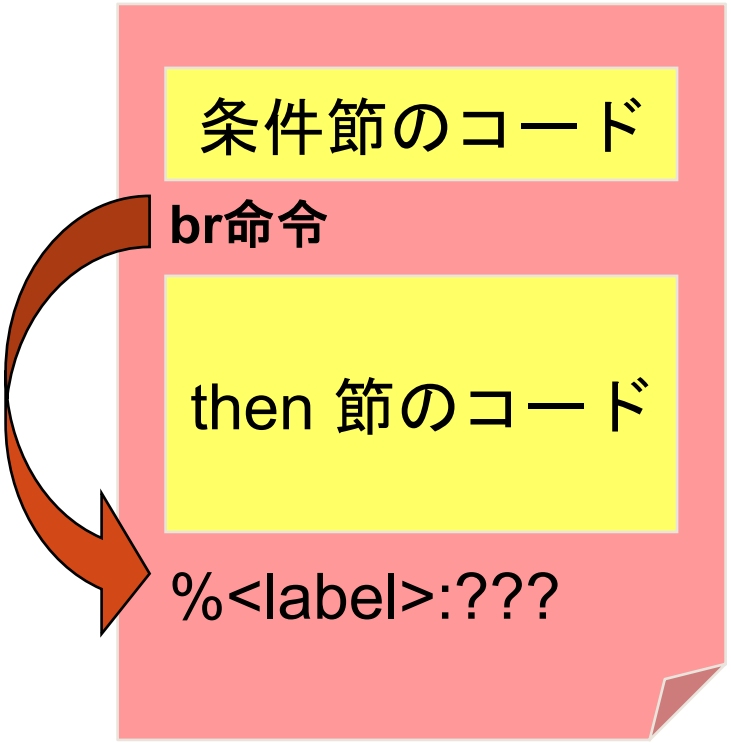
PL-0コンパイラの実現に必要なこと

- 手続き
  - 手続き呼出しと終了
  - 局所変数へのアクセス
- if, while, for 文などの制御文
- read, write

etc.

if…then…

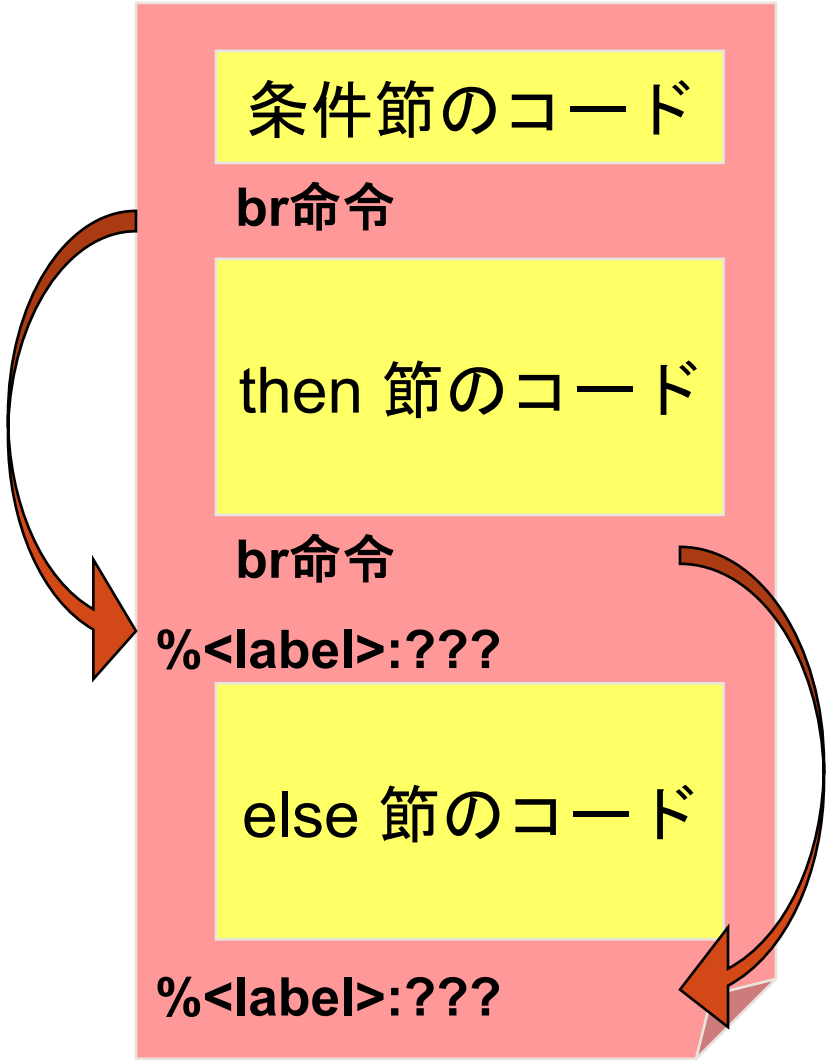
- すべてのコードを生成した後で，アドレスを決定する必要がある




```
if n = 0 then
  begin
  …
  …
  end
…
```

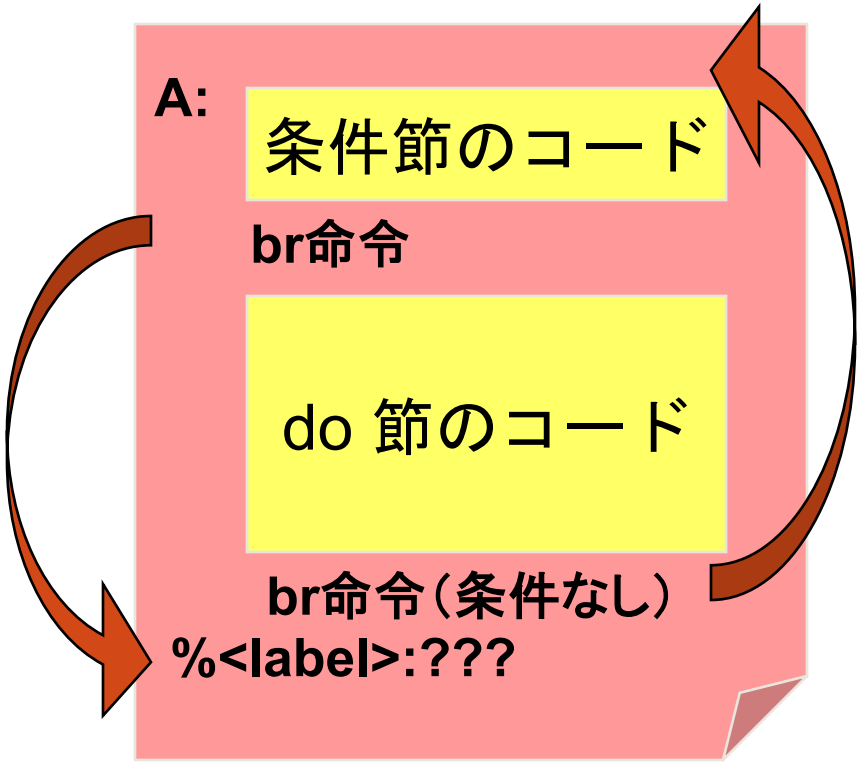
br命令を生成したときに，
そのポインタを保存しておき，
ジャンプ先が確定した時点で修正する
（バックパッチ）

if …then… else…




```
if n = 0 then
  begin
  …
  end
else
  begin
  …
  end
…
```

while … do …




```
while n > 0 do
  begin
  …
  …
  end
…
```

for文

- 自分で考え，レポートで解説すること

入出力処理

- C言語でscanf/printfを実行したときのコード
  - 以下のコードをコンパイルしてみる（付録G参照）

clang -S -O0 -emit-llvm sample.c

コード例

```
#include <stdio.h>
int main(){
    int x;
    scanf(“%d”, &x);
}
```

コード例

```
#include <stdio.h>
int main(){
    printf(“%d”, 100);
}
```

printfに関する定義

```
@.str = private unnamed_addr constant [3 x i8] c"%d¥00"
declare i32 @printf(i8*, ...) #1
```

呼び出し箇所では

```
 %1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds
```

こういうコードを生成する

レポートではこれの意味を説明してください

プログラム及びレポートの提出

- プログラムの提出
  - プログラムファイル
    - Makefile, parser.y, scanner.l
    - その他作成した全ファイル
  - 上記ファイルを kadai4 というディレクトリに保存
    - 以下のコマンドで圧縮し，kadai4.tar.gzを提出

      (kadai4 のディレクトリがある階層で)
      tar zcvf kadai5.tar.gz kadai4
  - レポートのPDF
    - 生成したコードの種類と構文中でのコード生成箇所を解説
    - ex1.p に対して生成されたコードをlli で実行し，結果を確認

プログラム及びレポートの提出

- プログラムの提出
  - プログラムファイル
    - Makefile, parser.y, scanner.l
    - その他作成した全ファイル
  - 上記ファイルを kadai5 というディレクトリに保存
    - 以下のコマンドで圧縮し，kadai5.tar.gzを提出

      (kadai5 のディレクトリがある階層で)
      tar zcvf kadai5.tar.gz kadai5
  - レポートのPDF
    - 「forループの実現方法」，「入出力の説明」
      - 生成したコードの種類と、構文中でのコードの生成箇所について解説
    - pl0a.p～pl0d.p に対して生成されたコードを lli で実行し，結果を確認