Project Assignment 2 - Due Mar 29, 2020

Overview

In this phase of the project, you will implement an intermediate representation within your compiler. The compiler at this stage should be a able to convert the parse tree into a linear intermediate representation that provides a framework for optimization and code generation.

Documentation

The deliverable should include a document describing the specification and design of the program, along with a user guide. Please include the following sections at a minimum:

Program Overview
Usage
Design discussion with limitations and tradeoffs
Language specification - what do you and don't you recognize
Intermediate language design - This should include an overview of the features and tradeoffs in the representation.

In addition, code should be commented reasonably (not excessively), including a comment for each function, and in-function comments for any part of the code that needs additional explanation.

Intermediate Representation

There is a lot of flexibility in what the intermediate representation looks like. The overall design of the representation will be considered, along with how completely it represents the program. The representation needs to be linear and complete enough to represent the program sufficiently to move on to code generation.

As an example, take the following C function:

```
int main()
{
  int i, j;
  i = 12;
  j = (i/2) * 15;
  i = add(i);
  j = add2(i, j);
  return j;
}
```

The following is the assembly language produced by CLANG:

```
_main:                          ## @main
        .cfi_startproc
## %bb.0:
        pushq  %rbp
```

```
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq   %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq   $16, %rsp
        movl   $2, %eax
        movl   $0, -4(%rbp)
        movl   $12, -8(%rbp)
        movl   -8(%rbp), %ecx
        movl   %eax, -16(%rbp)
        movl   %ecx, %eax
        cltd
        movl   -16(%rbp), %ecx
        idivl  %ecx
        imull  $15, %eax, %eax
        movl   %eax, -12(%rbp)
        movl   -8(%rbp), %edi
        callq  _add
        movl   %eax, -8(%rbp)
        movl   -8(%rbp), %edi
        movl   -12(%rbp), %esi
        callq  _add2
        movl   %eax, -12(%rbp)
        movl   -12(%rbp), %eax
        addq   $16, %rsp
        popq   %rbp
        retq
```

The intermediate representation is a representation that lies between the two. It needs to be more explicit than the C language, for example, the expression (i/2) * 15 needs to be broken down into individual instructions. It needs to be less explicit than x86 assembly, for example, we shouldn't need to specify which precise registers are involved or where on the stack we are getting a certain variable.

An example IR is gimple, that is used internally by GCC. Here is the gimple representation of the same function:

```
main ()
{
  int D.2440;

  {
    int i;
    int j;

    i = 12;
    _1 = i / 2;
    j = _1 * 15;
    i = add (i);
    j = add2 (i, j);
    D.2440 = j;
    return D.2440;
  }
  D.2440 = 0;
```

```
  return D.2440;
}
```

In this case gimple is preserving some things such as the variable declarations that aren't strictly required in an IR.

The LLVM IR is more explicit in many ways than gimple. The LLVM for our function is:

```
; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 12, i32* %2, align 4
  %4 = load i32, i32* %2, align 4
  %5 = sdiv i32 %4, 2
  %6 = mul nsw i32 %5, 15
  store i32 %6, i32* %3, align 4
  %7 = load i32, i32* %2, align 4
  %8 = call i32 @add(i32 %7)
  store i32 %8, i32* %2, align 4
  %9 = load i32, i32* %2, align 4
  %10 = load i32, i32* %3, align 4
  %11 = call i32 @add2(i32 %9, i32 %10)
  store i32 %11, i32* %3, align 4
  %12 = load i32, i32* %3, align 4
  ret i32 %12
}
```

This is clearly more detail than you need to represent in your IR. For example, the alignment information, while useful for optimization, is not necessary. Note that LLVM uses the i32 designation in place of "int" and every variable name (here represented by %num) is preceded by it's type. Again, these are decisions that are made to aid in optimization, and you won't need that level of detail.

The IR presented in the book is ILOC, and you can see a lot of examples in the chapter on code shape. The table representation is just one way of creating a data structure to store the IR, and is not the preferred representation when presenting it or thinking about the general form.

Program

The programming language used is up to your project team, but everyone on the team should be able to understand and explain the code. Your program should accept command line arguments dictating its behavior. For this stage, the project should include at least the following command line options.

- An option to output the sequence of tokens and labels. These should look something like this:

- An option to output the parse tree. Any type of pretty-printing for a tree is fine. The parse tree does not need to be complete, but should be correct as far as it goes. Please document limitations.

- An option to output the symbol table. This should include a list of discovered symbols and their types. You may use nesting or any other reasonable method to represent different levels of name space.

- An option to output the intermediate representation. This should be a list of instructions in your representation.

- An option to write out the intermediate representation to a file with a specified name.

- An option to read in an intermediate representation specified instead of a source file.

Anything above and beyond that contributes to the additional 15% outlined below.

Grading

Your grade for this assignment will be a composite of the overall group deliverable and your individual contribution. For the group deliverable, the grade will be based on:

75% - Working program containing minimum requirements
10% - Appropriate documentation
15% - Additional features from the optional feature list

This score will then be scaled based on your level of participation. I will use the git commit logs to determine whether you had a reasonable level of contribution. If there is an issue, I will contact your teammates to get their perspective on your contribution, and may invite you to explain parts of the code to demonstrate understanding if needed.