Now, to overload, we say that each of the types assembly:

representation of x86 assembly:

Second Step: Transforms

An

De

Note:

De

Lets

Our first step will be to

To go from source to assembly, we must do:

Recall, that compilers were invented to

Hooray!

We can now run it:

Lets test our "run-time" without even building the compiler.

Test Systems in Isolation

Lets work

Adder-1

Next, lets see how to do (1) and (2) using our sequence of

Approach:

source.

Simpli

Compiler Pipeline

Recall: What does a Compiler

In 131

Numbers, Unary Operations, Variables

our_code

asm (instance

= a -> Text

class ToX86 a where

reg ::

instr ::

arg ::

expr ::

parse ::

asm ::

asm     ::

4. Numbers + Increment + Decrement + Local Variables

e.g.

let x = add1(7), y = add1(x) in add1(y)

1. Numbers + Increment + Decrement

e.g.

as

Reg

Const

Arg

IMov

Register

Arg

IMov

Instruction

Number

Text

Text

Text

12

"%d"

argv) {

C

printf

argc,

chars

"I wanna hold your hand..."

"It was a dark and stormy night..."

forty_two.run

clang

-f macho64 -o forty_two.o forty_two.s

rax

let x = add1(7), y = add1(x) in add1(y)

clang

-f macho64 -o forty_two.run c-bits/main.c forty_two.o

rax

rax

global our_code_label

... 3. Transform assembly into output-string

... 2. Transform AST into assembly

... 1. Parse text back to the run-time.
Let's zoom into the stack region, which when we start looks like this:

Focusing on "The Stack"

Here's what the memory—i.e. storage—looks like:

Need to dig for more storage space!

There is only a

Problem: Registers are Not Enough

Need to store 3 variables—

Example: let2

Example: let1

Let's look at some examples

Repeat our Recipe

Can you think why

Adder-4

Adder-3

Examples Revisited

Finally, the key step is

Asm

Parse

Let's do the easy bits first, namely

Now let's go back and suitably extend the transforms:

Example-2 Revisited

Source Expressions

Extend Type for Source and Assembly

We will write compiler in

Note correspondence between sub-expressions of

In ASM

In English

How should we compile

Aha, note that

In ASM

Example 1

First, let's look at some examples.

Add

Move

Types…

Note
Using a Recipe

Let's take our first steps in building a simple compiler for a small assembly language!

Step 1: Numbers

- Numbers are fundamental in any programming language.
- We'll start with simple arithmetic operations: addition and subtraction.

Step 2: Types

- At this stage, we'll define the types of variables and how they interact with memory.

Step 3: Transforms

- We'll transform source expressions into a form suitable for compilation.

Step 4: Tests

- Almost done: just write code formalizing the algorithm.

Almost done: just write code formalizing the algorithm.

Assembly Instructions

Now, we're ready to move to the implementation!

Example: let

The assembly looks like

When we compile

QUIZ: let

To compile

Strategy: Variable Definition

At what position on the stack do we store variable \( x \)?

QUIZ: let

Let-bindings and Stacks: Example-2

Required

How to compute mapping from

We have

The stack

...