Language Implementation Basics

Polyglot Programming DC 2015

Brock Wilcox
awwaiid@thelackthereof.org
bwilcox@optoro.com
Full Stack.
Pft.
Transistors → Logic Gates → Logic Units
* Fetch contents of memory
* CASE statement of what to do next
* Local storage (registers)
Input is binary things in memory.

Output is binary things in memory.
Hello world!

.shstrtab.tex
t.data
Assembler:
Assembly text -> binary
section .data
    hello: db 'Hello world!',0x0A
    helloLen: equ $-hello

section .text
    global _start

_start:
    mov eax, 4 ; Output
    mov ebx, 1 ; to stream 1, STDOUT
    mov ecx, hello ; pointer
    mov edx, helloLen ; len
    int 80h ; DO IT

    mov eax, 1 ; exit
    mov ebx, 0 ; status 0
    int 80h ; DO IT
Compiler:

"High Level Language" (HLL) -> binary
Interpreter:

Immediately execute instructions
Interpreted vs Compiled
Typical compiler pieces:

* Lexical Analyzer
* Parser
* AST Builder / Optimizer
* Emitter
* Run-time support
Lexer
characters → words
Parser / AST

words → tree
while
  test
    expr
      binary
        expr
          name: x
        expr
          int: 3
  body
    compound stmt
      stmt
        expr
        func call
          expr
            name: f
          arg list
            expr
              name: 

Emitter

tree → destination language
Run-time support

* Get things started
* Keep track of variables / memory
* Bridge to outside world
Bytecode Systems
Compile:
Lex/Parse → AST → Bytecode

Interpret:
Bytecode → Data Structures → interpreted+compiled blobs
So!
Let's write an Interpreter
First: The run time
Next: The parser
Finally: Execute!
Well, that was fun.
Let's write a Compiler!