Programming Language Creation and Open Source Software

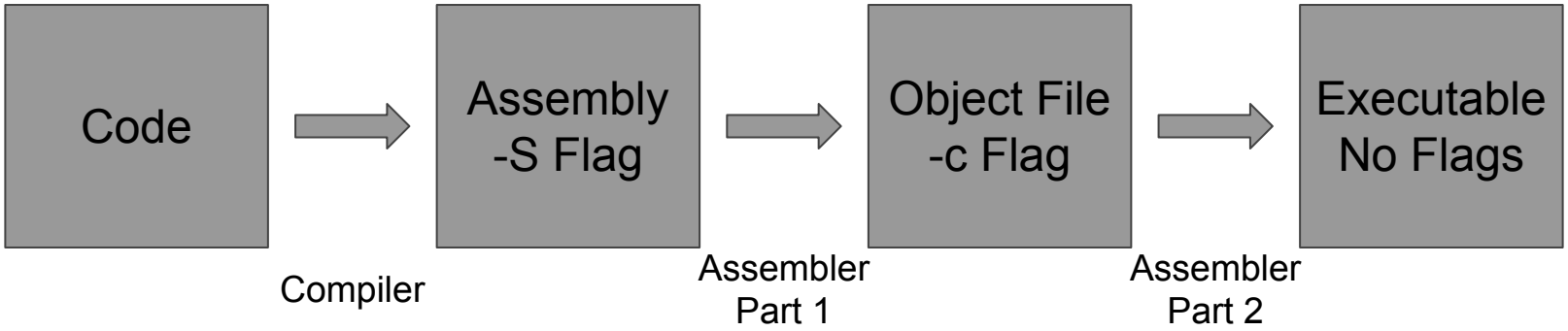
ND LUG: November 15, 2018

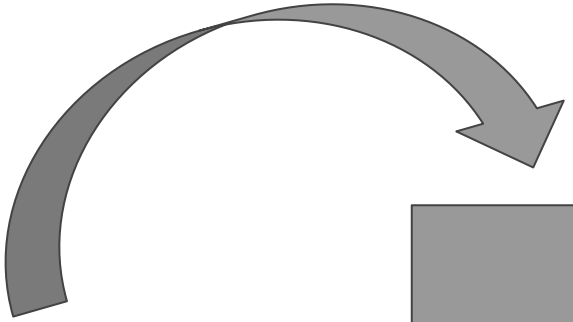
```c
#include "stdio.h"

int main(int argc, char
*argv[]){
    printf("hello world\n");
}
```

```
$ ./a.out
hello world
$
```

The General Compilation Pipeline



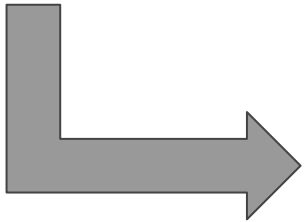


```c
#include "stdio.h"

int main(int argc, char
*argv[]){
    printf("hello world\n");
}
```

Compiler/Interpreter Magic

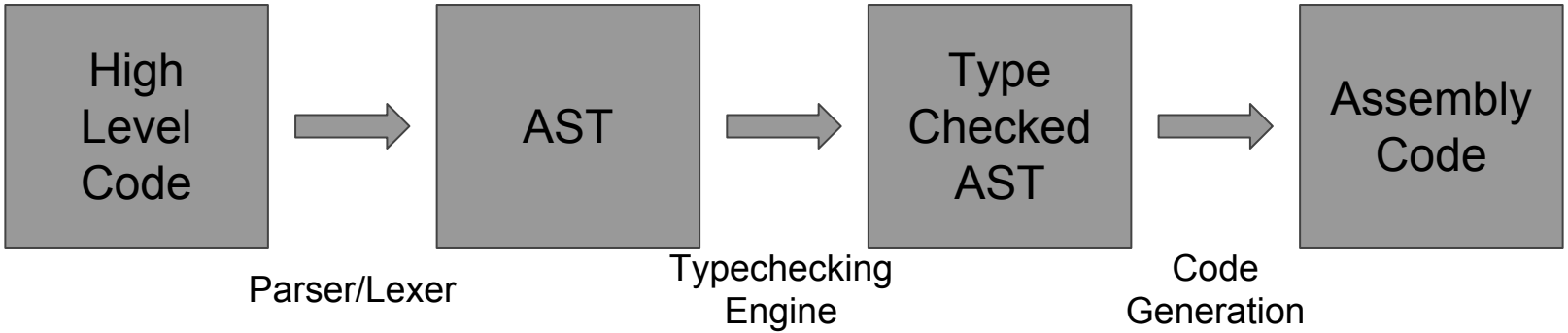
gcc -S hello.c -o hello.s



```
    .globl  _main                   ##
-- Begin function main
    .p2align    4, 0x90
_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    $32, %rsp
    leaq    L_.str(%rip), %rax
    movl    %edi, -4(%rbp)
    movq    %rsi, -16(%rbp)
    movq    %rax, %rdi
    movb    $0, %al
    callq   _printf
    xorl    %ecx, %ecx
    movl    %eax, -20(%rbp)         ##
4-byte Spill
    movl    %ecx, %eax
    addq    $32, %rsp
    popq    %rbp
    retq
    .cfi_endproc

## -- End function
    .section
__TEXT,__cstring,cstring_literals
L_.str:
## @.str
    .asciz  "hello world\n"
```

The General Compiler Pipeline



Lexing and Parsing

- Where you look for certain patterns in you code, and you name the patterns
- Not just for compilers! Your syntax highlighting and linters are dependent on this step as well
- Ultimately transformed into an Abstract Syntax Tree

A Simple Grammar

```
E → E + I | E - I | I
I → [1-9][0-9]*
```

Lexing and Parsing with Open Source

Flex/Lex

- Based on Regular Expressions to identify certain tokens in the High Level Code
- Usually these are just characters, but can be helpful in matching Comment Blocks or string literals
- https://github.com/westes/flex

```
 +   { return TOKEN_ADD; }
 -   { return TOKEN_MINUS; }
[1-9][0-9]* { return TOKEN_INT; }
```

Bison/Yacc

- Most languages are too complicated for Regular Expressions alone
- Use the tokens from Flex to create a grammar
- https://www.gnu.org/software/bison

```
E: E TOKEN_ADD I
   { $$ = expr(EADD, $1, $3); }
   | E TOKEN_MINUS I
   { $$ = expr(EMINUS, $1, $3); }
   | I
   { $$ = expr(INT, NULL, $1); }
   ;
I : TOKEN_INTEGER
     { $$ = expr(INT_LIT, atoi(yytext) };
```

Abstract Syntax Tree

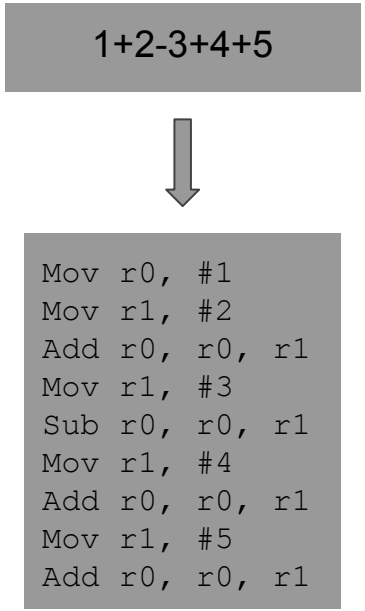
- The main representation of what the compiler works with
- Each structure in a language will have its own node type, and different actions will act in certain ways depending on the type of node
- VERY implementation specific, but generally, a multi-language compiler will work to move each frontend towards the same tree

```
`-FunctionDecl 0x7fd1b60cb8b8 <test.c:3:1, line:5:1> line:3:5 main 'int
(int, char **)'
  |-ParmVarDecl 0x7fd1b60cb6c0 <col:10, col:14> col:14 argc 'int'
  |-ParmVarDecl 0x7fd1b60cb7a0 <col:20, col:31> col:26 argv 'char
**':'char **'
  `-CompoundStmt 0x7fd1b60cbaa0 <col:33, line:5:1>
    `-CallExpr 0x7fd1b60cba40 <line:4:5, col:27> 'int'
      |-ImplicitCastExpr 0x7fd1b60cba28 <col:5> 'int (*)(const char *,
...)' <FunctionToPointerDecay>
      | `-DeclRefExpr 0x7fd1b60cb968 <col:5> 'int (const char *, ...)'
Function 0x7fd1b60b5160 'printf' 'int (const char *, ...)'
      `-ImplicitCastExpr 0x7fd1b60cba88 <col:12> 'const char *'
<BitCast>
        `-ImplicitCastExpr 0x7fd1b60cba70 <col:12> 'char *'
<ArrayToPointerDecay>
          `-StringLiteral 0x7fd1b60cb9c8 <col:12> 'char [13]' lvalue
"hello world\n"
```

Type checking Demo

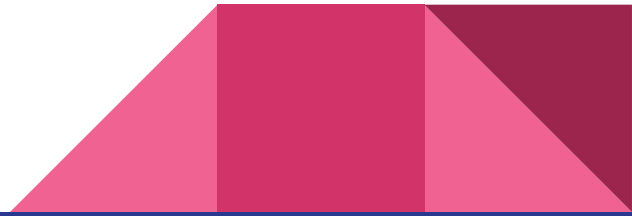
Code Generation

- This is where we take the AST to the assembly code
- This involves looping over the AST multiple times to set up data, global variables, and then following the patterns of the AST to actually have runnable assembly code



Intermediate Representation

- There are a lot of weirdnesses in individual assembly languages
- Register spilling, stack conventions, long variant functions, optimizations, and more and more and more
- So we use an abstraction on assembly!

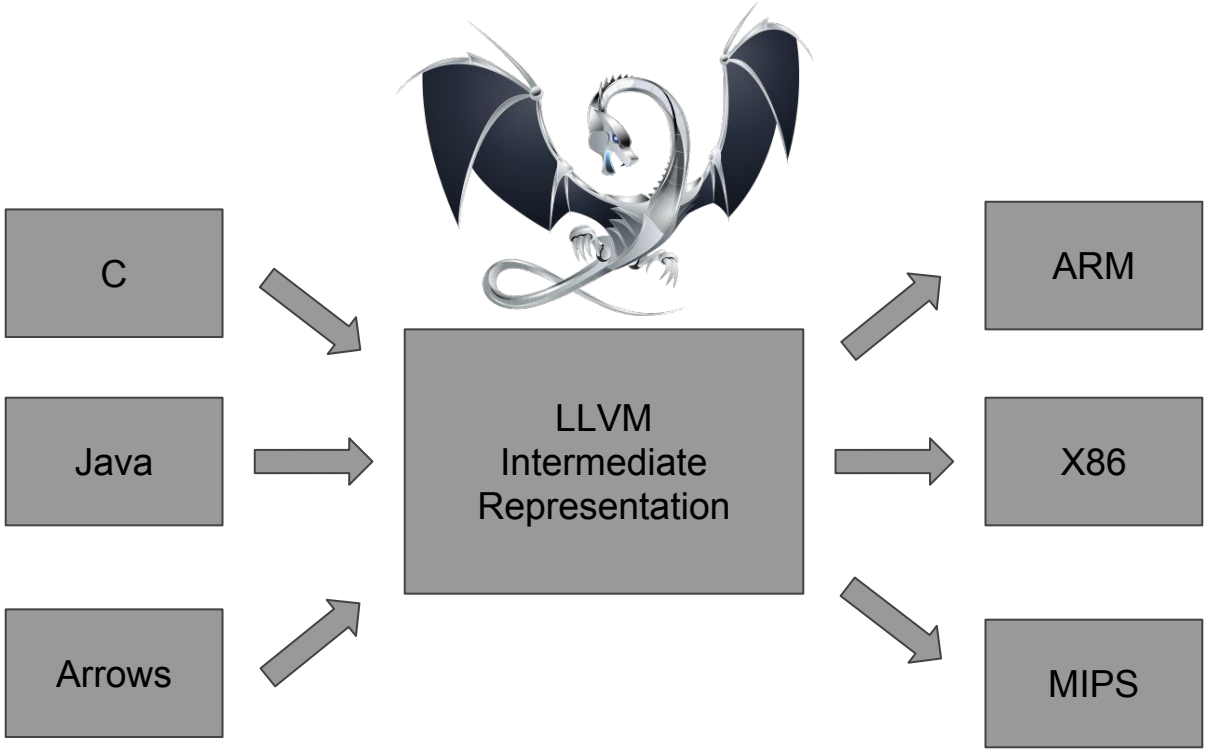


Q: Don’t we have to get to Assembly eventually?

A: Let someone else do it for you

LLVM: A multisource, multitarget Open Source Project

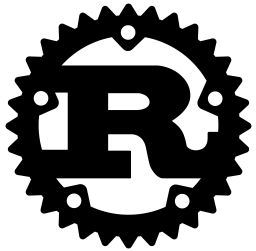
https://llvm.org



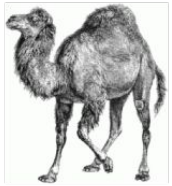
Apple’s Open Source Languages and Tools

Google’s Open Source Languages and Tools

More Open Source Languages and Tools



python™



Perl

Questions?