LLVM Overview

Brandon Starcheus & Daniel Hackney

Outline

- What is LLVM?
- History
- Language Capabilities
- Where is it Used?

LLVM
COMPILER
INFRASTRUCTURE

What is LLVM?

What is LLVM?

- Compiler infrastructure used to develop a front end for any programming language and a back end for any instruction set architecture.

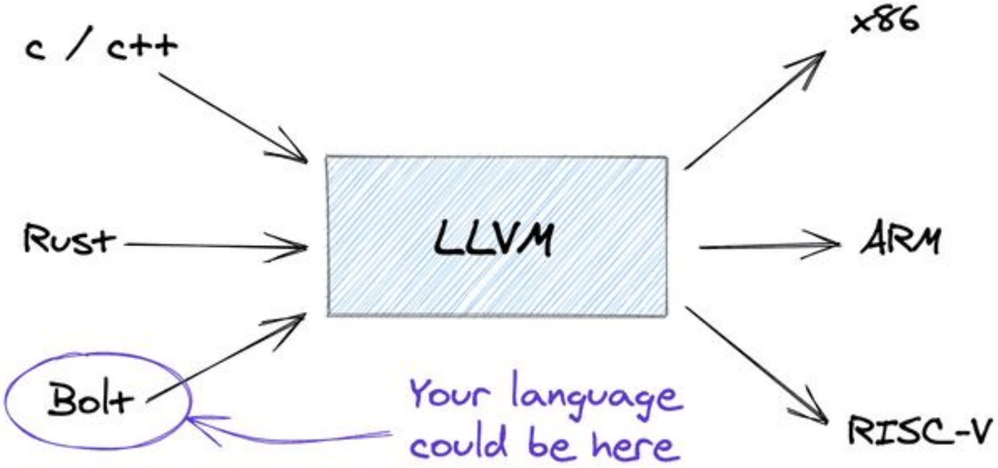
- Framework to generate object code from any kind of source code.

- Originally an acronym for “Low Level Virtual Machine”, now an umbrella project

- Intended to replace the GCC Compiler

What is LLVM?

- Designed to be compatible with a broad spectrum of front ends and computer architectures.



What is LLVM?

LLVM Project

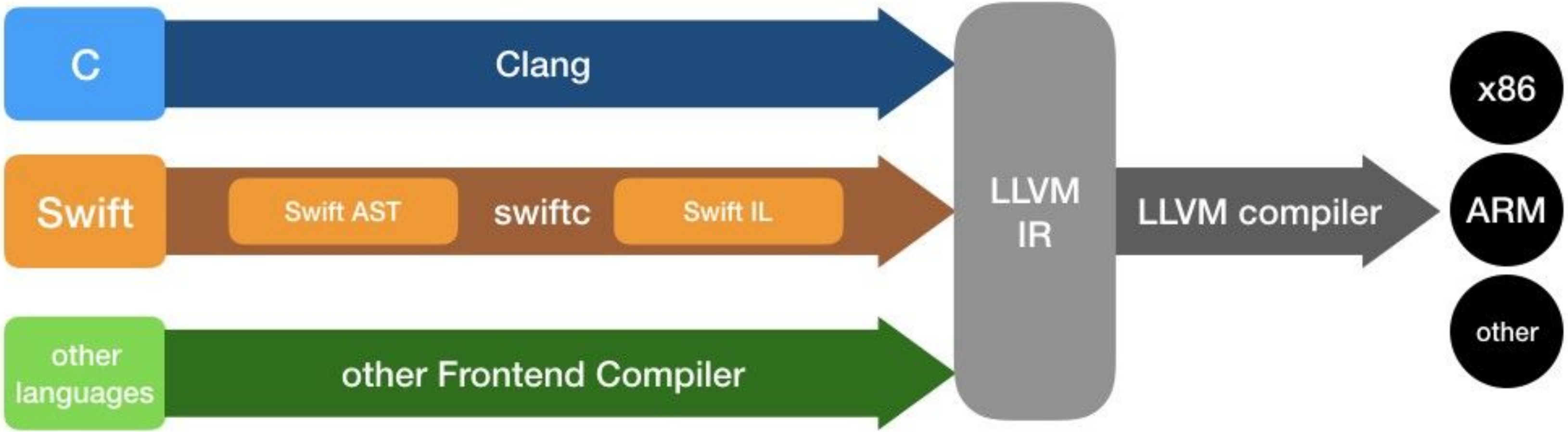
- LLVM (Compiler Infrastructure, our focus)
- Clang (C, C++ frontend)
- LLDB (Debugger)
- Other libraries (Parallelization, Multi-level IR, C, C++)

What is LLVM?

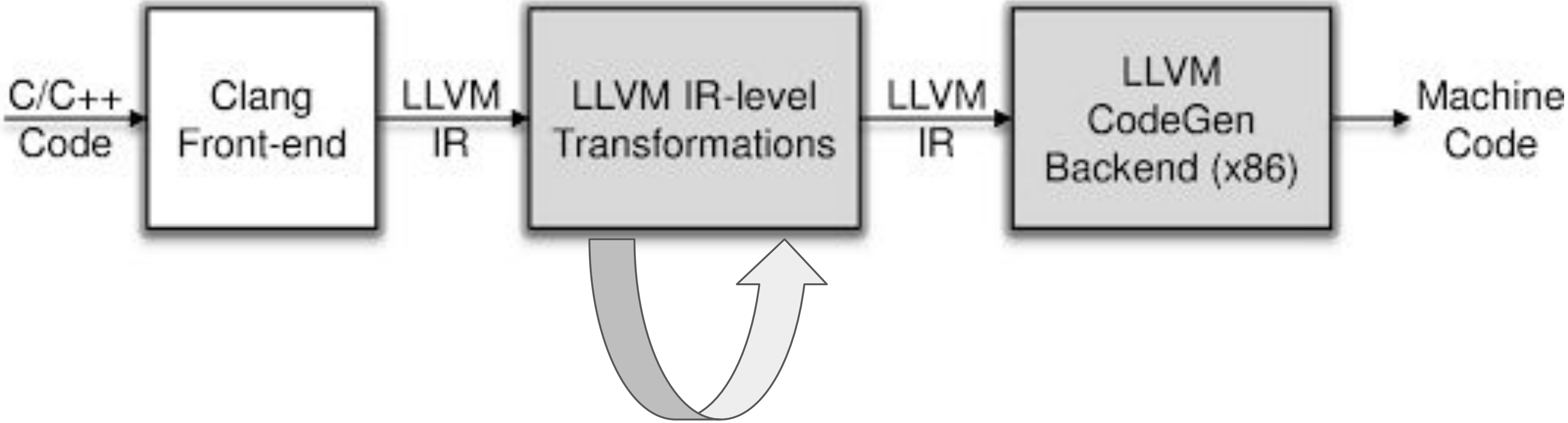
LLVM Project

- LLVM (Compiler Infrastructure, our focus)
  - API
  - llc Compiler: IR (.ll) or Bitcode (.bc) -> Assembly (.s)
  - lli Interpreter: Executes Bitcode
  - llvm-link Linker: Bitcode (.bc) -> Bitcode (.bc)
  - llvm-as Assembler: IR (.ll) -> Bitcode (.bc)
  - llvm-dis Disassembler: Bitcode (.bc) -> IR (.ll)

What is LLVM?


C
Clang
Swift
Swift AST
swiftc
Swift IL
other languages
other Frontend Compiler
LLVM IR
LLVM compiler
x86
ARM
other

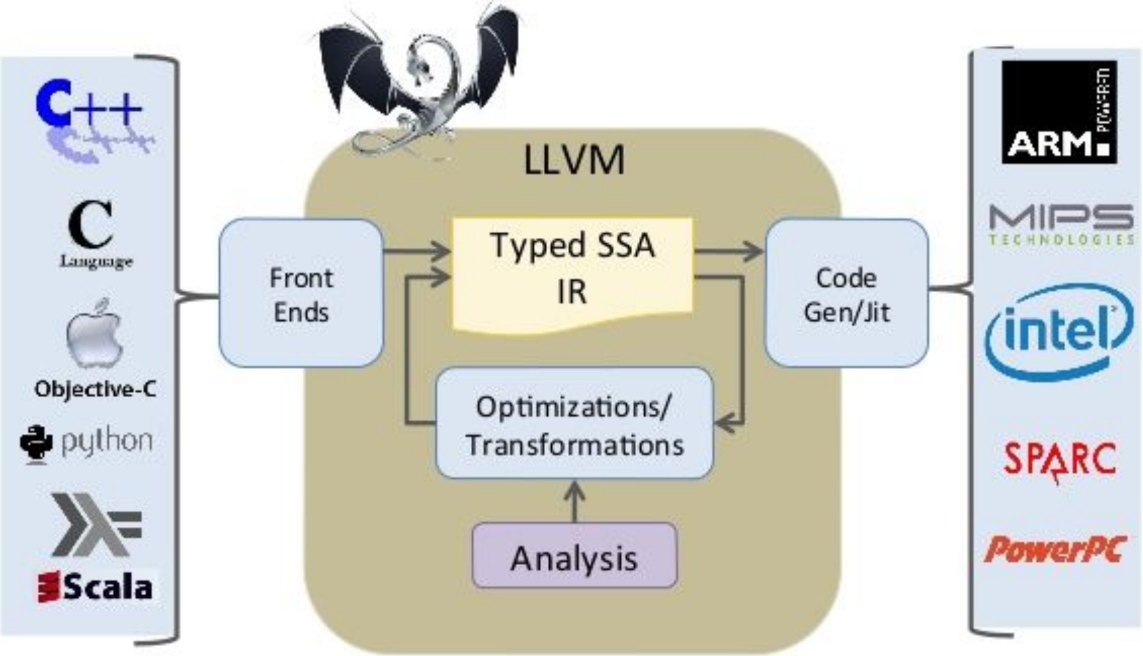
What is LLVM?



Optimizations

LLVM Compiler Infrastructure

[Lattner et al.]

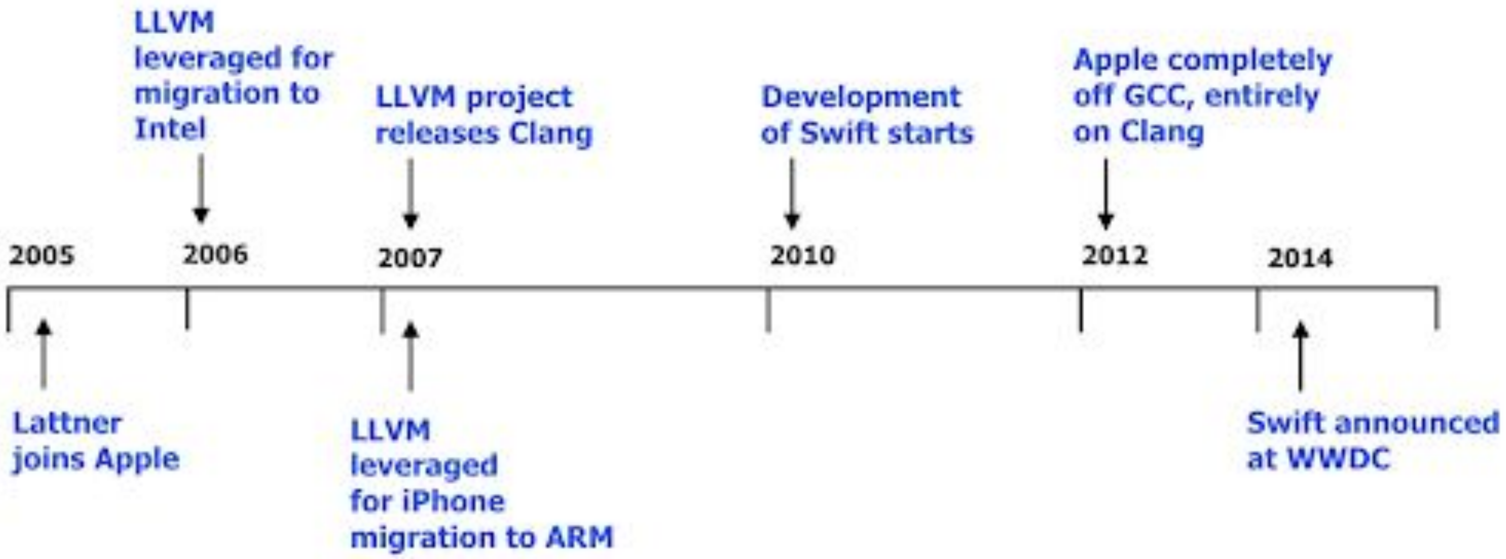


History

History

- Developed by Chris Lattner in 2000 for his grad school thesis
    - Initial release in 2003
- Lattner also created:
    - Clang
    - Swift
- Other work:
    - Apple - Developer Tools, Compiler Teams
    - Tesla - VP of Autopilot Software
    - Google - Tensorflow Infrastructure
    - SiFive - Risc-V SoC’s

History



2005

Lattner joins Apple

2006

LLVM leveraged for migration to Intel

2007

LLVM project releases Clang

LLVM leveraged for iPhone migration to ARM

2010

Development of Swift starts

2012

Apple completely off GCC, entirely on Clang

2014

Swift announced at WWDC

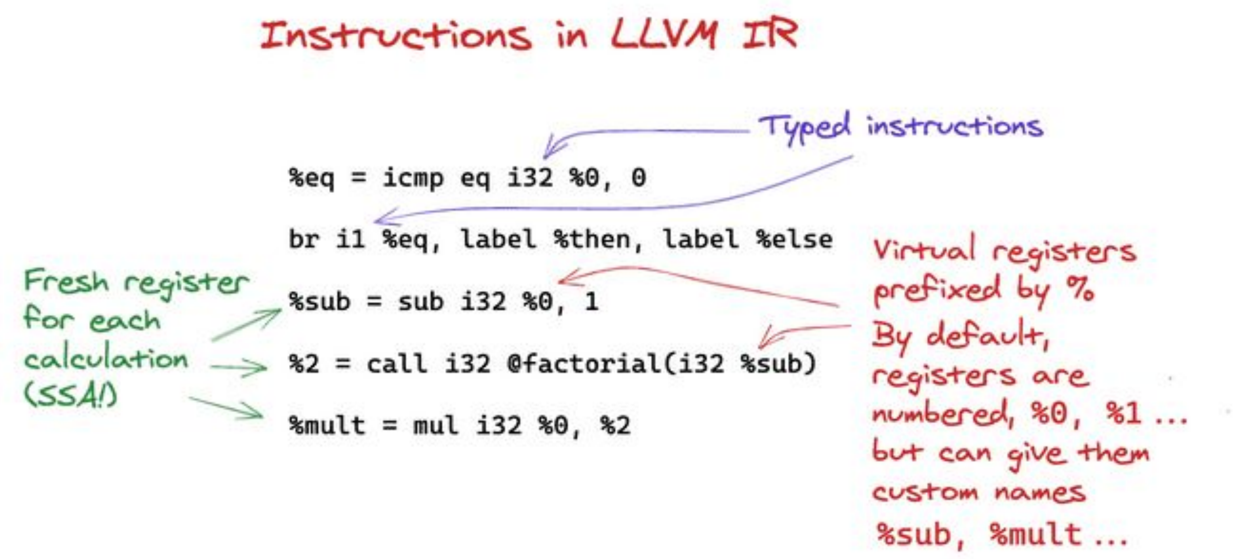
Language Capabilities

Language Capabilities

- Infinite virtual registers
- Strongly typed
- Multiple Optimization Passes
- Link-time and Install-time Optimization
- Target Independent

Language Capabilities

- LLVM IR looks like assembly with types, without machine-specific details.
    - Must be in SSA (Static Single Assignment) form, which makes it easier to optimize.



Instructions in LLVM IR

Typed instructions

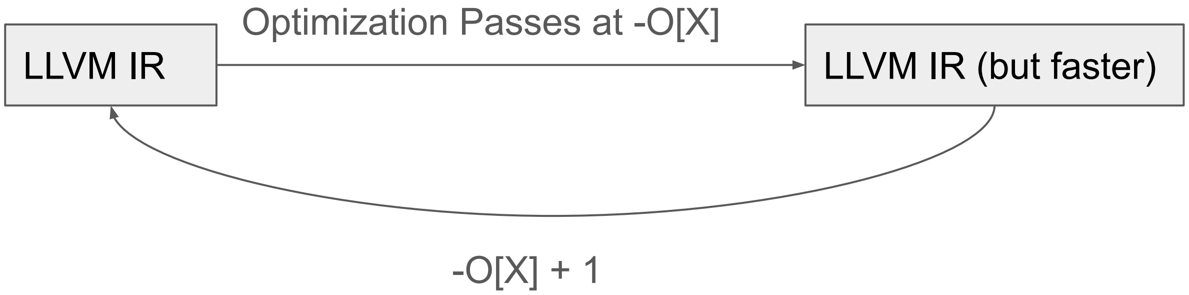
```
%eq = icmp eq i32 %0, 0
br i1 %eq, label %then, label %else
%sub = sub i32 %0, 1
%2 = call i32 @factorial(i32 %sub)
%mult = mul i32 %0, %2
```

Fresh register for each calculation (SSA!)

Virtual registers prefixed by %
By default, registers are numbered, %0, %1 ... but can give them custom names %sub, %mult ...

Language Capabilities

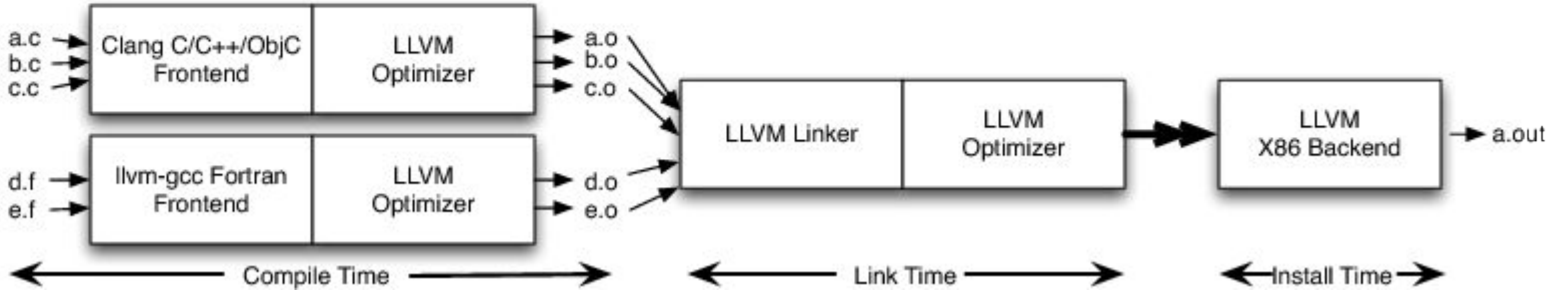
- Multi-pass Optimizations



- Different types of passes depending on the Optimization Level (-O[X])
- Loosely coupled optimization levels
- Implementers can customize pass-order and add custom passes
    - Each LLVM pass is a C++ Class derived from the Pass class

Language Capabilities

- Link-time and Install-time Optimization (and more)
  - Allows partial compilation: save progress to disk, continue work in the future
  - Allows optimizations across file boundaries (between .o files)
  - Allows hardware-specific optimizations (Install-Time Compilation)



Basic Optimizations

- X - 0 -> X

  ```
  if (match(Op1, m_Zero()))
    return Op0;
  ```

- X - X -> 0

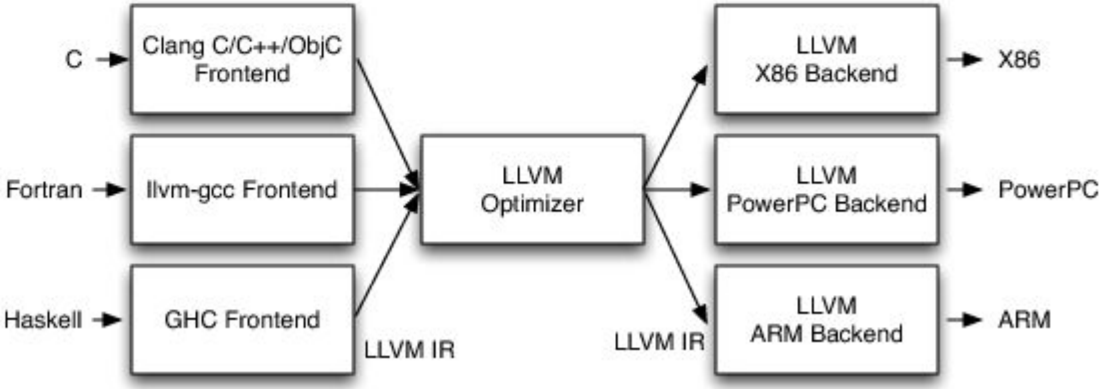
  ```
  if (Op0 == Op1)
    return Constant::getNullValue(Op0->getType());
  ```

- (X*2) - X -> X

  ```
  if (match(Op0, m_Mul(m_Specific(Op1), m_ConstantInt<2>())))
    return Op1;
  ```

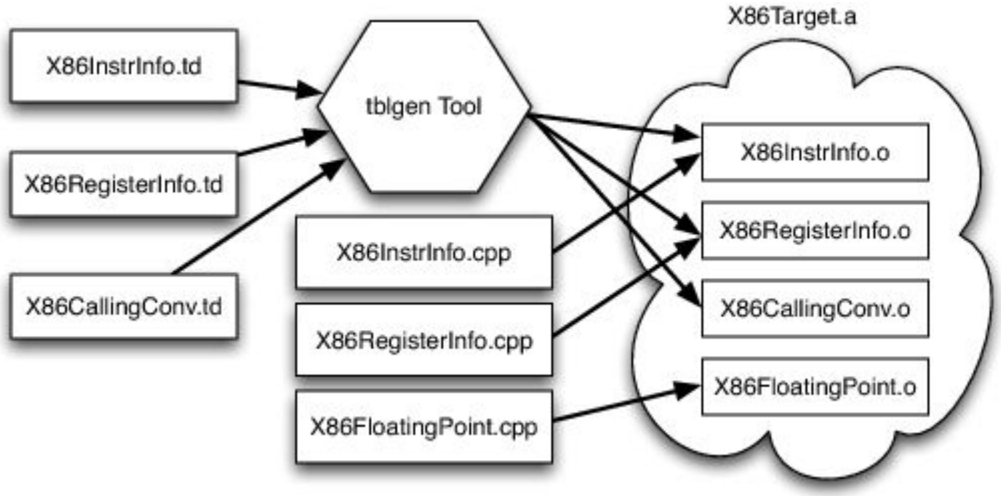
Language Capabilities

- Target Independent



Language Capabilities

- Target authors can create “Target Definition” (.td) files processed by the LLVM tblgen tool
  - Eliminates ambiguity around particular computer architectures (x86, ARM, etc.)



Key Components of the API

Context

Module

IRBuilder

Function

BasicBlock

Value

API Usage

```
Value *IfExprAST::codegen() {
    Value *CondV = Cond->codegen();
    if (!CondV)
        return nullptr;

    // Convert condition to a bool by comparing non-equal to 0.0.
    CondV = Builder->CreateFCmpONE(
        CondV, ConstantFP::get(*TheContext, APFloat(0.0)), "ifcond");

    Function *TheFunction = Builder->GetInsertBlock()->getParent();

    // Create blocks for the then and else cases.  Insert the 'then' block at the
    // end of the function.
    BasicBlock *ThenBB = BasicBlock::Create(*TheContext, "then", TheFunction);
    BasicBlock *ElseBB = BasicBlock::Create(*TheContext, "else");
    BasicBlock *MergeBB = BasicBlock::Create(*TheContext, "ifcont");

    Builder->CreateCondBr(CondV, ThenBB, ElseBB);
    …
}
```

Source: LLVM’s Kaleidoscope Example

Where is it Used?

Where is it Used?

- Rust - static/native compilation
- Swift
- Julia
- OpenCL: Apple, Nvidia, Intel
- Apple OS’s & Dev Tools
- Apple maintains a fork for their use
- Sony: CPU Compiler for PS4
- Nvidia - GPUs and internally
- ARM maintains a fork LLVM 9 as the "Arm Compiler"
- IBM - C/C++ and Fortran compilers
- And many more than these...

References

http://www.aosabook.org/en/llvm.html

https://mukulrathi.co.uk/create-your-own-programming-language/llvm-ir-cpp-api-tutorial/

https://llvm.org/docs/tutorial/