Compiler Construction

What is a Compiler?

Some History

“ *A **compiler** was originally a program that **compiled** subroutines. When in 1954 the combination algebraic compiler came into use (or rather into misuse), the meaning of the term had already shifted into the present one*

—

Bauer and Eikel [1975]

First Definition

A **compiler** is a program that accepts as **input a program text** in a certain language and produces as **output a program text** in another language, while preserving the meaning of that text.

First Definition

A **compiler** is a program that accepts as **input a program text** in a certain language and produces as **output a program text** in another language, while preserving the meaning of that text.

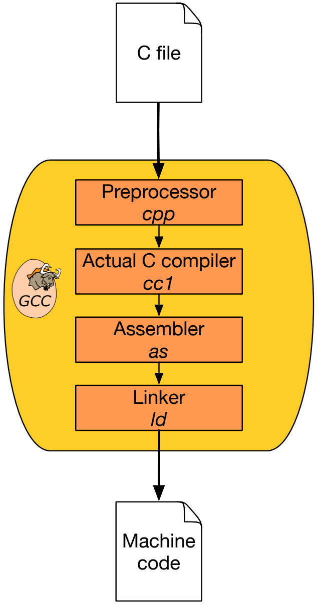
$\Rightarrow$ A translator!

Some vocabulary

- **A transpiler** is a program that converts a source langugae into a target language
  $\Rightarrow$ *Same level of abstraction*

- **A compiler** is a program that converts a source langugae into a target **machine** language
  $\Rightarrow$ *Different level of abstraction*

Coarse-grained steps in GCC



The CPP Preprocessor

A **preprocessor** is a source-to-source transpiler: it *simplifies* the input code and produces pure source code. It applies the following translations:

- Macro expansions
- File expansion
- Conditionnal expressions
- Miscellaneous directives
- Remove comments
- Trigraph conversion

Try it yourself:

```
echo "#include <stdio.h>" | gcc -E -
```

The actual compiler

The hardest part !

This lecture !

Many challenges:

- Lexing, Parsing,
- Type checking
- Linearizing
- SSA
- Register allocation
- Optimisation
- ...

Assembler

An **assembler** translates assembly language programs **into machine code**.

The output of an assembler is called an object file, which contains a combination of machine instructions as well as the data required to place these instructions in memory.

Try it yourself:

```
gcc -S foo.c && as foo.s -o foo.o
```

Linker

A **linker** is a program that links and merges various object files together in order to make an executable file.

The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

Try it yourself:

```
ld  foo.o -o foo -lSystem
```

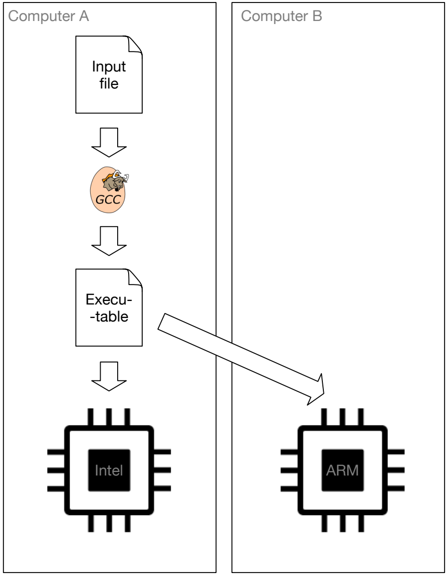
Cross compiler

A **cross-compiler** is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.

Try it yourself:

```
aarch64-linux-gnu-gcc -o main main.c
qemu-aarch64 main
```

Cross compiler



Bootstrapping: the chicken-or-egg problem

Bootstrapping compilers

Bootstrapping is the technique for producing a self-compiling compiler.

Bootstrapping advantages:

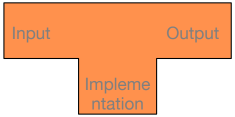
- Developers only need to know and work in one language
- Non-trivial test of the language being compiled
- Improvement of compiler generated code benefits both compiler and users programs

Some Bootstrapped compilers

- C++ *(clang)*
- C *(gcc)*
- Go
- Java (not the runtime)
- Ada *(gnat)*
- Haskell *(ghc)*
- Delphi
- Common Lisp
- Eiffel
- Rust
- Ocaml
- Zig
- Tiger (WIP)
- ...

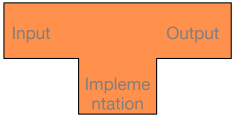
T(ombstone)-Diagrams

T-diagrams are an efficient way to describe a compiler

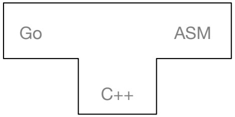


T(ombstone)-Diagrams

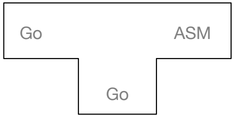
T-diagrams are an efficient way to describe a compiler



Example:



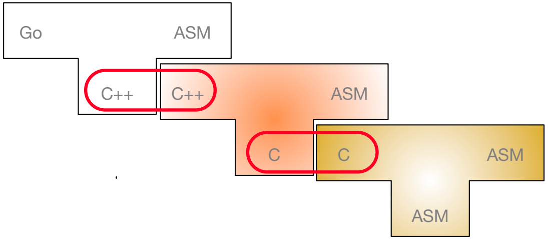
Some Go compiler



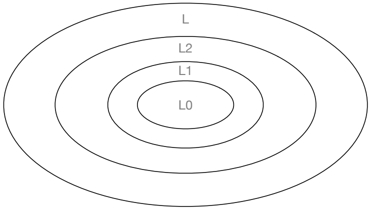
Some bootstrapped Go compiler

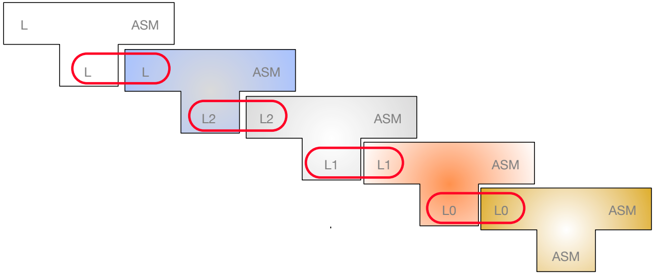
How to build a compiler?

Use other languages to write the compiler.



How to build a compiler?





Summary

Compiler

Assembler

Linker

Bootstraping

Preprocessor

Cross compilation