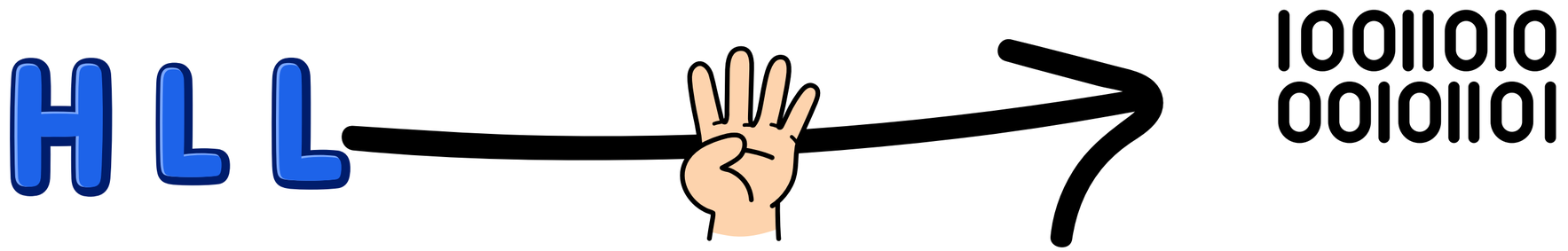
Abhinav Tiwari
osc@compilersutra.com

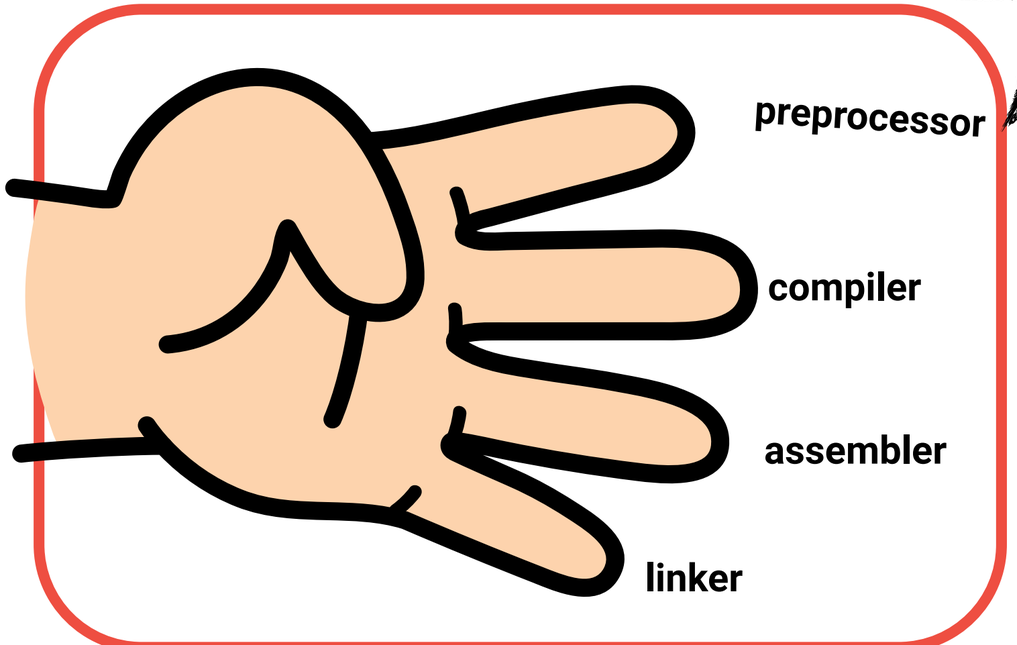
https://www.compilersutra.com

HoW Source Code Converted to Executable



HLL —preprocessor→ HLL.i —compiler→ HLL.s —assembler→ HLL.o —linker→ HLL.elf

TOOLCHAIN



> ***A combination of multiple tools working together in a fixed order to convert source code into a final executable is called a toolchain***

What Is Compiler?

*The compiler takes **HLL.i** and translates it into **assembly***

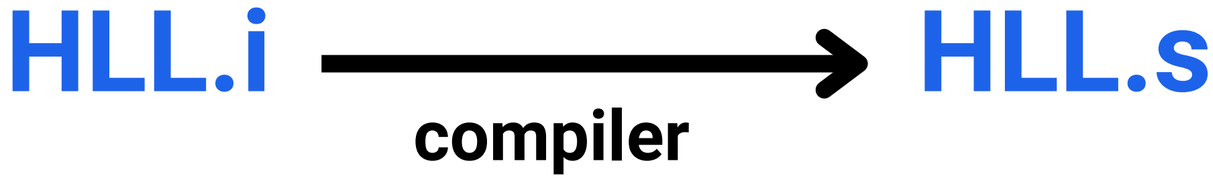
A compiler translates **preprocessed source** code into **semantically equivalent assembly** while performing analysis and optimizations.

Compiler responsibilities:

“

- **Syntax analysis**
- **Semantic analysis**
- **Type checking**
- **Optimizations**
- **Instruction selection**

”


HLL.i
compiler
HLL.s


Front end
Understand the preprocessed file and generate ir
Middle End
Improve the program
by working on IR
Backend
Generate machine code
Works on optimized IR

Front end

Role

👉 **Understand the program and check correctness**

preprocessed file → **Front end** → *Intermediate Representation*

Main Responsibilities

- **Syntax Analysis**
  - **Checks grammar**
  - **Builds AST**
- **Semantic Analysis**
  - **Checks meaning**
  - **Scope & symbol resolution**
- **Type Checking**
  - **Verifies data types**
  - **Detects invalid operations**

What Front End Does NOT Do

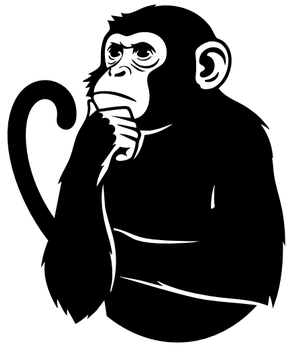
- ❌ **Optimizations**

End Goal

- **Correct, well-typed IR**

> **Front End checks *correctness*, not *performance*.**

How to See IR



- **Generate LLVM IR (Readable)**
    - **clang -S -emit-llvm file.c**

IR is the Front End output used by later compiler stages.

Middle End

Role

👉 **Improve the program without changing its meaning**

IR generated by Front end → **Middle End** → *Optimized IR*

Optimization

- **Improves performance**
- **Removes redundant computation**
- **Preserves program meaning**

What Middle End Does NOT Do

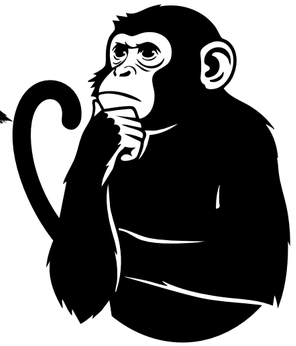
- ❌ **Machine-specific code generation**
- ❌ **Hardware-dependent decisions**

End Goal

- **Efficient, semantically equivalent IR**

> **Middle End *optimizes* the *IR* without changing *correctness*.**

How to See Optimization



- **Using OPT**
  - **opt -O2 -debug-pass-manager test.ll -disable-output**
  - **opt -O2 -print-after-all test.ll -S**
  - **opt -O2 -print-changed test.ll -S**

Middle end optimized the IR which is passed to backend

Backend

Role

👉 **Convert optimized IR into machine code**

IR optimized by middle end → **Backend** —*Machine IR, target dependent optimization*→ **Assembly (.s)**

Optimization

- **Target Dependent Optimization**
- **Lowering & Code Generation**

What Back End Does NOT Do

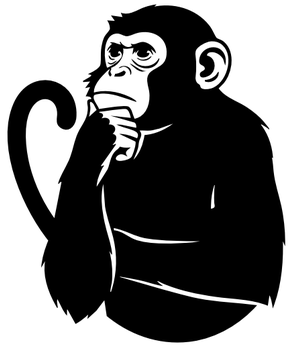
- ❌ Does not convert .s → .o
- ❌ Does not combine object files

End Goal

- **Correct and efficient machine code which can pass to assembler**

> ***Back End maps optimized IR to real hardware.***

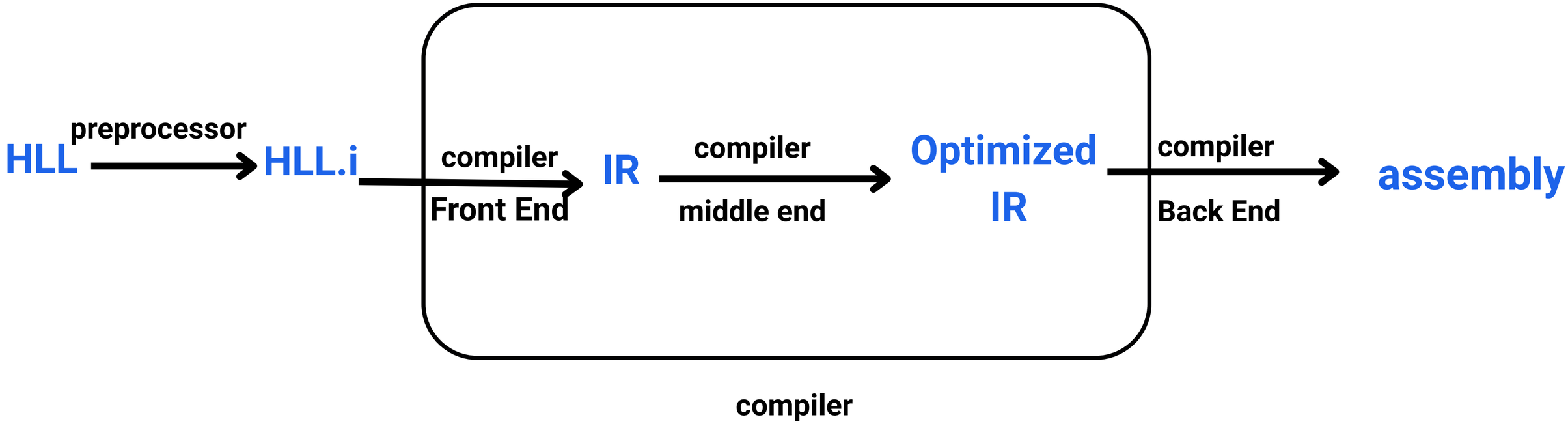
How to See Each Optimization



- **Using LLC**
    - **llc -O2 -print-after-all test.ll -S**
    - **llc -O2 -print-changed test.ll -S**

Back End lowers the optimized IR into target-specific machine code (assembly).

Combine all Together how compiler will work?



Why we need IR?