EECS 483: Compiler Construction

Lecture 0: Course Overview, Baby's First Compiler

January 7, 2026

Introductions

Instructor: Max S. New

GSI: Yuchen Jiang

Programming Language Implementation

When you write a program in your favorite programming language, how does it get executed?

Interpreted

Another program, an "interpreter" reads in your program and implements its behavior

Compiled

Another program, a "compiler" and outputs another program, which we already know how to run. Most commonly, the binary format that your CPU executes.

Just-in-time compilation (Javascript v8): an interpreter that selectively compiles part of the program to speed up execution

Bytecode interpreters (Java): compile to a language for which you have a fast interpreter implemented.

Fundamentally, a compiler is a kind of **translator**

```
compiler : SourceProgram -> TargetProgram
```

Usually the source programs are easy to write, and the target programs are easy to use.

```
gcc, clang   : C/C++      -> Binary          /* a.out, .exe */
emcc         : C/C++      -> WebAssembly
rustc        : Rust       -> Binary
javac        : Java       -> JvmByteCode     /* .class */
scalac       : Scala      -> JvmByteCode
gwt          : Java       -> JavaScript      /* .js */
v8           : JavaScript -> Binary
nasm         : X64        -> Binary
pdftex       : LaTeX      -> PDF
pandoc       : Markdown   -> PDF or Html or Doc
```

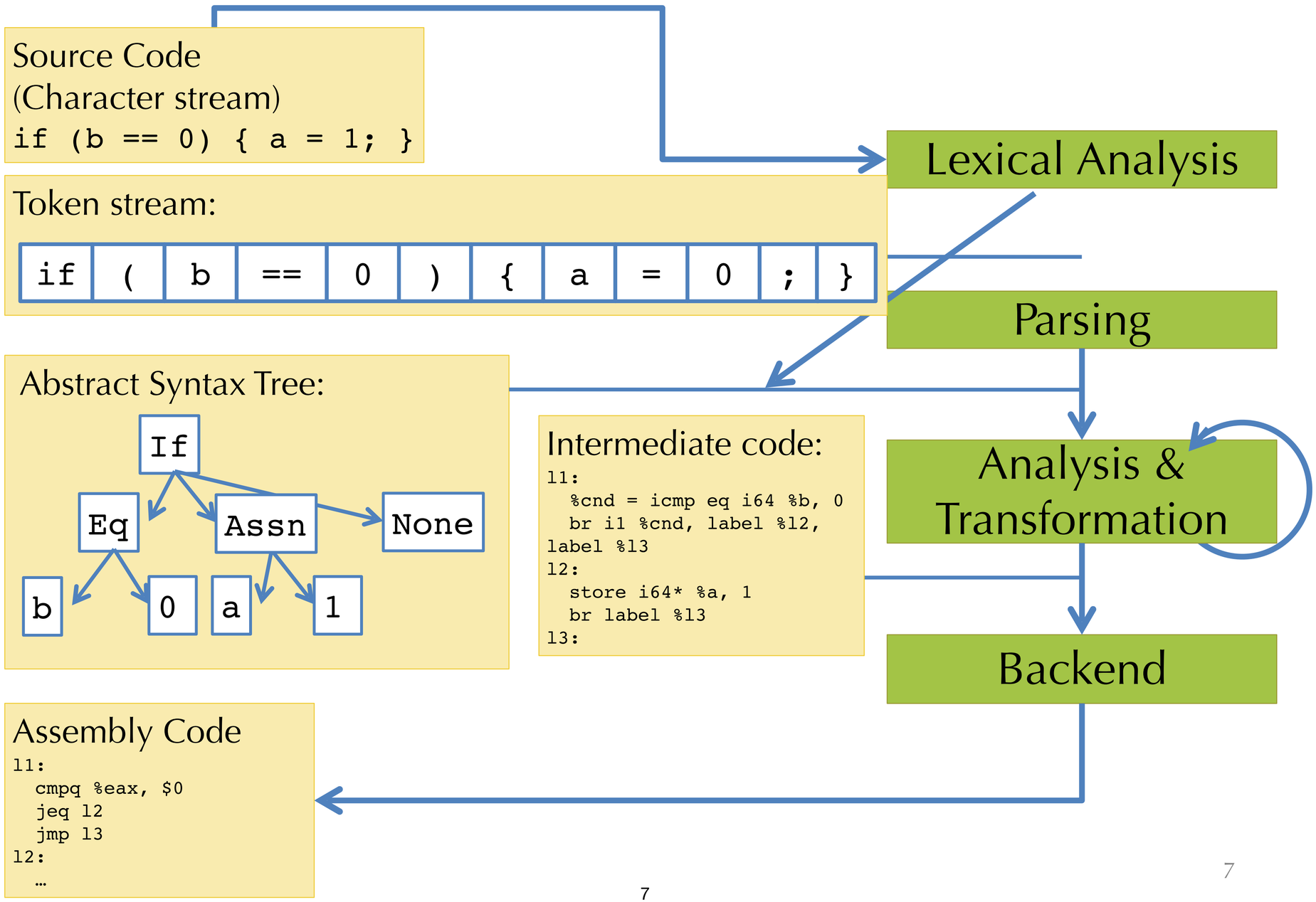
Why Study Compilers?

- Compilers are **software infrastructure** programmers interact with every day. Understanding them gives you a deeper understanding of how your programs are executed.
- Give you the tools to **design** and **implement** your own programming languages.
- Learn techniques like lexing/parsing, constraint solving, tree data types that can be applied to other programming tasks.
- Build up a medium-sized code base with complex but precise specifications.

Learning Objectives

- What does it mean for a compiler to be correct?
- How are high-level programming features implemented in machine code?
- How are modern compilers architected?
- How should programs be represented as data?
- How can linear structures be parsed into tree-like data?
- How to analyze programs to produce more efficient code?
- How to design, implement, grow and test a codebase over time?

What do compilers look like on the inside?



7

Course Topics Outline

- Part 1: Interpreters, Semantics, Basic Translation to Machine Code
- Part 2: SSA Intermediate Representation
- Part 3: Memory Management
- Part 4: Program Analysis and Optimization
- Part 5: Compiler Frontends (Lexing and Parsing)

Programming Assignments

Over the course of 5 homework assignments, we will build up to an optimizing compiler from the "Snake" language to SSA form to x86_64 assembly code.

We will build up features of the Snake language over time. Start today with almost nothing and build up to a reasonable dynamically typed, functionalish programming language, a simplified version of Python, JavaScript or Scheme.

Languages

We will be working with many programming languages in this course. You are not required to have had direct experience with any of them:

1. Rust for the compiler and runtime system
2. x86 Assembly code, our target
3. SSA, our intermediate representation
4. Snake lang, our source language
5. Regular expressions, context-free grammars, for the front-end.

Use these first couple of weeks to get up to speed on Rust. Other languages will be introduced over the course of the semester.

Course Webpage

Full syllabus and all the details here are on my personal webpage:
https://maxsnew.com/teaching/eecs-483-wn26

Evaluation

Your final grade in this course will be based on your performance in three areas:

- 65% Programming Assignments, each weighted equally
- 30-35% exams, midterm and final, each weighted equally
- 0-5% attendance and participation

Attendance and Participation

- Attendance and active engagement in class are very helpful to your understanding of the material.
- Attendance policy: attendance is incentivized but not necessary.
  - We will have a sign in sheet and we will note who participates in class.
  - You will get a participation score of 0-5 at the end of the course.
  - Each participation point will count as one percentage point towards your final grade, **replacing a percentage point from the exams**.
  - If you get a 0 participation score, you can still get a perfect grade in the class if you ace the exams and homeworks.

Homework Assignments

Completed solo or in groups of 2. Do not collaborate with anyone unless you are in a group together.

Autograded on gradescope with some public and some hidden test cases. Unlimited submissions.

Homeworks due on Friday with late submissions accepted for 2 days afterwards at a 10% penalty per day.

Exams

Midterm exam: programming language basics, scope, SSA form, x86 assembly

Final exam: second half of the course. Data representations, optimization, compiler frontend

Exams are complementary to programming assignments. Programming assignments demonstrate practical knowledge while exams demonstrate theoretical understanding.

Baby's First Compiler

When we implement a compiler (to assembly) we need to address the following questions:

1. What is the syntax of the language we are compiling?
2. What is the semantics of the language we are compiling?
3. How can we implement that semantics in assembly code?
4. How can we generate that assembly code programmatically?

Snake v0: "Neonate"

1. What is the syntax of the language we are compiling?

   Integer literals

2. What is the semantics of the language we are compiling?

   Print out the number to stdout

3. How can we implement that semantics in assembly code?

   Produce a function that outputs that number, call that function from Rust.

4. How can we generate that assembly code programmatically?

For Next Time

- Read the course webpage: http://maxsnew.com/teaching/eecs-483-wn25/
- Install Rust, nasm, run today's neonate implementation locally
- Work through the first 4 chapters of the Rust book: https://rust-book.cs.brown.edu/
- Attend discussion section on Friday for more intro to Rust