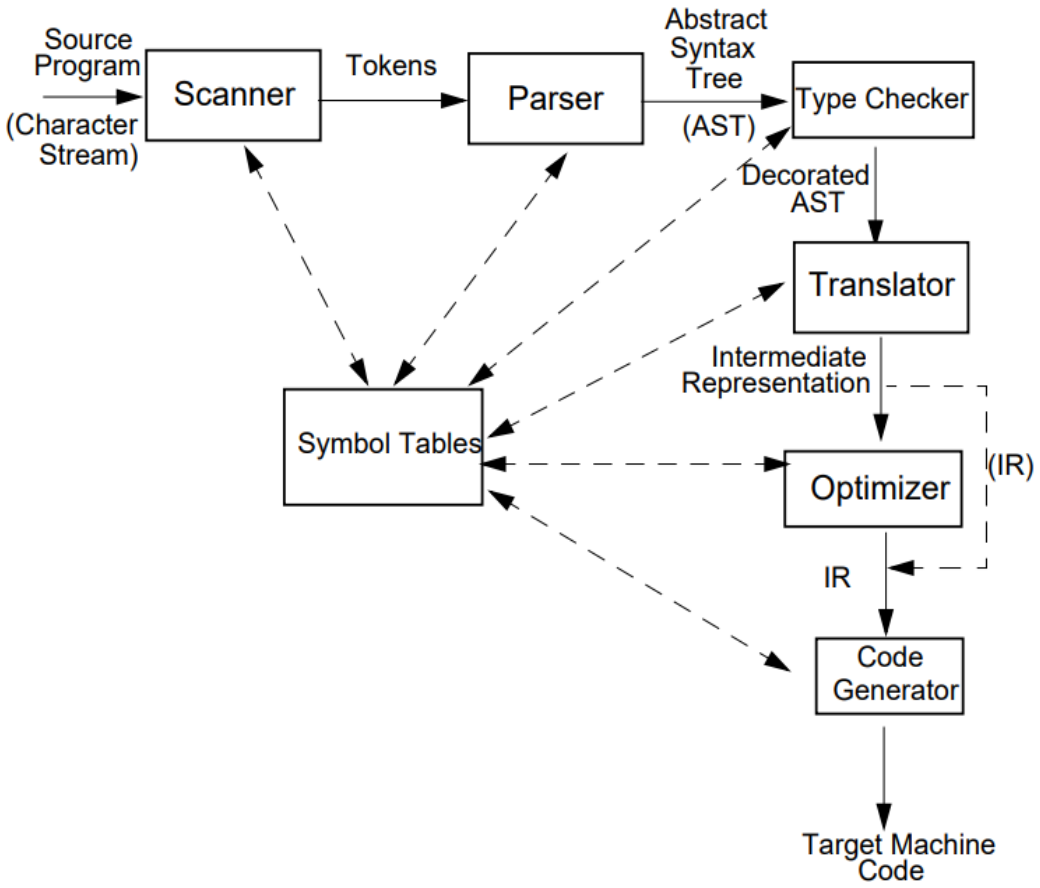

Scanner

掃描儀逐個字符地讀取源程序。它將單個字符分組為標記（標識符、整數、保留字、分隔符等）。必要時，還傳遞包含令牌的實際字符串以供語義階段使用。

掃描儀執行以下操作：

- 它將程序置於緊湊和統一的格式（令牌流）。
- 它消除了不需要的信息(such as comments)。
- 它有時會在符號表中輸入初步信息（例如 29 CS 536 Spring 2006©，以記錄特定標籤或標識符的存在）。
- 它可以選擇格式化並列出源程序

構建 tokens 由使用 regular expression notation 的 token 描述驅動。

Regular expressions 是一種能夠描述現代編程中使用的標記的正式符號語言。此外，僅給定 tokens 規範，它們就可以驅動工作掃描儀的自動生成。掃描儀生成器（如 Lex、Flex 和 Jlex）很有價值的編譯器構建工具

Parser

給定語法規範（as a context-free grammar, CFG)），解析器讀取標記並將它們分組到語言結構中。

解析器通常是使用解析器生成器（如 Yacc、Bison 或 Java CUP）從 CFG 創建的。

解析器驗證正確的語法並可能發出語法錯誤消息。

當句法結構被識別時，解析器通常構建 abstract syntax tree (AST)，這是程序結構的簡潔表示，它指導語義處理。

Type Checker (Semantic Analysis)

類型檢查器檢查每個 AST 節點的靜態語義。它驗證構造是否合法且有意義（所有涉及的標識符都已聲明，類型是否正確，等等）。

如果構造在語義上是正確的，則類型檢查器“裝飾”AST 節點，向其添加類型或符號表信息。如果發現語義錯誤，則會發出適當的錯誤消息。

類型檢查完全依賴於源語言的語義規則。它獨立於編譯器的目標機器。

Translator (Program Synthesis)

如果一個 AST 節點在語義上是正確的，它就可以被翻譯。翻譯涉及捕獲構造的運行時“含義”。

例如，while 循環的 AST 包含兩個子樹，一個用於循環的控製表達式，另一個用於循環的主體。 AST 中沒有任何內容顯示 while 循環循環！當 while 循環的 AST 被翻譯時，就會捕捉到這個“意義”。
在 IR 中，測試循環控製表達式的值和有條件地執行循環體的概念變得明確。

Optimizer

它轉換代碼，使其消耗更少的資源並產生更快的速度。被轉換的代碼的含義不會改變。優化可以分為兩種類型：機器相關和機器無關。

Code Generator

代碼生成是編譯過程的最後階段。它將優化的中間代碼作為輸入並將其映射到目標機器語言。代碼生成器將中間代碼翻譯成指定計算機的機器代碼。

Symbol Tables

它是一種由編譯器使用和維護的數據結構，由所有標識符的名稱及其類型組成。它通過快速找到標識符來幫助編譯器順利運行。

源程序的分析主要分為三個階段。他們是：

1. 線性分析
   這涉及掃描階段，其中字符流從左到右讀取。然後將其分組為具有集體意義的各種標記。
2. 層次分析
   在這個分析階段，基於集體意義，令牌被分層分類為嵌套組。
3. 語義分析
   此階段用於檢查源程序的組件是否有意義。