Concepts of programing Languages

Lecture 2 : Language Translation

Sudan University of Science and Technology

Dr. Aghabi Nabil Abosaif

13/10/2021

Lecture Contents

- Influences on Language Design.
  - Von Neumann Architecture.
  - Imperative languages.
  - Programming Design Methodologies History.
- Implementation Characteristics of languages.
  - Compilation.
  - Pure Interpretation.
  - Hybrid Implementation.
- Preprocessors.
- Integrated Development Environments

Influences on Language Design

1. Computer **Architecture**
   - Architecture as **driving factor** of language design.
   - E.g. Von Neumann Architecture.

2. Programming Design **Methodologies**
   - Abstract data types.
   - Concept of object orientation

3. **Computational** models / **Mathematical** models for computation
   - Lambda Calculus, Predicate Logic.

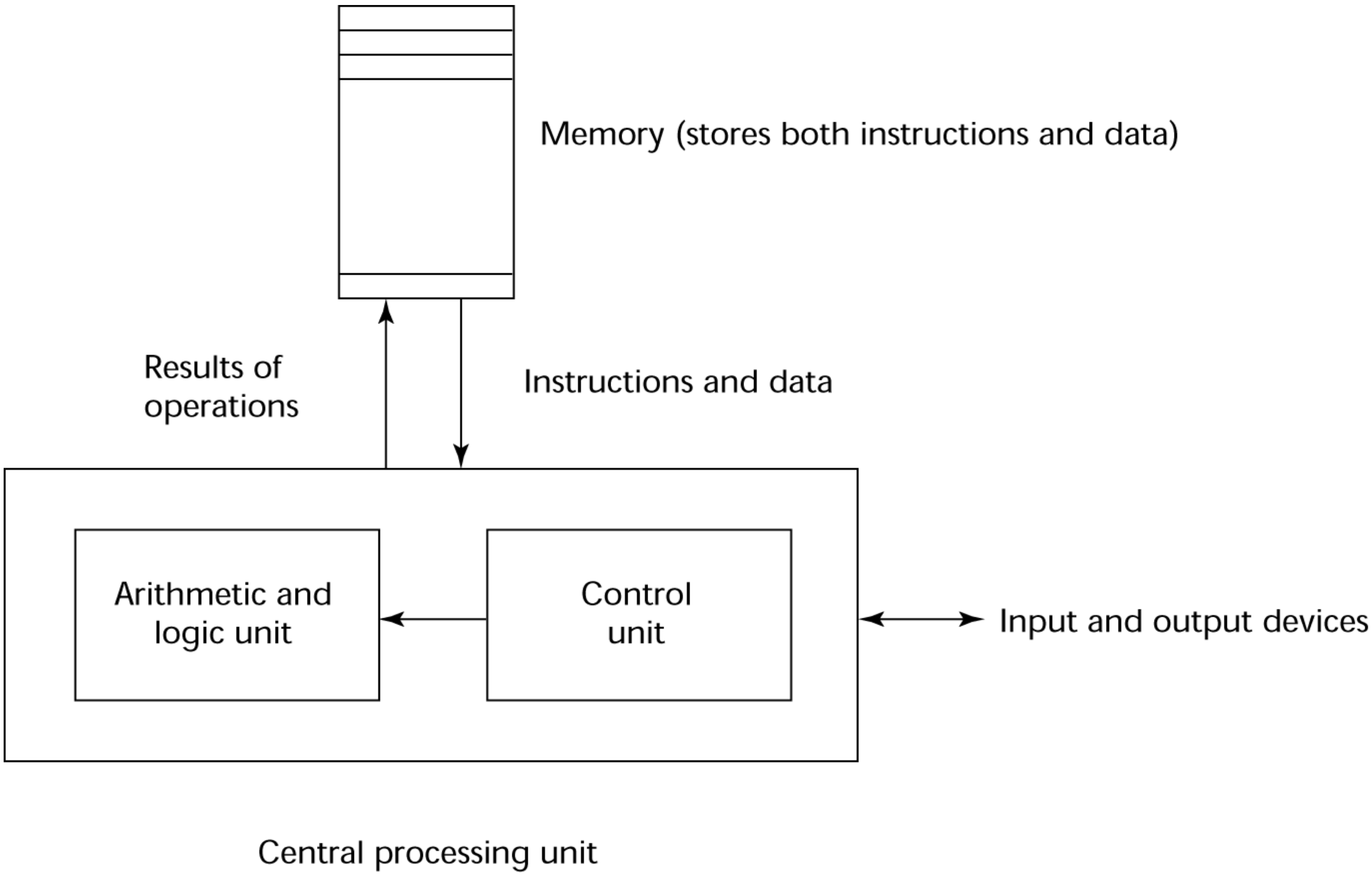
Computer Architecture
Von Neumann Architecture

- The basic architecture of computers has had a profound effect on language design.
- Most of the **popular** languages of the past 60 years have been **designed around** the **prevalent** computer architecture, called the **Von Neumann architecture**.
- One of its initiators, *John Von Neumann.*
- These languages are called **imperative languages.**

Von Neumann Architecture (Continuous)

- In this architecture, both **data** and **programs** are stored in **the same memory.**
- The CPU, which **executes instructions**, is **separate** from the memory.
- Therefore, **instructions and data** must **be transmitted**, or **piped**, from memory to the CPU.
- **Results of operations** in the CPU must be **moved back** to memory.
- Nearly all digital computers built since the 1940s have been based on the Von Neumann architecture.

Von Neumann Architecture



Von Neumann Architecture (Continuous)

- The **execution of a machine code** program on a Von Neumann architecture computer occurs in a process called the **fetch-execute cycle**.

- **Programs** reside in **memory** but are **executed** in the **CPU**.

- Each instruction to be executed must be ***moved from memory to the processor***.

- The address of the next instruction to be executed is maintained in a register called the **program counter**.

Imperative languages

- Inspired from the **von Neumann architecture**, the central features of imperative languages are:
  - **Variables** model **memory cells.**
  - **Assignment statements** used for assigning values to memory cells, which are based on the piping operation.
  - **Iteration form** of repetition, represents central concept to implement repetition on this architecture.
  - **Operands** in expressions are piped from memory to the CPU,
  - The **result of evaluating** the expression is piped back to the memory cell represented by the left side of the assignment

Initialize Program Counter

Iteration is **fast on von Neumann** computers **<u>because</u> instructions** are stored in **adjacent cells** of memory and repeating the execution of a section of code requires only a branch instruction.

```
initialize the program counter
repeat forever
    fetch the instruction pointed to by the program counter
    increment the program counter to point at the next instruction
    decode the instruction
    execute the instruction
end repeat
```

Programming Design Methodologies History / Mainstream developments

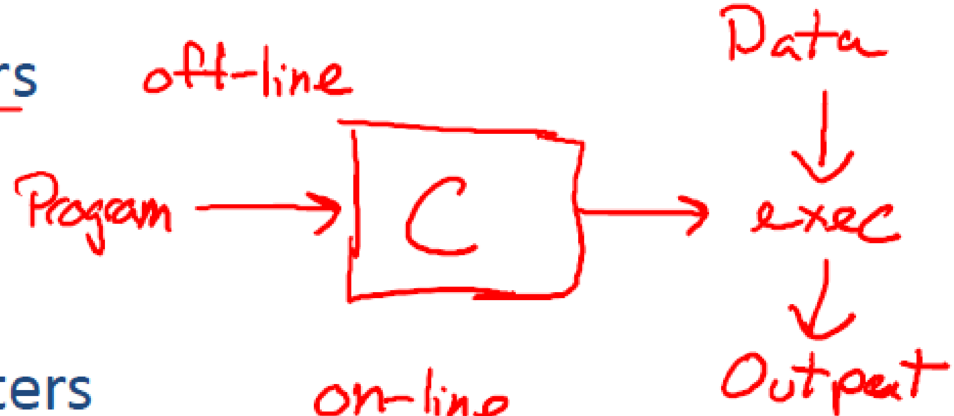
- 1950s and early 1960s: Simple applications; worry about **machine efficiency**.
- Late 1960s: **People efficiency** became more important; readability, better control structures
  - structured programming
  - top-down design and step-wise refinement
- Late 1970s: **Process-oriented** to data-oriented
  - data abstraction
- Middle 1980s: **Object-oriented programming**
  - Data abstraction + **Inheritance** + **Polymorphism**
  - Appearance of C++, C#,java …

Implementation Characteristics of languages

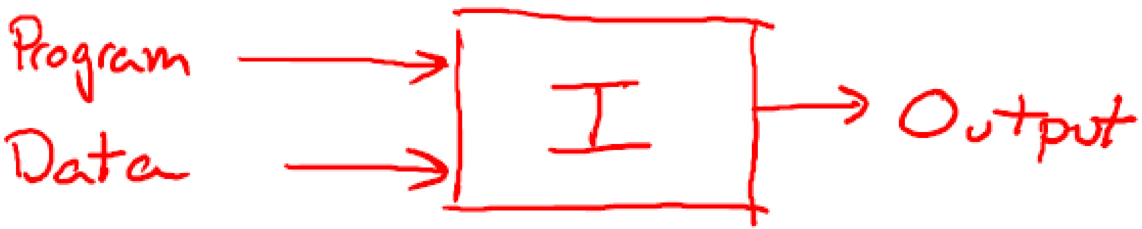
- **Compilation**
  - Programs are **translated directly** into machine language.
- Pure **Interpretation**
  - Programs are **interpreted by another program** known as an interpreter.
- **Hybrid** Implementations
  - A **compromise** between compilers and pure interpreters.

Compilers VS Interpreters

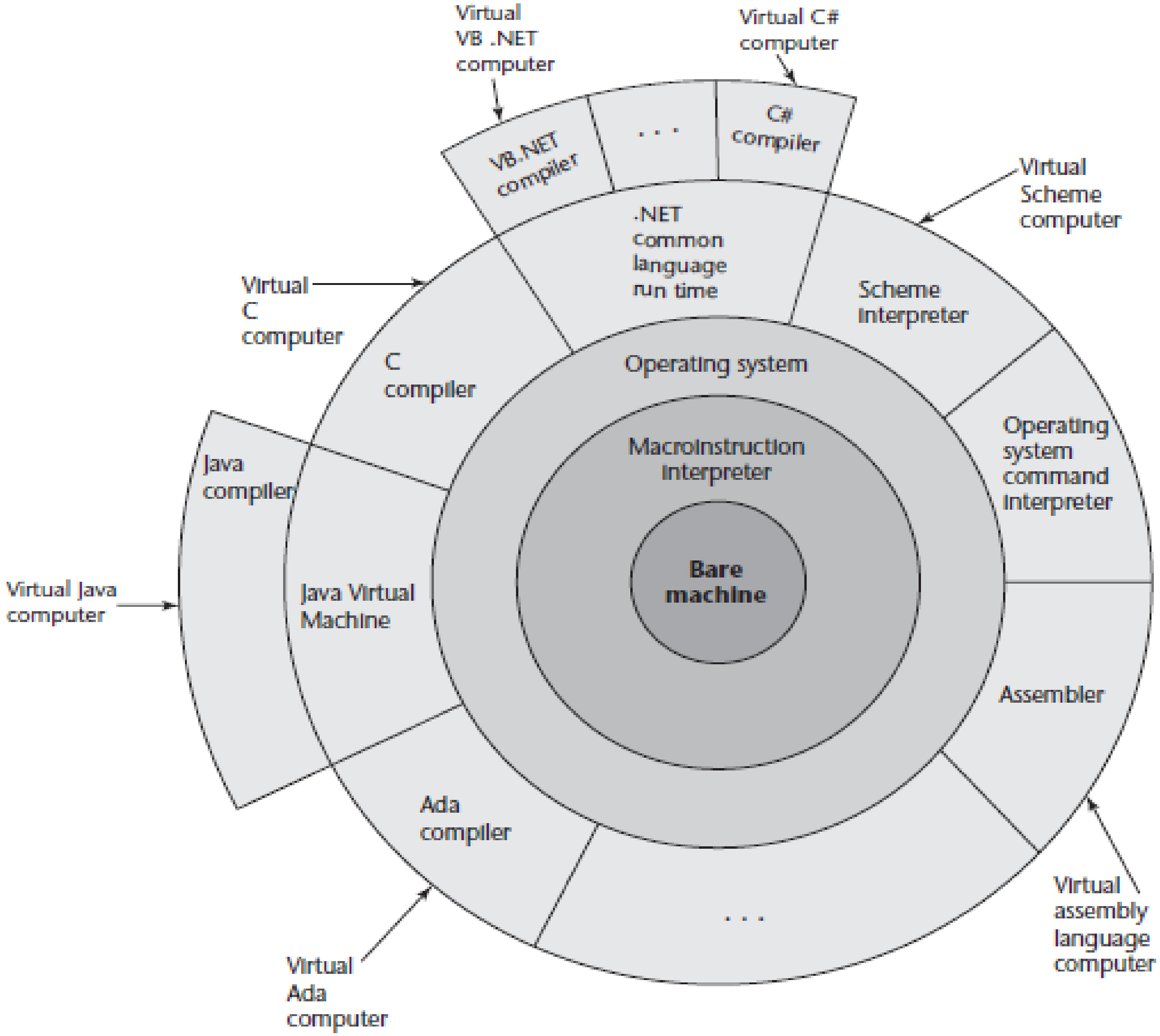
- Compilers off-line



- Interpreters on-line



Layered interface of virtual computers, provided by a typical computer system



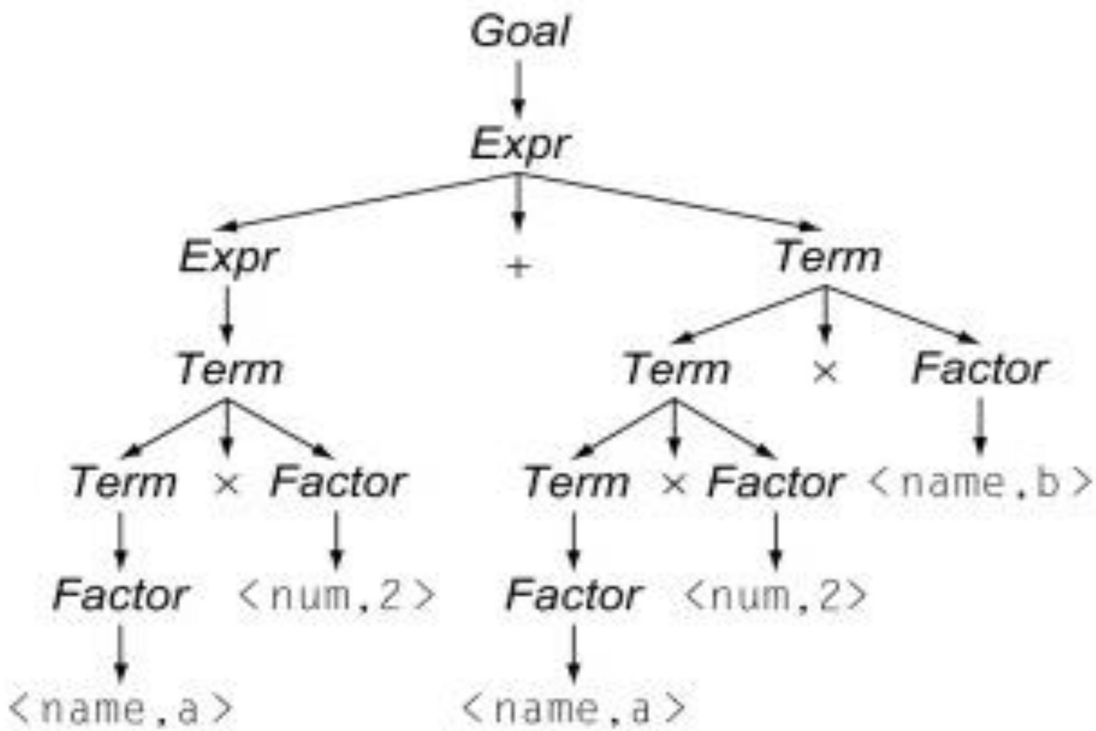
Compilation

- Translate **high-level program** (source code) into **machine code** (executable file) Slow translation, fast execution.

- Phases of compilation process:
  1. **Lexical analysis**: converts characters in the source program into lexical units (identifiers, special words, operators, and punctuation symbols).

  2. **Parsing / Syntax analysis**: transforms lexical units into *parse trees* which represent the syntactic structure of program

Parse Tree Example



Parse Tree for a × 2 + a × 2 × b

Phases of compilation process(Cons.)

3. The **intermediate code generator**: produces a program in a different language, at an intermediate level between the **source program** and the **machine language program**.

It sometimes look very much like **assembly languages**, and in fact, sometimes are actual assembly languages.

4. **Semantics analysis**: is an integral part of the intermediate code generator. It checks for errors, such as type errors, that are difficult, if not impossible, to detect during syntax analysis.

Phases of compilation process(Cons.)

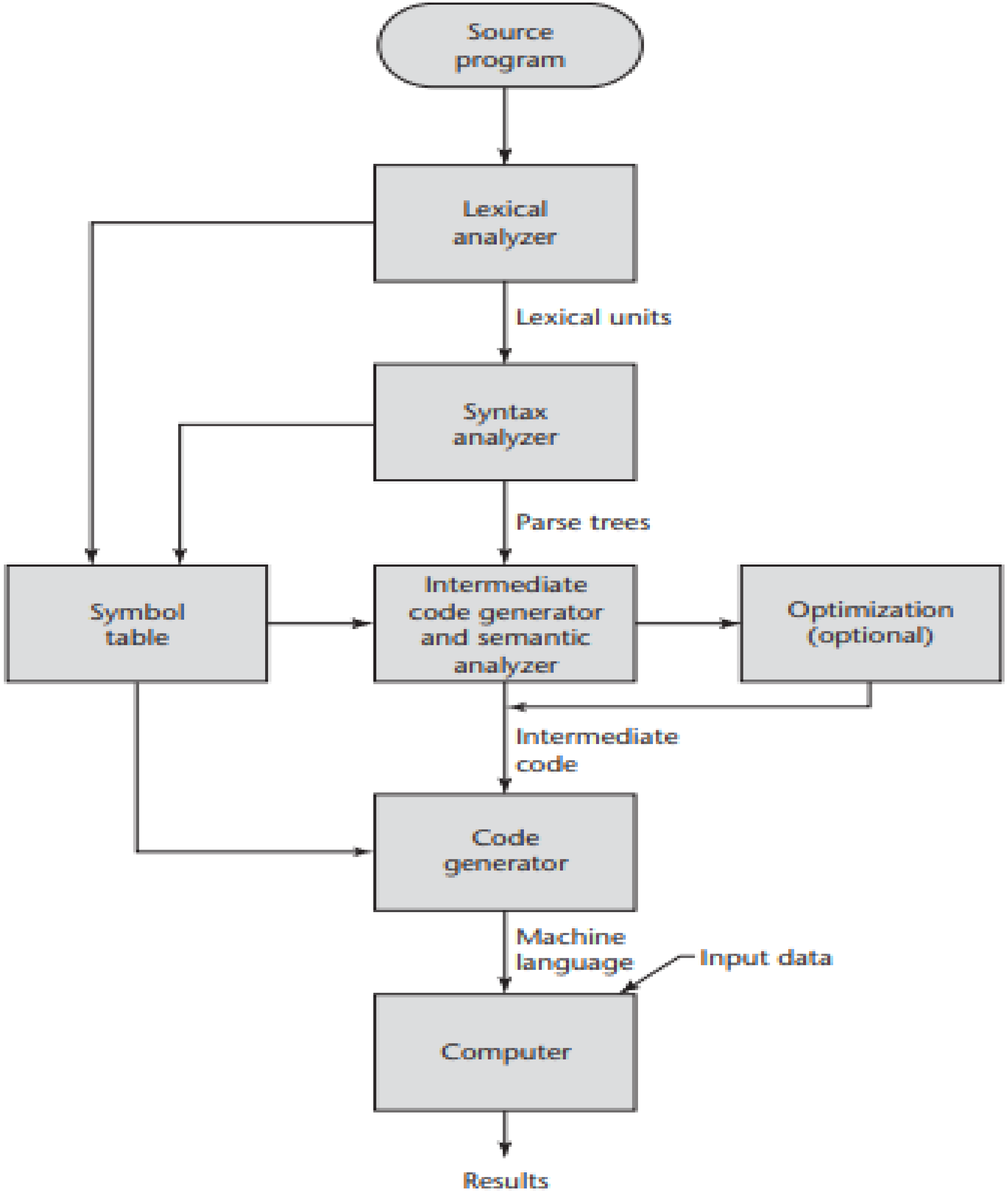
5. **Optimization**: which improves programs (usually in their intermediate code version) by making them smaller or faster or both.

Because many kinds of optimization are difficult to do on machine language, most optimization is done on the intermediate code.

6. **Code generation**: machine code is generated.

7. **Linking**: the process of collecting "objects files" for creating an executable file as output.

The compilation process



Compilation Processes-FORTRAN
1.Lexical Analysis

- First step: recognize words.
  - Smallest unit above letters

This is a sentence.

ist his ase nte nce

Alex Aiken

Compilation Processes- FORTRAN
1.Lexical Analysis

- Lexical analysis divides program text into “words” or “tokens”

if x == y then z = 1; else z = 2;

Alex Aiken

Compilation Processes- FORTRAN
2.Parsing

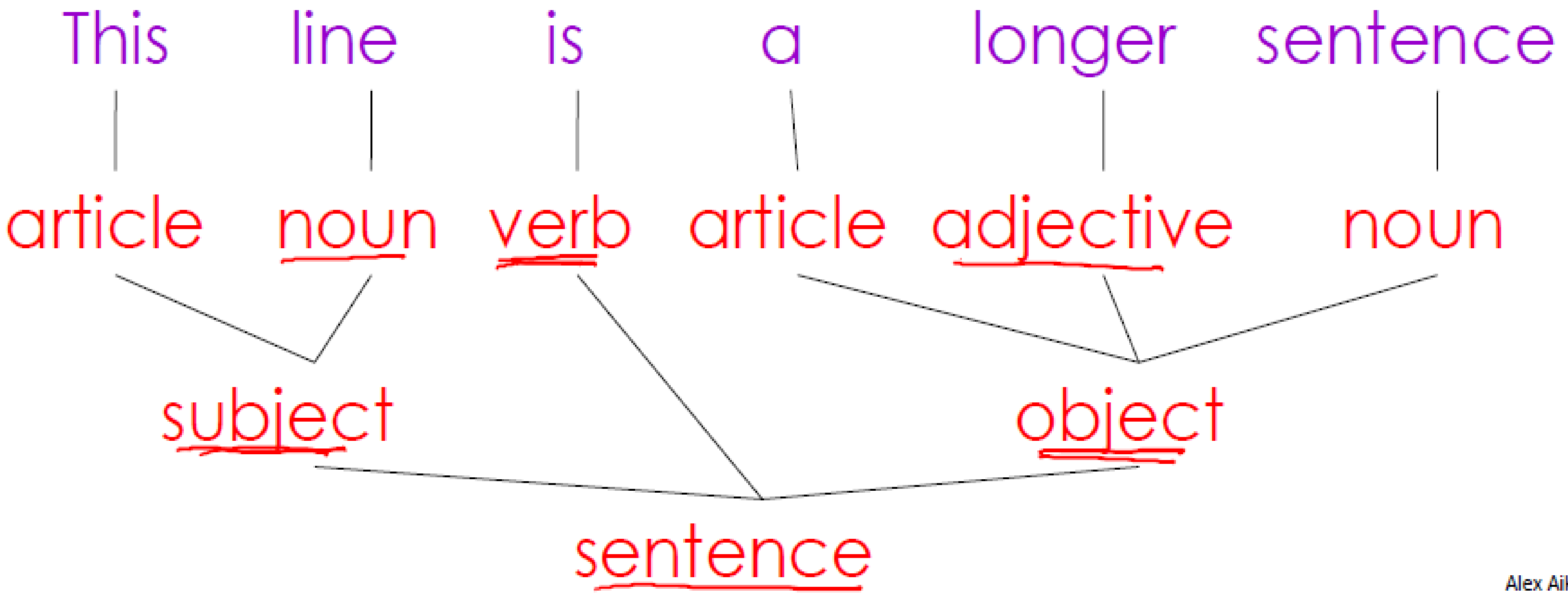
- Once words are understood, the next step is to understand sentence structure

- Parsing = Diagramming Sentences
  - The diagram is a tree

Alex Aiken

- Putting sentence in higher level structure

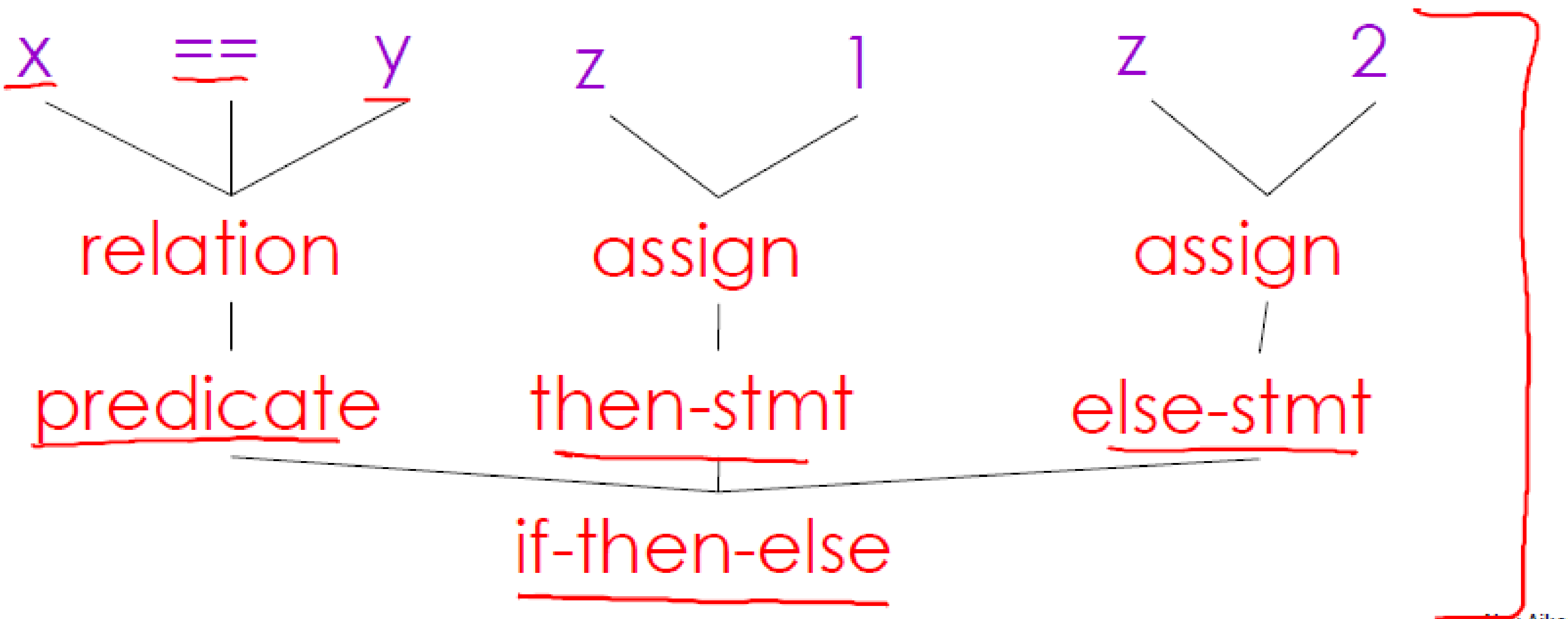
Compilation Processes- FORTRAN 2.Parsing



Alex Aiken

Compilation Processes- FORTRAN
2.Parsing

if x == y then z = 1; else z = 2;



Compilation Processes- FORTRAN
3. Semantic Analysis

- Once sentence structure is understood, we can try to understand "meaning"
  - This is too hard!
- Compilers perform limited semantic analysis to catch inconsistencies

Compilation Processes- FORTRAN
3. Semantic Analysis

- Example:

  Jack said Jerry left his assignment at home.

- Even worse:

  Jack said Jack left his assignment at home?

Compilation Processes- FORTRAN
3. Semantic Analysis

- Programming languages define strict rules to avoid such ambiguities

```
{
    int Jack = 3;
    {
        int Jack = 4;
        cout << Jack;
    }
}
```

Alex Aiken

- Compilers perform many semantic checks besides variable bindings

Compilation Processes- FORTRAN
4. Optimization

- Optimization has no strong counterpart in English
  - But a ~~little bit like~~ akin to editing
- Automatically modify programs so that they
  - Run faster
  - Use less memory

Compilation Processes- FORTRAN
4. Optimization

X = Y * 0 is the same as X = 0

NO!

NAN * 0 = NAN

valid for integers

invalid for FP

Compilation Processes- FORTRAN 5. Code Generation

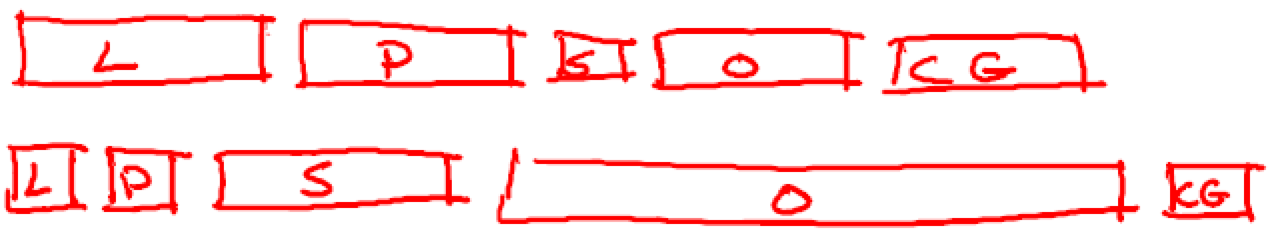
Code Gen

- Produces assembly code (usually)
- A translation into another language
  - Analogous to human translation

Compilation Processes- FORTRAN
5. Code Generation

- The overall structure of almost every compiler adheres to our outline

- The proportions have changed since FORTRAN



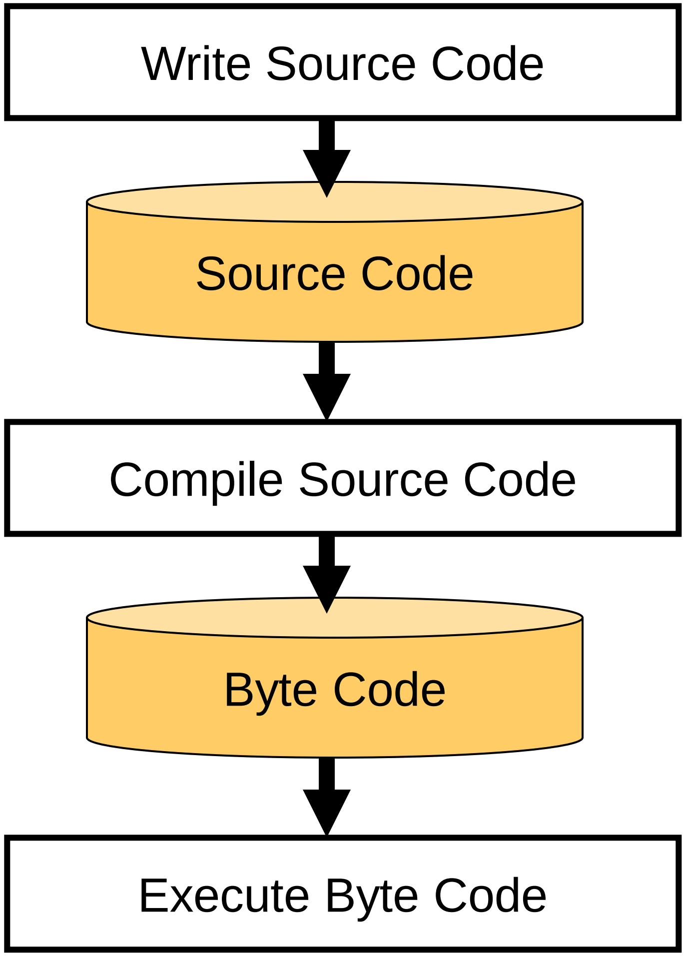
Pure Interpretation

- **Advantages**
  - Online Translation
  - No translation/compilation required.
- **Disadvantages**
  - Errors are recognized during runtime
  - Slow execution speed (10 to 100 times slower than compiled programs).
  - No static type-check, because of the absence of compilation.
- Significant in the area of Web scripting languages (e.g. JavaScript, PHP)

Hybrid Implementation

- A **compromise** between compilers and pure interpreters.
- A program code is **first translated to an intermediate code (called byte code)** for later execution **on a virtual machine.**
- **Faster** than pure interpretation.
- More portable than compiled code
- Examples : Java, C#

Hybrid Implementations



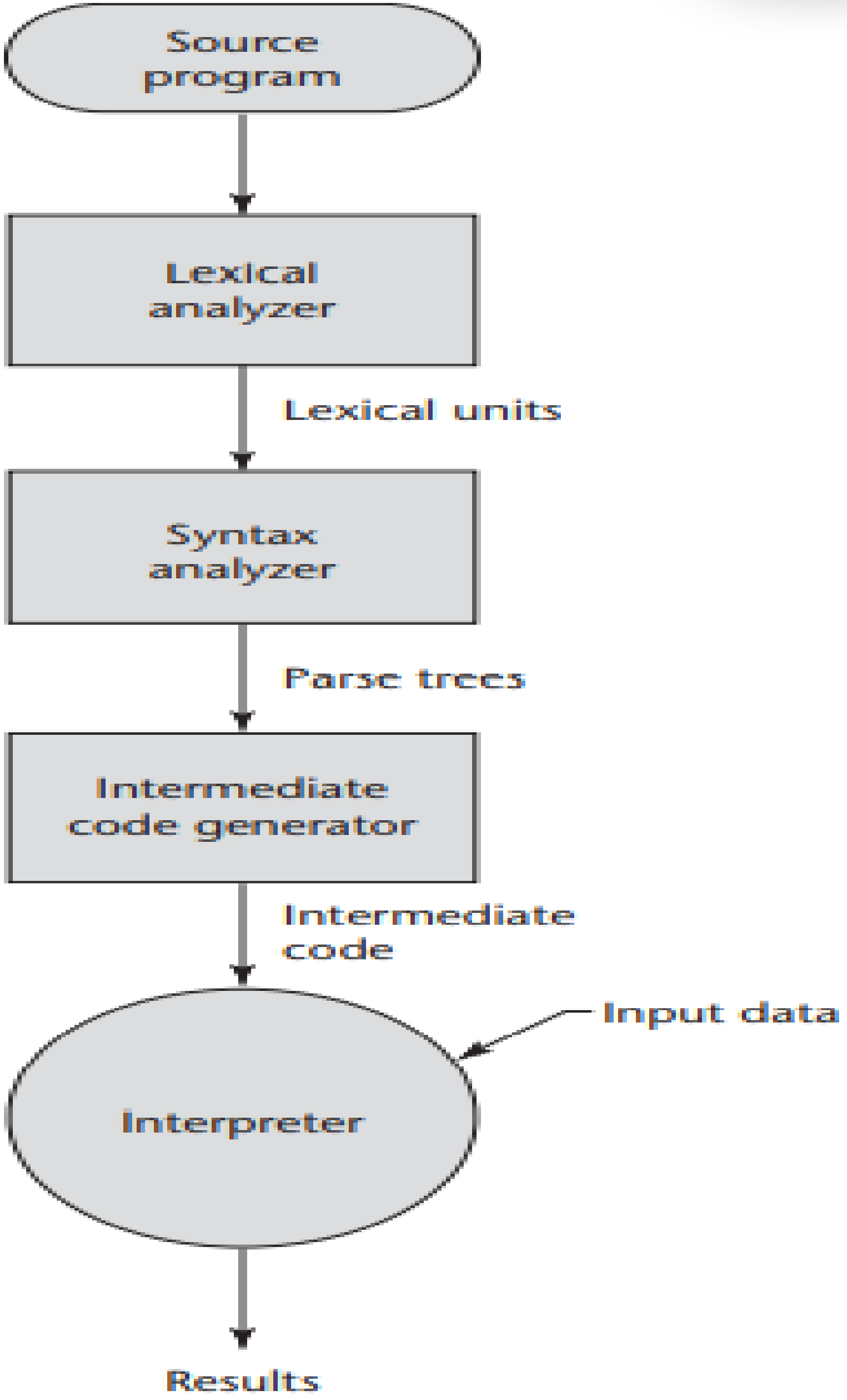
program text in human readable form

all 5 steps of compilation

represents an intermediate code

using a byte-code interpreter

Hybrid implementation system



Just-in-Time Compilation

- Your First Assignment

Thank You