Assembly Programming

第一周 汇编语言简介

庞彦

yanpang@gzhu.edu.cn

Fall 2023

Instructor Information

授课教师: 庞彦 副教授

课程名字: 汇编语言与接口技术

课程编号: (2023-2024-1)-210600058-1

上课地点: 理科南110阶

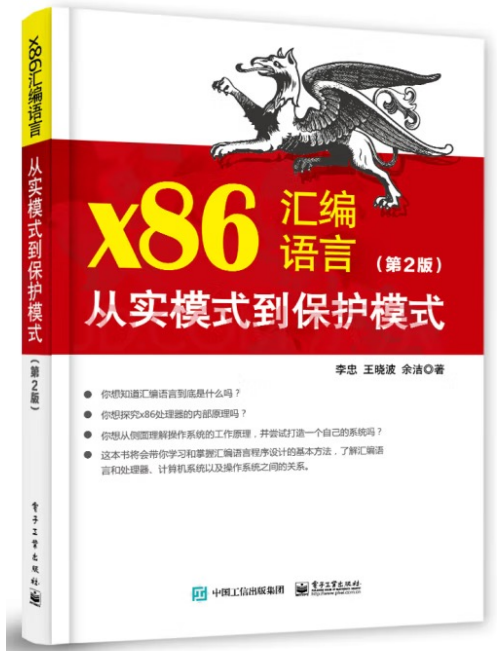
上课时间: 15:45 PM ~ 17:20 PM F

办公地点: 黄埔研究院 B4-808

电子邮件: yanpang@gzhu.edu.cn

个人网站: https://pangyan.me/

Textbooks

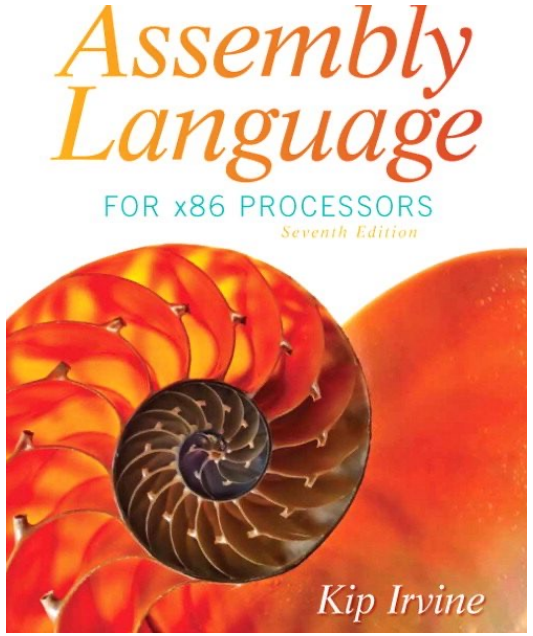


x86汇编语言：从实模式到保护模式（第2版）

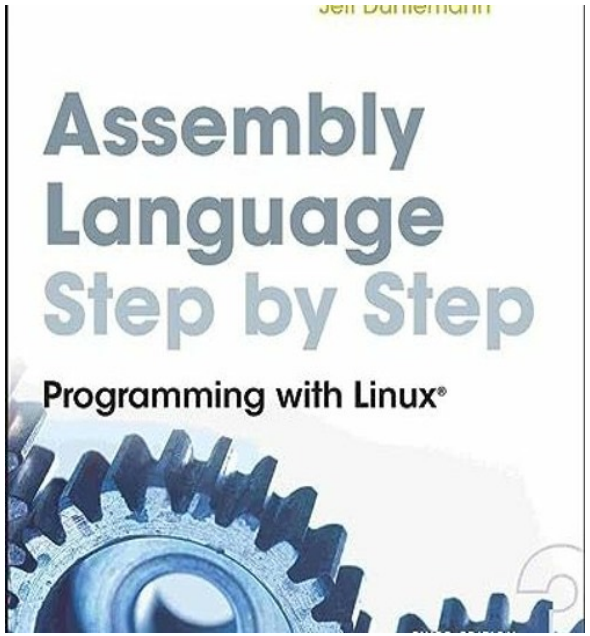
Link 1

Link 2

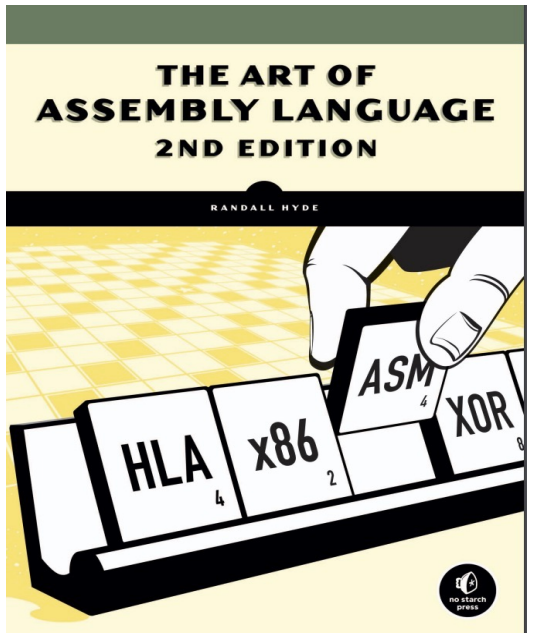
Reference



Link 1



Link 2



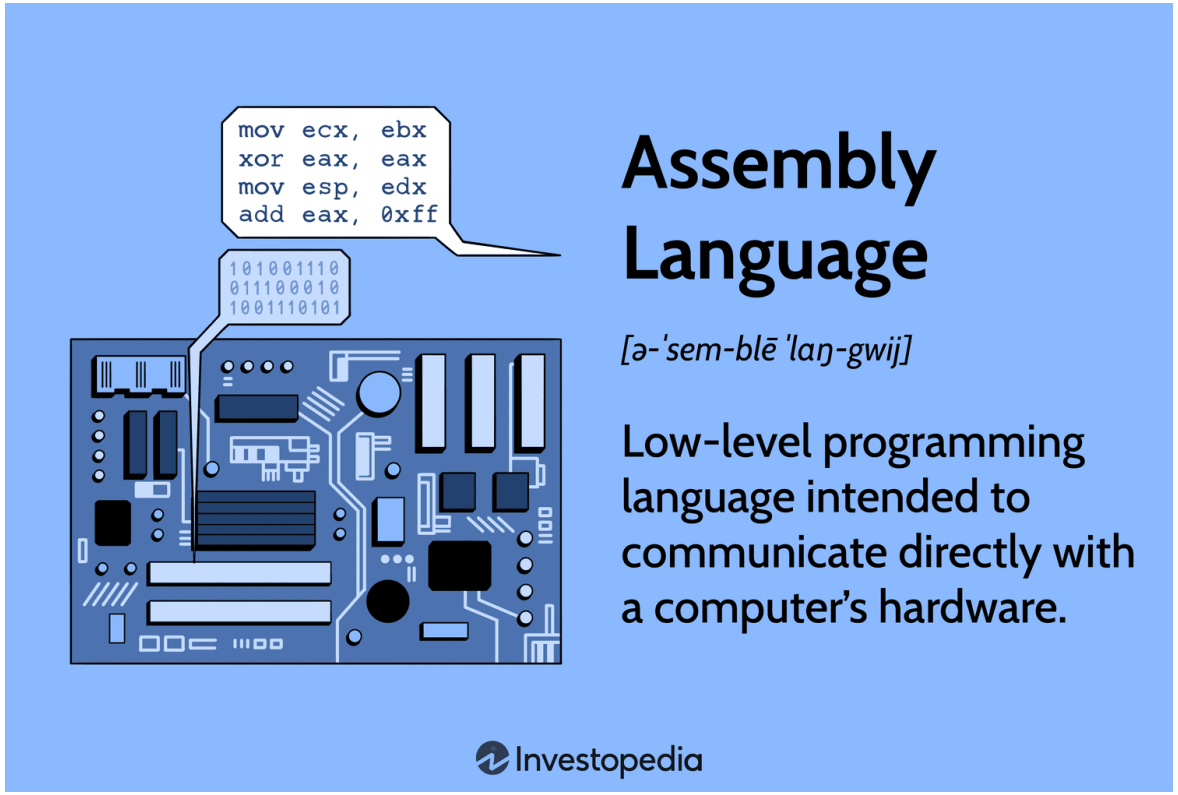
Link 3

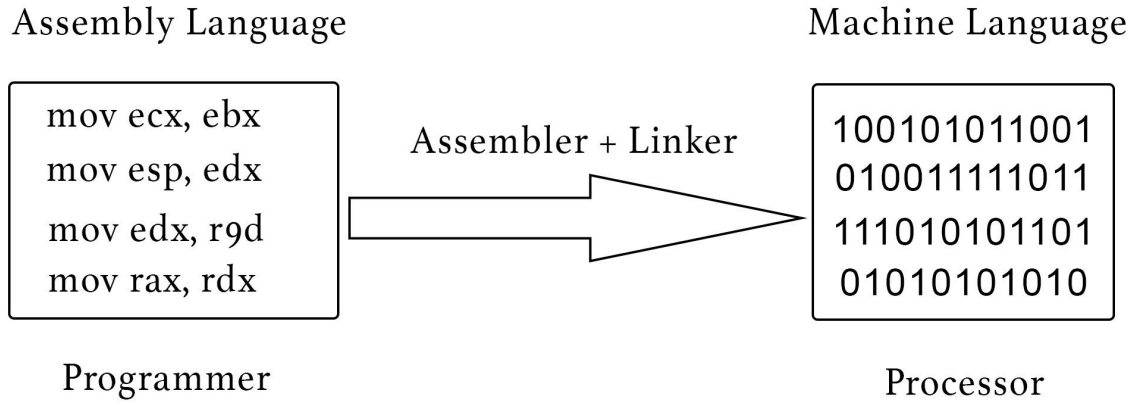
Software

https://visualstudio.microsoft.com/

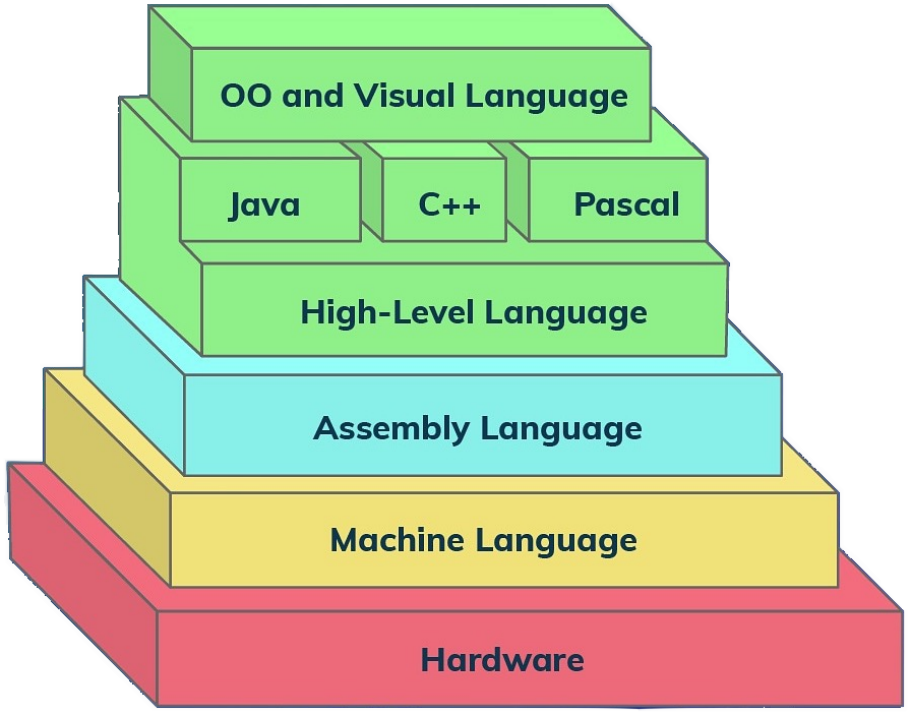
100s

Assembly Programming





Programming Language



An **assembly language** is a type of low-level programming language that is intended to communicate directly with a computer's hardware.

Unlike machine language, which consists of binary and hexadecimal characters, assembly languages are designed to be readable by humans.

Low-level programming languages such as assembly language are a necessary bridge between the underlying hardware of a computer and the higher-level programming languages—such as Python or JavaScript—in which modern software programs are written.

C vs Assembly

How Do C++ and Java Relate to Assembly Language? High-level languages such as Python, C++, and Java have a *one-to-many* relationship with assembly language and machine language. A single statement in C++, for example, expands into multiple assembly language or machine instructions. Most people cannot read raw machine code, so in this book, we examine its closest relative, assembly language. For example, the following C++ code carries out two arithmetic operations and assigns the result to a variable. Assume X and Y are integers:

```
int   Y;
int   X = (Y + 4) * 3;
```

Following is the equivalent translation to assembly language. The translation requires multiple statements because each assembly language statement corresponds to a single machine instruction:

```
mov   eax,Y                    ; move Y to the EAX register
add   eax,4                    ; add 4 to the EAX register
mov   ebx,3                    ; move 3 to the EBX register
imul  ebx                      ; multiply EAX by EBX
mov   X,eax                    ; move EAX to X
```

(*Registers* are named storage locations in the CPU that hold intermediate results of operations.) The point of this example is not to claim that C++ is superior to assembly language or vice versa, but to show their relationship.

Hello, World

C

```c
#include <stdio.h>

int main() {
printf("Hello World");
return 0;
}
```

Assembly

```asm
; ------------------------------------------------------------------------------
; Writes "Hello, World" to the console using only system calls. Runs on 64-bit Linux only.
; To assemble and run:
;
;     nasm -felf64 hello.asm && ld hello.o && ./a.out
; ------------------------------------------------------------------------------

          global    _start

          section   .text
_start:   mov       rax, 1                  ; system call for write
          mov       rdi, 1                  ; file handle 1 is stdout
          mov       rsi, message            ; address of string to output
          mov       rdx, 13                 ; number of bytes
          syscall                           ; invoke operating system to do the write
          mov       rax, 60                 ; system call for exit
          xor       rdi, rdi                ; exit code 0
          syscall                           ; invoke operating system to exit

          section   .data
message:  db        "Hello, World", 10      ; note the newline at the end
```

Assembly Programming

汇编语言的特点：

- **符号化语言**：汇编语言使用助记符（mnemonics）来代表不同的机器指令，这使得编写汇编程序相对容易理解和维护。
- **直接硬件访问**：汇编语言允许程序员直接控制计算机的硬件，包括处理器、内存、寄存器等。这使得汇编语言非常适合编写系统级和嵌入式程序。
- **低级别**：汇编语言是一种低级别语言，与高级编程语言相比，它更接近计算机硬件。这意味着程序员需要更多地关注底层细节。

Assembly Programming

- **可移植性差**：汇编语言通常是与特定的计算机架构或处理器相关的，因此不太具有跨平台可移植性。编写的汇编程序通常只能在特定的硬件上运行。
- **性能控制**：由于可以直接访问硬件，汇编语言程序员可以对程序的性能进行更精细的控制，优化程序以获得更高的执行速度。
- **较少的高级结构**：与高级编程语言相比，汇编语言缺乏高级的控制结构，如条件语句和循环。程序员必须使用分支指令和跳转来实现这些控制结构。
- **调试复杂**：由于汇编语言的低级别特性，调试汇编程序更加复杂，因为程序员需要深入了解程序的内部工作原理。

Assembly Programming

汇编语言基本概念：

- ✓ 汇编语言是机器语言的一种助记符形式，它使用容易理解的符号代替二进制代码。
- ✓ 汇编语言程序由一系列指令组成，每个指令执行特定的操作，如加载数据、进行算术运算、控制程序流程等。
- ✓ 汇编语言通常包括众多寄存器，用于存储和操作数据。
- ✓ 汇编语言程序需要通过汇编器转换成机器码，以便计算机硬件能够执行。

Assembly Programming

汇编语言的重要性：

- 接近硬件
- 效率
- 学习计算机体系结构
- 调试和优化
- 系统编程

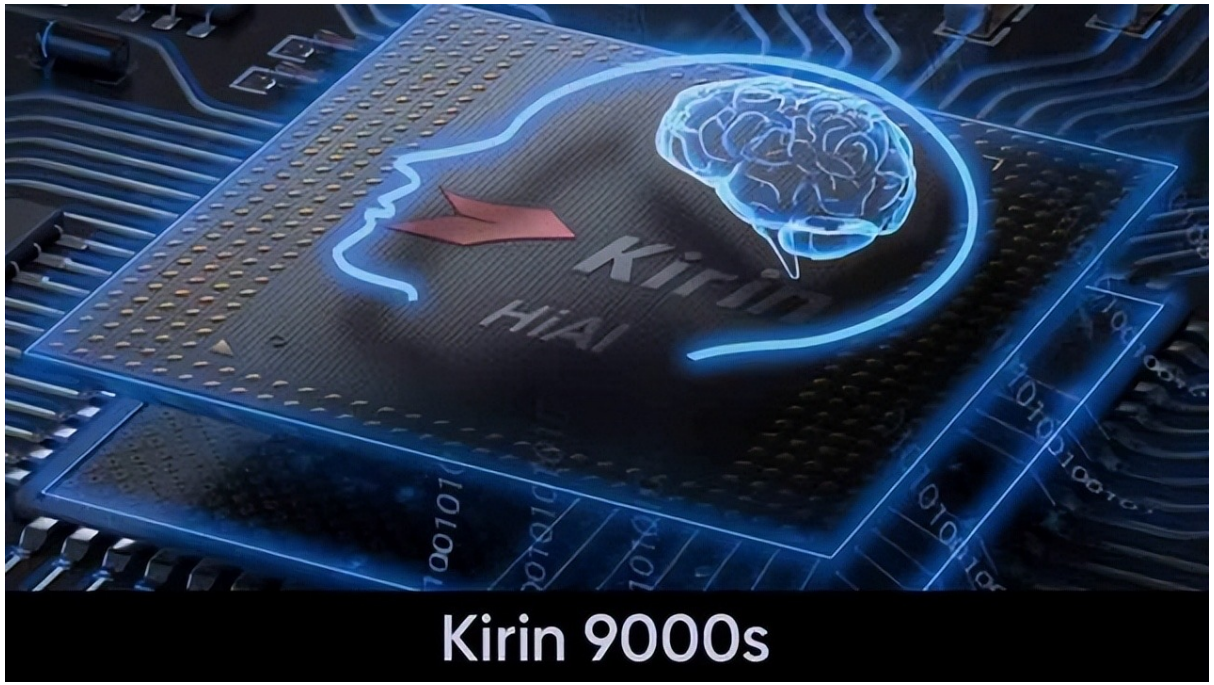
Applications

Table 1-1 Comparison of Assembly Language to High-Level Languages.

Type of Application	High-Level Languages	Assembly Language
Commercial or scientific application, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	The language may not provide for direct hardware access. Even if it does, awkward coding techniques may be required, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Commercial or scientific application written for multiple platforms (different operating systems).	Usually portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	May produce large executable files that exceed the memory capacity of the device.	Ideal, because the executable code is small and runs quickly.

- ❖ 嵌入式系统开发
- ❖ 操作系统内核开发
- ❖ 设备驱动程序
- ❖ 实时系统和嵌入式控制
- ❖ 安全和漏洞研究
- ❖ 反汇编和逆向工程
- ❖ 性能优化

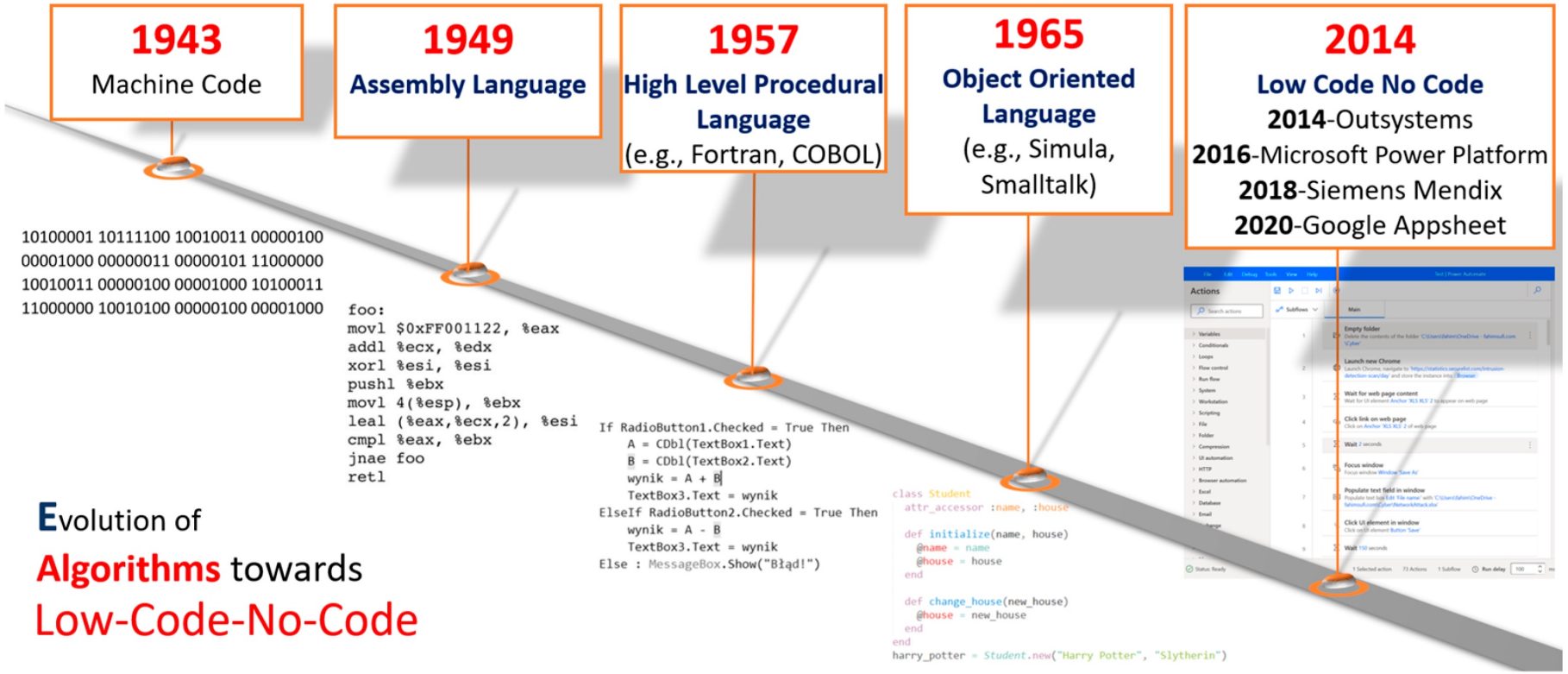
Applications



汇编 & 芯片

1.芯片架构
2.底层控制
3.性能优化
4.嵌入式系统
5.操作系统内核
6.硬件驱动程序

History



Sufi, Fahim. "Algorithms in low-code-no-code for research applications: a practical review." *Algorithms* 16.2 (2023): 108.

Concepts

汇编语言的核心概念包括指令、寄存器、内存、地址、操作数、程序计数器和标志寄存器。

Concepts

汇编语言的核心概念包括指令、寄存器、内存、地址、操作数、程序计数器和标志寄存器。

指令（Instruction）：

指令是汇编语言程序的基本构建块，每个指令表示一个特定的计算机操作，例如加载数据到寄存器、执行算术运算、分支跳转等。

指令通常使用助记符（mnemonic）来表示，例如，"MOV"表示数据移动，"ADD"表示加法操作。

Concepts

汇编语言的核心概念包括指令、寄存器、内存、地址、操作数、程序计数器和标志寄存器。

寄存器（Register）：

寄存器是计算机内部的临时存储器，用于存储数据和执行操作。它们是计算机中最快的存储器，通常位于CPU内部。

不同的计算机体系结构具有不同的寄存器集合，每个寄存器有唯一的名称，如EAX、EBX、ECX等。这些寄存器用于存储整数、浮点数、地址等不同类型的数据。

Concepts

汇编语言的核心概念包括指令、寄存器、内存、地址、操作数、程序计数器和标志寄存器。

内存（Memory）：

内存是计算机用于存储程序和数据的地方，它可以被看作是一个大型字节数组。

内存中的每个位置都有一个唯一的地址，程序可以通过地址来读取和写入数据。汇编语言中使用内存地址来操作数据。

内存通常分为指令内存（存储程序指令）和数据内存（存储程序数据）。

Concepts

汇编语言的核心概念包括指令、寄存器、内存、地址、操作数、程序计数器和标志寄存器。

地址（Address）：

地址是内存中的位置标识，用于访问内存中的数据。地址可以是一个数值或一个表达式，例如，0x0012表示内存中的地址12。

汇编语言中的指令可以使用地址来指定要读取或写入的内存位置。

Concepts

汇编语言的核心概念包括指令、寄存器、内存、地址、操作数、程序计数器和标志寄存器。

操作数（Operand）：

操作数是指令中的数据，它们可以是立即数（常数值）、寄存器中的值或内存中的值。

指令通常指定了操作数的来源和目的地，例如，"MOV AX, 10"表示将值10移到寄存器AX中。

Concepts

汇编语言的核心概念包括指令、寄存器、内存、地址、操作数、程序计数器和标志寄存器。

程序计数器（Program Counter）：

程序计数器是一个寄存器，用于跟踪当前正在执行的指令的位置。

每当执行一条指令时，程序计数器的值会自动增加，以指向下一条要执行的指令。

Concepts

汇编语言的核心概念包括指令、寄存器、内存、地址、操作数、程序计数器和标志寄存器。

标志寄存器（Flag Register）：

标志寄存器包含一组标志位，用于存储关于前一条指令执行结果的信息，如进位、零、溢出等。

这些标志位可用于条件分支指令，以根据前一条指令的执行结果来决定程序的流程。

Assembly Instruction

标签（Label）（可选）：

标签是指令的可选部分，用于标识指令的位置或作为跳转目标。标签通常以冒号（:）结尾，例如：

makefile Copy code

```
LoopStart:
```

Assembly Instruction

助记符（Mnemonic）：

助记符是指令的核心部分，用于指示要执行的操作。

助记符通常是字母缩写

```sql
MOV    ; 数据移动指令
ADD    ; 加法指令
JMP    ; 无条件跳转指令
```

Assembly Instruction

操作数（Operands）：

操作数是指令的参数，用于指定操作的目标和源数据。

操作数可以是立即数（常数值）、寄存器中的值或内存中的值。

操作数通常由逗号分隔。

```sql
MOV AX, 10      ; 将值10移动到寄存器AX中
ADD BX, AX      ; 将寄存器AX的值加到寄存器BX中
```

Assembly Instruction

注释（Comment）（可选）：

注释是用来解释指令或提供文档的可选部分，不会被计算机执行。

注释通常以分号（;）开头

Assembly Instruction

末尾换行符：

汇编指令通常以换行符结束，以分隔不同的指令。

assembly Copy code

```
LoopStart:
    MOV AX, 10     ; 将值10移动到寄存器AX中
    ADD BX, AX     ; 将寄存器AX的值加到寄存器BX中
    CMP BX, 100    ; 比较寄存器BX和值100
    JG  LoopStart ; 如果BX > 100，则跳转到标签LoopStart
```

Assembly

汇编语言通常不具备可移植性，这意味着编写的汇编程序通常特定于某种计算机体系结构或硬件平台。主要原因：

- 指令集体系结构差异
- 寄存器和内存布局
- 操作系统接口
- 汇编器和汇编语言规范

Assembly Instruction

尽管汇编语言通常不可移植，但仍然有一些方式可以提高可移植性：

- 使用宏汇编语言：
- 编写抽象层：
- 使用跨平台汇编工具：

Assembler

- ✓ MASM (Microsoft Macro Assembler)：MASM 是由微软开发的汇编器，适用于 x86 架构。
- ✓ NASM (Netwide Assembler)：NASM 是一个开源的、跨平台的汇编器，支持多种体系结构，包括 x86、x86-64、ARM 等。
- ✓ GNU Assembler (Gas)：GNU Assembler，也称为 Gas，是 GNU 工具链中的一部分，支持多种体系结构，包括 x86、x86-64、ARM 等。

Q&A