

# Computer System & Application

## Introduction

### 目录

Computer System & Application Introduction .....	1
序言 .....	2
计算机系统（硬件视角） .....	2
计算与计算机.....	2
模拟电路、数字电路 .....	5
微型计算机原理、计算机组成原理.....	11
计算机体系结构.....	19
计算机系统（软件视角） .....	19
指令、指令集与汇编 .....	19
编译原理与高级语言 .....	22
操作系统 .....	23
计算机网络.....	25
数据结构 .....	26
算法设计与分析.....	28
自动机与计算理论 .....	30
数据库理论.....	31
软件工程 .....	32
计算机应用.....	34
机器学习 .....	34
参考文献 .....	35

## 序言

目前，计算机科学技术已成为我们日常生活中不可或缺的一部分，各行各业都在某种程度上使用着计算机科学技术。例如，路径规划、机器翻译、语音识别、人脸识别、自动驾驶、GPT-3、GPT-4 等。本文档简要的介绍了计算机科学技术的发展和联系。主要分为计算机系统与计算机应用两部分。在计算机系统部分，分别从硬件角度（计算机发展史->逻辑电路->计算机主机）与软件角度（计算机指令->计算机软件系统）展开说明。在计算机应用部分，介绍了目前较为主流的机器学习相关知识。本文档旨在帮助计算机专业学生、老师、计算机相关方向从业人员更好的了解计算机知识、复习和巩固计算机基础。本文档不需一遍读通，可多次反复阅读，最后构建自己的计算机科学技术体系。

## 计算机系统（硬件视角）

### 计算与计算机

随着人类文明及科技的高速发展，计算始终贯穿着各个时代。为了减少手工计算的时间及误差，我们发明了很多辅助计算或自动计算工具。其中，计算机的计算能力最为出色。在今天，计算机已成为各个领域重要的计算工具。

在计算机较为稀有的年代，我们对于问题的解决方案通常是写死在表格纸上（像是曲线图和列线图解），用来一并解决相似的问题，比如说暖气机里的温度和压力分布。第二次世界大战之前，当时的最高科技是机械式和电动式的模拟计算机，也被认为是前途光明的计算机趋势。模拟计算机使用连续变化的物理量，像是电势、流体压力、机械运动等，处理表示待解问题中相应量的器件。例如，1930 年发明的微分分析器，1936 年制作的水流积算器。此外，部分模拟计算机广泛应用在军事瞄准用途。例如，美国诺顿轰炸机的瞄准器和火力控制系统，美国海军开发的马克一号火力控制电脑。与现代数字计算机相比，模拟计算机不具弹性，必须手动装配（类似重新编写程序）才能处理下一个待解问题。同时，早期的数字计算机能力有限，无法解决太过复杂的问题。故在当时，模拟计算机还是占有优势。

随着数字电路技术的发展，数字计算机运算速度越来越快，存储能力越来越强。此时，模拟计算机迅速受到淘汰，程序设计也成为我们另一项专业技能。

1930 年后期到 1940 年，受第二次世界大战的影响，该时期是计算机发展中最混乱的时期。战争开启了现代计算机的时代，电子电路、继电器、电容及真空管在该时期相继登场，取代了原来的机械器件，类比计算器也被数字计算器所代替。

1936 年，阿兰·图灵发表的研究报告对计算机和计算机科学领域造成了巨大冲击。这篇报告主要是为了证明循环处理程式的死角，即停机问题的存在。图灵也以算法概念为通用计算机做出定义，后来也称为图灵机，取代了哥德尔渐趋累赘的通用语言。

1937 年，美国数学家兼工程师克劳德·香农在麻省理工学院发表了他的硕士论文《中继和交换电路的符号分析》（A Symbolic Analysis of Relay and Switching Circuits）。该论文中提出的观点和理论成为了数字电路设计的实践基础。他是史上首个将布尔代数应用在电子继电器和电闸上的人。

1937 年 11 月，在贝尔实验室工作的乔治·史提比兹在他家厨房组装出一部以继电器表示二进位制的计算机“K 模型机”。贝尔实验室在 1938 年通过了史提比兹提出的所有研究计划。1940 年 1 月，复数计算器完工。1940 年 9 月，在达特茅斯学院召开的美国数学学会会议上，史提比兹透过电话线向复数计算器传送远端指令，这是计算机远端遥控的首度实例。参与会议的主要有约翰·冯·诺伊曼、约翰·莫克利和诺伯特·维纳等。

1939 年，爱荷华州立大学的约翰·阿塔纳索夫和克里夫·贝理开发出阿塔纳索夫-贝瑞计算机（ABC）。这是一台解决一次方程问题的电子计算机。ABC 使用超过 300 个真空管提高运算速度，以固定在机械旋转磁鼓上的电容器作为记忆器件。虽然不可编程化，但是采用二进位制和电子线路等各方面，都使其成为第一部现代计算机的先驱。

同年，由哈佛大学数学家霍华·艾肯总筹指挥，IBM 赞助人力资金的马克一号在 IBM 安迪卡特实验室起手开发，其正式名称为自动化循序控制计算器。这是一个电动机械计算机。马克一号主要参考巴贝奇分析机，使用十进制制、转轮式储存器、旋转式开关以及电磁继电器。并由数个计算单元平行控制，经由打孔纸带进行程式化（改良后改由纸带读取器控制，并可依条件切换读取器）。虽然马克一号被认为是第一部通用计算机，但其并没有达到图灵完全的条件。

研制电子计算机的想法主要产生于第二次世界大战期间。当时各国的武器装备比较差，武器大多为飞机、大炮，因此研制和开发新型大炮和导弹就十分必要。为此，美国陆军军械部在马里兰州的阿伯丁设立了“弹道研究实验室”。美国军方要求该实验室每天为陆军炮弹部队提供 6 张射表以便对导弹的研制进行技术鉴定。然而，每张射表都要计算几百条弹道，而每条弹道的数学模型是一组非常复杂的非线性方程组，这些方程组是没有办法求出准确解的，因此只能用数值方法近似地进行计算。然而即使用数值方法近似求解，实验室雇用 200 多名计算员加班加点工作也大约需要二个多月的时间才能算完一张射表。为了改变这种不利的状况，当时任职宾夕法尼亚大学莫尔电机工程学院的约翰·莫希利于 1942 年提出了试制第一台电子计算机的设想——“高速电子管计算装置的使用”，期望用电子管代替继电器以提高机器的计算速度。

美国军方得知这一设想后，马上拨款进行大力支持，成立了一个以莫希利、埃克特为首的研制小组，并着手准备研制工作，该项目预算经费大概为 15 万美元。与此同时，当时任弹道研究所顾问、正在参加美国第一颗原子弹研制工作的数学家冯诺伊曼带着原子弹研制过程中遇到的大量计算问题，在研制过程中期加入了计算机研制小组。1945 年，冯诺依曼和计算机研制小组共同发表了一个全新的“存储程序通用电子计算机方案”——EDVAC（Electronic Discrete Variable Automatic Computer）。在此过程中，冯诺依曼对计算机的许多关键性问题的解决做出了重要贡献，从而保证了计算机的顺利问世。虽然 ENIAC 体积庞大，耗电惊人，运算速度不过几千次，但它比当时已有的计算机要快 1000 倍，而且还可按事先编好的程

序自动进行算术运算、逻辑运算和存储数据等。ENIAC 理论的提出，开启了一个全新的，科学计算的时代。

ENIAC 每秒能进行 5000 次加法运算，400 次乘法运算。此外，它还能进行平方、立方、正弦、余弦等计算及其它更为复杂的运算。原来需要 20 多分钟时间计算出来的一条弹道，现在只要短短 30 秒。与此同时，冯诺依曼当时也在参与原子弹的研制工作，ENIAC 为世界上第一颗原子弹的诞生也出了不少力。

随着电子电路、数字电路、集成电路等技术及相关数学理论的大力发展，数字计算机技术也在不断进化，大致可分为以下 4 代：

- 第 1 代：电子管数字计算机（1946—1958 年）。
- 第 2 代：晶体管数字计算机（1958—1964 年）。
- 第 3 代：集成电路数字计算机（1964—1970 年）。
- 第 4 代：大规模集成电路数字计算机（1970 年至今）。

模拟电路、数字电路

1 二极管、三极管与门电路

随着集成电路技术的发展，我们利用各种复合材料制作了二极管电路元件，图 1 为二极管示意图。二极管元件只有电压超过某一数值时，才会使电流通过。故可通过控制电路电压，控制二极管的输出内容。更进一步，可以使用二极管组成复合电路完成简单的逻辑运算（与、或、非）。

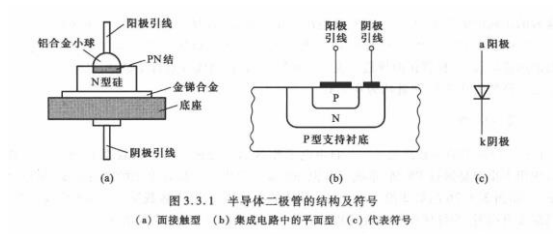


图 1 二极管示意图

然而，二极管电路存在着“输出电平漂移”问题，无法制作为具有标准化输出电平的集成电路。之后，随着 CMOS 技术与 TTL 技术的兴起，使用这两种技术可制作一种三极管电路元件。三极管元件很好避免了二极管电路产生的问题，是一种较为稳定的电路。根据以上三极管的制作工艺，三极管元件可分为 CMOS 型三极管和 TTL 型三极管，如图 2、3 所示。

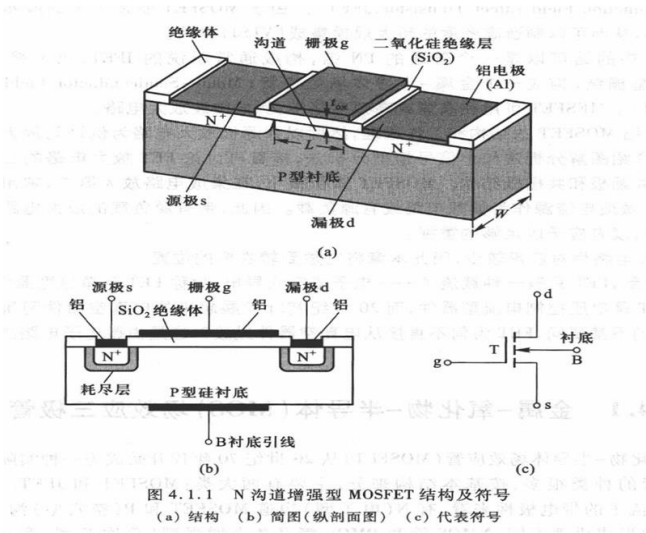
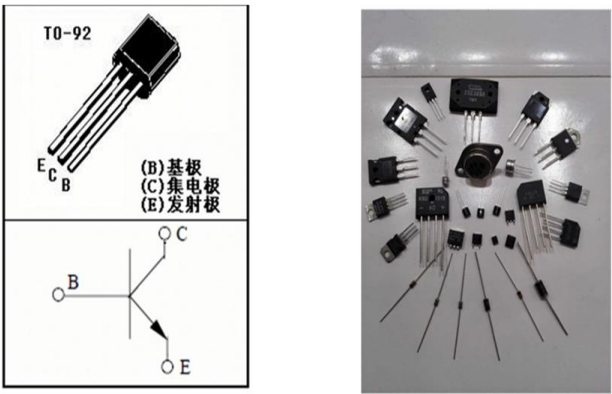


图 2 CMOS 型三极管



与二极管元件类似，可将三极管元件进行进一步组合，形成更为复杂的电路结构，一般称为门电路。分为 COMS 门电路和 TTL 门电路，如图 4 所示。

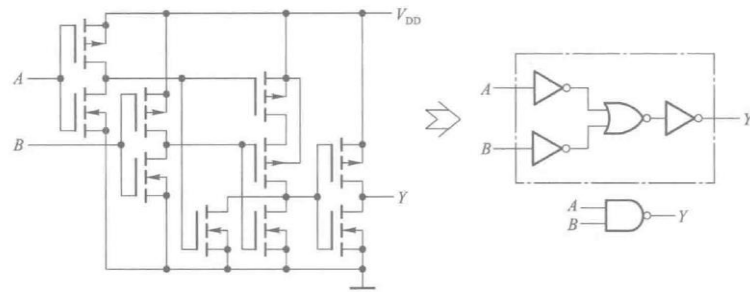


图 3.3.30 带缓冲级的 CMOS 与非门电路

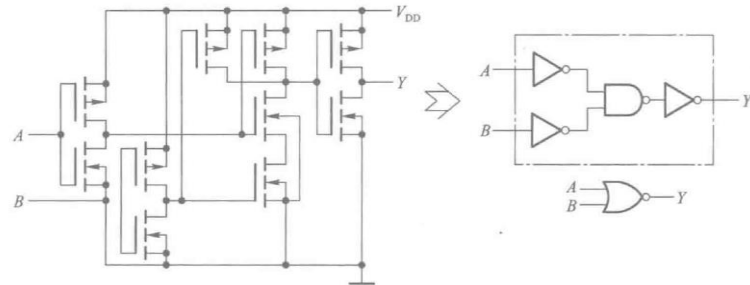


图 3.3.31 带缓冲级的 CMOS 或非门电路

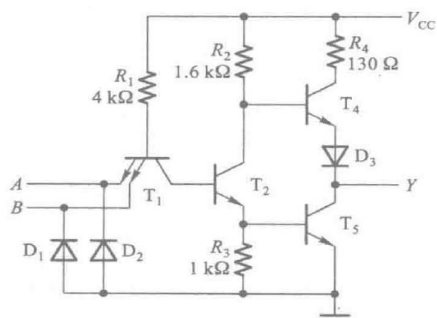


图 3.4.27 TTL 与非门电路

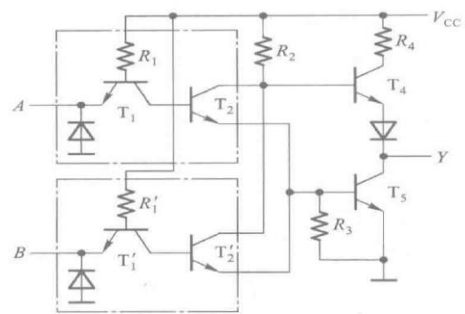


图 3.4.29 TTL 或非门电路

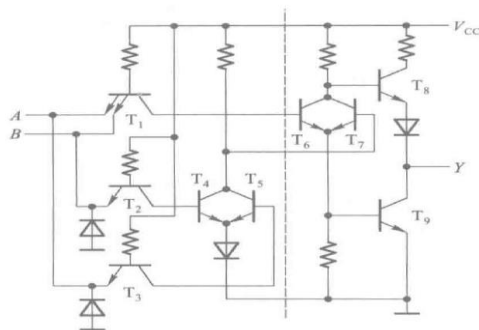


图 3.4.31 TTL 异或门

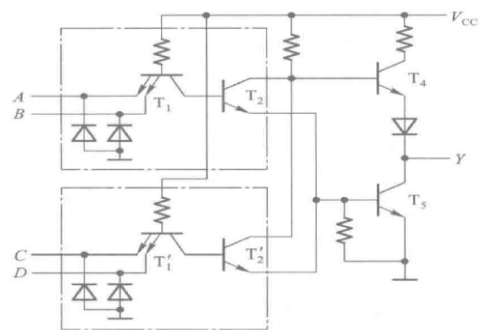


图 3.4.30 TTL 与或非门

## 图 4 CMOS 型与 TTL 型门电路



图 5 为门电路电路标识。

序号	名称	GB/T 4728.12-1996		国外流行图形符号	常用图形符号
		限定符号	国标图形符号		
1	与门	$\&$			
2	或门	$\geq 1$			
3	非门	  逻辑非入和出	 	 	 
4	与非门				
5	或非门				
6	与或非门				
7	异或门	$=1$			
8	同或门	$=$	 		
9	集电极开路 OC 门、漏极 开路 OD 门	 L 型开路输出			
10	缓冲器				

图 5 门电路标识图

2 组合逻辑电路与时序逻辑电路

可将门电路进一步进行组合，形成更复杂的组合逻辑电路或时序逻辑电路。组合逻辑电路实时输入，实时输出。时序逻辑电路有输入信号后，会将输出信号进行存储。图 6 是几种常见的组合逻辑电路。图 7 触发器（锁存器）是一种基础的时序逻辑电路。



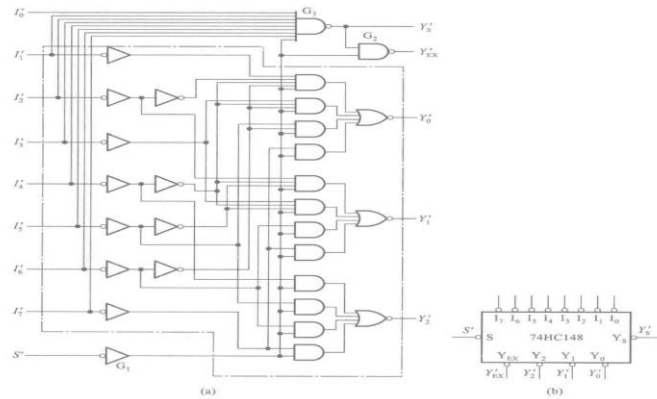


图 4.4.3 8 线-3 线优先编码器 74HC148  
(a) 内部逻辑图 (b) 逻辑框图

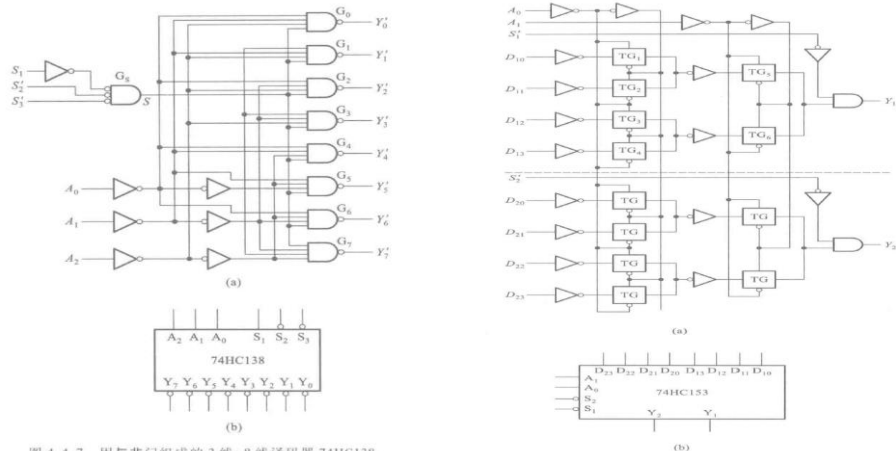


图 4.4.7 用与非门组成的 3 线-8 线译码器 74HC138  
(a) 内部逻辑图 (b) 逻辑框图

图 4.4.20 双 4 选 1 数据选择器 74HC153  
(a) 内部逻辑图 (b) 逻辑框图

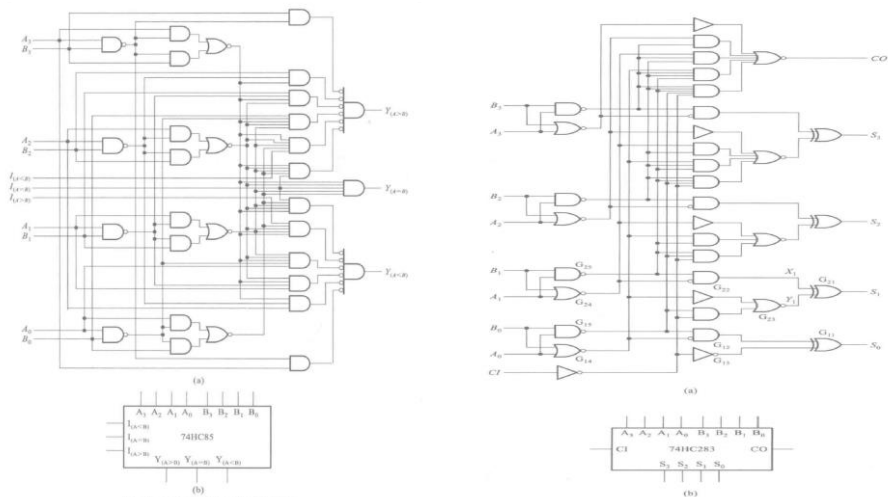


图 4.4.27 4 位数值比较器 74HC85  
(a) 内部逻辑图 (b) 逻辑框图

图 4.4.25 4 位超前进位加法器 74HC283  
(a) 内部逻辑图 (b) 逻辑框图

图 6 常见的组合逻辑电路

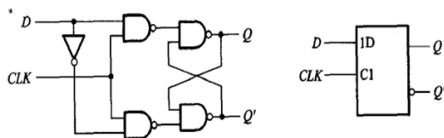


图 5.3.4 电平触发 D 触发器(D 型锁存器)

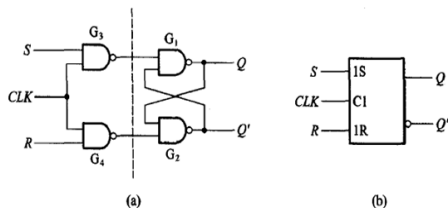


图 5.3.1 电平触发 SR 触发器(同步 SR 触发器)  
(a) 电路结构 (b) 图形符号

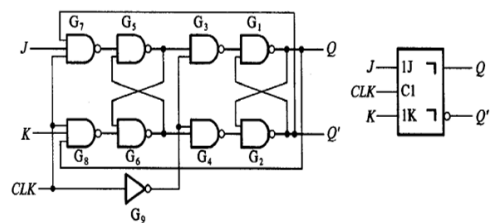


图 5.4.3 主从 JK 触发器

## 图 7 常见的触发器

根据以上触发器的电路图可知，触发器可以存储电路中的通断信息，也就是一位二进制数据。可进一步将触发器进行组合，用于存储多位二进制数据，既寄存器。图 8 为几种常见的组合方式。

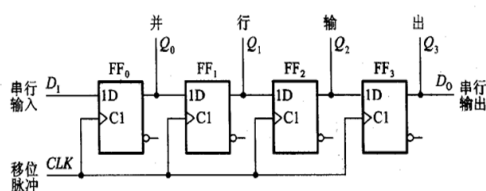


图 6.3.3 用 D 触发器构成的移位寄存器

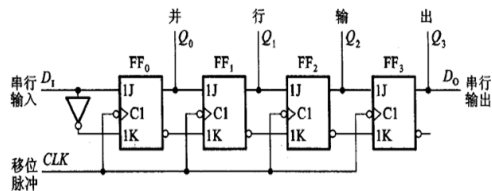


图 6.3.5 用 JK 触发器构成的移位寄存器

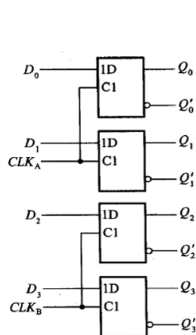


图 6.3.1 74LS75 的逻辑图

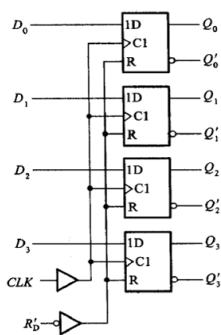


图 6.3.2 74HC175 的逻辑图



## 图 8 几种寄存器及实物图

## 微型计算机原理、计算机组成原理

### 1 冯·诺依曼体系

在第二次世界大战期间，为了给美国军械试验提供准确而及时的弹道火力表，迫切需要一种高速的计算工具。因此在美国军方的大力支持下，冯诺依曼作为技术顾问，参与了通用计算机 ENIAC 的研究。

在冯诺依曼体系结构中，计算机主要由存储器、运算器、控制器、输入设备、输出设备组成。目前的电子计算机也是遵循着这种结构。图 9 为冯诺依曼体系结构图。

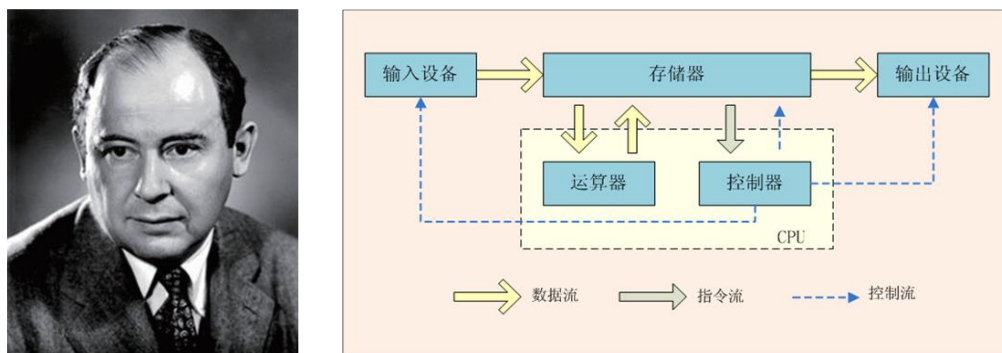


图 9 冯诺依曼及冯诺依曼体系结构

其中，运算器、控制器和寄存器组成中央控制单元（CPU），用于从存储器中获取数据进行具体计算。存储器存储输入设备输入的数据。输出设备输出最终计算结果。

### 2 CPU 结构

#### 2.1 寄存器

图 10 展示了 Intel 8086 CPU 中的各类寄存器。在 8086 中，每个寄存器均为 16 位。

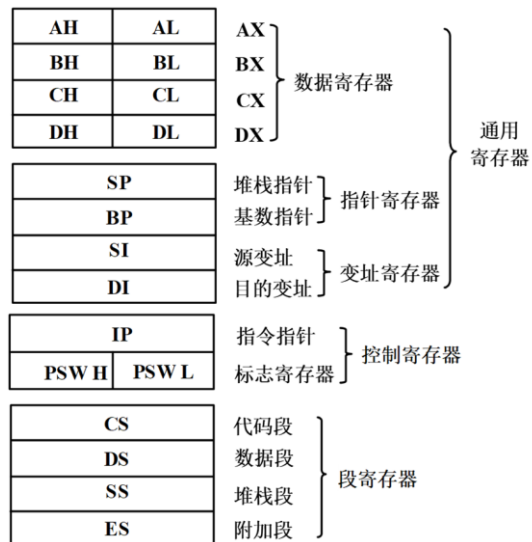


图 10 8086CPU 中的寄存器

为了让计算机进行更为复杂的运算，可将 CPU 中的寄存器进一步扩展为 32 位、64 位。如图 11 所示。

寄存器	64 位模式			传统的 IA-32 环境		
	名称	数目	大小( 位 )	名称	数目	大小( 位 )
通用寄存器	RAX、RBX、RCX、RDX、RBP、RSI、RDI、RSP、R8:15	16	64	EAX、EBX、ECX、EDX、EBP、ESI、EDI、ESP	8	32
指令寄存器	RIP	1	64	EIP	1	32
标志寄存器	EFLAGS	1	32	EFLAGS	1	32
流 SIMD 扩展 ( SSE ) 寄存器	XMM0:15	16	128	XMM0:7	8	128

图 11 32 位、64 位寄存器

## 2.2 运算器、控制器、总线、引脚

图 12 为 8086 CPU 的具体结构。其中，ALU 为算术逻辑单元（运算器电路），负责具体计算（加法、减法、乘法、除法、逻辑）。EU 为控制器，负责控制 CPU 如何从存储器中获取数据、CPU 获取数据后如何操作及 CPU 中脉冲信号等。总线

（Bus）是一种可以传输电信号的导线，负责寄存器、ALU、控制器等传输二进制数据。通常分为数据总线、控制总线、地址总线。

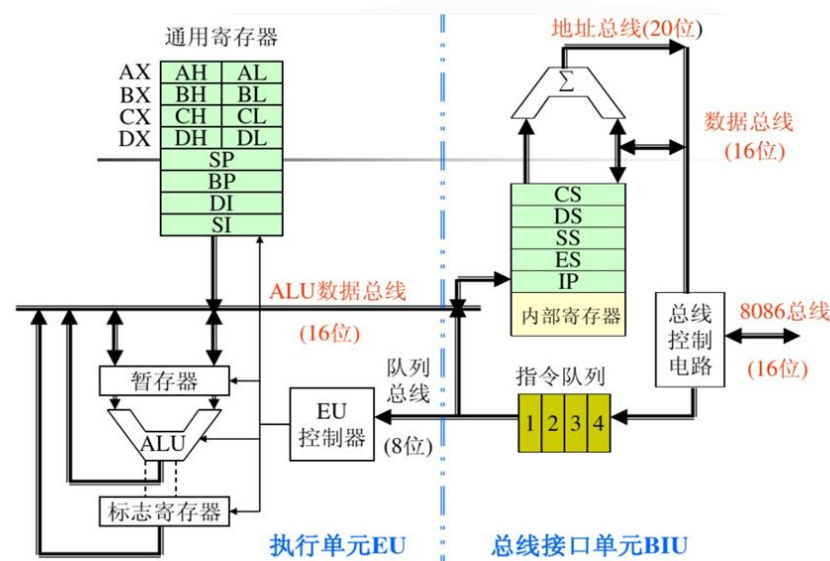


图 12 CPU 内部结构

进一步，可使用集成电路技术，将 8086 CPU 内部进行封装制成 CPU 芯片。图 13 为 8086 CPU 芯片引脚结构图及其实物图。

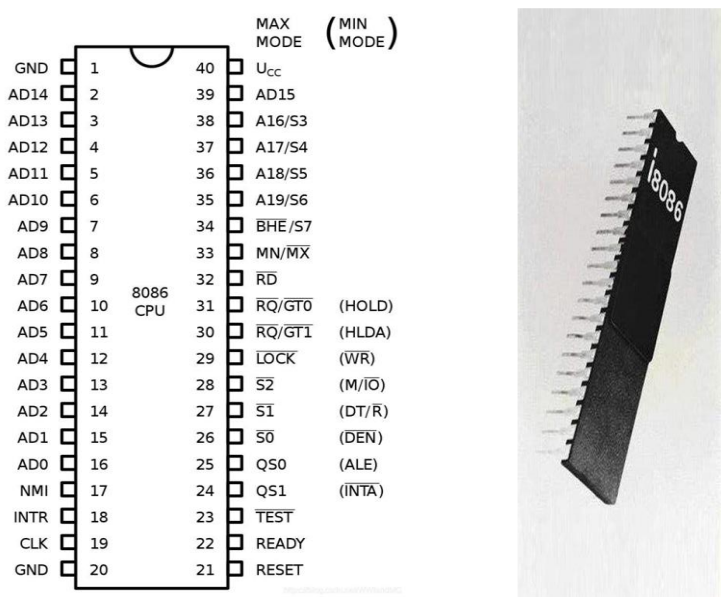


图 13 8086 CPU 芯片引脚结构及实物图

随着集成电路技术的进一步发展，目前 CPU 结构更为小巧、精密（纳米(nm)制程工艺越来越小）。图 14 为 Intel core 11 代 CPU i7-11700K 实物图。此外，在手机系统中，有着和 CPU 类似结构的芯片，一般称作 SOC，如图 15 所示。但 SOC 的构造和功能比 CPU 芯片更为复杂。



图 14 Intel core i7-11700K



图 15 SOC 芯片

同时，为了降低 CPU 温度，目前 CPU 大多都配置了风冷装置或水冷装置，如图 16 所示。笔记本电脑一般会配置散热管。



图 16 CPU 风冷及水冷



3 存储器

存储器一般可分为 ROM 与 RAM。ROM 负责存储不再更改的重要数据。RAM 存储与 CPU 交互的数据，一般也叫内存。ROM 与 RAM 的电路主要也是由组合逻辑与时序逻辑构成。图 17、18 分别为 ROM 与 RAM 的电路框架、结构图。此外也可设计符合 RAM 或 ROM 电路框架的其他存储器电路。

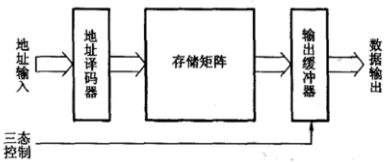


图 7.2.1 ROM 的电路结构框图

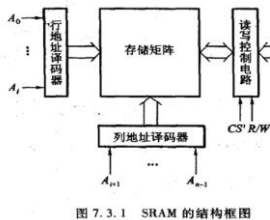


图 7.3.1 SRAM 的结构框图

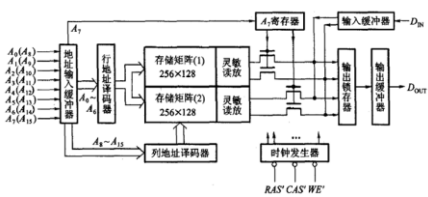


图 7.3.9 DRAM 的总体结构框图

图 17 ROM 与 RAM 电路总体框架图

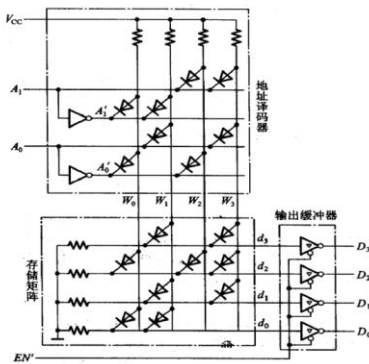


图 7.2.2 二极管 ROM 的电路结构图

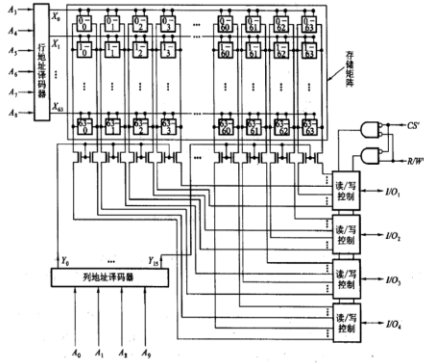


图 7.3.2 1024 x 4 位 RAM (2114) 的结构框图

图 18 ROM 与 RAM 电路结构图

为了将多个存储空间较小的 RAM 芯片组合为存储空间更大的芯片，可使用位扩展或字扩展法，如图 19 所示。

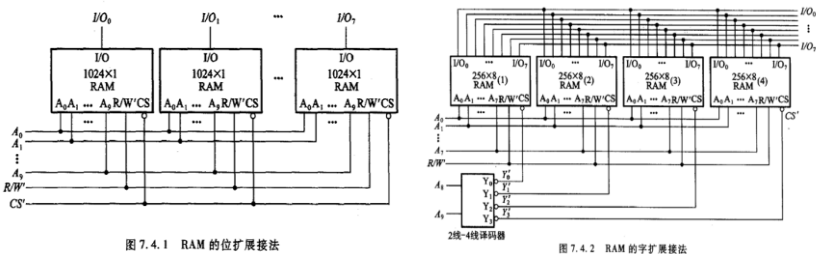


图 19 RAM 的位扩展和字扩展

图 20 为内存条实物图，可以看出，内存条是由多个 RAM 芯片扩展组成。



图 20 内存条实物图

#### 4 机械硬盘与固态硬盘

为了实现大规模计算，可提前将输入数据写入至外部设备中。目前计算机大多采用机械硬盘或固态硬盘存储大规模数据，如图 21 所示。之后不断将数据传输至内存，通过 CPU 完成计算。

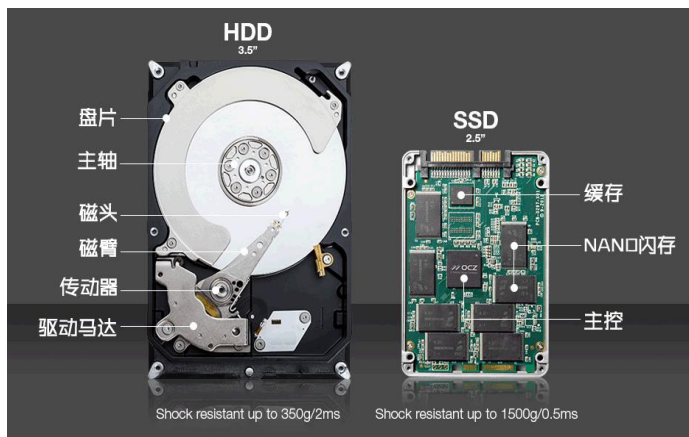


图 21 机械硬盘的固态硬盘

固态硬盘内部一般采用闪存存储技术，其数据读写速度远大于使用磁盘的机械硬盘。然而，闪存芯片存在擦写次数限制的问题，其寿命通常比机械硬盘短。闪存芯片一般分为 SLC、MLC、TLC 等规格。SLC 大概有 10 万次的写入寿命，MLC 大概为 1 万次，TLC 大概为 1000-2000 次。

由于固态硬盘的价格要远高于机械硬盘，当存储大数据时（TB、PB 级），通常还是会选择机械硬盘。机械硬盘可分为叠瓦盘（SMR）和垂直盘（CMR）。SMR 价格通常低于 CMR，但 SMR 写入速度一般慢于 CMR，其综合性能也低于 CMR。目前，移动硬盘（机械）通常为 SMR，其总体性能较差。此外，当机械硬盘容量较大时，常常会将其中填充的空气替换为氮气，以增加磁盘数，物理上扩充容量而不影响总体大小。但氮气磁盘挂载过程中，与空气磁盘相比，会发出较大声响。

另外，保存与分享大量数据时，也可构建网络存储器（NAS）。当我们需要保存一些重要、安全性极高的数据时，可利用多个磁盘构建磁盘冗余阵列 (RAID)，以此确保数据安全。

## 5 显卡

计算机通常会接入显卡以提供更高清的图形、视频输出，如图 22 所示。在显卡中，含有图形计算单元（GPU），用于并行处理图像、视频等数据。此外，在深度学习中，也依靠着 GPU 进行神经网络的相关计算。



图 22 显卡 NVIDIA RTX 3080Ti

## 6 主板与主机

随着大规模集成电路技术的进一步发展，目前，计算机主要使用各个厂商设计的主板电路，如图 23 所示。然后，将以上冯诺依曼结构的几种部件通过对应接口接入主板，并在主板中接入电源，之后将接好的主板放入机箱，并加入散热风扇，既形成了目前主机结构，如图 24 所示。此外，主板中一般也会提供其他外部设备接口，可扩展性较强，可以较为方便的接入符合接口标准的新外部设备。随后，显示器与显卡连接，接入鼠标、键盘，安装操作系统，计算机就可以开始工作了。

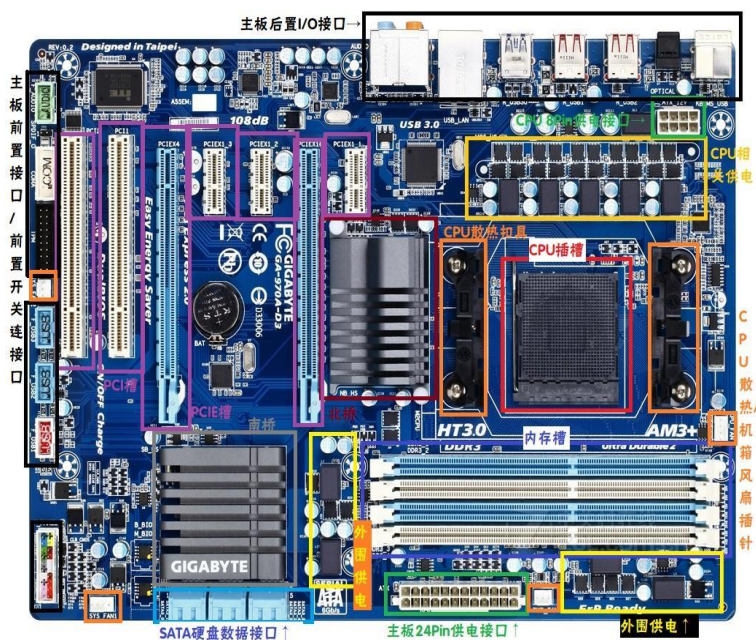


图 23 计算机主板



图 24 计算机主机

计算机体系结构

为进一步提高计算机中 CPU 运算速度及其稳定性、内存读取速度、输入输出设备灵敏度，同时降低计算机电路复杂度，减少计算机耗电量，可分别对 CPU、内存及相关电路结构进行优化，以达到更优的体系结构。例如 CPU 中利用多级流水线加速指令处理、CPU 与内存之间增加缓存加快数据处理速度（Intel core i7-11700K 中使用了 3 级缓存结构，L1、L2、L3）、利用中断电路使 CPU 合理的处理不同任务、使用 DMA 技术提高输入输出设备灵敏度、使用更先进的电路工艺、更简单的电路结构等。

计算机系统（软件视角）

指令、指令集与汇编

1 指令执行过程

在冯诺依曼结构中，CPU 需从内存中获取数据，之后进行相应计算。假设内存、CPU 中某一时刻有图 25 的数据。同时，CPU 要计算两数相加的结果，并将结果暂时存放于寄存器 AX。

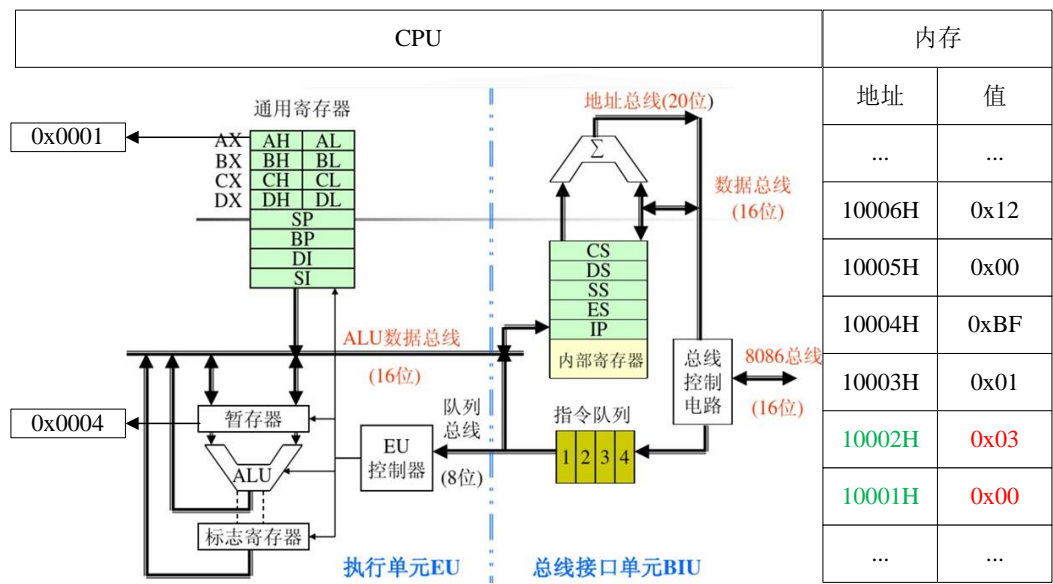


图 25 cpu 与内存的交互数据

可使用以下步骤完成计算过程：

- 1) 首先需定义“加法”操作。假设定义 00000001 为加法。只要 CPU 从内存中取出 00000001，就立刻在获取两个数据，通过 ALU 进行加法运算。然而内存中存储的都是二进制数据，需区分 00000001 表示的是“加法”操作还是表示数字 1。目前在 CPU 中，由时间信息进行区分，将 CPU 时间定义为取指令时间和执行指令时间。在取指令时间间隔内，CPU 从内存中获取的数据即为“操作”，也叫做“指令”。在执行指令时间间隔内，CPU 从内存中获取的数据即为操作数。取指与执行的具体时间划分和其他一些控制功能由 CPU 中的控制器（电路）完成。
- 2) 模仿步骤 1，继续定义“存储”操作，假设定义为 00000002。之后，CPU 根据控制器控制，执行存储操作，将 ALU 计算结果存入 AX 寄存器。

以上过程使用二进制数据可表示为：

00000001（执行加法） 00000001（1），00000011（3）

00000002（执行存储） 10000000（AX），00000100（ALU 结果 4）

通常，多个操作数数据不能同时来自于内存（汇编指令固有问题）。二进制数据代表的指令或特殊值也可按照不同时间对应的不同含义进行划分。

## 2 指令集与汇编

二进制输入指令，计算机电路可以很好执行，然而二进制数据不方便我们记忆。故可建立一种映射规则。假设存在以下规则：00000002-> mov, 00000100-> 4, 10000000-> ax。

根据以上映射，指令可重新表示为 mov ax, 4。这样表示就方便了我们的记忆，也方便了代码的编写。以上映射规则一般可建立一个映射文件，这个映射文件一般称为汇编程序（汇编器），映射之后的代码称为汇编语言。



由于不同机器逻辑电路结构可能不同，故对应的映射文件也可能不同，相同的二进制数据可能代表不同的汇编指令。例如，在某些计算机中，00000001 代表加法，而在另一类计算机中，00000001 代表乘法。目前较为常用的指令集有 x86（CISC）、arm、MIPS（RISC）指令集等。可以使用 cpu-z 等工具查看当前计算机支持的指令集，如图 26 所示。

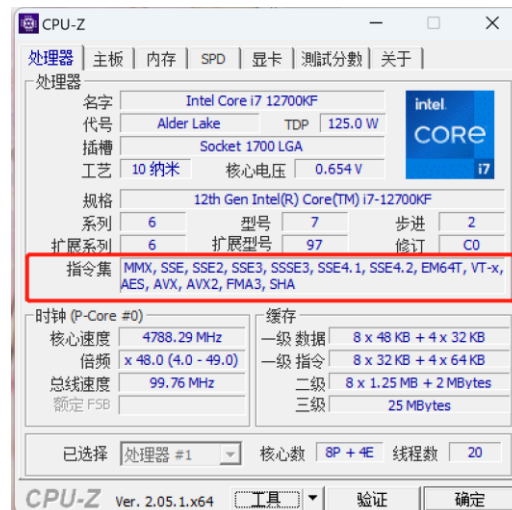


图 26 计算机指令集

汇编语言有很多优点：

1. 可以轻松的读取存储器状态以及硬件 I/O 接口情况。
2. 与二进制数据相比，我们很容易去阅读已经完成的程序或者理解程序正在执行的功能。
3. 作为一种低级语言，可扩展性很高。

然而,它也有诸多缺点：

1. 因为代码非常单调，特殊指令字符很少，所以造成了代码的冗长以及编写的困难。

2. 因为汇编仍然需要自己去调用存储器存储数据，很容易出现 bug，而且调试起来也不容易。
3. 就算完成了一个程序，后期维护时候也需要耗费大量的时间。
4. 因为机器的特殊性造成了代码兼容性差的缺陷。

## 编译原理与高级语言

由于汇编语言的诸多缺点，使得我们编写易移植的程序与调试程序 bug 较为困难。为了使编程语言更便于我们理解与 bug 调试，同时可以在各种机器中移植，可模仿汇编语言与二进制机器码的关系，可以设计更便于我们理解的语言（高级语言），并配合对应的编译程序（编译原理），将高级语言转换为对应的汇编代码：

高级语言->编译器->汇编语言->汇编程序->机器码->CPU 与内存交互执行指令

除以上过程，还有几种常见的过程：

高级语言->编译器->机器码 ->CPU 与内存交互执行指令

高级语言->解释器->机器码 ->CPU 与内存交互执行指令

C,C++,C#（编译型语言）、Java, Python（解释型语言）

将高级语言编写的程序，编译程序及汇编程序存入硬盘的指定位置，之后，可设计一个将硬盘程序读入内存的逻辑电路，将编译程序和汇编程序加载至内存，并与 CPU 交互执行对应编译、汇编指令。

有了编译和汇编程序后，我们就可利用高级语言编写我们自定义的程序，完成我们想设计的功能。之后通过数据读取电路，将程序读入内存，并作为编译程序输入。最终，由汇编程序输出机器码与 CPU 交互执行我们的程序指令。

## 操作系统

通过上述方法，可以让计算机方便的执行由高级语言编写的程序。我们可以编写程序 1，程序 2，程序 3， $\dots$ ，程序  $n$ ，交于计算机执行。

假设我们按顺序执行我们编写的程序。但是顺序执行时，程序  $n$  等待时间较长，可能不会在有限时间内执行到，进而影响其它任务。此时，可尝试令程序 1，程序 2，程序 3， $\dots$ ，程序  $n$  并行执行。然而并行执行对于每一个程序  $i$  都要设计对应的 CPU 结构，较为浪费硬件资源。也可尝试令程序 1，程序 2，程序 3， $\dots$ ，程序  $n$  在一个 CPU 上分时执行，分时操作可以等时分时，也可不等时分时。等时分时一般也称为 CPU 时间片轮转。此外，也可使程序 1 在 CPU 中执行一段时间，切换到程序 2 执行，在切换到程序  $n$  执行，在切换至程序 2，程序 3，程序 1， $\dots$ 。此时，CPU 在不同的程序中被轮换占用，这种方式一般称为程序的并发执行。用并发的方式，可以更有效的利用硬件资源，同时也可为每个程序更合理的分配运行时间。目前，大多数计算机都是采用并发方式控制正在执行的程序。以上正在执行的程序也称为进程。

若使用并发方式控制进程，需要考虑进程执行顺序和每个进程的执行时间。同时，需考虑每个进程的内存分配方式，以保证内存空间的高效利用。当内存空间不够时，也需考虑哪些进程需要立刻执行，哪些进程执行并不紧急。此外，利用高级语言编写多个程序时，以何种方式组织多个程序，可快速查找对应程序。而且，多个程序存储在硬盘上，类似于内存，需考虑硬盘以何种方式分配，可以高效的利用硬盘空间。一般，硬盘是以外接设备方式与主板相连，除了硬盘，还有 U 盘，鼠标，键盘，显示器，音箱等外接设备，需考虑如何管理这些外接设备，使这些设备工作不冲突。

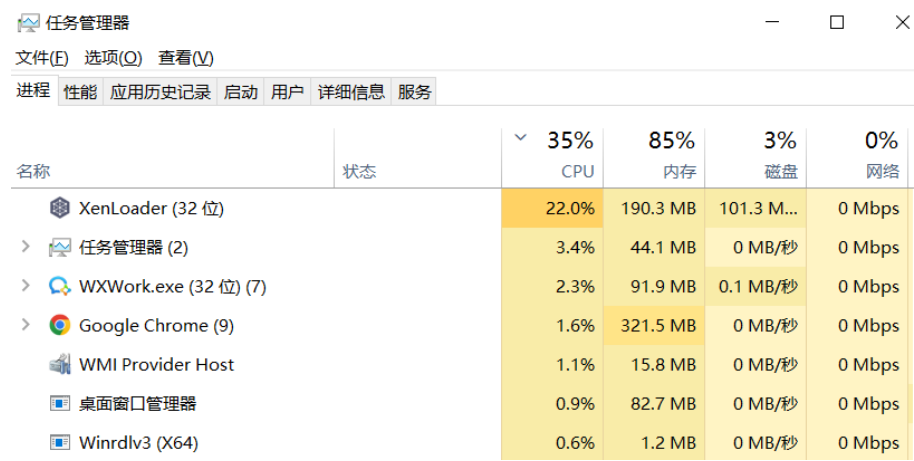
目前，操作系统通过进程管理，处理机管理，内存管理，文件管理，设备管理等方式实现了上述过程。通过操作系统，我们可以更为方便的使用计算机。例如：可以方便的安装和使用高级语言编写的应用程序，可以方便的建立文件，文件夹，

可以直接安装对应外设的驱动程序，使计算机直接识别鼠标、键盘等外接设备。我们不用在关心具体的硬件细节和电路结构。

当计算机开机时，操作系统依靠引导程序载入内存运行。首先由系统数据加载电路从ROM中读取引导载入程序，通常称为基本输入输出系统（Basic Input Output System，BIOS），并将其载入内存运行。然后通过 BIOS，将硬盘中的操作系统引导程序（通常为 system.efi）载入内存运行，利用该引导程序，将操作系统读入内存运行。之后通过操作系统，管理计算机的硬件资源，使我们方便的使用计算机进行各种操作。

现在，较为主流的操作系统有 Windows，Linux，Android，IOS 等。

在 Windows 系统中，通过任务管理器，可以看出操作系统管理着各种进程的信息。通过设备管理器，可以看出操作系统对各种设备也进行着相应管理。我们可以忽略这些设备的硬件信息，方便的使用这些设备。如图 27、28 所示



The image shows a screenshot of the Windows Task Manager application. The 'Processes' tab is selected, displaying a list of running processes with columns for Name, Status, CPU usage, Memory usage, Disk usage, and Network usage. The processes listed include XenLoader (32 bit), Task Manager (2), WXWork.exe (32 bit) (7), Google Chrome (9), WMI Provider Host, Desktop Window Manager, and Winrdlv3 (X64).

名称	状态	35% CPU	85% 内存	3% 磁盘	0% 网络
XenLoader (32 位)		22.0%	190.3 MB	101.3 M...	0 Mbps
> 任务管理器 (2)		3.4%	44.1 MB	0 MB/秒	0 Mbps
> WXWork.exe (32 位) (7)		2.3%	91.9 MB	0.1 MB/秒	0 Mbps
> Google Chrome (9)		1.6%	321.5 MB	0 MB/秒	0 Mbps
WMI Provider Host		1.1%	15.8 MB	0 MB/秒	0 Mbps
桌面窗口管理器		0.9%	82.7 MB	0 MB/秒	0 Mbps
Winrdlv3 (X64)		0.6%	1.2 MB	0 MB/秒	0 Mbps

图 27 windows 任务管理器



图 28 windows 设备管理器

在 Linux 系统中，也可使用 top 命令查看进程使用情况，如图 29 所示。Android，IOS 系统等，也可使用相应的系统信息查询软件，查看操作系统执行的具体功能。

```
top - 10:20:23 up 1:20, 0 users, load average: 29.32, 29.57, 29.86
Tasks: 295 total, 2 running, 293 sleeping, 0 stopped, 0 zombie
%Cpu(s): 6.4 us, 4.6 sy, 0.0 ni, 87.6 id, 0.0 wa, 0.0 hi, 1.4 si, 0.0 st
MiB Mem : 1240.8 total, 106.5 free, 781.7 used, 352.6 buff/cache
MiB Swap: 900.0 total, 899.2 free, 0.8 used, 389.4 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
712	root	5	-15	223576	55032	14584	R	12.8	4.3	3:46.12	com.webos+
2896	root	20	0	51172	9948	2608	S	9.8	0.8	5:33.54	wmpChart-+
1987	root	20	0	130320	5488	3948	S	3.0	0.4	2:42.57	lginput2
1203	root	20	0	85872	3984	2772	S	2.3	0.3	1:43.86	micomserv+
453	root	20	0	317216	49972	12928	S	2.0	3.9	0:52.20	surface-m+
26990	root	rt	0	5440	1532	912	R	1.6	0.1	0:00.19	top
1629	root	20	0	0	0	0	S	1.3	0.0	1:14.27	hdmi_task
476	root	20	0	664728	29744	12588	S	1.0	2.3	0:59.52	legacy-br+
891	root	20	0	282108	5840	3928	S	1.0	0.5	1:05.62	pqcontrol+

图 29 linux top 查看进程信息

### 计算机网络

通过操作系统，我们可以方便的使用计算机完成我们需要的操作。然而，当前仅为单机操作，不同计算机之间无信息交互。若需在不同计算机之间传输数据，只能使用外接设备如 U 盘、硬盘等。此外，若不同计算机物理距离较远，则数据传输过程会变得较为困难。在计算机主板中，CPU、内存等设备利用总线进行数据传输。故在不同计算机进行数据传输时，可模仿主板中的方式，在不同的计算机之间通过

线缆连接并进行数据传输，这样就可较好解决数据远距离传输的问题，同时也方便了数据在不同计算机之间交互。

若使用以上方法，首先需考虑不同计算机之间传输信息的线缆材质，什么线缆具有高的抗干扰性，可以在传输过程中不丢失数据，同时保证高效的传输速度。其次，传输的数据是否要进行加密，以保证数据安全。是否要合理编码，以保证数据具有较高的信息量。若两个地区不能直接通过线缆相连，是否可以建立中转区域，使数据成功传输，同时，否可以按区域划分，减少中转区域的数量，以简化传输结构。之后，在不同计算机之间进行输入传输，如何对计算机进行合理编号，以区别不同计算机。此外，当数据传输至计算机中，如何区分该数据应传输给何种应用程序。

计算机网络通过分层结构较为合理的实现了不同计算机之间的数据传输任务。计算机网络通过物理层、数据链路层保证了线缆的一些性质，通过网络层、传输层保证了数据合理编码，合理在不同计算机之间传输，通过应用层，保证了数据到达计算机后分配至具体应用。

除了通过线缆有线传输数据，随着计算机网络技术的发展，又出现了无线传输技术，例如 wlan、蓝牙等。通过 wlan、蓝牙等，可使手机、耳机、平板电脑、鼠标、键盘等方便的接入网络，以进行更为快捷数据交互与传输。

## 数据结构

计算机网络方便了数据在不同计算机之间的传输。操作系统屏蔽了硬件的具体细节，使我们可以方便的使用计算机。编译原理将高级语言编译为机器码，使我们方便的编写应用程序。

在计算机网络中，多台计算机之间使用线缆相互连接，若将多台计算机抽象为多个节点，并在这些节点之间连线，可构成一种“图”结构。在操作系统中，我们可以按文件类别建立多级文件夹，之后将文件存入对应的文件夹中。若将文件夹及文件抽象为节点，并按照打开文件夹顺序连线，可构成一种“树”结构。此外，操



作系统也管理着计算机中的不同进程，同时，这些进程相互间一般没有特殊关系。若将这些进程抽象为节点，可进行顺序排列，构成一种“线性”结构。在编译原理中，也会利用大量的“树”结构，对高级语言进行转换。如图 30 所示。

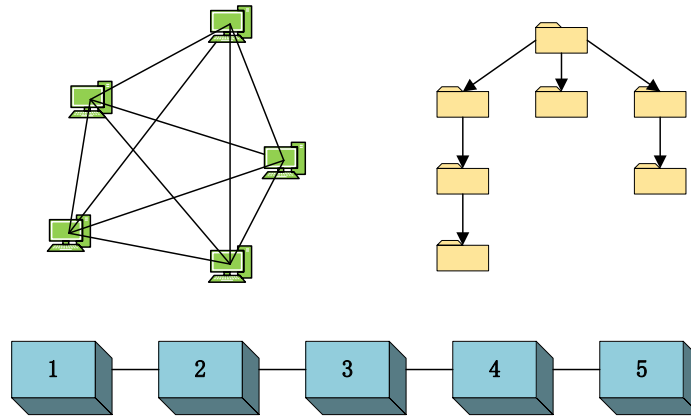


图 30 非线性结构（图、树）与线性结构

在计算机系统中，经常会抽象出各种结构。数据结构既相互之间存在一种或多种特定关系的数据元素的集合，即带“结构”的数据元素的集合。“结构”一般指数据元素之间存在的关系，可分为逻辑结构和存储结构。

逻辑结构：集合、线性结构、非线性结构（树、图等）。

存储结构（数据在内存中的存储方式）：顺序存储、链式存储。

假设有以下数据结构：

```

Typedef struct Link_Memory* Link_Memory_point;
Typedef struct Link_Memory
{
    Link_Memory_Data data;
    Link_Memory_point m_point;
}Link_Memory;
Link_Memory* link_memory;
Memory memory[4];
    
```

其中，Link\_Memory\_Data 为自定义数据类型（假设为 char）。同时假设 memory=0x2001, link\_memory=0x2005，则在内存中有以下关系，如图 31：

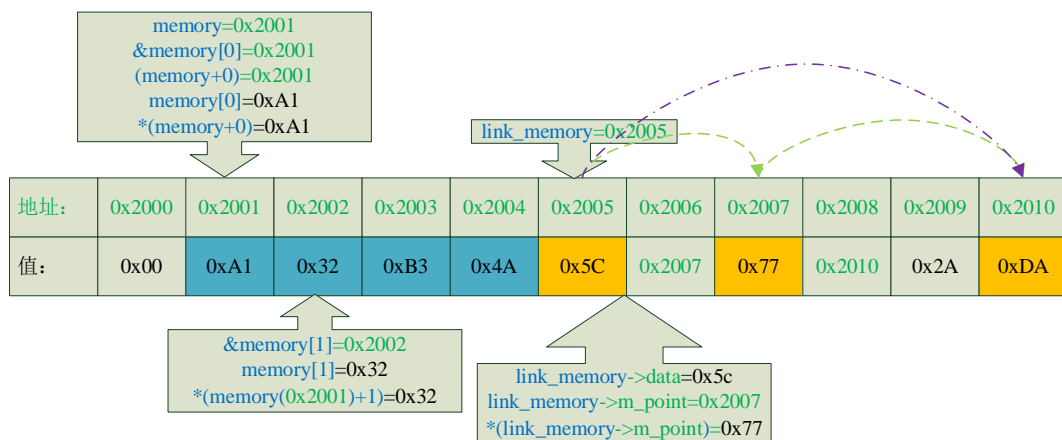


图 31 内存数据关系图

其中，memory 为顺序存储，link\_memory 为链式存储。顺序存储由于其存储地址连续，可以方便的计算每个存储空间地址和对应的值。然而，若需删除中间存储空间的数据，需要移动其他数据覆盖要删除的数据，较为浪费时间（例如删除 0x32）。链式存储其地址空间不一定连续，若要访问对应的存储空间和该存储空间对应的值，需从头节点开始遍历，较为浪费时间。然而，若要删除数据，只需移动对应指针，并释放相应空间内存（例如删除 0x77）。此外，链式存储需要额外的内存空间存储下一个数据对应的地址值，以便访问下一个地址值和其对应数据。

根据以上定义的 Memory 数据结构，可方便的定义一些常用的数据结构，例如数组，链表，栈，队列，树，图，散列，堆，不相交集等。这些常用数据结构既可采用顺序存储，也可采用链式存储。需要根据不同的应用场景，选择合适的存储方式。

## 算法设计与分析

当使用高级语言编写程序时，可将实际问题提供的数据抽象为某一种数据结构。在抽象过程中，通常需要选择合适的数据结构，使我们编写的程序运行速度更快，占用资源更少。一般，可使用该数据结构对应的算法对我们程序的运行速度和资源占用情况进行度量。

算法（Algorithm）是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令，算法代表着用系统的方法描述解决问题的策略机制。换句话说，算法能够对一定规范的输入，在有限时间内获得所要求的输出。如果一个算法有缺陷，或不适合某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间，空间或效率来完成同样的任务。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。复杂度可使用渐进界描述，分为渐进上界( $O$ )、渐进下界( $o$ )和渐进紧确界( $\theta$ )。

算法中的指令描述的是一个计算，当其运行时能从一个初始状态和（可能为空的）初始输入开始，经过一系列有限而清晰定义的状态，最终产生输出并停止于一个终态。一个状态到另一个状态的转移不一定是确定的。随机化算法在内的一些算法，包含了一些随机输入。

一个算法应该具有以下五个重要的特征：

1. 有穷性（Finiteness）：算法必须能在执行有限个步骤之后终止。
2. 确切性（Definiteness）：算法的每一步骤必须有确切的定义。
3. 输入（Input）：一个算法有零个或多个输入，以刻画运算对象的初始情况，所谓零个输入是指算法本身定出了初始条件。
4. 输出（Output）：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。
5. 可行性（Effectiveness）：算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步骤，即每个计算步骤都可以在有限时间内完成（也称之为有效性）。

通常会使用以下常见思想设计一个针对具体数据结构的算法：穷举法、递推法（迭代法）、递归法、分治法、动态规划法、贪心算法、分支界限法、回溯法。此

外，还有一些针对数学公式算法。例如：数论算法、代数算法、计算几何算法、数值分析算法、机器学习算法、凸函数优化算法（梯度下降法、最小二乘法等）、神经网络算法。

## 自动机与计算理论

我们可以针对我们设计的数据结构，选择合适的算法完成程序设计。然而，针对某一些实际问题，可能没有合适的数据结构。或者，我们对该问题选择某种相对合适的数据结构，但没有时间复杂度或空间复杂度合适的算法。

针对这类问题，需考虑这类问题可不可以被计算机计算。之后可以进一步思考，什么问题可以被计算机计算，可以被计算的问题其问题复杂度是多少，如何衡量问题的复杂度。

分析一个问题可不可以被计算，其问题复杂度多少，可采用理论计算模型图灵机和其他图灵机的变种自动机，如图 32。

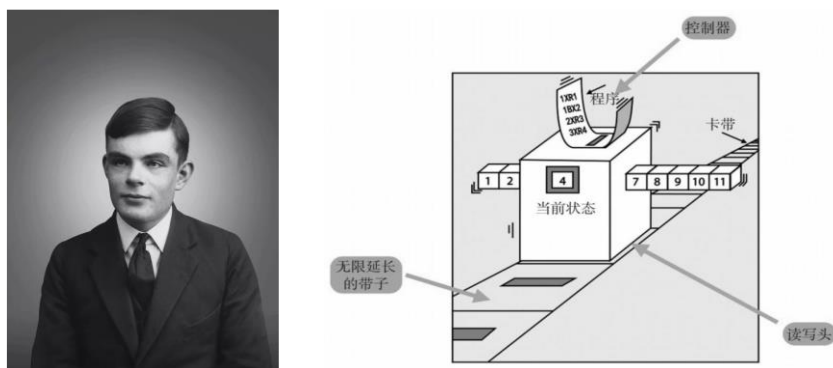


图 32 图灵及图灵机

研究一个问题的可计算性和复杂度的理论也称为可计算理论。在可计算理论中，通常将问题分为以下四类：

- 1) 多项式复杂度的确定性问题（P 问题）
- 2) 多项式复杂度的非确定性问题（NP 问题）

- 3) 多项式复杂度的非确定性难问题（NPH 问题）
- 4) 多项式复杂度的非确定性完全问题（NPC 问题）

这四类问题可能存在以下关系，如图 33：

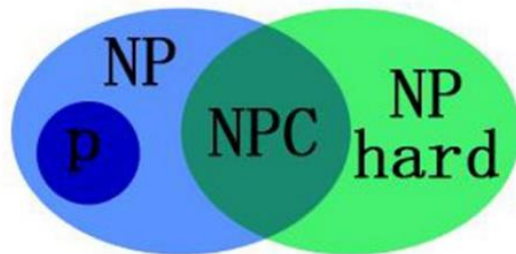


图 33 问题复杂性

其中，NPC 问题是目前研究较多的一类问题。因为很多 NPC 问题都是对现实生活的实际问题进行的抽象。若存在解法，实际问题就有解法。此外，NPC 问题有一个比较好的性质：若某一个 NPC 问题可在多项式时间内求解，则所有 NPC 问题均可在多项式时间内求解。目前学术界研究重点为  $P=NP?$ ，同时该问题也是千禧年数学难题之一。

## 数据库理论

通过操作系统，可将数据以文件的方式存储在硬盘中，例如文本文件、音频文件、视频文件等。然而，随着现在互联网的发展，数据量与日俱增，若将大量数据都以文件形式存储在硬盘中，可能出现以下问题：

1. 数据冗余或数据不一致
2. 数据访问困难
3. 数据孤立或不完整

#### 4. 数据并发访问问题

#### 5. 数据安全性与原子性问题

为了避免以上文件存储的缺点，更好的存储海量数据，通常使用数据库对抽象的数据进行存储。同时，使用由数据实例、数据模式、数据模型、数据库语言构成的数据库管理系统对数据库进行控制、管理，更好的实现数据“增”、“删”、“改”、“查”操作。数据库与数据库管理系统组合也叫做数据库系统。

数据->数据库(关系型)->数据库管理系统->数据库系统。

数据库理论通过定义数据实例、数据模式、数据库语言、选取数据模型，帮助我们设计合理的数据库管理系统。同时，为了对数据进行更优化的“增”、“删”、“改”、“查”操作，数据库管理系统中还使用了存储优化、查询优化、事务管理等技术。使用数据库存储数据时，利用存储优化技术，可以帮助我们选择合适的数据结构。例如线性、树、图、散列等。使用数据库查询数据时，数据库通常提供“选择”，“投影”，“连接”，三种操作。这三种操作可组成六种数据查询方式，利用查询优化技术，可以帮助我们选择一种最优的查询数据方式。使用数据库操作数据时，利用事务管理技术，可保证该操作的原子性、隔离性、一致性、持久性。此外，通过该理论，数据也进行了合理抽象，数据信息可以更好的存储至数据库。

目前，常用的数据模型为关系数据模型，数据库语言为 SQL 语言，数据库管理系统有：SQL Server、MySQL、SQLite、Oracle 等。

### 软件工程

目前我们可以使用各种各样的高级语言结合合适的数据结构和算法，基于操作系统，编写可以实现特定功能的程序。之后，可将这些程序进一步组合，形成更为复杂的软件系统。一款功能多样的软件通常是由很多开发人员一起编写，每个开发人员负责一个或多个模块。



然而，随着个人、企业和政府的信息技术需求日益复杂，过去几个人可以构建的计算机软件，现在则需要一个庞大的团队进行管理，需要一种高效的方法对软件进行管理。同时，个人、企业和政府在进行日常运作管理以及战略战术决策时越来越依靠软件。软件失效会给个人和企业带来诸多不便，甚至是毁灭性的打击。故需要保证软件的高质量性。其次，当要开发一个新的应用领域软件系统时，随着软件开发人员的增多，很多时候，每个开发人员对发布的软件应该具备什么样的特性和功能可能有着不同看法。故在实现软件产品前，各个开发人员应尽力理解该软件需要解决的问题，通过适当协调，最终达到一个统一的目标。然后，随着使用软件用户群体的增加，软件的适应性和可扩展性需求也会同时增加，故软件需具备一定的可维护性。

基于上述软件开发中的一些事实，我们需要制定一种系统化的、规范的、可量化的方法应用于软件的开发、运行、维护。也就是将一种工程化的方法应用于软件开发。而软件工程，就是对这种工程化的方法进行研究。

目前，软件工程通常对软件过程、软件建模、软件质量管理、软件项目管理等几部分进行研究。

在软件过程中，通常研究软件过程模型、软件敏捷开发方法。在软件建模中，通常研究如何制作软件原则、如何充分理解软件需求，对软件产品进行合理建模。在软件质量管理中，通常研究软件质量定义、软件质量评审方法、软件质量保证方法、软件质量测试方法。在软件项目管理中，通常研究项目管理定义、项目管理度量方法、项目预算估计方法、项目进度安排方法、项目风险管理方法、项目维护方法。

最终，通过软件工程，使软件开发方法更加完善，使开发的软件更加健壮、高质量。

## 计算机应用

### 机器学习

通过上述计算机系统结构的知识，我们可以大致了解构造一台个人计算机的过程及高级语言至计算机指令的执行过程。构造完成一台计算机，我们就可以使用该计算机完成我们实际的计算任务。例如，可以计算某个方程的解，某个函数的导数、积分等数学问题。由于计算机的运算速度远大于人类的运算速度，使用计算机计算某个数学问题，可以大大的减少人工计算时间，从而提高工作效率。

于此同时，在 20 世纪 40 年代和 50 年代，来自不同领域（数学，心理学，神经科学，工程学，经济学和政治学）的一批科学家开始探讨制造人工大脑的可能性。在此期间，出现了基于符号推理的符号主义和基于模拟人类神经元关系的联结主义。符号主义由于其逻辑推理的复杂性，慢慢的退出了人工智能研究的舞台。在联结主义中，最简单的算法是感知机算法。然而，由于单层感知机无法解决非线性分类问题，很多分类识别任务不能很好的去完成，不能像人类一样智能，这也使得联结主义停滞的一段时间。随着对人工智能理论的继续探索，科学家发现多层感知机可以很好解决非线性分类问题，使得联结主义再次回到我们的关注中。基于多层感知机，科学家们提出了前馈神经网络、卷积神经网络、循环神经网络等一系列智能感知算法，并发明了反向传播算法，结合梯度下降等优化算法，优化各类神经网络，使其具有强大的感知能力。除了神经网络算法，还发明了决策树、支持向量机、聚类、朴素贝叶斯、EM 算法等智能算法。然而，即使使用计算机计算这些优化公式，由于公式的复杂性，还是会耗费我们不可接受的时间，联结主义再次停滞了下来。之后，随着计算机软、硬件技术的发展（摩尔定律），CPU 性能越来越强，GPU 的出现，使得神经网络优化等公式可以在合理的时间计算完成。联结主义再次发展，并重新命名为深度学习，再次回到人工智能领域。同时，随着互联网，移动互联网的发展，我们可以获取的数据量也无限激增。激增的数据量和软、硬件技术的发展，使深度学习算法成为了目前主流的人工智能算法，在很多领域都有着一定的应用。

机器学习可以简单理解为通过某种算法，让机器获得智能（感知能力、认知能力等）。与计算机系统类算法不同，机器学习算法大多从神经科学、心理学、数学等领域演化而来，最终抽象为某些数学公式表示。之后，通过计算机，完成这些数学公式的计算与解析。目前，由于计算机系统的高度抽象，我们只需要设计我们自己的机器学习算法，将抽象出的数学公式输入计算机，而不需要了解这些公式如何存储在计算机中，计算机如何对这些公式进行计算，即可获得机器学习算法输出结果。例如，目前较常用的 `sklearn`、`pytorch` 框架等。我们只需要利用 `sklearn`、`pytorch` 等输入一些程序语句，即可实现感知机、支持向量机、神经网络等算法。而不需要了解存储神经网络、存储支持向量机中间结果需要何种数据结构，计算机系统需要如何做具体运算。

## 参考文献

- [1]. 康华光. 电子技术基础:模拟部分[M]. 高等教育出版社, 2013.
- [2]. 阎石. 数字电子技术基础[M]. 高等教育出版社, 2016.
- [3]. 王爽. 汇编语言[M]. 清华大学出版社, 2013.
- [4]. DavidA.Patterson, JohnL.Hennessy 等. 计算机组成与设计:硬件、软件接口[M]. 机械工业出版社, 2015.
- [5]. HennessyJL, PattersonDA. 计算机体系结构 : 量化研究方法 [M]. 人民邮电出版社, 2013.
- [6]. CharlesPetzold. 编码:隐匿在计算机软硬件背后的语言 [M]. 电子工业出版社, 2012.
- [7]. RandalE.Bryant. 深入理解计算机系统[M]. 机械工业出版社, 2016.
- [8]. AlfredV.Aho. 编译原理[M]. 机械工业出版社, 2009.
- [9]. AndrewS.Tanenbaum. 现代操作系统[M]. 机械工业出版社, 2009.

- [10]. JamesF.Kurose, KeithW.Ross. 计算机网络:自顶向下方法 [M]. 机械工业出版社, 2014.
- [11]. MarkAllenWeiss. 数据结构与算法分析:C 语言描述[M]. 机械工业出版社, 2004.
- [12]. ThomasH.Cormen. 算法导论[M]. 机械工业出版社, 2014.
- [13]. Michael Sipser. 计算理论导引[M]. 机械工业出版社, 2015.
- [14]. AbrahamSilberschatz, HenryF.Korth, S.Sudarshan. 数据库系统概念.[M]. 机械工业出版社, 2012.
- [15]. RogerS.Pressman. 软件工程:实践者的研究方法 [M]. 机械工业出版社, 2016.
- [16]. 李航. 统计学习方法[M]. 清华大学出版社, 2012.
- [17]. Ian Goodfellow. 深度学习[M]. 机械工业出版社, 2022.

此外，部分背景介绍及文档图片来源于维基百科、百度百科、百度图片等。

To Be Continued.