计算机是如何运行的

作者: 李晓峰

邮箱:CY_LXF@163.COM

个人课程主页:HTTPS://BUUER_XXTXIAOFENG.GITEE.IO/LXF/

前言

2014 年开始在高校任教，慢慢的觉得学信息类的学生应该对计算机的运行机理有一个大致的了解，于是在上各种课程时，都会把计算机的运行机理抽出或多或少的时间进行讲解，这个讲解过程中需要查阅大量的资料，涉及面非常广，也看到一些国内外学者写的类似这样的书籍，但是或多或少与我的思路有点不同，也希望按自己的思路进行整理编著一本尽量薄的书，使得刚开始学习的学生或者想对计算机运行机理进行了解的人，能够快速有个了解。

在根据思路整理材料的过程，在互联网上共享出来的大量资料，对此书的编著提供了非常大的帮助，能直接用的我就直接引用了，并且给出了链接，在此表示感谢，如有侵权，请联系我，我会替换掉相应内容。

另外写出来的文字尽量短，但是对于很多初学者来说，可能一句话里面就有很多基本概念，希望读者在读的时候能够真的理解了这句话在讲什么，不了解的概念可以通过在互联网搜一下。通过了解一个个基本概念，并且通过本书能够将这些概念的基本关系构建起来，对于初学者来说，应该会对计算机系统一个系统认识。

Hope so.

李晓峰

Contents

1 中央处理器 **CPU** **1**
- 1.1 晶体三极管 . . . 1
- 1.2 与非门 . . . 5
- 1.3 锁存器 . . . 7
  - 1.3.1 RS 锁存器 . . . 7
  - 1.3.2 D 锁存器 (电平触发) . . . 7
  - 1.3.3 D 触发器 (边沿触发) . . . 9
- 1.4 加/减法器 . . . 9
- 1.5 冯诺依曼体系结构 . . . 11
- 1.6 算术逻辑单元 ALU . . . 12
- 1.7 指令执行 . . . 13
  - 1.7.1 PC 寄存器 . . . 16
- 1.8 现代 CPU 架构 . . . 17
  - 1.8.1 8086 . . . 17
  - 1.8.2 Intel Skylake (client) 微架构 . . . 19
- 1.9 启动 . . . 22
  - 1.9.1 Firmware Support Package (FSP) . . . 28
  - 1.9.2 UEFI . . . 28

2 机器码/指令集 **31**

3 汇编语言 **35**

4 可执行程序/文件 **37**

5 高级语言 **39**
- 5.1 C . . . 39
- 5.2 VB . . . 42
- 5.3 C++ . . . 42
- 5.4 C ♯ . . . 45
- 5.5 Java . . . 45

5.6 Python 50

6 操作系统 **55**

A IC 产业链 **59**

Appendices **59**

List of Figures

1.1 三极管逻辑示意图[2] 2
1.2 三极管测试电路[2] 2
1.3 三极管逻辑示意图 [2] 3
1.4 三极管测试电路 [2] 3
1.5 集成电路中的三极管在电子显微镜下的照片 4
1.6 三极管的两种封装 4
1.7 非运算电路的逻辑符号和基本电路[5] 6
1.8 与运算电路的逻辑符号和基本电路[5] 6
1.9 RS 锁存器的原理图 [6] 8
1.10 D 型锁存器的原理图 [7] 8
1.11 D 型边沿触发器的原理图 9
1.12 全加器的逻辑电路实现 11
1.13 1 bit ALU 逻辑电路图 13
1.14 由 1 bit ALU 组成 4 bit ALU 逻辑电路图[10] 15
1.15 由 1 bit ALU 组成的 1 bit CPU 16
1.16 8086 系统架构图[12] 19
1.17 8086 系统架构图[12] 20
1.18 Skylake(client) 微架构[14] 21
1.19 Skylake(client) 双核 SOC 框图[14] 21
1.20 Skylake(client) 四核 SOC 框图[14] 22

2.1 ARM 实现的指令集 32

5.1 丹尼斯·里奇 (Dennis MacAlistair Ritchie) 被世人尊称为“C 语言之父”“Unix 之父”，C 语言的诞生是现代程序语言革命的起点，今天，C 语言依旧在系统编程、嵌入式编程等领域占据着统治地位，而 Unix 在操作系统的发展历史上也是一个里程碑的系统，其不仅还用于大型机上，而且 Linux 和 MacOS 都是在其基础上发展而来。其获得美国国家技术奖章和图灵奖。 42

5.2 本贾尼•斯特劳斯特卢普（Bjarne Stroustrup，1950 年 6 月 11 日-），丹麦人，计算机科学家，在德克萨斯 A&M 大学担任计算机科学的主席教授。他最着名的贡献就是开发了 C++ 程序设计语言。其个人网站是https://www.stroustrup.com/index.html。 44
5.3 詹姆斯•高斯林（James Gosling），1955 年 5 月 19 日出生于加拿大，Java 编程语言的共同创始人之一，一般公认他为“Java 之父”。1977 年获得了加拿大卡尔加里大学计算机科学学士学位，1983 年获得了美国卡内基梅隆大学计算机科学博士学位。 46
5.4 Java 的 logo。 49
5.5 吉多•范罗苏姆（Guido van Rossum），荷兰计算机程序员，他作为 Python 程序设计语言的作者而为人们熟知。在 Python 社区，吉多•范罗苏姆被人们认为是“仁慈的独裁者（BDFL:Benevolent Dictator For Life）”，意思是他仍然关注 Python 的开发进程，并在必要的时刻做出决定。他在 Google 工作，在那里他把一半的时间用来维护 Python 的开发。2020 年 11 月 12 日，64 岁的 Python 之父 Guido van Rossum 在自己社交网站上宣布：由于退休生活太无聊，自己决定加入 Microsoft 的 DevDiv Team 。 51
5.6 python 的 Logo。 53

A.1 IC 产业链 60

Chapter 1

中央处理器 CPU

1.1 晶体三极管

三极管一个迷人的特性是其开关特性，也就是当输入到某个值是，有输出。

晶体三极管（以下简称三极管）按材料分有两种：锗管和硅管。而每一种又有 NPN 和 PNP 两种结构形式，但使用最多的是硅 NPN 和锗 PNP 两种三极管 (逻辑示意图见1.1),（其中，N 是负极的意思（代表英文中 Negative），N 型半导体在高纯度硅中加入磷取代一些硅原子，在电压刺激下产生自由电子导电，而 P 是正极的意思（Positive）是加入硼取代硅，产生大量空穴利于导电）。两者除了电源极性不同外，其工作

原理都是相同的。[1]

我们以 NPN 为例搭建一个测试电路 (如图1.2)，发现一个非常有意思的物理现象 (如图1.3、1.4)，B、E 两点间的电压 U_{BE} 达到某个数值时 (记为 U_{BER})，三极管导通，U_{CE} 增大，特性曲线右移，但当 $U_{CE} > 1.0V$ 后，特性曲线几乎不再移动。[2]

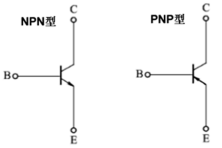

Figure 1.1: 三极管逻辑示意图[2]

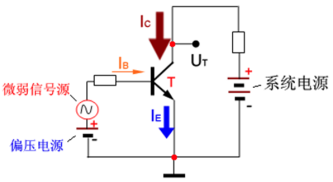

Figure 1.2: 三极管测试电路[2]

当 $I_B = 0$ 时，I_C 趋近 0，称为三极管处于截止状态，相当于开关断开；当 $I_B > 0$ 时，I_B 轻微的变化，会在 I_C 上以几十甚至百多倍放大表现出来；当 I_B 很大时，I_C 变得很大，不能继续随 I_B 的增大而增大，三极管失去放大功能，表现为开关导通。

我们从三级管的电特性上可以看出，三极管核心功能一是放大功能：小电流微量变化，在大电流上放大表现出来；二是开关功能：以小电流控制大电流的通断。[2]

在集成电路中如何使得集成的基本组成单元越来越小，对

[1]引自https://baike.baidu.com/item/

[2]引自“史上最详细图解三极管”http://www.elecfans.com/d/652842.html

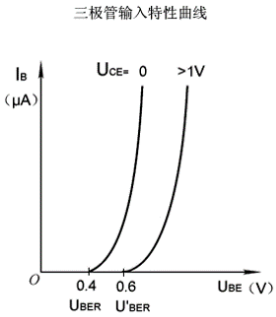

Figure 1.3: 三极管逻辑示意图 [2]

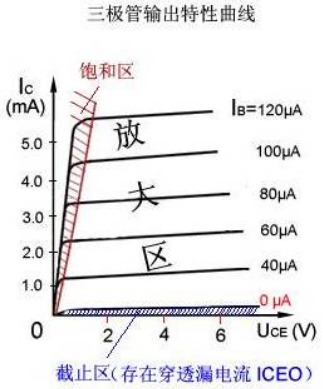

Figure 1.4: 三极管测试电路 [2]

于集成电路小型化和低功耗都非常关键，图1.5[3]是集成电路在电子显微镜下的照片。也有一些三极管作为独立芯片，如1.6[4]。

[3]此图引自http://www.semiconductor-today.com/news_items/2018/jun/epfl_290618.shtml

[4]来自https://www.digikey.cn/product-detail/zh/microchip-technology/TC8020K6-G/TC8020K6-G-ND/4902536

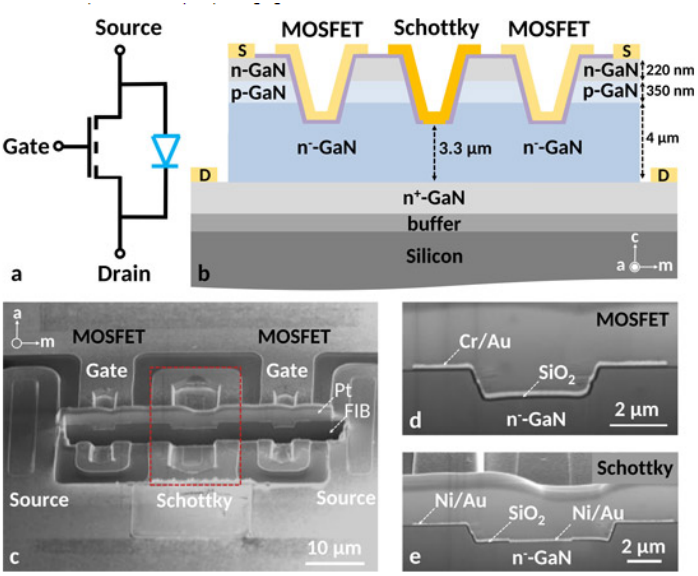

Figure 1: (a) Equivalent circuit. (b) Schematic of integrated vertical MOSFET-Schottky barrier diode (SBD). Scanning electron microscope images of (c) integrated vertical MOSFET-SBD, and (d) cross-sections of integrated vertical MOSFET, and (e) integrated vertical SBD.

Figure 1.5: 集成电路中的三极管在电子显微镜下的照片

Figure 1.6: 三极管的两种封装

1.2 与非门

布尔代数 $< B, and, or, not, 0, 1 >$ 的三个基本运算与或非都可以用三极管表示，这也就意味着，任何布尔代数预算都可以用三极管搭建电路来实施。

我们知道布尔代数的定义了一个代数结构，元素为 0、1，三个基本运算为与、或、非，分别记为 &、-、+，而或可以用非运算表示，所以如果我们可以用三极管电路表示 0,、1，表示非和与运算，是不是就意味着可以搭建任何的代数运算电路？

我们知道，在数字电路中中我们应用的三极管的开关特性，那么我们可以用地电平 (通常是 0v) 表示 0，用导通电压 (通常有 5v、3.6v) 表示 1。

非运算电路如1.7[5]所示，输入为高电平，输出为低，输入为低电平，输出为高，真值表同非运算。与运算电路如1.8所示，两个输入一个输出，其真值表与与运算相同。

[5]图片引自“NPN and PNP logic gates” http://codeperspectives.com/computer-design/npn-pnp-logic-gates/

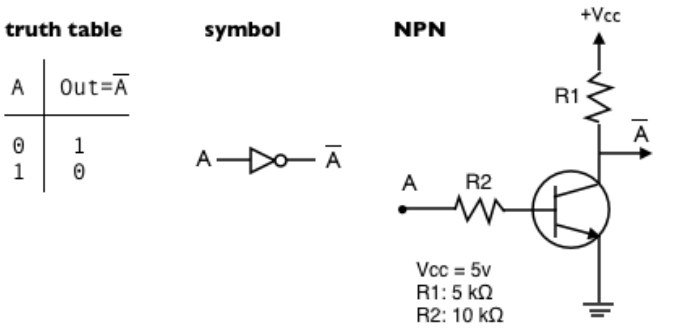

Figure 1.7: 非运算电路的逻辑符号和基本电路[5]

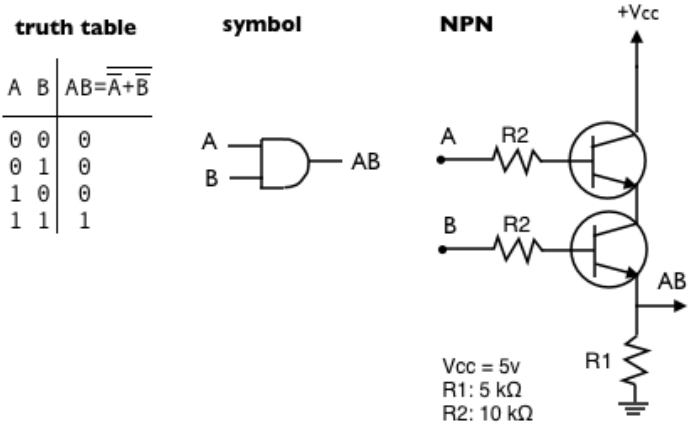

Figure 1.8: 与运算电路的逻辑符号和基本电路[5]

1.3 锁存器

可以用三极管搭建一个电路，进行某个状态的锁存，也就是记忆，解决了计算过程中的存储问题。

1.3.1 RS 锁存器

RS 锁存器 (RS Latch) 实现的电路如图1.9[6]所示：

当 R=1 时，输出为 0，故 R 又称为直接置“0”端，或“复位”端（reset）

当 S=1 时，输出也为 1，故 S 又称为直接置“1”端，或“置位”端

当 R=S=0 时，输出保持不变，这个特性实现了 1bit 的存储

1.3.2 D 锁存器 (电平触发)

D 锁存器 (D-type Latch) 也具有存储功能，电路如图1.10[7]所示：

[6]引自http://www.c-jump.com/CIS77/CPU/Storage/lecture.html

[7]引自https://arith-matic.com/notebook/logic-gates-registers，其他参考见“Constructing Memory Circuits (Registers) with SR-latches” www.mathcs.emory.edu/~jallen/Courses/355/Syllabus/2-seq-circuits/memory.html

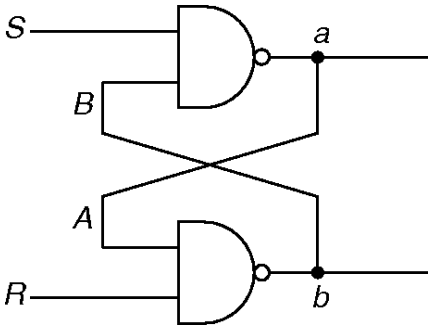

Figure 1.9: RS 锁存器的原理图 [6]

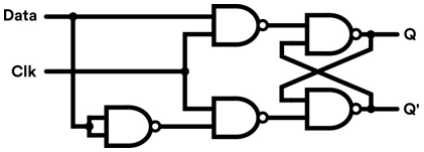

Figure 1.10: D 型锁存器的原理图 [7]

Clk 是时钟信号，当 Clk=1 时，Q=Data；当 Clk=0 时，不管 Data 是什么值，Q 保持不变。电路中 $Q' = notQ$，可以看出 D 型触发器也对状态有存储功能。

1.3.3 D 触发器 (边沿触发)

D 型触发器 (D-type flip-flop) 实现原理如图1.11所示，在这里我们要注意到图中非门的延时效果，我们可以看到只有 clock 端从 0 变成 1 时，才完成一次数据的锁存，也就是在 clock 的上升沿完成锁存，所以我们称其为边沿触发器。

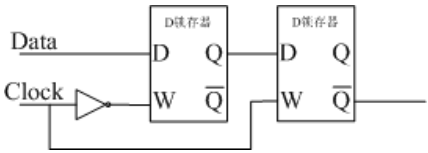

Figure 1.11: D 型边沿触发器的原理图

1.4 加/减法器

可以用门电路构建基本的算术运算，为高级的运算奠定基础。

我们先看看一位 (1bit) 加法器的真值表，如表1.1，其中 A、B 是输入，C_{in} 是进位输入，也可以看做输入，S 是加法器的输出，C_{out} 是进位输出。

Table 1.1: 全加器的真值表

C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

全加器的逻辑电路实现如图1.12，同样我们可以实现减法器等等，任何算术运算电路。

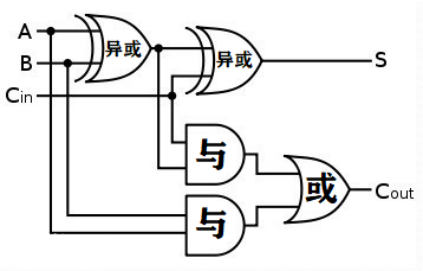

Figure 1.12: 全加器的逻辑电路实现

1.5 冯诺依曼体系结构

> 冯诺依曼和他的同事共同设计了第一台现代意义上的计算机 ENIAC，虽然是一台电子管计算机，但其提出的设计思想被一直沿用下来。

目前的计算机体系结构时冯诺依曼体系结构（von Neumann architecture）[①]，冯诺依曼结构也称普林斯顿结构，是一种将程序指令存储器和数据存储器合并在一起的计算结构，冯诺依曼提出了计算机制造的三个基本原则，即采用二进制逻辑、程序存储执行以及计算机由五个部分组成（运算器、控制器、存储器、输入设备、输出设备），这套理论被称为冯·诺依曼体系结构[8]。

[8]引自https://baike.baidu.com/item/ů

1:

冯·诺依曼 (John von Neumann)

A von Neumann architecture machine, designed by physicist and mathematician John von Neumann (1903–1957) is a theoretical design for a stored program computer that serves as the basis for almost all modern computers. A von Neumann machine consists of a central processor with an arithmetic/logic unit and a control unit, a memory, mass storage, and input and output. [9]

1.6 算术逻辑单元 ALU

ALU 是 Arithmetic and Logic Unit 的缩写，中文通常翻译为“算术逻辑单元”，我们为了用最简单的例子来说明基本原理，下面我们举一个 1bit ALU 的示例。

首先我们要确定设计的 ALU 包括那些算术、逻辑单元，在这里我们假设包括三个单元：加、与、或非、异或，前面我们已经介绍了这四种运算的电路实现。

在每次执行时，我们还要告诉 ALU 执行那种操作，是加？与？或？异或？我们可以同时进行四种运算，然后选择一个输出，也就是我们可以使用数据选择器 (Multiplexer，简写为 MUX) 来实现运算的选择，电路如图1.13[10]所示，

我们假设 MUX 的逻辑如表1.2所示，这时整个 ALU 单元的输入为 M1 M0 B0 A0，如果我们从坐到右排列这四位，并且认为左为最高位，那么知道当输入的四比特是"00**" 形式时，就表示执行加运算，如输入"0010"，ALU 执行"1+0" 运算，而输入的"0010" 是 ALU 可以执行的编码，也就是我们通常说的

[10]图片来自于https://www.exploreembedded.com/wiki/ALU_in_Detail

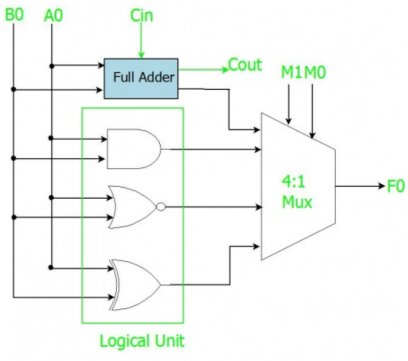

Figure 1.13: 1 bit ALU 逻辑电路图

“机器码", 下面我们把所有类型的机器码都写出来，如表1.3所示。

在上面的例子中，我们在不考虑优化的前提下，可以直接用 1 bit ALU 构建一个 4 bit ALU，如图1.14所示。

1.7 指令执行

下面我们在 1 bit ALU 的基础上构造一个 1 bit CPU(Central Processing Unit, 翻译为中央处理器，或，中央处理单元)，ALU 现在已经能够执行机器码了，根据冯诺依曼结构，CPU 需要完成取指从，然后进行执行，程序是顺序执行的，所以我们需要来记录指令存储的地址，每次执行完后，这个地址需要加 1，得到下一条指令的地址，我们把 CPU 具有存储功能的单元叫寄存器 (register)，而记录取指令的寄存器，我们称为程序计数器 (Program Counter，简写为 PC)，PC 的

Table 1.2: 1 bit ALU 中的 MUX 逻辑

M1	M0	F0 输出
0	0	全加器结果
0	1	与结果
1	0	或非结果
1	1	异或结果

Table 1.3: 1 bit ALU 可执行的机器码

运算名	机器码/指令集			
加	0	0	a	b
与	0	1	a	b
或非	1	0	a	b
异或	1	1	a	b

a,b 是运算的操作数，因为是 1bit，所以可能的取值为 0 或 1.

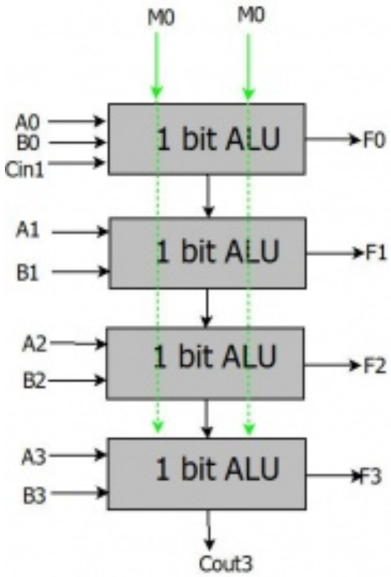

Figure 1.14: 由 1 bit ALU 组成 4 bit ALU 逻辑电路图[10]

位数决定了其可以寻址的范围。

1.7.1 PC 寄存器

我们假设 PC 寄存器是 3bit，这也就意味着 PC 的寻址能力为 $2^3 = 8$，也就是最多有 8 个存储单元，从上面我们 1 bit ALU 的机器指令设计中，我们可以看到一个指令为 4bit，我们可以将每个存储单元设计为 4 比特宽。

我们简化一下冯诺依曼体系架构，执行指令简化为两步，首先要根据 PC 中的地址取指令，用一个时钟，然后在 ALU 中执行，再用一个时钟，执行完一个指令，也就是两个时钟后，PC 要加一，接着再执行，实现框图如1.15所示。

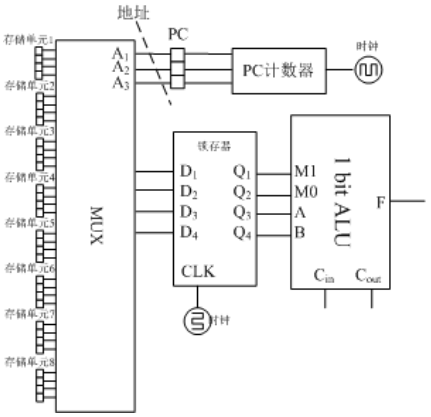

Figure 1.15: 由 1 bit ALU 组成的 1 bit CPU

1.8 现代 CPU 架构

从前面给出的一个极简 1 bit CPU 可以看出，要想让 CPU 能够进行更大规模运算、结构更加简单、运行速度更快、能够适应更广泛的运算，需要至少在以下几方面进行深化设计：

- 连接线路的复用————总线设计技术。
- 能够更快执行复杂运算————增加内部存储单元，我们称为寄存器，使得在运算能够直接在寄存器间进行，这也意味着需要有加载数据到寄存器的指令和寄存器运算指令。
- 执行过程中需要程序跳转————JMP 语句和函数调用，以及相应的现场保护思想、堆栈管理技术、增加相应的寄存器。
- 在执行中断时保存返回现场————现场保护思想、堆栈管理技术、增加相应的寄存器。
- 为了增加程序和数据读取速度———— 增加了 Cache(缓冲区)。

1.8.1 8086

8086[11]是 1978 年英特尔 (Intel) 公司推出的世界上第一个 16 位微处理器，芯片上有 4 万个晶体管，采用 HMOS 工艺制造，用单一的 +5V 电源，时钟频率为 4.77MHz 10MHz。8086

[11]8086 的英文介绍可参考"Internal Architecture of 8086"https://www.eeeguide.com/internal-architecture-of-8086/

有 16 根数据线和 20 根地址线，它既能处理 16 位数据，也能处理 8 位数据，可寻址的内存空间为 1MB，每一个存储单元可以存放一个字节（8 位）二进制信息。[12]

8086 系统架构图如1.16所示，总线接口单元 (BIU, bus interface unit) 包含 4 个段地址寄存器、16 位的指令指针寄存器 IP、20 位的地址加法器、6 字节的指令队列缓冲器。其中 4 个段地址寄存器包含：CS(code segment)16 位的代码段寄存器；DS(data segment)16 位的数据段寄存器；ES(extra segment)16 位的扩展段寄存器；SS(stack segment)16 位的堆栈段寄存器。执行单元执行部件 (EU，execute unit) 包括 8 个通用寄存器 (AX、BX、CX、DX、BP、SP、SI、DI)、4 个数据寄存器 (AX、BX、CX、DX)、2 个地址指针寄存器 (BP,base pointer;SP,stack pointer)、2 个变址寄存器 (SI，source index；DI，destination index)、标志寄存器 FR(flags register)、算术逻辑单元 ALU(arithmetic logic unit)。EU 负责全部指令的执行，同时向 BIU 输出数据（操作结果），并对寄存器和标志寄存器进行管理。在 ALU 中进行 16 位运算，数据传送和处理均在 EU 控制下执行。[12]

BIU 和 EU 的协同：[12]

1. BIU 和 EU 可以并行工作，提高 CPU 效率。BIU 监视着指令队列。当指令队列中有 2 个空字节时，就自动把指令取到队列中。

2. EU 执行指令时，从指令队列头部取指令，然后执行。如

[12] 内容引自https://baike.baidu.com/item/8086

需访问存储器，则 EU 向 BIU 发出请求，由 BIU 访问存储器。

3. 在执行转移、调用、返回指令时，需改变队列中的指令，要等新指令装入队列中后，EU 才继续执行指令。

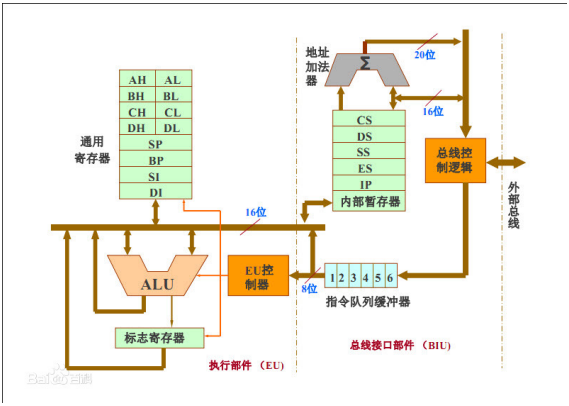

Figure 1.16: 8086 系统架构图[12]

8086 的封装采用双列直插 (通常说的 DIP，dual in-line package), 如图1.17所示，同一芯片会有不同的封装，虽然封装里面的集成电路模块是一样的，有时同一集成电路的不同封装，根据应用的要求，在管脚的引出上会有些许的差别。

1.8.2 Intel Skylake (client) 微架构

随着 CPU 不断加入新的加速执行机制，CPU 的性能不断提高，CPU 的结构也越来越复杂，CPU 已经是一个微系统，这种复杂的 CPU 通常称为微架构 (Microarchitectures)。Intel 的微架构及其对应工艺有

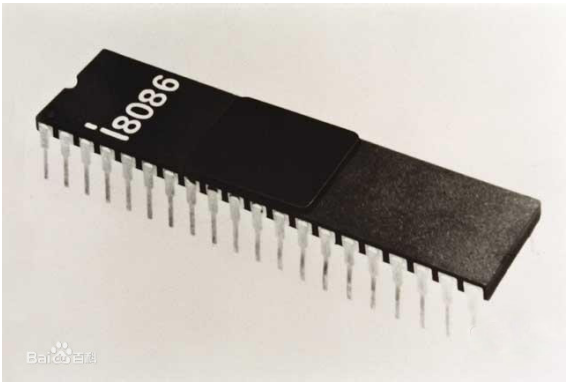

Figure 1.17: 8086 系统架构图[12]

P6(65nm),core(65nm,45nm),Nehalem(45nm,32nm),Sandy Bridge(32nm,22nm),Haswell(22nm,14nm),Skylake(14nm,10nm),Ice Lake(10nm,7nm)。[13]

下面我们看看 sklake 的微架构[14]，其单核微架构如图1.18所示。

Intel 的 Celeron CPU 就是利用 Skylak(client) 微架构搭建的双核 SOC, 总体框图如图1.19所示。

Intel 的 Core i3 CPU 就是利用 Skylak(client) 微架构搭建的四核 SOC, 总体框图如图1.20所示。

[13]此列表信息摘录自https://jcf94.com/2018/02/13/2018-02-13-intel/, 此贴的信息来源其称来自维基百科。

[14]详细介绍可参考https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client)

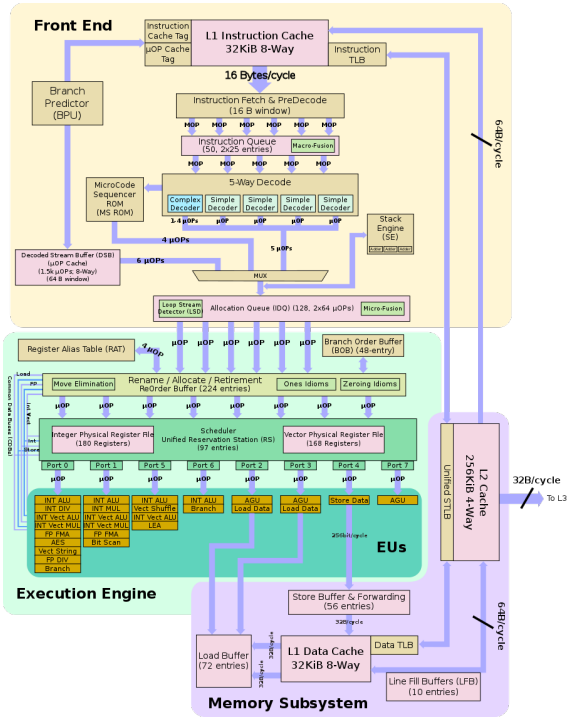

Figure 1.18: Skylake(client) 微架构[14]

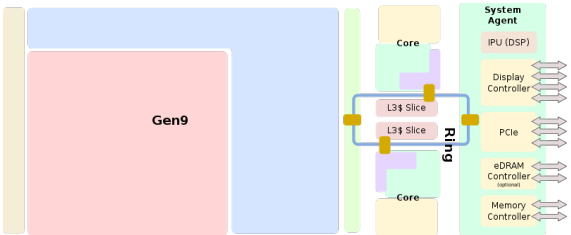

Figure 1.19: Skylake(client) 双核 SOC 框图[14]

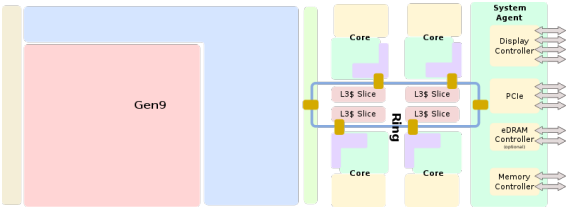

Figure 1.20: Skylake(client) 四核 SOC 框图[14]

1.9 启动

计算机在上电后，CPU 的程序执行寄存器都有一个初始值，这个系统的执行就是从这里开始，而这些信息是需要从 CPU 的设计厂家中获得。在解释计算机启动过程之前，我们先看两个基本概念[15]。

- **BIOS,** 全称 **Basic Input/Output System**，基本输入输出系统

BIOS 是直接与硬件打交道的底层代码，它为操作系统提供了控制硬件设备的基本功能。BIOS 包括有系统 BIOS（即常说的主板 BIOS）、显卡 BIOS 和其它设备（例如 IDE 控制器、SCSI 卡或网卡等）的 BIOS，其中系统 BIOS 是本文要讨论的主角，因为计算机的启动过程正是在它的控制下进行的。BIOS 一般被存放在 ROM(只读存储芯片) 之中，即使在关机或掉电以后，这些代码也不会消失。

- 内存的地址

[15] _tham 的“操作系统启动过程——启动引导 + 硬件自检 + 系统引导 + 系统加载 + 系统登录”，网址https://www.cnblogs.com/tham/p/6827151.html

内存的每一个字节都被赋予了一个地址，以便 CPU 访问内存。32MB 的地址范围用十六进制数表示就是 0 ～ 1FFFFFFH 其中 0～ FFFFFH 的低端 1MB 内存非常特殊，因为最初的 8086 处理器能够访问的内存最大只有 1MB，这 1MB 的低端 640KB 被称为基本内存，而 A0000H～BFFFFH 要保留给显示卡的显存使用，C0000H~FFFFFH 则被保留给 BIOS 使用，其中系统 BIOS 一般占用了最后的 64KB 或更多一点的空间，显卡 BIOS 一般在 C0000H~C7FFFH 处，IDE 控制器的 BIOS 在 C8000H~CBFFFH 处。

下面我们来看看计算机启动的基本流程[15] [16]：

1. 当我们按下电源开关时，电源就开始向主板和其它设备供电，此时电压还不太稳定，主板上的控制芯片组会向 CPU 发出并保持一个 RESET（重置）信号，让 CPU 内部自动恢复到初始状态，但 CPU 在此刻不会马上执行指令。当芯片组检测到电源已经开始稳定供电了（当然从不稳定到稳定的过程只是一瞬间的事情），它便撤去 RESET 信号（如果是手工按下计算机面板上的 Reset 按钮来重启机器，那么松开该按钮时芯片组就会撤去 RESET 信号），CPU 马上就从地址 FFFF0H 处开始执行指令，从前面的介绍可知，这个地址实际上在系统 BIOS 的地址范围内，无论是 Award BIOS 还是 AMI BIOS，放在这里的只是一条跳转指令，跳到系统 BIOS 中真正的启动代码处。

[16] 郑瀚 Andrew_Hann 博客的帖子“Windows 启动过程 (MBR 引导过程分析)”，网址https://www.cnblogs.com/LittleHann/p/6974928.html

2. 系统 BIOS 的启动代码首先要做的事情就是进行 POST（Power — On Self Test，加电后自检），POST 的主要任务是检测系统中一些关键设备是否存在和能否正常工作，例如内存和显卡等设备。由于 POST 是最早进行的检测过程，此时显卡还没有初始化，如果系统 BIOS 在进行 POST 的过程中发现了一些致命错误，例如没有找到内存或者内存有问题（此时只会检查 640K 常规内存），那么系统 BIOS 就会直接控制喇叭发声来报告错误，声音的长短和次数代表了错误的类型。在正常情况下，POST 过程进行得非常快，我们几乎无法感觉到它的存在，POST 结束之后就会调用其它代码来进行更完整的硬件检测。

3. 接下来系统 BIOS 将查找显卡的 BIOS，前面说过，存放显卡 BIOS 的 ROM 芯片的起始地址通常设在 C0000H 处，系统 BIOS 在这个地方找到显卡 BIOS 之后就调用它的初始化代码，由显卡 BIOS 来初始化显卡，此时多数显卡都会在屏幕上显示出一些初始化信息，介绍生产厂商、图形芯片类型等内容，不过这个画面几乎是一闪而过。系统 BIOS 接着会查找其它设备的 BIOS 程序，找到之后同样要调用这些 BIOS 内部的初始化代码来初始化相关的设备。

4. 查找完所有其它设备的 BIOS 之后，系统 BIOS 将显示出它自己的启动画面，其中包括有系统 BIOS 的类型、序列号和版本号等内容。

5. 接着系统 BIOS 将检测和显示 CPU 的类型和工作频率，

然后开始测试所有的 RAM，并同时在屏幕上显示内存测试的进度，我们可以在 CMOS 设置中自行决定使用简单耗时少或者详细耗时多的测试方式。

6. 内存测试通过之后，系统 BIOS 将开始检测系统中安装的一些标准硬件设备，包括硬盘、CD — ROM、串口、并口、软驱等设备，另外绝大多数较新版本的系统 BIOS 在这一过程中还要自动检测和设置内存的定时参数、硬盘参数和访问模式等。

7. 标准设备检测完毕后，系统 BIOS 内部的支持即插即用的代码将开始检测和配置系统中安装的即插即用设备，每找到一个设备之后，系统 BIOS 都会在屏幕上显示出设备的名称和型号等信息，同时为该设备分配中断、DMA 通道和 I/O 端口等资源。

8. 到此，硬件设备检测完毕，系统 BIOS 会重新清屏并在屏幕上显示出一个系统配置表，其中概略列出系统中安装的各种标准硬件设备，及他们使用的资源和一些相关参数。

9. 接下来系统 BIOS 将更新 ESCD（Extended System Configuration Data，扩展系统配置数据）。ESCD 是系统 BIOS 用来与操作系统交换硬件配置信息的一种手段，这些数据被存放在 CMOS（一小块特殊的 RAM，由主板上的电池来供电）之中。通常 ESCD 数据只在系统硬件配置发生改变后才会更新，所以不是每次启动机器时我们都能够看到“Update ESCD…Success”这样的信息，不过，某些主板的系统 BIOS 在保存 ESCD

数据时使用了与 Windows 系统不相同的数据格式，于是 Windows 在它自己的启动过程中会把 ESCD 数据修改成自己的格式，但在下一次启动机器时，即使硬件配置没有发生改变，系统 BIOS 也会把 ESCD 的数据格式改回来，如此循环，将会导致在每次启动机器时，系统 BIOS 都要更新一遍 ESCD，这就是为什么有些机器在每次启动时都会显示出相关信息的原因。

10. ESCD 更新完毕后，系统 BIOS 的启动代码将进行它的最后一项工作，即根据用户指定的启动顺序从软盘、硬盘或光驱启动。

    - BIOS 将磁盘第一个物理扇区 (MBR[2]) 加载到内存。
    - 接着将系统控制权交给 MBR 来进行 (其实就是跳转到加载到内存中 MBR 程序部分)。
    - MBR 运行后，搜索 MBR 中的分区表，查找活动分区（Active Partition）的起始位置。MBR 将活动分区的第一个扇区中的引导扇区 (分区引导记录) 载入到内存。MBR 检测当前使用的文件系统是否可用。
    - MBR 查找操作系统的启动器[3]，并将控制权转交给启动器，由启动器完成操作系统的启动。

2: MBR(Master Boot Record)——主引导记录，位于启动磁盘的第一个扇区，其中主要包含引导代码（Boot Code）和分区表（Partition Table）数据。引导代码主要用于引导系统。而分区表则主要用于标识基本分区和扩展分区。

3: 微软基于 NT 内核的操作系统启动器是 ntldr 文件；DOS 和 win9x 的系统启动器就是分区引导记录，分区引导记录将负责读取并执行 IO.SYS，Windows 9x 的 IO.SYS 首先要初始化一些重要的系统数据，然后就显示出我们熟悉的蓝天白云，在这幅画面之下，Windows 将继续进行 DOS 部分和 GUI（图形用户界面）部分的引导和初始化工作。

下面我们摘录 Intel 的一篇技术文档“Minimal Intel Architecture Boot Loader”[17]中的内容，对计算机的启动有一个

[17] https://www.intel.com/content/www/us/en/intelligent-systems/intel-boot-loader-development-kit/minimal-intel-architecture-boo

大致概况了解。

Power-Up (Reset Vector) Handling

When an IA bootstrap processor (BSP) powers on, the first address that is fetched and executed is at physical address 0xFFFFFFF0, also known as the reset vector. This accesses the ROM / Flash device at the top of the ROM -0x10. The boot loader must always contain a jump to the initialization code in these top 16 bytes.

Mode Selection

The processor must be placed into one of the following modes:

- Real Mode
- Flat Protected Mode
- Segmented Protected Mode (Not Recommended for Firmware)

Refer to the Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A section titled "Mode Switching" for more details.

---

t-loader-paper.html

1.9.1 Firmware Support Package (FSP)

Intel Firmware Support Package (Intel FSP) provides key programming information for initializing Intel silicon and can be easily integrated into a boot loader of the developer's choice. It is easy to adopt, scalable to design, reduces time-to-market, and is economical to build. Components include:

- CPU, memory controller, and Intelő chipset initialization functions as a binary package: Provides silicon initialization ingredients, preserves existing features and frameworks, and fits into existing boot loaders
- Integration guide: Describes the APIs available to communicate with Intel FSP and to integrate it with a boot-loader solution

Intel FSP is designed for integration into a variety of boot loaders, including coreboot and TianoCore (open source Unified Extensible Firmware Interface (UEFI)). [18]

1.9.2 UEFI

The Unified Extensible Firmware Interface (UEFI) Specification, previously known as the Extensible Firmware Interface (EFI) Specification, defines an interface between an

[18] 此 FSP 介绍引自https://software.intel.com/content/www/us/en/develop/articles/intel-firmware-support-package.html

operating system and platform firmware. The interface consists of data tables that contain platform-related information, boot service calls, and runtime service calls that are available to the operating system and its loader. These provide a standard environment for booting an operating system and running pre-boot applications. The UEFI Specification was primarily intended for the next generation of Intelő architecture–based computers, and is an outgrowth of the Intel Boot Initiative (IBI) program that began in 1998. Intel's original version of this specification was publicly named EFI, ending with the EFI 1.10 version.

In 2005, The Unified EFI Forum was formed as an industry-wide organization to promote adoption and continue the development of the EFI Specification. Using the EFI 1.10 Specification as the starting point, this industry group released the follow on specifications, renamed Unified EFI.

Find out more information about UEFI, the UEFI Forum, and the current version of the UEFI Specification at the UEFI Forum Website.[19] [20]

[19] 此英文介绍引自https://software.intel.com/content/www/us/en/develop/articles/unified-extensible-firmware-interface.html

[20] UEFI Forum Website https://uefi.org/

Chapter 2

机器码/指令集

机器码就是 CPU 能够直接执行的二进制串，我们把机器能执行的一个二进制串就称为一条机器指令。指令集就是 CPU 所能执行的所有机器指令的集合。

我们也可以换个角度来看指令集，指令集是在说明如何用软件操控 CPU。

有很多种指令集定义，而这些指令集定义是有知识产权的。

图2.1是 ARM 官网上的截图，其说明 ARM 支持的指令集，每个指令集都会衍生出来一系列或者几个系列的 CPU 芯片[1]。

[1]初学者注意理解这句话，这句话的意思是：定义了一个指令集，而我可以用各种不同的设计来实现这个指令集，只要 CPU 能够执行这些指令，我们就说他支持这个指令集，所以两个 CPU 支持同样的指令集，但是他们运

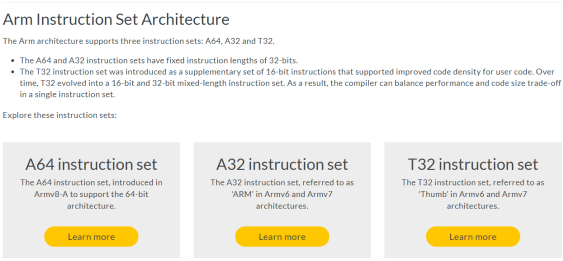

Figure 2.1: ARM 实现的指令集

在 AMD 开发者网站 (http://developer.amd.com/) 中，其对 AMD 64 架构的 CPU 的指令集描述如下[2]:

1.1.3 Instruction Set

The AMD64 architecture supports the full legacy x86 instruction set, with additional instructions to support long mode (see Table 1-1 on page 2 for a summary of operating modes). The application- programming instructions are organized into four subsets, as follows:

General-Purpose Instructions—These are the basic x86 integer instructions used in virtually all programs. Most of these instructions load, store, or operate on data located in the general-purpose registers (GPRs) or memory. Some of the instructions alter sequential program flow by branching

---

行效率可能差别很大

[2]来源于文档“AMD64 Architecture Programmer’s Manual-Volumes 1-5”,文档地址:https://www.amd.com/system/files/TechDocs/40332.pdf

to other program locations.

Streaming SIMD Extensions Instructions (SSE)—These instructions load, store, or operate on data located primarily in the YMM/XMM registers. 128-bit media instructions operate on the lower half of the YMM registers. SSE instructions perform integer and floating-point operations on vector (packed) and scalar data types. Because the vector instructions can independently and simultaneously perform a single operation on multiple sets of data, they are called single- instruction, multiple-data (SIMD) instructions. They are useful for high-performance media and scientific applications that operate on blocks of data.

Multimedia Extension Instructions—These include the MMX technology and AMD 3DNow! technology instructions. These instructions load, store, or operate on data located primarily in the 64-bit MMX registers which are mapped onto the 80-bit x87 floating-point registers. Like the SSE instructions, they perform integer and floating-point operations on vector (packed) and scalar data types. These instructions are useful in media applications that do not require high precision. Multimedia Extension Instructions use saturating mathematical operations that do not generate operation exceptions. AMD has de-emphasized the use of 3DNow! instructions, which have been superceded by their more efficient SSE counterparts. Relevant recommendations are provided in Chapter 5, "64-Bit Media Programming" on

page 239, and in the AMD64 Programmer's Manual Volume 4: 64-Bit Media and x87 Floating-Point Instructions.

x87 Floating-Point Instructions—These are the floating-point instructions used in legacy x87 applications. They load, store, or operate on data located in the 80-bit x87 registers.

龙芯的 CPU 芯片开始使用的 MIPS 指令集架构，2019 年 MIPS 开源，这虽然对于龙芯是利好消息，但是龙芯很注重核心知识产权的自主研发，所以并没有停止其研发的步伐，龙芯在 2020 年 8 月 13 日的全国计算机体系结构学术年会（ACA2020）上，龙芯中科董事长、中科院计算技术研究所研究员胡伟武作了名为《指令系统的自主与兼容》的特邀报告。在报告中，他透露了龙芯的新动向——研发既“自主”又“兼容”的 LoongArch 指令集。如果最终达成目标，这将是一个自带“完整”生态，且中国人能牢牢掌握的体系[3]。

[3]丢掉幻想！龙芯中科将出 LoongArch 自主指令集，深度兼容主流体系，网址https://baijiahao.baidu.com/s?id=1675553831699313296&wfr=spider&for=pc

Chapter 3

汇编语言

直接用机器码或者机器指令进行编程会是一个很繁琐的过程 (并不是不可以)，为了提高开发效率，人们用助记符代替机器指令的操作码，用地址符号或标号代替指令或操作数的地址，汇编语言对应着不同的机器指令，特定的汇编语言和特定的机器语言指令集是一一对应的，不同平台之间不可直接移植，汇编语言与机器指令相比，人书写更方便，但是需要通过汇编器 (Assembler) 将汇编语句转换成机器指令，用汇编语言写的程序通常会有多个，人们编写的一套工具来完成汇编程序到机器指令序列的转换，这个工具我们称为编译器，有不同的汇编语言编译器，如 MASM、NASM 等。

我们通常还会看到一个名词“编辑器 (Editor)”，编辑器通常只用来编辑文本的软件，比如 windows 操作系统自带的 notepad，以及第三方的 notepad plus 等。

还有一个概念就是 IDE，英文全称是 integrated development environment，翻译过来就是“集成开发环境”，从字面意思就可以猜到，这个软件就是把你开发中用到的软件或者功能都集成在一起了，这类软件有 visual Studio、Eclipse、code::block 等，通常一个开发环境会针对一种或者几种开发语言。另外需要强调的一点是，程序的调试是非常重要的，所以通常 IDE 中会集成调试工具，使得程序调试方便些。

Chapter 4

可执行程序/文件

用汇编语言写了程序，然后也编译为机器码了，现在就有个问题“编译后的机器指令是以二进制方式存储在文件中的，我们如何让他执行起来?”，我们知道 CPU 在上电的时候都有一个缺省启动地址，从这个缺省启动地址开始，最终会启动操作系统，我们通常的操作环境都是在操作系统中来操控计算机，所以上面那个问题，我们可以转换为“操作系统中，编译后的机器指令是以二进制方式存储在文件中的，我们如何让他执行起来?”，我们以 windows 为例，看看我们是如何执行一个程序的，比如我们要运行 windows 中的记事本，我们按“win+r”，然后在运行窗口中输入“cmd”，启动命令行终端，在命令行终端中，我们输入"notepad.exe", 操作系统会运行记事本。下面我们概念性地解释一下这个过程。

当你输入 notepad.exe 并回车后，命令行终端知道你是

要执行这个程序，也就是说你输入的文件是一个包含有机器指令的二进制文件，操作系统需要把你这个文件加载到内存中，加载的内存地址他知道，等他把程序都加载进去后，他执行程序跳转指令，跳转到程序的第一行就可以开始执行你要执行的程序了。但是这里有个问题需要解决，操作系统在加载程序的时候，他需要知道如何把文件中的内容加载到内存中，这里面有地址问题，有数据存储的问题，为了使得加载能够顺利进行，定义了可执行程序/文件的格式，这个格式其实就是告诉操作系统如何将文件中包含的代码和数据加载到内存里面，我们通常看到的可执行程序/文件的格式定义有 windows 使用的 PE(Portable Executable) 格式[1]，Linux 中使用的 ELF(Executable and Linkable Format) 格式。

[1]利用工具进行 PE 文件格式的分析，可以帮助深入了解 PE 文件格式，可以参考博客文章“PE 文件格式分析”，网址https://www.cnblogs.com/lqerio/p/11117579.html

Chapter 5

高级语言

汇编语言与机器指令相比，程序员能够更好理解程序的语义，也更好记忆，但是离算法的基本语义单元相比，仍然粒度太小，这就使得利用汇编语言编写实际算法效率任然很低[4]。所以计算机研究人员不断努力，设计出各种不同的计算机语言，希望能够使得程序开发效率更高。

4: 这句话希望读者能够好好理解一下其含义

5.1 C

面向过程
结构化
编译为本地机器码

C 语言诞生于美国的贝尔实验室，由 D.M.Ritchie(C 大的介绍见5.1) 以 B 语言为基础发展而来，在它的主体设计完成后，Thompson 和 Ritchie 用它完全重写了 UNIX，且随着 UNIX 的发展，c 语言也得到了不断的完善。为了利于 C 语言的全面推广，许多专家学者和硬件厂商联合组成了 C 语言标准委员会，并在之后的 1989 年，诞生了第一个完备的 C 标准，简称“C89”，也就是“ANSI c”，截至 2020 年，最新的 C 语言标准为 2017 年发布的“C17”。

C 语言之所以命名为 C，是因为 C 语言源自 Ken Thompson 发明的 B 语言，而 B 语言则源自 BCPL 语言。

1967 年，剑桥大学的 Martin Richards 对 CPL 语言进行了简化，于是产生了 BCPL（Basic Combined Programming Language）语言。20 世纪 60 年代，美国 AT&T 公司贝尔实验室（AT&T Bell Laboratory）的研究员 Ken Thompson 闲来无事，手痒难耐，想玩一个他自己编的，模拟在太阳系航行的电子游戏——Space Travel。他背着老板，找到了台空闲的机器——PDP-7。但这台机器没有操作系统，而游戏必须使用操作系统的一些功能，于是他着手为 PDP-7 开发操作系统。后来，这个操作系统被命名为——UNIX。

1970 年，美国贝尔实验室的 Ken Thompson，以 BCPL 语言为基础，设计出很简单且很接近硬件的 B 语言（取 BCPL 的首字母）。并且他用 B 语言写了第一个 UNIX 操作系统。

1971 年，同样酷爱 Space Travel 的 Dennis M.Ritchie 为了能早点儿玩上游戏，加入了 Thompson 的开发项目，合作开发 UNIX。他的主要工作是改造 B 语言，使其更成熟。

1972 年，美国贝尔实验室的 D.M.Ritchie 在 B 语言的基础上最终设计出了一种新的语言，他取了 BCPL 的第二个字母作为这种语言的名字，这就是 C 语言。

1973 年初，C 语言的主体完成。Thompson 和 Ritchie 迫不及待地开始用它完全重写了 UNIX。此时，编程的乐趣使他们已经完全忘记了那个"Space Travel"，一门心思地投入到了 UNIX 和 C 语言的开发中。随着 UNIX 的发展，C 语言自身也在不断地完善。直到 2020 年，各种版本的 UNIX 内核和周边工具仍然使用 C 语言作为最主要的开发语言，其中还有不少继承 Thompson 和 Ritchie 之手的代码。[1]

C 语言编译器有很多，比如鼻祖级编译器 gcc。

C 语言的 IDE 也很多，比如针对单片机的有 Keil，基于通用 PC 的有 Visual Studio，开源的也有很多，比如 Code::Blocks，也有些程序开发者使用一些通用的编辑器，比如 Notepad++，Sublime 编写源代码，然后利用编译器进行编译，用调试器 (如 gdb) 进行调试。

初学者通过会把下载、安装相应的软件，然后编写一个最简单的“Hello world”程序，来检验其基本开发环境是否构建成功，这种思维方式是我们常用的一种工程思维。

[1] 以上关于 C 语言的介绍文字，来源于百度百科，网址https://baike.baidu.com/item/c%E8%AF%AD%E8%A8%80

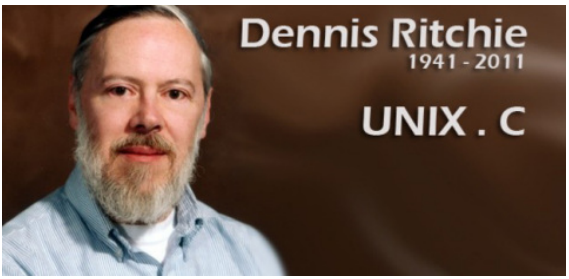

Figure 5.1: 丹尼斯·里奇 (Dennis MacAlistair Ritchie) 被世人尊称为“C 语言之父”“Unix 之父”，C 语言的诞生是现代程序语言革命的起点，今天，C 语言依旧在系统编程、嵌入式编程等领域占据着统治地位，而 Unix 在操作系统的发展历史上也是一个里程碑的系统，其不仅还用于大型机上，而且 Linux 和 MacOS 都是在其基础上发展而来。其获得美国国家技术奖章和图灵奖。

5.2 VB

5.3 C++

20 世纪 70 年代中期，Bjarne Stroustrup(C 大的图片介绍见5.2) 在剑桥大学计算机中心工作[5]。他使用过 Simula 和 ALGOL，接触过 C。他对 Simula 的类体系感受颇深，对 ALGOL 的结构也很有研究，深知运行效率的意义。既要编程简单、正确可靠，又要运行高效、可移植，是 Bjarne Stroustrup 的初衷。以 C 为背景，以 Simula 思想为基础，正好符合他的设想。1979 年，Bjame Sgoustrup 到了 Bell 实验室，开始从事将 C 改良为带类的 C（C with classes）的工作。1983 年该语言被正式命名为 C++。自从 C++ 被发明以来，它经历了 3 次主要的修订，每一次修订都为 C++ 增加了新的特征并作了一些修改。第一次修订是在 1985 年，第二次修订是在 1990 年，而第三次修订发生在 c++ 的标准化过程中。在 20 世纪

5: 有些介绍资料说他是在读博士期间接触到 Simula，如果是这样，那么他应该是在剑桥大学读的博士

90 年代早期，人们开始为 C++ 建立一个标准，并成立了一个 ANSI 和 ISO（Intemational Standards Organization）国际标准化组织的联合标准化委员会。该委员会在 1994 年 1 月 25 曰提出了第一个标准化草案。在这个草案中，委员会在保持 Stroustrup 最初定义的所有特征的同时，还增加了一些新的特征。在完成 C++ 标准化的第一个草案后不久，发生了一件事情使得 C++ 标准被极大地扩展了：Alexander stepanov 创建了标准模板库（Standard Template Library，STL）。STL 不仅功能强大，同时非常优雅，然而，它也是非常庞大的。在通过了第一个草案之后，委员会投票并通过了将 STL 包含到 C++ 标准中的提议。STL 对 C++ 的扩展超出了 C++ 的最初定义范围。虽然在标准中增加 STL 是个很重要的决定，但也因此延缓了 C++ 标准化的进程。委员会于 1997 年 11 月 14 日通过了该标准的最终草案，1998 年，C++ 的 ANSI/IS0 标准被投入使用。通常，这个版本的 C++ 被认为是标准 C++。所有的主流 C++ 编译器都支持这个版本的 C++，包括微软的 Visual C++ 和 Borland 公司的 C++Builder。[2]

关于 C++ 版本迭代的较详细信息，可以参考 github 上 xiaodongQ 维护的一个文档“C 和 C++ 的历史版本迭代整理”[3]。

[2]以上关于 C++ 历史介绍文字，来源于百度文库，网址：https://baike.baidu.com/item/C++

[3]http://xiaodongq.github.io/2019/10/18/CC++/

Figure 5.2: 本贾尼·斯特劳斯特卢普（Bjarne Stroustrup，1950 年 6 月 11 日-），丹麦人，计算机科学家，在德克萨斯 A&M 大学担任计算机科学的主席教授。他最着名的贡献就是开发了 C++ 程序设计语言。其个人网站是https://www.stroustrup.com/index.html。

5.4 C♯

C♯读作C Sharp。最初它有个更酷的名字，叫做COOL。微软从1998年12月开始了COOL项目，Delphi语言的设计者Hejlsberg带领着Microsoft公司的开发团队，开始了第一个版本C♯语言的设计,直到2000年2月，COOL被正式更名为C♯,在2000年9月，国际信息和通信系统标准化组织为C♯语言定义了一个Microsoft公司建议的标准,最终C♯语言在2001年得以正式发布。

C♯一种由C和C++衍生出来的面向对象的编程语言、运行于.NET Framework和.NET Core(完全开源，跨平台)之上的高级程序设计语言，C♯源代码会编译为MSIL(Microsoft Intermediate Language)，也称为通用中间语言（CIL：Common Intermediate Language），是一组与平台无关的指令，在程序执行时，通过JIT(just-in-time)编译器(compiler)将MSIL编译为本地码(native code)，而运行时由安装在系统中的CLR（(common language runtime)提供运行所需环境[4]。

，

5.5 Java

20世纪90年代，硬件领域出现了单片式计算机系统，这种价格低廉的系统一出现就立即引起了自动控制领域人员的注意，因为使用它可以大幅度提升消费类电子产品（如电视

[4]关于程序的运行过程可以参考微软官方网站上的介绍，网址为https://docs.microsoft.com/en-us/dotnet/standard/managed-execution-process

机顶盒、面包烤箱、移动电话等）的智能化程度。Sun 公司为了抢占市场先机，在 1991 年成立了一个称为 Green 的项目小组，帕特里克、詹姆斯·高斯林 (见图5.3)、麦克·舍林丹和其他几个工程师一起组成的工作小组在加利福尼亚州门洛帕克市沙丘路的一个小工作室里面研究开发新技术，专攻计算机在家电产品上的嵌入式应用[5]。

Figure 5.3: 詹姆斯·高斯林（James Gosling），1955 年 5 月 19 日出生于加拿大，Java 编程语言的共同创始人之一，一般公认他为“Java 之父”。1977 年获得了加拿大卡尔加里大学计算机科学学士学位，1983 年获得了美国卡内基梅隆大学计算机科学博士学位。

由于 C++ 所具有的优势，该项目组的研究人员首先考虑采用 C++ 来编写程序。但对于硬件资源极其匮乏的单片式系统来说，C++ 程序过于复杂和庞大。另外由于消费电子产品所采用的嵌入式处理器芯片的种类繁杂，如何让编写的程序跨平台运行也是个难题。为了解决困难，他们首先着眼于语

[5]本节关于 Java 的发展介绍文字来自于https://baike.baidu.com/item/java/85979

言的开发，假设了一种结构简单、符合嵌入式应用需要的硬件平台体系结构并为其制定了相应的规范，其中就定义了这种硬件平台的二进制机器码指令系统（即后来成为“字节码”的指令系统），以待语言开发成功后，能有半导体芯片生产商开发和生产这种硬件平台。对于新语言的设计，Sun 公司研发人员并没有开发一种全新的语言，而是根据嵌入式软件的要求，对 C++ 进行了改造，去除了留在 C++ 的一些不太实用及影响安全的成分，并结合嵌入式系统的实时性要求，开发了一种称为 Oak 的面向对象语言。

由于在开发 Oak 语言时，尚且不存在运行字节码的硬件平台，所以为了在开发时可以对这种语言进行实验研究，他们就在已有的硬件和软件平台基础上，按照自己所指定的规范，用软件建设了一个运行平台，整个系统除了比 C++ 更加简单之外，没有什么大的区别。1992 年的夏天，当 Oak 语言开发成功后，研究者们向硬件生产商进行演示了 Green 操作系统、Oak 的程序设计语言、类库和其硬件，以说服他们使用 Oak 语言生产硬件芯片，但是，硬件生产商并未对此产生极大的热情。因为他们认为，在所有人对 Oak 语言还一无所知的情况下，就生产硬件产品的风险实在太大了，所以 Oak 语言也就因为缺乏硬件的支持而无法进入市场，从而被搁置了下来。

1994 年 6、7 月间，在经历了一场历时三天的讨论之后，团队决定再一次改变了努力的目标，这次他们决定将该技术应用于互联网。他们认为随着 Mosaic 浏览器的到来，因特网正在向高度互动的远景演变，而这一远景正是他们在有线电视网中看到的。作为原型，帕特里克 • 诺顿写了一个小型万维网浏览器 WebRunner。

1995 年，互联网的蓬勃发展给了 Oak 机会。业界为了使死板、单调的静态网页能够“灵活”起来，急需一种软件技术来开发一种程序，这种程序可以通过网络传播并且能够跨平台运行。于是，世界各大 IT 企业为此纷纷投入了大量的人力、物力和财力。这个时候，Sun 公司想起了那个被搁置起来很久的 Oak，并且重新审视了那个用软件编写的试验平台，由于它是按照嵌入式系统硬件平台体系结构进行编写的，所以非常小，特别适用于网络上的传输系统，而 Oak 也是一种精简的语言，程序非常小，适合在网络上传输。Sun 公司首先推出了可以嵌入网页并且可以随同网页在网络上传输的 Applet（Applet 是一种将小程序嵌入到网页中进行执行的技术），并将 Oak 更名为 Java(LOGO 见图5.4)（在申请注册商标时，发现 Oak 已经被人使用了，再想了一系列名字之后，最终，使用了提议者在喝一杯 Java 咖啡时无意提到的 Java 词语）。1995 年 5 月 23 日，Sun 公司在 Sun world 会议上正式发布 Java 和 HotJava 浏览器。IBM、Apple、DEC、Adobe、HP、Oracle、Netscape 和微软等各大公司都纷纷停止了自己的相关开发项目，竞相购买了 Java 使用许可证，并为自己的产品开发了相应的 Java 平台。

1996 年 1 月，Sun 公司发布了 Java 的第一个开发工具包（JDK 1.0），这是 Java 发展历程中的重要里程碑，标志着 Java 成为一种独立的开发工具。9 月，约 8.3 万个网页应用了 Java 技术来制作。10 月，Sun 公司发布了 Java 平台的第一个即时（JIT）编译器。

1997 年 2 月，JDK 1.1 面世，在随后的 3 周时间里，达到了 22 万次的下载量。4 月 2 日，Java One 会议召开，参

Figure 5.4: Java 的 logo。

会者逾一万人，创当时全球同类会议规模之纪录。9 月，Java Developer Connection 社区成员超过 10 万。

1998 年 12 月 8 日，第二代 Java 平台的企业版 J2EE(Java 2Enterprise Edition) 发布。1999 年 6 月，Sun 公司发布了第二代 Java 平台（简称为 Java2）的 3 个版本：J2ME（Java2 Micro Edition，Java2 平台的微型版），应用于移动、无线及有限资源的环境；J2SE（Java 2 Standard Edition，Java 2 平台的标准版），应用于桌面环境；J2EE（Java 2Enterprise Edition，Java 2 平台的企业版），应用于基于 Java 的应用服务器。Java 2 平台的发布，是 Java 发展过程中最重要的一个里程碑，标志着 Java 的应用开始普及。

1999 年 4 月 27 日，HotSpot 虚拟机发布。HotSpot 虚拟机发布时是作为 JDK 1.2 的附加程序提供的，后来它成为了 JDK 1.3 及之后所有版本的 Sun JDK 的默认虚拟机 (也就是我们现在所说的 Java 运行环境 (JRE:java runtime environment))。

随后 Java 得到更广泛的业界支持。

2006 年 11 月 13 日，Java 技术的发明者 Sun 公司宣布，将 Java 技术作为免费软件对外发布。Sun 公司正式发布的有关 Java 平台标准版的第一批源代码，以及 Java 迷你版的可执行源代码。从 2007 年 3 月起，全世界所有的开发人员均可对 Java 源代码进行修改。2009 年，甲骨文 (Oracle) 公司宣布收购 Sun。2010 年，Java 编程语言的共同创始人之一詹姆斯·高斯林从 Oracle 公司辞职。2011 年，甲骨文公司举行了全球性的活动，以庆祝 Java7 的推出，随后 Java7 正式发布。2014 年，甲骨文公司发布了 Java8 正式版。

5.6 Python

Python 由荷兰数学和计算机科学研究学会的吉多·范罗苏姆（Guido van Rossum）于 1990 年代初设计，作为一门叫做 ABC 语言的替代品，Python 由于是一种解释型语言，所以很容易实现跨平台，最初被设计用于编写自动化脚本 (shell)，随着版本的不断更新和语言新功能的添加，逐渐被用于独立的、大型项目的开发。

1989 年圣诞节期间，在阿姆斯特丹，Guido 为了打发圣诞节的无趣，决心开发一个新的脚本解释程序，作为 ABC 语言的一种继承。之所以选中 Python（大蟒蛇的意思）作为该编程语言的名字，是取自英国 20 世纪 70 年代首播的电视喜剧《蒙提·派森的飞行马戏团》（Monty Python's Flying Circus）。

ABC 是由 Guido 参加设计的一种教学语言[6]。就 Guido 本人看来，ABC 这种语言非常优美和强大，是专门为非专业程序员设计的。但是 ABC 语言并没有成功，究其原因，Guido

6: 在国外很多教编译原理的课程都会有一个渐进练习，就是随着课程的推进，会让大家实现编译器的不同功能，最后课程结束，大家就编写了一个语言的编译器。

Figure 5.5: 吉多 • 范罗苏姆（Guido van Rossum），荷兰计算机程序员，他作为 Python 程序设计语言的作者而为人们熟知。在 Python 社区，吉多 • 范罗苏姆被人们认为是“仁慈的独裁者（BDFL:Benevolent Dictator For Life）”，意思是他仍然关注 Python 的开发进程，并在必要的时刻做出决定。他在 Google 工作，在那里他把一半的时间用来维护 Python 的开发。2020 年 11 月 12 日，64 岁的 Python 之父 Guido van Rossum 在自己社交网站上宣布：由于退休生活太无聊，自己决定加入 Microsoft 的 DevDiv Team 。

认为是其非开放造成的。Guido 决心在 Python 中避免这一错误。同时，他还想实现在 ABC 中脑海中闪现过的想法，但未曾实现的东西。

就这样，Python 在 Guido 手中诞生了。可以说，Python 是从 ABC 发展起来，主要受到了 Modula-3（另一种相当优美且强大的语言，为小型团体所设计的）的影响。并且结合了 Unix shell 和 C 的习惯。

Python(Python 的 Logo 如图5.6) 目前（指 2021 年）已经成为最受欢迎的程序设计语言之一。自从 2004 年以后，python 的使用率呈线性增长。Python 2 于 2000 年 10 月 16 日发布，稳定版本是 Python 2.7，Python 3 于 2008 年 12 月 3 日发布，不完全兼容 Python 2。，2011 年 1 月，它被 TIOBE 编程语言排行榜评为 2010 年度语言。

Python 在执行时，首先会将.py 文件中的源代码编译成 Python 的 byte code（字节码），然后再由 Python Virtual Machine（Python 虚拟机）来执行这些编译好的 byte code。这种机制的基本思想跟 Java，.NET 是一致的。然而，Python Virtual Machine 与 Java 或.NET 的 Virtual Machine 不同的是，Python 的 Virtual Machine 是一种更高级的 Virtual Machine。这里的高级并不是通常意义上的高级，不是说 Python 的 Virtual Machine 比 Java 或.NET 的功能更强大，而是说和 Java 或.NET 相比，Python 的 Virtual Machine 距离真实机器的距离更远。

目前 python 有着丰富的库支持，得到广泛应用，如：Web 和 Internet 开发、科学计算和统计、人工智能、桌面界面开

发、网络爬虫、数据统计、脚本编写等等。[6]

Figure 5.6: python 的 Logo。

[6]以上 python 的介绍文字来自于https://baike.baidu.com/item/Python/407313

Chapter 6

操作系统

操作系统（operation system，缩写为 OS），我们给出几种定义，通过这些定义，大家可以理解计算机操作系统概念。

操作系统（operation system，简称 OS）是管理计算机硬件与软件资源的计算机程序。操作系统需要处理如管理与配置内存、决定系统资源供需的优先次序、控制输入设备与输出设备、操作网络与管理文件系统等基本事务。操作系统也提供一个让用户与系统交互的操作界面。[1]

operating system :The low-level software that supports a computer's basic functions, such as scheduling tasks and controlling peripherals.[2]

[1]https://baike.baidu.com/item//192
[2]https://www.lexico.com/definition/operating_system

An operating system is the primary software that manages all the hardware and other software on a computer. The operating system, also known as an "OS," interfaces with the computer's hardware and provides services that applications can use.[3]

An Operating System (OS) is a software that acts as an interface between computer hardware components and the user. Every computer system must have at least one operating system to run other programs. Applications like Browsers, MS Office, Notepad Games, etc., need some environment to run and perform its tasks.The OS helps you to communicate with the computer without knowing how to speak the computer's language. It is not possible for the user to use any computer or mobile device without having an operating system.https://www.guru99.com/operating-system-tutorial.html

[3]https://www.howtogeek.com/361572/what-is-an-operating-system/

Appendices

Appendix A

IC 产业链

集成电路作为半导体产业的核心，市场份额达 83%，由于其技术复杂性，产业结构高度专业化。随着产业规模的迅速扩张，产业竞争加剧，分工模式进一步细化。目前市场产业链为 IC 设计、IC 制造和 IC 封装测试。

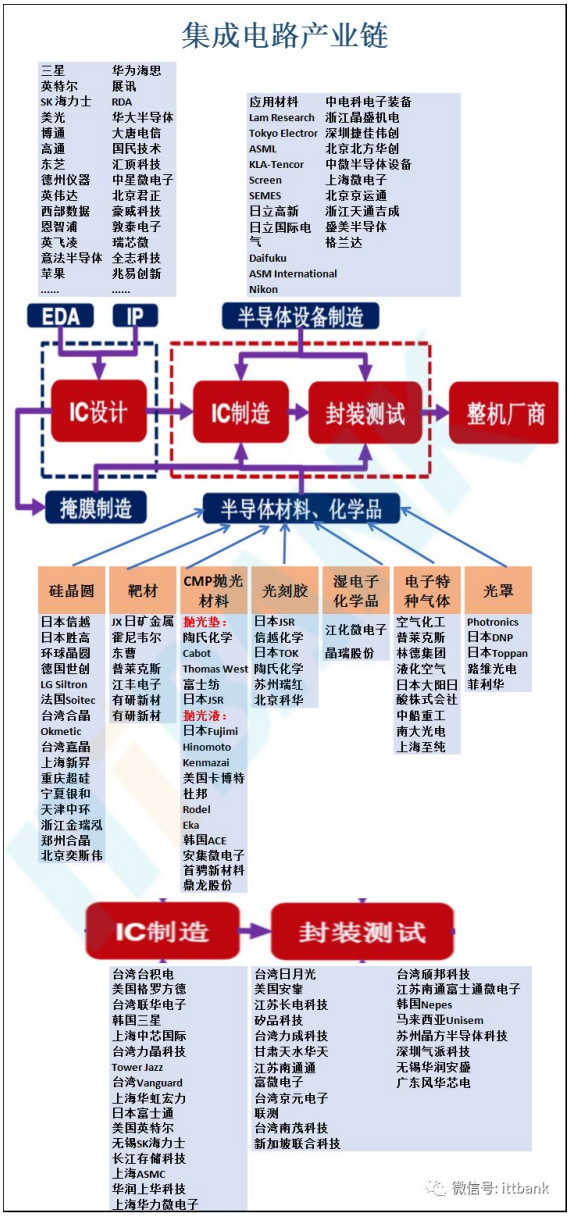

Figure A.1: IC 产业链

Bibliography