计算机是如何工作的

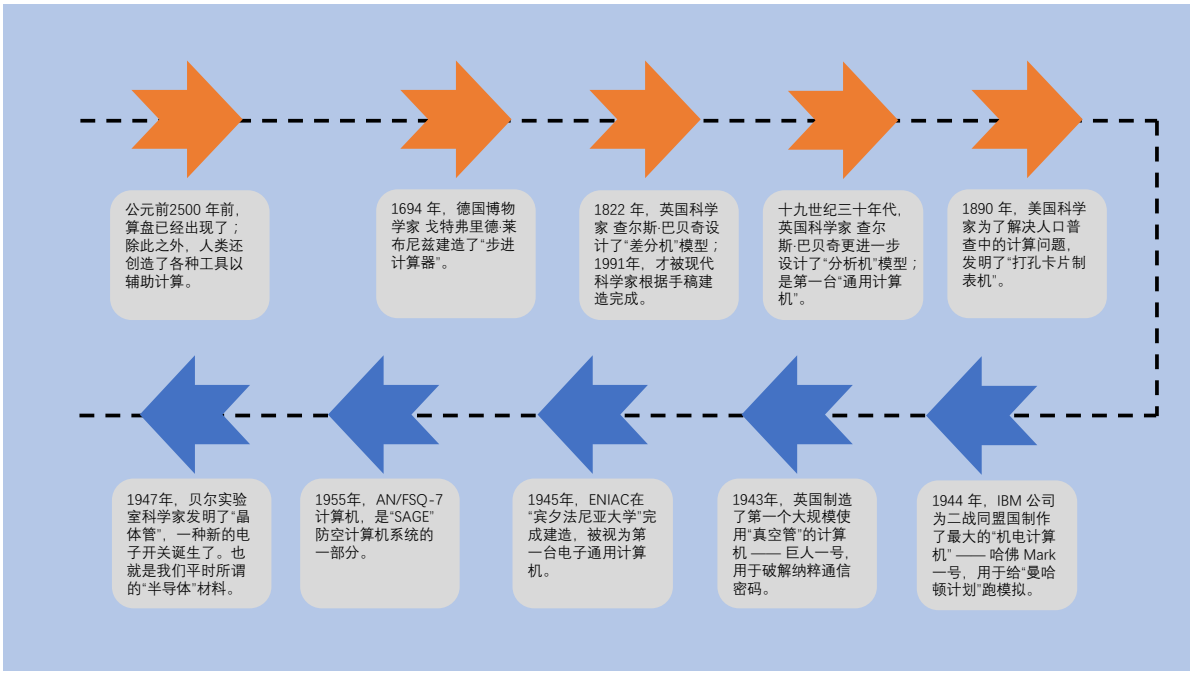
学习目标

这阶段的课程中，我们会从软件工程师的角度解释计算机是如何工作的，我们的主要目标既不是期待大家可以造出自己的计算机，也不是介绍如何编程，而是希望让大家了解计算机的核心工作机制后，打破计算机的神秘感，并且有利于理解我们平时编程时的一些行为、动作的历史渊源。

大家可以配合 Crash Course Computer Science 视频做更详尽的学习和了解。

计算机发展史

计算的需求在人类的历史中是广泛存在的，发展大体经历了从一般计算工具到机械计算机到目前的电子计算机的发展历程。

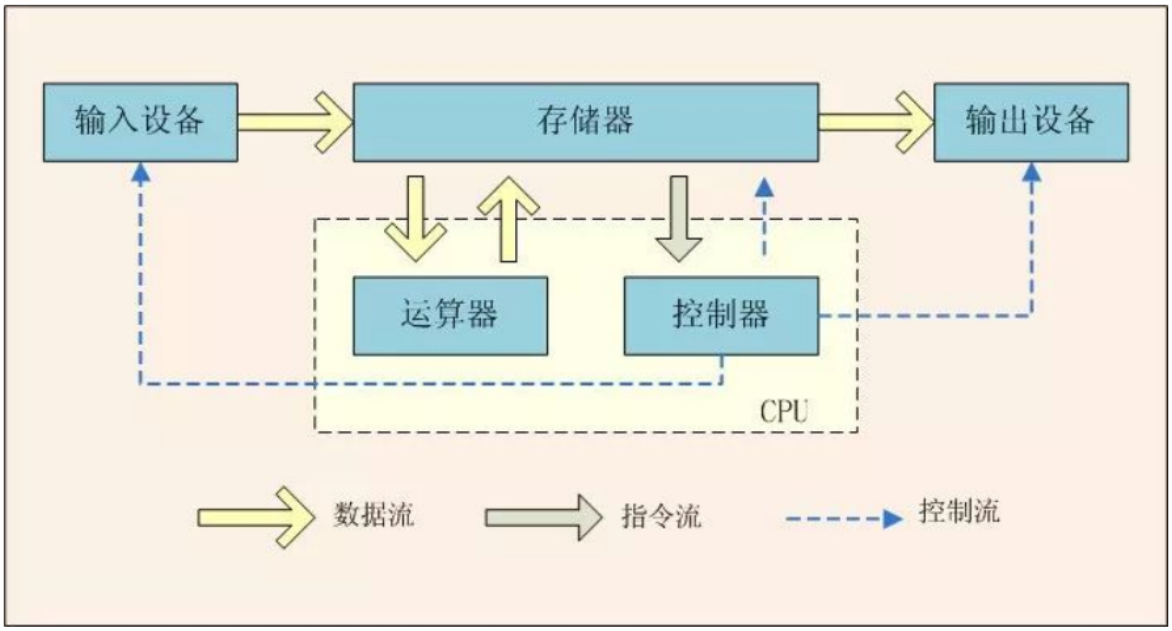


人类对计算的需求，驱动我们不断的发明、改善计算机。目前这个时代是“电子计算机”的时代，发展的潮流是：更快速、更稳定、更微型。计算机的以后将如何发展，期待大家的努力。

推荐书籍: 《计算机简史》

冯诺依曼体系（Von Neumann Architecture）

现代的计算机, 大多遵守 **冯诺依曼体系结构**



- CPU 中央处理器: 进行算术运算和逻辑判断.
- 存储器: 分为外存和内存, 用于存储数据(使用二进制方式存储)
- 输入设备: 用户给计算机发号施令的设备.
- 输出设备: 计算机个用户汇报结果的设备.

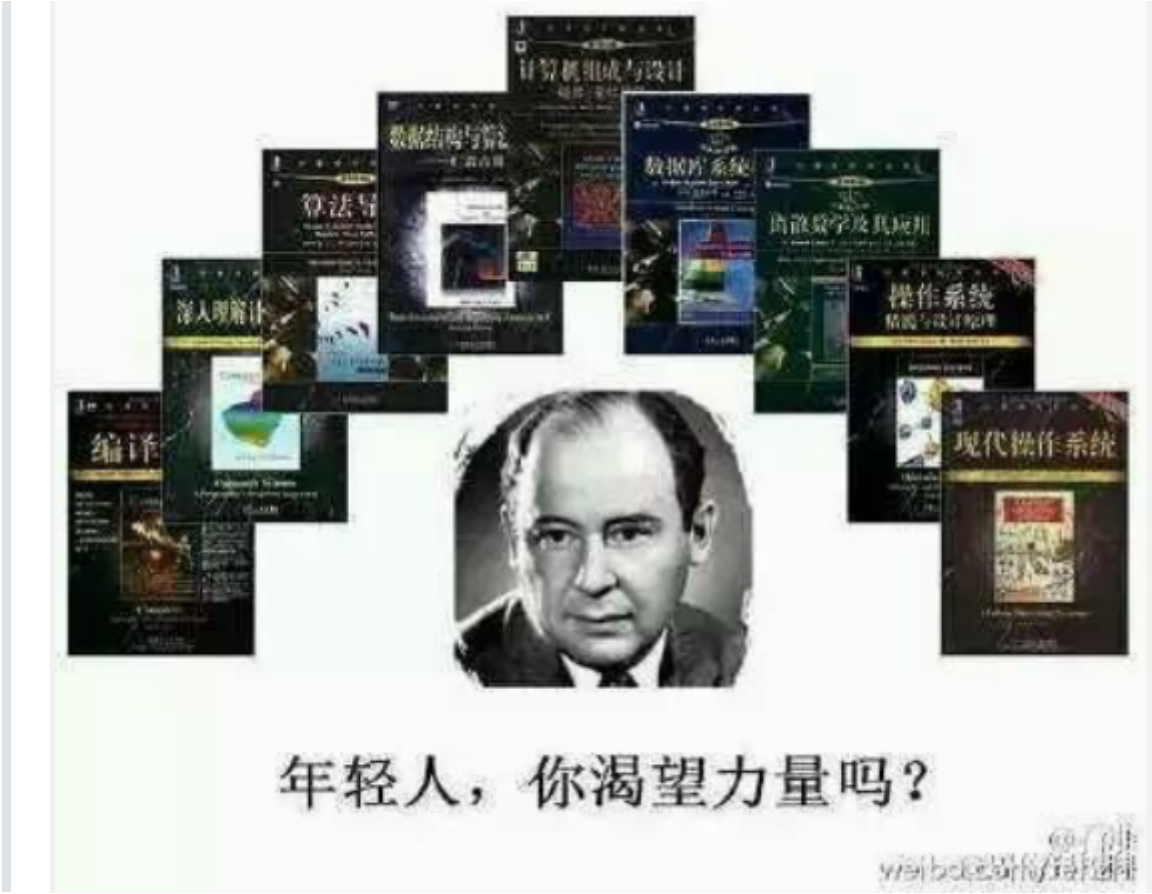
针对存储空间

硬盘 > 内存 >> CPU

针对数据访问速度

CPU >> 内存 > 硬盘

> **认识计算机的祖师爷 -- 冯诺依曼**
>
> 冯·诺依曼（John von Neumann，1903年12月28日-1957年2月8日），美籍匈牙利数学家、计算机科学家、物理学家，是20世纪最重要的数学家之一。冯·诺依曼是布达佩斯大学数学博士，在现代计算机、博弈论、核武器和生化武器等领域内的科学全才之一，被后人称为“现代计算机之父”、“博弈论之父”.

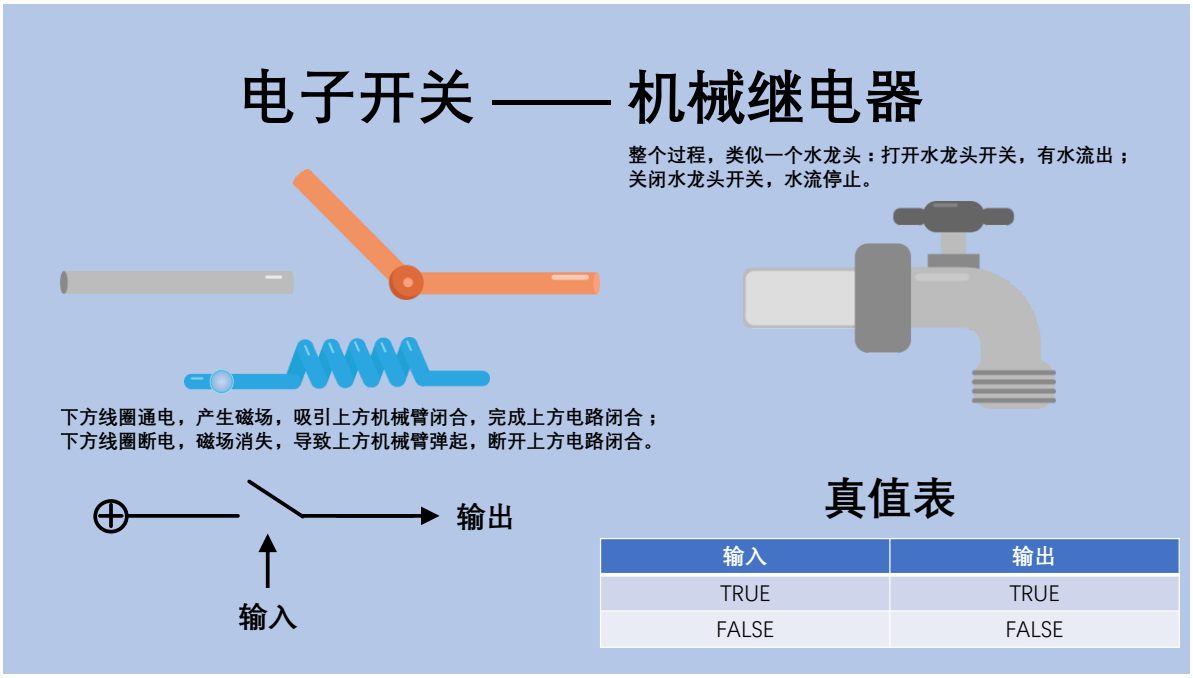


CPU 基本工作流程

接下来，我们用一个从无到有的过程，一步步搭建一个 CPU 出来，希望大家可以借助这个过程，理解 CPU、内存等计算机主要部件的工作原理。

逻辑门

电子开关 —— 机械继电器(Mechanical Relay)

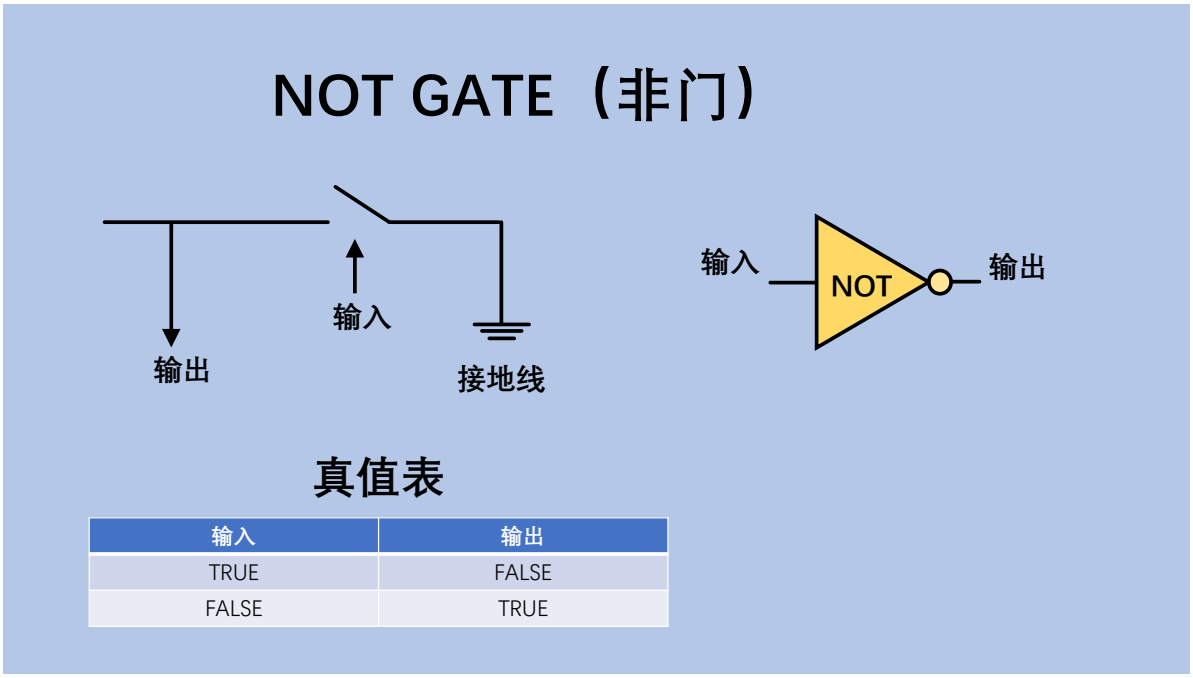


通过电子开关，我们可以实现 1 位(bit) 的看似无用的逻辑运算，但至少它工作起来了，不是么。怎么使用电子开关组合出真正有用的逻辑组件，我们接来下会做进一步的学习了解。

以后的真空管、晶体管的实质也是完成类似的工作，只是物理原理更加复杂，我们就不带着大家做深入解读了。

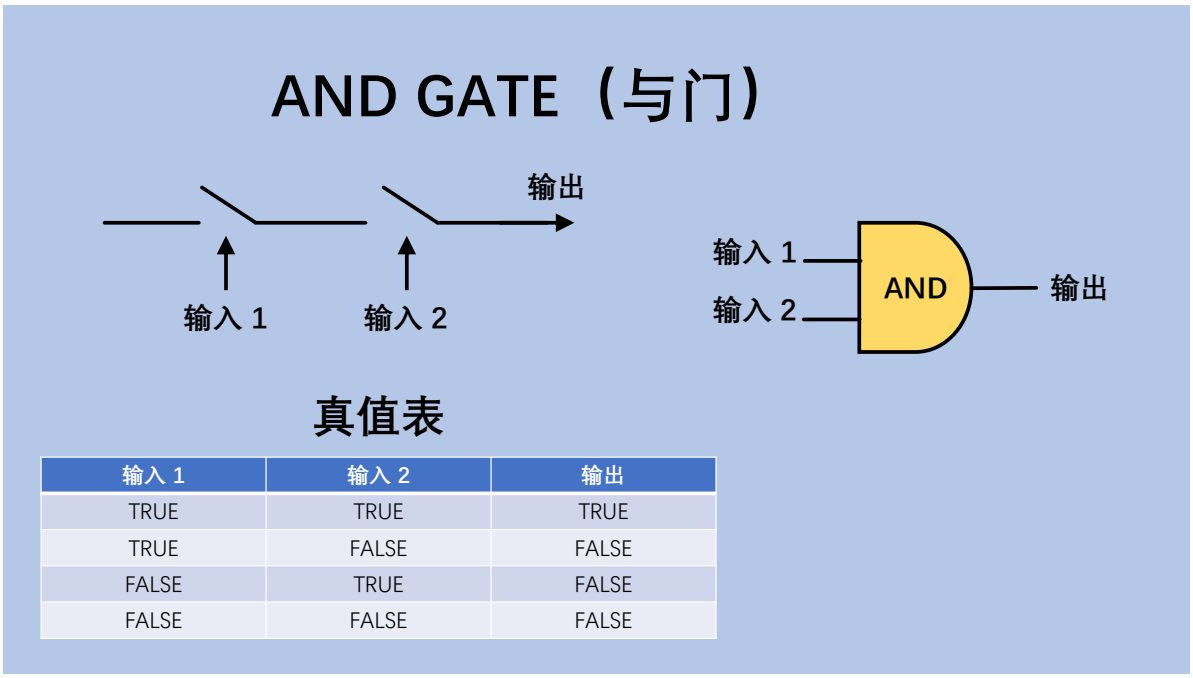
门电路(Gate Circuit)

接下来，我们学习如何使用电子开关构建一些有用的部件 —— 门电路。可以实现 1 位(bit) 的基本逻辑运算。



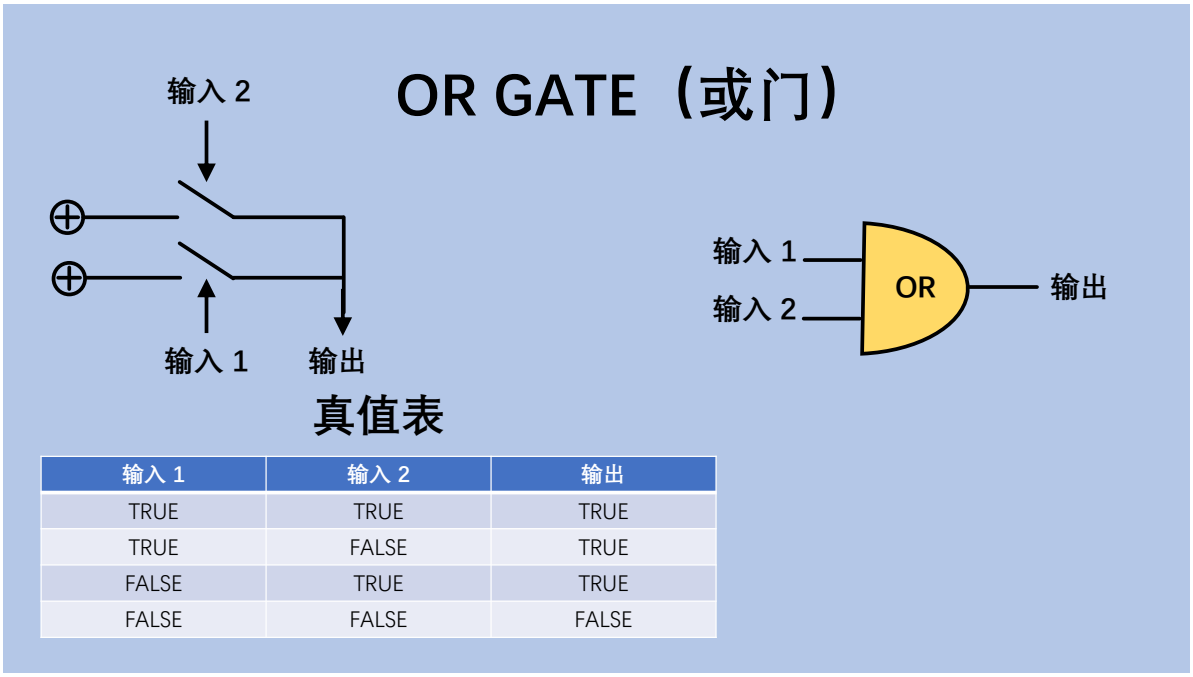
真值表

输入	输出
TRUE	FALSE
FALSE	TRUE



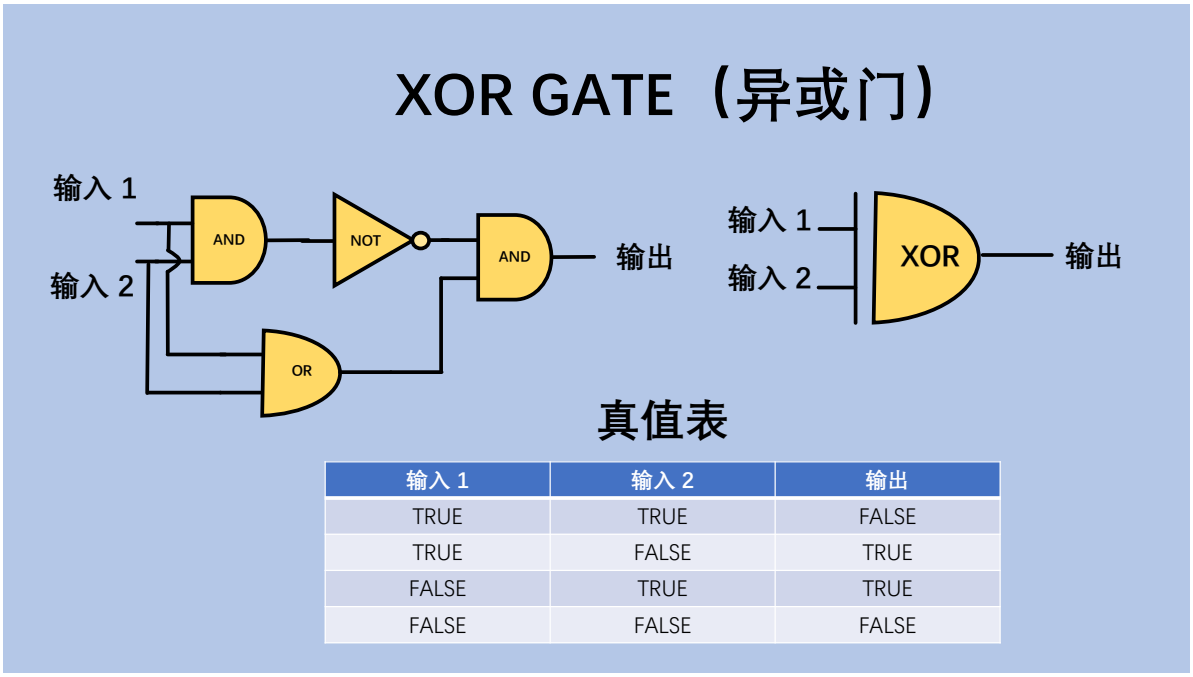
真值表

输入 1	输入 2	输出
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE



真值表

输入 1	输入 2	输出
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

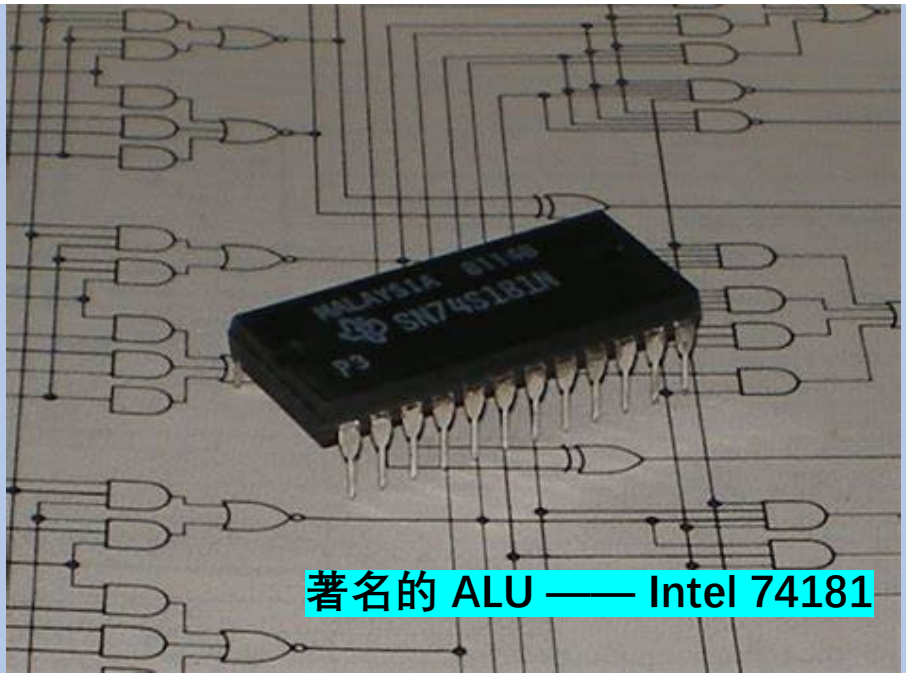


真值表

输入 1	输入 2	输出
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

算术逻辑单元 ALU（Arithmetic & Logic Unit）

ALU 是计算机中进行算数、逻辑运算的核心部件，是计算机的数学大脑。接下来，我们用上一节构建的逻辑门来完成自己的一个 ALU，去学习理解它的工作模式，以便作为我们进一步理解现代计算机工作原理的基石。



进制的理解

我们已经熟悉数字的各种表示了，让我们再简单回顾下进制。

进制的理解

十进制
(10 base notation / decimal notation)

10^2	10^1	10^0
100's	10's	1's
1	8	3

$$183 = 1 \times 10^2 + 8 \times 10^1 + 3 \times 10^0$$
$$= 1 \times 100 + 8 \times 10 + 3 \times 1$$

二进制
(2 base notation / binary notation)

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128's	64's	32's	16's	8's	4's	2's	1's
1	0	1	1	0	1	1	1

$$263 = 1 \times 2^7 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$
$$= 1 \times 128 + 1 \times 32 + 1 \times 16 + 1 \times 4 + 1 \times 2 + 1 \times 1$$

二进制相加

十进制相加

```
  1 1
  183
+  19
 ----
  202
```

二进制相加

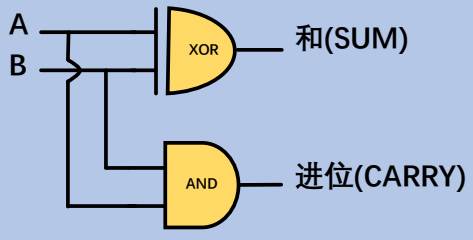
```
   1 1   1 1 1
  10110111
+ 00010011
 ---------
  11001010
```

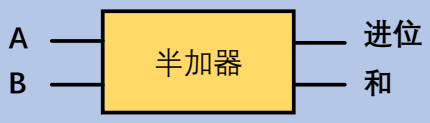
算术单元(Arithmetic Unit)

算数单元，负责计算机里的所有数字操作，比如四则运算，当然它能做的远远不止这些。接下来我们会带着大家实现一个 8 位（bits）的加法器（adder）来，以演示整个过程，其他的运算器就不再详细讲解了。

半加器(Half Adder)

进行两个 1 位(bit) 数的相加



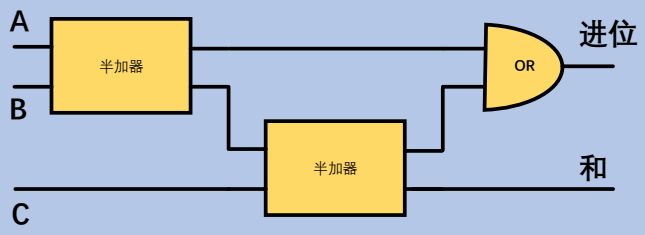


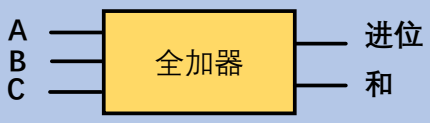
输入		输出	
A	B	进位	和
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

全加器(Full Adder)

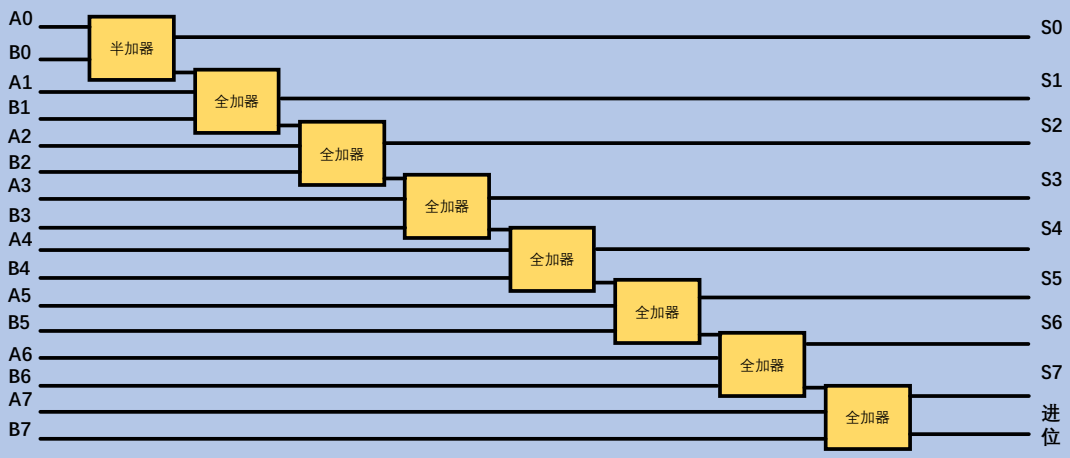
进行三个 1 位(bit) 数的相加

输入			输出	
A	B	C	进位	和
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
0	1	1	1	0
1	1	0	1	0
1	0	1	1	0
1	1	1	1	1





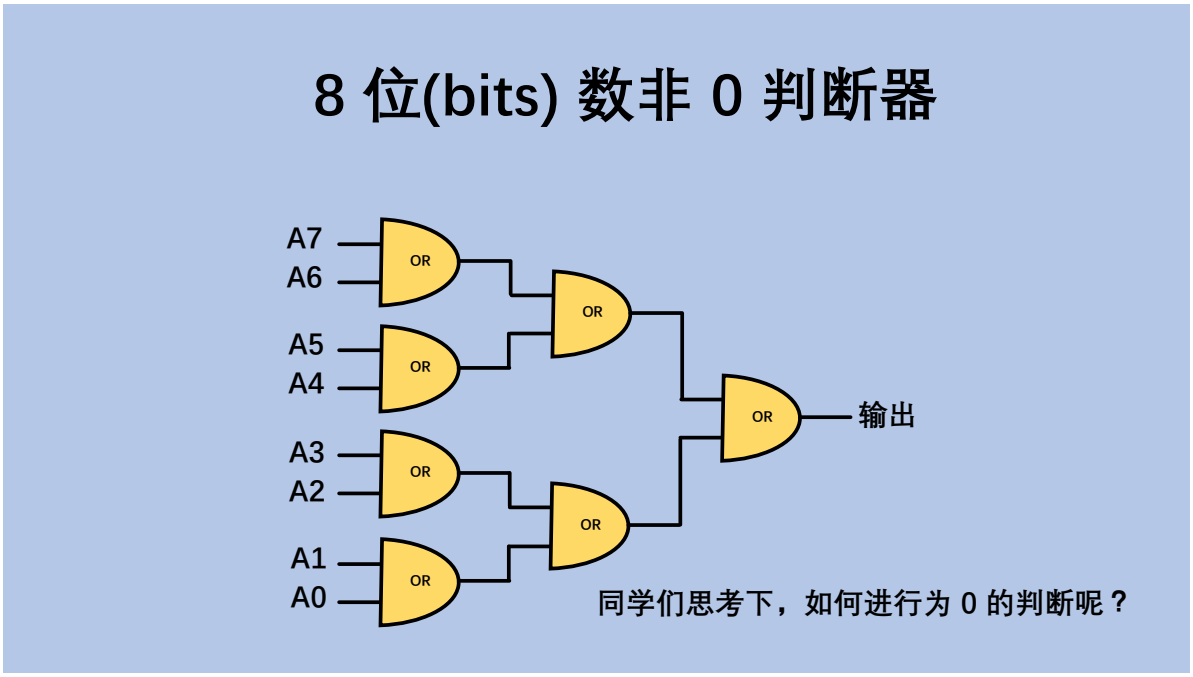
8 位(bits) 数加法器



至此，一个 8 位（bits) 加法器就被我们从无到有制作了出来。算术单元支持的操作当然远不止这些，通过继续组合逻辑门，算数单元可以做到加减乘除甚至更多的算术运算，但一个加法器作为演示已经足够了。实际上，乘法器和除法器的制作难度是要高于加、减法器的，有兴趣的同学可以尝试做更多的了解。

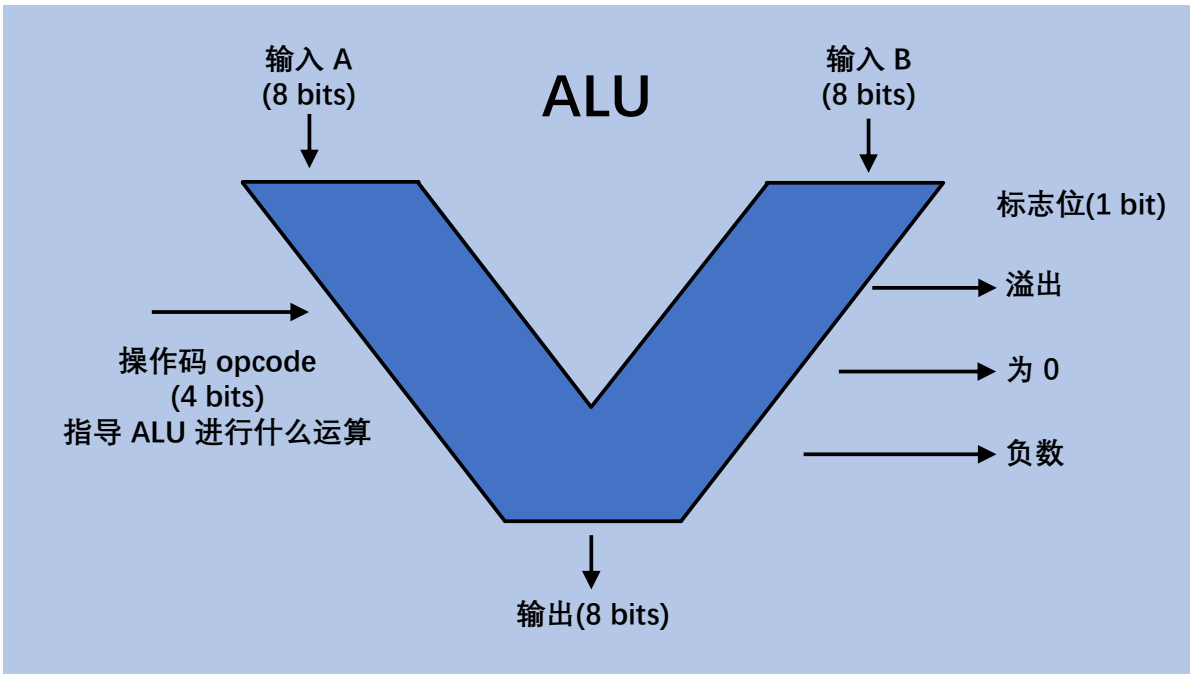
逻辑单元(Logic Unit)

逻辑单元主要用来进行逻辑操作，最基本的操作就是 与、或、非操作，但不只是一位(bit)数的比较。



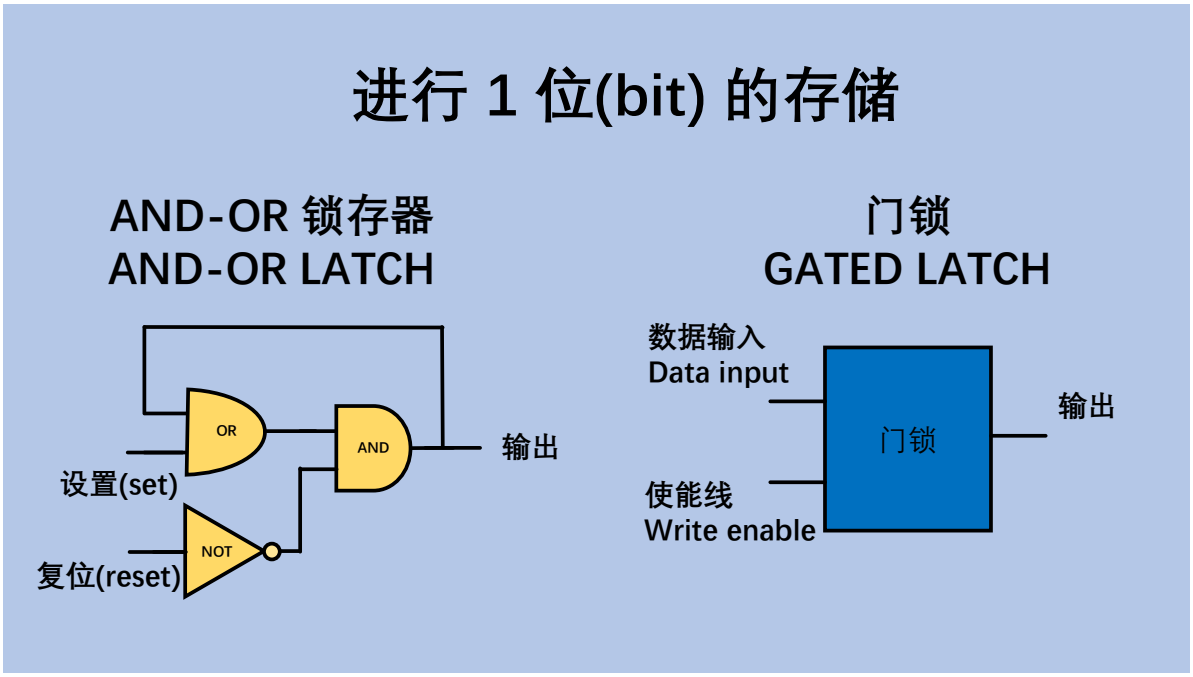
ALU 符号

经过我们的努力，通过基本的逻辑门电路，我们一步步地做出了一个 8 位(bits) ALU，甚至比 Intel 74181 还要强大，Intel 74181 只是一个 4 位(bits) ALU（😁）。当然现代的计算机中的 ALU 部件非常强大，复杂度远远超过了我们的想象，32 位 甚至 64 位基本已经普及全球了。但无论如何，再复杂的 ALU 也是芯片工程师像我们这样，一层又一层，一步又一步地将其抽象出来的。ALU 是第一次将人类历史上的数学和逻辑学学科有机地结合起来，可以视为人类智慧发展的现代巅峰。



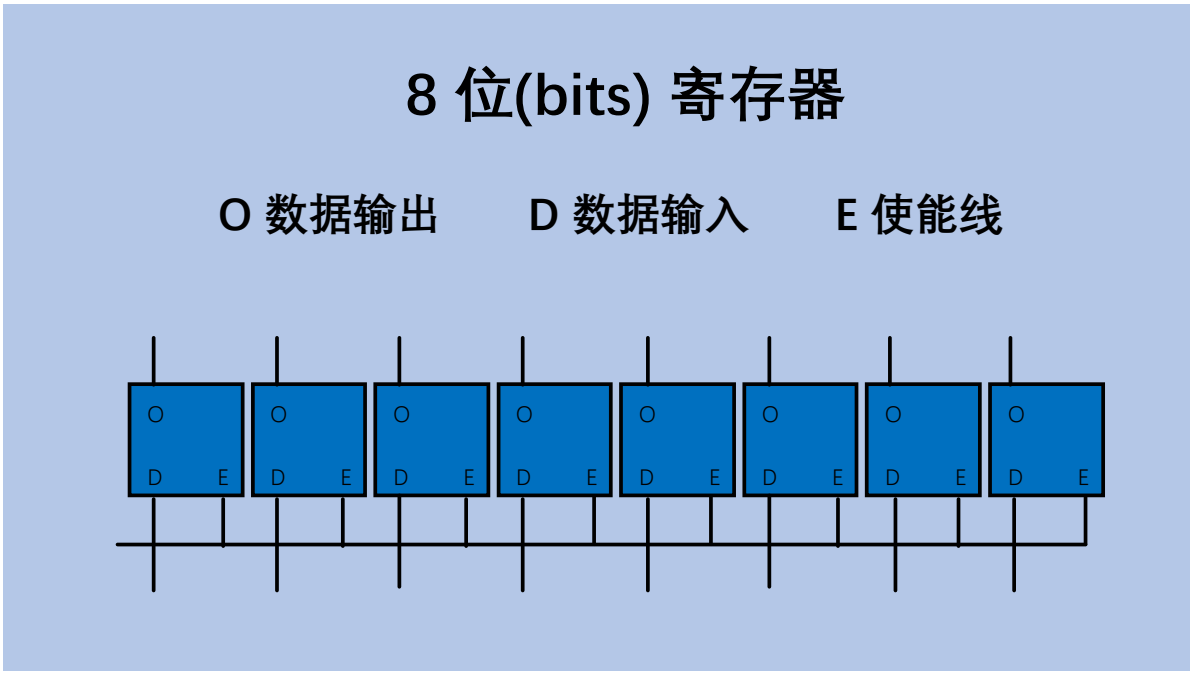
寄存器(Register) 和内存(RAM)

光有 ALU 还是远远不够的，我们无法为 ALU 提供存储的部件，所以接来下，我们利用门电路简单说明下存储的制作。注意，虽然图中没有明显的表示出来，但这些存储都是要求必须保持通电状态的，也就是这些存储都是易失的（volatile）。

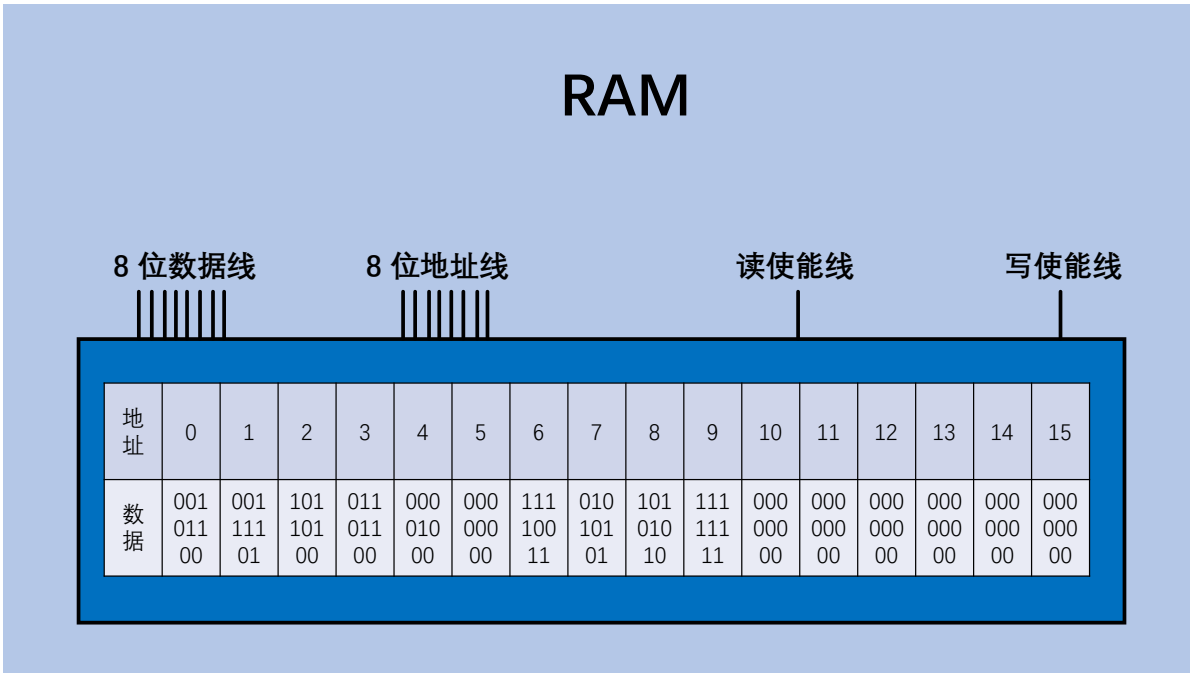


中间我们隐藏了一些实现细节，最后的效果就是：使能线置位时，输入为 1，保存 1；输入为 0，保存 0。使能线不置位时，则写入无效。

我们可以利用门锁，构建我们需要的寄存器和内存。



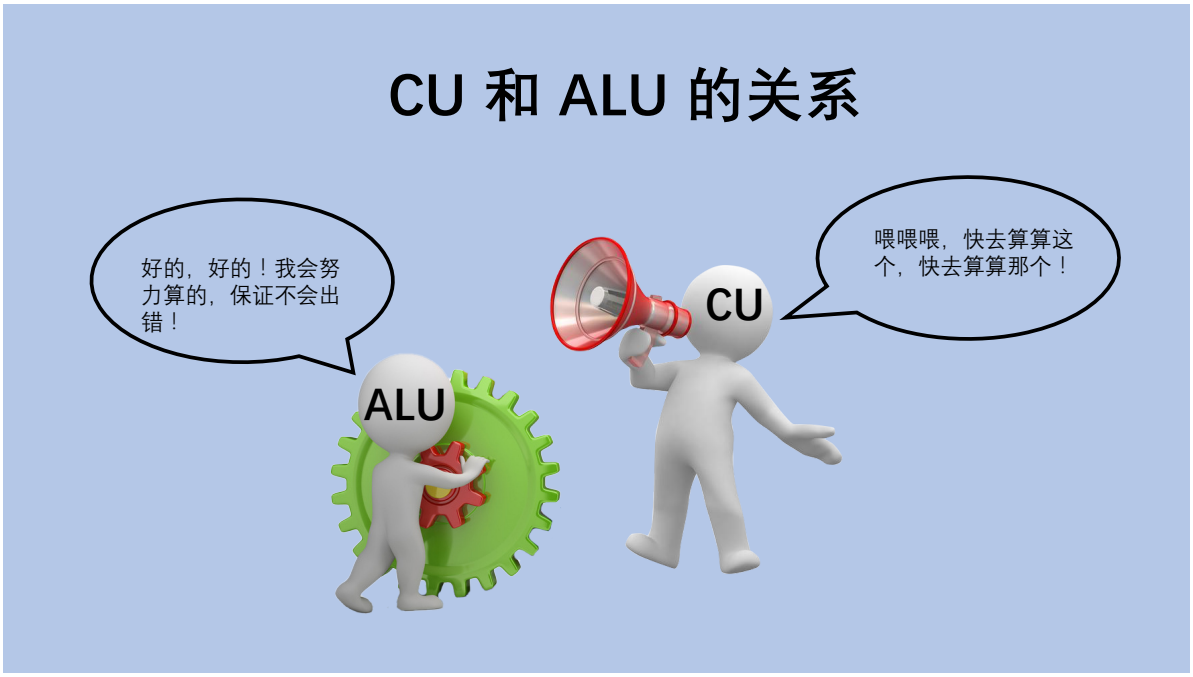
内存的构建要比这个复杂一点，但基本原理一致。如此构建的内存被称为 RAM(Random Access Memory)，可以支持 O(1) 时间复杂度访问任意位置的数据，这也就是我们数组下标访问操作是 O(1) 的硬件支持。



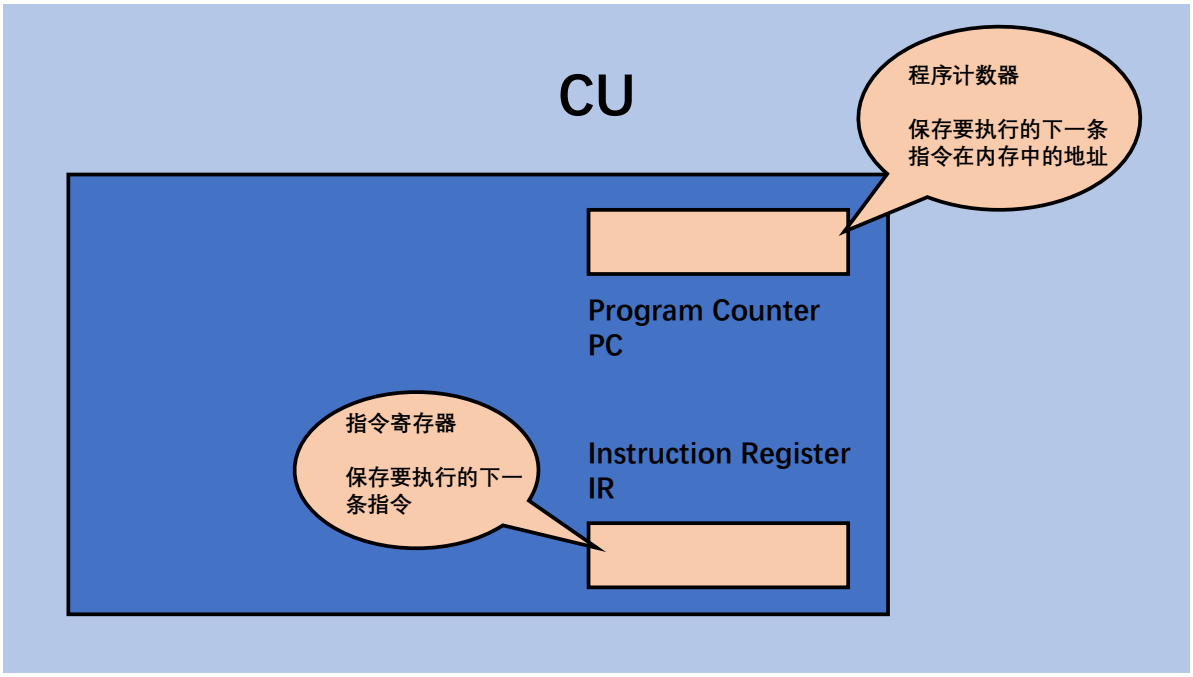
期间，为了我们学习的聚焦性，我们隐藏了大量的实现细节，对于这部分知识感兴趣的同学可以在课后做深入的学习。

控制单元 CU(Control Unit)

我们现在有 ALU、存储了，但这还是不足以让我们的计算机工作起来，我们需要有一个部件来指挥 ALU 进行何种的运算，而这个部件就是控制单元(CU)。



关于 CU 如何由门电路从无到有搭建，我们就进行抽象了，我们只需要理解 CU 可以驱动 ALU 进行具体的计算工作即可，至于 ALU 是如何驱动 ALU 进行工作的，我们在下一节进行详细的论述。



指令（Instruction）

首先，我们先介绍下我们需要到的指令(instruction)。

所谓指令，即指导 CPU 进行工作的命令，主要有操作码 + 被操作数组成。

其中操作码用来表示要做什么动作，被操作数是本条指令要操作的数据，可能是内存地址，也可能是寄存器编号等。

指令本身也是一个数字，用二进制形式保存在内存的某个区域中。

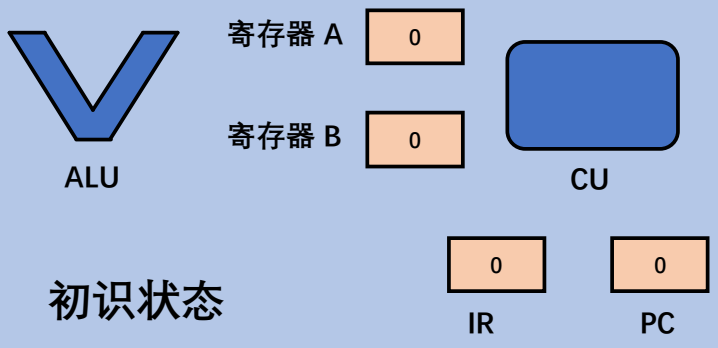
指令表(Instruction Table)

指令(instruction)	功能说明	4位 opcode	操作的地址或者寄存器
LOAD_A	从 RAM 的指定地址，将数据加载到 A 寄存器	0010	4 位 RAM 地址
LOAD_B	从 RAM 的指定地址，将数据加载到 B 寄存器	0001	4 位 RAM 地址
STORE_A	将数据从 A 寄存器写入 RAM 的指定地址	0100	4 位 RAM 地址
ADD	计算两个指定寄存器的数据的和，并将结果放入第二个寄存器	1000	2 位的寄存器 ID 2 位的寄存器 ID

CPU 的基本工作流程

接下来，我们演示指令运行的一个周期，希望同学们可以学习到其流程，并完成剩余指令的运行过程。

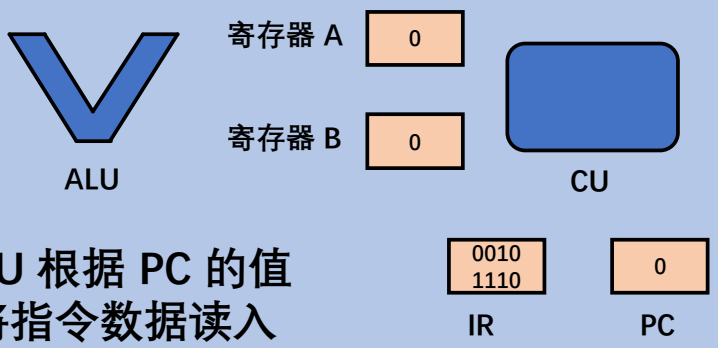
CU 和 ALU 的配合



初识状态

地址	数据
0	00101110
1	00011111
2	10000100
3	01001101
4	00000000
5	00000000
6	00000000
7	00000000
8	00000000
9	00000000
10	00000000
11	00000000
12	00000000
13	00000000
14	00000011
15	00001110

CU 和 ALU 的配合



CU 根据 PC 的值
将指令数据读入
IR 中

地址	数据
0	00101110
1	00011111
2	10000100
3	01001101
4	00000000
5	00000000
6	00000000
7	00000000
8	00000000
9	00000000
10	00000000
11	00000000
12	00000000
13	00000000
14	00000011
15	00001110

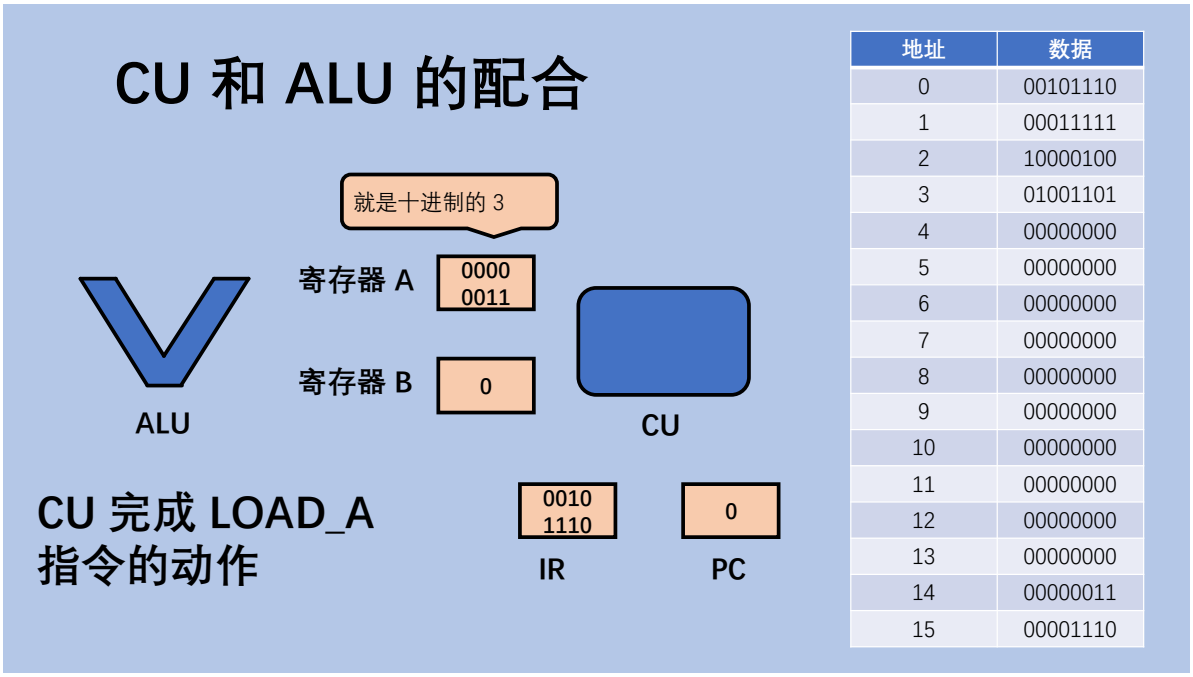
CU 和 ALU 的配合

指令(instruction)	功能说明	4位 opcode	操作的地址或者寄存器
LOAD_A	从 RAM 的指定地址，将数据加载到 A 寄存器	0010	4 位 RAM 地址
LOAD_B	从 RAM 的指定地址，将数据加载到 B 寄存器	0001	4 位 RAM 地址
STORE_A	将数		地址
ADD	计算数		器 ID 器 ID

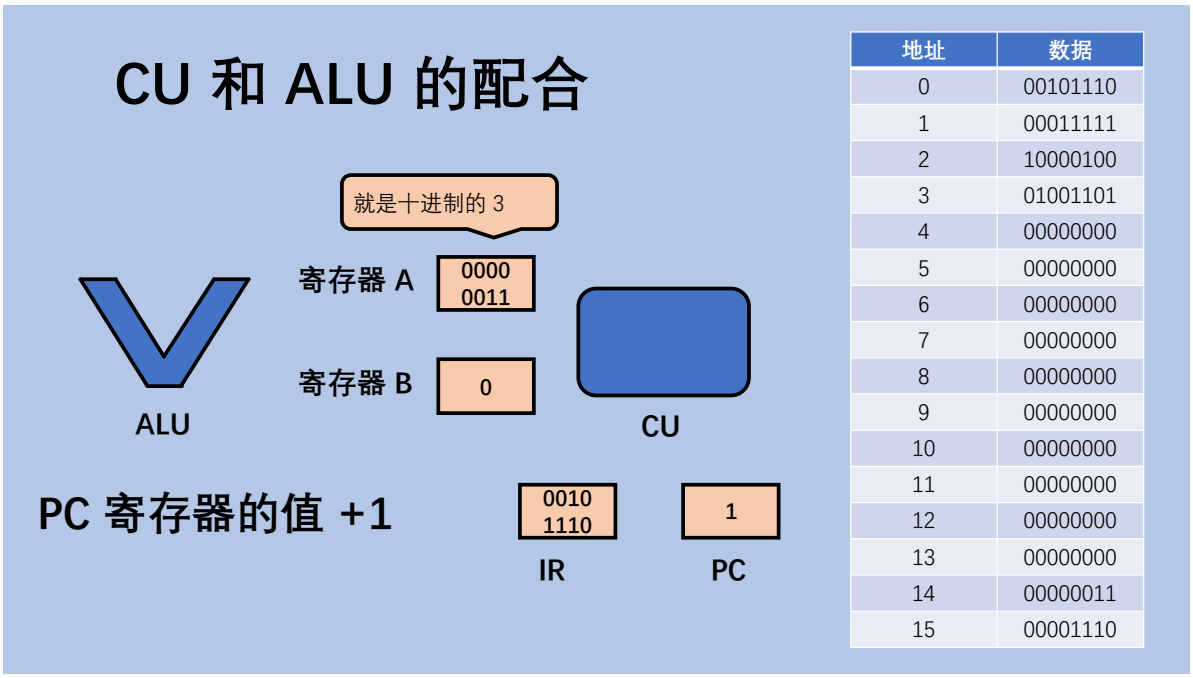
0010 查表可得是 LOAD_A 操作
1110 是十进制的 14
所以是要把 RAM 中地址 是 14 的数据读到寄存器 A 中

CU 分析 IR 中的
指令组成

地址	数据
0	00101110
1	00011111
2	10000100
3	01001101
4	00000000
5	00000000
6	00000000
7	00000000
8	00000000
9	00000000
10	00000000
11	00000000
12	00000000
13	00000000
14	00000011
15	00001110



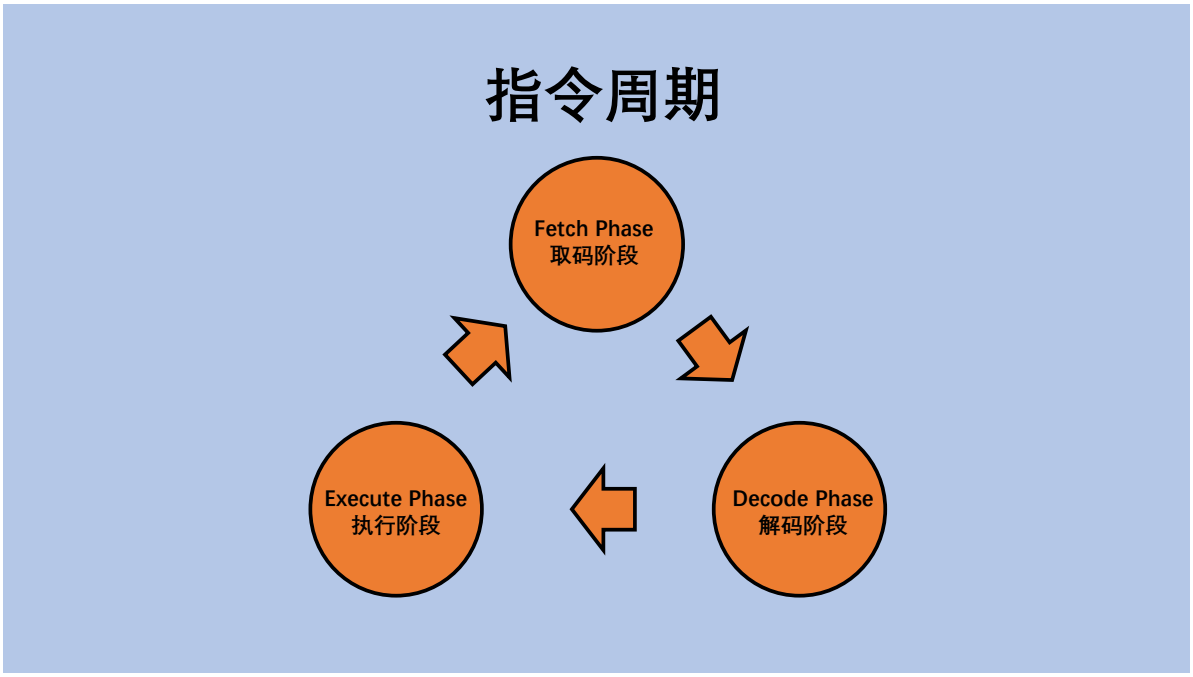
地址	数据
0	00101110
1	00011111
2	10000100
3	01001101
4	00000000
5	00000000
6	00000000
7	00000000
8	00000000
9	00000000
10	00000000
11	00000000
12	00000000
13	00000000
14	00000011
15	00001110



地址	数据
0	00101110
1	00011111
2	10000100
3	01001101
4	00000000
5	00000000
6	00000000
7	00000000
8	00000000
9	00000000
10	00000000
11	00000000
12	00000000
13	00000000
14	00000011
15	00001110

第一条指令的运行，其实没有用到我们之前制作的 ALU 部件，但这只是其中一些指令而已，大家尝试把剩余的 3 条指令自行运行一次，观察并理解这个过程。

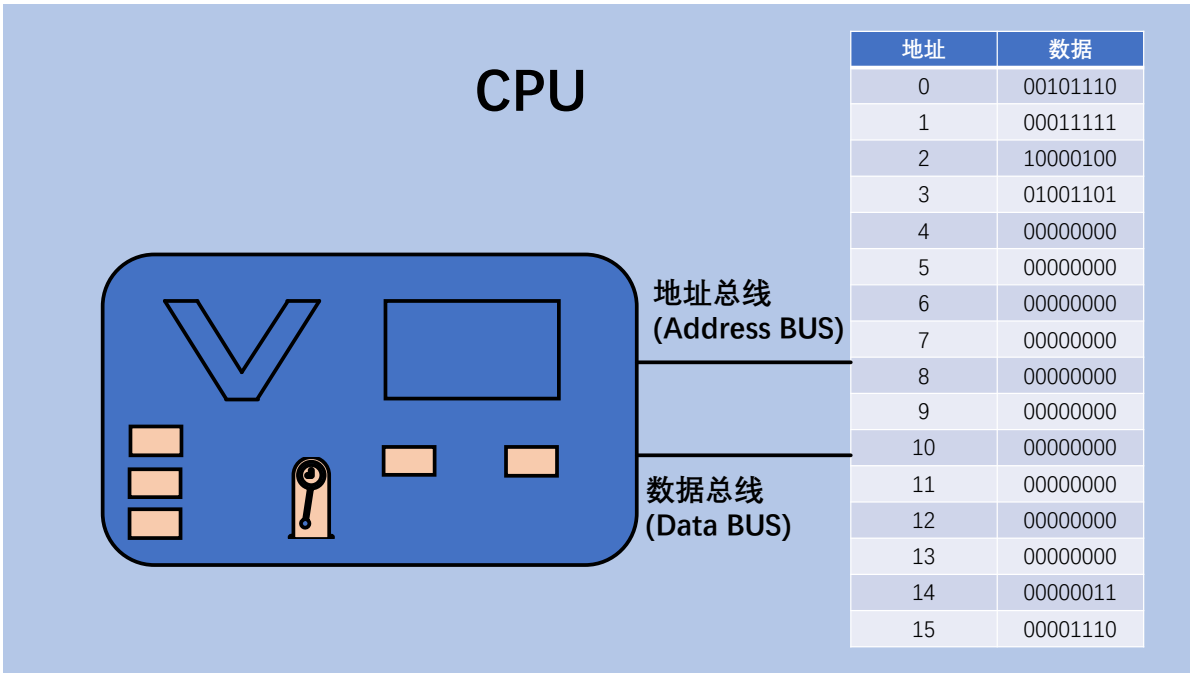
我们来总结下执行周期经过哪些阶段：



当然，电子计算机中的 CPU 可不像我们刚才那样，靠自己来驱动这个周期的运转，而是靠背后一个时钟来进行周期驱动的。

[知乎问题：时钟频率是什么概念]

最后，ALU + CU + 寄存器 + 时钟就组成了我们平时经常看到的一个词汇：中央处理器（Center Process Unit）简称 CPU。



地址	数据
0	00101110
1	00011111
2	10000100
3	01001101
4	00000000
5	00000000
6	00000000
7	00000000
8	00000000
9	00000000
10	00000000
11	00000000
12	00000000
13	00000000
14	00000011
15	00001110

小结

通过上述的章节，我们带领大家从基本的电子开关开始，一步步的搭建了一个CPU和内存出来，虽然中间还是对很多过程和细节做了隐藏和抽象，但主要流程已经体现了出来，希望这节学习完成之后，同学们不在对CPU充满了神秘感。

然后我们把这一节中一些要点给大家做一个文字总结：

1. CPU 中的 PC 寄存器，是决定 CPU 要执行哪条指令的关键；
2. 指令是由 动作 + 操作对象组成
3. CPU 眼中只有指令，没有其他的概念

编程语言（Program Language）

这一节，我们借助上一节制作的 CPU 和 内存，来尝试还原下我们已经熟悉的编程语言，例如 Java 是如何和 CPU 指令对应起来的。

程序（Program）

所谓程序，就是一组指令以及这组指令要处理的数据。狭义上来说，程序对我们来说，通常表现为一组文件。

程序 = 指令 + 指令要处理的数据。

早期编程

很久以前，那还是我用win98的时候有次我系统崩溃了，因为我是电脑白痴，我朋友给我介绍了一个高手来帮我修电脑。

他看了一下电脑，问我有没有98的盘，我说没有。

他想了一下，叫我把固定电话拿给他，我想修电脑要电话干什么，但人家是高手，我也不好说什么，就把电话拔下来给他了。

他把电话线空着的一头接在电脑的一个插孔内，然落后入了dos，然后就开始在电话上不停的按着键，他按键的速度异常快，但是只按0，1两个键，我搞不懂这有什么用，但也不敢问，看了半个多小时，他还是不停的按这两个键,两性，我徐徐的有些困，我问他这东西要搞多久，他说要几个小时，我给他倒了杯茶，就一个人去隔壁睡觉了。

醒来的时候，一看已经过了4个多小时，我起身到隔壁，看见他正在98里面调试，过了一会儿，他说，你试试，我坐上椅子用了一下，真的好了，我当时也不懂电脑，谢过人家就走了。
后来我慢慢对电脑有了了解，终于了解，原来当时那位高手是用机器语言编了一个98系统，我后来问我朋友那位高手的下落，我朋友说前几年去了美国之后，杳无音讯....

这是一个早先流传的趣味小故事，当然这件事不是真实的。但最早的电脑，要进行编程，是真的需要用0、1进行编程的（Σ(っ °Д °;)っ）

下面图给大家展示了 Altair 8800 计算机，是最早的一批微型电脑。用户需要控制开关，一个一个 bit 的将程序录入该电脑中。

Altair 8800，最早的微型电脑，
需要按照二进制进行编程

如果要求计算机的用户都必须使用二进制编程，那大家都要疯掉了，这可是一件门槛太高的事情了。所以编程语言应运而生了。

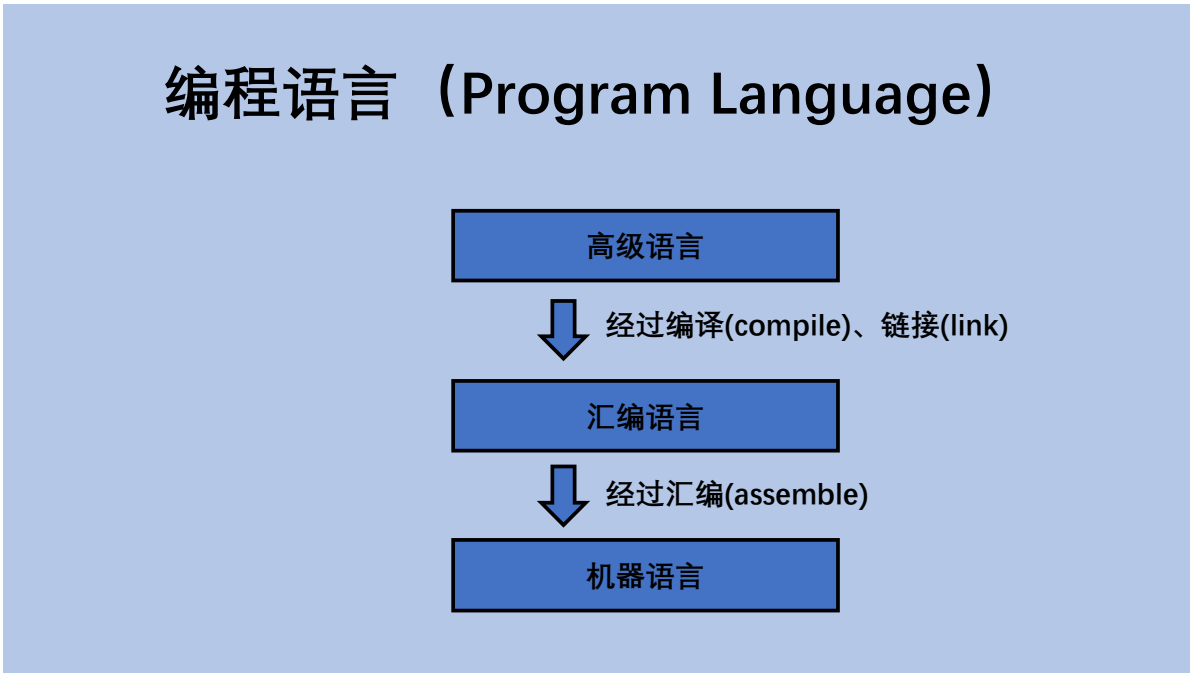
编程语言发展

为了提升编程效率，最早创造了汇编语言的概念。其实汇编语言和机器语言（也就是指令）直接是完全一一对应的，只是相对于 0、1 这些数字，发明了一些帮助人类记忆和理解的符号将其对应起来，也就是我们上面看到的类似 LOAD_A、LOAD_B 等。程序员完成编程之后，需要使用汇编器（assembler）将汇编语言翻译成机器语言。

虽然汇编降低了程序员的记忆成本，但要求程序还是必须掌握计算机硬件的所有知识，而且随着计算机厂商越来越多，一次编写的程序往往只适用于一类计算机。这个是远远不够的，所以更为高级的语言诞生了，高级语言屏蔽了硬件细节，让程序员可以站在更高的层面上思考自己的业务。这里以 C 语言为例，程序员完成程序的编写之后，需要使用编译器（compiler）和连接器（linker）将程序翻译成汇编语言，再借助汇编器变成最终的机器语言。

借助封装的思想，我们学习编程变得越来越容易。不过有利则有弊，高度的抽象，导致很多的程序员把计算机视为一个黑箱，完全无法理解自己的程序是如何工作起来的，希望我们大家不要做这种程序员。

我们使用的 Java 语言相对于 C 语言更高级一点，但基本抽象原理上没有太大的差异，我们暂时就不展开说明了。

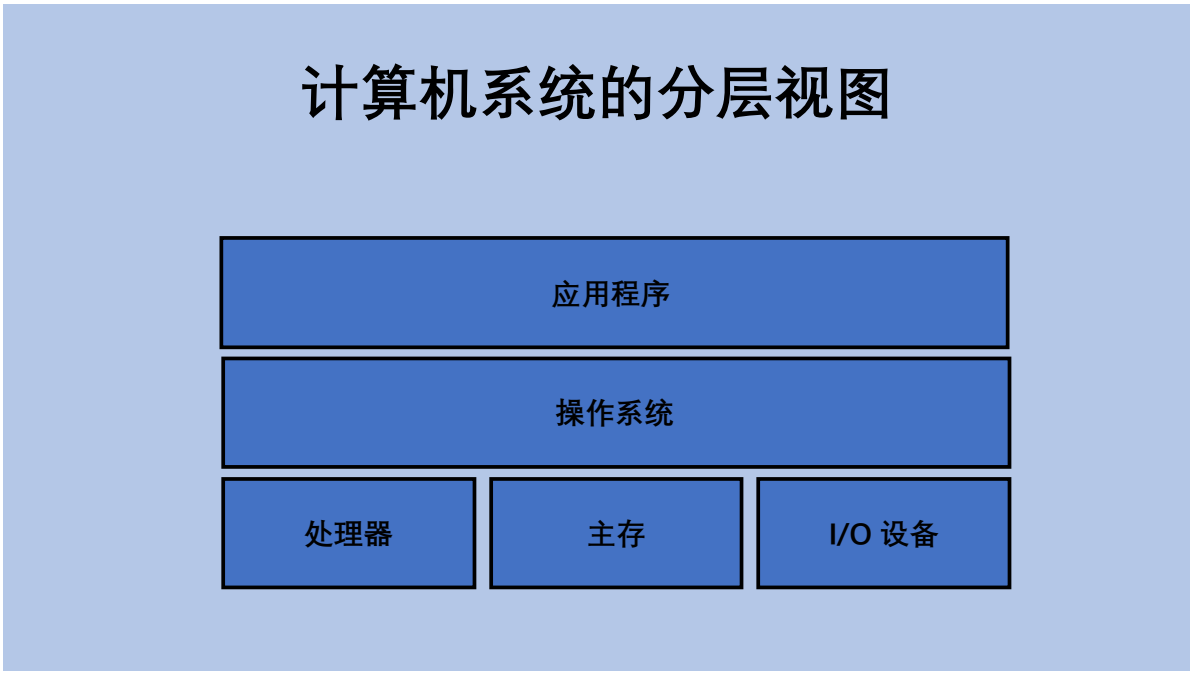


注意：高级语言的一条语句(Statement)往往对应很多条指令(Instruction)才能完成。

操作系统（Operating System）

操作系统是一组做计算机资源管理的软件的统称。目前常见的操作系统有：Windows系列、Unix系列、Linux系列、OSX系列、Android系列、iOS系列、鸿蒙等。

操作系统的定位



操作系统由两个基本功能：

1） 防止硬件被时空的应用程序滥用；

2） 向应用程序提供简单一致的机制来控制复杂而又通常大相径庭的低级硬件设备。

什么是进程/任务（Process/Task）

每个应用程序运行于现代操作系统之上时，操作系统会提供一种抽象，好像系统上只有这个程序在运行，所有的硬件资源都被这个程序在使用。这种假象是通过抽象了一个进程的概念来完成的，进程可以说是计算机科学中最重要和最成功的概念之一。

进程是操作系统对一个正在运行的程序的一种抽象，换言之，可以把进程看做程序的一次运行过程；

同时，在操作系统内部，进程又是操作系统进行资源分配的基本单位。

进程控制块抽象(PCB Process Control Block)

计算机内部要管理任何现实事物，都需要将其抽象成一组有关联的、互为一体的数据。在 Java 语言中，我们可以通过类/对象来描述这一特征。

```
// 以下代码是 Java 代码的伪码形式，重在说明，无法直接运行
class PCB {
    // 进程的唯一标识 —— pid;
    // 进程关联的程序信息，例如哪个程序，加载到内存中的区域等
    // 分配给该资源使用的各个资源
    // 进度调度信息（留待下面讲解）
}
```

这样，每一个 PCB 对象，就代表着一个实实在在运行着的程序，也就是进程。

操作系统再通过这种数据结构，例如线性表、搜索树等将 PCB 对象组织起来，方便管理时进行增删查改的操作。

CPU 分配 —— 进程调度（Process Scheduling）

为了便于讨论和理解，我们大部分的场景下假设是单CPU单核的计算机。

操作系统对CPU资源的分配，采用的是时间模式 —— 不同的进程在不同的时间段去使用 CPU 资源。

内存分配 —— 内存管理（Memory Manage）

操作系统对内存资源的分配，采用的是空间模式 —— 不同进程使用内存中的不同区域，互相之间不会干扰。

进程间通信(Inter Process Communication)

如上所述，进程是操作系统进行资源分配的最小单位，这意味着各个进程互相之间是无法感受到对方存在的，这就是操作系统抽象出进程这一概念的初衷，这样便带来了进程之间互相具备**"隔离性（Isolation）"**。

但现代的应用，要完成一个复杂的业务需求，往往无法通过一个进程独立完成，总是需要进程和进程进行配合地达到应用的目的，如此，进程之间就需要有进行**"信息交换"**的需求。进程间通信的需求就应运而生。

目前，主流操作系统提供的进程通信机制有如下：

1. 管道
2. 共享内存
3. 文件
4. 网络
5. 信号量
6. 信号

其中，网络是一种相对特殊的 IPC 机制，它除了支持同主机两个进程间通信，还支持同一网络内部非同一主机上的进程间进行通信。