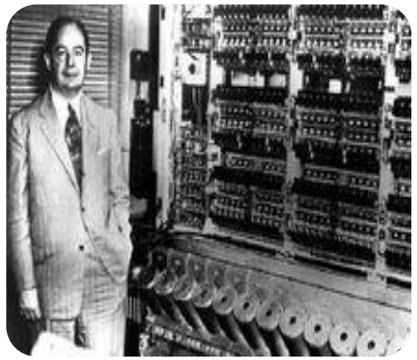
冯·诺依曼结构计算机工作原理及层次结构分析

1 冯·诺依曼结构



最开始的程序并未存储到计算机，而是一种如何布线（插线与开关实现硬件模块组合的改变）的操作手册。

ENIAC和EDVAC的建造者均为宾夕法尼亚大学的电气工程师约翰·莫奇利和普雷斯波·艾克特。1944年8月，EDVAC的建造计划就被提出；在ENIAC充分运行之前，其设计工作就已经开始。和ENIAC一样，EDVAC也是为美国陆军阿伯丁试验场的弹道研究实验室研制。冯·诺伊曼以技术顾问形式加入，总结和详细说明了EDVAC的逻辑设计，1945年6月发表了一份长达101页的报告，这就是著名的“关于EDVAC的报告草案”（First Draft of a Report on the EDVAC），报告提出的体系结构一直延续至今，即**冯·诺伊曼结构**。

2 冯·诺依曼计算机的工作原理

存储程序：将程序存放在计算机的存储器中；

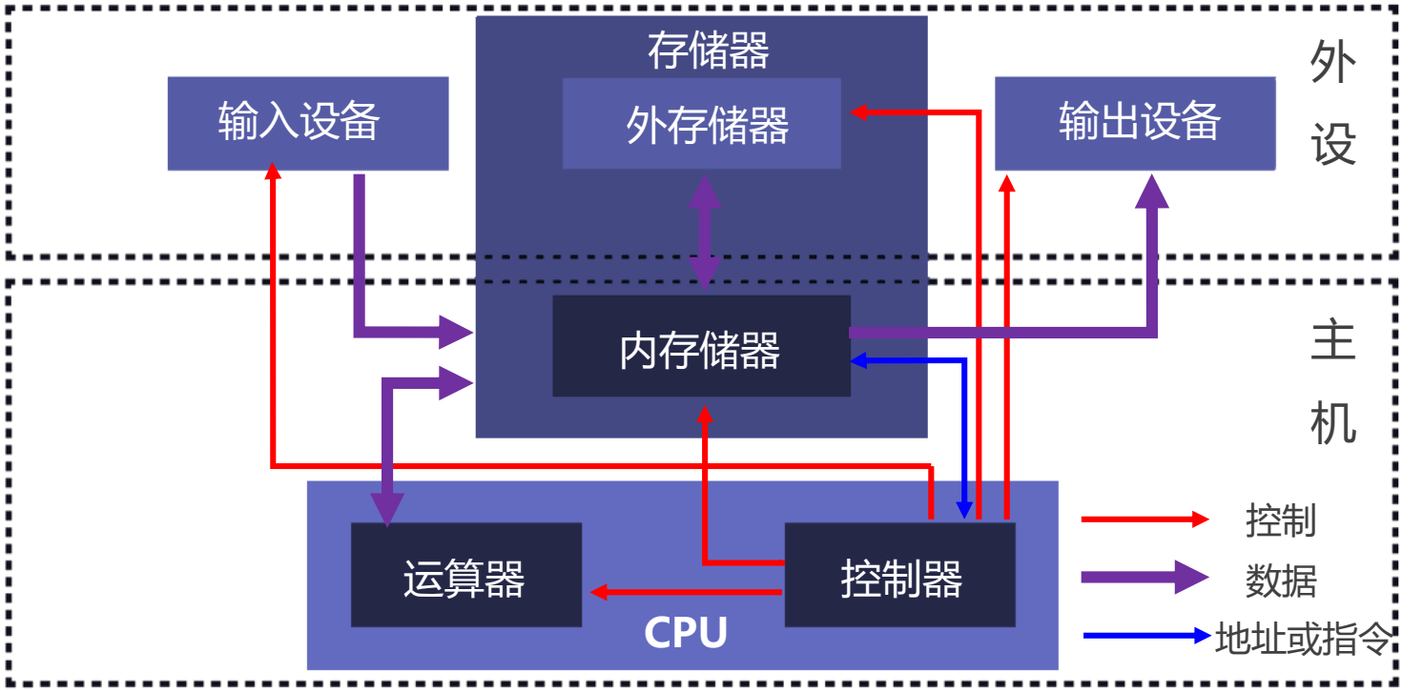
（存储系统构建与快速访问）

程序控制： 按指令地址访问存储器并取出指令，经译码依次产生指令执行所需的控制信号，实现对计算的控制，完成指令的功能。

（指令系统、控制器设计等）

3 冯·诺依曼计算机的组成（硬件+ 软件）

3.1 硬件系统（总体图）

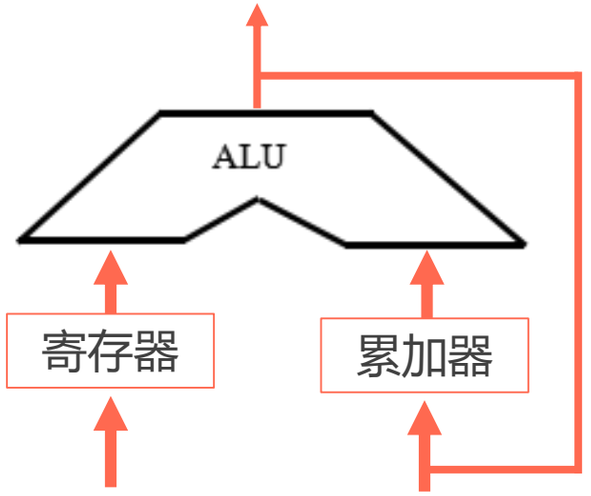


主机：CPU（运算器 + 控制器）、内存

外设：输入设备、输出设备、外存储器

总线：地址线、数据线、控制线

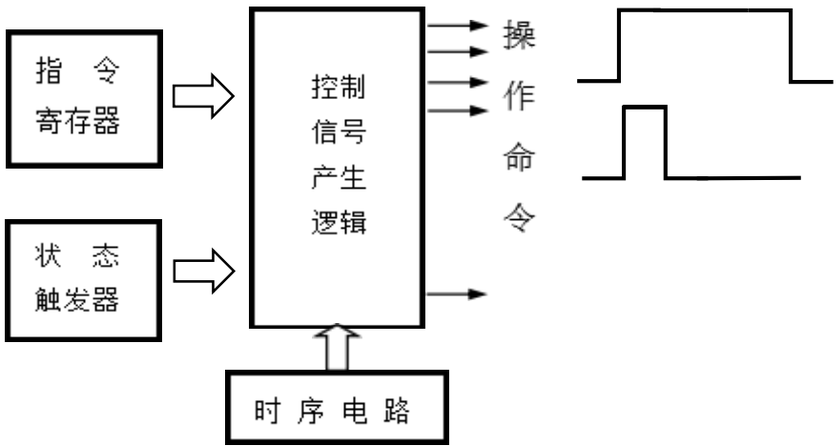
3.2 硬件系统 - **运算器**



算术运算	逻辑运算	基本结构
加、减、乘、除法等	与、或、非、移位等	ALU (**A**rithmetic **L**ogical **U**nit) 、寄存器、连接通路

◆ **注重功能与结构的关系？** -- 指令、数据类型、性能要求等等

3.3 硬件系统 - **控制器**



基本功能

产生指令执行过程所需要的所有控制信号，控制相关功能部件执行相应操作；

控制信号的形式

电平信号、
脉冲信号；

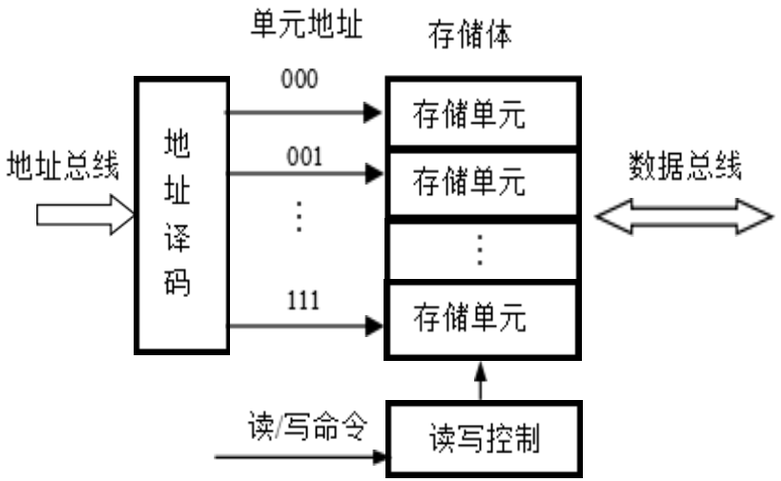
产生控制信号的依据

指令、状态、
时序；

控制信号的产生方式

微程序、
硬布线。

3.4 硬件系统 - **存储器**



功能

存储**原程序**、
原数据、
运算中间结果；

工作模式

读/写；

工作原理

按地址访问，
读/写数据。

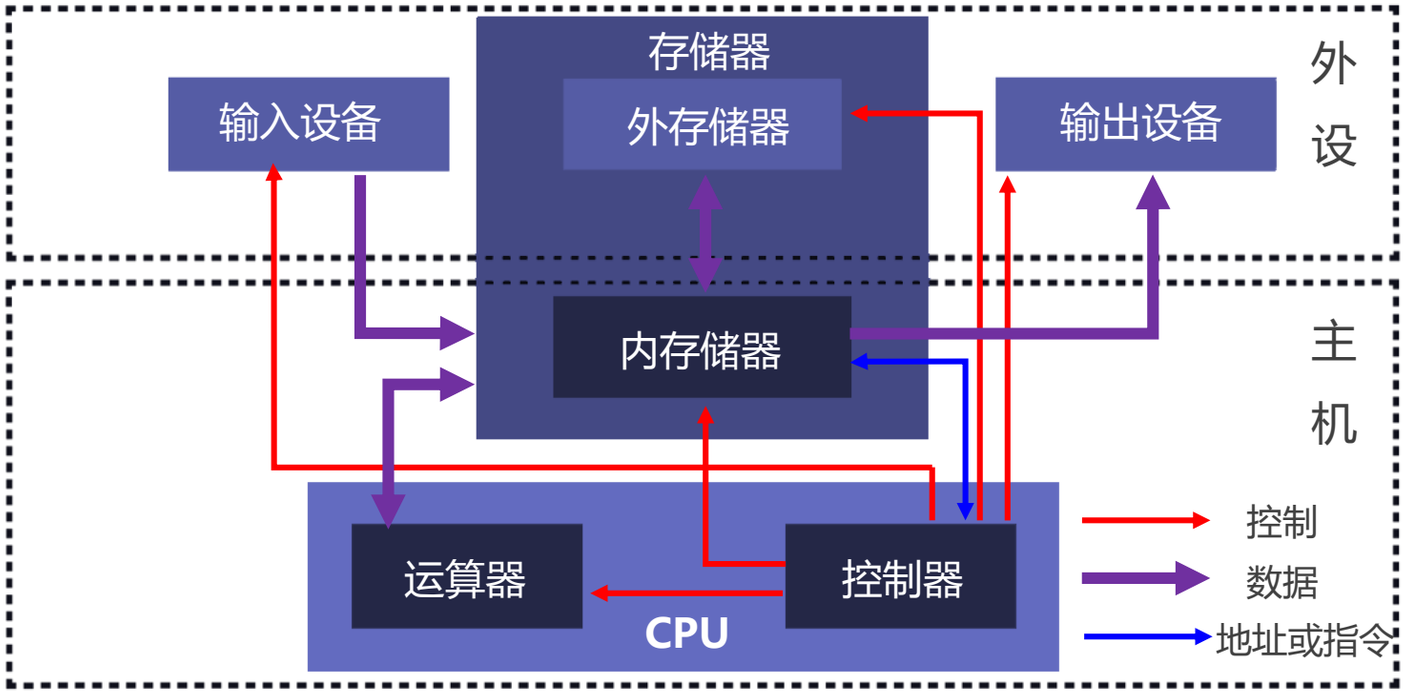
容量 → 地址线数量

1K → 10

1M → 20

1G → 30

3.5 硬件系统 -输入/输出设备



输入设备	向计算机输入数据(键盘、鼠标、网卡、扫描仪等)
输出设备	输出处理结果(显示器、声卡、网卡、打印机等)

3.6 软件系统

对软件的理解

- 可运行的思想和内容的数字化

  思想：算法、规律、方法---程序表达

  内容：图形、图像、数据、声音、文字等被处理的对象

- 软件的表现形式: 程序和数据（以二进制表示的信息）
- 软件的核心: 算法

3.7 软件系统分类

代码	计算机软件类别	说明	代码	计算机软件类别
10000	系统软件		60000	应用软件
11000	操作系统	包括实时、分时、颁布式、智能等操作系统	61000	科学和工程计算软件
			61500	文字处理软件
12000	系统实用程序		62000	数据处理软件
13000	系统扩充程序	包括操作系统的扩充、汉化	62500	图形软件
14000	网络系统软件		63000	图象处理软件
19900	其它系统软件		64000	应用数据库软件
30000	支持软件		65000	事务管理软件
31000	软件开发工具		65500	辅助类软件
32000	软件评测工具		66000	控制类软件
33000	界面工具		66500	智能软件
34000	转换工具		67000	仿真软件
35000	软件管理工具		67500	网络应用软件
36000	语言处理程序		68000	安全与保密软件
37000	数据库管理系统		68500	社会公益服务软件
38000	网络支持软件		69000	游戏软件
39900	其它支持软件		69900	其它应用软件

- 系统软件

  如操作系统、网络系统和编译系统

- 支持软件

  开发工具、界面工具等

- 应用软件

  字处理软件、游戏软件等

3.8 硬件与软件系统间的关系

相互依存

硬件是软件运行的基础，软件的正常运行是硬件发挥作用的重要途径。计算机系统必须要配备完善的软件系统才能正常工作，且应充分发挥其硬件的功能；

逻辑等效性

某些功能既可由硬件实现，也可由软件来实现；

协同发展

软件随硬件技术的迅速发展而发展，而软件的不断发展与完善又促进硬件的更新，两者密切地交织发展，缺一不可 。

4 计算机的层次结构

应用程序

高级语言 (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler(翻译)

汇编语言 (e.g., MIPS)

```
lw    $t0, 0($2)
lw    $t1, 4($2)
sw    $t1, 0($2)
sw    $t0, 4($2)
```

Assembler(翻译)

操作系统

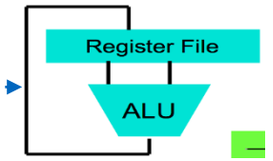
指令集架构层 (e.g., MIPS)

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

(解释)

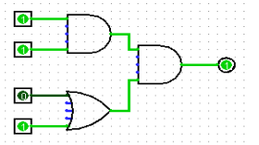
微代码层

```
RWrite = 1, Branch =Jump =0
RegDst=0
ALUOP= 00110,Jump=0
……
```



硬件逻辑层

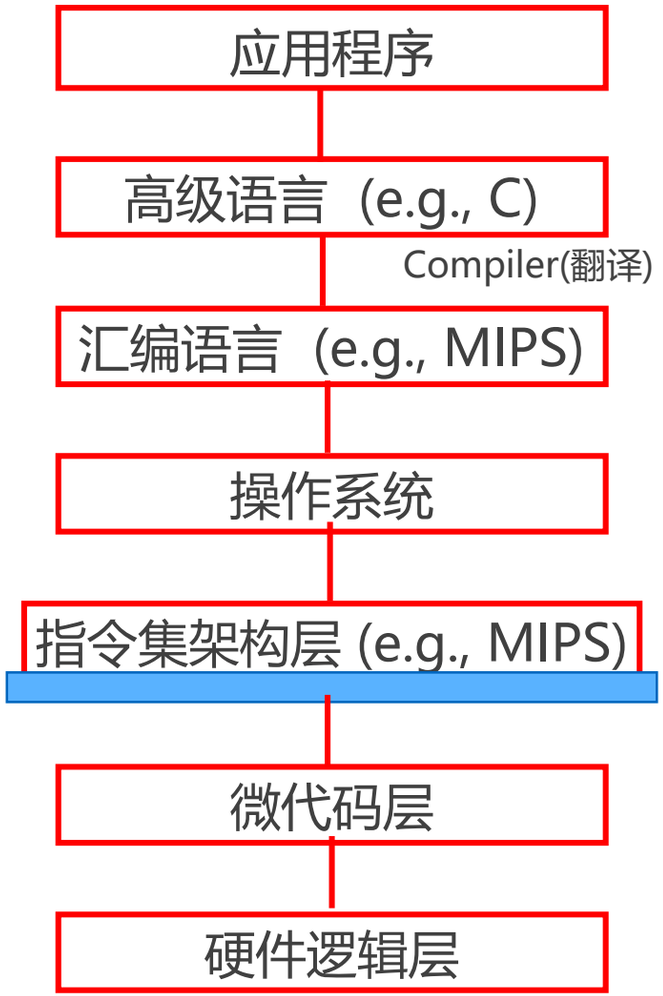
使用布尔代数和真值表



- 不同用户处在不同层次
- 不同层次具有不同属性
- 不同层次使用不同工具
- 不同层次代码效率不同

◆4.1 透明性概念

计算机的层次结构



- 本来存在的事物或属性，从某个角度去看，却好像不存在；
- 如硬件的特性对C语言程序设计者而言就具有透明性。

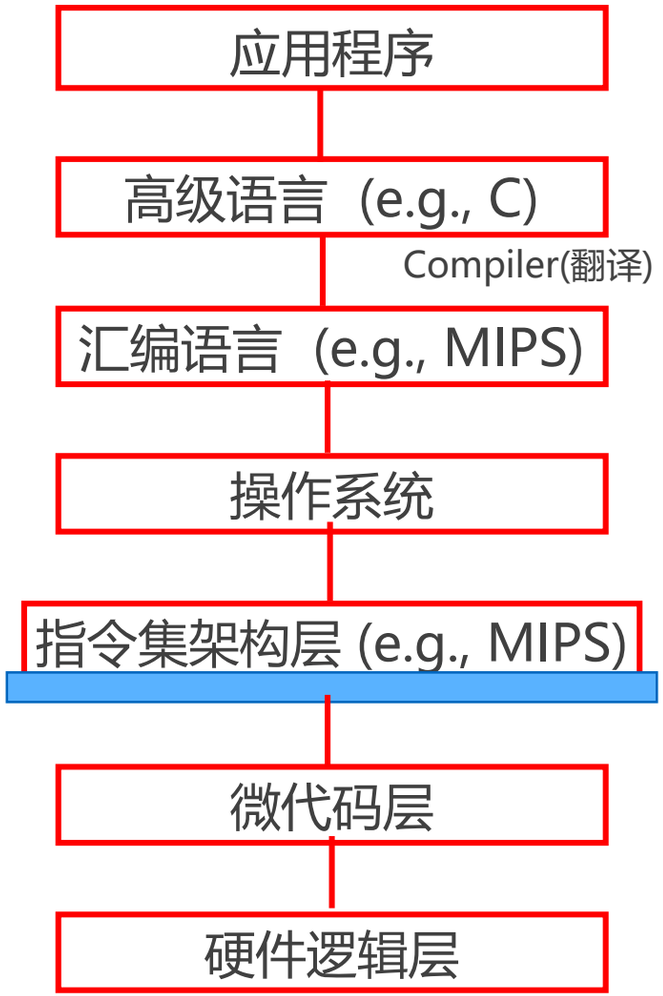
◆4.2 系统观

计算机的层次结构

- 当硬件结构发生变化时要想到可能对软件产生的影响；
- 不同类型的软件对硬件有不同的要求；
- 编程的CPU硬件相关性，编程应查阅对应CPU的编程手册。

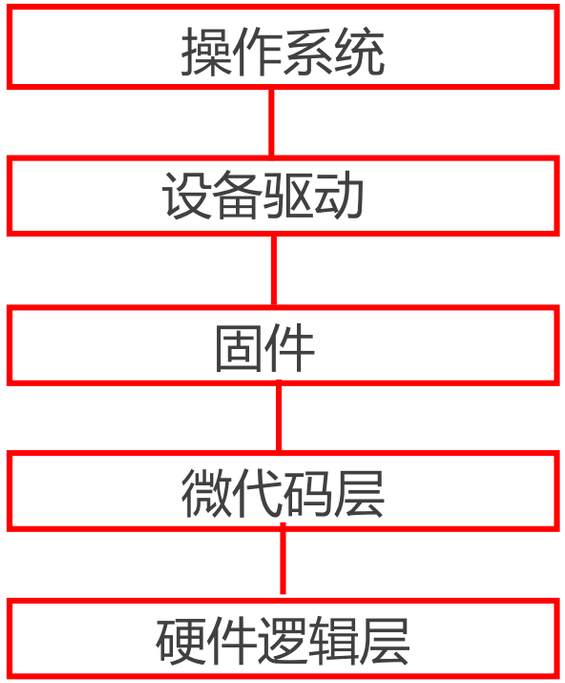
◆ 4.3 软/硬件的分界线

计算机的层次结构



- 分界线在哪里？
- 分界线即软、硬件的接口，是指令操作硬件的入口；
- 指令格式及指令的设计与硬件关联！

5 微代码层与固件



微代码层：CPU的“内部翻译器”
功能：将高级指令集架构（ISA）指令（如ADD、JMP）动态拆解为底层硬件能执行的微操作（Micro-operations）。
例如：x86的DIV指令在硬件中可能需数百步微操作完成。
位置：直接嵌入CPU的控制存储器（Control Store），物理上位于处理器芯片内部。
用户视角：程序员完全感知不到它的存在（对ISA透明），仅CPU设计者能修改。

固件：设备的“基础操作系统”
功能：为整个硬件设备提供最基础的控制逻辑，包括：
初始化硬件（如BIOS检测内存）
提供硬件抽象层（如硬盘固件管理坏块）
实现设备专属协议（如路由器处理网络包）
位置：存储在设备外部的非易失存储器（如主板的SPI Flash、硬盘的ROM芯片）。可通过工具更新（如UEFITool刷BIOS）

微代码层（Microcode Layer）和固件（Firmware）虽都属于“软硬交界”的底层代码，但作用范围、抽象层级和设计目标有本质区别。

特性	微代码层（Microcode）	固件（Firmware）
位置	**CPU内部**（固化在处理器控制存储器中）	**设备外部**（存储在ROM/Flash等非易失存储器中）
作用对象	**仅针对CPU自身**（将复杂指令翻译为电路操作）	**针对整个硬件设备**（控制设备基础功能）
可修改性	通常由厂商提供补丁，**用户不可直接编写**	**可由设备厂商更新**（如刷BIOS、升级路由器固件）
抽象层级	**ISA层之下**（硬件逻辑与指令集的桥梁）	**跨多个层级**（可能包含驱动、Bootloader等）
典型例子	x86 CPU将 `MOV` 指令拆解为微操作序列	BIOS/UEFI、路由器固件、硬盘控制器固件