7、基于CPU指令集的编程

以**ENIAC**为代表的第一代电子计算机，是一个完全可编程的平台，自它问世之后，程序员这个职业正式出现，软件行业也开始萌芽。

1. 编程平台与编程接口

今天的程序员编写应用软件，是基于N层软硬件平台之上的，如下图1所示：

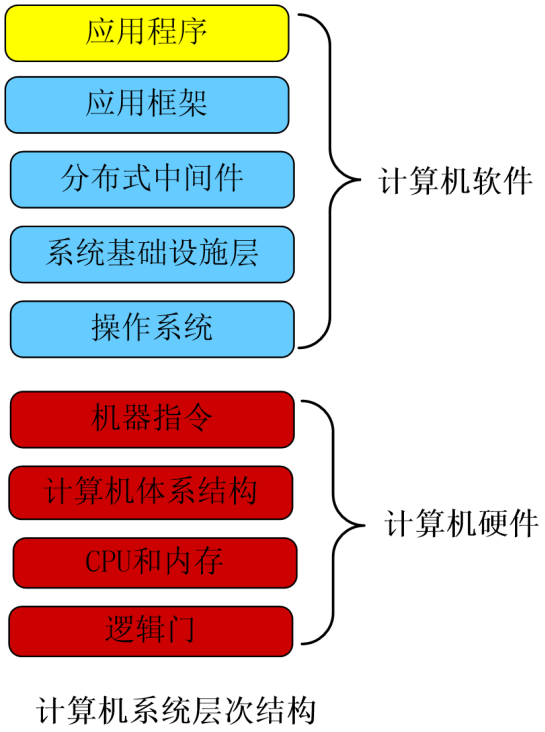

图1、计算机系统软硬件层次结构

然而在计算机的上古年代（1946~1976），程序员写程序，任何可以借力复用的软件基础都没有，什么应用框架、中间件，没有！连操作系统都没有，面对是一台“光板”计算机，如下图2所示：

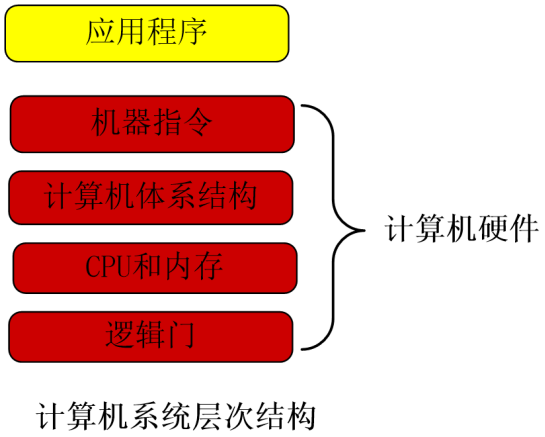

图2、上古时期的编程环境

中央处理器CPU包含运算器、控制器和寄存器，是计算机的大脑。

从**硬件**的角度来看，CPU就是一个大规模电路，通过电路实现了加法、乘法乃至各种各样的处理逻辑。

如果我们从**软件**工程师的角度来看，CPU就是一个执行各种**计算机指令**（Instruction Code）的逻辑机器。这里的计算机指令，就好比一门CPU能够听懂的语言，我们也可以把它叫做**机器语言**（Machine Language）。

不同的CPU能够听懂的语言不太一样。比如，我们个人电脑用的是Intel的CPU，苹果手机用的是ARM的CPU。两者能够听懂的语言就不太一样。类似这两种CPU各自支持的语言，就是两组不同的**计算机指令集**（Instruction Set）。

也就是说，**上古时期的编程接口就是CPU的指令集。**

2. 冯诺依曼平台程序的基本特点

1. **构成**：程序由一系列指令构成，指令由操作码和地址码组成，操作码用来表示操作的性质，地址码用来表示操作数在存储器中的地址；
2. **存储**：程序跟数据一样，以二进制形式存在内部存储器中；
3. **执行**：程序执行时，控制器按顺序从主存储器中取出指令一条一条地执行，这一概念称作**顺序执行程序。**

一个计算机程序，不可能只有一条指令，而是由成千上万条指令组成的。但是CPU里不能一直放着所有指令，所以计算机程序执行时是存储在存储器中的。这种程序指令存储在存储器里面的计算机，我们就叫做**存储程序计算机**（Stored-program Computer）。

说到这里，你可能要问了，难道还有不是存储程序型的计算机吗？还记得在“差分机”那一章介绍过的“插线板计算机”吗？它的程序就体现为用不同的电线来链接不同的插口和插座，从而来完成各种计算任务。这种计算机的程序就不是存储型的。

以上3点，是冯诺依曼平台上程序的基本特点，上古时期的编程还有以下特点：

4. **外部形式**：程序的外在存储形式是**穿孔纸带**，参见下图3；
5. **写程序**：使用专门的**穿孔机**在纸带上穿孔；
6. **程序读入**：程序执行前，**穿孔带读取器**将存储在穿孔纸带上的程序读入到计算机内存中，转换成二进制指令；
7. **函数**：在输入指令的时候，为了避免重复劳动复用某个功能，将这个功能的指令打包成一条纸带。每当需要用到这个功能时就输入这条纸带。这条纸带就可以视为最初的**函数**(function)，它将一组指令打包后允许重复地调用。

图3、8位机上的打孔纸带程序

注：以上几条，现在看来都是些常识，但这些常识却绝非无足轻重，相反对编程而言极其重要，因为理解最底层的程序运行原理，对程序员来说是坚实的**内功**。

一个程序少说也有几百上千条指令，所用的纸带足有好几米甚至十几米长，没有个三五天的时间是打不完的。

上大学的时候，老师跟我们说起来他年轻的时候，是要申请机房的计算时间的，拿了一大卷纸带过去以后，去掉卡纸，拿错了纸带等原因以外，算出来的结果有错误的话，在里面寻找错误就是一个让人崩溃的事情，所以当年老一辈的程序员都有很强的能够在纸上把代码写出来一次通过的能力。与现在各种的IDE相比，那简单就是长矛与钢炮的区别。

第一代的计算机是由许多庞大且昂贵的真空管组成，并利用大量的电力来使真空管发光。1947年9月9日，由于计算机运行产生的光和热，引得一只小虫子Bug钻进了一台Mark 2计算机并卡在了第70号继电器上，导致死机。研究人员费了半天时间，总算发现原因所在，把这只小虫子从真空管中取出后，计算机又恢复正常。这就是**历史上第一个Bug**！后来，Bug这个名词就沿用下来，表示电脑系统或程序中隐藏的错误、缺陷或问题。

3. 机器语言编程示例

为了简单明了地演示机器语言编程的原理，我们这里选择的平台，既不是Intel的CPU，也不是ARM的CPU，而是虚构了一个CPU，我们叫它MyCPU。

3.1 指令集

MyCPU的指令集很简单，只有10个指令：mov,add,and,or,xor,sub,shift,load,store,halt，对应的值分别是二进制的 0000 到 1001，即：

指令	mov	add	and	or	xor	sub	shift	load	store	halt
值	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

3.2 寄存器

此外，MyCPU带有8个寄存器（r），分别对应了二进制的 0000 到 0111（r0 - r7 ），即：

寄存器	r0	r1	r2	r3	r4	r5	r6	r7
地址	0000	0001	0010	0011	0100	0101	0110	0111

3.3 内存

假设MyCPU可以寻址8位的内存，即从0x00到0xff。

3.4 程序语义

下面我们设想一段程序如下：

```
将一个内存地址(例如 0x1a)放到寄存器r0
寄存器r0的地址所对应的数据读取到寄存器r1
将另一个内存地址(例如0x2c)放到寄存器r0
将寄存器r0的地址所对应的数据读取到寄存器r2
将寄存器r1和寄存器r2的数值相加，存储到寄存器r3
将第三个地址（例如0x3e）放到寄存器r0
将寄存器r3的数值写入寄存器r0所对应的地址
```

显然，以上这段程序执行的操作是两个数相加。

主要都是寄存器和内存地址的操作。

这就是基于CPU指令集编程的特点，也是计算机软件的本质。

3.5 汇编代码

以上程序语义，用汇编语言实现，大致看起来应该是：

```
#0 mov r0 0x1a
#1 load r1 [r0]
#2 mov r0 0x2c
#3 load r2 [r0]
#4 add r3 r1 r2
#5 mov r0 0x3e
#6 store r3 [r0]
#7 halt
```

下面我们来试着把以上汇编代码翻译成二进制代码。

3.6 指令规范

首先，规定MyCPU的指令规范是：

指令(4位)+寄存器地址(4位)+内存地址(8位)

MyCPU有10个指令，根据指令规范就是：

汇编命令	操作码	地址码1	地址码2	地址码3
mov	0000	xxxx	yyyy	zzzz
add	0001	xxxx	yyyy	zzzz
and	0010	xxxx	yyyy	zzzz
or	0011	xxxx	yyyy	zzzz
xor	0100	xxxx	yyyy	zzzz
sub	0101	xxxx	yyyy	zzzz
shift	0110	xxxx	yyyy	zzzz
load	0111	xxxx	yyyy	zzzz
store	1000	xxxx	yyyy	zzzz

不同的指令后面跟着的地址码参数的含义回有所不同，比如，mov命令(0000)后面跟着的地址码1表示的是寄存器地址，地址码2与地址码3合并起来，表示内存地址；而add命令(0001)后面跟着的3个地址码，都是寄存器地址。

3.7 机器语言代码

根据MyCPU的指令规范，我们将两数相加的程序翻译成机器语言，大致是这个样子：

```
#0 0000 0000 0001 1010 - mov r0 0x1a
#1 0111 0001 0000 0000 - load r1 [r0]
#2 0000 0000 0010 1100 - mov r0 0x2c
#3 1000 0010 0000 0000 - load r2 [r0]
#4 0001 0011 0001 0010 - add r3 r1 r2
#5 0000 0000 0011 1110 - mov r0 0x3e
#6 1000 0011 0000 0000 - store r3 [r0]
#7 1000 0000 0000 0000 - halt
```

注：不管是intel的CPU还是ARM的CPU，其指令集与指令规范要比MyCPU复杂，但核心思想是一样的。

4. 冯诺依曼平台软件的整体构架

上一章讲过，**冯诺依曼机是以存储器为核心。**这对冯诺依曼程序产生了深远的影响，这导致**冯诺依曼程序的本质是数据处理。**

数据处理是一个动态过程，因此理解**数据如何流动**是理解冯诺依曼程序结构的关键。

示例：在聊天软件中，与朋友之间的对话，在这个体系中，数据如何是流动的?

这个问题要分为两个方面去考虑：1、在自己电脑上的数据流动；2、在对方电脑的数据流动

在自己电脑上，数据流动过程参见下图5：

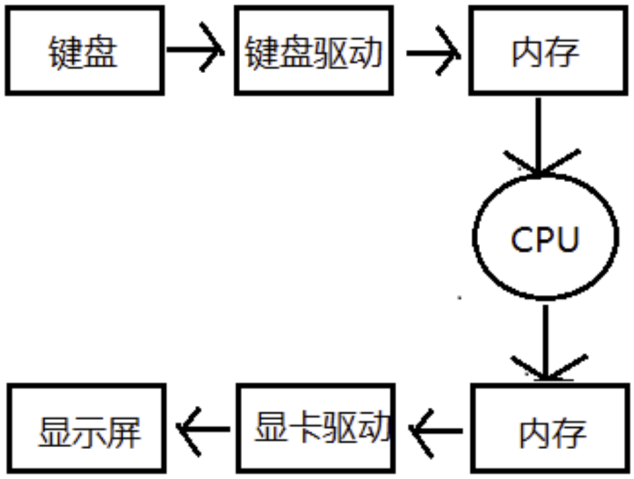

图5、单方面的数据流动

在双方电脑上，数据流动过程参见下图6：

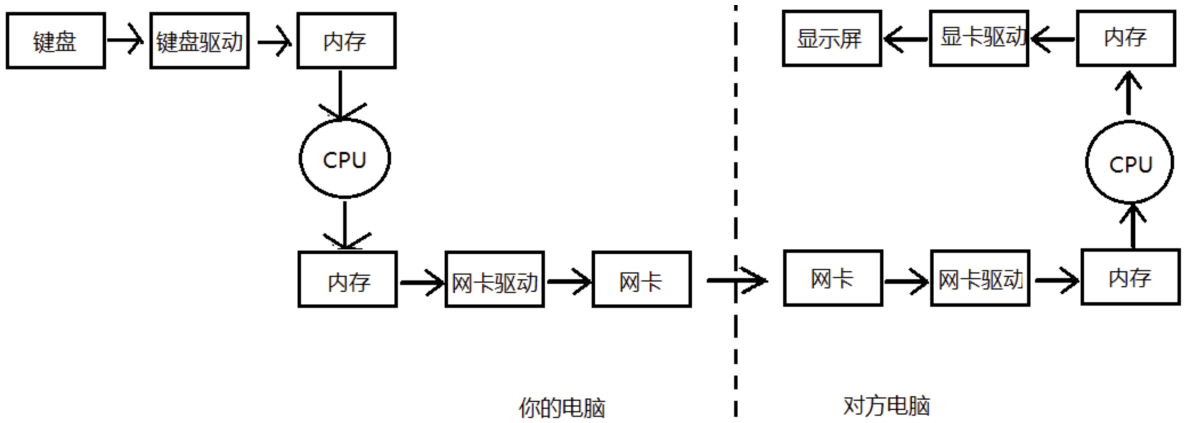

图6、双方电脑上的数据流动

图5可以反映出几个技术要点：

1. 输入输出设备与CPU都只与存储器打交道；
2. 键盘、网卡、显示屏，这些输入输出设备都需要安装驱动程序；
3. 通过网卡进行网络传输数据。