高级程序员的修养，不得不知的硬核知识——CPU

大家都是程序员，大家都是和计算机打交道的程序员，大家都是和计算机中软件硬件打交道的程序员，大家都是和 CPU 打交道的程序员，所以，不管你是玩儿硬件的还是做软件的，你的世界都少不了计算机最核心的 - CPU

■ CPU 是什么

CPU 的全称是 Central Processing Unit，它是你的电脑中最硬核的组件，这种说法一点不为过。CPU 是能够让你的计算机叫计算机的核心组件，但是它却不能代表你的电脑，CPU 与计算机的关系就相当于大脑和人的关系。它是一种小型的计算机芯片，它嵌入在台式机、笔记本电脑或者平板电脑的主板上。通过在单个计算机芯片上放置数十亿个微型晶体管来构建 CPU。 这些晶体管使它能够执行运行存储在系统内存中的程序所需的计算，也就是说 CPU 决定了你电脑的计算能力。

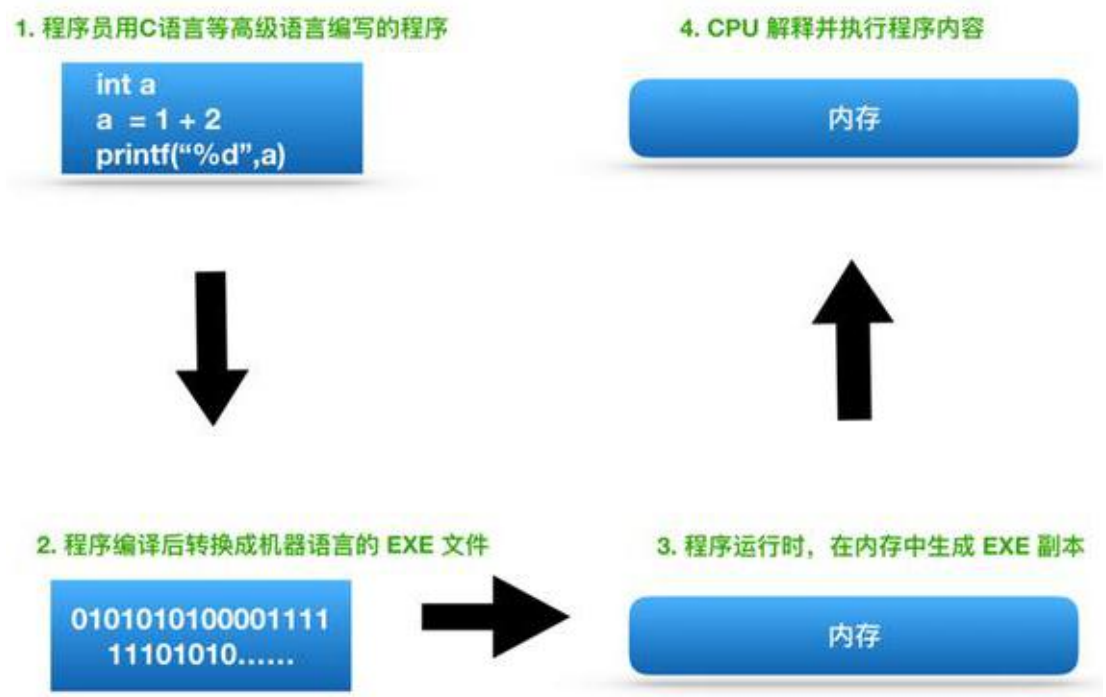
■ CPU 实际做什么

CPU 的核心是从程序或应用程序获取指令并执行计算。此过程可以分为三个关键阶段：提取，解码和执行。CPU 从系统的 RAM 中提取指令，然后解码该指令的实际内容，然后再由 CPU 的相关部分执行该指令。

RAM： 随机存取存储器（英语：Random Access Memory，缩写：RAM），也叫主存，是与 CPU 直接交换数据的内部存储器。它可以随时读写（刷新时除外），而且速度很快，通常作为操作系统或其他正在运行中的程序的临时数据存储介质

■ CPU 的内部结构

说了这么多 CPU 的重要性，那么 CPU 的内部结构是什么呢？又是由什么组成的呢？下图展示了一般程序的运行流程（以 C 语言为例），可以说了解程序的运行流程是掌握程序运行机制的基础和前提。



程序编译执行的过程

在这个流程中，CPU 负责的就是解释和运行最终转换成机器语言的内容。

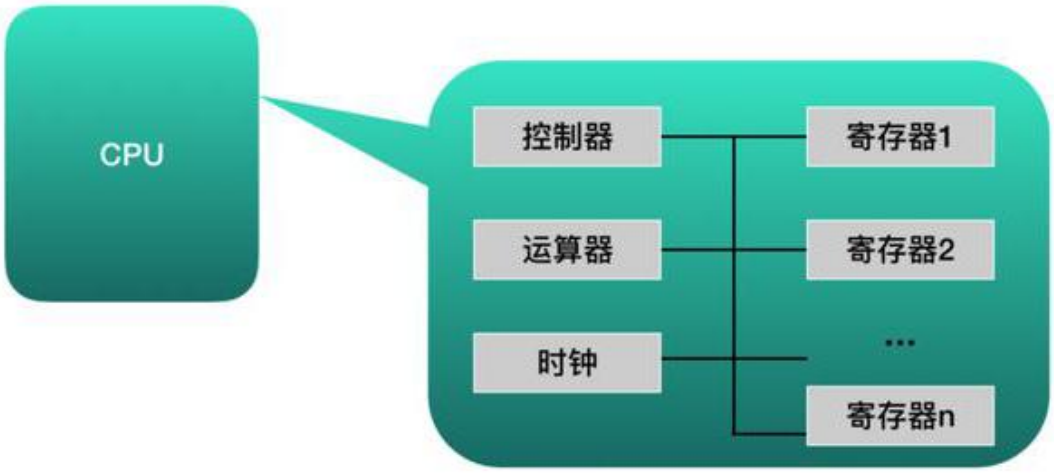
CPU 主要由两部分构成：控制单元 和 算术逻辑单元（ALU）

控制单元：从内存中提取指令并解码执行

算数逻辑单元（ALU）：处理算数和逻辑运算

CPU 是计算机的心脏和大脑，它和内存都是由许多晶体管组成的电子部件。它接收数据输入，执行指令并处理信息。它与输入/输出（I／O）设备进行通信，这些设备向 CPU 发送数据和从 CPU 接收数据。

从功能来看，CPU 的内部由**寄存器、控制器、运算器和时钟**四部分组成，各部分之间通过电信号连通。



CPU 内部结构图

寄存器是中央处理器内的组成部分。它们可以用来暂存指令、数据和地址。可以将其看作是内存的一种。根据种类的不同，一个 CPU 内部会有 20 - 100 个寄存器。

控制器负责把内存上的指令、数据读入寄存器，并根据指令的结果控制计算机运算器负责运算从内存中读入寄存器的数据时钟负责发出 CPU 开始计时的时钟信号。

接下来简单解释一下内存，为什么说 CPU 需要讲一下内存呢，因为内存是与 CPU 进行沟通的桥梁。计算机所有程序的运行都是在内存中运行的，内存又被称为主存，其作用是存放 CPU 中的运算数据，以及与硬盘等外部存储设备交换的数据。只要计算机在运行中，CPU 就会把需要运算的数据调到主存中进行运算，当运算完成后 CPU 再将结果传送出来，主存的运行也决定了计算机的稳定运行。

主存通过控制芯片与 CPU 进行相连，由可读写的元素构成，每个字节（1 byte = 8 bits）都带有一个地址编号，注意是一个字节，而不是一个位。CPU 通过地址从主存中读取数据和指令，也可以根据地址写入数据。注意一点：当计算机关机时，内存中的指令和数据也会被清除。

■ CPU 是寄存器的集合体

在 CPU 的四个结构中，我们程序员只需要了解寄存器就可以了，其余三个不用过多关注，为什么这么说？因为程序是把寄存器作为对象来描述的。

说到寄存器，就不得不说到汇编语言，我大学是学信息管理与信息系统的，我就没有学过汇编这门课（就算有这门课也不会好好学 hhhh），出来混总是要还的，要想作为一个硬核程序员，不能不了解这些概念。说到汇编语言，就不得不说到高级语言，说到高级语言就不得不牵扯出语言这个概念。

■ 计算机语言

我们生而为人最明显的一个特征是我们能通过讲话来实现彼此的交流，但是计算机听不懂你说的话，你要想和他交流必须按照计算机指令来交换，这就涉及到语言的问题，计算机是由二进制构成的，它只能听的懂二进制也就是机器语言，但是普通人是无法看懂机器语言的，这个时候就需要一种电脑既能识别，人又能理解的语言，最先出现的就是汇编语言。但是汇编语言晦涩难懂，所以又出现了像是 C，C++，Java 的这种高级语言。

所以计算机语言一般分为两种：低级语言（机器语言，汇编语言）和高级语言。使用高级语言编写的程序，经过编译转换成机器语言后才能运行，而汇编语言经过汇编器才能转换为机器语言。

■ 汇编语言

首先来看一段用汇编语言表示的代码清单

```
mov eax, dword ptr [ebp-8] /* 把数值从内存复制到 eax */
add eax, dword ptr [ebp-0Ch] /* 把 eax 的数值和内存的数值相加 */
mov dword ptr [ebp-4], eax /* 把 eax 的数值（上一步的结果）存储在内存中*/
```

这是采用汇编语言（assembly）编写程序的一部分。汇编语言采用 助记符(memonic) 来编写程序，每一个原本是电信号的机器语言指令会有一个与其对应的助记符，例如 mov,add 分别是数据的存储（move）和相加（addition）的简写。汇编语言和机器语言是一一对应的。这一点和高级语言有很大的不同，通常我们将汇编语言编写的程序转换为机器语言的过程称为 汇编；反之，机器语言转化为汇编语言的过程称为反汇编。

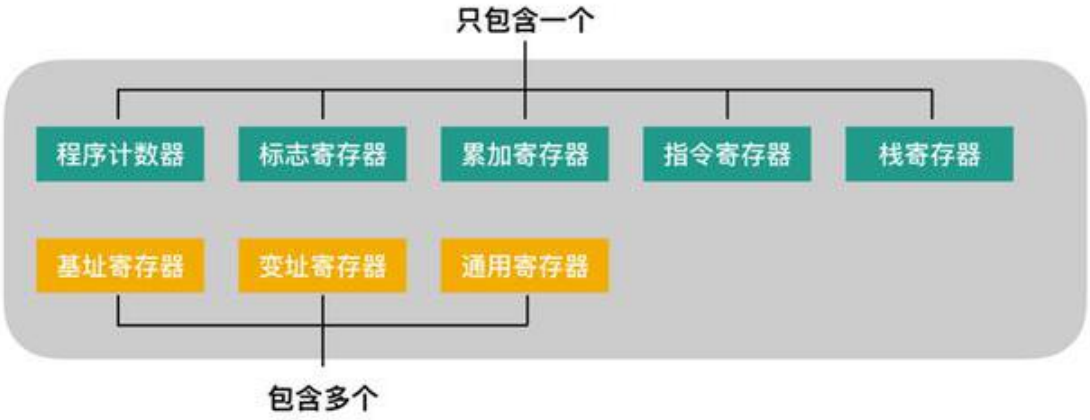
汇编语言能够帮助你理解计算机做了什么工作，机器语言级别的程序是通过寄存器来处理的，上面代码中的 eax,ebp 都是表示的寄存器，是 CPU 内部寄存器的名称，所以可以说 CPU 是一系列**寄存器的集合体**。在内存中的存储通过地址编号来表示，而寄存器的种类则通过名字来区分。

不同类型的 CPU ，其内部寄存器的种类，数量以及寄存器存储的数值范围都是不同的。不过，根据功能的不同，可以将寄存器划分为下面这几类

种类功能累加寄存器存储运行的数据和运算后的数据。标志寄存器用于反应处理器的状态和运算结果的某些特征以及控制指令的执行。程序计数器程序计数器是用于存放下一条指令所在单元的地址的地方。基址寄存器存储数据内存的起始位置变址寄存器存储基址寄存器的相对地址通用寄存器存储任意数据指令寄存器储存正在被运行的指令，CPU 内部使用，程序员无法对该寄存器进行读写栈寄存

器存储栈区域的起始位置

其中**程序计数器、累加寄存器、标志寄存器、指令寄存器和栈寄存器**都只有一个，其他寄存器一般有多个。



程序员眼中的 CPU

■ 程序计数器

程序计数器(Program Counter)是用来存储下一条指令所在单元的地址。程序执行时，PC 的初值为程序第一条指令的地址，在顺序执行程序时，控制器首先按程序计数器所指出的指令地址从内存中取出一条指令，然后分析和执行该指令，同时将 PC 的值加 1 指向下一条要执行的指令。

我们还是以一个事例为准来详细的看一下程序计数器的执行过程

内存中配置程序示例

这是一段进行相加的操作，程序启动，在经过编译解析后会由操作系统把硬盘中的程序复制到内存中，示例中的程序是将 123 和 456 执行相加操作，并将结果输出到显示器上。由于使用机器语言难以描述，所以这是经过翻译后的结果，实际上每个指令和数据都可能分布在不同的地址上，但为了方便说明，把组成一条指令的内存和数据放在了一个内存地址上。

地址 0100 是程序运行的起始位置。Windows 等操作系统把程序从硬盘复制到内存后，会将程序计数器作为设定为起始位置 0100，然后执行程序，每执行一条指令后，程序计数器的数值会增加 1（或者直接指向下一条指令的地址），然后，CPU 就会根据程序计数器的数值，从内存中读取命令并执行，也就是说，程序计数器控制着程序的流程。

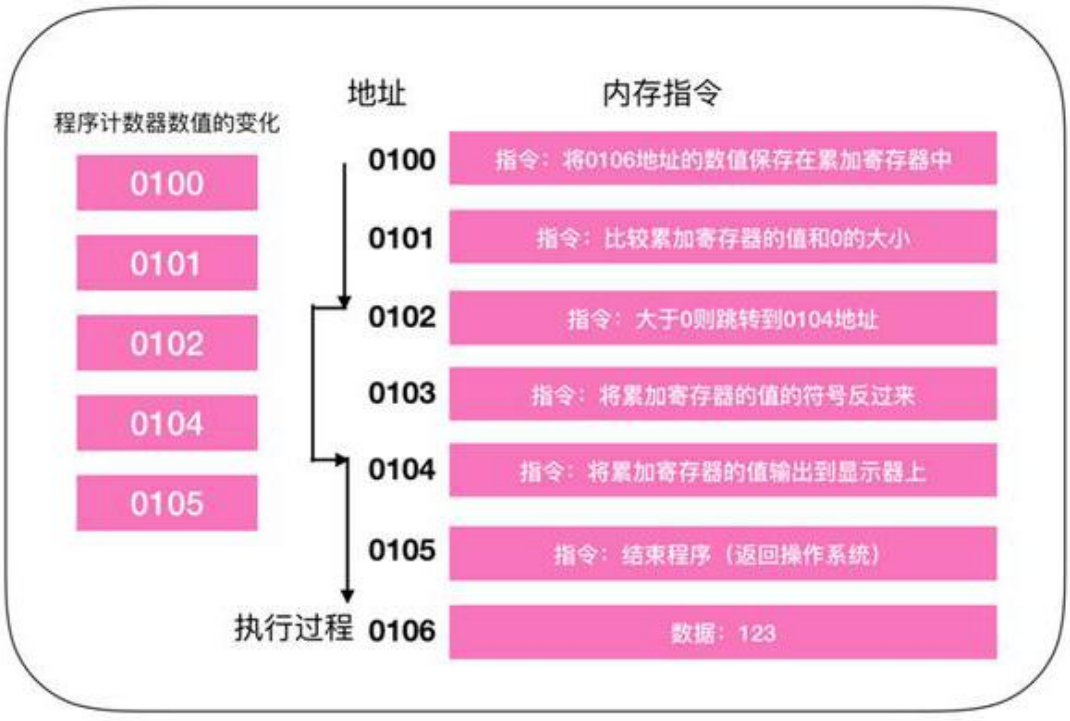
■ 条件分支和循环机制

我们都学过高级语言，高级语言中的条件控制流程主要分为三种：顺序执行、条件分支、循环判断三种，顺序执行是按照地址的内容顺序的执行指令。条件分支是根据条件执行任意地址的指令。循环是重复执行同一地址的指令。

顺序执行的情况比较简单，每执行一条指令程序计数器的值就是 + 1。

条件和循环分支会使程序计数器的值指向任意的地址，这样一来，程序便可以返回到上一个地址来重复执行同一个指令，或者跳转到任意指令。

下面以条件分支为例来说明程序的执行过程（循环也很相似）



条件循环的执行流程

条件循环的执行流程

程序的开始过程和顺序流程是一样的，CPU 从 0100 处开始执行命令，在 0100 和 0101 都是顺序执行，PC 的值顺序+1，执行到 0102 地址的指令时，判断 0106 寄存器的数值大于 0，跳转（jump）到 0104 地址的指令，将数值输出到显示器中，然后结束程序，0103 的指令被跳过了，这就和我们程序中的 if() 判断是一样的，在不满足条件的情况下，指令会直接跳过。所以 PC 的执行过程也就没有直接+1，而是下一条指令的地址。

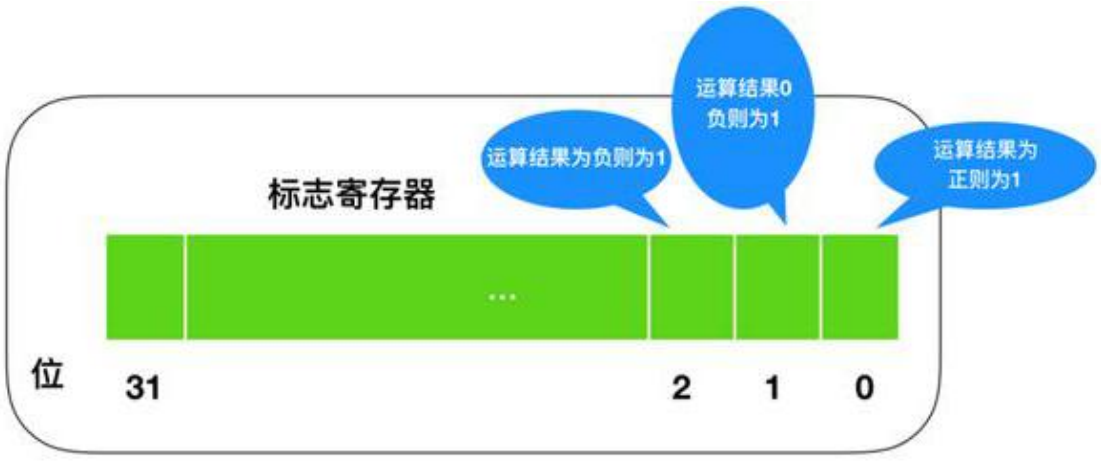
■ 标志寄存器

条件和循环分支会使用到 jump（跳转指令），会根据当前的指令来判断是否跳转，上面我们提到了标志寄存器，无论当前累加寄存器的运算结果是正数、负数还是零，标志寄存器都会将其保存（也负责溢出和奇偶校验）

溢出（overflow）：是指运算的结果超过了寄存器的长度范围

奇偶校验（parity check）：是指检查运算结果的值是偶数还是奇数

CPU 在进行运算时，标志寄存器的数值会根据当前运算的结果自动设定，运算结果的正、负和零三种状态由标志寄存器的三个位表示。标志寄存器的第一个字节位、第二个字节位、第三个字节位各自的结果都为 1 时，分别代表着正数、零和负数。

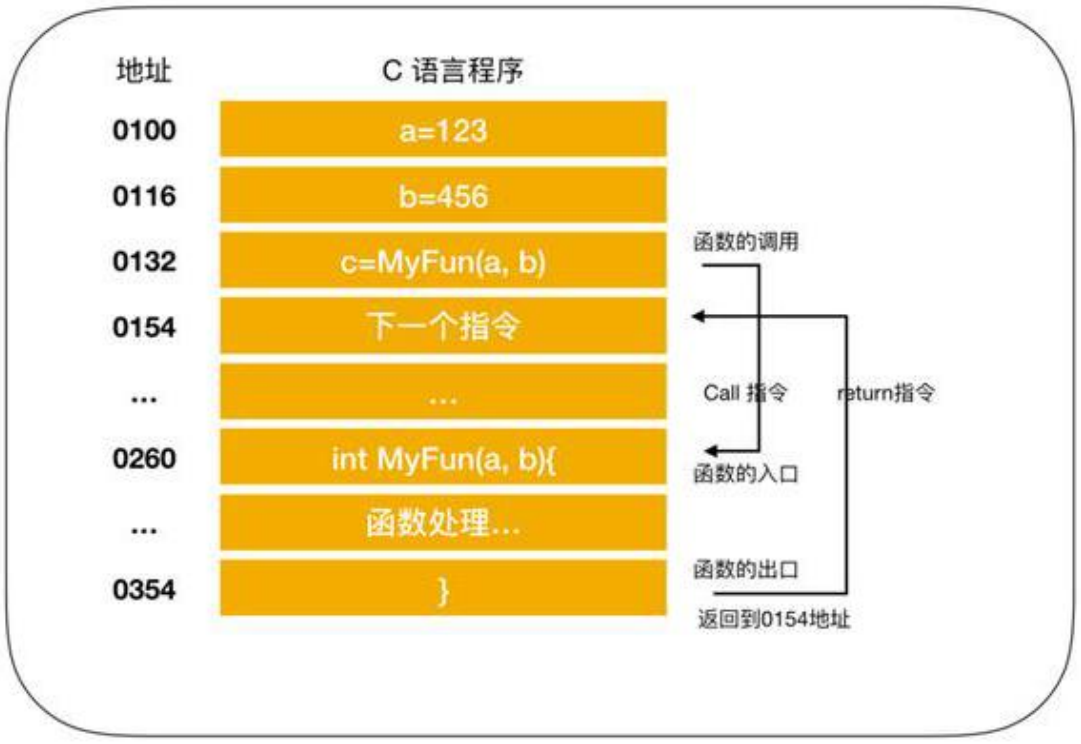


比较运算的标志寄存器示意图

CPU 的执行机制比较有意思，假设累加寄存器中存储的 XXX 和通用寄存器中存储的 YYY 做比较，执行比较的背后，CPU 的运算机制就会做减法运算。而无论减法运算的结果是正数、零还是负数，都会保存到标志寄存器中。结果为正表示 XXX 比 YYY 大，结果为零表示 XXX 和 YYY 相等，结果为负表示 XXX 比 YYY 小。程序比较的指令，实际上是在 CPU 内部做减法运算。

■ 函数调用机制

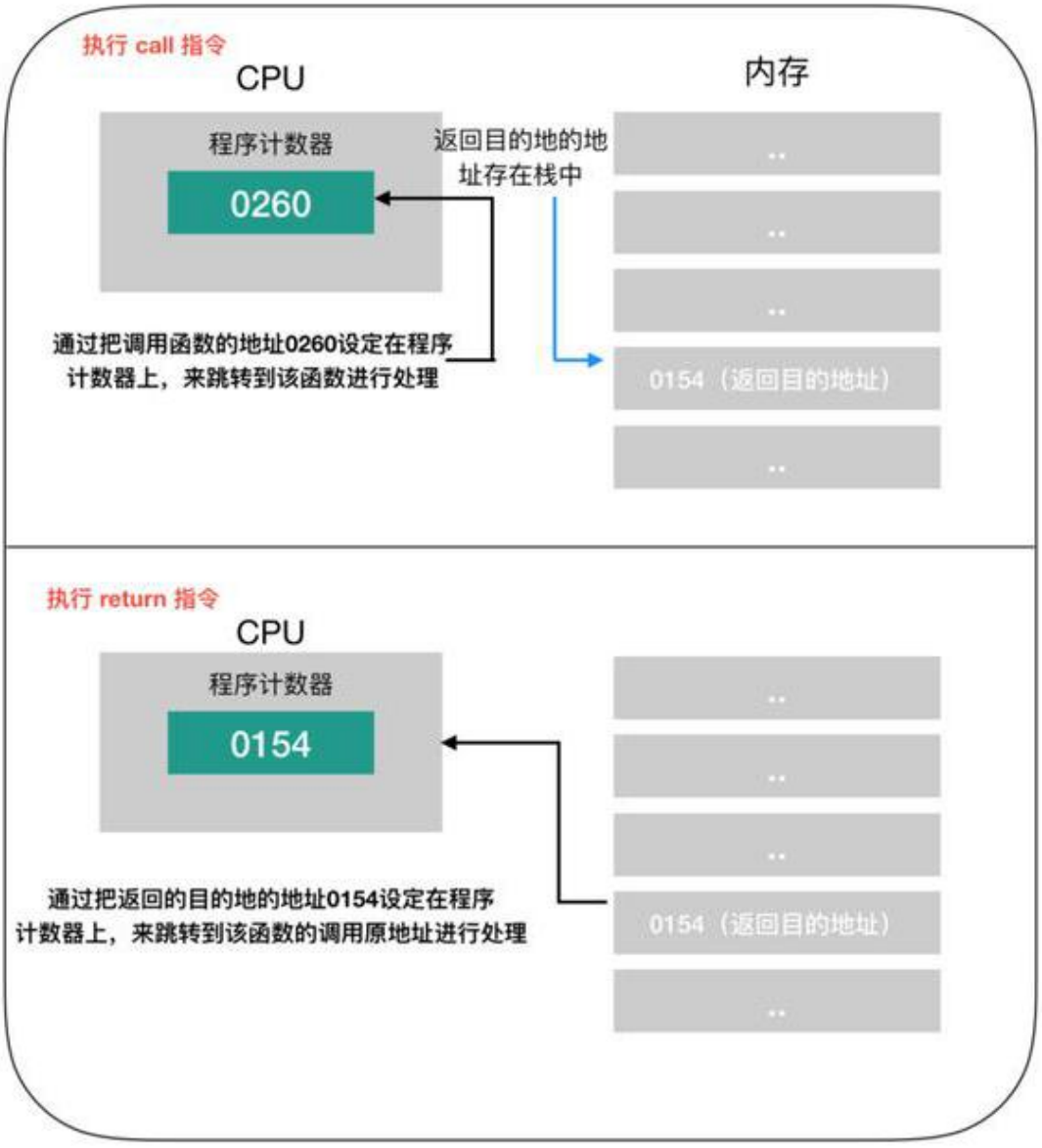
接下来，我们继续介绍函数调用机制，哪怕是高级语言编写的程序，函数调用处理也是通过把程序计数器的值设定成函数的存储地址来实现的。函数执行跳转指令后，必须进行返回处理，单纯的指令跳转没有意义，下面是一个实现函数跳转的例子



程序调用示意图

图中将变量 a 和 b 分别赋值为 123 和 456 ，调用 MyFun(a,b) 方法，进行指令跳转。图中的地址是将 C 语言编译成机器语言后运行时的地址，由于 1 行 C 程序在编译后通常会变为多行机器语言，所以图中的地址是分散的。在执行完 MyFun(a,b)指令后，程序会返回到 MyFun(a,b) 的下一条指令，CPU 继续执行下面的指令。

函数的调用和返回很重要的两个指令是 call 和 return 指令，再将函数的入口地址设定到程序计数器之前，call 指令会把调用函数后要执行的指令地址存储在名为栈的主存内。函数处理完毕后，再通过函数的出口来执行 return 指令。return 指令的功能是把保存在栈中的地址设定到程序计数器。MyFun 函数在被调用之前，0154 地址保存在栈中，MyFun 函数处理完成后，会把 0154 的地址保存在程序计数器中。这个调用过程如下



函数调用中程序计数器和栈的职能

在一些高级语言的条件或者循环语句中，函数调用的处理会转换成 call 指令，函数结束后的处理则会转换成 return 指令。

■ 通过地址和索引实现数组

接下来我们看一下基址寄存器和变址寄存器，通过这两个寄存器，我们可以对主存上的特定区域进行划分，来实现类似数组的操作，首先，我们用十六进制数将计算机内存上的 00000000 - FFFFFFFF 的地址划分出来。那么，凡是该范围的内存地址，只要有一个 32 位的寄存器，便可查看全部地址。但如果想要想数组那样分割特定的内存区域以达到连续查看的目的的话，使用两个寄存器会更加方便。

例如，我们用两个寄存器（基址寄存器和变址寄存器）来表示内存的值

基址寄存器和变址寄存器

这种表示方式很类似数组的构造，数组是指同样长度的数据在内存中进行连续排列的数据构造。用数组名表示数组全部的值，通过索引来区分数组的各个数据元素，例如：a[0] - a[4]，[]内的 0 - 4 就是数组的下标。

■ CPU 指令执行过程

那么 CPU 是如何执行一条条的指令的呢？

几乎所有的冯·诺伊曼型计算机的 CPU，其工作都可以分为 5 个阶段：取指令、指令译码、执行指令、访存取数、结果写回。

取指令阶段是将内存中的指令读取到 CPU 中寄存器的过程，程序寄存器用于存储下一条指令所在的地址

指令译码阶段，在取指令完成后，立马进入指令译码阶段，在指令译码阶段，指令译码器按照预定的指令格式，对取回的指令进行拆分和解释，识别区分出不同的指令类别以及各种获取操作数的方法。

执行指令阶段，译码完成后，就需要执行这一条指令了，此阶段的任务是完成指令所规定的各种操作，具体实现指令的功能。

访问取数阶段，根据指令的需要，有可能需要从内存中提取数据，此阶段的任务是：根据指令地址码，得到操作数在主存中的地址，并从主存中读取该操作

数用于运算。

结果写回阶段，作为最后一个阶段，结果写回（Write Back，WB）阶段把执行指令阶段的运行结果数据“写回”到某种存储形式：结果数据经常被写到 CPU 的内部寄存器中，以便被后续的指令快速地存取；

■ 总结

本篇文章我们主要讲述了

CPU 是什么，CPU 的重要性，CPU 执行程序的过程

还讲述了 CPU 的内部结构，它的组成部分

提到了汇编语言和高级语言

提到了 CPU 与 寄存器的关系

提到了主要的寄存器的功能，程序计数器，标志寄存器，基址寄存器和变址寄存器

还提到了函数调用机制是怎样的。

CPU 指令的执行过程