程序是怎样跑起来的

TURING
图灵程序设计丛书

[日] 矢泽久雄 / 著　日经Software / 审校　李逢俊 / 译

日文版重印41次!

How Program Works

“计算机组成原理”图解趣味版

蹲马桶就能看懂的编程基础知识

本书适合

1 菜鸟程序员入门进阶　2 中级程序员查漏补缺
3 高手程序员向家人（女友、老妈等）普及计算机知识

中国工信出版集团　人民邮电出版社 POSTS & TELECOM PRESS

版权信息

书名：程序是怎样跑起来的

作者：［日］矢泽久雄

译者：李逢俊

ISBN：978-7-115-38513-0

本书由北京图灵文化发展有限公司发行数字版。版权所有，侵权必究。

您购买的图灵电子书仅供您个人使用，未经授权，不得以任何方式复制和传播本书内容。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

版权声明

PROGRAM WA NAZE UGOKUNOKA DAI 2 HAN

written by Hisao Yazawa.

Copyright © 2007 by Hisao Yazawa.

All rights reserved.

Originally published in Japan by Nikkei Business Publications, Inc.

Simplified Chinese translation rights arranged with

Nikkei Business Publications, Inc. through CREEK & RIVER Co., Ltd.

本书中文简体字版由 Nikkei Business Publications, Inc. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

前言

大家还记得自己初次接触计算机时的情形吗？想必多数读者使用的都是 Windows 系统，应该也有不少读者使用 Visual Studio 和 Java 等集成开发环境（IDE，Integrated Development Environment，即集成了编程所需的各种工具的开发软件）开发过程序。Windows 的图形化操作界面，大大提高了计算机操作的便利性，而利用集成开发环境开发程序，就像用绘图软件画图一样简单。由此可见，这是一个便利的时代。

然而，现实却不容乐观，我们在享受这些方便的同时也付出了代价。虽然拥有一定的编程能力，却无法进一步提高自身技能；知识应用能力的不足导致无法编写源程序。越来越多的程序员正为这些问题而烦恼。个中原因在于，大家不了解程序运行的根本机制。

“双击程序图标，程序开始运行”，作为一名程序员，对程序的了解仅仅停留在这一表层是不行的。我们还应该了解更深层的机制：加载到内存中的机器语言程序，由 CPU 进行解析和运行，进而计算机系统整体的控制和数据运算也开始运行。了解了程序的运行机制后，就能找到编写源程序的方法。

本书以通俗易懂的方式来解析程序的运行机制，适合想要学习编程的读者，迫切希望提升技能的初级程序员，以及对计算机较为熟悉的中级用户阅读。为了便于说明，书中涉及了不少计算机硬件知识，不过本书的主题依然是编程，也就是软件。

《日经 Software》杂志上连载过名为“程序是怎样跑起来的”的文章，而本书就是在整合以上内容的基础上创作的。2001 年 10 月，本书第 1 版出版后，受到了众多读者的欢迎，我们也收到了很多反馈信息。大部分读者表示“了解了 CPU 的寄存器和内存的运行方式，也知道了自己编写的程序的运行机制，收获颇丰。不过也有编程经验较少的读者表示“内容有点难”。

值此第 2 版出版之际，我再次核对了全文，大幅增加了寄存器和栈等内容的相关说明，并作了详细的注释。实例程序的代码也由原来的 Visual BASIC 语言，换成了更便于说明程序运行机制的 C 语言，并在书的末尾添加了一个辅助章节，对 C 语言进行了简单的介绍。通过这样的改动，相信即便是觉得第 1 版有点难的读者，也会感到满意。

无论任何事情，了解其本质非常重要。只有了解了本质才能提高利用效率。这样一来，即使有新技术出现，也能很容易地理解并掌握。接下来，就让我们一起在本书中探索程序的奥秘，寻求程序的本质吧。

矢沢久雄

程序是怎样跑起来的——本书中涉及的主要关键词

读完本书，你就会了解从双击程序图标开始到程序运行的整个机制。

本书的结构

本书共 12 章，每章由“热身问答”“本章重点”“正文”三部分构成。对专业术语的解说，放在了正文的脚注部分。有些章节还设置了“专栏”。另外，本书的附录部分对 C 语言进行了介绍，刚开始学习编程的朋友，请一定阅读一下。

- 热身问答

  在每章的开头罗列了一些简单的问答，大家不妨在阅读时挑战一下。这样就可以带着问题来阅读正文了。

- 本章重点

  这部分是对正文内容的高度总结。通过阅读这部分，可以确定本章节是否有自己关心的内容。

- 正文

  在这部分中，笔者以简明易懂的方式，从各章节的主题出发，对程序的运行机制进行说明。虽然有时会出现 C 语言程序，但其中做了大量的注释，即使对编程语言不熟悉的朋友也能正常阅读。

- 专栏“如果是你，你会怎样介绍？”

  在这部分中，笔者为大家展示了他向那些不熟悉程序的朋友介绍程序运行机制的过程。通过向他人介绍，可以对自己的掌握程度进行充分的验证。各位读者在阅读时也不妨考虑一下：如果是你，你会怎样介绍呢？

* 本书在写作过程中，尽量避免内容局限在特定的硬件和软件上。但在一些具体的示例中，涉及电脑（特别是 AT 兼容机）、Windows（32 位）以及 Borland C++ 等。
另外，本书中所涉及的 Windows、Borland C++ 等软件，都是以笔者写作当时的最新版为准进行描述的，之后的软件版本可能会有所变化，这一点请注意。

第 **1** 章　对程序员来说 **CPU** 是什么

热身问答

阅读正文前，让我们先回答下面的问题来热热身吧。

问题

1. 程序是什么？

2. 程序是由什么组成的？

3. 什么是机器语言？

4. 正在运行的程序存储在什么位置？

5. 什么是内存地址？

6. 计算机的构成元件中，负责程序的解释和运行的是哪个？

怎么样？是不是发现有一些问题无法简单地解释清楚呢？下面是笔者的答案和解析，供大家参考。

答案

1. 指示计算机每一步动作的一组指令

2. 指令和数据

3. **CPU** 可以直接识别并使用的语言

4. 内存

5. 内存中，用来表示命令和数据存储位置的数值

6. **CPU**

解析

1. 一般所说的程序，譬如运动会、音乐会的程序等，指的是“行事的先后次序”。计算机程序也是一样的道理。

2. 程序是指令和数据的组合体。例如，C 语言“printf (" 你好 ");”这个简单的程序中，printf 是指令，" 你好 " 是数据。

3. CPU 能够直接识别和执行的只有机器语言。使用 C、Java 等语言编写的程序，最后都会转化成机器语言。

4. 硬盘和磁盘等媒介上保存的程序被复制到内存后才能运行。

5. 内存中保存命令和数据的场所，通过地址来标记和指定。地址由整数值表示。

6. 计算机的构成元件中，根据程序的指令来进行数据运算，并控制整个计算机的设备称作 CPU。大家熟知的奔腾（Pentium）就是 CPU 的一种。

本章重点

首先让我们来看一下解释和运行程序的 **CPU**。**CPU** 是英文 **Central Processing Unit**（中央处理器）的缩写，相当于计算机的大脑，它的内部由数百万至数亿个晶体管构成，这些都是大家所熟知的。不过，对 **CPU** 的了解如果只限于此的话，对编程是没有任何帮助的。程序员还需要理解 **CPU** 是如何运行的，特别是要弄清楚负责保存指令和数据的寄存器的机制。了解了寄存器，也就自然而然地理解了程序的运行机制。可能有很多读者会认为 **CPU** 的运行机制比较难，其实它非常简单。所以，不妨放松心情，跟随笔者一起往下阅读吧。

1.1 CPU 的内部结构解析

图 1-1 展示了程序运行的一般流程。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。详细内容会在接下来的章节中逐渐展开，这里主要是希望大家先有个大致印象。在这一流程中，**CPU** [1] 所负责的就是解释和运行最终转换成机器语言的程序内容。

1 CPU 是用来表示计算机内部元件功能的术语。另一方面，奔腾等半导体芯片，通常称为微处理器。不过，由于大部分计算机通常只有一个微处理器来承担 CPU 的功能，所以本章不对此进行区分，统一使用 CPU 这一称呼。CPU 由具有 ON/OFF 开关功能的晶体管构成。另外，有的 CPU 在一个集成电路中集成了两个 CPU 芯片，我们称之为双核（dual core）CPU。

图 **1-1**　程序运行流程示例

CPU 和内存是由许多晶体管组成的电子部件，通常称为 IC（Integrated Circuit，集成电路）。从功能方面来看，如图 1-2 所示，CPU 的内部由寄存器、控制器、运算器和时钟四个部分构成，各部分之间由电流信号相互连通。寄存器　可用来暂存指令、数据等处理对象，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、数据等读入寄存器，并根据指令的执行结果来控制整个计算机。运算器　负责运算从内存读入寄存器的数

据。时钟 负责发出 CPU 开始计时的时钟信号[2] 。不过，也有些计算机的时钟位于 CPU 的外部。

2 时钟信号英文叫作 clock puzzle。Pentium 2 GHz 表示时钟信号的频率为 2 GHz（1 GHz = 10 亿次 / 秒）。也就是说，时钟信号的频率越高，CPU 的运行速度越快。

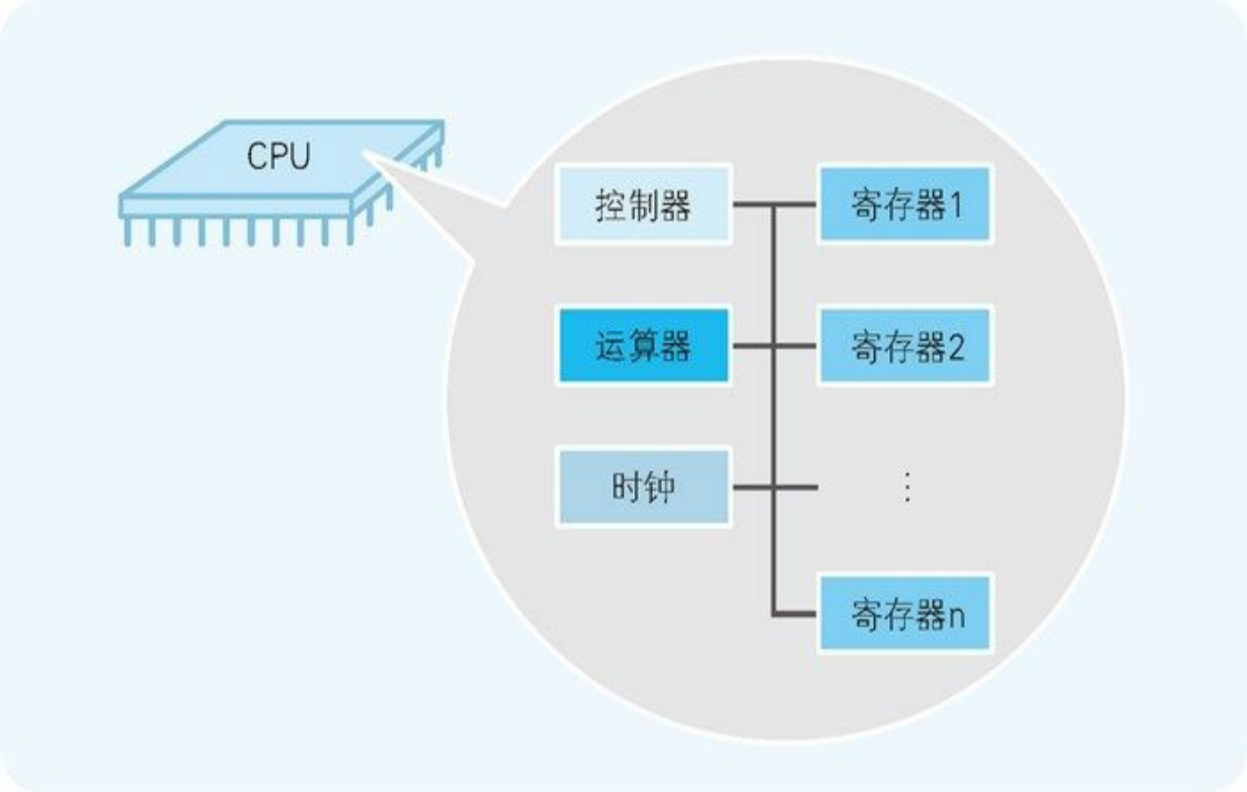

图 **1-2　CPU** 的四个构成部分

接下来简单地解释一下内存。通常所说的内存指的是计算机的主存储器（main memory）[3] ，简称主存。主存通过控制芯片等与 CPU 相连，主要负责存储指令和数据。主存由可读写的元素构成，每个字节（1 字节 = 8 位 ）都带有一个地址编号。CPU 可以通过该地址读取主存中的指令和数据，当然也可以写入数据。但有一点需要注意，主存中存储的指令和数据会随着计算机的关机而自动清除。

3　主存位于计算机机体内部，是负责存储程序、数据等的装置。主存通常使用 DRAM（Dynamic Random Access Memory，动态随机存取存储器）芯片。DRAM 可以对任何地址进行数据的读写操作，但需要保持稳定的电源供给并时常刷新（确保是最新数据），关机后内容将自动清除。关于内存 IC，第 4 章有详细介绍。

了解了 CPU 的构造后，大家对程序的运行机制的理解是不是也加深了一些？程序启动后，根据时钟信号，控制器会从内存中读取指令和数据。通过对这些指令加以解释和运行，运算器就会对数据进行运算，控制器根据该运算结果来控制计算机。看到“控制”一词时，大家可能会将事情想象得过于复杂，其实所谓的控制就是指数据运算以外的处理（主要是数据输入输出的时机控制）。比如内存和磁盘等媒介的输入输出、键盘和鼠标的输入、显示器和打印机的输出等，这些都是控制的内容。

1.2 CPU 是寄存器的集合体

CPU 的四个构成部分中，程序员只需要了解寄存器即可，其余三个都不用太过关注。那么，为什么必须要了解寄存器呢？这是因为程序是把寄存器作为对象来描述的 。

首先我们来看一下代码清单 1-1。这是用汇编语言（assembly）**4** 编写的程序的一部分。汇编语言 采用助记符（memonic）来编写程序，每一个原本是电气信号的机器语言**5** 指令都会有一个与其相应的助记符，助记符通常为指令功能的英语单词的简写。例如，mov 和 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基本上是一一对应的。这一点和 C 语言、Java 语言等高级编程语言**6** 有很大不同，这也是我们使用汇编语言来说明 CPU 运行的原因。通常我们将汇编语言编写的程序转化成机器语言的过程称为汇编 ；反之，机器语言程序转化成汇编语言程序的过程则称为反汇编 。

4 把汇编语言转化成机器语言的程序称为汇编器（assembler）。有时汇编语言也称为汇编。详情可参阅第 10 章。

5 机器语言是指 CPU 能直接解释和执行的语言。

6 高级编程语言是指能够使用类似于人类语言（主要是英语）的语法来记述的编程语言的总称。BASIC、C、C++、Java、Pascal、FORTRAN、COBOL 等语言都是高级编程语言。使用高级编程语言编写的程序，经过编译转换成机器语言后才能运行。与高级编程语言相对，机器语言和汇编语言称为低级编程语言。

代码清单 **1-1** 汇编语言编写的程序示例

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

通过阅读汇编语言编写的代码，能够了解转化成机器语言的程序的运行情况。从代码清单 1-1 的汇编语言程序示例中也可以看出，机器语言级

别的程序是通过寄存器来处理的。也就是说，在程序员看来“CPU 是寄存器的集合体”。至于控制器、运算器和时钟，程序员只需要知道 CPU 中还有这几部分就足够了。

代码清单 1-1 中，eax 和 ebp 表示的都是寄存器。通过阅读刚才的示例代码，想必大家对程序使用寄存器来实现数据的存储和加法运算这一情况应该有所了解了。汇编语言是 80386[7] 以上的 CPU 所使用的语言。eax 和 ebp 是 CPU 内部的寄存器的名称。内存的存储场所通过地址编号来区分，而寄存器的种类则通过名字来区分。

7 80386 是美国英特尔公司开发的微处理器的产品名。“80386 以上”是指 80386、80486、奔腾等微处理器。

上文可能有些难以理解，不过不用担心，因为我们并不要求大家必须掌握 CPU 的所有寄存器种类和汇编语言，大家只需对 CPU 是怎么处理程序的有一个大致印象即可。也就是说，使用高级语言编写的程序会在编译[8]后转化成机器语言，然后再通过 CPU 内部的寄存器来处理。例如，a = 1+2 这样的高级语言的代码程序在转化成机器语言后，就是利用寄存器来进行相加运算和存储处理的。

8 编译是指将使用高级编程语言编写的程序转换为机器语言的过程，其中，用于转换的程序被称为编译器（compiler）。

不同类型的 CPU，其内部寄存器的数量、种类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，我们可以将寄存器大致划分为八类，如表 1-1 所示。可以看出，寄存器中存储的内容既可以是指令也可以是数据。其中，数据分为“用于运算的数值”和“表示内存地址的数值”两种。数据种类不同，存储该数值的寄存器也不同。CPU 中每个寄存器的功能都是不同的。用于运算的数值放在累加寄存器中存储，表示内存地址的数值则放在基址寄存器和变址寄存器中存储。代码清单 1-1 的程序中用到的 eax 和 ebp 分别是累加寄存器和基址寄存器。

表 **1-1** 寄存器 的主要种类和功能

种类	功能

累加寄存器（accumulator register）	存储执行运算的数据和运算后的数据
标志寄存器（flag register）	存储运算处理后的CPU的状态
程序计数器（program counter）	存储下一条指令所在内存的地址
基址寄存器（base register）	存储数据内存的起始地址
变址寄存器（index register）	存储基址寄存器的相对地址
通用寄存器（general purpose register）	存储任意数据
指令寄存器（instruction register）	存储指令。CPU内部使用，程序员无法通过程序对该寄存器进行读写操作
栈寄存器（stack register）	存储栈区域的起始地址

对程序员来说，CPU 是什么呢？如图 1-3 所示，CPU 是具有各种功能的寄存器的集合体。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，其他的寄存器一般有多个。程序计数器和标志寄存器比较特殊，这一点在后面的章节中会详细说明。另外，存储指令的指令寄存器等寄存器，由于不需要程序员做多关注，因此图 1-3 中没有提到。

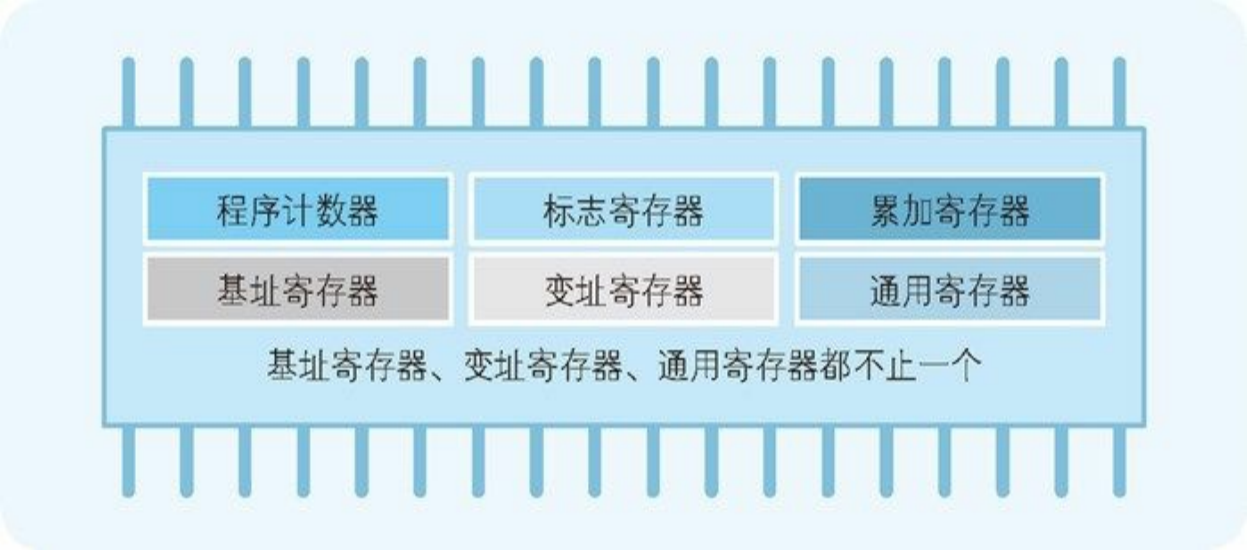

图 **1-3**　程序员眼中的 **CPU**（**CPU** 是寄存器的集合体）

1.3 决定程序流程的程序计数器

只有 1 行的有用程序是很少见的，机器语言的程序也是如此。在对 CPU 有了一个大体印象后，接下来我们看一下程序是如何按照流程来运行的。

图 1-4 是程序起动时内存内容的模型。用户发出启动程序的指示后，Windows 等操作系统[9] 会把硬盘中保存的程序复制到内存中。示例中的程序实现的是将 123 和 456 两个数值相加，并将结果输出到显示器上。正如前文所介绍的那样，存储指令和数据的内存，是通过地址来划分的。由于使用机器语言难以清晰地表明各地址存储的内容，因此这里我们对各地址的存储内容添加了注释。实际上，一个命令和数据通常被存储在多个地址上，但为了便于说明，图 1-4 中把指令、数据分配到了一个地址中。

9 操作系统（operating system）是指管理和控制计算机硬件与软件资源的计算机程序。关于操作系统的功能，第 9 章有详细说明。

地址 0100 是程序运行的开始位置。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（CPU 寄存器的一种）设定为 0100，然后程序便开始运行。CPU 每执行一个指令，程序计数器的值就会自动加 1。例如，CPU 执行 0100 地址的指令后，程序计数器的值就变成了 0101（当执行的指令占据多个内存地址时，增加与指令长度相应的数值）。然后，CPU 的控制器就会参照程序计数器的数值，从内存中读取命令并执行。也就是说，程序计数器决定着程序的流程。

图 **1-4**　内存中配置的程序示例（显示相加的结果）

1.4 条件分支和循环机制

程序的流程分为顺序执行、条件分支和循环三种。顺序执行　是指按照地址内容的顺序执行指令。条件分支　是指根据条件执行任意地址的指令。循环　是指重复执行同一地址的指令。顺序执行的情况比较简单，每执行一个指令程序计数器的值就自动加 1。但若程序中存在条件分支和循环，机器语言的指令就可以将程序计数器的值设定为任意地址（不是 +1）。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，或者跳转到任意地址。接下来，我们就以条件分支为例，来具体说明循环时程序计数器的数值设定机制也是一样的。

图 1-5 表示把内存中存储的数值（示例中是 123）的绝对值输出到显示器的程序的内存状态。程序运行的开始位置是 0100 地址。随着程序计数器数值的增加，当到达 0102 地址时，如果累加寄存器的值是正数，则执行跳转指令（jump 指令）跳转到 0104 地址。此时，由于累加寄存器的值是 123，为正数，因此 0103 地址的指令被跳过，程序的流程直接跳转到了 0104 地址。也就是说，“跳转到 0104 地址”这个指令间接执行了“将程序计数器设定成 0104 地址”这个操作。

图 **1-5**　执行条件分支的程序示例（显示绝对值）

条件分支和循环中使用的跳转指令　，会参照当前执行的运算结果来判断是否跳转。表 1-1 所列出的寄存器中，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负数、零还是正数，标志寄存器　都会将其保存（也负责存放溢出**10** 和奇偶校验**11** 的结果 **)**。

10 溢出（overflow）是指运算的结果超出了寄存器的长度范围。

11 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运算结果自动设定。条件分支在跳转指令前会进行比较运算。至于是否执行跳转指令，则由 CPU 在参考标志寄存器的数值后进行判断。运算结果的正、零、负三种状态由标志寄存器的三个位**12** 表示。图 1-6 是 32 位 CPU（寄存器的长度是 32　位）的标志寄存器的示例。标志寄存器的第一个字节位、第二个字节位和第三个字节位的值为　1　时，表示运算结果分别为正数、零和负数。

12 1 位（bit =binary digit）就是一个位数的二进制数，表示 0 或 1 的数值。32 位 CPU 指的就是用 32 位的二进制数来表示数据及地址的数值。关于二进制数的详细内容，请读者参阅第 2 章。

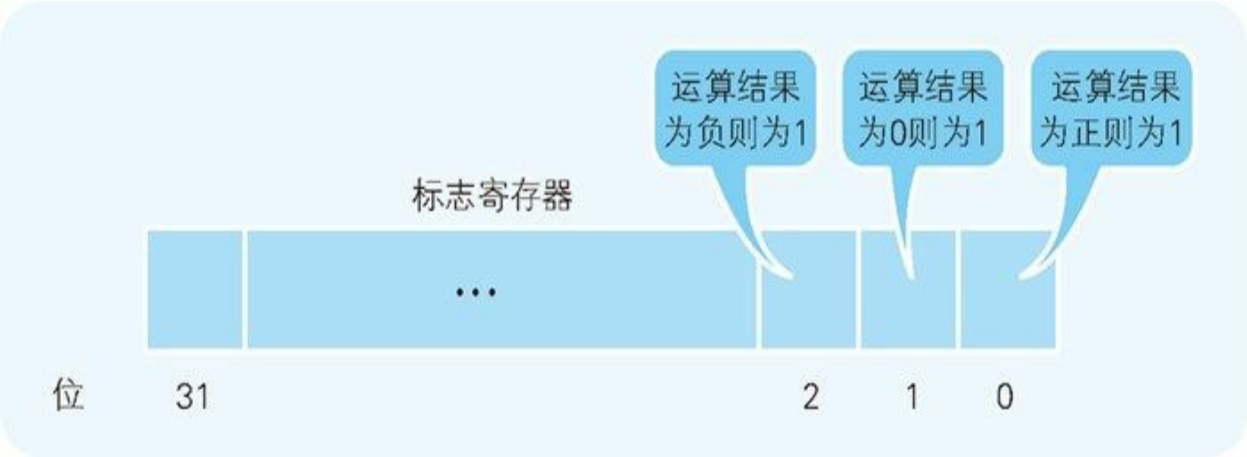

图 **1-6**　比较运算的结果存储在标志寄存器的三个位中

CPU 执行比较的机制很有意思，因此请大家务必牢记。例如，假设要比较累加寄存器中存储的 XXX 值和通用寄存器中存储的 YYY 值，执行