06-指令跳转：原来if...else就是goto

上一讲，我们讲解了一行代码是怎么变成计算机指令的。你平时写的程序中，肯定不只有int a = 1这样最最简单的代码或者指令。我们总是要用到if…else这样的条件判断语句、while和for这样的循环语句，还有函数或者过程调用。

对应的，CPU执行的也不只是一条指令，一般一个程序包含很多条指令。因为有if…else、for这样的条件和循环存在，这些指令也不会一路平铺直叙地执行下去。

今天我们就在上一节的基础上来看看，一个计算机程序是怎么被分解成一条条指令来执行的。

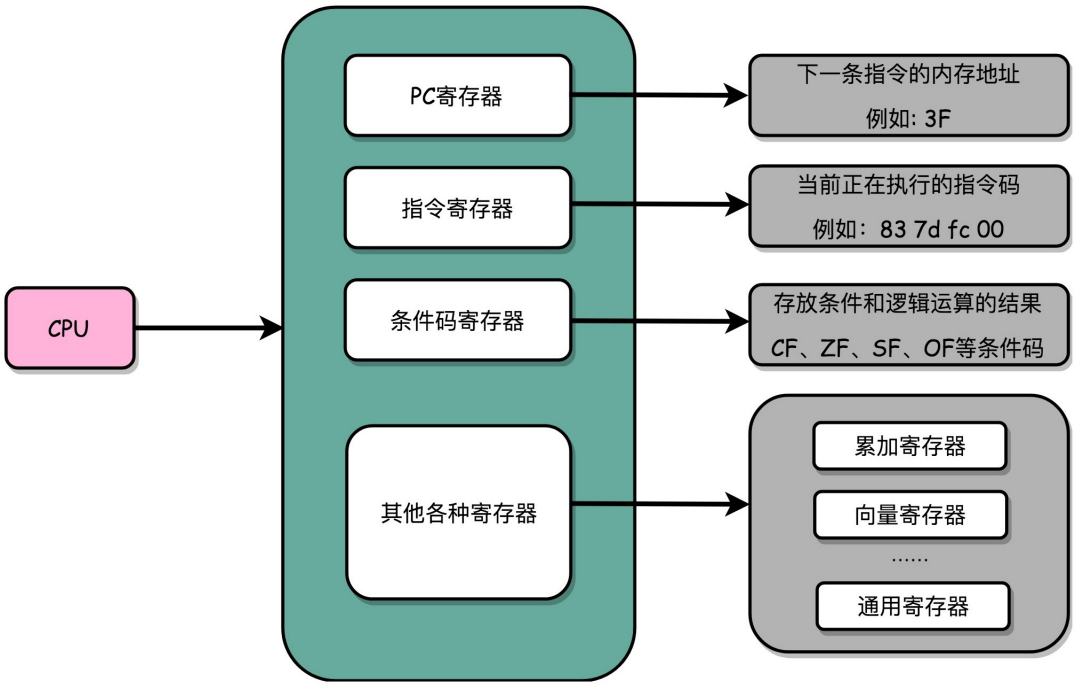
CPU是如何执行指令的？

拿我们用的Intel CPU来说，里面差不多有几百亿个晶体管。实际上，一条条计算机指令执行起来非常复杂。好在CPU在软件层面已经为我们做好了封装。对于我们这些做软件的程序员来说，我们只要知道，写好的代码变成了指令之后，是一条一条**顺序**执行的就可以了。

我们先不管几百亿的晶体管的背后是怎么通过电路运转起来的，逻辑上，我们可以认为，CPU其实就是由一堆寄存器组成的。而寄存器就是CPU内部，由多个触发器（Flip-Flop）或者锁存器（Latches）组成的简单电路。

触发器和锁存器，其实就是两种不同原理的数字电路组成的逻辑门。这块内容并不是我们这节课的重点，所以你只要了解就好。如果想要深入学习的话，你可以学习数字电路的相关课程，这里我们不深入探讨。

好了，现在我们接着前面说。N个触发器或者锁存器，就可以组成一个N位（Bit）的寄存器，能够保存N位的数据。比方说，我们用的64位Intel服务器，寄存器就是64位的。



一个CPU里面会有很多种不同功能的寄存器。我这里给你介绍三种比较特殊的。

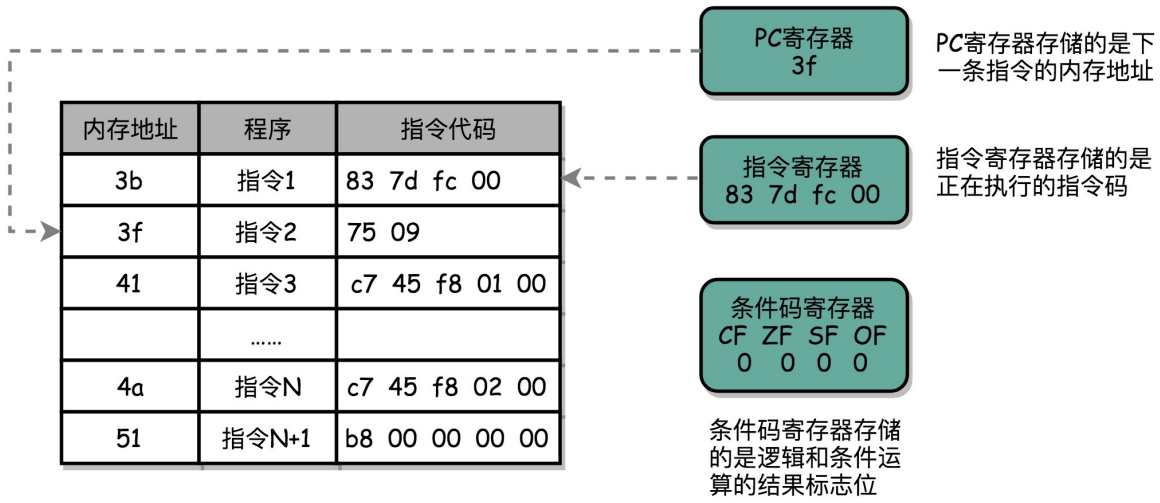
一个是**PC寄存器**（Program Counter Register），我们也叫**指令地址寄存器**（Instruction Address

Register）。顾名思义，它就是用来存放下一条需要执行的计算机指令的内存地址。

第二个是**指令寄存器**（Instruction Register），用来存放当前正在执行的指令。

第三个是**条件码寄存器**（Status Register），用里面的一个一个标记位（Flag），存放CPU进行算术或者逻辑计算的结果。

除了这些特殊的寄存器，CPU里面还有更多用来存储数据和内存地址的寄存器。这样的寄存器通常一类里面不止一个。我们通常根据存放的数据内容来给它们取名字，比如整数寄存器、浮点数寄存器、向量寄存器和地址寄存器等等。有些寄存器既可以存放数据，又能存放地址，我们就叫它通用寄存器。



实际上，一个程序执行的时候，CPU会根据PC寄存器里的地址，从内存里面把需要执行的指令读取到指令寄存器里面执行，然后根据指令长度自增，开始顺序读取下一条指令。可以看到，一个程序的一条条指令，在内存里面是连续保存的，也会一条条顺序加载。

而有些特殊指令，比如上一讲我们讲到J类指令，也就是跳转指令，会修改PC寄存器里面的地址值。这样，下一条要执行的指令就不是从内存里面顺序加载的了。事实上，这些跳转指令的存在，也是我们可以在写程序的时候，使用if…else条件语句和while/for循环语句的原因。

从if…else来看程序的执行和跳转

我们现在就来看一个包含if…else的简单程序。

```
// test.c


#include <time.h>
#include <stdlib.h>


int main()
{
  srand(time(NULL));
  int r = rand() % 2;
```

```
int a = 10;
if (r == 0)
{
  a = 1;
} else {
  a = 2;
}
```

我们用rand生成了一个随机数r，r要么是0，要么是1。当r是0的时候，我们把之前定义的变量a设成1，不然就设成2。

```
$ gcc -g -c test.c
$ objdump -d -M intel -S test.o
```

我们把这个程序编译成汇编代码。你可以忽略前后无关的代码，只关注于这里的if…else条件判断语句。对应的汇编代码是这样的：

```
  if (r == 0)
3b:   83 7d fc 00             cmp    DWORD PTR [rbp-0x4],0x0
3f:   75 09                   jne    4a <main+0x4a>
  {
      a = 1;
41:   c7 45 f8 01 00 00 00    mov    DWORD PTR [rbp-0x8],0x1
48:   eb 07                   jmp    51 <main+0x51>
  }
  else
  {
      a = 2;
4a:   c7 45 f8 02 00 00 00    mov    DWORD PTR [rbp-0x8],0x2
51:   b8 00 00 00 00          mov    eax,0x0
  }
```

可以看到，这里对于r == 0的条件判断，被编译成了cmp和jne这两条指令。

cmp指令比较了前后两个操作数的值，这里的DWORD PTR代表操作的数据类型是32位的整数，而[rbp-0x4]则是一个寄存器的地址。所以，第一个操作数就是从寄存器里拿到的变量r的值。第二个操作数0x0就是我们设定的常量0的16进制表示。cmp指令的比较结果，会存入到**条件码寄存器**当中去。

在这里，如果比较的结果是False，也就是0，就把零标志条件码（对应的条件码是ZF，Zero Flag）设置为1。除了零标志之外，Intel的CPU下还有**进位标志**（CF，Carry Flag）、**符号标志**（SF，Sign Flag）以及**溢出标志**（OF，Overflow Flag），用在不同的判断条件下。

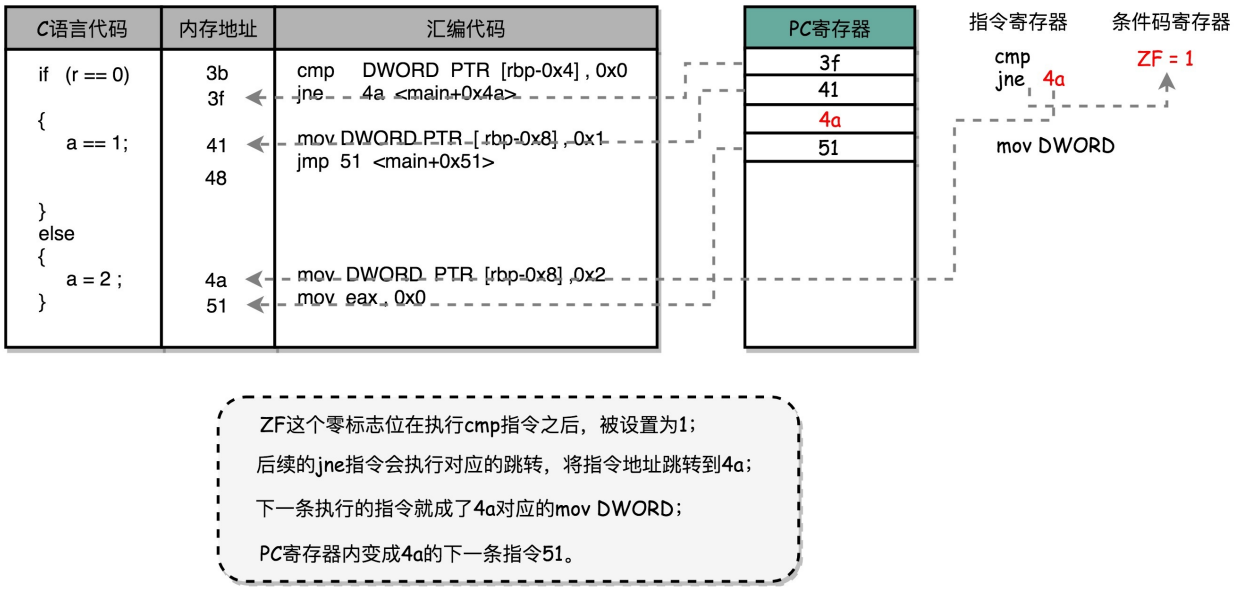
cmp指令执行完成之后，PC寄存器会自动自增，开始执行下一条jne的指令。

跟着的jne指令，是jump if not equal的意思，它会查看对应的零标志位。如果为0，会跳转到后面跟着的操作数4a的位置。这个4a，对应这里汇编代码的行号，也就是上面设置的else条件里的第一条指令。当跳转发

生的时候，PC寄存器就不再是自增变成下一条指令的地址，而是被直接设置成这里的4a这个地址。这个时候，CPU再把4a地址里的指令加载到指令寄存器中来执行。

跳转到执行地址为4a的指令，实际是一条mov指令，第一个操作数和前面的cmp指令一样，是另一个32位整型的寄存器地址，以及对应的2的16进制值0x2。mov指令把2设置到对应的寄存器里去，相当于一个赋值操作。然后，PC寄存器里的值继续自增，执行下一条mov指令。

这条mov指令的第一个操作数eax，代表累加寄存器，第二个操作数0x0则是16进制的0的表示。这条指令其实没有实际的作用，它的作用是一个**占位符。**我们回过头去看前面的if条件，如果满足的话，在赋值的mov指令执行完成之后，有一个jmp的无条件跳转指令。跳转的地址就是这一行的地址51。因为我们的if…else之后整个main函数就结束了，所以if指令执行完成之后没有合适的指令可以跳转，于是编译器自动生成了这样一条废指令，使得if…else条件里的内容结束之后，都能跳转到这同一个位置。



上一讲我们讲打孔卡的时候说到，读取打孔卡的机器会顺序地一段一段地读取指令，然后执行。执行完一条指令，它会自动地顺序读取下一条指令。如果执行的当前指令带有跳转的地址，比如往后跳10个指令，那么机器会自动将卡片带往后移动10个指令的位置，再来执行指令。同样的，机器也能向前移动，去读取之前已经执行过的指令。这也就是我们的while/for循环实现的原理。

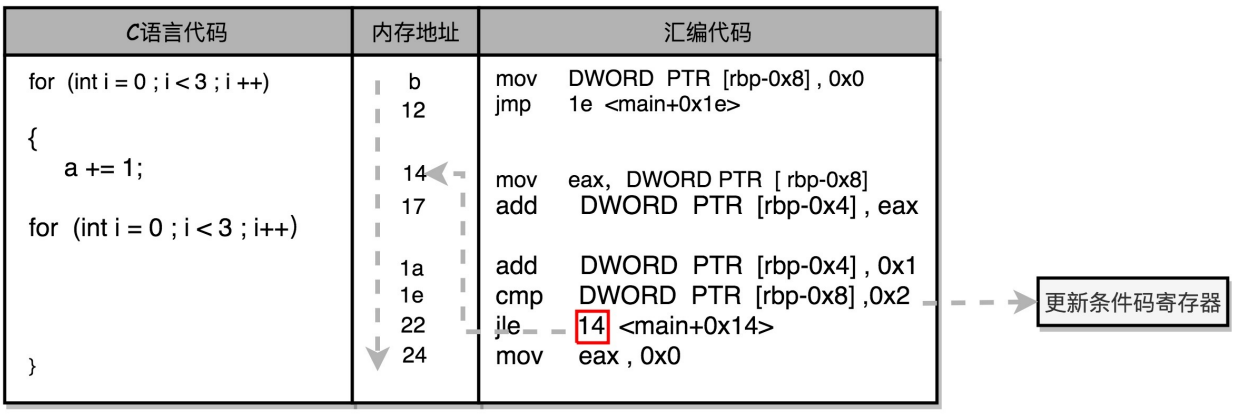
如何通过if…else和goto来实现循环？

```
int main()
{
    int a = 0;
    for (int i = 0; i < 3; i++)
    {
        a += i;
    }
}
```

我们再看一段简单的利用for循环的程序。我们循环自增变量i三次，三次之后，i>=3，就会跳出循环。整个程序，对应的Intel汇编代码就是这样的：

```
  for (int i = 0; i < 3; i++)
 b:   c7 45 f8 00 00 00 00    mov    DWORD PTR [rbp-0x8],0x0
12:   eb 0a                  jmp    1e <main+0x1e>
  {
      a += i;
14:   8b 45 f8               mov    eax,DWORD PTR [rbp-0x8]
17:   01 45 fc               add    DWORD PTR [rbp-0x4],eax
  for (int i = 0; i < 3; i++)
1a:   83 45 f8 01            add    DWORD PTR [rbp-0x8],0x1
1e:   83 7d f8 02            cmp    DWORD PTR [rbp-0x8],0x2
22:   7e f0                  jle    14 <main+0x14>
24:   b8 00 00 00 00         mov    eax,0x0
  }
```

可以看到，对应的循环也是用1e这个地址上的cmp比较指令，和紧接着的jle条件跳转指令来实现的。主要的差别在于，这里的jle跳转的地址，在这条指令之前的地址14，而非if…else编译出来的跳转指令之后。往前跳转使得条件满足的时候，PC寄存器会把指令地址设置到之前执行过的指令位置，重新执行之前执行过的指令，直到条件不满足，顺序往下执行jle之后的指令，整个循环才结束。



如果你看一长条打孔卡的话，就会看到卡片往后移动一段，执行了之后，又反向移动，去重新执行前面的指令。

其实，你有没有觉得，jle和jmp指令，有点像程序语言里面的goto命令，直接指定了一个特定条件下的跳转位置。虽然我们在用高级语言开发程序的时候反对使用goto，但是实际在机器指令层面，无论是if…else…也好，还是for/while也好，都是用和goto相同的跳转到特定指令位置的方式来实现的。

总结延伸

这一节，我们在单条指令的基础上，学习了程序里的多条指令，究竟是怎么样一条一条被执行的。除了简单地通过PC寄存器自增的方式顺序执行外，条件码寄存器会记录下当前执行指令的条件判断状态，然后通过跳转指令读取对应的条件码，修改PC寄存器内的下一条指令的地址，最终实现if…else以及for/while这样的程序控制流程。

你会发现，虽然我们可以用高级语言，可以用不同的语法，比如 if…else 这样的条件分支，或者 while/for 这样的循环方式，来实现不用的程序运行流程，但是回归到计算机可以识别的机器指令级别，其实都只是一个简单的地址跳转而已，也就是一个类似于goto的语句。

想要在硬件层面实现这个goto语句，除了本身需要用来保存下一条指令地址，以及当前正要执行指令的PC寄存器、指令寄存器外，我们只需要再增加一个条件码寄存器，来保留条件判断的状态。这样简简单单的三个寄存器，就可以实现条件判断和循环重复执行代码的功能。

下一节，我们会进一步讲解，如果程序中出现函数或者过程这样可以复用的代码模块，对应的指令是怎么样执行的，会和我们这里的if…else有什么不同。

推荐阅读

《深入理解计算机系统》的第3章，详细讲解了C语言和Intel CPU的汇编语言以及指令的对应关系，以及Intel CPU的各种寄存器和指令集。

Intel指令集相对于之前的MIPS指令集要复杂一些，一方面，所有的指令是变长的，从1个字节到15个字节不等；另一方面，即使是汇编代码，还有很多针对操作数据的长度不同有不同的后缀。我在这里没有详细解释各个指令的含义，如果你对用C/C++做Linux系统层面开发感兴趣，建议你一定好好读一读这一章节。

课后思考

除了if…else的条件语句和for/while的循环之外，大部分编程语言还有switch…case这样的条件跳转语句。switch…case编译出来的汇编代码也是这样使用jne指令进行跳转吗？对应的汇编代码的性能和写很多if…else有什么区别呢？你可以试着写一个简单的C语言程序，编译成汇编代码看一看。

欢迎留言和我分享你的思考和疑惑，你也可以把今天的内容分享给你的朋友，和他一起学习和进步。

精选留言：

- L 2019-05-08 10:11:31
  非计算机专业 表示看到这一章已经很懵逼了 [13赞]

- Out 2019-05-09 01:23:13
  老师您好，在文中您提到：“在这里，如果比较的结果是False，也就是0，就把零标志码设置为1” 这个

地方是不是有问题，根据我查到结果，cmp will ZF to 1 when two operands are equal. 所以如果比较的结果是True，才会把零标志码设置为1。 [3赞]

- Linuxer 2019-05-08 09:53:37

int main()
{
0: 55 push rbp
1: 48 89 e5 mov rbp,rsp
int i = 0;
4: c7 45 fc 00 00 00 00 mov DWORD PTR [rbp-0x4],0x0
int a = 0;
b: c7 45 f8 00 00 00 00 mov DWORD PTR [rbp-0x8],0x0
switch(i)
12: 8b 45 fc mov eax,DWORD PTR [rbp-0x4]
15: 83 f8 01 cmp eax,0x1
18: 74 07 je 21 <main+0x21>
1a: 83 f8 02 cmp eax,0x2
1d: 74 0b je 2a <main+0x2a>
1f: eb 12 jmp 33 <main+0x33>
{
case 1:
a = 1;
21: c7 45 f8 01 00 00 00 mov DWORD PTR [rbp-0x8],0x1
break;
28: eb 11 jmp 3b <main+0x3b>
case 2:
a = 2;
2a: c7 45 f8 02 00 00 00 mov DWORD PTR [rbp-0x8],0x2
break;
31: eb 08 jmp 3b <main+0x3b>
default:
a = 3;
33: c7 45 f8 03 00 00 00 mov DWORD PTR [rbp-0x8],0x3
break;
3a: 90 nop
}

return 1;
3b: b8 01 00 00 00 mov eax,0x1
}
40: 5d pop rbp
41: c3 ret

课后问题验证，这么看如果是单纯的两个分支采用if else更有利，另外 mov eax,0x1从这儿看象是main的返回值 [3赞]

- aiter 2019-05-09 07:42:12

徐老师好 ~
C语言我不会，。，努力看了半天，算是懂了大部分，但是for循环那里还是有点问题 ~ 汇编语言里，jmp 1e 之后，应该是做比较cmp，但是为什么不是0和3比较，而是和16进制的2（0x2）比较？

——————-

因为后面用的jle(jump if less or equal) <=2.如果是使用jl(jump if less) <3.应该是编译器的优化行为？可以自己写汇编代码，使用jl 0x3试试效果是否一样 [2赞]

- 胖胖胖 2019-05-08 15:18:09

  个人理解：这一讲的核心在于理解几个寄存器的作用，从而理解cpu运行程序的过程：cpu从PC寄存器中取地址，找到地址对应的内存位子，取出其中指令送入指令寄存器执行，然后指令自增，重复操作。所以只要程序在内存中是连续存储的，就会顺序执行这也是冯诺依曼体系的理念吧。而实际上跳转指令就是当前指令修改了当前PC寄存器中所保存的下一条指令的地址，从而实现了跳转。当然各个寄存器实际上是由数电中的一个一个门电路组合出来的，而各个门电路的具体电路形式也是属于模电的东西。对于我们来说，有个具体概念就行，实在需要的时候再回去翻翻课本捡起来就行。 [2赞]

- 不记年 2019-05-10 07:28:32

  cpu的在执行指令时还要有个转码的电路来将指令转换成不同的电信号，这些电信号可以控制各个寄存器的动作 ~ [1赞]

- 免费的人 2019-05-09 14:54:34

  switch case 要看编译器有没有生成跳表，没有的话跟if else效率应该是一样的，比如case个数比较少的情况 [1赞]

- 鸟人 2019-05-08 11:01:19

  大家不要把“汇编语言”当成是像C一样的一门统一编程语言。

  请问这节转换的汇编是哪里的汇编？ [1赞]

- Linuxer 2019-05-08 08:28:42

  51: b8 00 00 00 00 mov eax,0x0

  这个会不会是main的返回值呢? [1赞]

- 喜欢吃鱼 2019-05-10 09:45:05

  老师，您上面有个地方是不是笔误了？

  对于r==0的条件判断，如果比较的结果是False,即r!=0,就把零标志条件码（ZF）设置为1。但是您下面说cmp指令执行完后，开始执行下一条指令jne会检查零标志位，如果为0，则跳转到操作数4a的位置。前面不是在r!=0的时候把ZF置为1了吗？所以这里应该是检查到零标志位1才跳转到4a的位置吧，是我理解错了吗？

- 二进制 2019-05-10 09:01:21

  认真学一遍汇编课程，你会觉得这文章很简单。

- Geek 2019-05-08 22:41:48

  徐老师好 ~

  C语言我不会，。，努力看了半天，算是懂了大部分，但是for循环那里还是有点问题 ~ 汇编语言里，jmp 1e 之后，应该是做比较cmp，但是为什么不是0和3比较，而是和16进制的2（0x2）比较？

- 豫樟 2019-05-08 20:08:44

  老师这几节的安排很合理，循序渐进，虽然没学过汇编，但听完之后也能观其大略，拓展知识面，也是清爽的八达鸟!

- 小肚腩era 2019-05-08 15:26:22

今年大四，正在实习。在实际工作慢慢发现自己基础知识的薄弱，所以现在也是抓紧时间在补习这些知识。听老师这一讲，又想起了汇编的知识，比起以前，又有了更深的理解。十分期待老师更新专栏~

- Geek_56d656 2019-05-08 13:32:32
  之前听说java的switch case编译出来的就相当于if else

- Linuxer 2019-05-08 09:44:16
  请问gdb info reg 里面有当前指令寄存器吗？我印象中rip是PC

- 一步 2019-05-08 09:27:08
  编译了一下，switch...case是使用je指令跳转的

- sunyunjian 2019-05-08 09:26:15
  看这些都有些
  吃力我是没救了

- 一步 2019-05-08 09:22:42
  老师，我们常说的二进制执行文件，是指高级语言已经编译成一条条cpu 指令组成的文件吗？

- Sentry 2019-05-08 09:15:47
  switch语句性能要好点，貌似编译器对其进行了优化，编译产生的汇编代码和if else…的应该不一样。