

# 温故知新 | 思考题参考答案（一）

2023-01-02 LMOS 来自北京



天下无鱼

<https://shikey.com/>

[课程介绍 >](#)

《计算机基础实战课》



讲述：小新

时长 01:13 大小 1.11M



你好，我是编辑小新。首先，祝你元旦快乐。

计算机基础的学习并非一蹴而就，希望课程里讲到的内容，像火种一样点燃你的学习探索兴趣。为了辅助你检验每节课的学习效果，我们留下了很多思考题。

今天这节答疑课，就是为了把思考题环节做个“闭环”，我们会公布每节课的参考答案。在对答案之前，还是建议你先自己尝试回答问题，哪怕只是大致整理一下思路，然后再对比看看老师给的思路，查漏补缺。

后面就是前四章，第一节课到第二十二节课的思考题参考答案，希望对你有帮助。

我在结束语里，看到有同学留言说：“学习计算机基础真的很开心，打通的感觉最为舒畅！”看到这样的留言，我和老师都非常开心。也非常欢迎学完一遍课程的同学常回来二刷、三刷，温故知新，有什么新的体会，也欢迎继续在留言区记录分享。

## 🔗 第一节课

Q: 为什么 RISC 的 CPU 能同时执行多条指令?

A: 因为 CPU 内核中有多条指令流水线, 取指、译码、执行、访存、回写, 这些逻辑部件能同时和独立工作。

## 🔗 第二节课

Q: 为什么 RISC-V 要定义特权级?

A: 因为 RISC-V 要支持操作系统和虚拟化, 它们需要管理资源, 需要用相应的特权来保护资源不被其它软件恶意使用。

## 🔗 第三节课

Q: 为什么很多特定算法, 用 Verilog 设计并且硬件化之后, 要比用软件实现的运算速度快很多?

A: 因为 Verilog 设计的电路是并行执行的, 没有受到 CPU 流水线的限制, 所以速度会快很多。

## 🔗 第四节课

Q: 既然用 Verilog 很容易就可以设计出芯片的数字电路, 为什么我们国家还没有完全自主可控的高端 CPU 呢?

A: 这是一个开放性的话题, 这里根据我的理解列举几点:

1. 芯片是一个需要技术积累的行业, 从设计到生产, 每一个环节都有技术壁垒, 发展起来至少需要十到二十年。
2. 我国很多芯片行业起步晚, 在 CPU 方面, 国外早就有像 Intel、AMD 这样的公司形成垄断, 中国很难赶超 Intel。

3. 芯片行业，品牌效应很重要，初创公司做出的芯片可能面临没人敢买的尴尬局面。

4. 芯片是一个很烧钱的且需要长期投入的行业，整个中国在集成电路方面的投入可能还不如国外一个大公司的投入多。

5. 高端芯片是需要一个成熟生态支撑的，需要软件和硬件配套使用，两个需要同步更新、互相促进，才能一直保持领先。

## 🔗 第五节课

Q: 今天我们讲到了 RISC-V 中的分支跳转指令 JAL。想想看，为什么要通过调整立即数的某些位，从 U-TYPE 指令得到 J-TYPE 指令格式呢？这样调整以后有什么好处？

A: JAL 在立即数处编码了一个有符号偏移量，这个偏移量加到 pc 上后，形成跳转目标地址，并将跳转指令后面的指令地址 (pc+4) 加载到 rd，跳转范围为 $\pm 1\text{MB}$ ，这样就可以得到更大的跳转范围了。

## 🔗 第六节课

Q: 为什么要对指令进行预读取？直接取指然后译码、执行不可以吗？

A: 预读取是为了让流水线执行指令更高效，特别是在执行分支跳转指令的时候，预读取提供了简单的分支预测功能，可以在发生跳转之前预测跳转方向，并提前读取后续的指令。

## 🔗 第七节课

Q: 在 6 种指令格式中，S 型、J 型和 B 型指令里的立即数是不连续的，这是为什么？

A: 为了让不同指令格式中尽可能多的字段信息保持位置重合，降低译码难度，同时减少硬件通路上 mux 数量，从而减少硬件逻辑延迟。

## 🔗 第八节课

Q: 在 ALU 模块代码中，为什么要把左移操作转换为右移进行处理？

A: 把左移操作转换为右移操作，可以复用右移操作的电路，节省硬件电路的资源。

## 🔗 第九节课

Q: 除了数据冒险，我们的 CPU 流水线是否还存在其它的冲突问题，你想到解决方法了么？

A: 流水线中除了数据冒险，还可能存在结构冒险和控制冒险，下节课我们将会讲解控制冒险。

## 🔗 第十节课

Q: 除了流水线停顿和分支预测方法，是否还有其他解决控制冒险问题的办法？

A: 控制冒险的第三种解决方法称为延迟转移，也就是延迟转移顺序执行下一条指令，并在该指令后执行分支。这需要用到汇编器对指令进行自动排序，它会在延迟转移指令的后面放一条不受该分支影响的指令，并且指令重新编排了，后面的指令地址会发生变化。

## 🔗 第十一节课

Q: 计算机两大体系结构分别是冯诺依曼体系结构和哈弗体系结构，请问我们的 MiniCPU 属于哪一种体系结构呢？

A: 哈弗结构是一种将程序指令存储和数据存储分开的存储器结构，而冯·诺依曼结构的数据空间和地址空间不分开。显然，我们的 MiniCPU 是把数据空间和地址空间分开的，所以是哈弗结构。

## 🔗 第十二节课

Q: 请你说一说交叉编译的过程？

A: 首先在主环境上用相应的编辑器写好源代码，然后运行主环境上的交叉编译器对源代码进行编译，最后生成目标平台的可执行程序。

## 🔗 第十三节课

Q: 处理环境变量后为什么要执行 `source ./bashrc`，才会生效？

A: `source` 命令和“.”是一样的，所以也可以是 `./bashrc`，`source` 命令与终端 `bashrc` 脚本命令的区别是，`source` 是在当前 `bash` 环境下执行命令，而运行脚本是启动一个子终端进程来执行其中的命令。这样，如果把设置环境变量的命令写进 `bashrc` 脚本文件中，就只会影响子进程，无法改变当前的 `bash` 环境。所以，通过 `bashrc` 脚本文件设置环境变量时，需要 `source` 命令。

#### 第十四节课

Q: 为什么 C 语言中为什么要有流程控制？

A: 因为程序不能一直顺序执行，如果没有分支和循环，这是程序的三大流程结构。也正因此，我们才能实现各种算法，你可以再想想图灵机，就能明白了。

#### 第十五节课

Q: 请问 C 语言函数如何传递结构体类型的参数呢？

A: 如果结构体有多于 8 个成员的情况下，前 8 个成员会被放在寄存器中，剩下部分被存放在栈上，`sp` 指向第一个没有被存放在寄存器上的结构体成员。结构体中如果第  $i$  个成员是整型类型，那么就存放在整型寄存器 `a(i)` 上，如果第  $i$  个成员是浮点数类型，那么就存放在浮点寄存器 `fa(i)` 上 ( $0 \leq i \leq 7$ )。

#### 第十六节课

Q: 请写出机器码 `0x00000033` 对应的指令。

A: `0x00000033` 对应的指令是 `add x0, x0, x0`

#### 第十七节课

Q: 为什么指令编码中，目标寄存器，源寄存器 1，源寄存器 2，占用的位宽都是 5 位呢？

A: 因为 5 位二进制数据，就是 2 的 5 次方，所能表示的编码范围是 0~31，正好索引 RISC-V 的 32 个通用寄存器。

## 🔗 第十八节课

Q: 既然已经有 jal 指令了，为什么还需要 jalr 指令呢？

A: 因为 jal 只能通过立即数传递跳转地址，只能跳转 $\pm 2k$ 的地址空间，如果想要跳转到更远的地址，就得通过寄存器来传递跳转地址。

## 🔗 第十九节课

Q: 我们发现 RISC-V 指令集中没有大于指令和小于等于指令，为什么呢？

A: 因为实现大于指令和小于等于指令的功能，只需要把小于指令和大于等于指令的两个操作数互换一下位置就行了。

## 🔗 第二十节课

Q: 请你尝试用 LR、SC 指令实现自旋锁。

A: 代码如下所示：

📄 复制代码

```
1  /*****/
2  //lrsc.S
3  .text
4  .globl cas
5  #a0内存地址
6  #a1预期值
7  #a2所需值
8  #a0返回值，如果成功，则为0! 否则为1
9  cas:
10     lr.w t0, (a0)      #加载以前的值
11     bne t0, a1, fail    #不相等则跳转到fail
12     sc.w a0, a2, (a0)   #尝试更新
13     jr ra               #返回
14 fail:
15     li a0, 1            #a0 = 1
16     jr ra               #返回
```

```

17  /*****
18  //lock.c
19  //定义锁类型
20  typedef struct Lock
21  {
22      int LockVal;    //锁值
23  }Lock;
24  //自旋锁初始化
25  void SpinLockInit(Lock* lock)
26  {
27      //锁值初始化为0
28      lock->LockVal = 0;
29      return;
30  }
31  //自旋锁加锁
32  void SpinLock(Lock* lock)
33  {
34      int status;
35      do
36      {
37          status = cas(&lock->LockVal, 0, 1); //加锁
38      }while(status); //循环加锁，直到成功
39      return;
40  }
41  //自旋锁解锁
42  void SpinUnLock(Lock* lock)
43  {
44      SpinLockInit(lock); //直接初始化 解锁
45      return;

```

## 🔗第二十一节课

Q: 为什么加载字节与加载半字指令，需要处理数据符号问题呢，而加载字指令却不需要？

A: 首先，加载指令是从内存到寄存器。

其次加载到寄存器中的数据会参与运算，数据的运算就需要考虑数据的符号问题。


最后加载字指令是加载 32 位数据占用整个寄存器，不需要处理符号位问题，只需要原样加载内存中的数据就行了，内存中的数据有符号就有符号，没有符号那就是没有符号。

## 🔗第二十二节课

Q: 为什么三条储存指令，不需要处理数据符号问题呢？

**A:** 首先储存指令是把寄存器中的数据储存到内存，其次储存到内存中的数据不参与运算时，不需要考虑符号问题。只需要原样保存在内存中就行了。

分享给需要的人，Ta购买本课程，你将得 **20** 元

 生成海报并分享

 赞 1     提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

[上一篇](#)    期末测试 | 来赴一场100分之约！

[下一篇](#)    温故知新 | 思考题参考答案（二）

## 精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。