1. 这堂课的内容对你将来写程序有什么启示？为了写出更快的程序，你会考虑哪些内容?

   写程序需要考虑计算机底层是如何实现的，并行流水线，多核处理器，局部性原理，加速经常发生的事件，要用到底层中自己能够用到的东西来提高程序的执行效率。

   以 c 语言为例为了优化程序，我会考虑以下措施：

   ①代码移动：这类优化包括识别要执行多次（例如在循环里）但是计算结果不会改变的计算。因而可以将计算移动到代码前面不会被多次求值的部分。

```
1   /* Implementation with maximum use of data abstraction */
2   void combine1(vec_ptr v, data_t *dest)
3   {
4       long i;
5
6       *dest = IDENT;
7       for (i = 0; i < vec_length(v); i++) {
8           data_t val;
9           get_vec_element(v, i, &val);
10          *dest = *dest OP val;
11      }
12  }
```

   combine1 函数是对数组 v 中的元素进行累计操作运算（加，乘等等），将结果赋给 dest 指针对应的内存单元

```
1   /* Move call to vec_length out of loop */
2   void combine2(vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(v);
6
7       *dest = IDENT;
8       for (i = 0; i < length; i++) {
9           data_t val;
10          get_vec_element(v, i, &val);
11          *dest = *dest OP val;
12      }
13  }
```

combine2 在 combine1 的基础上做了改进:它在开始时调用 vec_length(v), 并将结果赋值给局部变量 length。可以看到 combine1 的复杂度是 O(n^2), 而 combine2 的复杂度是 O(n), 程序效率有明显提升。我们将对 vec_length 的调用从循环内部移动到循环的前面。编译器会试着自动进行代码移动。但不幸的是，对于怎么进行代码移动，编译器通常会非常小心。它们不能可靠地发现一个函数是否会有副作用，因而假设函数会有副作用。例如，如果 vec_length 有某种副作用，每一次调用 vec_length 都会使得一个全局 int 类型的变量 +1，那么 combine1 和 combine2 就会有完全不同的结果。为了改进代码，程序员必须经常帮助编译器显式地完成代码的移动。

②减少内存调用

```
1   /* Direct access to vector data */
2   void combine3(vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(v);
6       data_t *data = get_vec_start(v);
7
8       *dest = IDENT;
9       for (i = 0; i < length; i++) {
10          *dest = *dest OP data[i];
11      }
12  }
```

为了程序更加直观，我们把 combine2 改成了 combine3

```
1   /* Accumulate result in local variable */
2   void combine4(vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(v);
6       data_t *data = get_vec_start(v);
7       data_t acc = IDENT;
8
9       for (i = 0; i < length; i++) {
10          acc = acc OP data[i];
11      }
12      *dest = acc;
13  }
```

对于 combine3，指针 data 的值存放在寄存器之中，每次执行 for 循环，都需要从寄存器中取出 data 的值，然后去内存中找数据。对于

combine4, 局部变量 acc 的值存放在寄存器之中，每次执行 for 循环，只需要从寄存器中取出 acc 的值即可让它与 data[i] 进行 op 操作，acc 的取值不需要调用内存，效率明显提升。

③循环展开，提高程序并行性

```
1   /* 2 x 1 loop unrolling */
2   void combine5(vec_ptr v, data_t *dest)
3   {
4       long i;
5       long length = vec_length(v);
6       long limit = length-1;
7       data_t *data = get_vec_start(v);
8       data_t acc = IDENT;
9
10      /* Combine 2 elements at a time */
11      for (i = 0; i < limit; i+=2) {
12          acc = (acc OP data[i]) OP data[i+1];
13      }
14
15      /* Finish any remaining elements */
16      for (; i < length; i++) {
17          acc = acc OP data[i];
18      }
19      *dest = acc;
20  }
```

循环展开能够从两个方面改进程序的性能。首先，它减少了不直接有助于程序结果的操作的数量，例如循环索引计算和条件分支。第二，它提供了一些方法，可以进一步变化代码，减少整个计算中关键路径上的操作数量。combine5 相比于 combine4 在执行的过程中，在 acc 和 data[i] 进行 op 操作的同时，计算机可以去取出 data[i+1] 对应的值，提高了效率。

2. 什么是 ISA? 为什么 ISA 与硬件功能密切相关?

- ISA 即 Instruction Set Architecture（指令集体系结构），它定义了计算机体系结构中的指令集合、寄存器、数据类型和内存访问方法等。ISA 是软件与硬件之间的接口，它规定了软件可以使用的指令和操作，以及这些指令和操作如何与硬件交互。计算机系统工作的基本过程是：程序员编写的软件经编译器翻译成可执行程序，也就是一个机器指令的序列，然后由底层硬件一条条读取这些指令来执行。
- 指令集体系结构定义了基本数据类型、寄存器、指令、寻址模式、异常或者中断的处理方式等。一台计算机的指令系统反映了该计算机的全部功能，机器类型不同，其指令系统也不同，因而功能也不同。ISA 作为处理器的基础，对于处理器的整体性能起到了决定性的作用，不同架构的处理器同主频下，性能差距可以达到 2-5 倍。ISA 的实现需要通过编写与指令集中的指令对应的硬件实现的逻辑代码来完成。CPU 依靠指令来计算和控制计算机系统，每款 CPU 在设计时就规定了一系列与其硬件电路相配合的指令系统。指令系统的设置和机器的硬件结构密切相关，一台计算机要有较好的性能，必须设计功能齐全、通用性强、内含丰富的指令系统，这需要复杂的硬件结构来支持，硬件需要实现这些功能以使得处理器能够正确执行软件指令。

3. 请查阅资料，对比 CISC 和 RISC 的区别

RISC 全称 Reduced Instruction Set Computer，精简指令集计算机。CISC 全称 Complex Instruction Set Computer，复杂指令集计算机。

二者的区别在于：

- CISC 的指令能力强，指令多，大多数指令使用率低同时也增加了 CPU 的复杂度，电路随着指令的增多变得复杂，指令的时钟

周期很难对齐，流水线难管理，指令是可变长格式；RISC 的指令大部分为单周期指令，只保留最核心的指令，全部资源优化保留的指令速度，指令长度固定。

- CISC 架构中通常包含大量的寄存器，并且某些指令可以直接操作内存，支持多种寻址方式，寄存器的使用较为灵活。RISC 架构中寄存器的数量通常较少，指令只能通过寄存器进行操作，而不能直接访问内存，只有 Load/Store 操作能够访问内存，这样可以简化处理器的设计。
- RISC 比 CISC 更便于设计，可降低成本，提高可靠性，每条指令的功能相对简单，通常只执行一项基本操作，指令的执行速度更快，还能优化编译，有效支持高级语言程序，更符合加速经常发生的事件这一规则
- CICS 的指令系统比较丰富，有专用指令来完成特定的功能，因此处理特殊任务效率高。CICS 指令通常具有不同长度和复杂度，处理器需要使用微程序或复杂的控制逻辑来解码和执行这些指令，因此处理器的设计较为复杂。