家庭作业

•4.45　在 3.4.2 节中，x86-64 pushq 指令被描述成要减少栈指针，然后将寄存器存储在栈指针的位置。因此，如果我们有一条指令形如对于某个寄存器 *REG*，pushq *REG*，它等价于下面的代码序列：

```
subq $8,%rsp          Decrement stack pointer
movq REG, (%rsp)      Store REG on stack
```

A. 借助于练习题 4.7 中所做的分析，这段代码序列正确地描述了指令 pushq %rsp 的行为吗？请解释。

B. 你该如何改写这段代码序列，使得它能够像对 REG 是其他寄存器时一样，正确地描述 REG 是%rsp 的情况？

•4.46　在 3.4.2 节中，x86-64 popq 指令被描述为将来自栈顶的结果复制到目的寄存器，然后将栈指针减少。因此，如果我们有一条指令形如 popq *REG*，它等价于下面的代码序列：

```
movq (%rsp), REG      Read REG from stack
addq $8,%rsp          Increment stack pointer
```

A. 借助于练习题 4.8 中所做的分析，这段代码序列正确地描述了指令 popq %rsp 的行为吗？请解释。

B. 你该如何改写这段代码序列，使得它能够像对 *REG* 是其他寄存器时一样，正确地描述 REG 是%rsp 的情况？

**4.47　你的作业是写一个执行冒泡排序的 Y86-64 程序。下面这个 C 函数用数组引用实现冒泡排序，供你参考：

```
1   /* Bubble sort: Array version */
2   void bubble_a(long *data, long count) {
3       long i, last;
4       for (last = count-1; last > 0; last--) {
5           for (i = 0; i < last; i++)
6               if (data[i+1] < data[i]) {
7                   /* Swap adjacent elements */
8                   long t = data[i+1];
9                   data[i+1] = data[i];
10                  data[i] = t;
11              }
12      }
13  }
```

A. 书写并测试一个 C 版本，它用指针引用数组元素，而不是用数组索引。

B. 书写并测试一个由这个函数和测试代码组成的 Y86-64 程序。你会发现模仿编译你的 C 代码产生的 x86-64 代码来做实现会很有帮助。虽然指针比较通常是用无符号算术运算来实现的，但是在这个练习中，你可以使用有符号算术运算。

**4.48　修改对家庭作业 4.47 所写的代码，实现冒泡排序函数的测试和交换(6～11 行)，要求不使用跳转，且最多使用 3 次条件传送。

**4.49　修改对家庭作业 4.47 所写的代码，实现冒泡排序函数的测试和交换(6～11 行)，要求不使用跳转，且只使用 1 次条件传送。

**4.50　在 3.6.8 节中，我们看到实现 switch 的一种常见方法是创建一组代码块，再用跳转表对这些块进行索引。考虑图 4-69 中给出的函数 switchv 的 C 代码，以及相应的测试代码。

用跳转表以 Y86-64 实现 switchv。虽然 Y86-64 指令集不包含间接跳转指令，但是，你可以通过把计算好的地址入栈，再执行 ret 指令来获得同样的效果。实现类似于 C 语言所示的测试代码，证明你的 switchv 实现可以处理触发 default 的情况以及两个显式处理的情况。