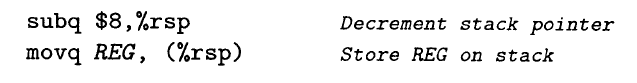
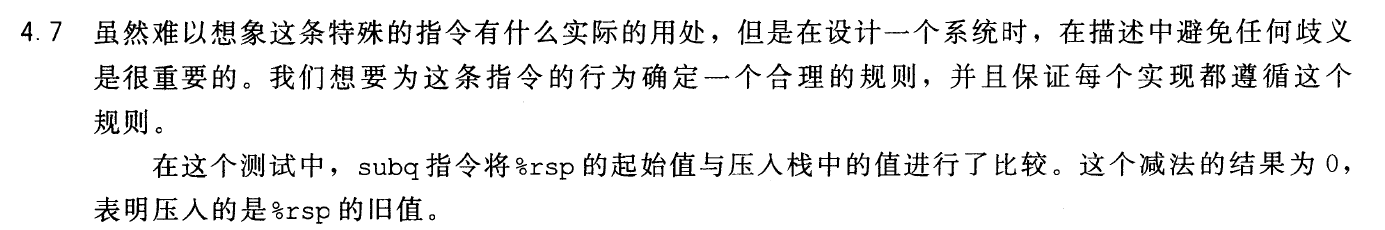
# 4.45



A**错，pushq %rsp的实际效果是将rsp变动前的原值存入栈，而非减小8以后的值，如果按照题目所给的操作，栈中保存的将是减小8以后的rsp。这里参考练习4.7的书本解释。**



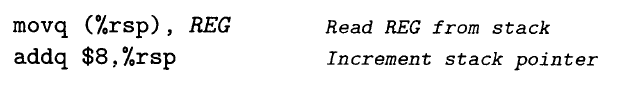
B **修改为如下**

**movq REG, -8(%rsp)**

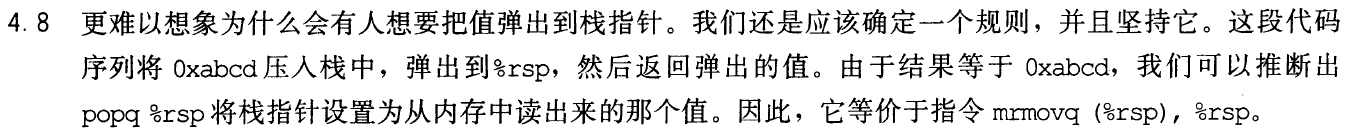
**subq $8, %rsp**

**目的就是在rsp变化前对其进行保存。**

# 4.46



A**错，popq %rsp的实际效果是将栈中的原值赋给rsp，而非增加8以后的值，如果按照题目所给的操作， rsp将变为栈中保存的数据加8。这里参考练习4.8的书本解释。**



B **修改为如下**

**addq $8, %rsp**

**movq -8(%rsp), REG**

**目的就是把对rsp的移动放在对其赋值之前，这样如果REG表示的是rsp，rsp能变成栈中原值，如REG不是rsp，也能正常运行。**

# 4.47

A**代码如下**

**void bubble\_p(long\* data, long count)**

**{**

**long \*i = data, \*last;**

**for (last = data+count-1; last > data; last--)**

**{**

**for (i = data; i < last; i++)**

**{**

**if (\*(i+1) < \*i)**

**{**

**long t = \*(i+1);**

**\*(i+1) = \*i;**

**\*i = t;**

**}**

**}**

**}**

**}**

**测试函数**

**int main()**

**{**

**long data[5]= {12345,1234,123,12,1};**

**bubble\_p(data,5);**

**for (int i=0; i<5; i++)**

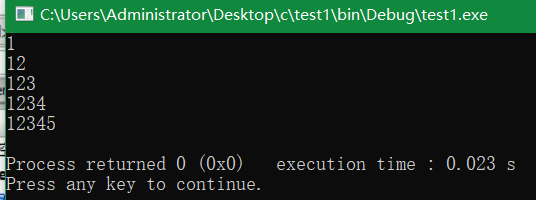
**{**

**printf("%d\n",data[i]);**

**}**

**}**

**运行结果：**

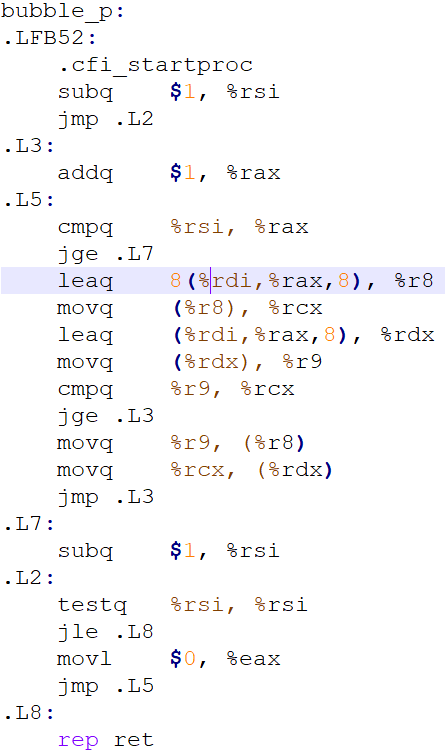


**编程思想：数组名的本质就是指向数组首地址的指针，与索引值0无异，所以，把原版本所有的数组索引值0改为data，比如last>0改为last>data，i=0改为i=data。把所有data[索引值]的形式换成\*(i+索引值)就行了。**

B



用gcc将.c文件编译成.s文件后打开，然后将x86-64汇编翻译成要求的 Y86-64

**long ys\_bubble\_p(long \*data, long \*end)**

**data in %rdi, end in %rsi**

**irmovq $1, %r8**

**subq %r8, %rsi**

**jmp L2**

**L3:**

**irmovq $1, %r8**

**addq %r8, %rax**

**L5:**

**rrmovq %rsi, %r8**

**subq %rax, %r8**

**jb .L7 //因为Y86没有jge所以把比较量对调，使用jb**

**rrmovq % rax, % rdx**

**addq % rdx, % rdx**

**addq % rdx, % rdx**

**addq % rdx, % rdx //rdx=8\*rax**

**rrmovq % rdx, % r8**

**irmovq $8, % r9**

**addq % r9, % r8**

**addq % rdi, % r8 //实现leaq 8(% rdi,% rax,8), % r8**

**mrmovq (% r8), % rcx //data[i+1]**

**addq % rsi, % rdx //实现leaq (% rdi,% rax,8), % rdx**

**mrmovq (% rdx), % r9 //data[i]**

**rrmovq %r9, %r10**

**subq %rcx, %r10**

**jb L3 //因为Y86没有jge所以把比较量对调，使用jb**

**rmmovq %r9, (% r8) //满足条件则交换后跳转，不满足直接跳转**

**rmmovq %rcx, (% rdx)**

**jmp L3**

**L7:**

**subq $1, %rsi**

**L2:**

**subq %rsi, $0**

**je L8**

**irmovq $0,%rax**

**jmp L5**

**L8:**

**ret**

**然后补上一个main函数调用它**

**Main：**

**irmovq $12345, %rsi**

**rmmovq %rsi , (rsp)**

**irmovq $1234, %rsi**

**rmmovq %rsi , (rsp)**

**irmovq $123, %rsi**

**rmmovq %rsi , (rsp)**

**irmovq $12, %rsi**

**rmmovq %rsi , (rsp)**

**irmovq $1, %rsi**

**rmmovq %rsi , (rsp)**

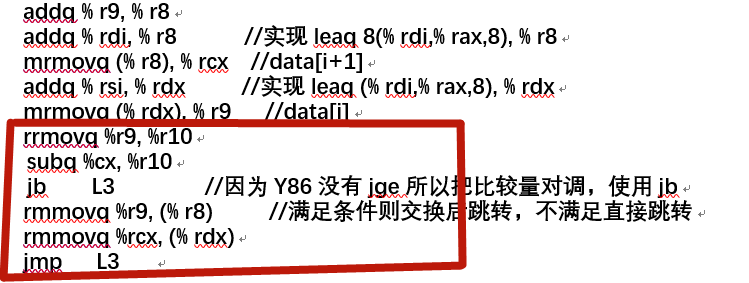
**movq %rsp, %rdi**

**movl $5, %rsi**

**call bubble\_p**

# 4.48

**题目代码的6-11行只涉及判断元素大小和交换环节于是对应下图的方框部分**

****

**rrmovq %rcx, %r10 //data[i+1]**

**subq %r9, %r10 //data[i+1]-data[i]**

**cmovl %r9, %r11**

**cmovl %rcx, %r9**

**cmovl %r11, %rcx**

**rmmovq %rcx, (% r8)**

**rmmovq %r9, (% rdx)**

# 4.49

同4.48题一样这里还是只需要修改方框部分。

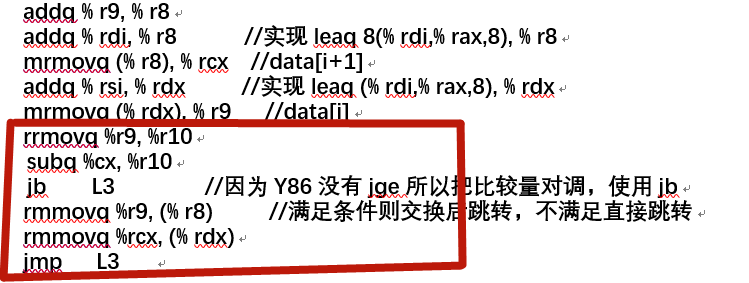
原理：三次异或交换两个变量。举个例子下面三次异或之后ab值互换

a = a^b; //把ab不同的位置为1

b = a^b; //把b中与a不同的位取反，则b=原a

a = a^b; //把a中与b不同的位取反，则a=原b

但是如果在第一步执行完成以后把a的原值付给b，交换将不会发生。

**rrmovq %rcx, %r11**

**rrmovq %r9, %r12**

**xorq %rcx, %r9**

**subq %r12, %r11 // data[i+1]-data[i]**

**cmovge %r11, %r9 //如果大于，则交换无需发生**

**xorq %r9, %rcx**

**xorq %rcx, %r9**

**rmmovq %rcx, (% r8)**

**rmmovq %r9, (% rdx)**

# 4.50

**# Array of 8 elements**

**//定义八个元素的数组来存放结果**

**.align 8**

**vals:**

**.quad 0x000000000000**

**.quad 0x000000000000**

**.quad 0x000000000000**

**.quad 0x000000000000**

**.quad 0x000000000000**

**.quad 0x000000000000**

**.quad 0x000000000000**

**.quad 0x000000000000**

**//定义跳转表**

**jump\_table:**

**.quad L1**

**.quad L4**

**.quad L2**

**.quad L3**

**.quad L4**

**.quad L2**

**main:**

**irmovq stack, %rsp //设置栈指针**

**irmovq vals, %r12 //把结果数组的首地址赋给%r12**

**irmovq $-1,%rdi //Y以下顺序以参数-1到7执行switch函数的，并将结果写入val数组**

**call switchv**

**rmmovq %rax, (%r12)**

**irmovq $0,%rdi**

**call switchv**

**rmmovq %rax, 0x8(%r12)**

**irmovq $1,%rdi**

**call switchv**

**rmmovq %rax, 0x10(%r12)**

**irmovq $2,%rdi**

**call switchv**

**rmmovq %rax, 0x18(%r12)**

**irmovq $3,%rdi**

**call switchv**

**rmmovq %rax, 0x20(%r12)**

**irmovq $4,%rdi**

**call switchv**

**rmmovq %rax, 0x28(%r12)**

**irmovq $5,%rdi**

**call switchv**

**rmmovq %rax, 0x30(%r12)**

**irmovq $6,%rdi**

**call switchv**

**rmmovq %rax, 0x38(%r12)**

**ret**

**# long switchv(long idx)**

**# idx in %rdi**

**switchv:**

**rrmovq %rdi, %r8**

**irmovq $5, %r9**

**subq %r9, %r8**

**jg L4 //如果参数大于5，直接转default**

**andq %rdi, %rdi //参数小于0，也直接转default**

**jl L4**

**irmovq jump\_table, %r8 //把跳转表的首地址给r8**

**irmovq $8, %r9**

**irmovq $1, %r10**

**loop:**

**subq %r10, %rdi**

**jl endloop**

**addq %r9, %r8**

**jmp loop**

**endloop:·· //运行完这个循环之后%r8的值就是jumptable首地址+rdi\*8-8**

**mrmovq (%r8), %r8 //这样我们就完成了根据rdi大小往跳转表记录的地址跳转的操作。**

**pushq %r8 //也因为Y86-64没有相对寻址，所以我们把跳转表中存的地址入栈之后再ret**

**ret**

**L1: //以下是具体的switch操作**

**irmovq 0xaaa, %rax**

**ret**

**L2:**

**irmovq 0xbbb, %rax**

**ret**

**L3:**

**irmovq 0xccc, %rax**

**ret**

**L4:**

**irmovq 0xddd, %rax #default**

**ret**

# 4.51

|  |  |
| --- | --- |
| **阶段** | **iaddq V,rB** |
| **取指** | **icode:ifun = M1[PC]; rA:rB = M1[PC+1]; valC = M8[PC+2]; valP = PC + 10;** |
| **译码** | **valB = R[rB]** |
| **执行** | **valE = valB + valC; set CC** |
| **访存** |  |
| **写回** | **R[rB] = valE** |
| **更新PC** | **PC = valP** |