第四章 处理器的体系结构-家庭作业

——1170300825 李大鑫

4.45

A:

没有正确执行指令 pushq 的行为。当 REG 是%rsp 的时候，pushq %rsp 的行为原本是将%rsp 的值压入栈中，而题目中所给代码的行为是 subq $8,%rsp 将%rsp-8，movq REG,(%rsp)向(%rsp-8)中存储%rsp-8，其实就是向栈中压入了%rsp-8，所以没有正确执行。

B:

```
movq REG,%eax
subq $8,%rsp
movq %eax,(%rsp)
```

4.46

A:

没有正确执行 popq 的行为。当 REG 是%rsp 的时候，popq %rsp 的行为原本是将栈顶指针向下移动，将原来栈顶的值覆盖到现在的栈顶，而题目中所给代码的行为是 movq (%rsp),%rsp，将%rsp 中的内容存放到%rsp 寄存器中，然后 add $8,%rsp，将栈顶指针向下移动，所以此时栈顶存放的是原来栈顶的第二个值而不是原来的栈顶值。

B:

```
movq (%rsp),%rax
addq $8,%rsp
movq %rax,REG
```

4.47

A

指针版本代码:

```
void bubble_a(int *list, int n){
    int i,j;
    for(j = 1; j < n; j++){
        for(i = j - 1; i >= 0; i--)
            if(*(list + i + 1) < *(list + i)){
                int t = *(list + i + 1);
```

```
                    *(list + i + 1) = *(list + i);
                    *(list + i) = t;
                }
            }
        }
```

代码测试：

```
int main() {
    int list[10] = {10,2,3,1,7,6,8,5,9,4};
    int i;
    bubble_a(list,10);
    for(i=0;i<10;i++) {
        printf("%d ",list[i]);
    }
    puts("");
    return 0;
}
```

```
E:\CodeTraining\ProgramingLanguageTest\C\csapp_chapter5\cmake-build-debug\csapp_chapter5.exe
1 2 3 4 5 6 7 8 9 10
```

B

```
    bubble_b:
.LFB22:
    .cfi_startproc
    pushl   %edi
    .cfi_def_cfa_offset 8
    .cfi_offset 7, -8
    pushl   %esi
    .cfi_def_cfa_offset 12
    .cfi_offset 6, -12
    pushl  %ebx
    .cfi_def_cfa_offset 16
    .cfi_offset 3, -16
    mrmovl   16(%esp), %edx
    mrmovl   20(%esp), %edi
    irmovl   $1, %eax
    subl    %eax, %edi
    jle     .L1
    subl    $1, %edi
    irmovl   $0, %esi
.L6:
    rrmovl  %esi, %eax
    irmovl   $0 , ebx
```

```
    subl    %ebx, %esi
    jl      .L3
.L7:
    rrmovl   %eax, %ecx
    addl    %ecx, %ecx
    addl    %ecx, %ecx
    addl    %edx, %ecx
    mrmovl   4(%ecx), %ecx
    rrmovl   %eax, %ebx
    addl    %ecx, %ebx
    addl    %ecx, %ebx
    addl    %edx, %ebx
    mrmovl   (%ebx), %ebx
    subl    %ebx, %ecx
    jge     .L4
    addl    %eax, %eax
    addl    %eax, %eax
    addl    %edx, %eax
    rmmovl   %ebx, 4(%eax)
    addl    %eax, %eax
    addl    %eax, %eax
    addl    %edx, %eax
    rmmovl   %ecx, 4(%eax)
.L4:
    subl   $1, %eax
    irmovl $-1, %edx
    subl   %edx, %eax
    jne .L7
.L3:
    addl   $1, %esi
    subl   %edi, %esi
    jne .L6a
.L1:
    popl   %ebx
    .cfi_def_cfa_offset 12
    .cfi_restore 3
    popl   %esi
    .cfi_def_cfa_offset 8
    .cfi_restore 6
    popl   %edi

    .cfi_def_cfa_offset 4
    .cfi_restore 7
    ret
```

```
    .cfi_endproc
.LFE22:
    .size  bubble_b, .-bubble_b
    .section   .rodata.str1.1,"aMS",@progbits,1
```

4.48

因为题目中的要求是不允许使用跳转，而 if 判断在汇编层面的对应都是通过跳转完成的，如果要用到条件传送，我们想到 c 语言中的三目操作符，它在汇编层面对应的就应该是条件传送的使用。如果要实验比较 A 和 B 的大小然后根据比较结果进行交换的效果，我们可以先存储 AB 的值，然后通过使用两次条件传送来完成，两次条件传送的条件比较都是一样的，都是比较 AB 的大小。

A

应该对应的c代码:

```
void bubble_c(int *list,int count)
{
    int i , next;
    int pre_ele,next_ele;
    for(next = 1;next < count;next++)
    {
        for(i = next -1;i >= 0;i--)
        {
            pre_ele = *(list + i);
            next_ele = *(list + i + 1);
            *(list + i) = next_ele < pre_ele ? next_ele : pre_ele;
            *(list + i + 1) = next_ele < pre_ele ? pre_ele : next_ele;
        }
    }
}
```

B

6-11 行所代表的循环内部 Y86 汇编代码:

```
.L6:
    mrmovl  (%ebx,%eax,4), %edx     #pre_ele = *(list+i)
    mrmovl  4(%ebx,%eax,4), %ecx    #next_ele = *(list+i+1)
    cmpl    %edx, %ecx              #比较 next_ele 和 pre_ele
    rrmovl  %edx, %ebp              #%ebp = pre_ele (对应next_ele>=pre_ele的默
认情况)
```

```
    cmovl   %ecx, %ebp              #%ebp = next_ele(对应next_ele < pre_ele的情
况)
    rmmovl  %ebp, (%ebx,%eax,4)     #*(list+i) = %ebp   ( *(list + i) =
next_ele < pre_ele ? next_ele : pre_ele; )
    cmovge  %ecx, %edx              # %edx=next_ele (对应next_ele >= pre_ele的
情况 否则%edx=pre_ele)
    rmmovl  %edx, 4(%ebx,%eax,4)    #*(list+i+1) = %edx (*(list + i + 1) =
next_ele < pre_ele ? pre_ele : next_ele;)
    subl    $1, %eax                #循环变量i-1
    cmpl    $-1, %eax               #判断循环终止
    jne .L6
```

4.49

上一题目中使用了两个条件跳转，我们发现上题之所以使用了两个条件跳转是因为进行了两个三目操作（其实就对应着汇编层面的条件跳转），然后汇编代码中，两个 cmov 都是利用这个条件码进行条件传送。类似于不适用第三变量交换 ab 的值，我们可以通过利用一定的运算来只是用一个条件传送完成交换的工作。

设 AB 为指针，两种不同情况下*A 和*B 的赋值情况如下：

1）next_ele >= pre_ele

*A = [A]

*B = [B] – (-[A]+[A]) = [B]-(-[A]+*A)

2）next_ele < pre_ele

*A = [B]

*B = [B]-(-[A]+[B]) = [B]-(-[A]+*A)

通过上面，我们可以发现，两种不同情况下，不同的只有*A 的值，我们可以通过相同的计算方法根据*A 的值计算出最终*B 的值。所以只需要使用条件跳转语句将*A 进行赋值即可。

代码如下：

A

```
void bubble_d(int *list,int count)
{
    int i , next;
    int pre_ele,next_ele,tmpa,tmpb;
    for(next = 1;next < count;next++)
    {
        for(i = next -1;i >= 0;i--)
        {
            pre_ele = *(list + i);
```

```
            next_ele = *(list + i + 1);
            tmpb = -pre_ele;
            tmpa = next_ele < pre_ele ? next_ele:pre_ele;
            tmpb = next_ele-(tmpa+tmpb);
            *(list + i) = tmpa;
            *(list + i + 1) = tmpb;
        }
    }
}
```

B

```
.L6:
    mrmovl (%ebx,%eax,4), %edx      #pre_ele = *(list+i)
    mrmovl 4(%ebx,%eax,4), %ecx #next_ele = *(list+i+1)
    irmovl 0x0,%esi
    subl   %edx,%esi            #tmpb = -pre_ele;
    cmpl   %edx, %ecx           #比较 next_ele 和 pre_ele
    rrmovl %edx,%edi            #tmpa = pre_ele (对应next_ele>=pre_ele
的默认情况)
    cmovl  %ecx,%edi            #tmpa = next_ele (next_ele <
pre_ele ? next_ele:pre_ele)
    leal   (%edi,%esi),%eax     #%eax = tmpa+tmpb
    rrmovl  %ecx,%esi           #%esi = next_ele
    subl    %eax,%esi           #tempb = next_ele-(tmpa+tempb)
    rmmovl %edi,(%ebx,%eax,4)       #*(list + i) = tmpa;
    rmmovl %esi,4(%ebx,%eax,4)      #*(list + i + 1) = tmpb;
    subl   $1, %eax             #循环变量i-1
    cmpl   $-1, %eax            #判断循环终止
    jne .L6
```

4.50

```
# Execution begins at address 0
    .pos 0
    irmovq stack, %rsp      # Set up stack pointer
    call main             # Execute main program
    halt              # Terminate program

# Array of 8 elements
    .align 8
```

```
vals:                   #数组声明
    .quad 0x000000000000
    .quad 0x000000000000
    .quad 0x000000000000
    .quad 0x000000000000
    .quad 0x000000000000
    .quad 0x000000000000
    .quad 0x000000000000
    .quad 0x000000000000

jump_table:                 #跳转表声明
    .quad L1
    .quad L4
    .quad L2
    .quad L3
    .quad L4
    .quad L2

main:
    irmovq  vals, %r12

    #将循环拆在下面
    #rdi是参数

    irmovq  $-1,%rdi
    call switchv        # switchv(-1)
    rmmovq  %rax, (%r12)

    irmovq  $0,%rdi
    call switchv        # switchv(0)
    rmmovq  %rax, 0x8(%r12)

    irmovq  $1,%rdi
    call switchv        # switchv(1)
    rmmovq  %rax, 0x10(%r12)

    irmovq  $2,%rdi
    call switchv        # switchv(2)
    rmmovq  %rax, 0x18(%r12)

    irmovq  $3,%rdi
    call switchv        # switchv(3)
    rmmovq  %rax, 0x20(%r12)
```

```
    irmovq  $4,%rdi
    call switchv        # switchv(4)
    rmmovq  %rax, 0x28(%r12)

    irmovq  $5,%rdi
    call switchv        # switchv(5)
    rmmovq  %rax, 0x30(%r12)

    irmovq  $6,%rdi
    call switchv        # switchv(6)
    rmmovq  %rax, 0x38(%r12)

    ret

# long switchv(long idx)
# idx in %rdi
# 根据idx进行跳转
switchv:
    rrmovq %rdi, %r8
    irmovq $5, %r9
    subq %r9, %r8
    jg L4               #idx>5 default的情况
    andq %rdi, %rdi
    jl L4               #idx<0 default的情况
    irmovq jump_table, %r8
    irmovq $8, %r9
    irmovq $1, %r10
loop:                   #利用case块的字节数目相同 利用循环累加应该跳转的偏移值
    subq %r10, %rdi
    jl endloop
    addq %r9, %r8
    jmp loop
endloop:
    mrmovq (%r8), %r8 #根据计算的跳转表的地址进行跳转
    pushq %r8           #不包含间接跳转指令，先push后ret实现跳转
    ret

#switch case块中的执行内容
L1:             # case 0
    irmovq 0xaaa, %rax
    ret
L2:             #case 2 or case 5
    irmovq 0xbbb, %rax
    ret
```

```
L3:
    irmovq 0xccc, %rax  #case 3
    ret
L4:
    irmovq 0xddd, %rax  #default
    ret

# Stack starts here and grows to lower addresses
    .pos 0x400
stack:
```

4.51

阶段	laddq V，rB
取值	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
访存	
写回	R[rB] <- valE
更新 PC	PC <- valP