**PartA**

PartA要求用Y86\_64实现三个C写的函数，代码框架可以从书中找到，需要修改的只有核心的代码部分，相当于是写汇编代码，不过要注意一些Y86\_64和X86\_64的区别。

1. **链表求和（非递归）**

main函数只需要将链表开头ele1写入rdi寄存器中作为参数，再调用sum\_list函数并返回即可，如下所示：

**main:**

**irmovq ele1, %rdi # 用 %rdi 来传递参数**

**call sum\_list**

**ret**

sum\_list函数需要计算链表之和，逻辑如下：首先将rax置零准备存放和。之后，将rdi指向的结构体中的值移到rsi寄存器中，将rsi寄存器中值加到rax中，修改rdi中的值为链表的下一个。通过andq和jne操作判断rdi是否指向了空位置，如果不为空则跳回循环的开头，否则直接返回rax的值。如下所示：

**sum\_list:**

**irmovq $0, %rax # 初始化%rax存储返回值**

**loop:**

**mrmovq 0(%rdi), %rsi**

**addq %rsi, %rax # 计算和**

**mrmovq 8(%rdi), %rdi # 指向下一个元素**

**andq %rdi, %rdi # 判断当前指针是否为空**

**jne loop # 不为空则继续求和**

**loopend:**

**ret**

1. **链表求和（递归）**

递归版本的链表求和main函数不变，rsum\_list的逻辑如下：初始化rax，判断当前指针是否为空，如果为空则直接返回%rax。不为空则将当前节点的值暂存在%r8寄存器中，之后调整存放指针的%rdi指向下一位，之后递归的调用自己求之后的链表和（结果被存在了%rax中），最后将%r8与%rax相加即可。这样写之后发现不能通过，观察之后发现当前层将值存放在%r8之后，会递归调用自己，从而覆盖了%r8中的值，丢失当前层的数据，最后的%r8+%rax将会计算错误，因此在每一层的开头都应该加上pushq %r8，保护上一层的%r8中的数据，并在最后popq %r8恢复上一层%r8的数据，代码如下所示：

**rsum\_list:**

**pushq %r8 # 将上层存在%r8中的数据入栈**

**irmovq $0, %rax # 初始化%rax存储返回值**

**andq %rdi, %rdi # 判断当前指针是否为空**

**je rend # 如果指针为空则直接返回**

**mrmovq 0(%rdi), %r8 # 将本层的值放入%r8**

**mrmovq 8(%rdi), %rdi # 调整%rdi指向下一个节点**

**call rsum\_list # 递归调用函数**

**addq %r8, %rax # 计算当前值和后面值之和**

**rend:**

**popq %r8 # 回复上一层在%r8中的数据**

**ret**

1. **内存块内容复制**

复制数据块的main函数需要设置三个参数之后再调用copy\_block函数，如下所示：

**main:**

**irmovq src, %rdi # 第一参数**

**irmovq dest, %rsi # 第二参数**

**irmovq $3, %rdx # 第三参数**

**call copy\_block**

**ret**

copy\_block函数需要循环完成复制数据的操作，需要注意的是Y86\_64不支持常数参加运算，因此要在函数开头将需要参与计算的常数(1和8)存储在寄存器中，同时在两个内存中传递的数据也同样要存储在寄存器中。代码如下所示：

**copy\_block:**

**irmovq $0, %rax # 初始化%rax存储返回值**

**irmovq $1, %r8 # 存入要参与运算的常数**

**irmovq $8, %r9 # 存入要参与运算的常数**

**loop:**

**andq %rdx, %rdx # 判断len是否大于0**

**je loopend # 如果len等于0则跳出循环**

**mrmovq 0(%rdi), %r10 # 将src的值放入%r10，即val=\*src**

**addq %r9, %rdi # 调整src指向下一位，即src++(src += 8)**

**rmmovq %r10, (%rsi) # 将val的值存入dest指向的位置**

**addq %r9, %rsi # 调整dest指向下一位**

**xorq %r10, %rax # result ^= val**

**subq %r8, %rdx # len -= 1**

**jmp loop**

**loopend:**

**ret**