1

1.1

`SET` 过程的返回地址是 `GET` 过程的返回地址，返回值是 1。

1.2

`eax` 中存放的是 `GET` 过程的第一个参数，即用于存储当前处理器以及栈信息的内存块地址。

`ecx` 中存放的是 `GET` 过程的返回地址。

`(C)` 指令的作用是保存记录 `GET` 过程中，栈顶地址的上一位地址，在之后 `SET` 过程中，可将此地址赋值给 `%exp`，使其恢复原栈状态（配合 `pushl`）。

`(E)` 指令的作用是将 `GET` 过程的返回地址压栈，使得 `SET` 过程返回地址恢复为 `GET` 过程的返回地址。

`(D)` 指令补全为`movl 72(%eax), %esp`。

2

2.1

首先将当前协程的栈地址（`current_ctx`）记录给 `%rsi`，并将
`%rsp`、`%rbx`、`%rbp`、`%r12`、`%r13`、`%r14`、`%r15` 这几个通用寄存器的值利用 `%rsi` 存入存储地址。

之后进行协程切换，将新协程的栈地址信息（即调用参数 `%rdi`）覆盖当前栈地址（`current_ctx`），并将新协程存储在栈地址对应位置的各个寄存器值还原给了上述几个通用寄存器，完成了协程的切换。

2.2

因为只有这些寄存器属于 `Callee Saved`，即被调用者负责保存，在使用这些寄存器（协程切换走）之前将其原有内容存储在它的栈帧内，在退出（协程切换回来）之前恢复数据。

3

首先，在函数调用前（`function1` 中），预先在栈上分配好 `struct TEST_Struct` 结构体所需要的内存，然后将此时的栈指针 `%rsp` 即返回结构体的首地址赋值给 `%rdi`，作为调用函数 `return_struct` 的第一个参数。

进入 `return_struct` 后，便根据 `%rdi` 和偏移量，为结构体的各个成员通过传入的第二个参数 `%esi` 即 `n` 赋值，并在函数开始时，就设置返回值 `%rax` 为 `%rdi` 即预先分配的返回结构体首地址。

4

4.1

首先在 `function2` 最开始，分配好两个 `struct TEST_Struct` 结构体所需要的内存，其中存储在较底部位置的是局部变量 `main_struct`，较顶部位置的是传入的参数 `in_struct`。

实际并没有使用 `%rdi`、`%rsi` 等通用寄存器进行传参，在 `input_struct` 中，直接根据栈指针 `%rsp` 获取存储在栈上的 `in_struct` 结构体的信息。

4.2

编译器对于将 `function2` 调用 `input_struct` 的结果在编译期间直接计算了出来（根据 `i` 的值表示），省去了 `input_struct` 的调用直接返回结果，进而省去了两个临时结构体所需要的存储空间，以及相关赋值、移动命令的时间损耗。

但由于可能有其他外部文件调用 `input_struct` 函数，因此此函数并未被优化，保留了相关指令。

4.3

由于将 `input_struct` 设置为了 `static` 即静态函数，因此此函数不会被外部其他文件调用，而调用其的 `function2` 函数也进行了编译优化，省去了调用它的过程，因此此时 `input_struct` 函数属于无效（无意义）函数，会被编译器优化去除。

5

首先通过 `%rax` 记录 `n` 的值，然后进一步根据 `n` 计算出应在栈上开辟的内存大小，将后续将要使用到的 `Callee Saved` 的通用寄存器 `%rbp`、`%r14`、`%r13`、`%r12`、`%rbx` 存入栈中，利用 `%rbp` 备份 `%rsp` 的值。

之后将 `%rsp` 即栈指针减小对应的大小，进入循环部分。

循环中不断地将 `read_val` 得到的值从栈顶向下存储（按 8 位一个保存），当最终循环结束（即 `%r12 == %rbx`），根据栈中保存的数据，记录返回值 `%rax`，恢复栈指针 `%rsp`，并从栈中取出原先保存的几个通用寄存器。