На второй неделе предлагаю немного познакомиться с x86-ассемблером.
Для лучшего понимания x86-ассемблера (его связи с исходным кодом на языке C/C++) настоятельно рекомендую производить компиляции с использованием опции -O0, то есть максимально ограничить компилятор (gcc/clang) в применении оптимизаций. Если потребуется другой уровень оптимизации, я об этом напишу.

Для начала необходимо разобраться с утилитой objdump. По сути, она нужна для получения дизассемблера бинарного кода, то есть основной вариант ее запуска выглядит так:

$ objdump -d bin-file/obj-file

Попробуйте разобраться с полученным при помощи objdump представлением кода. Разбирать все опять же не стоит, это довольно сложно. Вновь предлагается воспользоваться простыми задачами и изучить их код в части функции main, а также функций, вызываемых из нее.
В качестве простых задач предлагаю следующие:
- реализовать набор функций с разным количеством входных аргументов — от нуля до 10 включительно, скомпилировать в 32- и 64-битном режимах (обратить внимание на то, как передаются аргументы);
- сложение/вычитание/наложение_маски/исключающее_или/логическое_или 32-/64-битных целочисленных значений (сделать это надо для одного и того же исходного кода, но собранного в 32-/64-битном режиме;
- сложение вещественных чисел формата long double;
- использование локальной переменной, являющейся структурой — необходимо в функции main явно проинициализировать ее поля, а затем передать в другую функцию через указатель;
- реализовать функцию поэлементного сложения массивов целочисленных значений (**в этом случае потребуется собрать на уровне опцимизации -O3, также надо будет использовать два отдельных исходных файла, в одном из них должна располагаться функция main, а во втором — функция подсчета суммы**), цель этого теста — заставить компилятор использовать векторные sse- или avx-инструкции.

Для того, чтобы компилятор не свернул код, как ненужный, на всякий случай рекомендую печатать на экран результаты выполненных действий.
Все указанные простые действия необходимо реализовывать в виде отдельных функций, вызываемых из main, которым в качестве аргументов передаются константы, переменные или указатели на них.
Для того, чтобы понять дизассемблер необходимо знать, что такое стек и регистры, а также то, каким образом происходит передача аргументов из вызывающей функции в вызываемую. Данную информацию в первую очередь рекомендую читать в intel-документации (ее без проблем найдете сами):
Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 1: Basic Architecture
Читайте главы 3.2 (достаточно читать про Address space, Basic program execution registers, XMM registers, Stack), 3.4.1, 3.4.2, 3.4.3, 3.5.
Что касается ассемблерных операций, то их описание можно найти в другом руководстве от intel (см. главы 3 и 4):
Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 2 (2A, 2B & 2C): Instruction Set Reference, A-Z

На хабре можно почитать этот обзор:
https://habr.com/ru/companies/ruvds/articles/646629/

После того как справитесь с этим, рекомендую дизассемблировать тесты, написанные на

первой неделе и использующие функцию syscall.