**4)**Разберите дизассемблерный листинг функции countSetBit, уделите особое внимание ассемблерным инструкциям, с помощью которых реализованы арифметические действия, битовые операции, сравнения и логические операторы языка С.

Суть: производя операцию побитового И над аргументом и степенями двойки, мы обнуляем все биты, кроме бита, стоящего на позиции, равной степени двойки,-этот бит остаётся неизменённым. То есть если он был 0, то 0 и останется, аналогично для 1. Таким образом мы получаем отдельно друг от друга все биты числа. Сложим их. Эта сумма и есть кол-во единиц в двоичной записи числа.

0x080003AA F0010001 AND r0,r1,#0x01

Данная команда выполняет побитовое И аргумента функции и двойки в степени 0. Таким образом мы проверяем, стоит ли единица в 31-ом бите (последний по big-endian).

0x080003FC F3C13280 UBFX r2,r1,#14,#1

0x08000400 4410 ADD r0,r0,r2

Вплоть до 30 бита (по little-endian) программа выполняет аналогичные действия: считывает последовательно идущие один (я русский) бит, начиная с 1,2,3 и тд, и прибавляет его к сумме, хранящейся в отдельном регистре. Таким образом, считывая единичку, мы добавляем к сумме единицу, считывая нуль-соответственно нуль.

0x08000462 EB0070D1 ADD r0,r0,r1,LSR #31

Затем наше число сдвигается логически на 31 бит вправо (остаётся только первая по big-endian цифра). Оставшаяся цифра прибавляется к уже подсчитанной сумме.

0x08000466 B2C0 UXTB r0,r0

Подсчитанная сумма расширяется до 32 бит. Зачем это делается, если мы возвращаем 8-битное значение, – непонятно