char str[]="12345";//enter some stroke

0x08000448 A10D ADR r1,{pc}+0x38 ; @0x08000480

В функции main происходит загрузка адреса, по которому хранятся символы, которые будут в дальнейшем обрабатываться.

0x0800044A C903 LDM r1,{r0-r1}

Затем этот набор символов (строка) сохраняется в регистры. В данном случае, в регистры с нулевого по первый. Так как размеры регистра 8 бит, а мы загружаем 5\*2 бит, приходится использовать 2 регистра.

0x0800044C E9CD0101 STRD r0,r1,[sp,#0x04]

Наша строка сохраняется по адресу, хранящемуся в стеке (переменная str локальная и создана в main)

После этого дважды используется стандартная функция strlen.

А потом уже начинает работать функция, которая получает на вход строку с символьной записью целого числа и возвращает соответствующее целое число.

volatile int32\_t answer=strToInt(str);

0x08000464 A801 ADD r0,sp,#0x04

Записываем в нулевой регистр адрес переменной, являющейся параметром в вызываемой функции.

0x08000466 F7FFFF9F BL.W strToInt (0x080003A8)

Переходим по адресу, где расположено тело функции.

0x080003A8 E92D47F0 PUSH {r4-r10,lr}

Загружаем регистры с 4 по 10 и регистр lr (адрес возврата) в стек. Регистр lr загружается так как была выполнена команда BL.W (присутствует L). Это контекст вызова функции strToInt. При этом значение в регистре R13(SP) изменилось на 0x20, что и отражает запись значений из восьми регистров.

0x080003AC B08A SUB sp,sp,#0x28

Данная команда переставляет stack pointer на величину стекового кадра в сторону увеличения стека.

0x080003AC B08A SUB sp,sp,#0x28

Затем адрес переменной в данной функции переносится в другой регистр.

Сейчас начинается действие функции strToInt.

26: int32\_t number=0;

0x080003B0 F04F0A00 MOV r10,#0x00

27: int32\_t power=0;

0x080003B4 2500 MOVS r5,#0x00

Создаются две локальные переменные, инициализирующиеся нулями.

28: for(int32\_t i=lenstr\_crutch-1;i>=0;i--)//reading the number backwards

29: {

0x080003B6 482E LDR r0,[pc,#184] ; @0x08000470

0x080003B8 6800 LDR r0,[r0,#0x00]

0x080003BA 1E46 SUBS r6,r0,#1

0x080003BC E03F B 0x0800043E

Инициализируется счётчик в цикле

0x0800043E 2E00 CMP r6,#0x00

0x08000440 DABD BGE 0x080003BE

Счётчик сравнивается на Greater or equal и делается переход (в случае соблюдения условия) к внутреннему циклу

30: for(int32\_t j=0;j<lengoogle\_crutch;j++)

31: {

0x080003BE 2400 MOVS r4,#0x00

0x080003C0 E038 B 0x08000434

Инициализация счётчика во внутреннем цикле.

0x08000434 4811 LDR r0,[pc,#68] ; @0x0800047C

0x08000436 6800 LDR r0,[r0,#0x00]

0x08000438 4284 CMP r4,r0

0x0800043A DBC2 BLT 0x080003C2

Загрузка в регистры «контрольного» значения (длина массива google), с которым и сравнивается, но теперь строго, Less Than наш счётчик. Переход к сравнению символа строки с символом алфавита из цифр и знака минус:

32: if(str[i]==google[j])

33: {

0x080003C2 F8190006 LDRB r0,[r9,r6]

0x080003C6 492B LDR r1,[pc,#172] ; @0x08000474

0x080003C8 5D09 LDRB r1,[r1,r4]

0x080003CA 4288 CMP r0,r1

0x080003CC D131 BNE 0x08000432

В регистр r0 загружается str[i], в регистр r1 загружается адрес массива google, а потом элемент google[j]. Эти два элемента сравниваются. Если они не равны, то:

0x08000432 1C64 ADDS r4,r4,#1

0x08000434 4811 LDR r0,[pc,#68] ; @0x0800047C

0x08000436 6800 LDR r0,[r0,#0x00]

0x08000438 4284 CMP r4,r0

0x0800043A DBC2 BLT 0x080003C2

Счётчик j инкрементируется и сравнивается с «контрольным» значением в этом цикле.

Когда условие неравенства всё же не выполнится, перехода

0x080003CC D131 BNE 0x08000432 не произойдёт. Мы ныряем ещё глубже в функцию и проверяем, не минус ли мы рассматриваем.

0x080003CE 2C0A CMP r4,#0x0A

0x080003D0 D104 BNE 0x080003DC

В случае равенства будет возвращено значение:

47: return number;

0x080003D2 F1CA0000 RSB r0,r10,#0x00

Помещение в r0 результат разности 0 и r10 (number). Подробнее выход из функции рассмотрю на втором return.

Так вот. Если же наш символ не “-“, то переход осуществится, то будут происходить страшные вещи:

40: number+=j\*pow(10,power);//representing number as the summ of orders of 10 with coefficients

0x080003DC 4628 MOV r0,r5

0x080003DE F001F89C BL.W \_\_aeabi\_i2d (0x0800151A)

0x080003E2 4602 MOV r2,r0

0x080003E4 460B MOV r3,r1

0x080003E6 E9CD0100 STRD r0,r1,[sp,#0]

0x080003EA 2000 MOVS r0,#0x00

0x080003EC 4922 LDR r1,[pc,#136] ; @0x08000478

0x080003EE F000F93F BL.W pow (0x08000670)

0x080003F2 4607 MOV r7,r0

0x080003F4 4688 MOV r8,r1

0x080003F6 4620 MOV r0,r4

0x080003F8 F001F88F BL.W \_\_aeabi\_i2d (0x0800151A)

0x080003FC 463A MOV r2,r7

0x080003FE 4643 MOV r3,r8

0x08000400 E9CD0102 STRD r0,r1,[sp,#0x08]

0x08000404 F001F8DC BL.W \_\_aeabi\_dmul (0x080015C0)

0x08000408 E9CD0106 STRD r0,r1,[sp,#0x18]

0x0800040C 4650 MOV r0,r10

0x0800040E F001F884 BL.W \_\_aeabi\_i2d (0x0800151A)

0x08000412 E9CD0104 STRD r0,r1,[sp,#0x10]

0x08000416 E9DD2306 LDRD r2,r3,[sp,#0x18]

0x0800041A F000FE3B BL.W \_\_aeabi\_dadd (0x08001094)

0x0800041E E9CD0108 STRD r0,r1,[sp,#0x20]

0x08000422 F001F84B BL.W \_\_aeabi\_d2iz (0x080014BC)

0x08000426 4682 MOV r10,r0

Здесь происходило: преобразование типа (а иногда и в стиле си), умножение и сложение с double-точностью. В общем, данный кусок кода прибавляет к значению, хранящемуся в регистре10 (number) произведение счётчика внутреннего цикла (он совпадает с цифрой из соответствующего элемента массива google) на степень десятки (power), которая инкрементируется после каждого увеличения числа number:

0x08000428 1C6D ADDS r5,r5,#1

42: if(power==9) return number;//cuts leftover digits and signs)))

0x0800042A 2D09 CMP r5,#0x09

0x0800042C D101 BNE 0x08000432

0x0800042E 4650 MOV r0,r10

0x08000430 E7D1 B 0x080003D6

Затем степень power проверяется на равенство 9. В случае равенства функция возвращает посчитанное на данный момент число number. Это сделано грубо, чтобы не делать громоздких проверок на границы int32.

0x08000442 4650 MOV r0,r10

0x08000444 E7C7 B 0x080003D6

Если случилось так, что внешний цикл «прокрутился» полностью, то после него будет записано получившееся число в регистр r0.

0x080003D6 B00A ADD sp,sp,#0x28

0x080003D8 E8BD87F0 POP {r4-r10,pc}

Переставляем обратно значение stack pointer в состояние, которое было перед вызовом функции. Пропускаем стековый кадр и выгружаем из стека контекст вызова функции strToInt.

0x0800046A 9000 STR r0,[sp,#0x00]

Значение из r0 (там у нас хранится возвращённое функцией значение) записывается по адресу, хранящемуся в sp (а на вершине стека у нас хранится адрес последней созданной локальной переменной, то есть переменной answer).