И я начал собирать отчет из ваших шести вордовых документов -\_-'

А я начал вспоминать, что я делал месяц назад) + тоже пришлось покопаться в вордовых документах)))

1)

    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

А почему цикл начинается с безусловного перехода? Изменилось бы что-нибудь, если цикл бы "вывернуло" наоборот - чтобы он начинался с условного перехода, а в конце был бы безусловный переход в начало?

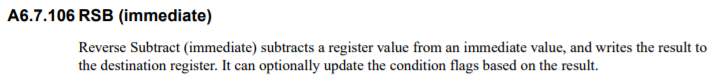
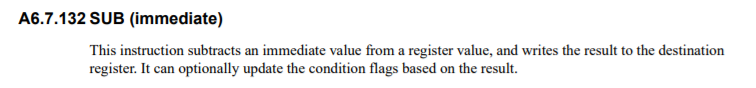
Цикл начинается с безусловного перехода так как хотя бы один цикл будет выполнен в любом случае (длина строки >=1 из-за наличия нуль-символа). Вторую часть вопроса не уверен, что понял корректно. Но описанный формат очень похож на цикл for, который, вроде как, я и пытался внедрить) В начале сравниваем значение счётчика с условием (условный переход), затем прогоняем итерацию, изменяем счётчик, переходим в начало (безусловный переход) и сравниваем там (снова условный переход).

2)

0x080003D2 F1CA0000  RSB      r0,r10,#0x00

Помещение в r0 результат разности 0 и r10 (number).

А почему тут используется именно команда RSB? Почему бы не использовать команду SUB?



Потому что мы из мгновенного значения вычитаем значение в регистре, а не наоборот. Собственно, инструкции заточены под «обратные» действия.

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

Это не очень хорошо. Т.к. функция pow оперирует аргументами типа double, а на нашем контроллере нет аппаратной поддержки плавающей точки, все операции с double и float реализуются с помощью функций и выполняются очень долго (и страшно).

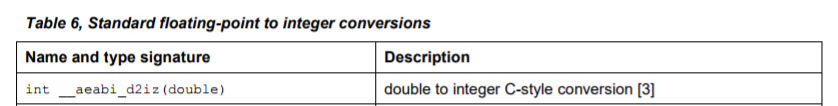
Имейте в виду на будущее.

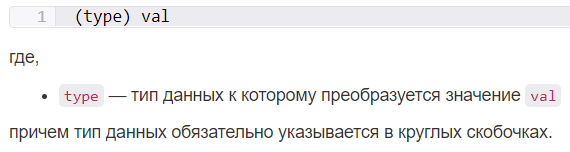
Принял) не пришло в голову посмотреть спецификацию функции перед использованием.

4) Здесь происходило: преобразование типа (а иногда и в стиле си)

Не понял вас. Что значит "в стиле си"?

Сам с недоумением пытался понять, кто это написал. Но выходит, это сделал я-мне и отвечать.





В общем, это тот простенький тип преобразования, которым мыы всегда пользовались) только на языке ассемблера. Небезопасно, но, интернет говорит, компилятор не против.

5)

0x08000466 B2C0      UXTB     r0,r0

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

Действительно, непонятно. Изменилось бы что-нибудь, если бы этой команды не было?

Попробовал даже насильно преобразовать в Си-стиле возвращаемое значение, всё равно расширяет до 32 бит. На stackoverflow говорят, что это чудо пропадает при оптимизации.

«The ARM ABI requires that short and char result types are extended to 32 bits (using signed or unsigned extension as appropriate) before being returned. Since the compiler can't see inside the asm block to work out what is being done it cannot remove the uxth instruction on the result.»

Магическое решение я вам засчитаю, потому что оно и правда магическое, хотя по сути это просто развернутый цикл. Думаю, вам уже известно, что существует магическое решение, которое еще и быстрее, чем цикл.

Говорили про отсутствие цикла) я и сделал) ну. Своего рода магия в извращённом считывании битов. Развёрнутый цикл, конечно, -не здорово. У меня была сразу мысль, что можно хитрым способом подключить арифметику, чтобы разобрать число на биты, но не было желания, а может, и умения эти числа вычислять и подбирать) Но я на хабре глянул статью про магию эту. Очень добротно.