Санкт-Петербургский Политехнический университет Петра Великого

Институт металлургии, машиностроения и транспорта

Кафедра: Мехатроника и роботостроение

Схемотехника и организация ЭВМ

Лабораторная работа № 3

### «[Арифметика и битовые операции](http://dl.avalon.ru/mod/assign/view.php?id=33861)»

Работу выполнил

студент группы 33328/2:

Никифоров А.А.

Санкт-Петербург, 2017

**1)**  
**а)**Напишите функцию, которая получает на вход строку с символьной записью целого числа и возвращает соответствующее целое число. Эта функция должна иметь следующий прототип:  
  int32\_t strToInt( const char \* str );

  Для входных данных гарантируется следующее:  
  - аргумент функции является корректным указателем на строку символов в кодировке ASCII;  
  - строка заканчивается нуль-символом ( '\0' );  
  - число в строке записано в десятичном виде;  
  - число в строке не содержит ведущих нулей;  
  - число в строке гарантированно влезает в диапазон, представимый типом int32\_t (т.е. находится в пределах от -2 147 483 648 до 2 147 483 647);  
  - отрицательные числа начинаются с символа '-', неотрицательные числа начинаются с цифры.

  Ваша функция должна игнорировать все символы, кроме цифр и знака минус. Если во входной строке вообще нет корректных символов, функция должна возвращать 0.  
  Ваша функция не должна изменять входную строку и не должна приводить к неопределенному поведению.

  Ваша функция не должна использовать библиотечные функции языка С для форматированного ввода/вывода (такие, как sprintf, sscanf, vsprintf, vprintf и т.д.), т.е. собственно чтение строки и формирование числа должен делать написанный вами код; но эти функции рекомендуется использовать для самопроверки.   
  Допустимо использовать библиотечные функции в качестве вспомогательных (такие, как strlen).

(Но если вам когда-нибудь еще потребуется переводить строку в число - ради всего сущего, пользуйтесь библиотечными функциями, а не изобретайте велосипед!)

  Пример использования:

  int32\_t a = strToInt("12345"); // a будет равно 12345  
  a = strToInt("292 134 517"); // a будет равно 292134517  
  a = strToInt("-1"); // a будет равно -1  
  a = strToInt("kokoko"); // a будет равно 0

**Рабочий код:**

#include "CMSIS/stm32f10x.h"

#include "stdint.h"

int32\_t a;

int32\_t strToInt( const char \* str )

{

int32\_t i=0;

int32\_t ans = 0;

int32\_t sign = 0;

int32\_t n = strlen(str);

if (str[i] == '-')

{

sign = 1;

i++;

}

for (i;i<n;i++)

{

if (str[i] >= 0x30 && str[i] <= 0x39)

{

ans = ans + (str[i] & 0x0F);

ans = ans\*10;

}

}

ans = ans/10;

if (sign == 1)

ans = -ans;

return ans;

}

int main(void)

{

a=strToInt("-1 2f3");

}

**б)** Разберите дизассемблерный листинг этой функции, уделите особое внимание ассемблерным инструкциям, с помощью которых реализованы арифметические действия, битовые операции, сравнения и логические операторы языка С. Вы можете разбирать весь листинг подряд или разбирать только различающиеся фрагменты, как вам удобнее. При этом находите соответствия между кодом на С и ассемблером, а не просто "переводите" ассемблер на русский!

**Дизассемблерный листинг:**

5: {

0x080003A4 E92D41F0 PUSH {r4-r8,lr}

0x080003A8 4605 MOV r5,r0

6: int32\_t i=0;

0x080003AA 2400 MOVS r4,#0x00

7: int32\_t ans = 0;

0x080003AC 2600 MOVS r6,#0x00

8: int32\_t sign = 0;

0x080003AE 2700 MOVS r7,#0x00

9: int32\_t n = strlen(str);

0x080003B0 4628 MOV r0,r5

0x080003B2 F000F82F BL.W strlen (0x08000414)

0x080003B6 4680 MOV r8,r0

10: if (str[i] == '-')

11: {

0x080003B8 5D28 LDRB r0,[r5,r4]

0x080003BA 282D CMP r0,#0x2D

0x080003BC D101 BNE 0x080003C2

12: sign = 1;

0x080003BE 2701 MOVS r7,#0x01

13: i++;

14: }

0x080003C0 1C64 ADDS r4,r4,#1

15: for (i;i<n;i++)

16: {

0x080003C2 E00D B 0x080003E0

17: if (str[i] >= 0x30 && str[i] <= 0x39)

18: {

0x080003C4 5D28 LDRB r0,[r5,r4]

0x080003C6 2830 CMP r0,#0x30

0x080003C8 DB09 BLT 0x080003DE

0x080003CA 5D28 LDRB r0,[r5,r4]

0x080003CC 2839 CMP r0,#0x39

0x080003CE DC06 BGT 0x080003DE

19: ans = (ans + (str[i] & 0x0F))\*10;

20: }

21: }

0x080003D0 5D28 LDRB r0,[r5,r4]

0x080003D2 F000000F AND r0,r0,#0x0F

0x080003D6 4430 ADD r0,r0,r6

0x080003D8 EB000080 ADD r0,r0,r0,LSL #2

0x080003DC 0046 LSLS r6,r0,#1

0x080003DE 1C64 ADDS r4,r4,#1

0x080003E0 4544 CMP r4,r8

0x080003E2 DBEF BLT 0x080003C4

22: ans = ans/10;

0x080003E4 200A MOVS r0,#0x0A

0x080003E6 FB96F6F0 SDIV r6,r6,r0

23: if (sign == 1)

0x080003EA 2F01 CMP r7,#0x01

0x080003EC D100 BNE 0x080003F0

24: ans = -ans;

0x080003EE 4276 RSBS r6,r6,#0

25: return ans;

0x080003F0 4630 MOV r0,r6

26: }

27: int main(void)

0x080003F2 E8BD81F0 POP {r4-r8,pc}

28: {

0x080003F6 B510 PUSH {r4,lr}

29: a=strToInt("-1 2f3");

0x080003F8 A003 ADR r0,{pc}+4 ; @0x08000408

0x080003FA F7FFFFD3 BL.W strToInt (0x080003A4)

0x080003FE 4904 LDR r1,[pc,#16] ; @0x08000410

0x08000400 6008 STR r0,[r1,#0x00]

**Разбор листинга:**

***5: {***

***0x080003A4 E92D41F0 PUSH {r4-r8,lr}***

***0x080003A8 4605 MOV r5,r0***

***6: int32\_t i=0;***

***0x080003AA 2400 MOVS r4,#0x00***

***7: int32\_t ans = 0;***

***0x080003AC 2600 MOVS r6,#0x00***

***8: int32\_t sign = 0;***

***0x080003AE 2700 MOVS r7,#0x00***

***9: int32\_t n = strlen(str);***

***0x080003B0 4628 MOV r0,r5***

***0x080003B2 F000F82F BL.W strlen (0x08000414)***

***0x080003B6 4680 MOV r8,r0***

Сначала функция помещает содержимое регистров r4-r8 и lr(хранит адрес возврата при вызове функции) в стек, и обнуляет их.   
r4 – счетчик цикла, r5 – адрес начала строки, r6 – искомое число, r7 – знак числа, r8 – длина строки.

***10: if (str[i] == '-')***

***11: {***

***0x080003B8 5D28 LDRB r0,[r5,r4]***

***0x080003BA 282D CMP r0,#0x2D***

***0x080003BC D101 BNE 0x080003C2***

***12: sign = 1;***

***0x080003BE 2701 MOVS r7,#0x01***

***13: i++;***

***14: }***

Сначала в регистр r0 помещаем значение адреса r5+r4: это первый элемент строки;

Далее сравнивается код первого элемента с кодом ‘-‘;  
Затем стоит постфикс NE который означает, что если на предыдущей строке в результате операции сравнения коды оказались неравными, то происходит переход по адресу 0x080003C2 (т.к. стоит В). Иначе осуществляется переход к следующей строке. Проверка осуществляется по флагу Z, который обновляется в соответствии с результатом сравнения CMP ( Z=0 - значения равны друг другу и Z=1 - значения не равны).

Так выполняется операция сравнения «==».

***15: for (i;i<n;i++)***

***16: {***

***0x080003C2 E00D B 0x080003E0***

***17: if (str[i] >= 0x30 && str[i] <= 0x39)***

***18: {***

***0x080003C4 5D28 LDRB r0,[r5,r4]***

***0x080003C6 2830 CMP r0,#0x30***

***0x080003C8 DB09 BLT 0x080003DE***

***0x080003CA 5D28 LDRB r0,[r5,r4]***

***0x080003CC 2839 CMP r0,#0x39***

***0x080003CE DC06 BGT 0x080003DE***

***19: ans = (ans + (str[i] & 0x0F))\*10;***

***20: }***

***21: }***

***0x080003D0 5D28 LDRB r0,[r5,r4]***

***0x080003D2 F000000F AND r0,r0,#0x0F***

***0x080003D6 4430 ADD r0,r0,r6***

***0x080003D8 EB000080 ADD r0,r0,r0,LSL #2***

***0x080003DC 0046 LSLS r6,r0,#1***

***0x080003DE 1C64 ADDS r4,r4,#1***

***0x080003E0 4544 CMP r4,r8***

***0x080003E2 DBEF BLT 0x080003C4***

Цикл начинается со сравнения значения счетчика цикла и размера строки(***0x080003E0 4544 CMP r4,r8)***

Следующая команда - (***0x080003E2 DBEF BLT 0x080003C4)***. Здесь стоит флаг LT , который проверяет результат сравнения (команда CMP в предыдущей строке) на основе флагов N и V. Если флаги равны, то выполняется переход на содержимое цикла: для проверки условия вхождения символа в диапазон кодов цифр идет комбинация сравнений, объединенных логическим «и»(&&).   
В цикле:   
***0x080003C4 5D28 LDRB r0,[r5,r4]***

***0x080003C6 2830 CMP r0,#0x30***

***0x080003C8 DB09 BLT 0x080003DE***

***0x080003CA 5D28 LDRB r0,[r5,r4]***

***0x080003CC 2839 CMP r0,#0x39***

***0x080003CE DC06 BGT 0x080003DE***

В r0 записывается код элемента и проверяются оба условия сравнения: BLT – меньше, BGT – больше (рассмотренная ранее операция сравнения «==» работает по схожему принципу).

В самом ассемблере логический «и» отсутствует. Элемент, не являющийся числом, пройдет по одному из условий, но не пройдет по другому.

***0x080003D0 5D28 LDRB r0,[r5,r4]***

***0x080003D2 F000000F AND r0,r0,#0x0F***

Здесь представлена битовая операция «и», выполняющаяся при помощи команды AND. В регистр r0 помещается код символа, умноженный на 0x0F.

***0x080003D8 EB000080 ADD r0,r0,r0,LSL #2***

***0x080003DC 0046 LSLS r6,r0,#1***

Здесь представлена арифметическая операция умножения:  
Сначала к значению из регистра r0 прибавляется значение, которое получается сдвигом значения из регистра r0 влево на 2 бита (эти два бита заполняются нулями), затем полученное значение записывается в регистр r0.  
Далее в r6 записывается содержимое r0, сдвинутое на 1 бит влево.

***0x080003DE 1C64 ADDS r4,r4,#1***Здесь происходит арифметическая операция сложения (команда ADDS). Команда добавляет 1 к значению, которое хранится в регистре r4 и записывает полученный результат в тот же регистр.

***22: ans = ans/10;***

***0x080003E4 200A MOVS r0,#0x0A***

***0x080003E6 FB96F6F0 SDIV r6,r6,r0***

Деление переменной на константу: в регистр r0 записывается делитель, а операция деления осуществляется за счет команды SDIV.

***24: ans = -ans;***

***0x080003EE 4276 RSBS r6,r6,#0***

Операция присваивания переменной ее обратного значения происходит так: в регистр r6 записывается значение полученное вычитанием из 0 значения этого регистра r6.

**в)** Измените тип всех использованных вами в функции strToInt локальных переменных на int64\_t и перекомпилируйте программу. Опишите, как изменился дизассемблерный листинг.

***6: int64\_t i=0;***

***0x080003AA 2100 MOVS r1,#0x00***

***0x080003AC 460C MOV r4,r1***

***0x080003AE 460D MOV r5,r1***

***7: int64\_t ans = 0;***

***0x080003B0 460E MOV r6,r1***

***0x080003B2 460F MOV r7,r1***

***8: int64\_t sign = 0;***

***0x080003B4 4688 MOV r8,r1***

***0x080003B6 4689 MOV r9,r1***

Для локальных переменных выделилось по 2 регистра (r6,r7-для переменной; r8,r9 - для знака; r4,r5 – для счетчика цикла).

***9: int64\_t n = strlen(str);***

***0x080003B8 4650 MOV r0,r10***

***0x080003BA F000F879 BL.W strlen (0x080004B0)***

***0x080003BE 17C1 ASRS r1,r0,#31***

***0x080003C0 E9CD0100 STRD r0,r1,[sp,#0]***

Регистр r10 –адрес начала строки. C помощью команды STRD(постфикс D обозначает двойное слово) в стек помещается размер строки. Для нахождения первой половины используется арифметический сдвиг вправо на 31 позицию со сменой флага(ASRS).

Проверка условий цикла выглядит так:

***0x08000410 E9DD1000 LDRD r1,r0,[sp,#0]***

В регистр r1 из стека помещается размер строки.

***0x08000414 1A61 SUBS r1,r4,r1***

В r1 записывается разность первых половин счетчика цикла и размера строки (r4-r1).

***0x08000416 EB750000 SBCS r0,r5,r0***

В r0 записывается разность вторых половин.

***0x0800041A DBDF BLT 0x080003DC***

Проверка флагов.

Ассемблер использует вычитание, так как не сможет с помощью команды CMP сравнивать сразу 2 пары чисел.

***20: ans = ans\*10;***

***21: }***

***22: }***

***0x080003FC F04F020A MOV r2,#0x0A***

***0x08000400 F04F0C00 MOV r12,#0x00***

***0x08000404 4631 MOV r1,r6***

***0x08000406 FBA10B02 UMULL r0,r11,r1,r2***

***0x0800040A FB07B202 MLA r2,r7,r2,r11***

***0x0800040E FB06270C MLA r7,r6,r12,r2***

***0x08000412 4606 MOV r6,r0***

Так как оба числа состоят из двух частей, умножение осуществляется таким образом:   
Результат команды(32-битной) умножения без знака UMULL - 64-битное слово, которое записывается в r11, r12. При этом данные, хранящиеся в r12 являются частью ответа и записывается в r6, где хранилась переменная. Далее командой MLA выполняется умножение первой части переменной на первую часть множителя с прибавлением данных из r11 и результат записывается в r2. Затем той же командой (MLA) вторая часть переменной умножается на вторую часть множителя с прибавлением данных из r1, результат записывается в r7.

**2)** Напишите функцию, которая подсчитывает (и возвращает) количество равных единице бит в двоичном представлении числа.   
  Прототип функции должен выглядеть так:  
  
  uint8\_t countSetBits( uint32\_t n);  
  
  Пример работы:   
  countSetBits(5); // возвращает 2 (в двоичном виде 5 = 101 - два бита равны единице)  
  countSetBits(15); // возвращает 4 (в двоичном виде 15 = 1111 - четыре бита равны единице)

Рабочий код:

uint8\_t countSetBits(uint32\_t arg)

{

arg = (arg & 0x55555555) + ((arg >> 1) & 0x55555555);

arg = (arg & 0x33333333) + ((arg >> 2) & 0x33333333);

arg = (arg & 0x0F0F0F0F) + ((arg >> 4) & 0x0F0F0F0F);

arg = (arg & 0x00FF00FF) + ((arg >> 8) & 0x00FF00FF);

arg = (arg & 0x0000FFFF) + ((arg >>16) & 0x0000FFFF);

return arg;

}

int main(void)

{

uint8\_t arg=countSetBits(5);

}

Здесь число 0x55555555=(01010101010101010101010101010101), значит, что (arg & 0x55555555) обнуляет нечетные биты числа, а ((arg >> 1) & 0x55555555) — четные. Результатом сложения полученных чисел будет число, в каждой паре бит которого записано количество единиц в этой паре. Число 0x33333333=00110011001100110011001100110011 для записи в каждую четверку бит количества единиц в ней, 0x0F0F0F0F=00001111000011110000111100001111 для записи в каждую восьмерку бит количества единиц в ней, 0x00FF00FF=00000000111111110000000011111111 для записи количества единиц в 16 бит и 0x0000FFFF= 00000000000000001111111111111111 для единиц в 32 битах.

**3)** Напишите функцию, которая подсчитывает количество ведущих нулей в двоичном представлении числа.  
  Ведущие нули - это нулевые биты в старших разрядах числа.   
  
  Прототип этой функции должен выглядеть так:  
  
  uint8\_t countLeadingZeros( uint32\_t n);

Рабочий код:

uint8\_t countLeadingZeros(uint32\_t k)

{

int16\_t l, m, n;

l = -(k >> 16);

m = (l >> 16) & 16;

n = 16 - m;

k = k >> m;

l = k - 0x100;

m = (l >> 16) & 8;

n = n + m;

k = k << m;

l = k - 0x1000;

m = (l >> 16) & 4;

n = n + m;

k = k << m;

l = k - 0x4000;

m = (l >> 16) & 2;

n = n + m;

k = k << m;

l = k >> 14;

m = l & ~(l >> 1);

n = n + 2;

n = n - m;

return n;

}

Функция смотрит, если левая половина k равна 0, тогда n=16, если же левая половина k не равна 0, то n=0, а k сдвигается вправо на 16 бит (принимает вид 0000kkkk). Если разряды с 8 по 15 равны 0, то к n прибавляется 8 и производится сдвиг k на 8 влево, если разряды 12-15=0, то n=n+4, k сдвигается влево на 4, если 14-15=0, n=n+2, k сдвигается влево на 2. y становится 0/1/2/3, n=0/1/2.

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

countSetBits

0x080003AE EA020251 AND r2,r2,r1,LSR #1

Битовая операция «и» между содержимым регистра r2 и содержимым регистра r1, сдвинутым вправо на 1.

0x080003CE 1881 ADDS r1,r0,r2  
Арифметическая операция сложения.

countLeadingZeros

0x080003EE EA240264 BIC r2,r4,r4,ASR #1  
Битовая операция «и-не».

0x080003AA 4240 RSBS r0,r0,#0

Обратное вычитание из нуля содержимого регистра r0.

ссылочку на гит скину в комментарий…