**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

**МИКРОПРОЕКТ**

“Программа на ассемблере для переворота строки”

Пояснительная записка

Выполнил студент группы БПИ196 (2)

Шестаков Михаил Сергеевич

**1. Условие**

Разработать программу, которая "переворачивает на месте" заданную ASCII-строку символов

**2. Методы решения**

**2.1. Считывание строки**

Для считывания строки используются функция **fgets**.[1] Для получения указателя на handle потока чтения пришлось использовать плохо документированную функцию \_\_iob\_func.

Так же поскольку размер строки заранее неизвестен используется динамическое увеличение буфера для строки с помощью функции **realloc**. [2]

Для нахождения символа перехода используется инструкция **rep scas**. [3] [4]

**2.2. Разворот строки**

Для поиска конца строки и нахождения количества элементов используется инструкция **rep** **scas**. [3] [4]

Сам разворот строки разбит на два этапа:

1. Разворот с использованием SSE3

Для этого алгоритм загружает 16 байт из начала строки и 16 байт из конца строки в регистры **xmm0** и **xmm1** с использованием инструкции **movups**, потом меняет порядок байт в них с помощью инструкции **pshufb**, и складывает обратно в строку с помощью **movups** (см. рис.1). [4][5]

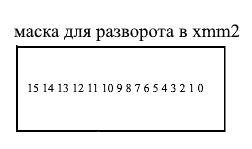
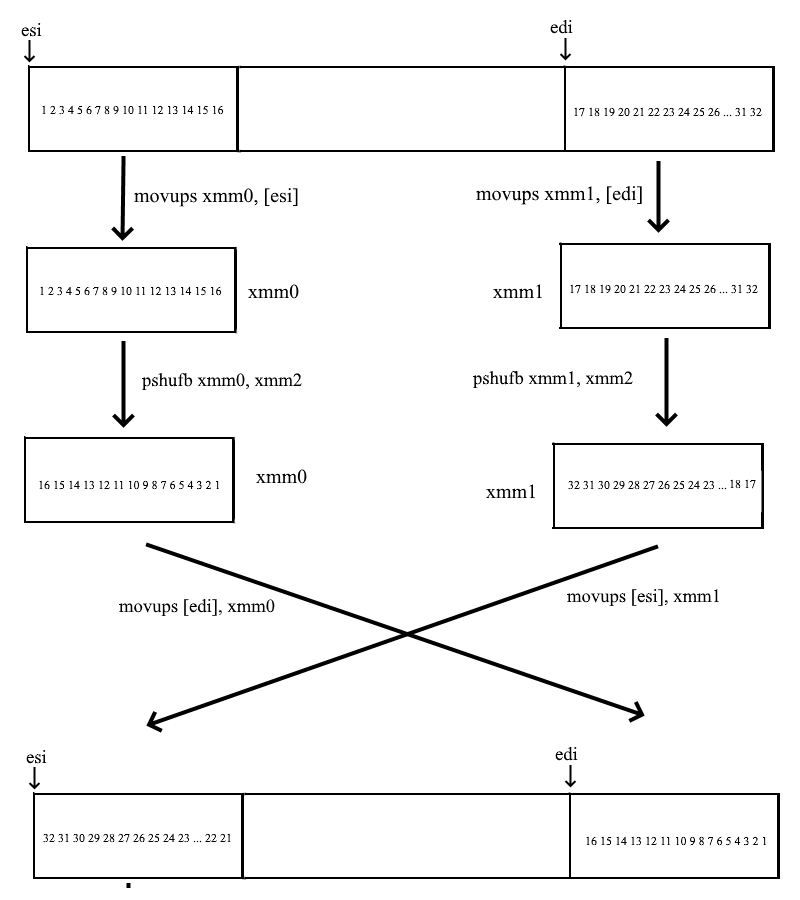
**** ****

Рис. 1 Разворот строки с использованием SSE3 инструкций

1. Разворот с использованием строковых операций

Поскольку строковые операции не поддерживают одновременное перемещение **esi** вправо и **edi** влево, то реализовать разворот с использованием инструкции **rep** невозможно. Поэтому разворот строки выполняется в цикле: с помощью команды **movsb** осуществляется перенос символов из **esi** в **edi**, а затем уже с помощью обычной **mov** копируется символ из **edi** в **esi**. (более подробно см. в коде) [7]

**3. Текст программы**

*; Шестаков Михаил Сергеевич*

*; Группа БПИ196*

format PE console

entry **start**

include 'win32a.inc'

**section** '.data' **data** readable writable

strAskString **db** 'Input string: ', 0

*; маска для разворота 16 байт в обратную сторону*

shuffleOrder **db** 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

strFormatInputStr **db** 'Initial string: "%s"', 10, 0

strFormatResultStr **db** 'Reversed string: "%s"', 10, 0

strFormatInt **db** '%d', 10, 0

strPointer **dd** 0

**section** '.code' **code** readable executable

**start**:

cinvoke printf, strAskString *; просим пользователя ввести строку*

ccall ReadString *; читаем строку*

**mov** **[**strPointer**]**, **eax**

cinvoke printf, strFormatInputStr, **[**strPointer**]** *; выводим исходную строку*

ccall ReverseString, strPointer *; переворачиваем строку на месте*

cinvoke printf, strFormatResultStr, **[**strPointer**]** *; выводим результат (перевёрнутая строка)*

cinvoke getch

invoke ExitProcess, 0

*; char\* ReadString(); -- читает строку (до перевода строки) с использованием WinApi*

proc ReadString c uses **edi**

locals

s **dd** 0

s\_size **dd** 0 *; изначально строка пустая*

handle **dd** 0

endl

cinvoke \_\_iob\_func *; полуаем FILE\* stdin*

**mov** **[**handle**]**, **eax**

cinvoke calloc, **[**s\_size**]** *; выделяем память под строку*

**mov** **[**s**]**, **eax**

**mov** **ecx**, -1

.readLoop:

**mov** **eax**, **[**s\_size**]** *; считываем размер*

**add** **eax**, 1024 *; увличиваем буффер на 1024 символа*

**mov** **[**s\_size**]**, **eax** *; сохраняем новый размер*

cinvoke realloc, **[**s**]**, **eax** *; увлечиваем строку*

**mov** **[**s**]**, **eax** *; сохраняем указатель на начало строки*

**mov** **ecx**, **[**s\_size**]**

**lea** **edi**, **[eax** + **ecx** - 1024**]** *; edi -- указатель на символ, начиная с котрого мы считываем новую часть строки*

cinvoke fgets, **edi**, 1025, **[**handle**]** *; указываем 1025, чтобы fgets прочитал 1024 символа*

**cmp** **eax**, 0 *; проверяем на налчие ошибок*

**je** .endRead *; если ничего не удалось прочитать, то выходим*

**mov** **ecx**, 1024

**mov** **al**, 10 *; символ перевода строки*

*; edi всё ещё указывает на символ, начиная с которого мы читали*

**repnz** **SCASB**

*; если zf = 1, значит мы встретили перевод строки, edi указывает на символ после него, тогда выходим*

**LOOPNE** .readLoop *; zf = 0*

.endLoop: *; edi указывает на \n, s указывает на символ после последнего*

**dec** **edi** *; возвращаем edi на символ перевода строки*

**mov** **ebx**, **[**s**]**

**cmp** **ebx**, **edi** *; провряем, вдруг пользователь ввёл просто перевод строки (без \r)*

**je** .endRead

**dec** **edi** *; смотрим на предыдущий символ*

**cmp** **byte** **[edi]**, 13 *; проверяем, что это \r*

**je** .endRead *; если да, то обрезаем строку в этом месте*

**inc** **edi** *; если нет, то возвращаемся обратно на \n*

.endRead: *; edi указывает на символ после последнего*

**mov** **byte** **[edi]**, 0 *; устанавливаем нуль-терминатор*

**mov** **eax**, **[**s**]** *; возвращаем указатель на строку*

**ret**

endp

*; void ReverseString(char\* str) -- переворачивает строку "на месте"*

proc ReverseString c uses **esi** **edi** **ebx**, \

stringPointer: **DWord**

**mov** **esi**, **[**stringPointer**]**

**mov** **esi**, **[esi]** *; помещаем в esi и edi указатель на начало строки*

**mov** **edi**, **esi**

**mov** **ecx**, -1 *; мы не знаем сколько символов в строке*

**xor** **al**, **al** *; al = 0*

**cld**

**repne** **SCASB**

**sub** **edi**, 2

**neg** **ecx**

**sub** **ecx**, 2

**je** .**ret** *; если строка оказалась пустая, то выходим*

*; в итоге в edi лежит указатель на последний символ строки, а в ecx количество символов (без нуль терминатора)*

**shr** **ecx**, 5 *; делим на 32, потому что мы будем с использованием SSE3 поворачивать*

*; по 16 символов за раз в начале и в конце, то есть в сумме за итерацию 32 символа*

**je** .simpleswap *; если ecx равен 0, то переходим к повороту строки без использования SSE*

*; загружаем в xmm2 нужный нам порядок замены байт*

**mov** **eax**, shuffleOrder

movups **xmm2**, **[eax]**

**sub** **edi**, 15 *; сдвигаемся на 15, потому что мы двигаем по 16 за раз*

.**loop**:

*; загружаем в xmm начало и конец*

movups **xmm0**, **[esi]**

movups **xmm1**, **[edi]**

*; поворачиваем начало и конец строки*

*; pshufb -- sse3 инструкция, которая меняет порядок байт в 128й битном*

*; слове src на основе порядка, указанного в dst*

pshufb **xmm0**, **xmm2**

pshufb **xmm1**, **xmm2**

*; вставляем конец в начало, а начало в конец*

movups **[esi]**, **xmm1**

movups **[edi]**, **xmm0**

*; сдвигаем начало и конец*

**add** **esi**, 16

**sub** **edi**, 16

**LOOP** .**loop**

**add** **edi**, 15 *; возвращаем лишний сдвиг*

.simpleswap: *; простой разворот строки по байту за раз*

**cld** *; на всякий случай очищаем флаг направления*

**mov** **ecx**, **edi** *; считаем число операций ecx = (edi - esi + 1) / 2*

**inc** **ecx**

**sub** **ecx**, **esi**

**shr** **ecx**, 1

.loop2:

**mov** **bl**, **[edi]** *; в bl сохраняем значение из edi*

**movsb** *; перемещаем символ из esi в edi, после этого esi++, edi++*

**sub** **edi**, 2 *; сдвигаем edi в обратную сторону*

**mov** **[esi** - 1**]**, **bl** *; записываем в esi ранее сохранённый символ*

**LOOP** .loop2

.**ret**:

**ret**

endp

*;-------------------------------third act - including HeapApi--------------------------*

**section** '.idata' **import** **data** readable

library kernel, 'kernel32.dll',\

msvcrt, 'msvcrt.dll',\

user32,'USER32.DLL'

include 'api\user32.inc'

include 'api\kernel32.inc'

**import** kernel,\

ExitProcess, 'ExitProcess',\

HeapCreate,'HeapCreate',\

ReadConsoleA,'ReadConsoleA',\

GetStdHandle,'GetStdHandle',\

HeapAlloc,'HeapAlloc'

include 'api\kernel32.inc'

**import** msvcrt,\

fgets, 'fgets', \

malloc, 'malloc', \

calloc, 'calloc', \

realloc, 'realloc', \

strlen, 'strlen', \

printf, 'printf',\

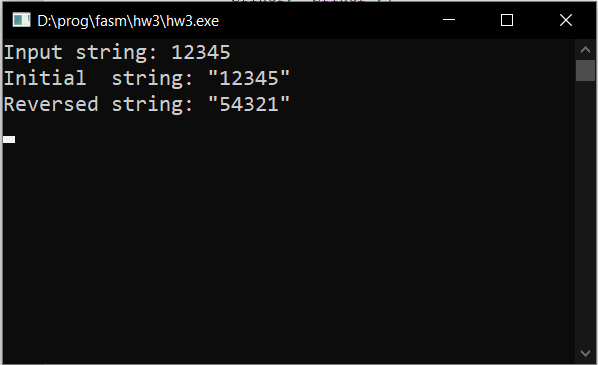
scanf, 'scanf',\

\_\_iob\_func, '\_\_iob\_func',\

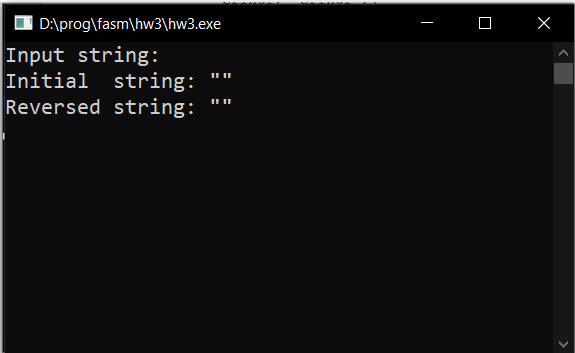
getch, '\_getch'

**4. Тестирование**

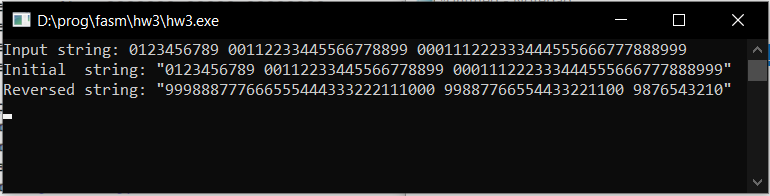
4.1. Простой тест



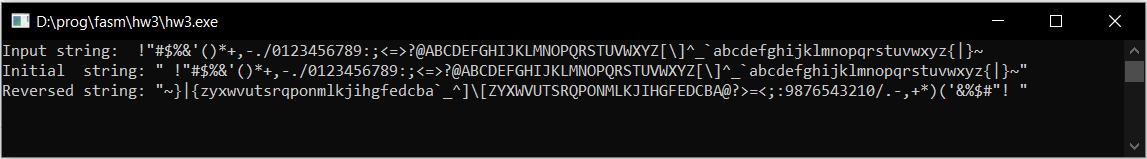
4.2. Пустая строка



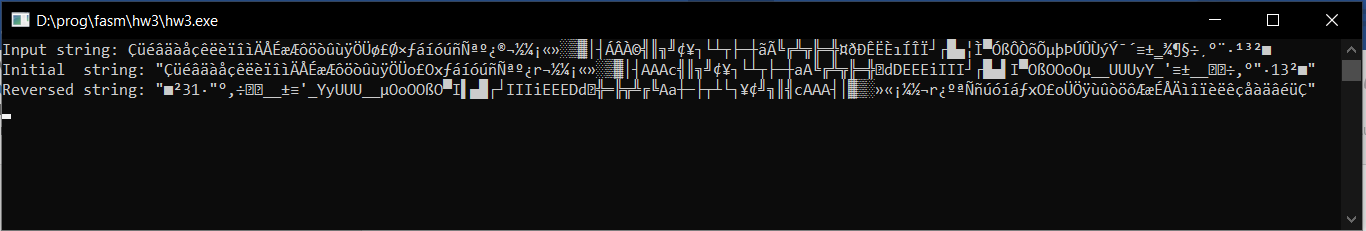
4.3 Длинная строка (проверяем работает ли разворот с помощью **SSE3**)



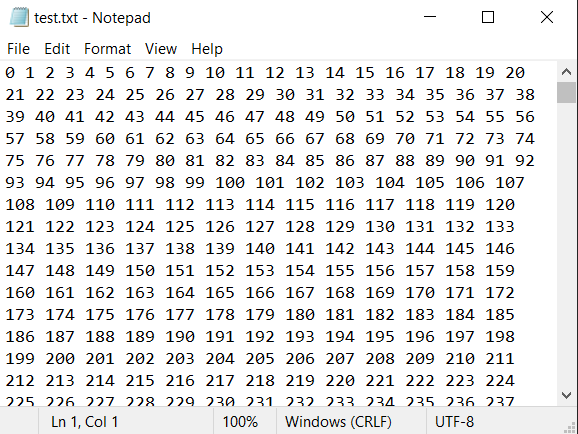
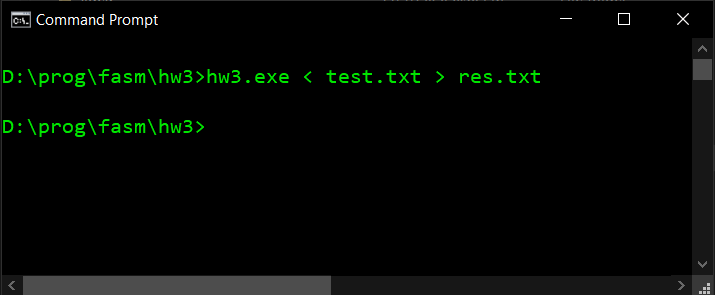
4.4 Все ASCII символы

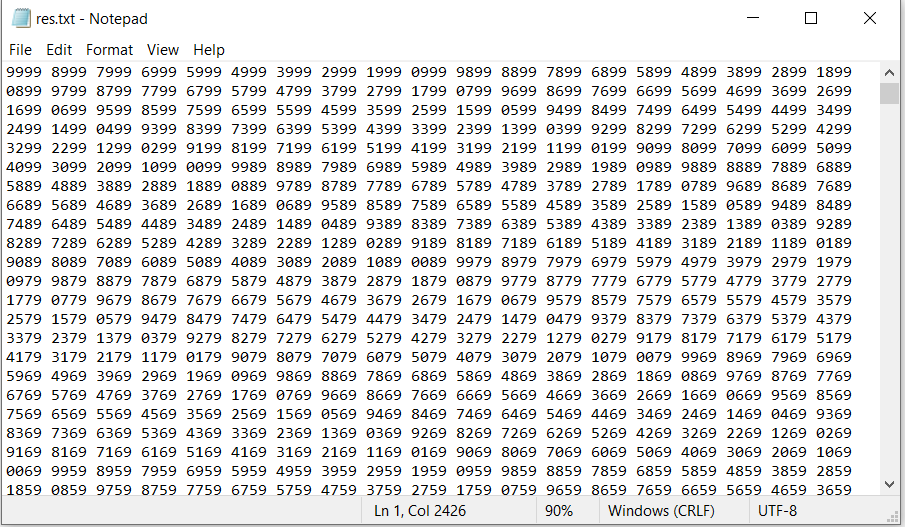


4.5. Дополнительные ASCII символы



4.6. Очень длинная строка (все числа от 0 до 9999)





**Источники**

[1] https://en.cppreference.com/w/c/io/fgets

[2] <https://en.cppreference.com/w/c/memory/realloc>

[3] http://www.club155.ru/x86cmd/SCAS

[4] <http://www.club155.ru/x86cmd/REP>

[5] https://www.felixcloutier.com/x86/pshufb

[6] <http://www.club155.ru/x86cmdsimd/MOVUPS>

[7] http://www.club155.ru/x86cmd/MOVSB