

Оглавление

Описание задачи:	3
Описание решения:	3
Документация:	4
Описание работы программы:	6
Блок схема программы:	7
Проверка работоспособности решения:	8
Требования к сборке решения:	8

Описание задачи:

Задача: Разработать решение, на языке Ассемблер, для перевода входной строки формата ASCII в число формата short integer.

Техническое задание:

1. Входная строка задаётся динамически через консоль пользователем.
2. Отформатированные данные (т.е. число со знаком) записывается в особое место в памяти.
3. Строка может содержать как положительное, так и отрицательное число.
4. Дробные числа не встречаются.

Пример входных данных:	Число на выходе должно иметь вид:
-21	FFFFFFEB
-337	FFFFEAF
334	014E

Описание решения:

Язык программирования: Ассемблер

Компилятор языка: Nasm x86¹

Линковщик: GoLink²

Система: Windows 32 bit

Необходимые библиотеке на устройстве пользователя: user32.dll, kernel32.dll

Компилятор Nasm x86 выбран по следующим причинам:

1. Совместимость с целевой системой
2. Открытый исходный код
3. Простой синтаксис

Линковщик GoLink выбран по следующим причинам:

1. Открытый исходный код
2. Наличие документации

¹ [Index of /pub/nasm/releasebuilds/3.01rc9/win32](https://pub.nasm-releasebuilds.com/builds/3.01rc9/win32/)

² [Go Tools for Windows \(assembler, resource compiler, linker, debugger and information\)](https://go.tools/for/windows/assembler/resource/compiler/linker/debugger/information/)

Система Windows 32 bit была выбрана по следующим причинам:

1. Доля рынка ноутбуков и персональных компьютеров у систем Windows составляет 71% (по данным википедии)³
2. Программы разработанные под 32 битную архитектуру корректно запускаются на системах 64 бита

Документация:

Полный код программы с метками (желтым цветом) приведен на странице .

Описание элементов кода по номерам меток (в коде желтым цветом):

1. Объявление дескриптора консоли.
2. Начало макроса input с нулем входных данных.
3. Enter 0,0 позволяет выполнить участок кода без изменения значения регистров и флагов.
4. Вызов функции чтения консоли.
5. Закрытие конструкции enter 0,0
6. Завершение макроса
7. Переменная, используемая для определения есть ли знак у числа.
8. Секция резервирования данных, без их объявления.
9. Резервирование пространства для отформатированного числа.
10. Метка старта.
11. extern подключает внешние функции. В этом случае: функция получения дескриптора, функция чтения консоли, функция завершения программы.
12. Само получения дескриптора консоли в начале программы.
13. Вызов макроса на чтение данных из консоли.
14. Проверка первого символа строки.
15. Метка показывающая, что первый символ строки (знак) пропущен.
16. Завершение программы.
17. Пропуск первого символа строки, если это минус.
18. Метка проверки, нужно ли добавлять к числу минус.
19. Метка, что со знаком все в порядке
20. Процедура перевода строки в число (работает только со строками, содержащими числа).

Такое большое количество меток обусловлено возможностью изменения решения под нужды конкретно проекта. Создав такой код, была оставлена возможность редактирования участков кода и добавления новых возможностей на разных этапах программы.

³ [Usage share of operating systems - Wikipedia](#)

Код программы:

```
stdin equ -10 | 1
%macro input 0 | 2
    enter 0,0 | 3
    push 0
    push InputLength
    push 256
    lea eax, [InputBuffer]
    push eax
    push dword [InputHandle]
    call _ReadFile@20 | 4
    leave | 5
%endmacro | 6
section .data
    sig db 0 | 7
section .bss | 8
    InputHandle resd 1
    InputBuffer resd 256
    InputLength resd 1
    Number resd 1 | 9
section .text
    global Start | 10
    extern _GetStdHandle@4, _ReadFile@20, _ExitProcess@4 | 11
Start:
    push stdin
    call _GetStdHandle@4
    mov dword [InputHandle], eax | 12
    input | 13
    mov edx, InputBuffer
    mov cl, [edx]
    cmp cl, '-' | 14
    je signed
    sign_deleted: | 15
    call string_to_number
    call num
    call _ExitProcess@4 | 16
signed: | 17
    inc eax
    inc edx
    inc [sig]
    jmp sign_deleted
num: | 18
    cmp [sig], 0
    je sign_ok
    neg eax
    sign_ok: | 19
    mov [Number], eax
    call _ExitProcess@4
string_to_number: | 20
    xor eax, eax
.top:
    movzx ecx, byte [edx]
    inc edx
    cmp ecx, '0'
    jb .done
    cmp ecx, '9'
    ja .done
    sub ecx, '0'
    mul eax, 10
    add eax, ecx
    jmp .top
.done:
    ret
```

Описание работы программы:

Программа получает дескриптор консоли и начинает считывать с неё данные. Пользователь вводит число (отрицательное или положительное). Код пропускает минус у строки, если он есть, и переводит строку в число. После этого число записывается в зарезервированное пространство в памяти (если у строки был минус, то число инвертируется и становится отрицательным).

Полный код программы занимает 1119 байт дискового пространства. Собранный программа для тестирования и отладки решения занимает 2560 байт дискового пространства.

Репозиторий: <https://github.com/sowlKALI/lab2.git>



Рисунок 1. QR-код ссылка на репозиторий с файлами.

Блок схема программы:

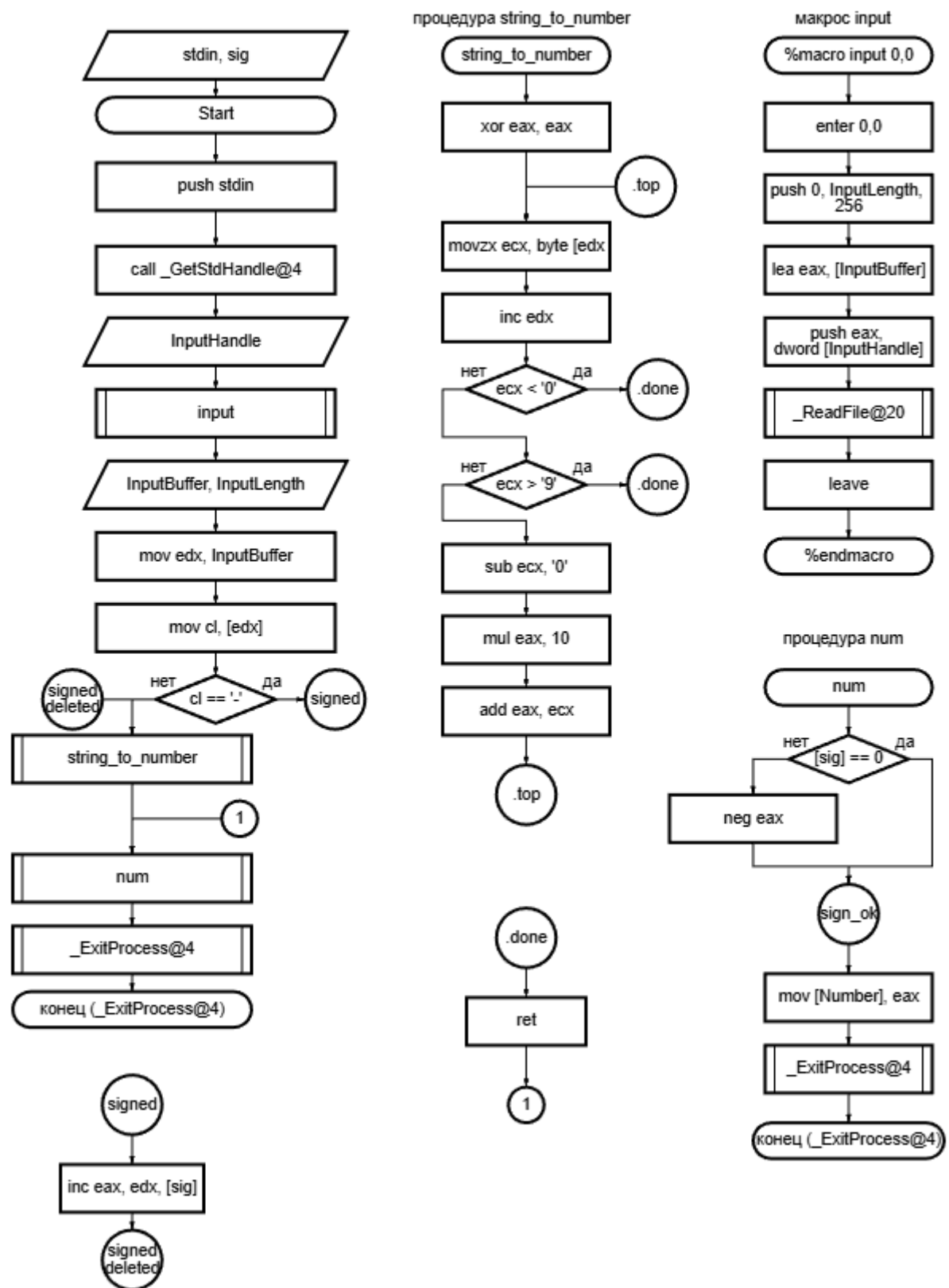
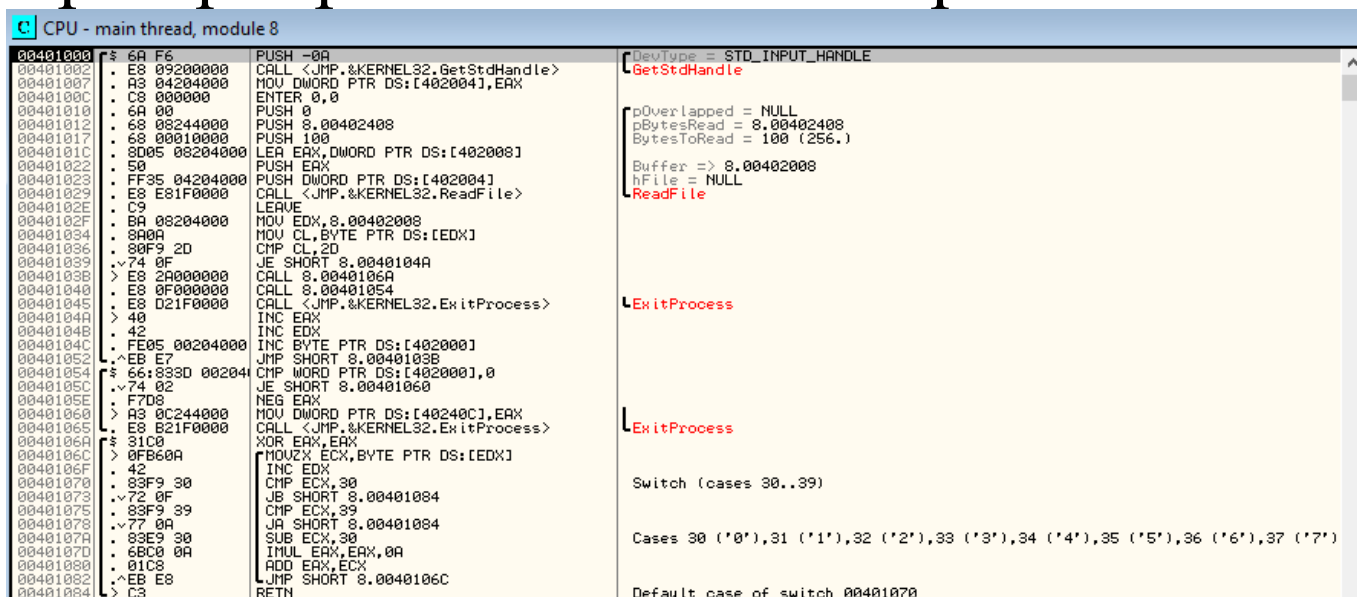


Рисунок 2. Блок схема программы.

Проверка работоспособности решения:



```
CPU - main thread, module 8
00401000 6A F6 PUSH -0A
00401002 E8 09200000 CALL <JMP.&KERNEL32.GetStdHandle>
00401007 A3 04204000 MOV DWORD PTR DS:[402004],EAX
0040100C C8 000000 ENTER 0,0
00401010 6A 00 PUSH 0
00401012 68 08244000 PUSH 8.00402408
00401017 68 00010000 PUSH 100
0040101C 8D05 08204000 LEA EAX,DWORD PTR DS:[402008]
00401022 50 PUSH EAX
00401023 FF35 04204000 PUSH DWORD PTR DS:[402004]
00401029 E8 E81F0000 CALL <JMP.&KERNEL32.ReadFile>
0040102E C9 LEAVE
0040102F BA 08204000 MOV EDX,8.00402008
00401034 8A0A MOV CL,BYTE PTR DS:[EDX]
00401036 80F9 2D CMP CL,2D
00401039 74 0F JE SHORT 8.0040104A
0040103B E8 2A000000 CALL 8.0040106A
00401040 E8 0F000000 CALL 8.00401054
00401045 E8 D21F0000 CALL <JMP.&KERNEL32.ExitProcess>
0040104A 40 INC EAX
0040104B 42 INC EDX
0040104C FE05 00204000 INC BYTE PTR DS:[402000]
00401052 EB E7 JMP SHORT 8.0040105B
00401054 66:833D 00204000 CMP WORD PTR DS:[402000],0
0040105C 74 02 JE SHORT 8.00401060
0040105E F7D8 NEG EAX
00401060 A3 0C244000 MOV DWORD PTR DS:[40240C],EAX
00401065 E8 E21F0000 CALL <JMP.&KERNEL32.ExitProcess>
0040106A 31C0 XOR EAX,EAX
0040106C 0FB60A MOVBX ECX,BYTE PTR DS:[EDX]
0040106F 42 INC EDX
00401070 83F9 30 CMP ECX,30
00401073 72 0F JB SHORT 8.00401084
00401075 83F9 39 CMP ECX,39
00401078 77 0A JA SHORT 8.00401084
0040107A 83E9 30 SUB ECX,30
0040107D 68C0 0A IMUL EAX,EAX,0A
00401080 01C8 ADD EAX,ECX
00401082 EB E8 JMP SHORT 8.0040106C
00401084 C3 RETN

Device = STD_INPUT_HANDLE
GetStdHandle

pOverlapped = NULL
pBytesRead = 8.00402408
BytesToRead = 100 (256.)
Buffer => 8.00402008
hFile = NULL
ReadFile

ExitProcess

ExitProcess

Switch (cases 30..39)

Cases 30 ('0'),31 ('1'),32 ('2'),33 ('3'),34 ('4'),35 ('5'),36 ('6'),37 ('7')

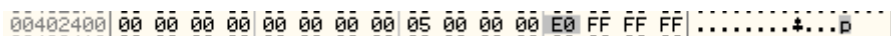
Default case of switch 00401070
```

Рисунок 3. Снимок решения в отладчике.



```
D:\assembler\8.exe
-32
```

Рисунок 4. Входная строка (вводится пользователем с клавиатуры).



```
00402400 00 00 00 00 00 00 00 00 05 00 00 00 E0 FF FF FF .....-32
```

Рисунок 5. Дамп памяти, выделенной под хранение числа. (FFFFFFE0 = -32).

Таблица с тестовым набором:

Пример входных данных:	Число на выходе должно иметь вид:
-21	FFFFFFFEB
-32	FFFFFFFE0
-337	FFFFFFEAF
334	014E

Требования к сборке решения:

1. Nasm x86 (рекомендованная версия: 3.01rc09)
2. GoLink (рекомендованная версия: 1.0.4.6)
3. Наличие стандартных библиотек Windows: user32.dll и kernel32.dll

Команды сборки:

1. *nasm -f win32 lab1.asm -o lab1.obj*
2. *golink /entry:Start /console kernel32.dll user32.dll lab1.obj*