



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования  
«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

## ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент \_\_\_\_\_  
*фамилия, имя, отчество*

Группа \_\_\_\_\_

Тип практики \_\_\_\_\_

Название предприятия \_\_\_\_\_

Студент \_\_\_\_\_  
*подпись, дата* *фамилия, и.о.*

Рекомендуемая оценка: \_\_\_\_\_  
Руководитель практики от предприятия: \_\_\_\_\_  
*подпись, дата* *фамилия, и.о.*

Руководитель практики \_\_\_\_\_  
*подпись, дата* *фамилия, и.о.*

Оценка \_\_\_\_\_

2025 г.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1. Характеристика предприятия.....	4
2. Реализация.....	5
2.1. Знаковое сложение 32-битных чисел с плавающей точкой.....	6
2.1.1. Сложение чисел с одинаковым знаком.....	6
2.1.2. Сложение чисел с разными знаками.....	6
2.2. Знаковое умножение двух чисел с плавающей точкой.....	7
2.3. Процедура разбора числа с плавающей точкой из строки.....	8
2.4. Вывод числа с плавающей точкой на экран.....	9
3. Демонстрация работы программы.....	10
3.1. Сложение чисел, с конечным количеством цифр после точки.....	10
3.2. Представление числа с бесконечным числом чисел после запятой.....	10
3.3. Умножение чисел без потери точности.....	10
3.4. Умножение с потерей точности.....	11
4. Вывод.....	12
5. Приложение.....	13

## ВВЕДЕНИЕ

В рамках производственной практики было дано задание разработать программу на языке «ассемблер» под платформу DOS, способную эмулировать работу 32-битных чисел с плавающей точкой, с поддержкой их чтения с клавиатуры, операций знакового сложения и умножения и вывода на экран.

Для демонстрации работы эмулятора, также необходимо написать программу, считывающую два числа с плавающей точкой от пользователя, выполняющая их сложение и выводящая результат сложения на экран.

## 1. Характеристика предприятия

НАМИ (Научно-исследовательский автомобильный и автомоторный институт) — ведущий государственный научный центр России, который занимается исследованиями, проектированием и испытаниями автотранспортной техники и двигателей. Созданный еще в 1918 году, этот институт стоял у истоков отечественного автомобилестроения, обеспечив теоретическую и технологическую базу для появления в стране собственных машин, тракторов и силовых агрегатов.

Сегодня НАМИ осуществляет полный цикл работ: от фундаментальных научных исследований до создания опытных и серийных образцов транспортных средств, организации испытаний, сертификации продукции и контроля ее эксплуатации. Институт располагает крупнейшим в Европе автополигоном, позволяющим комплексно проверять новые модели и технологии.

НАМИ активно внедряет инновационные решения, такие как электромобили, гибридные двигатели, беспилотные транспортные средства и проекты на альтернативных видах топлива. Большинство современных и исторически значимых отечественных автомобилей, в том числе знаковые модели и даже премиальные разработки для правительства (например, Aurus), появились при участии инженеров и ученых института.

Важной частью работы НАМИ является взаимодействие с российскими промышленными предприятиями и формирование современной базы компонентной отрасли. Институт также представляет Россию в ряде международных технических комитетов, занимается вопросами стандартизации, испытаний продукции по мировым требованиям и активно участвует в развитии транспортных технологий на государственном уровне.



представления для нуля — положительного и отрицательного, в зависимости от значения бита, отвечающего за знак числа.

Также, экспонента представлена в сдвинутом на 127 виде. Например, для представления  $2^3$  в экспоненте будет находиться число  $3 + 127 = 130$ .

## 2.1. Знаковое сложение 32-битных чисел с плавающей точкой

Реализация процедуры знакового сложения представлена на листинге 3.

### 2.1.1. Сложение чисел с одинаковым знаком

Для сложения двух чисел с плавающей точкой, производят следующие действия:

1. Узнают самое старшее по модулю число.
2. Выносят экспоненту старшего числа за скобки. Перед младшим теперь гарантированно экспонента в неположительной степени.
3. Умножают младшее число на экспоненту перед ним. Замечу, что при таком умножении младшая мантисса может либо уменьшиться либо не измениться. Получились такие условия на мантиссы:  
 $1_{10} \leq \text{Старшая мантисса} < 2_{10}$  на старшую и  $0_{10} < \text{Младшая мантисса} < 2_{10}$  на младшую.
4. Складывают мантиссы.
5. Из соотношений на мантиссы выводится формула:  
 $1_{10} \leq \text{Старшая мантисса} + \text{Старшая мантисса} < 4_{10}$ . Значит, если результат превысил два, то его делят на два и прибавляют к экспоненте результата единицу.
6. Комбинируют экспоненту и получившуюся мантиссу.

### 2.1.2. Сложение чисел с разными знаками

При вычитании выполняются следующие действия:

1. Узнают самое старшее по модулю число. Его знак и будет знаком результата.
2. Выносят экспоненту старшего числа за скобки. Перед младшим теперь

гарантированно экспонента в неположительной степени.

3. Умножают младшее число на экспоненту перед ним.
4. Вычитают из старшей мантиссы младшую.
5. Если результат разности мантисс оказался меньше единицы, то его домножают на два до тех пор, пока оно не станет большим или равным единицы, попутно каждый раз вычитая единицу из экспоненты
6. Комбинируют экспоненту и получившуюся мантиссу со знаком.

## 2.2. Знаковое умножение двух чисел с плавающей точкой

Умножение двух чисел проводится по следующей процедуре:

1. Экспоненты двух чисел складываются
  1. Из каждой экспоненты вычитается 127
  2. Экспоненты складываются
2. Перемножаются мантиссы. В результате получится число с плавающей точкой.
3. К экспоненте результата прибавляют результат сложения экспонент (пункт 1)
4. Если знаки исходных чисел различаются, то в результате записывается «минус». Иначе - «плюс»

Умножение мантисс производится «в столбик». Пример произведения чисел  $1.01001_2$  и  $1.00101_2$  изображена на рисунке . Так как процедура производится в двоичной системе счисления, то можно заметить, что при умножении одного числа на другое, первое число складывается само с собой, сдвинутым на несколько бит влево. Сдвиги же определяются установленными единицей битами второго числа. Процедура `float_mul` представлена в листинге 4.

$$\begin{array}{r}
 1.01001 \\
 \times 1.00101 \\
 \hline
 101001 \\
 101001 \\
 101001 \\
 \hline
 1.0111101101
 \end{array}$$

Рисунок 2: процедура умножения двух мантисс в двоичной системе счисления

### 2.3. Процедура разбора числа с плавающей точкой из строки

На вход процедуре `parse_float` подаётся

1. длина строки
2. указатель на строку

По ходу разбора строки строится целая часть заданного числа. Если на текущей итерации нашлась не цифра, а символ точки «.», то вызывается вспомогательная процедура `parse_decimal` (листинг 2), разбирающая дробную часть числа и возвращающая мантиссу.

Есть один важный момент: если целая часть поданного числа была равна нулю, то вызываемая процедура `parse_decimal` должна будет сделать целую часть единицей, путем домножений мантиссы на два, попутно вычитая каждый раз единицу из экспоненты. Процесс домножения на два останавливается после встречи первой единицы (первая единица не учитывается при разборе мантиссы так как именно она сделала целую часть единицей). Для этого процедуре нужно подать на вход:

1. разбираемую строку, содержащую только дробную часть числа;
2. длину этой строки
3. флаг того, нужно ли «исправлять» целую часть
4. Указатель на экспоненту, откуда вычитать при исправлении целой части

Процедура `parse_decimal` устроена следующим образом: сначала из



каждого байта строки, в которой находится дробная часть числа, вычитается символ '0', чтобы в каждом байте стояло число от нуля до девяти. Затем, если требуется сделать целую часть единицей, нужно пропустить первые нули дробной части, одновременно вычитая из экспоненты по единице за каждый пропущенный ноль. После, чего, дробная часть начинается процедура получения двоичного представления дробной части из десятичной. Это делается путем умножения дробной десятичной части на 2. Так получается первая цифра после запятой в двоичном представлении. Чтобы получить следующую цифру, умножим результат предыдущей итерации на 2. Продолжать выполнение нужно 23 раза, по размеру мантииссы. Или пока дробная часть в десятичном представлении не станет равна нулю.

После получения мантииссы и целой части, нужно скомбинировать результат. Для этого целую часть необходимо делить на два до тех пор пока она не окажется равна единице, прибавляя при каждом делении к экспоненте единицу. Также, при таком делении нужно приписывать вперед к мантииссе последний бит из целой части. При такой процедуре, например, из числа  $11.010_2 \cdot 2^0$  получится  $1.1010_2 \cdot 2^1$ .

Далее, путем битовых операций, мантиисса комбинируется с экспонентой, получая 32-битное число с плавающей точкой.

## 2.4. Вывод числа с плавающей точкой на экран

Вывод на экран устроен следующим образом: сначала получается целая часть от поданного числа, которая первой выводится на экран. Затем, эта целая часть преобразуется к числу с плавающей точкой и вычитается из исходного числа, вычлняя дробную часть. Затем процедура получения дробной части в десятичной системе счисления из дробной части двоичной системы схожа с той, используемой в процедуре `parse_decimal`: множим дробную часть на 10 с помощью `float_mul`, вычлняем целую часть из результата и распечатываем её на экран. Такой алгоритм выполняется столько раз, сколько нужно чисел после запятой. Процедура `float_display` представлена в листинге 5.

### 3. Демонстрация работы программы

#### 3.1. Сложение чисел, с конечным количеством цифр после точки

При выполнении сложения не ожидается что возникнет потеря точности (при условии, что дробная часть в двоичной системе счисления поместится в выделенные 23 бита под мантиссу). Проверим на числе  $2.1640625_{10} = 10.0010101_2$  и числе  $7.2890625_{10} = 111.0100101_2$ . В сумме должно выйти  $9.453125_{10}$ . Так и вышло. Результат представлен на рисунке 3.

```
D:\>D:\test
2.1640625
7.2890625
9.4531250000
```

Рисунок 3: результат сложения  
без потери точности

#### 3.2. Представление числа с бесконечным числом чисел после запятой

Чтобы увидеть потерю точности, достаточно сложить такое число с нулем и посмотреть на вывод программы. Возьмем число  $0.1_{10} = 0.0(0011)_2$ . Получилось число  $0.9999999904_{10}$  при выводе первых десяти чисел после запятой. Вывод программы представлен на рисунке 4.

```
D:\>test
0.1
0
0.09999999904
```

Рисунок 4: демонстрация потери  
точности

#### 3.3. Умножение чисел без потери точности

Для демонстрации умножения без потери точности возьмем числа с конечным числом нулей после запятой:  $100000_{10} = 11000011010100000_2$  и

$0.25_{10}=0.01_2$  . В результате получится  $25000_{10}=110000110101000_2$  . Вывод программы показан на рисунке 5.

```
D:\>test
1000000
0.25
25000.000000000000
```

Рисунок 5: умножение без потерь  
точности

### 3.4. Умножение с потерей точности

Для демонстрации потери точности, возьмем число  $0.1_{10}=0.0(0011)_2$  и  $2_{10}=10_2$  . Действительно, из-за ограниченности размера мантиссы потеря точности неизбежна. В результате ожидается число, которое лишь приближенно равно  $0.2_{10}$ . Вывод программы показан на рисунке .

```
D:\>test
2
0.1
0.19999999809
```

Рисунок 6: умножение с потерей  
точности

#### 4. Вывод

В рамках производственной практики была реализована программа эмулятор чисел с плавающей точкой. Были реализованы операции знакового сложения чисел, а также их знакового умножения. Были реализованы функции разбора чисел с плавающей точкой из строки, а также их вывода на экран.

## 5. Приложение

Ссылка на репозиторий с код данной работы:

<https://github.com/Blackdeer1524/float32-8086>

Листинг 1: процедура float\_parse

```
1 float_parse proc ; (uint16 len, char *str)
2     push ebp
3     mov ebp, esp
4
5     sub ESP, 12;
6     sign equ dword ptr [EBP - 4]
7     mov sign, 0 ; sign data
8
9     buffer EQU dword ptr [EBP - 8]
10    mov buffer, 0 ; mantissa buffer
11    exp_from_mantissa equ dword ptr [EBP - 12]
12    mov exp_from_mantissa, 0
13
14    ; callee-safe registers
15    push bx
16    push si
17    push di
18
19    ; subroutine body
20    len EQU di
21    mov len, word ptr [ebp + 6]
22
23    str_ptr EQU bx
24    mov str_ptr, word ptr [ebp + 6 + 2] ; string ptr
25
26    xor eax, eax
27    whole_part EQU EAX
28    whole_part_l EQU AL
29
30    xor si, si
31
32    cmp byte ptr [str_ptr], '-'
33    jne _after_sign_check
34    mov sign, 0800000000h ; 1 << 31
35    inc si
36
37    _after_sign_check:
38    exponent EQU EDI
39    xor exponent, exponent
```

```

40
41     mantissa EQU ECX
42     xor mantissa, mantissa
43
44 _loop:
45     cmp si, len
46     je _check_for_value_triviality
47
48     cmp byte ptr [str_ptr + si], '.'
49     je _found_dot
50
51     cmp byte ptr [str_ptr + si], '0'
52     jl _error
53     cmp byte ptr [str_ptr + si], '9'
54     jg _error
55
56     imul whole_part, whole_part, 10
57     add whole_part_l, byte ptr [str_ptr + si]
58     sub whole_part_l, '0'
59
60     inc si
61     jmp _loop
62
63 _found_dot:
64     inc si ; skipping the dot
65
66     push whole_part
67
68     sub ebp, 12 ; exp_from_mantissa's address
69     push ebp
70     add ebp, 12
71
72     cmp whole_part, 0
73     jle _whole_part_is_empty
74     push 0
75     jmp _done_whole_part_cmp
76 _whole_part_is_empty:
77     push 1
78     jmp _done_whole_part_cmp
79 _done_whole_part_cmp:
80
81     add str_ptr, si
82     push str_ptr
83     sub len, si
84     push len
85     call parse_decimal
86     add esp, 10

```

```

87
88     mov exponent, exp_from_mantissa
89
90     mov mantissa, eax
91     pop whole_part
92
93     _check_for_value_triviality:
94         cmp whole_part, 0
95         jne _build_float
96
97         cmp exponent, 0
98         je _check_mantissa_triviality
99         mov whole_part, 1
100        jmp _build_float
101
102    _check_mantissa_triviality:
103        cmp mantissa, 0
104        je _value_is_zero
105        mov whole_part, 1
106        jmp _build_float
107
108    _value_is_zero:
109        mov eax, 0
110        jmp _epilogue
111
112    _build_float:
113
114    _loop2:
115        cmp whole_part, 1
116        je _loop2_end
117
118        inc exponent
119
120        mov buffer, whole_part
121        and buffer, 1
122        shl buffer, 31
123
124        shr mantissa, 1
125        or mantissa, buffer
126
127        shr whole_part, 1
128        jmp _loop2
129
130    _loop2_end:
131        add exponent, 127
132        shl exponent, 24
133        shr exponent, 1

```

```

134
135     or exponent, sign
136     shr mantissa, 9
137     or mantissa, exponent
138
139 _epilogue:
140     mov EAX, mantissa
141
142     ; restoring registers
143     pop di
144     pop si
145     pop bx
146
147     mov esp, ebp
148     pop ebp
149     ret
150
151 _error:
152     exit_with_message err_unexpected_chr
153 float_parse endp

```

## Листинг 2: процедура parse\_decimal

```

1  parse_decimal proc ; (uint16 len, char [data *] str, uint16
skip, uint32 [stack *] exponent)
2      push ebp
3      mov ebp, esp
4
5      sub ESP, 2
6
7      push bx
8      push si
9      push edi
10     ; subroutine body
11
12     xor eax, eax ; mantissa
13
14     len EQU word PTR [EBP + 6]
15
16     str_ptr EQU bx
17     mov str_ptr, WORD PTR [EBP + 6 + 2] ; str_ptr. points
after a dot symbol
18
19     still_skipping_flag equ word ptr [ebp + 6 + 4]
20
21     exponent_ptr equ EDI

```



```

22     mov exponent_ptr, DWORD PTR [EBP + 6 + 6]
23     mov dword ptr [exponent_ptr], 0
24
25     MAX_MANTISSA_SIZE = 23
26     cmp len, MAX_MANTISSA_SIZE
27     jle _mantissa_is_at_most_23
28         mov len, MAX_MANTISSA_SIZE
29
30     _mantissa_is_at_most_23:
31         xor si, si
32     _normalize_loop:
33         cmp si, len
34         jge _normalize_loop_end
35
36         cmp byte ptr [str_ptr + si], '0'
37         jl _error
38
39         cmp byte ptr [str_ptr + si], '9'
40         jg _error
41
42         sub byte ptr [str_ptr + si], '0'
43         inc si
44         jmp _normalize_loop
45     _normalize_loop_end:
46
47     has_decimal_part EQU byte ptr [EBP - 1]
48     iteration_count EQU byte ptr [EBP - 2]
49     mov iteration_count, 0
50
51     mov cl, 31
52     _decimal_part_outer_start:
53         cmp iteration_count, MAX_MANTISSA_SIZE
54         je _decimal_part_outer_end
55
56         inc iteration_count
57
58         mov si, len
59         dec si
60
61         carry equ edx
62         xor carry, carry
63         carry_l equ dl
64
65         mov has_decimal_part, 0
66     _decimal_part_inner:
67         digit EQU byte ptr [str_ptr + si]
68

```

```

69      add carry_l, digit
70      add digit, carry_l ; multiply digit by 2 with a carry
71
72      cmp digit, 10
73      jl _decimal_part_inner_digit_lt_10
74          sub digit, 10
75          mov carry_l, 1
76          jmp _decimal_part_inner_digit_lt_10_done
77      _decimal_part_inner_digit_lt_10:
78          xor carry_l, carry_l
79          jmp _decimal_part_inner_digit_lt_10_done
80
81      _decimal_part_inner_digit_lt_10_done:
82          cmp digit, 0
83          je cmp_done
84          mov has_decimal_part, 1
85      cmp_done:
86
87          cmp si, 0
88          je _decimal_part_inner_end
89
90      dec si
91      jmp _decimal_part_inner
92  _decimal_part_inner_end:
93      cmp has_decimal_part, 1
94      jne _no_decimal_part_left
95
96      cmp still_skipping_flag, 1
97      jne _after_still_skipping_flag
98      cmp carry, 0
99      je _carry_cmp_done
100         mov still_skipping_flag, 0
101
102         _carry_cmp_done:
103         dec iteration_count
104         dec dword ptr ss:[exponent_ptr]
105         jmp _decimal_part_outer_start
106
107         _after_still_skipping_flag:
108         shl carry, CL
109         dec CL
110
111         or eax, carry
112         jmp _decimal_part_outer_start
113
114     _no_decimal_part_left:
115         cmp carry, 0

```

```

116         je _decimal_part_outer_end
117
118         cmp still_skipping_flag, 1
119         jne _after_still_skipping_flag_no_decimal
120         dec dword ptr ss:[exponent_ptr]
121         jmp _decimal_part_outer_end
122
123         _after_still_skipping_flag_no_decimal:
124         shl carry, CL
125         dec CL
126
127         or eax, carry
128
129         jmp _decimal_part_outer_end
130 _decimal_part_outer_end:
131     ; epilogue
132     pop edi
133     pop si
134     pop bx
135
136     mov esp, ebp
137     pop ebp
138     ret
139
140 _error:
141     exit_with_message err_unexpected_chr
142 parse_decimal endp

```

### Листинг 3 процедура float\_add

```

1  float_add proc ; (uint32 [float] left, uint32 [float] right)
→ uint32 [float]
2      push ebp
3      mov ebp, esp
4
5      sub esp, 28
6
7      push EBX
8      push EDI
9      push ESI
10
11     left equ EBX
12     right equ EDX
13     buffer equ ECX
14     buffer_l equ CX
15     buffer_ll equ CL
16
17     mov left, dword ptr [EBP + 6]

```

```

18     mov right, dword ptr [EBP + 10]
19
20     float_check_zero left, buffer
21     jne _left_not_trivial
22     mov eax, right
23     jmp _epilogue
24
25 _left_not_trivial:
26     float_check_zero right, buffer
27     jne _not_trivial
28     mov eax, left
29     jmp _epilogue
30
31 _not_trivial:
32     left_sign      equ dword ptr [EBP - 4]
33     left_exponent  equ dword ptr [EBP - 8]
34     left_mantissa  equ dword ptr [EBP - 12]
35
36     right_sign     equ dword ptr [EBP - 16]
37     right_exponent equ dword ptr [EBP - 20]
38     right_mantissa equ dword ptr [EBP - 24]
39
40     float_decompose left, left_sign, left_exponent,
left_mantissa
41     float_decompose right, right_sign, right_exponent,
right_mantissa
42 _init_done:
43     mov EDI, left
44     shl edi, 1
45     shr edi, 1
46
47     mov ESI, right
48     shl esi, 1
49     shr esi, 1
50
51     cmp edi, esi
52     jge _left_exp_ge_right
53     xchg left, right
54
55     exchange_memory left_sign,      right_sign,      buffer
56     exchange_memory left_exponent,  right_exponent,  buffer
57     exchange_memory left_mantissa,  right_mantissa,  buffer
58
59 _left_exp_ge_right:
60     mov buffer, left_exponent
61     sub buffer, right_exponent
62

```

```

63     cmp buffer, 24
64     jle _exp_diff_is_at_most_24
65     mov eax, left
66     jmp _epilogue
67
68 _exp_diff_is_at_most_24:
69     or left_mantissa, 08000000h ; 1 << 23
70     or right_mantissa, 08000000h ; 1 << 23
71
72     shr right_mantissa, cl
73     cmp_memory left_sign, right_sign, buffer
74     jne _diff_signs
75
76 _same_signs:
77     mov eax, left_mantissa
78     add eax, right_mantissa
79     cmp eax, 010000000h ; 1 << 24 = 10 << 23
80     jl _not_greater_2
81     inc left_exponent
82     shr eax, 1
83
84 _not_greater_2:
85     xor eax, 08000000h
86
87     shl left_sign, 31
88     shl left_exponent, 23
89     or eax, left_sign
90     or eax, left_exponent
91     jmp _epilogue
92
93 _diff_signs:
94     mov eax, left_mantissa
95     sub eax, right_mantissa
96     cmp eax, 0
97     je _epilogue
98
99 _while:
100    mov buffer, eax
101    and buffer, 08000000h ; 1 << 23.
102    cmp buffer, 0
103    jg _while_end
104    shl eax, 1
105    dec left_exponent
106    jmp _while
107
108 _while_end:
109     xor eax, 08000000h

```

```

110     shl left_sign, 31
111     shl left_exponent, 23
112     or eax, left_sign
113     or eax, left_exponent
114     jmp _epilogue
115
116 _epilogue:
117     pop ESI
118     pop EDI
119     pop EBX
120
121     mov esp, ebp
122     pop ebp
123     ret
124 float_add endp

```

#### Листинг 4: процедура float\_mul

```

1  float_mul proc ; (uint32 [float] left, uint32 [float] right)
→ uint32 [float]
2      push ebp
3      mov ebp, esp
4
5      sub esp, 28
6
7      push EBX
8      push EDI
9      push esi
10     push edi
11
12     left equ EBX
13     right equ EDX
14     buffer equ ECX
15     buffer_l equ CX
16     buffer_ll equ CL
17
18     mov left, dword ptr [EBP + 6]
19     mov right, dword ptr [EBP + 10]
20
21     float_check_zero left, buffer
22     jne _left_not_trivial
23     mov eax, 0
24     jmp _epilogue
25
26 _left_not_trivial:
27     float_check_zero right, buffer
28     jne _not_trivial

```

```

29     mov eax, 0
30     jmp _epilogue
31
32     _not_trivial:
33         left_sign      equ dword ptr [EBP - 4]
34         left_exponent equ dword ptr [EBP - 8]
35         left_mantissa equ dword ptr [EBP - 12]
36
37         right_sign     equ dword ptr [EBP - 16]
38         right_exponent equ dword ptr [EBP - 20]
39         right_mantissa equ dword ptr [EBP - 24]
40
41         old_left_mantissa equ dword ptr [EBP - 28]
42
43         float_decompose left, left_sign, left_exponent,
left_mantissa
44         float_decompose right, right_sign, right_exponent,
right_mantissa
45
46         or right_mantissa, 08000000h ; 1 << 23
47
48         mov buffer, 03f8000000h
49         or buffer, left_mantissa
50         mov old_left_mantissa, buffer ; 2 ^ 0 * left_mantissa
51
52     _init_done:
53         result_mantissa equ esi
54         mov result_mantissa, 0
55
56         counter equ di
57         xor counter, counter
58     _while:
59         cmp right_mantissa, 0
60         je _while_done
61
62         mov buffer, right_mantissa
63         and buffer, 1
64         cmp buffer, 0
65         je _after_addition
66
67         mov buffer, old_left_mantissa
68
69         push ecx
70         push edx
71         push counter
72         push buffer
73         call float_power_2_mult

```

```

74     add esp, 6
75     pop edx
76     pop ecx
77
78     push ecx
79     push edx
80     push eax
81     push result_mantissa
82     call float_add
83     add esp, 8
84     pop edx
85     pop ecx
86
87     mov result_mantissa, eax
88 _after_addition:
89     shr right_mantissa, 1
90     inc counter
91     jmp _while
92 _while_done:
93     xor buffer_l, buffer_l
94     mov buffer_ll, -23
95     push buffer_l
96     push result_mantissa
97     call float_power_2_mult
98     add esp, 6
99
100    mov buffer, left_exponent
101    sub buffer_ll, 127
102    push buffer_l
103    push eax
104    call float_power_2_mult
105    add esp, 6
106
107    mov buffer, right_exponent
108    sub buffer_ll, 127
109    push buffer_l
110    push eax
111    call float_power_2_mult
112    add esp, 6
113
114    mov buffer, left_sign
115    cmp buffer, right_sign
116    je _epilogue
117    or eax, 080000000h ; 1 << 31
118    jmp _epilogue
119
120 _epilogue:

```



```

121     pop edi
122     pop esi
123     pop EDI
124     pop EBX
125
126     mov esp, ebp
127     pop ebp
128     ret
129 float_mul endp

```

#### Листинг 5: процедура float\_display

```

1  float_display proc ; (uint32 float) → void
2      push ebp
3      mov ebp, esp
4
5      sub esp, 9
6
7      float_10      equ dword ptr [ebp - 4]
8      new_decimal_part equ dword ptr [ebp - 8]
9      iter_count     equ byte ptr [ebp - 9]
10
11     mov eax, 10
12     push eax
13     call int32_to_float
14     mov float_10, eax
15
16     num equ dword ptr [ebp + 6]
17     float_check_positivity num
18     je _float_is_positive
19
20     mov ah, 2
21     mov dx, '-'
22     int 21h
23
24     float_negate num
25
26 _float_is_positive:
27     push num
28     call float_to_int32
29     add esp, 4
30
31     push eax ; save int(num)
32     push eax
33     call print_i32
34     add esp, 4
35     pop eax ; restore int(num)

```

```

36
37     push eax
38     call int32_to_float
39     add esp, 4
40
41     float_negate eax
42     push eax
43     push num
44     call float_add
45     add esp, 8
46
47     decimal_part equ edx
48     mov decimal_part, eax
49
50     push dx
51     mov ah, 2
52     mov dx, '.'
53     int 21h
54     pop dx
55
56     mov iter_count, 0
57     _while:
58         cmp iter_count, 10 ; how many digits to print after the
point
59         je _while_done
60
61         push float_10
62         push decimal_part
63         call float_mul
64         mov new_decimal_part, eax
65         add esp, 8
66
67         push eax
68         call float_to_int32
69         add esp, 4
70
71         push eax
72         mov dx, ax
73         add dx, '0'
74         mov ah, 2
75         int 21h
76         pop eax
77
78         push eax
79         call int32_to_float
80         add esp, 4
81

```

```

82     mov decimal_part, new_decimal_part
83     float_negate eax
84     push eax
85     push decimal_part
86     call float_add
87     add esp, 8
88
89     mov decimal_part, eax
90
91     inc iter_count
92     jmp _while
93 _while_done:
94
95 _epilogue:
96     mov esp, ebp
97     pop ebp
98     ret
99 float_display endp

```