

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Департамент программной инженерии

**Проект по дисциплине «Архитектура вычислительных систем»**

**Программа для вычисления значения функции гиперболического  
синуса от заданного значения на языке ассемблера FASM**

Исполнитель

студент группы БПИ196-1

Махнач Ф. О.

01.11.2020 г.

## Оглавление

1. Текст задания .....	2
2. Теоретическая база .....	2
3. Алгоритм вычисления .....	2
3.1. Код на языке ассемблера FASM.....	3
4. Ограничения, крайние случаи .....	4
5. Особенности программы .....	7
6. Тестирование.....	9
7. Исходный код программы на языке ассемблера FASM.....	12
8. Используемые источники.....	15

## 1. Текст задания

Разработать программу, вычисляющую с помощью степенного ряда с точностью не хуже 0,1% значение функции гиперболического синуса  $sh(x) = (e^x - e^{-x})/2$  для заданного параметра  $x$  (использовать FPU).

Вывод данных следует осуществлять в консоль.

## 2. Теоретическая база

Приближённое значение гиперболического синуса в точке  $x$  можно получить с помощью степенного ряда, который в свою очередь выводится аналитически из степенного ряда функции  $e^x$  [1]. Итого имеем ряд:

$$sh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + \frac{x^{2n-1}}{(2n-1)!} + \dots$$

## 3. Алгоритм вычисления

Заметим, что  $n + 1$ -ое слагаемое ряда можно получить из  $n$ -ого путём домножения на  $\frac{x^2}{(2n)(2n+1)}$  (т. е.  $\frac{x^{2n+1}}{(2n+1)!} = \frac{x^{2n-1}}{(2n-1)!} \cdot \frac{x^2}{(2n)(2n+1)}$ ).

Разумно будет вычислять значение суммы степенного ряда в цикле. Тогда на каждой итерации цикла нам понадобится:

- 1) непосредственно число  $x$  (а точнее, можно сразу вычислить  $x^2$ );
- 2) промежуточное значение суммы (**res**);
- 3) значение предыдущего слагаемого (**term**);
- 4) значение  $m = 2n - 1$  – последнее число, на которое мы делили слагаемое.

Перед первой итерацией мы вычисляем значение  $xSqr = x^2$ , а также устанавливаем значения  $res = x$ ,  $term = x$ ,  $m = 1$ .

На каждой итерации цикла вычисляем новое слагаемое как

$term *= xSqr / ((m + 1) \cdot (m + 2))$

и добавляем к сумме

$res += term$

После этого нам необходимо проверить условие выхода из цикла. В тексте задачи указано, что точность должна быть не менее 0,1%, причём определяется она через разность

полученного значения со значением предыдущей частичной суммы:  $\frac{|res_i - res_{i-1}|}{|res_i|} < 0.001$

(раз речь о процентах, мы по видимому должны разделить на текущее значение).

Нетрудно догадаться, что разность есть просто слагаемое:  $\frac{|term_i|}{|res_i|} < 0.001$

Причём модули также могут быть опущены исходя из того, что  $term$  и  $res$  имеют один знак.

Псевдокод для приведённого алгоритма:

```
x = readFloat()
xSqr = x * x, res = x, term = x, m = 1
while term / res > 0.001:
    term *= xSqr
    term /= (m + 1) * (m + 2)
    m += 2
    res += term
```

На языке ассемблера FASM данные вычисления будут выполняться при помощи FPU (Floating Point Unit). В частности, здесь используется сравнение, сложение, деление и умножение вещественных чисел. Помимо этого, вместо выполнения вычисления  $(m + 1) * (m + 2)$  использована команда `inc` и деление на `m` дважды.

### 3.1. Код на языке ассемблера FASM

Ниже приведён фрагмент кода на FASM, аналогичный псевдокоду выше. Вопрос ввода, вывода, обработки ошибок пока опущен.

```
start:
    FINIT
    ; Ввод числа x
calculation: ; Movf - макрос, перемещающий fp значение из одного адреса в другой
    Movf res, x      ; res = x
    Movf term, x     ; term = x
    Movf xSqr, x     ; xSqr = x * x
    fld [xSqr]
    fmul [x]
    fstp [xSqr]
    mov [lastDenomN], 1 ; lastDenomN = 1
calculation_loop:
    ; Вычисляем слагаемое
    fld [term]
    fmul [xSqr] ; term *= xSqr
    inc [lastDenomN] ; lastDenomN++
    fidiv dword[lastDenomN] ; term /= lastDenomN
```

```

inc     [lastDenomN]          ; lastDenomN++
fdiv    dword[lastDenomN]     ; term /= lastDenomN
fstp    [term]

fld     [res]
fadd    [term]                ; res += term
fstp    [res]

; Сравниваем term / res с accuracy (= 0.001)
fld     [res]
fld     [term]
fdivrp  st1, st0
fld     [accuracy]
fcompp
fstsw   ax
sahf
jb calculation_loop ; Пока term / res > accuracy, продолжаем вычисления

; Вывод результата, повтор решения

```

## 4. Ограничения, крайние случаи

На вход программе подаётся действительное число. Дополнительных ограничений на это число не налагается, так как функция гиперболического синуса определена для всех действительных чисел.

В программе отдельно обработаны следующие случаи:

1) Ввод числа 0 (обычный алгоритм приведёт к делению на ноль при обработке условия выхода из цикла). Проверяем перед началом цикла:

```

fld     [x]
fldz
fcompp
fstsw   ax
sahf
je      output

```

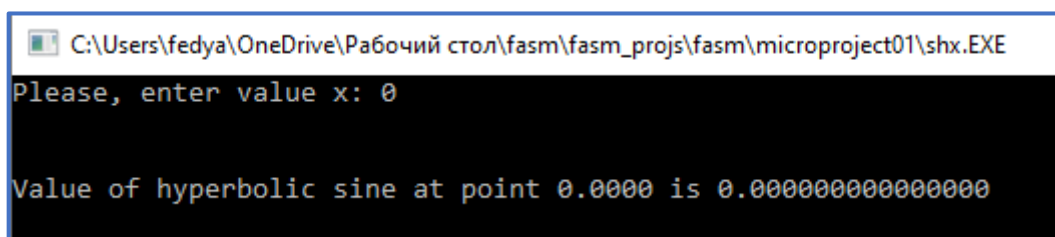


Рисунок 1

2) Ввод слишком большого (по модулю) числа, что приводит к переполнению при вычислении значения гиперболического синуса. Такая ситуация возникает (приблизительно) при достижении значения  $|x| = 710$  (значение гиперболического синуса превышает  $1e308$  по модулю). Проверка на бесконечность производится после каждой итерации вычисления суммы с помощью проверки соответствующего флага. Обнаружение бесконечности в процессе вычисления приводит к выходу из цикла:

```

fstsw ax
and ax, 1000b
cmp eax, 0
jg infty_case

```

В `infty_case` расположены инструкции, проверяющие знак бесконечности (проверяем, сравнивая `x` с нулём):

```

infty_case:
    fld     [x]
    fldz
    fcompp
    fstsw   ax
    sahf
    jbe     sv_inf ; Если значение больше нуля, то это пол. беск.
    invoke  printf, outMsgStr, dword[x], dword[x+4], negInfty
    jmp     endprog
sv_inf:
    invoke  printf, outMsgStr, dword[x], dword[x+4], infty
    jmp     endprog

```

Рисунок 2

Рисунок 3

Технически можно было бы просто проверить входное значение на принадлежность диапазону (допустим,  $[-710; 710]$ ), но по какой-то причине я решил проверять переполнение на месте.

Последующие тесты показали, что по загадочным причинам при вводе числа, изначально выходящего за границы FP значений двойной точности (напр. `1e310`) данный подход не работает и программа «ломается». Для решения этой проблемы было решено всё-таки прикрутить проверку на входного значение. Границей допустимого я решил использовать `upperBound = 1e10`, который также использую для определения типа вывода.

```

Please, enter value x: 1e400

Value of hyperbolic sine at point 1.#INF is INFTY

Please, enter value x: -1e500

Value of hyperbolic sine at point -1.#INF is NEG_INFTY

```

Рисунок 4

3) Ввод символов, не представляющих вещественное число (напр. “abc”). В этом случае выводится сообщение «Wrong input!» и программа предлагает ввести значение снова. Это достигается проверкой значения регистра `eax` после вызова `scanf`.

```

Please, enter value x: abc
Wrong input!

Please, enter value x:

```

Рисунок 5

Однако здесь возникает проблема: `scanf(“%lf”, &x)` буферизирует ввод, т. е. при вводе «1.5abc» он считывает 1.5, сохраняет “abc” и после вычисления значения для 1.5 пытается прочесть FP значение из “abc”. Также (например) ввод “2.2.2.2” будет прочитан как последовательность “2.2”, “.2”, “.2”, “.2” и обработан соответственно.

```

Please, enter value x: 1.5abc

Cycle #1:      2.062500000000000
Cycle #2:      2.125781250000000
Cycle #3:      2.129171316964286
Cycle #4:      2.129277256556920
Cycle #5:      2.129279423503133
Cycle #6:      2.129279454757165
Cycle #7:      2.129279455092029
Cycle #8:      2.129279455094800
Cycle #9:      2.129279455094818

Value of hyperbolic sine at point 1.5000 is 2.129279455094818

Please, enter value x: Wrong input!

Please, enter value x:

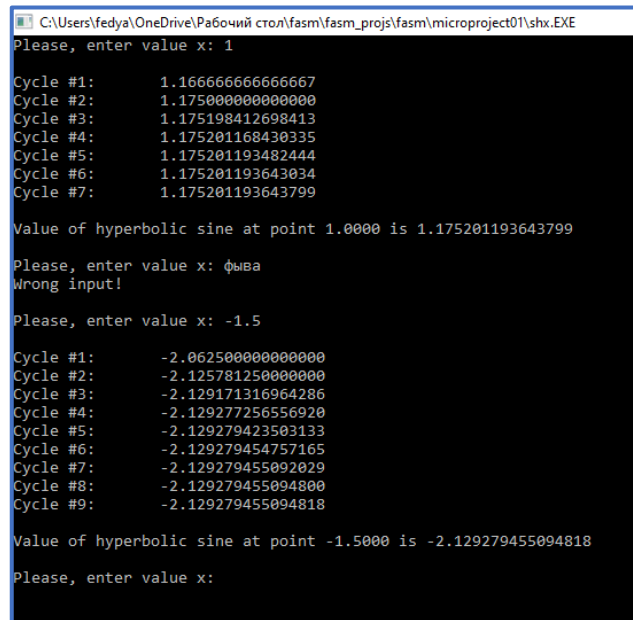
```

Рисунок 6

После ряда попыток исправить данную ситуацию было принято решение оставить всё в текущем состоянии. Испробованные варианты решения (попытки очистить буфер, считать всю оставшуюся строку) либо потерпели неудачу, либо повлекли более неприятные побочные эффекты. Другие не испробованные на практике способы (посимвольный перевод строки в вещественное число) видятся мне чересчур сложными в контексте отсутствия острой необходимости такой жёсткой обработки ввода.

## 5. Особенности программы

1. Принято решение «зациклить» программу. После каждой итерации (запрос ввода => вычисление результата => вывод) программа возвращается в исходное состояние и снова запрашивает у пользователя ввод. Для выхода из программы можно воспользоваться стандартными средствами операционной системы (нажать на крестик, сочетание Ctrl+C).



```
C:\Users\fedya\OneDrive\Рабочий стол\fas\asm_proj\asm\microproject01\shx.EXE
Please, enter value x: 1
Cycle #1:      1.166666666666667
Cycle #2:      1.175000000000000
Cycle #3:      1.175198412698413
Cycle #4:      1.175201168430335
Cycle #5:      1.175201193482444
Cycle #6:      1.175201193643034
Cycle #7:      1.175201193643799
Value of hyperbolic sine at point 1.0000 is 1.175201193643799
Please, enter value x: фыва
Wrong input!
Please, enter value x: -1.5
Cycle #1:      -2.062500000000000
Cycle #2:      -2.125781250000000
Cycle #3:      -2.129171316964286
Cycle #4:      -2.129277256556920
Cycle #5:      -2.129279423503133
Cycle #6:      -2.129279454757165
Cycle #7:      -2.129279455092029
Cycle #8:      -2.129279455094800
Cycle #9:      -2.129279455094818
Value of hyperbolic sine at point -1.5000 is -2.129279455094818
Please, enter value x:
```

Рисунок 7

2. Программа выводит все промежуточные значения при вычислении суммы ряда. Этот функционал был необходим при разработке, в финальной версии я решил его не убирать.

3. Точность я увеличил с 0,1% до  $10^{-12}$ , так как для больших входных данных точность 0,1% даёт довольно неточные значения.

4. В программе используется небольшой макрос `Movf`, который копирует значение второго аргумента в первый, где оба аргумента представляют floating-point значения.

```
macro Movf dstf, srcf {
    mov edx, dword[srcf]
    mov dword[dstf], edx
    mov edx, dword[srcf+4]
    mov dword[dstf+4], edx
}
```

5. Как указано в пункте 4.3, использование `scanf` для считывания ввода приводит к тому, что при введении нескольких «слов» каждое «слово» будет обработано отдельно. Так, ввод «1abc 2.2 gb1» приведет к обработке частей «1», «abc», «2.2», «gb1», как представлено на рисунке 8.

```
C:\Users\fedya\OneDrive\Рабочий стол\fas\fasm_proj\fas\microproject01\shx.EXE
Please, enter value x: 1abc 2.2  rg1

Cycle #1: 1.166666666666667
Cycle #2: 1.175000000000000
Cycle #3: 1.175198412698413
Cycle #4: 1.175201168430335
Cycle #5: 1.175201193482444
Cycle #6: 1.175201193643034
Cycle #7: 1.175201193643799

Value of hyperbolic sine at point 1.0000 is 1.175201193643799

Please, enter value x: Wrong input!

Please, enter value x:
Cycle #1: 3.974666666666667
Cycle #2: 4.404136000000001
Cycle #3: 4.453627227936509
Cycle #4: 4.456954138258907
Cycle #5: 4.457100522313093
Cycle #6: 4.457105063972210
Cycle #7: 4.457105168646639
Cycle #8: 4.457105170509228
Cycle #9: 4.457105170535588
Cycle #10: 4.457105170535892

Value of hyperbolic sine at point 2.2000 is 4.457105170535892

Please, enter value x: Wrong input!

Please, enter value x:
```

Рисунок 8

6. При достаточно больших промежуточных значениях и значениях результата программа выводит эти значения в экспоненциальном виде (рис. 9), что реализовано в «функциях» PrintMidResult и PrintResult.

```
Please, enter value x: 25

Cycle #1: 2629.166666666666500
Cycle #2: 84009.375000000000000
Cycle #3: 1295024.379960317400000
Cycle #4: 11807307.409129739000000
Cycle #5: 71536188.256683260000000
Cycle #6: 310834589.088227870000000
Cycle #7: 1023032210.610682000000000
Cycle #8: 2659515715.947203600000000
Cycle #9: 5650165396.752250700000000
Cycle #10: 1.010053694556928e+010
Cycle #11: 1.559753737642036e+010
Cycle #12: 2.132357949189024e+010
Cycle #13: 2.642155146079148e+010
Cycle #14: 3.034548308710093e+010
Cycle #15: 3.298253391123363e+010
Cycle #16: 3.454328842362277e+010
Cycle #17: 3.536301243223052e+010
Cycle #18: 3.574764269152469e+010
Cycle #19: 3.590985180897331e+010
Cycle #20: 3.597166930800099e+010
Cycle #21: 3.599306240705542e+010
Cycle #22: 3.599981527923169e+010
Cycle #23: 3.600176742775628e+010
Cycle #24: 3.600228617470690e+010
Cycle #25: 3.600241331856735e+010
Cycle #26: 3.600244215198998e+010
Cycle #27: 3.600244821962943e+010
Cycle #28: 3.600244940768539e+010
Cycle #29: 3.600244962467390e+010
Cycle #30: 3.600244966172805e+010
Cycle #31: 3.600244966765708e+010
Cycle #32: 3.600244966854786e+010
Cycle #33: 3.600244966867376e+010
Cycle #34: 3.600244966869053e+010

Value of hyperbolic sine at point 25.0000 is 3.600244966869053e+010
```

Рисунок 9



## 6. Тестирование

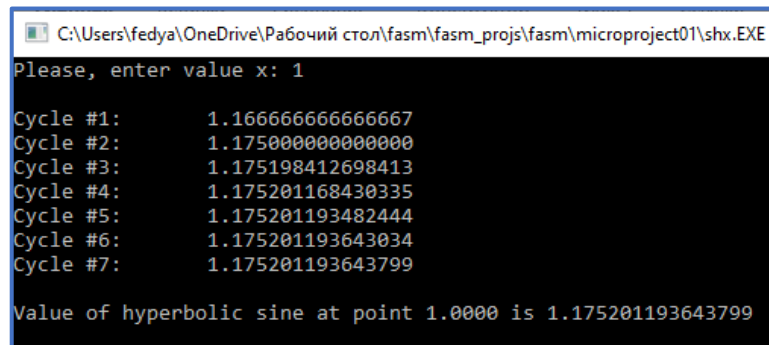
Ниже приведены скриншоты, демонстрирующие вывод программы при различных входных данных. Также предоставлены значения гиперболического синуса, полученные в онлайн калькуляторе [5].

P.S. Я отключил вывод значения на каждой итерации в некоторых тестах для компактности

1)  $x = 1$ ,

Значение в онлайн калькуляторе: 1.1752011936438014

Значение, полученное в программе: 1.175201193643799



```
C:\Users\fedya\OneDrive\Рабочий стол\fasn\fasn_projs\fasn\microproject01\shx.EXE
Please, enter value x: 1

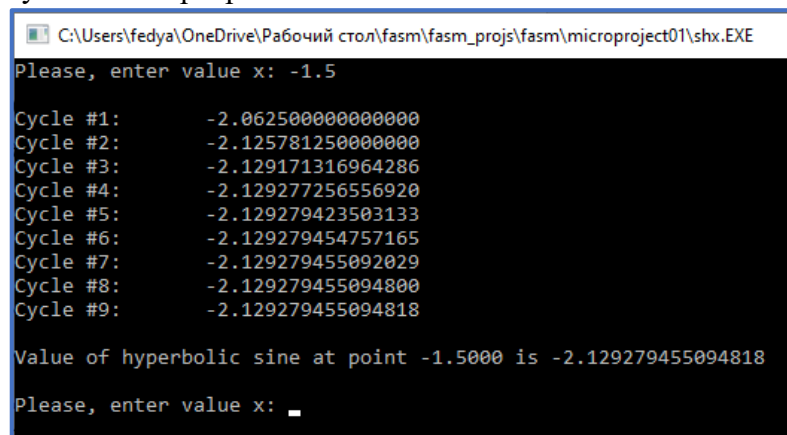
Cycle #1:      1.166666666666667
Cycle #2:      1.175000000000000
Cycle #3:      1.175198412698413
Cycle #4:      1.175201168430335
Cycle #5:      1.175201193482444
Cycle #6:      1.175201193643034
Cycle #7:      1.175201193643799

Value of hyperbolic sine at point 1.0000 is 1.175201193643799
```

2)  $x = -1.5$ ,

Значение в онлайн калькуляторе: -2.1292794550948173

Значение, полученное в программе: -2.129279455094818



```
C:\Users\fedya\OneDrive\Рабочий стол\fasn\fasn_projs\fasn\microproject01\shx.EXE
Please, enter value x: -1.5

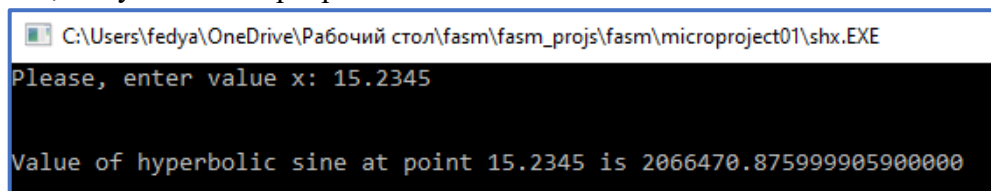
Cycle #1:      -2.062500000000000
Cycle #2:      -2.125781250000000
Cycle #3:      -2.129171316964286
Cycle #4:      -2.129277256556920
Cycle #5:      -2.129279423503133
Cycle #6:      -2.129279454757165
Cycle #7:      -2.129279455092029
Cycle #8:      -2.129279455094800
Cycle #9:      -2.129279455094818

Value of hyperbolic sine at point -1.5000 is -2.129279455094818
Please, enter value x: 
```

3)  $x = 15.2345$ ,

Значение в онлайн калькуляторе: 2066470.8760000272

Значение, полученное в программе: 2066470.875999905900000



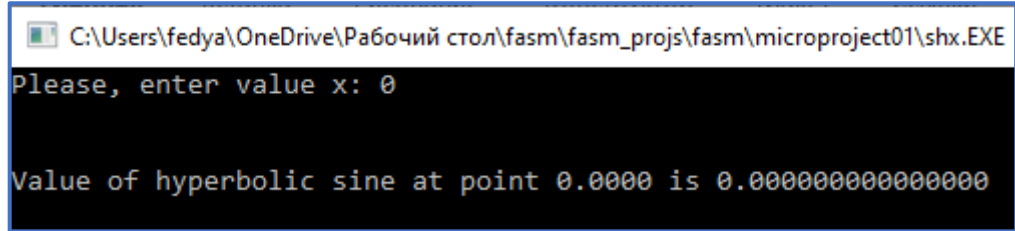
```
C:\Users\fedya\OneDrive\Рабочий стол\fasn\fasn_projs\fasn\microproject01\shx.EXE
Please, enter value x: 15.2345

Value of hyperbolic sine at point 15.2345 is 2066470.875999905900000
```

4)  $x = 0$ ,

Значение в онлайн калькуляторе: 0

Значение, полученное в программе: 0



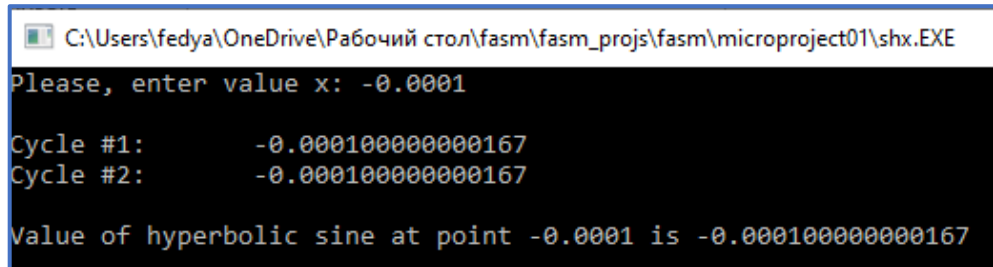
```
C:\Users\fedya\OneDrive\Рабочий стол\fasм\fasм_projс\fasм\microproject01\shx.EXE
Please, enter value x: 0

Value of hyperbolic sine at point 0.0000 is 0.0000000000000000
```

5)  $x = -0.0001$ ,

Значение в онлайн калькуляторе: -0.000100000000016668897

Значение, полученное в программе: -0.0001000000000167



```
C:\Users\fedya\OneDrive\Рабочий стол\fasм\fasм_projс\fasм\microproject01\shx.EXE
Please, enter value x: -0.0001

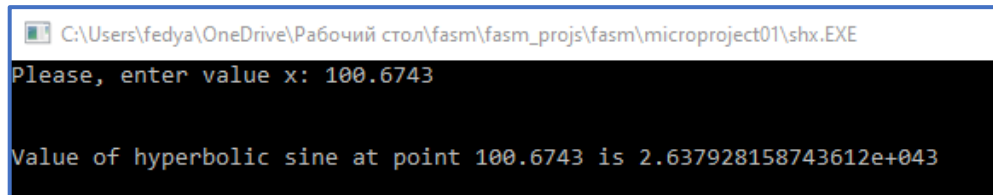
Cycle #1:      -0.0001000000000167
Cycle #2:      -0.0001000000000167

Value of hyperbolic sine at point -0.0001 is -0.0001000000000167
```

6)  $x = 100.6743$ ,

Значение в онлайн калькуляторе: 2.6379281587439964e+43

Значение, полученное в программе: 2.637928158743612e+043



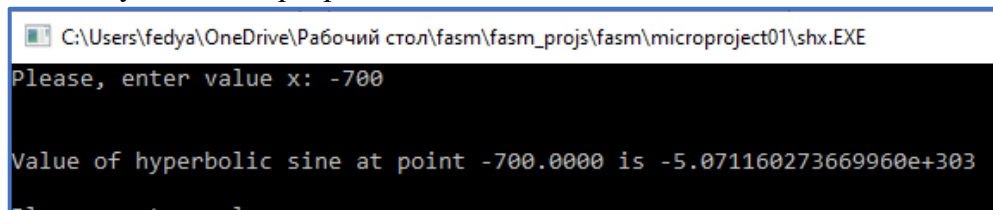
```
C:\Users\fedya\OneDrive\Рабочий стол\fasм\fasм_projс\fasм\microproject01\shx.EXE
Please, enter value x: 100.6743

Value of hyperbolic sine at point 100.6743 is 2.637928158743612e+043
```

7)  $x = -700$ ,

Значение в онлайн калькуляторе: -5.0711602736748336e+303

Значение, полученное в программе: -5.071160273669960e+303



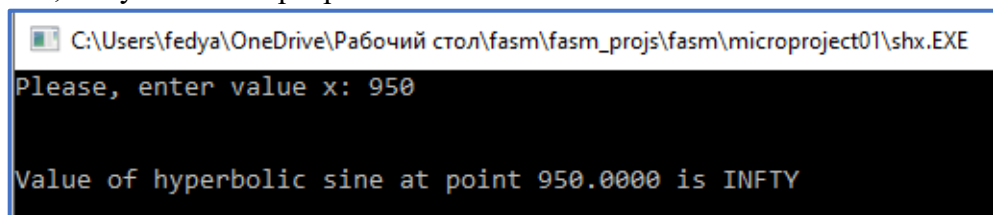
```
C:\Users\fedya\OneDrive\Рабочий стол\fasм\fasм_projс\fasм\microproject01\shx.EXE
Please, enter value x: -700

Value of hyperbolic sine at point -700.0000 is -5.071160273669960e+303
```

8)  $x = 950$ ,

Значение в онлайн калькуляторе:  $\infty$

Значение, полученное в программе: INFY



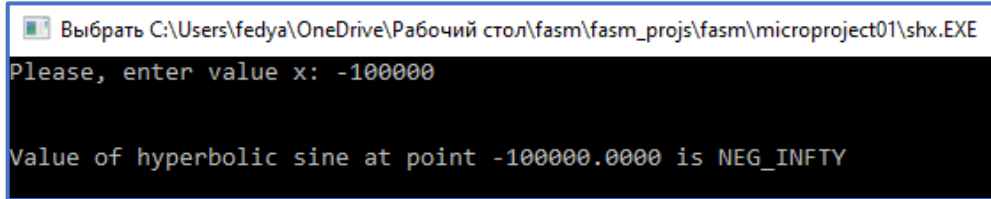
```
C:\Users\fedya\OneDrive\Рабочий стол\fasм\fasм_projс\fasм\microproject01\shx.EXE
Please, enter value x: 950

Value of hyperbolic sine at point 950.0000 is INFY
```

9)  $x = -100000$ ,

Значение в онлайн калькуляторе: -

Значение, полученное в программе: NEG\_INFINITY



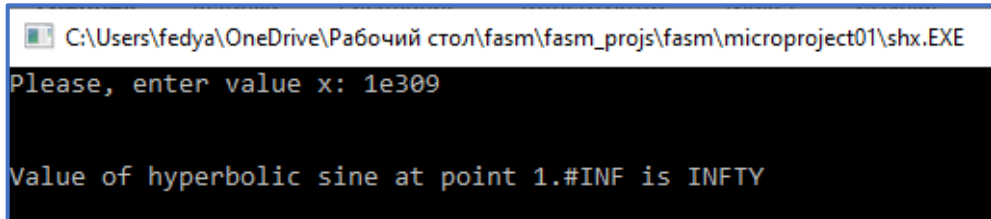
```
Выбрать C:\Users\fedya\OneDrive\Рабочий стол\fas\fas\proj\fas\microproject01\shx.EXE
Please, enter value x: -100000

Value of hyperbolic sine at point -100000.0000 is NEG_INFINITY
```

10)  $x = 1e309$ ,

Значение в онлайн калькуляторе:  $\infty$

Значение, полученное в программе: INFTY



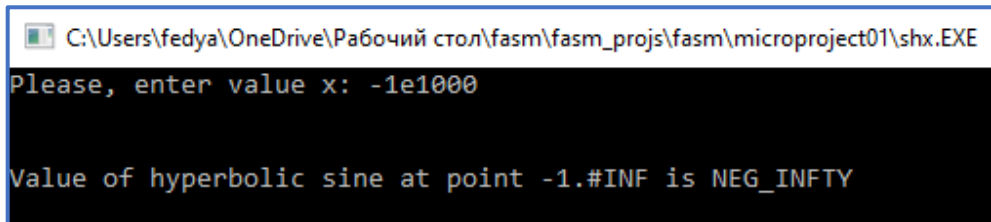
```
C:\Users\fedya\OneDrive\Рабочий стол\fas\fas\proj\fas\microproject01\shx.EXE
Please, enter value x: 1e309

Value of hyperbolic sine at point 1.#INF is INFTY
```

11)  $x = -1e1000$ ,

Значение в онлайн калькуляторе: -

Значение, полученное в программе: NEG\_INFINITY



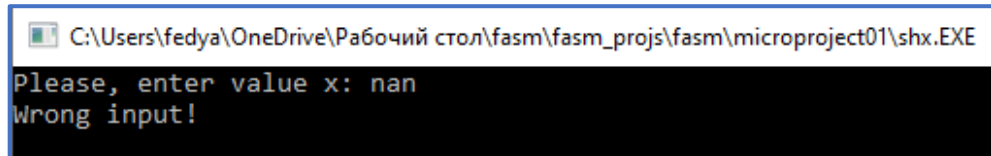
```
C:\Users\fedya\OneDrive\Рабочий стол\fas\fas\proj\fas\microproject01\shx.EXE
Please, enter value x: -1e1000

Value of hyperbolic sine at point -1.#INF is NEG_INFINITY
```

12)  $x = \text{nan}$ ,

Значение в онлайн калькуляторе: «Ожидается число.»

Значение, полученное в программе: «Wrong input!»



```
C:\Users\fedya\OneDrive\Рабочий стол\fas\fas\proj\fas\microproject01\shx.EXE
Please, enter value x: nan
Wrong input!
```

13)  $x = \text{lala.2 21.3.3 6lala}$

Значения, полученные в программе:

«lala.2»	=>	«Wrong input!»
«21.3»	=>	890107517.3809628500000000
«.3»	=>	0.304520293447143
«6»	=>	201.713157370273590
«lala»	=>	«Wrong input!»

```

C:\Users\fedya\OneDrive\Рабочий стол\fasm\fasm_projs\fasm\microproject01\shx.EXE
Please, enter value x: lala.2 21.3.3 6lala
Wrong input!

Please, enter value x:

Value of hyperbolic sine at point 21.3000 is 890107517.3809628500000000

Please, enter value x:

Value of hyperbolic sine at point 0.3000 is 0.304520293447143

Please, enter value x:

Value of hyperbolic sine at point 6.0000 is 201.713157370273590

Please, enter value x: Wrong input!

```

## 7. Исходный код программы на языке ассемблера FASM

; Махнач Федор, БПИ 196

; Вариант 15

format PE console

entry start

include 'win32a.inc'

; Перемещает fp значение из одного адреса в другой через регистр edx

```

macro Movf dstf, srcf {
    mov edx, dword[srcf]
    mov dword[dstf], edx
    mov edx, dword[srcf+4]
    mov dword[dstf+4], edx
}

```

; -----

section '.data' data readable writable

```

inputRequest db 'Please, enter value x: ', 0
wrongInput   db 'Wrong input!', 13, 10, 0
calcMsg      db 'Cycle #d:', 9, '%.15lf', 13, 10, 0
calcMsgExp   db 'Cycle #d:', 9, '%.15e', 13, 10, 0
outMsg       db 'Value of hyperbolic sine at point %.4lf is %.15lf', 13, 10, 0
outMsgExp    db 'Value of hyperbolic sine at point %.4lf is %.15e', 13, 10, 0
outMsgStr    db 13, 10, 'Value of hyperbolic sine at point %.4lf is %s', 13, 10, 0
fmtFlt      db '%lf', 0
fmts        db '%s', 0
newLine     db 13, 10, 0
infty       db 'INFTY', 0
negInfty    db 'NEG_INFTY', 0
holder      db 0

```

```

x          dq 1 ; Вводимое пользователем значение
res        dq 1 ; Результат
xSqr       dq 1 ; Квадрат значения x (чтобы не вычислять каждый раз)

```

```

term      dq 1 ; Предыдущее слагаемое суммы степенного ряда
lastDenomN dd 1 ; Последнее число, на которое мы делили

accuracy  dq 1e-12 ; "с точностью не хуже 0,1%" <=> можно и точнее.
upperBound dq 1e10 ; Значение, после которого мы начинаем выводить в
                    экспоненциальном формате. Также это граница допустимого ввода (по модулю)

section '.code' code readable executable
;-----
start:
    FINIT
    invoke printf, inputRequest
    invoke scanf, fmtFlt, x, holder
    cmp     eax, 0
    jne     calculation
    invoke  scanf, fmts, holder ; Считываем строку до конца, освобождаясь от
буфера (чтобы сканф сработал на следующей итерации)
    invoke  printf, wrongInput ; Сообщаем о неверном вводе
    jmp     endprog

calculation:
    invoke  printf, newline
    Movf    res, x ; Записываем в результат число x
    fld     [x]
    fldz
    fcomp           ; Проверка на ноль
    fstsw    ax
    sahf
    je       output

    fabs
    fld     [upperBound]
    fcompp           ; Проверка на слишком большое значение
    fstsw    ax
    sahf
    jb      infty_case

correct_input_val:
    Movf    term, x ; Первое слагаемое -- само число x
    Movf    xSqr, x ; Записываем в xSqr значение x^2 (не считать его каждый раз)
    fld     [xSqr]
    fmul    [x]
    fstp    [xSqr]
    mov     [lastDenomN], 1
    mov     ebx, 1 ; Счётчик

calculation_loop:
    ; Вычисляем очередное слагаемое, умножая предыдущее на x^2 / (n*(n + 1))
    fld     [term]
    fmul    [xSqr]
    inc     [lastDenomN]
    fidiv   dword[lastDenomN]
    inc     [lastDenomN]
    fidiv   dword[lastDenomN]

```

```

    fstp    [term]
    ; Добавляем слагаемое к результату
    fld     [res]
    fadd    [term]
    fstp    [res]
    ; Выводим сообщение, содержащее значение промежуточного результата
    call    PrintMidResult
    inc     ebx
    ; Проверяем, является ли значение нулём или бесконечностью
    fstsw   ax
    and     ax, 1000b
    cmp     eax, 0
    jg      infty_case
    ; Необходимо вычислить отклонение: |res_i - res_{i - 1}| / |res_i|
    ; Значение под модулем это просто term, term имеет тот же знак, что и res,
    ; поэтому просто сравниваем term/res с accuracy
    fld     [res]
    fld     [term]
    fdivrp  st1, st0
    fld     [accuracy]
    fcompp
    fstsw   ax
    sahf
    ; Если accuracy больше, чем term/res -- продолжаем вычислять сумму
    jb      calculation_loop

output:
    ; Выводим результат
    invoke  printf, newline
    call    PrintResult

endprog:
    invoke  printf, newline
    jmp     start      ; Бесконечный цикл, да

infty_case:
    fld     [x]
    fldz
    fcompp
    fstsw   ax
    sahf
    jbe     sv_inf     ; Если значение больше нуля, то это полож. беск.
    invoke  printf, outMsgStr, dword[x], dword[x+4], negInfty
    jmp     endprog

sv_inf:
    invoke  printf, outMsgStr, dword[x], dword[x+4], infty
    jmp     endprog

;-----
; При большом значении (>upperBound) мы выводим значение в экспоненциальной форме
PrintMidResult:
    fld     [res]
    fabs
    fld     [upperBound]

```

```

        fcompp
        fstsw  ax
        sahf
        jb     PrintMidResult_exp_output

        invoke printf, calcMsg, ebx, dword[res], dword[res+4]
        add    esp, 16
ret
PrintMidResult_exp_output:
        invoke printf, calcMsgExp, ebx, dword[res], dword[res+4]
        add    esp, 16
ret
;-----
; При большом значении (>upperBound) мы выводим значение в экспоненциальной форме
PrintResult:
        fld    [res]
        fabs
        fld    [upperBound]
        fcompp
        fstsw  ax
        sahf
        jb     PrintResult_exp_output
        invoke printf, outMsg, dword[x], dword[x+4], dword[res], dword[res+4]
        add    esp, 20
ret
PrintResult_exp_output:
        invoke printf, outMsgExp, dword[x], dword[x+4], dword[res], dword[res+4]
        add    esp, 20
ret
;-----
section '.idata' import data readable

        library kernel, 'kernel32.dll',\
                msvcrt, 'msvcrt.dll'

        import kernel,\
                ExitProcess, 'ExitProcess'

        import msvcrt,\
                printf, 'printf',\
                getch,  '_getch',\
                scanf,  'scanf'

```

## 8. Используемые источники

- 1) Степенной ряд функции гиперболического синуса:  
[https://scask.ru/m\\_book\\_gf.php?id=10](https://scask.ru/m_book_gf.php?id=10)
- 2) Презентация с семинара про FPU:  
<http://softcraft.ru/edu/comparch/practice/asm86/05-fpu/fpu.pdf>
- 3) Команды сравнения FPU: <http://osinavi.ru/asm/FPUexpansion/5.html>
- 4) Инструкции FPU: <http://flatassembler.narod.ru/fasm.htm#2-1-13>
- 5) Онлайн калькулятор гиперболических функций: <https://planetcalc.ru/1116/>

- 6) Команды форматирования для функций `printf`, `scanf`:  
<https://prog-cpp.ru/c-input-output/>
- 7) Бесчисленное множество сайтов/форумов по разным вопросам