Lab 6

1. Assembly listing

```c
float a = 3.1;
double b = 3.4;

int main()
{
        double c = b + 1.2;
        float d = a + 4.3;
}
```

```
a:
        .long   1078355558
        .globl  b
        .align 8
        .type   b, @object
        .size   b, 8
b:
        .long   858993459
        .long   -1073007821
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movsd   b(%rip), %xmm1
        movsd   .LC0(%rip), %xmm0
        addsd   %xmm1, %xmm0
        movsd   %xmm0, -8(%rbp)
        movss   a(%rip), %xmm0
        cvtss2sd        %xmm0, %xmm1
        movsd   .LC1(%rip), %xmm0
        addsd   %xmm1, %xmm0
        cvtsd2ss        %xmm0, %xmm0
        movss   %xmm0, -12(%rbp)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .section        .rodata
        .align 8
.LC0:
        .long   858993459
        .long   1072902963
        .align 8
.LC1:
        .long   858993459
        .long   1074869043
```

- movsd - moves scalar double-precision floating-point value(b to %xmm1(first 64 bits are rewritten))
- movss – same as movsd, but performs with float values
- addsd – adds two double values
- cvtss2sd - converts one single-precision floating-point value to one double-precision floating-point value(%xmm0 to %xmm1 - first 32 bits of %xmm0 are rewritten as double in first 64 bits of %xmm1)

Interestingly, float and double numbers are written as huge numbers, e.g. float a is written as 10783555588. That is because floating-point number representation(in binary) is converted to decimal. As for doubles, it's value is divided in 2 rows: first row is for last 32 bits of double(mantissa), second row is for first 32 bits(sign + exp + mantissa)

```c
#include <stdio.h>

float a = 3.1;
double b = -3.4;

int main()
{
        double c = b * 1.2;
        float d = a * 4.3f;
}
```

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movsd   b(%rip), %xmm1
        movsd   .LC0(%rip), %xmm0
        mulsd   %xmm1, %xmm0
        movsd   %xmm0, -8(%rbp)
        movss   a(%rip), %xmm1
        movss   .LC1(%rip), %xmm0
        mulss   %xmm1, %xmm0
        movss   %xmm0, -12(%rbp)
        movl    $0, %eax
        popq    %rbp
```

- mulsd – multiplies 2 double values
- mulss – multiplies 2 float values

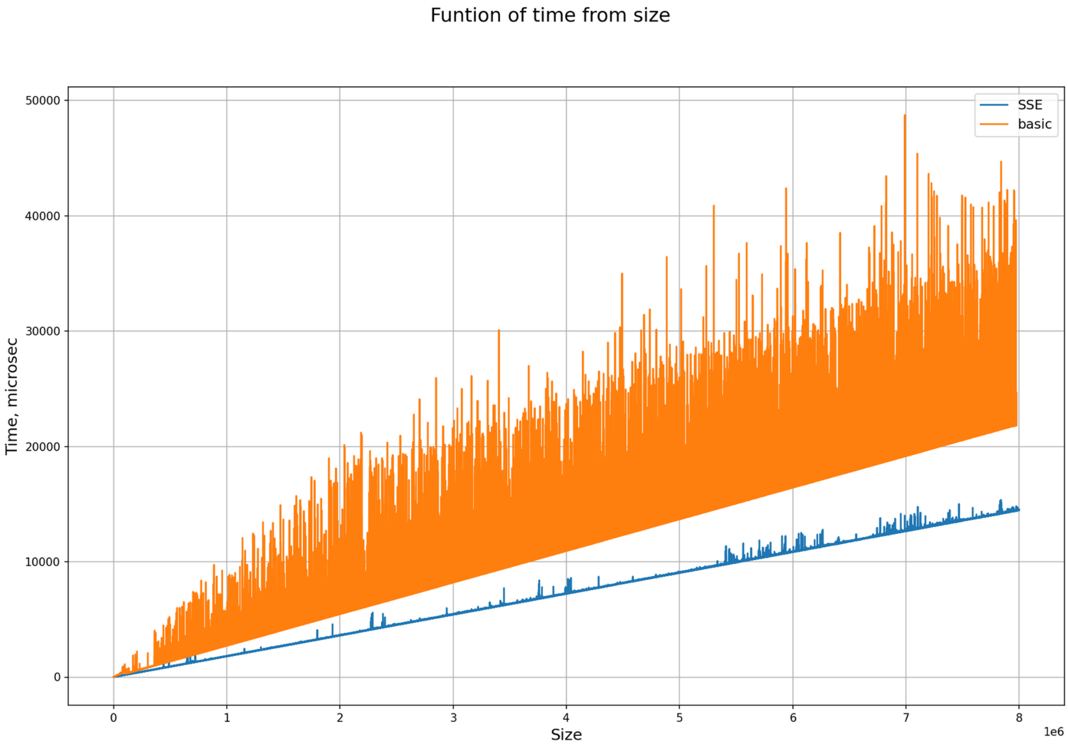
2. Mean value

Basic code:

```cpp
#include <iostream>
#include <chrono>
#include <fstream>
#include <random>

void basic()
{
        std::ofstream out;
        out.open("basic.txt");
        std::chrono::steady_clock::time_point begin, end;
        for(int i = 16; i <= 16000000; i += 4)
        {
                float *arr = new float[i];
                for(int j = 0; j < i; j ++) arr[j] = 0.7f;
                float sum = 0.0f, mean = 0.0f;
                begin = std::chrono::steady_clock::now();   // start
                for(int j = 0; j < i; j ++) sum += arr[j];
                mean = sum /= i;
                end = std::chrono::steady_clock::now();     // finish
                delete [] arr;
                out << i << " " << std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count() << "\n";
        }
}
```

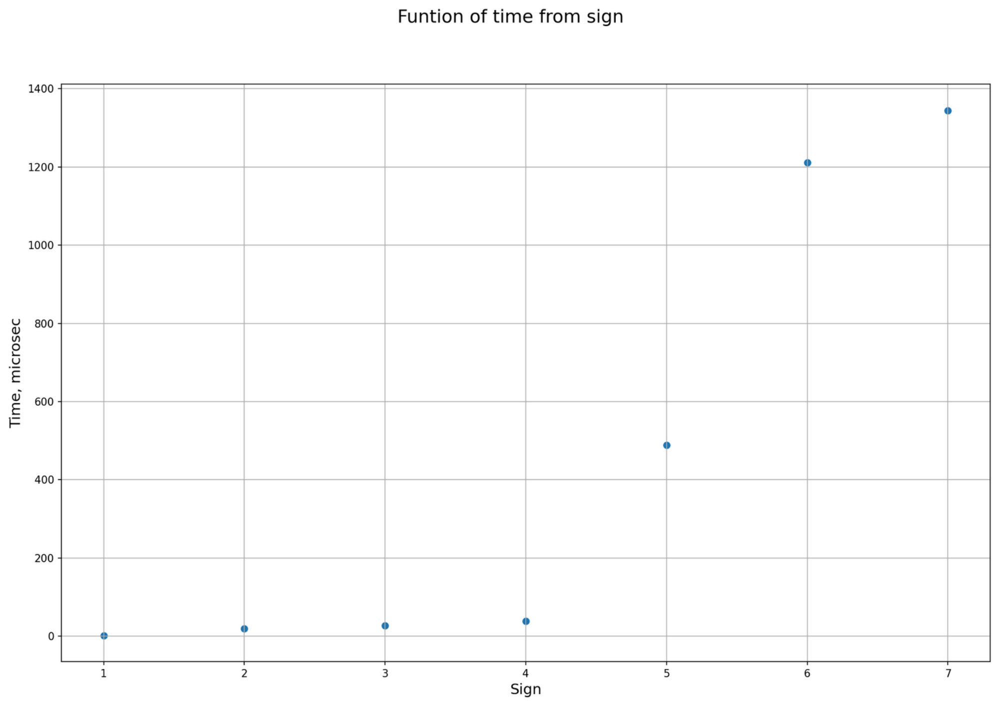
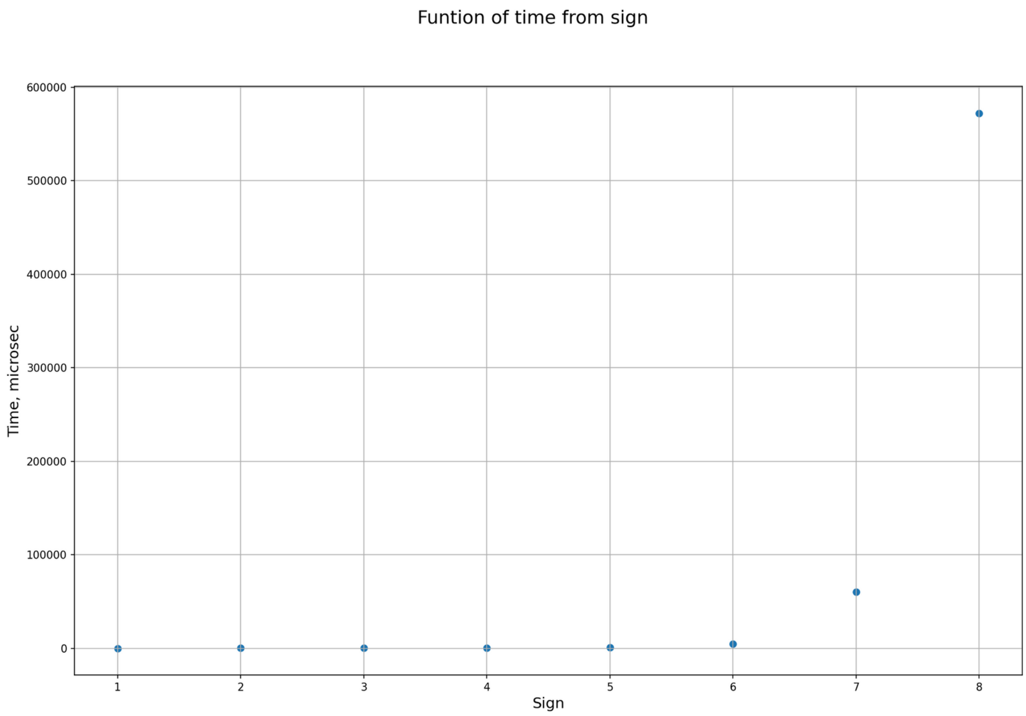
Code, using SSE:

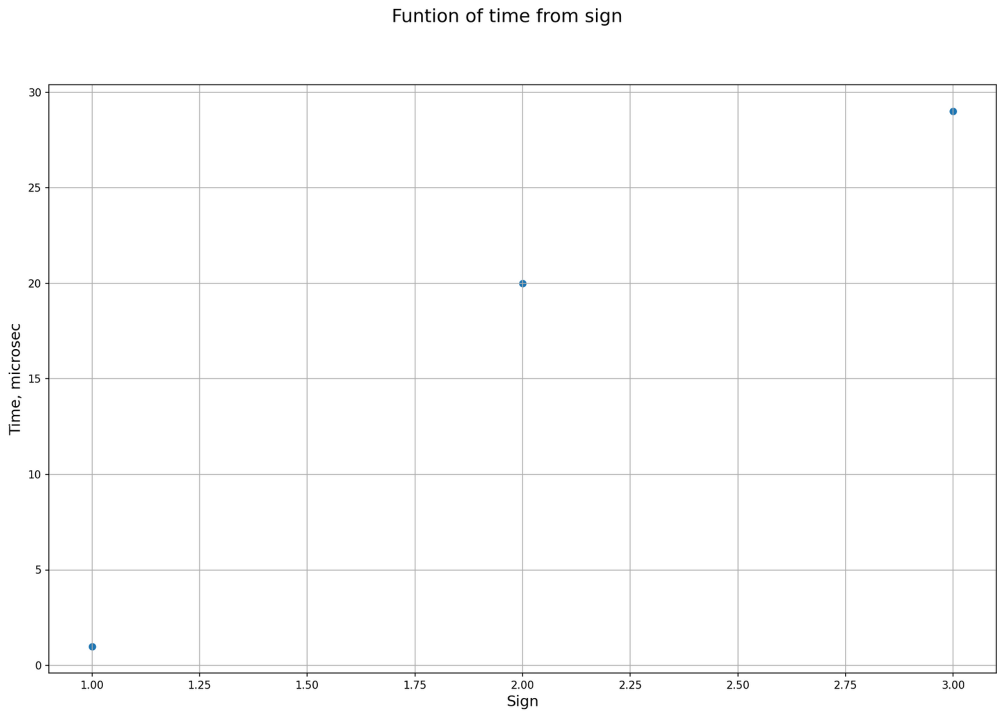
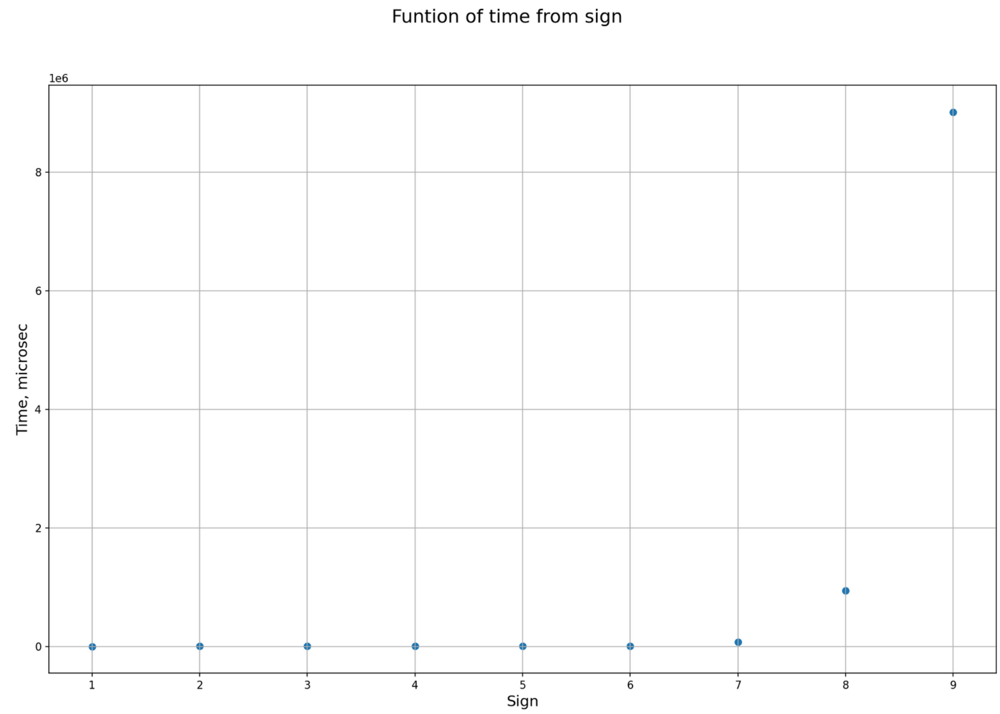
```cpp
float supporting[4];
float result[4] = {0.0f};
float x = 3.0f;


void SSE()
{
        std::ofstream out;
        out.open("sse.txt");
        std::chrono::steady_clock::time_point begin, end;
        for(int i = 16; i <= 8000000; i += 16)
        {
                float *arr = new float[i];
                asm("movq %rax, %r15 \n");
                for(int j = 0; j < i; j ++) arr[j] = 0.7f;
                //std::cout << arr[0] << "\n";
                float sum = 0.0f, mean = 0.0f;
                begin = std::chrono::steady_clock::now();   // start
                asm("movq %r15, %r12 \n");
                for(int j = 0; j < i; j += 4)
                {
                        //std::cout << "norm\n";
                        asm(
                        "movq %r12, %rax \n"
                        "movss (%rax), %xmm0 \n"
                        "movss %xmm0, supporting(%rip) \n"
                        "addq $4, %rax \n"
                        "movss (%rax), %xmm0 \n"
                        "movss %xmm0, 4+supporting(%rip) \n"
                        "addq $4, %rax \n"
                        "movss (%rax), %xmm0 \n"
                        "movss %xmm0, 8+supporting(%rip) \n"
                        "addq $4, %rax \n"
                        "movss (%rax), %xmm0 \n"
                        "movss %xmm0, 12+supporting(%rip) \n"
                        "addq $4, %rax \n"
                        "movq %rax, %r12 \n"
                        "movups supporting(%rip), %xmm0 \n"
                        "movups result(%rip), %xmm1 \n"
                        "addps %xmm1, %xmm0 \n"
                        "movups %xmm0, result(%rip) \n"
                        );
                        //std::cout << result[0] << "\n";
                }
                mean = (result[0] + result[1] + result[2] + result[3]) / i;
                end = std::chrono::steady_clock::now();     // finish
                //std::cout << mean << "\n";
                delete [] arr;
                out << i << " " << std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count() << "\n";
        }

}
```



Basic algorithm works almost twice slower, than with using SSE, as was expected.

3. π(sign)

Method	Plot
Monte-Carlo float	
Wallis double	

<table>
<tr><td>Wallis float</td><td>
</td></tr>
<tr><td>Leibniz double</td><td>
</td></tr>
</table>

Leibniz float