power(long double, unsigned long):

        pushq   %rbp

        movq    %rsp, %rbp

        movq    %rdi, -40(%rbp)

        fld1

        fstpt   -16(%rbp)

        movq    $1, -24(%rbp)

        jmp     .L8

.L9:

        fldt    -16(%rbp)

        fldt    16(%rbp)

        fmulp   %st, %st(1)

        fstpt   -16(%rbp)

        addq    $1, -24(%rbp)

.L8:

        movq    -24(%rbp), %rax

        cmpq    %rax, -40(%rbp)

        jnb     .L9

        fldt    -16(%rbp)

        popq    %rbp

        ret

factorial(unsigned long):

        pushq   %rbp

        movq    %rsp, %rbp

        movq    %rdi, -40(%rbp)

        fld1

        fstpt   -16(%rbp)

        movq    $1, -24(%rbp)

        jmp     .L12

.L14:

        fildq   -24(%rbp)

        cmpq    $0, -24(%rbp)

        jns     .L13

        fldt    .LC2(%rip)

        faddp   %st, %st(1)

.L13:

        fldt    -16(%rbp)

        fmulp   %st, %st(1)

        fstpt   -16(%rbp)

        addq    $1, -24(%rbp)

.L12:

        movq    -24(%rbp), %rax

        cmpq    %rax, -40(%rbp)

        jnb     .L14

        fldt    -16(%rbp)

        popq    %rbp

        ret

Sin(long double, unsigned long):

        pushq   %rbp

        movq    %rsp, %rbp

        subq    $96, %rsp

        movq    %rdi, -72(%rbp)

        fldt    16(%rbp)

        fldz

        fcomip  %st(1), %st

        fstp    %st(0)

        jbe     .L32

        movl    $-1, %eax

        jmp     .L19

.L32:

        movl    $1, %eax

.L19:

        movl    %eax, -4(%rbp)

        pushq   24(%rbp)

        pushq   16(%rbp)

        call    std::fabs(long double)

        addq    $16, %rsp

        movq    .LC4(%rip), %rax

        leaq    -16(%rsp), %rsp

        fstpt   (%rsp)

        movq    %rax, %xmm0

        call    \_\_gnu\_cxx::\_\_promote\_2<decltype (((\_\_gnu\_cxx::\_\_promote\_2<long double, std::\_\_is\_integer<long double>::\_\_value>::\_\_type)(0))+((\_\_gnu\_cxx::\_\_promote\_2<double, std::\_\_is\_integer<double>::\_\_value>::\_\_type)(0))), std::\_\_is\_integer<decltype (((\_\_gnu\_cxx::\_\_promote\_2<long double, std::\_\_is\_integer<long double>::\_\_value>::\_\_type)(0))+((\_\_gnu\_cxx::\_\_promote\_2<double, std::\_\_is\_integer<double>::\_\_value>::\_\_type)(0)))>::\_\_value>::\_\_type std::fmod<long double, double>(long double, double)

        addq    $16, %rsp

        fstpt   16(%rbp)

        fldt    .LC5(%rip)

        fldt    16(%rbp)

        fcomip  %st(1), %st

        fstp    %st(0)

        jbe     .L20

        fldt    16(%rbp)

        fldt    .LC5(%rip)

        fsubrp  %st, %st(1)

        fstpt   16(%rbp)

        negl    -4(%rbp)

.L20:

        fldt    .LC6(%rip)

        fldt    16(%rbp)

        fcomip  %st(1), %st

        fstp    %st(0)

        jbe     .L22

        fldt    .LC5(%rip)

        fldt    16(%rbp)

        fsubrp  %st, %st(1)

        fstpt   16(%rbp)

.L22:

        fldz

        fstpt   -32(%rbp)

        movq    $1, -40(%rbp)

        jmp     .L24

.L27:

        movq    -40(%rbp), %rax

        subq    $1, %rax

        fld1

        fchs

        leaq    -16(%rsp), %rsp

        fstpt   (%rsp)

        movq    %rax, %rdi

        call    power(long double, unsigned long)

        addq    $16, %rsp

        fstpt   -96(%rbp)

        movq    -40(%rbp), %rax

        addq    %rax, %rax

        subq    $1, %rax

        pushq   24(%rbp)

        pushq   16(%rbp)

        movq    %rax, %rdi

        call    power(long double, unsigned long)

        addq    $16, %rsp

        fldt    -96(%rbp)

        fmulp   %st, %st(1)

        fstpt   -96(%rbp)

        movq    -40(%rbp), %rax

        addq    %rax, %rax

        subq    $1, %rax

        movq    %rax, %rdi

        call    factorial(unsigned long)

        fldt    -96(%rbp)

        fdivp   %st, %st(1)

        fstpt   -64(%rbp)

        pushq   -56(%rbp)

        pushq   -64(%rbp)

        call    std::isnan(long double)

        addq    $16, %rsp

        testb   %al, %al

        je      .L25

        fldt    -32(%rbp)

        fldz

        faddp   %st, %st(1)

        fstpt   -32(%rbp)

        jmp     .L26

.L25:

        fldt    -32(%rbp)

        fldt    -64(%rbp)

        faddp   %st, %st(1)

        fstpt   -32(%rbp)

.L26:

        addq    $1, -40(%rbp)

.L24:

        movq    -40(%rbp), %rax

        cmpq    %rax, -72(%rbp)

        jnb     .L27

        fildl   -4(%rbp)

        fldt    -32(%rbp)

        fmulp   %st, %st(1)

        leave

        ret

.LC9: // Метки с константными строками

        .string "Answer: "

.LC10:

        .string "Time: "

.LC12:

        .string "s"

main:

        pushq   %rbp // Запоминает предыдущее значение регистра %rbp, ложа его в стек

        movq    %rsp, %rbp // Помещает значение регистра %rsp в регистр %rbp

        pushq   %rbx // Запоминает значение регистра %rbx в стеке

        subq    $72, %rsp // Отнимает из значения регистра %rsp число 72, работает как выделение памяти

        movl    %edi, -52(%rbp) // Копирует значение регистра %edi в стек по адресу %rbp-52 (записывает 4 байта) (argc)

        movq    %rsi, -64(%rbp) // Копирует значение регистра %rsi в стек по адресу %rbp-64 (записывает 8 байт) (\*argv [])

        movq    -64(%rbp), %rax // Копию значения регистра %rsi из стека помещает в регистр %rax (Записывает \*argv [] === \*argv[0])

        addq    $8, %rax // Прибавляет к значению %rsi, хранящемуся в регистре %rax, число 8 и оставляет в %rax (\*argv[0] => \*argv[1])

        movq    (%rax), %rax // Переходит по адресу, хранящемуся в %rax, и записывает значение по этому адресу в %rax (\*argv[1] => argv[1], то есть переходит к конкретной строке из массива строк и записывает адрес ее начала)

        movq    %rax, %rdi // Копирует значение в регистре %rax в регистр %rdi (записывает в %rdi указатель на начало строки argv[1])

        call    atoll // Запоминает адрес команды movq и вызывает функцию atoll, которая переводит строку в 8-байтовое число и возвращает через регистр %rax, после чего возвращается к функции movq

        movq    %rax, -24(%rbp) // Копирует значение из регистра %rax в стэк по адресу (8 байт) (записывает результат функции atoll(argv[1]) в стек)

        fldt    .LC8(%rip) // Загружает вещественное число по метке .LC8 на вершину стека сопроцессора (long double x = 10)

        fstpt   -48(%rbp) // Записывает вещественное число в стек (оперативную память) и «выталкивает» его из стека сопроцессора

        movl    $.LC9, %esi // помещает адрес метки со строкой “Answer: “ в регистр %esi

        movl    $\_ZSt4cout, %edi // помещает адрес cout в регистр %edi

        call    std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*) // помещает в стек следующую функцию movq и вызывает перегруженный оператор вывода для строк (предыдущие две функции были подготовкой, так как адрес строки, идущей на вывод, поместился в регистр, и адрес объекта cout тоже был помещен в регистр), после выполнения оператора возвращается к вызову функции movq

        movq    %rax, %rbx // Копирует значение регистра %rax в регистр %rbx

        movq    -24(%rbp), %rax // Копирует значение из оперативной памяти (numberOfElements = atoll(argv[1])) в регистр %rax

        pushq   -40(%rbp) // Копирует на вершину стека первую половину значения long double x

        pushq   -48(%rbp) // Копирует на вершину стека вторую половину значения long double x

        movq    %rax, %rdi // Копирует в регистр %rdi значение регистра %rax (size\_t numberOfElements)

        call    Sin(long double, unsigned long) // Сохраняет адрес функции addq в стек, вызывает функцию Sin(), после чего возвращается к функции addq, в дальнейшем запоминание функции, к которой нужно вернуться, считаем по умолчанию выполненным для функции call

        addq    $16, %rsp // Возвращает значение %rsp на предыдущее положение (до вызова функции Sin(), то есть по факту затирая результат вызова двух функций pushq)

        leaq    -16(%rsp), %rsp // Выделяет 16 байт стеку (помещает в %rsp адрес %rsp – 16)

        fstpt   (%rsp) // Записывает по адресу, хранящемуся в %rsp long double число – результат функции Sin() в оперативную память (стек), и удаляет это число со стека сопроцессора

        movq    %rbx, %rdi // Копирует значение из регистра %rbx в регистр %rdi

// Предположу, что в C++ компиляторах %rbx отвечает за работу с буфером

// ввода/вывода. Это объясняет почему сохраняется предыдущее значение %rbx (может // быть подпрограмма задействует регистр %rbx для каких-то специфичных задач, при // этом не выводя ничего), также это объясняет почему сейчас вместо ссылки на

// объект cout в функцию передается значение, которое хранилось в регистре %rax // после вызова функции

        call    std::basic\_ostream<char, std::char\_traits<char> >::operator<<(long double) // Выводит long double Sin()

        addq    $16, %rsp // Затирает 16 байт, в которых хранился long double

// Sin()

        movl    $\_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_, %esi // Перемещает адрес функции std::endl в регистр %esi

        movq    %rax, %rdi // Копирует значение регистра %rax в регистр %rdi (адрес потока cout)

        call    std::basic\_ostream<char, std::char\_traits<char> >::operator<<(std::basic\_ostream<char, std::char\_traits<char> >& (\*)(std::basic\_ostream<char, std::char\_traits<char> >&)) // Вызывает оператор вывода в поток cout для std::endl;

        movl    $.LC10, %esi // Помещает адрес метки со строкой “Time: “ в регистр %esi

        movl    $\_ZSt4cout, %edi // Помещает в регистр %edi адрес объекта cout

// Получается, что при каждом вызове “cout <<” заново в регистр %edi помещается // адрес объекта cout, а при многократном вызове оператора “<<” используется

// возвращаемое оператором значение. Следовательно, если подпрограмма вызывается

// при выводе в какой-то поток, то значение потока надо запомнить

        call    std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*) // Выводит строку, адрес которой хранится в %esi

        movq    %rax, %rbx // Копируем значение из регистра %rax в регистр %rbx, которое будет применятся для последующих вызовов оператора вывода

        call    clock // Вызывает функцию, которая вернет количество квантов процессора, прошедших с момента запуска подпрограммы

        movq    %rax, -72(%rbp) // Помещает значение регистра %rax (результат функции clock()) по адресу %rbp-72

        fildq   -72(%rbp) // берет результат функции clock() и загружает в сопроцессор на вершину стека

        fldt    .LC11(%rip) // Загружает вещественное число по метке .LC11 в сопроцессор на вершину стека (CLOCKS\_PER\_SEC)

        fdivrp  %st, %st(1) // Делит значение в регистре %st(1) на значение в регистре %st, записывает его в регистр %st(1), и выталкивает из стекового регистра %st, делая %st(1) - вершиной

        leaq    -16(%rsp), %rsp // Выделяет 16 байт на стеке

        fstpt   (%rsp) // помещает на вершину стека результат деления в сопроцессоре (clock() / CLOCKS\_PER\_SEC)

        movq    %rbx, %rdi // Копирует значение из регистра %rbx в регистр %rdi (cout)

        call    std::basic\_ostream<char, std::char\_traits<char> >::operator<<(long double) // Вызывает оператор вывода long double

        addq    $16, %rsp // Затирает только что выведенное значение

        movl    $.LC12, %esi // Помещает адрес строки в регистр %esi

        movq    %rax, %rdi // Помещает значение регистра %rax в регистр %rdi (cout)

        call    std::basic\_ostream<char, std::char\_traits<char> >& std::operator<< <std::char\_traits<char> >(std::basic\_ostream<char, std::char\_traits<char> >&, char const\*) // Вызывает оператор для вывода строк

        movl    $\_ZSt4endlIcSt11char\_traitsIcEERSt13basic\_ostreamIT\_T0\_ES6\_, %esi // Помещает адрес функции std::endl в регистр %esi

        movq    %rax, %rdi // Помещает значение регистра %rax в регистр %rdi (cout)

        call    std::basic\_ostream<char, std::char\_traits<char> >::operator<<(std::basic\_ostream<char, std::char\_traits<char> >& (\*)(std::basic\_ostream<char, std::char\_traits<char> >&)) // Вызывает оператор для вывода std::endl

        movl    $0, %eax // Помещает 0 в регистр %rax

        movq    -8(%rbp), %rbx // Восстанавливает значение %rbx

        leave // Эквивалентно movq %rbp %rsp; popq %rbp – восстанавливаем значения кадра и стека до исходных состояний

        ret // Выходим из подпрограммы

\_\_static\_initialization\_and\_destruction\_0(int, int):

        pushq   %rbp // Запоминает значение регистра %rbp – указатель кадра вызвавшей подпрограммы

        movq    %rsp, %rbp // Помещает в регистр %rbp значение регистра %rsp, тем самым формируя указатель начало кадра локальных переменных

        subq    $16, %rsp // Резервирует место из 16 байт в стеке

        movl    %edi, -4(%rbp) // Копирует значение из регистра %edi в стек

        movl    %esi, -8(%rbp) // Копирует значение из регистра %edi в стек

        cmpl    $1, -4(%rbp) // Сравнивает значения 1 и

        jne     .L39

        cmpl    $65535, -8(%rbp)

        jne     .L39

        movl    $\_ZStL8\_\_ioinit, %edi

        call    std::ios\_base::Init::Init() [complete object constructor]

        movl    $\_\_dso\_handle, %edx

        movl    $\_ZStL8\_\_ioinit, %esi

        movl    $\_ZNSt8ios\_base4InitD1Ev, %edi

        call    \_\_cxa\_atexit

.L39:

        nop

        leave

        ret

\_GLOBAL\_\_sub\_I\_power(long double, unsigned long):

        pushq   %rbp

        movq    %rsp, %rbp

        movl    $65535, %esi

        movl    $1, %edi

        call    \_\_static\_initialization\_and\_destruction\_0(int, int)

        popq    %rbp

        ret

.LC2: //

        .long   0

        .long   -2147483648

        .long   16447

        .long   0

.LC4:

        .long   1413754136

        .long   1075388923

.LC5:

        .long   560513024

        .long   -921707870

        .long   16384

        .long   0

.LC6:

        .long   560513024

        .long   -921707870

        .long   16383

        .long   0

.LC8: // long double x = 10;

        .long   0

        .long   -1610612736

        .long   16386

        .long   0

.LC11: // CLOCKS\_PER\_SEC

        .long   0

        .long   -198967296

        .long   16402

        .long   0