МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт (факультет) \_\_\_\_\_\_\_\_\_Институт информационных технологий\_\_\_\_\_\_\_\_\_

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_МПО ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КУРСОВАЯ РАБОТА

по дисциплине Программирование на ассемблере

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

на тему Программирование на языке низкого уровня

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнил студент группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1ПИб-02-1оп-22

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

направления подготовки (специальности)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Программная инженерия

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*шифр, наименование*

\_\_\_\_Микуцких Григорий Андреевич\_\_\_\_

*фамилия, имя, отчество*

Руководитель

Виноградова Людмила Николаевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*должность*

Дата представления работы

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г.

Заключение о допуске к защите

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

количество баллов

Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Череповец, \_2023\_

*год*

Аннотация

Курсовая работа посвящена программированию на языке низкого уровня. В ходе работы были изучены базовые возможности разработки с использованием языка ассемблера и разработана программа, вычисляющая значение функции вида на отрезке a = 1.01, b = 1.1, на данном языке.

Оглавление

[2](#_Toc156212348)

[Введение 4](#_Toc156212349)

[1. Описание предметной области 5](#_Toc156212350)

[2. Описание созданной программы 12](#_Toc156212351)

[2.1 Постановка задачи 12](#_Toc156212352)

[2.2 Логическое проектирование – алгоритм работы программы 12](#_Toc156212353)

[2.3 Физическое проектирование – структуры данных 14](#_Toc156212354)

[2.4 Тестирование 17](#_Toc156212355)

[2.5 Результаты работы 21](#_Toc156212356)

[Заключение 22](#_Toc156212357)

[Список литературы 23](#_Toc156212358)

[Приложение 1. Техническое задание 24](#_Toc156212359)

[Приложение 2. Руководство пользователя 31](#_Toc156212360)

[Приложение 3. Текст программы 38](#_Toc156212361)

# Введение

В наше время программирование на языках низкого уровня является непопулярным, но важным умением в изучении языков программирования. Эти языки обеспечивают более прямое управление аппаратными ресурсами, такими как память, процессор, регистры и другие устройства. Они широко используются для создания системного программного обеспечения, драйверов устройств, операционных систем, встраиваемых систем, игр и других высокопроизводительных приложений, которым нужен быстрый доступ к аппаратным ресурсам компьютера.

Цель: разработка программы на языке низкого уровня.

Задача: написать программу для вычисления значения функции вида: на отрезке a = 1.01, b = 1.1.

План решения: реализовать вычисление заданной функции с помощью программы «Turbo Assembler», используя язык ассемблера.

# Описание предметной области

Язык ассемблера — представление команд процессора в виде, доступном для чтения человеком. Язык ассемблера считается языком программирования низкого уровня, в противовес высокоуровневым языкам, не привязанным к конкретной реализации вычислительной системы. Программы, написанные на языке ассемблера, однозначным образом переводятся в инструкции конкретного процессора и, в большинстве случаев, не могут быть перенесены без значительных изменений для запуска на машине с другой системой команд [3].

Сердцем компьютера является процессор, называемый часто центральным процессором (ЦП) или микропроцессором. Именно центральный процессор выполняет все вычисления [2].

Обычно одна строка на ассемблере – одна инструкция, однако это зависит от процессора, для которого написана программа, поэтому ассемблеру нужно, например, знать где находится первая инструкция. «Подсказками» для ассемблера являются директивы, которые не переводятся в инструкции процессора, а лишь управляют программой-ассемблером.

Так как компьютер способен воспринимать только два управляющих сигнала: 0 и 1, то и любая программа должна быть ему представлена только в двоичных кодах, т. е. в машинных кодах [2]. Чтобы перевести эту запись в понимаемые процессором числа, нужна особая программа, которая тоже называется ассемблером. Эта программа читает написанный человеком текст и затем переводит его в последовательность чисел, понимаемую процессором [1].

Однако записывать и запоминать огромные двоичные цепочки, первым программистам было неудобно, поэтому они стали вместо двоичной системы использовать другие системы счисления, например десятичную, восьмеричную или шестнадцатеричную. Двоичное число 11001000 будет представлено в десятичном виде как 200, а в восьмеричной и шестнадцатеричной соответственно как 310 и C8 [2].

Оперативная память делится на ячейки в 8 разрядов (байт). Разряды байтов нумеруются справа налево от 7 до 0 (рис. 1). В байте могут быть представлены целые числа в диапазоне от 0 до 255, или числа со знаком от –128 до +127 – DB.



Рис. 1. Состав байта

Слово состоит из двух байт (16 разрядов), при этом биты с 0 по 7 составляют младший байт в слове, а биты с 8 по 15 — старший. Может принимать значения от 0 до 65535 (целые числа) или от -32768 до 32767 (числа со знаком) – DW.



Рис. 2. Состав слова

Двойное слово состоит из двух слов или четырех байт, т. е. из 32-х бит, а два двойных слова составляют четверное слово (64 бита) – DD.

Для кратковременного хранения данных используют регистры – ячейки в ЦП, доступные из машинных программ (рис. 3). Все регистры имеют размер слова, за каждым из них закреплено имя. По назначению и способу использования они делятся на группы:

* регистры общего назначения;
* сегментные регистры;
* указатели команд;
* регистры флагов.

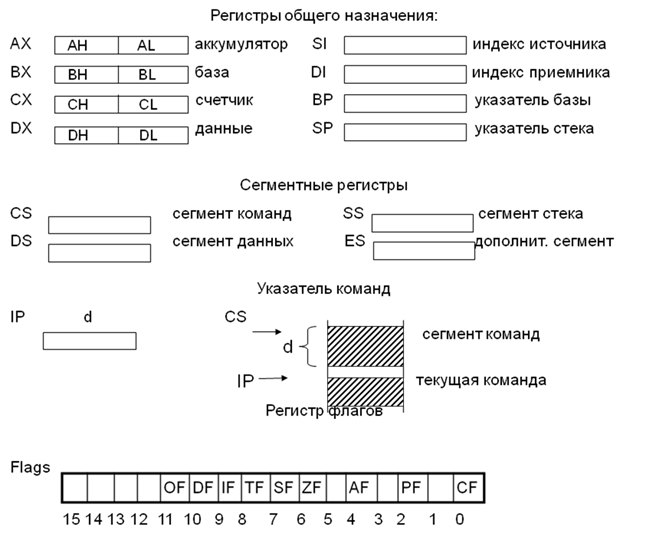


Рис. 3. Регистры

Наиболее интенсивно используемыми в процессоре являются регистры общего назначения. В процессорах первого поколения регистры общего назначения были 16-разрядными. Начиная с третьего поколения (с процессора 80386) регистры общего назначения стали 32-разрядными (рис. 4) [2].

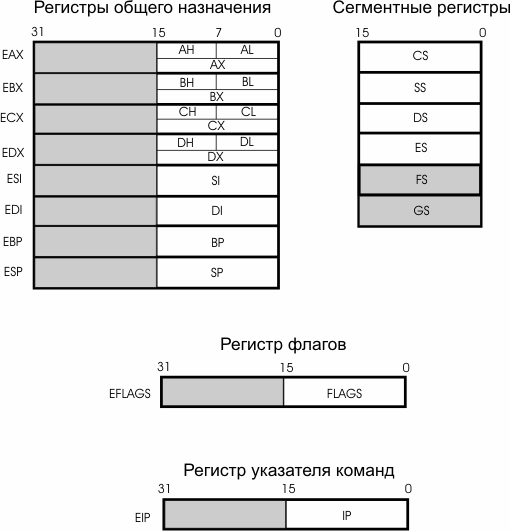


Рис. 4. 32-разрядные регистры

Ниже для каждого регистра описано его назначение и те ситуации, в которых чаще всего применяется регистр помимо хранения каких-либо промежуточных данных:

* AX – умножение, деление, ввод и вывод слова;
* AL – умножение, деление, ввод и вывод байта, двоично-десятичная арифметика;
* AH – умножение и деление байта;
* BX – трансляция, косвенная адресация, модификация;
* CX – счетчик циклов и указатель длины в строковых операциях;
* CL – логические и арифметические сдвиги;
* DX – умножение и деление слова, ввод и вывод с косвенной адресацией;
* BP – модификация;
* SP – операции со стеком;
* SI, DI – строковые операции, модификация.

В реальном режиме работы процессора процессор может аппаратно делить программу в памяти на 3 части, которые прозвали сегментами, а сегментные регистры предназначены для доступа к этим сегментам [2]. Ни в каких логических, арифметических и других операциях эти регистры не участвуют.

Сегмент кода. В этом сегменте содержатся машинные команды. Для доступа к этому сегменту служит регистр CS (code segment register) — сегментный регистр кода.

Сегмент данных. Содержит обрабатываемые программой данные. Для доступа к этому сегменту служит регистр DS (data segment register) — сегментный регистр данных.

Сегмент стека. В этом сегменте содержится стек. Для доступа к этому сегменту служит регистр SS (stack segment register) — сегментный регистр стека.

Если программисту недостаточно одного сегмента данных, адресуемого регистром DS, то он может задействовать в своей программе дополнительные сегменты данных с помощью сегментных регистров ES, GS, FS (extension data segment registers) [2].

Регистр IP хранит адрес текущей команды – это необходимо, чтобы процессор мог определить, по какому адресу в памяти считать очередную команду. После выполнения команды значение регистра IP увеличивается на величину, равную размеру команды – таким образом происходит переход на следующую команду программы. Исключение составляют команды переходов и вызова процедур, по которым происходит переход не на следующую команду, а на адрес, определяемый операндом команды.

Регистр флагов хранит управляющие биты, а также особенности результатов выполнения арифметических и логических команд (команды пересылки данных влияния на регистр флагов не оказывают) (рис. 5).

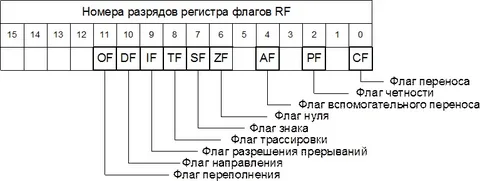


Рис. 5. Регистр флагов

Каждый бит регистра флагов (флаг) характеризует присутствие той или иной особенности в результате выполнения команды, является управляющим битом для некоторых команд или отображает состояние МП. Назначение флагов:

* CF (Carry Flag) – флаг переноса старшего бита в арифметических командах;
* PF (Parity Flag) – флаг четности, устанавливается при четном чис-ле единиц в результате;
* AF (Auxiliary Flag) – флаг дополнительного переноса в тетраде (4 бита) для двоично-десятичной арифметики;
* ZF (Zero Flag) – флаг нулевого результата, устанавливается, когда значение результата равно 0.
* SF (Sign Flag) – флаг знака, устанавливается, когда старший бит результата равен 1 (признак отрицательного числа);
* TF (Trap Flag) – флаг трассировки (пошагового режима), при его установке после выполнения каждой команды вызывается внутреннее прерывание процессора;
* IF (Interrupt-enable Flag) – флаг управления прерываниями, при его установке разрешается выполнение маскируемых аппаратных прерываний;
* DF (Direction Flag) – флаг управления направлением в строковых командах. При единичном значении флага DF значение индексных регистров, участвующих в строковых командах, автоматически уменьшается на размер операнда, при нулевом – увеличивается;
* OF (Overflow Flag) – флаг переполнения, устанавливается, если результат арифметической команды не умещается в операнде назначения.

Установка флага означает запись 1 в бит, соответствующий этому флагу, сброс – запись 0. Флаги CF, PF, AF, ZF, SF, OF автоматически устанавливаются или сбрасываются процессором после выполнения команд, влияющих на эти флаги. Флаги TF, IF, DF устанавливаются или сбрасываются программистом с помощью специальных команд (исключение составляет механизм прерываний, когда флаги TF и IF модифицируется самим МП).

Базовой конструкцией языка ассемблера является мнемоника, или мнемокод — краткое символьное представление команды процессора. Как правило, она состоит из нескольких символов, обозначающих производимое действие (например, mov — для пересылки из одного регистра в другой, add — для сложения значений и т.д.). Также в мнемонику может входить объект, над которым производится операция (регистр, память, стек), либо другие особенности (влияние на регистр флагов, условия исполнения и т.д.), однако в других диалектах те же особенности могут задаваться в операндах [3].

Как правило, ассемблер каждого процессора имеет свой традиционный набор мнемоник, но встречаются ассемблеры с кроссплатформенным синтаксисом (такие, как синтаксис AT&T), однако кроссплатформенными в них остаются только обозначения, код одного процессора не может быть напрямую перенесён на другой [3].

В качестве операндов могут указываться регистры, константные значения, адреса ячеек памяти и портов ввода-вывода, константы, метки и т.д. Как правило, операнды отделяются от мнемоник команд пробелами [3].

Общая структура программы может выглядеть следующим образом:

<директивы>

<сегмент\_1> SEGMENT <параметры>

<имя\_метки>:

<текст>

<сегмент\_1> ENDS

<сегмент\_N> SEGMENT <параметры>

<текст>

<сегмент\_N> ENDS

END <имя\_метки>

# Описание созданной программы

## Постановка задачи

Программа должна:

* принимать данные вещественного типа;
* иметь промежутки вычисления от 1.01 до 1.1 и шаг 0.01;
* вычислять значение функции с точностью до десятитысячных.

## Логическое проектирование – алгоритм работы программы

Программа будет работать по следующему алгоритму:

1. Инициализация сопроцессора;
2. Проверка «X» на принадлежность промежутку [1.01; 1.1]:

* занесение «X» и 1.1 в стек сопроцессора;
* если X > 1.1, то пропуск вычислений;
* если X < 1.01, то пропуск вычислений;

1. Вычисление функции на отрезке a =1.01, b = 1.1.

* занесение «X» и числа 2 в стек сопроцессора;
* вычисление x\*2x;
* занесение «X» в стек сопроцессора;
* вычисление ln(x-1);
* вычисление x\*2x - ln(x-1);
* занесение числа 6 в стек сопроцессора;
* вычисление x\*2x - ln(x - 1) - 6;

1. Проверка «X»:

* занесение «X» и 1.1 в стек сопроцессора;
* сравнение «X» и 1.1:
  + если X ≤ 1.1, то увеличить X на 0.01 и вернуться к 2);
  + иначе закончить вычисления;

Алгоритм работы программы представлен в виде блок-схемы на рис. 6:

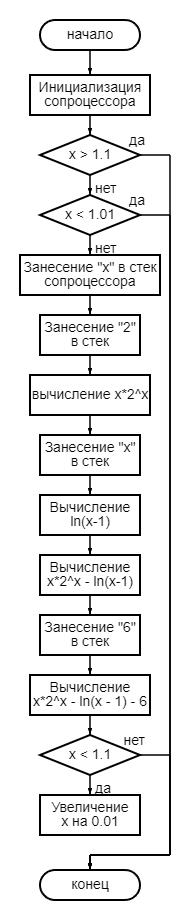


Рис. 6. Блок-схема работы алгоритма вычисления

## Физическое проектирование – структуры данных

В программе используются типы данных, представленные в табл. 1.

Таблица 1

Структуры данных

|  |  |  |
| --- | --- | --- |
| Наименование | Обозначение | Тип данных |
| Переменная x | x | DD |
| Минимальный порог x | minX | DD |
| Максимальный порог x | maxX | DD |
| Шаг | plus | DD |
| Число «2» | two | DD |
| Число «6» | six | DD |
| Результат | result | DD |

Используемые в программе модули представлены в табл. 2.

Таблица 2

Спецификации модулей

|  |  |  |  |
| --- | --- | --- | --- |
| Имя модуля | Заголовок процедуры  или функции | Формальные  параметры | Выполняемое  действие |
| 1 | 2 | 3 | 4 |
| Инициализация сопроцессора | finit | Отсутствуют | Инициализирует управляющие регистры сопроцессора |

Продолжение табл. 2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Загрузка операнда в стек | fld | src (переменная или регистр) | Загружает данные в вершину стека сопроцессора |
| Сохранение в память | fst | dst (переменная или регистр) | Сохраняет вершину стека сопроцессора в память |
| Сохранение с выталкиванием | fstp | dst (переменная или регистр) | Сохраняет вершину стека сопроцессора в память (регистр или переменную), выталкивает вершину стека и сдвигает стек «вверх» |
| Обмен значений | fxch | ST(i) или по умолчанию ST(0) и ST(1) | Обмен значений ST(0) и ST(i) |
| Сложение | fadd | dst, src или регистры стека (по умолчанию ST(0) и ST(1)) | Сложение операндов и запись результата на место ST(0) |
| Вычитание | fsub | dst, src или регистры стека (по умолчанию ST(0) и ST(1)) | Вычитание операндов и запись результата на место ST(0) |
| Умножение | fmul | dst, src или регистры стека (ST(0) и ST(1)) | Умножение операндов и запись результата на место ST(0) |

Продолжение табл. 2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Число 1 | fld1 | Отсутствуют | Загружает константу 1 в вершину стека сопроцессора |
| ln(2) | fldln2 | Отсутствуют | Загружает ln(2) в вершину стека сопроцессора |
| Масштабирование по степеням 2 | fscale | Отсутствуют | ST(0) = ST(0) · 2ST(1) |
| Вычисление y\*log2x | fyl2x | Отсутствуют | x = ST(0), y = ST(1), ST(0) = y\*log2x |
| Частичный остаток от деления | fprem | Отсутствуют | ST(0) = ST(0)-Q\*ST(1) |
| Вещественное сравнение с двойным выталкиванием | fcompp | По умолчанию ST(0) и ST(1), если операнд указан: src и ST(0) | Сравнивает операнды (вычитание первого из второго без сохранения результата) и меняет флаги C0, C2, C3 |
| Запись состояния FPU | fstsw | dst AX | Записывают текущее значение слова состояния FPU (SW) в регистр AX |
| Запись регистра AH во флаги | sahf | Отсутствуют | Копирует регистр AH в биты кода состояния FLAGS: C0 копирует- |

Продолжение табл. 2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
|  |  |  | ся во флаг CF, C2 - в PF, C3 - в ZF |
| Безусловные переход | jmp | Адрес перехода (метка) | Выполняет переход без условия |
| Условный переход «>» | ja | Адрес перехода (метка) | Выполняет переход, если первый операнд больше второго |
| Условный переход «≥» | jnb | Адрес перехода (метка) | Выполняет переход, если первый операнд больше или равен второму |

## Тестирование

Для тестирования программы были подобраны тестовые данные в табл. 3.

Таблица 3

Тестовые данные

|  |  |  |
| --- | --- | --- |
| Исходные данные | Тестируемый модуль или подпрограмма | Ожидаемый результат |
| 1 | 2 | 3 |
| x = 1.00 | Переход на флаг «exit» при x < minX | Вычисления не проводятся, программа завершается |
| x = 1.00 | fld и fst | В вершину стека сопроцессора загрузится x, потом вершина стека загрузится в x |

Продолжение табл. 3

|  |  |  |
| --- | --- | --- |
| x = 1.00 | fld и fstp | В вершину стека загрузится x, потом вершина стека загрузится в x с выталкиванием |
| x = 1.01 | Вычисление значения функции при x = 1.01 | result = 0,6392 |
| x = 1.05 | Вычисление значения функции при x = 1.05 | result = -0,8302 |
| x = 1.1 | Вычисление значения функции при x = 1.1 | result = -1,3395 |
| x = 1.2 | Переход на флаг «exit» при x > maxX | Вычисления не проводятся, программа завершается |

Результаты тестирования представлены в табл. 4.

Таблица 4

Результаты тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата и время  тестирования | Тестируемый  модуль или  подпрограмма | Кто прово-дил тести-рование | Описание  теста | Результаты  тестирования |
| 05.01.24 | Переход на флаг «exit» при x < minX | Микуцких Г. А. | Вводится x меньшее minX | Результаты соответствуют ожидаемым |
| 05.01.24 | fld и fst | Микуцких Г. А. | fld dword ptr [x]  fld1  fadd  fst x | Результаты соответствуют ожидаемым |

Продолжение табл. 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 05.01.24 | fld и fstp | Микуцких Г. А. | fld dword ptr [x]  fld1  fadd  fstp x | Результаты соответствуют ожидаемым |
| 05.01.24 | Вычисление значения функции при x = 1.01 | Микуцких Г. А. | x = 1.01, отсле-живание зна-чений в стеке сопроцессора | Результаты соответствуют ожидаемым |
| 05.01.24 | Вычисление значения функции при x = 1.05 | Микуцких Г. А. | x = 1.05, отсле-живание зна-чений в стеке сопроцессора | Результаты соответствуют ожидаемым |
| 05.01.24 | Вычисление значения функции при x = 1.1 | Микуцких Г. А. | x = 1.1, отсле-живание зна-чений в стеке сопроцессора | Результаты соответствуют ожидаемым |
| 05.01.24 | Переход на флаг «exit» при x > maxX | Микуцких Г. А. | Вводится x большее maxX | Результаты соответствуют ожидаемым |

Результаты тестирования представлены на рис. 7-12:

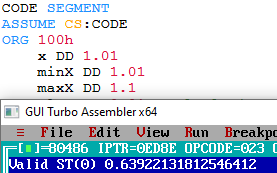


Рис. 7. Тест 1

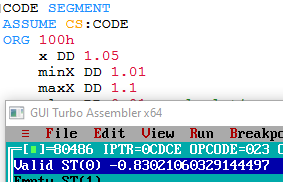


Рис. 8. Тест 2

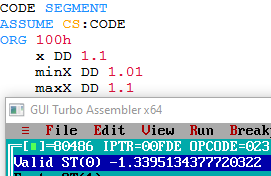


Рис. 9. Тест 3

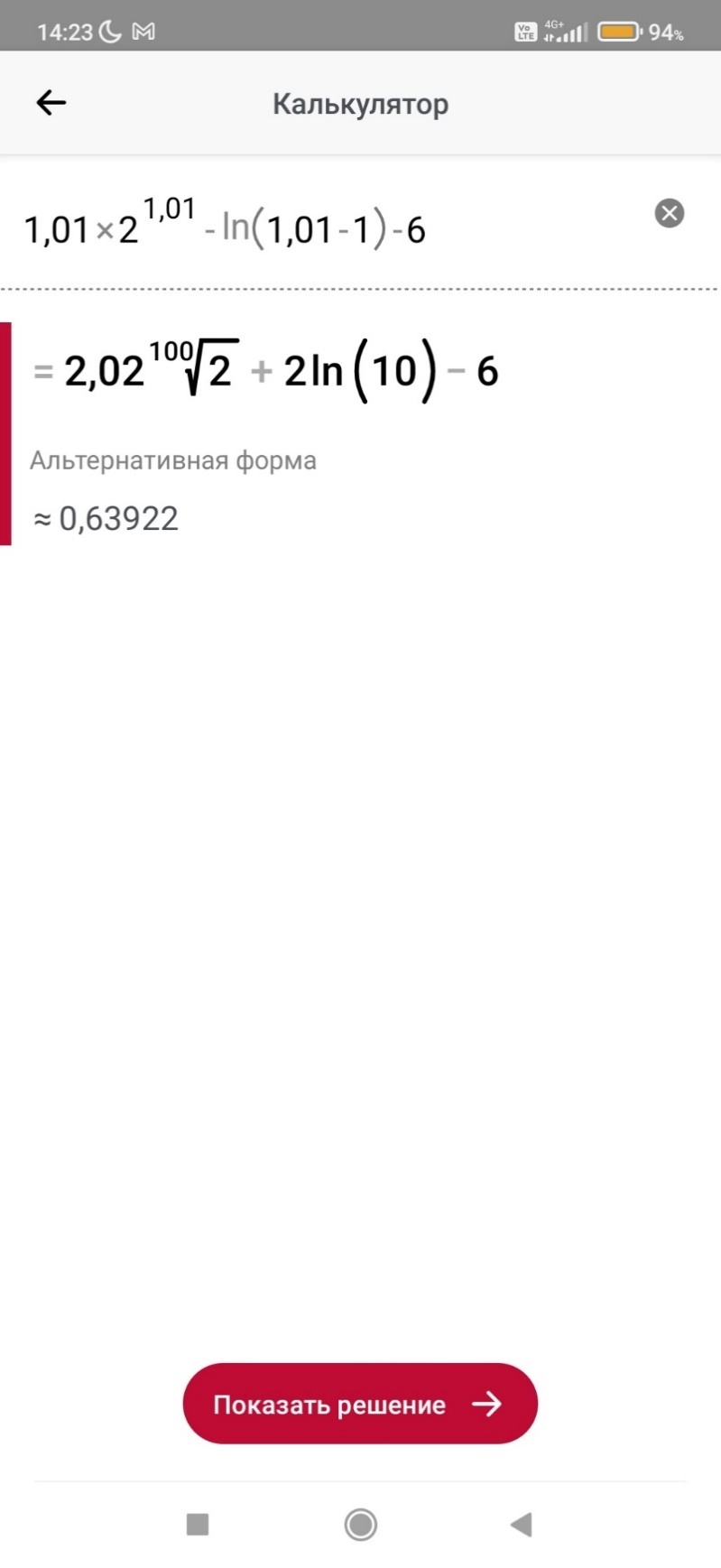


Рис. 10. Проверка результатов 1

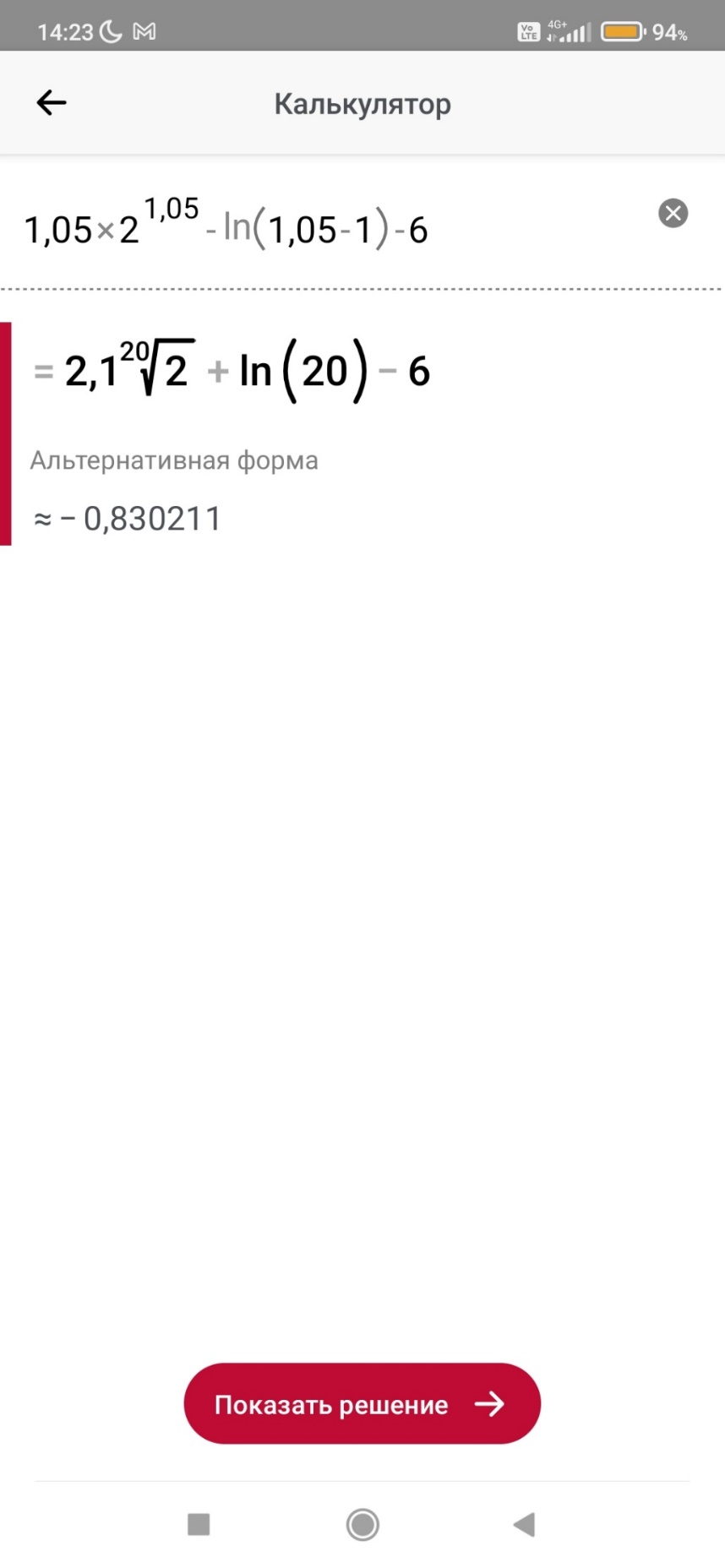


Рис. 11. Проверка результатов 2

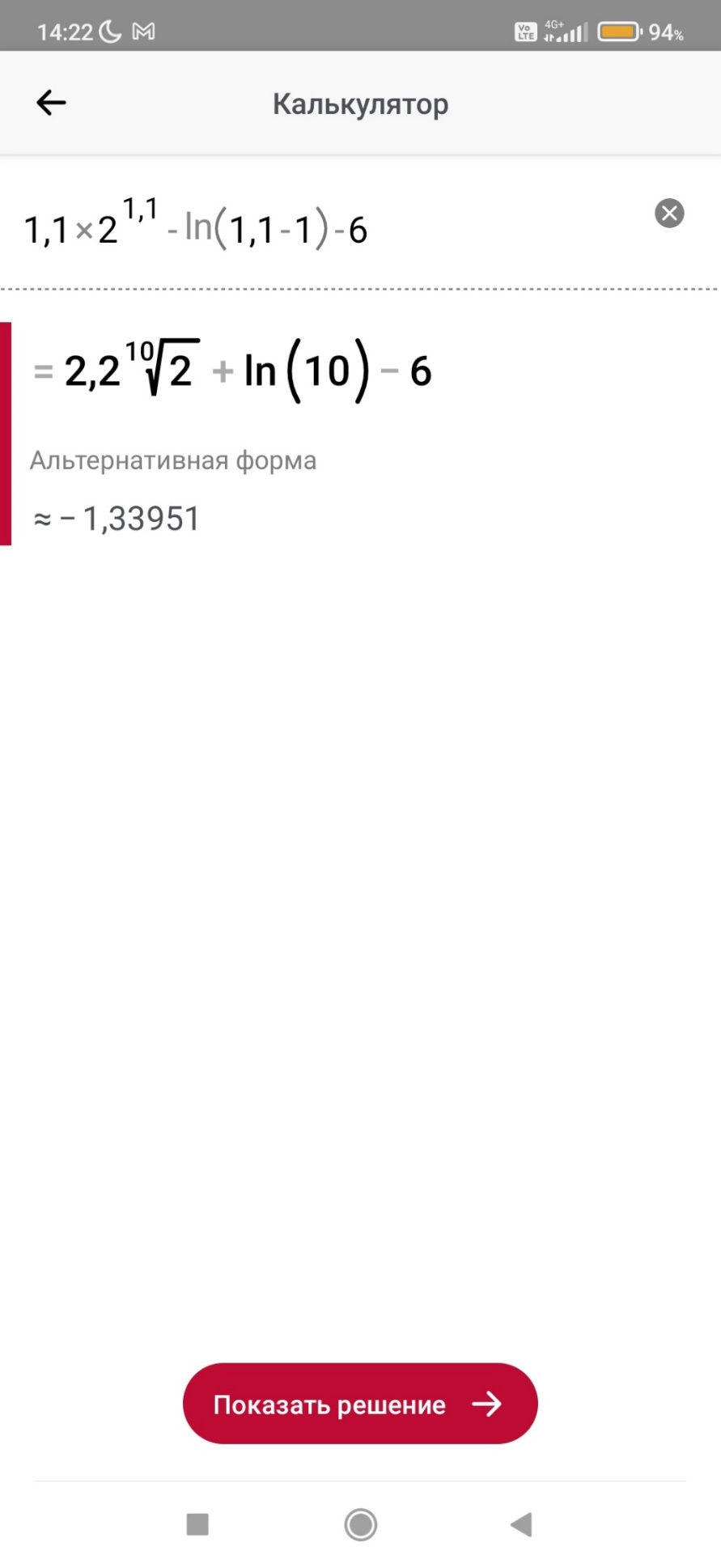


Рис. 12. Проверка результатов 3

## Результаты работы

В результате работы была получена программа для вычисления заданной функции (рис. 13).

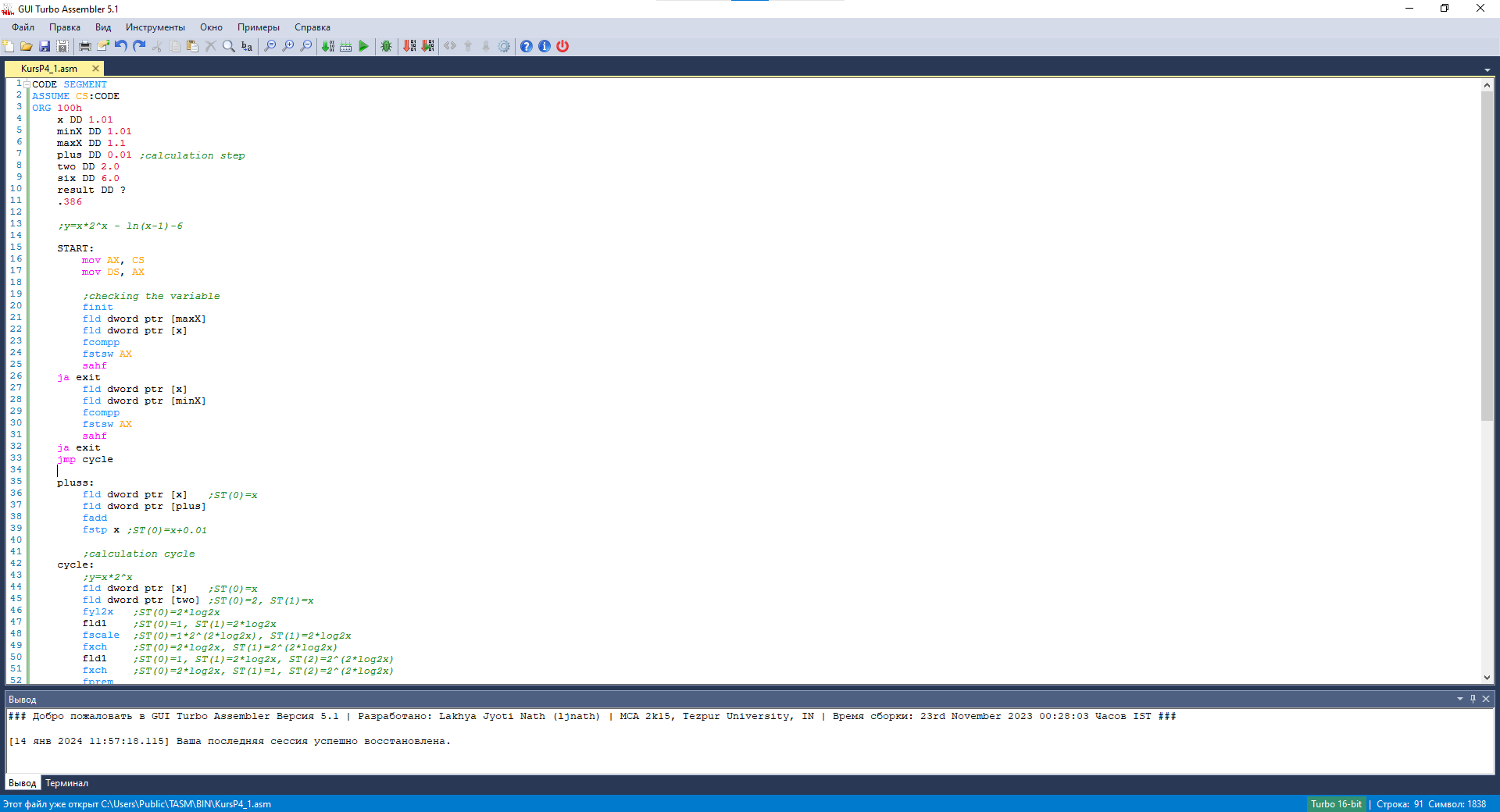


Рис. 13. Общий вид программы

# Заключение

В процессе выполнения курсовой работы были изучены базовые возможности разработки с использованием языка ассемблера и разработана программа, вычисляющая значение функции вида на отрезке a = 1.01, b = 1.1, на данном языке. Одним из основных её достоинств является открытый исходный код, который позволяет пользователю изучать алгоритм работы программы и вносить какие-либо изменения в её структуре.

# Список литературы

[1] Крупник А. Б. К84 Изучаем Ассемблер — СПб.: Питер, 2005. — 249 с.: ил

[2] Иван Скаляров. Изучаем ассемблер за 7 дней.

[3] <https://ru.wikipedia.org/wiki/Язык_ассемблера>

# Приложение 1

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Институт информационных технологий\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

наименование института (факультета)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_МПО ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

наименование кафедры

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Программирование на ассемблере\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ,

д.т.н., профессор \_\_\_\_\_\_\_\_\_\_ Ершов Е.В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ НИЗКОГО УРОВНЯ

Техническое задание на курсовую работу

Листов \_7\_

Руководитель Виноградова Л. Н. \_\_

Ф.И.О. преподавателя

Исполнитель

студент студент гр. 1ПИб-02-1оп-22

группа

\_\_\_\_\_\_Микуцких Г.А.\_\_\_\_

Фамилия, имя, отчество

\_2023\_ год

Введение

Проблема вычислений сложных функций, исключающих ошибки на любом из этапов, стоит достаточно остро. Необходимо иметь возможность быстрой проверки проведённых вычислений и поиска этапа, на котором и допущена ошибка, в качестве самопроверки.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине "Программирование на ассемблере", выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения: 16 декабря 2023 года.

Наименование темы разработки: «Программирование на языке низкого уровня».

1. Назначение разработки

Программа позволит выполнить вычисление заданной функции.

1. Требования к программе

3.1. Требования к функциональным характеристикам

В процессе разработки было утверждено: программа должна вычислять значение функции вида  на отрезке a = 1.01, b = 1.1.

3.2. Требования к надежности

Работа программы должна соответствовать следующим требованиям:

* программа принимает данные вещественного типа;
* шаг вычислений 0.01;
* вычисление значения функции с точностью до десятитысячных.

3.3. Условия эксплуатации

Пользователь может эксплуатировать программу сразу после установки необходимого ПО (ПО должно поставляться вместе с программой).

3.4. Требования к составу и параметрам технических средств

ЭВМ должна соответствовать следующим параметрам:

* исходные данные и результаты представляются в виде текста и графических изображений;
* время отклика системы не более 1 секунды;
* минимальные системные требования:
* 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц или выше;
* 1 ГБ (для 32-разрядного процессора) или 2 ГБ (для 64-разрядного процессора) ОЗУ, 16 ГБ (для 32-разрядной системы) или 20 ГБ (для 64-разрядной системы) свободного места на жестком диске;
* графическое устройство DirectX 9 с драйвером WDDM 1.0 или более поздней версии;
* поддержка клавиатуры, компьютерной мыши, монитора разрешением 1920×1080 пикселей.

3.5. Требования к информационной и программной совместимости

Программа должна корректно работать на операционной системе Windows 7 и выше в приложении «Turbo Assembler» («TASM»).

3.6. Требования к маркировке и упаковке

Архив с программой и «TASM» будет распространяться в социальной сети «ВКонтакте» в личных чатах и на USB-флеш-накопителе.

3.7. Требования к транспортированию и хранению

Всем пользователям запрещается осуществлять действия под видом оригинального ПО, нарушающие статью 273 «Создание, использование и распространение вредоносных компьютерных программ».

3.8. Специальные требования

Для эффективной работы с программой рекомендуется наличие опыта нахождения в компьютерной и информационной средах и умение в них ориентироваться.

4. Требование к программной документации

4.1. Содержание расчётно-пояснительной записки

Программная документация должна содержать расчётно-пояснительную записку, которая содержит:

Титульный лист

Аннотацию

Оглавление

Введение

1. Описание предметной области

2. Описание созданной программы

Заключение

Список литературы

Приложения.

4.2. Технико-экономические показатели

Требования не предъявляются.

4.3. Требования к оформлению

Элементы курсовой работы оформлены в соответствии с параметрами ниже (табл. П1.1).

Таблица П1.1

Требования к оформлению

|  |  |
| --- | --- |
| Элемент | Требования |
| Документ | 1. Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б.  2. Файлы предъявляются на компакт-диске: РПЗ с ТЗ; программный код.  3. Листы и диск в конверте вложены в пластиковую папку скоросшивателя. |
| Страницы | 1. Ориентация – книжная; отдельные страницы, при необходимости, альбомная.  2. Поля: верхнее, нижнее – по 2 см, левое – 3 см, правое – 1 см. |
| Абзацы | Межстрочный интервал – 1.5, перед и после абзаца – 0. |
| Шрифты | Кегль – 14. В таблицах шрифт 12. Шрифт программного кода – 8 (возможно в 2 колонки). |
| Рисунки | Подписывается под ним по центру: Рис. Х. Название  В приложениях: Рис. П1.3. Название |
| Таблицы | 1. Подписывается: над таблицей, выравнивание по правому: «Таблица Х».  2. В следующей строке по центру Название  3. Надписи в «шапке» (имена столбцов, полей) – по центру.  4. В теле таблицы (записи) текстовые значения – выровнены по левому краю, числа, даты – по правому. |

5. Стадии и этапы разработки

Курсовая работа и программа будут разрабатываться в течение следующих этапов (табл. П1.2):

Таблица П1.2

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование  этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Оформление технического задания | 13 декабря 2023 г. | Составлено техническое задание |  |
| Логическое проектирование | 13 – 23 декабря 2023 г. | Составлен алгоритм работы программы |  |
| Физическое проектирование | 23 декабря 2023 г. – 5 января 2024 г. | Программа, реализующая поставленную задачу, написана |  |
| Тестирование |  |  |  |
| Оформление расчётно-поясни-тельной записки | 23 декабря 2023 г. – 14 января 2024 г. | Оформлена расчётно-поясни-тельная записка |  |

6. Порядок контроля и приемки

Курсовая работа будет оформляться в течение этапов в табл. П1.3.

Таблица П1.3

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование  контрольного этапа  выполнения  курсовой работы | Сроки  контроля | Результат выполнения | Отметка о приемке  результата  контрольного этапа |
| Сдача технического задания | 13 декабря 2023 г. | Техническое задание принято |  |
| Демонстрация программы | 5 января 2024 г. | Программа, реали-зующая постав-ленную задачу, принята |  |
| Сдача расчётно-пояснительной записки | 15 января 2024 г. | Расчётно-поясни-тельная записка принята |  |
| Защита курсовой работы | 16 января 2024 г. | Курсовая работа защищена |  |

# Приложение 2

Руководство пользователя.

Общие сведения о программе

Программа написана на языке ассемблера и вычисляет значение функции вида на отрезке a = 1.01, b = 1.1.

Описание установки

Пройдите установку программы «Turbo Assembler», два раза нажав на файл «GUI Turbo Assembler v5.1.msi», который находится в одной папке с программой. Переместите файл «KursP4\_1.asm» по пути: «папка\_установки\_Turbo\_Assembler» → «BIN». Пример установки показан на рис. П2.1–П2.7 (в примере «Turbo Assembler» установлен в папку «Пользователи» («Users») → «Общие» («Public») → «TASM»):

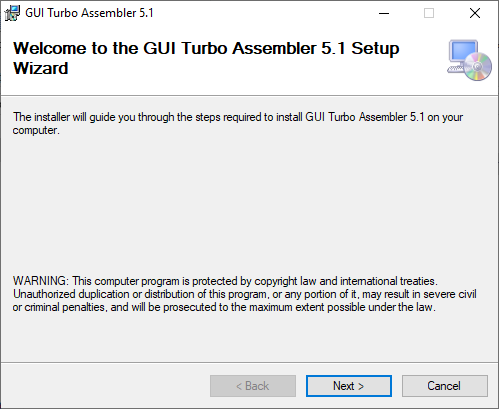


Рис. П2.1. Установка «Turbo Assembler»

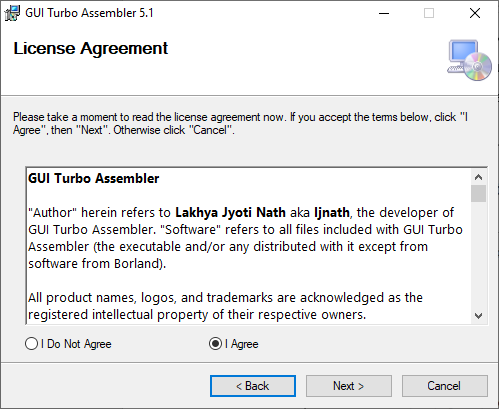


Рис. П2.2. Лицензионное соглашение

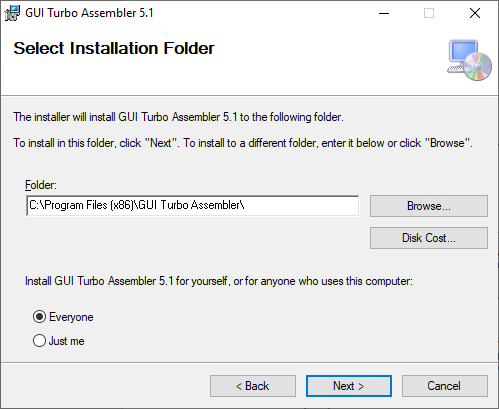


Рис. П2.3. Выбор папки установки «Turbo Assembler»

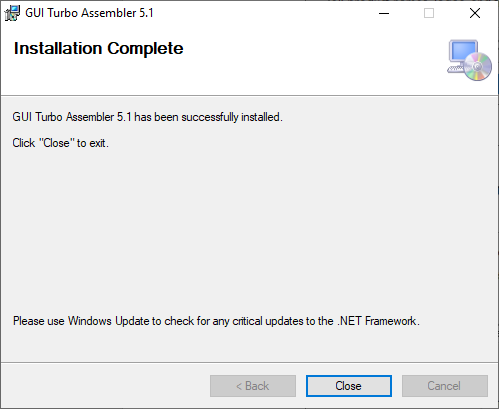


Рис. П2.4. Установка завершена

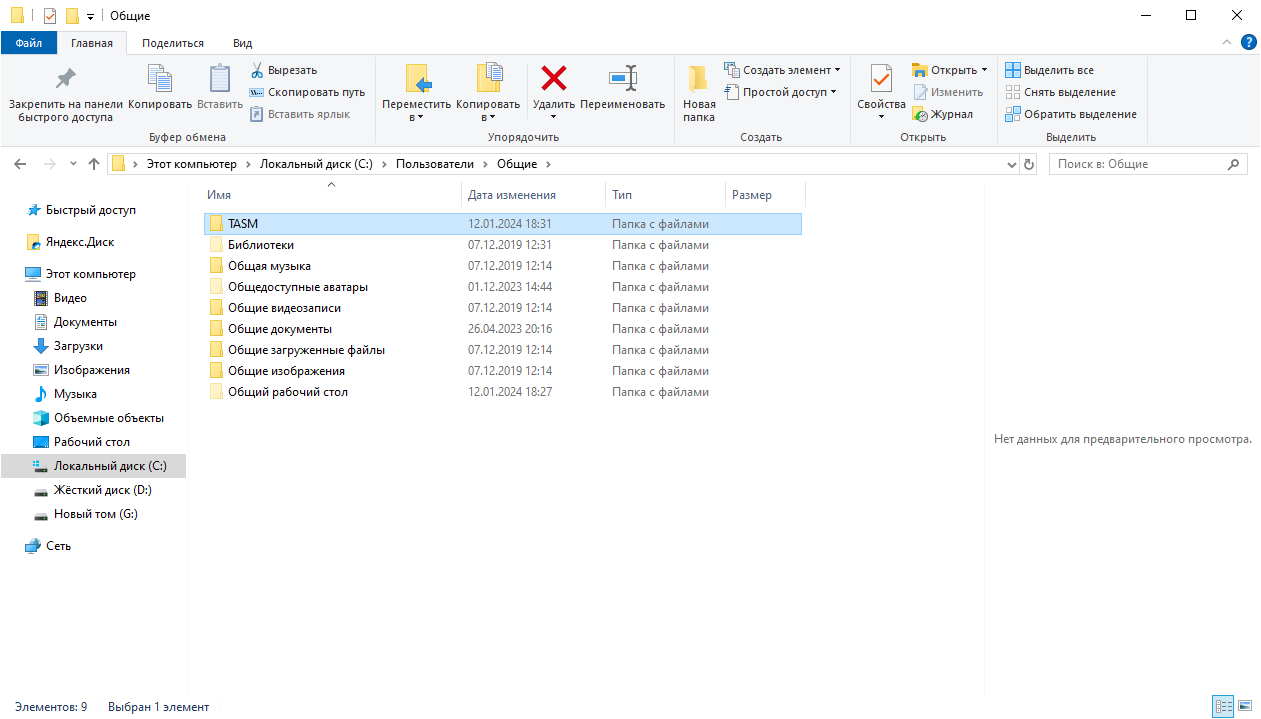


Рис. П2.5. Пример папки установки «Turbo Assembler»

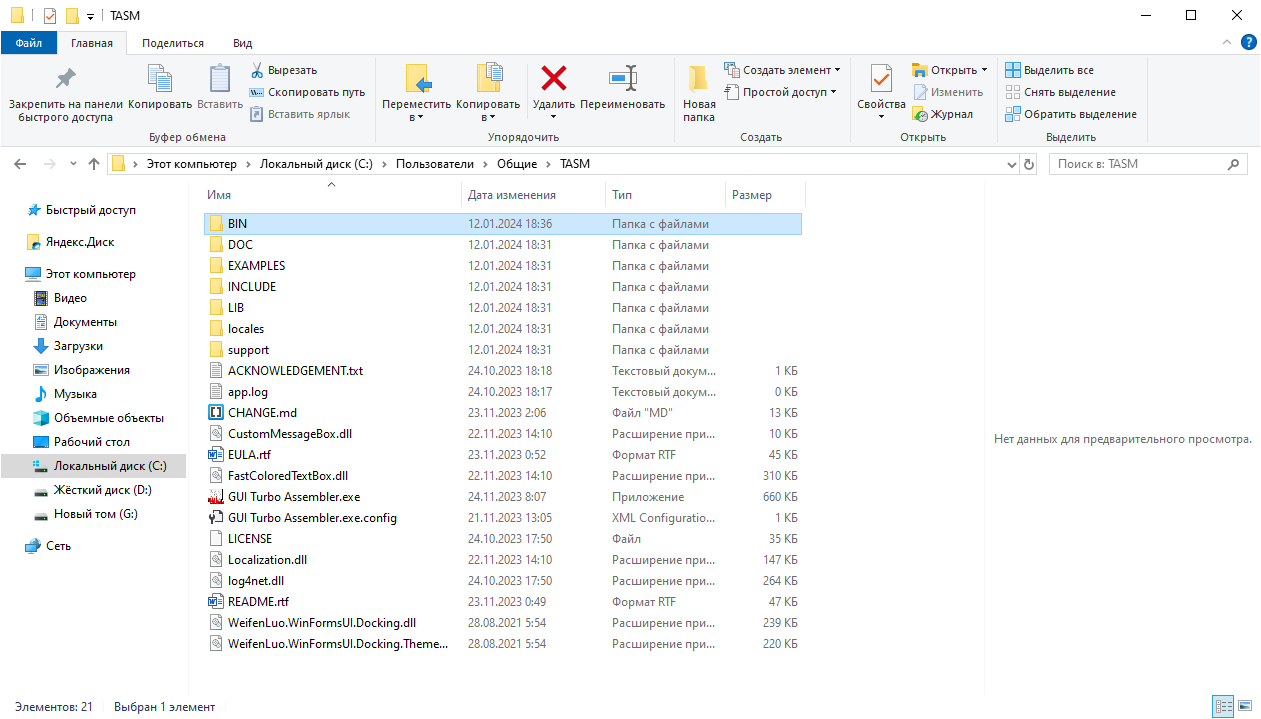


Рис. П2.6. Папка «BIN»

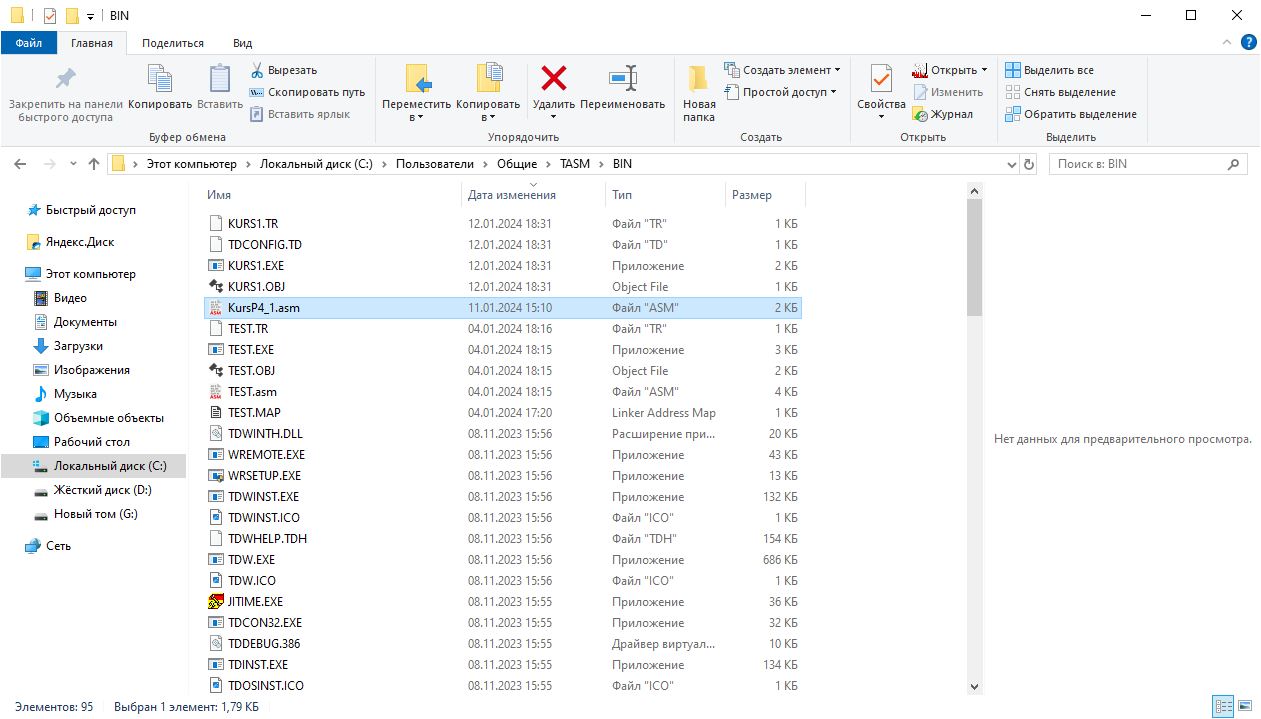


Рис. П2.7. Файл в папке «BIN»

Описание запуска

Откройте файл «KursP4\_1.asm» с помощью программы «Turbo Assembler». Нажмите на кнопку «Отладка исполняемого файла», нажмите «Да» на всех появившихся сообщениях (рис. П2.8 –П2.12).

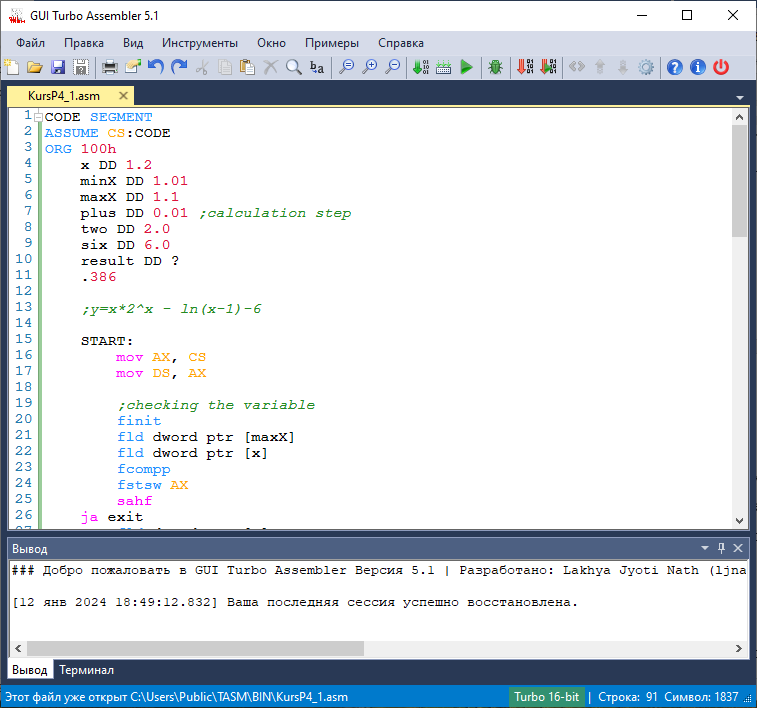


Рис. П2.8. Программа «Turbo Assembler»

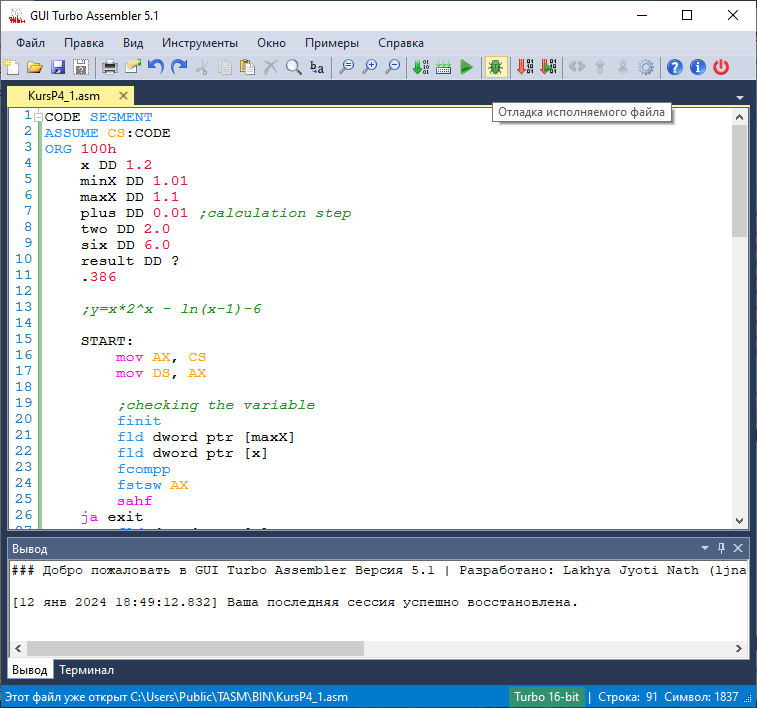


Рис. П2.9. Запуск отладчика «TASM»

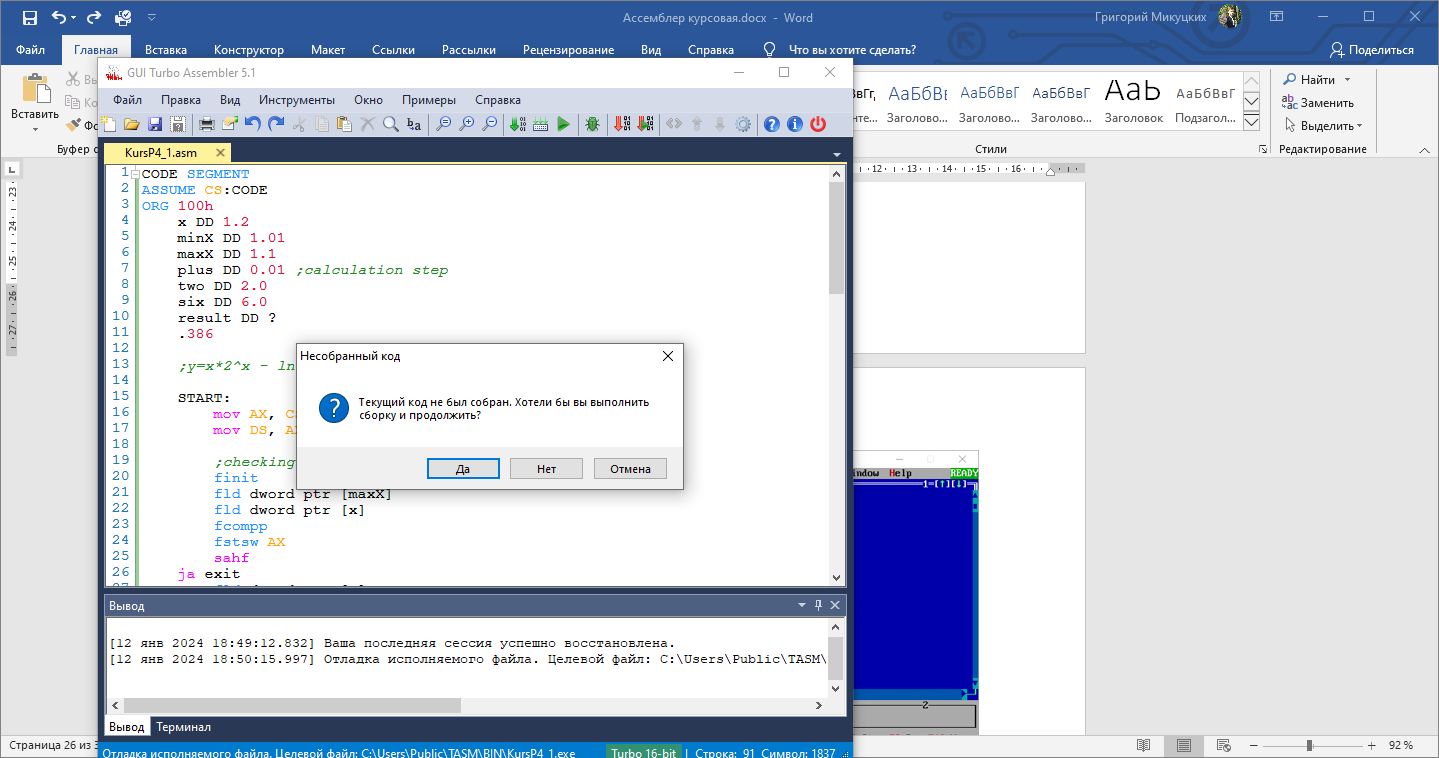


Рис. П2.10. Предупреждение перед запуском № 1

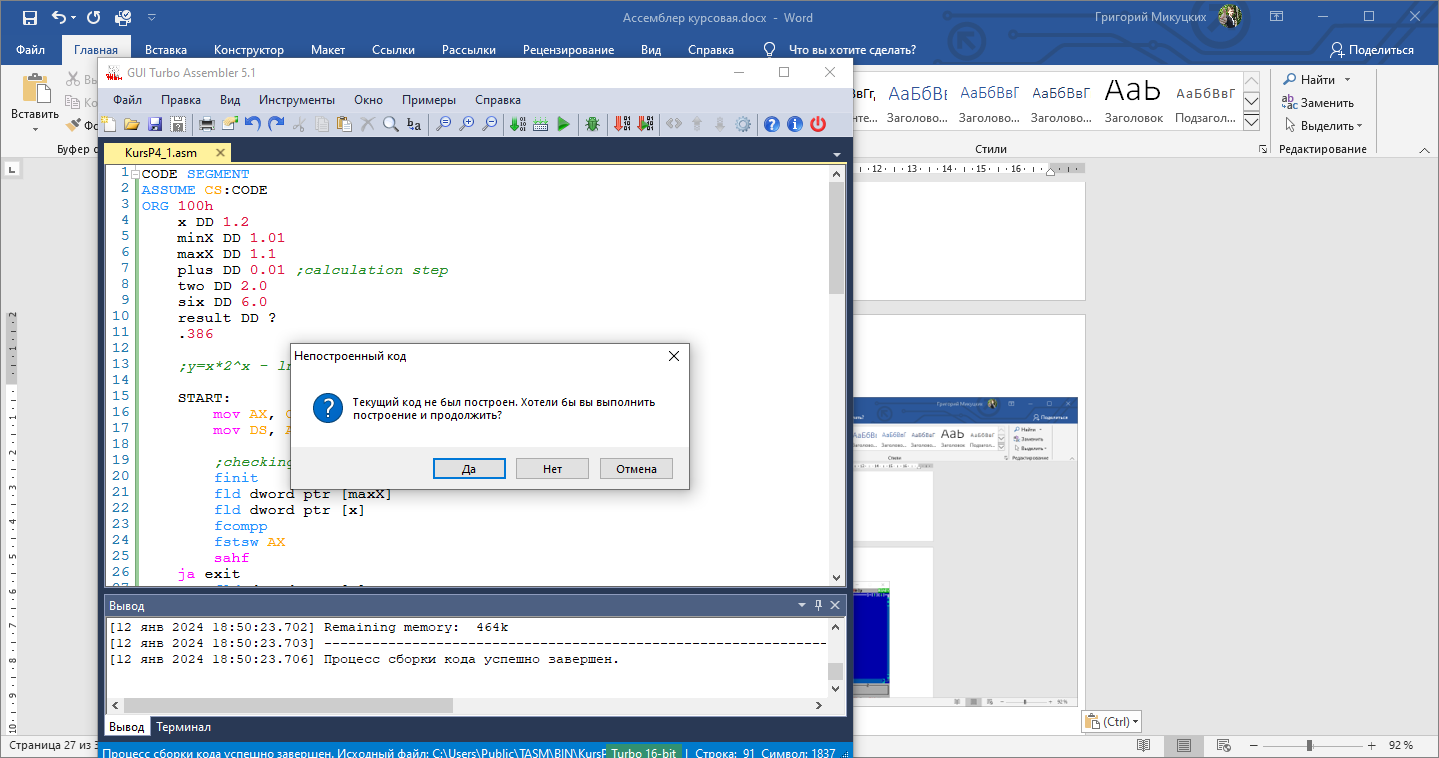


Рис. П2.11. Предупреждение перед запуском № 2

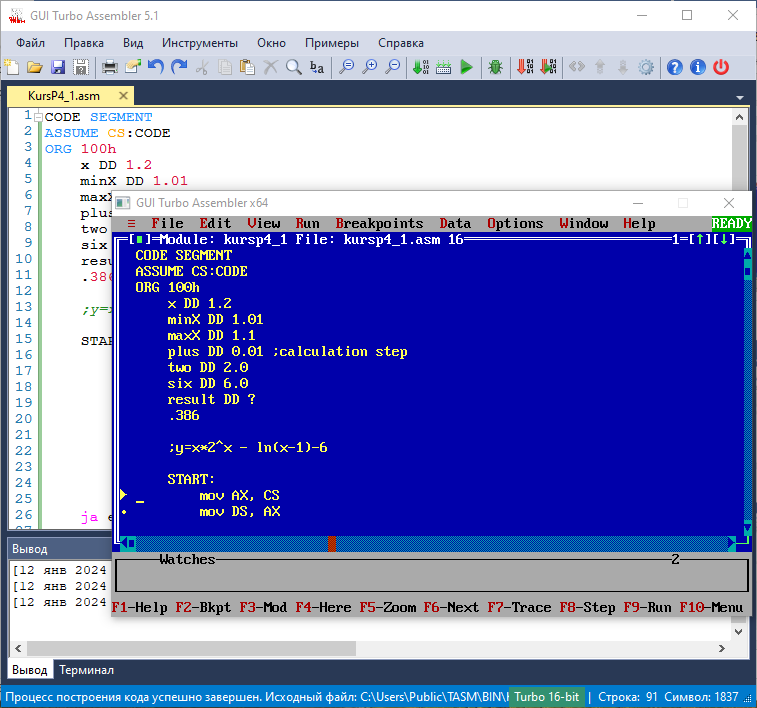


Рис. П2.12. Отладчик «TASM»

Инструкции по работе

После запуска отладчика мышкой или сочетанием клавиш «Alt + V» откройте раздел «View», мышью или стрелками «←→↑↓» на клавиатуре выберите пункт «Numeric processor». Откроется окно, показывающее значения, находящиеся в стеке на данный момент (рис. П2.13–П2.14).

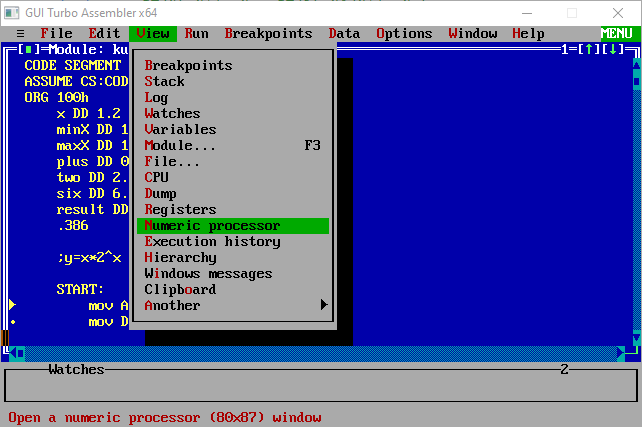


Рис. П2.13. Вкладка «Numeric processor»

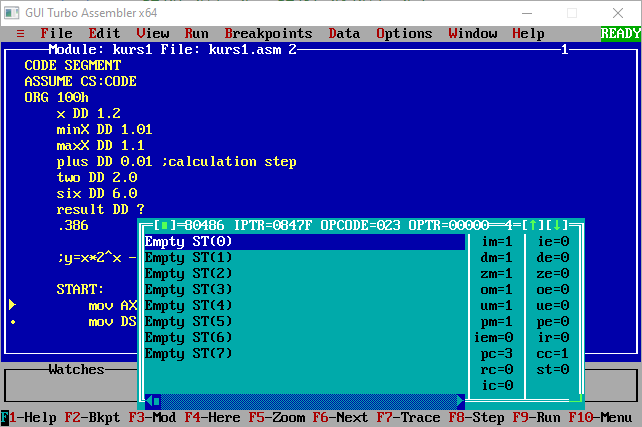


Рис. П2.14. Содержимое стека сопроцессора

С помощью клавиши «F7» или «F8» программа будет построчно выполняться. Когда выполнение программы будет на 74 строке, в вершине стека сопроцессора можно будет увидеть результат вычислений (рис. П2.15– П2.16).

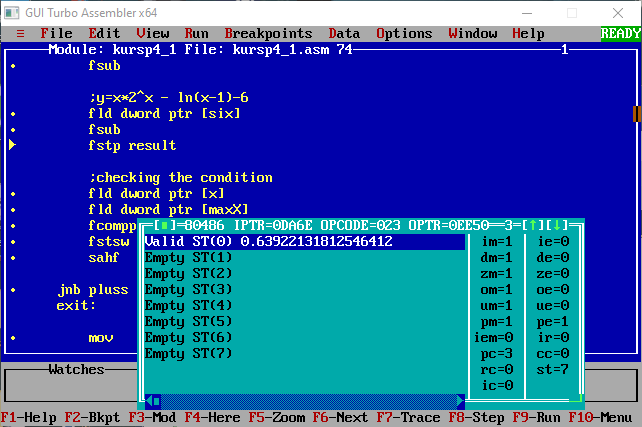


Рис. П2.15. Результат в стеке сопроцессора

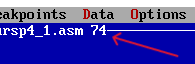


Рис. П2.16. Счётчик строк выполняемой программы

Сообщения пользователю

Отсутствуют

# Приложение 3

Текст программы.

CODE SEGMENT

ASSUME CS:CODE

ORG 100h

x DD 1.01

minX DD 1.01

maxX DD 1.1

plus DD 0.01 ;calculation step

two DD 2.0

six DD 6.0

result DD ?

.386

;y=x\*2^x - ln(x-1)-6

START:

mov AX, CS

mov DS, AX

;checking the variable

finit

fld dword ptr [maxX]

fld dword ptr [x]

fcompp

fstsw AX

sahf

ja exit

fld dword ptr [x]

fld dword ptr [minX]

fcompp

fstsw AX

sahf

ja exit

jmp cycle

pluss:

fld dword ptr [x] ;ST(0)=x

fld dword ptr [plus]

fadd

fstp x ;ST(0)=x+0.01

;calculation cycle

cycle:

;y=x\*2^x

fld dword ptr [x] ;ST(0)=x

fld dword ptr [two] ;ST(0)=2, ST(1)=x

fyl2x ;ST(0)=2\*log2x

fld1 ;ST(0)=1, ST(1)=2\*log2x

fscale ;ST(0)=1\*2^(2\*log2x), ST(1)=2\*log2x

fxch ;ST(0)=2\*log2x, ST(1)=2^(2\*log2x)

fld1 ;ST(0)=1, ST(1)=2\*log2x, ST(2)=2^(2\*log2x)

fxch ;ST(0)=2\*log2x, ST(1)=1, ST(2)=2^(2\*log2x)

fprem

fmul

f2xm1

fld1

fadd

fmul ;ST(0)=2^x

fmul x

;y=x\*2^x - ln(x-1)

fld dword ptr [x]

fld1

fsub

fld1

fxch

fyl2x

fldln2

fmul

fsub

;y=x\*2^x - ln(x-1)-6

fld dword ptr [six]

fsub

fstp result

;checking the condition

fld dword ptr [x]

fld dword ptr [maxX]

fcompp

fstsw AX

sahf

jnb pluss

exit:

mov AX,4C00H

int 21H

ret

CODE ENDS

END START