|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«*Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по лабораторной работе № 2**

**Дисциплина:** Машинно-зависимые языки и основы компиляции

**Название лабораторной работы:** Программирование целочисленных

вычислений



Студент гр. ИУ6-45Б **\_\_\_\_**19.03.2024**\_\_\_\_ \_\_\_**И.А.Дулина**\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_** С.С.Данилюк **\_\_\_\_**

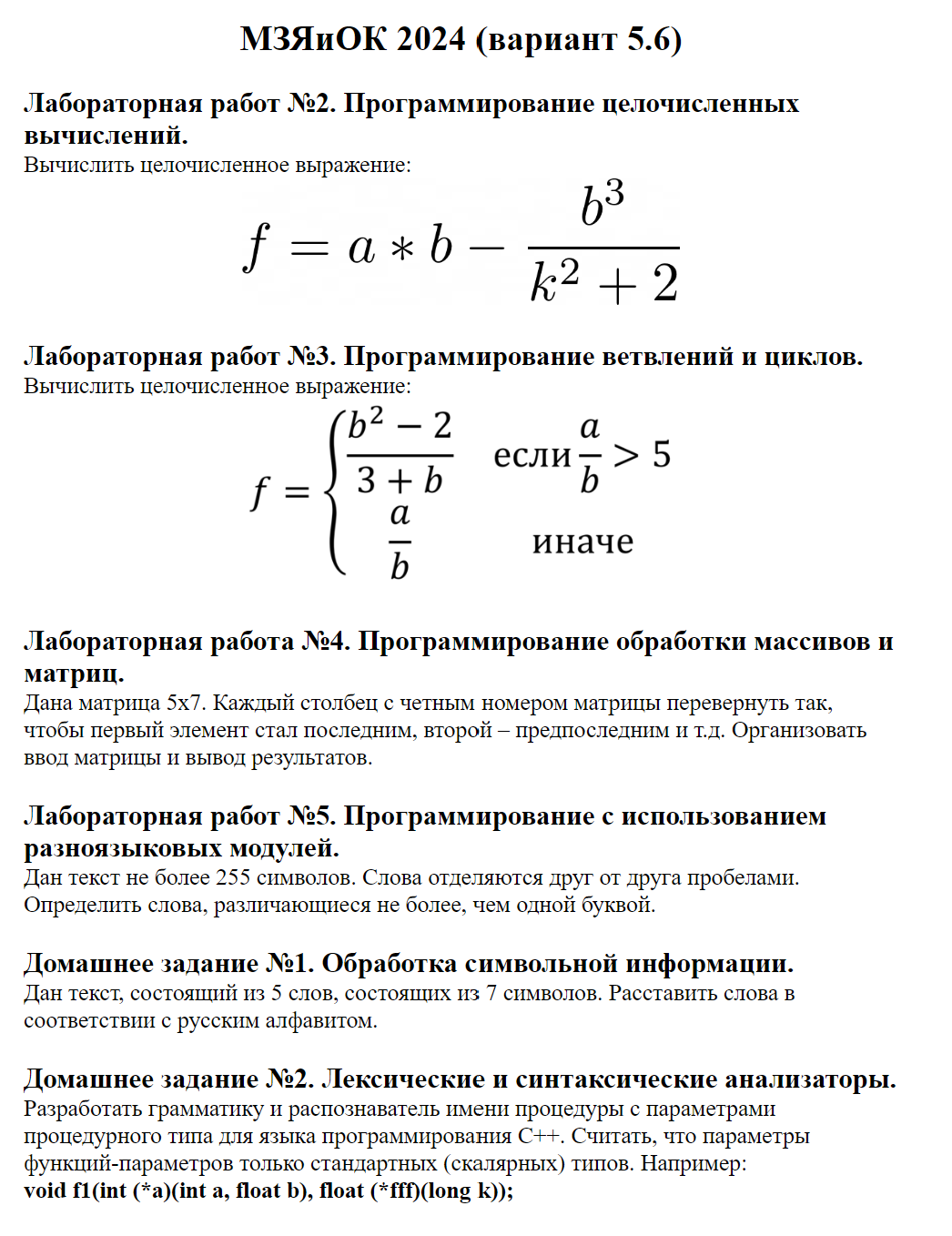
(Подпись, дата) (И.О. Фамилия)

Москва, 2024

**Вариант 5.6**

**Цель работы:** изучение форматов машинных команд, команд целочисленной арифметики ассемблера и программирование целочисленных вычислений.

**Задание**: вычислить целочисленное выражение:



В листинге 1 показан код программы.

Листинг 1 – Код программы

|  |
| --- |
| section .data  InputA db "Input A", 10  lenMsgA equ $-InputA  InputB db "Input B", 10  lenMsgB equ $-InputB  InputK db "Input K", 10  lenMsgK equ $-InputK  ResultMsg db "Result = ", 10  lenMsgResult equ $-ResultMsg  ErrorStr db "Error: Invalid input format", 10  lenError equ $-ErrorStr  section .bss  InBuf resb 10 ; буфер для вводимой строки  lenIn equ $-InBuf  OutBuf resb 10  lenOut equ $-OutBuf  A resd 1  B resd 1  K resd 1  F resd 1  section .text  global \_start  \_start:  ;input A  ;write  mov eax, 4 ; системная функция 4 (write)  mov ebx, 1 ; дескриптор файла stdout=1  mov ecx, InputA ; адрес выводимой строки  mov edx, lenMsgA ; длина выводимой строки  int 80h ; вызов системной функции  ; read  call Buffer  mov esi, InBuf  call StrToInt  cmp ebx, 0  jne Error  mov [A], eax  ;input B  ;write  mov eax, 4 ; системная функция 4 (write)  mov ebx, 1 ; дескриптор файла stdout=1  mov ecx, InputB ; адрес выводимой строки  mov edx, lenMsgB ; длина выводимой строки  int 80h ; вызов системной функции  ; read  call Buffer  mov esi, InBuf  call StrToInt  cmp ebx, 0  jne Error  mov [B], eax  ;input K  ;write  mov eax, 4 ; системная функция 4 (write)  mov ebx, 1 ; дескриптор файла stdout=1  mov ecx, InputK ; адрес выводимой строки  mov edx, lenMsgK ; длина выводимой строки  int 80h ; вызов системной функции  ; read  call Buffer  mov esi, InBuf  call StrToInt  cmp ebx, 0  jne Error  mov [K], eax  ;program  mov eax, [K]  imul eax ; k^2  add eax, 2  mov ECX, EAX; сохраним значение знаминателя  mov eax, [B]  imul eax ; B^2  mov ebx, [B]  imul ebx ; B^3  idiv ecx ; AX = B^3 / K^2+2  mov ecx, eax ;сохраним значение дроби  mov eax, [A]  imul ebx  sub eax, ecx  mov [F], eax  int 80h ; вызов системной функции  ;output  mov eax, 4 ; системная функция 4 (write)  mov ebx, 1 ; дескриптор файла stdout=1  mov ecx, ResultMsg ; адрес выводимой строки  mov edx, lenMsgResult ; длина строки  int 80h ; вызов системной функции  ;вывод F  mov eax, [F]  mov esi, OutBuf  call IntToStr  mov esi, eax  mov eax, 4 ; системная функция 1 (write)  mov ebx, 1 ; дескриптор файла stdout=1  mov ecx, OutBuf ; адрес буфера  mov edx, esi ; длина буфера  int 80h ; вызов системной функции  jmp Exit  Error:  mov eax, 4 ; системная функция 4 (write)  mov ebx, 1 ; дескриптор файла stdout=1  mov ecx, ErrorStr ; адрес сообщения об ошибке  mov edx, lenError ; длина сообщения об ошибке  int 80h ; вызов системной функции  ; Выход  Buffer:  mov eax, 3 ; системная функция 3 (read)  mov ebx, 0 ; дескриптор файла stdin=0  mov ecx, InBuf ; адрес буфера ввода  mov edx, lenIn ; размер буфера  int 80h  ret  Exit:  mov eax, 1 ; системная функция 1 (exit)  xor ebx, ebx ; код возврата 0  int 80h ; вызов системной функции  %include "../lib.asm" |

На рисунке 1 показана схема алгоритма.



Рисунок 1 – Схема алгоритма

На рисунке 2 показан пример выполнения программы в консоли с вводом переменных и выводом результата.

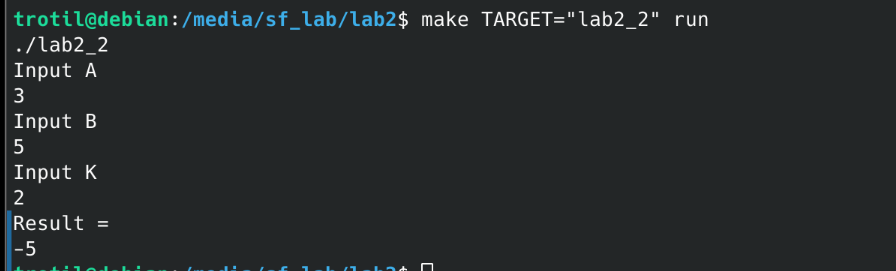


Рисунок 2 – Пример вывода в консоли

На рисунках 3-6 показано выполнение программы в дебаггере при вводе следующих значений: A=4, B=5, K=4.

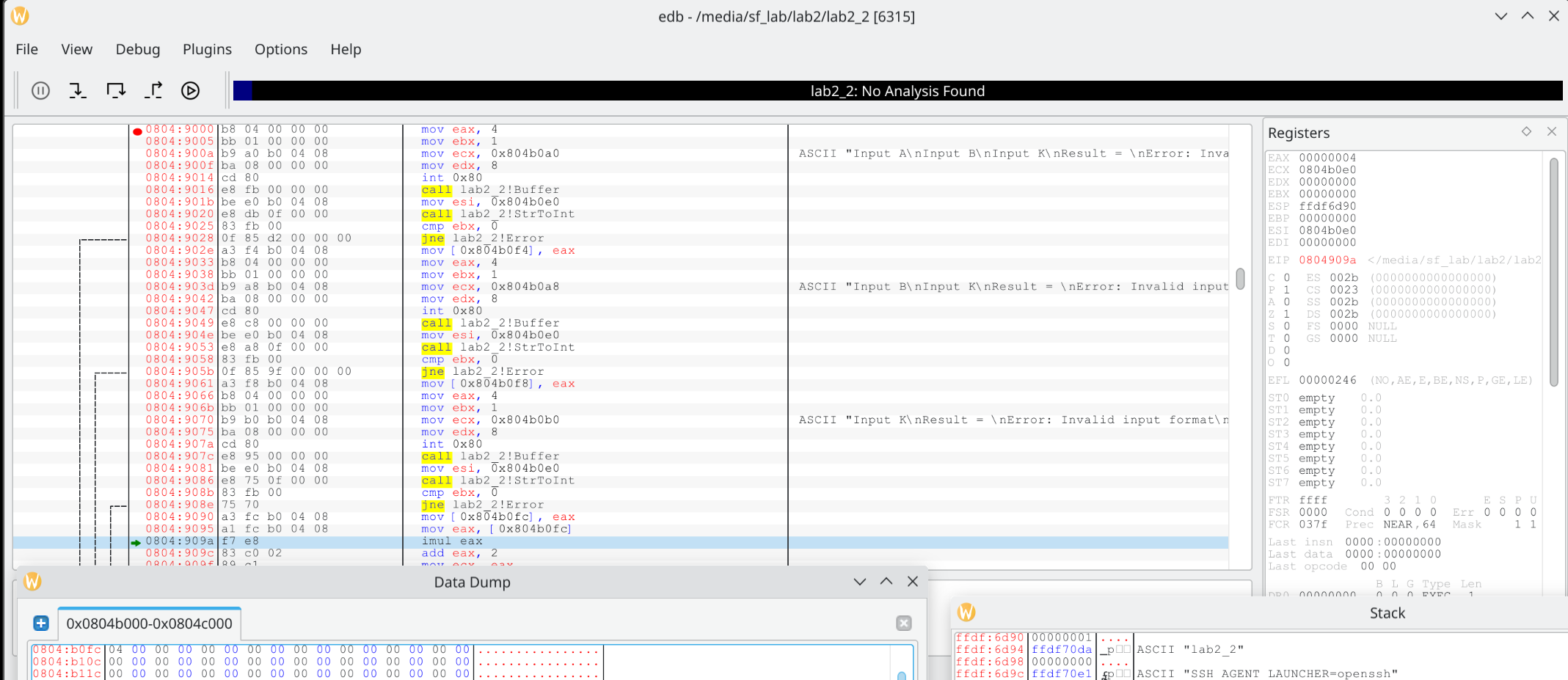


Рисунок 3 – Выполнение команды mov eax, [K]

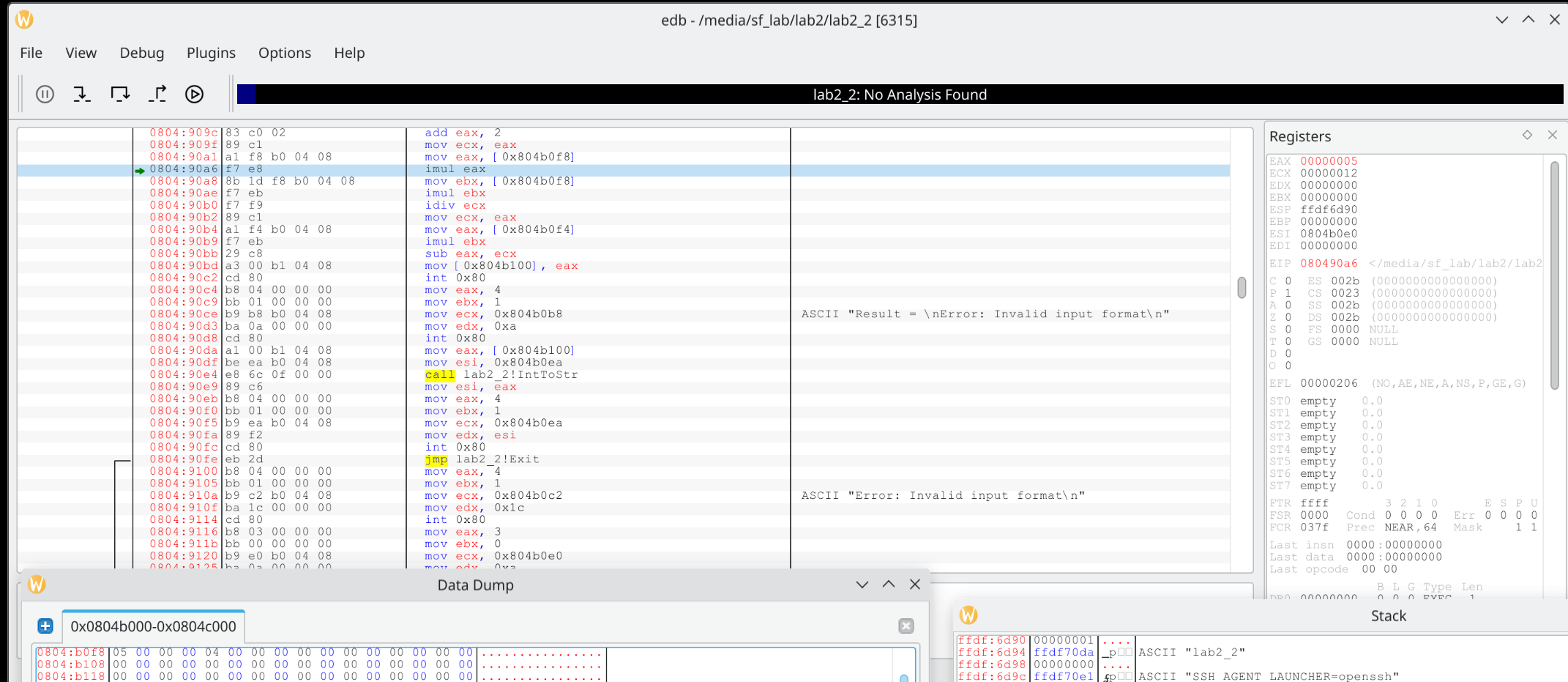


Рисунок 4 – Выполнение команды mov eax, [B]

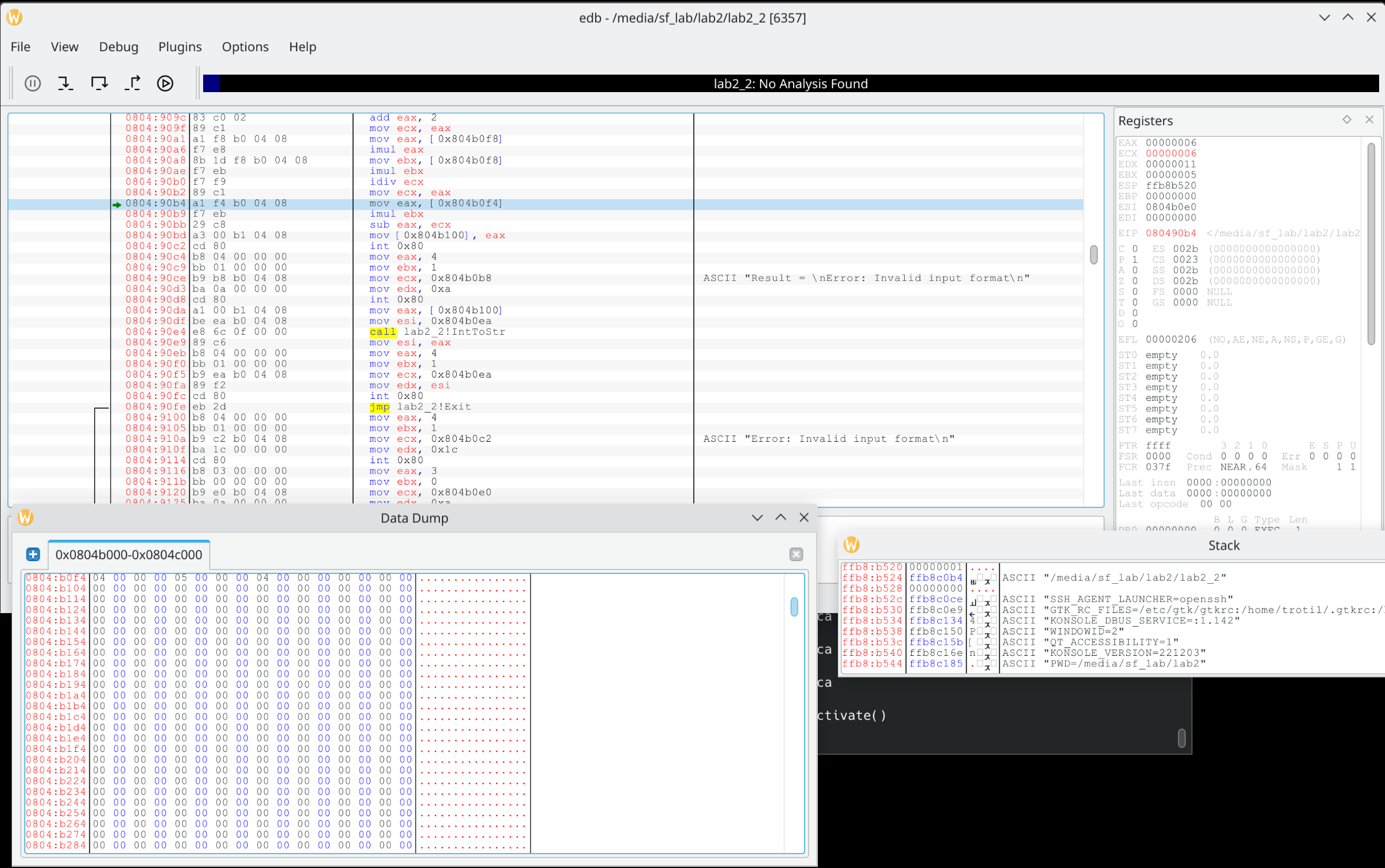


Рисунок 5 – Выполнение команды mov eax, [A]

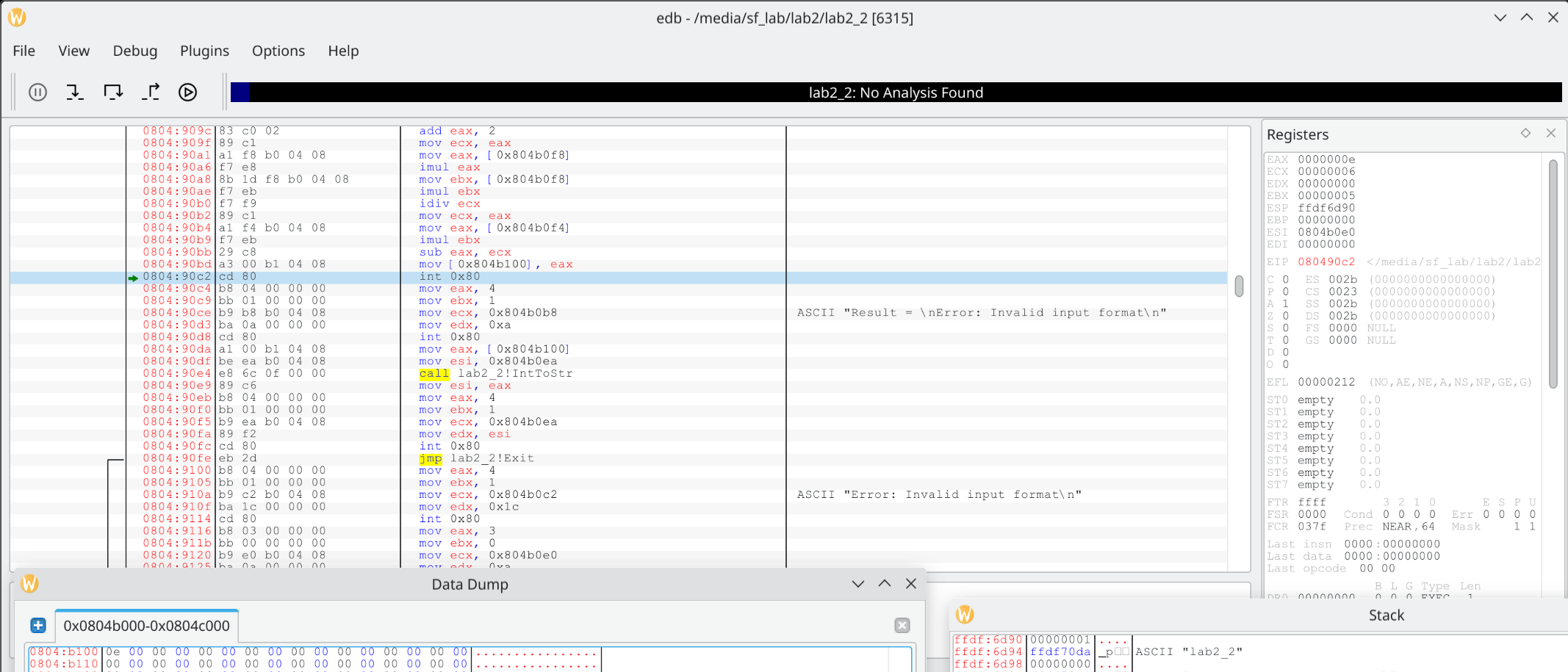


Рисунок 6 – Выполнение команды mov [F], eax

В таблице 1 представлены тесты.

Таблица 1 – Тесты

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Исходные данные | Ожидаемый результат | Полученный результат |
| 1 | A = 5, B = 10, K = 4 | -5,5 | -5 |
| 2 | A = -6, B = 7, K = 3 | -73,18 | -73 |
| 3 | A = 11, B = -30, K = 7 | 199,41 | 199 |

Расшифровка команд mov:

1. 89 C1

На рисунке 7 показана расшифровка команды 89 C1.

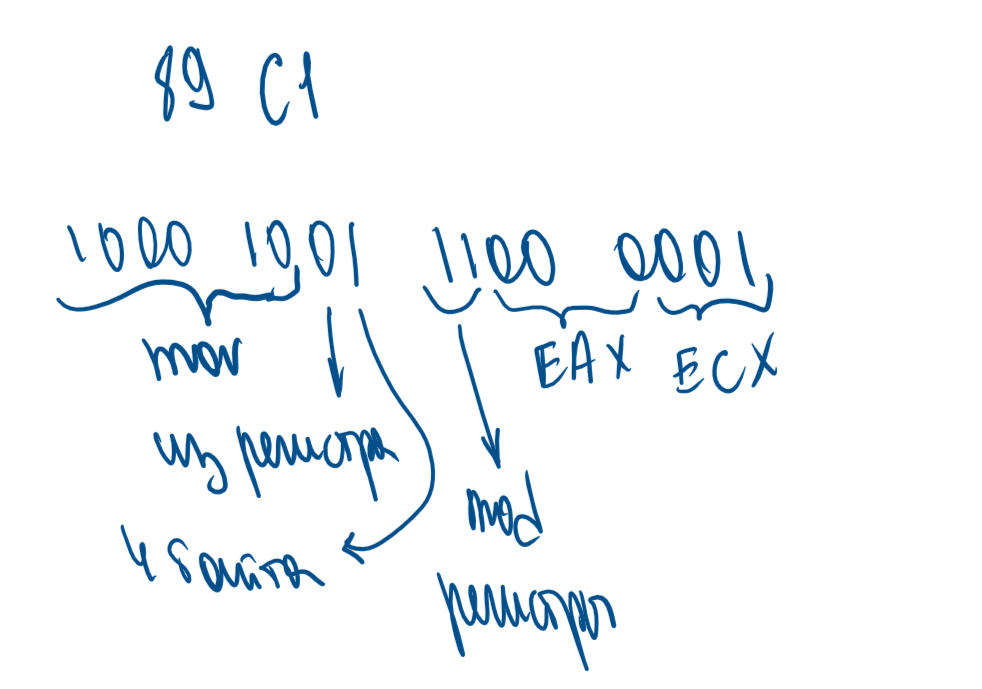


Рисунок 7 – Расшифровка первой команды mov

mov ecx, eax

1. 8B 1D F8 B0 04 08

На рисунке 8 показана расшифровка команды 8B 1D F8 B0 04 08.

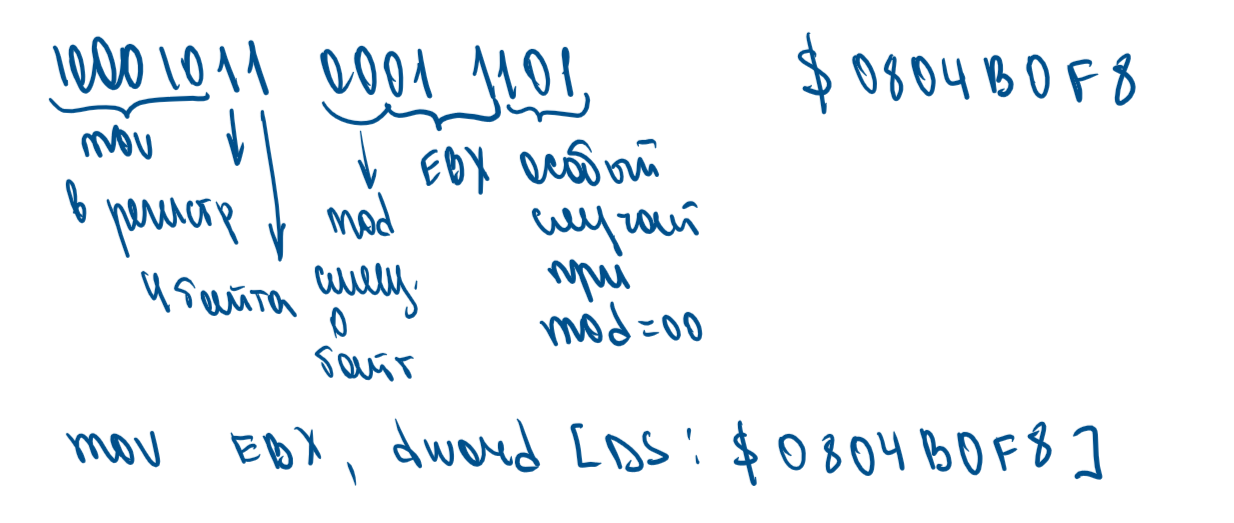


Рисунок 8 – Расшифровка второй команды mov

1. A1 F4 B0 04 08

На рисунке 9 показана расшифровка команды A1 F4 B0 04 08.

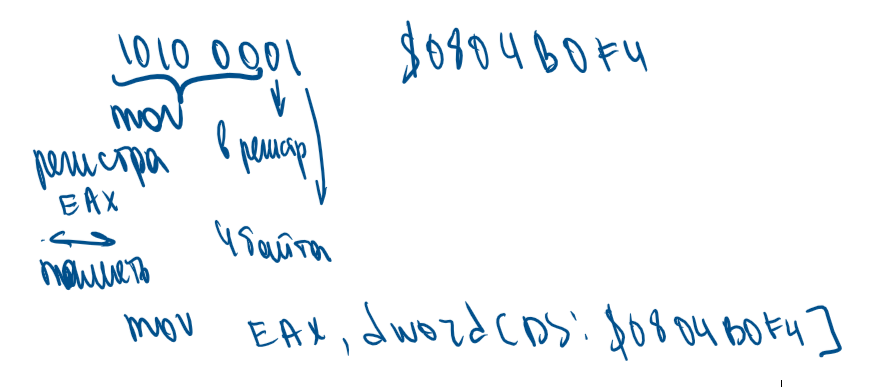


Рисунок 9 – Расшифровка третьей команды mov

**Вывод:** были изучены форматы машинных команд, команд целочисленной арифметики ассемблера и программирование целочисленных вычислений, была реализована программа на языке Assembler по вычислению данного по варианту выражения с вводом значений через консоль и выводом результата

**Контрольные вопросы**

1. Что такое машинная команда? Какие форматы имеют машинные команды процессора IA32? Чем различаются эти форматы?

Машинная команда — это инструкция, предназначенная для выполнения процессором компьютера. Она представляет собой набор битов, который интерпретируется процессором как определенное действие.

Форматы машинных команд в процессорах IA32 включают:

1. Формат команд переменной длины (Variable-Length Instruction Format):
   1. В этом формате длина команды может изменяться.
   2. Опкод (код операции) занимает переменное количество байт в зависимости от конкретной команды.
2. Формат команд фиксированной длины (Fixed-Length Instruction Format):
   1. Здесь каждая команда имеет фиксированную длину.
   2. Опкод занимает фиксированное количество бит.

Различие между форматами заключается в том, как представлены и кодируются инструкции процессора. Формат переменной длины обеспечивает более компактное представление инструкций, что полезно для оптимизации использования памяти. Формат фиксированной длины обычно более прост в реализации и обеспечивает более простую декодировку команд.

1. Назовите мнемоники основных команд целочисленной арифметики. Какие форматы для них можно использовать?
2. Сложение (ADD):

* Форматы команды: ADD dest, src
* Пример: ADD eax, ebx (сложить содержимое регистра ebx с содержимым регистра eax и сохранить результат в регистре eax).

1. Вычитание (SUB):

* Форматы команды: SUB dest, src
* Пример: SUB ecx, edx (вычесть содержимое регистра edx из содержимого регистра ecx и сохранить результат в регистре ecx).

1. Умножение (IMUL):

* Форматы команды: IMUL dest, src (умножение с расширением знака), IMUL dest, src, immediate (умножение с константой)
* Пример: IMUL esi, edi (умножить содержимое регистра edi на содержимое регистра esi и сохранить результат в регистре esi).

1. Деление (IDIV):

* Форматы команды: IDIV operand (деление)
* Пример: IDIV ecx (разделить содержимое регистра edx:eax на значение регистра ecx).

Форматы могут включать в себя указание регистров, констант и других операндов для выполнения нужной арифметической операции.

1. Сформулируйте основные правила построения линейной программы вычисления заданного выражения.
   1. Секции:

- Программа обычно состоит из различных секций, таких как .data (для данных), .bss (для неинициализированных данных), .text (для кода) и т.д.

- Используйте секции в соответствии с их предназначением.

* 1. Определение данных:

- Используйте директивы .data для определения секции данных.

- Определите переменные и константы, а также задайте начальные значения, если это необходимо.

* 1. Определение кода:

- Используйте секцию .text для определения кода программы.

- Начинайте код с метки \_start, которая является точкой входа программы.

* 1. Использование регистров:

- Изучите регистры, поддерживаемые вашей архитектурой (например, x86), и используйте их в соответствии с требованиями программы.

* 1. Инструкции:

- Используйте соответствующие инструкции для выполнения операций. Например, MOV для перемещения данных, ADD для сложения, SUB для вычитания и т.д.

* 1. Обработка ввода и вывода:

- Используйте соответствующие системные вызовы для ввода и вывода данных, такие как int 0x80 для Linux или другие, зависящие от целевой платформы.

* 1. Завершение программы:

- В конце программы предусмотрите выход, например, вызвав системный вызов для завершения программы.

1. Почему ввод-вывод на языке ассемблера не программируют с использованием соответствующих машинных команд? Какая библиотека используется для организации ввода вывода в данной лабораторной?

Прямые машинные команды могут различаться на разных архитектурах процессоров. В данной лабораторной используется библиотека lib.asm

1. Расскажите, какие операции используют при организации ввода-вывода.

В данном коде на ассемблере NASM реализованы две процедуры: StrToInt и IntToStr, предназначенные для преобразования строкового представления числа в целое число и обратно.

StrToInt:

Вход: ESI – адрес строки, содержащей запись числа и завершающаяся байтом 0x0A (положительные числа должны вводиться без знака, отрицательные – содержать знак в первой позиции).

Выход: EAX – 32-х разрядное число, EBX содержит 0, если преобразование прошло без ошибок, и 1, если в процессе преобразования обнаружен ввод недопустимого символа или введенное число не попадает в заданный интервал.

IntToStr:

Вход: EAX – число, ESI – адрес области памяти для размещения строки результата (7 байт).

Выход: EAX – размер строки результата – запись числа будет прижата к левой границе области по адресу ESI и завершаться байтом 0x0A.