|  |  |
| --- | --- |
| 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 № 1**

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

**Название лабораторной работы:** Изучение среды и отладчика

ассемблера

Студент гр. ИУ6 - 42Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г. Д. Нефедов**

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Москва, 2024

**Цель работы:** изучение процессов создания, запуска и отладки программ

на ассемблере Nasm под управлением операционной системы Linux, а также

особенностей описания и внутреннего представления данных.

1. Ввод заготовки 32-х разрядной программы на ассемблере в текстовый редактор, предсталено на рисунках 1.1:

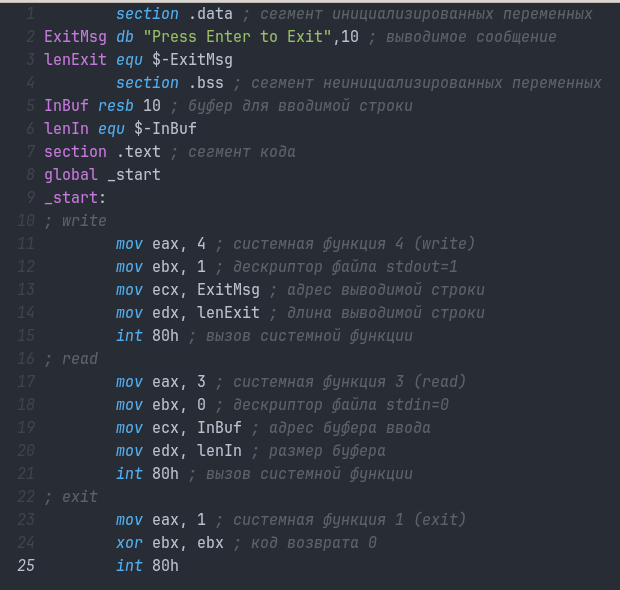
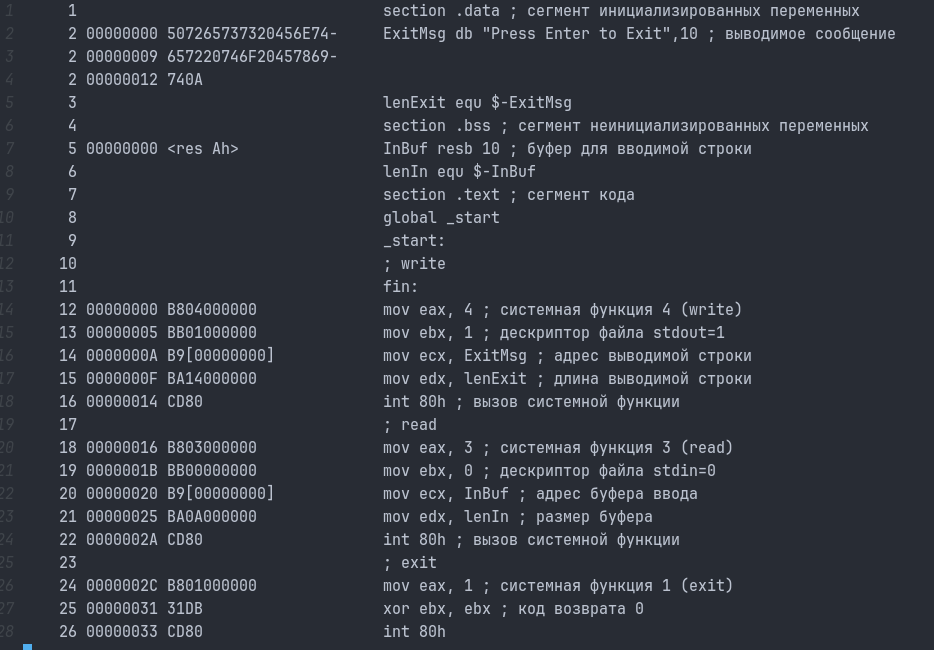


Рисунок 1.1 — Код ассемблера в текстовом редакторе

2. Выполнение трансляции программы с листингом , предсталено на рисунках 1.2:

Рисунок 1.2 — Трансляция программы

В результате выполнения программы сгенерировался файл листинга 1\_zad.list содержащий подробную информацию о процессе трансляции (ассемблирования) исходного кода программы, предсталено на рисунках 1.3.

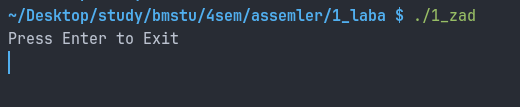
Рисунок 1.3 - Файл листинга 1\_zad.list

Файл листинга представляет собой файл, создаваемый компилятором в процессе компиляции. В этом файле содержится ассемблерный код, представленный в удобном для чтения текстовом формате.

3. Выполнение компановки программы на 64-х разрядном компьютерею, предсталено на рисунках 1.4.

Рисунок 1.4 — Компановка программы

4. Запуск программы на выполнение, предсталено на рисунках 1.5:

Рисунок 1.5 — Запуск программы

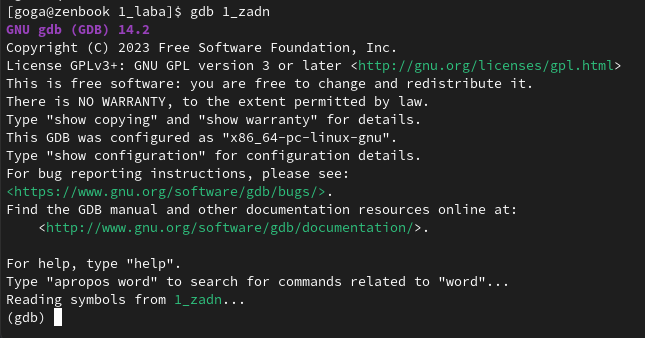


Рисунок 1.6.1— Запуск отладчика

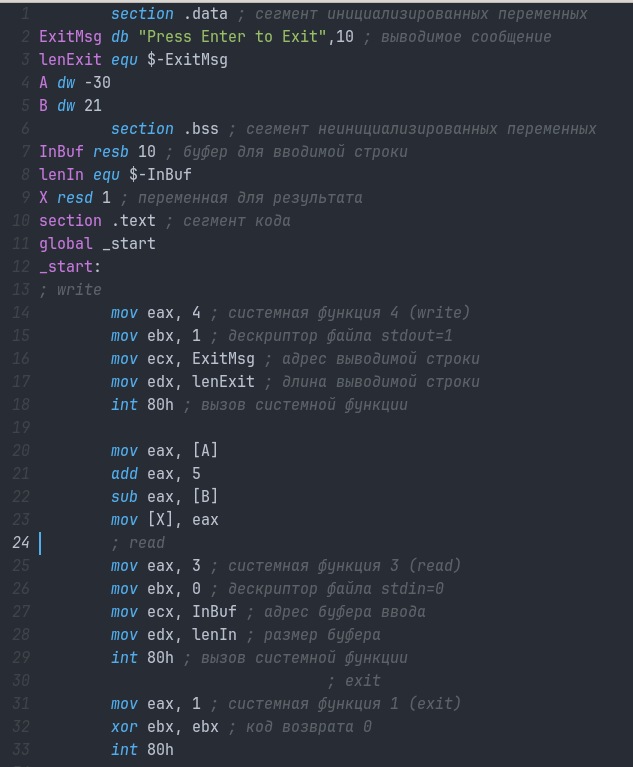
Посмотрим внутреннее представление ExitMsg , предсталено на рисунках 1.6.2:

Рисунок 1.6.2— Внутреннее представление ExitMsg

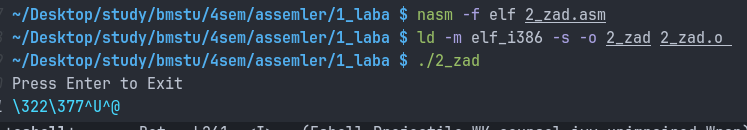
Разберем данное представление данных:

* 50 — буква «P»
* 72 — буква «r»
* 65 — буква «e»
* 73 — буква «s»
* 73 — буква «s»
* 20 — пробел
* 45 — буква «E»
* 6e — буква «n»
* 74 — буква «t»
* 65— буква «e»
* 72— буква «r»
* 20 — пробел
* 74— буква «t»
* 6f— буква «o»
* 20 — пробел
* 45 — буква «E»
* 78 — буква «x»
* 69 — буква «i»
* 74 — буква «t»

5. Допишем программу, чтобы она смогла вычислять выражение: X=A+5-B , предсталено на рисунках 1.7.

Рисунок 1.7 — Код программы

6. Проведем трансляцию, компановку и запуск программы , предсталено на рисунках 1.8.

Рисунок 1.8 — Запуск программы

7. Внутреннее представление данных , предсталено на рисунках 1.9. e2 ff это «-30», 15 это «21»



Рисунок 1.9 — Внутреннее представление данных

Отладим арифметические операции:

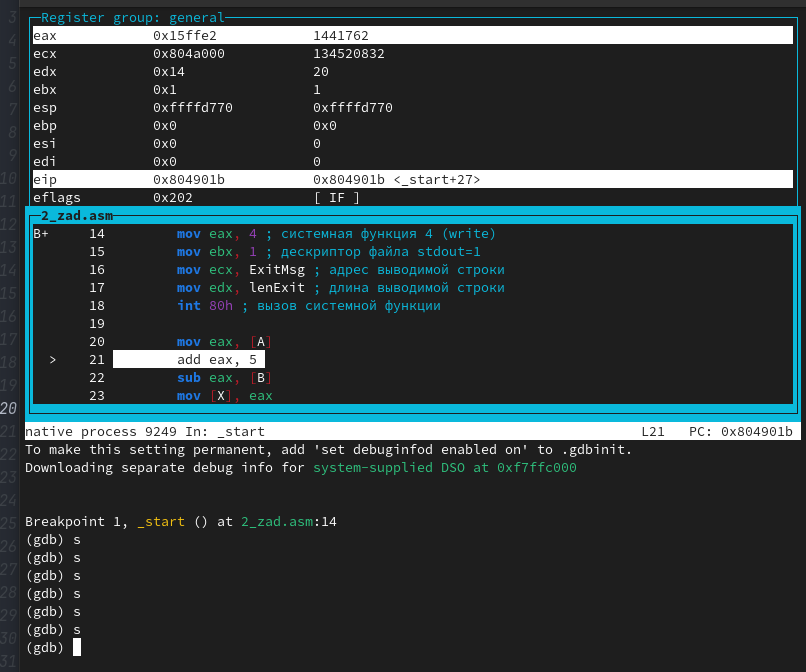
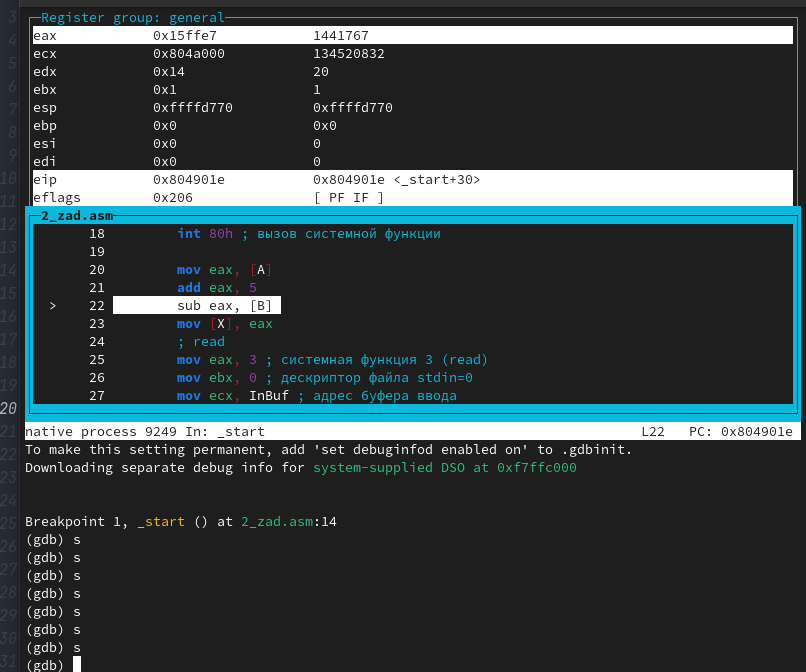
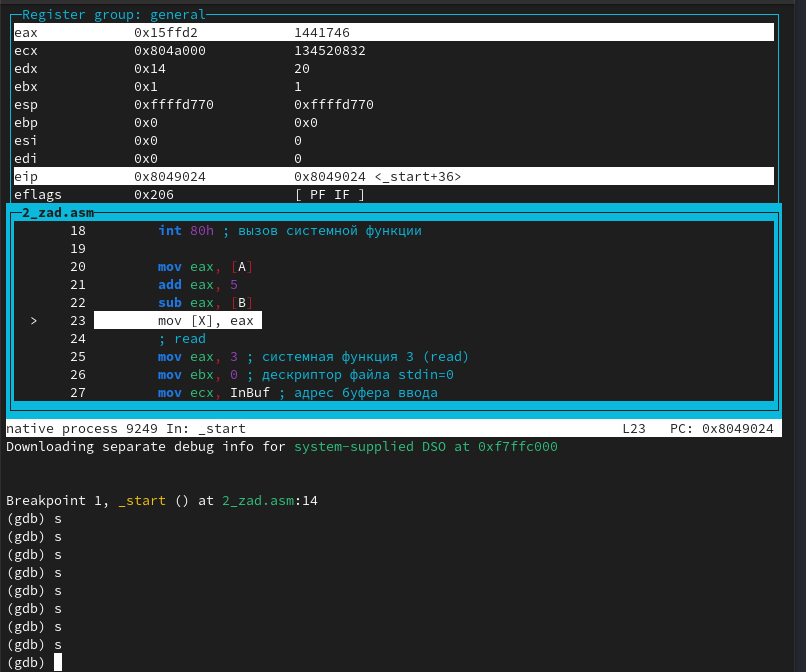
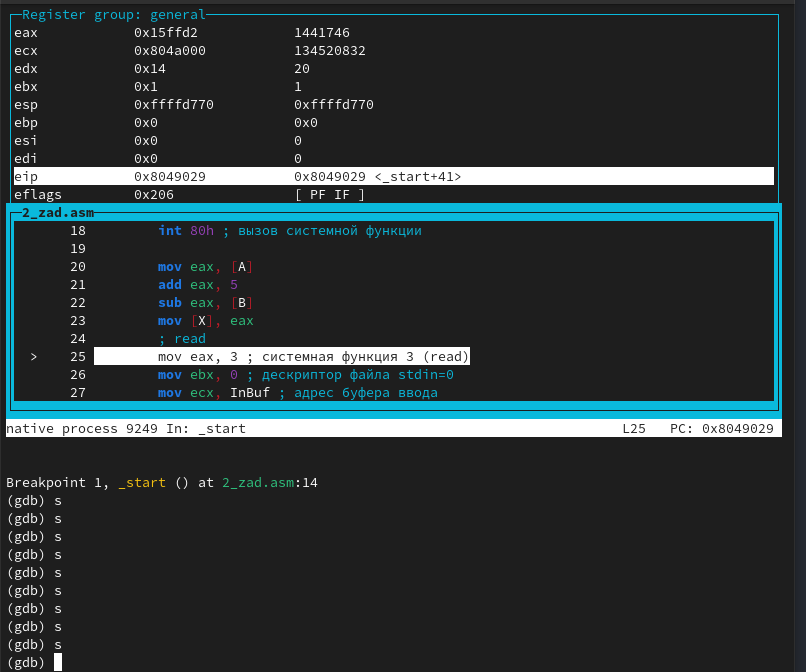
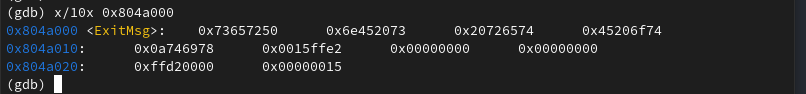


Рисунок 1.10.1— Запись в регистр eax значение переменной А

Рисунок 1.10.2— Прибавили к регистру eax значение 5

Рисунок 1.10.3— Вычли из регистра eax значение переменной B

Рисунок 1.10.4— Записали в переменную X значение регистра eax

Рисунок 1.10.5— Регистры после арифметических действий

8. Добавим некоторые переменные в программу , предсталено на рисунках 1.11:

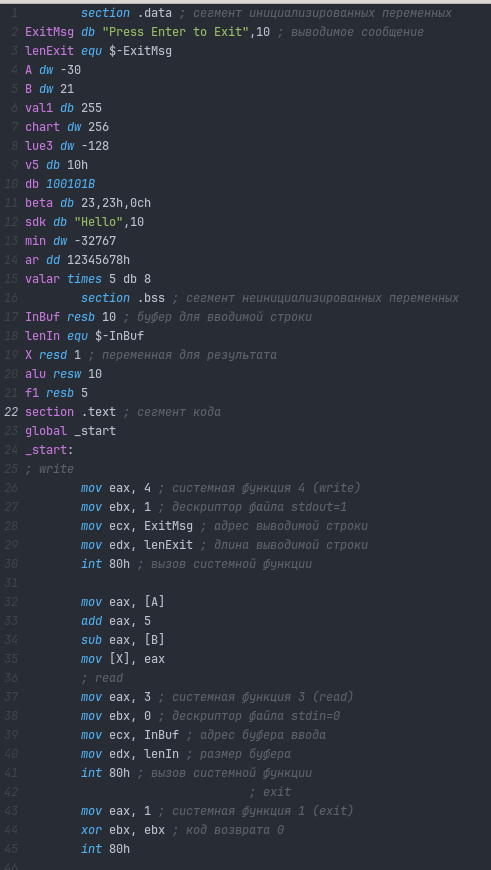


Рисунок 1.11 — Исправленный код

Внутренне представление данных, предсталено на рисунках 1.12, 1.13

Рисунок 1.12.1 — Внутреннее представление данных

Разберем данное представление данных:

0x0015ffe2:

* e2ff — значение переменной А
* 15 00 — значение переменной B

0x800100ff:

* ff — значение переменной val1
* 00 01 — значение переменной chart

0x172510ff:

* 80 ff — значение переменной lue 3
* 10 25 — значение переменной v5

0x65480c23:

* 17 23 0c — значение переменной beta
* 48 — значение переменной sdk («H»)
* 65 — значение переменной sdk («e»)

0x0a6f6c6c:

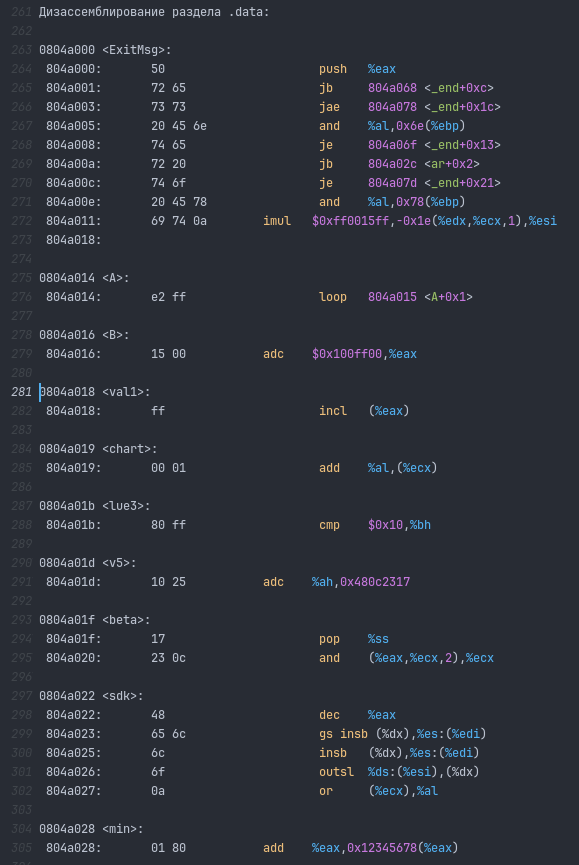
* 6c — значение переменной sdk («l»)
* 6c — значение переменной sdk («l»)
* 6f — значение переменной sdk («o»)
* 0a — значение переменной sdk («\n»)

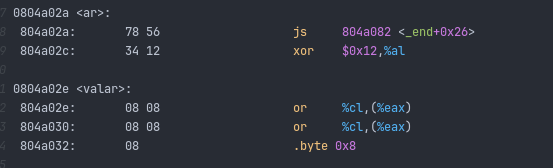
0x56788001:

* 01 80 — значение переменной min
* 78 56 34 12 — значение переменной ar

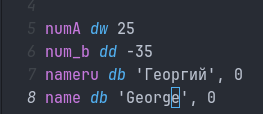
0x08081234:

* 08 08 08 08 08 — значение переменной valar

Рисунок 1.12.2 — Внутреннее представление данных (часть 1)

Рисунок 1.13— Внутреннее представление данных (часть 2)

9. Определим в памяти следующие данные, предсталено на рисунке 1.14:

Рисунок 1.14 — Новые данные

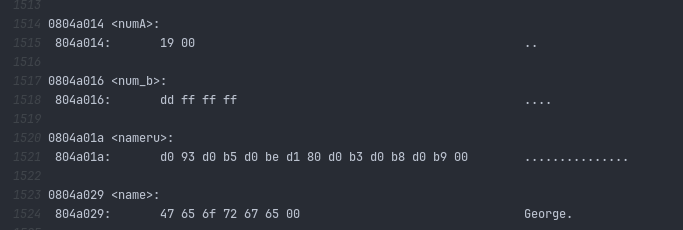
Представление в памяти новых данных (рис. 1.15)

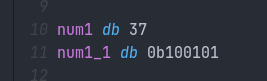
Рисунок 1.15 — Внутреннее представление данных

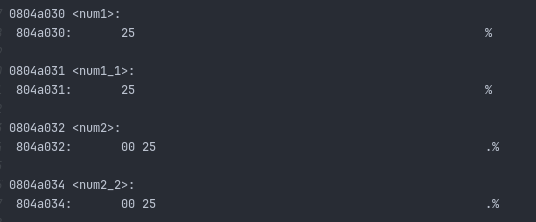
Пояснение:

* nameru содержит русское имя "Георгий" в кодировке UTF-8.
* name содержит латинское имя "George" ASCII кодировке.
* numA содержит шестнадцатиричную запись, которой соответствует число 25
* num\_b содержит шестнадцатиричную запись, которой соответствует число -35

10. Представление в памяти как 00 25 можно поучить записями: (рис. 1.16)

Рисунок 1.16

Рисунок 1.16 — Внутреннее представление данных

Представление в памяти как 25 00 можно поучить записями:

Реальное представление в памяти (рис. 1.18)

Рисунок 1.18 — Внутреннее представление данных

11. Добавим в программу переменную F1=65535 размером слово и

переменную F2= 65535 размером двойное слово.

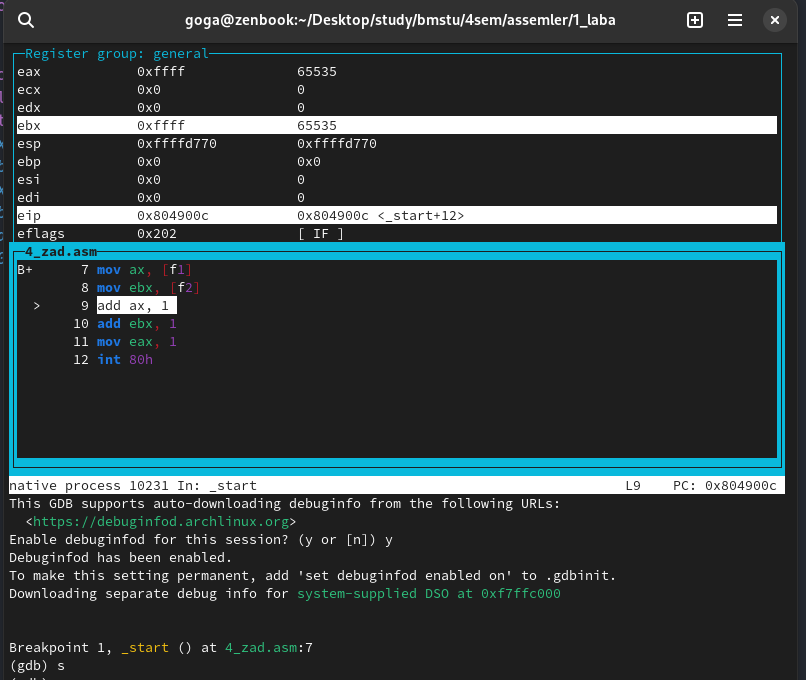
Перейдем в отладчик, предсталено на рисунках 19 - 20:

Рисунок 1.19 — Записали значения переменных в регистры

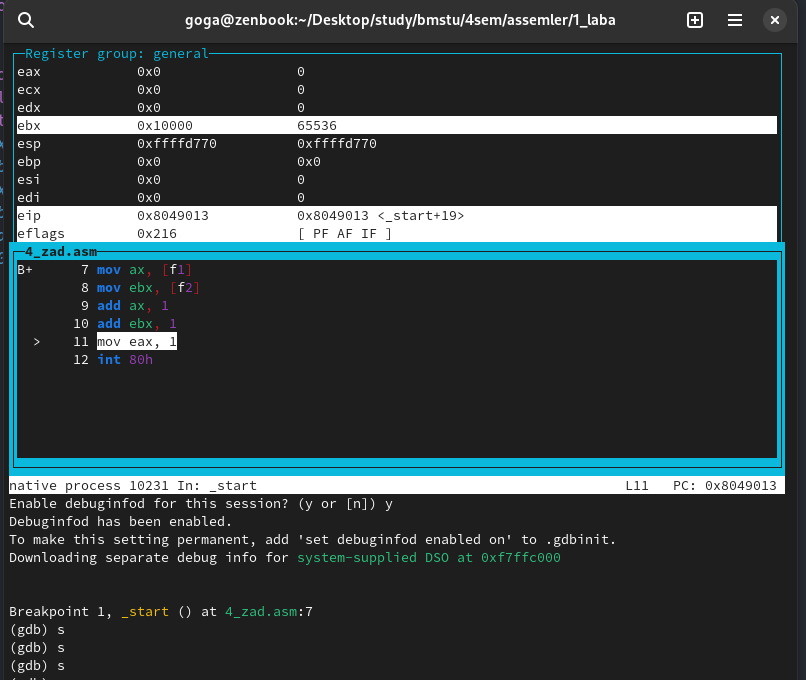
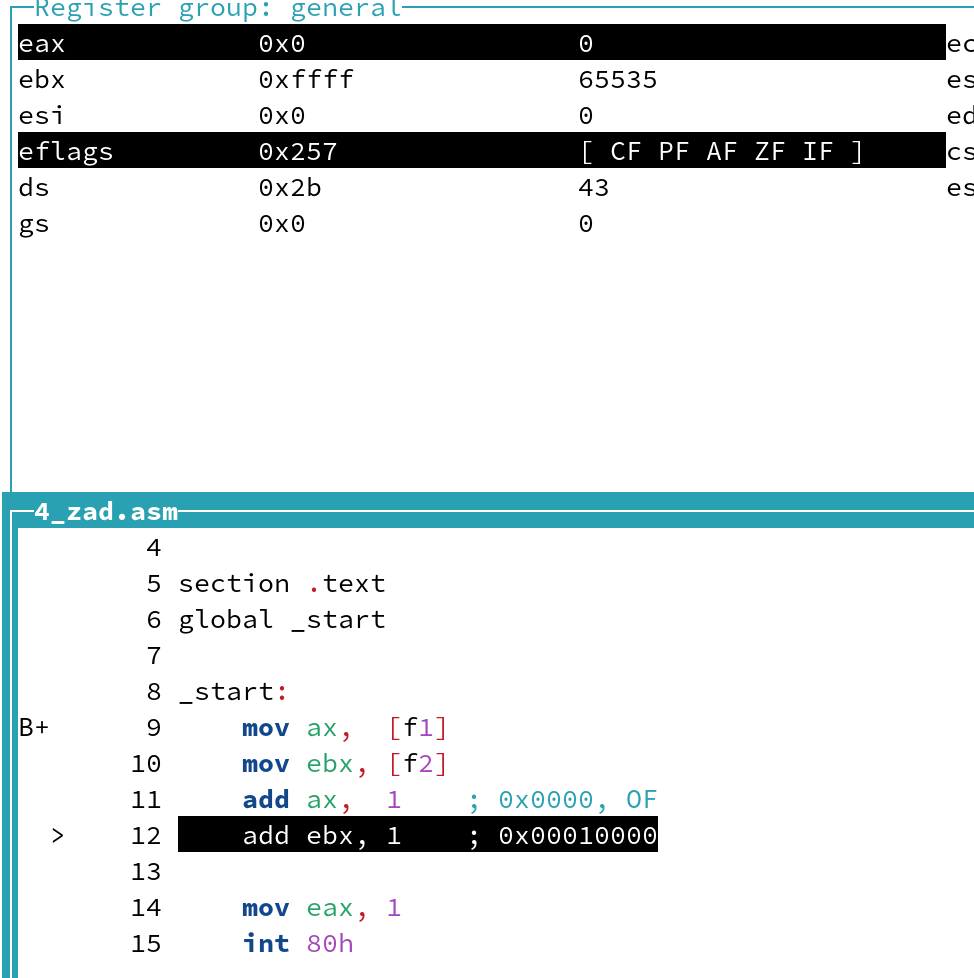


Рисунок 1.20 — Регистр ebx не переполняется, в отличие от ax

Рисунок 1.21 — Регистр ebx переполнился

**Вывод:**

Изучив ассемблер NASM под Linux, мы освоили создание, запуск и отладку программ. Глубоко погрузившись в описание данных и взаимодействие с процессором через прерывания, мы освоили арифметику, работу с памятью и отслеживание переполнений. Этот опыт расширил наши знания о внутренних механизмах компьютера, что пригодится в разработке и оптимизации кода.

**Контрольные вопросы:**

**1. Дайте определение ассемблеру. К какой группе языков он относится?** Ассемблер ­ низкоуровневый язык программирования, посылающий команды процессору. Язык ассемблера относится к машинно­зависимым языкам программирования.

**2. Из каких частей состоит заготовка программы на ассемблере?** Заготовка программы на языке ассемблера состоит из трех частей:

• section .text  (сегмент кода)

• section .data  (сегмент инициализированных данных)

• section .bss  (сегмент неинициализированных данных)

**3.Как запустить программу на ассемблере на выполнение? Что происходит с программой на каждом этапе обработки?**

Для  подготовки  программы  к  выполнению  сперва  вызывают  транслятор  nasm   и  компоновщик  ld.   В  результате  работы  транслятор  создает  объектный  файл,  которые  затем  подается  на  вход  компоновщика, компоновщик формирует исполняемую программу.

**4.** **Назовите основные режимы работы отладчика. Как осуществить пошаговое  выполнение  программы  и  просмотреть  результаты выполнения машинных команд.**

• si  – выполнить шаг с заходом в тело процедуры;

• s  – выполнить шаг, не заходя в тело процедуры.

• r  ­ запуск программы

• br  ­ установить точку останова

• lay  ­ переключиться между слоями

**5.** **В  каком  виде  отладчик  показывает  положительные  и  отрицательные целые числа? Как будут представлены в памяти числа: A dw 5,­5 ? Как те же числа будут выглядеть после загрузки в регистр AX?**

5 в шестнадцатеричной системе счисления равно  00000005 , а −5 будет выглядеть как  fffffffb . В регистре  AX  число 5 будет выглядеть как  0005 , а число −5 будет выглядеть как  fffb .

**6.Каким образом в ассемблере программируются выражения? Составьте фрагмент программы для вычисления С=A+B, где A, В и С – целые числа формата BYTE.**

section .data

A db 1

B db 2

section .bss

C resb 1

section .text

global \_start

\_start:

mov al, [A]

mov ah, [B]

add al, ah

mov byte[C], al