|  |  |
| --- | --- |
| 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. ИУ6-41Б  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Самодурова**

(Подпись, дата) (И.О. Фамилия)

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

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

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

на ассемблере Nasm под управлением операционной системы Linux, а также

особенностей описания и внутреннего представления данных.

1.2.3. Используя меню Пуск, вызовите текстовый редактор Kate (раздел

Офис) и введите заготовку 32-х или 64-х разрядной программы на ассемблере.

Сохраните программу с именем lab1.asm в подкаталоге labs/lab1.

Внимательно изучите структуру программы и способ программирования

системных вызовов для выполнения операций ввода-вывода и завершения

программы. Зафиксируйте текст программы с комментариями в отчете.

1.2.4. Выполните трансляцию программы с листингом. Для этого

- для 32-х разрядной программы следует ввести команду:

... :~/labs/lab1$ nasm -f elf lab1.asm -l lab1.lst

- для 64-х разрядной:

... :~/labs/lab1$ nasm -f elf64 lab1.asm -l lab1.lst

В результате вы должны получить объектный модуль lab1.o и файл

листинга lab1.lst. Убедитесь, что операция прошла без ошибок.

1.2.5. Для компоновки 32-х или 64-х разрядной программы, которая будет

выполняться на компьютере той же размерности, следует ввести:

... :~/labs/lab1 $ ld -o lab1 lab1.o

Для компоновки 32-разрядной программы, которая будет выполняться на

64-х разрядном компьютере, необходимо запросить режим эмуляции:

... :~/labs/lab1 $ ld -m elf\_i386 -o lab1 lab1.o

После команды в окне терминала, если они есть, выводятся сообщения об

ошибках. Если все прошло без ошибок, то в том же каталоге появится файл

исполняемой программы lab1.

1.2.6. Запустите программу на выполнение:

... :~/labs/lab1 $ ./lab1

Запущенная программа должна вывести текст:

Press Enter to Exit

и ожидать нажатия клавиши Enter. После нажатия клавиши Enter выполнение

программы завершится.

Следует иметь в виду, что буфер ввода предусматривает ввод до 10

символов в то время как при нажатии клавиши Enter в буфер записывается

единственный символ с кодом 1010 =0A16. Символ с кодом 1310, заносимый в

буфер ввода в операционных системах семейства Windows, в операционных

системах типа Linux при нажатии на клавишу Enter в буфер не заносится.

1.2.7. Запустите отладчик edb. Для этого следует в окне терминала ввести

команду:

... :~/labs/lab1 $ edb

Средствами графического интерфейса отладчика откройте в нем

исполняемую программу lab1 и проанализируйте, что вы видите в его окне.

Найдите машинное представление программы, ее дисассемблированный код,

содержимое регистров и т.д. Выполните программу по шагам, контролируя

содержимое регистров и оперативной памяти.

Код программы (результат работы программы представлен на рисунке 1):

%include "io64.asm"

section .data

ExitMsg db "Press Enter to Exit",10

lenExit equ $-ExitMsg

A dw -39

B dw 21

section .bss

InBuf resb 10

lenIn equ $-InBuf

X resd 1

section .text ; сегмент кода

global \_start

\_start:

mov rax, 1 ; системная функция 1 (write)

mov rdi, 1 ; дескриптор файла stdout=1

mov rsi, ExitMsg ; адрес выводимой строки

mov rdx, lenExit ; длина строки

syscall ; вызов системной функции

; read

mov rax, 0 ; системная функция 0 (read)

mov rdi, 0 ; дескриптор файла stdin=0

mov rsi, InBuf ; адрес вводимой строки

mov rdx, lenIn ; длина строки

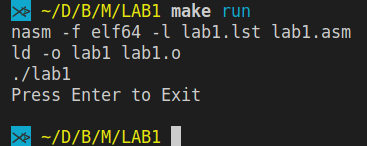
syscall ; вызов системной функции

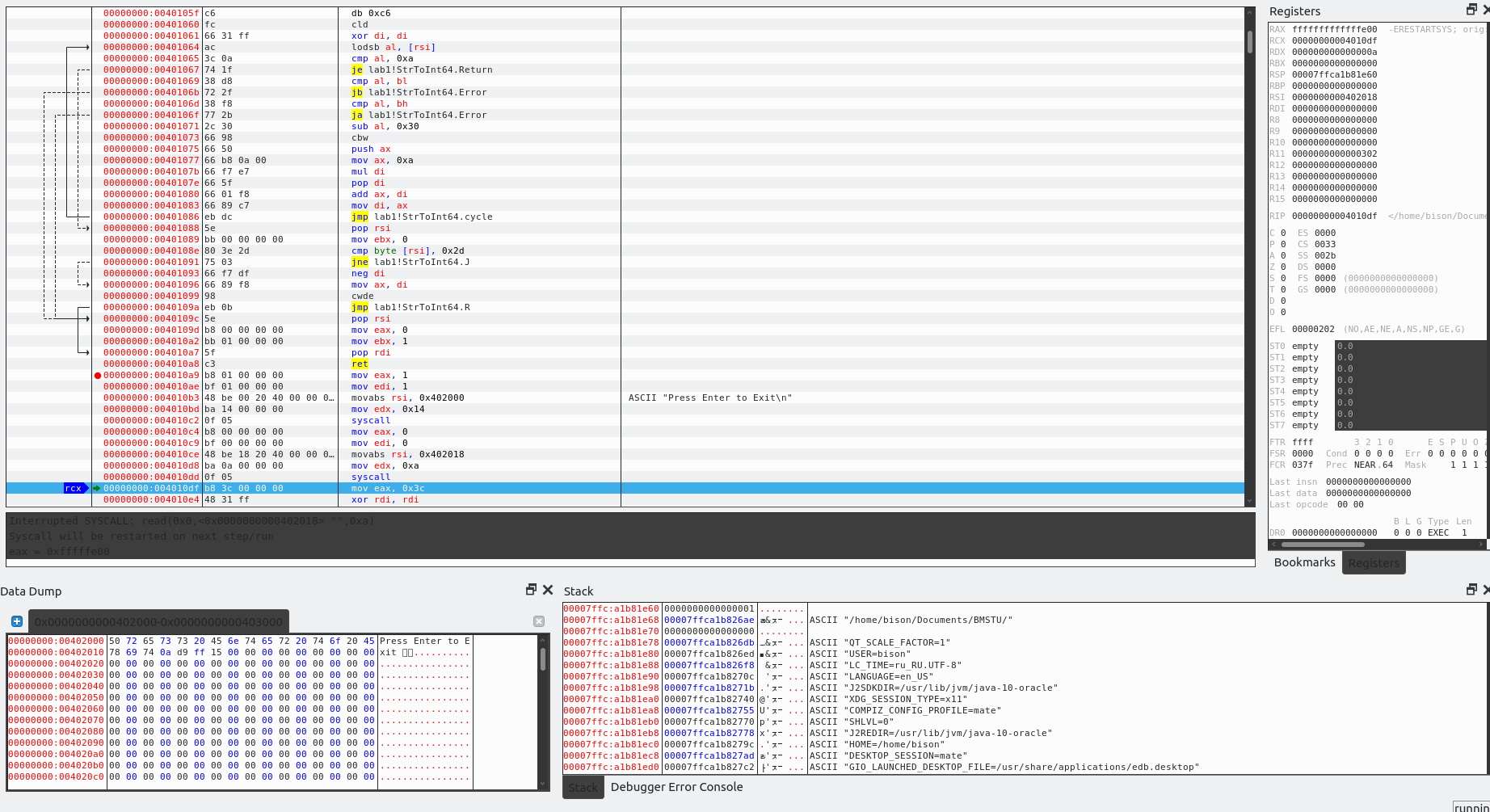
; exit

mov rax, 60 ; системная функция 60 (exit)

xor rdi, rdi ; return code 0

syscall ; вызов системной функции

Рисунок 1 - результат работы

Рисунок 2 - запуск отладчика edb

1.2.8. Для изучения возможностей отладчика добавьте в заготовку

несколько команд для вычисления результата следующего выражения:

X=A+5-B

1.2.9. Найдите в отладчике внутреннее представление исходных данных,

отразите его в отчете и поясните.

Проследите в отладчике выполнение программы и зафиксируйте в отчете

результаты выполнения каждой добавленной команды (изменение регистров,

флагов и полей данных).

Код программы для подсчета простого выражения, результат работы программы представлен на рисунке 1, работа с отладчиком на рисунках 4-6:

%include "io64.asm"

section .data

ExitMsg db "Press Enter to Exit",10

lenExit equ $-ExitMsg

A dw -30

B dw 21

section .bss

InBuf resb 10

lenIn equ $-InBuf

X resd 1

section .text ; сегмент кода

global \_start

\_start:

mov eax,[A] ; загрузить число A в регистр EAX

add eax,5 ; сложить EAX и 5, результат в EAX

sub eax,[B] ; вычесть число B, результат в EAX

mov [X],eax ; сохранить результат в памяти

mov esi,InBuf

mov ax, [X]

cwde

call IntToStr64

; write

mov rdx, rax

mov rax, 1 ; системная функция 1 (write)

mov rdi, 1 ; дескриптор файла stdout=1

syscall

mov rax, 1 ; системная функция 1 (write)

mov rdi, 1 ; дескриптор файла stdout=1

mov rsi, ExitMsg ; адрес выводимой строки

mov rdx, lenExit ; длина строки

syscall ; вызов системной функции

; read

mov rax, 0 ; системная функция 0 (read)

mov rdi, 0 ; дескриптор файла stdin=0

mov rsi, InBuf ; адрес вводимой строки

mov rdx, lenIn ; длина строки

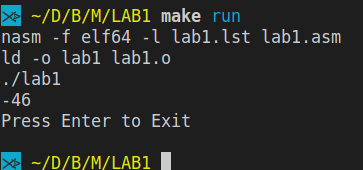
syscall ; вызов системной функции

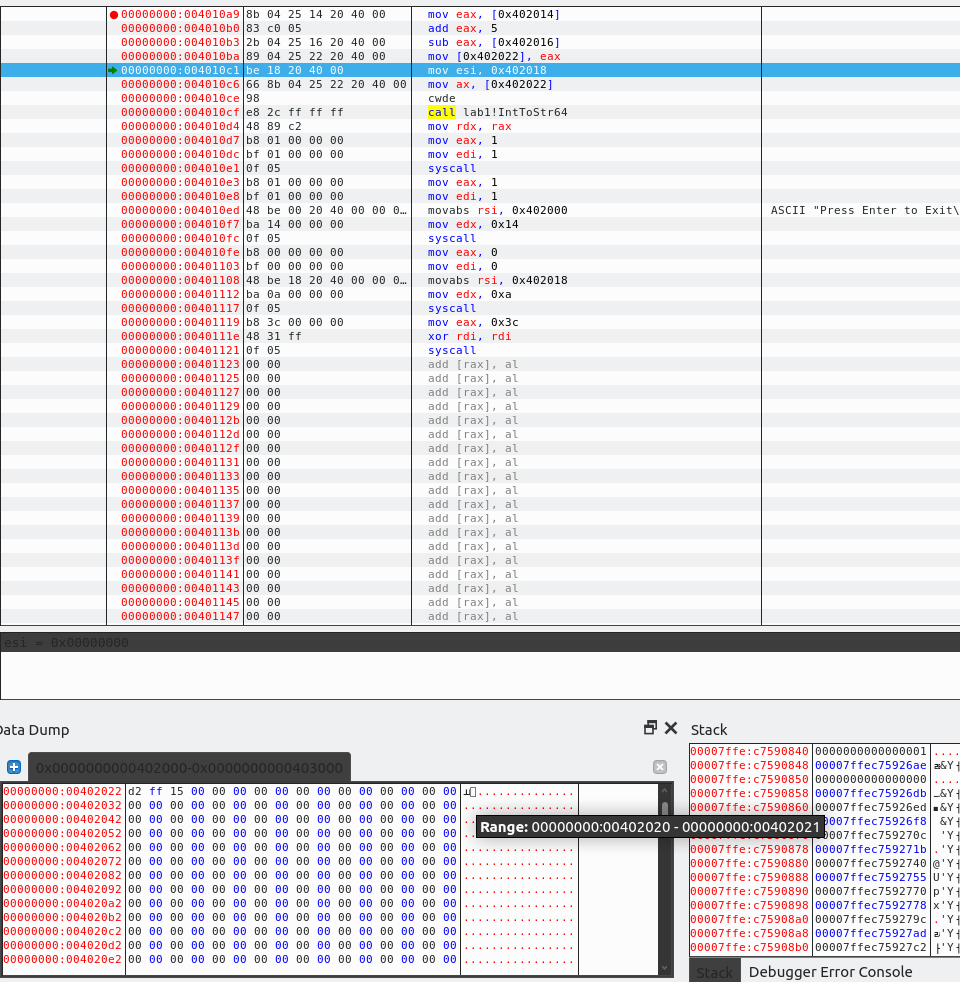
; exit

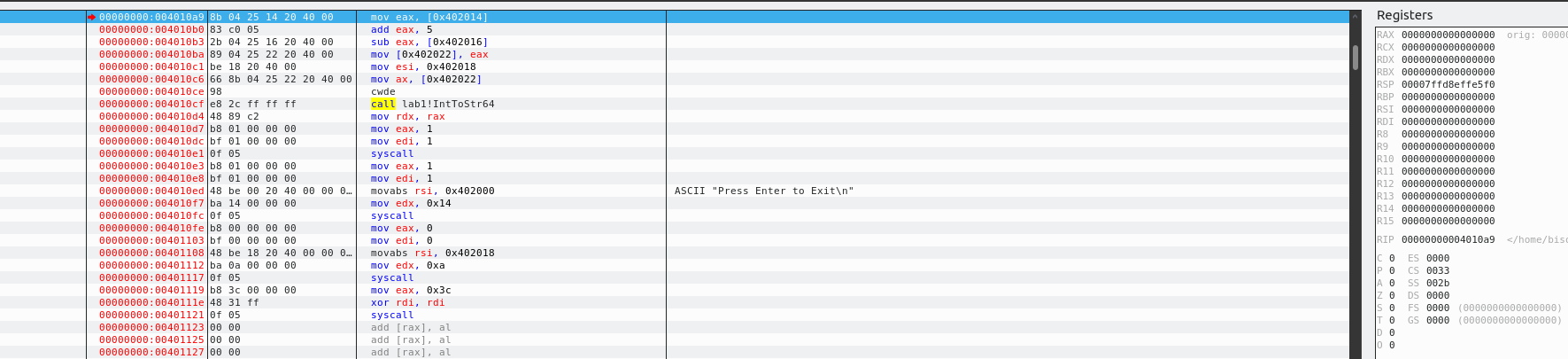
mov rax, 60 ; системная функция 60 (exit)

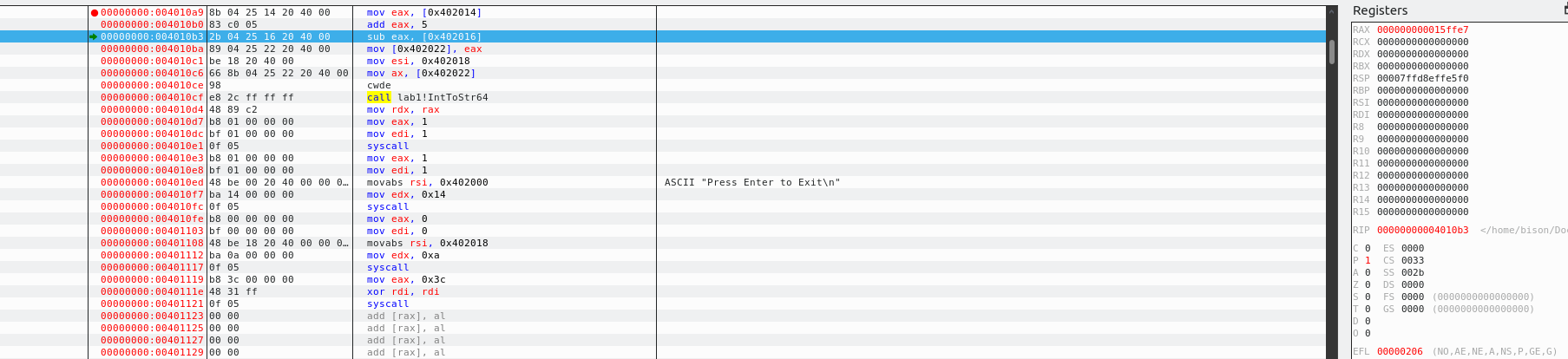
xor rdi, rdi ; return code 0

syscall ; вызов системной функции

Рисунок 3 — результат работы программы

Рисунок 4 — отображение в памяти данных программы

Рисунок 5 — изменение регистров

Рисунок 6 — изменение регистров

1.2.10. Введите следующие строки в разделы описания инициированных и

неинициализированных данных и определите внутренние представление этих данных.

Код программы с добавлением данных в секции инициализированных и неинициализированных данных, результат работы программы представлен на рисунке 7, а внутреннее представление данных в памяти на рисунке 8:

%include "io64.asm"

section .data

ExitMsg db "Press Enter to Exit",10

lenExit equ $-ExitMsg

A dw -30

B dw 21

val1 db 255

chart dw 256

lue3 dw -128

v5 db 10h

db 100101B

beta db 23,23h,0ch

sdk db "Hello",10

min dw -32767

ar dd 12345678h

valar times 5 db

section .bss

InBuf resb 10

lenIn equ $-InBuf

X resd 1

alu resw 10

f1 resb 5

section .text ; сегмент кода

global \_start

\_start:

mov eax,[A] ; загрузить число A в регистр EAX

add eax,5 ; сложить EAX и 5, результат в EAX

sub eax,[B] ; вычесть число B, результат в EAX

mov [X],eax ; сохранить результат в памяти

mov esi,InBuf

mov ax, [X]

cwde

call IntToStr64

; write

mov rdx, rax

mov rax, 1 ; системная функция 1 (write)

mov rdi, 1 ; дескриптор файла stdout=1

syscall

mov rax, 1 ; системная функция 1 (write)

mov rdi, 1 ; дескриптор файла stdout=1

mov rsi, ExitMsg ; адрес выводимой строки

mov rdx, lenExit ; длина строки

syscall ; вызов системной функции

; read

mov rax, 0 ; системная функция 0 (read)

mov rdi, 0 ; дескриптор файла stdin=0

mov rsi, InBuf ; адрес вводимой строки

mov rdx, lenIn ; длина строки

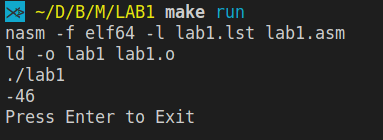
syscall ; вызов системной функции

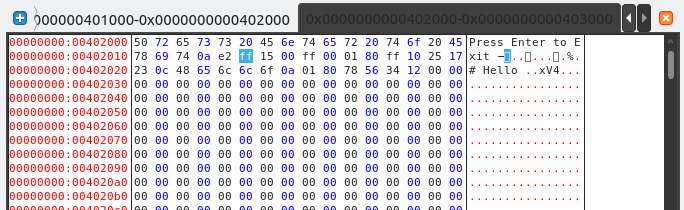
; exit

mov rax, 60 ; системная функция 60 (exit)

xor rdi, rdi ; return code 0

syscall ; вызов системной функции

Рисунок 7 — результат работы программы

Рисунок 8 — внутреннее представление данных в памяти

1.2.11. Определите в памяти следующие данные:

а) целое число 25 размером 2 байта со знаком; num1 dw 25

б) двойное слово, содержащее число -35; num2 dd -35

в) символьную строку, содержащую ваше имя (русскими буквами и

латинскими буквами).strok db "Sofia", 10 st db "Софья", 10

Зафиксируйте в отчете описание и внутреннее представление этих

данных и дайте пояснение, оно изображено на рисунке 9.

Рисунок 9 — внутреннее представление данных

1.2.12. Определите несколькими способами в программе числа, которые

во внутреннем представлении (в отладчике) будут выглядеть как 25 00 и 00 25. Проверьте правильность ваших предположений, введя соответствующие

строки в программу. Зафиксируйте результаты в отчете.

Внутреннее представление данных в памяти можно увидеть на рисунке 10.

Преобразуем внутреннее представление 25 00 в десятичное число:

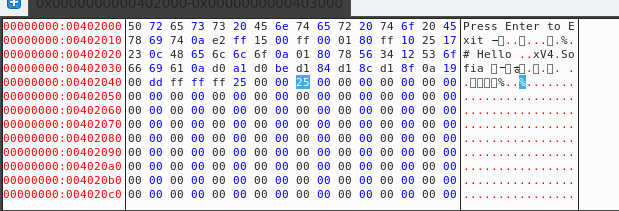
25 00 –> 002516 –> 3710

Аналогично преобразуем 00 25:

00 25 –> 250016 –> 947210

num3 dw 37

num4 dw 9472

Рисунок 10 — внутреннее представление данных

1.2.13. Добавьте в программу переменную F1=65535 размером слово и

переменную F2= 65535 размером двойное слово. Вставьте в программу

команды сложения этих чисел с 1:

add[F1],1

add[F2],1

Проанализируйте и прокомментируйте в отчете полученный результат

(обратите внимание на флаги).

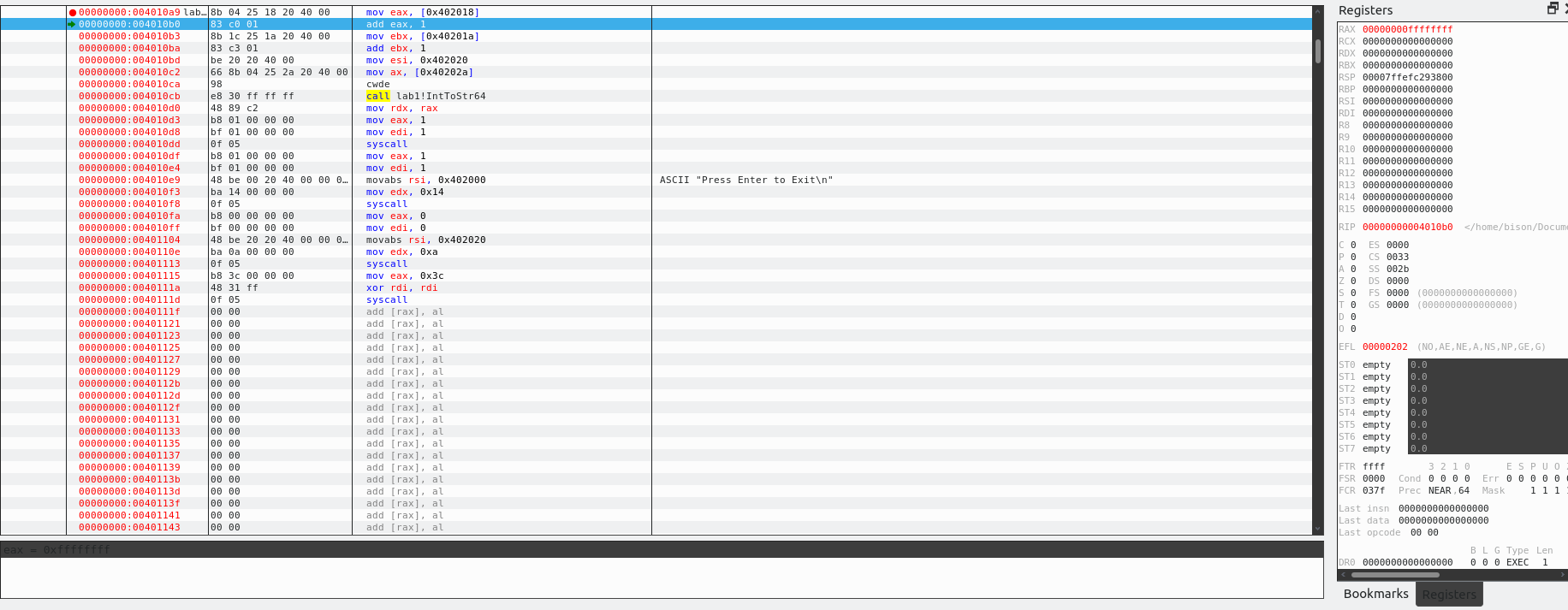
Добавим в секцию .data следующие команды:

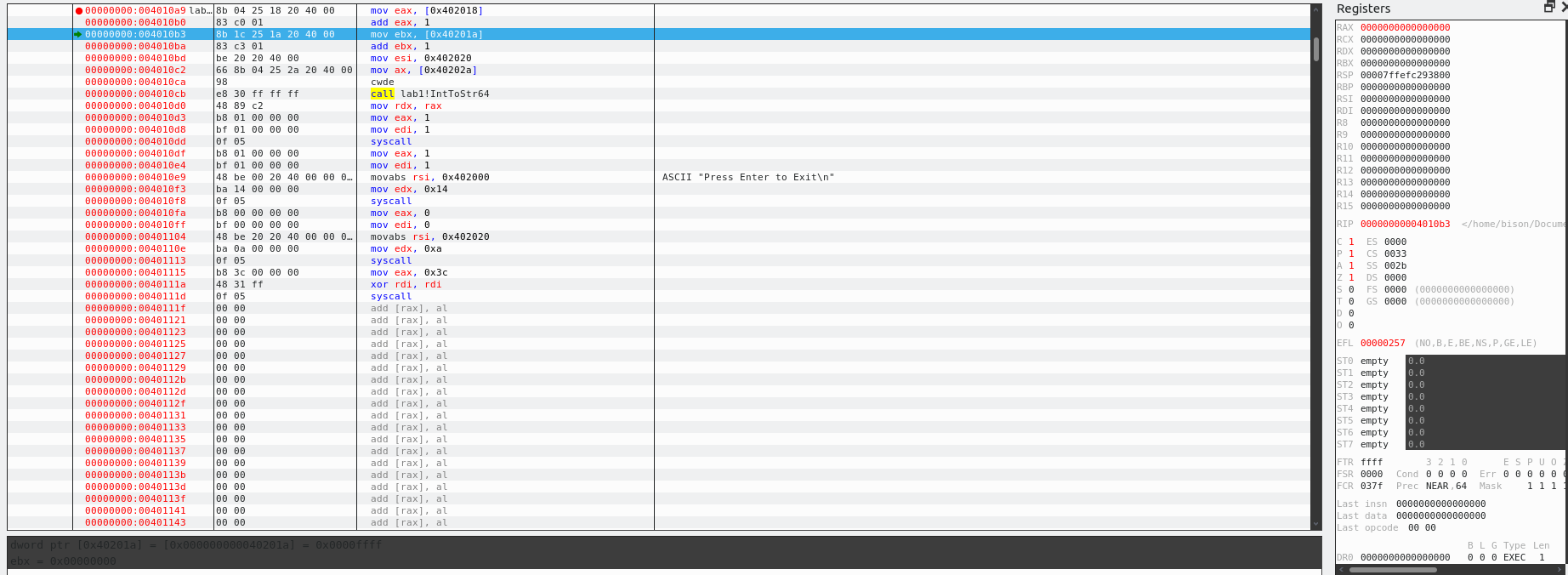
F1 dw 65535

F2 dd 65535

Также добавим в текст программы данные выражения

Проверим команды в отладчике, отладка изображена на рисунках 11 - 13

Рисунок 11 — флаги не установлены

Рисунок 12 — флаги установлены

Мы можем видеть, что после первой команды флаги CF и ZF были установлены в ненулевое значение, так как случилось переполнение.

**Контрольные вопросы**

1. Дайте определение ассемблеру. К какой группе языков он относится?

Язык ассемблера или ассемблер (англ. assembly language) — язык низкого уровня с командами, обычно соответствующими командам процессора (так называемым машинным командам). Это соответствие позволяет отнести язык к группе машинно-зависимых, к которой относятся также машинные языки.

2. Из каких частей состоит заготовка программы на ассемблере?

.data – секция инициализированных данных

.bss – секция неинициализированных данных

.text – секция теста программы

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

Для запуска программы необходимо ввести заготовки, преобразовать в машинный код и скомпоновать со всеми требуемыми подпрограммами. Первую операцию выполняет текстовый редактор, вторую — транслятор, третью — компоновщик. И только после этого полученную выполняемую программу можно запустить на выполнение.

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

Отладчик позволяет выполнять программу по шагам, заходя или не заходя в подпрограммы, и при этом анализировать изменения содержимого регистров, памяти и стека. Для этого используют соответствующие пункты меню или следующие клавиши:

- F7 – выполнить шаг с заходом в подпрограмму;

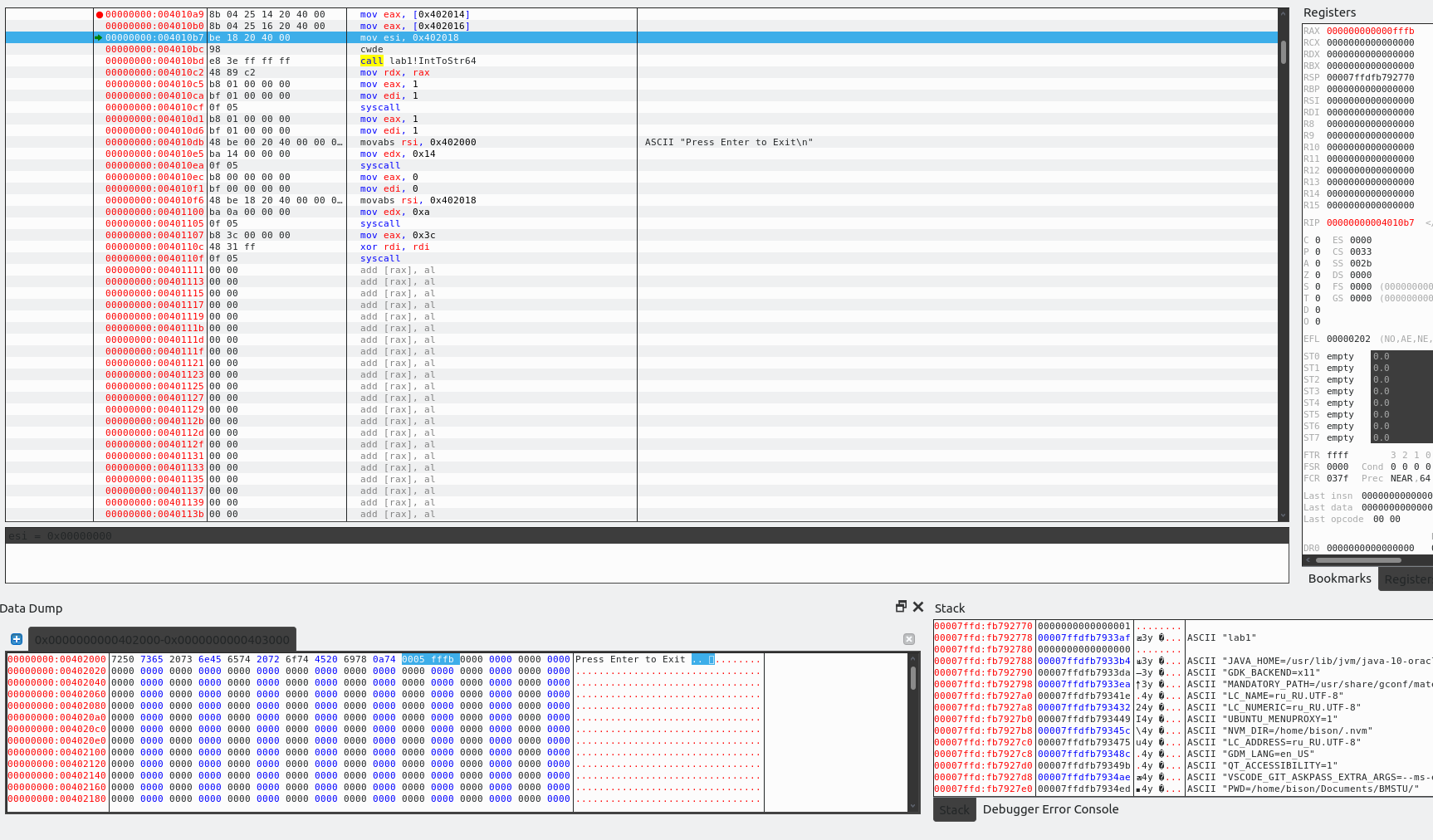
- F8 – выполнить шаг без захода в подпрограмму;

- Ctrl + F2 – начать отладку сначала;

- Ctrl + F9 – выполнить подпрограмму до возврата из нее.

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?

Числа в памяти:



В регистре AX байты чисел будут идти в прямом порядке.

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

.data

A db 3

B db 2

.bss

C resb 1

.text

global \_start

\_start:

mov AX,[A]

add AX,[B]

mov [C],AX

**Вывод:** в ходе выполнения данной лабораторной работы я изучила процесс создания, запуска и отладки программ на ассемблере Nasm, а также особенностей описания и внутреннего представления данных.