|  |  |
| --- | --- |
| 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**

Вариант 2.21

Студент гр. ИУ6-42Б  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.С. Твердюк**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ C.C. Данилюк**

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

***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.

***Ход работы:***

***Задание 1-6.***

1. Для хранения всех программ лабораторных работ на компьютере в операционной системе создайте специальные каталог и подкаталог.
2. Объявите подкаталог lab1 текущим.
3. Сохраните программу с именем LR1.asm в подкаталоге LR1.
4. Выполните трансляцию программы с листингом.
5. Совершите компоновку 64-х разрядной программы.
6. Запустите программу на выполнение.

**Программа LR1.asm**

section .data ; сегмент инициализированных переменных

ExitMsg db "Press Enter to Exit", 10 ; выводимое сообщение

lenExit equ $-ExitMsg ; сегмент неинициализированных переменных

section .bss

InBuf resb 10 ; буфер для вводимой строки

lenIn equ $-InBuf

X resb 1

section .text ; сегмент кода

global \_start

\_start:

; write

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

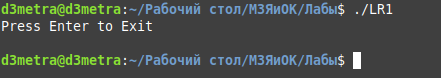
syscall ; вызов системной функции

На рисунке 1 представлены команды трансляции и компоновки программы.

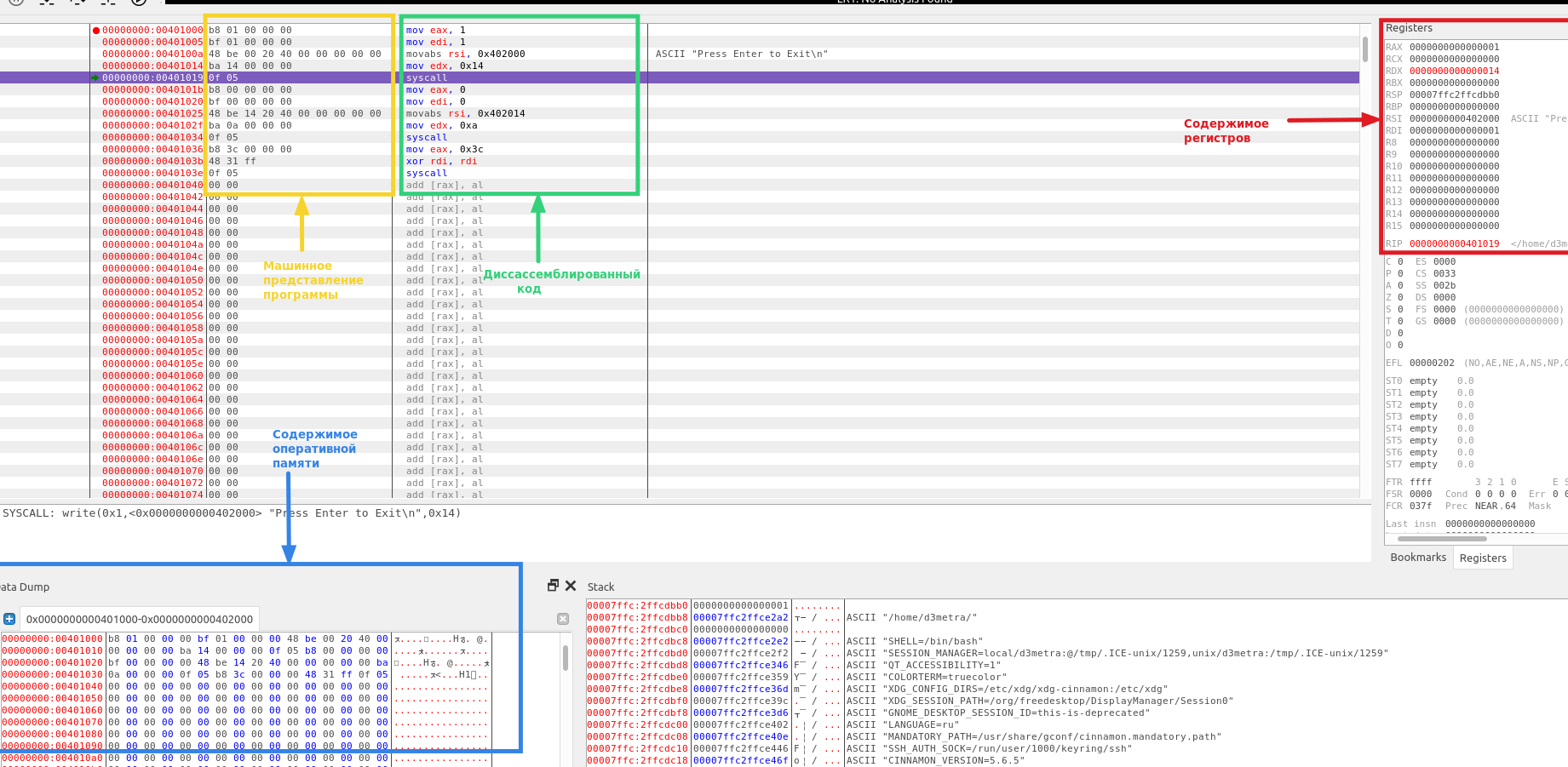
На рисунке 2 показан ход выполнения написанной программы.

Рисунок 1 - Трансляция и компоновка программы

***Задание 7*.** Запустите отладчик edb. Найдите машинное представление программы, ее дисассемблированный код, содержимое регистров и т.д. Выполните программу по шагам, контролируя содержимое регистров и оперативной памяти.

Рисунок 2 - Выполнение программы

На рисунке 3 представлены возможности отладочной среды EDB.

Рисунок 3 - Исполняемая программа LR1 в отладчике EDB

***Задание 8.*** Для изучения возможностей отладчика добавьте в заготовку несколько команд для вычисления результата следующего выражения: X=A+5-B. Сохраните программу с тем же именем, затем выполните ее трансляцию, компоновку и загрузку в отладчик.

**Программа LR1.asm**

section .data ; сегмент инициализированных переменных

ExitMsg db "Press Enter to Exit", 10 ; выводимое сообщение

lenExit equ $-ExitMsg ; сегмент неинициализированных переменных

A dw -30

B dw 21

section .bss

InBuf resb 10 ; буфер для вводимой строки

lenIn equ $-InBuf

X resb 1

section .text ; сегмент кода

global \_start

\_start:

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

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

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

mov [X], rax ; сохранить результат в памяти

; write

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)

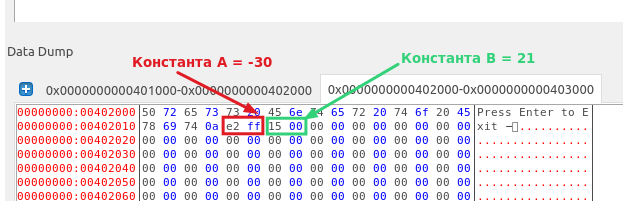
xor rdi, rdi ; return code 0

syscall ; вызов системной функции

***Задание 9.*** Найдите в отладчике внутреннее представление исходных данных,

отразите его в отчете и поясните.

На рисунке 4 наглядно продемонстрировано внутреннее представление объявленных в программе констант.

Рисунок 4 - Внутреннее представление исходных данных

Во внутренней памяти байты располагаются в обратном порядке.

FFE2*h =* [дополнительный код]1.111 1111 1110 0010 *b* = [обратный код] 1.111 1111 1110 0001 *b =* [прямой код] 1.000 0000 0001 1110 *b* = -30

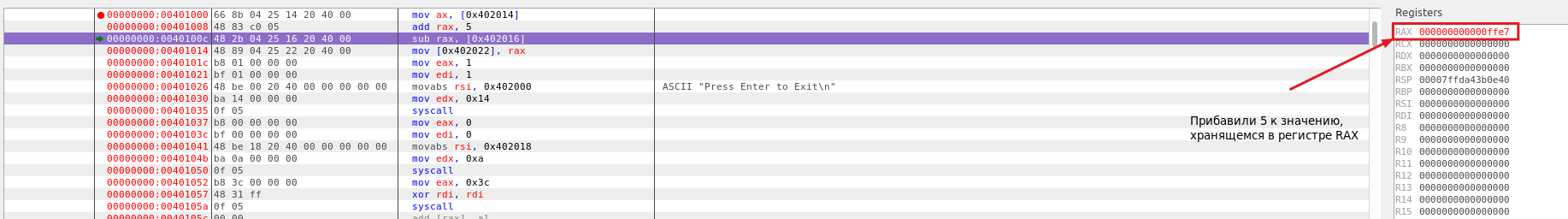
0015*h* = 15*h* = [прямой код] 0001 0101 *b* = 21

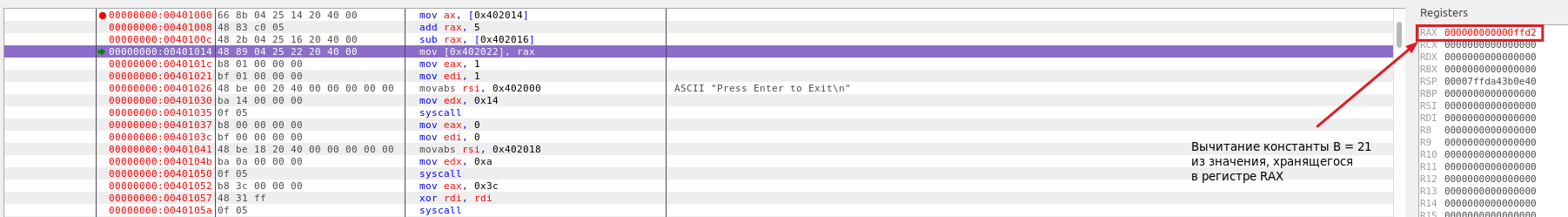
Проследите в отладчике выполнение программы и зафиксируйте в отчете

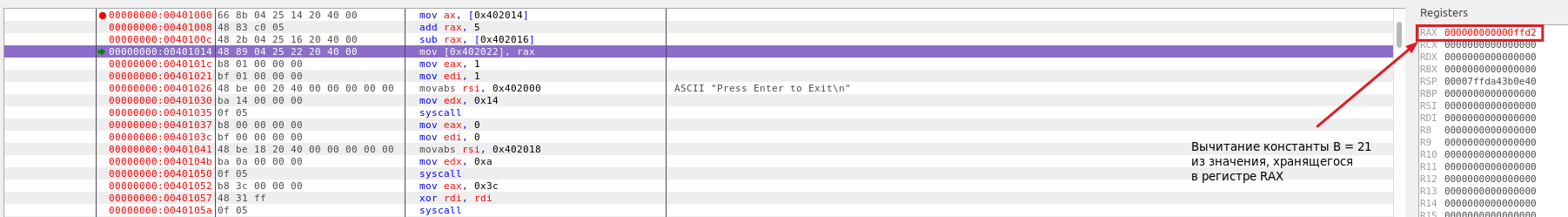
результаты выполнения каждой добавленной команды (изменение регистров,

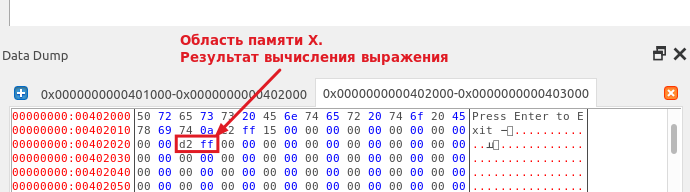
флагов и полей данных).

На рисунке 5 показан ход вычисления выражения X = A+5-B.



Рисунок 5 - Выполнение программы с константами

Рисунок 5 - Выполнение программы с константами

Рисунок 6 - Результат вычисления выражения, представленный в памяти

FFD2*h* = [дополнительный код] 1.111 1111 1101 0010*b* = = [обратный код] 1.111 1111 1101 0001*b* = [прямой код] 1.000 0000 0010 1110*b* = -46

***Задание 10.*** Введите данные в разделы описания инициированных и неинициализированных данных и определите с помощью отладчика внутренние данных представление и определите этих данных в памяти.

**Программа LR1.asm**

section .data ; сегмент инициализированных переменных

ExitMsg db "Press Enter to Exit", 10 ; выводимое сообщение

lenExit equ $-ExitMsg ; сегмент неинициализированных переменных

val1 db 255

chart dw 256

lue3 dw -128

v5 db 10h

bin db 100101b

beta db 23, 23h, 0ch

sdk db "Hello", 10

min dw -32767

ar dd 12345678h

valar times 5 db 8

section .bss

InBuf resb 10 ; буфер для вводимой строки

lenIn equ $-InBuf

alu resw 10

f1 resb 5

section .text ; сегмент кода

global \_start

\_start:

; write

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

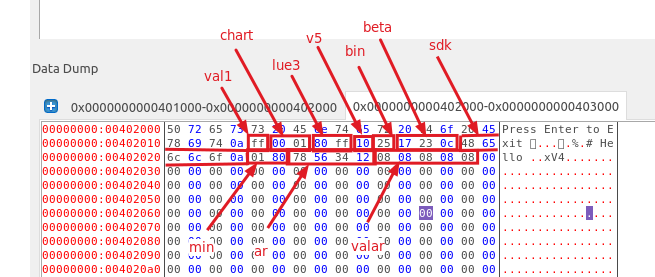
mov rax, 60 ; системная функция 60 (exit)

xor rdi, rdi ; return code 0

syscall ; вызов системной функции

На рисунке 7 продемонстрировано внутреннее представление объявленных констант.

Неинициализированные данные расположены после инициализированных,  
но не сохранены в памяти.

Рисунок 7 - Внутреннее представление данных в памяти

***Задание 11.*** Определите в памяти следующие данные:

а) целое число 25 размером 2 байта со знаком;

б) двойное слово, содержащее число -35;

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

Зафиксируйте описание и внутреннее представление этих данных и дайте пояснение.

**Программа LR1.asm**

section .data ; сегмент инициализированных переменных

ExitMsg db "Press Enter to Exit", 10 ; выводимое сообщение

lenExit equ $-ExitMsg ; сегмент неинициализированных переменных

A dw -25

B dd -30

C db "Дмитрий Dmitriy", 10

section .bss

InBuf resb 10 ; буфер для вводимой строки

lenIn equ $-InBuf

section .text ; сегмент кода

global \_start

\_start:

; write

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

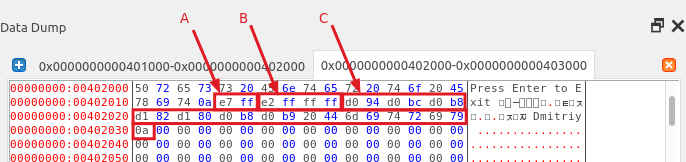
; exit

mov rax, 60 ; системная функция 60 (exit)

xor rdi, rdi ; return code 0

syscall ; вызов системной функции

На рисунке 8 продемонстрировано внутреннее представление объявленных в программе констант.

Рисунок 8 - Внутреннее представление данных

***Задание 12.*** Определите несколькими способами в программе числа, которые

во внутреннем представлении будут выглядеть как 25 00 и 00 25. Проверьте правильность ваших предположений, введя соответствующие строки в программу.

**Программа LR1.asm**

section .data ; сегмент инициализированных переменных

ExitMsg db "Press Enter to Exit", 10 ; выводимое сообщение

lenExit equ $-ExitMsg ; сегмент неинициализированных переменных

A dw 37

B dw 9472

section .bss

InBuf resb 10 ; буфер для вводимой строки

lenIn equ $-InBuf

section .text ; сегмент кода

global \_start

\_start:

; write

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

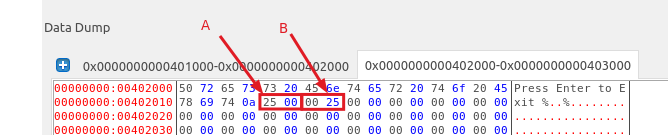
syscall ; вызов системной функции

; exit

mov rax, 60 ; системная функция 60 (exit)

xor rdi, rdi ; return code 0

syscall ; вызов системной функции

Рисунок 9 - Внутреннее представление данных в памяти

Представление значений А и B в шестнадцатеричной системе счисления представлены на рисунке 9:

А = 37 = 0025*h*

B = 9472 = 2500*h*

***Задание 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.

**Программа LR1.asm**

section .data ; сегмент инициализированных переменных

ExitMsg db "Press Enter to Exit", 10 ; выводимое сообщение

lenExit equ $-ExitMsg ; сегмент неинициализированных переменных

F1 dw 65535

F2 dd 65535

section .bss

InBuf resb 10 ; буфер для вводимой строки

lenIn equ $-InBuf

section .text ; сегмент кода

global \_start

\_start:

add word[F1], 1

add dword[F2], 1

; write

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

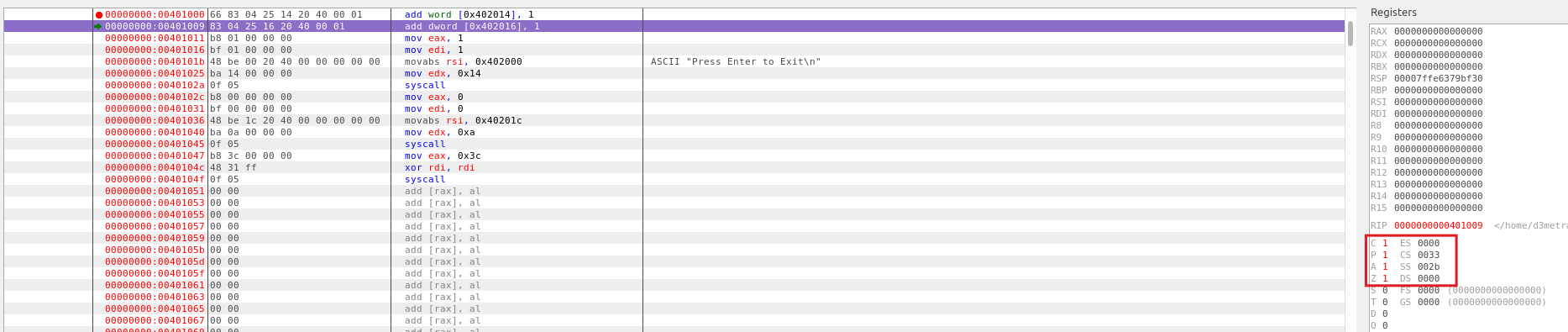
mov rax, 60 ; системная функция 60 (exit)

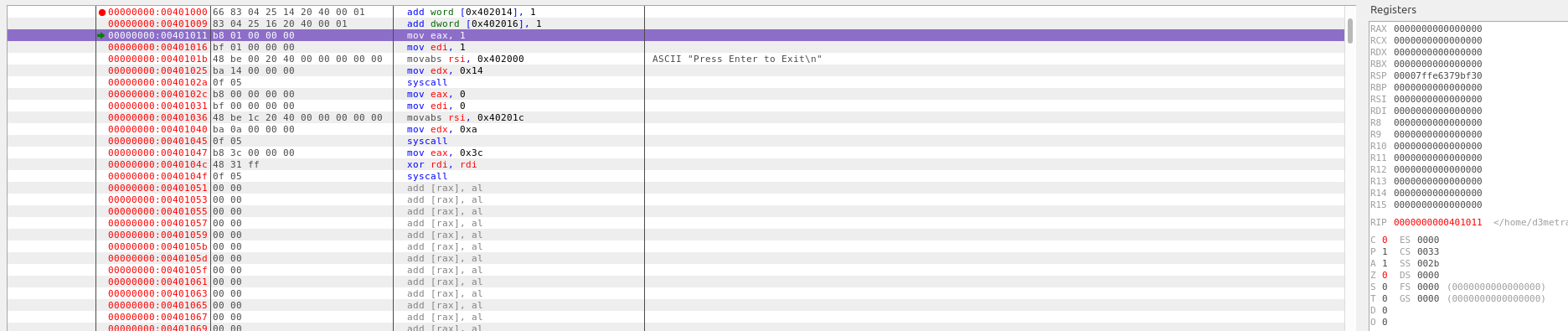
xor rdi, rdi ; return code 0

syscall ; вызов системной функции

При сложении 1 и F1 изменяются флаги: CF (Флаг переноса. Устанавливается в 1, если результат предыдущей операции не уместился в приёмнике и произошёл перенос из старшего бита), PF (Флаг чётности. Устанавливается в 1, если младший байт результата предыдущей команды содержит чётное количество битов, равных 1), AF (Вспомогательный флаг переноса (или флаг полупереноса). Устанавливается в 1, если в результате предыдущей операции произошёл перенос (или заём) из третьего бита в четвёртый. Этот флаг используется автоматически командами двоично-десятичной коррекции.), ZF (Флаг нуля. Устанавливается 1, если результат предыдущей команды равен 0.)

При сложении 1 и F2 изменяются флаги: CF, ZF

Рисунок 10 - Значения флагов при сложении F1 и 1

Рисунок 11 - Значения флагов при сложении F2 и 1

***Контрольные вопросы***

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

Язык ассемблера — низкоуровневый язык программирования, состоящий из операций, которые представляют собой команды процессора. Язык ассемблера относится к группе машинно-зависимых языков.

1. Из каких частей состоит заготовка программы на ассемблере?

- .text (сегмент кода);

- .data (сегмент инициализированных данных);

- .bss (сегмент неинициализированных данных).

1. 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 следующей командой:

nasm -f elf64 lab1.asm -l lab1.lst

В результате работы транслятор создает объектный файл, которые затем подается на вход компоновщика:

ld -o lab1 lab1.o

Компоновщик формирует исполняемую программу.

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

– F7 – выполнить шаг с заходом в тело процедуры;

– F8 – выполнить шаг, не заходя в тело процедуры;

– F9 - выполнить выход из тела процедуры.

Пошаговое выполнение программы можно осуществить с помощью F7.

1. 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?

Отладчик представляет числа в прямом порядке записи байт: 5 в шестнадцатеричной системе счисления будет равно 00|05, а -5 равно FF|FB. В памяти байты имеют обратный порядок, т.е. для 5: 05|00, для -5: FB|FF. Если эти числа загрузить в регистр AX, то его содержимое будет равняться: для 5 - 00|05, для -5 - FF|FB.

1. 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 ; сегмент инициализированных переменных

ExitMsg db "Press Enter to Exit",10 ; выводимое сообщение

lenExit equ $-ExitMsg

A db 15

B db 23

section .bss ; сегмент неинициализированных переменных

InBuf resb 10 ; буфер для вводимой строки

lenIn equ $-InBuf

C resb 1

section .text ; сегмент кода

global \_start

\_start:

mov eax, [A]

add eax. [B]

mov [C], eax

; write

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

***Вывод:*** в процессе выполнения лабораторной работы я изучил процессы создания, запуска и отладки программ на ассемблере Nasm под управлением операционной системы Linux, а также особенностей описания и внутреннего представления данных.