|  |  |
| --- | --- |
| 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 **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Д. Шатский**

(Подпись, дата) (И.О. Фамилия)

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

(Подпись, дата) (И.О. Фамилия)

Москва, 2023

**Вариант 2.23**

**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.

**Задание.** Для хранения всех программ лабораторных работ на компьютере создайте специальные каталог и подкаталог. Объявите подкаталог labs/lab1 текущим. Введите заготовку 64-х разрядной программы на ассемблере. Сохраните программу с именем lab1.asm в подкаталоге labs/lab1. Выполните трансляцию программы с листингом. Выполнить компоновку 64-х разрядной программы. Запустить программу на выполнение.

**Программа 64 разрядного процессора**

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

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

; вызов системной функции

syscall

; 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

; вызов системной функции

syscall

; exit

mov rax, 60; системная функция 60 (exit)

xor rdi, rdi; return code 0

syscall

; вызов системной функции

На рисунке 1 показан запуск программы в консоли.

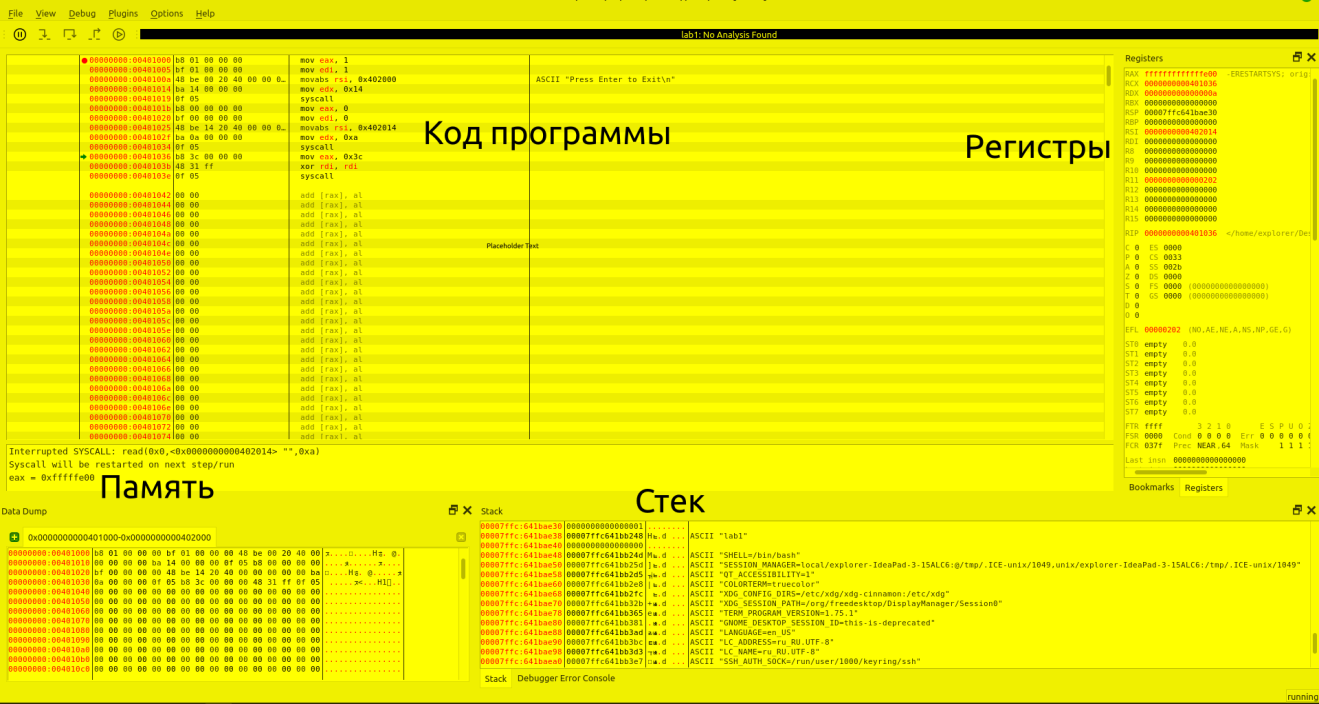


*Рисунок 1 – Запуск программы на выполнение*

**EDB debugger**

**Задание.** Запустите отладчик 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.

Отладчик edb debugger и подписи окон отладчика представлены на рисунке 2



*Рисунок 2 – Просмотр программы в edb debugger*

**Задание.** Добавьте в заготовку несколько команд для вычисления результата следующего выражения:   
X=A+5-B

Найдите в отладчике внутреннее представление исходных данных,   
отразите его в отчете и поясните.

**Добавим решение выражения X=A+5-B**

section .data

; сегмент инициализированных переменных

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

lenExit equ $-ExitMsg

A dd -30

B dd 21

; сегмент неинициализированных переменных

section .bss

InBuf resb 10 ; буфер для вводимой строки

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

; 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

; вызов системной функции

syscall

; 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

; вызов системной функции

syscall

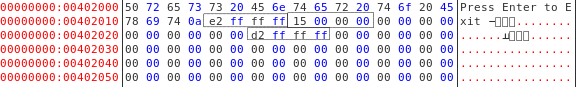
; exit

mov rax, 60; системная функция 60 (exit)

xor rdi, rdi; return code 0

syscall

; вызов системной функции



*Рисунок 3 – Переменные записанные в памяти*

Расшифровка переменных из отладчика по рисунку 3:

E2 FF FF FF – число -30

15 00 00 00 – число 21

D2 FF FF FF – результат сложения(число-9)

Пошаговая проверка регистров изображена на таблице 1

*Таблица 1 – Переменные записанные в памяти*

|  |  |  |  |
| --- | --- | --- | --- |
| mov ax,[A] | add ax,5 | sub ax,[B] | mov [X],ax |
|  |  |  |  |

**Задание.** Введите следующие строки в разделы описания инициированных и неинициализированных данных и определите с помощью отладчика внутренние представление этих данных в памяти. Результаты проанализируйте и занесите в отчет.

val1 db 255   
chart dw 256   
lue3 dw -128   
v5 db 10h   
alpha db 100101B   
beta db 23,23h,0ch   
sdk db "Hello",10   
min dw -32767   
ar dd 12345678h   
valar times 5 db 8   
alu resw 10   
f1 resb 5

**Введём новые переменные согласно заданию**

section .data

; сегмент инициализированных переменных

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

lenExit equ $-ExitMsg

val1 db 255

chart dw 256

lue3 dw -128

v5 db 10h

alpha db 100101B

beta db 23,23h,0ch

sdk db "Hello",10

min dw -32767

ar dd 12345678h

valar times 5 db 8

; сегмент неинициализированных переменных

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

; вызов системной функции

syscall

; 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

; вызов системной функции

syscall

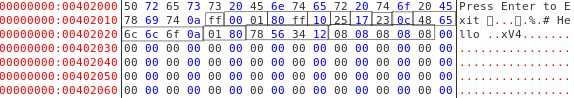
; exit

mov rax, 60; системная функция 60 (exit)

xor rdi, rdi; return code 0

syscall

; вызов системной функции



*Рисунок 4 – Переменные записанные в памяти*

Расшифровки переменных из отладчика по рисунку 4

FF – val1(число 255)

00 01 – chart (число 256)

80 FF – lue3 (число -128)

10 – v5 (число 10h)

25 – alpha (число 100101b)

17 23 0c – beta (числа 23 23h 0ch)

48 65 6c 6c 6f 0a – sdk (Строка “Hello” и перевод на новую строку)

01 80 – min (число -32767)

78 56 34 12 – ar (число 12345678h)

08 08 08 08 08 – valar (число 8, записанное 5 раз)

Переменные alu и f1 не инициализированы поэтому их не видно в памяти.

**Задание.** 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).

**Добавим значения, согласно заданию**

section .data

; сегмент инициализированных переменных

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

lenExit equ $-ExitMsg

twfive dw 25

minthfive dw -35

name db "АртёмArtyom"

; сегмент неинициализированных переменных

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

; вызов системной функции

syscall

; 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

; вызов системной функции

syscall

; exit

mov rax, 60; системная функция 60 (exit)

xor rdi, rdi; return code 0

syscall

; вызов системной функции

**Z:\home\explorer\Pictures\Screenshot from 2023-02-18 11-18-05 edit.png**

*Рисунок 5 – Переменные записанные в памяти*

Расшифровки переменных из отладчика по рисунку 5

19 00 – twfive (число 25)

DD FF – minthfive (число -35)

D0 90 D1 80 D1 82 D1 91 D0 BC – строка “Артём”

41 72 74 79 6F – строка “Artyom”

**Задание.** 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.

Так как первый и последний байт в памяти меняются местами, значит для записи числа 2500 в память нужно использовать число 25h, а для 25 нужно использовать 2500h. Для записи будем использовать слова:

section .data

; сегмент инициализированных переменных

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

lenExit equ $-ExitMsg

num2500h dw 25h

num2500 dw 37

num2500b dw 100101B

num0025h dw 2500h

num0025 dw 9472

num0025b dw 10010100000000B

; сегмент неинициализированных переменных

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

; вызов системной функции

syscall

; 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

; вызов системной функции

syscall

; exit

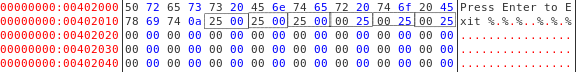
mov rax, 60; системная функция 60 (exit)

xor rdi, rdi; return code 0

syscall

; вызов системной функции

Внутреннее предствление инициализированных переменных можно посмотреть на рисунке 6



*Рисунок 6 – Переменные записанные в памяти*

**Задание.** 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.

**Добавим переменные F1 и F2**

section .data

; сегмент инициализированных переменных

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

lenExit equ $-ExitMsg

F1 dw 65535

F2 dd 65535

; сегмент неинициализированных переменных

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

; вызов системной функции

syscall

; 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

; вызов системной функции

syscall

; exit

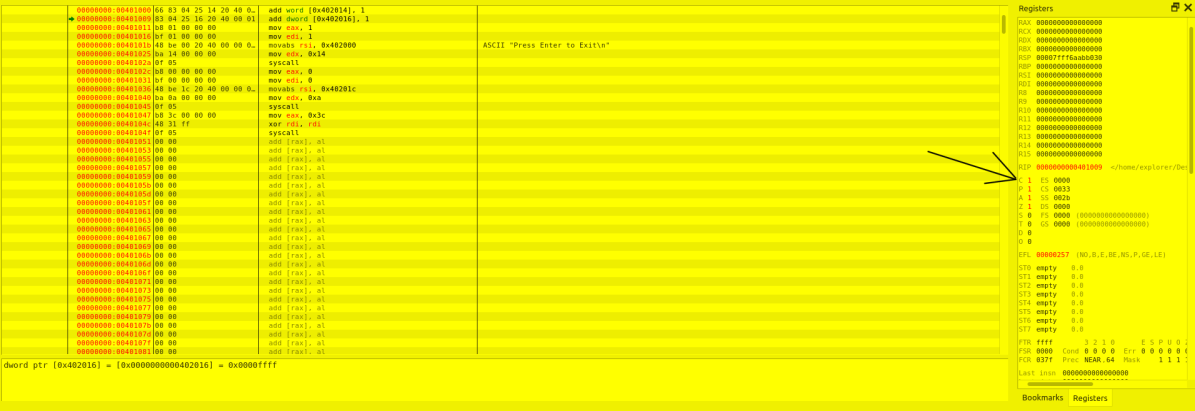
mov rax, 60; системная функция 60 (exit)

xor rdi, rdi; return code 0

syscall

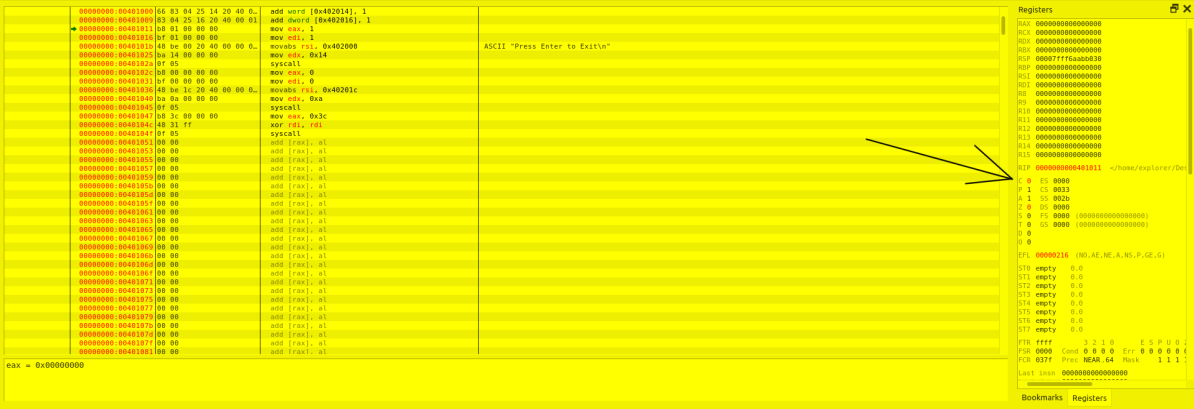
; вызов системной функции

Так как переменная F1 не может содержать числа большие 65535 происходит переполнение и carry flag равен 1. Поэтому на рисунке 7 carry flag после выполнения команды сложения равен 1.



*Рисунок 7 – Переполнение переменной F1 при сложении*

Так как переменная F2 может содержать числа большие 65535 переполнения не происходит и carry flag равен 0. Поэтому на рисунке 7 carry flag после выполнения команды сложения равен 0.



*Рисунок 8 – Сложение и запись в перемнную F2*

**Контрольные вопросы**

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

Ассемблер – это язык низкого уровня, предназначенный для создания программ с помощью непосредственного ввода команд процессора.

**2. Из каких частей состоит заготовка программы на ассемблере?**

Она состоит из сегмента инициализированных данных в которой находятся все данные, которые необходимо инициализировать до выполнения программы, сегмента неинициализированных данных, в котором выделяется место в памяти для дальнейшего использования и сегмента кода, в котором написан весь код, необходимый для выполнения программы.

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

Для запуска программы её необходимо ассемблировать и скомпоновать. Затем её можно запустить на выполнение.

На этапе ассемблирования программа преобразуется в двоичный код. Затем, на этапе компоновки, к двоичному коду добавляются объектные коды используемых подпрограмм.

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

Есть два режима работы отладчика: шаг с заходом в тело процедуры и шаг без захода в тело процедуры.

Для осуществления пошагового выполнения программы необоходимо выполнить программу до необходимого шага, и затем посмотреть результаты выполнения на “Содержимое регистров” и “Содержимое памяти”.

**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?**

Отладчик показывает положительные и отрицательные числа в шестнадцатеричной системе, как они записаны в памяти.

Число 5 будет представлено в памяти в виде: 05 00 , в регистре в

виде: 0005

Число -5 будет представлено в памяти в виде: FB FF , в регистре в

виде: 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 10

B db 35

section .bss

C resb 1

section .text

global \_start

\_start:

mov AL,[A]

add AL,[B]

mov [C],AL

mov rax, 60

xor rdi, rdi

syscall

**Вывод.** Былиизучены процессы создания, запуска и отладки программ   
на ассемблере, а также особенности описания и внутреннего представления данных.