



«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

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

## О Т Ч Е Т

### по лабораторной работе № 1

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

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

Студент гр. ИУ6-43Б

01.03.2023  
(Подпись, дата)

Таринду Херат  
(И.О. Фамилия)

Преподаватель

01.03.2023  
(Подпись, дата)

Данилюк.С.С  
(И.О. Фамилия)

## Изучение среды и отладчика ассемблера

**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 1. Выполнение шаблона и работа с отладчиком.

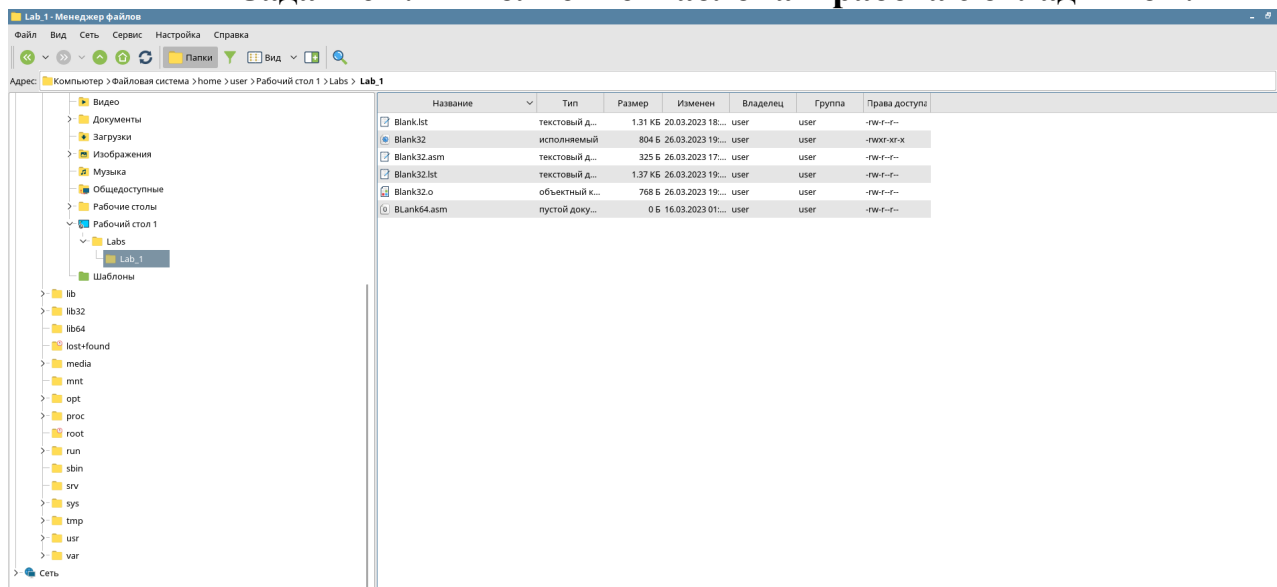


Рисунок 1 - Шаблоны для 32-разрядной и 64-разрядной программ

Запустим на 64-разрядном компьютере 32-разрядную программу в режиме эмуляции (смотри рисунок 2).

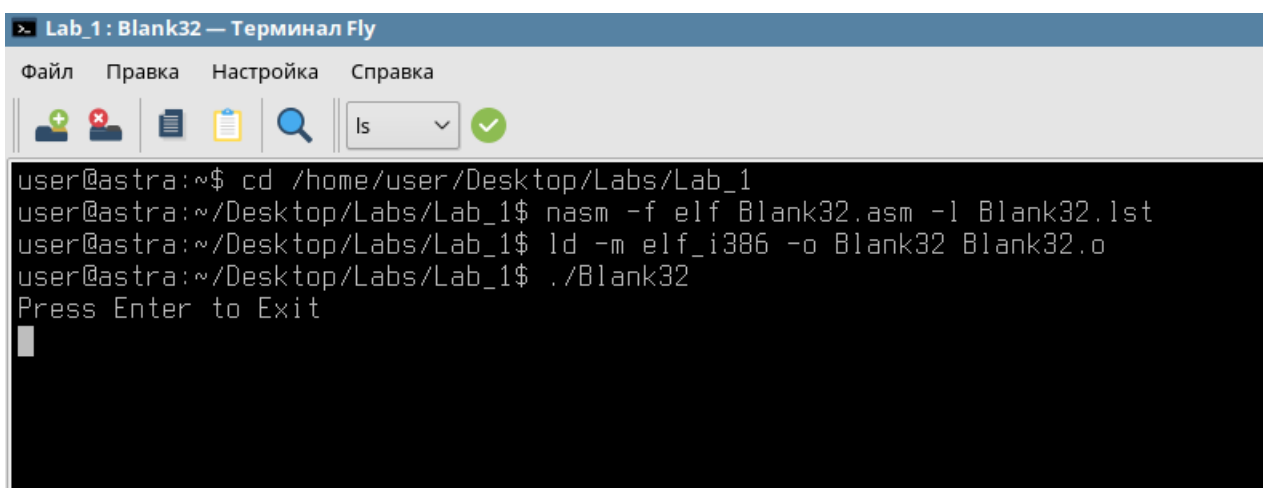
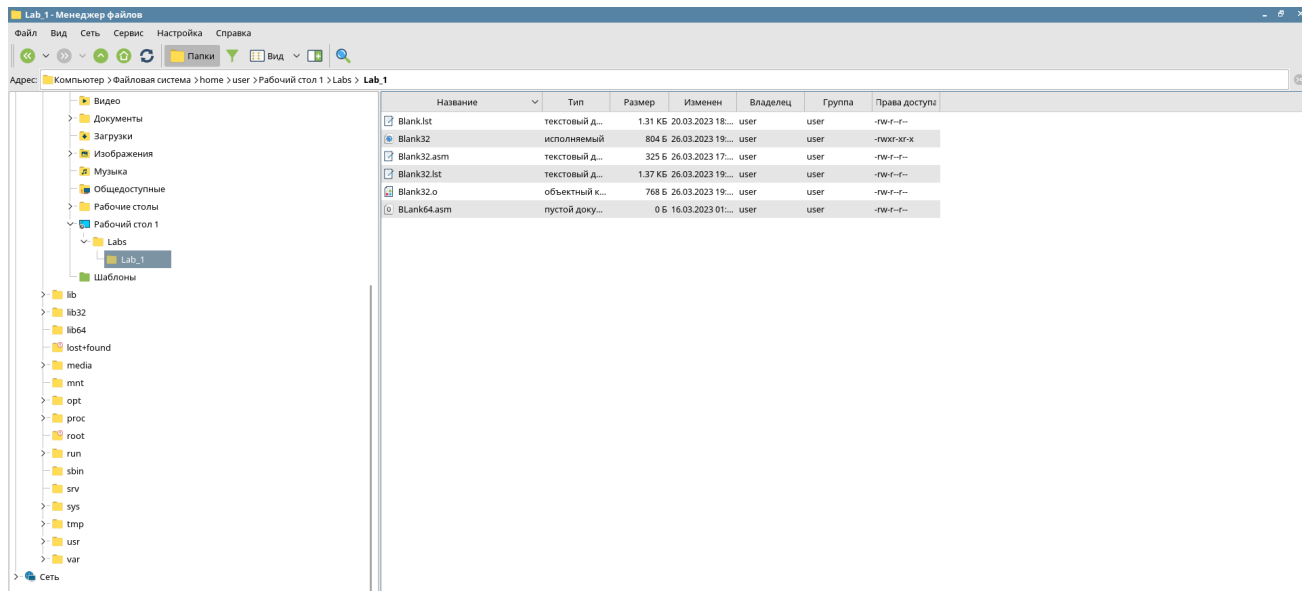


Рисунок 2 - Трансляция, компоновка и выполнение программы Blank32



*Рисунок 3 - Созданные файлы*

```
nasm -f elf Blank32.asm -l Blank32.lst
```

Трансляция программы. Создание объектного файла Blank32.o и файла листинга Blank32.lst (смотри рисунок 5), в который помещается листинг программы и сообщения об ошибках.

```
ld -m elf_i386 -o Blank32 Blank32.o
```

Компоновка в режиме эмуляции 32-х разрядной адресации. Создание исполняемого файла Blank32.out (смотри рисунок 3).

```
./Blank32
```

Выполнение программы (см. рисунок 4).

```
edb --run Blank32
```

Запуск отладки программы (см. рисунок 6).

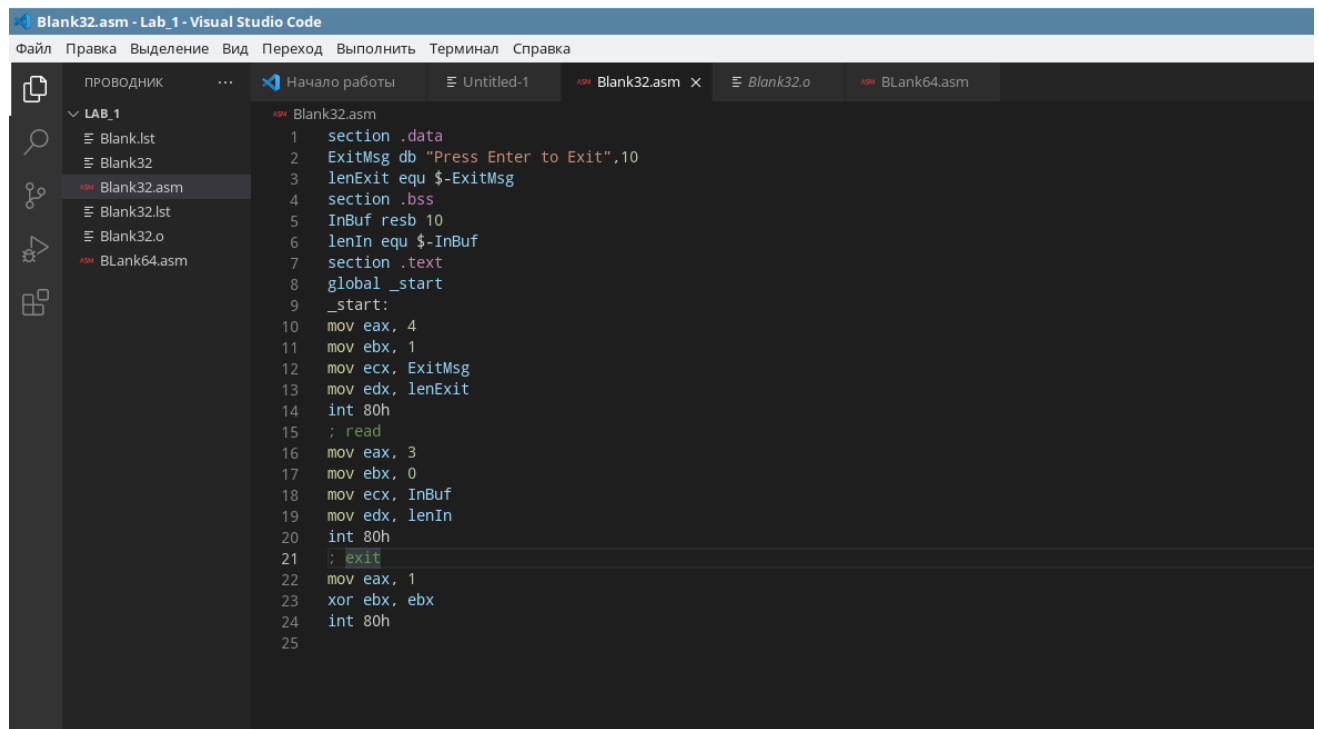


Рисунок 4 - Программа на языке ассемблера в редакторе Kate

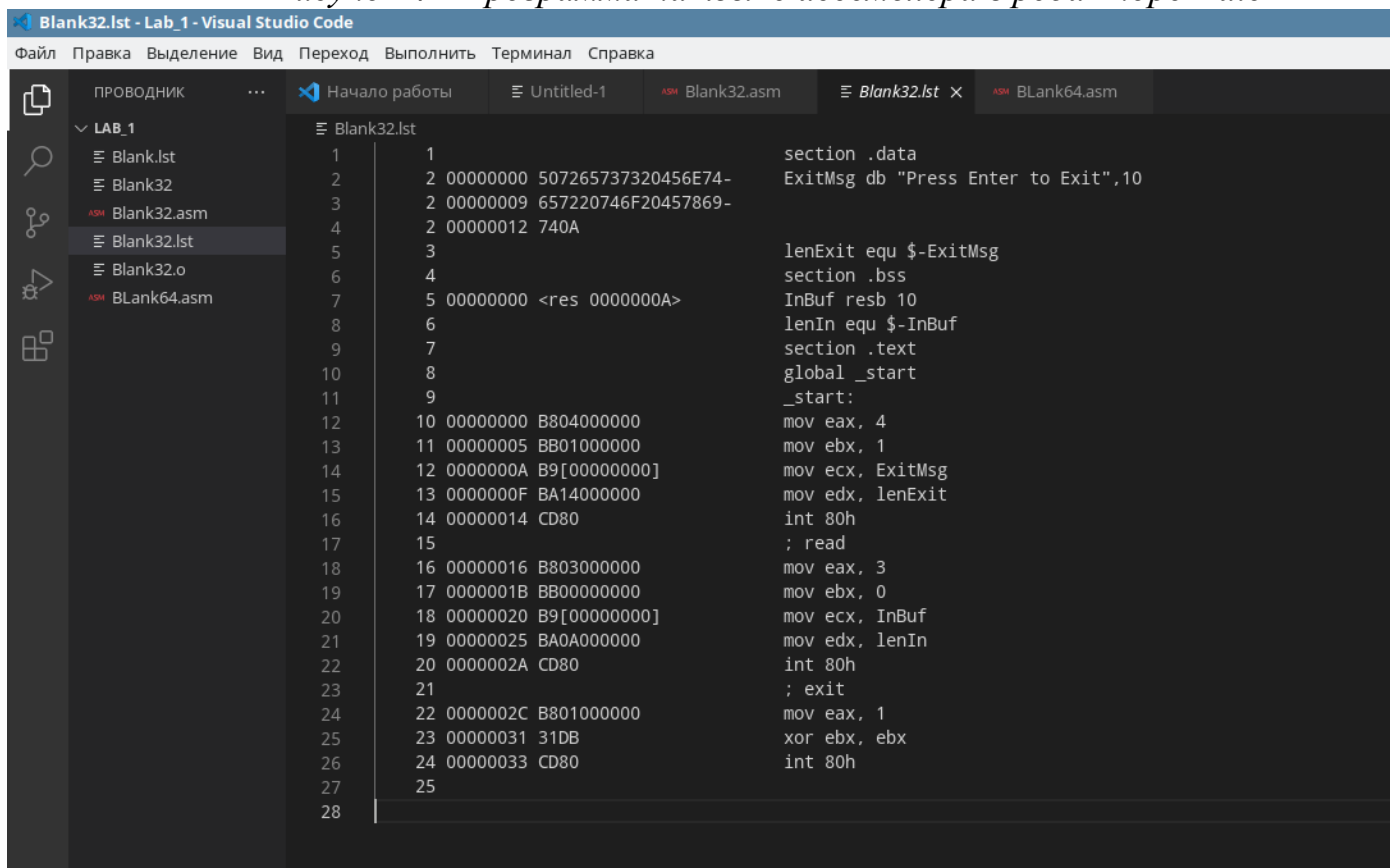


Рисунок 5 - Листинг программы

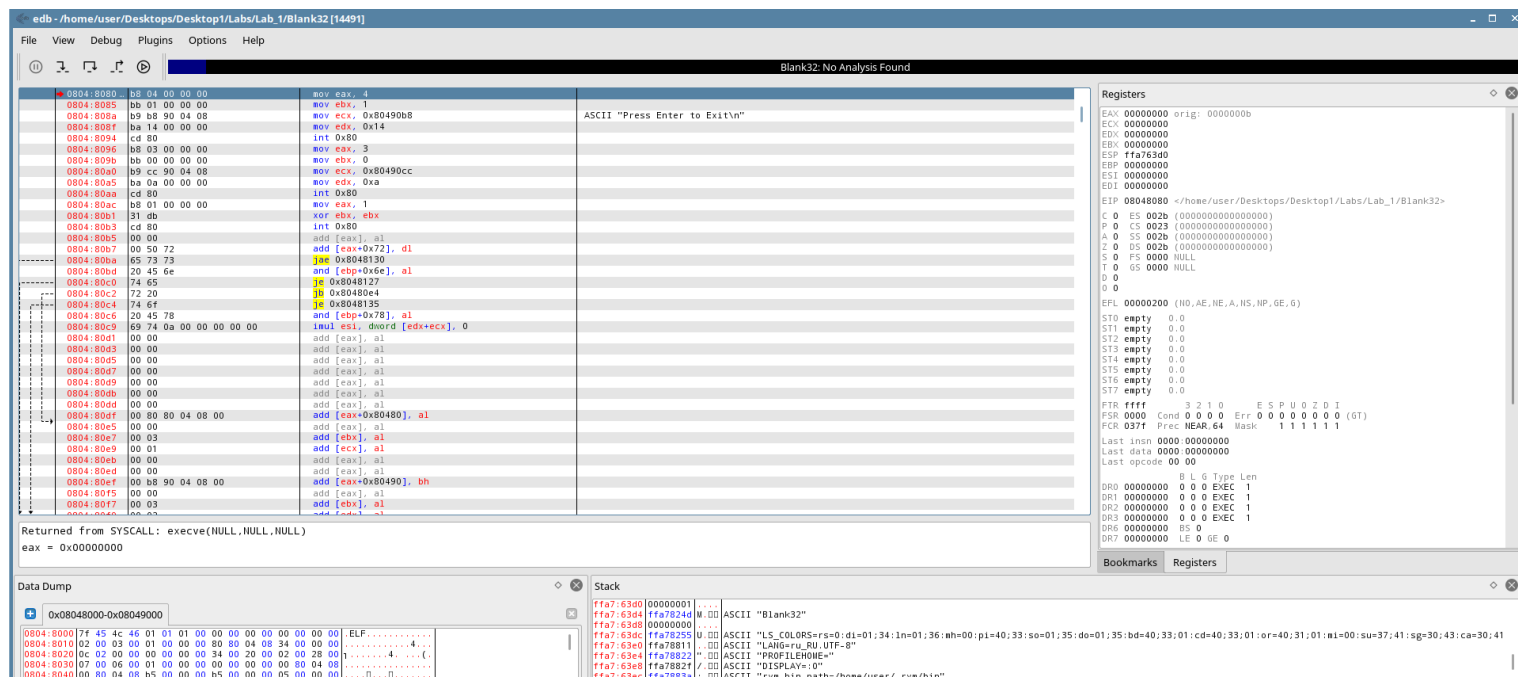


Рисунок 6 - Отладчик edb с исполняемой программой Blank32.out

Настройка на нужный адрес области сегмента данных (смотри рисунок 7).

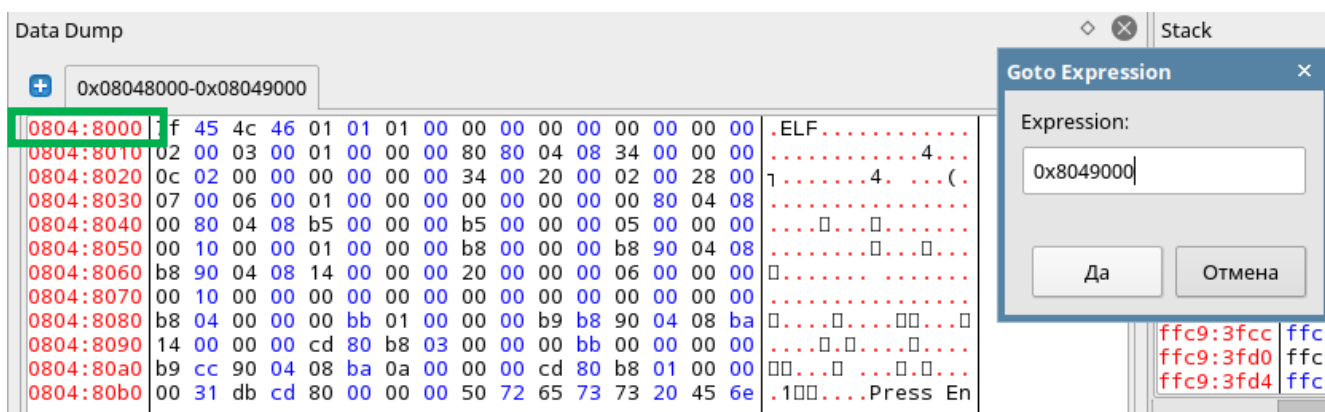


Рисунок 7 - Goto Expression

Относительный адрес данных, т.е. смещение в соответствующем сегменте, стал равным указанному в «Goto Expression» (смотри рисунок 8).

Data Dump																	
0x08049000-0x0804a000																	
0804:9000	7f	45	4c	46	01	01	01	00	00	00	00	00	00	00	00	.ELF.....	
0804:9010	02	00	03	00	01	00	00	00	80	80	04	08	34	00	00	.....4..	
0804:9020	0c	02	00	00	00	00	00	34	00	20	00	02	00	28	00	.....4..	(
0804:9030	07	00	06	00	01	00	00	00	00	00	00	00	80	04	08	.....	
0804:9040	00	80	04	08	b5	00	00	b5	00	00	05	00	00	00	00	.....	
0804:9050	00	10	00	00	01	00	00	b8	00	00	b8	90	04	08	.....		
0804:9060	b8	90	04	08	14	00	00	20	00	00	06	00	00	00	00	.....	
0804:9070	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
0804:9080	b8	04	00	00	00	bb	01	00	00	b9	b8	90	04	08	ba	.....	
0804:9090	14	00	00	00	cd	80	b8	03	00	00	bb	00	00	00	00	.....	
0804:90a0	b9	cc	90	04	08	ba	0a	00	00	cd	80	b8	01	00	00	.....	
0804:90b0	00	31	db	cd	80	00	00	50	72	65	73	73	20	45	6e	.1...Press En	

Рисунок 8 - Data Dump

```
mov eax, 4
```

Пересылка литерала со значение 4 в регистр EAX (смотри рисунок 9).

Registers	
EAX	00000004

Рисунок 9 - Значение регистра EAX

```
mov ecx, ExitMsg
```

Пересылка адреса, который символизирует идентификатор ExitMsg, в регистр ECX (смотри рисунки 10, 11 и 12).

mov eax, 4	
mov ebx, 1	
mov ecx, 0x80490b8	ASCII "Press Enter to Exit\n"
mov edx, 0x14	

Рисунок 10 - Дисассемблированный код в отладчике

Registers	
EAX	00000004
ECX	080490b8 ASCII "Press Enter to Exit\n"

Рисунок 11 - Значение регистра ECX

0804:90a0	b9 cc 90 04 08 ba 0a 00 00 00 cd 80 b8 01 00 00	□□...□ ...□.□...
0804:90b0	00 31 db cd 80 00 00 00 50 72 65 73 73 20 45 6e	.1□□...Press En
0804:90c0	74 65 72 20 74 6f 20 45 78 69 74 0a 00 00 00	ter to Exit ....

Рисунок 12 - Шестнадцатеричный дамп сегментов данных

## Задание 2. Вычисление выражения.

Вычислим выражение (смотри рисунок 13)

$$X = A + 5 - B$$

Рисунок 13 – Выражение

Напишем программу, выполняющую данную задачу (смотри рисунки 14, 15).

```

section .data ; сегмент инициализированных переменных
ExitMsg db "Press Enter to Exit",10 ; выводимое сообщение
lenExit equ $-ExitMsg
A dw -30
B dw 21

section .bss ; сегмент неинициализированных переменных
InBuf resb 10 ; буфер для вводимой строки
lenIn equ $-InBuf
X resw 1

section .text ; сегмент кода
global _start
_start:
mov AX, [A]
add AX, 5
sub AX, [B]
mov [X], AX

```

Рисунок 14 - Реализация программы по вычислению выражения (16-разрядные регистры)

66 a1 e0 90 04 08	mov ax, [0x80490e0]
66 83 c0 05	add ax, 5
66 2b 05 e2 90 04 08	sub ax, [0x80490e2]
66 a3 ee 90 04 08	mov [0x80490ee], ax

Рисунок 15 - Вычисление выражения – просмотр кода в отладчике

```
mov AX, [A]
```

Заносим в младшую часть регистра EAX, т.е. AX, содержимое по адресу A, т.е. -30.

Переведем значение регистра AX (2 байта) после пересылки в дополнительный код с учетом знакового бита (смотри таблицу 1 и результат – рисунок 17).

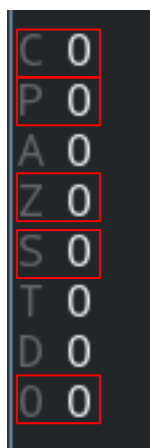
*Таблица 1. Внутреннее представление числа*

Десятичная система	-30
Прямой код	10000000 00011110 (первый бит знаковый)
Обратный код	11111111 11100001
Дополнительный код	11111111 11100010 = <b>FF E2</b> (в 16-ричной)

Рассмотрим флажковый регистр.

- 1) **CF** = 1 при переполнении в операциях с числами без знака (первый на рисунке 16).
- 2) **OF** = 1 при переполнении в операциях с числами со знаком (последний на рисунке 16).
- 3) **PF** = 1, если младший значащий байт результата содержит чётное число единичных (ненулевых) битов.
- 4) **ZF** = 1, если результат равен 0.
- 5) **SF** равен знаковому биту числа (0 – положительное число, 1 – отрицательное)

Эти флаги состояния отражают результат выполнения арифметических инструкций, таких как ADD, SUB, MUL, DIV (смотри рисунки 16, 17).



*Рисунок 16 - EFLAGS*





*Рисунок 17 - Результат первой команды*

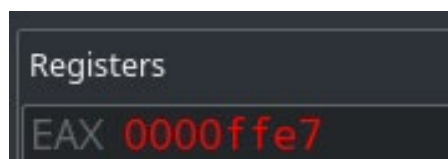
**add AX, 5**

Произведем расчет:  $-30 + 5 = -25 = \text{FF E7}$  (в дополнительном коде).

Флаги **CF, OF, ZF** равны **0**. Флаг **PF** = **1**, т.к. младший байт (из двух) числа - 25 (11100111) содержит четное количество единиц, **SF** = **1**, т.к. у отрицательного числа знаковый бит равен 1 (смотри рисунки 18, 19).



*Рисунок 18 - EFLAGS после add*



*Рисунок 19 - Результат второй команды*

**sub AX, [B]**

Произведем расчет:  $-25 - 21 = -46 = \text{FF D2}$  (в дополнительном коде).

Состояния флажкового регистра не изменились (смотри рисунки 20, 21).

C	0
P	1
A	0
Z	0
S	1
T	0
D	0
O	0

Рисунок 20 - EFLAGS после sub

Registers	
EAX	0000fd2

Рисунок 21 - Результат третьей команды

**mov [X], AX**

Поместим значение, находящиеся в регистре EAX в ячейку адреса, которую символизирует имя X = 0x080490ee.

После выполнения всех команд оперативной памяти остались числа, находящиеся по адресам A, B, X (смотри рисунок 22).

**A = -30 = FF E2** (в дополнительном коде в 16-ричной системе) – находится по адресу 0x080490e0 и занимает 2 байта.

**B = 21 = 00 15** (в 16-ричной системе) – находится по адресу 0x080490e2 и занимает 2 байта.

**X = -46 = FF D2** (в дополнительном коде в 16-ричной системе) – находится по адресу 0x080490ee и занимает 2 байта.

0804:90e0	e2 ff	15 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	d2 ff
0804:90f0	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00	00 00

Рисунок 22 - Dump Data после выполнения вычислений

Как мы можем заметить, числа в оперативной памяти расположены в обратном порядке, сначала младший байт, потом старший.

Проверим результат при использовании 32-битных регистров (смотри рисунок 23).

```

        section .data ; сегмент инициализированных переменных
ExitMsg db "Press Enter to Exit",10 ; выводимое сообщение
lenExit equ $-ExitMsg
A       dd -30
B       dd 21

        section .bss ; сегмент неинициализированных переменных
InBuf   resb 10 ; буфер для вводимой строки
lenIn   equ $-InBuf
X       resd 1

        section .text ; сегмент кода
global _start
_start:
        mov     EAX, [A]
        add     EAX, 5
        sub     EAX, [B]
        mov     [X], EAX

```

Рисунок 23 - Реализация программы по вычислению выражения (32-разрядные регистры)

Можно заметить, что теперь коды команд не содержат префикс размера операнда 66h (смотри рисунок 24).

a1 dc 90 04 08	mov eax, [0x80490dc]
83 c0 05	add eax, 5
2b 05 e0 90 04 08	sub eax, [0x80490e0]
a3 ee 90 04 08	mov [0x80490ee], eax

Рисунок 24 - Машинный и дисассемблированный коды

Используя регистр EAX, а не только его младшую часть AX, знаковый бит в дополнительном коде будет распространяться на все 32 бита (не на 16 бит, как в прошлом примере). Соответственно, в оперативной памяти также будут наблюдаться 4-х байтные числа в обратном порядке (смотри рисунок 25).



Рисунок 25 - Результат вычислений

### Задание 3. Внутреннее представление данных.

Введем следующие инициализированные и неинициализированные данные (смотри рисунок 26).

```

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
    db 100101B
beta db 23,23h,0ch
sdk db "Hello",10
min dw -32767
ar dd 12345678h
valar times 5 db 8

InBuf section .bss ; сегмент неинициализированных переменных
resb 10 ; буфер для вводимой строки
lenIn equ $-InBuf

alu resw 10
f1 resb 5

```

Рисунок 26 - Дisasсемблированный код

Рассмотрим размещение в памяти инициализированных переменных (смотри рисунок 27).

Data Dump																	
+		0x08049000-0x0804a000															
0804:9090	14	00	00	00	cd	80	b8	03	00	00	00	bb	00	00	00	00	....
0804:90a0	b9	e8	90	04	08	ba	0a	00	00	00	cd	80	b8	01	00	00	...
0804:90b0	00	31	db	cd	80	00	00	00	50	72	65	73	73	20	45	6e	.1... Press En
0804:90c0	74	65	72	20	74	6f	20	45	78	69	74	0a	ff	00	01	80	ter to Exit
0804:90d0	ff	10	25	17	23	0c	48	65	6c	6c	6f	0a	01	80	78	56	...%.#_Hello ..xV
0804:90e0	34	12	08	08	08	08	00	00	00	00	00	00	00	00	00	00	4.....
0804:90f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Рисунок 27 - Data Dump. Выделен байт, с которого начинаются значения.

Сопоставим значения данных, которые мы задаем на языке ассемблера в задании 3, с их внутренним представлением в оперативной памяти (смотри таблицу 2).

*Таблица 2 - Инициализированные данные.*

Имя поля данных	Директива (тип)	Константы (последовательность констант)	Внутреннее представление
val1	db	255	FF
chart (1)	dw	256	00 01
lue3 (2)	dw	-128	80 ff
v5	db	10h	10
	db	100101B	25
Beta	db	23, 23h, 0ch	17 23 0c
sdk (3)	db	“Hello”, 10	48 65 6c 6c 6f 0a
min (4)	dw	-32767	01 80
Ar	dd	12345678h	78 56 34 12
Valar	times 5 db	8	08 08 08 08 08

Байты данных в оперативной памяти располагаются в обратном порядке. Объясним внутреннее представление некоторых «переменных», обозначенных цифрой в скобках.

1) 256 = 00000001 00000000 (в 2-ичной) = **01 00** (в 16-ричной)

2) -128 = 10000000 10000000 (прямой код со знаковым битом) = 11111111 01111111 (обратный код) = 11111111 10000000 (дополнительный код) = **ff 80** (в 16-ричной)

3) По таблице ASCII символу ‘H’ сопоставлен код 72 (в 10-ричной) = **48** (в 16-ричной). Символу ‘e’ сопоставлен код 101 (в 10-ричной) = **65** (в 16-ричной). Последнее число в списке инициализации 10 = **0a** (в 16-ричной), обозначающее конец строки в ОС Linux.

4)  $-32767 = 1111\ 1111\ 1111\ 1111$  (прямой код со знаковым битом) = **1000 0000 0000 0000** (обратный код) = **1000 0000 0000 0001** (дополнительный код) = **80 01** (в 16-ричной)

Адреса неинициализированных данных символизируются с именами `alu` и `fl`. Для `alu` будет зарезервировано  $10 * 2 = 20$  байт, так как использована директива `resw`, для `fl` –  $5 * 1 = 5$  байт, т.к. использована директива `resb`.

#### Задание 4.

Определим в памяти следующие данные (смотри рисунки 28 и 29).

- 1) целое число 25 размером 2 байта со знаком;
- 2) двойное слово, содержащее число -35;
- 3) символьную строку, содержащую ваше имя (русскими буквами и латинскими буквами).

*Рисунок 29 - Data Dump. Выделен байт, с которого начинаются значения.*

Сопоставим значения данных, которые мы задаем на языке ассемблера в задании 4, с их внутренним представлением в оперативной памяти (смотри таблицу 3).

Таблица 3 - Инициализированные данные.

Имя поля данных	Директива (тип)	Константы (последовательность констант)	Внутреннее представление
val1	dw	25	19 00
val2	dd	-35	dd ff ff ff
str_en	db	"Tharindu", 10	4d 69 6b 68 61 69 6c 0a
str_ru	db	"Таринду", 10	cc e8 f5 e0 e8 eb 0a

*Замечание.* Русские символы не отображаются из-за того, что используется кодировка UTF-16 (таблица Unicode), в которой русские символы соответствуют кодам, начиная с 1040 ('А'), т.е. кодируются двумя байтами. Коду **CC** (16-ричный) соответствует данный символ – Ĭ (смотри рисунок 30).

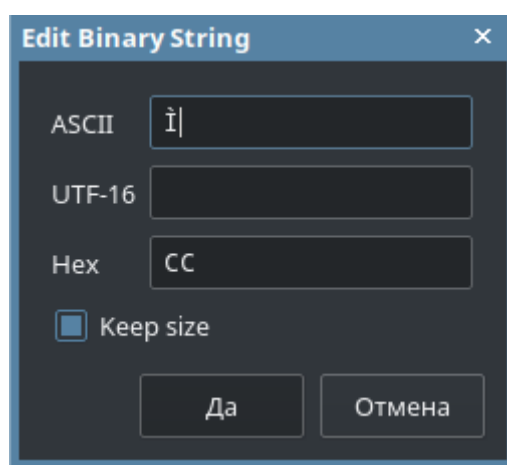


Рисунок 30 - Символ с кодом CC в таблице Unicode

### Задание 5.

Определим константы в программе, которые во внутреннем представлении выглядят как **25 00** и **00 25** (смотри рисунки 31, 32).

```
val1_1    dw 25h
val1_2    dw 37
val1_3    dw 100101b
val1_4    dw "%"
val1_5    dw 45o

val2_1    dw 2500h
val2_2    dw 9472
val2_3    dw 100101000000000b
val2_5    dw 22400o
```

Рисунок 31 - Инициализированные данные.

0804:90c0	74 65 72 20 74 6f 20 45 78 69 74 0a 25 00 25 00	ter to Exit %.
0804:90d0	25 00 25 00 25 00 00 25 00 25 00 25 00 25 00 00	%.%.%.%.%.%.%.%
0804:90e0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Рисунок 32 - Внутреннее представление значений.

### Задание 6.

Добавим в программу переменную F1 = 65535 размером слово и переменную F2 = 65535 размером двойное слово (смотри рисунок 33). Сложим эти переменные с единицей и проанализируем результат (смотри рисунки 34, 35).

```
Blank32_5.asm

section .data ; сегмент инициализированных переменных
ExitMsg db "Press Enter to Exit",10 ; выводимое сообщение
lenExit equ $-ExitMsg

F1      dw 65535
F2      dd 65535

section .bss ; сегмент неинициализированных переменных
InBuf   resb 10 ; буфер для вводимой строки
lenIn   equ $-InBuf

section .text ; сегмент кода
global _start

_start:

    add [F1], 1
    add [F2], 1
```

Рисунок 33 - Дисассемблированный код.



0804:90d0	74	6f	20	45	78	69	74	0a	ff	ff	ff	ff	00	00	00	00
0804:90e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0804:90f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Рисунок 34 - Исходные значения в памяти

C	1
P	1
A	1
Z	1
S	0
T	0
D	0
O	0

Рисунок 35 - EFLAGS после сложения  $F1 + 1$ .

Состояния флажкового регистра после первой операции (смотри рисунок 35):

- 1) **CF** = 1 – произошло переполнение, т.к.  $65536 = 01\ 00\ 00$  (в 16-ричной). Видим, что для хранения числа 65536 нужно минимум 3 байта (директива dw определяет 2 байта)
- 2) **PF** = 1 – младший байт числа 65536 содержит четное количество единиц (0 единиц).
- 3) **ZF** = 1 – результат равен 0 (00 00 в 16-ричной).

C	0
P	1
A	1
Z	0
S	0
T	0
D	0
O	0

Рисунок 36 - EFLAGS после сложения  $F2 + 1$ .

Состояния флажкового регистра после второй операции (смотри рисунок 36):

- 1) **CF** = 0 – **не** произошло переполнение, т.к. 65536 = 00 01 00 00 (в 16-ричной).
- 2) **PF** = 1 – младший байт числа 65536 содержит четное количество единиц (0 единиц).
- 3) **ZF** = 0 – результат **не** равен 0.

Убедится в правильности рассуждений можно, посмотрев в Dump памяти (смотри рисунок 37)

0804:90d0	74	6f	20	45	78	69	74	0a	00	00	00	00	01	00	00	00
0804:90e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0804:90f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Рисунок 37 - Новые значения в памяти

### Контрольные вопросы.

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

Язык ассемблера или ассемблер — язык низкого уровня с командами, соответствующими командам процессора (так называемым машинным командам).

- 2) *Из каких частей состоит заготовка программы на ассемблере?*

Заготовка программы состоит из сегмента инициализированных и неинициализированных переменных и сегмента кода, в котором описаны системные функции 4, 3 и 1 (write, read и exit соответственно).

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

Ответ на странице 3

- 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:  $A \text{ dw } 5, -5$  ? Как те же числа будут выглядеть после загрузки в регистр AX?

Положительные числа представляются в прямом двоичном коде, отрицательные – в дополнительном. 5 будет представлена как **05 00**, -5 будет представлена как **FB FF**. В регистре AX значения данных хранятся в прямом порядке, в отличие от оперативной памяти, поэтому 5 будет выглядеть как **00 05**, -5 – как **FF FB**.

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

Программа представлена на рисунке 38.

```
section .data ; сегмент инициализированных переменных
ExitMsg db "Press Enter to Exit",10 ; выводимое сообщение
lenExit equ $-ExitMsg
A db 5
B db 3

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     AL, [A]
    add     AL, [B]
    mov     [C], AL
    ; write
```

Рисунок 38 - Дисассемблированный код. Задание.

**Вывод:** была выполнена лабораторная работа 1, которая была связана с изучением среды и отладчика ассемблера. В ходе выполнения заданий были изучены структура простейшей программы на ассемблере NASM, ввод, трансляция, компоновка и выполнение программы, работа с отладчиком для анализа значений регистров и внутреннего представления данных в оперативной памяти. Результаты тестирования корректны, работы выполнена успешно!