



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 (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.03 Прикладная информатика.

О Т Ч Е Т

по лабораторной работе № 1

Название: Изучение среды и отладчика ассемблера

Дисциплина: Машинно-зависимые языки и основы компиляции

Студент

ИУ6-45Б
(Группа)

(Подпись, дата)

Л.Э. Барсегян
(И.О. Фамилия)

Преподаватель

(Подпись, дата)

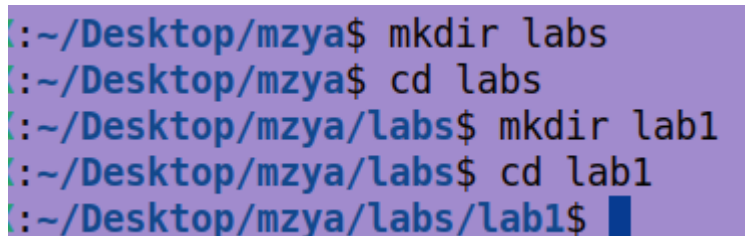
Я. С. Петрова
(И.О. Фамилия)

Москва, 2022

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.

Задания 1.2.1-1.2.2

Создадим каталог и подкаталог, используя команду `mkdir` и объявим подкаталог `labs/lab1` текущим (см. Рисунок 1)



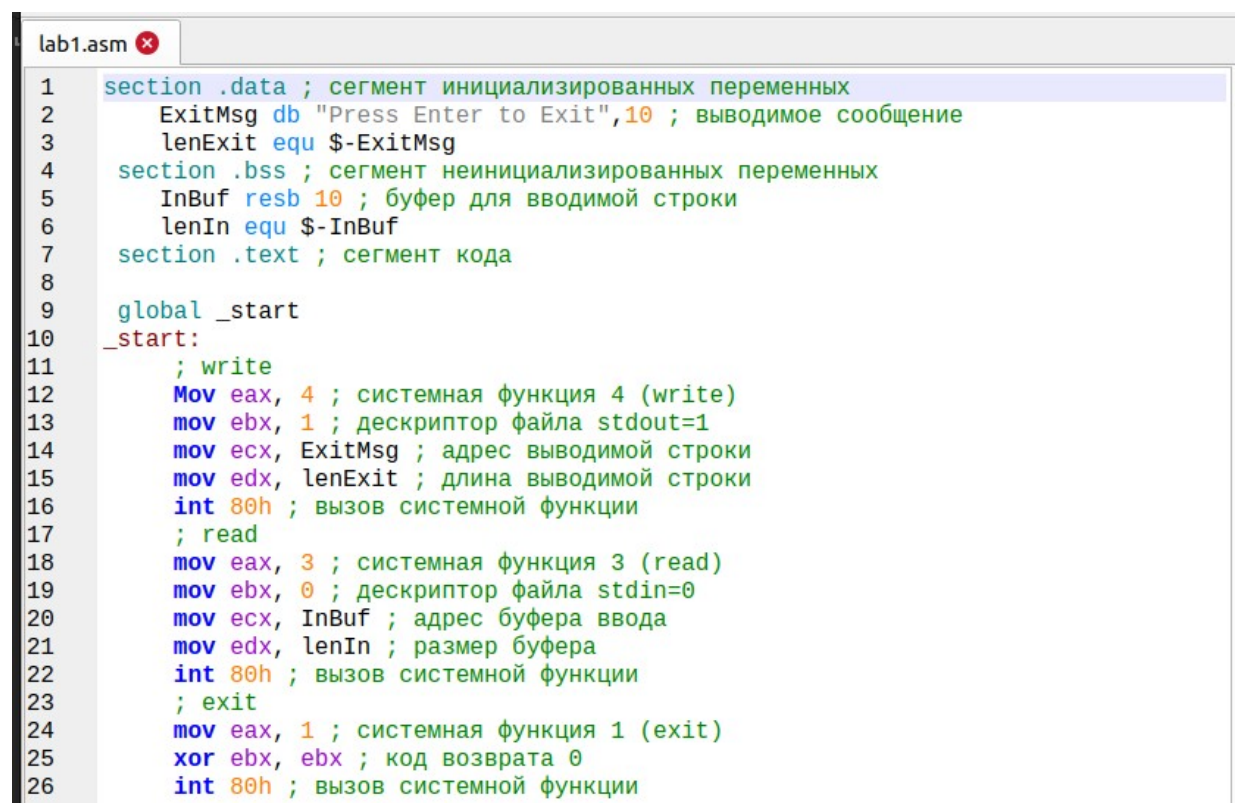
```
:~/Desktop/mzya$ mkdir labs
:~/Desktop/mzya$ cd labs
:~/Desktop/mzya/labs$ mkdir lab1
:~/Desktop/mzya/labs$ cd lab1
:~/Desktop/mzya/labs/lab1$
```

Рисунок 1 — создание каталога `labs` и подкаталога `lab1`

Задание 1.2.3

Введем заготовку 32-х или 64-х разрядной программы на ассемблере.

Сохраним программу с именем `lab1.asm` в подкаталоге `labs/lab1`. (см. Рисунок 2)



```
lab1.asm
1  section .data ; сегмент инициализированных переменных
2      ExitMsg db "Press Enter to Exit",10 ; выводимое сообщение
3      lenExit equ $-ExitMsg
4  section .bss ; сегмент неинициализированных переменных
5      InBuf resb 10 ; буфер для вводимой строки
6      lenIn equ $-InBuf
7  section .text ; сегмент кода
8
9  global _start
10 _start:
11     ; write
12     mov eax, 4 ; системная функция 4 (write)
13     mov ebx, 1 ; дескриптор файла stdout=1
14     mov ecx, ExitMsg ; адрес выводимой строки
15     mov edx, lenExit ; длина выводимой строки
16     int 80h ; вызов системной функции
17     ; read
18     mov eax, 3 ; системная функция 3 (read)
19     mov ebx, 0 ; дескриптор файла stdin=0
20     mov ecx, InBuf ; адрес буфера ввода
21     mov edx, lenIn ; размер буфера
22     int 80h ; вызов системной функции
23     ; exit
24     mov eax, 1 ; системная функция 1 (exit)
25     xor ebx, ebx ; код возврата 0
26     int 80h ; вызов системной функции
```

Рисунок 2 - Текст 64-разрядной программы

Задание 1.2.4

Выполним трансляцию программы с листингом. Для 64-разрядной программы введем в терминале команду:

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

Убедимся, что операция прошла без ошибок, воспользовавшись командой ls. (см. Рисунок 3)

```
liubava@liubava-NBLK-WAX9X:~/Desktop/mzya/labs/lab1$ nasm -f elf64 lab1.asm -l lab1.lst
liubava@liubava-NBLK-WAX9X:~/Desktop/mzya/labs/lab1$ ls -l
total 56
-rw-rw-r-- 1 liubava liubava 1245 фев 22 21:03 lab1.asm
-rw-rw-r-- 1 liubava liubava 2358 фев 22 21:29 lab1.lst
-rw-rw-r-- 1 liubava liubava 1072 фев 22 21:29 lab1.o
-rw-rw-r-- 1 liubava liubava 41694 фев 22 21:10 'ЛРМ1 Барсегян Люба МЗЯиОК.docx'
```

Рисунок 3 — Проверка трансляции программы с листингом.

Задание 1.2.5

Для компоновки 64-х разрядной программы, которая будет выполняться на компьютере той же размерности, следует ввести следующую команду:

```
ld -o lab1 lab1.o
```

Если все прошло без ошибок, то в том же каталоге появится файл исполняемой программы lab1. Проверим с помощью команды ls. (см. Рисунок 4)

```
liubava@liubava-NBLK-WAX9X:~/Desktop/mzya/labs/lab1$ ls -l
total 68
-rwxrwxr-x 1 liubava liubava 9016 фев 22 21:36 lab1
-rw-rw-r-- 1 liubava liubava 1245 фев 22 21:03 lab1.asm
-rw-rw-r-- 1 liubava liubava 2358 фев 22 21:36 lab1.lst
-rw-rw-r-- 1 liubava liubava 1072 фев 22 21:36 lab1.o
-rw-rw-r-- 1 liubava liubava 41694 фев 22 21:10 'ЛРМ1 Барсегян Люба МЗЯиОК.docx'
```

Рисунок 4 — Проверка создания исполняемого файла lab1

Задание 1.2.6

Запустим программу на выполнение, как указано на рисунке 5. Запущенная программа выводит текст «Press Enter to Exit» и ожидает нажатия клавиши Enter. После нажатия клавиши Enter выполнение программы завершится.

```
liubava@liubava-NBLK-WAX9X:~/Desktop/mzya/labs/lab1$ ./lab1
Press Enter to Exit
liubava@liubava-NBLK-WAX9X:~/Desktop/mzya/labs/lab1$
```

Рисунок 5 — Запуск программы

Задание 1.2.7

Запустим отладчик edb. Для этого следует в окне терминала ввести команду «edb» (см. Рисунок 6) и открыть исполняемую программу lab1 (см. Рисунок 7).

```
liubava@liubava-NBLK-WAX9X:~/Desktop/mzya/labs/lab1$ edb
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway.
Starting edb version: 1.3.0
Please Report Bugs & Requests At: https://github.com/eteran/edb-debugger/issues
comparing versions: [4864] [4864]
```

Рисунок 6 — запуск отладчика edb

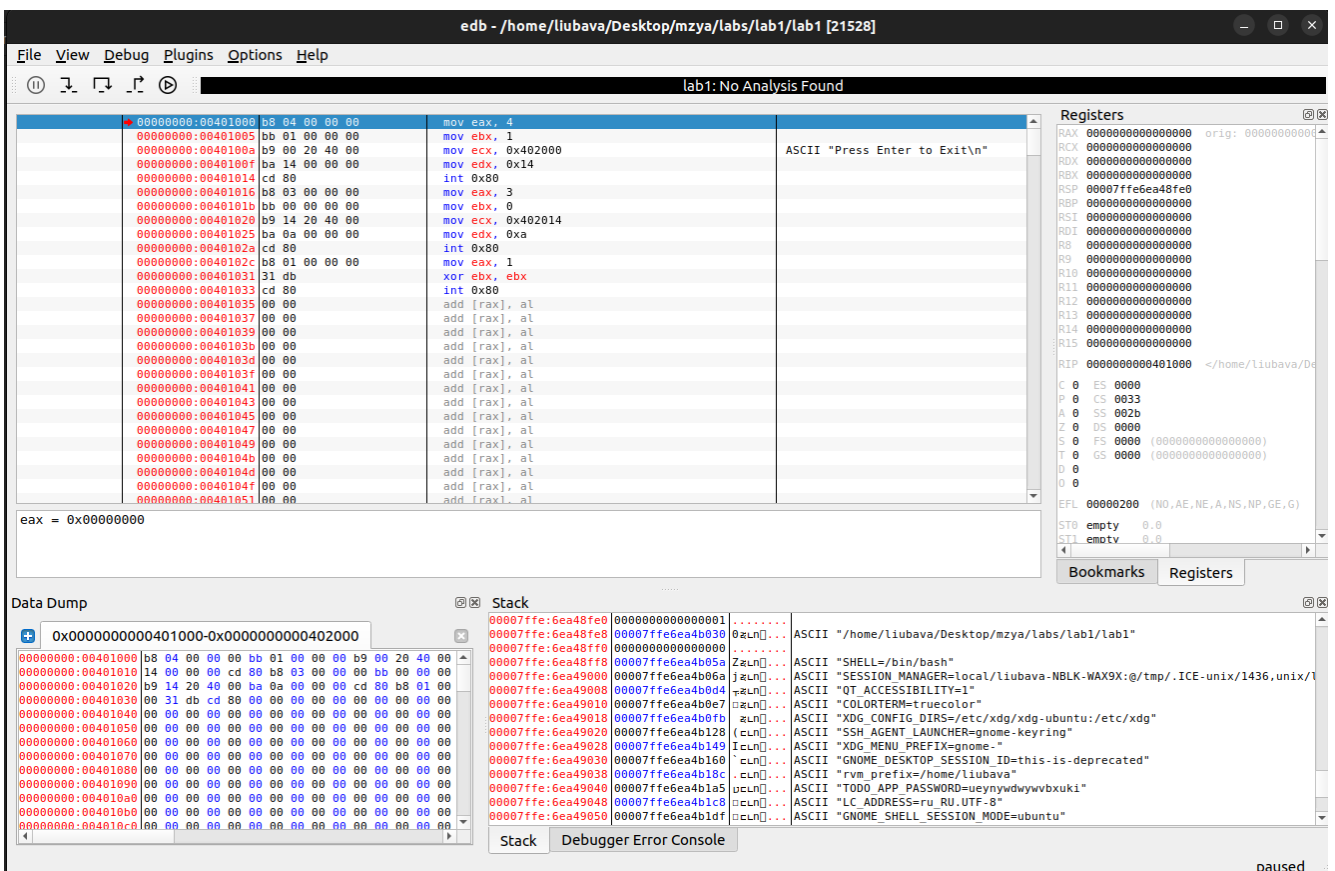


Рисунок 7 — открытие исполняемой программы lab1 в «edb»

Задание 1.2.8

Для изучения возможностей отладчика добавим в код программы несколько команд для вычисления результата следующего выражения: $X = A + 5 - B$.

Данные программы зададим константами, поместив их описание в раздел инициализированных данных .data. Для результата вычислений — переменной X — необходимо зарезервировать место, поместив описание соответствующей

неинициализированной переменной в раздел неинициализированных данных .bss. Фрагмент кода программы, выполняющей сложение и вычитание, поместим в сегмент кодов после метки start. Изменения показаны на рисунке 8.

```

lab1.asm
1  section .data ; сегмент инициализированных переменных
2      ExitMsg db "Press Enter to Exit",10 ; выводимое сообщение
3      lenExit equ $-ExitMsg
4      A dw -30
5      B dw 21
6  section .bss ; сегмент неинициализированных переменных
7      InBuf resb 10 ; буфер для вводимой строки
8      lenIn equ $-InBuf
9      X resd 1
10 section .text ; сегмент кода
11
12 global _start
13 _start:
14     ; write
15     mov eax, 4 ; системная функция 4 (write)
16     mov ebx, 1 ; дескриптор файла stdout=1
17     mov ecx, ExitMsg ; адрес выводимой строки
18     mov edx, lenExit ; длина выводимой строки
19     int 80h ; вызов системной функции
20     ;calculate X
21     mov EAX,[A] ; загрузить число A в регистр EAX
22     add EAX,5
23     sub EAX,[B] ; вычесть число B, результат в EAX
24     mov [X],EAX ;
25     ;output X
26     mov eax, 4 ; системная функция 4 (write)
27     mov ebx, 1 ; дескриптор файла stdout=1
28     mov ecx, X ; адрес выводимой строки
29     mov edx, 1 ; длина выводимой строки
30     int 80h ; вызов системной функции
31     ; read
32     mov eax, 3 ; системная функция 3 (read)
33     mov ebx, 0 ; дескриптор файла stdin=0
34     mov ecx, InBuf ; адрес буфера ввода
35     mov edx, lenIn ; размер буфера
36     int 80h ; вызов системной функции
37     ; exit
38     mov eax, 1 ; системная функция 1 (exit)
39     xor ebx, ebx ; код возврата 0
40     int 80h ; вызов системной функции

```

Рисунок 8 — Код программы lab1.asm

Выполним трансляцию, компоновку программы и загрузку в отладчик.

Задание 1.2.9

Проследим в отладчике выполнение программы и зафиксируем результаты выполнения каждой выполненной программы в таблицу 1.

Код операции	Код операции отладчике	Код операции в шестнадцатеричном представлении
mov EAX,[A]	mov eax, [0x402014]	8B 04 25 14 20 40 00
add EAX,5	add eax, 5	83 C0 05
sub EAX,[B]	sub eax, [0x402016]	2B 04 25 16 20 40 00
mov [X],EAX	mov [0x402022], eax	89 04 25 22 20 40 00

Таблица 1

Рассмотрим подробнее первую операцию с переменной А. Как можно заметить, в отладчике А представлено в виде адреса. Перейдем по этому адресу (см. рисунок 9) и переведем содержимое из шестнадцатеричной системы счисления в десятичную, учитывая обратный порядок записи. (см. рисунок 10).

00000000:00402014	e2 ff	loop 0x402015
00000000:00402016	15 00 00 00 00	adc eax, 0

Рисунок 9 — адрес переменной А

Enter hex number

ffe2

16

= Convert

× Reset

↕ Swap

Decimal number (5 digits)

65506

10

Decimal from signed 2's complement (2 digits)

-30

10

Рисунок 10 — перевод числа А в десятичную систему счисления

Как можно заметить, получили то же число, что и присвоили переменной A в секции инициализированных переменных.

При выполнении операции `mov EAX,[A]`, в регистр записывается адрес переменной A (см. рисунок 11)



RAX 0000000015ffe2

Рисунок 11 — содержимое регистра RAX

после выполнения первой операции

После выполнения операции `add EAX,5` содержимое регистра RAX увеличивается на 5, что показано на рисунке 12.



RAX 0000000015ffe7

Рисунок 12 — содержимое регистра RAX

после выполнения второй операции

Задание 1.2.10

Занесем следующие строки в разделы описания инициализированных и неинициализированных данных и определим с помощью отладчика внутреннее представление этих данных в памяти.

```
val  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

alu resw 10

f1 resb 5

С помощью отладчика определим внутреннее представление этих данных в памяти и занесем результаты в таблицу 2.

Название переменной

Внутреннее представление

val

3 ff 00	inc dword [rax]
---------	-----------------

chart

9 00 01	add [rcx], al
---------	---------------

lue3

b 80 ff 10	cmp bh, 0x10
------------	--------------

v5

10 25 17 23 0c 48	adc [rel 0x484c433a], ah
-------------------	--------------------------

beta

17	db 0x17
23 0c 48	and ecx, [rax+rcx*2]

sdk

48 65 6c	insb [rdi], dx
6c	insb [rdi], dx
6f	outsd dx, [rsi]
0a 01	or al, [rcx]

min

01 80 78 56 34 12	add [rax+0x12345678], eax
-------------------	---------------------------

ar

78 56	js 0x402082
-------	-------------

valar

08 08	or [rax], cl
08 08	or [rax], cl
08 00	or [rax], al

alu

00 00	add [rax], al
-------	---------------

f1

00 00	add [rax], al
-------	---------------

Задание 1.2.11

Определим 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).

Описание данных в сегменте инициализированных данных представлено на рисунке 13.

```
integer dw 25
negative_word dd -35
my_name db "Люба Барсегян", "Liuba Barsegian"
```

Рисунок 13 — описание данных задания 1.2.11

Внутреннее представление данных запишем в таблицу 3.



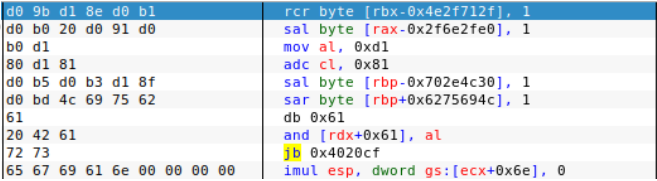
Описание	Внутреннее представление	Пояснение
Integer dw 25		Так как порядок записи байтов обратный, то содержимое переменной — 0019, причем первые два нуля незначащие. Переводя число 19 из шестнадцатеричной СС в десятичную, получаем число 25.
negative_word dd -35		DD (16) = -35 (10)
my_name db «Люба Барсегян», «Liuba Barsegian»		Пояснение ниже

Таблица 3 — Внутреннее представление переменных задания 1.2.11

На рисунках 14 и 15 можно увидеть перевод строк, содержащих мое имя, на русском и на английском соответственно.

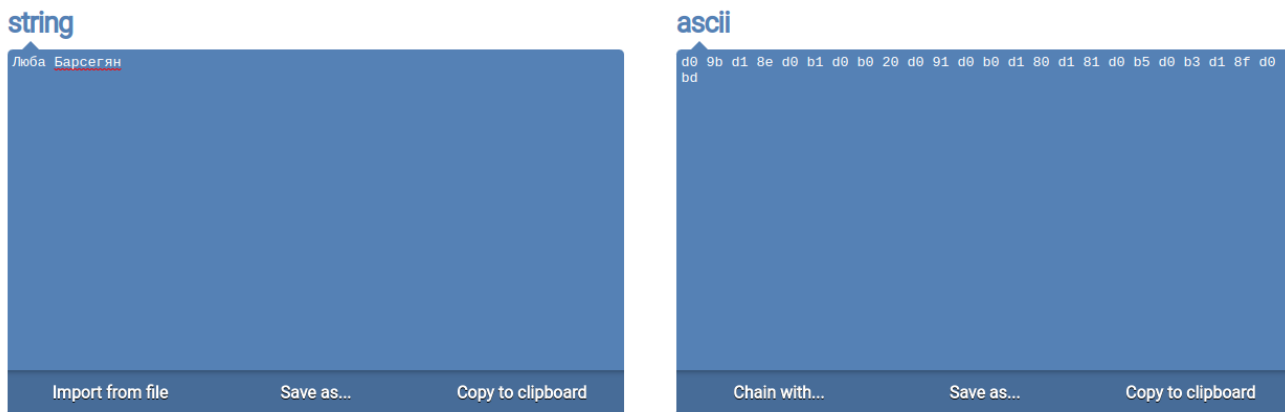


Рисунок 14 — Перевод строки с именем на русском в шестнадцатеричную СС

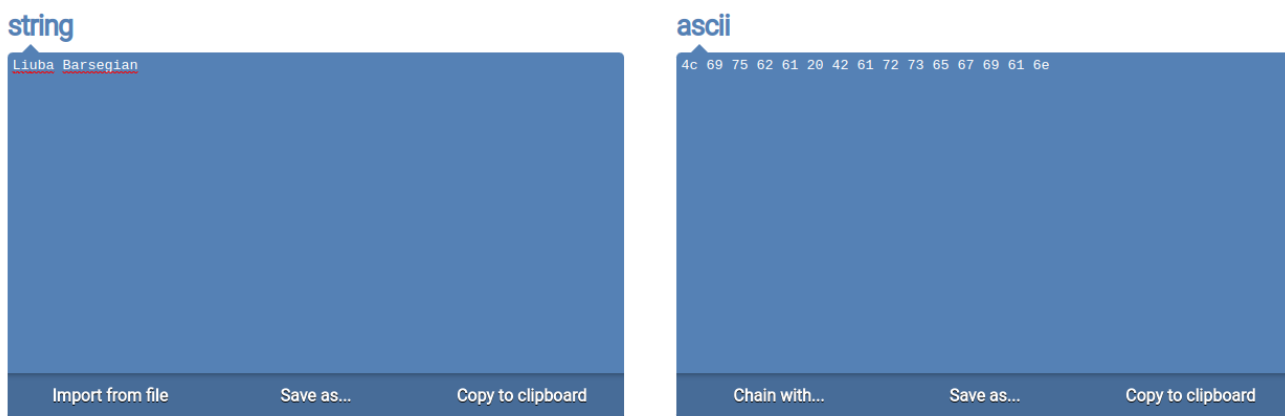


Рисунок 15 — Перевод строки с именем на английском в шестнадцатеричную СС

Как можно заметить по результатам, внесенным в таблицу 3, эти строки записаны последовательно и совпадают с переводом в шестнадцатеричную систему счисления, а также заканчиваются терминирующим нулем, что свойственно строкам.

Задание 1.2.12

Определим в программе числа, которые во внутреннем представлении отладчика будут выглядеть как **25 00**.

Для этого вспомним, что порядок записи обратный, следовательно, само число должно быть равно числу **0025**. Переведем это число из шестнадцатеричной системы счисления в десятичную и получим **37**.

Запишем число 37 несколькими способами (см. рисунок 16)

```
var_1 dw 37
var_2 dd 37
var_3 db 37
var_4 db "%"
var_5 dw "%"
var_6 dw 25h
var_7 dw 100101b
```

Рисунок 16 — объявление переменных

Внутреннее представление данных занесем в таблицу 4.

Объявление	Внутреннее представление
Var_1 dw 37	00000000:00402061 25 00 25 00 00
Var_2 dd 37	00000000:00402063 25 00 00 00 25
Var_3 db 37	00000000:00402067 25 25 25 00 25
Var_4 db «%»	00000000:00402068 25 25 00 25 25
Var_5 dw «%»	00000000:00402069 25 00 25 00 25
Var_6 dw 25h	00000000:0040206b 25 00 25 00 00
Var_7 dw 100101b	00000000:0040206d 25 00 00 00 00

Таблица 4 — внутренне представление данных задания 1.2.12

Замечание: в случаях 3 и 4, где используется db, выделяется лишь один байт памяти, поэтому незначащие нули не записываются и число остается в виде 25.

Теперь определим в программе числа, которые во внутреннем представлении отладчика будут выглядеть как **00 25**.

Помня об обратном порядке записи, запишем само число, как равное числу **2500**. Переведем это число из шестнадцатеричной системы счисления в десятичную и получим **9472**.

Запишем число 9472 несколькими способами (см. рисунок 17)

```
var_8 dw 9472
var_9 dd 9472
var_10 dw 100101000000000b
var_11 dw 2500h
```

Рисунок 17 — объявление переменных

Внутреннее представление данных занесем в таблицу 5.

Объявление переменных	Внутреннее представление
-----------------------	--------------------------

var_8 dw 9472	00000000:0040206f 00 25 00 25 00 00
var_9 dd 9472	00000000:00402071 00 25 00 00 00 25
var_10 dw 100101000000000b	00000000:00402075 00 25 00 25 00 00
var_11 dw 2500h	00000000:00402077 00 25 00 00 00 00

Таблица 5 — внутреннее представление данных задания 1.2.12

Задание 1.2.13

Добавим в программу переменную F1=65535 размером слово и переменную F2= 65535 размером двойное слово. Также вставим в программу команды сложения этих чисел с 1:

add[F1],1

add[F2],1

На рисунке 18 код программы сложения переменных F1 и F2 с 1.

```

;add f1 + 1
add WORD[F1],1
;add f2 + 1
add DWORD[F2],1

```

Рисунок 18 — код программы 1.2.13

Для начала найдем обратимся по адресу переменных и посмотрим содержимое (см. рисунки 19 и 20 для переменных F1 и F2 соответственно)

00000000:00402079	ff	db 0xff
00000000:0040207a	ff	db 0xff

Рисунок 19 — внутреннее представление F1

00000000:0040207b	ff	db 0xff
00000000:0040207c	ff 00	inc dword [rax]

Рисунок 20 — внутреннее представление F2

Выполним программу и зафиксируем изменения.

В случае с dw произошло переполнение и остались нули, так как изначально был выделен всего лишь один байт (см. рисунок 21)

00000000:00402079	00 00	add [rax], al
-------------------	-------	---------------

Рисунок 21 — прибавление 1 к F1

Рассмотрим случай с dd, представленный на рисунке 22.

00000000:0040207b	00 00	add [rax], al
00000000:0040207d	01 00	add [rax], eax

Рисунок 22 — прибавление 1 к F2

Так как под переменную F2 выделено 4 байта памяти, то рассмотрим 4 байта в обратном порядке. Число 0001000 переведем из шестнадцатеричной СС в десятичную и получим число 65536. Следовательно, переполнение не произошло из-за достаточного количества выделенной памяти, операция добавления 1 к содержимому переменной прошла успешно.

Обратим внимание на флаги при выполнении данных операций. Флаги при добавлении 1 к F1 показаны на рисунке 23, а флаги при добавлении 1 к F2 — на рисунке 24.

```

C 1  ES 0000
P 1  CS 0033
A 1  SS 002b
Z 1  DS 0000
S 0  FS 0000 (0000000000000000)
T 0  GS 0000 (0000000000000000)
D 0
O 0

```

Рисунок 23 — флаги при выполнении первой операции

```

C 0  ES 0000
P 1  CS 0033
A 1  SS 002b
Z 0  DS 0000
S 0  FS 0000 (0000000000000000)
T 0  GS 0000 (0000000000000000)
D 0
O 0

```

Рисунок 24 — флаги при выполнении второй операции

Распишем флаги и их назначения.

C — флаг переноса из старшего байта

P — флаг четности

A — флаг переноса из младшего байта в старший

Z — флаг, указывающий, что возвращаемый результат равен нулю

O — флаг переполнения

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.

2. Из каких частей состоит заготовка программы на ассемблере?

Заготовка программы на языке ассемблера состоит из трех частей:

- .text (сегмент кода)
- .data (сегмент инициализированных данных)
- .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` 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.

4. 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.

Результаты выполнения программы

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 СС будет равно 5 в то время, как -5 равняется FFFB.

Проверим, в каком виде отладчик покажет эти числа. На рисунке 25 первые два байта — это число 5 во внутреннем представлении. На рисунке 26 — число -5.

00000000:0040207f	05 00 fb ff 00	add eax, 0xffffb00
-------------------	----------------	--------------------

Рисунок 25 — 5 во внутреннем представлении отладчика

00000000:00402081	fb	sti
00000000:00402082	ff 00	inc dword [rax]

Рисунок 26 — -5 во внутреннем представлении отладчика

Загрузим эти числа в регистр AX и проверим содержимое (см. рисунки 27 и 28 соответственно)

RAX 0000000000000005

Рисунок 27 — содержимое регистра AX после записи в него числа 5

RAX 00000000fffffffb

Рисунок 28 — содержимое регистра AX после записи в него числа -5

6. Каким образом в ассемблере программируются выражения?

Составьте

фрагмент программы для вычисления $C=A+B$, где A, B и C – целые числа формата BYTE.

Код программы на рисунке 29.

```
task6.asm x lab1.asm x sum.asm x
1  section .data
2      A db 97
3      B db 23
4  section .bss
5      C resb 10;
6  section .text
7
8  global _start
9  _start:
10     ;calc C
11     mov esi, [A]
12     add esi, [B]
13     mov [C], esi
14
15     ; exit
16     mov eax, 1 ; системная функция 1 (exit)
17     xor ebx, ebx ; код возврата 0
18     int 80h ; вызов системной функции
```

Рисунок 29 — Код программы нахождения суммы

Найдем внутреннее представление переменных А, В, С в отладчике.

Переменная А показана на рисунке 30— 61 в шестнадцатеричной это 97 в десятичной. Переменная В показана на рисунке 31 — 17 в шестнадцатеричной это 23 в десятичной.



Рисунок 30 — переменная А



Рисунок 31 — переменная В

Рассмотрим подробнее операции. Сперва содержимое переменной А помещается в регистр ESI, результат чего изображен на рисунке 32.

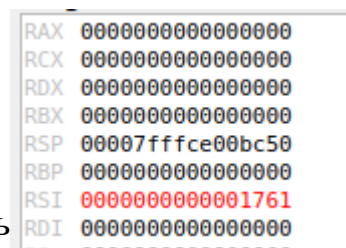


Рисунок 32 — запись

содержимого переменной А в регистр ESI

Вторая операция добавляет содержимое переменной В в регистр ESI. Число 23, содержащееся в переменной В, в шестнадцатеричной СС равняется 17. Следовательно, к значению регистра ESI добавилось 17, что наблюдается на рисунке 33.

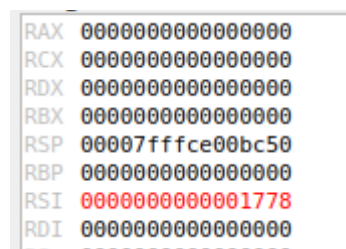


Рисунок 33 — добавление содержимого числа В в регистр ESI

После этого, содержимое регистра ESI помещается в переменную C. Найдем переменную C и посмотрим внутреннее представление отладчика (см. рисунок 33)



Рисунок 34 — содержимое переменной C

Так как размер переменных программы sum.asm всего лишь 1 байт, то нас интересует 1 младший байт, который на рисунке 32 находится слева. 78 переводим в десятичную систему и получаем 120, что является правильным результатом сложения чисел 97 и 23.

Вывод: ознакомилась со средой разработки SASM, с отладчиком edb, научилась пользоваться отладчиком, находить данные как по переменным, так и по адресу. Кроме того, научилась писать простейшие программы на языке ассемблера.