

# **Лабораторная работа №9**

**Архитектура компьютера**

Казначеева Кристина Никитична

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>3</b>
<b>2</b>	<b>Задание</b>	<b>4</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
<b>4</b>	<b>Вывод</b>	<b>23</b>

# 1 Цель работы

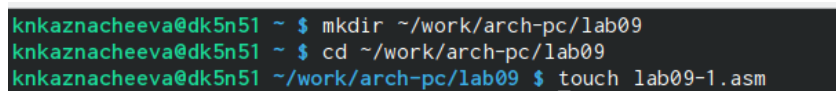
Лабораторная работа посвящена практическому освоению программирования с подпрограммами и отладке кода с помощью отладчика GDB.

## 2 Задание

В ходе лабораторной работы мы освоим программирование на NASM с использованием подпрограмм, отладку программ с помощью GDB (включая установку точек останова и работу с данными программы), а также обработку аргументов командной строки. В качестве примеров будут рассмотрены вывод сообщения “Hello, world!” и вычисление заданного выражения.

### 3 Выполнение лабораторной работы

Создадим новый каталог, перейдём в него и создадим файл lab09-1.asm:(рис. 3.1).



```
knkaznacheeva@dk5n51 ~ $ mkdir ~/work/arch-pc/lab09
knkaznacheeva@dk5n51 ~ $ cd ~/work/arch-pc/lab09
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ touch lab09-1.asm
```

Рис. 3.1: Создание каталога и файла

Введём в файл lab9-1.asm текст программы с использованием вызова подпрограммы (рис. 3.2).

```

lab09-1.asm [-M--] 11 L: [ 1+14 15/ 3
#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
1По~щъ 2Со~ан 3Блок 4Замена 5Копия 6Пе~т

```

Рис. 3.2: Ввод текста программы

Создадим исполняемый файл и проверим его работу (рис. 3.3).

```

knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1
.o
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 3
2x+7=13

```

Рис. 3.3: Проверка работы исходного файла

Затем заменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$  (рис. 3.4).

```

lab09-1.asm      [----]  0 L:[ 34+19  53/ 53] *(1208/1208)
_calcul:
push eax ; Сохраняем x на стеке перед вызовом _subcalcul
call _subcalcul
add esp, 4 ; Удаляем x со стека
mov ebx, eax ; Результат g(x) в ebx
mov eax, 2
mul ebx ; 2 * g(x)
add eax, 7 ; + 7
mov [res], eax
ret

; -----
; Подпрограмма вычисления g(x) = 3x - 1
; -----
_subcalcul:
mov ebx, 3
mul ebx ; 3 * x
sub eax, 1 ; - 1
ret

```

1 Поиск 2 Созд-н 3 Блок 4 Замена 5 Копия 6 Лить 7 Поиск 8 Уда-

Рис. 3.4: Изменение текста программы

Запустим исполняемый файл и проверим его (рис. 3.5).

```

knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 3
f(g(x)) = 23

```

Рис. 3.5: Проверка работы исходного файла

Создадим файл lab09-2.asm (рис. 3.6).

```

knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ touch lab09-2.asm

```

Рис. 3.6: Создание файла

Введём в файл lab9-2.asm текст программы вывода сообщения Hello world (рис. 3.7).

```

lab09-2.asm      [-M--]
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0

```

Рис. 3.7: Ввод текста программы

Для работы с GDB в исполняемый файл добавим отладочную информацию, для этого проведём трансляцию программ необходимо с ключом '-g' (рис. 3.8):



```
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
```

Рис. 3.8: Трансляция программы

Загрузим исполняемый файл в отладчик gdb (рис. 3.9).

```
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/g
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
```

Рис. 3.9: Загрузка исполняемый файл в отладчик gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 3.10).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/n/knkaznacheeva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 8663) exited normally]
(gdb)
```

Рис. 3.10: Проверка работы программы с помощью команды run

Для более подробного анализа программы установим брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы (рис. 3.11).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb)
```

Рис. 3.11: Проверка работы исходного файла

Затем запустим программу (рис. 3.12).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/n/knkaznacheeva/work/arch-pc/la
b09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) █
```

Рис. 3.12: Запуск программы

Посмотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. 3.13).

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
      0x08049005 <+5>:      mov     $0x1,%ebx
      0x0804900a <+10>:     mov     $0x804a000,%ecx
      0x0804900f <+15>:     mov     $0x8,%edx
      0x08049014 <+20>:     int     $0x80
      0x08049016 <+22>:     mov     $0x4,%eax
      0x0804901b <+27>:     mov     $0x1,%ebx
      0x08049020 <+32>:     mov     $0x804a008,%ecx
      0x08049025 <+37>:     mov     $0x7,%edx
      0x0804902a <+42>:     int     $0x80
      0x0804902c <+44>:     mov     $0x1,%eax
      0x08049031 <+49>:     mov     $0x0,%ebx
      0x08049036 <+54>:     int     $0x80
End of assembler dump.
(gdb) █
```

Рис. 3.13: Команда `disassemble`

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 3.14).

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     mov     eax,0x4
    0x0804901b <+27>:     mov     ebx,0x1
    0x08049020 <+32>:     mov     ecx,0x804a008
    0x08049025 <+37>:     mov     edx,0x7
    0x0804902a <+42>:     int     0x80
    0x0804902c <+44>:     mov     eax,0x1
    0x08049031 <+49>:     mov     ebx,0x0
    0x08049036 <+54>:     int     0x80
End of assembler dump.

```

Рис. 3.14: Переключение на отображение команд с Intel'овским синтаксисом

Включим режим псевдографики для более удобного анализа программы (рис. 3.15).

```

[ Register Values Unavailable ]

B+>0x8049000 <_start>      mov     eax,0x4
    0x8049005 <_start+5>    mov     ebx,0x1
    0x804900a <_start+10>   mov     ecx,0x804a000
    0x804900f <_start+15>   mov     edx,0x8
    0x8049014 <_start+20>   int     0x80
    0x8049016 <_start+22>   mov     eax,0x4

native process 8775 In: _start
(gdb) layout regs
(gdb)

```

Рис. 3.15: Изменение текста программы

Основные различия в отображении синтаксиса машинных команд в режимах

АТТ (AT&T) и Intel заключаются в следующем:

- Регистры:
  - \* Intel: Имена регистров пишутся без префикса (например, `eax`, `ebx`).
  - \* АТТ: Имена регистров предваряются знаком `%` (например, `%eax`, `%ebx`).
- Операторы размера:
  - \* Intel: Размер операндов обычно указывается суффиксом (например, `mov ax, bx` для 16-битных регистров, `mov eax, ebx` для 32-битных).
  - \* АТТ: Размер операндов обычно указывается префиксом перед именем инструкции (например, `movw %bx, %ax` для 16-битных, `movl %ebx, %eax` для 32-битных).
- Также используются префиксы `b` (байт), `w` (слово), `l` (длинное слово - 32 бита), `q` (четверное слово - 64 бита).
- Константы:
  - \* Intel: Константы записываются непосредственно (например, `mov eax, 10`).
  - \* АТТ: Константы предваряются символом `$` (например, `movl $10, %eax`).
- Метки:
  - \* Intel: Метки используются непосредственно (например, `jmp mylabel`).
  - \* АТТ: Метки предваряются знаком `.` (например, `.mylabel`).

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints` (рис. 3.16).

```
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
```

Рис. 3.16: Команда `info breakpoints`

Установим ещё одну точку останова по адресу инструкции (рис. 3.17).

```
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) □
```

Рис. 3.17: Установление точки останова по адресу инструкции

Посмотрим информацию о всех установленных точках останова (рис. 3.18).

```
(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
2        breakpoint      keep y   0x08049031 lab09-2.asm:20
(gdb) □
```

Рис. 3.18: Команда `info breakpoints`

Выполним 5 инструкций с помощью команды `si` и проследим за изменением значений регистров (рис. 3.19).

```
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) □
```

Рис. 3.19: Команда `si`

Значения регистров `eax`, `ebx`, и `ecx` изменяются (рис. 3.20).

Register group: general		
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffc3c0	0xffffc3c0
ebp	0x0	0x0

0x804900a	<_start+10>	mov	ecx, 0x804a000
0x804900f	<_start+15>	mov	edx, 0x8
0x8049014	<_start+20>	int	0x80
>0x8049016	<_start+22>	mov	eax, 0x4
0x804901b	<_start+27>	mov	ebx, 0x1
0x8049020	<_start+32>	mov	ecx, 0x804a008

Рис. 3.20: Команда `si`

Посмотрим содержимое регистров также можно с помощью команды `info registers` (рис. 3.21).

```

native process 8775 In: _start L14 PC: 0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb)

```

Рис. 3.21: Команда info registers

Посмотрим значение переменной msg1 по имени (рис. 3.22).

```

(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb)

```

Рис. 3.22: Просмотр значения переменной msg1 по имени

Посмотрим значение переменной msg2 (рис. 3.23).

```

(gdb) x/1sb &msg2
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 3.23: Просмотр значения переменной msg2 по имени

Изменим первый символ переменной msg1 (рис. 3.24).

```

(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb)

```

Рис. 3.24: Изменение символа переменной msg1

Заменяем любой символ во второй переменной msg2 (рис. 3.25).

```
(gdb) set {char}&msg2='W'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "World!\n\034"
, .., □
```

Рис. 3.25: Изменение символа переменной msg2

Чтобы посмотреть значения регистров воспользуемся командой print /F (рис. 3.26).

```
(gdb) p/t $eax
$2 = 1000

(gdb) p/s $eax
$3 = 8

(gdb) p/x $ecx
$4 = 0x804a000

(gdb) p/s $ecx
$5 = 134520832
, .., □
```

Рис. 3.26: Команда print

С помощью команды set изменим значение регистра ebx (рис. 3.27),(рис. 3.28).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
^_ _ _ _ _ □
```

Рис. 3.27: Команда set

```
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
^_ _ _ _ _ □
```

Рис. 3.28: Команда set

Разница в выводе команды `p/s $ebx` обусловлена тем, как GDB интерпретирует значение, присвоенное регистру `$ebx`. • `set $ebx='2'`: Здесь регистру `$ebx` присваивается строковое значение '2'. Символ ' указывает на строку. • `set $ebx=2`: Здесь регистру `$ebx` присваивается числовое значение 2. Отсутствие кавычек указывает на число. Поэтому `p/s $ebx` отображает числовое значение 2. В сущности, в первом случае мы работаем со строкой, а во втором — с целым числом.

Завершим выполнение программы с помощью команды `si` и выйдем из GDB с помощью команды `quit` (рис. 3.29).



```
(gdb) si
(gdb) quit
A debugging session is active.

        Inferior 1 [process 8775] will be killed.

Quit anyway? (y or n) ☐
```

Рис. 3.29: Завершение выполнения программы и выход из GDB

Скопируем файл lab8-2.asm с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm:(рис. 3.30).

```
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Рис. 3.30: Копирование файла lab8-2.asm с именем lab09-3.asm

Создадим исполняемый файл (рис. 3.31).

```
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 3.31: Создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузим исполняемый файл в отладчик, указав аргументы (рис. 3.32).

```
knkaznacheeva@dk5n51 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент2
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
```

Рис. 3.32: Загрузка файла в отладчик

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. 3.33).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/n/knkaznacheeva/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в 'ecx' количество
```

Рис. 3.33: Создание исполняемого файла

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’(рис. 3.34).

```
(gdb) x/x $esp
0xfffffc370:      0x00000005
```

Рис. 3.34: Команда x/x \$esp

Посмотрим остальные позиции стека – по адресу [esp+4] располагается адрес в памяти, где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. Смещение на 4 байта происходит потому, что в 64-битной архитектуре указатели (в данном случае, указатели на строки аргументов) занимают 8 байт, и стек выравнивается по 8 байтам (для повышения производительности). Это делает адреса аргументов выровненными по 8 байтам, а смещение между ними равно 8 байтам (рис. 3.35).

```

(gdb) x/x $esp
0xfffffc370:      0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc5cf:      "/afs/.dk.sci.pfu.edu.ru/home/k/n/knkaznacheeva/work/arch-pc/lab
09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc619:      "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc62b:      "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc63c:      "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc63e:      "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <error: Cannot access memory at address 0x0>

```

Рис. 3.35: Просмотр остальных позиций стека

Создадим файл lab09-4.asm (рис. 3.36).

```

knkaznacheeva@dk2n21 ~/work/arch-pc/lab09 $ touch lab09-4.asm

```

Рис. 3.36: Создание файла

Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)=7(x+1)$  как подпрограмм (рис. 3.37).

```

lab09-4.asm      [----] 20 L:[ 1+ 3 4/ 59
%include 'in_out.asm'

SECTION .data
prim DB 'f(x)=7(x+1)',0
otv DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

%include 'in_out.asm'

SECTION .data
prim DB 'f(x)=7(x+1)',0
otv DB 'Результат: ',0

```

Рис. 3.37: Ввод текста программы

Проверим программу исходного файла(рис. 3.38).

```

knkaznacheeva@dk2n21 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
knkaznacheeva@dk2n21 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
knkaznacheeva@dk2n21 ~/work/arch-pc/lab09 $ ./lab09-4 4
f(x)=7(x+1)
Результат: 35

```

Рис. 3.38: Запуск программы

Создадим файл lab09-5.asm (рис. 3.39).

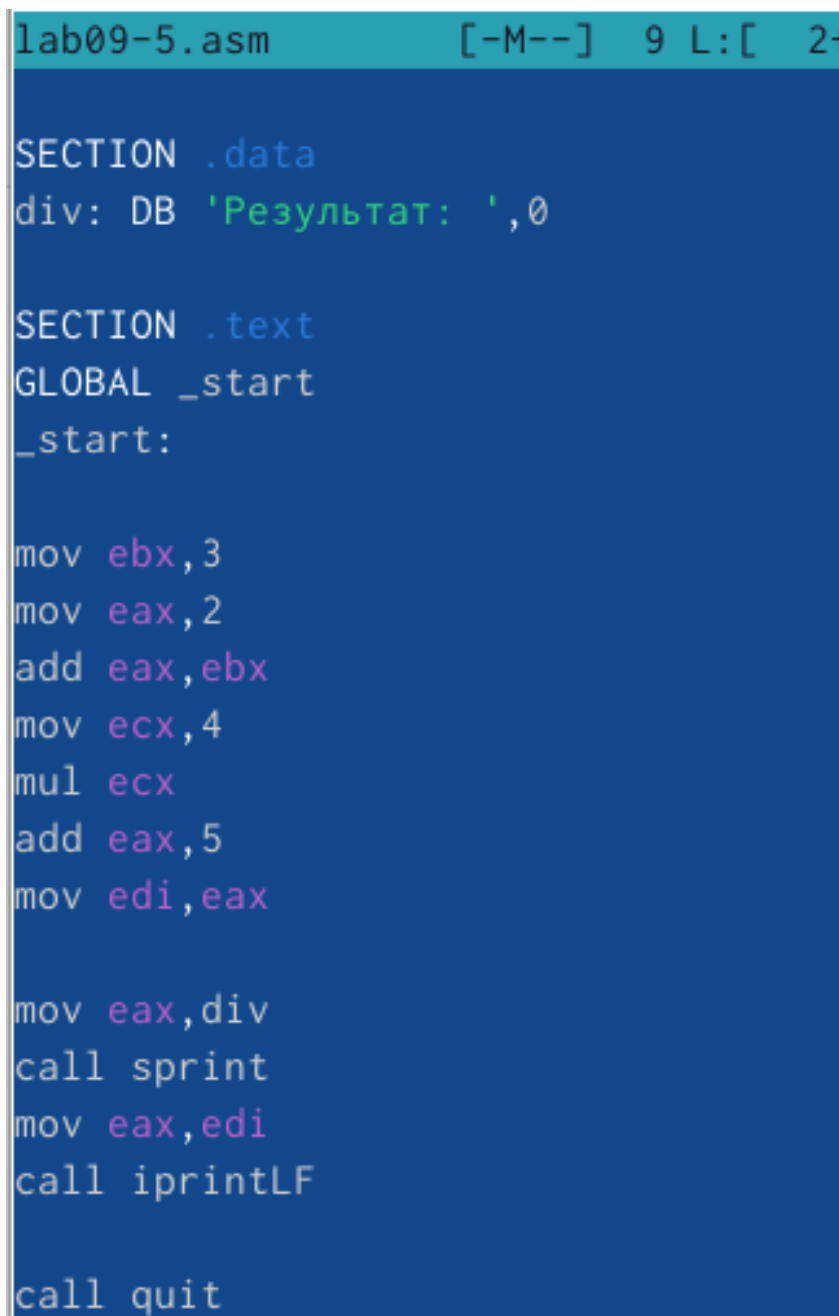
```

knkaznacheeva@dk2n21 ~/work/arch-pc/lab09 $ touch lab09-5.asm

```

Рис. 3.39: Создание файла

Программа вычисления выражения  $(3 + 2) \cdot 4 + 5$  при запуске дает неверный результат. Проверим это. С помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправим ее (рис. 3.40).



```
lab09-5.asm      [-M--]  9 L:[ 2-

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax

mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit
```

Рис. 3.40: Ввод текста программы

Проверим программу исходного файла (рис. 3.41).

```
knkaznacheeva@dk2n21 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
knkaznacheeva@dk2n21 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
knkaznacheeva@dk2n21 ~/work/arch-pc/lab09 $ ./lab09-5 4
Результат: 25
```

Рис. 3.41: Запуск программы

## 4 Вывод

В рамках лабораторной работы были изучены и отработаны навыки программирования на языке ассемблера NASM, включающие создание и использование подпрограмм, отладку программ с помощью отладчика GDB (с применением точек останова и анализа содержимого памяти), а также обработку аргументов командной строки. В качестве иллюстративных примеров были использованы вывод сообщения “Hello, world!” и вычисление математического выражения.