

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра фундаментальной информатики и информационных технологий

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №9

Тема: Понятие подпрограммы. Отладчик GDB.

Дисциплина: Архитектура компьютеров

Студент: Герчет Вячеслав

Группа: НКАбд-03-25

Студ. билет № 1132255650

Преподаватель: Штепа Кристина Александровна

МОСКВА

2025 г.

Содержание

1. Цель работы
2. Задание
3. Теоретическое введение
4. Выполнение лабораторной работы
 - 4.1 Реализация подпрограмм в NASM
 - 4.2 Отладка программ с помощью GDB
5. Выполнение самостоятельной работы
6. Выводы
7. Список литературы

Список иллюстраций

- 3.1 Создаем файл с помощью команды touch
- 3.2 Заполняем файл
- 3.3 Запускаем файл и проверяем его работу
- 3.4 Изменяем файл, добавляя еще одну подпрограмму
- 3.5 Запускаем файл и смотрим на его работу
- 3.6 Создаем файл
- 3.7 Заполняем файл
- 3.8 Загружаем исходный файл в отладчик
- 3.9 Запускаем программу командой run
- 3.10 Запускаем программу с брейкпоинтом
- 3.11 Смотрим дисассимилированный код программы
- 3.12 Переключаемся на синтаксис Intel
- 3.13 Включаем отображение регистров, их значений и результат дисас-симилирования программы
- 3.14 Используем команду info breakpoints и создаем новую точку останова
- 3.15 Смотрим информацию
- 3.16 Отслеживаем регистры
- 3.17 Смотрим значение переменной
- 3.18 Смотрим значение переменной
- 3.19 Меняем символ
- 3.20 Меняем символ
- 3.21 Смотрим значение регистра
- 3.22 Изменяем регистор командой set
- 3.23 Прописываем команды c и quit
- 3.24 Копируем файл
- 3.25 Создаем и запускаем в отладчике файл
- 3.26 Устанавливаем точку останова

- 3.27 Изучаем полученные данные
- 3.28 Копируем файл
- 3.29 Изменяем файл
- 3.30 Проверяем работу программы
- 3.31 Создаем файл
- 3.32 Изменяем файл
- 3.33 Создаем и смотрим на работу программы(работает неправильно)
- 3.34 Ищем ошибку регистров в отладчике
- 3.35 Меняем файл
- 3.36 Создаем и запускаем файл(работает корректно)

1. Цель работы

Целью лабораторной работы является приобретение навыков написания программ с использованием подпрограмм, а также знакомство с методами отладки программ и основными возможностями отладчика GDB.

2. Задание

Написать программы с использованием подпрограмм и выполнить их отладку с помощью GDB.

3. Теоретическое введение

Отладка программы - это процесс поиска, анализа и исправления ошибок в программе. Выделяют синтаксические, семантические и ошибки выполнения. Основное назначение отладки - определить причину неправильной работы программы и устраниить её.

Для отладки широко используется отладчик GDB, который позволяет запускать программу в пошаговом режиме, устанавливать точки останова, просматривать и изменять значения регистров и памяти, а также анализировать дизассемблированный код программы.

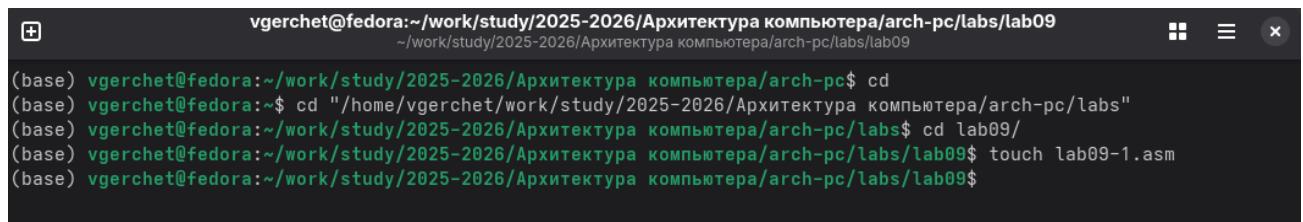
Подпрограмма - это отдельный функциональный участок кода, который может вызываться из разных мест программы. Вызов подпрограммы осуществляется инструкцией call, которая

сохраняет адрес возврата в стеке, а возврат выполняется инструкцией ret. Использование подпрограмм делает программу более компактной и удобной для сопровождения.

4. Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаем файл (рис. 3.1).



```
vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09
~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09

(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc$ cd
(base) vgerchet@fedora:~$ cd "/home/vgerchet/work/study/2025-2026/Архитектура компьютера/arch-pc/labs"
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs$ cd lab09/
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ touch lab09-1.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.1: Создаем файл с помощью команды touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. 3.2).

```
GNU nano 8.3                               lab09-1.asm                                Изменён
%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:
    mov eax, msg
    call sprint

    mov ecx, x
    mov edx, 80
    call sread

    mov eax, x
    call atoi

    call _calcul

    mov eax, result
    call sprint

    mov eax, [res]
    call iprintLF

    call quit

._calcul:
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret
```

Рис. 3.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. 3.3).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab09-1.asm
ld -m elf_i386 -o lab09-1 lab09-1.o
./lab09-1
Введите x: 5
2x+7=17
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ █
```

Рис. 3.3: Запускаем файл и смотрим на его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию)(рис. 3.4)

```
_calcul:  
    call _subcalcul  
    mov ebx, 2  
    mul ebx  
    add eax, 7  
    mov [res], eax  
    ret  
  
_subcalcul:  
    mov ebx, 3  
    mul ebx  
    sub eax, 1  
    ret
```

Рис. 3.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. 3.5).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab09-1.asm  
ld -m elf_i386 -o lab09-1 lab09-1.o  
.lab09-1  
Введите x: 5  
2(3x-1)+7=35  
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.5: Запускаем файл и смотрим на его работу

2.2 Отладка программам с помощью GDB

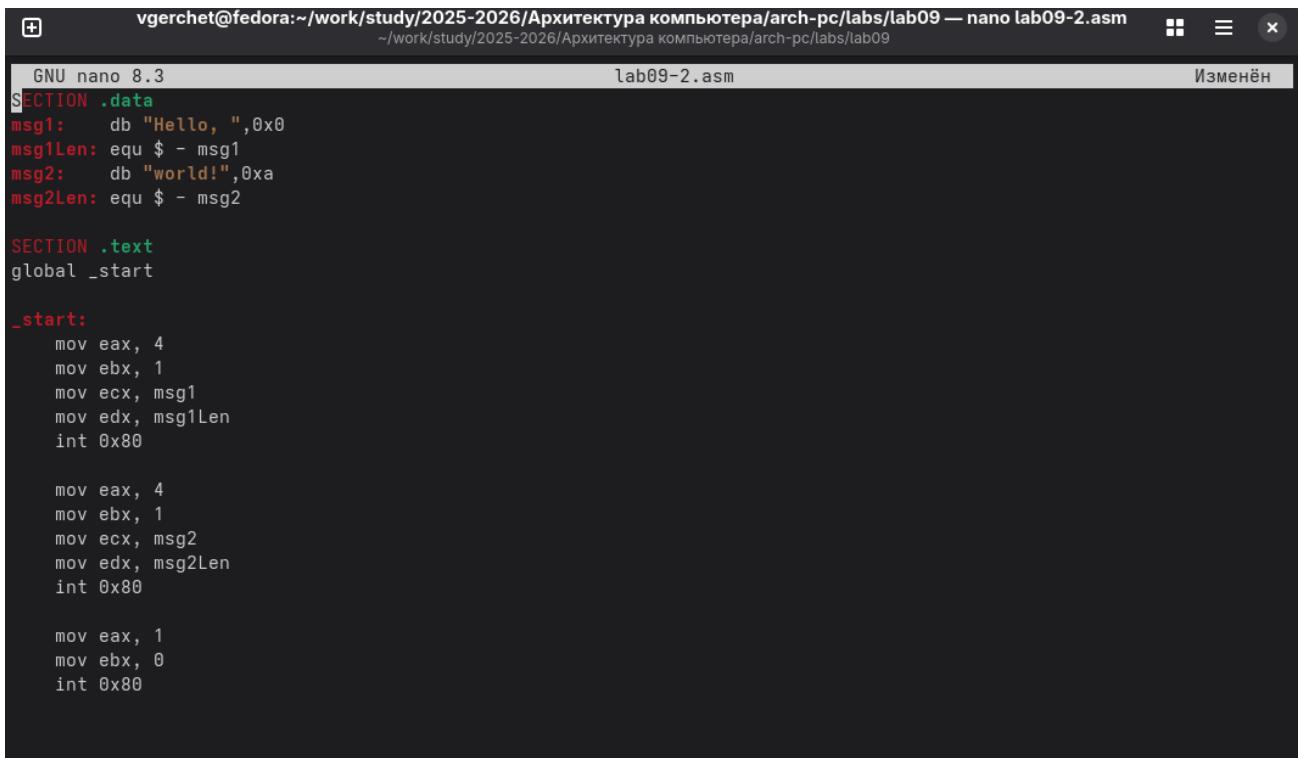
Создаем новый файл в каталоге (рис. 3.6).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ touch lab09-2.asm  
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с

листингом 9.2 (рис. 3.7).



GNU nano 8.3
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ \$ - msg1
msg2: db "world!",0xa
msg2Len: equ \$ - msg2

SECTION .text
global _start

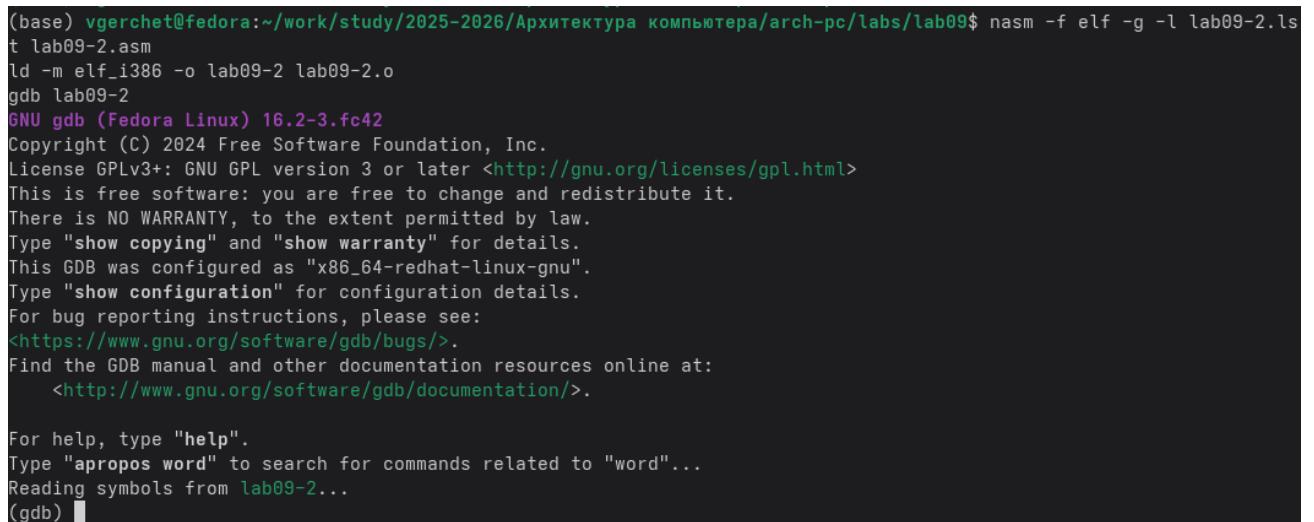
.start:
 mov eax, 4
 mov ebx, 1
 mov ecx, msg1
 mov edx, msg1Len
 int 0x80

 mov eax, 4
 mov ebx, 1
 mov ecx, msg2
 mov edx, msg2Len
 int 0x80

 mov eax, 1
 mov ebx, 0
 int 0x80

Рис. 3.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb(рис. 3.8).



```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf -g -l lab09-2.ls
t lab09-2.asm
ld -m elf_i386 -o lab09-2 lab09-2.o
gdb lab09-2
GNU gdb (Fedora Linux) 16.2-3.fc42
Copyright (C) 2024 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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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...
(gdb) 
```

Рис. 3.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. 3.9).

```
(gdb) run
Starting program: /home/vgerchet/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading 51.96 K separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5523) exited normally]
(gdb)
```

Рис. 3.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку _start и запускаем программу (рис.

3.10).

```
(gdb) break _start
Breakpoint 1 at 0x8048080: file lab09-2.asm, line 11.
(gdb) run
Starting program: /home/vgerchet/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
(gdb)
```

Рис. 3.10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды

disassemble, начиная с метки _start (рис. 3.11).

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:    mov    $0x4,%eax
  0x08048085 <+5>:    mov    $0x1,%ebx
  0x0804808a <+10>:   mov    $0x8049000,%ecx
  0x0804808f <+15>:   mov    $0x8,%edx
  0x08048094 <+20>:   int    $0x80
  0x08048096 <+22>:   mov    $0x4,%eax
  0x0804809b <+27>:   mov    $0x1,%ebx
  0x080480a0 <+32>:   mov    $0x8049008,%ecx
  0x080480a5 <+37>:   mov    $0x7,%edx
  0x080480aa <+42>:   int    $0x80
  0x080480ac <+44>:   mov    $0x1,%eax
  0x080480b1 <+49>:   mov    $0x0,%ebx
  0x080480b6 <+54>:   int    $0x80
End of assembler dump.
(gdb) █
```

Рис. 3.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. 3.12).

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:    mov    eax,0x4
  0x08048085 <+5>:    mov    ebx,0x1
  0x0804808a <+10>:   mov    ecx,0x8049000
  0x0804808f <+15>:   mov    edx,0x8
  0x08048094 <+20>:   int    0x80
  0x08048096 <+22>:   mov    eax,0x4
  0x0804809b <+27>:   mov    ebx,0x1
  0x080480a0 <+32>:   mov    ecx,0x8049008
  0x080480a5 <+37>:   mov    edx,0x7
  0x080480aa <+42>:   int    0x80
  0x080480ac <+44>:   mov    eax,0x1
  0x080480b1 <+49>:   mov    ebx,0x0
  0x080480b6 <+54>:   int    0x80
End of assembler dump.
(gdb) █
```

Рис. 3.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1.Порядок operandов: В ATT синтаксисе порядок operandов обратный, сначала указывается исходный operand, а затем - результирующий operand. В Intel синтаксисе порядок обычно прямой, результирующий operand указывается первым, а исходный - вторым.

2.Разделители: В ATT синтаксисе разделители operandов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3. Префиксы размера операндов: В ATT синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как "b" (byte), "w" (word), "l" (long) и "q" (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как "b", "w", "d" и "q".

4. Знак операндов: В ATT синтаксисе операнды с позитивными значениями предваряются символом '".**000000**".

5. Обозначение адресов: В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6. Обозначение регистров: В ATT синтаксисе обозначение регистра начинается с символа "%". В Intel синтаксисе обозначение регистра может начинаться с символа "R" или "E" (например, "%eax" или "RAX").

Включаем режим псевдографики (рис. 3.13).

```

vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09 — gdb lab09-2
~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09

Register group: general
eax      0x0          0           ecx      0x0          0
edx      0x0          0           ebx      0x0          0
esp     0xfffffc40    0xfffffc40   ebp      0x0          0x0
esi      0x0          0           edi      0x0          0
eip     0x8048080    0x8048080 <_start> eflags   0x202        [ IF ]
cs       0x23         35          ss       0x2b         43
ds       0x2b         43          es       0x2b         43
fs       0x0          0           gs       0x0          0

0x80480fe    add    BYTE PTR [eax],al
0x8048100    add    BYTE PTR [eax],al
0x8048102    add    BYTE PTR [eax],al
0x8048104    add    BYTE PTR [eax],al
0x8048106    add    BYTE PTR [eax],al
0x8048108    add    BYTE PTR [eax],al
0x804810a    add    BYTE PTR [eax],al
0x804810c    add    BYTE PTR [eax],al
0x804810e    add    BYTE PTR [eax],al
0x8048110    add    BYTE PTR [eax],al
0x8048112    add    BYTE PTR [eax],al
0x8048114    add    BYTE PTR [eax],al
0x8048116    add    BYTE PTR [eax],al
0x8048118    add    BYTE PTR [eax],al

native process 5547 (asm) In: _start
(gdb) layout regs
(gdb)

```

Рис. 3.13: Включаем отображение регистров, их значений и результат дисасси-милирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции(рис. 3.14).

```

(gdb) layout asm
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:    mov    eax,0x4
  0x08048085 <+5>:    mov    ebx,0x1
  0x0804808a <+10>:   mov    ecx,0x8049000
  0x0804808f <+15>:   mov    edx,0x8
  0x08048094 <+20>:   int    0x80
  0x08048096 <+22>:   mov    eax,0x4
  0x0804809b <+27>:   mov    ebx,0x1
  0x080480a0 <+32>:   mov    ecx,0x8049008
  0x080480a5 <+37>:   mov    edx,0x7
  0x080480aa <+42>:   int    0x80
  0x080480ac <+44>:   mov    eax,0x1
  0x080480b1 <+49>:   mov    ebx,0x0
  0x080480b6 <+54>:   int    0x80
End of assembler dump.
(gdb) Quit
(gdb) break 0x08048094
Function "0x08048094" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 2 (0x08048094) pending.
(gdb)

```

Рис. 3.14:Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова(рис. 3.15).

```
(gdb) info breakpoints
Num      Type            Disp Enb Address      What
1        breakpoint      keep y  0x08048080  lab09-2.asm:11
          breakpoint already hit 1 time
2        breakpoint      keep y  <PENDING>  0x08048094
(gdb)
```

Рис. 3.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. 3.16).

The screenshot shows a terminal window with a blue header bar. The title bar reads "vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09 — gdb lab09-2". The main area displays assembly code for the '_start' function:

```
B+ 0x08048080 <_start>    mov    eax,0x4
0x08048085 <_start+5>    mov    ebx,0x1
0x0804808a <_start+10>   mov    ecx,0x8049000
0x0804808f <_start+15>   mov    edx,0x8
0x08048094 <_start+20>   int    0x80
>0x08048096 <_start+22>  mov    eax,0x4
0x0804809b <_start+27>  mov    ebx,0x1
0x080480a0 <_start+32>  mov    ecx,0x8049008
0x080480a5 <_start+37>  mov    edx,0x7
0x080480aa <_start+42>  int    0x80
0x080480ac <_start+44>  mov    eax,0x1
0x080480b1 <_start+49>  mov    ebx,0x0
0x080480b6 <_start+54>  int    0x80
0x080480b8      add    BYTE PTR [eax],al
0x080480ba      add    BYTE PTR [eax],al
0x080480bc      add    BYTE PTR [eax],al
0x080480be      add    BYTE PTR [eax],al
0x080480c0      add    BYTE PTR [eax],al
0x080480c2      add    BYTE PTR [eax],al
0x080480c4      add    BYTE PTR [eax],al
0x080480c6      add    BYTE PTR [eax],al
0x080480c8      add    BYTE PTR [eax],al
0x080480ca      add    BYTE PTR [eax],al
0x080480cc      add    BYTE PTR [eax],al
0x080480ce      add    BYTE PTR [eax],al
0x080480d0      add    BYTE PTR [eax],al
0x080480d2      add    BYTE PTR [eax],al
```

Below the assembly code, the terminal shows the native process state and a GDB session:

```
native process 7271 (asm) In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type            Disp Enb Address      What
1        breakpoint      keep y  0x08048080  lab0
          breakpoint already hit 1 time
(gdb) si 5Hello,
(gdb)
```

Рис. 3.16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip.

Смотрим значение переменной msg1 по имени (рис. 3.17).

```
(gdb) x/1sb &msg1
0x8049000 <msg1>:      "Hello, "
(gdb)
```

Рис. 3.17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. 3.18).

```
(gdb) x/1sb &msg1
0x8049000 <msg1>:      "Hello, "
(gdb) x/1sb &msg2
0x8049008 <msg2>:      "world!\n\034"
(gdb)
```

Рис. 3.18: Смотрим значение переменной

Изменим первый символ переменной msg1(рис. 3.19).

```
(gdb) set {char}&msg1 = 'h'
(gdb) x/1sb &msg1
0x8049000 <msg1>:      "hello, "
(gdb)
```

Рис. 3.19: Меняем символ

Изменим первый символ переменной msg2 (рис. 3.20).

```
(gdb) set {char}&msg2 = 'W'
(gdb) x/1sb &msg2
0x8049008 <msg2>:      "World!\n\034"
(gdb)
```

Рис. 3.20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. 3.21).

```
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb)
```

Рис. 3.21: Смотрим значение регистра

Изменяем регистор ebx (рис. 3.22).

```
(gdb) set $ebx = '2'  
(gdb) p/s $ebx  
$4 = 50  
(gdb) set $ebx = 2  
(gdb) p/s $ebx  
$5 = 2  
(gdb)
```

Рис. 3.22: Изменяем регистор командой set

Выводится разные значения, так как команда без кеавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. 3.23).

```
(gdb) layout asm  
layout regs  
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.23: Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. 3.24).

```
cp ./lab08/lab8-2.asm lab09-3.asm  
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. 3.25).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf -g -l lab09-3.l  
st lab09-3.asm  
ld -m elf_i386 -o lab09-3 lab09-3.o  
gdb --args ./lab09-3 arg1 arg2 "arg 3"  
GNU gdb (Fedora Linux) 16.2-3.fc42  
Copyright (C) 2024 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-redhat-linux-gnu".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
<https://www.gnu.org/software/gdb/bugs/>.  
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...
(gdb)
```

Рис. 3.25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. 3.26).

```
(gdb) break _start
run
Function "_start"
run" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (_start
run) pending.
(gdb)
```

Рис. 3.26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. 3.27).

```
Starting program: /home/vgerchet/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09/lab09-3 arg1 arg2 arg
\ 3

Breakpoint 2, _start () at lab09-3.asm:7
7      pop ecx          ; количество аргументов
(gdb) x/x $esp
0xfffffc20: 0x00000004
(gdb) x/s *(void**)$esp + 4
x/s *(void**)$esp + 8
x/s *(void**)$esp + 12
x/s *(void**)$esp + 16
A syntax error in expression, near `x/s *(void**)$esp + 8'
x/s *(void**)$esp + 12
x/s *(void**)$esp + 16'.
(gdb) x/s *(void**)$esp + 16
0xfffffceaa: "arg 3"
(gdb)
```

Рис. 3.27: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

5. Выполнение самостоятельной работы

Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. 3.28).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ cd "/home/vgerchet/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09"
cp ..//lab08/Lab8-4.asm lab09-4.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ █
```

Рис. 3.28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму

(рис. 3.29).

The screenshot shows a terminal window with the command `vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09 — nano lab09-4.asm`. The file content is as follows:

```
GNU nano 8.3
%include 'in_out.asm'

SECTION .data
msg_f    db "Функция: f(x)=15x+2",0
msg_res  db "Результат: ",0

SECTION .text
global _start

_start:
    pop ecx
    pop edx
    sub ecx, 1

    mov esi, 0

next:
    cmp ecx, 0
    jz _end

    pop eax
    call atoi

    call _f

    add esi, eax
    loop next

_end:
    mov eax, msg_f
    call sprintLF

    mov eax, msg_res
    call sprint

    mov eax, esi
    call iprintLF

    call quit

_f:
    imul eax, 15
```

At the bottom of the terminal window, there is a menu bar with Russian labels and keyboard shortcuts:

- `^G Справка`
- `^O Записать`
- `^F Поиск`
- `^K Вырезать`
- `^T Выполнить`
- `^C Позиция`
- `M-U Отмена`
- `^X Выход`
- `^R ЧитФайл`
- `^V Замена`
- `^U Вставить`
- `^J Выровнять`
- `^/ К строке`
- `M-E Повтор`

Рис. 3.29: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. 3.30).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab09-4.asm
ld -m elf_i386 -o lab09-4 lab09-4.o
./lab09-4 1 2 3
Функция: f(x)=15x+2
Результат: 96
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.30: Проверяем работу программы

Задание 2

Создаем новый файл в директории (рис. 3.31).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ touch lab09-5.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.31: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. 3.32).



```
GNU nano 8.3
%include 'in_out.asm'

SECTION .data
div: db 'Результат: ',0

SECTION .text
global _start

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

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

Рис. 3.32: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. 3.33).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab09-5.asm
ld -m elf_i386 -o lab09-5 lab09-5.o
./lab09-5
Результат: 10
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.33: Создаем и смотрим на работу программы(работает неправильно)

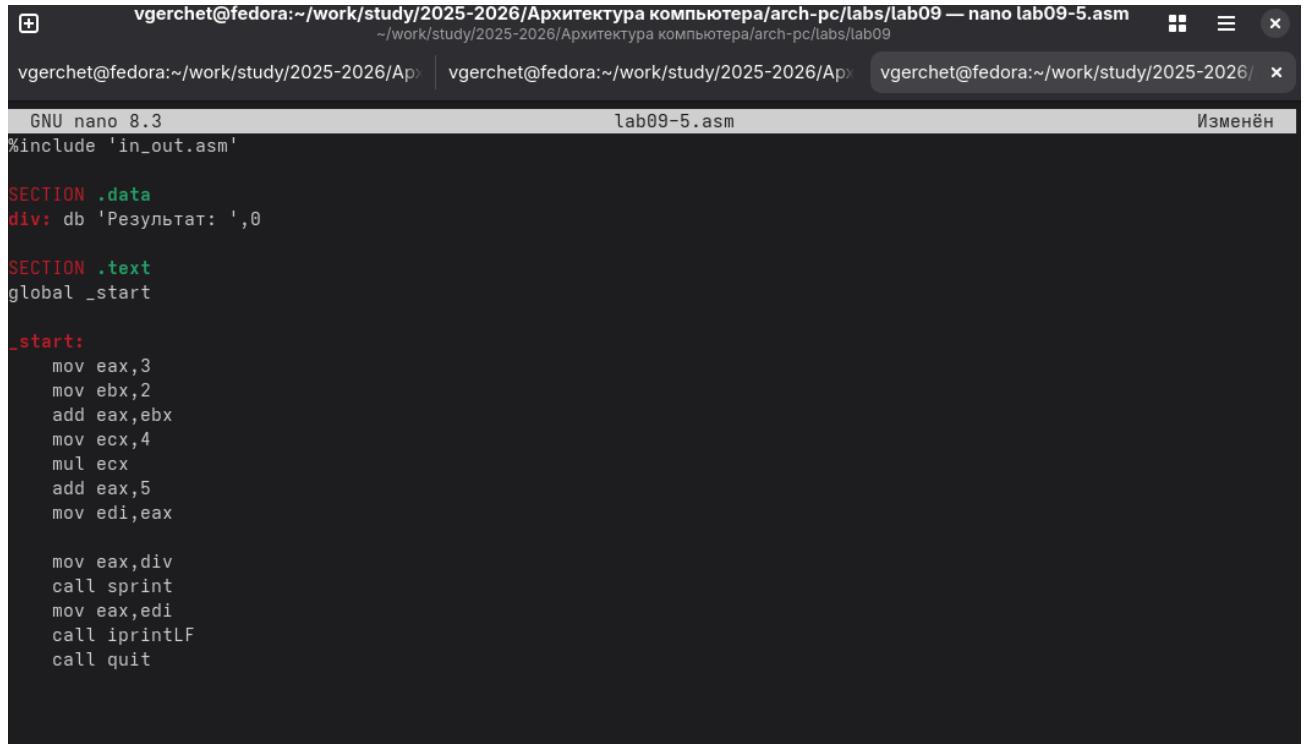
Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si (рис. 3.34).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ gdb lab09-5
GNU gdb (Fedora Linux) 16.2-3.fc42
Copyright (C) 2024 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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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 lab09-5...
(No debugging symbols found in lab09-5)
(gdb) break _start
run
layout asm
layout regs
si 5
Function "_start"
run
layout asm
layout regs
si 5" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (_start
run
layout asm
layout regs
si 5) pending.
(gdb)
```

Рис. 3.34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. 3.35).



```
vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09 — nano lab09-5.asm
vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09 — nano lab09-5.asm
GNU nano 8.3
lab09-5.asm
Изменён

%include 'in_out.asm'

SECTION .data
div: db 'Результат: ',0

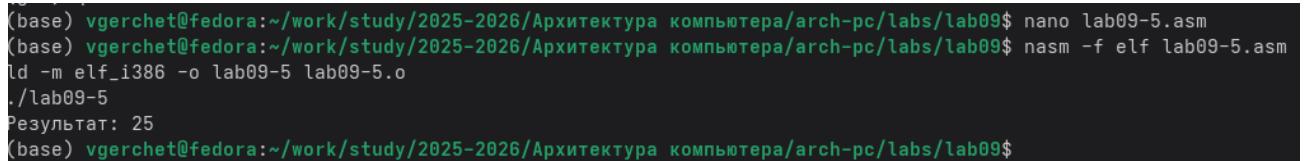
SECTION .text
global _start

_start:
    mov eax,3
    mov ebx,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. 3.35: Меняю файл

Создаем исполняемый файл и запускаем его (рис. 3.36).



```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ nano lab09-5.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab09-5.asm
ld -m elf_i386 -o lab09-5 lab09-5.o
./lab09-5
Результат: 25
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab09$
```

Рис. 3.36: Создаем и запускаем файл(работает корректно)

4. Выводы

В ходе выполнения лабораторной работы №9 я научился использовать подпрограммы в программах на языке ассемблера NASM и понял принцип работы инструкций call и ret. Также я освоил базовые приёмы работы с отладчиком GDB: запуск программы, установку точек останова, пошаговое выполнение, просмотр и изменение значений регистров и памяти.

Дополнительно я получил практический опыт поиска и исправления ошибок с помощью отладчика, а также научился анализировать выполнение программы на уровне машинных инструкций. В результате выполненной работы я стал лучше понимать процесс выполнения программ и методы их отладки на низком уровне.

5. Список литературы

NASM Documentation. <https://www.nasm.us/docs.html>

Демидова А. В. Архитектура ЭВМ. Лабораторная работа №9

GDB: The GNU Project Debugger. <https://sourceware.org/gdb>