

# **Лабораторная работа №10. Понятие подпрограммы. Отладчик GDB.**

**НММ-6д-02-22**

Крухмалев Артём Владиславович

# 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>Самостоятельная работа</b>	<b>13</b>
<b>5</b>	<b>Выводы</b>	<b>17</b>

# **1 Цель работы**

Научиться работать с отладчиком.

## 2 Задание

НС помощью отладчика исправить ошибку в коде.

### 3 Выполнение лабораторной работы

1. С помощью терминала создадим подкаталог, создадим файл lab9-1.asm

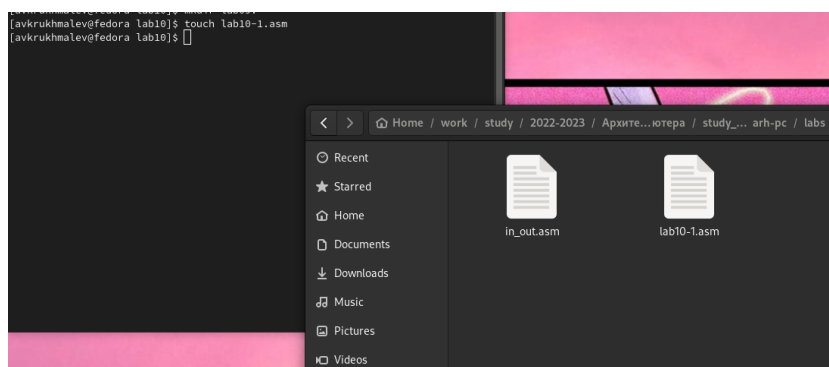


Рис. 3.1: Новый каталог

2. Изучим и запишем в него код из листинга, откомпилируем и запустим файл.

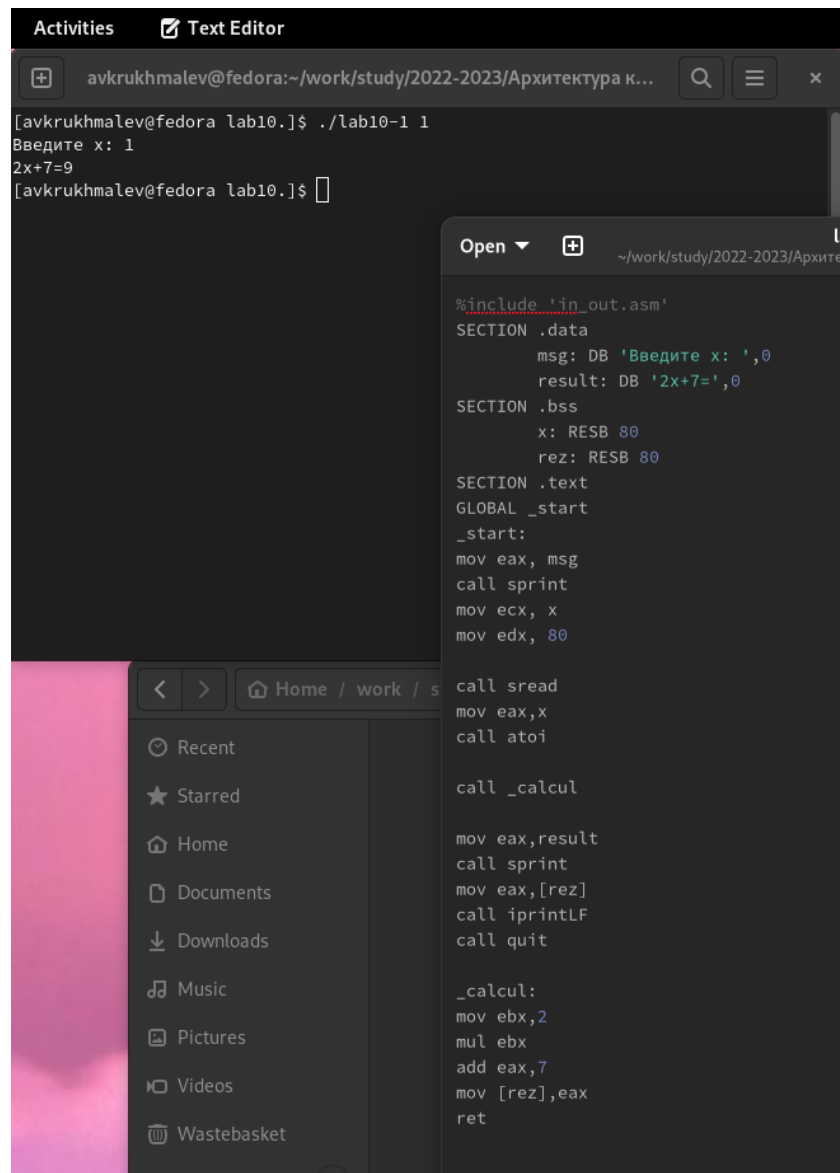


Рис. 3.2: Листинг 1

3. Добавим в подпрограмму ещё одну подпрограмму

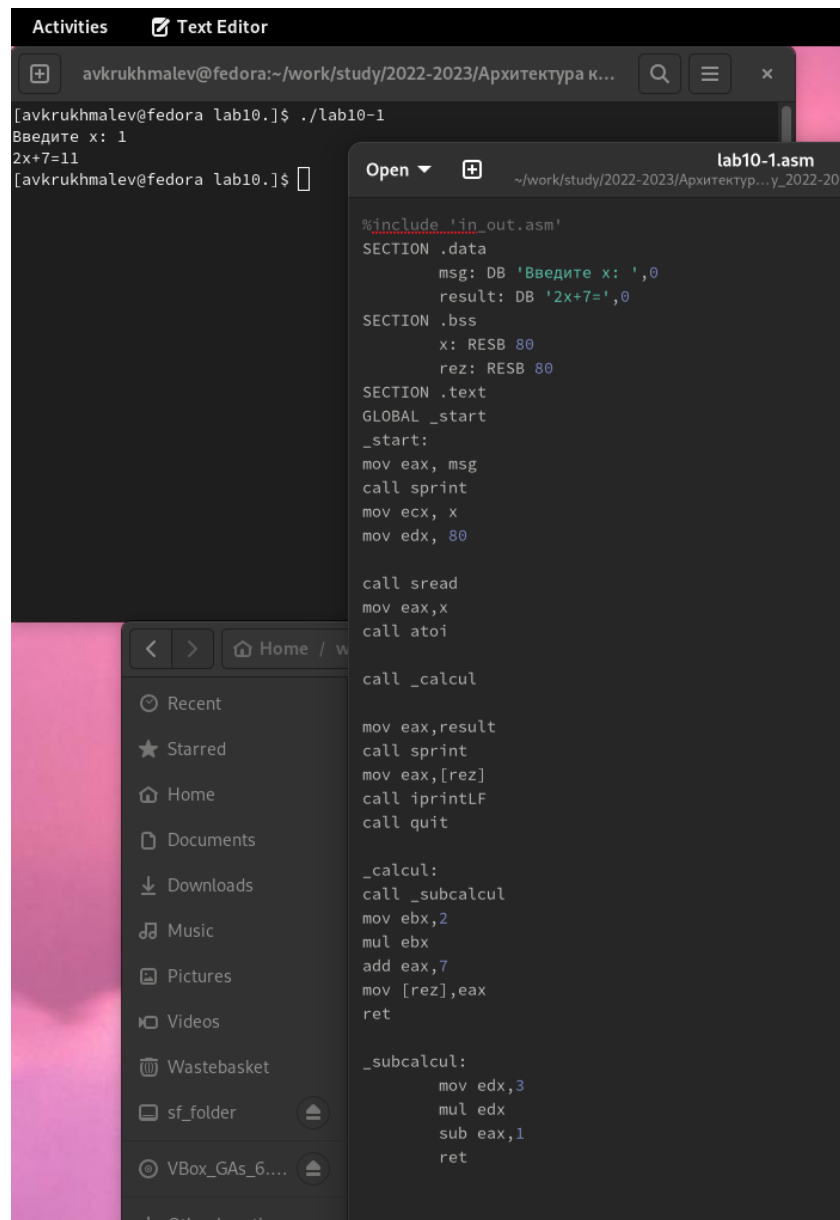


Рис. 3.3: Ввод 2-ой подпрограммы

4. Создадим новый файл, запишем в него предложенный код, запустим отладчик и в нем запустим программу





6. Рассмотрим отличия между синтаксисами. Как видно на скриншоте, ячейки памяти находятся с разных сторон от значений в них и в АТТ добавляются символы \$ и %.

```
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) 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.
(gdb)
```

Рис. 3.7: Различие синтаксисов

7. Выведем режимы псевдографики, по началу layout asm будет пустой.

```

0x8049092      add     BYTE PTR [eax],al
0x8049094      add     BYTE PTR [eax],al
0x8049096      add     BYTE PTR [eax],al
0x8049098      add     BYTE PTR [eax],al
0x804909a      add     BYTE PTR [eax],al
0x804909c      add     BYTE PTR [eax],al
0x804909e      add     BYTE PTR [eax],al
0x80490a0      add     BYTE PTR [eax],al
0x80490a2      add     BYTE PTR [eax],al
0x80490a4      add     BYTE PTR [eax],al
0x80490a6      add     BYTE PTR [eax],al
0x80490a8      add     BYTE PTR [eax],al
0x80490aa      add     BYTE PTR [eax],al
0x80490ac      add     BYTE PTR [eax],al
0x80490ae      add     BYTE PTR [eax],al
0x80490b0      add     BYTE PTR [eax],al
0x80490b2      add     BYTE PTR [eax],al
0x80490b4      add     BYTE PTR [eax],al
0x80490b6      add     BYTE PTR [eax],al
0x80490b8      add     BYTE PTR [eax],al
0x80490ba      add     BYTE PTR [eax],al
0x80490bc      add     BYTE PTR [eax],al
0x80490be      add     BYTE PTR [eax],al
0x80490c0      add     BYTE PTR [eax],al

```

native process 2941 In: \_start  
(gdb) layout regs  
(gdb)

Рис. 3.8: Псевдографика

## 8. Добавим точки останова

```

(gdb) layout regs
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 <_start>
        breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 <_start>
        breakpoint already hit 1 time
2        breakpoint    keep y  0x08049031 <_start+49>
(gdb)

```

Рис. 3.9: Точки останова

## 9. С помощью команды i r посмотрим содержимое регистров, проделав операцию 5 раз заметим, что значения регистров не меняются

```

(gdb) info registers
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd0f0      0xffffd0f0
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x8049000        0x8049000 <_start>
eflags         0x202          [ IF ]
cs             0x23            35
ss             0x2b            43
ds             0x2b            43
es             0x2b            43
fs             0x0            0
gs             0x0            0
(gdb) x/1sb &msg1
0x804a000:      "Hello, "
(gdb)

```

Рис. 3.10: Значение регистров

10. Теперь поменяем значение в 1 регистре на другое.

```

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

```

Рис. 3.11: Изменение значения с помощью отладчика

11. Воспользуемся другой функцией (set) и поменяем значение.

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb)

```

Рис. 3.12: Set

12. Запустим программу из 9 лабораторной, установим брейкпоинт и изучим, что лежит в стеке. Шаг равен 4, потому что в 1 ячейке стека 4 байта информации.

```

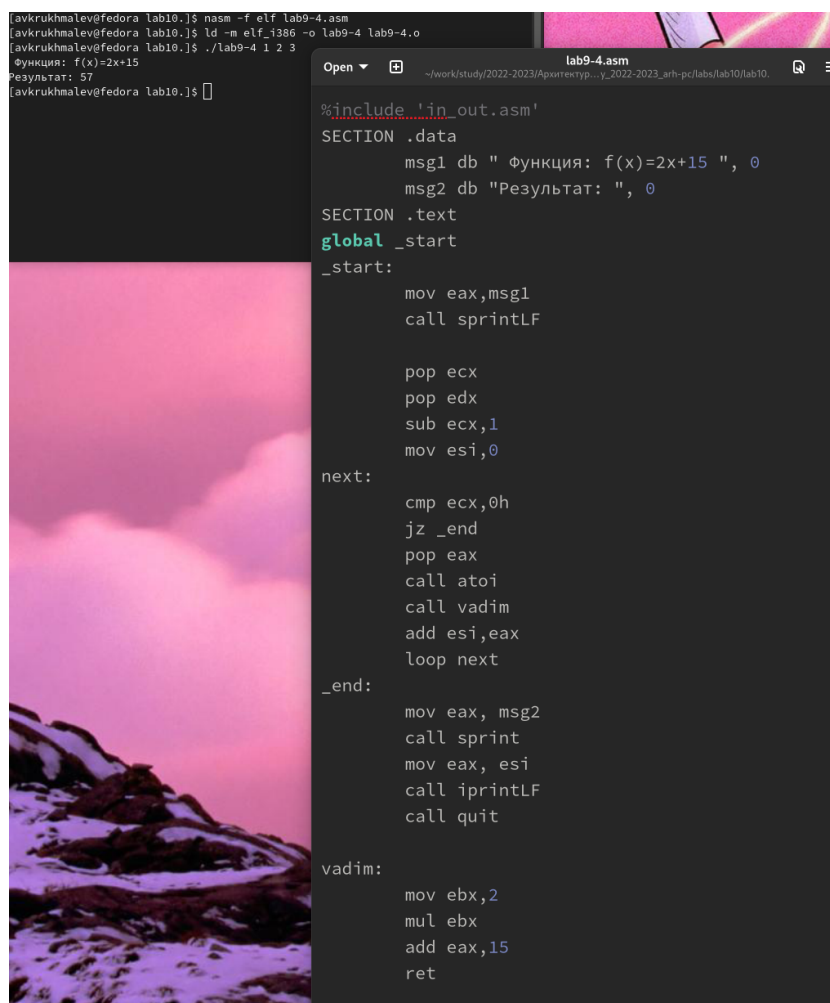
Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd0b0: 0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd264: "/home/avkrukhmalev/work/study/2022-2023/Архитектура компьютера/
study_2022-2023_arh-pc/labs/lab10/lab10./lab10-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd2e9: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd2fb: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd30c: "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd30e: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.13: Стэк

## 4 Самостоятельная работа

### 1. Скопируем файл и изменим код



The image shows a terminal window on the left and an assembly editor on the right. The terminal displays the following commands and output:

```
lavkrukhmalev@fedora lab10.js$ nasm -f elf lab9-4.asm
lavkrukhmalev@fedora lab10.js$ ld -m elf_i386 -o lab9-4 lab9-4.o
lavkrukhmalev@fedora lab10.js$ ./lab9-4 1 2 3
Функция: f(x)=2x+15
Результат: 57
lavkrukhmalev@fedora lab10.js$
```

The assembly editor on the right shows the following code:

```
%include 'in_out.asm'
SECTION .data
    msg1 db " Функция: f(x)=2x+15 ", 0
    msg2 db "Результат: ", 0
SECTION .text
global _start
_start:
    mov eax,msg1
    call sprintLF

    pop ecx
    pop edx
    sub ecx,1
    mov esi,0
next:
    cmp ecx,0h
    jz _end
    pop eax
    call atoi
    call vadim
    add esi,eax
    loop next
_end:
    mov eax, msg2
    call sprint
    mov eax, esi
    call iprintLF
    call quit
vadim:
    mov ebx,2
    mul ebx
    add eax,15
    ret
```

Рис. 4.1: Самостоятельная работа номер 1

### 2. Предложенный код выводит ошибку, с помощью gdb и функций X/NFU посмотрим содержание регистра умножения, ещё надо поставить на нем

брейкпоинт, заметим, что в нем изменяется `eax`, а суммируем мы с `ebx` и выводим значение в `ebx`, поэтому заменим в суммирование `ebx` на `eax` и получим правильный ответ 20.

```
[avkrukhmalev@fedora lab10.]$ nasm -f elf lab10-4.asm
[avkrukhmalev@fedora lab10.]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[avkrukhmalev@fedora lab10.]$ ./lab10-4
Результат: 10
[avkrukhmalev@fedora lab10.]$
```

Рис. 4.2: Первоначальный вывод

```

--Register group: general--
eax      0x0                                0                                ecx
esp      0xffffd0f0                        0xffffd0f0                      ebp
eip      0x80490e8                        0x80490e8 <_start>             eflags
ds       0x2b                             43                                es

0x80490d9 <atoi.restore+3>    pop    %ebx
0x80490da <atoi.restore+4>    ret
0x80490db <quit>                mov     $0x0,%ebx
0x80490e0 <quit+5>              mov     $0x1,%eax
0x80490e5 <quit+10>             int     $0x80
0x80490e7 <quit+12>             ret
B+> 0x80490e8 <_start>          mov     $0x3,%ebx
0x80490ed <_start+5>            mov     $0x2,%eax
0x80490f2 <_start+10>           add     %eax,%ebx
0x80490f4 <_start+12>           mov     $0x4,%ecx
b+> 0x80490f9 <_start+17>        mul     %ecx
0x80490fb <_start+19>           add     $0x5,%eax
0x80490fe <_start+22>           mov     %eax,%edi
0x8049100 <_start+24>           mov     $0x804a000,%eax
0x8049105 <_start+29>           call    0x804900f <sprint>
0x804910a <_start+34>           mov     %edi,%eax
0x804910c <_start+36>           call    0x8049086 <iprintLF>
0x8049111 <_start+41>           call    0x80490db <quit>
0x8049116                       add     %al,(%eax)
0x8049118                       add     %al,(%eax)
0x804911a                       add     %al,(%eax)
0x804911c                       add     %al,(%eax)
0x804911e                       add     %al,(%eax)
0x8049120                       add     %al,(%eax)

native process 2650 In: _start
breakpoint keep y 0x80490e8 <_start>
breakpoint already hit 1 time
(gdb) break *0x80490f9
breakpoint 2 at 0x80490f9
(gdb) i b
Type      Disp Enb Address      What
breakpoint keep y 0x80490e8 <_start>
breakpoint already hit 1 time
breakpoint keep y 0x80490f9 <_start+17>
(gdb) info registers
eax      0x0                                0
ecx      0x0                                0
edx      0x0                                0
ebx      0x0                                0
esp      0xffffd0f0                        0xffffd0f0
ebp      0x0                                0x0
esi      0x0                                0
edi      0x0                                0
eip      0x80490e8                        0x80490e8 <_start>
eflags   0x202                            [ IF ]
cs       0x23                             35
ss       0x2b                             43
ds       0x2b                             43
fs       0x2b                             43
fs       0x0                                0
fs       0x0                                0
(gdb)

```

Рис. 4.3: Просмотр регистров

```
[avkrukhmalev@fedora lab10.]$ nasm -f elf lab10-4.asm
[avkrukhmalev@fedora lab10.]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[avkrukhmalev@fedora lab10.]$ ./lab10-4
Результат: 25
[avkrukhmalev@fedora lab10.]$
```

Open ▾

lab10-4

~/work/study/2022-2023/Архитектур...

report.md

```
%include 'in_out.asm'
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. 4.4: Исправленный код



## **5 Выводы**

В данной работе мы познакомились с отладчиком и с помощью него научились изменять программу.