

# **Отчёт по лабораторной работе № 10**

**Архитектура компьютера**

Старцева Алина Сергеевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
3.1	Реализация подпрограмм в NASM . . . . .	7
3.2	Отладка программ с помощью GDB . . . . .	9
3.3	Добавление точек останова . . . . .	12
3.4	Работа с данными программы в GDB . . . . .	13
3.5	Обработка аргументов командной строки в GDB . . . . .	17
3.6	Задание для самостоятельной работы . . . . .	18
<b>4</b>	<b>Выводы</b>	<b>26</b>

# Список иллюстраций

3.1	7
3.2	8
3.3	8
3.4	9
3.5	9
3.6	9
3.7	10
3.8	10
3.9	11
3.10	11
3.11	11
3.12	12
3.13	12
3.14	13
3.15	13
3.16	13
3.17	14
3.18	14
3.19	14
3.20	15
3.21	15
3.22	15
3.23	15
3.24	16
3.25	16
3.26	16
3.27	16
3.28	17
3.29	17
3.30	17
3.31	17
3.32	18
3.33	18
3.34	18
3.35	19
3.36	19
3.37	19

3.38 . . . . .	20
3.39 . . . . .	20
3.40 . . . . .	20
3.41 . . . . .	20
3.42 . . . . .	21
3.43 . . . . .	21
3.44 . . . . .	21
3.45 . . . . .	21
3.46 . . . . .	22
3.47 . . . . .	22
3.48 . . . . .	22
3.49 . . . . .	23
3.50 . . . . .	23
3.51 . . . . .	24
3.52 . . . . .	24
3.53 . . . . .	25
3.54 . . . . .	25

# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

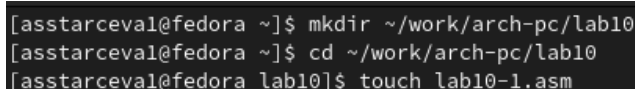
## 2 Задание

1. Реализовать подпрограммы в NASM.
2. Выполнить отладку программ с помощью GDB.
3. Отработать добавление точек останова.
4. Поработа с данными программы в GDB.
5. Отработать обработку аргументов командной строки в GDB.
6. Выполнить задание для самостоятельной работы.

## 3 Выполнение лабораторной работы

### 3.1 Реализация подпрограмм в NASM

1. Создали каталог для выполнения лабораторной работы № 10, перешли в него и создали файл lab10-1.asm: (рис. 3.1)

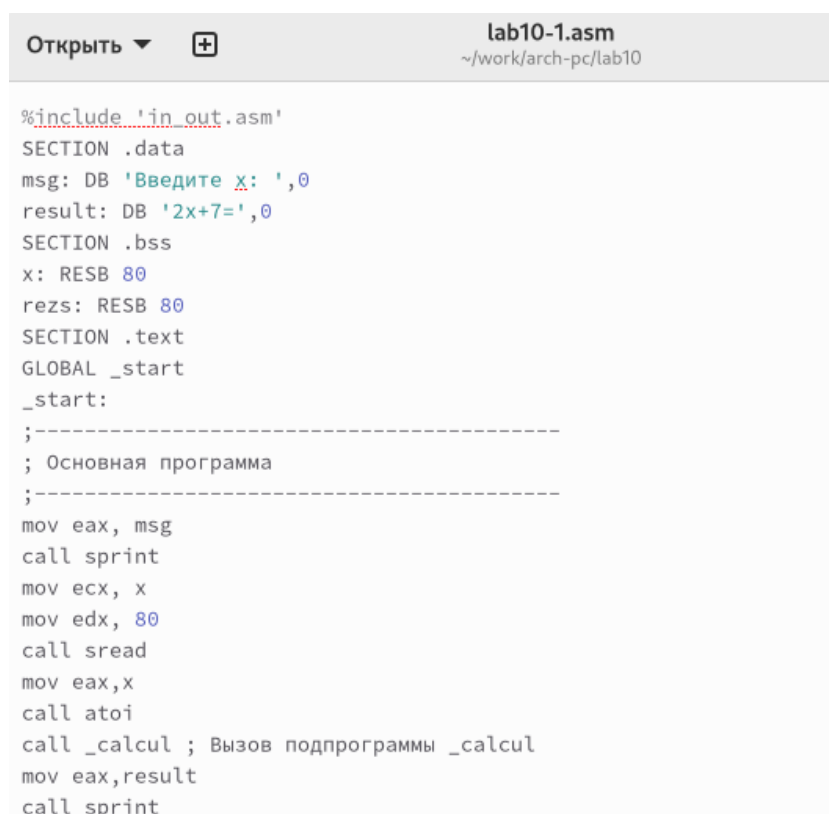


```
[asstarceval@fedora ~]$ mkdir ~/work/arch-pc/lab10  
[asstarceval@fedora ~]$ cd ~/work/arch-pc/lab10  
[asstarceval@fedora lab10]$ touch lab10-1.asm
```

Рис. 3.1: .

2. В качестве примера рассмотрели программу вычисления арифметического выражения  $f(x) = 2x + 7$  с помощью подпрограммы `_calcul`. В данном примере `x` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучили текст программы (Листинг 10.1).

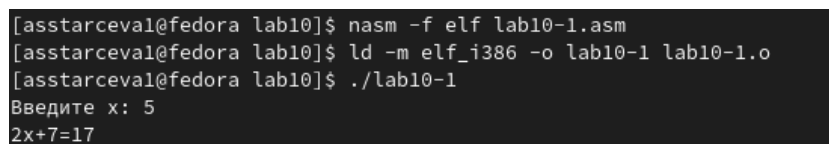
Введите в файл lab10-1.asm текст программы из листинга 10.1. (рис. 3.2) Создайте исполняемый файл и проверьте его работу. (рис. 3.3)



```
Открыть ▾ + lab10-1.asm
~/work/arch-pc/lab10

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rezs: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
```

Рис. 3.2: .

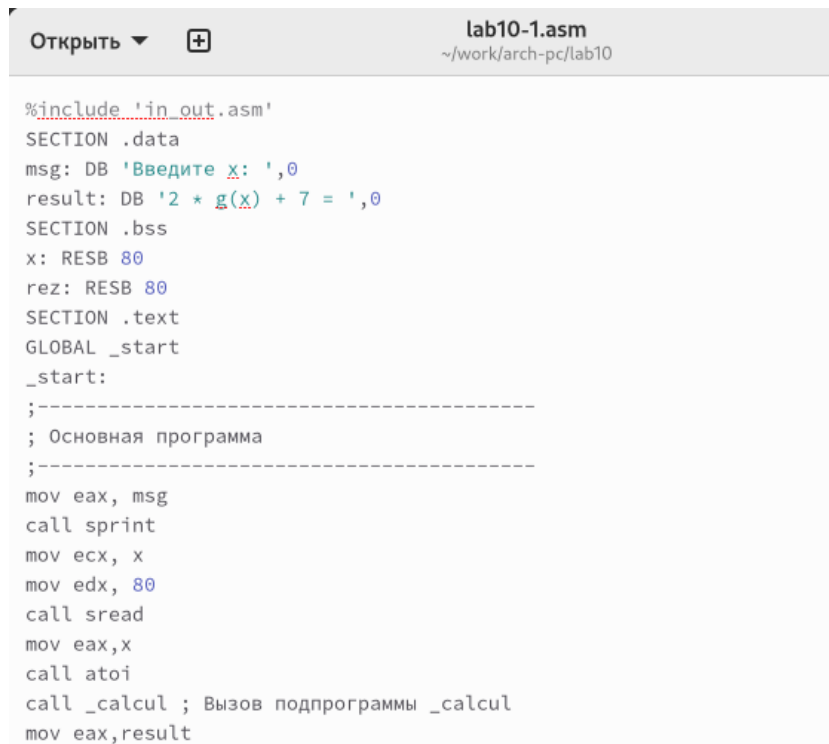


```
[asstarceval@fedora lab10]$ nasm -f elf lab10-1.asm
[asstarceval@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[asstarceval@fedora lab10]$ ./lab10-1
Введите x: 5
2x+7=17
```

Рис. 3.3: .

Изменили текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . Т.е.  $x$  передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение  $g(x)$ , результат возвращается в `_calcul` и вычисляется выражение  $f(g(x))$ . Результат возвращается в основную программу для вывода результата на экран. (рис. 3.4), (рис. 3.5)

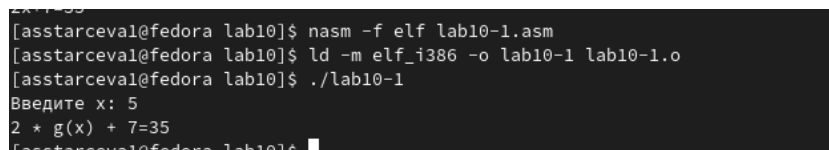




```
Открыть ▾ + lab10-1.asm
~/work/arch-pc/lab10

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2 * g(x) + 7 = ',0
SECTION .bss
x: RESB 80
rez: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
```

Рис. 3.4: .

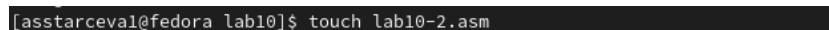


```
[asstarceval@fedora lab10]$ nasm -f elf lab10-1.asm
[asstarceval@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[asstarceval@fedora lab10]$ ./lab10-1
Введите x: 5
2 * g(x) + 7=35
[asstarceval@fedora lab10]$
```

Рис. 3.5: .

## 3.2 Отладка программ с помощью GDB

Создали файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!): (рис. 3.6), (рис. 3.7)



```
[asstarceval@fedora lab10]$ touch lab10-2.asm
```

Рис. 3.6: .

```
• lab10-2.asm
~/work/arch-pc/lab10

Открыть ▼ +

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: .

Получили исполняемый файл. Для работы с GDB в исполняемый файл добавили отладочную информацию, для этого трансляцию программ провели с ключом '-g'. Загрузили исполняемый файл в отладчик gdb. (рис. 3.8)

```
[asstarceval@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[asstarceval@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[asstarceval@fedora lab10]$ gdb lab10-2
```

Рис. 3.8: .

Загрузили исполняемый файл в отладчик gdb. Проверили работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r): (рис. 3.9)

```
(gdb) r
Starting program: /home/asstarceval/work/arch-pc/lab10/lab10-2

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 separate debug info for /home/asstarceval/work/arch-pc/lab10/system-
supplied DSO at 0xf7ffc000...
Hello, world!
[Inferior 1 (process 3854) exited normally]
```

Рис. 3.9: .

Для более подробного анализа программы установили брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустили её.(рис. 3.10)

```
(gdb) r
Starting program: /home/asstarceval/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
```

Рис. 3.10: .

Посмотрели дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`. (рис. 3.11)

```
(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.
```

Рис. 3.11: .

Переключились на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. (рис. 3.12)

```

(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. 3.12: .

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel: в АТТ перед адресом регистра ставится \$, а перед названием регистра %, сначала записывается адрес, а потом - регистр. В Intel сначала регистр, а потом адрес, и перед ними ничего не ставится.

Включили режим псевдографики для более удобного анализа программы.(рис. 3.13)

```

[ 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
    0x804901b <_start+27> mov     ebx,0x1

native process 3868 In: _start          L9  PC: 0x8049000
(gdb) layout regs
(gdb) █

```

Рис. 3.13: .

### 3.3 Добавление точек останова

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки

программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверили это с помощью команды `info breakpoints` (кратко `i b`). (рис. 3.14)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y   0x08049000 lab10-2.asm:9
      breakpoint already hit 1 time
```

Рис. 3.14: .

Установили еще одну точку останова по адресу инструкции. Адрес инструкции увидели в средней части экрана в левом столбце соответствующей инструкции. Определили адрес предпоследней инструкции (`mov ebx,0x0`) и установили точку останова. (рис. 3.15)

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab10-2.asm, line 20.
```

Рис. 3.15: .

Посмотрели информацию о всех установленных точках останова: (рис. 3.16)

```
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y   0x08049000 lab10-2.asm:9
2     breakpoint       keep y   0x08049031 lab10-2.asm:20
```

Рис. 3.16: .

## 3.4 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполнили 5 инструкций с помощью команды `stepi` (или `si`) и проследили за изменением значений регистров. (рис. 3.17), (рис. 3.18)

eax	0x0	0
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xffffd1f0	0xffffd1f0
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

Рис. 3.17: .

```
(gdb) stepi
10      mov ebx, 1
(gdb) stepi
11      mov ecx, msg1
(gdb) stepi
12      mov edx, msg1Len
(gdb) stepi
13      int 0x80
(gdb) stepi
Hello, 14      mov eax, 4
```

Рис. 3.18: .

Изменяются значения регистров: eax, ecx, edx, ebx.

Посмотрели содержимое регистров с помощью команды info registers (или i r).  
(рис. 3.19)

eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd1f0	0xffffd1f0
ebp	0x0	0x0
esi	0x0	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

Рис. 3.19: .

Для отображения содержимого памяти можно использовать команду x, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором

выводятся данные, можно задать после имени команды через косую черту: x/NFU . С помощью команды x & также можно посмотреть содержимое переменной. Посмотрели значение переменной msg1 по имени. (рис. 3.20)

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

Рис. 3.20: .

Посмотрели значение переменной msg2 по адресу. Адрес переменной определили по дизассемблированной инструкции. Посмотрели инструкцию mov esx,msg2 которая записывает в регистр esx адрес переменной msg2. (рис. 3.21)

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

Рис. 3.21: .

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс \$, а перед адресом нужно указать в фигурных скобках тип данных. Изменили первый символ переменной msg1. (рис. 3.22)

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

Рис. 3.22: .

Замените первый символ во второй переменной msg2. (рис. 3.23)

```
(gdb) set {char}0x804a008='g'
(gdb) x /1sb &msg2
0x804a008 <msg2>: "gorld!\n\034"
```

Рис. 3.23: .

Чтобы посмотреть значения регистров используется команда print /F . Вывели в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. 3.24)

```
(gdb) p/s $edx
$1 = 8
(gdb) p/x $edx
$2 = 0x8
(gdb) p/t $edx
$3 = 1000
```

Рис. 3.24: .

С помощью команды set измените значение регистра ebx: (рис. 3.25)

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

Рис. 3.25: .

Вывод команд p/s \$ebx различается, так как сначала в регистре ebx хранится строковое значение, а потом целочисленное.

Завершили выполнение программы с помощью команды continue (сокращенно c) и вышли из GDB с помощью команды quit (сокращенно q). (рис. 3.26), (рис. 3.27)

```
(gdb) c
Continuing.
gorld!

Breakpoint 2, _start () at lab10-2.asm:20
20      mov ebx, 0
(gdb) c
Continuing.
[Inferior 1 (process 4005) exited normally]
```

Рис. 3.26: .

```
[Inferior 1 (process 4005) exited normally]
(gdb) q
```

Рис. 3.27: .



## 3.5 Обработка аргументов командной строки в GDB

Скопировали файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки (Листинг 9.2) в файл с именем lab10-3.asm: (рис. 3.28)

```
[asstarceval@fedora lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
```

Рис. 3.28: .

Создали исполняемый файл. (рис. 3.29)

```
[asstarceval@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[asstarceval@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
```

Рис. 3.29: .

Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузили исполняемый файл в отладчик, указав аргументы: (рис. 3.30)

```
[asstarceval@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3
```

Рис. 3.30: .

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследовали расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установили точку останова перед первой инструкцией в программе и запустили ее. (рис. 3.31)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) r
Starting program: /home/asstarceval/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2
т\ 3

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.

Breakpoint 1, _start () at lab10-3.asm:5
5      pop еск ; Извлекаем из стека в `еск` количество
```

Рис. 3.31: .

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): (рис. 3.32)

```
(gdb) x/x $esp
0xffffd1a0: 0x00000005
```

Рис. 3.32: .

Как видно, число аргументов равно 5 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрели 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. (рис. 3.33)

```
(gdb) x/s *(void**)($esp + 4)
0xffffd35c: "/home/asstarceval/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd389: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd39b: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd3ac: "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd3ae: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 3.33: .

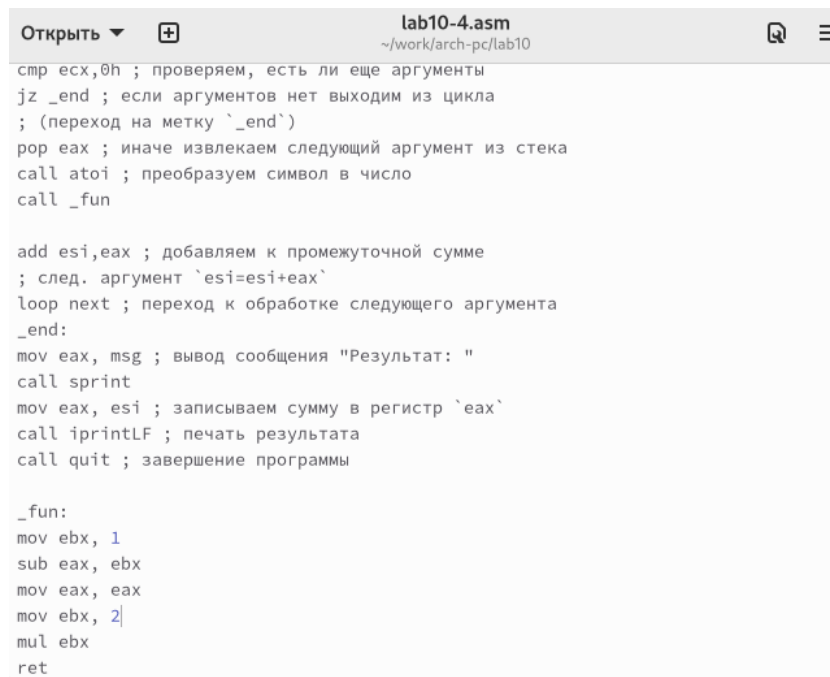
Шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.) потому что в теле цикла next 4 строки кода.

## 3.6 Задание для самостоятельной работы

1. Преобразовали программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму. (рис. 3.34), (рис. 3.35), (рис. 3.36)

```
[asstarceval@fedora lab10]$ cp ~/work/arch-pc/lab09/lab9-4.asm ~/work/arch-pc/lab10/lab10-4.asm
```

Рис. 3.34: .



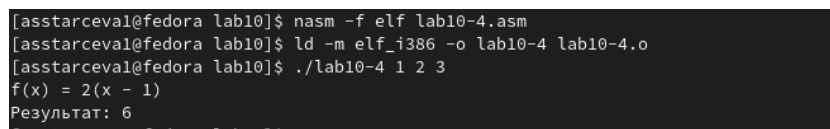
```
Открыть ▾ + lab10-4.asm
~/work/arch-pc/lab10

cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _fun

add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы

_fun:
mov ebx, 1
sub eax, ebx
mov eax, eax
mov ebx, 2
mul ebx
ret
```

Рис. 3.35: .

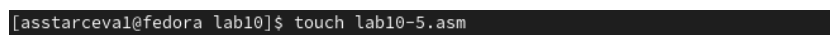


```
[asstarceval@fedora lab10]$ nasm -f elf lab10-4.asm
[asstarceval@fedora lab10]$ ld -m elf_i386 -o lab10-4 lab10-4.o
[asstarceval@fedora lab10]$ ./lab10-4 1 2 3
f(x) = 2(x - 1)
Результат: 6
```

Рис. 3.36: .

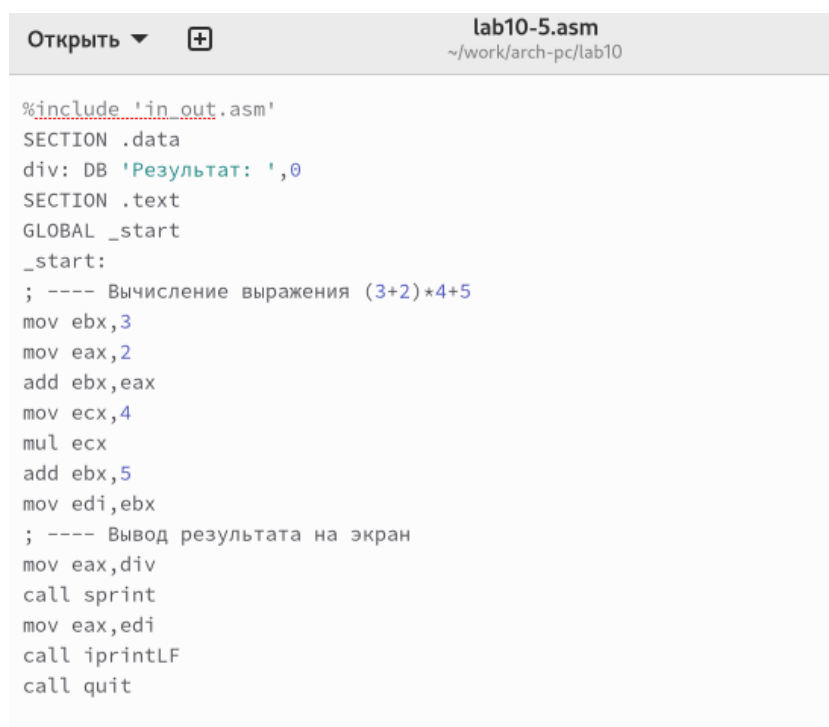
2. В листинге 10.3 приведена программа вычисления выражения  $(3 + 2) \times 4 + 5$ .

Создали файл (рис. 3.37), записали туда код листинга (рис. 3.38), создали исполняющий файл (рис. 3.39), при запуске обнаружили вывод неверного результата (рис. 3.40).



```
[asstarceval@fedora lab10]$ touch lab10-5.asm
```

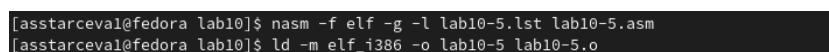
Рис. 3.37: .



```
Открыть ▾ + lab10-5.asm
~/work/arch-pc/lab10

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 3.38: .



```
[asstarceval@fedora lab10]$ nasm -f elf -g -l lab10-5.lst lab10-5.asm
[asstarceval@fedora lab10]$ ld -m elf_i386 -o lab10-5 lab10-5.o
```

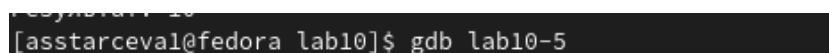
Рис. 3.39: .



```
[asstarceval@fedora lab10]$ ./lab10-5
Результат: 10
```

Рис. 3.40: .

Запустили файл в отладчике GDB (рис. 3.41), установили точку останова (рис. 3.42), запустили код (рис. 3.43), включили режим псевдографики (рис. 3.44), пошагово прошли все строчки кода (рис. 3.45), (рис. 3.46), (рис. 3.47), (рис. 3.48), (рис. 3.49), (рис. 3.50), (рис. 3.51), (рис. 3.52), обнаружили ошибку: вместо регистра ebx на 4 умножался eax, а 5 прибавлялась не к произведению, а только к ebx, исправили её (рис. 3.53), проверили результат работы программы (рис. 3.54).



```
[asstarceval@fedora lab10]$ gdb lab10-5
```

Рис. 3.41: .

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-5.asm, line 8.
```

Рис. 3.42: .

```
(gdb) r
Starting program: /home/asstarceval/work/arch-pc/lab10/lab10-5

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.

Breakpoint 1, _start () at lab10-5.asm:8
8      mov ebx,3
```

Рис. 3.43: .

```
[ Register Values Unavailable ]

B+> 0x80490e8 <_start> mov $0x3,%ebx
0x80490ed <_start+5> mov $0x2,%eax
0x80490f2 <_start+10> add %eax,%ebx
0x80490f4 <_start+12> mov $0x4,%ecx
0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19> add $0x5,%ebx
0x80490fe <_start+22> mov %ebx,%edi

native process 4708 In: _start L8 PC: 0x80490e8
(gdb) layout regs
(gdb)
```

Рис. 3.44: .

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd1f0 0xffffd1f0
ebp      0x0      0x0

B+> 0x80490e8 <_start> mov $0x3,%ebx
> 0x80490ed <_start+5> mov $0x2,%eax
0x80490f2 <_start+10> add %eax,%ebx
0x80490f4 <_start+12> mov $0x4,%ecx
0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19> add $0x5,%ebx
0x80490fe <_start+22> mov %ebx,%edi

native process 4708 In: _start L9 PC: 0x80490ed
(gdb) layout regs
(gdb) stepi
(gdb)
```

Рис. 3.45: .

```

Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffd1f0 0xffffd1f0
ebp      0x0      0x0

B+ 0x80490e8 <_start>    mov    $0x3,%ebx
0x80490ed <_start+5>    mov    $0x2,%eax
> 0x80490f2 <_start+10> add    %eax,%ebx
0x80490f4 <_start+12>    mov    $0x4,%ecx
0x80490f9 <_start+17>    mul    %ecx
0x80490fb <_start+19>    add    $0x5,%ebx
0x80490fe <_start+22>    mov    %ebx,%edi

native process 4708 In: _start L10 PC: 0x80490f2
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 3.46: .

```

Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffd1f0 0xffffd1f0
ebp      0x0      0x0

B+ 0x80490e8 <_start>    mov    $0x3,%ebx
0x80490ed <_start+5>    mov    $0x2,%eax
0x80490f2 <_start+10>    add    %eax,%ebx
> 0x80490f4 <_start+12>    mov    $0x4,%ecx
0x80490f9 <_start+17>    mul    %ecx
0x80490fb <_start+19>    add    $0x5,%ebx
0x80490fe <_start+22>    mov    %ebx,%edi

native process 4708 In: _start L11 PC: 0x80490f4
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 3.47: .

```

Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd1f0 0xffffd1f0
ebp      0x0      0x0

B+ 0x80490e8 <_start>    mov    $0x3,%ebx
0x80490ed <_start+5>    mov    $0x2,%eax
0x80490f2 <_start+10>    add    %eax,%ebx
0x80490f4 <_start+12>    mov    $0x4,%ecx
> 0x80490f9 <_start+17>    mul    %ecx
0x80490fb <_start+19>    add    $0x5,%ebx
0x80490fe <_start+22>    mov    %ebx,%edi

native process 4708 In: _start L12 PC: 0x80490f9
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 3.48: .

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd1f0 0xffffd1f0
ebp      0x0      0x0

0x80490f2 <_start+10> add    %eax,%ebx
0x80490f4 <_start+12> mov    $0x4,%ecx
0x80490f9 <_start+17> mul    %ecx
> 0x80490fb <_start+19> add    $0x5,%ebx
0x80490fe <_start+22> mov    %ebx,%edi
0x8049100 <_start+24> mov    $0x804a000,%eax
0x8049105 <_start+29> call   0x804900f <sprint>

native process 4708 In: _start L13 PC: 0x80490fb
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi

```

Рис. 3.49: .

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1f0 0xffffd1f0
ebp      0x0      0x0

0x80490f2 <_start+10> add    %eax,%ebx
0x80490f4 <_start+12> mov    $0x4,%ecx
0x80490f9 <_start+17> mul    %ecx
0x80490fb <_start+19> add    $0x5,%ebx
> 0x80490fe <_start+22> mov    %ebx,%edi
0x8049100 <_start+24> mov    $0x804a000,%eax
0x8049105 <_start+29> call   0x804900f <sprint>

native process 4708 In: _start L14 PC: 0x80490fe
(gdb) layout regs
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi

```

Рис. 3.50: .

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1f0 0xffffd1f0
ebp      0x0      0x0

0x80490f9 <_start+17> mul    %ecx
0x80490fb <_start+19> add    $0x5,%ebx
0x80490fe <_start+22> mov    %ebx,%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>

native process 4708 In: _start L16 PC: 0x8049100
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi

```

Рис. 3.51: .

```

Register group: general
eax      0x804a000 134520832
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1f0 0xffffd1f0
ebp      0x0      0x0

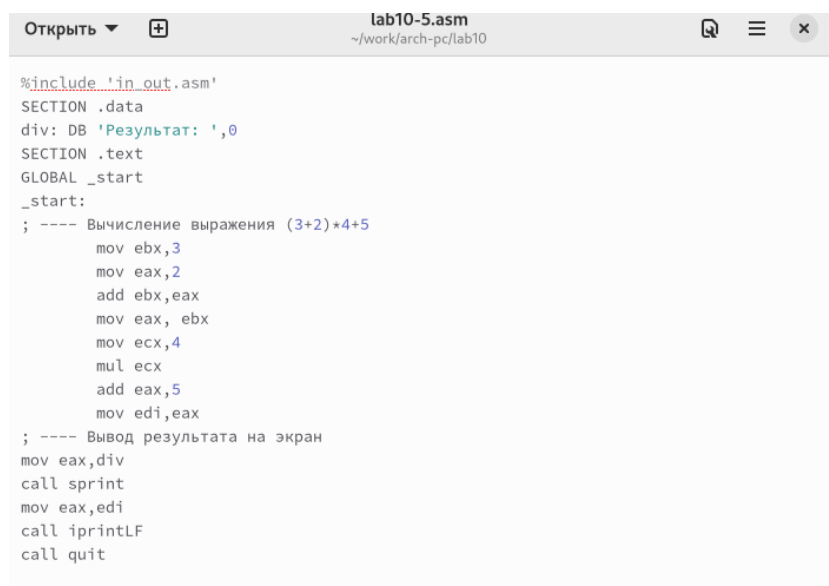
0x80490f9 <_start+17> mul    %ecx
0x80490fb <_start+19> add    $0x5,%ebx
0x80490fe <_start+22> mov    %ebx,%edi
0x8049100 <_start+24> mov    $0x804a000,%eax
> 0x8049105 <_start+29> call   0x804900f <sprint>
0x804910a <_start+34> mov    %edi,%eax
0x804910c <_start+36> call   0x8049086 <iprintLF>

native process 4708 In: _start L17 PC: 0x8049105
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi

```

Рис. 3.52: .

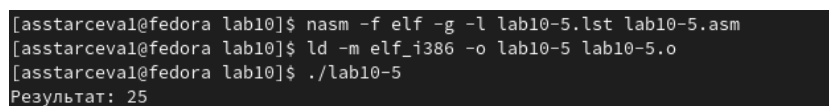




The screenshot shows a text editor window titled "lab10-5.asm" with the path "~/work/arch-pc/lab10". The code is as follows:

```
Открыть + lab10-5.asm ~/work/arch-pc/lab10
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
    mov ebx,3
    mov eax,2
    add ebx,eax
    mov eax, ebx
    mov ecx,4
    mul ecx
    add eax,5
    mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 3.53: .



The screenshot shows a terminal window with the following commands and output:

```
[asstarceval@fedora lab10]$ nasm -f elf -g -l lab10-5.lst lab10-5.asm
[asstarceval@fedora lab10]$ ld -m elf_i386 -o lab10-5 lab10-5.o
[asstarceval@fedora lab10]$ ./lab10-5
Результат: 25
```

Рис. 3.54: .

## 4 Выводы

В ходе выполнения лабораторной работы были приобретены навыки написания программ с использованием подпрограмм, ознакомились с методами отладки при помощи GDB и его основными возможностями.