

# **Отчёт по лабораторной работе №10**

Нефёдова Наталия Николаевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>5</b>
2.1	Реализация подпрограмм в NASM 1. . . . .	5
2.2	Пример программы с использованием вызова подпрограммы . .	5
2.3	Отладка программ с помощью GDB . . . . .	8
2.4	Добавление точек останова . . . . .	11
2.5	Работа с данными программы в GDB . . . . .	12
2.6	Обработка аргументов командной строки в GDB . . . . .	13
2.7	Задание для самостоятельной работы . . . . .	15
<b>3</b>	<b>Выводы</b>	<b>18</b>

## Список иллюстраций

[illegible]

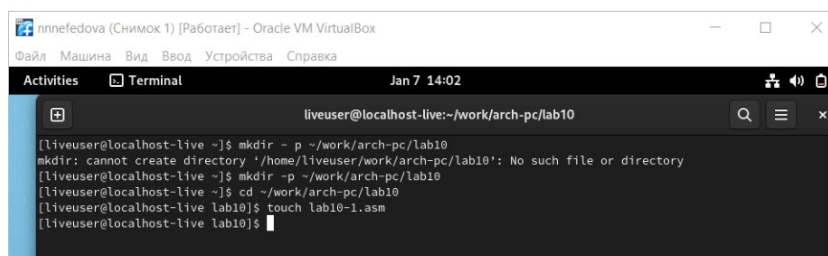
# 1 Цель работы

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

## 2 Выполнение лабораторной работы

### 2.1 Реализация подпрограмм в NASM 1.

Создадим каталог для выполнения лабораторной работы № 10, перейдем в него и создадим файл lab10-1.asm (рис. 2.1)

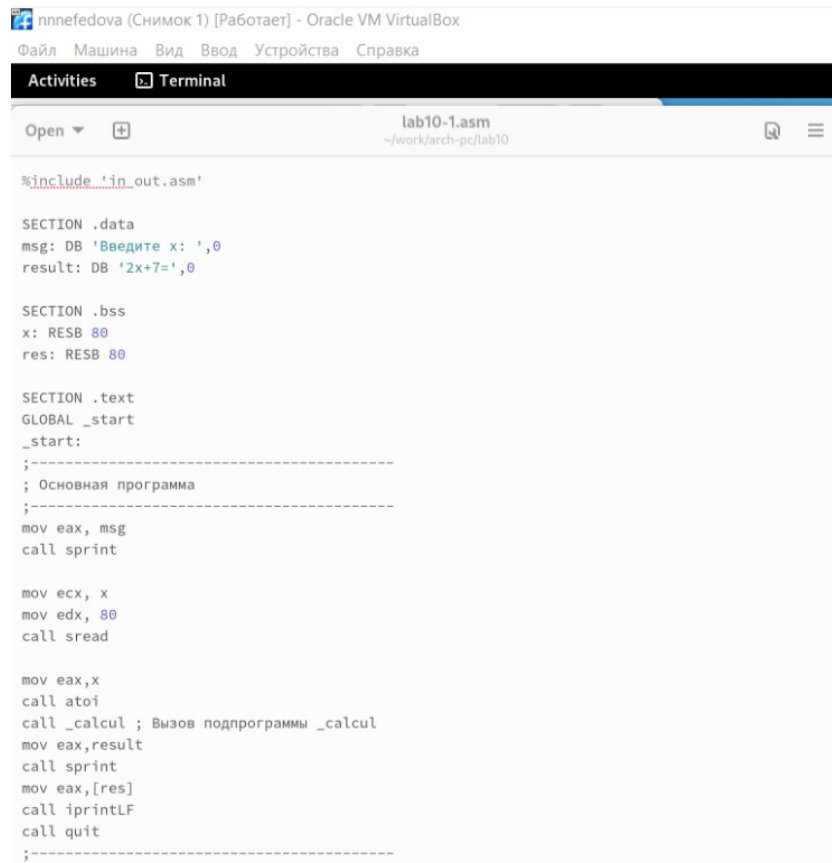


```
nnenedova (Снимок 1) [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
Activities  Terminal  Jan 7 14:02
liveuser@localhost-live:~/work/arch-pc/lab10
[liveuser@localhost-live ~]$ mkdir -p ~/work/arch-pc/lab10
mkdir: cannot create directory '/home/liveuser/work/arch-pc/lab10': No such file or directory
[liveuser@localhost-live ~]$ mkdir -p ~/work/arch-pc/lab10
[liveuser@localhost-live ~]$ cd ~/work/arch-pc/lab10
[liveuser@localhost-live lab10]$ touch lab10-1.asm
[liveuser@localhost-live lab10]$
```

Рис. 2.1: 1

### 2.2 Пример программы с использованием вызова подпрограммы

Рассмотрим программу с использованием вызова подпрограммы(рис. 2.2), (рис. 2.3)



nnnefedova (Снимок 1) [Работает] - Oracle VM VirtualBox

Файл Машина Вид Ввод Устройства Справка

Activities Terminal

lab10-1.asm  
~/work/arch-pc/lab10

```
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0

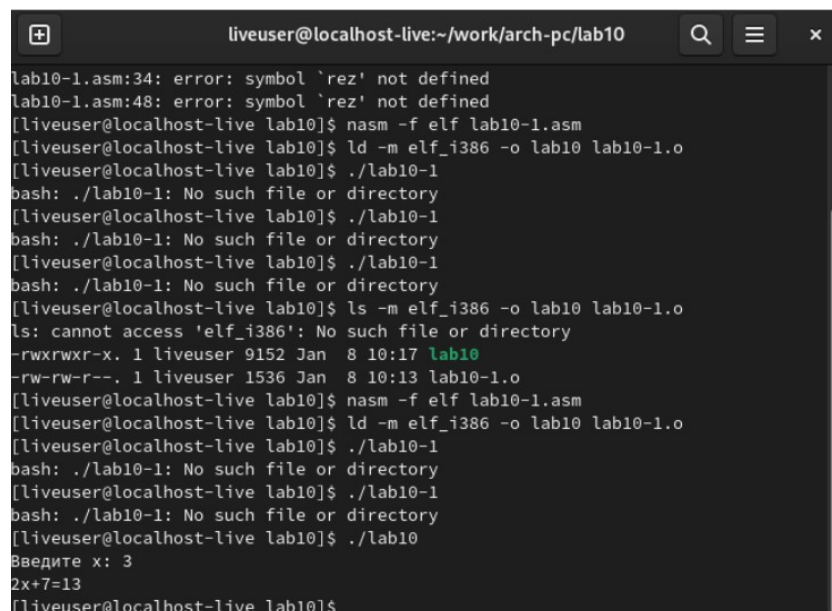
SECTION .bss
x: RESB 80
res: RESB 80

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
mov eax, [res]
call iprintLF
call quit
;-----
```

Рис. 2.2: 2



liveuser@localhost-live:~/work/arch-pc/lab10

```
lab10-1.asm:34: error: symbol 'rez' not defined
lab10-1.asm:48: error: symbol 'rez' not defined
[liveuser@localhost-live lab10]$ nasm -f elf lab10-1.asm
[liveuser@localhost-live lab10]$ ld -m elf_i386 -o lab10 lab10-1.o
[liveuser@localhost-live lab10]$ ./lab10-1
bash: ./lab10-1: No such file or directory
[liveuser@localhost-live lab10]$ ./lab10-1
bash: ./lab10-1: No such file or directory
[liveuser@localhost-live lab10]$ ./lab10-1
bash: ./lab10-1: No such file or directory
[liveuser@localhost-live lab10]$ ls -m elf_i386 -o lab10 lab10-1.o
ls: cannot access 'elf_i386': No such file or directory
-rwxrwxr-x. 1 liveuser 9152 Jan  8 10:17 lab10
-rw-rw-r--. 1 liveuser 1536 Jan  8 10:13 lab10-1.o
[liveuser@localhost-live lab10]$ nasm -f elf lab10-1.asm
[liveuser@localhost-live lab10]$ ld -m elf_i386 -o lab10 lab10-1.o
[liveuser@localhost-live lab10]$ ./lab10-1
bash: ./lab10-1: No such file or directory
[liveuser@localhost-live lab10]$ ./lab10-1
bash: ./lab10-1: No such file or directory
[liveuser@localhost-live lab10]$ ./lab10
Введите x: 3
2x+7=13
[liveuser@localhost-live lab10]$
```

Рис. 2.3: 3

Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введенных данных из символьного вида в численный (call atoi). После следующей инструкции call \_calcul, которая передает управление подпрограмме \_calcul, будут выполнены инструкции подпрограммы: mov ebx,2 mul ebx add eax,7 mov [rez],eax ret

Инструкция ret является последней в подпрограмме и ее исполнение приводит к возвращению в основную программу к инструкции, следующей за инструкцией call, которая вызвала данную подпрограмму. Последние строки программы реализуют вывод сообщения (call sprint), результата вычисления (call iprintLF) и завершение программы (call quit). Проверим работу программы и выведем результат на экран.

Изменим 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 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. 2.4)

```

;-----
; Подпрограмма вычисления
; выражения "2x+7"

_calcul:
call _subcalcul ; вызов подпрограммы _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [rez],eax

ret ; выход из подпрограммы

;-----
; Подпрограмма вычисления
; выражения "3x-1"

_subcalcul:
mov ebx,3
mul ebx
sub eax,1
mov [rez],eax

ret ; выход из подпрограммы

```


Рис. 2.4: 4

## 2.3 Отладка программ с помощью GDB

Создадим файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печатает сообщения Hello world!): Листинг 10.2. Программа вывода сообщения Hello world! (рис. 2.5), (рис. 2.6)

```
2x7-13
[liveuser@localhost-live lab10]$ nasm -f elf lab10-2.asm
nasm: fatal: unable to open input file `lab10-2.asm' No such file or directory
[liveuser@localhost-live lab10]$ touch lab10-2.asm
[liveuser@localhost-live lab10]$ nasm -f elf lab10-2.asm
[liveuser@localhost-live lab10]$ ld -m elf_i386 -o lab10 lab10-2.o
[liveuser@localhost-live lab10]$ ./lab10
Hello, world!
```

Рис. 2.5: 5



```
Open ▾ + lab10-2.asm
~/work/arch-pc/lab10

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. 2.6: 6

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. `nasm -f elf -g -l lab10-2.lst lab10-2.asm` `ld -m elf_i386 -o lab10-2 lab10-2.o` Загрузим исполняемый файл в отладчик gdb: (рис. 2.7)



```

[liveuser@localhost-live lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[liveuser@localhost-live lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[liveuser@localhost-live lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora 11.2-3.fc36
Copyright (C) 2022 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 lab10-2...
(gdb)

```

Рис. 2.7: 7

Проверим работу программы, запустив ее в оболочке GDB с помощью команды `run` (сокращённо `r`) (рис. 2.8), (рис. 2.9)

```

(gdb) run
Starting program: /home/liveuser/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])

```

Рис. 2.8: 8

```

[Inferior 1 (process 37546) exited normally]
(gdb) r
Starting program: /home/liveuser/work/arch-pc/lab10/lab10-2
Hello, world!

```

Рис. 2.9: 9

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её. (рис. 2.10), (рис. 2.11)

```

Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
Undefined command: "". Try "help".
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
No source file named _start
Breakpoint 1 at 0x8049000.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (_start
Breakpoint 1 at 0x8049000: file lab10-2.asm , line 12.) pending.
(gdb)

```

Рис. 2.10: 10

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 12.
No source file named _start
Breakpoint 1 at 0x8049000.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 2 (_start
Breakpoint 1 at 0x8049000: file lab10-2.asm , line 12.) pending.
(gdb) run
Starting program: /home/liveuser/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 37636) exited normally]
(gdb)

```

Рис. 2.11: 11

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 2.12)

```

(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. 2.12: 12

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (gdb) `set disassembly-flavor intel` (gdb) `disassemble _start` (рис. 2.13)

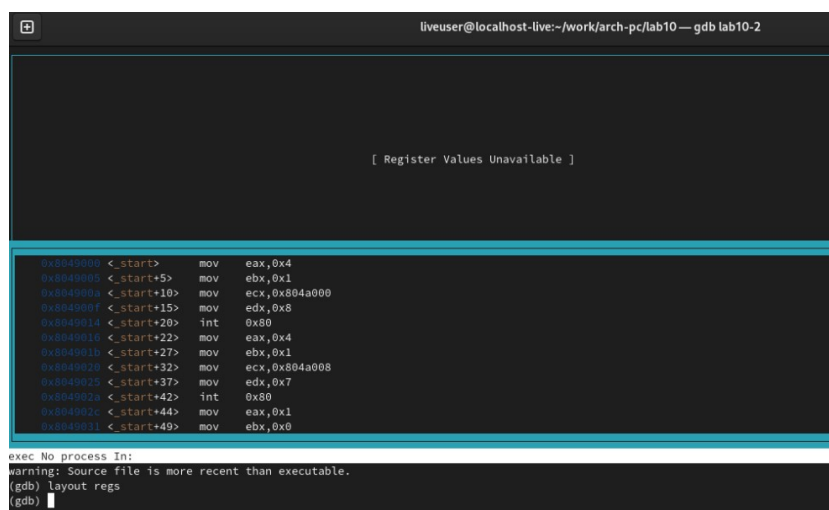
```

(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. 2.13: 13

В этом режиме есть три окна: - В верхней части видны названия регистров и их текущие значения; - В средней части виден результат дисассимилирования программы; - Нижняя часть доступна для ввода команд. (рис. 2.14)



```

liveuser@localhost-live:~/work/arch-pc/lab10 — gdb lab10-2

[ Register Values Unavailable ]

0x08049000 <_start>    mov     eax,0x4
0x08049005 <_start+5>  mov     ebx,0x1
0x0804900a <_start+10> mov     ecx,0x804a000
0x0804900f <_start+15> mov     edx,0x8
0x08049014 <_start+20> int     0x80
0x08049016 <_start+22> mov     eax,0x4
0x0804901b <_start+27> mov     ebx,0x1
0x08049020 <_start+32> mov     ecx,0x804a008
0x08049025 <_start+37> mov     edx,0x7
0x0804902a <_start+42> int     0x80
0x0804902c <_start+44> mov     eax,0x1
0x08049031 <_start+49> mov     ebx,0x0

exec No process in:
warning: Source file is more recent than executable.
(gdb) layout regs
(gdb)

```

Рис. 2.14: 14

## 2.4 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 компилиро-

валась 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`). Проверим это с помощью команды `info breakpoints` (кратко `i b`): `(gdb) info breakpoints`. (рис. 2.15)

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   <PENDING>   _start
Breakpoint 1 at 0x8049000: file lab10-2.asm , line 12.
2        breakpoint      keep y   <PENDING>   _start
Breakpoint 1 at 0x8049000: file lab10-2.asm , line 12.
(gdb)
```

Рис. 2.15: 15

## 2.5 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследим за изменением значений регистров. Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`). `(gdb) info registers` (рис. 2.16)

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

Рис. 2.16: 16

Посмотрите значение переменной `msg2` по адресу. (рис. 2.17)

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

Рис. 2.17: 17

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 (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си). Изменим первый символ переменной `msg1`. (рис. 2.18)

```
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
```

Рис. 2.18: 18

Заменяем второй символ в переменной `msg2` (рис. 2.19)

```
(gdb) set {char}0x804a009='0'
Cannot access memory at address 0x804a009
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "world!\n"<error: Cannot acc
```

Рис. 2.19: 19

Чтобы посмотреть значения регистров используется команда `print /F` (перед именем регистра обязательно ставится префикс `$`).

Завершим выполнение программы с помощью команды `continue` (сокращенно `c`) и выйдем из GDB с помощью команды `quit` (сокращенно `q`).

## 2.6 Обработка аргументов командной строки в GDB

Скопируем файл `lab9-2.asm`, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки в файл с именем `lab10-3.asm` и создадим исполняемый файл. (рис. 2.20)

```

lliveuser@localhost-live lab09]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
lliveuser@localhost-live lab09]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
nasm: fatal: unable to open input file 'lab10-3.asm' No such file or directory
lliveuser@localhost-live lab09]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
nasm: fatal: unable to open input file 'lab10-3.asm' No such file or directory
lliveuser@localhost-live lab09]$ nasm -f elf lab9-3.asm
nasm: fatal: unable to open input file 'lab9-3.asm' No such file or directory
lliveuser@localhost-live lab09]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
nasm: fatal: unable to open input file 'lab10-3.asm' No such file or directory
lliveuser@localhost-live lab09]$ cd lab10
bash: cd: lab10: No such file or directory
lliveuser@localhost-live lab09]$ cd ~/work/arch-pc/lab10
lliveuser@localhost-live lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
lliveuser@localhost-live lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
lliveuser@localhost-live lab10]$

```

Рис. 2.20: 20

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. (рис. 2.21)

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 11.
(gdb) run
Starting program: /home/adutkina/work/arch-pc/lab10/lab10-3 1 2 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.
Downloading separate debug info for /home/adutkina/work/arch-pc/lab10/system-sup
plied DSO at 0xf7ffc000...

Breakpoint 1, _start () at lab10-3.asm:11
11      pop ecx          ; Извлекаем из стека в `ecx` количество
(gdb)

```

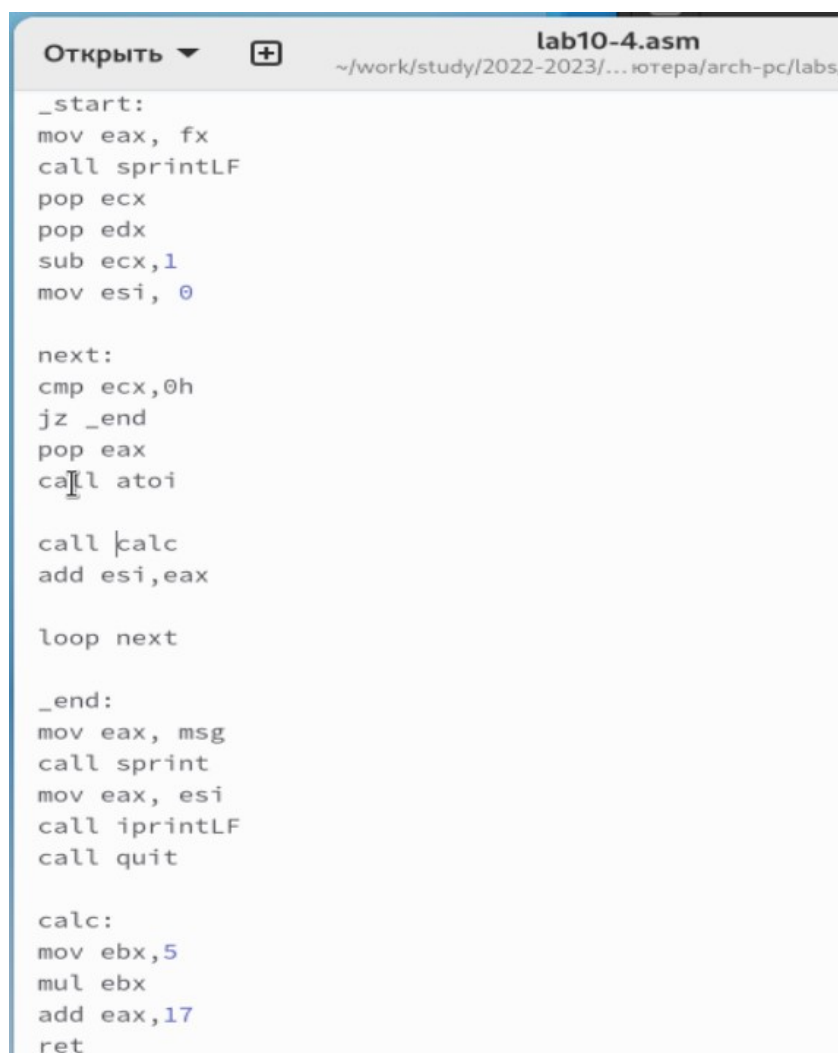
Рис. 2.21: 21

Адрес вершины стека храниться в регистре есп и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 4 - расположение программы и три аргумента.

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.

## 2.7 Задание для самостоятельной работы

1. Преобразуем программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму.
2. В листинге 10.3 приведена программа вычисления выражения  $(3+2)*4+5$ . При запуске данная программа дает неверный результат. Проверим это. С помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправим ее. (рис. 2.22), (рис. 2.23), (рис. 2.24)



```
lab10-4.asm
~/work/study/2022-2023/...ютеpa/arch-pc/labs/

_start:
mov eax, fx
call sprintLF
pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

call calc
add esi, eax

loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

calc:
mov ebx, 5
mul ebx
add eax, 17
ret
```

Рис. 2.22: 22

```

Открыть ▾ + lab10-5.asm
~/work/study/2022-2023/...ютепа/arch-pc/labs/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. 2.23: 23

```

Register group: general
eax 0x2 2 ecx 0x4 4
edx 0x0 0 ebx 0x5 5
esp 0xffffd150 0xffffd150 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17> eflags 0x206 [ PF IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

0+ 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f3 <_start+12> mov ecx,0x4
> 0x80490f9 <_start+17> mul ecx
0x80490fd <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804a00f <sprint>

Active process 2835 In: _start L12 PC: 0x80490f9

breakpoint 1, _start () at lab10-5.asm:8
gdb) si
gdb) sis
undefined command: "sis". Try "help".
gdb) is
undefined command: "is". Try "help".
gdb) si
gdb) si
gdb) si
gdb) si

```

Рис. 2.24: 24

Отметим, что перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax (рис. 2.25)



```
%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 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. 2.25: 25

## **3 Выводы**

В ходе лабораторной работы были приобретены навыки написания программ с использованием подпрограмм и изучены методы отладки при помощи GDB и его основные возможности