

# **Отчет по лабораторной работе №10**

**Дисциплина: Архитектура компьютера**

**Никулина Ксения Ильинична**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Самостоятельная работа</b>	<b>23</b>
<b>6</b>	<b>Выводы</b>	<b>27</b>

## Список иллюстраций

4.1	Содание файла . . . . .	8
4.2	Текст . . . . .	9
4.3	Проверка работы файла . . . . .	9
4.4	Текст . . . . .	10
4.5	Проверка работы файла . . . . .	11
4.6	Создание файла . . . . .	11
4.7	Текст . . . . .	11
4.8	Загрузка файла . . . . .	12
4.9	Проверка работы . . . . .	12
4.10	Запуск . . . . .	12
4.11	Код программы . . . . .	13
4.12	Команды . . . . .	14
4.13	1 этап . . . . .	15
4.14	2 этап . . . . .	16
4.15	Проверка . . . . .	16
4.16	Выполнение . . . . .	17
4.17	Информация . . . . .	17
4.18	Содержимое регистров . . . . .	18
4.19	Значение переменной . . . . .	18
4.20	Замена 'H' на 'h' . . . . .	18
4.21	Замена 'W' на 'k' . . . . .	18
4.22	Значения регистра edx . . . . .	19
4.23	1 вариант . . . . .	20
4.24	2 вариант . . . . .	20
4.25	Копирование файла . . . . .	20
4.26	Значения регистра edx . . . . .	21
4.27	Запуск . . . . .	21
4.28	Количество аргументов . . . . .	21
4.29	Остальные позиции . . . . .	22
5.1	Текст . . . . .	24
5.2	Проверка работы . . . . .	25
5.3	Текст без ошибок . . . . .	25
5.4	Проверка работы . . . . .	26

## **Список таблиц**

# 1 Цель работы

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

## 2 Задание


Приобрести навыки написания программ с использованием подпрограмм. Познакомиться с методами отладки при помощи GDB и его основными возможностями.

### 3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: • обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

## 4 Выполнение лабораторной работы

1. Создала каталог для выполнения лабораторной работы No 10, перешла в него и создала файл lab10-1.asm (рис. 4.1)



```
kinikulina@dk8n72:~/work/arch-pc/lab10
kinikulina@dk8n72 ~ $ mkdir ~/work/arch-pc/lab10
kinikulina@dk8n72 ~ $ cd ~/work/arch-pc/lab10
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ touch lab10-1.asm
kinikulina@dk8n72 ~/work/arch-pc/lab10 $
```

Рис. 4.1: Содание файла

2. Ввела в файл lab10-1.asm текст программы из листинга 10.1. Создала исполняемый файл и проверила его работу(рис. 4.2, рис.4.3 )



```
kinikulina@dk8n72:~/work/arch-pc/lab10
...dk.sci.pfu.edu.ru/home/k/i/kinikulina/work/arch-pc/lab10/lab1
%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
mov eax, [rez]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [rez], eax
```

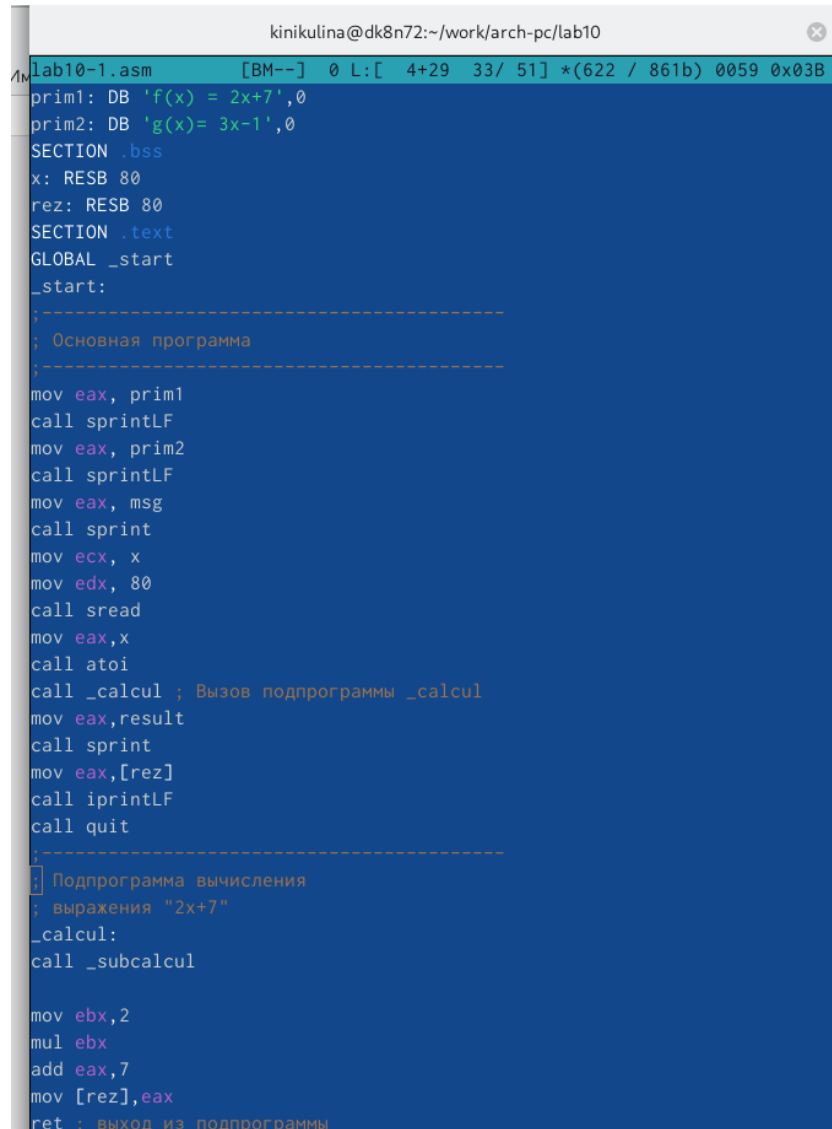
Рис. 4.2: Текст

```
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 4
2x+7=15
kinikulina@dk8n72 ~/work/arch-pc/lab10 $
```

Рис. 4.3: Проверка работы файла

3. Изменила текст программы, добавив подпрограмму `_subcalcul` в подпро-

грамму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . Создала исполняемый файл и проверила его работу (рис. 4.4, рис.4.5 )



```
lab10-1.asm [BM--] 0 L:[ 4+29 33/ 51] *(622 / 861b) 0059 0x03B
prim1: DB 'f(x) = 2x+7',0
prim2: DB 'g(x)= 3x-1',0
SECTION .bss
x: RESB 80
rez: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, prim1
call sprintf
mov eax, prim2
call sprintf
mov eax, msg
call sprintf
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprintf
mov eax, [rez]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul

mov ebx, 2
mul ebx
add eax, 7
mov [rez], eax
ret ; выход из подпрограммы
```

Рис. 4.4: Текст

```

kinikulina@dk8n72 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ./lab10-1
f(x) = 2x+7
g(x)= 3x-1
Введите x: 4
f(g(x))=29

```

Рис. 4.5: Проверка работы файла

4. Создала файл lab10-2.asm с текстом программы из Листинга 10.2 (рис. 4.6, рис.4.7 )

```

kinikulina@dk8n72 ~/work/arch-pc/lab10 $ touch lab10-2.asm

```

Рис. 4.6: Создание файла

```

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. 4.7: Текст

5. Загрузила исполняемый файл в отладчик gdb (рис. 4.8)

```

kinikulina@dk8n72 ~/work/arch-pc/lab10 $ touch lab10-2.asm
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ nasm -f elf -g -l lab10-2.lst lab10-2.asm
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-2 lab10-2.o
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ gdb lab10-2
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-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 lab10-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/i/kinikulina/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 3532) exited normally]
(gdb)

```

Рис. 4.8: Загрузка файла

6. Загрузила исполняемый файл в отладчик gdb. Проверила работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 4.9)

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/i/kinikulina/work/arch-pc/lab10/lab10-2

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

```

Рис. 4.9: Проверка работы

7. Для более подробного анализа программы установила брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустила её (рис. 4.10)

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb)

```

Рис. 4.10: Запуск

8. Посмотрела дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 4.11)

```

5      mov     ecx, 4
(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. 4.11: Код программы

9. Переключилась на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 4.12 )

```

(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.
(gdb) █

```

Рис. 4.12: Команды

10. Включил режим псевдографики для более удобного анализа программы (рис. 4.13, рис.4.14 )

```
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
0x8049020 <_start+32> mov    ecx,0x804a008
0x8049025 <_start+37> mov    edx,0x7
0x804902a <_start+42> int    0x80
0x804902c <_start+44> mov    eax,0x1
0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038 add    BYTE PTR [eax],al
0x804903a add    BYTE PTR [eax],al
0x804903c add    BYTE PTR [eax],al
0x804903e add    BYTE PTR [eax],al
0x8049040 add    BYTE PTR [eax],al
0x8049042 add    BYTE PTR [eax],al
0x8049044 add    BYTE PTR [eax],al
0x8049046 add    BYTE PTR [eax],al
0x8049048 add    BYTE PTR [eax],al
0x804904a add    BYTE PTR [eax],al
0x804904c add    BYTE PTR [eax],al

native process 3556 In: _start
(gdb) 
```

Рис. 4.13: 1 этап

```
[ 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
0x8049020 <_start+32>  mov    ecx,0x804a008
0x8049025 <_start+37>  mov    edx,0x7
0x804902a <_start+42>  int     0x80
0x804902c <_start+44>  mov    eax,0x1

native process 3556 In: _start
(gdb) layout regs
(gdb) □
```

Рис. 4.14: 2 этап

11. На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверила это с помощью команды info breakpoints (рис. 4.15 )

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

Рис. 4.15: Проверка

12. Определила адрес предпоследней инструкции (mov ebx,0x0) и установила точку останова.(рис. 4.16 )



```

0x8049014 <_start+20>  int    0x80
0x8049016 <_start+22>  mov     eax,0x4
0x804901b <_start+27>  mov     ebx,0x1
0x8049020 <_start+32>  mov     ecx,0x804a008
0x8049025 <_start+37>  mov     edx,0x7
0x804902a <_start+42>  int     0x80
0x804902c <_start+44>  mov     eax,0x1
b+ 0x8049031 <_start+49>  mov     ebx,0x0
0x8049036 <_start+54>  int     0x80
0x8049038             add     BYTE PTR [eax],al
0x804903a             add     BYTE PTR [eax],al

native process 3556 In: _start L9 PC: 0x80
(gdb) layout regs
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y  0x08049000 lab10-2.asm:9
        breakpoint already hit 1 time
(gdb) break
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab10-2.asm, line 9.
(gdb) break *0x8049031
Breakpoint 3 at 0x8049031: file lab10-2.asm, line 20.

```

Рис. 4.16: Выполнение

13. Посмотрела информацию о всех установленных точках останова (рис. 4.17 )

```

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

```

Рис. 4.17: Информация

14. Посмотрела содержимое регистров также можно с помощью команды info registers (или i r) (рис. 4.18 )

```

native process 3556 in: _start
ebx      0x0      0
esp      0xffffc630 0xffffc630
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
--Type <RET> for more, q to quit, c to continue without paging--ces
3
fs       0x0      0
gs       0x0      0
(gdb)

```

Рис. 4.18: Содержимое регистров

15. Посмотрела значение переменной msg1 по имени (рис. 4.19)

```

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

```

Рис. 4.19: Значение переменной

16. Изменила первый символ переменной msg1 (рис. 4.20)

```

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

```

Рис. 4.20: Замена 'H' на 'h'

17. Заменяла символ во второй переменной msg2 (рис. 4.21)

```

(gdb) set {char}&msg2='k'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "kor1d!\n\034"

```

Рис. 4.21: Замена 'W' на 'k'

18. Вывела в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 4.22 )

```
edx      0x0      0
ebx      0x0      0
esp      0xffffc630 0xffffc630
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+> 0x8049000 <_start> mov $0x4,%eax
      0x8049005 <_start+5> mov $0x1,%ebx
      0x804900a <_start+10> mov $0x804a000,%
      0x804900f <_start+15> mov $0x8,%edx
      0x8049014 <_start+20> int $0x80
      0x8049016 <_start+22> mov $0x4,%eax
      0x804901b <_start+27> mov $0x1,%ebx
      0x8049020 <_start+32> mov $0x804a008,%
      0x8049025 <_start+37> mov $0x7,%edx
      0x804902a <_start+42> int $0x80

native process 3838 In: _start
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='k'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "korld!\n\034"
(gdb) p/x $edx
$1 = 0x0
(gdb) p/t $edx
$2 = 0
(gdb) p/s $edx
$3 = 0
```

Рис. 4.22: Значения регистра edx

19. С помощью команды set изменила значение регистра ebx (рис. 4.23, рис.4.24 )

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

Рис. 4.23: 1 вариант

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

Рис. 4.24: 2 вариант

20. Скопировала файл lab9-2.asm, созданный при выполнении лабораторной работы No9, с программой выводящей на экран аргументы командной строки (Листинг 9.2) в файл с именем lab10-3.asm (рис. 4.25 )

```
kinikulina@dk8n72 ~ $ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
```

Рис. 4.25: Копирование файла

21. Создала исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ -args. Загрузила исполняемый файл в отладчик, указав аргументы (рис. 4.26 )

```

kinikulina@dk8n72 ~/work/arch-pc/lab10 $ nasm -f elf -g -l lab10-3.lst lab10-3.asm
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-3 lab10-3.o
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-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 lab10-3...
(gdb)

```

Рис. 4.26: Значения регистра edx

22. Для начала установила точку останова перед первой инструкцией в программе и запустила ее.(рис. 4.27 )

```

Breakpoint 1 at 0x00000000: File lab10-3.asm, line 5
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/i/kinikulina/work/arch-pc/lab10/lab10-3 аргумент 2 аргумент\ 3

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

```

Рис. 4.27: Запуск

23. Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы) (рис. 4.28 )

```

(gdb) x/x $esp
0xfffffc5e0:      0x00000005
(gdb)

```

Рис. 4.28: Количество аргументов

24. Посмотрела остальные позиции стека (рис. 4.29 )

```

5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffffc5e0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc84e: "/afs/.dk.sci.pfu.edu.ru/home/k/i/kinikulina/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc895: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc8a7: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc8b8: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc8ba: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
A syntax error in expression, near `'.
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) 

```

Рис. 4.29: Остальные позиции

## 5 Самостоятельная работа

1. Преобразовала программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму. Проверила работу (рис. 5.1, рис.5.2 )

```
/afs/.dk.sci.pfu.edu.ru/home/
mov eax,prim
call sprintLF
next:
cmp ecx,0
jz _end

call vadim

add esi, eax
loop next

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

vadim:

mov ebx,2
pop eax
call atoi
mul ebx

add eax,7
ret
```

Рис. 5.1: Текст



```

Результат: 4283517068
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ nasm -f elf sam_rabota10-1.asm
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o sam_rabota10-1 sam_rabota10-1.o
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ./sam_rabota10-1 3 2 3 40
f(x) = 7 + 2x
3
2
3
40
Результат: 124
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ./sam_rabota10-1 1 2 3
f(x) = 7 + 2x
1
2
3
Результат: 33

```

Рис. 5.2: Проверка работы

2. С помощью отладчика GDB, анализируя изменения значений регистров, определила ошибку и исправила ее (рис. 5.3, рис.5.4)

```

#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. 5.3: Текст без ошибок

```
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ nasm -f elf sam_rabota10-2.asm
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o sam_rabota10-2 sam_rabota10-2.o
kinikulina@dk8n72 ~/work/arch-pc/lab10 $ ./sam_rabota10-2
Результат: 25
```

Рис. 5.4: Проверка работы

## 6 Выводы

Приобрела навыки написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.