

Отчёт по лабораторной работы №9

Дисциплина: архитектура компьютера

Худдыева Дженнет

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация подпрограмм в NASM	9
4.2	Отладка программ с помощью GDB	11
4.2.1	Добавление точек останова	15
4.2.2	Работа с данными программы в GDB	16
4.2.3	Обработка аргументов командной строки в GDB	20
4.3	Задания для самостоятельной работы	22
5	Выводы	26

Список иллюстраций

4.1	Создание файлов для лабораторной работы	10
4.2	Запуск исполняемого файла	10
4.3	Изменение текста программы	11
4.4	Запуск исполняемого файла	11
4.5	Ввод текста программы	12
4.6	Получение исполняемого файла	13
4.7	Загрузка исполняемого файла в отладчик	13
4.8	Проверка работы файла с помощью команды run	13
4.9	Установка брейкпоинта и запуск программы	13
4.10	Использование команд disassemble и disassembly-flavor intel . . .	14
4.11	Использование команд disassemble и disassembly-flavor intel . . .	14
4.12	Включение режима псевдографики	15
4.13	Установление точек останова и просмотр информации о них . . .	15
4.14	До использования команды stepi	16
4.15	До использования команды stepi	16
4.16	Просмотр значений переменных	17
4.17	Использование команды set	18
4.18	Вывод значения регистра в разных представлениях	19
4.19	Использование команды set для изменения значения регистра . .	20
4.20	Создание файла	20
4.21	Создание файла	21
4.22	Загрузка файла с аргументами в отладчик	21
4.23	Установление точки останова и запуск программы	21
4.24	Просмотр значений, введенных в стек	22
4.25	Написание кода подпрограммы	23
4.26	Запуск программы и проверка его вывода	24
4.27	Ввод текста программы из листинга 9.3	24
4.28	Создание и запуск исполняемого файла	25

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

- 1.Реализация подпрограмм в NASM
- 2.Отладки подпрограмм с помощью GDB.
- 3.Добавление точек останова.
- 4.Работа с данными подпрограмм в GDB.
- 5.Обработка аргументов командной строки в GDB.
- 6.Задания для самостоятельной работы.

3 Теоретическое введение

Отладка - это процесс поиска и исправления ошибок в программе. Отладчики позволяют управлять ходом выполнения подпрограммы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправления. Наиболее популярные способы работы с отладчиком - это использование точек останова и выполнение программы по шагам.

GDB (GNU Debugger - отладчик проекта GNU) работает на многих UNIX - подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит “внутри” программы в момент её выполнения или что делает программа в момент сбоя.

Команда `run` - запускает отлаживаемую программу в оболочке GDB.

Команда `kill` - прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (да), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точка отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда `quit(q)`.

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`.

Установить точку останова можно командой `break`. Типичный аргумент этой команды - место установки. Его можно задать как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится *

Команда `si` позволяет выполнять программу по шагам.

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. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволяет уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `esp` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `esp`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab9-1.asm. Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис.[4.1])

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [res], eax
35 ret ; выход из подпрограммы

```

Рис. 4.1: Создание файлов для лабораторной работы

Создаю исполняемый файл и проверяю его работу. (рис. [4.2])

```

dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 4
2x+7=29

```

Рис. 4.2: Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. [4.3])

```

21 call _subcalcul ; Вызов подпрограммы _calcul
22 call _calcul
23 mov eax,result
24 call sprint
25 mov eax,[res]
26 call iprintLF
27 call quit
28 ;-----
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32 mov ebx,2
33 mul ebx
34 add eax,7
35 mov [res],eax
36 ret ; выход из подпрограммы
37
38 _subcalcul:
39 mov ebx,3
40 mul ebx
41 add eax,-1
42 ret

```

Рис. 4.3: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. [4.4])

```

dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 4
2x+7=15
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$

```

Рис. 4.4: Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. [4.5])

```
1 SECTION .data
2 msg1:db "Hello, ",0x0
3 msg1Len:equ $ - msg1
4 msg2:db "world!",0xa
5 msg2Len:equ $ - msg2
6 SECTION .text
7 global _start
8 _start:
9 mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
14 mov eax, 4
15 mov ebx, 1
16 mov ecx, msg2
17 mov edx, msg2Len
18 int 0x80
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
```

Рис. 4.5: Ввод текста программы

Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. [4.6])

```

dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ touch lab9-2.asm
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ gedit lab9-2.asm
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ gdb lab9-2

```

Рис. 4.6: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. [4.7])

```

GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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 lab9-2...

```

Рис. 4.7: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. [4.8])

```

(gdb) run
Starting program: /home/dkhuddiheva/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 3913) exited normally]

```

Рис. 4.8: Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start и запускаю её. (рис.[4.9])

```

(gdb) break _start
Breakpoint 1 at 0x80490e8: file lab9-2.asm, line 10.
(gdb) run
Starting program: /home/dkhuddiheva/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:10
10      mov eax, 4

```

Рис. 4.9: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. [4.11])

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:      mov     $0x4,%eax
    0x080490ed <+5>:      mov     $0x1,%ebx
    0x080490f2 <+10>:     mov     $0x804a000,%ecx
    0x080490f7 <+15>:     mov     $0x8,%edx
    0x080490fc <+20>:     int     $0x80
    0x080490fe <+22>:     mov     $0x4,%eax
    0x08049103 <+27>:     mov     $0x1,%ebx
    0x08049108 <+32>:     mov     $0x804a008,%ecx
    0x0804910d <+37>:     mov     $0x7,%edx
    0x08049112 <+42>:     int     $0x80
    0x08049114 <+44>:     mov     $0x1,%eax
    0x08049119 <+49>:     mov     $0x0,%ebx
    0x0804911e <+54>:     int     $0x80
End of assembler dump.
```

Рис. 4.10: Использование команд `disassemble` и `disassembly-flavor intel`

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:      mov     eax,0x4
    0x080490ed <+5>:      mov     ebx,0x1
    0x080490f2 <+10>:     mov     ecx,0x804a000
    0x080490f7 <+15>:     mov     edx,0x8
    0x080490fc <+20>:     int     0x80
    0x080490fe <+22>:     mov     eax,0x4
    0x08049103 <+27>:     mov     ebx,0x1
    0x08049108 <+32>:     mov     ecx,0x804a008
    0x0804910d <+37>:     mov     edx,0x7
    0x08049112 <+42>:     int     0x80
    0x08049114 <+44>:     mov     eax,0x1
    0x08049119 <+49>:     mov     ebx,0x0
    0x0804911e <+54>:     int     0x80
End of assembler dump.
```

Рис. 4.11: Использование команд `disassemble` и `disassembly-flavor intel`

В режиме АТТ имена регистров начинаются с символа `%`, а имена операндов с `$`, в то время как в Intel используется привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы с помощью команд `layout asm` и `layout regs`. (рис. [4.12])

```

[ Register Values Unavailable ]

B+> 0x80490e8 <_start>    mov     eax,0x4
0x80490ed <_start+5>      mov     ebx,0x1
0x80490f2 <_start+10>     mov     ecx,0x804a000
0x80490f7 <_start+15>     mov     edx,0x8
0x80490fc <_start+20>     int     0x80
0x80490fe <_start+22>     mov     eax,0x4
0x8049103 <_start+27>     mov     ebx,0x1

native process 3933 In: _start          L10    PC: 0x80490e8
(gdb) layout regs
(gdb)

```

Рис. 4.12: Включение режима псевдографики

4.2.1 Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова. (рис. [4.13])

```

[ Register Values Unavailable ]

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

native process 3486 In: _start          L9     PC
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file laba9-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000  laba9-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031  laba9-2.asm:20
(gdb)

```

Рис. 4.13: Установление точек останова и просмотр информации о них

4.2.2 Работа с данными программы в GDB

Выполняю 4 инструкции с помощью команды `stepi` и слежу за изменением значений регистров. (рис. [4.15])

```
B+> 0x80490e8 <_start> mov    eax,0x4
0x80490ed <_start+5> mov    ebx,0x1
0x80490f2 <_start+10> mov    ecx,0x804a000
0x80490f7 <_start+15> mov    edx,0x8
0x80490fc <_start+20> int     0x80
0x80490fe <_start+22> mov    eax,0x4
0x8049103 <_start+27> mov    ebx,0x1
0x8049108 <_start+32> mov    ecx,0x804a008
0x804910d <_start+37> mov    edx,0x7

native process 3933 In: _start L10 PC: 0x80490e8
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd180 0xffffd180
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.14: До использования команды `stepi`

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd180 0xffffd180
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 3486 In: _start L14 PC: 0x8049016
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si 5
(gdb)
```

Рис. 4.15: До использования команды `stepi`

Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной msg1 с помощью команды x/1sb &msg1 значение переменной msg2 по ее адресу. (рис. [4.16])

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd180 0xffffd180
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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 3486 In: start L14 PC: 0x8049016
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) si 5
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)
```

Рис. 4.16: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. [4.17])

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd180 0xffffd180
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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 3486 In: _start L14 PC: 0x8049016
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='z'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "zorld!\n\034"
```

Рис. 4.17: Использование команды set

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра edx с помощью команды print p/F \$val. (рис. [4.18])

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd180 0xffffd180
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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 3486 In: _start L14 PC: 0x8049016
(gdb) x/1sb &msg2
0x804a008 <msg2>: "zorld!\n\034"
(gdb) p/s $edx
$1 = 8
(gdb) p/x $edx
$2 = 0x8
(gdb) p/t $ebx
$3 = 1
(gdb) p/c $eax
$4 = 8 '\b'

```

Рис. 4.18: Вывод значения регистра в разных представлениях

С помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. [4.19])

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x2      2
esp      0xffffd180 0xffffd180
ebp      0x0      0x0

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

native process 3376 In: _start L9
fs      0x0      0
gs      0x0      0
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2

```

Рис. 4.19: Использование команды set для изменения значения регистра

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit.

4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. [4.21])

```

dkhuddiheva@dkhuddiheva-VirtualBox: ~/work/arch-pc/lab09$ touch lab9-3.asm
dkhuddiheva@dkhuddiheva-VirtualBox: ~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm

```

Рис. 4.20: Создание файла

```

lab9-2.asm lab9-1.asm lab9-3.asm lab9-2.asm lab9-2.o
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ ld -n elf_i386 -o lab9-3 lab9-3.o

```

Рис. 4.21: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `-args`. (рис. [4.22])

```

dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент2 'аргумент3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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 lab9-3...

```

Рис. 4.22: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. [4.23])

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/dkhuddiheva/work/arch-pc/lab09/lab9-3 аргумент1 аргумент2 аргумент3

Breakpoint 1, _start () at lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в ecx количество

```

Рис. 4.23: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис.[4.24])

```

(gdb) x/x $esp
0xfffffd140: 0x00000004
(gdb) x/s *(void**)( $esp + 4)
0xfffffd2f8: "/home/dkhuddiheva/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffd324: "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffd336: "аргумент2"
(gdb) x/s *(void**)( $esp + 16)
0xfffffd348: "аргумент3"
(gdb) x/s *(void**)( $esp + 20)
0x0: <error: Cannot access memory at address 0x0>

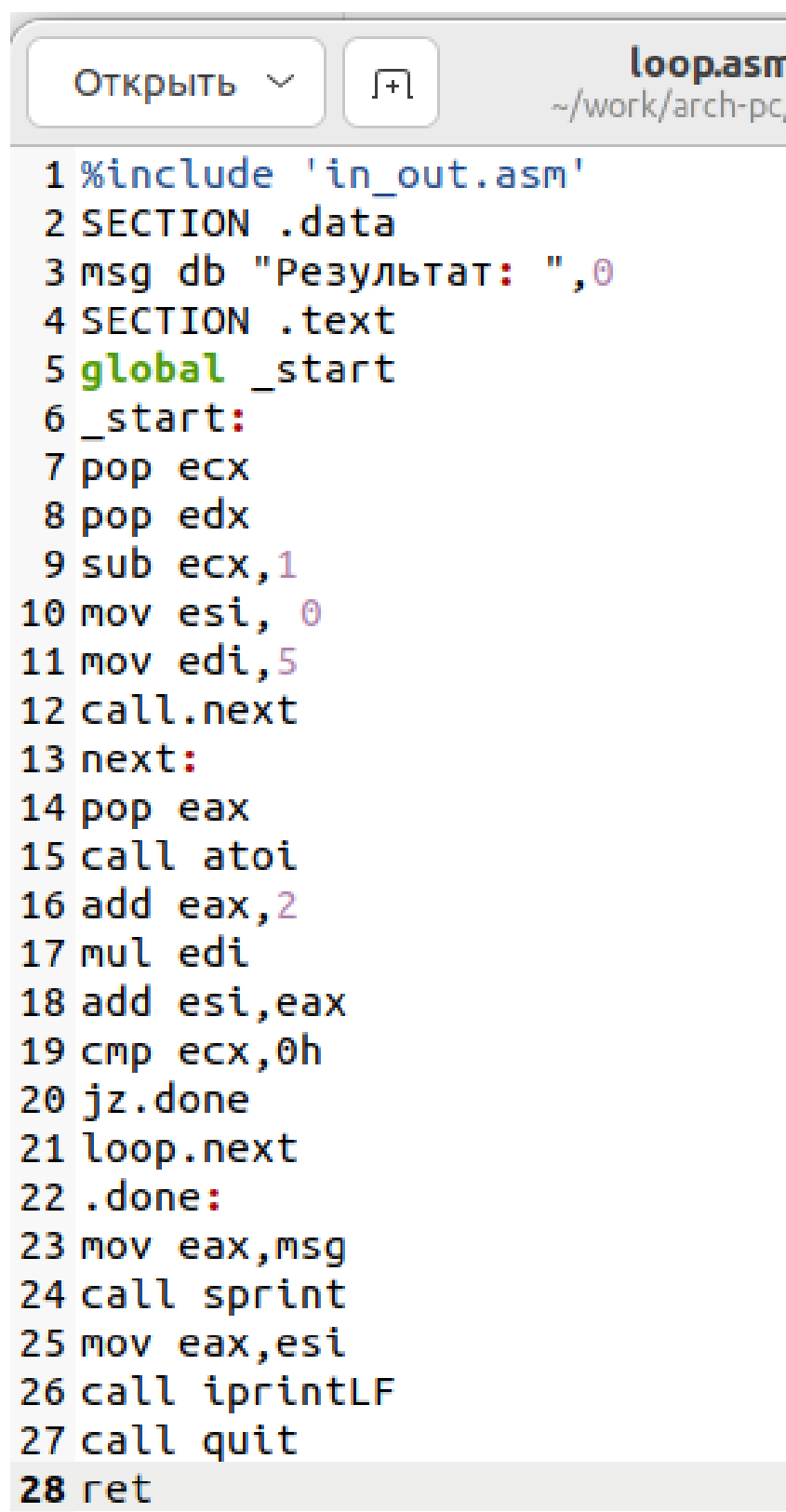
```

Рис. 4.24: Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

4.3 Задания для самостоятельной работы

1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. [4.25])



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx
9 sub ecx,1
10 mov esi, 0
11 mov edi,5
12 call.next
13 next:
14 pop eax
15 call atoi
16 add eax,2
17 mul edi
18 add esi,eax
19 cmp ecx,0h
20 jz.done
21 loop.next
22 .done:
23 mov eax,msg
24 call sprint
25 mov eax,esi
26 call iprintLF
27 call quit
28 ret
```

Рис. 4.25: Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно. (рис. [4.26])

```
dkhuddhevagdkhuddheva-VirtualBox: /work/arch-pc/lab08$ touch loop.asm
dkhuddhevagdkhuddheva-VirtualBox: /work/arch-pc/lab08$ cp ~/work/arch-pc/lab08/task1.asm ~/work/arch-pc/lab08/loop.asm
dkhuddhevagdkhuddheva-VirtualBox: /work/arch-pc/lab08$ gedit loop.asm
dkhuddhevagdkhuddheva-VirtualBox: /work/arch-pc/lab08$ nasm -f elf loop.asm
dkhuddhevagdkhuddheva-VirtualBox: /work/arch-pc/lab08$ ld -m elf_i386 -o loop loop.o
dkhuddhevagdkhuddheva-VirtualBox: /work/arch-pc/lab08$ ./loop 1 2 3
Результат: 30
```

Рис. 4.26: Запуск программы и проверка его вывода

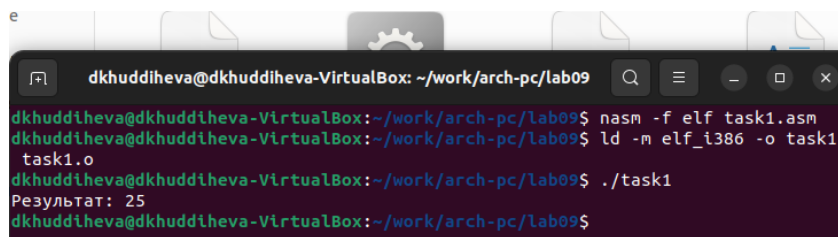
2. Ввожу в файл task1.asm текст программы из листинга 9.3. (рис. [??])

```
1 %include
2 'in_out.asm'
3 SECTION .data
4 div:
5 DB 'Результат: ',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 ; ---- Вычисление выражения (3+2)*4+5
10 mov ebx,3
11 mov eax,2
12 add ebx,eax
13 mov ecx,4
14 mul ecx
15 add ebx,5
16 mov edi,ebx
17 ; ----
18 mov
19 Вывод результата на экран
20 eax,div
21 call sprint
22 mov
23 eax,edi
24 call iprintLF
25 call quit
```

Рис. 4.27: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняе-

мый файл и запускаю его. (рис. [4.28])



```
dkhuddiheva@dkhuddiheva-VirtualBox: ~/work/arch-pc/lab09
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf task1.asm
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o task1
task1.o
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$ ./task1
Результат: 25
dkhuddiheva@dkhuddiheva-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 4.28: Создание и запуск исполняемого файла

5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.