

Отчёт по лабораторной работе №9

дисциплина: архитектура компьютера

Терещенкова Маргарита Владимировна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
3.1	Понятие об отладке	6
3.2	Методы отладки	6
3.3	Основные возможности отладчика GDB	7
4	Выполнение лабораторной работы	9
4.1	Реализация подпрограмм в NASM	9
4.2	Отладка программ с помощью GDB	11
4.3	Добавление точек останова	15
4.4	Работа с данными программы в GDB	16
4.5	Обработка аргументов командной строки в GDB	18
5	Задания для самостоятельной работы	21
6	Выводы	28
	Список литературы	29

Список иллюстраций

4.1	Создание файла	9
4.2	Редактирование файла	10
4.3	Запуск файла	11
4.4	Редактирование файла	12
4.5	Запуск файла	13
4.6	Брейкпоинт на метку <code>_start</code>	13
4.7	Дисассимилированный код программы	14
4.8	Код программы	14
4.9	Режим псевдографики	15
4.10	Проверка точки останова	15
4.11	Установка и проверка точки останова	15
4.12	Просмотр содержимого регистров	16
4.13	Просмотр значения переменной <code>msg1</code> и <code>msg2</code>	16
4.14	Изменение первого символа с помощью команды <code>set</code>	16
4.15	Изменение первого символа с помощью команды <code>set</code>	17
4.16	Различные форматы значения регистра <code>edx</code>	17
4.17	Изменение значения регистра <code>ebx</code>	17
4.18	Завершение выполнения программы и выход из GDB	18
4.19	Копирование файла	18
4.20	Создание исполняемого файла	19
4.21	Установка точки останова	19
4.22	Количество аргументов командной строки	19
4.23	Позиции стека	20
5.1	Редактирование файла	22
5.2	Запуск исполняемого файла	23
5.3	Копирование файла из листинга 9.3	25
5.4	Просмотр изменений регистров	26
5.5	Проверка работы программы	26

1 Цель работы

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

2 Задание

1. Релизация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Задание для самостоятельной работы.

3 Теоретическое введение

3.1 Понятие об отладке

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.

3.2 Методы отладки

Наиболее часто применяют следующие методы отладки:

- создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);
- использование специальных программ-отладчиков. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия.

Точки останова — это специально отмеченные места в программе, в которых

программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:

- **Breakpoint** — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);

- **Watchpoint** — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

3.3 Основные возможности отладчика GDB

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.

GDB может выполнять следующие действия:

- начать выполнение программы, задав всё, что может повлиять на её поведение;
- остановить программу при указанных условиях;

- исследовать, что случилось, когда программа остановилась;
- изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

4 Выполнение лабораторной работы

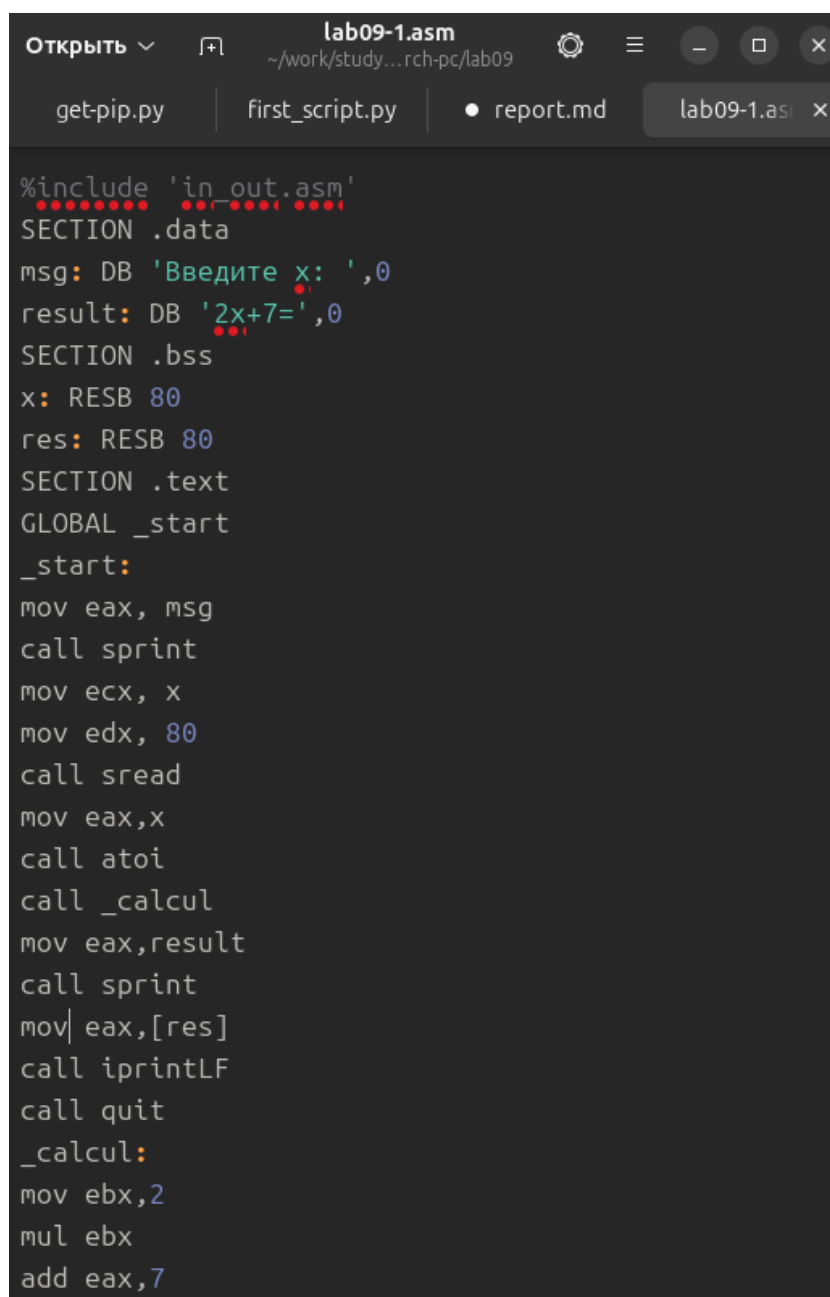
4.1 Реализация подпрограмм в NASM

1. Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm:

```
mytereshenkova@margo-pc:~$ mkdir ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab09
mytereshenkova@margo-pc:~$ cd ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab09
mytereshenkova@margo-pc:~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab09$ touch lab09-1.asm
```

Рис. 4.1: Создание файла

2. Ввожу в файл lab09-1.asm текст программы из листинга 9.1.



```
Открыть ▾  lab09-1.asm  ~/work/study...rch-pc/lab09  get-pip.py  first_script.py  ● report.md  lab09-1.asm x

%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:
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
_calcul:
mov ebx,2
mul ebx
add eax,7
```

Рис. 4.2: Редактирование файла

Создаю исполняемый файл и запускаю его.

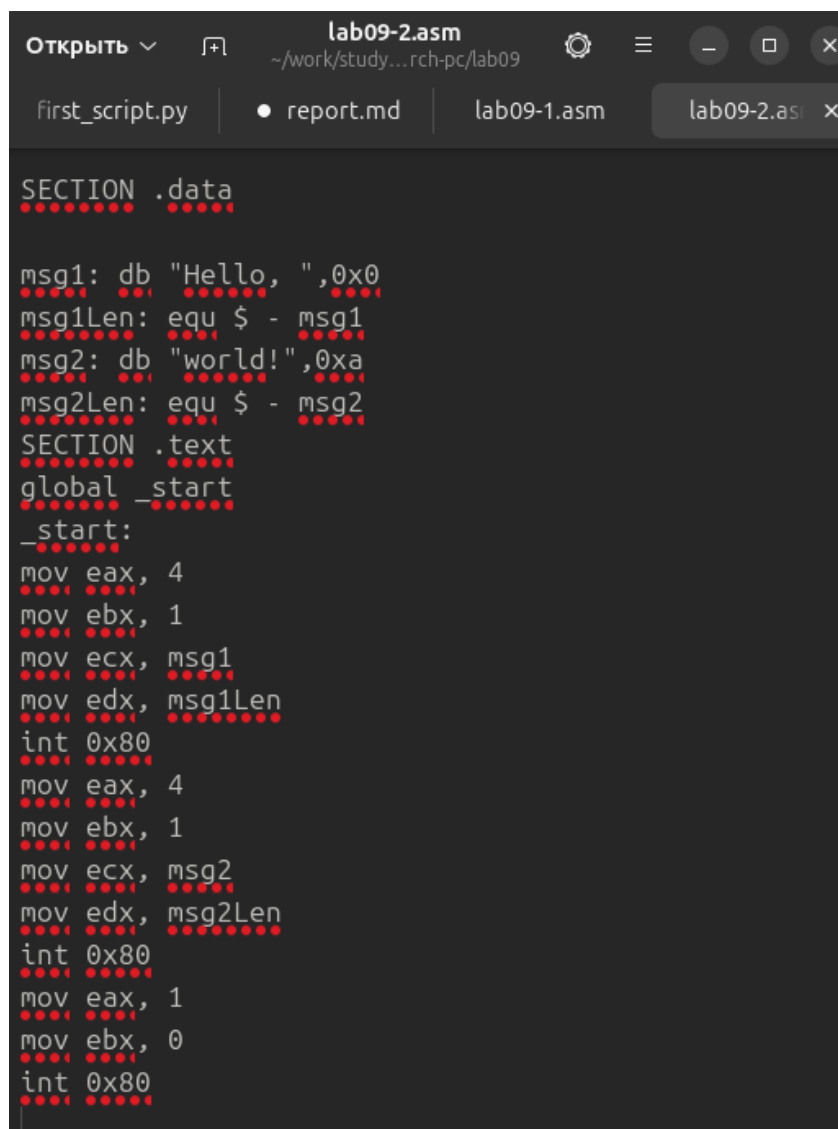
```
mytereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ nasm -f elf lab09-1.asm
mytereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ ld -m elf_i386 -o lab09-1 lab09-1.o
mytereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ ./lab09-1
Введите x: 9
2x+7=25
```

Рис. 4.3: Запуск файла

Программа работает корректно.

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2.



```
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. 4.4: Редактирование файла

Загружаю исполняемый файл в отладчик gdb. Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run.

```

mvtereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ touch lab09-2.asm
mvtereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2.asm
mvtereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
mvtereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ ld -m elf_i386 -o lab09-2 lab09-2.o
mvtereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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 lab09-2...
(gdb) run
Starting program: /home/mvtereshenkova/work/study/2024-2025/Архитектура компьюте
pa/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
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 system-supplied DSO at 0xf7ffc000
Hello, world!

```

Рис. 4.5: Запуск файла

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаю её.

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 10.
(gdb) run
Starting program: /home/mvtereshenkova/work/study/2024-2025/Архитектура компьюте
pa/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:10
10      mov eax, 4
(gdb)

```

Рис. 4.6: Брейкпоинт на метку `_start`

Посмотрела дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`.

```
(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. 4.7: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`.

```
(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. 4.8: Код программы

Включаю режим псевдографики для более удобного анализа программы.

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf60 0xffffcf60
ebp      0x0      0x0

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

native process 16146 (asm) In: _start          L10  PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.9: Режим псевдографики

4.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверяю это с помощью команды `info breakpoints` (кратко `i b`).

```

(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y   0x08049000 lab09-2.asm:10
       breakpoint already hit 1 time

```

Рис. 4.10: Проверка точки останова

Точка установлена.

Установила еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определяю адрес предпоследней инструкции (`mov ebx,0x0`) и устанавливаю точку останова. Посмотрела информацию о всех установленных точках останова.

```

(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 21.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y   0x08049000 lab09-2.asm:10
2      breakpoint      keep y   0x08049031 lab09-2.asm:21

```

Рис. 4.11: Установка и проверка точки останова

4.4 Работа с данными программы в GDB

Посмотрела содержимое регистров с помощью команды `info registers`.

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf60 0xffffcf60
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
b+ 0x8049031 <_start+49> mov     ebx,0x0

native process 2993 (asm) In: _start L10 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf60 0xffffcf60
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.12: Просмотр содержимого регистров

Посмотрела значение переменной `msg1` по имени.

```
The program has no registers now.
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n"
00f>
```

Рис. 4.13: Просмотр значения переменной `msg1` и `msg2`

Изменила значение для регистра или ячейки памяти с помощью команды `set`, задав ей в качестве аргумента имя регистра или адрес. Изменила первый символ переменной `msg1` “H” на “h”.

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

Рис. 4.14: Изменение первого символа с помощью команды `set`

Заменяла первый символ во второй переменной msg2 (“w” на “g”).

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

Рис. 4.15: Изменение первого символа с помощью команды set

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.

- /x - шестнадцатеричная
- /t - двоичная
- /c - символьный

```
(gdb) print /x $edx
$1 = 0x0
(gdb) print /t $edx
$2 = 0
(gdb) print /c $edx
$3 = 0 '\000'
```

Рис. 4.16: Различные форматы значения регистра edx

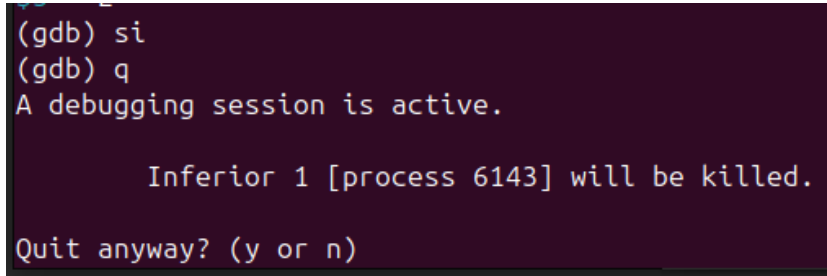
С помощью команды set изменила значение регистра ebx.

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

Рис. 4.17: Изменение значения регистра ebx

Разница в том, что при установке $ebx = '2' * *, ASCII - ., GDBASCII - ., *$ ***ebx=2**, значение напрямую интерпретируется как число, которое GDB не может отобразить как строку, и просто возвращает его.

Завершаю выполнение программы с помощью команды `stepi` (сокращенно `si`) и вышла из GDB с помощью команды `quit` (сокращенно `q`).



```
(gdb) si
(gdb) q
A debugging session is active.

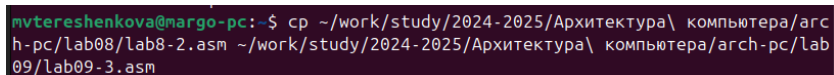
        Inferior 1 [process 6143] will be killed.

Quit anyway? (y or n)
```

Рис. 4.18: Завершение выполнения программы и выход из GDB

4.5 Обработка аргументов командной строки в GDB

Скопировала файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем `lab09-3.asm`.



```
nvtereshenkova@margo-pc: ~$ cp ~/work/study/2024-2025/Архитектура\ компьютера/arch-h-pc/lab08/lab8-2.asm ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab09/lab09-3.asm
```

Рис. 4.19: Копирование файла

Создаю исполняемый файл.

```

mvtereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
mvtereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ ld -m elf_i386 -o lab09-3 lab09-3.o
mvtereshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/la
b09$ gdb --args lab09-3 5 2 '5'
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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 lab09-3...

```

Рис. 4.20: Создание исполняемого файла

Для начала установила точку останова перед первой инструкцией в программе и запустим её.

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/mvtereshenkova/work/study/2024-2025/Архитектура компьюте
pa/arch-pc/lab09/lab09-3 5 2 5

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
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 lab09-3.asm:7
7      pop ecx

```

Рис. 4.21: Установление точки останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки.

```

(gdb) x/x $esp
0xffffcf50:      0x00000004

```

Рис. 4.22: Количество аргументов командной строки

Посмотрела 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.

```
(gdb) x/s *(void**) ($esp + 4)
0xfffffd110:    "/home/mvtereshenkova/work/study/2024-2025/Архитектура компьютер
a/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**) ($esp + 8)
0xfffffd17c:    "5"
(gdb) x/s *(void**) ($esp + 12)
0xfffffd17e:    "2"
(gdb) x/s *(void**) ($esp + 16)
0xfffffd180:    "5"
(gdb) x/s *(void**) ($esp + 20)
0x0:    <error: Cannot access memory at address 0x0>
```

Рис. 4.23: Позиции стека

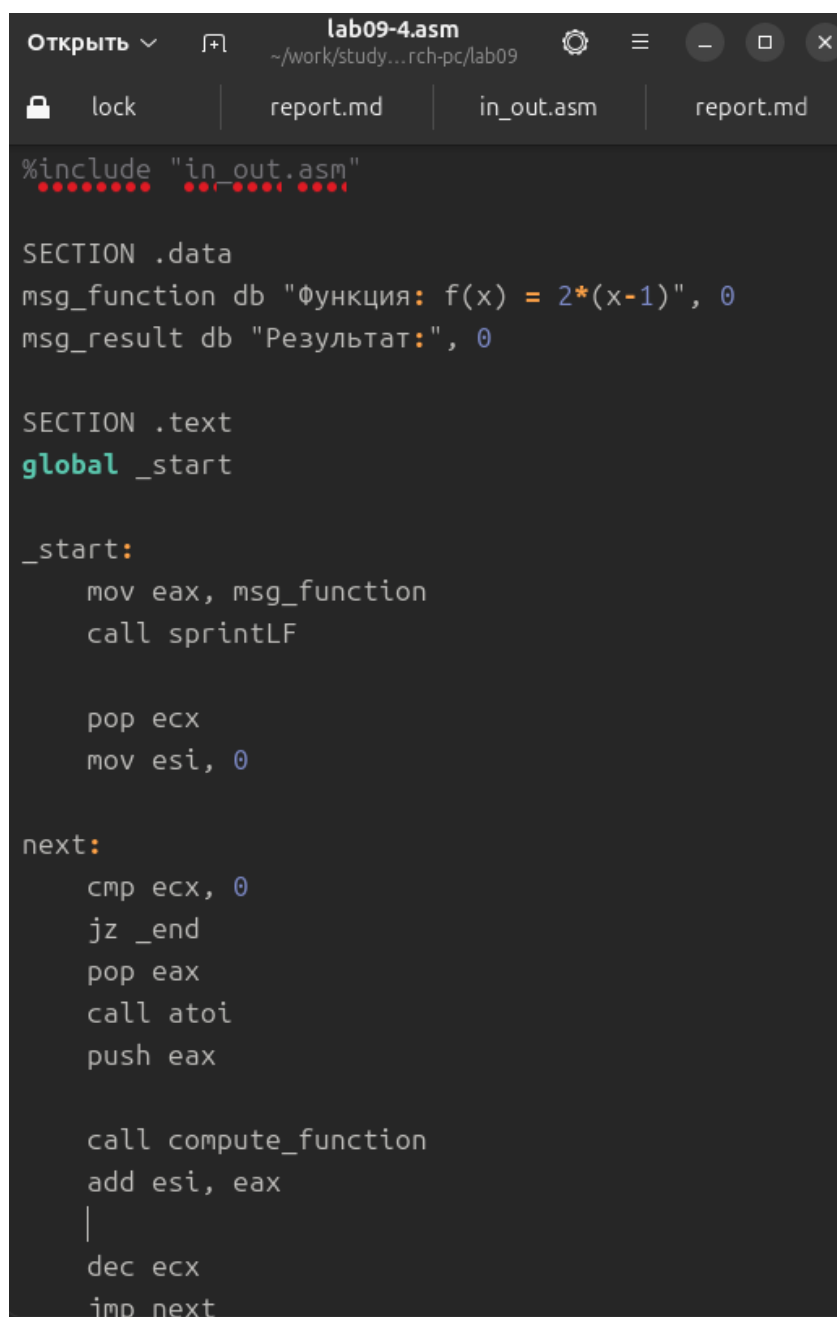
Шаг изменения адреса равен 4, потому что:

- Процессор обрабатывает данные 32-битными словами (4 байта).
- Аргументы на стеке выравниваются по границе 4 байт.
- Это упрощает доступ и обеспечивает корректность выполнения программы

на архитектуре x86.

5 Задания для самостоятельной работы

1. Открываю программу из лабораторной работы №8 и начинаю её редактировать.



```
lab09-4.asm
~/work/study...rch-pc/lab09

lock | report.md | in_out.asm | report.md

%include "in_out.asm"

SECTION .data
msg_function db "Функция: f(x) = 2*(x-1)", 0
msg_result db "Результат:", 0

SECTION .text
global _start

_start:
    mov eax, msg_function
    call sprintf

    pop ecx
    mov esi, 0

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

    call compute_function
    add esi, eax
    |
    dec ecx
    jmp next
```

Рис. 5.1: Редактирование файла

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы.

```

mvtereshenkova@margo-pc: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
b09$ nasm -f elf lab09-4.asm
mvtereshenkova@margo-pc: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
b09$ ld -m elf_i386 -o lab09-4 lab09-4.o
mvtereshenkova@margo-pc: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
b09$ ./lab09-4 6 8 7
Функция: f(x) = 2*(x-1)
Результат:34
mvtereshenkova@margo-pc: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09

```

Рис. 5.2: Запуск исполняемого файла

Код программы:

```
%include "in_out.asm"
```

```
SECTION .data
```

```
msg_function db "Функция: f(x) = 2*(x-1)", 0 msg_result db "Результат:", 0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
mov eax, msg_function
```

```
call sprintfLF
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0
```

```
next:
```

```
cmp ecx, 0
```

```
jz _end
```

```
pop eax
```

```
call atoi
```

```
call _calculate_fx
```

```
add esi, eax
```

```
loop next
```

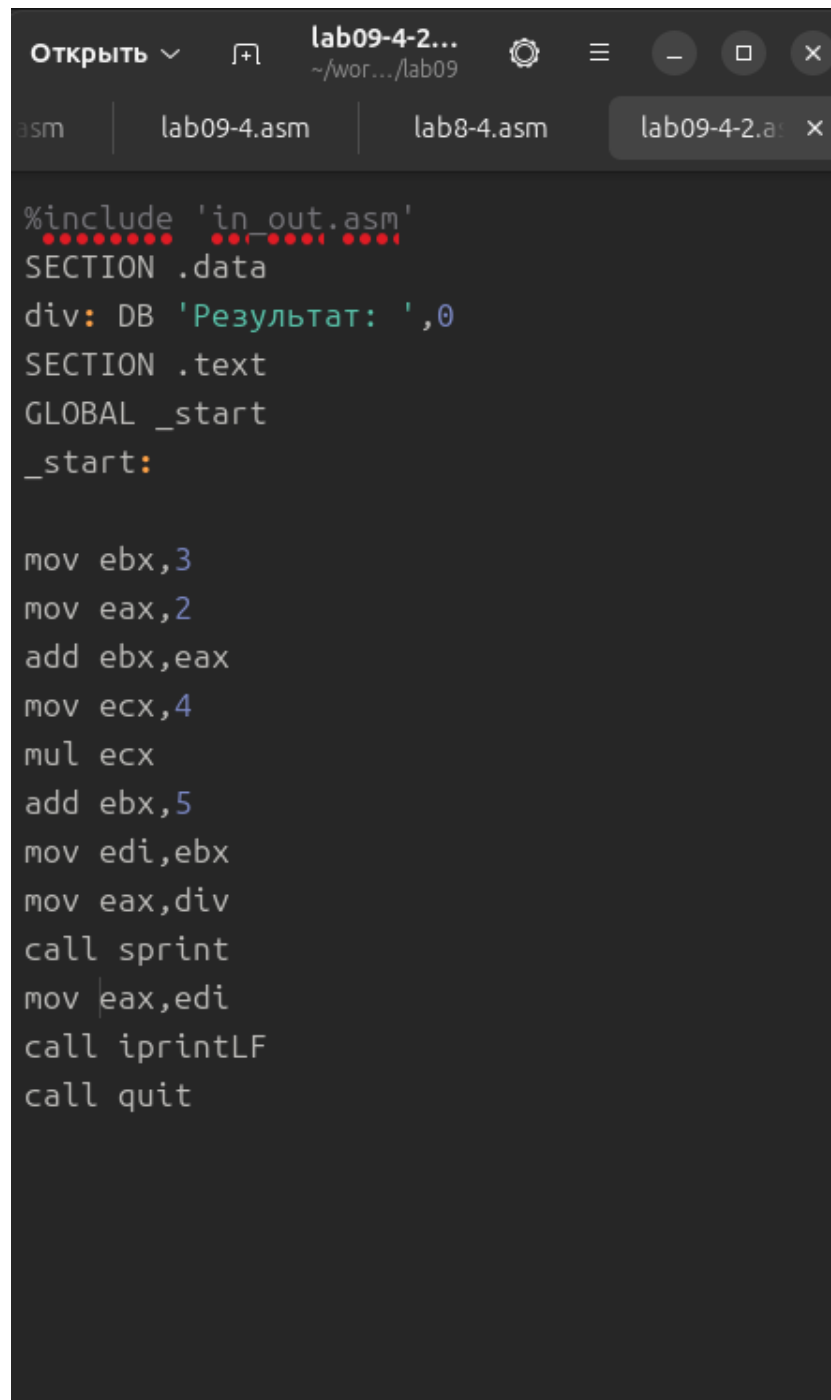
```
_end:
```

```
mov eax, msg_result
```

```
call sprint
```

```
mov eax, esi
call iprintLF
call quit
_calculate_fx:
sub eax, 1
mov ecx, 2
mul ecx
ret
```

2. Копирую код программы из листинга 9.3



```
Открыть ▾  lab09-4-2...  ~/\wog.../lab09  ⚙  ≡  -  □  ×  
asm  |  lab09-4.asm  |  lab8-4.asm  |  lab09-4-2.as  ×  
  
%include 'in_out.asm'  
SECTION .data  
div: DB 'Результат: ',0  
SECTION .text  
GLOBAL _start  
_start:  
  
mov ebx,3  
mov eax,2  
add ebx,eax  
mov ecx,4  
mul ecx  
add ebx,5  
mov edi,ebx  
mov eax,div  
call sprint  
mov eax,edi  
call iprintLF  
call quit
```

Рис. 5.3: Копирование файла из листинга 9.3

Запускаю программу в режиме отладчика и пошагово через si просматриваю изменение значений регистров через i r.

```

Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab09-4-2.asm:8
8      mov ebx,3
(gdb) info registers
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffcf40      0xffffcf40
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x80490e8        0x80490e8 <_start>
eflags         0x202          [ IF ]
cs             0x23            35
ss             0x2b            43
ds             0x2b            43
es             0x2b            43
fs             0x0            0
gs             0x0            0
(gdb)

```

Рис. 5.4: Просмотр изменений регистров

При выполнении инструкции `mul ecx` можно заметить, что результат умножения записывается в регистр `eax`, но также меняет и `edx`. Значение регистра `ebx` не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию.

Исправляю найденную ошибку, теперь программа верно считает значение функции.

```

mvtreshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf lab09-4-2.asm
mvtreshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-4-2 lab09-4-2.o
mvtreshenkova@margo-pc:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./lab09-4-2
Результат: 25

```

Рис. 5.5: Проверка работы программы

Код измененной программы:

```

#include 'in_out.asm'

SECTION .data
div: DB 'Результат:', 0

SECTION .text
GLOBAL _start

```

```
_start:
mov ebx, 3
mov eax, 2
add ebx, eax
mov eax, ebx
mov ecx, 4
mul ecx
add eax, 5
mov edi, eax
mov eax, div
call sprint
mov eax, edi
call iprintLF
call quit
```

6 Выводы

Благодаря данной лабораторной работе приобрела навыки написания программ с использованием подпрограмм; Познакомились с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. Архитектура компьютеров