

# **Отчёт по лабораторной работе №9**

**Дисциплина: архитектура компьютеров**

**Чувакина Мария Владимировна**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Реализация подпрограмм в NASM .....	9
4.2	Отладка программ с помощью GDB. ....	11
4.2.1	Добавление точек останова .....	14
4.2.2	Работа с данными программы в GDB .....	14
4.2.3	Обработка аргументов командной строки в GDB . . . . .	18
4.3	Задания для самостоятельной работы .....	19
<b>5</b>	<b>Выводы</b>	<b>26</b>
<b>6</b>	<b>Список литературы</b>	<b>27</b>

# Список иллюстраций

4.1	Создание файлов для лабораторной работы . . . . .	9
4.2	Ввод текста программы из листинга 9.1. . . . .	9
4.3	Запуск исполняемого файла.....	10
4.4	Изменение текста программы согласно заданию . . . . .	10
4.5	Запуск исполняемого файла.....	10
4.6	Ввод текста программы из листинга 9.2. . . . .	11
4.7	Получение исполняемого файла . . . . .	11
4.8	Загрузка исполняемого файла в отладчик . . . . .	12
4.9	Проверка работы файла с помощью команды run . . . . .	12
4.10	Установка брейкпоинта и запуск программы . . . . .	12
4.11	Использование команд disassemble и disassembly-flavor intel . . . . .	13
4.12	Включение режима псевдографики.....	13
4.13	Установка точек останова и просмотр информации о них . . . . .	14
4.14	До использования команды stepi . . . . .	15
4.15	После использования команды stepi . . . . .	15
4.16	Просмотр значений переменных . . . . .	16
4.17	Использование команды set.....	16
4.18	Вывод значения регистра в разных представлениях . . . . .	17
4.19	Использование команды set для изменения значения регистра . . . . .	17
4.20	Завершение работы GDB.....	18
4.21	Создание файла . . . . .	18
4.22	Загрузка файла с аргументами в отладчик . . . . .	18
4.23	Установка точки останова и запуск программы.....	19
4.24	Просмотр значений, введенных в стек . . . . .	19
4.25	Написание кода подпрограммы.....	20
4.26	Запуск программы и проверка его вывода . . . . .	20
4.27	Ввод текста программы из листинга 9.3. . . . .	22
4.28	Создание и запуск исполняемого файла . . . . .	22
4.29	Нахождение причины ошибки . . . . .	23
4.30	Неверное изменение регистра . . . . .	23
4.31	Исправление ошибки . . . . .	24
4.32	Ошибка исправлена . . . . .	24

# 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. Отладка программ с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы 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 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.

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` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

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

Для выхода из отладчика используется команда `quit` (или сокращённо `q`). Если есть файл с исходным текстом программы, а в исполняемый файл вклю-

чена 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` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`.

Обратно точка останова активируется командой `enable`.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`.

Для продолжения остановленной программы используется команда `continue` (`c`). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке).

Команда `stepi` (кратко `sI`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

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 позволит 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 lab09-1.asm. (рис. 4.1)

```
mvchuvakina@dk8n52 ~ $ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arh-pc/lab09
mvchuvakina@dk8n52 ~ $ cd work/study/2023-2024/Архитектура\ компьютера/arh-pc/lab09
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ touch lab09-1.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ gedit lab09-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. 4.2)

```
1 %include 'in_out.asm'
2
3 SECTION .data
4     msg: DB 'Введите x: ',0
5     result: DB '2x+7=',0
6
7 SECTION .bss
8     x: RESB 80
9     res: RESB 80
10
11 SECTION .text
12 GLOBAL _start
13     _start:
14
15 ;-----
16 ; Основная программа
17 ;-----
18
19 mov eax, msg
20 call sprint
21
22 mov ecx, x
23 mov edx, 80
24 call sread
25
26 mov eax, x
27 call atoi
28
29 call _calcul ; Вызов подпрограммы _calcul
30
31 mov eax, result
32 call sprint
33 mov eax, [res]
34 call iprintLF
35
36 call quit
37
38 ;-----
39 ; Подпрограмма вычисления
40 ; выражения "2x+7"
41
42 _calcul:
43     mov ebx, 2
44     mul ebx
45     add eax, 7
46
47     mov [res], eax
```

Рис. 4.2: Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу. (рис. 4.3)

```
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ nasm -f elf lab09-1.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ./lab09-1
Введите x: 7
2x+7=21
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $
```

Рис. 4.3: 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. 4.4)

```
6
7 SECTION .bss
8     x: RESB 80
9     res: RESB 80
10
11 SECTION .text
12 GLOBAL _start
13     _start:
14
15 -----
16 ; Основная программа
17 ; -----
18
19 mov eax, msg
20 call sprint
21
22 mov ecx, x
23 mov edx, 80
24 call sread
25
26 mov eax, x
27 call atoi
28
29 call _subcalcul ; Вызов подпрограммы _calcul
30 call _calcul
31
32 mov eax, result
33 call sprint
34 mov eax, [res]
35 call iprintLF
36
37 call quit
38
39 -----
40 ; Подпрограмма вычисления
41 ; выражения "2x+7"
42
43 _calcul:
44     mov ebx, 2
45     mul ebx
46     add eax, 7
47
48     mov [res], eax
49
50     ret ; выход из подпрограммы
51
```

Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.5)

```
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ nasm -f elf lab09-1.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ./lab09-1
Введите x: 7
2x+7=47
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $
```

Рис. 4.5: Запуск исполняемого файла

## 4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. 4.6)

```
1 SECTION .data
2     msg1: db "Hello, ",0x0
3     msg1Len: equ $ - msg1
4
5     msg2: db "world!",0xa
6     msg2Len: equ $ - msg2
7
8 SECTION .text
9     global _start
10
11 _start:
12     mov eax, 4
13     mov ebx, 1
14     mov ecx, msg1
15     mov edx, msg1Len
16     int 0x80
17
18     mov eax, 4
19     mov ebx, 1
20     mov ecx, msg2
21     mov edx, msg2Len
22     int 0x80
23
24     mov eax, 1
25     mov ebx, 0
26     int 0x80
```

Рис. 4.6: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. 4.7)

```
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ touch lab09-2.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ gedit lab09-2.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ nasm -f elf lab09-2.asm
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $
```

Рис. 4.7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. 4.8)

```

mvchuvakina@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 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-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 lab09-2...
(no debugging symbols found in lab09-2)
(gdb)

```

Рис. 4.8: Загрузка исполняемого файла в отладчик

Проверяю 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/m/v/mvchuvakina/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3930) exited normally]
(gdb)

```

Рис. 4.9: Проверка работы файла с помощью команды `run`

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start` и запускаю ее. (рис. 4.10)

```

(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb)

```

Рис. 4.10: 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)

```

Breakpoint 2 at 0x8049000
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/v/mvchuvakina/work/study/2023-2024/Архитектура компьютера/arih-pc/lab09/lab09-2

Breakpoint 1, 0x8049000 in _start ()
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x8049000 <+0>: mov $0x4,%eax
0x8049005 <+5>: mov $0x1,%ebx
0x804900a <+10>: mov $0x804a000,%ecx
0x804900f <+15>: mov $0x8,%edx
0x8049014 <+20>: int $0x80
0x8049016 <+22>: mov $0x4,%eax
0x804901b <+27>: mov $0x1,%ebx
0x8049020 <+32>: mov $0x804a008,%ecx
0x8049025 <+37>: mov $0x7,%edx
0x804902a <+42>: int $0x80
0x804902c <+44>: mov $0x1,%eax
0x8049031 <+49>: mov $0x0,%ebx
0x8049036 <+54>: int $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Undefined command: "disassemble". Try "help".
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x8049000 <+0>: mov eax,0x4
0x8049005 <+5>: mov ebx,0x1
0x804900a <+10>: mov ecx,0x804a000
0x804900f <+15>: mov edx,0x8
0x8049014 <+20>: int 0x80
0x8049016 <+22>: mov eax,0x4
0x804901b <+27>: mov ebx,0x1
0x8049020 <+32>: mov ecx,0x804a008
0x8049025 <+37>: mov edx,0x7
0x804902a <+42>: int 0x80
0x804902c <+44>: mov eax,0x1
0x8049031 <+49>: mov ebx,0x0
0x8049036 <+54>: int 0x80
End of assembler dump.
(gdb)

```

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 ]

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

native process 3980 In: _start
(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 ]
```

```
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
```

```
native process 3980 In: _start
(gdb) layout regs
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint        keep y  0x08049000 <_start>
      breakpoint already hit 1 time
2      breakpoint        keep y  0x08049000 <_start>
      breakpoint already hit 1 time
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint        keep y  0x08049000 <_start>
      breakpoint already hit 1 time
2      breakpoint        keep y  0x08049000 <_start>
      breakpoint already hit 1 time
(gdb)
```

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

### 4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. 4.14)

```

--Register group: general--
eax      0x4      4      ecx      0x0      0      edx      0x0      0
ebx      0x0      0      esp      0xffffc280  0xffffc280  ebp      0x0      0
esi      0x0      0      edi      0x0      0      eip      0x8049005  0x8049005 <_start+5>
eflags   0x202    [ IF ]  cs      0x23     35      ss      0x2b     43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

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

native process 3980 In: _start L77 PC: 0
eax      0x4      4      ecx      0x0      0      edx      0x0      0
ebx      0x0      0      esp      0xffffc280  0xffffc280  ebp      0x0      0
esi      0x0      0      edi      0x0      0      eip      0x8049005  0x8049005 <_start+5>
eflags   0x202    [ IF ]  cs      0x23     35      ss      0x2b     43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.14: До использования команды stepi

```

--Register group: general--
eax      0x4      4      ecx      0x804a000  134520832  edx      0x8      8
ebx      0x1      1      esp      0xffffc280  0xffffc280  ebp      0x0      0
esi      0x0      0      edi      0x0      0      eip      0x8049020  0x8049020 <_start+32>
eflags   0x202    [ IF ]  cs      0x23     35      ss      0x2b     43
ds       0x2b     43      es      0x2b     43      fs      0x0      0
gs       0x0      0

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

native process 3980 In: _start L77 PC: 0x8049020
edi      0x0      0
eip      0x8049005  0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
fs       0x0      0
gs       0x0      0
--Type <RET> for more, q to quit, c to continue without paging--ces      0x2b      43
(gdb) si
0x804900a in _start ()
(gdb) si 5
hello, 0x8049020 in _start ()
(gdb)

```

Рис. 4.15: После использования команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

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

```

Register group: general
eax 0x4 4 ecx 0x804a000 134520832 edx 0x8 8
ebx 0x1 1 esp 0xfffffc280 0xfffffc280 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0 eip 0x8049020 0x8049020 <_start+32>
eflags 0x202 [ IF ] cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43 fs 0x0 0
gs 0x0 0

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,0x804a000
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 3980 In: _start
ss 0x2b 43
ds 0x2b 43
--Type <RET> for more, q to quit, c to continue without paging--
fs 0x0 0
gs 0x0 0
(gdb) si
0x804900a in _start ()
(gdb) si 5
Hello, 0x8049020 in _start ()
(gdb) x/15b &msg1
0x804a000: "Hello, "
(gdb) x/15b 0x804a008
0x804a008: "world!\n"
(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)

```

(gdb) set (char)&msg1='h'
(gdb) x/15b &msg1
0x804a000: "Hello, "
(gdb) set (char)&msg2='b'
(gdb) x/15b &msg2
0x804a008: "borld!\n"
(gdb)

```

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      0x4      4      ecx      0x804a000      134520832      edx      0x8      8
ebx      0x1      1      esp      0xffffc280      0xffffc280      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x8049020      0x8049020 <_start+32>
eflags   0x202    0      cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43      fs       0x0      0
gs       0x0      0

B+ 0x8049000 <_start> mov     eax,0x4
0x8049005 <_start+5> mov     ebx,0x1
0x804900a <_start+10> mov     ecx,0x804a000
0x804900f <_start+15> mov     edi,0x0
0x8049014 <_start+20> int     0x80
0x8049016 <_start+22> mov     eax,0x4
0x804901b <_start+27> mov     ebx,0x1
> 0x8049020 <_start+32> mov     ecx,0x804a000
0x8049025 <_start+37> mov     edi,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 3980 In: _start
0x804a008: "world\n"
(gdb) set (char)&msg1='h'
(gdb) x/1sb &msg1
0x804a008: "hello, "
(gdb) set (char)&msg2='b'
(gdb) x/1sb &msg2
0x804a008: "borld\n"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb)

```

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      0x4      4      ecx      0x804a000      134520832      edx      0x8      8
ebx      0x2      2      esp      0xffffc280      0xffffc280      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x8049020      0x8049020 <_start+32>
eflags   0x202    0      cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43      fs       0x0      0
gs       0x0      0

B+ 0x8049000 <_start> mov     eax,0x4
0x8049005 <_start+5> mov     ebx,0x1
0x804900a <_start+10> mov     ecx,0x804a000
0x804900f <_start+15> mov     edi,0x8
0x8049014 <_start+20> int     0x80
0x8049016 <_start+22> mov     eax,0x4
0x804901b <_start+27> mov     ebx,0x1
> 0x8049020 <_start+32> mov     ecx,0x804a000
0x8049025 <_start+37> mov     edi,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov     eax,0x1
0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80

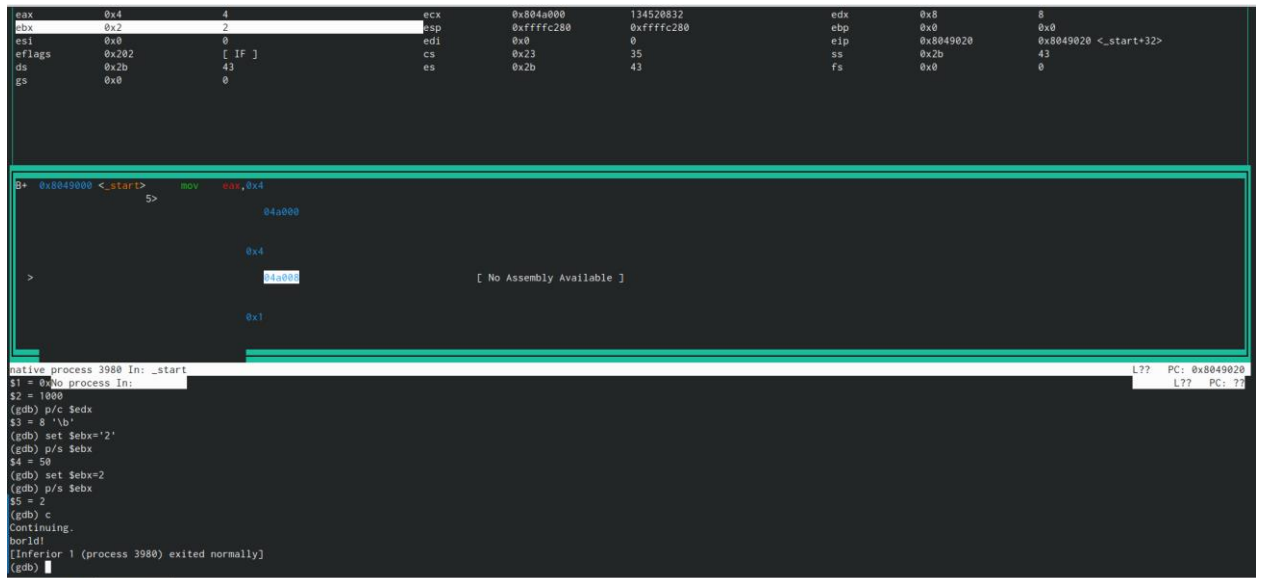
native process 3980 In: _start
0x804a008: "borld\n"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb) set $ebx="2"
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)

```

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. (рис. 4.20)



```
eax    0x4      4      ecx    0x804a000    134520832    edx    0x8      8
ebx    0x2      2      esp    0xffffc280    0xffffc280    ebp    0x0      0x0
esi    0x0      0      edi    0x0      0      eip    0x8049020    0x8049020 <_start+32>
eflags 0x202    [ IF ]   cs     0x23      35      ss     0x2b      43
ds      0x2b      43      es     0x2b      43      fs     0x0      0
gs      0x0      0

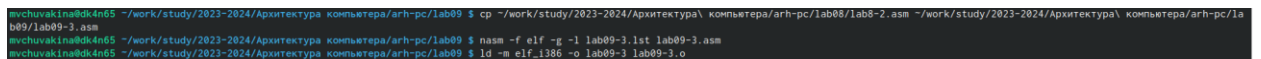
B* 0x8049000 <_start> mov     eax,0x4
S>
0x4
0x4
>
[ No Assembly Available ]
0x1

native process 3980 In: _start
$1 = 0x0 No process in:
$2 = 1800
(gdb) p/c $edx
$3 = 8 '\b'
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 58
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) c
Continuing.
borld!
[Inferior 1 (process 3980) exited normally]
(gdb)
```

Рис. 4.20: Завершение работы GDB

### 4.2.3 Обработка аргументов командной строки в GDB

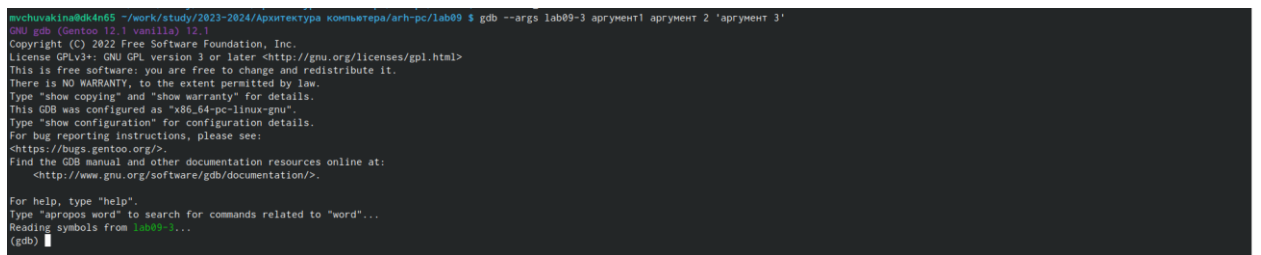
Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09- 3.asm и создаю исполняемый файл. (рис. 4.21)



```
michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ cp ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab08/lab8-2.asm ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09/lab09-3.asm
michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 4.21: Создание файла

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



```
michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 12.1 vanilla) 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-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 lab09-3...
(gdb)
```

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 0x00490e0: file lab09-3.asm, line 7.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/v/mvchuvakina/work/study/2023-2024/Архитектура компьютера/арх-пс/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:7
7      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)
```

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
0xffffc200: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffc49f: "/afs/.dk.sci.pfu.edu.ru/home/m/v/mvchuvakina/work/study/2023-2024/Архитектура компьютера/арх-пс/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc522: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffc534: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffc545: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffc547: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

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 .next:
2 pop eax
3 call atoi
4 mov ebx,15
5 mul ebx
6 sub eax,9
7 add esi,eax
8 mul edi
9 add esi,eax
10 cmp ecx,0h
11 jz.done
12 loop .next
13
14 .done:
15 mov eax, msg
16 call sprint
17 mov eax, esi
18 call iprintLF
19 call quit
20 ret

```

Рис. 4.25: Написание кода подпрограммы

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

```

michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/арх-пс/lab09 $ cp ~/work/study/2023-2024/Архитектура\ компьютера/арх-пс/lab08/task.asm ~/work/study/2023-2024/Архитектура\ компьютера/арх-пс/lab09/task1.asm
michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/арх-пс/lab09 $ nasm -f elf task1.asm
michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/арх-пс/lab09 $ ld -m elf_i386 -o task1 task1.o
michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/арх-пс/lab09 $ ./task1 1 2 3
Результат: 147
michuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/арх-пс/lab09 $

```

Рис. 4.26: Запуск программы и проверка его вывода

Код программы:

```

%include 'in_out.asm'
SECTION data
msg db "Результат:",0
SECTION.text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,5
next:
.next:

```

```
pop eax
call atoi
mov ebx,15
mul ebx
sub eax,9
add esi,eax
mul edi
add esi,eax
cmp ecx,0h
jz.done
loop .next
.done:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
ret
```

2. Ввожу в файл task1.asm текст программы из листинга 9.3. (рис. 4.27)

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 div: DB 'Результат: ',0
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 ; ---- Вычисление выражения (3+2)*4+5
11     mov ebx,3
12     mov eax,2
13     add ebx,eax
14     mov ecx,4
15     mul ecx
16     add ebx,5
17     mov edi,ebx
18
19 ; ---- Вывод результата на экран
20 mov eax,div
21 call sprint
22 mov eax,edi
23 call iprintLF
24
25 call quit
26

```

Рис. 4.27: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его. (рис. 4.28)

```

mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ touch task2.asm
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ gedit task2.asm
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ nasm -f elf task2.asm
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ld -m elf_i386 -o task2 task2.o
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ./task2
Результат: 10
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $

```

Рис. 4.28: Создание и запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ.

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды `continue` прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

При выполнении инструкции `mul ecx` происходит умножение `ecx` на `eax`, то есть 4 на 2, вместо умножения 4 на 5 (регистр `ebx`). Происходит это из-за того, что

стоящая перед `mov ecx,4` инструкция `add ebx,ecx` не связана с `mul ecx`, но связана инструкция `mov eax,2`. (рис. 4.29)

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f4 <_start+12> mov ecx,0x4
B+ 0x80490f9 <_start+17> mul ecx
B+> 0x80490fb <_start+19> add ebx,0x5
b+ 0x80490fe <_start+22> mov edi,ebx
   0x8049100 <_start+24> mov eax,0x804a000

native process 14573 In: _start L13 PC: 0x80490fb
Continuing.

Breakpoint 5, _start () at task2.asm:12
(gdb) c
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb) █
```

Рис. 4.29: Нахождение причины ошибки

Из-за этого мы получаем неправильный ответ. (рис. 4.30)

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f9 <_start+17> mul ecx
B+ 0x80490fb <_start+19> add ebx,0x5
B+> 0x80490fe <_start+22> mov edi,ebx
   0x8049100 <_start+24> mov eax,0x804a000
   0x8049105 <_start+29> call 0x804900f <sprint>

native process 14573 In: _start L14 PC: 0x80490fe
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb) c
Continuing.

Breakpoint 7, _start () at task2.asm:14
(gdb) █
```

Рис. 4.30: Неверное изменение регистра

Исправляем ошибку, добавляя после `add ebx, eax` `mov eax, ebx` и заменяя `ebx` на `eax` в инструкциях `add ebx, 5` и `mov edi, ebx`. (рис. 4.31)

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 div: DB 'Результат: ', 0
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 ; ---- Вычисление выражения (3+2)*4+5
11     mov ebx, 3
12     mov eax, 2
13     add ebx, eax
14     mov eax, ebx
15     mov ecx, 4
16     mul ecx
17     add eax, 5
18     mov edi, eax
19
20 ; ---- Вывод результата на экран
21 mov eax, div
22 call sprint
23 mov eax, edi
24 call iprintLF
25
26 call quit
27
```

Рис. 4.31: Исправление ошибки

Также, вместо того, чтобы изменять значение `eax`, можно было изменять значение неиспользованного регистра `edx`.

Создаем исполняемый файл и запускаем его. Убеждаемся, что ошибка исправлена. (рис. 4.32)

```
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ nasm -f elf task2.asm
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ld -m elf_i386 -o task2 task2.o
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $ ./task2
Результат: 25
mvchuvakina@dk4n65 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab09 $
```

Рис. 4.32: Ошибка исправлена

Код программы:

```
%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,edi
call sprint
mov eax,edi
call iprintLF
call quit
```

## 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.

## 6 Список литературы

1. GDB: The GNU Project Debugger.—URL:<https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual.—2016.—URL:<https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center.—2021.—URL:<https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials.—2021.—URL:<https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005 — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation.—2021.—URL:<https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В.Д., Лупин С.А. Архитектура ЭВМ.—М.: Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О.П. Архитектура ЭВМ и систем.—М.: Юрайт, 2016.
12. Расширенный ассемблер: NASM.—2021.—URL:<https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX.—2-е изд. — БХВ Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix.—2-е изд. — М. : МАКС Пресс, 2011. — URL: [http://www.stolyarov.info/books/asm\\_unix](http://www.stolyarov.info/books/asm_unix).
15. Таненбаум Э. Архитектура компьютера.—6-е изд.—СПб.: Питер, 2013.— 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).