

# **Отчёта по лабораторной работе №9**

**НПИ-03-23**

Махмудов Суннатилло Баходир угли

# 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>4.1 Задания для самостоятельной работы</b>	<b>15</b>
<b>6</b>	<b>Выводы</b>	<b>17</b>
	<b>Список литературы</b>	<b>18</b>

## Список иллюстраций

4.1	Создание каталога . . . . .	8
4.2	Программа . . . . .	8
4.3	Программа . . . . .	8
4.4	Отладчик . . . . .	9
4.5	Отладчик . . . . .	9
4.6	Отладчик . . . . .	9
4.7	Брейкпоинт . . . . .	9
4.8	Дисассимилированный код . . . . .	10
4.9	Отображение . . . . .	10
4.10	Псевдографика . . . . .	11
4.11	Точки останова . . . . .	11
4.12	Точки останова . . . . .	11
4.13	stepi . . . . .	12
4.14	Значения регистров . . . . .	12
4.15	Значение переменной . . . . .	12

## 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. Отладка программ с помощью GDB.
3. Работа с данными программы в GDB.
4. Обработка аргументов командной строки в GDB.
5. Преобразование программы из лабораторной работы №8, реализовав вычисление значения функции как подпрограмму.
7. Проверить неправильную работу программы, проанализировав изменения значения регистров. Определить ошибку и исправить ее

### 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. Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

## 4 Выполнение лабораторной работы

Создадим каталог и файл для лабораторной работы(рис. 4.15).

```
mmulitina@ubuntu:~$ mkdir ~/work/arch-pc/lab09
mmulitina@ubuntu:~$ cd ~/work/arch-pc/lab09
mmulitina@ubuntu:~/work/arch-pc/lab09$ touch lab09-1.asm
mmulitina@ubuntu:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание каталога

Введём текст программы и запустим её для проверки(рис. 4.15).

```
mmulitina@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
mmulitina@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
mmulitina@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
2x+7=11
```

Рис. 4.2: Программа

Добавим подпрограмму `_subcalcul`, запустим программу для проверки(рис. 4.15).

```
mmulitina@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
mmulitina@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
mmulitina@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 1
2x+7=11
mmulitina@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
2x+7=17
```

Рис. 4.3: Программа

Создадим файл `lab09-2.asm`, введём в него текст программы, получим исполняемый



файл и загрузим его в отладчик(рис. 4.15).

```
mmulittina@ubuntu:~/work/arch-pc/lab09$ touch lab09-2.asm
mmulittina@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
mmulittina@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
mmulittina@ubuntu:~/work/arch-pc/lab09$ gdb lab09-2
```

Рис. 4.4: Отладчик

Проверим работу программы, запустим ее в оболочке GDB(рис. 4.15).

```
(gdb) run
```

Рис. 4.5: Отладчик

(рис. 4.15).

```
Hello, world!
[Inferior 1 (process 71914) exited normally]
(gdb)
```

Рис. 4.6: Отладчик

Установим брейкпоинт и запустим программу(рис. 4.15).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/mmultipina/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 4.7: Брейкпоинт

Посмотрим дисассимилированный код (рис. 4.15).

```

(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. 4.8: Дисассимилированный код

Переключимся на отображение команд с Intel синтаксисом (рис. 4.15).

```

(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. 4.9: Отображение

Включим режим псевдографики (рис. 4.15).

```

[ Register Values Unavailable ]

0x80491a4      add     BYTE PTR [eax],al
0x80491a6      add     BYTE PTR [eax],al
0x80491a8      add     BYTE PTR [eax],al
0x80491aa      add     BYTE PTR [eax],al
0x80491ac      add     BYTE PTR [eax],al
0x80491ae      add     BYTE PTR [eax],al

native process 71924 In: _start          L9    PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.10: Псевдографика

Проверим точки останова с помощью команды `info breakpoints` (рис. 4.15).

```

(gdb) layout regs
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y  0x08049000  lab09-2.asm:9
         breakpoint already hit 1 time

```

Рис. 4.11: Точки останова

Установим ещё одну точку останова и снова посмотрим информацию о точках останова (рис. 4.15).

```

(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y  0x08049000  lab09-2.asm:9
         breakpoint already hit 1 time
2        breakpoint     keep y  0x08049031  lab09-2.asm:20
(gdb)

```

Рис. 4.12: Точки останова

Выполним 5 инструкций с помощью команды `stepi` и проследим изменения регистров(рис. 4.15).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd030 0xffffd030
ebp      0x0      0x0

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

native process 72040 In: _start L14 PC: 0x8049016
breakpoint already hit 1 time
2 breakpoint keep y 0x00049031 lab09-2.asm:20
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)

```

Рис. 4.13: stepi

Изменились значения регистров eax, ecx, ebx, edx.

Посмотрим значения регистров с помощью info registers(рис. 4.15).

```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd030 0xffffd030
ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.14: Значения регистров

Посмотрим значение переменной msg1 по имени(рис. 4.15).

```

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

```

Рис. 4.15: Значение переменной

Изменим первый символ переменной (рис. 4.15).

```

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

```

Изменим любой символ второй переменной msg2 (рис. 4.15).

```
(gdb) set {char}&msg2 = 'a'
(gdb) x/lsb &msg2
0x804a008 <msg2>:      "aorld!\n\034"  {#fig:001 width=70%
```

Посмотрим значений регистра edx (рис. 4.15).

```
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'  {#fig:001 width=70%
```

С помощью set изменим значение регистра ebx (рис. 4.15).

```
(gdb) set $ebx = '2'  {#fig:001 width=70%
```

Проверим его значение(рис. 4.15).

```
(gdb) p/s $ebx
$4 = 50  {#fig:001 width=70%
```

Снова изменим значение ebx(рис. 4.15).

```
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2  {#fig:001 width=70%
```

В первом случае мы ввели символьное значение, во втором цифру.

Скопируем файл из прошлой лабораторной работы(рис. 4.15).

```
mmulitina@ubuntu:~$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm  {#fig:001
width=70%
```

Создадим исполняемый файл (рис. 4.15).

```
mmulitina@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
mmulitina@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o  {#fig:001
width=70%
```

Загрузим исполняемый файл в отладчик, указав аргументы(рис. 4.15).

```
mmulitina@ubuntu:~/work/arch-pc/lab09$ gdb --args lab09-3 arg1 arg 2 'arg3'  {#fig:001
width=70%
```

Установим точку останова перед первой инструкцией в программе и запустим

её(рис. 4.15).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
```

{#fig:001

width=70%

Посмотрим значение регистра esp, где хранится адрес вершины стека(рис. 4.15).

```
(gdb) x/x $esp
0xffffd010: 0x00000005
```

{#fig:001 width=70%

Посмотрим остальные позиции стека по адресу (рис. 4.15).

```
(gdb) x/s *(void**)(esp + 4)
0xffffd1f4: "/home/mmultipina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd21f: "arg1"
(gdb) x/s *(void**)(esp + 12)
0xffffd224: "arg"
(gdb) x/s *(void**)(esp + 16)
0xffffd228: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd22a: "arg3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

{#fig:001

width=70%

Шаг изменения равен 4, т.к. у нас 4 аргумента.(рис. 4.15).

```
_calcul:
mov edx,10
mul edx
sub eax,5
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
ret
```

{#fig:001

width=70%

Шаг изменения равен 4, т.к. у нас 4 аргумента.

## 5 4.1 Задания для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8, реализовав вычисление значения функции как подпрограмму (рис. 4.15).

```
mmulitina@ubuntu:~/work/arch-pc/lab09$ ./prog1 1 2 3 4
Результат: 80
```

{#fig:001

width=70%

Запустим программу для проверки(рис. 4.15).

```
mmulitina@ubuntu:~/work/arch-pc/lab09$ touch lab09-4.asm
mmulitina@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-4.lst lab09-4.asm
mmulitina@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
mmulitina@ubuntu:~/work/arch-pc/lab09$ gdb lab09-4
```

{#fig:001

width=70%

2. Создадим файл для программы, введём в него текст программы, запустим его в отладчике GDB (рис. 4.15).

eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5

{#fig:001 width=70%

При умножении с помощью mul, мы умножаем eax на ecx и записываем в eax.

Получаем 24=9 вместо (3+2)4(рис. 4.15).

```
mmulitina@ubuntu:~/work/arch-pc/lab09$ ./lab09-4
Результат: 10
```

{#fig:001

width=70%

Потом складываем с регистром ebx 5 и получаем 10. Проверим это, запустив программу(рис. 4.15).

```
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
```

{#fig:001 width=70%

Исправим программу(рис. 4.15).

```
mmulitina@ubuntu:~$ mkdir ~/work/arch-pc/lab09
mmulitina@ubuntu:~$ cd ~/work/arch-pc/lab09
mmulitina@ubuntu:~/work/arch-pc/lab09$ touch lab09-1.asm
mmulitina@ubuntu:~/work/arch-pc/lab09$
```

{#fig:001

width=70%

Запустим её для проверки (рис. 4.15).

```
mmulitina@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
mmulitina@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
mmulitina@ubuntu:~/work/arch-pc/lab09$ ./lab09-4
Результат: 25
```

{#fig:001

width=70%



## 6 Выводы

В процессе выполнения работы я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.

## 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/Learning-bash-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,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. 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).
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
18. — 1120 с. — (Классика Computer Science).