

# **Отчет по лабораторной работе №9**

**Дисциплина: архитектура компьютера**

Пихтовникова Алёна 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>7</b>
4.1	Релаксация подпрограмм в NASM . . . . .	7
4.1.1	Отладка программ с помощью GDB . . . . .	9
4.1.2	Добавление точек останова . . . . .	13
4.1.3	Работа с данными программы в GDB . . . . .	14
4.1.4	Обработка аргументов командной строки в GDB . . . . .	15
4.2	Задание для самостоятельной работы . . . . .	16
<b>5</b>	<b>Выводы</b>	<b>19</b>
<b>6</b>	<b>Список литературы</b>	<b>20</b>

## Список иллюстраций

4.1	Создание рабочего каталога . . . . .	7
4.2	Запуск программы из листинга . . . . .	7
4.3	Изменение программы первого листинга . . . . .	7
4.4	Запуск программы в отладчике . . . . .	10
4.5	Проверка программы отладчиком . . . . .	10
4.6	Запуск отладчика с брейкпоинтом . . . . .	11
4.7	Дисассимилирование программы . . . . .	12
4.8	Режим псевдографики . . . . .	13
4.9	Список брейкпоинтов . . . . .	13
4.10	Добавление второй точки останова . . . . .	14
4.11	Просмотр содержимого регистров . . . . .	14
4.12	Просмотр содержимого переменных двумя способами . . . . .	14
4.13	Изменение содержимого переменных двумя способами . . . . .	15
4.14	Просмотр значения регистра разными представлениями . . . . .	15
4.15	Примеры использования команды set . . . . .	15
4.16	Подготовка новой программы . . . . .	16
4.17	Проверка работы стека . . . . .	16
4.18	Измененная программа предыдущей лабораторной работы . . . . .	17

# 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. Самостоятельное выполнение заданий по материалам лабораторной работы

### 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, отработывает, 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

### 4.1 Релазиация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9 (рис. 4.1).

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

Рис. 4.1: Создание рабочего каталога

Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. 4.2).

```
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2x+7=27
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$
```

Рис. 4.2: Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения  $f(g(x))$  (рис. 4.3).

```
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2(3x-1)+7=65
```

Рис. 4.3: Изменение программы первого листинга

Код программы:

```

#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ', 0
result: DB '2(3x-1)+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:
push eax
call _subcalcul

mov ebx, 2
mul ebx
add eax, 7

mov [res], eax
pop eax
ret

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

#### 4.1.1 Отладка программ с помощью GDB

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике (рис. 4.4).

```

pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
ld: предупреждение: невозможно найти символ входа _start; начальный адрес не устанавливается
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.1-1.fc40
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-redhat-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...
(No debugging symbols found in lab09-2)
(gdb)

```

Рис. 4.4: Запуск программы в отладчике

Запустив программу командой `run`, я убедилась в том, что она работает исправно (рис. 4.5).

```

GNU gdb (Fedora Linux) 15.1-1.fc40
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-redhat-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...
(No debugging symbols found in lab09-2)
(gdb) run
Starting program: /home/pikhtovnikovaav/work/arch-pc/lab09/lab09-2
/bin/bash: строка 1: /home/pikhtovnikovaav/work/arch-pc/lab09/lab09-2: не удаётся запустить бинарный файл: Ошибка формата выполняемого файла
/bin/bash: строка 1: /home/pikhtovnikovaav/work/arch-pc/lab09/lab09-2: Выполнено
During startup program exited with code 126.
(gdb)

```

Рис. 4.5: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку `_start` и снова запускаю отладку (рис. 4.6).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb) run
Starting program: /home/pikhtovnikovaav/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
(gdb)

```

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel *amd топчик* (рис. 4.7).

Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ax, eax, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

```

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) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     eax,0x4
    0x08049005 <+5>:    mov     ebx,0x1
    0x0804900a <+10>:   mov     ecx,0x804a000
    0x0804900f <+15>:   mov     edx,0x8
    0x08049014 <+20>:   int     0x80
    0x08049016 <+22>:   mov     eax,0x4
    0x0804901b <+27>:   mov     ebx,0x1
    0x08049020 <+32>:   mov     ecx,0x804a008
    0x08049025 <+37>:   mov     edx,0x7
    0x0804902a <+42>:   int     0x80
    0x0804902c <+44>:   mov     eax,0x1
    0x08049031 <+49>:   mov     ebx,0x0
    0x08049036 <+54>:   int     0x80
End of assembler dump.
(gdb)

```

Рис. 4.7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. 4.8).

```

Register group: general
eax 0x0 0 ecx 0x0 0 eds 0x0 0
ebx 0x0 0 esp 0xffff0010 0x00000010 ebp 0x0 0x00490000
esi 0x0 0 edi 0x0 0 eip 0x00490000 0x00490000 <_start>
eflags 0x202 43 [ IF ] cs 0x2b 43 fs 0x2b 43
ds 0x2b 43 es 0x2b 43 fs 0x2b 43
gs 0x0 0

0x00490000 add BYTE PTR [eax],al
0x00490001 add BYTE PTR [eax],al
0x00490002 add BYTE PTR [eax],al
0x00490003 add BYTE PTR [eax],al
0x00490004 add BYTE PTR [eax],al
0x00490005 add BYTE PTR [eax],al
0x00490006 add BYTE PTR [eax],al
0x00490007 add BYTE PTR [eax],al
0x00490008 add BYTE PTR [eax],al
0x00490009 add BYTE PTR [eax],al
0x0049000a add BYTE PTR [eax],al
0x0049000b add BYTE PTR [eax],al

native process 6954 (asm) In: _start L11 PC: 0x00490000
(gdb)
(gdb) layout regs
(gdb)

```

Рис. 4.8: Режим псевдографики

## 4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. 4.9).

```

0x004976e add BYTE PTR [eax],al
0x0049770 add BYTE PTR [eax],al
0x0049772 add BYTE PTR [eax],al
0x0049774 add BYTE PTR [eax],al
0x0049776 add BYTE PTR [eax],al
0x0049778 add BYTE PTR [eax],al
0x004977a add BYTE PTR [eax],al
0x004977c add BYTE PTR [eax],al

native process 6954 (asm) In: _start L11 PC: 0x00490000
Breakpoint 2 at 0x00490000: file lab09-2.asm, line 11.
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x00490000 lab09-2.asm:11
breakpoint already hit 1 time
2 breakpoint keep y 0x00490000 lab09-2.asm:11
(gdb) break *0x00490000
Note: breakpoints 1 and 2 also set at pc 0x00490000.
Breakpoint 3 at 0x00490000: file lab09-2.asm, line 11.
(gdb)

```

Рис. 4.9: Список брейкпоинтов

Устанавливаю еще одну точку останова по адресу инструкции (рис. 4.10).

```

0x804976e    add    BYTE PTR [eax],al
0x8049770    add    BYTE PTR [eax],al
0x8049772    add    BYTE PTR [eax],al
0x8049774    add    BYTE PTR [eax],al
0x8049776    add    BYTE PTR [eax],al
0x8049778    add    BYTE PTR [eax],al
0x804977a    add    BYTE PTR [eax],al
0x804977c    add    BYTE PTR [eax],al

native process 6954 (asm) In: _start                               L11    PC: 0x8049000
Breakpoint 2 at 0x8049000: file lab09-2.asm, line 11.
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint       keep y   0x08049000 lab09-2.asm:11
         breakpoint already hit 1 time
2        breakpoint       keep y   0x08049000 lab09-2.asm:11
(gdb) break *0x08049000
Note: breakpoints 1 and 2 also set at pc 0x8049000.
Breakpoint 3 at 0x8049000: file lab09-2.asm, line 11.
(gdb)

```

Рис. 4.10: Добавление второй точки останова

### 4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers` (рис. 4.11).

```

eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd010     0xffffd010
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x8049000     0x8049000 <_start>
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. 4.12).

```

native process 6954 (asm) In: _start                               L11    PC: 0x8049000
esi            0x0            0
edi            0x0            0
eip            0x8049000     0x8049000 <_start>
--Type <RET> for more, q to quit, c to continue without paging--
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "World!\n\034"
(gdb)

```

Рис. 4.12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. 4.13).

```

native process 6954 (asm) in: _start L11 PC: 0x8049000
0x804a008 <msg2>: "World!\n\034"
(gdb) set{char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set{char}&msg1='h'
(gdb) x/1sb &msg1
0x804a008 <msg1>: "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "xorld!\n\034"
(gdb)

```

Рис. 4.13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. 4.14).

```

native process 6954 (asm) in: _start L11 PC: 0x8049000
0x804a008 <msg2>: "xorld!\n\034"
(gdb) p/t $ecx
$1 = 0
(gdb) p/s $edx
$2 = 0
(gdb) p/t &edx
No symbol "edx" in current context.
(gdb) p/x &edx
No symbol "edx" in current context.
(gdb)

```

Рис. 4.14: Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. 4.15).

```

native process 6954 (asm) in: _start L11 PC: 0x8049000
(gdb) p/t &edx
No symbol "edx" in current context.
(gdb) p/x &edx
No symbol "edx" in current context.
(gdb) set $ebx='2'
(gdb) p/s
$3 = 0
(gdb) p/s &ebx
No symbol "ebx" in current context.
(gdb)

```

Рис. 4.15: Примеры использования команды set

#### 4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. 4.16).

```

pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/w
ork/arch-pc/lab09/lab09-3.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.
asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab09$

```

Рис. 4.16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпоинт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра `esp` на `+4`, число обусловлено разрядностью системы, а указатель `void` занимает как раз 4 байта, ошибка при аргументе `+24` означает, что аргументы на вход программы закончились. (рис. 4.17).

```

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) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/pikhtovnikovaav/work/arch-pc/lab09/lab09-3 аргумент1 аргуме
нт 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
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:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffcfc0: 0x00000005

```

Рис. 4.17: Проверка работы стека

## 4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 4.18).



```

lab09-1.asm      lab09-2.asm      lab9-4.asm
SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0
SECTION .text
GLOBAL _start
_start:
mov eax, msg_func
call sprintf
pop ecx
pop edx
sub ecx, 1
mov esi, 0
next:
cmp ecx, 0h
jz _end
pop eax
call atoi
call _calculate_fx
add esi, eax
loop next
_end:
mov eax, msg_result
call sprintf
mov eax, esi
call sprintf
call quit
_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4

```

Рис. 4.18: Измененная программа предыдущей лабораторной работы

Код программы:

```

#include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
mov eax, msg_func
call sprintf

pop ecx
pop edx
sub ecx, 1
mov esi, 0

```

```

next:
cmp ecx, 0h
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:
mov ebx, 10
mul ebx
sub eax, 4

```

## **5 Выводы**

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи GDB и его основными возможностями.

## **6 Список литературы**

1. Курс на ТУИС
2. Лабораторная работа №9
3. Программирование на языке ассемблера NASM Столяров А. В.