

# **Отчёт по лабораторной работе №9**

**Понятие подпрограммы. Отладчик GDB.**

Малкина Дарья Александровна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Отладка программ с помощью GDB . . . . .	8
2.1.1	Добавление точек останова . . . . .	11
2.1.2	Работа с данными программы в GDB . . . . .	12
2.1.3	Обработка аргументов командной строки в GDB . . . . .	15
<b>3</b>	<b>Выполнение задания для самостоятельной работы</b>	<b>18</b>
<b>4</b>	<b>Выводы</b>	<b>27</b>

# Список иллюстраций

2.1	Программа lab9-1 - $f(x)=2x+7$ . . . . .	6
2.2	Добавление подпрограммы _subcalcul . . . . .	7
2.3	Программа lab9-1 - $f((g)x)$ . . . . .	7
2.4	Отладка программы lab9-2 с помощью GDB . . . . .	8
2.5	Запуск программы lab9-2 в оболчке GDB . . . . .	8
2.6	Установка точки останова на метке _start . . . . .	9
2.7	Дизассемблированный код программы lab9-2 . . . . .	9
2.8	Дизассемблированный код - синтаксис Intel . . . . .	10
2.9	Режим псевдографики . . . . .	11
2.10	Точки останова . . . . .	12
2.11	Команда stepi . . . . .	13
2.12	Просмотр значений msg1 и msg2 . . . . .	14
2.13	Изменение переменных - команда set . . . . .	14
2.14	Команда print . . . . .	14
2.15	Команда print . . . . .	15
2.16	Создаём файл lab9-3.as . . . . .	15
2.17	Загружаем файл в GDB . . . . .	16
2.18	Точка останова перед _start . . . . .	16
2.19	Содержимое стека . . . . .	17
3.1	Переменная res . . . . .	18
3.2	Основная программа . . . . .	19
3.3	Подпрограмма _funcalcul . . . . .	19
3.4	Программа lab9-var5 . . . . .	20
3.5	Программа lab8-var5 . . . . .	20
3.6	Неверный результат . . . . .	20
3.7	Запуск программы lab9-2 в оболчке GDB . . . . .	21
3.8	Установка точки останова на метке _start . . . . .	21
3.9	Дизассемблированный код программы lab9-4 . . . . .	22
3.10	Добавление точек останова . . . . .	23
3.11	Значения регистров после add ebx,eax . . . . .	23
3.12	Первая ошибка - неправильное использование инструкции mul . . . . .	24
3.13	Значения регистров . . . . .	24
3.14	Значения регистров после add edi,ebx . . . . .	25
3.15	Исправленный код программы lab9-4 . . . . .	25
3.16	Корректная работа программы lab9-4 . . . . .	26

## **Список таблиц**

# 1 Цель работы

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

## 2 Выполнение лабораторной работы

1. Сначала создадим каталог lab09 и файл lab9-1.asm. Рассмотрим программу, вычисляющую  $f(x)=2x+7$  с помощью подпрограммы `_calcul`. Добавим в файл lab9-1.asm текст программы из листинга 9.1. Создадим исполняемый файл и проверим его работу:

```
[damalkina@ArchVBox lab09]$ nasm -f elf lab9-1.asm -o lab9-1.o
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 1
2x+7=9
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 2
2x+7=11
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 5
2x+7=17
```

Рис. 2.1: Программа lab9-1 -  $f(x)=2x+7$

После этого добавим в программу подпрограмму `_subcalcul` внутри `_calcul`, чтобы `_subcalcul` вычисляла  $g(x) = 3x-1$ , а после передавала значение  $x$  в `_calcul`, где будет вычислено  $f(g(x))$ :

```

35 ;-----
36 ; Подпрограмма вычисления выражения "2x+7"
37 ;-----
38 _calcul:
39
40 call _subcalcul ; Вызов подпрограммы _subcalcul
41
42 mov ebx,2
43 mul ebx
44 add eax,7
45 mov [res],eax
46
47 ret                ; возвращению в основную программу
48
49 ;-----
50 ; Подпрограмма вычисления выражения "3x-1"
51 ;-----
52 _subcalcul:
53 mov ebx,3
54 mul ebx
55 sub eax,1
56 mov [res],eax
57
58 ret                ; возвращению в основную программу

```

Рис. 2.2: Добавление подпрограммы \_subcalcul

Создадим исполняемый файл и проверим его работу:

```

[damalkina@ArchVBox lab09]$ nasm -f elf lab9-1.asm -o lab9-1.o
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 1
f(g)=2(3x-1)+7=11
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 2
f(g)=2(3x-1)+7=17
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 5
f(g)=2(3x-1)+7=35
[damalkina@ArchVBox lab09]$

```

Рис. 2.3: Программа lab9-1 -  $f((g)x)$

## 2.1 Отладка программ с помощью GDB

2. Создадим файл lab9-2.asm с программой из листинга 9.2 и соберем исполняемый файл. Добавим отладочную информацию, используя ключ -g:

```
[damalkina@ArchVBox lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
nasm: fatal: unable to open input file 'lab09-2.asm' No such file or directory
[damalkina@ArchVBox lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[damalkina@ArchVBox lab09]$ gdb lab9-2
GNU gdb (GDB) 15.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
```

Рис. 2.4: Отладка программы lab9-2 с помощью GDB

После этого загрузим исполняемый файл в отладчик GDB и проверим работу программы командой run:

```
(gdb) run
Starting program: /home/damalkina/work/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.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.
Hello, world!
[Inferior 1 (process 22489) exited normally]
```

Рис. 2.5: Запуск программы lab9-2 в оболчке GDB

Далее установим точку останова на метке \_start командой break \_start и запустим программу еще раз:



```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab9-2.asm, line 12.
(gdb) run
Starting program: /home/damalkina/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:12
12      mov eax, 4
```

Рис. 2.6: Установка точки останова на метке \_start

Затем посмотрим дизассемблированный код командой `disassemble _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.
```

Рис. 2.7: Дизассемблированный код программы lab9-2

Переключимся на синтаксис Intel командой `set disassembly-flavor intel` и снова выведем дизассемблированный код:

```

(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. 2.8: Дизассемблированный код - синтаксис Intel

Заметим, что отображения кода в режимах АТТ и Intel различаются:

Режим АТТ - операнды пишутся в порядке источник, затем приемник, при этом перед именем регистра ставится символ %, а непосредственные значения обозначаются знаком \$ перед числом.

Режим Intel - операнды пишутся в порядке приемник, затем источник. Префикса % перед регистрами нет и знак \$ не используется для обозначения непосредственных значений.

Включим режим псевдографики командами `layout asm` и `layout regs`:

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd840 0xffffd840
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]

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

native process 23767 (asm) In: _start      L12    PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 2.9: Режим псевдографики

### 2.1.1 Добавление точек останова

До этого мы установили точку останова на метке `_start`, проверим это командой `info breakpoints (i b)`. Затем по адресу инструкции установим еще одну точку останова командой `break *`. После посмотрим информацию о всех установленных точках останова командой `i b`:

```
0x804902c <_start+44> mov    eax,0x1
b+ 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038          add    BYTE PTR [eax],al

native process 23767 (asm) In: _start      L12    PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint     keep y  0x08049000 lab9-2.asm:12
        breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 25.
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint     keep y  0x08049000 lab9-2.asm:12
        breakpoint already hit 1 time
2        breakpoint     keep y  0x08049031 lab9-2.asm:25
(gdb) █
```

Рис. 2.10: Точки останова

## 2.1.2 Работа с данными программы в GDB

Воспользуемся командой `stepi (si)` пять раз, и понаблюдаем, как меняются значения регистров после каждой инструкции:

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd840 0xffffd840
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35

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

native process 25276 (asm) In: _start L18 PC: 0x8049016
cs      0x23      35
ss      0x2b      43
--Type <RET> for more, q to quit, c to continue without paging--
ds      0x2b      43
es      0x2b      43
fs      0x0       0
gs      0x0       0
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.11: Команда stepi

Воспользуемся командой x/1sb &msg1 для просмотра значения msg1. Затем мы найдем значение msg2 по адресу (определим адрес в дизассемблированной инструкции).

```

0x8049014 <_start+20> int    0x80
>0x8049016 <_start+22> mov    eax,0x4
0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a008
0x8049025 <_start+37> mov    edx,0x7
0x804902a <_start+42> int    0x80
0x804902c <_start+44> mov    eax,0x1

native process 25276 (asm) In: _start L18 PC: 0x8049016
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"

```

Рис. 2.12: Просмотр значений msg1 и msg2

Изменим первый символ msg1 командой `set {char}msg1='h'`, после аналогично заменим первый символ в переменной msg2:

Изменение переменных - команда `set`

Рис. 2.13: Изменение переменных - команда `set`

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`:

```

(gdb) p/x $edx
$15 = 0x8
(gdb) p/t $edx
$16 = 1000
(gdb) p/c $edx
$17 = 8 '\b'
(gdb) 

```

Рис. 2.14: Команда `print`

С помощью команды `set` изменим значение регистра `ebx` сначала на '2', затем на 2, и после каждого изменения выведем значение `edx` как строку:

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$13 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$14 = 2

```

Рис. 2.15: Команда print

Заметим разницу в выводе:

В первом случае `set $ebx='2'` - мы присвоили регистру `ebx` строковое значение '2', поэтому команда `p/s $ebx`, которая выводит значение как строку, отображает ASCII-код символа "2", который равен 50.

Во втором случае `set $ebx=2` - мы присвоили регистру `ebx` числовое значение 2, теперь команда `p/s $ebx` выводит число 2, потому что интерпретирует `ebx` как число, а не как строку.

Выходим из GDB.

### 2.1.3 Обработка аргументов командной строки в GDB

Скопируем файл `lab8-2.asm` в файл `lab09-3.asm`, создадим исполняемый файл:

```

[damalkina@ArchVBox lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
nasm: fatal: unable to open input file 'lab09-3.asm' No such file or directory
[damalkina@ArchVBox lab09]$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[damalkina@ArchVBox lab09]$ ls
in_out.asm  lab9-1.o    lab9-2.lst  lab9-3.asm
lab9-1      lab9-2     lab9-2.o    lab9-3.lst
lab9-1.asm  lab9-2.asm lab9-3      lab9-3.o

```

Рис. 2.16: Создаём файл lab9-3.as

После загрузим исполняемый файл в `gdb` с помощью ключа `-args`, передав ему аргументы:

```
[damalkina@ArchVBox lab09]$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'

GNU gdb (GDB) 15.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-3...
(gdb) █
```

Рис. 2.17: Загружаем файл в GDB

Установим точку останова перед `_start` и запустим программу:

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 6.
(gdb) run
Starting program: /home/damalkina/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2
аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.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.
Downloading separate debug info for system-supplied DSO at 0xf7fc0000

Breakpoint 1, _start () at lab9-3.asm:6
6      pop есх      ; Извлекаем из стека в `есх` количество аргументов (перв
ое значение в стеке)
(gdb) █
```

Рис. 2.18: Точка останова перед `_start`

Проверим количество аргументов с помощью команды `x/x $esp`. Результат - число 5, так как у нас есть имя программы и четыре аргумента. Далее, исследуем содержимое стека. По адресу `[esp+4]` находится адрес имени программы, по адресу `[esp+8]` — адрес первого аргумента, по адресу `[esp+12]` — второго, и так далее:



```

(gdb) x/x $esp
0xffffd800:    0x05
(gdb) x/s *(void**)(esp+4)
0xffffd9ba:    "/home/damalkina/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp+8)
0xffffd9e4:    "аргумент1"
(gdb) x/s *(void**)(esp+12)
0xffffd9f6:    "аргумент"
(gdb) x/s *(void**)(esp+16)
0xffffda07:    "2"
(gdb) x/s *(void**)(esp+20)
0xffffda09:    "аргумент 3"
(gdb) x/s *(void**)(esp+24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.19: Содержимое стека

Так как в 32-битных системах, адреса памяти в виде 32-битными числами (32 бита = 4 байта), поэтому мы перемещаемся по стеку с интервалом в 4 байта вперед, чтобы добраться до начала следующего 32-битного адреса.

### 3 Выполнение задания для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму.

В сегменте `.bss` объявим переменную `res` для хранения результатов вычислений:

```
3 SECTION .data
4 msg1 db "Функция: 4x+3",0
5 msg2 db "Результат: ",0
6
7 SECTION .text
8
9 SECTION .bss
10 res: RESB 80
11
12 SECTION .text
13 GLOBAL _start
```

Рис. 3.1: Переменная `res`

В основной программе добавим вызов подпрограммы `_funcalcul` (33 строка) и изменим операнды в сложении промежуточных результатов функции (35 строка):

```

23 ;-----
24 ; Основная программа
25 ;-----
26 next:
27 cmp ecx,0h      ; проверяем, есть ли еще аргументы
28 jz _end         ; если аргументов нет выходим из цикла
29
30 pop eax         ; иначе извлекаем следующий аргумент из стека
31 call atoi       ; преобразуем символ в число
32
33 call _funcalcul ; вызов подпрограммы
34
35 add esi, [res]   ; складываем результат с предыдущими результатами
36 loop next       ; переход к обработке следующего аргумента
37
38 _end:
39 mov eax, msg1    ; вывод сообщения "Функция: 4x+3"
40 call sprintf
41 mov eax, msg2    ; вывод сообщения "Результат: "
42 call sprintf
43 mov eax, esi     ; записываем произведение в регистр `eax`
44 call iprintLF    ; печать результата
45 call quit       ; завершение программы

```

Рис. 3.2: Основная программа

Вынесем вычисления функции за основной код, добавим сохранение результата в переменную `res` (54 строка):

```

47 ;-----
48 ; Подпрограмма вычисления выражения "4x+3"
49 ;-----
50 _funcalcul:
51 mov ebx, 4      ; 'ebx = 4'
52 mul ebx         ; домножаем '4' на аргумент 'eax=4*eax'
53 add eax, 3      ; прибавляем '3+4*eax'
54 mov [res],eax
55
56 ret            ; возвращению в основную программу

```

Рис. 3.3: Подпрограмма `_funcalcul`

Создаём исполняемый файл и запускаем его, проверим работу программы с разными аргументами, для простоты проверки введем те же аргументы, что и при проверке программы `lab8-var5`:

```
[damalkina@ArchVBox lab09]$ nasm -f elf lab9-var5.asm -o lab9-var5.o
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-var5 lab9-var5.o
[damalkina@ArchVBox lab09]$ ./lab9-var5 1 2 3 4
Функция: 4x+3
Результат: 52
[damalkina@ArchVBox lab09]$ ./lab9-var5 1 2 3
Функция: 4x+3
Результат: 33
[damalkina@ArchVBox lab09]$ ./lab9-var5 2 5 0
Функция: 4x+3
Результат: 37
[damalkina@ArchVBox lab09]$
```

Рис. 3.4: Программа lab9-var5

Для сравнения приведем результаты работы программы lab8-var5:

```
[damalkina@ArchVBox lab08]$ nasm -f elf lab8-var5.asm -o lab8-var5.o
[damalkina@ArchVBox lab08]$ ld -m elf_i386 -o lab8-var5 lab8-var5.o
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2
Функция: 4x+3
Результат: 18
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2 3 4
Функция: 4x+3
Результат: 52
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2 3
Функция: 4x+3
Результат: 33
[damalkina@ArchVBox lab08]$ ./lab8-var5 2 5 0
Функция: 4x+3
Результат: 37
[damalkina@ArchVBox lab08]$
```

Рис. 3.5: Программа lab8-var5

2. В листинге 9.3 приведена программа вычисления выражения  $(3+2) \cdot 4 + 5$ . При запуске данная программа дает результат 10, что неверно, убедимся в этом, посчитав аналитически  $5 \cdot 4 + 5 = 20 + 5 = 25$ :

```
[damalkina@ArchVBox lab09]$ nasm -f elf lab9-4.asm -o lab9-4.o
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[damalkina@ArchVBox lab09]$ ./lab9-4
Результат: 10
[damalkina@ArchVBox lab09]$
```

Рис. 3.6: Неверный результат

С помощью отладчика GDB, проанализируем изменения значений регистров и определим ошибку.

Соберем исполняемый файл, добавив отладочную информацию, после загрузим исполняемый файл в отладчик GDB и проверим работу программы:

```
[damalkina@ArchVBox lab09]$ nasm -f elf -g -l lab9-4.lst lab9-4.asm
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[damalkina@ArchVBox lab09]$ gdb lab9-4
GNU gdb (GDB) 15.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-4...
(gdb)
```

Рис. 3.7: Запуск программы lab9-2 в оболчке GDB

Установим точку останова на метке `_start` и запустим программу еще раз:

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-4.asm, line 14.
(gdb) r
Starting program: /home/damalkina/work/arch-pc/lab09/lab9-4

Breakpoint 1, _start () at lab9-4.asm:14
14      mov ebx,3
(gdb) █
```

Рис. 3.8: Установка точки останова на метке `_start`

Затем посмотрим дизассемблированный код:

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:      mov     $0x3,%ebx
    0x080490ed <+5>:      mov     $0x2,%eax
    0x080490f2 <+10>:     add     %eax,%ebx
    0x080490f4 <+12>:     mov     $0x4,%ecx
    0x080490f9 <+17>:     mul     %ecx
    0x080490fb <+19>:     add     $0x5,%ebx
    0x080490fe <+22>:     mov     %ebx,%edi
    0x08049100 <+24>:     mov     $0x804a000,%eax
    0x08049105 <+29>:     call    0x804900f <sprint>
    0x0804910a <+34>:     mov     %edi,%eax
    0x0804910c <+36>:     call    0x8049086 <iprintLF>
    0x08049111 <+41>:     call    0x80490db <quit>
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:      mov     ebx,0x3
    0x080490ed <+5>:      mov     eax,0x2
    0x080490f2 <+10>:     add     ebx,eax
    0x080490f4 <+12>:     mov     ecx,0x4
    0x080490f9 <+17>:     mul     ecx
    0x080490fb <+19>:     add     ebx,0x5
    0x080490fe <+22>:     mov     edi,ebx
    0x08049100 <+24>:     mov     eax,0x804a000
    0x08049105 <+29>:     call    0x804900f <sprint>
    0x0804910a <+34>:     mov     eax,edi
    0x0804910c <+36>:     call    0x8049086 <iprintLF>
    0x08049111 <+41>:     call    0x80490db <quit>
End of assembler dump.
(gdb) █

```

Рис. 3.9: Дизассемблированный код программы lab9-4

Включим режим псевдографики и добавим точки установа по адресу инструкции после арифметических команд:

```

B> 0x80490e8 <_start>      mov     ebx,0x3
    0x80490ed <_start+5>    mov     eax,0x2
b+  0x80490f2 <_start+10>   add     ebx,eax
    0x80490f4 <_start+12>   mov     ecx,0x4
b+  0x80490f9 <_start+17>   mul     ecx
b+  0x80490fb <_start+19>   add     ebx,0x5
    0x80490fe <_start+22>   mov     edi,ebx
    0x8049100 <_start+24>   mov     eax,0x804a000
    0x8049105 <_start+29>   call    0x804900f <sprint>
    0x804910a <_start+34>   mov     eax,edi
    0x804910c <_start+36>   call    0x8049086 <iprintLF>

native process 9850 (asm) In: _start          L14    PC: 0x80490e8
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint        keep y 0x080490e8 lab9-4.asm:14
      breakpoint already hit 1 time
2      breakpoint        keep y 0x080490f2 lab9-4.asm:16
3      breakpoint        keep y 0x080490f9 lab9-4.asm:18
4      breakpoint        keep y 0x080490fb lab9-4.asm:19
(gdb)

```

Рис. 3.10: Добавление точек останова

Воспользуемся командами stepi и print, и понаблюдаем, как меняются значения регистров после инструкций. После инструкции add ebx,eax, регистры принимают верные значения:

```

Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffd850 0xffffd850
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f4 0x80490f4 <_start+12>
eflags   0x10206  [ PF IF RF ]
cs       0x23     35
ss       0x2b     43

B+ 0x80490e8 <_start>      mov     ebx,0x3
    0x80490ed <_start+5>    mov     eax,0x2
B+ 0x80490f2 <_start+10>   add     ebx,eax
    >0x80490f4 <_start+12>   mov     ecx,0x4
b+ 0x80490f9 <_start+17>   mul     ecx
b+ 0x80490fb <_start+19>   add     ebx,0x5

```

Рис. 3.11: Значения регистров после add ebx,eax

Заметим, что после команды mul ecx, значение ebx остаётся неизменным, хотя должно принимать значение  $5 \cdot 4 = 20$ , но меняется значение eax, хотя должно оставаться равным 2. Делаем вывод, что инструкция mul ecx умножает содержимое регистра eax на ecx, а не ebx на ecx, как задумано, результат записывается в eax:

Register group: general		
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffd850	0xffffd850
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490fb	0x80490fb <_start+19>
eflags	0x10202	[ IF RF ]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

  

B+	0x80490e8	<_start>	mov	ebx,0x3
	0x80490ed	<_start+5>	mov	eax,0x2
B+	0x80490f2	<_start+10>	add	ebx,ebx
	0x80490f4	<_start+12>	mov	ecx,0x4
B+	0x80490f9	<_start+17>	mul	ecx
B+	0x80490fb	<_start+19>	add	ebx,0x5
	0x80490fe	<_start+22>	mov	edi,ebx
	0x8049100	<_start+24>	mov	eax,0x804a000

Рис. 3.12: Первая ошибка - неправильное использование инструкции mul

С помощью команды print сравним значения регистров eax, ebx, ecx до команды mul ecx и после:

```
Breakpoint 3, _start () at lab9-4.asm:18
(gdb) p $eax
$5 = 2
(gdb) p $ebx
$6 = 5
(gdb) p $ecx
$7 = 4
(gdb) si

Breakpoint 4, _start () at lab9-4.asm:19
(gdb) p $eax
$8 = 8
(gdb) p $ebx
$9 = 5
(gdb) p $ecx
$10 = 4
```

Рис. 3.13: Значения регистров

Чтобы исправить это, нужно сначала скопировать содержимое ebx в eax перед выполнением mul ecx. Однако код всё равно неверный из-за того, что после умножения 5 прибавляется к ebx, а не к результату умножения, который находится в eax. Это вторая ошибка - неправильное место для второго сложения:



Register group: general		
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0xa	10
esp	0xffffd850	0xffffd850
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490fe	0x80490fe <_start+22>
eflags	0x10206	[ PF IF RF ]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

  

B+	0x80490e8	<_start>	mov	ebx,0x3
	0x80490ed	<_start+5>	mov	eax,0x2
B+	0x80490f2	<_start+10>	add	ebx,ebx
	0x80490f4	<_start+12>	mov	ecx,0x4
B+	0x80490f9	<_start+17>	mul	ecx
B+	0x80490fb	<_start+19>	add	ebx,0x5
>	0x80490fe	<_start+22>	mov	edi,ebx
	0x8049100	<_start+24>	mov	eax,0x804a000

Рис. 3.14: Значения регистров после add edi,ebx

Исправим код, скопируем значение ebx в eax перед выполнением mul ecx и прибавим 5 напрямую к eax, после чего сохраним результат в edi:

```

11 ; -----
12 ;   Вычисление выражения (3+2)*4+5
13 ; -----
14 mov ebx,3
15 mov eax,2
16 add ebx,eax
17 mov ecx,4
18 mov eax, ebx    ; содержимое из ebx в eax
19 mul ecx
20 add eax,5      ; 5+(eax * ecx)
21 mov edi,eax

```

Рис. 3.15: Исправленный код программы lab9-4

Создим исполняемый файл и запустим его, для проверки работы программы, убедимся, что после внесения изменений программа работает корректно:

```
[damalkina@ArchVBox lab09]$ nasm -f elf lab9-4.asm -o lab9-4.o  
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o  
[damalkina@ArchVBox lab09]$ ./lab9-4  
Результат: 25  
[damalkina@ArchVBox lab09]$
```

Рис. 3.16: Коректтная работа программы lab9-4

## 4 Выводы

В ходе лабораторной работы и выполнения самостоятельного задания мы приобрели практические навыки написания ассемблерных программ с использованием подпрограмм, попрактиковались в отладке программ с помощью GDB, научились использовать GDB для пошагового выполнения кода, анализа значений регистров и памяти, что позволило нам обнаружить и исправить ошибки в программе.