

# **Понятие подпрограммы. Отладчик GDB.**

**Лабораторная работа №9.**

Дагделен Зейнап Реджеповна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Задание</b>	<b>7</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>8</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
4.1	Реализация подпрограмм в NASM . . . . .	11
4.2	Отладка программ с помощью GDB . . . . .	14
4.2.1	Добавление точек останова . . . . .	18
4.2.2	Работа с данными программы в GDB . . . . .	19
4.2.3	Обработка аргументов командной строки в GDB . . . . .	22
4.3	Задания для самостоятельной работы . . . . .	24
<b>5</b>	<b>Выводы</b>	<b>35</b>
<b>6</b>	<b>Список литературы</b>	<b>36</b>

## Список иллюстраций

4.1	Создание папки и файла для лабораторной работы . . . . .	11
4.2	Ввод текста программы из листинга 9.1 . . . . .	12
4.3	Запуск исполняемого файла . . . . .	12
4.4	Текст программы . . . . .	13
4.5	Запуск исполняемого файла и его создание . . . . .	14
4.6	Создание файла . . . . .	14
4.7	Ввод текста программы из листинга 9.2 . . . . .	15
4.8	Получение исполняемого файла, его загрузка в отладчик, проверка работы (run) . . . . .	16
4.9	Установка брейкпоинта и запуск программы . . . . .	16
4.10	Использование команды disassemble . . . . .	17
4.11	Использование команд disassemble и disassembly-flavor intel . . .	17
4.12	Включение режима псевдографики . . . . .	18
4.13	Просмотр информации о точке останова . . . . .	18
4.14	Установление точек останова и просмотр информации о них . . .	19
4.15	После использования команды stepi . . . . .	20
4.16	Просмотр значений переменной msg1 . . . . .	20
4.17	Просмотр значений переменной msg2 . . . . .	20
4.18	Использование команды set . . . . .	21
4.19	Использование команды set . . . . .	21
4.20	Использование команды set для изменения значения регистра . .	21
4.21	Использование команды set для изменения значения регистра . .	21
4.22	Завершение работы GDB . . . . .	22
4.23	Копирование файла . . . . .	22
4.24	Создание исполняемого файла . . . . .	22
4.25	Загрузка файла с аргументами в отладчик . . . . .	23
4.26	Установление точки останова и запуск программы . . . . .	23
4.27	Вершина стека . . . . .	23
4.28	Просмотр значений, введенных в стек . . . . .	24
4.29	Копирование из одной папки в другую . . . . .	24
4.30	Написание кода подпрограммы . . . . .	25
4.31	Создание исполняемого файла . . . . .	25
4.32	Запуск программы и проверка его вывода . . . . .	26
4.33	Создание файла . . . . .	27
4.34	Ввод текста программы из листинга 9.3 . . . . .	28
4.35	Создание исполняемого файла . . . . .	28
4.36	Запуск исполняемого файла . . . . .	29

4.37 Пошаговый просмотр работы регистров . . . . .	30
4.38 Пошаговый просмотр работы регистров . . . . .	31
4.39 Исправление ошибки . . . . .	32
4.40 Ошибка исправлена . . . . .	33

## 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. 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 включена 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 `sl`) 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 `еір` 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 `еір`. 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 с помощью `mkdir`, перехожу в него (утилита `cd`) и создаю файл `lab09-1.asm` благодаря `touch`. (рис. [4.1])

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

Рис. 4.1: Создание папки и файла для лабораторной работы

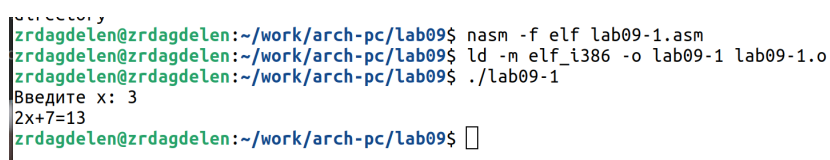
Ввожу в файл `lab09-1.asm` текст программы с использованием подпрограммы из листинга 9.1. (рис. [4.2])



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
```

Рис. 4.2: Ввод текста программы из листинга 9.1

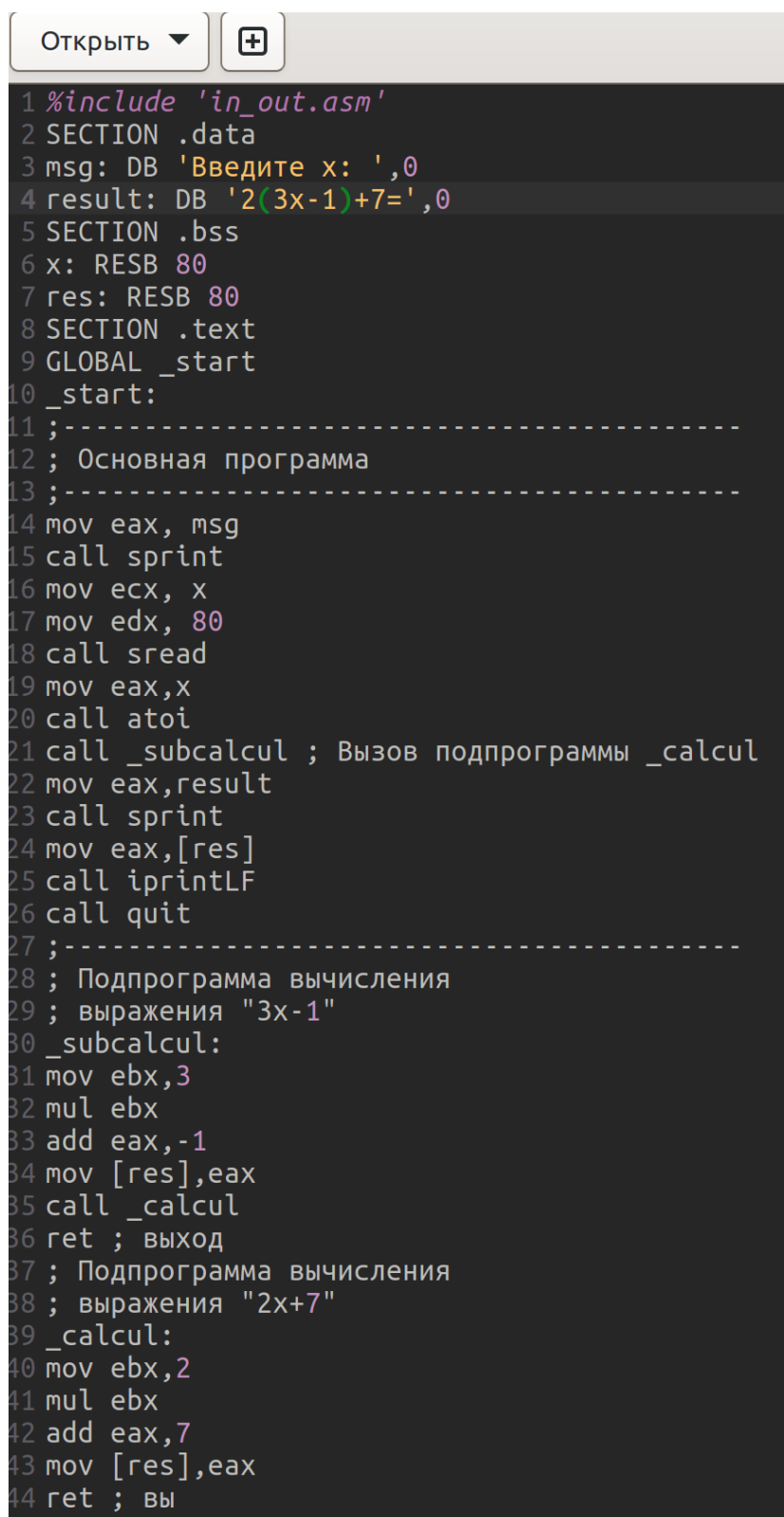
Создаю исполняемый файл и проверяю его работу. (рис. [4.3])



```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=13
zrdagdelen@zrdagdelen:~/work/arch-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])



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

Рис. 4.4: Текст программы

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

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
2(3x-1)+7=17
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$
```

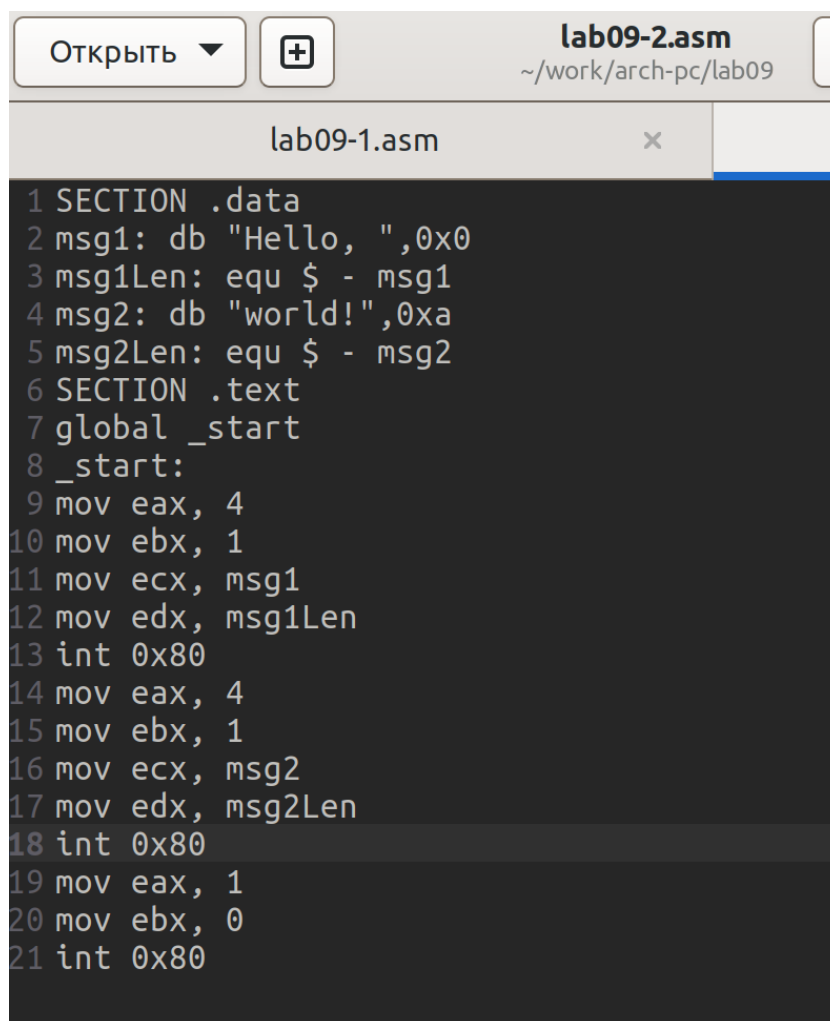
Рис. 4.5: Запуск исполняемого файла и его создание

## 4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm (с помощью touch) и ввожу в него текст программы из Листинга 9.2. (рис. [4.6]- [4.7])

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ touch lab09-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$
```

Рис. 4.6: Создание файла



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

Рис. 4.7: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g', загружаю исполняемый файл в отладчик gdb и проверяю работу программы, запустив ее в оболочке GDB с помощью команды run(рис. [4.8]).

```
zrdagdelen@zrdagdelen: ~/work/arch-pc/lab09
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/zrdagdelen/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4661) exited normally]
(gdb) □
```

Рис. 4.8: Получение исполняемого файла, его загрузка в отладчик, проверка работы (run)

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start` и запускаю её (рис. [4.9]).

```
[Inferior 1 (process 4661) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) r
Starting program: /home/zrdagdelen/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) □
```

Рис. 4.9: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` [4.10], и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel` (рис. [4.11]).



```
(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. 4.10: Использование команды disassemble

```
(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. 4.11: 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]).

```

zrdagdelen@zrdagdelen: ~/work/arch-pc/lab09
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      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

native process 4866 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/zrdagdelen/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
(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 (i b)` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова (рис. [4.13]-[4.14]).

```

zrdagdelen@zrdagdelen: ~/work/arch-pc/lab09
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      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

native process 4866 In: _start L9 PC: 0x8049000
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb)

```

Рис. 4.13: Просмотр информации о точке останова

```
zrdagdelen@zrdagdelen: ~/work/arch-pc/lab09
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd260 0xffffd260

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

native process 5203 In: _start L9 PC: 0x8049000
(gdb) b *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. 4.14: 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. [4.15])

```

zrdagdelen@zrdagdelen: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd260 0xffffd260
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
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 19263 In: _start L14 PC: 0x8049016
(gdb) layout regs
(gdb) r
Starting program: /home/zrdagdelen/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(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]-[4.17])

```

--Type <RET> for more, q to quit, c to continue without paging-- qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb)

```

Рис. 4.16: Просмотр значений переменной msg1

```

(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 4.17: Просмотр значений переменной msg2

С помощью команды `set` изменяю первый символ переменной `msg1` и заменяю первый символ в переменной `msg2`. (рис. [4.18]-[4.19])

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

---

Рис. 4.18: Использование команды `set`

```
(gdb) set {char}&msg2='s'
(gdb) x/2sb &msg2
0x804a008 <msg2>:      "sorld!\n\034"
0x804a011:      ""
(gdb) █
```

---

Рис. 4.19: Использование команды `set`

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием (рис. [4.20]-[4.21] ).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) █
```

---

Рис. 4.20: Использование команды `set` для изменения значения регистра

```
(gdb) set $ebx=2
(gdb) p/s $ebx
$3 = 2
(gdb) █
```

---

Рис. 4.21: 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 (c)` и выхожу из GDB с помощью команды `quit (q)`. (рис. [4.22])

```

zrdagdelen@zrdagdelen: ~/work/...
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x32      50
esp      0xffffd260 0xffffd260

B+> 0x8049000 <_start> mov eax,0x4
B+ 0x8049000 <_start> mov eax,0x4
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80

native process 4316 In: _start L9 PC: 0x8049000
(gdb) pNo process In: L?? PC: ??
(gdb) set $ebx='2'
(gdb) p/s $ebx
$2 = 50
(gdb) c
Continuing.
Hello, world!
[Inferior 1 (process 4316) exited normally]
(gdb) 

```

Рис. 4.22: 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. [4.23])

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2
.asm ~/work/arch-pc/lab09/lab09-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ls
in_out.asm lab09-1.asm lab09-2 lab09-2.lst lab09-3.asm
lab09-1 lab09-1.o lab09-2.asm lab09-2.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ 

```

Рис. 4.23: Копирование файла

Создаю исполняемый файл(рис. [4.24]).

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst
lab09-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab0
9-3.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ 

```

Рис. 4.24: Создание исполняемого файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `-args`. (рис. [4.25])

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
l>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) run
Starting program: /home/zrdagdelen/work/arch-pc/lab09/lab09-3 аргумент1 арг
умент 2 аргумент\ 3
аргумент1
аргумент
2
аргумент 3
[Inferior 1 (process 4409) exited normally]
(gdb) □
```

Рис. 4.25: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. [4.26])

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

Breakpoint 1, _start () at lab09-3.asm:6
6      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) □
```

Рис. 4.26: Установление точки останова и запуск программы

Посматриваю вершину стека (рис. [4.27]).

```
(gdb) x/x $esp
0xffffd210: 0x00000005
(gdb) □
```

Рис. 4.27: Вершина стека

Число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрю остальные позиции стека – по адресу [esp+4] располагается адрес в памяти, где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. ((рис. [4.28]))

```
0xffffd3cc: 0x00000000
(gdb) x/s *(void**)(esp + 4)
0xffffd3cc: "/home/zrdagdelen/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd3f8: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd40a: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd41b: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd41d: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) □
```

Рис. 4.28: Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к количество аргументов командной строки = 4.

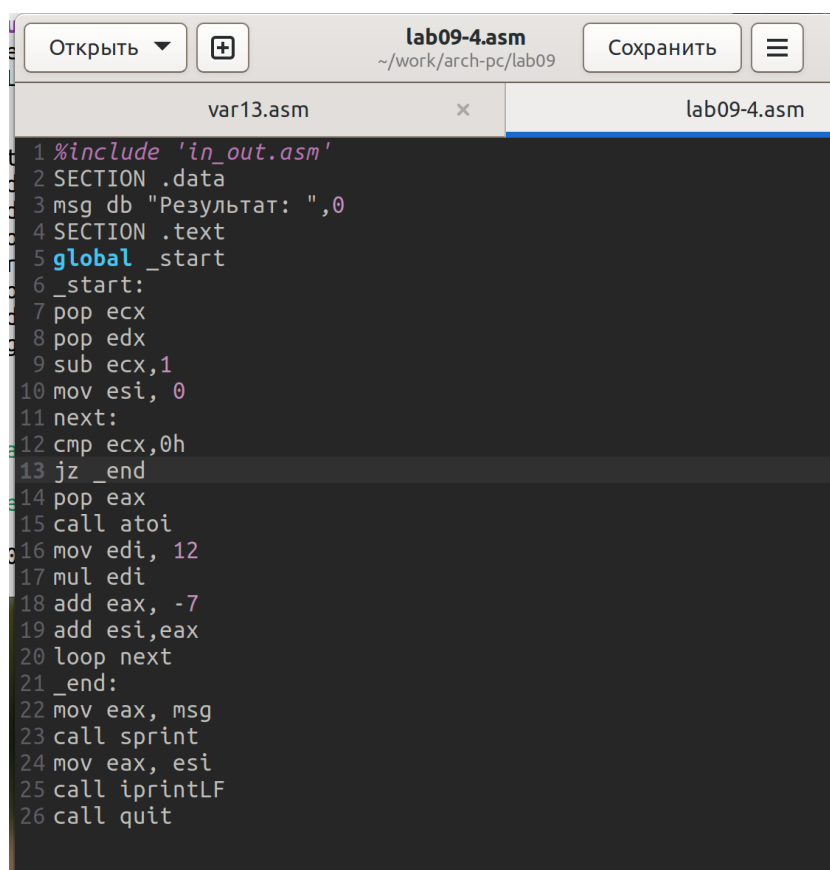
### 4.3 Задания для самостоятельной работы

1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму (рис. [4.29]). Для этого сначала копирую из lab06 нужный мне файл в нынешнюю папку(рис. [4.30]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/var13.asm ~/work/arch-pc/lab09/lab09-4.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ □
```

Рис. 4.29: Копирование из одной папки в другую

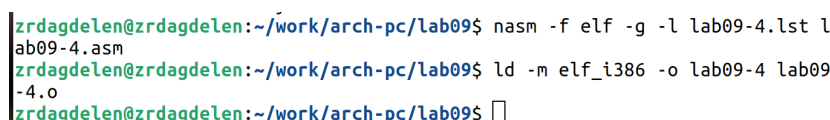




```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx
9 sub ecx,1
10 mov esi, 0
11 next:
12 cmp ecx,0h
13 jz _end
14 pop eax
15 call atoi
16 mov edi, 12
17 mul edi
18 add eax, -7
19 add esi,eax
20 loop next
21 _end:
22 mov eax, msg
23 call sprint
24 mov eax, esi
25 call iprintLF
26 call quit
```

Рис. 4.30: Написание кода подпрограммы

Создаю исполняемый файл (рис. [4.31]), запускаю код и проверяю, что она работает корректно(рис. [4.32]).



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

Рис. 4.31: Создание исполняемого файла

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ gdb --args lab09-4 1 2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
l>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-4...
(gdb) r
Starting program: /home/zrdagdelen/work/arch-pc/lab09/lab09-4 1 2
Результат: 22
[Inferior 1 (process 11275) exited normally]
(gdb) 

```

Рис. 4.32: 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
next:
cmp ecx,0h
jz _end
pop eax
call atoi
mov edi, 12

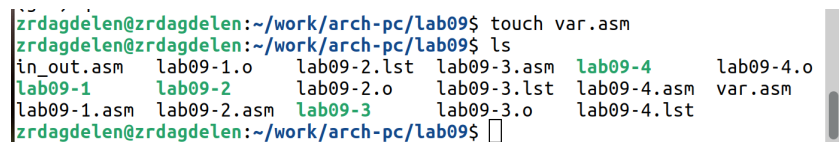
```

```

mul edi
add eax, -7
add esi, eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

```

2. Создаю файл (рис. [4.33]) и ввожу в файл task1.asm текст программы из листинга 9.3. (рис. [4.34])

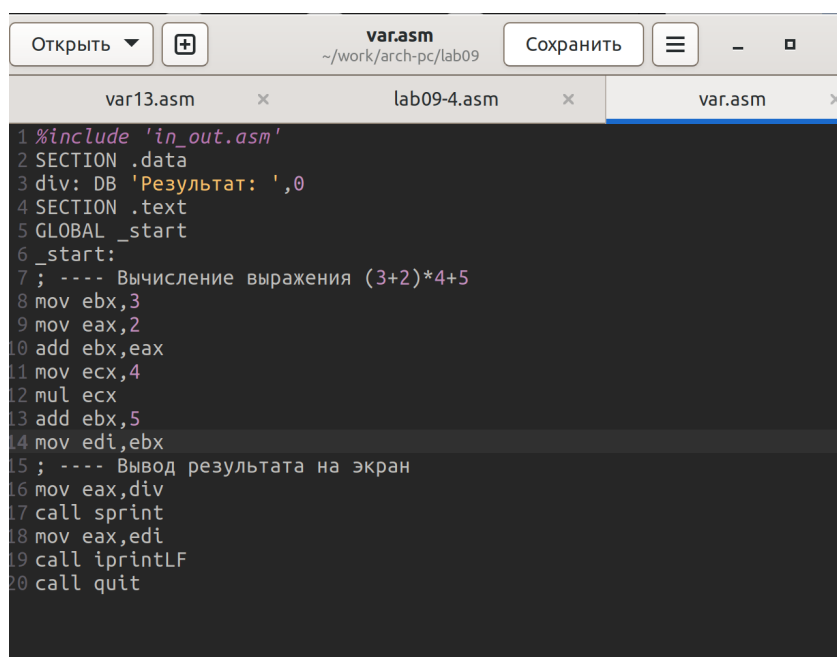


```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ touch var.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ls
in_out.asm  lab09-1.o    lab09-2.lst  lab09-3.asm  lab09-4    lab09-4.o
lab09-1     lab09-2     lab09-2.o    lab09-3.lst  lab09-4.asm  var.asm
lab09-1.asm lab09-2.asm lab09-3      lab09-3.o    lab09-4.lst
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$

```

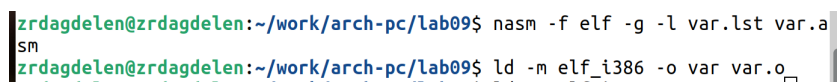
Рис. 4.33: Создание файла



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

Рис. 4.34: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его (рис. [4.35]-[4.36]).



```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ nasm -f elf -g -l var.lst var.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ld -m elf_i386 -o var var.o
```

Рис. 4.35: Создание исполняемого файла

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ gdb var
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
l>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from var...
(gdb) r
Starting program: /home/zrdagdelen/work/arch-pc/lab09/var
Результат: 10
[Inferior 1 (process 11767) exited normally]
(gdb) 

```

Рис. 4.36: Запуск исполняемого файла

Видим, что мы получили неправильный ответ.

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейк-поинты для некоторых инструкций, связанной с вычислениями. С помощью команды continue прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

При выполнении инструкции mul esx происходит умножение esx на eax, то есть 4 на 2, вместо умножения 4 на 5 (регистр ebx). Происходит это из-за того, что стоящая перед mov esx,4 инструкция add ebx,eax не связана с mul esx, но связана инструкция mov eax,2 (рис. [4.38]-[??]).

```
zrdagdelen@zrdagdelen: ~/work/arch-pc/lab09
Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd260 0xffffd260
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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

native process 12499 In: _start L12 PC: 0x80490f9
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /home/zrdagdelen/work/arch-pc/lab09/var

Breakpoint 1, _start () at var.asm:8
(gdb) c
Continuing.

Breakpoint 2, _start () at var.asm:12
(gdb) 
```

Рис. 4.37: Пошаговый просмотр работы регистров

```
zrdagdelen@zrdagdelen: ~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd260 0xffffd260
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

B+ 0x80490e8 <_start>    mov     ebx,0x3
0x80490ed <_start+5>    mov     eax,0x2
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>

native process 12499 In: _start L13 PC: 0x80490fb
Continuing.

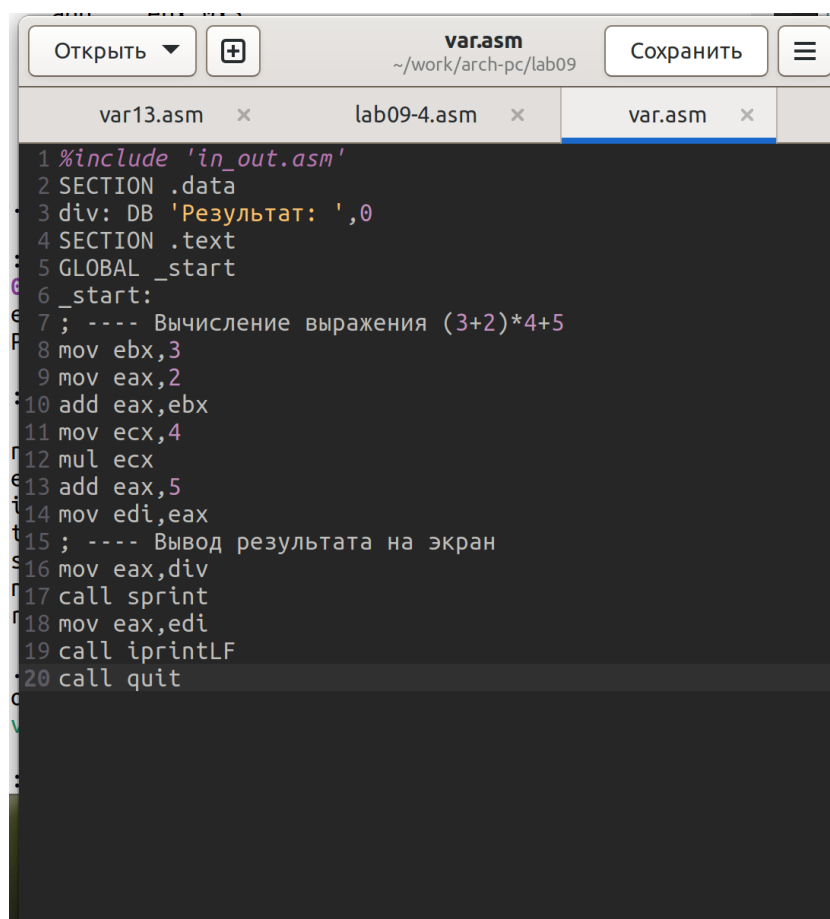
Breakpoint 2, _start () at var.asm:12
(gdb) b *0x80490fb
Breakpoint 3 at 0x80490fb: file var.asm, line 13.
(gdb) c
Continuing.

Breakpoint 3, _start () at var.asm:13
(gdb) 
```

Рис. 4.38: Пошаговый просмотр работы регистров

Из-за этого я получаю неправильный ответ.

Исправляю ошибку, изменив `add ebx,eax` на `add eax,ebx` и заменяя `ebx` на `eax` в инструкциях `add ebx,5` и `mov edi,ebx` (рис. [4.39]).



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

Рис. 4.39: Исправление ошибки

Создаю исполняемый файл и запускаем его. Убеждаюсь, что ошибка исправлена (рис. [4.40]).



```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ nasm -f elf -g -l var.lst var.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ ld -m elf_i386 -o var var.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab09$ gdb var
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
l>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from var...
(gdb) r
Starting program: /home/zrdagdelen/work/arch-pc/lab09/var
Результат: 25
[Inferior 1 (process 13409) exited normally]
(gdb) 

```

Рис. 4.40: 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 eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax

; ---- Вывод результата на экран

```

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

## 5 Выводы

Я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

## **6 Список литературы**

Архитектура ЭВМ