

# **Лабораторная работа №9**

**Понятие подпрограммы.Отладчик GDB**

Баранов Никита Дмитриевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>3</b>	<b>Самостоятельная работа</b>	<b>18</b>
<b>4</b>	<b>Выводы</b>	<b>22</b>

# Список иллюстраций

2.1	Создаем директории и файл . . . . .	7
2.2	Вставляем код в файл . . . . .	8
2.3	Создаем исполняемый файл и проверяем его работу . . . . .	8
2.4	Редактируем программу, добавляя subcalcul . . . . .	9
2.5	Создаем объектный файл и проверяем работу программы . . . . .	9
2.6	Создаем новый файл . . . . .	10
2.7	Вводим программу . . . . .	10
2.8	Загружаем файл в отладчик . . . . .	10
2.9	Проверяем работу программы . . . . .	11
2.10	Запускаем с брейкпоинтом . . . . .	11
2.11	Смотрим диссимилированный код программы . . . . .	11
2.12	Вводим команду set, переключаясь на синтаксис Intel . . . . .	12
2.13	Проверяем точку останова с помощью команды i b и ставим новую точку . . . . .	13
2.14	Смотрим инфо о всех точках останова . . . . .	13
2.15	Выполняем 5 команд командой si . . . . .	14
2.16	Просматриваем содержимое переменной по имени . . . . .	14
2.17	Просматриваем значение по адресу . . . . .	14
2.18	Изменяем первый символ переменной . . . . .	14
2.19	Изменяем первый символ второй переменной . . . . .	15
2.20	Смотрим значения регистра в различных форматах . . . . .	15
2.21	Изменяем значения ebx и смотрим выводы . . . . .	15
2.22	Завершаем выполнение программы и выходим из gdb . . . . .	15
2.23	Копируем программу в новый файл и создаем объектный файл и запускаем его в отладчике GDB . . . . .	16
2.24	Устанавливаем точку останова и запускаем ее . . . . .	16
2.25	Смотрим позиции стека по разным адресам . . . . .	17
3.1	Копируем программу . . . . .	18
3.2	Изменяем программу под условия . . . . .	18
3.3	Создаем объектный файл и проверяем работу программы . . . . .	19
3.4	Вводим программу из листинга в новый файл . . . . .	19
3.5	Создаем объектный файл и запускаем его(есть ошибка) . . . . .	19
3.6	Запускаем файл в отладчике и с помощью команды si находим ошибку, просматривая регистры . . . . .	20
3.7	Редактируем файл, исправляя ошибки . . . . .	20

3.8	Создаем объектный файл и запускаем его(нет ошибки)	21
-----	--	----

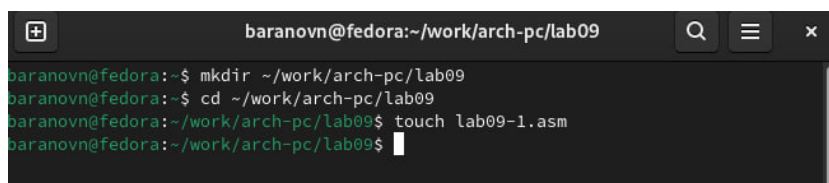
## **Список таблиц**

# 1 Цель работы

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

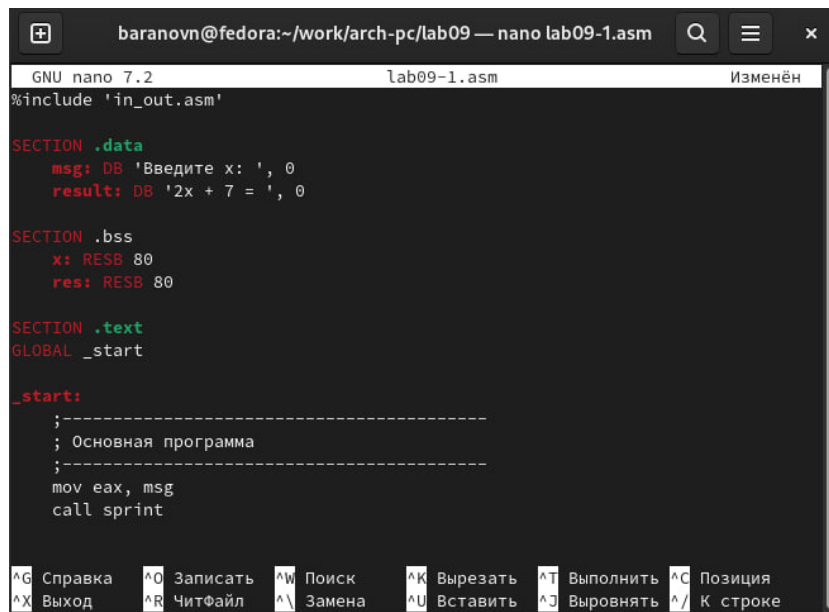
## 2 Выполнение лабораторной работы

Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm. В качестве примера рассмотрим программу вычисления арифметического выражения  $f(x) = 2x + 7$  с помощью подпрограммы `_calcul`. В данном примере  $x$  вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы. Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу. Измените текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения  $f(g(x))$ , где  $x$  вводится с клавиатуры,  $f(x) = 2x + 7$ ,  $g(x) = 3x - 1$ . Т.е.  $x$  передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение  $g(x)$ , результат возвращается в `_calcul` и вычисляется выражение  $f(g(x))$ . Результат возвращается в основную программу для вывода результата на экран. (рис. fig. 2.1)(рис. fig. 2.2)(рис. fig. 2.3)(рис. fig. 2.4)(рис. fig. 2.5).



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

Рис. 2.1: Создаем директорию и файл



```
baranovn@fedora:~/work/arch-pc/lab09 — nano lab09-1.asm
GNU nano 7.2 lab09-1.asm Изменён
%include 'in_out.asm'

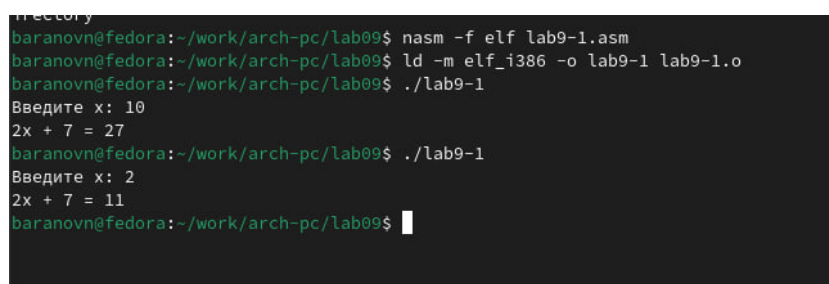
SECTION .data
    msg: DB 'Введите x: ', 0
    result: DB '2x + 7 = ', 0

SECTION .bss
    x: RESB 80
    res: RESB 80

SECTION .text
GLOBAL _start

_start:
;-----
; Основная программа
;-----
    mov eax, msg
    call sprint
```

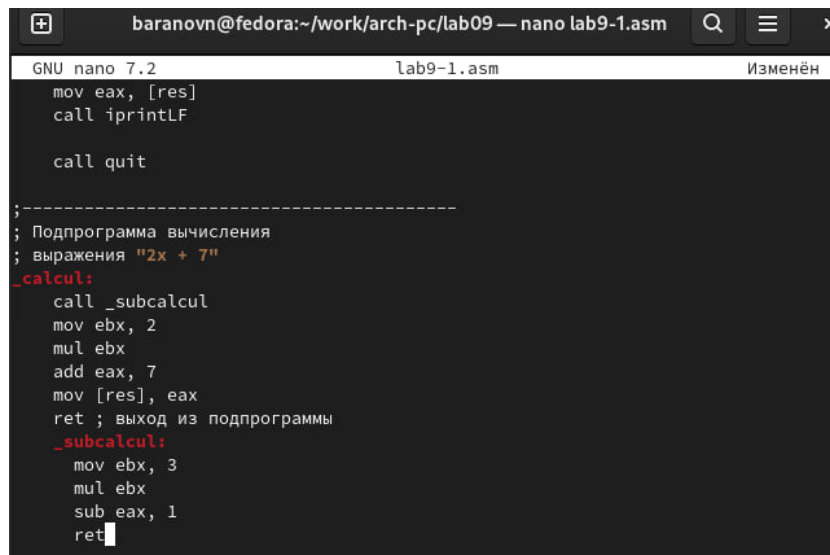
Рис. 2.2: Вставляем код в файл



```
baranovn@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
baranovn@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
baranovn@fedora:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2x + 7 = 27
baranovn@fedora:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 2
2x + 7 = 11
baranovn@fedora:~/work/arch-pc/lab09$
```

Рис. 2.3: Создаем исполняемый файл и проверяем его работу



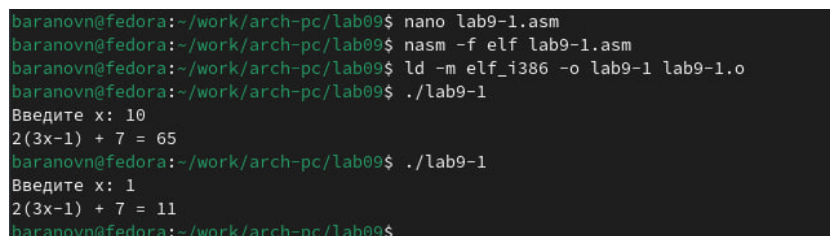


```
baranovn@fedora:~/work/arch-pc/lab09 — nano lab9-1.asm
GNU nano 7.2 lab9-1.asm Изменён
mov eax, [res]
call iprintLF

call quit

;-----
; Подпрограмма вычисления
; выражения "2x + 7"
_calcul:
    call _subcalcul
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret ; выход из подпрограммы
_subcalcul:
    mov ebx, 3
    mul ebx
    sub eax, 1
    ret
```

Рис. 2.4: Редактируем программу, добавляя subcalcul



```
baranovn@fedora:~/work/arch-pc/lab09$ nano lab9-1.asm
baranovn@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
baranovn@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
baranovn@fedora:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2(3x-1) + 7 = 65
baranovn@fedora:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 1
2(3x-1) + 7 = 11
baranovn@fedora:~/work/arch-pc/lab09$
```

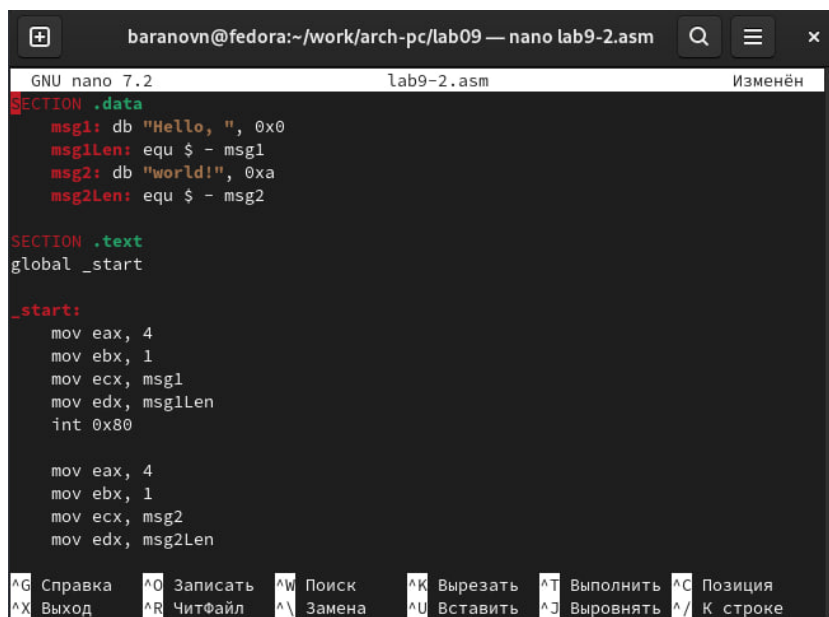
Рис. 2.5: Создаем объектный файл и проверяем работу программы

Создайте файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world! Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Загрузите исполняемый файл в отладчик gdb. Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r). Для более подробного анализа программы установите брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дисассемблированный код программы с помощью команды disassemble начиная с метки \_start. Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel. Перечислите различия

отображения синтаксиса машинных команд в режимах ATТ и Intel. Включите режим псевдографики для более удобного анализа программы(рис. fig. 2.6)(рис. fig. 2.7)(рис. fig. 2.8)(рис. fig. 2.9)(рис. fig. 2.10)(рис. fig. 2.11)(рис. fig. 2.12)

```
baranov@fedora:~/work/arch-pc/lab09$ touch lab9-2.asm
baranov@fedora:~/work/arch-pc/lab09$
```

Рис. 2.6: Создаем новый файл



```
GNU nano 7.2 lab9-2.asm Изменён
SECTION .data
msg1: db "Hello, ", 0x0
msg1Len: equ $ - msg1
msg2: db "world!", 0xa
msg2Len: equ $ - msg2

SECTION .text
global _start

_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, msg1
    mov edx, msg1Len
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, msg2
    mov edx, msg2Len
```

Рис. 2.7: Вводим программу

```
baranov@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
baranov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
baranov@fedora:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.2-1.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 lab9-2...
(gdb)
```

Рис. 2.8: Загружаем файл в отладчик

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

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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 6452) exited normally]
(gdb)
```

Рис. 2.9: Проверяем работу программы

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 11.
(gdb) run
Starting program: /home/baranovn/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:11
11      mov eax, 4
(gdb)
```

Рис. 2.10: Запускаем с брейкпоинтом

```
(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     $0x804a000,%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. 2.11: Смотрим дисассемблированный код программы

```

(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. 2.12: Вводим команду set, переключаясь на синтаксис Intel

Различия в отображении синтаксиса: Порядок операндов: Intel: Операнды записываются в порядке destination, source. АТТ: Операнды записываются в порядке source, destination. Суффиксы: Intel: Суффиксы для указания размера данных не используются. АТТ: Используются суффиксы для указания размера данных. Регистры: Intel: Регистры указываются без каких-либо префиксов. АТТ: Регистры указываются с префиксом. Константы: Intel: Константы записываются без специальных символов. АТТ: Константы обозначаются с помощью \$.

Включаем режим псевдографики. На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверьте это с помощью команды info breakpoints (кратко i b). Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3). Определите адрес предпоследней инструкции (mov ebx,0x0) и установите точку останова. Посмотрите информацию о всех установленных точках останова (рис. 2.13) (рис. 2.14)

```

baranovn@fedora:~/work/arch-pc/lab09 — gdb lab9-2
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
    b+ 0x8049031 <_start+49> mov ebx,0x0
    0x8049036 <_start+54> int 0x80

native process 6475 (asm) In: _start L11 PC: 0x8049000
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab9-2.asm:11
          breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb)

```

Рис. 2.13: Проверяем точку останова с помощью команды `i b` и ставим новую точку

```

Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab9-2.asm:11
          breakpoint already hit 1 time
2        breakpoint keep y  0x08049031 lab9-2.asm:24
(gdb)

```

Рис. 2.14: Смотрим инфо о всех точках останова

Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Значения каких регистров изменяются? Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`). Посмотрите значение переменной `msg1` по имени. Посмотрите значение переменной `msg2` по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию `mov ecx,msg2` которая записывает в регистр `ecx` адрес переменной `msg2`. Измените первый символ переменной `msg1`. Замените любой символ во второй переменной `msg2`. Выведите в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`. С помощью команды `set` измените значение регистра `ebx`. Объясните разницу вывода команд

p/s \$ebx. Завершите выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдите из GDB с помощью команды quit (сокращенно q)(рис. fig. 2.15)(рис. fig. 2.16)(рис. fig. 2.17)(рис. fig. 2.18)(рис. fig. 2.19)(рис. fig. 2.20)(рис. fig. 2.21)(рис. fig. 2.22)

```

baranovn@fedora:~/work/arch-pc/lab09 — gdb lab9-2
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
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54>   int     0x80

native process 6475 (asm) In: _start          L17  PC: 0x8049016
breakpoint already hit 1 time
breakpoint keep y 0x8049031 lab9-2.asm:24
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.15: Выполняем 5 команд командой si

Изменяются значения регистров - eax,ebx,ecx,edx,eip

```

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

```

Рис. 2.16: Просматриваем содержимое переменной по имени

```

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

```

Рис. 2.17: Просматриваем значение по адресу

```

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

```

Рис. 2.18: Изменяем первый символ переменной

```
(gdb) set {char}&msg2='L'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lor!d!\n\034"
(gdb)
```

Рис. 2.19: Изменяем первый символ второй переменной

```
(gdb) p/t $edx
$1 = 1000
(gdb) p/s $edx
$2 = 8
(gdb) p/x $edx
$3 = 0x8
(gdb)
```

Рис. 2.20: Смотрим значения регистра в различных форматах

```
(gdb) p/s $ebx
$4 = 50
(gdb) set &ebx =2
No symbol "ebx" in current context.
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 2.21: Изменяем значения ebx и смотрим выводы

Выводятся разные значения т.к. команда без кавычек присваивает регистру вводимое значение.

```
(gdb) c
Continuing.
Lor!d!

Breakpoint 2, _start () at lab9-2.asm:24
(gdb)
```

Рис. 2.22: Завершаем выполнение программы и выходим из gdb

Скопируйте файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводющей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm. Создайте исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки



(включая имя программы).Посмотрите 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.Объясните, почему шаг изменения адреса равен 4 (`esp+4`, `esp+8`, `esp+12` и т.д.)(рис. fig. 2.23)(рис. fig. 2.24)(рис. fig. 2.25)

```

baranov@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/
arch-pc/lab09/lab9-3.asm
baranov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
baranov@fedora:~/work/arch-pc/lab09$ gdb --args lab9-3 1 5 '10'
GNU gdb (Fedora Linux) 15.2-1.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 lab9-3...
(gdb)

```

Рис. 2.23: Копируем программу в новый файл и создаем объектный файл и запускаем его в отладчике GDB

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/baranov/work/arch-pc/lab09/lab9-3 1 5 10

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 lab9-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd060: 0x00000004
(gdb)

```

Рис. 2.24: Устанавливаем точку останова и запускаем ее



```
(gdb) x/s *(void**)(esp + 4)
0xffffd220:  "/home/baranovn/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd24f:  "1"
(gdb) x/s *(void**)(esp + 12)
0xffffd251:  "5"
(gdb) ) x/s *(void**)(esp + 16)
Undefined command: ". Try "help".
(gdb) x/s *(void**)(esp + 16)
0xffffd253:  "10"
(gdb) x/s *(void**)(esp + 20)
0x0:  <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)(esp + 24)
0xffffd256:  "SHELL=/bin/bash"
(gdb) █
```

Рис. 2.25: Смотрим позиции стека по разным адресам

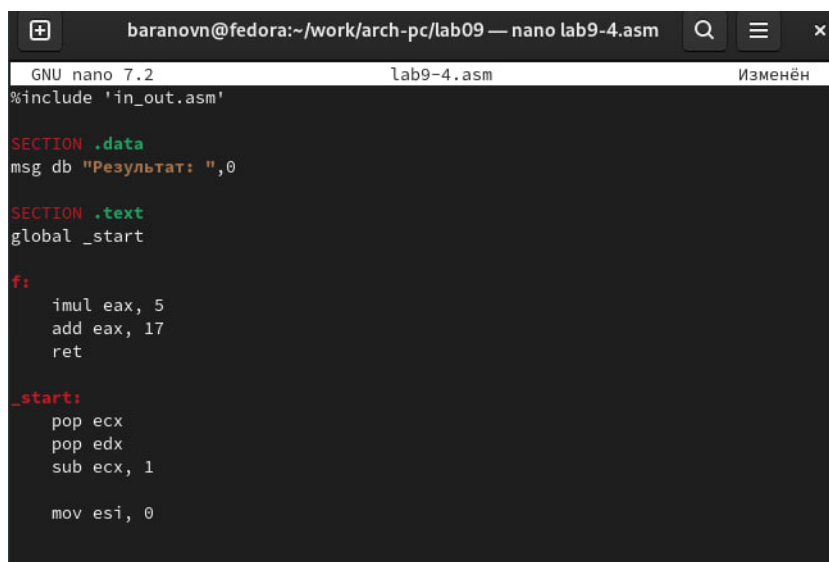
Шаг изменения равен 4 потому что регистры имеют размерность в 4 байта.

## 3 Самостоятельная работа

Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму. В листинге 9.3 приведена программа вычисления выражения  $(3 + 2) \cdot 4 + 5$ . При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.(рис. fig. 3.1)(рис. fig. 3.2)(рис. fig. 3.3)(рис. fig. 3.4)(рис. fig. 3.5)(рис. fig. 3.6)(рис. fig. 3.7)(рис. fig. 3.8)

```
baranovn@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab9-4.asm
```

Рис. 3.1: Копируем программу



```
baranovn@fedora:~/work/arch-pc/lab09 — nano lab9-4.asm
GNU nano 7.2 lab9-4.asm Изменён
#include 'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start

f:
    imul eax, 5
    add eax, 17
    ret

_start:
    pop ecx
    pop edx
    sub ecx, 1

    mov esi, 0
```

Рис. 3.2: Изменяем программу под условия

```

baranov@fedora:~/work/arch-pc/lab09$ nano lab9-4.asm
baranov@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
baranov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
baranov@fedora:~/work/arch-pc/lab09$ ./lab9-4 5 7 8
Результат: 151
baranov@fedora:~/work/arch-pc/lab09$ ./lab9-4 10 65 99
Результат: 921
baranov@fedora:~/work/arch-pc/lab09$

```

Рис. 3.3: Создаем объектный файл и проверяем работу программы

```

GNU nano 7.2 lab9-5.asm
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

<sup>^</sup>G Справка    <sup>^</sup>O Записать    <sup>^</sup>W Поиск    <sup>^</sup>K Вырезать    <sup>^</sup>T Выполнить    <sup>^</sup>C Позиция  
<sup>^</sup>X Выход    <sup>^</sup>R ЧитФайл    <sup>^</sup>\ Замена    <sup>^</sup>U Вставить    <sup>^</sup>J Выровнять    <sup>^</sup>/ К строке

Рис. 3.4: Вводим программу из листинга в новый файл

```

baranov@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
baranov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
baranov@fedora:~/work/arch-pc/lab09$ ./lab9-5
Результат: 10

```

Рис. 3.5: Создаем объектный файл и запускаем его(есть ошибка)

The screenshot shows a GDB window titled 'baranov@fedora:~/work/arch-pc/lab09 — gdb --args lab9-5'. The 'Registers' pane shows the following values:

Register	Value	Comment
eax	0x804a001	134520833
ecx	0x4	4
edx	0x0	[ Register Values Unavailable ]
ebx	0x804a000	134520832
esp	0xffffd054	0xffffd054
ebp	0x0	0x0

The 'Disassembly' pane shows the following assembly code:

```

8  cmp     byte [eax], 0
9  jz      finished
10 inc     eax
> 11 jmp    nextchar
12

```

The 'Console' pane shows the following output:

```

native p13 finished: L?? PC: ??
starting process 7546 (src) In: /work/arch-pc/lab09/lab9-5
to make process 7546 (src) In: nextchar debuginfo enabled L11 PC: 0x8049009
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Рис. 3.6: Запускаем файл в отладчике и с помощью команды `si` находим ошибку, просматривая регистры

The screenshot shows a nano text editor window titled 'baranov@fedora:~/work/arch-pc/lab09 — nano lab9-5.asm'. The editor contains the following assembly code:

```

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov eax,3
    mov ebx,2
    add eax,ebx
    mov ecx,4
    mul ecx
    add eax,5
    mov edi,eax
    mov eax,div
    call sprint
    mov eax,edi
    call iprintLF
    call quit

```

The status bar at the bottom shows the file name 'lab9-5.asm' and various menu options.

Рис. 3.7: Редактируем файл, исправляя ошибки

```
baranov@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
baranov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
baranov@fedora:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
baranov@fedora:~/work/arch-pc/lab09$
```

Рис. 3.8: Создаем объектный файл и запускаем его(нет ошибки)

## **4 Выводы**

Мы познакомились с методами отладки при помощи GDB и его возможностями.