

# **Отчёт по лабораторной работе №9**

**дисциплина: архитектура компьютеров**

Ведьмина Александра Сергеевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Выводы</b>	<b>24</b>

## Список иллюстраций

4.1	Создание файла lab09-1.asm . . . . .	8
4.2	Ввод программы в lab09-1.asm . . . . .	9
4.3	Запуск lab09-1 . . . . .	9
4.4	Изменение программы в lab09-1.asm . . . . .	10
4.5	Запуск файла lab09-1 . . . . .	10
4.6	Ввод программы в lab09-2.asm . . . . .	11
4.7	Загрузка lab09-2 в gdb . . . . .	12
4.8	Запуск lab09-2 . . . . .	12
4.9	Повторный запуск lab09-2 . . . . .	12
4.10	Открытие дисассимилированного кода программы . . . . .	13
4.11	Включение режима псевдографики . . . . .	14
4.12	Вывод точек останова . . . . .	14
4.13	Установка второй точки останова . . . . .	15
4.14	Вывод текущих значений регистров . . . . .	15
4.15	Значение msg1 . . . . .	15
4.16	Значение msg2 . . . . .	16
4.17	Изменение символа в msg1 . . . . .	16
4.18	Изменение символа в msg2 . . . . .	16
4.19	Вывод регистра в разных форматах . . . . .	17
4.20	Изменение значения ebx . . . . .	17
4.21	Повторное изменение значения ebx . . . . .	18
4.22	Копирование lab8-2.asm . . . . .	18
4.23	Создание исполняемого файла lab09-3 . . . . .	18
4.24	Передача файла в gdb . . . . .	19
4.25	Установка точки останова . . . . .	19
4.26	Вывод адреса стека . . . . .	20
4.27	Вывод отдельных позиций стека . . . . .	20
4.28	Программа в sumrub-1.asm . . . . .	21
4.29	Запуск sumrub-1 . . . . .	22
4.30	Анализ значений регистров . . . . .	22
4.31	Исправление ошибки в программе . . . . .	23
4.32	Запуск sumrub-2 . . . . .	23

## **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. Изучить понятие процесса отладки
2. Познакомиться с отладчиком gdb
3. Изучить листинг с подпрограммой
4. Сделать задания для самостоятельной работы

### 3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Этапы отладки: 1. обнаружение ошибки 2. поиск её местонахождения 3. определение причины ошибки 4. исправление ошибки

Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы. GDB (GNU Debugger — отладчик проекта GNU, который работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. 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.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. Применяется, если есть одинаковые участки кода. Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `esp` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы.

## 4 Выполнение лабораторной работы

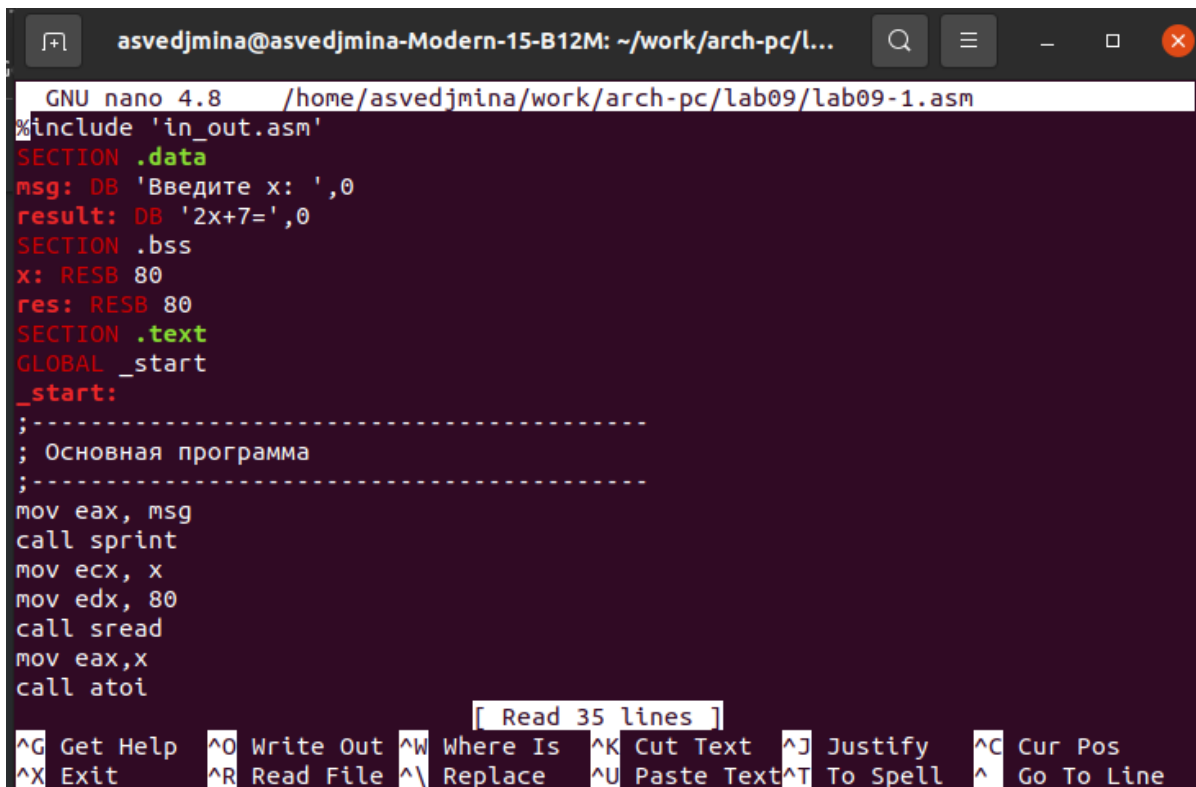
Создаю каталог lab09, перехожу в него и создаю там файл lab09-1.asm.

```
asvedjmina@asvedjmina-Modern-15-B12M:~$ mkdir ~/work/arch-pc/lab09
asvedjmina@asvedjmina-Modern-15-B12M:~$ cd ~/work/arch-pc/lab09
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ touch lab09-1.asm
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание файла lab09-1.asm

Ввожу в данный файл текст программы, использующую вызов подпрограммы.

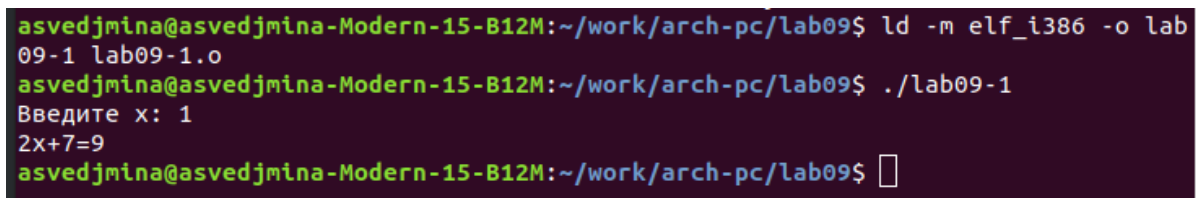




```
GNU nano 4.8 /home/asvedjmina/work/arch-pc/lab09/lab09-1.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
```

Рис. 4.2: Ввод программы в lab09-1.asm

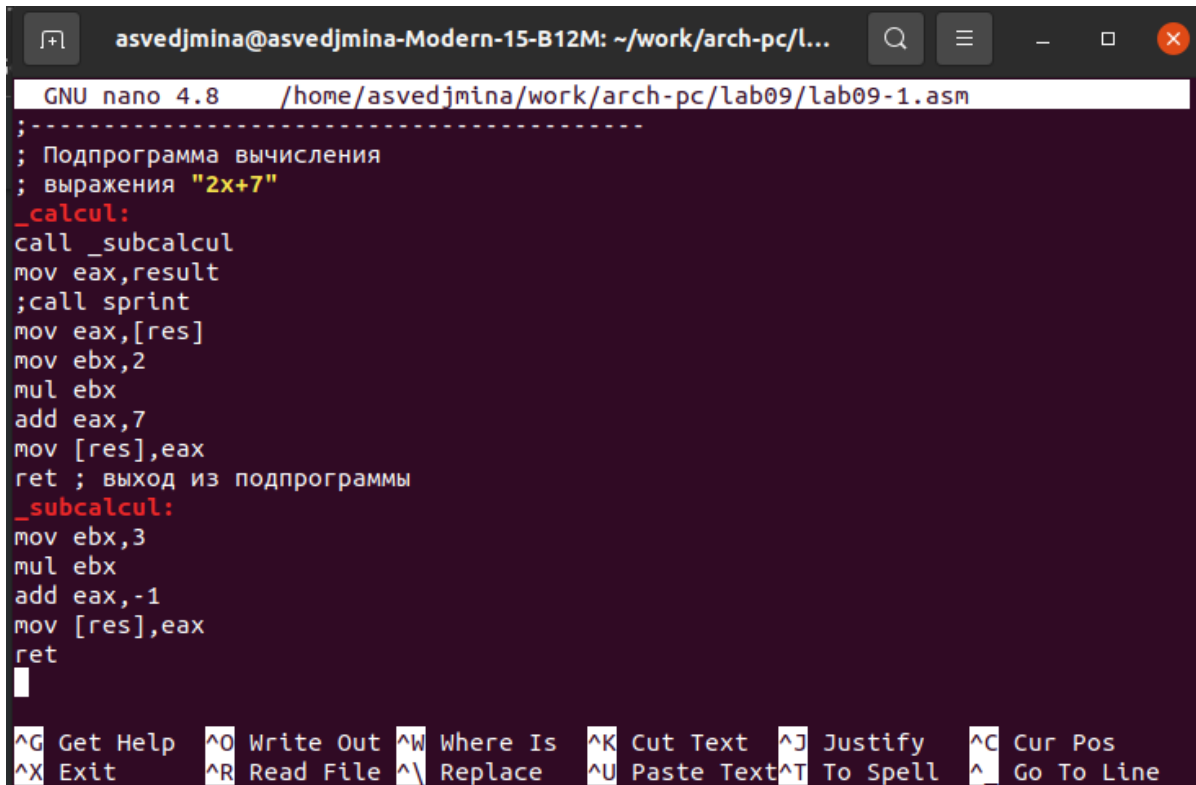
Создаю исполняемый файл и запускаю его.



```
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 1
2x+7=9
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$
```

Рис. 4.3: Запуск lab09-1

Затем изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, чтобы `_subcalcul` вычисляла выражение  $3 \cdot x - 1$ .

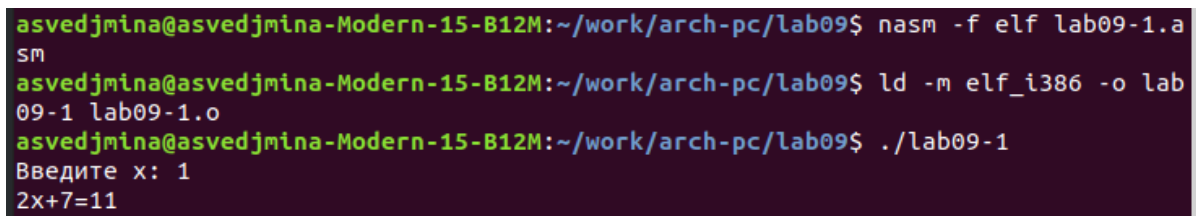


```
GNU nano 4.8 /home/asvedjmina/work/arch-pc/lab09/lab09-1.asm
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul
mov eax,result
;call sprint
mov eax,[res]
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx,3
mul ebx
add eax,-1
mov [res],eax
ret

```

Рис. 4.4: Изменение программы в lab09-1.asm

Создаю исполняемый файл и запускаю его.

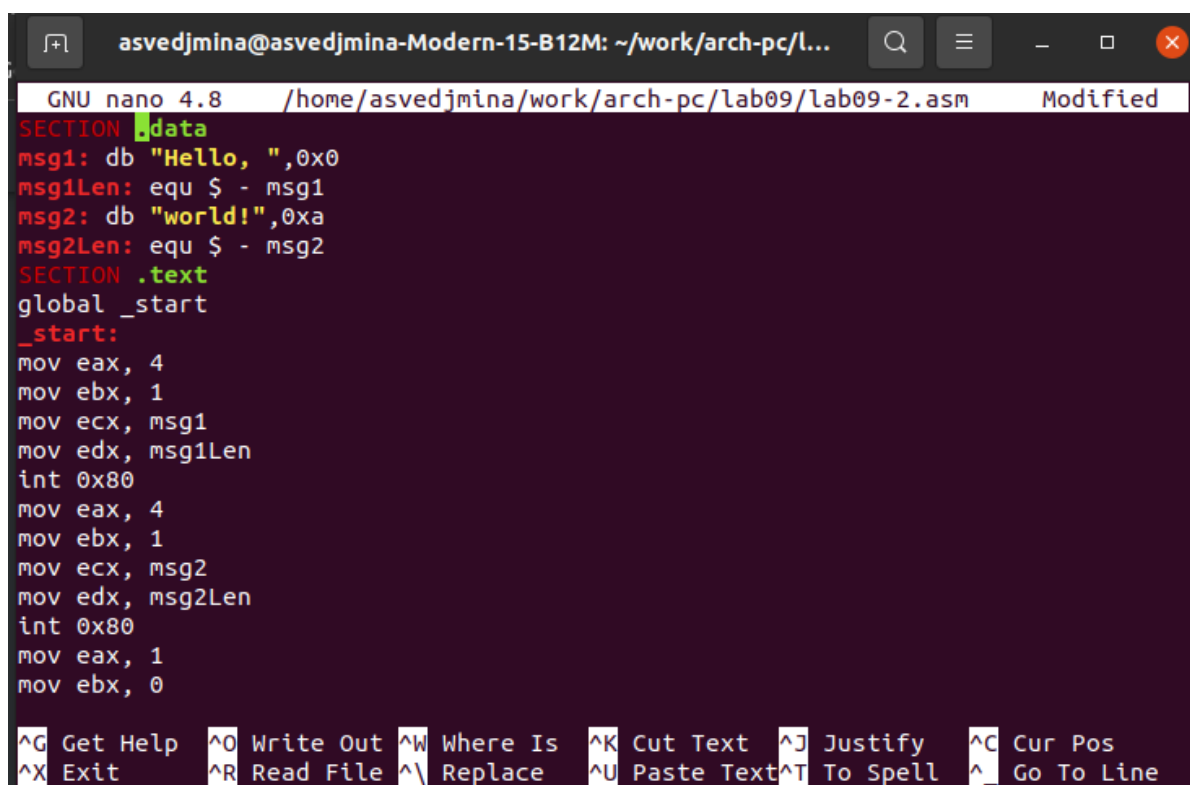


```
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 1
2x+7=11

```

Рис. 4.5: Запуск файла lab09-1

Создаю файл lab09-2.asm, ввожу в него текст программы с выводом сообщения Hello world!



```
GNU nano 4.8 /home/asvedjmina/work/arch-pc/lab09/lab09-2.asm Modified
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
int 0x80
mov eax, 1
mov ebx, 0

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line
```

Рис. 4.6: Ввод программы в lab09-2.asm

Получаю исполняемый файл, необходимый для работы с gdb, и загружаю его в отладчик.

```

asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<http://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)
(gdb)

```

Рис. 4.7: Загрузка lab09-2 в gdb

Запускаю программу в отладчике.

```

(gdb) run
Starting program: /home/asvedjmina/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 26348) exited normally]
(gdb)

```

Рис. 4.8: Запуск lab09-2

Устанавливаю брейкпоинт на метку `_start`, после чего вновь запускаю программу.

```

(gdb) run
Starting program: /home/asvedjmina/work/arch-pc/lab09/lab09-2

Breakpoint 1, 0x08049000 in _start ()
(gdb)

```

Рис. 4.9: Повторный запуск lab09-2

Открываю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, а затем переключаюсь на отображение с Intel'овским синтаксисом. В машинном коде используются `$` и `%` для отображения регистров.

```
Breakpoint 1, 0x08049000 in _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.
(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. 4.10: Открытие дисассимилированного кода программы

Далее включаю режим псевдографики.

```
asvedjmina@asvedjmina-Modern-15-B12M: ~/work/arch-pc/l...
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0d0 0xffffd0d0
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

native process 28289 In: _start L?? PC: 0x8049000
(gdb) layout regs
(gdb) 
```

Рис. 4.11: Включение режима псевдографики

Вывожу информацию о всех точках останова.

```
      0x8049025 <_start+37> mov    edx,0x7
      0x804902a <_start+42> int     0x80

native process 28289 In: _start L?? PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 <_start>
breakpoint already hit 1 time
(gdb) 
```

Рис. 4.12: Вывод точек останова

Устанавливаю вторую точку останова по адресу предпоследней инструкции.

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031
(gdb) █
```

Рис. 4.13: Установка второй точки останова

С помощью команды `info registers`, вывожу текущие значения регистров.

```
native process 28289 In: start L?? PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0d0 0xffffd0d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging-- █
```

Рис. 4.14: Вывод текущих значений регистров

Затем смотрю значение переменной `msg1`.

```
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) █
```

Рис. 4.15: Значение `msg1`

Смотрю значение переменной `msg2`.

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

Рис. 4.16: Значение msg2

Изменяю первый символ переменной msg1 с помощью команды set.

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

Рис. 4.17: Изменение символа в msg1

Заменяю также символ и в переменной msg2.

```
(gdb) set {char}&msg2='t'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "torld!\n"
(gdb) █
```

Рис. 4.18: Изменение символа в msg2

Вывожу значение регистра eax в шестнадцатеричном формате, в двоичном формате и в символьном виде.



```
(gdb) p/t $eax
$7 = 100
(gdb) p/x $eax
$8 = 0x4
(gdb) p/c $eax
$9 = 4 '\004'
(gdb) 
```

Рис. 4.19: Вывод регистра в разных форматах

С помощью команды set изменяю значение регистра ebx.

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$10 = 50
(gdb) 
```

Рис. 4.20: Изменение значения ebx

Вновь изменяю значение этого регистра.

```
(gdb) set $ebx=2
(gdb) p/s $ebx
$11 = 2
(gdb) 
```

Рис. 4.21: Повторное изменение значения ebx

Разница в том, что в первый раз я записала в ebx символ, а во второй раз - число.

Завершаю выполнение программы с помощью команды continue и выхожу из gdb, используя команду quit. После этого копирую файл lab8-2.asm в lab09-3.asm.

```
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ 
```

Рис. 4.22: Копирование lab8-2.asm

Создаю исполняемый файл.

```
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ 
```

Рис. 4.23: Создание исполняемого файла lab09-3

Передаю файл в gdb с тремя аргументами.

```

asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ gdb --args lab09-3 1
2 't'
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<http://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)

```

Рис. 4.24: Передача файла в gdb

Ставлю точку останова перед первой инструкцией и запускаю программу.

```

(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) run
Starting program: /home/asvedjmina/work/arch-pc/lab09/lab09-3 1 2 t

Breakpoint 1, 0x080490e8 in _start ()
(gdb)

```

Рис. 4.25: Установка точки останова

По адресу вершины стека узнаю количество аргументов, переданное программе (включая имя программы).

```
Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd0b0:      0x00000004
(gdb) x/s *(void**)(esp + 4)
```

Рис. 4.26: Вывод адреса стека

Затем смотрю отдельные позиции стека.

```
(gdb) x/s *(void**)(esp + 4)
0xffffd29f:      "/home/asvedjmina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2cb:      "1"
(gdb) x/s *(void**)(esp + 12)
0xffffd2cd:      "2"
(gdb) x/s *(void**)(esp + 16)
0xffffd2cf:      "t"
(gdb)
```

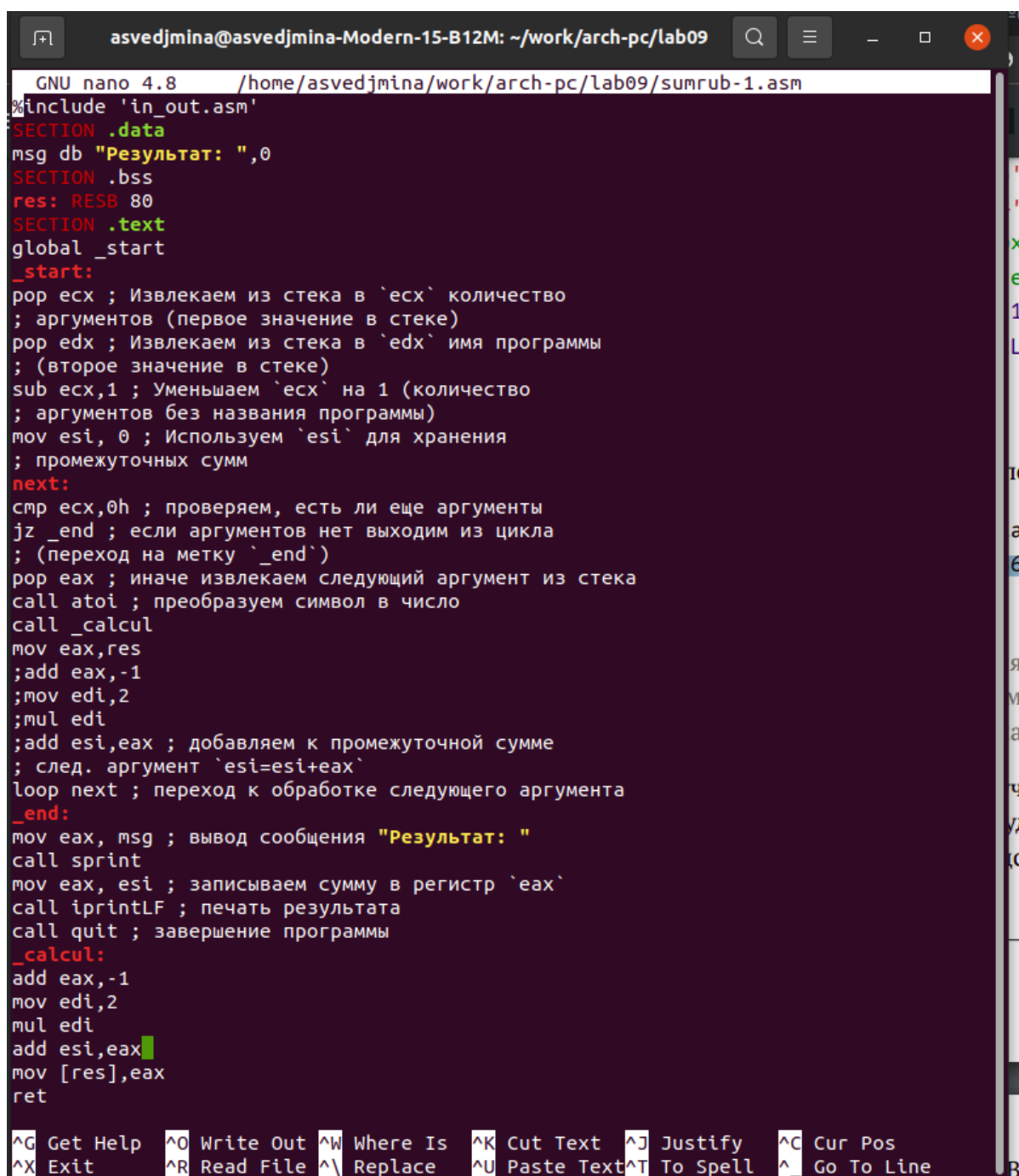
Рис. 4.27: Вывод отдельных позиций стека

Шаг адреса равен 4, так как размер переменных составляет 4 байта.

#Выполнение заданий для самостоятельной работы

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $\text{sumrub}(x)$  как подпрограмму.

Для выполнения задания создаю файл `sumrub-1.asm`, копирую в него текст программы из файла с заданием 1 из 8 лабораторной работы. После этого реализую вычисление функции как подпрограмму.



```
GNU nano 4.8 /home/asvedjmina/work/arch-pc/lab09/sumrub-1.asm
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .bss
res: RESB 80
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _calcul
mov eax,res
;add eax,-1
;mov edi,2
;mul edi
;add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
_calcul:
add eax,-1
mov edi,2
mul edi
add esi,eax
mov [res],eax
ret

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line
```

Рис. 4.28: Программа в sumrub-1.asm

Создаю исполняемый файл и запускаю его.

```

asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ nasm -f elf sumrub-1.asm
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ld -m elf_i386 -o sumrub
-1 sumrub-1.o
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ./sumrub-1
Результат: 0
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ./sumrub-1 1 2 3
Результат: 6
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$

```

Рис. 4.29: Запуск sumrub-1

2. В листинге 9.3 приведена программа вычисления выражения  $(3 + 2) \times 4 + 5$ . При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

Для выполнения задания создаю файл sumrub-2.asm. Затем с помощью gdb изучаю значения регистров.

```

Register group: general
eax      0x804a000      134520832
ecx      0x4           4
edx      0x0           0
ebx      0xa           10
esp      0xffffd0b4    0xffffd0b4

0x8049000 <slen>      push    ebx
>0x8049001 <slen+1>    mov     ebx, eax
0x8049003 <nextchar>  cmp     BYTE PTR [eax], 0x0
0x8049006 <nextchar+3> je      0x804900b <finished>
0x8049008 <nextchar+5> inc     eax
0x8049009 <nextchar+6> jmp     0x8049003 <nextchar>

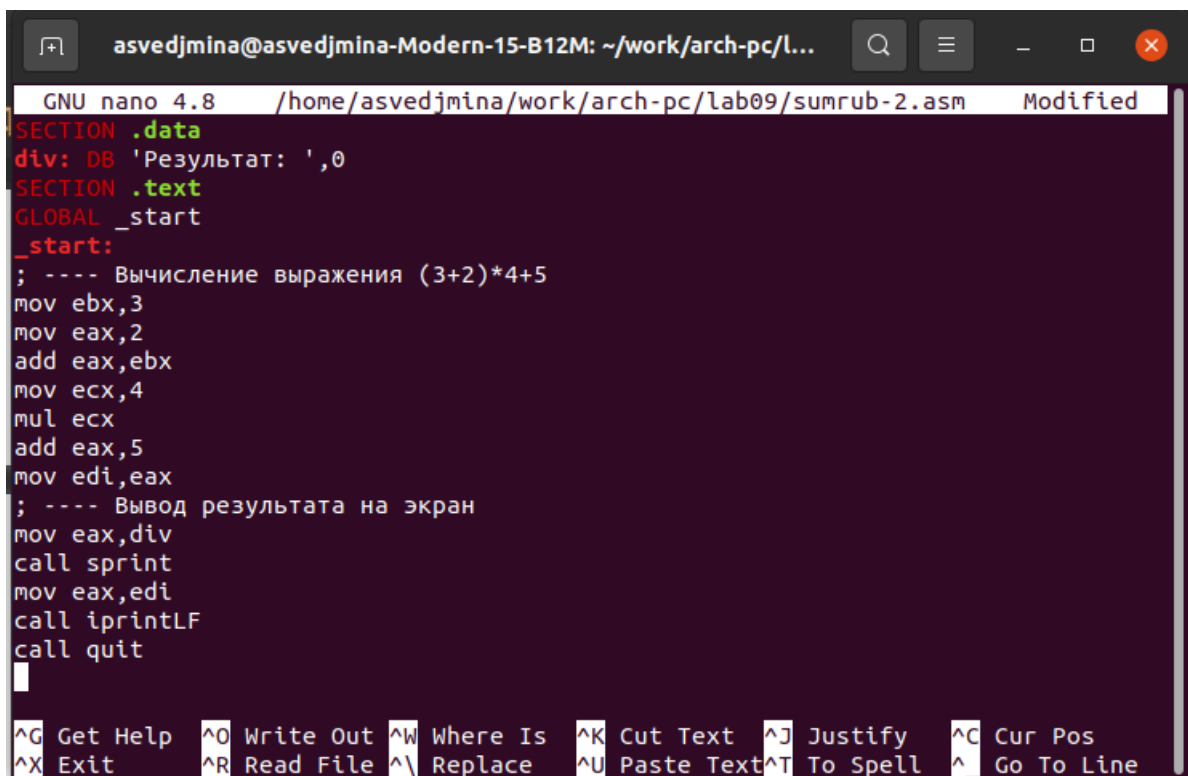
native process 3868 In: slen          L??  PC: 0x8049001
(gdb) si
0x08049013 in sprint ()
(gdb) si
0x08049000 in slen ()
(gdb) si
0x08049001 in slen ()
(gdb)

```

Рис. 4.30: Анализ значений регистров

Узнаю, что в тексте программы регистры перепутаны местами и исправляю

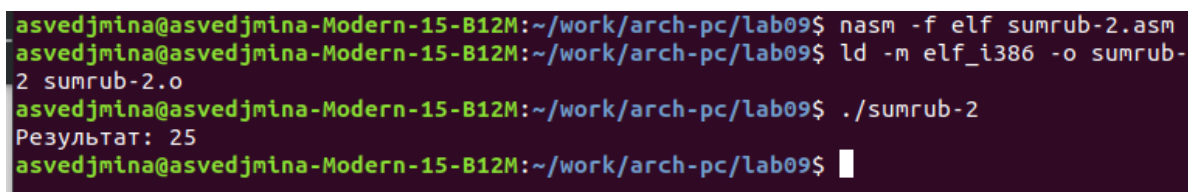
ошибку.



```
GNU nano 4.8 /home/asvedjmina/work/arch-pc/lab09/sumrub-2.asm Modified
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
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line
```

Рис. 4.31: Исправление ошибки в программе

Для проверки создаю исполняемый файл и запускаю его.



```
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ nasm -f elf sumrub-2.asm
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ld -m elf_i386 -o sumrub-2 sumrub-2.o
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$ ./sumrub-2
Результат: 25
asvedjmina@asvedjmina-Modern-15-B12M:~/work/arch-pc/lab09$
```

Рис. 4.32: Запуск sumrub-2

## 5 Выводы

В ходе лабораторной работы я научилась использовать подпрограммы и работать с gdb.