

# **Отчёт по лабораторной работе №9**

**Дисциплина: Архитектура компьютера**

Абакумов Тимофей Александрович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>6</b>
<b>2</b>	<b>Задание</b>	<b>7</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
3.1	Реализация подпрограмм в NASM . . . . .	8
3.2	Отладка программ с помощью GDB . . . . .	14
3.3	Задания для самостоятельной работы . . . . .	25
<b>4</b>	<b>Выводы</b>	<b>31</b>

# Список иллюстраций

3.1	Создание каталога и файла . . . . .	8
3.2	Код программы . . . . .	9
3.3	Работа программы . . . . .	11
3.4	Код программы . . . . .	12
3.5	Работа программы . . . . .	14
3.6	Создание файла . . . . .	14
3.7	Код программы . . . . .	14
3.8	Работа программы . . . . .	16
3.9	Подробный анализ программы . . . . .	16
3.10	Дисассимилированный код . . . . .	17
3.11	Intel'овский синтаксис . . . . .	18
3.12	Режим псевдографики . . . . .	18
3.13	Проверка точки . . . . .	19
3.14	Установка точки . . . . .	19
3.15	Проверка информации . . . . .	20
3.16	Изменение регистров . . . . .	20
3.17	Значение переменной msg1 . . . . .	20
3.18	Значение переменной msg2 . . . . .	21
3.19	Изменение значение переменной msg1 . . . . .	21
3.20	Изменение значение переменной msg2 . . . . .	21
3.21	Значения регистров . . . . .	22
3.22	Изменение значения регистра . . . . .	23
3.23	Завершение работы с файлом . . . . .	23
3.24	Копирование и создание исполняемого файла . . . . .	23
3.25	Загрузка файла в отладчик . . . . .	24
3.26	Запуск программы . . . . .	24
3.27	Проверка адреса . . . . .	24
3.28	Просмотр всех позиций стека . . . . .	25
3.29	Создание файла . . . . .	25
3.30	Код программы . . . . .	26
3.31	Работа программы . . . . .	26
3.32	Создание файла . . . . .	26
3.33	Код программы . . . . .	27
3.34	Работа программы . . . . .	28
3.35	Запуск программы в отладчике . . . . .	29
3.36	Изменение регистров . . . . .	30

3.37 Работа программы . . . . .	30
---------------------------------	----

## **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

### **Порядок выполнения лабораторной работы**

1. Реализация подпрограмм в NASM.
2. Отладка программ с помощью GDB.
3. Выполнение заданий для самостоятельной работы.

## 3 Выполнение лабораторной работы

### 3.1 Реализация подпрограмм в NASM

1. Создадим каталог для выполнения лабораторной работы № 9, перейдите в него и создадим файл lab09-1.asm (рис. 3.1).

```
taabakumov@dk3n35 ~ $ mkdir ~/work/arch-pc/lab09
taabakumov@dk3n35 ~ $ cd ~/work/arch-pc/lab09
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ touch lab09-1.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Создание каталога и файла

2. 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. Введём текст программы из Листинга 9.1 (рис. 3.2).



```

lab09-1.asm      [----]  9 L:[ 1+ 0  1/ 35] *(9  / 707b)
#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
call _calcul ; Вызов подпрограммы _calcul
mov eax, result

```

Рис. 3.2: Код программы

Код программы из пункта 2:

```

#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:
mov eax, msg(рис. 3.2).

```

```

lab09-1.asm      [----]  9 L:[ 1+ 0  1/ 35] *(9  / 707b)
%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
call _calcul ; Вызов подпрограммы _calcul
mov eax, result

```

заданий для самосто-

ятельной работы.

call sprint

mov ecx, x

mov edx, 80

call sread

mov eax, x

call atoi

call \_calcul ; Вызов подпрограммы \_calcul

mov eax, result

call sprint

mov eax, [res]

call iprintLF

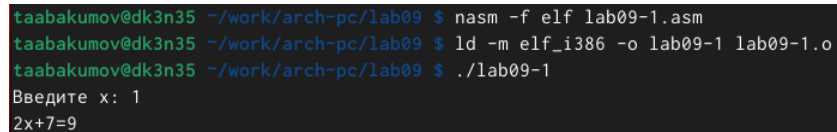
call quit

\_calcul:

mov ebx, 2

```
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
```

3. Создадим исполняемый файл и проверим его работу (рис. 3.3).



```
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
2x+7=9
```

Рис. 3.3: Работа программы

4. Изменим 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. 3.4).

```

lab09-1.asm      [-M--]  3 L:[ 35+19  54/ 56] *(538 /
mov  eax,result
call sprint
mov  eax,[res]
call iprintLF

call quit

_calcul:

call _subcalcul

mov  ebx,2
mul  ebx
add  eax,7
mov  [res],eax
ret

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

```

Рис. 3.4: Код программы

Код программы из пункта 4:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg: DB 'Введите x:',0
```

```
prim1: DB 'f(x) = 2x+7',0
```

```
prim2: DB 'g(x) = 3x-1',0
```

```
result: DB 'f(g(x))=',0
```

```
SECTION .bss
```

```
x: RESB 80
```

```
res: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```

mov eax,prim1
call sprintLF
mov eax,prim2
call sprintLF
mov eax,msg
call sprint
mov ecx,x
mov edx,80
call sread
mov eax,x
call atoi
call _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7 заданий для самостоятельной работы. mov [res],eax
ret
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
ret

```

5. Создадим исполняемый файл и проверим его работу (рис. 3.5).

```

taabakumov@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ./lab09-1
f(x) = 2x+7
g(x) = 3x-1
Введите x: 1
f(g(x))= 11

```

Рис. 3.5: Работа программы

## 3.2 Отладка программ с помощью GDB

6. Создадим файл lab09-2.asm (рис. 3.6).

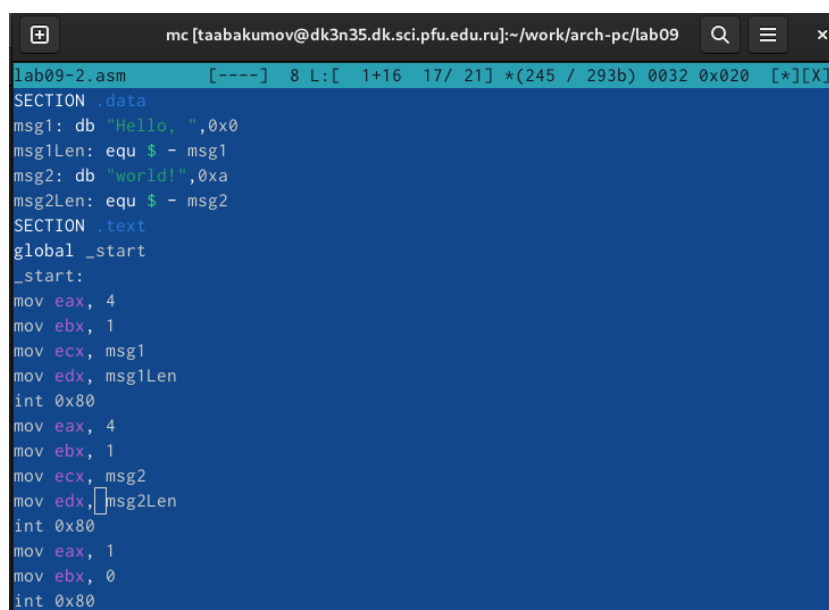
```

taabakumov@dk3n35 ~/work/arch-pc/lab09 $ touch lab09-2.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ 

```

Рис. 3.6: Создание файла

7. Введём в файл lab09-2.asm текст из Лисзаданий для самостоятельной работы. тинга 9.2 (Программа печати сообщения Hello world!) (рис. 3.7).



```

lab09-2.asm  [----]  8 L: [ 1+16 17/ 21] *(245 / 293b) 0032 0x020 [*][X]
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
int 0x80

```

Рис. 3.7: Код программы

Код программы из пункта 7:

```

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
int 0x80

```

8. Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. После этого загрузим исполняемый файл в отладчик gdb и проверим работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 3.8).

```

taabakumov@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 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://bugs.gentoo.org/>.
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: /afs/.dk.sci.pfu.edu.ru/home/t/a/taabakumov/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3503) exited normally]
(gdb)

```

Рис. 3.8: Работа программы

9. Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. 3.9).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/a/taabakumov/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 3.9: Подробный анализ программы

10. Посмотрите дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 3.10).



```

(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) 

```

Рис. 3.10: Дисассимилированный код

11. Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. Отличие заключается в командах, в дисассимилированном отображении в командах используют `%` и `$`, а в Intel отображение эти символы не используются. На такое отображение удобнее смотреть (рис. 3.11).

```

(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. 3.11: Intel'овский синтаксис

12. Для удобства включим режим псевдографики (рис. 3.12).

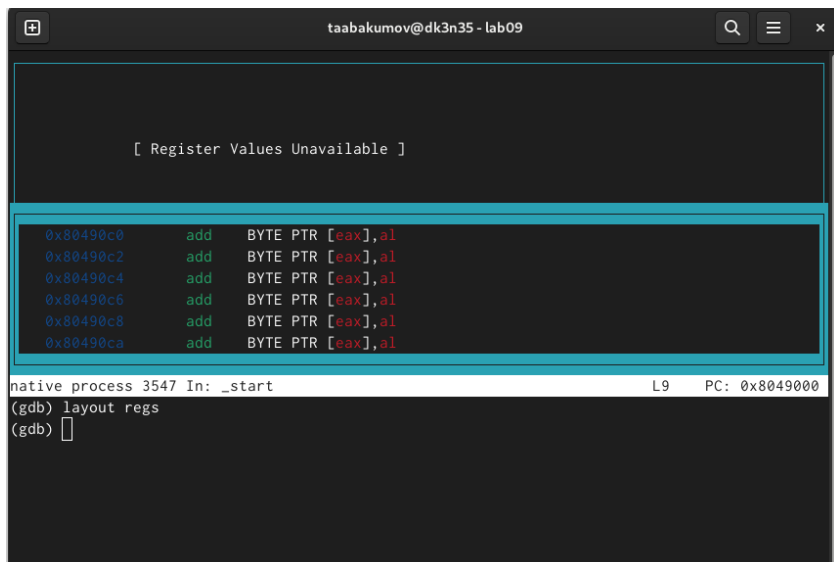
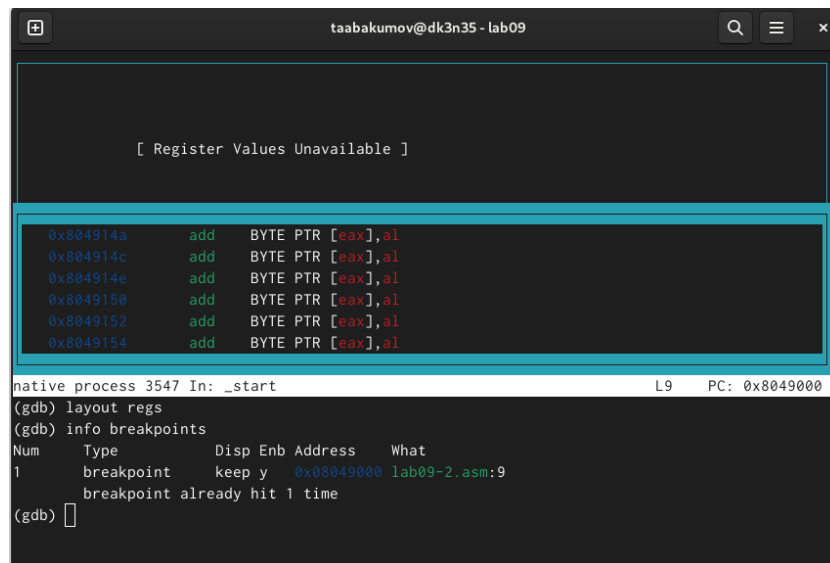


Рис. 3.12: Режим псевдографики

13. На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверим это с помощью команды `info breakpoints` (рис. 3.13).

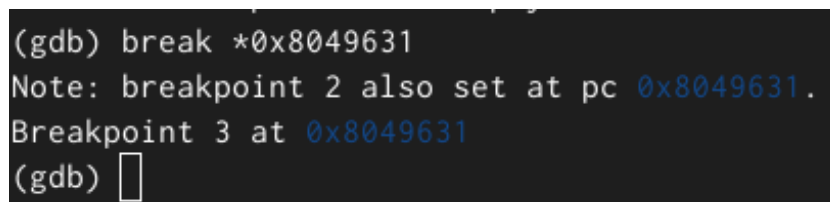


The screenshot shows a GDB window titled "taabakumov@dk3n35 - lab09". The main pane displays assembly code with instructions like `add BYTE PTR [eax],al` at various addresses. A cyan box highlights a portion of this code. The bottom pane shows the command `(gdb) info breakpoints` and its output, which lists a single breakpoint at address `0x08049000` for the label `lab09-2.asm:9`. It also shows the command `(gdb) layout regs` and the status of the native process.

```
native process 3547 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1      breakpoint      keep y   0x08049000  lab09-2.asm:9
breakpoint already hit 1 time
(gdb) 
```

Рис. 3.13: Проверка точки

14. Установим еще одну точку останова по адресу инструкции. Определим адрес предпоследней инструкции (`mov ebx,0x0`) и установим точку останова (рис. 3.14).



The screenshot shows a GDB window with the command `(gdb) break *0x8049631` being entered. The output indicates that breakpoint 2 is also set at the same address and that breakpoint 3 is being set at `0x8049631`.

```
(gdb) break *0x8049631
Note: breakpoint 2 also set at pc 0x8049631.
Breakpoint 3 at 0x8049631
(gdb) 
```

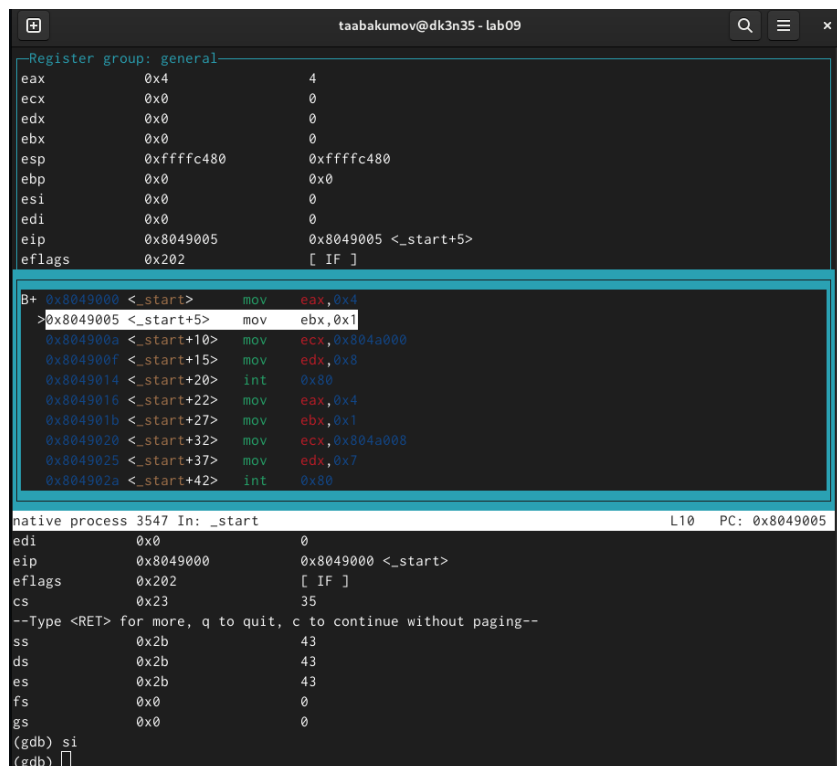
Рис. 3.14: Установка точки

15. Посмотрим информацию о всех установленных точках останова (рис. 3.15).

```
(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   0x08049631
3        breakpoint     keep y   0x08049631
(gdb) █
```

Рис. 3.15: Проверка информации

16. С помощью команды `si` посмотрим регистры и изменим их (рис. 3.16).



```
taabakumov@dk3n35 - lab09
Register group: general
eax      0x4          4
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xffffc480   0xffffc480
ebp      0x0          0x0
esi      0x0          0
edi      0x0          0
eip      0x8049005     0x8049005 <_start+5>
eflags   0x202        [ IF ]

B+ 0x8049000 <_start>   mov     eax, 4
>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 3547 In: _start          L10  PC: 0x8049005
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--
ss       0x2b         43
ds       0x2b         43
es       0x2b         43
fs       0x0          0
gs       0x0          0
(gdb) si
(gdb) █
```

Рис. 3.16: Изменение регистров

17. С помощью команды посмотрим значение переменной `msg1` (рис. 3.17).

```
(gdb) x/lb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) █
```

Рис. 3.17: Значение переменной `msg1`

18. Следом посмотрим значение второй переменной msg2 (рис. 3.18).

```
(gdb) x/lsb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) □
```

Рис. 3.18: Значение переменной msg2

19. С помощью команды set изменим значение переменной msg1 (рис. 3.19).

```
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/lsb &msg1
0x804a000 <msg1>:      "hh1lo, "
(gdb) □
```

Рис. 3.19: Изменение значение переменной msg1

20. Также изменим переменную msg2 (рис. 3.20).

```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/lsb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb) □
```

Рис. 3.20: Изменение значение переменной msg2

21. Выведем значения регистров еsx и еax (рис. 3.21).

```
$2 = void
(gdb) p/s $eax
$3 = 4
(gdb) p/t $eax
$4 = 100
(gdb) p/c $ecx
$5 = 0 '\000'
(gdb) p/x $ecx
$6 = 0x0
```

Рис. 3.21: Значения регистров

22. Изменим значение регистра `ebx`. Команда выводит два разных значения так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум, поэтому и значения разные (рис. 3.22).

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$7 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$8 = 2

```

Рис. 3.22: Изменение значения регистра

23. После всего завершим работу с файлов (рис. 3.23).

```

(gdb) c
Continuing.
hhllo, Lor d!
[Inferior 1 (process 3547) exited normally]

```

Рис. 3.23: Завершение работы с файлом

24. Далее скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm. После чего создадим исполняемый файл (рис. 3.24).

```

taabakumov@dk3n35 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
taabakumov@dk3n35 ~/work/arch-pc/lab09 $

```

Рис. 3.24: Копирование и создание исполняемого файла

25. Загрузим исполняемый файл в отладчик, указав аргументы. Установим точку останова перед первой инструкцией в программе (рис. 3.25).

```

taabakumov@dk3n35 ~/work/arch-pc/lab09 $ gdb --args lab09-3 x y z
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 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://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.

```

Рис. 3.25: Загрузка файла в отладчик

26. Далее запустим программу (рис. 3.26).

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/a/taabakumov/work/arch-pc/lab09/lab09-3 x y z

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) 

```

Рис. 3.26: Запуск программы

27. Проверим адрес вершины стека и убедимся, что там хранится 4 элемента (рис. 3.27).

```

(gdb) x/x $esp
0xffffc4b0:      0x00000004
(gdb) 

```

Рис. 3.27: Проверка адреса

28. Посмотрим все позиции стека. По первому адресу хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации (рис. 3.28).



```

(gdb) x/s *(void**)(esp + 4)
0xffffd358: "~/lab09-3"
A syntax error in expression, near `0xffffd358: "~/lab09-3"'.
(gdb) x/s *(void**)(esp + 4)
0xfffffc6f8: "/afs/.dk.sci.pfu.edu.ru/home/t/a/taabakumov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc73f: "x"
(gdb) x/s *(void**)(esp + 12)
0xfffffc741: "y"
(gdb) x/s *(void**)(esp + 16)
0xfffffc743: "z"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb) 

```

Рис. 3.28: Просмотр всех позиций стека

### 3.3 Задания для самостоятельной работы

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

29. Для начала создадим файл lab09-4.asm (рис. 3.29).

```

taabakumov@dk3n35 ~/work/arch-pc/lab09 $ touch lab09-4.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ 

```

Рис. 3.29: Создание файла

30. Преобразуем программу из лабораторной работы №8 и реализуем вычисления как подпрограмму (рис. 3.30).

```
lab09-4.asm [----] 3 L: [ 1+ 0 1/ 42] *(3 / 377b) 0099 0x063 [*][X]
%include 'in_out.asm'

SECTION .data
prim DB 'f(x)=10x-4',0
otv DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

mov eax,prim
call sprintf
next:
cmp ecx,0
jz _end

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюМС10Выход
```

Рис. 3.30: Код программы

31. Создадим исполняемый файл и проверим его работу (рис. 3.31).

```
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ./lab09-4
f(x)=10x-4
Результат: 0
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ./lab09-4 2 3
f(x)=10x-4
Результат: 42
taabakumov@dk3n35 ~/work/arch-pc/lab09 $
```

Рис. 3.31: Работа программы

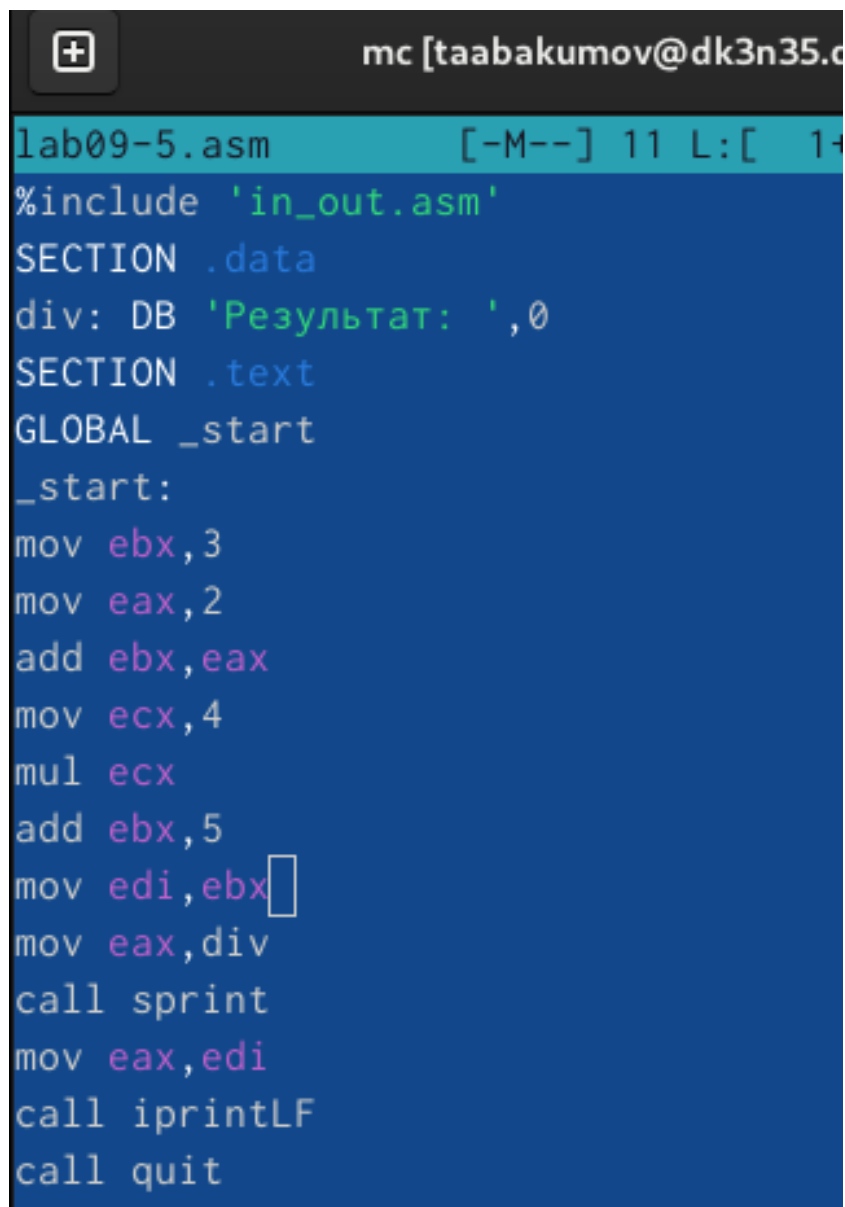
Задание 2. В листинге 9.3 приведена программа вычисления выражения  $(3 + 2) * 4 + 5$ . При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

32. Для начала создадим файл lab09-5.asm (рис. 3.32).

```
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ touch lab09-5.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $
```

Рис. 3.32: Создание файла

33. Перепишем программу из Листинга 9.3 в созданный файл (рис. 3.33).



```
lab09-5.asm [-M--] 11 L:[ 1+
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 3.33: Код программы

Код программы из пункта 33:

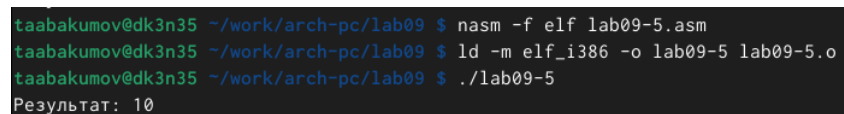
```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат:',0
SECTION .text
```

```

GLOBAL _start
_start:
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

34. Создадим исполняемый файл и проверим его работу. Ошибка оказалась арифметическая, так как вместо 25, программа выводит 10 (рис. 3.34).



```

taabakumov@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 10

```

Рис. 3.34: Работа программы

35. Из-за появления ошибки, запустим программу в отладчике (рис. 3.35).

```

taabakumov@dk3n35 ~/work/arch-pc/lab09 $ gdb lab09-5
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 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://bugs.gentoo.org/>.
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 lab09-5...
(No debugging symbols found in lab09-5)
(gdb) b _start
Breakpoint 1 at 0x080490e8
(gdb) r
quit
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/a/taabakumov/work/arch-pc/lab09/lab09-5

Breakpoint 1, 0x080490e8 in _start ()

(gdb) set disassemble-flavor intel

No symbol table is loaded. Use the "file" command.
(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

```

Рис. 3.35: Запуск программы в отладчике

36. Изменим регистры и запустим программу, программа вывела ответ 25, то есть все работает правильно (рис. 3.36).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1d0 0xffffd1d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

0x80490db <quit>      mov     ebx,0x0
0x80490e0 <quit+5>    mov     eax,0x1
0x80490e5 <quit+10>   int     0x80
0x80490e7 <quit+12>   ret
B+> 0x80490e8 <_start> mov     ebx,0x3
0x80490ed <_start+5>  mov     eax,0x2
0x80490f2 <_start+10> add     ebx,eax
0x80490f4 <_start+12> mov     ecx,0x4
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>
0x804910a <_start+34> mov     eax,edi
```

Рис. 3.36: Изменение регистров

(рис. 3.37).

```
taabakumov@dk3n35 - lab09
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
taabakumov@dk3n35 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 25
taabakumov@dk3n35 ~/work/arch-pc/lab09 $
```

Рис. 3.37: Работа программы

## 4 Выводы

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