

Лабораторная работа номер 9

Отчёт

Виноградова Мария Андреевна

Содержание

1	Цель работы	6
2	Задание	7
3	Выполнение лабораторной работы	8
3.1	Реализация подпрограмм в NASM	8
3.2	Отладка программ с помощью GDB	10
4	Выводы	25

Список иллюстраций

3.1	Создаем каталог с помощью команды <code>mkdir</code> и файл с помощью команды <code>touch</code>	8
3.2	Заполняем файл	9
3.3	Запускаем файл и проверяем его работу	9
3.4	Изменяем файл, добавляя еще одну подпрограмму	10
3.5	Запускаем файл и смотрим его работу	10
3.6	Создаем файл	10
3.7	Заполняем файл	11
3.8	Загружаем исходный файл в отладчик	12
3.9	Запускаем программу командой <code>run</code>	12
3.10	Запускаем программу с брейкпоинтом	12
3.11	Смотрим дисассимилированный код программы	13
3.12	Переключаемся на синтаксис Intel	13
3.13	Включаем отображение регистров, их значений и результат дисассимилирования программы	15
3.14	Используем команду <code>info breakpoints</code> и создаем новую точку останова	15
3.15	Смотрим информацию	16
3.16	Отслеживаем регистры	16
3.17	Смотрим значение переменной	16
3.18	Смотрим значение переменной	16
3.19	Меняем символ	16
3.20	Меняем символ	17
3.21	Смотрим значение регистра	17
3.22	Изменяем регистр командой <code>set</code>	17
3.23	Прописываем команды <code>s</code> и <code>quit</code>	17
3.24	Копируем файл	18
3.25	Создаем и запускаем в отладчике файл	18
3.26	Устанавливаем точку останова	18
3.27	Изучаем полученные данные	19
3.28	Копируем файл	19
3.29	Изменяем файл	20
3.30	Проверяем работу программы	20
3.31	Создаем файл	20
3.32	Изменяем файл	21
3.33	Создаем и смотрим на работу программы(работает неправильно)	21
3.34	Ищем ошибку регистров в отладчике	22
3.35	Меняем файл	23

3.36 Создаем и запускаем файл(работает корректно)	24
---	----

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

Написать программы с использованием циклов и обработкой аргументов командной строки.

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. fig. 3.1).

```
mavinogradova@10:~/work/arch-pc/lab08$ cd  
mavinogradova@10:~$ mkdir ~/work/arch-pc/lab09  
mavinogradova@10:~$ cd ~/work/arch-pc/lab09  
mavinogradova@10:~/work/arch-pc/lab09$ touch lab9-1.asm  
mavinogradova@10:~/work/arch-pc/lab09$ mc
```

Рис. 3.1: Создаем каталог с помощью команды `mkdir` и файл с помощью команды `touch`

Открываем файл в Midnight Commander и заполняем его так как показано в листинге 9.1 (рис. fig. 3.2).


```

#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
    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

```

Рис. 3.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. fig. 3.3).

```

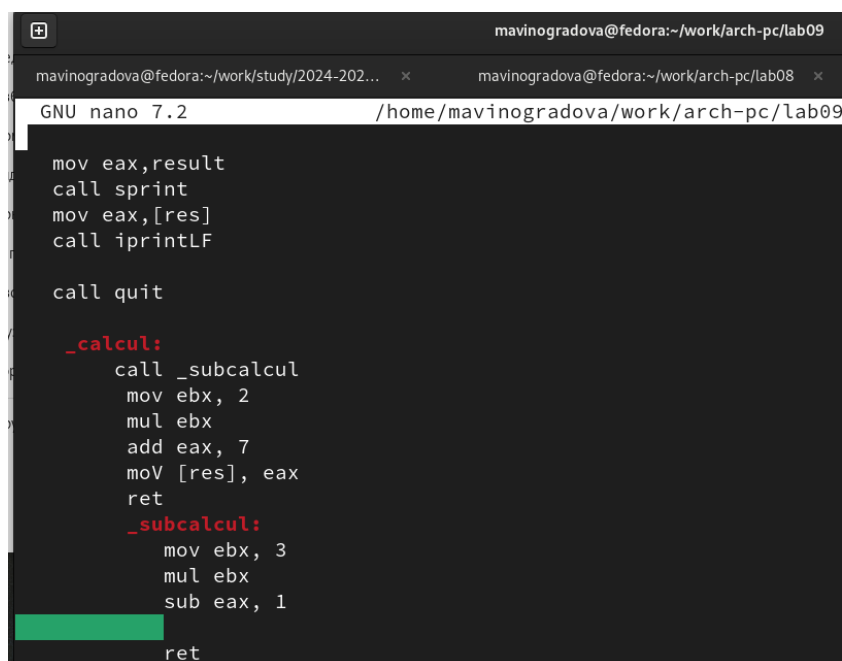
mavinogradova@10:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
mavinogradova@10:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab8-1 lab8-1.o
ld: невозможно найти lab8-1.o: Нет такого файла или каталога
mavinogradova@10:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
mavinogradova@10:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 7
2x+7=21
mavinogradova@10:~/work/arch-pc/lab09$

```

Рис. 3.3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпро-

грамму в подпрограмму (как сказано в условии) (рис. fig. 3.4).



```
mavinogradova@fedora:~/work/arch-pc/lab09
GNU nano 7.2 /home/mavinogradova/work/arch-pc/lab09

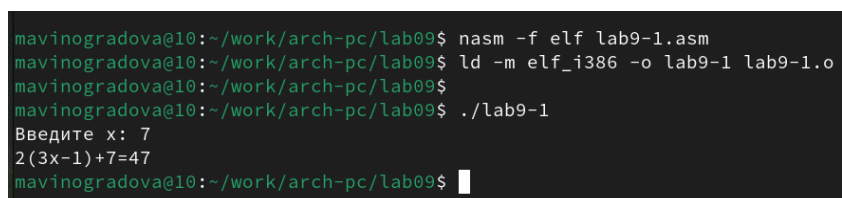
mov eax,result
call sprint
mov eax,[res]
call iprintLF

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: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его, чтобы посмотреть на работу изменённой программы (рис. fig. 3.5).

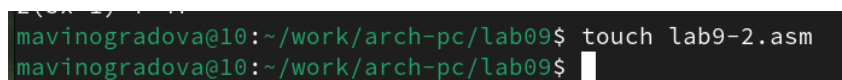


```
mavinogradova@10:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
mavinogradova@10:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
mavinogradova@10:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 7
2(3x-1)+7=47
mavinogradova@10:~/work/arch-pc/lab09$
```

Рис. 3.5: Запускаем файл и смотрим его работу

3.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге с помощью команды touch (рис. fig. 3.6).



```
mavinogradova@10:~/work/arch-pc/lab09$ touch lab9-2.asm
mavinogradova@10:~/work/arch-pc/lab09$
```

Рис. 3.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его так как показано в листинге 9.2 (рис. fig. 3.7).

```
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
```

Рис. 3.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. fig. 3.8).

```

mavinogradova@10:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
nasm: fatal: unable to open input file 'lab09-2.asm' No such file or directory
mavinogradova@10:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
mavinogradova@10:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
ld: не распознан режим эмуляции: elf_i386-o
Поддерживаемые эмуляции: elf_x86_64 elf32_x86_64 elf_i386 elf_iamcu i386pep i386pe elf64bpf
mavinogradova@10:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
mavinogradova@10:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 14.2-1.fc40
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-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. 3.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. fig. 3.9).

```

(gdb) run
Starting program: /home/mavinogradova/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 28321) exited normally]
(gdb)

```

Рис. 3.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку `_start` и запускаем программу (рис. fig. 3.10).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 10.
(gdb) run
Starting program: /home/mavinogradova/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:10
10      mov eax, 4
(gdb)

```

Рис. 3.10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. fig. 3.11).

```

(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.
(gdb)

```

Рис. 3.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. fig. 3.12).

```

(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. 3.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATТ и Intel:

1.Порядок операндов: В АТТ синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2.Разделители: В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3.Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.

4.Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом “*.Intel*”.

5.Обозначение адресов: В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6.Обозначение регистров: В АТТ синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

Включаем режим псевдографики (рис. fig. 3.13).

```
mavinogradova@fedora:~/work/study/2024-202... x mavinogradova@fedora:~/work/arch-pc/lab08 x mavinogradova@fedora:~/work/arch-pc/lab09 x
[ Register Values Unavailable ]

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

native process 28354 In: _start L10 PC: 0x8049000
(gdb) layout regs
(gdb) █
```

Рис. 3.13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. fig. 3.14).

```
mavinogradova@fedora:~/work/study/2024-202... x mavinogradova@fedora:~/work/arch-pc/lab08 x mavinogradova@fedora:~/work/arch-pc/lab09 x
[ Register Values Unavailable ]

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

native process 28354 In: _start L10 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab9-2.asm:10
breakpoint already hit 1 time
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab9-2.asm:10
breakpoint already hit 1 time
(gdb) █
```

Рис. 3.14: Используем команду info breakpoints и создаем новую точку останова

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

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000  lab9-2.asm:10
breakpoint already hit 1 time
(gdb)
```

Рис. 3.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. fig. 3.16).

```
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 3.16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip.

Смотрим значение переменной msg1 по имени (рис. fig. 3.17).

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

Рис. 3.17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. fig. 3.18).

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

Рис. 3.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. fig. 3.19).

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

Рис. 3.19: Меняем символ

Изменим первый символ переменной msg2 (рис. fig. 3.20).

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

Рис. 3.20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. fig. 3.21).

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

Рис. 3.21: Смотрим значение регистра

Изменяем регистр ebx (рис. fig. 3.22).

Изменяем регистр командой set

Рис. 3.22: Изменяем регистр командой set

Выводятся разные значения, так как команда без кавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. fig. 3.23).

```
(gdb) c
Continuing.
Lorld!
[Inferior 1 (process 28354) exited normally]
(gdb)
```

Рис. 3.23: Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. fig. 3.24).

```
mavinogradova@i0: ~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
mavinogradova@i0: ~/work/arch-pc/lab09$
```

Рис. 3.24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. fig. 3.25).

```
mavinogradova@i0: ~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
mavinogradova@i0: ~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
mavinogradova@i0: ~/work/arch-pc/lab09$ gdb--args lab09-3 2 3 '5'
bash: gdb--args: команда не найдена...
mavinogradova@i0: ~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 '5'
GNU gdb (Fedora Linux) 14.2-1.fc40
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-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 lab09-3...
(gdb)
```

Рис. 3.25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. fig. 3.26).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/mavinogradova/work/arch-pc/lab09/lab09-3 2 3 5

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 lab09-3.asm:7
7       pop ecx
(gdb) x/x $esp
0xffffd090: 0x00000004
(gdb)
```

Рис. 3.26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. fig. 3.27).

```

(gdb) x/x $esp
0xffffd090: 0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd24d: "/home/mavinogradova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd27c: "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd27e: "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd280: "5"
(gdb) x/s *(void**)(esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.27: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

###Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. fig. 3.28).

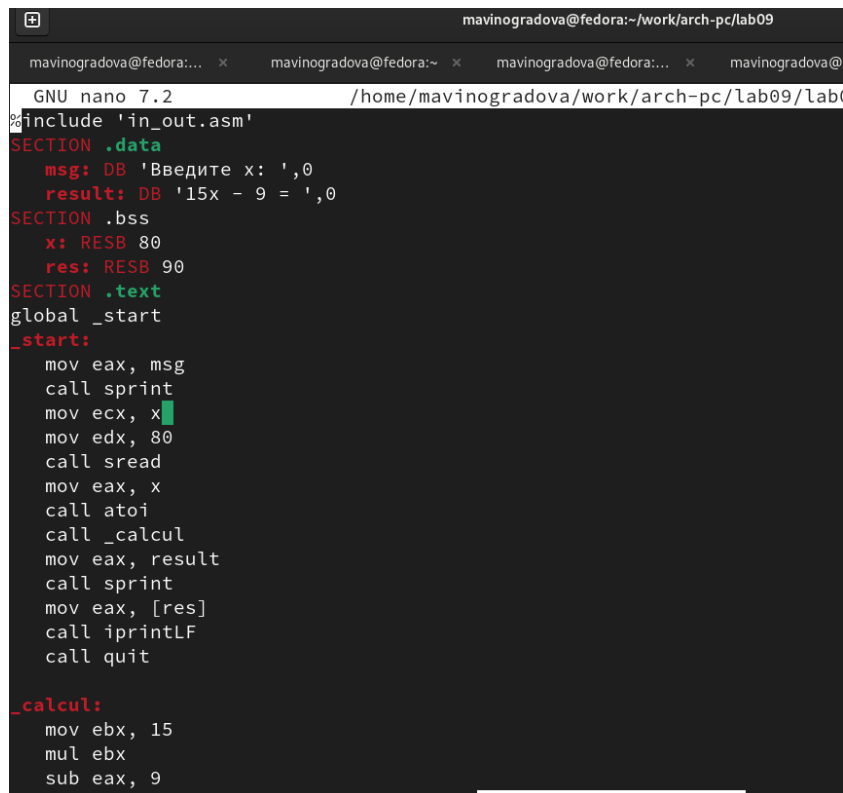
```

mavinogradova@18: ~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.
asm
mavinogradova@18: ~/work/arch-pc/lab09$

```

Рис. 3.28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. fig. 3.29).

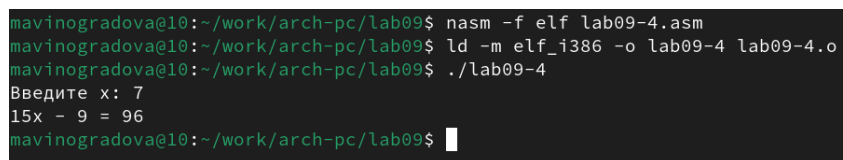


```
GNU nano 7.2 /home/mavinogradova/work/arch-pc/lab09/lab09-4.asm
%include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '15x - 9 = ',0
SECTION .bss
    x: RESB 80
    res: RESB 90
SECTION .text
global _start
_start:
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    call _calcul
    mov eax, result
    call sprint
    mov eax, [res]
    call iprintLF
    call quit

_calcul:
    mov ebx, 15
    mul ebx
    sub eax, 9
```

Рис. 3.29: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. fig. 3.30).

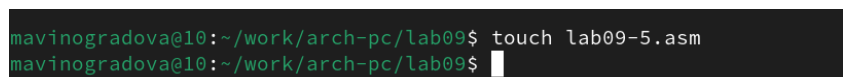


```
mavinogradova@10:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
mavinogradova@10:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
mavinogradova@10:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 7
15x - 9 = 96
mavinogradova@10:~/work/arch-pc/lab09$
```

Рис. 3.30: Проверяем работу программы

###Задание 2

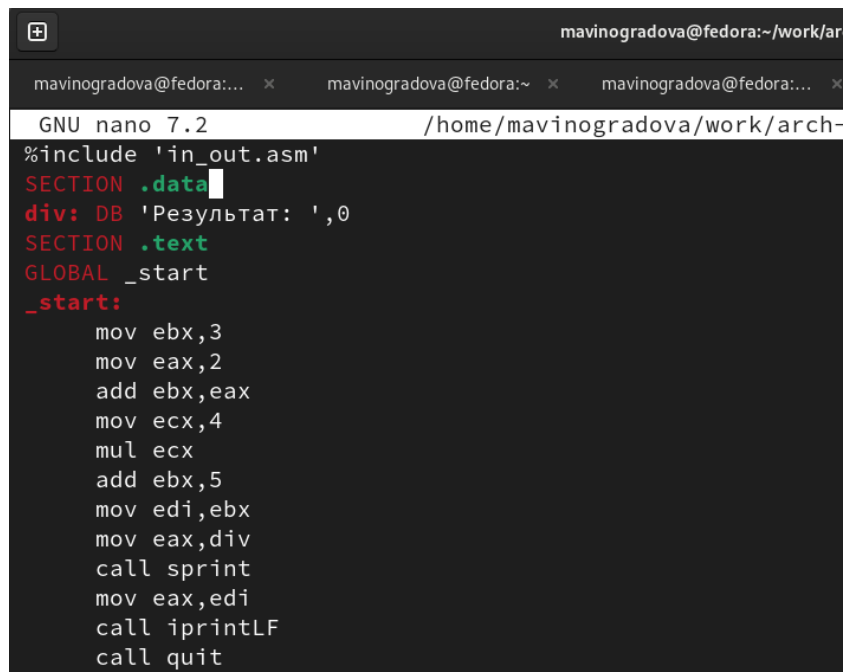
Создаем новый файл в директории с помощью команды touch (рис. fig. 3.31).



```
mavinogradova@10:~/work/arch-pc/lab09$ touch lab09-5.asm
mavinogradova@10:~/work/arch-pc/lab09$
```

Рис. 3.31: Создаем файл

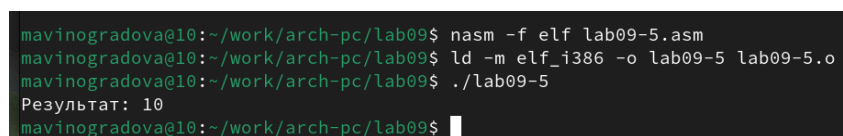
Открываем файл в Midnight Commander и заполняем его так как показано в листинге 9.3 (рис. fig. 3.32).



```
GNU nano 7.2 /home/mavinogradova/work/arch-
%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. 3.32: Изменяем файл

Создаем исполняемый файл и запускаем его (чтобы посмотреть на работу команды с ошибкой) (рис. fig. 3.33).



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

Рис. 3.33: Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si (рис. fig. 3.34).

```
mavinogradova@fedora: ~/work/arch-pc/lab09

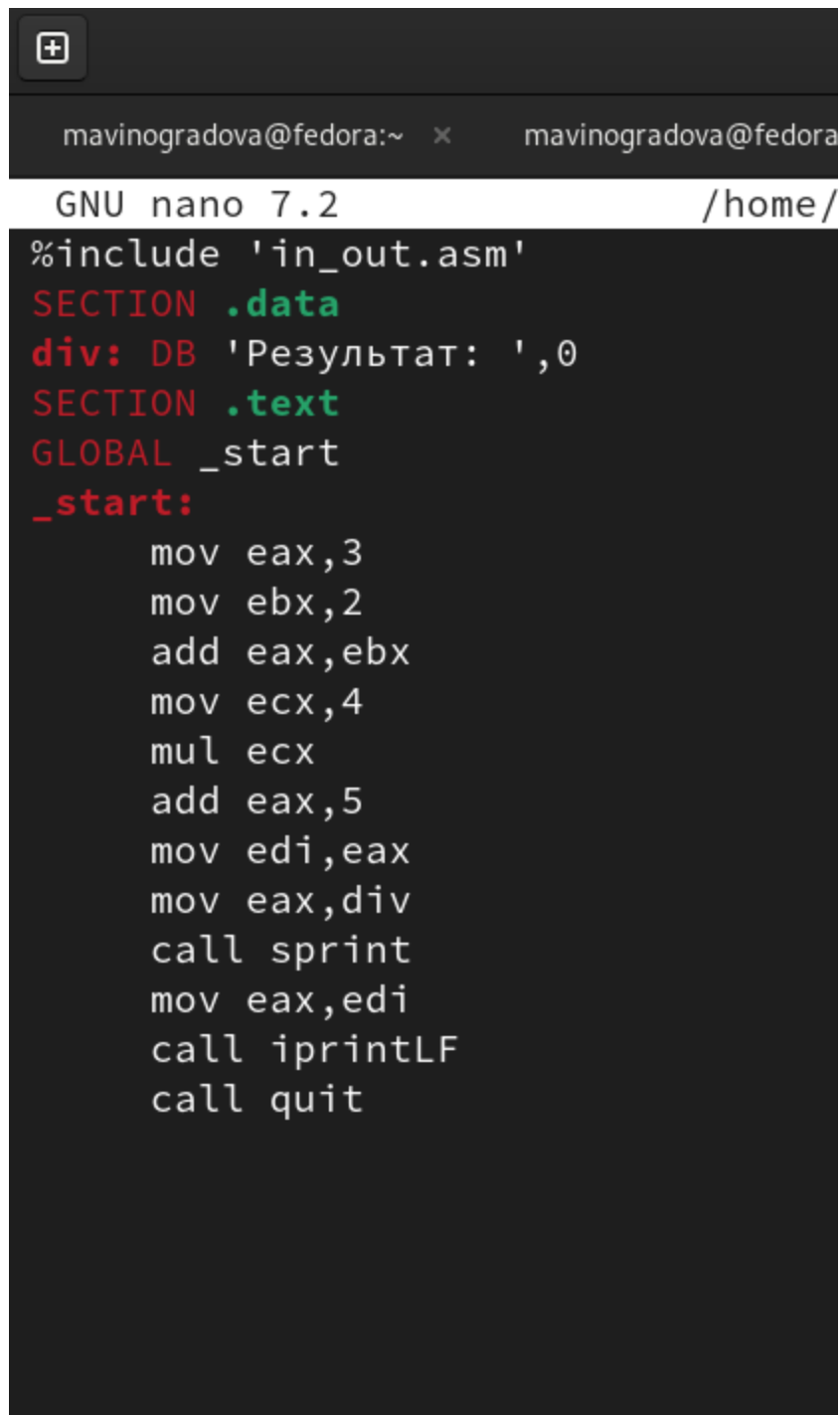
Register group: general
eax 0x2 2
ecx 0x4 4
edx 0x0 0
ebx 0x5 5
esp 0xffffd0b0 0xffffd0b0
ebp 0x0 0
esi 0x0 0
edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17>
eflags 0x206 [ PF IF ]
cs 0x23 35
ss 0x2b 43

B+ 0x80490eb <_start> mov ebx,0x3
0x80490ed <_start+5> mov ecx,0x2
0x80490f2 <_start+10> add ebx,ecx
0x80490f8 <_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
0x804910c <_start+36> call 0x8049000 <printf>
0x8049111 <_start+41> call 0x8049000 <quit>
0x8049116 <_start+46> add BYTE PTR [eax],al

native process 8879 In: _start L11 PC: 0x80490f9
esp 0xffffd0b0 0xffffd0b0
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490f4 0x80490f4 <_start+12>
eflags 0x206 [ PF IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
--Type <RET> for more, q to quit, c to continue without paging--fs 0x0
```

Рис. 3.34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. fig. 3.35).



```
GNU nano 7.2 /home/
%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
```

Рис. 3.35: Меняем файл

Создаем исполняемый файл и запускаем его (чтобы посмотреть на работу программы без ошибки) (рис. fig. 3.36).

```
mavinogradova@10:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
mavinogradova@10:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
mavinogradova@10:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
mavinogradova@10:~/work/arch-pc/lab09$
```

Рис. 3.36: Создаем и запускаем файл(работает корректно)

4 Выводы

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