

Отчёт по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Юсупова Ксения Равиловна

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
2.1	Реализация подпрограмм в NASM	7
2.2	Отладка программ с помощью GDB	9
2.3	Задание для самостоятельной работы	17
2.3.1	Задание 1	17
2.3.2	Задание 2	18
3	Выводы	22
	Список литературы	23

Список иллюстраций

2.1	Создали каталог с помощью команды <code>mkdir</code> и файл <code>lab9-1.asm</code> с помощью команды <code>touch</code>	7
2.2	Заполнили файл по листингу	8
2.3	Создали исполняемый файл и запустили его	8
2.4	Изменили файл	9
2.5	Создали исполняемый файл и запустили его	9
2.6	Создали файл <code>lab09-2.asm</code>	10
2.7	Заполнили файл	10
2.8	Загрузили исходный файл в отладчик	10
2.9	Запустили программу командой <code>run</code>	11
2.10	Запустили программу с брейкпоинтом	11
2.11	Просмотрели дисассимилированный код программы	11
2.12	Переключаемся на синтаксис Intel	12
2.13	Включили режим псевдографики	12
2.14	Использовали команду <code>info breakpoints</code> и создали новую точку останова	13
2.15	Посмотрели информацию	13
2.16	Отследили регистры	14
2.17	Посмотрели значение переменной	14
2.18	Посмотрели значение переменной	14
2.19	Изменили символ	15
2.20	Изменили символ	15
2.21	Изменили символ	15
2.22	Изменили регистр	15
2.23	Прописали команды <code>s</code> и <code>quit</code>	16
2.24	Скопировали файл	16
2.25	Создали и запустили в отладчике файл	16
2.26	Устанавливаем точку	16
2.27	Изучили полученные данные	17
2.28	Создали исполняемый файл и скопировали в него <code>lab8-4.asm</code> . . .	17
2.29	Изменили файл	18
2.30	Создали исполняемый файл и запустили его	18
2.31	Заполнили файл	19
2.32	Создали исполняемый файл и запустили его, заметили неправильный вывод	19
2.33	Загрузили исходный файл в отладчик	20
2.34	Определяем ошибку	20

2.35 Исправили ошибки в программе	21
2.36 Создали и запустили файл, убедились в верности вывода	21

Список таблиц

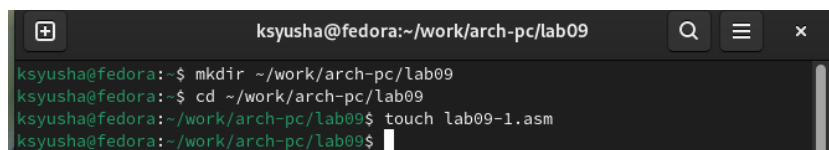
1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

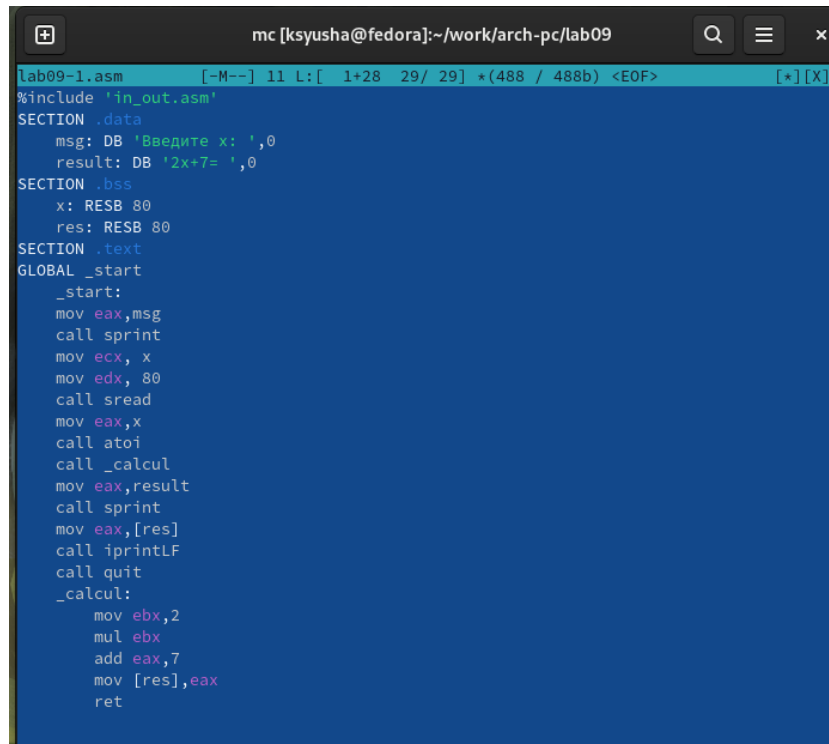
Создаем каталог для программ лабораторной работы № 9, переходим в него и создаём файл lab9-1.asm(рис. 2.1).



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

Рис. 2.1: Создали каталог с помощью команды `mkdir` и файл `lab9-1.asm` с помощью команды `touch`

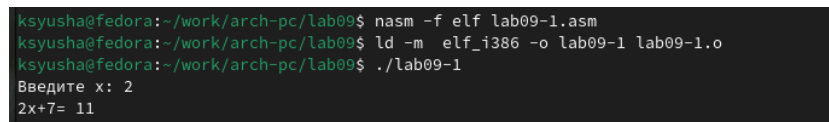
Вводим в файл `lab7-1.asm` текст программы из листинга 9.1. (рис. 2.2).



```
lab09-1.asm [-M--] 11 L: [ 1+28 29/ 29] *(488 / 488b) <EOF> [*][X]
#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
    call quit
    _calcul:
        mov ebx,2
        mul ebx
        add eax,7
        mov [res],eax
        ret
```

Рис. 2.2: Заполнили файл по листингу

Создаем исполняемый файл и запускаем его(рис. 2.3).



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

Рис. 2.3: Создали исполняемый файл и запустили его

Далее изменяем текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`.(рис. 2.4).


```
mc [ksyusha@fedora]:~/work/arch-pc/lab09
lab09-1.asm [-M--] 23 L:[ 1+24 25/ 35] *(431 / 617b) 0010 0x00A
#include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '2(3x-1)+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
    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. 2.4: Изменили файл

Создаем исполняемый файл и запускаем его(рис. 2.5).

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

Рис. 2.5: Создали исполняемый файл и запустили его

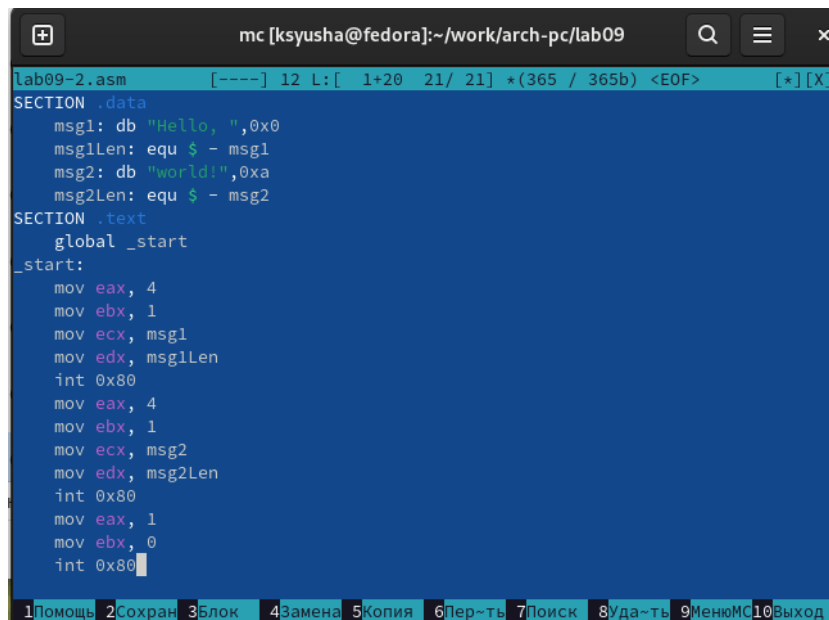
2.2 Отладка программ с помощью GDB

Создаем исполняемый новый файл lab09-2.asm в каталоге ~/work/arch-pc/lab09 с помощью команды touch(рис. 2.6).

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

Рис. 2.6: Создали файл lab09-2.asm

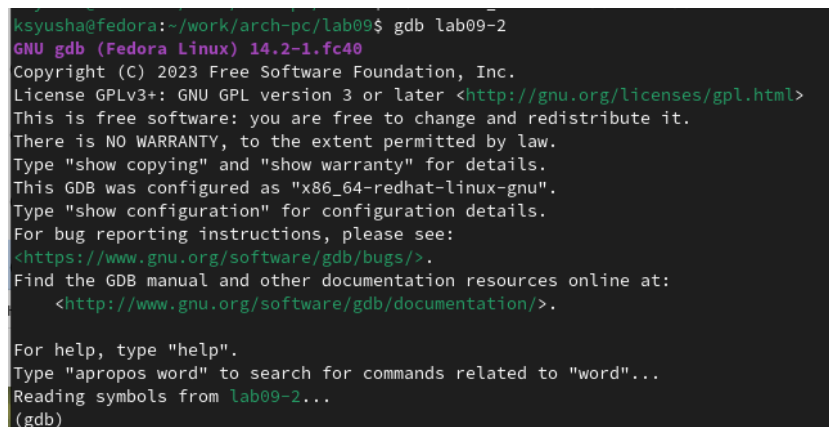
Открываем файл в Midnight Commander и заполняем его в соответствии с листингом(рис. 2.7).



```
lab09-2.asm  [----] 12 L: [ 1+20 21/ 21] *(365 / 365b) <EOF>  [*][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. 2.7: Заполнили файл

Получаем исходный файл с использованием отладчика gdb(рис. 2.8).



```
ksyusha@fedora:~/work/arch-pc/lab09$ gdb lab09-2
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 lab09-2...
(gdb)
```

Рис. 2.8: Загрузили исходный файл в отладчик

Проверяем работу программы, запустив ее в оболочке GDB с помощью команды

run.(рис. 2.9)

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

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.
Hello, world!
[Inferior 1 (process 3897) exited normally]
(gdb)
```

Рис. 2.9: Запустили программу командой run

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её(рис. 2.10).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/ksyusha/work/arch-pc/lab09/lab09-2

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

Рис. 2.10: Запустили программу с брейкпоином

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 2.11).

```
(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. 2.11: Просмотрели дисассимилированный код программы

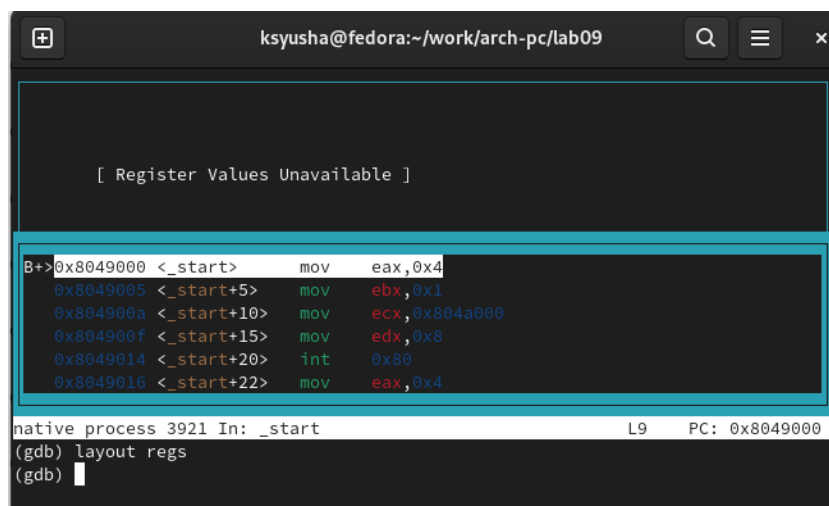
Переключаемся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`(рис. 2.12).

```
(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. 2.12: Переключаемся на синтаксис Intel

Главное различие между АТТ и Intel синтаксисом ассемблера — порядок операндов: АТТ использует источник, назначение, а Intel — назначение, источник. АТТ использует префикс `%` для регистров (например, `%eax`), а Intel — нет (например, `eax`). Формат мемориальных адресов также немного отличается. Многие ассемблеры поддерживают оба синтаксиса.

Включим режим псевдографики для более удобного анализа программы(рис. 2.13).



```
ksyusha@fedora:~/work/arch-pc/lab09
[ Register Values Unavailable ]
B> 0x08049000 <_start>    mov     eax,0x4
      0x08049005 <_start+5>    mov     ebx,0x1
      0x0804900a <_start+10>   mov     ecx,0x804a000
      0x0804900f <_start+15>   mov     edx,0x8
      0x08049014 <_start+20>   int     0x80
      0x08049016 <_start+22>   mov     eax,0x4
native process 3921 In: _start          L9      PC: 0x08049000
(gdb) layout regs
(gdb)
```

Рис. 2.13: Включили режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints`. Установим еще одну точку останова по адресу инструкции.(рис. 2.14).

```

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 3921 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) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)

```

Рис. 2.14: Использовали команду `info breakpoints` и создали новую точку останова

Посмотрим информацию о всех установленных точках останова(рис. 2.15).

```

(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   0x08049031 lab09-2.asm:20
(gdb)

```

Рис. 2.15: Посмотрели информацию

Выполняем 5 инструкций командой `si`(рис. 2.16).

```

ksyusha@fedora:~/work/arch-pc/lab09
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0d0 0xffffd0d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]

B+ 0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
>0x8049016 <_start+22>  mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1
0x8049020 <_start+32>   mov     ecx,0x804a008
0x8049025 <_start+37>   mov     edx,0x7
0x804902a <_start+42>   int     0x80

native process 3921 In: _start L14 PC: 0x8049016
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num   Type      Disp Enb Address      What
1     breakpoint keep  y  0x08049000 lab09-2.asm:9
      breakpoint already hit 1 time
2     breakpoint keep  y  0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.16: Отследили регистры

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

Смотрим значение переменной msg1 по имени(рис. 2.17).

```

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

```

Рис. 2.17: Посмотрели значение переменной

Смотрим значение переменной msg2 по адресу(рис. 2.18).

```

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

```

Рис. 2.18: Посмотрели значение переменной

Изменим первый символ переменной msg1.(рис. 2.19)

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

Рис. 2.19: Изменили символ

Изменим первый символ переменной msg2(рис. 2.20).

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

Рис. 2.20: Изменили символ

Смотрим значение регистра edx в разных форматах(рис. 2.21).

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

Рис. 2.21: Изменили символ

Изменяем регистр ebx(рис. 2.22).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
```

Рис. 2.22: Изменили регистр

Вывод команд p/s \$ebx имеет разные значения, так как команда без кавычек присваивает регистру вводимое значение.

Завершаем выполнение программы с помощью команды `continue` (сокращенно `c`) и выходим из GDB с помощью команды `quit` (сокращенно `q`). (рис. 2.23)

```
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) █
```

Рис. 2.23: Прописали команды `c` и `quit`

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`. (рис. 2.24).

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

Рис. 2.24: Скопировали файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. 2.25)

```
ksyusha@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ksyusha@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
ksyusha@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 2 3 5
```

Рис. 2.25: Создали и запустили в отладчике файл

Для начала установим точку останова перед первой инструкцией в программе и запустим ее. (рис. 2.26).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/ksyusha/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:5
5      pop ecx
(gdb) x/x $esp
0xffffd0b0: 0x00000004
(gdb) █
```

Рис. 2.26: Устанавливаем точку

Смотрим позиции стека по разным адресам. (рис. 2.27).

```
(gdb) x/s *(void**)(esp + 4)
0xffffd27a:  "/home/ksyusha/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2a3:  "2"
(gdb) x/s *(void**)(esp + 12)
0xffffd2a5:  "3"
(gdb) x/s *(void**)(esp + 16)
0xffffd2a7:  "5"
(gdb) x/s *(void**)(esp + 20)
0x0:  <error: Cannot access memory at address 0x0>
```

Рис. 2.27: Изучили полученную данные

Шаг изменения адреса равен 4, потому что адресные регистры имеют размерность 32 бита или же 4 байта.

2.3 Задание для самостоятельной работы

2.3.1 Задание 1

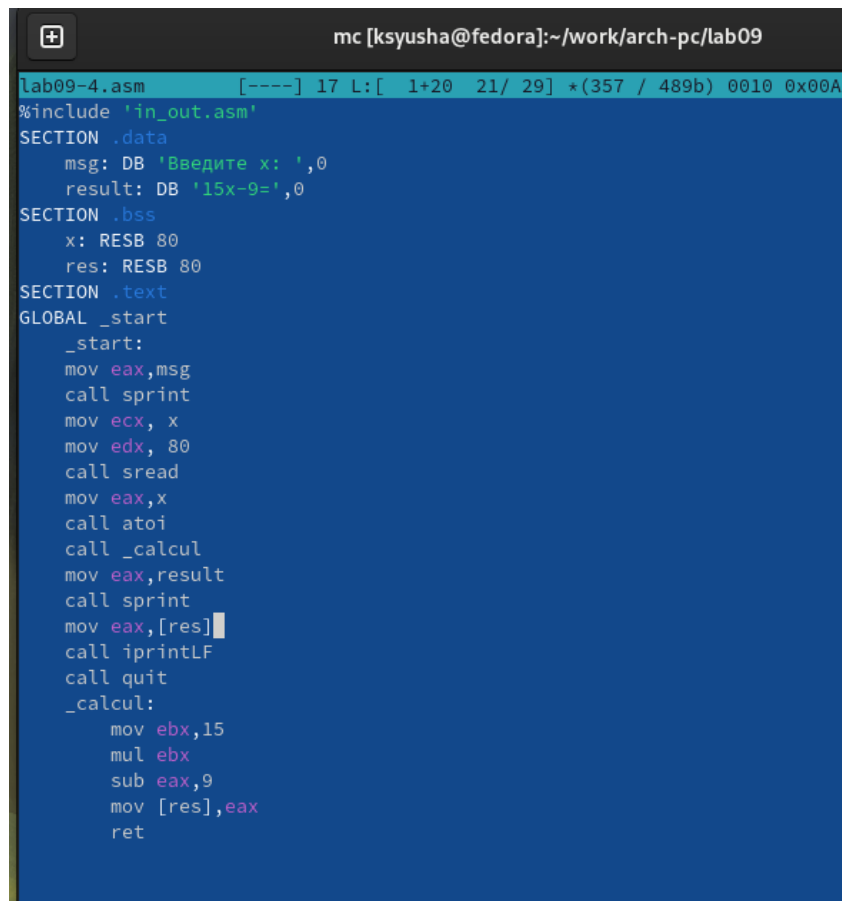
1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.

Создаем исполняемый новый файл lab09-4.asm в каталоге ~/work/arch-pc/lab09.c помощью команды touch и скопировали в него файл lab8-4.asm(рис. 2.28).

```
ksyusha@fedora:~/work/arch-pc/lab09$ touch lab09-4.asm
```

Рис. 2.28: Создали исполняемый файл и скопировали в него lab8-4.asm

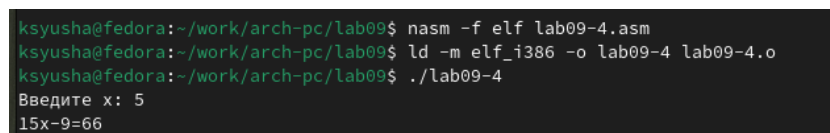
Открываем файл в Midnight Commander и меняем его, создавая подпрограмму(рис. 2.29).



```
mc [ksyusha@fedora]:~/work/arch-pc/lab09
lab09-4.asm [----] 17 L: [ 1+20 21/ 29] *(357 / 489b) 0010 0x00A
#include 'in_out.asm'
SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '15x-9=',0
SECTION .bss
    x: RESB 80
    res: RESB 80
SECTION .text
GLOBAL _start
_start:
    mov eax,msg
    call sprintf
    mov ecx, x
    mov edx, 80
    call sread
    mov eax,x
    call atoi
    call _calcul
    mov eax,result
    call sprintf
    mov eax,[res]
    call iprintLF
    call quit
_calcul:
    mov ebx,15
    mul ebx
    sub eax,9
    mov [res],eax
    ret
```

Рис. 2.29: Изменили файл

Создаем исполняемый файл и запускаем его((рис. 2.30).



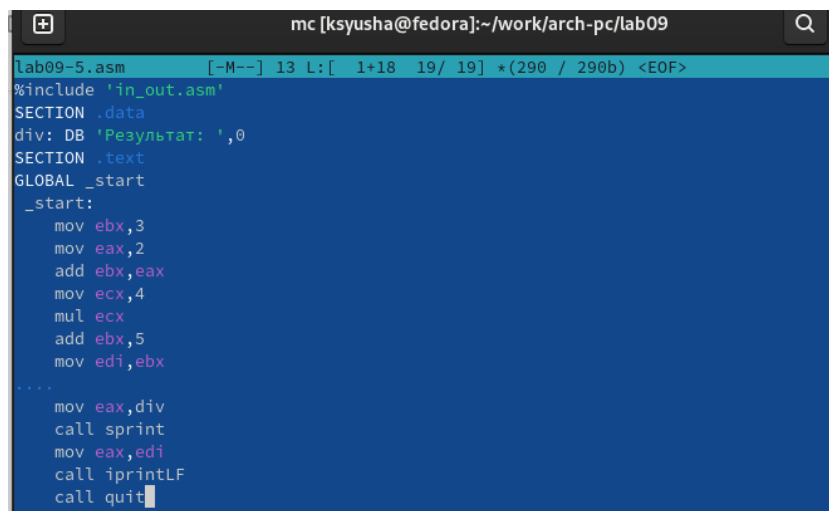
```
ksyusha@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
ksyusha@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
ksyusha@fedora:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 5
15x-9=66
```

Рис. 2.30: Создали исполняемый файл и запустили его

2.3.2 Задание 2

- 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.

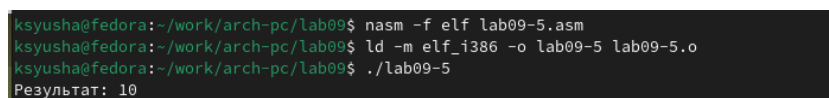
Создаём файл lab09-4.asm и заполняем его в соответствии с листингом 9.3 (рис. 2.31).



```
lab09-5.asm [-M--] 13 L:[ 1+18 19/ 19] *(290 / 290b) <EOF>
%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. 2.31: Заполнили файл

Создаем исполняемый файл и запускаем его, проверяя вывод ошибочного значения(рис. 2.32).



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

Рис. 2.32: Создали исполняемый файл и запустили его, заметили неправильный вывод

Загружаем исходный файл в отладчик(рис. 2.33).

```
ksyusha@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-5.lst lab09-5.asm
ksyusha@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
ksyusha@fedora:~/work/arch-pc/lab09$ gdb lab09-5
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 lab09-5...
(No debugging symbols found in lab09-5)
(gdb)
```

Рис. 2.33: Загрузили исходный файл в отладчик

Запускаем файл в отладчике GDB и смотрим на изменение регистров командой si(рис. 2.34)

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

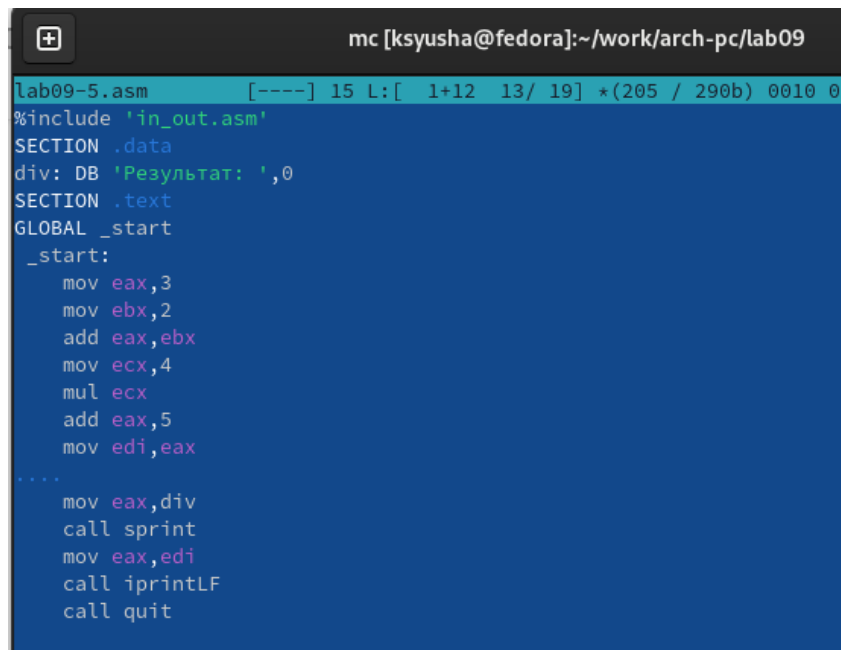
Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd0d0 0xffffd0d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>
eflags   0x206    [ PF IF ]

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
0x804910c <_start+36>  call    0x8049086 <iprintfLF>

native process 3712 In: _start L?? PC: 0x80490f9
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd0d0 0xffffd0d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>
eflags   0x206    [ PF IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.34: Определяем ошибку

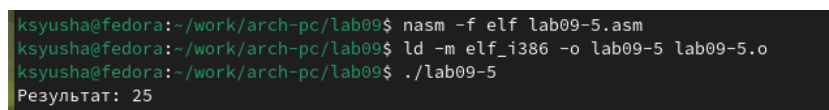
Изменяем программу так, чтобы вывод был верным 2.35).



```
mc [ksyusha@fedora]:~/work/arch-pc/lab09
lab09-5.asm [----] 15 L: [ 1+12 13/ 19] *(205 / 290b) 0010 0
%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. 2.35: Исправили ошибки в программе

Создаем исполняемый файл и проверяем корректность работы(рис. 2.36).



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

Рис. 2.36: Создали и запустили файл, убедились в верности вывода

3 Выводы

В ходе лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм, и ознакомились с методами отладки при помощи GDB и его основными возможностями.

Список литературы