

Отчёт по лабораторной работе № 9

**дисциплина: Архитектура компьютера. Понятие
подпрограммы. Отладчик GDB.**

Студент: Святашова Ксения Евгеньевна

Содержание

1	Цель работы	5
2	Теоритическое введение	6
3	Выполнение лабораторной работы	9
3.1	Реализация подпрограмм в NASM	9
3.2	Отладка программ с помощью GDB	15
3.2.1	Добавление точек останова	21
3.2.2	Работа с данными программы в GDB	23
3.2.3	Обработка аргументов командной строки в GDB . . .	28
4	Задания для самостоятельной работы	32
5	Вывод	41

Список иллюстраций

3.1	Каталог lab09	9
3.2	Ввод текста из листинга 9.1	10
3.3	Запуск исполняемого файла	12
3.4	Изменение текста программы	13
3.5	Запуск исполняемого файла	15
3.6	Создание файла lab09-2.asm	15
3.7	Ввод текста из листинга 9.2.	16
3.8	Получение исполняемый файл	17
3.9	Отладчик gdb	18
3.10	run	18
3.11	Установка брейкпоинта	19
3.12	Дисассимилированный код	19
3.13	Intel'овский синтаксис	20
3.14	Режим псевдографики	21
3.15	Информация о точках останова	22
3.16	Установка точки останова	22
3.17	Информация об установленных точках останова	23
3.18	Команда si	24
3.19	Измененные регистры	25
3.20	Значение переменной msg1	25
3.21	Значение переменной msg2	26
3.22	Изменение значения переменной msg1	26
3.23	Изменение значения переменной msg2	27
3.24	Изменение значения регистра ebx	27
3.25	Завершение и выход из программы	28
3.26	Копирование файла lab8-2.asm	28
3.27	Создание исполняемого файла	28
3.28	Загрузка исполняемого файла в отладчик	29
3.29	Точка останова	30
3.30	Адрес вершины стека	30
3.31	Позиции стека	31

4.1	Файл lab09-4.asm	32
4.2	Измененная программа	33
4.3	Запуск исполняемого файла	35
4.4	Запуск исполняемого файла	35
4.5	Создание файла lab09-5.asm	35
4.6	Ввод текста из листинга 9.3	37
4.7	Неправильный ответ программы	37
4.8	Запуск программы с помощью отладчика GDB	38
4.9	Анализ регистров	39
4.10	Запуск исправленной программы	39

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Теоритическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки;
- поиск её местонахождения;
- определение причины ошибки;
- исправление ошибки.

Можно выделить следующие типы ошибок:

- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;
- семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата;
- ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы.

Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

Наиболее часто применяют следующие методы отладки:

- создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);
- использование специальных программ-отладчиков.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

GDB может выполнять следующие действия:

- начать выполнение программы, задав всё, что может повлиять на её поведение;
- остановить программу при указанных условиях;
- исследовать, что случилось, когда программа остановилась;
- изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

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. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`.

Для продолжения остановленной программы используется команда

`continue (c) (gdb)` с [аргумент]. Выполнение программы будет происходить до следующей точки останова.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

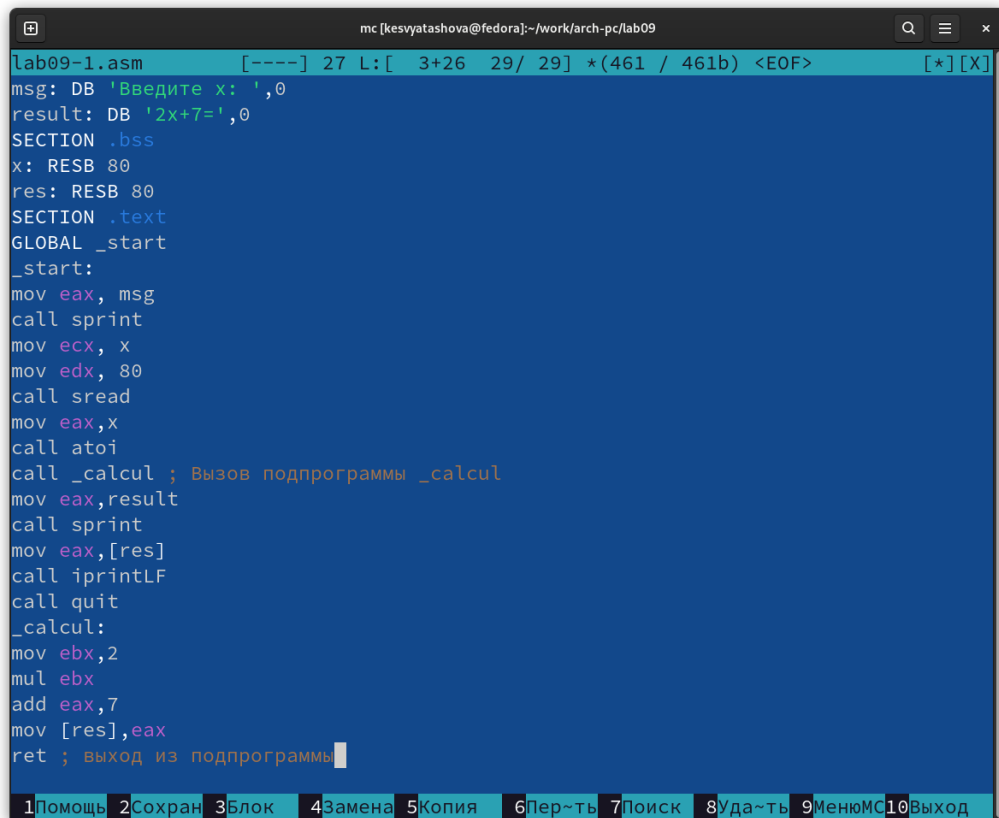
1. Создадим каталог для выполнения лабораторной работы № 9, перейдем в него и создайте файл lab09-1.asm:(рис. 3.1):

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

Рис. 3.1: Каталог lab09

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.

Введем в файл lab09-1.asm текст программы из листинга 9.1.(рис. 3.2):



```
mc [kesvyatashova@fedora]:~/work/arch-pc/lab09
lab09-1.asm [----] 27 L: [ 3+26 29/ 29] *(461 / 461b) <EOF> [*] [X]
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 ; 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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюМС10Выход
```

Рис. 3.2: Ввод текста из листинга 9.1

Листинг 9.1. Пример программы с использованием вызова подпрограммы

```
%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
call sprint
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

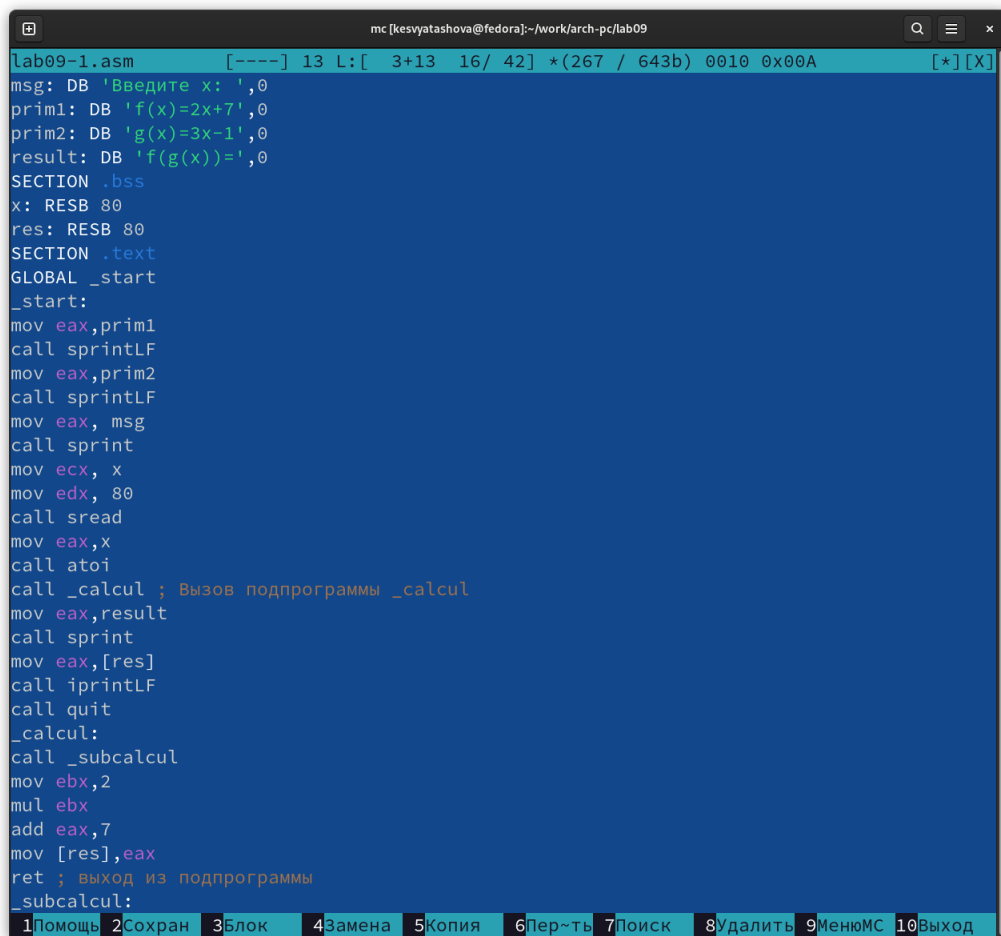
```

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

```
kesvyatashova@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
kesvyatashova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
kesvyatashova@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 17
2x+7=41
kesvyatashova@fedora:~/work/arch-pc/lab09$
```

Рис. 3.3: Запуск исполняемого файла

Изменим текст программы(рис. 3.4), 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.



```
lab09-1.asm  [----] 13 L: [ 3+13 16/ 42] *(267 / 643b) 0010 0x00A [*] [X]
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 ; Вызов подпрограммы _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 ; выход из подпрограммы
_subcalcul:
```

Рис. 3.4: Изменение текста программы

Текст программы:

```
%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
```

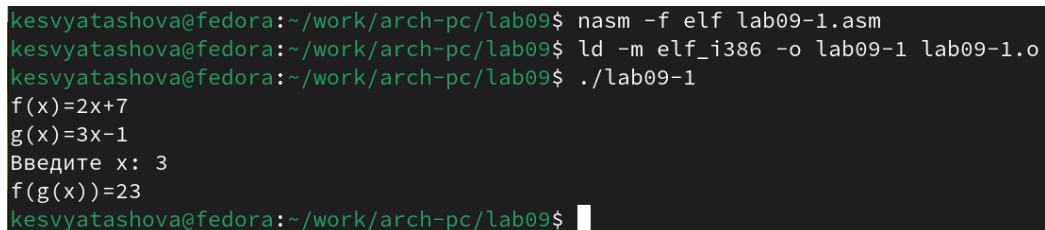
```

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 ; Вызов подпрограммы _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 ; выход из подпрограммы
_subcalcul:
mov ebx,3
mul ebx
add eax,-1
mov [res],eax
ret
```

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

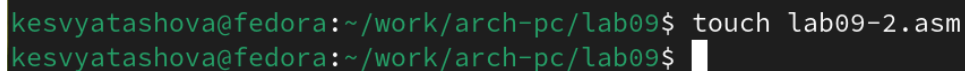


```
kesvyatashova@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
kesvyatashova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
kesvyatashova@fedora:~/work/arch-pc/lab09$ ./lab09-1
f(x)=2x+7
g(x)=3x-1
Введите x: 3
f(g(x))=23
kesvyatashova@fedora:~/work/arch-pc/lab09$
```

Рис. 3.5: Запуск исполняемого файла

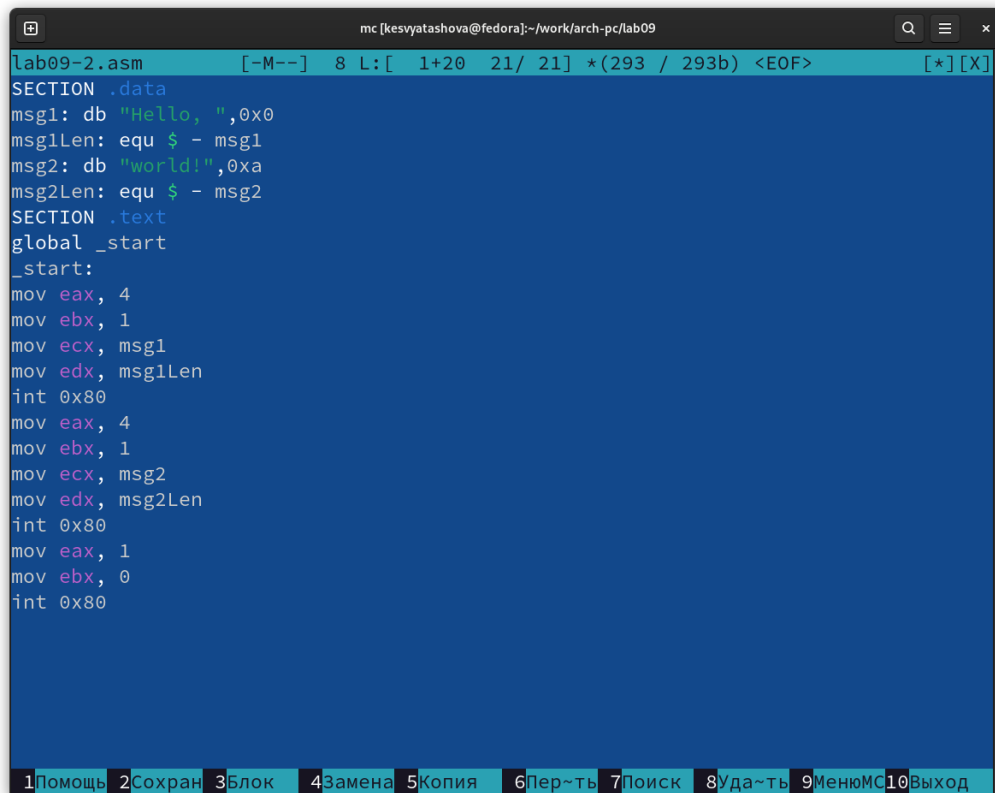
3.2 Отладка программ с помощью GDB

Создадим файл lab09-2.asm(рис. 3.6) с текстом программы из Листинга 9.2.(рис. 3.7)(Программа печати сообщения Hello world!):



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

Рис. 3.6: Создание файла lab09-2.asm



```
lab09-2.asm [-M--] 8 L: [ 1+20 21/ 21] *(293 / 293b) <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
```

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюМС 10Выход

Рис. 3.7: Ввод текста из листинга 9.2.

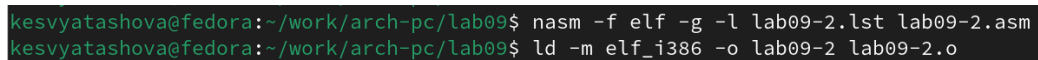
Листинг 9.2. Программа вывода сообщения Hello world!

```
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
```

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'(рис. 3.8):



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

Рис. 3.8: Получение исполняемый файл

Загрузим исполняемый файл в отладчик gdb(рис. 3.9):

```

kesvyatashova@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-3.fc40
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...

```

Рис. 3.9: Отладчик gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды `run` (сокращённо `r`)(рис. 3.10):

```

(gdb) run
Starting program: /home/kesvyatashova/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.
Downloading 47.71 K separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 7378) exited normally]
(gdb)

```

Рис. 3.10: run

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её(рис. 3.11):

```

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

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

```

Рис. 3.11: Установка брейкпоинта

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`(рис. 3.12):

```

(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. 3.12: Дисассимилированный код

Переключимся на отображение команд с Intel'овским синтаксисом,

введя команду `set disassembly-flavor intel`(рис. 3.13):

```
(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. 3.13: Intel'овский синтаксис

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel заключаются в командах. В диссасмилированном отображении в командах используются “%” и “\$”, а в Intel этих символов нет. На это отображение удобнее смотреть.

Включим режим псевдографики для более удобного анализа программы(рис. 3.14) с помощью команд:

```
(gdb) layout asm
```

```
(gdb) layout regs
```

```
kesvyatashova@fedora: ~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcfff 0xffffcfff
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

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 7587 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) 
```

Рис. 3.14: Режим псевдографики

В этом режиме есть три окна:

- В верхней части видны названия регистров и их текущие значения;
- В средней части виден результат дисассимилирования программы;
- Нижняя часть доступна для ввода команд.

3.2.1 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints` (кратко `i b`)(рис. 3.15):

```

native process 7587 (asm) In: _start
(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. 3.15: Информация о точках останова

Установим еще одну точку останова по адресу инструкции(рис. 3.16):

```

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

```

```

exec No process (asm) In:
(gdb) layout regs
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab09-2.asm:9
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) █

```

Рис. 3.16: Установка точки останова

Посмотрим информацию о всех установленных точках останова(рис. 3.17):

```
(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
2        breakpoint     keep y   0x08049031 lab09-2.asm:20
(gdb) █
```

Рис. 3.17: Информация об установленных точках останова

3.2.2 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных.

С помощью команды `si` посмотрим регистры и изменим их (рис. 3.18):

```
kesvyatashova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd010 0xffffd010
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>

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 3766 (asm) In: _start L10 PC: 0x8049005
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y  0x08049000 lab09-2.asm:9
2      breakpoint      keep y  0x08049031 lab09-2.asm:20
(gdb) si
The program is not being run.
(gdb) run
Starting program: /home/kesvyatashova/work/arch-pc/lab09/lab09-2

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

Рис. 3.18: Команда si

Измененные регистры выглядят так(рис. 3.19):


```

eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd010 0xffffd010
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 3.19: Измененные регистры

Для отображения содержимого памяти можно использовать команду `x`, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: `x/NFU`.

С помощью команды `x &` также можно посмотреть содержимое переменной.

Посмотрим значение переменной `msg1` по имени (рис. 3.20):

```

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

```

Рис. 3.20: Значение переменной `msg1`

Посмотрим значение переменной `msg2` по адресу (рис. 3.21):

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

Рис. 3.21: Значение переменной msg2

Изменить значение для регистра или ячейки памяти можно с помощью команды `set`, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс `$`, а перед адресом нужно указать в фигурных скобках тип данных.

Изменим первый символ переменной `msg1` (рис. 3.22):

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

Рис. 3.22: Изменение значения переменной msg1

Изменим символ переменной `msg2` (рис. 3.23):

```

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

```

Рис. 3.23: Изменение значения переменной msg2

С помощью команды set изменим значение регистра ebx(рис. 3.24):

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) █

```

Рис. 3.24: Изменение значения регистра ebx

Команда выводит два разных значения, потому что в первый раз мы вносим значение 2, а во второй - регистр равен двум, поэтому значения отличаются.

Завершим выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдем из GDB с помощью команды quit (сокращенно q)(рис. 3.25):

```
(gdb) continue
Continuing.
Hello, world!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) quit
A debugging session is active.

        Inferior 1 [process 4163] will be killed.

Quit anyway? (y or n)
```

Рис. 3.25: Завершение и выход из программы

3.2.3 Обработка аргументов командной строки в GDB

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. 3.26):

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

Рис. 3.26: Копирование файла lab8-2.asm

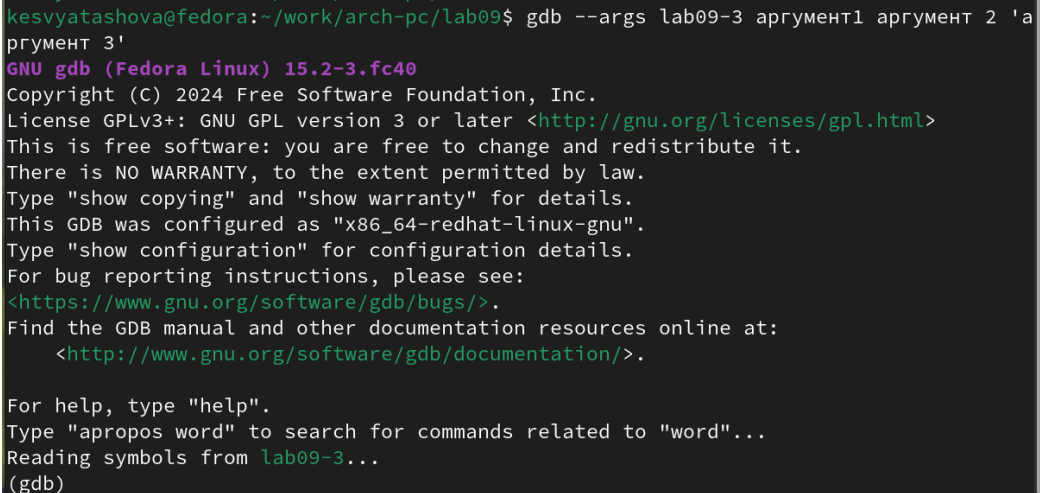
Создадим исполняемый файл(рис. 3.27):

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

Рис. 3.27: Создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ `--args`.

Загрузим исполняемый файл в отладчик, указав аргументы(рис. 3.28):



```
kesvyatashova@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'а  
ргумент 3'  
GNU gdb (Fedora Linux) 15.2-3.fc40  
Copyright (C) 2024 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law.  
Type "show copying" and "show warranty" for details.  
This GDB was configured as "x86_64-redhat-linux-gnu".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
<https://www.gnu.org/software/gdb/bugs/>.  
Find the GDB manual and other documentation resources online at:  
  <http://www.gnu.org/software/gdb/documentation/>.  
  
For help, type "help".  
Type "apropos word" to search for commands related to "word"...  
Reading symbols from lab09-3...  
(gdb)
```

Рис. 3.28: Загрузка исполняемого файла в отладчик

При запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb.

Для начала установим точку останова перед первой инструкцией в программе и запустим ее(рис. 3.29):

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/kesvyatashova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2
аргумент\ 3

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 ; Извлекаем из стека в `ecx` количество
(gdb)

```

Рис. 3.29: Точка останова

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы)(рис. 3.30):

```

(gdb) x/x $esp
0xffffc0b0:      0x00000005
(gdb)

```

Рис. 3.30: Адрес вершины стека

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрим остальные позиции стека(рис. 3.31):

```

(gdb) x/x $esp
0xfffffcfb0:    0x000000005
(gdb) x/s *(void**)(esp + 4)
0xfffffd171:    "/home/kesvyatashova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffd1a0:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffd1b2:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffd1c3:    "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffd1c5:    "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb) █

```

Рис. 3.31: Позиции стека

Элементы расположены с интервалом в 4 единицы, потому что стек может хранить до 4 байт, и для того, чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации.

4 Задания для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы).

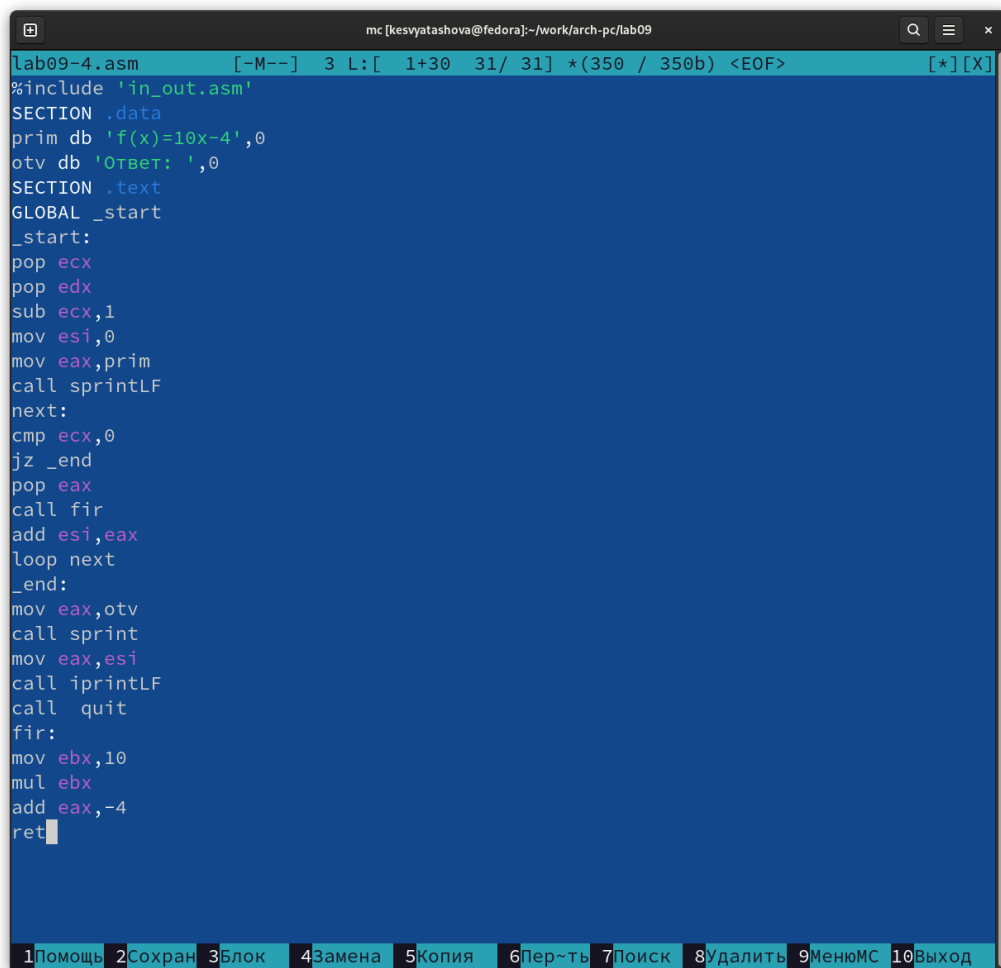
Скопируем файл и переименуем его в lab09-4.asm(рис. 4.1):

A terminal window with a dark background. The prompt is 'kesvyatashova@fedora: ~\$'. The command entered is 'cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm'. The output is the same prompt 'kesvyatashova@fedora: ~\$' on the next line.

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

Рис. 4.1: Файл lab09-4.asm

Реализуем вычисление значения функции $f(x)$ как подпрограмму(рис. 4.2):



```
lab09-4.asm [-M--] 3 L: [ 1+30 31/ 31] *(350 / 350b) <EOF> [*] [X]
#include 'in_out.asm'
SECTION .data
prim db 'f(x)=10x-4',0
otv db 'ОТВЕТ: ',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
pop eax
call fir
add esi,eax
loop next
_end:
mov eax,otv
call sprintf
mov eax,esi
call iprintLF
call quit
fir:
mov ebx,10
mul ebx
add eax,-4
ret
```

Рис. 4.2: Измененная программа

Текст программы:

```
%include 'in_out.asm'

SECTION .data
prim db 'f(x)=10x-4',0
otv db 'ОТВЕТ:',0

SECTION .text
GLOBAL _start
```

```

_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
mov eax,prim
call sprintLF
next:
cmp ecx,0
jz _end
pop eax
call fir
add esi,eax
loop next
_end:
mov eax,otv
call sprint
mov eax,esi
call iprintLF
call quit
fir:
mov ebx,10
mul ebx
add eax,-4
ret

```

Создадим исполняемый файл и запустим его(рис. 4.3) и (рис. 4.4):

```
kesvyatashova@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
kesvyatashova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
```

Рис. 4.3: Запуск исполняемого файла

```
kesvyatashova@fedora:~/work/arch-pc/lab09$ ./lab09-4 7 2 13 10
f(x)=10x-4
Ответ: 4048736770
```

Рис. 4.4: Запуск исполняемого файла

2. Создадим файл для решения №2 самостоятельной работы(рис. 4.5):

```
kesvyatashova@fedora:~/work/arch-pc/lab09$ touch lab09-5.asm
kesvyatashova@fedora:~/work/arch-pc/lab09$
```

Рис. 4.5: Создание файла lab09-5.asm

В листинге 9.3 приведена программа вычисления выражения $(3+2)*4+5$. При запуске данная программа дает неверный результат.

****Листинг 9.3. Программа вычисления выражения $(3+2)*4+5$ ****

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

```
SECTION .data
```

```
div: DB 'Результат:',0
```

```
SECTION .text
```

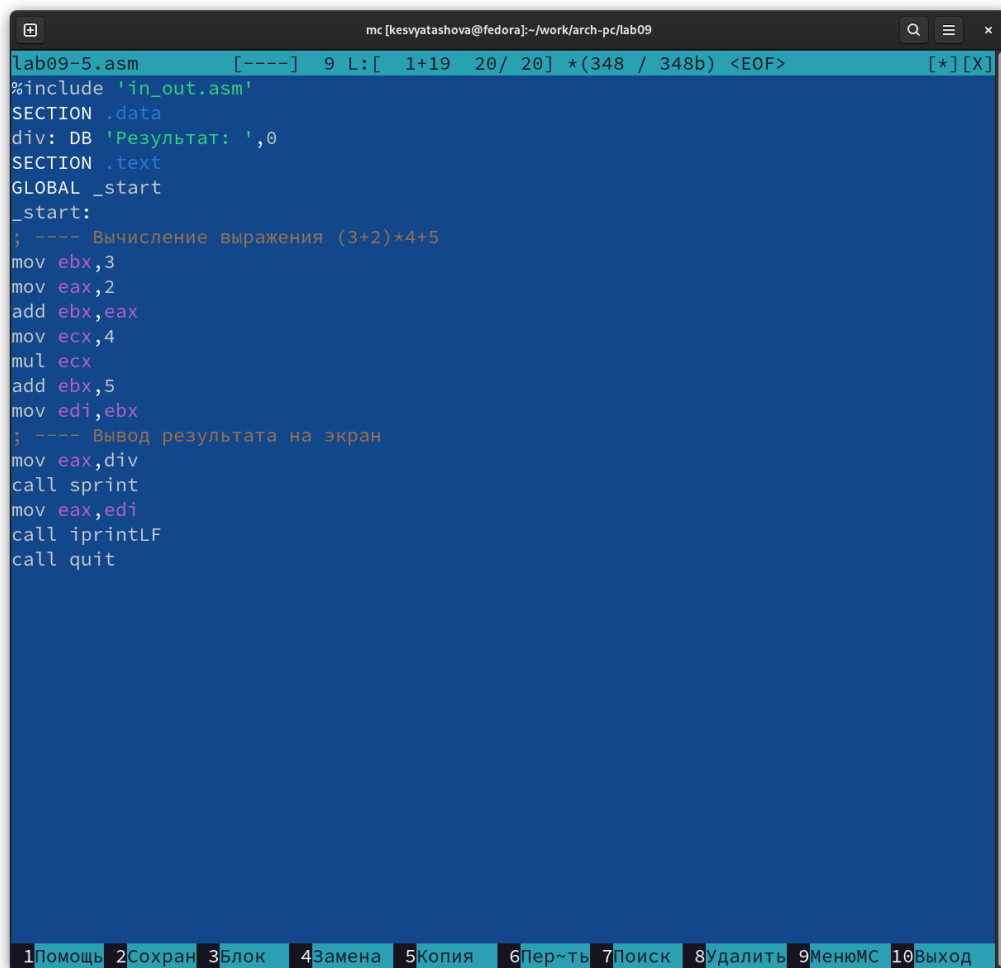
```
GLOBAL _start
```

```
_start:
```

```
; -- Вычисление выражения  $(3+2)*4+5$ 
```

```
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; -- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Ввод текста из листинга 9.3.(рис. 4.6):

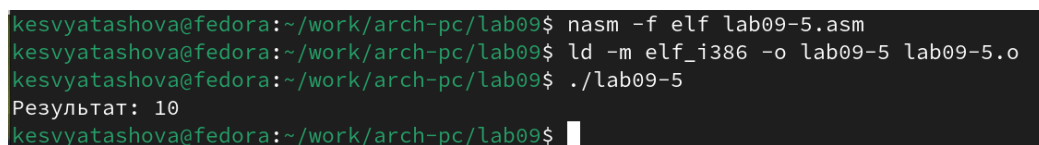


```
lab09-5.asm [----] 9 L: [ 1+19 20/ 20] *(348 / 348b) <EOF> [*] [X]
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Удалить 9МенюМС 10Выход

Рис. 4.6: Ввод текста из листинга 9.3

Проверим, что программа выводит неправильный ответ(рис. 4.7):



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

Рис. 4.7: Неправильный ответ программы

С помощью отладчика GDB запустим программу(рис. 4.8):

```
kesvyatashova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
kesvyatashova@fedora:~/work/arch-pc/lab09$ gdb lab09-5
GNU gdb (Fedora Linux) 15.2-3.fc40
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-5...
(no debugging symbols found in lab09-5)
(gdb) b _start
Breakpoint 1 at 0x080490e8
(gdb) r
Starting program: /home/kesvyatashova/work/arch-pc/lab09/lab09-5

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, 0x080490e8 in _start ()
(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
```

Рис. 4.8: Запуск программы с помощью отладчика GDB

Проанализировав изменения значений регистров(рис. 4.9), понятно, что некоторые регистры стоят не на своих местах.

```
kesvyatashova@fedora:~/work/arch-pc/lab09 — gdb lab09-5

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcfff 0xffffcfff
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]

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

native process 5288 (asm) In: _start          L??  PC: 0x80490e8
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kesvyatashova/work/arch-pc/lab09/lab09-5

Breakpoint 1, 0x80490e8 in _start ()
(gdb) █
```

Рис. 4.9: Анализ регистров

Исправив регистры, запустим программу(рис. 4.10):

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

Рис. 4.10: Запуск исправленной программы

Теперь программа действительно выводит правильный ответ. Программа работает верно.

5 Вывод

В результате выполнения работы я приобрела навыки написания программ с использованием подпрограмм и познакомилась с методами отладки при помощи GDB и его основными возможностями.