Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Калашникова Ольга Сергеевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью работы является 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

Создаём каталог для программам лабораторной работы № 9 (при помощи команды mkdir ~/work/arch-pc/lab09), переходим в него (при помощи команды cd ~/work/arch-pc/lab09) и создаём файл lab09-1.asm (при помощи команды touch lab09-1.asm) (рис. 2.1),(рис. 2.2)

```
oskalashnikova@dk6n62:~$ mkdir ~/work/arch-pc/lab09
oskalashnikova@dk6n62:~$ cd ~/work/arch-pc/lab09
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ touch lab09-1.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 2.1: Создание папки, её открытие и создание файла

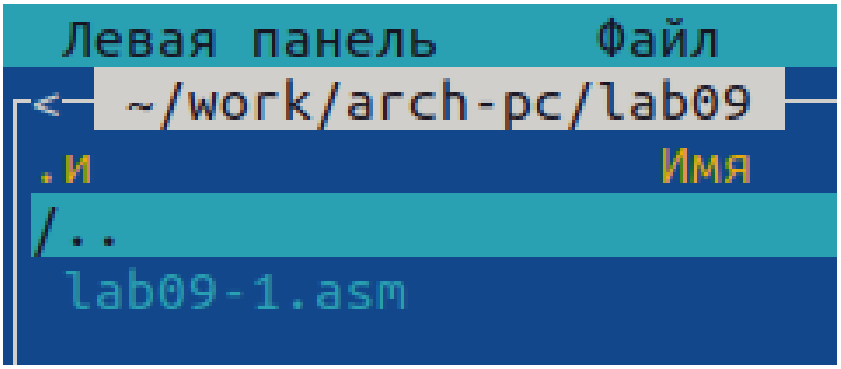

Рис. 2.2: Проверка наличия папки и файла

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 в файл lab09-1.asm текст программы из данного листинга. Программа вычисляет арифметическое выражение f(x) = 2x + 7 с помощью подпрограммы _calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме для корректной работы нужно переместить файл “in_out.asm” в тот же каталог, где лежит и файл с текстом программы.(рис. 2.3),(рис. 2.4)

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

Рис. 2.3: Текст программы

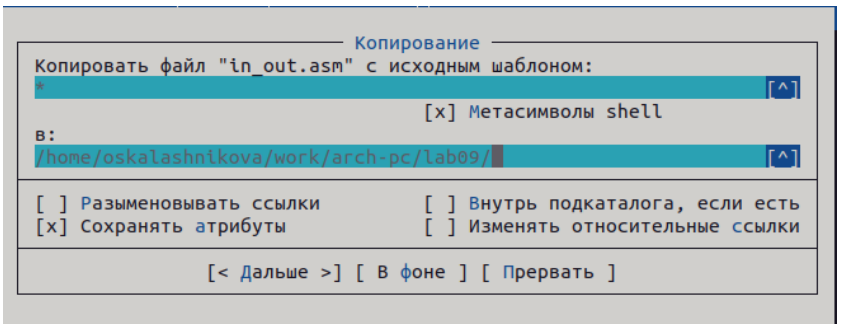

Рис. 2.4: Перемещение файла “in_out.asm”

Создаем исполняемый файл и запускаем его (компиляция: nasm -f elf lab09-1.asm , ld -m elf_i386 -o lab09-1 lab09-1.o , запуск: ./lab09-1) (рис. 2.5)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2x+7=27
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 2.5: Создание и запуск исполняемого файла

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. 2.6)

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

Рис. 2.6: Изменённый текст программы

Создаем исполняемый файл и запускаем его (компиляция: nasm -f elf lab09-1-2.asm , ld -m elf_i386 -o lab09-1-2 lab09-1-2.o , запуск: ./lab09-1-2) (рис. 2.7)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ nasm -f elf lab09-1-2.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1-2 lab09-1-2.o
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ./lab09-1-2
Введите x: 5
f(g(x))=35
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 2.7: Создание и запуск исполняемого файла

Листинг №1:

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',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
```

```
_subcalcul:
mov ebx,3
mul ebx
dec eax
mov [res],eax
ret
; Подпрограмма вычисления выражения "2x+7"
_calcul:
call _subcalcul ; Вызов подпрограммы _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
```

2.2 Отладка программам с помощью GDB

Создаём файл lab09-1.asm (при помощи команды touch lab09-2.asm) (рис. 2.8)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ touch lab09-2.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 2.8: Проверка наличия папки и файла

Введём в файл lab09-2.asm текст программы из данного листинга. Программа печатает сообщение Hello world! (рис. 2.9)

Рис. 2.9: Текст программы

Создаем исполняемый файл и запускаем его (компиляция: nasm -f elf lab09-

2.asm , ld -m elf_i386 -o lab09-2 lab09-2.o , запуск: ./lab09-2) (рис. 2.10)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ nasm -f elf lab09-2.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ./lab09-2
Hello, world!
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 2.10: Создание и запуск исполняемого файла

Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’ (nasm -f elf -g -l lab09-2.lst lab09-2.asm , ld -m elf_i386 -o lab09-2 lab09-2.o) (рис. 2.11)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 2.11: Трансляция с ключом ‘-g’

Загружаем исполняемый файл в отладчик gdb (при помощи команды gdb lab09-2), проверяем работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 2.12)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<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) run
Starting program: /home/oskalashnikova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 130149) exited normally]
(gdb)
```

Рис. 2.12: Загрузка исполняемого файла в отладчик gdb и проверка работы программы

Для более подробного анализа программы установим брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её (рис. 2.13)

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

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

Рис. 2.13: Установка брейкпоинта на метку _start и запуск программы

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start (рис. 2.14)

```
(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. 2.14: Дисассимилированный код программы начиная с метки _start

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel (рис. 2.15)

```
(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. 2.15: Синтаксис Intel

Вопрос: Перечислите различия отображения синтаксиса машинных команд в режимах ATT и Intel

Ответ:

Из того что я увидела, могу отметить

1. В синтаксисе ATT порядок операндов обычно другой, чем в Intel (В ATT сначала указывается исходный операнд, а потом результирующий). Например, в команде mov синтаксис ATT будет: mov $0x4, %eax, а в синтаксисе Intel: mov eax, 0x4.

2. В синтаксисе ATT со знака доллара ($) начинаются имена операндов, в то время как в синтаксисе Intel этот знак обычно не используется. Например, mov $0x4, %eax в синтаксисе ATT и mov eax, 0x4 в синтаксисе Intel

3. В синтаксисе ATT со знака процента (%) начинаются имена регистров, в то время как в синтаксисе Intel этот знак обычно не используется. Например, mov $0x4, %eax в синтаксисе ATT и mov eax, 0x4 в синтаксисе Intel

Включим режим псевдографики для более удобного анализа программы (при помощи команд layout asm и layout regs) (рис. 2.16)

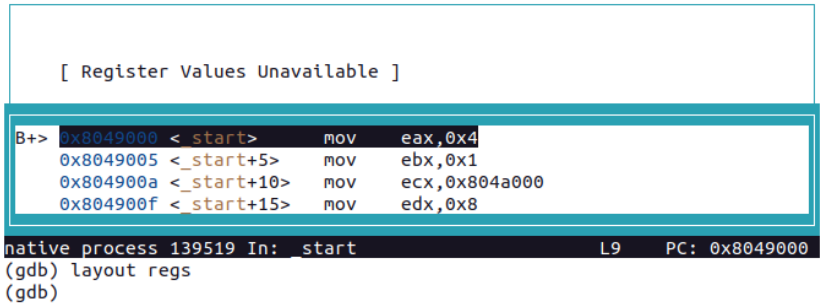

Рис. 2.16: Режим псевдографики

2.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (_start). Проверим это с помощью команды info breakpoints (i b)(рис. 2.17)

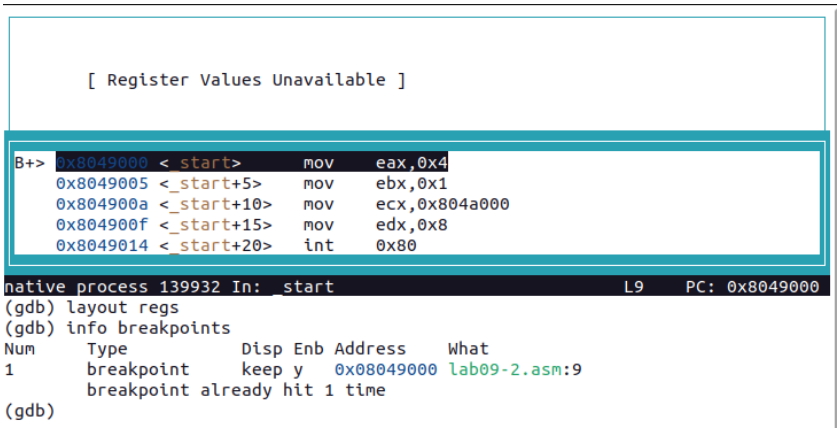

Рис. 2.17: Информация о точках останова

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определим адрес предпоследней инструкции (mov ebx,0x0) и установим

точку останова(b *0x8049031) и посмотрим информацию о всех установленных точках останова(i b)(рис. 2.18)

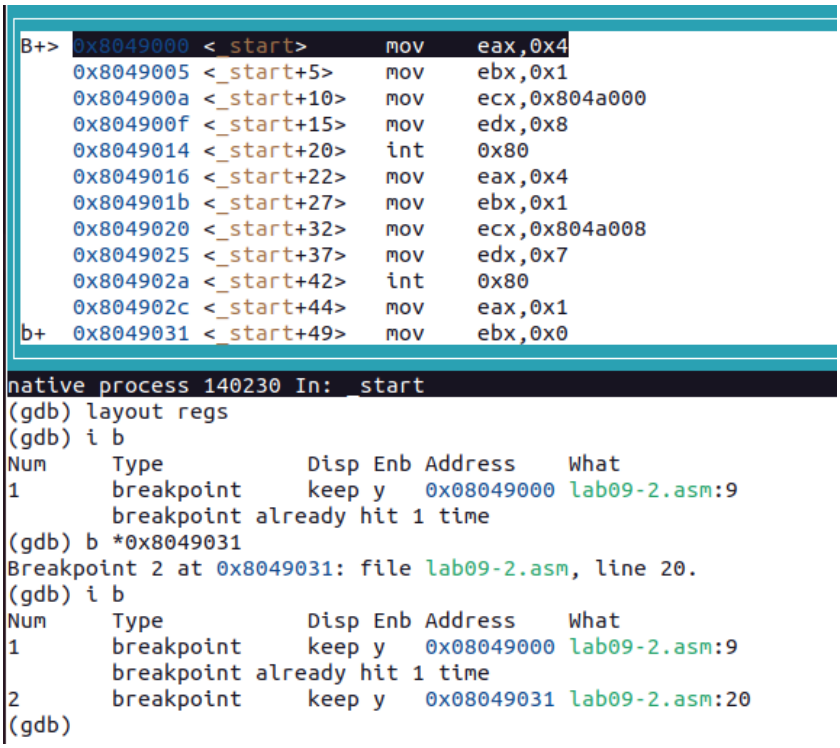

Рис. 2.18: Установка точки останова и информация о точках останова

2.4 Работа с данными программы в GDB

Выполним 5 инструкций с помощью команды stepi (si) и проследим за изменением значений регистров(рис. 2.19), (рис. 2.20)

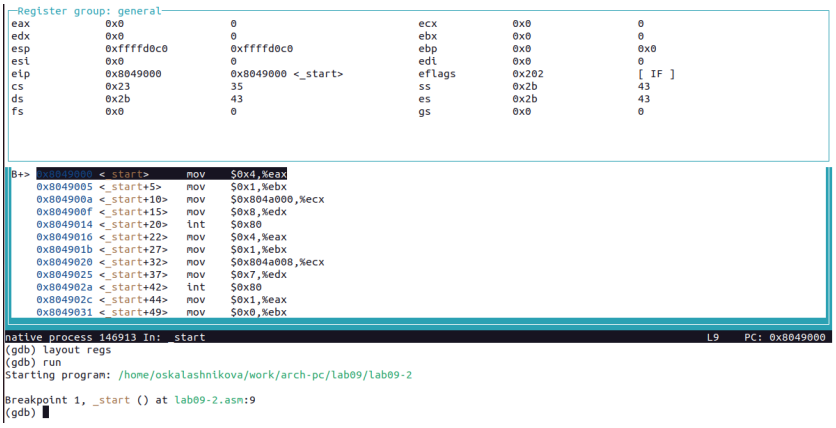

Рис. 2.19: До использования команды stepi

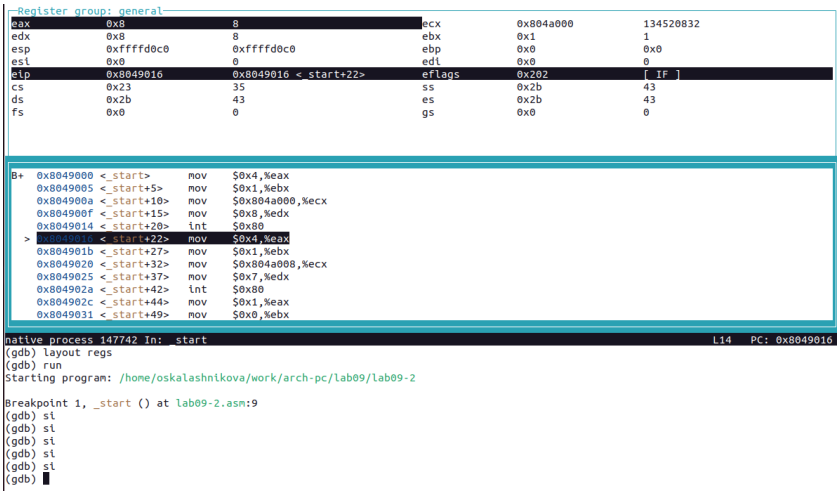

Рис. 2.20: 5 инструкций с помощью команды stepi

Вопрос: Значения каких регистров изменяются?

Ответ: eax, edx, eip, ecx, ebx

Посмотрим содержимое регистров теперь с помощью команды info registers(i r)(рис. 2.21)

```
native process 147742 In: _start
eax            0x8                8
ecx            0x804a000          134520832
edx            0x8                8
ebx            0x1                1
esp            0xffffd0c0         0xffffd0c0
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x8049016          0x8049016 <_start+22>
eflags         0x202              [ IF ]
cs             0x23               35
ss             0x2b               43
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.21: Просмотр содержимого регистров с помощью команды i r

С помощью команды x/1sb &msg1 посмотрим содержимое переменной msg1 по имени (рис. 2.22)

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

Рис. 2.22: Содержимое переменной msg1

С помощью команды x/1sb &msg2 посмотрим содержимое переменной msg2 по имени (рис. 2.23)

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

Рис. 2.23: Содержимое переменной msg2

Так выглядит инструкция mov ecx,msg2 которая записывает в регистр ecx адрес перемененной msg2 (рис. 2.24)

```
0x8049020 <_start+32>    mov    $0x804a008,%ecx
```

Рис. 2.24: Инструкция mov ecx,msg2

Изменим первый символ переменной msg1 с помощью команды set {char}&msg1='h' и проверим (команда x/1sb &msg1) (рис. 2.25)

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

Рис. 2.25: Изменение первого символа переменной msg1

Изменим первый символ переменной msg2 с помощью команды set {char}&msg2='t' и проверим (команда x/1sb &msg2) (рис. 2.26)

```
(gdb) set {char}&msg2='t'
(gdb) x/1sb &msg2
(gdb) 008 <msg2>:       "torld!\n\034"
```

Рис. 2.26: Изменение первого символа переменной msg1

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 2.27)

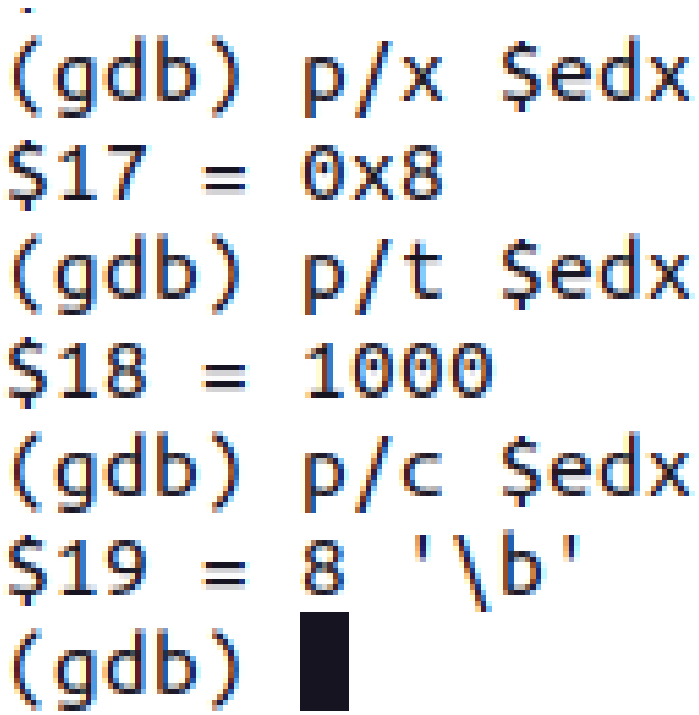

Рис. 2.27: Значение регистра edx в различных форматах

С помощью команды set изменим значение регистра ebx в соответсвии заданию(set $ebx=‘2’, p/s $ebx) (set $ebx=2 , p/s $ebx) (рис. 2.28)

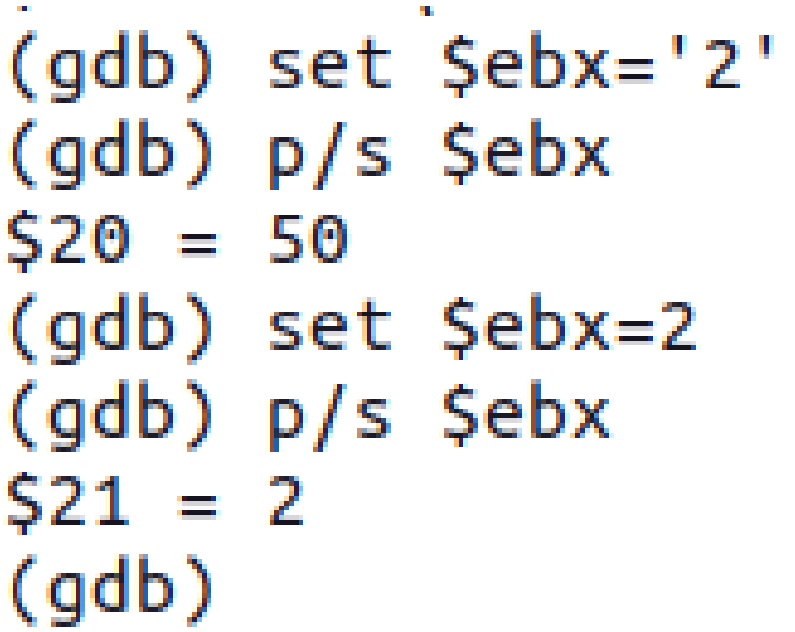

Рис. 2.28: Изменение значения регистра ebx

Вопрос: Объясните разницу вывода команд p/s $ebx

Ответ: В первом случае мы использовали символ “2” (по таблице ASCII мы видим что этому символу соответствует значение 50), а во втором само число

Завершим выполнение программы с помощью команды continue (c) и выйдем из GDB с помощью команды quit (q) (рис. 2.29)

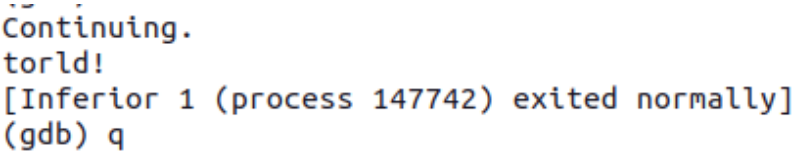

Рис. 2.29: Изменение значения регистра ebx

2.5 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 в файл с

именем lab09-3.asm (при помощи команды cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm) (рис. 2.30), (рис. 2.31)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 2.30: Копирование файла

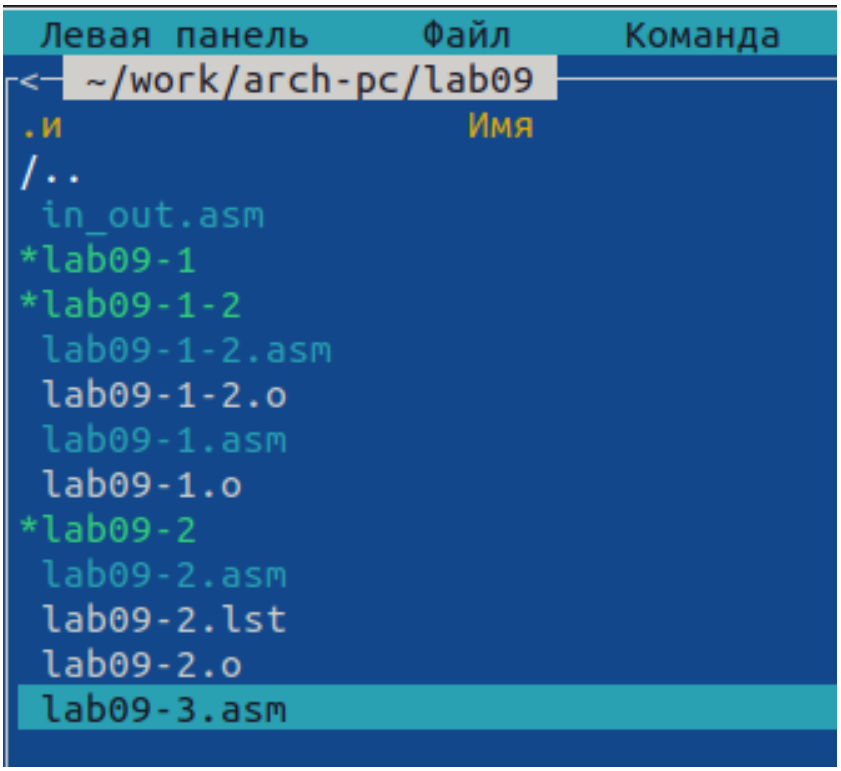

Рис. 2.31: Проверка наличия файла

Создаём исполняемый файл (при помощи команд nasm -f elf -g -l lab09-3.lst lab09-3.asm и ld -m elf_i386 -o lab09-3 lab09-3.o)(рис. 2.32)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 2.32: Создание исполняемого файла

Загружаем исполняемый файл в отладчик, указав аргументы: gdb –args lab09-3 аргумент1 аргумент 2 ‘аргумент 3’ (рис. 2.33)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<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. 2.33: Загрузка исполняемого файла в отладчик с аргументами

Установим точку останова перед первой инструкцией в программе и запустим ее (при помощи b _start и run) (рис. 2.34)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 10.
(gdb) run
Starting program: /home/oskalashnikova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:10
10      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 2.34: Установка точку останова и запуск

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’ (проверим при помощи x/x $esp) (рис. 2.35)

```
(gdb) x/x $esp
0xffffd080:     0x00000005
(gdb)
```

Рис. 2.35: Вершина стека

Посмотрим остальные позиции стека (рис. 2.36)

```
(gdb) x/s *(void**)($esp + 4)
0xffffd260:     "/home/oskalashnikova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd290:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd2a2:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd2b3:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd2b5:     "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.36: Вершина стека

Вопрос: Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.).

Ответ: Шаг изменения адреса равен 4, так как размер слова - 4 байта. Когда мы обращаемся к следующему элементу в стеке, мы увеличиваем адрес вершины стека на 4. Таким образом, шаг изменения адреса равен 4 для обеспечения эффективной работы с данными.

3 Задание для самостоятельной работы

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

Редактируем программу из лабораторной работы №8 с добавлением подпрограммы (рис. [3.1]).

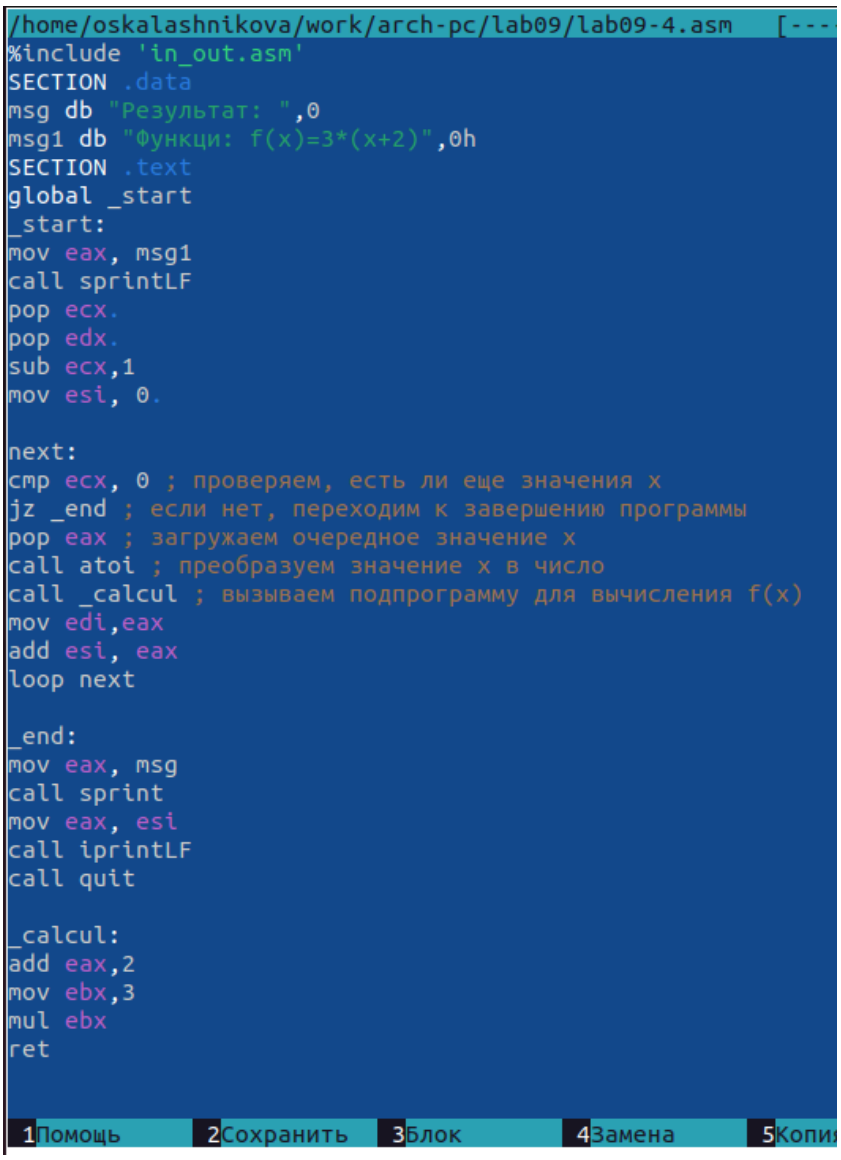

Рис. 3.1: Изменённый текст

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg1 db "Функци: f(x)=3*(x+2)",0h
SECTION .text
global _start
_start:
```

```
mov eax, msg1
call sprintLF
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx, 0 ; проверяем, есть ли еще значения x
jz _end ; если нет, переходим к завершению программы
pop eax ; загружаем очередное значение x
call atoi ; преобразуем значение x в число
call _calcul ; вызываем подпрограмму для вычисления f(x)
mov edi,eax
add esi, eax
loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit

_calcul:
add eax,2
mov ebx,3
mul ebx
ret
```

Создаем исполняемый файл и запускаем его (компиляция: nasm -f elf lab09-4.asm , ld -m elf_i386 -o lab09-4 lab09-4.o , запуск: ./lab09-4) (рис. 3.2)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ./lab09-4 1 2 3 4
Функци: f(x)=3*(x+2)
Результат: 54
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ./lab09-4 1 2 3 4 5
Функци: f(x)=3*(x+2)
Результат: 75
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 3.2: Создание и запуск исполняемого файла

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.

Вставляем неправильно работающую программу (рис. 3.3)

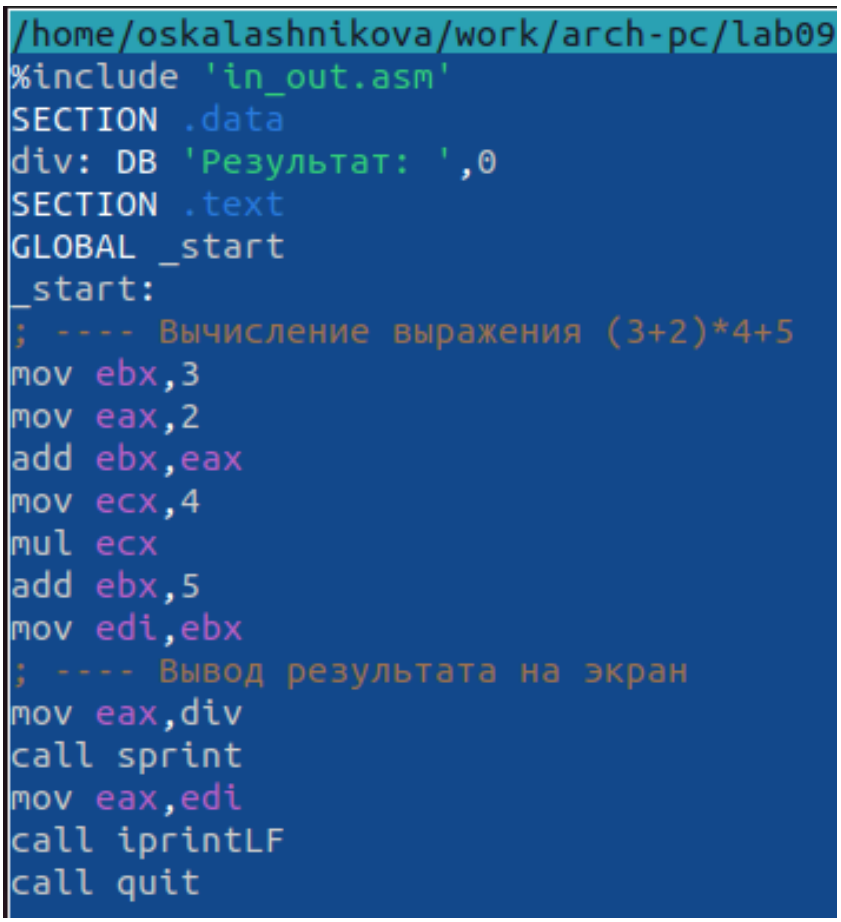

Рис. 3.3: Редактирование файла

Создаем исполняемый файл и запускаем его запускаем его в отладчике GDB (рис. 3.4)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ gdb lab09-5
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<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) run
Starting program: /home/oskalashnikova/work/arch-pc/lab09/lab09-5
Результат: 10
[Inferior 1 (process 184972) exited normally]
```

Рис. 3.4: Создание и запуск исполняемого файла

Находим ошибку, анализируя изменения значений регистров (рис. 3.5)

```
─Register group: general─
eax            0x8                8                  ecx            0x4            4
edx            0x0                0                  ebx            0x5            5
esp            0xffffd0c0         0xffffd0c0         ebp            0x0            0x0
esi            0x0                0                  edi            0x0            0
eip            0x80490fb          0x80490fb <_start+19>  eflags     0x10202        [ IF RF ]
cs             0x23               35                 ss             0x2b           43
ds             0x2b               43                 es             0x2b           43
fs             0x0                0                  gs             0x0            0

B+  0x80490e8 <_start>      mov    $0x3,%ebx
    0x80490ed <_start+5>    mov    $0x2,%eax
    0x80490f2 <_start+10>   add    %eax,%ebx
    0x80490f4 <_start+12>   mov    $0x4,%ecx
    0x80490f9 <_start+17>   mul    %ecx
  > 0x80490fb <_start+19>   add    $0x5,%ebx
    0x80490fe <_start+22>   mov    %ebx,%edi
    0x8049100 <_start+24>   mov    $0x804a000,%eax
    0x8049105 <_start+29>   call   0x804900f <sprint>
    0x804910a <_start+34>   mov    %edi,%eax
    0x804910c <_start+36>   call   0x8049086 <iprintLF>
    0x8049111 <_start+41>   call   0x80490db <quit>
native process 185393 In: _start                                          L
The program being debugged has been started already.
Start it from the beginning? (y or n) nProgram not restarted.
(gdb) si
0x080490ed in _start ()
(gdb) si
0x080490f2 in _start ()
(gdb) si
0x080490f4 in _start ()
(gdb) si
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb)
```

Рис. 3.5: Обнаружение ошибки

Обнаружив ошибку неправильной записи регистров, корректируем программу (рис. 3.6)

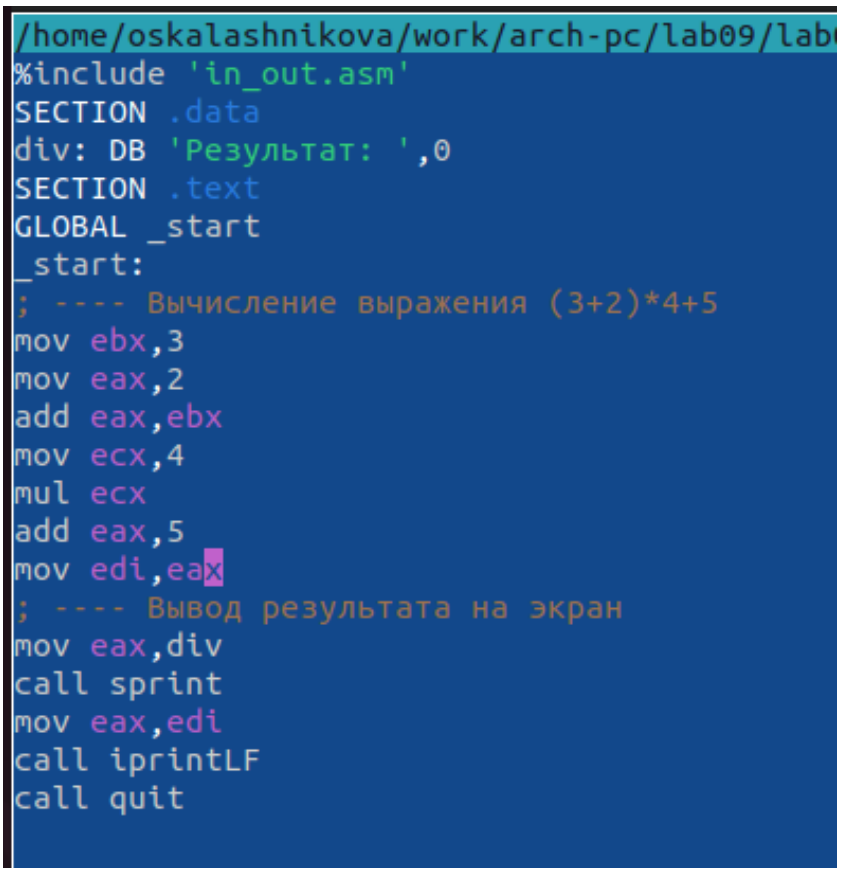

Рис. 3.6: Изменение программы

```
%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 eax,ebx
mov ecx,4
mul ecx
```

```
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Создаем исполняемый файл и запускаем его (компиляция: nasm -f elf lab09-5.asm , ld -m elf_i386 -o lab09-5 lab09-5.o , запуск: ./lab09-5) (рис. 3.7)

```
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
oskalashnikova@dk6n62:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
oskalashnikova@dk6n62:~/work/arch-pc/lab09$
```

Рис. 3.7: Создание и запуск исполняемого файла

4 Выводы

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