

Лабораторная работа №9

. Понятие подпрограммы. Отладчик GDB.

Карпова Есения Алексеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	28
	Список литературы	29

Список иллюстраций

4.1	Запуск исполняемого файла	9
4.2	Изменение текста программы	10
4.3	Запуск исполняемого файла	11
4.4	Текст программы из листинга 9.2	12
4.5	Создание исполняемого файла листинга 9.2	13
4.6	Запуск исполняемого файла	13
4.7	Установка брейкпоинта на метку <code>_start</code>	14
4.8	Переключение на отображение команд с синтаксисом Intel	15
4.9	Режим псевдографики	16
4.10	Установка точек останова	16
4.11	Просмотр информации	17
4.12	Просмотр значения переменной по имени	17
4.13	Просмотр значения переменной по адресу	17
4.14	Использование команды <code>set</code>	17
4.15	Использование команды <code>set</code>	18
4.16	Значения регистра в разных форматах	18
4.17	Использование команды <code>set</code> для изменения значения регистра	19
4.18	Использование команды <code>set</code> для изменения значения регистра	19
4.19	Создание исполняемого файла	20
4.20	Загрузка исполняемого файла в отладчик	20
4.21	Просмотр вершины стека	21
4.22	Изменение текста программы	22
4.23	Запуск исполняемого файла	23
4.24	Текст программы из листинга 9.3	24
4.25	Некорректная работа программы	25
4.26	Ошибка в работе программы	25
4.27	Измененный код	26
4.28	Запуск исполняемого файла	27

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

1. Реализация подпрограмм в NASM
2. Отладка подпрограмм с помощью GDB
3. Добавление точек останова
4. Работа с данными подпрограммы в GDB
5. Обработка аргументов командной строки
6. Задания для самостоятельной работы

3 Теоретическое 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. 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. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия.

Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова: • Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом); • Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его)

GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-

подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. 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.

Синтаксис команды для запуска отладчика имеет следующий вид: `gdb [опции] [имя_файла | ID процесса]` Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB. Команда `kill` (сокращённо `k`) прекращает отладку программы

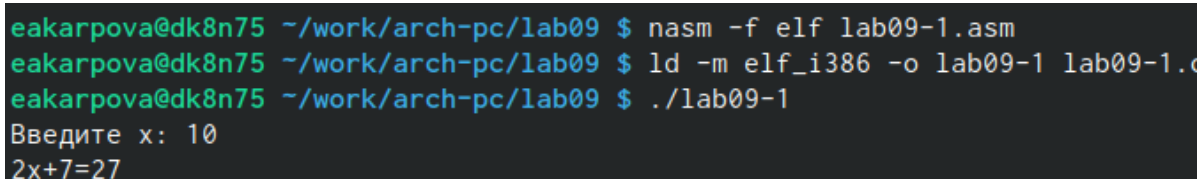
Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`. Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: `(gdb) break *` `(gdb) b` Для продолжения остановленной программы используется команда `continue` (`c`) `(gdb) c [аргумент]`.

Справку о любой команде `gdb` можно получить, введя `(gdb) help [имя_команды]`

4 Выполнение лабораторной работы

1. Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm. Ввожу в файл текст программы с использованием подпрограммы из листинга 9.1 и запускаю исполняемый файл, чтобы проверить его работу (рис. 4.1).



```
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 10
2x+7=27
```

Рис. 4.1: Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры $f(x) = 2x + 7$, $g(x) = 3x - 1$ (рис. 4.2).

```

call _subcalcul
call _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

_subcalcul:
mov ebx,3
mul ebx
add eax,-1
ret

```

Рис. 4.2: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 4.3).

```
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 10
2*(3x-1) + 7=65
```

Рис. 4.3: Запуск исполняемого файла

2. Отладка подпрограмм с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. 4.4).

```
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. 4.4: Текст программы из листинга 9.2

Создаю исполняемый файл и проверяю его работу (рис. 4.5).

```
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ nasm -f elf lab09-2.asm
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ ./lab09-2
Hello, world!
```

Рис. 4.5: Создание исполняемого файла листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g', загружаю исполняемый файл в отладчик gdb и проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (r) (рис. 4.6).

```
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 34340) exited normally]
```

Рис. 4.6: Запуск исполняемого файла

Для подробного анализа программы устанавливаю брейкпоинт на метку _start и запускаю её. Далее просматриваю дисассимилированный код программы с помощью команды disassemble начиная с метки _start (рис. 4.7).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 12.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:12
12      mov eax, 4
(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.

```

Рис. 4.7: Установление брейкпоинта на метку _start

Переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel` (рис. 4.8).

```

(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.

```

Рис. 4.8: Переключение на отображение команд с синтаксисом Intel

В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется привычный синтаксис.

Включая режим псевдографики для более удобного анализа программы с помощью команд `layout` и `layout regs`. (рис. 4.9).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc310 0xffffc310
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 34557 In: _start          L12    PC: 0x8049000
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/e/a/ekarpova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:12
(gdb) █
```

Рис. 4.9: Режим псевдографики

3. Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. (рис. 4.10).

```
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 25.
```

Рис. 4.10: Установление точек останова

Посмотрю информацию о всех установленных точках останова(рис. 4.11).


```
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y   0x08049000 lab09-2.asm:12
          breakpoint already hit 1 time
2        breakpoint    keep y   0x08049031 lab09-2.asm:25
```

Рис. 4.11: Просмотр информации

4. Работа с данными программы GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров: изменились значения регистров `eax`, `ecx`, `edx`, `ebx`.

Просматриваю значение переменной `msg1` по имени с помощью команды `x/1sb&msg1` (рис. 4.12).

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

Рис. 4.12: Просмотр значения переменной по имени

Просматриваю значение переменной `msg2` по ее адресу (рис. 4.13).

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

Рис. 4.13: Просмотр значения переменной по адресу

С помощью команды `set` изменяю первый символ переменной `msg1` (рис. 4.14).

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

Рис. 4.14: Использование команды `set`

С помощью команды `set` изменяю первый символ переменной `msg2` (рис. 4.15).

```
(gdb) set {char}&msg2='q'  
(gdb) x/1sb &msg2  
0x804a008 <msg2>: "qor1d!\n\034"
```

Рис. 4.15: Использование команды `set`

Ввожу в шестнадцатеричном, двоичном и символьном формате соответственно значение регистра `eax` с помощью команды `print p/F$val` (рис. 4.16).

```
(gdb) p/t $eax  
$3 = 100  
(gdb) p/x $eax  
$4 = 0x4  
(gdb) p/c $eax  
$5 = 4 '\004'
```

Рис. 4.16: Значения регистра в разных форматах

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием, ввожу число 2 в кавычках (рис. 4.17).

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$6 = 50
```

Рис. 4.17: Использование команды set для изменения значения регистра

С помощью команды set изменяю значение регистра ebx в соответствии с заданием, но ввожу число 2 без кавычек (рис. 4.18).

```
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$7 = 2
```

Рис. 4.18: Использование команды set для изменения значения регистра

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

5. Обработка аргументов командной строки GDB

Копирую 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 и создаю исполняемый файл (рис. 4.19).

```
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 4.19: Создание исполняемого файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа -args и устанавливаю точку останова перед первой инструкцией в программе, после чего запускаю её (рис. 4.20).

```
eakarpova@dk8n75 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 12.1 vanilla) 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 10.
(gdb) r
Starting program: /afs/.dk.scl.pfu.edu.ru/home/e/a/eakarpova/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
```

Рис. 4.20: Загрузка исполняемого файла в отладчик

Посмотриваю вершину стека и его позиции по их адресам (рис. 4.21).

```

(gdb) x/x $esp
0xfffffc2d0:      0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xfffffc566:      "/afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffc5ac:      "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffc5be:      "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xfffffc5cf:      "2"
(gdb) x/s *(void**)( $esp + 20)
0xfffffc5d1:      "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0:      <error: Cannot access memory at address 0x0>

```

Рис. 4.21: Просмотр вершины стека

Шаг изменения адреса - 4, так как количество аргументов командной строки - 4.

6. Задания для самостоятельной работы

- 1) Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $\Phi(\Phi)$ как подпрограмму (рис. 4.22).

```

%include      'in_out.asm'

SECTION .data
msg1: DB 'Функция:f(x) = 10x - 4 ',0
msg2: DB 'Результат: ', 0

SECTION .bss
res: RESB 80

SECTION .text
global _start

_start:
pop ecx      ; извлекаем из стека в 'ecx'
pop edx      ; извлекаем из стека в 'edx'
sub ecx, 1    ; уменьшение 'ecx' на 1
mov esi, 0    ; использование esi для хранения промежуточных сумм
mov edi, 10   ; хранение 10 в edi

next:
cmp ecx, 0h
jz _end
pop eax
call atoi
call _function
mov eax, res
loop next

_function:
mul edi      ; eax = eax * edi
add eax, -4   ; eax = eax - 4
add esi, eax  ; добавление к промежуточной сумме
ret

_end:
mov eax, msg1
call sprintf
mov eax, msg2
call sprintf
mov eax, esi
call iprintLF
call quit

```

Рис. 4.22: Изменение текста программы

Проверяю программу, запуская исполняемый файл (рис. 4.23).

```
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ nasm -f elf lab09-task1.asm
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-task1 lab09-task1.o
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ ./lab09-task1 1 2 3 4 5
Функция:  $f(x) = 10x - 4$ 
Результат: 80
```

Рис. 4.23: Запуск исполняемого файла

2) Ввожу в файл текст программы из листинга 9.3 (рис. 4.24).

```

#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

```

Рис. 4.24: Текст программы из листинга 9.3

Создаю исполняемый файл и запускаю его - программа выдает число 10, значит работа программы некорректна.(рис. 4.25).

```
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ nasm -f elf lab09-task2.asm
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-task2 lab09-task2.o
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ ./lab09-task2
Результат: 10
```

Рис. 4.25: Некорректная работа программы

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды continue прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров. При выполнении инструкции mul ecx происходит умножение ecx на eax, то есть 4 на 2, вместо умножения 4 на 5, так как инструкция add ebx,eax стоящая перед mov ecx,4 не связана с mul ecx, а инструкция mov eax,2 связана (рис. 4.26).

```
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>
```

Рис. 4.26: Ошибка в работе программы

Чтобы исправить ошибку, нужно добавить после add ebx, eax - mov eax,ebx и заменяя ebx на eax в двух инструкциях как показано на рисунке(рис. 4.27).

```

%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 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

```

Рис. 4.27: Измененный код

Создаю исполняемый файл и запускаю его, чтобы убедиться в правильности работы программы(рис. 4.28).

```
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ nasm -f elf lab09-task2.asm
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-task2 lab09-task2.o
eakarpova@dk3n37 ~/work/arch-pc/lab09 $ ./lab09-task2
Результат: 25
```

Рис. 4.28: Запуск исполняемого файла

5 Выводы

Во время лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями

Список литературы

Демидова А.В. - Лабораторная работа №9. Понятие подпрограммы. Отладчик GDB.