

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров

Гафоров Нурмухаммад Вомикович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Реализация подпрограмм в NASM Видим, что в выводе мы получаем неправильный ответ.	10
4.2	Отладка программ с помощью GDB	13
4.2.1	Добавление точек останова	15
4.2.2	Обработка аргументов командной строки в GDB	22
4.3	Задания для самостоятельной работы	24
5	Выводы	29
	Список литературы	30

Список иллюстраций

4.1	Создание файлов для лабораторной работы	10
4.2	Ввод текста программы из листинга 9.1	11
4.3	Запуск исполняемого файла	11
4.4	Изменение текста программы согласно заданию	12
4.5	Изменение текста программы согласно заданию	12
4.6	Запуск исполняемого файла	12
4.7	Ввод текста программы из листинга 9.2	13
4.8	Получение исполняемого файла	13
4.9	Загрузка исполняемого файла в отладчик	13
4.10	Проверка работы файла с помощью команды run	14
4.11	Установка брейкпоинта и запуск программы	14
4.12	Использование команд disassemble и disassembly-flavor intel	14
4.13	Включение режима псевдографики	15
4.14	Установление точек останова и просмотр информации о них	15
4.15	До использования команды stepi	16
4.16	После использования команды stepi	17
4.17	После использования команды stepi	17
4.18	После использования команды stepi	18
4.19	После использования команды stepi	19
4.20	После использования команды stepi	19
4.21	Просмотр значений переменных	20
4.22	Использование команды set	21
4.23	Вывод значения регистра в разных представлениях	22
4.24	Завершение работы GDB	22
4.25	Создание файла или копирование файл	23
4.26	Загрузка файла с аргументами в отладчик	23
4.27	Установление точки останова и запуск программы	23
4.28	Просмотр значений, введенных в стек	23
4.29	Написание кода программы	24
4.30	Запуск программы и проверка его вывода	24
4.31	ВВод текста программы из листинга 9.3	26
4.32	Создание и запуск исполняемого файла	26
4.33	Нахождение причины ошибки	27
4.34	Исправление ошибки	27
4.35	Ошибка исправлена	28

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 программ с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. 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.

GDB (GNU Debugger — отладчик проекта GNU) 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.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки

отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда `quit` (или сокращённо `q`).

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».

Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`.

Обратно точка останова активируется командой `enable`.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`.

Для продолжения остановленной программы используется команда `continue` (`c`). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке).

Команда `stepi` (кратко `sl`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

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. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При

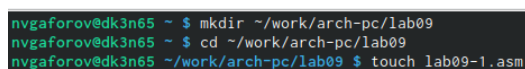
этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `еір`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM Видим, что в выводе мы получаем неправильный ответ.

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm. (рис. [4.1]).



```
nvgafarov@dk3n65 ~ $ mkdir ~/work/arch-pc/lab09
nvgafarov@dk3n65 ~ $ cd ~/work/arch-pc/lab09
nvgafarov@dk3n65 ~/work/arch-pc/lab09 $ touch lab09-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

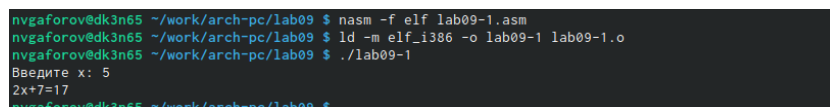
Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. [4.2]).



```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/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 iprintlnLF
call quit
;-----
; Подпрограмма вычисления
[ Прочитано 35 строк ]
^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить М-У Отмена М-А Установить ме
^X Выход ^R ЧитФайл ^N Замена ^U Вставить ^C Позиция М-Е Повтор М-Б Копировать
```

Рис. 4.2: Ввод текста программы из листинга 9.1

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



```
nvgaforov@dk3n65 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
nvgaforov@dk3n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
nvgaforov@dk3n65 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=17
nvgaforov@dk3n65 ~/work/arch-pc/lab09 $
```

Рис. 4.3: 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. [4.4]).

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-1.asm
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _subcalcul
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
_subcalcul:
mov ebx, 3
mul ebx
add eax, -1
ret

```

^O Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить M-U Отмена M-A Установить ме
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^C Позиция M-E Повтор M-B Копировать

Рис. 4.4: Изменение текста программы согласно заданию

Изменение текст 2 фото (рис. [4.5]).

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-1.asm
#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:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x

```

Рис. 4.5: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу.(рис. [4.6]).

```

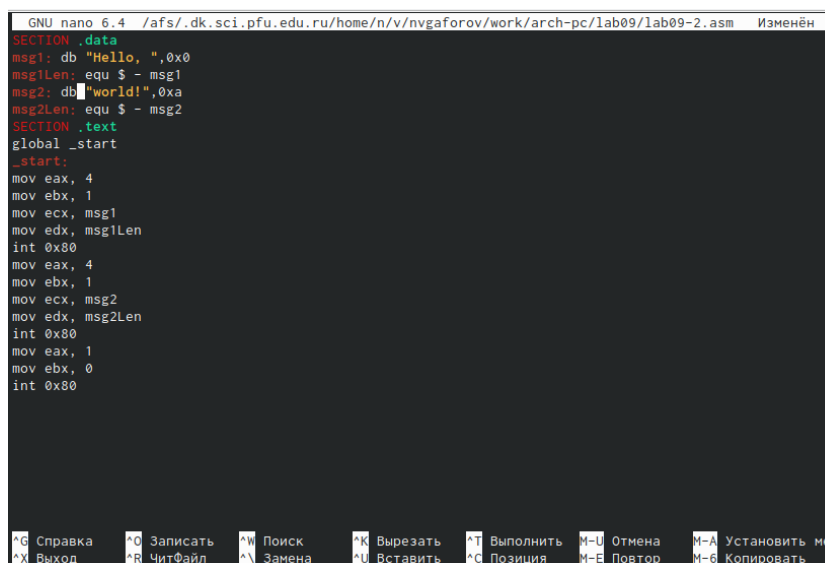
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
f(g(x))=35

```

Рис. 4.6: Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm и поставим программы из Листинга 9.2 (рис. [4.7]).

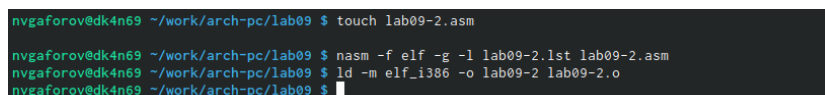


```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-2.asm  Изменён
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80

^C Справка      ^O Записать    ^N Поиск      ^K Вырезать   ^T Выполнить  ^M-U Отмена   ^M-A Установить ме
^X Выход      ^R ЧитФайл    ^M Замена    ^U Вставить   ^C Позиция   ^M-B Повтор   ^M-G Копировать
```

Рис. 4.7: Ввод текста программы из листинга 9.2

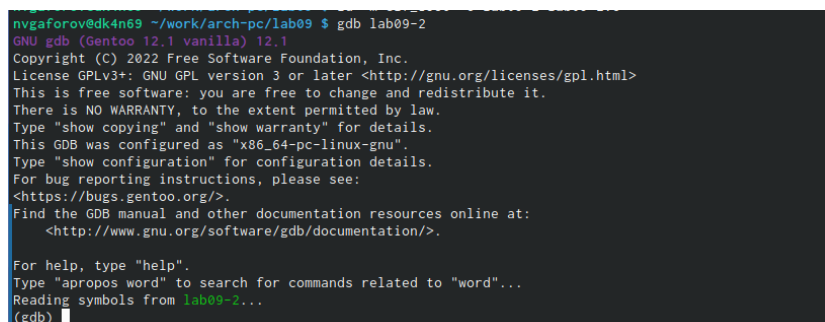
Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. [4.8]).



```
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ touch lab09-2.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $
```

Рис. 4.8: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb (рис. -[4.9]).



```
nvgaforov@dk4n69 ~/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)
```

Рис. 4.9: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `run`. (рис. [4.10]).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 7709) exited normally]
(gdb) □
```

Рис. 4.10: Проверка работы файла с помощью команды `run`

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start` и запускаю её (рис.[4.11]).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) □
```

Рис. 4.11: Установка брейкпоинта и запуск программы

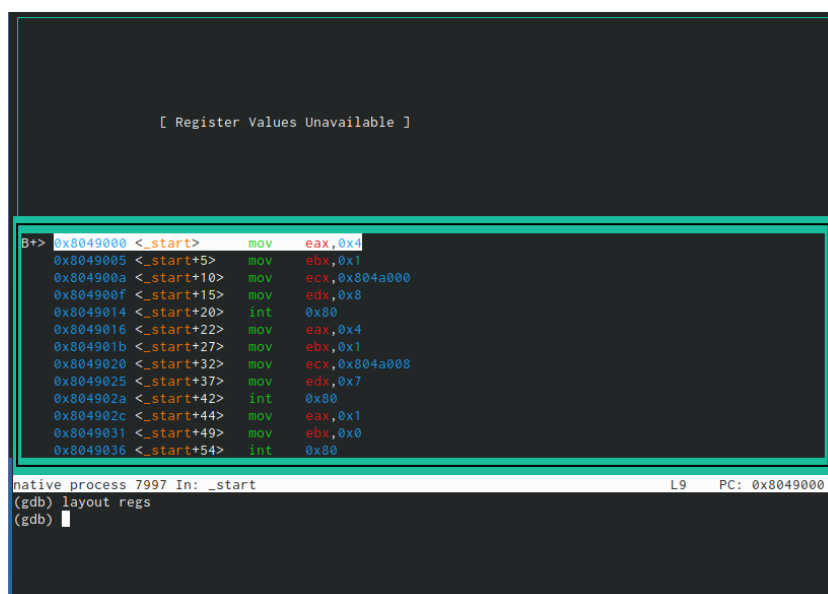
Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. [4.12]).

```
(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) 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. 4.12: Использование команд `disassemble` и `disassembly-flavor intel`

В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется привычный нам синтаксис. (рис. [4.13]).

Включаю режим псевдографики для более удобного анализа программы с помощью команд `layout asm` и `layout regs`. Видим, что в выводе мы получаем неправильный ответ.



```
[ Register Values Unavailable ]

B+> 0x8049000 <_start> mov eax,0x4
0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80

native process 7997 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 4.13: Включение режима псевдографики

4.2.1 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`. Просматриваю информацию о всех установленных точках останова. (рис. [4.14]).

Установка точек останова и просмотр информации о них

Рис. 4.14: Установка точек останова и просмотр информации о них

##Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. [4.15]).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
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 12758 In: _start L9 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.15: До использования команды `stepi`

(рис. [4.16]).


```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
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 12758 In: _start L10 PC: 0x8049005
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.16: После использования команды stepi

(рис. [4.17]).

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804900a 0x804900a <_start+10>
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 12758 In: _start L11 PC: 0x804900a
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804900a 0x804900a <_start+10>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.17: После использования команды stepi

([-рис. 4.18]).

```
--Register group: general--
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804900f 0x804900f <_start+15>
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 12758 In: _start L12 PC: 0x804900f
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804900f 0x804900f <_start+15>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.18: После использования команды stepi

(рис. [4.19]).

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049014 0x8049014 <_start+20>
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 12758 In: _start L13 PC: 0x8049014
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049014 0x8049014 <_start+20>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.19: После использования команды stepi

(рис. [4.20]).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0
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

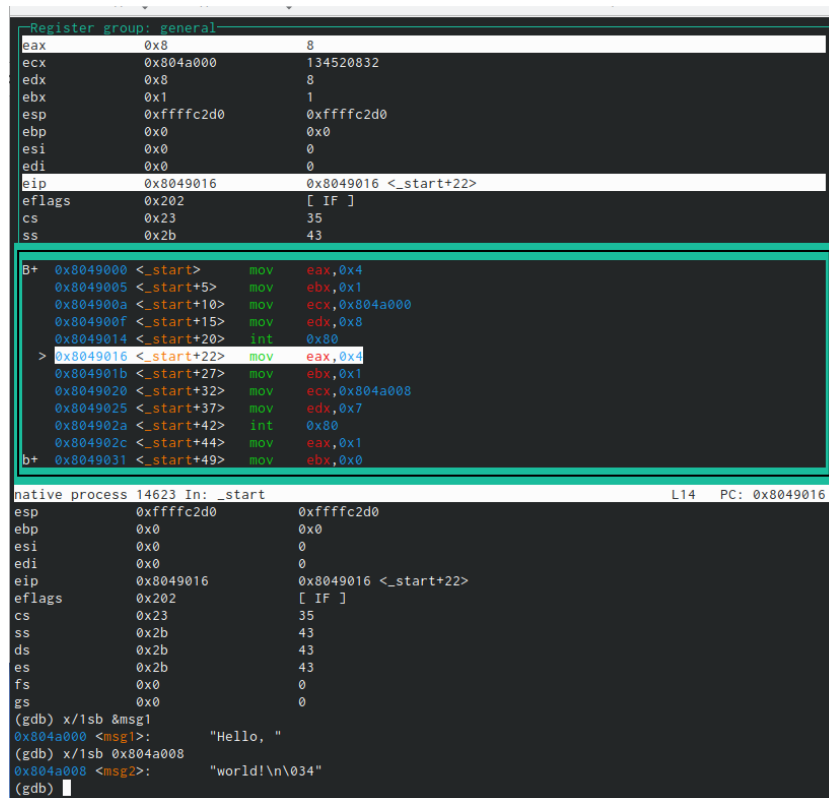
native process 12758 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.20: После использования команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу. (рис. [4.21]).



The screenshot shows a GDB debugger window with two main panes. The top pane displays the 'Register group: general' with the following values:

Register	Value
eax	0x8
ecx	0x804a000
edx	0x8
ebx	0x1
esp	0xffffc2d0
ebp	0x0
esi	0x0
edi	0x0
eip	0x8049016
eflags	0x202
cs	0x23
ss	0x2b

The bottom pane shows memory contents starting at address 0x8049000. The first few lines are highlighted in green:

```
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
0x804902c <_start+44> mov eax,0x1
b+ 0x8049031 <_start+49> mov ebx,0x0
```

The bottom pane also shows the 'native process' information and the current state of the registers, which matches the top pane. At the bottom, the GDB command prompt shows the following commands and output:

```
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)
```

Рис. 4.21: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю пВидим, что в выводе мы получаем неправильный ответ.ервый символ в переменной msg2. (рис. [4.22]).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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
0x804902c <_start+44> mov eax,0x1
b+ 0x8049031 <_start+49> mov ebx,0x0

native process 14623 In: _start L14 PC: 0x8049016
(gdb) set (char)&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hhllo, "
(gdb) set (char)&msg2 = 'b'
(gdb) x/1sb &msg2
0x804a000 <msg2>: "bor d!\n\034"
(gdb)

```

Рис. 4.22: Использование команды set

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра `edx` с помощью команды `print p/F $val./` (рис. [4.23]).

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием.

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.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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
0x804902c <_start+44> mov eax,0x1
b+ 0x8049031 <_start+49> mov ebx,0x0

native process 14623 In: _start L14 PC: 0x8049016
(gdb) set (char)&msg2 = 'b'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "bor d!\n\034"
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) p/t $eax
$3 = 1000
(gdb) p/s $ecx
$4 = 134520832
(gdb) p/x $ecx
$5 = 0x804a000
(gdb)

```

Рис. 4.23: Вывод значения регистра в разных представлениях

Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit. (рис. [4.24]).

```

(gdb) quit
A debugging session is active.

    Inferior 1 [process 17962] will be killed.

Quit anyway? (y or n)

```

Рис. 4.24: Завершение работы GDB

4.2.2 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. [4.25]).

```

nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $

```

Рис. 4.25: Создание файла или копирование файл

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `--args`. (рис. [4.26]).

```

nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент2 '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)

```

Рис. 4.26: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. ([рис. 4.27]).

```

(gdb) b _start
Breakpoint 1 at 0x00490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)

```

Рис. 4.27: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис. [4.28]).

```

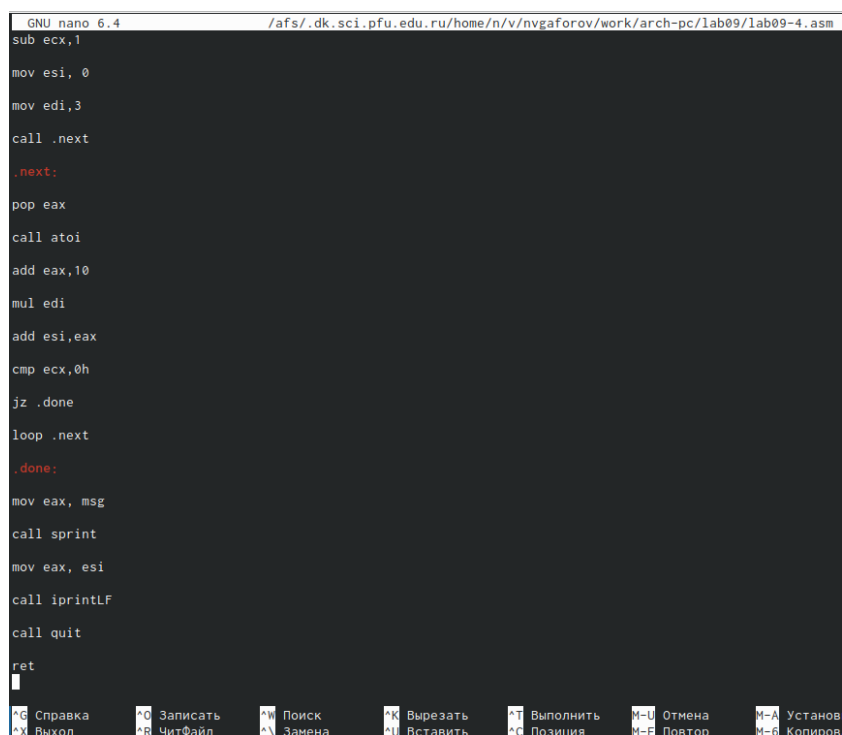
(gdb) x/x $esp
0xffffc280: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffc29: /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc56f: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffc581: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffc592: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffc594: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.28: Просмотр значений, введенных в стек

4.3 Задания для самостоятельной работы

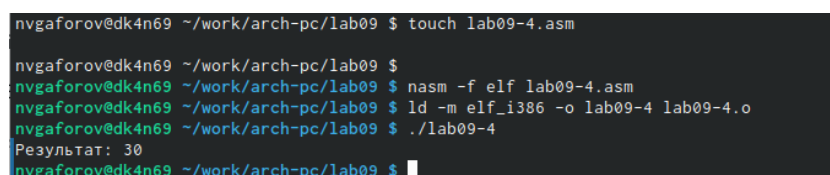
1. Создаем файл lab09-4.asm .Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. [4.29]).



```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-4.asm
sub ecx,1
mov esi, 0
mov edi,3
call .next
.next:
pop eax
call atoi
add eax,10
mul edi
add esi,eax
cmp ecx,0h
jz .done
loop .next
.done:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
ret
```

Рис. 4.29: Написание кода программы

Запускаю код и проверяю, что она работает корректно. (рис. [4.30]).



```
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ touch lab09-4.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ ./lab09-4
Результат: 30
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $
```

Рис. 4.30: Запуск программы и проверка его вывода

Код программы:

%include 'in_out.asm'Видим, что в выводе мы получаем неправильный ответ.


```

SECTION .data
msg db "Результат:",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
mov edi,3
call .next
.next:
pop eax
call atoi
add eax,10
mul edi
add esi,eax
cmp ecx,0h
jz .done
loop .next
.done:
mov eax,msg
call sprint
mov eax,esi
call iprintLF
call quit
ret

```

2. Ввожу в файл task1.asm текст программы из листинга 9.3. (рис. [4.31]).

```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab09/lab09-5.asm
#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 sprintf
mov eax,edi
call iprintLF
call quit
```

Рис. 4.31: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его. (рис. [4.32]).

```
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ touch lab09-5.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 10
nvgaforov@dk4n69 ~/work/arch-pc/lab09 $
```

Рис. 4.32: Создание и запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ.

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 (регистр ebx). Происходит это из-за того, что

стоящая перед `mov ecx,4` инструкция `add ebx,eax` не связана с `mul ecx`, но связана инструкция `mov eax,2` (рис. [4.33]).

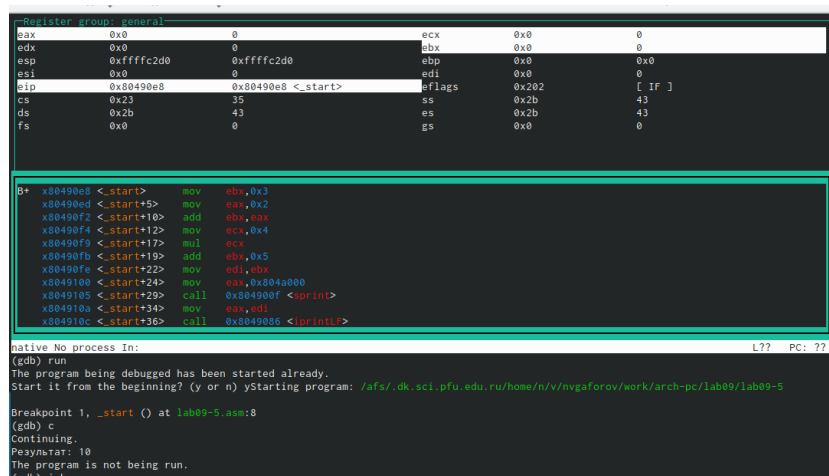


Рис. 4.33: Нахождение причины ошибки

Исправляем ошибку, добавляя после `add ebx,eax` `mov eax,ebx` и заменяя `ebx` на `eax` в инструкциях `add ebx,5` и `mov edi,ebx`. (рис. [4.34]).

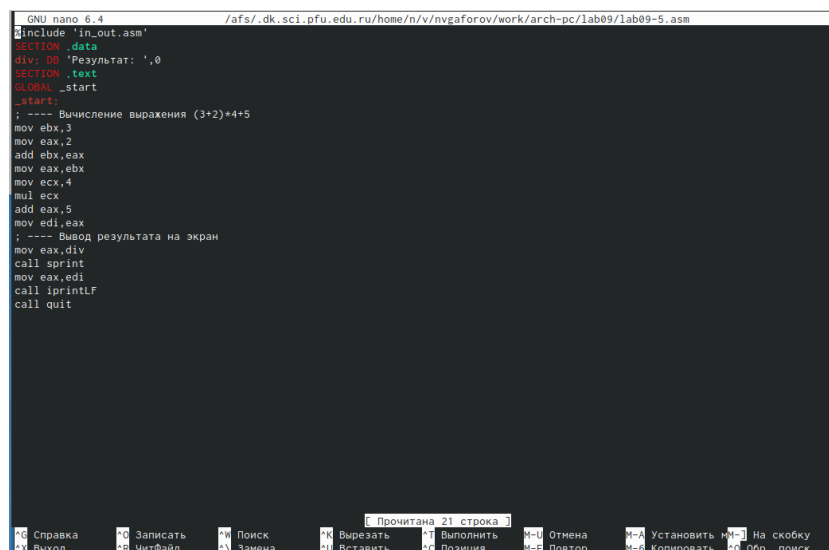


Рис. 4.34: Исправление ошибки

Создаем исполняемый файл и запускаем его. Убеждаемся, что ошибка исправлена (рис. [4.35]).

```
nvgaforov@dk3n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-5.lst lab09-5.asm
nvgaforov@dk3n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
nvgaforov@dk3n65 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 25
nvgaforov@dk3n65 ~/work/arch-pc/lab09 $
```

Рис. 4.35: Ошибка исправлена

код 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 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
```

5 Выводы

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

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-

- е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
18. — 1120 с. — (Классика Computer Science)