

Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Мосолов Александр Денисович

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
2.1	Реализация подпрограмм в NASM	7
2.2	Отладка программ с помощью GDB	9
2.2.1	Добавление точек останова	14
2.2.2	Работа с данными программы в GDB	15
2.2.3	Обработка аргументов командной строки в GDB	17
2.3	Выполнение заданий для самостоятельной работы	19
3	Выводы	24

Список иллюстраций

2.1	Создание каталога для работы	7
2.2	Текст программы lab09-1	8
2.3	Создание исполняемого файла	8
2.4	Текст изменённой программы lab09-1	9
2.5	Создание исполняемого файла обновленной программы	9
2.6	Текст программы lab09-2	10
2.7	Получение исполняемого файла lab09-2	10
2.8	Загрузка исполняемого файла в отладчик GDB	10
2.9	Проверка работы программы в оболочке GDB	11
2.10	Запуск программы с брэйкпоинтом на метке _start	11
2.11	Дисассимилированный код программы	12
2.12	Intel'овский синтаксис	12
2.13	Режим псевдографики №1	13
2.14	Режим псевдографики №2	14
2.15	Информация о точках останова	14
2.16	Установка точки останова	14
2.17	Информация о точках останова	14
2.18	Выполнение 5 инструкций stepi	15
2.19	Информация о регистрах	15
2.20	Изменяем первого символа переменной msg1	16
2.21	Изменяем первого символа переменной msg2	16
2.22	Выводим значение регистра edx	16
2.23	Изменение регистра ebx	17
2.24	Завершение программы	17
2.25	Выход из GDB	17
2.26	Копирование и создание исполняемого файла	18
2.27	Загрузка исполняемого файла в отладчик	18
2.28	Установка точки останова и запуск программы	18
2.29	Адрес вершины стека	18
2.30	Просмотр остальных позиций стека	19
2.31	Текст отредактированной программы	20
2.32	Создание и запуск исполняемого файла lab09-4	20
2.33	Текст программы листинга 9.3	21
2.34	Загрузка файла в отладчик	21
2.35	Запуск программы	21
2.36	Изменение регистров	22
2.37	Текст изменённой программы lab09-5	22

2.38 Создание и запуск исполняемого файла	23
---	----

Список таблиц

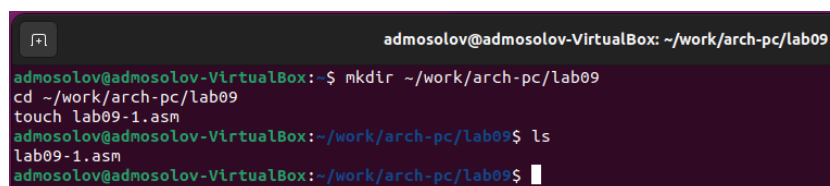
1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

С помощью *mkdir* создаем директорию, в которой будем работать во время выполнения лабораторной работы №9. Переходим в созданный каталог. С помощью *touch* создаем файл *lab09-1.asm*.



```
admosolov@admosolov-VirtualBox: ~/work/arch-pc/lab09
admosolov@admosolov-VirtualBox:~$ mkdir ~/work/arch-pc/lab09
cd ~/work/arch-pc/lab09
touch lab09-1.asm
admosolov@admosolov-VirtualBox:~/work/arch-pc/lab09$ ls
lab09-1.asm
admosolov@admosolov-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 2.1: Создание каталога для работы

Внимательно изучив текст программы из листинга 9.1, вводим его в файл *lab09-1.asm*.

```

admosolov@admosolov-VirtualBox: ~/work/arch-pc/lab09
GNU nano 6.2 /home/admosolov/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:

mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit

    calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret

```

Рис. 2.2: Текст программы lab09-1

Далее создаем исполняемый файл и запускаем его.

```

admosolov@admosolov-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
ld -m elf_i386 -o lab09-1 lab09-1.o
./lab09-1
lab09-1.asm:1: warning: unterminated string [-w+other]
Введите x: 1
2x+7=9
admosolov@admosolov-VirtualBox:~/work/arch-pc/lab09$

```

Рис. 2.3: Создание исполняемого файла

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


```

admosolov@admosolov-VirtualBox: ~/work/arch-pc/lab09
GNU nano 6.2 /home/admosolov/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:

mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit

_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret

```

Рис. 2.4: Текст изменённой программы lab09-1

Далее создаем исполняемый файл и запускаем его.

```

admosolov@admosolov-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
ld -m elf_i386 -o lab09-1 lab09-1.o
./lab09-1
lab09-1.asm:1: warning: unterminated string [-w+other]
Введите x: 1
2x+7=11
admosolov@admosolov-VirtualBox:~/work/arch-pc/lab09$

```

Рис. 2.5: Создание исполняемого файла обновленной программы

2.2 Отладка программ с помощью GDB

Создаем файл *lab09-2.asm* в каталоге с помощью команды *touch lab09-2.asm*.

Открываем файл и заполняем его в соответствии с листингом 9.2.

```

admosolov@admosolov-VirtualBox: ~/work/arch-pc/lab09
GNU nano 6.2 /home/admosolov/work/arch-pc/lab09/lab09-2.asm
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80

```

Рис. 2.6: Текст программы lab09-2

Получим исполняемый файл. Для работы с *GDB* в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом `'-g'`.

```

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

```

Рис. 2.7: Получение исполняемого файла lab09-2

Загрузим исполняемый файл в отладчик *GDB*.

```

admosolov@admosolov-VirtualBox:~/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)

```

Рис. 2.8: Загрузка исполняемого файла в отладчик GDB

Проверим работу программы, запустив ее в оболочке *GDB* с помощью команды

run.

```
(gdb) run
Starting program: /home/admosolov/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 5054) exited normally]
(gdb)
```

Рис. 2.9: Проверка работы программы в оболочке GDB

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

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

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

Рис. 2.10: Запуск программы с брейкпоинтом на метке `_start`

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`.

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
    0x08049005 <+5>:      mov     $0x1,%ebx
    0x0804900a <+10>:     mov     $0x804a000,%ecx
    0x0804900f <+15>:     mov     $0x8,%edx
    0x08049014 <+20>:     int     $0x80
    0x08049016 <+22>:     mov     $0x4,%eax
    0x0804901b <+27>:     mov     $0x1,%ebx
    0x08049020 <+32>:     mov     $0x804a008,%ecx
    0x08049025 <+37>:     mov     $0x7,%edx
    0x0804902a <+42>:     int     $0x80
    0x0804902c <+44>:     mov     $0x1,%eax
    0x08049031 <+49>:     mov     $0x0,%ebx
    0x08049036 <+54>:     int     $0x80
End of assembler dump.
(gdb)

```

Рис. 2.11: Дисассимилированный код программы

Переключимся на отображение команд с *Intel'овским синтаксисом*, введя команду *set disassembly-flavor intel*.

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     mov     eax,0x4
    0x0804901b <+27>:     mov     ebx,0x1
    0x08049020 <+32>:     mov     ecx,0x804a008
    0x08049025 <+37>:     mov     edx,0x7
    0x0804902a <+42>:     int     0x80
    0x0804902c <+44>:     mov     eax,0x1
    0x08049031 <+49>:     mov     ebx,0x0
    0x08049036 <+54>:     int     0x80
End of assembler dump.
(gdb)

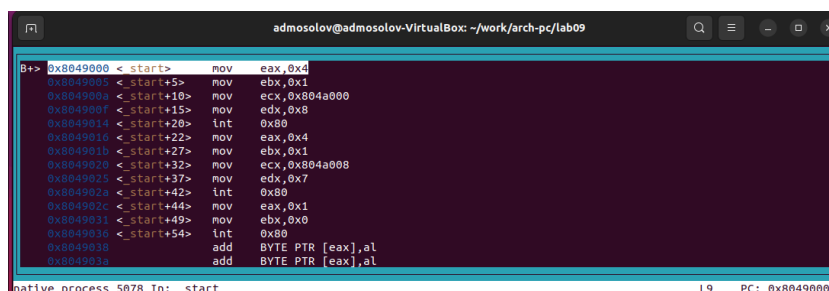
```

Рис. 2.12: Intel'овский синтаксис

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel:

1. Разделители операндов также различаются: в АТТ синтаксисе используются запятые, в то время как в Intel синтаксисе могут применяться запятые или косые черты (/).
2. В АТТ синтаксисе порядок операндов отличается от Intel: исходный операнд указывается первым, а результирующий - вторым, в то время как в Intel синтаксисе порядок обычно прямой, с результирующим операндом первым и исходным операндом вторым.
3. Адреса в АТТ синтаксисе указываются в круглых скобках, в то время как в Intel синтаксисе адреса указываются без скобок.
4. Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как “b” (byte), “w” (word), “l” (long) и “q” (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как “b”, “w”, “d” и “q”.
5. Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом “.Intel”.

Включим режим псевдографики для более удобного анализа программы.



The screenshot shows a debugger window titled 'admosolov@admosolov-VirtualBox: ~/work/arch-pc/lab09'. The assembly view displays the following code in AT&T syntax:

```
0+> 0x8049000 <_start> mov    eax,0x4
0x8049003 <_start+5> mov    ebx,0x1
0x8049006 <_start+10> mov    ecx,0x804a000
0x804900f <_start+15> mov    edx,0x8
0x8049014 <_start+20> int    0x80
0x8049010 <_start+22> mov    eax,0x4
0x804901b <_start+27> mov    ebx,0x1
0x8049026 <_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
0x8049038 add    BYTE PTR [eax],al
0x804903a add    BYTE PTR [eax],al
```

At the bottom of the window, it says 'native process 5078 In: _start' and 'L9 PC: 0x8049000'.

Рис. 2.13: Режим псевдографики №1

```

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
native process 5078 In: _start          L9      PC: 0x8049000

```

Рис. 2.14: Режим псевдографики №2

2.2.1 Добавление точек останова

Проверим установленные точки останова с помощью команды *info breakpoints*.

```

(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. 2.15: Информация о точках останова

Определим адрес предпоследней инструкции (*mov ebx,0x0*) и установим точку останова.

```

(gdb) b *0x8049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab09-2.asm, line 9.
(gdb)

```

Рис. 2.16: Установка точки останова

Посмотрим информацию о всех установленных точках останова.

```

(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint      keep y   0x08049000 lab09-2.asm:9
(gdb)

```

Рис. 2.17: Информация о точках останова

2.2.2 Работа с данными программы в GDB

Выполним 5 инструкций *stepi*.

```
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) si
10     mov ebx, 1
(gdb) si
11     mov ecx, msg1
(gdb) si
12     mov edx, msg1len
(gdb) si
13     int 0x80
(gdb) si
```

Рис. 2.18: Выполнение 5 инструкций *stepi*

Посмотрев содержимое регистров с помощью команды *info registers* (или *i r*), понимаем, что во время выполнения команд менялись регистры: *ebx*, *ecx*, *edx*, *eax*, *eip*.

```
eax            0x8                8
ecx            0x804a000          134520832
edx            0x8                8
ebx            0x1                1
esp            0xffffd260         0xffffd260
ebp            0x0                0
esi            0x0                0
edi            0x0                0
eip            0x8049016          0x8049016 <_start+22>
eflags         0x202             [ IF ]
cs             0x23              35
ss             0x2b              43
ds             0x2b              43
es             0x2b              43
fs             0x0                0
gs             0x0                0
(gdb) 
```

Рис. 2.19: Информация о регистрах

С помощью команды *x/ &* также можно посмотреть содержимое переменной. Просмотрим значение переменной *msg1* по имени и изменим первый символ слова на *h*.

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

Рис. 2.20: Изменяем первого символа переменной *msg1*

Можно смотреть значение переменных не только по имени, но и по адресу в памяти.

Поменяем первый символ слова переменной *msg2* на *W*.

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

Рис. 2.21: Изменяем первого символа переменной *msg2*

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

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

Рис. 2.22: Выводим значение регистра *edx*

Изменим регистр *ebx*.


```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$6 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$7 = 2  
(gdb) █
```

Рис. 2.23: Изменение регистра ebx

Выводятся разные значения, команда без кавычек присваивает регистру вводимое значение.

Завершим выполнение программы с помощью команды *continue* (сокращенно *c*).

```
(gdb) continue  
Continuing.  
hello, World!  
[Inferior 1 (process 5569) exited normal  
(gdb)
```

Рис. 2.24: Завершение программы

Выйдем из *GDB* с помощью команды *quit* (сокращенно *q*).

```
(gdb) quit  
admosolov@admosolov-VirtualBox: ~/work/arch-pc/lab09$ █
```

Рис. 2.25: Выход из GDB

2.2.3 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*. И создадим исполняемый файл.

```

admosolov@admosolov-VirtualBox: /work/arch-pc/lab09$ cp -r /work/arch-pc/lab09/lab08-2.asm -r /work/arch-pc/lab09/lab09-3.asm
admosolov@admosolov-VirtualBox: /work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ld -m elf_i386 -o lab09-3 lab09-3.o
admosolov@admosolov-VirtualBox: /work/arch-pc/lab09$ ls
ln_out.asm lab09-1.asm lab09-2 lab09-2.lst lab09-3 lab09-3.lst
lab09-1 lab09-1.o lab09-2.asm lab09-2.o lab09-3.asm lab09-3.o
admosolov@admosolov-VirtualBox: /work/arch-pc/lab09$

```

Рис. 2.26: Копирование и создание исполняемого файла

Загрузим исполняемый файл в отладчик, указав ключ `-args` и аргументы.

```

admosolov@admosolov-VirtualBox: /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. 2.27: Загрузка исполняемого файла в отладчик

Для начала установим точку останова перед первой инструкцией в программе и запустим ее.

```

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

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx
(gdb)

```

Рис. 2.28: Установка точки останова и запуск программы

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

```

(gdb) x/x $esp
0xfffffd220: 0x00000005
(gdb)

```

Рис. 2.29: Адрес вершины стека

Просмотрим остальные позиции стека.

```
(gdb) x/s *(void**)($esp + 4)
0xffffd3d1:    "/home/admosolov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd3fc:    "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd40e:    "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd41f:    "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd421:    "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.30: Просмотр остальных позиций стека

Шаг изменения адреса равен 4, т.к. в большинстве архитектур процессоров размер слова (или размер указателя) составляет 4 байта. Это означает, что каждый раз, когда мы обращаемся к следующему элементу в стеке, мы увеличиваем адрес на 4. Шаг изменения адреса равен 4 для обеспечения корректного доступа к данным в стеке.

2.3 Выполнение заданий для самостоятельной работы

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

Создаем файл для выполнения задания с помощью команды *touch lab09-4.asm*.

Вставим и отредактируем текст программы из лабораторной работы №8 с добавлением подпрограммы.

```

admosolov@admosolov-VirtualBox: ~/work/arch-pc/lab09
GNU nano 6.2 /home/admosolov/work/arch-pc/lab09/lab09-4.asm
#include 'in_out.asm'
SECTION .data
m db "f(x)=10x-4", 0
msg db "Результат: ",0
SECTION .text
global _start
_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

next:
cmp ecx,0h
jz _end

pop eax
call atoi

call _calcul
add eax, esi
mov esi, eax
loop next

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

_calcul:
mov ebx, 10
mul ebx
sub eax, 4
ret

```

Рис. 2.31: Текст отредактированной программы

Далее создаем исполняемый файл и запускаем его.

```

admosolov@admosolov-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
ld -m elf_i386 -o lab09-4 lab09-4.o
./lab09-4 1 3
Результат: 32
admosolov@admosolov-VirtualBox:~/work/arch-pc/lab09$

```

Рис. 2.32: Создание и запуск исполняемого файла lab09-4

В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \cdot 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB*, анализируя изменения значений регистров, определите ошибку и исправьте ее.

Создаем файл для выполнения задания с помощью команды `touch lab09-5.asm`. Вставим программу из листинга 9.3.

```

admosolov@admosolov-VirtualBox: ~/work/arch-pc/lab09
GNU nano 6.2 /home/admosolov/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 sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 2.33: Текст программы листинга 9.3

Создаем исполняемый файл и запускаем его в отладчике *GDB*. Смотрим на изменение регистров по ходу программы.

```

admosolov@admosolov-VirtualBox:~/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)

```

Рис. 2.34: Загрузка файла в отладчик

Запустим программу с помощью *run*.

```

(gdb) run
Starting program: /home/admosolov/work/arch-pc/lab09/lab09-5
Результат: 10
[Inferior 1 (process 6508) exited normally]
(gdb)

```

Рис. 2.35: Запуск программы

Смотрим изменение регистров.

```

(gdb) run
Starting program: /home/admosolov/work/arch-pc/lab09/lab09-5

Breakpoint 2, 0x080490e8 in _start ()
(gdb) i r
eax                0x0                0
ecx                0x0                0
edx                0x0                0
ebx                0x0                0
esp                0xfffffd260        0xfffffd260
ebp                0x0                0x0
esi                0x0                0
edi                0x0                0
eip                0x080490e8        0x080490e8 <_start>
eflags             0x202             [ IF ]
cs                 0x23             35
ss                 0x2b             43
ds                 0x2b             43
es                 0x2b             43
fs                 0x0                0
gs                 0x0                0
(gdb) s

```

Рис. 2.36: Изменение регистров

Корректируем программу, обнаружив ошибку неправильной записи регистров.

```

GNU nano 6.2 /home/admosolov/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 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. 2.37: Текст изменённой программы lab09-5

Далее создаем исполняемый файл и запускаем его, проверяем правильность выполненной программы.

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

Рис. 2.38: Создание и запуск исполняемого файла

3 Выводы

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