

Отчёта по лабораторной работе 10

Понятие подпрограммы. Отладчик GDB.

Игнатенкова В.Н

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	28
	Список литературы	29

Список иллюстраций

4.1	Файл lab10-1.asm	9
4.2	Работа программы lab10-1.asm	10
4.3	Файл lab10-1.asm	11
4.4	Работа программы lab10-1.asm	12
4.5	Файл lab10-2.asm	13
4.6	Работа программы lab10-2.asm в отладчике	14
4.7	дисассимилированный код	15
4.8	дисассимилированный код в режиме интел	16
4.9	точка остановки	17
4.10	изменение регистров	18
4.11	изменение регистров	18
4.12	изменение значения переменной	19
4.13	вывод значения регистра	20
4.14	вывод значения регистра	21
4.15	вывод значения регистра	22
4.16	Файл lab10-4.asm	23
4.17	Работа программы lab10-4.asm	24
4.18	код с ошибкой	25
4.19	отладка	26
4.20	код исправлен	27
4.21	проверка работы	27

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 Задание

1. Изучите примеры реализации подпрограмм
2. Изучите работу с отладчиком GDB
3. Выполните самостоятельное задание
4. Загрузите файлы на GitHub.

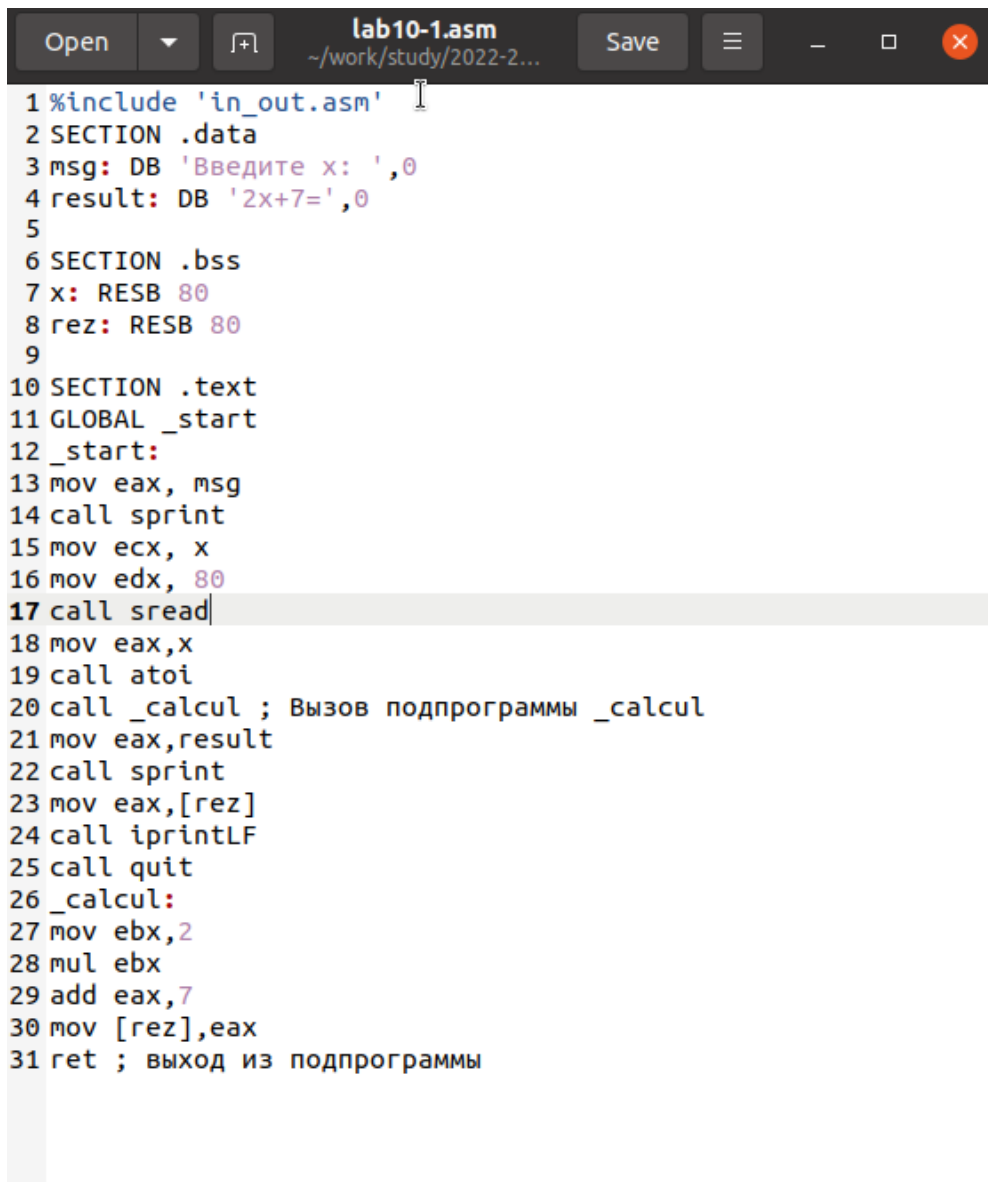
3 Теоретическое введение

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.

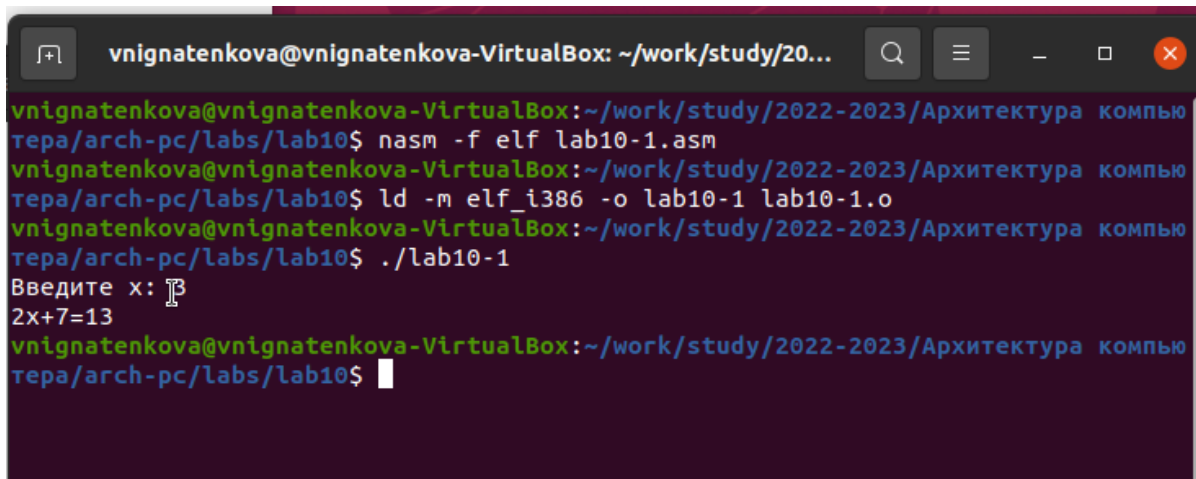
4 Выполнение лабораторной работы

1. Создайте каталог для выполнения лабораторной работы № 10, перейдите в него и создайте файл lab10-1.asm:
2. В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x+7$ с помощью подпрограммы calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 10.1). (рис. 4.1, 4.2)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5
6 SECTION .bss
7 x: RESB 80
8 rez: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprint
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax, x
19 call atoi
20 call _calcul ; Вызов подпрограммы _calcul
21 mov eax, result
22 call sprint
23 mov eax, [rez]
24 call iprintLF
25 call quit
26 _calcul:
27 mov ebx, 2
28 mul ebx
29 add eax, 7
30 mov [rez], eax
31 ret ; выход из подпрограммы
```

Рис. 4.1: Файл lab10-1.asm



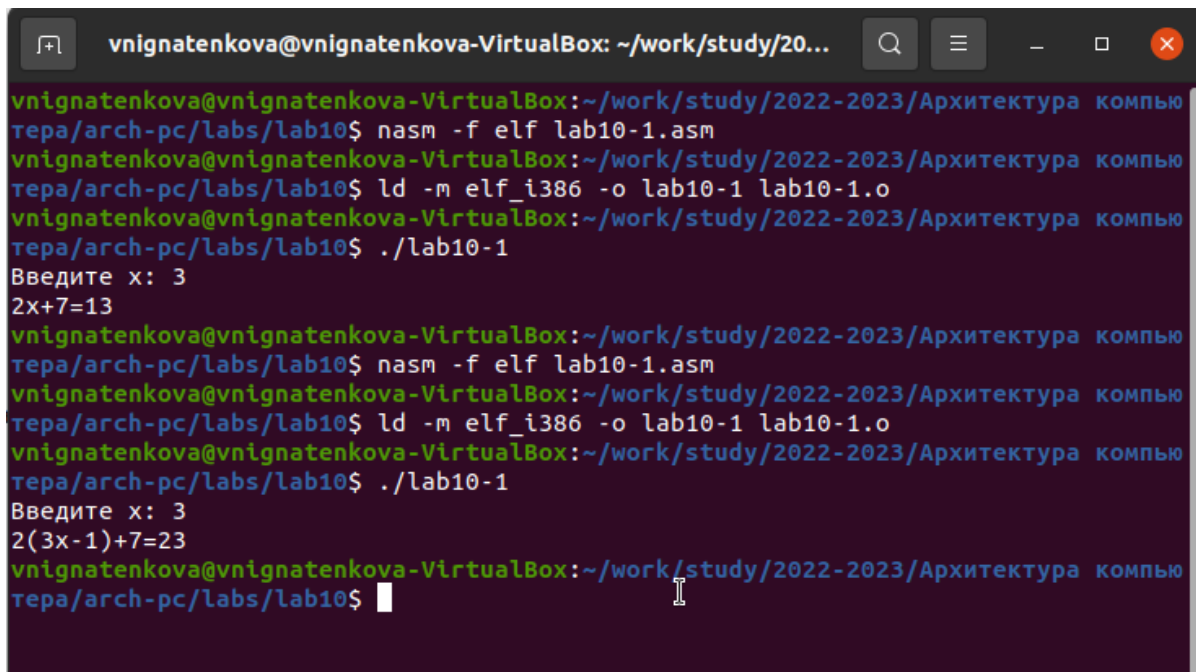
```
vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/20...
repa/arch-pc/labs/lab10$ nasm -f elf lab10-1.asm
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
repa/arch-pc/labs/lab10$ ld -m elf_i386 -o lab10-1 lab10-1.o
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
repa/arch-pc/labs/lab10$ ./lab10-1
Введите x: 13
2x+7=13
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
repa/arch-pc/labs/lab10$
```

Рис. 4.2: Работа программы lab10-1.asm

3. Измените 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. 4.3, 4.4)

```
Open  lab10-1.asm  Save  ~/work/study/2022-2...
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5
6 SECTION .bss
7 x: RESB 80
8 rez: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprint
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax, x
19 call atoi
20 call _calcul ; Вызов подпрограммы _calcul
21 mov eax, result
22 call sprint
23 mov eax, [rez]
24 call iprintLF
25 call quit
26
27 _calcul:
28 call _subcalcul
29 mov ebx, 2
30 mul ebx
31 add eax, 7
32 mov [rez], eax
33 ret ; выход из подпрограммы
34
35 _subcalcul:
36 mov ebx, 3
37 mul ebx
38 sub eax, 1
39 ret
```

Рис. 4.3: Файл lab10-1.asm



```
vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/20...
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
тера/arch-pc/labs/lab10$ nasm -f elf lab10-1.asm
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
тера/arch-pc/labs/lab10$ ld -m elf_i386 -o lab10-1 lab10-1.o
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
тера/arch-pc/labs/lab10$ ./lab10-1
Введите x: 3
2x+7=13
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
тера/arch-pc/labs/lab10$ nasm -f elf lab10-1.asm
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
тера/arch-pc/labs/lab10$ ld -m elf_i386 -o lab10-1 lab10-1.o
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
тера/arch-pc/labs/lab10$ ./lab10-1
Введите x: 3
2(3x-1)+7=23
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компью
тера/arch-pc/labs/lab10$
```

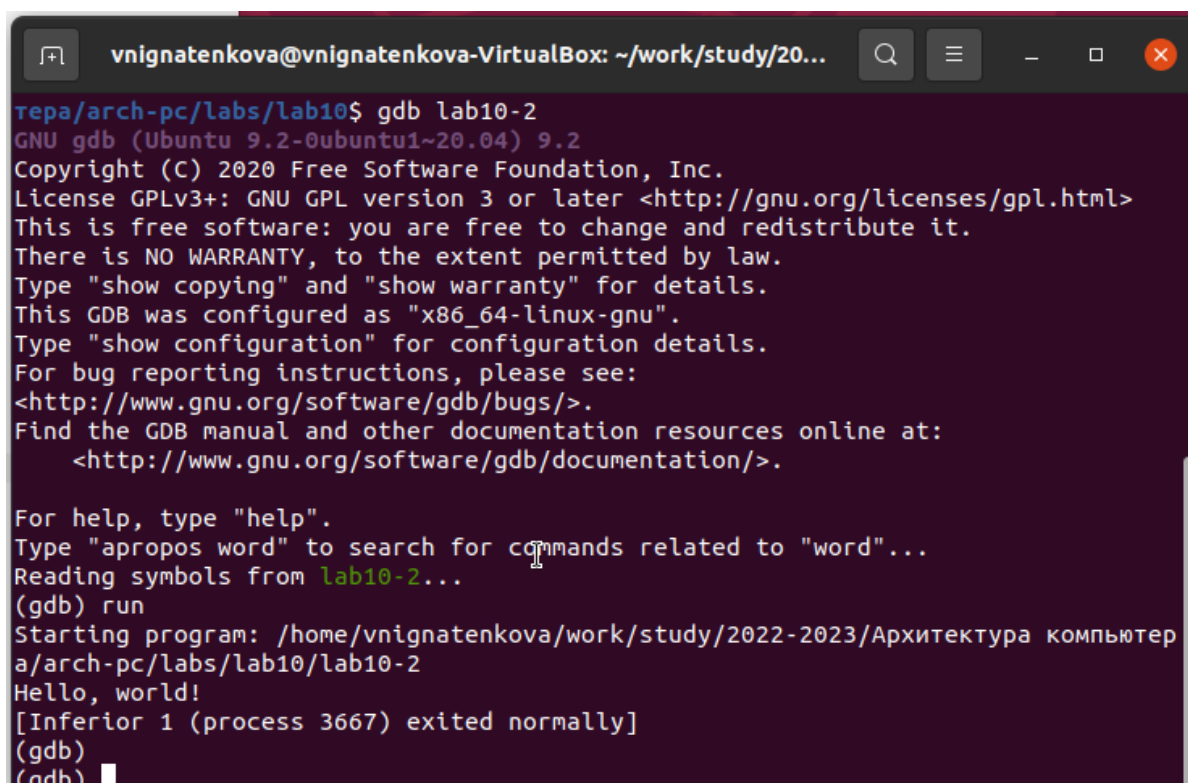
Рис. 4.4: Работа программы lab10-1.asm

4. Создайте файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!); (рис. 4.5)

```
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6
7 SECTION .text
8 global _start
9
10 _start:
11 mov eax, 4
12 mov ebx, 1
13 mov ecx, msg1
14 mov edx, msg1Len
15 int 0x80
16 mov eax, 4
17 mov ebx, 1
18 mov ecx, msg2
19 mov edx, msg2Len
20 int 0x80
21 mov eax, 1
22 mov ebx, 0
23 int 0x80
```

Рис. 4.5: Файл lab10-2.asm

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Загрузите исполняемый файл в отладчик gdb: Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r):(рис. 4.6)

A screenshot of a terminal window titled 'vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/20...'. The terminal shows the execution of 'gdb lab10-2'. It displays the GNU GDB version 9.2, copyright information, and license details. The user enters 'run', and the program starts, printing 'Hello, world!'. The terminal then shows '[Inferior 1 (process 3667) exited normally]' and returns to the '(gdb)' prompt.

```
repa/arch-pc/labs/lab10$ gdb lab10-2
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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:
<http://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 lab10-2...
(gdb) run
Starting program: /home/vnignatenkova/work/study/2022-2023/Архитектура компьютер
a/arch-pc/labs/lab10/lab10-2
Hello, world!
[Inferior 1 (process 3667) exited normally]
(gdb)
(gdb)
```

Рис. 4.6: Работа программы lab10-2.asm в отладчике

Для более подробного анализа программы установите брейкпоинт на метку start, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дисассимилированный код программы (рис. 4.7, 4.8)

```
vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/20...
(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb) run
Starting program: /home/vnignatenkova/work/study/2022-2023/Архитек
a/arch-pc/labs/lab10/lab10-2

Breakpoint 1, 0x08049000 in _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. 4.7: дисассимилированный код

```
vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/20...
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.
(gdb) █
```

Рис. 4.8: дисассимилированный код в режиме интел

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверьте это с помощью команды `info breakpoints` (кратко `i b`) Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку.(рис. 4.9)

The screenshot shows a GDB debugger window with the following content:

```
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd0e0 0xffffd0e0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  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

native process 3671 In: _start L?? PC: 0x8049000
(gdb)
(gdb) █
```

Рис. 4.9: точка остановки

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. (рис. 4.11 4.12)

```

vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/2022-2023/Архитектура компьютера/arch-...
Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd0e0 0xffffd0e0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000 <_start>  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

native process 3671 In: _start L?? PC: 0x8049000
ebx      0x0      0
esp      0xffffd0e0 0xffffd0e0
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--
fs       0x0      0
gs       0x0      0
(gdb)

```

Рис. 4.10: изменение регистров

```

vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/2022-2023/Архитектура компьютера/arch-...
Register group: general
eax      0x4      4      ecx      0x804a008 134520840
edx      0x7      7      ebx      0x1      1
esp      0xffffd0e0 0xffffd0e0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x804902a 0x804902a <_start+42>  eflags   0x202    [ IF ]
cs       0x23     35     ss       0x2b     43

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 3671 In: _start L?? PC: 0x804902a
(gdb)
(gdb) si 0x804900f in _start ()
(gdb) si
0x08049014 in _start ()
(gdb) si
0x08049016 in _start ()
(gdb) si
0x0804901b in _start ()
(gdb) si
0x08049020 in _start ()
(gdb) si
0x08049025 in _start ()
(gdb) si
0x0804902a in _start ()
(gdb)

```

Рис. 4.11: изменение регистров

Посмотрите значение переменной msg1 по имени Посмотрите значение пе-

переменной msg2 по адресу. Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. Измените первый символ переменной msg1. Замените любой символ во второй переменной msg2. (рис. 4.12)

```
native process 3671 In: _start
(gdb) si
0x08049025 in _start ()
(gdb) si
0x0804902a in _start ()
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>:      "Hello, "
(gdb)
(gdb) x/1sb 0x804a0000x804a008 <msg2>:    "world!\n"
(gdb)
(gdb)
(gdb) x/1sb &msg10x804a000 <msg1>:      "hello, "
(gdb)
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "Lorld!\n"
(gdb) █
```

Рис. 4.12: изменение значения переменной

Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью команды set измените значение регистра ebx:(рис. 4.13)

```
native process 3671 In: start
(gdb)
(gdb) p/s $eax$1 = 4
(gdb)
(gdb) p/t $eax$2 = 100
(gdb)
(gdb) p/s $ecx$3 = 134520840
(gdb)
(gdb) p/x $ecx$4 = 0x804a008
(gdb)
(gdb) p/s $edx$5 = 7
(gdb)
(gdb) p/t $edx$6 = 111
(gdb) p/x $edx
$7 = 0x7
(gdb) 
```

Рис. 4.13: вывод значения регистра

С помощью команды set измените значение регистра ebx:(рис. 4.14)

```
native process 3671 In: start
(gdb) p/x $ecx$4 = 0x804a008
(gdb)
(gdb) p/s $edx$5 = 7
(gdb)
(gdb) p/t $edx$6 = 111
(gdb) p/x $edx
$7 = 0x7
(gdb)
(gdb)
(gdb) p/s $ebx$8 = 50
(gdb)
(gdb)
(gdb) p/s $ebx$9 = 2
$10 = 2
(gdb)
```

Рис. 4.14: вывод значения регистра

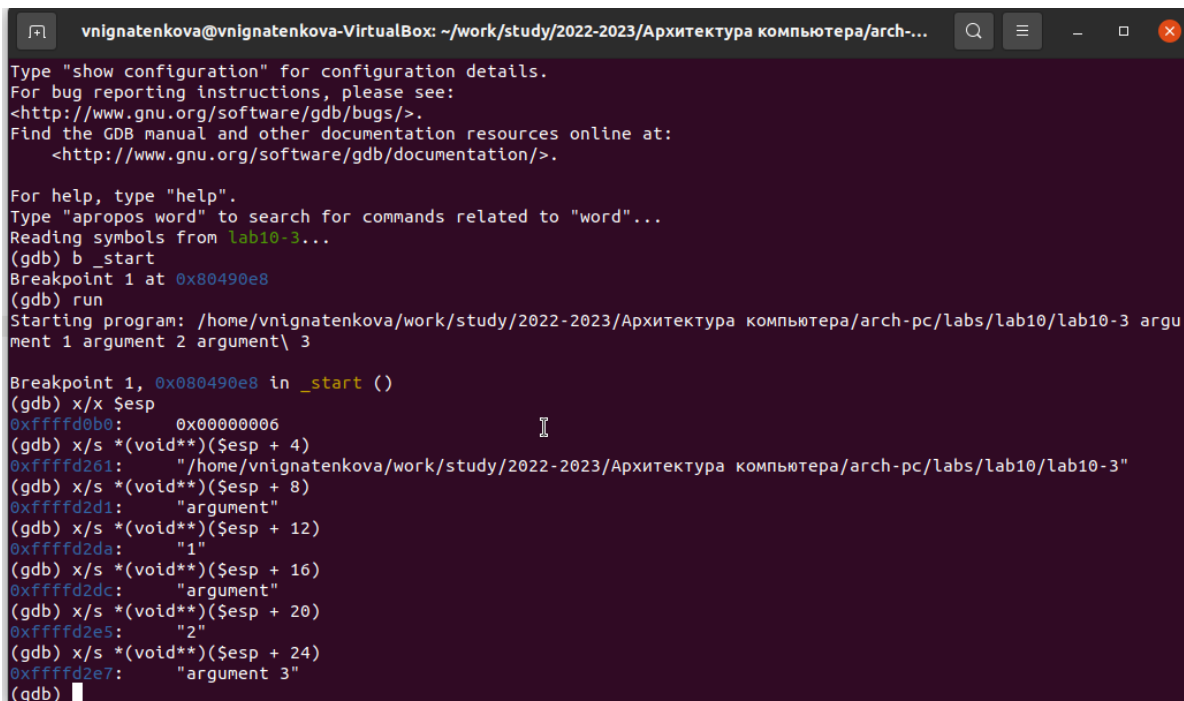
5. Скопируйте файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки. Создайте исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы

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

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): Как видно, число аргументов равно 5 – это имя программы lab10-3

и непосредственно аргументы: аргумент1, аргумент, 2 и 'аргумент 3'.

Посмотрите остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. (рис. 4.15)

The image shows a GDB terminal window with a dark purple background. The window title is 'vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/2022-2023/Архитектура компьютера/arch-...'. The terminal text shows the following sequence of commands and output:
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

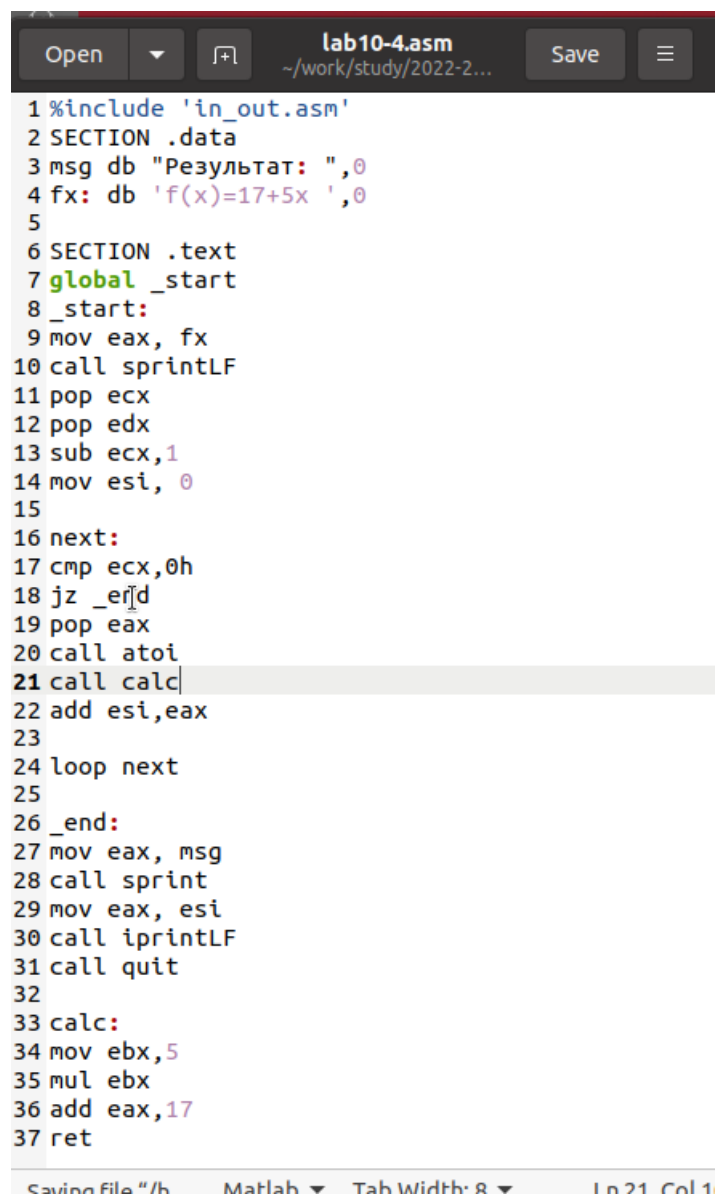
For help, type "help".
Type "apropos word" to search for commands related to "word"..
Reading symbols from lab10-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) run
Starting program: /home/vnignatenkova/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10/lab10-3 argument 1 argument 2 argument\ 3

Breakpoint 1, 0x80490e8 in _start ()
(gdb) x/x \$esp
0xffffd0b0: 0x00000006
(gdb) x/s *(void**)(esp + 4)
0xffffd261: "/home/vnignatenkova/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2d1: "argument"
(gdb) x/s *(void**)(esp + 12)
0xffffd2da: "1"
(gdb) x/s *(void**)(esp + 16)
0xffffd2dc: "argument"
(gdb) x/s *(void**)(esp + 20)
0xffffd2e5: "2"
(gdb) x/s *(void**)(esp + 24)
0xffffd2e7: "argument 3"
(gdb)
The output shows the stack frame at the start of the program. The stack pointer \$esp is at 0xffffd0b0. The stack grows downwards. At offset 4, the program path is stored. At offset 8, the first argument 'argument' is stored. At offset 12, the string '1' is stored. At offset 16, the string 'argument' is stored. At offset 20, the string '2' is stored. At offset 24, the string 'argument 3' is stored.

Рис. 4.15: вывод значения регистра

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] - шаг равен размеру переменной - 4 байтам.

6. Преобразуйте программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. 4.16 4.17)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)=17+5x ',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 call calc
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprintf
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 calc:
34 mov ebx,5
35 mul ebx
36 add eax,17
37 ret
```

Saving file "/h Matlab Tab Width: 8 Ln 21 Col 1

Рис. 4.16: Файл lab10-4.asm

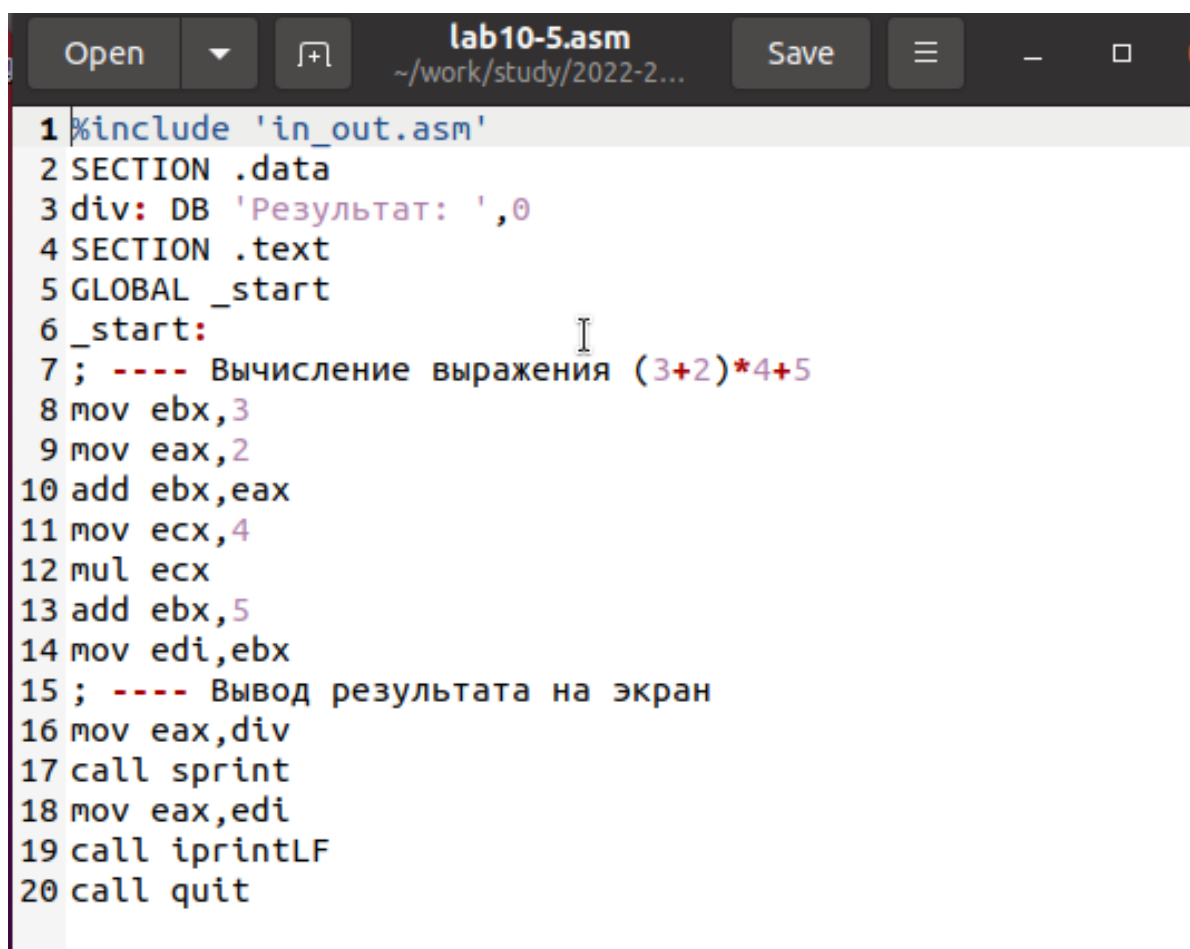
```

vsnignatenkova@vsnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$
vsnignatenkova@vsnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$
vsnignatenkova@vsnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$ nasm -f
elf lab10-4.asm
vsnignatenkova@vsnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$ ld -m el
f_i386 -o lab10-4 lab10-4.o
vsnignatenkova@vsnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$
vsnignatenkova@vsnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$ ./lab10-
4 2 3 4 5
f(x)=17+5x
Результат: 138
vsnignatenkova@vsnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$ ./lab10-
4 2 3 4 5 3 6 5 8 7
f(x)=17+5x
Результат: 368
vsnignatenkova@vsnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$

```

Рис. 4.17: Работа программы lab10-4.asm

7. В листинге 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.(рис. 4.18 4.19 4.20 4.21)



The screenshot shows a code editor window titled 'lab10-5.asm' with a file path '~/.work/study/2022-2...'. The editor contains 20 lines of assembly code. Line 7 contains a comment in Russian: '---- Вычисление выражения (3+2)*4+5'. The code calculates the value of the expression (3+2)*4+5 and prints it. The error is a missing semicolon at the end of line 7, which is highlighted by a cursor.

```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 4.18: код с ошибкой

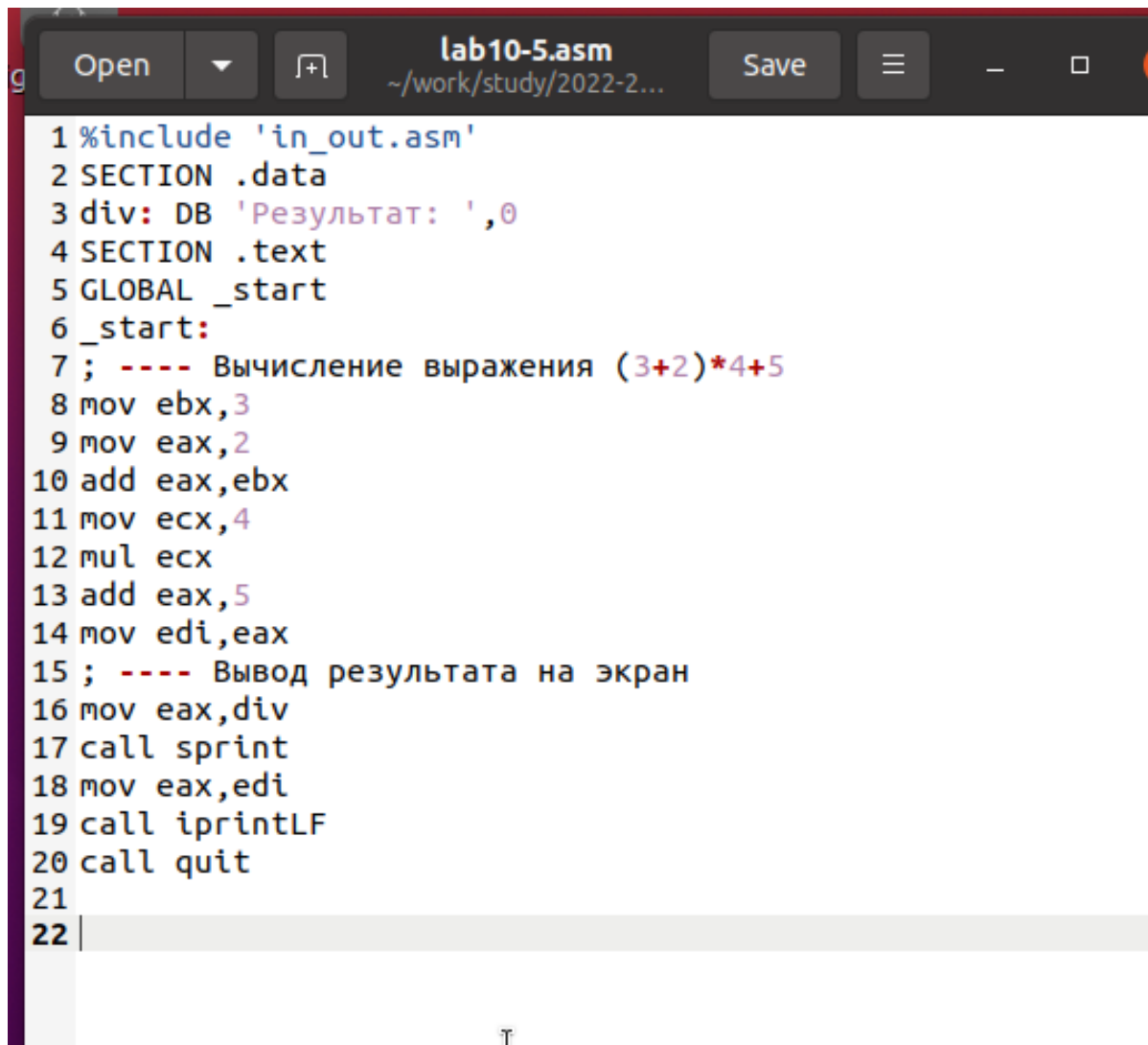
```
vnignatenkova@vnignatenkova-VirtualBox: ~/work/study/2022-2023/Архитектура компьютера/arch-...
Register group: general
eax 0x2 2 ecx 0x0 0
edx 0x0 0 ebx 0x3 3
esp 0xffffd0e0 0xffffd0e0 ebp 0x0 0x0
esi 0x0 0 edi 0x0 0
eip 0x80490f2 0x80490f2 <_start+10> eflags 0x202 [ IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+ 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
>0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>

native process 3712 In: _start L?? PC: 0x80490f2
(gdb)
(gdb) runStarting program: /home/vnignatenkova/work/study/2022-2023/ /arch-pc/labs/lab10/lab
10-5
Breakpoint 1, 0x080490e8 in _start ()
(gdb)
(gdb) si0x080490ed in _start ()
(gdb) si
0x080490f2 in _start ()
(gdb)
```

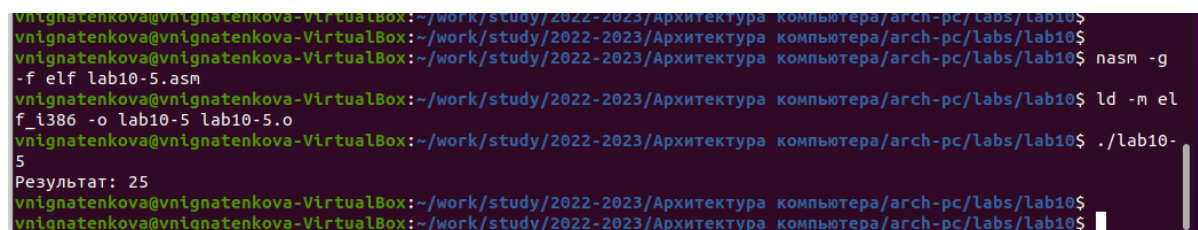
Рис. 4.19: отладка

Отметим, что перепутан порядок аргументов у инструкции add и что по окончании работы в edi отправляется ebx вместо eax



```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21
22 |
```

Рис. 4.20: код исправлен



```
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$ nasm -g
-f elf lab10-5.asm
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$ ld -m elf
f_i386 -o lab10-5 lab10-5.o
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$ ./lab10-
5
Результат: 25
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$
vnignatenkova@vnignatenkova-VirtualBox:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab10$
```

Рис. 4.21: проверка работы

5 Выводы

Освоили работу с подпрограммами и отладчиком.

Список литературы

1. Расширенный ассемблер: NASM
2. MASM, TASM, FASM, NASM под Windows и Linux