

Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Альсалем Шакер

1 Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. ??).

```
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ mkdir ~/work/arch-pc/lab09
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab04$ cd ~/work/arch-pc/lab09
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ touch lab09-1.asm
```

Создаем каталог с помощью команды `mkdir` и файл с помощью команды `touch`

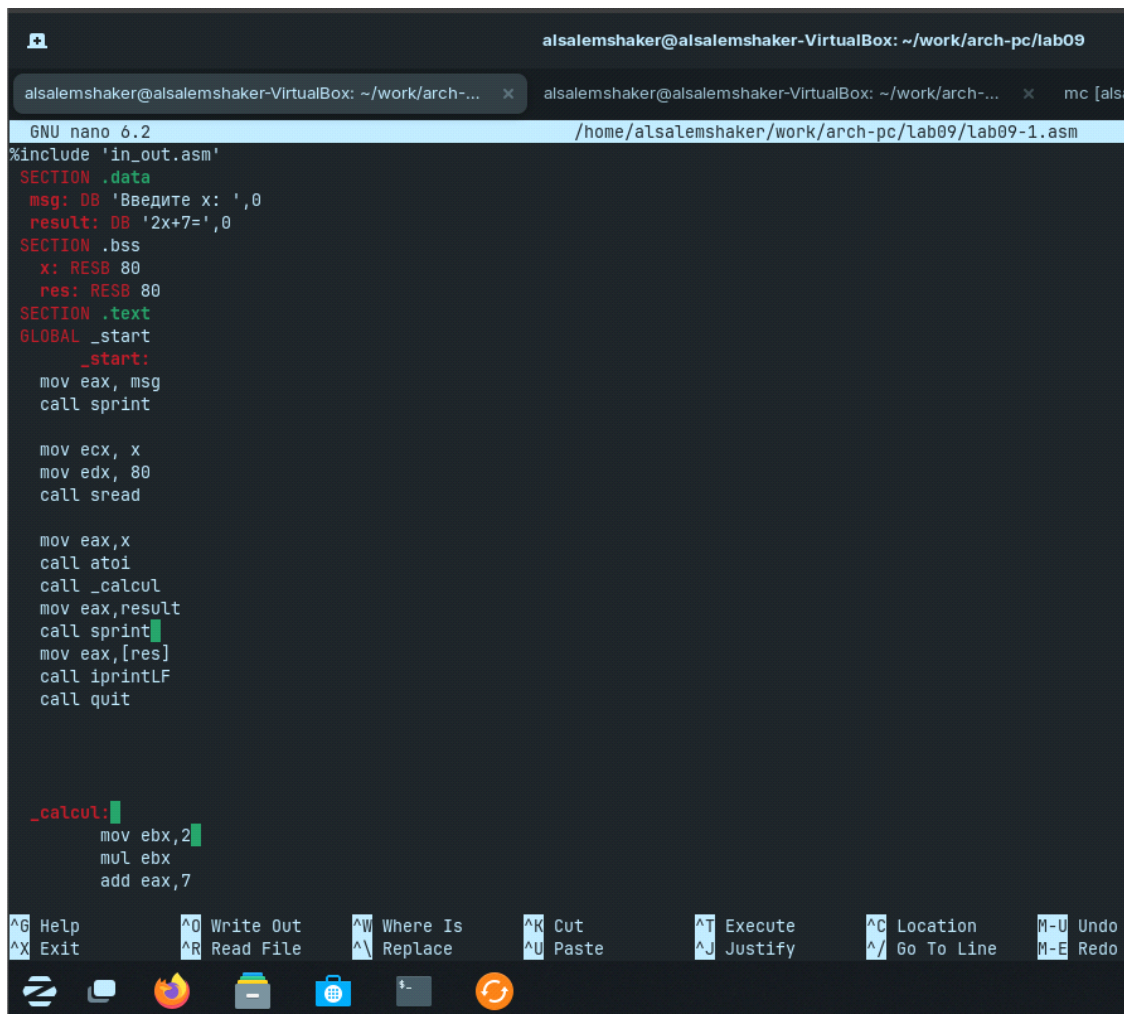
Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. ??).

Заполняем файл

Создаем исполняемый файл и запускаем его (рис. ??).

Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. ??).



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

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   ^M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/_ Go To Line ^M-E Redo
```

Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. ??).

```
alsalemsaker@alsalemsaker-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
alsalemsaker@alsalemsaker-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
ld: cannot find lab09-1.o: No such file or directory
alsalemsaker@alsalemsaker-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
alsalemsaker@alsalemsaker-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 1
2x+7=9
```

Запускаем файл и смотрим на его работу

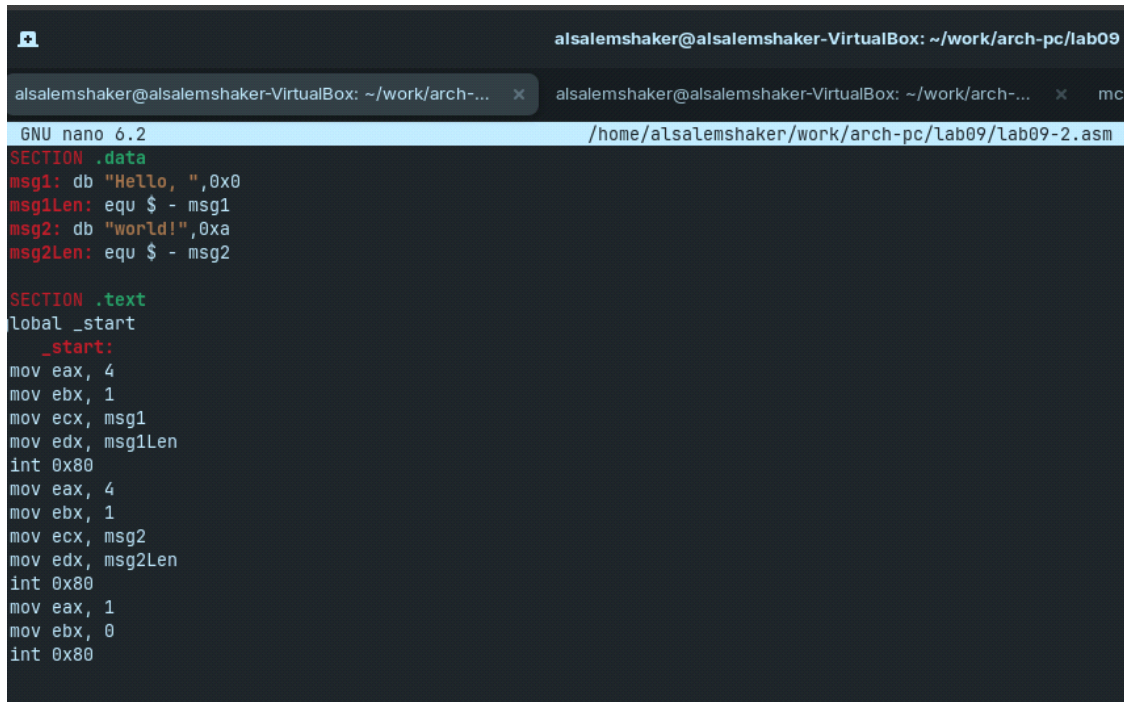
2.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. ??).

```
alsalemsaker@alsalemsaker-VirtualBox:~/work/arch-pc/lab09$ touch ~/work/arch-pc/lab09/lab09-2.asm
alsalemsaker@alsalemsaker-VirtualBox:~/work/arch-pc/lab09$ mc
```

Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. ??).

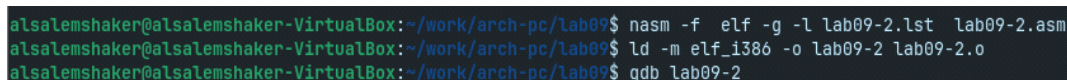


```
alsalemsaker@alsalemsaker-VirtualBox: ~/work/arch-pc/lab09
alsalemsaker@alsalemsaker-VirtualBox: ~/work/arch-... x alsalemsaker@alsalemsaker-VirtualBox: ~/work/arch-... x mc
GNU nano 6.2 /home/alsalemsaker/work/arch-pc/lab09/lab09-2.asm
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2

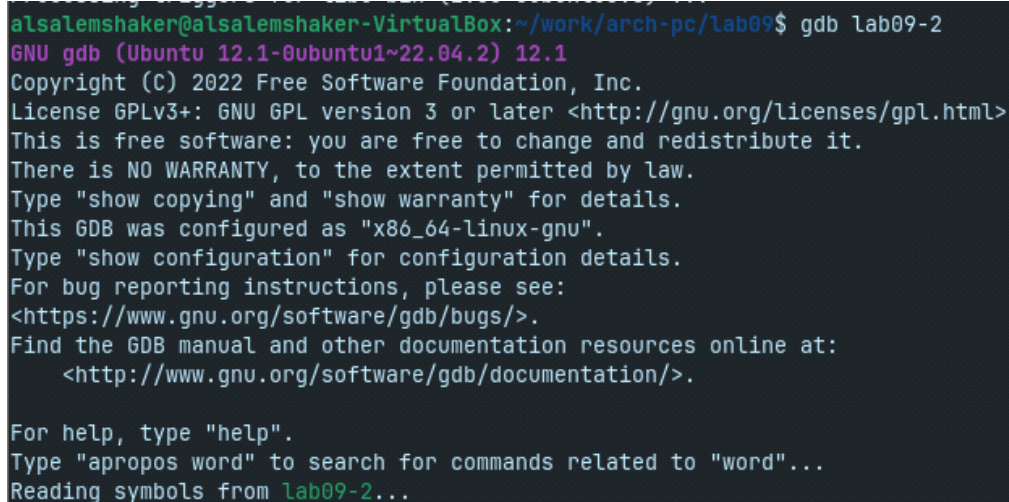
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. ??).



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



```
alsalemsaker@alsalemsaker-VirtualBox:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 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...
```

Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. ??).

```
(gdb) run
Starting program: /home/a/salemsaker/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 37902) exited normally]
```

Запускаем программу командой run

Устанавливаем брейкпоинт на метку `_start` и запускаем программу (рис. ??).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 10.
(gdb) run
Starting program: /home/a/salemsaker/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:10
10      mov eax, 4
```

Запускаем программу с брейкпоинтом

Смотрим дисассемблированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. ??).

```
(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.
```

Смотрим дисассемблированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. ??).

```

(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.

```

Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel:

1.Порядок операндов: В АТТ синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2.Разделители: В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3.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".

4.Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом "\$". В Intel синтаксисе операнды с позитивными значениями могут быть указаны без символа "\$".

5.Обозначение адресов: В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6.Обозначение регистров: В АТТ синтаксисе обозначение регистра начинается с символа "%". В Intel синтаксисе обозначение регистра может начинаться с символа "R" или "E" (например, "%eax" или "RAX").

Включаем режим pseudocode-графики (рис. ??).

```
alsalemsbaker@alsalemsbaker-VirtualBox: ~/work/arch-pc/lab09
alsalemsbaker@alsalemsbaker-VirtualBo... x alsalemsbaker@alsalemsbaker-VirtualBo... x alsalemsbaker@alsalemsbaker-VirtualBo...

[ Register Values Unavailable ]

-lab09-2.asm
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 _start:
10 mov eax, 4
11 mov ebx, 1

exec No process In:
warning: Source file is more recent than executable.
(gdb) layout regs
(gdb)
```

Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. ??).

```
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
0x8049036 <_start+54> int 0x80

native process 6117 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) □
```

Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. ??).

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y   0x08049000 lab09-2.asm:10
          breakpoint already hit 1 time
2        breakpoint keep y   0x08049031 lab09-2.asm:21
```

Смотрим информацию

Выполняем 5 инструкций командой si (рис. ??).

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0c0 0xffffd0c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35

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

native process 6117 In: _start L14 PC: 0x8049016
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(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   0x08049031 lab09-2.asm:20
```

Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip.

Смотрим значение переменной msg1 по имени (рис. ??).

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

Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. ??).

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

Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. ??).

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

Меняем символ

Изменим первый символ переменной msg2 (рис. ??).

```
(gdb) x/1sb & msg1
0x804a000 <msg1>: "hlllo, "
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=''
A character constant must contain at least one character.
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb & msg2
0x804a008 <msg2>: "Lor d!\n\034"
```

Меняем символ

```
(gdb) p/s $eax
$2 = 4
(gdb) p/t $eax
$3 = 100
(gdb) p/s $ecx
$4 = 0
(gdb) p/x $ecx
$5 = 0x0
```

Смотрим значение регистра edx в разных форматах (рис. ??).

Смотрим значение регистра

Изменяем регистр ebx (рис. ??).


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

Изменяем регистр командой set

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

Прописываем команды для завершения программы и выхода из GDB (рис. ??).

```
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) █
```

Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. ??).

```
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab08-2.asm ~/work/arch-pc/lab08/lab09-3.asm
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$
```

Копируем файл

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

```
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
(gdb)
```

Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. ??).

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

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx
(gdb) x/x $esp
0xffffd0c0: 0x00000005
```

Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. ??).

```
(gdb) x/s *(void**)( $esp + 4)
0xffffd2a1: "/home/alsalemsaker/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd2d0: "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd2e2: "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xffffd2f3: "2"
(gdb) x/s *(void**)( $esp + 20)
0xffffd2f5: "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

###Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. ??).

```
cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
```

Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. ??).

```

include 'in_out.asm'
SECTION .data
msg_func db "Функция f(x) = 12x -
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
    mov eax, msg_func
    call sprintf

    pop ecx
    pop edx
    sub ecx, 1
    mov esi, 0

next:
    cmp ecx, 0h
    jz _end
    pop eax
    call atoi

    call _calculate_fx

    add esi, eax
    loop next

_end:
    mov eax, msg_result
    call sprintf

    mov eax, esi

```

Изменяем файл

Создаем исполняемый файл и запускаем его (рис. ??).

```

alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ ./lab09-4
Функция f(x) = 12x - 7
Результат:

```

Проверяем работу программы

###Задание 2

Создаем новый файл в директории (рис. ??).

```

alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ touch lab09-5.asm
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ mc

```

Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. ??).

```

#include 'in_out.asm'

SECTION .data
div: DB 'Результат: ', 0

SECTION .text
GLOBAL _start

_start:
; ---- Calculate (3 + 2)
mov ebx, 3
mov eax, 2
add ebx, eax

mov ecx, 4
mov eax, ebx
mul ecx

add eax, 5

mov edi, eax

; ---- Print the result
mov eax, div
call sprint

mov eax, edi
call iprintLF

; ---- Exit the program
call quit

```

Изменяем файл

Создаем исполняемый файл и запускаем его (рис. ??).

```

alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
alsalemshaker@alsalemshaker-VirtualBox:~/work/arch-pc/lab09$ ./lab09-5
Результат: 39

```

Создаем и смотрим на работу программы(работает неправильно)

Создаем исполняемый файл и запускаем его в отладчике GDB и смотрим на изменение регистров командой si (рис. ??).

```
Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>
eflags   0x206    [ PF IF ]
cs       0x23     35
ss       0x2b     43

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 <sprintf>
0x804910a <_start+34>   mov     eax,edi
0x804910c <_start+36>   call    0x8049086 <fprintf>
0x8049111 <_start+41>   call    0x80490db <quit>
0x8049116             add     BYTE PTR [eax],al

native process 8879 In: _start L11 PC: 0x80490f9
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f4 0x80490f4 <_start+12>
eflags   0x206    [ PF IF ]
cs       0x23     35
```

Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. ??).

Меняем файл

Создаем исполняемый файл и запускаем его (рис. ??).

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

Создаем и запускаем файл(работает корректно)

3 Выводы

Мы познакомились с методами отладки при помощи GDB и его возможностями.