

Отчёт по лабораторной работе №9

Лобанова Екатерина Евгеньевна

Содержание

1 Цель работы	6
2 Выполнение лабораторной работы	7
2.1 Реализация подпрограмм в NASM	7
2.2 Отладка программам с помощью GDB	9
2.3 Задание для самостоятельной работы	18
3 Выводы	23

Список иллюстраций

2.1	Создаем каталог с помощью команды mkdir и файл с помощью команды touch	7
2.2	Заполняем файл	8
2.3	Запускаем файл и проверяем его работу	8
2.4	Изменяем файл, добавляя еще одну подпрограмму	9
2.5	Запускаем файл и смотрим на его работу	9
2.6	Создаем файл	10
2.7	Заполняем файл	10
2.8	Загружаем исходный файл в отладчик	10
2.9	Запускаем программу командой run	11
2.10	Запускаем программу с брейкпоинтом	11
2.11	Смотрим дисассимилированный код программы	11
2.12	Переключаемся на синтаксис Intel	12
2.13	Включаем отображение регистров, их значений и результат дисассимилирования программы	13
2.14	Используем команду info breakpoints и создаем новую точку останова	14
2.15	Смотрим информацию	14
2.16	Отслеживаем регистры	15
2.17	Смотрим значение переменной	15
2.18	Смотрим значение переменной	16
2.19	Меняем символ	16
2.20	Меняем символ	16
2.21	Смотрим значение регистра	16
2.22	Изменяем регистор командой set	16
2.23	Прописываем команды c и quit	17
2.24	Копируем файл	17
2.25	Создаем и запускаем в отладчике файл	17
2.26	Устанавливаем точку останова	18
2.27	Изучаем полученные данные	18
2.28	Копируем файл	18
2.29	Изменяем файл	19
2.30	Проверяем работу программы	19
2.31	Создаем файл	20
2.32	Изменяем файл	20
2.33	Создаем и смотрим на работу программы(работает неправильно)	20
2.34	Ищем ошибку регистров в отладчике	21

2.35 Меняем файл	21
2.36 Создаем и запускаем файл(работает корректно)	22

Список таблиц

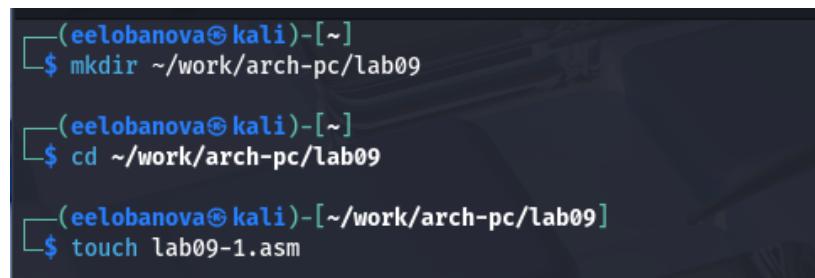
1 Цель работы

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. Рисунок 2.1).



```
(eelobanova㉿kali)-[~]
└─$ mkdir ~/work/arch-pc/lab09

(eelobanova㉿kali)-[~]
└─$ cd ~/work/arch-pc/lab09

(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ touch lab09-1.asm
```

Рисунок 2.1: 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. Рисунок 2.2).

```
GNU nano 8.4          lan9-1.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
```

Рисунок 2.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.3).

```
(eelobanova@kali)-[~/work/arch-pc/lab09]
$ nano lab09-1.asm
(eelobanova@kali)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-1.asm
(eelobanova@kali)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-1 lab09-1.o
(eelobanova@kali)-[~/work/arch-pc/lab09]
$ ./lab09-1
Введите x: 2
2x+7=11
```

Рисунок 2.3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. Рисунок 2.4).

```
GNU nano 8.4                               lab09-1.asm
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ', 0
result: DB '2(3x-1)+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
```

Рисунок 2.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.5).

```
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-1.asm
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-1 lab09-1.o
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ ./lab09-1
Введите x: 2
2(3x-1)+7=17
```

Рисунок 2.5: Запускаем файл и смотрим на его работу

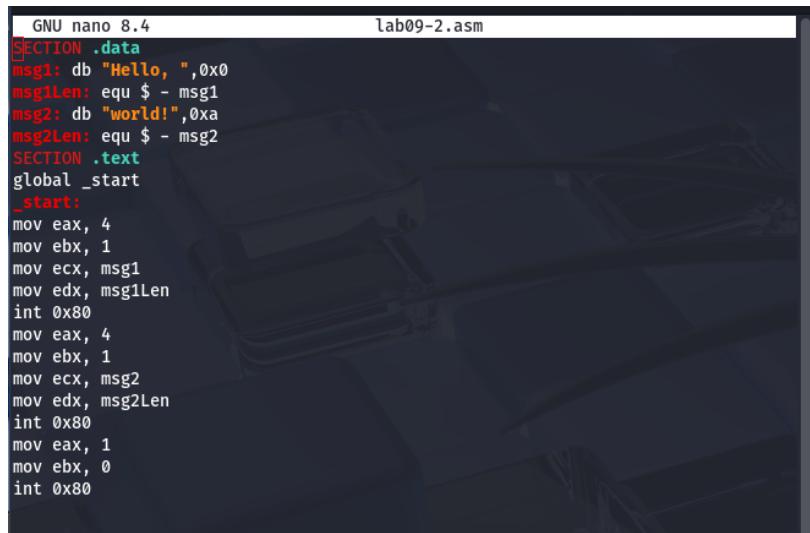
2.2 Отладка программам с помощью GDB

Создаем новый файл в каталоге(рис. Рисунок 2.6).

```
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ touch lab09-2.asm
```

Рисунок 2.6: Создаем файл

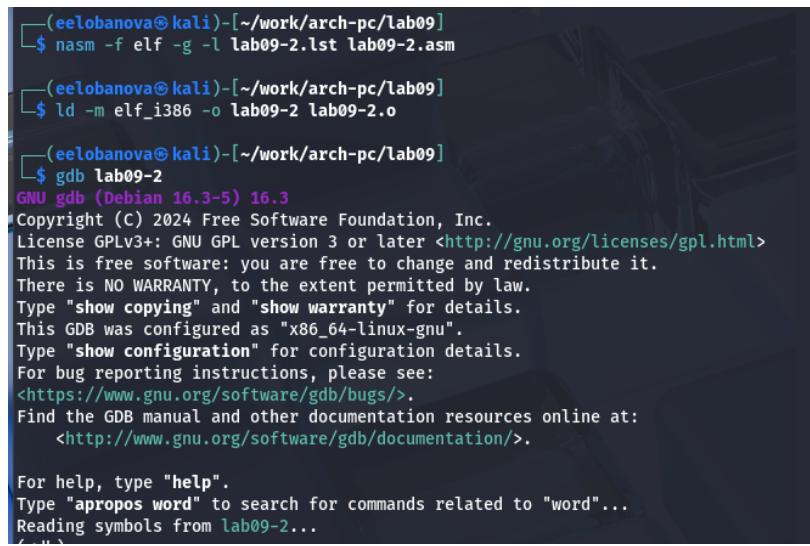
Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. Рисунок 2.7).



```
GNU nano 8.4                               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
```

Рисунок 2.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. Рисунок 2.8).



```
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ ld -m elf_i386 -o lab09-2 lab09-2.o

(eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ gdb lab09-2
GNU gdb (Debian 16.3-5) 16.3
Copyright (C) 2024 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...

```

Рисунок 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. Рисунок 2.9).

```
(gdb) run
Starting program: /home/eelobanova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 49926) exited normally]
(gdb) □
```

Рисунок 2.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку `_start` и запускаем программу (рис. Рисунок 2.10).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb) run
Starting program: /home/eelobanova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
11      mov eax, 4
(gdb) □
```

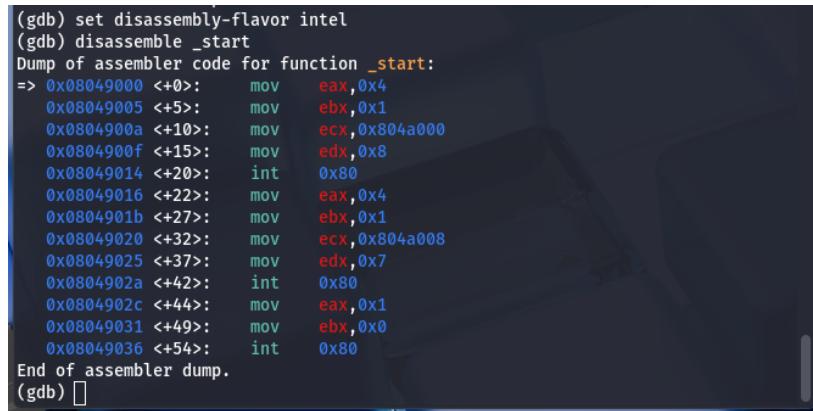
Рисунок 2.10: Запускаем программу с брейкпоинтом

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`(рис. Рисунок 2.11).

```
(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) □
```

Рисунок 2.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. Рисунок 2.12).



```
(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) 
```

Рисунок 2.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах ATT и Intel:

1. Порядок operandов: В ATT синтаксисе порядок operandов обратный, сначала указывается исходный operand, а затем - результирующий operand. В Intel синтаксисе порядок обычно прямой, результирующий operand указывается первым, а исходный - вторым.
2. Разделители: В ATT синтаксисе разделители operandов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).
3. Префиксы размера operandов: В ATT синтаксисе размер operandна указывается перед operandом с использованием префиксов, таких как «b» (byte), «w» (word), «l» (long) и «q» (quadword). В Intel синтаксисе размер operandна указывается после operandна с использованием суффиксов, таких как «b», «w», «d» и «q».
4. Знак operandов: В ATT синтаксисе operandы с позитивными значениями предваряются символом «*.Intel*».
5. Обозначение адресов: В ATT синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

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

Включаем режим псевдографики (рис. Рисунок 2.13).

The screenshot shows a terminal window with the following content:

```
Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp          0xfffffcf20    0xfffffcf20
ebp          0x0          0x0
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

B+>0x8049000 <_start>    mov   $0x4,%eax
0x8049005 <_start+5>      mov   $0x1,%ebx
0x804900a <_start+10>     mov   $0x804a000,%ecx
0x804900f <_start+15>     mov   $0x8,%edx
0x8049014 <_start+20>     int   $0x80
0x8049016 <_start+22>     mov   $0x4,%eax
0x804901b <_start+27>     mov   $0x1,%ebx
0x8049020 <_start+32>     mov   $0x804a008,%ecx
0x8049025 <_start+37>     mov   $0x7,%edx
0x804902a <_start+42>     int   $0x80
0x804902c <_start+44>     mov   $0x1,%eax
0x8049031 <_start+49>     mov   $0x0,%ebx
0x8049036 <_start+54>     int   $0x80
0x8049038                  add   %al,(%eax)

native process 60021 (asm) In: _start          L11   PC: 0x8049000
(gdb) layout regs
```

Рисунок 2.13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. Рисунок 2.14).

```

Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp    0xfffffcf20 0xfffffcf20
ebp          0x0          0x0
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

B+>0x8049000 <_start>    mov   $0x4,%eax
                           mov   $0x1,%ebx
                           mov   $0x804a000,%ecx
                           mov   $0x8,%edx
                           int   $0x80
                           mov   $0x4,%eax
                           mov   $0x1,%ebx
                           mov   $0x804a008,%ecx
                           mov   $0x7,%edx
                           int   $0x80
                           mov   $0x1,%eax
                           mov   $0x0,%ebx
                           b+ 0x8049031 <_start+49>    mov   $0x0,%eax
                           0x8049036 <_start+54>    int   $0x80
                           0x8049038           add   %al,(%eax)

native_process 61596 (asm) In: _start          L11   PC: 0x8049000
(gdb) layout regs
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 11.
(gdb) run
Starting program: /home/eelobanova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:11
(gdb) info breakpoints
Num  Type      Disp Enb Address  What
1    breakpoint keep y  0x08049000 lab09-2.asm:11
      breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 22.
(gdb) []

```

Рисунок 2.14: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. Рисунок 2.15).

```

(gdb) i b
Num  Type      Disp Enb Address  What
1    breakpoint keep y  0x08049000 lab09-2.asm:11
      breakpoint already hit 1 time
2    breakpoint keep y  0x08049031 lab09-2.asm:22
(gdb) []

```

Рисунок 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. Рисунок 2.16).

```

Register group: general
eax          0x8          8
ecx          0x804a000    134520832
edx          0x8          8
ebx          0x1          1
esp          0xfffffcf20  0xfffffcf20
ebp          0x0          0x0
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

B+ 0x8049000 <_start>    mov   $0x4,%eax
0x8049005 <_start+5>    mov   $0x1,%ebx
0x804900a <_start+10>   mov   $0x804a000,%ecx
0x804900f <_start+15>   mov   $0x8,%edx
0x8049014 <_start+20>   int    $0x80
>0x8049016 <_start+22>  mov   $0x4,%eax
0x804901b <_start+27>   mov   $0x1,%ebx
0x8049020 <_start+32>   mov   $0x804a008,%ecx
0x8049025 <_start+37>   mov   $0x7,%edx
0x804902a <_start+42>   int    $0x80
0x804902c <_start+44>   mov   $0x1,%eax
b+ 0x8049031 <_start+49>  mov   $0x0,%ebx
0x8049036 <_start+54>   int    $0x80
0x8049038          add   %al,(%eax)

native process 61596 (asm) In: _start          L16   PC: 0x8049016
Num  Type      Disp Enb Address   What
1   breakpoint  keep y  0x08049000 lab09-2.asm:11
      breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 22.
(gdb) i b
Num  Type      Disp Enb Address   What
1   breakpoint  keep y  0x08049000 lab09-2.asm:11
      breakpoint already hit 1 time
2   breakpoint  keep y  0x08049031 lab09-2.asm:22
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) 

```

Рисунок 2.16: Отслеживаем регистры

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

Смотрим значение переменной msg1 по имени (рис. Рисунок 2.17).

```

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

```

Рисунок 2.17: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. Рисунок 2.18).

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

Рисунок 2.18: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. Рисунок 2.19).

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

Рисунок 2.19: Меняем символ

Изменим первый символ переменной msg2 (рис. Рисунок 2.20).

```
(gdb) set {char}&msg2='L'  
(gdb) x/lrb &msg2  
0x804a008 <msg2>:      "Lorld!\n\034"  
(gdb) 
```

Рисунок 2.20: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. Рисунок 2.21).

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

Рисунок 2.21: Смотрим значение регистра

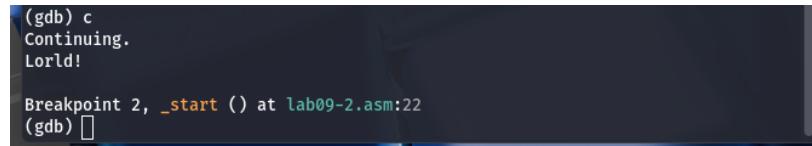
Изменяем регистор ebx (рис. Рисунок 2.22).

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$5 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$6 = 2  
(gdb) 
```

Рисунок 2.22: Изменяю регистор командой set

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

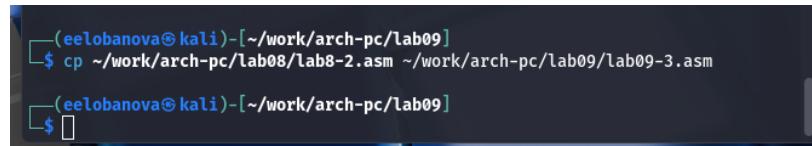
Прописываем команды для завершения программы и выхода из GDB (рис. Рисунок 2.23).



```
(gdb) c
Continuing.
World!
Breakpoint 2, _start () at lab09-2.asm:22
(gdb) 
```

Рисунок 2.23: Прописываем команды с и quit

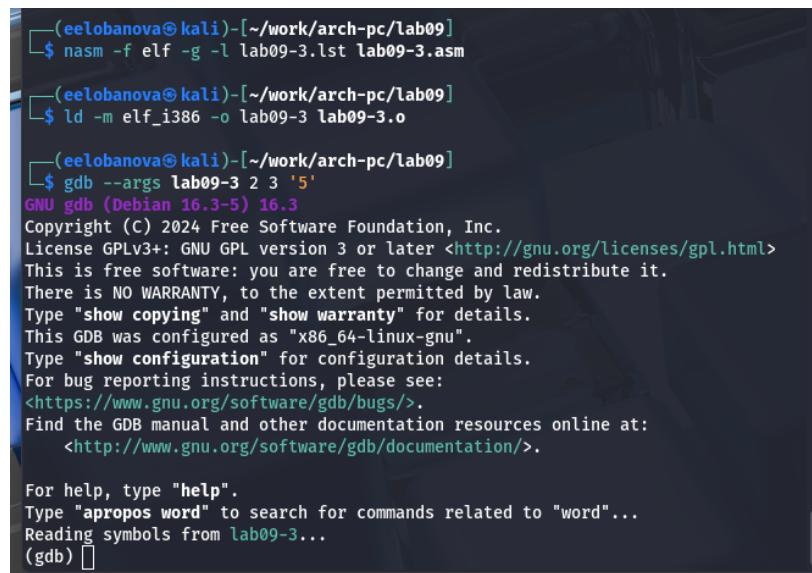
Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. Рисунок 2.24).



```
[eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ 
```

Рисунок 2.24: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. Рисунок 2.25).



```
[eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ ld -m elf_i386 -o lab09-3 lab09-3.o
[eelobanova㉿kali)-[~/work/arch-pc/lab09]
└─$ gdb --args lab09-3 2 3 '5'
GNU gdb (Debian 16.3-5) 16.3
Copyright (C) 2024 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) 
```

Рисунок 2.25: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. Рисунок 2.26).

```
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/eelobanova/work/arch-pc/lab09/lab09-3 2 3 5

Breakpoint 1, _start () at lab09-3.asm:5
5      pop  ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffcf10:    0x00000004
(gdb) 
```

Рисунок 2.26: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. Рисунок 2.27).

```
(gdb) x/s *(void**)$esp + 4
0xffffd0f2:    "/home/eelobanova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)$esp + 8
0xffffd11e:    "2"
(gdb) x/s *(void**)$esp + 12
0xffffd120:    "3"
(gdb) x/s *(void**)$esp + 16
0xffffd122:    "5"
(gdb) x/s *(void**)$esp + 20
0x0:    <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)$esp + 24
0xffffd124:    "SYSTEMD_EXEC_PID=1697"
(gdb) 
```

Рисунок 2.27: Изучаем полученные данные

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

2.3 Задание для самостоятельной работы

2.3.1 Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем lab09-3.asm (рис. Рисунок 2.28).

```
[eelobanova@kali] -[~/work/arch-pc/lab09]
$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
```

Рисунок 2.28: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. Рисунок 2.29).

```
GNU nano 8.4          lab09-4.asm *
%include "in_out.asm"
SECTION .data
msg db "Результат: ",0

SECTION .bss
prm: resb 80

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 esi, eax
    loop next

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

calcul:
    sub eax, 1
    mov ebx, 10
    mul ebx
    ret
```

Рисунок 2.29: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.30).

```
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ nano lab09-4.asm

(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-4.asm

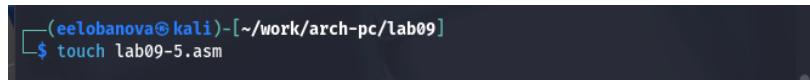
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-4 lab09-4.o

(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ ./lab09-4 4 4 5
Результат: 100
```

Рисунок 2.30: Проверяем работу программы

2.3.2 Задание 2

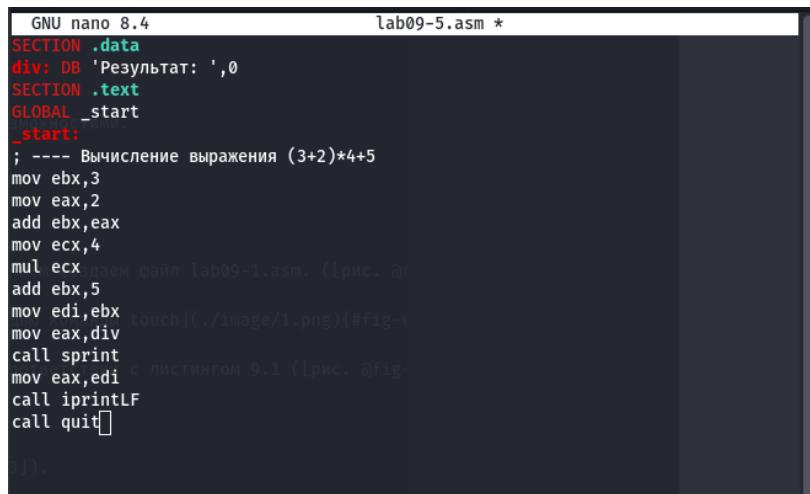
Создаем новый файл в директории (рис. Рисунок 2.31).



```
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ touch lab09-5.asm
```

Рисунок 2.31: Создаем файл

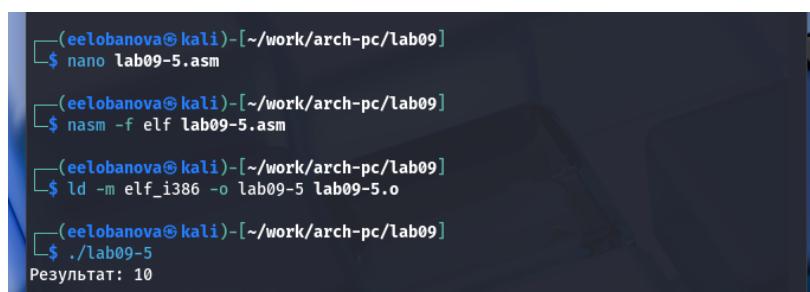
Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. Рисунок 2.32).



```
GNU nano 8.4                               lab09-5.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
; ----- добавляем файл lab09-1.asm. ([рис. @])
add ebx,5
mov edi,ebx
; ----- добавляем файл lab09-2.asm. ([рис. @])
mov eax,div
call sprint
; ----- добавляем файл lab09-3.asm. ([рис. @])
mov eax,edi
call iprintLF
call quit[]
```

Рисунок 2.32: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.33).



```
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ nano lab09-5.asm

(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-5.asm

(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-5 lab09-5.o

(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ ./lab09-5
Результат: 10
```

Рисунок 2.33: 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. Рисунок 2.34).

The screenshot shows a debugger interface with two main sections. The top section displays the general register group:

Register group: general		
eax	0x0	0
ecx	0x0	0
edx	0x0	0
ebx	0x3	3
esp	0xffffcef0	0xffffcef0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490ed	0x80490ed <_start+5>
eflags	0x10202	[IF RF]
cs	0x23	35
ss	0x2b	43

The bottom section shows the assembly code for the program:

```

B+ 0x80490e8 <_start>    mov    $0x3,%ebx
>0x80490f2 <_start+5>    mov    $0x2,%eax
    0x80490f2 <_start+10>   add    %eax,%ebx
    0x80490f4 <_start+12>   mov    $0x4,%ecx
    0x80490f9 <_start+17>   mul    %ecx
    0x80490fb <_start+19>   add    $0x5,%ebx
    0x80490fe <_start+22>   mov    %ebx,%edi
    0x8049100 <_start+24>   mov    $0x804a000,%eax
    0x8049105 <_start+29>   call   0x804900f <sprint>
    0x804910a <_start+34>   mov    %edi,%eax
    0x804910c <_start+36>   call   0x8049086 <iprintf>
    0x8049111 <_start+41>   call   0x80490db <quit>

```

At the bottom of the debugger window, it says "native process 102270 (asm) In: _start L8 PC: 0x80490ed".

Рисунок 2.34: Ищем ошибку регистров в отладчике

Изменяем программу для корректной работы (рис. Рисунок 2.35).

The screenshot shows a terminal window with the command "GNU nano 8.4" at the top. Below it is the assembly code for the program:

```

GNU nano 8.4                               lab09-5.asm *
#include "in_out.asm"
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov ebx, 3
    mov eax, 2
    add eax, ebx[]
    mov ecx, 4
    mul ecx
    add eax, 5
    mov edi, eax
    mov eax, div
    call sprint
    mov eax, edi
    call iprintf
    call quit

```

Рисунок 2.35: Меняю файл

Создаем исполняемый файл и запускаем его (рис. Рисунок 2.36).

```
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ nano lab09-5.asm
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-5.asm
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-5 lab09-5.o
(eelobanova㉿kali)-[~/work/arch-pc/lab09]
$ ./lab09-5
Результат: 25
```

Рисунок 2.36: Создаем и запускаем файл(работает корректно)

3 Выводы

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

::: {#refs} :::