

Лабораторная работа №9

НПИбд-01-25 №1032252598

Иванова Ангелина Олеговна

Содержание

1 Цель работы	5
2 Выполнение лабораторной работы	6
2.1 Задания лабораторной работы	6
2.2 Задание для самостоятельной работы	28
3 Выводы	33

Список иллюстраций

2.1	Создание каталога и файла	6
2.2	Измененный файл lab09-1.asm	7
2.3	Создание исполняемого файла и вывод его работы	7
2.4	Измененный текст программы	10
2.5	Создание исполняемого файла и вывод его работы	11
2.6	Измененный файл lab09-2.asm	12
2.7	Создание исполняемого файла и проверка его работы в оболочке GDB	13
2.8	Добавление брейкпоинта и запуск программы	13
2.9	Просмотр дисассимилированного кода программы	14
2.10	Переключение и просмотр дисассимилированного кода программы .	14
2.11	Включение режима псевдографики	16
2.12	Проверка меток	17
2.13	Установка точки останова и просмотр информации	18
2.14	Выполнение 1 инструкции	19
2.15	Выполнение 2 инструкции	20
2.16	Выполнение 3 инструкции	21
2.17	Выполнение 4 инструкции	22
2.18	Выполнение 5 инструкции	23
2.19	Использование команды info registers	24
2.20	Просмотр значения переменных msg1 и msg2	25
2.21	Изменение первого символа переменной msg1	25
2.22	Изменение первого символа переменной msg2	25
2.23	Выведение регистра в разных форматах	26
2.24	Изменение регистра	26
2.25	Копирование и создание исполняемого файла	27
2.26	Загрузка в отладчик, установка точки останова и запуск	27
2.27	Вывод позиций стека	28
2.28	Работа созданного нами файла	30
2.29	Неверный результат работы программы	30
2.30	Поиск ошибки	30
2.31	Работа созданного нами файла	32

Список таблиц

1 Цель работы

Целью данной лабораторной работы является приобретение навыков написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Выполнение лабораторной работы

2.1 Задания лабораторной работы

2.1.1 Реализация подпрограмм в NASM

Создали каталог для выполнения лабораторной работы № 9, перешли в него и создали файл lab09-1.asm

```
aoivanova@ubuntu:~/Desktop$ cd
aoivanova@ubuntu:~$ mkdir ~/work/arch-pc/lab09
aoivanova@ubuntu:~$ cd ~/work/arch-pc/lab09
aoivanova@ubuntu:~/work/arch-pc/lab09$ touch lab09-1.asm
aoivanova@ubuntu:~/work/arch-pc/lab09$ █
```

Рисунок 2.1: Создание каталога и файла

Внимательно изучили текст программы первого листинга. Ввели в файл lab09-1.asm текст программы из первого листинга.

```

GNU nano 7.2                               lab09-1.asm *
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF

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

```

Рисунок 2.2: Измененный файл lab09-1.asm

Создали исполняемый файл и запустили его. Получили корректный результат работы данной программы

```

aoivanova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
aoivanova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
aoivanova@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 12
2x+7=31
aoivanova@ubuntu:~/work/arch-pc/lab09$ 

```

Рисунок 2.3: Создание исполняемого файла и вывод его работы

Далее изменили программу, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $g(x)$, результат возвращается в `_calcul` и

вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран.

Листинг 1:

```
%include „in_out.asm“  
SECTION .data  
msg: DB „Ведите x:“,0  
result: DB ’f(g(x))=2*(3x-1)+7=’,0  
SECTION .bss  
x: RESB 80  
res: RESB 80  
SECTION .text  
GLOBAL _start  
  
_start:  
;-----  
; Основная программа  
;-----  
mov eax, msg  
call sprint  
mov ecx, x  
mov edx, 80  
call sread  
mov eax, x  
call atoi  
call _calcul  
mov eax, result  
call sprint  
mov eax, [res]  
call iprintLF  
call quit
```

```
;-----
; Подпрограмма вычисления f(g(x))
;-----

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

_subcalcul:
    push ebx
    mov ebx, 3
    mul ebx
    sub eax, 1
    pop ebx
    ret
```

The screenshot shows a terminal window titled "aoivanova@ubuntu: ~/work/arch-pc/lab09". The file being edited is "lab09-1.asm". The assembly code is as follows:

```
GNU nano 7.2                               lab09-1.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=2*(3x-1)+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintlnF
call quit
;-----
; Подпрограмма вычисления f(g(x))
;-----
_calcul:
    push ebx
    call _subcalcul
    mov ebx, 2
[ Read 45 lines ]

```

At the bottom of the terminal window, there is a menu bar with the following options:

- ^G Help
- ^O Write Out
- ^W Where Is
- ^K Cut
- ^T Execute
- ^X Exit
- ^R Read File
- ^\\ Replace
- ^U Paste
- ^J Justify

Рисунок 2.4: Измененный текст программы

Создали исполняемый файл и проверили его работу

The screenshot shows a terminal window with the following command history:

```
aoivanova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
aoivanova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
aoivanova@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Ведите x: 5
f(g(x))=2*(3x-1)+7=35
aoivanova@ubuntu:~/work/arch-pc/lab09$
```

Рисунок 2.5: Создание исполняемого файла и вывод его работы

2.1.2 Отладка программам с помощью GDB

Создали файл lab09-2.asm и ввели в него текст программы из второго листинга

The screenshot shows a terminal window with the title bar "aoivanova@ubuntu: ~/work/arch-pc/lab09". The window contains the assembly code for "lab09-2.asm". The code defines two strings in the .data section: "Hello, " and "world!". It then moves these strings to the .text section, sets up registers for the movsb instruction, and performs three iterations of the loop. The assembly code is as follows:

```
GNU nano 7.2                               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
```

The status bar at the bottom of the terminal window indicates "[Wrote 21 lines]". The keyboard shortcut menu below the status bar includes: Help (^G), Write Out (^O), Where Is (^W), Cut (^K), Execute (^T), Exit (^X), Read File (^R), Replace (^|), Paste (^U), and Justify (^J).

Рисунок 2.6: Измененный файл lab09-2.asm

Создали исполняемый файл. Для работы с GDB в исполняемый файл добавили отладочную информацию, для этого трансляцию программ провели с ключом „-g“. Загрузили исполняемый файл в отладчик gdb и проверили работу программы, запустив ее в оболочке GDB с помощью команды run

```
aoivanova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
aoivanova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
aoivanova@ubuntu:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
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...
(gdb) run
Starting program: /home/aoivanova/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit
.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 8760) exited normally]
```

Рисунок 2.7: Создание исполняемого файла и проверка его работы в оболочке GDB

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

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

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

Рисунок 2.8: Добавление брейктоинта и запуск программы

Посмотрели 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.
```

Рисунок 2.9: Просмотр дисассимилированного кода программы

Переключились 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.
```

Рисунок 2.10: Переключение и просмотр дисассимилированного кода программы

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 “».»
- 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»).

Включили режим псевдографики для более удобного анализа программы

The screenshot shows a terminal window with the following content:

```
aoivanova@ubuntu: ~/work/arch-pc/lab09
Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp 0xfffffd000 0xfffffd000
ebp          0x0          0x0

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 8965 (asm) In: _start
(gdb) layout regs
(gdb)
```

Рисунок 2.11: Включение режима псевдографики

2.1.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (`_start`).

Проверили это с помощью команды `info breakpoints`

The screenshot shows a terminal window with the following content:

```
aoivanova@ubuntu: ~/work/arch-pc/lab09
Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp 0xfffffd000 0xfffffd000
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip 0x8049000 0x8049000 <_start>
eflags       0x202      [ IF ]
```

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

```
native process 9158 (asm) 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)
```

Рисунок 2.12: Проверка меток

Определили адрес предпоследней инструкции (mov ebx,0x0) и установили точку останова. Посмотрели информацию о всех установленных точках останова

```

Register group: general
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp 0xfffffd000 0xfffffd000
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip 0x8049000 0x8049000 <_start>
eflags      0x202      [ IF ]

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
0x8049038          add    BYTE PTR [eax],al

native process 9158 (asm) In: _start          L9      PC: 0x8049000
(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) 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
(gdb)

```

Рисунок 2.13: Установка точки останова и просмотр информации

2.1.4 Работа с данными программы в GDB

Выполнili 5 инструкций с помощью команды stepi (или si) и проследили за изменением значений регистров. Итого, за 5 инструкций регистры изменились следующим образом:

1. eax = 4

2. ebx = 1
3. ecx = 134520832
4. edx = 8
5. eax = 8

The screenshot shows a terminal window titled "aoivanova@ubuntu: ~/work/arch-pc/lab09". The window displays the following information:

- Register group: general**

eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xfffffd000	0xfffffd000
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049005	0x8049005 <_start+5>
eflags	0x202	[IF]

- Assembly Instructions:** A list of assembly instructions starting at address 0x8049000. The instruction at address 0x8049005 is highlighted in red.

```

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

```

- GDB Session:**

```

native process 929 (asm) In: _start                                L10   PC: 0x8049005
(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
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) si
(gdb)

```

Рисунок 2.14: Выполнение 1 инструкции

```
aoivanova@ubuntu: ~/work/arch-pc/lab09
```

Register group: general		
eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x1	1
esp	0xfffffd000	0xfffffd000
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x804900a	0x804900a <_start+10>
eflags	0x202	[IF]

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

```
native process 9289 (asm) In: _start          L11    PC: 0x804900a
(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
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) si
(gdb) si
(gdb)
```

Рисунок 2.15: Выполнение 2 инструкции

```
aoivanova@ubuntu:~/work/arch-pc/lab09
```

Register group: general		
eax	0x4	4
ecx	0x804a000	134520832
edx	0x0	0
ebx	0x1	1
esp	0xfffffd000	0xfffffd000
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x804900f	0x804900f <_start+15>
eflags	0x202	[IF]

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

```
native process 9289 (asm) In: _start          L12  PC: 0x804900f
Num      Type            Disp Enb Address     What
1        breakpoint      keep y  0x08049000 lab09-2.asm:9
           breakpoint already hit 1 time
(gdb) break 0x8049031
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) si
(gdb) si
(gdb) si
(gdb) ■
```

Рисунок 2.16: Выполнение 3 инструкции

```
aoivanova@ubuntu:~/work/arch-pc/lab09
```

Register group: general		
eax	0x4	4
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xfffffd000	0xfffffd000
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049014	0x8049014 <_start+20>
eflags	0x202	[IF]

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

```
native process 9289 (asm) In: _start          L13  PC: 0x8049014
1      breakpoint    keep y  0x08049000 lab09-2.asm:9
      breakpoint already hit 1 time
(gdb) break 0x8049031
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рисунок 2.17: Выполнение 4 инструкции

The screenshot shows a terminal window with the following content:

```

aoivanova@ubuntu: ~/work/arch-pc/lab09
Register group: general
eax          0x8          8
ecx          0x804a000    134520832
edx          0x8          8
ebx          0x1          1
esp          0xfffffd000  0xfffffd000
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip          0x8049016    0x8049016 <_start+22>
eflags        0x202      [ IF ]

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

native process 9289 (asm) In: _start                                L14    PC: 0x8049016
breakpoint already hit 1 time
(gdb) break 0x8049031
Function "0x8049031" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рисунок 2.18: Выполнение 5 инструкции

Посмотрели содержимое регистров с помощью команды `info registers`

```

aoivanova@ubuntu: ~/work/arch-pc/lab09
Register group: general
eax      0x8          8
ecx      0x804a000    134520832
edx      0x8          8
ebx      0x1          1
esp      0xfffffd000  0xfffffd000
ebp      0x0          0x0
esi      0x0          0
edi      0x0          0
eip      0x8049016    0x8049016 <_start+22>
eflags   0x202       [ IF ]
B+ 0x8049000 <_start>    mov    eax,0x4
0x8049005 <_start+5>    mov    ebx,0x1
0x804900a <_start+10>   mov    ecx,0x804a000
0x804900f <_start+15>   mov    edx,0x8
0x8049014 <_start+20>   int    0x80
>0x8049016 <_start+22>  mov    eax,0x4
0x804901b <_start+27>   mov    ebx,0x1
0x8049020 <_start+32>   mov    ecx,0x804a008
0x8049025 <_start+37>   mov    edx,0x7
0x804902a <_start+42>   int    0x80
0x804902c <_start+44>   mov    eax,0x1
native process 9289 (asm) In: _start          L14   PC: 0x8049016
eax      0x8          8
ecx      0x804a000    134520832
edx      0x8          8
ebx      0x1          1
esp      0xfffffd000  0xfffffd000
ebp      0x0          0x0
esi      0x0          0
edi      0x0          0
eip      0x8049016    0x8049016 <_start+22>
eflags   0x202       [ IF ]
cs       0x23         35
--Type <RET> for more, q to quit, c to continue without paging--■

```

Рисунок 2.19: Использование команды info registers

Посмотрели значение переменной msg1 по имени и посмотрели значение переменной msg2 по адресу

```

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

native process 9289 (asm) In: _start          L14   PC: 0x8049016
eflags      0x202           [ IF ]
cs          0x23            35
--Type <RET> for more, q to quit, c to continue without paging--
ss          0x2b            43
ds          0x2b            43
es          0x2b            43
fs          0x0              0
gs          0x0              0
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"

```

Рисунок 2.20: Просмотр значения переменных msg1 и msg2

Изменили первый символ переменной msg1

```

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

```

Рисунок 2.21: Изменение первого символа переменной msg1

Заменили первый символ во второй переменной msg2

```

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

```

Рисунок 2.22: Изменение первого символа переменной msg2

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

```
(gdb) p/x $edx  
$1 = 0x8  
(gdb) p/t $edx  
$2 = 1000  
(gdb) p/c $edx  
$3 = 8 '\b'  
(gdb) []
```

Рисунок 2.23: Выведение регистра в разных форматах

С помощью команды set измените значение регистра ebx

```
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$4 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$5 = 2
```

Рисунок 2.24: Изменение регистра

Разница вывода заключается в том, что „2“ интерпретируется как ASCII-код, а 2 как обычное число.

set \$ebx=„2“ - „2“ это символ, его ASCII-код 50, поэтому „p/s \$ebx“ выводит число 50
set \$ebx=2 — просто число 2, „p/s \$ebx“ выводит 2.

Завершили выполнение программы с помощью команды continue (сокращенно c) и вышли из GDB с помощью команды quit (сокращено q).

2.1.5 Обработка аргументов командной строки в GDB

Скопировали файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm. Создали исполняемый файл.

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

Рисунок 2.25: Копирование и создание исполняемого файла

Загрузите исполняемый файл в отладчик, указав аргументы. Для начала установили точку останова перед первой инструкцией в программе и запустили ее.

```
aoivanova@ubuntu:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумен
т 2 'аргумент 3'
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm>
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) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/aoivanova/work/arch-pc/lab09/lab09-3 аргумент1 аргу
мент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit
.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `есх` количество
```

Рисунок 2.26: Загрузка в отладчик, установка точки останова и запуск

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки. Посмотрели 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.

```
(gdb) x/x $esp
0xfffffcf0:      0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xffffd195:      "/home/aoivanova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffd1c0:      "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffd1d2:      "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xfffffd1e3:      "2"
(gdb) x/s *(void**)( $esp + 20)
0xfffffd1e5:      "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
```

Рисунок 2.27: Вывод позиций стека

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

2.2 Задание для самостоятельной работы

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

Листинг 2

```
%include „in_out.asm“
SECTION .data
msg db «Результат:», 0
SECTION .bss
res: RESB 80
```

```
SECTION .text
```

```
global _start
```

```
    _start:
```

```
    pop ecx
```

```
    pop edx
```

```
    sub ecx, 1
```

```
    mov dword [res], 0
```

```
    next:
```

```
    cmp ecx, 0h
```

```
    jz _end
```

```
    pop eax
```

```
    call atoi
```

```
    call _calcul
```

```
    add [res], eax
```

```
    dec ecx
```

```
    jmp next
```

```
    _end:
```

```
    mov eax, msg
```

```
    call sprint
```

```
    mov eax, [res]
```

```
    call iprintLF
```

```
    call quit
```

```
    _calcul:
```

```
    mov ebx, 8
```

```
    mul ebx
```

```
    sub eax, 3
```

```
    ret
```

```

aoivanova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
aoivanova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
aoivanova@ubuntu:~/work/arch-pc/lab09$ ./lab09-4 1 2 3
Результат: 39
aoivanova@ubuntu:~/work/arch-pc/lab09$ 

```

Рисунок 2.28: Работа созданного нами файла

- В третьем листинге приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат.

```

aoivanova@ubuntu:~/work/arch-pc/lab09$ nano lab09-5.asm
aoivanova@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
aoivanova@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
aoivanova@ubuntu:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10

```

Рисунок 2.29: Неверный результат работы программы

С помощью отладчика GDB, анализируя изменения значений регистров, определили ошибку

The screenshot shows the GDB interface with two panes. The top pane displays the general register dump:

Register group: general		
eax	0x2	2
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xfffffcf80	0xfffffcf80
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490f9	0x80490f9 <_start+17>
eflags	0x206	[PF IF]

The bottom pane shows the assembly code:

```

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>
0x804910a <_start+34>   mov    eax,edi

```

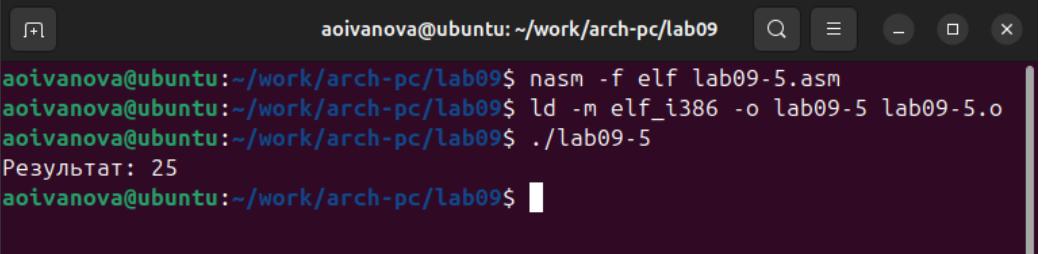
Рисунок 2.30: Поиск ошибки

Исправили ошибку в программе.

Листинг 3

```
%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
```

Создали исполняемый файл и проверили корректность выполнения программы



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

Рисунок 2.31: Работа созданного нами файла

3 Выводы

Приобрели навыки написания программ с использованием подпрограмм и познакомились с методами отладки при помощи GDB и его основными возможностями.