

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Хатамов Эзиз

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация подпрограмм в NASM	8
4.2	Отладка программ с помощью GDB	9
4.3	Работа с данными программы в GDB	14
4.4	Обработка аргументов командной строки в GDB	17
5	Самостоятельная работа	20
5.1	Первая задача	20
5.2	Вторая задача	21
6	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Создания каталога и файла	8
4.2	Введения программы в файл	9
4.3	Создание исполняемого файла и запуск его	9
4.4	Создание файла lab09-2.asm	10
4.5	Введения программы печати сообщения Hello world	10
4.6	Создание исполняемого файла lab09-2.asm и запуск его	10
4.7	Загрузка исполняемого файла в отладчик gdb	11
4.8	проверил работу программы	11
4.9	установка брейкпоинт на метку _start	12
4.10	Просмотр дисассимилированный код программы	12
4.11	Переключения на отображение команд с Intel'овским синтаксисом	13
4.12	режим псевдографики	13
4.13	Установка и Проверка точек сотанова	14
4.14	Просмотр содержимое регистров	15
4.15	Просмотр значение и изменения региястра	15
4.16	Просмотр значение переменной msg1	16
4.17	Просмотр значение переменной msg2	16
4.18	Изменения значения переменной msg1	16
4.19	Изменения переменной msg2	16
4.20	Вывод значения регистров еsx и еax	17
4.21	Изменения значения регистра ebx	17
4.22	Выход	17
4.23	Копирования файла	18
4.24	Запуск файла с аргументами	18
4.25	Запуск файла через метку	18
4.26	Адрес вершины стека	19
5.1	Копирования файла ил lab08	20
5.2	Преоброзования файла	21
5.3	Запуск файла	21
5.4	Запуск программы в окладчике	22
5.5	Исправления ошибок	23
5.6	Запуск исполняемого файла	23

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Задание для самостоятельной работы

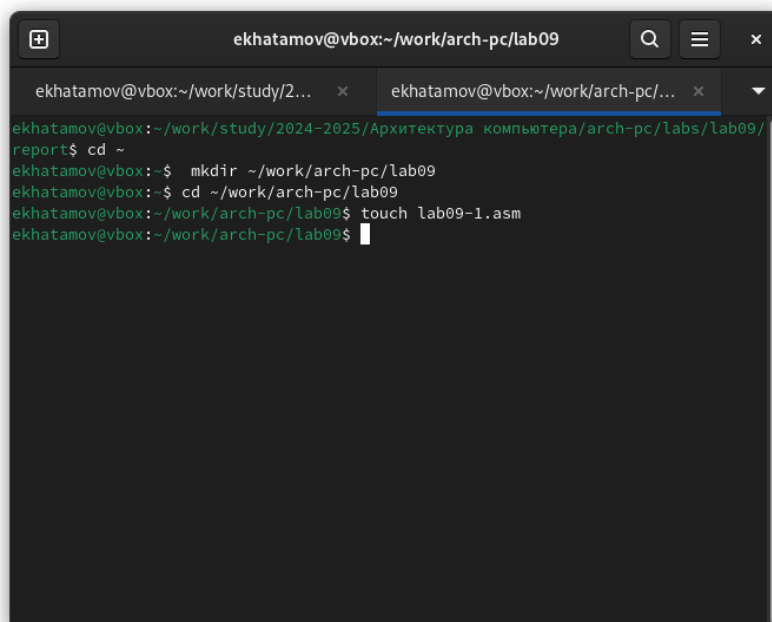
3 Теоретическое введение

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. Можно выделить следующие типы ошибок: • синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске пр

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

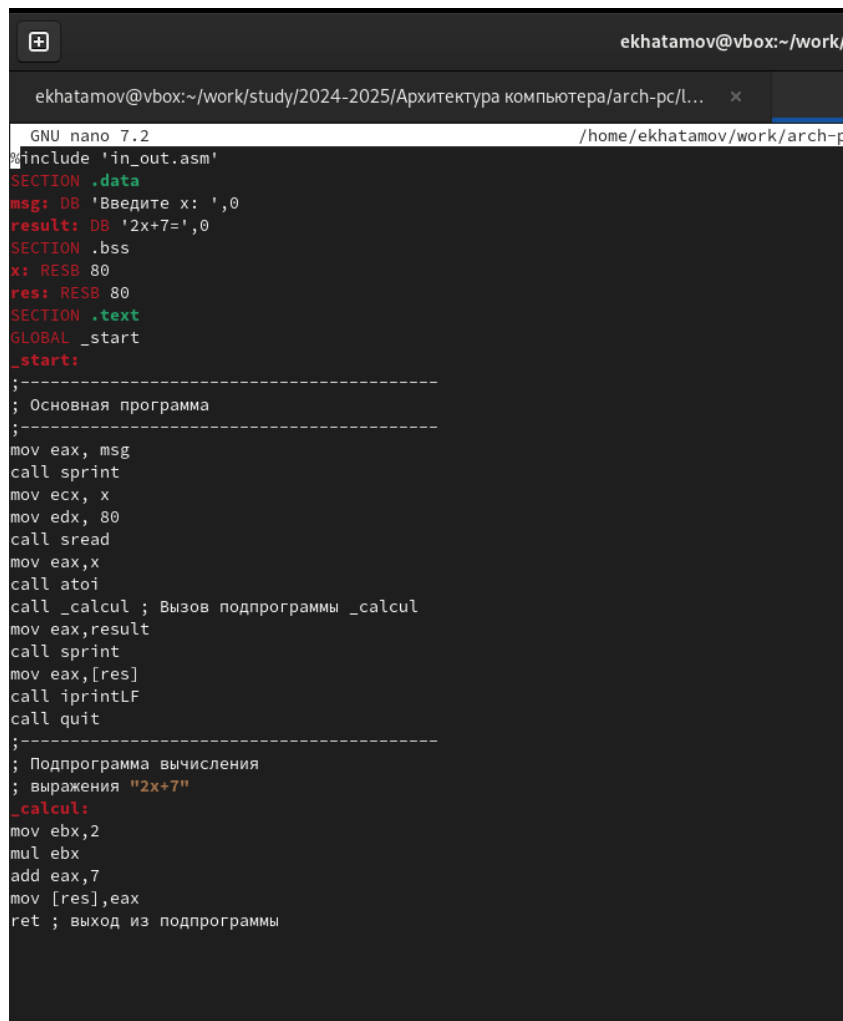
Создал каталог для выполнения лабораторной работы № 9, перешел в него и создал файл lab09-1.asm:



```
ekhatamov@vbox:~/work/arch-pc/lab09
ekhatamov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/
report$ cd ~
ekhatamov@vbox:~$ mkdir ~/work/arch-pc/lab09
ekhatamov@vbox:~$ cd ~/work/arch-pc/lab09
ekhatamov@vbox:~/work/arch-pc/lab09$ touch lab09-1.asm
ekhatamov@vbox:~/work/arch-pc/lab09$
```

Рис. 4.1: Создания каталога и файла

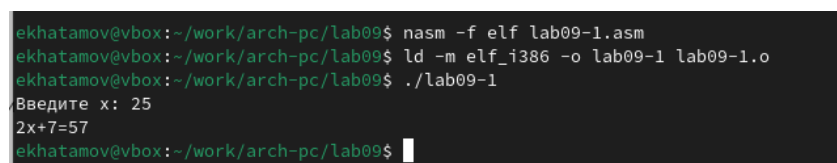
В качестве примера рассмотрел программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`. В данном примере `x` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Написал программу в Файл

A screenshot of a terminal window with a dark background. The window title is 'ekhatamov@vbox:~/work/'. The terminal shows the GNU nano 7.2 editor with an assembly file. The code includes a data section with a message and a result, a bss section for variables, and a text section with the main program logic. The main program prompts for input, reads it, calculates 2x+7, and prints the result. A subprogram _calcul is also defined.

```
GNU nano 7.2 /home/ekhatamov/work/arch-pc/...
#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
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
```

Рис. 4.2: Введения программы в файл

Создал исполняемый файл и запустил его

A screenshot of a terminal window showing the compilation and execution of the assembly program. The user runs 'nasm -f elf lab09-1.asm' to create an object file, then 'ld -m elf_i386 -o lab09-1 lab09-1.o' to create an executable. Finally, they run './lab09-1', which prompts for input '25' and outputs '2x+7=57'.

```
ekhatamov@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
ekhatamov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
ekhatamov@vbox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 25
2x+7=57
ekhatamov@vbox:~/work/arch-pc/lab09$
```

Рис. 4.3: Создание исполняемого файла и запуск его

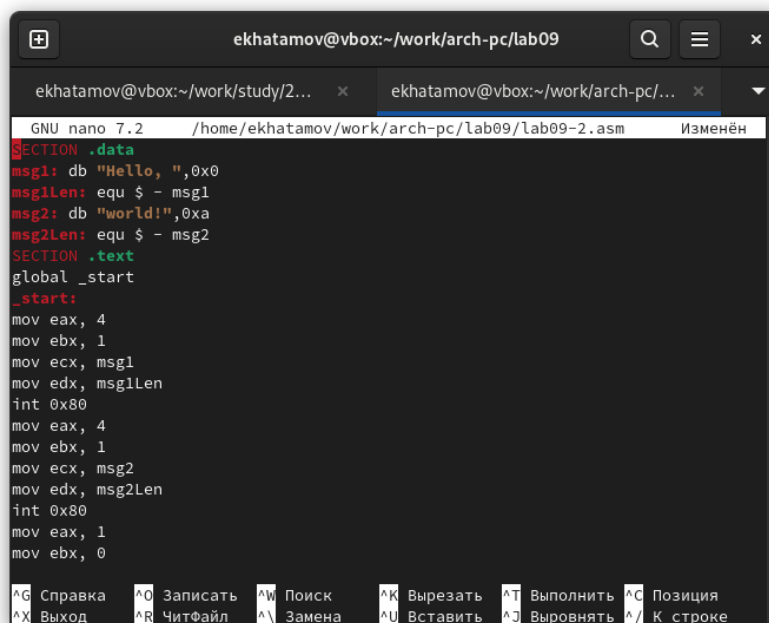
4.2 Отладка программ с помощью GDB

Я создал новый файл lab09-2.asm

```
ekhatamov@vbox:~/work/arch-pc/lab09$ touch lab09-2.asm
ekhatamov@vbox:~/work/arch-pc/lab09$
```

Рис. 4.4: Создание файла lab09-2.asm

ввел туда программу печати сообщения Hello world!



```
GNU nano 7.2 /home/ekhatamov/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
```

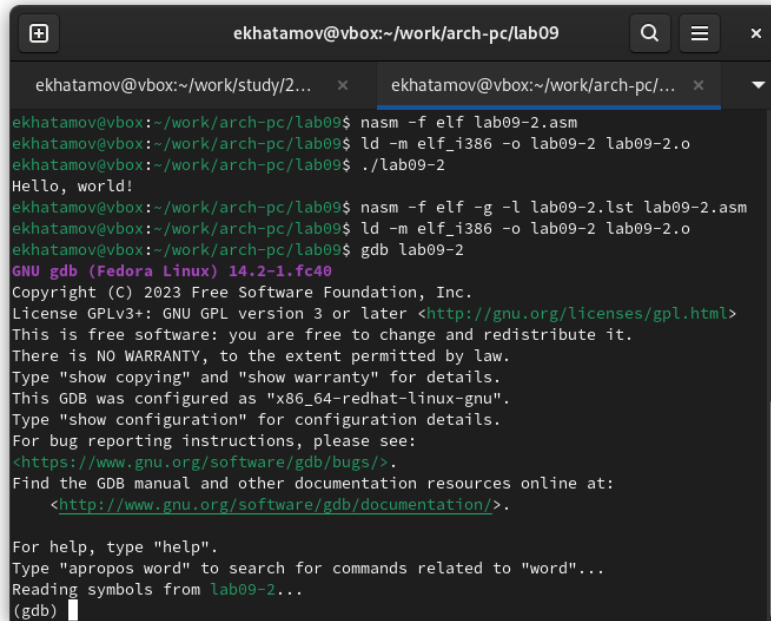
Рис. 4.5: Введения программы печати сообщения Hello world

Я создал исполняемый файл и запустил его

```
ekhatamov@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-2.asm
ekhatamov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
ekhatamov@vbox:~/work/arch-pc/lab09$ ./lab09-2
Hello, world!
ekhatamov@vbox:~/work/arch-pc/lab09$
```

Рис. 4.6: Создание исполняемого файла lab09-2.asm и запуск его

Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Загрузил исполняемый файл в отладчик gdb

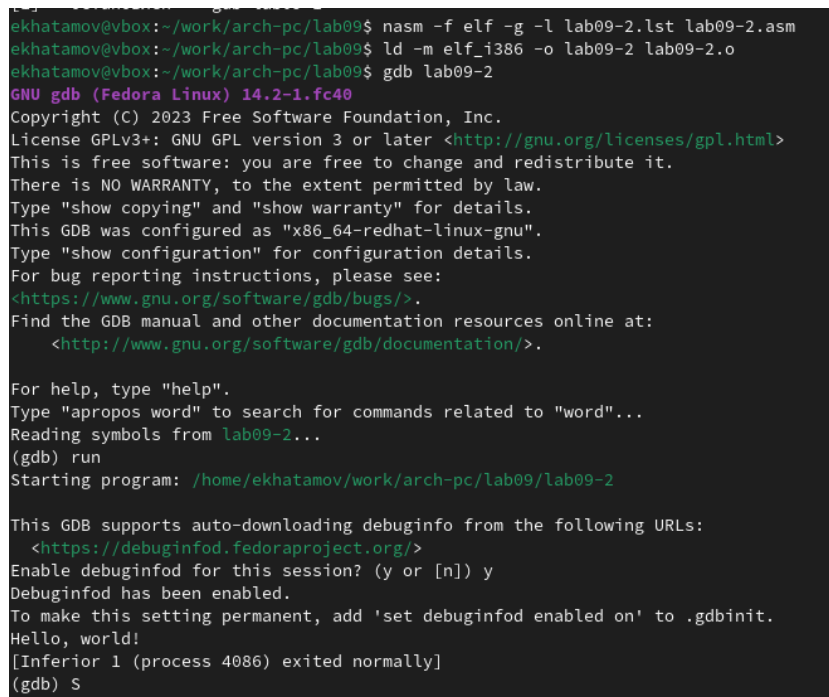


```
ekhatamov@vbox:~/work/arch-pc/lab09
ekhatamov@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-2.asm
ekhatamov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
ekhatamov@vbox:~/work/arch-pc/lab09$ ./lab09-2
Hello, world!
ekhatamov@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ekhatamov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
ekhatamov@vbox:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb)
```

Рис. 4.7: Загрузка исполняемого файла в отладчик gdb

Проверил работу программы, запустив ее в оболочке GDB с помощью команды
run



```
ekhatamov@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ekhatamov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
ekhatamov@vbox:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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-redhat-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/ekhatamov/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
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.
Hello, world!
[Inferior 1 (process 4086) exited normally]
(gdb) $
```

Рис. 4.8: проверил работу программы

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

```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 11.
(gdb) r
Starting program: /home/ekhatamov/work/arch-pc/lab09/lab09-2

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

Рис. 4.9: установка брейкпоинт на метку `_start`

Посмотрел дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`

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

Рис. 4.10: Просмотр дисассимилированный код программы

Переключился на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`

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

Рис. 4.11: Переключения на отображение команд с Intel'овским синтаксисом

Перечислил различия отображения синтаксиса машинных команд в режимах АТТ и Intel. Включил режим псевдографики для более удобного анализа программы

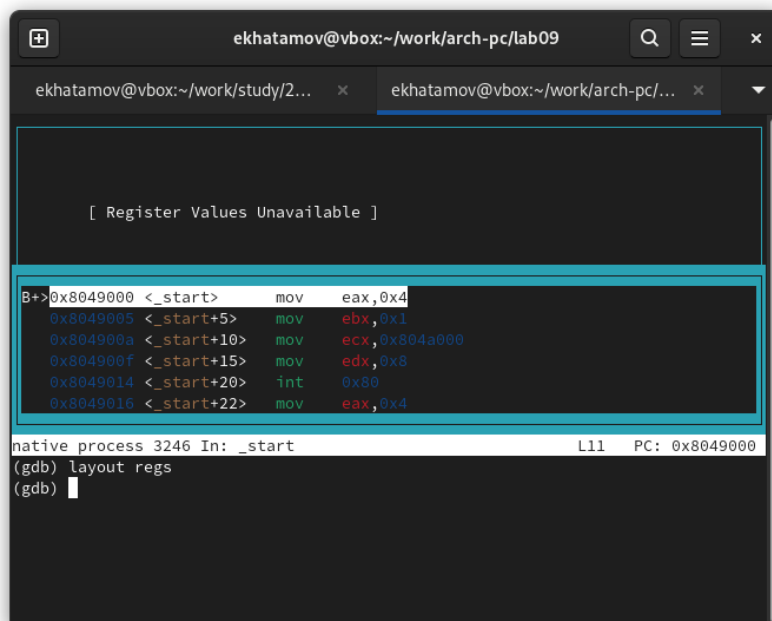
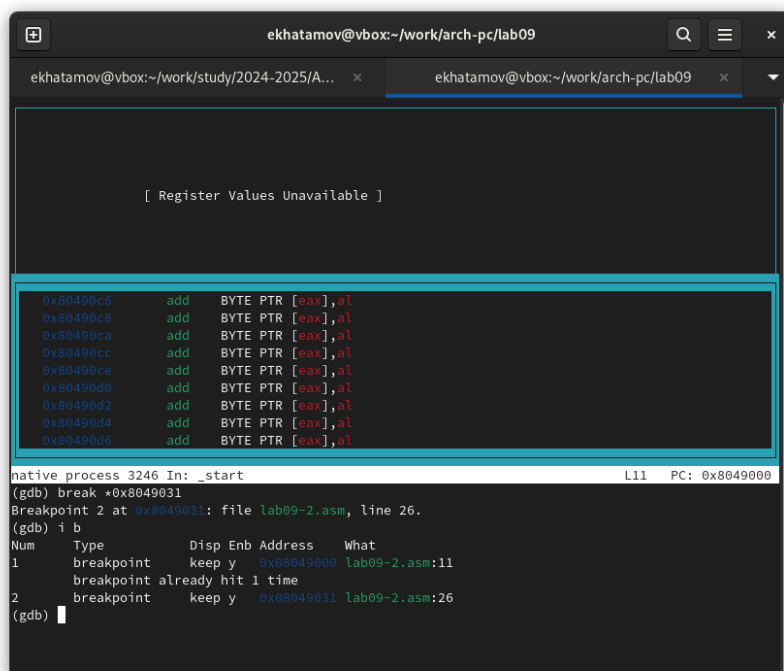


Рис. 4.12: режим псевдографики

Проверил установленные точки останова с помощью команды `info breakpoints`. Установил еще одну точку останова по адресу инструкции. Адрес инструкции смог увидеть в средней части экрана в левом столбце соответствующей инструк-

ции . Определил адрес предпоследней инструкции (mov ebx,0x0) и установил точку останова.потом посмотрел информацию о всех установленных точках останова:



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

```
native process 3246 In: _start L11 PC: 0x8049000
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 26.
(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint      keep y   0x8049000 lab09-2.asm:11
         breakpoint already hit 1 time
2        breakpoint      keep y   0x8049031 lab09-2.asm:26
(gdb)
```

Below the terminal output, a list of assembly instructions is displayed, each preceded by its address:

Address	Instruction
0x80490c6	add BYTE PTR [eax],al
0x80490c8	add BYTE PTR [eax],al
0x80490ca	add BYTE PTR [eax],al
0x80490cc	add BYTE PTR [eax],al
0x80490ce	add BYTE PTR [eax],al
0x80490d0	add BYTE PTR [eax],al
0x80490d2	add BYTE PTR [eax],al
0x80490d4	add BYTE PTR [eax],al
0x80490d6	add BYTE PTR [eax],al

Рис. 4.13: Установка и Проверка точек останова

4.3 Работа с данными программы в GDB

Для начала посмотрел содержимое регистров с помощью команды info registers

The screenshot shows a debugger window with the title bar "ekhatamov@vbox:~/work/arch-pc/lab09". The main window displays assembly code for a native process 3246. The assembly code is as follows:

```
0x80490c0 add BYTE PTR [eax],al
0x80490c8 add BYTE PTR [eax],al
0x80490ca add BYTE PTR [eax],al
0x80490cc add BYTE PTR [eax],al
0x80490ce add BYTE PTR [eax],al
0x80490d0 add BYTE PTR [eax],al
0x80490d2 add BYTE PTR [eax],al
0x80490d4 add BYTE PTR [eax],al
0x80490d6 add BYTE PTR [eax],al
```

Below the assembly code, the register values are displayed:

```
native process 3246 In: _start L11 PC: 0x8049000
eax 0x0 0
ecx 0x0 0
edx 0x0 0
ebx 0x0 0
esp 0xffffd020 0xffffd020
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x8049000 0x8049000 <_start>
eflags 0x202 [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.14: Просмотр содержимое регистров

С помощью команды `si` я посмотрел регистры и изменил их.

The screenshot shows a debugger window with the title bar "ekhatamov@vbox:~/work/arch-pc/lab09". The main window displays the register values for a native process 3658. The register values are as follows:

Register	Value
eax	0x4
ecx	0x0
edx	0x0
ebx	0x0
esp	0xffffd030
ebp	0x0
esi	0x0
edi	0x0
eip	0x8049005
cs	0x23
ds	0x2b
fs	0x0

Below the register values, the assembly code is displayed:

```
0x8049005 <_start> mov eax,ecx
0x8049007 <_start+10> mov ebx,edx
0x8049009 <_start+15> mov esi,esi
0x804900b <_start+20> int 0x0
0x804900d <_start+25> mov ecx,ecx
0x804900f <_start+30> mov ebx,ebx
0x8049011 <_start+35> mov esi,esi
0x8049013 <_start+40> int 0x0
```

The debugger window also shows the command prompt with the following text:

```
native process 3658 In: _start
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ekhatamov/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:11
(gdb) si
(gdb)
```

Рис. 4.15: Просмотр значение и изменения регистров

С помощью команды я посмотрел значение переменной `msg1`.

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

Рис. 4.16: Просмотр значение переменной msg1

Следом я посмотрел значение второй переменной msg2

```
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "wor!d!\n\034"
(gdb) █
```

Рис. 4.17: Просмотр значение переменной msg2

С помощью команды set я изменил значение переменной msg1.

```
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hh!llo, "
(gdb) █
```

Рис. 4.18: Изменения значения переменной msg1

Я изменил переменную msg2

```
0x804a008 <msg2>:      "wor!d!\n\034"
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb) █
```

Рис. 4.19: Изменения переменной msg2

Я вывел значение регистров ехх и еах.


```
(gdb) p/f $msg1
$1 = void
(gdb) p/s $eax
$2 = 4
(gdb) p/t $eax
$3 = 100
(gdb) p/c $ecx
$4 = 0 '\000'
(gdb) p/x $ecx
$5 = 0x0
(gdb) 
```

Рис. 4.20: Вывод значения регистров ecx и eax

Я изменил значение регистра ebx. Команда выводит два разных значения так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум, поэтому и значения разные.

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

Рис. 4.21: Изменения значения регистра ebx

Я завершил работу с файлов вышел

```
[Inferior 1 (process 3985) exited normally]
(gdb) 
```

Рис. 4.22: Выход

4.4 Обработка аргументов командной строки в GDB

Я скопировал файл lab9-2.asm и переименовал его. Я сделал это через МС так как по мне это удобнее всего

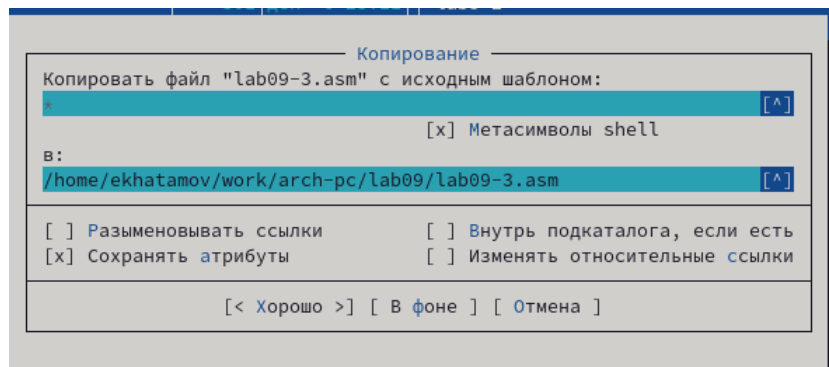


Рис. 4.23: Копирования файла

Запустил файл в отладчике и указал аргументы.

```
ekhatamov@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ekhatamov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
ekhatamov@vbox:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb)
```

Рис. 4.24: Запуск файла с аргументами

Поставил метку на _start и запустил файл

```
(gdb) b _start
Breakpoint 1 at 0x8049000: file lab09-3.asm, line 11.
(gdb) run
Starting program: /home/ekhatamov/work/arch-pc/lab09/lab09-3 аргумент1 аргумент
2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
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.

Breakpoint 1, _start () at lab09-3.asm:11
11      mov eax, 4
(gdb)
```

Рис. 4.25: Запуск файла через метку

Я проверил адрес вершины стека и убедился что там хранится 5 элементов.

```
(gdb) x/x $esp
0xffffcf90:      0x00000005
(gdb)
```

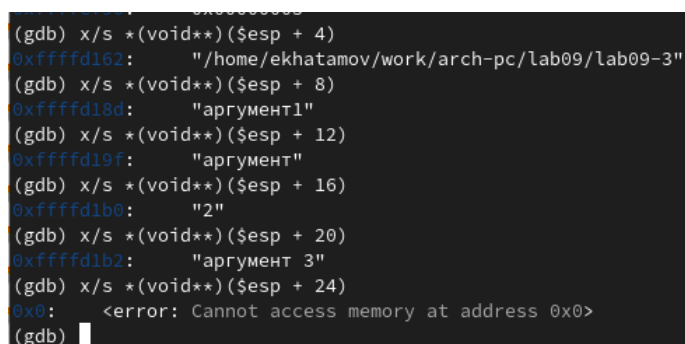
Рис. 4.26: Адрес вершины стека

Я посмотрел все позиции стека. По первому адресу хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации.

5 Самостоятельная работа

5.1 Первая задача

Я с начала копировал файл в котором делал самостоятельную работу 8-ой лабораторной работы. Я это сделал с помощью МС.



```
(gdb) x/s *(void**)($esp + 4)
0xffffd162:  "/home/ekhatamov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd18d:  "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd19f:  "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd1b0:  "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd1b2:  "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 5.1: Копирования файла ил lab08

Потом я преобразовал программу из лабораторной работы №8 и реализовал вычисления как подпрограмму.

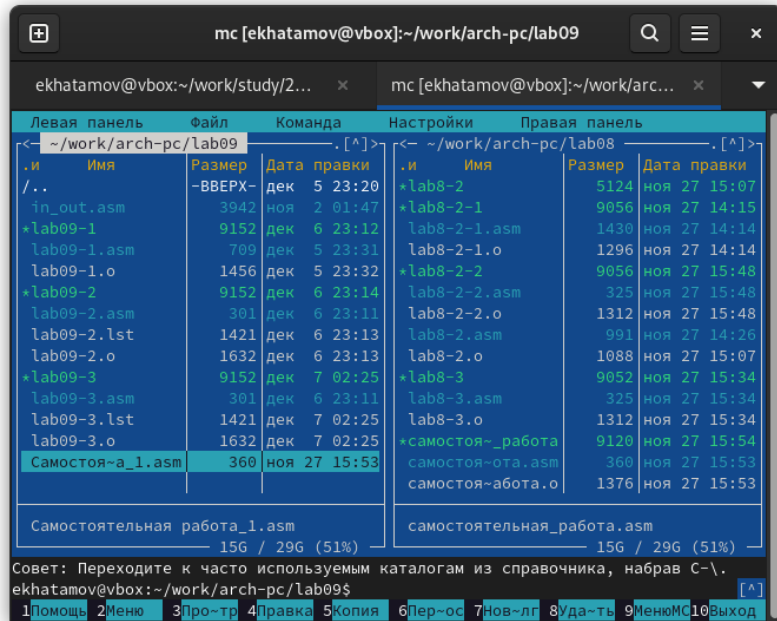


Рис. 5.2: Преобразования файла

5.2 Вторая задача

Я создал новый файл и ввел туда программу который был в задаче. После этого я сохдал исполняемый файл и запустил его чтоб увидеть в чем ошибка.

```
ekhatamov@vbox: ~/work/arch-pc/lab09$ nasm -f elf Самостоятельная_работа_2.asm
ekhatamov@vbox: ~/work/arch-pc/lab09$ ld -m elf_i386 -o Самостоятельная_работа_2
Самостоятельная_работа_2.o
ekhatamov@vbox: ~/work/arch-pc/lab09$ ./Самостоятельная_работа_2
Результат: 10
ekhatamov@vbox: ~/work/arch-pc/lab09$
```

Рис. 5.3: Запуск файла

После проявление ошибки я запустил программу в окладчике.

```
ekhatamov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ gdb -tui ./Самостоятельная_работа_2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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-redhat-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 Самостоятельная_работа_2...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file Самостоятельная_работа_2.asm, line 8.
(gdb) run
Starting program: /home/ekhatamov/work/arch-pc/lab09/Самостоятельная_работа_2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
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.

Breakpoint 1, _start () at Самостоятельная_работа_2.asm:8
8      mov ebx,3
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:    mov     ebx,0x3
      0x080490ed <+5>:    mov     eax,0x2
      0x080490f2 <+10>:   add     ebx,eax
      0x080490f4 <+12>:   mov     ecx,0x4
      0x080490f9 <+17>:   mul     ecx
      0x080490fb <+19>:   add     ebx,0x5
      0x080490fe <+22>:   mov     edi,ebx
      0x08049100 <+24>:   mov     eax,0x804a000
      0x08049105 <+29>:   call   0x804900f <sprint>
      0x0804910a <+34>:   mov     eax,edi
      0x0804910c <+36>:   call   0x8049086 <iprintLF>
      0x08049111 <+41>:   call   0x80490db <quit>
End of assembler dump.
```

Рис. 5.4: Запуск программы в окладчике

Я открыл регистры и проанализировал их, понял что некоторые регистры стоят не на своих местах и исправил это.

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

```
ekhatamov@vbox:~/work/arch-pc/lab09
ekhatamov@vbox:~/work/study/2... x ekhatamov@vbox:~/work/arch-pc/... x
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf80 0xffffcf80
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

0x80490e0 <quit+5>    mov     eax,0x1
0x80490e5 <quit+10>   int     0x80
0x80490e7 <quit+12>   ret
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

native process 5933 In: _start L8 PC: 0x80490e8
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ekhatamov/work/arch-pc/lab09/Самостоятельная_работа_2

Breakpoint 1, _start () at Самостоятельная_работа_2.asm:8
(gdb) █
```

Рис. 5.5: Исправления ошибок

После этого я занова создал исполняемый файл и запустил его чтобы проверить что все сделано правильно.

The screenshot shows a terminal window with the following commands and output:

```
ekhatamov@vbox:~/work/arch-pc/lab09$ nasm -f elf Самостоятельная_работа_2.asm
ekhatamov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o Самостоятельная_работа_2
Самостоятельная_работа_2.o
ekhatamov@vbox:~/work/arch-pc/lab09$ ./Самостоятельная_работа_2
Результат: 25
ekhatamov@vbox:~/work/arch-pc/lab09$
```

Рис. 5.6: Запуск исполняемого файла

6 Выводы

В результате выполнения данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм, а так же познакомился с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. https://esystem.rudn.ru/pluginfile.php/2089096/mod_resource/content/0/%D0%9B%D0%B0%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5%D0%BF%D0%BE%D0%B4%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA.pdf