Шаблон отчёта по лабораторной работе №9

Дисциплина: архитектура компьютера

Нурыева Гулсолтан

Группа : НКАбд-02-25

Содержание

1 Цель работы

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

2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программам с помощью GDB.
   2.1. Добавление точек останова.
   2.2. Работа с данными программы в GDB.
   2.3. Обработка аргументов командной строки в GDB.

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, отрабатывает, 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 до ноль но 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 программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. От ладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Одна код ля GDB существует несколько сторон них графических над строек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

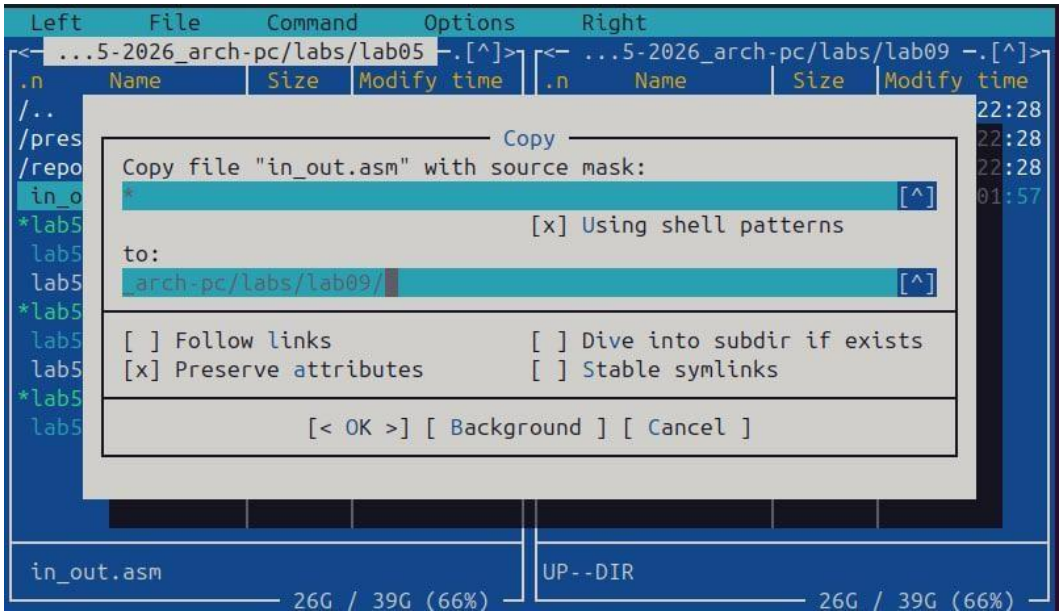
4 . Выполнение лабораторной работы

4.1. Реализация подпрограмм в NASM

Для начала я создала каталог для программ лабораторной работы, потом перешла в него и создала файл lab09-1.asm.

```
vboxuser@gulsoltan:~/work/arch-pc/lab09$ cd ~/work/study/2025-2026/"Архитектура компютера"/study_2025-2026_arh-pc/labs/lab09
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ touch lab09-1.asm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

Потом зашла на MC и через него скопировала файл in_out.asm в созданный каталог.



Пример: После этого я открыла созданной мною файл и ввела туда программу

```
.../Архитектура компютера/study_2025-2026_arch-pc/labs/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:
;-------------------------------------
; Основная программа
;-------------------------------------
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi

^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
```

Потом я создала исполняемый файл и запустила его.

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ nasm -f elf lab09-1.a
sm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ld -m elf_i386 -o lab
09-1 lab09-1.o
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ./lab09-1
Введите x: 5
2x+7=17
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

После этого я открыла созданной мною файл и исправила программу для вывода вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы _calcul.

```
.../Архитектура компютера/study_2025-2026_arch-pc/labs/lab09/lab09-1.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(x) = 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
                    [ Read 35 lines ]
^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
```

Потом я создала исполняемый файл и запустила его.

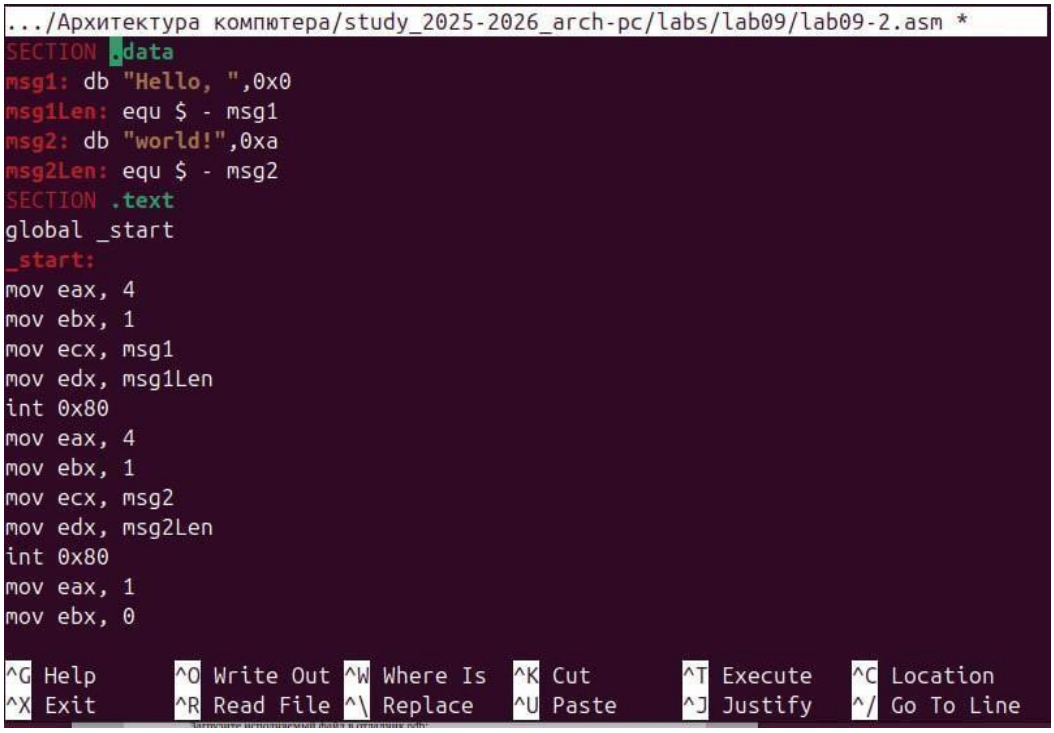
```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ nasm -f elf lab09-1.a
sm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ld -m elf_i386 -o lab
09-1 lab09-1.o
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ./lab09-1
Введите x: 5
f(x) = 2x + 7= 17
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

4.2. Отладка программам с помощью GDB.

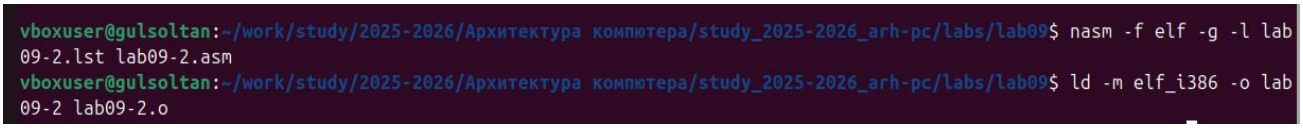
Потом я создала новый файл в том же каталоге lab09-2.asm.

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ touch lab09-2.asm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

После создания я открыла файл и ввела туда программу, которая выводит на экран сообщения Hello world!



Потом создала исполняемый файл и запустила его. И еще я проверила его работу.



Загрузила исполняемый файл в отладчик gdb:

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/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/vboxuser/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/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
```

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

```
(gdb) run
Starting program: /home/vboxuser/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/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 24442) exited normally]
```

Для более подробного анализа программы установила брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустила её:

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/vboxuser/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09/lab09-2

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

Посмотрела 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.
```

Переключилась на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel:

```
(gdb) set disassembly-flavor intel
(gdb) set disassemble _start
Ambiguous set command "disassemble _start": disassemble-next-line, disassembler-
options.
```

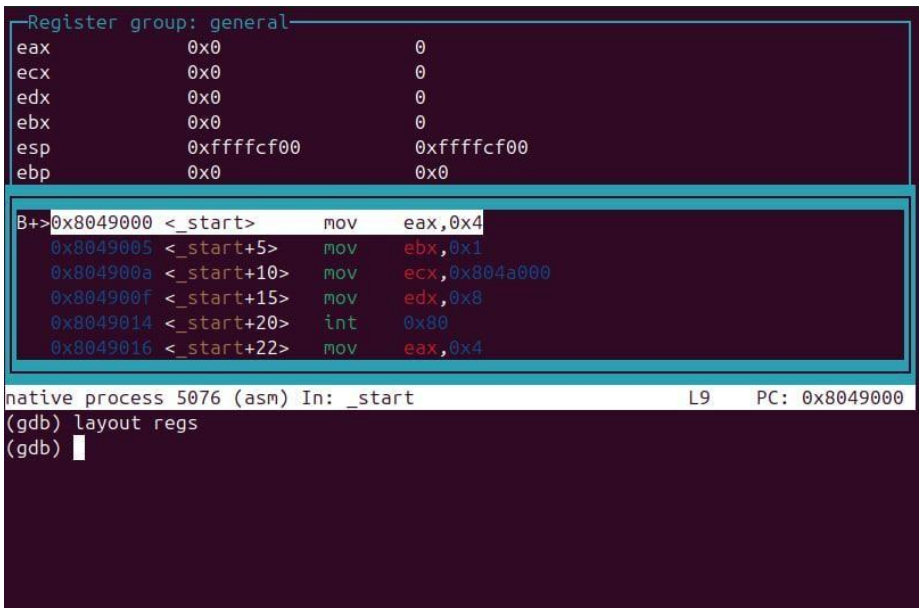
Перечислила различия отображения синтаксиса машинных команд в режимах ATT и Intel. Включила режим псевдографики для более удобного анализа программы:

(gdb) layout asm

(gdb) layout regs

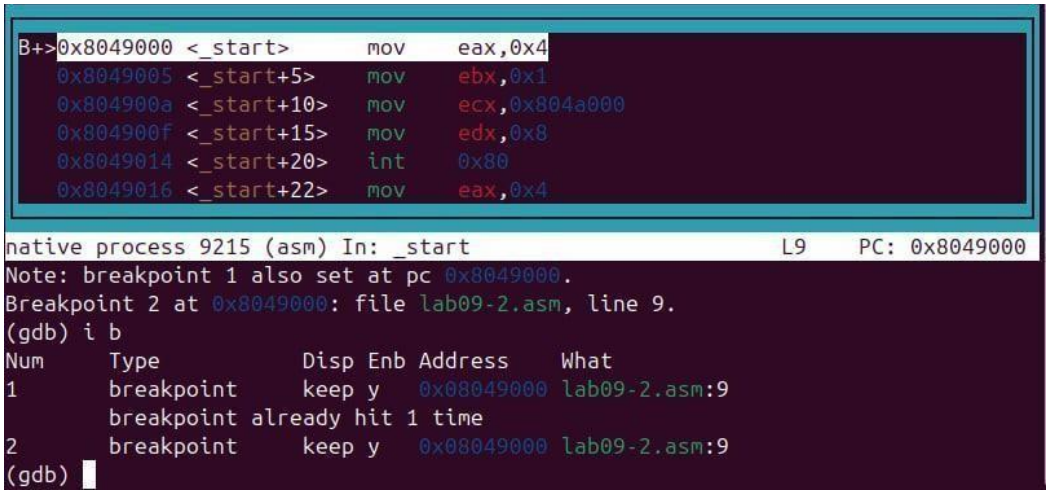
В этом режиме есть три окна:

- Вверхней части видны названия регистров и их текущие значения;
- В средней части виден результат дисассимилирования программы;
- Нижняя часть доступна для ввода команд



4.2.1. Добавление точек останова

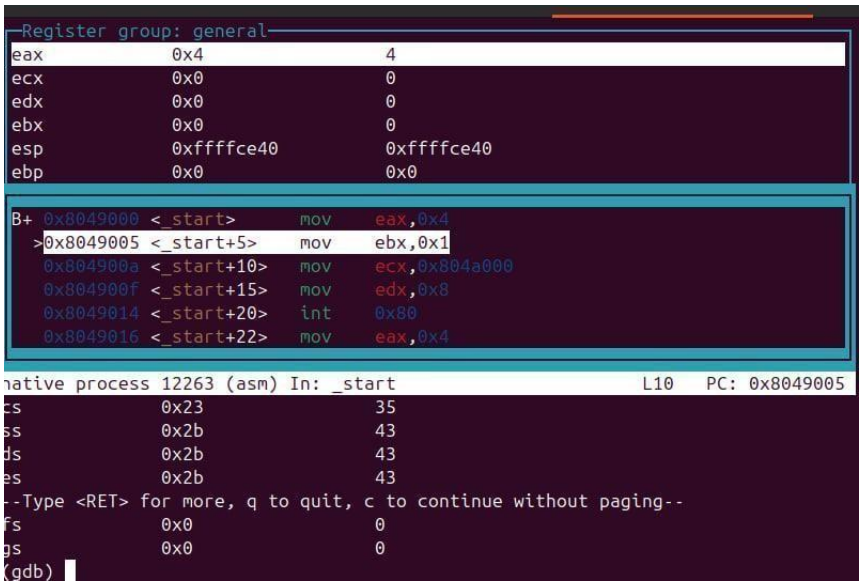
На предыдущих шагах была установлена точка останова по имени метки(_start).
Проверила это с помощью команды info breakpoints(кратко i b):



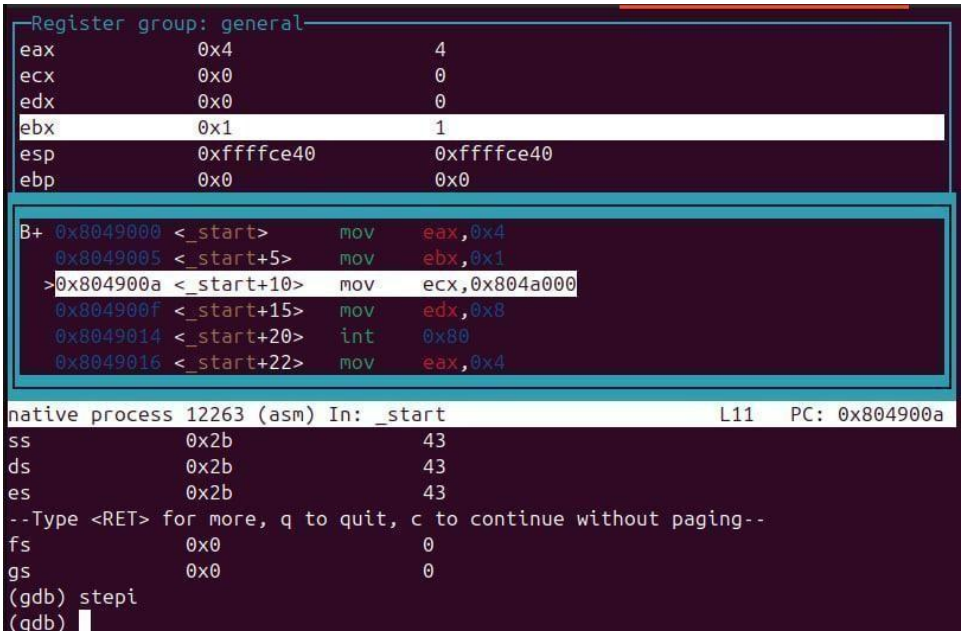
4.2.2. Работа с данными программы в GDB

Выполнила 5 инструкций с помощью команды stepi:

Stepi 1:



Stepi 2:



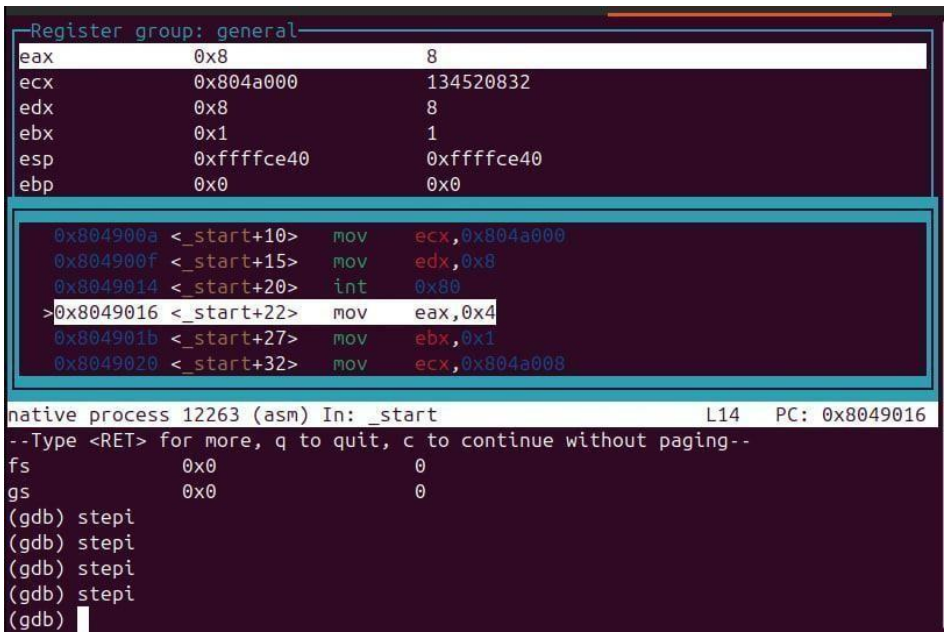
Stepi 3:

```
Register group: general
eax            0x4                4
ecx            0x804a000          134520832
edx            0x0                0
ebx            0x1                1
esp            0xffffce40         0xffffce40
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 12263 (asm) In: _start                          L12   PC: 0x804900f
ds             0x2b               43
es             0x2b               43
--Type <RET> for more, q to quit, c to continue without paging--
fs             0x0                0
gs             0x0                0
(gdb) stepi
(gdb) stepi
(gdb)
```

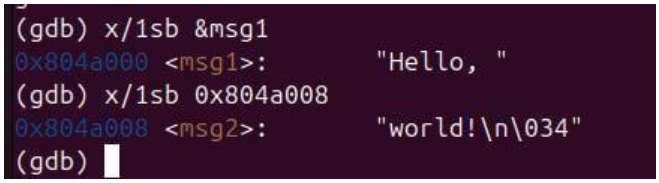
Stepi 4:

```
Register group: general
eax            0x4                4
ecx            0x804a000          134520832
edx            0x8                8
ebx            0x1                1
esp            0xffffce40         0xffffce40
ebp            0x0                0x0

   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

native process 12263 (asm) In: _start                          L13   PC: 0x8049014
es             0x2b               43
--Type <RET> for more, q to quit, c to continue without paging--
fs             0x0                0
gs             0x0                0
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)
```

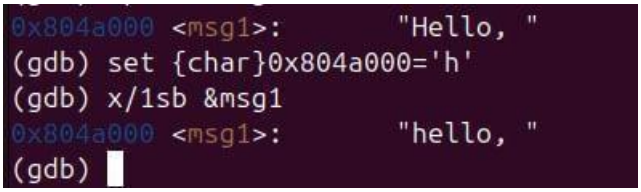
Stepi 5:



С помощью команды x &<имя переменной> также можно посмотреть содержимое переменной. Посмотрела значение переменной msg1 по имени:



Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес:



Чтобы посмотреть значения регистров используется команда print /F (перед именем регистра обязательно ставится префикс $:

С помощью команды set изменила значение регистра ebx:

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

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

4.2.3. Обработка аргументов командной строки в GDB

Для начала я создала каталог для программ лабораторной работы, потом перешла в него и создала файл lab09-3.asm.

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ touch lab09-3.asm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

После этого я скопировала файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки:

```
.../Архитектура компютера/study_2025-2026_arch-pc/labs/lab09/lab09-3.asm *
;-------------------------------------------------------------------
; Обработка аргументов командной строки
;-------------------------------------------------------------------
%include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего

^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
```

Потом я создала исполняемый файл и запустила его:

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ nasm -f elf -g -l lab
09-3.lst lab09-3.asm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ld -m elf_i386 -o lab
09-3 lab09-3.o
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

Для загрузки в gdb программы с аргументами необходимо использовать ключ--args. Загрузила исполняемый файл в отладчик, указав аргументы:

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ gdb --args lab09-3 as
hgabat moscow 'balkan'
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 lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
```

Для начала установим точку останова перед первой инструкцией в программе и запустим ее. Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки:

```
(gdb) b _start
Note: breakpoint 1 also set at pc 0x80490e8.
Breakpoint 2 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/vboxuser/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09/lab09-3 as
hgabat moscow balkan

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:8
8       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffcf00:     0x00000004
(gdb)
```

Посмотрела остальные позиции стека–по адресу [esp+4] располагается адрес в памяти где находиться имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12]–второго ит.д.

```
(gdb) x/s *(void**)($esp + 4)
0xffffd0c3:     "/home/vboxuser/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)($esp +8)
0xffffd13b:     "ashgabat"
(gdb) x/s *(void**)($esp + 12)
0xffffd144:     "moscow"
(gdb) x/s *(void**)($esp + 16)
0xffffd14b:     "balkan"
(gdb) x/s *(void**)($esp +20)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

В 32-битной архитектуре размер указателя — 4 байта. Поэтому адреса в стеке смещаются на 4 байта.

5. Самостоятельная работа

5.1. Преобразуйте программу излабораторной работы №8, реализовав вычисление значения функции $f(x)$ как подпрограмму.

Для начала я создала каталог для программ лабораторной работы, потом перешла в него и создала файл lab09-4.asm.

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ touch lab09-4.asm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

После этого я открыла созданной мною файл и ввела туда программу для вывода для вычисления функции $f(x)$:

```
%include 'in_out.asm'

SECTION .data
    msg db 'Введите x: ', 0
    res db 'Результат: ', 0
    x dd 0          ; переменная для хранения x

SECTION .text
GLOBAL _start

; ------------------------------------------------------------
; Подпрограмма вычисления f(x) = 10x - 5
; Вход:  EAX = x (целое число)
; Выход: EAX = f(x) = 10x - 5
; ------------------------------------------------------------
calc_fx:
    imul eax, 10     ; EAX = 10 * x
    add eax, 5       ; EAX = 5x - 5
    ret              ; возврат из подпрограммы

_start:
    ; --- Ввод значения x ---
    mov eax, msg     ; выводим приглашение
    call sprint

    mov ecx, x       ; адрес переменной x
    mov edx, 10      ; размер буфера
    call sread       ; читаем строку
                                [ Read 47 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

Потом я создала исполняемый файл и запустила его

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ nasm -f elf lab09-4.a
sm -o lab09-4.o
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ld -m elf_i386 -o lab
09-4 lab09-4.o
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ./lab09-4
Введите x: 3 4 2 1
Результат: 35
```

5.2. С помощью отладчика GDB, анализируя изменения значений регистров, определить ошибку и исправить ее.

Для начала я создала каталог для программ лабораторной работы, потом перешла в него и создала файл lab09-5.asm.

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ touch lab09-5.asm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

После этого я открыла созданной мною файл и ввела туда программу:

```
.../Архитектура компютера/study_2025-2026_arch-pc/labs/lab09/lab09-5.asm *
%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 ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

^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
```

Потом я создала исполняемый файл и запустила его.Программа дала ошибку:

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ nasm -f elf lab09-5.a
sm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ld -m elf_i386 -o lab
09-5 lab09-5.o
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ./lab09-5
Результат: 10
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

Потом исправила ошибку и ввела туда нужную программу:

```
.../Архитектура компютера/study_2025-2026_arch-pc/labs/lab09/lab09-5.asm *
%include 'in_out.asm'

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

SECTION .text
GLOBAL _start

_start:
    ; ---- Вычисление выражения (3+2)*4+5 ----
    mov ebx, 3      ; ebx = 3
    add ebx, 2      ; ebx = 5 (3+2)

    mov eax, ebx    ; eax = 5
    mov ecx, 4      ; ecx = 4
    mul ecx         ; eax = eax * ecx = 5 * 4 = 20

    add eax, 5      ; eax = 20 + 5 = 25
    mov edi, eax    ; сохраняем результат в edi

^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
```

Создала испольняемый файл и запустила его еще раз, но уже измененного:

```
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ nasm -f elf lab09-5.a
sm
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ld -m elf_i386 -o lab
09-5 lab09-5.o
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$ ./lab09-5
Результат: 25
vboxuser@gulsoltan:~/work/study/2025-2026/Архитектура компютера/study_2025-2026_arh-pc/labs/lab09$
```

Выводы

В ходе лабораторной работы:

1. Освоены принципы написания и использования подпрограмм в ассемблере NASM.
2. Приобретены навыки работы с отладчиком GDB:

- Установка и управление точками останова ○ Пошаговое выполнение программы ○ Просмотр и изменение регистров и памяти
- Анализ стека и аргументов командной строки

3. Выполнены задания по отладке и исправлению ошибок в программах.

Работа способствовала углублённому пониманию механизмов вызова подпрограмм и методов отладки низкоуровневого кода.

Ответы на вопросы :

1. **call** и **ret**
2. **call** сохраняет адрес возврата в стеке и передаёт управление подпрограмме
3. Стек используется для хранения адреса возврата и локальных данных
4. Операнд в **ret** указывает, сколько байт удалить из стека после возврата
5. Отладчик позволяет анализировать и контролировать выполнение программы
6. Отладочная информация включает номера строк и символы. Компиляция с ключом -g
7. **Breakpoint** — остановка в определённой точке, **watchpoint** — остановка при изменении переменной, **call stack** — стек вызовов
8. Основные команды GDB: run, break, step, next, print, info, x