

Отчет по лабораторной работе №9

Дисциплина: Архитектура компьютера

Орлов Илья Сергеевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Релаксация подпрограмм в NASM	8
4.1.1	Отладка программ с помощью GDB	10
4.1.2	Добавление точек останова	13
4.1.3	Работа с данными программы в GDB	14
4.1.4	Обработка аргументов командной строки в GDB	16
4.2	Задание для самостоятельной работы	17
5	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	Создание рабочего каталога	8
4.2	Запуск программы из листинга	8
4.3	Изменение программы первого листинга	8
4.4	Запуск программы в отладчике	11
4.5	Проверка программы отладчиком	11
4.6	Запуск отладчика с брейкпойнтом	12
4.7	Дисассимилирование программы	13
4.8	Режим псевдографики	13
4.9	Список брейкпойнтов	14
4.10	Добавление второй точки останова	14
4.11	Просмотр содержимого регистров	15
4.12	Просмотр содержимого переменных двумя способами	15
4.13	Изменение содержимого переменных двумя способами	15
4.14	Просмотр значения регистра разными представлениями	16
4.15	Примеры использования команды set	16
4.16	Подготовка новой программы	16
4.17	Проверка работы стека	17
4.18	Измененная программа предыдущей лабораторной работы	17
4.19	Поиск ошибки в программе через пошаговую отладку	19
4.20	Проверка корректировок в программе	19

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. Самостоятельное выполнение заданий по материалам лабораторной 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, 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 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 программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

4 Выполнение лабораторной работы

4.1 Релазиация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9 (рис. -fig. 4.1).

```
rutnixya@rutnix-VirtualBox:~$ mkdir ~/work/arch-pc/lab09
rutnixya@rutnix-VirtualBox:~$ cd ~/work/arch-pc/lab09
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ touch lab09-1.asm
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ ls
lab09-1.asm
```

Рис. 4.1: Создание рабочего каталога

Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. -fig. 4.2).

```
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2x+7=27
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 4.2: Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения $f(g(x))$ (рис. -fig. 4.3).

```
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
f(g(x))=65
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 4.3: Изменение программы первого листинга

Код программы:


```

#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x))=',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 "f(g(x))"
_calcul:
    ; Call the sub-calculation subroutine
    call _subcalcul
    ;  $f(x) = 2x + 7$ 
    mov ebx, 2
    mul ebx
    add eax, 7
    mov [res], eax
    ret

;-----
; Подпрограмма вычисления  $g(x) = 3x - 1$ 
_subcalcul:
    ;  $g(x) = 3x - 1$ 
    mov ebx, 3
    mul ebx
    sub eax, 1
    ret

```

4.1.1 Отладка программ с помощью GDB

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике (рис. -fig. 4.4).

```

rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm

rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o

rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab09$ gdb lab9-2

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 lab9-2...
(gdb)

```

Рис. 4.4: Запуск программы в отладчике

Запустив программу командой `gdb`, я убедился в том, что она работает исправно (рис. -fig. 4.5).

```

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 lab9-2...
(gdb) run
Starting program: /home/rutnixya/work/arch-pc/lab09/lab9-2

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 6921) exited normally]
(gdb)

```

Рис. 4.5: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку `_start` и снова запускаю отладку (рис. -fig. 4.6).

```
rutnixya@rutnix-VirtualBox: ~/work/arch-pc/lab09
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 lab9-2...
(gdb) run
Starting program: /home/rutnixya/work/arch-pc/lab09/lab9-2

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 0xf7ffc800
Hello, world!

[Inferior 1 (process 6921) exited normally]
(gdb) break _start
Breakpoint 1 at 0x0049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/rutnixya/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel (рис. -fig. 4.7).

Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ах, еах, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

```

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x00049000 <+0>: mov     $0x1,%eax
0x00049005 <+5>: mov     $0x1,%ebx
0x0004900a <+10>: mov     $0x004a000,%ecx
0x0004900f <+15>: mov     $0x0,%edx
0x00049014 <+20>: int     $0x0
0x00049016 <+22>: mov     $0x1,%eax
0x0004901b <+27>: mov     $0x1,%ebx
0x00049020 <+32>: mov     $0x004a000,%ecx
0x00049025 <+37>: mov     $0x1,%edx
0x0004902a <+42>: int     $0x0
0x0004902c <+44>: mov     $0x1,%eax
0x00049031 <+49>: mov     $0x0,%ebx
0x00049036 <+54>: int     $0x0
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x00049000 <+0>: mov     eax,%eax
0x00049005 <+5>: mov     ebx,%ebx
0x0004900a <+10>: mov     ecx,%ecx
0x0004900f <+15>: mov     edx,%edx
0x00049014 <+20>: int     $0x0
0x00049016 <+22>: mov     eax,%eax
0x0004901b <+27>: mov     ebx,%ebx
0x00049020 <+32>: mov     ecx,%ecx
0x00049025 <+37>: mov     edx,%edx
0x0004902a <+42>: int     $0x0
0x0004902c <+44>: mov     eax,%eax
0x00049031 <+49>: mov     ebx,%ebx
0x00049036 <+54>: int     $0x0
End of assembler dump.
(gdb)

```

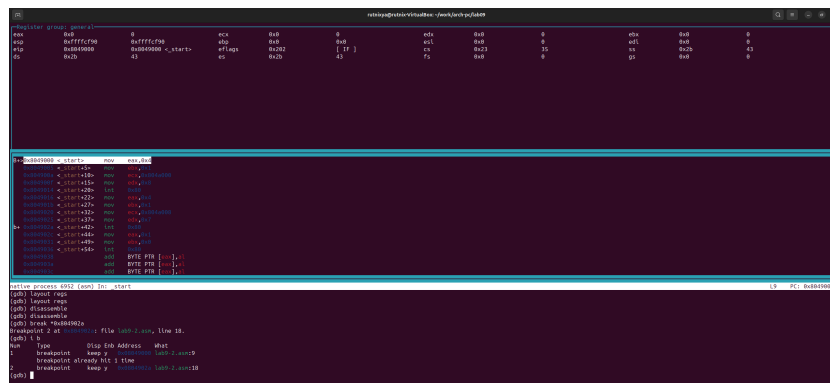
Рис. 4.7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. -fig. 4.8).

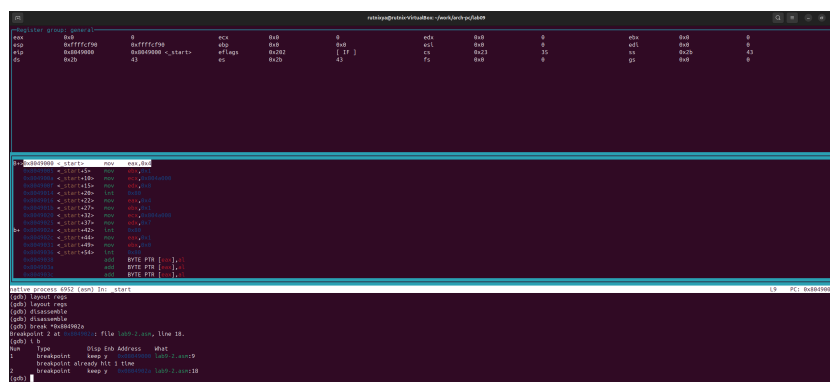
Рис. 4.8: Режим псевдографики

4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. -fig. 4.9).



Установлюю еще одну точку останова по адресу инструкции (рис. -fig. 4.10).



4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой info registers (рис. -fig. 4.11).

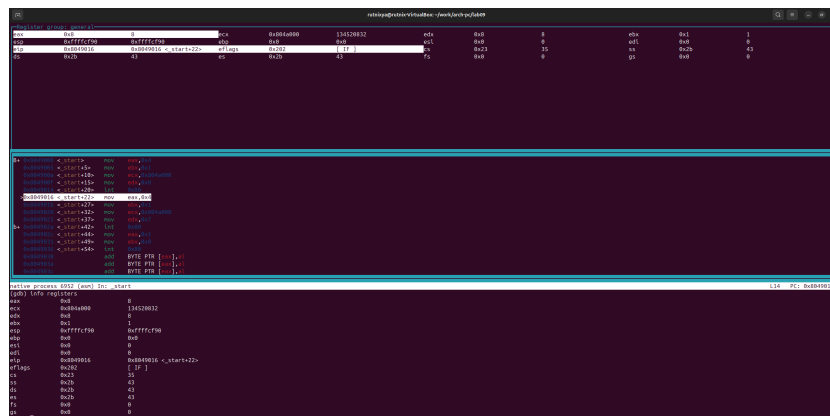


Рис. 4.11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -fig. 4.12).

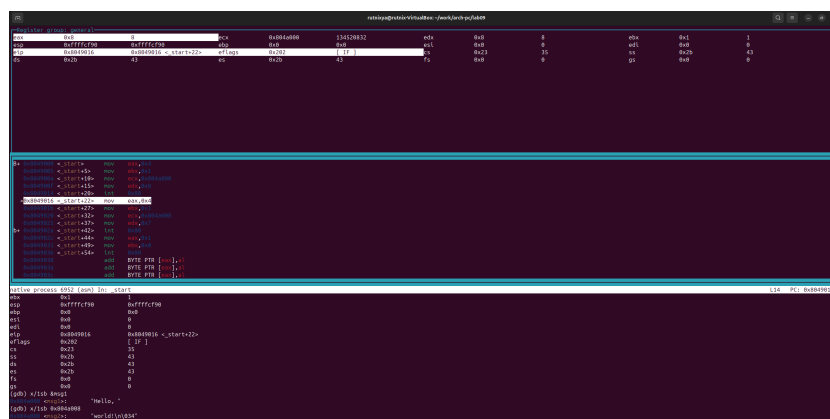


Рис. 4.12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. -fig. 4.13).

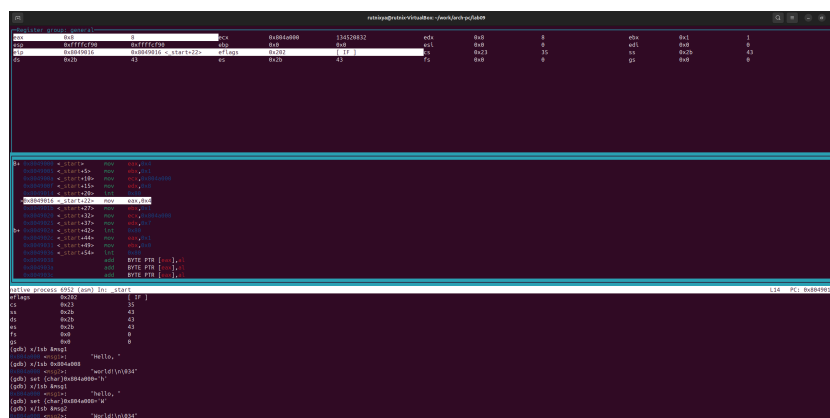


Рис. 4.13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра `edx` (рис. -fig. 4.14).

```
(gdb) print/x $edx
$1 = 0x8
(gdb) print/t $edx
$2 = 1000
(gdb) print/d $edx
$3 = 8
(gdb) print/c $edx
$4 = 8 '\b'
(gdb)
```

Рис. 4.14: Просмотр значения регистра разными представлениями

С помощью команды `set` меняю содержимое регистра `ebx` (рис. -fig. 4.15).

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

Рис. 4.15: Примеры использования команды `set`

4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. -fig. 4.16).

```
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab08$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab08$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab9-3 lab9-3.o
rutnixya@rutnix-VirtualBox:~/work/arch-pc/lab08$
```

Рис. 4.16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпоинт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра `esp` на `+4`, число обусловлено разрядностью системы, а указатель `void` занимает как раз 4 байта, ошибка при аргументе `+24` означает, что аргументы на вход программы закончились. (рис. -fig. 4.17).


```

rutnixya@rutnix-VirtualBox: /work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
rutnixya@rutnix-VirtualBox: /work/arch-pc/lab09$ ld -n elf -l386 -o lab9-3 lab9-3.o
rutnixya@rutnix-VirtualBox: /work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент2 аргумент3

GNU gdb (Ubuntu 15.0.50.20240403-Debian) 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:
  <http://www.gnu.org/software/gdb/bugs.html>
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 00004000: File lab9-3.asm, line 5.
(gdb) run
Starting program: /home/rutnixya/work/arch-pc/lab09/lab9-3 аргумент1 аргумент2 аргумент3

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 lib0 at 0x7ffffc0000

Breakpoint 1, _start () at lab9-3.asm:5
00004000: 00000000
(gdb) x/x $esp
00004000: 00000000
(gdb) x/s *(void**)(esp + 4)
00004004: "/home/rutnixya/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
00004008: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0000400c: "аргумент2"
(gdb) x/s *(void**)(esp + 16)
00004010: "3"
(gdb) x/s *(void**)(esp + 20)
00004014: "аргумент3"
(gdb) x/s *(void**)(esp + 24)
00004018: 
<error: Cannot access memory at address 0xb>
(gdb)

```

Рис. 4.17: Проверка работы стека

4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. -fig. 4.18).

```

GNU nano 7.2
#include "in_out.asm"

msg_func db "Функция: 3(x + 2)", 0xA, 0
msg_result db "Результат: ", 0

SECTION .text
c0000_start
calculate_function:
    add eax, 2
    mov ebx, 3
    mul ebx
    ret

_start:
    mov eax, msg_func
    call sprintf
    pop ecx
    mov esi, 0
next:
    cmp ecx, 1
    jle _end
    pop eax
    call atoi
    call calculate_function
    add esi, eax
    dec ecx
    jmp next
_end:
    mov eax, msg_result
    call sprintf
    mov eax, esi
    call sprintf
    call quit

```

Рис. 4.18: Измененная программа предыдущей лабораторной работы

Код программы:

```

#include 'in_out.asm'

SECTION .data

```

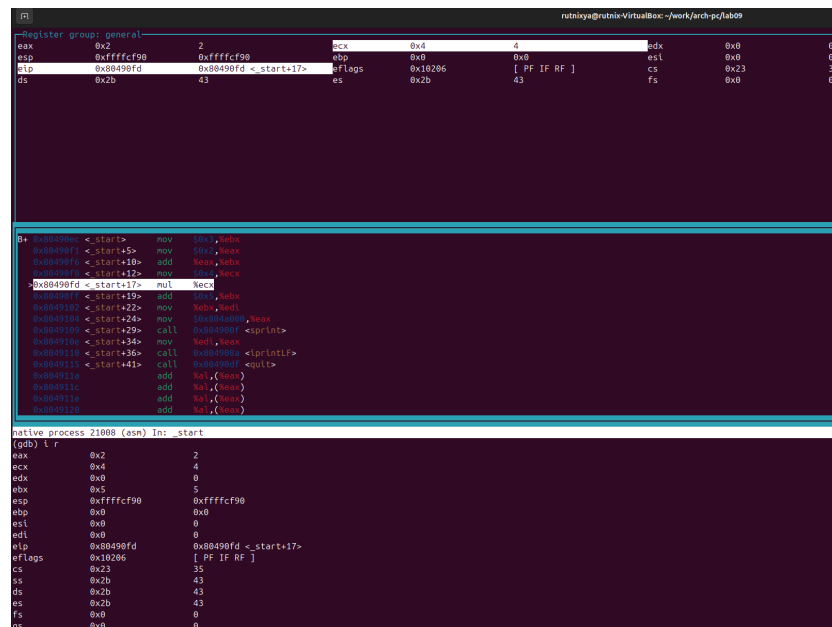
```

msg_func db "Функция: 3(x + 2)", 0xA, 0
msg_result db "Результат: ", 0
SECTION .text
GLOBAL _start
calculate_function:
add eax, 2
mov ebx, 3
mul ebx
ret
_start:
mov eax, msg_func
call sprintf
pop ecx
pop edx
mov esi, 0
next:
cmp ecx, 1
jle _end
pop eax
call atoi
call calculate_function
add esi, eax
dec ecx
jmp next
_end:
mov eax, msg_result
call sprintf
mov eax, esi
call iprintLF

```

call quit

2. Запускаю программу в режиме отладчика и пошагово через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul esx можно заметить, что результат умножения записывается в регистр eax, но также меняет и edx. Значение регистра ebx не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию (рис. -fig. 4.19).



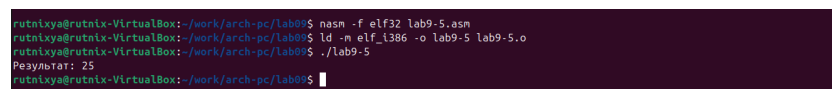
```
Register group: general
eax 0x2 2 ecx 0x4 4 edx 0x0 0
esp 0xffffcf90 0xffffcf90 ebp 0x0 0 esi 0x0 0
eip 0x00490fd 0x00490fd <start+17> eflags 0x10206 [ PF IF RF ] cs 0x23 35
ds 0x2b 43 es 0x2b 43 fs 0x0 0
gs 0x0 0

0: 0x00490ec <start> mov esi, ebx
0x00490ef <start+5> mov ebx, eax
0x00490f0 <start+10> add ebx, ebx
0x00490f2 <start+12> mov esi, ebx
0x00490fd <start+17> mul esx
0x00490ff <start+19> add esi, ebx
0x0049100 <start+22> mov ebx, edi
0x0049102 <start+24> mov esi, ebx
0x0049105 <start+29> call 0x0049000 <start+29>
0x0049108 <start+34> mov ebx, ebx
0x0049110 <start+36> call 0x0049000 <start+41>
0x0049112 <start+41> call 0x0049000 <start+41>
0x0049114 add eax, (eax)
0x0049116 add ebx, (eax)
0x0049118 add esi, (eax)
0x0049120 add eax, (eax)

native process 21008 (asm) In: .start
(gdb) i r
eax 0x2 2
ecx 0x4 4
edx 0x0 0
ebx 0x5 5
esp 0xffffcf90 0xffffcf90
ebp 0x0 0
esi 0x0 0
edi 0x0 0
eip 0x00490fd 0x00490fd <start+17>
eflags 0x10206 [ PF IF RF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
gs 0x0 0
```

Рис. 4.19: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -fig. 4.20).



```
rutnixya@rutnix-VirtualBox: ~/work/arch-pc/lab09$ nasm -f elf32 lab9-5.asm
rutnixya@rutnix-VirtualBox: ~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
rutnixya@rutnix-VirtualBox: ~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
rutnixya@rutnix-VirtualBox: ~/work/arch-pc/lab09$
```

Рис. 4.20: Проверка корректировок в программе

Код измененной программы:

```
%include 'in_out.asm'

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

SECTION .text
GLOBAL _start
_start:

mov ebx, 3
mov eax, 2
add ebx, eax
mov eax, ebx
mov ecx, 4
mul ecx
add eax, 5
mov edi, eax

mov eax, div
call sprint
mov eax, edi
call iprintLF

call quit
```

5 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. Курс на ТУИС
2. Лабораторная работа №9
3. Программирование на языке ассемблера NASM Столяров А. В.