

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

**Факультет физико-математических и естественных наук
Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9**

дисциплина: Архитектура компьютера

Студент: Ким Илья 1032253496

Группа: НКАбд-03-25

МОСКВА

2025 г.

Содержание

1. Цель работы.....	4
2. Задание.....	5
3. Теоретическое введение.....	6
4. Выполнение лабораторной работы.....	7
4.1. Реализация программ в NASM.....	7
4.1.1. Отладки программ с помощью GDB.....	8
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.....	17
4.2. Задание для самостоятельной работы.....	18
5. Выводы.....	23
6. Список литературы.....	24

Список иллюстраций

4.1. Создание рабочего каталога.....	7
4.2. Запуск программы из листинга.....	7
4.3. Изменение программы первого листинга.....	7
4.4. Запуск программы в отладчике.....	9
4.5. Проверка программы отладчиком.....	10
4.6. Запуск отладчика с брейкпойнтом.....	11
4.7. Дисассимилирование программы.....	12
4.8. Режим псевдографики.....	13
4.9. Список брейкпойнтов.....	13
4.10. Добавление второй точки останова.....	14
4.11. Просмотр содержимого регистров.....	15
4.12. Просмотр содержимого переменных двумя способами.....	16
4.13. Изменение содержимого переменных двумя способами.....	16
4.14. Просмотр значения регистра разными представлениями.....	16
4.15. Примеры использования команды set.....	17
4.16. Подготовка новой программы.....	17
4.17. Проверка работы стека.....	18
4.18. Измененная программа предыдущей лабораторной работы.....	19
4.19. Поиск ошибки в программе через пошаговую отладку.....	21
4.20. Проверка корректировок в программе.....	21

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 с 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).

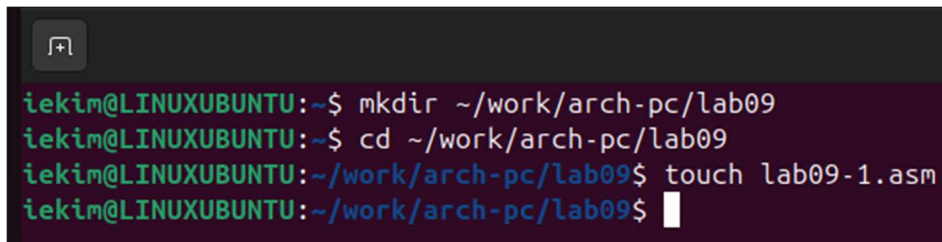
Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

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 программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

4. Выполнение лабораторной работы

4.1. Реализация подпрограмм в NASM

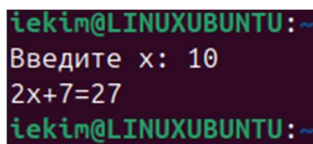
Создаю каталог для выполнения лабораторной работы №9 (рис.4.1).



```
iekim@LINUXUBUNTU:~$ mkdir ~/work/arch-pc/lab09
iekim@LINUXUBUNTU:~$ cd ~/work/arch-pc/lab09
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ touch lab09-1.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание рабочего каталога

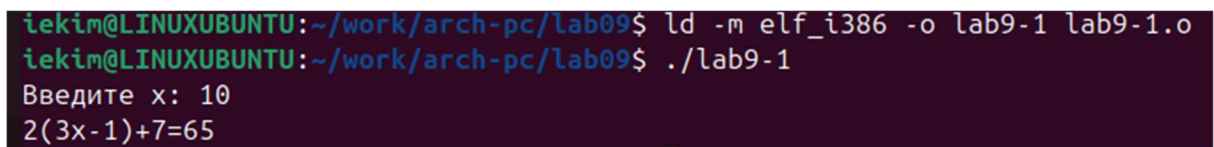
Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис.4.2).



```
iekim@LINUXUBUNTU:~$
Введите x: 10
2x+7=27
iekim@LINUXUBUNTU:~$
```

Рис. 4.2: Запуск программы из листинга

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



```
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2(3x-1)+7=65
```

Рис. 4.3: Изменение программы первого листинга

Код программы:

```
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ', 0
result: DB '2(3x-1)+7=', 0

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:
```

```
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
_calcul:
push eax
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
pop eax
ret
_subcalcul:
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.4.4).

```
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
iekim@LINUXUBUNTU:~/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"...
lab9-2: Нет такого файла или каталога.
(gdb) █
```

Рис. 4.4: Запуск программы в отладчике

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

```

iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ d -m elf_i386 -o lab9-2 lab9-2.o
d: команда не найдена
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
iekim@LINUXUBUNTU:~/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) run
Starting program: /home/iekim/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.
Hello, world!

Program received signal SIGSEGV, Segmentation fault.
0x08049036 in ?? ()
(gdb) █

```

Рис. 4.5: Проверка программы отладчиком

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

```

iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
iekim@LINUXUBUNTU:~/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) run
Starting program: /home/iekim/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.
Hello, world!
[Inferior 1 (process 8805) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 13.
(gdb) run
Starting program: /home/iekim/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:13
13      mov eax, 4
(gdb)

```

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel *amd топчик* (рис. 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 ax, eax, непосредственные операнды пишутся напрямую), именах регистров (АТТ имена регистров предваряются символом %, Intel – имена регистров пишутся без префиксов).

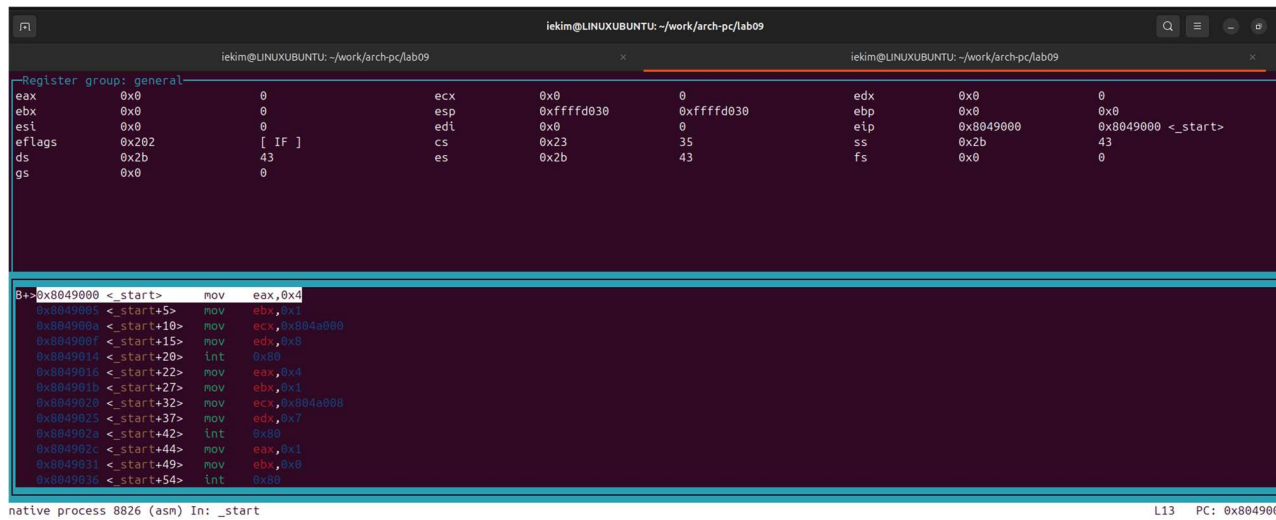
```

(gdb) disassemble _start
Undefined command: "disassemble". Try "help".
(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) set disassembly-flavor intel
No symbol "disassembly" in current context.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Undefined command: "disassembly". Try "help".
(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. 4.7: Дисассимилирование программы

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



```
Register group: general
eax      0x0      0      ecx      0x0      0      edx      0x0      0
ebx      0x0      0      esp      0xffffd030 0xffffd030  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0      eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]  cs      0x23      35      ss      0x2b      43
ds       0x2b     43      es      0x2b      43      fs       0x0      0
gs       0x0      0

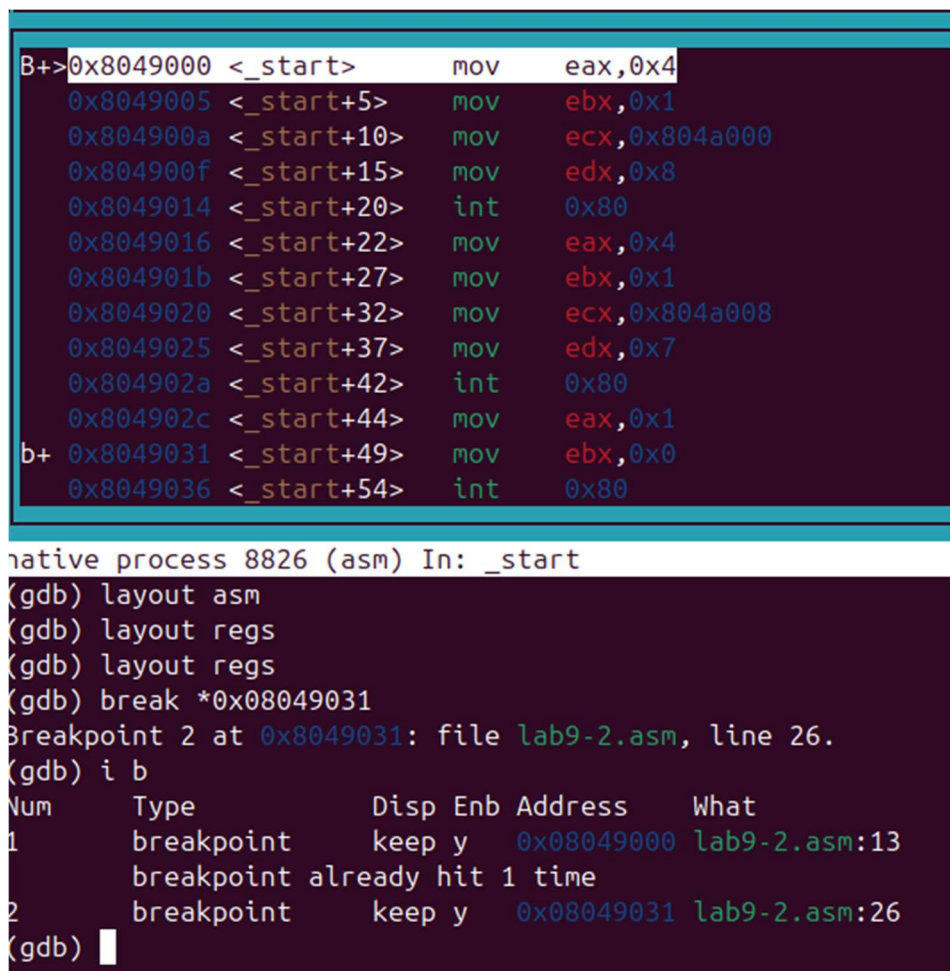
B+>0x8049000 <_start> mov    eax,0x4
0x8049005 <_start+5> mov    ebx,0x1
0x804900a <_start+10> mov    ecx,0x804a000
0x804900f <_start+15> mov    edx,0x8
0x8049014 <_start+20> int    0x80
0x8049016 <_start+22> mov    eax,0x4
0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a008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

native process 8826 (asm) In: _start
```

Рис. 4.8: Режим псевдографики

4.1.2. Добавление точек останова

Проверяю в режиме псевдографики, чтобы брейкпоинт сохранился (рис.4.9)



```
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
b+ 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80

native process 8826 (asm) In: _start
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x08049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 26.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000  lab9-2.asm:13
         breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031  lab9-2.asm:26
(gdb)
```

Рис. 4.9: Список брейкпоинтов

Устанавливаю ещё одну точку останова по адресу инструкции (рис.4.10).

```
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
b+ 0x8049031 <_start+49>     mov     ebx,0x0
0x8049036 <_start+54>     int     0x80

native process 8826 (asm) In: _start
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x08049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 26.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000 lab9-2.asm:13
         breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031 lab9-2.asm:26
(gdb) █
```

Рис. 4.10: Добавление второй точки останова

4.1.3 Работа с данными программы в GDB

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

```

ebx      0x0      0      esp      0xffffd030
esi      0x0      0      edi      0x0
eflags   0x202    [ IF ]   cs      0x23
ds       0x2b     43      es      0x2b
gs       0x0      0

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
b+ 0x8049031 <_start+49> mov     ebx,0x0

native process 8826 (asm) In: _start
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd030 0xffffd030
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ds       0x2b     43
ss       0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.11: Просмотр содержимого регистров

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

```

native process 8826 (asm) In: _start
edi            0x0                0
eip            0x8049000          0x8049000 <_start>
eflags        0x202              [ IF ]
cs             0x23              35
ss             0x2b              43
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) x/lsb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/lsb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) █

```

Рис. 4.12: Просмотр содержимого переменных двумя способами

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

```

(gdb) x/lsb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) set {char}&msg2='x'
(gdb) x/lsb &msg2
0x804a008 <msg2>:      "xorld!\n\034"
(gdb) set {char}&msg1='h'
(gdb) set {char}&msg1='h'
(gdb) x/lsb $msg1
Value can't be converted to integer.
(gdb) x/lsb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/lsb &msg2
0x804a008 <msg2>:      "xorld!\n\034"
(gdb)

```

Рис. 4.13: Изменение содержимого переменных двумя способами

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

```

(gdb) p/t $edx
$3 = 0
(gdb) p/t $edx
$4 = 0
(gdb) p/x $edx
$5 = 0x0

```

Рис. 4.14: Просмотр значения регистра разными представлениями

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

```
native process 8826 (asm) In: _start
(gdb) p/t $edx
$3 = 0
(gdb) p/t $edx
$4 = 0
(gdb) p/x $edx
$5 = 0x0
(gdb) set $ebx='2'
(gdb) p/s
$6 = 0
(gdb) p/s $ebx
$7 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$8 = 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.4.16).

```
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
Команда «nasm» не найдена. Возможно, вы имели в виду:
  команда 'ams' из deb-пакета ams (2.2.1-1)
  команда 'nasm' из deb-пакета nasm (2.16.01-1)
  команда 'nasm' из deb-пакета nasm (1.216-2)
  команда 'nasm' из deb-пакета nasm (1.15-6)
Попробуйте: sudo apt install <имя_deb-пакета>
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$
```

Рис. 4.16: Подготовка новой программы

Запускаю программу в 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.4.17).

```

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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490f5: file lab9-3.asm, line 7.
(gdb) run
Starting program: /home/iekim/work/arch-pc/lab09/lab9-3

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 lab9-3.asm:7
7      pop ecx
(gdb) x/s *(void**)($esp + 4)
0xffffd1ef: "/home/iekim/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)($esp + 12)
0xffffd215: "SHELL=/bin/bash"
(gdb) x/s *(void**)($esp + 16)
0xffffd225: "SESSION_MANAGER=local/LINUXUBUNTU:@/tmp/.ICE-unix/2067,unix/LINUXUBUNTU:/tmp/.ICE-unix/2067"
(gdb) x/s *(void**)($esp + 20)
0xffffd281: "QT_ACCESSIBILITY=1"
(gdb) x/s *(void**)($esp + 24)
0xffffd294: "COLORTERM=truecolor"
(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.4.18).

```

%include 'in_out.asm'
SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0
SECTION .text
GLOBAL _start
_start:
mov eax, msg_func
call sprintLF
pop ecx
pop edx
sub ecx, 1
mov esi, 0
next:
cmp ecx, 0h
jz _end
pop eax
call atoi
call _calculate_fx
add esi, eax
loop next
_end:
mov eax, msg_result
call sprint
mov eax, esi
call iprintLF
call quit
_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4
ret

```

Рис. 4.18: Изменённая программа предыдущей лабораторной работы

Код программы:

```

%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x- 4", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:

mov eax, msg_func
call sprintLF
pop ecx

```

```

pop edx
sub ecx, 1
mov esi, 0
next:
cmp ecx, 0h
jz _end
pop eax
call atoi
call _calculate_fx
add esi, eax
loop next
_end:
mov eax, msg_result
call sprint
mov eax, esi
call iprintLF
call quit
_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4
ret

```

2. Запускаю программу в режиме отладчика и пошагово, с помощью команды si, просматриваю изменение значений регистров после выполнения каждой инструкции. При выполнении инструкции inc %eax можно заметить, что значение регистра eax увеличивается на 1, однако при этом изменяются и флаги процессора (ZF, SF, PF и др.). Так как значение регистра eax позже участвует в вычислении функции и используется как аргумент для инструкции mul ebx, некорректное инкрементирование приводит к ошибочному результату вычислений. Кроме того, команда mul ebx после этого формирует 64-битный результат в регистрах edx:eax, что также влияет на итоговые данные. Из-за неверного изменения регистра eax программа неправильно

вычисляла значение функции. После исправления ошибки вычисление стало корректным, что подтверждается результатом работы программы (рис. 4.20).

```

esi      0x0      0      edi      0x0      0      eip      0x8049008
eflags   0x10206   [ PF IF RF ]   cs      0x23      35      ss      0x2b      43
ds       0x2b      43      es       0x2b      43      fs       0x0
gs       0x0      0

0x8049006 <nextchar+3> je 0x804900b <finished>
>0x8049008 <nextchar+5> inc %eax
0x8049009 <nextchar+6> jmp 0x8049003 <nextchar>
0x804900b <finished> sub %ebx,%eax
0x804900d <finished+2> pop %ebx
0x804900e <finished+3> ret
0x804900f <sprint> push %edx
0x8049010 <sprint+1> push %ecx
0x8049011 <sprint+2> push %ebx
0x8049012 <sprint+3> push %eax
0x8049013 <sprint+4> call 0x8049000 <slen>
0x8049018 <sprint+9> mov %eax,%edx
0x804901a <sprint+11> pop %eax

native process 13951 (asm) In: nextchar
eax      0x804a00e      134520846
ecx      0x0           0
edx      0x0           0
ebx      0x804a000      134520832
esp      0xffffd010     0xffffd010
ebp      0x0           0x0
esi      0x0           0
edi      0x0           0
eip      0x8049008      0x8049008 <nextchar+5>
eflags   0x10206       [ PF IF RF ]
cs       0x23          35
ss       0x2b          43
ds       0x2b          43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.19: Поиск ошибки в программе через пошаговую отладку

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

```

iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
iekim@LINUXUBUNTU:~/work/arch-pc/lab09$ ./lab9-4
Результат: 25
iekim@LINUXUBUNTU:~/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 также познакомился с методами отладки при помощи GDB и его основными возможностями.

6. Список литературы

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