

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Гашимова Эсма Эльшан кызы

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	13
4.1.4	Обработка аргументов командной строки в GDB	16
5	Выводы	25
	Список литературы	26

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	10
4.5	Проверка программы отладчиком	11
4.6	Запуск отладчика с брейкпойнтом	11
4.7	Дисассимилирование программы	12
4.8	Режим псевдографики	12
4.9	Список брейкпойнтов	13
4.10	Добавление второй точки останова	13
4.11	Просмотр содержимого регистров	14
4.12	Просмотр содержимого переменных двумя способами	14
4.13	Изменение содержимого переменных двумя способами	15
4.14	Просмотр значения регистра разными представлениями	15
4.15	Примеры использования команды set	16
4.16	Подготовка новой программы	16
4.17	Измененная программа предыдущей лабораторной работы	18
4.18	Поиск ошибки в программе через пошаговую отладку	22
4.19	Проверка корректировок в программме	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. 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 Релазация подпрограмм в NASM

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

```
esmagashimova@fedora:~$ mkdir ~/work/arch-pc/lab09
esmagashimova@fedora:~$ cd ~/work/arch-pc/lab09
esmagashimova@fedora:~/work/arch-pc/lab09$ touch lab09-1.asm
esmagashimova@fedora:~/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. -fig. 4.2).

```
esmagashimova@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 10
2x+7=27
esmagashimova@fedora:~/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).

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

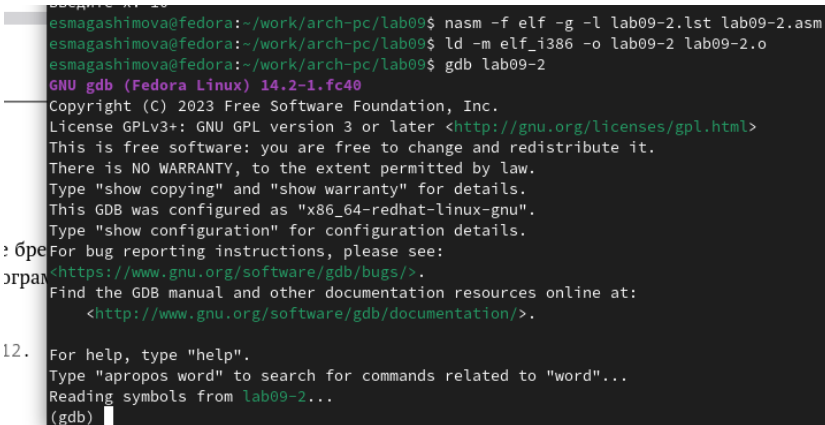
```

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. -fig. 4.4).



```

esmagashimova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
esmagashimova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
esmagashimova@fedora:~/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/>.
12. For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb)

```

Рис. 4.4: Запуск программы в отладчике

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

```

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/esmagashimova/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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5046) 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).

```

Reading symbols from lab09-2...
(gdb) run
Starting program: /home/esmagashimova/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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5046) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/esmagashimova/work/arch-pc/lab09/lab09-2

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

```

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд синтаксисом Intel *amd тончик* (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 `ax`, `eax`, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваря-

ются символом %, Intel - имена регистров пишутся без префиксов).

```
Enable debuginfo for this session? (y or [n]) y
Debuginfo has been enabled.
To make this setting permanent, add 'set debuginfo enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 5046) exited normally]
(gdb) break _start
Breakpoint 1 at 0x0049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/esmagashinova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x0049000 <+0>:    mov     $0x4,%eax
      0x0049005 <+5>:    mov     $0x1,%ebx
      0x004900a <+10>:   mov     $0x004a000,%ecx
      0x004900f <+15>:   mov     $0x8,%edx
      0x0049014 <+20>:   int     $0x80
      0x0049016 <+22>:   mov     $0x4,%eax
      0x004901b <+27>:   mov     $0x1,%ebx
      0x0049020 <+32>:   mov     $0x004a000,%ecx
      0x0049025 <+37>:   mov     $0x7,%edx
      0x004902a <+42>:   int     $0x80
      0x004902c <+44>:   mov     $0x1,%eax
      0x0049031 <+49>:   mov     $0x0,%ebx
      0x0049036 <+54>:   int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x0049000 <+0>:    mov     eax,0x4
      0x0049005 <+5>:    mov     ebx,0x1
      0x004900a <+10>:   mov     ecx,0x004a000
      0x004900f <+15>:   mov     edx,0x8
      0x0049014 <+20>:   int     0x80
      0x0049016 <+22>:   mov     eax,0x4
      0x004901b <+27>:   mov     ebx,0x1
      0x0049020 <+32>:   mov     ecx,0x004a000
      0x0049025 <+37>:   mov     edx,0x7
      0x004902a <+42>:   int     0x80
      0x004902c <+44>:   mov     eax,0x1
      0x0049031 <+49>:   mov     ebx,0x0
      0x0049036 <+54>:   int     0x80
End of assembler dump.
(gdb)
```

Рис. 4.7: Дисассимилирование программы

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

```
[ Register Values Unavailable ]

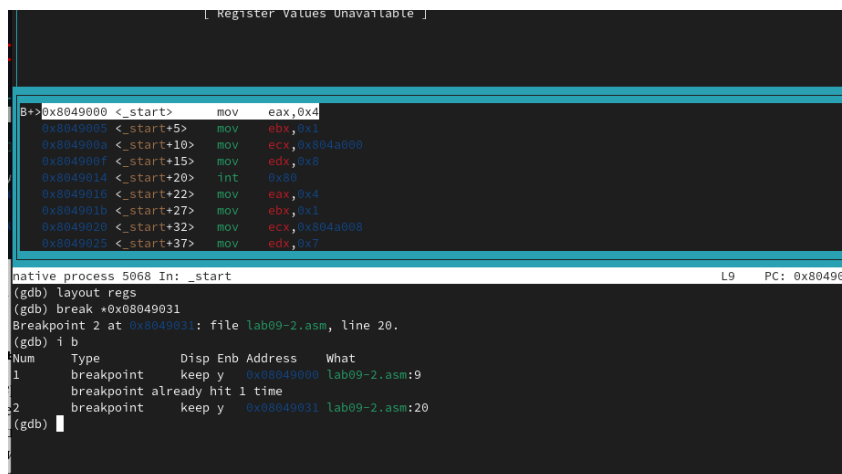
B->0x0049000 <_start>  mov     eax,0x4
0x0049005 <_start+5>  mov     ebx,0x1
0x004900a <_start+10> mov     ecx,0x004a000
0x004900f <_start+15> mov     edx,0x8
0x0049014 <_start+20> int     0x80
0x0049016 <_start+22> mov     eax,0x4
0x004901b <_start+27> mov     ebx,0x1
0x0049020 <_start+32> mov     ecx,0x004a000
0x0049025 <_start+37> mov     edx,0x7
0x004902a <_start+42> int     0x80

native process 5068 In: _start
(gdb) layout regs
(gdb)
```

Рис. 4.8: Режим псевдографики

4.1.2 Добавление точек останова

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



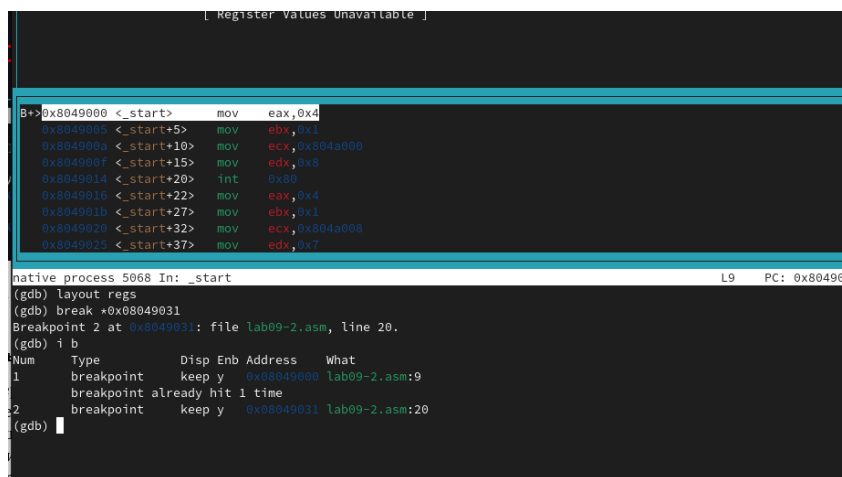
```
[ Register Values Unavailable ]

B->0x08049000 <_start> mov    eax,0x4
0x08049005 <_start+5> mov    ebx,0x1
0x0804900a <_start+10> mov    ecx,0x0804a000
0x0804900f <_start+15> mov    edx,0x8
0x08049014 <_start+20> int    0x80
0x08049016 <_start+22> mov    eax,0x4
0x0804901b <_start+27> mov    ebx,0x1
0x08049020 <_start+32> mov    ecx,0x0804a008
0x08049025 <_start+37> mov    edx,0x7

native process 5068 In: _start L9 PC: 0x080490
(gdb) layout regs
(gdb) break *0x08049031
Breakpoint 2 at 0x08049031: file lab09-2.asm, line 20.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 4.9: Список брейкпоинтов

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



```
[ Register Values Unavailable ]

B->0x08049000 <_start> mov    eax,0x4
0x08049005 <_start+5> mov    ebx,0x1
0x0804900a <_start+10> mov    ecx,0x0804a000
0x0804900f <_start+15> mov    edx,0x8
0x08049014 <_start+20> int    0x80
0x08049016 <_start+22> mov    eax,0x4
0x0804901b <_start+27> mov    ebx,0x1
0x08049020 <_start+32> mov    ecx,0x0804a008
0x08049025 <_start+37> mov    edx,0x7

native process 5068 In: _start L9 PC: 0x080490
(gdb) layout regs
(gdb) break *0x08049031
Breakpoint 2 at 0x08049031: file lab09-2.asm, line 20.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 4.10: Добавление второй точки останова

4.1.3 Работа с данными программы в GDB

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

```

[ Register values unavailable ]

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

native process 5068 In: _start L9 PC: 0x8049000
eax 0x0 0
ecx 0x0 0
edx 0x0 0
ebx 0x0 0
esp 0xffffd080 0xffffd080
ebp 0x0 0
esi 0x0 0
edi 0x0 0
eip 0x8049000 0x8049000 <_start>
eflags 0x202 [ IF ]
cs 0x23 35
--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. -fig. 4.12).

```

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

native process 5068 In: _start L9 PC:
esi 0x0 0
edi 0x0 0
eip 0x8049000 0x8049000 <_start>
eflags 0x202 [ IF ]
cs 0x23 35
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.12: Просмотр содержимого переменных двумя способами

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

```

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

native process 5068 In: _start
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:  "world!\n\034"
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a006 <msg1>:  "Hello, "
(gdb) set {char}msg2='x'
'msg2' has unknown type; cast it to its declared type
(gdb) set {char}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>:  "xorld!\n\034"
(gdb)

```

Рис. 4.13: Изменение содержимого переменных двумя способами

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

```

    0x80490d2      add     BYTE PTR [eax],al
    0x80490d4      add     BYTE PTR [eax],al
    0x80490d6      add     BYTE PTR [eax],al
    0x80490d8      add     BYTE PTR [eax],al
    0x80490da      add     BYTE PTR [eax],al
    0x80490dc      add     BYTE PTR [eax],al
    0x80490de      add     BYTE PTR [eax],al
    0x80490e0      add     BYTE PTR [eax],al
    0x80490e2      add     BYTE PTR [eax],al

native process 5068 In: _start
(gdb) set {char}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>:  "xorld!\n\034"
(gdb) p/t $ecx
$1 = 0
(gdb) p/s $edx
$2 = 0
(gdb) p/t $edx
$3 = 0
(gdb) p/x $edx
$4 = 0x0
(gdb)
msg1>:  "hello, "

```

Рис. 4.14: Просмотр значения регистра разными представлениями

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

```

0x88490f0    add    BYTE PTR [eax],al
0x88490f2    add    BYTE PTR [eax],al
0x88490f4    add    BYTE PTR [eax],al
0x88490f6    add    BYTE PTR [eax],al
0x88490f8    add    BYTE PTR [eax],al
0x88490fa    add    BYTE PTR [eax],al
0x88490fc    add    BYTE PTR [eax],al
0x88490fe    add    BYTE PTR [eax],al
0x8849100    add    BYTE PTR [eax],al

native process 5068 In: _start
$3 = 0
(gdb) p/x $edx
$4 = 0x0
(gdb) set $ebx='12'
(gdb) p/s
$5 = 0
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 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).

```

bash: p: команда не найдена...
esmagashimova@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
esmagashimova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
esmagashimova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
esmagashimova@fedora:~/work/arch-pc/lab09$

```

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. ??).


```

For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/esmagashimova/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:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd058: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd21a: "/home/esmagashimova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd243: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd25b: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd26c: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd26c: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Задание для само-

стоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. -fig. 4.17).

```

; Подготовка для чтения аргументов
pop ecx      ; В ecx количество аргументов
pop edx      ; В edx адрес списка аргументов
sub ecx, 1   ; Уменьшаем на 1, так как первый элемент - это количество

; Инициализируем сумму в esi
mov esi, 0   ; Сумма, которая будет накапливать результат

next:
cmp ecx, 0h  ; Проверяем, есть ли еще аргументы
jz _end      ; Если нет, переходим к выводу результата
pop eax      ; Получаем следующий аргумент (x_i)
call atoi    ; Преобразуем строку в число

; Вызов подпрограммы для вычисления f(x)
push eax     ; Сохраняем аргумент x в стеке
call _calculate_fx ; Вычисляем f(x)
add esi, eax  ; Добавляем результат в сумму
pop eax      ; Восстанавливаем x (необходимо для следующего вызова)

loop next    ; Переходим к следующему аргументу

_end:
; Выводим результат
mov eax, msg_result
call sprint

mov eax, esi ; Результат суммы
call iprintLF ; Выводим результат

; Завершаем программу
call quit

; Подпрограмма для вычисления f(x) = 2x + 15
_calculate_fx:
; На входе в eax - аргумент x
mov ebx, 2 ; Загружаем 2 в ebx
mul ebx    ; eax = eax * ebx (2 * x)
add eax, 15 ; eax = eax + 15 (2x + 15)
ret        ; Возвращаем результат в eax

```

Рис. 4.17: Измененная программа предыдущей лабораторной работы

Код программы:

```

#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  $f(g(x)) = 2 * g(x) + 7$ 
;-----
_calcul:
; Вызов подпрограммы _subcalcul для вычисления  $g(x)$ 
mov eax, [x] ; загружаем x
call _subcalcul ; eax =  $g(x)$ 

; Теперь вычисляем  $f(g(x)) = 2 * g(x) + 7$ 
mov ebx, 2
mul ebx ; eax =  $2 * g(x)$ 
add eax, 7 ; eax =  $2 * g(x) + 7$ 
mov [res], eax ; сохраняем результат в переменной res
ret

```

```

;-----
; Подпрограмма вычисления  $g(x) = 3 * x - 1$ 
;-----

```

```

_subcalcul:

```

```

mov eax, [x]      ; загружаем x
mov ebx, 3
mul ebx           ; eax = 3 * x
sub eax, 1        ; eax = 3 * x - 1
ret

```

```

mi, [07.12.2024 20:50]

```

```

%include 'in_out.asm'

```

```

SECTION .data

```

```

    msg_func db "Функция:  $f(x) = 2x + 15$ ", 0
    msg_result db "Результат: ", 0

```

```

SECTION .text

```

```

GLOBAL _start

```

```

_start:

```

```

    ; Выводим описание функции

```

```

    mov eax, msg_func
    call sprintf

```

```

    ; Подготовка для чтения аргументов

```

```

    pop ecx      ; В ecx количество аргументов
    pop edx      ; В edx адрес списка аргументов
    sub ecx, 1   ; Уменьшаем на 1, так как первый элемент - это количество

```

```

; Инициализируем сумму в esi
mov esi, 0          ; Сумма, которая будет накапливать результат

next:
    cmp ecx, 0h      ; Проверяем, есть ли еще аргументы
    jz _end          ; Если нет, переходим к выводу результата
    pop eax          ; Получаем следующий аргумент (x_i)
    call atoi         ; Преобразуем строку в число

; Вызов подпрограммы для вычисления f(x)
    push eax          ; Сохраняем аргумент x в стеке
    call _calculate_fx ; Вычисляем f(x)
    add esi, eax      ; Добавляем результат в сумму
    pop eax           ; Восстанавливаем x (необходимо для следующего вызова)

    loop next         ; Переходим к следующему аргументу

_end:
; Выводим результат
    mov eax, msg_result
    call sprint

    mov eax, esi      ; Результат суммы
    call iprintLF     ; Выводим результат

; Завершаем программу
    call quit

; Подпрограмма для вычисления  $f(x) = 2x + 15$ 

```

`_calculate_fx:`

```
; На входе в eax - аргумент x  
mov ebx, 2           ; Загружаем 2 в ebx  
mul ebx              ; eax = eax * ebx (2 * x)  
add eax, 15          ; eax = eax + 15 (2x + 15)  
ret                  ; Возвращаем результат в eax
```

2. Запускаю программу в режиме отладчика и пошагово через `si` просматриваю изменение значений регистров через `i r`. При выполнении инструкции `mul ebx` 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. 4.18).

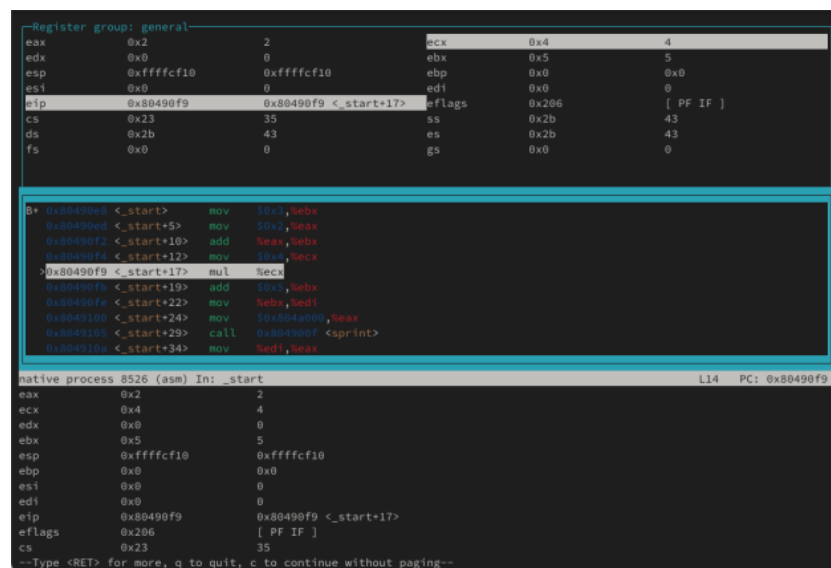


Рис. 4.18: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -fig. 4.19).

```
esmagashimova@fedora:~/work/arch-pc/lab09$ touch lab9-5.asm
esmagashimova@fedora:~/work/arch-pc/lab09$ mousepad lab9-5.asm
esmagashimova@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
esmagashimova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
esmagashimova@fedora:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
esmagashimova@fedora:~/work/arch-pc/lab09$
```

Рис. 4.19: 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.