

Отчёт по лабораторной работе №9

2023

Просина Ксения Максимовна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.1	Понятие об отладке	6
3	Выполнение лабораторной работы	7
3.1	Реализация подпрограмм в NASM	7
3.2	Добавление точек останова	14
3.3	Работа с данными программы в GDB	15
3.4	Обработка аргументов командной строки в GDB	17
3.5	Задание для самостоятельной работы	20
4	Выводы	25
	Список литературы	26

Список иллюстраций

3.1	Создание папки и файла	7
3.2	Проверка	9
3.3	Создание исполняемого файла	10
3.4	Загрузка в отладчик и проверка работы	11
3.5	Установка брейкпоинта и проверка	11
3.6	Просмотр дисассимилированного кода программы	12
3.7	Переключение на отображение команд с Intel'овским синтаксисом	13
3.8	Переключение на режим псевдографики	14
3.9	Проверка точки останова	15
3.10	Установка новой точки и проверка	15
3.11	Просмотр содержимого регистров	16
3.12	Посмотр значения переменной	16
3.13	Посмотр значения другой переменной	16
3.14	Замена символа	17
3.15	Замена	17
3.16	Копия файла	18
3.17	Создание исполняемого файла	18
3.18	Загрузка с аргументами	18
3.19	Установка точки останова и запуск	19
3.20	Адрес вершины стека	19
3.21	Остальные позиции стека	19
3.22	Копирование файла	20
3.23	Запуск	20
3.24	Проверка неправильного кода	21
3.25	Анализ изменений	22
3.26	Итоговый код	23
3.27	Правильный ответ	24

Список таблиц

1 Цель работы

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

2 Теоретическое введение

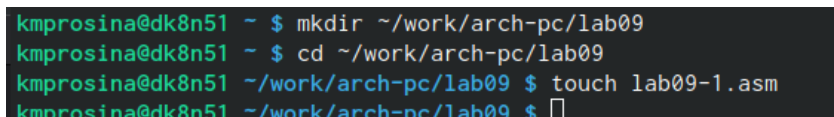
2.1 Понятие об отладке

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

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

1. Создайте каталог для выполнения лабораторной работы No 9, перейдите в него и создайте файл lab09-1.asm: `mkdir ~/work/arch-pc/lab09 cd ~/work/arch-pc/lab09 touch lab09-1.asm`



```
kmprosina@dk8n51 ~ $ mkdir ~/work/arch-pc/lab09
kmprosina@dk8n51 ~ $ cd ~/work/arch-pc/lab09
kmprosina@dk8n51 ~/work/arch-pc/lab09 $ touch lab09-1.asm
kmprosina@dk8n51 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Создание папки и файла

2. В качестве примера рассмотрим программу вычисления арифметического выражения $\text{X}(\text{X}) = 2\text{X} + 7$ с помощью подпрограммы `_calcul`. В данном примере `X` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 9.1).

Листинг 9.1. Пример программы с использованием вызова подпрограммы

```
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x:',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
```

```

res: RESB 80
SECTION .text
GLOBAL _start
_start:
;----- ; Основная программа ;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
;----- ; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

```

Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введен-

ных данных из символьного вида в численный (call atoi). После следующей инструкции call _calcul, которая передает управление подпрограмме _calcul, будут выполнены инструкции подпрограммы

Инструкция ret является последней в подпрограмме и ее исполнение приводит к возвращению в основную программу к инструкции, следующей за инструкцией call, которая вызвала данную подпрограмму. Последние строки программы реализуют вывод сообщения (call sprint), результата вычисления (call iprintLF) и завершение программы (call quit). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу.

```
kmprosina@dk8n51 ~/work/arch-pc/lab09 $ gedit lab09-1.asm
kmprosina@dk8n51 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
kmprosina@dk8n51 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
kmprosina@dk8n51 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 2
2x+7=11
```

Рис. 3.2: Проверка

С помощью команды gedit я отредактировала файл и вставила в него код из листинга. Далее я скомпилировала файл и проверила его работу

Измените текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения $x(x(x))$, где x вводится с клавиатуры, $x(x) = 2x + 7$, $x(x) = 3x - 1$. Т.е. x передается в подпрограмму _calcul из нее в подпрограмму _subcalcul, где вычисляется выражение $x(x)$, результат возвращается в _calcul и вычисляется выражение $x(x(x))$. Результат возвращается в основную программу для вывода результата на экран.

9.4.2. Отладка программ с помощью GDB Создайте файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!):

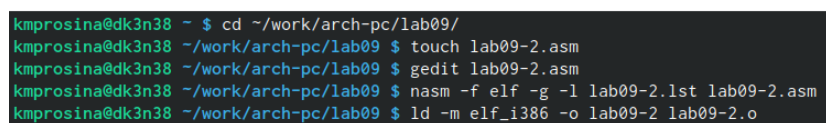
Листинг 9.2. Программа вывода сообщения Hello world! SECTION .data
msg1: db "Hello,",0x0
msg1Len: equ \$ - msg1
msg2: db "world!",0xa
msg2Len: equ \$ - msg2

```

SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80

```

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. `nasm -f elf -g -l lab09-2.lst lab09-2.asm ld -m elf_i386 -o lab09-2 lab09-2.o`



```

kmprosina@dk3n38 ~ $ cd ~/work/arch-pc/lab09/
kmprosina@dk3n38 ~/work/arch-pc/lab09 $ touch lab09-2.asm
kmprosina@dk3n38 ~/work/arch-pc/lab09 $ gedit lab09-2.asm
kmprosina@dk3n38 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
kmprosina@dk3n38 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o

```

Рис. 3.3: Создание исполняемого файла

Загрузите исполняемый файл в отладчик gdb: `user@dk4n31:~$ gdb lab09-2`
 Проверьте работу программы, запустив ее в оболочке GDB с помощью команды `run` (со- кращённо `r`): `(gdb) run` Starting program: `~/work/arch-pc/lab09/lab09-2` Hello,

world! [Inferior 1 (process 10220) exited normally] (gdb)

```
kmprosina@dk3n38 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 13.2 vanilla) 13.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4574) exited normally]
(gdb)
```

Рис. 3.4: Загрузка в отладчик и проверка работы

Для более подробного анализа программы установите брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустите её. (gdb) break `_start` Breakpoint 1 at 0x8049000: file lab09-2.asm, line 12. (gdb) run Starting program: `~/work/arch-pc/lab09/lab09-2` Breakpoint 1, `_start ()` at lab09-2.asm:12 12 mov eax, 4

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work/arch-pc/lab09/lab09-2

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

Рис. 3.5: Установка брейкпоинта и проверка

Посмотрите дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (gdb) `disassemble _start`

```

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

```

Рис. 3.6: Просмотр дисассимилированного кода программы

Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (gdb) `set disassembly-flavor intel` (gdb) `disassemble _start`

```

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

```

Рис. 3.7: Переключение на отображение команд с Intel'овским синтаксисом

Перечислите различия отображения синтаксиса машинных команд в режимах ATТ и Intel. Включите режим псевдографики для более удобного анализа программы (рис. 9.2): (gdb) layout asm (gdb) layout regs

```
[ 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
0x804902a <_start+42>     int      0x80
0x804902c <_start+44>     mov     eax,0x1
0x8049031 <_start+49>     mov     ebx,0x0
0x8049036 <_start+54>     int      0x80
0x8049038                add     BYTE PTR [eax],al

native process 4636 In: _start
(gdb) layout regs
(gdb) █
```

Рис. 3.8: Переключение на режим псевдографики

В этом режиме есть три окна: • В верхней части видны названия регистров и их текущие значения; • В средней части виден результат дисассимилирования программы; • Нижняя часть доступна для ввода команд.

3.2 Добавление точек останова

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: На предыду-

щих шагах была установлена точка останова по имени метки (`_start`). Проверьте это с помощью команды `info breakpoints` (кратко `i b`): `(gdb) info breakpoints`

```
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb)
```

Рис. 3.9: Проверка точки останова

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3). Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова. `(gdb) break *` Посмотрите информацию о всех установленных точках останова: `(gdb) i b`

```
(gdb) b *0x8049bbc
Breakpoint 2 at 0x8049bbc
(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   0x08049bbc
(gdb)
```

Рис. 3.10: Установка новой точки и проверка

3.3 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Значения каких регистров изменяются? Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`). `(gdb) info registers`

```

eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc230 0xffffc230
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
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 3.11: Просмотр содержимого регистров

Для отображения содержимого памяти можно использовать команду `x`, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: `x/NFU`. С помощью команды `x &` также можно посмотреть содержимое переменной. Посмотрите значение переменной `msg1` по имени (gdb) `x/1sb &msg1 0x804a000` : “Hello,”

```

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

```

Рис. 3.12: Посмотр значения переменной

Посмотрите значение переменной `msg2` по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию `mov esx,msg2` которая записывает в регистр `esx` адрес переменной `msg2` (рис. 9.4).

```

(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n"
(gdb)

```

Рис. 3.13: Посмотр значения другой переменной

Изменить значение для регистра или ячейки памяти можно с помощью команды `set`, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс `$`, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си). Измените первый символ переменной `msg1` (рис. 9.5): `(gdb) set {char}msg1='h'` `(gdb) x/1sb &msg1 0x804a000 : "hello,"` `(gdb)`

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

Рис. 3.14: Замена символа

Замените любой символ во второй переменной `msg2`.

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world \n"
```

Рис. 3.15: Замена

Чтобы посмотреть значения регистров используется команда `print /F` (перед именем регистра обязательно ставится префикс `$`) (рис. 9.6): `p/F $` Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`. С помощью команды `set` измените значение регистра `ebx`: `(gdb) set $ebx='2'` `(gdb) p/s $ebx $3 = 50` `(gdb) set $ebx=2` `(gdb) p/s $ebx $4 = 2`

3.4 Обработка аргументов командной строки в GDB

Скопируйте файл `lab8-2.asm`, созданный при выполнении лабораторной работы No8, с программой выводящей на экран аргументы командной строки

(Листинг 8.2) в файл с именем lab09-3.asm: `cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm`

```
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Рис. 3.16: Копия файла

Создайте исполняемый файл. `nasm -f elf -g -l lab09-3.lst lab09-3.asm ld -m elf_i386 -o lab09-3 lab09-3.o`

```
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 3.17: Создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы: `gdb -args lab09-3 аргумент1 аргумент2 'аргумент3'`

```
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент2 'аргумент3'
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb)
```

Рис. 3.18: Загрузка с аргументами

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. (gdb) `b _start` (gdb) `run`

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 аргумент3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)

```

Рис. 3.19: Установка точки останова и запуск

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): `(gdb) x/x $esp 0xffffd200: 0x05`

```

5      pop ecx ; Извлекаем из
(gdb) x/x $esp
0xfffffc300:      0x00000005
(gdb)

```

Рис. 3.20: Адрес вершины стека

Как видно, число аргументов равно 5 – это имя программы `lab09-3` и непосредственно аргументы: `аргумент1`, `аргумент, 2` и `‘аргумент 3’`. Посмотрите остальные позиции стека – по адресу `[esp+4]` располагается адрес в памяти где находится имя программы, по адресу `[esp+8]` храниться адрес первого аргумента, по адресу `[esp+12]` – второго и т.д.

```

(gdb) x/s *(void**)(esp + 4)
0xfffffc59d:  "/afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc5e3:  "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc5f5:  "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc606:  "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc608:  "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.21: Остальные позиции стека

Объясните, почему шаг изменения адреса равен 4 (`[esp+4]`, `[esp+8]`, `[esp+12]` и т.д.).

3.5 Задание для самостоятельной работы

1. Преобразуйте программу из лабораторной работы No8 (Задание No1 для самостоятельной работы), реализовав вычисление значения функции $\text{f}(x)$ как подпрограмму.

```
kmprosina@dk8n72 ~/work/arch-pc/lab08 $ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
```

Рис. 3.22: Копирование файла

Сначала я скопировала файл с помощью команды cp

```
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ gdb --args lab09-4 1 2 3 4
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab09-4...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work/arch-pc/lab09/lab09-4 1 2 3 4
Функция: f(x)=8x-3
Результат: 68
[Inferior 1 (process 5502) exited normally]
(gdb)
```

Рис. 3.23: Запуск

Далее, используя уже изученные команды, мне удалось выполнить задачу

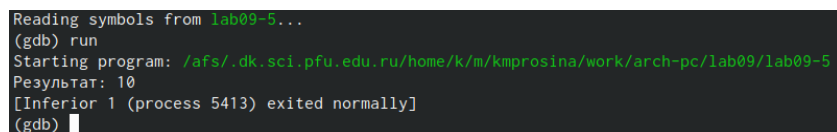
2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \times 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее. Листинг 9.3. Программа вычисления выражения $(3 + 2) \times 4 + 5$
- ```
%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

```



```

Reading symbols from lab09-5...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work/arch-pc/lab09/lab09-5
Результат: 10
[Inferior 1 (process 5413) exited normally]
(gdb)

```

Рис. 3.24: Проверка неправильного кода

```

Register group: general
eax 0x0 0
ecx 0x0 0
edx 0x0 0
ebx 0x0 0
esp 0xffffc350 0xffffc350
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490e8 0x80490e8 <_start>
eflags 0x202 [IF]
cs 0x23 35
ss 0x2b 43

B-> 0x80490e8 <_start> mov ebx,0x2
0x80490ed <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <printf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 <_start+46> add BYTE PTR [eax],al

native process 5658 In: _start L8 PC: 0x80490e8
(gdb) layout regs
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work
/arch-pc/lab09/lab09-5

Breakpoint 1, _start () at lab09-5.asm:8
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work
/arch-pc/lab09/lab09-5

Breakpoint 1, _start () at lab09-5.asm:8
(gdb)

```

Рис. 3.25: Анализ изменений

```

1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov eax,3
9 mov ebx,2
10 add eax,ebx
11 mov ebx,4
12 mul ebx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit

```

Рис. 3.26: Итоговый код

Итоговый код получился таким:

```

%include 'in_out.asm'
SECTION .data
div: DB 'Результат:',0
SECTION .text
GLOBAL _start
_start:
; -- Вычисление выражения (3+2)*4+5
mov eax,3 Главным числом я сделала eax
mov ebx,2 Вторым числом сделала
ebx
add eax,ebx Добавила к eax ebx
mov ebx,4 Далее я перезаписала ebx, чтобы

```

не использовать новые переменные `mul ebx` Умножила `eax` на `ebx` `add eax,5`  
Добавила к `eax` 5 и получила нужный результат! `mov edi, eax`

; -- Вывод результата на экран

`mov eax, div`

`call sprint`

`mov eax, edi`

`call iprintLF`

`call quit`

```
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-5.lst lab09-5.asm
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
kmprosina@dk8n72 ~/work/arch-pc/lab09 $ gdb lab09-5
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab09-5...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/k/m/kmprosina/work/arch-pc/lab09/lab09-5
Результат: 25
[Inferior 1 (process 6017) exited normally]
(gdb)
```

Рис. 3.27: Правильный ответ



## 4 Выводы

В ходе выполнения работы были приобретены навыки написания программ с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями

## Список литературы

[https://esystem.rudn.ru/pluginfile.php/2089671/mod\\_resource/content/0/%D0%9B%D0%B0%D0%](https://esystem.rudn.ru/pluginfile.php/2089671/mod_resource/content/0/%D0%9B%D0%B0%D0%)