

Отчёт по лабораторной работе №10

Понятие подпрограммы. Отладчик GDB.

Кочина Дарья Сергеевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	30

Список иллюстраций

4.1	Создание файла lab10-1.asm	8
4.2	Текст программы из листинга	9
4.3	Результат работы программы	10
4.4	Изменённый текст программы	11
4.5	Изменённый текст программы	12
4.6	Результат работы программы	13
4.7	Создание файла lab10-2.asm	13
4.8	Текст программы из листинга	14
4.9	Загрузка исполняемого файла в отладчик gdb	15
4.10	Команда run	15
4.11	Запуск программы	15
4.12	Дисассимилированный код программы	16
4.13	Команда disassemble	16
4.14	Команда set disassembly-flavor intel	17
4.15	Команда layout asm	17
4.16	Команда layout regs	18
4.17	Команда info breakpoints	18
4.18	Установка точки останова	19
4.19	Значение переменной msg1 по имени	19
4.20	Команда set	20
4.21	Команда set	20
4.22	Замена символа	21
4.23	Изменение значения регистра ebx	21
4.24	Изменение значения регистра ebx	21
4.25	Выход из GDB	22
4.26	Копирование	22
4.27	Создание исполняемого файла	22
4.28	Установка и запуск точки останова	23
4.29	Адрес вершины стека	23
4.30	Просмотр позиций стека	23
4.31	Преобразование программы	25
4.32	Проверка работы программы	26
4.33	Текст программы из листинга	26
4.34	Проверка программы	27
4.35	Изменённый текст программы	28
4.36	Результат работы программы	29

1 Цель работы

Целью данной лабораторной работы является приобретение навыков написания программ с использованием подпрограмм. А также знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

Приобрести навыки написания программ с использованием подпрограмм. А также ознакомиться с методами отладки при помощи GDB и его основными возможностями.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

1. обнаружение ошибки;
2. поиск её местонахождения;
3. определение причины ошибки;
4. 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.

Наиболее часто применяют следующие методы отладки:

1. Создание точек контроля значений на входе и выходе участка программы;
2. Использование специальных программ-отладчиков.

Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия.

Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд.

Наиболее популярные виды точек останова:

1. Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);
2. Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования.

GDB может выполнять следующие действия:

1. Начать выполнение программы, задав всё, что может повлиять на её поведение;
2. Остановить программу при указанных условиях;
3. Исследовать, что случилось, когда программа остановилась;
4. Изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

4 Выполнение лабораторной работы

1. Я создала каталог для выполнения лабораторной работы №10, перешла в него и создала файл lab10-1.asm. (рис. 4.1)

```
dskochina@dk8n70 ~ $ mkdir ~/work/study/2022-2023/Архитектура\ компьютера/arch-pc/lab10
dskochina@dk8n70 ~ $ cd ~/work/study/2022-2023/Архитектура\ компьютера/arch-pc/lab10
dskochina@dk8n70 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ touch lab10-1.asm
dskochina@dk8n70 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ls
lab10-1.asm
dskochina@dk8n70 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $
```

Рис. 4.1: Создание файла lab10-1.asm

2. Я ввела в файл lab10-1.asm текст программы из листинга (пример программы с использованием вызова подпрограммы), создала исполняемый файл и проверила его работу. (рис. 4.2, 4.3)


```

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
rezs: 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:
mov ebx,2
mul ebx
add eax,7
mov [rez],eax
ret

```

Рис. 4.2: Текст программы из листинга

```

dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ nasm -f elf lab10-1.asm
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ./lab10-1
Введите x: 3
2x+7=13
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ./lab10-1
Введите x: 5
2x+7=17
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ 

```

Рис. 4.3: Результат работы программы

3. Я изменила текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x+7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму `_calcul` из неё в подпрограмму `_subcalcul`, где вычисляется выражение $g(x)$, результат возвращается в `_calcul` и вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран. (рис. 4.4, 4.5)

```

#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
prim1: DB 'f(x) = 2x+7',0
prim2: DB 'g(x) = 3x-1',0
result: DB 'f(g(x))= ',0

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax,prim1
call sprintLF

mov eax,prim2
call sprintLF

mov eax,msg
call sprint

mov ecx,x
mov edx,80
call sread

mov eax,x
call atoi

call _calcul

mov eax,result

```

Рис. 4.4: Изменённый текст программы

```
call _calcul

mov eax,result
call sprint
mov eax,[res]
call iprintLF

call quit

_calcul:

call _subcalcul

mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret

_subcalcul:
mov ebx,3
mul ebx
sub eax,1
ret
```

Рис. 4.5: Изменённый текст программы

4. Я создала исполняемый файл и проверила его работу. (рис. 4.6)

```
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ nasm -f elf lab10-1.asm
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ./lab10-1
f(x) = 2x+7
g(x) = 3x-1
Введите x: 4
f(g(x))= 29
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ./lab10-1
f(x) = 2x+7
g(x) = 3x-1
Введите x: 5
f(g(x))= 35
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $
```

Рис. 4.6: Результат работы программы

5. Я создала файл lab10-2.asm с текстом программы из листинга (Программа печати сообщения Hello world!). (рис. 4.7, 4.8)

```
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ touch lab10-2.asm
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ls
in_out.asm lab10-1 lab10-1.asm lab10-1.o lab10-2.asm
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $
```

Рис. 4.7: Создание файла lab10-2.asm

```
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
```

Рис. 4.8: Текст программы из листинга

6. Я получила исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Я загрузила исполняемый файл в отладчик gdb. (рис. 4.9)

```
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ nasm -f elf -g -l lab10-2.lst lab10-2.asm
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ld -m elf_i386 -o lab10-2 lab10-2.o
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ gdb lab10-2
```

Рис. 4.9: Загрузка исполняемого файла в отладчик gdb

7. Я проверила работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r). (рис. 4.10)

```
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ nasm -f elf -g -l lab10-2.lst lab10-2.asm
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ld -m elf_i386 -o lab10-2 lab10-2.o
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ gdb lab10-2
GNU gdb (Gentoo 11.2 vanilla) 11.2
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 lab10-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 5533) exited normally]
(gdb)
```

Рис. 4.10: Команда run

8. Для более подробного анализа программы я установила брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустила её. (рис. 4.11)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb)
```

Рис. 4.11: Запуск программы

9. Я посмотрела дисассимилированный код программы с помощью команды disassemble начиная с метки _start. (рис. 4.12, 4.13)

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10/lab10-2
Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
(gdb) █
```

Рис. 4.12: Дисассимилированный код программы

```
(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. 4.13: Команда disassemble

10. Я переключилась на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. (рис. 4.14)


```

(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. 4.14: Команда set disassembly-flavor intel

11. Различия отображения синтаксиса машинных команд в режимах ATТ и Intel.
12. Я включила режим псевдографики для более удобного анализа программы.
(рис. 4.15, 4.16)



```

0x08049000 C:\start> mov     eax,0x4
0x08049005 C:\start+5> mov     ebx,0x1
0x0804900a C:\start+10> mov     ecx,0x804a000
0x0804900f C:\start+15> mov     edx,0x8
0x08049014 C:\start+20> int     0x80
0x08049016 C:\start+22> mov     eax,0x4
0x0804901b C:\start+27> mov     ebx,0x1
0x08049020 C:\start+32> mov     ecx,0x804a008
0x08049025 C:\start+37> mov     edx,0x7
0x0804902a C:\start+42> int     0x80
0x0804902c C:\start+44> mov     eax,0x1
0x08049031 C:\start+49> mov     ebx,0x0
0x08049036 C:\start+54> int     0x80

```

Рис. 4.15: Команда layout asm

```

[ Register Values Unavailable ]

0x08049000 <_start> mov     eax,0x1
0x08049005 <_start+5> mov     ebx,0x1
0x0804900a <_start+10> mov     ecx,0x804a000
0x0804900f <_start+15> mov     edx,0x0
0x08049014 <_start+20> int     0x58
0x08049019 <_start+25> mov     eax,0x4
0x0804901e <_start+30> mov     ebx,0x1
0x08049023 <_start+35> mov     ecx,0x804a008
0x08049028 <_start+40> mov     edx,0x7
0x0804902d <_start+45> int     0x58
0x08049032 <_start+50> mov     eax,0x1

native process 5863 In: _start
(gdb) layout regs
(gdb)

```

Рис. 4.16: Команда layout regs

13. На предыдущих шагах была установлена точка останова по имени метки (_start). Я проверила это с помощью команды info breakpoints (кратко i b). (рис. 4.17)

```

(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab10-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x8049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab10-2.asm, line 9.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab10-2.asm:9
breakpoint already hit 1 time
2        breakpoint      keep y   0x08049000 lab10-2.asm:9
(gdb)

```

Рис. 4.17: Команда info breakpoints

14. Я установила ещё одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Я определила адрес предпоследней инструкции (mov ebx,0x0) и установила точку останова. (рис. 4.18)

```

(gdb) b *0x8049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab10-2.asm, line 9.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab10-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049000 lab10-2.asm:9
(gdb) 

```

Рис. 4.18: Установка точки останова

15. Я выполнила 5 инструкций с помощью команды `stepi` (или `si`) и проследила за изменением значений регистров.
16. Я посмотрела содержимое регистров с помощью команды `info registers` (или `i r`).
17. Я посмотрела значение переменной `msg1` по имени. (рис. 4.19)

```

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

```

Рис. 4.19: Значение переменной `msg1` по имени

18. Я посмотрела значение переменной `msg2` по адресу. Адрес переменной можно определить по дисассемблированной инструкции. Также я посмотрела инструкцию `mov ecx,msg2` которая записывает в регистр `ecx` адрес переменной `msg2`.
19. Я изменила значение для регистра или ячейки памяти с помощью команды `set`. Также я изменила первый символ переменной `msg1`. (рис. 4.20, 4.21)

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

Рис. 4.20: Команда set

```
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

native process 3819 In: _start
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='k'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "korld!\n\034"
(gdb) p/x $edx
$1 = 0x0
(gdb) p/t $edx
$2 = 0
(gdb) p/s $edx
$3 = 0
(gdb) □
```

Рис. 4.21: Команда set

20. Я заменила символ во второй переменной msg2. (рис. 4.22)

```
(gdb) set {char}&msg2='k'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "kor1d!\n\034"
(gdb) □
```

Рис. 4.22: Замена символа

21. Я вывела в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.
22. С помощью команды set я изменила значение регистра ebx. (рис. 4.23, 4.24)

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) □
```

Рис. 4.23: Изменение значения регистра ebx

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

Рис. 4.24: Изменение значения регистра ebx

23. Разница вывода команд p/s \$ebx.

24. Я завершила выполнение программы с помощью команды `continue` (сокращенно `c`) и вышла из GDB с помощью команды `quit` (сокращенно `q`). (рис. 4.25)

```
(gdb) continue
Continuing.
hello, korld!
[Inferior 1 (process 3819) exited normally]
(gdb) quit
```

Рис. 4.25: Выход из GDB

25. Я скопировала файл `lab9-2.asm`, созданный при выполнении лабораторной работы №9, с программой, выводящей на экран аргументы командной строки в файл с именем `lab10-3.asm`. (рис. 4.26)

```
dskochina@dkn63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ cp ~/work/study/2022-2023/Архитектура
компьютера/arch-pc/lab9/lab9-2.asm ~/work/study/2022-2023/Архитектура
компьютера/arch-pc/lab10/lab10-3.asm
dskochina@dkn63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $
```

Рис. 4.26: Копирование

26. Я создала исполняемый файл. (рис. 4.27)

```
dskochina@dkn63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ nasm -f elf -g -l lab10-3.lst lab10-3.asm
dskochina@dkn63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ld -m elf_i386 -o lab10-3 lab10-3.o
dskochina@dkn63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ gdb --args lab10-3 аргумент1 аргумент2 'аргумент3'
GNU gdb (Gentoo 11.2 vanilla) 11.2
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 lab10-3...
(gdb)
```

Рис. 4.27: Создание исполняемого файла

27. Для загрузки в `gdb` программы с аргументами необходимо использовать ключ `-args`. Я загрузила исполняемый файл в отладчик, указав аргументы.

28. 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 её. (рис. 4.28)

```
(gdb) b _start
Breakpoint 1 at 0x00400000: file lab10-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10/lab10-3 аргумент1 аргумент2 аргумент3
Breakpoint 1, _start () at lab10-3.asm:5
>
0x00000000 esp; Извлекаем из стека в 'esp' количество
(gdb) █
```

Рис. 4.28: Установка и запуск точки останова

29. Адрес вершины стека хранится в регистре esp и по этому адресу располагается число, равное количеству аргументов командной строки (включая имя программы). (рис. 4.29)

```
(gdb) x/x $esp
0xfffffc4a0:      0x00000004
(gdb) █
```

Рис. 4.29: Адрес вершины стека

30. Как видно, число аргументов равно 4 – это имя программы lab10-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Я посмотрела остальные позиции стека. (рис. 4.30)

```
(gdb) x/s +(void**)(esp + 4)
0xffffc710:      "/afs/.dk.sci.pfu.edu.ru/home/d/s/dskochina/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10/lab10-3"
(gdb) x/s +(void**)(esp + 8)
0xffffc702:      "аргумент1"
(gdb) x/s +(void**)(esp + 12)
0xffffc704:      "аргумент2"
(gdb) x/s +(void**)(esp + 16)
0xffffc706:      "аргумент3"
(gdb) x/s +(void**)(esp + 20)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) x/s +(void**)(esp + 24)
0xffffc708:      "SHELL=/bin/bash"
(gdb) █
```

Рис. 4.30: Просмотр позиций стека

Самостоятельная работа

1. Я преобразовала программу из лабораторной работы №9 (задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.


```

#include 'in_out.asm'
SECTION .data
    ms1 db "Функция :f(x)=7(x+1) ", 0
    ms2 db "Результат: ", 0
SECTION .text
global _start
_start:
    mov eax,ms1
    call sprintf
    pop ecx
    pop edx
    sub ecx,1
    mov esi,0
    mov ebx,7
next:
    cmp ecx,0h
    jz _end
    pop eax
    call atoi
    call _calcul
    loop next
_end:
    mov eax, ms2
    call sprintf
    mov eax, esi
    call iprintf
    call quit
_calcul:
    add eax,1
    mul ebx
    add esi,eax
    ret

```

Рис. 4.31: Преобразование программы

```

dskochina@dk8n70 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ nasm -f elf lab10-5.asm
dskochina@dk8n70 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ld -m elf_i386 -o lab10-5 lab10-5.o
dskochina@dk8n70 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ./lab10-5 1 2 3
Функция :f(x)=7(x+1)
Результат: 63
dskochina@dk8n70 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ./lab10-5 3 2 4
Функция :f(x)=7(x+1)
Результат: 84
dskochina@dk8n70 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ 

```

Рис. 4.32: Проверка работы программы

- В листинге приведена программа вычисления выражения $(3+2)*4+5$. При запуске данная программа даёт неверный результат. С помощью отладчика GDB, анализируя изменения значений регистров, я определила ошибку и исправила её. (рис. 4.33, 4.34, 4.35, 4.36)

```

#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

```

Рис. 4.33: Текст программы из листинга

```
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ nasm -f elf lab10-4.asm
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ld -m elf_i386 -o lab10-4 lab10-4.o
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ./lab10-4
Результат: 10
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $
```

Рис. 4.34: Проверка программы

```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add 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
```

Рис. 4.35: Изменённый текст программы

```
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ nasm -f elf lab10-4.asm
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ld -m elf_i386 -o lab10-4 lab10-4.o
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $ ./lab10-4
Результат: 25
dskochina@dk8n63 ~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab10 $
```

Рис. 4.36: Результат работы программы

5 Выводы

В ходе выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм. Ознакомилась с методами отладки при помощи GDB и его основными возможностями.