

Лабораторная работа №9

Дисциплина: Архитектура компьютера

Малюга Валерия Васильевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	9
4.1	Реализация подпрограмм в NASM	9
4.2	Отладка программ с помощью GDB	12
4.2.1	Добавление точек останова	15
4.2.2	Работа с данными программы в GDB	16
4.2.3	Обработка аргументов командной строки в GDB	22
4.3	Задание для самостоятельной работы	23
5	Выводы	29

Список иллюстраций

4.1	Создание исполняемого файла lab9-1.asm и его запуск	10
4.2	Изменение текста программы	11
4.3	Создание и анализ программы	13
4.4	Использование команд disassemble и disassembly-flavor intel . . .	14
4.5	Включение режима псевдографики	15
4.6	Установка точек останова и просмотр информации о них . . .	16
4.7	Запуск исполняемого файла и проверка его работы	17
4.8	Просмотр значений переменных	18
4.9	Использование команды set	19
4.10	Вывод значения регистра в разных представлениях	20
4.11	Использование команды set для изменения значения регистра . .	21
4.12	Завершение работы GDB	21
4.13	Загрузка файла с аргументами в отладчик и установка точек останова	22
4.14	Просмотр значений, введенных в стек	22
4.15	Запуск программы и проверка его вывода	23
4.16	Создание и запуск исполняемого файла	24
4.17	Включение режима псевдографики	25
4.18	Поиск причины ошибки	26
4.19	Поиск причины ошибки	27

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. Добавление точек остановки
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Задания для самостоятельной работы

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIXподобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки

отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда `quit` (или сокращённо `q`).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программе можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`.

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`.

Обратно точка останова активируется командой `enable`.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`.

Для продолжения остановленной программы используется команда `continue` (`c`). Выполнение программы будет происходить до следующей точки останова.

В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке).

Команда `stepi` (кратко `si`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

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. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов.

При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `еір`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создала каталог для программ лабораторной работы № 9, перешла в него и создала файл lab09-1.asm. Ввела в файл lab09-1.asm текст программы из листинга 9.1. Создала исполняемый файл и проверила его работу (рис. 4.1).

```

vvmalyuga@Malyuga:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 0
f(g(x)) = 2(3x-1) + 7 = 5
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 1
f(g(x)) = 2(3x-1) + 7 = 11
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
f(g(x)) = 2(3x-1) + 7 = 23

```

```

8  x: RESB 80
9  res: RESB 80
10
11 SECTION .text
12     GLOBAL _start
13 ;-----
14 ; Основная программа
15 ;-----
16 _start:
17     mov eax, msg
18     call sprint
19     mov ecx, x
20     mov edx, 80
21     call sread
22     mov eax, x
23     call atoi
24     call _calcul          ; Вызов подпрограммы _calcul
25     mov eax, result
26     call sprint
27     mov eax, [res]
28     call iprintfLF
29     call quit
30 ;-----
31 ; подпрограмма вычисления f(x) = 2x + 7
32 ;-----
33 _calcul:
34     mov ebx, eax          ; сохраняем x
35     call _subcalcul       ; вызов подпрограммы _subcalcul для вычисления g(x)
36     mov ebx, 2
37     imul eax, ebx         ; умножаем g(x) на 2
38     add eax, 7            ; добавляем 7
39     mov [res], eax
40     ret
41 ;-----
42 ; подпрограмма вычисления g(x) = 3x - 1
43 ;-----
44 _subcalcul:
45     mov ebx, 3
46     imul eax, ebx         ; умножаем x на 3
47     sub eax, 1            ; вычитаем 1
48     ret

```

Рис. 4.1: Создание исполняемого файла lab9-1.asm и его запуск

Изменила текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. x передается в подпрограмму `_calcul`, из нее 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. 4.2).

```

vvmalyuga@Malyuga:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=13

```

Рис. 4.2: Изменение текста программы

Прилагаю код:

```

#include 'in_out.asm'

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

SECTION .bss
    x: RESB 80
    res: RESB 80

SECTION .text
    GLOBAL _start

;-----
; Основная программа
;-----

_start:
    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  $f(x) = 2x + 7$ 
;-----

_calcul:
    mov ebx, eax           ; сохраняем x
    call _subcalcul        ; вызов подпрограммы _subcalcul для вычисления g
    mov ebx, 2
    imul eax, ebx          ; умножаем g(x) на 2
    add eax, 7             ; добавляем 7
    mov [res], eax
    ret
;-----
; подпрограмма вычисления  $g(x) = 3x - 1$ 
;-----

_subcalcul:
    mov ebx, 3
    imul eax, ebx          ; умножаем x на 3
    sub eax, 1             ; вычитаем 1
    ret

```

4.2 Отладка программ с помощью GDB

Создала файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!). Получила исполняемый файл. Загрузила исполняемый файл в отладчик gdb. Проверила работу программы, запустив ее в оболочке GDB с помощью команды run. Для более подробного анализа программы уста-

новила брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустила её (рис. 4.3).

```
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ touch lab09-2.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ gedit lab09-2.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
vvmalyuga@Malyuga:~/work/arch-pc/lab09$
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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-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 lab09-2...
(gdb) run
Starting program: /home/vvmalyuga/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 637) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/vvmalyuga/work/arch-pc/lab09/lab09-2

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

Рис. 4.3: Создание и анализ программы

Посмотрела дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`. Переключилась на отображение команд с Intel'овским синтаксисом, введя команду `setdisassembly-flavor intel` (рис. 4.4).

```

(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
(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. 4.4: Использование команд disassemble и disassembly-flavor intel

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel: В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется привычный нам синтаксис.

Включила режим псевдографики для более удобного анализа программы (рис. 4.5).

```
[ 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
    0x804903a      add     BYTE PTR [eax],al

native process 1022 In: _start
(gdb) layout regs
(gdb)
```

Рис. 4.5: Включение режима псевдографики

4.2.1 Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова (рис. 4.6).

```

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

native process 1022 In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint    keep y 0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: 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. 4.6: Установление точек останова и просмотр информации о них

4.2.2 Работа с данными программы в GDB

Выполнила 5 инструкций с помощью команды `stepi` и проследила за изменением значений регистров (рис. 4.7). Изменились значения регистров `eax`, `ecx`, `edx` и `ebx`.


```

Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd260      0xffffd260      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      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
b+ 0x8049031 <_start+49>     mov     ebx,0x0

native process 1022 In: _start      L14 PC
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: 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) stepi 5
(gdb)

```

Рис. 4.7: Запуск исполняемого файла и проверка его работы

С помощью команды `x/1sb &msg1` просмотрела значение переменной `msg1` и значение переменной `msg2` по ее адресу (рис. 4.8).

```

Register group: general
eax      0x8      8      ecx
edx      0x8      8      ebx
esp      0xffffd260 0xffffd260  ebp
esi      0x0      0      edi
eip      0x8049016 0x8049016 <_start+22>  eflags
cs       0x23     35     ss
ds       0x2b     43     es
fs       0x0      0      gs

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 1022 In: _start
esp      0xffffd260 0xffffd260
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
--Type <RET> for more, q to quit, c to continue without paging--fs
gs       0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.8: Просмотр значений переменных

С помощью команды `set` изменила первый символ переменной `msg1` и заменила первый символ в переменной `msg2` (рис. 4.9).

```

(gdb) set {char}&msg1='h'
(gdb) x/lsb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2 = 'b'
(gdb) x/lsb &msg2
0x804a008 <msg2>:      "borld!\n\034"
(gdb)

```

Рис. 4.9: Использование команды set

Вывела в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра `edx` с помощью команды `print p/F $val` (рис. 4.10).

```

Register group: general
eax      0x8      8
edx      0x8      8
esp      0xffffd260 0xffffd260
esi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
cs       0x23     35
ds       0x2b     43
fs       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

native process 1022 In: _start
(gdb) set {char}&msg1='h'
(gdb) x/lb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2 = 'b'
(gdb) x/lb &msg2
0x804a008 <msg2>:      "borld!\n\034"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb)

```

Рис. 4.10: Вывод значения регистра в разных представлениях

С помощью команды `set` изменила значение регистра `ebx` в соответствии с заданием (рис. 4.11).

```

Register group: general
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x2      2
esp      0xffffd260      0xffffd260      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
fs       0x0      0      gs       0x0      0

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

native process 1022 In: _start L14 PC
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)

```

Рис. 4.11: Использование команды set для изменения значения регистра

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется. Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit (рис. 4.12).

```

Register group: general
eax      0x1      1      ecx      0x804a008      134520840
edx      0x7      7      ebx      0x1      1
esp      0xffffd260      0xffffd260      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049031      0x8049031 <_start+49>      eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
fs       0x0      0      gs       0x0      0

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
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al

native process 1022 In: _start L20 PC
(gdb) continue
Continuing.
world!
Breakpoint 2, _start () at lab09-2.asm:20
(gdb) quit
A debugging session is active.

Inferior 1 [process 1022] will be killed.
Quit anyway? (y or n)

```

Рис. 4.12: Завершение работы GDB

4.2.3 Обработка аргументов командной строки в GDB

Скопировала файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создала исполняемый файл (рис. 4.13).

```
vmalyuga@vmalyuga:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
vmalyuga@vmalyuga:~/work/arch-pc/lab09$ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2 lab09-2.asm lab09-2.lst lab09-2.o lab09-3.asm
vmalyuga@vmalyuga:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
vmalyuga@vmalyuga:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
vmalyuga@vmalyuga:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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-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 lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/vmalyuga/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx          ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 4.13: Загрузка файла с аргументами в отладчик и установка точек останова

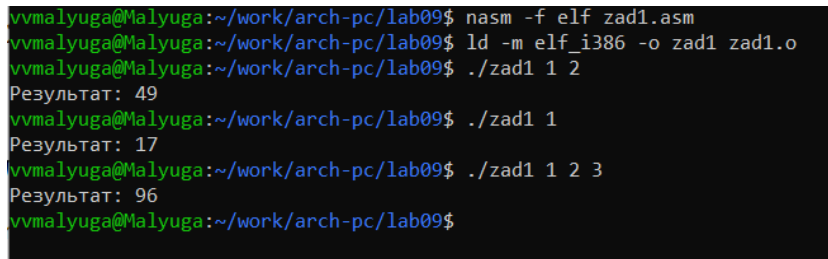
Посмотрела вершину стека и позиции стека по их адресам (рис. 4.14). Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

```
(gdb) x/x $esp
0xffffd220:  0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd382:  "/home/vmalyuga/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd3ad:  "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd3bf:  "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd3d0:  "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd3d2:  "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.14: Просмотр значений, введенных в стек

4.3 Задание для самостоятельной работы

1. Преобразовала программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $\text{X}(\text{X})$ как подпрограмму (рис. 4.15).



```
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ nasm -f elf zad1.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ld -m elf_i386 -o zad1 zad1.o
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./zad1 1 2
Результат: 49
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./zad1 1
Результат: 17
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./zad1 1 2 3
Результат: 96
vvmalyuga@Malyuga:~/work/arch-pc/lab09$
```

Рис. 4.15: Запуск программы и проверка его вывода

Прилагаю код:

```
%include 'in_out.asm'

SECTION .data
    msg db "Результат: ", 0

SECTION .text
    global _start

_calcul:
    imul eax, 15    ; умножаем x на 15
    add  eax, 2     ; добавляем 2
    ret

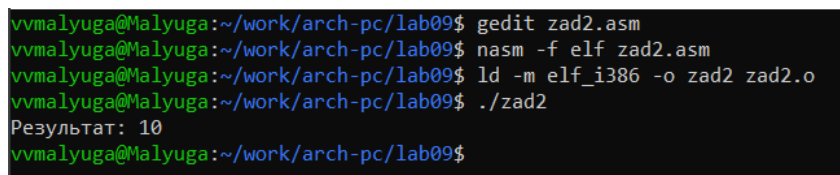
_start:
    pop ecx        ; Извлекаем из стека в `ecx` количество
                  ; аргументов (первое значение в стеке)
    pop edx        ; Извлекаем из стека в `edx` имя программы
                  ; (второе значение в стеке)
    sub  ecx, 1     ; Уменьшаем `ecx` на 1 (количество
```

```

; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
    cmp ecx, 0h ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет, выходим из цикла
; (переход на метку `_end`)
    pop eax ; иначе извлекаем следующий аргумент из стека
    call atoi ; преобразуем символ в число
    call _calcul ; вызываем подпрограмму для вычисления f(x)
    add esi, eax ; добавляем значение функции для
; конкретного аргумента к промежуточной сумме
    loop next ; переход к обработке следующего аргумента
_end:
    mov eax, msg ; вывод сообщения "Результат: "
    call sprint
    mov eax, esi ; записываем сумму в регистр `eax`
    call iprintLF ; печать результата
    call quit ; завершение программы

```

2. Ввела в файл zad2.asm текст программы из листинга 9.3. При корректной работе программы должно выводиться “25”. Создала исполняемый файл и запустила его. Получили неверный ответ: 10 (рис. 4.16).



```

vvmalyuga@Malyuga:~/work/arch-pc/lab09$ gedit zad2.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ nasm -f elf zad2.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ld -m elf_i386 -o zad2 zad2.o
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./zad2
Результат: 10
vvmalyuga@Malyuga:~/work/arch-pc/lab09$

```

Рис. 4.16: Создание и запуск исполняемого файла

Получила исполняемый файл для работы с GDB, запустила его и поставила брейкпоинт на `_start`. Затем включила режим псевдографики (рис. 4.17).

```
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ gdb ./zad2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 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-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 ./zad2...
(No debugging symbols found in ./zad2)
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) layout asm
```

Рис. 4.17: Включение режима псевдографики

С помощью команды `stepi` проследила за изменениями значений регистров в течение выполнения программы. После пятой команды (строка `mul esx`) заметила, что происходит умножение `esx` на `eax`, то есть 4 на 2, вместо умножения 4 на 5 (регистр `ebx`) (рис. 4.18).

```

Register group: general
eax      0x8      8      ecx      0x4      4
edx      0x0      0      ebx      0x5      5
esp      0xffffd270 0xffffd270 ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x80490fb 0x80490fb <_start+19> eflags  0x206    [ PF IF
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

0x80490e8 <_start> mov $0x3,%ebx
0x80490ed <_start+5> mov $0x2,%eax
0x80490f2 <_start+10> add %eax,%ebx
0x80490f4 <_start+12> mov $0x4,%ecx
0x80490f9 <_start+17> mul %ecx
> 0x80490fb <_start+19> add $0x5,%ebx
0x80490fe <_start+22> mov %ebx,%edi
0x8049100 <_start+24> mov $0x804a000,%eax
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov %edi,%eax
0x804910c <_start+36> call 0x8049086 <iprintLF>
0x8049111 <_start+41> call 0x80490db <quit>

native process 887 In: _start L??
(gdb) layout reg
(gdb) stepi
0x080490ed in _start ()
(gdb) stepi
0x080490f2 in _start ()
(gdb) stepi
0x080490f4 in _start ()
(gdb) stepi
0x080490f9 in _start ()
(gdb) stepi
0x080490fb in _start ()
(gdb)

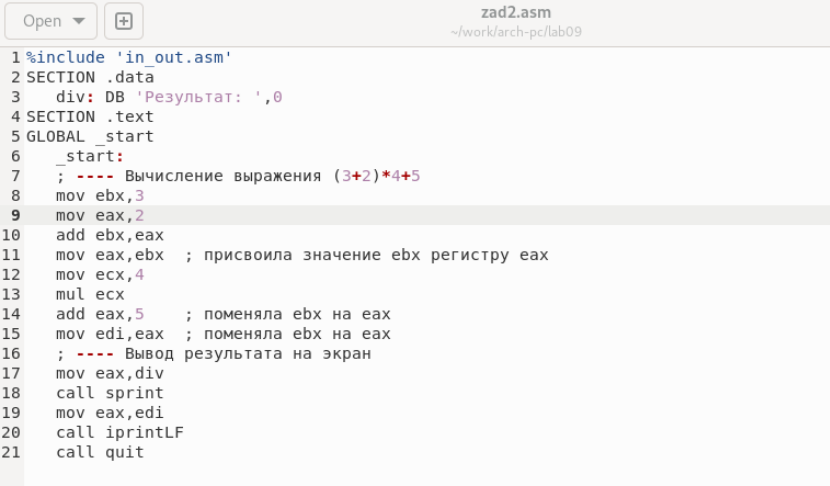
```

Рис. 4.18: Поиск причины ошибки

Исправила код таким образом: добавила после `add ebx,eax` команду `mov eax,ebx` и заменила `ebx` на `eax` в инструкциях `add ebx,5` и `mov edi,ebx`.

Создала исполняемый файл и запустила его (рис. 4.19). Теперь программы работает корректно.

```
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ nasm -f elf zad2.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ld -m elf_i386 -o zad2 zad2.o
vvmalyuga@Malyuga:~/work/arch-pc/lab09$ ./zad2
Результат: 25
```



```
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 ebx,3
9   mov eax,2
10  add ebx,eax
11  mov eax,ebx ; присвоила значение ebx регистру eax
12  mov ecx,4
13  mul ecx
14  add eax,5 ; поменяла ebx на eax
15  mov edi,eax ; поменяла ebx на eax
16  ; ---- Вывод результата на экран
17  mov eax,div
18  call sprint
19  mov eax,edi
20  call iprintlnLF
21  call quit
```

Рис. 4.19: Поиск причины ошибки

Прилагаю код:

```
%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 eax,ebx ; присвоила значение ebx регистру eax
    mov ecx,4
    mul ecx
    add eax,5 ; поменяла ebx на eax
    mov edi,eax ; поменяла ebx на eax
```

```
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.