

Отчёт по лабораторной работе №9

Архитектура компьютеров и операционные системы.

Брыляков Никита Евгеньевич

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	16
4.2.2	Работа с данными программы в GDB	17
4.2.3	Обработка аргументов командной строки в GDB	21
4.3	Задания для самостоятельной работы	23
5	Вывод	28
6	Список литературы	29

Список иллюстраций

4.1	Создание каталога и файла внутри	9
4.2	Ввод программы	10
4.3	Создание и запуск	11
4.4	Изменение программы	11
4.5	Создание и запуск	12
4.6	Создание и ввод	13
4.7	Получение файла	13
4.8	Загрузка исполняемого файла в отладчик	14
4.9	Проверка работы файла с помощью команды run	14
4.10	Установка брейкпоинта и запуск программы	14
4.11	Использование команд disassemble и disassembly-flavor intel	15
4.12	Включение режима псевдографики	16
4.13	Установление точек останова и просмотр информации о них	17
4.14	До использования команды stepi	18
4.15	После использования команды stepi	19
4.16	Просмотр значений переменных	19
4.17	Использование команды set	20
4.18	Вывод значения регистра	20
4.19	Использование команды set	21
4.20	Выход	21
4.21	Создание файла	22
4.22	Загрузка файла с аргументами в отладчик	22
4.23	Установление точки останова и запуск программы	22
4.24	Просмотр значений, введенных в стек	23
4.25	Написание кода подпрограммы	24
4.26	Запуск программы и проверка его вывода	25
4.27	Ввод программы	25
4.28	Поиск ошибки	26
4.29	Редактирование программы	27
4.30	Создание и запуск	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 останова.
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 UNIX-подобных 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 программу 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 `sl`) 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

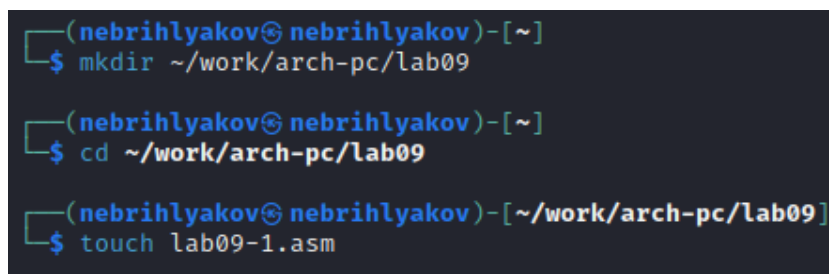
этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

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.

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm. (рис. [4.1]).



```
(nebrihlyakov@nebrihlyakov)~  
$ mkdir ~/work/arch-pc/lab09  
  
(nebrihlyakov@nebrihlyakov)~  
$ cd ~/work/arch-pc/lab09  
  
(nebrihlyakov@nebrihlyakov)~/work/arch-pc/lab09  
$ touch lab09-1.asm
```

Рис. 4.1: Создание каталога и файла внутри

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. [4.2]).

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax,x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax,result
23 call sprint
24 mov eax,[res]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx,2
32 mul ebx
33 add eax,7
34 mov [res],eax
35 ret ; выход из подпрограммы

```

Рис. 4.2: Ввод программы

Создаю исполняемый файл и проверяю его работу. (рис. [4.3]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-1.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-1 lab09-1.o

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ ./lab09-1
Введите x: 3
2x+7=13

```

Рис. 4.3: Создание и запуск

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. [4.4]).

```

23 call _subcalcul ; Вызов подпрограммы _calcul
24 call _calcul
25 mov eax,result
26 call sprint
27 mov eax,[res]
28 call iprintLF
29 call quit
30 ;
31 ; Подпрограмма вычисления
32 ; выражения "2x+7"
33 _calcul:
34 mov ebx,2
35 mul ebx
36 add eax,7
37 mov [res],eax
38 ret ; выход из подпрограммы
39 _subcalcul:
40 mov ebx,3
41 mul ebx
42 add eax,-1
43 ret

```

Рис. 4.4: Изменение программы

Создаю исполняемый файл и запускаю его. (рис. [4.5]).

```
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ nasm -f elf lab09-1.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-1 lab09-1.o

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ ./lab09-1
Введите x: 3
2x+7=23
```

Рис. 4.5: Создание и запуск

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. [4.6]).

```

1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6 SECTION .text
7 global _start
8 _start:
9 mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
14 mov eax, 4
15 mov ebx, 1
16 mov ecx, msg2
17 mov edx, msg2Len
18 int 0x80
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
22 |

```

Рис. 4.6: Создание и ввод

Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. [4.7]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ nasm -f elf -g -l lab09-2.lst lab09-2.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-2 lab09-2.o

```

Рис. 4.7: Получение файла

Загружаю исполняемый файл в отладчик gdb. (рис. [4.8]).

```
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ gdb lab09-2
GNU gdb (Debian 13.2-1) 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-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)
```

Рис. 4.8: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `run`. (рис. [4.9]).

```
(gdb) run
Starting program: /home/nebrihlyakov/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 1362105) exited normally]
(gdb)
```

Рис. 4.9: Проверка работы файла с помощью команды `run`

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start` и запускаю её. (рис. [4.10]).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/nebrihlyakov/work/arch-pc/lab09/lab09-2

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

Рис. 4.10: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. [4.11]).

```

(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.

```

Рис. 4.11: Использование команд disassemble и disassembly-flavor intel

В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется привычный нам синтаксис. Включаю режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs (рис. [4.12]).

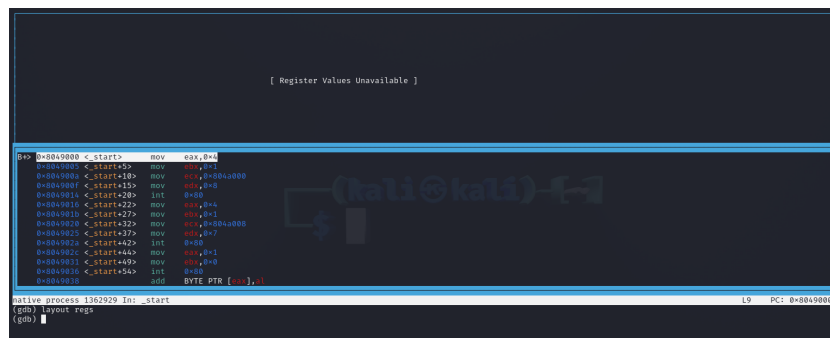


Рис. 4.12: Включение режима псевдографики

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. [4.13]).


```

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
      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

native process 1362929 In: _start
(gdb) layout regs
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
(gdb) b *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. 4.13: Установление точек останова и просмотр информации о них

4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. [4.14]).

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

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 1362929 In: _start
ebx      0x0      0
esp      0xffffd1a0 0xffffd1a0
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--fs    0x0
gs       0x0      0

```

Рис. 4.14: До использования команды stepi

(рис. [4.15]).

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43

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 1362929 In: _start L15 PC:
ebx      0x0
esp      0xffffd1a0 0xffffd1a0
ebp      0x0
esi      0x0
edi      0x0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--fs 0x0
gs       0x0

```

Рис. 4.15: После использования команды stepi

Изменились значения регистров eax, ecx, edx и ebx. Просматриваю значение переменной msg1 по имени с помощью команды x/lb &msg1 и значение переменной msg2 по ее адресу. (рис. [4.16]).

```

(gdb) x/lb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/lb 0x804a008
0x804a008 <msg2>: "world!\n\034"

```

Рис. 4.16: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. [4.17]).

```

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

```

Рис. 4.17: Использование команды set

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра `edx` с помощью команды `print p/F $val`. (рис. [4.18]).

```

(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'

```

Рис. 4.18: Вывод значения регистра

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием. (рис. [4.19]).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
```

Рис. 4.19: Использование команды set

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. [4.20]).

```
(gdb) c
Continuing.
world!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) q
A debugging session is active.

        Inferior 1 [process 1362929] will be killed.

Quit anyway? (y or n) █
```

Рис. 4.20: Выход

4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. [4.21]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ nasm -f elf -g -l lab09-3.lst lab09-3.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o lab09-3 lab09-3.o

```

Рис. 4.21: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `-args`. (рис. [4.22]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Debian 13.2-1) 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-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 ...

```

Рис. 4.22: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. [4.23]).

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/nebrihlyakov/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

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

```

Рис. 4.23: Установление точки останова и запуск программы

Просматриваю вершину стека и позиции стека по их адресам. Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4. (рис. [4.24]).

```

(gdb) x/x $esp
0xffffd150: 0x00000005
(gdb) x/x *(void**)( $esp+4)
0xffffd30b: 0x6d6f682f
(gdb) x/x *(void**)( $esp+8)
0xffffd339: 0x80d1b0d0
(gdb) x/x $esp
0xffffd150: 0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xffffd30b: "/home/nebrihlyakov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd339: "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd34b: "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xffffd35c: "2"
(gdb) x/s *(void**)( $esp + 20)
0xffffd35e: "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0: <error: Cannot access memory at address 0x0>

```

Рис. 4.24: Просмотр значений, введенных в стек

4.3 Задания для самостоятельной работы

1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. [4.25]).

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Ответ: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx
9 sub ecx,1
10 mov esi, 0
11 next:
12 cmp ecx,0h
13 jz _end
14 pop eax
15 call atoi
16 call _calcul
17 add esi,eax
18 loop next
19 _end:
20 mov eax, msg
21 call sprint
22 mov eax, esi
23 call iprintLF
24 call quit
25 _calcul:
26 mov ebx, 4
27 mul ebx
28 add eax, 3
29 ret
30

```

Рис. 4.25: Написание кода подпрограммы

Запускаю код и проверяю, что он работает корректно. (рис. [4.26]).


```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ touch z1.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ nasm -f elf z1.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o z1 z1.o

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab09]
$ ./z1 1 2 3 4
Ответ: 52

```

Рис. 4.26: Запуск программы и проверка его вывода

2. Ввожу в файл z2.asm текст программы из листинга 9.3. (рис. [4.27]).

```

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 ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ——— Вывод результата на экран
16 mov eax,div
17 call sprintf
18 mov eax,edi
19 call iprintLF
20 call quit
21 |

```

Рис. 4.27: Ввод программы

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды continue прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров. Нахожу ошибку. (рис. [4.28]).

```

nebrihlyakov@nebrihlyakov: ~/work/arch-pc/lab09
File Actions Edit View Help

eax 0x0 0
ecx 0x0 0
edx 0x0 0
ebx 0x0 0
ebp 0x3 [ Register Value 0x0Unavailable ]
esi 0x0 0
edi 0x0 0
eip 0x80490e8 0x80490e8 <_start>
rip 0x80490ed 0x80490ed <_start+5>
cs 0x23 35
ss 0x2b 43
ds 0x2b 43

B+ 0x80490e8 <_start> mov $0x3,%ebx
> 0x80490ed <_start+5> mov $0x2,%eax
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>
0x8049116 add %al,(%eax)
0x8049116 add %al,(%eax)
exe 0x8049116 In: add %al,(%eax)

native process 1466864 In: _start GDB text based interface.
(gdb) 0 process 1467184 In: _start L8 PC: 0x80490e8
Starting program: /home/nebrihlyakov/work/arch-pc/lab09/z2 L9 PC: 0x80490ed

Breakpoint 1, _start () at z2.asm:8
(gdb) c
Continuing.
Результат: 10
[Inferior 1 (process 1466864) exited normally]
(gdb) stepi
The program is not being run.
(gdb) run
Starting program: /home/nebrihlyakov/work/arch-pc/lab09/z2

Breakpoint 1, _start () at z2.asm:8
(gdb) stepi
(gdb)

```

Рис. 4.28: Поиск ошибки

Редактирую программу. (рис. [4.29]).

```

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
12 mov ecx,4
13 mul ecx
14 add eax,5
15 mov edi,eax
16 ; -- Вывод результата на экран
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 call quit
22

```

Рис. 4.29: Редактирование программы

Создаю исполняемый файл и запускаю (рис. [4.30]).

```

(nebrihlyakov@nebrihlyakov)~[~/work/arch-pc/lab09]
$ nasm -f elf z2.asm

(nebrihlyakov@nebrihlyakov)~[~/work/arch-pc/lab09]
$ ld -m elf_i386 -o z2 z2.o

(nebrihlyakov@nebrihlyakov)~[~/work/arch-pc/lab09]
$ ./z2
Результат: 25

```

Рис. 4.30: Создание и запуск

5 Вывод

При выполнении данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм и ознакомился с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

- [illegible]