

# **Отчёта по лабораторной работе №9**

**Понятие подпрограммы. Отладчик GDB.**

Павлюченков Сергей Витальевич

# Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	27
	Список литературы	28

# Список иллюстраций

3.1	Создание первого файла . . . . .	7
3.2	Программа из листинга 9.1 . . . . .	8
3.3	Запуск исполнительного файла . . . . .	9
3.4	Измененная программа . . . . .	10
3.5	Запуск исполнительного файла . . . . .	11
3.6	Создание файла . . . . .	11
3.7	Программа из листинга 9.2 . . . . .	12
3.8	Создание и загрузка исполнительного файла . . . . .	13
3.9	Запуск программы в отладчике . . . . .	13
3.10	Запуск программы в отладчике с брейкпоинтом . . . . .	13
3.11	Дисассимилированный код программы . . . . .	14
3.12	Дисассимилированный код программы с Intel'овским синтаксисом . . . . .	14
3.13	Точки останова . . . . .	15
3.14	Вторая точка останова . . . . .	15
3.15	Выполнение 5 инструкций . . . . .	16
3.16	Значение переменной msg1 . . . . .	16
3.17	Значение переменной msg2 . . . . .	17
3.18	Новое значение регистра msg1 . . . . .	17
3.19	Новое значение регистра msg2 . . . . .	17
3.20	Значение регистра в разных форматах . . . . .	18
3.21	Изменение значения регистра командой set . . . . .	18
3.22	Завершение выполнения программы и выход из GDB . . . . .	19
3.23	Загрузка файла с аргументами в отладчик . . . . .	20
3.24	Запуск отладчика . . . . .	20
3.25	Запуск отладчика . . . . .	21
3.26	Преобразованная программа . . . . .	22
3.27	Создание и запуск исполнительного файла . . . . .	22
3.28	Программа из листинга 9.3 . . . . .	23
3.29	Создание и запуск исполнительного файла . . . . .	23
3.30	Отладка . . . . .	24
3.31	Исправленная программа . . . . .	25
3.32	Создание и запуск исполнительного файла . . . . .	26

## Список таблиц

# 1 Цель работы

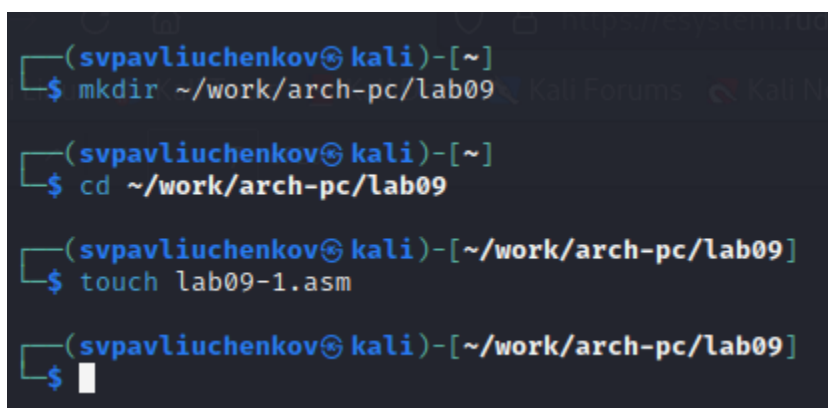
Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

## 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Задание для самостоятельной работы

### 3 Выполнение лабораторной работы

- Реализация циклов в NASM Создаю каталог для программ лабораторной работы № 9, перехожу в него и создаю файл lab9-1.asm (рис. 3.1).



```
(svpavliuchenkov@kali)-[~]  
$ mkdir ~/work/arch-pc/lab09  
  
(svpavliuchenkov@kali)-[~]  
$ cd ~/work/arch-pc/lab09  
  
(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]  
$ touch lab09-1.asm  
  
(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]  
$
```

Рис. 3.1: Создание первого файла

Ввожу в файл lab9-1.asm текст программы из листинга 9.1 (рис. 3.2).

```

GNU nano 7.2
#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

```

Рис. 3.2: Программа из листинга 9.1

Создаю исполняемый файл и проверяю его работу (рис. 3.3).



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

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

(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$ ./lab09-1
Введите x: 10
2x+7=27

(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$
```

Рис. 3.3: Запуск исполнительного файла

Эта программа считает и выводит выражение  $2 * x + 7$ , в котором  $x$  вводится пользователем.

Изменяю текст программы добавляя подпрограмму subcalcul(рис. 3.4).

```

GNU nano 7.2
#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:
call _subcalcul
mov ebx, eax
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
mov [res], eax
ret

```

Рис. 3.4: Измененная программа

Создаю исполняемый файл и проверяю его работу (рис. 3.5).

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

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

(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$ ./lab09-1
Введите x: 1
2x+7=11
```

Рис. 3.5: Запуск исполняемого файла

Программа работает корректно( $g(x)=3 * 1 - 1 = 2$ ,  $f(g(x)) = 2 * 2 + 7 = 11$ )

- Отладка программ с помощью GDB

Создаю файл lab9-2.asm (рис. 3.6)

```
(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$ touch lab09-2.asm
```

Рис. 3.6: Создание файла

Ввожу в файл lab9-2.asm текст программы из листинга 9.2 (рис. 3.7).

```
GNU nano 7.2
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. 3.7: Программа из листинга 9.2

Создаю исполняемый файл и загружаю его в отладчик GDB (рис. 3.8).

```

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

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

(svpavliuchenkov@kali)-[~/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. 3.8: Создание и загрузка исполнительного файла

Проверяю работу программы, запуская её в оболочке GDB (рис. 3.9).

```

(gdb) run
Starting program: /home/svpavliuchenkov/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 143224) exited normally]
(gdb)

```

Рис. 3.9: Запуск программы в отладчике

Устанавливаю брейкпоинт и опять запускаю программу (рис. 3.10).

```

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

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

```

Рис. 3.10: Запуск программы в отладчике с брейкпоинтом

Смотрю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `start` (рис. 3.11).

```

(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) █

```

Рис. 3.11: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`

```

(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. 3.12: Дисассимилированный код программы с Intel'овским синтаксисом

В режиме АТТ для обозначения регистров используется префикс %, а в режиме Intel префиксы для регистров не используются, они просто обозначаются именами регистров. В режиме АТТ для обозначения констант используется префикс \$, а в режиме Intel префикс \$ для обозначения констант не требуется, константы просто записываются как числа или адреса. При выводе дисассимилированного кода в режиме АТТ сначала идет константа, и потом регистр, а при выводе в режиме Intel сначала регистр, и потом константа.

Проверяю какая точка остановки у меня установлена (рис. 3.13).

```
(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. 3.13: Точки останова

Определяю адрес предпоследней строки программы из рисунка 12 и устанавливаю новую точку останова (рис. 3.14).

```
(gdb) break *0x08049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) █
```

Рис. 3.14: Вторая точка останова

Выполняю 5 инструкций с помощью команды stepi и прослеживаю за изменением значений регистров.

```

(gdb) ir
Undefined command: "ir". Try "help".
(gdb) i r
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd040     0xffffd040
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
fs             0x0            0
gs             0x0            0
(gdb) si 5
Hello, 14      mov eax, 4
(gdb) i r
eax            0x8            8
ecx            0x804a000     134520832
edx            0x8            8
ebx            0x1            1
esp            0xffffd040     0xffffd040
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
(gdb) █

```

Рис. 3.15: Выполнение 5 инструкций

Изменились значения регистров `eax`, `ecx`, `edx`, `ebx`, `eip`.

Смотрю значение переменной `msg1` по имени (рис. 3.16).

```

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

```

Рис. 3.16: Значение переменной `msg1`



Смотрю значение переменной msg2 по адресу (рис. 3.17).

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) █
```

Рис. 3.17: Значение переменной msg2

Изменяю значение регистра msg1 с помощью команды set, задав ей в качестве аргумента имя регистра (рис. 3.18).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) █
```

Рис. 3.18: Новое значение регистра msg1

‘H’ поменялся на ‘h’.

Изменяю значение регистра msg2 с помощью команды set, задав ей в качестве аргумента имя регистра (рис. 3.19).

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

Рис. 3.19: Новое значение регистра msg2

Вывожу значение регистра edx в разных форматах (рис. 3.20).

```
(gdb) p/f $msg2
$1 = void
(gdb) p/f $edx
$2 = 1.12103877e-44
(gdb) p/s $edx
$3 = 8
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb) █
```

Рис. 3.20: Значение регистра в разных форматах

Изменяю значение регистра `ebx` с помощью команды `set` (рис. 3.21).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb) █
```

Рис. 3.21: Изменение значения регистра командой `set`

Когда значение 2 (как целое число, без кавычек), gdb интерпретирует его как целое число и выводит его значение как 2, а когда с кавычками, то выводится номер соответствующий элементу в ASCII.

Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q) (рис. 3.22).

```
(gdb) c
Continuing.
sorld!

Breakpoint 2, _start () at lab09-2.asm:20
20      mov ebx, 0
(gdb) q
A debugging session is active.

        Inferior 1 [process 163463] will be killed.

Quit anyway? (y or n) y

(svpavliuchenkov@kali)~[~/work/arch-pc/lab09]
$
```

Рис. 3.22: Завершение выполнения программы и выход из GDB

- Обработка аргументов командной строки в GDB Скопировал файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки в файл с именем lab09-3.asm, создал исполняемый файл и загрузил исполняемый файл в отладчик, указав аргументы (рис. 3.23).

```

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

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

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

(svpavliuchenkov@kali)-[~/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 ...
(gdb)

```

Рис. 3.23: Загрузка файла с аргументами в отладчик

Устанавливаю брейкпоинт и запускаю отладчик (рис. 3.24).

```

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

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

```

Рис. 3.24: Запуск отладчика

Смотрю 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(рис. 3.25).

```

(gdb) x/x $esp
0xffffcfff: 0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xffffd1da: "/home/svpavliuchenkov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd20b: "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd21d: "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xffffd22e: "2"
(gdb) x/s *(void**)( $esp + 20)
0xffffd230: "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) x/s *(void**)( $esp)

```

Рис. 3.25: Запуск отладчика

Шаг изменения адреса равен 4 потому, что в архитектуре x86 данные обычно выравниваются по границе 4 байта.

- Задание для самостоятельной работы Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции  $f(x)$  как подпрограмму(рис. 3.26).

```

GNU nano 7.2
%include 'in_out.asm'
SECTION .data
msg1 db "Функция 12*x - 7", 0
msg2 db "Результат: ",0
newline db 10,0
SECTION .text
global _start
_start:
    pop ecx ; Извлекаем из стека в ecx количество
    pop edx ; Извлекаем из стека в edx имя программы
    sub ecx,1 ; Уменьшаем ecx на 1 (количество аргументов без названия программы)
    mov esi, 0 ; Устанавливаем начальное значение esi в 1 для хранения произведения
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
    add esi, eax;
    loop next ; переход к обработке следующего аргумента
_end:
    mov eax, msg1;
    call sprint;
    mov eax, newline
    call sprint
    mov eax, msg2 ; вывод сообщения "Результат: "
    call sprint;
    mov eax, esi ; записываем произведение в регистр eax
    call iprintf ; печать результата
    call quit ; завершение программы
_calcul:
    mov ebx, 12
    mul ebx
    add eax, -7
    ret

```

Рис. 3.26: Преобразованная программа

Запуск программы и проверки, что она работает правильно(рис. 3.27).

```

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

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

(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$ ./problem-1 1 2 3
Функция 12*x - 7
Результат: 51

```

Рис. 3.27: Создание и запуск исполнительного файла

Все работает правильно.

Ввожу в файл problem-2.asm текст программы из листинга 9.3 (рис. 3.28).

```

GNU nano 7.2
%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. 3.28: Программа из листинга 9.3

Запуск программы и проверка, что она работает некорректно (рис. 3.29).

```

$ nasm -f elf problem-2.asm

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

(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$ ./problem-2
Результат: 10

(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$

```

Рис. 3.29: Создание и запуск исполнительного файла

Отладка показа, что сначала в `ebx` записывается 3, потом в `eax` 2, после в `ebx` записывается 5 (сумма прошлых двух). после `ecx` становится 4 и `eax` становится 8

(2 \* 4), и наконец  $ebx = 10$  (рис. 3.30). Ошибка заключается в том, что умножение происходит на  $eax$ , а не на  $ebx$ .

```

Register group: general
eax      0x8      8      (gdb) x/s 0x8      0x8      0x4      4
edx      0x0      0      (gdb) x/s 0x0      0x0      0xa      10
esp      0xffffd030 0      0xffffd030      ebp      0x0      0x0
esi      0x0      0      (gdb) x/s 0x0      0x0      0x0      0
eip      0x80490fe 0x80490fe <_start+22>      eflags 0x206      [ PF IF ]
cs       0x23      35      (gdb) x/s 0x23      0x23      0x2b      43
ds       0x2b      43      (gdb) x/s 0x2b      0x2b      0x2b      43
fs       0x0      0      0xffffd1ed: gs <segment 3> 0x0      0
(gdb) x/s 0x0      0x0      0x0      0
(gdb) x/s 0x0      0x0      0x0      0
0x0: <error: Cannot access memory at address 0x0>
(gdb)

B+ 0x80490e8 <_start> mov $0x1, %ebx
0x80490ed <_start+5> mov $0x2, %ecx
0x80490f2 <_start+10> add %eax, %ecx
0x80490f4 <_start+12> mov $0x4, %ecx
0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19> add $0x5, %ecx
> 0x80490fe <_start+22> mov %ebx, %edi
0x8049100 <_start+24> mov $0x804a000, %eax
0x8049105 <_start+29> call 0x804900f <printf>
0x804910a <_start+34> mov $0d, %eax
0x804910c <_start+36> call 0x8049086 <printf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 <_start+46> add $0x1, %eax

Объясните, почему шаг изменения адреса равен 4 (esp+4), [esp
9.5. Задание для самостоятельной работы
1. Преобразуйте программу из лабораторной работы №6 (Задание 1) в программу, реализующую вычисление значения функции f(x) =
2. В листинге 9.3 приведена программа вычисления выражения (x + y) * z. Данная программа дает неверный результат. Проверьте это. С помощью анализатора изменения значений регистров, определите ошибку

```

Рис. 3.30: Отладка

Меняю умножение на `ebx` в программе. (рис. 3.31)



```
GNU nano 7.2
#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 eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ——— Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 3.31: Исправленная программа

Запускаю программу, чтобы проверить, что все верно. (рис. 3.32)

```
(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$ nasm -f elf problem-2.asm

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

(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
$ ./problem-2
Результат: 25

(svpavliuchenkov@kali)-[~/work/arch-pc/lab09]
```

Рис. 3.32: Создание и запуск исполняемого файла

В этот раз программа выводит верный ответ - 25.

## 4 Выводы

Выполнив эту работу я научился писать и использовать подпрограммы и ознакомился с отладчиком GDB, и его инструментами и функциями.

## Список литературы

::: Лабораторная работа №9 ::: GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/> ::: NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/> :::