

Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Шабалина Елизавета Андреевна

Содержание

1	Цель работы.....	1
2	Задание	1
3	Теоретическое введение	2
4	Выполнение лабораторной работы.....	2
4.1	Реализация программ в NASM.....	2
4.2	Отладка программ с помощью GDB	5
4.3	Добавление точек останова	10
4.4	Работа с данными программы в GDB	11
4.5	Обработка аргументов командной строки в GDB	13
5	Задания для самостоятельной работы	14
6	Выводы	18
	Список литературы.....	18

1 Цель работы

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

2 Задание

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $\square(\square)$ как подпрограмму.
2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

3 Теоретическое введение

4 Выполнение лабораторной работы

4.1 Реализация программ в NASM

1. Создаю каталог для программ лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm.

```
eashabalina@dk2n27 ~ $ mkdir ~/work/arch-pc/lab09
eashabalina@dk2n27 ~ $ cd ~/work/arch-pc/lab09
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ touch lab09-1.asm
eashabalina@dk2n27 ~/work/arch-pc/lab09 $
```

Рис. 1: Переход в каталог и создание файла

2. В качестве примера рассмотрим программу вычисления арифметического выражения $\square(\square) = 2\square + 7$ с помощью подпрограммы `_calcul`. В данном примере `□` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Запускаю исполняемый файл.

```

lab09-1.asm      [-M--]  0 L:[ 1+22 23[*][
%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:

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

_calcul:

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

ret ; выход из подпрограммы

```

Рис. 2: Программа с использованием подпрограммы

```
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ ld -m elf_i386 lab09-1.o -o lab09-1
ld: не распознан режим эмуляции: elf_i386
Поддерживаемые эмуляции: elf_x86_64 elf32_x86_64 elf_i386 elf_iamcu
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ ld -m elf_i386 lab09-1.o -o lab09-1
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=17
eashabalina@dk2n27 ~/work/arch-pc/lab09 $
```

Рис. 3: Исполнение программы из листинга 9.1

- Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\square(\square(\square))$, где \square вводится с клавиатуры, $\square(\square) = 2\square + 7$, $\square(\square) = 3\square - 1$. Запустим исправленную программу. Число проходов цикла не соответствует значению, введенному с клавиатуры.

```
_calcul:

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

ret

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

Рис. 4: Исправленный текст программы `lab09-1.asm`

```
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ ld -m elf_i386 lab09-1.o -o lab09-1
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 6
2x+7=19
```

Рис. 5: Исполнение программы `lab09-1`

4.2 Отладка программ с помощью GDB

4. Создаем файл lab09-2.asm. Вводим в него программу из листинга 9.2. Транслируем текст программы с ключом '-g'. Загружаем исполняемый файл в gdb.

lab09-2.asm [-M--]

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. 6: Текст программы из листинга 9.2

```
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) □
```

Рис. 7: Исполнение программы

5. Проверим работу программы, запустив ее в оболочке отладчика. Ошибок не обнаружено.

```
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/a/eashabalina/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 22456) exited normally]
(gdb) □
```

Рис. 8: Запуск программы в оболочке отладчика

6. Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её.

```
[Inferior 1 (process 22456) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 14.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/a/eashabalina/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:14
14      mov eax, 4
(gdb) □
```

Рис. 9: Исполнение программы с брейкпоинт

7. Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`.

```

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

```

Рис. 10: Просмотр дисассимилированного кода программы

8. Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом `$`; Intel - Размер операндов неявно определяется контекстом, как `ax`, `eax`, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом `%`, Intel - имена регистров пишутся без префиксов).


```

(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. 11: Просмотр дисассимилированного кода программы с синтаксисом Intel

9. Включим режим псевдографики для более удобного анализа программы.

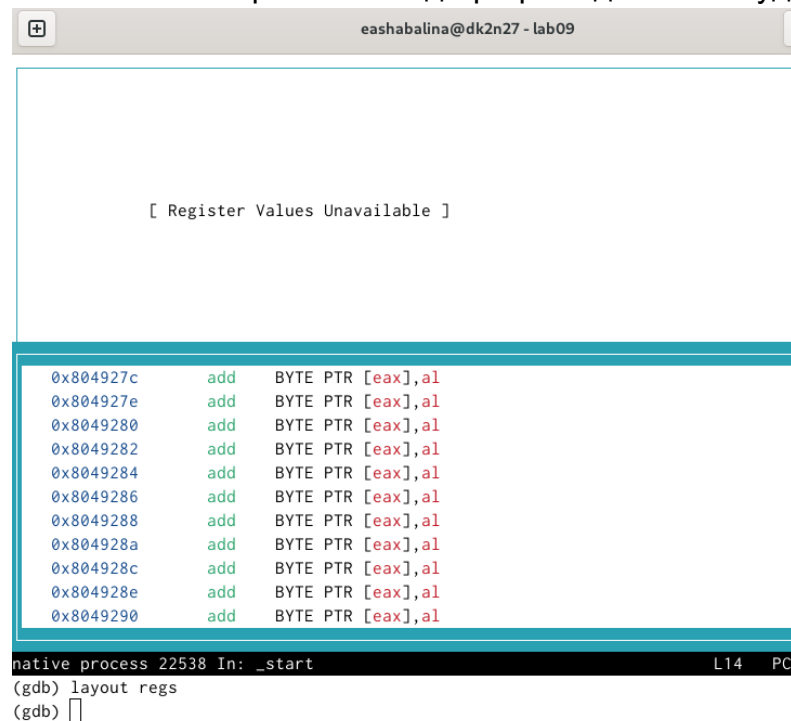


Рис. 12: Переход в режим псевдографики

4.3 Добавление точек останова

10. Проверим установку точки останова на метке '_start'.

```
8+>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 23073 In: _start L14
(gdb) layout regs
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab09-2.asm:14
breakpoint already hit 1 time
(gdb) □
```

Рис. 13: Проверка установки точки останова

11. Установим еще одну точку останова по адресу инструкции. Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова.

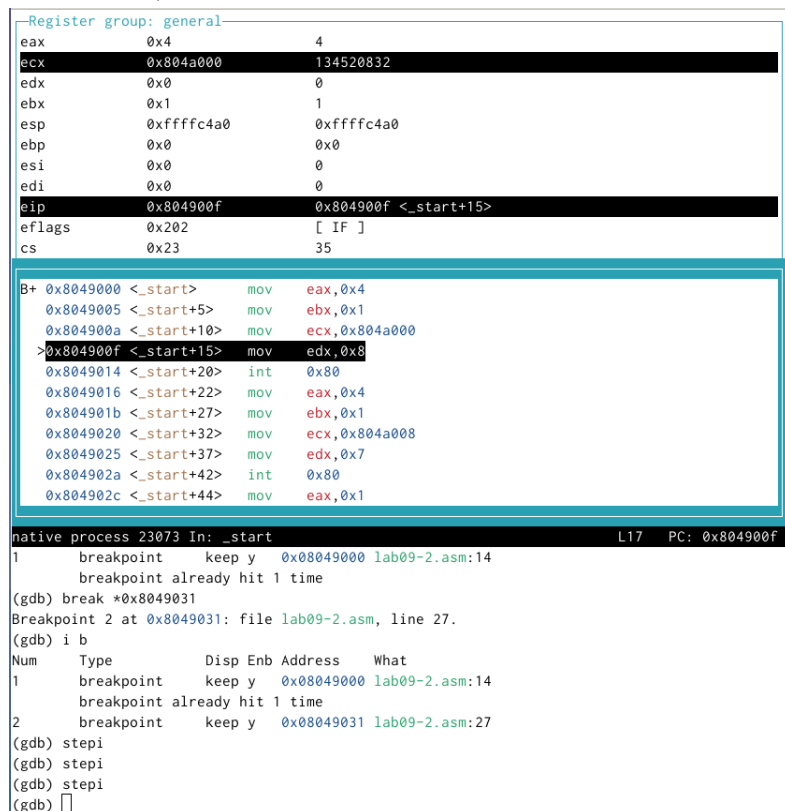
```
0x804900f <_start+15>    mov     edx,0x8
0x8049014 <_start+20>    int     0x80
0x8049016 <_start+22>    mov     eax,0x4
0x804901b <_start+27>    mov     ebx,0x1
0x8049020 <_start+32>    mov     ecx,0x804a008
0x8049025 <_start+37>    mov     edx,0x7
0x804902a <_start+42>    int     0x80
0x804902c <_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 23073 In: _start L14
(gdb) layout regs
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab09-2.asm:14
breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 27.
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 lab09-2.asm:14
breakpoint already hit 1 time
2        breakpoint    keep y  0x08049031 lab09-2.asm:27
(gdb) □
```

Рис. 14: Установка новой точки останова

4.4 Работа с данными программы в GDB

12. Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследим за изменением значений регистров. Изменяются значения регистров `eax`, `ebx`, `ecx`, `edx`.



The screenshot shows the GDB interface with the 'Register group: general' window at the top. It lists registers `eax` through `cs` with their current values. Below this, the assembly window shows instructions starting from `0x8049000`. The instruction `mov edx, 0x8` at address `0x804900f` is highlighted. The bottom window shows the GDB prompt with the command `stepi` being entered.

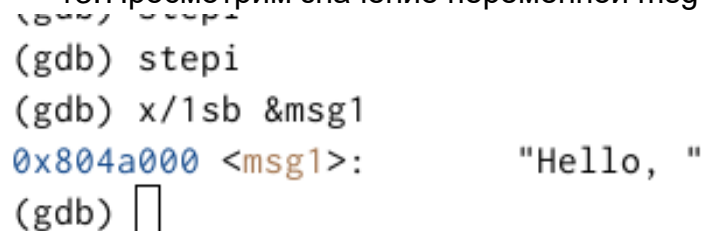
```
Register group: general
eax      0x4          4
ecx      0x804a000    134520832
edx      0x0          0
ebx      0x1          1
esp      0xffffc4a0   0xffffc4a0
ebp      0x0          0
esi      0x0          0
edi      0x0          0
eip      0x804900f    0x804900f <_start+15>
eflags   0x202        [ IF ]
cs       0x23         35

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 23073 In: _start L17 PC: 0x804900f
1 breakpoint keep y 0x08049000 lab09-2.asm:14
  breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 27.
(gdb) i b
Num   Type      Disp Enb Address  What
1     breakpoint keep y 0x08049000 lab09-2.asm:14
      breakpoint already hit 1 time
2     breakpoint keep y 0x08049031 lab09-2.asm:27
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb)
```

Рис. 15: Инструкция `stepi`

13. Просмотрим значение переменной `msg1` по имени.

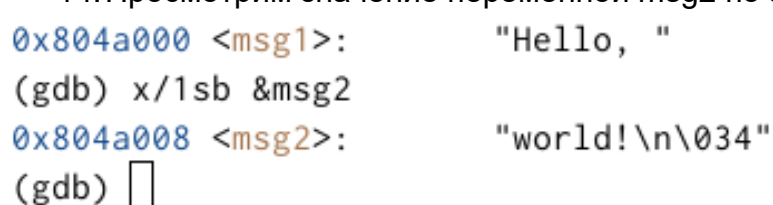


The screenshot shows the GDB prompt with the command `x/1sb &msg1`. The output displays the memory address `0x804a000` and the value `<msg1>: "Hello, "`.

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

Рис. 16: Просмотр значения переменной

14. Просмотрим значение переменной `msg2` по адресу.



The screenshot shows the GDB prompt with the command `x/1sb &msg2`. The output displays the memory address `0x804a008` and the value `<msg2>: "world!\n\034"`.

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

Рис. 17: Просмотр значения переменной

15. Изменим первый символ переменной msg1. Заменяем любой символ во второй переменной msg2.

```
0x804a000 <msg1>:      "Hello, "  
(gdb) set {char}&msg1='h'  
(gdb) x/1sb &msg1  
0x804a000 <msg1>:      "hello, "  
(gdb) set {char}0x804a008='L'  
(gdb) x/1sb &msg2  
0x804a008 <msg2>:      "Lor!d!\n\034"  
(gdb) 
```

Рис. 18: Изменение переменной

16. Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра ebx. С помощью команды set изменим значение регистра ebx. В первом случае программа выводит значение кодировки символа '2' в шестнадцатеричной системе, а во втором переводит цифру 2 в шестнадцатеричный вид. Завершаю выполнение программы и выхожу из отладчика.

```
(gdb) p/s $ebx  
$3 = 50  
(gdb) p/s $ebx  
$4 = 50  
(gdb) p/s $ebx  
$5 = 50  
(gdb) set $ebx=2  
(gdb) p/s $ebx  
$6 = 2  
(gdb) p/s $ebx  
$7 = 2  
(gdb) 
```

Рис. 19: Вывод значения регистра

4.5 Обработка аргументов командной строки в GDB

17. Скопируем файл lab8-2.asm в файл с именем lab09-3.asm. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы.

```
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/
lab09/lab09-3.asm
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент
3'
GNU gdb (Gentoo 14.2 vanilla) 14.2
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

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

Рис. 20: Загрузка файла lab09-3.asm в отладчик

18. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 7 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и 'аргумент 3'.

```
Reading symbols from lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/a/eashabalina/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:7
7      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xffffc460: 0x00000005
(gdb) █
```

Рис. 21: Проверка стека

19. Посмотрим 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. Шаг изменения адреса равен 4 байтам, потому что мы работаем с 32-битной системой (x86), а указатели (void **) в такой системе занимают 4 байта. Ошибка Cannot access memory at address 0x0 на \$esp + 24 указывает на то, что закончились аргументы командной строки.

```

Breakpoint 1, _start () at lab09-3.asm:7
7      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffffc460: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc6b6: "/afs/.dk.sci.pfu.edu.ru/home/e/a/eashabalina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc6fe: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc710: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc721: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc723: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 22: Проверка остальных позиций стека

5 Задания для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $\square(\square)$ как подпрограмму.

lab09-3.asm [-M--] 0 L:[1+23

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg1 db "Функция:  $f(x) = 5 \cdot (2+x)$ ",0
```

```
msg2 db "Результат: ",0
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
pop ecx
```

```
pop edx
```

```
sub ecx, 1
```

```
mov esi, 0
```

```
next:
```

```
cmp ecx, 0h
```

```
jz _end
```

```
pop eax
```

```
push ecx
```

```
call atoi
```

```
push eax
```

```
call _calcul
```

```
add esi, eax
```

```
pop eax
```

```
pop ecx
```

```
loop next
```

```
_end:
```

```
mov eax, msg1
```

```
call sprint
```

```
mov eax, msg2
```

```
call sprint
```

```
mov eax, esi
```

```
call iprintLF
```

```
call quit
```

```
_calcul:
```

```
push ebp
```

```
mov ebp, esp
```

```
mov ebx, [ebp+8]
```

Рис. 23: Текст программы lab09-3.asm

2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат. Ошибка в программе заключается в том, что инструкция `mul esx` умножает значение в регистре `eax` на `esx`, а результат записывает в `eax`. В исправленном варианте мы используем `ebx` для хранения промежуточного результата суммы, `mul esx` умножает `ebx` на `esx`, результат сохраняется в `eax`. Затем к результату в `eax` добавляется 5. Финальный результат сохраняется в `edi` и выводится на экран. Запускаем программу в режиме отладчика и пошагово через `si` просматриваем изменение значений регистров через `i r`. При выполнении инструкции `mul esx` можно заметить, что результат умножения записывается в регистр `eax`, но также меняет и `edx`. Значение регистра `ebx` не обновляется напрямую, поэтому программа неверно подсчитывает результат функции. Исправляем найденную ошибку, теперь программа верно считает значение функции.


```
eashabalina@dk2n27 - lab09 x easha

0x8049000 <slen>      push    ebx
0x8049001 <slen+1>     mov     ebx, eax
0x8049003 <nextchar>   cmp     BYTE PTR [eax], 0x0
0x8049006 <nextchar+3> je      0x804900b <finished>
0x8049008 <nextchar+5> inc     eax
0x8049009 <nextchar+6> jmp     0x8049003 <nextchar>
0x804900b <finished>  sub     eax, ebx
0x804900d <finished+2> pop     ebx
0x804900e <finished+3> ret
0x804900f <sprint>     push    edx
0x8049010 <sprint+1>     push    ecx
0x8049011 <sprint+2>     push    ebx
0x8049012 <sprint+3>     push    eax
0x8049013 <sprint+4>     call    0x8049000 <slen>
0x8049018 <sprint+9>     mov     edx, eax
0x804901a <sprint+11>    pop     eax
0x804901b <sprint+12>    mov     ecx, eax
0x804901d <sprint+14>    mov     ebx, 0x1
0x8049022 <sprint+19>    mov     eax, 0x4
0x8049027 <sprint+24>    int     0x80
0x8049029 <sprint+26>    pop     ebx
0x804902a <sprint+27>    pop     ecx
0x804902b <sprint+28>    pop     edx
0x804902c <sprint+29>    ret
0x804902d <sprintLF>   call    0x804900f <sprint>
0x8049032 <sprintLF+5> push    eax

exec No process in:
(gdb) ./lab09-3
Undefined command: ".". Try "help".
(gdb) 
```

Рис. 24: Поиск ошибок в работе программы lab09-5.asm

```
eashabalina@dk2n27 - lab09 x easha

eashabalina@dk2n27 ~/work/arch-pc/lab09 $ ./lab09-3
Результат: 10
eashabalina@dk2n27 ~/work/arch-pc/lab09 $ 
```

Рис. 25: Запуск исправленной программы

```
eashabalina@dk2n27 - lab09
lab09-3.asm [----] 11 L:[ 1+ 9
#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. 26: Исправленный текст программы

6 Выводы

В результате выполнения лабораторной работы я приобрёл навыки написания программ с использованием циклов и обработкой аргументов командной строки в NASM.

Список литературы