

Отчет по выполнению лабораторной работы №8

Дисциплина: архитектура компьютеров

Намруев Максим Саналович

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	Реализация подпрограмм в NASM	7
3.2	Отладка программ с помощью GDB	11
3.2.1	Добавление точек останова	15
3.2.2	Работа с данными программы в GDB	16
3.2.3	Обработка аргументов командной строки в GDB	20
4	Задание для самостоятельной работы.	23
5	Выводы	29
	Список литературы	30

Список иллюстраций

3.1	Создание каталога и файла	7
3.2	Ввод программы из листинга 9.1	8
3.3	Проверка работы программы	9
3.4	Изменение текста программы	10
3.5	Запуск программы	11
3.6	Создание файла печати сообщения Hello world!	11
3.7	Получение и загрузка в отладчик исполняемого файла	12
3.8	Установка брейкпоинта	12
3.9	Просмотр дисассимилированного кода программы	13
3.10	Переключение на Intel'ский синтаксис	14
3.11	Включение режима псевдографики	15
3.12	Проверка точки останова	15
3.13	Установка точки останова и её проверка	16
3.14	До использования команды stepi	17
3.15	После использования команды stepi	18
3.16	Просмотр значения переменной msg1	18
3.17	Просмотр значения переменной msg2	18
3.18	Изменение первого символа msg1	19
3.19	Изменение первого символа переменной msg2	19
3.20	Вывод значений регистра ebx	19
3.21	Изменение значения регистра ebx	20
3.22	Копирование файла	20
3.23	Создание исполняемого файла	20
3.24	Загрузка исполняемого файла в отладчик	21
3.25	Установка точки останова	21
3.26	Просмотр вершины стека	21
3.27	Просмотр остальных позиций стека	22
4.1	Редактирование программы	23
4.2	Проверка работы программы	24
4.3	Ввод программы из листинга 9.3	25
4.4	Запуск программы	26
4.5	Просмотр программы	26
4.6	Исправление ошибки	27
4.7	Просмотр программы	27

Список таблиц

1 Цель работы

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

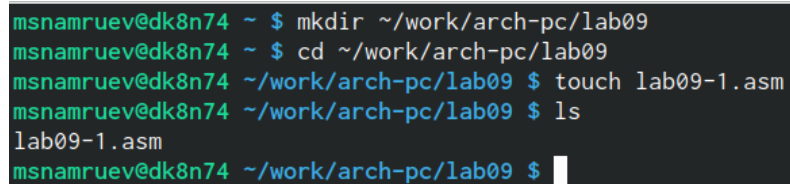
2 Задание

1. Релизация 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. Задание для самостоятельной работы.

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9, перехожу в него и создаю файл lab09-1.asm. (рис. [3.1]).

A screenshot of a terminal window with a dark background and light green text. It shows a series of commands being entered and executed. The prompt is 'msnamruev@dk8n74'. The commands are: 'mkdir ~/work/arch-pc/lab09', 'cd ~/work/arch-pc/lab09', 'touch lab09-1.asm', and 'ls'. The output of 'ls' is 'lab09-1.asm'.

```
msnamruev@dk8n74 ~ $ mkdir ~/work/arch-pc/lab09
msnamruev@dk8n74 ~ $ cd ~/work/arch-pc/lab09
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ touch lab09-1.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ ls
lab09-1.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Создание каталога и файла

Ввожу в файл lab09-1.asm текст программы из листинга 9.1.(рис. [3.2])

```

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. 3.2: Ввод программы из листинга 9.1

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


```
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ gedit lab09-1.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 10
2x+7=27
msnamruev@dk8n74 ~/work/arch-pc/lab09 $
```

Рис. 3.3: Проверка работы программы

Изменяю текст программы, добавляя подпрограмму `_subcalcul` в подпрограммы `_calcul`. (рис. [3.4])

```
Открыть  lab09-1.asm
~/work/arch-pc/lab09

1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+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 call _subcalcul
32 mov ebx, 2
33 mul ebx
34 add eax, 7
35 mov [res], eax
36 ret ; выход из подпрограммы
37 _subcalcul:
38 mov ebx, 3
39 mul ebx
40 sub eax, 1
41 ret
```

Рис. 3.4: Изменение текста программы

Создаю исполняемый файл и запускаю его.(рис. [3.5])

```

msnamruev@dk8n74 ~/work/arch-pc/lab09 $ gedit lab09-1.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 10
2(3x-1)+7=65
msnamruev@dk8n74 ~/work/arch-pc/lab09 $

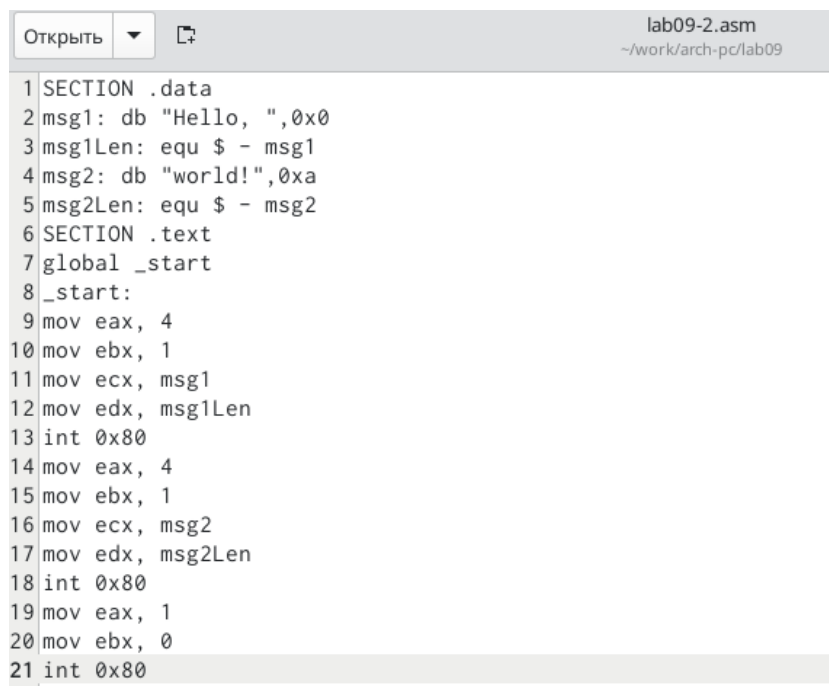
```

Рис. 3.5: Запуск программы

Программа работает корректно.

3.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2.(рис. [3.6])



```

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

```

Рис. 3.6: Создание файла печати сообщения Hello world!

Получаю исполняемый файл.Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'.Далее загружаю исполняемый файл в отладчик gdb и запускаю его с помощью команды run.(рис. [3.7])

```

msnamruev@dk8n74 ~/work/arch-pc/lab09 $ touch lab09-2.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ gedit lab09-2.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
msnamruev@dk8n74 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 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-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) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/s/msnamruev/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4284) exited normally]
(gdb)

```

Рис. 3.7: Получение и загрузка в отладчик исполняемого файла

Устанавливаю брейкпоинт на метку `_start` и запускаю программу.(рис. [3.8])

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/s/msnamruev/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 3.8: Установление брейкпоинта

Смотрю дисассимилированный код программы с помощью команды `disassemble` начинаю с метки `_start` .(рис. [3.9])

```

(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. 3.9: Просмотр дисассимилированного кода программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. (рис. [3.10])

```

(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. 3.10: Переключение на Intel'ский синтаксис

Отличия заключаются в том, что в режиме АТТ используются “%” перед именами регистров и “\$” перед именами операндов, а в режиме Intel используется обычный синтаксис.

Включаю режим псевдографики для более удобного анализа программы.(рис. [3.11])

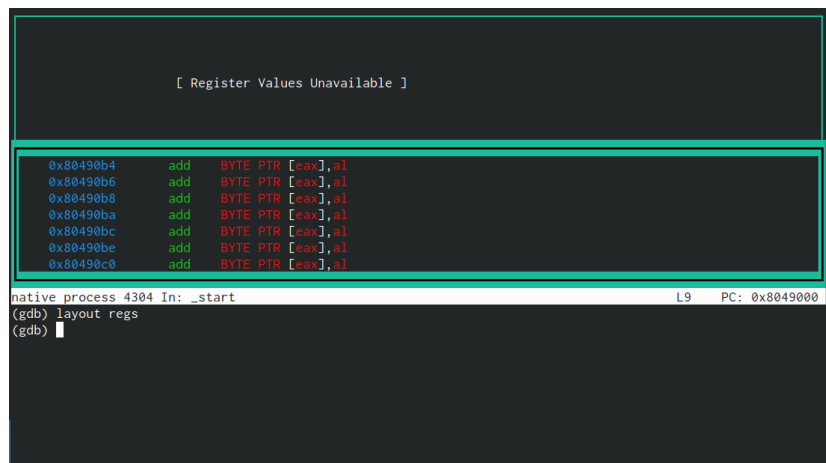


Рис. 3.11: Включение режима псевдографики

3.2.1 Добавление точек останова

Проверяю точку останова по имени метки.(рис. [3.12])

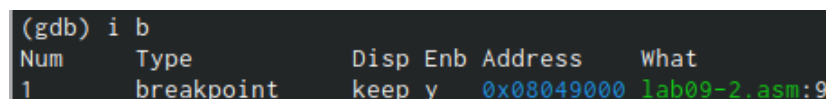


Рис. 3.12: Проверка точки останова

Определяю адрес предпоследней инструкции и устанавливаю точку останова. Далее смотрю информацию о всех установленных точках останова.(рис. [3.13])

```
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx
0x804902a <_start+42> int $0x80
0x804902c <_start+44> mov $0x1,%eax
b+ 0x8049031 <_start+49> mov $0x0,%ebx
0x8049036 <_start+54> int $0x80

exec No process in:
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
(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
2        breakpoint keep y  0x08049031 lab09-2.asm:20
(gdb) █
```

Рис. 3.13: Установка точки останова и её проверка

3.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением регистров.(рис. [3.14]) (рис. [3.15])


```

Register group: general
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffc320 0xffffc320  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

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
0x804903a add     BYTE PTR [eax],al

native process 4883 In: _start L9 P
(gdb) i r
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc320 0xffffc320
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)

```

Рис. 3.14: До использования команды stepi

```

REGISTER GROUP: General
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffc320  0xffffc320  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
0x8049036 <_start+54>     int     0x80
0x8049038      add     BYTE PTR [eax],al
0x804903a      add     BYTE PTR [eax],al

native process 4883 In: _start
(gdb) i r
eax      0x8      8
ecx      0x804a000  134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc320  0xffffc320
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: После использования команды stepi

Изменились регистры eax,ebx,ecx,edx

Просматриваю значение переменной msg1 по имени.(рис. [3.16])

```

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

```

Рис. 3.16: Просмотр значения переменной msg1

Также просматриваю значение переменной msg2.(рис. [3.17])

```

(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n"<error: Cannot access memory at address 0x804a00f>
(gdb)

```

Рис. 3.17: Просмотр значения переменной msg2

Изменяю первый символ переменной msg1.(рис. [3.18])

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

Рис. 3.18: Изменение первого символа msg1

Изменяю первый символ переменной msg2.(рис. [3.19])

```
(gdb) set {char}0x804a008='t'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "torld!\n\034"
(gdb)
```

Рис. 3.19: Изменение первого символа переменной msg2

Вывожу в различных форматах значение регистра ebx.(рис. [3.20])

```
(gdb) p/s $edx
$1 = 8
(gdb) p/t $edx
$2 = 1000
(gdb) p/x $edx
$3 = 0x8
(gdb)
```

Рис. 3.20: Вывод значений регистра ebx

С помощью команды set изменяю значение регистра ebx.(рис. [3.21])

```

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

```

Рис. 3.21: Изменение значения регистра ebx

Разница в том, что в первый раз мы вводим символ, который преобразуется в число, а во второй раз мы вводим само число.

3.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы No8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm.(рис. [3.22])

```

msnamruev@dk3n65 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
msnamruev@dk3n65 ~/work/arch-pc/lab09 $ 

```

Рис. 3.22: Копирование файла

Создаю исполняемый файл.(рис. [3.23])

```

msnamruev@dk3n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
msnamruev@dk3n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
msnamruev@dk3n65 ~/work/arch-pc/lab09 $ 

```

Рис. 3.23: Создание исполняемого файла

Заргружаю исполняемый файл в отладчик, указав аргументы.(рис. [3.24])

```
msnamruev@dk3n65 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 12.1 vanilla) 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-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. 3.24: Загрузка исполняемого файла в отладчик

Для начала устанавливаю точку останова перед первой инструкцией в программе и запускаю её.(рис. [3.25])

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/s/msnamruev/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент 3
\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop еск ; Извлекаем из стека в 'еск' количество
```

Рис. 3.25: Установка точки останова

Просматриваю вершину стека, то есть число аргументов строки(включая имя программы).(рис. [3.26])

```
(gdb) x/x $esp
0xfffffc2d0: 0x00000005
```

Рис. 3.26: Просмотр вершины стека

Просматриваю остальные позиции стека.(рис. [3.27])

```

(gdb) x/s *(void**)(esp + 4)
0xfffffc56b:    "/afs/.dk.sci.pfu.edu.ru/home/m/s/msnamruev/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc5b1:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc5c3:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc5d4:    "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc5d6:    "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:    <error: Cannot access memory at address 0x0>

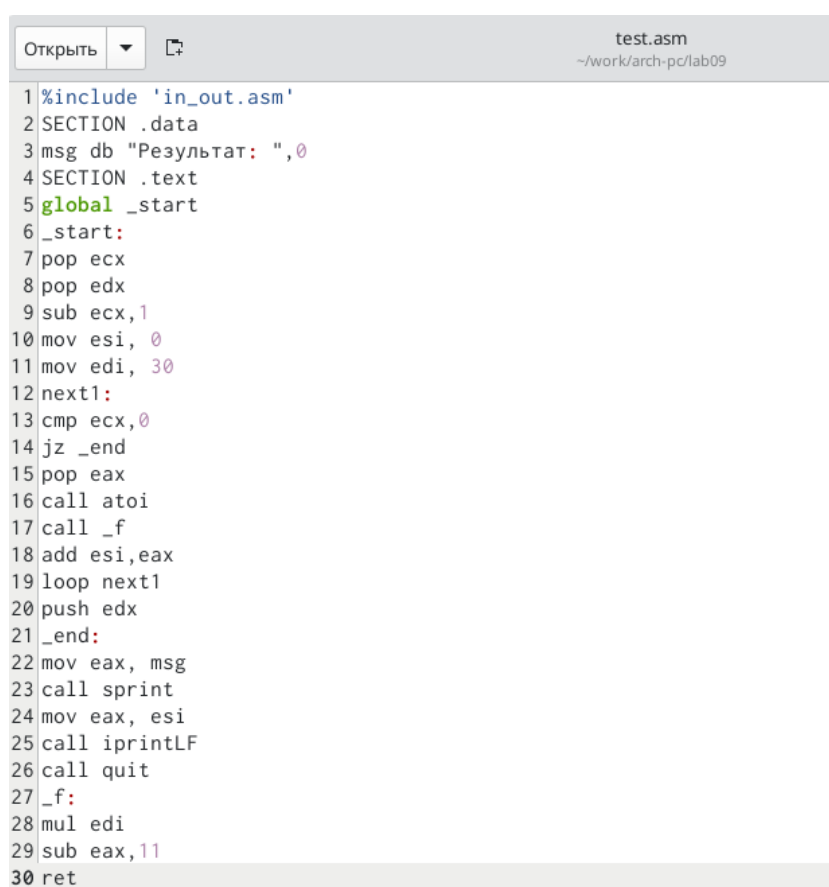
```

Рис. 3.27: Просмотр остальных позиций стека

Шаг изменения адреса равен 4, потому что значение регистра esp в стеке увеличивается на 4.

4 Задание для самостоятельной работы.

1. Открываю программу из лабораторной работы №8 и начинаю её редактировать.(рис. [3.19])



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx
9 sub ecx,1
10 mov esi, 0
11 mov edi, 30
12 next1:
13 cmp ecx,0
14 jz _end
15 pop eax
16 call atoi
17 call _f
18 add esi,eax
19 loop next1
20 push edx
21 _end:
22 mov eax, msg
23 call sprint
24 mov eax, esi
25 call iprintLF
26 call quit
27 _f:
28 mul edi
29 sub eax,11
30 ret
```

Рис. 4.1: Редактирование программы

Создаю исполняемый файл и запускаю его, чтобы проверить работу программы.(рис. [4.2])

```

msnamruev@dk3n38 ~/work/arch-pc/lab09 $ nasm -f elf test.asm
msnamruev@dk3n38 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o test test.o
msnamruev@dk3n38 ~/work/arch-pc/lab09 $ ./test 1 2
Результат: 68
msnamruev@dk3n38 ~/work/arch-pc/lab09 $ ./test 1 2 3
Результат: 147
msnamruev@dk3n38 ~/work/arch-pc/lab09 $ ./test 10 2 3 6 9
Результат: 845

```

Рис. 4.2: Проверка работы программы

Программа работает верно.

Текст программы:

```

#include 'in_out.asm'

SECTION .data
msg db "Результат:",0

SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi, 30
next1:
cmp ecx,0
jz _end
pop eax
call atoi
call _f
add esi,eax
loop next1
push edx
_end:

```



```

mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
_f:
mul edi
sub eax, 11
ret

```

2. Создаю файл test3.asm и ввожу туда текст программы из листинга 9.3.(рис. [4.3])

```

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 sprint
18 mov eax, edi
19 call iprintLF
20 call quit

```

Рис. 4.3: Ввод программы из листинга 9.3

Создаю исполняемый файл и запускаю его.(рис. [4.4])

```

msnamruev@dk3n38 ~/work/arch-pc/lab09 $ nasm -f elf test3.asm
msnamruev@dk3n38 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o test3 test3.o
msnamruev@dk3n38 ~/work/arch-pc/lab09 $ ./test3
Результат: 10

```

Рис. 4.4: Запуск программы

Убеждаюсь, что программа работает неверно.

Создаю исполняемый файл для работы с GDB и запускаю его через режим отладки. Создаю брейкпойнт и пошагово просматриваю программу.(рис. [4.5])

```

Register group: general
eax 0x804a000 134520832 ecx 0x4 4
edx 0x0 0 ebx 0xa 10
esp 0xffffc218 0xffffc218 ebp 0x0 0x0
esi 0x0 0 edi 0xa 10
eip 0x8049010 0x8049010 <sprint+1> eflags 0x206 [ PF IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

0x804900f <sprint> push %edx
> 0x8049010 <sprint+1> push %ecx
0x8049011 <sprint+2> push %ebx
0x8049012 <sprint+3> push %eax
0x8049013 <sprint+4> call 0x8049000 <len>
0x8049018 <sprint+9> mov %eax,%edx
0x804901a <sprint+11> pop %eax
0x804901b <sprint+12> mov %eax,%ecx
0x804901d <sprint+14> mov $0x1,%ebx
0x8049022 <sprint+19> mov $0x4,%eax
0x8049027 <sprint+24> int $0x80

native process 24231 In: sprint
(gdb) si
Breakpoint 3, _start () at test3.asm:10
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Рис. 4.5: Просмотр программы

Замечаю, что после выполнения инструкции mul программы умножит 4 на 2 на не на 5, как должно быть.Из-за этого программа выдает неверный результат.

Далле открываю файл с программой и исправляю ошибку.(рис. [4.6])

```

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

```

Рис. 4.6: Исправление ошибки

Создаю исполняемый файл и запускаю его.(рис. [4.7])

```

msnamruev@dk3n38 ~/work/arch-pc/lab09 $ nasm -f elf test3.asm
msnamruev@dk3n38 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o test3 test3.o
msnamruev@dk3n38 ~/work/arch-pc/lab09 $ ./test3
Результат: 25

```

Рис. 4.7: Просмотр программы

Теперь программа работает верно.

Текст программы:

```

#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
```

5 Выводы

После выполнения данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.

Список литературы