

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Ицков Андрей Станиславович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Релаксация подпрограмм в NASM	8
4.1.1	Отладка программ с помощью GDB	11
4.1.2	Добавление точек останова	14
4.1.3	Работа с данными программы в GDB	15
4.1.4	Обработка аргументов командной строки в GDB	19
4.2	Задание для самостоятельной работы	21
5	Выводы	26
6	Список литературы	27

Список иллюстраций

4.1	Создание рабочего каталога	8
4.2	Текст программы	8
4.3	Запуск программы	9
4.4	Текст программы	9
4.5	Запуск программы	11
4.6	Текст программы	12
4.7	Запуск программы	12
4.8	Добавление брейкпоинта	13
4.9	Дисассимилирование программы	13
4.10	Режим псевдографики	14
4.11	Список брейкпоинтов	15
4.12	Просмотр содержимого регистров	16
4.13	Просмотр содержимого переменных двумя способами	17
4.14	Изменение содержимого переменных двумя способами	18
4.15	Примеры использования команды set	19
4.16	Подготовка новой программы	20
4.17	Проверка работы стека	21
4.18	Текст программы	22
4.19	Поиск ошибки в программе через пошаговую отладку	24
4.20	Проверка корректировок в программе	24

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. Самостоятельное выполнение заданий по материалам лабораторной работы

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки.

Можно выделить следующие типы ошибок:

- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

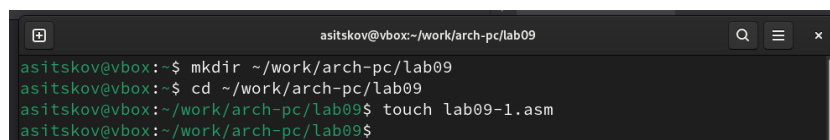
Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

4 Выполнение лабораторной работы

4.1 Релазиация подпрограмм в NASM

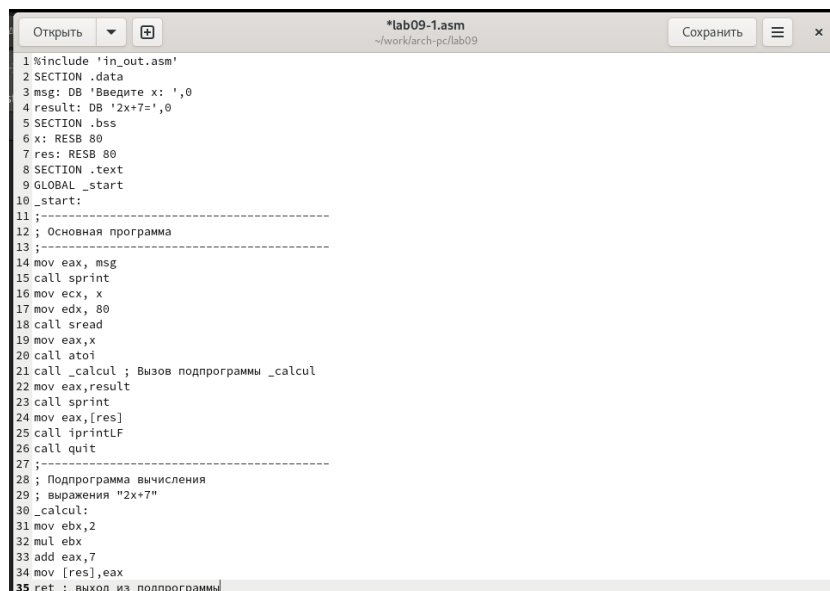
Создаю новый каталог и файл (рис. 4.1).



```
asitskov@vbox: ~/work/arch-pc/lab09
asitskov@vbox:~$ mkdir ~/work/arch-pc/lab09
asitskov@vbox:~$ cd ~/work/arch-pc/lab09
asitskov@vbox:~/work/arch-pc/lab09$ touch lab09-1.asm
asitskov@vbox:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание рабочего каталога

Копирую в файл код из листинга (рис. 4.2).



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

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 sprintf
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 sprintf
24 mov eax, [res]
25 call iprintf
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: Текст программы

Компилирую и запускаю программу (рис. 4.3).


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

Рис. 4.3: Запуск программы

Добавляю в текст программы подпрограмму чтобы она вычисляла значение функции для выражения $f(g(x))$ (рис. 4.4).

```
*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 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _calcul
19 mov eax, result
20 call sprint
21 mov eax, [res]
22 call iprintLF
23 call quit
24 _calcul:
25 push eax
26 call _subcalcul
27 mov ebx, 2
28 mul ebx
29 add eax, 7
30 mov [res], eax
31 pop eax
32 ret
33 _subcalcul:
34 mov ebx, 3
35 mul ebx
36 sub eax, 1
37 ret
```

Рис. 4.4: Текст программы

Код программы:

```
%include 'in_out.asm'

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

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit

_calcul:
push eax
call _subcalcul

```

```


mov ebx, 2
mul ebx
add eax, 7

mov [res], eax
pop eax
ret

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret

```

Компилирую и запускаю программу (рис. 4.5).



```

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

```

Рис. 4.5: Запуск программы

4.1.1 Отладка программ с помощью GDB

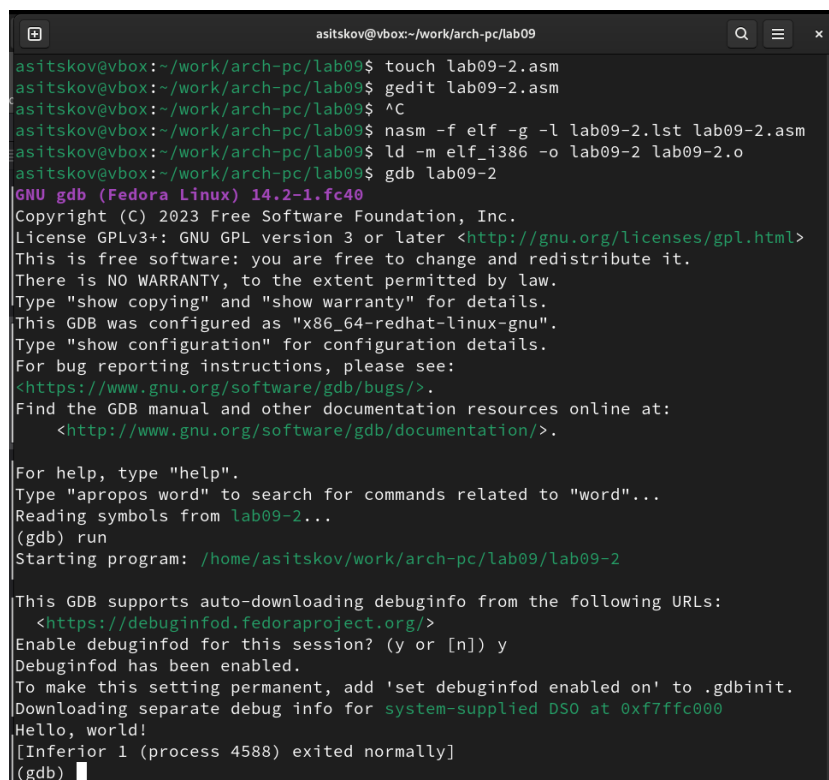
Создаю новый пустой файл и ввожу туда текст программы из листинга (рис. 4.6).



```
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
```

Рис. 4.6: Текст программы

Компилирую и запускаю программу с помощью gdb (рис. 4.7).



```
asitskov@vbox:~/work/arch-pc/lab09$ touch lab09-2.asm
asitskov@vbox:~/work/arch-pc/lab09$ gedit lab09-2.asm
asitskov@vbox:~/work/arch-pc/lab09$ ^C
asitskov@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
asitskov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
asitskov@vbox:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/asitskov/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 4588) exited normally]
(gdb)
```

Рис. 4.7: Запуск программы

Добавляю брейкпоинт на метку _start и снова запускаю отладку (рис. 4.8).

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

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

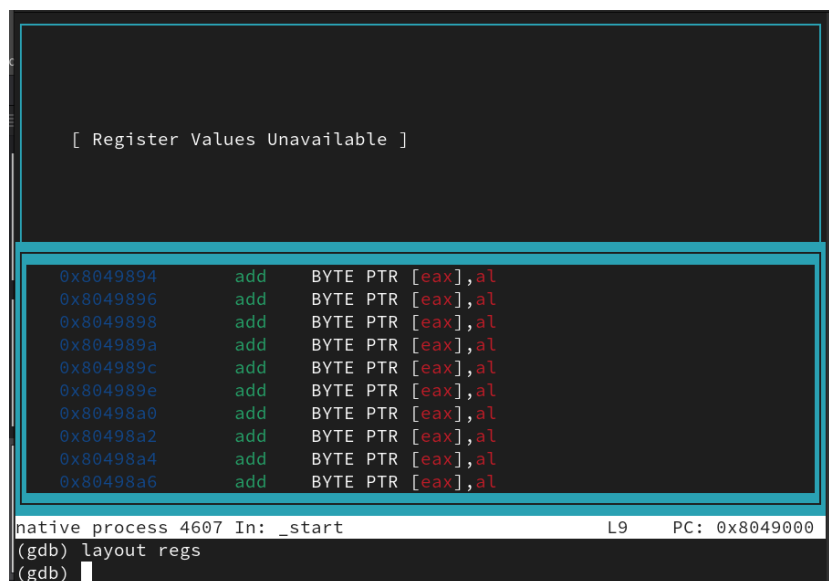
Рис. 4.8: Добавление брейкпоинта

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel (рис. 4.9). 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 `eax`, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

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

Рис. 4.9: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. 4.10).



```
[ Register Values Unavailable ]

0x8049894      add     BYTE PTR [eax], al
0x8049896      add     BYTE PTR [eax], al
0x8049898      add     BYTE PTR [eax], al
0x804989a      add     BYTE PTR [eax], al
0x804989c      add     BYTE PTR [eax], al
0x804989e      add     BYTE PTR [eax], al
0x80498a0      add     BYTE PTR [eax], al
0x80498a2      add     BYTE PTR [eax], al
0x80498a4      add     BYTE PTR [eax], al
0x80498a6      add     BYTE PTR [eax], al

native process 4607 In: _start          L9      PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 4.10: Режим псевдографики

4.1.2 Добавление точек останова

Устанавливаю еще одну точку останова и проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. 4.11).

```
asitskov@vbox:~/work/arch-pc/lab09

[ Register Values Unavailable ]

0x8049900  add  BYTE PTR [eax],al
0x8049902  add  BYTE PTR [eax],al
0x8049904  add  BYTE PTR [eax],al
0x8049906  add  BYTE PTR [eax],al
0x8049908  add  BYTE PTR [eax],al
0x804990a  add  BYTE PTR [eax],al
0x804990c  add  BYTE PTR [eax],al
0x804990e  add  BYTE PTR [eax],al
0x8049910  add  BYTE PTR [eax],al
0x8049912  add  BYTE PTR [eax],al

native process 4607 In: _start L9 PC: 0x8049000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint      keep y   0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 4.11: Список брейкпоинтов

4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers` (рис. 4.12).

```
asitskov@vbox:~/work/arch-pc/lab09

[ Register Values Unavailable ]

0x8049900 add BYTE PTR [eax],al
0x8049902 add BYTE PTR [eax],al
0x8049904 add BYTE PTR [eax],al
0x8049906 add BYTE PTR [eax],al
0x8049908 add BYTE PTR [eax],al
0x804990a add BYTE PTR [eax],al
0x804990c add BYTE PTR [eax],al
0x804990e add BYTE PTR [eax],al
0x8049910 add BYTE PTR [eax],al
0x8049912 add BYTE PTR [eax],al

native process 4607 In: _start L9 PC: 0x8049000
eax 0x0 0
ecx 0x0 0
edx 0x0 0
ebx 0x0 0
esp 0xffffd0c0 0xffffd0c0
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x8049000 0x8049000 <_start>
eflags 0x202 [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.12: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. 4.13).


```
asitskov@vbox:~/work/arch-pc/lab09

[ Register Values Unavailable ]

0x8049900 add BYTE PTR [eax],al
0x8049902 add BYTE PTR [eax],al
0x8049904 add BYTE PTR [eax],al
0x8049906 add BYTE PTR [eax],al
0x8049908 add BYTE PTR [eax],al
0x804990a add BYTE PTR [eax],al
0x804990c add BYTE PTR [eax],al
0x804990e add BYTE PTR [eax],al
0x8049910 add BYTE PTR [eax],al

native process 4607 In: _start L9 PC: 0x8049000
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) 
```

Рис. 4.13: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. 4.14).

The screenshot shows a GDB terminal window with the title bar 'asitskov@vbox: ~/work/arch-pc/lab09'. The main window is divided into two panes. The top pane, titled '[Register Values Unavailable]', is empty. The bottom pane displays assembly instructions for a native process 4607. The instructions are as follows:

Address	Instruction
0x8049900	add BYTE PTR [eax],al
0x8049902	add BYTE PTR [eax],al
0x8049904	add BYTE PTR [eax],al
0x8049906	add BYTE PTR [eax],al
0x8049908	add BYTE PTR [eax],al
0x804990a	add BYTE PTR [eax],al
0x804990c	add BYTE PTR [eax],al
0x804990e	add BYTE PTR [eax],al
0x8049910	add BYTE PTR [eax],al

Below the assembly instructions, the GDB prompt shows the following commands and their outputs:

```
native process 4607 In: _start L9 PC: 0x8049900
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='v'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "vorld!\n\034"
(gdb)
```

Рис. 4.14: Изменение содержимого переменных двумя способами

С помощью команды set меняю содержимое регистра ebx (рис. 4.15).

The screenshot shows the GDB interface with the following content:

Register group: general

Register	Value	Comment
eax	0x0	0
ecx	0x0	0
edx	0x0	0
ebx	0x2	2
esp	0xffffd0c0	0xffffd0c0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049000	0x8049000 <_start>

Assembly code:

```
0x8049900 add BYTE PTR [eax],al
0x8049902 add BYTE PTR [eax],al
0x8049904 add BYTE PTR [eax],al
0x8049906 add BYTE PTR [eax],al
0x8049908 add BYTE PTR [eax],al
0x804990a add BYTE PTR [eax],al
0x804990c add BYTE PTR [eax],al
0x804990e add BYTE PTR [eax],al
0x8049910 add BYTE PTR [eax],al
```

native process 4607 In: _start L9 PC: 0x8049000

(gdb) p/s \$eax
\$2 = 0
(gdb) p/t &eax
No symbol "eax" in current context.
(gdb) p/s \$ecx
\$3 = 0
(gdb) p/x \$ecx
\$4 = 0x0
(gdb) set \$ebx=2
(gdb) p/s \$ebx
\$5 = 2
(gdb)

Рис. 4.15: Примеры использования команды set

4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы и запускаю ее (рис. 4.16).

```

asitskov@vbox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/
arch-pc/lab09/lab09-3.asm
asitskov@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
asitskov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
asitskov@vbox:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'а
ргумент 3'
GNU gdb (Fedora Linux) 14.2-1.fc40
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-redhat-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. 4.16: Подготовка новой программы

Запускаю программу с режиме отладки с указанием аргументов, указываю брейкпоинт и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра esp на +4, число обусловлено разрядностью системы, а указатель void занимает как раз 4 байта, ошибка при аргументе +24 означает, что аргументы на вход программы закончились. (рис. 4.17).

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

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) x/x $esp
Please answer y or [n].
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xfffffd070: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffd23f: "/home/asitskov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffd269: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffd27b: "аргумент"
(gdb) x/s *(void**)(esp + 12
A syntax error in expression, near `'.
(gdb) x/s *(void**)(esp + 16)
0xfffffd28c: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffd28e: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.17: Проверка работы стека

4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 4.18).

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg_func db "Функция: f(x) = 10x - 4", 0
5 msg_result db "Результат: ", 0
6
7 SECTION .text
8 GLOBAL _start
9
10 _start:
11 mov eax, msg_func
12 call sprintf
13
14 pop ecx
15 pop edx
16 sub ecx, 1
17 mov esi, 0
18
19 next:
20 cmp ecx, 0h
21 jz _end
22 pop eax
23 call atoi
24
25 call _calculate_fx
26
27 add esi, eax
28 loop next
29
30 _end:
31 mov eax, msg_result
32 call sprintf
33 mov eax, esi
34 call iprintf
35 call quit
36
37 _calculate_fx:
38 mov ebx, 10
39 mul ebx
40 sub eax, 4

```

Рис. 4.18: Текст программы

Код программы:

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

```
SECTION .data
```

```
msg_func db "Функция: f(x) = 10x - 4", 0
```

```
msg_result db "Результат: ", 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msg_func
```

```
call sprintf
```

```
pop ecx
```

```
pop edx
```

```

sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

call _calculate_fx

add esi, eax
loop next

_end:
mov eax, msg_result
call sprint
mov eax, esi
call iprintLF
call quit

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4

```

2. Запускаю программу в режиме отладки и пошагово анализирую выполнение через команду si, отслеживая изменения значений регистров с помощью команды i r. При выполнении инструкции mul ecx видно, что результат умножения записывается в регистр eax, но при этом также изменяется зна-

чение регистра edx. Регистр ebx при этом остается неизменным, из-за чего программа неправильно вычисляет значение функции (рис. 4.19).

```

Register group: general
eax 0x2 2 ecx 0x4 4
edx 0x0 0 ebx 0x5 5
esp 0xffffcf10 0xffffcf10 ebp 0x0 0
esi 0x0 0 edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17> eflags 0x206 [ PF IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+ 0x80490e8 <_start> mov 30x3, %ebx
0x80490ed <_start+5> mov 30x2, %eax
0x80490f2 <_start+10> add %eax, %ebx
0x80490f4 <_start+12> mov 30x4, %ecx
> 0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19> add 30x5, %ebx
0x80490fa <_start+22> mov %ebx, %edi
0x8049100 <_start+24> mov 30x804a000, %eax
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov %edi, %eax

native process 8526 (asm) In: _start L14 PC: 0x80490f9
eax 0x2 2
ecx 0x4 4
edx 0x0 0
ebx 0x5 5
esp 0xffffcf10 0xffffcf10
ebp 0x0 0
esi 0x0 0
edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17>
eflags 0x206 [ PF IF ]
cs 0x23 35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.19: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. 4.20).

```

asitskov@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
asitskov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
asitskov@vbox:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
asitskov@vbox:~/work/arch-pc/lab09$

```

Рис. 4.20: Проверка корректировок в программе

Код измененной программы:

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

```
SECTION .data
```

```
div: DB 'Результат: ', 0
```

```
SECTION .text
```

```
GLOBAL _start
```



```
_start:

mov ebx, 3
mov eax, 2
add ebx, eax
mov eax, ebx
mov ecx, 4
mul ecx
add eax, 5
mov edi, eax

mov eax, div
call sprint
mov eax, edi
call iprintLF

call quit
```

5 Выводы

В результате выполнения данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм, а так же познакомился с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №9
3. Программирование на языке ассемблера NASM Столяров А. В.