

Отчет по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB

Тищенко Диана Борисовна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Самостоятельная работа	20
4	Вывод	26

Список иллюстраций

2.1	Создание каталога и файла	6
2.2	Текст программы	7
2.3	Работа программы	8
2.4	Измененный текст программы	9
2.5	Проверка работы программы	10
2.6	Текст второй программы	11
2.7	Отладка второго файла	12
2.8	Брежпоинт на метку _start	12
2.9	Дисассимплированный код	13
2.10	Intel’овское отображение	13
2.11	Псевдографика	14
2.12	Наличие меток	14
2.13	Просмотр регистров	15
2.14	Измененные регистры	15
2.15	Просмотр значения переменной	15
2.16	Значение переменной msg2	16
2.17	Изменение значения переменной	16
2.18	Изменение msg2	16
2.19	Значение регистров ехх и еах	16
2.20	Значение регистров ебх	17
2.21	Завершение работы с файлов	17
2.22	Запуск файла в отладчике	18
2.23	Запуск файла lab10-3 через метку	18
2.24	Адрес вершины стека	18
2.25	Все позиции стека	19
3.1	Текст программы	21
3.2	Запуск программы	22
3.3	Текст программы	23
3.4	Запуск программы	24
3.5	Запуск программы в отладчике	24
3.6	Анализ регистров	25
3.7	Повторный запуск программы	25

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1) Я создала каталог lab9 и создала файл lab9-1.asm

```
> mkdir ~/work/arch-pc/lab09
> cd ~/work/arch-pc/lab09
> touch lab9-1.asm
> ls
lab9-1.asm
~/work/arch-pc/lab09 master !227 ?2 > |
```

Рис. 2.1: Создание каталога и файла

2) Я ввела текст листинга в файл и запустила программу.

```
mov eax, x
call atoi

call _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
```

Рис. 2.2: Текст программы

```
> nasm -f elf lab9-1.asm
> ld -m elf_i386 -o lab9-1 lab9-1.o
> ./lab9-1
Введите x: 5
2x+7=17
~/work/arch-pc/lab09 master !227 ?2 > |
```

Рис. 2.3: Работа программы

3) Я изменила текст программы, чтобы она решала выражение $f(g(x))$.


```

%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

```

Рис. 2.4: Измененный текст программы

```
EX 9-1
> nasm -f elf lab9-1.asm
> ld -m elf_i386 -o lab9-1 lab9-1.o
> ./lab9-1
Введите x: 2
2(3x-1)+7=17
~/work/a/lab09 master !227 ?2 > |
```

Рис. 2.5: Проверка работы программы

4)Я создала файл lab9-2.asm и вписала туда программу.

```
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. 2.6: Текст второй программы

5) Я загрузила и запустила файл второй программы в отладчик gdb.

```
> nasm -f elf lab9-2.asm
> ld -m elf_i386 -o lab9-2 lab9-2.o
> gdb lab9-2
GNU gdb (GDB) 15.2
Copyright (C) 2024 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://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 lab9-2...
(no debugging symbols found in lab9-2)
(gdb) █
```

Рис. 2.7: Отладка второго файла

6) Я поставила брекпоинт на метку _start и запустила программу.

```
(gdb) break _start
Breakpoint 1 at 0x8049000
(gdb) run
Starting program: /home/dina/work/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.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.

Breakpoint 1, 0x08049000 in _start ()
(gdb) █
```

Рис. 2.8: Брекпоинт на метку _start

7) Я просмотрела дисассимплированный код программы начиная с метки.

```
(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. 2.9: Дисассимплированный код

- 8) С помощью команды я переключилась на intel'овское отображение синтаксиса. Отличие заключается в командах, в дисассимлированном отображении в командах используют % и \$, а в Intel отображение эти символы не используются. На такое отображение удобнее смотреть.

```
(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. 2.10: Intel'овское отображение

- 9) Для удобства я включила режим псевдографики.

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffced0 0xffffced0
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

[ No Source Available ]

native process 338877 (src) In: _start L?? PC: 0x8049000
(gdb) 
```

Рис. 2.11: Псевдографика

10) Я посмотрела наличие меток и добавила еще одну метку на предпоследнюю инструкцию.

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffced0 0xffffced0
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

[ No Source Available ]

native process 338877 (src) In: _start L?? PC: 0x8049000
(gdb) info breakpoints
Num   Type      Disp Enb Address  What
1     breakpoint keep y  0x08049000 <_start>
      breakpoint already hit 1 time
(gdb) i b
Num   Type      Disp Enb Address  What
1     breakpoint keep y  0x08049000 <_start>
      breakpoint already hit 1 time
(gdb) 
```

Рис. 2.12: Наличие меток

11) С помощью команды si я посмотрела регистры и изменил их.

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffced0 0xffffced0
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

[ No Source Available ]

native process 338877 (src) In: _start L?? PC: 0x8049005
(gdb) info breakpoints
Num      Type      Disp Enb Address  What
1        breakpoint keep y 0x08049000 <_start>
breakpoint already hit 1 time
(gdb) i b
Num      Type      Disp Enb Address  What
1        breakpoint keep y 0x08049000 <_start>
breakpoint already hit 1 time
(gdb) si
0x8049005 in _start ()

```

Рис. 2.13: Просмотр регистров

eax	0x4	4
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xffffced0	0xffffced0
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

Рис. 2.14: Измененные регистры

12) С помощью команды я посмотрела значение переменной msg1.

```

(gdb) x/lb &msg1
0x804a000: "Hello, "
(gdb)

```

Рис. 2.15: Просмотр значения переменной

13) Следом я посмотрела значение второй переменной msg2.

```
(gdb) x/lsb &msg2
0x804a008: "world!\n"
(gdb) █
```

Рис. 2.16: Значение переменной msg2

14) С помощью команды set я изменила значение переменной msg1.

```
(gdb) set {char}0x804a000='0'
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/lsb &msg1
0x804a000: "hhlllo, "
(gdb) █
```

Рис. 2.17: Изменение значения переменной

15) Я изменила переменную msg2.

```
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
warning: Expression is not an assignment (and might have no effect)
(gdb) set {char}0x804a00b=' '
(gdb) x/lsb &msg2
0x804a008: "Lor d!\n"
(gdb) █
```

Рис. 2.18: Изменение msg2

16) Я вывела значение регистров ecx и eax.

```
(gdb) p/t $eax
$1 = 100
(gdb) p/c $ecx
$2 = 0 '\000'
(gdb) p/x $ecx
$3 = 0x0
(gdb) █
```

Рис. 2.19: Значение регистров ecx и eax

- 17) Я изменила значение регистра ebx. Команда выводит два разных значения так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум, поэтому и значения разные.

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) [/s $ebx
Undefined command: "". Try "help".
(gdb) p/s $ebx
$5 = 2
(gdb) █
```

Рис. 2.20: Значение регистров ebx

- 18) Я завершила работу с файлов вышла.

```
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) layout regs
~/work/a/lab09 master !227 ?2 > █
```

Рис. 2.21: Завершение работы с файлов

- 19) Я скопировала файл lab8-2.asm и переименовала его. Запустила файл в отладчике и указала аргументы.

```

> gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) 15.2
Copyright (C) 2024 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://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 lab9-3...
(No debugging symbols found in lab9-3)
(gdb) █

```

Рис. 2.22: Запуск файла в отладчике

20) Поставила метку на `_start` и запустила файл.

```

(gdb) b _start
Breakpoint 1 at 0x080490e8
(gdb) r
Starting program: /home/dina/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 аргумент 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.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.

Breakpoint 1, 0x080490e8 in _start ()
(gdb) █

```

Рис. 2.23: Запуск файла lab10-3 через метку

21) Я проверила адрес вершины стека и убедилась что там хранится 5 элементов.

```

(gdb) x/x $esp
0xffffce80: 0x00000005
(gdb) █

```

Рис. 2.24: Адрес вершины стека

22) Я посмотрела все позиции стека. По первому адресу хранится адрес, в остальных адресах хранятся элементы. Элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации.

```
(gdb) x/x $esp
0xffffce80: 0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xffffd07a: "/home/dina/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd09f: "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd0b1: "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xffffd0c2: "2"
(gdb) x/s *(void**)( $esp + 20)
0xffffd0c4: "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.25: Все позиции стека

3 Самостоятельная работа

- 1) Я преобразовала программу из лабораторной работы №8 и реализовала вычисления как подпрограмму.

```

%include 'in_out.asm'
SECTION .data
msg_func db "Функция:  $f(x) = 8x - 3$ ", 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 sprintf
mov eax, esi
call iprintfLF
call quit
_calculate_fx:
mov ebx, 8
mul ebx
sub eax, 3
ret

```

Рис. 3.1: Текст программы

```
> touch lab9-4.asm  
> nasm -f elf lab9-4.asm  
> ld -m elf_i386 -o lab9-4 lab9-4.o  
> ./lab9-4
```

Рис. 3.2: Запуск программы

- 2) Я переписала программу и попробовала запустить ее чтобы увидеть ошибку. Ошибка была арифметическая, так как вместо 25, программа выводит 10.

```

%include 'in_out.asm'
SECTION .data
msg_func db "Функция: f(x) = 8x - 3", 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 sprintf
mov eax, esi
call iprintLF
call quit
_calculate_fx:
mov ebx, 8
mul ebx
sub eax, 3
ret

```

Рис. 3.3: Текст программы

```
> ./lab9-4 1
Функция:  $f(x) = 8x - 3$ 
Результат: 5
~/work/arch-pc/lab09 master !227 ?2 > █
```

Рис. 3.4: Запуск программы

После появления ошибки, я запустила программу в отладчике.

```
> gdb lab9-5
GNU gdb (GDB) 15.2
Copyright (C) 2024 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://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 lab9-5...
(No debugging symbols found in lab9-5)
(gdb) █
```

Рис. 3.5: Запуск программы в отладчике

Я открыла регистры и проанализировала их, поняла что некоторые регистры стоят не на своих местах и исправила это.


```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x8      8
esp      0xffffceb4 0xffffceb4
ebp      0x0      0x0
esi      0x0      0
edi      0xa      10
eip      0x8049003 0x8049003 <nextchar>
eflags   0x10206   [ PF IF RF ]
cs       0x23      35
ss       0x2b      43
ds       0x2b      43

>0x8049003 <nextchar> cmpb $0x0, (%eax)
0x8049006 <nextchar+3> je 0x804900b <finished>
0x8049008 <nextchar+5> inc %eax
0x8049009 <nextchar+6> jmp 0x8049003 <nextchar>
0x804900b <finished> sub %ebx, %eax
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 %eax, %edx
0x804901a <sprint+11> pop %eax

native process 340537 (asm) In: nextchar L?
(gdb) layout asm
(gdb) layout regs
(gdb) r
Starting program: /home/dina/work/arch-pc/lab09/lab9-5

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.archlinux.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.

Program received signal SIGSEGV, Segmentation fault.
0x08049003 in nextchar ()
(gdb) █

```

Рис. 3.6: Анализ регистров

Я изменила регистры и запустила программу, программа вывела ответ 25, то есть все работает правильно.

```

To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Program received signal SIGSEGV, Segmentation fault.
0x08049003 in nextchar ()
(gdb) █

```

Рис. 3.7: Повторный запуск программы

4 Вывод

Я приобрела навыки написания программ использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.