

Отчёт по лабораторной работе

Лабораторная №10

Дикач Анна Олеговна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация подпрограмм в NASM	6
2.2	Отладка программ с помощью GDB	7
2.3	Добавление точек останова	8
2.4	Работа с данными программы в GDB	9
2.5	Обработка аргументов командной строки в GDB	11
2.6	Задание для самостоятельной работы	13
3	Вывод	16

Список иллюстраций

2.1	пример работы исправленной программы	6
2.2	работа программы для вычисления выражения	6
2.3	добавление отладочной информации, загрузка файла в отладчик .	7
2.4	проверка работы программы в оболочке GDB с помощью команды run	7
2.5	пример работы программы	7
2.6	просмотр дисассимилированного кода	8
2.7	проверка точки останова	8
2.8	информация о всех установленных точках	9
2.9	значение переменной msg1	9
2.10	значение переменной msg2	9
2.11	изменение первого символа переменной msg1	9
2.12	изменение второго символа переменной msg2	10
2.13	вывод при разных значениях регистра	10
2.14	вид после завершения	11
2.15	копирование, создание исполняемого файла. загрузка файла в от- ладчик с указанием аргументов	12
2.16	пример работы	12
2.17	позиции стека	13

Список таблиц

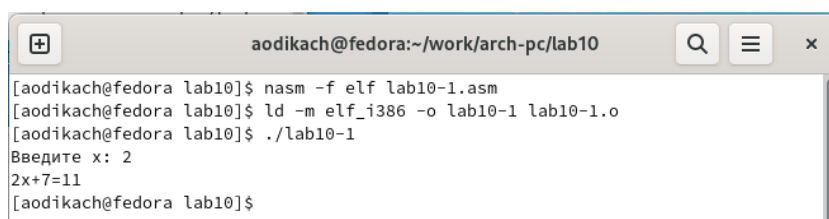
1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

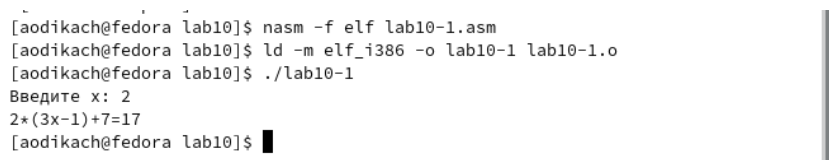
1. создаю каталог для выполнения лабораторной работы №10, перехожу в него и создаю файл lab10-1.asm
2. 1) внимательно изучаю текст программы, создаю исполняемый файл и проверяю его работу. так как файл не работал, вношу некоторые правки. (рис. 2.1)



```
aodikach@fedora:~/work/arch-pc/lab10
[aodikach@fedora lab10]$ nasm -f elf lab10-1.asm
[aodikach@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[aodikach@fedora lab10]$ ./lab10-1
Введите x: 2
2x+7=11
[aodikach@fedora lab10]$
```

Рис. 2.1: пример работы исправленной программы

- 2) меняю текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения $\text{X}(\text{X}(\text{X}))$, где X вводится с клавиатуры, $\text{X}(\text{X}) = 2\text{X} + 7$, $\text{X}(\text{X}) = 3\text{X} - 1$ (рис. 2.2)



```
aodikach@fedora lab10]$ nasm -f elf lab10-1.asm
[aodikach@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[aodikach@fedora lab10]$ ./lab10-1
Введите x: 2
2*(3x-1)+7=17
[aodikach@fedora lab10]$
```

Рис. 2.2: работа программы для вычисления выражения

2.2 Отладка программ с помощью GDB

3. создаю файл lab10-2.asm с текстом программы из листинга 10.2, получаю исполняемый файл, добавляю отладочную информацию (рис. 2.3)

```
[aodikach@fedora lab10]$ touch lab10-2.asm
[aodikach@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[aodikach@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[aodikach@fedora lab10]$ gdb lab10-2
```

Рис. 2.3: добавление отладочной информации, загрузка файла в отладчик

4. проверяю работу программы запустив её в оболочке GDB с помощью команды run (рис. 2.4)

```
.....
Hello, world!
[Inferior 1 (process 8073) exited normally]
(gdb) █
```

Рис. 2.4: проверка работы программы в оболочке GDB с помощью команды run

5. для более подробного анализа программы устанавливаю брейкпоинт на метку _start, запускаю её (рис. 2.5)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/aodikach/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
```

Рис. 2.5: пример работы программы

6. просматриваю дисассимилированный код программы с помощью команды disassemble (рис. 2.6)

```
(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.
```

Рис. 2.6: просмотр дисассимилированного кода

7. переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. ??). после смены интерфейса команды отображаются с привычным intel'ловским синтаксисом

```
(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.
```

2.3 Добавление точек останова

8. проверяю была ли установлена точка останова с помощью команды `info breakpoints` (рис. 2.7)

```
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y   0x08049000 lab10-2.asm:9
breakpoint already hit 1 time
```

Рис. 2.7: проверка точки останова

9. устанавливаю ещё одну точку останова по адресу инструкции с помощью команды `break *`
10. просматриваю информацию о всех установленных точках останова (рис. 2.8)

```
(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint       keep y  0x08049000  lab10-2.asm:9
          breakpoint       already hit 1 time
2        breakpoint       keep y  <PENDING>  0x08049031
(gdb)
```

Рис. 2.8: информация о всех установленных точках

2.4 Работа с данными программы в GDB

11. выполняю 5 инструкций с помощью команды `stepi` „,
12. просматриваю значение переменной `msg1` по имени (рис. 2.9)

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

Рис. 2.9: значение переменной `msg1`

13. просматриваю значение переменной `msg2` по адресу (рис. 2.10)

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

Рис. 2.10: значение переменной `msg2`

14. просматриваю инструкцию `mov ecx,msg2` которая записывает в регистр `ecx` адрес переменной `msg2`
15. изменяю первый символ переменной `msg1` (рис. 2.11)

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

Рис. 2.11: изменение первого символа переменной `msg1`

16. заменяю символ во второй переменной `msg2` с помощью команды `set {char}0x804a009='0'` (рис. 2.12)

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

Рис. 2.12: изменение второго символа переменной `msg2`

17. вывожу в различных форматах значение регистра `edx` (рис. 2.13).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
```

Рис. 2.13: вывод при разных значениях регистра

18. завершаю выполнение программы с помощью команды `continue` и выхожу из GDB с помощью команды `quit` (рис. 2.14).

```

0x0804902a <+42>:    int     $0x80
0x0804902c <+44>:    mov     $0x1,%eax
0x08049031 <+49>:    mov     $0x0,%ebx
0x08049036 <+54>:    int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     eax,0x4
0x08049005 <+5>:    mov     ebx,0x1
0x0804900a <+10>:   mov     ecx,0x804a000
0x0804900f <+15>:   mov     edx,0x8
0x08049014 <+20>:   int     0x80
0x08049016 <+22>:   mov     eax,0x4
0x0804901b <+27>:   mov     ebx,0x1
0x08049020 <+32>:   mov     ecx,0x804a008
0x08049025 <+37>:   mov     edx,0x7
0x0804902a <+42>:   int     0x80
0x0804902c <+44>:   mov     eax,0x1
0x08049031 <+49>:   mov     ebx,0x0
0x08049036 <+54>:   int     0x80
End of assembler dump.
(gdb) layout asm
[aodikach@fedora lab10]$

```

Рис. 2.14: вид после завершения

2.5 Обработка аргументов командной строки в GDB

19. копирую файл lab9-2.asm, создаю исполняемый файл, загружаю файл в отладчик, указав аргументы (рис. 2.15).

```
[aodikach@fedora lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
[aodikach@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[aodikach@fedora lab10]$ ld -m elf_i386 -o lab10-3.o
ld: отсутствуют входные файлы
[aodikach@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[aodikach@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 12.1-2.fc36
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-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 lab10-3...
(gdb)
```

Рис. 2.15: копирование, создание исполняемого файла. загрузка файла в отладчик с указанием аргументов

20. устанавливаю точку останова перед первой инструкцией и запуска её (рис. 2.16).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 8.
(gdb) run
Starting program: /home/aodikach/work/arch-pc/lab10/lab10-3 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]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

x/x $esp
Downloading 0.02 MB separate debug info for system-supplied DSO at 0xf7ffc000

Breakpoint 1, _start () at lab10-3.asm:8
8      pop ecx ; Извлекаем из стека в `ecx` количество
```

Рис. 2.16: пример работы

21. смотрю позиции стека по их адресам. шаг изменения равен 4 т.к размер переменной 4 байта (рис. 2.17).

```

(gdb) x/x $esp
0xffffd070:    0x00000005
(gdb) x/x $esp
0xffffd070:    0x00000005
(gdb) x/s *(void**)(esp +4)
0xffffd220:    "/home/aodikach/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp +8)
0xffffd24a:    "аргумент1"
(gdb) x/s *(void**)(esp +12)
0xffffd25c:    "аргумент"
(gdb) x/s *(void**)(esp +16)
0xffffd26d:    "2"
(gdb) x/s *(void**)(esp +20)
0xffffd26f:    "аргумент 3"
(gdb)

```

Рис. 2.17: позиции стека

2.6 Задание для самостоятельной работы

1. преобразование программы из лабораторной №9 с реализацией вычисления функции как подпрограммы (рис. ??)(рис. ??) (работая в файле samr-1)

```

mc [aodikach@fedora]:
GNU nano 6.0 /home/aodikach/work/a
%include 'in_out.asm'

SECTION .data
f db "Функция: f(x) = 5 * (2+x)", 0
msg db "Результат: ", 0

SECTION .text
global _start

_start:
mov eax, f
call sprintLF
pop ecx
pop edx
sub ecx,1
mov esi, 0 ; храним здесь сумму

next:
cmp ecx,0h
jz _end
pop eax
call atoi
call _calcul
add esi, eax
loop next

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

_calcul:
add eax, 2
mov edx, 5
mul edx
ret

[aodikach@fedora lab10]$ ./samr-1 5
Функция: f(x) = 5 * (2+x)
Результат: 35
[aodikach@fedora lab10]$

```

[^]G Справка [^]O Записать [^]W Поиск [^]K Вырезать
[^]X Выход [^]R ЧитФайл [^]\ Замена [^]U Вставить

- создаю файл samr-2, ввожу текст программы листинга, открываю отладчик, отлаживаю код (рис. ??). в коде регистр edi копирует данные из ebx, а то время как копировать должен из eax, 5 должно прибавляться к eax, а также

регистры ebx и eax перепутаны местами. (рис. ??)(рис. ??)

The image shows two windows from a Linux terminal. The left window is a GDB debugger session for a program named 'samr-2'. It displays the 'Register group: general' with values for eax, ebx, ecx, etc. Below this, it shows the assembly code for the '_start' function, with instructions like 'mov ebx, 0x3', 'mov eax, 0x2', 'add eax, ebx', etc. The right window is a nano editor showing the assembly source file 'in_out.asm'. It contains comments in Russian and assembly instructions like 'mov ebx, 3', 'mov eax, 2', 'add eax, ebx', 'mul ecx', 'add eax, 5', 'mov edi, eax', 'mov eax, div', 'call sprint', 'mov eax, edi', 'call iprintLF', and 'call quit'. At the bottom of the left window, there is a terminal prompt showing the execution of 'nasm -f elf samr-2.asm', 'ld -m elf_i386 -o samr-2 samr-2.o', and './samr-2', resulting in 'Результат: 25'.

```
[aodikach@fedora lab10]$ nasm -f elf samr-2.asm
[aodikach@fedora lab10]$ ld -m elf_i386 -o samr-2 samr-2.o
[aodikach@fedora lab10]$ ./samr-2
Результат: 25
[aodikach@fedora lab10]$
```

3 Вывод

научилась писать программы с использованием подпрограмм, а также попробовала переписать старую программу с использованием новой подпрограммы. узнала как пользоваться отладчиком и что такое отладка