

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Сокирка Анна Константиновна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	4.1 Релаксация подпрограмм в NASM	8
4.2	4.1.1 Отладка программ с помощью GDB	9
4.3	4.1.3 Работа с данными программы в GDB	12
4.4	4.1.4 Обработка аргументов командной строки в GDB	16
4.5	4.2 Задание для самостоятельной работы	16
5	Выводы	18
6	Список литературы	19

Список иллюстраций

4.1	Создание каталога	8
4.2	Запуск файла	8
4.3	Изменение текста программы	8
4.4	Запуск файла	9
4.5	Запуск программы	9
4.6	Запуск программы	10
4.7	Просмотр кода программы с помощью метки	10
4.8	Переключение	11
4.9	Режим псевдографики	12
4.10	Используем команду info breakpoints и создаем новую точку останова	12
4.11	Смотрим информацию	13
4.12	Отслеживаем регистры	13
4.13	Смотрим значение переменной	13
4.14	Смотрим значение переменной	14
4.15	Меняем символ	14
4.16	Меняем символ	14
4.17	Смотрим значение регистра	15
4.18	Изменяем регистр командой set	15
4.19	Прописываем команды с и quit	15
4.20	Подготовка новой программы	16
4.21	Устанавливаем точку останова	16
4.22	Изучаем полученные данные	16
4.23	Создаем файл	17

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 Релаксация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9. Перейду в него и создам файл lab09-1.asm (рис. 4.1).

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

Рис. 4.1: Создание каталога

Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. 4.2).

```
aksokirka@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
aksokirka@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
aksokirka@fedora:~/work/arch-pc/lab09$ ./lab9-1.asm
bash: ./lab9-1.asm: Отказано в доступе
aksokirka@fedora:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 21
2x+7=49
aksokirka@fedora:~/work/arch-pc/lab09$
```

Рис. 4.2: Запуск файла

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения $f(g(x))$ (рис. 4.3).

```
aksokirka@fedora:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
aksokirka@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
aksokirka@fedora:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 21
2(3x-1)+7=131
aksokirka@fedora:~/work/arch-pc/lab09$
```

Рис. 4.3: Изменение текста программы

4.2 4.1.1 Отладка программ с помощью GDB

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике (рис. 4.4).

```
aksokirka@fedora:~/work/arch-pc/lab09$ touch lab9-2.asm
aksokirka@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
aksokirka@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
aksokirka@fedora:~/work/arch-pc/lab09$ gdb lab9-2
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 lab9-2...
(gdb)
```

Рис. 4.4: Запуск файла

Запустив программу командой `run`, я убедилась в том, что она работает исправно (рис. 4.5).

```
Reading symbols from lab9-2...
(gdb) run
Starting program: /home/aksokirka/work/arch-pc/lab09/lab9-2

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 8254) exited normally]
(gdb)
```

Рис. 4.5: Запуск программы

Для более подробного анализа программы добавляю брейкпоинт на метку `_start` и снова запускаю отладку (рис. 4.6).

```
[Inferior 1 (process 8254) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/aksokirka/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb) █
```

Рис. 4.6: Запуск программы

Смотрю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 4.7).

```
(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. 4.7: Просмотр кода программы с помощью метки

Переключаюсь на отображение команд с Intel'овским (рис. 4.8).

```

(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. 4.8: Переключение

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 ах, еах, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

Включаю режим псевдографики для более удобного анализа программы (рис. 4.9).

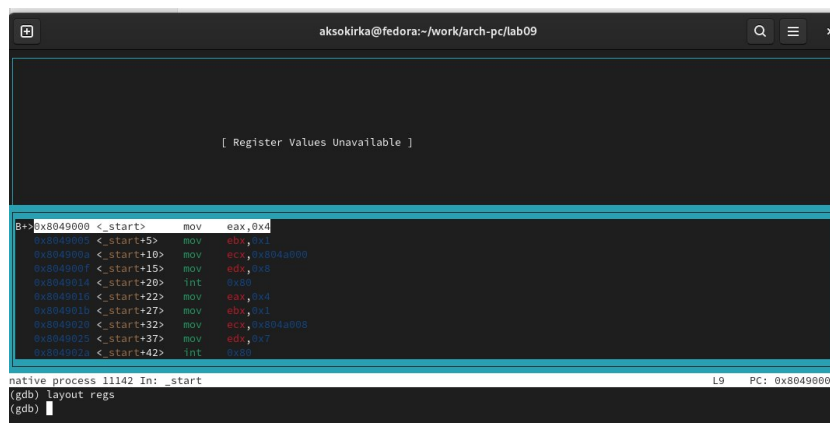


Рис. 4.9: Режим псевдографики

4.3 4.1.3 Работа с данными программы в GDB

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. 4.10).

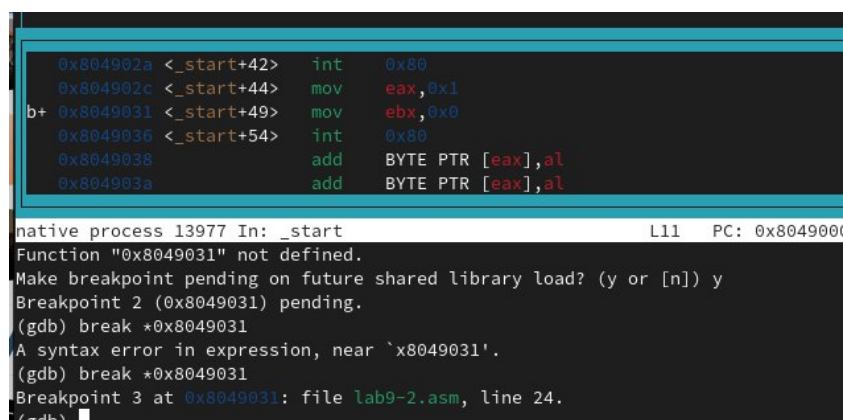


Рис. 4.10: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова(рис. 4.11).

```

native process 13977 In: _start L11 PC: 0x8049000
Breakpoint 3 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint       keep y 0x08049000 lab9-2.asm:11
       breakpoint already hit 1 time
2      breakpoint       keep y <PENDING> 0x8049031
3      breakpoint       keep y 0x08049031 lab9-2.asm:24
(gdb)

```

Рис. 4.11: Смотрим информацию

Выполняем 5 инструкций командой si (рис. 4.12).

```

aksokirka@fedora:~/work/arch-pc/lab09
--Register group: general--
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd080 0xffffd080
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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

native process 13977 In: _start L17
       breakpoint keep y <PENDING> 0x8049031
       breakpoint keep y 0x08049031 lab9-2.asm:24
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.12: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip. Смотрим значение переменной msg1 по имени (рис. 4.13).

```

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

```

Рис. 4.13: Смотрим значение переменной

Смотрим значение переменной msg2 по адресу (рис. 4.14).

```
(gdb) si
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "World!\n\034"
(gdb) █
```

Рис. 4.14: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. 4.15).

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

Рис. 4.15: Меняем символ

Изменим первый символ переменной msg2 (рис. 4.16).

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

Рис. 4.16: Меняем символ

Смотрим значение регистра edx в разных форматах (рис. 4.17).

```

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

```

Рис. 4.17: Смотрим значение регистра

Изменяем регистр ebx (рис. 4.18).

```

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

```

Рис. 4.18: Изменяем регистр командой set

Выводятся разные значения, так как команда без кавычек присваивает регистру вводимое значение. Прописываем команды для завершения программы и выхода из GDB (рис. 4.19).

```

(gdb) c
Continuing.
World!

Breakpoint 3, _start () at lab9-2.asm:24
(gdb)

```

Рис. 4.19: Прописываем команды c и quit

4.4 4.1.4 Обработка аргументов командной строки в GDB

Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. 4.20).

```
(gdb) layout asm
aksokirka@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
aksokirka@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
aksokirka@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
aksokirka@fedora:~/work/arch-pc/lab09$
```

Рис. 4.20: Подготовка новой программы

Запускаем его в отладчике GDB. Установим точку останова перед первой инструкцией в программе и запустим ее (рис. 4.21).

Устанавливаем точку останова

Рис. 4.21: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. 4.22).

```
(gdb) x/x $esp
0xffffd070: 0x00000004
(gdb) x/s *(void**)($esp + 4)
0xffffd233: "/home/aksokirka/work/arch-pc/lab09/lab9-3"
(gdb) x/s*(void**)($esp + 8)
0xffffd25d: "2"
(gdb) x/s *(void**)($esp + 12)
0xffffd26f: "3"
(gdb) x/s *(void**)($esp + 16)
0xffffd261: "5"
(gdb) x/s *(void**)($esp + 20)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.22: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

4.5 4.2 Задание для самостоятельной работы

Создаем новый файл в директории.Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3. Создаем исполняемый файл и

запускаем его (рис. 4.23).

```
GNU nano 7.2 /home/aksokirka/work/arch-pc/lab09/la
%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. 4.23: Создаем файл

5 Выводы

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

6 Список литературы

1 {#https://esystem.rudn.ru/pluginfile.php/2089096/mod_resource/content/0/Лабораторная%20работа%20№9.%20Понятие%20подпрограммы.%20Отладчик%20..pdf}