

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Никуленков Степан Сергеевич

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	10
4.1.2	Добавление точек останова	13
4.1.3	Работа с данными программы в GDB	14
4.1.4	Обработка аргументов командной строки в GDB	17
4.2	Задание для самостоятельной работы	17
5	Выводы	22
6	Список литературы	23

Список иллюстраций

4.1	Создание рабочего каталога	8
4.2	Запуск программы из листинга	8
4.3	Изменение программы первого листинга	8
4.4	Запуск программы в отладчике	11
4.5	Проверка программы отладчиком	11
4.6	Запуск отладчика с брейкпойнтом	12
4.7	Дисассимилирование программы	12
4.8	Режим псевдографики	13
4.9	Список брейкпойнтов	13
4.10	Добавление второй точки останова	14
4.11	Просмотр содержимого регистров	14
4.12	Просмотр содержимого переменных двумя способами	15
4.13	Изменение содержимого переменных двумя способами	15
4.14	Просмотр значения регистра разными представлениями	16
4.15	Примеры использования команды set	16
4.16	Подготовка новой программы	17
4.17	Измененная программа предыдущей лабораторной работы	17
4.18	Поиск ошибки в программе через пошаговую отладку	19
4.19	Проверка корректировок в программме	20

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, отработывает, 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 каталог для выполнения лабораторной работы №9 (рис. -fig. 4.1).

```
root@vbox:/home/ssnikulenkov/work/arch-pc# mkdir lab09
root@vbox:/home/ssnikulenkov/work/arch-pc# ls
lab04 lab05 lab06 lab07 lab08 lab09
root@vbox:/home/ssnikulenkov/work/arch-pc# cd lab09
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09# touch lab9-1.asm
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09#
```

Рис. 4.1: Создание рабочего каталога

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

```
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09# nasm -f elf lab9-1.asm
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09# ld -m elf_i386 -o lab9-1 lab9-1.o
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09# ./lab9-1
Введите x: 10
2x+7=27
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09#
```

Рис. 4.2: Запуск программы из листинга

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

```
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09# nasm -f elf lab9-1.asm
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09# ld -m elf_i386 -o lab9-1 lab9-1.o
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09# ./lab9-1
Введите x: 10
2(3x-1)+7=65
root@vbox:/home/ssnikulenkov/work/arch-pc/lab09#
```

Рис. 4.3: Изменение программы первого листинга

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

```

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. -fig. 4.4).

```

oot@vbox: /home/ssnikulenkoy/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 программы в отладчике

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

```

for help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(gdb) run
Starting program: /home/ssnikulenkoy/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 4866) exited normally]
(gdb) run
Starting program: /home/ssnikulenkoy/work/arch-pc/lab09/lab9-2
hello, world!
Inferior 1 (process 4872) exited normally]
(gdb)

```

Рис. 4.5: Проверка программы отладчиком

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

```
<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 4866) exited normally]
(gdb) run
Starting program: /home/ssnikulenkov/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 4872) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/ssnikulenkov/work/arch-pc/lab09/lab9-2

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

Рис. 4.6: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel(рис. -fig. 4.7).

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).

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

Включаю режим псевдографики для более удобного анализа программы (рис. -fig. 4.8).

```
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. 4.8: Режим псевдографики

4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. -fig. 4.9).

```
[ Register Values Unavailable ]

0x8049210    add     BYTE PTR [eax],al
0x8049212    add     BYTE PTR [eax],al
0x8049214    add     BYTE PTR [eax],al
0x8049216    add     BYTE PTR [eax],al
0x8049218    add     BYTE PTR [eax],al

native process 4874 In: _start          L9    PC: 0x8049000
(gdb) layout regs
(gdb) 
```

Рис. 4.9: Список брейкпоинтов

Устанавливаю еще одну точку останова по адресу инструкции (рис. -fig. 4.10).

```

root@vbox:~ x root@vbox:/home/ssnikulenkov/... x
[ Register Values Unavailable ]

0x804938a add BYTE PTR [eax],al
0x804938c add BYTE PTR [eax],al
0x804938e add BYTE PTR [eax],al
0x8049390 add BYTE PTR [eax],al
0x8049392 add BYTE PTR [eax],al

native process 4874 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab9-2.asm:9
breakpoint already hit 1 time
(gdb)

```

Рис. 4.10: Добавление второй точки останова

4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers` (рис. -fig. 4.11).

```

Register group: general
ds      0x2b      43
es      0x2b      43
fs      0x0       0
gs      0x0       0

0x804a6c9 add BYTE PTR [eax],al
0x804a6cb add BYTE PTR [eax],al
0x804a6cd add BYTE PTR [eax],al
0x804a6cf add BYTE PTR [eax],al
0x804a6d1 add BYTE PTR [eax],al

native process 4874 In: _start L9 PC: 0x8049000
gs      0x0       0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -fig. 4.12).

```

Register group: general
ds      0x2b      43
es      0x2b      43
fs      0x0       0
gs      0x0       0

0x804a6c9  add  BYTE PTR [eax],al
0x804a6cb  add  BYTE PTR [eax],al
0x804a6cd  add  BYTE PTR [eax],al
0x804a6cf  add  BYTE PTR [eax],al
0x804a6d1  add  BYTE PTR [eax],al

native process 4874 In: _start L9 PC: 0x8049000
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. -fig. 4.13).

```

Register group: general
eax      0x0       0
ecx      0x0       0
edx      0x0       0
ebx      0x32      50
esp      0xffffd430 0xffffd430
ebp      0x0       0x0
esi      0x0       0

0x804a6c9  add  BYTE PTR [eax],al
0x804a6cb  add  BYTE PTR [eax],al
0x804a6cd  add  BYTE PTR [eax],al
0x804a6cf  add  BYTE PTR [eax],al
0x804a6d1  add  BYTE PTR [eax],al
0x804a6d3  add  BYTE PTR [eax],al

native process 4874 In: _start L9 PC: 0x8049000
2 = 0
(gdb) p/t $edx
3 = 0
(gdb) p/x $edx
4 = 0x0
(gdb)

```

Рис. 4.13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. -fig. 4.14).

```
ssnikulenkov@vbox:~$ cd
ssnikulenkov@vbox:~$ cd work/arch-pc/lab09/
ssnikulenkov@vbox:~/work/arch-pc/lab09$ ls
in_out.asm lab9-1.asm lab9-2 lab9-2.lst
lab9-1 lab9-1.o lab9-2.asm lab9-2.o
ssnikulenkov@vbox:~/work/arch-pc/lab09$ touch lab9-3.asm
ssnikulenkov@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-
m
ssnikulenkov@vbox:~/work/arch-pc/lab09$ ^C
ssnikulenkov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-
ssnikulenkov@vbox:~/work/arch-pc/lab09$ gdb --args lab09-3 apry
2 'аргумент 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.
For bug reporting instructions, please see
<https://www.gnu.org/software/gdb/bugs/>.
```

Рис. 4.14: Просмотр значения регистра разными представлениями

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

```
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ssnikulenkov/work/arch-pc/lab09/lab9-3 аргумент1 аргум
ент 2 аргумент\ 3

Breakpoint 1, _start () at lab9-3.asm:7
7      pop ecx
(gdb) x/x $esp
0xffffd060: 0x00000005
(gdb) (gdb) x/s *(void**)(esp + 4)
Undefined command: "". Try "help".
(gdb) x/s *(void**)(esp + 4)
0xffffd22b: "/home/ssnikulenkov/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd258: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd26a: "аргумент"
(gdb)
0xffffd27b: "2"
(gdb) x/s *(void**)(esp + 16)
0xffffd27b: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd27d: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.15: Примеры использования команды set

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. -fig. 4.16).

```
snikulenkov@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-4.lst lab9-4.o
snikulenkov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
snikulenkov@vbox:~/work/arch-pc/lab09$ gbd
ash: gbd: command not found...
imilar command is: 'gdb'
snikulenkov@vbox:~/work/arch-pc/lab09$ ./lab9-4
ункция: f(x) = 10x - 4
езультат: 0
```

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.

4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. -fig. 4.17).

```
snikulenkov@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-4.lst lab9-4.o
snikulenkov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
snikulenkov@vbox:~/work/arch-pc/lab09$ gbd
ash: gbd: command not found...
imilar command is: 'gdb'
snikulenkov@vbox:~/work/arch-pc/lab09$ ./lab9-4
ункция: f(x) = 10x - 4
езультат: 0
```

Рис. 4.17: Измененная программа предыдущей лабораторной работы

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 отладчика и пошагово через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul esx можно заметить, что результат умножения записывается в регистр eax, но также меняет и edx. Значение регистра ebx не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию (рис. 4.18).

```

his GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
nable debuginfod for this session? (y or [n])
ebuginfod has been disabled.
o make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

reakpoint 1, _start () at lab9-4.asm:8
mov eax, msg_func
gdb) x/s *((void**)(esp + 4))
ffffffd22b: "/home/ssnikulenkov/work/arch-pc/lab09/lab9-4"
gdb) x/s *((void**)(esp + 8))
ffffffd258: "аргумент1"
gdb) x/s *((void**)(esp + 12))
ffffffd26a: "аргумент"
gdb) x/s *((void**)(esp + 16))
ffffffd27b: "2"
gdb) x/s *((void**)(esp + 20))
ffffffd27d: "аргумент 3"
gdb) x/s *((void**)(esp + 24))
x0: <error: Cannot access memory at address 0x0>
gdb)

```

Рис. 4.18: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -fig. 4.19).

```
ssnikulenkov@vbox:~$ nasm -f elf lab9-4.asm
nasm: fatal: unable to open input file `lab9-4.asm' No such file or directory
ssnikulenkov@vbox:~$ cd work/arch-pc/lab09/
ssnikulenkov@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
ssnikulenkov@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
ssnikulenkov@vbox:~/work/arch-pc/lab09$ ./lab9-4
Результат: 25
ssnikulenkov@vbox:~/work/arch-pc/lab09$
```

Рис. 4.19: 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.