

Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

-04-24

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	Реализация подпрограмм в NASM	7
3.2	Отладка программ с помощью GDB	9
3.3	Добавление точек останова	12
3.4	Работа с данными программы в GDB	13
3.5	Обработка аргументов командной строки в GDB	14
3.6	Задание для самостоятельной работы	16
4	Выводы	20

Список иллюстраций

3.1	Первые действия в лабораторной работе и результат программы .	7
3.2	Текст программы	8
3.3	Результат программы	9
3.4	Работа программы	9
3.5	Брейкпоинт	10
3.6	Дисассимилированный код программы	10
3.7	Отображение команд	11
3.8	Псевдографика.Режим	11
3.9	Информация о точке останова	12
3.10	Точка останова	12
3.11	Значения регистров	13
3.12	Просмотр значения переменной	13
3.13	Значение другой переменной	13
3.14	msg1=hello,	13
3.15	Измененные символы	14
3.16	Изменение ebx	14
3.17	Переход к отладчику	15
3.18	b-point + run	15
3.19	Позиции стека	16
3.20	Код программы	17
3.21	Результат	18
3.22	Ложный результат(символьный)	18
3.23	Код	19
3.24	Результат	19

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. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Задание для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

По аналогии с предыдущими лабораторными работами, файлы asm я буду создавать в каталоге “work/arch-pc”. В файле lab9-1.asm вписываю программу из листинга 9.1, которую я внимательно изучила. Создаю исполняемый файл и запускаю его в работу. (рис. 3.1).

```
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$ nasm -f elf lab9-1.asm
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$ ./lab9-1
Введите x: 2
2x+7=11
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$
```

Рис. 3.1: Первые действия в лабораторной работе и результат программы

После этого я переписываю программу согласно методическому материалу (рис. 3.2).

```

call sread
mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
;
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
mov [res],eax

ret ; выход из подпрограммы

```

Рис. 3.2: Текст программы

Создаю исполняемый файл и убеждаюсь в том, что все работает успешно.(рис. 3.3).

```
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$ nasm -f elf lab9-1.asm
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$ ./lab9-1
Введите x: 2
2(3x-1)+7=17
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$ ./lab9-1
Введите x: 3
2(3x-1)+7=23
aesandan@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/lab09$
```

Рис. 3.3: Результат программы

3.2 Отладка программ с помощью GDB

В новом файле lab9-2.asm записываю текст программы из листинга 9.2. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ провожу с ключом '-g'. И загружаю файл в отладчик gbd, а затем проверяю выполнение программы с помощью команды run (рис. 3.4).

```
GNU GDB (Free Software Foundation, Inc.
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) run
Starting program: /home/aesandan/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09/report/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.
Hello, world!
[Inferior 1 (process 57961) exited normally]
(gdb)
```

Рис. 3.4: Работа программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаю её. (рис. 3.5).

```

This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvmaksimova/work/arch-pc/lab9/lab9-2
Hello, world!
[Inferior 1 (process 4723) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/d/v/dvmaksimova/work/arch-pc/lab9/lab9-2

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

```

Рис. 3.5: Брейкпоинт

Просматриваю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 3.6).

```

(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.
(gdb)

```

Рис. 3.6: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 3.7).

```
(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. 3.7: Отображение команд

Есть некоторые различия в отображениях в этих режимах, а именно в виде колонки с текстом программы: в Intel'е она выглядит, как “\$0x{операнд},%{регистер}”, а в АТТ - “{регистер},0x{операнд}”

Включаю режим псевдографики для удобного анализа программ (рис. 3.8).

```
B> 0x08049000 <_start>    mov     eax,0x4
      0x08049005 <_start+5>    mov     ebx,0x1
      0x0804900a <_start+10>   mov     ecx,0x804a000
      0x0804900f <_start+15>   mov     edx,0x8
      0x08049014 <_start+20>   int     0x80
      0x08049016 <_start+22>   mov     eax,0x4
      0x0804901b <_start+27>   mov     ebx,0x1
      0x08049020 <_start+32>   mov     ecx,0x804a008
      0x08049025 <_start+37>   mov     edx,0x7

native process 58000 In: _start          L9    PC: 0x08049000
(gdb) layout regs
(gdb)
```

Рис. 3.8: Псевдографика.Режим

3.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверяю это с помощью команды `info breakpoints` (рис. 3.9).

```
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

native process 58000 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab9-2.asm:9
breakpoint already hit 1 time
(gdb)
```

Рис. 3.9: Информация о точке останова

Я установила еще одну точку останова по адресу инструкции, и теперь когда мы запрашиваем информацию о брейкпоинтах, нам показываются две точки останова: (рис. 3.10).

```
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

native process 58000 In: _start
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab9-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab9-2.asm:20
(gdb)
```

Рис. 3.10: Точка останова

3.4 Работа с данными программы в GDB

Для того, что бы посмотреть значения регистров можно воспользоваться командой “i r”. Вот какие значения регистров вывелись (рис. 3.11).

```
Active process 0000 0x_start 0
eax 0x0
ecx 0x0
edx 0x0
ebx 0x0
ecx 0x0
ebp 0x0
ecx 0x0
edi 0x0
esi 0x0
eip 0x00000000 0x00000000 0x_start
eflags 0x00000000 0x00000000 0x_start
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 3.11: Значения регистров

С помощью команды x & также можно посмотреть содержимое переменной. Я посмотрела значение переменной msg1 по имени и вот результат. (рис. 3.12).

```
gs 0x0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
```

Рис. 3.12: Просмотр значения переменной

Аналогично можно посмотреть значение переменной msg2 по адресу. (рис. 3.13).

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

Рис. 3.13: Значение другой переменной

Теперь я изменяю первый символ переменной msg1 (рис. 3.14).

```
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)
```

Рис. 3.14: msg1=hello,

А также я изменила разные символы в переменной msg2 (рис. 3.15).

```
0x804a000 <msg1>:      "Hello, "  
(gdb) set {char}0x804a00d='a'  
(gdb) x/1sb &msg2  
0x804a008 <msg2>:      "worlda\n\034"  
(gdb)
```

Рис. 3.15: Измененные символы

Теперь изменим значение регистра ebx: (рис. 3.16).

```
0x804a008 <msg2>:      "worlda\n\034"  
(gdb) set $ebx='2'  
(gdb) p/s $ebx  
$1 = 50  
(gdb)
```

Рис. 3.16: Изменение ebx

Команда set позволяе изменить значение регистра, проделываем это с регистром ebx, и использую команду p/s, чтоб промотреть значение.

3.5 Обработка аргументов командной строки в GDB

Создаю исполняемый файл из файла, с которым я работала в процессе выполнения лабораторной работы №8.

А затем загружаю данный файл в отладчик, предварительно указав ключ к работе с аргументами, и указываю эти самые аргументы (рис. 3.17).

```

aesandan@fedora:~/work/study/2024-2025/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
aesandan@fedora:~/work/study/2024-2025/lab09$ gdb --args lab9-3 аргумент1 аргумент 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 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"...

```

Рис. 3.17: Переход к отладчику

Для начала устанавливаем первую точку останова, а затем запускаем программу (рис. 3.18).

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) r
Starting program: /home/aesandan/work/study/2024-2025/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

```

Рис. 3.18: b-point + run

На данном этапе, мы можем посмотреть позиции стека отдельно по адресам (рис. 3.19).

```

(gdb) x/x $esp
0xfffffd050: 0x00000005
(gdb) x/s *(void**)($esp + 4)
0xfffffd210: "/home/aesandan/work/study/2024-2025/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xfffffd241: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xfffffd253: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xfffffd264: "2"
(gdb) x/s *(void**)($esp + 20)
0xfffffd266: "аргумент 3"

```

Рис. 3.19: Позиции стека

3.6 Задание для самостоятельной работы

Переходим к выполнению заданий самостоятельного характера.

№1

Я преобразовываю программу, с которой я работала в процесее 8 лабораторной работы. Я должна реализовать вычисление функции в подпрограмме (до этого вычисления были в основной программе).

Вот таким образом выглядит код программы: (рис. 3.20).


```

1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db "Функция: f(x)=3x-1", 0
4 msg2 db "Результат: ", 0
5 SECTION .text
6 global _start
7 _start:
8
9 pop ecx
10 pop edx
11 sub ecx, 1
12 mov esi, 0
13
14 next:
15 cmp ecx, 0h
16 jz _end
17 pop eax
18 call atoi
19
20 call mom
21
22 add esi, eax
23 loop next
24 _end:
25 mov eax, msg1
26 call sprintLF
27 mov eax, msg2
28 call sprint
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 mom:
34 mov ebx, eax
35 add eax, ebx
36 add eax, ebx
37 sub eax, 1
38 ret
39

```

Рис. 3.20: Код программы

4 Выводы

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