

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
Факультет физико-математических и естественных наук
Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №9
дисциплина: Архитектура компьютеров и операционные системы

Студент: Галиев Самир Салаватович

Группа: НКАбд-02-25

МОСКВА

2025 г.

Оглавление

1. Цель работы - стр. 3
2. Результаты выполнения лабораторной работы - стр. 3
 - 2.1 Реализация подпрограммы в NASM - стр. 3
 - 2.2 Отладка программы с помощью GDB- стр. 4
 - 2.3 Анализ аргументов командной строки в GDB - стр. 8
3. Задание для самостоятельной работы — стр. 9
 - 3.1 Реализация функции $f(x) = 15*x + 2$ в виде подпрограммы — стр. 9
 - 3.2 Исправление ошибки в листинге 9.3 — стр. 9
4. Выводы - стр. 11
5. Список литературы - стр. 12

1. Цель работы

Приобретение навыков написания программ с использованием подпрограмм на ассемблере NASM, а также освоение методов отладки с помощью отладчика GDB: установка точек останова, пошаговое выполнение, просмотр и изменение содержимого регистров и памяти.

2. Выполнение задания

2.1 Реализация подпрограмм в NASM

Описание задания: Написать программу lab09-1.asm, вычисляющую значение функции $f(x)=2x+7$ с использованием подпрограммы _calcul. Затем модифицировать программу, вычисляющую $g(x)=3x-1$, и реализовать композицию $f(g(x))$.

```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nasm -f elf32 lab09-1.asm -o lab09-1.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % ld -m elf_i386 lab09-1.o -o lab09-1
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % ./lab09-1
Введите x:3
2x+7=13
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 %
```

Рисунок 1: вывод программы lab09-1.asm при $x = 3$

Комментарий: Программа запрашивает x и вычисляет $2x+7$. При вводе $x=3$ выводится 13 (результат верный).

```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nano lab09-1.asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nasm -f elf32 lab09-1.asm -o lab09-1.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % ld -m elf_i386 lab09-1.o -o lab09-1
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % ./lab09-1
Введите x:2
2x+7=17
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 %
```

Рисунок 2: вывод программы вычисляющая $f(g(x))$

Комментарий: Программа вычисляет $f(g(x))=2(3x-1)+7$. При $x = 2$ вывод 17(ответ верный).

2.2 Откладка программы с помощью GDB

Описание задания: Отладить программу lab09-2.asm (Hello, world!) с использованием GDB: запуск с точкой останова, пошаговое выполнение, просмотр и изменение памяти и регистров.

```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nano lab09-2.asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nasm -f elf32 -g -l lab09-2.lst lab09-2.asm -o lab09-2.o
ld -m elf_i386 -o lab09-2 lab09-2.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % ./lab09-2
Hello,world!
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 %
```

Рисунок 3: Вывод программы lab09-2.asm

The screenshot shows a debugger interface with two main panes. The top pane displays assembly code with addresses and opcodes. The bottom pane shows the register state and a command-line interface for interacting with the debugger.

Registers:

Register	Value	Description
eax	0x7	7
ecx	0x804a000	134520832
edx	0x7	7
ebx	0x1	1
esp	0xfffffd630	0xfffffd630
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

Assembly Code:

```
B+ 0x8049000 <_start>    mov    eax, 0x4
0x8049005 <_start+5>    mov    ebx, 0x1
0x804900a <_start+10>   mov    ecx, 0x804a000
0x804900f <_start+15>   mov    edx, 0x7
0x8049014 <_start+20>   int    0x80
>0x8049016 <_start+22>  mov    eax, 0x4
0x804901b <_start+27>    mov    ebx, 0x1
0x8049020 <_start+32>    mov    ecx, 0x804a007
0x8049025 <_start+37>    mov    edx, 0x7
0x804902a <_start+42>    int    0x80
0x804902c <_start+44>    mov    eax, 0x1
0x8049031 <_start+49>    mov    ebx, 0x0
0x8049036 <_start+54>    int    0x80
0x8049038                  add    BYTE PTR [eax], al
```

Debugger Command Line:

```
native process 8204 (asm) In: _start          L16   PC: 0x8049016
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рисунок 4: После пяти si регистры содержат eax = 4, ebx = 1, ecx = msg1, edx = 6.

Комментарий: Включен удобный режим интерфейса (Верхняя панель — регистры, Средняя — дизассемблированный код, а Нижняя — командная строка)

The screenshot shows the Immunity Debugger interface. The top section displays assembly code, the middle section shows register values, and the bottom section is a command-line interface for interacting with the debugger.

Registers (General):

Register	Value	Description
eax	0x7	7
ecx	0x804a000	134520832
edx	0x7	7
ebx	0x1	1
esp	0xfffffd630	0xfffffd630
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

Assembly:

```
B+ 0x8049000 <_start>      mov    eax,0x4
 0x8049005 <_start+5>      mov    ebx,0x1
 0x804900a <_start+10>     mov    ecx,0x804a000
 0x804900f <_start+15>     mov    edx,0x7
 0x8049014 <_start+20>     int    0x80
>0x8049016 <_start+22>     mov    eax,0x4
 0x804901b <_start+27>     mov    ebx,0x1
 0x8049020 <_start+32>     mov    ecx,0x804a007
 0x8049025 <_start+37>     mov    edx,0x7
 0x804902a <_start+42>     int    0x80
 0x804902c <_start+44>     mov    eax,0x1
 0x8049031 <_start+49>     mov    ebx,0x0
 0x8049036 <_start+54>     int    0x80
 0x8049038           add    BYTE PTR [eax],al
```

Command Line (GDB):

```
native process 8204 (asm) In: _start
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) x/1sb (char*)&msg1
0x804a000 <msg1>:      "Hello,"
(gdb) x/1sb (char*)&msg2
0x804a007 <msg2>:      "world!\n\034"
(gdb) █
```

Рисунок 5: После двух команд вызова msg1 и msg2 нам показывают строки «Hello,» и «world!\n\034»

Комментарий: использование команд x/1sb (char*)&msg1 и x/1sb (char*)&msg2 показывают исходные строки.

Register group: general

eax	0x7	7
ecx	0x804a000	134520832
edx	0x7	7
ebx	0x1	1
esp	0xfffffd630	0xfffffd630
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

```
B+ 0x8049000 <_start>    mov    eax, 0x4
0x8049005 <_start+5>    mov    ebx, 0x1
0x804900a <_start+10>   mov    ecx, 0x804a000
0x804900f <_start+15>   mov    edx, 0x7
0x8049014 <_start+20>   int    0x80
>0x8049016 <_start+22>  mov    eax, 0x4
0x804901b <_start+27>   mov    ebx, 0x1
0x8049020 <_start+32>   mov    ecx, 0x804a007
0x8049025 <_start+37>   mov    edx, 0x7
0x804902a <_start+42>   int    0x80
0x804902c <_start+44>   mov    eax, 0x1
0x8049031 <_start+49>   mov    ebx, 0x0
0x8049036 <_start+54>   int    0x80
0x8049038          add    BYTE PTR [eax], al
```

native process 8204 (asm) In: _start L16 PC: 0x8049016

```
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) x/1sb (char*)&msg1
0x804a000 <msg1>:      "Hello,"
(gdb) x/1sb (char*)&msg2
0x804a007 <msg2>:      "world!\n\034"
(gdb) set *((char*)&msg1) = 'h'
(gdb) x/1sb (char*)&msg1
0x804a000 <msg1>:      "hello,"
(gdb)
```

Рисунок 6: замена первой буквы в слове «Hello»

Комментарий: С помощью `set *((char*)&msg1) = 'h'` первая буква изменена. Повторный вывод показывает: "hello,".

Register group: general

eax	0x7	7
ecx	0x804a000	134520832
edx	0x7	7
ebx	0x2	2
esp	0xfffffd630	0xfffffd630
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

```
B+ 0x8049000 <_start>    mov    eax, 0x4
0x8049005 <_start+5>    mov    ebx, 0x1
0x804900a <_start+10>   mov    ecx, 0x804a000
0x804900f <_start+15>   mov    edx, 0x7
0x8049014 <_start+20>   int    0x80
>0x8049016 <_start+22>  mov    eax, 0x4
0x804901b <_start+27>   mov    ebx, 0x1
0x8049020 <_start+32>   mov    ecx, 0x804a007
0x8049025 <_start+37>   mov    edx, 0x7
0x804902a <_start+42>   int    0x80
0x804902c <_start+44>   mov    eax, 0x1
0x8049031 <_start+49>   mov    ebx, 0x0
0x8049036 <_start+54>   int    0x80
0x8049038          add    BYTE PTR [eax], al
```

```
native process 8204 (asm) In: _start
(gdb) set *((char*)&msg1) = 'h'
(gdb) x/1sb (char*)&msg1
0x804a000 <msg1>:      "hello,"
(gdb) p/x $edx
$1 = 0x7
(gdb) p/t $edx
$2 = 111
(gdb) p/c $edx
$3 = 7 '\a'
(gdb) set $ebx = 2
(gdb) p $ebx
$4 = 2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рисунок 7: вывод переменных в шестизначном значении и д.р. и т. д.

Комментарий: Команда p/s \$ebx пытается интерпретировать число 2 как адрес строки, что в корректной отладочной среде приводит к ошибке или мусору. Это демонстрирует разницу между числовым значением и указателем на строку.
 (Вывод \$5 = 2 это упрощённое поведение GDB для удобства, но суть не отменяет)

2.3 Анализ аргументов командной строки в GDB

Описание задания: Использовать GDB для изучения расположения аргументов командной строки в стеке при запуске программы.

The screenshot shows the GDB interface with two panes. The top pane is titled 'Консоль' (Console) and displays the contents of the general register group:

Register group: general		
eax	0x0	0
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xfffffd610	0xfffffd610
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0xf7fe3460	0xf7fe3460 <_start>
eflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43

The bottom pane is titled '[No Assembly Available]' and contains the assembly dump of the program:

```
native process 11999 (asm) In: _start
(gdb) layout regs
(gdb) x/x $esp
0xfffffd610:    0x00000004
(gdb) x/s *(void**)( $esp + 4)
0xfffffd7f8:    "/home/ssgaliev/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xfffffd85e:    "arg1"
(gdb) x/s *(void**)( $esp + 12)
0xfffffd863:    "arg2"
(gdb) x/s *(void**)( $esp + 16)
0xfffffd868:    "test"
(gdb)
```

Рисунок 8: запуск программы с аргументами arg1, arg2, test

Комментарий: +8, +12, +16 при arg1, arg2, test.

Шаг = 4 байта, так как в 32-битной системе указатель занимает 4 байта.

3. Самостоятельная работа

3.1 Реализация функции $f(x) = 15*x+2$ в виде подпрограммы

Описание задания: Преобразовать программу из лабораторной работы №8 в версию с подпрограммой `_f`, вычисляющей $f(x)=15x+2$ для каждого аргумента и суммирующей результаты.

```
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nano lab09-4.asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nasm -f elf32 lab
09-4.asm -o lab09-4.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % gcc -m32 -nostdli
b lab09-4.o -o lab09-4
/usr/bin/ld: lab09-4.o: предупреждение: перемещение в разделе только для чтения «.text»
/usr/bin/ld: предупреждение: создаётся DT_TEXTREL в PIE
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % ./lab09-4 1 2 3
Функция: f(x)=15x+2
Результат:96
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 %
```

Рисунок 9: Вывод программы.

Комментарий: Программа запущена с аргументами 1 2 3. Вывод 96.

3.2 Исправление ошибки в листинге 9.3

Описание задания: В листинге 9.3 вычисляется выражение $(3+2)\cdot 4+5=25$, но из-за ошибки в использовании `mul` программа выводит неверный результат. Требуется найти и исправить ошибку с помощью GDB.

```
Результат:96
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nano lab09-5.asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nasm -f elf32 lab
09-5.asm -o lab09-5.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % gcc -m32 -nostdli
b lab09-5.o -o lab09-5
/usr/bin/ld: lab09-5.o: предупреждение: перемещение в разделе только для чтения «.text»
/usr/bin/ld: предупреждение: создаётся DT_TEXTREL в PIE
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % ./lab09-5
Результат: 10
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 %
```

Рисунок 10: Неверный вывод программы.

Комментарий: До исправления программа выводит 10, что неверно. Ошибка: `mul` есх умножает `eax`, а не `ebx`.

```
(gdb) layout asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nano lab09-5.asm
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % nasm -f elf32 lab
09-5.asm -o lab09-5.o
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % gcc -m32 -nostdli
b lab09-5.o -o lab09-5
/usr/bin/ld: lab09-5.o: предупреждение: перемещение в разделе только для чтения «.text»
/usr/bin/ld: предупреждение: создаётся DT_TEXTREL в PIE
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 % ./lab09-5
Результат: 25
ssgaliev@linux ~/work/study/2025-2026/Архитектура компьютера/arch-pc/lab09 %
```

Рисунок 11: Корректный вывод программы после исправления.

Комментарий: После исправления сумма 3+2 помещена в eax, mul ebx(где ebx = 4), затем add eax, 5. В итоге мы получаем верный результат(25).

4. Выводы

Подпрограммы (call/ret) позволяют структурировать код, изолировать логику и повторно использовать функциональность.

1) Стек используется для передачи адреса возврата, а также для временного хранения данных.

2) GDB — мощный инструмент для отладки:

- break, si, continue — управление выполнением,
- x, p, set — анализ и изменение памяти/регистров,
- layout asm, layout regs — удобный режим интерфейса.
- При работе с mul важно помнить: первый операнд — всегда eax.
- Аргументы командной строки размещаются в стеке: argc, имя программы, затем аргументы (по 4 байта на указатель в 32-битной системе).

Цель работы полностью достигнута.

5. Список литературы

- 1)Демидова А. В. Архитектура ЭВМ. Лабораторная работа №9.
- 2)Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — М.: МАКС Пресс, 2011.
- 3)GDB: The GNU Project Debugger. — <https://www.gnu.org/software/gdb/>
- 4)The NASM documentation. — <https://www.nasm.us/docs.php>
- 5)Debuginfod — Arch Linux Wiki. — <https://wiki.archlinux.org/title/Debuginfod>