

Отчёт по лабораторной работе №9

Кучерова Виктория Васильевна

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Задание для самостоятельной работы	21
4	Выводы	25
	Список литературы	26

Список иллюстраций

2.1	Создание файла	7
2.2	Программа	8
2.3	Запуск	8
2.4	Программа	9
2.5	Запуск	10
2.6	Программа	11
2.7	Исполняемый файл	12
2.8	Команда <code>run</code>	12
2.9	брейкпоинт	12
2.10	Дисассимилированный код	13
2.11	Intel'овский синтаксис	13
2.12	Режим псевдографики	14
2.13	Команда <code>info breakpoints</code>	14
2.14	Еще одна точка останова	14
2.15	Точки останова	15
2.16	<code>stepi 1</code>	15
2.17	<code>stepi 2</code>	16
2.18	<code>stepi 3</code>	16
2.19	<code>stepi 4</code>	17
2.20	<code>stepi 5</code>	17
2.21	Значение переменной <code>msg1</code>	18
2.22	Значение переменной <code>msg2</code>	18
2.23	Изменение <code>msg1</code>	18
2.24	Изменение <code>msg2</code>	18
2.25	Значение регистра <code>edx</code>	18
2.26	Изменение значения регистра <code>ebx</code>	19
2.27	Выход	19
2.28	Копирование	19
2.29	Отладчик	19
2.30	Запуск	19
2.31	Регистр <code>esp</code>	20
2.32	Остальные позиции стека	20
3.1	Программа	22
3.2	Запуск	22
3.3	Неверный результат	23
3.4	Ошибка	23

3.5	Исправление	24
3.6	Запуск	24

Список таблиц

1 Цель работы

Приобрести навыки написания программ с использованием подпрограмм. Познакомиться с методами отладки при помощи GDB и его основными возможностями.

2 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы № 9, перейдем в него и создадим файл lab09-1.asm:(рис. 2.1).

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

Рис. 2.1: Создание файла

Введем в файл lab09-1.asm текст программы. Создадим исполняемый файл и проверим его работу(рис. 2.2).(рис. 2.3).

```

%include 'in_out.asm'

SECTION .data
    msg: DB 'Введите x: ',0
    result: DB '2x+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:
    mov ebx, 2
    mul ebx
    add eax, 7

```

Рис. 2.2: Программа

```

vvkucheroa@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vvkucheroa@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vvkucheroa@vbox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17

```

Рис. 2.3: Запуск

Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры(рис. 2.4).(рис. 2.5).

```
_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:
    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
```

Рис. 2.4: Программа

```
vvkucheroval@vbox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
vvkucheroval@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
vvkucheroval@vbox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=35
```

Рис. 2.5: Запуск

Создадим файл lab09-2.asm с текстом программы(рис. 2.6).

```

SECTION .data
    msg1: db "Hello, ",0x0
    msg1len: equ $ - msg1

    msg2: db "world!",0xa
    msg2len: equ $ - msg2

SECTION .text
    global _start

_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, msg1
    mov edx, msg1len
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, msg2
    mov edx, msg2len
    int 0x80

    mov eax, 1
    mov ebx, 0
    int 0x80

```

Рис. 2.6: Программа

Получим исполняемый файл и загрузим исполняемый файл в отладчик gdb(рис. 2.7).

```

vvkucheroва@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
vvkucheroва@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
vvkucheroва@vbox:~/work/arch-pc/lab09$ gdb lab09-2
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.

```

Рис. 2.7: Исполняемый файл

Проверим работу программы, запустив ее в оболочке GDB с помощью команды `run`(рис. 2.8).

```

(gdb) run
Starting program: /home/vvkucheroва/work/arch-pc/lab09/lab09-2

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 4272) exited normally]

```

Рис. 2.8: Команда `run`

Для более подробного анализа программы установим брейкпоинт на метку `_start`(рис. 2.9).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 12.
(gdb) run
Starting program: /home/vvkucheroва/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:12
12      mov eax, 4

```

Рис. 2.9: брейкпоинт

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`(рис. 2.10).

```

(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. 2.10: Дисассимилированный код

Переключитесь на отображение команд с Intel'овским синтаксисом. В режиме АТТ регистры указываются после их адреса и перед регистром стоит % (рис. 2.11).

```

(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.

```

Рис. 2.11: Intel'овский синтаксис

Включим режим псевдографики для более удобного анализа программы (рис. 2.12).

```

[ Register Values Unavailable ]

B>0x8049000 <_start>    mov     eax,0x4
0x8049005 <_start+5>    mov     ebx,0x1
0x804900a <_start+10>   mov     ecx,0x804a000
0x804900f <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov     eax,0x4

native process 4300 In: _start L12 PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 2.12: Режим псевдографики

Ранее была установлена точка останова по имени метки (`_start`). Проверим это с помощью команды `info breakpoints`(рис. 2.13).

```

(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:12
breakpoint already hit 1 time
(gdb)

```

Рис. 2.13: Команда `info breakpoints`

Установим еще одну точку останова по адресу инструкции(рис. 2.14).

```

(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 25.

```

Рис. 2.14: Еще одна точка останова

Посмотрим информацию о всех установленных точках останова(рис. 2.15).

```
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:12
          breakpoint already hit 1 time
2        breakpoint keep y  0x08049031 lab09-2.asm:25
```

Рис. 2.15: Точки останова

Выполним 5 инструкций с помощью команды stepi(рис. 2.16).(рис. 2.17).(рис. 2.18).(рис. 2.19).(рис. 2.20).

```
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0

B+ 0x8049000 <_start>      mov     eax,0x4
>0x8049005 <_start+5>      mov     ebx,0x1
0x804900a <_start+10>      mov     ecx,0x804a000
0x804900f <_start+15>      mov     edx,0x8
0x8049014 <_start+20>      int     0x80
0x8049016 <_start+22>      mov     eax,0x4

native process 4300 In: _start L13 PC: 0x8049005
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 25.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:12
          breakpoint already hit 1 time
2        breakpoint keep y  0x08049031 lab09-2.asm:25
(gdb) si
```

Рис. 2.16: stepi 1

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0

B+ 0x8049000 <_start>    mov    eax,0x4
   0x8049005 <_start+5>  mov    ebx,0x1
>0x804900a <_start+10>   mov    ecx,0x804a000
   0x804900f <_start+15>  mov    edx,0x8
   0x8049014 <_start+20>  int    0x80
   0x8049016 <_start+22>  mov    eax,0x4

native process 4300 In: _start L14 PC: 0x804900a
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y 0x08049000 lab09-2.asm:12
          breakpoint already hit 1 time
2        breakpoint    keep y 0x08049031 lab09-2.asm:25
(gdb) si
(gdb) si

```

Рис. 2.17: stepi 2

```

Register group: general
eax      0x4      4
ecx      0x804a000    134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0

B+ 0x8049000 <_start>    mov    eax,0x4
   0x8049005 <_start+5>  mov    ebx,0x1
   0x804900a <_start+10>  mov    ecx,0x804a000
>0x804900f <_start+15>  mov    edx,0x8
   0x8049014 <_start+20>  int    0x80
   0x8049016 <_start+22>  mov    eax,0x4

native process 4300 In: _start L15 PC: 0x804900f
Num      Type          Disp Enb Address      What
1        breakpoint    keep y 0x08049000 lab09-2.asm:12
          breakpoint already hit 1 time
2        breakpoint    keep y 0x08049031 lab09-2.asm:25
(gdb) si
(gdb) si
(gdb) si

```

Рис. 2.18: stepi 3


```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0

0x8049005 <_start+5>  mov     ebx,0x1
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

native process 4300 In: _start L16 PC: 0x8049014
1 breakpoint keep y 0x08049000 lab09-2.asm:12
  breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:25
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Рис. 2.19: step1 4

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0x0

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

native process 4300 In: _start L18 PC: 0x8049016
  breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:25
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

```

Рис. 2.20: step1 5

Посмотрим значение переменной msg1 по имени(рис. 2.21).

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

Рис. 2.21: Значение переменной msg1

Посмотрим значение переменной msg2 по адресу(рис. 2.22).

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

Рис. 2.22: Значение переменной msg2

Изменим первый символ переменной msg1(рис. 2.23).

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

Рис. 2.23: Изменение msg1

Заменяем любой символ во второй переменной msg2(рис. 2.24).

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

Рис. 2.24: Изменение msg2

Выведем в различных форматах значение регистра edx(рис. 2.25).

```
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
```

Рис. 2.25: Значение регистра edx

С помощью команды set изменим значение регистра ebx. В первом случае мы вносим значение 2, а во втором регистр равен 2(рис. 2.26).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
```

Рис. 2.26: Изменение значения регистра ebx

Выйдем из GDB(рис. 2.27).

```
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:25
(gdb) q
```

Рис. 2.27: Выход

Скопируем файл lab8-2.asm в файл с именем lab09-3.asm и создадим исполняемый файл(рис. 2.28).

```
vvkucheroval@vbox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/
/arch-pc/lab09/lab09-3.asm
vvkucheroval@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
vvkucheroval@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 2.28: Копирование

Загрузим исполняемый файл в отладчик, указав аргументы(рис. 2.29).

```
vvkucheroval@vbox:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 '
аргумент 3'
GNU gdb (Fedora Linux) 14.2-1.fc40
```

Рис. 2.29: Отладчик

Для начала установим точку останова перед первой инструкцией в программе и запустим ее(рис. 2.30).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 7.
(gdb) run
```

Рис. 2.30: Запуск

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки(рис. 2.31).

```
(gdb) x/x $esp
0xffffd050: 0x00000005
```

Рис. 2.31: Регистр esp

Посмотрим остальные позиции стека. Элементы расположены с интервалом 4, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально без помех, компьютер использует новый стек для новой информации(рис. 2.32).

```
(gdb) x/s *(void**)(esp + 4)
0xffffd21a: "/home/vvkucherovalwork/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd247: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd259: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd26a: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd26c: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 2.32: Остальные позиции стека

3 Задание для самостоятельной работы

Преобразуем программу из лабораторной работы №8, реализовав вычисление значения функции $f(x)$ как подпрограмму(рис. 3.1).(рис. 3.2).

```

    sub ecx,1

    mov esi, 0

next:
    cmp ecx,0h
    jz _end

    pop eax
    call atoi

    call _calcul

    add esi,eax

    loop next

_end:
    mov eax, msg2
    call sprint
    mov eax, esi
    call iprintLF
    call quit

_calcul:
    mov ebx,8
    mul ebx
    sub eax,3
    mov [res],eax

    ret

```

Рис. 3.1: Программа

```

vvkucheroва@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab0
9$ nasm -f elf lab9-3.asm
vvkucheroва@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab0
9$ ld -m elf_i386 -o lab9-3 lab9-3.o
vvkucheroва@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab0
9$ ./lab9-3 12 13 7 10 5
Функция:  $f(x)=8x - 3$ 
Результат: 361

```

Рис. 3.2: Запуск

При запуске данная программа дает неверный результат. Проверим это. С по-

мощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправьте ее. Ошибка в том что сумма записывается в ebx, а на 4 умножается eax(рис. 3.3).(рис. 3.4).(рис. 3.5).(рис. 3.6).

```
vvkucheroval@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ nasm -f elf lab09-3.asm
vvkucheroval@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
vvkucheroval@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab09$ ./lab09-3
Результат: 10
```

Рис. 3.3: Неверный результат

Register group: general

Register	Value (hex)	Value (dec)
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffcfc0	0xffffcfc0
ebp	0x0	0x0

Address	Disassembly
0x80490f2 <_start+10>	add ebx, eax
0x80490f4 <_start+12>	mov ecx, 0x4
0x80490f9 <_start+17>	mul ecx
>0x80490fb <_start+19>	add ebx, 0x5
0x80490fe <_start+22>	mov edi, ebx
0x8049100 <_start+24>	mov eax, 0x804a000

native process 7274 In: _start L15 PC: 0x80490fb

Register	Value (hex)	Value (dec)
fs	0x0	0
gs	0x0	0

(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si

Рис. 3.4: Ошибка

```

%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

    mov ebx,3
    mov eax,2
    add 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

```

Рис. 3.5: Исправление

```

vvkucheroва@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab0
9$ nasm -f elf lab09-3.asm
vvkucheroва@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab0
9$ ld -m elf_i386 -o lab09-3 lab09-3.o
vvkucheroва@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab0
9$ ./lab09-3
Результат: 25

```

Рис. 3.6: Запуск

4 Выводы

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

Список литературы