

Отчёт по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB

Грачева Мария Валерьевна

Содержание

1	Цель работы	6
2	Теоретическое введение	7
3	Выполнение лабораторной работы	8
4	Самостоятельная работа	31
5	Выводы	37
	Список литературы	38

Список иллюстраций

3.1	Создание каталога и файл lab09-1	8
3.2	Введение программы	9
3.3	Проверка работы	9
3.4	Изменение текста	10
3.5	Проверка работы 2	10
3.6	Создание файла lab09-2	11
3.7	Введение программы	12
3.8	Создание исполняемого файла	13
3.9	Загрузка исполняемого файла в отладчик gdb	13
3.10	Проверка работы 3	13
3.11	Установка брейкпоинта	13
3.12	Запуск программы	14
3.13	Дисассимилированный код программы	14
3.14	Отображение команд с Intel'овским синтаксисом	15
3.15	Режим псевдографики 1	16
3.16	Режим псевдографики 2	16
3.17	Команда info breakpoints	17
3.18	Точка останова 1	17
3.19	Точка останова 2	17
3.20	Команда info breakpoints 2	17
3.21	Выполнение команды si 1	18
3.22	Выполнение команды si 2	19
3.23	Выполнение команды si 3	20
3.24	Выполнение команды si 4	21
3.25	Выполнение команды si 5	22
3.26	Команда info registers	23
3.27	Значение переменной msg1 по имени	23
3.28	Значение переменной msg2 по адресу	24
3.29	Замена символа 1	24
3.30	Замена символа 2	25
3.31	Замена символа 3	25
3.32	Изменение значение регистра ebx	26
3.33	Завершение программы	27
3.34	Копирование файла	27
3.35	Создание исполняемого файла	28
3.36	Загрузка в отладчик	28
3.37	Точка останова	29

3.38	Запуск файла	29
3.39	Команда x/x	29
3.40	Команда x/s	29
3.41	Команда x/s 2	29
3.42	Команда x/s 3	30
3.43	Команда x/s 4	30
3.44	Команда x/s 5	30
3.45	Команда x/s 6	30
4.1	Копирование файла, созданного в ходе выполнения предыдущей лабораторной	31
4.2	Изменения в текст	32
4.3	Проверка работы 4	33
4.4	Создание файла task2	33
4.5	Введение текста	34
4.6	Проверка работы 5	34
4.7	Поиск ошибки	35
4.8	Правки в исходной программе	35
4.9	Проверка работы 6	36

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм.
Знакомство с методами отладки при помощи GDB и его основными возможностями

2 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

Наиболее часто применяют следующие методы отладки: • создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения); • использование специальных программ-отладчиков.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ.

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки.

Информацию о всех установленных точках останова можно вывести командой `info`

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`

Обратно точка останова активируется командой `enable`

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

3 Выполнение лабораторной работы

Создаю каталог и файл lab09-1 (рис. 4.9).

```
mvgracheva@dk4n65 ~ $ mkdir ~/work/arch-pc/lab09
mvgracheva@dk4n65 ~ $ cd ~/work/arch-pc/lab09
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ touch lab09-1.asm
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Создание каталога и файл lab09-1

Ввожу программу (рис. 4.9).


```

lab09-1.asm      [-M--] 27 L:[ 1+34 35/ 35] *(707 / 70
#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:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
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:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы

```

Рис. 3.2: Введение программы

Проверяю работу файла (рис. 4.9).

```

mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 3
2x+7=13
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $

```

Рис. 3.3: Проверка работы

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $x(x(x))$, где x вводится с клавиатуры, $x(x) = 2x + 7$, $x(x) = 3x - 1$. (рис. 4.9)

```
_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx,3
mul ebx
sub eax, 1
mov [res],eax
ret
```

Рис. 3.4: Изменение текста

Проверяю его работу (рис. 4.9).

```
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 3
2x+7=23
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $
```

Рис. 3.5: Проверка работы 2

Создаю файл `lab09-2` (рис. 4.9).

```
2x17-23  
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ touch lab09-2.asm  
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $
```

Рис. 3.6: Создание файла lab09-2

Ввожу текст программы (рис. 4.9).

```
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. 3.7: Введение программы

Создаю исполняемый файл (рис. 4.9).

```
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $
```

Рис. 3.8: Создание исполняемого файла

Загружаю исполняемый файл в отладчик gdb (рис. 4.9).

```
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-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 lab09-2...
(gdb)
```

Рис. 3.9: Загрузка исполняемого файла в отладчик gdb

Проверяю работу программы (рис. 4.9).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/v/mvgracheva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4103) exited normally]
(gdb)
```

Рис. 3.10: Проверка работы 3

Устанавливаю брейкпоинт на метку _start (рис. 4.9).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb)
```

Рис. 3.11: Установка брейкпоинта

Запускаю (рис. 4.9).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/v/mvgracheva/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 3.12: Запуск программы

Смотрю дисассимилированный код программы(рис. 4.9).

```
(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. 3.13: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel'овским синтаксисом (рис. 4.9).

```

(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. 3.14: Отображение команд с Intel'овским синтаксисом

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

```

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
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
0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al
0x804903c add BYTE PTR [eax],al
0x804903e add BYTE PTR [eax],al
0x8049040 add BYTE PTR [eax],al
0x8049042 add BYTE PTR [eax],al
0x8049044 add BYTE PTR [eax],al
0x8049046 add BYTE PTR [eax],al
0x8049048 add BYTE PTR [eax],al
0x804904a add BYTE PTR [eax],al
0x804904c add BYTE PTR [eax],al
0x804904e add BYTE PTR [eax],al
0x8049050 add BYTE PTR [eax],al
native process 4134 In: _start L9 PC: 0x8049000

```

Рис. 3.15: Режим псевдографики 1

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B> 0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx
0x8049025 <_start+37> mov $0x7,%edx
0x804902a <_start+42> int $0x80
0x804902c <_start+44> mov $0x1,%eax
0x8049031 <_start+49> mov $0x0,%ebx
0x8049036 <_start+54> int $0x80
native process 4492 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/m
/v/mvgracheva/work/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
(gdb)

```

Рис. 3.16: Режим псевдографики 2

Ввожу команду info breakpoints (рис. 4.9).

```
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb)
```

Рис. 3.17: Команда info breakpoints

Устанавливаю новую точку останова (рис. 4.9), (рис. 4.9).

```
(gdb) b *0x8049016
Breakpoint 2 at 0x8049016: file lab09-2.asm, line 14.
(gdb)
```

Рис. 3.18: Точка останова 1

```
B+> 0x8049000 <_start>    mov    $0x4,%eax
0x8049005 <_start+5>      mov    $0x1,%ebx
0x804900a <_start+10>     mov    $0x804a000,%ecx
0x804900f <_start+15>     mov    $0x8,%edx
0x8049014 <_start+20>     int    $0x80
b+  0x8049016 <_start+22>  mov    $0x4,%eax
0x804901b <_start+27>     mov    $0x1,%ebx
0x8049020 <_start+32>     mov    $0x804a008,%ecx
0x8049025 <_start+37>     mov    $0x7,%edx
0x804902a <_start+42>     int    $0x80
0x804902c <_start+44>     mov    $0x1,%eax
0x8049031 <_start+49>     mov    $0x0,%ebx
0x8049036 <_start+54>     int    $0x80
```

Рис. 3.19: Точка останова 2

Смотрю инфу про все установленные точки (рис. 4.9).

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2        breakpoint     keep y   0x08049016 lab09-2.asm:14
(gdb)
```

Рис. 3.20: Команда info breakpoints 2

Выполняю пять раз команду si (рис. 4.9), (рис. 4.9), (рис. 4.9), (рис. 4.9), (рис. 4.9).

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov     $0x4,%eax
> 0x8049005 <_start+5>   mov     $0x1,%ebx
0x804900a <_start+10>    mov     $0x804a000,%ecx
0x804900f <_start+15>    mov     $0x8,%edx
0x8049014 <_start+20>    int     $0x80
b+ 0x8049016 <_start+22>    mov     $0x4,%eax
0x804901b <_start+27>    mov     $0x1,%ebx
0x8049020 <_start+32>    mov     $0x804a008,%ecx
0x8049025 <_start+37>    mov     $0x7,%edx
0x804902a <_start+42>    int     $0x80
0x804902c <_start+44>    mov     $0x1,%eax
0x8049031 <_start+49>    mov     $0x0,%ebx
0x8049036 <_start+54>    int     $0x80

native process 4492 In: _start L10 PC: 0
(gdb) break *<0x804900a>
A syntax error in expression, near '<0x804900a>'.
(gdb) break *<_start+49>
A syntax error in expression, near '<_start+49>'.
(gdb) b *<0x8049016>
A syntax error in expression, near '<0x8049016>'.
(gdb) b *0x8049016
Breakpoint 2 at 0x8049016: file lab09-2.asm, line 14.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y  0x08049000 lab09-2.asm:9
       breakpoint already hit 1 time
2      breakpoint      keep y  0x08049016 lab09-2.asm:14
(gdb) si
(gdb)

```

Рис. 3.21: Выполнение команды si 1

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804900a 0x804900a <_start+10>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov    $0x4,%eax
0x8049005 <_start+5>    mov    $0x1,%ebx
> 0x804900a <_start+10>  mov    $0x804a000,%ecx
0x804900f <_start+15>    mov    $0x8,%edx
0x8049014 <_start+20>    int    $0x80
b+ 0x8049016 <_start+22>    mov    $0x4,%eax
0x804901b <_start+27>    mov    $0x1,%ebx
0x8049020 <_start+32>    mov    $0x804a008,%ecx
0x8049025 <_start+37>    mov    $0x7,%edx
0x804902a <_start+42>    int    $0x80
0x804902c <_start+44>    mov    $0x1,%eax
0x8049031 <_start+49>    mov    $0x0,%ebx
0x8049036 <_start+54>    int    $0x80

native process 4492 In: _start L11 PC: 0
A syntax error in expression, near '<0x804900a>'.
(gdb) break *<_start+49>
A syntax error in expression, near '<_start+49>'.
(gdb) b *<0x8049016>
A syntax error in expression, near '<0x8049016>'.
(gdb) b *0x8049016
Breakpoint 2 at 0x8049016: file lab09-2.asm, line 14.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y   0x08049000 lab09-2.asm:9
       breakpoint already hit 1 time
2      breakpoint      keep y   0x08049016 lab09-2.asm:14
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.22: Выполнение команды si 2

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804900f 0x804900f <_start+15>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov $0x4,%eax
0x8049005 <_start+5>    mov $0x1,%ebx
0x804900a <_start+10>   mov $0x804a000,%ecx
> 0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20>   int $0x80
b+ 0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27>   mov $0x1,%ebx
0x8049020 <_start+32>   mov $0x804a008,%ecx
0x8049025 <_start+37>   mov $0x7,%edx
0x804902a <_start+42>   int $0x80
0x804902c <_start+44>   mov $0x1,%eax
0x8049031 <_start+49>   mov $0x0,%ebx
0x8049036 <_start+54>   int $0x80

native process 4492 In: _start L12 PC: 0
(gdb) break *<_start+49>
A syntax error in expression, near '<_start+49>'.
(gdb) b *<0x8049016>
A syntax error in expression, near '<0x8049016>'.
(gdb) b *0x8049016
Breakpoint 2 at 0x8049016: file lab09-2.asm, line 14.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint      keep y  0x08049000 lab09-2.asm:9
       breakpoint already hit 1 time
2      breakpoint      keep y  0x08049016 lab09-2.asm:14
(gdb) si
(gdb) si
(gdb) si
(gdb) 

```

Рис. 3.23: Выполнение команды si 3

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049014 0x8049014 <_start+20>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov $0x4,%eax
0x8049005 <_start+5>    mov $0x1,%ebx
0x804900a <_start+10>   mov $0x804a000,%ecx
0x804900f <_start+15>   mov $0x8,%edx
> 0x8049014 <_start+20> int $0x80
b+ 0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27>   mov $0x1,%ebx
0x8049020 <_start+32>   mov $0x804a008,%ecx
0x8049025 <_start+37>   mov $0x7,%edx
0x804902a <_start+42>   int $0x80
0x804902c <_start+44>   mov $0x1,%eax
0x8049031 <_start+49>   mov $0x0,%ebx
0x8049036 <_start+54>   int $0x80

native process 4492 In: _start L13 PC: 0
A syntax error in expression, near '<_start+49>'.
(gdb) b *<0x8049016>
A syntax error in expression, near '<0x8049016>'.
(gdb) b *0x8049016
Breakpoint 2 at 0x8049016: file lab09-2.asm, line 14.
(gdb) i b
Num    Type           Disp Enb Address      What
1      breakpoint     keep y   0x08049000 lab09-2.asm:9
       breakpoint already hit 1 time
2      breakpoint     keep y   0x08049016 lab09-2.asm:14
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.24: Выполнение команды si 4

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov    $0x4,%eax
0x8049005 <_start+5>    mov    $0x1,%ebx
0x804900a <_start+10>   mov    $0x804a000,%ecx
0x804900f <_start+15>   mov    $0x8,%edx
0x8049014 <_start+20>   int    $0x80
B+> 0x8049016 <_start+22> mov    $0x4,%eax
0x804901b <_start+27>   mov    $0x1,%ebx
0x8049020 <_start+32>   mov    $0x804a008,%ecx
0x8049025 <_start+37>   mov    $0x7,%edx
0x804902a <_start+42>   int    $0x80
0x804902c <_start+44>   mov    $0x1,%eax
0x8049031 <_start+49>   mov    $0x0,%ebx
0x8049036 <_start+54>   int    $0x80

native process 4492 In: _start L14 PC: 0
(gdb) b *0x8049016
Breakpoint 2 at 0x8049016: file lab09-2.asm, line 14.
(gdb) i b
Num   Type           Disp Enb Address      What
1     breakpoint      keep y  0x08049000 lab09-2.asm:9
      breakpoint already hit 1 time
2     breakpoint      keep y  0x08049016 lab09-2.asm:14
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
Hello,
Breakpoint 2, _start () at lab09-2.asm:14
(gdb)

```

Рис. 3.25: Выполнение команды si 5

Смотрю содержимое регистров с помощью команды info registers (рис. 4.9).

```

native process 4492 In: _start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds/lo,   0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 3.26: Команда info registers

Смотрю значение переменной msg1 по имени(рис. 4.9).

```

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

```

Рис. 3.27: Значение переменной msg1 по имени

Смотрю значение переменной msg2 по адресу (рис. 4.9).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov $0x4,%eax
0x8049005 <_start+5>    mov $0x1,%ebx
0x804900a <_start+10>   mov $0x804a000,%ecx
0x804900f <_start+15>   mov $0x8,%edx
0x8049014 <_start+20>   int $0x80
B+> 0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27>   mov $0x1,%ebx
0x8049020 <_start+32>   mov $0x804a008,%ecx
0x8049025 <_start+37>   mov $0x7,%edx
0x804902a <_start+42>   int $0x80
0x804902c <_start+44>   mov $0x1,%eax
0x8049031 <_start+49>   mov $0x0,%ebx
0x8049036 <_start+54>   int $0x80

native process 4492 In: _start L14 PC: 0x8049016
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds16,    0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 3.28: Значение переменной msg2 по адресу

Заменяю первый (рис. 4.9) и второй символы в первой переменной (рис. 4.9).

```

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

```

Рис. 3.29: Замена символа 1


```

0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhlllo, "
(gdb) 

```

Рис. 3.30: Замена символа 2

Заменяю первый символ во второй переменной(рис. 4.9)

```

No symbol "shar" in current context.
(gdb) set {char}0x804a008='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "Lorld!\n\034"
(gdb) 

```

Рис. 3.31: Замена символа 3

С помощью команды set изменяю значение регистра ebx (рис. 4.9).

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

lab09-2.asm
B+  9  mov eax, 4
    10 mov ebx, 1
    11 mov ecx, msg1
    12 mov edx, msg1Len
    13 int 0x80
B+> 14  mov eax, 4
    15 mov ebx, 1
    16 mov ecx, msg2
    17 mov edx, msg2Len
    18 int 0x80
    19 mov eax, 1
    20 mov ebx, 0
    21 int 0x80

native process 8709 In: _start L14 PC: 0x8049016
(gdb) x/lb 0x804a008
0x804a008 <msg2>: "Lor1d!\n\034"
(gdb) set $ebx='2'
(gdb) p/s &ebx
No symbol "ebx" in current context.
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) p/s $eax
$3 = 8
(gdb) p/t $eax
$4 = 1000
(gdb) |
```

Рис. 3.32: Изменение значение регистра ebx

Завершаю выполнение программы с помощью команды si (рис. 4.9).

```

eax      0x4      4
eax      0x7      7
edx      0x7      7
edx      0x7      7
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804902a 0x804902a <_start+42>
eflags   0x202    [ IF ] c 4
cs       0x23     35
ss       0x2b     43

lab09-2.asm
B+ 9 mov eax, 4
    10 mov ebx, 1
    11 mov ecx, msg1
    12 mov edx, msg1Len
    13 int 0x80
B+ 14 mov eax, 4
    15 mov ebx, 1
    16 mov ecx, msg2
    17 mov edx, msg2Len
> 18 int 0x80
    19 int 0x80 1
> 20 mov eax, 1
    21 int 0x80

native process 8709 In: _start L18 PC: 0x804902a
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) p/s $eax
$3 = 8
(gdb) p/t $eax
$4 = 1000
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
World!
(gdb)

```

Рис. 3.33: Завершение программы

Скопировала файл lab8-2.asm, созданный при выполнении лабораторной работы No8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm(рис. 4.9).

```

mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-
pc/lab09/lab09-3.asm
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ ls
in_out.asm lab09-1.asm lab09-2 lab09-2.lst lab09-3.asm
lab09-1 lab09-1.o lab09-2.asm lab09-2.o
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $

```

Рис. 3.34: Копирование файла

Создаю исполняемый файл (рис. 4.9).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xfffffc310 0xfffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x8049000 <_start>    mov    $0x4,%eax
0x8049005 <_start+5>    mov    $0x1,%ebx
0x804900a <_start+10>   mov    $0x804a000,%ecx
0x804900f <_start+15>   mov    $0x8,%edx
0x8049014 <_start+20>   int    $0x80
B+ 0x8049016 <_start+22> mov    $0x4,%eax
0x804901b <_start+27>   mov    $0x1,%ebx
0x8049020 <_start+32>   mov    $0x804a008,%ecx
0x8049025 <_start+37>   mov    $0x7,%edx
0x804902a <_start+42>   int    $0x80
0x804902c <_start+44>   mov    $0x1,%eax
0x8049031 <_start+49>   mov    $0x0,%ebx
0x8049036 <_start+54>   int    $0x80

native process 4492 In: _start L14 PC: 0x8049016
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds1lo,   0x2b     43
es       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--qQuit
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 3.35: Создание исполняемого файла

Загружаю исполняемый файл в отладчик, указав аргументы(рис. 4.9).

```

mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргум
ент 3'
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 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-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 lab09-3...
(gdb)

```

Рис. 3.36: Загрузка в отладчик

Устанавливаю точку останова (рис. 4.9) и запускаю файл (рис. 4.9).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb)
```

Рис. 3.37: Точка останова

```
Starting program: /afs/.dk.sci.pfu.edu.ru/home/m/v/mvgracheva/work/arch-pc/lab09/lab09-3
3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)
```

Рис. 3.38: Запуск файла

Адрес вершины стека храниться в регистре есп и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы)(рис. 4.9).

```
(gdb) x/x $esp
0xfffffc2c0: 0x00000005
(gdb)
```

Рис. 3.39: Команда x/x

Смотрю остальные позиции стека (рис. 4.9), (рис. 4.9), (рис. 4.9), (рис. 4.9), (рис. 4.9), (рис. 4.9).

```
(gdb) x/s *(void**)(esp + 4)
0xfffffc55d: "/afs/.dk.sci.pfu.edu.ru/home/m/v/mvgracheva/work/arch-pc/lab09/lab09-3"
"
```

Рис. 3.40: Команда x/s

```
"
(gdb) x/s *(void**)(esp + 8)
0xfffffc5a4: "аргумент1"
```

Рис. 3.41: Команда x/s 2

```
(gdb) x/s *(void**)(esp + 12)
0xffffc5b6: "аргумент"
```

Рис. 3.42: Команда x/s 3

```
(gdb) x/s *(void**)(esp + 16)
0xffffc5c7: "2"
```

Рис. 3.43: Команда x/s 4

```
(gdb) x/s *(void**)(esp + 20)
0xffffc5c9: "аргумент 3"
```

Рис. 3.44: Команда x/s 5

```
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 3.45: Команда x/s 6

4 Самостоятельная работа

Задание 1

Копирую файл, созданный в ходе выполнения предыдущей лабораторной (рис. 4.9), вношу изменения в текст (рис. 4.9) и проверяю работу (рис. 4.9).

```
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/task.asm ~/work/arch-pc/lab09/task.asm
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ ls
in_out.asm lab09-1.asm lab09-2 lab09-2.lst lab09-3 lab09-3.lst task.asm
lab09-1 lab09-1.o lab09-2.asm lab09-2.o lab09-3.asm lab09-3.o
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ mc

mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf task.asm
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o task task.o
```

Рис. 4.1: Копирование файла, созданного в ходе выполнения предыдущей лабораторной

```

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

```

Рис. 4.2: Изменения в текст


```
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ ./task 1 2 3
функция: 30x-11
результат: 147
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $
```

Рис. 4.3: Проверка работы 4

Задание 2

Создаю файл (рис. 4.9), вношу текст (рис. 4.9), проверяю работу файла (рис. 4.9)

```
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ touch task2.asm
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $
```

Рис. 4.4: Создание файла task2

```

task2.asm          [-M--]  9 L:[  1+19
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 4.5: Введение текста

```

mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf task2.asm
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ nasm -f elf -g -l task2.lst task2.asm
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o task2 task2.o
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $ ./task2
Результат: 10
mvgracheva@dk8n52 ~/work/arch-pc/lab09 $

```

Рис. 4.6: Проверка работы 5

Включите режим псевдографики (зарание изменив синтаксис для удобства). С помощью команды `si` поэтапно проверяю код и вижу, что значение после сложение сохраняется не в тот регистр(рис. 4.9).

```

Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffc310 0xffffc310
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f4 0x80490f4 <_start+12>
eflags   0x206    [ PF IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x80490e8 <_start>    mov     ebx,0x3
0x80490ed <_start+5>    mov     eax,0x2
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
0x8049105 <_start+29>   call   0x804900f <sprint>
0x804910a <_start+34>   mov     eax,edi
0x804910c <_start+36>   call   0x8049086 <iprintLF>
0x8049111 <_start+41>   call   0x80490db <quit>
0x8049116             add     BYTE PTR [eax],al

native process 4217 In: _start L11 PC: 0x80490f4
(gdb) layout regs
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) yStarting program: /afs/.dk.sci.pfu.edu.ru/home/m
/v/mvgracheva/work/arch-pc/lab09/task2

Breakpoint 1, _start () at task2.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.7: Поиск ошибки

Меняю местами регистр + в дальнейших шагах меняем еах на ебх(рис. 4.9).

```

mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l task2.lst task2.asm
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o task2 task2.o
mvgracheva@dk4n65 ~/work/arch-pc/lab09 $ ./task2
Результат: 25

```

Рис. 4.8: Правки в исходной программе

Проверяю работу файла (рис. 4.9).

```

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 4.9: Проверка работы 6

5 Выводы

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

Список литературы