

Шаблон отчёта по лабораторной работе

9

Талебу Тенке Франк Устон НКАбд-05-23

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы :	6
2.1	Реализация циклов в NASM :	6
2.2	Отладка программ с помощью GDB :	10
2.3	Добавление точек останова :	16
2.4	Работа с данными программы в GDB :	17
2.5	Обработка аргументов командной строки в GDB :	20
2.6	Выводы по результатам выполнения заданий :	22
3	Задание для самостоятельной работы :	23
3.1	Выводы по результатам выполнения заданий :	25
4	Выводы, согласованные с целью работы :	26
	Список литературы	27

Список иллюстраций

2.1	Рисунок	6
2.2	Рисунок	7
2.3	Рисунок	8
2.4	Рисунок	9
2.5	Рисунок	10
2.6	Рисунок	11
2.7	Рисунок	12
2.8	Рисунок	13
2.9	Рисунок	13
2.10	Рисунок	14
2.11	Рисунок	14
2.12	Рисунок	15
2.13	Рисунок	16
2.14	Рисунок	16
2.15	Рисунок	19
2.16	Рисунок	19
2.17	Рисунок	21
2.18	Рисунок	22
3.1	Работа программы lab10-4.asm	24
3.2	код с ошибкой	25
3.3	отладка	25
3.4	код исправлен	25

Список таблиц

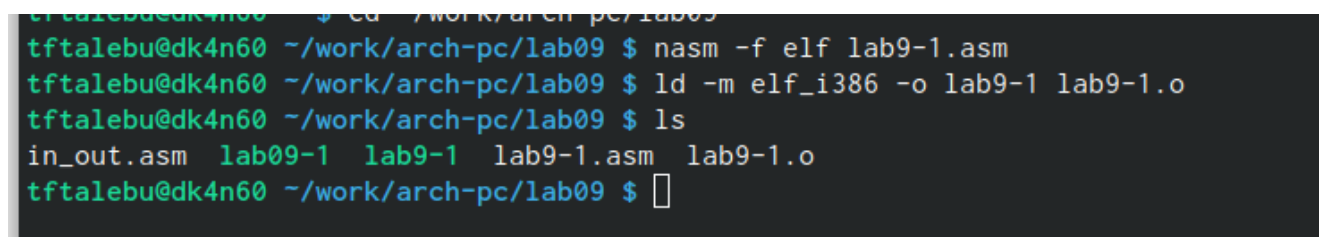
1 Цель работы

- В этой лабораторной работе мы научимся писать программы с использованием подпрограмм и познакомимся со способами отладки с использованием GDB и его основными функциями

2 Выполнение лабораторной работы :

2.1 Реализация циклов в NASM :

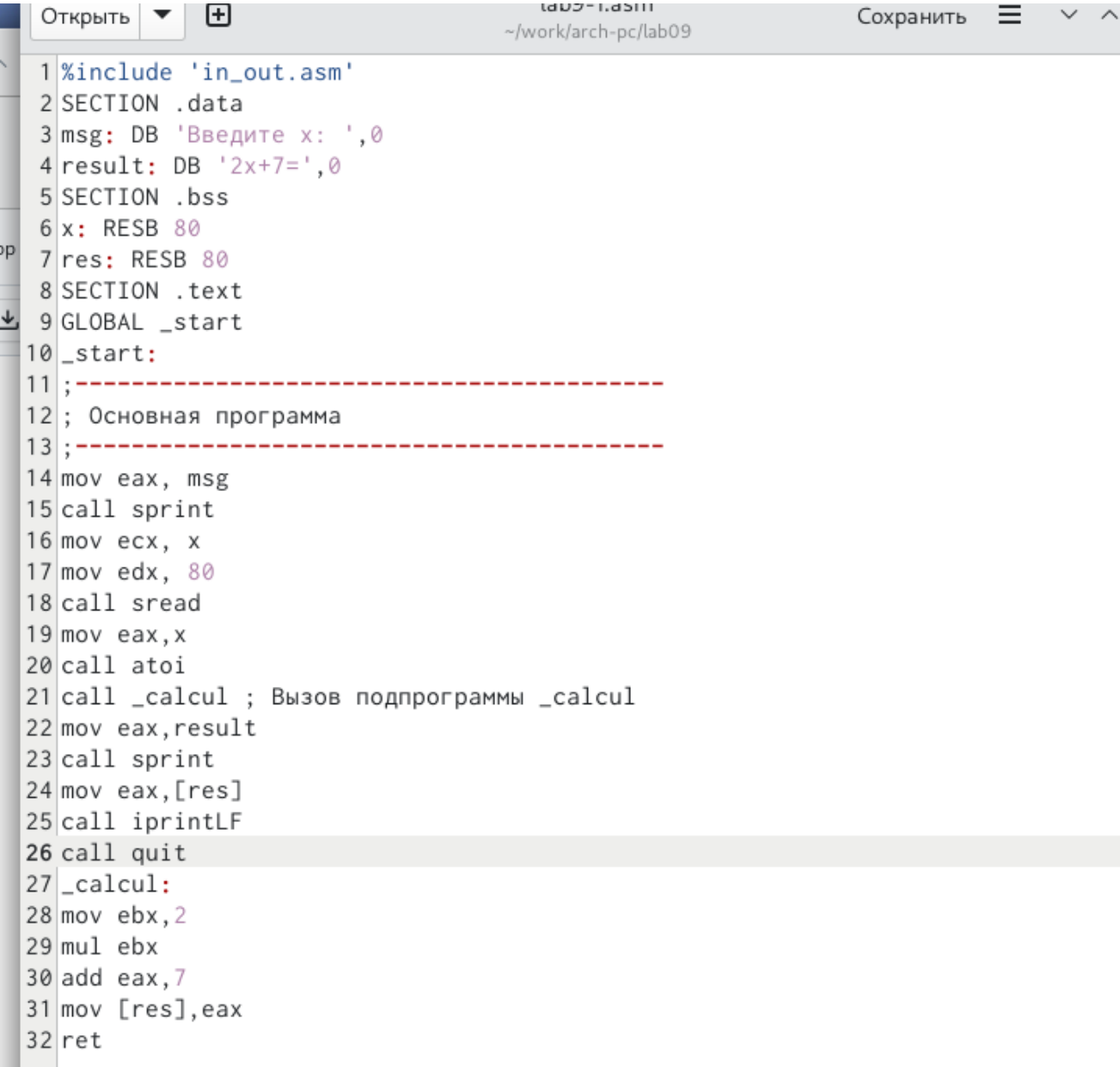
- Здесь мы начали с создания каталога для программы лабораторной работы No10, а затем переместились в десятый каталог лаборатории “~/work/arch-pc/lab09”, после чего мы создали файл “lab9-1.asm”. (рис. [2.1])



```
tftalebu@dk4n60 ~$ cd ~/work/arch-pc/lab09
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ls
in_out.asm lab09-1 lab9-1 lab9-1.asm lab9-1.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $
```

Рис. 2.1: Рисунок

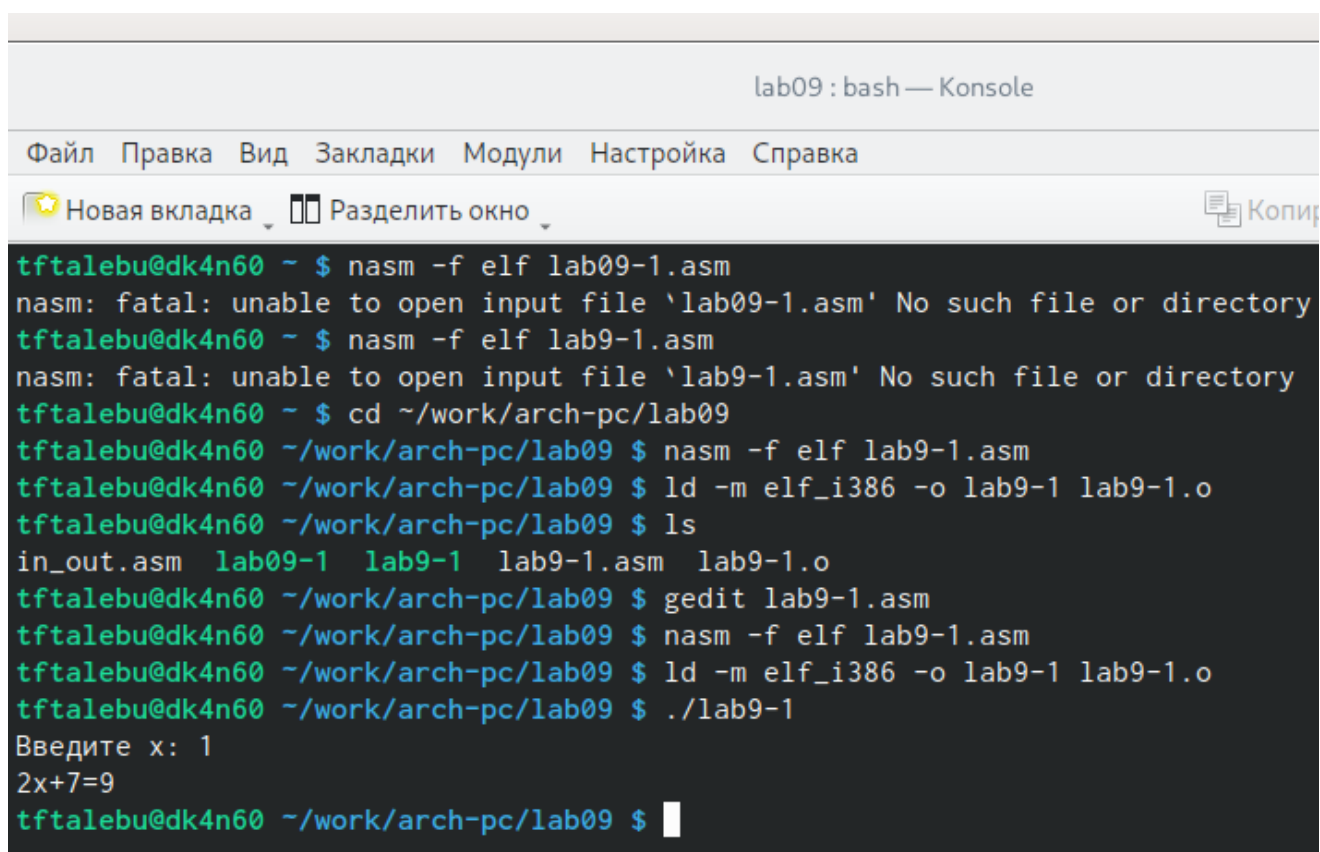
- Затем мы заполнили код нашей программы в файле lab9-1.asm.(рис. [2.2])



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ; -----
12 ; Основная программа
13 ; -----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call iprintLF
26 call quit
27 _calcul:
28 mov ebx, 2
29 mul ebx
30 add eax, 7
31 mov [res], eax
32 ret
```

Рис. 2.2: Ресунок

- После этого мы скомпилировали файл, создали исполняемый файл и проверили его работу.(рис. [2.3])



```
lab09 : bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать
tftalebu@dk4n60 ~ $ nasm -f elf lab09-1.asm
nasm: fatal: unable to open input file `lab09-1.asm' No such file or directory
tftalebu@dk4n60 ~ $ nasm -f elf lab9-1.asm
nasm: fatal: unable to open input file `lab9-1.asm' No such file or directory
tftalebu@dk4n60 ~ $ cd ~/work/arch-pc/lab09
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ls
in_out.asm lab09-1 lab9-1 lab9-1.asm lab9-1.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ gedit lab9-1.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ./lab9-1
Введите x: 1
2x+7=9
tftalebu@dk4n60 ~/work/arch-pc/lab09 $
```

Рис. 2.3: Ресунок

- Мы внесли изменения в наш код, чтобы она вычислила это уравнение $f(g(x))$, где x вводится с клавиатуры и $f(x) = 2x + 7$, $g(x) = 3x - 1$ а затем создали исполняемый файл.(рис. [2.4])(рис. [2.5])


```

5 result: DB 'f(x)=3(2x+7)=',0
6 SECTION .bss
7 x: RESB 80
8 res: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13
14 mov eax, msg
15 call sprint
16
17 mov ecx, x
18 mov edx, 80
19 call sread
20
21 mov eax, x
22 call atoi
23
24 call _calcul ; Вызов подпрограммы _calcul
25
26 mov eax, [res]
27
28 call _subcalcul
29
30 mov eax, result
31 call sprint
32 mov eax, [res]
33 call iprintLF
34
35 call quit
36
37 _calcul:
38 mov ebx, 2
39 mul ebx
40 add eax, 7
41 mov [res], eax
42 ret
43
44 _subcalcul:
45 mov ebx, 3
46 mul ebx
47 add eax, -1
48 mov [res], eax
49
50 ret

```

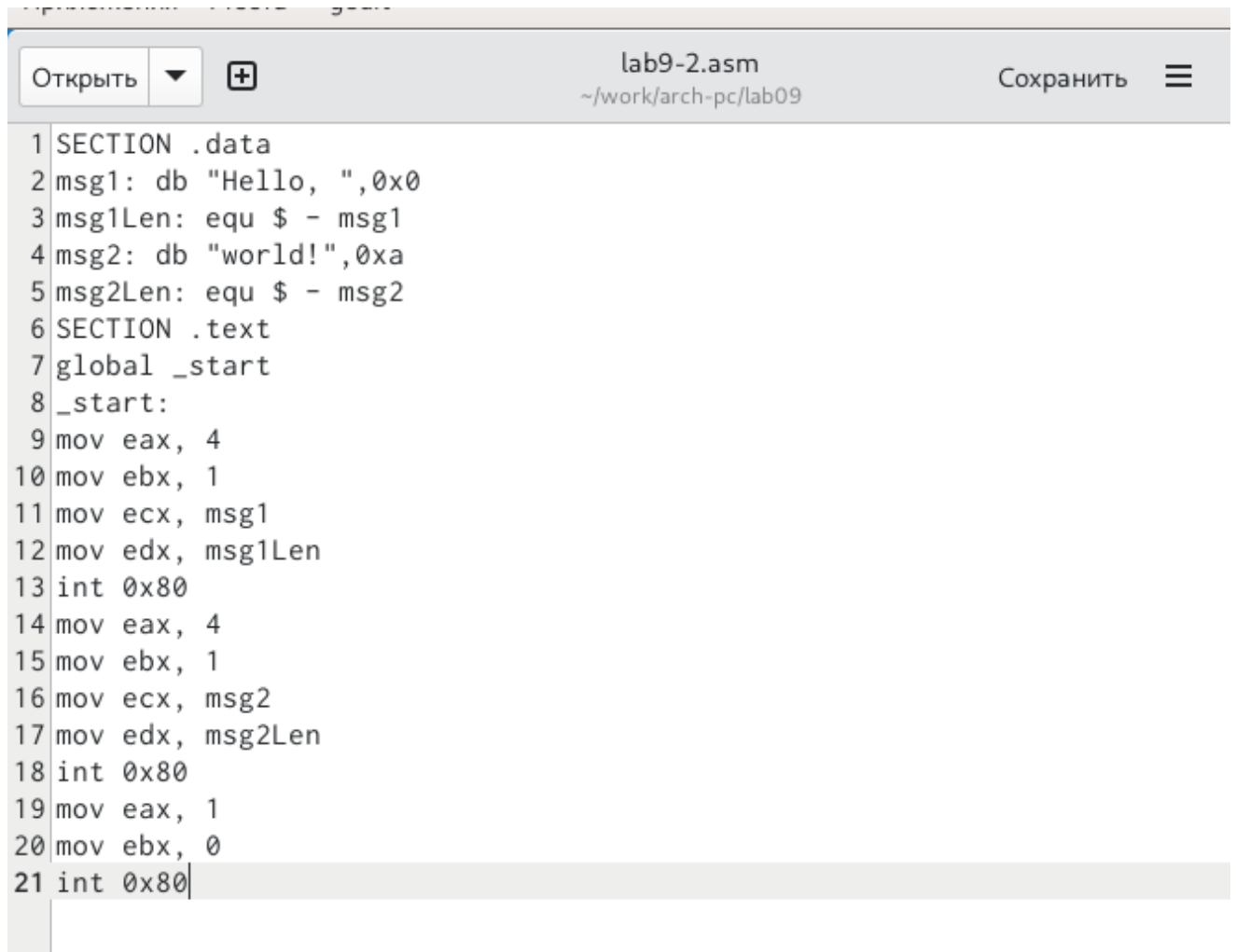
Рис. 2.4: Ресунок

```
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ./lab9-1
Введите x: 1
f(x)=3(2x+7)=26
tftalebu@dk4n60 ~/work/arch-pc/lab09 $
```

Рис. 2.5: Ресунок

2.2 Отладка программ с помощью GDB :

- На этом шаге мы создали файл lab9-2.asm с текстом программы из листинга 9.2.(рис. [2.6])



```
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6 SECTION .text
7 global _start
8 _start:
9 mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
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
```

Рис. 2.6: Рисунок

- После этого мы скомпилировали файл, создали исполняемый файл.Затем мы загрузили исполняемый файл в отладчик GDM. (рис. [2.7])

```

tftalebu@dk4n60 ~/work/arch-pc/lab09 $ touch lab9-2.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ gedit lab9-2.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ nasm -f elf lab9-2.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-2 lab9-2.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ gdb lab9-2
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 lab9-2...
(No debugging symbols found in lab9-2)
(gdb) █

```

Рис. 2.7: Ресунок

- затем мы проверили работу программы, запустив ее в оболочке GDB с помощью команды run.(рис. [2.8])

```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-2...
(No debugging symbols found in lab9-2)
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/f/tftalebu/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 6262) exited normally]
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/f/tftalebu/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 6317) exited normally]
(gdb) █

```

Рис. 2.8: Ресунок

- затем мы установили точку останова на метке `**_start**`, которая запускает выполнение любой программы на ассемблере, и запустили ее. (рис. [2.9])

```

tftalebu@dk4n60 ~ $ cd ~/work/arch-pc/lab09
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ ls
in_out.asm  lab09-1  lab9-1  lab9-1.asm  lab9-1.o
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ █

```

Рис. 2.9: Ресунок

- Затем мы просмотрели разобранный программный код, используя команду `disassemble`, начинающуюся с метки `**_start**`. (рис. [2.10])

```

(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. 2.10: Ресунок

- после этого мы переключились на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. [2.10])

Ресунок

Рис. 2.11: Ресунок

- Разница в синтаксисе между AT&T и INTEL заключается в том, что AT&T использует синтаксис `mov $0x4,%eax`, который популярен среди пользователей Linux, с другой стороны, INTEL использует синтаксис `mov eax,0x4`, который является популярен среди пользователей Windows.
- Затем мы включили псевдографический режим для более удобного анализа программы. (рис. [2.12])

```

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

```

native process 6889 In: _start

(gdb) layout regs

(gdb) info breakpoints

No breakpoints or watchpoints.

(gdb) info breakpoints

No breakpoints or watchpoints.

(gdb) break _start

Breakpoint 1 at 0x8049000

(gdb) run

Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/f/tftalebu/work/a

Breakpoint 1, 0x08049000 in _start ()

(gdb) disassemble _start

(gdb) layout asm

(gdb) layout regs

(gdb) █

Рис. 2.12: Ресунок

2.3 Добавление точек останова :

- Мы проверили точку останова с помощью информационных точек останова.(рис. [2.13])

```
(gdb) layout regs
(gdb) laout regs
Undefined command: "laout". Try "help".
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint      keep y  0x08049000  <_start>
        breakpoint already hit 1 time
(gdb) █
```

Рис. 2.13: Ресунок

- Мы определили адрес предпоследней инструкции (mov ebx,0x0) и установили точку останова.(рис. [2.14])

```
(gdb) b *0*8049031
Breakpoint 2 at 0x0
(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint      keep y  0x08049000  <_start>
        breakpoint already hit 1 time
2        breakpoint      keep y  0x00000000
(gdb) █
```

Рис. 2.14: Ресунок

2.4 Работа с данными программы в GDB :

- На этом шаге мы следовали 5 инструкциям, используя командный шаг i, и отслеживали изменение значений регистров, но перед этим мы проверили предыдущие значения регистров.(рис. [??]) (рис. [??])(рис. [2.15])

```
eax          0x0          0
ecx          0x0          0
edx          0x0          0
ebx          0x0          0
esp          0xffffc330    0xffffc330
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip          0x8049000    0x8049000 <_start>
eflags      0x202        [ IF ]
cs           0x23        35
ss           0x2b        43
ds           0x2b        43
es           0x2b        43
--Type <RET> for more, q to quit, c to continue without paging--
```

2.4 Работа с данными программы в GDB :

- На этом шаге мы следовали 5 инструкциям, используя командный шаг i, и отслеживали изменение значений регистров, но перед этим мы проверили предыдущие значения регистров.(рис. 2.15)(рис. ??)

```
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc5a0 0xffffc5a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
```

Рис. 2.15: Ресунок

```
(gdb) stepi
0x08049005 in _start ()
(gdb) stepi
0x0804900a in _start ()
(gdb) stepi
0x0804900f in _start ()
(gdb) stepi
0x08049014 in _start ()
(gdb) stepi
0x08049016 in _start ()
(gdb) 
```

Рисунок

Рис. 2.15: Рисунок

- После проверки мы видим, что регистры : eax,ecx,edx,ebx,esp изменили свое значение.

- Мы рассмотрели значение переменной msg1 по имени, используя команду

```
--Type <RET> for more, q to quit, c to continue without pag
(gdb) x/1sb &msg1
0x804a000:      "Hello, "
```

x/1sb.(рис. [??])

- Здесь мы рассмотрели значение переменной msg2, используя адрес.(рис.

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

[??])

- Здесь мы изменили первую букву переменной msg1, которая имеет тип

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

char.(рис. [??])

- После этого мы изменили первую букву переменной msg2.(рис. [2.16])

```
(gdb) set {char}&msg2='F'
(gdb) x/1sb &msg2
0x804a008:      "Forld!\n"
(gdb)
```

Рис. 2.16: Рисунок

- Затем мы выводим значение регистра edx в различных форматах (шестна-

```
(gdb) p/x $edx
$1 = 0x0
(gdb) p/s $edx
$2 = 0
(gdb) p/t $edx
$3 = 0
(gdb) p/s $edx
$4 = 0
(gdb) 
```

дцатеричном, двоичном и символьном).(рис. [??])

- Используя команду set, мы изменили значение регистра ebx, когда раз, вве-

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

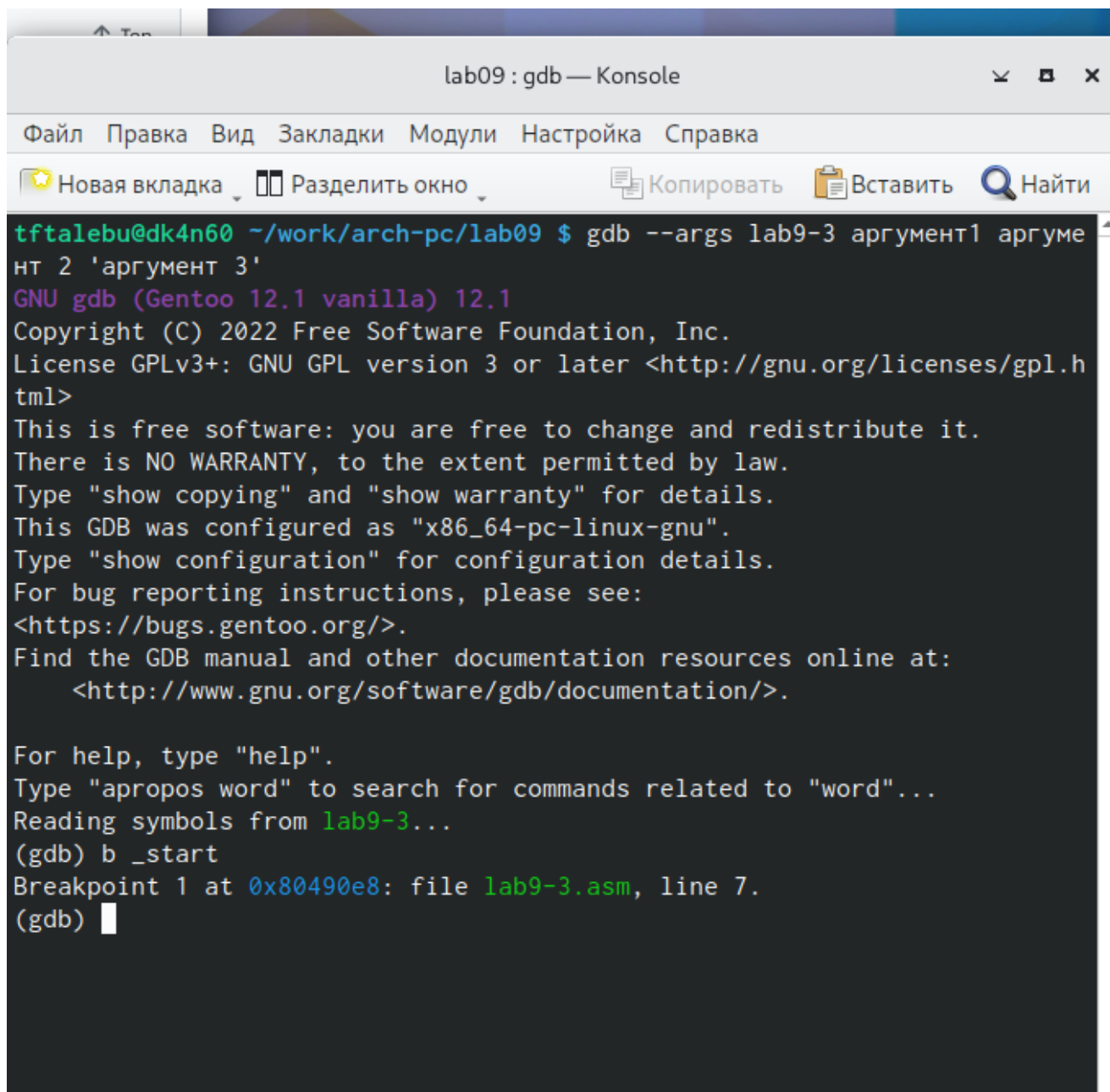
дя '2', а в другой раз, введя 2.(рис. [??])

- но когда мы напечатали значение регистра, мы получили значение 50 и это потому, что машина интерпретировала введенное значение как символ, и в таблице ASCII символ '2' имеет значение 50 в десятичной системе, но когда мы ввели значение 2 машина интерпретировала 2 как число в десятичной системе.
- Наконец, мы завершили программу с помощью stepi и вышли из GDB с помощью команды quit.

2.5 Обработка аргументов командной строки в GDB :

- На этом этапе мы скопировали файл lab9-2.asm, созданный при выполнении лабораторной работы No9 с программой, отображающей аргументы

командной строки на экране (листинг 9.2), в файл с именем lab 10-3.asm, а затем мы скомпилировали этот файл и установил точку останова в `**_start**` и запустил отладчик.(рис. [2.17])



```
lab09: gdb — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
tftalebu@dk4n60 ~/work/arch-pc/lab09 $ gdb --args lab9-3 аргумент1 аргуме
нт 2 'аргумент 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.h
tml>
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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 7.
(gdb) █
```

Рис. 2.17: Ресунок

- Затем мы посмотрели на остальные позиции стека – адрес в памяти, где

находится имя программы, находится в $[esp + 4]$, адрес первого аргумента хранится в $[esp + 8]$, в $[esp + 12]$. (рис. [2.18])

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/t/f/tftalebu/work/arch-pc/
lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab9-3.asm:7
7      pop ecx
(gdb) x/x $esp
0xfffffc310:      0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc5ad:      "/afs/.dk.sci.pfu.edu.ru/home/t/f/tftalebu/work/arch-pc/1
ab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc5f1:      "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc603:      "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc614:      "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc616:      "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) □
```

Рис. 2.18: Ресунок

- Шаг изменения адреса равен 4, потому что размер регистра esp равен 32бита = 4 байтам, а количество памяти равно количеству аргументов плюс имя программы, поэтому мы получили 5 шагов с 4 байтами для каждого шага.

2.6 Выводы по результатам выполнения заданий :

- В этой части работы мы узнали, как работать с отладчиком GDB, и получили более близкое представление о том, как работают подпрограммы.

3 Задание для самостоятельной работы :

- Преобразуйте программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. [3.1] [??])

2.4 Работа с данными программы в GDB :

- На этом шаге мы следовали 5 инструкциям, используя командный шаг i, и отслеживали изменение значений регистров, но перед этим мы проверили предыдущие значения регистров.(рис. 2.15)(рис. ??)

```
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc5a0 0xffffc5a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
```

Рис. 2.15: Ресунок

```
(gdb) stepi
0x08049005 in _start ()
(gdb) stepi
0x0804900a in _start ()
(gdb) stepi
0x0804900f in _start ()
(gdb) stepi
0x08049014 in _start ()
(gdb) stepi
0x08049016 in _start ()
(gdb) 
```

Рис. 3.1: Работа программы lab10-4.asm

7. В листинге приведена программа вычисления выражения $(3+2)*4+5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.(рис. [3.2] [3.3] [3.4] [??])


```

Invalid number: 004a000 .
(gdb) x/1sb 0x804a008
0x804a008:      "world!\n"
(gdb)

```

Рис. 3.2: код с ошибкой

```

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

```

Рис. 3.3: отладка

Отметим, что перепутан порядок аргументов у инструкции `add` и что по окончании работы в `edi` отправляется `ebx` вместо `eax`

```

(gdb) set {char}&msg2='F'
(gdb) x/1sb &msg2
0x804a008:      "Forld!\n"
(gdb)

```

Рис. 3.4: код исправлен

3.1 Выводы по результатам выполнения заданий :

- В этой части мы узнали, как превратить программу в подпрограмму, но у нас возникла проблема с подпрограммой `atoi` , поэтому мы не смогли вычислить результат.

4 Выводы, согласованные с целью работы :

- В этой лабораторной работе мы научимся писать программы с использованием подпрограмм и познакомимся со способами отладки с использованием GDB и его основными функциями.

Список литературы