

Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Маслова Анна Павловна

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Выполнение заданий для самостоятельной работы	24
4	Выводы	39
	Список литературы	40

Список иллюстраций

2.1	Создание каталога lab09 и файла lab09-1.asm	7
2.2	Текст файла lab09-1.asm	8
2.3	Создание и запуск исполняемого файла lab09-1	9
2.4	Изменённый текст файла lab09-1.asm	10
2.5	Изменённый текст файла lab09-1.asm	10
2.6	Создание и запуск изменённого исполняемого файла lab09-1 . . .	11
2.7	Текст файла lab09-2.asm	12
2.8	Создание исполняемого файла lab09-2 и файла листинга lab09-2.lst	12
2.9	Отладчик gdb	13
2.10	Запуск программы с помощью команды run	13
2.11	Установка брейкпоинта на метку _start	13
2.12	Дисассимилированный код программы lab09-2	14
2.13	Синтаксис Intel	14
2.14	Режим псевдографики	15
2.15	Проверка точек останова	16
2.16	Установка точки останова по адресу	16
2.17	Инструкция stepi	17
2.18	Инструкция stepi	17
2.19	Инструкция stepi	18
2.20	Инструкция stepi	18
2.21	Инструкция stepi	19
2.22	Значение переменной msg1 по имени	19
2.23	Значение переменной msg2 по адресу	19
2.24	Изменение символов переменных msg1 и msg2	20
2.25	Вывод значения регистра edx	20
2.26	Изменение значения регистра edx	20
2.27	Завершение выполнения программы	21
2.28	Создание исполняемого файла lab09-3	21
2.29	Загрузка программы lab09-3 в gdb	22
2.30	Установка точки останова перед первой инструкцией и запуск программы	22
2.31	Содержимое регистра esp	22
2.32	Содержимое стека	23
3.1	Текст файла func1.asm	26
3.2	Создание и запуск исполняемого файла func1	26
3.3	Текст файла func2.asm	27

3.4	Создание и запуск исполняемого файла func2	27
3.5	Загрузка файла func2 в gdb	28
3.6	Дисассимилированный код программы func2	28
3.7	Режим псевдографики для func2	29
3.8	Просмотр значений регистров в func2	30
3.9	Точка останова на инструкции add	31
3.10	Посмотр значений регистров	32
3.11	Команда si	33
3.12	Изменение значения регистра eax с помощью команды set	34
3.13	Переход на следующий шаг	35
3.14	Результат вычисления произведения	36
3.15	Завершение выполнения программы с помощью команды c . . .	37
3.16	Исправленный текст программы в файле func2	38
3.17	Проверка работы программы func2	38

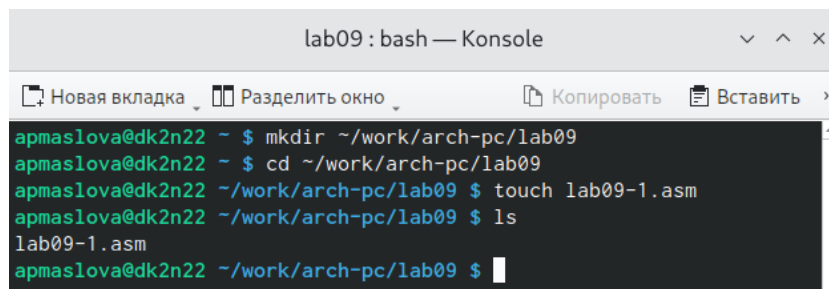
Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

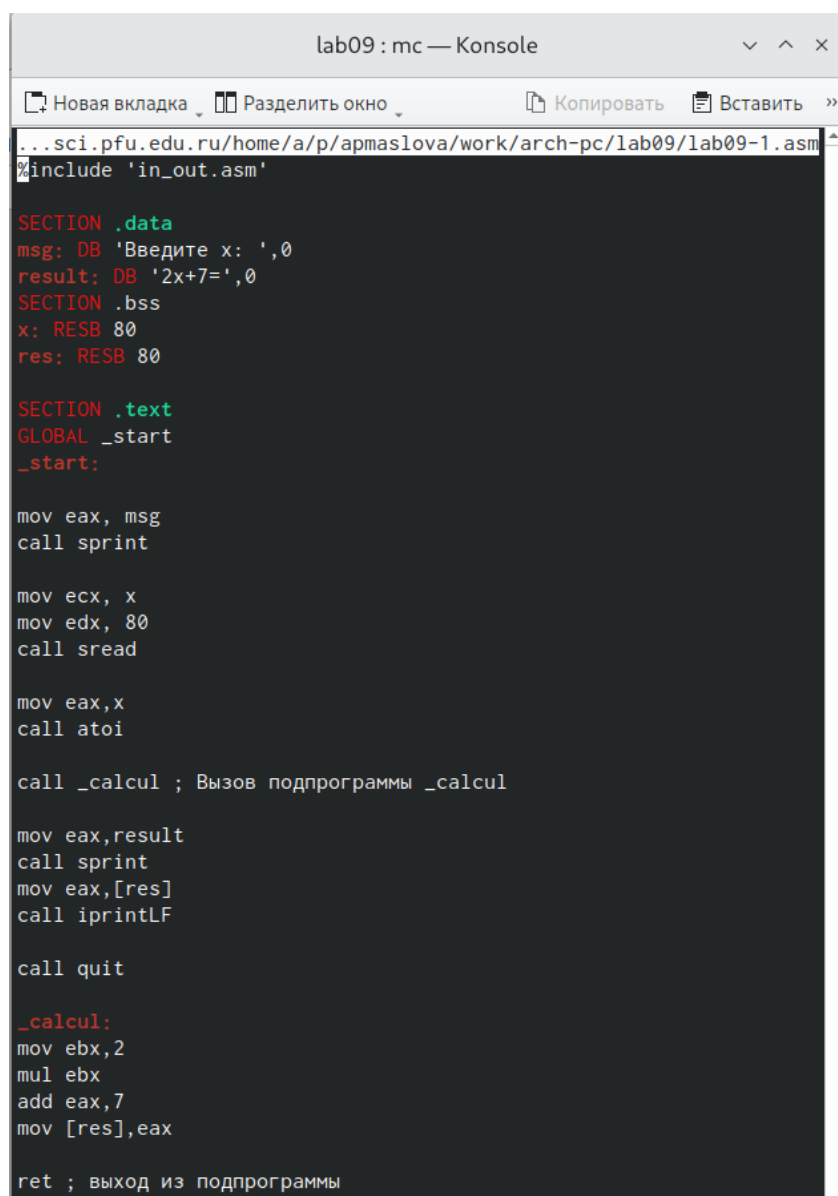
Создадим каталог для выполнения лабораторной работы №9, перейдём в него и создадим файл `lab09-1.asm`. (рис. 2.1)



```
lab09 : bash — Konsole
Новая вкладка Разделить окно Копировать Вставить »
армаслова@dk2n22 ~ $ mkdir ~/work/arch-pc/lab09
армаслова@dk2n22 ~ $ cd ~/work/arch-pc/lab09
армаслова@dk2n22 ~/work/arch-pc/lab09 $ touch lab09-1.asm
армаслова@dk2n22 ~/work/arch-pc/lab09 $ ls
lab09-1.asm
армаслова@dk2n22 ~/work/arch-pc/lab09 $
```

Рис. 2.1: Создание каталога `lab09` и файла `lab09-1.asm`

Рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$. Введём в файл `lab09-1.asm` текст этой программы. (рис. 2.1)



```
lab09: mc — Konsole
...sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab09/lab09-1.asm
#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 ; Вызов подпрограммы _calcul

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit

_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax

ret ; выход из подпрограммы
```

Рис. 2.2: Текст файла lab09-1.asm

Создадим исполняемый файл и проверим его работу. (рис. 2.3)


```

apmaslova@dk2n22 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ls
in_out.asm  lab09-1.asm  lab09-1.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 l
ab09-1.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ls
in_out.asm  lab09-1  lab09-1.asm  lab09-1.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
2x+7=9
apmaslova@dk2n22 ~/work/arch-pc/lab09 $

```

Рис. 2.3: Создание и запуск исполняемого файла lab09-1

Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Результат возвращается в основную программу для вывода результата на экран. (рис. 2.4 , рис. 2.5)

```
lab09: mc — Konsole
...sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab09/lab09-1.asm
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+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 ; Вызов подпрограммы _calcul

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit
```

Рис. 2.4: Изменённый текст файла lab09-1.asm

```
_calcul:
; 2x+7
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы f(g(x))

_subcalcul:
; 3x-1
mov ebx, 3
mul ebx
dec eax ; eax=3x-1
ret ; выход из подпрограммы g(x)
```

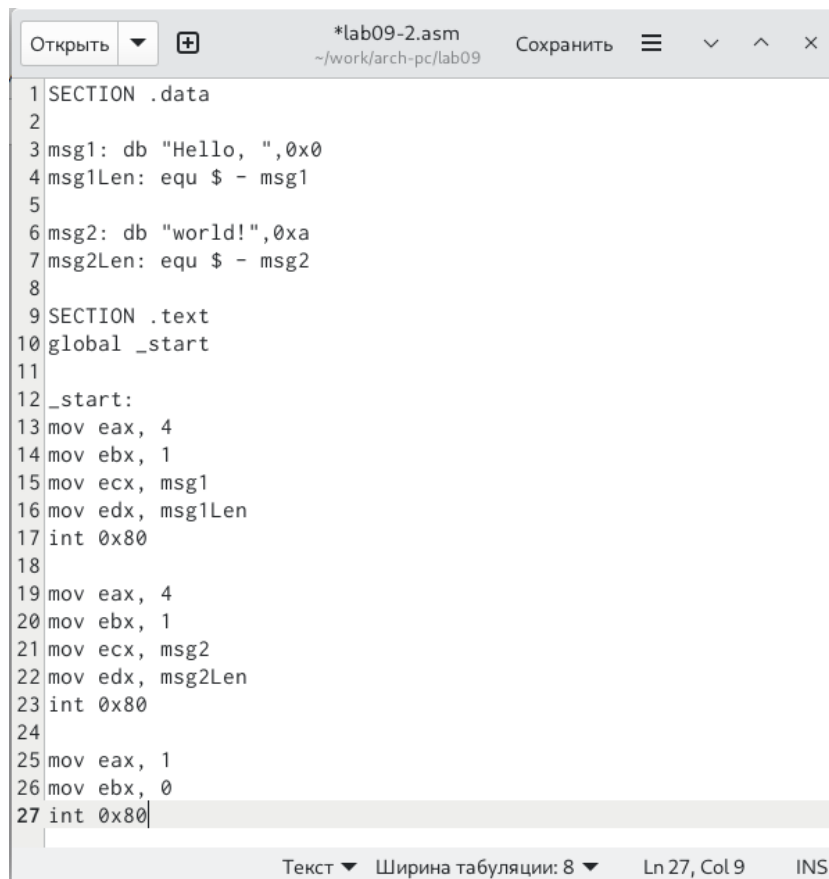
Рис. 2.5: Изменённый текст файла lab09-1.asm

Создадим исполняемый файл и проверим его работу. (рис. 2.6)

```
армаслова@dk2n22 ~/work/arch-pc/lab09 $ gedit lab09-1.asm
армаслова@dk2n22 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
армаслова@dk2n22 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
армаслова@dk2n22 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
2(3x-1)+7=11
армаслова@dk2n22 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2(3x-1)+7=35
армаслова@dk2n22 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 9
2(3x-1)+7=59
армаслова@dk2n22 ~/work/arch-pc/lab09 $
```

Рис. 2.6: Создание и запуск изменённого исполняемого файла lab09-1

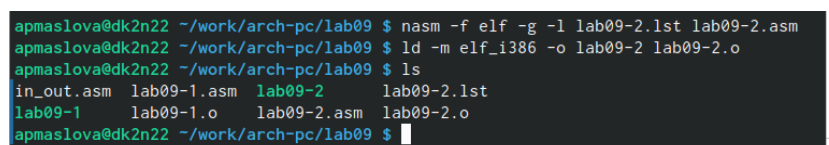
Создадим файл lab09-2.asm с текстом программы печати сообщения *Hello world!* (рис. 2.7)



```
1 SECTION .data
2
3 msg1: db "Hello, ",0x0
4 msg1Len: equ $ - msg1
5
6 msg2: db "world!",0xa
7 msg2Len: equ $ - msg2
8
9 SECTION .text
10 global _start
11
12 _start:
13 mov eax, 4
14 mov ebx, 1
15 mov ecx, msg1
16 mov edx, msg1Len
17 int 0x80
18
19 mov eax, 4
20 mov ebx, 1
21 mov ecx, msg2
22 mov edx, msg2Len
23 int 0x80
24
25 mov eax, 1
26 mov ebx, 0
27 int 0x80
```

Рис. 2.7: Текст файла lab09-2.asm

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом `-g`. (рис. 2.8)



```
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ls
in_out.asm  lab09-1.asm  lab09-2      lab09-2.lst
lab09-1     lab09-1.o    lab09-2.asm  lab09-2.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $
```

Рис. 2.8: Создание исполняемого файла lab09-2 и файла листинга lab09-2.lst

Загрузим исполняемый файл в отладчик `gdb`. (рис. 2.9)

```

apmaslova@dk2n22 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 13.2 vanilla) 13.2
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-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. 2.9: Отладчик gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды `run` (рис. 2.10).

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/
lab09/lab09-2
Hello, world!
[Inferior 1 (process 4919) exited normally]
(gdb)

```

Рис. 2.10: Запуск программы с помощью команды `run`

Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её. (рис. 2.11)

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 13.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/
lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:13
13      mov eax, 4
(gdb)

```

Рис. 2.11: Установка брейкпоинта на метку `_start`

Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`. (рис. 2.12)

```
(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. 2.12: Дисассимилированный код программы lab09-2

Переключимся на отображение команд с Intel'овским синтаксисом (рис. 2.13)

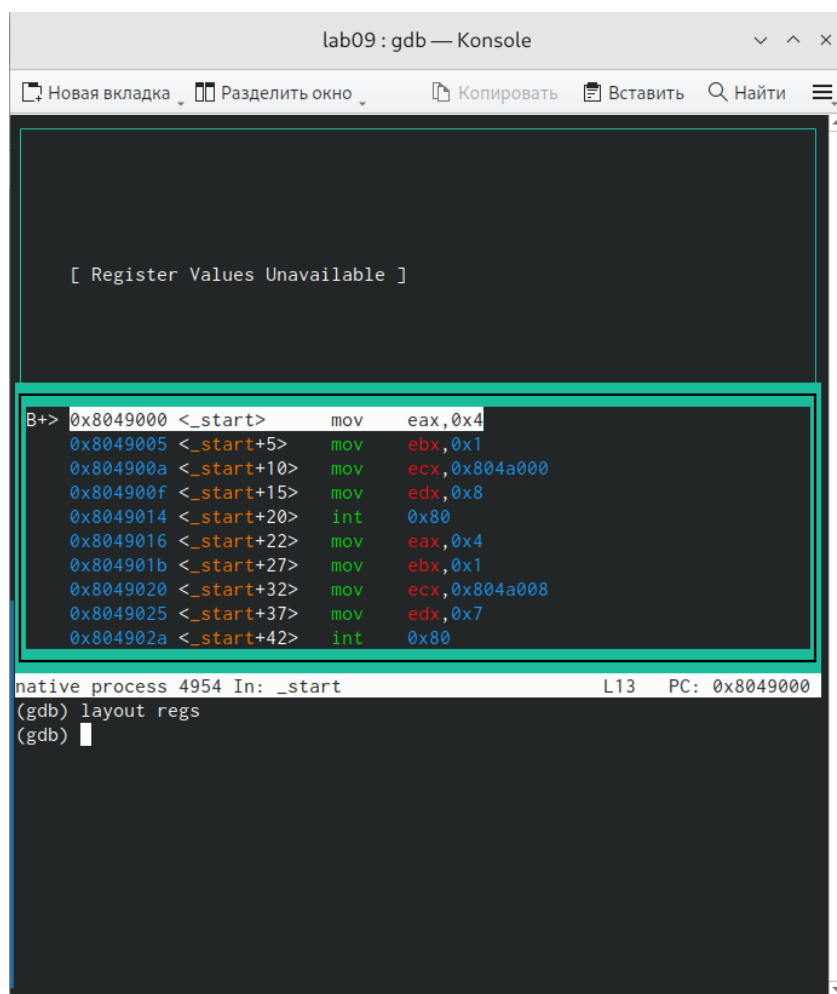
```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     eax,0x4
    0x08049005 <+5>:    mov     ebx,0x1
    0x0804900a <+10>:   mov     ecx,0x804a000
    0x0804900f <+15>:   mov     edx,0x8
    0x08049014 <+20>:   int     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. 2.13: Синтаксис Intel

Как мы видим, в синтаксисе Intel после команды выводится регистр, куда помещается значение, а затем адрес помещаемого значения или число. А в синтаксисе ATТ сначала выводится ссылка на заносимое в регистр значение (для этого перед значением есть символ `$`), а уже после неё регистр с символом `%`.

Включим режим псевдографики для более удобного анализа программы (рис.

2.14)



The screenshot shows a GDB console window titled "lab09 : gdb — Konsole". The window has a menu bar with options: "Новая вкладка", "Разделить окно", "Копировать", "Вставить", "Найти", and a hamburger menu icon. The main area displays the text "[Register Values Unavailable]". Below this, a list of assembly instructions is shown, each with its address, a comment, and the instruction itself. The instructions are:
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
At the bottom, the status bar shows "native process 4954 In: _start L13 PC: 0x8049000". Below the status bar, the command "(gdb) layout regs" is entered, and the prompt "(gdb) " is visible.

```
lab09 : gdb — Konsole

[ 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
      0x804901b <_start+27> mov    ebx,0x1
      0x8049020 <_start+32> mov    ecx,0x804a008
      0x8049025 <_start+37> mov    edx,0x7
      0x804902a <_start+42> int     0x80

native process 4954 In: _start L13 PC: 0x8049000
(gdb) layout regs
(gdb) 
```

Рис. 2.14: Режим псевдографики

На предыдущих шагах была установлена точка останова по имени метки `_start`. Проверим это с помощью команды `info breakpoints` (рис. 2.15)

The screenshot shows a GDB console window titled "lab09: gdb — Konsole". The main display area shows assembly instructions with addresses 0x80491ce to 0x80491d6, all being "add BYTE PTR [eax], al". Below this, the status bar indicates "native process 4954 In: _start L13 PC: 0x8049000". The command history shows "(gdb) layout regs" and "(gdb) info breakpoints". A table of breakpoints is displayed:

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep y		0x08049000	lab09-2.asm:13

Below the table, it says "breakpoint already hit 1 time". The prompt "(gdb) " is visible at the bottom.

Рис. 2.15: Проверка точек останова

Установим еще одну точку останова по адресу инструкции `mov ebx, 0x0`. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. В нашем случае адрес следующий: 0x8049031. После этого посмотрим информацию о всех точках останова (рис. 2.16).

The screenshot shows a GDB console window with assembly instructions. The instruction at address 0x8049031, `mov ebx, 0x0`, is highlighted with a red box. Below the assembly list, the status bar shows "native process 4954 In: _start L13 PC: 0x8049000". The command history shows "(gdb) disassemble _start", "(gdb) b *0x8049031", and "Breakpoint 2 at 0x8049031: file lab09-2.asm, line 26.". The command "(gdb) i b" has been executed, resulting in the following breakpoint table:

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep y		0x08049000	lab09-2.asm:13
2	breakpoint	keep y		0x08049031	lab09-2.asm:26

The prompt "(gdb) " is visible at the bottom.

Рис. 2.16: Установка точки останова по адресу

Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследим за изменением значений регистров (рис. 2.17 - 2.21).

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0

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

native process 4954 In: _start          L14  PC: 0x8049005
(gdb) si
(gdb)

```

Рис. 2.17: Инструкция `stepi`

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0

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

native process 4954 In: _start          L15  PC: 0x804900a
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.18: Инструкция `stepi`

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0
esi      0x0      0

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

native process 4954 In: _start L16 PC: 0x804900f
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.19: Инструкция stepi

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0
esi      0x0      0

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

native process 4954 In: _start L17 PC: 0x8049014
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.20: Инструкция stepi

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc220 0xffffc220
ebp      0x0      0
esi      0x0      0

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

native process 4954 In: _start L19 PC: 0x8049016
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.21: Инструкция stepi

Как мы видим на верхней панели, изменились значения регистров eax, ecx, edx и ebx. Остальные регистры остаются без изменений.

Теперь посмотрим значение переменной msg1 по имени (рис. 2.22).

```

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

```

Рис. 2.22: Значение переменной msg1 по имени

Вывелась строка "Hello,".

Теперь посмотрим значение переменной msg2 по адресу. Адрес переменной определим по дизассемблированной инструкции. Посмотрим инструкцию mov ecx, msg2 которая записывает в регистр ecx адрес переменной msg2 (рис. 2.23).

```

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

```

Рис. 2.23: Значение переменной msg2 по адресу

Изменим первый символ переменной `msg1` и первый символ переменной `msg2`(рис. 2.24).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='W'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "World!\n\034"
(gdb) █
```

Рис. 2.24: Изменение символов переменных `msg1` и `msg2`

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`(рис. 2.25).

```
(gdb) p/x $edx
$4 = 0x8
(gdb) p/t $edx
$5 = 1000
(gdb) p/s $edx
$6 = 8
(gdb) █
```

Рис. 2.25: Вывод значения регистра `edx`

Теперь с помощью команды `set` изменим значение регистра `ebx` (рис. 2.26).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb) █
```

Рис. 2.26: Изменение значения регистра `ebx`

В первом случае мы занесли в регистр `ebx` символ '2', поэтому после запроса `p/s` на вывод значения регистра на экране мы увидели код символа "2", а именно - 50. А в случае, когда в регистр изначально было занесено число 2, а не символ, команда `p/s $ebx` вывела значение 2.

Завершим выполнение программы с помощью команды `continue` (сокращенно `c`) (рис. 2.27).

```
$9 = 2
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:26
(gdb) █
```

Рис. 2.27: Завершение выполнения программы

И после этого с помощью команды `quit` вышли из отладчика `gdb`.

Скопируем файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой, выводящей на экран аргументы командной строки, в файл с именем `lab09-3.asm` и создадим исполняемый файл с ключом `-g` и с файлом листинга (рис. 2.28).

```
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm
~/work/arch-pc/lab09/lab09-3.asm
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ls
in_out.asm  lab09-1.o  lab09-2.lst  lab09-3.asm
lab09-1     lab09-2   lab09-2.o   lab09-3.lst
lab09-1.asm lab09-2.asm lab09-3     lab09-3.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ █
```

Рис. 2.28: Создание исполняемого файла `lab09-3`

Для загрузки в `gdb` программы с аргументами необходимо использовать ключ `--args`. Загрузим исполняемый файл в отладчик, указав аргументы (рис. 2.29).

```

apmaslova@dk2n22 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 13.2 vanilla) 13.2
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-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. 2.29: Загрузка программы lab09-3 в gdb

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

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ;извлекаем из стека в 'ecx' кол-во элементов
(gdb)

```

Рис. 2.30: Установка точки останова перед первой инструкцией и запуск программы

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

```

(gdb) x/x $esp
0xfffffc1e0: 0x00000005
(gdb)

```

Рис. 2.31: Содержимое регистра esp

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и аргумент 3.

Посмотрим остальные позиции стека (рис. 2.32).

```
(gdb) x/s *(void**)(esp + 4)
0xffffc472:  "/afs/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/1
ab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffc4b8:  "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffc4ca:  "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffc4db:  "2"
(gdb) x/s *(void**)(esp + 20)
0xffffc4dd:  "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.32: Содержимое стека

По адресу [esp+4] располагается адрес в памяти, где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. Как мы видим, шаг изменения равен 4. Шаг имеет такое значение, потому что при добавлении значения каждого аргумента в стек значение регистра esp увеличивается на 4.

3 Выполнение заданий для самостоятельной работы

Задание №1

Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.

В лабораторной работе №8 мы реализовывали вычисление в цикле следующей функции:

$$f(x) = 6x + 13$$

Теперь для вычисления значения этой функции в цикле мы будем вызывать вспомогательную подпрограмму. Создадим файл `func1.asm` и впишем в него текст программы из листинга 9.4 (рис. 3.1).

Листинг 9.4. Программа вычисления суммы значений функции с помощью подпрограммы

```
%include 'in_out.asm'

section .data
msg db "Результат: ",0

section .text
```



```

global _start
_start:

pop ecx
pop edx
sub ecx,1
mov esi,0 ;используем esi для хранения промежуточных сумм f(x)

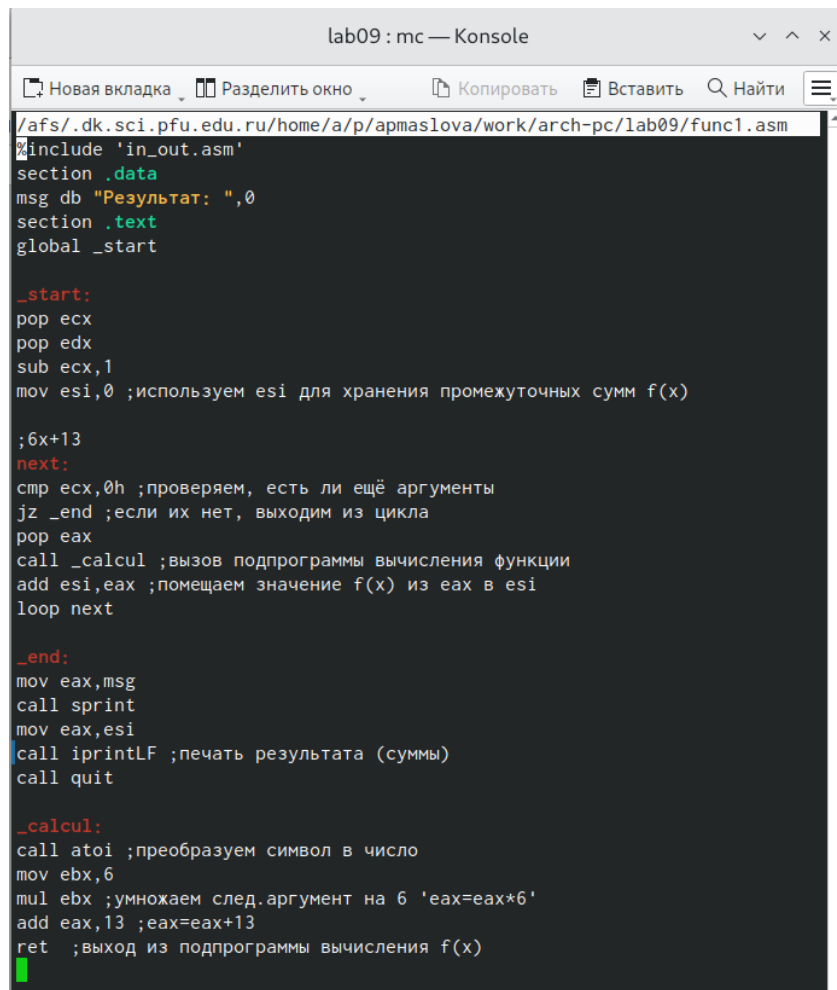
;6x+13
next:
cmp ecx,0h ;проверяем, есть ли ещё аргументы
jz _end ;если их нет, выходим из цикла
pop eax
call _calcul ;вызов подпрограммы вычисления функции
add esi,eax ;помещаем значение f(x) из eax в esi
loop next

_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF ;печать результата (суммы)
call quit

_calcul:
call atoi ;преобразуем символ в число
mov ebx,6
mul ebx; умножаем след.аргумент на 6 'eax=eax*6'
add eax,13 ;eax=eax+13

```

ret ;выход из подпрограммы вычисления $f(x)$



```
lab09: mc — Konsole
/afs/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab09/func1.asm
%include 'in_out.asm'
section .data
msg db "Результат: ",0
section .text
global _start

_start:
pop ecx
pop edx
sub ecx,1
mov esi,0 ;используем esi для хранения промежуточных сумм f(x)

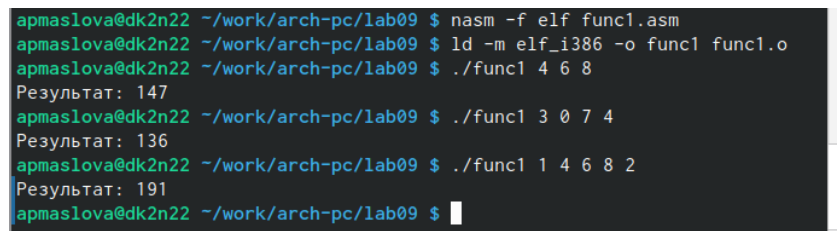
;6x+13
next:
cmp ecx,0h ;проверяем, есть ли ещё аргументы
jz _end ;если их нет, выходим из цикла
pop eax
call _calcul ;вызов подпрограммы вычисления функции
add esi,eax ;помещаем значение f(x) из eax в esi
loop next

_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF ;печать результата (суммы)
call quit

_calcul:
call atoi ;преобразуем символ в число
mov ebx,6
mul ebx ;умножаем след.аргумент на 6 'eax=eax*6'
add eax,13 ;eax=eax+13
ret ;выход из подпрограммы вычисления f(x)
```

Рис. 3.1: Текст файла func1.asm

Создадим исполняемый файл и проверим его работу с теми же аргументами (рис. 3.2).



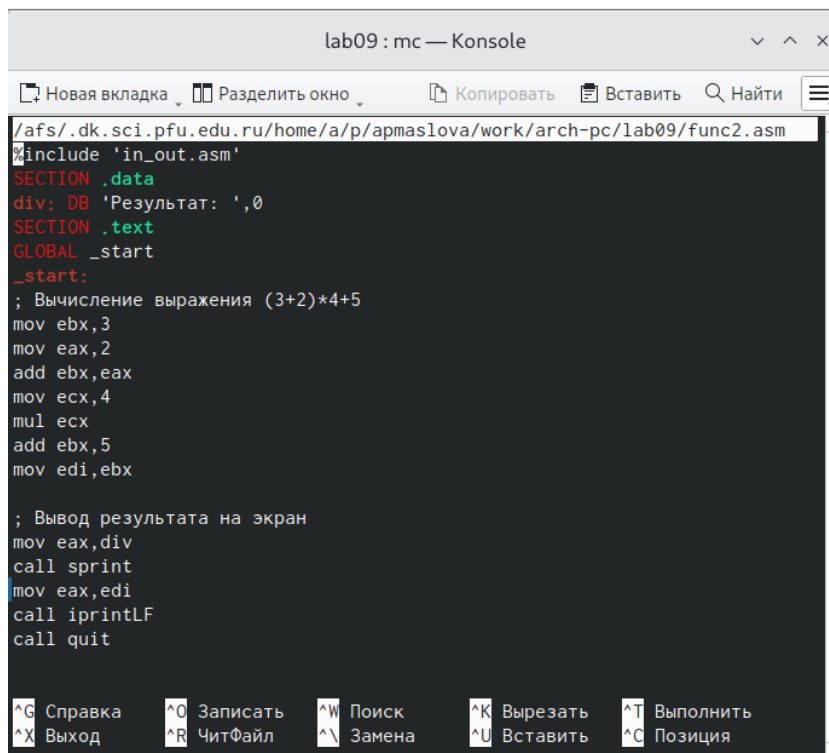
```
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ nasm -f elf func1.asm
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o func1 func1.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ./func1 4 6 8
Результат: 147
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ./func1 3 0 7 4
Результат: 136
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ./func1 1 4 6 8 2
Результат: 191
apmaslova@dk2n22 ~/work/arch-pc/lab09 $
```

Рис. 3.2: Создание и запуск исполняемого файла func1

Программа работает корректно: выдаются верные значения сумм. Приступим к выполнению следующего задания.

Задание №2

Создадим файл `func2.asm` и впишем в него текст программы вычисления выражения $(3 + 2) * 4 + 5$ (рис. 3.3).



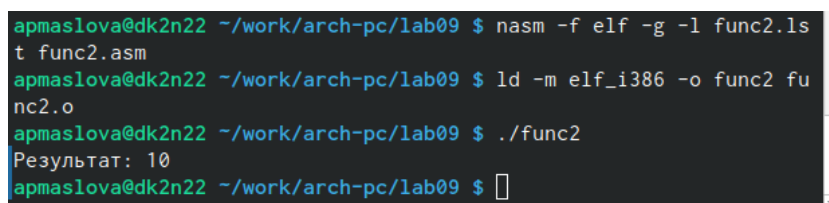
```
lab09 : mc — Konsole
/a/s/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab09/func2.asm
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx

; Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

^G Справка      ^O Записать     ^W Поиск        ^K Вырезать     ^T Выполнить
^X Выход        ^R ЧитФайл     ^\ Замена       ^U Вставить     ^C Позиция
```

Рис. 3.3: Текст файла `func2.asm`

Создадим исполняемый файл и запустим его (рис. 3.4).



```
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ nasm -f elf -g -l func2.ls
t func2.asm
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o func2 fu
nc2.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ./func2
Результат: 10
apmaslova@dk2n22 ~/work/arch-pc/lab09 $
```

Рис. 3.4: Создание и запуск исполняемого файла `func2`

Мы проверили, и действительно, при запуске программа даёт неверный ре-

зультат.

С помощью отладчика GDB проанализируем изменения значений регистров, определим ошибку и исправим её.

Для начала загрузим исполняемый файл в отладчик (рис. 3.5).

```
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ gdb func2
GNU gdb (Gentoo 13.2 vanilla) 13.2
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-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 func2...
(gdb) █
```

Рис. 3.5: Загрузка файла func2 в gdb

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

```
(gdb) disassemble _start
Dump of assembler code for function _start:
   0x080490e8 <+0>:  mov    $0x3,%ebx
   0x080490ed <+5>:  mov    $0x2,%eax
   0x080490f2 <+10>: add    %eax,%ebx
   0x080490f4 <+12>:  mov    $0x4,%ecx
   0x080490f9 <+17>:  mul    %ecx
   0x080490fb <+19>:  add    $0x5,%ebx
   0x080490fe <+22>:  mov    %ebx,%edi
   0x08049100 <+24>:  mov    $0x804a000,%eax
   0x08049105 <+29>:  call   0x804900f <sprint>
   0x0804910a <+34>:  mov    %edi,%eax
   0x0804910c <+36>:  call   0x8049086 <iprintf>
   0x08049111 <+41>:  call   0x80490db <quit>
End of assembler dump.
(gdb) █
```

Рис. 3.6: Дисассимилированный код программы func2

Включим режим псевдографики (рис. 3.7).

```
lab09: gdb — Konsole
[ Register Values Unavailable ]

0x8049000 <slen>      push    %ebx
0x8049001 <slen+1>    mov     %eax,%ebx
0x8049003 <nextchar>   cmpb    $0x0,(%eax)
0x8049006 <nextchar+3>   je      0x804900b <finished>
0x8049008 <nextchar+5>   inc     %eax
0x8049009 <nextchar+6>   jmp     0x8049003 <nextchar>
0x804900b <finished>   sub     %ebx,%eax

exec No process in: L?? PC: ??
(gdb) layout regs
(gdb) 
```

Рис. 3.7: Режим псевдографики для func2

Используем команду для просмотра значений регистров (рис. 3.8).

```
lab09: gdb — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить »
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>

B+> 0x80490e8 <_start> mov $0x3,%ebx
0x80490ed <_start+5> mov $0x2,%eax
0x80490f2 <_start+10> add %eax,%ebx
0x80490f4 <_start+12> mov $0x4,%ecx
0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19> add $0x5,%ebx
0x80490fe <_start+22> mov %ebx,%edi
0x8049100 <_start+24> mov $0x804a000,%eax
0x8049105 <_start+29> call 0x804900f <sprint>

native process 8101 In: _start L8 PC: 0x80490e8
edi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--ei
p        0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) 
```

Рис. 3.8: Просмотр значений регистров в func2

С помощью команды `break` установим точку останова по адресу на инструкции `add ebx, eax` (рис. 3.9).

```
lab09: gdb — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить »

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc220 0xffffc220
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>

B+> 0x80490e8 <_start> mov $0x3,%ebx
0x80490ed <_start+5> mov $0x2,%eax
b+ 0x80490f2 <_start+10> add %eax,%ebx
0x80490f4 <_start+12> mov $0x4,%ecx
0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19> add $0x5,%ebx
0x80490fe <_start+22> mov %ebx,%edi
0x8049100 <_start+24> mov $0x804a000,%eax
0x8049105 <_start+29> call 0x804900f <sprint>

native process 8101 In: _start L8 PC: 0x80490e8
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x080490e8 func2.asm:8
breakpoint already hit 1 time
(gdb) b *0x80490f2
Breakpoint 2 at 0x80490f2: file func2.asm, line 10.
(gdb) 
```

Рис. 3.9: Точка останова на инструкции add

Посмотрим значения регистров на этом этапе с помощью команды `i r` (рис. 3.10).

```
lab09: gdb — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить »

Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f2 0x80490f2 <_start+10>

B+ 0x80490e8 <_start>    mov    $0x3,%ebx
0x80490ed <_start+5>    mov    $0x2,%eax
B+> 0x80490f2 <_start+10> add    %eax,%ebx
0x80490f4 <_start+12>    mov    $0x4,%ecx
0x80490f9 <_start+17>    mul    %ecx
0x80490fb <_start+19>    add    $0x5,%ebx
0x80490fe <_start+22>    mov    %ebx,%edi
0x8049100 <_start+24>    mov    $0x804a000,%eax
0x8049105 <_start+29>    call   0x804900f <sprint>

native process 8305 In: _start L10 PC: 0x80490f2
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f2 0x80490f2 <_start+10>
eflags   0x202     [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 3.10: Посмотр значений регистров

Далее с помощью команды `si` перейдём к следующей инструкции и проследим за изменением значений регистров (рис. 3.11).


```
lab09: gdb — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить »

Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f4 0x80490f4 <_start+12>

B+ 0x80490e8 <_start>    mov    $0x3,%ebx
0x80490ed <_start+5>    mov    $0x2,%eax
B+ 0x80490f2 <_start+10> add    %eax,%ebx
> 0x80490f4 <_start+12> mov    $0x4,%ecx
0x80490f9 <_start+17>    mul    %ecx
0x80490fb <_start+19>    add    $0x5,%ebx
0x80490fe <_start+22>    mov    %ebx,%edi
0x8049100 <_start+24>    mov    $0x804a000,%eax
0x8049105 <_start+29>    call   0x804900f <sprint>

native process 8305 In: _start L11 PC: 0x80490f4
eip      0x80490f2 0x80490f2 <_start+10>
eflags   0x202    [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--cs
0x23     0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si
(gdb) 
```

Рис. 3.11: Команда si

Как мы видим, результат суммы чисел 2 и 3 записался в регистр ebx. Это могло послужить проблемой для дальнейшего вычисления произведения. Исправим это, изменив значение регистра eax и занеся в него значение 5 с помощью команды set (рис. 3.12).

```
lab09: gdb — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить »

Register group: general
eax      0x5      5
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f4 0x80490f4 <_start+12>

B+ 0x80490e8 <_start>      mov     $0x3,%ebx
0x80490ed <_start+5>      mov     $0x2,%eax
B+ 0x80490f2 <_start+10>   add     %eax,%ebx
> 0x80490f4 <_start+12>   mov     $0x4,%ecx
0x80490f9 <_start+17>      mul     %ecx
0x80490fb <_start+19>      add     $0x5,%ebx
0x80490fe <_start+22>      mov     %ebx,%edi
0x8049100 <_start+24>      mov     $0x804a000,%eax
0x8049105 <_start+29>      call    0x804900f <sprint>

native process 8305 In: _start L11 PC: 0x80490f4
0x23      35
ss        0x2b      43
ds        0x2b      43
es        0x2b      43
fs        0x0      0
gs        0x0      0
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) 
```

Рис. 3.12: Изменение значения регистра `eax` с помощью команды `set`

Переходим к следующей инструкции (рис. 3.13).

```
lab09: gdb — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить »

Register group: general
eax      0x5      5
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffc210 0xffffc210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>

B+ 0x80490e8 <_start>      mov     $0x3,%ebx
0x80490ed <_start+5>      mov     $0x2,%eax
B+ 0x80490f2 <_start+10>   add     %eax,%ebx
0x80490f4 <_start+12>   mov     $0x4,%ecx
> 0x80490f9 <_start+17>   mul     %ecx
0x80490fb <_start+19>   add     $0x5,%ebx
0x80490fe <_start+22>   mov     %ebx,%edi
0x8049100 <_start+24>   mov     $0x804a000,%eax
0x8049105 <_start+29>   call    0x804900f <sprint>

native process 8305 In: _start L12 PC: 0x80490f9
ss      0x2b      43
ds      0x2b      43
es      0x2b      43
fs      0x0      0
gs      0x0      0
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
(gdb) 
```

Рис. 3.13: Переход на следующий шаг

В результате этого шага мы поместили в регистр `ecx` значение 4 для вычисления произведения. После произведения значения регистров будут следующими: (рис. 3.14).

```
lab09: gdb — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить »

Register group: general
eax      0x14      20
ecx      0x4       4
edx      0x0       0
ebx      0x5       5
esp      0xffffc210 0xffffc210
ebp      0x0       0
esi      0x0       0
edi      0x0       0
eip      0x80490fb 0x80490fb <_start+19>

B+ 0x80490e8 <_start>      mov     $0x3,%ebx
0x80490ed <_start+5>      mov     $0x2,%eax
B+ 0x80490f2 <_start+10>   add     %eax,%ebx
0x80490f4 <_start+12>   mov     $0x4,%ecx
0x80490f9 <_start+17>   mul     %ecx
> 0x80490fb <_start+19>   add     $0x5,%ebx
0x80490fe <_start+22>   mov     %ebx,%edi
0x8049100 <_start+24>   mov     $0x804a000,%eax
0x8049105 <_start+29>   call    0x804900f <sprint>

native process 8305 In: _start L13 PC: 0x80490fb
ds      0x2b      43
es      0x2b      43
fs      0x0       0
gs      0x0       0
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
(gdb) si
(gdb) 
```

Рис. 3.14: Результат вычисления произведения

В результате в регистр `eax` было помещено значение произведения 20.

Далее к этому значению нужно прибавить 5. В программе за результат отвечает регистр `ebx`. Поместим в него значение $20 + 5 = 25$ и запустим программу на вывод конечного результата(рис. 3.15).

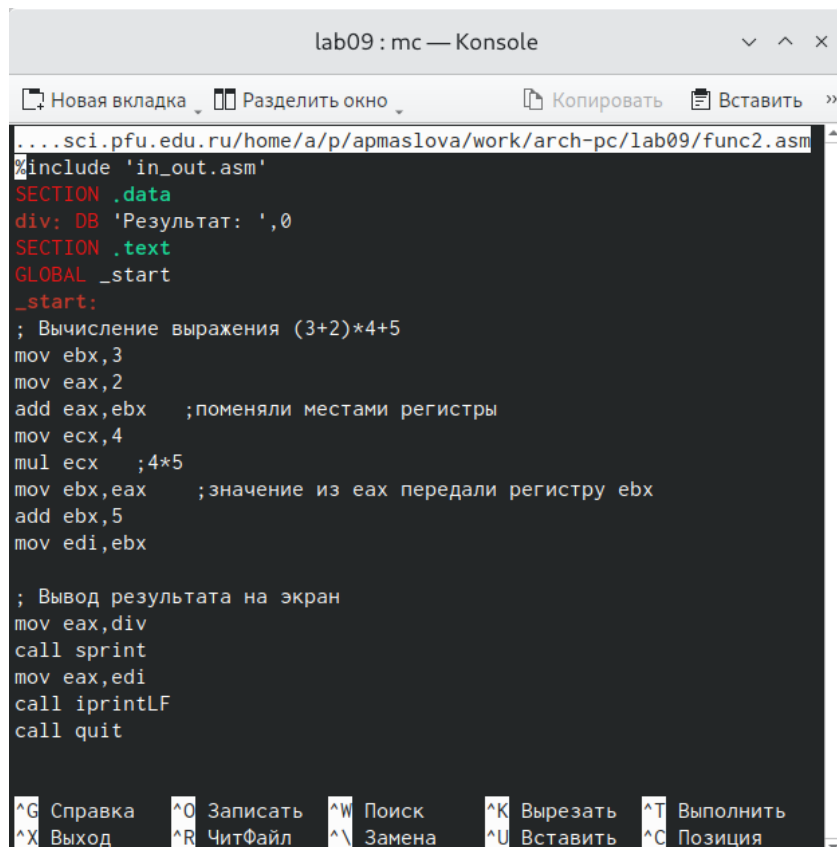
```
lab09: gdb — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить »
eax      0x14      20
ecx      0x4       4
edx      0x0       0
ebx      0x19      25
esp      0xffffc210 0xffffc210
ebp      0x0       0x0
esi      0x0       0
edi      0x0       0
eip      0x80490fe 0x80490fe <_start+22>

B+ 0x80490e8 <_start>    mov     $0x3,%ebx
0x80490fe <_start+22>    mov     %ebx,%edi
B+ 0x8049100 <_start+24>    add     %eax,%eax
0x8049105 <_start+29>    call    0x804900f <sprint>
0x804910a <_start+34>    mul     %edi,%eax
0x804910c <_start+36>    call    0x8049086 <iprintLF>
> 0x8049111 <_start+41>    call    0x80490db <quit>
                                0,%eax
                                nt>

native process 8305 In: _start L14 PC: 0x80490fe
(gdb) p No process In: L?? PC: ??
(gdb) si
(gdb) si
(gdb) si
(gdb) set $ebx=25
(gdb) p/s $ebx
$2 = 25
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 8305) exited normally]
(gdb)
```

Рис. 3.15: Завершение выполнения программы с помощью команды с

Как мы видим, с учётом всех изменений программа выдаёт верный результат. Теперь изменим код программы в файле func2.asm (рис. 3.16).



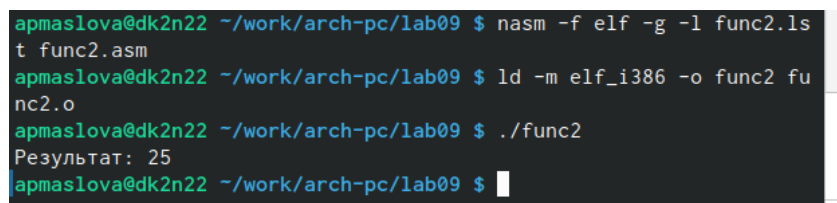
```
lab09: mc — Konsole
...sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab09/func2.asm
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx ;поменяли местами регистры
mov ecx,4
mul ecx ;4*5
mov ebx,eax ;значение из eax передали регистру ebx
add ebx,5
mov edi,ebx

; Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^C Позиция
```

Рис. 3.16: Исправленный текст программы в файле func2

Теперь создадим исполняемый файл и проверим корректность работы программы (рис. 3.17).



```
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ nasm -f elf -g -l func2.ls
t func2.asm
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o func2 fu
nc2.o
apmaslova@dk2n22 ~/work/arch-pc/lab09 $ ./func2
Результат: 25
apmaslova@dk2n22 ~/work/arch-pc/lab09 $
```

Рис. 3.17: Проверка работы программы func2

Теперь программа выдаёт верный результат.

4 Выводы

Мы научились писать программы с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.