

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров и операционные системы

Пестова Ева Константиновна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
5	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Создание файлов для лабораторной работы	10
4.2	Ввод текста программы из листинга 9.1	11
4.3	Запуск исполняемого файла	11
4.4	Изменение текста программы согласно заданию	12
4.5	Запуск исполняемого файла	12
4.6	Ввод текста программы из листинга 9.2	13
4.7	Загрузка исполняемого файла в отладчик	13
4.8	Проверка работы файла с помощью команды run	14
4.9	Установка брейкпоинта и запуск программы	14
4.10	Использование команд disassemble и disassembly-flavor intel	15
4.11	Включение режима псевдографики	16
4.12	Установление точек останова	16
4.13	Просмотр значений переменных	17
4.14	Использование команды set	17
4.15	Использование команды set для изменения значения регистра	18
4.16	Создание файла	18
4.17	Загрузка файла с аргументами в отладчик	19
4.18	Установление точки останова и запуск программы	19
4.19	Просмотр значений, введенных в стек	19
4.20	Написание кода подпрограммы	20
4.21	Запуск программы и проверка его вывода	21
4.22	Ввод текста программы из листинга 9.3	21
4.23	Создание и запуск исполняемого файла	21
4.24	Неверное изменение регистра	22
4.25	Исправление ошибки	23
4.26	Ошибка исправлена	23

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программ с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

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. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки

отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда `quit` (или сокращённо `q`).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`.

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`).

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

Обратно точка останова активируется командой `enable`.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`.

Для продолжения остановленной программы используется команда `continue` (`c`). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке).

Команда `stepi` (кратко `sl`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

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. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При

этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `еір`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

1. Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm (рис. [4.1]).

```
ekpestova@dk2n26 ~ $ mkdir ~/work/arch-pc/lab09  
ekpestova@dk2n26 ~ $ cd ~/work/arch-pc/lab09  
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ touch lab09-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1 (рис. [4.2]).

```

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 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [res], eax
35 ret ; выход из подпрограммы

```

Рис. 4.2: Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу (рис. [4.3]).

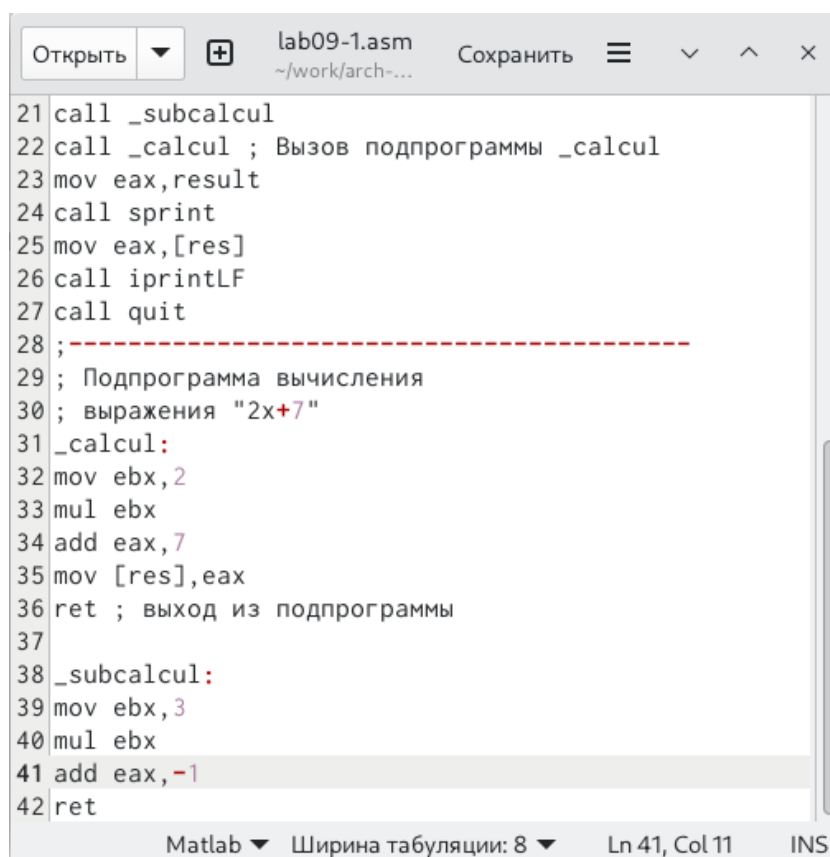
```

ekpestova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=17

```

Рис. 4.3: Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$ (рис. [4.4]).

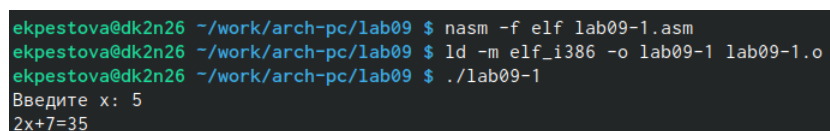


```
21 call _subcalcul
22 call _calcul ; Вызов подпрограммы _calcul
23 mov eax,result
24 call sprint
25 mov eax,[res]
26 call iprintLF
27 call quit
28 ; -----
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32 mov ebx,2
33 mul ebx
34 add eax,7
35 mov [res],eax
36 ret ; выход из подпрограммы
37
38 _subcalcul:
39 mov ebx,3
40 mul ebx
41 add eax,-1
42 ret
```

Matlab ▾ Ширина табуляции: 8 ▾ Ln 41, Col 11 INS

Рис. 4.4: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу (рис. [4.5]).



```
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 5
2x+7=35
```

Рис. 4.5: Запуск исполняемого файла

2. Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2 (рис. [4.6]).

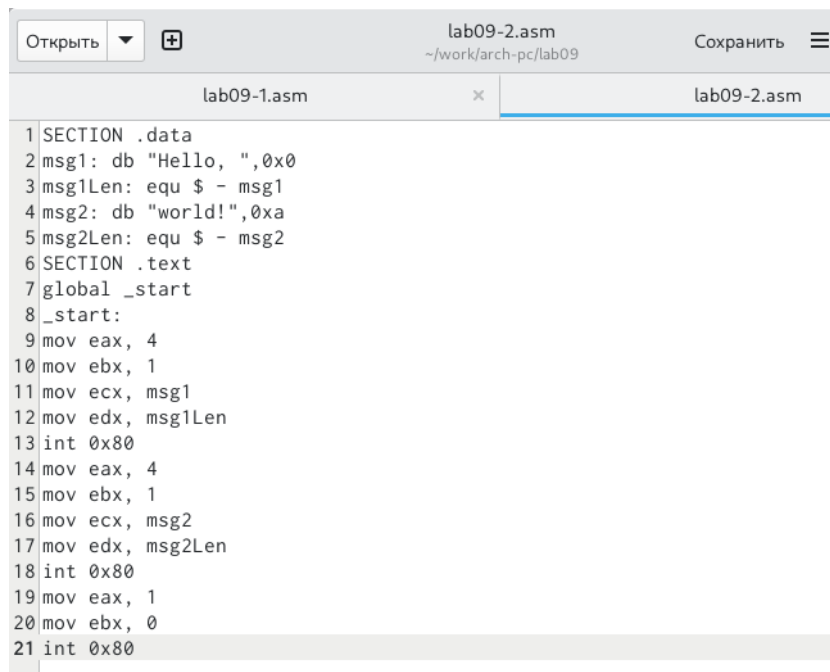


Рис. 4.6: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g' (рис. [??]).

```

ekpestova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o

```

Загружаю исполняемый файл в отладчик gdb (рис. [4.7]).

```

ekpestova@dk2n26 ~/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...

```

Рис. 4.7: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. [4.8]).

```
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/k/ekpestova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 7448) exited normally]
```

Рис. 4.8: Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start` и запускаю её (рис. [4.9]).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/k/ekpestova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
```

Рис. 4.9: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel` (рис. [4.10]).

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     $0x4,%eax
      0x08049005 <+5>:      mov     $0x1,%ebx
      0x0804900a <+10>:     mov     $0x804a000,%ecx
      0x0804900f <+15>:     mov     $0x8,%edx
      0x08049014 <+20>:     int     $0x80
      0x08049016 <+22>:     mov     $0x4,%eax
      0x0804901b <+27>:     mov     $0x1,%ebx
      0x08049020 <+32>:     mov     $0x804a008,%ecx
      0x08049025 <+37>:     mov     $0x7,%edx
      0x0804902a <+42>:     int     $0x80
      0x0804902c <+44>:     mov     $0x1,%eax
      0x08049031 <+49>:     mov     $0x0,%ebx
      0x08049036 <+54>:     int     $0x80
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
      0x08049005 <+5>:      mov     ebx,0x1
      0x0804900a <+10>:     mov     ecx,0x804a000
      0x0804900f <+15>:     mov     edx,0x8
      0x08049014 <+20>:     int     0x80
      0x08049016 <+22>:     mov     eax,0x4
      0x0804901b <+27>:     mov     ebx,0x1
      0x08049020 <+32>:     mov     ecx,0x804a008
      0x08049025 <+37>:     mov     edx,0x7
      0x0804902a <+42>:     int     0x80
      0x0804902c <+44>:     mov     eax,0x1
      0x08049031 <+49>:     mov     ebx,0x0
      0x08049036 <+54>:     int     0x80
End of assembler dump.

```

Рис. 4.10: Использование команд disassemble и disassembly-flavor intel

В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs (рис. [4.11]).

```

[ 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
    0x804902c <_start+44>  mov    eax,0x1

native process 7682 In: _start
(gdb) layout regs

```

Рис. 4.11: Включение режима псевдографики

3. Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова (рис. [4.12]).

```

(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(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   0x08049031 lab09-2.asm:20

```

Рис. 4.12: Установление точек останова

4. Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi`.

Просматриваю значение переменной `msg1` по имени с помощью команды `x/1sb &msg1` и значение переменной `msg2` по ее адресу (рис. [4.13]).

```
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) █
```

Рис. 4.13: Просмотр значений переменных

С помощью команды `set` изменяю первый символ переменной `msg1` и заменяю первый символ в переменной `msg2` (рис. [4.14]).

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

Рис. 4.14: Использование команды `set`

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием (рис. [4.15]).

```

(gdb) p/x $edx
$6 = 0x0
(gdb) p/s$eax
$7 = 4
(gdb) p/t $eax
$8 = 100
(gdb) p/s $ecx
$9 = 0

```

Рис. 4.15: Использование команды set для изменения значения регистра

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется

5. Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл (рис. [4.16]).

```

ekpestova@dk2n26 ~/work/arch-pc/lab09 $ cp /afs/.dk.sci.pfu.edu.ru/home/e/k/ekpestova/work/arch-
pc/lab08/lab8-2.asm /afs/.dk.sci.pfu.edu.ru/home/e/k/ekpestova/work/arch-pc/lab09/lab09-3.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o

```

Рис. 4.16: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа -args (рис. [4.17]).

```

ekpestova@dk2n26 ~/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...

```

Рис. 4.17: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. [4.18]).

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/e/k/ekpestova/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' количество

```

Рис. 4.18: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам (рис. [4.19]).

```

(gdb) x/x $esp
0xfffffc1b0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc454: "/afs/.dk.sci.pfu.edu.ru/home/e/k/ekpestova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc49a: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc4ac: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc4bd: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc4bf: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>

```

Рис. 4.19: Просмотр значений, введенных в стек

6. Задания для самостоятельной работы

- 1) Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. [4.20]).

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат:",0
4 SECTION .bss
5 x resb 10
6 SECTION .text
7 global _start
8 _start:
9 pop ecx
10 pop edx
11 sub ecx,1
12 mov esi, 0
13 mov edi,2
14 call .next
15 .next:
16 cmp ecx,0h
17 jz .done
18 pop eax
19 call atoi
20 mov [x],eax
21 mul edi
22 add eax,7
23 add esi,eax
24 loop .next
25 .done:
26 mov eax, msg
27 call sprint
28 mov eax, esi
29 call iprintLF
30 call quit
31 ret

```

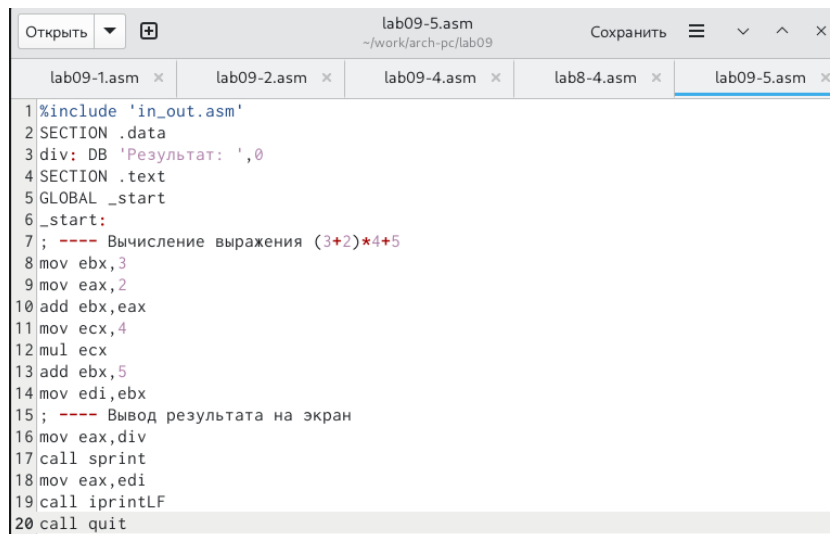
Рис. 4.20: Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно (рис. [4.21]).

```
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ./lab09-4 1 2 3
Результат: 27
```

Рис. 4.21: Запуск программы и проверка его вывода

2) Ввожу в файл lab09-5.asm текст программы из листинга 9.3 (рис. [4.22]).



```
lab09-5.asm
~/work/arch-pc/lab09
lab09-1.asm x lab09-2.asm x lab09-4.asm x lab8-4.asm x lab09-5.asm x
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 4.22: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его (рис. [4.23]).

```
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ touch lab09-5.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 10
```

Рис. 4.23: Создание и запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ. Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой

инструкции, связанной с вычислениями. С помощью команды continue проходим по каждому брейкпоинту и слежу за изменениями значений регистров. При выполнении инструкции mul ecx происходит умножение ecx на eax, то есть 4 на 2, вместо умножения 4 на 5 (регистр ebx). Происходит это из-за того, что стоящая перед mov ecx,4 инструкция add ebx,eax не связана с mul ecx, но связана инструкция mov eax,2. Из-за этого мы получаем неправильный ответ (рис. [4.24]).

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

B+ 0x80490f9 <_start+17> mul    ecx
B+ 0x80490fb <_start+19> add    ebx,0x5
B+ 0x80490fe <_start+22> mov    edi,ebx
0x8049100 <_start+24> mov    eax,0x804a000
0x8049105 <_start+29> call   0x804900f <sprint>

native process 14573 In: _start L14 PC: 0x80490fe
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb) c
Continuing.

Breakpoint 7, _start () at task2.asm:14
(gdb)

```

Рис. 4.24: Неверное изменение регистра

Исправляем ошибку, добавляя после add ebx,eax mov eax,ebx и заменяя ebx на eax в инструкциях add ebx,5 и mov edi,ebx (рис. [4.25]).

```

1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov eax,ebx
12 mov ecx,4
13 mul ecx
14 add eax,5
15 mov edi,eax
16 ; ---- Вывод результата на экран
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 call quit

```

Рис. 4.25: Исправление ошибки

Создаем исполняемый файл и запускаем его. Убеждаемся, что ошибка исправлена (рис. [4.26]).

```

ekpestova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
ekpestova@dk2n26 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 25

```

Рис. 4.26: Ошибка исправлена

5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась 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.