

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Гончарь Анастасия Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация подпрограмм в NASM	8
4.2	Отладка программ с помощью GDB	10
4.3	Добавление точек останова	14
4.4	Работа с данными программы в GDB	15
4.5	Обработка аргументов командной строки в GDB	17
4.6	Задание для самостоятельной работы	19
4.6.1	Листинг для файла lab9-4.asm	21
4.6.2	Листинг для файла lab9-5.asm	27
5	Выводы	29
	Список литературы	30

Список иллюстраций

4.1	Создание каталога и файла	8
4.2	Текст программы в файле	8
4.3	Запуск файла	9
4.4	Измененный текст программы	9
4.5	Запуск файла	10
4.6	Создание файла	10
4.7	Текст второй программы	11
4.8	Загрузка файла в отладчик	12
4.9	Проверка работы программы	12
4.10	Установка брейкпоинта на метку _start и запуск программы	12
4.11	Дисассимплированный код	13
4.12	Intel'овское отображение	13
4.13	Псевдографика	14
4.14	Проверка меток	15
4.15	Изменение регистров	15
4.16	Просмотр значений переменных	16
4.17	Изменение значений переменных	16
4.18	Значение регистров ехх и еах	16
4.19	Значение регистров ебх	17
4.20	Копирование файла	17
4.21	Создание исполняемого файла	17
4.22	Запуск файла	18
4.23	Запуск файла lab10-3 через метку	18
4.24	Адрес вершины стека	18
4.25	Все позиции стека	19
4.26	Текст программы	20
4.27	Запуск файла	20
4.28	Создание файла	22
4.29	Текст программы в файле	23
4.30	Запуск программы	23
4.31	Запуск программы в отладчике	24
4.32	Анализ регистров	25
4.33	Изменение программы	26
4.34	Запуск программы	27

Список таблиц

3.1	Описание некоторых каталогов файловой системы GNU Linux . . .	7
-----	---	---

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм, а также 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.Задание для самостоятельной работы

3 Теоретическое введение

Здесь описываются теоретические аспекты, связанные с выполнением работы.

Например, в табл. 3.1 приведено краткое описание стандартных каталогов Unix.

Таблица 3.1: Описание некоторых каталогов файловой системы GNU Linux

Имя каталога	Описание каталога
/	Корневая директория, содержащая всю файловую систему
/bin	Основные системные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям
/etc	Общесистемные конфигурационные файлы и файлы конфигурации установленных программ
/home	Содержит домашние директории пользователей, которые, в свою очередь, содержат персональные настройки и данные пользователя
/media	Точки монтирования для сменных носителей
/root	Домашняя директория пользователя root
/tmp	Временные файлы
/usr	Вторичная иерархия для данных пользователя

Более подробно про Unix см. в [1–4].

4 Выполнение лабораторной работы

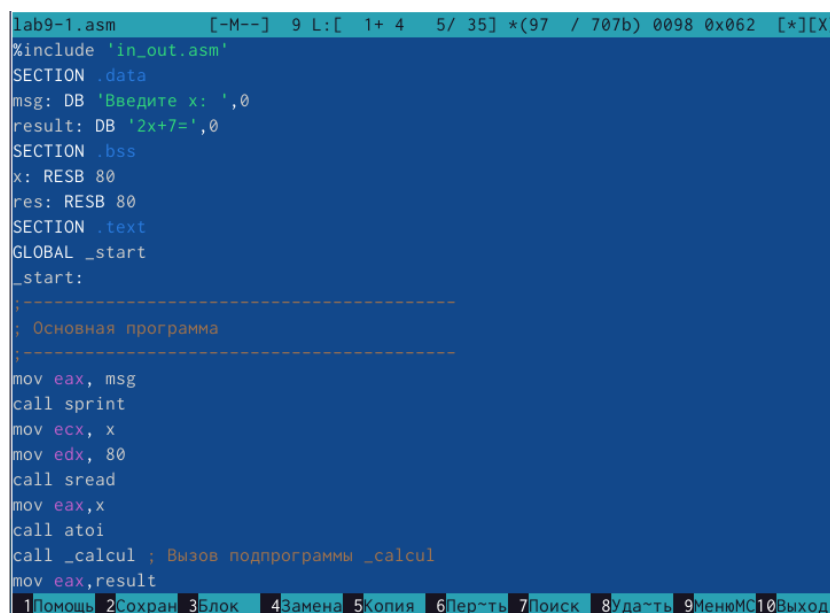
4.1 Реализация подпрограмм в NASM

Сначала я создаю каталог lab09 и файл lab9-1.asm (рис. 4.1).

```
aagoncharj@dk4n65 ~ $ mkdir ~/work/arch-pc/lab09
aagoncharj@dk4n65 ~ $ cd ~/work/arch-pc/lab09
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ touch lab9-1.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $
```

Рис. 4.1: Создание каталога и файла

Открываю созданный файл и ввожу в него тест программы из листинга 9.1 (рис. 4.2).



```
lab9-1.asm      [-M--]  9 L:[ 1+ 4  5/ 35] *(97 / 707b) 0098 0x062  [*][X]
#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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Переть 7Поиск 8Удалить 9МенюMC 10Выход
```

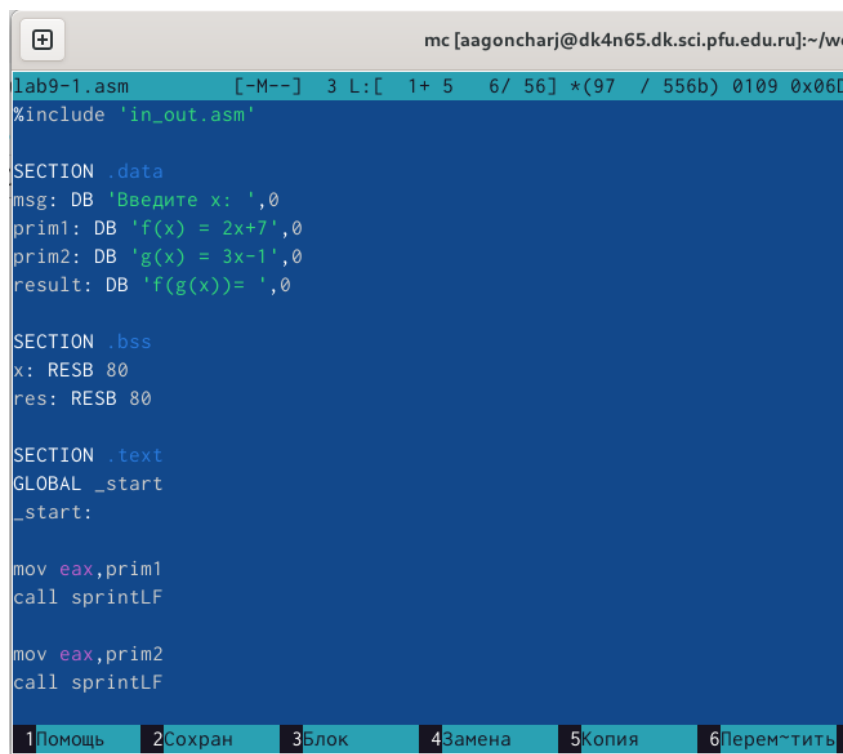
Рис. 4.2: Текст программы в файле

Создаю исполняемый файл и запускаю его (рис. 4.3).

```
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ./lab9-1
Введите x: 5
2x+7=17
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ █
```

Рис. 4.3: Запуск файла

Изменяю текст программы так, чтобы она вычисляла значение выражения $f(g(x))$ (рис. 4.4).



```
lab9-1.asm      [-M--]  3 L: [ 1+ 5  6/ 56] *(97 / 556b) 0109 0x060
#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
prim1: DB 'f(x) = 2x+7',0
prim2: DB 'g(x) = 3x-1',0
result: DB 'f(g(x))= ',0

SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax,prim1
call sprintLF

mov eax,prim2
call sprintLF
```

Рис. 4.4: Измененный текст программы

Создаю исполняемый файл и запускаю его (рис. 4.5).

```
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf lab9-1.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-1 lab9-1.o
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ./lab9-1
f(x) = 2x+7
g(x) = 3x-1
Введите x: 1
f(g(x))= 11
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $
```

Рис. 4.5: Запуск файла

4.2 Отладка программ с помощью GDB

Создаю файл lab9-2.asm (рис. 4.6).

```
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ touch lab9-2.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $
```

Рис. 4.6: Создание файла

Ввожу в него текст программы из листинга 9.2 (рис. 4.7).

```

lab9-2.asm      [-M--]  8 L:[ 1+20
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. 4.7: Текст второй программы

Далее получаю исполняемый файл с помощью ключа -g и загружаю этот файл в отладчик gdb (рис. 4.8).

```

aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab9-2.lst lab9-2.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-2 lab9-2.o
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ gdb lab9-2
GNU gdb (Gentoo 14.2 vanilla) 14.2
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 lab9-2...
(gdb) █

```

Рис. 4.8: Загрузка файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `run` (сокращённо `r`) (рис. 4.9).

```

(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aagoncharj/work/arch-pc/lab09/lab9-2
Hello, world!
[Inferior 1 (process 5052) exited normally]
(gdb)

```

Рис. 4.9: Проверка работы программы

Теперь устанавливаю брейкпоинт на метку `_start` и запускаю программу (рис. 4.10).

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab9-2.asm, line 9.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aagoncharj/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 4.10: Установка брейкпоинта на метку `_start` и запуск программы

Посматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. 4.11).

```

(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. 4.11: Дисассимплированный код

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 4.12).

```

(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. 4.12: Intel'овское отображение

Для удобства включаю режим псевдографики (рис. 4.13).



Рис. 4.13: Псевдографика

4.3 Добавление точек останова

На предыдущих шагах была установлена точка остановки по имени метки (`_start`), проверяю это с помощью команды `info breakpoints` (кратко `i b`). Также устанавливаю еще одну точку остановки по адресу инструкции и снова смотрю информацию о всех установленных метках (рис. 4.14).

```

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
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54>   int     0x80

native process 5090 In: _start          L9    PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep  y  0x08049000 lab9-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep  y  0x08049000 lab9-2.asm:9
        breakpoint already hit 1 time
2        breakpoint keep  y  0x08049031 lab9-2.asm:20
(gdb)

```

Рис. 4.14: Проверка меток

4.4 Работа с данными программы в GDB

С помощью команды `si` я посмотрела регистры и изменила их (рис. 4.15).

```

--Register group: general--
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc460 0xffffc460
ebp      0x0      0x0
esi      0x0      0

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 5090 In: _start          L15    PC: 0x804901b
es       0x2b      43
fs       0x0      0
gs       0x0      0
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 4.15: Изменение регистров

Далее я посмотрела значения переменных msg1 и msg2 (рис. 4.16).

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

Рис. 4.16: Просмотр значений переменных

С помощью команды set я изменяю значения переменных msg1 и msg2 (рис. 4.17).

```
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhlllo, "
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb) █
```

Рис. 4.17: Изменение значений переменных

Теперь вывожу значение регистров ecx и eax с помощью (рис. 4.18).

```
$1 = void
(gdb) p/s $eax
$2 = 4
(gdb) p/t $eax
$3 = 100
(gdb) p/s $ecx
$4 = 134520832
(gdb) p/x $ecx
$5 = 0x804a000
(gdb) █
```

Рис. 4.18: Значение регистров ecx и eax

Теперь изменяю значение регистра `ebx` (рис. 4.19). Команда выводит два разных значения, так как в первый раз мы вносим значение 2, а во второй раз регистр равен двум.

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb) █
```

Рис. 4.19: Значение регистров `ebx`

Завершаю выполнение программы с помощью команды `continue` (сокращенно `c`) и выхожу из GDB с помощью команды `quit` (сокращенно `q`).

4.5 Обработка аргументов командной строки в GDB

Копирую файл `lab8-2.asm` и переименовываю его (рис. 4.20).

```
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab9-3.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ █
```

Рис. 4.20: Копирование файла

Создаю исполняемый файл (рис. 4.21).

```
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab9-3.lst lab9-3.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-3 lab9-3.o
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ █
```

Рис. 4.21: Создание исполняемого файла

Запускаю файл, указав аргументы (рис. 4.22).

```
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ gdb --args lab9-3 аргумент1 аргумент 2
'аргумент 3'
GNU gdb (Gentoo 14.2 vanilla) 14.2
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 lab9-3...
(gdb)
```

Рис. 4.22: Запуск файла

Ставлю метку на `_start` и запустил файл (рис. 4.23).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 8.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aagoncharj/work/arch-pc/lab09
/lab9-3 аргумент1 аргумент 2 аргумент\ 3

Breakpoint 1, _start () at lab9-3.asm:8
8      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) █
```

Рис. 4.23: Запуск файла lab10-3 через метку

Проверяю адрес вершины стека, там хранится 5 элементов (рис. 4.24).

```
(gdb) x/x $esp
0xfffffc410: 0x00000005
(gdb) █
```

Рис. 4.24: Адрес вершины стека

Теперь просматриваю все позиции стека. По первому адресу хранится адрес, а в остальных адресах хранятся элементы. при этом элементы расположены с интервалом в 4 единицы, так как стек может хранить до 4 байт, и для того чтобы данные сохранялись нормально и без помех, компьютер использует новый стек для новой информации (рис. 4.25).

```

(gdb) x/s *(void**)(esp + 4)
0xfffffc66f:      "/afs/.dk.sci.pfu.edu.ru/home/a/aagoncharj/work/arch-pc/lab09/
lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc6b5:      "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc6c7:      "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc6d8:      "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc6da:      "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.25: Все позиции стека

4.6 Задание для самостоятельной работы

- 1) Копирую файл lab8-4.asm из лабораторной работы №8 в папку для лабораторной №9 с названием lab9-4.asm и изменяю текст программы так, чтобы она вычисляла значение функции как подпрограмму (рис. 4.26).

```
lab9-4.asm      [----]  0 L:[ 1+10 11/ 44] *(137 / 385b) 0010
%include 'in_out.asm'

SECTION .data
prim DB 'f(x)=6x+13',0
otv DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

pop ecx

pop edx

sub ecx,1

mov esi,0

mov eax,prim
call sprintf
next:
cmp ecx,0
jz _end

pop eax
call atoi
call fir
add esi,eax
mul ebx

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9М
```

Рис. 4.26: Текст программы

Создаю исполняемый файл и запускаю его (рис. 4.27).

```
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf lab9-4.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-4 lab9-4.o
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ./lab9-4 2
f(x)=6x+13
Результат: 25
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ./lab9-4 1
f(x)=6x+13
Результат: 19
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ./lab9-4 5
f(x)=6x+13
Результат: 43
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $
```

Рис. 4.27: Запуск файла

4.6.1 Листинг для файла lab9-4.asm

```
%include 'in_out.asm'

SECTION .data
prim DB 'f(x)=6x+13',0
otv DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

    pop ecx

    pop edx

    sub ecx,1

    mov esi,0

    mov eax,prim
    call sprintf
next:
    cmp ecx,0
    jz _end

    pop eax
    call atoi
    call fir
    add esi,eax
    mul ebx
```

```

loop next

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

fir:
mov ebx,6
mul ebx
add eax,13
ret

```

2) Создаю файл lab9-5.asm (рис. 4.28).

```

aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ touch lab9-5.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ █

```

Рис. 4.28: Создание файла

Открываю созданный файл и ввожу в него текст программы из листинга 9.3 (рис. 4.29).

```
lab9-5.asm [-M--] 13 L:[ 1+18 19/ 20] *(
#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. 4.29: Текст программы в файле

Создаю исполняемый файл и запускаю его (рис. 4.30). Ошибка арифметическая, так как вместо 25, программа выводит 10.

```
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf lab9-5.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-5 lab9-5.o
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ./lab9-5
Результат: 10
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ █
```

Рис. 4.30: Запуск программы

После появления ошибки, я запускаю программу в отладчике (рис. 4.31).

```

aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab9-5.lst lab9-5.asmaagoncharj@dk4n65 ~/work
/arch-pc/lab09 $ ld -m elf_i386 -o lab9-5 lab9-5.o
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ gdb lab9-5
GNU gdb (Gentoo 14.2 vanilla) 14.2
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 lab9-5...
(gdb) b _start
Breakpoint 1 at 0x00490e8: file lab9-5.asm, line 8.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aagoncharj/work/arch-pc/lab09/lab9-5

Breakpoint 1, _start () at lab9-5.asm:8
8      mov     ebx,3
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x00490e8 <+0>:      mov     ebx,0x3
0x00490ed <+5>:      mov     eax,0x2
0x00490f2 <+10>:     add     ebx,eax
0x00490f4 <+12>:     mov     ecx,0x4
0x00490f9 <+17>:     mul     ecx
0x00490fb <+19>:     add     ebx,0x5
0x00490fe <+22>:     mov     edi,ebx
0x0049100 <+24>:     mov     eax,0x804a000
0x0049105 <+29>:     call    0x804900f <sprint>
0x004910a <+34>:     mov     eax,edi
0x004910c <+36>:     call    0x8049086 <iprintLF>
0x0049111 <+41>:     call    0x80490db <quit>
End of assembler dump.

```

Рис. 4.31: Запуск программы в отладчике

Я открыла регистры, поняла что регистры стоят не на своих местах и исправила это (рис. 4.32).


```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffc4a0 0xffffc4a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fe 0x80490fe <_start+22>
eflags   0x206    [ PF IF ]
cs       0x23     35

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>

native process 8647 In: _start          L14  PC: 0x80490fe
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) █
```

Рис. 4.32: Анализ регистров

Я изменила регистры(рис. 4.33) и запустила программу(рис. 4.34). Программа вывела 25, то есть все работает правильно.

```
lab9-5.asm      [----] 13 L:[ 1+18 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 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. 4.33: Изменение программы

```

aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab9-5.lst lab9-5.asm
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab9-5 lab9-5.o
aagoncharj@dk4n65 ~/work/arch-pc/lab09 $ gdb lab9-5
GNU gdb (Gentoo 14.2 vanilla) 14.2
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 lab9-5...
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/a/aagoncharj/work/arch-pc/lab09/lab9-5
Результат: 25
[Inferior 1 (process 9600) exited normally]
(gdb) █

```

Рис. 4.34: Запуск программы

4.6.2 Листинг для файла lab9-5.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
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
```

5 Выводы

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а также ознакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.
2. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.