

Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Самойлова Софья Дмитриевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
4.1	Реализация подпрограмм в NASM	7
4.2	Отладка программ с помощью GDB	9
4.3	Добавление точек останова	12
4.4	Работа с данными программы в GDB	14
4.5	Обработка аргументов командной строки в GDB	15
4.6	Задание для самостоятельной работы	17
5	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	Создание каталога	7
4.2	Код из листинга	8
4.3	Запуск программы	8
4.4	Изменение программы первого листинга	9
4.5	Работа программы первого листинга	9
4.6	Работа программы второго листинга	10
4.7	Запуск программы в отладчике	10
4.8	Запуск отладчика с брейкпойнтом	11
4.9	Дисассимилирование программы	11
4.10	Режим псевдографики	12
4.11	Список брейкпойнтов	13
4.12	Добавление второй точки останова	13
4.13	Просмотр содержимого регистров	14
4.14	Просмотр содержимого переменных вторым способом	14
4.15	Изменение содержимого переменных	14
4.16	Просмотр значения регистра разными представлениями	15
4.17	Примеры использования команды set	15
4.18	Подготовка новой программы	16
4.19	Проверка работы стека	16
4.20	Измененная программа предыдущей лабораторной работы	17
4.21	Ошибка программы	19
4.22	Проверка корректировок в программе	19

1 Цель работы

Целью лабораторной работы является 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. Самостоятельное выполнение заданий по материалам лабораторной работы

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки.

Можно выделить следующие типы ошибок:

- *синтаксические ошибки* — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • *семантические ошибки* — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • *ошибки в процессе выполнения* — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

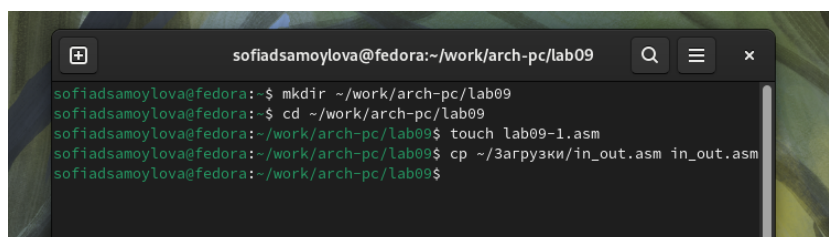
Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

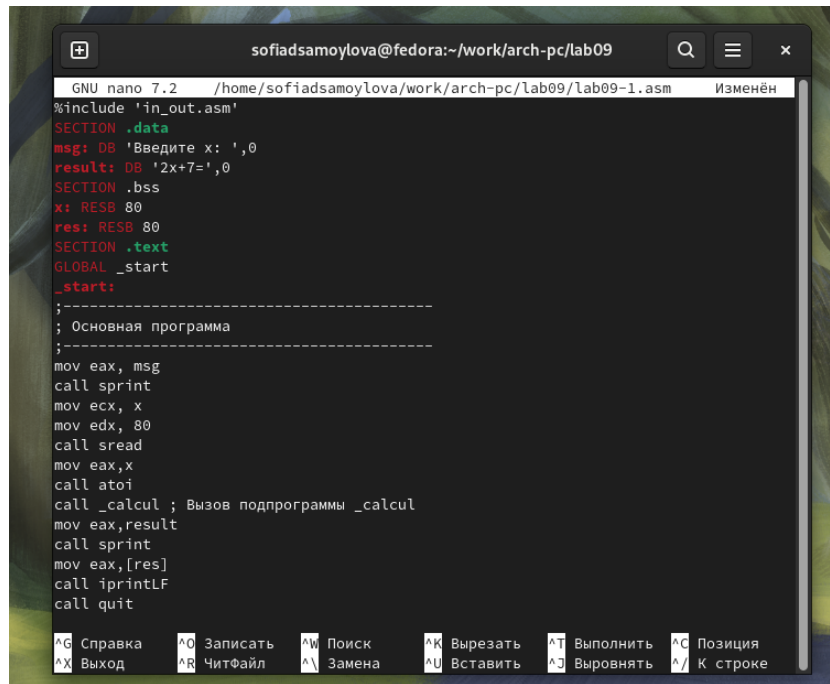
Создаю каталог для программ лабораторной работы № 9, перехожу в него и создаю файл `lab8-1.asm`, дополнительно копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, т.к. он будет использоваться в других программах (рис. 4.1).



```
sofiadsamoylova@fedora:~/work/arch-pc/lab09
sofiadsamoylova@fedora:~$ mkdir ~/work/arch-pc/lab09
sofiadsamoylova@fedora:~$ cd ~/work/arch-pc/lab09
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ touch lab09-1.asm
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ cp ~/Загрузки/in_out.asm in_out.asm
sofiadsamoylova@fedora:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание каталога

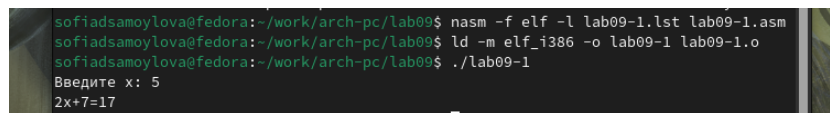
Копирую в файл код из листинга (рис. 4.2).



```
GNU nano 7.2 /home/sofiadsamoylova/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:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
```

Рис. 4.2: Код из листинга

Компилирую и запускаю его, данная программа выполняет вычисление функции (рис. 4.3).



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

Рис. 4.3: Запуск программы

Изменяю текст программы, добавив в нее подпрограмму (рис. 4.4).


```

call quit

;-----
; Подпрограмма вычисления выражения "f(g(x))"
_calcul:
    call _subcalcul ; Вызов подпрограммы _subcalcul для вычисления g(x)
    ; Результат g(x) теперь в eax, вычисляем f(g(x))
    mov ebx, 2
    mul ebx        ; Умножаем g(x) на 2
    add eax, 7     ; Добавляем 7
    mov [res], eax ; Сохраняем результат f(g(x)) в res
    ret           ; выход из подпрограммы

;-----
; Подпрограмма для вычисления g(x)
_subcalcul:
    mov ebx, 3
    mul ebx        ; Умножаем x на 3 (g(x) = 3x)
    sub eax, 1     ; Вычитаем 1 (g(x) = 3x - 1)
    ret           ; выход из подпрограммы

```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
 ^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^D Выровнять ^_ К строке

Рис. 4.4: Изменение программы первого листинга

Теперь она вычисляет значение функции для выражения $f(g(x))$ (рис. 4.5).

```

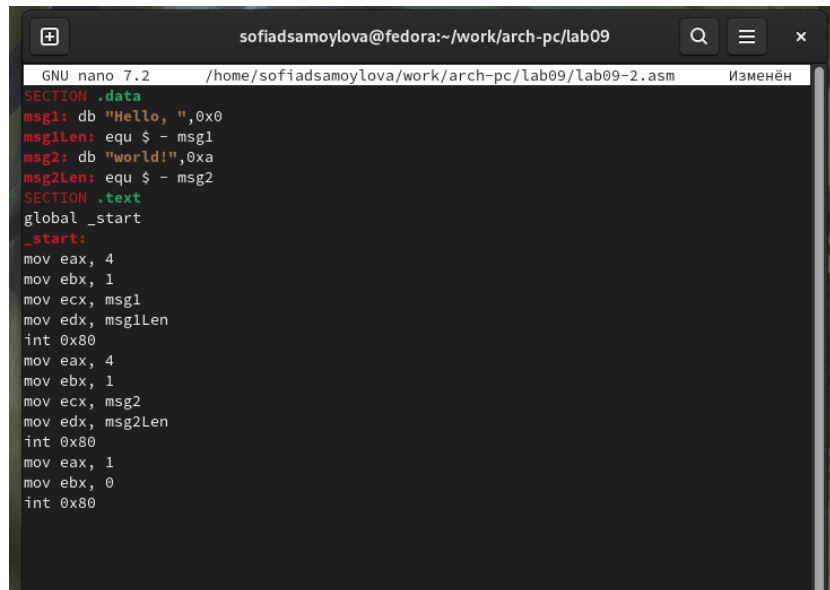
sofiadsamoylova@fedora:~/work/arch-pc/lab09
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ nasm -f elf -l lab09-1.lst lab09-1.asm
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 7
f(g(x)) = 47
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
f(g(x)) = 17
sofiadsamoylova@fedora:~/work/arch-pc/lab09$

```

Рис. 4.5: Работа программы первого листинга

4.2 Отладка программ с помощью GDB

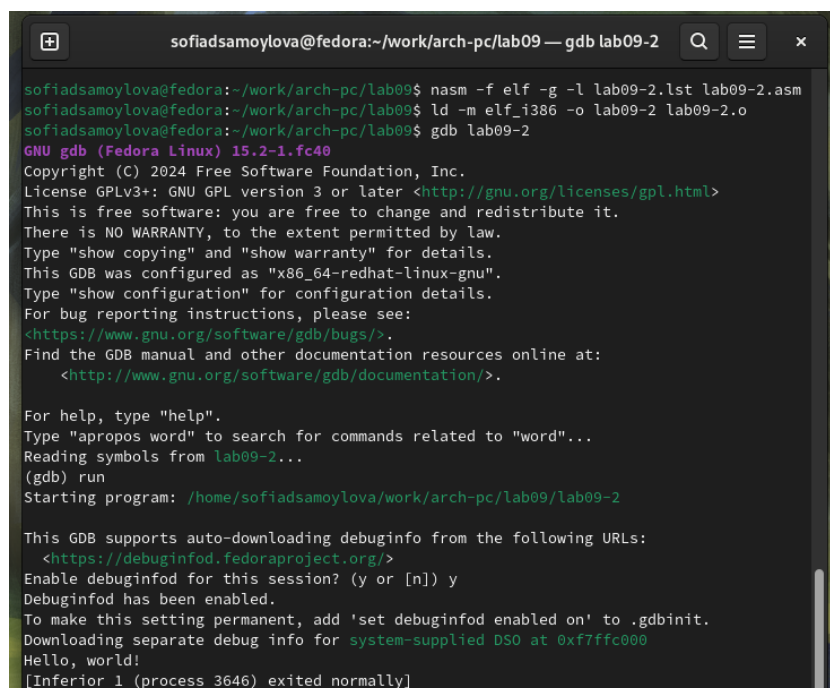
В созданный файл копирую программу второго листинга (рис. 4.6).



```
sofiadsamoylova@fedora:~/work/arch-pc/lab09
GNU nano 7.2 /home/sofiadsamoylova/work/arch-pc/lab09/lab09-2.asm
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. 4.6: Работа программы второго листинга

Транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике (рис. 4.7).



```
sofiadsamoylova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-1.fc40
Copyright (C) 2024 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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
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) run
Starting program: /home/sofiadsamoylova/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 3646) exited normally]
```

Рис. 4.7: Запуск программы в отладчике

Запустив программу командой run, я убедилась в том, что она работает исправ-

но.

Для более подробного анализа программы добавляю *брейкпоинт* на метку `_start` и снова запускаю отладку (рис. 4.8).

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/sofiadsamoylova/work/arch-pc/lab09/lab09-2

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

Рис. 4.8: Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом *Intel* (рис. 4.9).

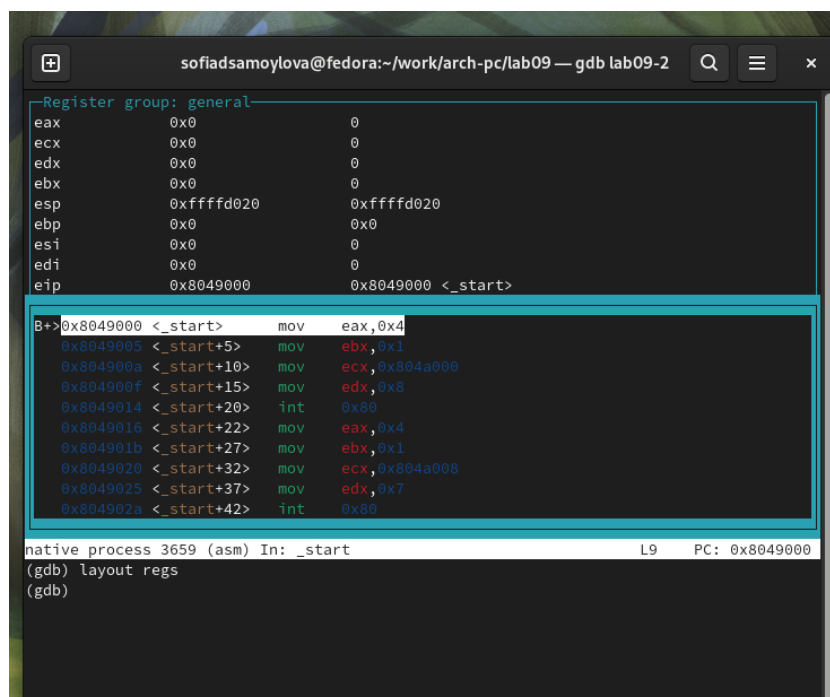
```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:  mov    $0x4,%eax
0x08049005 <+5>:  mov    $0x1,%ebx
0x0804900a <+10>:  mov    $0x804a000,%ecx
0x0804900f <+15>:  mov    $0x8,%edx
0x08049014 <+20>:  int    $0x80
0x08049016 <+22>:  mov    $0x4,%eax
0x0804901b <+27>:  mov    $0x1,%ebx
0x08049020 <+32>:  mov    $0x804a008,%ecx
0x08049025 <+37>:  mov    $0x7,%edx
0x0804902a <+42>:  int    $0x80
0x0804902c <+44>:  mov    $0x1,%eax
0x08049031 <+49>:  mov    $0x0,%ebx
0x08049036 <+54>:  int    $0x80
End of assembler dump.
(gdb) 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. 4.9: Дисассимилирование программы

Различия между синтаксисом *ATT* и *Intel* заключаются в порядке операндов (*ATT* - Операнд источника указан первым. *Intel* - Операнд назначения указан первым), их размере (*ATT* - размер операндов указывается явно с помощью суффиксов, непосредственные операнды представлены символом \$; *Intel* - Размер операндов неявно определяется контекстом, как `ax`, `eax`, непосредственные операнды

пишутся напрямую), имена регистров (*ATT* - имена регистров представлены символом %, *Intel* - имена регистров пишутся без префиксов).

Включаю режим псевдографики для более удобного анализа программы (рис. 4.10).



```
sofiadsamoylova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd020 0xffffd020
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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 3659 (asm) In: _start          L9    PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 4.10: Режим псевдографики

4.3 Добавление точек останова

Проверяю в режиме псевдографики, что *брейкпоинт* сохранился (рис. 4.11).

```

sofiadsamoylova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd020 0xffffd020
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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 3659 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb)

```

Рис. 4.11: Список брейкпоинтов

Устанавливаю еще одну точку останова по адресу инструкции (рис. 4.12).

```

sofiadsamoylova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd020 0xffffd020
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

0x804aa41 add    BYTE PTR [eax],al
0x804aa43 add    BYTE PTR [eax],al
0x804aa45 add    BYTE PTR [eax],al
0x804aa47 add    BYTE PTR [eax],al
0x804aa49 add    BYTE PTR [eax],al
0x804aa4b add    BYTE PTR [eax],al
0x804aa4d add    BYTE PTR [eax],al
0x804aa4f add    BYTE PTR [eax],al
0x804aa51 add    BYTE PTR [eax],al

native process 3659 (asm) In: _start L9 PC: 0x8049000
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x8049000 lab09-2.asm:9
breakpoint already hit 1 time
2        breakpoint keep y  0x8049031 lab09-2.asm:20
(gdb)

```

Рис. 4.12: Добавление второй точки останова

4.4 Работа с данными программы в GDB

Просматриваю содержимое регистров командой `info registers` (рис. 4.13).

```
native process 3659 (asm) In: _start L9 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd020 0xffffd020
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.13: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. 4.14).

```
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "World!\n\034"
(gdb)
```

Рис. 4.14: Просмотр содержимого переменных вторым способом

Меняю содержимое переменных по имени и по адресу (рис. 4.15).

```
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "World!\n\034"
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "xorld!\n\034"
(gdb)
```

Рис. 4.15: Изменение содержимого переменных

Вывожу в различных форматах значение регистра `edx` (рис. 4.16).

```

native process 10469 (asm) In: _start
(gdb) p/t $ecx
$2 = 100000000100101000000000000000
(gdb) p/s $edx
$3 = 8
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb)

```

Рис. 4.16: Просмотр значения регистра разными представлениями

С помощью команды `set` меняю содержимое регистра `ebx` (рис. 4.17).

```

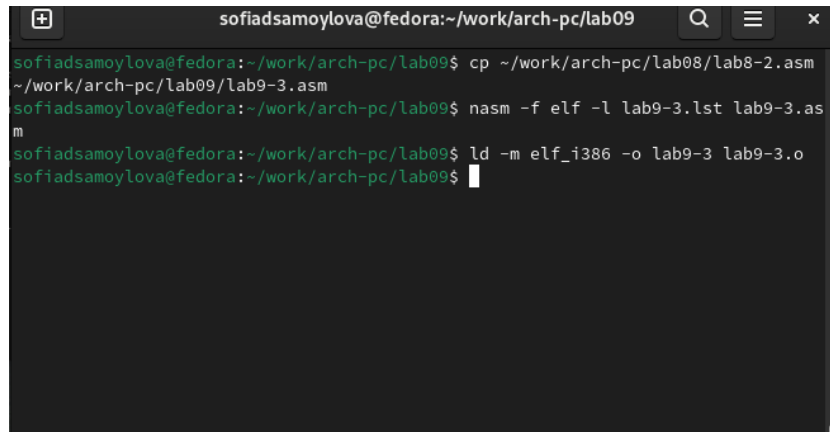
B+ 0x8049000 <_start>      mov     eax,0x4
    0x8049005 <_start+5>    mov     ebx,0x1
    0x804900a <_start+10>   mov     ecx,0x804a000
    0x804900f <_start+15>   mov     edx,0x8
    0x8049014 <_start+20>   int     0x80
>0x8049016 <_start+22>    mov     eax,0x4
    0x804901b <_start+27>   mov     ebx,0x1
native process 10469 (asm) In: _start
(gdb) set $ebx='2'
(gdb) p/s
$6 = 8
(gdb) p/s $ebx
$7 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$8 = 2
(gdb)

```

Рис. 4.17: Примеры использования команды `set`

4.5 Обработка аргументов командной строки в GDB

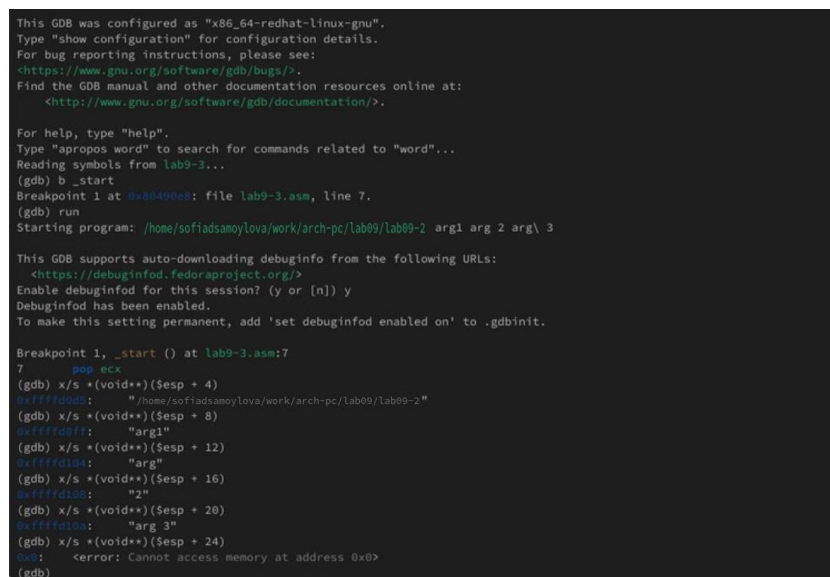
Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. 4.18).

A terminal window titled 'sofiadsamoylova@fedora:~/work/arch-pc/lab09' with search, menu, and close icons. It shows the following commands and their outputs:

```
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm  
~/work/arch-pc/lab09/lab9-3.asm  
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ nasm -f elf -l lab9-3.lst lab9-3.asm  
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o  
sofiadsamoylova@fedora:~/work/arch-pc/lab09$
```

Рис. 4.18: Подготовка новой программы

Запускаю программу в режиме отладки с указанием аргументов, указываю *брейкпоинт* и запускаю отладку. Проверяю работу стека, изменяя аргумент команды просмотра регистра `esp` на `+4`, число обусловлено разрядностью системы, а указатель `void` занимает как раз 4 байта, ошибка при аргументе `+24` означает, что аргументы на вход программы закончились. (рис. 4.19).

A GDB terminal session showing the execution of a program with arguments 'arg1', 'arg2', and 'arg3'. The user sets a breakpoint at the start of the program and runs it. The output shows the stack being checked at various offsets from the current stack pointer. The final error message is 'error: Cannot access memory at address 0x0'.

```
This GDB was configured as "x86_64-redhat-linux-gnu".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
<https://www.gnu.org/software/gdb/bugs/>.  
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 0x00400000: file lab9-3.asm, line 7.  
(gdb) run  
Starting program: /home/sofiadsamoylova/work/arch-pc/lab09/lab09-2 arg1 arg2 arg3  
  
This GDB supports auto-downloading debuginfo from the following URLs:  
<https://debuginfod.fedoraproject.org/>  
Enable debuginfod for this session? (y or [n]) y  
Debuginfod has been enabled.  
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.  
  
Breakpoint 1, _start () at lab9-3.asm:7  
7      pop ecx  
(gdb) x/s *(void+*)($esp + 4)  
0x00400000: "home/sofiadsamoylova/work/arch-pc/lab09/lab09-2"  
(gdb) x/s *(void+*)($esp + 8)  
0x00400008: "arg1"  
(gdb) x/s *(void+*)($esp + 12)  
0x0040000c: "arg2"  
(gdb) x/s *(void+*)($esp + 16)  
0x00400010: "2"  
(gdb) x/s *(void+*)($esp + 20)  
0x00400014: "arg 3"  
(gdb) x/s *(void+*)($esp + 24)  
0x0: <error: Cannot access memory at address 0x0>  
(gdb)
```

Рис. 4.19: Проверка работы стека

4.6 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 4.20).

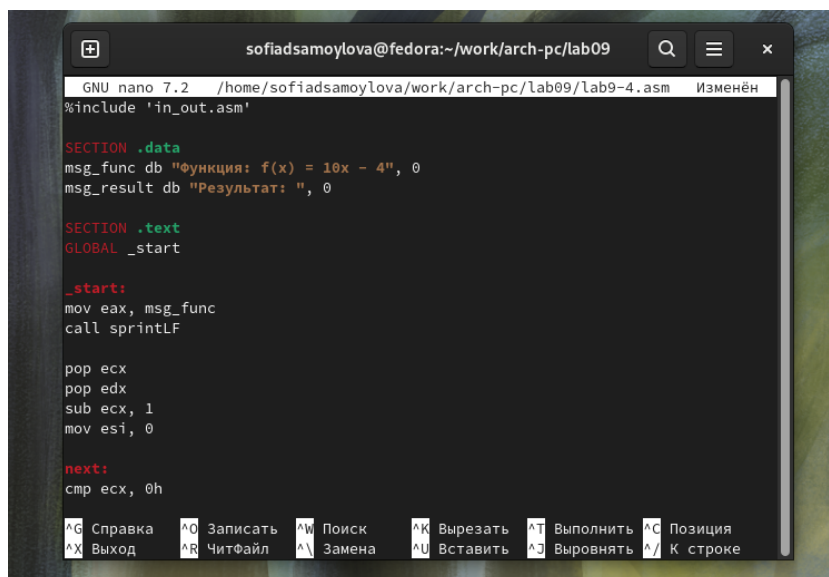


Рис. 4.20: Измененная программа предыдущей лабораторной работы

Код программы:

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10x - 4", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
mov eax, msg_func
call sprintLF

pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
```

```

call sprintf

pop ecx
pop edx
sub ecx, 1
mov esi, 0

next:
cmp ecx, 0h
jz _end
pop eax
call atoi

call _calculate_fx

add esi, eax
loop next

_end:
mov eax, msg_result
call sprintf
mov eax, esi
call iprintLF
call quit

_calculate_fx:
mov ebx, 10
mul ebx
sub eax, 4

```

2. Запускаю программу в режиме отладчика и пошагово через `si` просматриваю изменение значений регистров через `i`. При выполнении инструкции `mul ecx` можно заметить, что результат умножения записывается в регистр `eax`, но также меняет и `edx`. Значение регистра `ebx` не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию (рис. 4.21).

```
Register group: general
eax 0x2 2 ecx 0x4 4
edx 0x0 0 ebx 0x5 5
esp 0xffffcf10 0xbpf 0x0 0
esi 0x0 0 edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17> eflags 0x206 [ PF IF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+ 0x80490e8 <_start> mov 50x1, ebx
0x80490ed <_start+5> mov 50x2, leax
0x80490f2 <_start+10> add leax, ebx
0x80490f9 <_start+17> mov 50x4, ecx
> 0x80490f9 <_start+17> mul ecx
0x80490f0 <_start+19> add 50x5, ebx
0x80490fe <_start+22> mov ebx, edi
0x8049100 <_start+24> mov 50x0x0000, leax
0x8049105 <_start+29> call 0x004000f <sprintf>
0x804910a <_start+34> mov edi, leax

native process 8526 (asm) In: _start L14 PC: 0x80490f9
eax 0x2 2
ecx 0x4 4
edx 0x0 0
ebx 0x5 5
esp 0xffffcf10 0xbpf 0x0 0
ebp 0x0 0
esi 0x0 0
edi 0x0 0
eip 0x80490f9 0x80490f9 <_start+17>
eflags 0x206 [ PF IF ]
cs 0x23 35
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.21: Ошибка программы

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. 4.22).

```
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ nasm -f elf -l lab9-4.lst lab9-4.a
sm
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
sofiadsamoylova@fedora:~/work/arch-pc/lab09$ ./lab9-4
Результат: 25
sofiadsamoylova@fedora:~/work/arch-pc/lab09$
```

Рис. 4.22: Проверка корректировок в программе

Код измененной программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
div: DB 'Результат: ', 0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov ebx, 3
```

```
mov eax, 2
```

```
add ebx, eax
```

```
mov eax, ebx
```

```
mov ecx, 4
```

```
mul ecx
```

```
add eax, 5
```

```
mov edi, eax
```

```
mov eax, div
```

```
call sprint
```

```
mov eax, edi
```

```
call iprintLF
```

```
call quit
```

5 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм, а так же познакомилась с методами отладки при помощи *GDB* и его основными возможностями.

Список литературы

1. Курс на ТУИС
2. Лабораторная работа №9