

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Стещенко Артём Сергеевич

Содержание

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. 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.

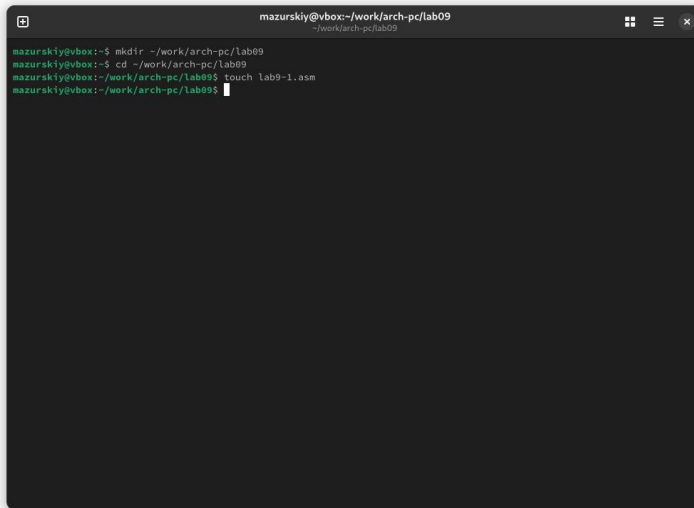
Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске

программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

4 Выполнение лабораторной работы

4.1 Релазация подпрограмм в NASM

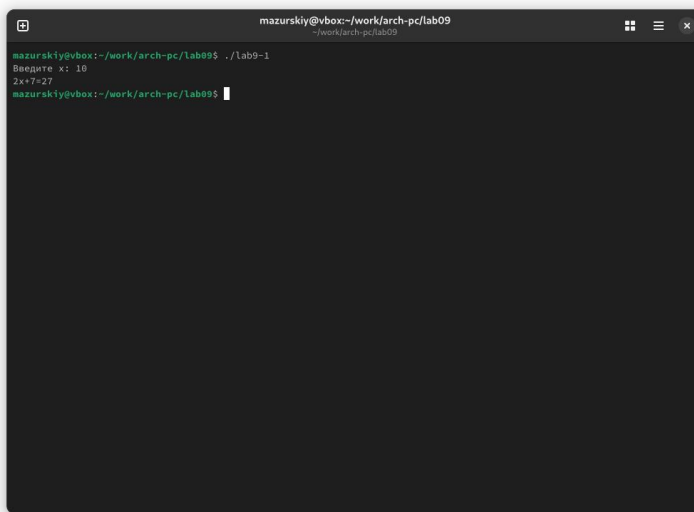
Создаю каталог для выполнения лабораторной работы №9 (рис. 1).



```
mazurskiy@vbox:~/work/arch-pc/lab09
~/work/arch-pc/lab09
mazurskiy@vbox:~$ mkdir -p ~/work/arch-pc/lab09
mazurskiy@vbox:~$ cd ~/work/arch-pc/lab09
mazurskiy@vbox:~/work/arch-pc/lab09$ touch lab9-1.asm
mazurskiy@vbox:~/work/arch-pc/lab09$
```

Рис. 1: Создание рабочего каталога

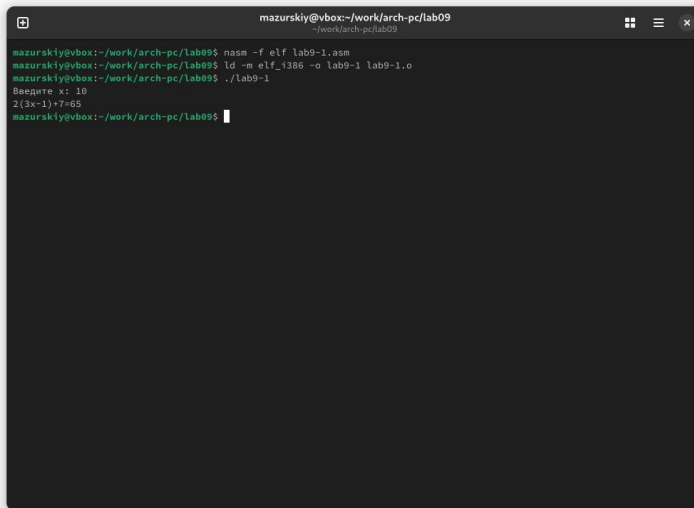
Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. 2).



```
mazurskiy@vbox:~/work/arch-pc/lab09
mazurskiy@vbox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2x+7=27
mazurskiy@vbox:~/work/arch-pc/lab09$
```

Рис. 2: Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения $f(g(x))$ (рис. 3).



```
mazurskiy@vbox:~/work/arch-pc/lab09
mazurskiy@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
mazurskiy@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
mazurskiy@vbox:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 10
2(3x-1)+7=65
mazurskiy@vbox:~/work/arch-pc/lab09$
```

Рис. 3: Изменение программы первого листинга

Код программы:

```
%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
```

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

```
call quit
```

```
_calcul:
push eax
call _subcalcul
```

```
mov ebx, 2
mul ebx
add eax, 7
```

```
mov [res], eax
pop eax
ret
```

```
_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret
```

4.1.1 Отладка программ с помощью GDB

В созданный файл копирую программу второго листинга, транслирую с созданием файла листинга и отладки, компоную и запускаю в отладчике (рис. 4).

```
mazurskiy@vbox: ~/work/arch-pc/lab09 — gdb lab9-2
mazurskiy@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
mazurskiy@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
mazurskiy@vbox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.2-3.fc41
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 lab9-2...
(gdb)
```

Рис. 4: Запуск программы в отладчике

Запустив программу командой `run`, я убедился в том, что она работает исправно (рис. 5).

```
mazurskiy@vbox: ~/work/arch-pc/lab09 — gdb lab9-2
mazurskiy@vbox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
mazurskiy@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
mazurskiy@vbox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.2-3.fc41
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 lab9-2...
(gdb) run
Starting program: /home/mazurskiy/work/arch-pc/lab09/lab9-2

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.
Hello, World!
[Inferior 1 (process 5871) exited normally]
(gdb)
```

Рис. 5: Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку `_start` и снова запускаю отладку (рис. 6).

```
mazurskiy@vbox: ~/work/arch-pc/lab09 — gdb lab9-2
mazurskiy@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-2 lab9-2.o
mazurskiy@vbox:~/work/arch-pc/lab09$ gdb lab9-2
GNU gdb (Fedora Linux) 15.2-3.fc41
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:
<http://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 lab9-2...
(gdb) run
Starting program: /home/mazurskiy/work/arch-pc/lab09/lab9-2

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.
Hello, World!
[Inferior 1 (process 5871) exited normally]
(gdb) break _start
Breakpoint 1 at 0x00400000: file lab9-2.asm, line 11.
(gdb) run
Starting program: /home/mazurskiy/work/arch-pc/lab09/lab9-2

Breakpoint 1, _start () at lab9-2.asm:11
11      mov eax, 4
(gdb)
```

Рис. 6: Запуск отладчика с брейкпоинтом

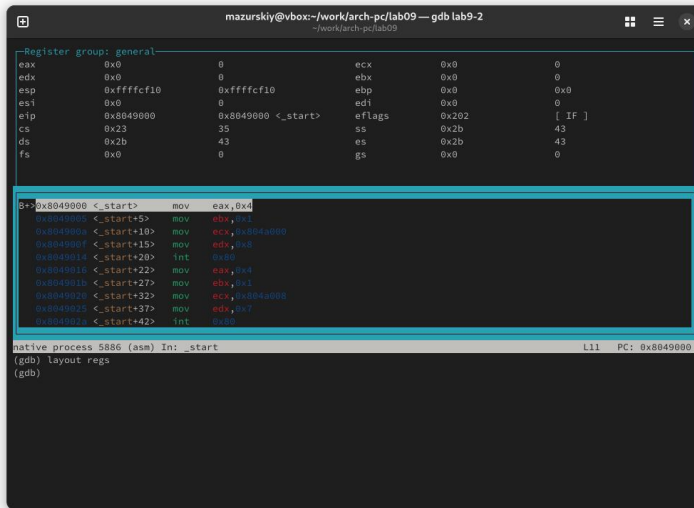
Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel amd топчик (рис. 7).

Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом \$; Intel - Размер операндов неявно определяется контекстом, как ax, eax, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом %, Intel - имена регистров пишутся без префиксов).

```
mazurskiy@vbox: ~/work/arch-pc/lab09 — gdb lab9-2
Breakpoint 1, _start () at lab9-2.asm:11
11      mov eax, 4
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x00400000 <+0>: mov     $0x4,%eax
0x00400005 <+5>: mov     $0x1,%eax
0x0040000a <+10>: mov     $0x0,%eax
0x0040000f <+15>: mov     $0x0,%eax
0x00400014 <+20>: int     $0x0
0x00400019 <+25>: mov     $0x4,%eax
0x0040001e <+30>: mov     $0x1,%eax
0x00400023 <+35>: mov     $0x0,%eax
0x00400028 <+40>: mov     $0x0,%eax
0x0040002d <+45>: int     $0x0
0x00400032 <+50>: mov     $0x4,%eax
0x00400037 <+55>: mov     $0x1,%eax
0x0040003c <+60>: int     $0x0
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x00400000 <+0>: mov     eax,%eax
0x00400005 <+5>: mov     ebx,%ebx
0x0040000a <+10>: mov     ecx,%ecx
0x0040000f <+15>: mov     edx,%edx
0x00400014 <+20>: int     $0x0
0x00400019 <+25>: mov     eax,%eax
0x0040001e <+30>: mov     ebx,%ebx
0x00400023 <+35>: mov     ecx,%ecx
0x00400028 <+40>: mov     edx,%edx
0x0040002d <+45>: int     $0x0
0x00400032 <+50>: mov     eax,%eax
0x00400037 <+55>: mov     ebx,%ebx
0x0040003c <+60>: int     $0x0
End of assembler dump.
(gdb)
```

Рис. 7: Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. 8).



```
mazurskiy@vbox:~/work/arch-pc/lab09 — gdb lab9-2
~/work/arch-pc/lab09

Register group: general
eax    0x0      0      ecx    0x0      0
edx    0x0      0      ebx    0x0      0
esp    0xffffcf10 0xffffcf10 ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x00490000 0x00490000 <_start> eflags 0x202    [ IF ]
cs     0x23     35      ss     0x2b     43
ds     0x2b     43      es     0x2b     43
fs     0x0      0      gs     0x0      0

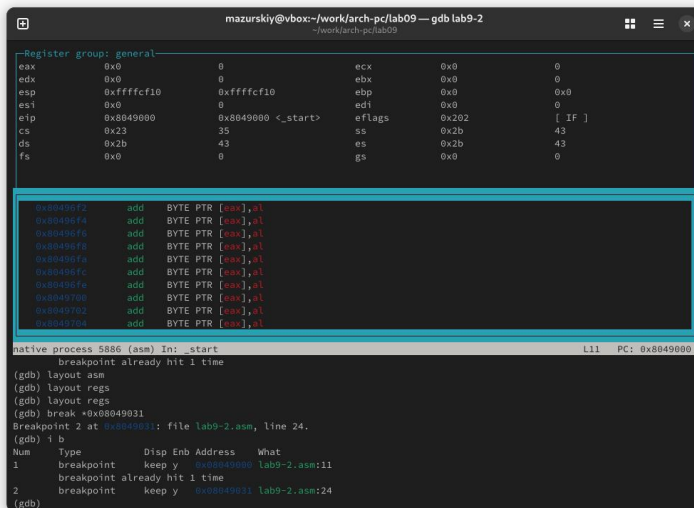
0x00490000 <_start> mov    eax,0x4
0x00490005 <_start+5> mov    ecx,0x1
0x0049000a <_start+10> mov    ecx,0x004a0000
0x0049000f <_start+15> mov    edx,0x0
0x0049001c <_start+20> int     0x0
0x0049001b <_start+22> mov    eax,0x4
0x0049001b <_start+27> mov    ebx,0x1
0x00490020 <_start+32> mov    ecx,0x004a0000
0x00490025 <_start+37> mov    edx,0x7
0x0049002a <_start+42> int     0x0

native process 5886 (asm) In: _start      L11  PC: 0x00490000
(gdb) layout regs
(gdb)
```

Рис. 8: Режим псевдографики

4.1.2 Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. 9).



```
mazurskiy@vbox:~/work/arch-pc/lab09 — gdb lab9-2
~/work/arch-pc/lab09

Register group: general
eax    0x0      0      ecx    0x0      0
edx    0x0      0      ebx    0x0      0
esp    0xffffcf10 0xffffcf10 ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x00490000 0x00490000 <_start> eflags 0x202    [ IF ]
cs     0x23     35      ss     0x2b     43
ds     0x2b     43      es     0x2b     43
fs     0x0      0      gs     0x0      0

0x004900f2 add    BYTE PTR [eax],al
0x004900f4 add    BYTE PTR [eax],al
0x004900f6 add    BYTE PTR [eax],al
0x004900f8 add    BYTE PTR [eax],al
0x004900fa add    BYTE PTR [eax],al
0x004900fc add    BYTE PTR [eax],al
0x004900fe add    BYTE PTR [eax],al
0x00490100 add    BYTE PTR [eax],al
0x00490102 add    BYTE PTR [eax],al
0x00490104 add    BYTE PTR [eax],al

native process 5886 (asm) In: _start      L11  PC: 0x00490000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x00490031: file lab9-2.asm, line 24.
Breakpoint 2 at 0x00490031: file lab9-2.asm, line 24.
(gdb) i b
Num   Type      Disp Enb Address      What
1     breakpoint keep y 0x00490031 lab9-2.asm:11
      breakpoint already hit 1 time
2     breakpoint keep y 0x00490031 lab9-2.asm:24
(gdb)
```

Рис. 9: Список брейкпоинтов

Устанавливаю еще одну точку останова по адресу инструкции (рис. 10).

```

mzaszkiy@vbox:~/work/arch-pc-lab09 — gdb lab9-2
~/work/arch-pc-lab09

Register group: general
eax    0x0      0      ecx    0x0      0
edx    0x0      0      ebx    0x0      0
esp    0xffffcf10  0      ebp    0x0      0
esi    0x0      0      edi    0x0      0
eip    0x8049080  0x8049080 <_start>  eflags  0x202      [ IF ]
cs     0x23     35     ss     0x2b     43
ds     0x2b     43     es     0x2b     43
fs     0x0      0      gs     0x0      0

0x00400f2f add    BYTE PTR [eax],al
0x00400f4f add    BYTE PTR [eax],al
0x00400f6f add    BYTE PTR [eax],al
0x00400f8f add    BYTE PTR [eax],al
0x00400faf add    BYTE PTR [eax],al
0x00400fcf add    BYTE PTR [eax],al
0x00400fef add    BYTE PTR [eax],al
0x00400f0e add    BYTE PTR [eax],al
0x00400f2d add    BYTE PTR [eax],al
0x00400f4d add    BYTE PTR [eax],al

native process 5886 (asm) In: start
Breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x00400f01
breakpoint 2 at 0x00400f01: file lab9-2.asm, line 24.
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint         keep y 0x00400f00 lab9-2.asm:11
      breakpoint already hit 1
2      breakpoint         keep y 0x00400f01 lab9-2.asm:24
(gdb)

```

Рис. 10: Добавление второй точки останова

4.1.3 Работа с данными программы в GDB

Просматриваю содержимое регистров командой info registers (рис. 11).

```

mazarisky@vbox:~/work/arch-pc/lab09 -- gdb lab9-2
~/work/arch-pc/lab09

Register group: general
eax    0x8      8      ecx    0x804a000    134520832
edx    0x8      8      ebx    0x1      1
esp    0xffffcf10 0xffffcf10  ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x8049016 0x8049016 <_start+22>  eflags  0x202     [ IF ]
cs     0x23     35     ss     0x2b     43
ds     0x2b     43     es     0x2b     43
fs     0x0      0      gs     0x0      0

0x8049000 <_start>: mov     ebx,ebx
0x8049005 <_start+5>: mov     ebx,ebx
0x804900a <_start+10>: mov     ecx,0x804a000
0x804900f <_start+15>: mov     edx,ebx
0x8049014 <_start+20>: int     0x80
0x8049016 <_start+22>: mov     eax,0x4
0x8049017 <_start+23>: mov     ebx,ebx
0x8049020 <_start+26>: mov     ecx,0x804a000
0x8049025 <_start+31>: mov     edx,ebx
0x804902a <_start+36>: int     0x80

native process 5886 (asm) In: _start
L17: PC: 0x8049016
eax    0x8      8
ecx    0x804a000 134520832
edx    0x8      8
ebx    0x1      1
esp    0xffffcf10 0xffffcf10
ebp    0x0      0x0
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--

```

Рис. 11: Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. 12).


```

mazurskiy@vbox:~/work/arch-pc/lab09 — gdb lab9-2
~/work/arch-pc/lab09

--Register group: general
eax      0x8      8      ecx      0x804a000      134528832
edx      0x8      8      ebx      0x1      1
esp      0xffffcf10      0xffffcf10      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
fs       0x0      0      gs       0x0      0

0* 0x8049000 <_start>      mov     eax,edx
0x8049005 <_start+5>      mov     ebx,edi
0x804900a <_start+10>     mov     ecx,0x804a000
0x804900f <_start+15>     mov     edx,0x8
0x8049014 <_start+20>     int     0x0
> 0x8049016 <_start+22>     mov     eax,0x4
0x804901b <_start+27>     mov     ebx,edi
0x8049020 <_start+32>     mov     ecx,0x804a008
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x0

native process 5886 (asm) In: _start      L17  PC: 0x8049016
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/15b &msg1
0x804a000: "Hello, "
(gdb) x/15b 0x804a008
0x804a008: "World!\n034"
(gdb)

```

Рис. 12: Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. 13).

```

mazurskiy@vbox:~/work/arch-pc/lab09 — gdb lab9-2
~/work/arch-pc/lab09

--Register group: general
eax      0x8      8      ecx      0x804a000      134528832
edx      0x8      8      ebx      0x1      1
esp      0xffffcf10      0xffffcf10      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      eflags      0x202      [ IF ]
cs       0x23      35      ss       0x2b      43
ds       0x2b      43      es       0x2b      43
fs       0x0      0      gs       0x0      0

0* 0x8049000 <_start>      mov     eax,edx
0x8049005 <_start+5>      mov     ebx,edi
0x804900a <_start+10>     mov     ecx,0x804a000
0x804900f <_start+15>     mov     edx,0x8
0x8049014 <_start+20>     int     0x0
> 0x8049016 <_start+22>     mov     eax,0x4
0x804901b <_start+27>     mov     ebx,edi
0x8049020 <_start+32>     mov     ecx,0x804a008
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x0

native process 5886 (asm) In: _start      L17  PC: 0x8049016
0x804a000 <msg1>: "Hello, "
(gdb) x/15b 0x804a008
0x804a008: "World!\n034"
0x804a008 <msg2>: "World!\n034"
(gdb) set (char)&msg1='\n'
"msg1" has unknown type; cast it to its declared type
(gdb) set (char)&msg1='\n'
(gdb) x/15b &msg1
0x804a000 <msg1>: "hello, "
(gdb) set (char)&msg2='x'
(gdb) x/15b &msg2
0x804a008 <msg2>: "xorld!\n034"
(gdb)

```

Рис. 13: Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. 14).

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd070 0xffffd070
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 10469 (asm) In: _start      L15      PC: 0x8049016
(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. 14: Просмотр значения регистра разными представлениями

С помощью команды `set` меняю содержимое регистра `ebx` (рис. 15).

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd070 0xffffd070
ebp      0x0      0x0
esi      0x0      0

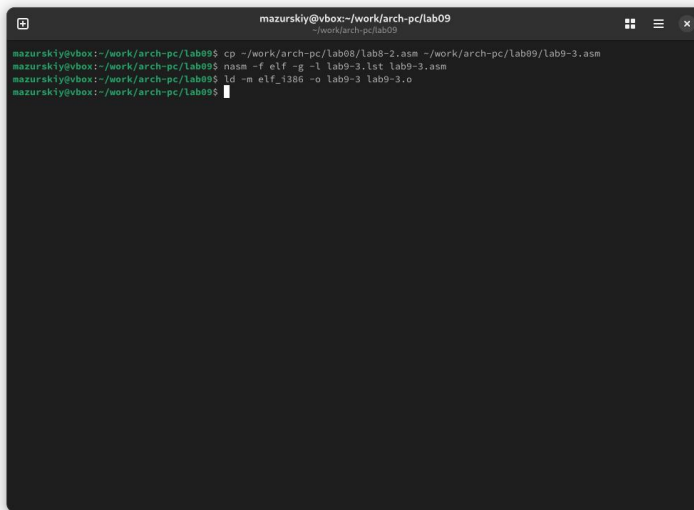
B+ 0x8049000 <_start>      mov     eax,0x4
    0x8049005 <_start+5>   mov     ebx,0x1
    0x804900a <_start+10>  mov     ecx,0x804a000
    0x804900f <_start+15>  mov     edx,0x8
    0x8049014 <_start+20>  int     0x80
>0x8049016 <_start+22>    mov     eax,0x4
0x804901b <_start+27>    mov     ebx,0x1

native process 10469 (asm) In: _start      L15      PC: 0x8049016
(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. 15: Примеры использования команды `set`

4.1.4 Обработка аргументов командной строки в GDB

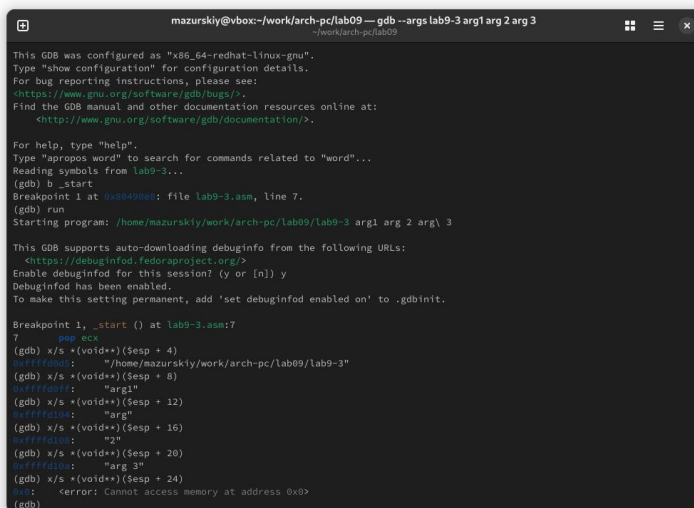
Копирую программу из предыдущей лабораторной работы в текущий каталог и создаю исполняемый файл с файлом листинга и отладки (рис. 16).



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

Рис. 16: Подготовка новой программы

Запускаю программу с 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. 17).



```
mazurskiy@vbox:~/work/arch-pc/lab09$ gdb --args lab9-3 arg1 arg2 arg3
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/mazurskiy/work/arch-pc/lab09/lab9-3 arg1 arg2 arg3

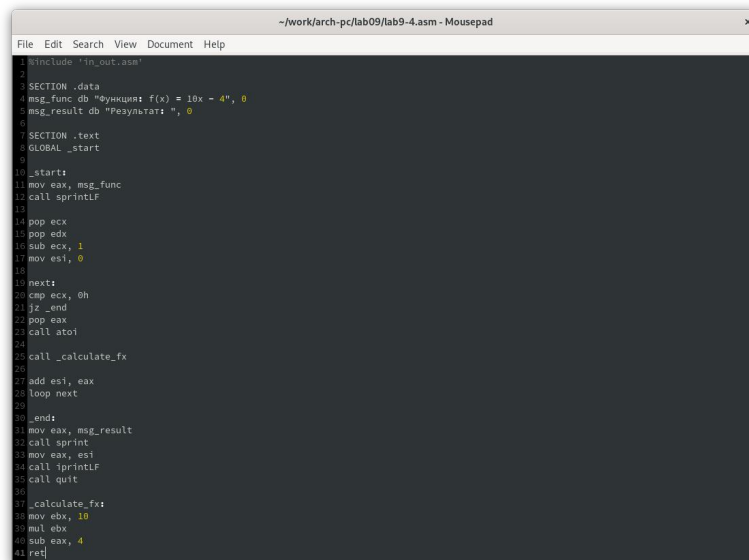
This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab9-3.asm:7
7       pop ecx
(gdb) x/s *(void++)($esp + 4)
0xfffff000: "/home/mazurskiy/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void++)($esp + 8)
0xfffff000: "arg1"
(gdb) x/s *(void++)($esp + 12)
0xfffff000: "arg"
(gdb) x/s *(void++)($esp + 16)
0xfffff000: ""
(gdb) x/s *(void++)($esp + 20)
0xfffff000: "arg 3"
(gdb) x/s *(void++)($esp + 24)
0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 17: Проверка работы стека

4.2 Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. 18).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg_func db "Функция: f(x) = 10x - 4", 0
5 msg_result db "Результат: ", 0
6
7 SECTION .text
8 GLOBAL _start
9
10 _start:
11 mov eax, msg_func
12 call sprintf
13
14 pop ecx
15 pop edx
16 sub ecx, 1
17 mov esi, 0
18
19 next:
20 cmp ecx, 9h
21 jz _end
22 pop eax
23 call atoi
24
25 call _calculate_fx
26
27 add esi, eax
28 loop next
29
30 _end:
31 mov eax, msg_result
32 call sprintf
33 mov eax, esi
34 call sprintf
35 call quit
36
37 _calculate_fx:
38 mov ebx, 10
39 mul ebx
40 sub eax, 4
41 ret
```

Рис. 18: 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 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
```

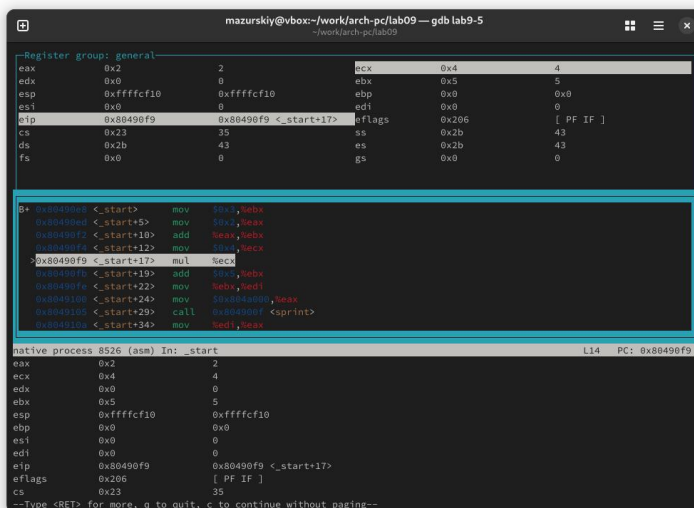
_end:

```
mov eax, msg_result
call sprint
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 r. При выполнении инструкции mul esx 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. 19).



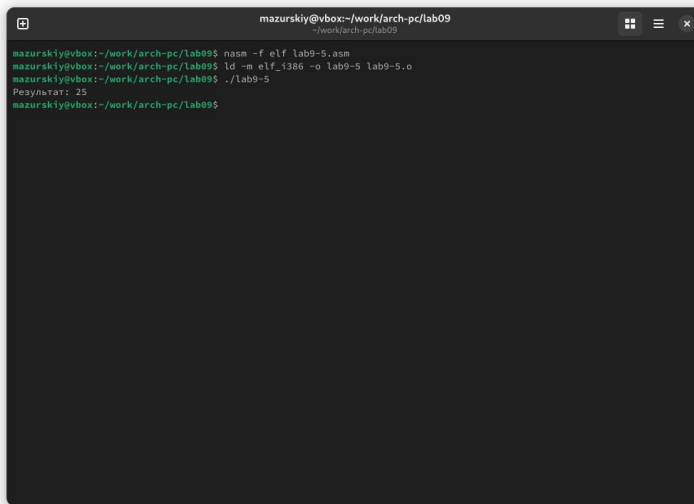
```
mazurskiy@vbox:~/work/arch-pc/lab09 — gdb lab9-5
--Register group: general--
eax 0x2      2      ecx 0x4      4
edx 0x0      0      ebx 0x5      5
esp 0xffffcf10 0xffffcf10 ebp 0x0      0x0
esi 0x0      0      edi 0x0      0
eip 0x80490f9 0x80490f9 <_start+17> eflags 0x206    [ PF IF ]
cs 0x23     35     ss 0x2b     43
ds 0x2b     43     es 0x2b     43
fs 0x0      0      gs 0x0      0

0x80490e0 <_start> mov  %eax,%eax
0x80490e5 <_start+5> mov  %eax,%eax
0x80490f2 <_start+10> add  %eax,%ebx
0x80490f4 <_start+12> mov  %eax,%ecx
0x80490f9 <_start+17> mul  %ecx
0x80490fd <_start+21> add  %eax,%ebx
0x80490fe <_start+22> mov  %ebx,%eax
0x8049100 <_start+24> mov  %eax,%eax
0x8049102 <_start+26> call 0x80490ef <sprint>
0x8049104 <_start+28> mov  %edi,%eax

native process 8526 (asm) In: _start L14 PC: 0x80490f9
eax 0x2      2
ecx 0x4      4
edx 0x0      0
ebx 0x5      5
esp 0xffffcf10 0xffffcf10
ebp 0x0      0x0
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. 19: Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. 20).

A terminal window with a dark background. The title bar shows 'mazurskiy@vbox: ~/work/arch-pc/lab09'. The terminal content shows the following commands and output:

```
mazurskiy@vbox:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
mazurskiy@vbox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
mazurskiy@vbox:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
mazurskiy@vbox:~/work/arch-pc/lab09$
```

Рис. 20: 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 приобрел навыки написания программ с использованием подпрограмм, а так же познакомился с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №9
3. Программирование на языке ассемблера NASM Столяров А. В.