

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Мазурский Александр Дмитриевич

Содержание

Цель работы

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

Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Самостоятельное выполнение заданий по материалам лабораторной работы

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.

Выполнение лабораторной работы

Релазация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы №9 (рис. -@fig:001).

```
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ mkdir -p /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ cd -p /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ touch lab09-1.asm
```

Создание рабочего каталога

Копирую в файл код из листинга, компилирую и запускаю его, данная программа выполняет вычисление функции (рис. -@fig:002).

```
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf lab09-1.asm
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./lab09-1
Введите x: 15
2x+7=37
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$
```

Запуск программы из листинга

Изменяю текст программы, добавив в нее подпрограмму, теперь она вычисляет значение функции для выражения $f(g(x))$ (рис. -@fig:003).

```
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf lab09-1.asm
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./lab09-1
Введите x: 15
2(3x-1)+7=95
morissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$
```

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
```

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. -@fig:004).

```
morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
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-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) 
```

Запуск программы в отладчике

Запустив программу командой run, я убедился в том, что она работает исправно (рис. -@fig:005).

```
morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
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-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/morissaladonzo/work/study/2024-2025/Архитектура компьютера/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.
Hello, world!
Inferior 1 (process 5040) exited normally
(gdb) 
```

Проверка программы отладчиком

Для более подробного анализа программы добавляю брейкпоинт на метку _start и снова запускаю отладку (рис. -@fig:006).

```

morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
morissaladonzo@vbox: /work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
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-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/morissaladonzo/work/study/2024-2025/Архитектура компьютера/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.
Hello, world!
[Inferior 1 (process 5040) exited normally]
(gdb) break _start
Breakpoint 1 at 0x00400000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/morissaladonzo/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)

```

Запуск отладчика с брейкпоинтом

Далее смотрю дисассимилированный код программы, перевожу на команд с синтаксисом Intel *амд топчик* (рис. -@fig:007).

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).

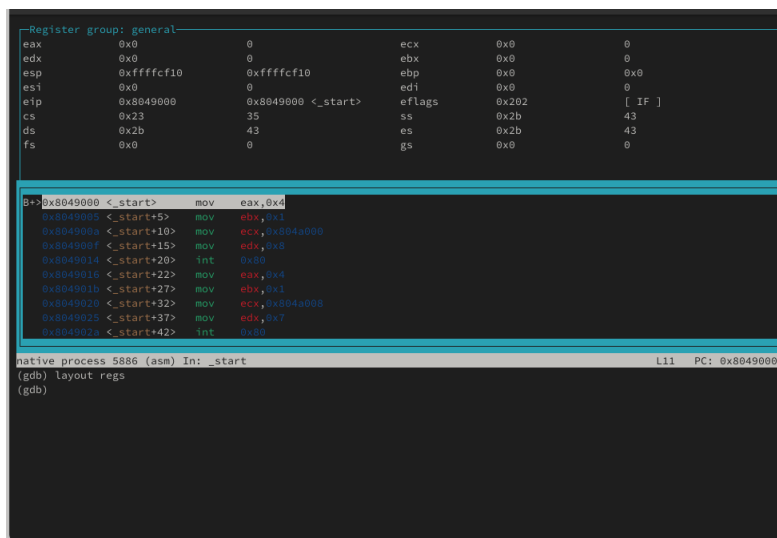
```

(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) █

```

Дисассимилирование программы

Включаю режим псевдографики для более удобного анализа программы (рис. -@fig:008).



Режим псевдографики

Добавление точек останова

Проверяю в режиме псевдографики, что брейкпоинт сохранился (рис. - @fig:009).

```
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      0x8049000 0x8049000 <_start>  eflags   0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

0x80490f2  add  BYTE PTR [eax],al
0x80490f4  add  BYTE PTR [eax],al
0x80490f6  add  BYTE PTR [eax],al
0x80490f8  add  BYTE PTR [eax],al
0x80490fa  add  BYTE PTR [eax],al
0x80490fc  add  BYTE PTR [eax],al
0x80490fe  add  BYTE PTR [eax],al
0x8049700  add  BYTE PTR [eax],al
0x8049702  add  BYTE PTR [eax],al
0x8049704  add  BYTE PTR [eax],al

native process 5886 (asm) In: _start      L11  PC: 0x8049000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num  Type      Disp Enb Address  What
1    breakpoint keep y  0x8049000 lab9-2.asm:11
    breakpoint already hit 1 time
2    breakpoint keep y  0x8049031 lab9-2.asm:24
(gdb)
```

Список брейкпоинтов

Устанавливаю еще одну точку останова по адресу инструкции (рис. -@fig:010).

```
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      0x8049000 0x8049000 <_start>  eflags   0x202    [ IF ]
cs       0x23     35      ss       0x2b     43
ds       0x2b     43      es       0x2b     43
fs       0x0      0      gs       0x0      0

0x80490f2  add  BYTE PTR [eax],al
0x80490f4  add  BYTE PTR [eax],al
0x80490f6  add  BYTE PTR [eax],al
0x80490f8  add  BYTE PTR [eax],al
0x80490fa  add  BYTE PTR [eax],al
0x80490fc  add  BYTE PTR [eax],al
0x80490fe  add  BYTE PTR [eax],al
0x8049700  add  BYTE PTR [eax],al
0x8049702  add  BYTE PTR [eax],al
0x8049704  add  BYTE PTR [eax],al

native process 5886 (asm) In: _start      L11  PC: 0x8049000
breakpoint already hit 1 time
(gdb) layout asm
(gdb) layout regs
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 24.
(gdb) i b
Num  Type      Disp Enb Address  What
1    breakpoint keep y  0x8049000 lab9-2.asm:11
    breakpoint already hit 1 time
2    breakpoint keep y  0x8049031 lab9-2.asm:24
(gdb)
```

Добавление второй точки останова

Работа с данными программы в GDB

Просматриваю содержимое регистров командой info registers (рис. -@fig:011).

```
-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

B+ 0x8049000 <_start>      mov     eax,0x4
0x8049005 <_start+5>      mov     ebx,0x1
0x804900a <_start+10>     mov     ecx,0x804a000
0x804900f <_start+15>     mov     edx,0x0
0x8049014 <_start+20>     int     0x80
>0x8049016 <_start+22>     mov     eax,0x4
0x804901b <_start+27>     mov     ebx,0x1
0x8049020 <_start+32>     mov     ecx,0x804a000
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     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--
```

Просмотр содержимого регистров

Смотрю содержимое переменных по имени и по адресу (рис. -@fig:012).

```
-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

B+ 0x8049000 <_start>      mov     eax,0x4
0x8049005 <_start+5>      mov     ebx,0x1
0x804900a <_start+10>     mov     ecx,0x804a000
0x804900f <_start+15>     mov     edx,0x0
0x8049014 <_start+20>     int     0x80
>0x8049016 <_start+22>     mov     eax,0x4
0x804901b <_start+27>     mov     ebx,0x1
0x8049020 <_start+32>     mov     ecx,0x804a000
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x80

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

Просмотр содержимого переменных двумя способами

Меняю содержимое переменных по имени и по адресу (рис. -@fig:013).


```

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

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,0x804a000
0x8049025 <_start+37>      mov     edx,0x7
0x804902a <_start+42>      int     0x80

native process 5886 (asm) In: _start      L17      PC: 0x8049016
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
0x804a008 <msg1>:      "hello, "
(gdb) set {char}&msg2='x'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "xorld!\n\034"
(gdb)

```

Изменение содержимого переменных двумя способами

Вывожу в различных форматах значение регистра edx (рис. -@fig:014).

```

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

```

Просмотр значения регистра разными представлениями

С помощью команды set меняю содержимое регистра ebx (рис. -@fig:015).

```
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    0x60
>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)
```

Примеры использования команды set

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. - @fig:016).

```
matissaladonov@home:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ cp ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab08/lab08-2.asm ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab09-3.asm
matissaladonov@home:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
matissaladonov@home:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
```

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. - @fig:017).

```

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 0x90490e0: file lab9-3.asm, line 7.
(gdb) run
Starting program: /home/mazurskiy/work/arch-pc/lab09/lab9-3 arg1 arg2 arg\ 3

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)
0xffffd000: "/home/mazurskiy/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd00f: "arg1"
(gdb) x/s *(void**)(esp + 12)
0xffffd014: "arg"
(gdb) x/s *(void**)(esp + 16)
0xffffd018: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd01d: "arg 3"
(gdb) x/s *(void**)(esp + 24)
0x: <error: Cannot access memory at address 0x0>
(gdb)

```

Проверка работы стека

Задание для самостоятельной работы

1. Меняю программу самостоятельной части предыдущей лабораторной работы с использованием подпрограммы (рис. -@fig:018).

```

Ouvrir ▾  lab9-4.asm
~/work/study/2024-2025/Архитектура компьютера/018

%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 sprintf
mov eax, esi
call iprintf
call quit

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

```

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 sprintfLF

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

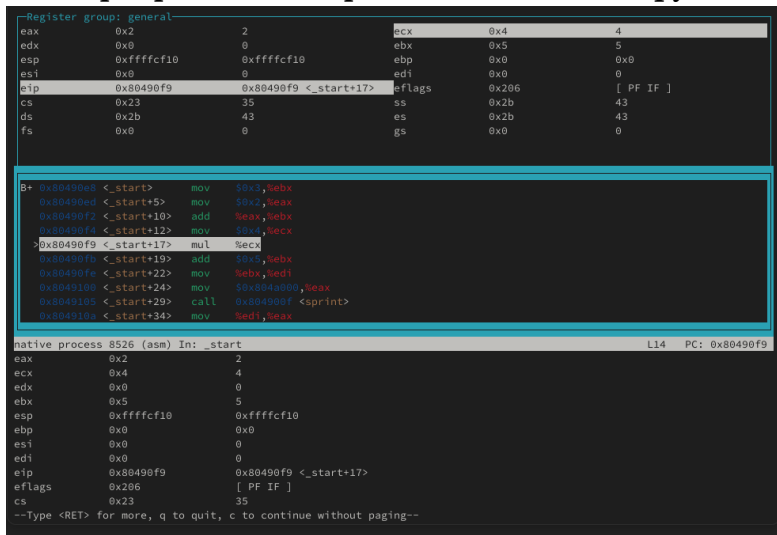
_end:
mov eax, msg_result
call sprintf
mov eax, esi
call sprintfLF
call quit

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

```

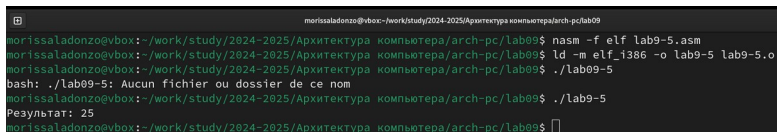
2. Запускаю программу в режиме отладчика и пошагово через si просматриваю изменение значений регистров через i r. При выполнении инструкции mul ecx можно заметить, что результат

умножения записывается в регистр `eax`, но также меняет и `edx`.
Значение регистра `ebx` не обновляется напрямую, поэтому результат программа неверно подсчитывает функцию (рис. -@fig:019).



Поиск ошибки в программе через пошаговую отладку

Исправляю найденную ошибку, теперь программа верно считает значение функции (рис. -@fig:020).



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
```

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.

Список литературы

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