

Отчет по лабораторной работе No.9

Дисциплины: Архитектура компьютера

Нджову Нелиа

Содержание

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

Список иллюстраций

3.1	Рис 1	7
3.2	Рис 2	8
3.3	Рис 3	8
3.4	Рис 4	9
3.5	Рис 5	9
3.6	Рис 6	10
3.7	Рис 7	10
3.8	Рис 8	11
3.9	Рис 9	11
3.10	Рис 10	11
3.11	Рис 11	12
3.12	Рис 12	12
3.13	Рис 13	13
3.14	Рис 14	13
3.15	Рис 15	13
3.16	Рис 16	14
3.17	Рис 17	14
3.18	Рис 18	14
3.19	Рис 19	15
3.20	Рис 20	15
3.21	Рис 21	15
3.22	Рис 22	15
3.23	Рис 23	16
3.24	Рис 24	16
3.25	Рис 25	16
3.26	Рис 26	16
3.27	Рис 27	17
3.28	Рис 28	17
3.29	Рис 29	18
3.30	Рис 30	18
3.31	Рис 31	18
3.32	Рис 32	19
3.33	Рис 33	19
3.34	Рис 34	20
3.35	Рис 35	20
3.36	Рис 36	21

Список таблиц

1 Цель работы

Цель лабораторной работы – приобретение навыков написания программ с использованием подпрограмм. Введение в методы отладки с использованием GDB и его основные возможности.

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Задание для самостоятельной работы

3 Выполнение лабораторной работы

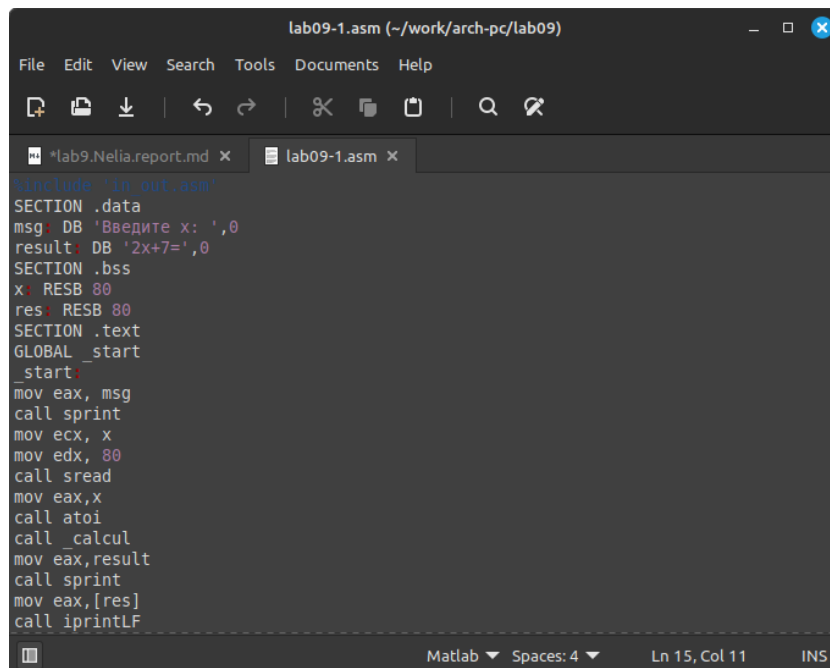
1. Реализация подпрограмм в NASM

Я создам каталог для lab09, зайду в него и создам файл lab09-1.asm(рис 1)

```
nelianjovu@nelianjovu-Aspire-5755G:~$ mkdir ~/work/arch-pc/lab09
nelianjovu@nelianjovu-Aspire-5755G:~$ cd
nelianjovu@nelianjovu-Aspire-5755G:~$ cd ~/work/arch-pc/lab09
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ cd ~/work/arch-pc/lab09
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ touch lab09-1.asm
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$
```

Рис. 3.1: Рис 1

Рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`. В этом примере `x` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Я скопирую текст программы ниже и скопирую его в созданный мной файл(рис 2)



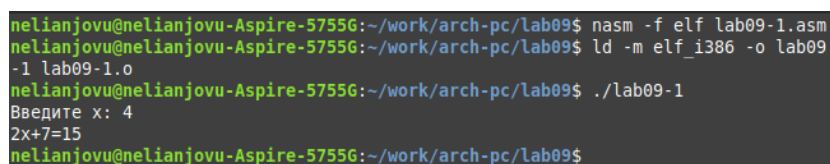
The screenshot shows a text editor window titled "lab09-1.asm (~/.work/arch-pc/lab09)". The editor contains the following assembly code:

```
%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:
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
```

The status bar at the bottom indicates "Matlab", "Spaces: 4", "Ln 15, Col 11", and "INS".

Рис. 3.2: Рис 2

Я создам исполняемый файл и проверю его работу(рис 3)



The screenshot shows a terminal window with the following commands and output:

```
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 4
2x+7=15
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$
```

Рис. 3.3: Рис 3

Я отредактирую программу так, чтобы она решала функцию $f(g(x))$, где $f(x)=2x+7$ и $g(x)=3x-1$ (рис 4)

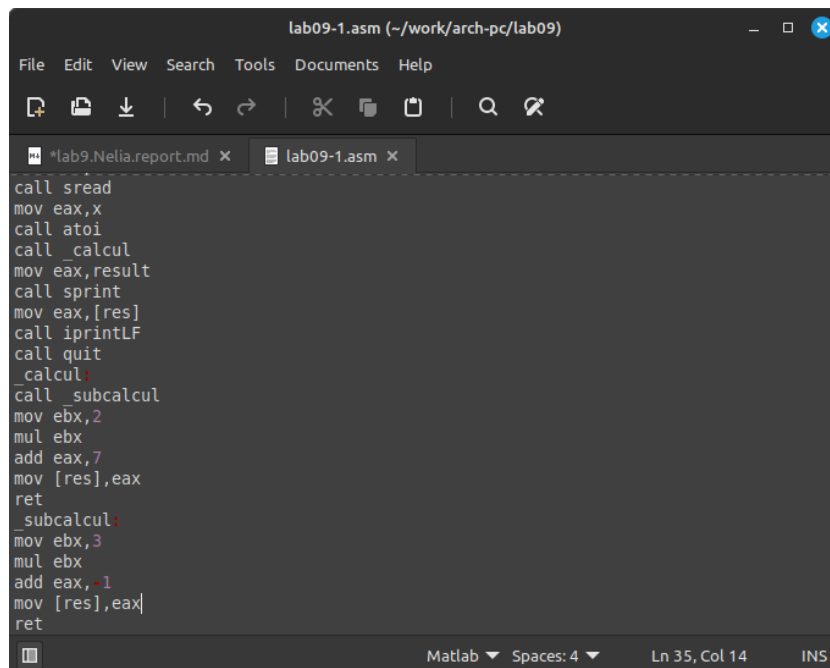


Рис. 3.4: Рис 4

Я создам исполняемый файл и проверю его работу(рис 5)

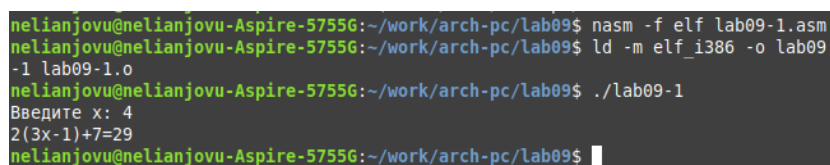


Рис. 3.5: Рис 5

2. Отладка программ с помощью GDB

Я создам новый файл lab09-2asm и скопирую в него данную программу(рис 6)

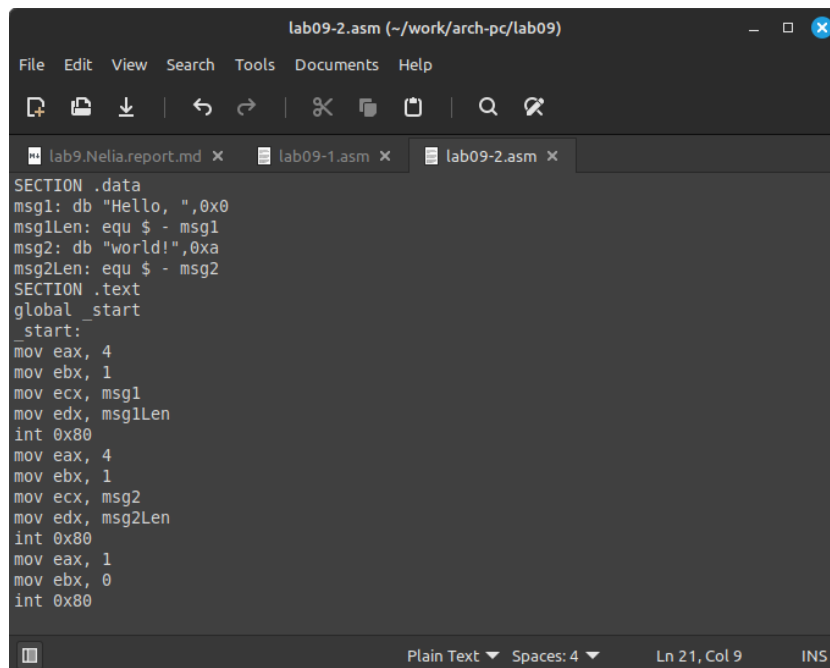


Рис. 3.6: Рис 6

Я создам исполняемый файл и запущу его с помощью отладчика GDB. Чтобы работать с GDB, мне нужно добавить в исполняемый файл отладочную информацию; для этого программы необходимо переводить с ключом «-g»(рис 7)

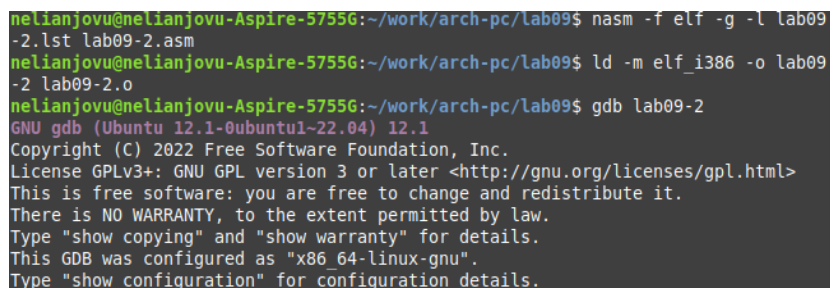


Рис. 3.7: Рис 7

Я протестирую программу, запустив ее в оболочке GDB с помощью команды запуска(рис 8)

```
(gdb) r
Starting program: /home/nelianjovu/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 5366) exited normally]
(gdb)
```

Рис. 3.8: Рис 8

Для более детального анализа программы я поставлю точку останова на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запущу ее(рис 9)

```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 9.
(gdb) r
Starting program: /home/nelianjovu/work/arch-pc/lab09/lab09-2

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

Рис. 3.9: Рис 9

Я буду просматривать дизассемблированный код программы с помощью команды дизассемблирования, начиная с метки `_start`(рис 10)

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     $0x4,%eax
0x08049005 <+5>:    mov     $0x1,%ebx
0x0804900a <+10>:   mov     $0x04a000,%ecx
0x0804900f <+15>:   mov     $0xb,%edx
0x08049014 <+20>:   int     $0x80
0x08049016 <+22>:   mov     $0x4,%eax
0x0804901b <+27>:   mov     $0x1,%ebx
0x08049020 <+32>:   mov     $0x04a000,%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. 3.10: Рис 10

Я переключусь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`(рис 11)

```

(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. 3.11: Рис 11

Для более удобного анализа программы включу режим псевдографики(рис 12)

```

[ Register Values Unavailable ]

B+> 0x08049000 <_start>    mov     eax,0x4
      0x08049005 <_start+5>  mov     ebx,0x1
      0x0804900a <_start+10> mov     ecx,0x804a000
      0x0804900f <_start+15> mov     edx,0x8
      0x08049014 <_start+20> int     0x80
      0x08049016 <_start+22> mov     eax,0x4
      0x0804901b <_start+27> mov     ebx,0x1

native process 6405 In:  start          L9    PC: 0x08049000
(gdb) layout regs
(gdb)

```

Рис. 3.12: Рис 12

В Intel все начинается с адреса, затем с источника, а в ATТ наоборот

2.1. Добавление точек останова

На предыдущих шагах точка останова была установлена по имени метки (`_start`).

Я проверю это с помощью команды `info Breakpoints` (сокращенно `i b`)(рис 13)

```
native process 6405 In: start L9 PC: 0x8049000
(gdb) layout regs
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) █
```

Рис. 3.13: Рис 13

Я поставлю еще одну точку останова по адресу инструкции(рис 14)

```
breakpoint already hit 1 time
(gdb) b *0x08049000
Note: breakpoint 1 also set at pc 0x08049000.
Breakpoint 2 at 0x08049000: file lab09-2.asm, line 9.
(gdb) █
```

Рис. 3.14: Рис 14

Теперь я посмотрю информацию обо всех установленных точках останова.

```
breakpoint already hit 1 time
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2        breakpoint keep y 0x08049000 lab09-2.asm:9
(gdb) █
```

Рис. 3.15: Рис 15

2.2. Работа с данными программы в GDB

Я выполню 5 инструкций с помощью команды Stepі (или Si)(рис 16)

```

nelianjovu@nelianjovu-Aspire-5755G: ~/work/arch-pc/lab09
File Edit View Search Terminal Help
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0

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

native process 6405 In: start L14 PC: 0x8049016
breakpoint already hit 1 time
2 breakpoint keep y 0x8049000 lab09-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.16: Рис 16

Значения `eax`, `ecx`, `esp` и `edx` изменились

Содержимое регистров также можно просмотреть с помощью команды `info Registers`(рис 17)

```

native process 6405 In: start L14 PC: 0x8049016
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0a0 0xffffd0a0
ebp      0x0      0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 3.17: Рис 17

С помощью команды `x &` можно посмотреть содержимое переменной. Я поищу значение переменной `msg1` по имени(рис 18)

```

Type <RET> for more, q to quit, c to continue without paging-- quit
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 3.18: Рис 18

Теперь я посмотрю на значение переменной msg2 по адресу. Адрес переменной можно определить из дизассемблированной инструкции(рис 19)

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

Рис. 3.19: Рис 19

Вы можете изменить значение регистра или ячейки памяти с помощью команды set, передав ей имя или адрес регистра в качестве аргумента. Я изменю первый символ переменной msg1(рис 20)

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a008 <msg1>: "hello, "
(gdb) █
```

Рис. 3.20: Рис 20

Теперь я заменяю символ во второй переменной msg2(рис 21)

```
(gdb) set {char}&msg2='n'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "norld!\n\034"
(gdb) █
```

Рис. 3.21: Рис 21

Я буду использовать команду set для изменения значения регистра ebx(рис 22)

```
$2 = 30
(gdb) set $ebx='2'
(gdb) p/s $ebx
$3 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 2
(gdb) █
```

Рис. 3.22: Рис 22

2.3. Обработка аргументов командной строки в GDB

Я скопирую файл lab8-2.asm, созданный во время лабораторной работы 8, с помощью программы, которая печатает аргументы командной строки, в файл с именем lab09-3.asm(рис 23)

```
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08
/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$
```

Рис. 3.23: Рис 23

Я создам исполняемый файл и загрузим исполняемый файл в отладчик с аргументами, для загрузки программ с аргументами в gdb я буду использовать ключ `--args`(рис 24)

```
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ cd ~/work/arch-pc/lab09
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09
-3.lst lab09-3.asm
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09
-3 lab09-3.o
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ gdb --args lab09-3 4 5
'3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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.
```

Рис. 3.24: Рис 24

Сначала я устанавливаю точку останова перед первой инструкцией в программе и запускаю ее(рис 25)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) r
Starting program: /home/nelianjovu/work/arch-pc/lab09/lab09-3 4 5 3

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)
```

Рис. 3.25: Рис 25

Адрес вершины стека хранится в регистре `esp` и по этому адресу можно увидеть число, равное количеству аргументов командной строки (включая имя программы)(рис 26)

```
(gdb) x/x $esp
0xffffd080: 0x00000004
(gdb)
```

Рис. 3.26: Рис 26

Как видите, количество аргументов равно 4 — это название программы *lab09-3* и сами аргументы: *аргумент1*, *аргумент 2* и *‘аргумент 3’*

Я взгляну на оставшиеся позиции стека — адрес [esp+4] хранит адрес в памяти, где находится имя программы, адрес [esp+8] хранит адрес первого аргумента, адрес [esp+12]] сохраняет второй аргумент и т. д.(рис 27)

```
(gdb) x/s *(void**)(esp + 4)
0xffffd25a:  "/home/nelianjovu/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd286:  "4"
(gdb) x/s *(void**)(esp + 12)
0xffffd288:  "5"
(gdb) x/s *(void**)(esp + 16)
0xffffd28a:  "3"
(gdb) x/s *(void**)(esp + 20)
0x0:  <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 3.27: Рис 27

В 32-битных компьютерах информация хранится именно так: первая память выделяется 4 бита, а вторая — 4x2

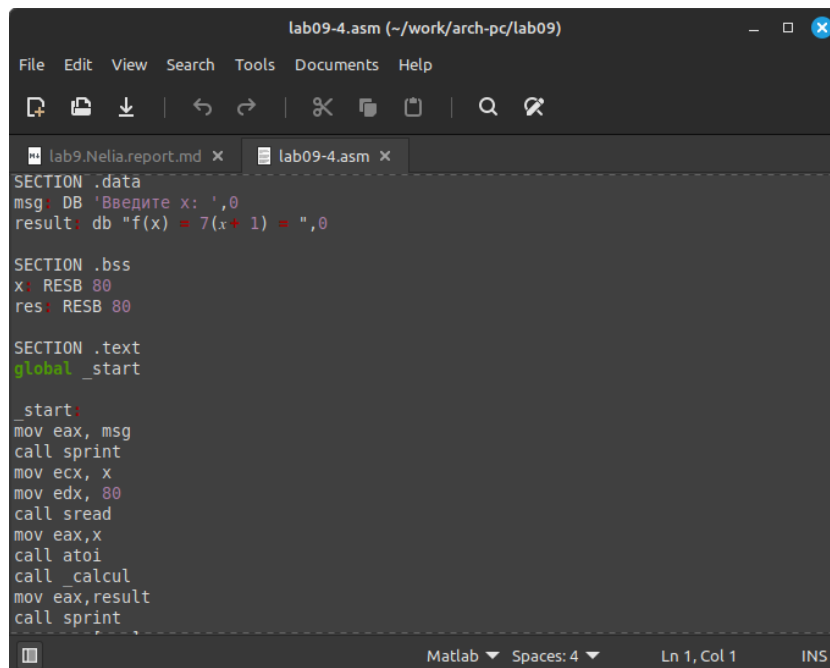
3. Задание для самостоятельной работы

1. Я создам новый файл с именем *lab09-4.asm*(рис 28)

```
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ touch lab09-4.asm
nelianjovu@nelianjovu-Aspire-5755G:~/work/arch-pc/lab09$ █
```

Рис. 3.28: Рис 28

Используя функцию ($f(x) = 7(x+1)$) которая была у меня при выполнении *lab08-задание 1*, я напишу программу, которая вычисляет значение функции как под-программу(рис 29)



The screenshot shows a text editor window titled "lab09-4.asm (~/.work/arch-pc/lab09)". The editor contains assembly code for a program that prompts the user for a value 'x' and calculates $f(x) = 7(x + 1)$. The code is organized into sections: .data for the message and result string, .bss for variables x and res, and .text for the main logic. The main logic starts at a label 'start', prompts the user, reads the input, converts it to an integer, calculates the result, and prints it.

```
SECTION .data
msg: DB 'Введите x: ',0
result: db "f(x) = 7(x + 1) = ",0

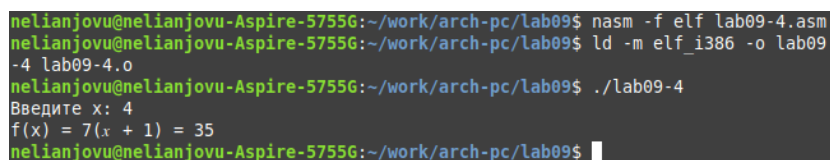
SECTION .bss
x: RESB 80
res: RESB 80

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
```

Рис. 3.29: Рис 29

Я создам исполняемый файл и запущу его(рис 30)

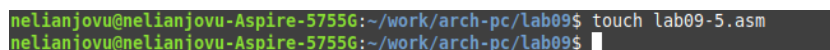


The screenshot shows a terminal window where the assembly file is compiled using nasm and ld, and then executed. The output shows the program prompting for 'x' and displaying the calculated result for x=4.

```
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$ ./lab09-4
Введите x: 4
f(x) = 7(x + 1) = 35
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$
```

Рис. 3.30: Рис 30

2. Я создам новый файл с именем lab09-5.asm(рис 31)

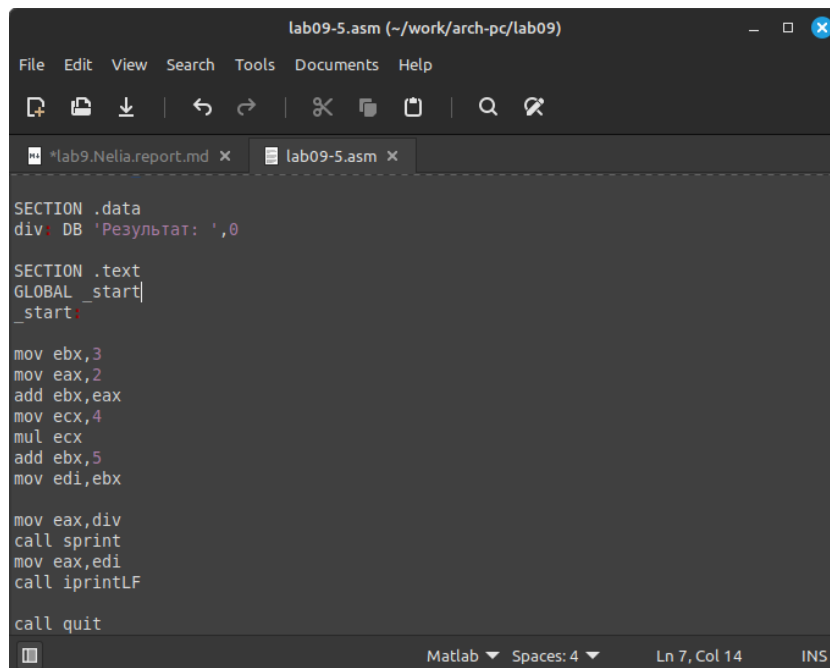


The screenshot shows a terminal window where a new file named lab09-5.asm is created using the touch command.

```
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$ touch lab09-5.asm
nelianjovu@nelianjovu-Aspire-57556:~/work/arch-pc/lab09$
```

Рис. 3.31: Рис 31

Я скопирую данную программу, которая вычисляет значение $(3+2)*4+5$ (рис 32)



The screenshot shows an Assembler IDE window titled "lab09-5.asm (~/.work/arch-pc/lab09)". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for file operations and editing. The code editor displays the following assembly code:

```
SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx

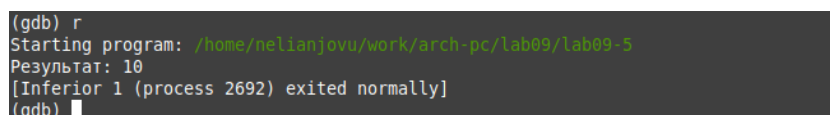
mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit
```

The status bar at the bottom indicates "Matlab", "Spaces: 4", "Ln 7, Col 14", and "INS".

Рис. 3.32: Рис 32

Я создам исполняемый файл и запущу его с помощью GDB(рис 33)



The screenshot shows a GDB terminal window with the following output:

```
(gdb) r
Starting program: /home/nelianjovu/work/arch-pc/lab09/lab09-5
Результат: 10
[Inferior 1 (process 2692) exited normally]
(gdb)
```

Рис. 3.33: Рис 33

Теперь я проверю, где ошибка: первый шаг нашей программы — сложить `ebx`, равный 3, и `eax`, равный 2, что делает `ebx=5`, затем она перемещает 4 в `ecx` и по умолчанию умножает `ecx` на `eax`. что дает `eax 8`. В-третьих, он добавит `ebx` к `ebx`, в результате чего получится 10(рис 34).

The screenshot shows a GDB terminal window with the title bar "nelianjovu@nelianjovu-Aspire-5755G: ~/work/arch-pc/lab09". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The main window displays the "Register group: general" section with the following values:

Register	Value	Comment
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffd0a0	0xffffd0a0

Below the registers, a list of assembly instructions is shown, with the instruction at address 0x80490fb highlighted in blue:

```
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>
```

At the bottom, the status bar shows "native process 4526 In: _start L15 PC: 0x80490fb". The command history includes "(gdb) layout regs", "(gdb) run", "Starting program: /home/nelianjovu/work/arch-pc/lab09/lab09-5", "Breakpoint 1, _start () at lab09-5.asm:10", "(gdb) si 4", "(gdb) si", and "(gdb) |".

Рис. 3.34: Рис 34

Я изменю программу так, чтобы она давала мне правильный ответ(рис 35)

The screenshot shows a text editor window titled "lab09-5.asm (~/work/arch-pc/lab09)". The menu bar includes "File", "Edit", "View", "Search", "Tools", "Documents", and "Help". The main window displays the following assembly code:

```
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax

mov eax,div
call sprint
mov eax,edi
call iprintLF
```

The status bar at the bottom shows "Matlab", "Spaces: 4", "Ln 16, Col 11", and "INS".

Рис. 3.35: Рис 35

Теперь я создам исполняемый файл и запущу его(рис 36)

```
(gdb) r
Starting program: /home/nelianjovu/work/arch-pc/lab09/lab09-5
Результат: 25
[Inferior 1 (process 3738) exited normally]
(gdb) █
```

Рис. 3.36: Рис 36

Текстовая программа для самостоятельной работы 1

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

```
SECTION .data
```

```
msg: DB 'Введите x: ',0
```

```
result: db "f(x) = 7(x + 1) = ",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:
add eax,1
mov ebx,7
mul ebx
mov [res],eax
ret
```

Текстовая программа для самостоятельной работы 2

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

```
SECTION .data
div: DB 'Результат: ',0
```

```
SECTION .text
GLOBAL _start
_start:
```

```
mov ebx,3
mov eax,2
add 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
```

4 Выводы

В ходе лабораторной работы я приобрел навыки написания программ с использованием подпрограмм. А также введение в методы отладки с использованием GDB и его основные возможности.

Список литературы

Архитектура ЭВМ