

Отчет по лабораторной работе №9

Дисциплины: Архитектура компьютера

Зоригоо Номун

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.

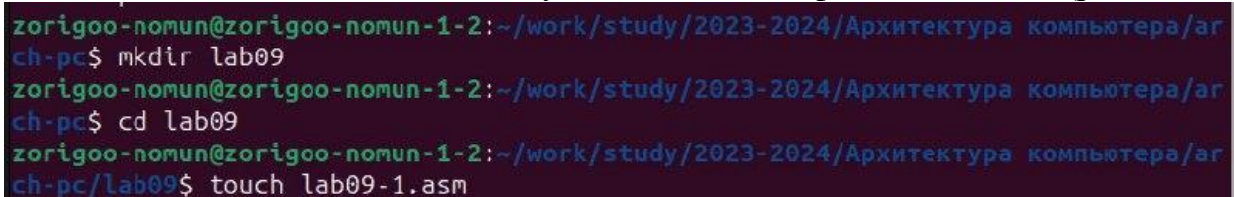
2 Задание

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



```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/ар  
ch-pc$ mkdir lab09  
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/ар  
ch-pc$ cd lab09  
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/ар  
ch-pc/lab09$ touch lab09-1.asm
```

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)

```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-1.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _calcul ; Вызов подпрограммы _calcul
19 mov eax, result
20 call sprint
21 mov eax, [res]
22 call iprintLF
23 call quit
24 _calcul:
25 mov ebx, 2
26 mul ebx
27 add eax, 7
28 mov [res], eax
29 ret ; выход из подпрограммы
```

Рис. 3.2: Рис 2

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

```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ nasm -o lab09-1.o -f elf -g -l list.lst lab09-1.asm
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 lab09-1.o -o lab09-1
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 lab09-1.o -o main
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ./lab09-1
Введите x: 2
2x+7=11
```

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)

```
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-1.asm - Mousepad
File Edit Search View Document Help
[Icons]
3 msg: DB 'Введите x: ',0
4 result: DB '3x-1=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16 mov eax, x
17 call atoi
18 call _calcul ; Вызов подпрограммы _calcul
19 mov eax, result
20 call sprint
21 mov eax, [res]
22 call iprintLF
23 call quit
24 ;-----
25 ; Подпрограмма вычисления
26 ; выражения "3x-1"
27 _calcul:
28 mov ebx, 3
29 mul ebx
30 add eax, -1
31 mov [res], eax
32 ret ; выход из подпрограммы
```

Рис. 3.4: Рис 4

Я создаю исполняемый файл и проверяю его работу(рис 5)

```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ nasm -o lab09-1.o -f elf -g -l list.lst lab09-1.asm
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 lab09-1.o -o lab09-1
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 lab09-1.o -o main
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ./lab09-1
Введите x: 2
3x-1=5
```

Рис. 3.5: Рис 5

2. Отладка программ с помощью GDB

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

```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-2.asm - Mousepad
File Edit Search View Document Help
[Icons]
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6 SECTION .text
7 global _start
8 _start:
9 mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
14 mov eax, 4
15 mov ebx, 1
16 mov ecx, msg2
17 mov edx, msg2Len
18 int 0x80
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
```

Рис. 3.6: Рис 6

Я создаю 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)

```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 14.0.50.20230907-0ubuntu1) 14.0.50.20230907-git
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-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/zorigoo-nomun/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-2
```

Рис. 3.7: Рис 7

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

```
(gdb) r
Starting program: /home/zorigoo-nomun/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-5

Результат: 10
[Inferior 1 (process 12027) exited normally]
```

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) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
```

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     $0x0,%eax
    0x08049005 <+5>: mov     $0x1,%ebx
    0x0804900a <+10>: mov     $0x004000,%ecx
    0x0804900f <+15>: mov     $0xb,%edx
    0x08049014 <+20>: int     $0x0
    0x08049016 <+22>: mov     $0x1,%eax
    0x0804901b <+27>: mov     $0x1,%ebx
    0x08049020 <+32>: mov     $0x004000,%ecx
    0x08049025 <+37>: mov     $0x7,%edx
    0x0804902a <+42>: int     $0x0
    0x0804902c <+44>: mov     $0x1,%eax
    0x08049031 <+49>: mov     $0xb,%ebx
    0x08049036 <+54>: int     $0x0
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,0x0804a000
    0x0804900f <+15>: mov    edx,0x8
    0x08049014 <+20>: int     0x80
    0x08049016 <+22>: mov    eax,0x4
    0x0804901b <+27>: mov    ebx,0x1
    0x08049020 <+32>: mov    ecx,0x0804a008
    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)

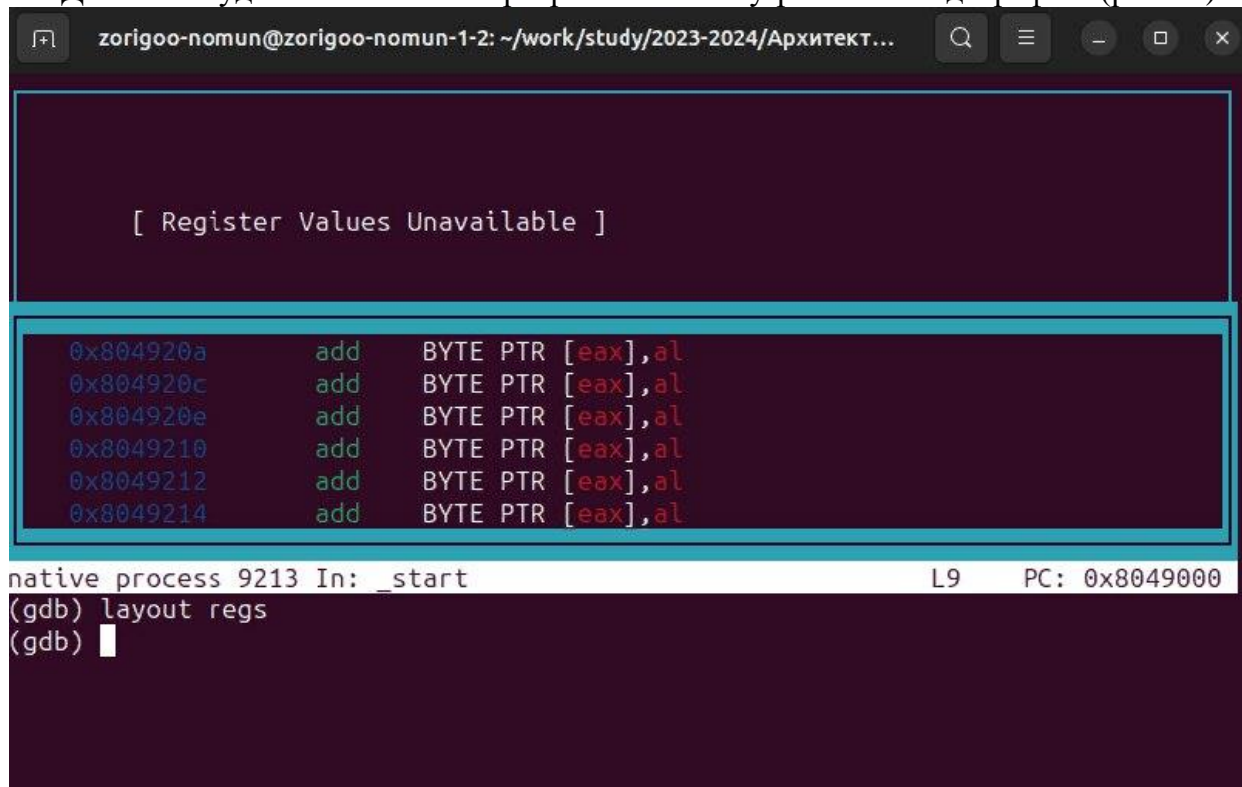


Рис. 3.12: Рис 12

В Intel все начинается с адреса, затем с источника, а в АТТ 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)


```
zorigoo-nomun@zorigoo-nomun-1-2: ~/work/study/2023-2024/Архитект...
[ Register Values Unavailable ]

0x804920a    add    BYTE PTR [eax],al
0x804920c    add    BYTE PTR [eax],al
0x804920e    add    BYTE PTR [eax],al
0x8049210    add    BYTE PTR [eax],al
0x8049212    add    BYTE PTR [eax],al
0x8049214    add    BYTE PTR [eax],al

native process 9213 In: _start          L9    PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
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 2 at 0x08049000: file lab09-2.asm, line 9
(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 Stepi (или Si)(рис 16)

```
zorigoo-nomun@zorigoo-nomun-1-2: ~/work/study/2023-2024/Архитект...
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd010 0xffffd010
ebp      0x0      0x0

0x80490f2 <_start+10> add    %eax,%ebx
0x80490f4 <_start+12> mov    $0x4,%ecx
0x80490f9 <_start+17> mul    %ecx
>0x80490fb <_start+19> add    $0x5,%ebx
0x80490fe <_start+22> mov    %ebx,%edi
0x8049100 <_start+24> mov    $0x804a000,%eax

native process 11829 In: _start L?? PC: 0x80490fb
```

Рис. 3.16: Рис 16

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

Содержимое регистров также можно посмотреть с помощью команды `info Registers`(рис17)

```
0x804925e add    BYTE PTR [eax],al
0x8049260 add    BYTE PTR [eax],al
0x8049262 add    BYTE PTR [eax],al
0x8049264 add    BYTE PTR [eax],al
0x8049266 add    BYTE PTR [eax],al
0x8049268 add    BYTE PTR [eax],al

native process 9213 In: _start L9 PC: 0x8049000
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd010 0xffffd010
ebp      0x0      0x0
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)

```
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
```

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"
```

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)

```
0x804a000 <msg1>: "Hello, "
(gdb) set {char}msg1='h'
```

Рис. 3.20: Рис 20

Теперь я заменю 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)

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 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)

```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ cp ~/work/study/2023-2024/'Архитектура компьютера'/arch-pc/lab08/lab8-2.asm ~/work/study/2023-2024/'Архитектура компьютера'/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)

```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ gdb --args lab09-3 4 5 '3'
GNU gdb (Ubuntu 14.0.50.20230907-0ubuntu1) 14.0.50.20230907-git
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.
```

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) run
Starting program: /home/zorigoo-nomun/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-3 4 5 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
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 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
0xffffcfff0:      0x00000004
(gdb) █
```

Рис. 3.26: Рис 26

Как видите, количество аргументов равно 4 — это название программы *lab09-3* и сами аргументы: аргумент 1, 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/x $esp
0xffffcfff0: 0x00000004
(gdb) x/s *(void**)(esp + 4)
0xffffd1bd: "/home/zorigoo-nomun/work/study/2023-2024/Архитектура компьютера
/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd228: "4"
(gdb) x/s *(void**)(esp + 12)
0xffffd22a: "5"
(gdb) x/s *(void**)(esp + 16)
0xffffd22c: "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)

```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ touch lab09-4.asm
```

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)

```
*/~/Downloads/lab09-4.asm - Mousepad
File Edit Search View Document Help
[Icons]
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите x: ',0
5 result: db "f(x) = 10(x - 1)| = ",0
6
7 SECTION .bss
8 x: RESB 80
9 res: RESB 80
10
11 SECTION .text
12 global _start
13
14 _start:
15 mov eax, msg
16 call sprint
17 mov ecx, x
18 mov edx, 80
19 call sread
20 mov eax, x
21 call atoi
22 call _calcul
23 mov eax, result
24 call sprint
25 mov eax, [res]
26 call iprintLF
27 call quit
28
29 _calcul:
30 add eax, 1
```

Рис. 3.29: Рис 29

Я создам исполняемый файл и запущу его(рис 30)

```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/ар
ch-pc/lab09$ nasm -f elf lab09-4.asm
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/ар
ch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/ар
ch-pc/lab09$ ./lab09-4
Введите x: 3
f(x) = 10(x - 1) = 20
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/ар
```

Рис. 3.30: Рис 30

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

```
zorigoo-nomun@zorigoo-nomun-1-2:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09$ touch lab09-5.asm
```

Рис. 3.31: Рис 31

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

```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-5.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 3.32: Рис 32

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

```
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.  
Результат: 10  
[Inferior 1 (process 11515) 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 the GDB interface with the following content:

Register group: general

Register	Value (Hex)	Value (Dec)
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffd010	0xffffd010
ebp	0x0	0x0

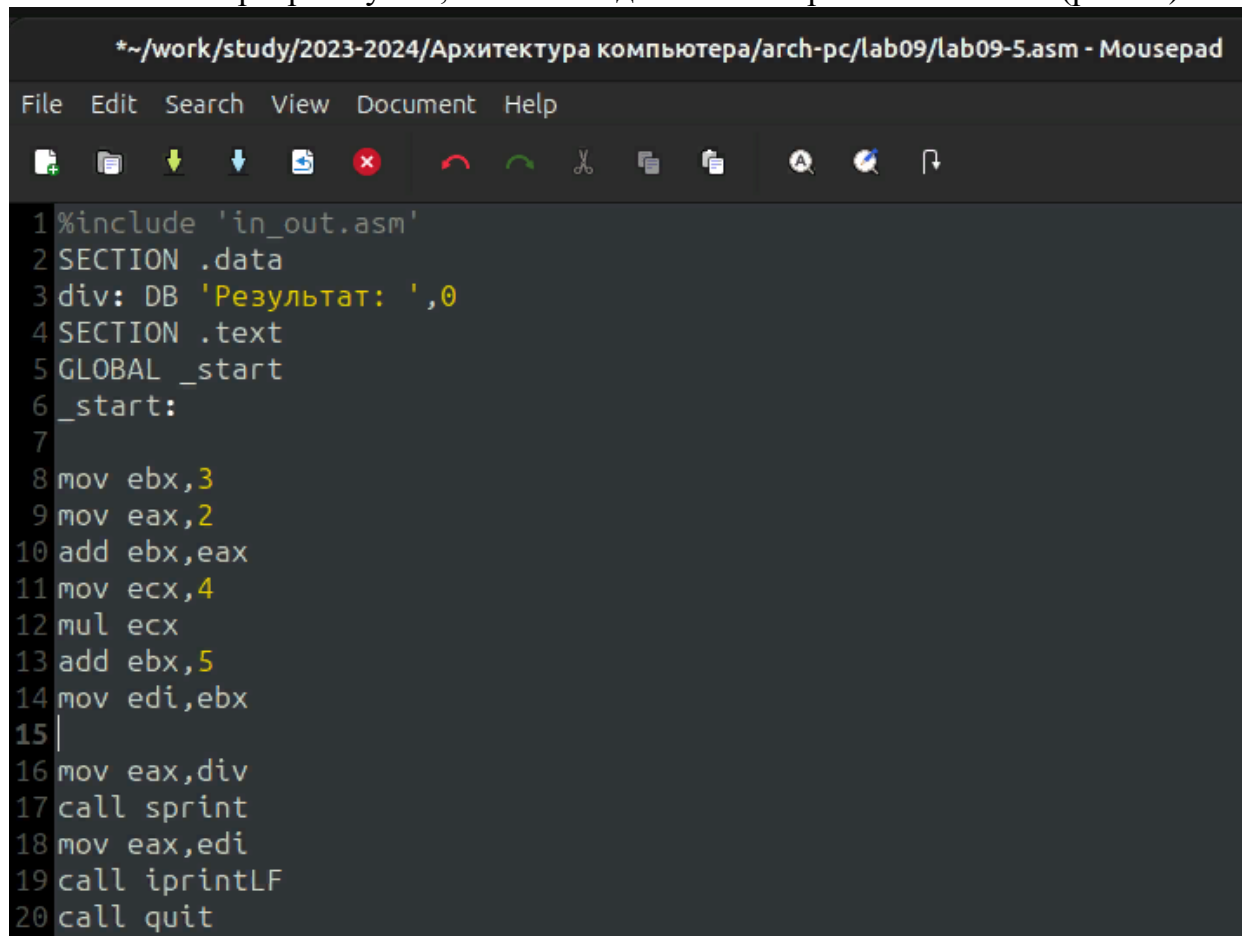
Assembly code window:

```
0x80490f2 <_start+10> add    %eax,%ebx  
0x80490f4 <_start+12> mov    $0x4,%ecx  
0x80490f9 <_start+17> mul    %ecx  
>0x80490fb <_start+19> add    $0x5,%ebx  
0x80490fe <_start+22> mov    %ebx,%edi  
0x8049100 <_start+24> mov    $0x804a000,%eax
```

native process 11829 In: _start L?? PC: 0x80490fb

Рис. 3.34: Рис 34

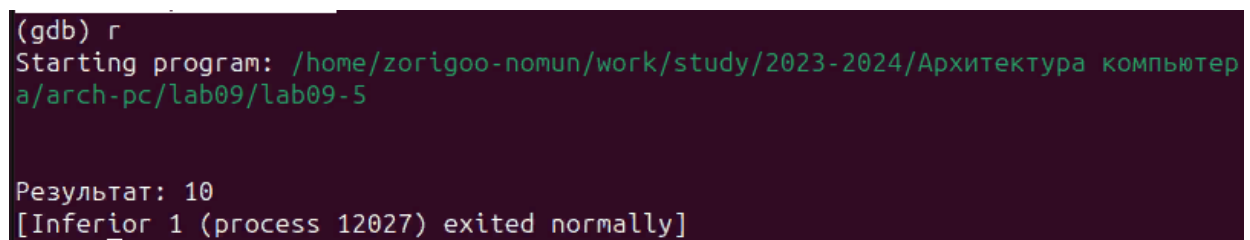
Я изменю программу так, чтобы она давала мне правильный ответ(рис 35)



```
*~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-5.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 |
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 3.35: Рис 35

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



```
(gdb) r
Starting program: /home/zorigoo-номун/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab09/lab09-5
Результат: 10
[Inferior 1 (process 12027) exited normally]
```

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) = 10(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,10

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