

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютера

Мирзоян Ян Игоревич

Содержание

1	Цель работы	6
2	Задание	7
3	Теоретическое введение	8
4	Выполнение лабораторной работы	9
5	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	Создаю рабочую директорию и файл в ней	9
4.2	Редактирую файл вводя текст листинга	9
4.3	Создаю исполняемый файл. Проверяю корректность работы программы	10
4.4	Изменяю текст программы чтобы она считала $f(g(x))$	10
4.5	Создаю исполняемый файл. Проверяю корректность работы программы	11
4.6	Создаю файл. Ввожу в него текст листинга	11
4.7	Запускаю программу в отладочной оболочке GDB	11
4.8	Для более подробного анализа программы устанавливаю брейк-поинт на метку <code>_start</code> , с которой начинается выполнение любой ассемблерной программы, и запускаю её	12
4.9	Смотрю дисассимилированный код программы с помощью команды <code>disassemble</code> начиная с метки <code>_start</code>	12
4.10	Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду <code>setdisassembly-flavor intel</code> . В представлении АТТ в виде 16-ричного числа записаны первые аргументы всех команд, а в представлении <code>intel</code> так записываются адреса вторых аргументов	13
4.11	Включаю режим псевдографики, с помощью которого отображается код программы и содержимое регистров	13
4.12	Проверяю наличие точки останова	14
4.13	Добавляю ещё одну точку останова и проверяю сколько точек останова есть	14
4.14	Выполняю 5 инструкций с помощью команды <code>si</code> . Изменились значения регистров <code>eax</code> <code>ecx</code> <code>edx</code> <code>ebx</code>	14
4.15	Просматриваю значение переменной <code>msg1</code> по имени	15
4.16	Просматриваю значение переменной <code>msg2</code> по адресу	15
4.17	Изменяю первый символ переменной <code>msg1</code>	15
4.18	Изменяю первый символ переменной <code>msg2</code>	15
4.19	Вывожу значение <code>edx</code> в разных форматах: строчном, 16-ричном, двоичном	16
4.20	С помощью команды <code>set</code> изменяю значение регистра <code>ebx</code> . Разница вывода из-за того что в первом случае 2 это символ а во втором число.	16

4.21 Скопировал файл lab8-2.asm, созданный при выполнении лабораторной работы No8, с программой выводящей на экран аргументы командной строки. в файл с именем lab09-3.asm, создал исполняемый файл	16
4.22 Запускаю программу в оболочке GDB	17
4.23 Узнаю количество аргументов	17
4.24 смотрю все позиции стека. Их адреса располагаются в 4 байтах друг от друга (именно столько занимает элемент стека)	17

Список таблиц

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 Теоретическое введение

Подпрограмма — поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий.

4 Выполнение лабораторной работы

##Реализация подпрограмм в NASM

```
[yimirzoyan@fedora arch-pc]$ mkdir lab09
[yimirzoyan@fedora arch-pc]$ cd lab09
[yimirzoyan@fedora lab09]$ touch lab09-1.asm
[yimirzoyan@fedora lab09]$
```

Рис. 4.1: Создаю рабочую директорию и файл в ней



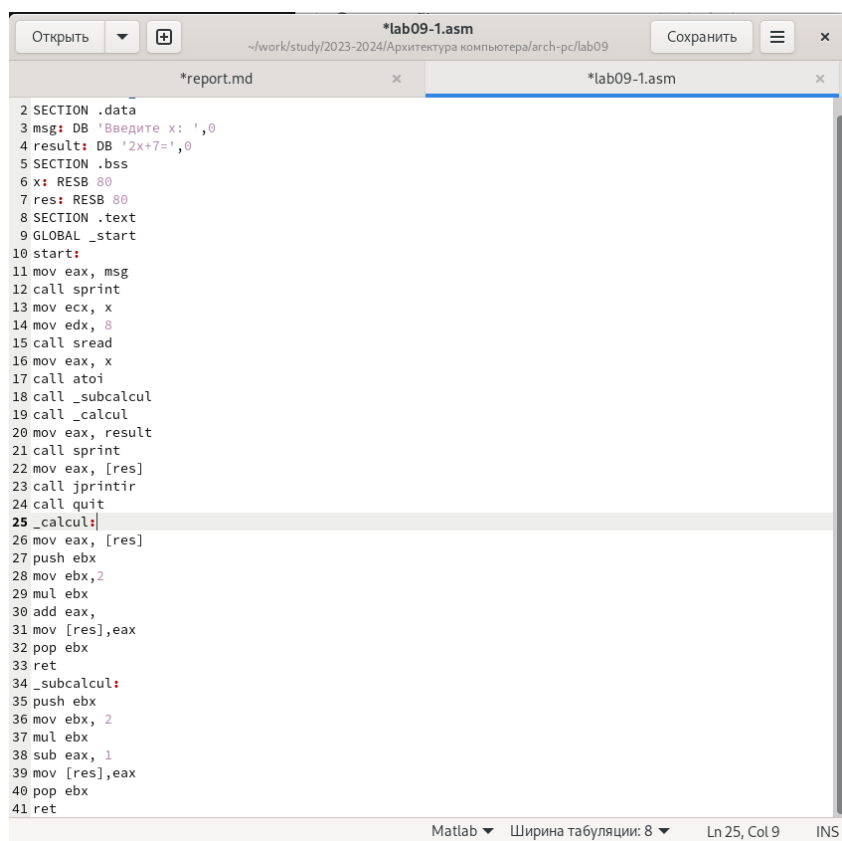
Рис. 4.2: Редактирую файл вводя текст листинга

```

[yimirzoyan@fedora lab09]$ nasm -f elf lab09-1.asm
[yimirzoyan@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[yimirzoyan@fedora lab09]$ ./lab09-1
Введите x: 2
2x+7=11
[yimirzoyan@fedora lab09]$ ./lab09-1
Введите x: 3
2x+7=13
[yimirzoyan@fedora lab09]$

```

Рис. 4.3: Создаю исполняемый файл. Проверяю корректность работы программы



```

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, 8
15 call sread
16 mov eax, x
17 call atoi
18 call _subcalcul
19 call _calcul
20 mov eax, result
21 call sprint
22 mov eax, [res]
23 call jprintir
24 call quit
25 _calcul:
26 mov eax, [res]
27 push ebx
28 mov ebx, 2
29 mul ebx
30 add eax,
31 mov [res],eax
32 pop ebx
33 ret
34 _subcalcul:
35 push ebx
36 mov ebx, 2
37 mul ebx
38 sub eax, 1
39 mov [res],eax
40 pop ebx
41 ret

```

Рис. 4.4: Изменяю текст программы чтобы она считала $f(g(x))$

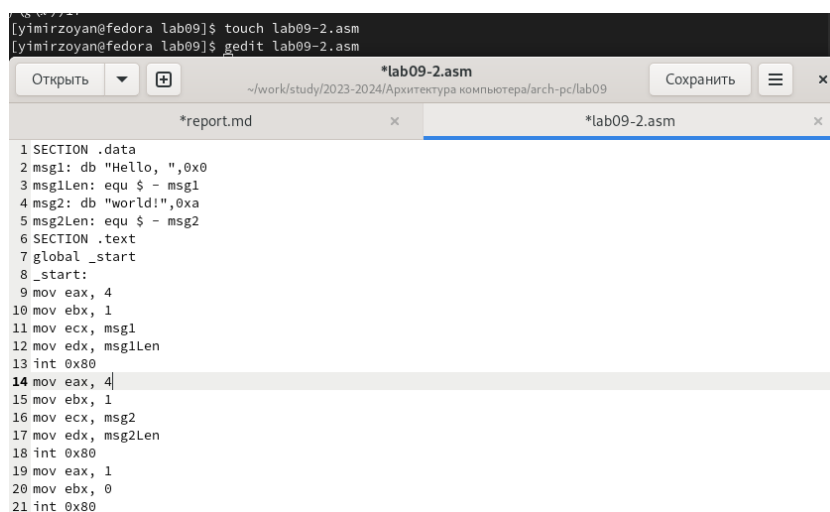
```

[yimirzoyan@fedora lab09]$ nasm -f elf lab09-1.asm
[yimirzoyan@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[yimirzoyan@fedora lab09]$ ./lab09-1
Введите x: 2
f(g(x))13
[yimirzoyan@fedora lab09]$ 3
bash: 3: команда не найдена...
[yimirzoyan@fedora lab09]$ ./lab09-1
Введите x: 3
f(g(x))17
[yimirzoyan@fedora lab09]$

```

Рис. 4.5: Создаю исполняемый файл. Проверяю корректность работы программы

##Отладка программ с помощью GDB



```

[yimirzoyan@fedora lab09]$ touch lab09-2.asm
[yimirzoyan@fedora lab09]$ gedit lab09-2.asm

```

Открыть + *lab09-2.asm ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 Сохранить x

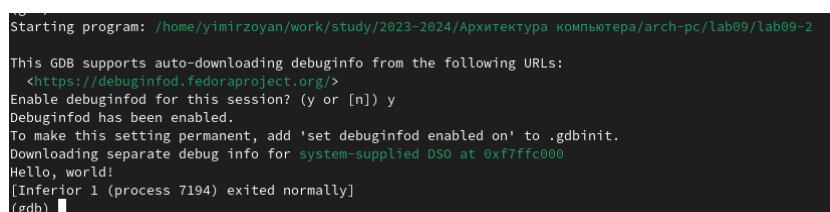
*report.md x *lab09-2.asm x

```

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. 4.6: Создаю файл. Ввожу в него текст листинга



```

Starting program: /home/yimirzoyan/work/study/2023-2024/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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 7194) exited normally]
(gdb)

```

Рис. 4.7: Запускаю программу в отладочной оболочке GDB

```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/yimirzoyan/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 4.8: Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запускаю её

```
(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)
```

Рис. 4.9: Смотрю дисассемблированный код программы с помощью команды `disassemble` начиная с метки `_start`

```

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

```

Рис. 4.10: Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `setdisassembly-flavor intel`. В представлении АТТ в виде 16-ричного числа записаны первые аргументы всех команд, а в представлении `intel` так записываются адреса вторых аргументов

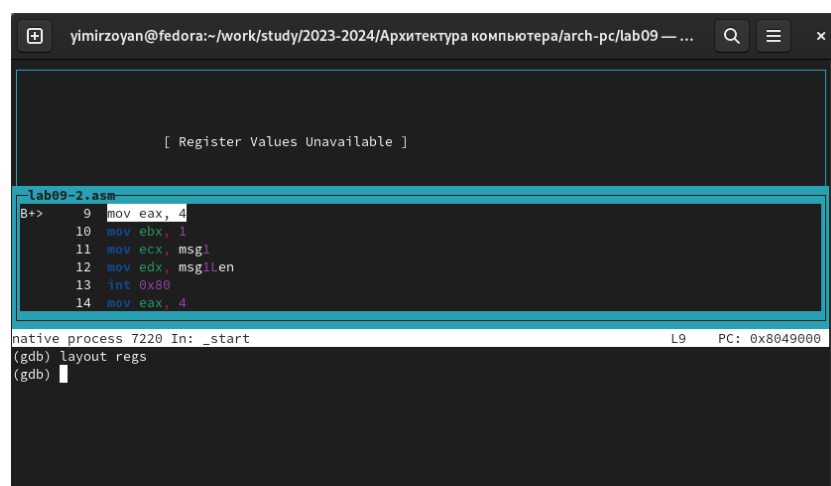


Рис. 4.11: Включаю режим псевдографики, с помощью которого отображается код программы и содержимое регистров

```
(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. 4.12: Проверяю наличие точки останова

```
yimirzoyan@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 — ...
[ Register Values Unavailable ]

~lab09-2.asm
B+> 9  mov eax, 4
10  mov ebx, 1
11  mov ecx, msg1
12  mov edx, msg1len
13  int 0x80
14  mov eax, 4

native process 7220 In: _start L9 PC: 0x08049000
(gdb) b *0x08049031
Breakpoint 2 at 0x08049031: file lab09-2.asm, line 20.
(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 0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 4.13: Добавляю ещё одну точку останова и проверяю сколько точек останова есть

```
yimirzoyan@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09 — ...
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
0x08049020 <_start+32> mov ecx,0x804a008
0x08049025 <_start+37> mov edx,0x7
0x0804902a <_start+42> int 0x80
0x0804902c <_start+44> mov eax,0x1
b+ 0x08049031 <_start+49> mov ebx,0x0
0x08049036 <_start+54> int 0x80
0x08049038 add BYTE PTR [eax],al
0x0804903a add BYTE PTR [eax],al

native process 7220 In: _start L14 PC: 0x08049016
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) layout asm
(gdb)
```

Рис. 4.14: Выполняю 5 инструкций с помощью команды si. Изменились значения регистров eax ecx edx ebx

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

Рис. 4.15: Просматриваю значение переменной msg1 по имени

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

Рис. 4.16: Просматриваю значение переменной msg2 по адресу

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) █
```

Рис. 4.17: Изменяю первый символ переменной msg1

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

Рис. 4.18: Изменяю первый символ переменной msg2

```
(gdb) p/s $edx
$1 = 8
(gdb) p/x
$2 = 0x8
(gdb) p/t
$3 = 1000
(gdb) █
```

Рис. 4.19: Вывожу значение `edx` в разных форматах: строчном, 16-ричном, двоичном

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) █
```

Рис. 4.20: С помощью команды `set` изменяю значение регистра `ebx`. Разница вывода из-за того что в первом случае `2` это символ а во втором число.

```
[yimirzoyan@fedora lab09]$ cp /home/yimirzoyan/work/study/2023-2024/'Архитектура компьютера'/arch-pc/lab08/lab8-2.asm /home/yimirzoyan/work/study/2023-2024/'Архитектура компьютера'/arch-pc/lab09/lab09-3.asm
[yimirzoyan@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[yimirzoyan@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[yimirzoyan@fedora lab09]$ █
```

Рис. 4.21: Скопировал файл `lab8-2.asm`, созданный при выполнении лабораторной работы No8, с программой выводящей на экран аргументы командной строки. в файл с именем `lab09-3.asm`, создал исполняемый файл


```
[yimirzoyan@fedora lab09]$ gdb --args lab09-3 arg1 arg 2 'arg 3'
GNU gdb (GDB) Fedora Linux 13.1-2.fc38
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 lab09-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/yimirzoyan/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-3 arg1 arg 2 arg\ 3
```

Рис. 4.22: Запускаю программу в оболочке GDB

```
(gdb) x/x $esp
0xffffd100:      0x00000005
(gdb)
```

Рис. 4.23: Узнаю количество аргументов

```
(gdb) x/s *(void**)(esp + 4)
0xffffd2a7:      "/home/yimirzoyan/work/study/2023-2024/Архитектура компьютера/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd30f:      "arg1"
(gdb) x/s *(void**)(esp + 12)
0xffffd314:      "arg"
(gdb) x/s *(void**)(esp + 16)
0xffffd318:      "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd31a:      "arg 3"
(gdb) x/s *(void**)(esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.24: смотрю все позиции стека. Их адреса располагаются в 4 байтах друг от друга (именно столько занимает элемент стека)

5 Выводы

В результате выполнения работы, я научился организовывать код в подпрограммы и познакомился с базовыми функциями отладчика gdb.

Список литературы

1. Лабораторная работа 9