

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров и операционные системы

Бизев Никита Владимирович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Реализация подпрограмм в NASM	10
4.2	Отладка программ с помощью GDB	13
4.2.1	Добавление точек останова	17
4.2.2	Работа с данными программы в GDB	18
4.2.3	Обработка аргументов командной строки в GDB	23
5	Выводы	25
6	Список литературы	26

Список иллюстраций

4.1	Создание файлов для лабораторной работы	10
4.2	Ввод текста программы из листинга 9.1	11
4.3	Запуск исполняемого файла	11
4.4	Изменение текста программы согласно заданию	12
4.5	Запуск исполняемого файла	13
4.6	Ввод текста программы из листинга 9.2	13
4.7	Получение исполняемого файла	14
4.8	Загрузка исполняемого файла в отладчик	14
4.9	Проверка работы файла с помощью команды run	14
4.10	Установка брейкпоинта и запуск программы	15
4.11	Использование команд disassemble и disassembly-flavor intel	16
4.12	Включение режима псевдографики	17
4.13	Установление точек останова и просмотр информации о них	18
4.14	До использования команды stepi	19
4.15	После использования команды stepi	20
4.16	Просмотр значений переменных	21
4.17	Использование команды set	21
4.18	Вывод значения регистра в разных представлениях	22
4.19	Использование команды set для изменения значения регистра	22
4.20	Завершение работы GDB	23
4.21	Создание файла	23
4.22	Загрузка файла с аргументами в отладчик	24
4.23	Установление точки останова и запуск программы	24
4.24	Просмотр значений, введенных в стек	24

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. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра

(watchpoints) и точки отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда quit (или сокращённо q).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g.

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой info (кратко i).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable.

Обратно точка останова активируется командой enable.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete.

Для продолжения остановленной программы используется команда continue (c). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число N, которое указывает отладчику проигнорировать N – 1 точку останова (выполнение остановится на N-й точке).

Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок

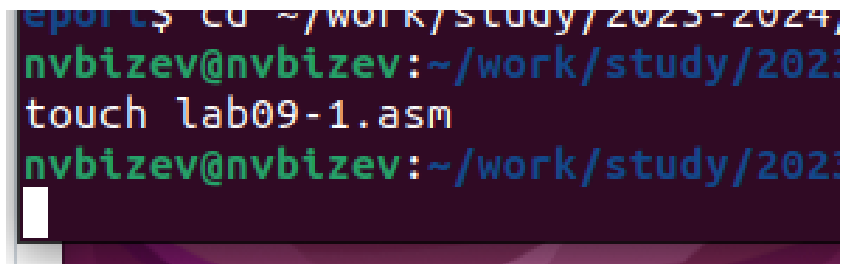
кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `еір`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

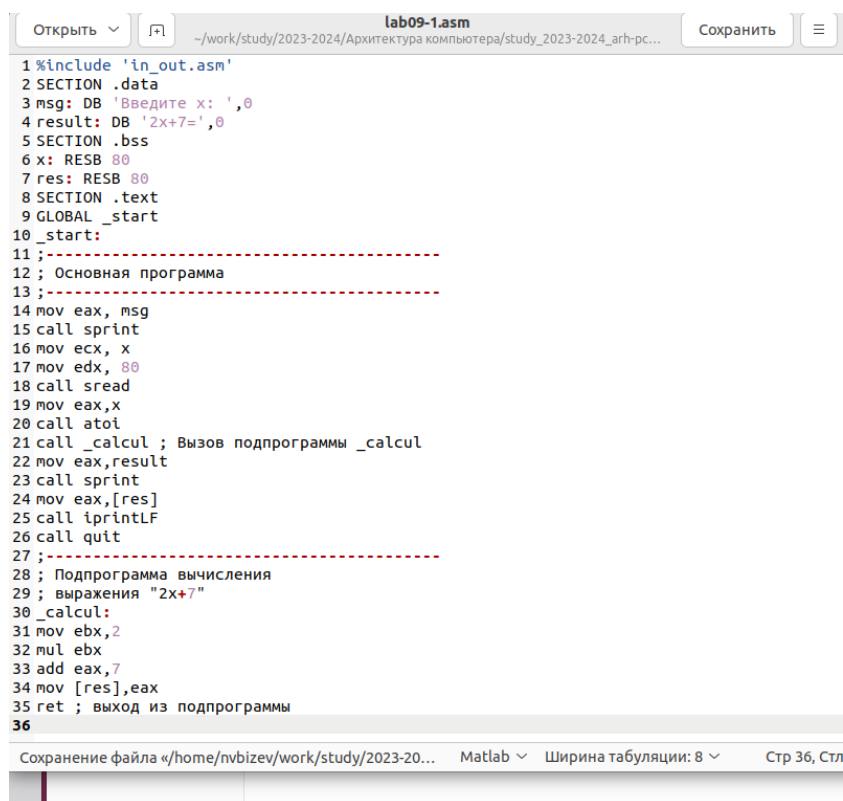
Создаю файл lab09-1.asm. (рис. 4.1)

A screenshot of a terminal window with a dark background and light-colored text. The text shows a user at a prompt creating a file. The prompt is 'nvbizev@nvbizev:~/work/study/2023-2024/'. The command entered is 'touch lab09-1.asm'. The output is 'nvbizev@nvbizev:~/work/study/2023-2024/'.

```
nvbizev@nvbizev:~/work/study/2023-2024/  
touch lab09-1.asm  
nvbizev@nvbizev:~/work/study/2023-2024/
```

Рис. 4.1: Создание файлов для лабораторной работы

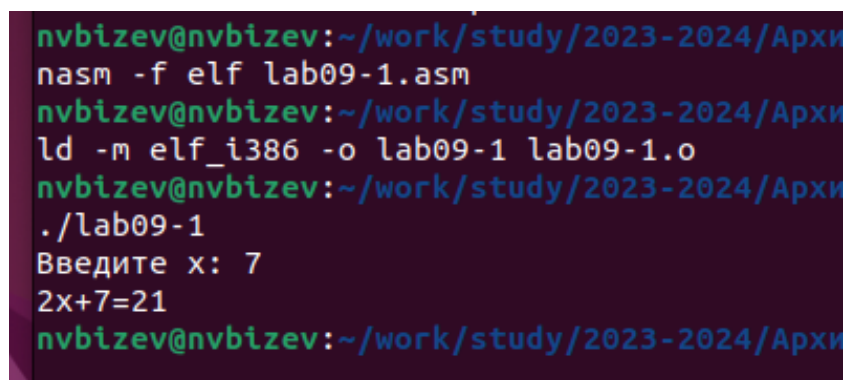
Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. 4.2)



```
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 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [res], eax
35 ret ; выход из подпрограммы
36
```

Рис. 4.2: Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу. (рис. 4.3)

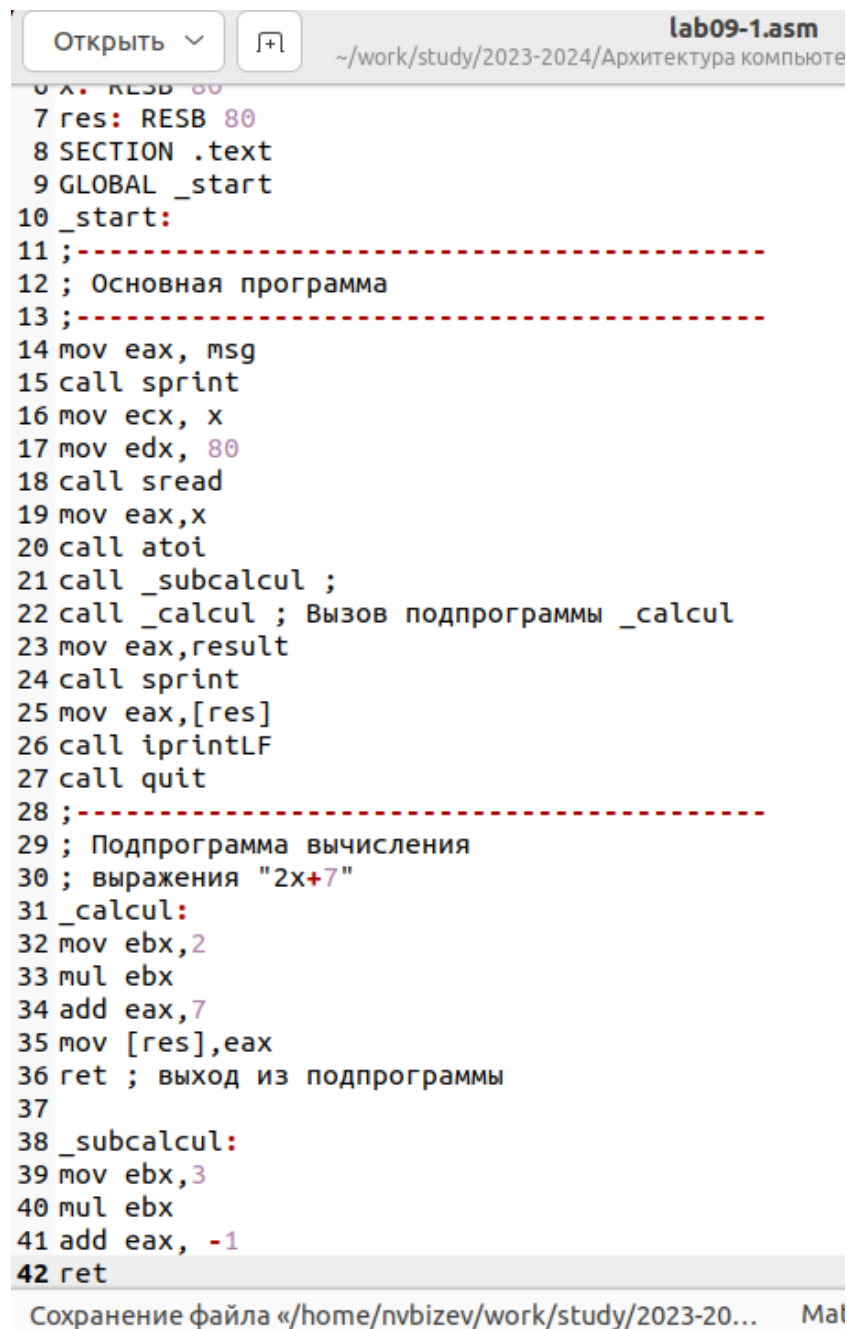


```
nvbizev@nvbizev:~/work/study/2023-2024/Архи
nasm -f elf lab09-1.asm
nvbizev@nvbizev:~/work/study/2023-2024/Архи
ld -m elf_i386 -o lab09-1 lab09-1.o
nvbizev@nvbizev:~/work/study/2023-2024/Архи
./lab09-1
Введите x: 7
2x+7=21
nvbizev@nvbizev:~/work/study/2023-2024/Архи
```

Рис. 4.3: Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x$

+ 7, $g(x) = 3x - 1$. (рис. 4.4)



```
lab09-1.asm
~/work/study/2023-2024/Архитектура компьюте

6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _subcalcul ;
22 call _calcul ; Вызов подпрограммы _calcul
23 mov eax, result
24 call sprint
25 mov eax, [res]
26 call iprintLF
27 call quit
28 ;-----
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32 mov ebx, 2
33 mul ebx
34 add eax, 7
35 mov [res], eax
36 ret ; выход из подпрограммы
37
38 _subcalcul:
39 mov ebx, 3
40 mul ebx
41 add eax, -1
42 ret

Сохранение файла «/home/nvbizev/work/study/2023-20... Mal
```

Рис. 4.4: Изменение текста программы согласно заданию

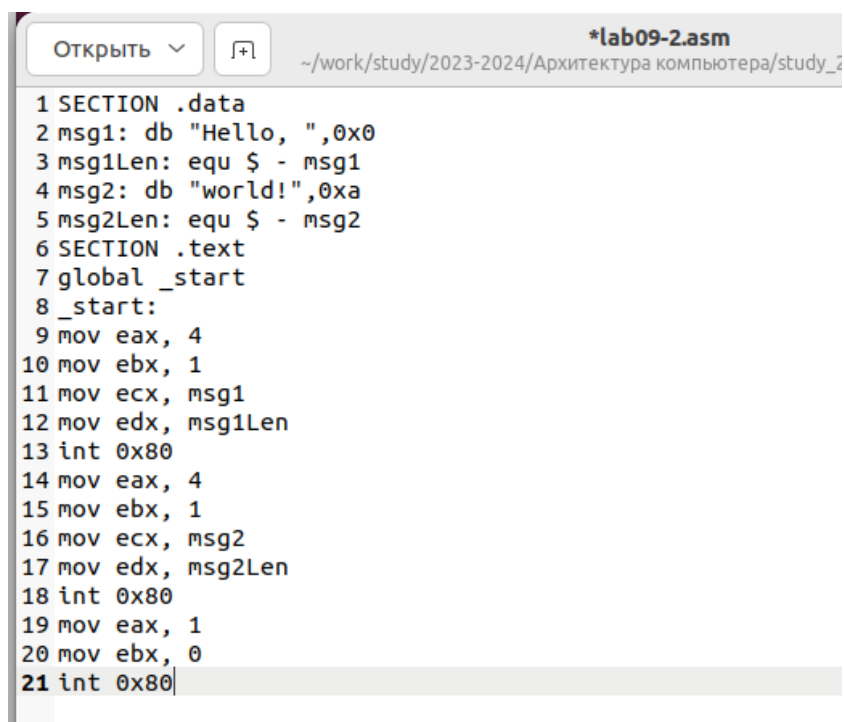
Создаю исполняемый файл и проверяю его работу. (рис. 4.5)

```
nvbizev@nvbizev:~/work/study/2023-2024/Архитект  
ld -m elf_i386 -o lab09-1 lab09-1.o  
nvbizev@nvbizev:~/work/study/2023-2024/Архитект  
./lab09-1  
Введите x: 7  
2x+7=47  
nvbizev@nvbizev:~/work/study/2023-2024/Архитект
```

Рис. 4.5: Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. 4.6)



```
*lab09-2.asm  
~/work/study/2023-2024/Архитектура компьютера/study_2  
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: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. 4.7)

```

nvbizev@nvbizev:~/work/study/2023-2024/Архитектура ко
nasm -f elf -g -l lab09-2.lst lab09-2.asm
nvbizev@nvbizev:~/work/study/2023-2024/Архитектура ко
ld -m elf_i386 -o lab09-2 lab09-2.o
nvbizev@nvbizev:~/work/study/2023-2024/Архитектура ко

```

Рис. 4.7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. 4.8)

```

gdb lab09-2
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.
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)

```

Рис. 4.8: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. 4.9)

```

(gdb) run
Starting program: /home/nvbizev/work/study/2023-2024/Архитектура ко
c/labs/lab09/lab09-2
Hello, world!
[Inferior 1 (process 37688) exited normally]
(gdb)

```

Рис. 4.9: Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start и запускаю её. (рис. 4.10)

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9
(gdb) run
Starting program: /home/nvbizev/work/study/2023-2024/sem4/labs/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:9
9      mov eax, 4
(gdb)

```

Рис. 4.10: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start`, и переключаюсь на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. 4.11)

```

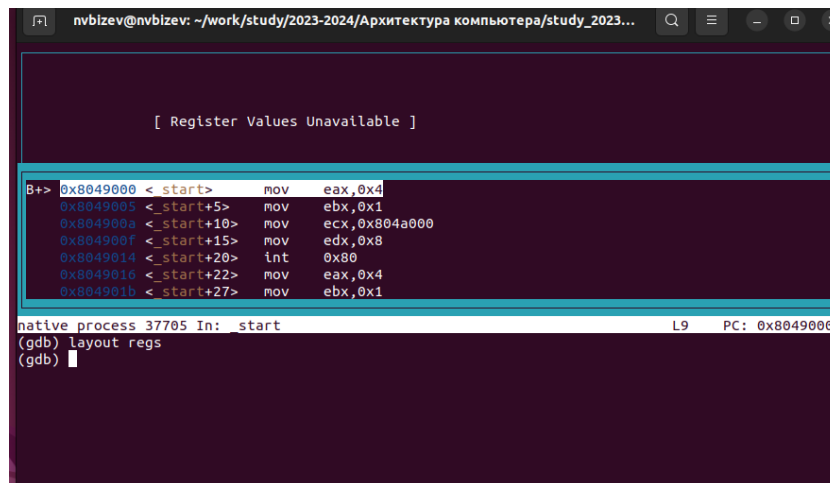
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     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) 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. 4.11: Использование команд disassemble и disassembly-flavor intel

В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs. (рис. 4.12)

A screenshot of a GDB terminal window. The window title is 'nvbizev@nvbizev: ~/work/study/2023-2024/Архитектура компьютера/study_2023...'. The main area shows assembly code with addresses and disassembled instructions. A light blue box highlights the first six lines of assembly. Below the assembly, the status bar shows 'native process 37705 In: _start L9 PC: 0x8049000'. At the bottom, the GDB prompt '(gdb)' is followed by the command 'layout regs' and a cursor.

```
[ Register Values Unavailable ]

B+> 0x8049000 <_start>    mov     eax,0x4
0x8049003 <_start+5>    mov     ebx,0x1
0x8049006 <_start+10>   mov     ecx,0x804a000
0x8049009 <_start+15>   mov     edx,0x8
0x8049014 <_start+20>   int     0x80
0x8049016 <_start+22>   mov     eax,0x4
0x804901b <_start+27>   mov     ebx,0x1

native process 37705 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) █
```

Рис. 4.12: Включение режима псевдографики

4.2.1 Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова. (рис. 4.13)

```
nvbizev@nvbizev: ~/work/study/2023-2024/Архитектура компьютера/study_2023...
[ Register Values Unavailable ]

0x8049060    add    BYTE PTR [eax],al
0x8049062    add    BYTE PTR [eax],al
0x8049064    add    BYTE PTR [eax],al
0x8049066    add    BYTE PTR [eax],al
0x8049068    add    BYTE PTR [eax],al
0x804906a    add    BYTE PTR [eax],al
0x804906c    add    BYTE PTR [eax],al
0x804906e    add    BYTE PTR [eax],al
0x8049070    add    BYTE PTR [eax],al
0x8049072    add    BYTE PTR [eax],al
0x8049074    add    BYTE PTR [eax],al

native process 37705 In: start                                L9    PC: 0x8049000
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: 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: Установление точек останова и просмотр информации о них

4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров. (рис. 4.14)

```
nvbizev@nvbizev: ~/work/study/2023-2024/Архитектура компьютера/study_2023...
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd100 0xffffd100
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35

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,0x804a008
0x8049025 <_start+37> mov    edx,0x7
0x804902a <_start+42> int     0x80
0x804902c <_start+44> mov    eax,0x1

native process 37705 In: _start L10 PC: 0x8049005
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: 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) stepi
(gdb)
```

Рис. 4.14: До использования команды stepi

(рис. 4.15)

```

nvbizev@nvbizev: ~/work/study/2023-2024/Архитектура компьютера/study_2023...
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd100 0xffffd100
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b < start+27>
eflags   0x202    [ IF ]
cs       0x23     35

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
0x804902a <_start+42> int    0x80
0x804902c <_start+44> mov    eax,0x1
b+ 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038 add    BYTE PTR [eax],al

native process 37705 In: start L15 PC: 0x804901b
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num    Type             Disp Enb Address      What
1      breakpoint        keep y  0x8049000 lab09-2.asm:9
       breakpoint already hit 1 time
2      breakpoint        keep y  0x8049031 lab09-2.asm:20
(gdb) stepi
(gdb) si 5
(gdb)

```

Рис. 4.15: После использования команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу. (рис. 4.16)

```

nvbizev@nvbizev: ~/work/study/2023-2024/Архитектура компьютера/study_2023...
Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd100 0xffffd100
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b < start+27>
eflags   0x202    [ IF ]
cs       0x23     35

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
0x804902a <_start+42> int    0x80
0x804902c <_start+44> mov    eax,0x1
b+ 0x8049031 <_start+49> mov    ebx,0x0
0x8049036 <_start+54> int    0x80
0x8049038      add    BYTE PTR [eax],al

native process 37705 In: _start L15 PC: 0x804901b
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: 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) stepi
(gdb) si 5
(gdb) x/1bs &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 4.16: Просмотр значений переменных

С помощью команды `set` изменяю первый символ переменной `msg1` и заменяю первый символ в переменной `msg2`. (рис. 4.17)

```

(gdb) set {char}&msg1='h'
(gdb) x/lbs &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2 = 'b'
(gdb) x/lbs &msg2
0x804a008 <msg2>:      "borld!\n\034"
(gdb)

```

Рис. 4.17: Использование команды `set`

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра `edx` с помощью команды `print p/F $val`. (рис. 4.18)

```
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb)
```

Рис. 4.18: Вывод значения регистра в разных представлениях

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием. (рис. 4.19)

```
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb)
```

Рис. 4.19: Использование команды `set` для изменения значения регистра

Разница вывода команд `p/s $ebx` отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы с помощью команды `continue` и выхожу из GDB с помощью команды `quit`. (рис. 4.20)

```
(gdb) c
Continuing.
borld!

Breakpoint 2, _start () at lab09-2.asm
(gdb) q
A debugging session is active.

        Inferior 1 [process 37705] will
        be killed if you choose to quit.

Quit anyway? (y or n) █
```

Рис. 4.20: Завершение работы GDB

4.2.3 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл. (рис. 4.21)

```
nvblizev@nvblizev:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/Lab09$ cp ~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/Lab08/lab8-2.asm ~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/Lab09/lab09-3.asm
nvblizev@nvblizev:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/Lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
nvblizev@nvblizev:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/Lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
nvblizev@nvblizev:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/Lab09$
```

Рис. 4.21: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа -args. (рис. 4.22)

```

nvblizev@nvblizev:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/lab09$
gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 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.
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 lab09-3...
(gdb)

```

Рис. 4.22: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. 4.23)

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/nvblizev/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

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

```

Рис. 4.23: Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам. (рис. 4.24)

```

(gdb) x/x $esp
0xffffd0c0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd280: "/home/nvblizev/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arh-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd2ff: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd311: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd322: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd324: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.24: Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к количество аргументов командной строки равно 4.

5 Выводы

Во время выполнения данной лабораторной работы я приобрел навыки написания программ с использованием подпрограмм и ознакомился с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

- 1.