

Отчёт по лабораторной работе №9

Хань Цзянтао

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	12
4.2.1	Добавление точек останова	16
4.2.2	Работа с данными программы в GDB	17
4.2.3	Обработка аргументов командной строки в GDB	22
4.3	Задания для самостоятельной работы.....	24
5	Выводы	31
6	Список литературы	32

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	12
4.6	Ввод текста программы из листинга 9.2	13
4.7	Получение исполняемого файла	13
4.8	Загрузка исполняемого файла в отладчик	14
4.9	Проверка работы файла с помощью команды <code>run</code>	14
4.10	Установка брейкпоинта и запуск программы	14
4.11	Использование команд <code>disassemble</code> и <code>disassembly-flavor intel</code>	15
4.12	Включение режима псевдографики	16
4.13	Установление точек останова и просмотр информации о них	17
4.14	До использования команды <code>stepi</code>	18
4.15	После использования команды <code>stepi</code>	19
4.16	Просмотр значений переменных	20
4.17	Использование команды <code>set</code>	20
4.18	Вывод значения регистра в разных представлениях	21
4.19	Использование команды <code>set</code> для изменения значения регистра	21
4.20	Завершение работы GDB	22
4.21	Создание файла	22
4.22	Загрузка файла с аргументами в отладчик	23
4.23	Установление точки останова и запуск программы	23
4.24	Просмотр значений, введенных в стек	24
4.25	Написание кода подпрограммы	25
4.26	Запуск программы и проверка его вывода	25
4.27	Ввод текста программы из листинга 9.3	27
4.28	Создание и запуск исполняемого файла	27
4.29	Нахождение причины ошибки	28
4.30	Неверное изменение регистра	28
4.31	Исправление ошибки	29
4.32	Ошибка исправлена	29

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

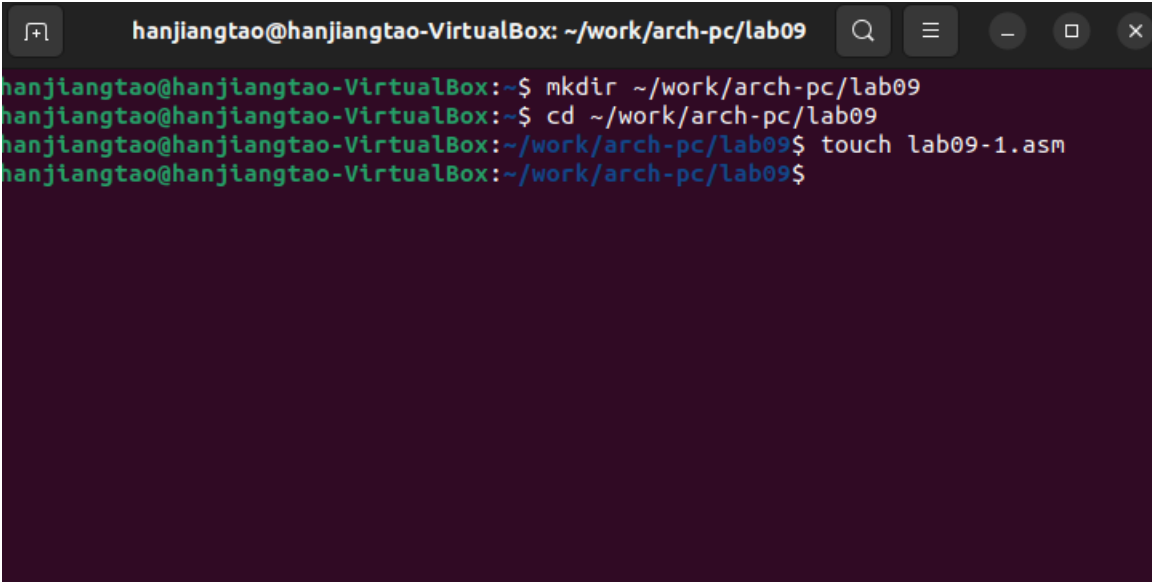
этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `esp` 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 `esp`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

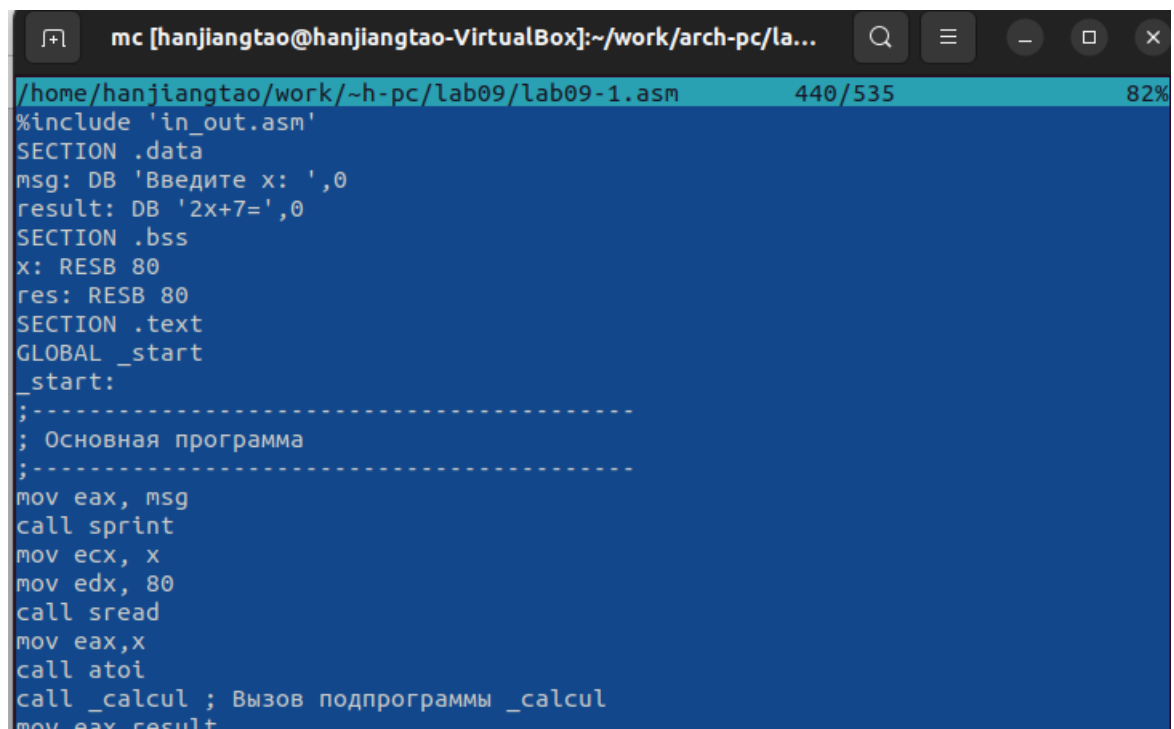
Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm.



```
hanjiangtao@hanjiangtao-VirtualBox: ~/work/arch-pc/lab09
hanjiangtao@hanjiangtao-VirtualBox:~$ mkdir ~/work/arch-pc/lab09
hanjiangtao@hanjiangtao-VirtualBox:~$ cd ~/work/arch-pc/lab09
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ touch lab09-1.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 4.1: Создание файлов для лабораторной работы

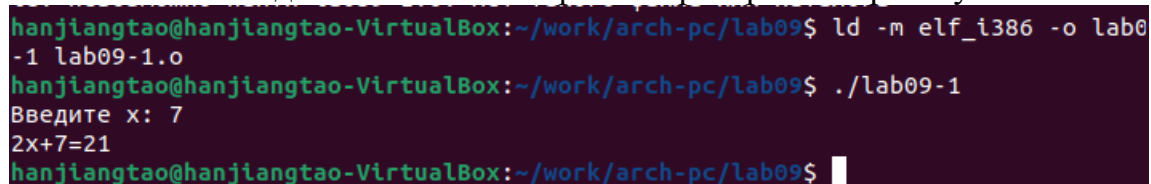
Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1.



```
mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/la...
/home/hanjiangtao/work/~h-pc/lab09/lab09-1.asm 440/535 82%
#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:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
```

Рис. 4.2: Ввод текста программы из листинга 9.1

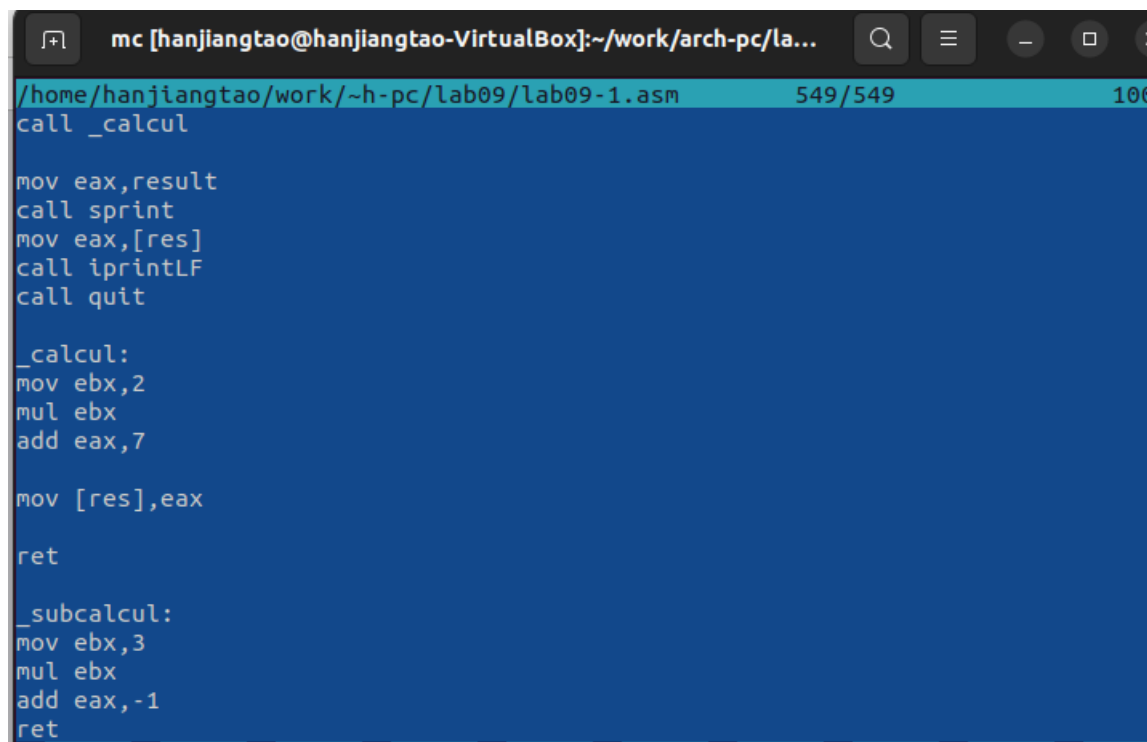
Создаю исполняемый файл и проверяю его работу.



```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab0
-1 lab09-1.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 7
2x+7=21
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$
```

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. 4.32)



```
mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/la...
/home/hanjiangtao/work/~h-pc/lab09/lab09-1.asm 549/549 100
call _calcul

mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit

_calcul:
mov ebx,2
mul ebx
add eax,7

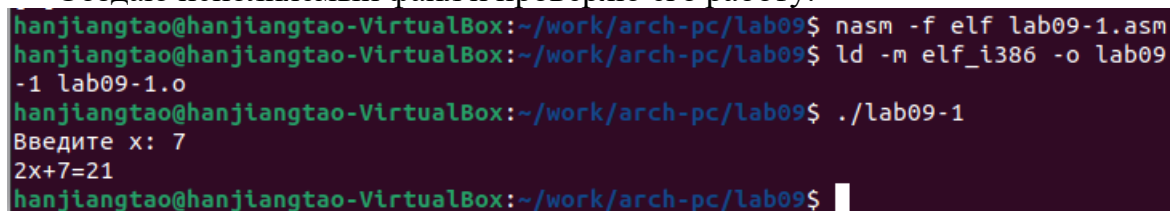
mov [res],eax

ret

_subcalcul:
mov ebx,3
mul ebx
add eax,-1
ret
```

Рис. 4.4: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу.



```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 7
2x+7=21
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$
```

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.

```
Терминал Вт, 20 февраля 23:42
mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/la...
/home/hanjiangtao/work/~h-pc/lab09/lab09-2.asm 277/298 92%
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2

SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
lnt 0x80

mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
lnt 0x80
```

Рис. 4.6: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. 4.32)

```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2 lab09-2.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 4.7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb.

```

hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ 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

```

(gdb) run
Starting program: /home/hanjiangtao/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3300) exited normally]
(gdb)

```

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды `run`.

Рис. 4.9: Проверка работы файла с помощью команды `run`

Для более подробного анализа программы устанавливаю брейкпоинт на метку `_start` и запускаю её.

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 10.
(gdb) run
Starting program: /home/hanjiangtao/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:10
10      mov eax, 4
(gdb)

```

ис. 4.10: Установка брейкпоинта и запуск программы Просматриваю

дисассимилированный код программы с помощью команды

disassemble, начиная с метки _start, и переключаясь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel.

```
hanjiangtao@hanjiangtao-VirtualBox: ~/work/arch-pc/lab09
x08049025 <+37>:  mov    $0x7,%edx
x0804902a <+42>:  int     $0x80
x0804902c <+44>:  mov     $0x1,%eax
x08049031 <+49>:  mov     $0x0,%ebx
x08049036 <+54>:  int     $0x80
of assembler dump.
) set disassembly-flavor intel
) disassemble _start
of assembler code for function _start:
x08049000 <+0>:      mov     eax,0x4
x08049005 <+5>:      mov     ebx,0x1
x0804900a <+10>:     mov     ecx,0x804a000
x0804900f <+15>:     mov     edx,0x8
x08049014 <+20>:     int     0x80
x08049016 <+22>:     mov     eax,0x4
x0804901b <+27>:     mov     ebx,0x1
x08049020 <+32>:     mov     ecx,0x804a008
x08049025 <+37>:     mov     edx,0x7
x0804902a <+42>:     int     0x80
x0804902c <+44>:     mov     eax,0x1
x08049031 <+49>:     mov     ebx,0x0
x08049036 <+54>:     int     0x80
of assembler dump.
)
```

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.

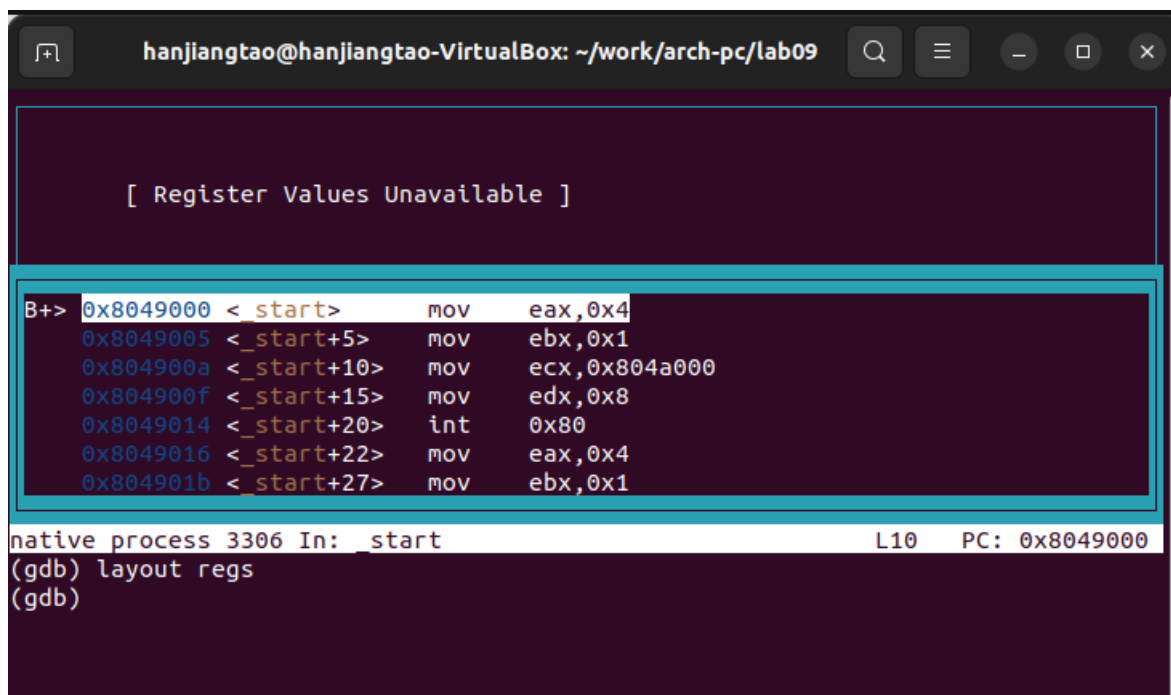


Рис.

4.12: 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.

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000  lab09-2.asm:10
breakpoint already hit 1 time
(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.

```
hanjiangtao@hanjiangtao-VirtualBox: ~/work/arch-pc/lab09

[ Register Values Unavailable ]

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 3306 In: _start                                L10    PC: 0x8049000
eax      0x0          0
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xffffd240   0xffffd240
ebp      0x0          0x0
esi      0x0          0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 4.14: До использования команды stepi

```
hanjiangtao@hanjiangtao-VirtualBox: ~/work/arch-pc/lab09

[ Register Values Unavailable ]

0x8049072    add    BYTE PTR [eax],al
0x8049074    add    BYTE PTR [eax],al
0x8049076    add    BYTE PTR [eax],al
0x8049078    add    BYTE PTR [eax],al
0x804907a    add    BYTE PTR [eax],al
0x804907c    add    BYTE PTR [eax],al
0x804907e    add    BYTE PTR [eax],al

native process 3306 In: _start L10 PC: 0x8049000
0x0 0
eip 0x8049000 0x8049000 <_start>
eflags 0x202 [ IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
--Type <RET> for more, q to quit, c to continue without paging--
```

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.

The screenshot shows a GDB terminal window with the title bar "hanjiangtao@hanjiangtao-VirtualBox: ~/work/arch-pc/lab09". The main display area shows a list of assembly instructions, each with an address, an opcode, and operands. The instructions are:

Address	Opcode	Operands
0x8049072	add	BYTE PTR [eax],al
0x8049074	add	BYTE PTR [eax],al
0x8049076	add	BYTE PTR [eax],al
0x8049078	add	BYTE PTR [eax],al
0x804907a	add	BYTE PTR [eax],al
0x804907c	add	BYTE PTR [eax],al
0x804907e	add	BYTE PTR [eax],al

Below the assembly list, the GDB prompt shows the current state of the process:

```

native process 3306 In: start L10 PC: 0x8049000
ds 0x2b 43
es 0x2b 43
--Type <RET> for more, q to quit, c to continue without paging--set {char}msg1='
h'fs 0x0 0
gs 0x0 0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 4.16: Просмотр значений переменных

С

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

```

0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hhello, "
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
0x804a008 <msg2>: "Lor d!\n"
(gdb)

```

помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2.

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`.

```
hanjiangtao@hanjiangtao-VirtualBox: ~/work/arch-pc/lab09
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x32     50
esp      0xffffd240 0xffffd240

0x804907e add BYTE PTR [eax],al
0x8049080 add BYTE PTR [eax],al
0x8049082 add BYTE PTR [eax],al
0x8049084 add BYTE PTR [eax],al
0x8049086 add BYTE PTR [eax],al

active process 3306 In: start L10 PC: 0x80490
s 0x2b 43
-Type <RET> for more, q to quit, c to continue without paging--set {char}
'fs 0x0 0
s 0x0 0
gdb) x/1sb &msg1
x804a000 <msg1>: "Hello, "
gdb) set $ebx='2'
gdb) p/s $ebx
1 = 50
gdb)
```

Рис. 4.18: Вывод значения регистра в разных представлениях

С помощью команды set изменяю значение регистра ebx в соответствии с заданием.

```
hanjiangtao@hanjiangtao-VirtualBox: ~/work/arch-pc/lab09
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x2      2
esp      0xffffd240 0xffffd240

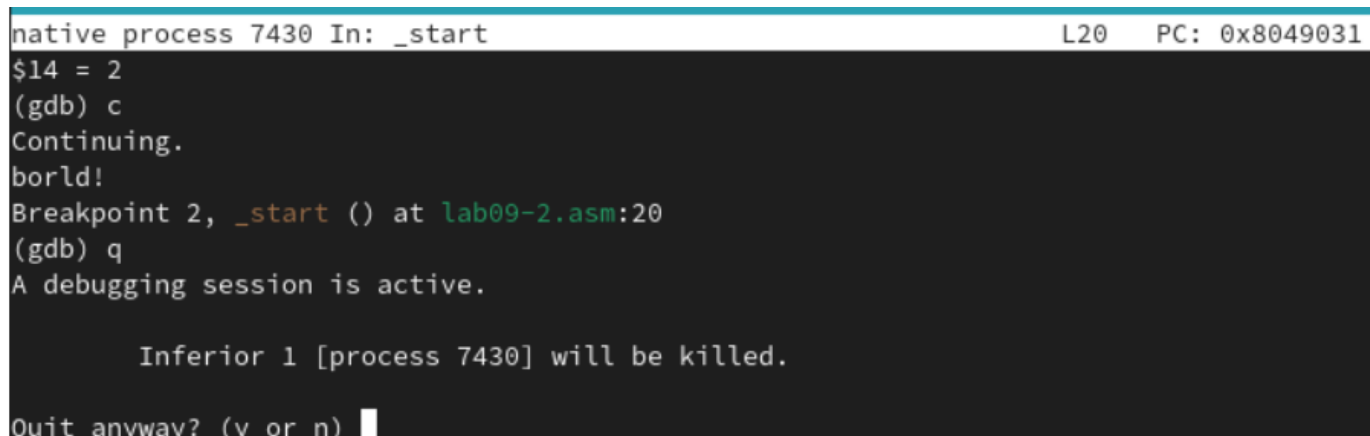
0x804907e  add    BYTE PTR [eax],al
0x8049080  add    BYTE PTR [eax],al
0x8049082  add    BYTE PTR [eax],al
0x8049084  add    BYTE PTR [eax],al
0x8049086  add    BYTE PTR [eax],al

native process 3306 In: start L10 PC: 0x80490
0x804a000 <msg1>: "Hello, "
gdb) set $ebx='2'
gdb) p/s $ebx
$1 = 50
gdb) $3 = 50
Undefined command: "$3". Try "help".
gdb) set $ebx=2
gdb) p/s $ebx
$2 = 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`.



```
native process 7430 In: _start                                L20    PC: 0x8049031
$14 = 2
(gdb) c
Continuing.
world!
Breakpoint 2, _start () at lab09-2.asm:20
(gdb) q
A debugging session is active.

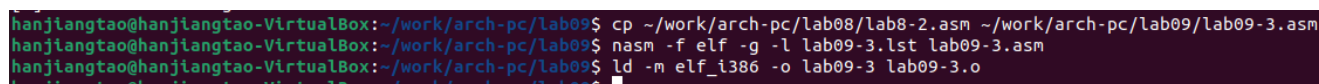
        Inferior 1 [process 7430] will be killed.

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.



```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab09$
```

Рис. 4.21: Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа `—args`.

```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/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.

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

```

(gdb) x/x $esp
0xffffd1f0:    0x00000005
(gdb) 0x05
Undefined command: "0x05".  Try "help".
(gdb) x/s *(void**)(esp + 4)
0xffffd3bd:    "/home/hanjiangtao/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd3ea:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd3fc:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd40d:    "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd40f:    "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 приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

6 Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005 — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM.— 2021.— URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix.— 2-е изд.— М.: МАКС Пресс, 2011.— URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).