

Отчёта по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Мантуров Татархан Бесланович

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
2.1	Реализация подпрограмм в NASM	5
2.2	Отладка программ с помощью GDB	7
3	Выводы	20

Список иллюстраций

2.1	Создаем каталог с помощью команды mkdir и файл с помощью команды touch	5
2.2	Заполняем файл	6
2.3	Запускаем файл и проверяем его работу	6
2.4	Изменяем файл, добавляя еще одну подпрограмму	7
2.5	Запускаем файл и смотрим на его работу	7
2.6	Создаем файл	7
2.7	Заполняем файл	8
2.8	Загружаем исходный файл в отладчик	8
2.9	Запускаем программу командой run	9
2.10	Запускаем программу с брейкпоинтом	9
2.11	Смотрим дисассимилированный код программы	9
2.12	Переключаемся на синтаксис Intel	10
2.13	Включаем отображение регистров, их значений и результат дисассимилирования программы	11
2.14	Используем команду info breakpoints и создаем новую точку останова	11
2.15	Смотрим информацию	12
2.16	Отслеживаем регистры	12
2.17	Смотрим значение переменной	12
2.18	Меняем символ	13
2.19	Смотрим значение регистра	13
2.20	Изменяем регистр командой set	14
2.21	Прописываем команды с и quit	14
2.22	Копируем файл	14
2.23	Создаем и запускаем в отладчике файл	15
2.24	Устанавливаем точку останова	15
2.25	Изучаем полученные данные	15
2.26	Копируем файл	16
2.27	Изменяем файл	16
2.28	Проверяем работу программы	16
2.29	Создаем файл	17
2.30	Изменяем файл	18
2.31	Создаем и смотрим на работу программы(работает неправильно)	18
2.32	Меняем файл	19
2.33	Создаем и запускаем файл(работает корректно)	19

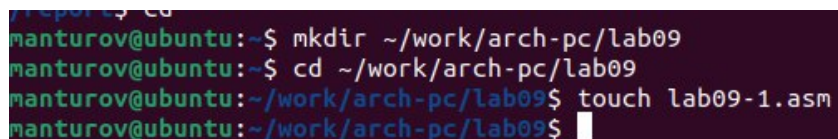
1 Цель работы

Познакомиться с методами отладки при помощи GDB, его возможностями.

2 Выполнение лабораторной работы

2.1 Реализация подпрограмм в NASM

Создаем каталог для программ ЛБ9, и в нем создаем файл (рис. 2.1).



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

Рис. 2.1: Создаем каталог с помощью команды `mkdir` и файл с помощью команды `touch`

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.1 (рис. 2.2).

```

%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

```

Рис. 2.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. 2.3).

```

manturov@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
manturov@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
manturov@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
bash: ./lab09-1: Нет такого файла или каталога
manturov@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 12
2x+7=31

```

Рис. 2.3: Запускаем файл и проверяем его работу

Снова открываем файл для редактирования и изменяем его, добавив подпрограмму в подпрограмму(по условию) (рис. 2.4).

```

mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _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 ; выход из подпрограммы
_subcalcul:
    mov ebx,3
    mul ebx
    add eax,1
    ret ; выход из подпрограммы

```

Рис. 2.4: Изменяем файл, добавляя еще одну подпрограмму

Создаем исполняемый файл и запускаем его (рис. 2.5).

```

manturov@ubuntu:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 12
2x+7=31

```

Рис. 2.5: Запускаем файл и смотрим на его работу

2.2 Отладка программ с помощью GDB

Создаем новый файл в каталоге(рис. 2.6).

```

manturov@ubuntu:~/work/arch-pc/lab09$ touch lab09-2.asm

```

Рис. 2.6: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.2 (рис. 2.7).

```

GNU nano 6.2 /home/manturov
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
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0

```

Рис. 2.7: Заполняем файл

Получаем исходный файл с использованием отладчика gdb (рис. 2.8).

```

manturov@ubuntu:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
manturov@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
manturov@ubuntu:~/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:

```

Рис. 2.8: Загружаем исходный файл в отладчик

Запускаем команду в отладчике (рис. 2.9).

```
(gdb) run
Starting program: /home/manturov/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 48554) exited normally]
(gdb)
```

Рис. 2.9: Запускаем программу командой run

Устанавливаем брейкпоинт на метку `_start` и запускаем программу (рис. 2.10).

```
[Inferior 1 (process 48554) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/manturov/work/arch-pc/lab09/lab09-2

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

Рис. 2.10: Запускаем программу с брейкпоином

Смотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. 2.11).

```
(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. 2.11: Смотрим дисассимилированный код программы

Переключаемся на отображение команд с Intel'овским синтаксисом (рис. 2.12).

```
(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. 2.12: Переключаемся на синтаксис Intel

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel:

1.Порядок операндов: В АТТ синтаксисе порядок операндов обратный, сначала указывается исходный операнд, а затем - результирующий операнд. В Intel синтаксисе порядок обычно прямой, результирующий операнд указывается первым, а исходный - вторым.

2.Разделители: В АТТ синтаксисе разделители операндов - запятые. В Intel синтаксисе разделители могут быть запятые или косые черты (/).

3.Префиксы размера операндов: В АТТ синтаксисе размер операнда указывается перед операндом с использованием префиксов, таких как "b" (byte), "w" (word), "l" (long) и "q" (quadword). В Intel синтаксисе размер операнда указывается после операнда с использованием суффиксов, таких как "b", "w", "d" и "q".

4.Знак операндов: В АТТ синтаксисе операнды с позитивными значениями предваряются символом "*Intel*".

5.Обозначение адресов: В АТТ синтаксисе адреса указываются в круглых скобках. В Intel синтаксисе адреса указываются без скобок.

6.Обозначение регистров: В АТТ синтаксисе обозначение регистра начинается с символа “%”. В Intel синтаксисе обозначение регистра может начинаться с символа “R” или “E” (например, “%eax” или “RAX”).

Включаем режим псевдографики (рис. 2.13).

The screenshot shows a debugger window with a dark background. At the top, a status bar indicates "[Register Values Unavailable]". Below this, a window titled "lab09-2.asm" displays assembly code. The code is as follows:

```

B+> 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
  
```

At the bottom of the window, a status bar shows "native process 48934 In: _start L9 PC: 0x8049000 (gdb)".

Рис. 2.13: Включаем отображение регистров, их значений и результат дисассимилирования программы

Проверяем была ли установлена точка останова и устанавливаем точку останова предпоследней инструкции (рис. 2.14).

The screenshot shows a debugger window with a dark background. At the top, a status bar indicates "[Register Values Unavailable]". Below this, a window titled "lab09-2.asm" displays assembly code. The code is as follows:

```

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
  
```

At the bottom of the window, a status bar shows "native process 25237 In: _start L9 PC: 0x8049000 (gdb)". Below the status bar, the following commands and their outputs are shown:

```

(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address          What
1      breakpoint       keep y 0x808049000 lab09-2.asm:9
      breakpoint already hit 1 time
(gdb)
  
```

Рис. 2.14: Используем команду info breakpoints и создаем новую точку останова

Посмотрим информацию о всех установленных точках останова (рис. 2.15).

```

native process 48934 In: _start L9
(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. 2.15: Смотрим информацию

Выполняем 5 инструкций командой si (рис. 2.16).

```

lab09-2.asm
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

native process 48934 In: _start L14 PC: 0x80
breakpoint keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.16: Отслеживаем регистры

Во время выполнения команд менялись регистры: ebx, ecx, edx, eax, eip.

Смотрим значение переменной msg1 по имени (рис. 2.17).

```

ebp      0x0      0x0
esi      0x0      0
--Type <RET> for more, q to quit, c to continue without
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 2.17: Смотрим значение переменной

Изменим первый символ переменной msg1 (рис. 2.18).

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

Рис. 2.18: Меняем символ

Смотрим значение регистра `edx` в разных форматах (рис. 2.19).

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

Рис. 2.19: Смотрим значение регистра

Изменяем регистр `ebx` (рис. 2.20).

```

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

```

Рис. 2.20: Изменяем регистр командой set

Выводятся разные значения, так как команда без кавычек присваивает регистру вводимое значение.

Прописываем команды для завершения программы и выхода из GDB (рис. 2.21).

```

(gdb) c
Continuing.
world!
[Inferior 1 (process 48934) exited normally]
(gdb) quit

```

Рис. 2.21: Прописываем команды c и quit

Копируем файл lab8-2.asm в файл с именем lab09-3.asm (рис. 2.22).

```

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

```

Рис. 2.22: Копируем файл

Создаем исполняемый файл и запускаем его в отладчике GDB (рис. 2.23).


```

manturov@ubuntu:~/work/arch-pc/lab09$ gdb --args lab09-3 1 2 '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...

```

Рис. 2.23: Создаем и запускаем в отладчике файл

Установим точку останова перед первой инструкцией в программе и запустим ее (рис. 2.24).

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/manturov/work/arch-pc/lab09/lab09-3 1 2 3

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

```

Рис. 2.24: Устанавливаем точку останова

Смотрим позиции стека по разным адресам (рис. 2.25).

```

(gdb) x/s *(void**)(($esp + 4)
0xffffd309:  "/home/manturov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(($esp + 8)
0xffffd333:  "1"
(gdb) x/s *(void**)(($esp + 12)
0xffffd335:  "2"
(gdb) x/s *(void**)(($esp + 16)
0xffffd337:  "3"
(gdb) x/s *(void**)(($esp + 20)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 2.25: Изучаем полученные данные

Шаг изменения адреса равен 4 потому что адресные регистры имеют размерность 32 бита(4 байта).

##Задание для самостоятельной работы

###Задание 1

Копируем файл lab8-4.asm(ср №1 в ЛБ8) в файл с именем samos9.asm (рис. 2.26).

```
manturov@ubuntu:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/samos9.asm
```

Рис. 2.26: Копируем файл

Открываем файл в Midnight Commander и меняем его, создавая подпрограмму (рис. 2.27).

```
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,10
    mov ebx,3
    mul ebx
    mov [res],eax
    ret
```

Рис. 2.27: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. 2.28).

```
manturov@ubuntu:~/work/arch-pc/lab09$ nasm -f elf samos9.asm
manturov@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o samos9 samos9.o
manturov@ubuntu:~/work/arch-pc/lab09$ ./samos9
Введите x: 12
3(10+x)=66
```

Рис. 2.28: Проверяем работу программы

###Задание 2

Создаем новый файл в директории (рис. 2.29).



Рис. 2.29: Создаем файл

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 9.3 (рис. 2.30).

```

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 2.30: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. 2.31).

```

manturov@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
manturov@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
manturov@ubuntu:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10

```

Рис. 2.31: Создаем и смотрим на работу программы(работает неправильно)

Изменяем программу для корректной работы (рис. 2.32).

```
GNU nano 6.2 /home/manturov/work/arch-pc/lab09/lab09-5.asm
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
    mov eax,3
    mov ebx,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
```

Рис. 2.32: Меняем файл

Создаем исполняемый файл и запускаем его (рис. 2.33).

```
manturov@ubuntu:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
manturov@ubuntu:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-5 lab09-5.o
manturov@ubuntu:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
manturov@ubuntu:~/work/arch-pc/lab09$
```

Рис. 2.33: Создаем и запускаем файл(работает корректно)

3 Выводы

Мы познакомились с методами отладки при помощи GDB и его возможностями.