

Отчёта по лабораторной работе №9

Архитектура компьютера

Еремина Оксана Андреевна НКАбд-02-23

Содержание

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

Список таблиц

Список иллюстраций

4.1	Создание папки и файла	8
4.2	Содержание файла lab09-1.asm	8
4.3	Выподнение программы	9
4.4	Изменение программы	9
4.5	Выподнение программы	9
4.6	Содержание файла lab09-2.asm	10
4.7	Выполнение программы	10
4.8	Анализ программы	11
4.9	Дисассимилированный код программы	11
4.10	Режим псевдографики	12
4.11	Добавление точек останова	12
4.12	Значение переменных msg1 иmsg2	13
4.13	Замена символов переменных msg1 иmsg2	13
4.14	Значения регистра edx	14
4.15	Изменение регистра edx	14
4.16	Действия с файлом lab8-2.asm	15
4.17	15
4.18	Вершины стека	15
4.19	Измененный код программы из лабораторной работы №8	16

1 Цель работы

Целью данной лабораторной работы является приобретение навыков в 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. Обработка аргументов командной строки в GDB
4. Выполнение заданий 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. Пошаговое выполнение — это выполнение программы с остановкой после каждой строки, чтобы программист мог проверить значения переменных и выполнить другие действия.

Точки останова — это специально отмеченные места в программе, в которых программаотладчик приостанавливает выполнение программы и ждёт команд.

GDB (GNU Debugger — отладчик проекта GNU) [1] 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.

4 Выполнение лабораторной работы

1. Реализация подпрограмм в NASM

Создаю каталог для программ лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm. (рис.1)

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

Рис. 4.1: Создание папки и файла

Ввожу в файл lab09-1.asm текст программы. (рис.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. 4.2: Содержание файла lab09-1.asm

Создаю исполняемый файл и проверяю работу программы. (рис.3)


```

oaoeremina@oaoeremina:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
oaoeremina@oaoeremina:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
oaoeremina@oaoeremina:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17

```

Рис. 4.3: Выподнение программы

Изменяю код 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)

```

20 call atoi
21 call _subcalcul ; Вызов подпрограммы _calcul
22 call _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
37
38 _subcalcul:
39 mov ebx,3
40 mul ebx
41 add eax,-1
42 ret

```

Рис. 4.4: Изменение программы

Создаю исполняемый файл и проверяю работу программы. (рис.5)

```

oaoeremina@oaoeremina:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
oaoeremina@oaoeremina:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
oaoeremina@oaoeremina:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=35

```

Рис. 4.5: Выподнение программы

2. Отладка программ с помощью GDB

Создаю файл `lab09-2.asm` в каталоге `~/work/arch-pc/lab08`. Ввожу код программы из листинга 9.2 в созданный файл. (рис.6)

```

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: Содержание файла lab09-2.asm

Получаю исполняемый файл для работы с GDB с ключом '-g'. Загружаю исполняемый файл в отладчик gdb. Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис.7)

```

oaeremina@oaeremina:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
oaeremina@oaeremina:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
oaeremina@oaeremina:~/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) run
Starting program: /home/oaeremina/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 10594) exited normally]
(gdb)

```

Рис. 4.7: Выполнение программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start и запускаю её. (рис.8)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/oaeremina/work/arch-pc/lab09/lab09-2

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

Рис. 4.8: Анализ 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.9)

```
(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) 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.
```

Рис. 4.9: Дисассимилированный код программы

Включаю режим псевдографики для более удобного анализа программы с помощью команд `layout asm` и `layout regs`. (рис.10)

```

[ Register Values Unavailable ]

0x804911a    add    BYTE PTR [eax],al
0x804911c    add    BYTE PTR [eax],al
0x804911e    add    BYTE PTR [eax],al
0x8049120    add    BYTE PTR [eax],al
0x8049122    add    BYTE PTR [eax],al
0x8049124    add    BYTE PTR [eax],al

native process 10675 In: start          L9    PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.10: Режим псевдографики

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.11)

```

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)

```

Рис. 4.11: Добавление точек останова

Работа с данными программы в GDB

Выполните 5 инструкций с помощью команды `stepi` (или `si`). (рис.12-16)

Register group: general	Register group: general
eax 0x4 4	eax 0x4 4
ecx 0x0 0	ecx 0x0 0
edx 0x0 0	edx 0x0 0
ebx 0x0 0	ebx 0x1 1
esp 0xffffd0b0 0xffffd0b0	esp 0xffffd0b0 0xffffd0b0
ebp 0x0 0	ebp 0x0 0
esi 0x0 0	esi 0x0 0
edi 0x0 0	edi 0x0 0

B+ 0x8049000 < _start>	B+ 0x8049000 < _start>
mov \$0x4,%eax	mov \$0x4,%eax
> 0x8049005 < _start+5> mov \$0x1,%ebx	0x8049005 < _start+5> mov \$0x1,%ebx
0x804900a < _start+10> mov \$0x804a000,%ecx	> 0x804900a < _start+10> mov \$0x804a000,%ecx
0x804900f < _start+15> mov \$0x8,%edx	0x804900f < _start+15> mov \$0x8,%edx
0x8049014 < _start+20> int \$0x80	0x8049014 < _start+20> int \$0x80
0x8049016 < _start+22> mov \$0x4,%eax	0x8049016 < _start+22> mov \$0x4,%eax
0x804901b < _start+27> mov \$0x1,%ebx	0x804901b < _start+27> mov \$0x1,%ebx
0x8049020 < _start+32> mov \$0x804a008,%ecx	0x8049020 < _start+32> mov \$0x804a008,%ecx

```

--Register group: general--
eax      0x4      4
ecx      0x804a000 134520832
edx      0x0      0
ebx      0x1      1
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

8+ 0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
> 0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx

--Register group: general--
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

8+ 0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
0x8049014 <_start+20> int $0x80
> 0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx

```

```

--Register group: general--
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd0b0 0xffffd0b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0

8+ 0x8049000 <_start> mov $0x4,%eax
0x8049005 <_start+5> mov $0x1,%ebx
0x804900a <_start+10> mov $0x804a000,%ecx
0x804900f <_start+15> mov $0x8,%edx
> 0x8049014 <_start+20> int $0x80
0x8049016 <_start+22> mov $0x4,%eax
0x804901b <_start+27> mov $0x1,%ebx
0x8049020 <_start+32> mov $0x804a008,%ecx

```

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.17)

```

(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.12: Значение переменных msg1 и msg2

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис.18)

```

(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='d'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "dorld!\n\034"

```

Рис. 4.13: Замена символов переменных msg1 и msg2

Вывожу значение регистра `edx` в шестнадцатеричном, двоичном формате и в символьном виде. (рис. 19)

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

Рис. 4.14: Значения регистра `edx`

С помощью команды `set` изменяю значение регистра `ebx`. (рис.20)

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

Рис. 4.15: Изменение регистра `edx`

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.

Выхожу из GDB с помощью команды `quit`.

3. Обработка аргументов командной строки в GDB

Копируйте файл `lab8-2.asm`, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем `lab09-3.asm`. Создаю исполняемый файл. Загружаю исполняемый файл в отладчик `gdb`. (рис.21)

```

oaeremina@oaeremina:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/
lab09/lab09-3.asm
oaeremina@oaeremina:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
oaeremina@oaeremina:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
oaeremina@oaeremina:~/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. 4.16: Действия с файлом lab8-2.asm

Устанавливаю точку останова перед первой инструкцией в программе и запускаем ее. Адрес вершины стека, который храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки. (рис.22)

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/oaeremina/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\
3

Breakpoint 1, _start () at lab09-3.asm:5
5   pop еск ; Извлекаем из стека в 'еск' количество
(gdb) x/x $esp
0xffffd070: 0x00000005
(gdb)

```

Рис. 4.17:

Посматриваю вершину стека и позиции стека по их адресам. (рис.23)

```

(gdb) x/s *(void**)(esp + 4)
0xffffd243: "/home/oaeremina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd26e: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd280: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd291: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd293: "аргумент 3"

```

Рис. 4.18: Вершины стека

4. Выполнение заданий для самостоятельной работы
5. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. Создаю исполняемый файл и выполняю программу. (рис.24)

```
oaeremina@oaeremina:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/
lab09/lab09-4.asm
oaeremina@oaeremina:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
oaeremina@oaeremina:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
oaeremina@oaeremina:~/work/arch-pc/lab09$ ./lab09-4 1 3 5
Результат: 60
```

Рис. 4.19: Измененный код программы из лабораторной работы №8

Код программы:

```
%include 'in_out.asm'
SECTION .data
msg db "Результат:",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
mov edi,10
next:
pop eax
call atoi
sub eax,1
mul edi
add esi,eax
```



```
cmp ecx,0h
jz .done
loop next
.done:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
ret
```

5 Вывод

При выполнении данной лабораторной работы я приобрела практические навыки написания программ с использованием подпрограмм, познакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы

Архитектура ЭВМ