

Отчет по лабораторной работе №8

Дисциплина: Архитектура компьютера

Стрижов Дмитрий Павлович

Содержание

1	Цель работы	3
2	Задание	4
3	Выполнение лабораторной работы	5
3.1	Реализация подпрограмм в NASM	5
3.2	Отладка программ с помощью GDB	6
3.3	Задание для самостоятельной работы	11
4	Выводы	14
	Список литературы	15

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. Задание для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm (рис. 3.1).

```
[dpstrizhov@fedora ~]$ mkdir ~/work/arch-pc/lab09
cd ~/work/arch-pc/lab09
touch lab09-1.asm
[dpstrizhov@fedora lab09]$
```

Рис. 3.1: Предварительная подготовка

Копирую программу из листинга 1, добавляю подпрограмму _subcalcul, которая вычисляет функцию $g(x)$ (рис. 3.2).

```
_calcul:
call _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
_subcalcul:
mov ebx,3
mul ebx
add eax,-1
ret
ret ; выход из подпрограммы
```

Рис. 3.2: Реализация подпрограммы _subcalcul

Вот что получается в итоге (рис. 3.3).

```
[dpstrizhov@fedora lab09]$ ./lab09-1
Введите x: 1
2(3x-1)+7=11
[dpstrizhov@fedora lab09]$
```

Рис. 3.3: Вывод программы с подпрограммой `_subcalcul`

3.2 Отладка программ с помощью GDB

Создаю файл `lab09-2.asm`, куда вписываю программу из листинга 2 (рис. 3.4, рис. 3.5).

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

Рис. 3.4: Создание `lab09-2.asm`

```
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
int 0x80
```

Рис. 3.5: Содержимое `lab09-2.asm`

Получаю исполняемый файл, при этом указывая ключ “-g” для работы с GDB (рис. 3.6).

```
[dpstrizhov@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ld -m elf_i386 -o lab09-2 lab09-2.o
[dpstrizhov@fedora lab09]$
```

Рис. 3.6: Подготовка для работы с отладчиком

Загружаю исполняемый файл в отладчик (рис. 3.7).

```
[dpstrizhov@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.2-5.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 lab09-2...
(gdb) █
```

Рис. 3.7: Загрузка исполняемого файла в отладчик

Проверяю работы программы (рис. 3.8).

```
(gdb) run
Starting program: /home/dpstrizhov/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 4181) exited normally]
```

Рис. 3.8: Проверка раоты программы

Ставим метку и снова запускаем программу (рис. 3.9).

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

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

Рис. 3.9: Проверка программу с меткой

Просматриваю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 3.10).

```
(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.
```

Рис. 3.10: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`, различия отображения в синтаксисе АТТ, а именно знак `%` перед регистром и регистры при написании программ меняются местами, например, в при выполнении команды `mov` значения из первого регистра переходят во второй (рис. 3.11).

```
(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. 3.11: Переключение на отображение команд с Intel'овским синтаксисом

Включаю режим псевдографики для более удобного анализа программы (рис. 3.12).



Рис. 3.12: Режим псевдографики

Проверяю точку останова (рис. 3.13).

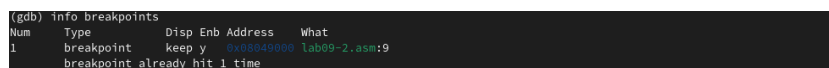


Рис. 3.13: Проверка точки останова

Установка новой точки останова (рис. 3.14).

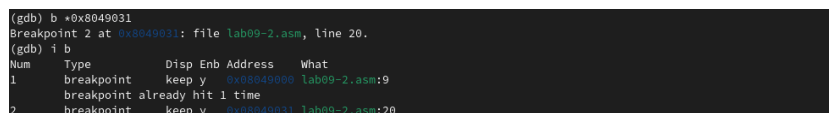


Рис. 3.14: Новая точка останова

Смотрю содержимое переменной msg1 (рис. 3.15).



Рис. 3.15: Содержимое переменной msg1

Смотрю значение переменной msg2 (рис. 3.16).

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

Рис. 3.16: Значение переменной msg2

Изменяем значения переменной msg1 (рис. 3.17).

```
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
0x804a005 <msg2>: "world!\n\034"
```

Рис. 3.17: Измененная переменная msg1

Делаю то же самое с переменной msg2 (рис. 3.18).

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

Рис. 3.18: Измененная переменная msg2

Меняю значение регистра ebx (рис. 3.19).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
```

Рис. 3.19: Регистр ebx

И снова, причина разницы значений заключается в том, что в первом случае мы передаем строку, а во втором число (рис. 3.20).

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

Рис. 3.20: Изменения регистра ebx

Копирую 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(рис. 3.21).

```
[dpstrizhov@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[dpstrizhov@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 3.21: Подготовка к дальнейшей работе

Загружаю программу в отладчик, устанавливаю точку останова и запускаю программу (рис. 3.22).

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/dpstrizhov/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в 'ecx' количество
```

Рис. 3.22: Работа переданной программы в отладчике

Смотрим, что находится в регистре esp (стек) (рис. 3.23).

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

Рис. 3.23: Регистр esp

Рассматриваем прочие значения стека (рис. 3.24).

```
(gdb) x/s *(void**)(esp + 4)
A syntax error in expression, near `'.
(gdb) x/s *(void**)(esp + 4)
0xffffd010: "/home/dpstrizhov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd047: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd059: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd06a: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd06c: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 3.24: Прочие значения стека

Знак изменения равен 4, вероятно, потому, что адрес записывается в 16-тиричной системе исчисления.

3.3 Задание для самостоятельной работы

1. Меняю программу, добавляя подпрограмму, которая считает функцию отдельно (рис. 3.25).

```

next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ
call summ
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
summ:
mov ebx, 6
mul ebx
add eax, 13
add esi,eax ; добавляем к промежуточной сумме
ret;

```

Рис. 3.25: Подпрограмма summ

Получаю следующий результат (рис. 3.26).

```

[dpstrizhov@fedora lab09]$ ./task1 1 2 3 4
Функция: f(x)=6x + 13
Результат: 112
[dpstrizhov@fedora lab09]$

```

Рис. 3.26: Результат вычислений

2. Меняю программу из листинга, чтобы выводился правильный результат (рис. 3.27).

```
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 3.27: Исправленная программа

Проверяю её исполнение и получаю правильный результат (рис. 3.28).

```
[dpstrizhov@fedora lab09]$ ./task2
Результат: 25
[dpstrizhov@fedora lab09]$
```

Рис. 3.28: Результат исправленной программы

4 Выводы

За время выполнения работы я получил навыки написания программ с использованием подпрограмм и познакомился с методами отладки с помощью GDB и его основными особенностями.

Список литературы

Debugging assembly with GDB. Источник: <https://ncona.com/2019/12/debugging-assembly-with-gdb/>