

Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Кудинец Максим Антонович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация программ в NASM	8
4.2	Отладка программ с помощью GDB	11
4.3	Добавление точек останова	15
4.4	Работа с данными программы в GDB	16
4.5	Обработка аргументов командной строки в GDB	21
5	Задания для самостоятельной работы	23
6	Выводы	28
	Список литературы	29

Список иллюстраций

4.1	Переход в каталог и создание файла	8
4.2	Программа с использованием подпрограммы	9
4.3	Исполнение программы из листинга 9.1	10
4.4	Исправленный текст программы lab09-1.asm	10
4.5	Исполнение программы lab09-1	11
4.6	Текст программы из листинга 9.2	12
4.7	Исполнение программы	13
4.8	Запуск программы в оболочке отладчика	13
4.9	Исполнение программы с брейкпоинт	13
4.10	Просмотр дисассимилированного кода программы	14
4.11	Просмотр дисассимилированного кода программы с синтаксисом Intel	14
4.12	Переход в режим псевдографики	15
4.13	Проверка установки точки останова	15
4.14	Установка новой точки останова	16
4.15	Инструкция stepi	17
4.16	Просмотр значения переменной	18
4.17	Просмотр значения переменной	19
4.18	Изменение переменной	19
4.19	Вывод значения регистра	20
4.20	Изменение значения регистра	20
4.21	Загрузка файла lab09-3.asm в отладчик	21
4.22	Проверка стека	22
4.23	Проверка остальных позиций стека	22
5.1	Текст программы lab09-4.asm	23
5.2	Запуск программы	24
5.3	Поиск ошибок в работе программы lab09-5.asm	25
5.4	Запуск исправленной программы	26
5.5	Исправленный текст программы	27

Список таблиц

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм, а также знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

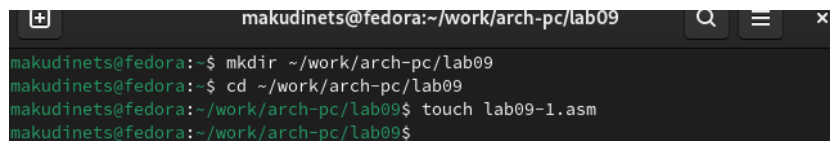
1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $\varphi(x)$ как подпрограмму.
2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \cdot 4 + 5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

3 Теоретическое введение

4 Выполнение лабораторной работы

4.1 Реализация программ в NASM

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



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

Рис. 4.1: Переход в каталог и создание файла

2. В качестве примера рассмотрим программу вычисления арифметического выражения $\square(\square) = 2\square + 7$ с помощью подпрограммы `_calcul`. В данном примере \square вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Запускаю исполняемый файл.


```

%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
call sprint
mov eax,[res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret

```

Рис. 4.2: Программа с использованием подпрограммы

```

makudinets@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
makudinets@fedora:~/work/arch-pc/lab09$ ld -m elf-i386 lab09-1.o -o lab09-1
ld: не распознан режим эмуляции: elf-i386
Поддерживаемые эмуляции: elf_x86_64 elf32_x86_64 elf_i386 elf_iamcu i386pep i386
pe elf64bpf
makudinets@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 lab09-1.o -o lab09-1
makudinets@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
makudinets@fedora:~/work/arch-pc/lab09$

```

Рис. 4.3: Исполнение программы из листинга 9.1

3. Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\square(\square(\square))$, где \square вводится с клавиатуры, $\square(\square) = 2\square + 7$, $\square(\square) = 3\square - 1$. Запустим исправленную программу. Число проходов цикла не соответствует значению, введенному с клавиатуры.

```

;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret
; Подпрограмма subcalcul
_subcalcul:
mov, ebx, 3
mul ebx
sub, eax, 1
mov [res], eax
ret

```

Рис. 4.4: Исправленный текст программы lab09-1.asm

```
makudinets@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
makudinets@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 lab09-1.o -o lab09-1
makudinets@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
makudinets@fedora:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 6
2x+7=19
makudinets@fedora:~/work/arch-pc/lab09$
```

Рис. 4.5: Исполнение программы lab09-1

4.2 Отладка программ с помощью GDB

4. Создаем файл lab09-2.asm. Вводим в него программу из листинга 9.2. Транслируем текст программы с ключом '-g'. Загружаем исполняемый файл в gdb.

```

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. 4.6: Текст программы из листинга 9.2

```

makudinets@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
makudinets@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
makudinets@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 15.2-1.fc40
Copyright (C) 2024 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. 4.7: Исполнение программы

5. Проверим работу программы, запустив ее в оболочке отладчика. Ошибок не обнаружено.

```

(gdb) run
Starting program: /home/makudinets/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 6500) exited normally]
(gdb)

```

Рис. 4.8: Запуск программы в оболочке отладчика

6. Для более подробного анализа программы установим брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустим её.

```

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

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

```

Рис. 4.9: Исполнение программы с брейкпоинт

7. Посмотрим дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`.

```
(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. 4.10: Просмотр дисассимилированного кода программы

8. Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel`. Различия между синтаксисом АТТ и Intel заключаются в порядке операндов (АТТ - Операнд источника указан первым. Intel - Операнд назначения указан первым), их размере (АТТ - размер операндов указывается явно с помощью суффиксов, непосредственные операнды предваряются символом `$`; Intel - Размер операндов неявно определяется контекстом, как `ax`, `eax`, непосредственные операнды пишутся напрямую), именах регистров (АТТ - имена регистров предваряются символом `%`, Intel - имена регистров пишутся без префиксов).

```
(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: Просмотр дисассимилированного кода программы с синтаксисом Intel

9. Включим режим псевдографики для более удобного анализа программы.

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al

native process 6814 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 4.12: Переход в режим псевдографики

4.3 Добавление точек останова

10. Проверим установку точки останова на метке '_start'.

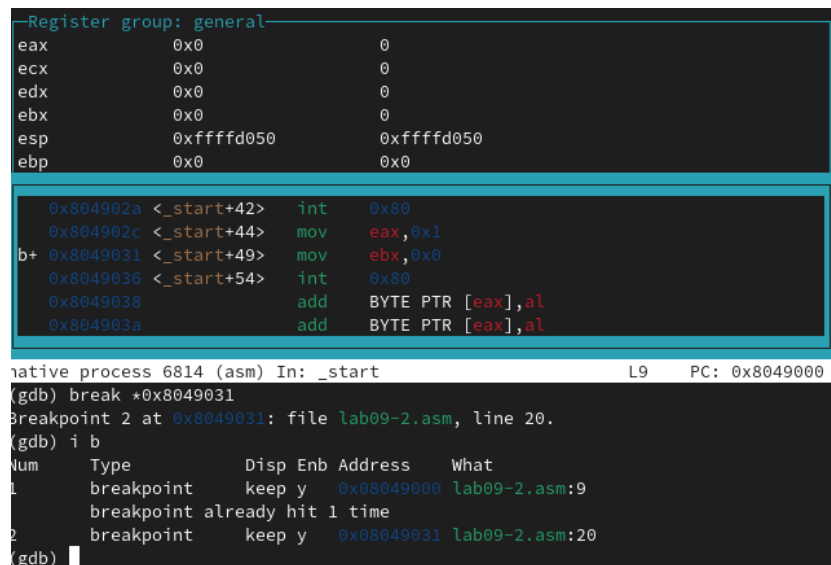
```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al

native process 6814 (asm) In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num    Type             Disp Enb Address      What
1      breakpoint       keep y  0x08049000 lab09-2.asm:9
       breakpoint already hit 1 time
(gdb)
```

Рис. 4.13: Проверка установки точки останова

11. Установим еще одну точку останова по адресу инструкции. Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова.



The screenshot shows the GDB interface with the following content:

```
--Register group: general--
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

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
0x804903a add BYTE PTR [eax],al

native process 6814 (asm) In: _start L9 PC: 0x8049000
(gdb) break *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. 4.14: Установка новой точки останова

4.4 Работа с данными программы в GDB

12. Выполним 5 инструкций с помощью команды stepi (или si) и проследим за изменением значений регистров. Изменяются значения регистров eax, ebx, ecx, edx.


```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

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

native process 6814 (asm) In: _start L14 PC: 0x8049016
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
```

Рис. 4.15: Инструкция stepi

13. Посмотрим значение переменной msg1 по имени.

Рис. 4.16: Просмотр значения переменной

14. Просмотрим значение переменной `msg2` по адресу.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

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

native process 6814 (asm) In: _start L14 PC: 0x8049016
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb)

```

Рис. 4.17: Просмотр значения переменной

15. Изменим первый символ переменной msg1. Заменяем любой символ во второй переменной msg2.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

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

native process 6814 (asm) In: _start L14 PC: 0x8049016
(gdb) stepi
(gdb) stepi
(gdb) stepi
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.18: Изменение переменной

16. Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. С помощью команды set изменим значение регистра ebx. В первом случае программа

выводит значение кодировки символа '2' в шестнадцатеричной системе, а во втором переводит цифру 2 в шестнадцатеричный вид. Завершаю выполнение программы и выхожу из отладчика.

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

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

native process 6814 (asm) In: _start L14 PC: 0x8049016
0x804a000 <msg1>: "Hello, "
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}0x804a008='L'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lor!d!\n\034"
(gdb)

```

Рис. 4.19: Вывод значения регистра

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

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

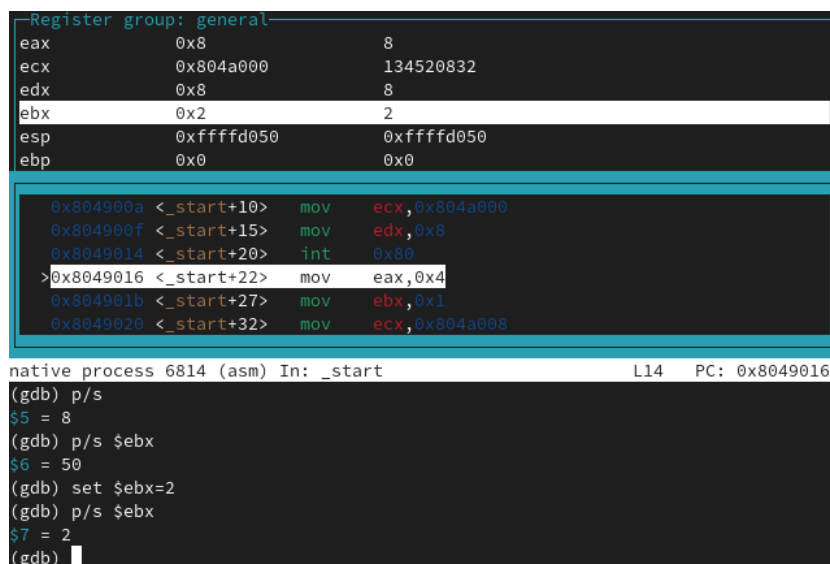
native process 6814 (asm) In: _start L14 PC: 0x8049016
$1 = 100000000100101000000000000000
(gdb) p/s $edx
$2 = 8
(gdb) p/t $edx
$3 = 1000
(gdb) p/x $edx
$4 = 0x8
(gdb)

```

Рис. 4.20: Изменение значения регистра

4.5 Обработка аргументов командной строки в GDB

17. Скопируем файл lab8-2.asm в файл с именем lab09-3.asm. Создадим исполняемый файл. Загрузим исполняемый файл в отладчик, указав аргументы.



The screenshot shows the GDB interface with the following content:

```
Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd050 0xffffd050
ebp      0x0      0x0

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

native process 6814 (asm) In: _start L14 PC: 0x8049016
(gdb) p/s
$5 = 8
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb)
```

Рис. 4.21: Загрузка файла lab09-3.asm в отладчик

18. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

```

makudinets@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
makudinets@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
makudinets@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
makudinets@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 15.2-1.fc40
Copyright (C) 2024 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 lab09-3...
(gdb)

```

Рис. 4.22: Проверка стека

19. Посмотрим остальные позиции стека – по адресу [esp+4] располагается адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. Шаг изменения адреса равен 4 байтам, потому что мы работаем с 32-битной системой (x86), а указатели (void **) в такой системе занимают 4 байта. Ошибка Cannot access memory at address 0x0 на \$esp + 24 указывает на то, что закончились аргументы командной строки.

```

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

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx          ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd010: 0x00000005
(gdb)

```

Рис. 4.23: Проверка остальных позиций стека

5 Задания для самостоятельной работы

1. Преобразуем программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $\Pi(\Pi)$ как подпрограмму.

```
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx          ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd010: 0x00000005
(gdb) x/s *(void**)(esp+4)
0xffffd1d0: "/home/makudinets/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp+8)
0xffffd1fc: "аргумент1"
(gdb) x/s *(void**)(esp+12)
0xffffd20e: "аргумент"
(gdb) x/s *(void**)(esp+16)
0xffffd21f: "2"
(gdb) x/s *(void**)(esp+20)
0xffffd221: "аргумент 3"
(gdb) x/s *(void**)(esp+24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 5.1: Текст программы lab09-4.asm

```

%include 'in_out.asm'

SECTION .data
msg1 db "Функция: f(x) = 5*(2+x)",0
msg2 db "Результат: ",0

SECTION .text
global _start

_start:
    pop ecx      ; Извлекаем из стека в ecx количество аргументов
    pop edx      ; Извлекаем из стека в edx имя программы
    sub ecx, 1    ; Уменьшаем ecx на 1 (количество аргументов без названия программы)
    mov esi, 0    ; Используем esi для хранения промежуточных сумм

next:
    cmp ecx, 0h   ; проверяем, есть ли еще аргументы
    jz _end       ; если аргументов нет выходим из цикла

    pop eax       ; извлекаем следующий аргумент из стека
    push ecx      ; сохраняем ecx
    call atoi     ; преобразуем символ в число
    push eax      ; сохраняем eax
    call _calcul   ; вызов подпрограммы
    add esi, eax   ; суммируем результат с текущим значением esi
    pop eax       ; восстанавливаем eax
    pop ecx       ; восстанавливаем ecx
    loop next     ; переход к обработке следующего аргумента

_end:
    mov eax, msg1 ; вывод первого сообщения
    call sprint
    mov eax, msg2 ; вывод второго сообщения
    call sprint
    mov eax, esi  ; записываем сумму в регистр eax
    call iprintfLF ; печать результата
    call quit     ; завершение программы

; Подпрограмма вычисления выражения "5*(2+x)"
_calcul:
    push ebp      ; Сохраняем ebp на стеке
    mov ebp, esp  ; Устанавливаем ebp как указатель на начало локальной области
    mov ebx, [ebp+8] ; x находится на [ebp+8] (первый аргумент)
    add ebx, 2     ; 2 + x
    mov eax, 5
    mul ebx        ; 5 * (2 + x)
    pop ebp        ; восстанавливаем ebp
    ret

```

Рис. 5.2: Запуск программы

2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \cdot 4 + 5$. При запуске данная программа дает неверный результат. Ошибка в программе заключается в том, что инструкция `mul ecx` умножает значение в регистре `eax` на `ecx`, а результат записывает в `eax`. В исправленном варианте мы используем `ebx` для хранения промежуточного результата

суммы, `mul ecx` умножает `ebx` на `ecx`, результат сохраняется в `eax`. Затем к результату в `eax` добавляется 5. Финальный результат сохраняется в `edi` и выводится на экран. Запускаем программу в режиме отладчика и пошагово через `si` просматриваем изменение значений регистров через `i r`. При выполнении инструкции `mul ecx` можно заметить, что результат умножения записывается в регистр `eax`, но также меняет и `edx`. Значение регистра `ebx` не обновляется напрямую, поэтому программа неверно подсчитывает результат функции. Исправляем найденную ошибку, теперь программа верно считает значение функции.

The screenshot shows a GDB window titled "makudinets@fedora:~/work/arch-pc/lab09 — gdb lab09-5". The "Register group: general" section displays the following values:

Register	Value	Comment
eax	0x0	0
ecx	0x0	0
edx	0x0	0
ebx	0x0	0
esp	0xffffd060	0xffffd060
ebp	0x0	0x0

Below the registers, the assembly instructions are listed:

```

B> 0x80490e8 <_start> mov ebx,0x3
    0x80490ed <_start+5> mov eax,0x2
b+ 0x80490f2 <_start+10> add ebx,eax
    0x80490f4 <_start+12> mov ecx,0x4
    0x80490f9 <_start+17> mul ecx
    0x80490fb <_start+19> add ebx,0x5
  
```

The status bar at the bottom indicates: "native process 5668 (asm) In: _start L?? PC: 0x80490e8". The command prompt shows "Start it from the beginning? (y or n) y", "Downloading source file /usr/src/debug/kernel-6.11.4/linux-6.11.4-201.fc40.x86_64/arch/x86/entry/vdso/vdso32/vgetcpu.c", "Starting program: /home/makudinets/work/arch-pc/lab09/lab09-5", and "Breakpoint 1, 0x080490e8 in _start () (gdb)".

Рис. 5.3: Поиск ошибок в работе программы lab09-5.asm

```
makudinets@fedora:~/work/arch-pc/lab09
(gdb) set disassembler-flavor intel
No symbol "disassembler" in current context.
(gdb) set disassembly-flavor intel
(gdb) didassembler _start
Undefined command: "didassembler". Try "help".
(gdb) disassembler _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>:    mov     ebx,0x3
      0x080490ed <+5>:    mov     eax,0x2
      0x080490f2 <+10>:   add     ebx,eax
      0x080490f4 <+12>:   mov     ecx,0x4
      0x080490f9 <+17>:   mul     ecx
      0x080490fb <+19>:   add     ebx,0x5
      0x080490fe <+22>:   mov     edi,ebx
      0x08049100 <+24>:   mov     eax,0x804a000
      0x08049105 <+29>:   call    0x804900f <sprint>
      0x0804910a <+34>:   mov     eax,edi
      0x0804910c <+36>:   call    0x8049086 <iprintf>
      0x08049111 <+41>:   call    0x80490db <quit>
End of assembler dump.
(gdb) layout asm
makudinets@fedora:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
makudinets@fedora:~/work/arch-pc/lab09$
```

Рис. 5.4: Запуск исправленной программы

```

%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. 5.5: Исправленный текст программы

6 Выводы

В результате выполнения лабораторной работы я приобрёл навыки написания программ с использованием циклов и обработкой аргументов командной строки в NASM.

Список литературы