

Лабораторная работа №9

Дисциплина: Архитектура компьютеров и операционные системы

Чистов Даниил Максимович

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	Реализация подпрограмм в NASM	7
3.2	Отладка программ с помощью GDB	10
3.3	Добавление точек останова	16
3.4	Работа с данными программы в GDB	18
3.5	Обработка аргументов командной строки в GDB	23
4	Задание для самостоятельной работы	25
4.1	Задание 1	25
4.2	Задание 2	27
5	Выводы	35
	Список литературы	36

Список иллюстраций

3.1	Создание каталога и файлов	7
3.2	Код программы	8
3.3	Работа программы	9
3.4	Изменение кода программы	9
3.5	Работа программы	10
3.6	Код программы	11
3.7	Запуск программы в GDB	12
3.8	Программа работает успешно	12
3.9	Метка <code>y_start</code>	13
3.10	Диассимилированный код	13
3.11	Отображение команд	14
3.12	Особое отображение <code>asm</code>	15
3.13	Особое отображение <code>asm</code>	16
3.14	Отображение точек останова	17
3.15	Установка точки останова	17
3.16	Отображение точек останова	19
3.17	Значения регистров	20
3.18	Значения переменных <code>msg1</code> и <code>msg2</code>	20
3.19	Изменения значений <code>msg1</code> и <code>msg2</code>	21
3.20	Различные форматы значений регистра <code>edx</code>	21
3.21	Завершение работы программы	22
3.22	Копирование файла <code>lab8-2</code>	23
3.23	Загрузка в <code>gdb</code>	23
3.24	Просмотр позиций стека	24
4.1	Код программы	26
4.2	Программа работает успешно!	27
4.3	Код программы	28
4.4	Программа работает некорректно!	29
4.5	Подготовка программы	29
4.6	Подготовка программы	30
4.7	Изменение переменной <code>eax</code>	31
4.8	Изменение переменной <code>ebx</code>	31
4.9	Изменение переменной <code>ecx</code>	31
4.10	Ошибка в коде	32
4.11	Исправленный код	33

4.12 Программа работает успешно!	34
--	----

1 Цель работы

Цель работы - Приобретение навыков написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями.

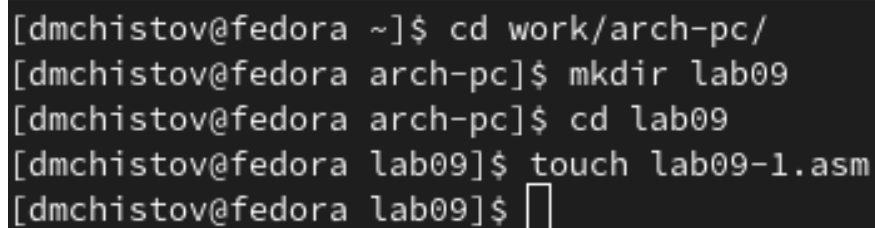
2 Задание

- Выполнение Лабораторной работы
- Задание для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы, а потом создаю файл lab09-1.asm (рис. 3.1).

A terminal window with a dark background and light gray text. It shows a series of commands being executed in a shell. The prompt is [dmchistov@fedora ~]. The first command is cd work/arch-pc/. The second prompt is [dmchistov@fedora arch-pc] and the command is mkdir lab09. The third prompt is [dmchistov@fedora arch-pc] and the command is cd lab09. The fourth prompt is [dmchistov@fedora lab09] and the command is touch lab09-1.asm. The final prompt is [dmchistov@fedora lab09] followed by a cursor.

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

Рис. 3.1: Создание каталога и файлов

Вставляю в созданный файл код (рис. 3.2).

```

lab09-1.asm      [----]  0 L:[  1+ 0   1/ 36]  *(0
%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 ; выход из подпрограмм

```

Рис. 3.2: Код программы

Создаю файл и проверяю его работу (рис. 3.3).

```
[dmchistov@fedora lab09]$ nasm -f elf lab09-1.asm
[dmchistov@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[dmchistov@fedora lab09]$ ./lab09-1
Введите x: 6
2x+7=19
[dmchistov@fedora lab09]$
```

Рис. 3.3: Работа программы

Изменяю код программы, добавив подпрограмму `_subcalcul` (рис. 3.4).

```
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret.

_subcalcul:
mov ebx,3
mul ebx
add eax,-1
mov [res],eax
ret.
```

Рис. 3.4: Изменение кода программы

Создаю файл и проверяю его работу (рис. 3.5).

```
[dmchistov@fedora lab09]$ nasm -f elf lab09-1.asm
[dmchistov@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[dmchistov@fedora lab09]$ ./lab09-1
Введите x: 6
2x+7=19
[dmchistov@fedora lab09]$
```

Рис. 3.5: Работа программы

3.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm, а затем вставляю код из листинга 9.2 (рис. 3.6).

```

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. 3.6: Код программы

Начинаю открывать файл в GDB (рис. 3.7).

```
[dmchistov@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[dmchistov@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[dmchistov@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.2-3.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: Запуск программы в GDB

Проверяю работу программы (рис. 3.8).

```
(gdb) run
Starting program: /home/dmchistov/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]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
Hello, world!
[Inferior 1 (process 3903) exited normally]
(gdb) █
```

Рис. 3.8: Программа работает успешно

Ставлю метку у кода _start (рис. 3.9).

```

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

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

```

Рис. 3.9: Метка у _start

Дисассимилирую код с помощью команды disassemble (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.
(gdb)

```

Рис. 3.10: Диассимилированный код

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel (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.
(gdb) 

```

Рис. 3.11: Отображение команд

Имена регистров в режиме АТТ начинаются с символа %, а имена операндов – с \$, в то время как в синтаксисе Intel используется более привычный формат.

Включаю специальные режимы отображения (рис. 3.12), (рис. 3.13).

```
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
      0x8049020 <_start+32> mov     ecx,0x804a008
      0x8049025 <_start+37> mov     edx,0x7
      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
      0x804903c          add     BYTE PTR [eax],al
      0x804903e          add     BYTE PTR [eax],al
      0x8049040          add     BYTE PTR [eax],al
      0x8049042          add     BYTE PTR [eax],al
      0x8049044          add     BYTE PTR [eax],al
      0x8049046          add     BYTE PTR [eax],al
      0x8049048          add     BYTE PTR [eax],al
      0x804904a          add     BYTE PTR [eax],al

native process 3922 In: _start                                L9    PC: 0x8049000
(gdb)
```

Рис. 3.12: Особое отображение asm

```
[ 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
0x8049020 <_start+32>     mov    ecx,0x804a008
0x8049025 <_start+37>     mov    edx,0x7
0x804902a <_start+42>     int     0x80
0x804902c <_start+44>     mov    eax,0x1

native process 3922 In: _start          L9    PC: 0x8049000
(gdb) layout regs
```

Рис. 3.13: Особое отображение asm

3.3 Добавление точек останова

Проверяю наличие точек останова командой `info breakpoints` (рис. 3.14).


```
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
      0x8049020 <_start+32>   mov     ecx,0x804a008
      0x8049025 <_start+37>   mov     edx,0x7
      0x804902a <_start+42>   int     0x80
      0x804902c <_start+44>   mov     eax,0x1

native process 3922 In: _start
(gdb) layout regs
(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. 3.14: Отображение точек останова

Определяю адрес инструкции `mov ebx,0x0`, а затем устанавливаю точку останова с помощью команды `break` (рис. 3.15).

```
(gdb) layout asm
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) █
```

Рис. 3.15: Установка точки останова

3.4 Работа с данными программы в GDB

С помощью команды `si` выполняю пять инструкций, затем проверяю значения каких регистров меняются (рис. 3.16).

```

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
eflags   0x10202  [ IF RF ]
cs       0x23     35

B+  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
    0x8049025 <_start+37>   mov    $0x7,%edx
    0x804902a <_start+42>   int    $0x80
    0x804902c <_start+44>   mov    $0x1,%eax

```

native process 4132 In: `_start`

```

<https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been
To make this setting permanent, add 'set debuginfod enabled off' to

Breakpoint 1, _start () at lab09-2.asm:9
(gdb) si
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.16: Отображение точек останова

Значения регистров `eax`, `ecx`, `edx` и `ebx` были изменены.

Просматриваю значения регистров с помощью команды `info registers` (рис. 3.17).

```
native process 4132 in: _start
eax          0x4          4
ecx          0x804a000    134520832
edx          0x8          8
ebx          0x1          1
esp          0xffffd1b0   0xffffd1b0
ebp          0x0          0x0
esi          0x0          0
edi          0x0          0
eip          0x804901b    0x804901b <_start+27>
eflags       0x10202     [ IF RF ]
cs           0x23        35
ss           0x2b        43
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 3.17: Значения регистров

Просматриваю значение переменной `msg1` и `msg2` (рис. 3.18).

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

Рис. 3.18: Значения переменных `msg1` и `msg2`

Изменения значений переменных `msg1` и `msg2` с помощью команды `set` (рис. 3.19).

```

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

```

Рис. 3.19: Изменения значений msg1 и msg2

Теперь вывожу значения регистра edx в различных форматах (рис. 3.20).

```

0x804a008 <msg2>:      "sorld!\n\034"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb)

```

Рис. 3.20: Различные форматы значений регистра edx

Команда вывода p/s \$ebx различается в том, что при первой мы преобразуем символ в его текстовое представление, в то время как при второй - числовое значение в строковом виде остается неизменным.

Завершаю работу программы с помощью команды s, после чего выхожу из GDB с помощью команды q (рис. 3.21).

```

eax      0x4      4
eax      0x1      1
ecx      0x804a008 134520840
edx      0x7      7
esp      0x1 ffd1b0 1 ffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
eip      0x8049031 0x8049031 <_start+49>
eflags   0x202    [ IF ]

B+  0x8049000 <_start>      mov    $0x4,%eax
    0x804901b <_start+27>   mov    $0x1,%ebx
    0x8049020 <_start+32>   mov    $0x804a008,%ecx
    0x8049025 <_start+37>   mov    $0x7,%edx
    0x804902a <_start+42>   int     $0x80
    0x804902c <_start+44>   mov    $0x1,%eax
B+> 0x8049031 <_start+49>   mov    $0x0,%ebx
    0x8049036 <_start+54>   int     $0x80      8,%ecx
    0x8049038              add     %al,(%eax)
    0x804903a              add     %al,(%eax)
    0x804903c              add     %al,(%eax)
    3e                    add     %al,(%eax)

native process 4132 In: _start
0x804a008 <msg2>:      "sorld!\n\034"
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) c
Continuing.
sorld!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb)

```

Рис. 3.21: Завершение работы программы

3.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, а затем компилирую его (рис. 3.22).

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

Рис. 3.22: Копирование файла lab8-2

Загружаюсь в gdb перед этим указав аргументы (рис. 3.23).

```
[dmchistov@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[dmchistov@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.2-3.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 lab09-3...
(gdb) █
```

Рис. 3.23: Загрузка в gdb

Просматриваю позиции стека по адресу [esp+4], [esp+8], [esp+12] и т.д. (рис. 3.24).

```
8      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/s *(void**)(esp + 4)
0xffffd31b:    "/home/dmchistov/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd346:    "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd358:    "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd369:    "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd36b:    "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb) 
```

Рис. 3.24: Просмотр позиций стека

Шаг изменения адреса составляет 4, поскольку количество аргументов в командной строке также равно 4.

4 Задание для самостоятельной работы

4.1 Задание 1

Копирую программу из задания 1 лабораторной работы №8, а затем изменяю её, реализовав вычисление значения функции $f(x)$ как подпрограмму (рис. 4.1).

```

global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
<-----> ; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
<-----> ; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
<-----> ; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
<-----> ; промежуточных сумм
mov ebx, 3 ; Храним 3, как один из операндов в ф-ии f(x)
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
call _calcul; преобразуем символ в число
loop next ; переход к обработке следующего аргумента
_end:

mov eax, msg1.
call sprint

mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
;-----
; Подпрограмма вычисления
_calcul:
mul eax
mov ebx, eax ; умножаем x на 3
sub ebx, 1 ; вычитаем 1
add esi, ebx ; добавляем готовый результат к общей сумме
ret.

```

Рис. 4.1: Код программы

Проверяю работу программы (рис. 4.2).

```

[dmchistov@fedora lab09]$ nasm -f elf lab09-4.asm
[dmchistov@fedora lab09]$ ld -m elf_i386 -o lab09-4 lab09-4.o
[dmchistov@fedora lab09]$ ./lab09-4
Функция: f(x)=3x-1
Результат: 0
[dmchistov@fedora lab09]$ ./lab09-4 1 2 3 4
Функция: f(x)=3x-1
Результат: 26

```

Рис. 4.2: Программа работает успешно!

Код программы: %include 'in_out.asm' SECTION .data msg1 db "Функция: f(x)=3x-1", 0x0A msg2 db "Результат:" SECTION .text global _start _start: pop ecx ; Извлекаем из стека в ecx количество ; аргументов (первое значение в стеке) pop edx ; Извлекаем из стека в edx имя программы ; (второе значение в стеке) sub ecx,1 ; Уменьшаем ecx на 1 (количество ; аргументов без названия программы) mov esi, 0 ; Используем esi для хранения ; промежуточных сумм mov ebx, 3 ; Храним 3, как один из операндов в ф-ии f(x) 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 call _calcul; преобразуем символ в число loop next ; переход к обработке следующего аргумента _end:

```
    mov eax, msg1 call sprint
```

```
    mov eax, esi ; записываем сумму в регистр eax call iprintLF ; печать результата
call quit ; завершение программы ;————— ; Подпрограмма
вычисления _calcul: mul eax mov ebx, eax ; умножаем x на 3 sub ebx, 1 ; вычитаем
1 add esi, ebx ; добавляем готовый результат к общей сумме ret
```

4.2 Задание 2

Открываю код программы из листинга 9.3 (рис. 4.3).

```
%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. 4.3: Код программы

Проверяю работу программы, результат должен быть равен 25 (рис. 4.4).

```
[dmchistov@fedora lab09]$ nasm -f elf lab09-5.asm
[dmchistov@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[dmchistov@fedora lab09]$ ./lab09-5
Результат: 10
[dmchistov@fedora lab09]$
```

Рис. 4.4: Программа работает некорректно!

Программа выводит неправильный результат, нужно искать ошибку. Компилирую код для работы с gdb, затем запускаю её (рис. 4.5).

```
[dmchistov@fedora lab09]$ nasm -f elf -g -l lab09-5.lst lab09-5.asm
[dmchistov@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[dmchistov@fedora lab09]$ gdb lab09-5
GNU gdb (GDB) Fedora Linux 13.2-3.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 lab09-5...
```

Рис. 4.5: Подготовка программы

Ставлю точку останова с момента `_start` (рис. 4.6).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+> 0x80490e8 <_start> mov $0x3,%ebx
      0x80490ed <_start+5> mov $0x2,%eax
      0x80490f2 <_start+10> add %eax,%ebx
      0x80490f4 <_start+12> mov $0x4,%ecx
      0x80490f9 <_start+17> mul %ecx
      0x80490fb <_start+19> add $0x5,%ebx
      0x80490fe <_start+22> mov %ebx,%edi
      0x8049100 <_start+24> mov $0x804a000,%eax
      0x8049105 <_start+29> call 0x804900f <sprint>
      0x804910a <_start+34> mov %edi,%eax
      0x804910c <_start+36> call 0x8049086 <iprintf>
      0x8049111 <_start+41> call 0x80490db <quit>
      0x8049116          add %al,(%eax)

native process 5349 In: _start L8 PC: 0x8049
(gdb) layout regs
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-5.asm, line 8.
(gdb) run
Starting program: /home/dmchistov/work/arch-pc/lab09/lab09-5

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

Breakpoint 1, _start () at lab09-5.asm:8
(gdb) █
```

Рис. 4.6: Подготовка программы

Начинаю следить за значениями переменных. На (рис. 4.7), (рис. 4.8) и (рис. 4.9) всё идёт хорошо.

Register group: general		
eax	0x2	2
ecx	0x0	0
edx	0x0	0
ebx	0x3	3
esp	0xffffd1c0	0xffffd1c0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
pin	0v80100af2	0v80100af2 < start+10\

Рис. 4.7: Изменение переменной eax

Register group: general		
eax	0x2	2
ecx	0x0	0
edx	0x0	0
ebx	0x5	5
esp	0xffffd1c0	0xffffd1c0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
pin	0v80100af1	0v80100af1 < start+12\

Рис. 4.8: Изменение переменной ebx

Register group: general		
eax	0x2	2
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffd1c0	0xffffd1c0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
pin	0v80100afa	0v80100afa < start+17\

Рис. 4.9: Изменение переменной ecx

А вот в этом моменте явно происходит ошибка, происходит перемножение значений переменных eax и ecx, а должно ecx и ebx (рис. 4.10).

Register group: general		
eax	0x8	8
ecx	0x4	4
edx	0x0	0
ebx	0x5	5
esp	0xffffd1c0	0xffffd1c0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
win	0x80100fh	0x80100fh / start+10\

Рис. 4.10: Ошибка в коде

Чтобы исправить ошибку, мы добавляем после `add ebx, eax` строку `mov eax, ebx` и заменяем `ebx` на `eax` в инструкциях `add eax, 5` и `mov edi, eax` (рис. 4.11).


```
lab09-5.asm [----] 0 L: [ 1+ 0 1/ 23]
%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 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. 4.11: Исправленный код

Проверяю работу программы (рис. 4.12).

```
(gdb) layout asm
[dmchistov@fedora lab09]$ nasm -f elf lab09-5.asm
[dmchistov@fedora lab09]$ ld -m elf_i386 -o lab09-5 lab09-5.o
[dmchistov@fedora lab09]$ ./lab09-5
Результат: 25
```

Рис. 4.12: Программа работает успешно!

Код 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 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

5 Выводы

Благодаря выполнению работы, я научился приобретению навыков написания программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.

Список литературы

Лабораторная работы №9