

Лабораторная работа номер 9

Понятие подпрограммы. Отладчик GDB.

Соловьев Богдан Михайлович

Содержание

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки; • поиск её местонахождения; • определение причины ошибки; • исправление ошибки.

Можно выделить следующие типы ошибок:

- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; • семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы.

Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

Наиболее часто применяют следующие методы отладки:

- создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);
- использование специальных программ-отладчиков. 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 строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:
- Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);
- Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы

3 Выполнение лабораторной работы

Создаю файл lab09-1.asm (рис. 1).

```
c0af56d..78d3045 master -> master
[bmsolovjev@fedora report]$ cd /home/bmsolovjev/work/arch-pc
[bmsolovjev@fedora arch-pc]$ mkdir ~/work/arch-pc/lab09
[bmsolovjev@fedora arch-pc]$ cd ~/work/arch-pc/lab09
[bmsolovjev@fedora lab09]$ touch lab09-1.asm
[bmsolovjev@fedora lab09]$
```

Figure 1: Создание файла lab09-1.asm

Ввожу в созданный файл код программы, считающей выражение $2x + 7$ с помощью подпрограммы `_calcul`. (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
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 подпрограммы

```

Figure 2: Код программы lab09-1.asm

Проверяю код (рис. 3).

```
[bmsolovjev@fedora lab09]$ nasm -f elf lab09-1.asm
[bmsolovjev@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[bmsolovjev@fedora lab09]$ ./lab09-1
Введите x: 67
2x+7=141
[bmsolovjev@fedora lab09]$
```

Figure 3: Работа программы lab09-1.asm

Изменяю текст программы таким образом, чтобы подпрограмма `_subcalcul` считала $3x - 1$ (рис. 4).

```

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:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx, 3
mul ebx
add eax, 1
mov [res], eax
ret ; выход из подпрограммы

```

Figure 4: Код программы lab09-1.asm с _subcalcul

Проверяю код (рис. 5).

```
[bmsolovjev@fedora lab09]$ nasm -f elf lab09-1.asm
[bmsolovjev@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[bmsolovjev@fedora lab09]$ ./lab09-1
Введите x: 1
2x+7=15
[bmsolovjev@fedora lab09]$
```

Figure 5: Работа программы lab09-1.asm с _subcalcul

Создаю новый файл lab09-2.asm (рис. 6).

```

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
|

```

Figure 6: Код программы lab09-2.asm

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (1) (рис. 7).

```
[bmsolovjev@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[bmsolovjev@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[bmsolovjev@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.1-2.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) run
Starting program: /home/bmsolovjev/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
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-supplied DSO at 0xf7ffc000.
Hello, world!
[Inferior 1 (process 22053) exited normally]
(gdb)
```

Figure 7: Проверка программы в оболочке GDB

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (2) (рис. 8).

```
(gdb) break _start
Breakpoint 1 at 0x4010e0: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/bmsolovjev/work/arch-pc/lab09/lab09-2
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-supplied DSO at 0xf7ffc000.

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

Figure 8: Код программы lab09-2.asm (2)

Сравниваю различия отображения синтаксиса машинных команд в режимах AT&T и Intel. Отличие есть только в порядке отображения адреса регистра и самого регистра(рис. 9).


```
(gdb) disassemble _start
```

```
Dump of assembler code for function _start:
```

```
=> 0x004010e0 <+0>:      mov     $0x4,%eax
    0x004010e5 <+5>:      mov     $0x1,%ebx
    0x004010ea <+10>:     mov     $0x402118,%ecx
    0x004010ef <+15>:     mov     $0x8,%edx
    0x004010f4 <+20>:     int     $0x80
    0x004010f6 <+22>:     mov     $0x4,%eax
    0x004010fb <+27>:     mov     $0x1,%ebx
    0x00401100 <+32>:     mov     $0x402120,%ecx
    0x00401105 <+37>:     mov     $0x7,%edx
    0x0040110a <+42>:     int     $0x80
    0x0040110c <+44>:     mov     $0x1,%eax
    0x00401111 <+49>:     mov     $0x0,%ebx
    0x00401116 <+54>:     int     $0x80
```

```
End of assembler dump.
```

```
(gdb) set disassembly-flavor intel
```

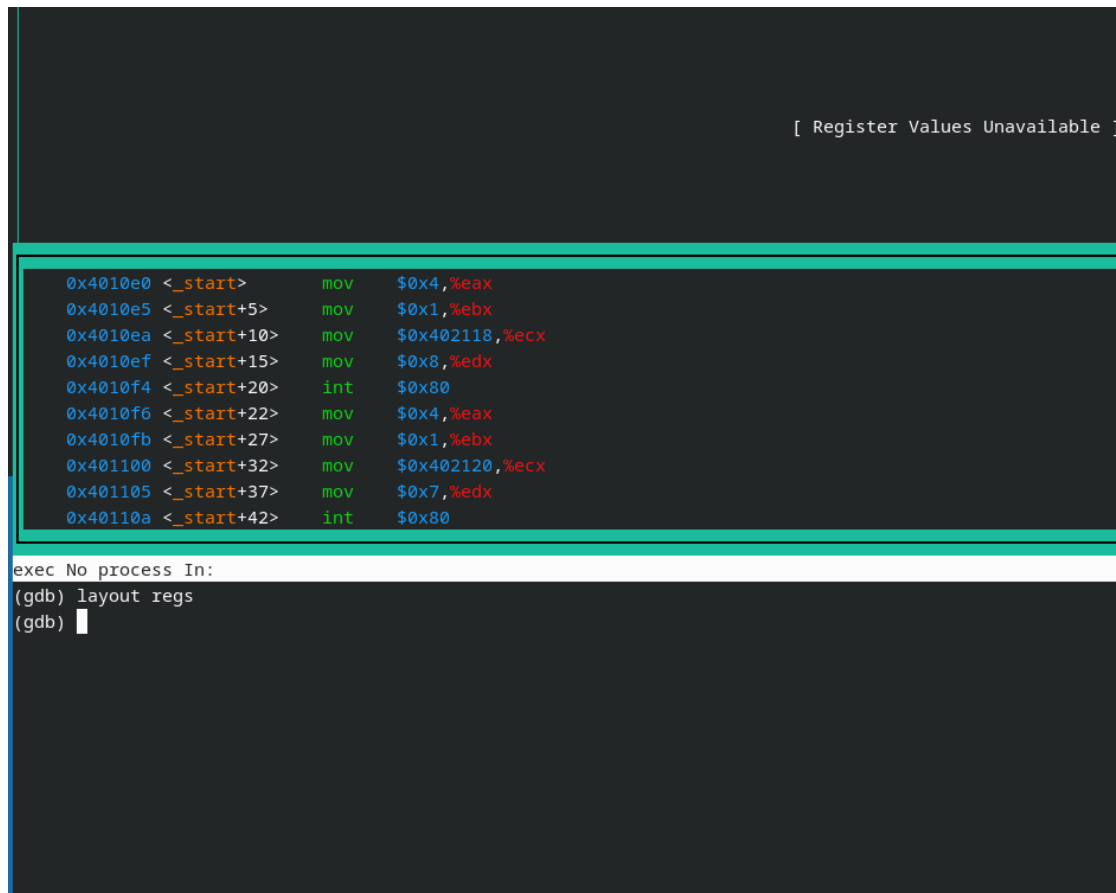
```
(gdb) disassemble _start
```

```
Dump of assembler code for function _start:
```

```
=> 0x004010e0 <+0>:      mov     eax,0x4
    0x004010e5 <+5>:      mov     ebx,0x1
    0x004010ea <+10>:     mov     ecx,0x402118
    0x004010ef <+15>:     mov     edx,0x8
    0x004010f4 <+20>:     int     0x80
    0x004010f6 <+22>:     mov     eax,0x4
    0x004010fb <+27>:     mov     ebx,0x1
    0x00401100 <+32>:     mov     ecx,0x402120
    0x00401105 <+37>:     mov     edx,0x7
    0x0040110a <+42>:     int     0x80
```

Figure 9: Отображение синтаксиса в разных режимах

Включаю режим псевдографики для более удобного анализа программы (1) (рис. 10).



The screenshot shows a debugger window with a dark background. At the top, there is a section labeled "[Register Values Unavailable]". Below this, a list of assembly instructions is displayed, each with its address, a comment, and the instruction itself. The instructions are as follows:

Address	Comment	Instruction
0x4010e0	<_start>	mov \$0x4,%eax
0x4010e5	<_start+5>	mov \$0x1,%ebx
0x4010ea	<_start+10>	mov \$0x402118,%ecx
0x4010ef	<_start+15>	mov \$0x8,%edx
0x4010f4	<_start+20>	int \$0x80
0x4010f6	<_start+22>	mov \$0x4,%eax
0x4010fb	<_start+27>	mov \$0x1,%ebx
0x401100	<_start+32>	mov \$0x402120,%ecx
0x401105	<_start+37>	mov \$0x7,%edx
0x40110a	<_start+42>	int \$0x80

Below the assembly list, there is a command prompt showing the following text:

```
exec No process In:  
(gdb) layout regs  
(gdb) █
```

Figure 10: Режим псевдографики

Включаю режим псевдографики для более удобного анализа программы (2) (рис. 11).

```
[ Register Values Unavailable ]

0x4010e0 <_start>    mov    $0x4,%eax
0x4010e5 <_start+5>    mov    $0x1,%ebx
0x4010ea <_start+10>   mov    $0x402118,%ecx
0x4010ef <_start+15>   mov    $0x8,%edx
0x4010f4 <_start+20>   int    $0x80
0x4010f6 <_start+22>   mov    $0x4,%eax
0x4010fb <_start+27>   mov    $0x1,%ebx
0x401100 <_start+32>   mov    $0x402120,%ecx
0x401105 <_start+37>   mov    $0x7,%edx
0x40110a <_start+42>   int    $0x80

exec No process In:
(gdb) layout reg
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) █
```

Figure 11: Режим псевдографики с регистрами

Устанавливая break point (рис. 12).

```
[ Register Values Unavailable ]

0x4010ea <_start+10>  mov    $0x402118,%ecx
0x4010ef <_start+15>  mov    $0x8,%edx
0x4010f4 <_start+20>  int    $0x80
0x4010f6 <_start+22>  mov    $0x4,%eax
0x4010fb <_start+27>  mov    $0x1,%ebx
0x401100 <_start+32>  mov    $0x402120,%ecx
0x401105 <_start+37>  mov    $0x7,%edx
0x40110a <_start+42>  int    $0x80
0x40110c <_start+44>  mov    $0x1,%eax
b+ 0x401111 <_start+49>  mov    $0x0,%ebx

exec No process In:
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) break *<_start+49>
A syntax error in expression, near `<_start+49>'.
(gdb) break *<0x401111>
A syntax error in expression, near `<0x401111>'.
(gdb) break *0x401111
Breakpoint 1 at 0x401111: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x00401111 lab09-2.asm:20
(gdb) 
```

Figure 12: Установка break point

Просматриваю значение msg1 по имени (рис. 13).

```
[ Register Values Unavailable ]

0x4010ea <_start+10>  mov    $0x402118,%ecx
0x4010ef <_start+15>  mov    $0x8,%edx
0x4010f4 <_start+20>  int     $0x80
0x4010f6 <_start+22>  mov    $0x4,%eax
0x4010fb <_start+27>  mov    $0x1,%ebx
0x401100 <_start+32>  mov    $0x402120,%ecx
0x401105 <_start+37>  mov    $0x7,%edx
0x40110a <_start+42>  int     $0x80
0x40110c <_start+44>  mov    $0x1,%eax
b+ 0x401111 <_start+49>  mov    $0x0,%ebx

exec No process In:
(gdb) break *<0x401111>
A syntax error in expression, near '<0x401111>'.
(gdb) break *0x401111
Breakpoint 1 at 0x401111: file lab09-2.asm, line 20.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y   0x00401111 lab09-2.asm:20
(gdb) info registers
The program has no registers now.
(gdb) x/1sb &msg1
0x402118 <msg1>:      "Hello, "
(gdb) 
```

Figure 13: Просматривание значение по имени

Просматриваю значение msg2 по имени (рис. 14).

```
[ Register Values Unavailable ]

0x4010fb <_start+27>  mov    $0x1,%ebx
0x401100 <_start+32>  mov    $0x402120,%ecx
0x401105 <_start+37>  mov    $0x7,%edx
0x40110a <_start+42>  int    $0x80
0x40110c <_start+44>  mov    $0x1,%eax
b+ 0x401111 <_start+49>  mov    $0x0,%ebx
0x401116 <_start+54>  int    $0x80

exec No process in:
(gdb) break *0x401111
Breakpoint 1 at 0x401111: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x00401111 lab09-2.asm:20
(gdb) info registers
The program has no registers now.
(gdb) x/1sb &msg1
0x402118 <msg1>:      "Hello, "
(gdb) x/1sb 0x402120
0x402120 <msg2>:      "world!\n"<error: Cannot access memory at address 0x402127>
(gdb) █
```

Figure 14: Просматривание значение по имени часть 2

Пытаюсь заменить символ (почему-то выдаёт ошибку)(рис. 15).

```
[ Register Values Unavailable ]

0x4010e0 <_start>      mov     $0x4,%eax
0x4010e5 <_start+5>     mov     $0x1,%ebx
0x4010ea <_start+10>    mov     $0x402118,%ecx
0x4010ef <_start+15>    mov     $0x8,%edx
0x4010f4 <_start+20>    int     $0x80
0x4010f6 <_start+22>    mov     $0x4,%eax
0x4010fb <_start+27>    mov     $0x1,%ebx
0x401100 <_start+32>    mov     $0x402120,%ecx
0x401105 <_start+37>    mov     $0x7,%edx
0x40110a <_start+42>    int     $0x80

exec No process In:
'msg2' has unknown type; cast it to its declared type
(gdb) si
The program is not being run.
(gdb) set {char}&msg2='W'
No symbol "msg2" in current context.
(gdb) set {char}&msg2='W'
Cannot access memory at address 0x402120
(gdb) stepi
The program is not being run.
(gdb) x/1sb &msg2
0x402120 <msg2>:      "world!\n"<error: Cannot access memory at address 0x402127>
(gdb) █
```

Figure 15: Ошибка

Повторяю действия из конспекта, потом загружаю в gdb программу с аргументами командой `gdb -args lab09-3 аргумент1 аргумент 2 'аргумент 3'` (рис. 16).

```

[bmsolovjev@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[bmsolovjev@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[bmsolovjev@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[bmsolovjev@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[bmsolovjev@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.1-2.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)

```

Figure 16: Программы в gdb с аргументами

Узнаю адрес вершины стека, в которого загрузились аргументы (рис. 17).

```

(gdb) b _start
Breakpoint 1 at 0x4011a8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/bmsolovjev/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.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Download failed: Нет маршрута до узла. Continuing without separate debug info for system-supplied DSO

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xffffd080:    0x00000005
(gdb)

```

Figure 17: Адрес вершины стека

Смотрю остальные позиции из стека. Их адреса находятся на одинаковом, потому что программа заранее не знает сколько места займёт каждая переменная, и поэтому места выделено на каждый элемент одинаково (рис. 18).


```

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd080:    0x00000005
(gdb) x/s *(void**)( $esp + 4)
0xffffd25f:    "/home/bmsolovjev/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd28b:    "аргумент1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd29d:    "аргумент"
(gdb) x/s *(void**)( $esp + 16)
0xffffd2ae:    "2"
(gdb) x/s *(void**)( $esp + 20)
0xffffd2b0:    "аргумент 3"
(gdb) x/s *(void**)( $esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)

```

Figure 18: Адреса в стеке

4 Самостоятельная работа

Я изменил код программы из прошлой лабораторной таким образом, чтобы там была подпрограмма (рис. 19).

```

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
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 в число
call _calcul
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg0
call printf
mov eax, msg ; вывод сообщения "Результат: "
call printf
mov eax, esi ; записываем сумму в регистр `eax`
call printf ; печать результата
call quit ; завершение программы
_calcul:
mov ebx,10
mul ebx
add eax,-5
add esi, eax
ret ; выход из подпрограммы

```

Figure 19: Код программы с подпрограммой

Проверяю правильность кода (рис. 20).

```
ошибка сегментирования (образ памяти сброшен на диск)
[bmsolovjev@fedora lab09]$ nasm -f elf zadanie1.asm
[bmsolovjev@fedora lab09]$ ld -m elf_i386 -o zadanie1 zadanie1.o
[bmsolovjev@fedora lab09]$ ./zadanie1 6 7
f(x) = 10x - 5
Результат: 120
[bmsolovjev@fedora lab09]$
```

Figure 20: Проверка кода

Чтобы код работал правильно, мне пришлось заменить `add ebx, eax` на `add eax,ebx`, а `mov edi, ebx` на `mov edi, eax` (рис. 21).

```
%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 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
```

Figure 21: Измененный код

Проверка правильности кода (рис. 22).

```

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
|

```

Figure 22: Код работает корректно

5 Выводы

Выполнив данную лабораторную работу, я приобрёл навыки написания программ с использованием подпрограмм. Познакомился с методами отладки при помощи GDB и его основными возможностями.

Список литературы