

Отчёт по лабораторной работе №9

Дисциплина: Архитектура Компьютера

Дарина Андреевна Куокконен

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	10
5	Выводы	30
	Список литературы	31

Список иллюстраций

4.1	Работа с директориями и копирование, создание файла	10
4.2	Открытие и редактирование файла	11
4.3	Создание и запуск исполняемого файла	11
4.4	Редактирование файла	12
4.5	Создание и запуск исполняемого файла	12
4.6	Редактирование файла	13
4.7	Создание исполняемого файла, отладчик gdb	14
4.8	Отладчик gdb	14
4.9	Установка метки	14
4.10	Работа метки	15
4.11	Дисассимилированный код программы с синтаксисом intel	15
4.12	Режим псевдографики	16
4.13	Режим псевдографики	16
4.14	Проверка точек останова	17
4.15	Редактирование файла	17
4.16	Создание и запуск исполняемого файла	17
4.17	Изменение значений регистров и переменных	18
4.18	Изменение значений регистров и переменных	19
4.19	Содержимое переменной, изменение в ней символов	20
4.20	Содержимое переменной, изменение в ней символов	20
4.21	Значения регистра edx	21
4.22	Изменение значений регистра	21
4.23	Копирование файла, создание исполняемого файла	22
4.24	Загрузка файла в отладчик	22
4.25	Установка точки останова, запуск программы	23
4.26	Просмотр позиций стека	23
4.27	Копирование файла	23
4.28	Редактирование файла	24
4.29	Создание и запуск исполняемого файла	24
4.30	Редактирование файла	25
4.31	Создание и запуск исполняемого файла	25
4.32	Изменение значений регистров в gdb	26
4.33	Редактирование файла	26
4.34	Создание и запуск исполняемого файла	27

1 Цель работы

Цель данной лабораторной работы - это приобретение практического опыта в написании программ с использованием подпрограмм, а также знакомство с методами отладки при помощи *gdb* и его основными возможностями.

2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программ при помощи gdb.
3. Выполнение заданий для самостоятельной работы

3 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 строки, 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. 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. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя. GDB может выполнять следующие действия:

- начать выполнение программы, задав всё, что может повлиять на её поведение;
- остановить программу при указанных условиях;
- исследовать, что случилось, когда программа остановилась;
- изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

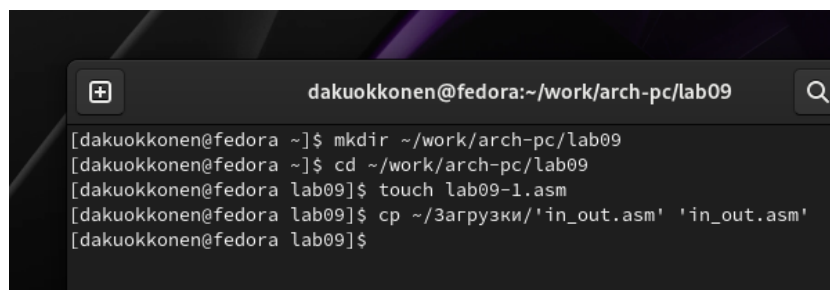
После запуска `gdb` выводит текстовое сообщение — так называемое «nice GDB logo». В следующей строке появляется приглашение (`gdb`) для ввода команд. Далее приведён список некоторых команд GDB. Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB. Если точки останова были заданы, то отладчик останавливается на соответствующей команде и выдаёт номер точки останова, адрес и дополнительную информацию — текущую строку, имя процедуры, и др. Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (`breakpoints`), точки просмотра (`watchpoints`) и точки отлова (`catchpoints`) сохраняются. Для выхода из отладчика используется команда `quit` (или сокращённо `q`). Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`. Установить точку останова можно командой `break` (кратко `b`). Типич-

ный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка». Для продолжения остановленной программы используется команда `continue (c)` (gdb). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число \star , которое указывает отладчику проигнорировать $\star - 1$ точку останова (выполнение остановится на \star -й точке). Команда `stepi` (кратко `sI`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию. Как уже упоминалось, отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Посмотреть содержимое регистров можно с помощью команды `info registers` (или `i r`). Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы. Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `esp` адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `esp`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

4 Выполнение лабораторной работы

4.1) Реализация подпрограмм в NASM.

С помощью утилиты `mkdir` создаю директорию `lab09` для выполнения соответствующей лабораторной работы. Перехожу в созданный каталог с помощью утилиты `cd`. С помощью `touch` создаю файл `lab09-1.asm`. Копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, так как он будет использоваться в дальнейшем. (рис. 4.1).



```
dakuokkonen@fedora:~/work/arch-pc/lab09
[dakuokkonen@fedora ~]$ mkdir ~/work/arch-pc/lab09
[dakuokkonen@fedora ~]$ cd ~/work/arch-pc/lab09
[dakuokkonen@fedora lab09]$ touch lab09-1.asm
[dakuokkonen@fedora lab09]$ cp ~/Загрузки/in_out.asm 'in_out.asm'
[dakuokkonen@fedora lab09]$
```

Рис. 4.1: Работа с директориями и копирование, создание файла

Открываю созданный файл `lab09-1.asm`, вставляю в него следующую программу: (рис. 4.2).

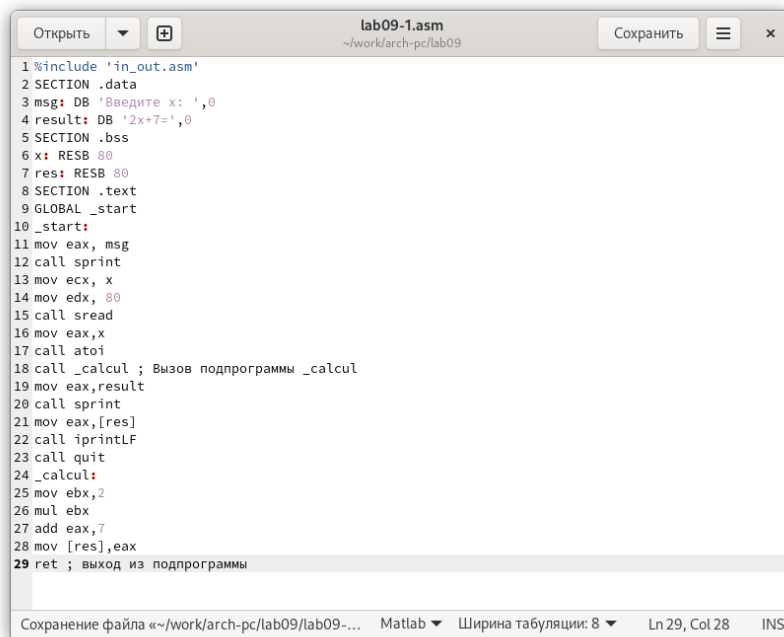


Рис. 4.2: Открытие и редактирование файла

Создаю исполняемый файл и запускаю его. (рис. 4.3).

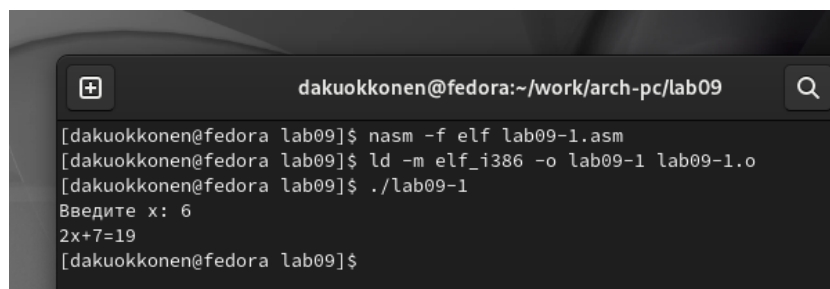


Рис. 4.3: Создание и запуск исполняемого файла

Добавляю подпрограмму *subcalcul_*, чтобы программа вычисляла значение $f(g(x))$. (рис. 4.4).

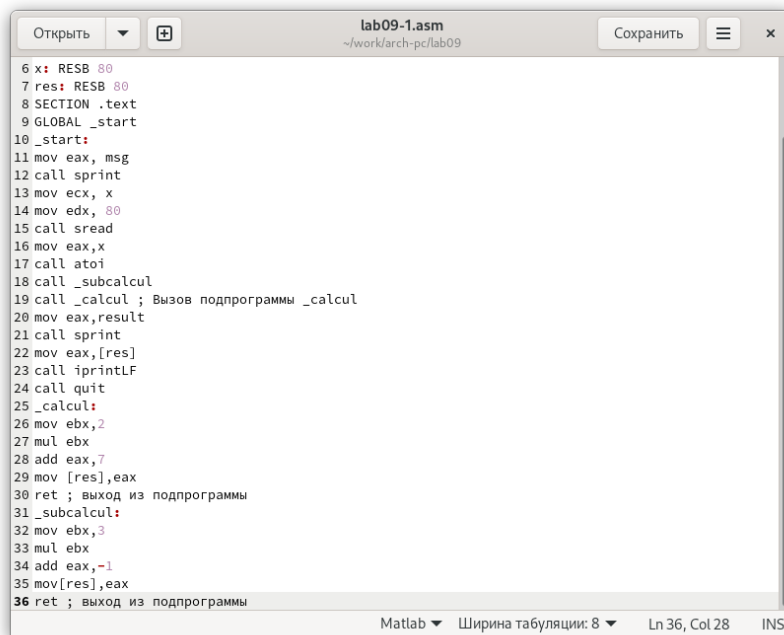


Рис. 4.4: Редактирование файла

Создаю исполняемый файл и убеждаюсь в правильности его работы. (рис. 4.5).

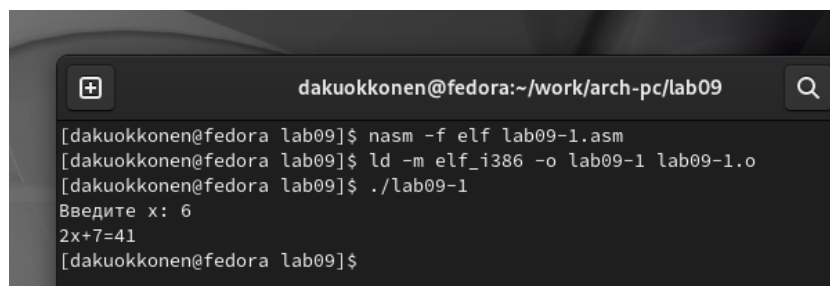
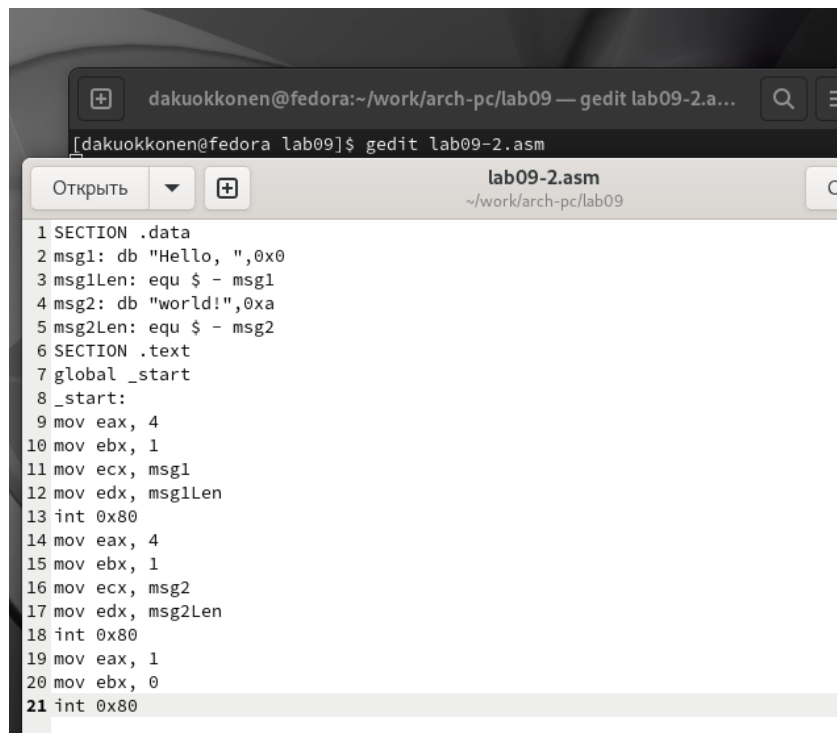


Рис. 4.5: Создание и запуск исполняемого файла

4.2) Отладка программ при помощи gdb.

Создаю файл lab09-2.asm и вношу в него следующий текст программы: (рис. 4.6).



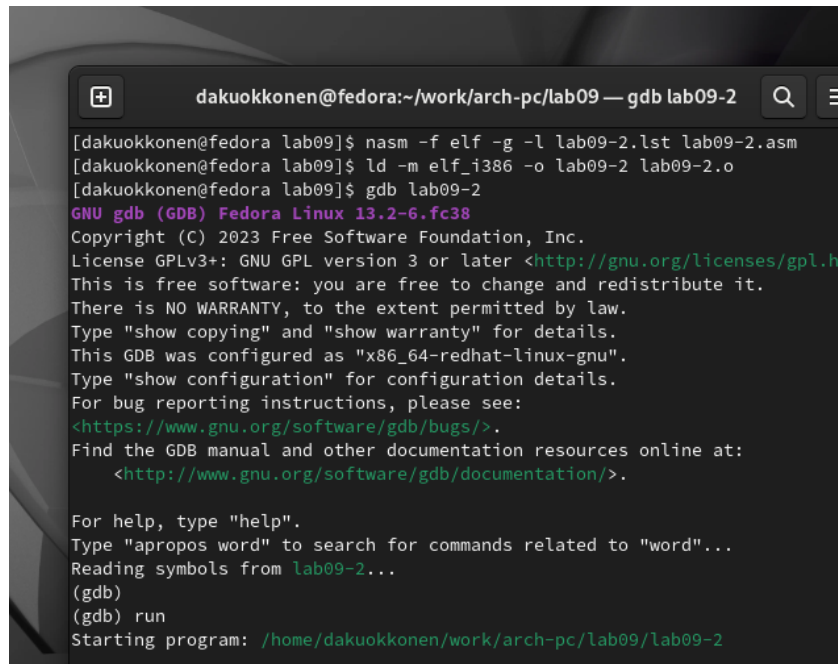
```
[dakuokkonen@fedora lab09]$ gedit lab09-2.asm

Открыть ▼ + lab09-2.asm
~/work/arch-pc/lab09

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: Редактирование файла

Создаю исполняемый файл и загружаю его в отладчик *gdb*, запускаю программу с помощью команды *run*. (рис. 4.7).

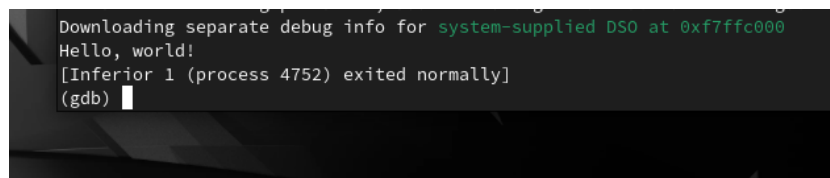


```
dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2
[dakuokkonen@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[dakuokkonen@fedora lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
[dakuokkonen@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.2-6.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.h
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)
(gdb) run
Starting program: /home/dakuokkonen/work/arch-pc/lab09/lab09-2
```

Рис. 4.7: Создание исполняемого файла, отладчик gdb

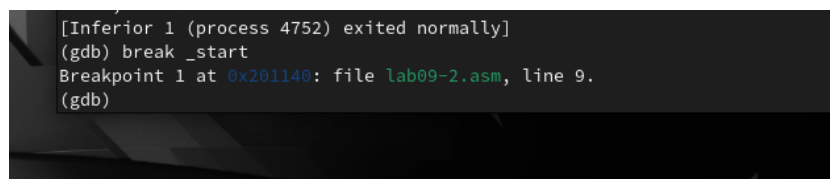
Убеждаюсь в правильности работы программы. (рис. 4.8).



```
Downloading separate debug info for system-supplied DS0 at 0xf7ffc000
Hello, world!
[Inferior 1 (process 4752) exited normally]
(gdb)
```

Рис. 4.8: Отладчик gdb

Устанавливаю метку _start. (рис. 4.9).



```
[Inferior 1 (process 4752) exited normally]
(gdb) break _start
Breakpoint 1 at 0x201140: file lab09-2.asm, line 9.
(gdb)
```

Рис. 4.9: Установка метки

Запускаю программу, видим работу метки. (рис. 4.10).

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

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

Рис. 4.10: Работа метки

Смотрю дисассимилированный код программы сначала обычный, потом с синтаксисом *intel*. (рис. 4.11).

```
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
0x00201140 <+0>:      mov     eax,0x4
0x00201145 <+5>:      mov     ebx,0x1
0x0020114a <+10>:     mov     ecx,0x203188
0x0020114f <+15>:     mov     edx,0x8
0x00201154 <+20>:     int     0x80
0x00201156 <+22>:     mov     eax,0x4
0x0020115b <+27>:     mov     ebx,0x1
0x00201160 <+32>:     mov     ecx,0x203190
0x00201165 <+37>:     mov     edx,0x7
0x0020116a <+42>:     int     0x80
0x0020116c <+44>:     mov     eax,0x1
0x00201171 <+49>:     mov     ebx,0x0
0x00201176 <+54>:     int     0x80
End of assembler dump.
(gdb)
```

Рис. 4.11: Дисассимилированный код программы с синтаксисом *intel*

Различия отображения синтаксиса можно наблюдать в правой части окна. Затем я включаю режим псевдографики (рис. 4.12).

The screenshot shows a GDB window titled 'dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2'. The main display area shows assembly code in pseudo-graphic mode, with instructions and their addresses. The instructions are:
0x201140 <_start> mov eax,0x4
0x201145 <_start+5> mov ebx,0x1
0x20114a <_start+10> mov ecx,0x203188
0x20114f <_start+15> mov edx,0x8
0x201154 <_start+20> int 0x80
0x201156 <_start+22> mov eax,0x4
0x20115b <_start+27> mov ebx,0x1
0x201160 <_start+32> mov ecx,0x203190
0x201165 <_start+37> mov edx,0x7
0x20116a <_start+42> int 0x80
0x20116c <_start+44> mov eax,0x1
0x201171 <_start+49> mov ebx,0x0
0x201176 <_start+54> int 0x80
The status bar at the bottom indicates 'native process 4776 In: _start', 'L9', and 'PC: 0x201140'.

Рис. 4.12: Режим псевдографики

(рис. 4.13).

The screenshot shows a GDB window titled 'dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2'. The main display area shows a message '[Register Values Unavailable]' in the center. Below the message, the assembly code is displayed in pseudo-graphic mode, showing instructions from 0x201140 to 0x201156. The instructions are:
0x201140 <_start> mov eax,0x4
0x201145 <_start+5> mov ebx,0x1
0x20114a <_start+10> mov ecx,0x203188
0x20114f <_start+15> mov edx,0x8
0x201154 <_start+20> int 0x80
0x201156 <_start+22> mov eax,0x4
The status bar at the bottom indicates 'native process 4776 In: _start', 'L9', and 'PC: 0x201140'. Below the status bar, the GDB prompt shows '(gdb) layout regs' and '(gdb) '.

Рис. 4.13: Режим псевдографики

Проверяю точки останова. (рис. 4.14).


```
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x00201140 lab09-2.asm:9
breakpoint already hit 1 time
(gdb)
```

Рис. 4.14: Проверка точек останова

Устанавливаю точку останова в последней инструкции. (рис. 4.15).

```
0x20116a <_start+42>    int     0x80
0x20116c <_start+44>    mov     eax,0x1
b+ 0x201171 <_start+49>    mov     ebx,0x0
0x201176 <_start+54>    int     0x80

exec No process in:
(gdb) layout regs
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x00201140 lab09-2.asm:9
(gdb) b *0x201171
Breakpoint 2 at 0x201171: file lab09-2.asm, line 20.
(gdb)
```

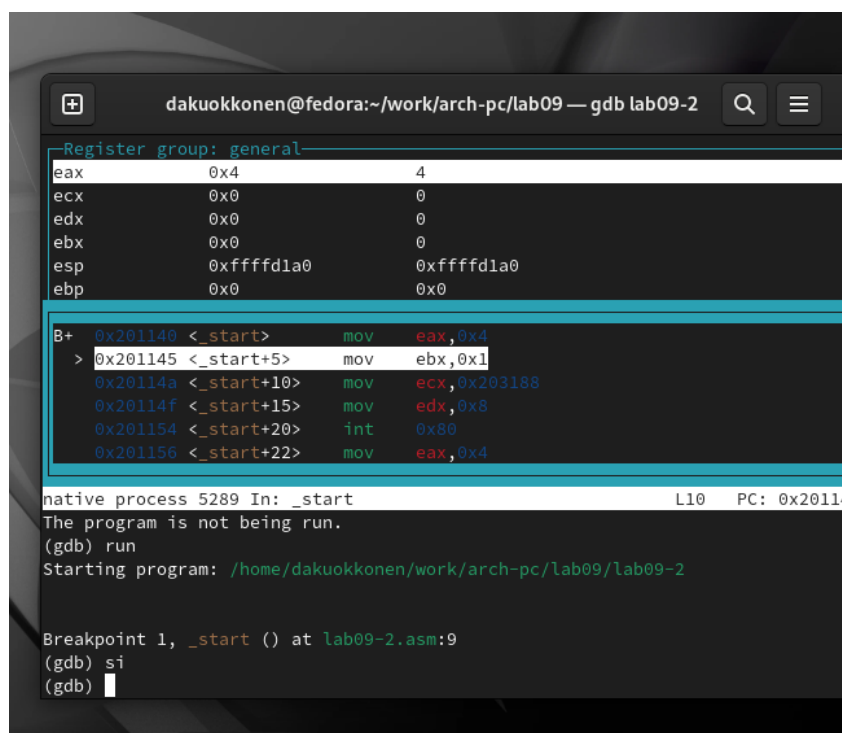
Рис. 4.15: Редактирование файла

Опять же, смотрю информацию обо всех установленных точках останова. (рис. 4.16).

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x00201140 lab09-2.asm:9
2        breakpoint     keep y   0x00201171 lab09-2.asm:20
(gdb)
```

Рис. 4.16: Создание и запуск исполняемого файла

Вручную изменяю значений регистров и переменных с помощью инструкции si. (рис. 4.17).



The screenshot shows a GDB terminal window with the title bar "dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2". The window is divided into several sections. At the top, under "Register group: general", a table lists the values of general-purpose registers: eax (0x4), ecx (0x0), edx (0x0), ebx (0x0), esp (0xffffd1a0), and ebp (0x0). Below this, a list of assembly instructions is displayed, with the instruction at address 0x201145, "mov ebx, 0x1", highlighted. The instruction list includes: "B+ 0x201140 <_start> mov eax, 0x4", "> 0x201145 <_start+5> mov ebx, 0x1", "0x20114a <_start+10> mov ecx, 0x203188", "0x20114f <_start+15> mov edx, 0x8", "0x201154 <_start+20> int 0x80", and "0x201156 <_start+22> mov eax, 0x4". The bottom section shows the GDB prompt with the command "(gdb) run" and the output "Starting program: /home/dakuokkonen/work/arch-pc/lab09/lab09-2". It also shows a breakpoint set at "_start () at lab09-2.asm:9" and the command "(gdb) si" being entered.

```
dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0

B+ 0x201140 <_start>    mov     eax, 0x4
> 0x201145 <_start+5>  mov     ebx, 0x1
0x20114a <_start+10>   mov     ecx, 0x203188
0x20114f <_start+15>   mov     edx, 0x8
0x201154 <_start+20>   int     0x80
0x201156 <_start+22>   mov     eax, 0x4

native process 5289 In: _start L10 PC: 0x2011...
The program is not being run.
(gdb) run
Starting program: /home/dakuokkonen/work/arch-pc/lab09/lab09-2

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

Рис. 4.17: Изменение значений регистров и переменных

Выполняю 5 инструкций si, и последовательно замечанию изменение значений регистров на экране соответственно. (рис. 4.18).

```
dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2

Register group: general
eax      0x8      8
ecx      0x203188  2109832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0

0x20114a <_start+10> mov ecx,0x203188
0x20114f <_start+15> mov edx,0x8
0x201154 <_start+20> int 0x80
> 0x201156 <_start+22> mov eax,0x4
0x20115b <_start+27> mov ebx,0x1
0x201160 <_start+32> mov ecx,0x203190

native process 5289 In: _start L14 PC: 0x201156

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

Рис. 4.18: Изменение значений регистров и переменных

Далее, я просматриваю содержимое переменной `msg1` и изменяю в ней символ с помощью команды `{char}`. (рис. 4.19).

```
dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2

Register group: general
eax      0x8      8
ecx      0x203188 2109832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0

0x20114a <_start+10> mov ecx,0x203188
0x20114f <_start+15> mov edx,0x8
0x201154 <_start+20> int 0x80
> 0x201156 <_start+22> mov eax,0x4
0x20115b <_start+27> mov ebx,0x1
0x201160 <_start+32> mov ecx,0x203190

native process 5289 In: _start L14 PC: 0x201156
(gdb) x/1sb &msg1
0x203188 <msg1>: "Hello, "
(gdb) x/1sb 0x203190
0x203190 <msg2>: "world!\n"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x203188 <msg1>: "hello, "
(gdb)
```

Рис. 4.19: Содержимое переменной, изменение в ней символов

Аналогичные действия проделываю с переменной msg2. (рис. 4.20).

```
dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2

Register group: general
eax      0x8      8
ecx      0x203188 2109832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0

0x20114a <_start+10> mov ecx,0x203188
0x20114f <_start+15> mov edx,0x8
0x201154 <_start+20> int 0x80
> 0x201156 <_start+22> mov eax,0x4
0x20115b <_start+27> mov ebx,0x1
0x201160 <_start+32> mov ecx,0x203190

native process 5289 In: _start L14 PC: 0x201156
0x203190 <msg2>: "world!\n"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x203188 <msg1>: "hello, "
(gdb) set {char}&msg2='b'
(gdb) x/1sb &msg2
0x203190 <msg2>: "borld!\n"
(gdb)
```

Рис. 4.20: Содержимое переменной, изменение в ней символов

Ввожу в различных форматах значение регистра *edx*. (рис. 4.21).

```
dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x8      8
ecx      0x203188 2109832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0

0x20114a <_start+10> mov ecx,0x203188
0x20114f <_start+15> mov edx,0x8
0x201154 <_start+20> int 0x80
> 0x201156 <_start+22> mov eax,0x4
0x20115b <_start+27> mov ebx,0x1
0x201160 <_start+32> mov ecx,0x203190

native process 5289 In: _start L14 PC: 0x201156
0x203190 <msg2>: "borld!\n"
(gdb) p/s $edx
$1 = 8
(gdb) p/t $edx
$2 = 1000
(gdb) p/x $edx
$3 = 0x8
(gdb)
```

Рис. 4.21: Значения регистра *edx*

Изменяю значение регистра *ebx* с помощью команды *set*. (рис. 4.22).

```
dakuokkonen@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x8      8
ecx      0x203188 2109832
edx      0x8      8
ebx      0x2      2
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0

0x20114a <_start+10> mov ecx,0x203188
0x20114f <_start+15> mov edx,0x8
0x201154 <_start+20> int 0x80
> 0x201156 <_start+22> mov eax,0x4
0x20115b <_start+27> mov ebx,0x1
0x201160 <_start+32> mov ecx,0x203190

native process 5289 In: _start L14 PC: 0x201156
$3 = 0x8
(gdb) set $ebx = '2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 4.22: Изменение значений регистра

Разница в выводе команд объясняется в значении: при бескавычном значении

2, мы её и получаем в итоге, а в другом случае переменная воспринимается иначе, и на выходе мы видим значение 50.

Завершаю выполнение программы с помощью *continue* и выхожу из *gdb* с помощью *quit*.

Копирую файл `lab8-2.asm`, полученный во время выполнения лабораторной работы №8, содержащий программу для вывода аргументов командной строки. Затем создаю исполняемый файл. (рис. 4.23).

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

Рис. 4.23: Копирование файла, создание исполняемого файла

Загружаю исполняемый файл в отладчик, указав нужные аргументы. (рис. 4.24).

```
[dakuokkonen@fedora lab09]$ gdb --args lab09-3 argument1 argument 2 'argument 3'
GNU gdb (GDB) Fedora Linux 13.2-6.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. 4.24: Загрузка файла в отладчик

Устанавливаю точку останова перед первой инструкцией и запускаю программу. (рис. 4.25).

```
(gdb) b _start
Breakpoint 1 at 0x201228: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/dakuokkonen/work/arch-pc/lab09/lab09-3 argument1 argument 2 argument\ 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)
```

Рис. 4.25: Установка точки останова, запуск программы

Далее просматриваю позиции стека. (рис. 4.26).

```
(gdb) x/x $esp
0xffffd170: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd327: "/home/dakuokkonen/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd354: "argument1"
(gdb) x/s *(void**)(esp + 12)
0xffffd35e: "argument"
(gdb) x/s *(void**)(esp + 16)
0xffffd367: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd369: "argument 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.26: Просмотр позиций стека

Шаг изменения равен 4, т.к. каждый следующий адрес на стеке находится на расстоянии в 4 байта от предыдущего.

4.3) Выполнение заданий для самостоятельной работы.

Копирую файл задания для самостоятельной работы. (рис. 4.27).

```
[dakuokkonen@fedora lab09]$ cp ~/work/arch-pc/lab08/lab8-4.asm ~/work/arch-pc/lab09/lab09-4.asm
[dakuokkonen@fedora lab09]$ gedit lab09-4.asm
```

Рис. 4.27: Копирование файла

Реализую вычисление значения функции через подпрограмму. (рис. 4.28).

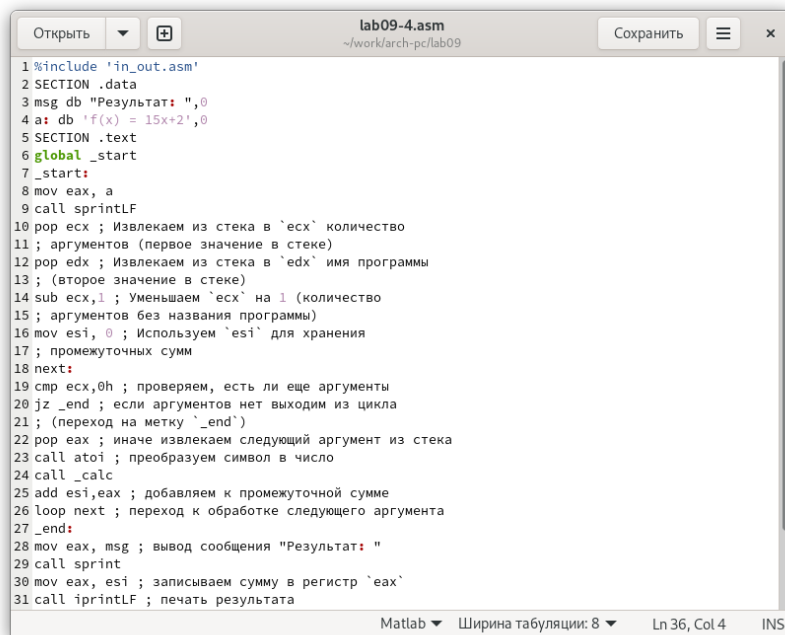


Рис. 4.28: Редактирование файла

Создаю исполняемый файл и убеждаюсь в правильности работы программы.
(рис. 4.29).

```

[dakuokkonen@fedora lab09]$ nasm -f elf lab09-4.asm
[dakuokkonen@fedora lab09]$ ld -m elf_i386 lab09-4.o -o lab09-4
[dakuokkonen@fedora lab09]$ ./lab09-4 1
f(x) = 15x+2
Результат: 17
[dakuokkonen@fedora lab09]$ ./lab09-4 1 2 3 4
f(x) = 15x+2
Результат: 158
[dakuokkonen@fedora lab09]$

```

Рис. 4.29: Создание и запуск исполняемого файла

Создаю файл lab09-4-2.asm и вношу в него программу из последнего листинга.
(рис. 4.30).

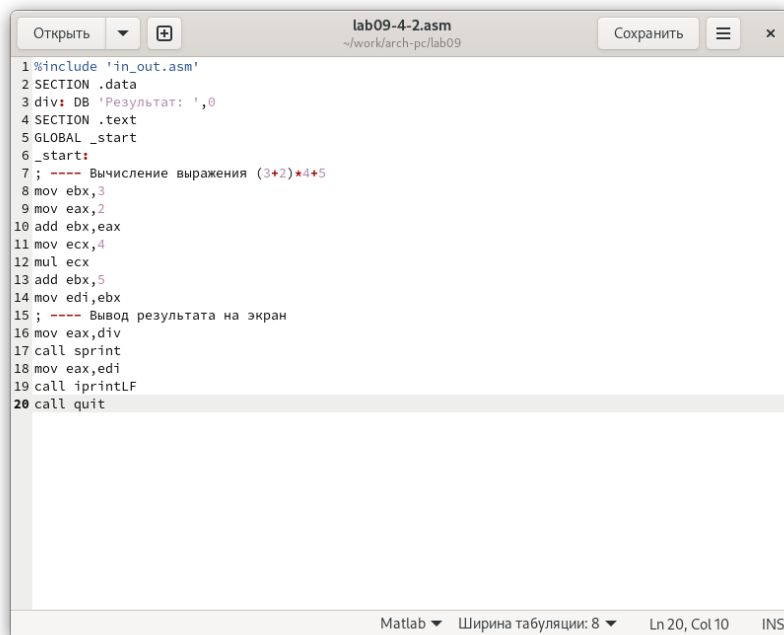


Рис. 4.30: Редактирование файла

При запуске программа дает неверный результат, мы видим результат 10, но он должен быть 25. (рис. 4.31).

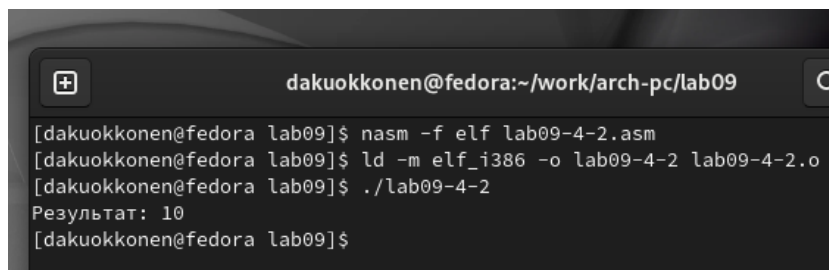


Рис. 4.31: Создание и запуск исполняемого файла

Анализирую изменения значений регистров, чтобы выяснить, в чем заключается ошибка. (рис. 4.32).

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0

0x201234 <_start+12>  mov     ecx,0x4
0x201239 <_start+17>  mul     ecx
0x20123b <_start+19>  add     ebx,0x5
> 0x20123e <_start+22> mov     edi,ebx
0x201240 <_start+24>  mov     eax,0x203268
0x201245 <_start+29>  call    0x20114f <sprint>

native process 3958 In: _start      L??  PC: 0x2012
0x00201234 in _start ()
(gdb) si
```

Рис. 4.32: Изменение значений регистров в gdb

Благодаря этому мне удалось вычислить ошибку и исправить её в тексте программы. (рис. 4.33).

```
Открыть  lab09-4-2.asm  Сохранить
~/work/arch-pc/lab09

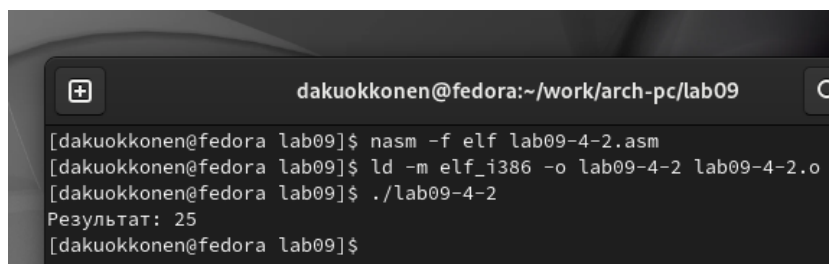
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov eax,ebx
12 mov ecx,4
13 mul ecx
14 add eax,5
15 mov edi,eax
16 ; ---- Вывод результата на экран
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 call quit

Сохранение файла «~/work/arch-pc/lab09/lab09-...»  Matlab  Ширина табуляции: 8  Ln 20, Col 7  INS
```

Рис. 4.33: Редактирование файла

Создаю исполняемый файл, и, выполнив устную проверку, убеждаемся в пра-

вильности работы программы. (рис. 4.34).



```
dakuokkonen@fedora:~/work/arch-pc/lab09
[dakuokkonen@fedora lab09]$ nasm -f elf lab09-4-2.asm
[dakuokkonen@fedora lab09]$ ld -m elf_i386 -o lab09-4-2 lab09-4-2.o
[dakuokkonen@fedora lab09]$ ./lab09-4-2
Результат: 25
[dakuokkonen@fedora lab09]$
```

Рис. 4.34: Создание и запуск исполняемого файла

Листинг 4.1 - Преобразованная программа из лабораторной работы №8.

```
%include 'in_out.asm'

SECTION .data
msg db "Результат: ",0
a: db 'f(x) = 15x+2',0

SECTION .text
global _start
_start:
mov eax, a
call sprintLF
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 _calc
add esi, eax ; добавляем к промежуточной сумме
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
_calc:
imul eax, 15
add eax, 2
ret

```

Листинг 4.2 - Исправленная программа для вычисления значения выражения.

```

#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

При выполнении данной лабораторной работы, я приобрела практический опыт в написании программ с использованием подпрограмм, а также ознакомилась с методами отладки при помощи *gdb* и его основными возможностями.

Список литературы

Архитектура компьютера и ЭВМ