

Отчёт по лабораторной работе № 9

Дисциплина: архитектура компьютера

Кайнова Алина Андреевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	9
4.1	Реализация подпрограмм в NASM	9
4.2	Отладка программ с помощью GDB	12
4.3	Добавление точек останова	17
4.4	Работа с данными программы в GDB	18
4.5	Обработка аргументов командной строки в GDB	22
4.6	Выполнение заданий для самостоятельной работы	23
5	Выводы	30
	Список литературы	31

Список иллюстраций

4.1	Создание каталога и файла	9
4.2	Ввод текста программы	10
4.3	Создание и запуск исполняемого файла	10
4.4	Изменение текста программы	11
4.5	Создание и запуск исполняемого файла	12
4.6	Ввод текста программы	13
4.7	Получение исполняемого файла	14
4.8	Загрузка исполняемого файла в отладчик	14
4.9	Проверка работы файла	14
4.10	Установка брейкпоинта и запуск программы	15
4.11	Просмотр кода программы	16
4.12	Режим псевдографики	17
4.13	Установление точек останова и просмотр информации о них	18
4.14	До использования команды stepi	19
4.15	После использования команды stepi	19
4.16	Просмотр значений переменных	20
4.17	Изменение первого символа переменных	20
4.18	Вывод значения регистра в разных форматах	21
4.19	Изменение значения регистра	21
4.20	Завершение работы GDB	22
4.21	Копирование и создание файла	22
4.22	Загрузка файла с аргументами в отладчик	22
4.23	Установление точки останова и запуск программы	23
4.24	Просмотр значений, введённых в стек	23
4.25	Написание кода программы	24
4.26	Запуск программы и проверка её вывода	25
4.27	Ввод текста программы	25
4.28	Создание и запуск исполняемого файла	26
4.29	Поиск ошибки в GDB	27
4.30	Ошибка в программе	27
4.31	Правка текста программы	28
4.32	Создание и проверка файла	29

1 Цель работы

Приобрест навыки написания программ с использованием подпрограмм и познакомиться с методами отладки при помощи GDB и с его основными возможностями.

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. 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. GDB (GNU Debugger — отладчик проекта GNU) 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. Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB. 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

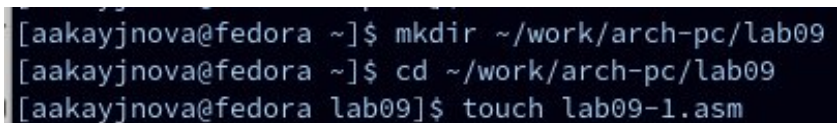
номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`. Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка». Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`). Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`. Обратно точка останова активируется командой `enable`. Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`. Для продолжения остановленной программы используется команда `continue` (`c`). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N – 1` точку останова (выполнение остановится на `N`-й точке). Команда `stepi` (кратко `sl`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию. 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 eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

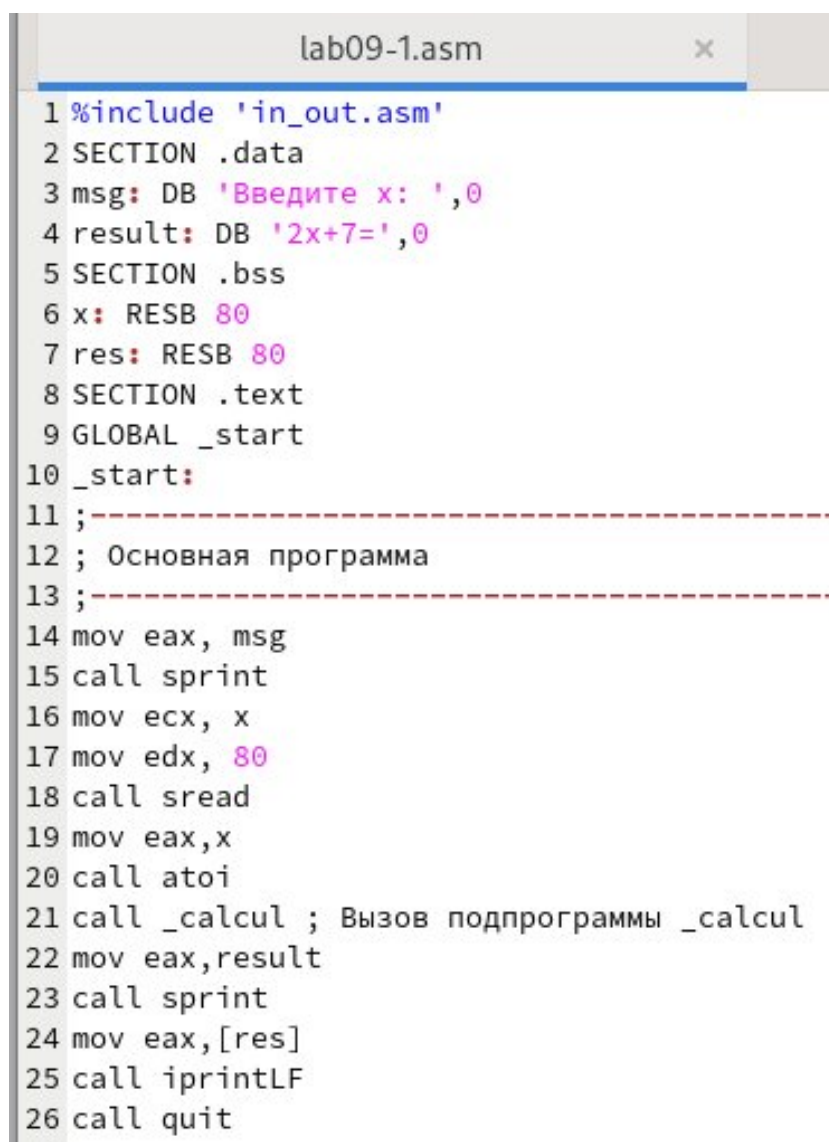
Создаю каталог для выполнения данной лабораторной работы и в нём файл lab09-1.asm



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

Рис. 4.1: Создание каталога и файла

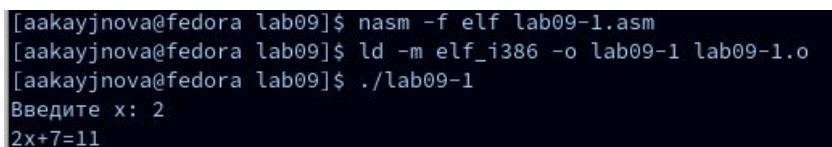
Ввожу в этот файл текст программы с использованием подпрограммы из листинга 9.1



```
lab09-1.asm
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 res: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call iprintLF
26 call quit
--
```

Рис. 4.2: Ввод текста программы

Создаю исполняемый файл и проверяю его работу

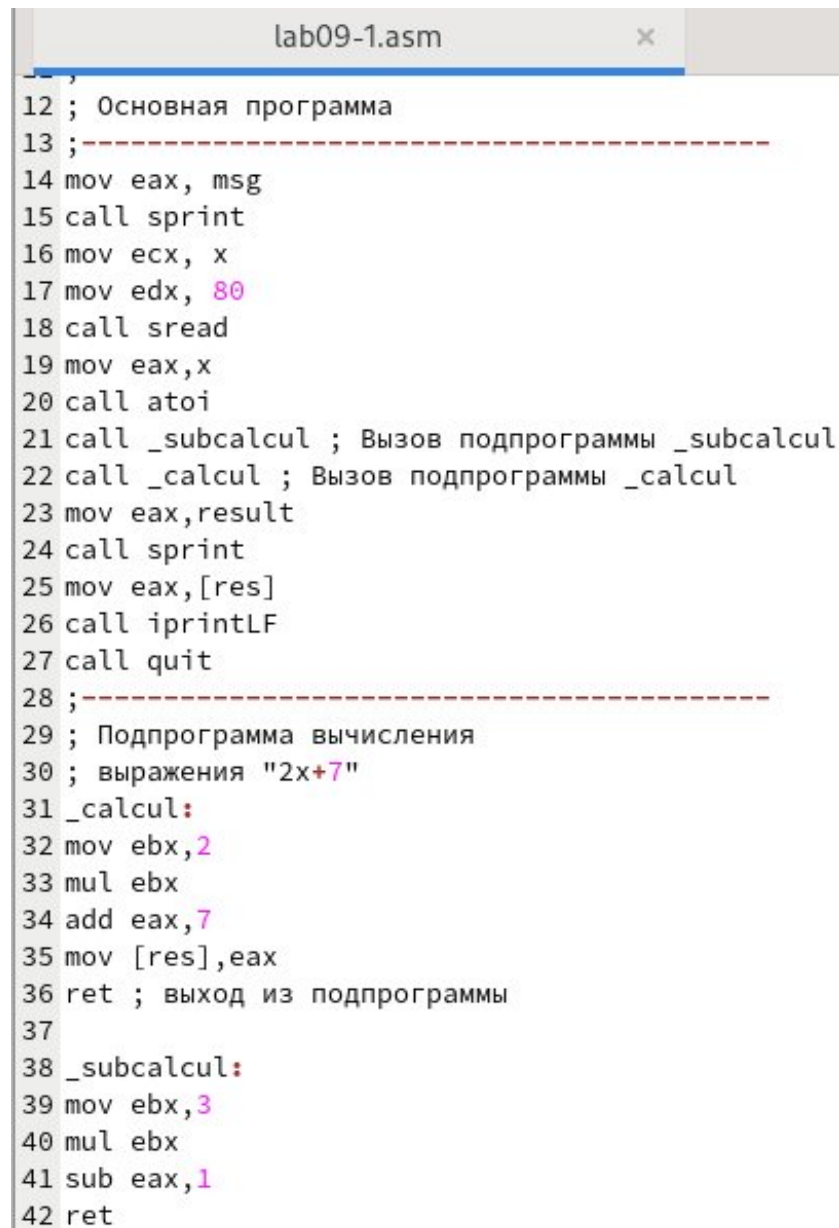


```
[aakayjnova@fedora lab09]$ nasm -f elf lab09-1.asm
[aakayjnova@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[aakayjnova@fedora lab09]$ ./lab09-1
Введите x: 2
2x+7=11
```

Рис. 4.3: Создание и запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму

_calcul для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x)=2x+7$,
 $g(x)=3x-1$



```
lab09-1.asm
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _subcalcul ; Вызов подпрограммы _subcalcul
22 call _calcul ; Вызов подпрограммы _calcul
23 mov eax, result
24 call sprint
25 mov eax, [res]
26 call iprintLF
27 call quit
28 ;-----
29 ; Подпрограмма вычисления
30 ; выражения "2x+7"
31 _calcul:
32 mov ebx, 2
33 mul ebx
34 add eax, 7
35 mov [res], eax
36 ret ; выход из подпрограммы
37
38 _subcalcul:
39 mov ebx, 3
40 mul ebx
41 sub eax, 1
42 ret
```

Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу

```
[aakayjnova@fedora lab09]$ nasm -f elf lab09-1.asm
[aakayjnova@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[aakayjnova@fedora lab09]$ ./lab09-1
Введите x: 2
2x+7=17
```

Рис. 4.5: Создание и запуск исполняемого файла

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm и вставляю туда текст программы из листинга 9.2

```
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: Ввод текста программы

Получаю исполняемый файл для работы с GDB с ключом '-g'

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

Рис. 4.7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик GDB

```
[aakayjnova@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.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. 4.8: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив её в GDB

```
(gdb) run
Starting program: /home/aakayjnova/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 10047) exited normally]
```

Рис. 4.9: Проверка работы файла

Устанавливаю брейкпоинт на метку `_start` и запускаю программу для более подробного анализа

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

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

Рис. 4.10: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы начиная с метки `_start` и переключаюсь на отображение команд с Intel'овским синтаксисом

```

(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) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:      mov     eax,0x4
    0x08049005 <+5>:      mov     ebx,0x1
    0x0804900a <+10>:     mov     ecx,0x804a000
    0x0804900f <+15>:     mov     edx,0x8
    0x08049014 <+20>:     int     0x80
    0x08049016 <+22>:     mov     eax,0x4
    0x0804901b <+27>:     mov     ebx,0x1
    0x08049020 <+32>:     mov     ecx,0x804a008
    0x08049025 <+37>:     mov     edx,0x7
    0x0804902a <+42>:     int     0x80
    0x0804902c <+44>:     mov     eax,0x1
    0x08049031 <+49>:     mov     ebx,0x0
    0x08049036 <+54>:     int     0x80
End of assembler dump.

```

Рис. 4.11: Просмотр кода программы

В режиме АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в Intel используется уже привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы


```
[ 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

native process 10152 In: _start
(gdb) layout regs
```

Рис. 4.12: Режим псевдографики

4.3 Добавление точек останова

Проверяю правильность установки точки останова по имени метки `_start`, устанавливаю ещё одну точку по адресу `mov ebx,0x0` и просматриваю информацию о всех установленных точках останова

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

[ 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

exec No process in:
(gdb) layout regs
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint       keep y  0x08049000 lab09-2.asm:9
(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
2        breakpoint       keep y  0x08049031 lab09-2.asm:20
```

Рис. 4.13: Установление точек останова и просмотр информации о них

4.4 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды `stepi` и слежу за изменением значений регистров

```

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

--Register group: general--
eax      0x0      0      ecx      0x0      0
edx      0x0      0      ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049000 0x8049000  <_start>  eflags    0x202     [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

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

native process 11639 In: _start L9
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1b0 0xffffd1b0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags    0x202     [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.14: До использования команды stepi

```

--Register group: general--
eax      0x8      8      ecx      0x804a000 134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>  eflags    0x202     [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

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

native process 11639 In: _start L14 PC:
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags    0x202     [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--stepics 0x23
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si 5

```

Рис. 4.15: После использования команды stepi

Изменились значения регистров eax,ecx,edx,ebx.

Просматриваю значение переменной msg1 и msg2 по их адресам

```

aakaynova@fedora:~/work/arch-pc/lab09 — gdb lab09-2
Register group: general
eax      0x8      8      ecx      0x804a000    134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd1b0 0xffffd1b0  ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>  eflags   0x202    [ IF ]
cs       0x23     35     ss       0x2b     43
ds       0x2b     43     es       0x2b     43
fs       0x0      0      gs       0x0      0

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

native process 11639 In: _start L14 PC: 0
ss      0x2b     43
ds      0x2b     43
es      0x2b     43
fs      0x0      0
gs      0x0      0
(gdb) si 5
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"

```

Рис. 4.16: Просмотр значений переменных

Изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2

```

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

```

Рис. 4.17: Изменение первого символа переменных

Вывожу в 16-тиричном формате, в 2-ичном формате и в символьном виде значение регистра edx

```
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
```

Рис. 4.18: Вывод значения регистра в разных форматах

Изменяю значение регистра ebx согласно заданию

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
```

Рис. 4.19: Изменение значения регистра

Разница вывода команд в том, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы и выхожу из GDB

```
native process 11639 In: _start
(gdb) p/c $edx
(gdb) continue
Continuing.
borld!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) quit
A debugging session is active.

        Inferior 1 [process 11639] will be killed.

Quit anyway? (y or n) |
```

Рис. 4.20: Завершение работы GDB

4.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm в файл lab09-2.asm и создаю исполняемый файл

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

Рис. 4.21: Копирование и создание файла

Загружаю исполняемый файл в отладчик GDB, указав необходимые аргументы

```
[aakayjnova@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 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...
```

Рис. 4.22: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю её

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/aakayjnova/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
```

Рис. 4.23: Установление точки останова и запуск программы

Просматриваю вершину стека и позиции стека по адресам

```
(gdb) x/x $esp
0xffffd180: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd333: "/home/aakayjnova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd35f: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd371: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd382: "2"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
```

Рис. 4.24: Просмотр значений, введённых в стек

Шаг изменения адреса равен 4, так как количество аргументов командной строки - 4.

4.6 Выполнение заданий для самостоятельной работы

Преобразовываю программу из лабораторной работы № 8 (задание № 1), реализовав вычисление значения функции $f(x)$ как подпрограмму

```

1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Результат: ',0h
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx
9 sub ecx,1
10 mov esi,0
11 call _next
12 _next:
13 cmp ecx, 0h
14 jz _end
15 mov ebx,4
16 pop eax
17 call atoi
18 mul ebx
19 add eax,3
20 add esi,eax
21 loop _next
22 _end:
23 mov eax,msg1
24 call sprint
25 mov eax,esi
26 call iprintLF
27 call quit

```

Рис. 4.25: Написание кода программы

Запускаю код и проверяю правильность его работы

```
[aakayjnova@fedora lab09]$ nasm -f elf task1.asm  
[aakayjnova@fedora lab09]$ ld -m elf_i386 -o task1 task1.o  
[aakayjnova@fedora lab09]$ ./task1 1 2 3  
Результат: 21
```

Рис. 4.26: Запуск программы и проверка её вывода

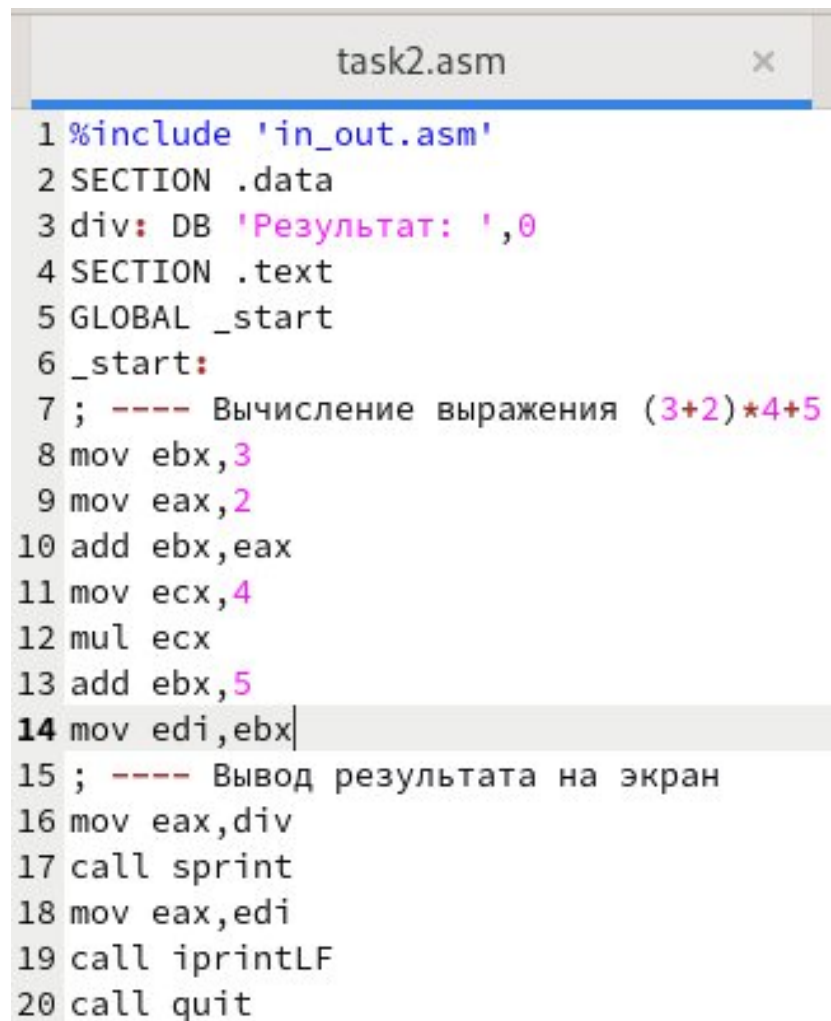
Ввожу в файл task2.asm текст программы из листинга 9.3

```
[aakayjnova@fedora lab09]$ nasm -f elf task2.asm  
[aakayjnova@fedora lab09]$ ld -m elf_i386 -o task2 task2.o  
[aakayjnova@fedora lab09]$ ./task2  
Результат: 10
```

Рис. 4.27: Ввод текста программы

Если программа работает верно, то на экран выведется “25”.

Создаю исполняемый файл и проверяю его



```
task2.asm
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 ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,ebx
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
```

Рис. 4.28: Создание и запуск исполняемого файла

Выводится неправильный ответ.

Получаю исполняемый файл для GDB, запускаю его и ставлю брейкпоинты на каждой инструкции, связанной с вычислениями. Прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

```
aakayjnova@fedora:~/work/arch-pc/lab09 — gdb task2

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi 0x8049000 <0x0n> push 0xb

0x80490f2 <_start+10> add ebx,eax
B+ 0x80490f4 <_start+12> mov ecx,0x4
B+ 0x80490f9 <_start+17> mul ecx
B+ 0x80490fb <_start+19> add ebx,0x5
b+ 0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
(gdb) layout regs
2 process 18747 In: _start 0x080490e8 task2.asm:8
(gdb) continue
Continuing.

Breakpoint 5, _start () at task2.asm:12
(gdb) continue
Continuing.

Breakpoint 6, _start () at task2.asm:13
```

Рис. 4.29: Поиск ошибки в GDB

Тут мы и получаем ошибку

```
aakayjnova@fedora:~/work/arch-pc/lab09 — gdb task2

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd1c0 0xffffd1c0
ebp      0x0      0x0
esi 0x8049000 <0x0n> push 0xb

0x80490f2 <_start+10> add ebx,eax
B+ 0x80490f4 <_start+12> mov ecx,0x4
B+ 0x80490f9 <_start+17> mul ecx
B+ 0x80490fb <_start+19> add ebx,0x5
B+ 0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
(gdb) layout regs
2 process 18747 In: _start 0x080490e8 task2.asm:8
(gdb) continue
Continuing.

Breakpoint 6, _start () at task2.asm:13
(gdb) continue
Continuing.

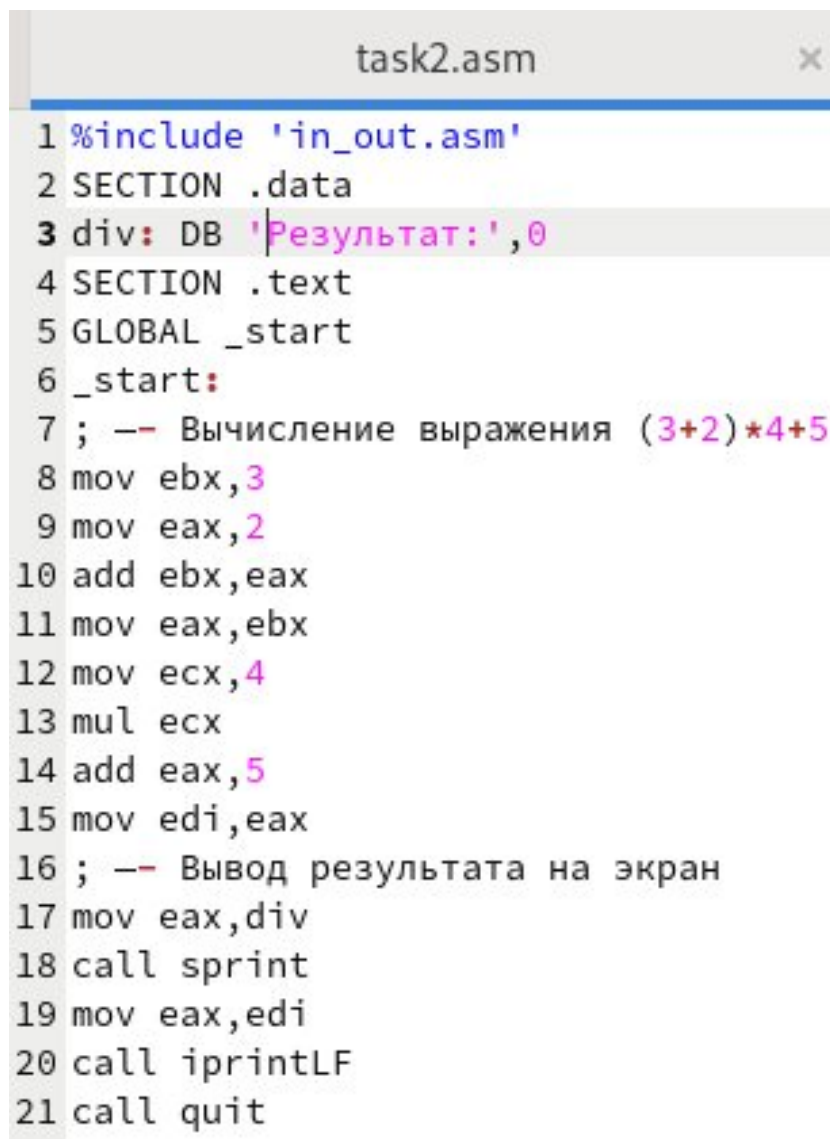
Breakpoint 7, _start () at task2.asm:14
```

Рис. 4.30: Ошибка в программе

Мы получаем неправильный ответ при выполнении инструкции mul ecx, во

время которой есх умножается на еах, то есть 4 умножается на 2, вместо правильного умножения на 5(то есть на регистр ебх). Причина в том, что инструкция dd ебх,еах (перед mov есх,4) никак не связана с mul есх, однако с mul есх связана инструкция mov еах,2.

Исправлю ошибку, добавив необходимые инструкции



```
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
```

Рис. 4.31: Правка текста программы

Создаю исполняемый файл и проверяю его

```
[aakaynova@fedora lab09]$ nasm -f elf task2.asm  
[aakaynova@fedora lab09]$ ld -m elf_i386 -o task2 task2.o  
[aakaynova@fedora lab09]$ ./task2  
Результат:25
```

Рис. 4.32: Создание и проверка файла

Выводится правильный результат, значит ошибка исправлена.

5 Выводы

В ходе данной лабораторной работы мы научились писать программы с использованием подпрограмм и ознакомились с методами отладки через GDB.

Список литературы

- [illegible]