

Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров

Симонова Виктория Игоревна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Понятие об отладке	7
3.2	Методы отладки	8
3.3	Понятие подпрограммы	8
4	Выполнение лабораторной работы	10
4.1	Реализация подпрограмм в NASM	10
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	23
4.6	Задания для самостоятельной работы	24
5	Выводы	31
	Список литературы	32

Список иллюстраций

4.1	Создание каталога и файла	10
4.2	Ввожу код программы	11
4.3	Запускаю исполняемый файл	11
4.4	Добавляю подпрограмму	12
4.5	Запускаю исполняемый файл	12
4.6	Создаю файл	13
4.7	Ввожу код программы вывода	13
4.8	Запускаю исполняемый файл	14
4.9	Загрузка	14
4.10	Проверка	14
4.11	Установка и запуск	15
4.12	Использование команд	16
4.13	Включение режима псеадографики	17
4.14	Установление точек останова и просмотр информации о них	18
4.15	До использование команды stepi	19
4.16	После использование команды stepi	20
4.17	Просмотр	20
4.18	Изменение переменных	21
4.19	Вывод регистра в разных форматах	21
4.20	Изменение значения регистра	22
4.21	Завершение работы	22
4.22	Копирую и загружаю в отладчик	23
4.23	Запуск программы с точкой	23
4.24	Просмотр значений	24
4.25	Ввожу код в файл	25
4.26	Запускаю исполняемый файл	25
4.27	Исходный код	27
4.28	Запускаю исполняемый файл	27
4.29	Передача	28
4.30	Передача	28
4.31	Неверное изменение	29
4.32	Исправленная программа	29
4.33	Запускаю исполняемый файл	30

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и 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. Задания для самостоятельной работы.

3 Теоретическое введение

3.1 Понятие об отладке

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, отрабатывает, 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.

3.2 Методы отладки

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.

3.3 Понятие подпрограммы

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.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm (рис. [4.1]).

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

Рис. 4.1: Создание каталога и файла

Ввожу в файл текст программы из листинга с использованием подпрограмм (рис. [4.2]).

```

#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

```

1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть

Рис. 4.2: Ввожу код программы

Создаю исполняемый файл и проверяю его работу (рис. [4.3]).

```

[visimonova@fedora lab09]$ nasm -f elf lab09-1.asm
[visimonova@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[visimonova@fedora lab09]$ ./lab09-1
Введите x: 3
2x+7=13
[visimonova@fedora lab09]$

```

Рис. 4.3: Запускаю исполняемый файл

Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. [4.4]).

```

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

```

1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Удс

Рис. 4.4: Добавляю подпрограмму

Создаю исполняемый файл и проверяю его работу (рис. [4.5]).

```

[visimonova@fedora lab09]$ nasm -f elf lab09-1.asm
[visimonova@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[visimonova@fedora lab09]$ ./lab09-1
Введите x: 3
2x+7=23
[visimonova@fedora lab09]$

```

Рис. 4.5: Запускаю исполняемый файл

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга (рис. [4.6]).

```
[visimonova@fedora lab09]$ touch lab09-2.asm  
[visimonova@fedora lab09]$
```

Рис. 4.6: Создаю файл

Ввожу код программы(рис. [4.7]).

```
lab09-2.asm [ M ] 8 L. [ 1720  
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  
1Помощь 2Сохранить 3Блок 4Замена 5Копия
```

Рис. 4.7: Ввожу код программы вывода

Создаю исполняемый файл и проверяю его работу (рис. [4.8]).

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

Рис. 4.8: Запускаю исполняемый файл

Загружаю исполняемый файл в отладчик (рис. [4.9]).

```
[visimonova@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. 4.9: Загрузка

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. [4.10]).

```
[Inferior 1 (process 3980) exited normally]
(gdb) run
Starting program: /home/visimonova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3980) exited normally]
(gdb)
```

Рис. 4.10: Проверка

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаю её.(рис. [4.11]).

```
(gdb) break _start
Breakpoint 1 at 0x4010e0: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/visimonova/work/arch-pc/lab09/lab09-2

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

Рис. 4.11: Установка и запуск

Посматриваю дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` и переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. [4.12]).

```

(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
    0x0040110c <+44>:     mov     eax,0x1
    0x00401111 <+49>:     mov     ebx,0x0
    0x00401116 <+54>:     int     0x80
End of assembler dump.
(gdb)

```

Рис. 4.12: Использование команд

В режиме АТТ имена операндов начинаются со знака “\$”, а имена регистров со знака “%”. Включите режим псевдографики для более удобного анализа программы (рис. [4.13]).

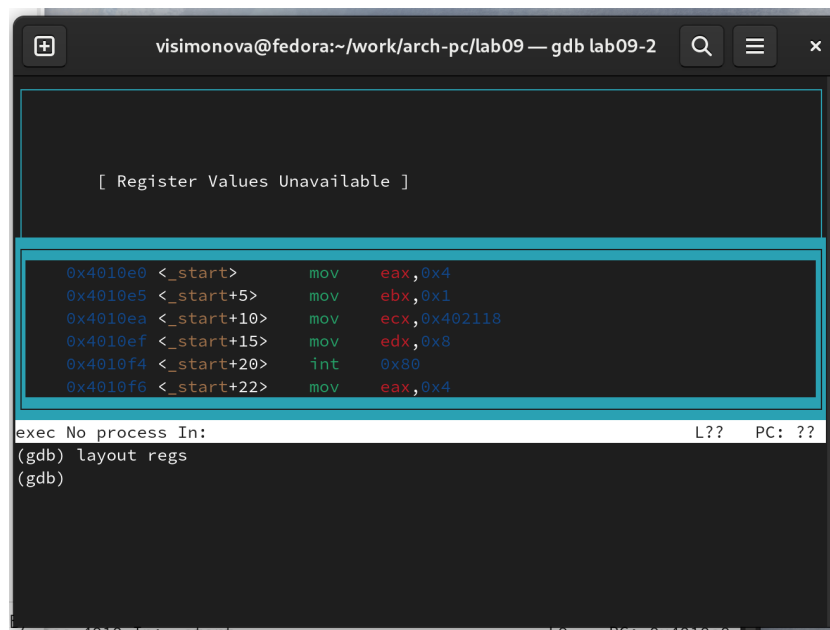


Рис. 4.13: Включение режима псеадографики

4.3 Добавление точек останова

Проверяю точку останова по метке `_start` (с помощью команды `i b`) и устанавливаю ещё одну точку останова по адресу инструкции `mov ebx,0x0`, и просматриваю информацию о всех точках останова. (рис. [4.14]).

```
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

native process 6819 In: _start L9 PC: 0x
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2        breakpoint keep y  0x08049031 lab09-2.asm:20
(gdb)
```

Рис. 4.14: 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 значения регистров (рис. [4.15]).

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>

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 7430 In: _start L9 PC: 0
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd160 0xffffd160
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. 4.15: До использование команды stepi

После использования команды (рис. [4.16]).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

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 7430 In: _start L14 PC: 0x8049016
eip      0x8049000 0x8049000 <_start>
eflags   0x202      [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--ccs
35
ss       0x2b      43
ds       0x2b      43
es       0x2b      43
fs       0x0      0
gs       0x0      0
(gdb) si 5

```

Рис. 4.16: После использование команды stepi

Изменились значения регистров eax,ebx,ecx,edx. Просматриваю значение msg1 с помощью команды x/1sb &msg1 (рис. [4.17]).

```

(gdb) stepi
(gdb) x/1sb &msg1
0x402118 <msg1>: "Hello, "
0x402120 <msg2>: "world!\n\034"
(gdb)

```

Рис. 4.17: Просмотр

Изменяю первый символ переменной msg1, заменяю любой символ во второй переменной msg2 (рис. [4.18]).

```

(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. 4.18: Изменение переменных

Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx. (рис. [4.19]).

```

--Register group: general--
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd160 0xffffd160
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]

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

native process 7430 In: _start L14 PC: 0x8049016
(gdb) p/x $edx
8 = 0x8
(gdb) p/t $edx
9 = 1000
(gdb) p/c $edx
10 = 8 '\b'

```

Рис. 4.19: Вывод регистра в разных форматах

С помощью команды set измените значение регистра ebx:(рис. [4.20]).

```

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
> 0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008

native process 19345 In: _start L14 PC: 0x8049016
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb)

```

Рис. 4.20: Изменение значения регистра

Разница: в первом случае символ переведён в строковый вид, во втором случае число в строковом виде не изменяется

Завершаю выполнение программы с помощью команды continue, выхожу командой quit (рис. [4.21]).

```

Register group: general
eax      0x1      1
ecx      0x804a008 134520840
edx      0x7      7
ebx      0x1      1
esp      0xffffd160 0xffffd160
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049031 0x8049031 <_start+49>

0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
B+> 0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al

native process 7430 In: _start L20 PC: 0x8049031
$14 = 2
(gdb) c
Continuing.
world!
Breakpoint 2, _start () at lab09-2.asm:20
(gdb) q
A debugging session is active.

Inferior 1 [process 7430] will be killed.

Quit anyway? (y or n)

```

Рис. 4.21: Завершение работы

4.5 Обработка аргументов командной строки в GDB

Скопирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm. Создаю исполняемый файл. Загружаю исполняемый файл в отладчик, используя ключ `--args`, указав аргументы (рис. [4.22]).

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

Рис. 4.22: Копирую и загружаю в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю программу (рис. [4.23]).

```
(gdb) b _start
Breakpoint 1 at 0x4011a8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/visimonova/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]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) █
```

Рис. 4.23: Запуск программы с точкой

Просматриваю позиции стека по адресам (рис. [4.24]).

```

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd150:  0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd30f:  "/home/visimonova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd33b:  "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd34d:  "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd35e:  "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd360:  "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 4.24: Просмотр значений

Аргументов командной строки 4, поэтому и шаг изменения адреса равен четырём.

4.6 Задания для самостоятельной работы

1. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. [4.25]).


```

task1.asm      [----]  0 L: [ 1+ 5  6/ 36] *(88 /1466b) 0095 0x05F  [*
#include 'in_out.asm'
SECTION .data
msg db "Ответ: ",0
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

```

Рис. 4.25: Ввожу код в файл

Создаю исполняемый файл и проверяю его работу (рис. [4.26]).

```

[visimonova@fedora lab09]$ nasm -f elf task1.asm
[visimonova@fedora lab09]$ ld -m elf_i386 -o task1 task1.o
[visimonova@fedora lab09]$ ./task1
Ответ: 0
[visimonova@fedora lab09]$ mc

[visimonova@fedora lab09]$ ./task1 1 2 3
Ответ: 93
[visimonova@fedora lab09]$

```

Рис. 4.26: Запускаю исполняемый файл

Код первого задания:

```

#include 'in_out.asm'
SECTION .data
msg db "Ответ: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в ecx количество

```

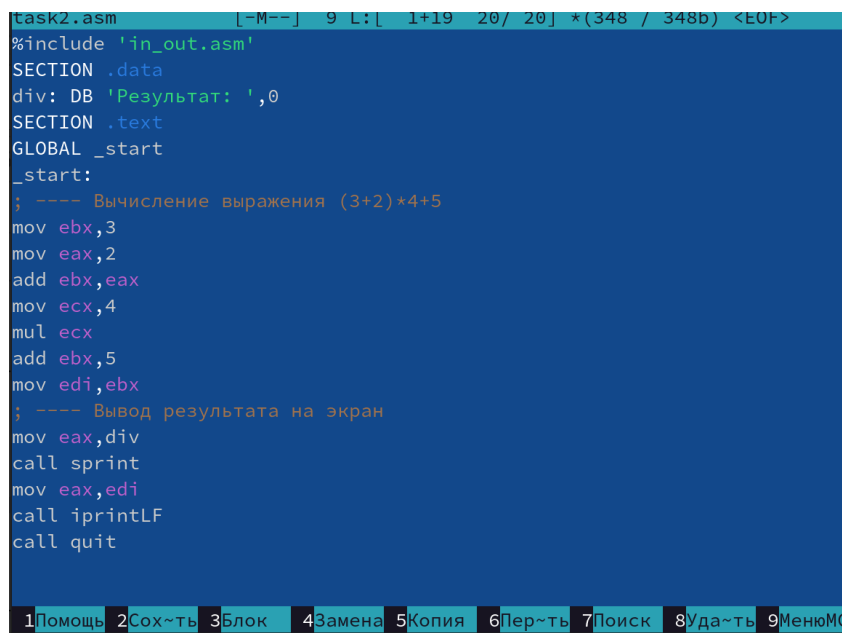
```

; 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
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент esi=esi+eax
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр eax
call iprintLF ; печать результата
call quit ; завершение программы
_calcul:
mov ebx,12
mul ebx
sub eax,-7
ret

```

2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$.

(рис. [4.27]).



```
task2.asm [-M--] 9 L: [ 1+19 20/ 20] *(348 / 348b) <EOF>
#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 iprintfLF
call quit
```

Рис. 4.27: Исходный код

При запуске данная программа дает неверный результат. Проверяю это.(рис. [4.28]).



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

Рис. 4.28: Запускаю исполняемый файл

С помощью отладчика GDB, анализируя изменения значений регистров, определяю ошибку и исправляю ее.

Передаю в отладчик (рис. [4.29]).

```
[visimonova@fedora lab09]$ nasm -f elf -g -l task2.lst task2.asm
[visimonova@fedora lab09]$ ^C
[visimonova@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[visimonova@fedora lab09]$ gdb task2
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/gplv3.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 task2...
(No debugging symbols found in task2)
(gdb)
```

Рис. 4.29: Передача

Ставлю брейкпоинты для всех вычислительных инструкций, прохожусь по ним. Ошибка: в момент выполнения инструкции `mul ecx(4)` на `eax(2)`, а умножать надо на `ebx(5)`. Нужно инструкцию `add ebx, eax` связать с `mul ecx`

Причина (рис. [4.30]).

```
Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

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

native process 14573 In: _start L13 PC: 0x8049
continuing.

Breakpoint 5, _start () at task2.asm:12
(gdb) c
continuing.

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

Рис. 4.30: Передача

Значение регистра (рис. [4.31]).

```

--Register group: general--
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd160 0xffffd160
ebp      0x0      0x0

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>

native process 14573 In: _start L14 PC
Continuing.

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

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

```

Рис. 4.31: Неверное изменение

Ввожу исправленную программу в файл (рис. [4.32]).

```

task2.asm  [----]  9 L: [ 1+14 15/ 21] *(243 / 360b) 0097 0x0
#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. 4.32: Исправленная программа

Создаю исполняемый файл и проверяю его работу , всё работает корректно(рис. [4.33]).

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

Рис. 4.33: Запускаю исполняемый файл

Код программы задания 2:

```
%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

Приобрела навыки написания программ с использованием подпрограмм, познакомилась с методами отладки при помощи GBD и его основными возможностями.

Список литературы

::: {#refs} ::