

Отчет по лабораторной работе №9

Дисциплина: архитектура компьютера

Гомазкова Алина

Содержание

| | | |
|----------|---|-----------|
| 1 | Цель работы | 5 |
| 2 | Задание | 6 |
| 3 | Теоретическое введение | 7 |
| 4 | Выполнение лабораторной работы | 10 |
| 4.1 | Реализация подпрограмм в NASM | 10 |
| 4.2 | Отладка программ с помощью GDB | 13 |
| 4.3 | Добавление точек останова | 16 |
| 4.4 | Работа с данными программы в GDB | 17 |
| 4.5 | Обработка аргументов командной строки в GDB | 22 |
| 4.6 | Задания для самостоятельной работы | 24 |
| 5 | Выводы | 29 |
| | Список литературы | 30 |

Список иллюстраций

| | | |
|------|---|----|
| 4.1 | Рис. 1 Создание файлов для лабораторной работы | 10 |
| 4.2 | Рис. 2 Ввод текста из листинга 9.1 | 11 |
| 4.3 | Рис. 3 Запуск исполняемого файла | 11 |
| 4.4 | Рис. 4 Изменение текста программы согласно заданию | 12 |
| 4.5 | Рис. 5 Запуск обновленной программы | 12 |
| 4.6 | Рис. 6 Ввод текста программы из листинга 9.2 | 13 |
| 4.7 | Рис. 7 Получение исполняемого файла | 13 |
| 4.8 | Рис. 8 Загрузка исполняемого файла в отладчик | 14 |
| 4.9 | Рис. 9 Проверка работы файла с помощью команды run | 14 |
| 4.10 | Рис. 10 Установка брейкпоинта и запуск программы | 14 |
| 4.11 | Рис. 11 Использование команд disassemble и disassembly-flavor intel | 15 |
| 4.12 | Рис. 12 Включение режима псевдографики | 16 |
| 4.13 | Рис. 13 Установление точек останова и просмотр информации о них | 17 |
| 4.14 | Рис. 14 До использования команды stepi | 18 |
| 4.15 | Рис. 15 После использования команды stepi | 19 |
| 4.16 | Рис. 16 Просмотр значений переменных | 19 |
| 4.17 | Рис. 17 Использование команды set | 20 |
| 4.18 | Рис. 18 Вывод значения регистра в разных представлениях | 20 |
| 4.19 | Рис. 19 Использование команды set для изменения значения регистра | 21 |
| 4.20 | Рис. 20 Завершение работы GDB | 22 |
| 4.21 | Рис. 21 Создание файла | 22 |
| 4.22 | Рис. 22 Загрузка файла с аргументами в отладчик | 23 |
| 4.23 | Рис. 23 Установление точки останова и запуск программы | 23 |
| 4.24 | Рис. 24 Просмотр значений, введенных в стек | 24 |
| 4.25 | Рис. 25 Написание кода подпрограммы | 24 |
| 4.26 | Рис. 26 Запуск программы и проверка его вывода | 25 |
| 4.27 | Рис. 27 Ввод текста программы из листинга 9.3 | 25 |
| 4.28 | Рис. 28 Создание и запуск исполняемого файла | 25 |
| 4.29 | Рис. 29 Нахождение причины ошибки | 26 |
| 4.30 | Рис. 30 Нахождение причины ошибки | 27 |
| 4.31 | Рис. 31 Исправление ошибки | 27 |
| 4.32 | Рис. 32 Ошибка исправлена | 28 |

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

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 `esp`. 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 лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm (рис.1)

```
[alinagomazkova@10 ~]$ mkdir -p ~/work/arch-pc/lab09  
[alinagomazkova@10 ~]$ cd ~/work/arch-pc/lab09  
[alinagomazkova@10 lab09]$ touch lab09-1.asm  
[alinagomazkova@10 lab09]$
```

Рис. 4.1: Рис. 1 Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1 (рис. 2)

```

mc [alinagomazkova@10.0.2.15]:~/work/arch-pc/lab09
/home/alinagomazkova/work/arch-pc/lab09/lab09-1.asm
%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 _subcalcul ; Вызов подпрограммы _calcul

```

Рис. 4.2: Рис. 2 Ввод текста из листинга 9.1

Создаю исполняемый файл и проверяю его работу (рис. 3)

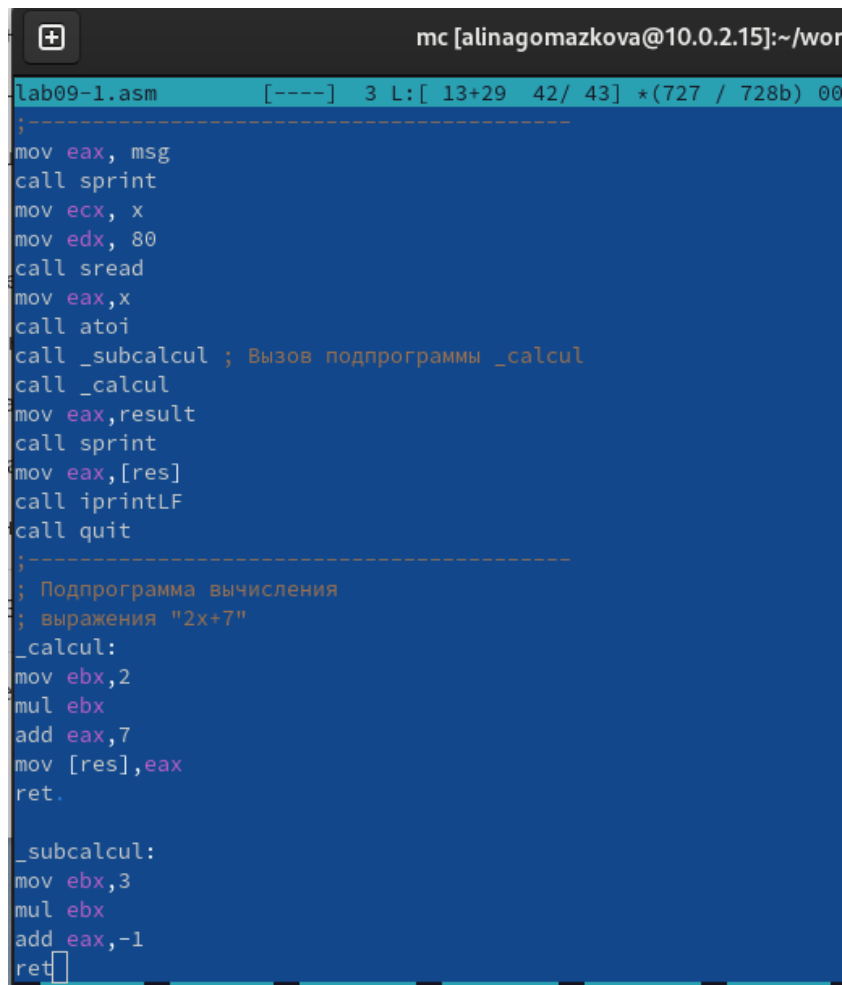
```

[alinagomazkova@10 lab09]$ nasm -f elf lab09-1.asm
[alinagomazkova@10 lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[alinagomazkova@10 lab09]$ ./lab09-1
Введите x: 6
2x+7=19

```

Рис. 4.3: Рис. 3 Запуск исполняемого файла

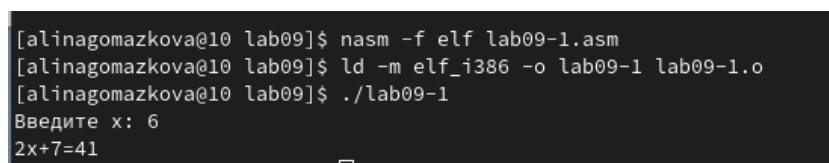
Изменяю текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul` для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$ (рис. 4)



```
lab09-1.asm [----] 3 L: [ 13+29 42/ 43] *(727 / 728b) 00
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _subcalcul ; Вызов подпрограммы _calcul
call _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.
_subcalcul:
mov ebx, 3
mul ebx
add eax, -1
ret
```

Рис. 4.4: Рис. 4 Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу (рис. 5)

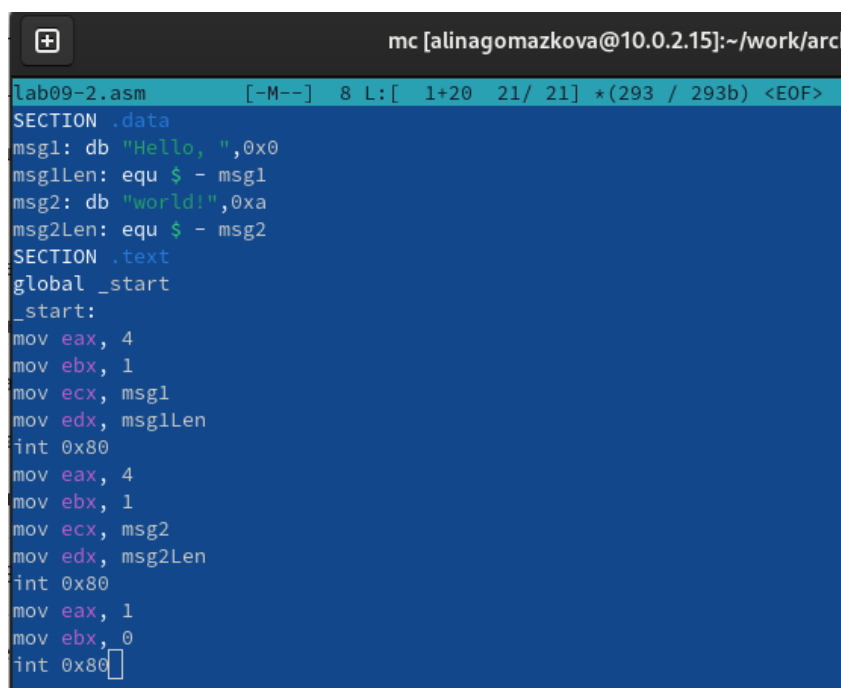


```
[alinagomazkova@10 lab09]$ nasm -f elf lab09-1.asm
[alinagomazkova@10 lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[alinagomazkova@10 lab09]$ ./lab09-1
Введите x: 6
2x+7=41
```

Рис. 4.5: Рис. 5 Запуск обновленной программы

4.2 Отладка программ с помощью GDB

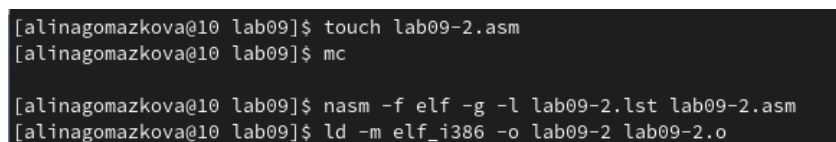
Создаю файл lab09-2.asm с текстом программы из Листинга 9.2 (рис. 6)



```
mc [alinagomazkova@10.0.2.15]:~/work/arc
lab09-2.asm [-M--] 8 L:[ 1+20 21/ 21] *(293 / 293b) <EOF>
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Рис. 4.6: Рис. 6 Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g' (рис. 7)



```
[alinagomazkova@i0 lab09]$ touch lab09-2.asm
[alinagomazkova@i0 lab09]$ mc
[alinagomazkova@i0 lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[alinagomazkova@i0 lab09]$ ld -m elf_i386 -o lab09-2 lab09-2.o
```

Рис. 4.7: Рис. 7 Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb (рис. 8)

```
[alinagomazkova@i0 lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora Linux 13.2-3.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) 
```

Рис. 4.8: Рис. 8 Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. 9)

```
(gdb) run
Starting program: /home/alinagomazkova/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 4234) exited normally]
(gdb) 
```

Рис. 4.9: Рис. 9 Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start и запускаю её (рис. 10)

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

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

Рис. 4.10: Рис. 10 Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды

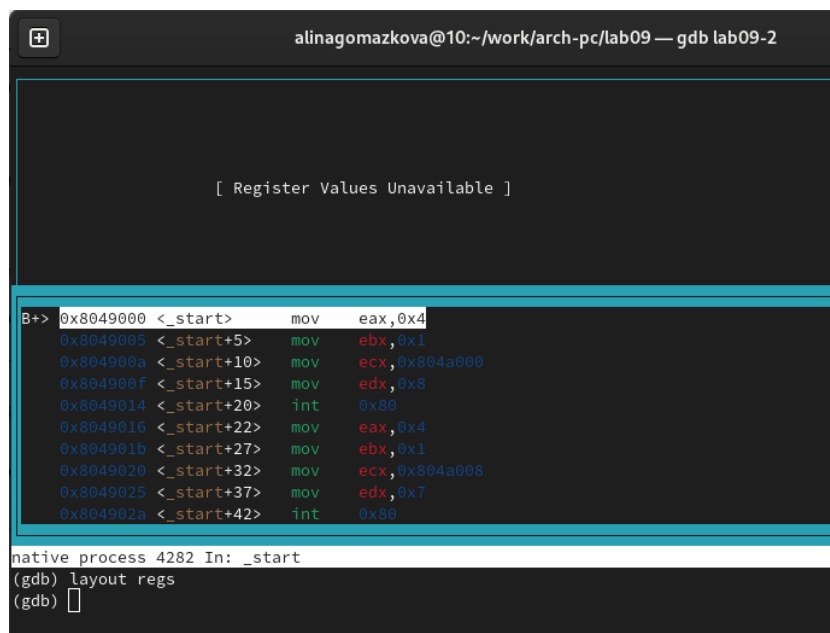
disassemble, начиная с метки _start, и переключаясь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel (рис. 11)

```
(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: Рис. 11 Использование команд disassemble и disassembly-flavor intel

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

псевдографики для более удобного анализа программы с помощью команд `layout asm` и `layout regs` (рис. 12)



The screenshot shows a GDB terminal window with the title bar "alinagomazkova@10:~/work/arch-pc/lab09 — gdb lab09-2". The main area displays the message "[Register Values Unavailable]". Below this, a list of assembly instructions is shown, starting with "B+> 0x8049000 <_start>". The instructions are: `mov eax,0x4`, `mov ebx,0x1`, `mov ecx,0x804a000`, `mov edx,0x8`, `int 0x80`, `mov eax,0x4`, `mov ebx,0x1`, `mov ecx,0x804a008`, `mov edx,0x7`, and `int 0x80`. At the bottom, the text "native process 4282 In: _start" is visible, followed by the commands `(gdb) layout regs` and `(gdb)` .

Рис. 4.12: Рис. 12 Включение режима псевдографики

4.3 Добавление точек останова

Проверяю, что точка останова по имени метки `_start` установлена с помощью команды `info breakpoints` и устанавливаю еще одну точку останова по адресу инструкции `mov ebx,0x0`. Просматриваю информацию о всех установленных точках останова (рис. 13)


```

[ 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

native process 4282 In: _start
(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. 4.13: Рис. 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 (рис. 14) (рис. 15)

```

+ alinagomazkova@10:~/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
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>

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 4282 In: _start
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 4.14: Рис. 14 До использования команды stepi

```

alinagomazkova@10:~/work/arch-pc/lab09 — gdb lab09-2
--Register group: general--
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>

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 4282 In: _start L15 PC: 0x804901b
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
--Type <RET> for more, q to quit, c to continue without paging--
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si 5
(gdb)

```

Рис. 4.15: Рис. 15 После использования команды stepi

Изменились значения регистров eax, ecx, edx и ebx.

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу (рис. 16)

```

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 4282 In: _start L15 PC: 0x804901b
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
Invalid number "804a008".
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb)

```

Рис. 4.16: Рис. 16 Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2 (рис. 17)

```

(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"
(gdb)

```

Рис. 4.17: Рис. 17 Использование команды set

Вывожу в шестнадцатеричном формате, в двоичном формате и в символьном виде соответственно значение регистра `edx` с помощью команды `print p/F $val` (рис. 18)

The screenshot shows a GDB window titled "alinagomazkova@10:~/work/arch-pc/lab09 — gdb lab09-2". The "Register group: general" window displays the following values:

| Register | Value | Comment |
|----------|------------|-----------------------|
| eax | 0x4 | 4 |
| ecx | 0x804a000 | 134520832 |
| edx | 0x8 | 8 |
| ebx | 0x1 | 1 |
| esp | 0xffffd1a0 | 0xffffd1a0 |
| ebp | 0x0 | 0x0 |
| esi | 0x0 | 0 |
| edi | 0x0 | 0 |
| eip | 0x804901b | 0x804901b <_start+27> |

The command history at the bottom shows the following sequence of commands and their outputs:

```

(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2 = 'b'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "borld!\n\034"
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'

```

Рис. 4.18: Рис. 18 Вывод значения регистра в разных представлениях

С помощью команды `set` изменяю значение регистра `ebx` в соответствии с заданием (рис. 19)

```

alinagomazkova@10:~/work/arch-pc/lab09 — gdb lab09-2

Register group: general
eax      0x4      4
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x2      2
esp      0xffffd1a0 0xffffd1a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>

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 4282 In: _start
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/c $edx
$3 = 8 '\b'
(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2

```

Рис. 4.19: Рис. 19 Использование команды set для изменения значения регистра

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы с помощью команды continue и выхожу из GDB с помощью команды quit (рис. 20)

```

eax      0x4      4
eax      0x1      1
ecx      0x804a008 134520840
edx      0x7      7
esp      0xffffd1a0 1xffffd1a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x804901b 0x804901b <_start+27>
eip      0x8049031 0x8049031 <_start+49>
B+ 0x8049000 <_start>    mov     eax,0x4
   0x804901b <_start+27> mov     ebx,0x1
   0x8049020 <_start+32> mov     ecx,0x804a008
   0x8049025 <_start+37> mov     edx,0x7
   0x804902a <_start+42> int     0x80
   0x804902c <_start+44> mov     eax,0x1
B+> 0x8049031 <_start+49> mov     ebx,0x0
   0x8049036 <_start+54> int     0x80 04a008
   0x8049038 add     BYTE PTR [eax],al
   0x804903a add     BYTE PTR [eax],al
   0x804903c add     BYTE PTR [eax],al
native process 4282 In: _start
$2 = 1000
$5 = 2
(gdb) c
Continuing.
borld!

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

    Inferior 1 [process 4282] will be killed.

```

Рис. 4.20: Рис. 20 Завершение работы GDB

4.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл (рис. 21)

```

[alinagomazkova@10 lab09]$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
[alinagomazkova@10 lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[alinagomazkova@10 lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o

```

Рис. 4.21: Рис. 21 Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргумен-

ты с использованием ключа `-args` (рис. 22)

```
[alinagomazkova@10 lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.2-3.fc38
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb) █
```

Рис. 4.22: Рис. 22 Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее (рис. 23)

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

Рис. 4.23: Рис. 23 Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам (рис. 24)

```

(gdb) x/x $esp
0xffffd150: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd30b: "/home/alina gomazkova/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: Рис. 24 Просмотр значений, введенных в стек

Шаг изменения адреса равен 4, т.к. количество аргументов командной строки равно 4.

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. 25)

```

mc [alinagomazkova@10.0.2.15]:~/work/arch-pc/lab09
task1.asm [----] 3 L: [ 1+16 17/ 17] *(159 / 159b) <EOF>
3 .next:
pop eax
call atoi
add eax,2
mul edi
add esi,eax
cmp ecx,0h
jz .done
loop .next
.done:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
ret

```

Рис. 4.25: Рис. 25 Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно (рис. 26)

```
[alinagomazkova@10 lab09]$ cp ~/work/arch-pc/lab08/task.asm ~/work/arch-pc/lab09/task1.asm
[alinagomazkova@10 lab09]$ nasm -f elf task1.asm
[alinagomazkova@10 lab09]$ ld -m elf_i386 -o task1 task1.o
[alinagomazkova@10 lab09]$ ./task 1 2 3
bash: ./task: Нет такого файла или каталога
[alinagomazkova@10 lab09]$ ./task1 1 2 3
Результат: 48
[alinagomazkova@10 lab09]$
```

Рис. 4.26: Рис. 26 Запуск программы и проверка его вывода

2. Ввожу в файл task1.asm текст программы из листинга 9.3 (рис. 27)

```
mc [alinagomazkova@10.0.2.15]:~/work/arch-pc/lab09
task2.asm          [----]  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,ebx
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 4.27: Рис. 27 Ввод текста программы из листинга 9.3

При корректной работе программы должно выводиться “25”. Создаю исполняемый файл и запускаю его (рис. 28)

```
[alinagomazkova@10 lab09]$ touch task2.asm
[alinagomazkova@10 lab09]$ mc

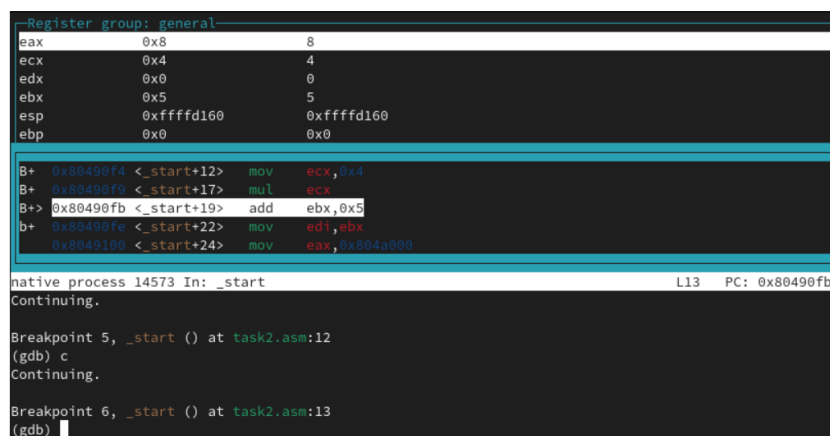
[alinagomazkova@10 lab09]$ nasm -f elf task2.asm
[alinagomazkova@10 lab09]$ nasm -f elf task2.asm
[alinagomazkova@10 lab09]$ ld -m elf_i386 -o task2 task2.o
[alinagomazkova@10 lab09]$ ./task2
Результат: 10
```

Рис. 4.28: Рис. 28 Создание и запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ.

Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды `continue` прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

При выполнении инструкции `mul ecx` происходит умножение `ecx` на `eax`, то есть 4 на 2, вместо умножения 4 на 5 (регистр `ebx`). Происходит это из-за того, что стоящая перед `mov ecx,4` инструкция `add ebx,ecx` не связана с `mul ecx`, но связана инструкция `mov eax,2` (рис. 29)



```
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,ecx
b+ 0x80490fe <_start+22> mov    edi,ebx
0x8049100 <_start+24> mov    eax,0x504a000

native process 14573 In: _start L13 PC: 0x80490fb
Continuing.

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

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

Рис. 4.29: Рис. 29 Нахождение причины ошибки

Из-за этого мы получаем неправильный ответ (рис. 30)

```

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: 0x80490fe
Continuing.

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

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

```

Рис. 4.30: Рис. 30 Нахождение причины ошибки

Исправляем ошибку, добавляя после `add ebx,eax` `mov eax,ebx` и заменяя `ebx` на `eax` в инструкциях `add ebx,5` и `mov edi,ebx` (рис. 31)

```

mc [alinagomazkova@10.0.2.15]:~/work/arch-pc/lab09
task2.asm  [----] 6 L:[ 1+13 14/ 21] *(230 / 360b) 0120 0x078
#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. 4.31: Рис. 31 Исправление ошибки

Также, вместо того, чтобы изменять значение `eax`, можно было изменять значение неиспользованного регистра `edx`.

Создаем исполняемый файл и запускаем его. Убеждаемся, что ошибка исправлена (рис. 32)

```
[alinagomazkova@10 lab09]$ nasm -f elf task2.asm  
[alinagomazkova@10 lab09]$ ld -m elf_i386 -o task2 task2.o  
[alinagomazkova@10 lab09]$ ./task2  
Результат: 25
```

Рис. 4.32: Рис. 32 Ошибка исправлена

5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1.Архитектура ЭВМ