

Лабораторная работа №6

Дисциплина: Архитектура компьютера

Савостин Олег

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 | 12 |
| 4.3 | Выполнение самостоятельной работы. | 22 |
| 5 | Выводы | 27 |
| | Список литературы | 28 |

Список иллюстраций

| | | |
|------|--|----|
| 4.1 | Создание файлов | 10 |
| 4.2 | Листинг 9.1 | 11 |
| 4.3 | Исполняемый файл Листинга 9.1 | 11 |
| 4.4 | Измененный Листинг | 12 |
| 4.5 | Запуск исполняемого файла | 12 |
| 4.6 | Листинг 9.2 | 13 |
| 4.7 | Создание исполняемых файлов | 13 |
| 4.8 | Использование gdb | 14 |
| 4.9 | Disassembly Intel | 15 |
| 4.10 | layout asm | 16 |
| 4.11 | layout regs | 16 |
| 4.12 | Проверка на наличие меток | 17 |
| 4.13 | Создание метки | 17 |
| 4.14 | Регистры si | 18 |
| 4.15 | Значения и изменения msg1 msg2 | 19 |
| 4.16 | Изменения значения регистра | 19 |
| 4.17 | Значения регистров | 20 |
| 4.18 | Выход | 20 |
| 4.19 | Файл из 8 Лабораторной работы | 21 |
| 4.20 | Позиции стека | 22 |
| 4.21 | Измененный код | 23 |
| 4.22 | Проверка на правильность | 23 |
| 4.23 | Проверка на правильность файла из Листинга 9.3 | 24 |
| 4.24 | Запуск gdb | 24 |
| 4.25 | Регистры. | 25 |
| 4.26 | Ошибочный код | 25 |
| 4.27 | Исправленный код | 26 |
| 4.28 | Результат исправления кода | 26 |

Список таблиц

1 Цель работы

Целью работы является 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. 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, 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.

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

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

Синтаксис команды для запуска отладчика имеет следующий вид: `gdb [опции] [имя_файла | ID процесса]`

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`. Посмотреть дизассемблированный код программы можно с помощью команды `disassemble`: `(gdb) disassemble _start`

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»: `(gdb) break *` `(gdb) b` Информацию о всех установленных точках останова можно вывести командой `info` (кратко `i`): `(gdb) info breakpoints` `(gdb) i b` Как уже упомина-

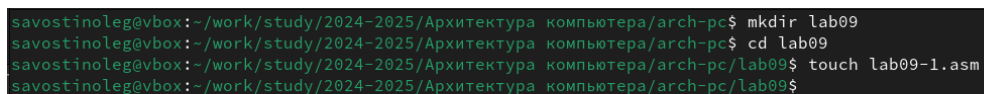
лось, отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Посмотреть содержимое регистров можно с помощью команды `info registers` (или `i r`): `(gdb) info registers`

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

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создаю нужные файлы и вставляю в .asm файл текст из листинга (рис. 4.1).(рис. 4.2).

A screenshot of a terminal window showing four commands being executed. The first command creates a directory named 'lab09'. The second command changes the current directory to 'lab09'. The third command creates a file named 'lab09-1.asm'. The fourth command shows the current directory path.

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ mkdir lab09
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cd lab09
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ touch lab09-1.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$
```

Рис. 4.1: Создание файлов

```
lab09-1.asm [----] 0 L: [ 1+ 0 1/ 36] *(0 / 708b) 0037 0x025 [*] [X]
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

1 Помощь 2 Сохран 3 Блок 4 Замена 5 Копия 6 Пере-ить 7 Поиск 8 Удалить 9 МенюМС 10 Выход
```

Рис. 4.2: Листинг 9.1

Проверяю его на работу и создаю исполняемый файл (рис. 4.3).

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$
```

Рис. 4.3: Исполняемый файл Листинга 9.1

Изменяю текст файла Листинга (рис. 4.4), чтобы он имел выводил сложную функцию (рис. 4.5).

```

lab09-1.asm      [----]  0 L:[ 20+ 7  27/ 57] *(437 / 876b) 0099 0x063      [*][X]
mov eax, m2
call sprintLF
mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _calcul ; Вызов подпрограммы _calcul

mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:

call _subcalcul

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

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
ret

```

Рис. 4.4: Измененный Листинг

```

savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf lab09-1.asm
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./lab09-1
f(x) = 2x +7
g(x) = 3x-1
Введите x: 5
2x+7=35
savostinoleg@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$

```

Рис. 4.5: Запуск исполняемого файла

4.2 Отладка программ с помощью GDB

Создаю файл и вставляю в него Листинг 9.2(рис. 4.6).

```
Архитектура компьютера/arch-pc/lab09$ touch lab09-2.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ cat lab09-2.asm
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: Листинг 9.2

Создаю исполняемый файл и .lst (рис. 4.7) и запускаю gdb (рис. 4.8). Ставлю брекпоинт на _start.

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb lab09-2
```

Рис. 4.7: Создание исполняемых файлов

```

savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/savostinoleg/work/study/2024-2025/Архитектура компьютера/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 5974) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/savostinoleg/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/lab09-2

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

```

Рис. 4.8: Использование gdb

Начиная с метки, просматриваю дисассимплированный код (рис. 4.9) и переключаюсь на intel'овское отображение синтаксиса. Отличие заключается в командах, в дисассимплированном отображении в командах используют % \$, а в Intel эти символы не отображены, тем самым так удобнее.

```

(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.
(gdb)

```

Рис. 4.9: Disassembly Intel

Используя layout asm (рис. 4.10) layout regs (рис. 4.11).

```

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
0x804902c <_start+44>   mov     eax,0x1
0x8049031 <_start+49>   mov     ebx,0x0
0x8049036 <_start+54>   int      0x80
0x8049038             add     BYTE PTR [eax],al
0x804903a             add     BYTE PTR [eax],al
0x804903c             add     BYTE PTR [eax],al
0x804903e             add     BYTE PTR [eax],al
0x8049040             add     BYTE PTR [eax],al
0x8049042             add     BYTE PTR [eax],al
0x8049044             add     BYTE PTR [eax],al
0x8049046             add     BYTE PTR [eax],al
0x8049048             add     BYTE PTR [eax],al
0x804904a             add     BYTE PTR [eax],al
0x804904c             add     BYTE PTR [eax],al
native process 6010 In: _start                                L9    PC: 0x8049000

```

Рис. 4.10: layout asm

```

[ 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
0x804902c <_start+44>   mov     eax,0x1
0x8049031 <_start+49>   mov     ebx,0x0
native process 6010 In: _start                                L9    PC: 0x8049000
(gdb) layout regs
(gdb)

```

Рис. 4.11: layout regs

Проверяю на наличие меток (рис. 4.12). Добавляю одну метку(рис. 4.13).


```
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
0x804902c <_start+44>     mov     eax,0x1

native process 6010 In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint       keep y  0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
```

Рис. 4.12: Проверка на наличие меток

```
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 6715 In: _start                                L9      PC: 0x8049000
(gdb) layout regs
(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
breakpoint already hit 1 time
2        breakpoint       keep y  0x08049031 lab09-2.asm:20
(gdb) █
```

Рис. 4.13: Создание метки

С помощью команды si я посмотрел регистры (рис. 4.14).

```
Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcf80  0xffffcf80
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005  0x8049005 <_start+5>
eflags   0x10202   [ IF RF ]

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 6715 In: _start L10 PC: 0x8049005
ebx      0x0      0
esp      0xffffcf80  0xffffcf80
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000  0x8049000 <_start>
eflags   0x202     [ IF ]
cs       0x23      35
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
(gdb) si
(gdb) █
```

Рис. 4.14: Регистры si

Просматриваю значения msg1 msg2 и изменяю их(рис. 4.15).

```

(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:      "world!\n\034"
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "Lor d!\n\034"
(gdb) █

```

Рис. 4.15: Значения и изменения msg1 msg2

Изменяю значение регистра ebx. Выводятся два разных значения так как в первом случае вводится значение 2, а во второй раз регистр равен 2. (рис. 4.16).

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$2 = 2
(gdb) █

```

Рис. 4.16: Изменения значения регистра

Проверка значений регистров (рис. 4.17).

```
(gdb) p/s $eax
$4 = 4
(gdb) p/t $eax
$5 = 100
(gdb) p/s $ecx
$6 = 0
(gdb) p/x $ecx
$7 = 0x0
(gdb)
```

Рис. 4.17: Значения регистров

Выхожу из программы(рис. 4.18).

```
(gdb) quit
```

Рис. 4.18: Выход

Теперь, я скопировал файл lab8-2.asm и переименовал его в lab09-3.asm. Добавляю метку и запускаю файл(рис. 4.19).

```

savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb --args
lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 14.2-1.fc40
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) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 6.
(gdb) run
Starting program: /home/savostinoleg/work/study/2024-2025/Архитектура компьютера/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:6
6       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffcf40: 0x00000005
(gdb)

```

Рис. 4.19: Файл из 8 Лабораторной работы

Проверяю адрес вершины стека. Там хранится 5 элементов. По первому адресу находится адрес а в остальных хранятся элементы. Элементы расположены с интервалом в 4 единицы. Стек может хранить до 4 байт. Компьютер использует новый стек для новой информации чтобы данные сохранялись нормально и без помех (рис. 4.20).


```

code.asm  [----]  0 L:[ 7+37 44/ 44] *(393 / 393b) <EOF>  [*][X]
_start:

pop ecx

pop edx.

sub ecx,1.

mov esi,0

mov eax,funct
call sprintLF

next:
cmp ecx,0h ;
jz _end.

pop eax.
call atoi
call sub
add esi,eax

loop next

_end:
mov eax, msg.
call sprint
mov eax, esi.
call iprintLF.
call quit.

sub:
mov ebx,12
mul ebx
sub eax,7
ret

```

Рис. 4.21: Измененный код

Проверяю на правильность работы (рис. 4.22).

```

(gdb) run 1 2 3 4
Starting program: /home/savostinoleg/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/code
1 2 3 4

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.
f(x)=12x-7
Результат: 92
[Inferior 1 (process 6584) exited normally]
(gdb) run 1
Starting program: /home/savostinoleg/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/code
1
f(x)=12x-7
Результат: 5
[Inferior 1 (process 6597) exited normally]
(gdb)

```

Рис. 4.22: Проверка на правильность

2. Создаю файл, и вставляю в него код из лабораторной работы. Проверяю на

правильность. Выходит ошибка, так как $(3+2)*4+5$ не равно 10. (рис. 4.23).

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ nasm -f elf -g -l code2.lst code2.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ld -m elf_i386 -o code2 code2.o
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./code2
bash: ./code2: Нет такого файла или каталога
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ ./code2
Результат: 10
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$
```

Рис. 4.23: Проверка на правильность файла из Листинга 9.3

Запускаю gdb, ошибка арифметическая (рис. 4.24). Анализирую код(рис. 4.25).

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09$ gdb code2
GNU gdb (Fedora Linux) 14.2-1.fc40
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 code2...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file code2.asm, line 11.
(gdb) r
Starting program: /home/savostinoleg/work/study/2024-2025/Архитектура компьютера/arch-pc/lab09/code2

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 code2.asm:11
11      mov ebx,3
(gdb) set disassembly-flavor intel
```

Рис. 4.24: Запуск gdb

| | | | | | |
|-------------------------|------------|----------------------|--------|---------|-----------|
| Register Group: General | | | | | |
| eax | 0x0 | 0 | ecx | 0x0 | 0 |
| edx | 0x0 | 0 | ebx | 0x3 | 3 |
| esp | 0xffffcf80 | 0xffffcf80 | ebp | 0x0 | 0x0 |
| esi | 0x0 | 0 | edi | 0x0 | 0 |
| eip | 0x80490ed | 0x80490ed <_start+5> | eflags | 0x10202 | [IF RF] |
| cs | 0x23 | 35 | ss | 0x2b | 43 |
| ds | 0x2b | 43 | es | 0x2b | 43 |
| fs | 0x0 | 0 | gs | 0x0 | 0 |


```

B+ 0x80490e8 <_start>      mov     ebx,0x2
>0x80490ed <_start+5>     mov     eax,0x2
0x80490f2 <_start+10>     add     ebx,eax
0x80490f4 <_start+12>     mov     ecx,0x4
0x80490f9 <_start+17>     mul     ecx
0x80490fb <_start+19>     add     ebx,0x5
0x80490fe <_start+22>     mov     edi,ebx
0x8049100 <_start+24>     mov     eax,0x804a000
0x8049105 <_start+29>     call    0x804900f <sprint>
0x804910a <_start+34>     mov     eax,edi
0x804910c <_start+36>     call    0x8049086 <iprintLF>

```

Рис. 4.25: Регистры.

Изменяю текст кода (рис. 4.26). Исправленный код(рис. 4.27).

```

code2.asm      [----]  0 L:[ 1+ 0  1/ 25] *(0  / 353b) 0037 [*]
%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 iprintLF
call quit

```

Рис. 4.26: Ошибочный код

```
code2.asm [----] 0 L:[ 1+ 0 1/ 25] *(0 / 353b) 0037 [*]
%include 'in_out.asm'

SECTION .data
div: DB 'Результат: ',0

SECTION .text
GLOBAL _start
_start:

; ---- Вычисление выражения (3+2)*4+5
mov eax,3
mov ebx,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax

; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 4.27: Исправленный код

Проверяю на правильность. Все верно(рис. 4.28).

```
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/1
ab09$ nasm -f elf -g -l code2.lst code2.asm
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/1
ab09$ ld -m elf_i386 -o code2 code2.o
savostinoleg@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/1
ab09$ ./code2
Результат: 25
```

Рис. 4.28: Результат исправления кода

5 Выводы

В ходе работы, я приобрел навыки написания программ с использованием подпрограмм. Ознакомился с методами отладки при помощи GDB и его основными возможностями

Список литературы

Лабораторная работа №9