

Лабораторная работа №9.

Понятие подпрограммы. Отладчик GDB

Павленко Сергей

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	8
4	Выводы	21
5	Самостоятельная работа	22
6	Выводы по самостоятельной работе	24
	Список литературы	25

Список иллюстраций

[illegible]

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм.
Знакомство с методами отладки при помощи GDB и его основными возможностями

2 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа: * обнаружение ошибки * поиск её местонахождения * определение причины ошибки * исправление ошибки. Можно выделить следующие типы ошибок: * синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка; * семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата * ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга. Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы. Последний этап — исправление ошибки. После этого при повторном запуске программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

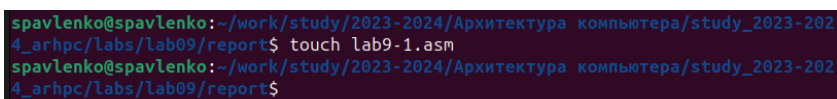
Наиболее часто применяют следующие методы отладки: * создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения); * использование специальных программ-отладчиков. Отладчики

позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. Пошаговое выполнение — это выполнение программы с остановкой после каждой строки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова: * Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом); * Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его). Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы.

3 Выполнение лабораторной работы

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm:

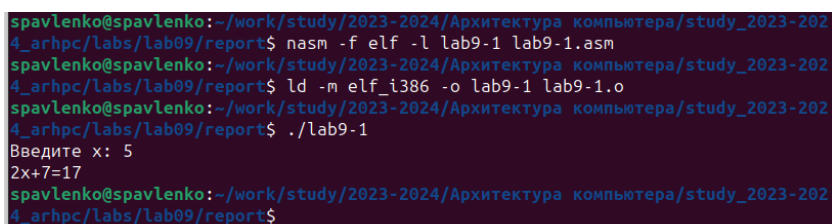
`cd ~/work/arch-pc/lab09 touch lab09-1.asm`



```
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$ touch lab9-1.asm
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$
```

Рис. 3.1: 1

2. В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы `_calcul`. В данном примере `x` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 9.1) Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу



```
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$ nasm -f elf -l lab9-1 lab9-1.asm
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$ ld -m elf_i386 -o lab9-1 lab9-1.o
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$ ./lab9-1
Введите x: 5
2x+7=17
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$
```

Рис. 3.2: 2

3. Измените текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $g(x)$, результат возвращается в `_calcul` и вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран.

```
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ nasm -f elf -l lab9-1 lab9-1.asm
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ ld -m elf_i386 -o lab9-1 lab9-1.o
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ ./lab9-1
Введите x: 5
2x+7=77
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$
```

Рис. 3.3: 3

4. Создайте файл `lab09-2.asm` с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!): Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом `'-g'`. `nasm -f elf -g -l lab09-2.lst lab09-2.asm ld -m elf_i386 -o lab09-2 lab09-2.o` Проверьте работу программы, запустив ее в оболочке GDB с помощью команды `run` (со- кращённо `r`): `(gdb) run`

```

spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ ld -m elf_i386 -o lab9-2 lab9-2.o
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ ./lab9-2
Hello, world!
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ gdb lab9-2
GNU gdb (Ubuntu 14.0.50.20230907-0ubuntu1) 14.0.50.20230907-git
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-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 lab9-2...
(gdb) run
Starting program: /home/spavlenko/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
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 3714) exited normally]
(gdb)

```

Рис. 3.4: 4

5. Для более подробного анализа программы установите брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустите её. (gdb) `break _start` Breakpoint 1 at 0x8049000: file lab09-2.asm, line 12. (gdb) `run` Starting program: `~/work/arch-pc/lab09/lab09-2` Breakpoint 1, `_start ()` at lab09-2.asm:12 `12 mov eax, 4` Посмотрите дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (gdb) `disassemble _start` Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (gdb) `set disassembly-flavor intel` (gdb) `disassemble _start`

```

(gdb) break _start
Breakpoint 1 at 0x08049000: file lab9-2.asm, line 9.
(gdb) run
Starting program: /home/spavlenko/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
9      mov eax, 4
(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. 3.5: 5

6. Перечислите различия отображения синтаксиса машинных команд в режимах АТТ и Intel. Включите режим псевдографики для более удобного анализа программы (рис. 9.2): (gdb) layout asm (gdb) layout regs В этом режиме есть три окна:

- В верхней части видны названия регистров и их текущие значения;
- В средней части виден результат дисассимилирования программы;
- Нижняя часть доступна для ввода команд.

```
[ 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
0x8049036 <_start+54>   int     0x80

native process 3776 In: _start          L9      PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 3.6: 6

7. На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверьте это с помощью команды `info breakpoints` (кратко `i b`): `(gdb) info breakpoints` Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3). Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова. `(gdb) break` Посмотрите информацию о всех установленных точках останова Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции (см. рис. 9.3). Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова. `(gdb) break` Посмотрите информацию

о всех установленных точках останова: (gdb) i b

```
Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcfb0 0xffffcfb0
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

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
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80
0x8049038      add     BYTE PTR [eax],al

native process 3945 In: _start L9 PC: 0x8049000
1 breakpoint keep y 0x00049000 lab9-2.asm:9
  breakpoint already hit 1 time
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/spavlenko/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
(gdb) break *0x8049031
No symbol "0x8049031" in current context.
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb)
```

Рис. 3.7: 7

```

Register group: general
eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffcfb0 0xffffcfb0
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

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
b+ 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54>  int     0x80
0x8049038          add     BYTE PTR [eax],al

native process 3945 In: _start L9 PC: 0x8049000
Start it from the beginning? (y or n) y
Starting program: /home/spavlenko/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report/lab9-2

Breakpoint 1, _start () at lab9-2.asm:9
(gdb) break *0x8049031
No symbol "0x8049031" in current context.
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab9-2.asm, line 20.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab9-2.asm:9
      breakpoint already hit 1 time
2     breakpoint       keep y  0x08049031 lab9-2.asm:20
(gdb)

```

Рис. 3.8: 8

8. Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. Значения каких регистров изменяются? Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`). `(gdb) info registers`

```

ecx      0x804a008      134520840
edx      0x7           7
esp      0xffffcfb0     0xffffcfb0
ebp      0x0           0x0
esi      0x0           0
edi      0x0           0
eip      0x804902a      0x804902a <_start+42>
eflags   0x102031      [ IF RF31 9
eflags   0x10202       [ IF RF ]
ss       0x2b          43
ds       0x2b          43

B+ 0x8049000 <_start>    mov     eax,0x4
B+ 0x8049031 <_start+49> mov     ebx,0x004a008
0x8049036 <_start+54>    int     0x80
>
b+

native process 3945 In: _start L18 PC: 0x804902a
Breakpoint 1, lab9-2.asm, line 20. L?? PC: ??
stepl, si
Step one instruction exactly.
Usage: stepl [N]
Argument N means step N times (or till program stops for another reason).
(gdb) stepl 2
(gdb) stepl 3
(gdb) stepl 4
(gdb) stepl 5
world!

Breakpoint 2, _start () at lab9-2.asm:20
(gdb) stepl 6
[Inferior 1 (process 3945) exited normally]
(gdb) i r
The program has no registers now.
(gdb)

```

Рис. 3.9: 9

9. С помощью команды `x/1sb &msg1` также можно посмотреть содержимое переменной. Посмотрите значение переменной `msg1` по имени (gdb) `x/1sb &msg1 0x804a000` : “Hello,” Посмотрите значение переменной `msg2` по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию `mov esx,msg2` которая записывает в регистр `esx` адрес переменной `msg2`

```

[ 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 3267 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "world!\n\034"
(gdb) 

```

Рис. 3.10: 10

10. Изменить значение для регистра или ячейки памяти можно с помощью команды `set`, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс `$`, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си). Измените первый символ переменной `msg1`: (gdb) `set {char}msg1='h'` (gdb) `x/1sb &msg1`
0x804a000 : "hello," (gdb)

```

native process 3267 In: _start L9 PC: 0x8049000
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "Hello, "
(gdb) set {char}msg1='h'
'msg1' has unknown type; cast it to its declared type
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) 

```

Рис. 3.11: 11

11. Замените любой символ во второй переменной `msg2`.


```

native process 3267 In: _start L9 PC: 0x8049000
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}&msg2='Lor d!\n'
No symbol "Lor d!\n" in current context.
(gdb) set {char}&msg2='L'
(gdb) p/F $eax
No symbol "F" in current context.
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Lorld!\n\034"
(gdb)

```

Рис. 3.12: 12

12. Чтобы посмотреть значения регистров используется команда print /F (перед именем регистра обязательно ставится префикс \$) p/F \$

```

native process 3267 In: _start L9 PC: 0x8049000
0x804a008 <msg2>: "Lorld!\n\034"
(gdb) p/s $eax
$1 = 0
(gdb) p/t $eax
$2 = 0
(gdb) p/s $ecx
$3 = 0
(gdb) p/x $ecx
$4 = 0x0
(gdb)

```

Рис. 3.13: 13

13. С помощью команды set измените значение регистра ebx: (gdb) set \$ebx='2' (gdb) p/s \$ebx \$3 = 50 (gdb) set \$ebx=2 (gdb) p/s \$ebx \$4 = 2 (gdb)

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb)

```

Рис. 3.14: 14

14. Объясните разницу вывода команд p/s \$ebx. Обе команды делают одно и тоже - устанавливают значение переменной равным 2, однако в первом случае значение указывается внутри кавычек, что может быть полезно, если нужно установить значение переменной, которое содержит пробелы

или специальные символы. Завершите выполнение программы с помощью команды `continue` (сокращенно `c`) или `stepi` (сокращенно `si`) и выйдите из GDB с помощью команды `quit` (сокращенно `q`)

```
(gdb) continue
Continuing.
hello, World!
[Inferior 1 (process 3267) exited normally]
(gdb)
```

Рис. 3.15: 15

```
0x08049036 <+54>: int 0x80
End of assembler dump.
(gdb) layout asm
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$
```

Рис. 3.16: 16

15. Скопируйте 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`: `cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm` Создайте исполняемый файл. `nasm -f elf -g -l lab09-3.lst lab09-3.asm ld -m elf_i386 -o lab09-3 lab09-3.o` Для загрузки в `gdb` программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы: `gdb -args lab09-3 аргумент1 аргумент 2 'аргумент 3'`

```

spavlenko@spavlenko:~$ cp ~/work/study/2023-2024/Архитектура\ компьютера/study_2
023-2024_arhpc/labs/lab08/report/lab8-2.asm ~/work/study/2023-2024/Архитектура\
компьютера/study_2023-2024_arhpc/labs/lab09/report/lab9-3.asm
spavlenko@spavlenko:~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/study_2
023-2024_arhpc/labs/lab09/report/
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-202
4_arhpc/labs/lab09/report$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
nasm: fatal: unable to open input file `lab09-3.asm' No such file or directory
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-202
4_arhpc/labs/lab09/report$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-202
4_arhpc/labs/lab09/report$ ld -m elf_i386 -o lab9-3 lab9-3.o
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-202
4_arhpc/labs/lab09/report$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 14.0.50.20230907-0ubuntu1) 14.0.50.20230907-git
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-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 lab9-3...
(gdb)

```

Рис. 3.17: 17

16. Для начала установим точку останова перед первой инструкцией в программе и запустим ее. (gdb) b _start (gdb) run Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): (gdb) x/x \$esp 0xffffd200: 0x05

```

Reading symbols from lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) run
Starting program: /home/spavlenko/work/study/2023-2024/Архитектура компьютера/s
udy_2023-2024_arhpc/labs/lab09/report/lab9-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
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 lab9-3.asm:5
5      pop ecx
(gdb)

```

Рис. 3.18: 18

17. Посмотрите остальные позиции стека – по адресу [esp+4] располагается

адрес в памяти где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д. 110 Демидова А. В. Архитектура ЭВМ (gdb) x/s *(void**)(esp + 4)0xffffd358 : "/lab09 – 3"(gdb)x/s *(void **)(esp + 8) 0xffffd3bc: “аргумент1” (gdb) x/s *(void**)(esp+12)0xffffd3ce : ""(gdb)x/s*(void**)(esp + 16) 0xffffd3df: “2” (gdb) x/s *(void**)(esp + 20)0xffffd3e1 : "3"(gdb)x/s *(void **)(esp + 24) 0x0: <error: Cannot access memory at address 0x0> (gdb)

```

5      pop ecx
(gdb) x/s *(void**)(esp + 4)
0xffffd11c: "/home/spavlenko/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report/lab9-3"
(gdb) x/s *(void**)(esp + 8)
0xffffd19c: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xffffd1ae: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xffffd1bf: "2"
(gdb) x/s *(void**)(esp + 20)
0xffffd1c1: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.19: 19

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.). Это может зависеть от разрядности процессора, архитектуры ОС и других факторов. Однако в x86 архитектуре процессоры используют систему адресации, которая предполагает, что каждый байт в памяти имеет адрес, кратный 4.

4 Выводы

Таким образом мы приобрели навыки написания программ с использованием подпрограмм. Познакомились с методами отладки при помощи GDB и его основными возможностями

5 Самостоятельная работа

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.

```
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$ nasm -f elf -g -l lab9-4.lst lab9-4.asm
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$ ld -m elf_i386 -o lab9-4 lab9-4.o
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$ ./lab9-4
Результат: 0
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$ ./lab9-4 1 2 3 4
Результат: 28
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_4_arhpc/labs/lab09/report$
```

Рис. 5.1: 20

2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) * 4 + 5$. При запуске данная программа дает неверный результат.

```

4_arhpc/labs/lab09/report$ gdb lab9-5
GNU gdb (Ubuntu 14.0.50.20230907-0ubuntu1) 14.0.50.20230907-git
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-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 lab9-5...
(gdb) run
Starting program: /home/spavlenko/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report/lab9-5

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
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
Результат: 10

```

Рис. 5.2: 21

Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.

```

spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ nasm -f elf -g -l lab9-5.lst lab9-5.asm
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ ld -m elf_i386 -o lab9-5 lab9-5.o
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$ ./lab9-5
Результат: 25
spavlenko@spavlenko:~/work/study/2023-2024/Архитектура компьютера/study_2023-2024_arhpc/labs/lab09/report$

```

Рис. 5.3: 22

6 Выводы по самостоятельной работе

Таким образом в ходе самостоятельной работы мы закрепили знания по данной теме на практике

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-

- е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).