Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров и операционные системы

Авдадаев Джамал Геланиевич

Содержание

1	Цель работы		
2	Задание		1
3	Теоретическое введение		
4	Выполнение лабораторной работы		
	4.1 Реализация подпрограмм в NASM		
	4.2 Отладка программам с помощью GDB		
	4.2.1	Добавление точек останова	
	4.2.2	Работа с данными программы в GDB	10
	4.2.3	Обработка аргументов командной строки в GDB	15
	4.3 3a	дания для самостоятельной работы	16
5	Выводы		21
6	Список литературы		2.1

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 программам 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 у (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 sI) 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 еір 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 еір. 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. 32)

```
тура компьютера/study_2023-2024_arh-pc$ mkdir lab09
тура компьютера/study_2023-2024_arh-pc$ cd lab09
тура кdgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура кdgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура кdgav vdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура компьютера/s
тура компьютера/study_2023-2024_arh-pc/lab09$ touch lab09-1.asm
тура компьютера/study_2023-2024_arh-pc/lab09$
```

Figure 1: Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1. (рис. 32)

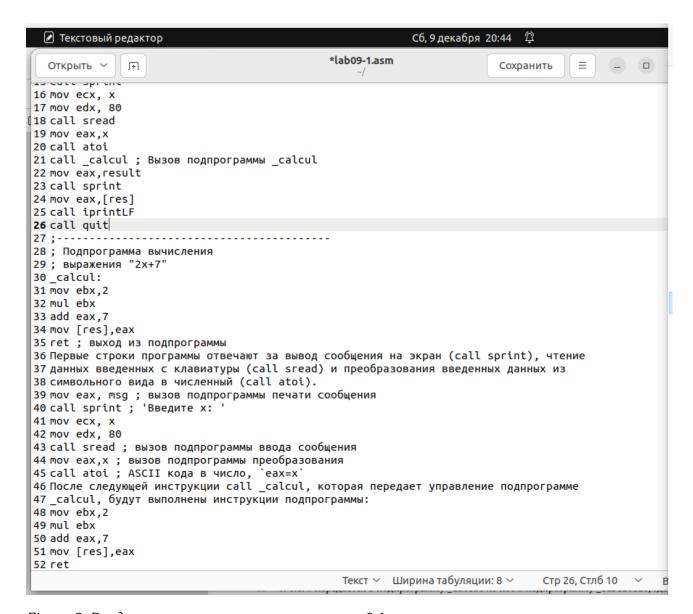


Figure 2: Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу. (рис. 32)

```
[dgavdadaev@fedora lab09]$ nasm -f elf lab09-1.asm
[dgavdadaev@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[dgavdadaev@fedora lab09]$ ./lab09-1
Введите х: 7
2x+7=47
```

Figure 3: Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul для вычисления выражения f(g(x)), где x вводится c клавиатуры, f(x) = 2x + 7, g(x) = 3x - 1. (рис. 32)

```
⊞
                    mc [dgavdadaev@fedora]:~/work/arch-pc/lab09
lab09-1.asm
                   [---] 0 L:[ 21+ 4 25/ 43] *(468 / 727b)
call _subcalcul ; Вызов подпрограммы _calcul
call _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
calcul:
mul ebx
add eax,7
mov [res],eax
ret
_subcalcul:
mov ebx,3
mul ebx
add eax,-1
ret
```

Figure 4: Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу. (рис. 32)

```
[dgavdadaev@fedora lab09]$ nasm -f elf lab09-1.asm
[dgavdadaev@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[dgavdadaev@fedora lab09]$ ./lab09-1
Введите х: 7
2х+7=47
```

Figure 5: Запуск исполняемого файла

4.2 Отладка программам с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (рис. 32)

```
⊞
                    mc [dgavdadaev@fedora]:~/work/arch-pc/lab09
                           0 L:[ 1+ 0
                                         1/ 22] *(0
lab09-2.asm
                                                       / 294b)
SECTION .data
msg1: db "Hello, ",0x0
msglLen: equ $ - msgl
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
start:
mov eax, 4
mov ebx, 1
mov ecx, msgl
mov edx, msglLen
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Figure 6: Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом '-g'. (рис. 32)

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

Figure 7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик gdb. (рис. 32)

```
[dgavdadaev@fedora 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 <a href="http://gnu.org/licenses/gpl.html">http://gnu.org/licenses/gpl.html</a>
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:
<a href="https://www.gnu.org/software/gdb/bugs/">https://www.gnu.org/software/gdb/bugs/</a>
Find the GDB manual and other documentation resources online at:
<a href="https://www.gnu.org/software/gdb/documentation/">https://www.gnu.org/software/gdb/documentation/</a>
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
```

Figure 8: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run. (рис. 32)



Figure 9: Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start и запускаю её. (рис. 32)

```
(gdb) break _start

Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.

(gdb) run

Starting program: /home/eapostnova/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9

__mov_eax, 4
```

Figure 10: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды disassemble, начиная с метки _start, и переключаюсь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel. (рис. 32)

Figure 11: Использование команд disassemble и disassembly-flavor intel

В режиме ATT 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. 32)

```
[ Register Values Unavailable ]

0x8049126 add BYTE PTR [eax],al
0x8049128 add BYTE PTR [eax],al
0x804912c add BYTE PTR [eax],al
0x804912c add BYTE PTR [eax],al
0x804912c add BYTE PTR [eax],al
0x804912a add BYTE PTR [eax],al
0x8049130 add BYTE PTR [eax],al
0x8049132 add BYTE PTR [eax],al
0x8049134 add BYTE PTR [eax],al
0x8049134 add BYTE PTR [eax],al
0x8049134 add BYTE PTR [eax],al
```

Figure 12: Включение режима псевдографики

4.2.1 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. 32)

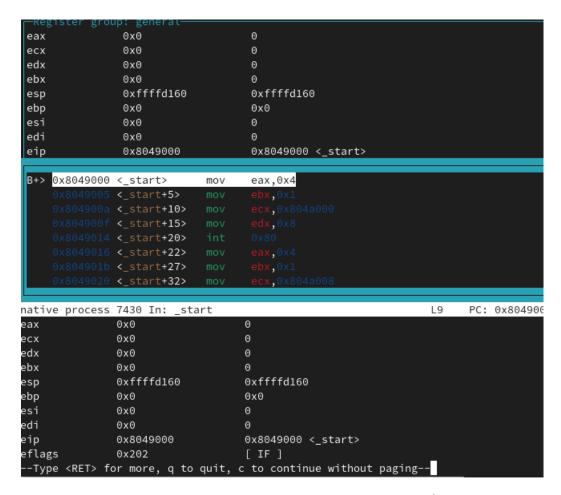


Figure 13: Установление точек останова и просмотр информации о них

4.2.2 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды stepi и слежу за изменением значений регистров. (рис. 32)

```
[ Register Values Unavailable ]
B+> 0x8049000 <_start>
                                     eax,0x4
                             mov
        049005 <_start+5>
     0x804900f <_start+15>
0x8049014 <_start+20>
                                                                               PC: 0x8049000
native process 6819 In: _start
                                                                         L9
(gdb) i b
                       Disp Enb Address
Num
                                            What
        Type Disp Enb Address What 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
                       Disp Enb Address
Num
        Туре
                                            What
        breakpoint keep y 0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
                       keep y 0x08049031 lab09-2.asm:20
        breakpoint
(gdb)
```

Figure 14: До использования команды stepi

(рис. 32)

```
0x8
                                       134520832
                 0x804a000
 есх
 edx
                 0x8
                                       8
 ebx
                 0x1
                                       0xffffd160
                 0xffffd160
 ebp
                 0x0
                                       0x0
 esi
                 0x0
 edi
                 0x0
                                       0
                                       0x8049016 <_start+22>
eip
                 0x8049016
                                       eax,0x4
ebx,0x1
ecx,0x804a000
     0x8049016 <_start+22>
                                moν
                                       eax,0x4
                <_start+27>
                <_start+32>
native process 7430 In: _start
                                                                            PC: 0x8049016
                0x8049000
                                      0x8049000 <_start>
eip
eflags
                0x202
                                      [ IF ]
--Type <RET> for more, q to quit, c to continue without paging--ccs
                                                                                        0x23
                                      43
                0x2b
                                      43
ds
                0x2b
                0x2b
es
fs
                0x0
                                      0
                0x0
gs
(gdb) si 5
```

Figure 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. 32)

```
eax
                0x8
                                     8
 есх
                0x804a000
                                     134520832
edx
                0x8
                                     8
                0x1
 ebx
                                     1
                0xffffd160
                                     0xffffd160
ebp
                0x0
                                     0x0
                0x0
esi
edi
                0x0
                0x8049016
                                     0x8049016 <_start+22>
eip
В+
     0x8049005 <_start+5>
      x804900f <_start+15>
     0x8049016 <_start+22>
                                     eax,0x4
                              mov
native process 7430 In: _start
                                                                   L14
                                                                         PC: 0x8049016
                35
               0x2b
ds
               0x2b
es
               0x2b
                                    43
               0x0
               0x0
                                    0
gs
(gdb) x/1sb &msgl
                         "Hello, "
(gdb) x/1sb 0x804a008
                         "world!\n\034"
(gdb)
```

Figure 16: Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2. (рис. 32)

```
0x8
                0x804a000
                                      134520832
есх
 edx
 ebx
                0xffffd160
                                      0xffffd160
esp
ebp
                0x0
                                      0x0
     0x804900a <<u>start+10></u>
       (8049014 <_start+20>
    0x8049016 <_start+22>
                                      eax,0x4
                              moν
          901b <_start+27>
         49020 <_start+32>
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)
```

Figure 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. 32)

```
eax
                 0x8
                 0x804a000
                                      134520832
 есх
 edx
                 0x8
                                      8
 ebx
                 0x1
                                      1
                 0xffffd160
                                      0xffffd160
 esp
                 0x0
                                      0x0
 ebp
                                      Θ
                 0x0
 esi
                                      Θ
 edi
                 0x0
 eip
                 0x8049016
                                      0x8049016 <_start+22>
eflags
                0x202
                                      [ IF ]
     0x8049005 < start+5>
     0x804900a <_start+10>
     0x804900f <_start+15>
     0x8049016 <_start+22>
                                      eax,0x4
     0x8049020 <_start+32>
               <_start+42>
native process 7430 In: _start
                                                                          PC: 0x8049016
(gdb) p/x $edx
$8 = 0x8
(gdb) p/t $edx
$9 = 1000
(gdb) p/c $edx
$10 = 8 '\b'
```

Figure 18: Вывод значения регистра в разных представлениях

С помощью команды set изменяю значение регистра ebx в соответствии с заданием. (рис. 32)

```
(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"
```

Figure 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. 32)

```
0x1
 eax
                                      1
                                      134520840
 есх
                0x804a008
 edx
                                      7
                0x7
 ebx
                0x1
                                      0xffffd160
                 0xffffd160
 ebp
                0x0
                                      0x0
 esi
                0x0
 edi
 eip
                0x8049031
                                      0x8049031 <_start+49>
                                      ecx,0x804a008
edx,0x7
      0x804902c <_start+44>
B+> 0x8049031 <_start+49>
                                      ebx,0x0
                              mov
     0x8049036 <_start+54>
                                      BYTE PTR [eax],al
                                      BYTE PTR [eax],
native process 7430 In: _start
                                                                     L20
                                                                            PC: 0x8049031
$14 = 2
(gdb) c
Continuing.
borld!
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)
```

Figure 20: Завершение работы GDB

4.2.3 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. 32)

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргументы с использованием ключа –args. (рис. 32)

```
[dgavdadaev@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'aргумент 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 <a href="http://gnu.org/licenses/gpl.html">http://gnu.org/licenses/gpl.html</a>

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:

<a href="https://www.gnu.org/software/gdb/bugs/">https://www.gnu.org/software/gdb/bugs/</a>

Find the GDB manual and other documentation resources online at:

<a href="https://www.gnu.org/software/gdb/documentation/">https://www.gnu.org/software/gdb/documentation/</a>

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from lab09-3...

(gdb)
```

Figure 21: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю ее. (рис. 32)

Посматриваю вершину стека и позиции стека по их адресам. (рис. 32)

Figure 22: Просмотр значений, введенных в стек

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

4.3 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. 32)

```
(gdb) x/x $esp

0xffffdl20: 0x00000005
(gdb) x/s *(void**)($esp + 4)

0xffffdl20: "/home/eapostnova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)

0xffffdl20: "аргумент!"
(gdb) x/s *(void**)($esp + 12)

0xffffdl20: "аргумент"
(gdb) x/s *(void**)($esp + 16)

0xffffdl20: "2"
(gdb) x/s *(void**)($esp + 20)

0xffffdl20: "аргумент 3"
(gdb) x/s *(void**)($esp + 20)

0xffffdl20: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)

0x0: <error: Cannot access memory at address 0x0>
```

```
Figure 23: Написание кода подпрограммы
Код программы:
%include 'in_out.asm'
SECTION .data
msg db "Результат:",0
SECTION .text
global_start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,5
call .next
.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
```

2. Ввожу в файл task1.asm текст программы из листинга 9.3. (рис. 32)

```
task1.asm [----] 0 L:[ 14+ 7 21/ 31] *(229 / 323b) 0106 0x06A

.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
```

Figure 24: Ввод текста программы из листинга 9.3

При корректной работе программы должно выводится "25". Создаю исполняемый файл и запускаю его. (рис. 32)

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 есх происходит умножение есх на еах, 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,eax не связана с mul ecx, но связана инструкция mov eax,2. (рис. 32)

Figure 25: Нахождение причины ошибки

Из-за этого мы получаем неправильный ответ. (рис. 32)

Figure 26: Неверное изменение регистра

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. 32)

```
task2.asm [-M--] 11 L:[ 1+14 15/ 22] *(245 / 361b) 0010 0х00A %include 'in_out.asm'
SECTION .data
div: DB 'Peзультат: ',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
```

Figure 27: 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): Ошибка исправлена

```
Код программы:
%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

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.За непредоставленные скриншоты если они нужны будут прошу меня простить, не смог их добавить

6 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 c. (In a Nutshell). ISBN 0596009658. URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.
- 6. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 c. ISBN 978-1491941591.
- 7. The NASM documentation. 2021. URL: https://www.nasm.us/docs.php.
- 8. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 c. 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, 2017.
- 11. Новожилов О. П. Архитектура ЭВМ и систем. М.: Юрайт, 2016.
- 12. Расширенный ассемблер: NASM. 2021. URL: https://www.opennet.ru/docs/RUS/nasm/.
- 13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. 2-е изд. БХВПетербург, 2010. 656 с. ISBN 978-5-94157-538-1.
- 14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. 2-е изд. М.: MAKC Пресс, 2011. URL: http://www.stolyarov.info/books/asm_unix.
- 15. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб. : Питер, 2013. 874 с. (Классика Computer Science).

16. Таненбаум Э., Бос X. Современные операционные системы. — 4-е изд. — СПб. : Питер,2015. — 1120 с. — (Классика Computer Science).