

Лабораторная работа №9

Понятие подпрограммы

Светцова Анна Дмитриевна

Содержание

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. Задание 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 работы №9 (рис. 1).

```
adsvettsova@annpc ~$ mkdir -p /work/study/2023-2024/Computer_architecture/arch-pc/labs/lab09/lab9
```

рис.1 Создание каталога

Перехожу в созданную директорию (рис. 2).

```
adsvettsova@annpc ~$ cd /work/study/2023-2024/Computer_architecture/arch-pc/labs/lab09/lab9
```

рис.2 Перемещение по директории

Создаю файл `lab09-1.asm` в новом каталоге (рис. 3).

```
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab09/lab9 master *3 131 712 $ touch lab09-1.asm
```

рис.3 Создание файла

Открываю файл и переписываю код программы из листинга 9.1 (рис. 4).

```

1  ; lab09.asm
2 SECTION .data
3 msg DB 'Введите x: ',0
4 result DB '2x+7= ',0
5 SECTION .bss
6 x RESB 80
7 res RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;
12 ; Основная программа
13 ;
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [res]
25 call printf
26 call quit
27 ;
28 ; Подпрограмма вычисления
29 ; выражения "2x 7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [res], eax
35 ret; выход из подпрограммы
36

```

рис.4 Редактирование файла

Создаю объектный файл программы и после компоновки запускаю его (рис. 5). Код с подпрограммой работает успешно.

```

adsvettsova@annpc ~/w/s/2/C/ark-pc/l/lab09/lab9 master +3 131 713 $ nasm -f elf
lab09-1.asm
adsvettsova@annpc ~/w/s/2/C/ark-pc/l/lab09/lab9 master +3 131 713 $ ld -m elf_i
386 -o lab09-1 lab09-1.o
adsvettsova@annpc ~/w/s/2/C/ark-pc/l/lab09/lab9 master +3 131 713 $ ./lab09-1
Введите x: 5
2x+7=17

```

рис.5 Запуск программы

Изменяю текст файла, добавив подпрограмму sub_calcul в подпрограмму _calcul (рис. 6).

```

adsvettsova@annpc ~/w/s/2/C/ark-pc/l/lab09/lab9 master +3 131 713 $ nasm -f elf
lab09-1.asm
adsvettsova@annpc ~/w/s/2/C/ark-pc/l/lab09/lab9 master +3 131 713 $ ld -m elf_i
386 -o lab09-1 lab09-1.o
adsvettsova@annpc ~/w/s/2/C/ark-pc/l/lab09/lab9 master +3 131 713 $ ./lab09-1
Введите x: 5
2(3x-1)+7=35

```

рис.6 Редактирование файла

```

#include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+7= ',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
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

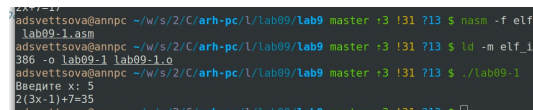
```

```

call sprint
mov eax,[res]
call iprintLF
call quit
_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

```

Запускаю исполняемый файл (рис. 7).Программа работает верно.



```

adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab09/lab9 master +3 !31 713 $ nasm -f elf
lab09-1.asm
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab09/lab9 master +3 !31 713 $ ld -m elf_i
386 -o lab09-1 lab09-1.o
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab09/lab9 master +3 !31 713 $ ./lab09-1
Введите x: 5
2(3x-1)+7=35

```

рис.7 Запуск программы

4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm, используя команду touch (рис. 8).



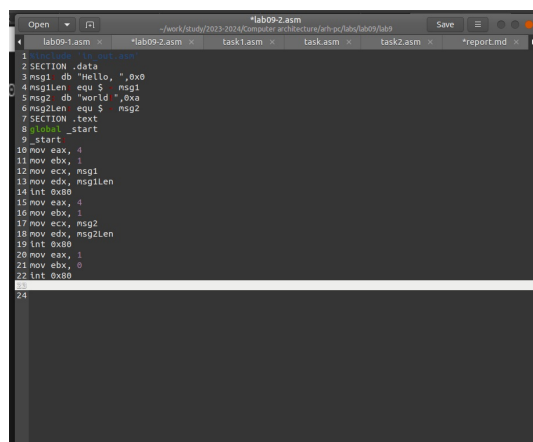
```

adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab09/lab9 master +3 !31 713 $ touch lab09-
2.asm

```

рис.8 Создание файла

Записываю код программы из листинга 9.2,который выводит сообщение Hello world (рис. 9).



```

1  ;lab09-2.asm
2 SECTION .data
3 msg1 db "Hello",0x0
4 msglen equ $ - msg1
5 msg2 db "world",0x0
6 msg2len equ $ - msg2
7 SECTION .text
8 global _start
9 _start
10 mov eax, 4
11 mov ebx, 1
12 mov ecx, msg1
13 mov edx, msglen
14 int 0x80
15 mov eax, 4
16 mov ebx, 1
17 mov ecx, msg2
18 mov edx, msg2len
19 int 0x80
20 mov eax, 1
21 mov ebx, 0
22 int 0x80
23
24

```

РИС.9 Редактирование файла

Получаю исполняемый файл. для работы с GDB провожу трансляцию программ с ключом “-g” и загружаю исполняемый файл в отладчик (рис. 10).

```
adsvettsova@annpc: ~/w/s/2/C/rrh-pc/l/lab09/lab9 master +3 131 713 $ touch lab09-2.asm
adsvettsova@annpc: ~/w/s/2/C/rrh-pc/l/lab09/lab9 master +3 131 713 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
adsvettsova@annpc: ~/w/s/2/C/rrh-pc/l/lab09/lab9 master +3 131 713 $ ld -m elf_i386 -o lab09-2 lab09-2.o
```

рис.10 Запуск исполняемого файла

Проверяю работу программы в оболочке GDB с помощью команды run (рис. 11).

```
gdb lab09-2
86 -o lab09-2 lab09-2.o
adsvettsova@annpc: ~/w/s/2/C/rrh-pc/l/lab09/lab9 master +3 131 713 $ gdb lab09-2
GNU gdb (Ubuntu 8.2-2ubuntu1-20.04.1) 8.2
Copyright (C) 2020 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:
<http://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/adsvettsova/work/study/2023-2024/Computer architecture/a
rh-pc/labs/lab09/lab9/lab09-2
Hello, world!
[Inferior 1 (process 20113) exited normally]
(gdb)
```

рис.11 Запуск программы в отладчике

Для более подробного анализа устанавливаю брейкпоинт на метку _start, с которой начинается выполнение ассемблерной программы (рис. 12).

```
[Inferior 1 (process 20113) exited normally]
(gdb) break _start
Breakpoint 1 at 0x080490e8
```

рис.12 Установка брейкпоинта

Запускаю её (рис. 13).

```
(gdb) run
Starting program: /home/adsvettsova/work/study/2023-2024/Computer architecture/a
rh-pc/labs/lab09/lab9/lab09-2
Breakpoint 1, 0x080490e8 in _start ()
(gdb)
```

рис.13 Запуск

С помощью команды “disassemble _start” просматриваю дисассимилированный код программы (рис. 14).

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x080490e8 <+0>: mov $0x4,%eax
0x080490ed <+5>: mov $0x1,%ebx
0x080490f2 <+10>: mov $0x804a000,%ecx
0x080490f7 <+15>: mov $0x8,%edx
0x080490fc <+20>: int $0x80
0x080490fe <+22>: mov $0x4,%eax
0x08049103 <+27>: mov $0x1,%ebx
0x08049108 <+32>: mov $0x804a000,%ecx
0x0804910d <+37>: mov $0x7,%edx
0x08049112 <+42>: int $0x80
0x08049114 <+44>: mov $0x1,%eax
0x08049119 <+49>: mov $0x0,%ebx
0x0804911e <+54>: int $0x80
End of assembler dump.
(gdb)
```

рис.14 Дисассимилированный код программы

Переключаюсь на отображение команд с Intel’овским синтаксисом, введя команду “set disassembly-flavor intel” (рис. 15).

```

(gdb) set disassembly-flavor intel
(gdb) disassemble start
Dump of assembler code for function start:
=> 0x080490e8 <+0>: mov eax,0x4
0x080490e9 <+5>: mov ebx,0x1
0x080490f2 <+10>: mov ecx,0x804a000
0x080490f7 <+15>: mov edx,0x8
0x080490fc <+20>: int 0x80
0x080490fe <+22>: mov eax,0x4
0x08049103 <+27>: mov ebx,0x1
0x08049106 <+32>: mov ecx,0x804a008
0x0804910d <+37>: mov edx,0x7
0x08049112 <+42>: int 0x80
0x08049114 <+44>: mov eax,0x1
0x08049119 <+49>: mov ebx,0x0
0x0804911e <+54>: int 0x80
End of assembler dump.
(gdb)

```

рис.15 Отображение с Intel'овским синтаксисом

Основное различие заключается в том, что в режиме Intel пишется сначала сама команда, а потом её машинный код, в то время как в режиме АТТ идет сначала машинный код, а только потом сама команда.

4.3 Добавление точек останова

Проверяю наличие точки останова с помощью команды info breakpoints (i b) (рис. 16).

```

(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint       keep y  0x08049031 <sprintf+4>
(gdb)

```

рис.16 Точка останова

Устанавливаю ещё одну точку останова по адресу инструкции, которую можно найти в средней части в левом столбце соответствующей инструкции (рис. 17).

```

(gdb) b *0x08049000
Breakpoint 2 at 0x08049000

```

рис.17 Установка точки останова

Просматриваю информацию о точках останова (рис. 18).

```

(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint       keep y  0x08049031 <sprintf+4>
2        breakpoint       keep y  0x08049000 <slen>
(gdb)

```

рис.18 Точки останова

4.4 Работа с данными программы в GDB

Просматриваю содержимое регистров с помощью команды info register (i r) (рис. 19).

```

native process z431 in: start
eax      0x0          0
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xffffcf90   0xffffcf90
ebp      0x0          0
esi      0x0          0
edi      0x0          0
eip      0x080490e8   0x080490e8 <_start>
eflags   0x202        [ IF ]
cs       0x23         35
ss       0x2b         43
--Type <RET> for more, q to quit, c to continue without paging--

```

рис.19 info register

Узнаю значение переменной msg1 по имени (рис. 20).

```
0x804a000 <msg1>: "Hello, "
```

рис.20 Значение переменной по имени

Меняю первый символ переменной msg1 (рис. 21).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
```

рис.21 Изменение переменной

Также меняю первый символ переменной msg2 (рис. 22).

```
(gdb) set {char}msg2='W'
'msg2' has unknown type; cast it to its declared type
(gdb) set {char}&msg2
```

рис.22 Изменение второй переменной

Вывожу значение регистра edx в различных форматах (в шестнадцатеричном, двоичном и символьном форматах) (рис. 23).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$3 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 2
```

рис.23 Изменение значений в разные форматы

С помощью команды set изменяю значение регистра ebx (рис. 24).

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$6 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$7 = 2
(gdb) 
```

рис.24 Изменение значений ebx

Значение регистра отличаются, так как в первом случае мы выводим код символа 2, который в десятичной системе счисления равен 50, а во втором случае выводится число 2, представленное в этой же системе.

4.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, который выводит на экран аргументы, в файл с именем lab09-3.asm (рис. 25).

```
adsvettsova@nnp ~/w/s/2/C/arth-pc/l/lab09/lab9 master i3 i31 713 $ cd ~/work/study/2023-2024/Computer architecture/arth-pc/labs/lab09/lab9
adsvettsova@nnp ~/w/s/2/C/arth-pc/l/lab09/lab9 master i3 i31 713 $ cp ~/work/study/2023-2024/Computer architecture/arth-pc/labs/lab08/lab8-2.asm ~/work/study/2023-2024/Computer architecture/arth-pc/labs/lab09/lab09-3.asm
adsvettsova@nnp ~/w/s/2/C/arth-pc/l/lab09/lab9 master i3 i31 713 $ [ 23:24:45]
```

рис.25 Копирование файла

Создаю исполняемый файл,использую ключ `-args` для загрузки программы в GDB. Загружаю исполняемый файл,указав аргументы (рис. 26).

```
~/study/2023-2024/Computer architecture/rh-pc/labs/lab09/labs/lab09-3.asm
adsvettsova@nnp ~/w/s/2/C/rh-pc/l/lab09/lab9 master +3 131 713 $ nasm -f elf
-o lab09-3.lst lab09-3.asm
adsvettsova@nnp ~/w/s/2/C/rh-pc/l/lab09/lab9 master +3 131 713 $ ld -m elf_i
386 -o lab09-3.o lab09-3.o
adsvettsova@nnp ~/w/s/2/C/rh-pc/l/lab09/lab9 master +3 131 713 $ gdb --args
lab09-3 аргумент1 аргумент 2 'аргумент 3'

GNU gdb (Ubuntu 9.2-0ubuntu1-20.04.1) 9.2
Copyright (C) 2020 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:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
```

рис.26 Создание файла

Устанавливаю точку останова перед первой инструкцией в программе и запускаю её (рис. 27).

```
Reading symbols from lab09-3.o...
(gdb) b start
Breakpoint 1 at 0x00490e8
(gdb) run
Starting program: /home/adsvettsova/work/study/2023-2024/Computer architecture/
rh-pc/labs/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, 0x00490e8 in start ()
```

рис.27 Запуск программы с точкой останова

Просматриваю адрес вершины стека,который хранится в регистре `esp` (рис. 28).

```
**value can't be converted to integer.
(gdb) x/x $esp
0xffffcf50: 0x00000005
(gdb) []
```

рис.28 Регистр `esp`

Ввожу другие позиции стека- в отличие от адресов,располагается адрес в памяти: имя,первый аргумент,второй и т.д (рис. 29).

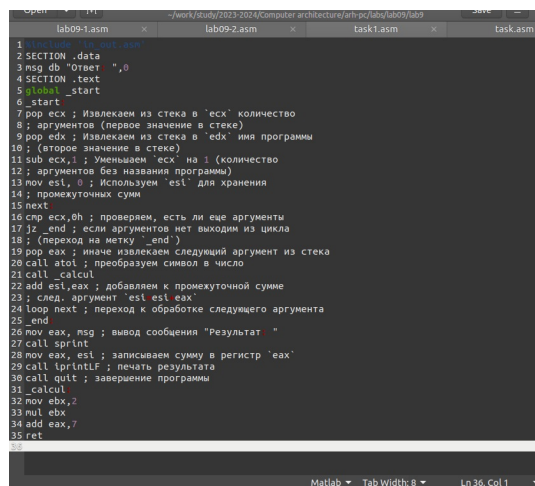
```
0xffffcf50: 0x00000005
(gdb) x/s *(void**)(($esp + 4)
0xffffd146: "/home/adsvettsova/work/study/2023-2024/Computer architecture/a
rh-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)(($esp + 8)
0xffffd122: "аргумент1"
(gdb) x/s *(void**)(($esp + 12)
0xffffd1b4: "аргумент"
(gdb) x/s *(void**)(($esp + 16)
0xffffd1c5: "2"
(gdb) x/s *(void**)(($esp + 20)
0xffffd1d7: "аргумент 3"
(gdb) x/s *(void**)(($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) []
```

рис.29 Позиции стека

Количество аргументов командной строки 4,следовательно и шаг равен четырем.

4.6 Задание для самостоятельной работы

Создаю файл для первого самостоятельного задания,который будет называться `task1.asm`. Редактирую код программы `lab8-4.asm`,добавив подпрограмму,которая вычисляет значения функции `f(x)` (рис. 30).



```
1  _start:
2  SECTION .data
3  msg db "Ответ: ",0
4  SECTION .text
5  global _start
6  _start:
7  pop ecx ; Извлекаем из стека в `ecx` количество
8  ; аргументов (первое значение в стеке)
9  pop edx ; Извлекаем из стека в `edx` имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем `esi` для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку `_end`)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 call _calcul
22 add esi,eax ; добавляем к промежуточной сумме
23 ; след. аргумент `esi=esi+eax`
24 loop next ; переход к обработке следующего аргумента
25 _end:
26 mov eax, msg ; вывод сообщения "Результат: "
27 call sprint
28 mov eax, esi ; записываем сумму в регистр `eax`
29 call iprintLF ; печать результата
30 call quit ; завершение программы
31 _calcul:
32 mov ebx,2
33 mul ebx
34 add eax,7
35 ret
```

рис.30 Редактирование файла

```
%include 'in_out.asm'
SECTION .data
msg db "Ответ: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
mov esi, 0 ; Используем `esi` для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
call _calcul
add esi,eax ; добавляем к промежуточной сумме
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр `eax`
call iprintLF ; печать результата
call quit ; завершение программы
_calcul:
mov ebx,2
mul ebx
```

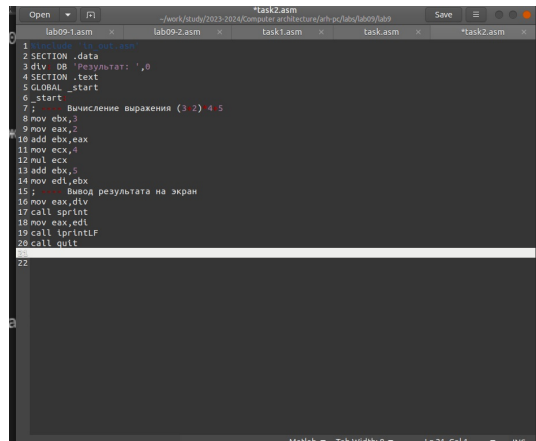
```
add eax,7
ret
```

Создаю исполняемый файл и ввожу аргументы (рис. 31).
Программа работает верно.

```
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab09/lab9 master +3 131 713 $ nasm -f elf
task1.asm
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab09/lab9 master +3 131 713 $ ld -m elf_i
386 task1.o -o task1
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab09/lab9 master +3 131 713 $ ./task1 24
188 15 19
01bet: 520
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab09/lab9 master +3 131 713 $
```

рис.31 Запуск программы

Создаю файл и ввожу код из листинга 9.3 (рис. 32).



```
1 section .text
2 section .data
3 div_00: resb 1024
4 section .text
5 global _start
6 _start:
7     ; Вычисление выражения (3*2)*4*5
8     mov ebx,3
9     mov ecx,2
10    add ebx,ecx
11    mov ecx,4
12    mul ecx
13    add ebx,5
14    mov edi,ebx
15    ; Вывод результата на экран
16    mov eax,div
17    call sprintf
18    mov eax,edi
19    call printf
20    call quit
21
22
```

рис.32 Редактирование файла

Открываю файл в отладчике GDB и запускаю программу (рис. 33, рис.34). Программа выдает ответ 10.

```
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab09/lab9 master +3 131 713 $ nasm -f elf
-g -l task2.lst task2.asm
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab09/lab9 master +3 131 713 $ ld -m elf_i
386 -o task2 task2.o
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab09/lab9 master +3 131 713 $ gdb task2
GNU gdb (Ubuntu 9.2-2ubuntu1~20.04.1) 9.2
```

рис.33 Запуск программы в отладчике

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from task2...
(gdb) run
Starting program: /home/adsvetsova/work/study/2023-2024/Computer architecture/
arrh-pc/labs/lab09/lab9/task2
Результат: 10
[Inferior 1 (process 24608) exited normally]
```

рис.34 Запуск программы в отладчике

Просматриваю дисассимилированный код программы, ставлю точку останова перед прибавлением 5 и открываю значения регистров на данном этапе (рис. 35).

```
(gdb) disassemble _start
Dump of assembler code for function _start:
0x080490e0 <+0>: mov $0x3,%ebx
0x080490e1 <+5>: mov $0x2,%eax
0x080490f2 <+10>: add %eax,%ebx
0x080490f4 <+12>: mov $0x4,%ecx
0x080490f9 <+17>: mul %ecx
0x080490fb <+19>: add $0x5,%ebx
0x080490fe <+22>: mov %ebx,%edi
0x08049100 <+24>: mov $0x804a000,%eax
0x08049105 <+29>: call 0x0804900f <sprintf>
0x0804910a <+34>: mov %edi,%eax
0x0804910c <+36>: call 0x08049006 <printf>
0x08049111 <+41>: call 0x08049000 <quit>
```

рис.35 Действия в отладчике

Как можно увидеть, регистр `ecx` со значением 4 умножается не на `ebx`, сложенным с `eax`, а только с `eax` со значением 2. Значит нужно поменять значения регистров (например присвоить `eax` значение 3 и просто прибавит 2. После изменений программа будет выглядеть следующим образом:

```
%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
```

Пробуем запустить программу (рис. 36). Она работает верно.

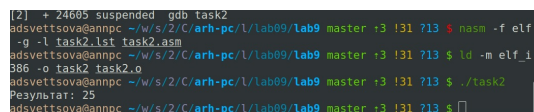


рис.36 Запуск программы

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. 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/Learningbash-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, 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. : МАКС 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 Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).