**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

# ОТЧЕТ
# ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

*дисциплина: Архитектура компьютера*

Студент: Бешкуров Михаил

Группа: НКНбд-01-18

**МОСКВА**

2018 г.

## Цель работы

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

## Ход работы

1. Написал программу со следующим алгоритмом:
– ввести с клавиатуры символьную строку в буфер;
– изменить порядок следования символов в строке на противоположный; положение символа 10 (\n) остается без изменений;
– вывести результат на экран;
– завершить программу.

Код программы.

```
SECTION .data
    text:<----->DB 'Input str:',10
    textLen:<-->EQU $-text

SECTION .bss
    input:<---->RESB 80

SECTION .text
    GLOBAL _start

_start:
    mov eax,4
    mov ebx,1
    mov ecx,text
    mov edx,textLen
    int 80h

    mov eax,3
    mov ebx,0
    mov ecx,input
    mov edx,80
    int 80h

    mov esi,input
    mov ecx,0

lp:
    mov bl,[esi+ecx]
    cmp bl,10
    je quit
    push bx
    inc ecx
    jmp lp

quit:
    pop bx
    mov [esi],bl
    inc esi
    loop quit

    mov eax,4
    mov ebx,1
    mov ecx,input
    mov edx,80
    int 80h

    mov eax,1
    mov ebx,0
    int 80h
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перем~тить 7Поиск 8Удалить 9МенюМС 10Выход
```

Листинг программы.

```
     1                                  SECTION .data
     2 00000000 496E70757420737472-         text:   DB 'Input str:',10
     3 00000009 3A0A
     4                                      textLen:    EQU $-text
     5
     6                                  SECTION .bss
     7 00000000 <res 00000050>              input:  RESB 80
     8
     9                                  SECTION .text
    10                                      GLOBAL _start
    11
    12                                  _start:
    13 00000000 B804000000                  mov eax,4
    14 00000005 BB01000000                  mov ebx,1
    15 0000000A B9[00000000]                mov ecx,text
    16 0000000F BA0B000000                  mov edx,textLen
    17 00000014 CD80                        int 80h
    18
    19 00000016 B803000000                  mov eax,3
    20 0000001B BB00000000                  mov ebx,0
    21 00000020 B9[00000000]                mov ecx,input
    22 00000025 BA50000000                  mov edx,80
    23 0000002A CD80                        int 80h
    24
    25 0000002C BE[00000000]                mov esi,input
    26 00000031 B900000000                  mov ecx,0
    27
    28                                  lp:
    29 00000036 8A1C0E                      mov bl,[esi+ecx]
    30 00000039 80FB0A                      cmp bl,10
    31 0000003C 7405                        je quit
    32 0000003E 6653                        push bx
    33 00000040 41                          inc ecx
    34 00000041 EBF3                        jmp lp
    35
    36                                  quit:
    37 00000043 665B                        pop bx
    38 00000045 881E                        mov [esi],bl
    39 00000047 46                          inc esi
    40 00000048 E2F9                        loop quit
    41
    42 0000004A B804000000                  mov eax,4
    43 0000004F BB01000000                  mov ebx,1
    44 00000054 B9[00000000]                mov ecx,input
    45 00000059 BA50000000                  mov edx,80
    46 0000005E CD80                        int 80h
    47
    48 00000060 B801000000                  mov eax,1
    49 00000065 BB00000000                  mov ebx,0
    50 0000006A CD80                        int 80h
```

Текст ▾ Ширина табуляции: 4 ▾ Стр 1, Стлб 1 ВСТ

Программа работает.

```
mihail@mihail-beshkurov ~/Рабочий стол/rudn/lab5 $  make build
nasm -f elf32 -l lab.lst -o lab.o lab5.asm
ld -o lab lab.o
mihail@mihail-beshkurov ~/Рабочий стол/rudn/lab5 $  ./lab
Input str:
MishaBesh
hseBahsiM
mihail@mihail-beshkurov ~/Рабочий стол/rudn/lab5 $
```

2. Загрузить программу в отладчик можно двумя способами.
– С помощью команды gdb [имя файла]
– Запустить gdb. И выполнить команду file <имя файла>.

Загрузил программу в отладчик.

```
mihail@mihail-beshkurov ~/Рабочий стол/rudn/lab5 $  gdb lab
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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 "i686-linux-gnu".
Type "show configuration" for configuration details.
```

3. Просмотрел дизассемблированный код программы, введя команду disassemble _start

```
(gdb) disassemble _start
Dump of assembler code for function _start:
   0x08048080 <+0>:     mov    $0x4,%eax
   0x08048085 <+5>:     mov    $0x1,%ebx
   0x0804808a <+10>:    mov    $0x80490ec,%ecx
   0x0804808f <+15>:    mov    $0xb,%edx
   0x08048094 <+20>:    int    $0x80
   0x08048096 <+22>:    mov    $0x3,%eax
   0x0804809b <+27>:    mov    $0x0,%ebx
   0x080480a0 <+32>:    mov    $0x80490f8,%ecx
   0x080480a5 <+37>:    mov    $0x50,%edx
   0x080480aa <+42>:    int    $0x80
   0x080480ac <+44>:    mov    $0x80490f8,%esi
   0x080480b1 <+49>:    mov    $0x0,%ecx
End of assembler dump.
```

Вывод дисассемблера

4. Переключил дизассемблер GDB с синтаксиса ATT на синтаксис Intel и снова выполнил показ дизассемблированного кода.
Команды: (set disassembly-flavor intel) и (disassemble _start)

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
   0x08048080 <+0>:     mov    eax,0x4
   0x08048085 <+5>:     mov    ebx,0x1
   0x0804808a <+10>:    mov    ecx,0x80490ec
   0x0804808f <+15>:    mov    edx,0xb
   0x08048094 <+20>:    int    0x80
   0x08048096 <+22>:    mov    eax,0x3
   0x0804809b <+27>:    mov    ebx,0x0
   0x080480a0 <+32>:    mov    ecx,0x80490f8
   0x080480a5 <+37>:    mov    edx,0x50
   0x080480aa <+42>:    int    0x80
   0x080480ac <+44>:    mov    esi,0x80490f8
   0x080480b1 <+49>:    mov    ecx,0x0
End of assembler dump.
```

Отличия синтаксиса ATT от Intel:
- Перед именем регистра ставиться %
- Перед инструкцией ставиться префикс $
- Адрес инструкции пишется перед именем регистра (в Intel наоборот)

Адрес второй инструкции в формате 0x12345678: 0x08048085

5. Установил точку останова на второй инструкции, указав её с помощью команды break и адреса инструкции на которой необходимо остановиться.
Команда: break *0x08048085

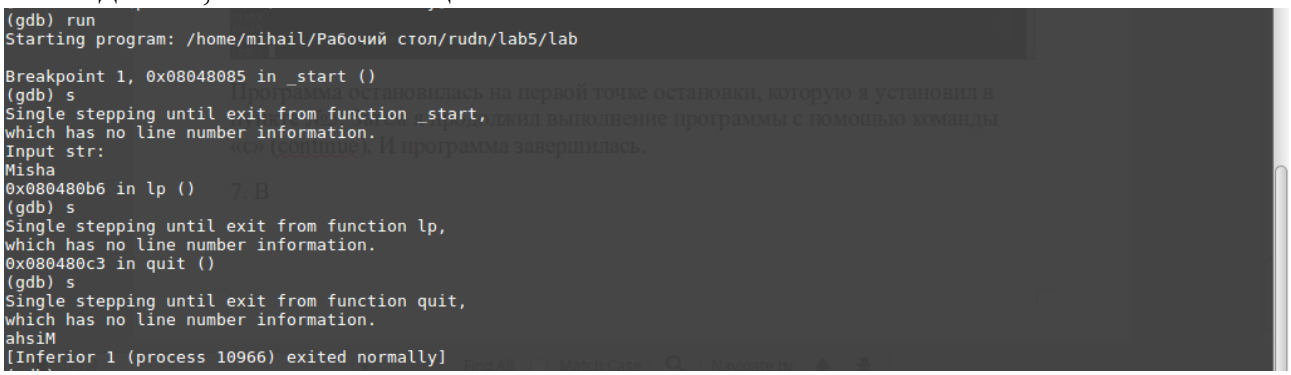
```
(gdb) break *0x08048085
Breakpoint 1 at 0x8048085
(gdb)
```

6. Выполнил программу с помощью команды run.

```
(gdb) run
Starting program: /home/mihail/Рабочий стол/rudn/lab5/lab

Breakpoint 1, 0x08048085 in _start ()
(gdb) c
Continuing.
Input str:
Misha
ahsiM
[Inferior 1 (process 10499) exited normally]
(gdb)
```

Программа остановилась на первой точке остановки, которую я установил в пункте №5. Затем я продолжил выполнение программы с помощью команды «c» (continue). И программа завершилась.

7. Выполнил программу пошагово. Сначала запустил программу с помощью команды run, затем с помощью s шаг за шагом.

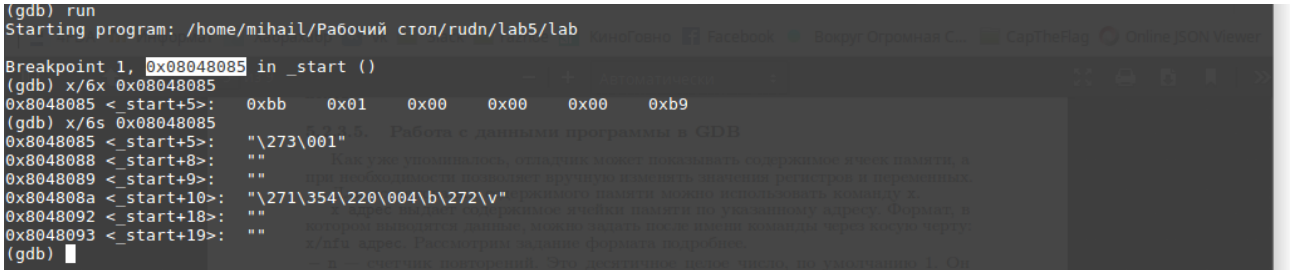


```
(gdb) run
Starting program: /home/mihail/Рабочий стол/rudn/lab5/lab

Breakpoint 1, 0x08048085 in _start ()
(gdb) s
Single stepping until exit from function _start,
which has no line number information.
Input str:
Misha
0x080480b6 in lp ()
(gdb) s
Single stepping until exit from function lp,
which has no line number information.
0x080480c3 in quit ()
(gdb) s
Single stepping until exit from function quit,
which has no line number information.
ahsiM
[Inferior 1 (process 10966) exited normally]
```

8. Посмотрел содержимое регистров в окне и с помощью команды info r. Предварительно запустив программу.

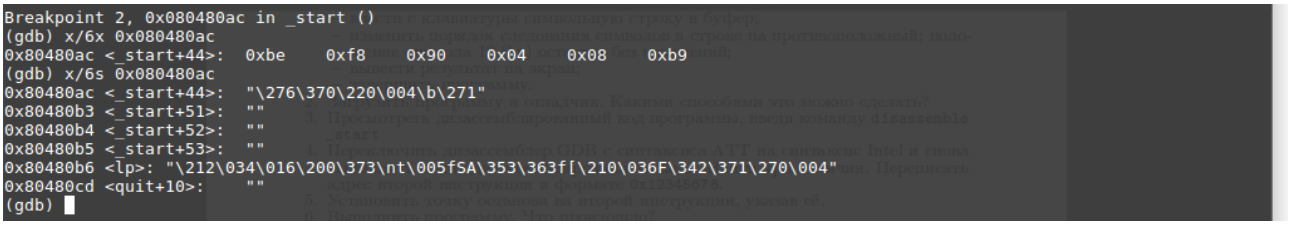
```
(gdb) run
Starting program: /home/mihail/Рабочий стол/rudn/lab5/lab

Breakpoint 1, 0x08048085 in _start ()
(gdb) info r
eax            0x4      4
ecx            0x0      0
edx            0x0      0
ebx            0x0      0
esp            0xbffff110       0xbffff110
ebp            0x0      0x0
esi            0x0      0
edi            0x0      0
eip            0x8048085        0x8048085 <_start+5>
eflags         0x202    [ IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x0      0
(gdb)
```

9. Выполнил программу до места заполнения входного буфера. Вывел содержимое входного буфера в шестнадцатеричном формате и в символьном виде.
Команды: x/6x 0x08048085 — для вывода в 16-ом формате,
x/6s 0x08048085 — для вывода в символьном виде.



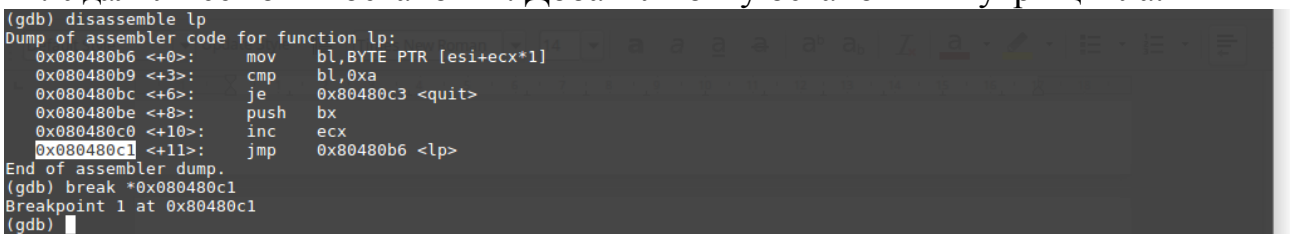
```
(gdb) run
Starting program: /home/mihail/Рабочий стол/rudn/lab5/lab

Breakpoint 1, 0x08048085 in _start ()
(gdb) x/6x 0x08048085
0x8048085 <_start+5>:   0xbb    0x01    0x00    0x00    0x00    0xb9
(gdb) x/6s 0x08048085
0x8048085 <_start+5>:   "\273\001"
0x8048088 <_start+8>:   ""
0x8048089 <_start+9>:   ""
0x804808a <_start+10>:  "\271\354\220\004\b\272\v"
0x8048092 <_start+18>:  ""
0x8048093 <_start+19>:  ""
(gdb)
```

10. Установил вторую точку остановки после заполнения буфера. Вывел содержимое входного буфера в шестнадцатеричном формате и в символьном виде после его заполнения.

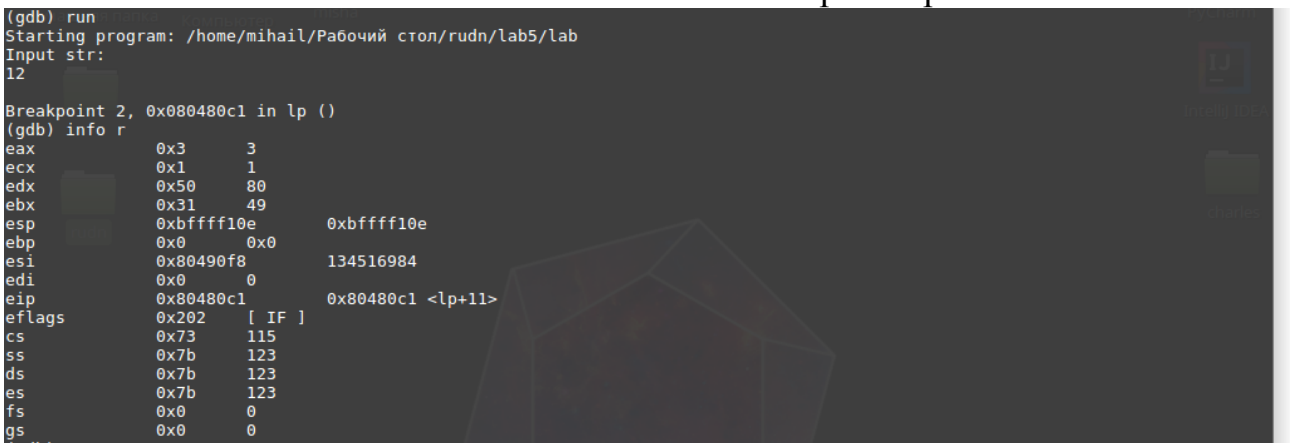


```
Breakpoint 2, 0x080480ac in _start ()
(gdb) x/6x 0x080480ac
0x80480ac <_start+44>:  0xbe    0xf8    0x90    0x04    0x08    0xb9
(gdb) x/6s 0x080480ac
0x80480ac <_start+44>:  "\276\370\220\004\b\271"
0x80480b3 <_start+51>:  ""
0x80480b4 <_start+52>:  ""
0x80480b5 <_start+53>:  ""
0x80480b6 <lp>: "\212\034\016\200\373\nt\005fSA\353\363f[\210\036F\342\371\270\004"
0x80480cd <quit+10>:    ""
(gdb)
```

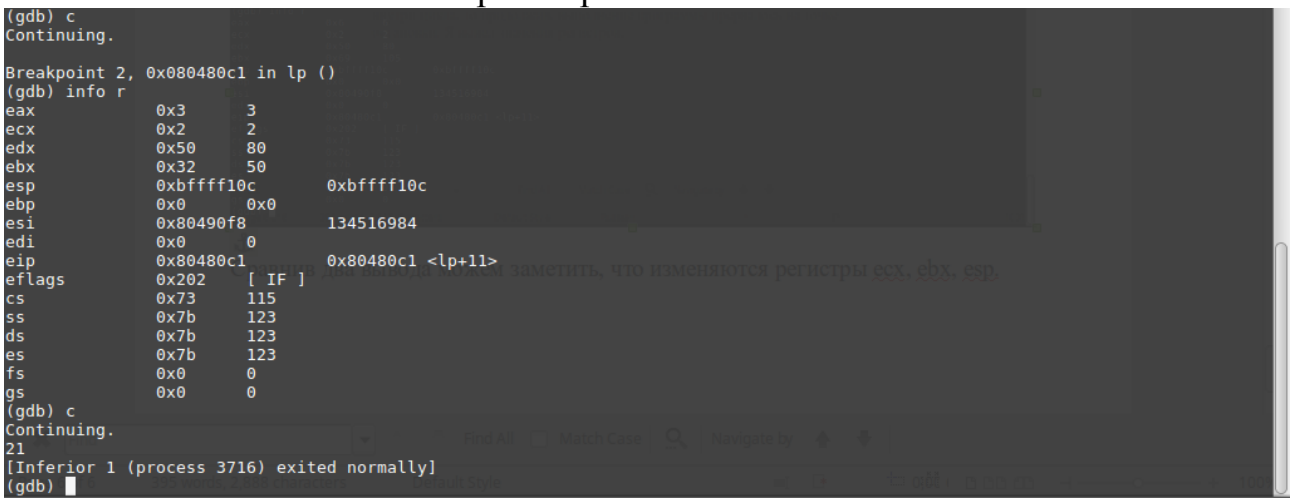
11. Удалил все точки остановки. Добавил точку остановки внутри цикла.




Затем запустил дизассемблер. Программа прошла один проход по циклу и остановилась на точке останова. Я вывел значения регистров.

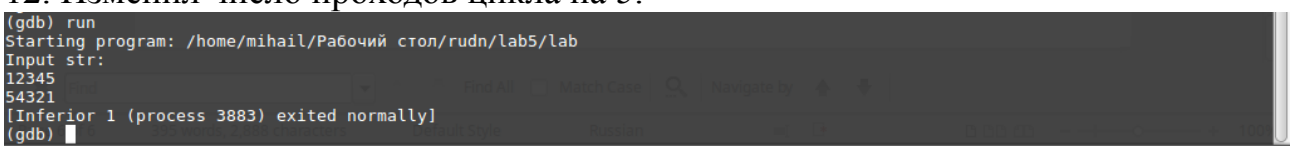



Затем продолжил выполнение программы. Так как точка остановки находилась внутри цикла, то продолжив, выполнение программы прервалось на точке остановки. Я вывел значения регистров.

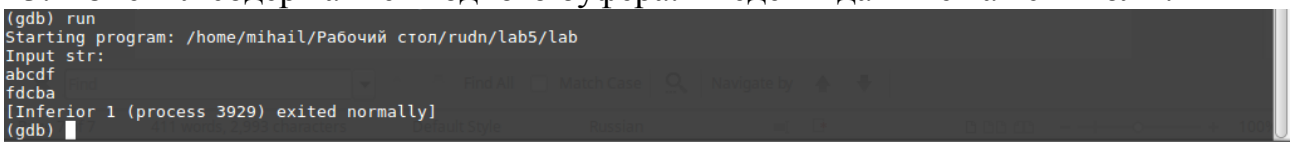



Сравнив два вывода можем заметить, что изменяются регистры ecx, ebx, esp.

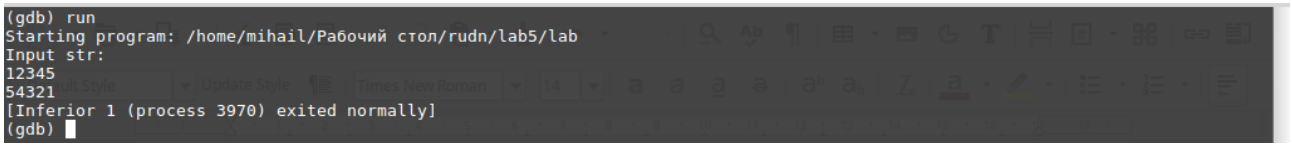
12. Изменил число проходов цикла на 5.



13. Изменил содержание входного буфера. Введены данные как символы.




И как числа.




**Вывод**

Познакомился с методами отладки при помощи GDB и его основными возможностями. Приобрёл навыки написания программ с использованием циклов.

**Контрольные вопросы**

1. Для поиска и исправления ошибок в программе.

2. Breakpoint — Точка останова. Остановка происходит, когда выполнение программы доходит до определённой строки, адреса, функции.
Watchpoint — точка простмотра. Выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его.
Catchpoint — точка отлова. Приостановка программы происходит при определённом событии(например, exception).
Call stack — LOFI-стек, хранящий информацию для возврата управления из подпрограмм (процедур, функций) в программу (или подпрограмму, при вложенных или рекурсивных вызовах) и/или для возврата в программу из обработчика прерывания (в том числе при переключении задач в многозадачной среде).

3. Не всегда при работе программы видны ошибки, поэтому отладочная иформация помогает найти их все.Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g.

4. Запуск: gdb <имя_файла>
Поставить точку останова: break *<адрес_строки>
Продолжить: continue (кратко c)
Распечатать локальные переменные: info locals
Завершить работу: quit

5. Прямая адресация предполагает, что эффективный адрес является частью команды. В случае косвенной адресации адрес ячейки памяти заносят в регистр

и используют его в качестве указателя.

6. Если некоторую группу команд необходимо повторить определённое количество раз, то можно воспользоваться взаимными переходами между метками.

7. Стек — это абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»)

8. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек, указатель стека уменьшается, а при извлечении - увеличивается.