

Информатика

Занятие 7

Грозов Владимир Андреевич

va_groz@mail.ru

Assembler. Целочисленное умножение и деление

- **Умножение**
 - Беззнаковое: **mul**
 - `mov ax, 10`
`mov r8w, 20`
`mul r8w`
 - Со знаком: **imul**
 - `mov ax, -10`
`mov r12w, 20`
`imul r12w`

Assembler. Целочисленное умножение и деление

- **Деление**

- Беззнаковое: **div**

- `mov eax, 110`
`mov r9d, 20`
`div r9d`

- Со знаком: **idiv**

- `mov eax, 120`
`mov r8d, -30`
`idiv r8d`

Assembler. Очень простая программа

- Найти десятое число Фибоначчи

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

```
mov r8b, 0
```

```
mov r9b, 1
```

```
mov cx, 9
```

```
lp_fib:
```

```
    add r8b, r9b
```

```
    mov r10b, r9b
```

```
    mov r9b, r8b
```

```
    mov r8b, r10b
```

```
    loop lp_fib
```

Assembler. Вывод на печать

- **Язык C:**

```
Int main()  
{  
    ...  
    printf("%d", fib_10);  
}
```

- **NASM:**

В явном виде универсальной команды, которая выводила бы на печать любое значение, не существует

Assembler. Вывод на печать

- **Посимвольный (поразрядный) вывод на печать**
 - Регистры
- **Системные вызовы**
 - `int ...` (например, `int 80h`). Устаревший инструмент.
 - `Syscall`
 - ...
- **Использование макросов из файла (`stud_io.inc`)**
- **Использование функций языка C (`printf`)**

Assembler. Прерывания

- **Виды прерываний**
 - Аппаратные
 - Программные
- **Вызов программного прерывания:**
 - `int x`, `x` – номер обработчика прерывания
 - $0 \leq x \leq 255$
- **Суть:**
 - Пользователь передаёт управление ядру, с целью выполнения тех или иных действий

Assembler. Прерывания

Необходимые регистры для корректной работы команды `int`

- `rax`
 - Номер системного вызова
- `rbx`
 - Номер потока ввода/вывода
- `rcx`
 - Вводимое/выводимое значение
- `rdx`
 - Длина ввода/вывода

Assembler. Прерывания

Значения регистров при вводе и выводе

Регистр	Консольный ввод	Вывод на консоль
rax	3	4
rbx	2	1

- Значения регистров rcx и rdx – сообщение, которое будет выведено на экран и его длина (в байтах) соответственно

Assembler. Прерывания

Пример ввода числа с клавиатуры

- ```
section .text
 global _start
_start:
 mov eax, 3
 mov ebx, 2
 mov ecx, num
 mov edx, 5
 int 80h
 ...
section .bss
 num resb 5
```

# Assembler. Прерывания

## Пример вывода числа на консоль

```
section .text
 global _start
_start:
 mov eax, 4
 mov ebx, 1
 mov ecx, Text
 mov edx, lenText
 int 80h
 ...
section .data
 Text db "AbraCababra!!!", 10
 lenText db $ - Text
```

# Assembler. Syscall

## Таблица системных вызовов

- <https://filippo.io/linux-syscall-table/>

| %rax | Name           | Entry point        | Implementation                               |
|------|----------------|--------------------|----------------------------------------------|
| 0    | read           | sys_read           | <a href="#">fs/read_write.c</a>              |
| 1    | write          | sys_write          | <a href="#">fs/read_write.c</a>              |
| 2    | open           | sys_open           | <a href="#">fs/open.c</a>                    |
| 3    | close          | sys_close          | <a href="#">fs/open.c</a>                    |
| 4    | stat           | sys_newstat        | <a href="#">fs/stat.c</a>                    |
| 5    | fstat          | sys_newfstat       | <a href="#">fs/stat.c</a>                    |
| 6    | lstat          | sys_newlstat       | <a href="#">fs/stat.c</a>                    |
| 7    | poll           | sys_poll           | <a href="#">fs/select.c</a>                  |
| 8    | lseek          | sys_lseek          | <a href="#">fs/read_write.c</a>              |
| 9    | mmap           | sys_mmap           | <a href="#">arch/x86/kernel/sys_x86_64.c</a> |
| 10   | mprotect       | sys_mprotect       | <a href="#">mm/mprotect.c</a>                |
| 11   | munmap         | sys_munmap         | <a href="#">mm/mmap.c</a>                    |
| 12   | brk            | sys_brk            | <a href="#">mm/mmap.c</a>                    |
| 13   | rt_sigaction   | sys_rt_sigaction   | <a href="#">kernel/signal.c</a>              |
| 14   | rt_sigprocmask | sys_rt_sigprocmask | <a href="#">kernel/signal.c</a>              |
| 15   | rt_sigreturn   | stub_rt_sigreturn  | <a href="#">arch/x86/kernel/signal.c</a>     |
| 16   | ioctl          | sys_ioctl          | <a href="#">fs/ioctl.c</a>                   |
| 17   | pread64        | sys_pread64        | <a href="#">fs/read_write.c</a>              |
| 18   | pwrite64       | sys_pwrite64       | <a href="#">fs/read_write.c</a>              |

# Assembler. Syscall

## Необходимые регистры для корректной работы syscall

- rax
  - Номер системного вызова
- rdi
  - Номер потока ввода/вывода
- rsi
  - Вводимое/выводимое значение
- rdx
  - Длина ввода/вывода

# Assembler. Syscall

## Значения регистров при вводе и выводе

| Регистр | Консольный ввод | Вывод на консоль |
|---------|-----------------|------------------|
| rax     | 0               | 1                |
| rdi     | 1               | 1                |

- Значения регистров rsi и rdx – сообщение, которое будет выведено на экран и его длина (в байтах) соответственно

# Assembler. Syscall

## Пример ввода числа с клавиатуры

- ```
section .text
    global _start
_start:
    mov eax, 0
    mov rdi, 1
    mov rsi, num
    mov edx, 5
    syscall
    ...
section .bss
    num resb 5
```

Assembler. Syscall

Пример вывода числа на консоль

- ```
section .text
 global _start
_start:
 mov eax, 1
 mov rdi, 1
 mov rsi, Text
 mov edx, lenText
 syscall

...
section .data
 Text db "AbraCababra!!!", 10
 lenText db $ - Text
```



# Assembler. Посимвольный вывод значения регистра

## Причины

- Надёжность (и контролируемость) работы
- Реализация с небольшими изменениями под конкретную ситуацию
- Возможность лучше понять принципы работы операторов вывода
- Just for fun

## Недостатки

- Не самая высокая эффективность
- Представленный далее вариант «заточен» на числа размером до 16 бит.

# Assembler. Посимвольный вывод значения регистра

**Задача:** Вывести на экран результат сложения двух чисел

## Общий алгоритм

- Объявление переменных

```
section .data
 x dw 0 ; длина числа (количество разрядов)
 n dw 10 ; основание системы счисления
 c dw 0 ; переменная, в которой хранится число
 m1 db 1 ; переменная, в которой хранится разряд числа
 ms_e db 10 ; номер символа «перенос строки» в таблице
 ; ASCII
```

# Assembler. Посимвольный вывод значения регистра

## Общий алгоритм

- Разбиение числа на разряды

```
section .text
```

```
;.....
```

```
mov r8w, 100
```

```
mov r9w, 50
```

```
add r8w, r9w
```

; вычисление суммы двух чисел

```
mov [c], r8w
```

```
mov dx, 0
```

```
mov ax, [c]
```

```
lp1:
```

; занесение цифр в стек

```
div word [n]
```

```
push dx
```

```
inc word [x]
```

```
mov dx, 0
```

```
sub ax, 0
```

```
jnz lp1
```

; пока не будет занесено всё число

# Assembler. Посимвольный вывод значения регистра

## Общий алгоритм

- Вывод полученных разрядов числа на экран (в правильном порядке)

```
lp2: pop r10w ; извлечение цифр из стека
 add r10w, 48 ; перевод символов в цифры
 mov [m1], r10b
 dec word [x]
 mov rax, 1
 mov rdi, 1
 mov rsi, m1
 mov rdx, 1
 syscall ; вывод на экран
 sub word [x], 0 ; проверка конца цикла
 jnz lp2
mov rsi, ms_e
mov rdx, 1
syscall
```

# Assembler. Посимвольный вывод значения регистра

## Общий алгоритм

- Корректное завершение программы

```
mov eax, 60
```

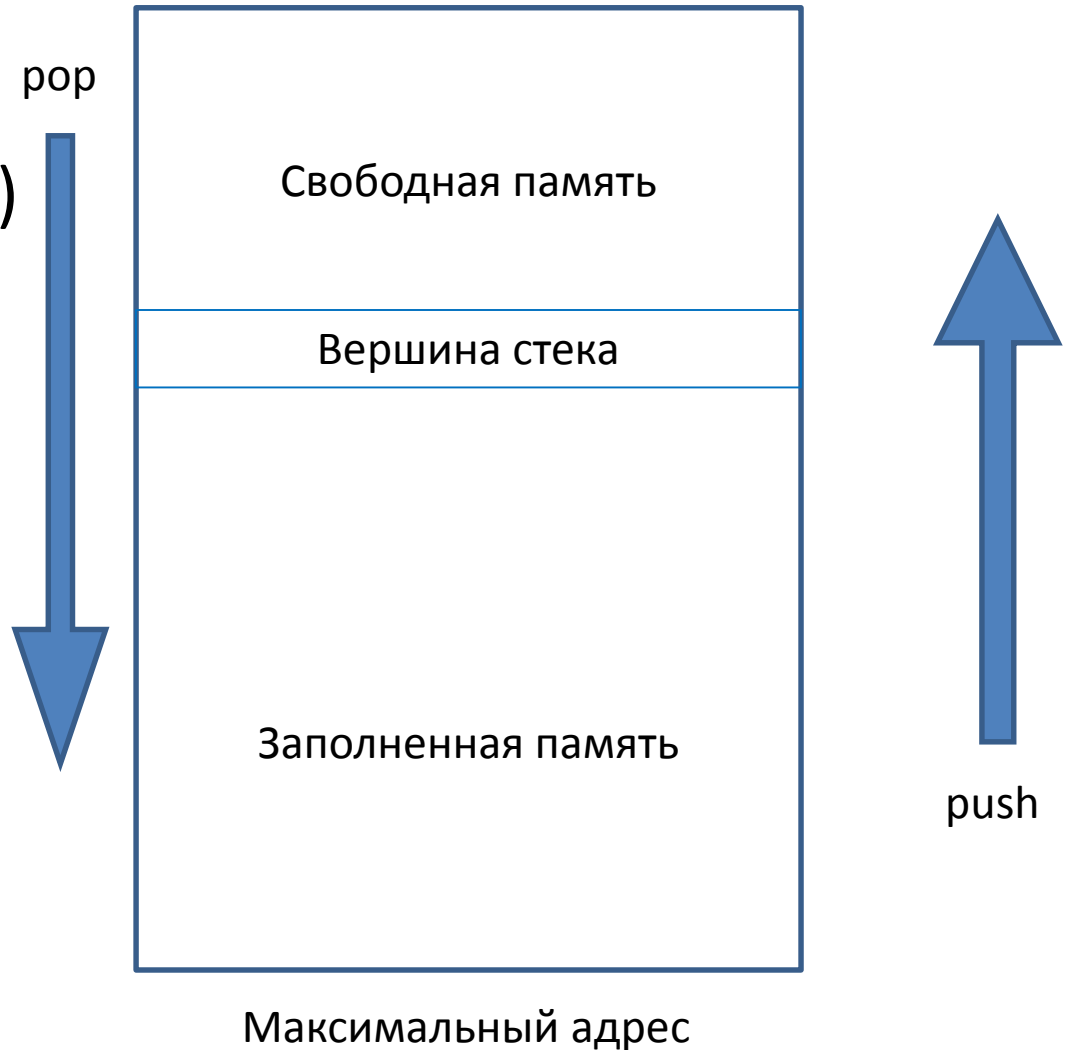
```
xor rdi, rdi
```

```
syscall
```

# Assembler. Работа со стеком

## Стек

- LIFO (Last in – first out)
- Последний элемент, помещённый в стек, находится в вершине стека
- push
- pop



# Assembler. Работа со стеком

- Регистр, используемый для работы со стеком в NASM – RSP
- В регистре RSP хранится адрес вершины стека (адрес, по которому располагается последний занесённый в стек элемент)
- Занесение элемента в стек => адрес вершины (значение RSP) уменьшается
- Извлечение элемента из стека => адрес вершины (значение RSP) увеличивается

# Assembler. Работа со стеком

- Занесение элемента в стек (push)
  - push ecx
  - push x
  - push word [x]
- Извлечение элемента из стека (pop)
  - pop r10d
  - pop word [x]

При работе со стеком операнды должны иметь размер 2 байта, 4 байта или 8 байт!

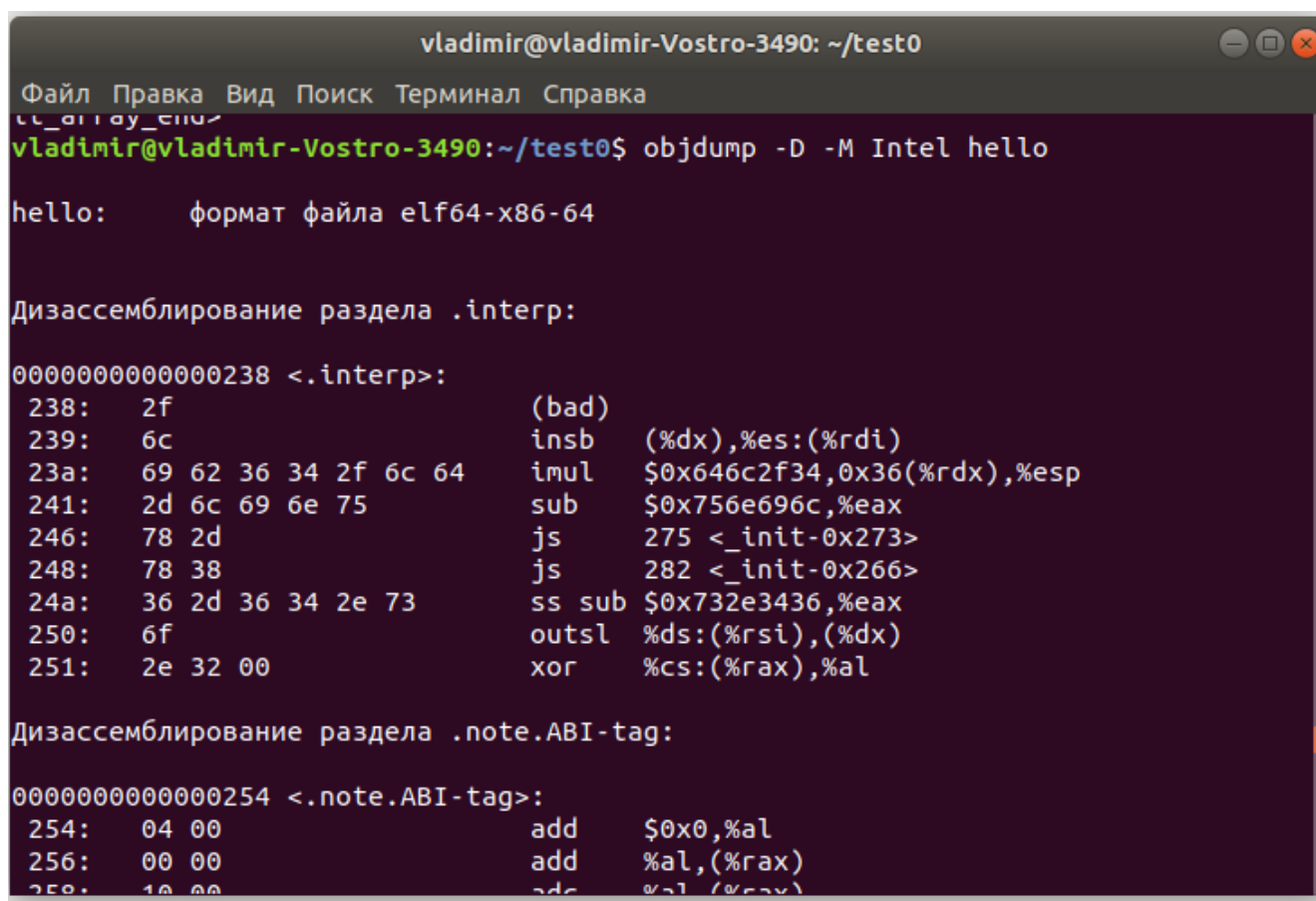


# Дизассемблирование

- Инструментов для дизассемблирования – много
- Используем objdump
- objdump – инструмент исследования объектных файлов. Выводит подробную информацию, содержащуюся в объектном файле
- objdump встроен в ОС Linux, запуск – посредством терминала

# Дизассемблирование. objdump

- Пример:
  - `objdump -D -M Intel prog1`



```
vladimir@vladimir-Vostro-3490: ~/test0
Файл Правка Вид Поиск Терминал Справка
vladimir@vladimir-Vostro-3490:~/test0$ objdump -D -M Intel hello

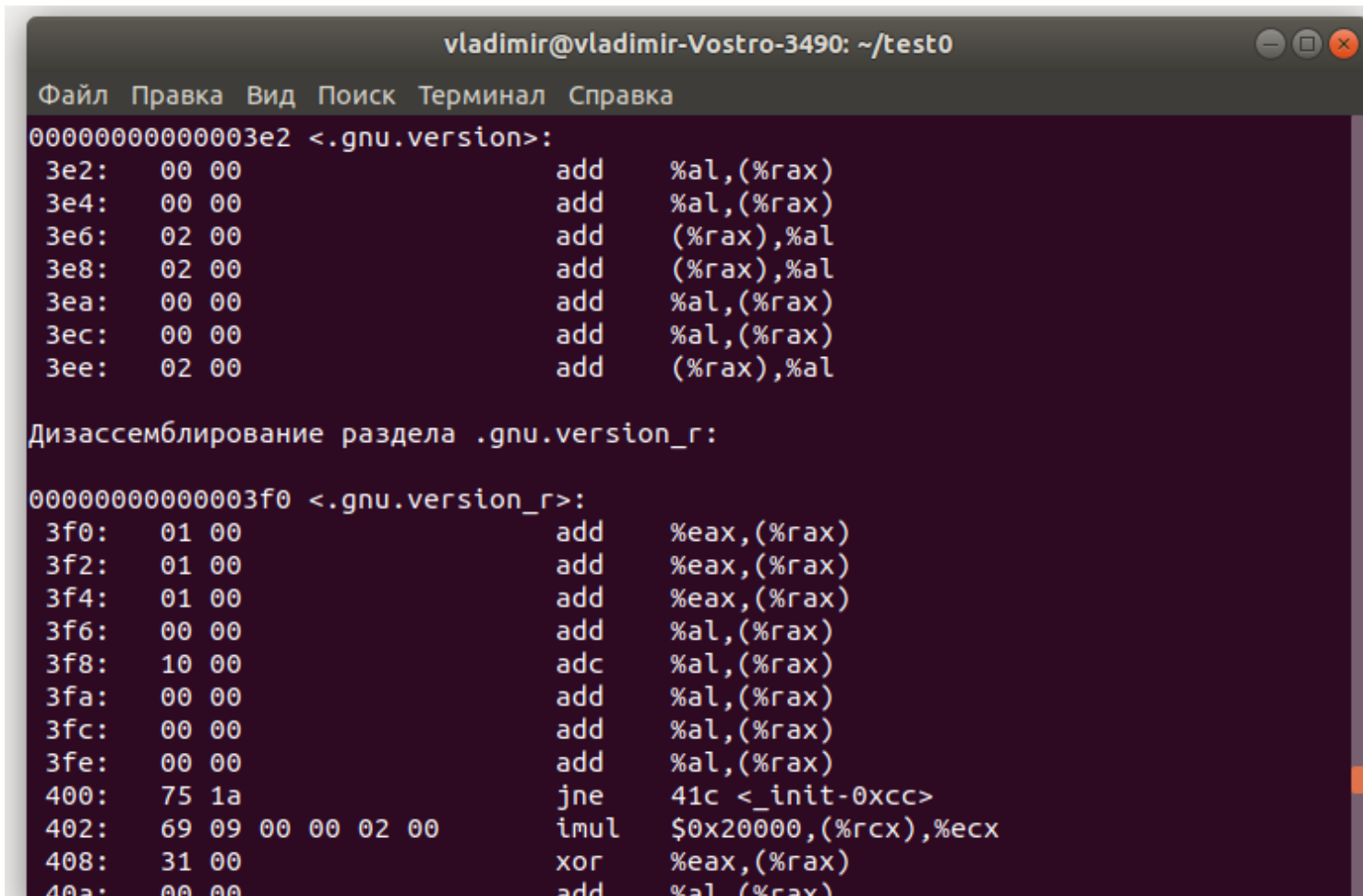
hello: формат файла elf64-x86-64

Дизассемблирование раздела .interp:
0000000000000238 <.interp>:
238: 2f (bad)
239: 6c insb (%dx),%es:(%rdi)
23a: 69 62 36 34 2f 6c 64 imul $0x646c2f34,0x36(%rdx),%esp
241: 2d 6c 69 6e 75 sub $0x756e696c,%eax
246: 78 2d js 275 <_init-0x273>
248: 78 38 js 282 <_init-0x266>
24a: 36 2d 36 34 2e 73 ss sub $0x732e3436,%eax
250: 6f outsl %ds:(%rsi),(%dx)
251: 2e 32 00 xor %cs:(%rax),%al

Дизассемблирование раздела .note.ABI-tag:
0000000000000254 <.note.ABI-tag>:
254: 04 00 add $0x0,%al
256: 00 00 add %al,(%rax)
258: 10 00 add %al,(%rax)
```

# Дизассемблирование. objdump

- Пример: `objdump -D -M intel prog1`  
где-то в дебрях вывода...



```
vladimir@vladimir-Vostro-3490: ~/test0
Файл Правка Вид Поиск Терминал Справка
000000000000003e2 <.gnu.version>:
3e2: 00 00 add %al,(%rax)
3e4: 00 00 add %al,(%rax)
3e6: 02 00 add (%rax),%al
3e8: 02 00 add (%rax),%al
3ea: 00 00 add %al,(%rax)
3ec: 00 00 add %al,(%rax)
3ee: 02 00 add (%rax),%al

Дизассемблирование раздела .gnu.version_r:
000000000000003f0 <.gnu.version_r>:
3f0: 01 00 add %eax,(%rax)
3f2: 01 00 add %eax,(%rax)
3f4: 01 00 add %eax,(%rax)
3f6: 00 00 add %al,(%rax)
3f8: 10 00 adc %al,(%rax)
3fa: 00 00 add %al,(%rax)
3fc: 00 00 add %al,(%rax)
3fe: 00 00 add %al,(%rax)
400: 75 1a jne 41c <_init-0xcc>
402: 69 09 00 00 02 00 imul $0x20000,(%rcx),%ecx
408: 31 00 xor %eax,(%rax)
40a: 00 00 add %al,(%rax)
```

# Дизассемблирование. objdump

- Пример: `objdump -D -M intel prog1`

Где-то в совсем уж непроходимых джунглях...

```
vladimir@vladimir-Vostro-3490: ~/test0
Файл Правка Вид Поиск Терминал Справка
5ed: 00 00 00

000000000000005f0 <__do_global_dtors_aux>:
5f0: 80 3d 19 0a 20 00 00 cmpb $0x0,0x200a19(%rip) # 201010 <__TM
C_END__>
5f7: 75 2f jne 628 <__do_global_dtors_aux+0x38>
5f9: 48 83 3d f7 09 20 00 cmpq $0x0,0x2009f7(%rip) # 200ff8 <__cx
a_finalize@GLIBC_2.2.5>
600: 00
601: 55 push %rbp
602: 48 89 e5 mov %rsp,%rbp
605: 74 0c je 613 <__do_global_dtors_aux+0x23>
607: 48 8b 3d fa 09 20 00 mov 0x2009fa(%rip),%rdi # 201008 <__ds
o_handle>
60e: e8 0d ff ff ff callq 520 <__cxa_finalize@plt>
613: e8 48 ff ff ff callq 560 <deregister_tm_clones>
618: c6 05 f1 09 20 00 01 movb $0x1,0x2009f1(%rip) # 201010 <__TM
C_END__>
61f: 5d pop %rbp
620: c3 retq
621: 0f 1f 80 00 00 00 00 nopl 0x0(%rax)
628: f3 c3 repz retq
62a: 66 0f 1f 44 00 00 nopw 0x0(%rax,%rax,1)
```

# Дизассемблирование. objdump

- Пример: `objdump -D -M intel prog1`  
и вот наступил конец...

```
vladimir@vladimir-Vostro-3490: ~/test0
Файл Правка Вид Поиск Терминал Справка

00000000000201010 <__bss_start>:
 ...

Дизассемблирование раздела .comment:

00000000000000000 <.comment>:
 0: 47 rex.RXB
 1: 43 rex.XB
 2: 43 3a 20 rex.XB cmp (%r8),%spl
 5: 28 55 62 sub %dl,0x62(%rbp)
 8: 75 6e jne 78 <_init-0x470>
 a: 74 75 je 81 <_init-0x467>
 c: 20 37 and %dh,(%rdi)
 e: 2e 33 2e xor %cs:(%rsi),%ebp
 11: 30 2d 31 36 75 62 xor %ch,0x62753631(%rip) # 62753648 <_
end+0x62552630>
 17: 75 6e jne 87 <_init-0x461>
 19: 74 75 je 90 <_init-0x458>
1b: 33 29 xor (%rcx),%ebp
1d: 20 37 and %dh,(%rdi)
1f: 2e 33 2e xor %cs:(%rsi),%ebp
22: 30 00 xor %al,(%rax)

vladimir@vladimir-Vostro-3490:~/test0$
```

# Дизассемблирование. objdump

- Пример:

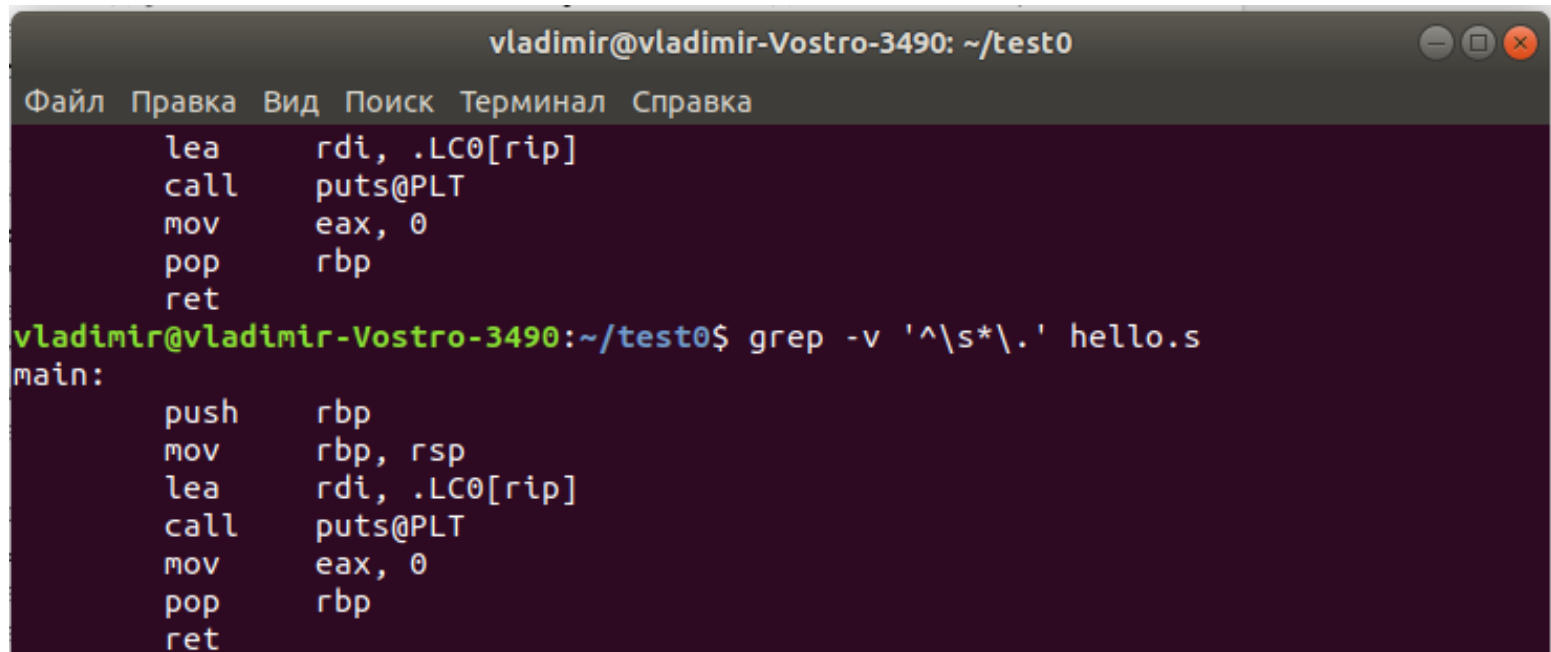
- `objdump -D -M intel prog1 | grep main.: -A20`

```
vladimir@vladimir-Vostro-3490:~/test0$ objdump -D -M Intel hello | grep main.: -A20
0000000000000063a <main>:
 63a: 55 push %rbp
 63b: 48 89 e5 mov %rsp,%rbp
 63e: 48 8d 3d 9f 00 00 00 lea 0x9f(%rip),%rdi # 6e4 <_IO_stdin_u
sed+0x4>
 645: e8 c6 fe ff ff callq 510 <puts@plt>
 64a: b8 00 00 00 00 mov $0x0,%eax
 64f: 5d pop %rbp
 650: c3 retq
 651: 66 2e 0f 1f 84 00 00 nopw %cs:0x0(%rax,%rax,1)
 658: 00 00 00
 65b: 0f 1f 44 00 00 nopl 0x0(%rax,%rax,1)

00000000000000660 <__libc_csu_init>:
 660: 41 57 push %r15
 662: 41 56 push %r14
 664: 49 89 d7 mov %rdx,%r15
 667: 41 55 push %r13
 669: 41 54 push %r12
 66b: 4c 8d 25 46 07 20 00 lea 0x200746(%rip),%r12 # 200db8 <__fr
ame_dummy_init_array_entry>
 672: 55 push %rbp
 673: 48 8d 2d 46 07 20 00 lea 0x200746(%rip),%rbp # 200dc0 <__in
it_array_end>
vladimir@vladimir-Vostro-3490:~/test0$
```

# Дизассемблирование

- Вывод ассемблерного представления программы
- **Пример:**
  - `grep -v '^$*\. ' hello.s`



```
vladimir@vladimir-Vostro-3490: ~/test0
Файл Правка Вид Поиск Терминал Справка
 lea rdi, .LC0[rip]
 call puts@PLT
 mov eax, 0
 pop rbp
 ret
vladimir@vladimir-Vostro-3490:~/test0$ grep -v '^$*\. ' hello.s
main:
 push rbp
 mov rbp, rsp
 lea rdi, .LC0[rip]
 call puts@PLT
 mov eax, 0
 pop rbp
 ret
```