1. Hello World

В данном уроке мы выведем Hello World на языке NASM

Язык ассемблера.

Единственный интерфейс, который программист имеет над реальным оборудованием, - это само ядро. Для написания программ нам нужно использовать **системные вызовы Linux (Windows)**, предоставляемые ядром. Эти системные вызовы представляют собой библиотеку (набор функций), встроенную в операционную систему для обеспечения таких функций, как *чтение ввода с клавиатуры* и *запись вывода на экран*, а также многое другое.

Когда вы вызываете **системный вызов**, ядро немедленно приостанавливает выполнение вашей программы. Затем он свяжется с необходимыми драйверами, необходимыми для выполнения запрошенной вами задачи на оборудовании, а затем вернет управление обратно в вашу программу.

Мы можем выполнить все основные действия (чтение, вывод), *загрузив в EAX число, которое определяет, что мы хотим сделать* (это ещё называется код операции или *OPCODE*), который мы хотим выполнить, и заполнив оставшиеся регистры аргументами, которые мы хотим передать системному вызову. Программное прерывание запрашивается инструкцией INT, после запроса ядро само управляет данными и вызывает функцию из библиотеки с нашими аргументами. Это если объяснять доходчиво.

Таблица с такими кодами

Простейшими кодами вызова являются **sys_write** и **sys_exit**
Для того, чтобы вывести что-то на экран нужно передать значение 4 в eax, 1 в bx, само сообщение в cx, длину сообщения в dx

Разберём первый пример:

```
1  SECTION .data ; объяснение секций будет ниже
2  msg db 'Hello World!', 0Ah
3  ;инициализируем переменную msg задаём ей значение "Hello World"
4  ;0Ah - символ новой строки (в десятичной - 10)
5
6  SECTION .text
7  global  _start ;метка для линкера
8  ;тоже самое что и int main() в C++
9
10 _start: ; начало работы программы
11     mov eax, 4
```

```
12  ;Вызываем sys_write, помещая 4 в EAX
13
14      mov ebx, 1
15  ;Помещая 1 в EBX, мы говорим что хотим стандартный вывод (на экран)
16
17      mov ecx, msg
18  ;Перемещаем адрес, по которому хранится строка в ECX
19
20      mov edx, 13
21  ;Перемещаем длину строки + 1 (символ новой строки) в EDX
22      int 80h
23      ;Вызываем системное прерывание с помощью int 80h
24      ;(128 - в десятиричной)
```

Как вы уже могли заметить, регистр в ассемблере не играет значения NASM все равно напишете вы **EAX** или **eax**

Секции - это блоки кода специального назначения.
В NASM 3 секции:

- .data
- .bss
- .text

.data - нужна для объявления статических переменных, размер которых не будет изменяться. Эта секция хорошо подходит для констант
.bss - секция для резервирования места для переменных. Хорошо подходит для динамических переменных
.data - секция для кода программы

Линкер и этапы создания программы

Линкер нужен для соединения множества программ на NASM.

```
1  global _start
```

Объявляется глобальной **меткой** (метка - названный адрес памяти, для удобства, если по простому, то это просто ссылка, на какую строку программы нужно перейти). Глобальной **_start** объявляется именно для *линкера*

Линкер также называют сборщиком, собирателем ит.д.

Для того, чтобы скомпилировать программу на NASM следует ввести следующее с консоль (*Linux*):

```
1  nasm -f elf имя_файла.asm
2  ld -m elf_i386 имя_файла.o -o имя_исполняемого_файла
```

Имя исполняемого файла может быть каким угодно, также как и исходного (с кодом)

nasm - компилятор

ld - линкер