Лабораторная работа №3. «Составление PE файла и внедрение сигнатуры»

Цель работы: Изучить принципы работы и получить навыки составление PE файла.

2.1 Краткие теоретические сведения

Portable Executable — формат исполняемых файлов, объектного кода и динамических библиотек, используемый в 32- и 64-разрядных версиях операционной системы Microsoft Windows. Формат PE представляет собой структуру данных, содержащую всю информацию, необходимую PE-загрузчику для отображения файла в память. Исполняемый код включает в себя ссылки для связывания динамически загружаемых библиотек, таблицы экспорта и импорта API-функций, данные для управления ресурсами и данные локальной памяти потока (TLS). В операционных системах семейства Windows NT формат PE используется для EXE, DLL, SYS (драйверов устройств) и других типов исполняемых файлов.

Hex-редактор — приложение для редактирования данных, в котором данные представлены в «сыром виде» — как последовательность байтов. Он может быть, как отдельным самостоятельным приложением, так и компонентом другого, более сложного приложения, такого как дизассемблер, отладчик, интегрированная среда разработки и т.п.

Ассемблер (от англ. assembler — сборщик) — транслятор программы из текста на языке ассемблера, в программу на машинном языке.

Машинный код, машинный язык — система команд (набор кодов операций) конкретной вычислительной машины, которая интерпретируется непосредственно процессором или микропрограммами этой вычислительной машины.

2.2 Пример выполнения лабораторной работы

1. Скачать HEX редактор (пример: https://mh-nexus.de/en/hxd).
2. Путем ввода HEX символов в HEX редакторе, составить свой PE файл.

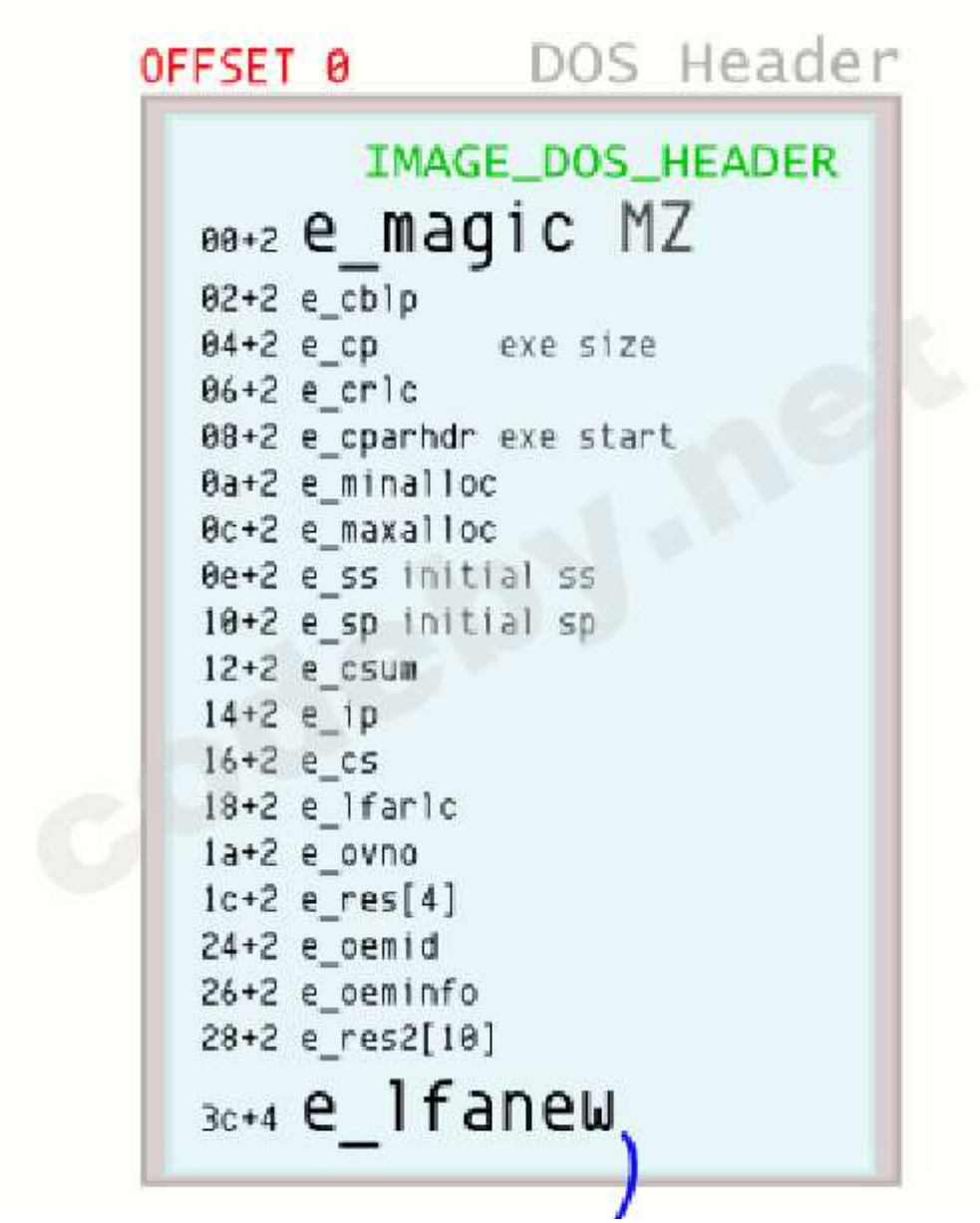
Для начала нужно определиться, что будет выполнять PE файл. Пусть наша программа будет бесконечно выводить приветственное окно с фамилией автора в заголовке окна. Для облегчения задачи наш PE файл будет 32х битным.

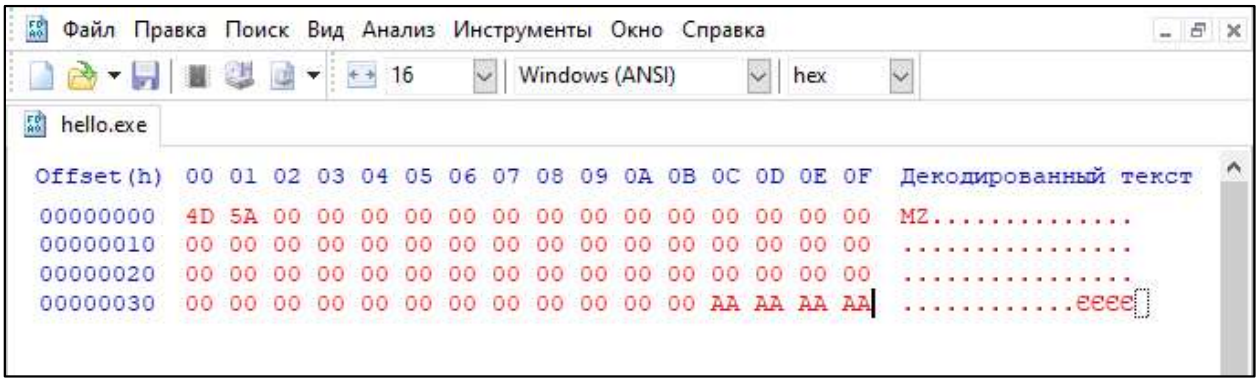
На языке C код программы выглядел бы так:

```
int main()
{
    while (true)
    {
        MessageBoxA(0, "Hello World!", "Полковников", 0);
    }
}
```

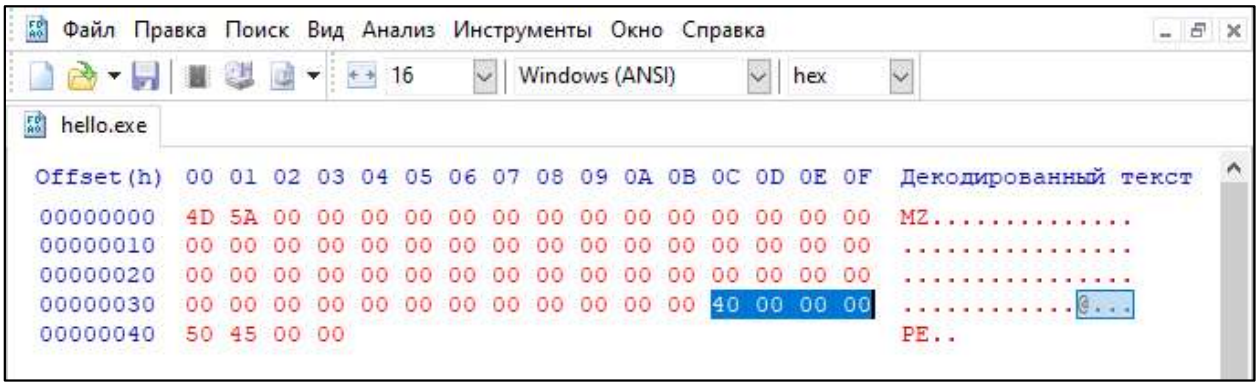
Создадим в HEX редакторе новый файл и начнем заполнять заголовки файла:

1. **DOS-заголовок**. Необходимыми полями для загрузки у DOS-заголовка являются, как мы знаем, e_magic и e_lfanew. Поле e_magic хранит в себе специальную сигнатуру MZ, а поле e_lfanew смещение до PE-заголовка. Заполним только эти поля в DOS заголовке, а остальные заполним нулями. Так как пока не знаем смещения до PE заголовка, заполним это поле байтами AA (Позже поменяем его на смещение до PE-заголовка).

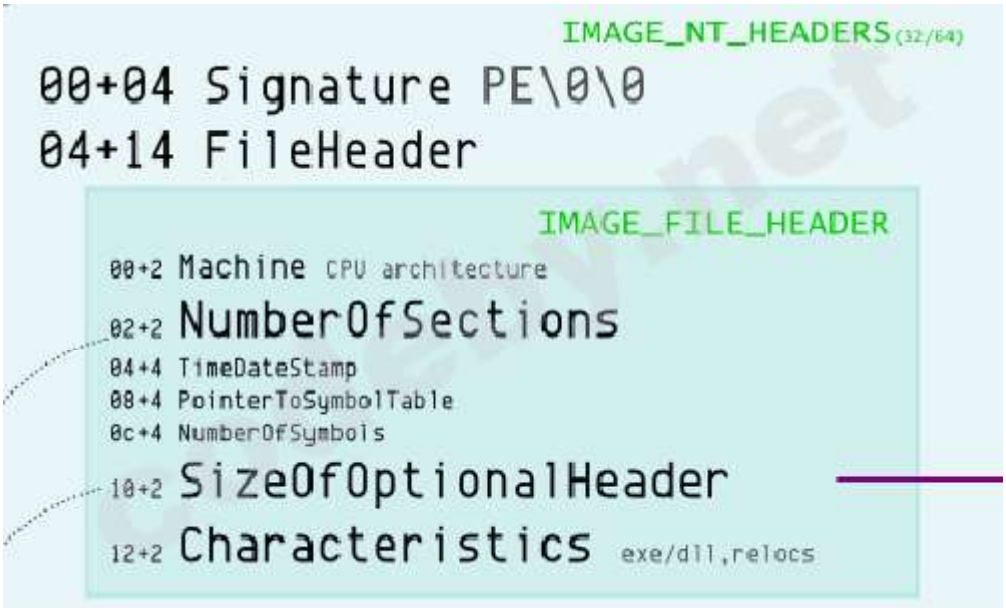




2. **PE-сигнатура.** Запишем сигнатуру PE файла ("PE\x00\x00") и изменим поле e_lfanew (смещение до PE-заголовка) в DOS-заголовке (смещением до PE-заголовка будет число 0x40).



3. **Файловый заголовок:**



a. **Архитектура процессора [2 байта].** Запишем значение, указывающее на 32 битную архитектуру процессора - 0x014c.

Таблица 2.1 – Возможные значения архитектуры процессора

Метка	Значение	Архитектура
IMAGE_FILE_MACHINE_I386	0x014c	x86
IMAGE_FILE_MACHINE_IA64	0x0200	Intel Itanium
IMAGE_FILE_MACHINE_AMD64	0x8664	x64

b. **Количество секций [2 байта].** У нас используется 3 секции.

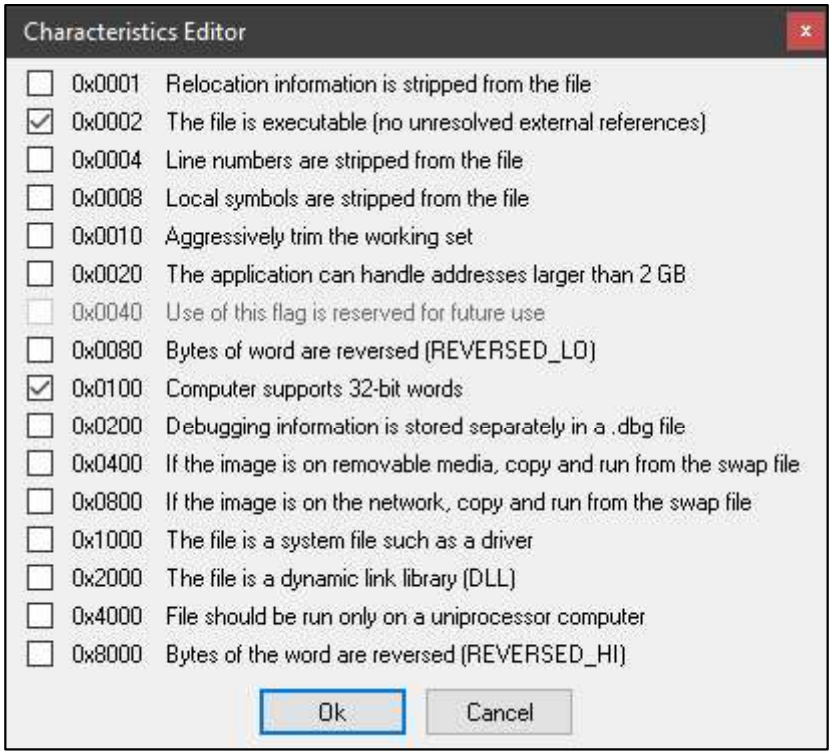
c. **Временная метка [4 байта].** Необязательные поле. Заполним NULL.

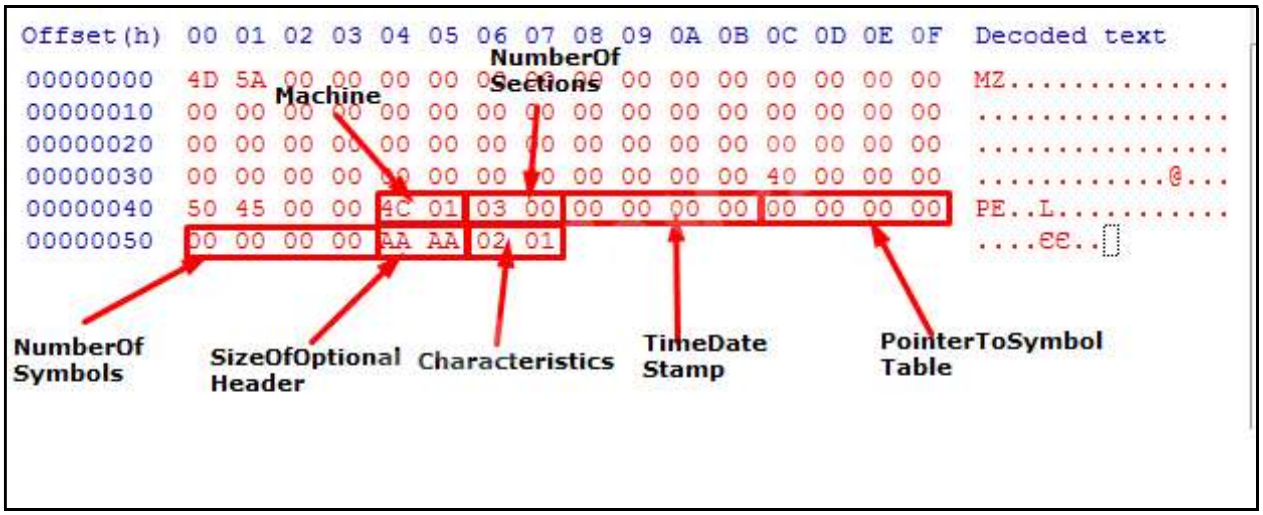
d. **PointerToSymbolTable [4 байта].** Необязательные поле. Заполним NULL.

e. **NumberOfSymbols [4 байта].** Необязательные поле. Заполним NULL.

f. **Размер дополнительного заголовка [2 байта].** Пока не знаем. Заполним AA.

g. **Характеристики файла [2 байта].** Запишем значение, указывающее на то, что это исполняемый 32 битный файл – 0x102.





4. Опциональный заголовок:

a. **Магия [2 байта].** Запишем значение, указывающее на 32 битность программы - 0x01b.

Таблица 2.2 – Возможные значения битности программы

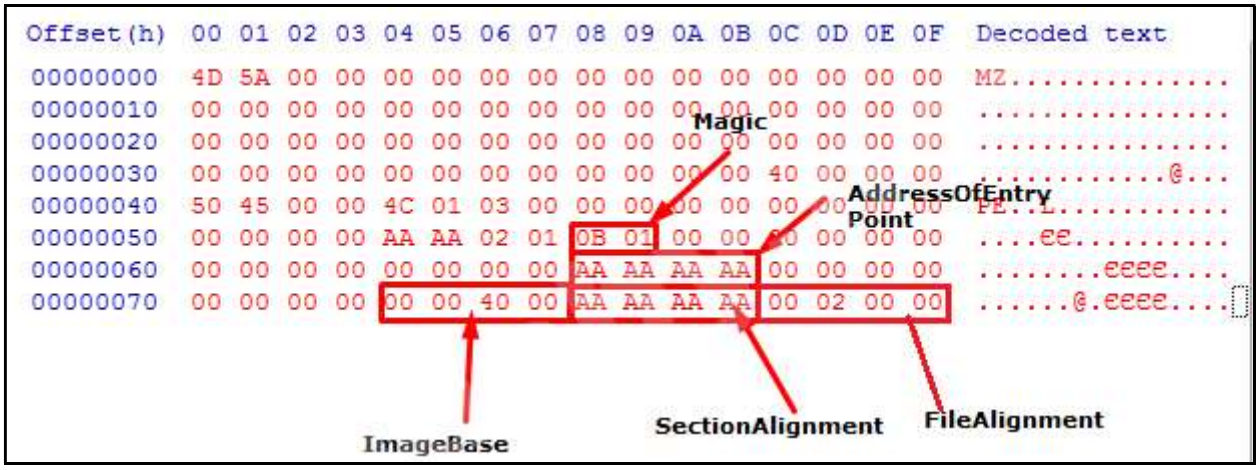
Метка	Значение	Битность
IMAGE_NT_OPTIONAL_HDR32_MAGIC	0x10b	32 бит
IMAGE_NT_OPTIONAL_HDR64_MAGIC	0x20b	64 бит
IMAGE_ROM_OPTIONAL_HDR_MAGIC	0x107	ROM

b. **Относительный виртуальный адрес точки [4 байта].** Пока не знаем. Заполним AA.

c. **ImageBase – предпочитаемый адрес базовой выгрузки файла [4 байта].** Запишем значение по умолчанию: 0x00400000.

d. **SectionAlignment – смещение начала заголовков программы в виртуальной памяти [4 байта].** Пока не знаем. Заполним AA.

e. **FileAlignment – смещение начала заголовков относительно начала файла [4 байта].** Может принимать значения $2^n, n \in 9,10,..,16$. По умолчанию 512 ($= 2^9$) или 0x200 в шестнадцатеричном виде.



f. **MajorSubsystemVersion [].** Запишем значение, указывающее Windows NT – 4.

g. **SizeOfImage – размер образа в памяти [].** Значение должно быть кратно значению SectionAlignment. Пока не знаем. Заполним AA.

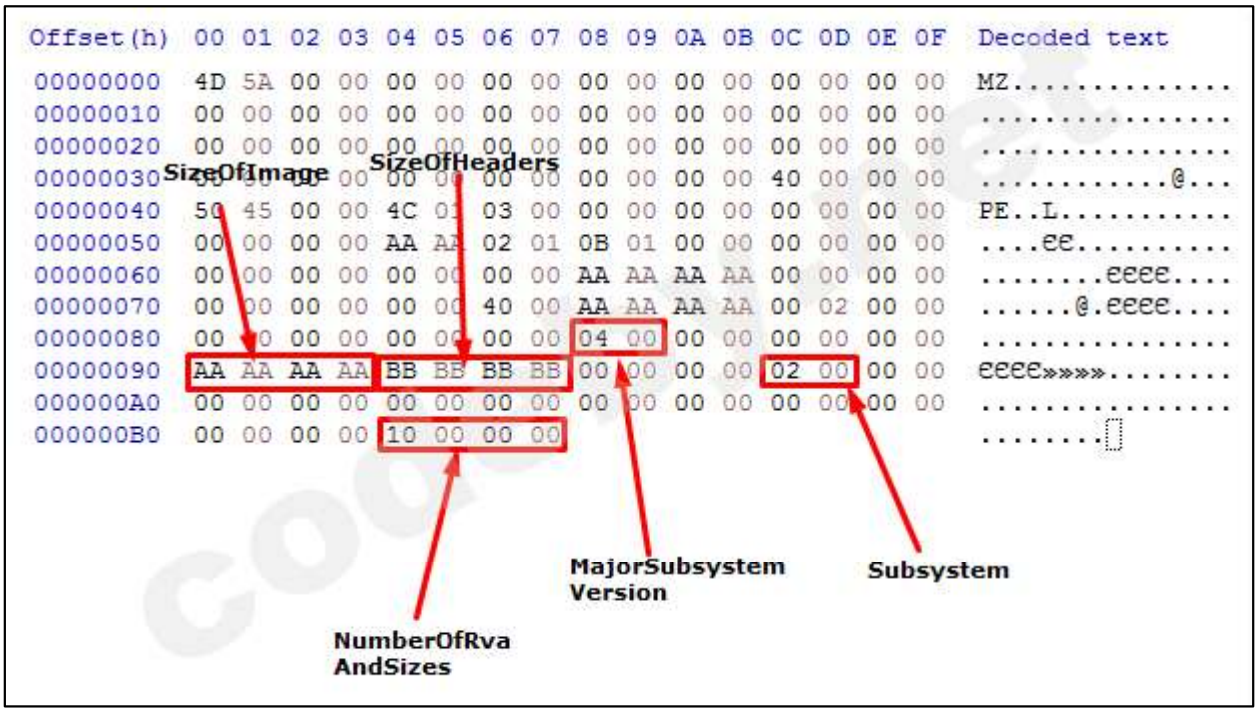
h. **SizeOfHeaders – размер всех заголовков [].** Значение должно быть кратно значению FileAlignment (или равно). Заполним BB.

i. **Subsystem – тип подсистемы [].** Укажем 2, так как наша программа является графической программой Windows.

IMAGE_SUBSYSTEM_UNKNOWN 0	Unknown subsystem.
IMAGE_SUBSYSTEM_NATIVE 1	No subsystem required (device drivers and native system processes).
IMAGE_SUBSYSTEM_WINDOWS_GUI 2	Windows graphical user interface (GUI) subsystem.
IMAGE_SUBSYSTEM_WINDOWS_CUI 3	Windows character-mode user interface (CUI) subsystem.
IMAGE_SUBSYSTEM_OS2_CUI 5	OS/2 CUI subsystem.
IMAGE_SUBSYSTEM_POSIX_CUI 7	POSIX CUI subsystem.
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI 9	Windows CE system.
IMAGE_SUBSYSTEM_EFI_APPLICATION 10	Extensible Firmware Interface (EFI) application.
IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER 11	EFI driver with boot services.
IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER 12	EFI driver with run-time services.

j. NumberOfRvaAndSizes – число элементов в массиве DataDirectory [].

По умолчанию 16 (0x10).



k. **DataDirectory – массив элементов VirtualAddress и Size.** Из всех этих индексов нас интересует только один - IMAGE_DIRECTORY_ENTRY_IMPORT. Элемент с этим индексом содержит информацию о таблице импорта. Так нас интересует только элемент с данным индексом, мы заполним только его, а остальные заполним NULL байтами. Так как мы не определились с адресом таблицы импорта, заполним его байтами AA.

Структура каждого элемента:

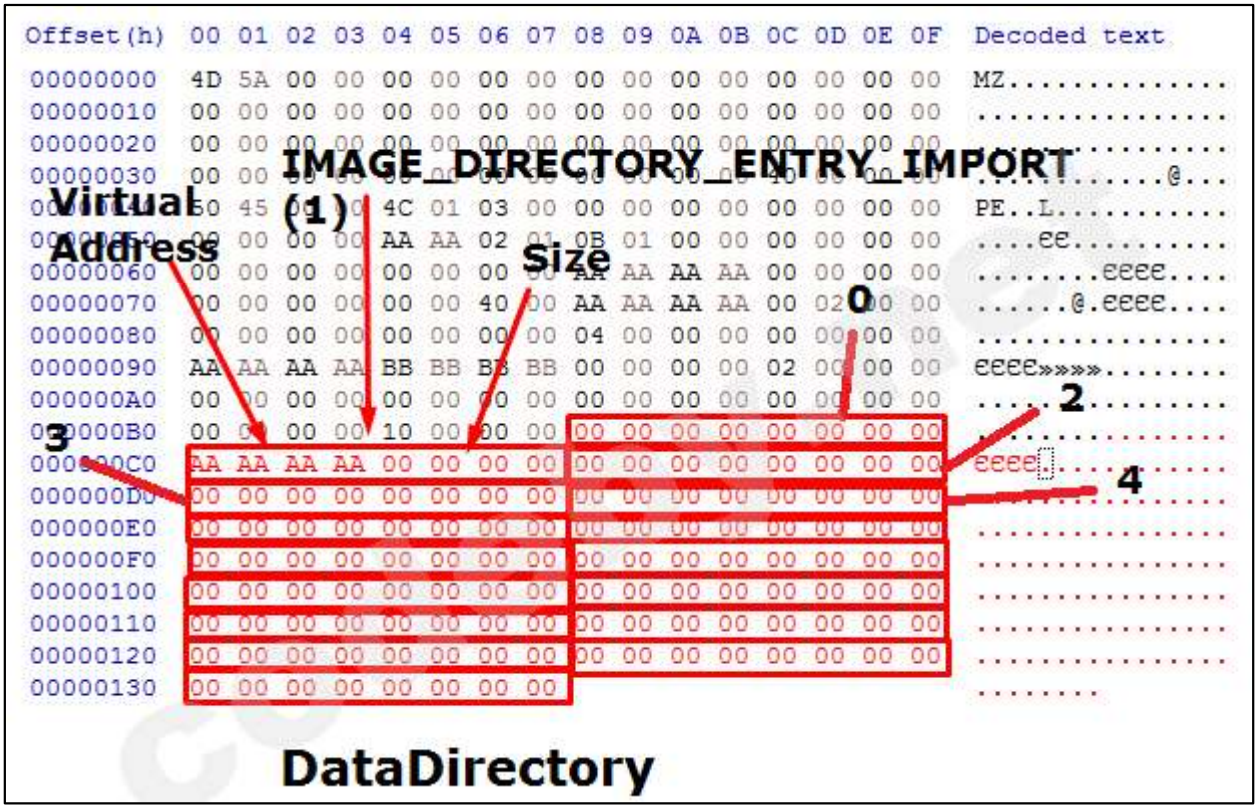
```
typedef struct _IMAGE_DATA_DIRECTORY {
  DWORD VirtualAddress;
  DWORD Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

VirtualAddress – адрес, куда будет выгружен секция**;**

Size – размер секции**.**

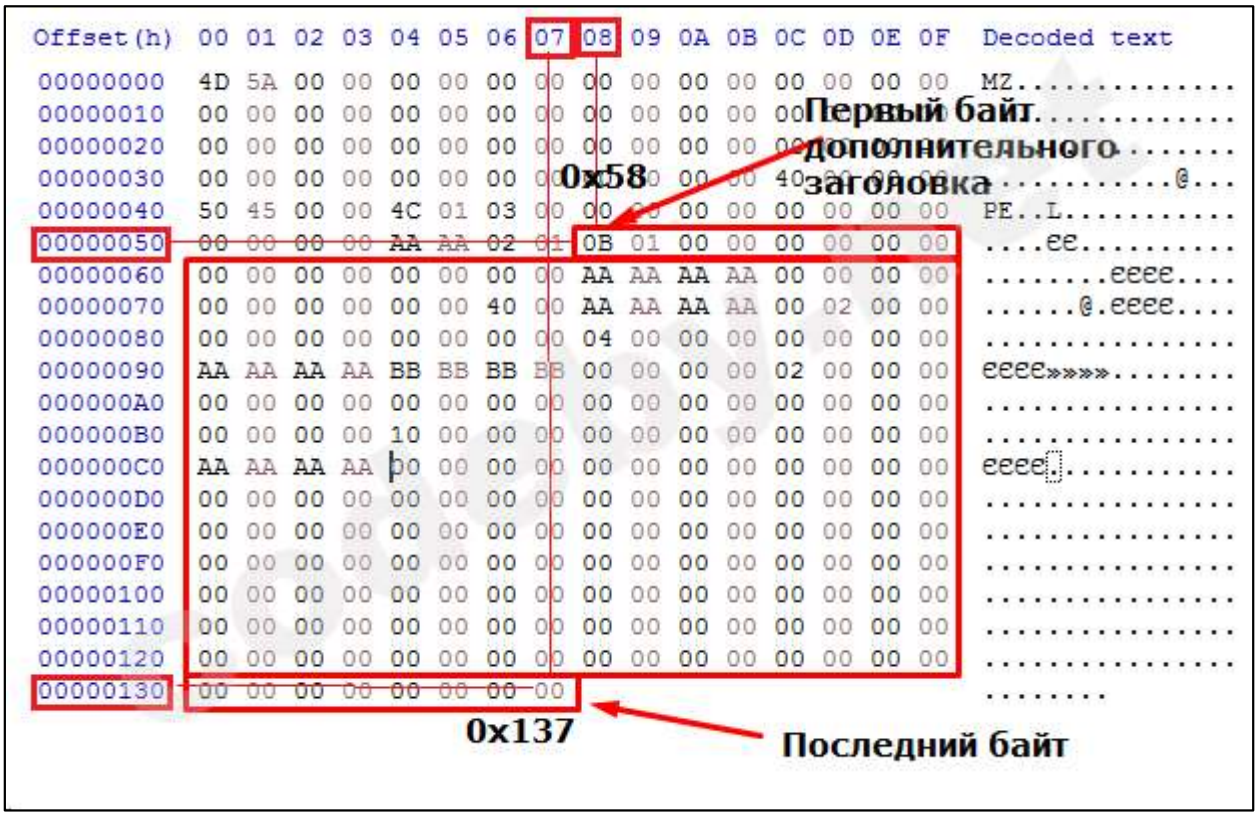
Индексы элементов

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT          0   // Export Directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT              1   // Import Directory
#define IMAGE_DIRECTORY_ENTRY_RESOURCE        2   // Resource Directory
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION       3   // Exception Directory
#define IMAGE_DIRECTORY_ENTRY_SECURITY        4   // Security Directory
#define IMAGE_DIRECTORY_ENTRY_BASERELOC       5   // Base Relocation Table
#define IMAGE_DIRECTORY_ENTRY_DEBUG               6   // Debug Directory
//      IMAGE_DIRECTORY_ENTRY_COPYRIGHT           7   // (X86 usage)
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE    7   // Architecture Specific Data
#define IMAGE_DIRECTORY_ENTRY_GLOBALPTR       8   // RVA of GP
#define IMAGE_DIRECTORY_ENTRY_TLS             9   // TLS Directory
#define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG     10  // Load Configuration Directory
#define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT    11  // Bound Import Directory in headers
#define IMAGE_DIRECTORY_ENTRY_IAT             12  // Import Address Table
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT    13  // Delay Load Import Descriptors
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR  14  // COM Runtime descriptor
```

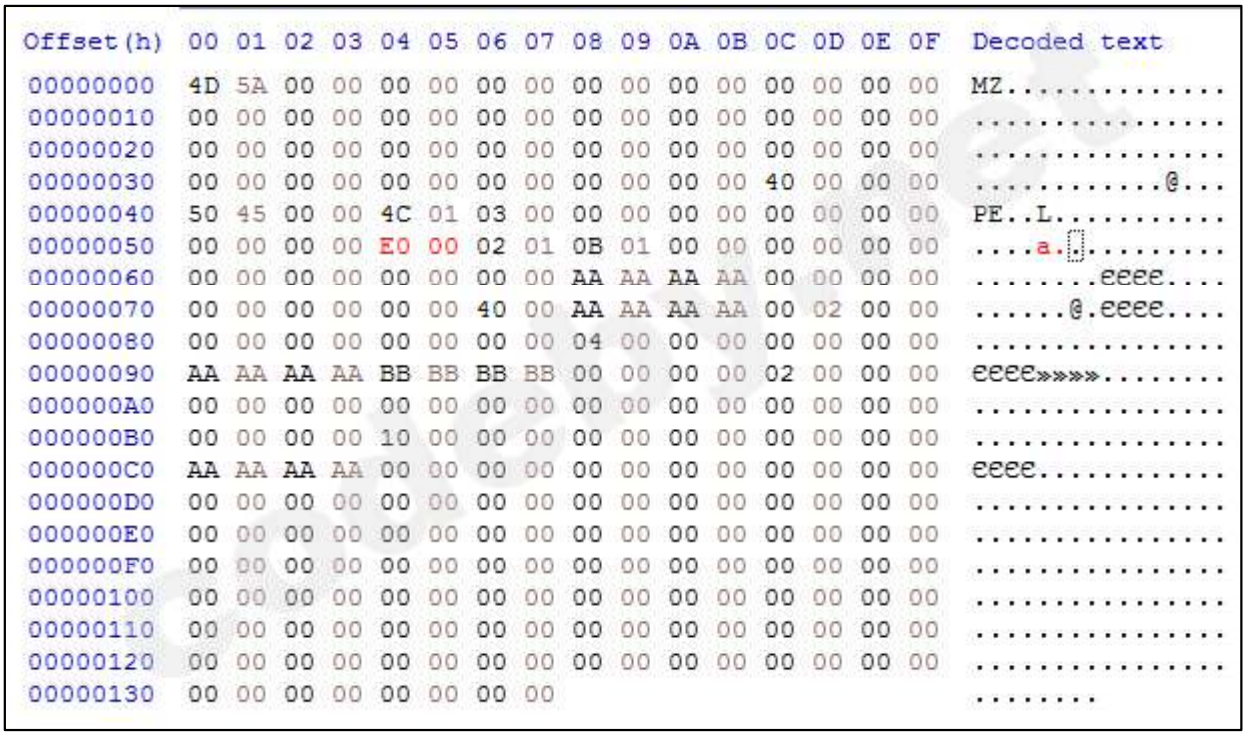


После заполнения опционального заголовка можно узнать какой у него размер (в байтах) и занести это значение в байтовый заголовок. Для того, чтобы узнать его размер, получим разность смещения первого байта заголовка и последнего и прибавим единицу:

$$0x137 - 0x58 + 0x1 = 0xE0$$



Получившееся значение и запишем в поле SizeOfOptionalHeader файлового заголовка.



5. Таблица секций – массив элементов IMAGE_SECTION_HEADER.

Структура каждого IMAGE_SECTION_HEADER

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  DWORD VirtualSize;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Name [8 байт] – имя секции**;**

VirtualSize – размер секции в виртуальной памяти**;**

VirtualAddress – относительный виртуальный адрес секции (равен сумме VirtualAddress и VirtualSize прошлого элемента)**;**

SizeOfRawData – размер секции в файле (должно быть кратно FileAlignment);

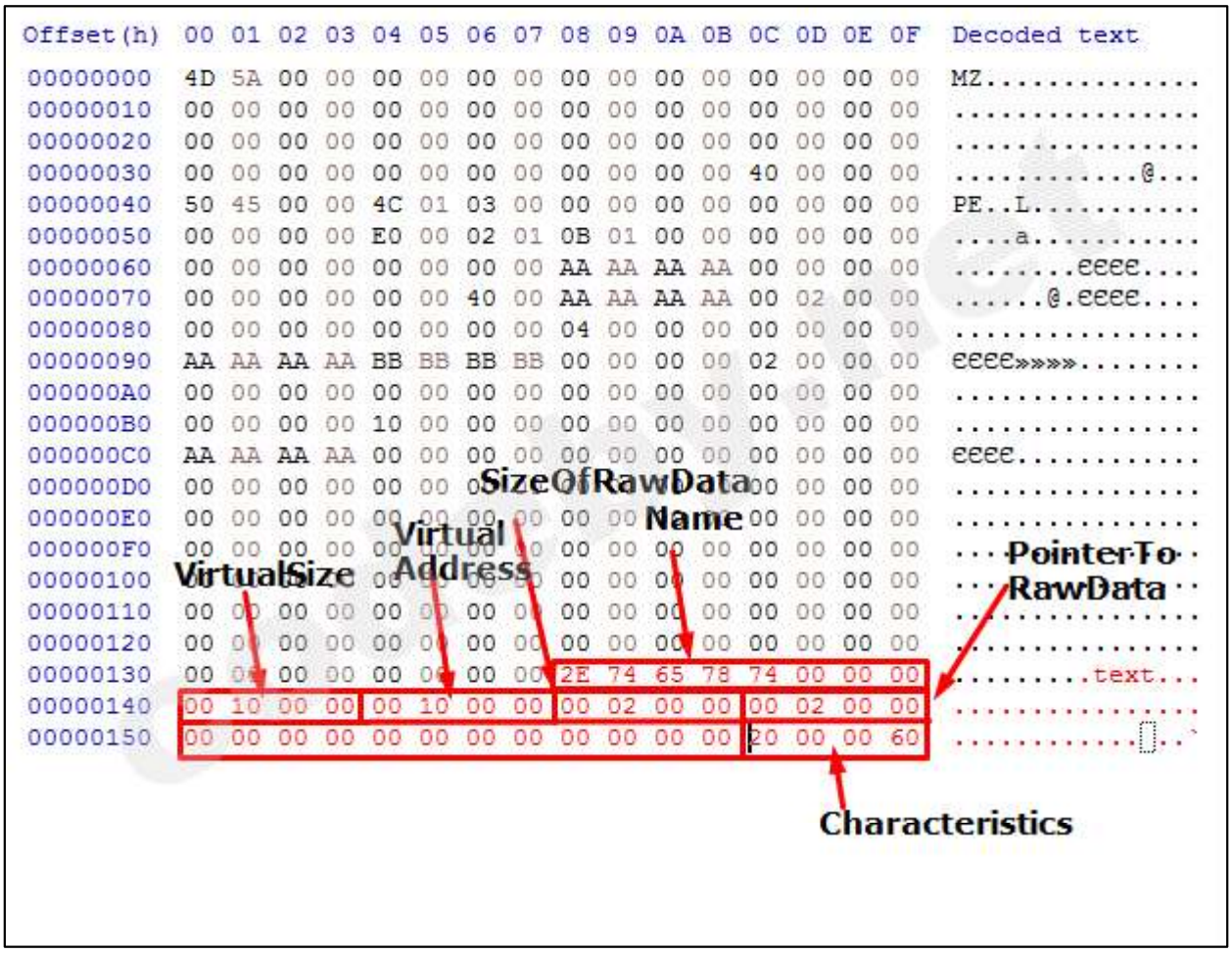
PointerToRawData – адрес секции в файле (должно быть кратно FileAlignment) (равен сумме PointerToRawData и SizeOfRawData прошлого элемента);

Characteristics – характеристики.

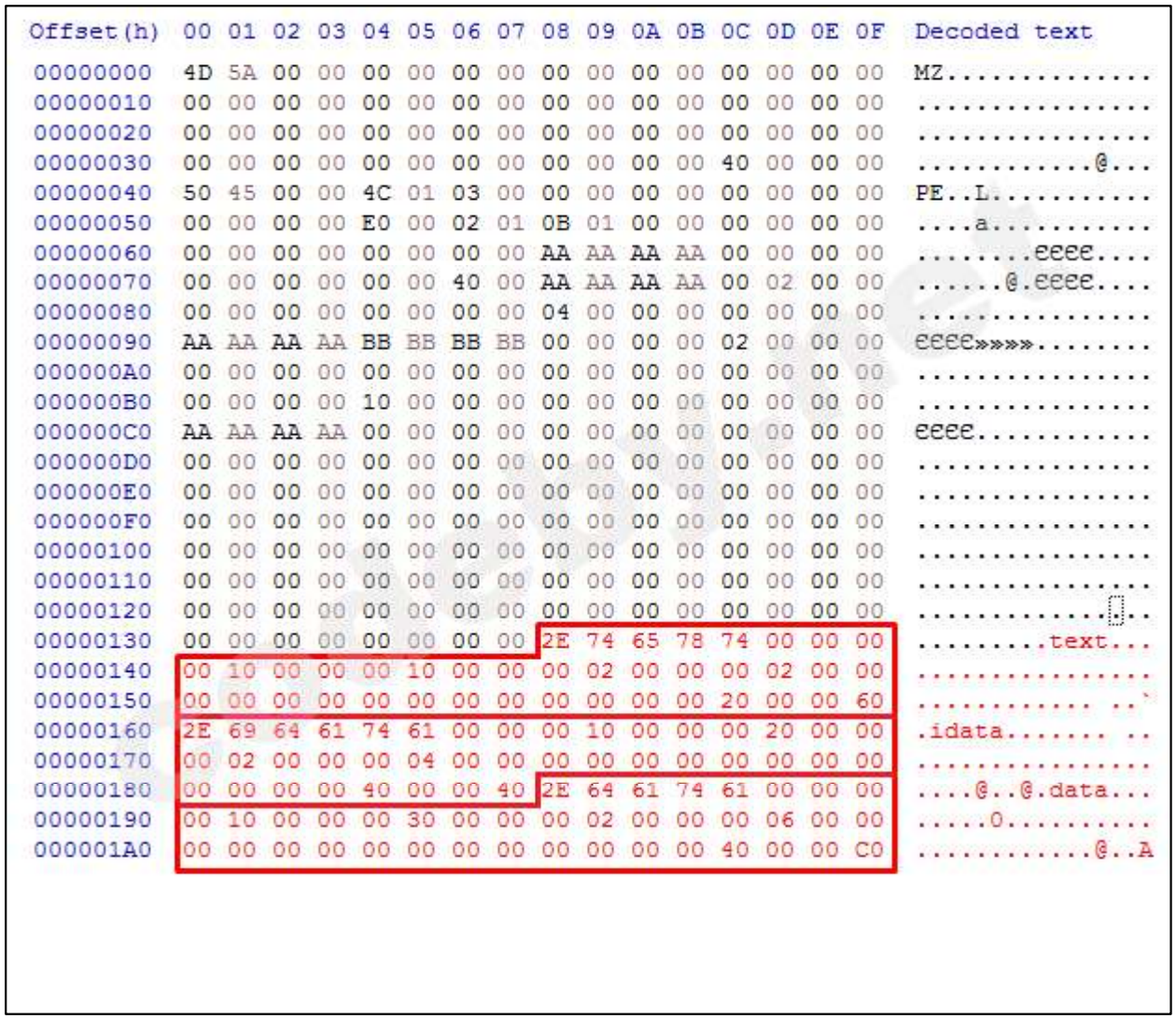
Запишем:

```
Name                = .text
VirtualSize         = 0x1000
VirtualAddress      = 0x1000
SizeOfRawData       = 0x200
PointerToRawData    = 0x200
Characteristics     = 0x60000020
```

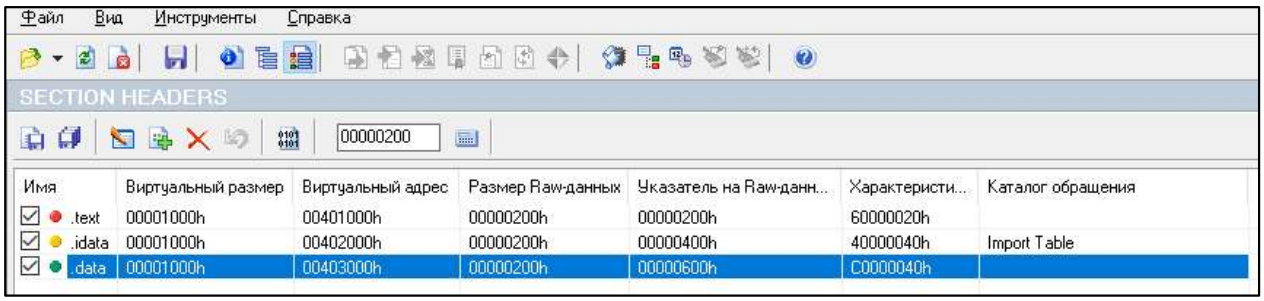
Остальные поля null, они не столь важны.



Аналогично заполним секции «.idata» и «.data».



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	MZ..............
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	00	............@...
00000040	50	45	00	00	4C	01	03	00	00	00	00	00	00	00	00	00	PE..L...........
00000050	00	00	00	00	E0	00	02	01	0B	01	00	00	00	00	00	00	....a...........
00000060	00	00	00	00	00	00	00	00	AA	AA	AA	AA	00	00	00	00	........eeee....
00000070	00	00	00	00	00	00	40	00	AA	AA	AA	AA	00	02	00	00	......@.eeee....
00000080	00	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00	....
00000090	AA	AA	AA	AA	BB	BB	BB	BB	00	00	00	00	02	00	00	00	eeee»»»»........
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000B0	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00	....
000000C0	AA	AA	AA	AA	00	00	00	00	00	00	00	00	00	00	00	00	eeee............
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..............[].
00000130	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00	.........text...
00000140	00	10	00	00	00	10	00	00	00	02	00	00	00	02	00	00	....
00000150	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60	....`
00000160	2E	69	64	61	74	61	00	00	00	10	00	00	00	20	00	00	.idata....... ..
00000170	00	02	00	00	00	04	00	00	00	00	00	00	00	00	00	00	....
00000180	00	00	00	00	40	00	00	40	2E	64	61	74	61	00	00	00	....@..@.data...
00000190	00	10	00	00	00	30	00	00	00	02	00	00	00	06	00	00	.....0..........
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	............@..A



Имя	Виртуальный размер	Виртуальный адрес	Размер Raw-данных	Указатель на Raw-данн...	Характеристи...	Каталог обращения
.text	00001000h	00401000h	00000200h	00000200h	60000020h	
.idata	00001000h	00402000h	00000200h	00000400h	40000040h	Import Table
.data	00001000h	00403000h	00000200h	00000600h	C0000040h	

Теперь, когда мы знаем виртуальный адрес (0x1000) загрузки секции кода (.text) в виртуальную память, мы можем изменить значения AddressOfEntryPoint, SectionAlignment, SizeOfImage, SizeOfHeaders, а также виртуальный адрес таблицы импорта.

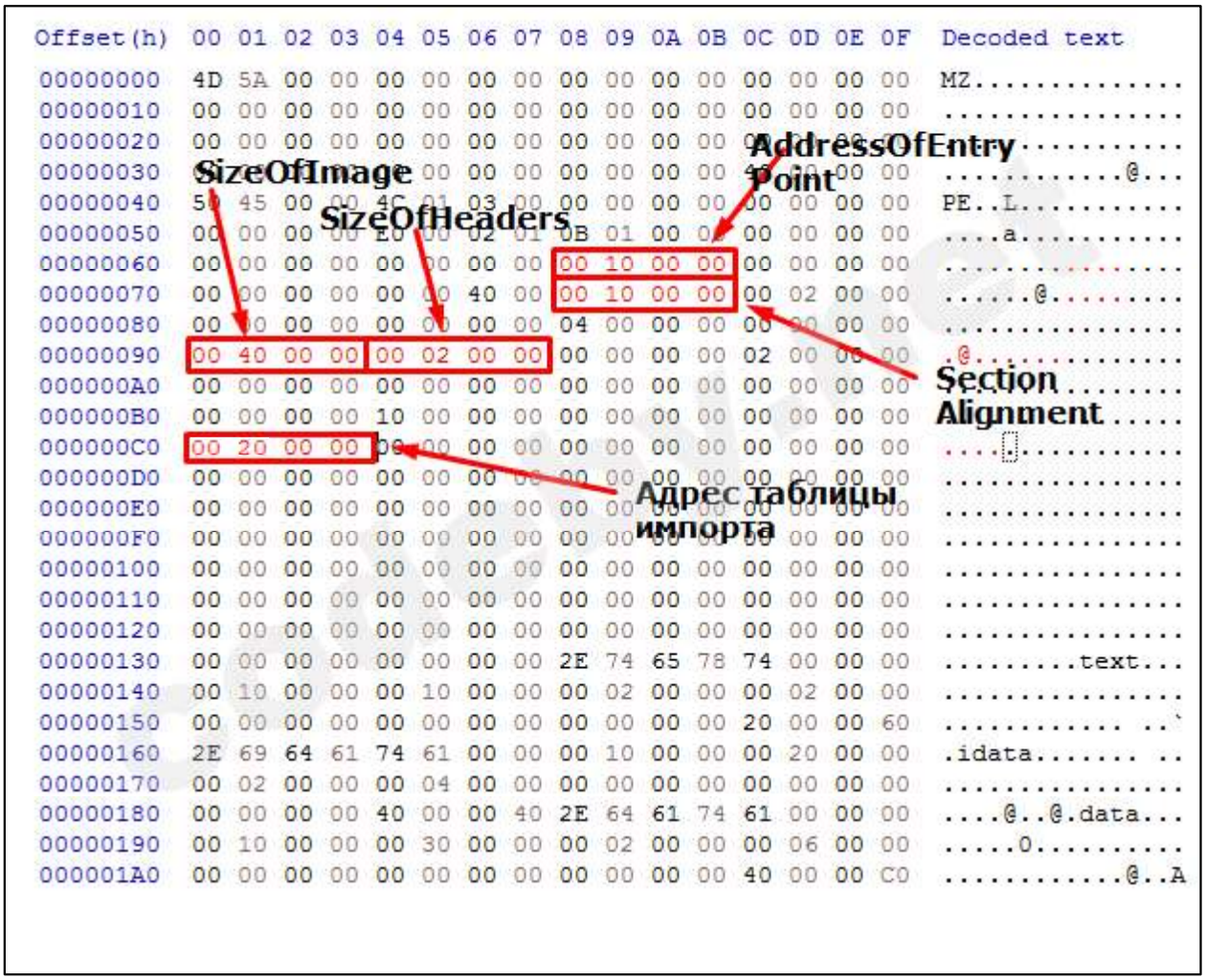
AddressOfEntryPoint будет равен 0x1000, так как секция кода загрузится по этому адресу, а в секции кода будет находится наш код. (Поле AddressOfEntryPoint хранит адрес точки входа в программу).

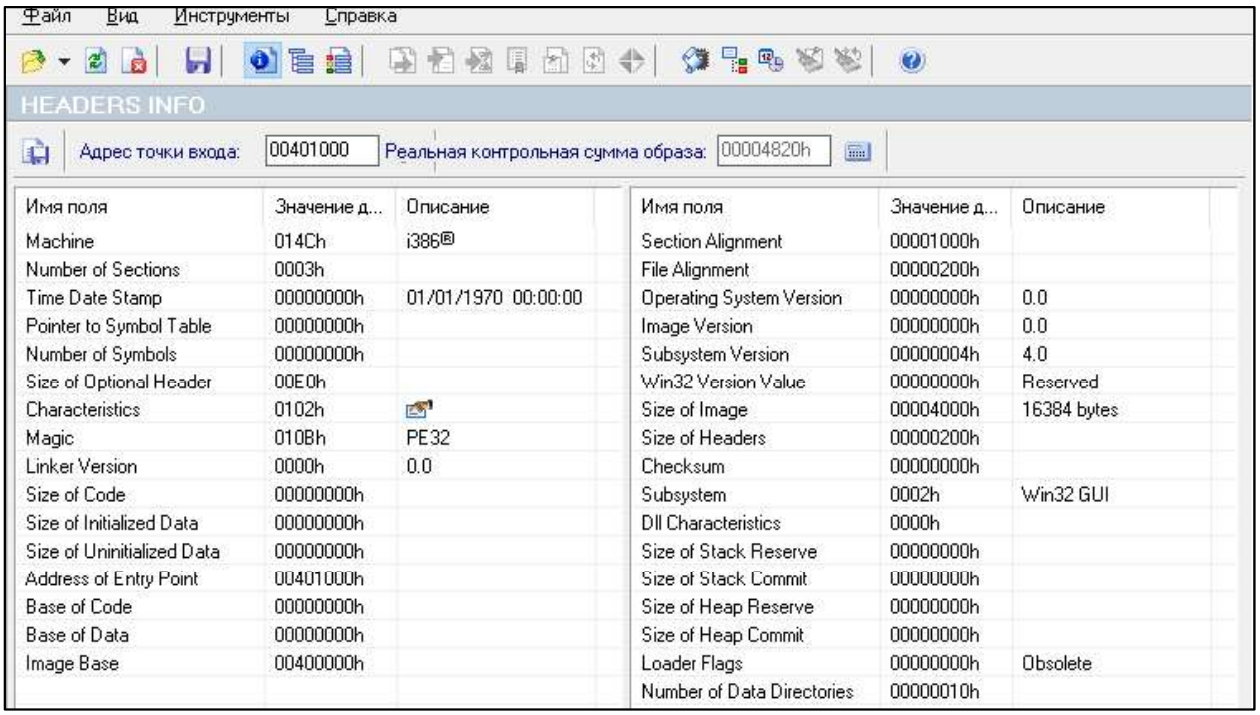
SectionAlignment будет также равен 0x1000, так как секция кода является первой, и она будет загружена по этому адресу (Поле SectionAlignment хранит адрес начала секций в виртуальной памяти).

Условимся, что SizeOfImage будет равен 0x4000, так как размер файла не будет превышать данного значения. (Потому что последняя секция файла будет находиться по смещению 0x600 и иметь размер 0x800) (Поле SizeOfImage хранит размер виртуального образа).

Поле SizeOfHeaders будет равен 0x200, так как секции в файле будут начинаться именно с этого значения, а как мы помним секции следуют сразу после заголовков (Поле SizeOfHeaders хранит размер всех заголовков (в байтах)).

Адрес таблицы импорта будет равен 0x2000 (VirtualAddress таблицы .idata), так как таблица импорта загрузится именно по этому адресу.

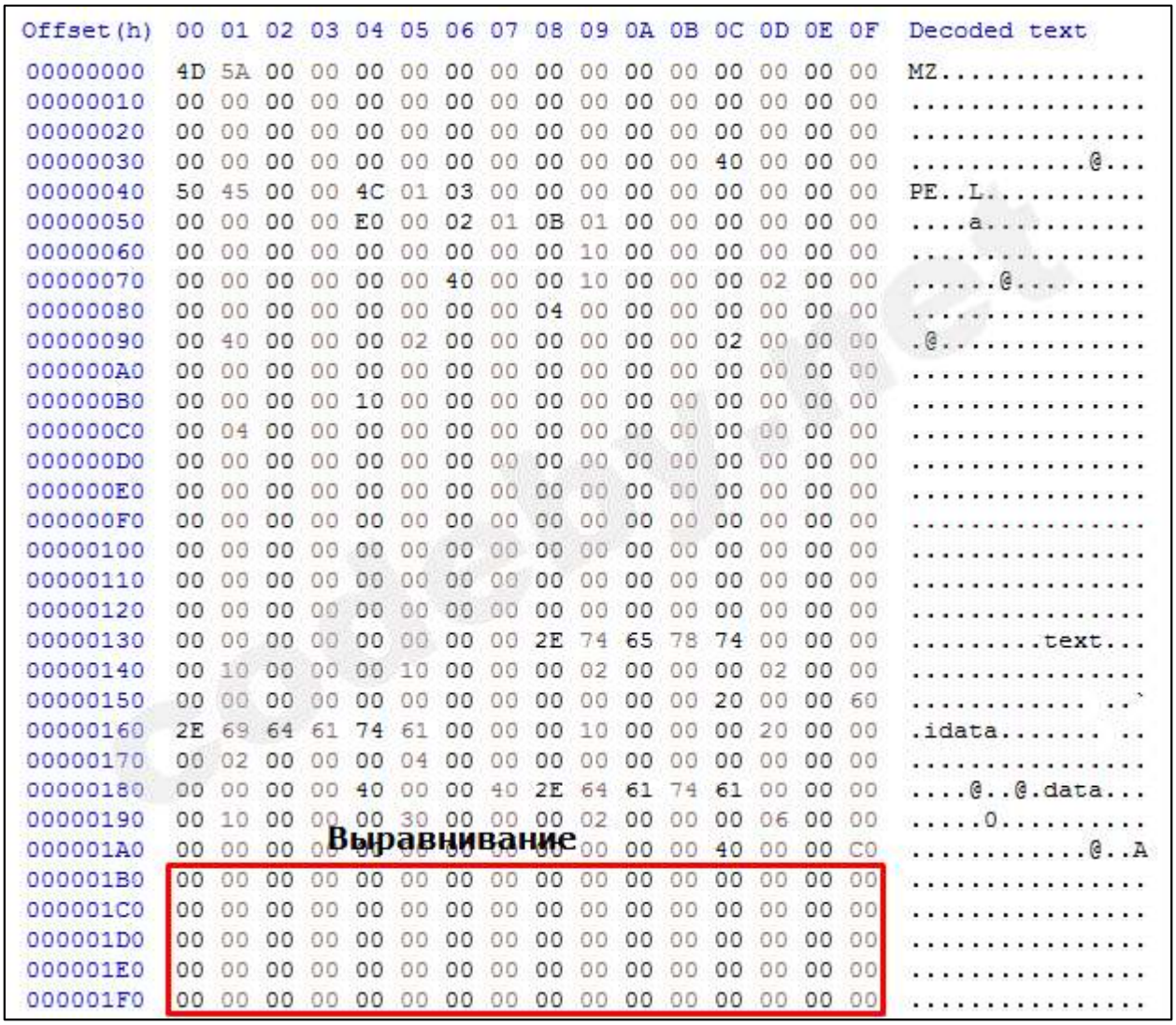




6. Секция кода.

Каждая секция в файле будет занимать по 0x200 (512) байт, а в памяти по 0x1000 (4096) байт. В файле они будут начинаться со смещения 0x200 (512), а в памяти по смещению 0x1000 (4096).

Пробел между концом заголовка и началом секций заполним null.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	MZ..............
00000010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	00	............@...
00000040	50	45	00	00	4C	01	03	00	00	00	00	00	00	00	00	00	PE..L...........
00000050	00	00	00	00	E0	00	02	01	0B	01	00	00	00	00	00	00	....a...........
00000060	00	00	00	00	00	00	00	00	00	10	00	00	00	00	00	00	....
00000070	00	00	00	00	00	00	40	00	00	10	00	00	00	02	00	00	......@.........
00000080	00	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00	....
00000090	00	40	00	00	00	02	00	00	00	00	00	00	02	00	00	00	.@..............
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000B0	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00	....
000000C0	00	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000130	00	00	00	00	00	00	00	00	2E	74	65	78	74	00	00	00	.........text...
00000140	00	10	00	00	00	10	00	00	00	02	00	00	00	02	00	00	....
00000150	00	00	00	00	00	00	00	00	00	00	00	00	20	00	00	60	....`
00000160	2E	69	64	61	74	61	00	00	00	10	00	00	00	20	00	00	.idata....... ..
00000170	00	02	00	00	00	04	00	00	00	00	00	00	00	00	00	00	....
00000180	00	00	00	00	40	00	00	40	2E	64	61	74	61	00	00	00	....@..@.data...
00000190	00	10	00	00	00	30	00	00	00	02	00	00	00	06	00	00	.....0..........
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	40	00	00	C0	............@..A
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Как мы знаем, для показа сообщения у нас есть функция MessageBoxA. Она будет загружена загрузчиком при разборе секции импорта, а её адрес будет помещён в определённое место в памяти.

Мы пока не знаем, что это за адрес, поэтому давайте пока представим, что адрес функции будет загружен по адресу `0xAAAAAAAA` (в дальнейшем мы его заменим на настоящий).

Также, мы не знаем адрес строк, которые должны быть отображены в сообщении ("Hello World!" и "Полковников"). Поэтому также представим, что они будут находится по адресу `0xAAAAAAAA`. На самом деле эти строки будут находится в секции данных.

На языке ассемблера наша программа будет выглядеть так

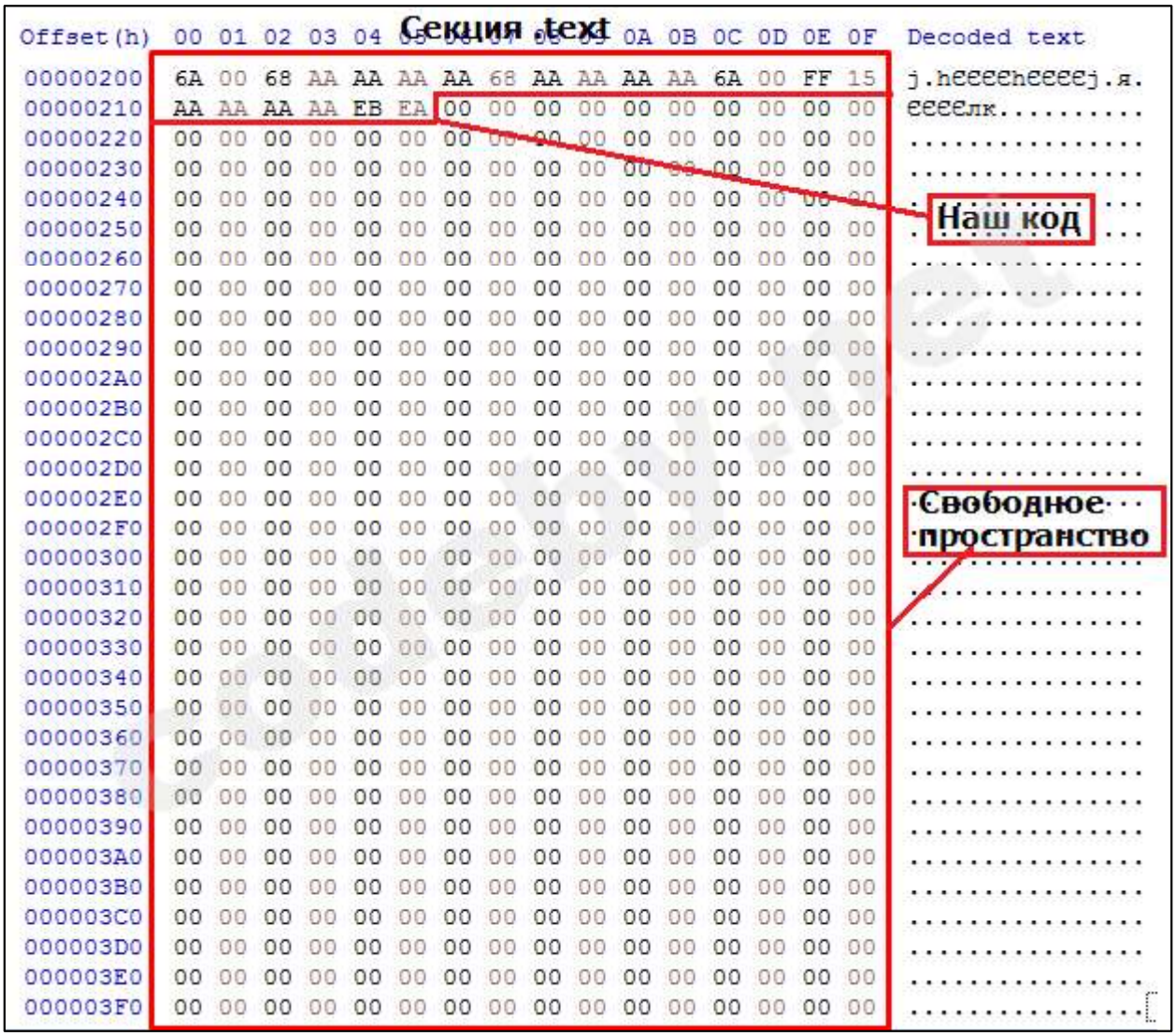
```
show_message:
    ; Передаём параметры
    push 0              ; MB_OK
    push 0xAAAAAAAA     ; Полковников
    push 0xAAAAAAAA     ; Hello World!
    push 0              ; NULL

    ; Вызываем функцию
    call [0xAAAAAAAA] ; MessageBoxA

    ; Переходим на шаг 1
    jmp  show_message
```

Когда параметры передаются через стэк (с помощью инструкций push), то они передаются в обратном порядке.

Преобразуем наш код на языке ассемблера в байты (Можно воспользоваться сервисом: https://defuse.ca/online-x86-assembler.htm#disassembly) и запишем в нашу секцию.



7. Секция импорта.

Таблица импорта - неотъемлемая часть любой программы, использующая функции из динамически подключаемых библиотек (DLL). При загрузке PE-файла, загрузчик разбирает эту таблицу на части, загружает нужные нам библиотеки и предоставляет адреса на функции, которые мы импортируем.

Таблица импорта - массив элементов IMAGE_IMPORT_DESCRIPTOR. Таблица импорта заканчивается нулевым элементом, т.е. элементом IMAGE_IMPORT_DESCRIPTOR, у которого все поля равны нулю.

Структура элемента IMAGE_IMPORT_DESCRIPTOR на языке C

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
         DWORD OriginalFirstThunk;
         DWORD TimeDateStamp;
         DWORD ForwarderChain;
         DWORD Name;
         DWORD FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR, *PIMAGE_IMPORT_DESCRIPTOR;
```

OriginalFirstThunk – хранит относительный виртуальный адрес указателя на элемент таблицы Import Lookup Table - IMAGE_IMPORT_BY_NAME.

Name – указатель на строку с именем DLL.

FirstThunk – виртуальный адрес указатель на таблицу Import Address Table. В эту таблицу загрузчик помещает адрес импортируемой функции.

Структура элемента IMAGE_IMPORT_BY_NAME на языке C

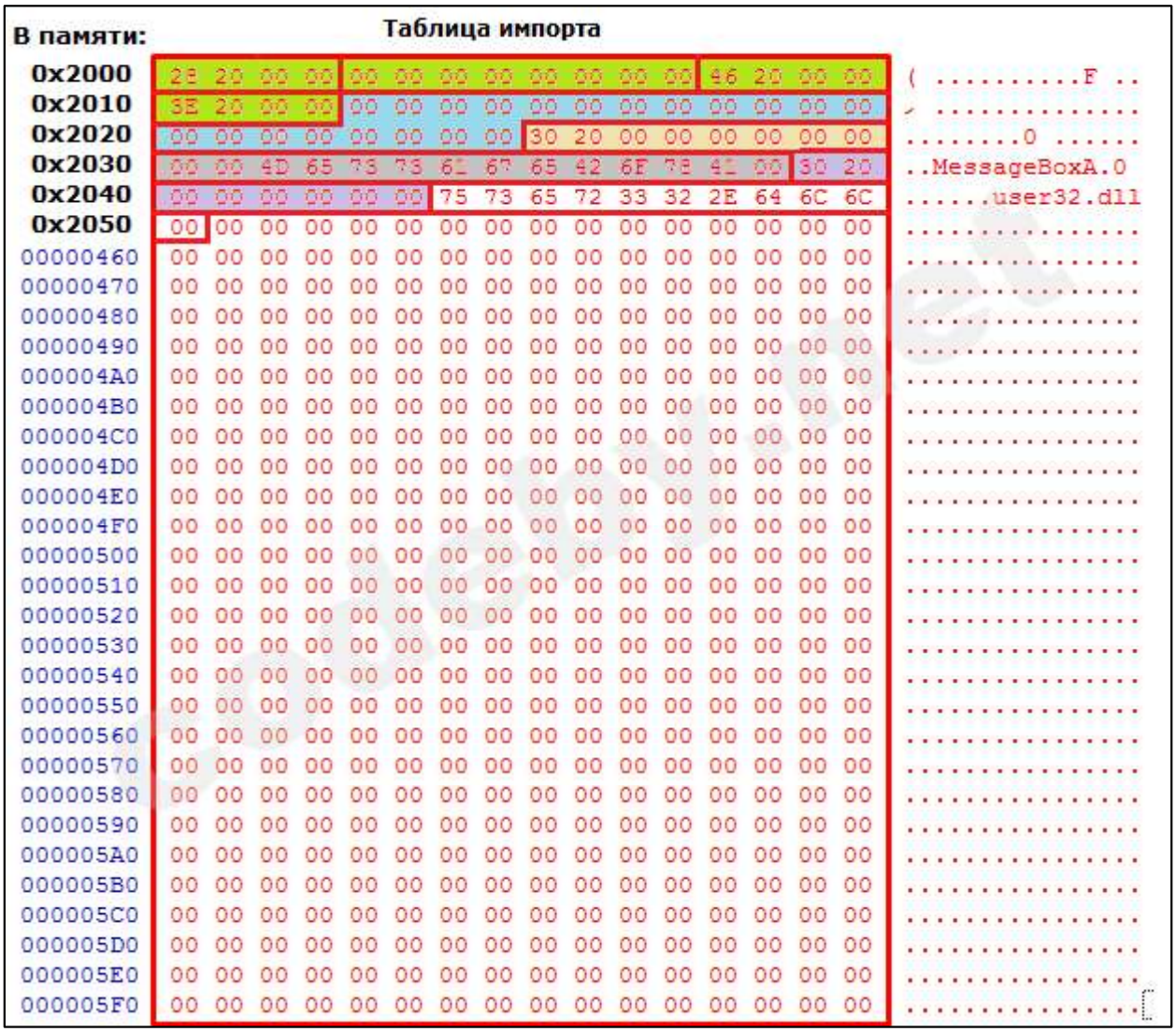
```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD Hint;
    CHAR[] Name;
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

Hint – индекс функции в LDD. Может быть равно нулю.

Name – имя функции. Заканчивается null символом.

Мы будем импортировать только одну функцию - MessageBoxA. Она находится в user32.dll. Поэтому у нас будет только 2 элемента в таблице импорта. Первый - отвечающий за функцию MessageBoxA, а второй - нулевой.

Заполним таблицу секций.



Зелёный элемент - элемент IMAGE_IMPORT_DESCRIPTOR. Его первое поле, как мы можем заметить, содержит виртуальный адрес (0x2028) на адрес (0x2030) (бежевый элемент), который указывает на элемент IMPORT_BY_NAME (серый элемент). Также, он хранит адрес (0x2046) на имя DLL'ки (белое поле) и адрес на Import Address Table (сереневое поле).

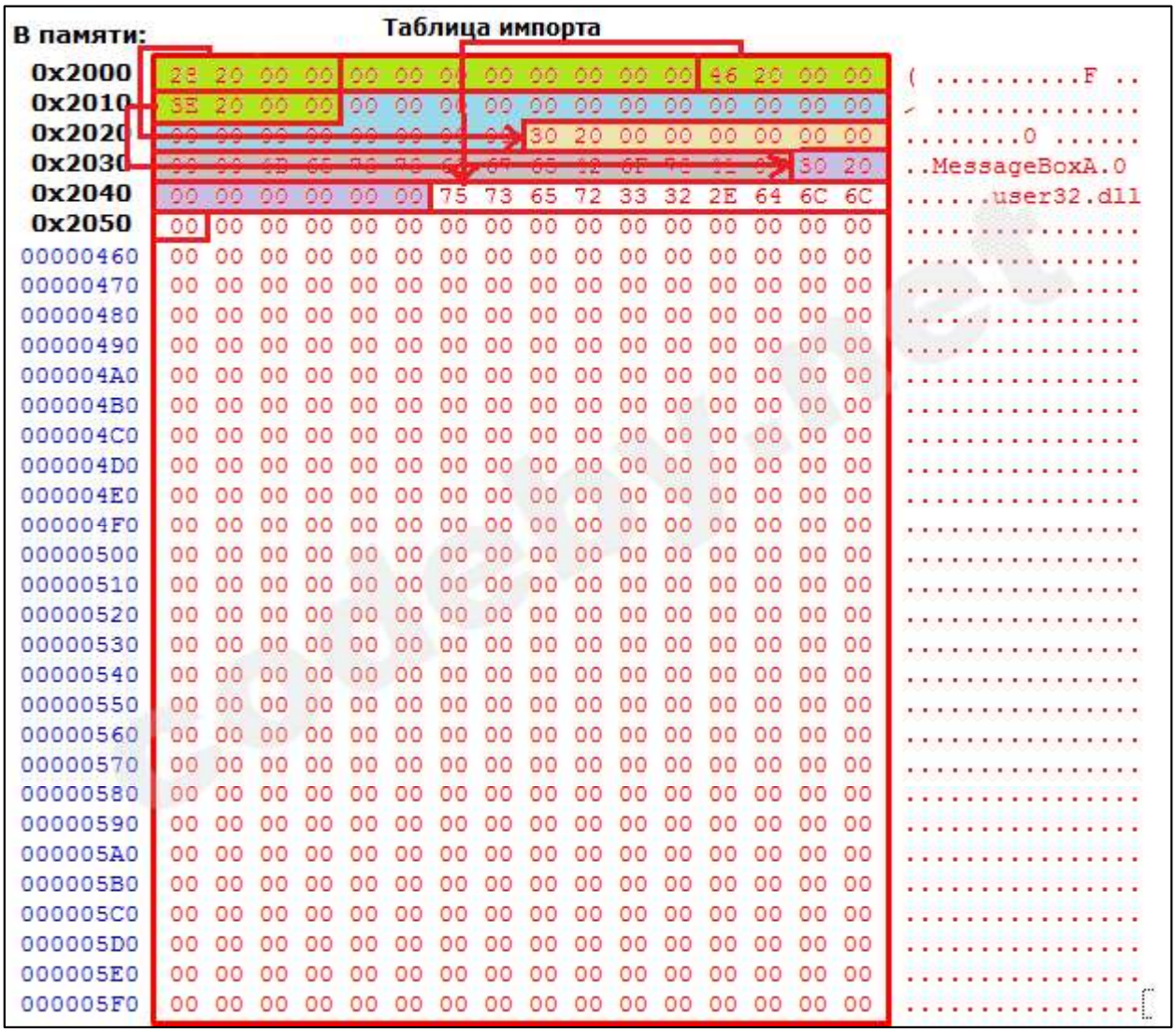
Голубой элемент - нулевой элемент таблицы дескрипторов.

Бежевый элемент - таблица указателей на элементы IMPORT_BY_NAME (серое поле).

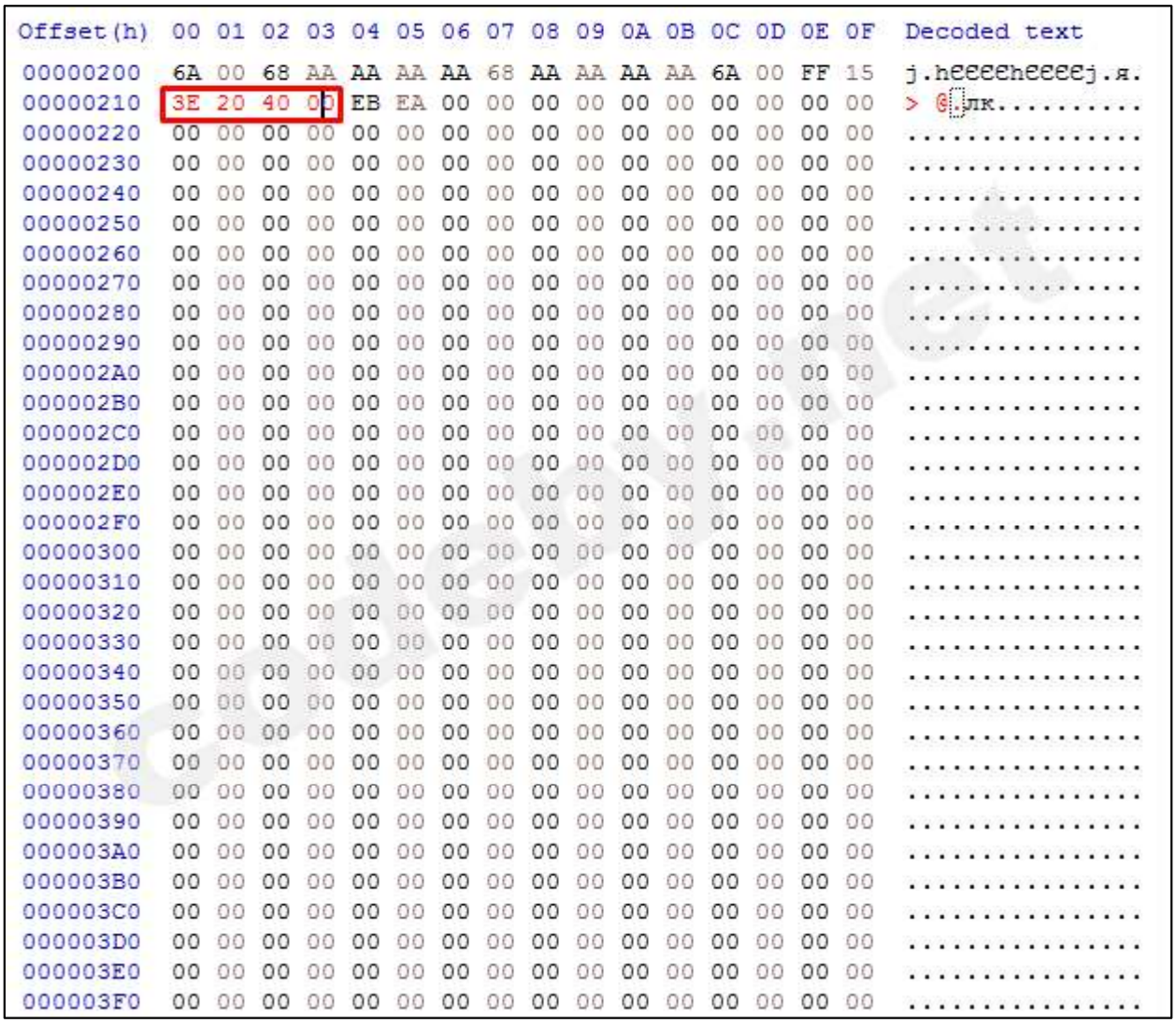
Серый элемент - элемент IMPORT_BY_NAME. Он хранит индекс и имя функции в DLL.

Сиреневый элемент - таблица Import Address Table. В неё будут адресы импортируемых функций.

Белый элемент - имя DLL.



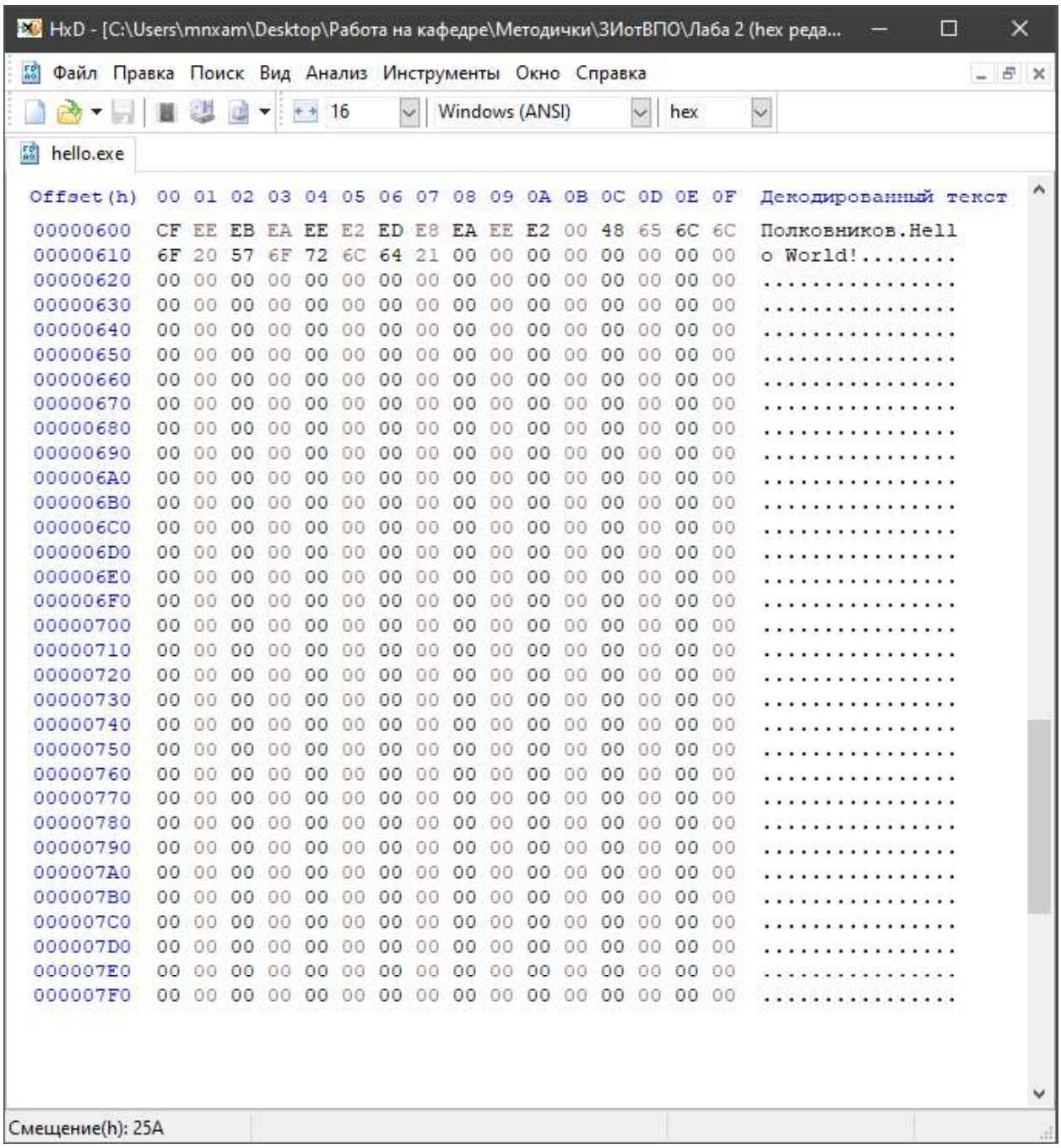
Теперь мы знаем, куда загрузится адрес нашей функции. Он загрузится по адресу 0x0040203E (ImageBase + относительный виртуальный адрес Import Address Table). Поэтому заменим его значение в секции кода.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000200	6A	00	68	AA	AA	AA	AA	68	AA	AA	AA	AA	6A	00	FF	15	j.hªªªªhªªªªj.ÿ.
00000210	3E	20	40	00	EB	EA	00	00	00	00	00	00	00	00	00	00	> @.лк..........
00000220	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000250	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000260	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000002A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000002B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000002C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000002D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000002E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000002F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000300	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000310	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000320	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000330	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000340	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000350	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000360	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000380	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000390	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000003A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000003B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000003C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000003D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000003E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000003F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

8. Секция данных.

Секция данных будет хранить наши данные. У нас это две строки - "Hello World" и "Полковников" (Фамилия студента). Каждая из этих строк должна заканчиваться NULL-байтом (00). Поэтому давайте просто запишем эти данные в секцию и выравним её размер до 0x200 (512) байт.



Адрес каждой строки вычисляется по формуле:

$$ImageBase + VirtualAddress + DataOffset$$

где VirtualAddress – относительный виртуальный адрес секции;

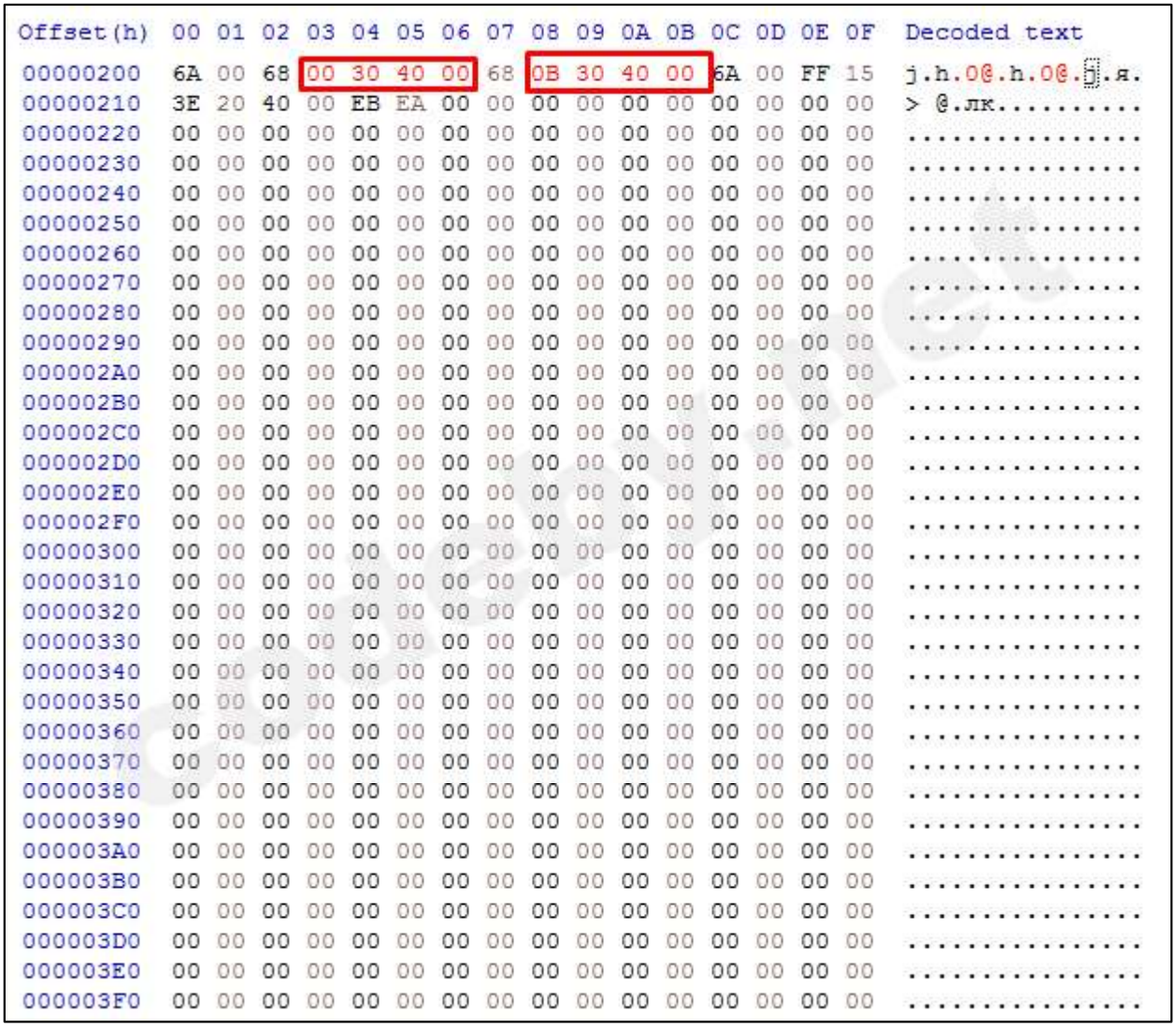
DataOffset – смещение данных относительно начала секции.

Теперь мы знаем виртуальные адреса двух нужных нам строк:

Для фамилии автора - 0x403000

Для "Hello World!" - 0x40300C

Заменим и эти значения в секции кода (где ранее было 0xAAAAAAAA).



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000200	6A	00	68	00	30	40	00	68	0B	30	40	00	6A	00	FF	15	j.h.0@.h.0@.j.я.
00000210	3E	20	40	00	EB	EA	00	00	00	00	00	00	00	00	00	00	> @.лк.....
00000220	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000240	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000250	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000260	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000280	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000300	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000310	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000320	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000330	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000340	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000350	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000360	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000380	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000390	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000003A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000003B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000003C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000003D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000003E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000003F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Заполнение файла закончено. Можно запустить для проверки.

Найти произвольный рабочий PE файл и внести в него «вредоносную» сигнатуру. Например, изменить файл HxD.exe так, чтобы после его запуска бесконечно появлялось окно с сообщением «Файл инфицирован!» и фамилией автора в заголовке.

Для более удобного чтения полей PE файла, можно воспользоваться утилитой PE Explorer.

2.3 Задание на выполнение лабораторной работы

1. Изучить структуру PE файла;
2. Составить свой PE файл;
3. Добавить «вредоносную» сигнатуру в любой PE другой файл.

2.4 Содержание отчета

1. Титульный лист;
2. Цель работы;
3. HEX Код ключевых моментов файла;
4. Скриншоты работы файла;
5. Вывод.