Форматы исполняемых файлов

В практической деятельности большинства программистов рано или поздно возникают проблемы, для решения которых необходимо знать форматы исполняемых файлов. Если ассемблер является отражением архитектуры компьютера, то формат исполняемого файла является отражением архитектуры операционной системы.

Этот и следующие уроки будут посвящены вопросам организации программирования на ассемблере для операционной системы Windows 95/98/NT. Формат исполняемого файла в этих системах отличается от формата файла операционной системы MS-DOS. Таким образом, мы будем иметь дело уже с несколькими форматами исполняемых файлов. Для повышения эффективности разработки программ на ассемблере необходимо понимать их отличия.

Основная программно-аппаратная платформа, на которой работают операционные системы фирмы Microsof, — компьютеры с архитектурой IBM PC. Номенклатура операционных систем, которые можно встретить сегодня на компьютерах данной архитектуры, следующая:

- MS-DOS;
- Windows 3.1;
- Windows 95/98;
- Windows NT;
- UNIX-подобные системы, в частности различные варианты Linux.

С точки зрения архитектуры, данные системы имеют достаточно сильные отличия. Формат исполняемого файла должен в той или иной мере отражать эти отличия. В настоящее время существуют четыре формата исполняемых файлов для перечисленных операционных систем:

- **`.com`** (CP/M и MS-DOS);
- **`.exe`** (MS-DOS);
- **`.exe`** (Windows 3.1) или NE-формат исполняемых файлов;
- **`.exe`** (Windows 95/98/NT) или PE-формат исполняемых файлов;
- COFF- и ELF-форматы исполняемого файла UNIX.

Далее обсудим эти форматы в порядке их появления на свет. Заодно раскроем и исторический аспект этого вопроса.

Исполняемый файл COM-формата

Данный формат исполняемого файла является самым старым. Если считать началом «нашей эры» конец 70-х годов, когда появились компьютеры на базе

микропроцессора i8086, то основы формата COM появились еще «до нашей эры». В то время были популярны компьютеры на базе 8-разрядных микропроцессоров i8080 (отечественный аналог — КР580ВМ80). Одной из отличительных характеристик этих микропроцессоров была поддержка размера оперативной памяти не более 64 Кбайт. Таким образом, размер программы, выполняющейся на компьютере с микропроцессором i8080, изначально был ограничен его архитектурой. А с учетом того, что некоторая часть оперативной памяти отводилась для программ операционной системы, — реальный размер программы был еще меньше. COM-формат поддерживался операционной системой CP/M фирмы Digital Research и показал свою эффективность. С появлением 16-разрядного микропроцессора фирмы Intel i8086/88, возникла необходимость в операционной системе, которая в полной мере использовала бы его возможности. Фирма Microsoft разработала операционную систему MS-DOS. Большинство программ в то далекое время имело небольшой размер, и фирма Microsoft, не изобретая заново велосипед, ввела поддержку COM-формата в операционную систему MS-DOS. На уроке 10 мы уже обсуждали данный формат исполняемого файла. При этом отмечалось, что исходный файл должен удовлетворять определенным требованиям. Поэтому мы не будем повторять то, что уже было сказано, а проникнем немного вглубь вопроса. Для этого в качестве примера рассмотрим исходный текст программы, оформленный в соответствии с требованиями COM-формата (листинг 1).

Листинг 1. Пример исходного файла программы в COM-формате

```
;---------Prg18_1.asm---------------------------------
;Пример программы в COM-формате.
;Основа примера – программа Prg_3_1.asm (урок 3)
;-----------------------------------------------------
codesegment      para public 'code'  ;начало сегмента кода
    assume cs:code, ds: code, ss: code
    org    100h
mainproc
    jmp    EndData
message    db     'Введите две шестнадцатеричные цифры,$'
EndData:
;ds инициализирует система
    mov    ah, 9
    mov    dx, offset message
    int    21h
    xor    ax, ax              ;очистить регистр ax
    mov    ah, 1h              ;1h в регистр ah
    int    21h                 ;генерация прерывания с номером 21h
    mov    dl, al              ;содержимое регистра al в регистр dl
    sub    dl, 30h             ;вычитание: (dl)=(dl)-30h
    cmp    dl, 9h              ;сравнить (dl) с 9h
    jle    M1     ;перейти на метку M1, если dl<9h или dl=9h
    sub    dl, 7h              ;вычитание: (dl)=(dl)-7h
M1:                ;определение метки M1
    mov    cl, 4h        ;пересылка 4h в регистр cl
    shl    dl, cl        ;сдвиг содержимого dl на 4 разряда влево
    int    21h           ;вызов прерывания с номером 21h
```

```
    sub    al, 30h       ;вычитание: (dl)=(dl)-30h
    cmp    al, 9h        ;сравнить (al) с 9h 28
    jle    M2            ;перейти на метку M2, если al<9h или al=9h
    sub    al, 7h        ;вычитание: (al)=(al)-7h
M2:              ;определение метки M2
    add    dl, al        ;сложение: (dl)=(dl)+(al)
    mov    ax, 4c00h     ;пересылка 4c00h в регистр ax
    int    21h           ;вызов прерывания с номером 21h
mainendp
codeends          ;конец сегмента кода
end main          ;конец программы с точкой входа main
```

Такая структура исходного файла для COM-формата обусловлена тем, что построенный на его основе исполняемый файл является слепком участка памяти. После его загрузки операционной системе не требуется производить никаких настроек.

Интересно посмотреть на дамп COM-файла (рис. 1).

```
  0000:0000  EB 26 90 C2 E2 E5 E4 E8 F2 E5 20 E4 E2 E5 20 F8
..&Введите две ш
  0000:0010  E5 F1 F2 ED E0 E4 F6 E0 F2 E5 F0 E8 F7 ED FB E5
естнадцатеричные
  0000:0020  20 F6 E8 F4 F0 FB 2C 24 B4 09 BA 03 01 CD 21 33
цифры,$......!3
  0000:0030  C0 B4 01 CD 21 8A D0 80 EA 30 80 FA 09 7E 03 80
....!..Ъ.0Ъ..~.Ъ
  0000:0040  EA 07 B1 04 D2 E2 CD 21 2C 30 3C 09 7E 02 2C 07
.......!,0<.~.,.
  0000:0050  02 D0 B8 00 4C CD 21
....L.!
```

Рис. 1. Дамп COM-файла

В отличие от других форматов исполняемых файлов, как мы увидим ниже, по внешнему виду приведенного дампа трудно определить (см. рис. 1), что этот файл тоже может быть загружен и выполнен. Это действительно образ памяти, который загружается операционной системой в сегмент со смещением 100h и исполняется. Необязательным признаком COM-файла является наличие в качестве первой инструкции команды перехода. В нашем случае — это **`EB 26`** (**`jmp EndData`**).

Исполняемый файл MZ-формата (MS-DOS)

Разработанный фирмой Microsoft COM-формат исполняемых файлов мог применяться лишь для небольших программ. Для MS-DOS, поддерживающей сегментную организацию программы, был разработан специальный формат исполняемых файлов — *MZ-формат*. Это название условное, но оно будет полезно

нам для дальнейшего обсуждения, и вот почему. В отличие от COM-формата (**.com**), расширения файлов всех остальных форматов — **.exe**. Требуется заглянуть внутрь содержимого исполняемого файла, чтобы понять, с каким конкретно форматом файла мы имеем дело. Поэтому для однозначности с каждым типом исполняемого файла связывается короткая аббревиатура, по которой понятно, о каком формате идет речь.

Итак, MZ-формат был разработан фирмой Microsoft для поддержки многосегментных программ в среде MS-DOS. С точки зрения структуры, файл MZ-формата имеет три части:

- заголовок;
- таблицу размещения;
- программный код.

Заголовок состоит из полей фиксированного размера (табл. 1).

Таблица 1. Содержимое полей заголовка файла MZ-формата

Смещение	Длина	Значение
0000h	2 байта	Сигнатура (фиксированная последовательность символов) — «MZ» (4d 5ah) или «ZM» (5a 4dh) (MZ — Mark Zbikowski)
		Весь исполняемый файл делится на страницы по 512 байт. Понятно, что совсем необязательно условие кратности длины файла этому значению. Поэтому следующее поле содержит длину остатка кода последней страницы исполняемого файла.
0002h	2 байта	Количество байт в последней странице файла MZ-формата. Равно остатку от деления длины файла на значение 512
		Поле 0004h содержит длину исполняемого файла в 512-байтных страницах.
0004h	2 байта	Длина файла MZ-формата в 512-байтных страницах
0006h	2 байта	Количество элементов в таблице размещения
		Так как заголовок может иметь переменную длину, то поле 0008h сообщает его длину в 16-байтных параграфах. Эта информация используется загрузчиком операционной системы для определения начала загрузочного модуля.
0008h	2 байта	Длина заголовка в параграфах
		Следующие два поля сообщают минимальное и максимальное количество памяти (в параграфах), необходимое для исполнения программы, помимо размера ее программного кода.
000Ah	2 байта	Минимальное количество дополнительных параграфов, необходимых для исполнения программы
000Ch	2 байта	Максимальное количество дополнительных параграфов, необходимых для исполнения программы

Поля 000Eh и 00010h предназначены для хранения указателя стека **ss:sp**, но реально они формируются загрузчиком ОС на стадии загрузки программы в память для выполнения.

000Eh	2 байта	Начальное значение, загружаемое в регистр **ss**
0010h	2 байта	Начальное значение, загружаемое в регистр **sp**
0012h	2 байта	0 или поразрядная дополненная контрольная сумма всех 16-битных слов в файле, за исключением данного поля

Следующие два поля определяют точку входа в программу после ее загрузки.

0014h	2 байта	Значение, загружаемое в регистр **ip** во время загрузки программы в память
0016h	2 байта	Значение, загружаемое в регистр **cs** во время загрузки программы в память (формируется загрузчиком)
0018h	2 байта	Смещение от начала файла таблицы размещения или значение 40h, если это исполняемый файл одного из форматов NE, PE и т. д. (см. ниже)

Последнее поле предназначено для организации оверлейных программ. В нем содержится номер, в соответствии с которым впоследствии будет организована загрузка модулей, входящих в оверлейную программу. Для главной программы поле имеет значение 0.

001Ah	2 байта	Номер оверлейного модуля

На этом заголовок заканчивается. Далее следует информация, зависящая от того программного средства, которое сформировало данный исполняемый файл. Представленный на рис. 2 исполняемый файл был сформирован программой TLINK версии 7.1 (фирмы Borland). Эта программа формирует следующую опознавательную информацию:

001Ch	2 байта	Сигнатура 01 00h
001Eh	1 байт	Сигнатура 0fbh
001Fh	1 байт	Версия TLINK
0020h	2 байта	Сигнатура 6a72h

Другие программы, например архиваторы, формирующие саморазворачивающиеся файлы (self-extracting), также помещают в эту область уникальную информацию. Зная это, вы можете уточнить информацию о программе, сформировавшей исполняемый файл.

В поле со смещением 0018h находится значение смещения таблицы размещения. Что это за таблица? Таблица размещения представляет собой совокупность элементов, предназначенных для связи сегментов в программе. Число элементов таблицы содержится в поле со смещением 0006h. Каждый элемент имеет следующий формат:

0000h	2 байта	Смещение внутри сегмента
0002h	2 байта	Сегмент размещения

Во время загрузки программы в память к ней добавляется префикс программного сегмента (PSP — program segment prefix) размером 256 байт. Сама программа загружается вслед за этим префиксом. Информация из префикса может быть использована для выделения символов из командной строки, определения объема доступной памяти, информации об окружении и т. д. В заключение отметим содержимое регистров после загрузки файла MZ-формата:

- **ax** — количество символов в командной строке;

- **bx:cx** — размер загрузочного модуля;
- **ds** и **es** — сегментные составляющие адреса начала PSP данной программы. При необходимости их нужно подстраивать. Мы это делали постоянно при написании программ, вставляя в начало исполняемой части программы команды код, подобный следующему:

```
mov    ax,data
mov    ds,ax
```

- **ss:sp** — указатель на вершину стека;
- **cs:ip** — адрес точки входа программы, то есть той инструкции, адрес которой указан меткой в заключительной директиве программы **end**.

```
0000:0000 4D 5A 64 01 02 00 01 00 20 00 00 00 FF FF 03 00
          MZd..... ....... 
0000:0010 00 01 00 00 00 00 13 00 3E 00 00 00 01 00 FB 71
          ........>......q
0000:0020 6A 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00
          jr..............
0000:0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00
          ................
0000:0040 13 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
          ................
0000:0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
          ................
… … … далее тоже самое
0000:01F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
          ................
0000:0200 C2 E2 E5 E4 E8 F2 E5 20 E4 E2 E5 20 F8 E5 F1 F2
          ....... ... ....
0000:0210 ED E0 E4 F6 E0 F2 E5 F0 E8 F7 ED FB E5 20 F6 E8
          ............. ..
0000:0220 F4 F0 FB 2C 24 00 00 00 00 00 00 00 00 00 00 00
          ...,$...........
0000:0230 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F
          ????????????????
… … … далее тоже самое
0000:0320 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F 3F
          ????????????????
0000:0330 B8 00 00 8E D8 B4 09 BA 00 00 CD 21 33 C0 B4 01
          ...........!3...
0000:0340 CD 21 8A D0 80 EA 30 80 FA 09 7E 03 80 EA 07 B1
          .!..Ъ.0Ъ..~.Ъ...
0000:0350 04 D2 E2 CD 21 2C 30 3C 09 7E 02 2C 07 02 D0 B8
          ....!,0<.~.,....
0000:0360 00 4C CD 21                                     .L.!
```

Рис. 2. Дамп EXE-файла MZ-формата

Исполняемый файл NE-формата

(Windows 3.x)

NE-формат (New Executeable code file) исполняемого файла был разработан для операционной системы Windows 2.0. Этот же формат использовался в операционной системе OS/2 фирмы IBM. Исполняемый файл NE-формата состоит из двух исполняемых файлов:

- исполняемый файл MZ-формата;
- исполняемый файл NE-формата.

Если посмотреть на пример дампа исполняемого файла NE-формата (рис. 4), то видно, что первые несколько десятков байт являются не чем иным, как только что рассмотренным нами файлом MZ-формата (MS-DOS). В его задачу входит вывод сообщения типа «This program required Microsoft Windows». Наверняка оно вам известно, как и та ситуация, при которой это сообщение выводится. Загрузчик Windows знает об этой особенности загрузочного файла и сразу начинает работать со второй его частью. Загрузчик MS-DOS впрямую запускает программу с расширением **`.exe`** на исполнение. Данная маленькая программа добросовестно исполняется, выводя вышеуказанное (или подобное ему) сообщение, и благополучно завершается. В принципе, вместо этой маленькой программы может быть полноценная программа, представляющая собой DOS-версию приложения.

Определить тот факт, что исполняемый файл с расширением **`.exe`** является файлом формата NE, достаточно просто. Для этого достаточно посмотреть на значение в байте со смещением **`18h`** от начала файла. Если там значение **`40h`**, то перед вами не MZ-формат. Далее необходимо посмотреть на содержимое слова со смещением **`3Ch`**. Значение в нем показывает смещение заголовка NE исполняемого файла относительно начала файла. Общий формат исполняемого файла NE-формата показан на рис. 3. Фрагменты дампа файла NE-формата показаны на рис. 4. В начале файла расположена программа-заглушка MS-DOS. Формат этой части файла NE-формата показан в табл. 2.

Таблица 2. Содержимое полей DOS-заголовка файла NE-формата

Смещение	Длина поля	Содержимое
0000h	32 байта	Заголовок исполняемого файла MS-DOS (см. табл. 1, поля 0000-0018h)
0020h	28 байт	0 (резерв)
003ch	2 байта	Если значение поля со смещением 0018h равно или больше значения 40h, то это приложение Windows и данное поле содержит смещение от начала файла до начала заголовка Windows-приложения
003eh	2 байта	0 (резерв)
0040h	? байт	Программа-заглушка MS-DOS

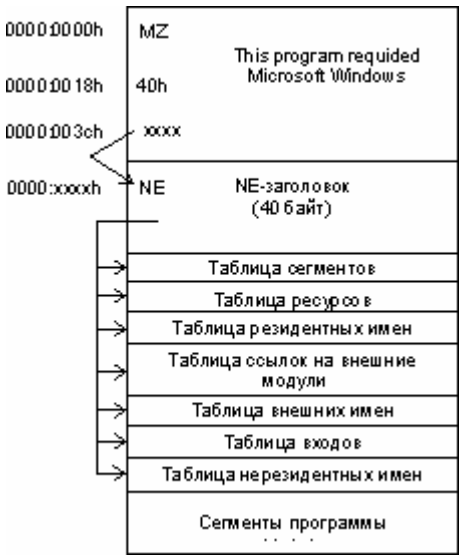

Рис. 3. Структура исполняемого файла NE-формата

Как уже было отмечено, в поле со смещением **003ch** от начала файла содержится смещение начала данных, относящихся непосредственно к Windows-приложению. Эти данные начинаются с заголовка, содержимое которого описано в табл. 3 (все смещения, если не оговорено специально, считаются от начала NE-заголовка, а не от начала файла).

```
0000:0000  4D 5A 6B 00 BD 04 00 00 20 00 00 00 FF FF 07 00
MZk..... .......
0000:0010  00 01 65 40 00 00 00 00 40 00 00 00 01 00 00 00
..e@....@.......
0000:0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
0000:0030  00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00
................
0000:0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
… … … далее тоже самое
0000:01F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
0000:0200  E8 53 00 54 68 69 73 20 70 72 6F 67 72 61 6D 20
.S.This program
0000:0210  72 65 71 75 69 72 65 73 20 4D 69 63 72 6F 73 6F
requires Microso
0000:0220  66 74 20 57 69 6E 64 6F 77 73 2E 0D 0A 24 20 20   ft
Windows...$
0000:0230  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0000:0240  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0000:0250  20 20 20 20 20 20 5A 0E 1F B4 09 CD 21 B8 01 4C
```

```
Z.....!..L
0000:0260  CD 21 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.!..............
0000:0270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
… … … далее тоже самое
0000:03F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
0000:0400  4E 45 05 3C 75 07 CF 0C 00 00 00 00 01 83 22 00
NE.<u.........".
0000:0410  00 00 00 00 02 1E 15 00 00 00 00 00 22 00 07 00
............"...
0000:0420  5E 22 40 00 50 01 15 07 3A 07 48 07 44 18 00 00
^"@.P...:.H.D...
0000:0430  FA 01 05 00 00 00 02 00 00 00 00 00 00 00 00 04
................
0000:0440  D6 01 2A 18 40 0D 2C 18 A2 02 A4 1E 50 0D A4 1E
..*.@.,.....P...
0000:0450  A0 03 A7 AC 50 0D A8 AC 1D 09 AF 63 50 0D B0 63
....P......cP..c
0000:0460  4B 0C DC 2D 50 1D DC 2D C6 0D DA 83 50 1D DA 83   K..-
P..-....P...
0000:0470  FA 11 97 B8 50 1D 98 B8 D0 17 62 60 50 1D 62 60
....P.....b`P.b`
0000:0480  E2 1A 00 34 50 1D 00 34 89 1C 09 4F 50 1D 0A 4F
...4P..4...OP..O
0000:0490  0E 1F 6C 23 50 1D 6C 23 33 20 B3 5F 50 1D B4 5F
..l#P.l#3 ._P.._
0000:04A0  3D 23 2A 6B 50 1D 2A 6B A8 26 41 31 50 1D 42 31
=#*kP.*k.&A1P.B1
0000:04B0  3A 28 7A 45 50 1D 7A 45 78 2A 11 3D 50 1D 12 3D
:(zEP.zEx*.=P..=
… … …
```

Рис. 4. Фрагмент дампа EXE-файла NE-формата

Таблица 3. Содержимое полей Windows-заголовка файла NE-формата

Смещение	Длина	Значение
0000h	2 байта	Сигнатура — «NE» (4e 45h)
0002h	2 байта	Версия и модификация версии программы компоновщика TLINK, создавшей данный исполняемый файл
0004h	2 байта	Смещение таблицы входов (относительно начала NE-заголовка)
0006h	2 байта	Длина таблицы входов (в байтах)
0008h	4 байта	0
000ch	2 байта	Байт флагов, описывающих содержимое исполняемого файла

Содержимое последних байт представляет собой комбинацию бит, по которой можно

судить, что собой представляет исполняемый файл и каковы его особенности размещения в памяти. Местоположение некоторых из этих флагов показано в табл. 4.

Таблица 3. Поля Windows-заголовка файла NE-формата (*продолжение*)

Смещение	Длина	Значение
000eh	2 байта	Число сегментов автоматических данных. Если биты 0 и 1 в поле флагов (см. табл. 4) равны 0, то это поле равно 0
0010h	2 байта	Определяет размер (в байтах) локальной кучи
0012h	2 байта	Определяет размер стека (в байтах)
0014h	4 байта	Значение в **`cs:ip`** — адрес точки входа: **`cs`** (старшее слово) — индекс сегмента кода в таблице сегментов (см. ниже); **`ip`** (младшее слово) — смещение в сегменте кода
0018h	4 байта	Значение в **`ss:sp`** — адрес вершины стека **`ss`** (старшее слово) — индекс сегмента стека в таблице сегментов (см. ниже); **`sp`** (младшее слово) — смещение в сегменте стека
001ch	2 байта	Количество сегментов в модуле (число элементов в таблице сегментов)
001eh	2 байта	Количество элементов в таблице модулей (то есть это, фактически, количество импортируемых модулей). Поле связано с таблицей ссылок, адрес которой находится в слове 0028h
0020h	2 байта	Размер в байтах таблицы нерезидентных имен
0022h	2 байта	Относительное смещение от начала NE-заголовка до начала таблицы сегментов
0024h	2 байта	Относительное смещение от начала NE-заголовка до начала таблицы ресурсов
0026h	2 байта	Относительное смещение от начала NE-заголовка до начала таблицы резидентных имен
0028h	2 байта	Относительное смещение от начала NE-заголовка до начала таблицы ссылок на модули
002ah	2 байта	Относительное смещение от начала NE-заголовка до начала таблицы внешних (импортируемых) имен
002ch	2 байта	Относительное смещение от начала NE-заголовка до начала таблицы нерезидентных имен
002eh	2 байта	Не используется
0030h	2 байта	Количество перемещаемых точек входа
0032h	2 байта	Определяет величину, в соответствии с которой производится разбиение исполняемого файла в NE-формате на логические сектора (возможно, не равные по размеру физическим). В этих сегментах располагается содержимое сегментов исходной программы и ресурсов. Учет занимаемого ими пространства также производится в этих логических сегментах. Для получения размера логического сектора необходимо число 2 воз-

		вести в степень равную значению этого поля. Устанавливается опциями **`/Align:n`** (**`link`** фирмы Microsoft) или **`/A=n`** (**`tlink`** фирмы Borland (Inprise)). Если **`n=16`**, то значение в этом поле равно 4, если **`n=512`**, то значение в этом поле равно 9
0034h	2 байта	Число сегментов с ресурсами

Таблица 3 (*продолжение*)

Смещение	Длина	Значение
0036h	1 байт	Операционная система. Значения бит: бит 0 — неизвестная операционная система; бит 1 — OS/2; бит 2 — Microsoft Windows Остальные биты не используются
0037h	1 байт	Дополнительная информация об исполняемом файле Значения бит: 0 — поддержка длинных имен файлов; 1 и 2 — для приложений Windows 2.x (не актуально); 3 — исполняемый файл содержит область быстрой загрузки (gangload area или fastload area). Эта область представляет собой набор сегментов и ресурсов, хранящихся в одном разделе файла. В этом случае эти данные помещаются загрузчиком в память одной командой чтения, что ускоряет загрузку модуля
0038h	2 байта	Смещение в 128-байтовых секторах до начала области быстрой загрузки
003ah	2 байта	Длина в 128-байтовых секторах области быстрой загрузки
003ch	2 байта	Не используется
003eh	2 байта	Ожидаемая версия Windows (в которой программа будет нормально функционировать)

Таблица 4. Назначение флагов поля 000ch Windows-заголовка файла NE-формата

Номер бита в слове	Длина поля, бит	Значение
0	1	Устанавливается в 1, если исполняемый файл имеет один сегмент данных
1	1	Устанавливается в 1, если исполняемый файл имеет несколько сегментов данных
2	1	Не используется
3	1	Программа будет выполняться в защищенном режиме
4–7	4	Не используется
8–9	2	Используется OS/2
10	1	Не используется
11	1	Самозагружающееся приложение
12	1	Не используется

13	1	Во время компоновки была не фатальная ошибка
14	1	Устанавливается для библиотек, загружаемых выше кадра EMS
15	1	Устанавливается, если приложение использует библиотечные модули. Если бит установлен в 1, то Windows вызывает процедуру инициализации по адресу **cs:ip**. Этот бит используется совместно с битом 0. Если бит 0 равен 1, то регистр **ds** содержит адрес сегмента данных библиотеки, иначе **ds** содержит адрес сегмента данных приложения

За заголовком следуют таблицы (см. рис. 3), начальные адреса (точнее, относительные смещения) которых были указаны в различных полях заголовка. Это таблица сегментов, таблица ресурсов, таблицы резидентных и нерезидентных имен, таблица ссылок, таблица входов. Рассмотрим их.

Таблица сегментов

Таблица сегментов представляет собой совокупность записей, описывающих каждый сегмент исполняемого файла. Относительное смещение начала этой таблицы находится в поле заголовка **0022h**. Но это значение можно игнорировать, так как загрузчик всегда считает, что таблица сегментов располагается сразу за заголовком NE-файла. Размер записи в таблице 8 байт. Поля записи содержат информацию о длине и типе сегмента, расположении сегмента в файле и некоторые другие его характеристики (табл. 5).

Таблица 5. Элемент таблицы сегментов

Смещение	Длина, байт	Значение
0000h	2	Смещение в секторах к сегменту (от начала файла). Чтобы понять, о каких секторах идет речь, см. описание поля 0032h в Windows-заголовке файла NE-формата (табл. 3)
0002h	2	Длина сегмента (в байтах). Если 0, то размер сегмента равен 64 Кбайт
0004h	2	Флаги, характеризующие сегмент (табл. 6)
0006h	2	Определяет минимальный размер памяти для размещения сегмента. Если 0, то минимальный размер сегмента равен 64 Кбайт

В табл. 6 представлены флаги, которые характеризуют сегмент NE-файла.

Таблица 6. Флаги, характеризующие сегмент

Номер бита	Назначение
0	Тип сегмента: 0 — сегмент кода;

	1 — сегмент данных
1	Устанавливается, если память для сегмента выделяется загрузчиком
2	Устанавливается, если сегмент загружен
3	Не используется
4	0 — сегмент неперемещаемый (Fixed), то есть его нельзя перемещать в памяти; 1 — сегмент перемещаемый (MovAble)
5	0 — сегмент не допускает совместного использования (NonShareAble) 1 — сегмент допускает совместное использование (ShareAble)

Таблица 6 (*продолжение*)

Номер бита	Назначение
6	0 — сегмент типа LoadOnCall (то есть загружается при обращении к сегменту); 1 — сегмент типа Preload (то есть должен быть загружен в память целиком во время загрузки модуля)
7	0 — сегмент доступен для чтения и записи; 1 — сегмент типа ExecuteOnly (только для выполнения). Если это сегмент данных (см. бит 1), то сегмент доступен только для чтения (ReadOnly)
8	1 — сегмент сразу за данными содержит информацию о размещении. Если сегмент имеет объекты, значение которых зависит от адреса загрузки, то за сегментом следует таблица настройки. Эта таблица содержит *записи размещения*, которые характеризуют каждый из таких объектов (см. также конец данного раздела)
9	Используется OS/2
10–11	Не используется
12	1 — сегмент типа DiscardAble (при необходимости система может выгрузить сегмент из памяти)
13–15	Не используется

Максимальное число сегментов, которое может содержать модуль, — 253. Это ограничение связано с полем в таблице входов (табл. 9), где адреса экспортируемых функций хранятся в виде логических адресов. В этих логических адресах для хранения сегментной составляющей используется всего один байт, в который заносится номер сегмента в соответствии с таблицей сегментов. Недостающие три сегмента (с номерами 0, 0feh и 0ffh) используются загрузчиком Windows специальным образом.

Таблица описания ресурсов

Таблица описания ресурсов определяет расположение и характеристики каждого ресурса в исполняемом файле. Сами ресурсы при этом содержатся в другом месте. Таблица описания ресурсов располагается сразу за таблицей сегментов. Значение смещения ее начала содержится в поле NE-заголовка **0024h**. Почему эта таблица может представлять интерес? Первое, что приходит в голову, — использовать информацию о ресурсах для адаптации программ к определенным условиям функционирования (например русификация англоязычных программ). Конечно, для этой цели можно использовать другие специально предназначенные для этого средства, например редакторы ресурсов. Мы не будем рассматривать, как описывается в памяти каждый тип ресурса, это заняло бы достаточно много места. Рассмотрим только один из них. Таблица описания ресурсов имеет формат, приведенный в табл. 7.

Таблица 7. Содержимое таблицы описания ресурсов

Смещение поля в записи	**Длина поля**	**Назначение**
0	2 байта	Размер сектора. Значение в этом поле должно быть равным полю со смещением 0032h в Windows-заголовке файла NE-формата (см. табл. 3)
(2-??)h	??	Массив секций переменной длины, каждая из которых описывает ресурсы определенного типа для данного приложения (табл. 8)
(??+1)h	2 байта	Нулевое слово — обозначает конец массива со структурами, описывающими типы ресурсов в исполняемом файле
(??+1)+1)h	??	Строки символов, ассоциируемые с именованными ресурсами в данной таблице ресурсов. Каждое имя начинается с байта, содержащего число символов в имени (табл. 9).
??	1 байт	Нулевой байт, обозначающий конец таблицы описания ресурсов

В табл. 8 представлена структура описания ресурса.

Таблица 8. Структура описания ресурса

Смещение поля в записи	**Длина поля**	**Назначение**
0	2 байта	Идентификатор ресурса — определяет тип ресурса. Старший бит этого поля установлен, если ресурс заранее определен. Возможные значения поля без учета этого старшего бита приведены в табл. 9. Если старший бит поля сброшен, то значение в этом поле представляет собой смещение имени ресурса в массиве строк символов (смещение (??+1)+1)h в табл. 7)
2	2 байта	Количество ресурсов данного типа. Исходя из значения этого поля, можно определить длину секции описания ресурсов данного типа

4	4 байта	0
8	??	Массив структур, содержащих дополнительные характеристики ресурсов данного типа. Количество этих структур равно значению в поле со смещением +2 таблицы. Назначение полей структур этого массива показано в табл. 10.

В табл. 9 представлены возможные идентификаторы ресурсов.

Таблица 9. Идентификаторы ресурсов

Идентификатор	Тип ресурса
1	Курсор
2	Растровое изображение
3	Значок

Таблица 9 (*продолжение*)

Идентификатор	Тип ресурса
4	Меню
5	Окно диалога, управляющая кнопка
6	Таблица строк
7	Каталог шрифтов
8	Шрифт
9	Клавиша быстрого вызова
0ah	Данные пользователя
0bh	Таблица ошибок
0ch	Каталог курсоров
0eh	Каталог значков
0fh	Таблица имен
10h	Информация о версии

В табл. 10 представлена структура с характеристиками ресурса определенного типа.

Таблица 10. Содержимое структуры с характеристиками ресурса

Смещение поля	Длина, байт	Назначение
0	2	Смещение в единицах (см. поле 0 табл. 7) от начала файла до месторасположения ресурса
2	2	Размер ресурса в байтах
4	2	Флаги ресурса (табл. 11)
6	2	Если старший бит поля равен 1, то в нем содержится целочисленный идентификатор ресурса. Иначе это поле содержит смещение в байтах от начала таблицы ресурсов до символьной строки с именем ресурса. Для полного понимания назначения этого поля обратитесь к материалу урока 18, в котором

		рассматриваются вопросы разработки приложений для Windows, в том числе с использованием ресурсов. Здесь пока отметим, что ресурсы в приложении для Windows могут идентифицироваться как с помощью целочисленного значения, так и с помощью своего имени, то есть символьной строки
8	4	Не используется

В табл. 11 представлены флаги, характеризующие ресурс.

Таблица 11. Флаги, характеризующие ресурс

Номер бита	Назначение
0–3	Не используется
4	Устанавливается в 1, если ресурс перемещаемый (MoveAble), сбрасывается в 0, если ресурс неперемещаемый (Fixed)
5	Устанавливается, если ресурс разделяемый
6	Устанавливается в 1, если ресурс должен быть предварительно загружен (PreLoad). Устанавливается в 0, если ресурс загружается по запросу
7–15	Не используется

Месторасположение конкретного ресурса в файле определяется значением в поле со смещением 0 структуры, содержащей характеристики ресурса определенного типа (см. табл. 10). Каждый тип ресурса описывается структурой определенного типа. Этот материал достаточно громоздок, поэтому мы для примера рассмотрим только структуры для описания самого простого ресурса — растрового изображения (табл. 12) — и чуть более сложного — ресурса значка (табл. 13). Структура ресурса растрового изображения аналогична структуре файла bmp-формата, за исключением заголовка в начале этого файла. Каждый ресурс типа растровое изображение имеет собственный экземпляр в массиве структур, описывающем типы ресурсов для данного приложения (см. поле 2–?? табл. 7).

Таблица 12. Структура описания ресурса растровое изображение

Смещение	Длина	Назначение
0	40 байт	Структура, полностью описывающая растровое изображение (урок 20)
28h	?? байт	Массив цветов (урок 20)

Описание ресурса значок отличается тем, что в массиве структур, описывающем типы ресурсов для данного приложения (см. поле 2–?? табл. 7), располагается только одна структура, описывающая тип ресурса значок, а ресурс для конкретного значка располагается в массиве структур (поле со смещением 6 в табл. 13).

Таблица 13. Структура описания ресурса типа значок

Смещение	Длина,	Назначение

	байт	
0	2	0
2h	2	Тип ресурса. Для значка поле равно 1
4h	2	Число ресурсов значок — определяет количество структур в следующем поле
6	??	Массив структур, описывающих ресурсы значков (табл. 14)

В табл. 14 представлена структура описания ресурса значок.

Таблица 14. Структура описания ресурса значок

Смещение поля в структуре	Длина, байт	Назначение
0	1	Ширина изображения значка в пикселах (16, 32 или 64)
1	1	Высота изображения значка в пикселах (16, 32 или 64)

Таблица 14 (*продолжение*)

Смещение поля в структуре	Длина, байт	Назначение
2	1	Число цветов в изображении значка (2 — монохромное изображение, 8 или 16 — количество цветов в изображении)
3h	5	0
8h	4	Размер в байтах ресурса значок
0ch	4	Уникальный номер данного значка, генерируемый компилятором ресурсов

Далее следует описание изображения значка, которое очень похоже на описание растрового изображения, поэтому мы его приводить не будем (см. урок 20).

Главная цель, которую мы преследовали, приводя данные таблицы, это согласовать размерности различных структур. Владея этой информацией читатель при необходимости сможет определить местонахождение ресурсов в исполняемом файле и выполнить простейшие действия, к примеру, такое действие, как редактирование ресурсов с символьной информацией. Большая детализация темы, связанной с ресурсами, смысла не имеет, так как появляется слишком много подробностей, из-за которых вести однозначную линию обсуждения, соответствующую тематике урока, становится достаточно трудно.

Таблица резидентных имен

Таблица резидентных имен содержит имена экспортируемых модулем функций, то есть тех функций, которые могут быть вызваны в другой программе. Таблица резидентных имен состоит из совокупности записей с информацией об именах

экспортируемых функций в исполняемом файле. Таблица резидентных имен постоянно находится в памяти (этим она отличается от таблицы нерезидентных имен (см. далее)). Содержимое одной записи таблицы резидентных имен представлено в табл. 15.

Таблица 15. Структура записи таблицы резидентных имен

Смещение	Назначение
00	Длина строки с именем экспортируемой функции. Конец таблицы резидентных имен обозначается нулевым значением этого поля (при этом два других поля записи отсутствуют)
01-??	Имя экспортируемой функции (без конечного нуля, то есть не ASCIIZ-строка). В имени экспортируемой функции различаются строчные и прописные буквы
(??+1)h	Индекс (порядковый номер) записи в таблице точек входа (см. далее), которая адресует идентифицируемую данной записью функцию

Первое имя, которое содержится в этой таблице, — строка с именем модуля, определенным в операторе **`NAME`** файла определений (**`.def`**).

Таблица ссылок на внешние модули

Таблица ссылок на внешние модули (или таблица ссылок на импортируемые функции) содержит смещения в таблице внешних имен (см. ниже) для каждой записи с именем внешнего модуля. Количество записей в этой таблице равно количеству импортируемых модулей (функций). Размер записи таблицы ссылок на импортируемые модули равен двум байтам.

Таблица внешних имен

Таблица внешних имен содержит имена функций, импортируемых данным исполняемым файлом. Каждая запись таблицы содержит два элемента. Первый, размером в один байт, определяет длину символьного имени. Второй содержит имя используемой импортируемой функции (это имя также не заканчивается нулем). Конец таблицы обозначается нулевым байтом.

Таблица входов

При создании исполняемого файла компоновщик строит таблицу точек входа. В этой таблице каждый элемент пронумерован, поэтому возможно обращение к точкам входа по их номерам. Нумерация точек входа не является последовательной, поэтому номера собираются в так называемые «связки», или группы, внутри которых номера следуют последовательно. Связки формируются компоновщиком. Единственное исключение в системе нумерации — основная точка входа в модуль имеет номер 1. Таким образом, таблица входов представляет собой совокупность связок, имеющих определенный формат. Каждая связка начинается с заголовка, формат которого

следующий:

- первый байт заголовка определяет количество входов в связке. Если этот байт равен 00h, то это означает конец таблицы входов;
- второй байт заголовка определяет, является данная точка входа перемещаемой (0ffh) или фиксированной (0feh). Если значение второго байта не равно этим двум значениям, то это — индекс сегмента.

Далее следуют элементы, описывающие точки входа данной связки. Элементы могут быть двух форматов, в зависимости от типа точки входа (перемещаемая (0ffh) или фиксированная (0feh) (табл. 16 и 17).

Таблица 16. Элементы входа (6 байт) для перемещаемого сегмента

Смещение	Размер	Назначение
00h	1 байт	Значение бит: 0 — если равен 1, то вход экспортируемый данным модулем; 1 — если равен 1, то функция, соответствующая данной точке входа, использует общий сегмент данных для всех вызывающих ее программ (нужно только для dll-файлов); 3–7 — если исполняемый файл содержит программный код, выполняющий переходы через кольца защиты, то значение битов 3–7 трактуется как количество слов (16 бит), формирующих стек. При межкольцевом переходе эти слова копируются из стека одного кольца в стек другого кольца

Таблица 16 (*продолжение*)

Смещение	Размер	Назначение
01h	2 байта	Инструкция `int 3fh`
03h	1 байт	Номер сегмента, согласно таблице сегментов (см. табл. 5)
04	2 байта	Смещение в сегменте к точке входа

Таблица 17. Элемент входа (3 байта) для неперемещаемого сегмента

Смещение	Размер	Назначение
00h	1 байт	Биты в байте имеют определенное значение: 0 — если равен 1, то вход экспортируемый; 1 — если равен 1, то функция использует общий сегмент данных для всех вызывающих программ (нужно только для dll-файлов); 3–7 — если исполняемый файл содержит программный код, выполняющий переходы через кольца защиты, то значение битов 3–7 трактуется как количество слов, формирующих стек. При межкольцевом переходе эти слова копируются из стека одного кольца в стек другого кольца
01h	2 байта	Смещение в сегменте

А где же сами имена функций? Ответ на это дают таблицы резидентных и нерезидентных имен. Именно они устанавливают соответствие между именами и -

номерами экспортируемых функций. Элементы этих таблиц имеют одинаковый формат. Потребность в двух одинаковых по формату таблицах возникает из соображений экономии памяти. Если нет необходимости постоянно держать имя экспортируемой функции в памяти (так как обращение к ним осуществляется по их номерам), то это имя необходимо поместить в нерезидентную таблицу имен. И наоборот. Таблицу резидентных имен мы уже рассмотрели. Рассмотрим таблицу нерезидентных имен.

Таблица нерезидентных имен

Назначение таблицы нерезидентных имен аналогично таблице резидентных имен (см. выше). Эта таблица также состоит из записей, которые определяют имена экспортируемых функций в исполняемом файле. Ее отличие от таблицы резидентных имен в том, что данная таблица может быть выгружена на диск, если необходимо освободить память. Символы в именах экспортируемых функций чувствительны к регистру и не являются ASCIIZ-строками. Формат каждого элемента данной таблицы представлен в табл.

Таблица Структура записи в таблице нерезидентных имен

Смещение	**Назначение**
00h	Длина строки с именем в байтах
01h–??h	Имя
??h+01h	Порядковый номер, который является индексом в таблице входов

Таким образом, мы описали заголовок и служебные таблицы исполняемого файла NE-формата. Далее следуют сегменты программы. Их размещение осуществляется в соответствии с информацией поля со смещением **`32h`** относительно начала NE-заголовка.

Некоторые сегменты кода содержат вызовы функций из других сегментов, для чего соответственно нужны данные для разрешения этих дальних ссылок. Ведь истинные сегментные составляющие адреса будут известны лишь на этапе выполнения. Эту настройку выполняет загрузчик. Для этого используются данные из *таблицы настройки*, которая размещается сразу за сегментами кода и данных. В нашем примере это адрес 041bh. Первые два байта таблицы настройки содержат количество элементов в таблице настройки. Далее идет группа элементов размером 8 байт, называемых *записями размещения*. Эти записи содержат информацию о типе адреса, типе ссылки и ее расположении. Структура таблицы настройки показана в табл. 19, 20, 21.

Первый байт элемента таблицы настройки (табл. 19) определяет тип адресной ссылки.

Таблица 19. Значения первого байта элемента таблицы настройки

Величина	**Значение**

0	Младший байт по указанному смещению
2	16-битный селектор
3	32-битный указатель (сегмент:смещение)
5	16-битное смещение
11 (0bh)	48-битный указатель (селектор:32-битное смещение)
13 (0dh)	32-битное смещение

Второй байт определяет один из типов настройки (табл. 20).

Таблица 20. Значения второго байта элемента таблицы настройки

Величина	**Значение**
0	Внутренняя ссылка (межсегментная ссылка)
1	Импортируемый порядковый номер (например, номер функции из dll-библиотеки)
2	Импортируемое имя (из другого модуля)
3	Ссылка на операционную систему

Третий и четвертый байты определяют смещение элемента, требующего настройки, внутри сегмента. Значение остальных четырех байт определяется типом ссылки, указанным вторым байтом (табл. 21).

Таблица 21. Значения 5, 6, 7 и 8 байтов элемента таблицы настройки

Тип ссылки	**Значение**
Внутренняя ссылка	Для фиксированного сегмента: 5 байт — номер сегмента; 6 байт — нулевой; 7 и 8 байты — смещение в сегменте. Для перемещаемого сегмента: 5 байт — 0ffh; 6 байт — нулевой; 7 и 8 байты — соответствуют индексу в таблице входов для сегмента, содержащего данную ссылку
Импортируемое имя	5 и 6 байт — определяют индекс в таблице ссылок на модули; 7 и 8 байты — смещение в таблице внешних (импортируемых) имен
Импортируемый порядковый номер	5 и 6 байт — определяют индекс в таблице ссылок на модули; 7 и 8 байты — порядковый номер функции

Таким образом, мы рассмотрели структуру исполняемого файла в NE-формате. Перейдем теперь к реалиям нашего времени и рассмотрим структуру файлов, которая используется операционной системой Windows NT/95/98.

Исполняемый файл PE-формата (Windows NT 3.5/95/98)

Для 32-разрядных операционных систем фирма Microsoft разработала специальный формат исполняемого файла. Он получил название *переносимый формат* исполняемого файла (PE — Portable Executable). В нашем изложении мы будем называть его *PE-форматом*. Детальное рассмотрение PE-формата требует очень много места, но вряд ли это так необходимо. В большинстве случаев достаточно представлять общую структуру файла PE-формата («знать, что где лежит»). Для получения детальной информации всегда можно обратиться к документации. Необходимо отметить, что идеи, заложенные в PE-формат, не новы. Основы этого формата были заложены в операционной системе UNIX, где аналогичный формат назывался COFF-формат (Common Object File Format — стандартный формат объектного файла). Почему формат файла в 32-разрядных операционных системах назван переносимым (portable)? Это сделано из-за стремления фирмы Microsoft создать единый формат исполняемого файла для реализаций операционной системы Windows NT на различных аппаратных платформах (что, конечно, не означает совместимости на уровне машинных команд).

Перечислим особенности данного формата:

- простота загрузки файла PE-формата. Если сравнивать этот процесс с загрузкой файла NE-формата, то для PE-исполняемого файла он значительно проще. Чтобы загрузить NE-файл, загрузчик должен создать рабочие структуры в памяти, найти информацию для их заполнения внутри NE-файла;
- поддержка сплошной модели памяти. Конструкция PE-файла максимально упрощена с точки зрения загрузки в память. Фактически он представляет собой слепок участка оперативной памяти;
- не требуется настраивать сегменты команд или данных, так как их нет в том виде, как это было в 16-разрядной среде;
- расположение многих полей в PE-файле задается с помощью относительного смещения от начала PE-файла — *Относительного Адреса Поля* (ОАП). Сам PE-файл располагается по *Базовому Адресу Памяти* (БАП) — физическому адресу памяти, с которого начинается загруженный в память модуль.

Информация в PE-файле является, в основном, высокоуровневой и используется системой или приложениями, чтобы определить правила обращения с конкретным исполняемым файлом.

Перед тем как углубиться в рассмотрение деталей PE-формата, давайте рассмотрим его общую структуру (рис. 5).

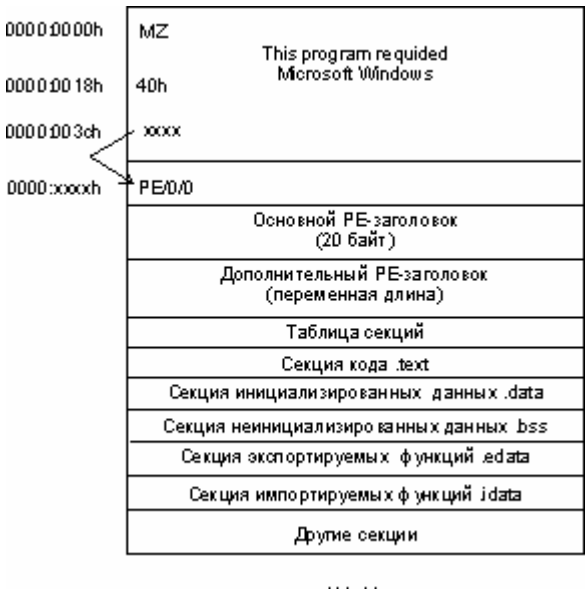

Рис. 5. Общая структура файла PE-формата

Из рис. 5 видно, что в структуре файла PE-формата выделяются следующие составные части:

- программа-заглушка MS-DOS с аналогичными, как в файлах NE-формата, функциями;
- заголовок PE-файла;
- раздел секций;
- область отладочной информации (необязательно).

Рассмотрим их подробнее.

Программа-заглушка MS-DOS

Основные соображения, по которым в начале PE-файла находится программа-заглушка MS-DOS, те же, что и для NE-файла. Ее основная функция — при запуске в среде MS-DOS выдать сообщение типа «This program cannot be run in DOS mode». Сама программа-заглушка выполнена как полноценный MZ-файл, все поля которого обсуждались нами выше. Для нас интерес представляет 32-разрядное поле со смещением `3Ch`. В нем содержится смещение, указывающее на начало PE-файла.

Заголовок PE-файла

Заголовок PE-файла начинается с четырехбайтовой сигнатуры 50 45 00 00h, что соответствует строке «PE» с последующими двумя байтами нулей.

Заголовок PE-файла можно разбить на следующие части:

- сигнатура PE-файла;
- основной заголовок PE-файла (PE header);
- дополнительный заголовок PE-файла (PE optional header).

Формат основного заголовка представлен в табл. 22. Он содержит самую общую информацию о PE-файле.

Таблица 22. Структура основного заголовка PE-файла

Смещение	Длина, байт	Назначение
00h	2	Тип компьютера, для которого создавалась программа. Некоторые возможные значения: 0000h — неизвестный тип компьютерной платформы; 014ch — Intel i80x86; 0160h 0162h, 166, 168 — MIPS; 0184h — DEC Alpha AXP; 01f0h — Power PC; 0268h — Motorola 68000. Эти константы определены во включаемом файле `winnt.h` (Visual C++). Система использует их для организации правильного обращения с файлом
02h	2	Количество секций в PE-файле
04h	4	Время создания файла. Отсчет времени производится в секундах, прошедших с 16:00 31.12.69. (Как насчет проблемы Y2K?)
08h	4	0
0ch	4	0
10h	2	Размер дополнительного заголовка PE-файла (PE optional header). Его размер обычно 224 байта

Смещение	Длина, байт	Назначение
12h	2	Информационные флаги: 0001h — информация о перемещении удалена из файла; 0002h — файл представляет собой исполняемое отображение, то есть не имеет неразрешенных внешних ссылок; 0004h — информация о номерах строк удалена из файла; 0008h — локальные символы удалены из файла; 0080h — расположение байт в машинном слове обратное, то есть соответствует принципу «младший байт по младшему адресу»; 0100h — размер машинного слова 32 бита; 0200h — отладочная информация удалена из исполняемого

	файла в файл **.dbg**; 1000h — системный файл; 2000h —dll-библиотека; и др. Полный перечень флагов приведен во включаемом файле **winnt.h** (Visual C++)

За основным заголовком PE-файла следует дополнительный заголовок. В табл. 23 представлены расположение и значения полей, представляющих наибольший интерес. Все смещения считаются от начала дополнительного заголовка.

Таблица 23. Структура дополнительного заголовка PE-файла

Смещение	Размер, байт	Назначение
+00h	2	Сигнатура, определяющая состояние PE-файла. Возможные значения: 0107h — отображение постоянного запоминающего устройства; 010bh — обычный исполняемый файл
+02h	2	Версия компоновщика, создавшего исполняемый файл
+04h	4	Размер секции кода (**.text**) в байтах
+08h	4	Размер секции инициализированных данных
+0ch	4	Размер секции неинициализированных данных
+10h	4	Адрес (ОАП), с которого начинается выполнение программы модуля. Он находится в секции **.text**
+14h	4	ОАП, с которого начинаются программные секции PE-файла. Если файл был создан компоновщиком Microsoft, то программная секция начинается с адреса 1000h. Компоновщик Borland (Inprise) размещает секцию кода с ОАП 10000h
+18h	4	ОАП, с которого начинаются секции данных PE-файла. Обычно они расположены за секциями кода. Все объекты PE-файла, в том числе секции кода и данных выравниваются на кратную границу: компоновщик Microsoft выравнивает объекты на границу, кратную 4 Кбайт, компоновщик Borland — на границу 64 Кбайт

Таблица 23 (*продолжение*)

Смещение	Размер, байт	Назначение
+1Ch	4	Адрес оперативной памяти, куда предполагается загрузка файла. Этот адрес формирует компоновщик. Если файл действительно загружается по этому адресу, то загрузчику не нужно делать больше никаких настроек, поэтому процесс загрузки происходит быстрее. Если загрузчик не может разместить файл по указанному адресу, то он вынужден произ-

		водить настройку адресов
+20h	4	Значение выравнивания для секции. Каждая секция в PE-файле выравнивается на границу, численное значение которой кратно величине, указанной в данном поле (см. также описание поля 0018h)
+24h	4	Значение кратности, в соответствии с которым выравниваются данные в PE-файле. Смысл этого поля аналогичен полю в NE-файле 0032h (см. табл. 3)
+28	4	Номер самой старой версии (модификации) операционной системы, в среде которой может работать данный файл (обычно 1.0)
+2ch	4	Определяется пользователем с помощью ключа компоновщика **`/VERSION:n.m`** (для **`link.exe`** (MASM)) или **`/Vn.m`** (для **`tlink32.exe`** (TASM)). Значения **`n`** и **`m`** заносятся в два слова этого поля
30h	4	Номера самой старой версии и (модификации) операционной системы Windows NT, в среде которой может работать данный файл (обычно 4.0)
34h	4	0
+38h	4	Размер PE-файла, выровненный на ближайшую границу секции
+3Ch	4	Общий размер всех заголовков в PE-файле, включая заголовок MS-DOS, PE-заголовок, дополнительный PE-заголовок и заголовки PE-секций
+40h	4	Для исполняемых файлов поле равно 0. Для файлов dll-библиотек поле содержит контрольную сумму (**`crc`**)
+44h	2	Тип пользовательского интерфейса, используемого PE-файлом: 0000h — неизвестная подсистема; 0001h — подсистема не требуется (драйвер устройства); 0002h — Windows GUI; 0003h — консольное приложение Windows; 0005h — OS/2; 0007h — Posix
+46h	2	0
+48h	4	Потребный объем памяти для стека
+4ch	4	Выделяемое количество памяти для стека
+50h	4	Потребный объем памяти для локальной кучи
+54h	4	Выделяемое количество памяти для локальной кучи

Смещение	**Размер, байт**	**Назначение**
+58h	4	0
+5Ch	4	Количество элементов массива структур, описывающих местоположение и размер некоторых значимых таблиц и областей исполняемого файла. Значение обычно равно 10h

+60h	10h*2*4	Массив структур, каждая из которых состоит из двух полей (размерность каждого поля **dword**): ОАП расположения данных в PE-файле; размер данных. Реальное наполнение имеют первые 12 структур в данном массиве. Их перечень можно посмотреть во включаемом файле **winnt.h** (Visual C++). Они описывают расположение следующих данных в PE-файле: 0 — таблица экспортируемых функций; 1 — таблица импортируемых функций; 2 — таблица ресурсов; 3 — таблица исключений; 4 — таблица безопасности; 5 — таблица настройки; 6 — таблица отладки; 7 — строки описания; 8 — характеристика скорости компьютера, измеряемая в MIPS (million of instructions per second — миллионов инструкций в секунду); 9 — область TLS (Thread Local Storage, локальная память цепочки); 10 — область таблицы конфигурации; 11 — таблица адресов импорта (Bound Import Directory in headers)

За массивом структур, которым заканчивается заголовок PE-файла, начинается таблица секций.

Таблица секций

Структура PE-файла проще, чем NE-файла. Если не считать программы-заглушки для MS-DOS и необязательной области отладочной информации, то PE-файл состоит из двух больших частей — заголовка и области общих объектов, называемых *секциями*. Секцию можно сравнить с сегментом в структуре NE-файла. Основное отличие секции от сегмента в том, что в PE-файле в виде секции оформляется любой объект, а не только данные или код, как в NE-файле. По этой причине секции иногда называют *объектами*. Информация о всех секциях PE-файла хранится в *таблице секций* (таблице объектов). Таблица секций располагается сразу после заголовка PE-файла. Количество секций определяется по значению поля в начале основного заголовка +2h (см. табл. 22).

Таблица секций состоит из структур размером по 40 байт каждая. Структуры детально описывают соответствующие секции. Поля, составляющие структуру таблицы секций, представлены в табл. 24.

Таблица 24. Содержимое структуры в таблице секций

Смещение	Размер	Назначение
+00h	8 символов	Имя секции (если имя меньше 8 байт, то оно дополнено

		справа нулевыми байтами)
+08h	4 байта	Размер секции (потребное количество памяти)
+0ch	4 байта	ОАП, по которому загрузчик должен загрузить секцию (см. также поле +14h)
+10h	4 байта	Размер секции, выровненный на ближайшую (в большую сторону) границу, в соответствии с размером выравнивания (см. поля +20h и +24h в табл. 22)
+14h	4 байта	ОАП в PE-файле, где находятся данные для секции. Это поле совместно с полем +0ch необходимо для того, чтобы позволить программисту самому управлять загрузкой PE-файла
+18h	12 байтов	0
+24h	4 байта	Флаги, характеризующие атрибуты секции (табл. 25)

В табл. 25 представлены значения флагов в описании секции.

Таблица 25. Значения флагов в описании секции

Значение	**Значение (если бит установлен)**
00000020h	Секция содержит программный код
00000040h	Секция содержит инициализированные данные
00000080h	Секция содержит неинициализированные данные
00000200h	Используется компилятором
00000800h	Используется компилятором
04000000h	Секция не может кэшироваться
08000000h	Секция не страничной организации
10000000h	Совместно используемая секция
20000000h	Секция является исполняемой (см. флаг 00000020h)
40000000h	Секция только для чтения
80000000h	Секция может использоваться для записи

Раздел секций

Вслед за заголовком следует основная часть PE-файла — раздел секций. Необходимо отметить, что компиляторы фирм Microsoft и Borland (Inprise), создающие файлы PE-формата, используют несколько отличающихся по составу и наименованию секций. Для однозначности далее будут обсуждаться секции, создаваемые компиляторами фирмы Microsoft. Но если вы столкнетесь с необходимостью работать с PE-файлами, создаваемыми компиляторами фирмы Borland (Inprise), то по названиям и содержанию секций вы относительно легко сможете провести аналогию и правильную интерпретацию находящихся в них данных. Типичное приложение Windows реально содержит около девяти секций:

- **`.text`** — секция содержит исполняемый код. Так как в 32-разрядном режиме адресации используется сплошная модель памяти, то содержимое секций **`.text`**

всех объектных файлов, подаваемых на вход компоновщика, собирается в одной секции **.text** исполняемого файла PE- формата;

- **.bss** — секция содержит все неинициализированные данные из секций **.bss** компонуемых объектных файлов, включая все переменные, объявленные как статические в пределах функций или исходного модуля;
- **.rdata** — секция содержит данные, доступные только для чтения: литеральные строки, константы и отладочную информацию;
- **.data** — секция содержит инициализированные и глобальные переменные (кроме динамических локальных переменных, которые располагаются в стеке). В основном, это инициализированные и глобальные данные приложения, собранные из всех секций **.data** компонуемых объектных файлов. Делается это так же, как и для кода, который собирается в секции **.text**;
- **.rsrc** — секция содержит информацию о ресурсах приложения. Эта секция начинается с каталога ресурса, и данные этой секции структурированы в дерево ресурсов (см. ниже информацию об описании ресурсов в PE-файле);
- **.edata** — секция содержит данные об экспортируемых функциях приложения или dll-библиотеки. В начале секции расположен каталог для получения доступа к информации об экспортируемых функциях (см. ниже);
- **.idata** — секция содержит данные об импортируемых приложением функциях (см. ниже);
- **.debug** — секция содержит отладочную информацию. Формат PE-файла также поддерживает работу с отдельным файлом отладки (обычно имеющим расширение **.dbg**), в котором собирается вся отладочная информация.

Рассмотрим более подробно некоторые секции. При этом раскроем некоторые механизмы, на которых основана работа платформы Win32. С секциями кода и данных особых вопросов не возникает, так как их содержание зависит от конкретного приложения. Другие секции имеют более формализованную структуру.

Секция описания ресурсов PE-файла

Описание и расположение ресурсов в PE-файле отличаются от того, как это было реализовано в NE-файлах. Сам формат конкретного ресурса остался практически неизменным (детально они описаны в файле **resfmt.txt** из Win32 SDK), но теперь все ресурсы сведены в сложную иерархию. Отдаленно эта иерархия напоминает структуру каталога диска (рис. 6). Основу этой иерархии составляют два типа элементов, которые, если следовать аналогии с файловой системой, описывают каталоги (типы ресурсов) и файлы (двоичные описания ресурсов). В отличие от структуры оглавления диска, иерархия описания ресурсов имеет четыре уровня. На первых трех уровнях находятся элементы одинаковой структуры, а на четвертом находятся элементы, описывающие конкретный ресурс и имеющие соответствующую описанию этого ресурса структуру.

В начале секции **.rsrc** находится элемент первого уровня иерархии, через который можно попасть на ветвь дерева каталога, описывающую нужный тип ресурса: меню,

окно диалога, растровые изображения и т. д. (табл. 26).

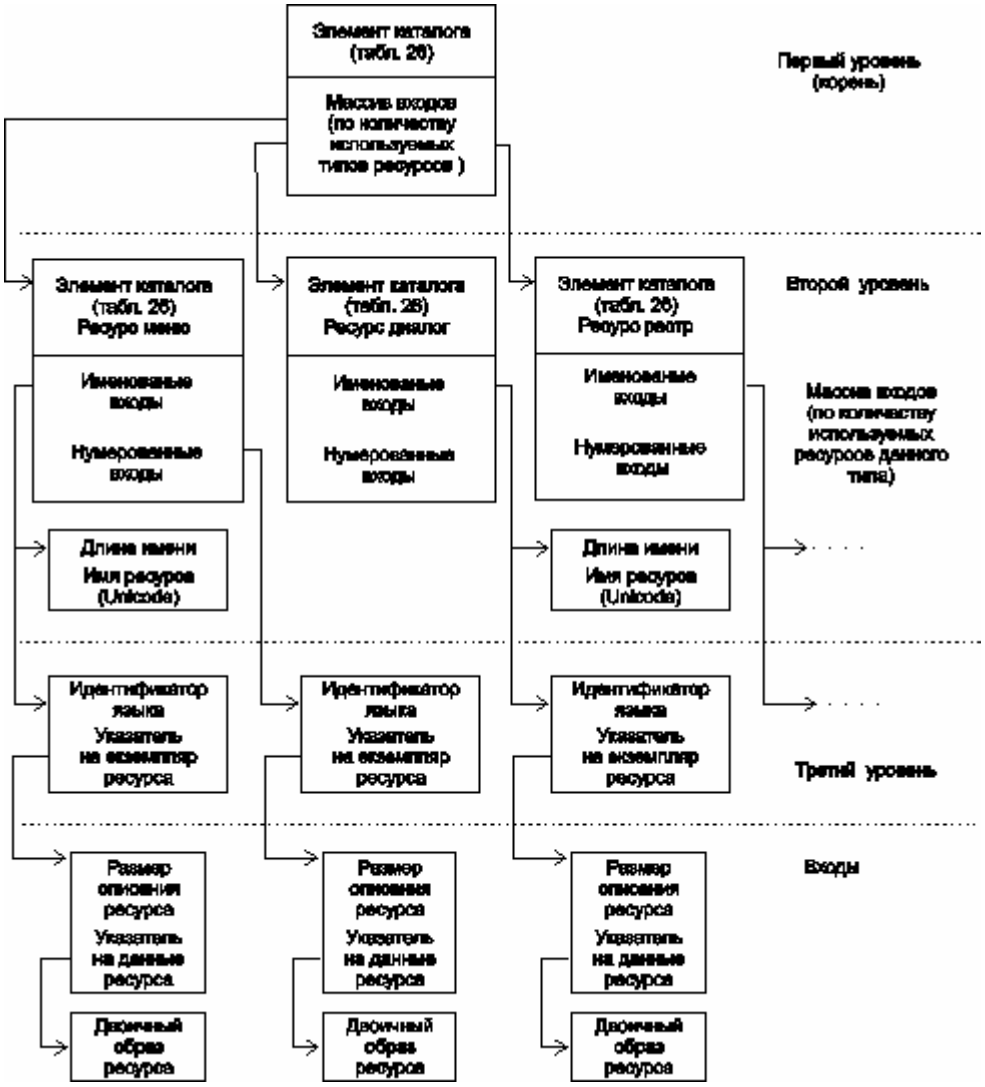

Рис. 6. Пример иерархической структуры описания ресурсов в PE-файле

В начале секции `.rsrc` находится элемент первого уровня иерархии, через который можно попасть на ветвь дерева каталога, описывающую нужный тип ресурса: меню, окно диалога, растровые изображения и т. д. (см. табл. 26).

За этим элементом следует массив элементов, количество которых равно значению последнего поля табл. 26. Эти элементы имеют структуру, показанную в табл. 27.

Элементы второго уровня описывают экземпляры ресурсов определенного типа. При этом различаются именованные и неименованные (нумерованные) ресурсы. Рассмотрим формат элементов каталога. Они имеют структуру, показанную в табл. 28.

Таблица 26. Структура элемента первого уровня иерархии ресурсов

Смещение	Размер, байт	Назначение
+00h	4	Характеристики ресурса или 0
+04h	4	Время и дата создания ресурса
+08h	4	0
+0Ch	2	0
+0Eh	2	Общее количество типов ресурсов, используемых в данном исполняемом файле

Таблица 27. Структура элемента массива, следующего за элементом каталога первого уровня

Смещение	Размер, байт	Назначение
+00h	4	Тип ресурса
+04h	4	Младшие 31 бит представляют собой значение смещения от начала секции ресурсов до структуры с описанием элемента каталога для данного типа ресурсов

Таблица 28. Структура элемента второго уровня иерархии ресурсов

Смещение	Размер, байт	Назначение
+00h	4	0
+04h	4	Время и дата создания ресурса
+08h	4	0
+0ch	2	Количество поименованных элементов, следующих за данным экземпляром структуры (для корневого элемента это поле равно 0)
+0Eh	2	Количество элементов, идентифицируемых целым значением и следующих за данным экземпляром структуры (для корневого элемента это поле равно общему количеству типов ресурсов, используемых в данном исполняемом файле)

За элементом каталога ресурсов второго уровня следует массив элементов, количество которых равно сумме значений двух последних полей структуры элемента каталога (см. табл. 28), то есть количества поименованных и нумерованных

элементов. Эти элементы упорядочены по именам и номерам, соответственно. Формат этих элементов показан в табл. 29.

Последнее поле в табл. 28 содержит указатель на структуру третьего уровня иерархии. Эта структура имеет формат элемента каталога (см. табл. 26 и 28), но массив элементов, следующий за данной структурой, имеет один элемент, структура которого показана в табл. 30.

Таблица 29. Структура элемента массива, следующего за элементом каталога второго уровня

Смещение	Размер, байт	Назначение
+00h	4	Интерпретация этого поля зависит от состояния его старшего бита (80000000h). Если старший бит нулевой, то это поле является целочисленным идентификатором ресурса. Если старший бит единичный, то это поле содержит смещение (по отношению к началу секции ресурсов) к структуре, содержащей имя ресурса. Эта структура, в свою очередь, состоит из двух полей: первые два байта содержат количество символов в имени ресурса, а далее следуют символы имени, закодированные (!) в Unicode
+04h	4	Значение смещения от начала секции ресурсов до структуры с описанием языка

Таблица 30. Структура элемента массива, следующего за элементом каталога третьего уровня

Смещение	Размер, байт	Назначение
+00h	4	Идентификатор языка
+04h	4	Значение смещения от начала секции ресурсов до структуры с описанием ресурса

Идентификатор языка представляет собой предопределенную константу из файла **winnt.h**. Ниже приведен фрагмент из этого файла с определениями констант языка.

```
//Language IDs.
//The following two combinations of primary language ID and
//sublanguage ID have special semantics:
//  Primary Language ID Sublanguage ID Result
//  LANG_NEUTRAL SUBLANG_NEUTRAL Language neutral
//  LANG_NEUTRAL SUBLANG_DEFAULT User default language
//LANG_NEUTRAL SUBLANG_SYS_DEFAULT System default language
//Primary language IDs.
#define LANG_NEUTRAL           0x00
#define LANG_AFRIKAANS  0x36
#define LANG_ALBANIAN          0x1c
… … …
#define LANG_ENGLISH           0x09
… … …
```

```
//Sublanguage IDs.
//The name immediately following SUBLANG_ dictates which primary
//language ID that sublanguage ID can be combined with to form a
//valid language ID.
//
… … …
#define SUBLANG_ENGLISH_CAN    0x04 // English (Canadian)
… … …
```

Второе поле элемента массива третьего уровня иерархии (см. табл. 30) содержит указатель на структуру, с которой начинается непосредственное описание ресурса (табл. 31).

Таблица 31. Структура описания ресурса

Смещение	Размер, байт	Назначение
+00h	4	Размер памяти, занимаемой двоичным образом ресурса
+04h	4	Значение смещения от начала секции ресурсов до области памяти, содержащей двоичное описание ресурса

В качестве примера покажем содержимое секции описания ресурсов некоторого приложения. Допустим, что это приложение использует следующие ресурсы: два меню, одно из которых (**`MyMenu`**) является именованным ресурсом, а другое идентифицировано целочисленным значением (5), одно окно диалога (именованный ресурс) — **`DialogWin`** и растровое изображение (также именованный ресурс) — **`BitImage`**. Таким образом, имеется три именованных ресурса и один ресурс, идентифицированный целочисленным значением. Тогда содержимое секции ресурсов данного исполняемого файла будет представлять собой дерево, изображенное на рис. 6.

Секция описания импортируемых функций PE-файла

Технике импорта функций в PE-файле следует уделить особое внимание, так как именно эти исполняемые файлы работают на платформе Win32. Данная платформа, как известно, обеспечивает работу 32-разрядных версий операционной системы Windows. Особенность работы приложений в этих операционных системах заключается в том, что они вынуждены, помимо своей внутренней работы, связанной с обработкой некоторых данных, постоянно обращаться к операционной системе за различными сервисами (подробнее этот вопрос рассмотрен на уроке 18). Физически такое обращение осуществляется в форме вызова функций, которые разбросаны по различным системным dll-библиотекам. В принципе, программист может разработать и свои dll-библиотеки. Информация о том, какие функции и из каких библиотек используются в данном файле, содержится в секции **`.idata`**. Эту секцию называют также *таблицей импорта*. С точки зрения структуры секция **`.idata`** состоит из трех частей:

- массив с описанием используемых dll-библиотек — **`ArrayDllDef`**;

- два массива с адресами импортируемых функций — **ArrayAdressImpFunc**;
- массив с описанием имен импортируемых функций — **ArrayNameImpFunc**.

Эти четыре массива расположены в секции **.idata** последовательно друг за другом. Массив с описанием используемых dll-библиотек **ArrayDllDef** предназначен для указания местоположения в секции **.idata** имен dll-библиотек, массивов с адресами импортируемых функций **ArrayAdressImpFunc** для каждой из используемых dll-библиотек. Массив **ArrayDllDef** состоит из 20-байтовых элементов, структура которых показана в табл. 32.

Таблица 32. Элементы массива ArrayDllDef

Смещение	Размер, байт	Назначение
+00h	4	Значение смещения от начала секции **.idata** к началу массива с адресами импортируемых функций **ArrayAdressImpFunc** (первая копия)
+04h	4	0
+08h	4	0
+0ch	4	Значение смещения от начала секции **.idata** до начала ASCIIZ строки с именем dll-библиотеки
+10h	4	Значение смещения от начала секции **.idata** к началу массива с адресами импортируемых функций **ArrayAdressImpFunc** (вторая копия)

Количество элементов массива **ArrayDllDef** равно количеству dll-библиотек, функции которых импортируются данным приложением. Конец массива **ArrayDllDef** обозначается элементом с нулевыми полями (его общий размер 20 байт). За этим массивом следуют массивы с адресами импортируемых функций **ArrayAdressImpFunc** для каждой из dll-библиотек. Количество этих массивов равно количеству dll-библиотек. Для удобства обсуждения нужно рассматривать совокупность этих массивов, как единое целое. Тогда можно говорить, что секция **.idata** PE-файла содержит две копии этой совокупности массивов **ArrayAdressImpFunc**. Смещение относительно начала секции **.idata** к началу каждого массива в первой копии совокупности массивов **ArrayAdressImpFunc** для конкретной dll-библиотеки содержится в поле со смещением **+00h** в соответствующем элементе массива **ArrayDllDef**. Смещение к началу каждого массива **ArrayAdressImpFunc** во второй копии находится в поле со смещением **+10h** в элементе массива **ArrayDllDef** для соответствующей dll-бибилиотеки. Элемент массива **ArrayAdressImpFunc** представляет собой двойное слово, содержимое которого соответствует импортируемой функции и зависит от того, в какой момент времени оно рассматривается и в какой копии совокупности массивов **ArrayAdressImpFunc** оно содержится. Ответы на эти вопросы следуют из ответа на вопрос: а зачем вообще нужны две копии совокупности массивов **ArrayAdressImpFunc**?

В полной мере необходимость в двух копиях массивов **ArrayAdressImpFunc**

проявляется лишь после того, как файл загружен в память для выполнения. До этого, пока PE-файл находится на диске, содержимое этих двух копий абсолютно одинаково, что вы можете наблюдать, открыв файл для просмотра в шестнадцатеричном виде. Каждый элемент массива **ArrayAdressImpFunc** в виде двойного слова соответствует одной импортируемой функции и является смещением относительно начала секции **.idata** к элементу массива **ArrayNameImpFunc**. В массиве **ArrayName-ImpFunc** содержатся имена импортируемых функций, их номера экспорта (см. далее) из соответствующих dll-библиотек. Конец каждого массива **ArrayAdressImpFunc** обозначается нулевым двойным словом. Массивы **ArrayAdressImpFunc** в первой копии не изменяются загрузчиком при загрузке в память файла для исполнения. Массивы **ArrayAdressImpFunc** во второй копии изменяются в процессе загрузки следующим образом. Загрузчик последовательно просматривает элементы массива **ArrayAdressImpFunc**, по содержащимся в них смещениям относительно начала секции **.idata** находит адреса (или экспортные номера) функций в соответствующих dll-библиотеках и замещает содержимое двойных слов на истинные значения адресов импортируемых функций в памяти. Таким образом, массивы **Array-AdressImpFunc** в первой копии не являются, строго говоря, массивами адресов, и вообще, будьте готовы к тому, что компиляторы фирм, отличных от Microsoft, могут создавать PE-файлы с одной копией массивов **ArrayAdressImpFunc**. Судя по вышесказанному, необходимость в наличии двух копий представляется сомнительной, если только за этим не скрывается нечто большее.

Как уже было отмечено выше, в массиве **ArrayNameImpFunc** хранятся имена и экспортные номера функций, используемых в данном исполняемом файле. Элемент этого массива имеет формат, показанный в табл. 33.

Таблица 33. Структура элемента массива ArrayNameImpFunc

Смещение	Размер, байт	Назначение
+00h	2	Номер экспорта из dll-библиотеки для данной импортируемой функции
+02h	?	ASCIIZ-строка с именем импортируемой функции

Для чего нужна такая схема формирования адресов импортируемых функций? Дело в том, что вызов импортируемых функций в исполняемом файле осуществляется командами **jmp** или **call**. В качестве адресов перехода в этих командах используются адреса элементов во второй копии массивов **ArrayAdressImpFunc**, соответствующих импортируемым функциям. Как мы уже отметили ранее, в элементах этого массива загрузчиком были сформированы истинные адреса этих функций в памяти. То есть элементы массива **ArrayAdressImpFunc** играют роль переходников между воображаемыми и истинными адресами импортируемых функций.

Секция описания экспортируемых функций PE-файла

Файл PE-формата может содержать секцию **.edata**, в которую помещается информация об экспортируемых данным файлом функциях. Обычные приложения редко содержат подобную секцию. Ее наличие характерно для dll-библиотек, в которых экспорт функций является основной задачей. Поэтому для ознакомления с данным материалом откройте в шестнадцатеричном виде любой файл с расширением **.dll** и найдите в нем секцию **.edata**.

Секция **.edata** состоит из заголовка и трех массивов, указатели на которые содержатся в заголовке секции. Структура оглавления показана в табл. 34.

Таблица 34. Структура оглавления секции .edata

Смещение	Размер, байт	Назначение
+00h	4	0
+04h	4	Время и дата создания файла или 0
+08h	4	0

Таблица 34 (*продолжение*)

Смещение	Размер, байт	Назначение
+0ch	4	Смещение относительно начала секции **.edata** к ASCIIZ-строке с именем данной dll-бибилиотеки
+10h	4	Начальное значение, с которого начинаются экспортные номера функций данной dll-библиотеки
+14h	4	Количество элементов в массиве адресов экспортируемых функций **ArrAdrExpFunc**
+18h	4	Количество элементов в массиве имен экспортируемых функций **ArrNameExpFunc**
+1ch	4	Смещение относительно начала секции **.edata** к массиву **ArrAdrExpFunc**. Этот массив содержит смещения относительно начала PE-файла точек входа для каждой экспортируемой функции
+20h	4	Смещение относительно начала секции **.edata** к массиву **ArrNameExpFunc**, который содержит указатели на строки с именами экспортируемых функций
+24h	4	Смещение относительно начала секции **.edata** к массиву **ArrNumExpFunc**, который содержит слова с номерами экспортируемых функций. Для получения истинного экспортного номера функции эти номера необходимо складывать со значением в поле +10h оглавления секции **.edata**

Центральное место в секции **.edata** занимает массив двойных слов **ArrAdrExpFunc**, содержащий адреса экспортируемых функций в PE-файле. Записи в этом массиве упорядочены в соответствии с экспортными номерами функций в dll-библиотеке. Фактически, номер функции является индексом в массиве **ArrAdrExpFunc**. И если есть разрывы в нумерации функций, то пропущенным номерам будут

соответствовать нулевые элементы в таблице **ArrAdrExpFunc**.

За массивом **ArrAdrExpFunc** по адресу, находящемуся в оглавлении секции (поле со смещением **+10h**), расположен массив двойных слов **ArrNameExpFunc**, который содержит смещения относительно начала секции **.edata** к ASCIIZ-строкам с именами экспортируемых функций. Этот массив следует рассматривать совместно с массивом слов **ArrNumExpFunc** (его адрес в поле со смещением **+24h**), в котором содержатся номера экспортируемых функций. Как уже было отмечено выше, нумерация экспортируемых функций не обязательно должна быть линейной. Значения слов в массиве **ArrNumExpFunc** являются индексами в массиве **ArrAdrExpFunc**. Что же касается самих массивов **ArrNameExpFunc** и **ArrNumExpFunc**, то между ними существует взаимно однозначное соответствие, так как они содержат одинаковое количество элементов. Первый элемент массива **ArrNumExpFunc** является номером функции с именем (если оно есть), доступ к которому можно получить, используя смещение, содержащееся в первом элементе массива **ArrNameExpFunc**. При этом не забывайте, что для получения истинного номера экспортируемой функции к значениям, содержащимся в элементах массива **ArrNumExpFunc**, необходимо прибавлять значение в поле со смещением **+10h** заголовка секции **.edata**. Такая схема организации секции **.edata** позволяет экспортировать функции по их именам и номерам.

На этом мы закончим описание форматов исполняемых файлов. Нам не удалось обсудить все тонкости, но это и не являлось нашей конечной целью. Главное, чего мы достигли, — это еще большей уверенности в своих силах. По сути, компьютер — это более или менее «большой кусок (хотя и высокотехнологичный) железа». Магического в нем на самом деле очень мало. Его поведение (по крайней мере на сегодняшнем этапе технологического развития) полностью зависит от управляющей им программы. Настоящий урок показал нам возможные внутренние форматы этой программы. Разобравшись с ними, вы, может быть, даже почувствовали в себе прилив творческих сил и захотели написать нечто похожее на собственный дизассемблер или, на худой конец, утилиту, исследующую и корректирующую внутренности исполняемых файлов. Только большая просьба к вам: не нужно увлекаться написанием вирусов. Займитесь лучше изучением материала следующих уроков — они достойны вашего внимания.

Подведем некоторые итоги:

- ☑ Работая с операционными системами фирмы Microsoft, пользователь может иметь дело с четырьмя типами исполняемых файлов. Изучая их, мы фактически можем проследить, как эволюционировали операционные системы этой фирмы.
- ☑ Самый простой формат исполняемого файла — файл com-формата. Данный тип исполняемого файла достался по наследству от 8-разрядной операционной системы CP/M. Ему свойственны жесткие требования к объему занимаемой памяти и оформлению исходного текста программы.
- ☑ По мере развития аппаратных возможностей компьютера на базе микропроцессоров Intel и для удовлетворения возросших потребностей программ,

был разработан формат исполняемых файлов MS-DOS. Этот формат учитывает требования по сегментации памяти и при соблюдении этих требований дает возможность разрабатывать достаточно большие программы.

☑ Появление операционной системы Windows потребовало внесения изменений в формат исполняемого файла. Основных причин было несколько. Например, появились новые объекты (ресурсы), поддержка которых была необходима на уровне операционной системы. Также со стороны микропроцессора появились новые возможности: дополнительные режимы работы, аппаратная поддержка многозадачности, организация защиты на основе привилегий и т. д. Старый формат исполняемого файла для операционной системы MS-DOS даже приблизительно не мог обеспечить все эти возможности.

☑ Windows 3.1 работает с исполняемыми файлами NE-формата (New Executable — новый исполняемый файл). Этот формат полностью обеспечивает потребности 16-разрядной платформы — Win16, хотя и имеет ряд недостатков.

☑ Наиболее перспективным форматом исполняемого файла стал PE-формат. Источником некоторых реализованных в нем идей явился формат исполняемого файла COFF, который существует на операционной платформе UNIX. Достоинство файла PE-формата в том, что он в значительной степени реализует возможности 32-разрядного программирования. Перечислим некоторые из этих достоинств.

- Простота PE-формата. В NE-файле смещения различных объектов формата хранятся либо в смещениях относительно начала NE-заголовка, либо в значениях в виде величины сектора. В PE-файле все смещения — это ОАП относительно начала отображения файла в памяти. Такая реализация упрощает работу загрузчика операционной системы. Он загружает файл на заранее известное место в оперативной памяти, поэтому его действия по настройке программы минимальны.
- Нет атрибутов для объектов различной природы типа **`PRELOAD`**.
- Организация расположения ресурсов в исполняемом файле более структурирована, что повышает удобство работы с ними.