01 PE File Format

1. Portable Executable; PE

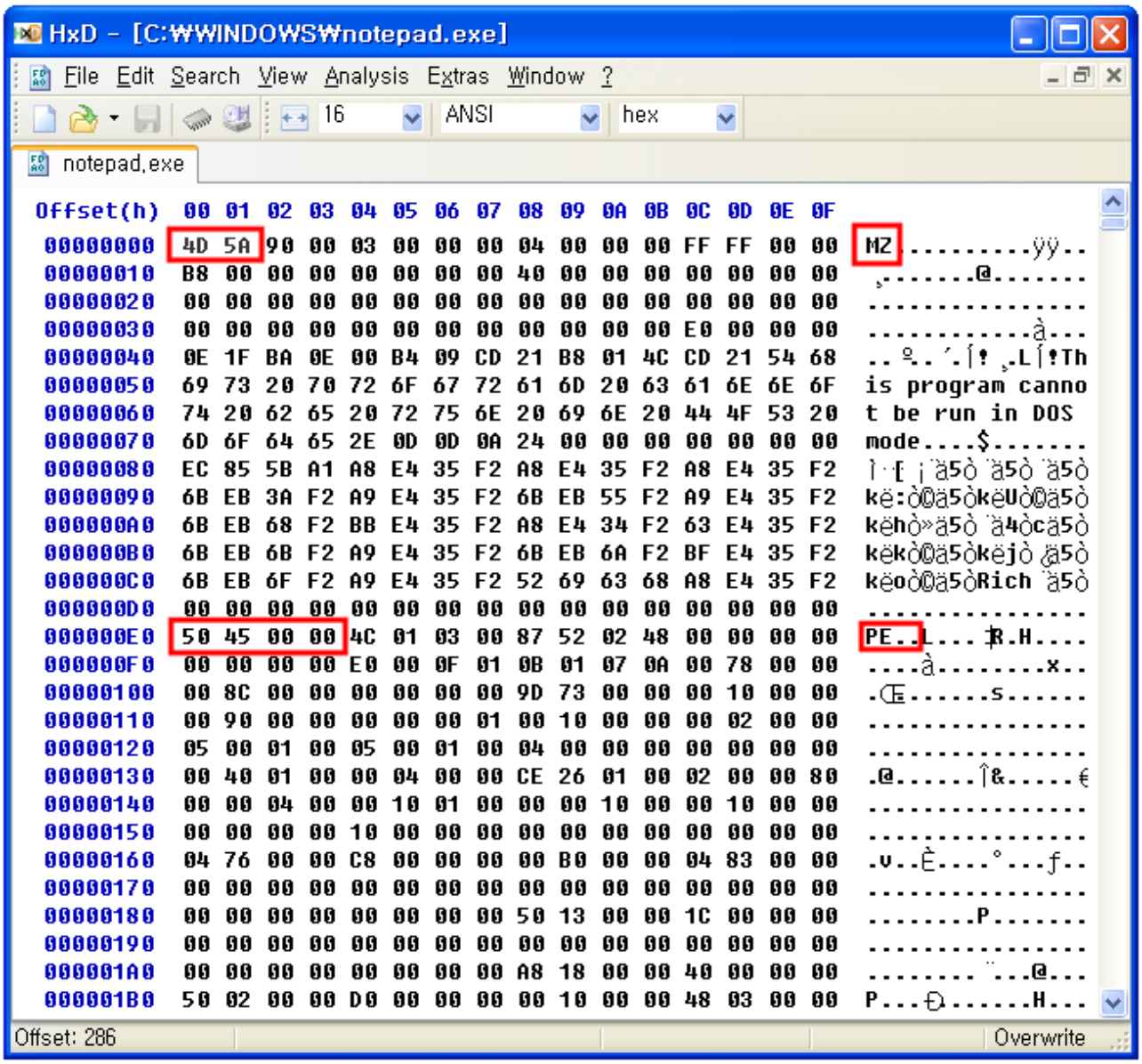
- Windows 운영체제에서 사용되는 실행 파일 형식
- 기존 Unix에서 사용되는 COFF(Common Object File Format)를 기반으로 MS에서 만듬듬

2. Format

- 실행 계열
  - EXE, SCR
- 라이브러리 계열
  - DLL, OCX, CPL, DRV
- 드라이버 계열
  - SYS, VXD
- 오브젝트 파일 계열
  - OBJ

실행 계열	.scr	- Screen Saver의 약자. 윈도우 화면 보호기 파일. 이 파일은 혼자서도 실행된다. - 윈도우 디렉터리나 System 디렉터리에 설치되며, 디스플레이 등록정보에서 설정하여 사용한다.
	.exe	- Executable의 약자로 가장 기본적인 실행파일. - 그 외 실행 파일로는 .bat, .com등이 있다.
드라이버 계열	.sys	- System의 약자로 시스템 운영에 꼭 필요한 파일이며, config.sys, msdows.sys처럼 부팅과 관련이 있거나 운영체제의 시스템에 관련된 내용이 들어 있다.
	.vxd	- Virtual Device Driver(가상 디바이스 드라이버 파일)의 확장자로 하드웨어 또는 소프트웨어의 동작을 관리한다.
라이브러리 계열	.dll	- 동적 링크 라이브러리(dynamic link lirary)의 약자로 독립된 개체들을 하나로 종합한 라이브러리 파일이다.
	.ocx	- 프로그램에서 실행에 필요한 기능을 한데 모아 놓은 파일로 dll 파일과 비슷하다.
	.drv	- driver의 약자. PC에 설치된 수많은 하드웨어의 인식과 구동을 제어하는 디바이스 드라이버 파일로 데이터가 들어있는 파일이다.
오브젝트 계열	.obj	- 컴파일되었지만 링크되지 않은 개체 파일

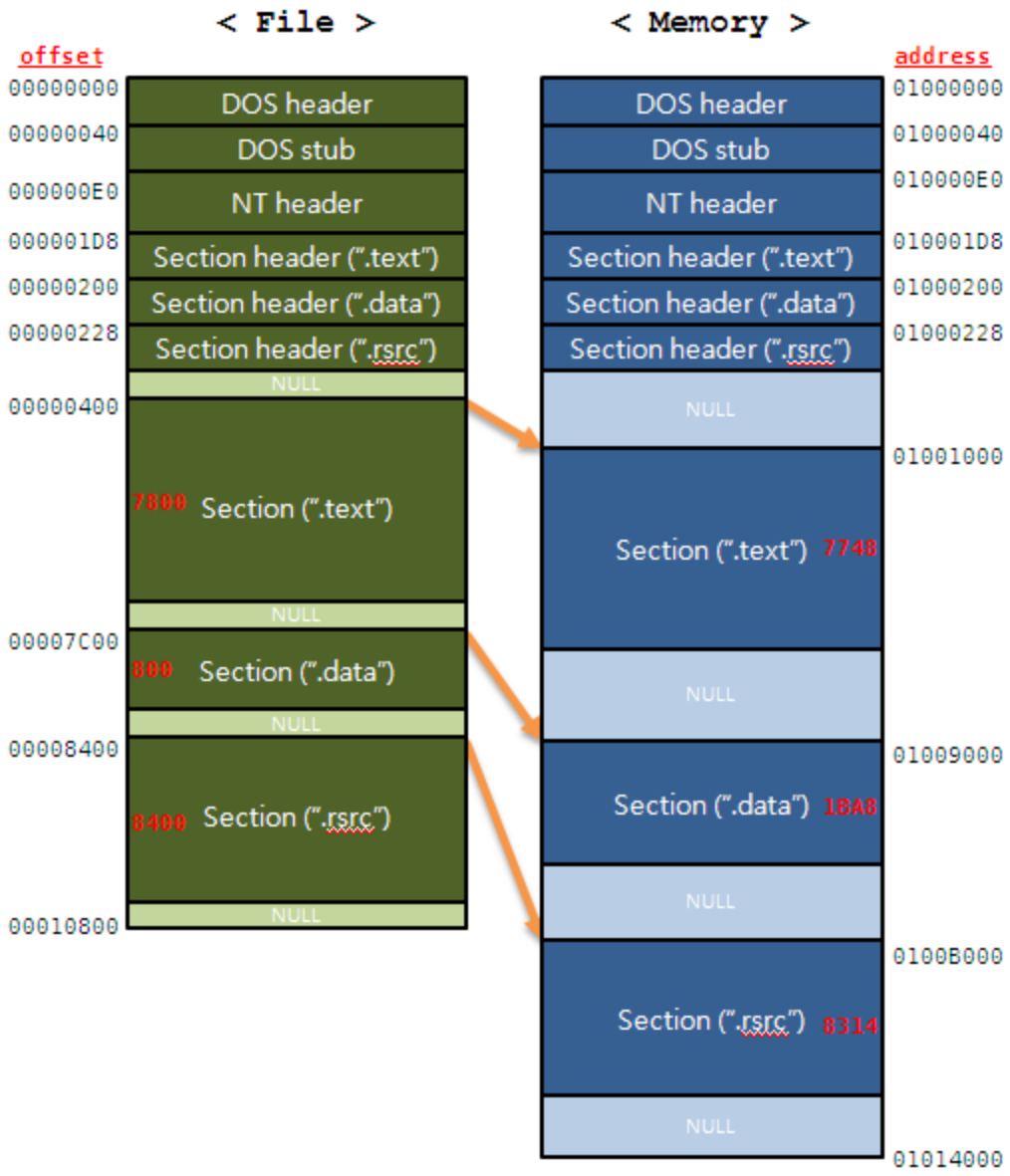
1) Notepad.exe



PE file의 헤더부분이며, 노트패드가 실행되기 위해 필요한 모든 정보가 적혀 있다.

2) PE Basic Struct

- PE Header : DOS header ~ Section header
- PE Body : Section header 이후 Section Group
- 주소 위치 표현
  - File : Offset
  - Memory : Virtual Address; VA, 절대 주소
- 파일이 메모리에 로딩되면 모양이 달라 진다. (Section의 크기, 위치 등)
  - 파일의 내용은 각 데이터에 따라 여러 섹션에 저장됨
    - 코드(.text)
    - 데이터(.code)
    - 리소스(.rsrc)



- Section header
  - 각 Section에 대한 파일/메모리에서 크기, 위치, 속성 등이 정의
- PE header의 끝 부분과 각 섹션의 끝에는 NULL padding이라고 불리우는 영역이 존재합니다.
  - 컴퓨터에서 파일, 메모리, 네트워크 패킷 등을 처리할 때 효율을 높이기 우해 최소 기본 단위 개념을 사용하는 것과 같은 개념
  - 파일/메모리에서 섹션의 시작 위치는 각 파일/메모리의 최소 기본 단위의 배수에 해당하는 위치여야 하고, 빈공간을 NULL로 채워버린다.
    - 만약 100이라는 사이즈를 최소사이즈 10단위라면 총 10개의 Section으로 구분되며,

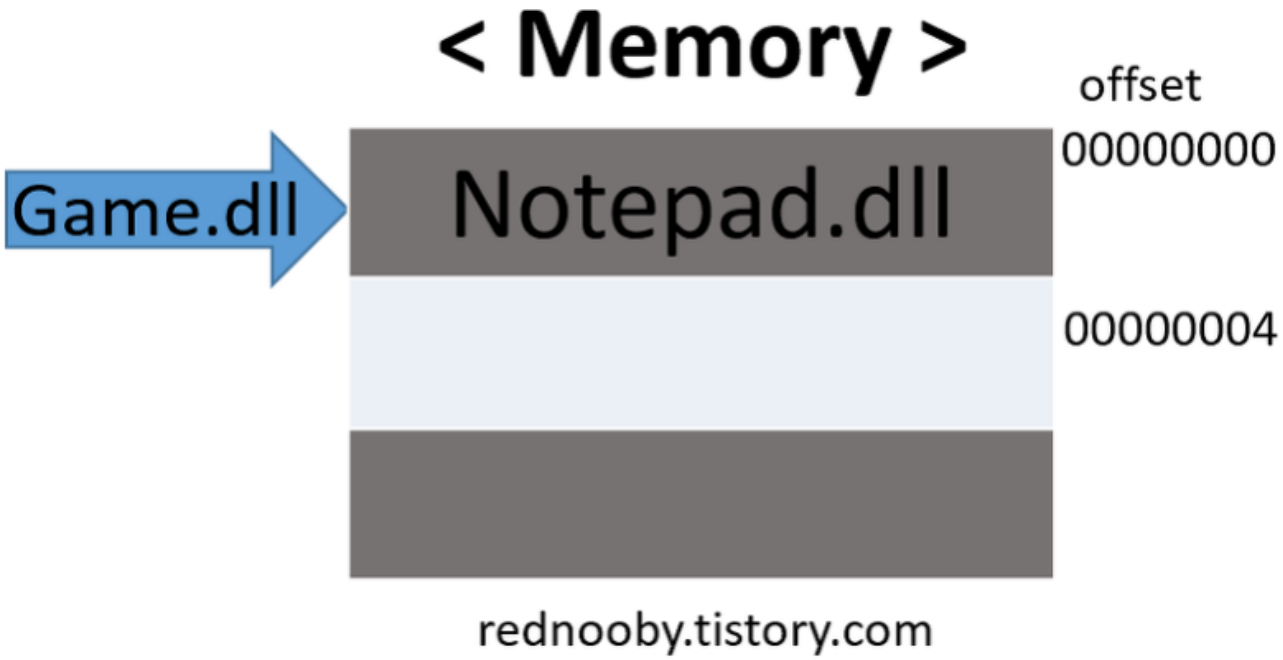
각 Section에서 10이라는 사이즈 만큼 데이터를 채우지 못한다면 NULL로 채운다는 뜻이다.

3) VA & RVA

> *Virtual Address; VA : 프로세스 가상 메모리의 절대 주소*
> *Relative Virtual Address; RVA : 어느 기준 위치(Image Base; 0040000)에서부터의 상대 주소*
> *RVA + ImageBase = VA*

(1) PE 헤더 내의 정보는 RVA 형태로 된 것이 많다.

- PE 파일(주로 DLL)이 프로세스 가상 메모리의 특정 위치에 로딩되는 순간 이미 그 위치에 다른 PE 파일이 로딩되어 있을 수 있다.
  - 이 때, 재배치(Relocation) 과정을 통해서 비어 있는 다른 위치에 로딩



[재배치(Relocation) 상황]

3. 절대 / 상대 주소

> *절대주소 : 메모리의 위치를 식별하는 메모리 고유 주소*
> *(즉, 기억장치 고유의 주소, 기억 장소를 직접 숫자로 지정)*
> *상대 주소 : 고유 주소가 아니며 특정영역에 상대적인 주소를 지정*

[물리주소와 가상주소]

빨간색 : 사용중 | 파랑, 초록 : 미 사용중, 임의의 설명용 그림이다.

- 물리주소의 0x01~0x03, 0x07이 사용 불능
- 프로세스는 연속적인 메모리를 사용해야 한다.
  - 0x00~0x04까지의 주소를 사용하는 프로세스가 있을 때, 물리영역 0x04 ~ 0x06, 0x08 ~ 0x09 메모리 영역은 연속되지 않기 때문에 사용할 수 없다.
    - 그러므로 사용가능한 메모리 영역을 찾기보다는 가상의 주소로 맵핑하여 연속된 메모리를 사용한 것 처럼 사용된다.
- 가상주소는 0x00000000 ~ 0xFFFFFFFF 까지 이므로, 한 프로세스는 4byte 메모리 주소를 사용새야 합니다.

---

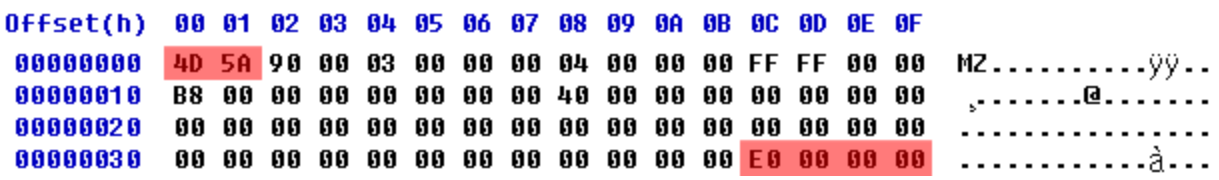
4. PE Header

1) DOS Header

- MS는 PE File format을 만들 때, DOS 파일에 대한 하위 호환성을 고려해서 만들었다.
  - PE헤더 맨 앞부분에는 기존 DOS EXE Header를 확장시킨 IMAGE_DOS_HEADER 구조체가 존재하며, IMAGE_DOS_HEADER는 DOS EXE Header를 확장한 것이다.

```
typedef struct _IMAGE_DOS_HEADER {
    WORD   e_magic;          // DOS signature : 4D5A ("MZ")
    WORD   e_cblp;
    WORD   e_cp;
    WORD   e_crlc;
    WORD   e_cparhdr;
```

```
    WORD   e_minalloc;
    WORD   e_maxalloc;
    WORD   e_ss;
    WORD   e_sp;
    WORD   e_csum;
    WORD   e_ip;
    WORD   e_cs;
    WORD   e_lfarlc;
    WORD   e_ovno;
    WORD   e_res[4];
    WORD   e_oemid;
    WORD   e_oeminfo;
    WORD   e_res2[10];
    LONG   e_lfanew;          // offset to NT header, 가변적인 값
을 가짐
  } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
* MS Visual C++ : winnt.h
* _IMAGE_DOS_HEADER의 사이즈는 64(0x40)입니다.
```



- 파일 시작 2byte는 4D5A이며, e_lfanew 값은 000000E0입니다.

---

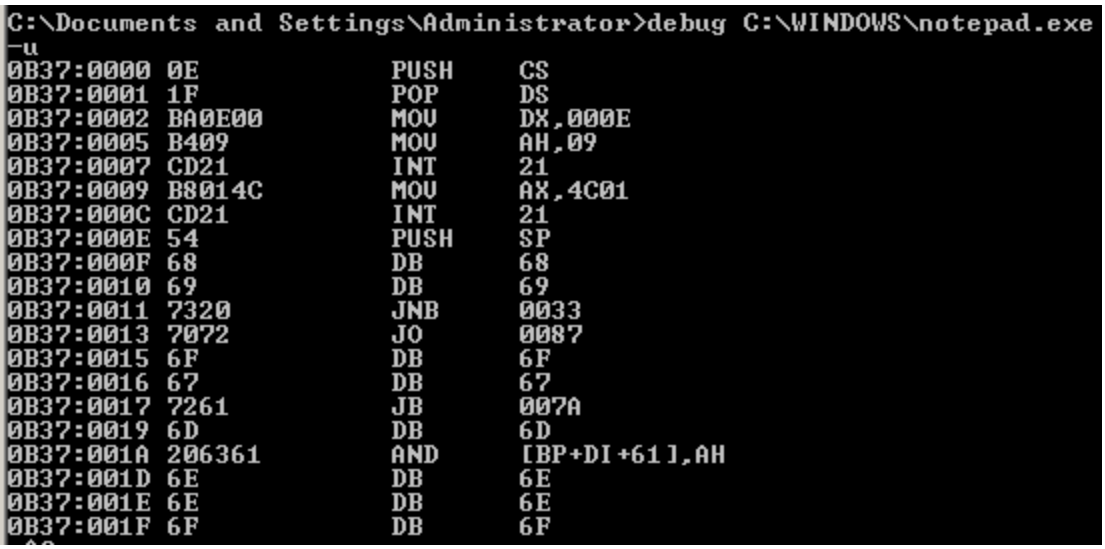
2) DOS Stub

> *해당 데이터의 존재여부는 옵션이며, 크기도 일정하지 않습니다.*
> *코드와 데이터의 혼합으로 이루어져 있습니다*

pFile	Raw Data	Value
00000040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	........!..L.!Th
00000050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
00000060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00000070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....$.......
00000080	02 22 E8 1F 46 43 86 4C 46 43 86 4C 46 43 86 4C	."..FC.LFC.LFC.L
00000090	4F 3B 15 4C 16 43 86 4C 1D 2B 82 4D 51 43 86 4C	O;.L.C.L.+.MQC.L
000000A0	1D 2B 83 4D 40 43 86 4C 1D 2B 85 4D 43 43 86 4C	.+.M@C.L.+.MCC.L
000000B0	1D 2B 87 4D 4F 43 86 4C 46 43 87 4C 62 42 86 4C	.+.MOC.LFC.LbB.L
000000C0	1D 2B 8F 4D 5F 43 86 4C 1D 2B 79 4C 47 43 86 4C	.+.M_C.L.+yLGC.L
000000D0	1D 2B 84 4D 47 43 86 4C 52 69 63 68 46 43 86 4C	.+.MGC.LRichFC.L
000000E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

- 40~4D 영역은 16bit Assembly command

(1) XP → C:\WINDOWS\notepad.exe → u



  - 화면 문자열(“This program cannot be run in DOS mode”) 출력 후 종료

---

3) NT Header

```
typedef struct _IMAGE_NT_HEADERS {
  DWORD                   Signature; // PE Signature : 50450000 ("PE"00)
  IMAGE_FILE_HEADER       FileHeader;
  IMAGE_OPTIONAL_HEADER32 OptionalHeader;
```

```
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

pFile	Raw Data	Value
000000F0	50 45 00 00 4C 01 06 00 17 2C A4 2F 00 00 00 00	PE..L....,./....
00000100	00 00 00 00 E0 00 02 01 0B 01 0E 0F 00 FC 01 00	....
00000110	00 74 00 00 00 00 00 00 B0 F8 01 00 00 10 00 00	.t..............
00000120	00 10 02 00 00 00 40 00 00 10 00 00 00 02 00 00	......@.........
00000130	0A 00 00 00 0A 00 00 00 0A 00 00 00 00 00 00 00	....
00000140	00 B0 02 00 00 04 00 00 40 C5 02 00 02 00 40 C1	........@.....@.
00000150	00 00 04 00 00 10 01 00 00 00 10 00 00 10 00 00	....
00000160	00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00	....
00000170	B8 34 02 00 70 03 00 00 00 70 02 00 E0 0B 00 00	.4..p....p......
00000180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00000190	00 80 02 00 AC 21 00 00 80 4A 00 00 54 00 00 00	.....!...J..T...
000001A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
000001B0	D4 13 00 00 18 00 00 00 30 13 00 00 A4 00 00 00	........0.......
000001C0	00 00 00 00 00 00 00 00 00 30 02 00 B4 04 00 00	.........0......
000001D0	84 07 02 00 E0 00 00 00 00 00 00 00 00 00 00 00	....
000001E0	00 00 00 00 00 00 00 00	....

IMAGE_NT_HEADERS 구조체의 크기는 0xF8(248)이다.

(1) IMAGE_FILE_HEADER

```
typedef struct _IMAGE_FILE_HEADER {
  WORD  Machine;
  WORD  NumberOfSections;
  DWORD TimeDateStamp;
  DWORD PointerToSymbolTable;
  DWORD NumberOfSymbols;
  WORD  SizeOfOptionalHeader;
  WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

- Machine : CPU별로 고유한 값이며 32bit Intel x86 호환 칩은 14C의 값을 가진다.

값	의미
IMAGE_FILE_MACHINE_I386 0x014c	x86
IMAGE_FILE_MACHINE_IA64 0x0200	인텔 아이테니엄

IMAGE_FILE_MACHINE_AMD64 0x8664	x64

- NumberOfSections : 섹션의 개수 (반드시 0보다 크다.)
- SizeOfOptionalHeader : IMAGE_OPTAIONAL_HEADER32 구조체의 크기
  - IMAGE_OPTIONAL_HEADER32는 C언어 구조체이기 때문에 이미 그 크기가 결정되어 있다.
    PE32+ 형태의 파일인 경우 IMAGE_OPTAIONAL_HEADER64 구조체를 사용함
    - 해당 변수의 값을 보고 IMAGE_OPTAIONAL_HEADER구조체의 크기를 인식한다.
- Characteristics : 파일의 속성을 나타내는 값
  - 실행 가능한 형태(Executable or not) or DLL 파일 여부 등의 정보들이 bit OR 형식으로 조합

값	값	의미
IMAGE_FILE_RELOCS_STRIPPED	0x0001	재배치 정보가 파일에서 제거되었습니다. 파일은 기본 주소로로드해야합니다. 기본 주소를 사용할 수 없으면 로더가 오류를보고합니다.
IMAGE_FILE_EXECUTABLE_IMAGE	0x0002	파일이 실행 가능합니다 (해결되지 않은 외부 참조가 없음).
IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004	COFF 줄 번호가 파일에서 제거되었습니다.
IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008	COFF 기호 테이블 항목이 파일에서 제거되었습니다.
IMAGE_FILE_AGGRESIVE_WS_TRIM	0x0010	작업 세트를 적극적으로 다듬습니다. 이 값은 더 이상 사용되지 않습니다.
IMAGE_FILE_LARGE_ADDRESS_AWARE	0x0020	응용 프로그램은 2GB보다 큰 주소를 처리 할

		수 있습니다.
IMAGE_FILE_BYTES_REVERSED_LO	0x0080	워드의 바이트가 반전됩니다. 이 플래그는 더 이상 사용되지 않습니다.
IMAGE_FILE_32BIT_MACHINE	0x0100	컴퓨터는 32 비트 단어를 지원합니다.
IMAGE_FILE_DEBUG_STRIPPED	0x0200	디버깅 정보가 제거되어 다른 파일에 별도로 저장되었습니다.
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x0400	이미지가 이동식 미디어에있는 경우 이미지를 복사하여 스왑 파일에서 실행하십시오.
IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	이미지가 네트워크에있는 경우 이미지를 복사하여 스왑 파일에서 실행하십시오.
IMAGE_FILE_SYSTEM	0x1000	이미지는 시스템 파일입니다.
IMAGE_FILE_DLL	0x2000	이미지는 DLL 파일입니다. 실행 파일이지만 직접 실행할 수 없습니다.
IMAGE_FILE_UP_SYSTEM_ONLY	0x4000	파일은 단일 프로세서 컴퓨터에서만 실행해야 합니다.
IMAGE_FILE_BYTES_REVERSED_HI	0x8000	워드의 바이트가 반전됩니다. 이 플래그는 더 이상 사용되지 않습니다.

(2) IMAGE_OPTIONAL_HEADER32

```
typedef struct _IMAGE_DATA_DIRECTORY {
  DWORD VirtualAddress; // 테이블의 상대 가상 주소입니다.
  DWORD Size; // 바이트 단위의 테이블 크기입니다.
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

```
typedef struct _IMAGE_OPTIONAL_HEADER {
  WORD                 Magic;
  BYTE                 MajorLinkerVersion;
  BYTE                 MinorLinkerVersion;
  DWORD                SizeOfCode;
  DWORD                SizeOfInitializedData;
  DWORD                SizeOfUninitializedData;
  DWORD                AddressOfEntryPoint;
  DWORD                BaseOfCode;
  DWORD                BaseOfData;
  DWORD                ImageBase;
  DWORD                SectionAlignment;
  DWORD                FileAlignment;
  WORD                 MajorOperatingSystemVersion;
  WORD                 MinorOperatingSystemVersion;
  WORD                 MajorImageVersion;
  WORD                 MinorImageVersion;
  WORD                 MajorSubsystemVersion;
  WORD                 MinorSubsystemVersion;
  DWORD                Win32VersionValue;
  DWORD                SizeOfImage;
  DWORD                SizeOfHeaders;
  DWORD                CheckSum;
  WORD                 Subsystem;
  WORD                 DllCharacteristics;
```

```
  DWORD                 SizeOfStackReserve;
  DWORD                 SizeOfStackCommit;
  DWORD                 SizeOfHeapReserve;
  DWORD                 SizeOfHeapCommit;
  DWORD                 LoaderFlags;
  DWORD                 NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_
ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

```
typedef struct _IMAGE_OPTIONAL_HEADER64 {
  WORD                  Magic;
  BYTE                  MajorLinkerVersion;
  BYTE                  MinorLinkerVersion;
  DWORD                 SizeOfCode;
  DWORD                 SizeOfInitializedData;
  DWORD                 SizeOfUninitializedData;
  DWORD                 AddressOfEntryPoint;
  DWORD                 BaseOfCode;
  ULONGLONG             ImageBase;
  DWORD                 SectionAlignment;
  DWORD                 FileAlignment;
  WORD                  MajorOperatingSystemVersion;
  WORD                  MinorOperatingSystemVersion;
  WORD                  MajorImageVersion;
  WORD                  MinorImageVersion;
  WORD                  MajorSubsystemVersion;
  WORD                  MinorSubsystemVersion;
  DWORD                 Win32VersionValue;
  DWORD                 SizeOfImage;
```

```
  DWORD                 SizeOfHeaders;
  DWORD                 CheckSum;
  WORD                  Subsystem;
  WORD                  DllCharacteristics;
  ULONGLONG             SizeOfStackReserve;
  ULONGLONG             SizeOfStackCommit;
  ULONGLONG             SizeOfHeapReserve;
  ULONGLONG             SizeOfHeapCommit;
  DWORD                 LoaderFlags;
  DWORD                 NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_
ENTRIES];
} IMAGE_OPTIONAL_HEADER64, *PIMAGE_OPTIONAL_HEADER64;
```

- Magic : (x86)10B, (x64)20B

IMAGE_NT_OPTIONAL_HDR_MAGIC	파일은 실행 가능한 이미지입니다. 이 값은 32 비트 응용 프로그램에서 **IMAGE_NT_OPTIONAL_HDR32_MAGIC** 로, 64 비트 응용 프로그램에서 **IMAGE_NT_OPTIONAL_HDR64_MAGIC** 로 정의됩니다 .
IMAGE_NT_OPTIONAL_HDR32_MAGIC 0x10b	파일은 실행 가능한 이미지입니다.
IMAGE_NT_OPTIONAL_HDR64_MAGIC 0x20b	파일은 실행 가능한 이미지입니다.
IMAGE_ROM_OPTIONAL_HDR_MAGIC 0x107	파일은 ROM 이미지입니다.

- AddressOfEntryPoint
  - EP의 RVA(Relative Virtual Address) 값
  - 프로그램에서 최초로 실행되는 코드의 시작 주소 (Ex. Main Func)
- ImageBase
  - 프로세스의 가상 메모리는 x86기준 0~FFFFFFFF이다.
  - EXE, DLL 파일은 user memory 영역인 0~7FFFFFFF범위에 로딩
  - SYS 파일은 kernel memory 영역인 80000000~FFFFFFFF 범위에 로딩

        - 보통 개발 도구들이 만들어내는 EXE 파일의 Image Base 값은 00400000이고,
          DLL 파일은 10000000이다.
    - PE 로더는 PE 파일을 실행시키기 위해 프로세스를 생성하고 파일을 메모리에 로딩한 후 EIP 레지스터 값을 ImageBase + AddressOfEntryPoint 값으로 세팅한다.
- SectionAlignment, FileAlignment

  PE 파일의 Body 부분은 섹션(Section)으로 나뉘어져 있습니다.
    - FileAlignment : 파일에서 섹션의 최소단위를 나타낸다.
    - SectionAlignment : 섹션의 최소 단위를 타나낸다.
        - 파일/메모리의 섹션 크기는 반드시 각각 두 값의 배수가 되어야 한다.
- SizeOfImage
    - PE 파일이 메모리에 로딩되었을 때 가상 메모리에서 PE Image가 차지하는 크기
- SizeOfHeader
    - PE 헤더의 전체 크기
    - FileAlignment의 배수여야 한다.
    - 파일 시작에서 SizeOfHeader Offset 만큼 덜어진 위체에 첫 번째 섹션이 위치
- Subsystem
    - 시스템 드라이버 파일(.sys) or 일반 실행 파일(.exe, .dll) 구분

값		의미
IMAGE_SUBSYSTEM_UNKNOWN	0	알 수없는 서브시스템.
IMAGE_SUBSYSTEM_NATIVE	1	서브 시스템이 필요하지 않습니다 (장치 드라이버 및 기본 시스템 프로세스).
IMAGE_SUBSYSTEM_WINDOWS_GUI	2	Windows 그래픽 사용자 인터페이스 (GUI) 하위 시스템.
IMAGE_SUBSYSTEM_WINDOWS_CUI	3	Windows 문자 모드 사용자 인터페이스 (CUI) 서브 시스템.

IMAGE_SUBSYSTEM_OS2_CUI	5	OS / 2 CUI 서브 시스템.
IMAGE_SUBSYSTEM_POSIX_CUI	7	POSIX CUI 서브 시스템.
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI	9	Windows CE 시스템.
IMAGE_SUBSYSTEM_EFI_APPLICATION	10	EFI (Extensible Firmware Interface) 응용 프로그램
IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER	11	부팅 서비스가 포함 된 EFI 드라이버
IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER	12	런타임 서비스가 포함 된 EFI 드라이버
IMAGE_SUBSYSTEM_EFI_ROM	13	EFI ROM 이미지
IMAGE_SUBSYSTEM_XBOX	14	Xbox 시스템.
IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION	16	부팅 응용 프로그램.

- NumberOfRvaAndSizes
  - IMAGE_OPTIONAL_HEADER32 구조체 DataDirectory 배열의 개수
- DataDirectory
  - IMAGE_DATA_DIRECTORY 구조체의 배열

---

4) Section Header

- 각 섹션의 속성(Property)을 정의한 것
  - file/memory에서의 시작 위치, 크기, 엑세스 권한 등

code	실행, 읽기 권한
data	비실행, 읽기, 쓰기 권한
resource	비실행, 읽기 권한

(1) IMAGE_SECTION_HEADER

```
#define IMAGE_SIEOF_SHORT_NAME 0
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

VirtualSize	메모리에서 섹션이 차지하는 크기
VirtualAddress	메모리에서 섹션의 시작 주소(RVA)
SizeOfRawData	파일에서 섹션이 차지하는 크기
PointerToRawData	파일에서 섹션의 시작 위치
Characteristic	섹션의 속성(bit OR)
Name	.text, .data 등의 이름이 지정되는 멤버 ※ PE 스펙에서는 섹션 Name에 대한 어떠한 명시적인 규칙이 없기 때문에 어떠한 값을 넣어도 되고 심지어 NULL로 채워도 상관이 없다.

- Virtual Address와 PointerToRawData는 아무 값이나 가질수 없고, 각각 SectionAlignment와 FileAlignment에 맞게 결정된다.
- VirtualSize와 SizeOfRawData는 일반적으로 서로 다른 값을 가진다.
  - 파일에서의 섹션크기와 메모리에 로딩된 섹션의 크기는 다르다.

IMAGE_SCN_CNT_CODE 0x00000020	이 섹션에는 실행 코드가 포함되어 있습니다.
IMAGE_SCN_CNT_INITIALIZED_DATA 0x00000040	이 섹션에는 초기화 된 데이터가 포함되어 있습니다.
IMAGE_SCN_CNT_UNINITIALIZED_DATA 0x00000080	이 섹션에는 초기화되지 않은 데이터가 포함되어 있습니다.
IMAGE_SCN_MEM_EXECUTE 0x20000000	이 섹션은 코드로 실행될 수 있습니다.
IMAGE_SCN_MEM_READ 0x40000000	섹션을 읽을 수 있습니다.
IMAGE_SCN_MEM_WRITE 0x80000000	섹션을 쓸 수 있습니다.

> *Image : 메모리에 로딩된 상태*

---

5. RVA to RAW

> *1. RVA가 속해 있는 섹션을 찾는다.*
> *간단한 비례식을 사용해서 파일 Offset을 계산한다.*

- IMAGE_SECTION_HEADER 구조체에 의한 비례식

```
RAW - PointerToRawData = RVA - VirtualAddress
RAW                    = RVA - VirtualAddress + PointerToRawData
```