Rootkit Arsenal

Day 2. Into the Catacombs: IA-32 Protected Mode

n0fate@live.com

Contents

- Protected Mode
- Segmentation & Paging
- Implementing Memory Protection
- Example

n0fate@live.com

Protected Mode

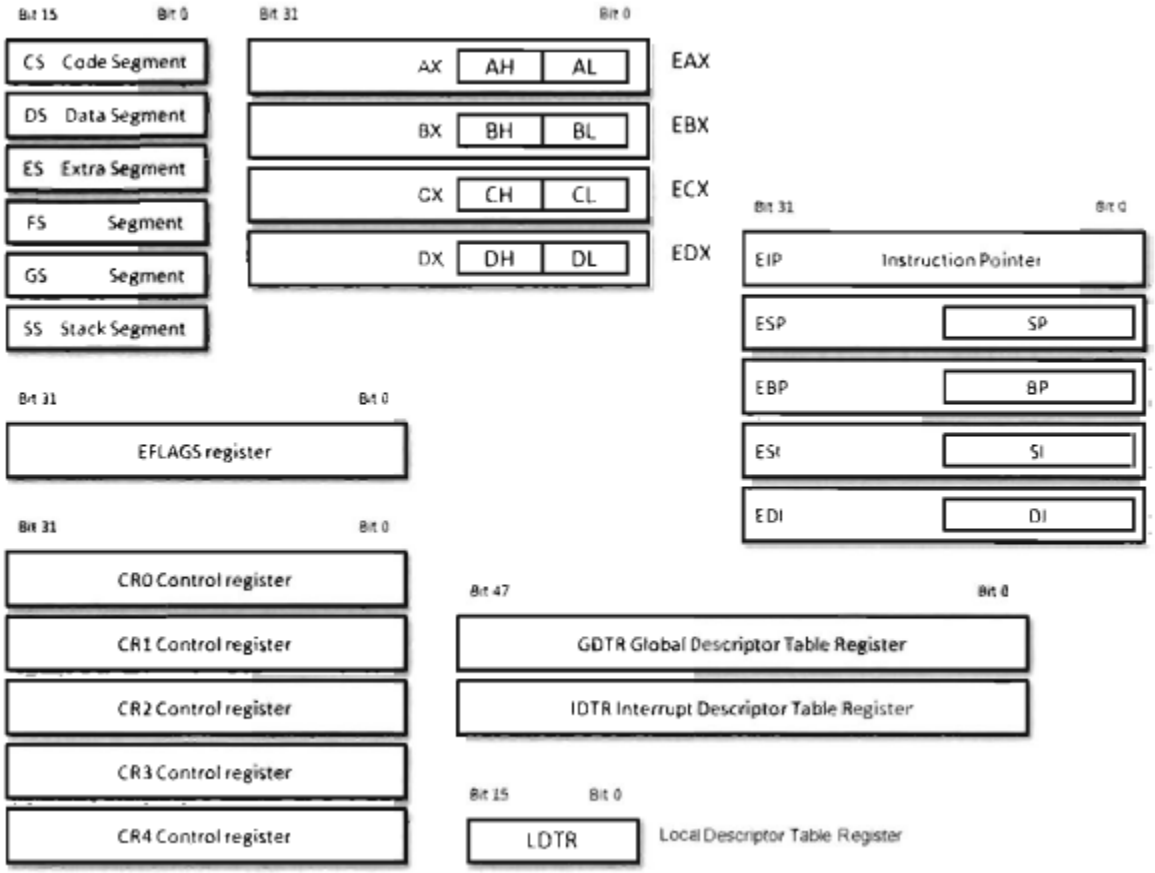
- Protected mode is an instance of the segmented memory model

- Kernel Mode와 User Mode를 분리하여 보안성 향상
  - 커널 메모리 침범 방지
  - Write Protection을 통해 프로그램의 위변조 방지

Protected-Mode Execution Env

- The protected-mode execution environment can be seen as an extension of the real-mode execution env
- real mode
  - six segment registers, four general registers, three pointer registers, two indexing registers, flags registers
- protected mode
  - real mode registers, five control registers(CRx), gdtr, ldtr, idtr

Protected-Mode Execution Env



n0fate@live.com

SEGMENTATION & PAGING

n0fate@live.com

Segmentation & Paging

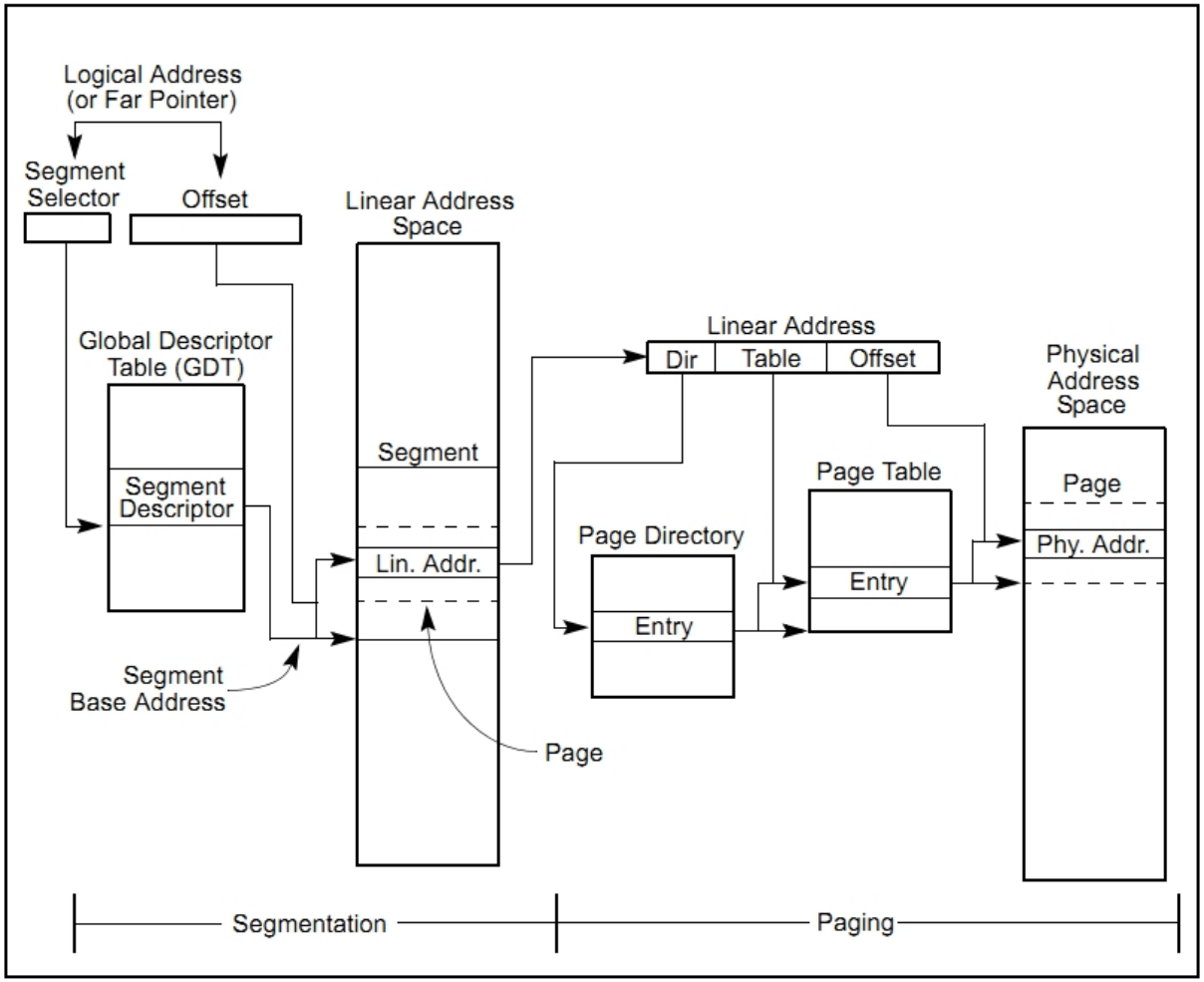


Figure 3-1. Segmentation and Paging

Segmentation

- Segmentation
  - 사용자가 지정한 메모리 주소로부터 선형 메모리까지 얻어지는 과정
  - 세그먼트 레지스터, 셀렉터, 디스크립터에 의해 동작

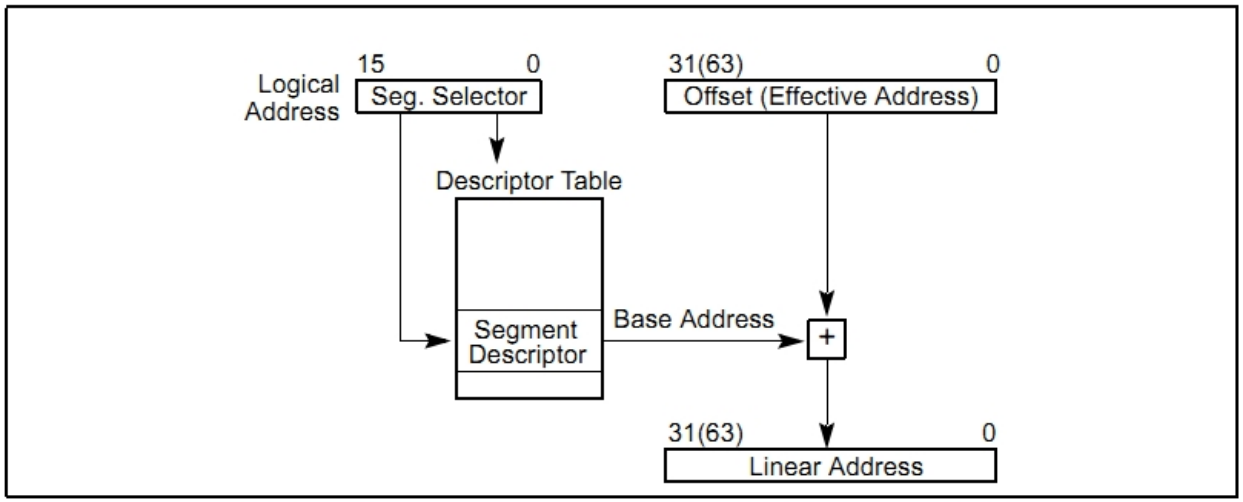


Figure 3-5. Logical Address to Linear Address Translation

Segmentation

- Segment Selector
  - 세그먼트 레지스터를 이용해 디스크립터 테이블에서 세그먼트 디스크립터를 선택하는 것
  - 2^13 → 8192개의 Segment Descriptor를 가리킬 수 있음

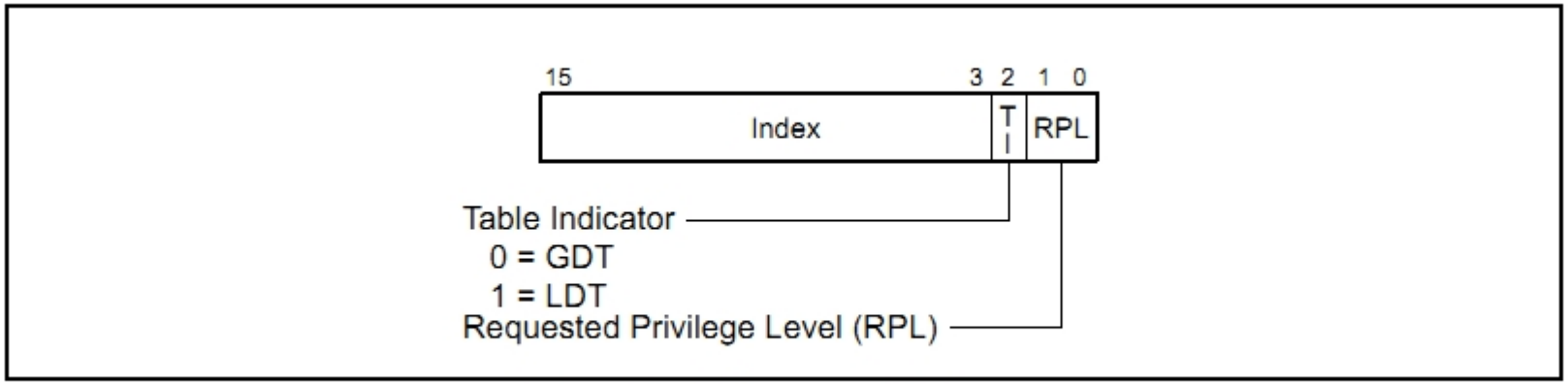


Figure 3-6. Segment Selector

Segmentation

- Segment Register
  - Visible Part에서 가리킨 Segment Descriptor를 Hidden Part에 로드함.
  - DS, ES, FS, GS는 데이터 세그먼트를 가리키는 용도

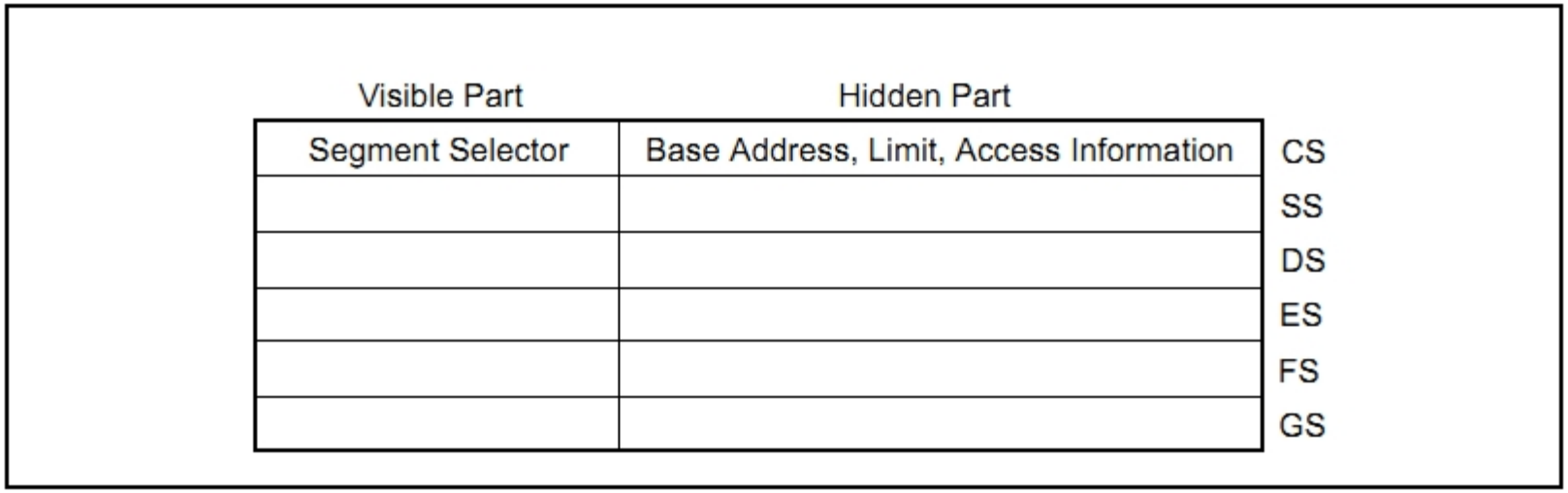


Figure 3-7. Segment Registers

n0fate@live.com

Segmentation

- GDT
  - Kernel Mode 용 테이블
  - 커널용으로 하나 생성
  - GDTR이 가리키고 있음

- LDT
  - User Mode용 테이블
  - 각 프로세스 별로 만들어짐
  - LDTR이 가리키고 있음

Segmentation

- Segment Descriptors
  - Data structure in a GDT, LDT
  - Structure
    - size of a segment
    - location of a segment
    - access control
    - status information

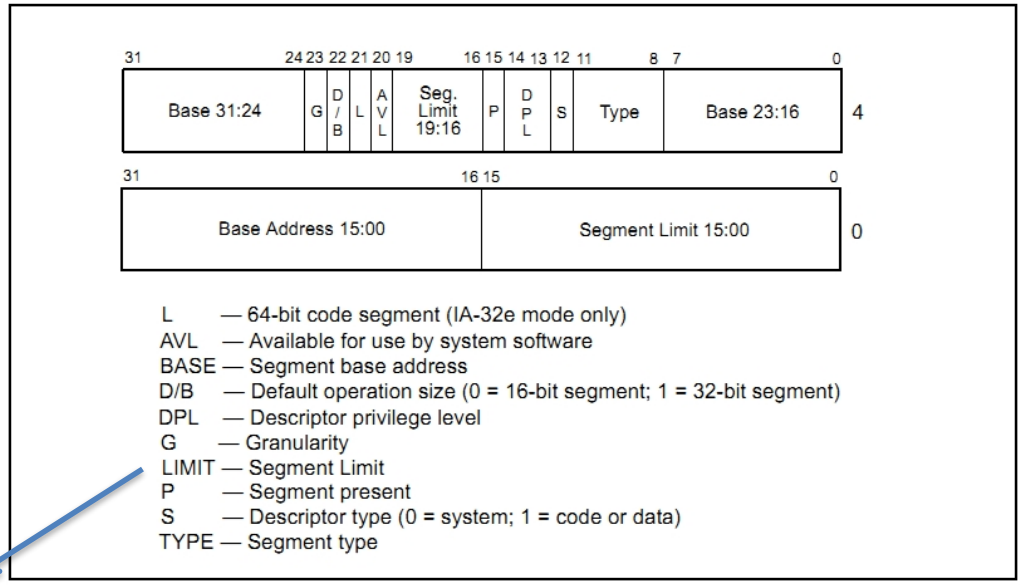


Figure 3-8. Segment Descriptor

If G bit == 1 then limit size = 4kb
G bit == 0 then limit size = 1Bit

n0fate@live.com

Segmentation

- Segment Descriptors
  - Base Address Field → 세그먼트 시작 주소
  - Segment Limit Field → 세그먼트의 한계 주소
  - Type Field → Segment Type(CS(1), DS(0))
  - System Flag → System(0) / CS or DS(1) Segment
    - System Segment descriptor는 더 높은 권한이 필요할 때 사용
      - EX. System Call을 수행할 경우
  - DPL → Descriptor Privilege Level
  - P Flag → 세그먼트의 물리 메모리 존재 여부

Segmentation

Bit 11	Bit 10	Bit 09	Bit 08	Type	Description
0	0	0	0	Data	Read Only
0	0	0	1	Data	Read Only, Recently Accessed
0	0	1	0	Data	Read/Write
0	0	1	1	Data	Read/Write, Recently Accessed
0	1	0	0	Data	Read Only, Expand Down
0	1	0	1	Data	Read Only, Recently Accessed, Expand Down
0	1	1	0	Data	Read/Write, Expand Down
0	1	1	1	Data	Read/Write, Recently Accessed, Expand Down

Data Segment

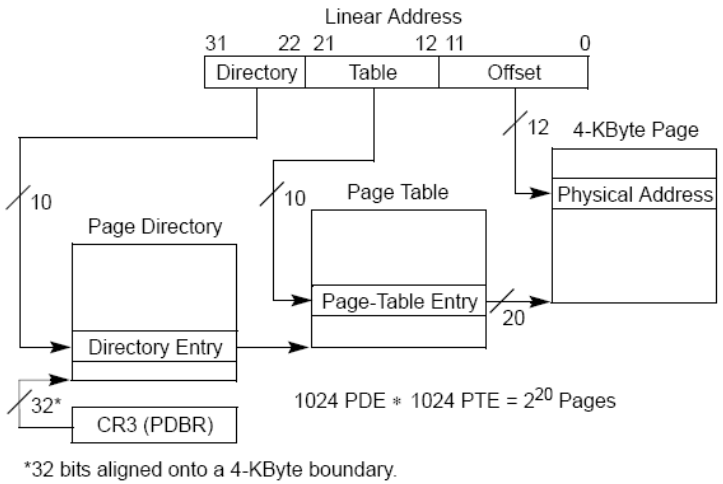
Bit 11	Bit 10	Bit 09	Bit 08	Type	Description
1	0	0	0	Code	Execute-Only
1	0	0	1	Code	Execute-Only, Recently Accessed
1	0	1	0	Code	Execute-Read
1	0	1	1	Code	Execute-Read, Recently Accessed
1	1	0	0	Code	Execute-Only, Conforming
1	1	0	1	Code	Execute-Only, Recently Accessed, Conforming
1	1	1	0	Code	Execute-Read, Conforming
1	1	1	1	Code	Execute-Read, Recently Accessed, Conforming

Code Segment

n0fate@live.com

Paging

- Paging은 optional
  - 초기엔 linear address space가 physical memory와 일치했음.
  - 32비트 선형 주소 체계에 해당하는 4GByte 메모리를 작은 용량의 실제 메모리 주소와 매핑시켜 줌
  - 메모리가 부족할 경우 디스크의 Swap 영역을 지원



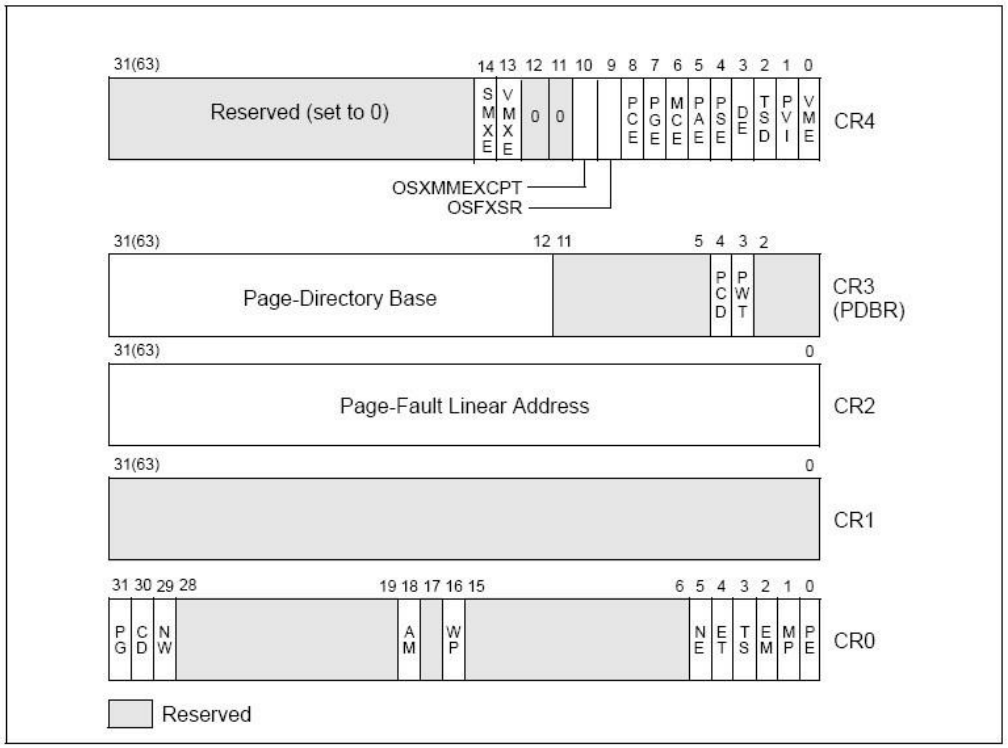
Paging

- CR3 Register
  - 페이지 디렉터리의 위치 정보를 가짐
  - 4KB 단위
  - CR3를 이용하여 각 프로세스가 독립적인 메모리 주소 공간을 가질 수 있음.
  - CR0 레지스터의 WP flag가 Set(1) 이면, supervisor-level code를 read-only user-level memory page에 쓸 수 없음

```
060
061 /** Structure of the x86 CR3 register: the Page Directory Base
062     Register. See Intel x86 doc Vol 3 section 2.5 */
063 struct x86_pdbr
064 {
065   sos_ui32_t zero1          :3; /* Intel reserved */
066   sos_ui32_t write_through  :1; /* 0=write-back, 1=write-through */
067   sos_ui32_t cache_disabled :1; /* 1=cache disabled */
068   sos_ui32_t zero2          :7; /* Intel reserved */
069   sos_ui32_t pd_paddr       :20;
070 } __attribute__ ((packed));
071
```

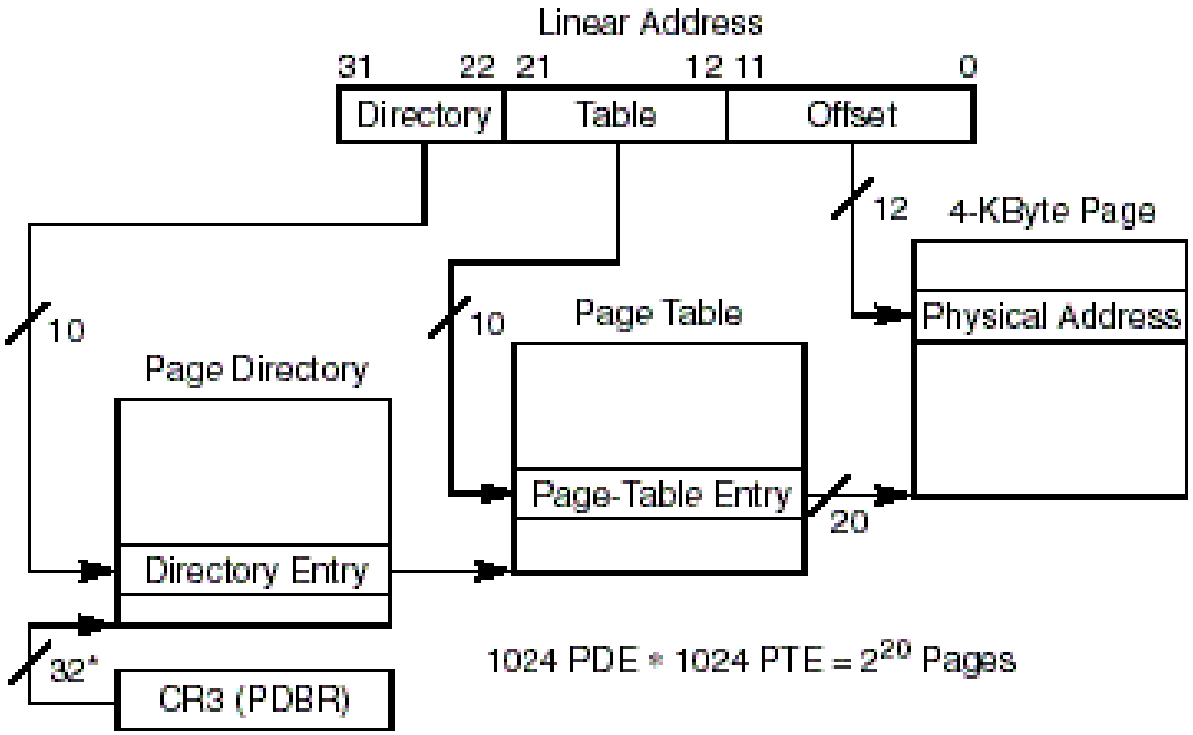
Paging

- Control Register



Paging

- Page Table



n0fate@live.com

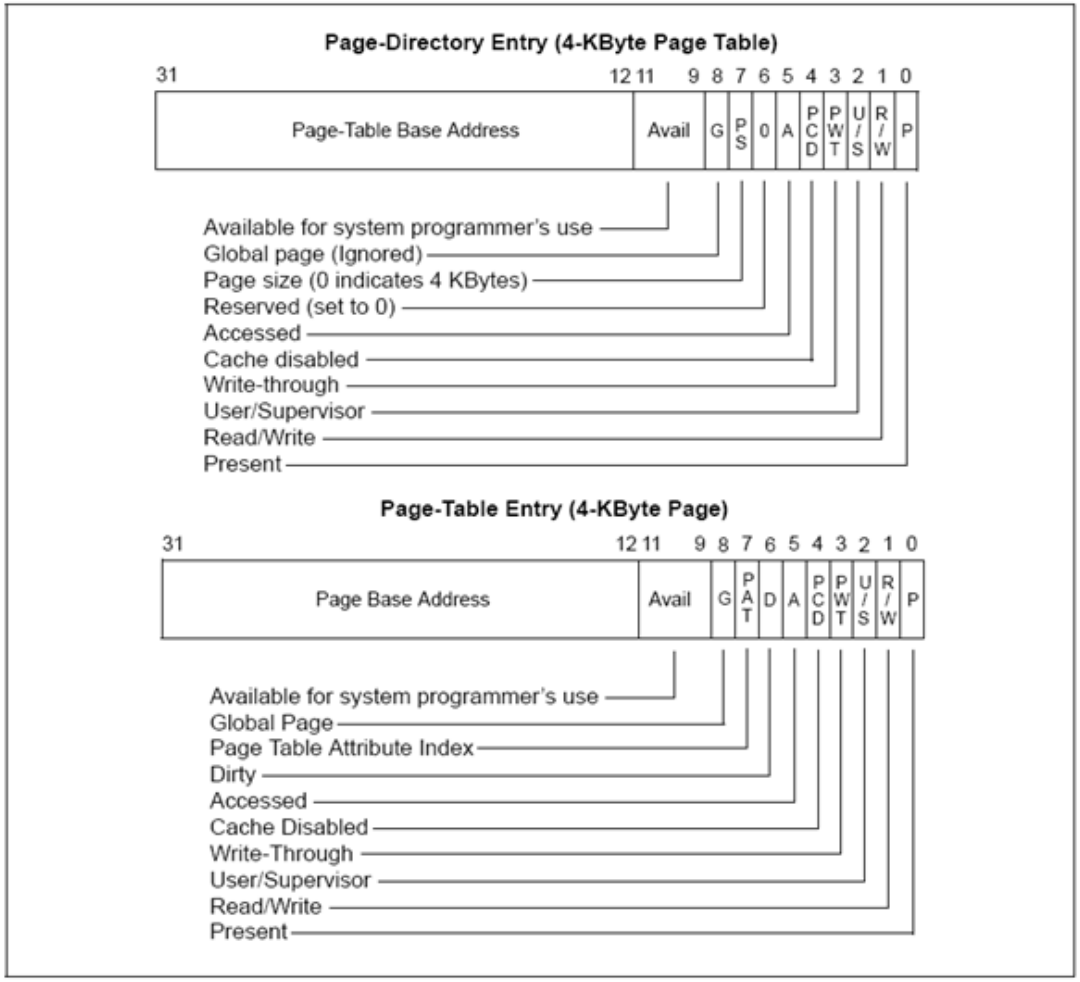
Paging

- Paging Structure

	[63..36]	[35..32]	[31..21]	[20..12]	[11..09]	08	07	06	05	04	03	02	01	00
CR3	Not Applicable		Page Directory Base Pointer [31..05]							PCD	PWT	0		
Page Directory Pointer	RSV	Page Directory Address [35..12]			AVL	RSV	RSV	RSV	RSV	PCD	PWT	RSV	RSV	P
Page Directory Entry *2M*	RSV	Page Frame Address [35..21]		RSV	AVL	G	PS*	D	A	PCD	PWT	U	W	P
Page Directory Entry *4K*	RSV	Page Table Address [35..12]			AVL	0	PS†	0	A	PCD	PWT	U	W	P
Page Table Entry *4K*	RSV	Page Frame Address [35..12]			AVL	G	RSV	D	A	PCD	PWT	U	W	P

n0fate@live.com

Paging



n0fate@live.com

We discuss how they’re used to offer memory protection

IMPLEMENTING MEMORY PROTECTION

n0fate@live.com

Protection through Segmentation

- segment-based protection
  - Limit checks
  - Segment type checks
  - Privilege-level checks
  - Restricted-instruction checks
- All of these checks will occur before the memory access cycle begins
  - If a violation occurs, a general-protection exception will be generated by the processor
  - there is no performance penalty associated with these checks

n0fate@live.com

Protection through Segmentation

- Limit Checks
  - GDT의 크기를 넘어가지 않도록 limit field로 제한
- Type Checks
  - S Flag와 Type Field를 이용하여 부적절한 메모리 세그먼트로의 접근을 차단
  - 예
    - CS Register 값은 코드 세그먼트를 통해 불러올 수 있음
    - far call 이나 far jump만이 다른 코드 세그먼트 디스크립터나 콜게이트로 접근 가능

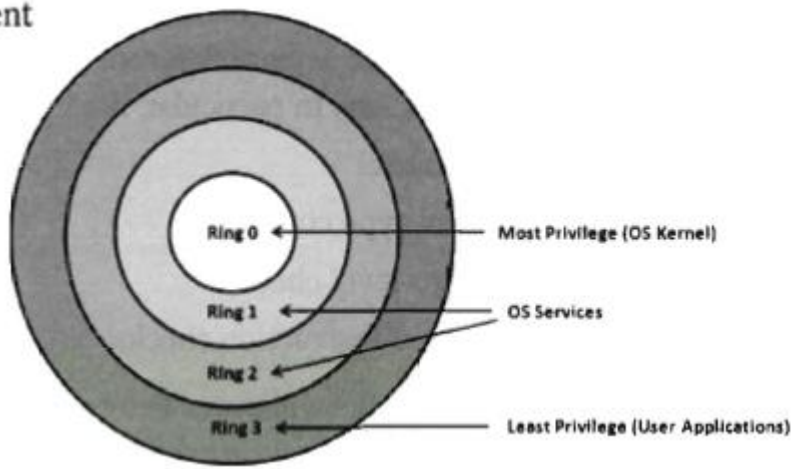
Protection through Segmentation

- Privilege Checks
  - CPL, RPL, DPL을 이용하여 체크
  - CPL
    - 실행 중인 프로세스의 CS, SS 레지스터 내의 셀렉터에 RPL값
    - 프로그램의 CPL 값은 현재 코드세그먼트의 권한을 의미
    - far jump or far call 시 변경될 수 있음
  - 이 체크는 세그먼트 셀렉터가 로드될 때 수행
  - 프로세스 권한 위배 발생 시
    - General-Protection exception(GP#) 발생

Protection through Segmentation

- Privilege Checks
  - 다른 세그먼트의 데이터에 접근 시
    - 프로세서는 DPL이 RPL과 CPL 값과 같거나 더 높은지 확인함.
    - 같거나 더 높으면 데이터 세그먼트 셀렉터와 레지스터 로드
  - 스택 세그먼트 접근 시
    - 스택세그먼트의 DPL과 세그먼트 셀렉터의 RPL이 CPL과 일치해야함.

ent



Protection through Segmentation

- Privilege Checks
  - nonconforming code segment로 제어를 넘길 경우
    - 호출 루틴의 CPL이 대상 세그먼트의 DPL과 일치해야 함
    - 대상 코드 세그먼트를 가리키는 세그먼트 셀렉터의 RPL은 CPL 과 같거나 낮아야 함.
  - conforming code segment로 제어를 넘길 경우
    - EX) General Exception Handler
    - 호출 루틴의 CPL이 대상 세그먼트의 DPL과 일치해야 함.
    - 세그먼트 셀렉터의 RPL은 확인하지 않음.

Protection through Segmentation

- Restricted-Instruction Checks
  - CPL을 확인해서 명령어의 사용 가능 여부를 판단함.
  - 표의 명령은 CPL 0으로 제한하였음
  - 이는 명령어가 시스템이 관리하는 구조체 정보를 가져올 수 있기 때문임.

Instruction	Description
LGDT	Load value into GDTR register
LIDT	Load value into LDTR register
MOV	Move a value into a control register
HLT	Halt the processor
WRMSR	Write to a model-specific register

n0fate@live.com

Gate Descriptors

- Gate Descriptor는 다른 권한의 코드 세그먼트 접근 시 사용함.
- Gate Descriptor는 System Descriptor임
  - 즉, 모든 세그먼트 디스크립터가 0

- Gate Descriptor의 종류
  - Call-Gate descriptor
  - Interrupt-gate descriptor
  - Trap-gate descriptor

Gate Descriptors

- Gate Descriptor는 Type필드로 구분

Bit 11	Bit 10	Bit 9	Bit 8	Gate Type
0	1	0	0	16-bit call-gate descriptor
0	1	1	0	16-bit interrupt-gate descriptor
0	1	1	1	16-bit trap-gate descriptor
1	1	0	0	32-bit call-gate descriptor
1	1	1	0	32-bit interrupt-gate descriptor
1	1	1	1	32-bit trap-gate descriptor

- 게이트는 16/32비트 둘 다 가능
  - 코드 세그먼트 점프에 의해 스택 전환 발생 시 이전 스택 정보는 16비트 32비트 둘 다 존재할 수 있음.

n0fate@live.com

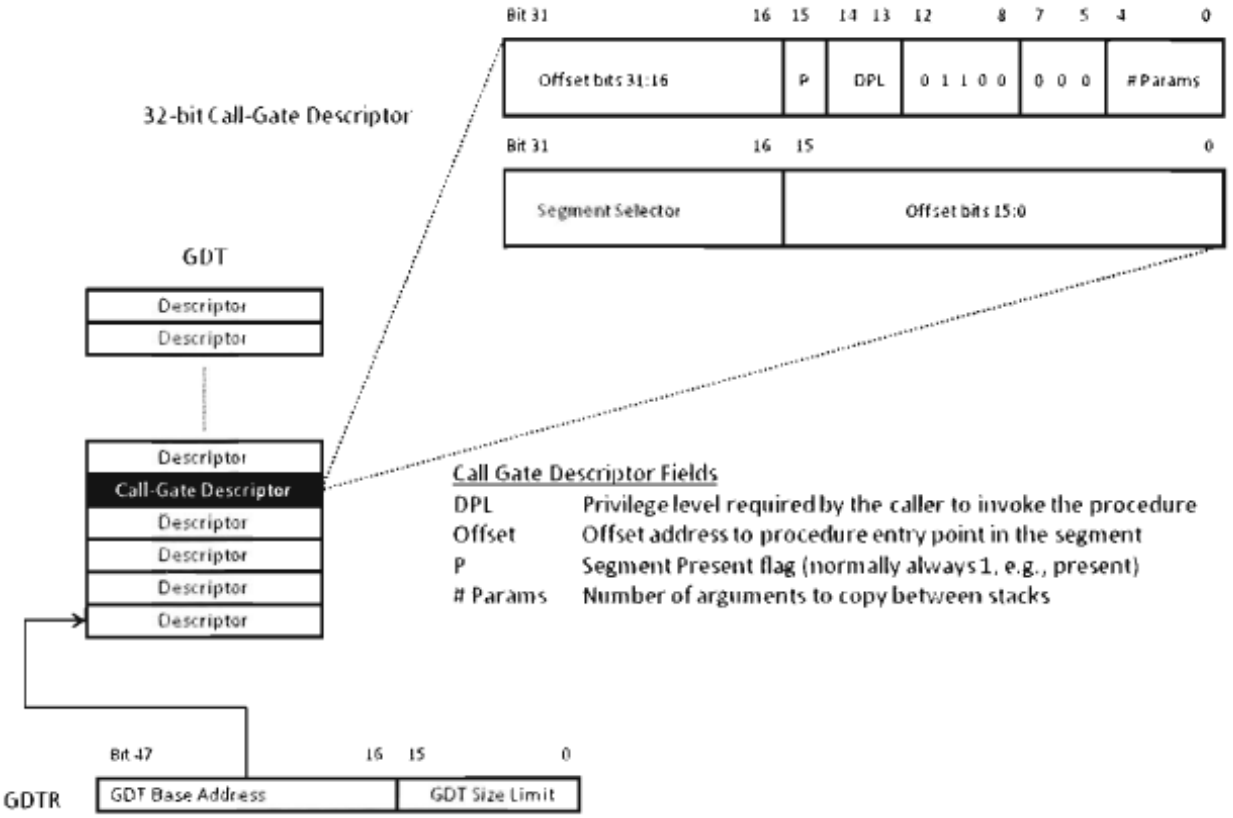
Gate Descriptors

- Call-gate descriptor
  - GDT에 존재함.
  - 다른 Privilege level로 제어권을 넘기기 위함
  - 세그먼트 디스크립터와 유사함.
    - 16비트 세그먼트 셀렉터와 32비트 오프셋 주소를 가짐
  - 프로그램이 콜게이트를 호출 시 권한 검사
    - 프로그램의 CPL과 콜게이트를 가리키는 세그먼트 셀렉터의 RPL이 콜게이트 디스크립터의 DPL과 같거나 높은 권한이여야 함.
    - 프로그램의 CPL은 대상 코드 세그먼트의 DPL과 동일하거나 더 높은 권한이여야 함.

n0fate@live.com

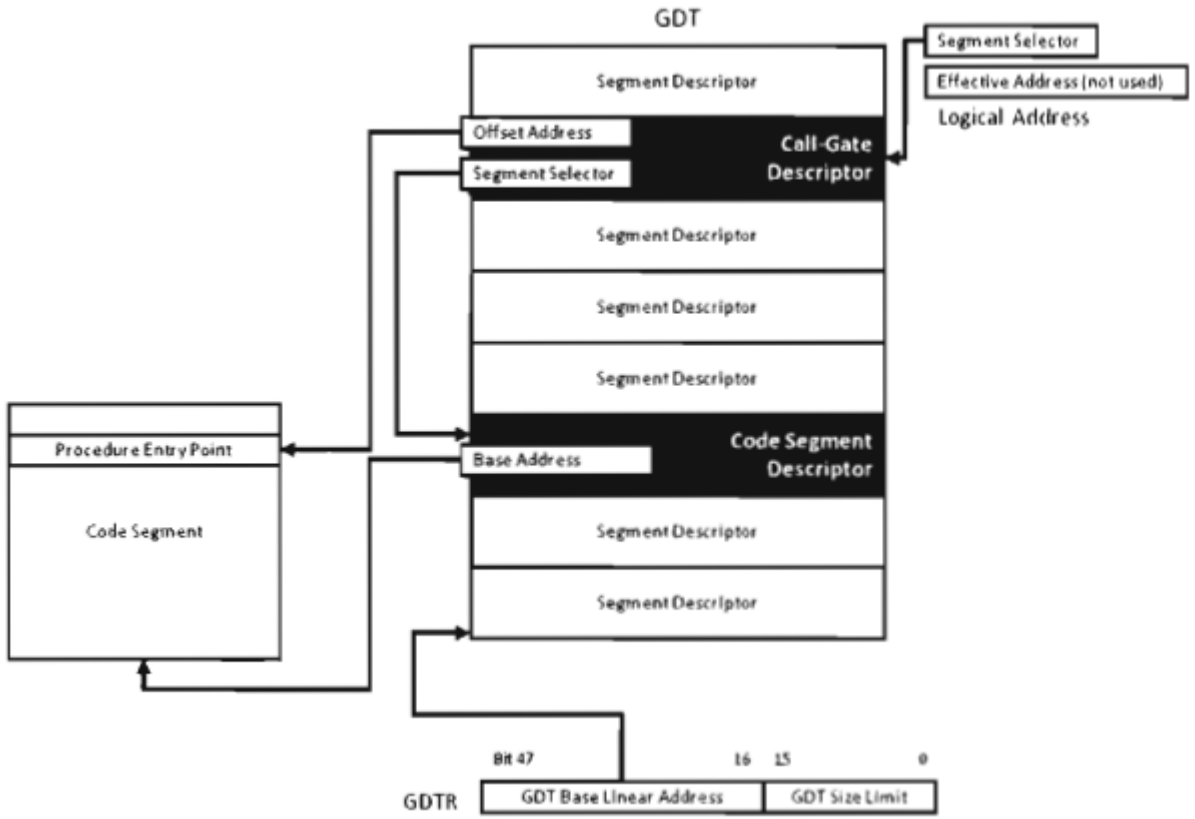
Gate Descriptors

- Call-gate descriptor



Gate Descriptors

- Call-gate descriptor



Gate Descriptors

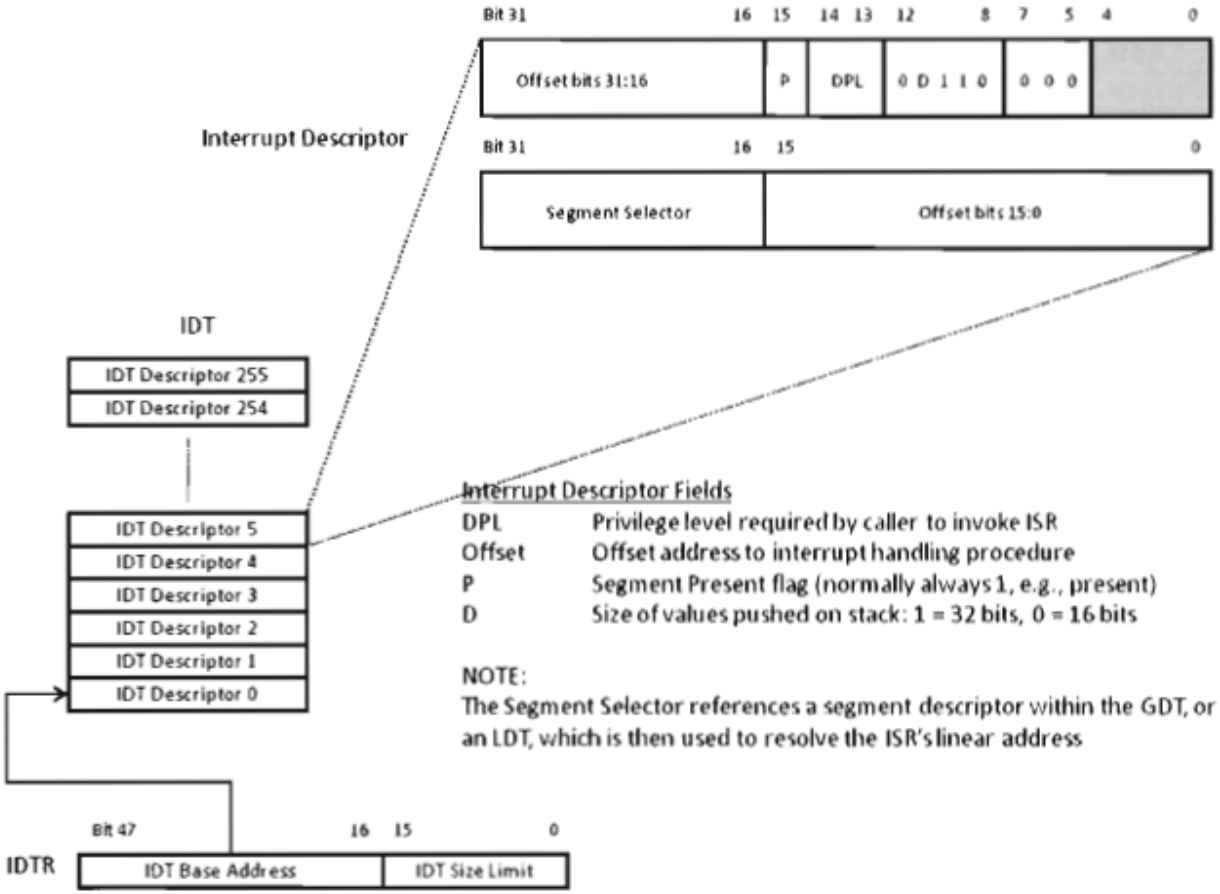
- Interrupt/Trap gate descriptor
  - IDT에 존재
  - segment selector와 effective address를 저장
  - 두 descriptor는 EFLAGS 레지스터의 IF Flag 수정에 차이점을 가짐
    - 인터럽트 핸들링 루틴이 Interrupt gate descriptor를 이용하여 접근 시, 프로세서는 IF 플래그를 0으로 설정
    - Trap gate descriptor는 IF 플래그에 관여하지 않음

Gate Descriptors

- Interrupt/Trap gate descriptor
  - 권한 체크
    - 핸들링 루틴을 실행한 프로그램의 CPL이 Interrupt/Trap gate의 DPL과 일치하거나 그 권한이 높을 경우 가능 함.
    - 핸들링 루틴은 소프트웨어에 의해 실행될 때만 수행 함.
    - 최종적으로 코드 세그먼트를 가리키는 세그먼트 디스크립터의 DPL은 CPL 은 권한이 같거나 높아야 함.

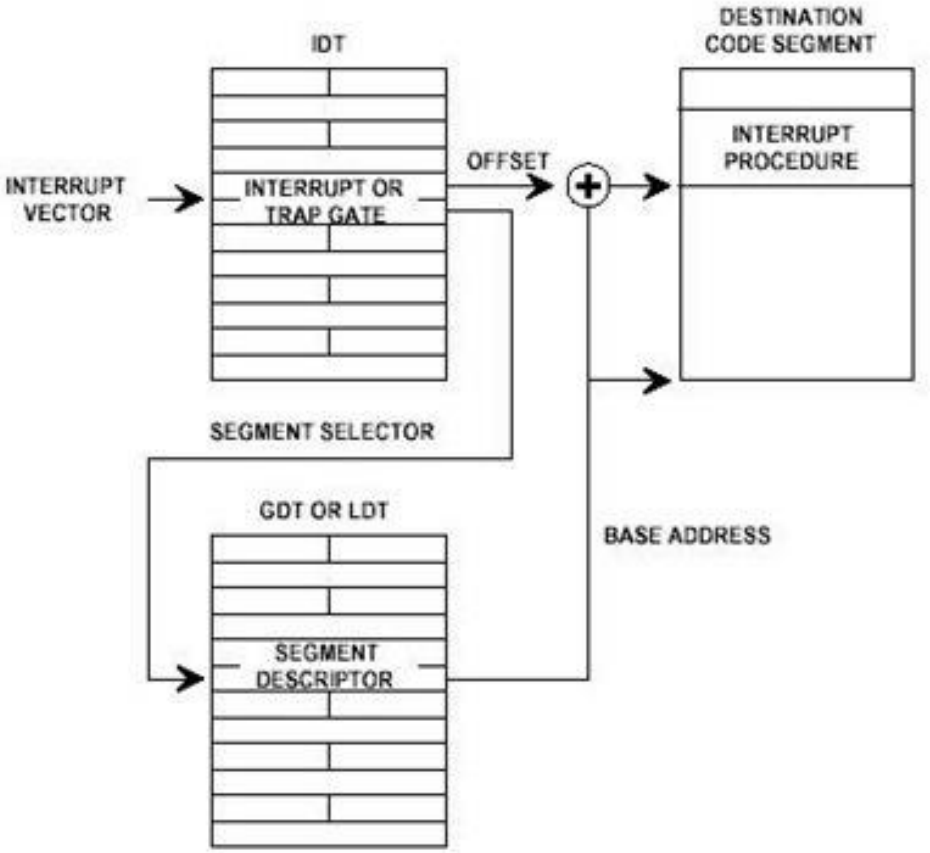
Gate Descriptors

- Interrupt/Trap gate descriptor



Gate Descriptors

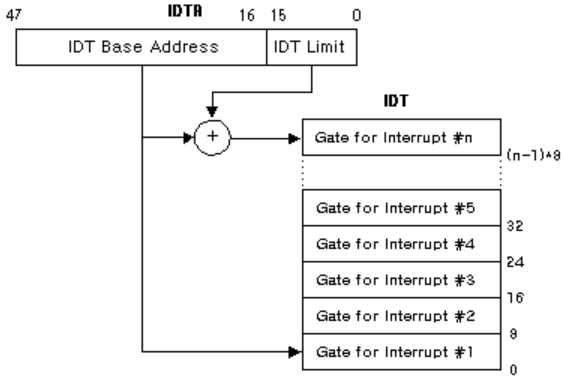
- Interrupt/Trap gate descriptor



n0fate@live.com

Protected-Mode Interrupt Tables

- real mode
  - Interrupt Vector Table에 Interrupt handlers의 주소를 저장
  - 총 256개의 주소를 가질 수 있었음
- Protected Mode
  - Interrupt Descriptor Table로 변경
  - 64비트 gate descriptor 배열을 저장 함.
  - IDT Limit을 초과하여 접근 시
    - general-protection exception 발생



Protected-Mode Interrupt Tables

Vector	Code	Type	Description
00	#DE	Fault	Divide-by-zero error
01	#DB	Trap/Fault	Debug exception (e.g., single-step, task-switch)
02	-	-	NMI interrupt, nonmaskable external interrupt
03	#BP	Trap	Breakpoint
04	#OF	Trap	Overflow (e.g., arithmetic instructions)
05	#BR	Fault	Bound range exceeded (i.e., signed array index is out of bounds)
06	#UD	Fault	Invalid opcode
07	#NM	Fault	No math coprocessor
08	#DF	Abort	Double fault (i.e., CPU detects an exception while handling exception)
09	-	Abort	Coprocessor segment overrun (Intel reserved; do not use)
0A	#TS	Fault	Invalid TSS (e.g., related to task switching)
0B	#NP	Fault	Segment not present (P flag in a descriptor is clear)
0C	#SS	Fault	Stack fault exception
0D	#GP	Fault	General protection exception
0E	#PF	Fault	Page fault exception
0F	-	-	Reserved by Intel
10	#MF	Fault	x87 FPU error
11	#AC	Fault	Alignment check (i.e., detected an unaligned memory operand)
12	#MC	Abort	Machine check (i.e., internal machine error, abandon ship!)
13	#XM	Fault	SIMD floating-point exception
14-1F	-	-	Reserved by Intel
20-FF	-	Interrupt	User-defined interrupts

n0fate@live.com

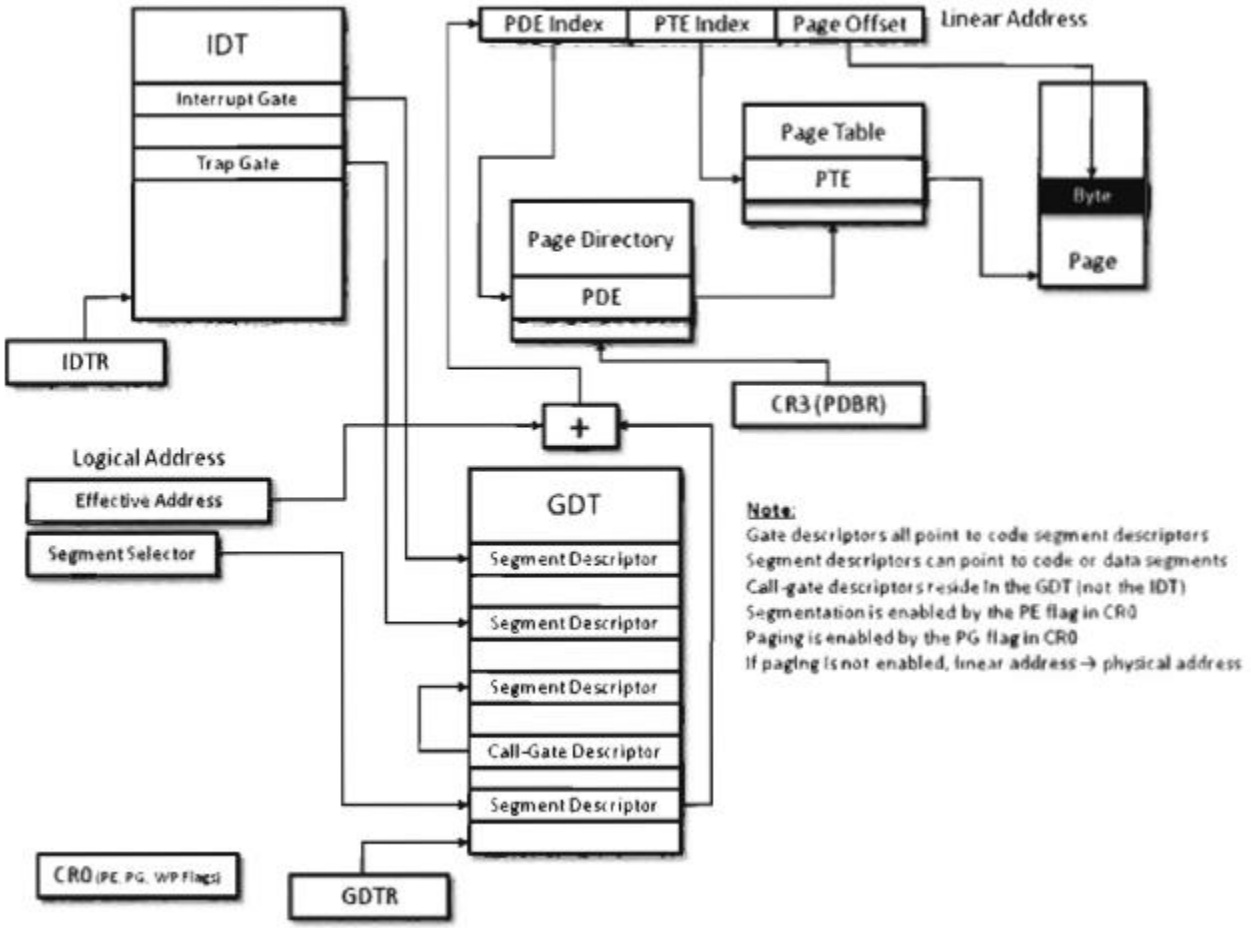
Protection through Paging

- page-level check
  - occur before the memory cycle is initiated
  - no performance overhead is incurred
  - If a violation of page-level check occurs, a page-fault exception(#PF) is emitted by the processor
- 크게 3가지 요소로 보호
  - CR0의 WP Flag로 Page Table 보호
  - User/Supervisor mode checks
  - Page type checks(R/W flag)

Flag	Set (1)	Clear(0)
U/S	User mode	Supervisor mode
R/W	Read and Write	Read-only

n0fate@live.com

Summary



n0fate@live.com

EXAMPLE

n0fate@live.com

Logical to Physical Address

- PsGetCurrentThread 주소 변경하기

```
kd> x nt!PsGetCurrentThread
8086c4fa nt!PsGetCurrentThread (<no parameter info>)
kd> u 8086c4fa
nt!PsGetCurrentThread:
8086c4fa 64a124010000    mov     eax,dword ptr fs:[00000124h]
8086c500 c3              ret
8086c501 cc              int     3
8086c502 cc              int     3
8086c503 cc              int     3
8086c504 cc              int     3
8086c505 cc              int     3
nt!PsGetCurrentThreadStackBase:
8086c506 64a124010000    mov     eax,dword ptr fs:[00000124h]
kd> db 8086c4fa
8086c4fa  64 a1 24 01 00 00 c3 cc-cc cc cc cc 64 a1 24 01  d.$.........d.$.
8086c50a  00 00 8b 80 58 01 00 00-c3 cc cc cc cc cc 64 a1  ....X.........d.
8086c51a  24 01 00 00 8b 40 1c c3-cc cc cc cc cc cc 64 a1  $....@........d.
8086c52a  24 01 00 00 8a 80 d7 00-00 00 c3 cc cc cc cc cc  $...............
8086c53a  64 a1 24 01 00 00 8b 40-38 8b 80 1c 01 00 00 c3  d.$....@8.......
8086c54a  cc cc cc cc cc cc 64 a1-24 01 00 00 8b 80 18 02  ......d.$.......
8086c55a  00 00 c3 cc cc cc cc cc-64 a1 24 01 00 00 8b 40  ........d.$....@
8086c56a  74 c3 cc cc cc cc cc cc-64 a1 24 01 00 00 8b 80  t.......d.$.....
```

Logical to Physical Address

3번째 Bit가 0 → GDT
Index Bit가 1 → 1번째 디스크립터

```
kd> .formats 0x08
Evaluate expression:
  Hex:     00000008
  Decimal: 8
  Octal:   00000000010
  Binary:  00000000 00000000 00000000 00001000
  Chars:   ....
  Time:    Thu Jan 01 09:00:08 1970
  Float:   low 1.12104e-044 high 0
  Double:  3.95253e-323
```

```
kd> r gdtr
gdtr=8003f000
kd> dq 8003f000
8003f000  00000000`00000000 00cf9a00`0000ffff
8003f010  00cf9200`0000ffff 00cffa00`0000ffff
8003f020  00cff200`0000ffff 80008b04`200020ab
8003f030  ffc092df`f0000001 0040f300`00000fff
8003f040  0000f200`0400ffff 00000000`00000000
8003f050  80008989`48b00068 80008989`49180068
8003f060  00009202`30c0ffff 0000920b`80003fff
8003f070  ff0092ff`700003ff 80009a40`0000ffff
```

n0fate@live.com

Logical to Physical Address

```
kd> dt _KGDTENTRY -r 8003f008
ACPI!_KGDTENTRY
   +0x000 LimitLow         : 0xffff
   +0x002 BaseLow          : 0
   +0x004 HighWord         : __unnamed
      +0x000 Bytes            : __unnamed
         +0x000 BaseMid          : 0 ''
         +0x001 Flags1           : 0x9a ''
         +0x002 Flags2           : 0xcf ''
         +0x003 BaseHi           : 0 ''
      +0x000 Bits             : __unnamed
         +0x000 BaseMid          : 0y00000000 (0)
         +0x000 Type             : 0y11010 (0x1a)
         +0x000 Dpl              : 0y00
         +0x000 Pres             : 0y1
         +0x000 LimitHi          : 0y1111
         +0x000 Sys              : 0y0
         +0x000 Reserved_0       : 0y0
         +0x000 Default_Big      : 0y1
         +0x000 Granularity      : 0y1
         +0x000 BaseHi           : 0y00000000 (0)
```

Granularity가 1이므로
Limit → 0xffffffff
BaseAddr → 0x00

n0fate@live.com

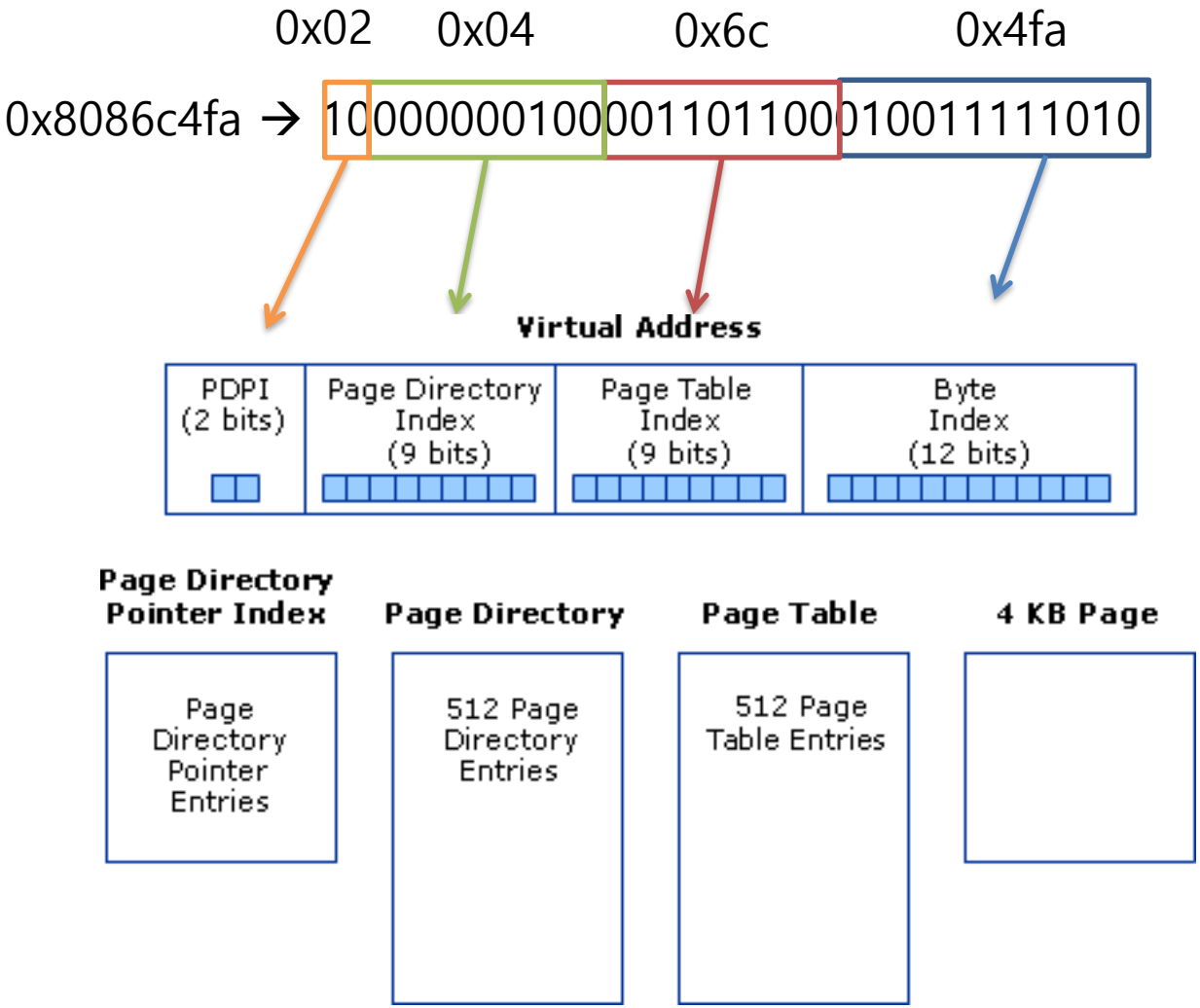
Logical to Physical Address

```
kd> dg 0 255
                                    P Si Gr Pr Lo
Sel    Base     Limit     Type     l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0000 00000000 00000000 <Reserved> 0 Nb By Np Nl 00000000
0008 00000000 ffffffff Code RE    0 Bg Pg P  Nl 00000c9a
0010 00000000 ffffffff Data RW    0 Bg Pg P  Nl 00000c92
0018 00000000 ffffffff Code RE    3 Bg Pg P  Nl 00000cfa
0020 00000000 ffffffff Data RW    3 Bg Pg P  Nl 00000cf2
0028 80042000 000020ab TSS32 Busy 0 Nb By P  Nl 0000008b
0030 ffdff000 00001fff Data RW    0 Bg Pg P  Nl 00000c92
0038 00000000 00000fff Data RW Ac 3 Bg By P  Nl 000004f3
0040 00000400 0000ffff Data RW    3 Nb By P  Nl 000000f2
0048 00000000 00000000 <Reserved> 0 Nb By Np Nl 00000000
0050 808948b0 00000068 TSS32 Avl  0 Nb By P  Nl 00000089
0058 80894918 00000068 TSS32 Avl  0 Nb By P  Nl 00000089
```

즉, 선형 주소는 0x8086c4fa + 0x00000000 → 0x8086c4fa

n0fate@live.com

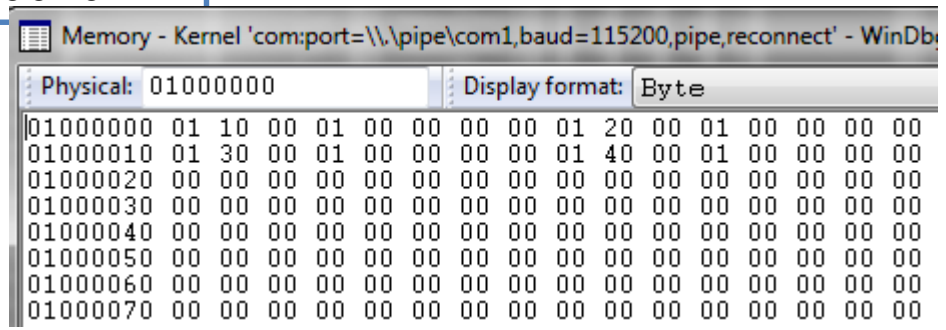
Logical to Physical Address



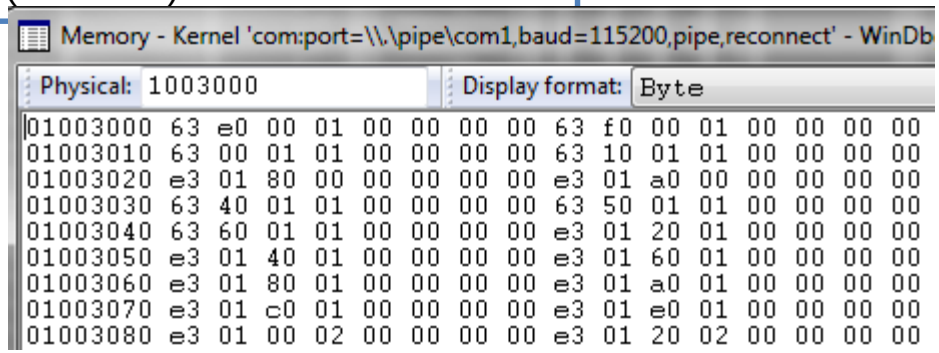
n0fate@live.com

Logical to Physical Address

CR3 + Page Directory Pointer Entry Offset
→ 0x01000000 + (0x02*8) = 01000010



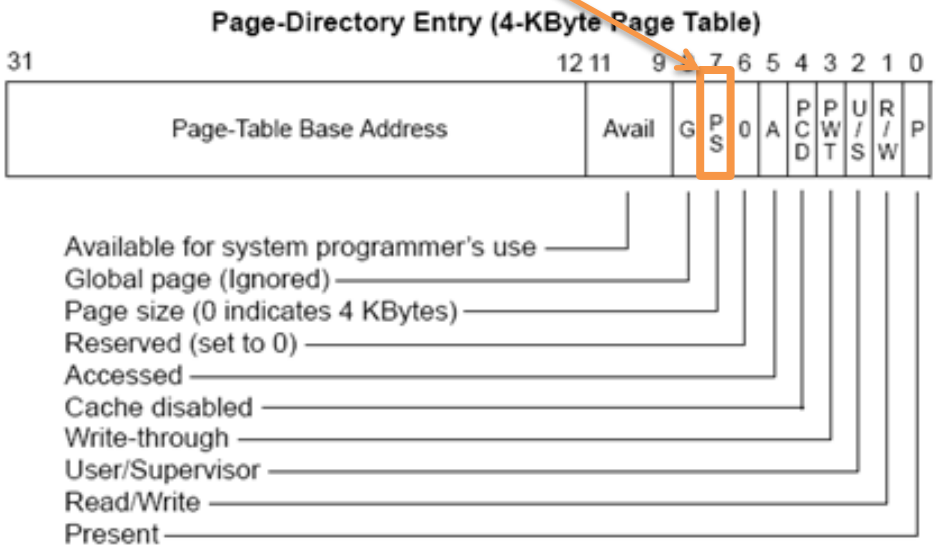
Page Directory Pointer Entry + Page Directory Entry Offset
→ (0x01003001 & 0xFFFFF000) + (0x04*8) = 0x008001e3



n0fate@live.com

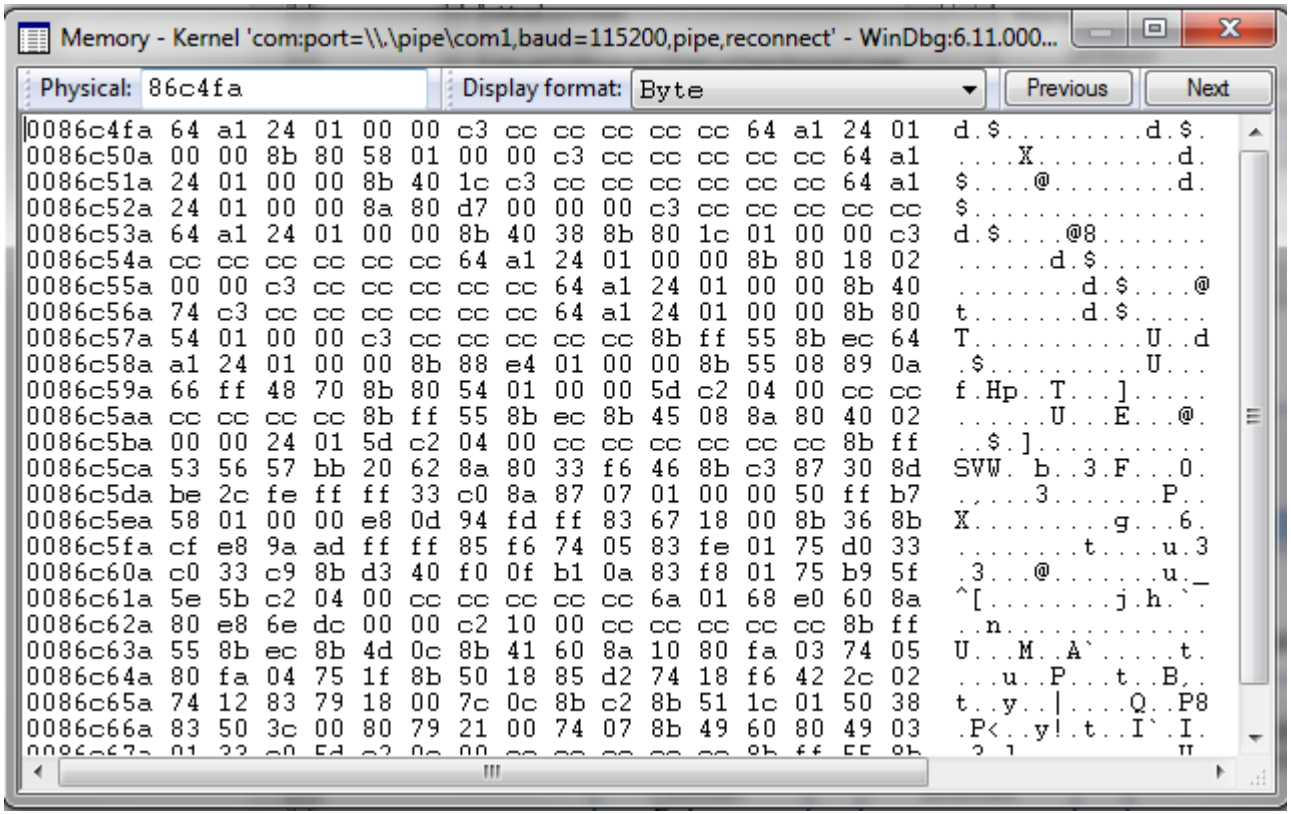
Logical to Physical Address

0x008001e3→ 10000000000000011100011



Page size 가 1이므로 Large Page Mapped
Page Directory Entry + Page Table Entry Offset
→ (0x008001e3 & 0xFFFFF000) + 0x6c4fa = 0x0086c4fa

Logical to Physical Address



n0fate@live.com

Q & A

Contact: rapfer@gmail.com